Department of Informatics
King's College London
United Kingdom

7CCSMPRJ - Individual Project

Explainable Deep Learning for Portfolio Optimisation

Name: **Ingrid Pérez Aguilera**

Student Number: K24087939

Course: Computational Finance M.Sc.

Supervisor: **Riaz Ahmad**

Abstract

It is a précis of the report (normally in one page), which should include:

- A brief introduction to the project objectives
- A brief description of the main work of the project
- A brief description of the contributions, major findings, results achieved and principal conclusion of the project

Acknowledgements

It is a short paragraph to thank those who have contributed to the project work.

Table of Contents

List of Figures

List of Tables

List of Algorithms

Nomenclature

c	Speed of light in a vacuum
h	Planck constant

Glossary

Advantage Actor-Critic Algorithm that uses both an actor (policy) and a critic (value function) to learn optimal policies by estimating the advantage of actions taken. xi, 3, 16, 17, 37

Algorithmic Trading Use of computer algorithms to automate trading decisions and execute trades in financial markets. 1

Artificial Intelligence Simulation of human intelligence processes by machines, especially computer systems, enabling them to perform tasks that typically require human intelligence, such as visual perception, speech recognition, decision-making, and language translation. xi, 2, 9, 23

Bellman Equations Set of equations that describe the relationship between the value of a state or action and the values of subsequent states or actions in dynamic programming and reinforcement learning, used to compute optimal policies. 15, 20

Deep Deterministic Policy Gradient Algorithm that uses deep neural networks to learn policies for continuous action spaces, combining the benefits of deep learning and policy gradient methods. x, xi, 3, 16, 19, 37

Deep Learning Subset of machine learning that uses neural networks with many layers

to learn from large amounts of data, enabling the model to automatically learn complex patterns and representations. xi, 2, 4, 6, 10, 17, 23

Deep Neural Network Type of artificial neural network with multiple layers that can learn complex representations and patterns in data. xi, 10

Deep Reinforcement Learning Combination of deep learning and reinforcement learning, where deep neural networks are used to approximate the value function or policy in reinforcement learning tasks. xi, 3, 6, 10, 33, 34

Efficient Frontier Set of optimal portfolios that offer the highest expected return for a given level of risk, or the lowest risk for a given level of expected return, in the context of portfolio optimisation. 8

Entropy Measure of uncertainty or randomness in a system, often used in the context of information theory and reinforcement learning to encourage exploration. 21

Explainable Artificial Intelligence Methods and techniques in the application of artificial intelligence that make the results of the models understandable by humans, providing insights into how decisions are made. xii, 2, 23

Feature Importance Technique for determining the contribution of each feature in a machine learning model to its predictions, helping to identify which features are most influential. 4, 6, 24

Financial Markets Marketplaces where people trade financial securities, commodities, and other fungible items of value at low transaction costs and at prices that reflect supply and demand. 1

Hyper-parameter Parameter that is set before the learning process begins and control the learning process of a machine learning model, such as learning rate, batch size, and number of layers in a neural network. 2

Local Interpretable Model-agnostic Explanations Technique for explaining the predictions of any machine learning model by approximating it with a locally interpretable model, allowing users to understand the model's behaviour in a specific instance. xi, 4–6, 23, 25

Machine Learning Subset of artificial intelligence that enables systems to learn from data and improve their performance over time without being explicitly programmed. xi, 1, 6, 9, 25

Markov Decision Process Mathematical framework for modelling decision-making in situations where outcomes are partly random and partly under the control of a decision maker, characterised by states, actions, rewards, and transition probabilities. ix, xi, 11–13, 33, 34

Modern Portfolio Theory Investment theory that aims to construct a portfolio of assets that maximises expected return for a given level of risk, or minimises risk for a given level of expected return, through diversification. xi, 7

Partially Observable Markov Decision Process Extension of Markov Decision Process where the agent does not have access to the complete state information, requiring the use of belief states or observations to make decisions. xi, 13

Portfolio Optimisation Process of selecting the best distribution of assets in a portfolio to achieve specific investment goals, such as maximising returns or minimising risk, while considering constraints and preferences. 1, 6, 33

Proximal Policy Optimisation Algorithm that optimises policies by ensuring that updates to the policy are not too large, maintaining a balance between exploration and exploitation. xii, 3, 16, 18, 37

Reinforcement Learning Subset of machine learning where an agent learns to make decisions by taking actions in an environment to maximise cumulative reward. xii, 2, 4, 6, 10, 16, 17

Reward Function Function that defines the feedback signal received by an agent in reinforcement learning, guiding the agent's learning process by providing rewards or penalties based on its actions. 2

SHapley Additive exPlanations Method for interpreting machine learning models by assigning each feature an importance value for a particular prediction, based on cooperative game theory. xii, 4, 6, 23, 26

Soft Actor-Critic Algorithm that combines the benefits of off-policy learning and entropy regularisation, allowing for more exploration and better stability in learning policies for continuous action spaces. xii, 3, 16, 21, 37

Supervised Learning Machine learning task where a model is trained on labelled data to learn a mapping from inputs to outputs. 10

Twin Delayed Deep Deterministic Policy Gradient Extension of Deep Deterministic Policy Gradient that addresses the overestimation bias in value function estimation by using two critic networks and delaying policy updates. xii, 3, 16, 20, 37

Unsupervised Learning Machine learning task where a model learns patterns or structures in unlabelled data without explicit supervision. 10

Acronyms

A2C Advantage Actor-Critic. 3, 4, 16–18, 22, 37

A3C Asynchronous Advantage Actor-Critic. 16, 18

AI Artificial Intelligence. 2, 9, 23

DDPG Deep Deterministic Policy Gradient. 3, 4, 16, 19, 20, 22, 37

DL Deep Learning. 2, 4, 6, 10, 17, 23

DNN Deep neural networks. 10, 17

DPG Deterministic Policy Gradient. 19

DQN Deep Q-Network. 16, 19, 20

DRL Deep Reinforcement Learning. 3–6, 10, 17, 22, 33, 34, 37

LIME Local Interpretable Model-agnostic Explanations. 4–6, 23, 25, 31

MDP Markov Decision Process. 11–14, 33, 34

ML Machine Learning. 1–6, 9, 10, 25, 28

MPT Modern Portfolio Theory. 7, 9

POMDP Partially Observable Markov Decision Process. 13

PPO Proximal Policy Optimisation. 3, 4, 16, 18, 19, 22, 37

RL Reinforcement Learning. 2, 4, 6, 10, 11, 15–17, 21, 22

SAC Soft Actor-Critic. 3, 4, 16, 21, 22, 37

SHAP SHapley Additive exPlanations. 4–6, 23, 26, 28, 30, 31

TD3 Twin Delayed Deep Deterministic Policy Gradient. 3, 4, 16, 20–22, 37

TRPO Trust Region Policy Optimisation. 18, 19

XAI Explainable Artificial Intelligence. 2, 23, 24

Chapter 1

Introduction

Financial markets are highly complex systems influenced by numerous factors, including financial and political events, social trends and technological advancements. Moreover, their evolving and stochastic nature requires using the most advance computational developments to model the financial environment. The tasks of financial time series prediction and Portfolio optimisation are considerably intricate, due to the semi-strong form of market efficiency and the high level of noise. [3]

Algorithmic trading focuses on the application of analytical methods to automatically execute trading actions based on an algorithm without human intervention. In its early days, the field mainly studied the usage of a computer program to follow a predefined strategy [4]. Nonetheless, in recent years, algorithmic trading has evolved to a problem in which environment perception entails learning feature representation from highly non-stationary and noisy financial time series data and decision-making requires the algorithm to explore the environment and simultaneously make correct decisions in an online manner without supervision [5].

Machine Learning (ML) is at an advantage for the task given its capability to learn from

historical data and make predictions about the future state of an environment. In the past years, research has explored the application of Deep Learning (DL) in future price prediction of financial assets [3, 6, 7, 8]. However, its main disadvantage is the inability to directly deal with trading, requiring an additional step to convert the predictions into actionable strategies. In contrast, Reinforcement Learning (RL) would allow the algorithm to learn a trading strategy directly from the environment, without the need for a separate step [9, 10]. In this case, there are two main approaches: first, the algorithm can learn the amount of assets to buy, sell or hold at each time step [11], or second, the algorithm can learn the optimal portfolio allocation and automatically rebalance the portfolio weights at each time step [12].

Despite the potential of RL in portfolio optimisation, its widespread adoption in the financial industry remains limited. This is primarily due to following challenges [13]:

1. difficulty in finding the appropriate algorithm with a suitable reward function and hyper-parameters to ensure efficiency and performance,
2. challenge of testing the algorithm in a real-world environment, and
3. lack of transparency of ML models, often referred to as black boxes, making it increasingly complex to interpret the algorithm's decisions.

In recent years, the rise in popularity of Artificial Intelligence (AI) and its widespread use have led to concerns regarding its decisions due to its black-box nature. The concept of explainability in AI, known as Explainable Artificial Intelligence (XAI), refers to a model's ability to provide details and reasons to make itself understandable [14]. The term was first coined in 2016 to describe the need for users to effectively understand, trust and manage artificial intelligence applications [15]. The need for explainability becomes particularly relevant in the context of financial usage, where the regulatory framework requires transparency and accountability in automated decision-making. Various relevant

applications, including volatility models [16], credit risk assessment [17] and portfolio construction [13] have explored the concept of explainability in financial applications. This highlights its importance and the need to explore its advantages in more complex models, without inadvertently increasing the complexity of the overall methodology.

Consequently, this thesis will focus on addressing the aforementioned challenges by exploring the application of Deep Reinforcement Learning (DRL) to portfolio optimisation and implementing post-hoc explainability techniques.

1.1 Objectives

The objective of this thesis is to develop an explainable Deep Reinforcement Learning model for portfolio optimisation. A DRL model has the ability to leverage historical financial data to learn an investment strategy that efficiently allocates financial assets while maximising expected returns and minimising risk. Moreover, the incorporation of advanced explainability techniques enhances the interpretability and transparency of the model's decision-making. This project aims to bridge the gap between cutting-edge machine learning techniques and their practical application in finance by addressing the challenges of algorithm selection, simulation of real-world scenarios, and black box nature of ML models.

First, DRL models, such as Advantage Actor-Critic (A2C), Proximal Policy Optimisation (PPO), Deep Deterministic Policy Gradient (DDPG), Twin Delayed Deep Deterministic Policy Gradient (TD3) and Soft Actor-Critic (SAC), will be implemented to learn the optimal portfolio allocation from high-dimensional environment representations. The algorithms will be trained on historical financial data, including technical and macroeconomic indicators, with the goal of capturing the complex market dynamics. Each of the algorithms is better suited to a particular scenario, for instance, DDPG

encourages maximum returns, while A2C reduces the variance.

Second, post-hoc explainability techniques: Feature Importance, SHapley Additive exPlanations (SHAP) values and Local Interpretable Model-agnostic Explanations (LIME) analysis, will be applied to interpret the model's decisions. The goal of these is to understand which market conditions, represented by financial data, lead to the actions/decisions, encoded as portfolio weights.

Finally, the performance of the DRL models will be analysed in different scenarios, including the impact of a larger financial environment representation, portfolio size and asset composition. The results will be compared with traditional portfolio optimisation methods to evaluate the effectiveness of the proposed approach.

1.2 Report Structure

This report is organised into six chapters, each of which focuses on a concrete area related to the problem. Additional material, together with source code, is included in the appendices.

The current chapter, 1, presents the motivation and the objectives of this thesis. It gives an overview of the potential of DRL in portfolio optimisation and its main challenges, particularly the lack of transparency of ML models.

Chapter 2 provides an overview of the theoretical background of the project, including financial markets and machine learning concepts. The problem of portfolio optimisation in the financial domain is outlined and the potential of DRL in this context is discussed. The chapter provides a comprehensive background explanation of the fundamentals of Deep Learning and Reinforcement Learning, including the main algorithms (A2C, PPO, DDPG, TD3, SAC) and techniques in their intersection, DRL. In addition, it gives an overview of the post-hoc explainability techniques: feature importance, SHapley Addi-

tive exPlanations (SHAP) values and Local Interpretable Model-agnostic Explanations (LIME) analysis, which will be used to interpret the model's decisions. Finally, it includes a comprehensive in-depth literature review on the topics of ML applied to portfolio optimisation and relevant applications of explainability techniques.

The methodology chapter 3 describes the techniques and methods used to solve the problem and outlines the implementation of the proposed solution. The chapter provides a detailed explanation of the architecture and components of the proposed DRL model, including the state representation, reward function, and training process.

The results of the experiments are presented in chapter 4, which analyses and evaluates the results obtained from the proposed implementation, while critically discussing the findings. It provides a detailed comparison of the proposed DRL strategies with traditional portfolio optimisation methods. Furthermore, it consists of an in-depth analysis of the model's decisions using post-hoc explainability techniques, in particular, SHAP values, feature importance and LIME analysis.

Chapter 5 discusses the legal, social, ethical and professional implications within the context of the project. By addressing these issues, the project aims to ensure that the proposed solution adheres to industry standards, while considering the implications of the technology.

Finally, the report concludes with a summary of the main points of the work, the contributions made, the results achieved as well as potential applications and future work in chapter 6.

Chapter 2

Background

This chapter provides an overview of the problem of portfolio optimisation in the financial domain, followed by a comprehensive explanation of the fundamentals of Deep Learning (DL) and Reinforcement Learning (RL), including the relevant algorithms in the field of Deep Reinforcement Learning (DRL). In addition, it discusses the need for explainability in Machine Learning (ML) and the main techniques used to achieve it: SHapley Additive exPlanations (SHAP), Local Interpretable Model-agnostic Explanations (LIME) and Feature Importance. Finally, it presents the state of the art in portfolio optimisation using DRL and the recent advancements in explainability techniques in the field.

2.1 Portfolio Optimisation

Portfolio optimisation is the process of selecting optimal weights for a portfolio of assets in order to maximise expected returns for a given level of risk, or conversely, to minimise risk for a given level of expected returns [18]. In mathematical terms, the problem requires finding a solution to the specified objective function, which is typically a function of the expected returns and the risk associated with the portfolio [19]. The task becomes

further complicated if a time dimension is introduced, as the portfolio weights need to be adjusted over time to capture the changes in market conditions and asset prices [20].

2.1.1 Modern Portfolio Theory

There exist several traditional frameworks that formalise the problem of portfolio allocation. Markowitz's Modern Portfolio Theory (MPT) was proposed in 1952 [21] and it provides a mathematical framework where investors choose optimal portfolios based on risk and return, by either minimising the risk given a specified return or, maximising the return given a specified risk [22]. The theory extends the concept of diversification by suggesting that owning financial assets of different kinds is less risky than owning assets of the same kind, due to the correlations between assets.

The main assumptions in MPT are:

- investors are risk-averse, rational, and seek to maximise return for a given risk;
- returns are normally distributed;
- markets are frictionless, meaning there are no transaction costs; and
- assets are infinitely divisible.

Under these assumptions, portfolio risk and return can be modelled as an optimisation problem. Let $\mathbf{w} = (w_1, w_2, \ldots, w_N)^T$ denote the portfolio weight vector, where each w_i indicates the proportion of capital allocated to asset i, subject to the budget constraint:

$$\sum_{i=1}^{N} w_i = 1 \quad \Leftrightarrow \quad \mathbf{w}^T \mathbf{1} = 1 \tag{2.1}$$

with $\mathbf{1} \in \mathbb{R}^N$ being a vector of ones, and subject to the non-negativity constraint,

meaning that short-selling is not allowed:

$$w_i \geq 0 \quad \forall i = 1, 2, \ldots, N. \tag{2.2}$$

Let $\boldsymbol{\mu} = (R_1, R_2, \ldots, R_N)^T$ represent the vector of expected returns, and $\Sigma \in \mathbb{R}^{N \times N}$ the covariance matrix of asset returns. The expected return of the portfolio is then given by:

$$R_p = \mathbf{w}^T \boldsymbol{\mu}, \tag{2.3}$$

and the portfolio risk is quantified by the variance of returns:

$$\sigma_p^2 = \mathbf{w}^T \Sigma \mathbf{w}. \tag{2.4}$$

This formulation provides the foundation for solving the mean-variance optimisation problem, by either:

- minimising portfolio variance σ_p^2 subject to a target expected return R_p, or
- maximising expected return R_p subject to a risk constraint σ_p.

The Markowitz mean-variance optimisation problem can be expressed as:

$$\begin{aligned} &\min_{\mathbf{w}} \quad \mathbf{w}^T \Sigma \mathbf{w} \\ &\text{subject to} \quad \begin{cases} \mathbf{w}^T \boldsymbol{\mu} = R_p \\ \mathbf{w}^T \mathbf{1} = 1 \\ \mathbf{w} \geq 0 \end{cases} \end{aligned} \tag{2.5}$$

Solving the mean-variance optimisation problem for varying levels of target return leads to a set of optimal portfolios that form the efficient frontier. It is typically visualised

in a risk-return space, where the x-axis represents the risk (standard deviation) and the y-axis represents the expected return, as shown in Figure 2.1. Portfolios below the curve are suboptimal, while those on the frontier represent the best achievable combinations of risk and return.

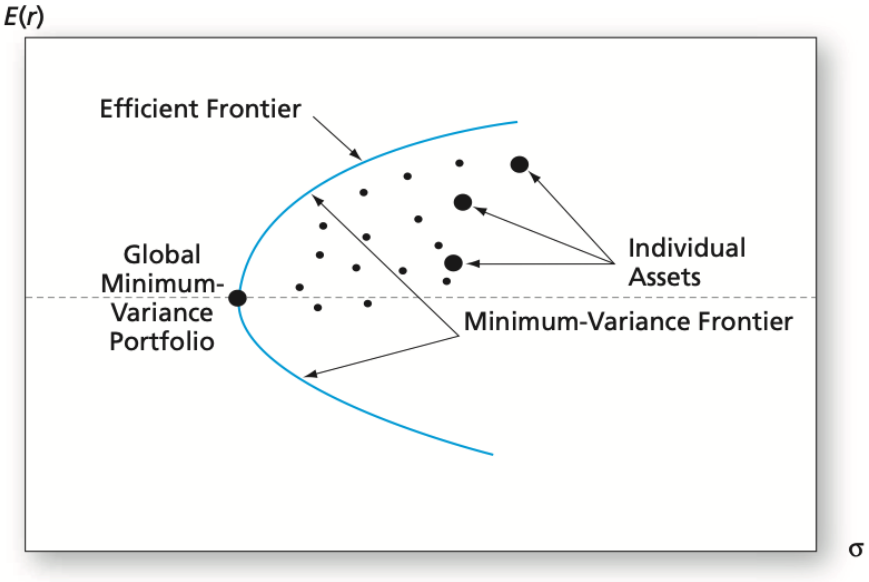

Figure 2.1: Efficient Frontier in Risk-Return Space. [1]

Despite the simplicity in the formulation of MPT, its assumptions do not reflect the behaviour of real markets. Moreover, modern markets are dynamic, non-stationary, and feature non-linear relationships, which have driven research into other approaches better suited to capture the complexities of modern financial markets.

2.2 Deep Reinforcement Learning

Machine Learning is a branch of Artificial Intelligence (AI) that focuses on the use of data and algorithms to imitate the way humans learn, gradually improving their accuracy over time [23]. There are three main tasks in ML [24]:

- **Supervised Learning**: Task of training a classification or regression model from labelled training data, where the model learns to map inputs to outputs based on examples.
- **Unsupervised Learning**: Task of drawing inferences from datasets consisting of unlabelled input data, where the model learns to identify patterns or structures in the data.
- **Reinforcement Learning**: Task of training an agent to sequentially make decisions by taking actions in an environment with the goal of maximising cumulative reward, using feedback from the environment to learn an optimal strategy.

Deep Learning is a set of methods and techniques to solve such ML tasks, specially in supervised and unsupervised learning tasks. DL focuses on the use of Deep neural networks (DNN) [25], which are characterised by a succession of layers of non-linear transformation that allow the model to learn a representation of the data with various levels of abstraction.

Therefore, Deep Reinforcement Learning (DRL) combines Deep Learning (DL) and Reinforcement Learning (RL) to solve sequential decision-making problems with high-dimensionality in the environment representation. This approach has gained significant attention in recent years due to its success in various applications, including robotics [26] and game playing [27, 28].

2.2.1 Reinforcement Learning

As mentioned, RL is a type of ML that solves the problem of sequential decision-making through continuous interaction with an environment. The agent learns to take actions given a representation of the environment's state with the goal of optimising a pre-defined notion of reward. The agent learns by successively adjusting its policy based on

its observations and interactions with the environment.

The RL problem can be formalised as a discrete-time stochastic control process where an agent interacts with the environment. At each time step t, the agent observes the state of the environment $s_t \in \mathcal{S}$, takes an action $a_t \in \mathcal{A}$ to obtain a reward $r_t \in \mathbb{R}$ and transition to a new state $s_{t+1} \in \mathcal{S}$, where $\mathcal{S}$ is the state space and $\mathcal{A}$ is the action space [24]. The agent's interaction with the environment is visually represented in Figure 2.2.

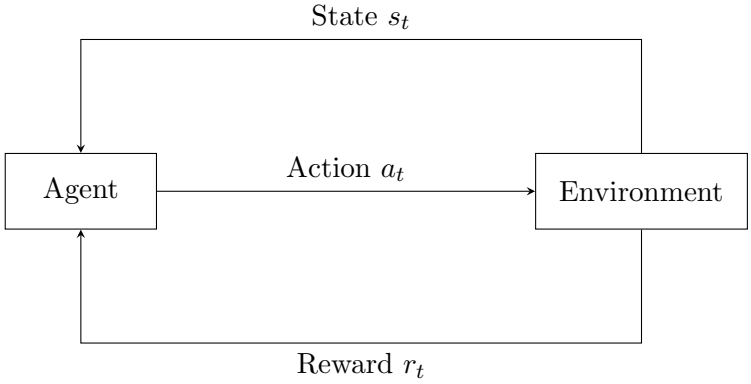

Figure 2.2: Agent interaction with environment

A discrete time stochastic control process can be formalised as a Markov Decision Process (MDP), if it fulfils the Markov Property.

Definition 2.2.1 (Markov Property)**.** A discrete time stochastic control process satisfies the Markov Property if:

$$P(s_{t+1}|s_t, a_t) = P(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, \ldots, s_0, a_0) \tag{2.6}$$

$$P(r_t|s_t, a_t) = P(r_t|s_t, a_t, s_{t-1}, a_{t-1}, \ldots, s_0, a_0) \tag{2.7}$$

where $P(s_{t+1}|s_t, a_t)$ is the transition probability of moving to state s_{t+1} given the current state s_t and action a_t, and $P(r_t|s_t, a_t)$ is the reward function that defines the expected reward received at time t given the current state and action.

This implies that the state s_{t+1} at a future time step $t+1$ only depends on the current state s_t and action a_t. Similarly, the reward r_t at time step t only depends on the current state and action and not on the history of previous states and actions. Consequently, a Markov Decision Process [29] is a discrete time stochastic control process defined as:

Definition 2.2.2 (Markov Decision Process)**.** An MDP is a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, T, R, \gamma)$, where:

- $\mathcal{S}$ is the state space: $s_t \in \mathcal{S}$,
- $\mathcal{A}$ is the action space: $a_t \in \mathcal{A}$,
- $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition function,
- $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathcal{R}$ is the reward function, where $\mathcal{R} \in [0, R_{max}]$ is the set of all possible rewards bounded by $R_{max} \in \mathbb{R}^+$, and
- $\gamma \in [0, 1)$ is the discount factor.

At each time step, the probability of advancing to the next state s_{t+1} is given by the transition function $T(s_t, a_t, s_{t+1})$ and the reward r_t is given by the reward function $R(s_t, a_t, s_{t+1})$. This can be visualised in Figure 2.3.

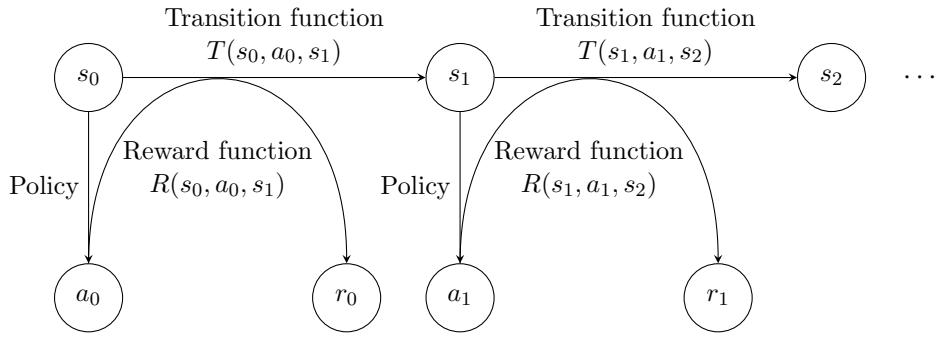

Figure 2.3: Markov Decision Process with policy, transition, and reward functions

The agent's objective is to learn a policy $\pi : \mathcal{S} \to \mathcal{A}$ that maps states to actions, in order to maximise the expected cumulative reward over time. Policies can be categorised as:

- deterministic: $\pi(s) : \mathcal{S} \rightarrow \mathcal{A}$, at a given state s, the policy specifies the only available action to take, or
- stochastic: $\pi(s, a) : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$, at a given state s, the policy specifies the probability of taking action a.

Markov Decision Process are based on the idea that the current state is fully representative of the environment. However, in most real world scenarios, the agent does not have access to the complete state. In such cases, Partially Observable Markov Decision Process (POMDP) can be used to model the uncertainty in the agent's observations and actions.

The goal of the agent is to maximise the cumulative long-term reward G_t, which is defined as the sum of discounted rewards over time:

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = r_{t+1} + \gamma G_{t+1} \tag{2.8}$$

where $\gamma \in [0, 1)$ is the discount factor and is used to balance the importance between immediate and future rewards. If the discount factor is set to 0, the agent is myopic and only maximises the immediate reward; whereas, as γ approaches 1, the agent becomes more far-sighted and places greater importance on future rewards.

The expected cumulative reward is defined as the state value function $V^\pi(s) : \mathcal{S} \rightarrow \mathbb{R}$, which is the expected return when starting from state s and following policy π:

$$V^\pi(s) = \mathbb{E}_\pi \left[G_t | s_t = s \right] \tag{2.9}$$

where $\mathbb{E}_\pi$ denotes the expectation over the policy π.

Similarly, the state-action value function $Q^\pi(s, a) : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is defined as the expected

return when starting from state s, taking action a, and then following policy π:

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left[G_t | s_t = s, a_t = a\right] \tag{2.10}$$

The state value function $V^{\pi}(s)$ and the state-action value function $Q^{\pi}(s,a)$ are related as follows:

$$V^{\pi}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left[Q^{\pi}(s,a) \mid s_t = s\right] \tag{2.11}$$

Moreover, the advantage function $A^{\pi}(s,a)$ combines both the state value function $V^{\pi}(s)$ and the state-action value function $Q^{\pi}(s,a)$, and is defined as:

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s) \tag{2.12}$$

A policy π is said to be optimal if the policy's value function is the optimal value function of the MDP, defined as:

$$V^{*}(s) = \max_{\pi'} V^{\pi'}(s), \forall s \in \mathcal{S} \tag{2.13}$$

$$Q^{*}(s,a) = \max_{\pi'} Q^{\pi'}(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A} \tag{2.14}$$

The optimal policy π^{*} is:

$$\pi^{*}(s) = \arg\max_{a \in \mathcal{A}} Q^{*}(s,a) \tag{2.15}$$

As a result, the optimal policy is the greedy policy that performs the optimal actions at each time step as determined by the optimal value functions. This framework enables the agent to determine optimal actions that maximise long-term returns by evaluating immediate information, without requiring knowledge of the values of future states and actions.

The Bellman equations [30] provide a recursive relation between the value functions in terms of the future state/action values. There are four main Bellman equations, classified in two groups: the Bellman expectation equations and the Bellman optimality equations. The Bellman expectation equations are defined as follows:

$$V^{\pi}(s) = \sum_{a \in \mathcal{A}} \pi\left(a \mid s\right) \sum_{s' \in \mathcal{S}} T\left(s, a, s'\right) \left[R\left(s, a\right) + \gamma V^{\pi}(s')\right] \tag{2.16}$$

$$Q^{\pi}(s, a) = \sum_{s' \in \mathcal{S}} T\left(s, a, s'\right) \left[R\left(s, a\right) + \gamma \sum_{a' \in \mathcal{A}} \pi\left(a' \mid s'\right) Q^{\pi}(s', a')\right] \tag{2.17}$$

and the Bellman optimality equations are defined as:

$$V^{*}(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} T\left(s, a, s'\right) \left[R\left(s, a\right) + \gamma V^{*}(s')\right] \tag{2.18}$$

$$Q^{*}(s, a) = \sum_{s' \in \mathcal{S}} T\left(s, a, s'\right) \left[R\left(s, a\right) + \gamma \max_{a' \in \mathcal{A}} Q^{*}(s', a')\right] \tag{2.19}$$

Although explicitly solving the Bellman equations would lead to the optimal policy, it is often intractable due to the size of the state and action spaces. Therefore, in RL algorithms, the goal is to learn an approximation of the optimal value functions, which can be used to derive the optimal policy. Another problem that arises is that of balancing exploration and exploitation [31]. Theoretically, following the greedy action yields the optimal policy, but this is only true if all the action values are known. In practice, the agent at each time step and given state chooses either an action whose value is higher, thus exploiting its current knowledge, or picks an action at random, thus exploring the environment and gaining more information about the state-action space, leading to potentially discovering a better action than the greedy one at the current time.

2.2.2 Deep Reinforcement Learning Algorithms

A Reinforcement Learning agent includes one or more of the following components [24]:

- a representation of the value function that provides a prediction of the value of each state or state-action pair,
- a direct representation of the policy π, and
- a model of the environment, consisting of estimates of transition and reward functions.

Depending on the components, the main Reinforcement Learning paradigms are:

- **Model-free** algorithms do not learn a representation of the environment, but focus on either the value function or the policy.
  - **Value-based** algorithms learn an approximation of the value function, which is used to compute the state or state-action values. The policy is not learnt explicitly but can be derived from the value function. Examples include Deep Q-Network (DQN) [32] and C51 [33].
  - **Policy-based** algorithms learn a direct representation of the policy, which is used to select actions. Advantage Actor-Critic (A2C), Asynchronous Advantage Actor-Critic (A3C) [34] and Proximal Policy Optimisation (PPO) [35] are examples of policy-based algorithms.
  - **Actor-Critic** algorithms combine both value-based and policy-based approaches, where the actor learns the policy and the critic learns the value function. The critic provides feedback to the actor to improve the policy. Some examples are Deep Deterministic Policy Gradient (DDPG) [36], Twin Delayed Deep Deterministic Policy Gradient (TD3) [37], and Soft Actor-Critic (SAC) [38].

- **Model-based** algorithms include a model of the environment, which can be used to simulate future states and rewards. For example, World Models [39] learns a model of the environment and AlphaZero [40] has a representation of the model.

The taxonomy of the RL algorithms is shown in Figure 2.4.

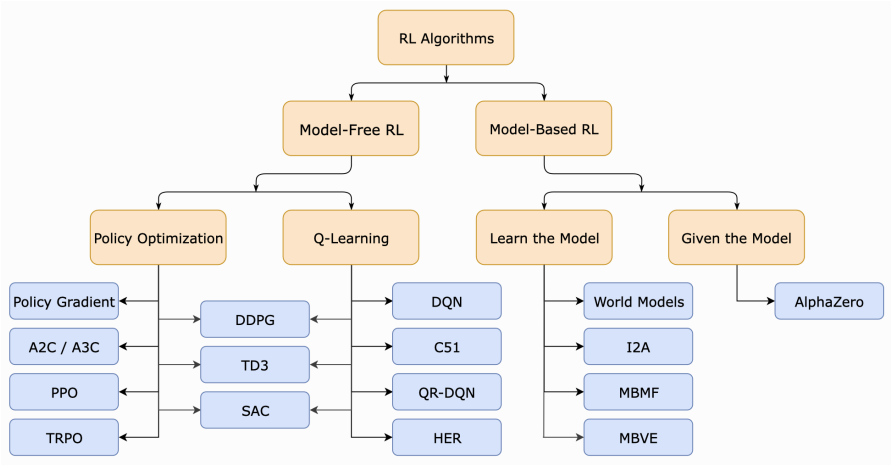

Figure 2.4: Taxonomy of Reinforcement Learning Algorithms [2]

The full potential of Reinforcement Learning algorithms is achieved when leveraging the power of Deep Learning to solve dynamic stochastic control problems that have high-dimensionality in their representation of the state and action spaces. DRL algorithms use DNNs to approximate the value functions, or use gradient ascent to find the optimal policy parameters. This thesis will focus on the following DRL algorithms on policy-based and actor-critic algorithms.

2.2.2.1 Advantage Actor-Critic (A2C)

The Advantage Actor-Critic (A2C) algorithm was developed by Mnih et al. (2016) in their paper on *Asynchronous methods for deep reinforcement learning* [34]. The main contribution is the usage of the advantage function to address the variance issues present

in policy gradient methods. The A2C is the synchronous version of A3C, and is preferred due to its better performance in terms of training time and cost-effectiveness [41].

The algorithm consists of a dual-network architecture, where the actor network learns a stochastic policy $\pi(a_t \mid s_t; \theta)$ and the critic network learns the value function $V(s_t; \theta_v)$. The policy and the value function are updated after every t_max actions or when a terminal state is reached. The advantage function is estimated as follows:

$$A(s_t, a_t; \theta, \theta_v) = \sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V(s_{t+k}; \theta_v) - V(s_t; \theta_v) \tag{2.20}$$

where k represents the n-step return and is upper-bounded by the maximum number of steps t_{max}, and γ is the discount factor. The algorithm's objective function is defined as:

$$J(\theta, \theta_v) = \mathbb{E}_\pi \left[\log \pi(a_t \mid s_t; \theta) A(s_t, a_t; \theta, \theta_v)\right] \tag{2.21}$$

The pseudo-code for the algorithm for A2C is outlined in appendix A.

2.2.2.2 Proximal Policy Optimisation (PPO)

Proximal Policy Optimisation (PPO) is a policy gradient algorithm that was introduced by Schulman et al. (2017) in their paper on *Proximal Policy Optimization Algorithms* [35] with the objective of constraining policy updates. The algorithm balances sufficiently large policy updates in order to improve the policy, while avoiding excessively large changes that could hinder performance.

PPO improves the performance of Trust Region Policy Optimisation (TRPO) [42] by using a clipped surrogate objective function, by ensuring that the probability ratio r_t is bounded within a range of $[1 - \epsilon, 1 + \epsilon]$, where ϵ is a hyperparameter that controls the

clipping range. The objective function is defined as:

$$L_t^{CLIP}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t \right) \right] \tag{2.22}$$

where

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \tag{2.23}$$

is the probability ratio between the current policy π_θ and the old policy $\pi_{\theta_{old}}$

Another improvement with respect to TRPO is the use of simple first-order optimisation methods as opposed to the second-order methods used in TRPO. Consequently, PPO maintains the stability and reliability of other trust-region methods, while being easier to implement and more computationally efficient. The pseudo-code for the algorithm for PPO is outlined in appendix A.

2.2.2.3 Deep Deterministic Policy Gradient (DDPG)

Deep Deterministic Policy Gradient (DDPG) is an off-policy actor-critic algorithm that was introduced by Lillicrap et al. (2015) in their paper on *Continuous control with deep reinforcement learning* [36]. The algorithm arises to solve the challenges of applying Deep Q-Network [32] to continuous action spaces by applying Deterministic Policy Gradient (DPG), which enables the efficient computation of the policy gradient without the need to integrate over the action space.

The algorithm uses the following networks: the actor network $\mu(s_t; \theta_\mu)$, which learns a deterministic policy, the critic network $Q(s_t, a_t; \theta_Q)$, which learns the state-action value function, and the actor's and critic's target networks. The actor network is updated using DPG:

$$\nabla_{\theta_\mu} J \approx \mathbb{E}_{s_t \sim \mathcal{D}} \left[\nabla_a Q(s_t, a_t; \theta_Q) \nabla_{\theta_\mu} \mu(s_t; \theta_\mu) \right] \tag{2.24}$$

where $\mathcal{D}$ is the replay buffer that stores the agent's experiences, and the critic network is updated using the Bellman equations:

$$\nabla_{\theta_Q} J \approx \mathbb{E}_{s_t \sim \mathcal{D}} \left[\left(r_t + \gamma Q(s_{t+1}, \mu(s_{t+1}; \theta_\mu); \theta'_Q) - Q(s_t, a_t; \theta_Q) \right) \nabla_{\theta_Q} Q(s_t, a_t; \theta_Q) \right] \quad (2.25)$$

where θ'_Q are the parameters of the target critic network.

From Deep Q-Network, the algorithm incorporates a replay buffer to store the agent's experiences, which allows the agent to learn from past experiences and improve its performance over time. Moreover, DDPG incorporates noise, typically Ornstein-Uhlenbeck noise [43], to the actions taken by the actor network to encourage exploration of the action space. The pseudo-code for the algorithm for DDPG is outlined in appendix A.

2.2.2.4 Twin Delayed Deep Deterministic Policy Gradient (TD3)

Twin Delayed Deep Deterministic Policy Gradient (TD3) is an extension of DDPG introduced by Fujimoto et al. (2018) in their paper on *Addressing Function Approximation Error in Actor-Critic Methods* [37]. The proposal addresses the hyper-parameter sensitivity and overestimation bias present in the critic network of DDPG. There are three main improvements to DDPG:

- **Clipped Double Q-Learning**: The algorithm employs two critic networks and uses their minimum value to compute the target value for the actor network, which reduces the overestimation bias.
- **Delayed Policy Updates**: The actor and critic networks updates are performed at different frequencies. The critic networks are updated every time step, whereas the actor and target networks are updated less frequently. The main benefit is that it allows the critic to improve the accuracy of the value estimates before the actor updates its policy.

- **Target Policy Smoothing**: The algorithm adds noise to the target action during the critic updates to smooth the value function over similar actions, resulting in a lower impact of approximation errors and more robust policies.

The pseudo-code for the algorithm for TD3 is outlined in appendix A.

2.2.2.5 Soft Actor-Critic (SAC)

Soft Actor-Critic (SAC), despite being an off-policy actor-critic algorithm, represents a paradigm shift in RL. The algorithm was presented by Haarnoja et al. (2018) in their paper on *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor* [38] and is based on the maximum entropy framework, which aims to maximise both the expected return and the entropy of the policy, encouraging the agent to succeed at the task while acting as randomly as possible.

The algorithm uses two critic networks to estimate the state-action value function, and a stochastic actor network that learns a policy that maximises the expected return while also maximising the entropy of the policy. The critic networks are updated using the Bellman equations, and the actor network is updated using the policy gradient method. The objective function for the actor network is defined as:

$$J(\theta) = \mathbb{E}_{s_t \sim \mathcal{D}} \left[\mathbb{E}_{a_t \sim \pi_\theta} \left[\alpha \log \pi_\theta(a_t|s_t) + Q(s_t, a_t; \theta_Q) \right] \right] \tag{2.26}$$

where α is a temperature parameter that controls the trade-off between exploration and exploitation, and $\mathcal{D}$ is the replay buffer that stores the agent's experiences. The critic networks are updated using the Bellman equations:

$$J(\theta_Q) = \mathbb{E}_{s_t \sim \mathcal{D}} \left[\left(r_t + \gamma \min_{i=1,2} Q(s_{t+1}, \mu_\theta(s_{t+1}) + \mathcal{N}(0, \sigma); \theta_{Q_i}) - Q(s_t, a_t; \theta_{Q_i}) \right)^2 \right] \tag{2.27}$$

where $\mathcal{N}(0, \sigma)$ is the noise added to the action to encourage exploration, and θ_{Q_i} are the parameters of the two critic networks. The temperature parameter α is learned automatically by maximising the entropy of the policy, which is defined as:

$$J(\alpha) = \mathbb{E}_{s_t \sim \mathcal{D}} \left[\alpha \log \pi_\theta(a_t|s_t) \right] \tag{2.28}$$

This parameter is used to control the trade-off between the entropy and the reward terms, effectively controlling the stochasticity of the policy.

The pseudo-code for the algorithm for SAC is outlined in appendix A.

2.2.2.6 Algorithm comparison

The five algorithms presented above are among the most widely used in the field of DRL, each addressing specific challenges in policy estimation and value function approximation. A2C is an on-policy actor-critic method that reduces variance through advantage estimation, while PPO changed on-policy learning by introducing a clipped surrogate objective that ensures stable policy updates. Although DDPG pioneered continuous control through deterministic policies, TD3 addressed the overestimation bias of its predecessor via twin critics and delayed updates. SAC revolutionised the field of DRL by incorporating maximum entropy principles.

An overview of the algorithms and their key characteristics is summarised in Table 2.1.

Algorithm	Policy Type	On/Off Policy	Key Idea
A2C	Stochastic	On-Policy	Advantage Actor-Critic
PPO	Stochastic	On-Policy	Clipped Policy Updates
DDPG	Deterministic	Off-Policy	Actor-Critic + Replay buffer
TD3	Deterministic	Off-Policy	Double critics + Delayed updates
SAC	Stochastic	Off-Policy	Max-entropy RL

Table 2.1: Comparison of Deep Reinforcement Learning Algorithms

2.3 Explainable Artificial Intelligence

Explainable Artificial Intelligence (XAI) refers to a set of processes, methods and techniques that enable human users to comprehend and trust the outcomes and decisions made by Artificial Intelligence (AI) systems. The goal is to make AI algorithms more transparent, interpretable, and accountable, allowing end-users to understand the outputs of the models. This is particularly important in Deep Learning (DL) where the models are often considered black boxes due to their complexity, non-linearity and high-dimensionality, making it difficult to understand how they arrive at their decisions. With explainable systems, the benefits are numerous ranging from informed decision making to increased user adoption and better governance [44].

Although there are many possible classifications of XAI methods, they can be broadly categorised into two main categories. First, transparent algorithms are inherently understandable and interpretable, such as linear regression, decision trees and rule-based systems. Second, post-hoc explanations are methods that require the usage of an additional algorithm to clarify the decisions made by a model. Examples include saliency maps [45] and interaction data [46]. The post-hoc methods can be further sub-divided between global and local explanations. The goal of global explainability is to provide an understanding of the model's overall behaviour across an entire dataset, while local explainability focuses on explaining the individual predictions.

Another classification relates to whether the explanation method depends on the type of model it is being applied to. On the one hand, model-agnostic methods can be applied to any model regardless of its internal architecture, effectively treating all types of models as black boxes. The analysis is conducted by understanding the input-output behaviour and how small perturbations to the input impact the output. Examples include Local Interpretable Model-agnostic Explanations (LIME) [47] and SHapley Additive exPlanations (SHAP) [48]. On the other hand, model-specific methods are tailored to a

particular architecture and leverage its components as part of the explanation process. In the case of neural networks, this can be achieved by analysing the learned features or by incorporating attention mechanisms [49].

The taxonomy of the XAI methods is shown in Figure 2.5.

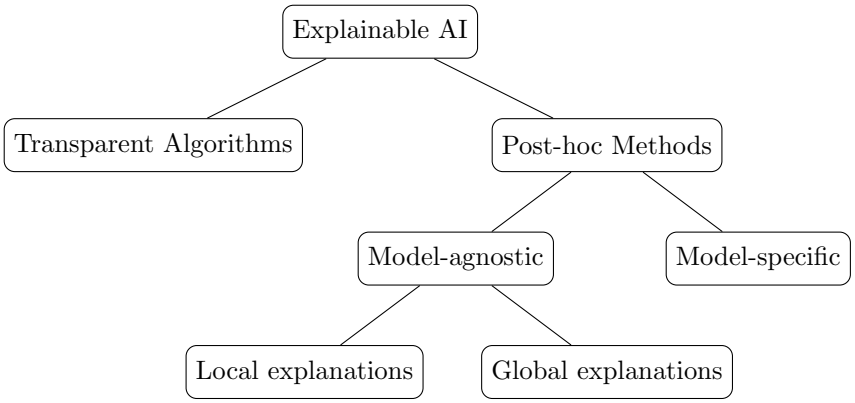

Figure 2.5: Taxonomy of Explainable Artificial Intelligence Methods

2.3.1 Feature Importance

Feature Importance is a global model-specific method that quantifies the contribution of each feature to the model's predictions. In particular, permutation feature importance [50] measures the contribution of each feature by evaluating the model's performance on the original dataset and comparing it to the performance on a permuted version of the dataset, where a specific feature is randomly shuffled. The process allows to understand how much the model's relies on a particular feature for its prediction, by measuring the decrease in predictive power if the input feature's values vary.

An alternative method, well-suited for tree-based models, is impurity-based feature importance. Random forests split their features with the goal of reducing an impurity measure at each node, normally Gini impurity for classification tasks or mean squared error for regression. As such, a split with a large decrease of impurity is considered

importance and as such the variable responsible for the split is considered important [51]. As a result, the impurity importance of a feature is the sum of all impurity reductions across all nodes and trees in the forest where the particular feature was used to split the data. While powerful, this method suffers from bias towards features with high cardinality and they do not necessarily reflect the ability of a feature to make useful predictions on the test set, given that importance is computed on the training set.

2.3.2 Local Interpretable Model-agnostic Explanations (LIME)

Local Interpretable Model-agnostic Explanations (LIME) is a model-agnostic explanation method that explains individual predictions of a Machine Learning model by analysing the model locally around the prediction of interest. The method, introduced by Ribeiro et al. (2016) in their paper on *Why should I trust you? Explaining the predictions of any classifier* [47], utilises the black box model to understand what happens to the outputs when the input data is slightly modified, then fits a simpler, interpretable model to the perturbed data to approximate the black box model's behaviour in that local region.

For a more rigorous definition, let $\mathcal{L}(f, g, \pi_x)$ be a measure of how unfaithful an explanation model g is in approximating the model $f : \mathbf{R}^d \to \mathbf{R}$ in the neighbourhood of the instance x defined as π_x. The goal is to find an interpretable model $g \in G$ that minimises the following objective function:

$$\xi(x) = \mathcal{L}(f, g, \pi_x) + \Omega(g) \tag{2.29}$$

where $\Omega(g)$ is a regularisation term that penalises the complexity of the explanation model g. This encourages interpretability, meaning a qualitative understanding of the relationship between inputs and outputs, and promotes local fidelity, ensuring the ex-

planation accurately approximates the model's behaviour in that region.

The method works as follows:

- **Instance Selection**: Start with a specific prediction instance that requires explanation.
- **Perturbation**: Generate synthetic data points by perturbing the original instance.
- **Model Querying**: Obtain predictions from the black-box model for these perturbed instances.
- **Weighting**: Assign weights to the synthetic data points based on their proximity to the original instance.
- **Surrogate Training**: Train a simple, interpretable model (typically linear regression) on the weighted synthetic dataset.
- **Explanation**: Use the surrogate model to explain the original prediction by interpreting its coefficients or structure.

Its main advantages are its ability to work across different types of data and models, due to its model-agnostic nature, and the fact that its explanations are human-friendly and include a fidelity measure that indicates how well the explanation approximates the black-box model's local behaviour. Nonetheless, the method offers only local explanations, which may not generalise well to the entire model; it relies on the choice of neighbourhood; and the complexity of the surrogate model needs to be defined in advance and can compromise the interpretability of the explanation [52].

2.3.3 SHapley Additive exPlanations (SHAP)

Another model-agnostic technique is SHapley Additive exPlanations (SHAP), which was introduced by Lundberg and Lee (2017) in their paper on *A unified approach to*

interpreting model predictions [48]. The method is based on cooperative game theory and the concept of Shapley values [53].

The Shapley value arises in cooperative game theory to answer the question of how to fairly distribute the contribution of each player in a coalition. In this framework, a coalitional game is represented as a tuple (N, v), where $N = \{1, 2, \ldots, n\}$ is the set of players and $v : 2^N \to \mathbb{R}$ is the characteristic function that assigns a value $v(S)$ to each possible coalition $S \subseteq N$, with the convention that $v(\emptyset) = 0$. The characteristic function $v(S)$ represents the total value that coalition S can achieve through cooperation.

Mathematically, the Shapley value for a feature i is defined as:

$$\phi_i = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(|N| - |S| - 1)!}{|N|!} \left[v_{S \cup \{i\}}(x_{S \cup \{i\}}) - v_S(x_S) \right] \tag{2.30}$$

where $S \subseteq N$ is a subset of all features N, $v_{S \cup \{i\}}$ is marginal contribution of player i to coalition S, and v_S is the value of coalition S without player i. This formula can be understood as computing the weighted average of marginal contributions of feature i across all possible coalitions.

The advantage of Shapley values is that it is the only attribution method that results in a fair payout. In particular, it satisfies the four fundamental properties that define a fair allocation:

- **Efficiency:** The sum of all Shapley values equals the value of the grand coalition:

$$\sum_{i \in N} \phi_i = v(N)$$

- **Symmetry:** The contributions of two players i and j should be equal if they make

identical marginal contributions to all possible coalitions:

$$v(S \cup \{i\}) = v(S \cup \{j\}), \forall S \subseteq N \setminus \{i, j\} \implies \phi_i = \phi_j$$

- **Dummy:** If a player contributes nothing to any coalition they join, their Shapley value should be zero:

$$v(S \cup \{i\}) = v(S), \forall S \subseteq N \setminus \{i\} \implies \phi_i = 0$$

- **Additivity:** For two games (N, v_1) and (N, v_2), the Shapley value of the combined game $(N, v_1 + v_2)$ equals the sum of individual Shapley values:

$$\phi_i(v_1 + v_2) = \phi_i(v_1) + \phi_i(v_2)$$

The algorithm for approximating Shapley values is outlined in 1. However, the method is computationally expensive, as it requires evaluating the model for all possible coalitions of features.

The use of Shapley values in ML was not introduced until 2011, when Štrumbelj and Kononenko [54] proposed it as a method for explaining black-box regression models. However, it was popularised in 2017 by Lundberg and Lee [48], who introduced the SHAP framework. Although their proposal relies on the principles of Shapley values, their main contribution is to represent the Shapley value as an additive feature attribution method. The explanation is given as:

$$g(\mathbf{z}') = \phi_0 + \sum_{j=1}^{M} \phi_j z'_j \tag{2.31}$$

where g is the explanation model, $\mathbf{z}' \in \{0, 1\}^M$ is the coalition vector, M is the maximum coalition size and ϕ_j is the feature attribution for feature j.

Algorithm 1 Shapley Value Approximation

Require: Number of iterations M, instance of interest $\mathbf{x}$, feature index j, data matrix $\mathbf{X}$, model $\hat{f}$

Ensure: Estimated Shapley value $\phi_j(\mathbf{x})$

for $m = 1$ to M **do**

Draw a random instance $\mathbf{z}$ from data matrix $\mathbf{X}$

Choose a random permutation $\mathbf{o}$ of the feature indices

Order $\mathbf{x}$ according to $\mathbf{o}$: $\mathbf{x_o} = \left(x_{(1)}, \ldots, x_{(j)}, \ldots, x_{(p)}\right)$

Order $\mathbf{z}$ according to $\mathbf{o}$: $\mathbf{z_o} = \left(z_{(1)}, \ldots, z_{(j)}, \ldots, z_{(p)}\right)$

Construct two new instances:

$\mathbf{x}_{+j} = \left(x_{(1)}, \ldots, x_{(j-1)}, x_{(j)}, z_{(j+1)}, \ldots, z_{(p)}\right)$

$\mathbf{x}_{-j} = \left(x_{(1)}, \ldots, x_{(j-1)}, z_{(j)}, z_{(j+1)}, \ldots, z_{(p)}\right)$

Compute marginal contribution:

$$\phi_j^{(m)} = \hat{f}(\mathbf{x}_{+j}) - \hat{f}(\mathbf{x}_{-j})$$

end for

Compute average Shapley value:

$$\phi_j(\mathbf{x}) = \frac{1}{M} \sum_{m=1}^{M} \phi_j^{(m)}$$

2.3.3.1 KernelSHAP

The KernelSHAP algorithm builds on the SHAP framework by using a kernel-based approach to estimate Shapley values. It approximates the Shapley value by sampling different coalitions and using a weighted linear regression model to fit the contributions of each feature. The method relies on not all coalitions contributing equally to the final Shapley value and as such uses kernel weights to identify the most important coalitions. The main steps of the KernelSHAP algorithm are:

- Sample a set of coalitions from the feature space: $\mathbf{z}'_k \in \{0,1\}^M, k \in \{1,\ldots,K\}$, where K is the total number of samples.
- For each coalition $\mathbf{z}'_k$, compute the model's prediction by first converting $\mathbf{z}'_k$ to the original feature space and then applying the model $\hat{f}$: $\hat{f}\left(h_{\mathbf{x}}\left(\mathbf{z}'_k\right)\right)$.
- Compute the weight for each coalition with the SHAP kernel: $\pi_{\mathbf{x}}(\mathbf{z}')$
- Fit a weighted linear regression model to the sampled coalitions and their corresponding predictions.
- Return shapley values ϕ_k as the coefficients of the fitted model.

The sample coalitions are generated by randomly selecting subsets of the features, or equivalently, a vector of 0s and 1s indicating whether a feature is included in the coalition or not. The sampled coalitions represent the dataset for the regression model whose target is the prediction for a coalition. However, the sampled coalitions are not on the target feature space and, as such, it is necessary to implement another function $h_{\mathbf{x}}\left(\mathbf{z}'\right) = \mathbf{z}$ that maps the sampled coalitions to the original feature space. In the case of tabular data, this is done by replacing the features that are not included in the coalition with their mean value or a random sample from its possible values. As a result, sampling from the marginal distribution ignores the dependence structure between features.

Although there are similarities between SHAP and LIME, their main difference lies in feature weighting in the regression model. The SHAP weighting assumes that small and large coalitions provide the most information about its isolated effects or its total effects, respectively. Whereas coalitions with half the features add little information about a specific feature contributions. The proposed weighting function for KernelSHAP is:

$$\pi_{\mathbf{x}}(\mathbf{z}') = \frac{(M-1)}{\binom{M}{|\mathbf{z}'|}|\mathbf{z}'|\,(M-|\mathbf{z}'|)} \tag{2.32}$$

where M is the maximum coalition size and $|\mathbf{z}'|$ is the number of features in the coalition $\mathbf{z}'$.

In addition, since the coalitions with the smallest and the largest number of features are the most informative, the sampling process is biased towards coalitions of size 1 and $M-1$, resulting in $2M$ possible coalitions. Then, the remaining budget $K-2M$ is used to sample coalitions of size 2 to $M-2$. This process continues until the sampled coalitions reach the desired number of samples K.

Consequently, given the samples, the KernelSHAP algorithm fits a weighted linear regression model to estimate the Shapley values. The model is defined as:

$$g(\mathbf{z}') = \phi_0 + \sum_{j=1}^{M} \phi_j z'_j \tag{2.33}$$

The goal is to minimise the following loss function:

$$\mathcal{L}(\hat{f}, g, \pi_{\mathbf{x}}) = \sum_{\mathbf{z}' \in \mathcal{Z}} \left[\hat{f}\left(h_{\mathbf{x}}\left(\mathbf{z}'\right)\right) - g(\mathbf{z}')\right]^2 \pi_{\mathbf{x}}(\mathbf{z}') \tag{2.34}$$

where $\mathbf{Z}$ is the dataset of sampled coalitions.

2.3.3.2 TreeSHAP

A variant of their original proposal is TreeSHAP [55], designed specifically for tree-based models, such as decision trees, random forests and gradient-boosted trees. In this case, the method is model-specific and by leveraging the structure of the tree, it improves the computational efficiency by reducing computation time from exponential for KernelSHAP to polynomial.

The method works by exploiting the tree structure and the main steps of the algorithm are as follows:

- For each feature, traverse the tree and compute the contribution of the feature to the prediction at each node.
- For each node, compute the contribution of the feature to the prediction by considering the difference between the prediction at the node and the prediction at the parent node.
- For each feature, compute the Shapley value by averaging the contributions across all nodes in the tree.
- Return the Shapley values as the feature importances.

Chapter 3

Methodology

This chapter covers the methodology and framework established to provide an explainable Deep Reinforcement Learning (DRL) model capable of optimising a portfolio. The chapter is structured as follows: first, it describes the architecture and components of the proposed DRL model, including the state representation, reward function and training process. Second, it discusses the evaluation metrics and experimental setup used to assess the performance of the proposed solution. Finally, it outlines the implementation of the post-hoc explainability techniques used to interpret the model's decisions.

3.1 Problem Definition

The problem of portfolio optimisation is the task of finding an optimal allocation of financial assets in a portfolio to maximise expected returns while minimising risk. Thus, it is necessary to decide how to rebalance the portfolio at each time step in a highly stochastic and complex financial market. This can be formulated using a Markov Decision Process (MDP) framework, where the agent interacts with the environment by deciding the optimal allocation based on the state of the environment at each time step

to maximise the expected cumulative reward over time. Deep Reinforcement Learning (DRL) gives the agent the ability to learn the optimal policy directly from the environment by taking actions and receiving rewards.

3.2 MDP Model

Due to the dynamic, stochastic and interactive nature of financial markets, a Markov Decision Process is a suitable framework to model the problem. The main elements of the MDP model are defined as follows:

- State space $\mathcal{S}$
- Action space $\mathcal{A}$
- Reward function $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$
- Transition function $T : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$
- Discount factor $\gamma \in [0, 1]$

The state space $\mathcal{S}$ is a vector representation of the financial environment. For a portfolio of D assets, the features that describe the state include asset prices, technical indicators and macroeconomic indicators:

- Close price $p_t \in \mathbb{R}_+^D$: Adjusted close prices of the assets at time t.
- Open price $o_t \in \mathbb{R}_+^D$: Opening prices of the assets at time t.
- High price $h_t \in \mathbb{R}_+^D$: Highest prices of the assets at time t.
- Low price $l_t \in \mathbb{R}_+^D$: Lowest prices of the assets at time t.
- Volume $v_t \in \mathbb{R}_+^D$: Trading volume of the assets at time t.

- Technical indicators $i_t \in \mathbb{R}_+^{D \times I}$: A vector of I technical indicators, such as moving averages, relative strength index (RSI), and Bollinger Bands, calculated from the asset prices.
- Macroeconomic indicators $m_t \in \mathbb{R}_+^{D \times M}$: A vector of M macroeconomic indicators, such as volatility index and interest rates, which provide additional context about the financial environment.
- Correlation matrix $C_t \in \mathbb{R}^{D \times D}$: A matrix representing the correlation between the assets in the portfolio at time t.

The description of technical and macroeconomic indicators is provided in more detail in appendix B.

The action space $\mathcal{A}$ is the set of possible actions that the agent can take at each time step. For the portfolio optimisation problem, the actions correspond to portfolio weights and are defined as follows:

$$a_t = w_t : w_t \in [0, 1]^D \tag{3.1}$$

where w_t is a vector of portfolio weights at time t, representing the allocation of the portfolio to each asset. The weights are constrained to be non-negative and sum to one:

$$\sum_{d=1}^{D} w_{t,d} = 1, \quad w_{t,d} \geq 0 \quad \forall d \in \{1, \ldots, D\} \tag{3.2}$$

Moreover, they are initialised to be equal for all assets, meaning that the agent starts with an equal allocation to each asset in the portfolio. The reason behind this is to avoid an initial bias and allowing the agent to learn an allocation from the environment rather than favour any particular asset.

The transition function T describes how the state of the environment changes in response

to the action taken. It is defined as:

$$s_{t+1} = T(s_t, a_t) \tag{3.3}$$

where s_{t+1} is the new state of the environment after taking action a_t in state s_t. The transition function is determined by the dynamics of the financial market, which are influenced by the asset prices, trading volume and other factors.

The reward function R models the direct reward of taking an action a_t in state s_t and transitioning to a new state s_{t+1}. It is defined as the change in the portfolio value from time t to time $t+1$:

$$R_{t+1} = R(s_t, a_t, s_{t+1}) = V_{t+1} - V_t \tag{3.4}$$

where the value of the portfolio at time t is given by the dot product of the portfolio weights and the asset close prices:

$$V_t = w_t \cdot p_t \tag{3.5}$$

An alternative formulation of the reward function is to use the Sharpe ratio [56], which is defined as the ratio of the expected return to the standard deviation of the returns:

$$R(s_t, a_t, s_{t+1}) = \frac{E[r_{t+1}]}{\sigma[r_{t+1}]} \tag{3.6}$$

where $E[r_{t+1}]$ is the expected return of the portfolio at time $t+1$ and $\sigma[r_{t+1}]$ is the standard deviation of the returns at time $t+1$. This formulation encourages the agent to maximise the expected return while minimising the risk of the portfolio.

Regardless of the choice of reward function, the goal of the agent is to learn a policy $\pi : \mathcal{S} \to \mathcal{A}$ that maximises the expected cumulative reward over time, which can be

expressed as:

$$J(\pi) = \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t R(s_t, a_t, s_{t+1})\right] \tag{3.7}$$

where T is the time horizon and γ is the discount factor that determines the importance of future rewards.

3.3 DRL Models

The proposed solution is based on the DRL framework, which allows the agent to learn the optimal policy directly from the environment by taking actions and receiving rewards. The algorithms used are:

- Advantage Actor-Critic (A2C)
- Proximal Policy Optimisation (PPO)
- Deep Deterministic Policy Gradient (DDPG)
- Twin Delayed Deep Deterministic Policy Gradient (TD3)
- Soft Actor-Critic (SAC)

The implementation is done using the Stable Baselines3 library [57], which provides a set of state-of-the-art DRL algorithms with a consistent interface and easy-to-use API. The pseudo-code for each algorithm is provided in appendix A.

Chapter 4

Results

Chapter 5

Legal, Social, Ethical and Professional Issues

Chapter 6

Conclusion

References

[1] Z. Bodie, A. Kane, and A. J. Marcus, *Investments*. McGraw-Hill Education, 10 ed., 2014.

[2] J. Achiam, "Part 2: Kinds of rl algorithms," 2018.

[3] J. Shen and M. O. Shafiq, "Short-term stock market price trend prediction using a comprehensive deep learning system," *Journal of Big Data*, vol. 7, pp. 1–33, 12 2020.

[4] K. Lei, B. Zhang, Y. Li, M. Yang, and Y. Shen, "Time-driven feature-aware jointly deep reinforcement learning for financial signal representation and algorithmic trading," *Expert Systems with Applications*, vol. 140, p. 112872, 2 2020.

[5] C. Ma, J. Zhang, J. Liu, L. Ji, and F. Gao, "A parallel multi-module deep reinforcement learning algorithm for stock trading," *Neurocomputing*, vol. 449, pp. 290–302, 8 2021.

[6] I. K. Nti, A. F. Adekoya, and B. A. Weyori, "A comprehensive evaluation of ensemble learning for stock-market prediction," *Journal of Big Data*, vol. 7, pp. 1–40, 12 2020.

[7] J. M. T. Wu, Z. Li, N. Herencsar, B. Vo, and J. C. W. Lin, "A graph-based cnn-lstm stock price prediction algorithm with leading indicators," *Multimedia Systems*, vol. 29, pp. 1751–1770, 6 2023.

[8] M. Hasan, M. Z. Abedin, P. Hajek, K. Coussement, M. N. Sultan, and B. Lucey, "A blending ensemble learning model for crude oil price forecasting," *Annals of Operations Research*, pp. 1–31, 1 2024.

[9] J. Moody and M. Saffell, "Learning to trade via direct reinforcement," *IEEE Transactions on Neural Networks*, vol. 12, pp. 875–889, 7 2001.

[10] H. Yang, X. Y. Liu, S. Zhong, and A. Walid, "Deep reinforcement learning for automated stock trading: An ensemble strategy," *ICAIF 2020 - 1st ACM International Conference on AI in Finance*, 10 2020.

[11] X.-Y. Liu, Z. Xiong, S. Zhong, H. Yang, and A. Walid, "Practical deep reinforcement learning approach for stock trading," *NeurIPS 2018 AI in Finance Workshop.*, 11 2018.

[12] M. Guan and X. Y. Liu, "Explainable deep reinforcement learning for portfolio management: An empirical approach," *ICAIF 2021 - 2nd ACM International Conference on AI in Finance*, 11 2021.

[13] D. G. Cortés, E. Onieva, I. Pastor, L. Trinchera, and J. Wu, "Portfolio construction using explainable reinforcement learning," *Expert Systems*, vol. 41, p. e13667, 11 2024.

[14] A. B. Arrieta, N. Díaz-Rodríguez, J. D. Ser, A. Bennetot, S. Tabik, A. Barbado, S. Garcia, S. Gil-Lopez, D. Molina, R. Benjamins, R. Chatila, and F. Herrera, "Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai," *Information Fusion*, vol. 58, pp. 82–115, 10 2019.

[15] D. Gunning, M. Stefik, J. Choi, T. Miller, S. Stumpf, and G. Z. Yang, "Xai—explainable artificial intelligence," *Science Robotics*, vol. 4, 12 2019.

[16] D. Brigo, X. Huang, A. Pallavicini, and H. S. de Ocariz Borde, "Interpretability in deep learning for finance: a case study for the heston model," *SSRN Electronic Journal*, 4 2021.

[17] R. García-Céspedes, F. J. Alias-Carrascosa, and M. Moreno, "On machine learning models explainability in the banking sector: the case of shap," *Journal of the Operational Research Society*, 2025.

[18] Y. Sato, "Model-free reinforcement learning for financial portfolios: A brief survey," *arXiv preprint arXiv:1904.04973*, 4 2019.

[19] B. Bruce and J. Greene, *Chapter 4 - Portfolio Construction*, pp. 133–178. Academic Press, 2014.

[20] X. Li, Y. Li, Y. Zhan, and X.-Y. Liu, "Optimistic bull or pessimistic bear: Adaptive deep reinforcement learning for stock portfolio allocation," *arXiv preprint arXiv:1907.01503*, 6 2019.

[21] H. Markowitz, "Portfolio selection," *The Journal of Finance*, vol. 7, pp. 77–91, 3 1952.

[22] F. M. Clinic, "Mean variance portfolio theory."

[23] IBM, "What is machine learning (ml)?," 9 2021.

[24] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, "An introduction to deep reinforcement learning," *Foundations and Trends in Machine Learning*, vol. 11, pp. 219–354, 12 2018.

[25] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[26] C. Tang, B. Abbatematteo, J. Hu, R. Chandra, R. Martín-Martín, and P. Stone, "Deep reinforcement learning for robotics: A survey of real-world successes," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 8, pp. 153–188, 8 2024.

[27] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. V. D. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature 2016 529:7587*, vol. 529, pp. 484–489, 1 2016.

[28] K. Shao, Z. Tang, Y. Zhu, N. Li, and D. Zhao, "A survey of deep reinforcement learning in video games," *arXiv preprint arXiv:1912.10944*, 12 2019.

[29] R. Bellman, "A markovian decision process," *Journal of Mathematics and Mechanics*, vol. 6, pp. 679–684, 1957.

[30] R. Bellman, *Dynamic Programming*. Princeton University Press, 1 1957.

[31] S. Thrun, "Efficient exploration in reinforcement learning," tech. rep., Carnegie Mellon University, 1 1992.

[32] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 12 2013.

[33] M. G. Bellemare, W. Dabney, and R. Munos, "A distributional perspective on reinforcement learning," *34th International Conference on Machine Learning, ICML 2017*, vol. 1, pp. 693–711, 7 2017.

[34] V. Mnih, A. P. Badia, L. Mirza, A. Graves, T. Harley, T. P. Lillicrap, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," *33rd*

International Conference on Machine Learning, ICML 2016, vol. 4, pp. 2850–2869, 2 2016.

[35] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. K. Openai, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 7 2017.

[36] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, 9 2015.

[37] S. Fujimoto, H. V. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," *35th International Conference on Machine Learning, ICML 2018*, vol. 4, pp. 2587–2601, 2 2018.

[38] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," *35th International Conference on Machine Learning, ICML 2018*, vol. 5, pp. 2976–2989, 1 2018.

[39] D. H. G. B. Tokyo and J. Schmidhuber, "Recurrent world models facilitate policy evolution," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[40] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *arXiv preprint arXiv:1712.01815*, 12 2017.

[41] Y. Wu, E. Mansimov, S. Liao, R. Grosse, and J. Ba, "Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation," *Advances in Neural Information Processing Systems*, vol. 2017-December, pp. 5280–5289, 8 2017.

[42] J. Schulman, S. Levine, P. Moritz, M. Jordan, and P. Abbeel, "Trust region policy optimization," *32nd International Conference on Machine Learning, ICML 2015*, vol. 3, pp. 1889–1897, 2 2015.

[43] G. E. Uhlenbeck and L. S. Ornstein, "On the theory of the brownian motion," *Physical Review*, vol. 36, p. 823, 9 1930.

[44] P. J. Phillips, C. A. Hahn, P. C. Fontana, A. N. Yates, K. Greene, D. A. Broniatowski, and M. A. Przybocki, "Four principles of explainable artificial intelligence," *National Institute of Standards and Technology*, vol. Internal Report 8312, 9 2021.

[45] P. Sequeira and M. Gervasio, "Interestingness elements for explainable reinforcement learning: Understanding agents' capabilities and limitations," *Artificial Intelligence*, vol. 288, p. 103367, 11 2020.

[46] S. Greydanus, A. Koul, J. Dodge, and A. Fern, "Visualizing and understanding atari agents," *Proceedings of the 35th International Conference on Machine Learning*, vol. 80, pp. 1792–1801, 7 2018.

[47] M. T. Ribeiro, S. Singh, and C. Guestrin, ""why should i trust you?": Explaining the predictions of any classifier," *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1135–1144, 8 2016.

[48] S. M. Lundberg and S. I. Lee, "A unified approach to interpreting model predictions," *Advances in Neural Information Processing Systems*, vol. 2017-December, pp. 4766–4775, 5 2017.

[49] B. Amirshahi and S. Lahmiri, "Investigating the effectiveness of twitter sentiment in cryptocurrency close price prediction by using deep learning," *Expert Systems*, vol. 42, p. e13428, 8 2023.

[50] L. Breiman, "Random forests," *Machine Learning*, vol. 45, pp. 5–32, 10 2001.

[51] S. Nembrini, I. R. König, and M. N. Wright, "The revival of the gini importance?," *Bioinformatics (Oxford, England)*, vol. 34, pp. 3711–3718, 11 2018.

[52] C. Molnar, *LIME*, ch. 14. Christoph Molnar, 3 ed., 2025.

[53] L. Shapley, *A value for n-person games*, pp. 307–317. Princeton University Press, 1953.

[54] E. Štrumbelj and I. Kononenko, "A general method for visualizing and explaining black-box regression models," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6594 LNCS, pp. 21–30, 2011.

[55] S. M. Lundberg, G. G. Erion, and S.-I. Lee, "Consistent individualized feature attribution for tree ensembles," 2019.

[56] W. Sharpe, "The sharpe ratio," *The Journal of Portfolio Management*, vol. Fall, 1994.

[57] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, pp. 1–8, 2021.

Appendix A

DRL Algorithms

Algorithm 2 Advantage Actor-Critic (A2C) Pseudo-code

Initialise:

Global shared policy parameters θ and value parameters θ_v

Number of parallel workers N

Global step counter $T \leftarrow 0$

Hyper-parameters: discount γ, max steps per update $t_{\max}$, max total steps $T_{\max}$, learning rates α_π, α_v

repeat

Reset gradients: $d\theta \leftarrow 0$, $d\theta_v \leftarrow 0$

Initialise empty batch storage for all workers

for worker $i = 1$ to N **do**

$t_{\text{start}} \leftarrow t$

Get initial state $s_t^{(i)}$ from worker i

repeat

Select action $a_t^{(i)} \sim \pi_\theta(\cdot|s_t^{(i)})$

Execute $a_t^{(i)}$, observe reward $r_t^{(i)}$ and next state $s_{t+1}^{(i)}$

Store $(s_t^{(i)}, a_t^{(i)}, r_t^{(i)})$ in worker i's trajectory

$t \leftarrow t + 1$

until terminal $s_t^{(i)}$ or $t - t_{\text{start}} == t_{\max}$

$$R^{(i)} = \begin{cases} 0 & \text{if terminal } s_t^{(i)} \\ V_{\theta_v}(s_t^{(i)}) & \text{otherwise} \end{cases}$$

for $j \in \{t-1, \ldots, t_{\text{start}}\}$ **do**

$R^{(i)} \leftarrow r_j^{(i)} + \gamma R^{(i)}$

Accumulate gradients w.r.t. θ:

$d\theta \leftarrow d\theta + \nabla_\theta \log \pi_\theta(a_j^{(i)}|s_j^{(i)})(R^{(i)} - V_{\theta_v}(s_j^{(i)}))$

Accumulate gradients w.r.t. θ_v:

$d\theta_v \leftarrow d\theta_v + \nabla_{\theta_v}(R^{(i)} - V_{\theta_v}(s_j^{(i)}))^2$

end for

end for

// Synchronous update: wait for all workers to complete

Average gradients: $d\theta \leftarrow \frac{1}{N} d\theta$, $d\theta_v \leftarrow \frac{1}{N} d\theta_v$

Update $\theta \leftarrow \theta + \alpha_\pi d\theta$, $\theta_v \leftarrow \theta_v - \alpha_v d\theta_v$

$T \leftarrow T + N \times t_{\max}$

until $T > T_{\max}$

Algorithm 3 Proximal Policy Optimization (PPO) Pseudo-code

Initialise:

Policy parameters θ_0 and value function parameters ϕ_0

Global step counter $T \leftarrow 0$

Hyper-parameters: discount γ, GAE parameter λ, clipping parameter ϵ, learning rates α_π, α_v, epochs per update K_{epochs}, minibatch size $N_{\text{minibatch}}$, loss coefficients c_1, c_2

for $k = 0, 1, 2, \ldots$ **do**

Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy π_{θ_k} in environment

Store transitions: $\{(s_t, a_t, r_t, s_{t+1}, \text{done}_t)\}$

for each trajectory τ in $\mathcal{D}_k$ **do**

Compute value estimates: $V_t = V_{\phi_k}(s_t)$

Compute TD residuals: $\delta_t = r_t + \gamma V_{t+1}(1 - \text{done}_t) - V_t$

Compute GAE advantages: $\hat{A}_t = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}$

Compute returns: $\hat{R}_t = \hat{A}_t + V_t$

end for

for epoch $e = 1$ to K_{epochs} **do**

Shuffle dataset $\mathcal{D}_k$

for each minibatch $\mathcal{B}$ in $\mathcal{D}_k$ **do**

$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)}$

$L^{\text{CLIP}}(\theta) = \frac{1}{|\mathcal{B}|} \sum_{t \in \mathcal{B}} \min\left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)$

$L^{VF}(\phi) = \frac{1}{|\mathcal{B}|} \sum_{t \in \mathcal{B}} \left(V_\phi(s_t) - \hat{R}_t\right)^2$

$L(\theta, \phi) = L^{\text{CLIP}}(\theta) - c_1 L^{VF}(\phi) + c_2 S[\pi_\theta]$

$\theta \leftarrow \theta + \alpha_\pi \nabla_\theta L^{\text{CLIP}}(\theta)$

$\phi \leftarrow \phi - \alpha_v \nabla_\phi L^{VF}(\phi)$

end for

end for

Update policy: $\theta_{k+1} = \theta$

Update value function: $\phi_{k+1} = \phi$

$T \leftarrow T + |\mathcal{D}_k|$

end for

Algorithm 4 Deep Deterministic Policy Gradient (DDPG) Pseudo-code

Initialise:

Critic network $Q_{\theta^Q}(s, a)$ and actor $\mu_{\theta^\mu}(s)$ with random weights θ^Q, θ^μ

Target networks: $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$

Replay buffer $\mathcal{B}$

Hyper-parameters: discount γ, soft update rate τ, batch size N, exploration noise process $\mathcal{N}$, learning rates α_Q, α_μ, total episodes M, steps per episode T

for episode $= 1$ to M **do**

Initialise random process $\mathcal{N}$ for action exploration

Receive initial state s_1

for $t = 1$ to T **do**

Select action $a_t = \mu_{\theta^\mu}(s_t) + \mathcal{N}_t$

Execute a_t, observe reward r_t and next state s_{t+1}

Store (s_t, a_t, r_t, s_{t+1}) in $\mathcal{B}$

Sample mini-batch $\{(s_i, a_i, r_i, s_{i+1})\}_{i=1}^N$ from $\mathcal{B}$

Compute target: $y_i = r_i + \gamma Q_{\theta^{Q'}}(s_{i+1}, \mu_{\theta^{\mu'}}(s_{i+1}))$

Update critic by minimising: $L(\theta^Q) = \frac{1}{N}\sum_i (y_i - Q_{\theta^Q}(s_i, a_i))^2$

Update actor by policy gradient:

$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q_{\theta^Q}(s_i, a)|_{a=\mu_{\theta^\mu}(s_i)} \nabla_{\theta^\mu} \mu_{\theta^\mu}(s_i)$

Update target networks:

$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$

$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$

end for

end for

Algorithm 5 Twin Delayed Deep Deterministic Policy Gradient (TD3) Pseudo-code

Initialise:

Critic networks $Q_{\theta_1}(s,a)$ and $Q_{\theta_2}(s,a)$ with random weights θ_1, θ_2

Actor policy $\mu_\phi(s)$ with random weights ϕ

Target networks: $\theta'_1 \leftarrow \theta_1$, $\theta'_2 \leftarrow \theta_2$, $\phi' \leftarrow \phi$

Replay buffer $\mathcal{B}$

Hyper-parameters: discount γ, target policy noise $\tilde{\sigma}$, noise clip c, policy delay d, exploration noise σ, update rate τ, batch size N, total steps T

for $t = 1$ to T **do**

Observe state s_t

Sample action with exploration: $a_t = \mu_\phi(s_t) + \epsilon$, $\epsilon \sim \mathcal{N}(0, \sigma)$

Execute a_t, observe reward r_t and next state s_{t+1}

Store (s_t, a_t, r_t, s_{t+1}) in $\mathcal{B}$

Sample mini-batch $\{(s_i, a_i, r_i, s_{i+1})\}_{i=1}^{N}$ from $\mathcal{B}$

Sample clipped noise: $\tilde{\epsilon} \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$

Compute target action: $\tilde{a}_{i+1} = \mu_{\phi'}(s_{i+1}) + \tilde{\epsilon}$

Compute target Q-value: $y_i = r_i + \gamma \min_{j=1,2} Q_{\theta'_j}(s_{i+1}, \tilde{a}_{i+1})$

Update each critic by minimising $L(\theta_j) = \frac{1}{N} \sum_i \big(y_i - Q_{\theta_j}(s_i, a_i)\big)^2$

if $t \bmod d = 0$ **then**

Update actor by policy gradient: $\nabla_\phi J \approx \frac{1}{N} \sum_i \nabla_a Q_{\theta_1}(s_i, a) \mid_{a=\mu_\phi(s_i)} \nabla_\phi \mu_\phi(s_i)$

Update target networks:

$\theta'_j \leftarrow \tau\, \theta_j + (1-\tau)\, \theta'_j$ for $j = 1, 2$

$\phi' \leftarrow \tau\, \phi + (1-\tau)\, \phi'$

end if

end for

Algorithm 6 Soft Actor-Critic (SAC) Pseudo-code

Initialise:

Actor network $\pi_\theta(a|s)$ with parameters θ

Two critic networks $Q_{\phi_1}(s,a)$ and $Q_{\phi_2}(s,a)$ with parameters ϕ_1, ϕ_2

Target critic networks: $\phi_1' \leftarrow \phi_1$, $\phi_2' \leftarrow \phi_2$

Replay buffer $\mathcal{D}$

Hyper-parameters: discount γ, temperature α (fixed or learnable), target entropy $\mathcal{H}$ (if α is learnable), batch size N, learning rates $\lambda_Q, \lambda_\pi, \lambda_\alpha$, soft update rate τ

for each training step **do**

Observe state s_t

Sample action: $a_t \sim \pi_\theta(\cdot|s_t)$

Execute a_t, observe reward r_t and next state s_{t+1}

Store (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$

if time to update **then**

Sample mini-batch $\{(s_i, a_i, r_i, s_{i+1})\}_{i=1}^N$ from $\mathcal{D}$

Update Critics:

for $j = 1, 2$ **do**

Sample next actions: $\tilde{a}_{i+1} \sim \pi_\theta(\cdot|s_{i+1})$

Compute target Q-values:

$y_i = r_i + \gamma \left(\min_{k=1,2} Q_{\phi_k'}(s_{i+1}, \tilde{a}_{i+1}) - \alpha \log \pi_\theta(\tilde{a}_{i+1}|s_{i+1}) \right)$

Update critic: $\phi_j \leftarrow \phi_j - \lambda_Q \nabla_{\phi_j} \frac{1}{N} \sum_i (Q_{\phi_j}(s_i, a_i) - y_i)^2$

end for

Update Actor:

Sample actions with reparametrisation: $\tilde{a}_i = f_\theta(\epsilon_i; s_i)$ where $\epsilon_i \sim \mathcal{N}(0, I)$

Compute policy loss:

$J(\theta) = \frac{1}{N} \sum_i \left[\alpha \log \pi_\theta(\tilde{a}_i|s_i) - \min_{j=1,2} Q_{\phi_j}(s_i, \tilde{a}_i) \right]$

Update actor: $\theta \leftarrow \theta - \lambda_\pi \nabla_\theta J(\theta)$

if α is learnable **then**

Update Temperature:

$J(\alpha) = \frac{1}{N} \sum_i \alpha \left(\log \pi_\theta(\tilde{a}_i|s_i) + \mathcal{H} \right)$

Update temperature: $\alpha \leftarrow \alpha - \lambda_\alpha \nabla_\alpha J(\alpha)$

end if

Update Target Networks:

$\phi_j' \leftarrow \tau \phi_j + (1 - \tau) \phi_j'$ for $j = 1, 2$

end if

end for

Appendix B

State Representation

B.1 Technical Indicators

The following technical indicators are used to represent the state of the financial environment. They are calculated based on the historical price data of the assets in the portfolio:

B.2 Macroeconomic Indicators

The following macroeconomic indicators are used to represent the state of the financial environment. They provide additional context about the market conditions and are calculated based on external data sources: