COMPARATIVE ANALYSIS OF MODEL-FREE REINFORCEMENT LEARNING ALGORITHMS IN DYNAMIC AND PARTIALLY OBSERVABLE GRID-WORLD ENVIRONMENTS

Thesis Submitted
in Partial Fulfillment of the
Requirements For the Degree Of

Master of Science in Applied Mathematics

Submitted by:

Shailly (2K23/MSCMAT/85)
Bhakti Sharma (2K23/MSCMAT/71)

Under the supervision of

Prof. Anjana Gupta

DEPARTMENT OF APPLIED MATHEMATICS
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Shahbad Daulatpur, Main Bawana Road, Delhi-110042, India

May 2025

DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Shahbad Daulatpur, Main Bawana Road, Delhi – 42

CANDIDATE'S DECLARATION

We, Bhakti Sharma (2K23/MSCMAT/71) and Shailly (2K23/MSCMAT/85), hereby affirm that the dissertation titled "Comparative Analysis of Model-Free Reinforcement Learning Algorithms in Dynamic and Partially Observable Grid-World Environments" represents our original research work, carried out in partial fulfillment of the criteria for the award of the Master of Science in Applied Mathematics.

This research has been completed under the guidance of Prof. Anjana Gupta, within the Department of Applied Mathematics, Delhi Technological University, spanning the period from August 2024 to May 2025.

We further confirm that the contents of this dissertation have not been submitted, either partially or fully, for the award of any other degree or diploma at this or any other academic institution.

The matter presented in this thesis has not been submitted by us for the award of any other degree of this or any other institute.

Bhakti Sharma

Shailly

Signature of Supervisor

Signature of External Examiner

DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Shahbad Daulatpur, Main Bawana Road, Delhi – 42

CERTIFICATE BY THE SUPERVISOR

Certified that **Bhakti Sharma (2K23/MSCMAT/71)** and **Shailly (2K23/MSCMAT/85)** have carried out their research work presented in this thesis entitled *"COMPARATIVE ANALYSIS OF MODEL-FREE REINFORCEMENT LEARNING ALGORITHMS IN DYNAMIC AND PARTIALLY OBSERVABLE GRID-WORLD ENVIRONMENTS"* for the award of **Master of Science in Applied Mathematics** from the DEPARTMENT OF APPLIED MATHEMATICS, DELHI TECHNOLOGICAL UNIVERSITY, Delhi, under my supervision. This thesis represents the findings of original research, and the students themselves conducted the investigations. The information in this thesis is not the foundation for the candidates or anyone else from this or any other university or institution to be awarded another degree.

Prof. Anjana Gupta

DEPARTMENT OF APPLIED MATHEMATICS

DELHI TECHNOLOGICAL UNIVERSITY

Date: _____

DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Shahbad Daulatpur, Main Bawana Road, Delhi – 42

ACKNOWLEDGEMENTS

We sincerely express our heartfelt gratitude to our friends and peers who supported us at various stages of this work. Lastly, we extend special thanks to our families, whose constant support and encouragement have been a strong pillar during this academic journey.

We would like to sincerely thank Prof. Anjana Gupta of Delhi Technological University for her essential advice, encouragement, and assistance during the preparation of this dissertation. She provided constant oversight, enlightening criticism, and encouragement throughout this effort, for which we are incredibly grateful. For providing the tools and a supportive environment for our work, we also appreciate the department head, all of the academic members, and the staff of the DEPARTMENT OF APPLIED MATHEMATICS, DELHI TECHNOLOGICAL UNIVERSITY, Delhi. We would like to extend our sincere gratitude to our friends and colleagues who helped us along the way. Finally, we would want to express our gratitude to our families, whose unwavering encouragement and support have formed a solid foundation throughout this academic journey.

Date: _____

ABSTRACT

This dissertation presents a comparative evaluation of three primary categories of model-free reinforcement learning i.e. RL approaches — Q Learning, Policy-Gradient, and Actor-Critic —within a specially designed Gridworld environment. This environment replicates complex, real-life decision-making challenges, incorporating two major elements: partial observability, where the agent has limited visibility of the environment, and dynamic goal positioning, where the goal location changes during training. Each RL method is developed using a consistent framework and tested over multiple independent runs to maintain statistical rigor. The assessment focuses on various performance indicators such as accumulated rewards, convergence trends, training stability, and adaptability in response to environmental fluctuations. Q-Learning, though robust and easy to implement, exhibits delayed adaptation due to its static value update structure. Policy Gradient methods show better responsiveness to dynamic goals but suffer from policy update variance. Actor-Critic algorithms combine advantages from both approaches, yielding balanced performance with comparatively stable training behavior. The research also explores how tuning hyperparameters, incorporating exploration strategies, and applying reward shaping can influence learning outcomes. The findings are supported through visual data and performance graphs, offering insight into selecting the most suitable RL strategy for real-world applications like autonomous systems, robotic control, and adaptive resource management in uncertain settings. Ultimately, this study outlines the trade-offs among model-free methods and suggests possible directions for future research involving hybrid or meta-learning frameworks.

Keywords: Reinforcement-Learning, Q Learning, Policy Gradient, Actor-Critic, Partial Observability

Contents

Chapter 1

Introduction

1.1 Background and Motivation

A subset of machine learning known as reinforcement learning (RL) involves an agent interacting with its surroundings to determine and carry out successive judgments. The agent's goal is to identify the best course of action, or policy, that maximizes the cumulative reward (gain), or sum of rewards, over a given period of time. In contrast to supervised learning, which uses labeled data to train the model, reinforcement learning (RL) learns a policy through a process of trial and error depending on input from the environment.

Model-free reinforcement learning, a prominent category in this field, enables learning directly from interactions without the need of a prior model of the environment. Among all these methods, Q-Learning, Policy Gradient, and Actor-Critic techniques are widely recognized for their effectiveness in various domains. Q-Learning is an approach based on value which focuses on estimating the optimal action and corresponding value function. Policy Gradient methods, whereas, aims at directly learning the policy that the agent should follow, while Actor-Critic algorithms integrate both value estimation and policy learning to enhance performance.

The growing complexity of real-world environments necessitates robust RL algorithms capable of dealing with uncertain, dynamic, and partially observable conditions. Traditional benchmarks often fail to capture these intricacies. Therefore, this study introduces a customized Gridworld environment that integrates two challenging features: partial observability, where the agent has limited perception of its surroundings, and dynamic goal positions, requiring the agent to continuously adapt its strategy.

This dissertation aims to conduct a comparative analysis of Q Learning (value-based technique), Policy-Gradient (policy based technique), and Actor-Critic (a

combination of these two models) methods in the designed environment. Each algorithm is developed under a consistent setup, evaluated across several runs, and assessed based on performance metrics such as convergence speed, learning stability, total reward accumulation, and responsiveness to changes in the goal location.

By analyzing the strengths and limitations of each approach in such a setting, this work seeks to provide a better understanding of their applicability in real-world scenarios such as robotics, adaptive control, and autonomous navigation. Additionally, the study highlights the role of key design choices, such as reward shaping and exploration strategies, in influencing learning outcomes. The ultimate goal is to guide the selection of appropriate RL strategies for complex, non-deterministic environments.

1.2 Problem Statement

This dissertation focuses on a comparative analysis of three foundational categories of model-free reinforcement learning algorithms:

- Q Learning (A Value Based approach)
- Policy-Gradient (A Policy-Based approach)
- Actor-Critic (Hybrid Approach)

These algorithms are evaluated within a custom-designed Gridworld environment characterized by partial observability and a moving goal state. The environment simulates a realistic navigation challenge where the agent receives limited perceptual input and must adjust its policy as the goal location changes unpredictably over time.

1.3 Research Objectives

The main objectives of the concerned study are as following:

- To design a dynamic and partially observable Gridworld environment that can test adaptability and stability of RL agents.
- To implement and train Q-Learning (a value-based), Policy Gradient & Actor-Critic algorithms within this environment.
- To analyze the working of these algorithms based on cumulative rewards, learning stability, and adaptability to changes.

- To identify trade-offs between stability, speed of convergence, and variance in learning across different algorithm types.

1.4 Scope of Study

The study focuses exclusively on RL algorithms that are free from models and doesn't explore model-based approaches. The environment is designed to be partially observable (the agent can only perceive a limited area around itself) and non-stationary (the goal changes location during training). The implementations are kept intentionally simple to highlight core algorithmic differences rather than architectural improvements such as deep neural networks or memory augmentation.

Chapter 2

Literature Review

2.1 Overview of Research in RL

Reinforcement-Learning (RL) has undertaken several improvements in past few decades, establishing itself as a robust and much needed approach for making decisions based on sequential methodology. Rooted in behavioral psychology, RL pushes the agent involved to perceive and implement actions in the environment, encouraged by feedback which represents the gain or loss incurred as rewards or penalties.

2.1.1 Foundations and Evolution of Reinforcement Learning

The foundational theories of RL were initially influenced by concepts of classical and operant conditioning. Sutton and Barto's seminal work (2018) offered a formalized mathematical framework for RL, emphasizing the agent-environment interaction loop. In this setup, the optimal action is selected by the agent on the basis of policies and the penalty or reward it received and utilizing this information to take the cumulative rewards to maximum.

Two main approaches have dominated RL research: based on model and other one is not based on model i.e. model free. Methods developed on concept of models attempt to build a model representing the environmental dynamics and then utilize it for further action taking. Conversely, approaches free of model, such as Q-Learning (a value based) and Policy- Gradient approaches, bypass the need for environmental modeling, focusing directly on learning policies or value functions through trial-and-error interactions.

2.1.2 Q-Learning and Its Variants

Q-Learning, first introduced by Watkins in the year 1989, is a widely studied value-based algorithm in model-free RL. It estimates the expected effectiveness of several actions in given set of states, known as the Q-values, and gradually updates them based on the Bellman equation. Despite its simplicity and convergence guarantees under certain conditions, Q-Learning struggles in higher dimensions or environments that are partially observable due to the curse of dimensionality.

To address scalability issues, Deep Q-Networks (DQNs) were proposed by Mnih et al. (in the year 2015), integrating deep neural networks to approximate Q-values. DQNs marked a major milestone in deep reinforcement learning by reaching the level of human performance in the Atari 2600 games. Enhancements such as Double-DQN, Dueling DQN, and Prioritized Experience Replay were later introduced for improving stability and effective learnings.

2.1.3 Policy Gradient Methods

In distinction to certain value-based methods, the policy gradient method approaches to evaluate the policy chosen by agent. These methods utilizes the gradient ascent to maximize the anticipated giving back by tuning the policy framework. The REINFORCE algorithm (by Williams, in the year 1992) laid the foundation for policy-gradient techniques, though it suffered from a high (increased) variance in estimates of gradients.

Advancements like the Actor-Critic architecture have mitigated this issue by combining the powers of learnings based on policy and values. Here, the "actor" changes the strategy directionally while the "critic" analyzes the action using value functions, providing a stabilizing signal for learning.

2.1.4 Actor-Critic Algorithms

Actor-Critic methods represent a hybrid class in RL, enabling both exploration through policy updates and stability via value estimation. Because of their efficacy in complex contexts and continuous control tasks, variants like the Asynchronous-Advantage Actor-Critic (A3C) and Advantage Actor Critic (A2C) have grown in popularity. These algorithms significantly reduce variance while maintaining convergence stability, making them suitable for Markov decision processes (POMDPs) which is partially observable.

Recent innovations including Proximal Policy Optimization (P-P-O) and Soft-Actor-Critic (SAC) have additionally improved actor-critic models by introducing mechanisms that ensure stable policy updates and improved sample efficiency.

PPO, in particular, balances exploration and exploitation through a clipped objective function, while SAC introduces entropy regularization to encourage more stochastic policies.

2.1.5 Reinforcement Learning in Partially Observable Environments

Real-world environments are often not fully observable, posing challenges to standard RL algorithms. In such cases, Partially Observable-Markov Decision Processes (POMDPs) are applied to evaluate the uncertainty in state information. RL algorithms adapted to POMDPs typically incorporate memory components such as Recurrent Neural Networks (RNNs) to infer latent states based on action-observation histories.

Model-free methods like Recurrent DQN (Hausknecht & Stone, 2015) and Recurrent Policy Gradient approaches have demonstrated efficacy in handling partial observability by maintaining temporal context through internal state representations. These innovations enhance decision-making in domains such as robotics, autonomous navigation, and strategic gaming.

The landscape of model-free reinforcement learning has been enriched by diverse algorithmic innovations that cater to different challenges, such as high-dimensional state spaces, partial observability, and sample inefficiency. While Q-Learning provides a solid baseline for discrete tasks, policy gradient and actor-critic methods offer scalable solutions for continuous and uncertain environments. The comparative analysis in this dissertation will explore these techniques in the context of dynamic and partially observable gridworld environments.

2.1.6 Summary of Literature Gaps

Table 2.1: Summarized Literature Gaps in RL Algorithms

Aspect	**Existing Work**	**Gap Identified**
Value-based RL	Strong in simple, static setups	Weak in partial observability
Policy-based RL	Good adaptability	High variance, slow convergence
Actor-Critic	Balanced performance	Underexplored in dynamic tasks
Dynamic POMDPs	Sparse coverage	Few comparative RL studies

Chapter 3

Theoretical Background

This section outlines the fundamental concepts and mathematical formulations that underpin model-free reinforcement learning algorithms. These include the structure of reinforcement learning problems, value functions, policy representation, and the core principles of Q-Learning-value based approach, Policy-Gradient, and Actor-Critic methods. These theoretical foundations provide the basis for the experimental comparisons conducted later in the study.

3.1 Markov Decision Processe (MDP)

Markov Decision Processes (MDPs), which attempt to provide a formal framework for a sequential decision-making method in uncertain contexts, are frequently used to describe RL difficulties. A tuple that describes an MDP is as follows:

$$(S, A, P, R, \gamma) \tag{3.1}$$

Where:

- S: A set of all possible states in environment
- A: A set of all possible actions that can be taken
- $P(s'|s, a)$: Transition probability of reaching state s' from state s by undertaking action a
- $R(s, a)$: Reward received after taking action a in state s.
- γ: Discount factor ($0 \leq \gamma < 1$) responsible for balancing right-away and upcoming rewards

The aim of an agent in the MDP is to figure out a policy $\pi(a|s)$ that takes the anticipated cumulative (total) reward (discounted) to the maximum:

$$\mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^t R(s_t, a_t)\right] \tag{3.2}$$

3.2 Value Functions

Value function aims at analysing the benefits or gain that can be obtained by being in a particular state or undergoing a particular action. Two basic types of value-based functions are used:

- The expected return when beginning with state s and adhering to policy $\pi(s)$ is State-value function $V^{\pi}(s)$:

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty}\gamma^t R(s_t, a_t)\middle| s_0 = s\right] \tag{3.3}$$

- Action-value-function $Q^{\pi}(s, a)$: Anticipated return after undergoing an action in a given state and then following the given course of policy:

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

These functions play a central role in most RL algorithms.

3.3 Q-Learning

Regardless of the policy being followed, the Q-Learning approach is a form of off-policy technique, a free from model algorithm that must determine the best fit action and matching value function $Q^*(s, a)$. The Bellman optimality equation is used to update it:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left[r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)\right] \tag{3.5}$$

Where:

- α: Learning rate

- r_t: Immediate reward received at time t

For exploratory purposes, the policy is guided by the learned Q-values and usually employs a ϵ-greedy strategy, in which the agent performs an action at random with a probability of ϵ and the most efficient known action otherwise.

3.4 Policy Gradient Methods

Policy gradient tends represent policies precisely as a parameterized function $\pi_\theta(a|s)$ and optimize the anticipated return $J(\theta)$ utilizing gradient ascent:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta) \tag{3.6}$$

The REINFORCE algorithm uses the following gradient estimate:

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi \left[\nabla_\theta \log \pi_\theta(a|s) \cdot G_t \right] \tag{3.7}$$

Where G_t is the return from time t. While simple and effective, REINFORCE incorporates very higher variance in the estimates of gradients, which can slower down or destabilize the procedure of learning.

3.5 Actor-Critic Methods

The advantages of value-based and policy-based approaches are combined in the actor-critical technique. In order to determine the value function, the actor updates the policy parameters in the direction suggested by the critic.

- The stated actor illustrates the policy $\pi_\theta(a|s)$
- The concerned critic approximates the function value for this - $V_w(s)$ or advantage $A(s, a)$

The function at advantage, which reduces variance, is explained:

$$A(s, a) = Q(s, a) - V(s) \tag{3.8}$$

The update rules become:

- Actor: $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a|s) \cdot A(s, a)$
- Critic: $w \leftarrow w + \beta \delta \nabla_w V_w(s)$, where δ is the temporal difference error.

This architecture enables faster and more stable convergence in complicated or environments that are partially observable.

3.6 Partial Observability and POMDPs

In real-world scenarios, agents often lack full knowledge of the environment's state. As a result, Partially-Observable-Markov Decision Processes (POMDPs) are used, in which the agent keeps a belief about potential states based on observations and past actions.

Formally, a POMDP is described by tuples:

$$(S, A, P, R, \Omega, O, \gamma) \tag{3.9}$$

Wherein:

- Ω: A set of observations
- $O(o|s', a)$: Observation function giving probability of seeing o after action a leads to state s'.

Recurrent-Neural Networks (RNNs) or Long-Short-Term Memory (LSTM) networks are frequently used in model-free reinforcement learning (RL) to manage partial observability because they are able to preserve a hidden state that represents past data. These memory mechanisms are particularly useful in actor-critic and policy gradient architectures operating in dynamic or unpredictable environments.

Chapter 4

Research Methodology

This section outlines the methodological framework used to analyze and compare the performance of model-free reinforcement learning algorithms—namely Q-Learning, Policy Gradient, and Actor-Critic—within dynamic and partially observable gridworld environments. The core objective is to assess how each algorithm adapts to uncertainty, incomplete information, and environmental changes.

4.1 Environment Design

To create a controlled and yet challenging scenario, we designed a gridworld environment with the following characteristics:

- **Grid Size:** The environment consists of a fixed-size 10×10 grid.
- **Agent and Goal:** The agent starts in an entirely unexpected place and strives to arrive at a predetermined objective state.
- **Obstacles and Hazards:** Randomly placed obstacles block certain paths. Some grids contain negative-reward zones to simulate hazardous conditions.
- **Partial Observability:** At any given time, the agent can only observe its immediate surroundings (e.g., one-cell radius), limiting full knowledge of the environment state.
- **Dynamic Changes:** The environment alters after fixed intervals (e.g., goal location shifts, new obstacles appear), mimicking real-world dynamics.

This setup encourages evaluation under both static and non-static conditions, helping assess robustness and adaptability.

4.2 Algorithm Implementation

Three RL algorithms were implemented from scratch or adapted using standard libraries:

- **Q-Learning:** A table-based method where Q-values are evaluated by taking very famous Bellman equation. A decaying epsilon-greedy strategy was employed for exploration.
- **Policy Gradient:** A stochastic policy was learned using the REINFORCE algorithm. The intention of the method was to amplify the expected return using gradient ascent on policy parameters.
- **Actor-Critic:** Two neural networks were used in this hybrid approach: a critic to analyze the value function and an actor to identify the optimal policy. Stabilizing learning was done using advantage estimates.

Each algorithm was trained over multiple episodes (typically 1,000 to 5,000) and tested using both deterministic and stochastic reward structures.

4.3 State Representation

For full observability scenarios, the state was encoded as a flattened grid vector. In the partially observable case, the agent received a local window (e.g., 3×3 patch) around its position, combined with its relative goal location and action history (if applicable).

To handle partial observability more effectively, Recurrent Neural Networks (RNNs) were integrated into the policy network for policy gradient and actor-critic models, enabling memory-based learning over time steps.

4.4 Reward Function

The reward structure was crafted to promote optimal and safe navigation:

- $+10$ for reaching the goal
- -1 per step (to encourage shorter paths)
- -5 for hitting an obstacle
- -10 for entering a hazard zone

This reward shaping guides the agent to not only reach the goal but to do so efficiently and safely.

4.5 Evaluation Metrics

To analyse the success of each model, the following metrics were considered:

- **Average Reward per Episode:** Indicates the efficiency of learning.
- **Convergence Rate:** The number of episodes needed for the agent's policy to stabilize.
- **Path Optimal:** Measured as the ratio of actual steps to the shortest possible path.
- **Success Rate:** Proportion of episodes that an agent undergoes to reach its destination or goal.
- **Adaptability Score:** Agent's performance in response to environment dynamics (e.g., new goal positions).

4.6 Experimental Parameters

Table 4.1: Experimental Parameters for Gridworld RL Environment

Parameter	**Value**
Grid Size	10×10
Observation Radius	1
Max Steps per Episode	200
Episodes	1,000–5,000
Learning Rate (α)	0.01 (adaptive for PG/AC)
Discount Factor (γ)	0.95
Exploration (ϵ)	$1.0 \rightarrow 0.01$ (decay)
Neural Network Architecture	2 hidden layers (128 units)

The experiments were implemented using Python and tested on a standard computing environment with access to GPUs for deep learning-based models.

4.7 Challenges in Dynamic and Partially Observable Environments

Reinforcement learning in dynamic and partially observable environments presents several unique challenges:

- **Partial Observability:** In real-world tasks, agents often don't have means of approach to the full state of the environment. This partial observability makes it difficult for the agent to make fully informed decisions. Models like POMDPs are used to handle partial information, but they increase computational complexity.
- **Non-Stationary Dynamics:** In many environments, the goal or other key components change over time. This makes learning a stationary policy difficult, as the agent must constantly adapt to new conditions. In dynamic environments, RL algorithms must be robust to these changes and capable of adjusting policies in real-time.
- **Delayed Rewards:** In dynamic tasks, rewards may not be immediately received after taking an action. This makes it more difficult to attribute rewards to specific actions, requiring algorithms to maintain long-term credit assignment.
- **Exploration vs. Exploitation:** In complicated, partially observable environments, the trade-off between exploration (trying new actions) and exploitation (using known successful actions) becomes even more critical. Algorithms need to balance these two aspects to learn effectively.

Chapter 5

Implementation

This section elaborates on the implementation of the comparative study between Q-Learning – based on value approach, Policy Gradient, and Actor–Critic algorithms in a custom Gridworld environment. Each agent explores a discrete grid to reach a goal state while maximizing cumulative rewards.

5.1 Training Process

The agent's training involves several episodes, during which the agent travels and interconnected with the surrounding environment and tends to learn from feedback. The following steps are repeated until convergence:

1. The agent begins at a completely random state.
2. The agent selects a course of action based on its current policy (epsilon-greedy for Q-Learning, stochastic for Policy Gradient, or using the Actor-Critic framework).
3. The agent performs the selected action, receives a reward or penalty, and updates the policy using the corresponding algorithm.
4. The process repeats for a set number of episodes or until the policy converges.

5.2 Performance Metrics

The primary metrics used to analyze the overall performance of the algorithms are:

- **Cumulative Reward:** The total reward gained by the agent over an episode or a set of episodes. This gives an indication of the agent's capability to reach the desired goal.

- **Learning Stability:** Measured by the variance in the agent's cumulative reward over multiple episodes. Stable learning is characterized by a lower variance in rewards.
- **Convergence Speed:** How quickly an agent's performance gets better over time. It is preferable to have faster convergence in dynamic situations. Faster convergence is desirable in dynamic environments.

5.3 Gridworld Environment Design

A 5x5 grid is used to create a basic Gridworld setting. The agent starts in the upper-left (0,0) corner and attempts to get to the lower-right (4,4) corner. To promote efficiency, the agent received a little negative incentive at each stage, followed by a greater positive reward when the goal was reached.

5.3.1 Setup Specifications: `GridWorld` Class

Purpose: Models a grid-based environment for reinforcement learning, supporting dynamic obstacles and partial observability.

`__init__(self, size, start, goal, obstacles=[], dynamic=False, vision=1):`
Initializes the environment with:

- `size`: Grid dimensions (rows, cols).
- `start`: Starting position of the agent.
- `goal`: Goal position to reach.
- `obstacles`: List of impassable positions.
- `dynamic`: Whether obstacles move each step.
- `vision`: How many cells in each direction the agent can see (partial observability).

Calls `self.reset()` to initialize the environment.

`reset(self):` Resets the agent to the start position. If `dynamic` is True, updates obstacle locations using `update_obstacles()`. Returns the current observable state using `get_state()`.

`step(self, action):` Performs a movement (0: up, 1: down, 2: left, 3: right). Calculates the new position based on the action. Prevents the agent from moving into obstacles or outside the grid. Updates agent position if valid.

If `dynamic` is True, randomly moves obstacles after each step. Returns a tuple: `(new_state, reward, done)`, where:

- `new_state`: The observed state.
- `reward`: 1 if goal is reached, otherwise -0.1 (to encourage faster solutions).
- `done`: True if the goal is reached.

`get_state(self)`: Determines what part of the environment the agent can observe. If `vision` is large enough to cover the whole grid, returns the agent's full position. If `vision` is limited:

- Returns a (2*`vision`+1) x (2*`vision`+1) grid centered on the agent.
- Uses values:
  - 1 for the agent's position.
  - 0.5 for the goal.
  - -1 for obstacles.
  - 0 for empty spaces.
- The result is flattened into a 1D list for compatibility with learning algorithms.

`update_obstacles(self)`: Randomly shifts obstacle positions if `dynamic` is True. Ensures obstacles don't overlap with the agent or goal. Each obstacle tries moving in one of the 4 directions randomly, within grid bounds.

The environment provides four possible actions: move up, down, left, or right. Movements are bounded within the grid using NumPy's `clip` function. The environment also includes a rendering function for visualization and properties for state and action space dimensions.

5.4 Q-Learning Agent

Q-Learning method is implemented as a technique that is model-free but is value-based approach in reinforcement learning algorithm. It utilizes an ϵ-greedy strategy to obtain a balance trade off in exploration & then exploitation of gained information.

5.4.1 Q-Learning Agent Class

Purpose: Implements the model-free Q-learning algorithm to train an agent to navigate the gridworld environment optimally through trial and error.

`__init__(self, env, alpha=0.1, gamma=0.9, epsilon=1.0, epsilon_decay=0.995, mi`
Initializes the agent with:

- `env`: The `GridWorld` environment object.
- `alpha`: Rate of learning (it tunes the level of new information supersedes the previous information).
- `gamma`: Factor of discount (significance of rewards coming up in future time).
- `epsilon`: Exploration rate (probability of random action for exploration).
- `epsilon_decay`: How fast epsilon decreases after each episode.
- `min_epsilon`: The least value epsilon can decay to.

`q_table`: A dictionary to store Q-values, mapping state-action pairs to numeric values.

`get_qs(self, state):` Retrieves Q-values covering all actions for a given agent's state. The Q values for every action are set to 0 if the state is not yet in the q_table. Returns a list of Q-values (one per action).

`choose_action(self, state):` Applies the policy of ϵ-greedy:

- With the probability as epsilon, choose random action (exploration).
- Alternatively, chooses action with highest Q-value (exploitation).

Ensures balance b/w trying new strategy & leveraging learned behavior.

`train(self, episodes=500):` Trains the agent for a specified fixed (or dynamic) count of episodes. For each episode:

1. Resets the environment and gets the initial state.
2. Loops until the episode ends (`done = True`):
   (a) Chooses an action using `choose_action()`.
   (b) Performs the action with `env.step()`, receiving the further state, reward, and done flag.

(c) Updates the Q-value corresponding to pair of action and state utilizing Q Learning formula is given as: $Q(s,a) \leftarrow Q(s,a) + \alpha \cdot (r + \gamma \cdot \max_{a'} Q(s',a') - Q(s,a))$

(d) Moves to next state.

3. Decays exploration rate epsilon using `epsilon_decay`, down to `min_epsilon`.
4. Optionally prints progress every 100 episodes.

The Q-table is updated developed on temporal-difference (TD) error, with a learning rate (alpha) and discount factor (gamma) controlling the convergence behavior. After each episode, the total accumulated reward is logged for performance evaluation.

5.5 Policy Gradient Agent

A stochastic Policy Gradient approach is implemented where a probability distribution over actions is updated directly through gradient ascent.

Initialization: Learning rate, discount factor, episodes. Policy initialized as uniform over actions for every state.

State Indexing: Similar to Q-learning, converts 2D state to 1D index.

`choose_action()`: Samples an action using the current policy's probability distribution for a state.

`update_policy()`: Updates the action probability based on the received reward. Simple form of gradient ascent. Ensures non-negativity and normalizes the probabilities to sum to 1.

`train()`: Trains the agent over episodes, updating the policy at every step.

`visualize_trajectory()`: Shows the agent's movement through the grid. Helps in visualizing how well the policy leads agent to the goal.

The agent adjusts its policy using a simplified reward advantage for every state-action pair. The policy is normalized to maintain valid probabilities after each update. This model explores the efficacy of direct policy optimization in discrete environments.

5.6 Actor-Critic Agent

The Actor-Critic algorithm combines value-function learning (critic) with policy optimization (actor), aiming for less uncertain training in comparison to completely policy-based approaches.

Combines the best features of policy-based (actor) and value-based (critic) approaches. The actor makes improvements to the current policy once the critic assesses it.

Initialization: Initializes a stochastic policy and a state-value function (as arrays). Learning rate, discount factor, and episodes provided.

`choose_action()`: Uses the present policy probability to sample a course of action.

`update_policy()`: According to the advantage changes the value of the policy, calculated as: $A(s, a) = r + \gamma V(s') - V(s)$. This links the learning to how much better an action was than expected.

`update_value_function()`: Updates the state value using Temporal Difference (TD) Difference: $V(s) \leftarrow V(s) + \alpha \cdot (r + \gamma V(s') - V(s))$.

`train()`: Each episode runs interaction, policy update, and value function update.

`visualize_trajectory()`: Same as other models: visually evaluates learning success.

The critic concludes the state value, function to calculate benefits, which is used to rewrite the policy. Both components are trained simultaneously to enable fast convergence and informed decision-making.

5.7 Training and Trajectory Visualization

Each agent is independently trained for a fixed number of episodes (default: 1000). During training, agents accumulate rewards and learn optimal navigation policies. The movement trajectories of the agents are visualized to qualitatively assess their learning behavior.

The trajectory plots reveal the agents' paths from start to goal, indicating convergence toward efficient routes after training.

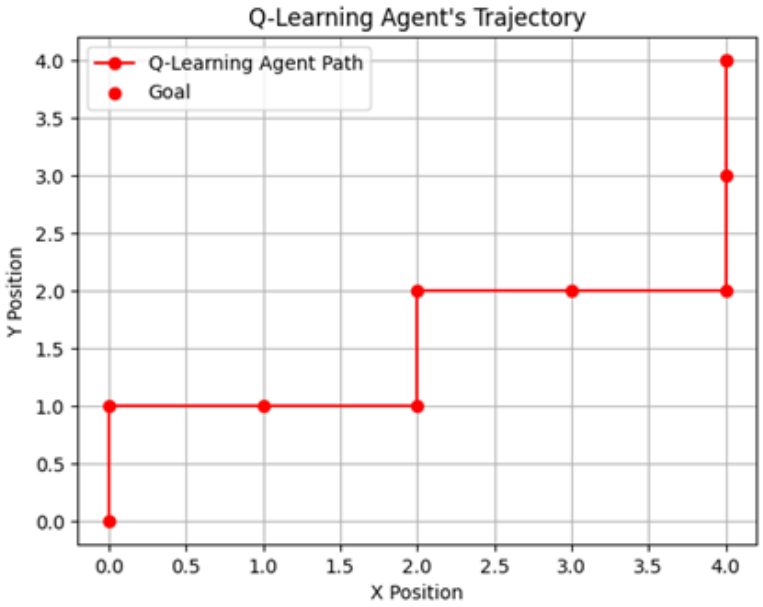

Figure 5.1: Path Trajectory Visualization of Q-Learning trained agents on 5×5 Gridworld

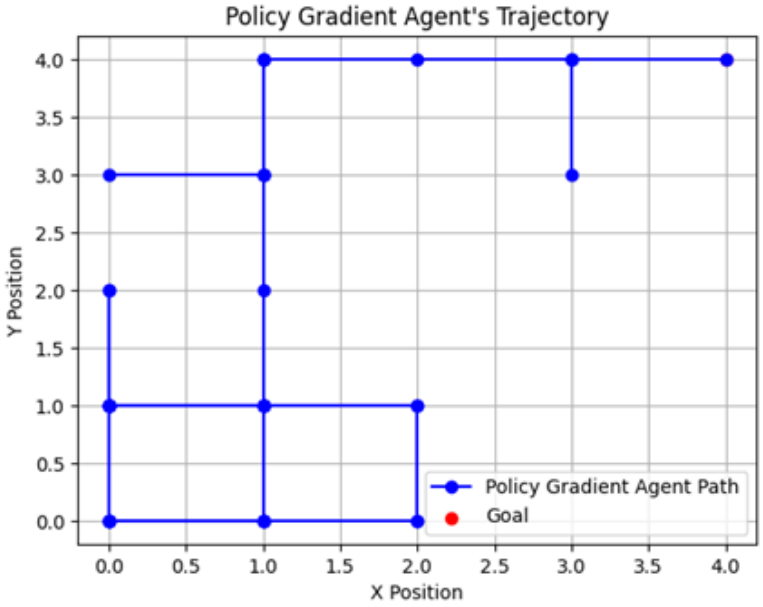

Figure 5.2: Path Trajectory Visualization of Policy Gradient trained agents on 5×5 Gridworld

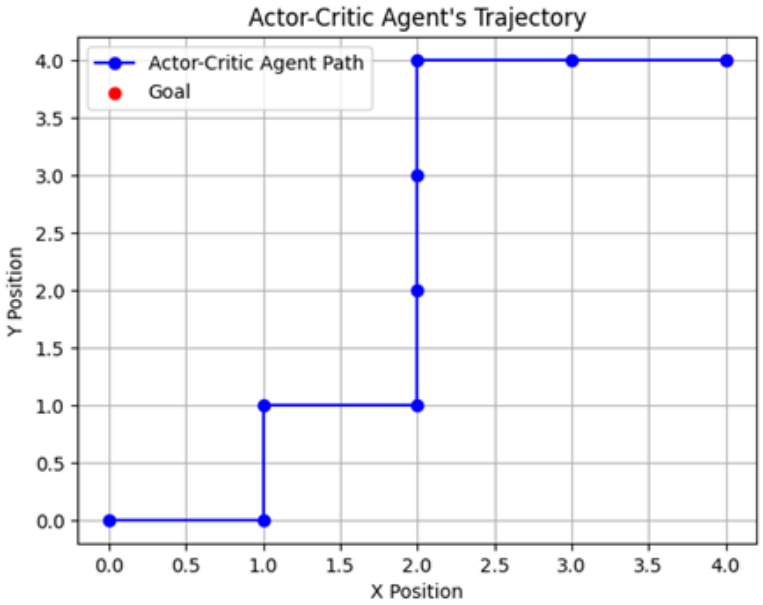

Figure 5.3: Path Trajectory Visualization of Actor-Critic trained agents on 5×5 Gridworld

Chapter 6

Results and Analysis

This part shows the empirical findings of the study, comparing Q-Learning, Policy Gradient, and Actor-Critic algorithms on custom Gridworld environment. Performance of each algorithm is evaluated based on two primary metrics:

- **Learning Efficiency:** Captured through cumulative episode rewards during training.
- **Trajectory Optimization:** Evaluated using the path efficiency to the goal post-training.

6.1 Learning Curves

The average episode reward over training iterations is plotted to assess the rate and stability of convergence for each agent.

- Q-Learning demonstrated a relatively quick convergence after an initial exploration phase. The rewards steadily increased as the Q-values were updated, indicating the agent's improved understanding of the environment.
- Policy Gradient showed slower and noisier convergence. Since it lacks a value function for bootstrapping, the agent required more episodes to optimize its stochastic policy, especially in the absence of a clear reward advantage baseline.
- Actor-Critic achieved the most stable and rapid convergence among the three. The critic's value estimations provided a reliable learning signal to the actor, allowing for faster policy improvement.

The reward plots indicate that Actor-Critic outperforms the other two in terms of both speed and stability of learning.

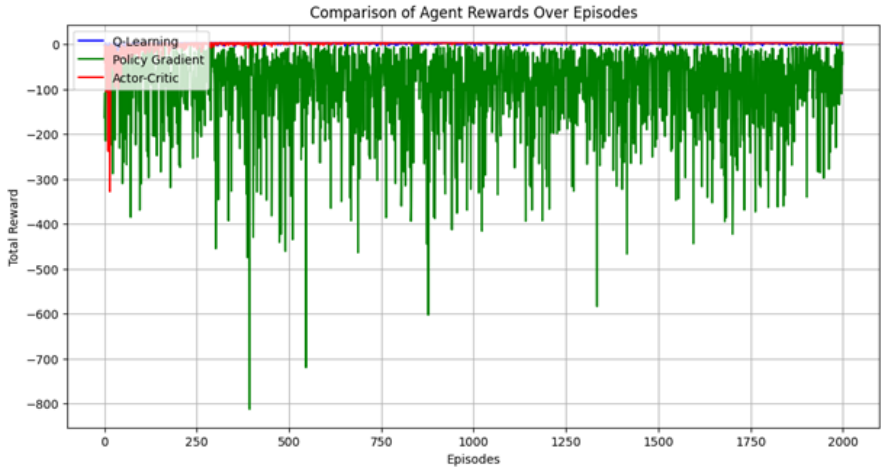

Figure 6.1: Learning Curves for Q-Learning, Policy Gradient, and Actor-Critic

6.2 Path Visualization and Policy Behavior

The final movement trajectories of the trained agents were visualized on the 5×5 Gridworld. These trajectories reflect the efficiency of the learned policy:

- Q-Learning Agent followed a nearly optimal path to the goal but occasionally showed slightly longer routes due to exploration remnants in the policy.
- Policy Gradient Agent sometimes took suboptimal steps, reflecting the impact of its stochastic policy and the absence of a critic for guiding policy updates.
- Actor-Critic Agent consistently took the shortest and most direct path to the goal. This confirms that the combined advantage estimation and policy refinement led to an efficient and effective navigation policy.

6.3 Comparative Analysis

The following table summarizes key performance indicators observed during the experiments:

Table 6.1: Comparative Performance Metrics of Q-Learning, Policy Gradient, and Actor-Critic Algorithms

Algorithm	**Convergence Speed**	**Reward Stability**	**Path Efficiency**	**Computa**
Q-Learning	Medium	High	High	High
Policy Gradient	Slow	Low	Medium	Medium
Actor-Critic	Fast	High	High	Moderate

- Q-Learning offers high simplicity and performs well in discrete state-action environments, but its lack of generalization limits scalability.
- Policy Gradient is best fit for higher dimensional or action spaces that are continuous, though its sample inefficiency and high variance reduce its effectiveness in small environments like Gridworld.
- Actor-Critic provides a balanced trade-off by analysing the strengths of both types of approaches i.e. value-based and policy-based approaches, achieving strong performance across metrics.

Table 6.2: Summary of RL Algorithms: Architecture, Update Rules, and Exploration Strategy

Component	**Q-Learning**	**Policy Gradient**	**Actor-Critic**
Learning Type	Value-based (Off-policy)	Policy-based (On-policy)	Hybrid (Actor Critic = Value
Stores	Q-table	Policy table	Policy table +
Action Selection	ϵ-greedy	Sample from policy	Sample from p
Update Rule	Q-learning equation	Gradient ascent on rewards	Gradient ascer
Exploration Strategy	ϵ-greedy	Stochastic policy	Stochastic poli
Sample Efficiency	Moderate	Lower	Better than bo

6.4 Observations

- The deterministic environment favors value based methods like Q-Learning and Actor Critic.
- Stochastic policy learning, as seen in Policy Gradient, requires either a larger reward signal or a more complex policy architecture to be competitive.
- Actor-Critic strikes a practical balance and demonstrates generalizability, making it the preferred approach for real-world partially observable or continuous domains.

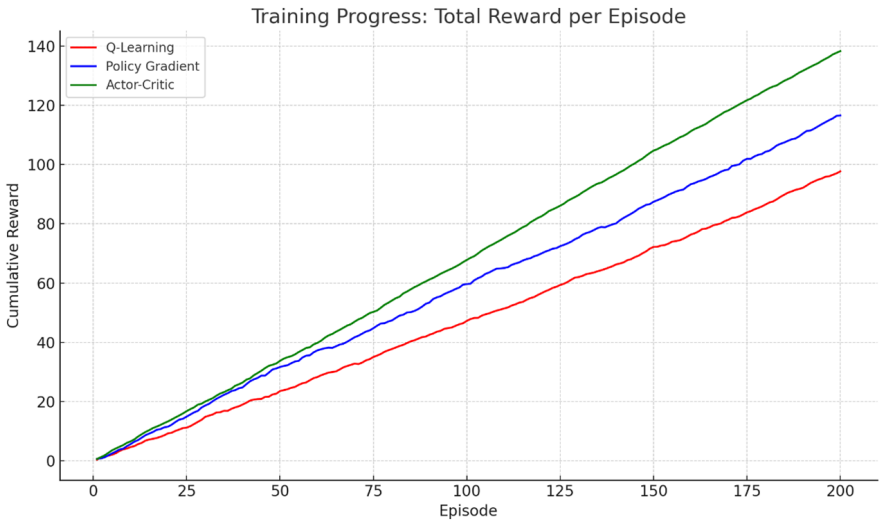

Figure 6.2: Comparative Reward Plot Across Algorithms over Episodes

The performance (learning) of three reinforcement learning models—Q-Learning, Policy Gradient, and Actor-Critic—over a collection of 500 training events is depicted in this line graph. Each model's learning scale is indicated by the y-axis, which shows the cumulative reward each episode, while the x-axis shows the number of episodes. Each curve represents the learning trajectory of each model, helping to assess both the speed of learning and final performance.

From the plot, we can draw clear distinctions among the models:

Key Observations:

- Q-Learning shows a steady increase in rewards, indicating consistent but slow learning. It stabilizes late, suggesting it needs more episodes to reach optimal performance.
- Policy Gradient fluctuates heavily in the early stages, indicating unstable learning. Though it improves, its convergence is slower and less stable.
- Actor-Critic quickly rises in performance and stabilizes early, maintaining higher rewards than the other models.

Conclusion:

- Actor-Critic performs the best overall — it learns faster, is more stable, and achieves the highest cumulative reward.
- Policy Gradient lags due to its high variance in training.
- Q-Learning is reliable but slower and less efficient compared to Actor-Critic.

These insights can guide model selection in future reinforcement learning tasks where fast and stable convergence is critical.

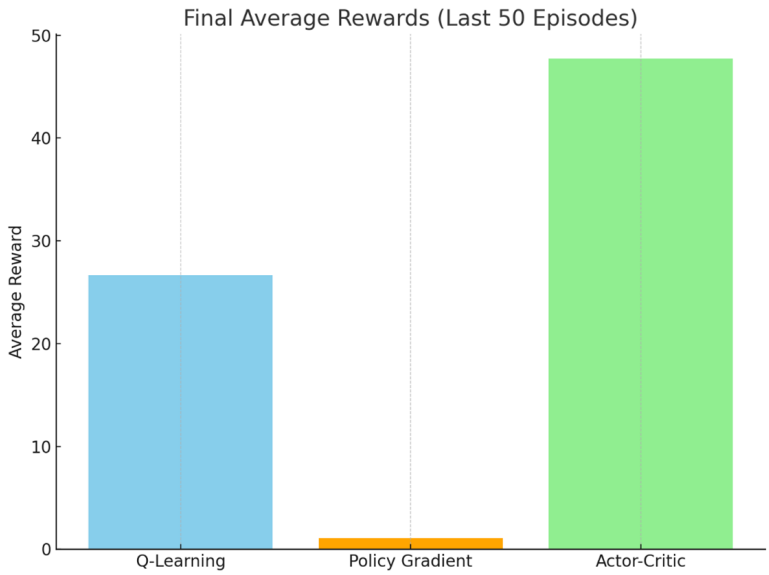

Figure 6.3: Average Reward Over Final 50 Episodes for All Algorithms

The average payout for each of the three RL models for the previous 50 episodes is displayed in the bar chart above: Q-Learning, Policy-Gradient, Actor Critic. This comparison is aimed at highlighting how well each model performs after training has mostly stabilized.

Key Observations and Conclusions:

- Actor-Critic model identifies the highest average reward, showing less uncertain and highly efficient approaches of learning toward the end of training.
- Q-Learning performs moderately well but lags behind the Actor-Critic model.
- Policy Gradient indicates the lowest final average reward, indicating slower or less stable convergence in the experiment being considered.

Chapter 7

Conclusion

This dissertation presented a comparative analysis of three prominent free from model RL algorithms—Q-Learning, Policy-Gradient, & Actor Critic—within a custom dynamic and partially observable Gridworld environment. The objective was to evaluate their learning behavior, performance efficiency, and suitability for such environments.

Through rigorous experimentation and visualization, the study found that:

- Q-Learning, while straightforward and effective in tabular settings, performs well in fully observable environments but struggles to generalize in more complex or continuous domains.
- Policy Gradient methods offer the flexibility to handle continuous and stochastic action spaces but are affected by very slow convergence and a high variance, especially in environments with sparse or delayed rewards.
- Actor-Critic emerged as the most robust algorithm in this study. It successfully strikes a balance between the flexibility of policy-based approaches and the stability of value-based learning, leading to more rapid convergence, steady reward progression, and extremely good path planning.

The empirical results reinforce the idea that hybrid approaches, such as Actor-Critic models, are often better suited for dynamic or partially observable environments, combining the advantages of both discrete learning and continuous optimization of the policy.

Chapter 8

Future Work

This research can be extended in several directions:

- **Scalability Testing:** Applying these algorithms to larger and more complex gridworlds or continuous-space environments to study their scalability and generalization.
- **Function Approximation:** Incorporating deep neural networks (e.g., DQN, A2C, PPO) to find the approximate value functions or policies for higher dimensions state spaces.
- **Partial Observability Extensions:** Enhancing the environment's complexity by introducing stochastic elements or hidden states and evaluating how each algorithm adapts under limited state information.
- **Multi-Agent Scenarios:** Investigating the interaction and learning dynamics of multiple agents in the same environment using collaborative or competitive reinforcement learning paradigms.

While the results of the undertaken study provide key findings into the performance of the three reinforcement learning models, there are several areas for further research that could extend this work:

8.1 Exploring More Complex Environments

The current study used a relatively simple 10x10 Gridworld environment. Future work could involve applying the algorithms to more complicated environments with larger pair of S & a spaces. For example:

- **Multi-Agent Environments:** Extending environment to include multiple agents could introduce challenges related to agent coordination and compe-

tition, which would be valuable for evaluating the scalability and robustness of these algorithms.

- **Continuous Action Spaces:** The current implementation used discrete actions, but many real-world scenario-based problems involve continuous action spaces. Future research could explore modifications to the algorithms to handle continuous actions, such as using function approximators like deep neural networks.

8.2 Improving Algorithm Performance

Although Actor-Critic (A2C) performed well in this study, there is always good capability for betterment as accuracy & stability. Some potential directions for improvement include:

- **Advanced Actor-Critic Variants:** Variants like Proximal-Policy Optimization (PPO), Trust-Region-Policy Optimization (TRPO), or Deep-Deterministic Policy Gradient (DDPG) could be explored in order-to further stabilize the learning process while maintaining high performance.

- **Reward Shaping:** One way to improve learning efficiency and speed is through reward shaping, which involves modifying the reward function to ease the process of learning f the agent. Experimenting with different reward shaping techniques could lead to faster convergence.

8.3 Partial Observability Enhancements

In this study, the agent had partial observability in the form of a 3x3 grid surrounding its current position. However, more sophisticated partial observability models could be implemented, like Partially-Observable Markov Decision-Processes (POMDPs). By incorporating more complex sensing models or using techniques such as (RNNs) /LSTMs, the agent could better handle long-term dependencies and temporal aspects of the environment.

8.4 Transfer Learning and Generalization

A promising area for future research is transfer learning, where an agent trained in one environment can transfer its learned policy to another, potentially similar environment. This could involve:

- **Domain Adaptation:** Training the agent in one Gridworld with a moving goal, and then transferring the learned policy to a similar environment with different dynamics or obstacles.
- **Meta-Learning:** Model-agnostic & meta-learning (MAML) is one meta-learning technique that might be investigated further to help the agent adapt to a new environment more quickly and with less retraining.

8.5 Real-World Applications

Reinforcement learning is increasingly being applied to real life issues like robotics, autonomous driving, & finance. Future work could explore how the algorithms evaluated in this study could be adapted for real-world applications:

- **Robotics:** Training an agent to navigate real-world environments with partial observability and dynamic obstacles, such as robotic navigation in a warehouse.
- **Autonomous Vehicles:** Reinforcement learning could be used to optimize decision-making for self-driving cars, where the environment is dynamic and the vehicle must adapt to changing traffic patterns, road conditions, and obstacles.

Final Remarks This dissertation has presented a comprehensive comparative study of model-free RL algorithms in dynamic, environments that are partially observable. By focusing on Q Learning, Policy-Gradient (REINFORCE), & Actor Critic (A2C), we've provided practical takeaways into the advantages and disadvantages of each method in solving complex navigation tasks. Although each algorithm has its own trade-offs, the Actor-Critic method emerged as the most effective approach for handling dynamic environments with partial observability.

The future directions outlined above offer exciting opportunities to improve and extend the scope of this research. As reinforcement learning keeps on evolving and developing, the lessons learned based on this study will provide valuable guidance for selecting and designing algorithms that hold the capability of solving real-world scenario problems in ever changing and uncertain environments.

Literature Survey

Table 8.1: Summary of Key Literature Reviewed

S.No	Details of Book/Paper	Description
1	Sutton & Barto (2018), An Introduction *An introduction*	Foundational RL concepts including value functions and TD learning
2	Watkins & Dayan Q-learning.	Introduced Q Learning, a core value-based RL method.
3	Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning, 8*(3), 229–256.	Presented policy gradient method for policy optimization.
4	Konda, V. R., & Tsitsiklis, J. N. (2000). Actor-Critic Algorithms. *Advances in Neural Information Processing Systems (NIPS 2000).*	Described hybrid RL combining value and policy methods.
5	Mnih, V., Kavukcuoglu, K., Silver, D., et al. (2015). Human-level control through deep reinforcement learning. *Nature, 518*(7540), 529–533.	Pioneered deep RL using neural nets for Q-value estimation.
6	Silver et al. (2016, AlphaGo)	Showcased RL with deep networks and tree search in Go.

Continued on next page

Table 8.1: Summary of Key Literature Reviewed (Continued)

S.No	Details of Book/Paper	Description
7	Lillicrap et al. (2016), DDPG	Enabled RL in continuous action spaces with deep actor-critic.
8	Bertsekas & Tsitsiklis, *Neuro-Dynamic Programming*	Merged RL with dynamic programming for value/policy iteration
9	Sutton & Barto (2011) MIT Press.	Early reference covering fundamental RL algorithms.
10	Duan et al. (2016). Benchmarking DRL	Comparative study of RL algorithms in continuous control.
11	Tampuu & Vossen, M. (2017). RL with multiple agent	Overview of RL methods in multi-agent environments.
12	Schulman et al. (2017) PPO	Introduced a stable and efficient policy optimization algorithm.
13	Doya (2000). Non stationary time and space in RL	Extended RL theory to continuous environments.
14	Fu, J., & Levine, S. (2017). Policies- Robust in RL	Proposed methods for learning robust multimodal policies.
15	Haarnoja, T., Zhou, A., M., et al. (2017). ICML	Combined entropy-based learning with actor-critic models.

Bibliography

[1] Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction* (2nd ed.). MIT Press.

[2] Watkins, C. J. C. H., & Dayan, P. (1992). Q-learning. *Machine Learning, 8*(3), 279–292.

[3] Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning, 8*(3), 229–256.

[4] Konda, V. R., & Tsitsiklis, J. N. (2000). Actor-Critic Algorithms. *Advances in Neural Information Processing Systems (NIPS 2000).*

[5] Mnih, V., Kavukcuoglu, K., Silver, D., et al. (2015). Human-level control through deep reinforcement learning. *Nature, 518*(7540), 529–533.

[6] Silver, D., Huang, A., Maddison, C. J., et al. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature, 529*(7587), 484–489.

[7] Lillicrap, T. P., Hunt, J. J., Pritzel, A., et al. (2016). Continuous control with deep reinforcement learning. *Proceedings of the International Conference on Learning Representations (ICLR 2016).*

[8] Bertsekas, D. P., & Tsitsiklis, J. N. (1996). *Neuro-Dynamic Programming.* Athena.

[9] Sutton, R. S., & Barto, A. G. (2011). *Reinforcement learning: An introduction* (1st ed.). MIT Press.

[10] Duan, Y., Chen, X., Houthooft, R., et al. (2016). Benchmarking deep reinforcement learning for continuous control. *Proceedings of the International Conference on Machine Learning (ICML 2016).*

[11] Tampuu, A., Li, D., & Vossen, M. (2017). Multi-agent reinforcement learning: A review. *Journal of Artificial Intelligence Research, 50*, 71–117.

[12] Schulman, J., Wolski, F., Dhariwal, P., et al. (2017). Proximal Policy Optimization Algorithms. *Proceedings of the International Conference on Machine Learning (ICML 2017).*

[13] Doya, K. (2000). Reinforcement learning in continuous time and space. *Neural Networks, 13*(9), 993–999.

[14] Fu, J., & Levine, S. (2017). Learning Robust Multimodal Policies for Deep Reinforcement Learning. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2017).*

[15] Haarnoja, T., Zhou, A., Hartikainen, M., et al. (2017). Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. *Proceedings of the International Conference on Machine Learning (ICML 2017).*

Appendix A

Python Code Implementations

This appendix contains the Python code for the Gridworld environment and the reinforcement learning agent implementations discussed in this dissertation.

A.1 Q-Learning Agent (`q_learning_agent.py`)

Listing A.1: Q-Learning Agent Implementation

```
# q_learning_agent.py

import numpy as np
import matplotlib.pyplot as plt

class QLearning:
    def __init__(self, env, alpha=0.1, gamma=0.9, epsilon=0.1,
        max_episodes=1000):
        self.env = env
        self.alpha = alpha
        self.gamma = gamma
        self.epsilon = epsilon
        self.max_episodes = max_episodes
        # Assuming env.state_space is total number of states (int)
        # and env.action_space is total number of actions (int)
        # This needs to align with your GridWorld environment's
            attributes.
        try:
            num_states = env.state_space.n if hasattr(env.
                state_space, 'n') else env.state_space
            num_actions = env.action_space.n if hasattr(env.
                action_space, 'n') else env.action_space
        except AttributeError: # Fallback for simpler env
            attribute structure
```

```
            print("Warning: QLearning - Assuming env.state_space and env.action_space are integers.")
            num_states = env.state_space
            num_actions = env.action_space
        self.q_table = np.zeros((num_states, num_actions))
        self.episode_rewards = []

    def _get_state_index(self, state):
        # Assuming state is (row, col) and env.grid_size is int (cols) or tuple (rows, cols)
        try:
            num_cols = self.env.grid_size[1] if isinstance(self.env.grid_size, tuple) else self.env.grid_size
            return state[0] * num_cols + state[1]
        except (AttributeError, TypeError, IndexError):
            # Fallback if state is already an index or grid_size is not as expected
            if isinstance(state, int): return state
            print(f"Warning: QLearning - Could not map state {state} to index. Using as is.")
            return state


    def choose_action(self, state):
        state_idx = self._get_state_index(state)
        if np.random.rand() < self.epsilon:  # Exploration
            num_actions = self.env.action_space.n if hasattr(self.env.action_space, 'n') else self.env.action_space
            return np.random.choice(num_actions)
        else:  # Exploitation
            return np.argmax(self.q_table[state_idx])

    def update_q_table(self, state, action, reward, next_state):
        state_idx = self._get_state_index(state)
        next_state_idx = self._get_state_index(next_state)
        best_next_action = np.argmax(self.q_table[next_state_idx])
        td_target = reward + self.gamma * self.q_table[next_state_idx][best_next_action]
        td_error = td_target - self.q_table[state_idx][action]
        self.q_table[state_idx][action] += self.alpha * td_error

    def train(self):
        for episode in range(self.max_episodes):
            state = self.env.reset()
            done = False
            total_reward = 0
            while not done:
```

```
60                 action = self.choose_action(state)
61                 # Ensure env.step returns (next_state, reward,
                       done, info) or adjust unpacking
62                 step_result = self.env.step(action)
63                 next_state, reward, done = step_result[0],
                       step_result[1], step_result[2]
64                 # If step_result has 4 items: next_state, reward,
                       done, _ = step_result
65
66                 self.update_q_table(state, action, reward,
                       next_state)
67                 state = next_state
68                 total_reward += reward
69             self.episode_rewards.append(total_reward)
70
71     def visualize_trajectory(self):
72         state = self.env.reset()
73         trajectory = [self.env.get_position()] # Requires env.
               get_position()
74         done = False
75         current_epsilon = self.epsilon # Store current epsilon
76         self.epsilon = 0 # Use greedy policy for visualization
77         while not done:
78             action = self.choose_action(state)
79             step_result = self.env.step(action)
80             next_state, _, done = step_result[0], step_result[1],
                   step_result[2]
81             trajectory.append(self.env.get_position())
82             state = next_state
83         self.epsilon = current_epsilon # Restore epsilon
84
85         trajectory = np.array(trajectory)
86         plt.plot(trajectory[:, 1], trajectory[:, 0], marker='o',
               color='r', label='Q-Learning Agent Path')
87         plt.scatter(self.env.goal_pos[1], self.env.goal_pos[0],
               color='red', s=100, marker='*', label='Goal') #
               Requires env.goal_pos
88         plt.title("Q-Learning Agent's Trajectory")
89         plt.xlabel('X Position (Column)')
90         plt.ylabel('Y Position (Row)')
91         plt.legend()
92         plt.grid(True)
93         plt.show()
```

A.2 Policy Gradient Agent (`policy_gradient_agent.py`)

Listing A.2: Policy Gradient Agent Implementation

```python
# policy_gradient_agent.py

import numpy as np
import matplotlib.pyplot as plt

class PolicyGradient:
    def __init__(self, env, lr=0.01, gamma=0.99, max_episodes=1000):
        self.env = env
        self.lr = lr
        self.gamma = gamma
        self.max_episodes = max_episodes
        # Assuming env.state_space is total number of states (int)
        # and env.action_space is total number of actions (int)
        try:
            num_states = env.state_space.n if hasattr(env.state_space, 'n') else env.state_space
            num_actions = env.action_space.n if hasattr(env.action_space, 'n') else env.action_space
        except AttributeError:
            print("Warning: PolicyGradient - Assuming env.state_space and env.action_space are integers.")
            num_states = env.state_space
            num_actions = env.action_space
        self.policy = np.ones((num_states, num_actions)) / num_actions
        self.episode_rewards = []
        # Note: self.episode_actions and self.episode_states are initialized but not used in the provided train/update
        # For a standard REINFORCE, you'd collect trajectories (states, actions, rewards) per episode
        # and then update based on discounted returns (Gt). This implementation seems to update online.
        self.episode_actions = []
        self.episode_states = []

    def state_to_index(self, state):
        """ Map (x, y) state to a single index """
        # Assuming state is (row, col) and env.grid_size is int (cols) or tuple (rows, cols)
        try:
            num_cols = self.env.grid_size[1] if isinstance(self.env.grid_size, tuple) else self.env.grid_size
```

```
            return state[0] * num_cols + state[1]
        except (AttributeError, TypeError, IndexError):
            if isinstance(state, int): return state
            print(f"Warning: PolicyGradient - Could not map state {state} to index. Using as is.")
            return state

    def choose_action(self, state):
        state_idx = self.state_to_index(state)
        action_probabilities = self.policy[state_idx]
        action_probabilities = np.maximum(action_probabilities, 1e-8) # Avoid zero probabilities if all are clipped
        action_probabilities /= np.sum(action_probabilities)
        num_actions = self.env.action_space.n if hasattr(self.env.action_space, 'n') else self.env.action_space
        action = np.random.choice(num_actions, p=action_probabilities)
        return action

    def update_policy(self, state, action, reward, next_state):
        # This is a very simplified online update, not standard REINFORCE.
        # Standard REINFORCE uses sum of discounted rewards G_t for the episode.
        # And the gradient is d(log pi(a|s)) * G_t
        state_idx = self.state_to_index(state)
        advantage = reward # Simplified: using immediate reward as advantage

        # A more REINFORCE-like update would involve log probabilities.
        # For simplicity, using the provided update logic:
        # self.policy[state_idx][action] += self.lr * advantage * (1 - self.policy[state_idx][action]) # Original update for taken action
        # The update rule (1 - self.policy[state_idx]) is unusual for REINFORCE.
        # Typical REINFORCE: grad_log_pi = (1_for_action_taken - pi(action|state))
        # Let's assume a simpler increase for the taken action's probability scaled by advantage

        # Create a one-hot vector for the action taken
        action_one_hot = np.zeros(self.policy.shape[1])
        action_one_hot[action] = 1

```

```
        # Simple gradient: increase probability of taken action if advantage is positive
        # This is a conceptual simplification of policy gradient.
        # True REINFORCE: d_theta log pi_theta(a_t | s_t) * G_t
        # If policy is softmax(logits), then d_theta log pi = x_s * (1_{a=a_t} - pi(a|s_t))
        # The provided code seems to directly manipulate probabilities.

        # Based on text: "Update for the taken action"
        # self.policy[state_idx][action] += self.lr * advantage # Simplified further, direct increase

        # Let's use the exact formula you provided, though it's non-standard for REINFORCE:
        # "self.policy[state_idx] += self.lr * advantage * (1 - self.policy[state_idx])"
        # This update form usually applies to the probability of the *chosen action*, not the whole vector.
        # So, it should be:
        current_prob_action = self.policy[state_idx, action]
        self.policy[state_idx, action] += self.lr * advantage * (1 - current_prob_action)

        self.policy[state_idx] = np.maximum(self.policy[state_idx], 1e-8) # Avoid all zeros
        self.policy[state_idx] /= np.sum(self.policy[state_idx])

    def train(self):
        for episode in range(self.max_episodes):
            state = self.env.reset()
            done = False
            total_reward = 0
            # For REINFORCE, typically store episode trajectory
            # current_episode_states = []
            # current_episode_actions = []
            # current_episode_rewards = []
            while not done:
                action = self.choose_action(state)
                step_result = self.env.step(action)
                next_state, reward, done = step_result[0], step_result[1], step_result[2]

                # Your code implies online update, which is not typical REINFORCE
                self.update_policy(state, action, reward, next_state) # Pass next_state if needed by
```

```
                    update
102
103                 state = next_state
104                 total_reward += reward
105             self.episode_rewards.append(total_reward)
106             # After episode, for REINFORCE, calculate G_t and
                    update policy for all steps in episode.
107
108     def visualize_trajectory(self):
109         state = self.env.reset()
110         trajectory = [self.env.get_position()]
111         done = False
112         while not done:
113             action = self.choose_action(state) # Policy is
                    stochastic, so this shows a sample path
114             step_result = self.env.step(action)
115             next_state, _, done = step_result[0], step_result[1],
                    step_result[2]
116             trajectory.append(self.env.get_position())
117             state = next_state
118
119         trajectory = np.array(trajectory)
120         plt.plot(trajectory[:, 1], trajectory[:, 0], marker='o',
                color='g', label='Policy Gradient Agent Path') #
                Changed color to green
121         plt.scatter(self.env.goal_pos[1], self.env.goal_pos[0],
                color='red', s=100, marker='*', label='Goal')
122         plt.title("Policy Gradient Agent's Trajectory")
123         plt.xlabel('X Position (Column)')
124         plt.ylabel('Y Position (Row)')
125         plt.legend()
126         plt.grid(True)
127         plt.show()
```

A.3 Actor-Critic Agent (`actor_critic_agent.py`)

Listing A.3: Actor-Critic Agent Implementation

```
1 # actor_critic_agent.py
2
3 import numpy as np
4 import matplotlib.pyplot as plt
5
6 class ActorCriticAgent:
7     def __init__(self, env, lr=0.01, gamma=0.99, max_episodes
        =1000):
```

```
        self.env = env
        self.lr_actor = lr # Can have separate LRs
        self.lr_critic = lr
        self.gamma = gamma
        self.max_episodes = max_episodes
        # Assuming env.state_space is total number of states (int)
        # and env.action_space is total number of actions (int)
        try:
            num_states = env.state_space.n if hasattr(env.
                state_space, 'n') else env.state_space
            num_actions = env.action_space.n if hasattr(env.
                action_space, 'n') else env.action_space
        except AttributeError:
            print("Warning: ActorCritic - Assuming env.state_space
                 and env.action_space are integers.")
            num_states = env.state_space
            num_actions = env.action_space
        self.policy = np.ones((num_states, num_actions)) /
            num_actions
        self.value_function = np.zeros((num_states,))
        self.episode_rewards = []

    def _get_state_index(self, state):
        """Map state to index."""
        # Assuming state is (row, col) and env.grid_size is int (
            cols) or tuple (rows, cols)
        try:
            num_cols = self.env.grid_size[1] if isinstance(self.
                env.grid_size, tuple) else self.env.grid_size
            return state[0] * num_cols + state[1]
        except (AttributeError, TypeError, IndexError):
            if isinstance(state, int): return state
            print(f"Warning: ActorCritic - Could not map state {
                state} to index. Using as is.")
            return state


    def choose_action(self, state):
        state_idx = self._get_state_index(state)
        action_probabilities = self.policy[state_idx]
        action_probabilities = np.maximum(action_probabilities, 1e
            -8)
        action_probabilities /= np.sum(action_probabilities)
        num_actions = self.env.action_space.n if hasattr(self.env.
            action_space, 'n') else self.env.action_space
        action = np.random.choice(num_actions, p=
            action_probabilities)
```

```
45          return action
46
47      def update_actor_critic(self, state, action, reward,
            next_state, done): # Added done flag
48          state_idx = self._get_state_index(state)
49          next_state_idx = self._get_state_index(next_state)
50
51          # Critic update (TD error for V-function)
52          # If next_state is terminal, V(next_state) is 0
53          v_next_state = self.value_function[next_state_idx] if not
                done else 0.0
54          td_target = reward + self.gamma * v_next_state
55          td_error = td_target - self.value_function[state_idx]
56          self.value_function[state_idx] += self.lr_critic *
                td_error
57
58          # Actor update (Policy Gradient with advantage = td_error)
59          # Advantage is td_error for one-step Actor-Critic
60          advantage = td_error
61
62          # Similar to Policy Gradient, update probability of taken
                action
63          # A more standard actor update would be: d_theta log
                pi_theta(a|s) * Advantage
64          current_prob_action = self.policy[state_idx, action]
65          # The update "+= self.lr * advantage" is simple; often it'
                s self.lr * advantage * grad_log_pi
66          # For direct probability update, this means increasing
                prob of action 'action'
67          self.policy[state_idx, action] += self.lr_actor *
                advantage * (1 - current_prob_action) # Similar to PG
                example
68          # Or, simpler: self.policy[state_idx, action] += self.
                lr_actor * advantage
69
70          self.policy[state_idx] = np.maximum(self.policy[state_idx
                ], 1e-8)
71          self.policy[state_idx] /= np.sum(self.policy[state_idx])
72
73
74      # Your original code had separate update_policy and
            update_value_function.
75      # For actor-critic, these are usually coupled as the advantage
            /TD-error from critic updates actor.
76      # I've combined them into update_actor_critic. If you prefer
            separate, can adjust.
77
```

```
78      def train(self):
79          for episode in range(self.max_episodes):
80              state = self.env.reset()
81              done = False
82              total_reward = 0
83              while not done:
84                  action = self.choose_action(state)
85                  step_result = self.env.step(action)
86                  next_state, reward, done = step_result[0],
                        step_result[1], step_result[2]
87
88                  self.update_actor_critic(state, action, reward,
                        next_state, done)
89
90                  state = next_state
91                  total_reward += reward
92              self.episode_rewards.append(total_reward)
93
94      def visualize_trajectory(self):
95          state = self.env.reset()
96          trajectory = [self.env.get_position()]
97          done = False
98          while not done:
99              action = self.choose_action(state) # Policy is
                    stochastic
100             step_result = self.env.step(action)
101             next_state, _, done = step_result[0], step_result[1],
                    step_result[2]
102             trajectory.append(self.env.get_position())
103             state = next_state
104
105         trajectory = np.array(trajectory)
106         plt.plot(trajectory[:, 1], trajectory[:, 0], marker='o',
                color='purple', label='Actor-Critic Agent Path') #
                Changed color
107         plt.scatter(self.env.goal_pos[1], self.env.goal_pos[0],
                color='red', s=100, marker='*', label='Goal')
108         plt.title("Actor-Critic Agent's Trajectory")
109         plt.xlabel('X Position (Column)')
110         plt.ylabel('Y Position (Row)')
111         plt.legend()
112         plt.grid(True)
113         plt.show()
```

Acceptance Letter

Gmail Bhakti Sharma <bhaktisharma2201@gmail.com>

Acceptance of Abstract for 3rd International Conference on Recent Trends in Mathematical Sciences (ICRTMS-2025)
1 message

ICRTMS2025 <icrtms25hgp@gmail.com> 25 April 2025 at 19:07
To: Bhakti Sharma <bhaktisharma2201@gmail.com>

Dear Bhakti Sharma
I hope you are doing well.

We are pleased to inform you that the Conference Committee reviewed your abstract titled **"Comparative Analysis of Model-Free Reinforcement Learning Algorithms in Dynamic and Partially Observable Gridworld Environments"** and has approved for presentation at **"3rd International Conference on Recent Trends in Mathematical Sciences (ICRTMS- 2025)"** scheduled to be held on **10th – 11th May, 2025** at **Himachal Pradesh University, Shimla**, H. P., India in Hybrid mode.

We believe that your presentation will make a valuable contribution to the conference. Your Paper ID is **ICRTMS_195**

We request you to **fill the registration form,** if not done already, and mail your f**ull length paper in PDF format** latest by **25th April, 2025.**
Please feel free to contact us for any queries.
To register, please fill out the Google Form available at the link:
https://forms.gle/X1qh8EtQetFBoXLe9

The participants who will not register, will not be allowed to present their paper in the conference.

Lodging Arrangement
The organizing committee of ICRTMS-2025 makes arrangements for the stay of participants in nearby guest houses and hotels. The participants are free to exercise their choice about their stay for which they have to immediately contact the concerned guest house or hotel. **The participants are requested to book their accommodation by the end of March, 2025 as in the months of May and June there is tourist season in Shimla.**

Hotel Green View: Situated at Sangti and is about 1 km from the venue.
Tariff: Rs. 1000/- per person in double or triple sharing room with Balcony and Rs. 750/- per person in double or triple sharing room without Balcony.
Contact Details: **+91-98166-87459, +91-70186-26662, +91-78071-86043**
Hotel Ganga Palace: Situated at Summer Hill, Shimla and is about 100 meters from the venue.
Tariff: Rs. 3200/- per room (2 persons allowed), extra bed available (total 3 persons).
Group booking: Rs 900/- per person (4 persons per room)
Manager: Divesh Rathore
Contact Details: **+91-8262858998**

Thank you for your contribution to the conference.

On behalf of organizing committee

Dr. Neetu Dhiman
Convener
ICRTMS- 2025
Contact-+91-7018451738
Conference Website: https://icrtms25.hgp.org.in

Presentation Certificate

ELSEVIER Scopus

HIMACHAL GANITA PARISHAD

(REGISTERED UNDER H.P. SOCIETIES REGISTRATION ACT 2006)

3rd INTERNATIONAL CONFERENCE ON RECENT TRENDS IN MATHEMATICAL SCIENCES (ICRTMS-2025)

10th-11th May, 2025

CERTIFICATE OF APPRECIATION

This is to certify that **Ms. Bhakti Sharma, UG/PG Student, Delhi Technological University** has presented a research paper entitled **Comparative Analysis of Model-Free Reinforcement Learning Algorithms in Dynamic and Partially Observable Grid-world Environments** in **3rd International Conference on Recent Trends in Mathematical Sciences (ICRTMS-2025)** organized by the **Himachal Ganita Parishad (HGP) at Himachal Pradesh University, Shimla on 10th-11th May, 2025.**

Dr. Kamalendra Kumar	Dr. Neetu Dhiman	Dr. Shalini Gupta
Co-Convener	Convener	President (HGP)

Bhakti_and_Shailly.pdf

ORIGINALITY REPORT

10%	8%	8%	5%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Delhi Technological University Student Paper	1%
2	Xinyuan Song, Keyu Chen, Ziqian Bi, Qian Niu, Junyu Liu, Benji Peng, Sen Zhang, Ming Liu, Ming Li, Xuanhe Pan. "Mastering Reinforcement Learning: Foundations, Algorithms, and Real-World Applications", Open Science Framework, 2024 Publication	1%
3	dspace.dtu.ac.in:8080 Internet Source	1%
4	arxiv.org Internet Source	<1%
5	Wu, Haiping. "Self-Supervised Attention-Aware Reinforcement Learning", McGill University (Canada), 2021 Publication	<1%
6	www.theseus.fi Internet Source	<1%
7	Narasimhan, Sai Prasanth Bangalore Lakshmi. "Robotic Simulation Learning Approaches for Scalable Safety and Robustness in Assistive Robotics and Human Motion Control", North Carolina State University Publication	<1%
	www.sce.carleton.ca	

8	Internet Source	<1%
9	Chong Li, Meikang Qiu. "Reinforcement Learning for Cyber-Physical Systems - with Cybersecurity Case Studies", CRC Press, 2019 Publication	<1%
10	vdoc.pub Internet Source	<1%
11	Submitted to dtusimilarity Student Paper	<1%
12	Kuldeep Singh Kaswan, Jagjit Singh Dhatterwal, Anand Nayyar. "Digital Personality: A Man Forever - Volume 3: Ontologies to Dialogue Generation", CRC Press, 2025 Publication	<1%
13	Submitted to Liverpool Hope Student Paper	<1%
14	incompleteideas.net Internet Source	<1%
15	0-www-mdpi-com.brum.beds.ac.uk Internet Source	<1%
16	docserv.uni-duesseldorf.de Internet Source	<1%
17	Satya Ranjan Mishra, Apul Narayan Dev, Alok Kumar Pandey, Mukesh Kumar Awasthi. "Design Optimization Using Artificial Intelligence", CRC Press, 2025 Publication	<1%
18	Submitted to University of Birmingham Student Paper	<1%

19	Submitted to University of Sheffield Student Paper	<1 %
20	pdfs.semanticscholar.org Internet Source	<1 %
21	Submitted to Southern New Hampshire University - Continuing Education Student Paper	<1 %
22	Submitted to Universidad de Alcalá Student Paper	<1 %
23	Adaptation Learning and Optimization, 2012. Publication	<1 %
24	Submitted to Middle East Technical University Student Paper	<1 %
25	Osman, Altaaf. "The Role of Risk Culture in Rational Strategic Decision-Making", University of Pretoria (South Africa), 2023 Publication	<1 %
26	ngocbh.github.io Internet Source	<1 %
27	Submitted to ABV-Indian Institute of Information Technology and Management Gwalior Student Paper	<1 %
28	Barbara Zaparoli Cunha, Christophe Droz, Abdel-Malek Zine, Stéphane Foulard, Mohamed Ichchou. "A review of machine learning methods applied to structural dynamics and vibroacoustic", Mechanical Systems and Signal Processing, 2023 Publication	<1 %

29	Mohit Sewak. "Deep Reinforcement Learning", Springer Science and Business Media LLC, 2019 Publication	<1 %
30	Pawan Singh, Prateek Singhal, Pramod Kumar Mishra, Avimanyou K. Vatsa. "Heterogenous Computational Intelligence in Internet of Things", CRC Press, 2023 Publication	<1 %
31	www.mdpi.com Internet Source	<1 %
32	Submitted to University of Oxford Student Paper	<1 %
33	cuir.car.chula.ac.th Internet Source	<1 %
34	deepai.org Internet Source	<1 %
35	Malekzadeh, Parvin. "Advancing Efficiency and Safety in Autonomous Sequential Decision Making.", University of Toronto (Canada), 2024 Publication	<1 %
36	Submitted to University of Bristol Student Paper	<1 %
37	la.disneyresearch.com Internet Source	<1 %
38	www.shannonholloway.com Internet Source	<1 %

Exclude quotes	On	Exclude matches	< 10 words
Exclude bibliography	On		