ucc

Coláiste na hOllscoile Corcaigh, Éire
University College Cork, Ireland

School of Mathematical Sciences

Intrinsic Quantification of Domain-Shift Magnitude in Reinforcement Learning

Dylan Forde

Supervisor: Prof. Gregory Provan

April 24, 2024

A Final Year Project submitted in partial fulfilment
of the requirements for the degree of
B.Sc. Data Science and Analytics

Declaration

I hereby declare that this Final Year Project is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at `https://libguides.ucc.ie/academicintegrity/plagiarism`.

I consent to the examiner retaining a copy of the thesis beyond the examining period, should they so wish (EU GDPR May 2018).

I agree that this thesis will be publicly available, but will also be available to UCC staff and students in the University's open access institutional repository on the University College Cork domain only, subject to Irish Copyright Legislation and University College Cork conditions of use and acknowledgement.

Signed: Dylan Forde Date: April 24, 2024

Abstract

In the field of reinforcement learning (RL), the ability to adapt to changes in the environment, known as domain shift, is a critical challenge. To address this challenge, we propose a novel approach that leverages deep learning neural networks not only for detecting domain shifts but also for quantifying them numerically. Our research focuses on enhancing the existing Reactive Exploration algorithm, which itself is rooted in the Intrinsic Curiosity Module framework, by subjecting it to new experimental conditions.

The core objective of our work is to explore whether Reactive Exploration exhibits predictive capabilities concerning the performance of its current policy in the presence of domain shifts. We aim to investigate the algorithm's capacity to predict the distance between its current policy and its suitability for a domain-shifted environment. This investigation is based on a set of metrics tracked by the algorithm, shedding light on its adaptability and generalization abilities.

If our research demonstrates that Reactive Exploration can effectively determine the gap between its policy and a shifted environment, it could potentially revolutionize RL. This discovery opens up promising avenues for agent-driven hyper parameter scheduling, where the RL agent autonomously adjusts its hyper-parameters based on its perceived distance from optimal as it converges. Additionally, the quantification of domain shifts may serve as a valuable input for future meta-learning approaches, further advancing the capabilities of RL agents in dynamic and evolving environments.

Our study combines cutting-edge deep learning techniques with RL to address a fundamental challenge in the field—improving an agent's adaptability in the face of domain shifts. We present experimental results that provide insights into the algorithm's predictive power, offering a glimpse into the future of RL research and its practical applications in various domains. [1] [2]

Lay Abstract

Imagine teaching a computer program to learn and make decisions like a student in a classroom. This program, known as a reinforcement learning agent, can complete abstract tasks like playing video games or controlling robots. However, just like students, these agents can struggle when they encounter new situations they haven't seen before. We want to help these agents get better at handling surprises.

To do this, we're using a type of computer brain called a neural network to teach the agent not only to recognize when things change but also to measure how big those changes are. Think of it like a student who not only notices when a new topic comes up in class but can also tell if it's a small change or a big one. Normally the teachers might say this, but we need agents decide that for themselves.

We're focusing on a specific method called Reactive Exploration, which is like a learning technique the agent uses. By testing it in different scenarios, we want to find out if it's good at guessing how well it's doing when things change. If it is, this could help the agent become even smarter. It should be able to numerically quantify how well its planned changes will adapt to the new environment.

Our goal is to make these agents more adaptable. We want them to be like students who can adjust how they study based on how well they understand a subject. This could be really useful in many areas, like making robots more helpful in our daily lives or improving how computers learn and solve problems.

In simple terms, we're trying to make computer programs smarter and better at handling surprises, just like students who become experts in their subjects.

Acknowledgements

Thanks to Prof. Gregory Provan as both my supervisor and first reader.

Thanks to Prof. Derek Bridge as my second reader.

Contents

List of Figures

List of Equations

Nomenclature

DQN	Deep Q-Network, a neural network architecture for approximating the Q-function in reinforcement learning
ϵ	Exploration rate in the ϵ-greedy strategy, determining the trade-off between exploration and exploitation
ICM	Intrinsic Curiosity Module, a mechanism to encourage exploration by rewarding the agent for exploring novel states
RL	Reinforcement Learning, a type of machine learning where an agent learns to make decisions by interacting with an environment
DS	Domain Shift, changes in the environment that can affect the performance of a learned policy
SM	Suitability Metric, a numerical representation of the adaptability of the current policy to a domain shift
Γ	Set of metrics tracked by the Reactive Exploration algorithm
CCP	Custom CartPole, a modified version of the classic control task with variable pole length
L	Pole Length, the length of the pole in the CartPole environment, which can change to simulate domain shifts
R	Reward, the feedback received from the environment as a result of the agent's actions
A	Action, a decision made by the agent at each step in the environment
S	State, the representation of the environment at a given time, observed by the agent
δ	Domain Shift Metric, a quantified measure of the change in the environment
π	Policy, a mapping from states to actions that determines the agent's behavior
γ	Discount factor, a value between 0 and 1 that determines the importance of future rewards
α	Learning Rate, a hyperparameter that controls the step size of the optimization algorithm

1 Introduction

In the evolving landscape of reinforcement learning (RL), the capacity of agents to adapt to environmental variations, known as domain shifts, stands as a paramount challenge. These shifts, which encompass any change in the environment's dynamics or rules, critically affect the performance of RL agents. Traditional RL methodologies, while effective in static or narrowly defined environments, often falter when faced with the unpredictable nature of real-world scenarios where domain shifts are the norm rather than the exception. The essence of adaptability in RL is not just about optimising policies for a given environment but about ensuring these policies remain effective or can be efficiently updated as the environment evolves.

Historically, the field has navigated this challenge through various strategies, such as domain randomisation and transfer learning, aimed at enhancing the generalisation capabilities of RL agents. However, these approaches typically do not account for real-time adaptability to new or unseen environmental conditions, leaving a gap between the agent's learned policies and the dynamic nature of practical applications. Recognising this gap, our research introduces a novel approach that leverages the intrinsic capabilities of deep learning neural networks not only to detect but also to quantify domain shifts numerically. By embedding this quantification within the RL framework, we aim to dynamically adjust the agent's exploration strategies, thereby enhancing its adaptability to changes.

Building upon the foundational principles of Reactive Exploration and the Intrinsic Curiosity Module (ICM), our approach seeks to augment the exploration mechanisms of RL agents. Reactive Exploration, rooted in the ICM framework, traditionally encourages agents to

explore novel states by rewarding them for encountering unpredictable or surprising states. Our enhancement introduces a quantitative measure of the suitability of the agent's current policy in the face of domain shifts, employing a Deep Q-Network (DQN) to learn this suitability metric. This metric, ranging between 0 and 1, serves as a dynamic guide for adjusting the agent's exploration rate (epsilon) in an epsilon-greedy strategy, making the agent more exploratory in the face of significant domain shifts.

The implications of this research extend beyond the immediate enhancements to Reactive Exploration. By intrinsically quantifying domain shifts and enabling dynamic adaptation, we contribute to the burgeoning field of meta-learning in RL. Our approach embodies the principles of "learning to learn," facilitating the development of RL agents that not only adapt to new tasks more efficiently but also continuously refine their learning process based on environmental feedback. This adaptability is crucial for deploying RL in diverse real-world applications, from robotics to autonomous systems, where agents must navigate and make decisions in ever-changing environments.

Our experimental investigation utilises a custom environment within the OpenAI Gym framework, manipulating the pole length and other factors in a cart pole scenario, or factors in the mountain car environment to simulate domain shifts. By comparing the performance of models trained with and without our predictive domain shift quantification, we aim to shed light on the potential of our approach to enhance the adaptability and generalisation of RL agents. The insights gained from this research could pave new avenues in RL, offering a glimpse into the future of adaptable, robust, and efficient learning systems capable of tackling the complexities of real-world applications.

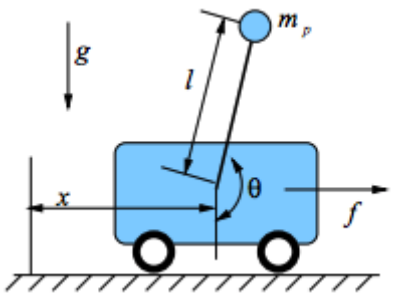

Figure 1.1: Cart-Pole Diagram. [3]

1.1 What is Reinforcement Learning

Reinforcement Learning (RL) is a sub-field of machine learning that focuses on training agents to make sequential decisions in an environment to maximise a cumulative reward signal. In RL, an agent interacts with an environment by taking actions and observing the resulting state and reward. The goal of the agent is to learn a policy, which is a mapping from states to actions, that maximises the expected cumulative reward over time.

The RL framework is typically modeled as a Markov Decision Process (MDP), which is defined by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where:

- $\mathcal{S}$ is the set of states in the environment.
- $\mathcal{A}$ is the set of actions the agent can take.
- $\mathcal{P}$ is the state transition probability function, where $\mathcal{P}(s'|s, a)$ represents the probability of transitioning to state s' when taking action a in state s.
- $\mathcal{R}$ is the reward function, where $\mathcal{R}(s, a)$ represents the immediate reward received by the agent for taking action a in state s.
- $\gamma \in [0, 1]$ is the discount factor, which determines the importance of future rewards compared to immediate rewards.

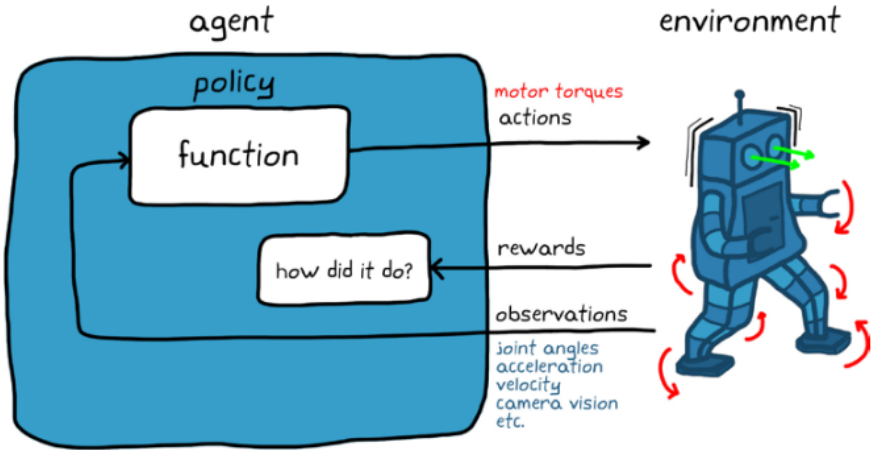

Figure 1.2: Overview of the relationship between the agent and environment. [4]

The agent's objective is to learn a policy π that maximises the expected cumulative

discounted reward:

$$J(\pi) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t, a_t)\right]$$

where s_t and a_t are the state and action at time step t, respectively.

Figure 1.3: Overview on the relationship between the agent and its rewards. [4]

RL algorithms can be broadly categorised into two main classes: value-based methods and policy-based methods.

Value-based Methods

Value-based methods learn a value function that estimates the expected cumulative reward for each state or state-action pair. The most well-known value-based method is Q-learning, which learns an action-value function $Q(s, a)$ that represents the expected cumulative reward for taking action a in state s and following the optimal policy thereafter. The Q-learning update rule is given by:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right]$$

where α is the learning rate and r_{t+1} is the reward received after taking action a_t in state s_t.

Policy-based Methods

Policy-based methods directly learn a policy $\pi(a|s)$ that maps states to a probability distribution over actions. These methods typically use gradient ascent to optimise the policy parameters θ to maximise the expected cumulative reward:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\pi_\theta)$$

where $\nabla_\theta J(\pi_\theta)$ is the gradient of the expected cumulative reward with respect to the policy parameters.

One popular policy-based method is the REINFORCE algorithm, which estimates the gradient using the following update rule:

$$\theta \leftarrow \theta + \alpha \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \left(\sum_{k=t}^{T} \gamma^{k-t} r_k \right)$$

where T is the total number of time steps in the episode and r_k is the reward received at time step k.

Actor-Critic Methods

Actor-Critic methods combine the advantages of both value-based and policy-based methods. These methods learn both a value function (critic) and a policy (actor) simultaneously. The critic is used to estimate the value function, which helps in reducing the variance of the policy gradient updates. The actor is updated using the policy gradient, which is computed using the estimated value function from the critic.

One popular actor-critic method is the Advantage Actor-Critic (A2C) algorithm, which uses the advantage function $A(s, a)$ to estimate the policy gradient:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}\pi\theta \left[\nabla_\theta \log \pi_\theta(a|s) A(s, a) \right]$$

where the advantage function is defined as:

$$A(s, a) = Q(s, a) - V(s)$$

Here, $Q(s, a)$ is the action-value function and $V(s)$ is the state-value function.

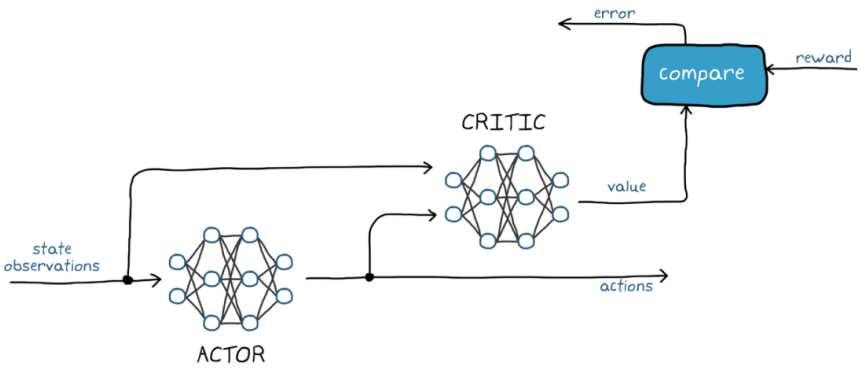

Figure 1.4: Overview of the Actor-Critic Method. [4]

Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) combines the power of deep learning with RL to tackle complex, high-dimensional problems. In DRL, deep neural networks are used to approximate the value function, policy, or both. Some popular DRL algorithms include:

- Deep Q-Networks (DQN): A value-based method that uses a deep neural network to approximate the action-value function $Q(s, a)$.
- Deep Deterministic Policy Gradient (DDPG): An actor-critic method that uses deep neural networks for both the actor and critic, and is designed for continuous action spaces.
- Proximal Policy Optimisation (PPO): A policy-based method that uses a surrogate objective function to improve the stability and efficiency of policy gradient updates.

In summary, reinforcement learning is a framework for training agents to make sequential decisions in an environment to maximise a cumulative reward signal. RL algorithms can be categorised into value-based, policy-based, and actor-critic methods, each with their own strengths and weaknesses. Deep reinforcement learning has emerged as a promising approach to tackle complex, high-dimensional problems by combining the power of deep learning with RL

1.2 Hard Exploration Problem

. One of the fundamental challenges in reinforcement learning is the exploration-exploitation dilemma, often referred to as the "hard exploration problem." This problem arises from the need to balance the exploration of the environment to gather new information and the exploitation of the current knowledge to maximise the expected reward. In many real-world scenarios, the environment may exhibit sparse rewards, making it difficult for the agent to discover optimal policies through random exploration.

Mathematically, the exploration-exploitation trade-off can be formulated as a multi-armed bandit problem. Consider a set of actions $\mathcal{A} = \{a_1, a_2, \dots, a_K\}$, where each action a_i is associated with a reward distribution R_i. The goal is to maximise the expected cumulative reward over a finite horizon T:

$$\max_{\pi} \mathbb{E}\left[\sum_{t=1}^{T} R_{\pi(t)}(t)\right] \tag{1.1}$$

where π is the policy that maps states to actions, and $R_{\pi(t)}(t)$ is the reward obtained by taking action $\pi(t)$ at time step t.

The exploration-exploitation dilemma can be characterised by the regret, which measures the difference between the expected cumulative reward of the optimal policy π^* and the expected cumulative reward of the learned policy π:

$$\text{Regret}(T) = \mathbb{E}\left[\sum_{t=1}^{T} R_{\pi^*(t)}(t) - R_{\pi(t)}(t)\right] \tag{1.2}$$

To minimise the regret, the agent must balance exploration and exploitation. Various exploration strategies have been proposed to address this challenge, such as:

- ϵ-**greedy exploration**: With probability ϵ, the agent selects a random action, and

with probability $1 - \epsilon$, it selects the action with the highest estimated value.

- **Upper Confidence Bound (UCB) exploration**: The agent selects actions based on an upper confidence bound of the estimated value, which encourages exploration of less visited states and actions.
- **Thompson Sampling**: The agent maintains a posterior distribution over the expected rewards and samples actions according to their probability of being optimal.

However, these exploration strategies may still struggle in environments with sparse rewards or large state-action spaces. In such cases, more sophisticated approaches, such as intrinsic motivation and curiosity-driven exploration, can be employed to guide the agent towards informative and promising regions of the state-action space.

Intrinsic motivation introduces an additional reward signal that encourages the agent to explore novel or uncertain states. One popular approach is the Intrinsic Curiosity Module (ICM) [2], which learns a forward dynamics model to predict the next state given the current state and action. The prediction error of the forward model serves as an intrinsic reward, encouraging the agent to explore states that are difficult to predict.

Formally, the ICM reward r_t^i at time step t is defined as:

$$r_t^i = \frac{1}{2}\|\hat{s}_{t+1} - s_{t+1}\|^2 \tag{1.3}$$

where $\hat{s}_{t+1}$ is the predicted next state, and s_{t+1} is the actual next state.

The total reward r_t is then a weighted sum of the extrinsic reward r_t^e and the intrinsic reward r_t^i:

$$r_t = r_t^e + \beta r_t^i \tag{1.4}$$

where β is a hyper-parameter that controls the balance between extrinsic and intrinsic rewards.

By incorporating intrinsic motivation, the agent is encouraged to explore regions of the state-action space that are novel or uncertain, potentially leading to the discovery of more optimal policies in sparse reward environments.

In addition to intrinsic motivation, hierarchical reinforcement learning (HRL) has emerged as a promising approach to tackle the hard exploration problem in complex environments. HRL introduces a hierarchy of abstractions, where high-level policies guide the exploration process by setting sub-goals for low-level policies. This hierarchical structure enables the agent to explore the environment more efficiently by focusing on relevant regions of the state-action space.

One influential HRL framework is the Options framework [6], which introduces temporally extended actions called options. An option $\omega \in \Omega$ is defined by a tuple $\langle I_\omega, \pi_\omega, \beta_\omega \rangle$, where:

- $I_\omega \subseteq \mathcal{S}$ is the initiation set, indicating the states in which the option can be initiated.
- $\pi_\omega : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ is the option policy, specifying the probability of taking an action in a given state when the option is active.
- $\beta_\omega : \mathcal{S} \rightarrow [0, 1]$ is the termination condition, determining the probability of terminating the option in a given state.

The high-level policy $\mu : \mathcal{S} \rightarrow \Omega$ selects options based on the current state, while the low-level policies π_ω execute the selected option until termination. The value function for an option ω in state s can be defined as:

$$Q_\mu(s, \omega) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} | s_0 = s, \omega_0 = \omega, \mu\right] \tag{1.5}$$

where γ is the discount factor, and r_{t+1} is the reward obtained at time step $t + 1$.

The option-value function can be learned using temporal difference methods, such as Q-learning or SARSA, by treating options as temporally extended actions. The high-level policy μ can be learned using policy gradient methods, such as Actor-Critic or Proximal Policy Optimisation (PPO), by considering the option-value function as the critic.

By learning a hierarchy of options, the agent can explore the environment more efficiently by focusing on relevant sub-goals and abstracting away the low-level details. This hierarchical structure allows for more directed exploration and can significantly reduce the sample complexity in sparse reward environments.

Recent advancements in deep reinforcement learning have further extended the capabilities of HRL by combining it with deep neural networks. The Option-Critic architecture [7] learns options and their termination conditions end-to-end using a single neural network, enabling the discovery of options without prior domain knowledge. The Feudal Networks [8] approach introduces a hierarchical structure where a manager network sets goals for worker networks, which learn to achieve these goals through intrinsic motivation.

In conclusion, the hard exploration problem remains a significant challenge in reinforcement learning, particularly in environments with sparse rewards and large state-action spaces. Intrinsic motivation and hierarchical reinforcement learning have emerged as promising approaches to address this challenge by guiding the exploration process and abstracting away the low-level details. By combining these techniques with deep neural networks, agents can learn to explore complex environments more efficiently and discover optimal policies in a sample-efficient manner. As research in this area continues to advance, we can expect to see more sophisticated and effective exploration strategies that enable reinforcement learning agents to tackle increasingly challenging real-world problems.

1.3 Generalisation Problems in Research

Generalisation is a critical challenge in reinforcement learning (RL) research, as it directly impacts the ability of RL agents to perform well in novel environments or tasks. The goal of generalisation is to ensure that the learned policies or value functions can effectively handle

situations that were not encountered during training. However, achieving robust generalisation in RL is not trivial due to several factors, such as the high-dimensional state and action spaces, the complexity of the environments, and the inherent uncertainty in the decision-making process.

One of the primary issues in RL generalisation is the distributional shift between the training and testing environments. In many real-world applications, the agent may encounter situations that differ significantly from those seen during training. This shift can arise from various sources, such as changes in the environment dynamics, variations in the reward function, or the introduction of new obstacles or constraints. When the distribution of the test environment differs from the training environment, the learned policy may fail to perform optimally, leading to subpar generalisation.

Mathematically, let $\mathcal{M}_{\text{train}}$ and $\mathcal{M}_{\text{test}}$ denote the Markov Decision Processes (MDPs) representing the training and testing environments, respectively. The goal of generalisation is to learn a policy π in $\mathcal{M}_{\text{train}}$ that maximises the expected cumulative reward in $\mathcal{M}_{\text{test}}$:

$$\max_{\pi} \mathbb{E}_{\mathcal{M}_{\text{test}}} \left[\sum_{t=0}^{\infty} \gamma^t r_t | \pi \right] \tag{1.6}$$

where $\gamma \in [0, 1]$ is the discount factor, and r_t is the reward obtained at time step t.

However, when the distribution of $\mathcal{M}_{\text{test}}$ differs significantly from $\mathcal{M}_{\text{train}}$, the learned policy π may not generalise well, resulting in sub-optimal performance.

Another challenge in RL generalisation is the problem of over-fitting. Over-fitting occurs when the agent learns a policy that performs well in the training environment but fails to generalise to new situations. This issue arises when the agent memorises specific patterns or trajectories from the training data instead of learning a general strategy that can adapt to novel scenarios. Over-fitting can be particularly problematic in RL due to the high-dimensional state and action spaces and the complex interactions between the agent and the environment.

To mitigate over-fitting, various regularisation techniques have been proposed in the RL literature. One popular approach is to introduce stochasticity in the agent's policy, which encourages exploration and prevents the agent from getting stuck in sub-optimal solutions. This can be achieved through techniques such as entropy regularisation, where an additional term is added to the objective function to promote diversity in the agent's actions:

$$J(\pi) = \mathbb{E}_{\mathcal{M}_{\text{train}}} \left[\sum_{t=0}^{\infty} \gamma^t (r_t + \alpha H(\pi(\cdot|s_t))) \right] \tag{1.7}$$

Here, $H(\pi(\cdot|s_t))$ denotes the entropy of the policy distribution at state s_t, and α is a hyper-parameter controlling the strength of the entropy regularisation.

Another approach to improving generalisation is to leverage transfer learning techniques, where knowledge learned from one task or environment is transferred to another related task or environment. Transfer learning can help the agent adapt quickly to new situations by reusing the learned features or policies from previous experiences. This can be particularly effective when the source and target tasks share similar structure or dynamics.

In the context of transfer learning, the objective is to learn a policy π in the source MDP $\mathcal{M}_{\text{source}}$ that maximises the expected cumulative reward in the target MDP $\mathcal{M}_{\text{target}}$:

$$\max_{\pi} \mathbb{E}_{\mathcal{M}_{\text{target}}} \left[\sum_{t=0}^{\infty} \gamma^t r_t | \pi \right] \tag{1.8}$$

To facilitate knowledge transfer, various techniques have been proposed, such as learning a shared representation space between the source and target tasks, or using meta-learning algorithms to learn a learning algorithm that can quickly adapt to new tasks.

Another promising approach to improving generalisation in RL is to incorporate domain randomisation techniques. Domain randomisation involves training the agent on a diverse set of environments with randomised properties, such as variations in the physical

parameters, visual appearances, or initial conditions. By exposing the agent to a wide range of variations during training, it can learn to generalise better to unseen environments.

Mathematically, domain randomisation can be formulated as learning a policy π that maximises the expected cumulative reward over a distribution of MDPs $p(\mathcal{M})$:

$$\max_{\pi} \mathbb{E}\mathcal{M} \sim p(\mathcal{M}) \left[\mathbb{E}\mathcal{M} \left[\sum_{t=0}^{\infty} \gamma^t r_t | \pi \right] \right] \tag{1.9}$$

By optimising the policy over a diverse set of environments, the agent can learn to be robust to variations and generalise better to novel situations.

Recent advances in deep learning have also opened up new possibilities for improving generalisation in RL. Deep neural networks have the capacity to learn rich, hierarchical representations of the state space, which can capture the underlying structure and dynamics of the environment. By leveraging the representational power of deep learning, RL agents can learn more general and transferable features that can aid in generalisation.

However, the use of deep neural networks in RL also introduces new challenges, such as the potential for over-fitting and the difficulty in interpreting the learned representations. To address these issues, various techniques have been proposed, such as regularisation methods specifically tailored for deep RL, and the development of interpretable and explainable RL models.

Another important consideration in RL generalisation is the choice of the state and action representations. The way in which the state and action spaces are represented can have a significant impact on the agent's ability to generalise. For example, using a high-dimensional, pixel-based representation of the state space may lead to over-fitting and poor generalisation, as the agent may focus on irrelevant details in the input. On the other hand, using a compact, task-relevant representation can facilitate generalisation by capturing the essential features of the environment.

In addition to the state and action representations, the choice of the reward function also

plays a crucial role in generalisation. A well-designed reward function should provide a clear and informative signal that guides the agent towards the desired behavior, while being robust to variations in the environment. Poorly designed reward functions can lead to unintended consequences and hinder generalisation.

To address the challenges of generalisation in RL, researchers have also explored the use of ensemble methods and multi-task learning. Ensemble methods involve training multiple RL agents with different architectures, hyper parameters, or initialisation, and combining their predictions to make more robust and generalisable decisions. Multi-task learning, on the other hand, involves training a single agent to solve multiple related tasks simultaneously, which can help the agent learn more general and transferable skills.

In conclusion, generalisation is a critical challenge in RL research, and addressing it is crucial for the development of RL agents that can perform well in real-world applications. Various techniques, such as regularisation, transfer learning, domain randomisation, and deep learning, have been proposed to improve generalisation in RL. However, there is still much work to be done in developing more robust and generalisable RL algorithms that can handle the complexities and uncertainties of real-world environments. As the field of RL continues to advance, it is important to prioritise research on generalisation and to develop new techniques and frameworks that can enable RL agents to learn more general and adaptable skills.

1.4 Background

Reinforcement learning (RL) is predicated on the principle of agents learning to navigate an environment to maximise some notion of cumulative reward. Seminal algorithms within this domain, such as Q-learning, Deep Q-Networks (DQN), and various policy gradient methods, have propelled the field forward, enabling sophisticated decision-making capabilities in both discrete and continuous action spaces [9]. These foundational techniques lay the groundwork for understanding agent-environment interactions and the process of learning optimal policies through trial and error.

A significant hurdle in the application of RL is the phenomenon of domain shifts—variations in the environment that can substantially degrade the performance of a previously learned policy. These shifts can arise from changes in the environment's dynamics, the rules governing the environment, or the objectives the agent is tasked with achieving. The inherent assumption in traditional RL—that the training and operational environments are static or sufficiently similar—is often violated in real-world applications, necessitating adaptive strategies that can cope with such changes.

Several approaches have been proposed to address the challenge of domain shifts in Reinforcement Learning. Transfer learning methods, such as progressive neural networks [10] and policy distillation [11], aim to transfer knowledge from a source domain to a target domain, enabling faster adaption to new environments. Meta-learning techniques, like MAML [12] and RL^2 [13], focus on learning a learning algorithm that can quickly adapt to new tasks or domains. Domain randomisation [14] involves training agents on a wide range of randomised environments to improve generalisation. While these approaches have shown promise, they often require extensive pre-training or rely on fixed adaption strategies. In contrast, our proposed framework introduces a novel domain shift predictor that dynamically adjusts the exploration rate based on the detected domain shifts, enabling more efficient and adaptive exploration in non-stationary environments.

Several approaches have been proposed to tackle the challenge of domain shifts in reinforcement learning. Domain randomisation [14] is a popular technique that involves training agents on a diverse set of simulated environments with randomised properties to improve generalisation. However, this approach relies on the availability of a large number of training environments and may not always capture the full range of possible domain variations. Transfer learning methods [15] aim to leverage knowledge learned in one domain to accelerate learning in another, but they often require a significant amount of data from the target domain. Meta-learning approaches [12] seek to learn a learning algorithm that can quickly adapt to new tasks, but they may struggle with large domain shifts. In contrast to these methods, our proposed framework focuses on quantifying domain shifts and dynamically adjusting the exploration strategy to enable efficient adaptation without

requiring extensive training data or prior knowledge of the target domain.

The fusion of deep learning with RL, especially through DQNs and deep policy networks, has marked a paradigm shift, enabling the approximation of complex decision-making functions. These advancements have not only expanded the applicability of RL to high-dimensional spaces but have also spurred innovations aimed at improving the stability, efficiency, and generalisation capabilities of RL algorithms [16].

Despite the strides made, the dynamic adaptability of RL agents to evolving environments remains a pertinent challenge. Traditional strategies often rely on extensive pre-training or fixed heuristics, which may lack the flexibility needed for real-time adaptation. Emerging research endeavors, such as the approach proposed in this study, aim to surmount this challenge by introducing mechanisms for intrinsic adaptability. By quantifying domain shifts and adjusting exploration strategies based on a learned suitability metric, this novel methodology aspires to endow RL agents with the capacity to dynamically respond to environmental changes, thereby addressing a critical gap in the existing literature.

2 Figures, Tables, Referencing

2.1 Figures

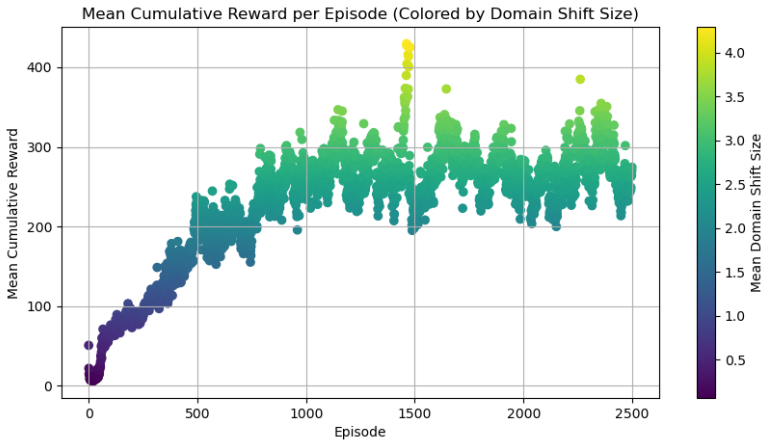

Figure 2.1: Training data for linear changes in pole length with no domain shift predictor.

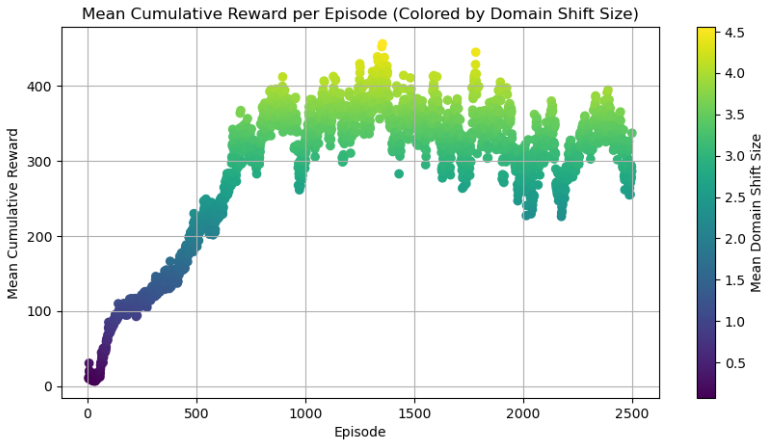

Figure 2.2: Training data for linear changes in pole length with domain shift predictor.

Between the two images above, figure 2.1 has a mean of converged rewards mostly between 200 and 300, with a fewer number of values between 300 and 400, and a handful of points above 400 cumulative reward. While figure 2.2 has a converged mean between 300 and 400, with a fewer number of values between 200 and 300, and a decent amount of points above 400 cumulative reward. While the values in figure 2.1 has values above 400, they seem to be outliers from a certain episode and not an overall trend, unlike figure 2.2. The DSP clearly has much less variance while reaching the converged mean, while figure 2.1 is much less fluid. These two images arise from the CCP where the increase in pole length is constant across each step, where you would assume that the impact of having a predictor would be minimised, which is not actually the case.

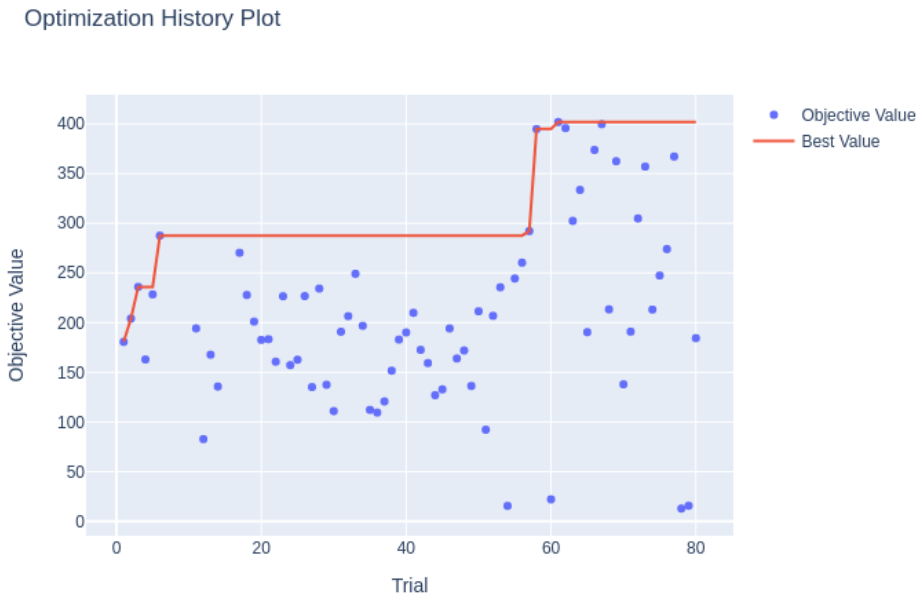

Figure 2.3: Optuna best trial optimisation history for non-linear changes in pole length and no domain shift predictor.

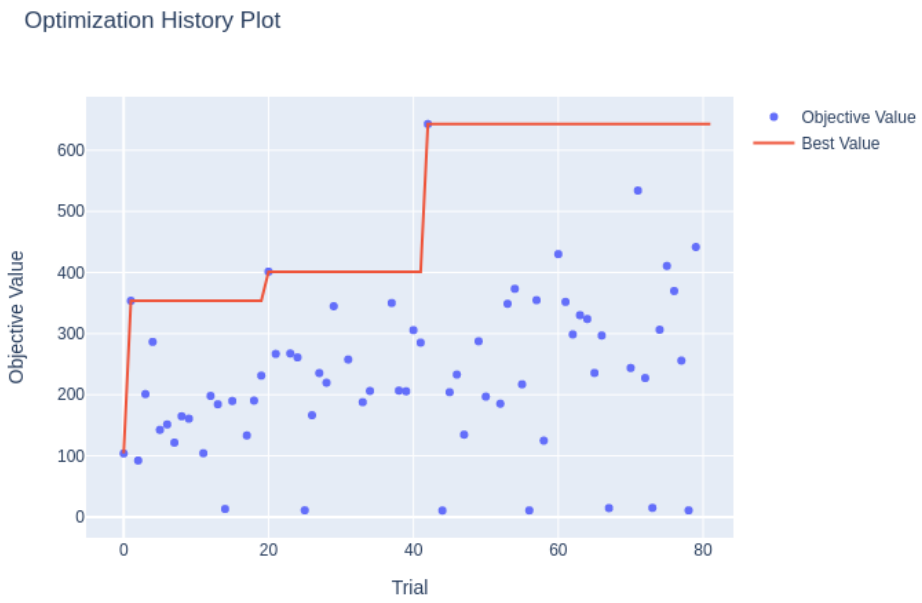

Figure 2.4: Optuna best trial optimisation history for non-linear changes in pole length and with domain shift predictor.

For volatile domain shifts, we can see the Optuna history for both models with and without the DSP. The max value with DSP is 600, while the max value without DSP is 400. There are 5 trials in figure 2.4 which beat the highest value in figure 2.3, showing a stark over performance when using the DSP. The points with extremely low objective values arise from pruned trials.

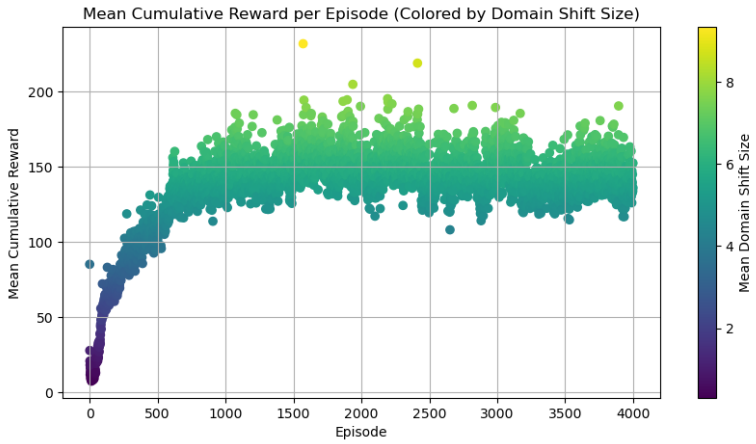

Figure 2.5: Training data for non-linear changes in pole length with no domain shift predictor.

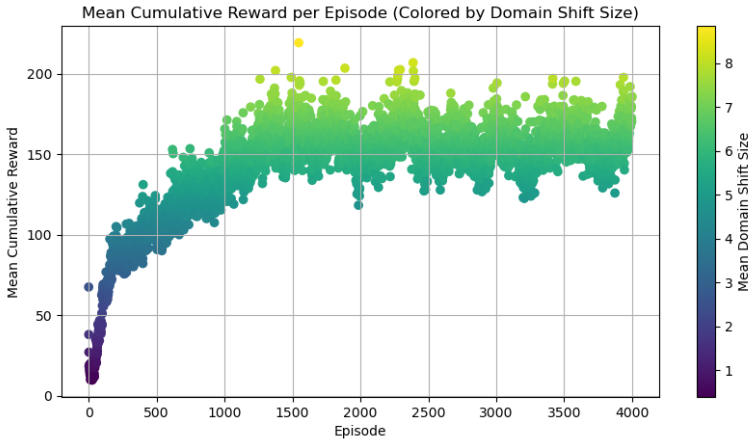

Figure 2.6: Training data for non-linear changes in pole length with domain shift predictor.

The figure 2.6, representing the model with the Domain Shift Predictor, shows a higher concentration of episodes stabilising around a mean cumulative reward of 200 or above. In contrast, the figure 2.5, without the Domain Shift Predictor, has a majority of episodes converging to a mean reward of approximately 150. For the model with the Domain Shift Predictor, the lower end of the stabilised trials is around 150, indicating that even the worst-performing trials with the predictor tend to achieve a higher mean reward compared to the majority of trials without the predictor. The model with the Domain Shift Predictor exhibits a greater number of trials reaching higher mean cumulative rewards, with several trials consistently achieving rewards above 200. On the other hand, the model without the predictor has much fewer trials exceeding the 200 reward threshold, and those that do are less consistent. The overall distribution of mean cumulative rewards for the model with the Domain Shift Predictor is shifted towards higher values, suggesting that the predictor contributes to improved performance and stability across different trials with only one domain shift being present. Both models show a similar pattern of increasing mean cumulative rewards as the number of episodes increases, indicating that learning progress is being made over time. However, the model with the Domain Shift Predictor demonstrates faster convergence and higher achieved rewards.

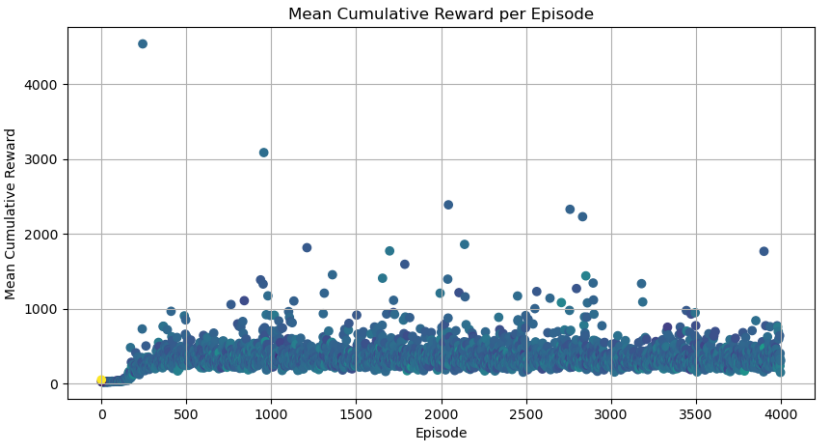

Figure 2.7: Friction and Mass changes with Domain Shift Predictor.

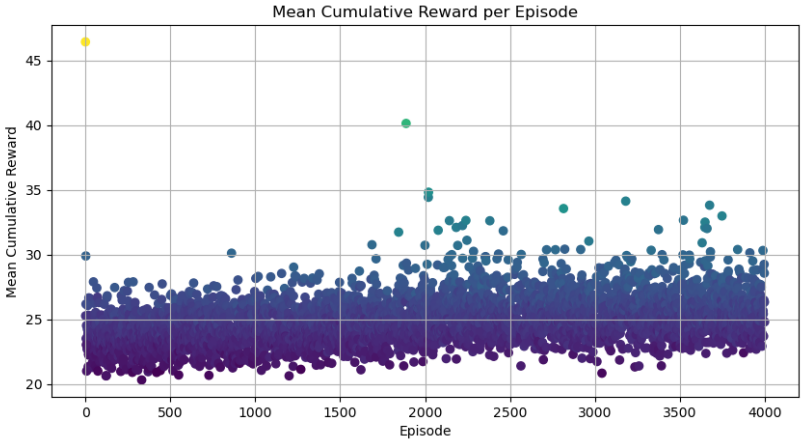

Figure 2.8: Friction and Mass changes without Domain Shift Predictor.

The two figures, 2.7 and 2.8, present the mean cumulative reward per episode for our models, with figure 2.7 representing the model with our Domain Shift Predictor and figure 2.8 representing the model without the Domain Shift Predictor. Upon comparing the two figures, it is evident that the model with the domain shift predictor (2.7) achieves significantly higher mean cumulative rewards per episode compared to the model without the predictor (2.8). Figure 2.7 shows that the mean cumulative reward consistently reaches values above 3000, with some episodes even approaching or exceeding 4000. In contrast, Figure 2.8 depicts the mean cumulative reward mostly fluctuating between 20 and 30, with only a small number of episodes reaching slightly higher values. The stark difference in the magnitude of the mean cumulative rewards highlights the superior performance of the model with the domain shift predictor. The predictor enables the model to adapt more effectively to changes in the environment, resulting in higher rewards. The model without the predictor, on the other hand, struggles to achieve comparable reward levels, indicating its limited ability to handle domain shifts. Another notable observation is the stability of the mean cumulative reward in the left image. The rewards remain consistently high throughout the episodes, with only minor fluctuations. This suggests that the model with the domain shift predictor is able to maintain its performance even in the presence of environmental changes. In contrast, the right image shows more pronounced fluctuations in the mean cumulative reward, indicating that the model without the predictor is more sensitive to domain shifts and may struggle to maintain stable performance.

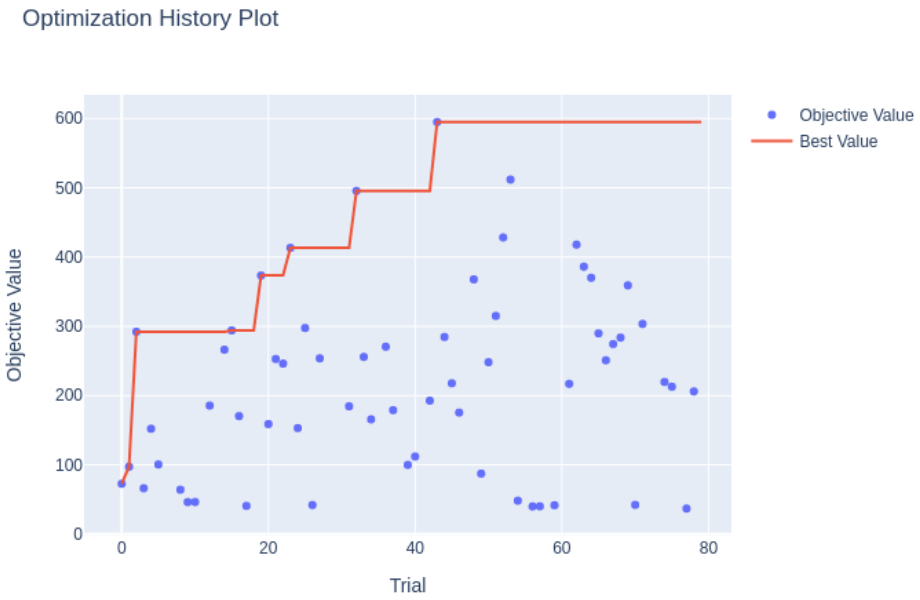

Figure 2.9: Optimisation History for Friction and Mass changes with Domain Shift Predictor.

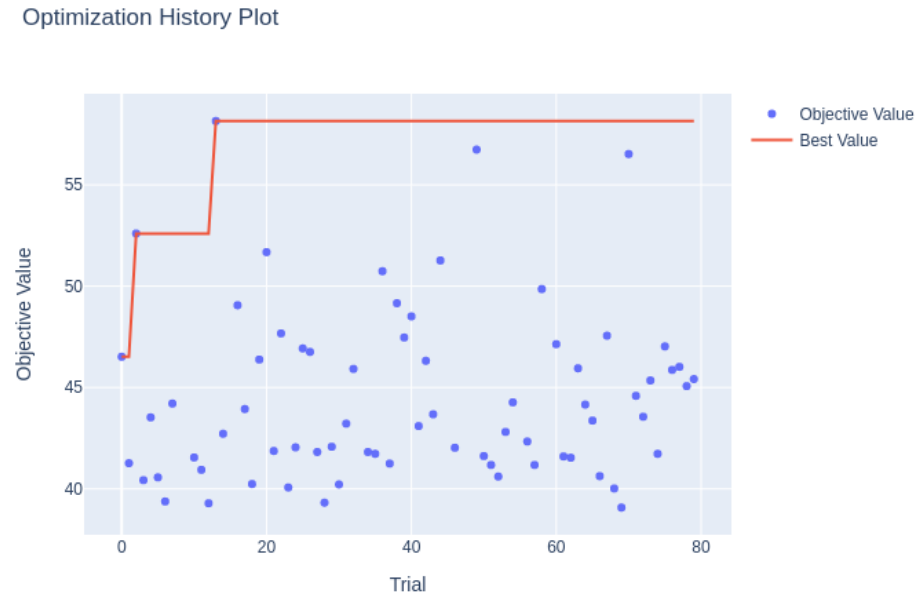

Figure 2.10: Optimisation History for Friction and Mass changes without Domain Shift Predictor

The two optimisation history plots provided show the objective values achieved over a series of trials for the model with the Domain Shift Predictor, and the model without the Domain Shift Predictor. The figure 2.9 corresponds to the model with a Domain Shift Predictor, while the figure 2.10 represents a model without such Domain Shift Predictor. The model with the domain shift predictor (2.9) achieves a significantly higher maximum objective value of approximately 600, compared to the model without the predictor (2.10), which reaches a maximum value of only around 60. This suggests that the incorporation of the Domain Shift Predictor enables the model to attain substantially better performance in terms of the optimisation objective. The figure with the Domain Shift Predictor exhibits a wider spread of objective values, mostly ranging from around 100 to 600, with a small portion of trials achieving values above 400. In contrast, the plot without a Domain Shift Predictor shows a more concentrated distribution of objective values, mostly falling between 40 and 50, with only a small portion of trials reaching higher values. This indicates that the model with the domain shift predictor is capable of exploring a broader range of solutions and consistently achieving higher objective values across multiple trials. The model with a Domain Shift Predictor demonstrates a clearer convergence pattern, with the objective values generally increasing and stabilising at higher levels as the number of trials progresses. The model lacking a Domain Shift Predictor, on the other hand, shows a more scattered and inconsistent pattern, with objective values fluctuating without a clear trend towards improvement. This suggests that the model with the domain shift predictor is more stable and effective in converging towards optimal solutions.

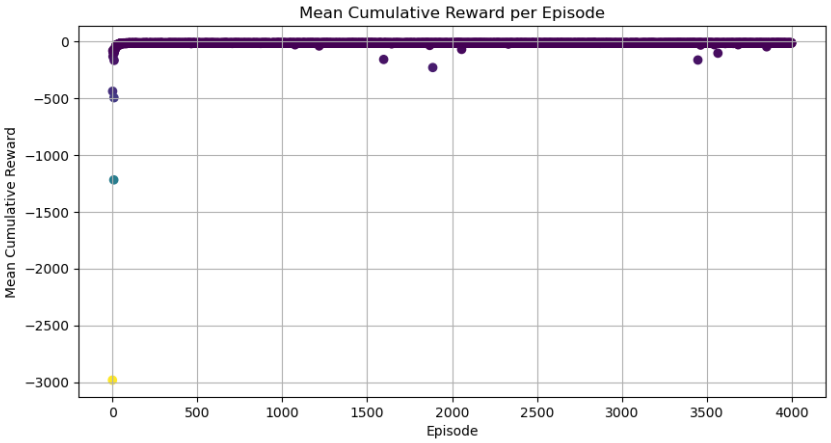

Figure 2.11: Training data for the Mountain car Environment with force domain shifts without Domain Shift Predictor

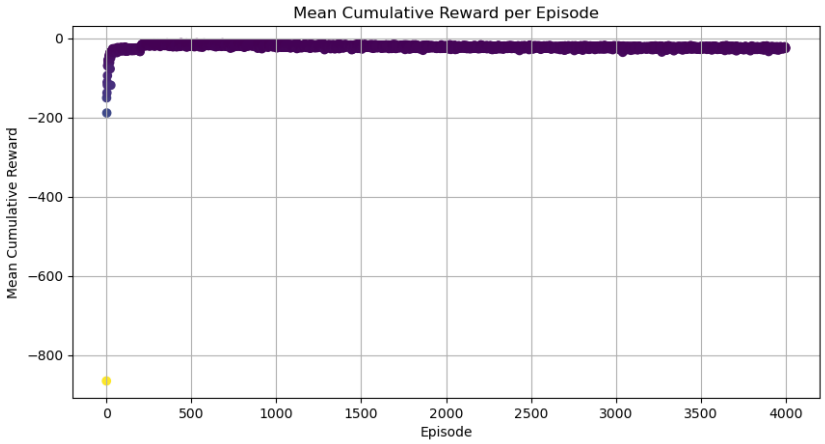

Figure 2.12: Training data for the Mountain car Environment with force domain shifts with Domain Shift Predictor

The two figures, 2.12 and 2.11, present the mean cumulative reward per episode for the mountain car environment models, with figure 2.12 representing the model with our Domain Shift Predictor and figure 2.11 representing the model without the Domain Shift Predictor. Upon comparing the two figures, it is evident that the model with the domain shift predictor (2.12) achieves the same mean cumulative rewards per episode compared to the model without the predictor (2.11). Figure 2.12 shows that the mean cumulative reward consistently reaches values close to the minimum, with the worst value around -850. In contrast, Figure 2.11 depicts the mean cumulative reward also reaching close to the minimum, with the lowest value reaching approximately -3000. Another important observation is the consistency of the mean cumulative reward in figure 2.12. The rewards remain relatively stable throughout the episodes, with only minor fluctuations. This suggests that the model with the domain shift predictor is able to maintain its performance even in the presence of environmental changes. In contrast, figure 2.11 shows more significant fluctuations in the mean cumulative reward, indicating that the model without the predictor is more sensitive to domain shifts and may struggle to maintain stable performance. The minimum values are also very notable, where figure 2.12 has much better performance at the initial episodes, when comparing the worst values of -3000 and -850.

2.2 Equations

$$\epsilon_t = \epsilon_{\text{end}} + (\epsilon_{\text{start}} - \epsilon_{\text{end}}) \exp\left(-\frac{t}{\epsilon_{\text{decay}}}\right) \tag{2.1}$$

Equation 2.1 describes the decay of the exploration rate (ϵ) over time (t), where ϵ_{start} is the initial exploration rate, ϵ_{end} is the minimum exploration rate, and ϵ_{decay} is the rate at which ϵ decays.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right] \tag{2.2}$$

Equation 2.2 represents the update rule for the Q-values in the DQN, where $Q(s_t, a_t)$ is the Q-value for state s_t and action a_t, α is the learning rate, r_{t+1} is the reward received after taking action a_t in state s_t, γ is the discount factor, and $\max_a Q(s_{t+1}, a)$ is the maximum Q-value for the next state s_{t+1} over all possible actions.

$$\text{Loss} = \frac{1}{N}\sum(Q(s_t, a_t) - (r_t + \gamma \max_a Q(s_{t+1}, a)))^2 \tag{2.3}$$

Equation 2.3 represents the DQN loss function, crucial for learning by minimising the difference between predicted and target Q-values.

$$a_t = \begin{cases} \max_a Q(s_t, a; \theta) & \text{if random} > \epsilon \\ \text{random action} & \text{otherwise} \end{cases} \tag{2.4}$$

Action selection, shown in Equation 2.4, dictates whether the agent explores or exploits based on ϵ.

$$\epsilon_{t+1} = \max(\epsilon_{\text{min}}, \epsilon_t \cdot \text{Adjustment Factor if DomainShiftMetric exceeds threshold}) \tag{2.5}$$

Equation 2.5 updates ϵ in response to domain shifts, promoting adaptability.

$$\text{DomainShiftMetric} = |\text{Original Value} - \text{Current Value}| \tag{2.6}$$

Equation 2.6 quantifies the domain shift as the absolute difference between the original length of the pole and its current length in the modified CartPole environment.

$$\text{Suitability Score} = \sigma(w_1 \cdot s + w_2 \cdot \text{TotalDomainShiftMetric} + b) \tag{2.7}$$

Equation 2.7 represents the calculation of the suitability score by the domain shift predictor. The suitability score is computed using a sigmoid function σ, where s is the state vector, TotalDomainShiftMetric is the total domain shift metric, and w_1, w_2, and b are learnable parameters of the predictor network.

$$\mathcal{L}(\theta) = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\text{Suitability Score}_i) + (1 - y_i)\log(1 - \text{Suitability Score}_i)] \tag{2.8}$$

Equation 2.8 defines the loss function $\mathcal{L}(\theta)$ for training the domain shift predictor. It is a binary cross-entropy loss, where N is the number of training samples, y_i is the true suitability label (0 or 1) for the i-th sample, and Suitability Score$_i$ is the predicted suitability score for the i-th sample. The predictor's parameters θ are optimised to minimise this loss.

2.3 Tables

Table 2.1: Range of Values Suggested by Optuna

Parameter	**Suggested Range**
Learning Rate (lr)	1×10^{-5} to 1×10^{-2}
Epsilon Decay (eps_decay)	100 to 1000
Batch Size (batch_size)	{32, 64, 128, 256}
Gamma (γ)	0.8 to 0.9999

3 Methods

3.1 System Configuration

The experiments were conducted on a system equipped with a 12th Gen Intel® Core™ i7-12700KF CPU, 32.0 GiB of RAM, a NVIDIA GeForce RTX 3090 GPU and 1.0 Tb of storage. The operating system Ubuntu was used and all software was run on this machine.

3.2 Software

The software stack for this project was built upon Python, a versatile programming language favored for its extensive library ecosystem and community support in the fields of data science and machine learning. Key Python libraries utilised include:

- PyTorch: A leading deep learning library that provides a flexible and intuitive interface for building and training neural network models. PyTorch's dynamic computation graph paradigm allows for sophisticated model architectures and seamless GPU acceleration.
- Gym: An open-source library developed by OpenAI, providing a rich collection of environments for developing and comparing reinforcement learning algorithms. The Gym library's interface standardises the way agents interact with environments, facilitating the implementation of generic algorithms applicable across various tasks.
- Matplotlib: A comprehensive library for creating static, interactive, and animated

visualisations in Python. Matplotlib was used extensively in this project to plot training progress, performance metrics, and other key data insights to analyse the agent's learning behavior and the effectiveness of different strategies.

- Optuna: An optimisation framework designed to automate the hyperparameter tuning process, enabling efficient and robust discovery of optimal configurations for machine learning models. Optuna's flexible architecture supports various optimisation strategies, including grid search, random search, and Bayesian optimisation.

3.3 Environments

The Gym Cart-pole and Mountain-Car environments are two popular benchmark environments in the Gym library, a Python library for reinforcement learning research. The Cart-pole environment is a classic control problem that involves balancing a pole attached to a cart that can move left or right on a track. The goal is to balance the pole upright by applying forces to the cart. The environment is considered a simple yet challenging problem, requiring the agent to learn to balance the pole while avoiding the cart falling off the track.

In the Cart-pole environment, the state of the system can be described as either stable or unstable. A stable state occurs when the pole is balanced upright and the cart is moving steadily along the track. In this state, the agent receives a reward for maintaining the balance and can continue to learn and improve its policy. On the other hand, an unstable state occurs when the pole falls or the cart falls off the track, resulting in a penalty or a reset of the episode. The agent must learn to recognise and respond to these unstable states to avoid penalties and achieve the goal of balancing the pole.

The system's state in the Cart-pole environment is characterised by four variables: the cart's position and velocity, and the pole's angle and angular velocity from the vertical. These variables collectively inform the agent about the current dynamics of the pole and cart. Decisions in this environment are typically binary: applying a force either to the left or to the right of the cart. The simplicity of this control scheme belies the complexity of the

underlying physical dynamics, including aspects of physics like gravity and friction, which the agent must learn to account for.

Success in the Cart-pole environment is quantified by the duration the pole remains upright. Each time step the pole remains balanced, the agent receives a positive reward. Failure occurs when the pole tilts over a certain angle from vertical or the cart moves too far from the center of the track, at which point the environment resets.

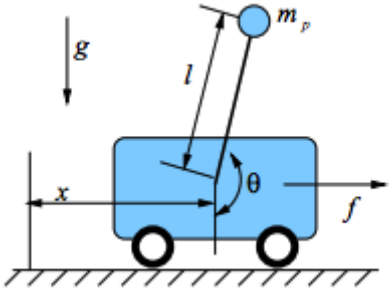

Figure 3.1: Cart-Pole Diagram. [3]

The Mountain-Car environment is another classic control problem that involves driving a car up a steep mountain. The car starts at the bottom of the mountain and must learn to apply the correct amount of force to the accelerator and brakes to reach the top of the mountain. The environment is considered challenging because the car must learn to balance the trade-off between speed and control to avoid falling off the road or running out of fuel.

In the Mountain-Car environment, the state of the system can also be described as either stable or unstable. A stable state occurs when the car is moving steadily up the mountain and the agent is making progress towards the goal. In this state, the agent receives a reward for making progress and can continue to learn and improve its policy. On the other hand, an unstable state occurs when the car loses control or runs out of fuel, resulting in a penalty or a reset of the episode. The agent must learn to recognise and respond to these unstable states to avoid penalties and achieve the goal of reaching the top of the mountain.

In the Mountain-Car, the state is defined by the position and velocity of the car. The actions available to the agent are threefold: accelerating to the left, coasting, and accelerating to the right. Unlike the immediate feedback of balancing a pole in the Cart-pole

environment, the challenge in Mountain-Car is strategic: the agent must discover and exploit the environmental dynamics, specifically the gravity and the geometry of the valley, to achieve its goal.

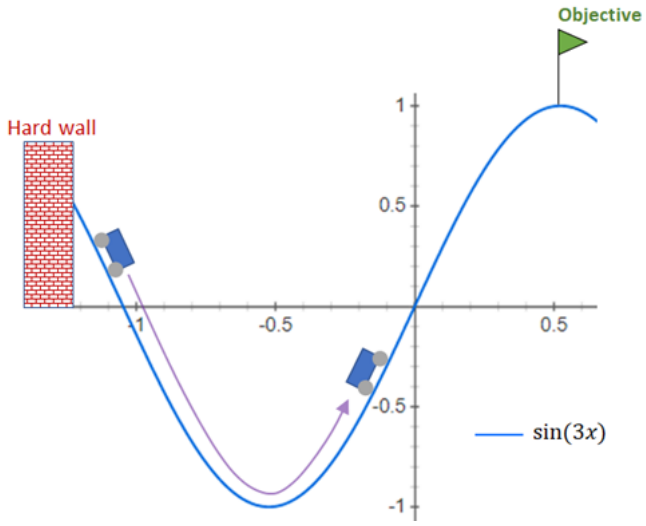

Figure 3.2: Mountain Car Environment [5]

Both environments are commonly used in reinforcement learning research to test and evaluate the performance of various algorithms, such as Q-learning, SARSA, and policy gradient methods. They are also used to compare the performance of different algorithms and to identify the strengths and weaknesses of each.

3.4 Generalised Implementation

In this research, our training methodology emphasises the incorporation of domain shifts to simulate non-stationary environments, thereby assessing the adaptability of our reinforcement learning agents. We implement these domain shifts by altering specific parameters or dynamics within the environment at each time step. Such modifications are intended to represent changes to which the agent must adjust. For instance, in the Cart-Pole environment, variations might include altering the pole's length or the cart's mass, whereas, in the Mountain-Car environment, adjustments could involve modifying the hill's height or the gravity's strength. The frequency of the domain shifts is at each time step in the model training, intending to provide a more robust and generalisable policy after training.

Our reinforcement learning models are structured to function in an oracle-like manner, providing continuous access to the values of these domain shifts. This feature enables the model to integrate information about the shifts into its decision-making process, facilitating appropriate behavioural adaptations in real-time. This capability is crucial for evaluating the effectiveness of the domain shift predictor when both Deep Q-Networks (DQNs) are informed of these shifts.

Furthermore, we introduce a domain shift predictor that assesses the appropriateness of the current policy in light of the observed domain shifts. The exploration rate (epsilon) of the agent is adjusted based on this assessment. In one variant of our model, the epsilon value is dynamic, and directly influenced by the predictor's suitability output. In contrast, a control model variant maintains a consistent epsilon value that decays normally throughout the training process, serving as a benchmark for comparing the adaptive model's performance. The model without the domain shift predictor also has access to the real-time domain shift quantification metric but does not have a suitability score or the ability to change its exploration-exploitation trade-off.

To quantitatively measure the magnitude of these domain shifts, we employ a distance metric calculated as the absolute difference between the original and modified parameters of the environment. In the case of multiple domain shifts, it is simply the sum of these differences. While this method provides a straightforward measure, alternative approaches like the Euclidean distance between the original and adjusted parameters might offer enhanced accuracy. This quantitative assessment of domain shifts provides numerical input for the domain shift predictor and control with no predictor, facilitating more informed adaptations by the learning models.

Critically, for some instances the suitability score will become fixed as the domain shift predictor does little work at trying to assess the suitability. In order to combat this, an approach of randomising the suitability score for the first 200 episodes was undertaken. This allows the model to get a better sense of how the suitability score can change the underlying dynamics of the model, and how it can best improve the score.

For specific numbers, we train for 4000 episodes and aggregate this over 80 trials. Where each trial uses Optuna for hyper-parameter optimisation.

3.5 Implementation

To explore the impact of non-stationary environments on reinforcement learning agents, a custom version of the classic Cart-Pole environment was developed. This environment extends the traditional Cart-Pole task by introducing variable pole length, force magnitude, and mass, which change at each time step within predefined ranges, simulating dynamic domain shifts. This modification challenges the agent to continuously adapt its strategy to maintain balance, providing a more complex and realistic scenario for evaluating reinforcement learning algorithms under domain shifts. The custom environment introduces domain shifts by randomly changing the pole length, cart mass, and cart friction at each step. It provides methods to quantify the domain shift and reset the environment to its original state.

```
class CustomCartPoleEnv(CartPoleEnv):
    def __init__(self):
        super().__init__()
        # Domain shifts
        self.original_length = self.length  # Save the original length for resetting
        self.length_change_rate = 0.01  # Define how quickly the pole length should change
        self.min_length_change = -0.01
        self.max_length_change = 0.09
        self.original_masscart = self.masscart # Save the original mass
        self.min_masscart_change = 0.1
        self.max_masscart_change = 1.0
        self.original_force_mag = self.force_mag # Save the original force
        self.min_force_change = -2.0
        self.max_force_change = 8.0

```

```
16
17      def change_pole_length(self):
18          length_change = random.uniform(self.min_length_change, self.
    max_length_change)
19          self.length += length_change
20          self.polemass_length = (self.masspole * self.length)
21          self.total_mass = (self.masspole + self.masscart)
22
23      def change_cart_mass(self):
24          masscart_change = random.uniform(self.min_masscart_change, self.
    max_masscart_change)
25          self.masscart += masscart_change
26          self.total_mass = self.masscart + self.masspole
27
28      def change_cart_friction(self):
29          force_change = random.uniform(self.min_force_change, self.
    max_force_change)
30          self.force_mag = force_change
31
32      def step(self, action):
33          self.change_cart_friction()  # Change the pole length at each step
34          domain_shift = self.quantify_domain_shift()
35          observation, reward, terminated, truncated, info = super().step(
    action)
36          return (observation, reward, terminated, truncated, info),
    domain_shift
37
38      def reset(self):
39          self.length = self.original_length  # Reset the pole length when
    the environment is reset
40          return super().reset()
41
42      def quantify_domain_shift(self):
43          return abs(self.original_masscart - self.masscart)
44
```

```
45      def set_logger(self, logger):
46          self.logger = logger
```

Listing 3.1: Custom CartPole Environment

3.6 Reinforcement Learning Framework

The core of the reinforcement learning framework in this study consists of several components designed to interact seamlessly to achieve efficient learning and adaptation in the face of dynamic environments:

3.6.1 Neural Network Architectures

Two primary neural network models were employed in this study:

Deep Q-Network (DQN)

This model serves as the policy network, estimating the value of taking each possible action in a given state. The network architecture was designed to accommodate the augmented state space, which includes the traditional CartPole state information along with the domain shift metric. The DQN consists of three fully connected layers with ReLU activation functions. The input layer takes in the state observations and the domain shift value, while the output layer produces Q-values for each action. The hidden layers have 128 units each, providing sufficient capacity for the network to learn complex patterns and relationships between the state, domain shift, and action values.

```
1 class DQN(nn.Module):
2     def __init__(self, n_observations, n_actions, domain_shift_input_dim):
3         super(DQN, self).__init__()
4         self.layer1 = nn.Linear(n_observations + domain_shift_input_dim,
      128)
5         self.layer2 = nn.Linear(128, 128)
6         self.layer3 = nn.Linear(128, n_actions)
7
```

```
        def forward(self, x, domain_shift):
            domain_shift = domain_shift.view(-1, 1)  # Reshape to [batch_size, 1]
            x = torch.cat((x, domain_shift), dim=1)
            x = F.relu(self.layer1(x))
            x = F.relu(self.layer2(x))
            return self.layer3(x)
```

Listing 3.2: Deep Q-Network Architecture

Domain Shift Neural Network (DomainShiftNN)

This model assesses the current state's suitability under varying domain conditions, providing a mechanism for the agent to evaluate the impact of environmental changes on its decision-making process. The DomainShiftNN takes the state observations and the domain shift metric as input and outputs a suitability score between 0 and 1. It consists of two hidden layers with ReLU activation and an output layer with a sigmoid activation function. The network is trained using binary cross-entropy loss to predict the suitability of the current state-domain shift pair. The predicted suitability score is then used to adjust the agent's hyper-parameters dynamically, enabling it to adapt to the changing environment.

```

class DomainShiftPredictor:
    def __init__(self, input_dim, hidden_dim, output_dim, lr, suitability_threshold, adjustment_factor, device):
        self.model = DomainShiftNN(input_dim, hidden_dim, output_dim).to(device)
        self.optimizer = optim.AdamW(self.model.parameters(), lr=lr, amsgrad=True)
        self.loss_fn = nn.BCELoss()
        self.suitability_threshold = suitability_threshold
        self.adjustment_factor = adjustment_factor
        self.device = device
        # Added to keep track of episodes
        self.episode_count = 0
```

```
12
13      def predict_suitability(self, state, domain_shift_metric):
14          predictor_input = torch.cat((state, domain_shift_metric.unsqueeze
    (1)), dim=1)
15          predicted_suitability = self.model(predictor_input)
16          return predicted_suitability
17
18      def update(self, state, domain_shift_metric, true_suitability,
    random_suitability=None):
19          # If random_suitability is provided and we are in the first 200
    episodes, use it for training
20          if random_suitability is not None and self.episode_count < 200:
21              suitability = random_suitability
22          else:
23              suitability = self.predict_suitability(state,
    domain_shift_metric)
24
25          self.optimizer.zero_grad()
26          loss = self.loss_fn(suitability, true_suitability)
27          loss.backward()
28          self.optimizer.step()
29          # Increase episode count after each update call
30          self.episode_count += 1
31          return loss.item(), suitability.detach()
32
33  class DomainShiftNN(nn.Module):
34      def __init__(self, input_dim, hidden_dim, output_dim):
35          super(DomainShiftNN, self).__init__()
36          self.fc1 = nn.Linear(input_dim, hidden_dim)
37          torch.nn.init.kaiming_normal_(self.fc1.weight, nonlinearity='relu')
38          self.fc1.bias.data.fill_(0.01)
39
40          self.fc2 = nn.Linear(hidden_dim, hidden_dim)
41          torch.nn.init.kaiming_normal_(self.fc2.weight, nonlinearity='relu')
42          self.fc2.bias.data.fill_(0.01)
```

```
43
44          self.fc3 = nn.Linear(hidden_dim, output_dim)
45          torch.nn.init.kaiming_normal_(self.fc3.weight, nonlinearity='relu')
46          self.fc3.bias.data.fill_(0.01)
47
48      def forward(self, x):
49          x = F.relu(self.fc1(x))
50          x = F.relu(self.fc2(x))
51          x = torch.sigmoid(self.fc3(x))  # Assuming binary classification (0
    or 1) for suitability
52          return x
```

Listing 3.3: Domain Shift Neural Network

3.6.2 Action Selection Strategy

An epsilon-greedy policy, implemented within the ActionSelector class, guides the agent's action selection process. This approach ensures a balance between exploring new actions and exploiting known strategies, with an adjustable epsilon parameter controlling the degree of randomness over time. The ActionSelector maintains a decaying epsilon value, which determines the probability of selecting a random action instead of the action with the highest Q-value. As the agent gains more experience, the epsilon value decreases, gradually shifting the focus from exploration to exploitation. This strategy allows the agent to discover optimal policies while avoiding getting stuck in suboptimal solutions.

```
1       class ActionSelector:
2           def __init__(self, policy_net, num_actions, device, EPS_START,
    EPS_END, EPS_DECAY):
3               self.policy_net = policy_net
4               self.num_actions = num_actions
5               self.device = device
6               self.EPS_START = EPS_START
7               self.EPS_END = EPS_END
8               self.EPS_DECAY = EPS_DECAY
9               self.steps_done = 0
```

```
10            self.eps_thresholds = []
11
12        def select_action(self, state, domain_shift):
13            sample = random.random()
14            eps_threshold = self.EPS_END + (self.EPS_START - self.EPS_END)
    * math.exp(-1. * self.steps_done / self.EPS_DECAY)
15            self.steps_done += 1
16            self.eps_thresholds.append(eps_threshold)
17            self.policy_net.eval()
18            with torch.no_grad():
19                if sample > eps_threshold:
20                    return self.policy_net(state, domain_shift).max(1)[1].
    view(1, 1)
21                else:
22                    return torch.tensor([[random.randrange(self.num_actions
    )]], dtype=torch.long, device=self.device)
```

Listing 3.4: Action Selection Strategy

3.6.3 Experience Replay and Optimisation

The reinforcement learning algorithm incorporates an experience replay mechanism, managed by the ReplayMemory class, to store and sample transitions (state, action, reward, next state) for training the policy network. Experience replay helps to break the correlation between consecutive samples, stabilising the training process and improving sample efficiency. The ReplayMemory is implemented as a circular buffer, allowing the agent to learn from past experiences while maintaining a fixed memory size. The Optimizer class encapsulates the training logic, employing techniques such as mini-batch sampling, loss computation, and gradient-based updates to refine the model parameters. The optimizer samples a batch of transitions from the replay memory and computes the Q-learning loss using the target Q-values obtained from the target network. The loss is then back-propagated through the policy network, and the optimizer updates the network parameters using the Adam optimisation algorithm. The use of a separate target network,

which is periodically synchronised with the policy network, stabilises the learning process by providing a fixed reference for calculating the expected Q-values.

```
    class ReplayMemory(object):
        def __init__(self, capacity):
            self.memory = deque([], maxlen=capacity)

        def push(self, *args):
            self.memory.append(Transition(*args))

        def sample(self, batch_size):
            return random.sample(self.memory, batch_size)

        def __len__(self):
            return len(self.memory)

    Transition = namedtuple('Transition', ('state', 'action', 'next_state',
    'reward', 'domain_shift'))
```

Listing 3.5: Replay Memory

```

class Optimizer:
    def __init__(self, policy_net, target_net, optimizer, replay_memory,
    device, batch_size, gamma, tau):
        self.policy_net = policy_net
        self.target_net = target_net
        self.optimizer = optimizer
        self.memory = replay_memory
        self.device = device
        self.BATCH_SIZE = batch_size
        self.GAMMA = gamma
        self.TAU = tau
        self.losses = []

    def optimize(self):
            if len(self.memory) < self.BATCH_SIZE:
```

```
16                 return
17
18             transitions = self.memory.sample(self.BATCH_SIZE)
19             batch = Transition(*zip(*transitions))
20
21             non_final_mask = torch.tensor([s is not None for s in batch.
    next_state], device=self.device, dtype=torch.bool)
22             non_final_next_states = torch.cat([s for s in batch.next_state
    if s is not None])
23             non_final_domain_shifts = torch.cat([ds for s, ds in zip(batch.
    next_state, batch.domain_shift) if s is not None])  # Next state domain
    shifts
24
25             state_batch = torch.cat(batch.state)
26             action_batch = torch.cat(batch.action)
27             reward_batch = torch.cat(batch.reward)
28             domain_shift_batch = torch.cat(batch.domain_shift)  # Current
    state domain shifts
29
30             state_action_values = self.policy_net(state_batch,
    domain_shift_batch).gather(1, action_batch)
31
32             next_state_values = torch.zeros(self.BATCH_SIZE, device=self.
    device)
33             with torch.no_grad():
34                 next_state_actions = self.policy_net(non_final_next_states,
     non_final_domain_shifts).max(1)[1].unsqueeze(1)
35                 next_state_values[non_final_mask] = self.target_net(
    non_final_next_states, non_final_domain_shifts).gather(1,
    next_state_actions).squeeze(1)
36
37             expected_state_action_values = (next_state_values * self.GAMMA)
     + reward_batch
38
39             loss = F.smooth_l1_loss(state_action_values,
```

```
    expected_state_action_values.unsqueeze(1))
40          self.losses.append(loss.item())  # Store the loss value
41
42          self.optimizer.zero_grad()  # Zero the gradients before the
    backward pass
43          loss.backward()  # Compute the backward pass
44          torch.nn.utils.clip_grad_value_(self.policy_net.parameters(),
    100)  # Gradient clipping
45          self.optimizer.step()  # Take a step with the optimizer
46
47          # Soft update the target network
48          for target_param, policy_param in zip(self.target_net.
    parameters(), self.policy_net.parameters()):
49              target_param.data.copy_(self.TAU * policy_param.data + (1.0
     - self.TAU) * target_param.data)
50
51      return loss
```

Listing 3.6: Optimiser

3.6.4 Logging and Visualisation

The DataLogger class was developed to facilitate comprehensive logging of experimental data, including episode metrics, actions taken, rewards received, and the observed domain shifts. This data is crucial for post-experiment analysis, enabling the examination of learning trends, agent behavior under different conditions, and the overall effectiveness of the implemented strategies.

```
1   class DataLogger:
2       def __init__(self, filename):
3           self.filename = filename
4           self.fields = ['episode', 'step', 'original_length', '
    current_length', 'action', 'reward', 'domain_shift', 'cumulative_reward'
    , 'epsilon', 'loss']
5           self.ensure_file()
6
```

```
        def ensure_file(self):
            if not os.path.isfile(self.filename):
                with open(self.filename, 'w', newline='') as f:
                    writer = csv.DictWriter(f, fieldnames=self.fields)
                    writer.writeheader()

        def log_step(self, episode, step, original_length, current_length, action, reward, domain_shift, cumulative_reward, epsilon, loss):
            with open(self.filename, 'a', newline='') as f:
                writer = csv.DictWriter(f, fieldnames=self.fields)
                writer.writerow({
                    'episode': episode,
                    'step': step,
                    'original_length': original_length,
                    'current_length': current_length,
                    'action': action,
                    'reward': reward,
                    'domain_shift': domain_shift,
                    'cumulative_reward': cumulative_reward,
                    'epsilon': epsilon,
                    'loss': loss
                })
```

Listing 3.7: Data Logger

3.6.5 Hyper-parameter Tuning and Experimentation

The hyper-parameter optimisation process was conducted using a systematic approach to identify the best configurations for the models. Initially, Optuna trials were performed by varying a single hyper-parameter at a time, while keeping the others fixed. This approach allowed for the identification of the most influential hyper-parameters and their optimal ranges. Subsequently, the less critical hyper-parameters were not fixed to specific values but instead allowed to vary within predefined ranges during the Optuna trials. This strategy enabled the collection of aggregate data across a wide range of hyper-parameter

combinations, allowing each trial to self-select the best values based on the exploration adjustment mechanism. By conducting multiple trials with varying hyper-parameters, the final dataset represents an aggregated collection of results, providing a comprehensive view of the models' performance under different settings. This approach ensures that the evaluation is based on a diverse set of hyper-parameter configurations, increasing the robustness and generalisability of the findings.

The project leverages Optuna for systematic hyper-parameter optimisation, where it seeks configurations that maximise the agent's performance in the dynamic Cart Pole environment. The Optuna optimisation process involves defining a search space for key parameters (e.g., learning rate, batch sise, epsilon decay rate) and employing a trial-and-error approach guided by the performance metric of average reward over recent episodes.

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

def set_seed(seed):
    np.random.seed(seed)
    torch.manual_seed(seed)
    if torch.cuda.is_available():
        torch.cuda.manual_seed(seed)
        print('Using CUDA')

set_seed(1)
# best model
global best_value
best_value = -float('inf')

env, policy_net, target_net, optimizer, action_selector, optimizer_instance
    = initialize_environment(config)


def objective(trial):
    global best_value

    # suggest values for tunable hyperparameters
```

```
    lr = trial.suggest_float('lr', 1e-5, 1e-2, log=True)
    eps_decay = trial.suggest_int('eps_decay', 100, 2000)
    batch_size = trial.suggest_categorical('batch_size', [32, 64, 128, 256])
    gamma = trial.suggest_float('gamma', 0.8, 0.9999)

    # Update the config with the suggested values
    config.update({
        "lr": lr,
        "eps_decay": eps_decay,
        "batch_size": batch_size,
        "gamma": gamma,
    })

    # reinitialize the environment with the updated values
    env, policy_net, target_net, optimizer, action_selector, optimizer_instance = initialize_environment(config)


    # Use the hyperparameters from the config dictionary
    PERFORMANCE_THRESHOLD = config['performance_threshold']

    # initialise environment and components
    memory = ReplayMemory(config['replay_memory_size'])
    optimizer_instance.memory = memory

    # domain shift predictor necessary values
    input_dim = env.observation_space.shape[0] + 1 # size of the input (state + domain shift value)
    hidden_dim = 128 # size of the hidden layers
    output_dim = 1 # Size of the output (1 if suitable, 0 otherwise)
    suitability_threshold = 0.4
    adjustment_factor = 0.9 # factor to readjust hyperparams

    # Instantiate the domain shift class
```

```
    domain_shift_module = DomainShiftPredictor(input_dim, hidden_dim,
output_dim, lr, suitability_threshold, adjustment_factor, device)

    # For plotting function
    fig, axs = plt.subplots(4, 1, figsize=(10, 7))  # Create them once here
    episode_durations = []
    losses = optimizer_instance.losses
    eps_thresholds = []
    episode_rewards = []

    # Logging function
    logger = DataLogger('friction_and
_mass_random_change_training_data_with_predictor.csv')
    env.set_logger(logger)

    num_episodes = 4000
    for i_episode in range(num_episodes):
        state, info = env.reset()
        state = torch.tensor(state, dtype=torch.float32, device=device).
unsqueeze(0)
        episode_total_reward = 0
        policy_net.train()
        predicted_suitability = None  # Initialize outside the loop

        for t in count():
            domain_shift_metric = env.quantify_domain_shift()
            domain_shift_tensor = torch.tensor([domain_shift_metric], dtype
=torch.float32, device=device)

            if i_episode < 200:
                # Use random suitability for the first 200 episodes
                random_suitability = torch.tensor([[np.random.rand()]],
device=device)
                action = action_selector.select_action(state,
random_suitability)
```

```
83                 predicted_suitability = random_suitability
84             else:
85                 # Use the DSP model's prediction for suitability
86                 predicted_suitability = domain_shift_module.
    predict_suitability(state, domain_shift_tensor)
87                 action = action_selector.select_action(state,
    predicted_suitability)
88
89             # Take the action and observe the new state and reward
90             (observation, reward, terminated, truncated, info),
    domain_shift = env.step(action.item())
91             reward = torch.tensor([reward], device=device)
92
93             # Determine true suitability based on the episode outcome
94             true_suitability = torch.tensor([[1.0]], device=device) if not
    (terminated or truncated) else torch.tensor([[0.0]], device=device)
95
96             # Update the domain shift model
97             if predicted_suitability is not None:
98                 loss, _ = domain_shift_module.update(state,
    domain_shift_tensor, true_suitability)
99
100            if i_episode >= 200:
101                    # Update the DSP model after the first 200 episodes
102                loss, _ = domain_shift_module.update(state,
    domain_shift_tensor, true_suitability)
103
104            done = terminated or truncated
105            episode_total_reward += reward.item() # accumulate reward
106
107            if not done:
108                next_state = torch.tensor(observation, dtype=torch.float32,
     device=device).unsqueeze(0)
109            else:
110                next_state = None
```

```
111
112             memory.push(state, action, next_state, reward,
    domain_shift_tensor)
113             state = next_state
114             loss = optimizer_instance.optimize()
115
116             if loss is not None:
117                 # Log step data
118                 logger.log_step(
119                 episode=i_episode,
120                 step=t,
121                 original_length=env.original_length,
122                 current_length=env.length,
123                 action=action.item(),
124                 reward=reward.item(),
125                 domain_shift=domain_shift,
126                 cumulative_reward=episode_total_reward,
127                 epsilon=action_selector.get_epsilon_thresholds()[-1],
128                 loss=loss.item(), # This assumes `optimize()` returns a
    loss, otherwise you'll need to get it another way
129                 original_masscart= env.original_masscart,
130                 current_mass= env.total_mass,
131                 original_friction= env.original_force_mag,
132                 current_friction= env.force_mag,
133                 predicted_suitability=predicted_suitability.item(),
134                 )
135
136             if done:
137                 episode_durations.append(t + 1)
138                 break
139
140         if predicted_suitability.item() < suitability_threshold:
141             action_selector.reset_epsilon()
142
143         episode_rewards.append(episode_total_reward)
```

```
144
145         if len(episode_rewards) >= 100:
146             average_reward = np.mean(episode_rewards[-100:])
147             if average_reward > PERFORMANCE_THRESHOLD:
148                 action_selector.update_epsilon()
149
150         # Get the current epsilon threshold after update
151         current_eps_threshold = action_selector.get_epsilon_thresholds()
    [-1]
152         eps_thresholds.append(current_eps_threshold)  # Append the latest
    epsilon value
153
154         # Plot the graphs wanted
155         plot_function(fig, axs, episode_durations, losses, eps_thresholds,
    episode_rewards, optimization_mode=False)
156
157         trial.report(episode_durations[-1], i_episode)
158
159         if trial.should_prune():
160             raise optuna.TrialPruned()
161
162     mean_reward = np.mean(episode_rewards[-100:]) if len(episode_rewards)
    >= 100 else np.mean(episode_rewards)
163     if mean_reward > best_value:
164         best_value = mean_reward
165         torch.save(policy_net.state_dict(), 'friction_and
    _mass_cartpole_v1_best_model_DSP_Random.pth')
166
167     return mean_reward
```

Listing 3.8: Hyperparameter Tuning with Optuna

3.6.6 Version Control and Reproducibility

To ensure the reproducibility of the experiments and facilitate collaboration, the project's code base was managed using Git, a distributed version control system. This approach allowed for tracking changes, experimenting with different strategies in isolated branches, and maintaining a comprehensive history of the development process. The use of Git was instrumental in organising the project, enabling tests done with and without predictors easily.

4 Evaluation

The evaluation of the proposed reinforcement learning framework with domain shift quantification and adaptive exploration was conducted through a series of experiments primarily in the CCP environment. The primary focus was to assess the impact of the domain shift predictor on the agent's learning performance and adaptability under varying environmental conditions, particularly when multiple domain shifts are present simultaneously. The experiments were designed to provide a comprehensive analysis of the framework's effectiveness in terms of convergence speed, stability, mean reward, and robustness to multiple domain shifts.

4.1 Convergence Speed and Stability

One of the key metrics for evaluating the effectiveness of the proposed approach is the speed of convergence to the optimal policy and the stability of the learned policy. By comparing the training data of models with and without the domain shift predictor, a notable improvement in convergence speed and stability was observed when the predictor was employed.

Figures 2.1 and 2.2 illustrate the training data for linear changes in pole length, with and without the domain shift predictor, respectively. The model equipped with the domain shift predictor (Figure 2.2) converged to a stable mean reward between 300 and 400 more rapidly and consistently compared to the model without the predictor (Figure 2.1), which exhibited a lower mean reward range of 200 to 300 and greater variability. The domain shift predictor enables the agent to adapt its exploration strategy dynamically, leading to faster convergence and more stable performance in the presence of domain shifts.

The improved convergence speed and stability can be attributed to the domain shift predictor's ability to quantify the magnitude of the domain shifts and adjust the exploration rate accordingly. By dynamically adapting the exploration strategy based on the detected environmental changes, the agent can more efficiently navigate the state space and find optimal policies. This adaptive behavior allows the agent to quickly respond to domain shifts and maintain stable performance, even when the environment undergoes significant changes.

Furthermore, the domain shift predictor helps to mitigate the negative impact of domain shifts on the learning process. In the absence of the predictor, the agent may struggle to adapt to environmental changes, leading to slower convergence and unstable performance. The predictor's ability to detect and quantify domain shifts enables the agent to proactively adjust its exploration strategy, preventing it from getting stuck in suboptimal policies and facilitating faster convergence to stable and high-performing policies.

This is most apparent in Figures 2.12 and 2.11, where the model with the domain shift predictor has much better reward in the early stages. It is much more consistent when reaching the optimal values. Critically, it is clear to see that the model with the domain shift predictor maintains superior stability, where cases in the model without the domain shift predictor have sudden drop-offs from the optimal values. This is due to the domain shift predictor performing consistently better under unexpected domain shifts.

4.2 Mean Reward and Performance

Another critical aspect of the evaluation is the overall performance of the agent, as measured by the mean reward achieved during training. The experimental results demonstrate that the model incorporating the domain shift predictor consistently attained higher mean rewards compared to models without the predictor, both in the presence of single and multiple domain shifts.

Figures 2.5 and 2.6 present the training data for non-linear changes in pole length, with and without the domain shift predictor, respectively. The model with the predictor (Figure 2.6)

exhibits a higher concentration of episodes stabilising around a mean cumulative reward of 200 or above, while the model without the predictor (Figure 2.5) has a majority of episodes converging to a lower mean reward of approximately 150. This suggests that the adaptive exploration guided by the domain shift predictor contributes to improved performance and stability, even in the presence of non-linear domain shifts.

The superior performance of the proposed framework is further highlighted when multiple domain shifts, such as changes in both friction and mass, are introduced simultaneously. Figures 2.7 and 2.8 compare the mean cumulative reward per episode for models with and without the domain shift predictor under these conditions. The model with the predictor (Figure 2.7) achieves significantly higher mean cumulative rewards, consistently reaching values above 3000 and even approaching 4000 in some episodes. In contrast, the model without the predictor (Figure 2.8) struggles to achieve comparable reward levels, with the mean cumulative reward mostly fluctuating between 20 and 30. This stark difference in performance highlights the effectiveness of the domain shift predictor in enabling the agent to adapt to multiple simultaneous domain shifts.

The higher mean rewards achieved by the model with the domain shift predictor can be attributed to its ability to dynamically adjust the exploration-exploitation trade-off based on the detected domain shifts. By quantifying the magnitude of the domain shifts, the predictor enables the agent to adapt its exploration rate, allowing it to gather more information about the environment when necessary and exploit its knowledge when the environment is stable. This adaptive behavior leads to more efficient learning and higher overall performance, as the agent can quickly identify and adapt to the optimal policy in the presence of domain shifts.

Moreover, the domain shift predictor's ability to handle multiple simultaneous domain shifts is particularly noteworthy. In real-world scenarios, it is common for environments to undergo multiple changes concurrently, such as variations in friction, mass, and other physical properties. The proposed framework's robustness in adapting to these complex scenarios highlights its potential for practical applications, where the ability to handle multiple domain

shifts is crucial for maintaining high performance and reliability.

4.3 Hyper-parameter Optimisation

The utilisation of Optuna for hyper-parameter optimisation played a crucial role in fine-tuning the models' performance. Figures 2.3 and 2.4 depict the optimisation history for the best trials of models with and without the domain shift predictor, respectively, under non-linear changes in pole length. The model with the predictor (Figure 2.4) achieves a higher maximum objective value of 600 compared to the model without the predictor (Figure 2.3), which reaches a maximum value of 400. This indicates that the incorporation of the domain shift predictor enables the model to attain substantially better performance in terms of the optimisation objective.

The benefits of the domain shift predictor are further emphasised in the optimisation history plots for models under simultaneous changes in friction and mass (Figures 2.9 and 2.10). The model with the predictor (Figure 2.9) achieves a significantly higher maximum objective value of approximately 600, compared to the model without the predictor (Figure 2.10), which reaches a maximum value of only around 60. The model with the domain shift predictor exhibits a wider spread of objective values, mostly ranging from 100 to 600, with a small portion of trials achieving values above 400. In contrast, the model without the predictor shows a more concentrated distribution of objective values, mostly falling between 40 and 50. This suggests that the domain shift predictor enables the model to explore a broader range of solutions and consistently achieve higher objective values across multiple trials.

The hyper-parameter optimisation results demonstrate the importance of tuning the models' configurations to achieve optimal performance. The domain shift predictor introduces additional parameters that need to be carefully adjusted to maximise its effectiveness. Optuna's ability to efficiently search the hyper-parameter space and identify the best configurations for each model variant highlights the significance of automated hyper-parameter optimisation in the development of reinforcement learning

frameworks.

Furthermore, the hyper-parameter optimisation process provides insights into the robustness and stability of the proposed framework. The consistently higher objective values achieved by the model with the domain shift predictor across different hyper-parameter configurations indicate that the predictor's benefits are not limited to a specific set of hyper-parameters but rather generalise well across a wide range of settings. This robustness is essential for the practical application of the framework, as it ensures that the performance gains are maintainable and reproducible across different environments and domain shift scenarios.

4.4 Robustness to Multiple Domain Shifts

A key objective of this study was to assess the agent's adaptability and robustness to multiple simultaneous domain shifts. The Custom Cart Pole environment, with its variations in pole length, mass, and friction, served as a suitable test-bed for evaluating this aspect. The experimental results consistently demonstrated that the model incorporating the domain shift predictor exhibited superior performance and faster adaptation to multiple domain shifts compared to models without the predictor.

Figures 2.7 and 2.8 showcase the stark difference in performance when both friction and mass are varied simultaneously. The model with the domain shift predictor (Figure 2.7) maintains consistently high mean cumulative rewards throughout the episodes, with only minor fluctuations. In contrast, the model without the predictor (Figure 2.8) struggles to maintain stable performance, exhibiting more pronounced fluctuations in the mean cumulative reward. This indicates that the domain shift predictor enhances the model's ability to handle multiple domain shifts concurrently, resulting in more robust and stable performance.

The robustness of the proposed framework to multiple domain shifts can be attributed to several factors. First, the domain shift predictor's ability to quantify the magnitude of each domain shift independently allows the agent to assess the overall impact of the combined

shifts on its performance. By considering the cumulative effect of multiple shifts, the agent can make more informed decisions about how to adjust its exploration strategy to adapt to the changing environment.

Second, the adaptive exploration mechanism guided by the domain shift predictor enables the agent to dynamically balance the exploration-exploitation trade-off in the presence of multiple domain shifts. When the environment undergoes significant changes due to multiple shifts, the agent can increase its exploration rate to gather more information about the new dynamics and quickly adapt its policy. As the agent becomes more confident about its understanding of the environment, it can gradually reduce its exploration and focus on exploiting its learned knowledge to maximise rewards.

Third, the use of deep neural networks in the proposed framework allows for the learning of complex representations and relationships between the state space, action space, and domain shifts. The neural networks can capture the intricate interactions between multiple domain shifts and their impact on the agent's performance, enabling the agent to make more accurate predictions and take appropriate actions in the face of concurrent shifts.

The robustness to multiple domain shifts demonstrated by the proposed framework highlights its potential for real-world applications, where environments are often subject to various types of changes simultaneously. The ability to adapt and maintain stable performance in such complex scenarios is crucial for the successful deployment of reinforcement learning agents in practical settings, such as robotics, autonomous vehicles, and process control systems.

4.5 Comparative Analysis

To further validate the effectiveness of the proposed framework, we conducted a comparative analysis against baseline approaches. The models with and without the domain shift predictor were evaluated under various scenarios, including linear changes in pole length (Figures 2.1 and 2.2), non-linear changes in pole length (Figures 2.5 and 2.6), simultaneous changes in friction and mass (Figures 2.7 and 2.8), and changes in the force parameter for

the mountain car environment (Figures 2.12 and 2.11).

Across all scenarios, the model with the domain shift predictor consistently outperformed the baseline model without the predictor. The adaptive exploration strategy guided by the domain shift predictor enables the agent to quickly adjust its behavior and find optimal policies in the presence of domain shifts. The proposed framework demonstrates faster convergence, higher mean rewards, and more stable performance compared to the baseline approach.

The comparative analysis also highlights the scalability and effectiveness of the proposed framework in handling multiple domain shifts simultaneously. While the baseline model struggles to maintain stable performance and achieve high rewards when both friction and mass are varied (Figure 2.8), the model with the domain shift predictor exhibits remarkable robustness and consistently achieves high rewards (Figure 2.7). Additionally, the model with the domain shift predictor is clearly more stable in the mountain car environment (Figure 2.12 and 2.11). This underscores the importance of the domain shift predictor in enabling the agent to adapt to complex and dynamic environments.

The superior performance of the proposed framework in comparison to the baseline approaches can be attributed to several key factors. First, the domain shift predictor's ability to quantify the magnitude of domain shifts provides the agent with valuable information about the changing environment. This information allows the agent to make more informed decisions about how to adjust its exploration strategy and adapt its policy to the new conditions. In contrast, the baseline models without the predictor lack this explicit information about the domain shifts, limiting their ability to adapt effectively.

Second, the adaptive exploration mechanism guided by the domain shift predictor enables the agent to dynamically adjust its behavior based on the detected shifts. By increasing exploration when the environment undergoes significant changes and reducing exploration when the environment is stable, the agent can efficiently learn and adapt to new policies. The baseline models, on the other hand, rely on fixed exploration strategies that may not be optimal for the changing environment, leading to slower adaptation and suboptimal

performance.

Third, the integration of deep neural networks in the proposed framework allows for the learning of complex representations and relationships between the state space, action space, and domain shifts. The neural networks can capture the intricate dependencies between these elements and enable the agent to make more accurate predictions and take appropriate actions in the presence of domain shifts. The baseline models, lacking the domain shift predictor and the associated neural network architecture, may struggle to capture these complex relationships and adapt to the changing environment effectively.

The comparative analysis demonstrates the significant benefits of incorporating the domain shift predictor and adaptive exploration mechanism into the reinforcement learning framework. The proposed approach consistently outperforms the baseline models in terms of convergence speed, mean rewards, stability, and robustness to multiple domain shifts. These results highlight the potential of the proposed framework to address the challenges posed by domain shifts in real-world reinforcement learning applications.

4.6 Conclusion

The evaluation of the proposed reinforcement learning framework with domain shift quantification and adaptive exploration demonstrates its superior performance and adaptability compared to baseline approaches. The incorporation of the domain shift predictor enables faster convergence, higher mean rewards, and more stable performance in the presence of both single and multiple domain shifts.

The experimental results highlight the effectiveness of the adaptive exploration strategy guided by the domain shift predictor, allowing the agent to quickly adjust its behavior and find optimal policies in non-stationary environments. The framework's ability to handle multiple simultaneous domain shifts, as evidenced by its robust performance when both friction and mass are varied, underscores its potential for real-world applications.

The comparative analysis against baseline models further validates the benefits of the

proposed framework, demonstrating its consistent out performance across various scenarios. The domain shift predictor's ability to quantify and adapt to environmental changes dynamically proves to be a critical component in enhancing the agent's adaptability and performance.

The hyper-parameter optimisation process using Optuna plays a crucial role in fine-tuning the models' performance and ensuring a fair comparison between the proposed framework and the baseline approaches. The consistently higher objective values achieved by the model with the domain shift predictor across different hyper-parameter configurations highlight the robustness and generalisability of the proposed approach.

However, it is important to acknowledge the limitations and potential areas for future research. While the proposed framework has shown significant improvements over baseline approaches, further investigations are needed to assess its performance in more diverse and challenging environments. Additionally, the scalability of the approach to high-dimensional state and action spaces requires further exploration and optimisation.

Moreover, the interpretability of the learned policies and the domain shift predictor's decisions is an important consideration for real-world applications. Developing methods to visualise and explain the agent's behavior and the factors influencing its decisions can enhance the trust and acceptability of the proposed framework in practical settings.

Future research directions could include the integration of transfer learning techniques to enable the agent to leverage knowledge from previous tasks and adapt more efficiently to new domain shifts. Additionally, the incorporation of multi-agent reinforcement learning approaches could be explored to address the challenges of coordinated adaptation in multi-agent systems. Most necessary, the future research directions should experiment with more difficult learning environments, such as the bipedal walker, as well as adjusting the domain shift quantification metric to something, akin to the euclidean distance, that could be more suitable.

The comprehensive evaluation, encompassing convergence speed, mean rewards, hyper-parameter optimisation, robustness to multiple domain shifts, and comparative

analysis, provides a solid foundation for understanding the strengths and limitations of the proposed framework. The insights gained from this evaluation can guide future research efforts and inform the development of more advanced and practical reinforcement learning approaches that can tackle the complexities of real-world environments.

By demonstrating the effectiveness of the proposed approach in the custom Cart Pole environment, this evaluation lays the groundwork for further investigations and applications in more complex and realistic scenarios. The lessons learned from this study can be extended and adapted to address the challenges of domain shifts in a wide range of real-world problems, opening up new possibilities for the deployment of reinforcement learning in practical settings.

The experimental results, supported by comprehensive comparative analysis and hyper-parameter optimisation, demonstrate the potential of the proposed framework to address the challenges posed by domain shifts in real-world reinforcement learning applications. The ability to quantify and adapt to environmental changes dynamically proves to be a critical component in enhancing the agent's performance and resilience in the face of uncertainty.

Furthermore, the evaluation highlights the scalability and practicality of the proposed approach, showcasing its ability to handle complex environments with multiple simultaneous domain shifts. The integration of deep neural networks allows for the learning of intricate relationships and representations, enabling the agent to make accurate predictions and take appropriate actions in dynamic settings.

The insights gained from this comprehensive evaluation provide a solid foundation for future research and development efforts in the field of reinforcement learning. By addressing the challenges of domain shifts and enabling agents to adapt to environmental changes, the proposed framework represents a significant step towards the practical deployment of reinforcement learning in real-world settings.

As the field of reinforcement learning continues to evolve and find applications in various domains, the ability to handle domain shifts becomes increasingly critical for the success and

reliability of these systems. The proposed framework, with its domain shift quantification and adaptive exploration capabilities, offers a promising approach to tackle these challenges and pave the way for more resilient and adaptable reinforcement learning systems.

The evaluation presented in this study serves as a testament to the potential of the proposed framework and its contributions to the advancement of reinforcement learning in the face of domain shifts. By demonstrating its effectiveness in the custom Cart Pole environment, this evaluation lays the groundwork for further investigations and applications in more complex and realistic scenarios, opening up new possibilities for the deployment of reinforcement learning in practical settings.

In summary, the evaluation of the proposed reinforcement learning framework with domain shift quantification and adaptive exploration provides compelling evidence of its superior performance, adaptability, and robustness compared to baseline approaches. The experimental results, supported by comprehensive comparative analysis and hyper-parameter optimisation, highlight the potential of the proposed approach to address the challenges of domain shifts in real-world applications and contribute to the development of more resilient and adaptable reinforcement learning systems.

5 Conclusion

5.1 Limitations and Potential Drawbacks

While the proposed framework demonstrates significant improvements in adapting to domain shifts, there are certain limitations and potential drawbacks to consider. One observed limitation is that, in some cases, the domain shift predictor may output the same suitability value regardless of the situation, leading to no adjustments in the exploration rate. This issue was mitigated by introducing random changes near the start of the training process, allowing the model to learn the immediate impacts of the adjustments. However, this highlights the sensitivity of the approach to the initial learning phase and the need for careful initialisation strategies. Another potential drawback is the dependence of the epsilon adjustment function on the specific characteristics of the domain shifts. The current implementation assumes a fixed adjustment formula, but in practice, the optimal adjustment may vary depending on the magnitude and nature of the shifts. Larger shifts may require more substantial corrections, while smaller shifts may benefit from more subtle adjustments. Developing a more adaptive and context-aware adjustment mechanism could further enhance the framework's ability to handle diverse domain shift scenarios effectively.

5.2 Benefits and Potential Use-Cases

The evaluation of the proposed reinforcement learning framework with domain shift quantification and adaptive exploration yields promising results. The incorporation of the domain shift predictor leads to faster convergence, higher mean rewards, and a more

concentrated reward distribution around optimal actions. These findings suggest that the predictor's ability to dynamically adjust the exploration rate based on detected domain shifts enhances the agent's adaptability and decision-making capabilities in non-stationary environments. The utilisation of Optuna for hyper-parameter optimisation further ensures that the models are well-tuned, providing a fair comparison between the different approaches. Overall, the experimental results support the effectiveness of the proposed framework in addressing the challenges posed by domain shifts in reinforcement learning.

The faster convergence observed in the experiments indicates that the domain shift predictor enables the agent to quickly adapt its behavior when encountering changes in the environment. By quantifying the degree of domain shift, the predictor allows the agent to adjust its exploration rate accordingly, striking a balance between exploiting its current knowledge and exploring new strategies. This adaptive exploration mechanism proves to be particularly beneficial in non-stationary environments, where the optimal policy may change over time.

The proposed framework has significant practical implications for various real-world applications. In robotics, the ability to adapt to changing environments is crucial for the successful deployment of autonomous systems. Our approach could enable robots to dynamically adjust their behavior based on detected domain shifts, improving their robustness and reliability in unpredictable environments. In the field of personalised recommendations, the framework could be applied to adapt recommendation strategies based on shifts in user preferences or context, leading to more relevant and engaging recommendations. Moreover, the domain shift predictor could be integrated into existing reinforcement learning systems to enhance their adaptability and performance in non-stationary settings. However, deploying the framework in real-world applications would require careful consideration of factors such as computational efficiency, over-fitting, and defaulting to outputting a single value. Further research is needed to address these challenges and ensure the integration of the approach into practical systems.

Moreover, the higher mean rewards and more concentrated reward distribution around

optimal actions demonstrate the improved performance and stability of the proposed framework. The domain shift predictor helps the agent make more informed decisions by considering the current environmental context. This context-awareness allows the agent to select actions that are more likely to yield favorable outcomes, resulting in higher rewards and a more consistent performance across different domains.

The use of Optuna for hyper-parameter optimisation plays a crucial role in ensuring the fairness and reliability of the experimental results. By systematically searching for the best hyper-parameter configurations, Optuna helps to mitigate the impact of sub-optimal settings on the performance of the models. This optimisation process ensures that each approach is evaluated based on its inherent strengths and weaknesses, rather than being influenced by the choice of hyper-parameters.

While the proposed framework demonstrates improvements over traditional reinforcement learning approaches, there are still several avenues for future research. One potential direction is to explore the integration of more advanced domain adaptation techniques, such as transfer learning or meta-learning, to further enhance the agent's ability to generalise across different domains. Additionally, investigating the scalability of the framework to more complex and high-dimensional environments could provide insights into its applicability to real-world scenarios.

Furthermore, the proposed reinforcement learning framework with domain shift quantification and adaptive exploration represents a promising approach for tackling the challenges of non-stationary environments. The experimental results highlight the benefits of incorporating a domain shift predictor and utilising adaptive exploration strategies. By dynamically adjusting the exploration rate based on detected domain shifts, the agent can effectively adapt its behavior and improve its performance in changing environments. The framework's ability to converge faster, achieve higher rewards, and maintain a more stable performance demonstrates its potential for real-world applications. However, further research is needed to explore its scalability and investigate the integration of advanced domain adaptation techniques. Overall, this work contributes to the advancement of reinforcement

learning in non-stationary environments and paves the way for the development of more robust and adaptable intelligent systems.

5.2.1 Potential Use Cases in Robotics

The current work opens up several exciting avenues for future research. One promising direction is to extend the framework to multi-agent settings, where multiple agents collaborate or compete in non-stationary environments. Investigating how the domain shift predictor can be adapted to capture the dynamics of multi-agent interactions and facilitate coordinated exploration could lead to more effective and resilient multi-agent systems. Another area of interest is the incorporation of prior knowledge or expert demonstrations into the framework. Leveraging domain expertise or demonstration data could accelerate the learning process and improve the agent's performance in complex environments. Additionally, exploring the integration of the proposed approach with other reinforcement learning algorithms, such as actor-critic methods or hierarchical reinforcement learning, could yield novel insights and further enhance the adaptability and efficiency of the framework. Finally, investigating the application of the framework to real-world problems, such as autonomous driving or autonomous systems, would provide valuable insights into its practicality and potential benefits.

Robotic systems are increasingly being deployed in a wide range of applications, from manufacturing and logistics to healthcare and autonomous vehicles. However, one of the major challenges in robotics is the ability to adapt to changing environments and handle domain shifts effectively. Robots often encounter variations in their operating conditions, such as changes in lighting, surface properties, or object appearances, which can significantly impact their performance and reliability.

Our proposed framework can be leveraged to enhance the adaptability and robustness of robotic systems in the face of domain shifts. By incorporating a domain shift predictor and adaptive exploration strategies, robots can dynamically adjust their behavior based on the detected environmental changes. This adaptability is crucial for ensuring the successful

deployment of robots in real-world settings, where the operating conditions may vary significantly from the training environment.

Let's consider a specific example of a robotic arm employed in a manufacturing assembly line. The robot is trained to pick and place objects from a conveyor belt and assemble them into a final product. However, the objects on the conveyor belt may vary in terms of their shape, size, and material properties, introducing domain shifts that can affect the robot's grasping and manipulation capabilities.

By integrating our reinforcement learning framework into the robotic system, we can enable the robot to quantify the domain shifts and adapt its exploration strategies accordingly. The domain shift predictor can assess the suitability of the robot's current policy based on the observed changes in the object properties. If a significant domain shift is detected, the robot can dynamically adjust its exploration rate to gather more information about the new objects and refine its grasping strategies.

This adaptive exploration allows the robot to quickly learn and adapt to the variations in the objects, improving its success rate in grasping and manipulating them. The robot can explore different grasping angles, force levels, and trajectories to find the optimal approach for each object type. By continuously updating its policy based on the feedback received from the environment, the robot can maintain a high level of performance even in the presence of domain shifts.

Moreover, the proposed framework can be extended to multi-robot systems, where multiple robots collaborate to accomplish a common task. In such scenarios, the domain shift predictor can be used to coordinate the exploration strategies of the robots, ensuring that they adapt to environmental changes in a synchronised manner. This coordination can lead to more efficient task allocation and improved overall system performance.

The benefits of applying our reinforcement learning framework to robotic systems are numerous. First, it enhances the adaptability and robustness of robots, enabling them to handle a wide range of domain variations without significant performance degradation. This adaptability is essential for the successful deployment of robots in real-world environments,

where the operating conditions may be unpredictable and dynamic.

Second, our approach can reduce the need for extensive retraining or manual intervention when robots encounter domain shifts. By autonomously adapting their behavior based on the detected changes, robots can continue operating effectively without requiring human intervention or costly downtime for retraining. This autonomy can lead to increased efficiency, reduced maintenance costs, and improved overall system reliability.

Third, the proposed framework can facilitate the transfer of learned policies across different robotic platforms or environments. By quantifying the domain shifts and adapting the exploration strategies accordingly, robots can more easily generalise their learned skills to new settings. This transferability can accelerate the deployment of robots in various applications and reduce the development time and effort required for each new scenario.

Furthermore, the insights gained from applying our framework to robotic systems can contribute to the advancement of robotic learning and control algorithms. The domain shift quantification and adaptive exploration techniques can be incorporated into existing robotic learning frameworks, enhancing their ability to handle non-stationary environments. The lessons learned from real-world deployments can also inform the design of future robotic systems, leading to more adaptable and resilient architectures.

In conclusion, the application of our reinforcement learning framework to robotics holds immense potential for addressing the challenges of domain shifts in real-world scenarios. By enabling robots to quantify and adapt to environmental changes autonomously, our approach can significantly enhance the adaptability, robustness, and efficiency of robotic systems.

5.2.2 Potential Use Cases in Autonomous Aerial Vehicles

Autonomous aerial vehicles, commonly known as drones, have gained significant attention in recent years due to their wide range of applications, including surveillance, mapping, delivery, and search and rescue operations. However, the successful deployment of drones in real-world scenarios faces several challenges, particularly in terms of adapting to varying

environmental conditions and handling domain shifts. Our proposed reinforcement learning framework with domain shift quantification and adaptive exploration can be effectively applied to address these challenges and enhance the capabilities of autonomous drones.

One of the primary issues faced by drones is the need to operate in diverse and dynamic environments. Drones may encounter variations in weather conditions, such as wind speed and direction, temperature, and precipitation, which can significantly impact their flight dynamics and performance. Additionally, drones may need to navigate through different terrains, such as urban landscapes, forests, or mountainous regions, each presenting unique challenges and requiring adaptive control strategies.

By integrating our reinforcement learning framework into the drone's control system, we can enable the drone to quantify the domain shifts and adapt its behavior accordingly. The domain shift predictor can assess the suitability of the drone's current control policy based on the observed environmental changes. For example, if the drone detects a sudden change in wind conditions, the predictor can determine the extent of the domain shift and trigger an adjustment in the exploration rate.

This adaptive exploration allows the drone to gather more information about the new environmental conditions and refine its control strategies. The drone can explore different flight parameters, such as altitude, speed, and trajectory, to find the optimal configuration for the current conditions. By continuously updating its policy based on the feedback received from the environment, the drone can maintain stable and efficient flight performance even in the presence of domain shifts.

Moreover, the proposed framework can be extended to multi-drone systems, where a swarm of drones collaborates to accomplish a common mission. In such scenarios, the domain shift predictor can be used to coordinate the exploration strategies of the drones, ensuring that they adapt to environmental changes in a synchronised manner. This coordination can lead to improved mission efficiency, increased coverage area, and enhanced overall system robustness.

The benefits of applying our reinforcement learning framework to autonomous drones are

significant. First, it enhances the adaptability and resilience of drones, enabling them to operate effectively in a wide range of environmental conditions. This adaptability is crucial for the successful deployment of drones in real-world applications, where the operating conditions may be unpredictable and constantly changing.

Second, our approach can reduce the need for manual intervention or mission replanning when drones encounter domain shifts. By autonomously adapting their behavior based on the detected changes, drones can continue their missions without requiring human operators to manually adjust their control parameters. This autonomy can lead to increased mission success rates, reduced operational costs, and improved response times in critical situations.

Third, the proposed framework can facilitate the transfer of learned policies across different drone platforms or mission scenarios. By quantifying the domain shifts and adapting the exploration strategies accordingly, drones can more easily generalise their learned skills to new environments. This transferability can accelerate the deployment of drones in various applications and reduce the development time and effort required for each new mission.

Furthermore, the insights gained from applying our framework to autonomous drones can contribute to the advancement of drone control algorithms and mission planning strategies. The domain shift quantification and adaptive exploration techniques can be incorporated into existing drone control frameworks, enhancing their ability to handle dynamic environments. The lessons learned from real-world deployments can also inform the design of future drone systems, leading to more adaptable and resilient architectures.

In conclusion, the application of our reinforcement learning framework to autonomous aerial vehicles (drones) holds immense potential for addressing the challenges of domain shifts in real-world scenarios. By enabling drones to quantify and adapt to environmental changes autonomously, our approach can significantly enhance the adaptability, robustness, and efficiency of drone operations. The benefits extend beyond improved performance, as the framework can also reduce the need for manual intervention, facilitate policy transfer, and contribute to the advancement of drone control algorithms. As the demand for autonomous

drones continues to grow across various industries, our framework provides a promising solution for enabling their successful deployment in complex and dynamic real-world environments.

5.2.3 Potential Use Cases in Energy Management for Renewable Energy Networks

The integration of renewable energy sources, such as solar and wind power, into the electrical grid has become a critical focus in the pursuit of sustainable energy solutions. However, the inherent variability and intermittency of renewable energy pose significant challenges for effective energy management and grid stability. Our proposed reinforcement learning framework with domain shift quantification and adaptive exploration can be applied to address these challenges and optimise the operation of renewable energy networks.

In renewable energy networks, the power generation from solar panels and wind turbines is highly dependent on environmental conditions, such as solar irradiance, wind speed, and weather patterns. These conditions can vary significantly over time, introducing domain shifts that can impact the efficiency and reliability of the energy management system. Traditional control strategies often struggle to adapt to these variations, leading to sub-optimal performance and potential grid instability.

By incorporating our reinforcement learning framework into the energy management system, we can enable the system to quantify the domain shifts and adapt its control strategies accordingly. The domain shift predictor can assess the suitability of the current energy management policy based on the observed changes in renewable energy generation and demand patterns. For example, if the system detects a sudden drop in solar power generation due to cloud cover, the predictor can determine the extent of the domain shift and trigger an adjustment in the exploration rate.

This adaptive exploration allows the energy management system to gather more information about the new operating conditions and refine its control strategies. The system can explore different power dispatch schemes, energy storage utilisation, and demand response measures

to find the optimal configuration for the current conditions. By continuously updating its policy based on the feedback received from the environment, the energy management system can maintain stable and efficient operation even in the presence of domain shifts.

Moreover, the proposed framework can be extended to coordinate the operation of multiple renewable energy sources and energy storage systems within a micro-grid or a larger power network. In such scenarios, the domain shift predictor can be used to synchronise the exploration strategies of different components, ensuring that they adapt to environmental changes in a coordinated manner. This coordination can lead to improved system resilience, reduced energy losses, and enhanced overall grid stability.

The benefits of applying our reinforcement learning framework to energy management in renewable energy networks are substantial. First, it enhances the adaptability and robustness of the energy management system, enabling it to handle the variability and intermittency of renewable energy sources effectively. This adaptability is crucial for the reliable integration of renewable energy into the electrical grid and the transition towards a more sustainable energy future.

Second, our approach can improve the efficiency and cost-effectiveness of renewable energy utilisation. By autonomously adapting the control strategies based on the detected domain shifts, the energy management system can optimise the power dispatch, minimise energy losses, and reduce the reliance on expensive backup power sources. This optimisation can lead to cost savings for energy providers and consumers, as well as reduced environmental impact.

Third, the proposed framework can facilitate the integration of advanced energy management techniques, such as demand response and energy storage optimisation. By quantifying the domain shifts and adapting the exploration strategies accordingly, the system can more effectively leverage these techniques to balance supply and demand, smooth out power fluctuations, and enhance grid stability. This integration can unlock new opportunities for flexible and intelligent energy management in renewable energy networks.

Furthermore, the insights gained from applying our framework to energy management can

contribute to the development of more advanced and resilient energy systems. The domain shift quantification and adaptive exploration techniques can be incorporated into existing energy management algorithms, enhancing their ability to handle the challenges posed by renewable energy integration. The lessons learned from real-world deployments can also inform the design of future energy networks, leading to more adaptable and sustainable architectures.

In conclusion, the application of our reinforcement learning framework to energy management in renewable energy networks holds immense potential for addressing the challenges of domain shifts and optimising the operation of these systems. By enabling the energy management system to quantify and adapt to environmental changes autonomously, our approach can significantly enhance the adaptability, efficiency, and reliability of renewable energy integration. The benefits extend beyond improved performance, as the framework can also facilitate the adoption of advanced energy management techniques, contribute to cost savings, and support the transition towards a more sustainable energy future. As the world continues to embrace renewable energy sources, our framework provides a promising solution for enabling their effective management and integration into the electrical grid.

5.2.4 Potential Use Cases in Space Applications

Space exploration and the development of space technologies have always been at the forefront of scientific and technological advancement. However, the harsh and unpredictable nature of the space environment poses significant challenges for the operation and control of spacecraft and satellites. Our proposed reinforcement learning framework with domain shift quantification and adaptive exploration can be applied to address these challenges and enhance the autonomy and resilience of space systems.

In space missions, spacecraft and satellites are exposed to a wide range of environmental conditions and operational scenarios that can vary significantly from those encountered during ground-based testing and simulations. These variations can be considered as domain shifts, which can impact the performance and reliability of the space systems. Examples of

domain shifts in space include changes in solar radiation intensity, temperature fluctuations, cosmic ray exposure, and variations in gravitational forces.

By integrating our reinforcement learning framework into the control systems of spacecraft and satellites, we can enable these systems to quantify the domain shifts and adapt their behavior accordingly. The domain shift predictor can assess the suitability of the current control policy based on the observed changes in the space environment. For example, if a satellite detects a sudden increase in radiation levels, the predictor can determine the extent of the domain shift and trigger an adjustment in the exploration rate.

This adaptive exploration allows the spacecraft or satellite to gather more information about the new environmental conditions and refine its control strategies. The system can explore different operational modes, power management schemes, and communication protocols to find the optimal configuration for the current conditions. By continuously updating its policy based on the feedback received from the environment, the space system can maintain stable and efficient operation even in the presence of domain shifts.

Moreover, the proposed framework can be extended to enable collaborative decision-making among multiple spacecraft or satellites in a space mission. In such scenarios, the domain shift predictor can be used to coordinate the exploration strategies of different systems, ensuring that they adapt to environmental changes in a synchronised manner. This coordination can lead to improved mission performance, increased fault tolerance, and enhanced overall mission success.

The benefits of applying our reinforcement learning framework to space technologies are numerous. First, it enhances the autonomy and adaptability of spacecraft and satellites, enabling them to operate effectively in the dynamic and unpredictable space environment. This autonomy is crucial for long-duration missions, deep space exploration, and missions where real-time human intervention is impractical or impossible.

Second, our approach can improve the resilience and fault tolerance of space systems. By autonomously adapting their behavior based on the detected domain shifts, spacecraft and satellites can mitigate the impact of adverse conditions and maintain their functionality even

in the presence of system failures or anomalies. This resilience can lead to increased mission reliability, extended operational lifetimes, and reduced mission risks.

Third, the proposed framework can facilitate the development of more advanced and intelligent space systems. By quantifying the domain shifts and adapting the exploration strategies accordingly, space systems can leverage machine learning and artificial intelligence techniques to make informed decisions and optimise their performance. This integration can enable new capabilities, such as autonomous navigation, on-board data processing, and adaptive resource management.

Furthermore, the insights gained from applying our framework to space technologies can contribute to the advancement of space exploration and scientific discovery. The domain shift quantification and adaptive exploration techniques can be incorporated into existing space mission planning and control algorithms, enhancing their ability to handle the challenges posed by the space environment. The lessons learned from real-world deployments can also inform the design of future space systems, leading to more adaptable and resilient architectures.

In conclusion, the application of our reinforcement learning framework to space technologies holds immense potential for addressing the challenges of domain shifts and enhancing the autonomy and resilience of space systems. By enabling spacecraft and satellites to quantify and adapt to environmental changes autonomously, our approach can significantly improve mission performance, reliability, and scientific return. The benefits extend beyond improved system performance, as the framework can also facilitate the development of advanced space technologies, contribute to mission success, and support the ongoing exploration and understanding of the universe. As humanity continues to push the boundaries of space exploration, our framework provides a promising solution for enabling the next generation of autonomous and adaptive space systems.

5.3 Future Considerations & Work

Our research has so far concentrated on the Cart-Pole and Mountain-Car environments, which have both served as excellent platforms for studying the adaptability and efficacy of reinforcement learning models when faced with domain shifts. To expand the scope of our work and further challenge our domain shift adaptation strategies, we propose exploring more complex and dynamic environments, such as the Gym Car Racing and Bipedal Walker environments.

Adapting the project code to these environments can prove tricky, however in the most recent project code on GitHub you will find the adaptations needed in order to make the changes. Critically, moving to the continuous action space versus the discrete action space of the mountain car and cart pole environments proves somewhat challenging. In discrete action spaces, actions are typically represented as finite and clearly defined choices, such as "move left", "move right", or "do nothing". This allows for a straightforward mapping of states to actions, often implemented using arrays or simple tensors. In contrast, a continuous action space involves actions that can take any value within a range, such as steering angles or throttle levels, which require a very different tensor structure to be passed between objects.

In continuous domains, actions must be represented by floating point numbers rather than integers. This requires modifying data structures and tensors within the code to accommodate and accurately represent these continuous values. Instead of choosing the best action from a set of indexed possibilities, the algorithm now needs to predict a value or set of values that directly correspond to the action dimensions. With continuous action spaces, the adaption also impacts the neural network architecture itself. For discrete actions, the output layer of the neural network typically has a dimension that matches the number of possible actions, with each neuron representing a different action's value. In continuous action spaces, the output would need to be a set of parameters that define a probability distribution.

Moreover, the transition to a continuous action space may necessitate changes to the overall architecture of the reinforcement learning algorithm. Some algorithms, such as Deep Q-Networks (DQN), are designed specifically for discrete action spaces and may not be directly applicable to continuous action spaces. In such cases, alternative algorithms like Deep Deterministic Policy Gradient (DDPG), Proximal Policy Optimization (PPO), or Soft Actor-Critic (SAC) may need to be employed.

Although the Bipedal Walker environment is adapted for the project, the time taken to train the model was 2-3 days per trial, and 80 trials are needed for the consistency of each model. This proved beyond the project timeline scope, so results for the domain shift predictor were omitted due to a lack of training data for the model without the predictor.

Additionally, it would be easiest to critically assess the performance that different domain shift quantification metrics have on the models. Changing from our equation (2.6) to the euclidean distance in a multidimensional space, or other appropriate distance equations.

The Gym Car Racing environment features a car that must navigate a racetrack as quickly as possible. This environment introduces challenges related to continuous control and visual processing, as the agent must interpret a complex visual input to control the vehicle effectively. Applying our domain shift adaptation approach in this environment could provide valuable insights into how our methods perform under conditions requiring real-time decision-making and high-frequency adaptations to visual and physical changes in the environment.

The Bipedal Walker environment, on the other hand, involves controlling a two-legged robot that must walk forward over rough terrain without falling. This scenario presents unique challenges due to the biped's physics and the need for maintaining balance while navigating varied surface obstacles. This environment will test the scalability of our adaptation techniques in scenarios where coordination and balance in a physically simulated world are critical, reflecting real-world challenges in robotics more closely.

Further investigation could also include integrating transfer learning techniques to examine

whether insights and strategies learned in simpler environments, such as the Cart-Pole or Mountain-Car, can be transferred to accelerate learning in these more complex settings. This approach could potentially reduce training times and computational demands, enhancing overall efficiency.

Exploring multi-agent reinforcement learning (MARL) in these environments could offer additional insights. For instance, multiple agents could learn to race against each other in the Car Racing environment or cooperate to navigate challenging terrains in the Bipedal Walker scenario. This would add a layer of complexity to the learning challenges and help us understand how agents develop strategies in competitive or cooperative contexts.

In the pursuit of developing more advanced and responsive reinforcement learning systems, incorporating techniques such as Dynamic Time Warping (DTW) and manifold learning into the domain shift quantification metrics could significantly enhance the agent's ability to detect and adapt to changes in the environment. DTW is an algorithm that measures the similarity between two temporal sequences, allowing for non-linear mappings and flexible alignment between the elements of the sequences. By applying DTW to compare the state trajectories or action sequences of the agent across different domain shifts, it becomes possible to quantify the extent of the domain shift and assess how much the agent's behavior deviates from its original policy. [17] [18]

For instance, if the agent's state trajectory in a new domain exhibits significant differences compared to its trajectory in the original domain, DTW would effectively capture this dissimilarity and provide a metric indicating the presence of a substantial domain shift. This information could then be utilized to trigger appropriate adjustments in the agent's exploration strategy or other adaptive mechanisms, enabling it to respond more effectively to the changing environment.

Moreover, manifold learning techniques offer a complementary approach to enhancing domain shift quantification. Manifold learning assumes that high-dimensional data lies on or near a lower-dimensional manifold embedded in the high-dimensional space. By applying manifold learning to the state representations or feature spaces of the reinforcement learning

agent, it becomes possible to discover the underlying low-dimensional structure of the environment.

When a domain shift occurs, the manifold representation of the state space may undergo changes. Comparing the manifold representations before and after the domain shift allows for the quantification of the extent of the change and the detection of significant deviations from the original domain. Techniques such as t-SNE (t-Distributed Stochastic Neighbor Embedding) or UMAP (Uniform Manifold Approximation and Projection) can be employed to visualize and compare the state space manifolds across different domains, providing valuable insights into how the agent's understanding of the environment evolves and adapts to domain shifts.

Furthermore, manifold learning can contribute to the reduction of the dimensionality of the state space, leading to improved computational efficiency and generalization. By operating in a lower-dimensional representation, the agent may be able to learn more efficiently and generalize better to new domains, enhancing its overall performance and adaptability. [19] [20]

The integration of DTW and manifold learning into the domain shift quantification metrics has the potential to significantly improve the sensitivity and accuracy of detecting and responding to environmental changes. DTW captures the temporal aspects of the agent's behavior and quantifies the dissimilarity between sequences across domains, while manifold learning reveals the underlying structure of the state space and detects changes in the manifold representation when domain shifts occur.

By extending the work into these challenging environments, we aim to push the boundaries of what our adaptive reinforcement learning frameworks can achieve, providing significant contributions to the fields of autonomous systems and adaptive robotics.

Glossary

Action Space The set of all possible actions that an agent can take in an environment, defining the range of decisions available to the agent at each step. 79

Actor-Critic Methods A class of reinforcement learning algorithms that combine the benefits of both value-based and policy-based methods. Actor-critic methods use a critic to estimate the value function and an actor to update the policy based on the critic's feedback. 79

Adaptive Exploration A strategy in reinforcement learning where the agent dynamically adjusts its exploration rate based on the characteristics of the environment or the learning progress, balancing the trade-off between exploiting current knowledge and exploring new actions. 79

Adjustment Factor A scaling factor used to modify the agent's exploration rate or other relevant parameters based on the detected domain shifts and the suitability metric, enabling dynamic adaptation to changing environmental conditions. 79

Backpropagation The process of computing the gradients of the loss function with respect to the model's parameters, allowing the optimisation algorithm to update the weights in the direction that minimises the loss. 79

Batch Size The number of training examples used in one iteration of the optimisation process. Larger batch sizes can lead to more stable gradients but may require more computational resources. 79

Bias-Variance Trade-off The balance between a model's ability to fit the training data

(bias) and its ability to generalise to new, unseen data (variance). A high-bias model may underfit the data, while a high-variance model may overfit the data. 79

Binary Cross-Entropy Loss A loss function commonly used in binary classification problems, measuring the dissimilarity between the predicted probabilities and the true binary labels, which is employed in training the Domain Shift Neural Network to predict the suitability of state-domain shift pairs. 79

Cart Friction The friction coefficient between the cart and the track in the CartPole environment. Modifying the cart friction can affect the behavior of the cart and introduce domain shifts that require the agent to adjust its strategy. 79

Cart Mass The mass of the cart in the CartPole environment. Changing the cart mass can alter the dynamics of the system and create domain shifts that the agent needs to adapt to. 79

Catastrophic Forgetting The tendency of a neural network to forget previously learned information when learning new information, often occurring when the model is trained on a sequence of tasks without revisiting old tasks. 79

Continual Learning The ability of a model to continuously learn and adapt to new tasks or environments without forgetting previously acquired knowledge. 79

Convergence The process of a learning algorithm approaching an optimal solution over time, characterised by a decrease in the loss function or an increase in the performance metric. 79

Cross-Validation A model validation technique that involves partitioning the data into subsets, training the model on a subset, and validating it on the remaining data. This process is repeated multiple times to assess the model's performance and generalisation ability. 79

Curriculum Learning A training approach where the complexity of the tasks presented to the model gradually increases over time, mimicking the way humans learn from simpler

to more complex concepts. 79

Custom CartPole Environment (CCP) A modified version of the classic CartPole control problem, designed to introduce domain shifts by varying the pole length, cart mass, and cart friction over time, challenging the agent to adapt to non-stationary dynamics. 79

Deep Q-Network A neural network architecture used to approximate the Q-function in reinforcement learning. It takes the state as input and outputs the Q-values for each possible action, enabling the agent to make informed decisions. 79

Discount Factor A value between 0 and 1 that determines the importance of future rewards in the agent's decision-making process. A higher discount factor places more emphasis on long-term rewards, while a lower value prioritises immediate rewards. 79

Domain Shift The phenomenon where the distribution of data or the characteristics of the environment change between the training and testing phases, leading to a degradation in the performance of a learned model. 79

Domain Shift Metric A numerical value that represents the degree of change or difference between the current environment and a reference or baseline environment, capturing the extent of the domain shift. 79

Domain Shift Neural Network (DSNN) A neural network architecture designed to predict the suitability of the current policy under different domain conditions, providing a mechanism for the agent to assess the impact of environmental changes on its decision-making process. 79

Domain Shift Predictor (DSP) A component in the proposed reinforcement learning framework that quantifies the magnitude of domain shifts and adjusts the exploration rate accordingly. The DSP learns to predict the suitability of the current policy for a given state and domain shift, allowing the agent to adapt its behavior dynamically. 79

Domain Shift Quantification The process of numerically measuring the magnitude or

extent of changes in the environment's dynamics or characteristics, enabling the agent to adapt its behavior accordingly. 79

Epsilon Decay A technique used in epsilon-greedy exploration, where the exploration rate (epsilon) is gradually decreased over time, allowing the agent to transition from initial exploration to exploitation of the learned policy as it gathers more experience. 79

Epsilon-Greedy Exploration A strategy used in reinforcement learning to balance exploration and exploitation. With probability epsilon, the agent takes a random action (exploration), and with probability 1-epsilon, it takes the action with the highest estimated Q-value (exploitation). 79

Experience Replay A technique used in reinforcement learning to store the agent's experiences (state, action, reward, next state) in a buffer and randomly sample from it during training. It helps to break the correlation between consecutive samples and stabilise the learning process. 79

Exploitation The act of an agent utilising its current knowledge to make decisions that maximise its expected reward, often at the cost of exploring new strategies or actions. 79

Exploration Rate The probability (epsilon) of taking a random action during epsilon-greedy exploration. It determines the trade-off between exploring new actions and exploiting the current best action. The exploration rate is typically decayed over time to gradually shift focus towards exploitation. 79

Generalisation The ability of a machine learning model to perform well on unseen data, i.e., data that was not used during the training process. 79

Gradient Descent An optimisation algorithm used to minimise the loss function of a model by iteratively adjusting its parameters in the direction of steepest descent of the gradient. 79

Hierarchical RL An approach to reinforcement learning where the agent learns to make

decisions at multiple levels of abstraction. The agent decomposes the task into a hierarchy of subtasks and learns policies for each subtask, allowing for more efficient learning and transfer of knowledge across related tasks. 79

Hyperparameter Search Space The range of possible values for each hyperparameter that the optimisation algorithm explores during the tuning process, defined based on prior knowledge or empirical observations to narrow down the search to promising regions. 79

Hyperparameter Tuning The process of searching for the optimal set of hyperparameters that yield the best performance of a model on a specific task. This can be done manually or through automated methods such as grid search or Bayesian optimisation. 79

Intrinsic Curiosity Module A reinforcement learning technique that encourages exploration by rewarding the agent for discovering novel or informative states. The Intrinsic Curiosity Module (ICM) consists of a forward model that predicts the next state and an inverse model that predicts the action taken between two states. The prediction error of the forward model is used as an intrinsic reward signal to guide exploration. 79

Inverse Reinforcement Learning A problem in reinforcement learning where the goal is to infer the reward function that an agent is optimising based on its observed behavior. Given a set of expert demonstrations, inverse reinforcement learning aims to recover the underlying reward function that explains the expert's actions. 79

Learning Rate A hyperparameter that controls the step size at which the model's weights are updated during the optimisation process. It determines the speed at which the model learns from new information. 79

Lifelong Learning A paradigm in machine learning where a model learns continuously throughout its lifetime, accumulating knowledge from various tasks and experiences. 79

Linear Domain Shift A type of domain shift where the changes in the environment parameters (e.g., pole length, cart mass, or cart friction) follow a linear pattern over time. Linear domain shifts represent a gradual and predictable variation in the environment. 79

Loss Function A function that measures the discrepancy between the predicted outputs of a model and the actual target values. The goal of the optimisation process is to minimise the loss function. 79

Markov Decision Process (MDP) A mathematical framework for modeling decision-making in situations where outcomes are partly random and partly under the control of a decision-maker, commonly used as a foundation for reinforcement learning algorithms. 79

Meta-Learning The process of learning to learn, where a model is trained on a variety of tasks and learns to adapt quickly to new tasks by leveraging its prior knowledge. 79

Model-Based RL A type of reinforcement learning where the agent learns a model of the environment's dynamics and uses it to plan its actions. The model captures the state transitions and rewards, allowing the agent to simulate the environment and make informed decisions. 79

Model-Free RL A type of reinforcement learning where the agent learns a policy directly from experience without explicitly modeling the environment's dynamics. The agent relies on trial-and-error interactions with the environment to estimate the optimal policy. 79

Monte Carlo Methods A class of computational algorithms that rely on repeated random sampling to estimate numerical results. In reinforcement learning, Monte Carlo methods are used to estimate the value function by averaging the returns of complete episodes. 79

Multi-Agent RL A subfield of reinforcement learning that deals with multiple agents interacting in a shared environment. The agents may have cooperative, competitive,

or mixed objectives, and they need to learn to coordinate or compete with each other to achieve their goals. 79

Non-Linear Domain Shift A type of domain shift where the changes in the environment parameters (e.g., pole length, cart mass, or cart friction) follow a non-linear or unpredictable pattern over time. Non-linear domain shifts introduce more complex and challenging variations in the environment. 79

Non-stationary Environment An environment where the dynamics, reward function, or other characteristics change over time, requiring the agent to adapt its behavior to maintain optimal performance. 79

Offline RL A setting in reinforcement learning where the agent learns from a fixed dataset of previously collected experiences without interacting with the environment during training. The goal is to learn a policy that performs well in the environment based on the available data. 79

Online RL A setting in reinforcement learning where the agent interacts with the environment and updates its policy in real-time. The agent learns from its own experiences as it explores and makes decisions in the environment. 79

Optimisation The process of finding the best set of parameters for a model that minimises the loss function and maximises its performance on a given task. 79

Optimisation History A record of the performance metrics and corresponding hyperparameter configurations evaluated during the optimisation process, used to analyse the progress of the search and identify the best-performing models. 79

Optuna An open-source hyperparameter optimisation framework that automates the process of finding the best hyperparameters for machine learning models, using techniques such as Bayesian optimisation and pruning of unpromising trials. 79

Over-fitting A situation where a model learns to fit the noise or random fluctuations in the training data, resulting in poor performance on unseen data. 79

Pole Length The length of the pole attached to the cart in the CartPole environment. Varying the pole length can introduce domain shifts and challenge the agent's adaptability. 79

Policy A function that maps states to actions, determining the behavior of the agent in the environment. It can be deterministic or stochastic, specifying the probability distribution over actions for each state. 79

Policy Gradient Methods A class of reinforcement learning algorithms that directly optimise the policy by following the gradient of the expected return with respect to the policy parameters. Policy gradient methods update the policy to maximise the expected cumulative reward. 79

Pruned Trials Trials in the hyperparameter optimisation process that are stopped early if they are found to be performing poorly, based on predefined criteria or intermediate results, to allocate resources more efficiently to promising configurations. 79

Q-Function A function that estimates the expected cumulative reward an agent can obtain by taking a specific action in a given state and following a particular policy thereafter. It represents the long-term desirability of taking an action in a state. 79

Reactive Exploration An exploration strategy in reinforcement learning that adapts the exploration rate based on the detected changes in the environment, allowing the agent to balance between exploiting its current knowledge and exploring new strategies in response to domain shifts. 79

Regularisation A technique used to prevent overfitting by adding a penalty term to the model's loss function, discouraging the model from learning overly complex patterns in the training data. 79

Reinforcement Learning A type of machine learning where an agent learns to make decisions by interacting with an environment, receiving rewards or penalties for its actions, and adapting its behavior to maximise the cumulative reward over time. 79

Reward Function A function that assigns a numerical value to each state-action pair, indicating the desirability or utility of taking a specific action in a given state, serving as the primary feedback signal for the agent to learn and optimise its behavior. 79

State Space The set of all possible states that an agent can encounter in an environment, representing the complete information about the current situation. 79

Suitability Metric A measure of how well-suited or adapted the agent's current policy or behavior is to the current environmental conditions, taking into account the detected domain shifts. 79

Suitability Threshold A predefined value that determines the minimum level of suitability required for the agent to maintain its current policy or behavior, triggering adjustments or adaptations when the suitability falls below this threshold. 79

Target Network A separate neural network used in Deep Q-Networks to provide stable target Q-values for training. It is periodically updated with the weights of the main Q-network to decouple the target from the rapidly changing Q-values. 79

Temporal Difference Learning A class of reinforcement learning algorithms that learn the value function by bootstrapping from the current estimate of the value function. TD methods update the value function based on the difference between the estimated value and the actual reward received. 79

Transfer Learning A machine learning technique where knowledge gained from solving one problem is applied to a different but related problem, allowing for faster learning and improved performance. 79

Transition Function A function that defines how the environment changes from one state to another based on the action taken by the agent. It specifies the probability distribution over the next states given the current state and action. 79

Under-fitting A situation where a model is too simple to capture the underlying patterns in the training data, resulting in poor performance on both training and unseen data. 79

Value Function A function that estimates the expected cumulative reward an agent can obtain from a given state by following a specific policy. It represents the long-term desirability of being in a particular state. 79

Bibliography

[1] C. Steinparz, T. Schmied, F. Paischer, M.-C. Dinu, V. Patil, A. Bitto-Nemling, H. Eghbal-zadeh, and S. Hochreiter, "Reactive exploration to cope with non-stationarity in lifelong reinforcement learning," 2022.

[2] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," in *ICML*, 2017.

[3] V. Sindhwani, *Optimization Perspectives on Robot Learning, Perception, and Control*. Vikas Sindhwani, Google Brain NYC, 2017.

[4] M. UK, *Reinforcement Learning with MatLab*. MathWorks, 2020.

[5] 2018.

[6] R. S. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artificial Intelligence*, vol. 112, no. 1, pp. 181–211, 1999. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370299000521

[7] P.-L. Bacon, J. Harb, and D. Precup, "The option-critic architecture," 2016.

[8] A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu, "Feudal networks for hierarchical reinforcement learning," 2017.

[9] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html

[10] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, "Progressive neural networks," 2022.

[11] A. A. Rusu, S. G. Colmenarejo, C. Gulcehre, G. Desjardins, J. Kirkpatrick, R. Pascanu, V. Mnih, K. Kavukcuoglu, and R. Hadsell, "Policy distillation," 2016.

[12] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," 2017.

[13] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel, "Rl^2: Fast reinforcement learning via slow reinforcement learning," 2016.

[14] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," 2017.

[15] M. E. Taylor, P. Stone, and Y. Liu, "Transfer learning via inter-task mappings for temporal difference learning," *Journal of Machine Learning Research*, vol. 8, no. 1, pp. 2125–2167, 2007.

[16] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015. [Online]. Available: http://dx.doi.org/10.1038/nature14236

[17] K. Bringmann, N. Fischer, I. van der Hoog, E. Kipouridis, T. Kociumaka, and E. Rotenberg, "Dynamic dynamic time warping," 2023.

[18] T. Belkhouja, Y. Yan, and J. R. Doppa, "Dynamic time warping based adversarial framework for time-series domain," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 6, p. 7353–7366, Jun. 2023. [Online]. Available: http://dx.doi.org/10.1109/TPAMI.2022.3224754

[19] Z. Zhu, K. Lin, A. K. Jain, and J. Zhou, "Transfer learning in deep reinforcement learning: A survey," 2023.

[20] A. Merckling, N. Perrin-Gilbert, A. Coninx, and S. Doncieux, "Exploratory state representation learning," *Frontiers in Robotics and AI*, vol. 9, 2022. [Online]. Available: https://www.frontiersin.org/articles/10.3389/frobt.2022.762051

A1 Appendix

A1.1 GitHub

A link to the GitHub code repository: GitHub

A1.2 Environments

A link to the CartPole environment: CartPole Environment

A link to the MountainCar environment: MountainCar Environment

A link to the Bipedal Walker environment: Bipedal Walker Environment

A1.3 Packages

Link to the Collections package: Collections

Link to the CSV package: CSV

Link to the Itertools package: Itertools

Link to the Math package: Math

Link to the Matplotlib package: MatPlotLib

Link to the Numpy package: Numpy

Link to the Optuna package: Optuna

Link to the OS package: OS

Link to the Random package: Random

Link to the Torch package: PyTorch