Helvetica

University of Zurich[UZH]

UZH Blockchain Center

# A Deep Reinforcement Learning approach with Explainability for cryptocurrency trading

Master Thesis

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Informatics


Author

**MANASWI MONDOL**

address

Matriculation Number: **21-744-438**

Email: MANASWI.MONDOL@UZH.CH

Supervisor

Mostafa Chegenizadeh

Prof. Dr Claudio J. Tessone

Blockchain & Distributed Ledger Technologies

Department of Informatics

University of Zurich



Date of Submission: 27.10.2024

## Abstract

The highly volatile and rapidly evolving cryptocurrency market presents unique challenges for Artificial Intelligence-based automated trading systems. In this study, four Deep Reinforcement Learning (DRL) algorithms—namely Deep Q-Network (DQN), Advantage Actor-Critic (A2C), Proximal Policy Optimization (PPO), and Deep Deterministic Policy Gradient (DDPG)—are implemented, taking into account the nuances of the cryptocurrency market (BTC and ETH) to develop robust trading systems. The models are trained on comprehensive datasets containing OHLCV data, key technical indicators, and blockchain metrics. The addition of fractional trading functionality to the DRL models improves risk management and facilitates learning a more efficient trading strategy. The results demonstrate that DDPG outperforms the other DRL algorithms in terms of average profit, Sharpe ratio, and trading efficiency, with significant gains realized when blockchain metrics are incorporated alongside traditional OHLCV data in the decision-making process. When compared to other studies in this field, the DDPG model significantly outperformed the existing next-best model in terms of return metrics. The DDPG model achieved an average return of 8.99% and an annual return of 107.95%, significantly outperforming the existing next-best model, which yielded an average return of 2.36% and an annual return of 28.26%. Another significant contribution of this study is the incorporation of Explainable AI (XAI) techniques, namely saliency maps and integrated gradients, to provide post-hoc explainability and to better understand the decision-making processes of the DRL models. The XAI layer reveals the most influential features guiding the models' decisions, offering valuable insights into the behavior of the DRL models. This research advances the field of automated cryptocurrency trading by demonstrating that DRL models, coupled with blockchain data can yield superior performance and provide transparency for traders and investors.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

In recent years, Blockchain and Cryptocurrency have emerged as a highly volatile and rapidly evolving domain, offering new opportunities for automated trading strategies powered by cutting-edge technologies. Cryptocurrencies are based on blockchain technology which is decentralized and enables transactions without intermediaries, making them resistant to interference from government or other organizations. Their value is driven by market demand and has recently surged, attracting significant investment from private and institutional investors (Kumlungmak and Vateekul 2023). This also makes cryptocurrencies significantly more volatile than traditional financial assets like stocks, bonds, ETFs etc. Traditional approaches to financial trading often struggle to keep pace with the high volatility and unique market dynamics of the cryptocurrency market. As a result, Machine Learning and Artificial Intelligence (AI) have been increasingly employed to predict cryptocurrency price movements, as indicated by a notable rise in recent publications (Yang et al. 2020). In this context, Deep Reinforcement Learning (DRL) has emerged as a powerful tool for developing automated trading systems that can dynamically adapt to changing market conditions (Kumlungmak and Vateekul 2023, Kochliaridis et al. 2023). Unlike static rule-based systems, DRL algorithms can learn from market data through trial and error, continuously refining their strategies based on the rewards received, enabling them to make optimal decisions (buy, sell, or hold an asset) in the cryptocurrency market. Among the popular DRL algorithms, Deep Q-Network (DQN), Proximal Policy Optimization (PPO), Advantage Actor Critic (A2C), and Deep Deterministic Policy Gradient (DDPG) have shown particular promise in financial applications due to their ability to handle high-dimensional state spaces and both discrete and continuous action spaces. These algorithms can be specifically designed for cryptocurrency trading, where the lack of traditional financial data like earnings reports and the extreme price volatility demand more advanced decision-making processes and frameworks (Kumlungmak and Vateekul 2023, Lucarelli and Borrotti 2019, S. Wang and Klabjan 2023, Kumlungmak and Vateekul 2023, Gort et al. 2022, Schnaubelt 2022). Explainable Artificial Intelligence (XAI) is an emerging field of research and an important subtopic of Artificial Intelligence. Explainable AI (XAI) is crucial in sensitive domains like autonomous driving, medical diagnosis, finance and

security, where understanding how AI systems arrive at certain decisions is essential (Heuillet et al. 2020, Arrieta et al. 2019,Das and Rad 2020). Deep Learning including Deep Reinforcement Learning models is often considered to be a black box due to their complex nature, making it challenging for users to understand and sometimes even trust the outputs. Lack of transparency can lead to doubt and rejection when the systems produce unexpected results. Additionally, interpretability is crucial for ensuring reliability, transparency and avoiding biases in deep learning models which is especially applicable to the prediction of cryptocurrency prices (Das and Rad 2020). Considering this factor, an additional layer of XAI has been added to the best-performing DRL model implemented and tested in this paper. This additional layer of XAI is considered to be post-hoc explainability where after the model has been trained and tested, the results are explained. In this paper, the aspect of determining the key and relevant features in the dataset which play a crucial role in assisting the DRL agent to make its decision has been prioritised.

The contributions of this paper are as follows:

- In this study four of the most commonly used and best-performing DRL algorithms, namely Deep Q-Network (DQN), Proximal Policy Optimization (PPO), Advantage Actor-Critic (A2C), and Deep Deterministic Policy Gradient (DDPG), have been implemented from the ground up while taking into account the nuances of the cryptocurrency market (BTC and ETH) to determine and compare their effectiveness and performance.
- Comprehensive datasets containing OHLCV data, key technical indicators, and relevant blockchain metrics were created for both BTC and ETH to train the DRL algorithms
- The four DRL models were trained and tested using these datasets across 100, 200, and 500 episodes, and key performance metrics including Average Profit (%), Maximum Profit (%), Annual Profit (%), Sharpe Ratios, and Average Trades were evaluated to determine the best-performing model at different levels of training.
- Each DRL model was equipped with the ability to conduct fractional trades, allowing the system to manage risks, diversify actions, and learn more comprehensive trading strategies for the assets being traded. The models were also tested across a varying range of fractional trade percentages to determine the optimal value for each system.
- The best-performing DRL model was tested using only OHLCV data to highlight and provide conclusive evidence of the importance of incorporating blockchain metrics when making trading decisions.
- The best-performing model was compared against other notable papers in this domain to find the overall best system. This comparative analysis revealed that the best model in this study significantly outperforms the next best system in terms of return metrics.

- A layer of Explainable AI was added to the best-performing model to investigate the key and relevant features in the dataset prioritised by the DRL agent during the decision-making process. The insights from the XAI layer can be utilized to further improve the performance of the models.

The remainder of this paper is organized as follows: Section 2 reviews related work on DRL-based trading systems and their application in both cryptocurrency and stock markets. Additionally, Section 2 discusses the current state of Explainable AI (XAI) applicability in the field of DRL. Section 3 outlines the relevant and necessary background information on key topics such as blockchain, Bitcoin, Ethereum, Reinforcement Learning (RL), Deep Reinforcement Learning (DRL), and XAI. Section 4 describes the data collection and processing methods, along with a detailed explanation and specifications of all the DRL algorithms implemented in this study. Section 4 also explains the implementation details of the XAI methods used, as well as the key performance metrics employed to evaluate the performance of each model. Section 5 evaluates the performance of all four DRL models and interprets the results from the XAI layer. Additionally, the performance of the best model is compared with results from other papers. Section 6 examines the trends observed during the experimental phase and highlights key insights. Additionally, section 6 also discusses future research directions. Finally, Section 7 concludes the paper with a summary of the key results.

# Chapter 2

# Related Work

## 2.1 Deep Reinforcement Learning

### 2.1.1 Deep Reinforcement Learning in Cryptocurrency

Reinforcement Learning was predominantly used for video games and robotics (Mnih, Kavukcuoglu, et al. 2013). In recent years it has been increasingly used in Quantitative Finance which utilizes mathematical models and data-driven techniques to analyze the financial market. New models and techniques related to Reinforcement Learning are being developed to tackle problems in the domain of Quantitative trading (Sun et al. 2023). Due to the high volatility in cryptocurrency markets, Artificial Intelligence and machine learning are frequently being used to solve various problems and provide a better chance at making profitable trades with the assets. Reinforcement Learning has shown promising results in the context of high-volatility cryptocurrency markets. Starting with portfolio management which is a key aspect of asset management. Studies indicate that deep reinforcement learning can help select and optimize portfolio selection strategies that maximize profit and reduce risk even in the highly volatile cryptocurrency market (Jiang and Liang 2017). Moving on from portfolio management to the actual prediction of cryptocurrency prices showcases the complexity of the task at hand. Many studies have approached this problem from different perspectives. One of those perspectives is the optimal placement and execution of limit orders on cryptocurrency exchanges. In the context of Deep Reinforcement Learning, a specific environment and an appropriate reward function that is tailor-made for the task and the environment can be developed. This study by (Schnaubelt 2022) demonstrates the first large-scale application of deep reinforcement learning (DRL) for optimizing limit order placements in cryptocurrency exchanges. Using a virtual limit order exchange for training and evaluation, the study rewards agents based on realized shortfall over time. The researchers utilize 18 months of high-frequency data, including 300 million trades and 3.5 million order book states, to empirically compare state-of-the-art DRL algorithms with benchmarks. Proximal Policy Optimization (PPO) is found to outperform Deep Double Q-Networks (DDQN) and

other benchmarks, effectively learning superior order placement strategies. The key features identified for predicting the prices include current liquidity costs and queue imbalances, with the latter predicting short-term mid-price returns. The results from this paper suggest a preference for aggressive order placements to avoid additional fees and anticipate unfavourable price movements (Schnaubelt 2022). A similar study with the same objective of finding optimal trading points for cryptocurrencies was conducted using Reinforcement Learning. This study also showed promising results further highlighting the effectiveness of Reinforcement Learning in this domain (Sattarov et al. 2020). This study by (Lucarelli and Borrotti 2019) evaluates the performance of trading systems based on Deep Reinforcement Learning (DRL) using hourly cryptocurrency prices. The study compares systems based on Double and Dueling Double Deep Q-learning Networks with a simpler Deep Q-learning Network, testing them with different reward functions like Sharpe Ratio and profits earned. Positive returns were observed for shorter trading periods, with systems based on Double Q-learning and the Sharpe ratio reward function showing higher returns. However, the study has limitations such as including broader performance indicators, comparison with recent AI approaches, and any kind of parameter optimization. Moreover, the paper also suggests future work that could explore incorporating sentiment analysis into DRL models (Lucarelli and Borrotti 2019). Another notable study by (Gort et al. 2022) addresses the challenge of designing robust and profitable trading strategies in the volatile cryptocurrency market, focusing on mitigating the issue of backtest overfitting when applying deep reinforcement learning (DRL). Acknowledging the tendency for DRL models to perform well in backtesting but fail in real-world scenarios due to overfitting, the authors propose a practical solution. They frame the detection of overfitting as a hypothesis test and develop a methodology to train DRL agents while estimating and controlling the probability of overfitting. By rejecting overfitted agents, the approach aims to enhance the likelihood of achieving reliable trading performance. Evaluations conducted on 10 cryptocurrencies during a testing period marked by significant market crashes demonstrate that less overfitted DRL agents outperform more overfitted ones, an equal weight strategy, and a market benchmark like the S&P DBM Index. These findings suggest promising prospects for deploying less overfitted DRL strategies in real-world cryptocurrency trading environments (Gort et al. 2022). This study by (Kochliaridis et al. 2023) addresses the challenges of developing successful cryptocurrency trading strategies by combining deep reinforcement learning (DRL) approaches with rule-based safety mechanisms to maximize profit and loss (PNL) returns while minimizing trading risks. Despite the popularity of technical analysis and its integration with machine learning, creating effective trading strategies is quite difficult. DRL algorithms have recently shown promising results in formulating profitable strategies but often lack risk-awareness. The method proposed in this study first trains a DRL agent with a unique reward function that maximizes PNL returns. During the exploitation phase, a rule-based mechanism is introduced to prevent uncertain actions, and a novel safety mechanism considers the actions of a more conservatively trained agent to avoid high-risk trading periods. Experiments on five popular cryptocurrencies demonstrate that integrating these methods yields promising results, enhancing both profitability and risk man-

agement (Kochliaridis et al. 2023). This recent study by (S. Wang and Klabjan 2023) proposes an ensemble method to enhance the generalization performance of trading strategies trained using deep reinforcement learning algorithms in the highly volatile environment of intraday cryptocurrency portfolio trading. By adopting a model selection method that evaluates multiple validation periods and introducing a novel mixture distribution policy to effectively ensemble the selected models, the approach aims to improve robustness in evolving market conditions. The paper provides a distributional view of out-of-sample performance on granular test periods, demonstrating the strategies' resilience, and periodically retrains the models to address the non-stationary aspect of financial data. The proposed ensemble method shows improved out-of-sample performance compared to benchmarks, including a standard deep reinforcement learning strategy and a passive investment strategy (S. Wang and Klabjan 2023). All the studies mentioned so far approach the problem of deep reinforcement learning and cryptocurrency price prediction through a single-agent DRL structure and approach. This introduces the question of whether a multi-agent approach would be more effective. This study by (Kumlungmak and Vateekul 2023) introduces a new approach to cryptocurrency trading using multi-agent proximal policy optimization (MAPPO) to address challenges posed by market volatility, especially during bearish periods. Existing single-agent techniques have limitations in effectively managing losses during market downturns. The method proposed in this study employs a collaborative multi-agent scheme and a local-global reward function to optimize both individual and collective agent performance. It utilizes multi-objective optimization and a multi-scale continuous loss (MSCL) reward to train agents, with a progressive penalty mechanism to prevent consecutive portfolio value losses. Comparative evaluation against baseline methods demonstrates superior cumulative returns, particularly during bearish market conditions where only the proposed method achieves profitability. Compared to a FinRL-Ensemble, the proposed approach in this study achieves significantly higher cumulative returns in bullish market scenarios (Kumlungmak and Vateekul 2023).

### 2.1.2 Deep Reinforcement Learning in Stock Market

All the studies reviewed so far have primarily focused on the application of Deep Reinforcement Learning (DRL) in the cryptocurrency domain. However, DRL has long been applied in both quantitative and traditional finance, particularly for tasks like automated stock trading and predicting stock market prices and trends. To gain a broader understanding of DRL's application to financial data, some key and relevant papers in the field of DRL and stock markets will be explored. This study by (Wu et al. 2020) addresses the challenges of stock market complexity and volatility by proposing adaptive stock trading strategies using deep reinforcement learning (DRL). Utilizing Gated Recurrent Unit (GRU) for extracting financial features from time-series data, the study introduces two strategies: Gated Deep Q-learning (GDQN) and Gated Deterministic Policy Gradient (GDPG). These strategies are designed with tailored state and action spaces. Experimental results, tested in both trending and volatile markets, indicate that GDQN and GDPG outperform the Turtle trading strategy and provide more stable returns than other DRL methods. GDPG, utilizing an

actor-critic framework, is found to be more stable compared to GDQN (Wu et al. 2020). This paper by (Yuming Li et al. 2020) explores the application of deep reinforcement learning (DRL) in stock trading and forecasting, aiming to enhance the acquisition of practical trading signals to maximize benefits. Experimental data validate the model's reliability and advantages over traditional models. The study demonstrates the feasibility and effectiveness of DRL in financial markets, highlighting its superiority in strategic decision-making for stock market forecasting and trading (Yuming Li et al. 2020). In this study by (Deng et al. 2017) propose a recurrent deep neural network integrating deep learning (DL) and reinforcement learning (RL) for enhancing financial trading strategies. The DL component autonomously extracts informative features from dynamic market conditions, crucial for informed trading decisions, while the RL module optimizes these decisions to maximize cumulative rewards in unpredictable market environments. Their approach leverages a sophisticated neural network architecture with deep and recurrent layers to handle temporal dependencies in financial data. They also introduce a task-aware backpropagation through time method to mitigate training challenges and validate their model's robustness across diverse market conditions, including stocks and commodity futures (Deng et al. 2017). This study by (Yang Li et al. 2019) introduces a novel approach to algorithmic trading using deep reinforcement learning (DRL) to tackle the challenges of feature extraction and adaptive strategy design in dynamic financial markets. Unlike traditional methods that rely on manually engineered features and static strategies, the proposed trading agent autonomously learns to make profitable decisions resembling human-like adaptability. Building upon deep Q-network (DQN) and asynchronous advantage actor-critic (A3C) frameworks, the agent incorporates Stacked Denoising Autoencoders (SDAEs) and long short-term memory (LSTM) networks to extract robust market representations and manage temporal dependencies in financial data. Practical enhancements including position-controlled actions and n-step rewards are implemented to enhance the agent's effectiveness in real-world trading environments. Experimental results across both stock and futures markets demonstrate superior performance over baseline approaches, achieving stable and risk-adjusted returns, thereby showcasing the potential of DRL in advancing algorithmic trading strategies (Deng et al. 2017). This study by (Théate and Ernst 2021) introduces an innovative application of deep reinforcement learning (DRL) to address the challenge of optimal trading position determination in algorithmic trading within stock markets. The proposed Trading Deep Q-Network algorithm (TDQN) adapts the principles of the well-known Deep Q-Network (DQN) to optimize the Sharpe ratio performance indicator across diverse stock market environments. Unlike traditional approaches, TDQN trains its reinforcement learning agent exclusively on artificial trajectories generated from a limited historical dataset, tailored to mimic real-world trading dynamics. The paper also introduces a novel performance assessment methodology to rigorously evaluate trading strategy effectiveness. Results indicate promising performance for the TDQN algorithm, showcasing its potential to advance algorithmic trading strategies through DRL innovation (Théate and Ernst 2021). This study by (Guan and Liu 2021) presents an empirical approach to explain the strategies of deep reinforcement learning (DRL) agents in portfolio management, addressing the challenge of the black-box

nature of deep neural networks. The method involves using a linear hindsight model as a reference, which calculates optimal portfolio weights based on actual stock returns, providing baseline feature weights. For the DRL agents, integrated gradients are employed to determine feature weights by measuring the coefficients between rewards and features within a linear regression model. The study examines prediction power in single-step and multi-step scenarios by assessing the linear correlations between DRL agents' feature weights and those of the hindsight model. Evaluating this approach on the Dow Jones 30 constituent stocks from 01/01/2009 to 09/01/2021, the findings reveal that DRL agents demonstrate stronger multi-step prediction power compared to traditional machine learning methods, offering deeper insights into their decision-making processes (Guan and Liu 2021). This study by (Zhang et al. 2019) explores the use of Deep Reinforcement Learning (DRL) to develop trading strategies for continuous futures contracts. The authors apply DRL algorithms to both discrete and continuous action spaces, incorporating volatility scaling to adjust trade positions based on market volatility. The algorithms are tested on 50 of the most liquid futures contracts across multiple asset classes, including commodities, equity indices, fixed income, and FX markets, from 2011 to 2019. The performance of these algorithms is compared to traditional time-series momentum strategies. The results show that the DRL-based strategies outperform the baseline models, delivering consistent profits even when accounting for high transaction costs. The algorithms are particularly effective in capturing large market trends without frequent position changes, and they can scale down or hold positions during periods of market consolidation (Zhang et al. 2019). This study by (Yang et al. 2020) presents a novel ensemble strategy for stock trading that leverages deep reinforcement learning (DRL) to maximize investment returns and is one of the most frequently cited papers in this field. The strategy in this study combines three actor-critic-based algorithms: Proximal Policy Optimization (PPO), Advantage Actor-Critic (A2C), and Deep Deterministic Policy Gradient (DDPG). By integrating the strengths of these algorithms, the ensemble approach adapts robustly to varying market conditions. To address the challenge of high memory consumption associated with training in continuous action spaces, the authors employ a load-on-demand technique for efficient data processing. The method is tested on the 30 Dow Jones stocks, ensuring liquidity and its performance is compared with both the Dow Jones Industrial Average index and a traditional min-variance portfolio allocation strategy. The results demonstrate that the ensemble strategy outperforms the individual algorithms and baselines, achieving better risk-adjusted returns, as measured by the Sharpe ratio (Yang et al. 2020).

Many studies so far have suggested that a multimodal approach might be more effective for the prediction of stock market and cryptocurrency prices so it's worthwhile to explore three of the most widely cited and relevant studies in this area. In this first study by (Chen and Huang 2021), a novel multimodal reinforcement trading system is introduced to address the limitations of previous studies in investment decision-making. The system combines reinforcement learning, sentiment analysis, and multimodal learning techniques to incorporate both price fluctuations and news information into trad-

ing decisions. An influence model that captures the relationship between news sentiment and market impact over time is proposed in this study. Experimental results demonstrate that multimodal agents outperform price-focused agents by a significant margin, highlighting the effectiveness of incorporating news sentiment into trading strategies. The influence model is also effective in shaping the impact of news on the market, aiding the RL agents in evaluating market conditions. The robustness of the proposed system is validated across different sectors and evaluation metrics, making it applicable for advanced research and practical financial applications. Moreover, since the data used in this study are publicly available, investors can readily implement the findings of the study to enhance their trading strategies and potentially increase profits while trading (Chen and Huang 2021). This second study by (Koratamaddi et al. 2021) addresses the challenge of stock portfolio allocation by proposing a novel deep reinforcement learning approach that incorporates market sentiment alongside historical stock price data. While existing methods have shown promise in automating portfolio allocation, they often overlook the influence of investor and public sentiment, which can significantly impact decisions. By training an RL agent on both historical data and market sentiment, the approach developed in this study aims to enhance the robustness of portfolio allocation strategies. The effectiveness of this approach is demonstrated using standardized metrics such as the Sharpe ratio and annualized investment return, showing superior performance compared to existing baselines (Koratamaddi et al. 2021). Even though these studies are conducted on the stock market, the methodology should be transferable to the cryptocurrency market. This third study by (Nan et al. 2020) presents a novel reinforcement learning approach for algorithmic trading, aiming to overcome the limitations of traditional methods that struggle with the numerous variables and lack of reliable labelled data in markets. By combining traditional time series stock price data with sentiment analysis of news headlines and leveraging knowledge graphs to capture implicit relationships in the news, the proposed method enhances the learning of effective trading policies. This integrated approach addresses the complexity of market dynamics and aims to provide more reliable automated trading algorithms (Nan et al. 2020).

Most of the studies explored so far use a FinRl ensemble as a benchmark to compare the results of their experiments and use it as the measurement of success and failure. This study by (Liu et al. 2021) introduces FinRL, an open-source framework designed to facilitate the application of deep reinforcement learning (DRL) in quantitative finance. FinRL aims to simplify the development process for quantitative traders by providing a full pipeline solution. It features a three-layer architecture with modular structures, implementing state-of-the-art DRL algorithms and common reward functions while reducing debugging efforts. The framework enables users to design trading strategies and offers various training environments using historical data and live trading APIs. FinRL is highly extensible, allowing users to incorporate trading constraints and providing step-by-step tutorials for typical trading tasks. Overall, FinRL aims to make DRL more accessible and applicable in quantitative finance by providing simplicity, applicability, and extensibility under key principles such as full-stack framework, customization, reproducibility, and hands-on tutoring (Liu et al. 2021).

The table below table 2.1 summarises and compares the different DRL algorithms used, modality, type of market and the evaluation metrics discussed in all the papers explored in this section.

| No | Author | Description | Algorithm | Modality | Market | Evaluation Metrics |
|---|---|---|---|---|---|---|
| 1 | (Lucarelli and Borrotti 2019) | A Deep Reinforcement Learning Approach for Automated Cryptocurrency Trading. | Deep Q-Networks (DQNs), Double Deep Q-Networks (D-DQNs), Dueling Double Deep Q-Networks (DD-DQNs) | OHCLV data | Cryptocurrency - Bitcoin | Average return, Max return, Min return and St. dev. |
| 2 | (Sattarov et al. 2020) | Recommending Cryptocurrency Trading Points with Deep Reinforcement Learning Approach | Deep Reinforcement Learning (DRL) | OHCLV data with technical indicators | Cryptocurrency - Litecoin, Monero | Average return |
| 3 | (Kumlungmak and Vateekul 2023) | Multi-Agent Deep Reinforcement Learning With Progressive Negative Reward for Cryptocurrency Trading | Multi-agent proximal policy optimization (MAPPO) | OHCLV data | Cryptocurrency - Bitcoin | Cumulative return, Sharpe Ratio, Calmar Ratio, Volatility |
| 4 | (Yang et al. 2020) | Deep Reinforcement Learning for Automated Stock Trading: An Ensemble Strategy | Advantage Actor Critic (A2C), Deep Deterministic Policy Gradient (DDPG), Proximal Policy Optimization (PPO) | OHCLV data | Stock Market | Cumulative return, Annualized return, Annualized volatility, Sharpe ratio, Max Drawdown |
| 5 | (Chen and Huang 2021) | Sentiment-influenced trading system based on multimodal deep reinforcement learning | Multimodal deep recurrent neural network | OHCLV data with sentiment analysis | Stock Market | Mean square error (MSE), Mean absolute percentage error (MAPE), and R squared (R2) |
| 6 | (Koratamaddi et al. 2021) | Market sentiment-aware deep reinforcement learning approach for stock portfolio allocation | Deep Deterministic Policy Gradients (DDPG), Adaptive Deep Deterministic Policy Gradients (Adaptive DDPG) | OHCLV data with sentiment analysis | Stock Market | Annualized rate of return |
| 7 | (Schnaubelt 2022) | Deep reinforcement learning for the optimal placement of cryptocurrency limit orders | Proximal Policy Optimization (PPO) and deep double Q-networks | OHCLV data | Cryptocurrency - Bitcoin, Ethereum | Realized shortfall and Total Shortfall |
| 8 | (Wu et al. 2020) | Adaptive stock trading strategies with deep reinforcement learning methods | Gated Deep Q-learning (GDQN) and Gated Deterministic Policy Gradient (GDPG) | OHCLV data and technical indicators | Stock Market | Realized return and Sharpe Ratio |
| 9 | (Yuming Li et al. 2020) | Application of deep reinforcement learning in stock trading strategies and stock forecasting | Deep Q network, Double Deep Q network, Dueling Deep Q network | OHCLV data | Stock Market | Profit on each stock |
| 10 | (Deng et al. 2017) | Deep Direct Reinforcement Learning for Financial Signal Representation and Trading | Deep Reinforcement Learning | OHCLV data | Stock Market | Sharpe Ratio and Total Profit |
| 11 | (Yang Li et al. 2019) | Deep Robust Reinforcement Learning for Practical Algorithmic Trading | Deep Q-network (DQN) and asynchronous advantage actor-critic (A3C) | OHCLV data | Stock Market | Cumulative Profit and Sharpe Ratio |
| 12 | (Zhang et al. 2019) | Deep Reinforcement Learning for Trading | Deep Q-learning Networks (DQN), Policy Gradients (PG), Advantage Actor-Critic (A2C) | OHCLV data and technical indicators | Stock Market | Sharpe ratio, Sortino Ratio, Percentage of positive trade return, ratio between positive and negative trade returns |
| 13 | (Jiang and Liang 2017) | Cryptocurrency Portfolio Management with Deep Reinforcement Learning | Deep Reinforcement Learning | OHCLV data | Cryptocurrency – Top 12 coins | Sharpe Ratio, Maximum drawdown |
| 14 | (Liu et al. 2021) | FinRL: A Deep Reinforcement Learning Library for Automated Stock Trading in Quantitative Finance | All D Deep Reinforcement Learning algorithms | OHCLV data and technical indicators | Stock Market | All standard evaluation baselines |
| 15 | (Théate and Ernst 2021) | An application of deep reinforcement learning to algorithmic trading | Trading Deep Q-Network algorithm (TDQN) | OHCLV data | Stock Market | Sharpe Ratio, expected daily return |
| 16 | (Gort et al. 2022) | Deep Reinforcement Learning for Cryptocurrency Trading: Practical Approach to Address Backtest Overfitting | Proximal policy optimization (PPO), Twin Delayed DDPG, (TD3) Soft Actor Critic (SAC) | OHCLV data and technical indicators | Cryptocurrency – Top 10 coins | Sharpe Ratio, Cumulative return |
| 17 | (Nan et al. 2020) | Sentiment and Knowledge Based Algorithmic Trading with Deep Reinforcement Learning | Deep Q-Network (DQN) | OHCLV data with sentiment analysis | Stock Market | Sharpe Ratio |
| 18 | (Guan and Liu 2021) | Explainable Deep Reinforcement Learning for Portfolio Management: An Empirical Approach | Advantage Actor Critic (A2C) and Proximal Policy Optimization (PPO) | OHCLV data and technical indicators | Stock Market | Cumulative return, Annualized return, Annualized volatility, Sharpe ratio, Calmar ratio, Average Correlation Coefficient |
| 19 | (S. Wang and Klabjan 2023) | An Ensemble Method of Deep Reinforcement Learning for Automated Cryptocurrency Trading | Long Short-Term Memory Network (LSTM) and Proximal Policy Optimization (PPO) | OHCLV data and technical indicators | Cryptocurrency – Top 5 coins | Annualized return, cumulative returns, Sharpe ratio, and Calmar ratio |
| 20 | (Kochliaridis et al. 2023) | Combining deep reinforcement learning with technical analysis and trend monitoring on cryptocurrency markets | Actor Critic based Proximal Policy Optimization (PPO) | OHCLV data and technical indicators | Cryptocurrency – Top 5 coins | Sharpe ratio, Sortino Ratio, Maximum drawdown, investment risk, cumulative returns and cumulative PNL |

**Table 2.1** – Comparison of Deep Reinforcement Learning approaches in trading systems (Cryptocurrency and Stocks).

## 2.2 Explainable Artificial Intelligence - XAI

Explainable Artificial Intelligence (XAI) itself is an emerging field so there is not an abundance of studies that investigate the application of XAI to Reinforcement Learning. The following studies have reviewed and summarized the current approaches and significance of XAI in Reinforcement Learning. The first paper examines the significance of XAI as Artificial Intelligence becomes more prevalent, focusing on the ethical, trust, transparency, and safety concerns associated with AI. The review focuses on XAI within Reinforcement Learning (RL) and indicates that the field is still in its early stages. The most used approaches for XAI in RL include human collaboration, visualization techniques, policy

summarization explanations, query-based explanations, and verification approaches. The paper also identifies limitations such as lack of detail in human experiments, scalability issues, and challenges in comprehension for non-experts. The paper suggests that combining clear explanations with advanced visualization techniques is crucial for overcoming the black-box nature of RL and improving understandability and transparency (Arrieta et al. 2019). The second paper examines state-of-the-art approaches in Reinforcement Learning (RL) and their integration with Explainable AI (XAI) techniques to improve training, debugging, and communication with stakeholders. It emphasizes the importance of XAI in RL scenarios involving human interaction and suggests the development of theoretical foundations to support explainable RL. The paper calls for the development of tailored XAI techniques for DRL models and emphasizes the importance of accounting for feature interactions in explanations. The paper further highlights the critical role of these approaches in addressing challenges in RL applications and advocates the usage of XAI tools for responsible RL deployment (Heuillet et al. 2020). XAI seeks to address these challenges by developing tools and techniques that provide clear, understandable explanations for AI decisions, making these systems more transparent and trustworthy. This paper offers a comprehensive overview of XAI in deep learning, proposing a taxonomy that categorizes XAI techniques based on the scope of explanations they offer, the underlying methodologies, and the level of interpretability or usage. These categories aim to support the creation of interpretable and self-explanatory models. The paper also reviews key principles in XAI research and traces the evolution of landmark XAI studies from 2007 to 2020. Additionally, the authors evaluate explanation maps generated by eight different XAI algorithms on image data, highlighting the limitations of current evaluation methods (Das and Rad 2020). For these reasons, this paper along with the others mentioned here played a key role in selecting the XAI techniques used in this research.

# Chapter 3

# Background

## 3.1 Blockchain and Cryptocurrency

### 3.1.1 Blockchain

Blockchain is the core technology behind cryptocurrencies like Bitcoin and Ethereum, valued primarily for its decentralization. It enables peer-to-peer transactions and coordination in distributed systems without the need for centralized control or trust between participants. Blockchain achieves this through data encryption, time-stamping, consensus algorithms, and economic incentives. By removing intermediaries,it addresses the high costs, inefficiencies, and security risks found in traditional centralized systems. With the rise of Bitcoin, Ethereum and other cryptocurrencies, blockchain research and applications have grown rapidly. It is currently perceived as a disruptive innovation, potentially becoming the fifth major computing paradigm shift, following the mainframe, PCs, the Internet, and mobile/social networks. Beyond cryptocurrencies, blockchain is expected to revolutionize various sectors by transitioning from the Internet of Information to the Internet of Value, enabling the secure, decentralized transfer of assets and value across the web. Blockchain's potential applications extend beyond finance, with promising use cases in smart contracts, supply chain management, voting systems, and healthcare, offering secure, transparent, and tamper-proof record-keeping across industries (Yuan and F.-Y. Wang 2018).

### 3.1.2 Bitcoin

Bitcoin is a decentralized digital currency designed to enable peer-to-peer electronic payments without the need for intermediaries like banks. Introduced in 2009 by an anonymous entity known as Satoshi Nakamoto, Bitcoin's system was developed by engineers with minimal input from regulators or legal experts. At its core, Bitcoin operates on a blockchain, a public, distributed ledger maintained by a network of computers (or nodes), which verifies and records all transactions. A key feature of Bitcoin is the process of mining, in which participants (miners) compete to solve cryptographic puzzles

that validate transactions and add new blocks to the blockchain. Miners receive the newly created Bitcoin tokens as a reward for their contribution. This serves as the key mechanism for releasing new Bitcoin tokens into circulation, and the rewards are halved every few years (a process colloquially called "halving") until the maximum supply of 21 million bitcoins has been reached. This process ensures decentralization and incentivizes participation. Bitcoin allows users to create accounts and engage in transactions without the need for centralized approval or personal identification, offering privacy and flexibility. Its design enables irreversible transactions, a fixed path of money creation, and a fully transparent transaction history. While Bitcoin presents itself as more private and less regulated compared to traditional payment systems, these benefits are not without limitations and risks, especially concerning regulatory oversight. Economically, Bitcoin has the potential to disrupt traditional payment networks and even monetary systems, offering a decentralized alternative that challenges the central authority of established financial institutions (Böhme et al. 2015), Nakamoto 2008)

### 3.1.3 Ethereum

Ethereum is a decentralized platform introduced in 2013 by Vitalik Buterin through its whitepaper. Ethereum is based on Bitcoin's blockchain technology but it extends that functionality by supporting smart contracts which are self-executing contracts with the terms of the contract's agreement directly written into the code. Solidity which is statically typed programming language was solely developed for writing smart contracts on the Ethereum blockchain. Ethereum's blockchain enables developers to build decentralized applications (dApps) that run without downtime, fraud, or third-party interference. The platform uses a native cryptocurrency called Ether, which facilitates transactions, incentivizes miners, and powers dApps and smart contract executions. Ethereum's long term mission is to act as a "world computer" by providing a decentralized environment for various applications beyond simple digital currency transactions. Its versatility has made it the backbone of innovations like decentralized finance (DeFi), non-fungible tokens (NFTs), and initial coin offerings (ICOs) (Buterin 2013). Initially, Ethereum operated under a Proof-of-Work (PoW) consensus mechanism, similar to Bitcoin. However, in September 2022, Ethereum transitioned to a Proof-of-Stake (PoS) mechanism through an upgrade called Ethereum 2.0 or colloquially called "The Merge." In PoS, validators have replaced miners where they stake Ether to secure the network and validate transactions. This has significantly reduced the energy consumption of Ethereum by 99.98% compared to PoW. This shift also brought improvements in scalability, security, and sustainability. PoS introduced features like sharding, which allows the network to process more transactions in parallel, significantly increasing capacity and reducing congestion. The new Ethereum, post-Merge, is more energy-efficient, decentralized, and cost-effective, and positions itself for future upgrades while continuing to play a leading role in the blockchain ecosystem (Kapengut and Mizrach 2023).

## 3.2 Reinforcement Learning

In reinforcement learning (RL), an agent interacts with the environment and makes decisions i.e. takes actions to maximize cumulative rewards over time. The agent's primary goal is to learn how to make decisions i.e. take actions that yield the highest overall or expected cumulative reward (François-Lavet et al. 2018, Yuxi Li 2018).

Key concepts in reinforcement learning are:

- **States, Actions, and Rewards:** At each time step $t$, the agent observes a state $s_t$, chooses an action $a_t$, receives a reward $r_t$, and transitions to a new state $s_{t+1}$ in the environment.
- **Policy:** A policy $\pi$ is the strategy used by the agent for making decisions or taking actions by mapping states to actions. The primary objective in RL is to find an optimal policy $\pi^*$, which will maximize the expected cumulative reward, also called the return.
- **Value Function $V(s)$:** The value function represents the expected return (cumulative reward) starting from state $s$ and following the policy $\pi$.

Markov Decision Process (MDP) is a key component of Reinforcement Learning. It provides a framework for modelling the process of decision-making when the outcomes are partly random and partly under the control of a decision-maker i.e. agent (François-Lavet et al. 2018, Yuxi Li 2018). MDP is defined by a tuple $(S, A, P, R, \gamma)$ where:

- $S$ is the set of states.
- $A$ is the set of actions.
- $P(s'|s, a)$ is the transition probability from state $s$ to state $s'$ given action $a$.
- $R(s, a)$ is the reward function.
- $\gamma$ is the discount factor.

### 3.2.1 Deep Q-Network

**Q-Learning** is a model-free reinforcement learning algorithm that finds the optimal policy by learning the Q-function, which represents the expected cumulative reward for taking an action in a given state (François-Lavet et al. 2018, Mnih, Kavukcuoglu, et al. 2013).

**Q-Function:** The Q-function, or action-value function, $Q(s, a)$, represents the expected cumulative reward for taking action $a$ in state $s$ and following the optimal policy from there on. **Bellman Equation:** Q-Learning is based on the Bellman equation:

$$Q(s_t, a_t) = r_t + \gamma \max_{a'} Q(s_{t+1}, a') \tag{3.1}$$

**Update Rule:** The Q-Learning update rule adjusts the Q-values iteratively based on new experiences:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[ r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right] \tag{3.2}$$

Deep Q-Learning extends Q-Learning by using a deep neural network (DNN) to approximate the Q-function. This is very useful in environments with high-dimensional state spaces, where tabular methods are practical. A deep Q-network is a neural network parameterized by weights $\theta$. The network takes the state $s_t$ as input and outputs the Q-values $Q(s_t, a; \theta)$ for all possible actions $a$ (Mnih, Kavukcuoglu, et al. 2013).

The loss function used in DQN is based on the temporal difference (TD) error:

$$\text{Loss}(\theta) = \mathbb{E}_{(s,a,r,s')} \left[ (y_t - Q(s, a; \theta))^2 \right] \tag{3.3}$$

where the target $y_t$ is:

$$y_t = r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \theta^-) \tag{3.4}$$

Here, $\theta^-$ represents the weights of a target network, which is a copy of the Q-network's weights that is periodically updated to stabilize training.

Some key elements of a DQN are inherently different for a Q-learning model. DQN uses an **experience replay** buffer to break the correlation between consecutive samples and improve learning stability, The agent stores experiences $(s_t, a_t, r_t, s_{t+1})$ in a replay buffer and samples random mini-batches from this buffer and updates the network. **A target network** $\theta^-$ help to counter the instability of training by keeping the target $y_t$ fixed for a few iterations before updating it to match the current Q-network's weights $\theta$ (Mnih, Kavukcuoglu, et al. 2013).

### 3.2.2 Advantage Actor-Critic

Advantage Actor-Critic (A2C) is an implementation of the actor-critic framework with some enhancements. It is often seen as a synchronous version of Asynchronous Advantage Actor-Critic (A3C), where multiple environments are used in parallel to collect data, which is then used to update a shared model. Advantage Actor-Critic (A2C) is a reinforcement learning algorithm which combines the benefits of both value-based and policy-based methods. It builds on the actor-critic framework by using the advantage function to reduce the variance of policy gradient estimates, thereby improving the learning efficiency (Mnih, Badia, et al. 2016, François-Lavet et al. 2018 ).

**Advantage Function** $A(s, a)$**:** The advantage function is a way to measure how much better or worse an action $a$ is compared to the average action in the state $s$, given by $A(s, a) = Q(s, a) - V(s)$. The advantage function helps to reduce the variance of the policy gradient estimates. Instead of using $Q(s, a)$, which can be noisy, the advantage function provides a more stable estimate.

**Policy Gradient:** Policy gradient methods directly optimize the policy by maximizing the expected

return using the gradient ascent method. The policy is usually parameterized by $\theta$, and the objective is to maximize $J(\theta) = \mathbb{E}_{\pi_\theta}[R]$, where $R$ is the return.

The policy gradient theorem states that the gradient of the expected return concerning the policy parameters $\theta$ can be expressed as:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) Q(s,a)] \tag{3.5}$$

**Actor-Critic:** The actor-critic framework combines policy-based and value-based methods. The "actor" updates the policy $\pi(a|s;\theta)$, while the "critic" evaluates the policy by learning the value function $V(s;w)$.

#### Mathematical foundation of A2C

The actor is a policy network parameterized by $\theta$ that outputs a probability distribution over actions given a state $s$. The critic is a value network parameterized by $w$ which estimates the value function $V(s;w)$ for the current policy.

The advantage function in A2C is typically estimated using the TD error:

$$A(s_t, a_t) = r_t + \gamma V(s_{t+1}; w) - V(s_t; w) \tag{3.6}$$

This advantage is used to update the actor's policy. The policy or the (actor) is updated by maximizing the expected advantage:

$$\nabla_\theta J(\theta) = \mathbb{E}[\nabla_\theta \log \pi_\theta(a_t|s_t) A(s_t, a_t)] \tag{3.7}$$

The value function (critic) is updated by minimizing the squared TD error:

$$L(w) = (r_t + \gamma V(s_{t+1}; w) - V(s_t; w))^2 \tag{3.8}$$

### 3.2.3 Proximal Policy Optimization

Proximal Policy Optimization (PPO) is a reinforcement learning algorithm, designed to address the instability issues commonly faced in policy gradient methods while still being simple enough for practical implementation. **KL Divergence Constraint:** TRPO is a policy gradient method that improves stability by constraining the policy update using the KL divergence between the old and new policies. This way the new policy does not diverge too far from the old policy, which in turn will destabilize training. PPO is an advancement over TRPO, designed to retain the benefits of TRPO while being more computationally efficient and easier to implement (Schulman et al. 2017, Yuxi Li 2018).

**Mathematical foundation of PPO**

The central innovation of PPO is the clipped surrogate objective, which modifies the policy gradient objective to include a constraint that prevents large updates to the policy. This constraint is implemented by clipping the probability ratio $r_t(\theta)$, defined as:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}, \tag{3.9}$$

where $\pi_{\theta_{\text{old}}}(a_t|s_t)$ is the probability of taking action $a_t$ under the old policy, and $\pi_\theta(a_t|s_t)$ is the probability under the new policy.

The PPO objective function is defined as:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t \left[ \min \left( r_t(\theta) A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon) A_t \right) \right], \tag{3.10}$$

where $A_t$ is the advantage function at time step $t$, and $\epsilon$ is a hyperparameter that controls the extent of clipping (typically set to 0.2).

**Unclipped Objective:** The term $r_t(\theta)A_t$ represents the standard policy gradient objective. If the advantage $A_t$ is positive, increasing $r_t(\theta)$ increases the objective; if $A_t$ is negative, decreasing $r_t(\theta)$ increases the objective. **Clipping:** The function $\text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)$ ensures that the ratio $r_t(\theta)$ stays within the interval $[1-\epsilon, 1+\epsilon]$. This clipping prevents $r_t(\theta)$ from deviating too far from 1, thereby limiting the policy update and preventing the new policy from becoming too different from the old policy. By taking the minimum of the unclipped and clipped objectives, PPO ensures that the policy update is conservative, which helps maintain training stability.

In addition to optimizing the policy, PPO also optimizes the value function $V(s; w)$, where $w$ are the parameters of the value function. The loss for the value function is typically the mean squared error between the predicted value and the actual return:

$$L^{\text{VF}}(w) = \mathbb{E}_t \left[ \left( V(s_t; w) - R_t \right)^2 \right], \tag{3.11}$$

where $R_t$ is the actual return (sum of discounted rewards) at time step $t$.

PPO includes an entropy bonus, which is added to the objective function to encourage exploration and find a balance between exploration and exploitation. The entropy of the policy $\pi_\theta(a|s)$ is given by:

$$H(\pi_\theta) = \mathbb{E}_s \left[ -\sum_a \pi_\theta(a|s) \log \pi_\theta(a|s) \right]. \tag{3.12}$$

The final objective function, combining the policy loss, value function loss, and entropy bonus, is:

$$L(\theta, w) = \mathbb{E}_t \left[ L^{\text{CLIP}}(\theta) - c_1 L^{\text{VF}}(w) + c_2 H(\pi_\theta) \right], \tag{3.13}$$

where $c_1$ and $c_2$ are coefficients that balance the contributions of the value function loss and entropy bonus (Schulman et al. 2017).

### 3.2.4 Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient (DDPG), is a reinforcement learning algorithm that is well-suited for environments with continuous action spaces. It combines elements from both Q-learning (commonly used in discrete action spaces) and policy gradient methods which are effective in continuous action spaces to create a powerful algorithm that can handle high-dimensional, continuous control problems (Lillicrap et al. 2019). Unlike stochastic policies, where actions are sampled from a probability distribution, deterministic policies are deterministic functions of the state, i.e., $a = \mu_\theta(s)$. The deterministic policy gradient theorem states that the gradient of the expected return $J(\theta)$ with respect to the policy parameters $\theta$ can be expressed as:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim\rho^\mu}\left[\nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s,a)\big|_{a=\mu_\theta(s)}\right], \tag{3.14}$$

where $\rho^\mu$ is the discounted state distribution under the policy $\mu_\theta$, and $Q^\mu(s,a)$ is the Q-value under policy $\mu_\theta$.

In DDPG, we adapt the idea of Q-learning to continuous action spaces. Since the action space is continuous, we can't use the max operation over all actions in the Bellman equation. Instead, we use the policy $\mu_\theta(s)$ to approximate the action (Lillicrap et al. 2019):

$$Q(s,a) = r + \gamma Q(s', \mu_\theta(s')). \tag{3.15}$$

#### Mathematical foundation of DDPG

**Actor Network**Parameterized by $\theta$, the actor network approximates the deterministic policy $\mu_\theta(s)$.

**Critic Network**Parameterized by $w$, the critic network approximates the Q-function $Q(s,a;w)$.

Target networks $\mu'_{\theta'}(s)$ and $Q'(s,a;w')$ are used to stabilise training. These are the gradually updated versions of the actor and critic networks:

$$\theta' \leftarrow \tau\theta + (1-\tau)\theta', \tag{3.16}$$

$$w' \leftarrow \tau w + (1-\tau)w', \tag{3.17}$$

where $\tau$ is a small constant (e.g., $\tau = 0.001$). Experiences $(s,a,r,s')$ are stored in a replay buffer, and during training, mini batches are sampled from this buffer to de-correlate experiences and stabilize the learning process.

The critic network is trained by minimizing the loss:

$$L(w) = \mathbb{E}_{(s,a,r,s')\sim\text{Replay Buffer}}\left[(Q(s,a;w) - y)^2\right], \tag{3.18}$$

where the target value $y$ is given by:

$$y = r + \gamma Q'(s', \mu'_{\theta'}(s'); w'). \tag{3.19}$$

The actor-network is updated using the deterministic policy gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim\text{Replay Buffer}}\left[\nabla_\theta \mu_\theta(s) \nabla_a Q(s,a;w)\big|_{a=\mu_\theta(s)}\right]. \tag{3.20}$$

## 3.3 XAI

Explainable AI (XAI) is an emerging field in machine learning (ML) and artificial intelligence (AI) that aims to enhance the transparency, interpretability, and understandability of AI models for humans. Traditional AI models are often regarded as "black boxes," where the inputs and outputs are visible but the internal decision-making processes remain obscure. XAI seeks to open up these black boxes, offering insights into how models function, the reasoning behind their decisions, and the factors influencing those outcomes. This transparency is particularly critical in high-stakes fields like healthcare, finance, and criminal justice, where trust, fairness, and accountability are paramount. XAI aims to achieve these four key objectives **Transparency**: Ensuring that the internal structure and mechanisms of an AI model are accessible, understandable, and open to scrutiny, **Interpretability**: Providing human-understandable explanations of the model's decision-making process, making it easier for users to follow and comprehend how conclusions are reached, **Accountability**: Establishing clear and traceable paths for how AI decisions are made, enabling these decisions to be evaluated, justified, and audited, and **Fairness**: Identifying and addressing any biases or unfair treatment present in AI models to promote more equitable outcomes.

XAI methods can be broadly classified into two main categories: **Intrinsic interpretability**: These methods involve models that are inherently interpretable due to their simplicity and structure, such as decision trees, linear regression, or rule-based models, where the decision-making process is inherently clear. And **Post-hoc interpretability**: Techniques applied after a complex model has been trained, aiming to explain its behavior. These methods include techniques like LIME (Local Interpretable Model-agnostic Explanations), SHAP (SHapley Additive exPlanations), saliency maps, and integrated gradients, which offer insights into how complex models like deep neural networks reach decisions.

XAI is increasingly seen as essential for building trust in AI systems, particularly when deployed in sensitive and impactful domains. It not only helps users and developers understand how decisions are made but also contributes to the responsible deployment of AI by ensuring the systems are more transparent, accountable, and aligned with ethical considerations. As the field advances, XAI

continues to evolve, with research focusing on improving the quality, clarity, and usability of AI explanations across various industries.

Two of the most widely used, effective and straightforward XAI techniques that determine the importance of different input features in the context of a trained DRL model are **Saliency Maps** & **Integrated Gradients**. These two XAI techniques have been chose for this research after careful examination.

### 3.3.1 Saliency Maps

Saliency maps are a widely used tool in Explainable Artificial Intelligence (XAI), designed to shed light on the decision-making process of complex, often opaque machine learning models, especially deep neural networks. In the context of black-box models, saliency maps provide visual explanations by highlighting the input regions—such as pixels in images or features in other data—that are most influential in driving the model's predictions. Essentially, they indicate where the model is "looking" when making a decision, helping users understand its internal workings. This fosters trust and transparency, as users can see what parts of the input data are considered important by the model. Saliency maps are particularly useful in tasks like image classification, where they can reveal which pixels or regions led to the model's decision for a particular class. However, while saliency maps can enhance the interpretability of models, they are not without limitations. Some methods might produce visually appealing explanations without genuinely reflecting the model's internal logic, making them less reliable for sensitive tasks like identifying data outliers or debugging models. Thus, it's crucial to evaluate the quality and accuracy of these explanations carefully, as visual appeal alone can be misleading (Das and Rad 2020).

The saliency of each feature is determined by computing the gradient of the model's output with respect to the input features. For a given model output $y$ and an input feature $x_i$, the saliency map is computed as the absolute value of the gradient of $y$ with respect to $x_i$:

$$\text{Saliency}(x_i) = \left| \frac{\partial y}{\partial x_i} \right|$$

### 3.3.2 Integrated Gradients

Integrated Gradients (IG) is a method used to assign importance scores to input features in deep learning models by calculating the integral of gradients along a straight path from a baseline input to the actual input. This approach overcomes the issue of saturation—where gradients may diminish or become zero—by considering the entire path of the input, making it more comprehensive than traditional gradient-based attribution methods. IG ensures that the model's sensitivity to each feature is captured fully by summing up the gradients between a baseline (which signifies feature absence) and the actual input. This method satisfies key properties like sensitivity and implementation invariance,

providing a more reliable way to interpret complex models. IG is frequently employed in healthcare and finance, where model explainability is vital (Das and Rad 2020).

For a model $f$ and an input instance $x$, integrated gradients for feature $i$ are computed as:

$$\mathrm{IG}_i(x) = (x_i - x_i') \int_{\alpha=0}^{1} \frac{\partial f(x' + \alpha(x - x'))}{\partial x_i} \, d\alpha$$

where:

- $x'$ is the baseline input (e.g., an input with all zeros).
- $\alpha$ is a scaling factor that varies from 0 to 1.
- $\frac{\partial f(x)}{\partial x_i}$ is the gradient of the model's output with respect to feature $x_i$.

# Chapter 4

# Methodology

## 4.1 Data Collection and Data Processing

### 4.1.1 Data Collection

In this study, historical cryptocurrency data for Bitcoin and Ethereum is retrieved and processed using an API (Application Programming Interface) provided by CryptoCompare. Web requests and data wrangling operations are integrated to structure the data into a usable format for financial analysis. This approach leverages RESTful API principles and the Python library pandas, to prepare and collect the data for deeper investigation into cryptocurrency market behavior. The resulting OHLCV dataset includes critical financial indicators like open, high, low, and close prices, and volume at an hourly interval which are essential for technical analysis, predictive modelling, and algorithmic trading strategies. Apart from OHLCV data, blockchain metrics have also been collected and added to the OHLCV data to create a very comprehensive dataset for both cryptocurrencies. The OHLCV data and blockchain data for both BTC and ETH have been collected from, 18/11/2022 till 30/06/2024 i.e. for a period of 600 days. After the transition of Ethereum from PoW to PoS two key blockchain metrics specific to PoW namely **hashrate** and **difficulty** are no longer relevant. As a result, Ethereum staking data is gathered for a specified limit of 600 days same as the OHLCV data, which allows for a more comprehensive data set and longitudinal analysis of key metrics. The data has been exported to a CSV file to ensure portability, allowing for further processing, and integration with the DRL models to forecast future price movements or volatility and thereby execute trades in the cryptocurrency market.

### 4.1.2 Data Processing

The raw OHLCV data for BTC and ETH are available at hourly intervals but quite a few blockchain metrics like difficulty, hashrate, staking rate .etc are available only at daily intervals so these daily metrics have been resampled to match hourly intervals. The TA library in Python has been used

to generate prominent Technical Analysis indicators from the hourly OHLCV data like EMA, SMA, RSI etc which take into account both volume, volatility and price fluctuations (Padial 2018, Schwager 1995). The merged OHLCV and Blockchain dataset for BTC and ETH allows for the examination of price movements and trading volumes, essential for volatility and liquidity analysis in cryptocurrency markets.

| Column | Description |
|---|---|
| datetime | Timestamp for the data point, likely in hourly intervals. |
| open | The opening price of BTC for the period. |
| high | The highest price of BTC for the period. |
| low | The lowest price of BTC for the period. |
| close | The closing price of BTC for the period. |
| volume | The trading volume of BTC during the period. |
| SMA_7 | Simple Moving Average over 7 periods. |
| SMA_14 | Simple Moving Average over 14 periods. |
| SMA_30 | Simple Moving Average over 30 periods. |
| EMA_7 | Exponential Moving Average over 7 periods. |
| EMA_14 | Exponential Moving Average over 14 periods. |
| EMA_30 | Exponential Moving Average over 30 periods. |
| RSI_14 | Relative Strength Index over 14 periods. |
| MACD | Moving Average Convergence Divergence. |
| MACD_Signal | Signal line of the MACD. |
| MACD_Hist | The difference between MACD and its signal line. |
| VWAP | Volume Weighted Average Price of BTC. |
| CMF | Chaikin Money Flow indicator for buying/selling pressure. |
| ATR | Average True Range, indicating the volatility of the price. |
| id | Unique identifier for each row. |
| zero_balance_addresses_all_time | Total addresses with zero balance since inception. |
| unique_addresses_all_time | Total number of unique addresses ever created. |
| new_addresses | Number of new addresses created in the period. |
| active_addresses | Number of active addresses in the period. |
| transaction_count | Number of transactions within the period. |
| transaction_count_all_time | Total number of transactions since inception. |
| large_transaction_count | Number of large transactions in the period. |
| average_transaction_value | Average value of transactions in the period. |
| block_height | The height of the blockchain at this point in time. |
| hashrate | Mining hashrate at the time of recording. |
| difficulty | The mining difficulty of the Bitcoin blockchain. |
| block_time | The time taken to create a new block. |
| block_size | The size of the block in bytes. |
| current_supply | The total circulating supply of Bitcoin. |

**Table 4.1** – Description of the Bitcoin(BTC) Dataset

| Column | Description |
|---|---|
| datetime | Timestamp for the data point, likely in hourly intervals. |
| open | The opening price of ETH for the period. |
| high | The highest price of ETH for the period. |
| low | The lowest price of ETH for the period. |
| close | The closing price of ETH for the period. |
| volume | The trading volume of ETH during the period. |
| SMA_7 | Simple Moving Average over 7 periods. |
| SMA_14 | Simple Moving Average over 14 periods. |
| SMA_30 | Simple Moving Average over 30 periods. |
| EMA_7 | Exponential Moving Average over 7 periods. |
| EMA_14 | Exponential Moving Average over 14 periods. |
| EMA_30 | Exponential Moving Average over 30 periods. |
| RSI_14 | Relative Strength Index over 14 periods. |
| MACD | Moving Average Convergence Divergence. |
| MACD_Signal | Signal line of the MACD. |
| MACD_Hist | The difference between MACD and its signal line. |
| VWAP | Volume Weighted Average Price of ETH. |
| CMF | Chaikin Money Flow indicator for buying/selling pressure. |
| ATR | Average True Range, indicating the volatility of the price. |
| id | Unique identifier for each row. |
| zero_balance_addresses_all_time | Total addresses with zero balance since inception. |
| unique_addresses_all_time | Total number of unique addresses ever created. |
| new_addresses | Number of new addresses created in the period. |
| active_addresses | Number of active addresses in the period. |
| transaction_count | Number of transactions within the period. |
| transaction_count_all_time | Total number of transactions since inception. |
| large_transaction_count | Number of large transactions in the period. |
| average_transaction_value | Average value of transactions in the period. |
| block_height | The height of the blockchain at this point in time. |
| hashrate | Mining hashrate at the time of recording. |
| difficulty | The mining difficulty of the Ethereum blockchain. |
| block_time | The time taken to create a new block. |
| block_size | The size of the block in bytes. |
| current_supply | The total circulating supply of Ethereum. |
| rate | Staking reward rate for that time period. |

**Table 4.2** – Description of the Ethereum(ETH) Dataset

The table 4.1 describes the attributes of the BTC dataset used for training the DRL models in this study. The dataset comprises 14,184 rows containing Bitcoin (BTC) financial and blockchain-related metrics. Each row represents an hourly data point, providing crucial price indicators like the open, high, low, and closing prices of BTC, along with its trading volume. The dataset also includes various technical indicators such as moving averages (SMA and EMA), RSI, MACD, and VWAP, which are valuable for market analysis. Additionally, it includes blockchain statistics such as transaction counts, address metrics, block height, mining difficulty, hashrate, and supply information, making it useful for both financial and blockchain research.

The table 4.2 describes the attributes of the ETH dataset used for training the DRL models in this

study.The dataset comprises 14,184 entries detailing various financial and blockchain metrics related to Ethereum (ETH), with each entry timestamped in hourly intervals. Key financial metrics include open, high, low, close prices, and trading volume. Additionally, various technical indicators, such as simple and exponential moving averages (SMA and EMA), relative strength index (RSI), and MACD are included to assist in market analysis. The dataset also contains blockchain-related metrics, such as transaction counts, address statistics, block height, and staking rate, providing a comprehensive overview of Ethereum's market and blockchain behavior.

## 4.2 Implementation

The comparison table in the related work section reveals that DQN, A2C, PPO, and DDPG are some of the most widely used DRL algorithms for cryptocurrency price prediction and trading. In this study, all 4 algorithms have been implemented from the ground up for trading cryptocurrencies and generating profit. The implementation of each algorithm is very different from one another, however, all four algorithms have been developed with the sole purpose of learning the trends in the market data and blockchain and use that knowledge to make successful trades. All algorithms implemented in this study have been trained across 100, 200 and 500 episodes to determine and compare their performance as the DRL models learn the market dynamics over each iteration. Given the size of the dataset and each episode being 30 days, all of the DRL models loop over the entire dataset 5, 10 and 25 times for 100, 200 and 500 episodes respectively.

Most papers in this domain implement the buying and selling trading aspect of the system in an all-in or all-out manner i.e. the agent in the system either buys and sells assets equivalent to their entire allocated trading amount. This is computationally lighter and easy to implement but is very impractical and has many limitations most important of which is a very high-risk factor, significant losses in case of a bad trade or unfavourable financial market conditions and the complete inability of the DRL agent to learn an effective trading policy.

### 4.2.1 Fractional Trading

Every implementation in this study supports **fractional trading**. **Fractional trading** is useful because it reduces risk by spreading out trades over time, rather than making all-in or all-out decisions. It allows the agent to build positions gradually, making the strategy more realistic in volatile markets like cryptocurrency. By adjusting trade-fraction, more conservative strategies that prevent the agent from losing large amounts of capital in one trade while still allowing for gradual profit accumulation can be implemented. This adds flexibility to the agent's learning process and helps manage risks associated with high volatility in crypto markets.

The DRL models : DQN, A2C and PPO implemented in this study have been trained tested for

varying levels of fractional trading ranging from 10% to 50% for both BTC and ETH to determine the optimum level of fractional trading that produces the best results. The only exception to this is the DDPG model which has been implemented from the ground up with optimized fractional trading in mind i.e. it automatically selects the optimum fraction of the asset it is trading along with the best possible trading decision (Buy, Sell or Hold) at any given moment. This is possible by leveraging the ability of DDPG to work on continuous action spaces.

### 4.2.2 Deep Q-Network

**Q-Network Architecture:** The Q-Network is a neural network that estimates the expected future reward (Q-value) for each action the agent can take (Buy, Sell, Hold) in a given state. It consists of:

- **Input Layer**: Receives the current state (market indicators) as a vector of size 33 or 34 (adjustable based on the dataset).
- **Hidden Layers**: Two fully connected (dense) layers, each with 64 neurons and ReLU activations. These layers learn to extract useful features from the input data.
- **Output Layer**: Outputs Q-values for the three possible actions. Each Q-value represents the expected reward for taking a specific action in the given state.

**Target Network:** A second Q-network, the **target network**, provides stable target Q-values during training. The target network's weights are updated periodically (every 18 episodes) to match the main Q-network. This prevents large oscillations in Q-value estimates during training and makes learning more stable.

**Experience Replay:** To improve the efficiency of training, the algorithm uses an **experience replay buffer** (implemented with `deque`). This buffer stores past experiences (state, action, reward, next state, done) during trading episodes. Instead of updating the Q-network with each step, experiences are sampled in batches (128 experiences) from the buffer and used for training. This helps break the correlation between consecutive steps and allows the agent to reuse valuable past experiences multiple times.

**Epsilon-Greedy Action Selection:** The agent follows an epsilon-greedy policy to balance exploration and exploitation:

- **Exploration**: With a probability $\epsilon$, the agent randomly selects an action to discover new strategies.
- **Exploitation**: With a probability $1 - \epsilon$, it chooses the action that has the highest predicted Q-value, thus exploiting what it has learned so far.
- **Epsilon Decay**: Over time, $\epsilon$ decays (from 1.0 to 0.01), gradually shifting the agent's behavior from exploration to exploitation as it becomes more confident in its learned strategy.

**Training Process:** Training happens at each step of the episode:

1. The agent selects an action based on the current state and epsilon-greedy policy.
2. It executes the action in the trading environment, transitioning to a new state and receiving a reward.
3. This transition (state, action, reward, next state, done) is stored in the replay buffer.
4. If there are enough experiences in the buffer, a batch of experiences is sampled for training.
5. The Q-network is updated by minimizing the Mean Squared Error (MSE) between the predicted Q-values and target Q-values:

$$\text{Target Q} = \text{reward} + \gamma \times \max(\text{next Q-value})$$

   (for non-terminal steps).
6. **Loss Function**: MSE loss between the Q-network's output and the target values.

**Environment and Reward System:** The trading environment simulates a cryptocurrency market. At each step, the agent takes one of three actions (Buy, Sell, or Hold). The environment keeps track of the agent's balance, crypto holdings, and net worth. Rewards are based on changes in net worth:

- **Positive reward**: If the agent increases its net worth.
- **Negative reward**: If the agent loses net worth or holds too long (small penalty for holding).

The goal is to maximize the agent's net worth and the Sharpe Ratio (risk-adjusted returns).

**Performance Metrics:** At the end of each episode, key performance metrics are logged:

- **Total Profit**: Cumulative rewards during the episode.
- **Sharpe Ratio**: A measure of risk-adjusted returns, calculated based on the history of the agent's net worth.
- **Number of Trades**: Tracks the total number of Buy/Sell actions executed.

These metrics help monitor the agent's learning progress and performance in the trading environment.

**Target Network Update:** To ensure stability during training, the weights of the target network are updated every 18 episodes to match the Q-network's weights. This ensures that the target values used during training don't change too quickly, preventing the agent from chasing moving targets.

In this DQN implementation, fractional trading is executed through the variable `trade_fraction`, which currently controls the percentage of the agent's balance or crypto holdings to trade. When the agent decides to buy, it calculates the amount to buy as:

$$\text{amount\_to\_buy} = \text{balance} \times \text{trade\_fraction}$$

and increases its holdings accordingly. Similarly, for selling, the agent sells `trade_fraction` of its current crypto holdings. This design allows for flexible control over how much of the agent's assets are traded at any given time. `trade_fraction` can be adjusted to different values, such as 0.1 or 0.5, to simulate trading 10% or 50% of the balance or holdings.

**Sequential steps for the DQN algorithm**

- *Initialize Hyperparameters:*
  - Set learning rate, gamma, epsilon, epsilon decay, epsilon min, target update, batch size, episode length, memory size.
- *Initialize Q-Network:*
  - Create a neural network with input size equal to state size and output size equal to action size (3 actions: Buy, Sell, Hold).
  - Apply ReLU activations between layers.
- *Initialize Target Network:*
  - Copy weights from Q-network to target network.
- *Initialize Replay Buffer:*
  - Create a replay buffer to store experience tuples $(state, action, reward, next\_state, done)$.
- *Select Action (Epsilon-Greedy):*
  - If the random value is less than epsilon, select a random action.
  - Otherwise, forward pass the current state through the Q-network and select the action with the highest Q-value.
- *Train the DQN:*
  - If the replay buffer has enough samples:
    - Sample a random batch of transitions from the replay buffer.
    - Calculate Q-values for the current states using the Q-network.

        * Calculate Q-values for the next states using the target network.
        * Compute the target Q-values using the reward and discounted future Q-values.
        * Compute loss and backpropagate through the Q-network.

- *Initialize Trading Environment:*
  - Initialize environment with data, balance, crypto holdings, and net worth.
  - Track net worth over time.
- *Reset Environment:*
  - Reset current step, balance, and net worth to initial values.
  - Return the initial state.
- *Step Through the Environment:*
  - Take action (Buy, Sell, Hold).
  - Update balance and crypto holdings based on action.
  - Update net worth based on new price.
  - Calculate reward as the change in net worth.
  - If the action is Hold, apply a small penalty to the reward.
  - Otherwise, apply a small bonus to the reward.
  - Return the next state, reward, and done flag.
- *Main Training Loop:*
  - For each episode:
    * Reset environment and get the initial state.
    * For each step within the episode:
      · Select an action using epsilon-greedy policy based on the current state.
      · Perform the action in the environment.
      · Store the transition in the replay buffer.
      · Train the DQN.
    * Periodically update the target network.
    * Decay the exploration rate (epsilon).
    * Log and track total profit, number of trades, and Sharpe Ratio.

The table 4.3 describes and specifies all the parameters used for developing and then training and testing the DQN model on BTC and ETH data. The table can be used to implement and reproduce the results obtained in this study.

| Parameter | Value | Description |
|---|---|---|
| initial_balance | 10,000 | Starting capital for trading. |
| state_size | 33/34 | Number of features representing the state (BTC & ETH data). |
| action_size | 3 | Possible actions: Buy, Sell, Hold. |
| learning_rate | 0.0001 | Learning rate for the Adam optimizer. |
| gamma | 0.99 | Discount factor for future rewards. |
| epsilon | 1.0 | Initial exploration rate (100% exploration). |
| epsilon_decay | 0.995 | Factor by which epsilon decays after each episode. |
| epsilon_min | 0.01 | Minimum value epsilon can decay to (1% exploration). |
| target_update | 18 | Frequency (in episodes) to update the target network. |
| batch_size | 128 | Number of samples per training batch. |
| episode_length | 720 | Steps per episode (e.g., 30 days * 24 hours/day). |
| memory_size | 20,000 | Maximum number of experiences in the replay buffer. |
| **Q-Network Architecture** | | |
| fc1 (Layer 1) | $34 \rightarrow 64$ | First fully connected layer from state to 64 neurons. |
| fc2 (Layer 2) | $64 \rightarrow 64$ | Second fully connected layer with 64 neurons. |
| fc3 (Output Layer) | $64 \rightarrow 3$ | Output layer producing Q-values for each action. |
| **Training Parameters** | | |
| num_episodes | 100 | Total number of training episodes. |
| trade_fraction | 0.25 | Fraction of balance or holdings to trade (25%). |
| reward_penalty_hold | -0.01 | Penalty for taking the Hold action. |
| reward_bonus_action | +0.005 | Bonus for taking Buy or Sell actions. |
| risk_free_rate | 0.0 | Used in Sharpe Ratio calculation (set to zero). |

**Table 4.3** – The parameters in the DQN implementation.

### 4.2.3 Advantage Actor-Critic

**Actor-Critic Network Architecture:**

The **Actor-Critic network** has two heads: the **Actor** and the **Critic**:

- **Actor**: Responsible for choosing the action (Buy, Sell, or Hold) given the current state of the environment. The actor head outputs a probability distribution over the available actions.
- **Critic**: Estimates the value of the current state, helping the agent understand whether the state is good or bad based on the future expected rewards.

The network has two fully connected layers with 64 neurons each, followed by two separate heads:

- **Actor head**: Outputs an action probability vector of size `action_size` (3 in this case for Buy, Sell, Hold).
- **Critic head**: Outputs a scalar value representing the expected reward for the current state.

**Trading Environment:**

The `CryptoTradingEnv` class simulates a cryptocurrency trading environment:

- **State**: The environment's state consists of 33 or 34 features (adjustable based on the dataset). These could be technical indicators, market prices, or other financial metrics.
- **Actions**: The agent can choose one of three actions:
  - **Buy**: Purchase cryptocurrency using a fraction (25%) of the available balance.
  - **Sell**: Sell 25% of the current cryptocurrency holdings.
  - **Hold**: Do nothing, which incurs a small penalty to encourage active trading.
- **Reward**: The reward is based on changes in net worth, encouraging the agent to increase its wealth. Holding results in a small penalty to push the agent toward more decisive actions.

**A2C Training Process:**

The **A2C algorithm** uses the **actor-critic** approach:

- **Actor**: Learns to select actions that maximize cumulative rewards using the **advantage**—a measure of how much better an action is compared to the expected value of the state.
- **Critic**: Learns to evaluate how good a particular state is by estimating the value function.

The **Advantage** is calculated as the difference between the return (future cumulative rewards) and the estimated value of the state.

1. **Transition Buffer**: The `A2CBuffer` class is used to store the agent's transitions during an episode (states, actions, rewards, values, and log probabilities).
2. **Advantage Calculation**: After each episode, the agent computes returns and advantages using the stored transitions. The return is computed by backpropagating the rewards from the future, while the advantage is the difference between the return and the critic's estimated value.
3. **Actor-Critic Loss**:
   - **Actor Loss**: Encourages actions that result in higher rewards. It's computed as the negative log probability of the chosen action weighted by the advantage.
   - **Critic Loss**: Helps the Critic improve its value estimates by minimizing the difference (MSE) between predicted values and actual returns.
4. **Optimization**: The total loss (a combination of actor and critic losses) is used to update the parameters of the network.

**Action Selection and Training Loop:**

In the training loop:

1. **Action Selection**: At each time step, the agent computes the action probabilities using the actor head. An action is sampled from this distribution and executed in the environment.
2. **Environment Interaction**: The agent receives a reward based on the net worth change after executing the action.
3. **Buffer Storage**: The agent stores the state, action, log-probabilities, reward, and the value of the current state in the buffer for training after the episode ends.
4. **Training**: After each episode, the A2C network is trained using the buffer. The actor and critic are updated to improve action selection and value estimation.
5. **Sharpe Ratio Calculation**: The Sharpe ratio is calculated after each episode to measure the agent's risk-adjusted returns.

**Performance Metrics:**

At the end of each episode, the following performance metrics are logged:

- **Total Profit**: The cumulative profit achieved by the agent.
- **Sharpe Ratio**: A measure of the agent's risk-adjusted return.
- **Number of Trades**: Tracks the number of Buy/Sell decisions made.

**Sequential steps for the A2C algorithm**

- *Initialize Hyperparameters:*
  - Set initial balance, state size, action size, learning rate, gamma, number of episodes, episode length.
- *Define Actor-Critic Network:*
  - Create a neural network with shared layers.
  - Actor outputs action probabilities.
  - Critic outputs state value.
- *Define A2C Buffer:*
  - Create buffer to store states, actions, log_probs, rewards, and values.
- *Initialize Networks and Optimizer:*
  - Initialize ActorCritic network.
  - Use Adam optimizer with specified learning rate.

- *Define Trading Environment:*
  - Manage balance, crypto holdings, and net worth.
  - Track net worth history and step through the environment based on actions.
  - Return next state, reward, and done flag after each action.
- *Compute Returns and Advantages:*
  - Compute discounted returns and advantages from rewards and values.
- *Train A2C Model:*
  - Compute actor loss from action log probabilities and advantages.
  - Compute critic loss from predicted values and returns.
  - Backpropagate and update network weights using the optimizer.
- *Main Training Loop:*
  - For each episode:
    - Reset environment and buffer.
    - For each step:
      - Select action from Actor network.
      - Perform action in environment, store transition in buffer.
    - After episode ends, compute returns and train A2C.
    - Track and print profit, Sharpe ratio, and number of trades.

The table 4.4 describes and specifies all the parameters used for developing and then training and testing the A2C model on BTC and ETH data. The table can be used to implement and reproduce the results obtained in this study.

| Parameter | Value | Description |
|---|---|---|
| initial_balance | 10,000 | Starting capital for trading. |
| state_size | 33/34 | Number of features representing the state (ETH and BTC data). |
| action_size | 3 | Number of possible actions: Buy, Sell, Hold. |
| learning_rate | 0.0001 | Learning rate for the Adam optimizer. |
| gamma | 0.99 | Discount factor for future rewards. |
| num_episodes | 100 | Total number of training episodes. |
| episode_length | 720 | Number of steps per episode (e.g., 30 days * 24 hours/day). |
| trade_fraction | 0.25 | Fraction of balance or holdings to trade (25%). |
| reward_penalty_hold | -0.01 | Penalty for taking the Hold action (encourages action). |
| reward_bonus_action | +0.005 | Bonus for taking Buy or Sell actions. |
| risk_free_rate | 0.0 | Risk-free rate used in Sharpe Ratio calculation. |
| **Neural Network Architecture** | | |
| fc1 (Layer 1) | $33 \rightarrow 64$ | First fully connected layer from state to 64 neurons. |
| fc2 (Layer 2) | $64 \rightarrow 64$ | Second fully connected layer with 64 neurons. |
| Actor Head | $64 \rightarrow 3$ | Outputs action probabilities (Buy, Sell, Hold). |
| Critic Head | $64 \rightarrow 1$ | Outputs the scalar value for the current state. |

**Table 4.4** – The parameters in the A2C implementation.

### 4.2.4 Proximal Policy Optimization

**Actor-Critic Network Architecture:**

The **Actor-Critic** network is composed of two parts:

- **Actor**: Responsible for selecting an action (Buy, Sell, Hold). The actor outputs a probability distribution over the available actions.
- **Critic**: Responsible for estimating the value of the current state, which is used to calculate the advantage.

The network consists of two hidden layers, `fc1` and `fc2`, each with 64 neurons. The actor head outputs a vector of size equal to the number of actions (3), and the critic head outputs a scalar representing the value of the state.

**PPO Buffer:**

The `PPOBuffer` class is used to store trajectories of states, actions, log probabilities, rewards, and value estimates during trading episodes. The buffer is cleared at the start of each episode and filled with the transitions from the agent's interaction with the environment. These transitions are later used for the PPO update, where advantages and returns are computed for policy optimization.

**Trading Environment:**

The `CryptoTradingEnv` simulates the trading environment for the PPO algorithm. It manages the agent's balance, cryptocurrency holdings, net worth, and tracks the number of trades made. The

environment provides the following key features:

- **State**: The environment's state is composed of 33 or 34 features, which could include prices, technical indicators, or other market data.
- **Actions**: The agent can choose from three actions:
  - **Buy**: The agent buys 25% of its balance in cryptocurrency.
  - **Sell**: The agent sells 25% of its current cryptocurrency holdings.
  - **Hold**: The agent takes no action but incurs a small penalty to discourage inactivity.
- **Rewards**: The reward is calculated as the change in the agent's net worth after each action. Positive rewards encourage increasing the agent's net worth, while holding results in a penalty.

**PPO Loss and Training:**

The key to the PPO algorithm is the use of a clipped loss function, which ensures that policy updates are constrained to prevent large, unstable changes. This is implemented in the `ppo_loss` function:

- **Old vs. New Log Probs**: PPO compares the ratio of new log probabilities to old log probabilities for the selected actions. If the ratio remains within a range defined by the clipping factor $\epsilon = 0.2$, the policy is updated. Otherwise, changes are restricted.
- **Clipping**: The ratio of log probabilities is clipped within the range $[1 - \epsilon, 1 + \epsilon]$ to prevent large updates, ensuring smoother and more stable learning.

The `train_ppo` function performs several epochs of PPO updates based on the stored transitions in the buffer:

- **Advantages and Returns**: Advantages are calculated as the difference between the returns (computed using Generalized Advantage Estimation or GAE) and the critic's value estimates. The critic provides feedback on the quality of the state, while the advantage quantifies how much better the action performed relative to expectations.
- **Optimization**: The total loss is a combination of actor and critic losses. The actor is updated using the PPO loss function, while the critic is updated by minimizing the Mean Squared Error (MSE) between the predicted values and the returns.

**Training Loop:**

The training loop is structured as follows:

1. **Reset Environment**: At the beginning of each episode, the environment is reset to the initial state with a starting balance of capital.

2. **Action Selection**: For each time step, the actor part of the network generates action probabilities. The agent samples an action from this probability distribution, executes the action in the environment, and receives a reward.

3. **Buffer Storage**: Each transition, including the state, action, log probabilities, reward, value, and done flag, is stored in the PPO buffer.

4. **PPO Update**: After the episode, PPO optimization is performed based on the transitions stored in the buffer. The actor and critic are both updated to improve action selection and value estimation.

5. **Performance Logging**: Metrics like total profit, Sharpe ratio, and the number of trades are tracked at the end of each episode. The Sharpe ratio is calculated to assess the agent's risk-adjusted performance.

**Generalized Advantage Estimation (GAE):**

The **compute_gae** function calculates advantages and returns using Generalized Advantage Estimation (GAE). This technique reduces variance in the estimation of advantages by balancing the rewards received and the value estimates from the critic, which smooths the learning process.

**Performance Metrics:**

At the end of each episode, the following metrics are calculated and logged:

- **Total Profit**: The agent's cumulative profit during the episode.
- **Sharpe Ratio**: A measure of the agent's risk-adjusted returns, helping evaluate the trading strategy in relation to its volatility.
- **Number of Trades**: The number of Buy/Sell actions executed by the agent.

**Sequential steps for the PPO algorithm**

- *Initialize Hyperparameters:*
  - Set initial balance, state size, action size, learning rate, gamma, epsilon, PPO epochs, batch size.
- *Define Actor-Critic Network:*
  - Create a neural network with shared layers.
  - Actor outputs action probabilities.
  - Critic outputs state value.
- *Define PPO Buffer:*

  - Create a buffer to store states, actions, log_probs, rewards, and values.

- *Define Trading Environment:*
  - Manage balance, crypto holdings, and net worth.
  - Track net worth history and step through the environment based on actions.
  - Return next state, reward, and done flag after each action.

- *Compute GAE:*
  - Calculate advantages and returns using rewards, values, and dones.
  - Apply Generalized Advantage Estimation (GAE).

- *PPO Loss:*
  - Compute the ratio of new and old log probabilities.
  - Apply PPO clipping to calculate the actor loss.
  - Compute critic loss based on MSE between values and returns.

- *Main Training Loop:*
  - For each episode:
    * Reset environment and buffer.
    * For each step:
      · Select action using the Actor network.
      · Perform action in environment and store transition in buffer.
    * After the episode ends, train PPO using stored trajectories.
    * Track and print profit, Sharpe ratio, and number of trades.

The table 4.5 describes and specifies all the parameters used for developing and then training and testing the PPO model on BTC and ETH data. The table can be used to implement and reproduce the results obtained in this study.

| Parameter | Value | Description |
|---|---|---|
| initial_balance | 10,000 | Starting capital for trading. |
| state_size | 33/34 | Number of features representing the state (BTC & ETH data). |
| action_size | 3 | Number of possible actions: Buy, Sell, Hold. |
| learning_rate | 0.00001 | Learning rate for the Adam optimizer. |
| gamma | 0.99 | Discount factor for future rewards. |
| epsilon | 0.2 | Clipping threshold for PPO update. |
| ppo_epochs | 4 | Number of training epochs per PPO update. |
| batch_size | 128 | Number of samples per training batch. |
| episode_length | 720 | Number of steps per episode (e.g., 30 days * 24 hours/day). |
| **Neural Network Architecture** | | |
| fc1 (Layer 1) | $33 \rightarrow 64$ | First fully connected layer from state to 64 neurons. |
| fc2 (Layer 2) | $64 \rightarrow 64$ | Second fully connected layer with 64 neurons. |
| Actor Head | $64 \rightarrow 3$ | Outputs action probabilities (Buy, Sell, Hold). |
| Critic Head | $64 \rightarrow 1$ | Outputs the scalar value for the current state. |
| **Trading Environment Parameters** | | |
| trade_fraction | 0.25 | Fraction of balance or holdings to trade (25%). |
| reward_penalty_hold | -0.01 | Penalty for holding action to encourage more active trading. |
| reward_bonus_action | +0.005 | Bonus reward for taking Buy or Sell actions. |
| risk_free_rate | 0.0 | Risk-free rate used in Sharpe ratio calculation. |

**Table 4.5** – The parameters in the PPO implementation

### 4.2.5 Deep Deterministic Policy Gradient

**Actor and Critic Networks:**

The DDPG algorithm uses two neural networks:

- **Actor Network**: Responsible for outputting actions given the current state. In this case, the actor outputs a single continuous action representing a trading decision between -1 and 1, where:
  - Negative values represent selling,
  - Positive values represent buying,
  - Zero represents holding.
- **Critic Network**: The critic estimates the **Q-value**, which represents the expected future reward given the current state and action. The critic uses two input streams: one for the state and one for the action. These streams are combined in later layers to output the Q-value.

Both the actor and critic networks contain fully connected layers (`fc1`, `fc2` for state processing and `fc3` for action processing in the critic), and the output layer for the critic is a single value (Q-value). The actor's final layer is scaled between the action space limits (i.e., between -1 and 1).

**Target Networks:**

To stabilize learning, **target networks** (one for the actor and one for the critic) are used. These target networks are soft copies of the main actor and critic networks and are updated using a soft update rule:

$$\text{target} = \tau \times \text{source} + (1 - \tau) \times \text{target}$$

This slow update process ($\tau = 0.001$) ensures that the target networks evolve gradually, preventing the training from oscillating.

**Ornstein-Uhlenbeck Noise:**

To promote exploration in continuous action spaces, **Ornstein-Uhlenbeck (OU) noise** is added to the actor's actions. This noise has a temporal correlation, making it more suitable for environments with inertia (such as financial markets). The noise ensures that the agent explores a variety of actions while still being able to refine its policy based on observed rewards.

**Replay Buffer:**

The **Replay Buffer** stores transitions (state, action, reward, next state, done). This allows the agent to learn from a variety of experiences, not just the most recent ones, breaking the correlation between consecutive updates and improving stability. During training, mini-batches of transitions are randomly sampled from the replay buffer to update the actor and critic networks.

**DDPG Update Process:**

The training process updates both the actor and critic networks:

- **Critic Update**: The critic is updated by minimizing the difference between the Q-values predicted by the critic network and the Q-targets, calculated using the target networks:

  $$\text{Q-target} = \text{reward} + \gamma \times \text{Q-value(next state, next action)} \times (1 - \text{done})$$

  The loss function for the critic is the Mean Squared Error (MSE) between the Q-target and the predicted Q-values.

- **Actor Update**: The actor is updated by maximizing the Q-value predicted by the critic (i.e., finding actions that yield higher rewards). This is done by minimizing the negative Q-values output by the critic for the actor's predicted actions:

  $$\text{Actor Loss} = -\text{mean}(Q(\text{state}, \text{actor}(\text{state})))$$

After updating both networks, a **soft update** of the target networks is performed to ensure they closely track the main networks.

**Trading Environment**

The `CryptoTradingEnv` class simulates the cryptocurrency trading environment:

- **State**: The state is derived from market data (e.g., BTC/ETH prices and features). This state is normalized before being fed into the networks.
- **Actions**: The agent selects a continuous action in the range [-1, 1], where:
  - Positive values represent buying cryptocurrency.
  - Negative values represent selling cryptocurrency.
  - The magnitude of the action controls how much of the balance or holdings to trade.
- **Rewards**: The reward is based on changes in the agent's net worth after taking an action. A small penalty is applied for small actions to encourage the agent to avoid making inconsequential trades.
- **Net Worth**: The net worth is updated at each step based on the agent's actions (buying/selling) and the market price.

The environment also includes a function to compute the **Sharpe Ratio**, a key performance metric for evaluating the risk-adjusted returns of the agent's strategy.

**Training Loop**

The training loop is structured as follows:

1. The environment is reset, and the agent begins interacting with the market.
2. At each step:
   - The actor network predicts an action based on the current state.
   - Ornstein-Uhlenbeck noise is added to the action to encourage exploration.
   - The action is executed in the environment, and the agent receives the next state, reward, and a done signal indicating whether the episode has ended.
   - The transition is stored in the replay buffer.
   - The actor and critic networks are updated using the DDPG algorithm.
3. After each episode, the performance metrics (total profit, Sharpe ratio, number of trades) are logged for evaluation.

**Sequential steps for the DDPG algorithm**

- *Initialize Hyperparameters:*
  - Set initial balance, state size, action size, learning rate, gamma, tau, batch size.
- *Define Actor and Critic Networks:*

  - Actor outputs continuous actions between $[-1, 1]$.
  - Critic estimates Q-value for state-action pairs.

- *Target Networks:*
  - Initialize target actor and target critic networks.
  - Copy weights from actor and critic to target networks.

- *Define Replay Buffer:*
  - Store transitions in buffer: states, actions, rewards, next_states, dones.

- *Ornstein-Uhlenbeck Noise:*
  - Generate noise to explore the action space during training.

- *Soft Update of Target Networks:*
  - Perform soft update for target networks using parameter $\tau$.

- *DDPG Update Function:*
  - Update the critic using the Q-learning target.
  - Update the actor by maximizing the critic's output.
  - Perform soft update of target networks.

- *Define Trading Environment:*
  - Manage balance, crypto holdings, and net worth.
  - Take action in environment and receive next state, reward, and done flag.

- *Main Training Loop:*
  - For each episode:
    - Reset environment.
    - For each step:
      - Select action using actor network.
      - Add noise to the action for exploration.
      - Perform action in environment and store transition in replay buffer.
      - Update the networks using DDPG.
    - Track and print profit, Sharpe ratio, and number of trades.

The table 4.6 describes and specifies all the parameters used for developing and then training and testing the DDPG model on BTC and ETH data. The table can be used to implement and reproduce the results obtained in this study.

| **Parameter** | **Value** | **Description** |
|---|---|---|
| initial_balance | 10,000 | Starting capital for trading. |
| state_size | 33/34 | Number of features representing the state (BTC and ETH Data). |
| learning_rate | 0.005 | Learning rate for the Adam optimizer. |
| gamma | 0.99 | Discount factor for future rewards. |
| tau | 0.001 | Soft update parameter for the target networks. |
| batch_size | 128 | Number of samples per training batch. |
| episode_length | 720 | Number of steps per episode. |
| num_episodes | 18 | Total number of training episodes. |
| replay_buffer_capacity | 100,000 | Maximum capacity of the replay buffer. |
| action_size | 1 | The number of continuous actions output by the actor network. |
| ou_theta | 0.15 | Theta parameter for Ornstein-Uhlenbeck noise. |
| ou_sigma | 0.2 | Sigma parameter for Ornstein-Uhlenbeck noise. |
| **Neural Network Architecture** | | |
| Actor: fc1 | $34 \rightarrow 64$ | First fully connected layer of the actor network. |
| Actor: fc2 | $64 \rightarrow 64$ | Second fully connected layer of the actor network. |
| Actor: fc3 | $64 \rightarrow 1$ | Output layer of the actor network (continuous action). |
| Critic: fc1 | $34 \rightarrow 64$ | First fully connected layer for state processing in the critic. |
| Critic: fc2 | $64 \rightarrow 64$ | Second fully connected layer for state processing in the critic. |
| Critic: fc3 | $1 \rightarrow 64$ | Action processing layer in the critic network. |
| Critic: fc4 | $128 \rightarrow 64$ | Combined state and action processing layer in the critic. |
| Critic: fc5 | $64 \rightarrow 1$ | Output layer of the critic (Q-value). |

**Table 4.6** – The parameters in the DDPG implementation

## 4.3 Post-training and Storage of results

The list of episode metrics, which include total profit, net worth, Sharpe ratio, and number of trades, is converted into a Pandas DataFrame for easy analysis and this data is saved to a CSV file. The key performance metrics such as average profit and average Sharpe ratio over all episodes and cumulative return over the 12 episodes or annual return are calculated, providing a measure of the agent's recent performance. The results offer a summary of the agent's profitability and risk-adjusted return. This is done for each algorithm while training it on both BTC and ETH data. This process ensures a systematic evaluation of all 4 agents' trading strategies and facilitates deeper analysis for further optimization.

## 4.4 Evaluation Metrics

### 4.4.1 Average Return

The average return refers to the arithmetic mean of returns over a certain period, generally used to evaluate the overall performance of an investment over time. It is calculated by summing all individual returns and dividing by the number of periods.

$$\text{Average Return} = \frac{1}{n}\sum_{i=1}^{n} r_i$$

Where: $r_i$ is the return in the $i$-th period, and $n$ is the total number of periods.

This formula provides a simple measure of the average performance of an investment over time.

In this experiment, the different DRL algorithms like DQN, PPO, A2C and DDPG are trained over 100, 200 and 500 episodes respectively where each episode is 1 month or 30 days. The results obtained from training will be the **average monthly return**.

### 4.4.2 Annual Return

The annual return will be calculated from the average monthly returns obtained from training the model. The monthly return will be annualized using the geometric average which helps with the compounding effect of returns over time.

Given the average monthly return $r_{\text{avg monthly}}$, the formula to calculate the annual return is:

$$r_{\text{annual}} = (1 + r_{\text{avg monthly}})^{12} - 1$$

Where:

- $r_{\text{avg monthly}}$ is the average monthly return.
- 12 is the number of months in a year (since returns are annualized over 12 months).

If we do not account for compounding, the arithmetic average approach simply multiplies the average monthly return by 12. The formula for this would be:

$$r_{\text{annual}} = r_{\text{avg monthly}} \times 12$$

This method, however, can lead to an overestimation of the actual annual return, especially when there is volatility, as it does not account for the compounding effect of returns over time.

### 4.4.3 Cumulative Return

Cumulative return represents the total return on an investment over a specified period, expressed as a percentage. It measures the overall gain or loss of an investment from the starting point to the endpoint. Unlike average return, cumulative return focuses on the total growth over time without considering how long it took to achieve that growth.

For a series of periodic returns $r_1, r_2, \ldots, r_n$, the cumulative return $R_{\text{cumulative}}$ is calculated as:

$$R_{\text{cumulative}} = (1 + r_1)(1 + r_2) \ldots (1 + r_n) - 1$$

Alternatively, if the initial value of the investment is $V_0$ and the final value after $n$ periods is $V_n$, the cumulative return can be calculated as:

$$R_{\text{cumulative}} = \frac{V_n - V_0}{V_0} = \frac{V_n}{V_0} - 1$$

Where:

- $r_i$ is the return for period $i$,
- $V_0$ is the initial value of the investment,
- $V_n$ is the final value of the investment after $n$ periods.

### 4.4.4 Sharpe Ratio

The Sharpe ratio measures an investment's return relative to its risk, evaluating how much excess return is generated for each unit of risk taken. A higher Sharpe ratio indicates better risk-adjusted returns, while a lower ratio suggests the returns may not justify the level of risk. It adjusts returns by subtracting the risk-free rate and dividing by the portfolio's standard deviation, offering a standardized way to compare investments. Additionally, the Sharpe ratio can be refined by incorporating the turbulence index, which adjusts for market volatility and risk aversion during evaluation.

The Sharpe ratio is calculated as:

$$\text{Sharpe ratio} = \frac{\mathbb{E}[r_p] - r_f}{\sigma_p}$$

where:

- $\mathbb{E}[r_p]$ is the expected portfolio return,
- $r_f$ is the risk-free rate, and
- $\sigma_p$ is the portfolio's standard deviation.

In this study, average monthly return,annual return, cumulative return and shape ratio will be calculated from average monthly returns to compare the results obtained here with other reputable studies in this field of research.

## 4.5 XAI integration for feature importance

It is very important to know the subset of features which play a vital role in helping the agent in the DRL algorithms make their decision i.e we are interested in which features predominantly led the agent to decide whether to buy or sell the asset. XAI methods will be integrated into the best-performing DRL algorithm to find the most important and the least important features for both BTC and ETH.

### 4.5.1 Integrated Gradients

This enhances the algorithm by integrating a feature attribution method ( to analyze which features were most important in driving the agent's decisions. By collecting feature importance scores alongside the agent's performance (net worth), insights can be gained into how the agent utilizes input features (e.g., market indicators) during trading. The post-processing step helps identify trends in feature importance across episodes, providing actionable insights for improving the trading model or understanding the market dynamics.

We compute the importance of each input feature for the agent's decision-making process. The method works by:

- Taking the difference between the current state and a baseline (a zero tensor by default).
- Computing gradients for several interpolated states between the baseline and the input.
- Accumulating these gradients and averaging them to estimate the contribution of each feature.

This method provides an interpretable measure of how much each input feature contributed to the agent's chosen action at a given state.

**Training Loop with Feature Importance Collection:**

In the main training loop, feature importance is collected alongside the standard DDPG updates:

- **Feature Importance Initialization**: For each episode, an array (`episode_feature_importances`) is initialized to accumulate feature importance scores over all time steps in the episode.
- **Actor Action with Noise**: The agent selects an action using the actor network, adding Ornstein-Uhlenbeck noise for exploration. The action is clipped within the valid range $[-1, 1]$.

- **Integrated Gradients Calculation**: After every action, the `integrated_gradients` function is called to compute the importance of the features in the current state. These gradients are accumulated across the episode.
- **Episode Completion**: Once the episode is complete, the average feature importance across all steps in the episode is calculated and stored, alongside the final net worth for that episode.

**Storing Results:**

At the end of each episode, two key outputs are saved:

- **Feature Importances**: The average feature importances across all steps in the episode are stored in `feature_importances_per_episode`.
- **Net Worth**: The agent's final net worth at the end of the episode is stored in `net_worth_per_episode`.

These results are then organized into a Pandas DataFrame and saved to a CSV file

**Post-Processing: Analyze Feature Importance:**

After the training loop, post-processing is performed to identify the most important features for each episode:

- **Most Important Features**: For each episode, the top 33/34 features (based on their average importance) are identified using `np.argsort`.
- **Mapping to Feature Names**: The indices of the important features are mapped back to their actual feature names from the `filtered_data` DataFrame, providing interpretable insights.
- **Results Storage**: The most important features, along with the agent's net worth for each episode, are saved into another CSV file.

### 4.5.2 Saliency Map

A saliency map for the input features in a DDPG (Deep Deterministic Policy Gradient) model is calculated and visualized, providing insights into which features most influence the agent's decisions. Saliency maps help identify how sensitive the agent's action is to changes in each input feature, thereby revealing which features (such as market indicators) are the most significant in driving the agent's trading decisions.

The process begins by converting the current state into a PyTorch tensor (‘state_tensor‘), which is then passed to the actor-network. The actor outputs an action, which represents the agent's decision to either buy, sell, or hold cryptocurrency, based on the input state. The key part of this process is setting ‘requires_grad=True‘ for the state tensor, enabling the computation of gradients with respect to the input features. By calling ‘backward()‘, the code calculates the gradient of the action with

respect to each input feature. These gradients form the saliency map, where larger gradient values indicate that a feature has a greater influence on the agent's action.

The computed saliency map is then visualized using a bar chart. Each bar in the chart corresponds to a feature, and the height of the bar represents the magnitude of that feature's impact on the action taken by the agent. This visualization helps identify which features are driving the agent's decisions, offering insights into the model's behavior and helping assess which market indicators are the most relevant for trading decisions. By analyzing these saliency maps easier to interpret and improve the performance of the DDPG agent in cryptocurrency trading.

# Chapter 5

# Results

In this chapter, the performances of each DRL model are presented, explained and visually represented for better understanding. The results of the best-performing model are compared against models from other notable studies in this domain. Lastly, the results and the key insights obtained from the XAI layer are also explained and visually represented for better understanding.

## 5.1 Deep Q-Network

| Asset | Episodes | Average Profit (%) | Maximum Profit (%) | Annual Profit (%) | Average Sharpe | Maximum Sharpe | Average Trades |
|---|---|---|---|---|---|---|---|
| BTC | 100 | 3.74 | 29.54 | 44.92 | 0.024 | 0.111 | 505 |
| | 200 | 4.92 | 45.40 | 59.05 | 0.023 | 0.114 | 530 |
| | 500 | 6.06 | 50.98 | 72.78 | 0.025 | 0.109 | 514 |
| ETH | 100 | 3.45 | 35.53 | 41.41 | 0.016 | 0.116 | 500 |
| | 200 | 4.45 | 45.65 | 53.88 | 0.015 | 0.107 | 520 |
| | 500 | 4.53 | 60.01 | 54.36 | 0.015 | 0.129 | 417 |

**Table 5.1** – DQN trained with OHLCV and Blockchain data - Fractional Trading 25%

**Figure 5.1** – BTC - DQN

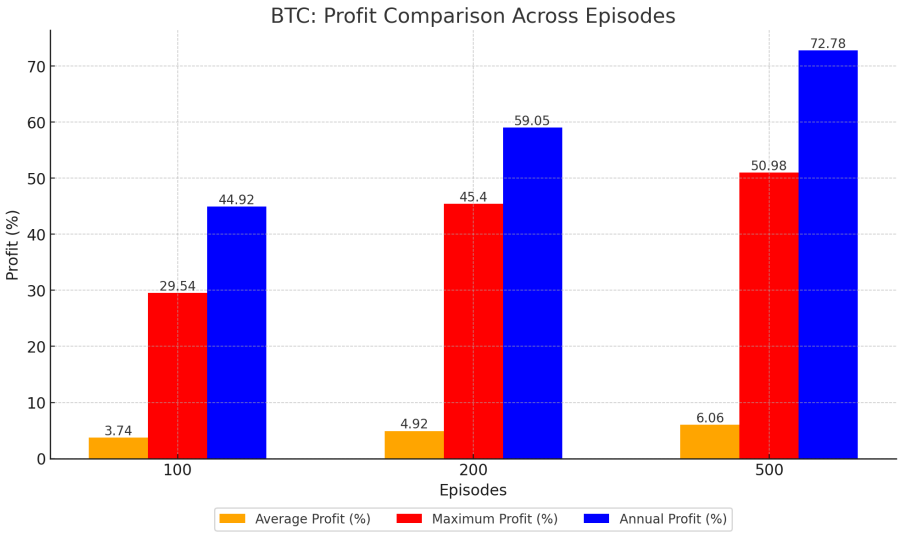



**Figure 5.2** – BTC - DQN - Sharpe ratio

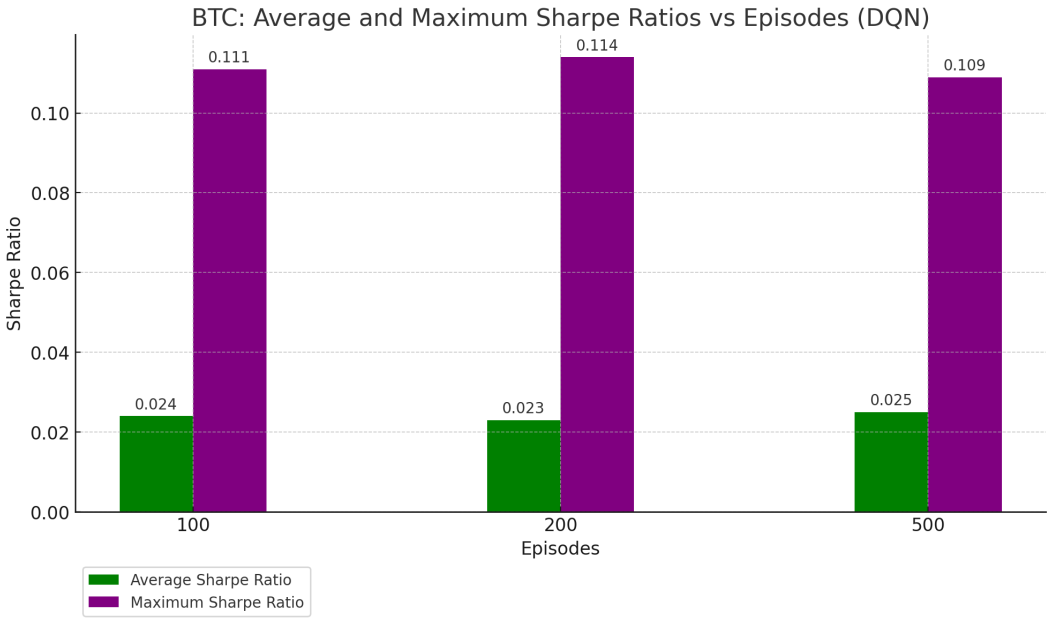


**Figure 5.3** – ETH - DQN

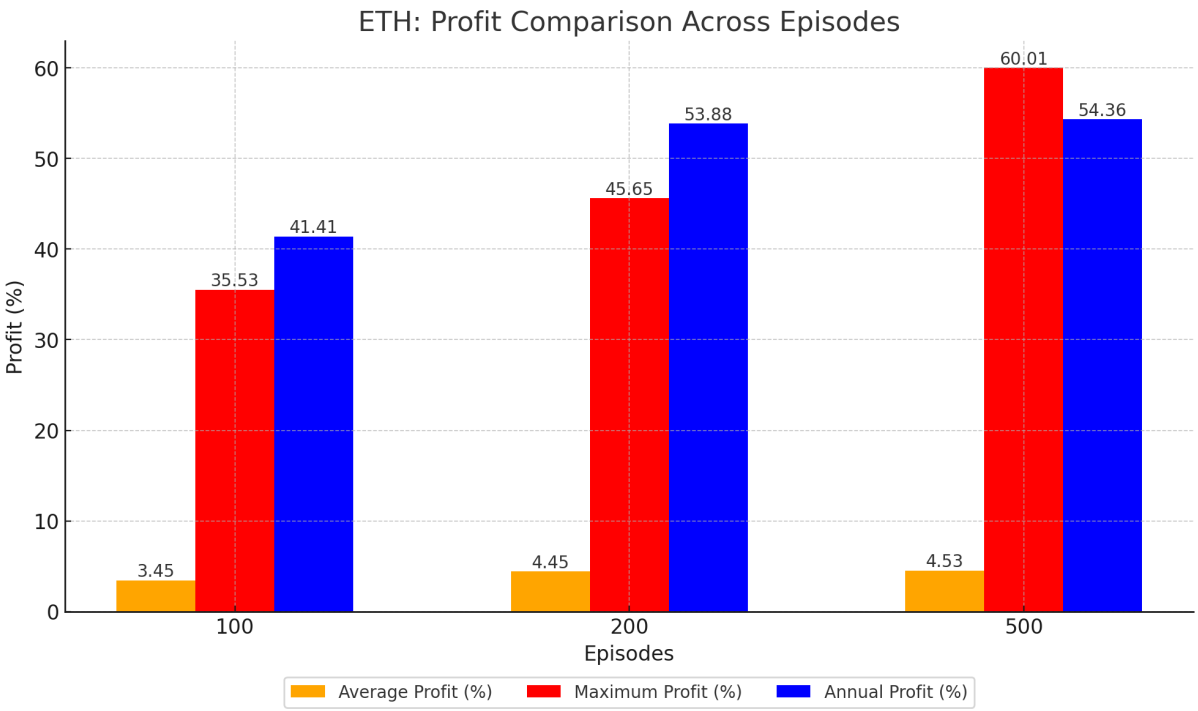



**Figure 5.4** – ETH - DQN - Sharpe ratio

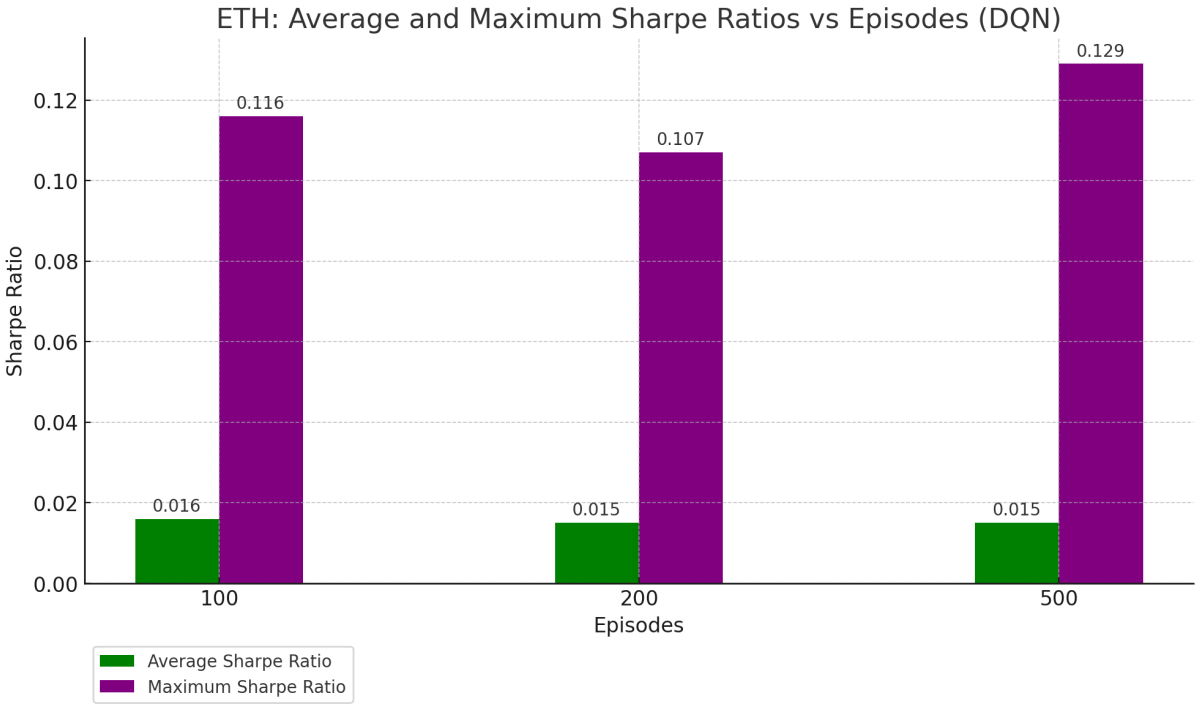
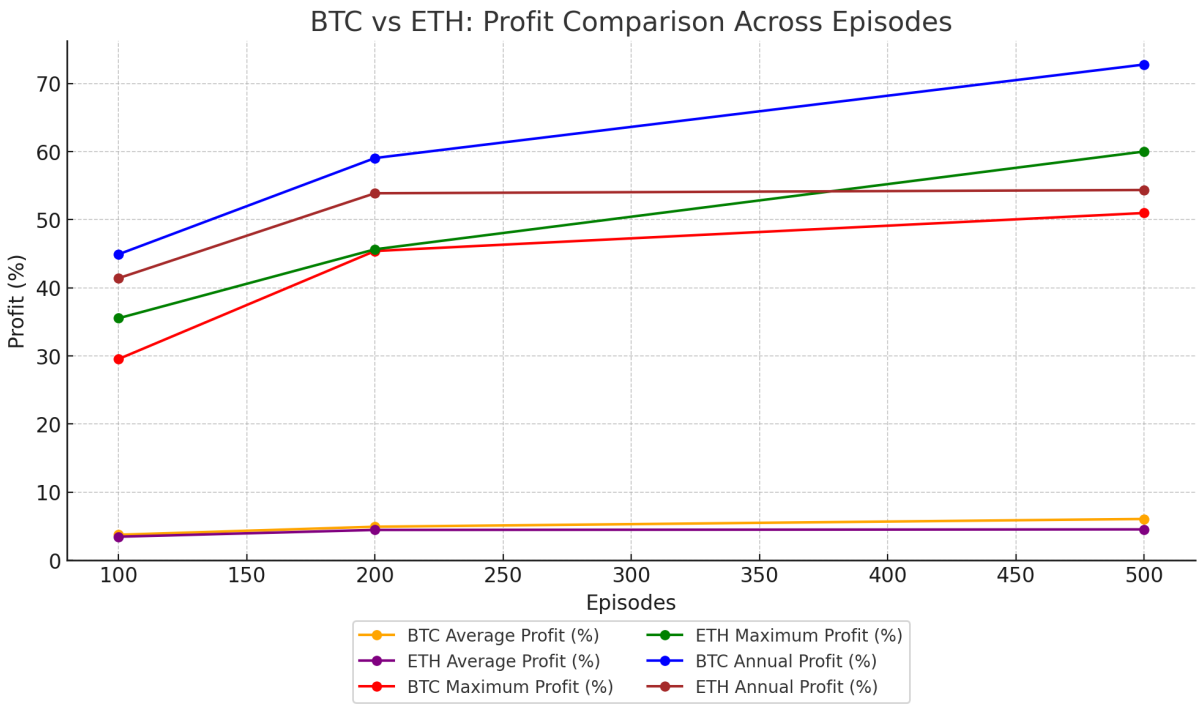


**Figure 5.5** – DQN Performance: BTC vs ETH




The table 5.1, fig. 5.1, fig. 5.2, fig. 5.3, and fig. 5.4 compares the performance of the DQN model trained on BTC and ETH data across 100, 200, and 500 episodes, highlighting key metrics such as average profit, maximum profit, annual profit, Sharpe ratios, and average trades. In fig. 5.1, BTC demonstrates a steady increase in profitability with longer training, with average profit rising from 3.74% to 6.06% and annual profit reaching 72.78% after 500 episodes. In fig. 5.2, BTC also maintains a stable Sharpe ratio, indicating relatively consistent risk-adjusted returns, though maximum Sharpe slightly declines after 200 episodes. Trading activity for BTC remains high, with average trades staying above 500. In fig. 5.3 ETH, on the other hand, shows a more modest improvement in profits, with average profit reaching 4.53% and annual profit peaking at 54.36%. In fig. 5.4 while ETH's maximum profit grows significantly, its average Sharpe ratio remains lower than BTC's, suggesting less favorable risk-adjusted returns. Interestingly, ETH sees a reduction in trading activity over time, with average trades decreasing to 417 by the 500th episode, potentially reflecting a more refined trading strategy. From fig. 5.5, it can be seen that overall, BTC outperforms ETH in terms of both profitability and risk-adjusted returns as the number of episodes increases.

### 5.1.1 Fractional Trading

| Asset | Fractional Trade (%) | Average Profit (%) | Maximum Profit (%) | Annual Profit (%) | Average Sharpe | Maximum Sharpe | Average Trades |
|---|---|---|---|---|---|---|---|
| BTC | 10 | 4.25 | 30.99 | 50.98 | 0.026 | 0.103 | 480 |
| | 20 | 5.42 | 33.79 | 65.03 | 0.025 | 0.102 | 516 |
| | 25 | 3.74 | 29.54 | 44.92 | 0.024 | 0.111 | 505 |
| | 33 | 4.34 | 30.32 | 52.09 | 0.022 | 0.111 | 510 |
| | 50 | 4.31 | 34.90 | 51.72 | 0.021 | 0.108 | 477 |
| ETH | 10 | 2.87 | 29.90 | 34.44 | 0.016 | 0.112 | 514 |
| | 20 | 3.52 | 33.44 | 42.21 | 0.015 | 0.116 | 422 |
| | 25 | 3.45 | 35.53 | 41.41 | 0.016 | 0.116 | 500 |
| | 33 | 4.71 | 41.44 | 56.56 | 0.018 | 0.094 | 510 |
| | 50 | 4.68 | 50.04 | 56.15 | 0.017 | 0.107 | 479 |

**Table 5.2** – DQN - Fractional Trading % Comparison

**Figure 5.6** – BTC - DQN - Fractional Trading

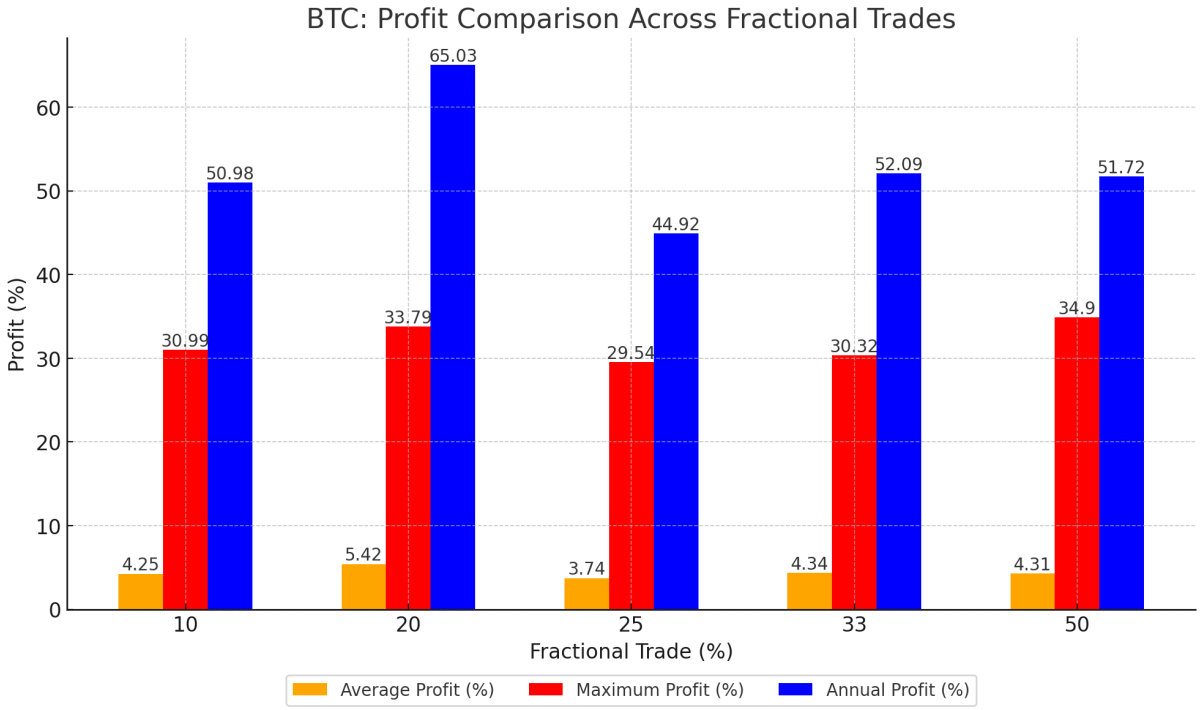


**Figure 5.7** – ETH - DQN - Fractional Trading

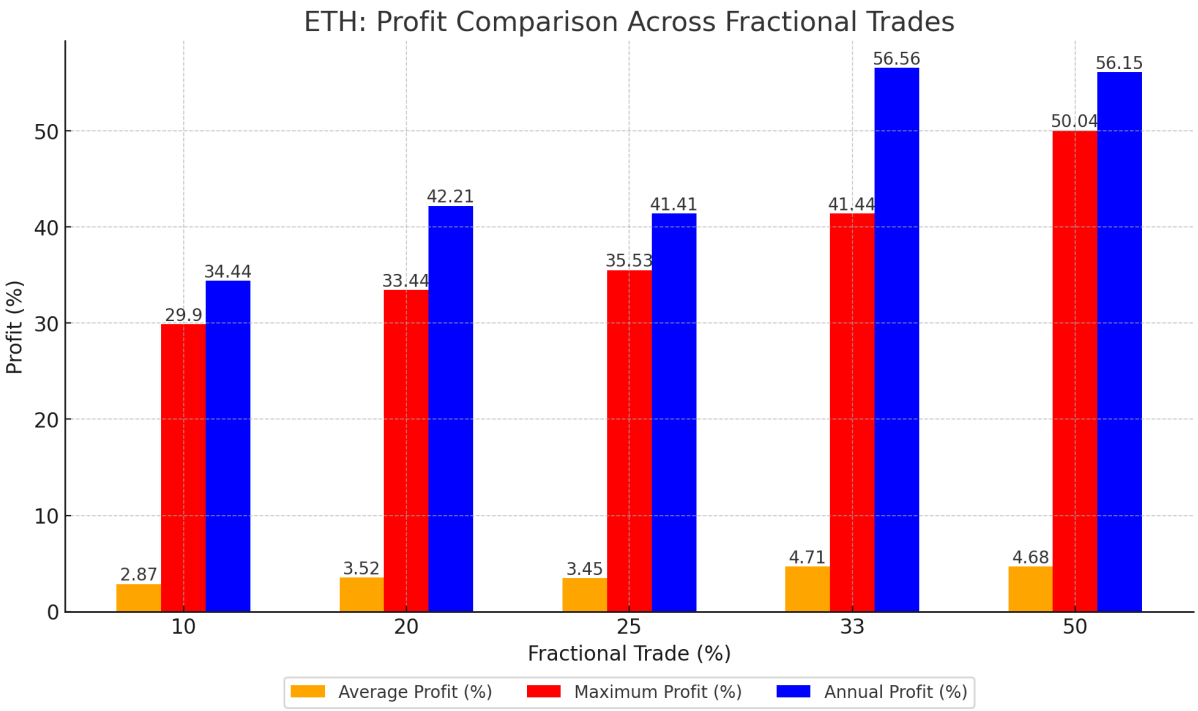
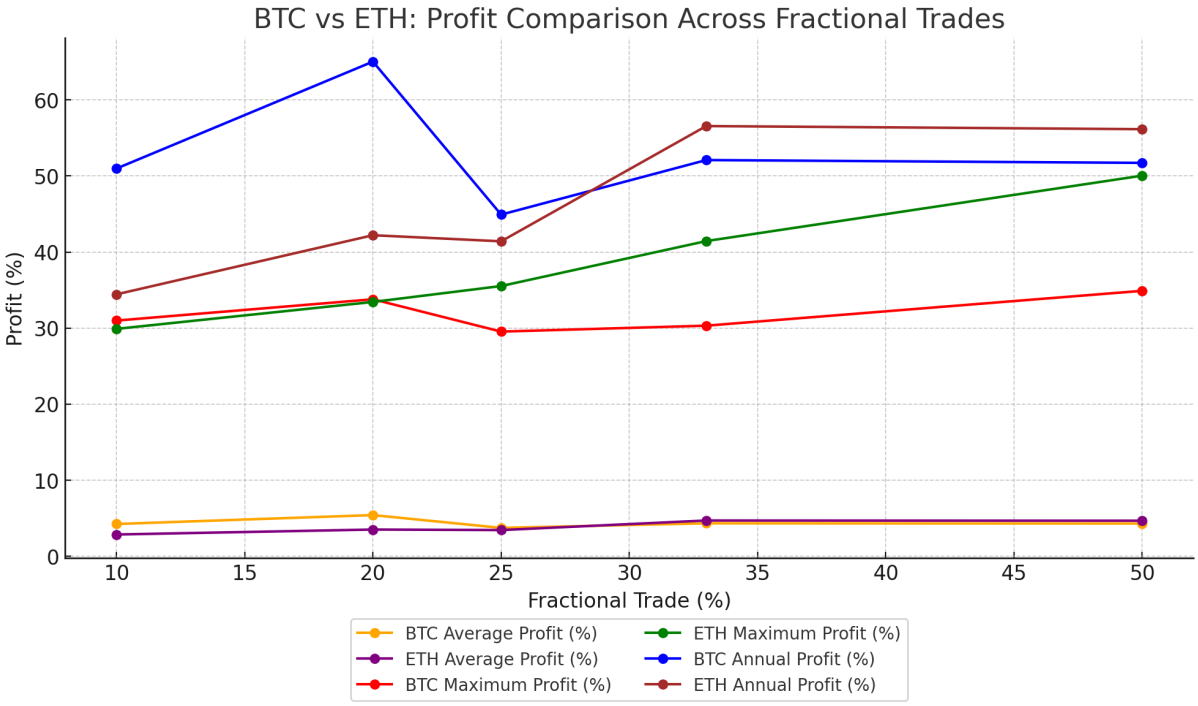



**Figure 5.8** – DQN - Fractional Trading: BTC vs ETH



The table 5.2, fig. 5.6, and fig. 5.7 provides performance data of the DQN model trained on BTC and ETH across different Fractional Trade (%) levels, showing metrics such as Average Profit (%), Maximum Profit (%), Annual Profit (%), Average Sharpe Ratio, Maximum Sharpe Ratio, and Average Trades.

In fig. 5.6, it can be seen that for BTC, Average Profit shows moderate variability, ranging from 3.74% to 5.42% as the fractional trade level increases. Maximum Profit follows a more stable upward trend, peaking at 34.90%. Annual Profit reaches its highest at 65.03% when fractional trade is 20%. However, the Sharpe Ratios (both average and maximum) stay relatively flat, with Average Sharpe decreasing slightly at higher fractional trade levels. Average Trades fluctuate between 477 and 516, showing minor changes in trading frequency.

In fig. 5.7, it can be seen that For ETH, Average Profit steadily increases from 2.87% to 4.71% as fractional trade level grows, with Maximum Profit peaking at 50.04%. Annual Profit shows substantial improvement, reaching 56.56% and 56.15% at higher fractional trade levels. Similar to BTC, the Sharpe Ratios are stable but lower overall than BTC. Average Trades decrease significantly at higher fractional trades, which may suggest a more refined trading strategy for ETH.

The fig. 5.8 provides a comparison between BTC and ETH for the same metrics across different Fractional Trade (%) levels. It highlights the trends and differences between the two cryptocurrencies' performance over varying trade fractions.

## 5.2 Advantage Actor Critic

| Asset | Episodes | Average Profit (%) | Maximum Profit (%) | Annual Profit (%) | Average Sharpe | Maximum Sharpe | Average Trades |
|---|---|---|---|---|---|---|---|
| BTC | 100 | 4.30 | 31.46 | 51.65 | 0.025 | 0.123 | 456 |
| | 200 | 4.22 | 34.27 | 50.60 | 0.023 | 0.119 | 455 |
| | 500 | 4.91 | 33.64 | 58.94 | 0.023 | 0.107 | 523 |
| ETH | 100 | 4.31 | 33.96 | 51.74 | 0.018 | 0.113 | 424 |
| | 200 | 4.40 | 61.74 | 52.81 | 0.014 | 0.099 | 658 |
| | 500 | 4.85 | 24.32 | 58.18 | 0.029 | 0.109 | 468 |

**Table 5.3** – A2C trained with OHLCV and Blockchain data - Fractional Trading 25%

**Figure 5.9** – BTC - A2C

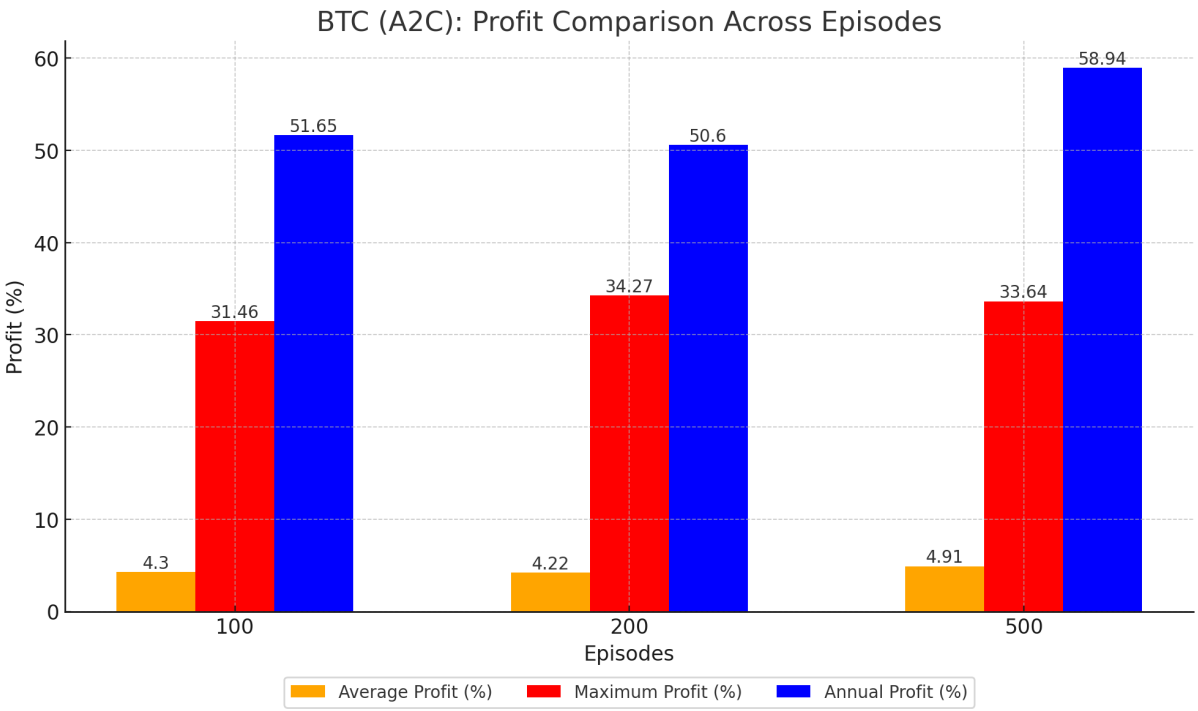



**Figure 5.10** – BTC - A2C - Sharpe ratio

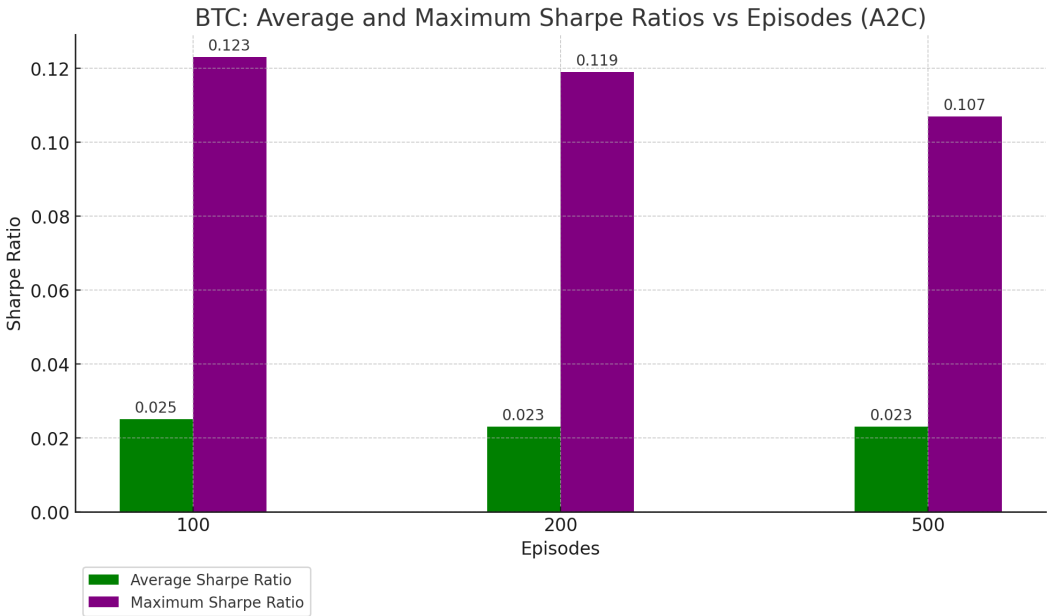


**Figure 5.11** – ETH - A2C

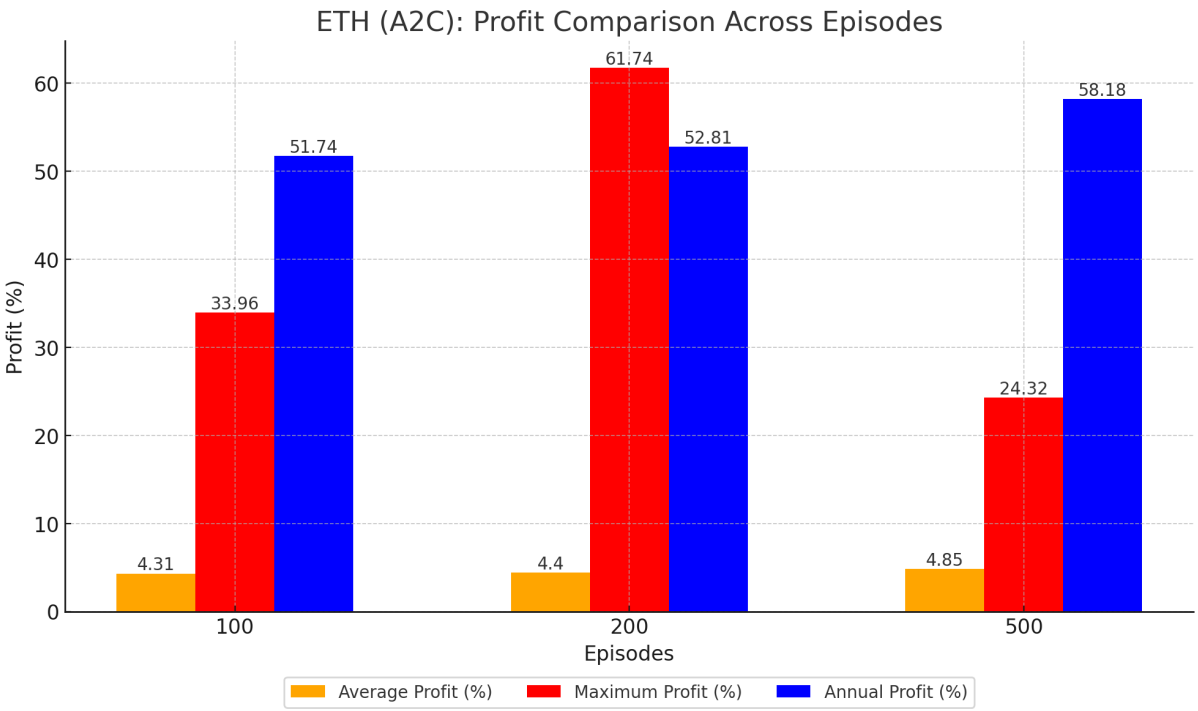



**Figure 5.12** – ETH - A2C - Sharpe ratio

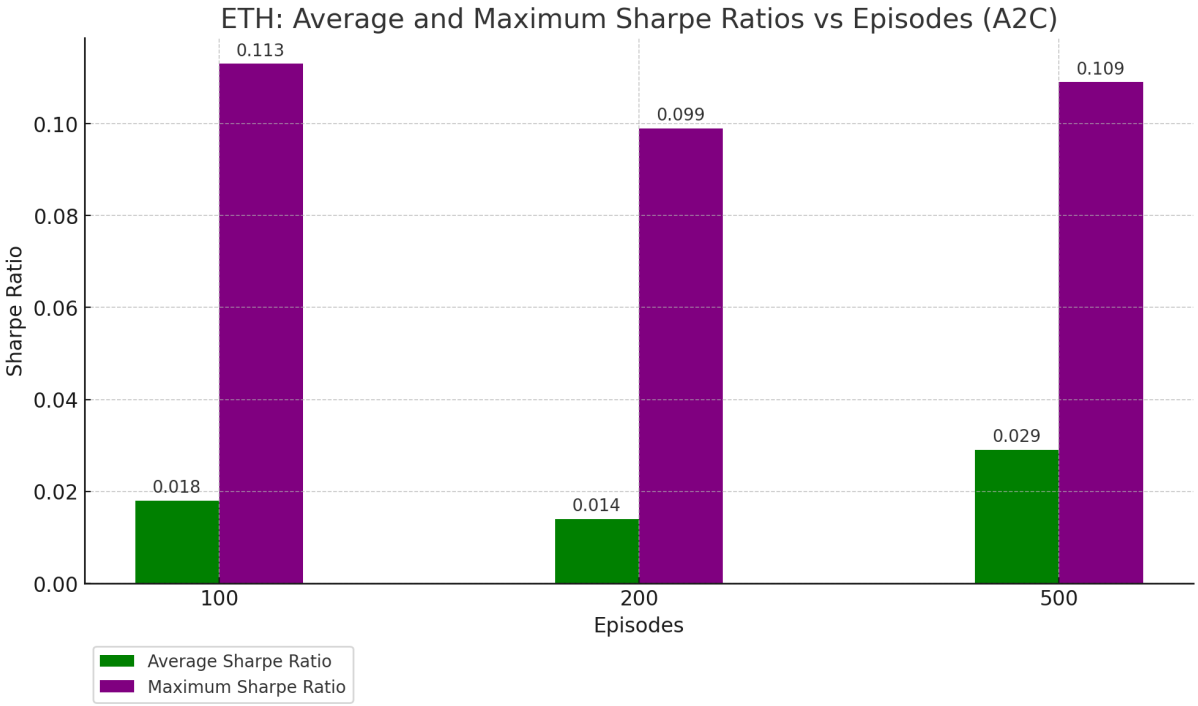
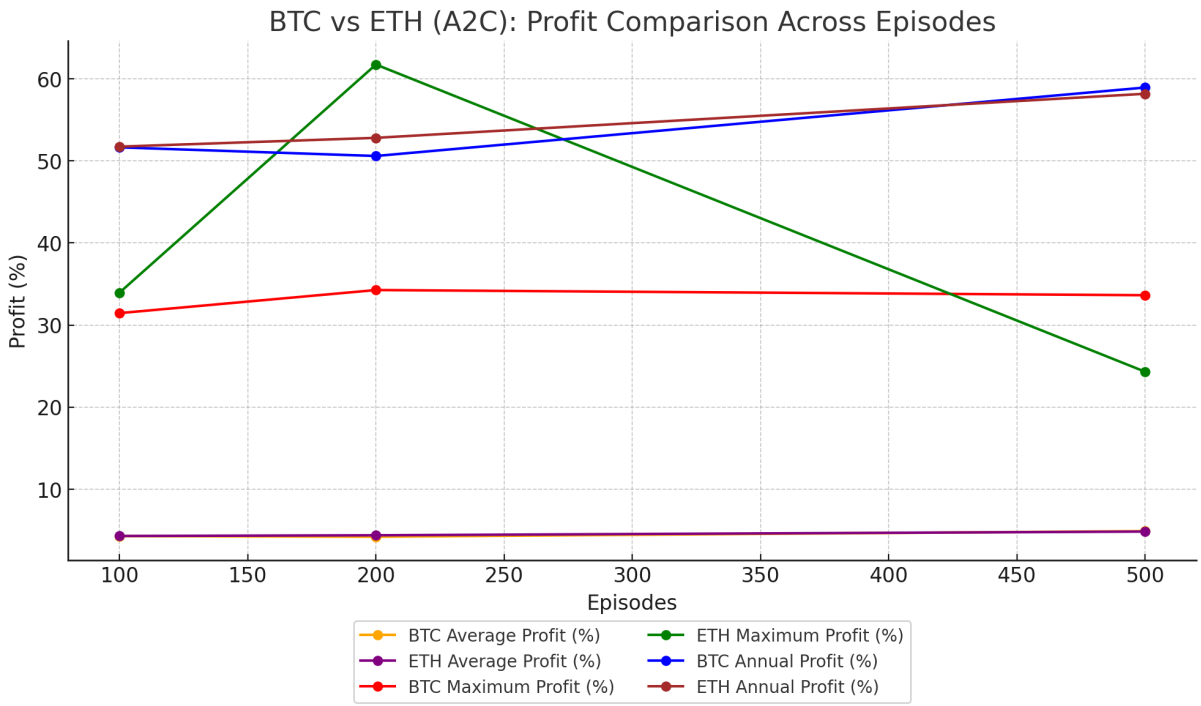


**Figure 5.13** – A2C Performance: BTC vs ETH




The table 5.3, fig. 5.9, fig. 5.10, fig. 5.11, and fig. 5.12 compares performance metrics for the A2C (Advantage Actor-Critic) model trained on BTC and ETH data across 100, 200, and 500 episodes, with key metrics such as Average Profit (%), Maximum Profit (%), Annual Profit (%), Average Sharpe Ratio, Maximum Sharpe Ratio, and Average Trades.

In fig. 5.9, for BTC, the Average Profit is relatively stable, peaking at 4.91% after 500 episodes, while Maximum Profit fluctuates slightly, peaking at 34.27% after 200 episodes. Annual Profit increases over time, reaching 58.94% after 500 episodes, indicating that profitability improves with more training. In fig. 5.10, the Sharpe Ratios are stable, with minor changes, and the number of trades increases from 456 to 523 as episodes increase.

In fig. 5.11, for ETH, the Average Profit improves over time, reaching 4.85% after 500 episodes. Interestingly, Maximum Profit fluctuates significantly, peaking at 61.74% after 200 episodes but dropping to 24.32% after 500 episodes. Annual Profit stays relatively stable, peaking at 58.18% after 500 episodes. In fig. 5.12, ETH's Sharpe Ratios show more variability, with the highest Average Sharpe Ratio of 0.029 occurring after 500 episodes. Average Trades fluctuate considerably, especially in the 200-episode mark where it peaks at 658, suggesting more trading activity at this point.

The fig. 5.13 compares BTC and ETH across the same metrics (average profit, maximum profit, and annual profit) over the episodes. It highlights the differing trends between the two assets, with ETH showing higher volatility in maximum profit.

### 5.2.1 Fractional Trading

| Asset | Fractional Trade (%) | Average Profit (%) | Maximum Profit (%) | Annual Profit (%) | Average Sharpe | Maximum Sharpe | Average Trades |
|---|---|---|---|---|---|---|---|
| BTC | 10 | 4.39 | 24.76 | 52.66 | 0.025 | 0.105 | 460 |
| | 20 | 4.55 | 23.63 | 54.59 | 0.026 | 0.100 | 500 |
| | 25 | 4.30 | 31.46 | 51.65 | 0.025 | 0.123 | 456 |
| | 33 | 4.36 | 29.05 | 52.32 | 0.023 | 0.113 | 447 |
| | 50 | 4.18 | 30.59 | 50.13 | 0.023 | 0.122 | 467 |
| ETH | 10 | 3.77 | 26.82 | 45.24 | 0.017 | 0.098 | 486 |
| | 20 | 4.07 | 33.99 | 48.84 | 0.016 | 0.113 | 429 |
| | 25 | 4.31 | 33.96 | 51.74 | 0.018 | 0.113 | 424 |
| | 33 | 4.13 | 37.06 | 49.56 | 0.017 | 0.123 | 494 |
| | 50 | 4.02 | 35.12 | 48.24 | 0.016 | 0.137 | 469 |

**Table 5.4** – A2C - Fractional Trading % Comparison

**Figure 5.14** – BTC - A2C - Fractional Trading

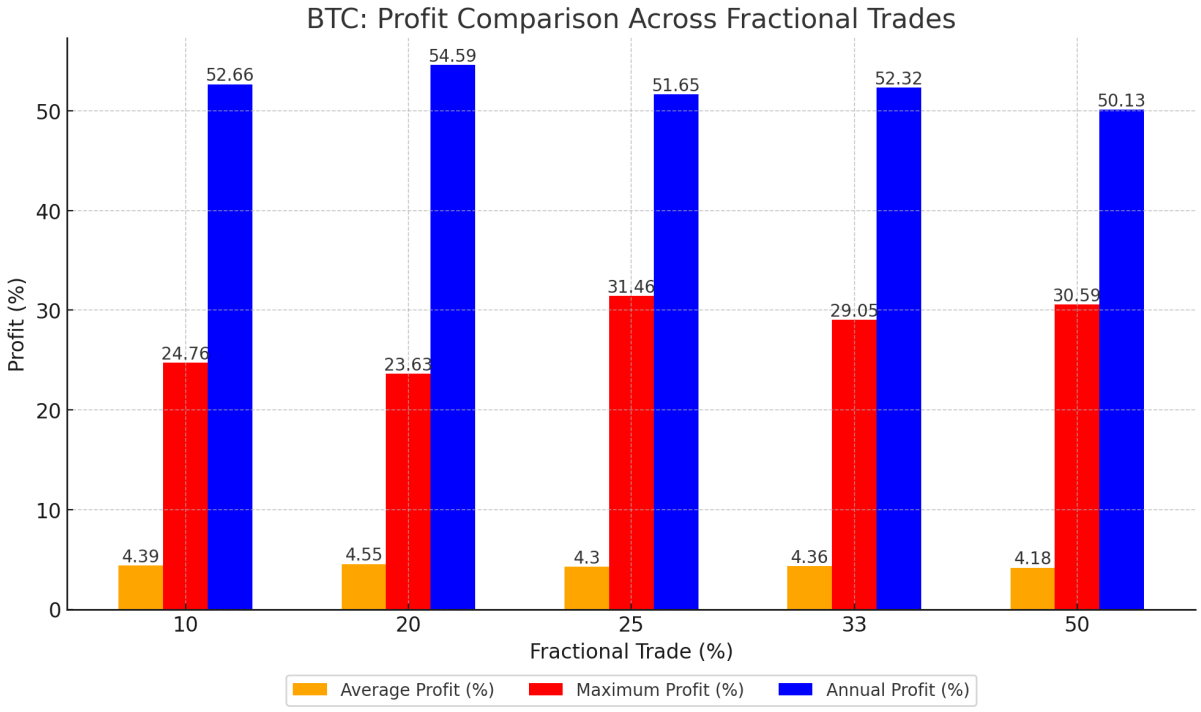


**Figure 5.15** – ETH - A2C - Fractional Trading

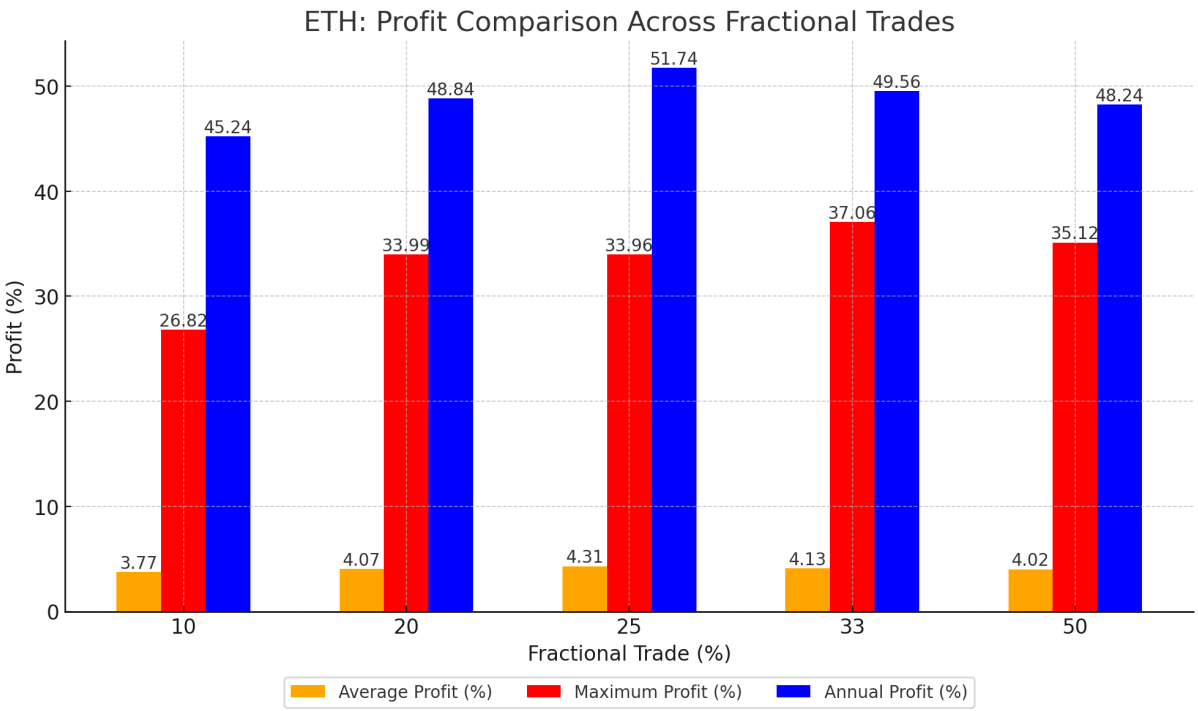
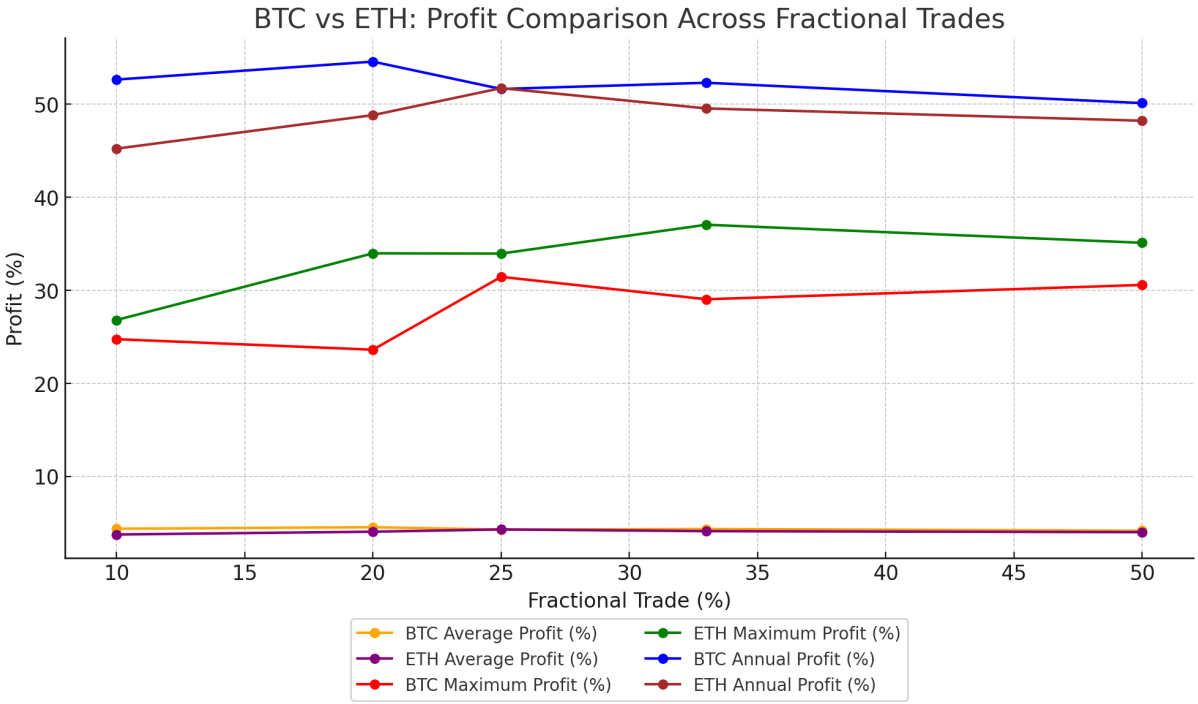



**Figure 5.16** – A2C - Fractional Trading: BTC vs ETH



The table 5.4, fig. 5.14, and fig. 5.15 provides performance data of the A2C model trained on BTC and ETH across different Fractional Trade (%) levels, showing metrics such as Average Profit (%), Maximum Profit (%), Annual Profit (%), Average Sharpe Ratio, Maximum Sharpe Ratio, and Average Trades.

In fig. 5.14, it can be seen that for BTC, the Average Profit remains fairly consistent across the fractional trade levels, peaking at 4.55% at a 20% trade fraction. The Maximum Profit fluctuates more, with a peak of 31.46% at 25%, while Annual Profit reaches its highest at 54.59% at 20%. Average Sharpe Ratios are stable, staying around 0.023 to 0.026. The number of trades remains fairly constant, with a slight increase at 20%.

In fig. 5.15, it can be seen that for ETH, Average Profit shows a modest upward trend, reaching 4.31% at 25% fractional trade, while Maximum Profit peaks at 37.06% at 33%. Annual Profit follows a similar pattern, with the highest value of 51.74% occurring at 25%. The Sharpe Ratios for ETH are slightly lower than BTC overall, but they also show consistency across fractional trade levels. Average Trades fluctuate a bit more, particularly at 33%, where there's a peak of 494 trades.

The fig. 5.16 compares the performance of A2C trained on BTC and ETH for the same metrics (average profit, maximum profit, and annual profit) across fractional trade levels. The trends reveal the differences between the two assets, with ETH showing greater volatility in maximum profit.

## 5.3 Proximal Policy Optimisation

| Asset | Episodes | Average Profit (%) | Maximum Profit (%) | Annual Profit (%) | Average Sharpe | Maximum Sharpe | Average Trades |
|---|---|---|---|---|---|---|---|
| BTC | 100 | 3.59 | 25.75 | 43.04 | 0.023 | 0.102 | 268 |
| | 200 | 6.92 | 48.77 | 83.02 | 0.023 | 0.105 | 606 |
| | 500 | 8.35 | 51.50 | 100.18 | 0.024 | 0.107 | 675 |
| ETH | 100 | 2.97 | 31.88 | 35.69 | 0.014 | 0.117 | 464 |
| | 200 | 3.95 | 48.18 | 47.51 | 0.015 | 0.099 | 544 |
| | 500 | 7.10 | 62.11 | 85.25 | 0.016 | 0.119 | 689 |

**Table 5.5** – PPO trained with OHLCV and Blockchain data - Fractional Trading 25%

**Figure 5.17** – BTC - PPO

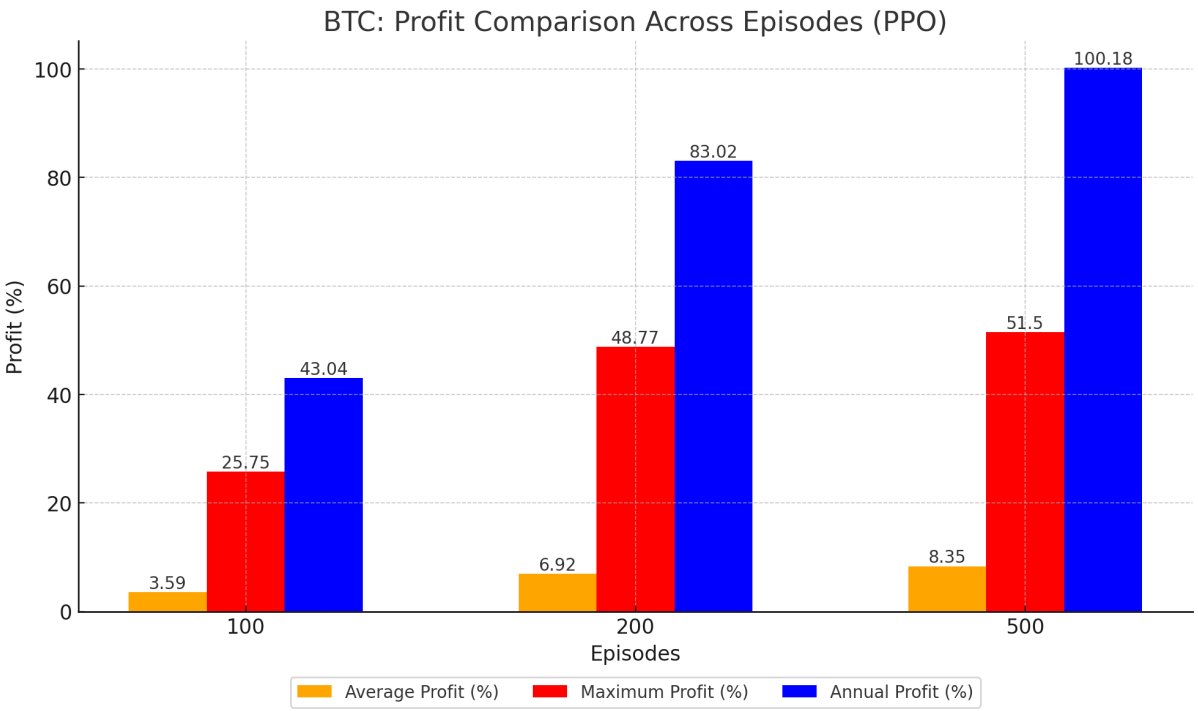



**Figure 5.18** – BTC - PPO - Sharpe ratio

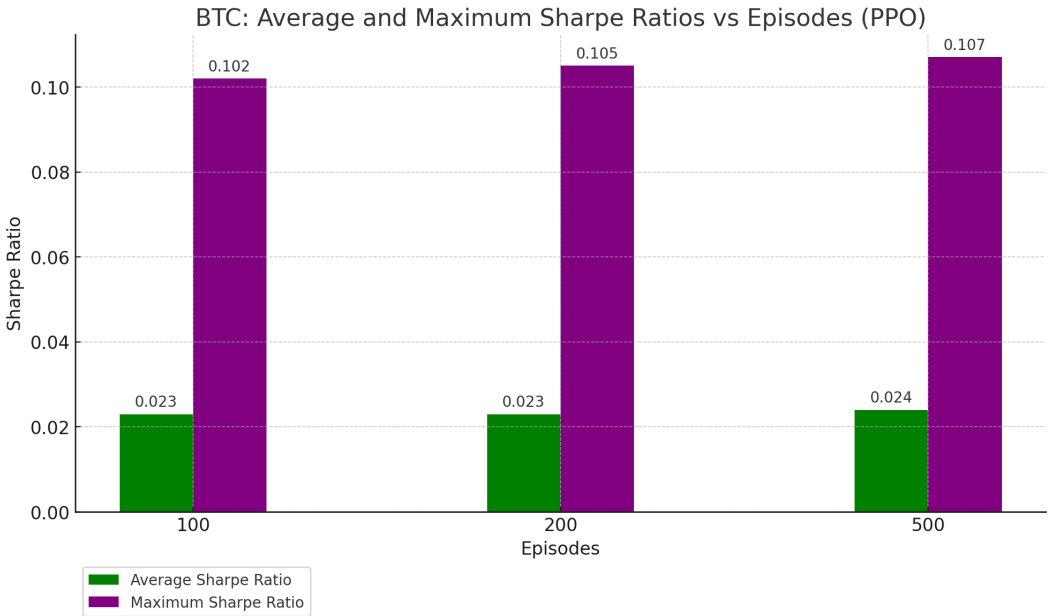


**Figure 5.19** – ETH - PPO

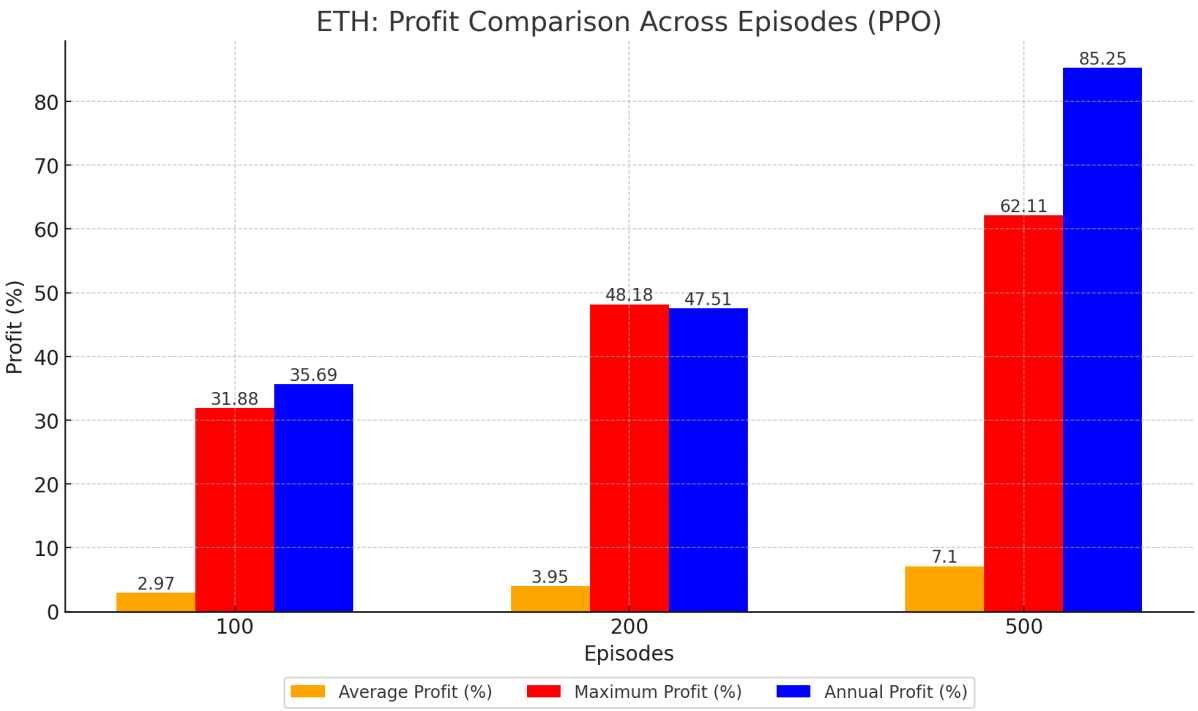



**Figure 5.20** – ETH - PPO - Sharpe ratio

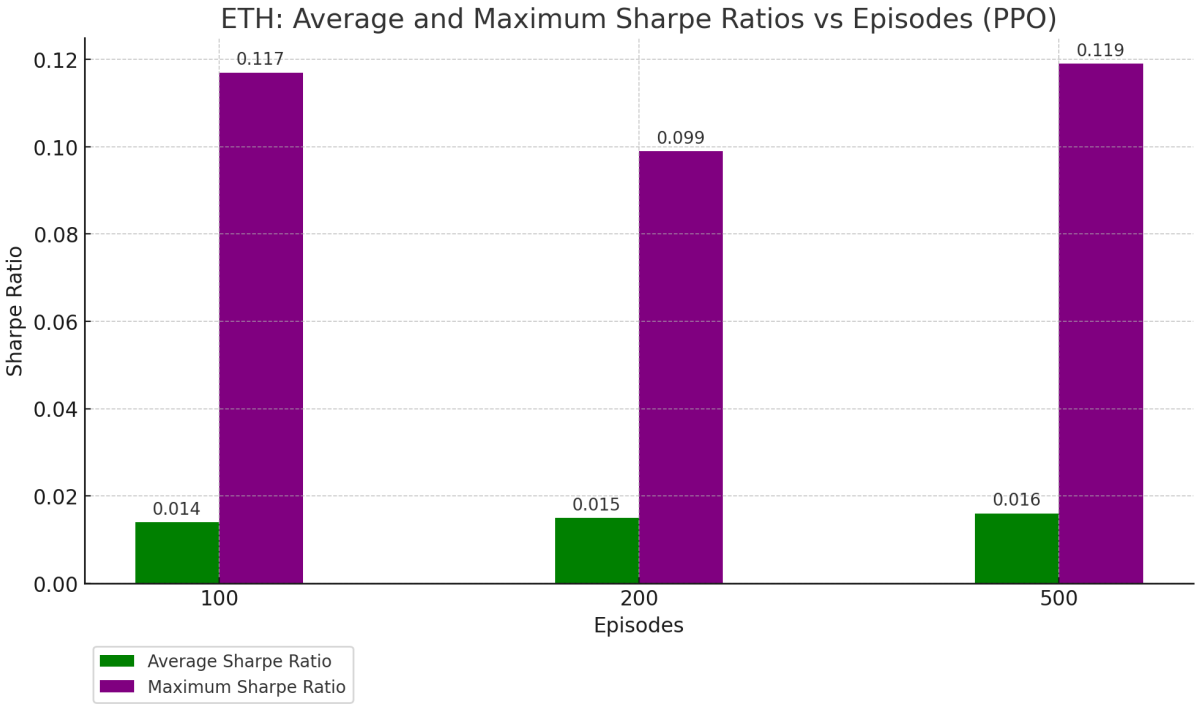
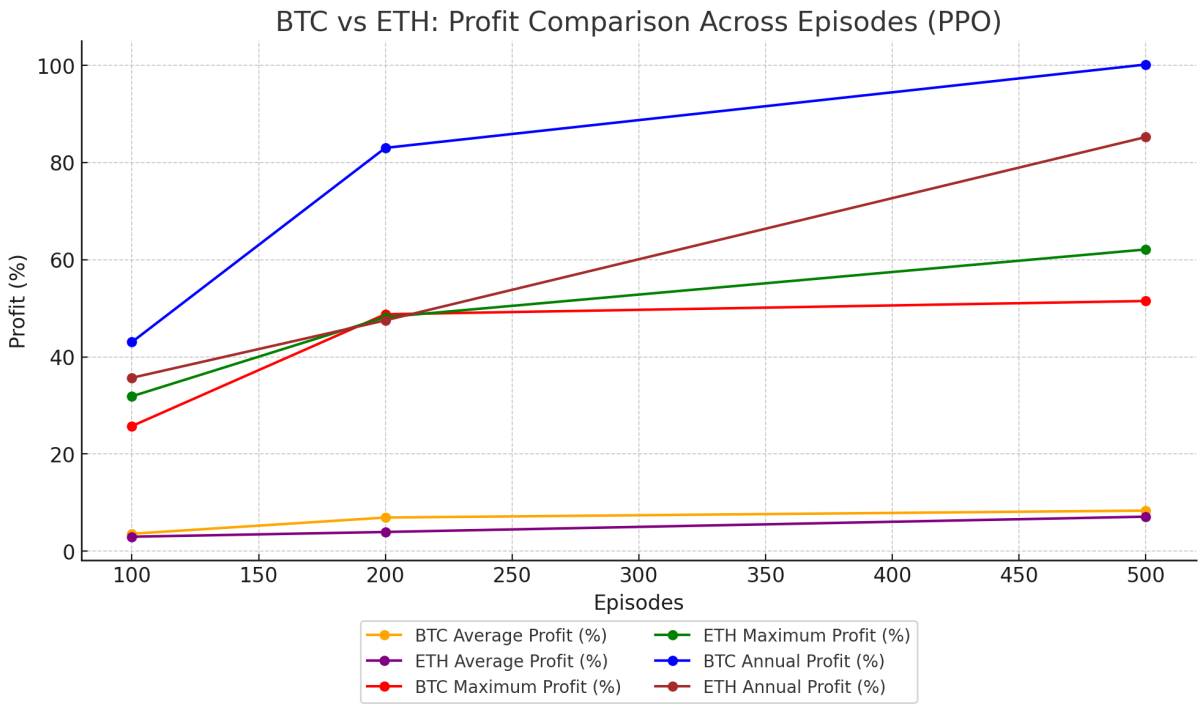


**Figure 5.21** – PPO Performance: BTC vs ETH




The table 5.5, fig. 5.17, fig. 5.18, fig. 5.19, and fig. 5.20 compares performance metrics for the PPO (Proximal Policy Optimisation) model trained on BTC and ETH data across 100, 200, and 500 episodes, with key metrics such as Average Profit (%), Maximum Profit (%), Annual Profit (%), Sharpe Ratios, and Average Trades.

In fig. 5.17, it can be seen that for BTC, there is a significant increase in both Average Profit and Maximum Profit as the number of episodes increases. The Average Profit grows from 3.59% after 100 episodes to 8.35% after 500 episodes, while Maximum Profit rises from 25.75% to 51.50%. Similarly, Annual Profit improves drastically, peaking at 100.18% after 500 episodes. In fig. 5.18 it can be seen that the Sharpe Ratios are relatively stable, with slight improvements as episodes increase. Average Trades also increase significantly, growing from 268 trades after 100 episodes to 675 trades after 500 episodes.

In fig. 5.19, it can be seen that for ETH, the results show similar improvements, though with more volatility. Average Profit grows from 2.97% after 100 episodes to 7.10% after 500 episodes, while Maximum Profit shows a sharp rise from 31.88% to 62.11%. Annual Profit increases to 85.25% after 500 episodes. In fig. 5.20 it can be seen that the Sharpe Ratios for ETH are lower compared to BTC, but there is a slight improvement over time. The number of trades increases from 464 to 689 trades as episodes increase. The line chart in fig. 5.21 compares BTC and ETH across the same metrics over the episodes. The trends highlight the significant performance improvements for both assets, with BTC showing higher annual profits, while ETH demonstrates higher volatility in maximum profit.

### 5.3.1 Fractional Trading

| Asset | Fractional Trade (%) | Average Profit (%) | Maximum Profit (%) | Annual Profit (%) | Average Sharpe | Maximum Sharpe | Average Trades |
|---|---|---|---|---|---|---|---|
| BTC | 10 | 2.95 | 21.57 | 35.39 | 0.022 | 0.100 | 536 |
| | 20 | 4.49 | 23.39 | 53.94 | 0.022 | 0.099 | 595 |
| | 25 | 3.59 | 25.75 | 43.04 | 0.023 | 0.102 | 268 |
| | 33 | 6.22 | 42.18 | 74.67 | 0.024 | 0.125 | 342 |
| | 50 | 5.33 | 36.06 | 64.03 | 0.024 | 0.128 | 401 |
| ETH | 10 | 6.95 | 60.30 | 83.40 | 0.018 | 0.102 | 617 |
| | 20 | 3.40 | 25.11 | 40.81 | 0.016 | 0.110 | 436 |
| | 25 | 2.97 | 31.88 | 35.69 | 0.014 | 0.117 | 464 |
| | 33 | 6.22 | 59.59 | 74.70 | 0.017 | 0.116 | 540 |
| | 50 | 5.99 | 62.78 | 71.68 | 0.017 | 0.100 | 512 |

**Table 5.6** – PPO - Fractional Trading % Comparison

**Figure 5.22** – BTC - PPO - Fractional Trading

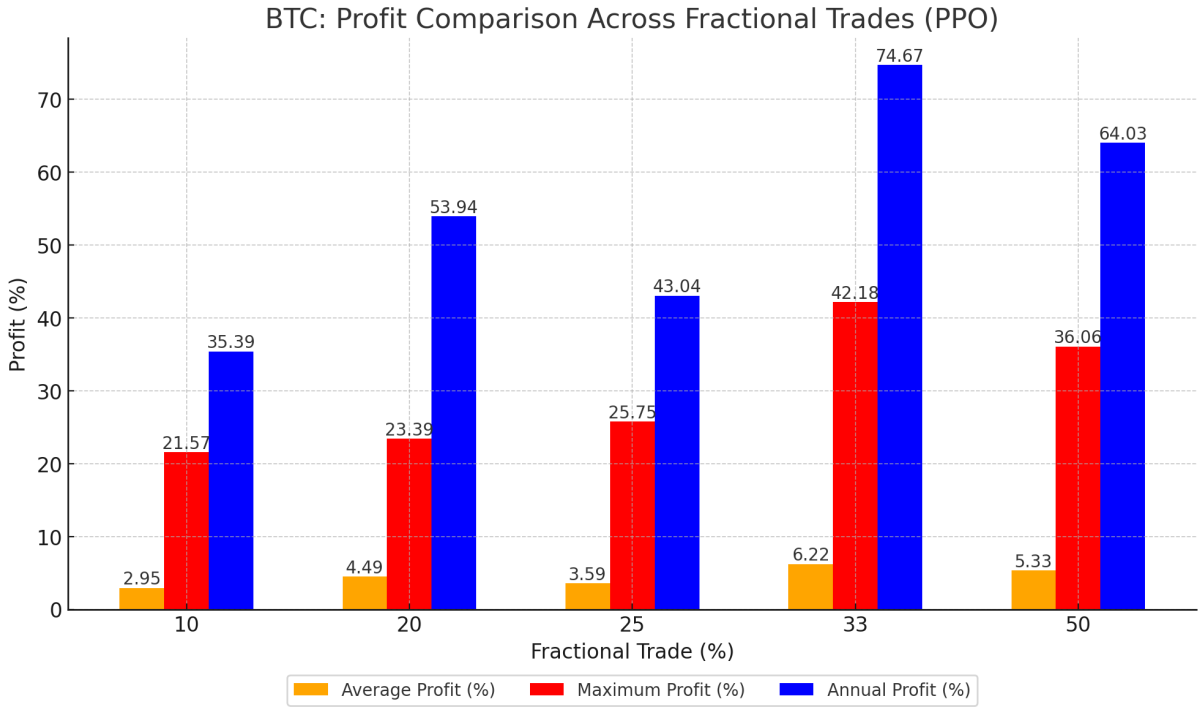


**Figure 5.23** – ETH - PPO - Fractional Trading

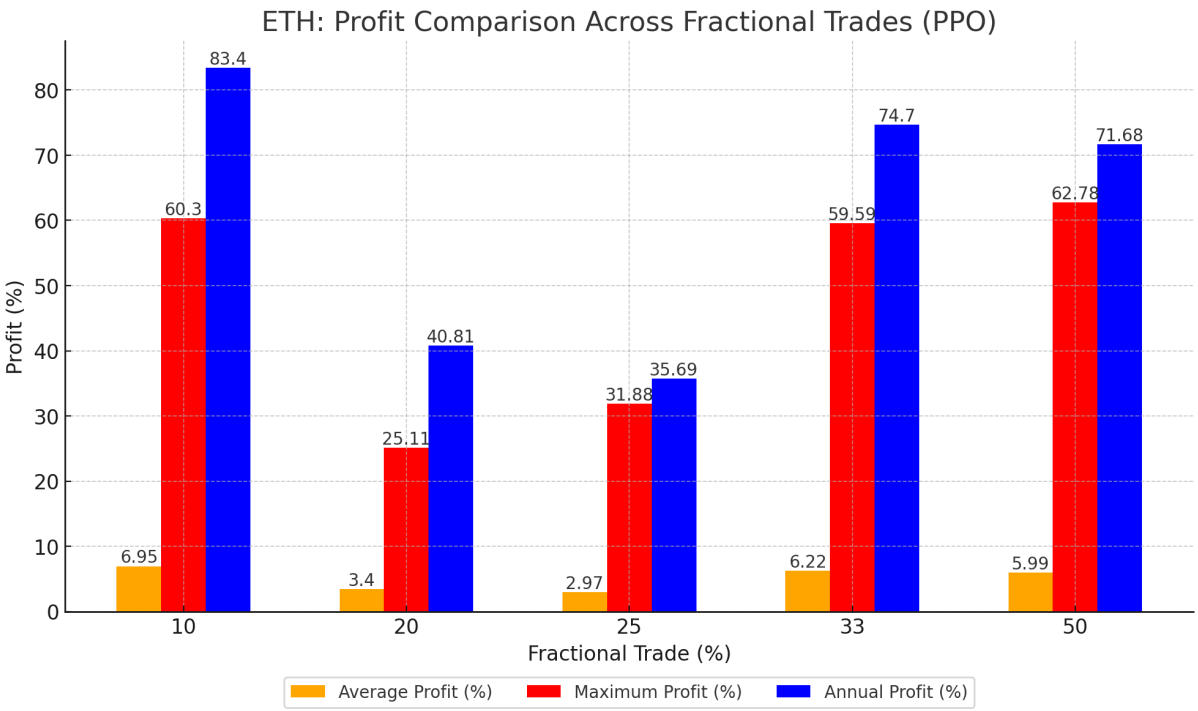
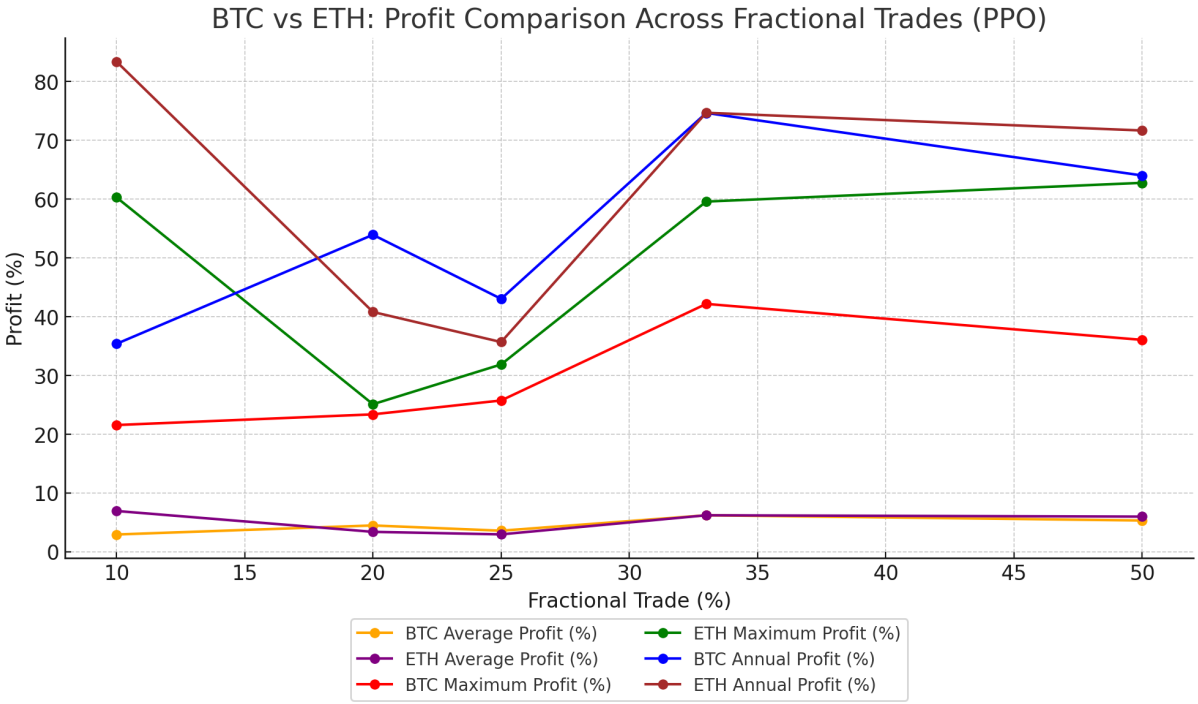



**Figure 5.24** – PPO - Fractional Trading: BTC vs ETHg



The table 5.6, fig. 5.22, fig. 5.23, and fig. 5.24 compares performance metrics for the PPO (Proximal Policy Optimisation) trained on BTC and ETH across different Fractional Trade (%) levels, showing metrics such as Average Profit (%), Maximum Profit (%), Annual Profit (%), Average Sharpe Ratio, Maximum Sharpe Ratio, and Average Trades.

In fig. 5.22, it can be seen that for BTC, the Average Profit increases with higher fractional trades, reaching a peak of 6.22% at 33% trade. Maximum Profit follows a similar trend, peaking at 42.18% at the 33% level. Annual Profit increases significantly, with a high of 74.67% at 33%, and 64.03% at 50%. The Sharpe Ratios are relatively stable, with slight improvements at higher fractional trade levels, reaching a maximum of 0.128 at 50%. The Average Trades fluctuate, with a sharp decrease at 25%, and then gradually increasing with higher fractional trades.

In fig. 5.23, it can be seen that for ETH, the Average Profit is more volatile, peaking at 6.95% for 10% fractional trades and then decreasing before climbing again. Maximum Profit is also volatile, with a high of 62.78% at 50%. Annual Profit follows a similar pattern, with 83.40% at 10% and 74.70% at 33%. Sharpe Ratios remain low overall but show some improvement at higher fractional trade levels. Average Trades are higher for ETH compared to BTC, with the highest value of 617 trades at 10% fractional trades.

The line chart in fig. 5.24 compares BTC and ETH across the same metrics (average profit, maximum profit, and annual profit) over different fractional trade levels. It highlights the trends in profitability for both assets, with ETH showing more volatility but also higher peaks in maximum and annual profits at certain levels.

## 5.4 Deep Deterministic Policy Gradient

| Asset | Episodes | Average Profit (%) | Maximum Profit (%) | Annual Profit (%) | Average Sharpe | Maximum Sharpe | Average Trades |
|---|---|---|---|---|---|---|---|
| BTC | 100 | 8.99 | 51.05 | 107.95 | 0.025 | 0.097 | 26 |
| | 200 | 7.42 | 51.02 | 89.00 | 0.022 | 0.104 | 127 |
| | 500 | 8.37 | 52.73 | 100.41 | 0.023 | 0.102 | 72 |
| ETH | 100 | 6.16 | 60.39 | 73.91 | 0.014 | 0.102 | 113 |
| | 200 | 7.20 | 62.69 | 86.46 | 0.015 | 0.099 | 49 |
| | 500 | 7.62 | 63.86 | 91.45 | 0.016 | 0.102 | 23 |

**Table 5.7** – DDPG trained with OHLCV and Blockchain data - Optimized Fractional Trading

**Figure 5.25** – BTC - DDPG

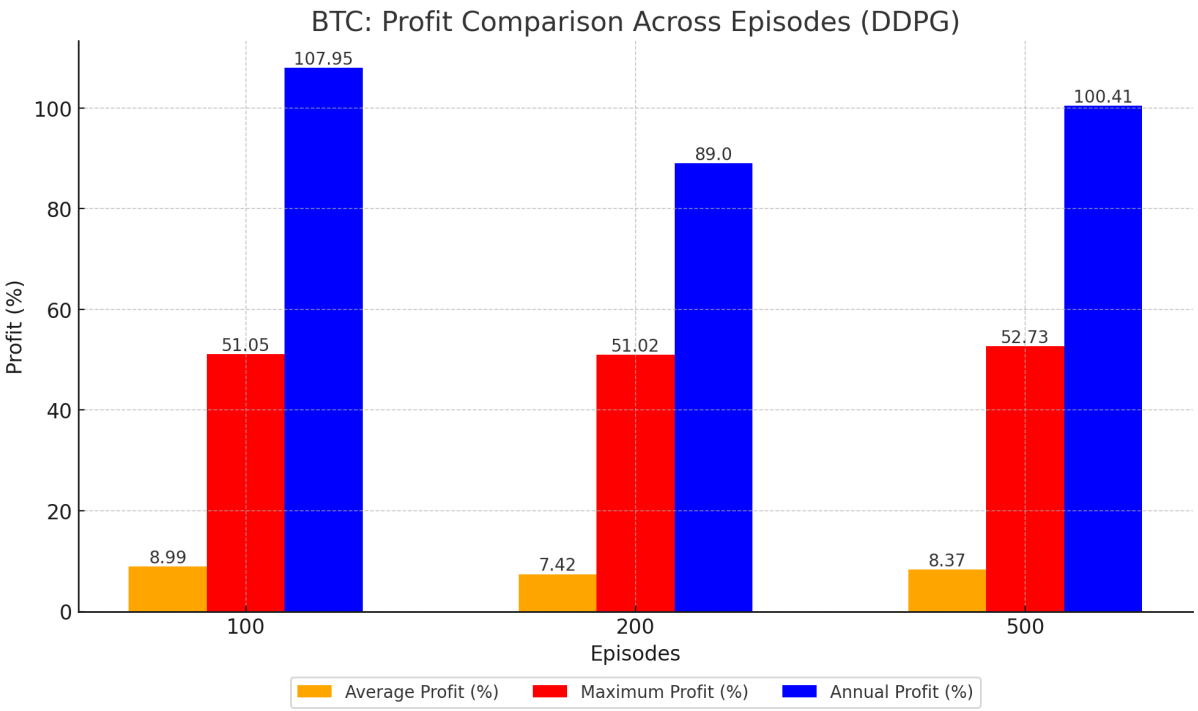



**Figure 5.26** – BTC - DDPG - Sharpe ratio

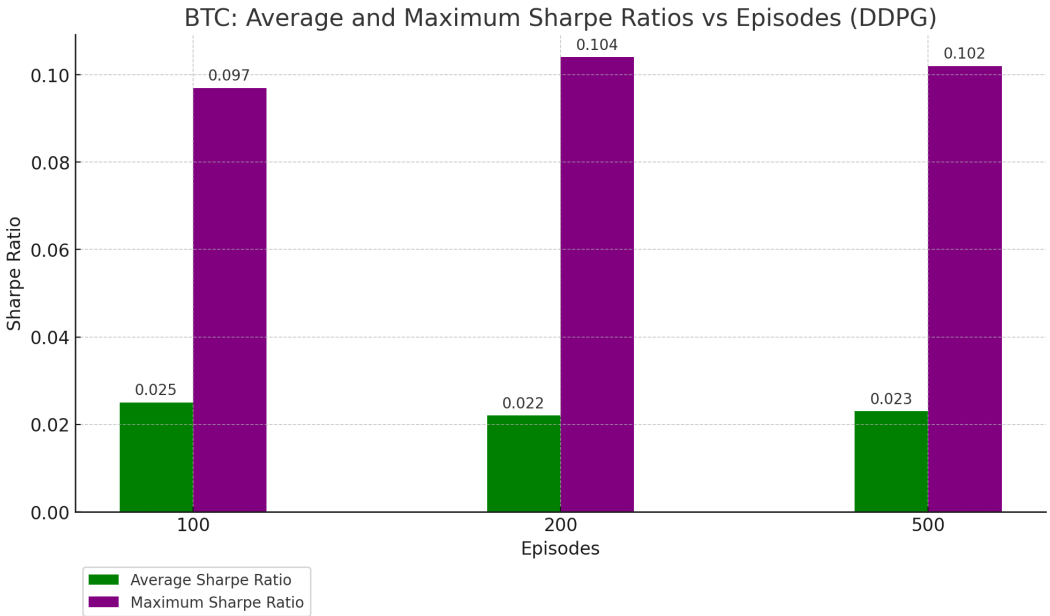


**Figure 5.27** – ETH - DDPG

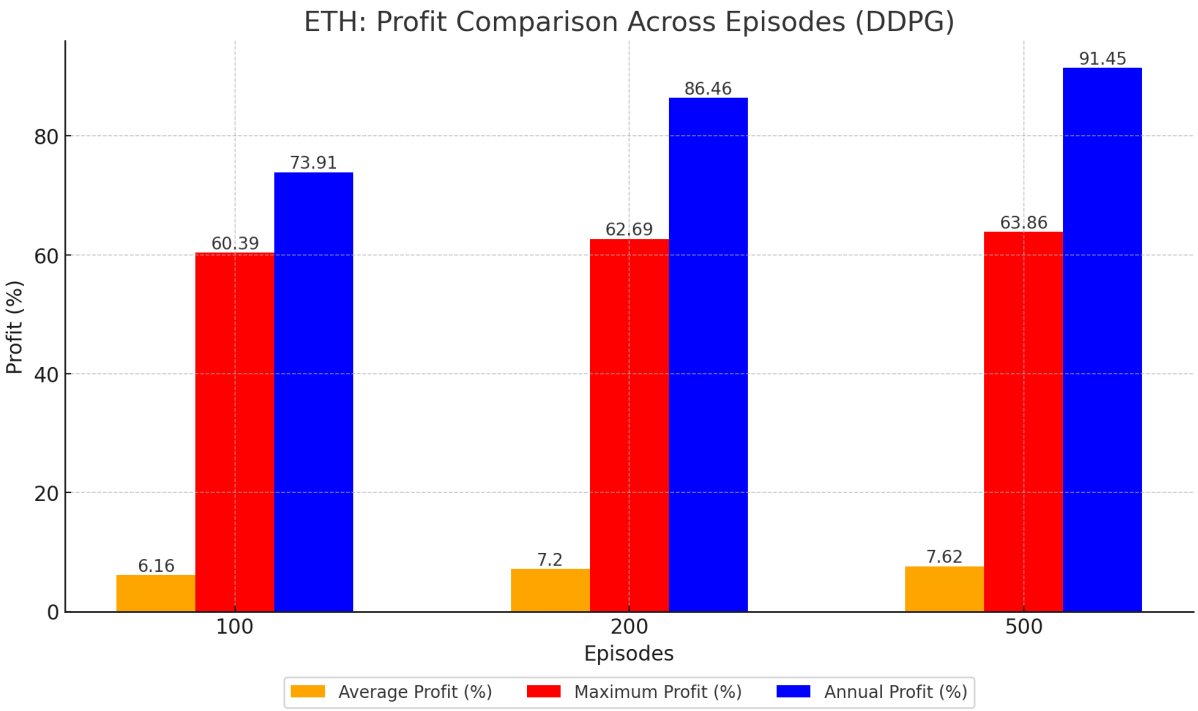



**Figure 5.28** – ETH - DDPG - Sharpe ratio

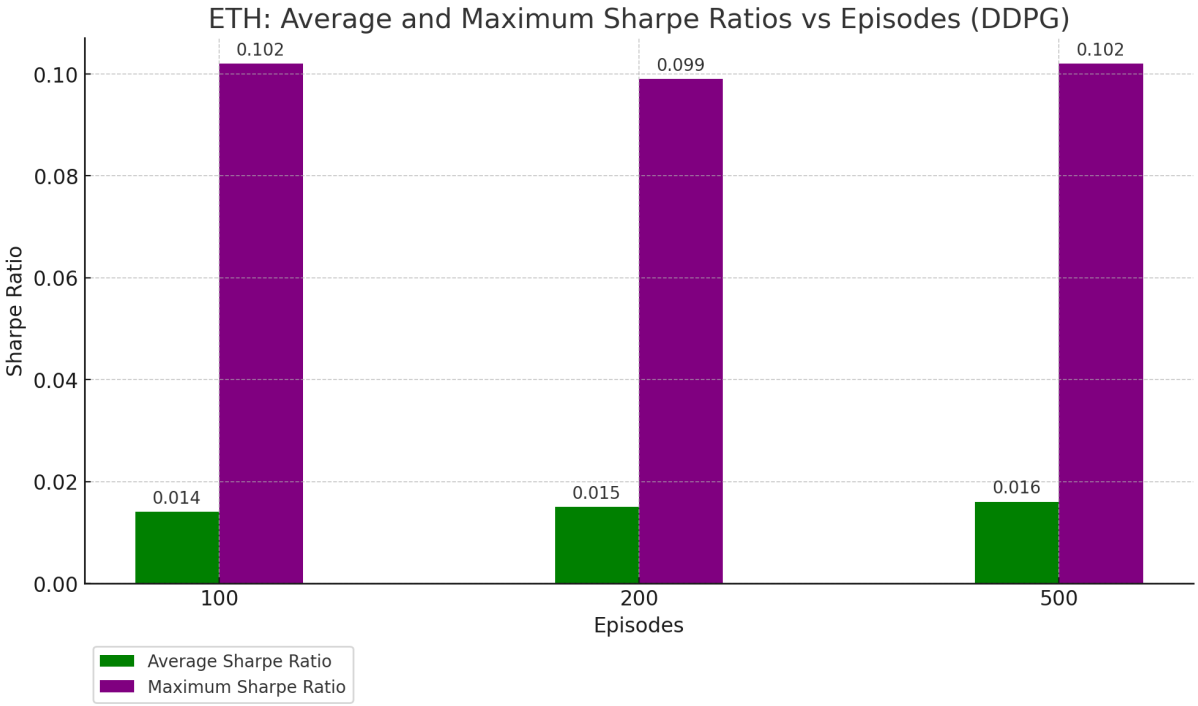
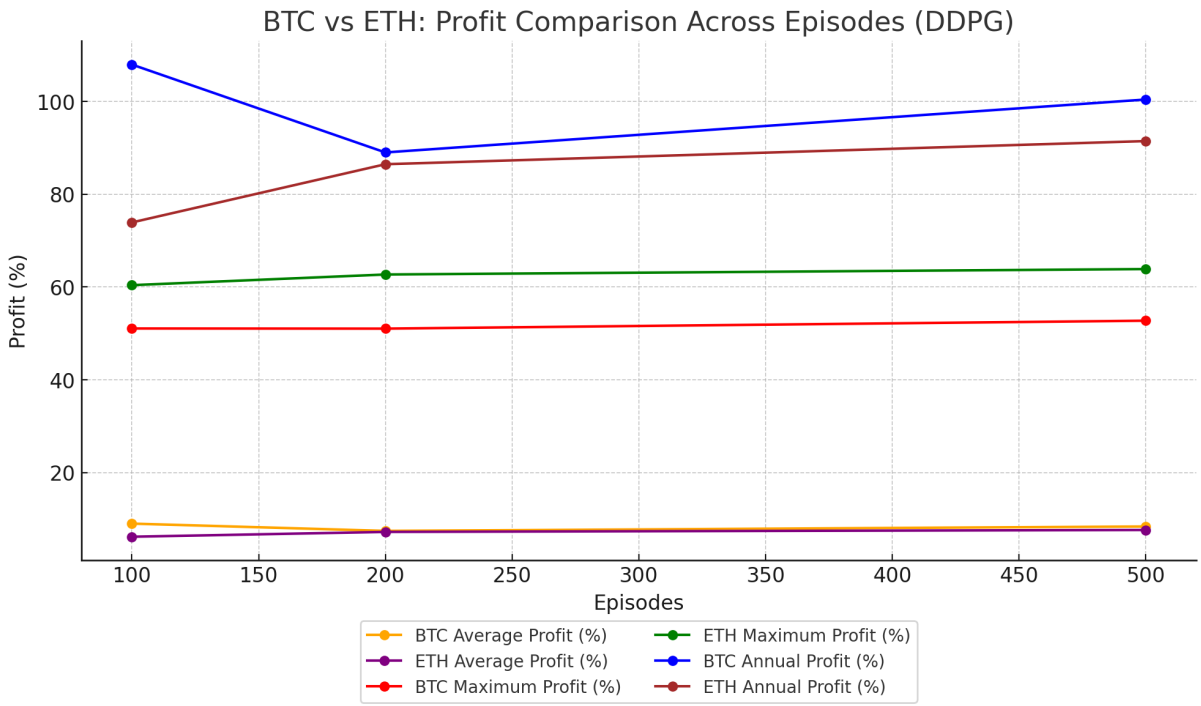


**Figure 5.29** – DDPG - Performance: BTC vs ETH




The table 5.7, fig. 5.25, fig. 5.26, fig. 5.27, and fig. 5.28 compares performance metrics for the DDPG (Deep Deterministic Policy Gradient) model trained on BTC and ETH data across 100, 200, and 500 episodes, with key metrics such as Average Profit (%), Maximum Profit (%), Annual Profit (%), Sharpe Ratios, and Average Trades. The DDPG model in this study has been implemented with optimised fractional trading i.e it automatically selects the optimum fraction of the asset along with the best possible trading decision at any given moment.

In fig. 5.25, it can be seen that for BTC, the Average Profit remains fairly consistent across episodes, peaking at 8.99% after 100 episodes, and slightly decreasing to 8.37% after 500 episodes. Maximum Profit is relatively stable, reaching up to 52.73% after 500 episodes. Annual Profit peaks at 107.95% after 100 episodes but stays above 89% across all episodes. In fig. 5.26, it can be seen that the Sharpe Ratios remain stable, with a slight variation, and Average Trades fluctuate significantly, peaking at 127 trades after 200 episodes but dropping to 72 trades after 500 episodes.

In fig. 5.27, it can be seen that for ETH, the Average Profit increases steadily, reaching 7.62% after 500 episodes. Maximum Profit also improves, reaching 63.86% after 500 episodes. Annual Profit follows the same upward trend, peaking at 91.45%. In fig. 5.28, it can be seen that the Sharpe Ratios are lower than BTC but show slight improvement over time. Average Trades drop significantly, from 113 trades after 100 episodes to just 23 after 500 episodes, suggesting a more refined trading strategy.

The line chart in fig. 5.29 compares BTC and ETH for the same metrics (average profit, maximum profit, and annual profit) over different episodes. It shows that while BTC has higher average annual

profits, ETH has consistently high maximum profits.

### 5.4.1 DDPG trained using OHLCV Data and Technical Indicators

| Asset | Data Type | Average Profit (%) | Maximum Profit (%) | Annual Profit (%) | Average Sharpe | Maximum Sharpe | Average Trades |
|---|---|---|---|---|---|---|---|
| BTC | OHLCV | 3.83 | 40.00 | 45.95 | 0.013 | 0.135 | 204 |
| | OHLCV + Blockchain | 8.99 | 51.05 | 107.95 | 0.025 | 0.097 | 26 |
| ETH | OHLCV | 5.24 | 50.94 | 62.88 | 0.013 | 0.108 | 24 |
| | OHLCV + Blockchain | 7.62 | 63.86 | 91.45 | 0.016 | 0.102 | 23 |

**Table 5.8** – Comparison between DPPG (Optimized Fractional Trading) trained with OHLCV vs OHLCV + Blockchain

**Figure 5.30** – DDPG - Profit Comparison between Data Types: OHLCV vs OHLCV + Blockchain

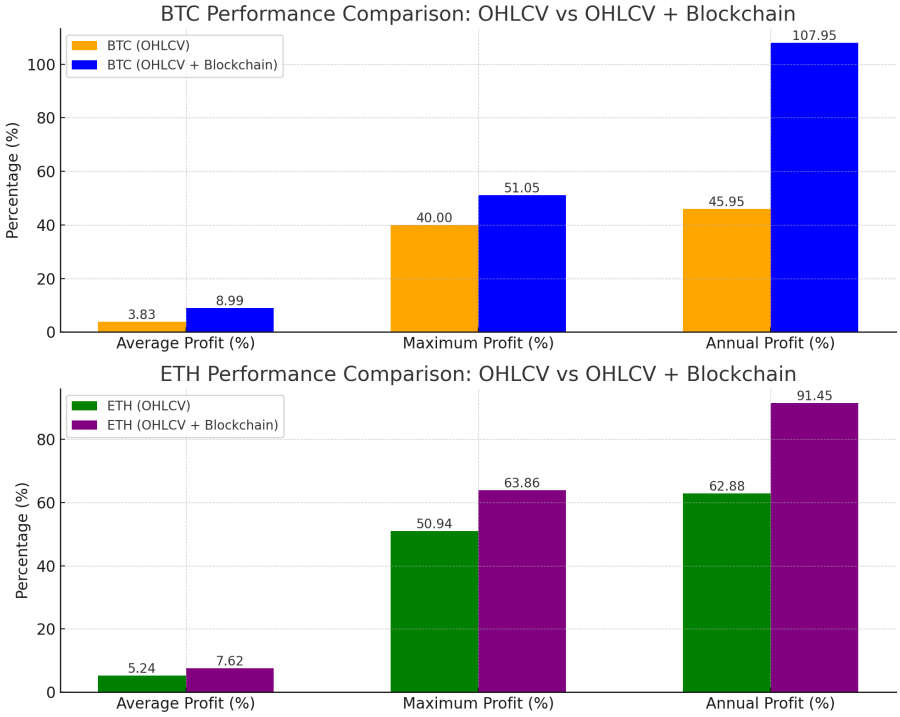


**Figure 5.31** – DDPG - Sharpe Ratio Comparison between Data Types: OHLCV vs OHLCV + Blockchain

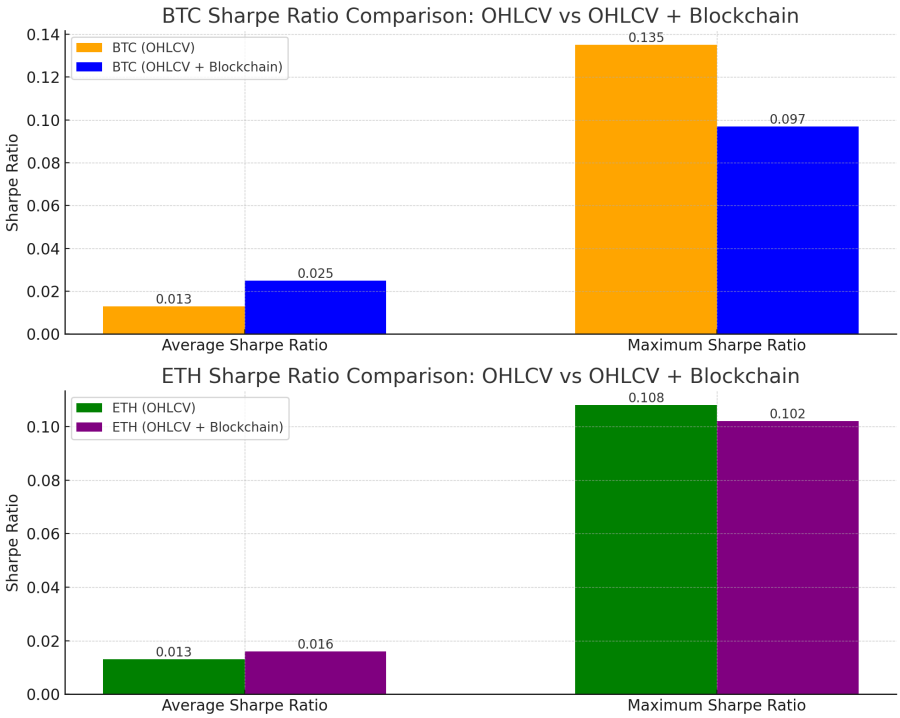



**Figure 5.32** – BTC vs ETH Performance - Profit Comparison between Data Types: OHLCV vs OHLCV + Blockchain

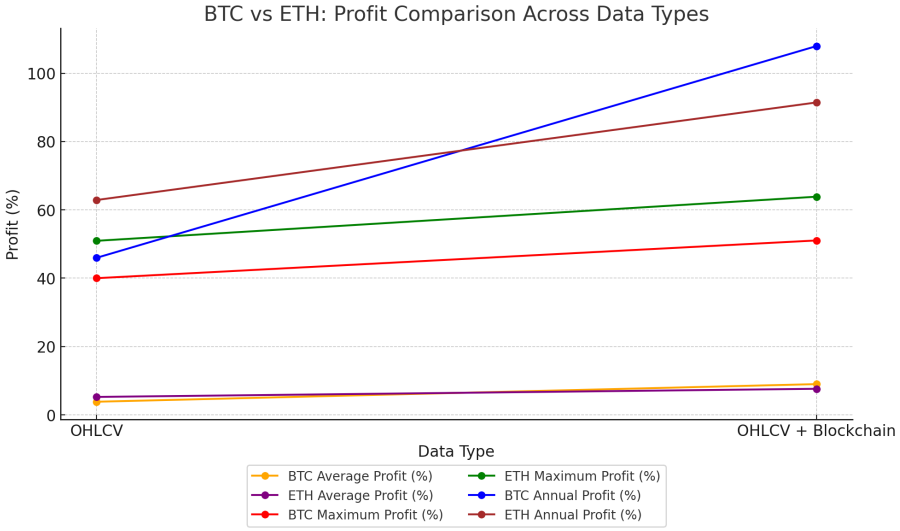
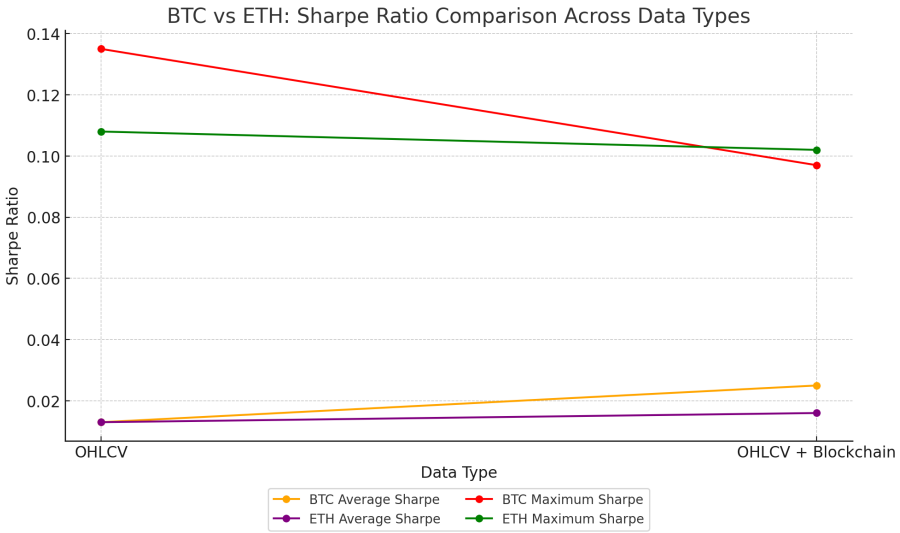


**Figure 5.33** – DDPG - Sharpe Ratio Comparison between Data Types: OHLCV vs OHLCV + Blockchain




The table 5.8, fig. 5.30 and fig. 5.31 compares the performance metrics for the DDPG model when it is trained with only OHLCV data and when it is trained with a combination of OHLCV and Blockchain data for both BTC and ETH. In table 5.8 and fig. 5.30, it can be seen that DDPG model trained with BTC - OHLCV and Blockchain data significantly outperforms the DDPG model trained only with OHLCV data in terms of Average Profit, Maximum Profit, and Annual Profit. The average profit jumps from 3.83% to 8.99%, while annual profit more than doubles from 45.95% to 107.95%. However, in fig. 5.31, it can be seen that the Sharpe Ratios (both average and maximum) are relatively modest for both data types, though the OHLCV and Blockchain model has a slightly better risk-adjusted return (higher average Sharpe ratio of 0.025). Another key observation from table 5.8, is the significant difference in Average Trades: 204 for OHLCV and only 26 for OHLCV and Blockchain data, indicating that the model with blockchain data executes far fewer trades while generating much higher returns. This suggests that integrating blockchain data helps the model to identify better-quality trades rather than relying on quantity.

In table 5.8 and fig. 5.30, it can be seen that the DDPG model trained with ETH - OHLCV and Blockchain data also outperforms the DDPG model trained only with OHLCV data in terms of Average Profit, Maximum Profit, and Annual Profit. The average profit increases from 5.24% to 7.62%, and the maximum profit rises from 50.94% to 63.86%. In fig. 5.31, the Sharpe Ratios remain fairly low for both data types, with slight improvements when blockchain data is integrated. The number of Average Trades remains almost the same, with only a slight reduction (from 24 to 23) when using blockchain data.

The line chart in fig. 5.32 compares BTC and ETH across different data types (OHLCV vs. OHLCV + Blockchain) for Average Profit, Maximum Profit, and Annual Profit. The table 5.8 and fig. 5.32 highlight how both assets perform under different data inputs, with BTC showing larger gains in

Annual Profit and ETH maintaining a strong Maximum Profit.

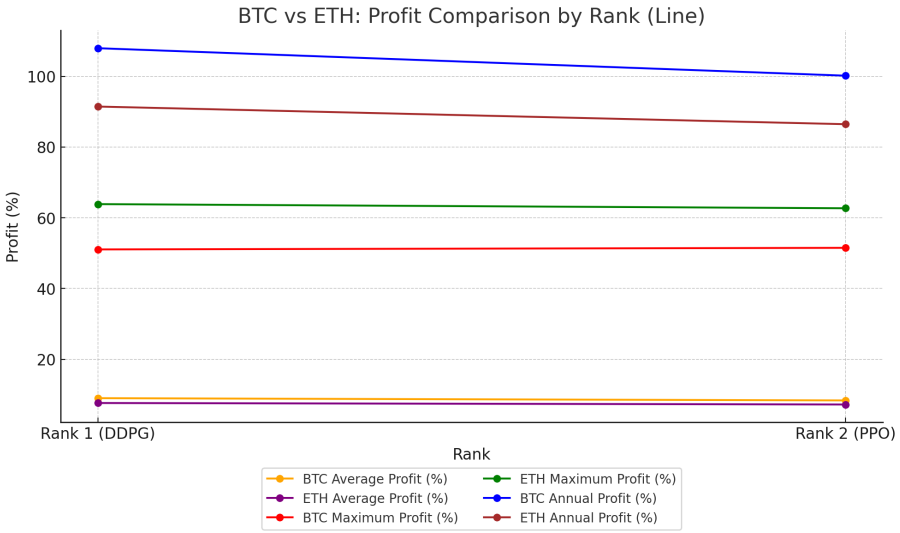
The line chart in fig. 5.33 compares Average Sharpe and Maximum Sharpe Ratios for BTC and ETH across the two data types. This highlights how risk-adjusted returns vary between the data types for both assets.

## 5.5 Best Performance and Comparison

| Asset | Rank | DRL | Average Profit (%) | Maximum Profit (%) | Annual Profit (%) | Average Sharpe | Maximum Sharpe | Average Trades |
|---|---|---|---|---|---|---|---|---|
| BTC | 1 | DDPG | 8.99 | 51.05 | 107.95 | 0.025 | 0.097 | 26 |
| | 2 | PPO | 8.35 | 51.50 | 100.18 | 0.024 | 0.107 | 675 |
| ETH | 1 | DDPG | 7.62 | 63.86 | 91.45 | 0.016 | 0.102 | 23 |
| | 2 | DDPG | 7.20 | 62.69 | 86.46 | 0.015 | 0.099 | 49 |

**Table 5.9** – The best-performing models for BTC and ETH in this study

**Figure 5.34** – BTC vs ETH - Comparison between 2 best models




The table 5.9, and fig. 5.34, compares the performance of BTC and ETH for two of the best performing Deep Reinforcement Learning (DRL) models in this study: DDPG (Deep Deterministic Policy Gradient) and PPO (Proximal Policy Optimization). In table 5.9 it can be seen that DDPG outperforms PPO in terms of Average Profit (8.99% vs. 8.35%) and Annual Profit (107.95% vs. 100.18%). However, PPO achieves a slightly higher Maximum Profit (51.50% vs. 51.05%). But when it comes to risk-adjusted returns, PPO has a slightly higher Maximum Sharpe Ratio (0.107 vs. 0.097), though DDPG offers a higher Average Sharpe Ratio (0.025 vs. 0.024). The Average Trades under PPO are significantly higher (675 trades vs. 26 trades with DDPG), suggesting that PPO is a more active

trading strategy compared to DDPG, which adopts a more conservative approach with fewer trades. For ETH, DDPG performs best with an Average Profit of 7.62% and Annual Profit of 91.45%. The Maximum Profit (63.86%) is notably high, reflecting DDPG's capacity for large gains. The Sharpe ratios are lower compared to BTC, but the trading frequency remains very low, indicating a conservative strategy with few trades (23). In the second ETH - DDPG run, the Average Profit (7.2%) and Annual Profit (86.46%) are slightly lower compared to the first DDPG instance. The Maximum Profit also slightly decreases, and the Sharpe Ratios remain stable, reflecting a similar level of risk-adjusted returns. However, this model trades more frequently (49 trades). The line chart in fig. 5.34, shows BTC delivers higher overall Annual Profits, particularly with DDPG. ETH, on the other hand, consistently delivers higher Maximum Profits, suggesting it may capture more significant individual gains but with slightly less consistency in annualized performance compared to BTC.

| No | Asset Class & Data Type | Paper | Best Model | Average Profit (%) | Cumulative /Annual Return (%) | Maximum Return (%) | Maximum Sharpe Ratio | Volatility (%) |
|---|---|---|---|---|---|---|---|---|
| 1 | Stock Market (OHLCV) | Yang et al. 2020 | Ensemble(1) | 1.08 | 13 | NA | 1.30 | 9.7 |
| 2 | Crypocurrency (OHLCV) | S. Wang and Klabjan 2023 | Ensemble(2) | 0.93 | 7.93 | NA | 1.07 | 0.87 |
| 3 | Crypocurrency (OHLCV) | Lucarelli and Borrotti 2019 | Sharpe - D-DQN | 0.05 | 5.81 | 26.14 | NA | 5.26 |
| 4 | Crypocurrency (OHLCV) | Kumlungmak and Vateekul 2023 | MAPPO | 2.36 | 28.26 | NA | 4.64 | 68.25 |
| 5 | Crypocurrency (OHLCV + Blockchain) | This paper | DDPG (Fractional Trading) | 8.99 | 107.95 | 51.05 | 0.097 | 16.06 |

**Table 5.10** – Comparison of best-performing models from different papers

**Figure 5.35** – Profit Comparison with other papers

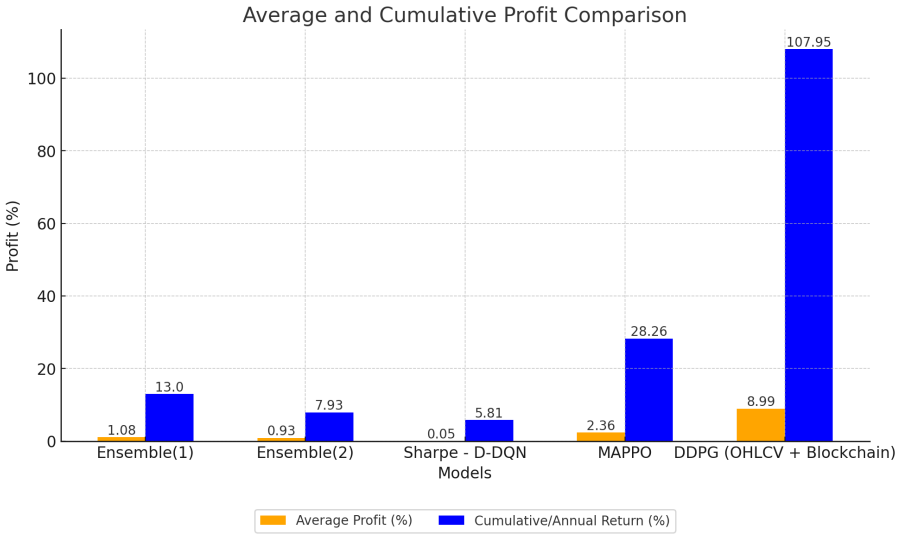
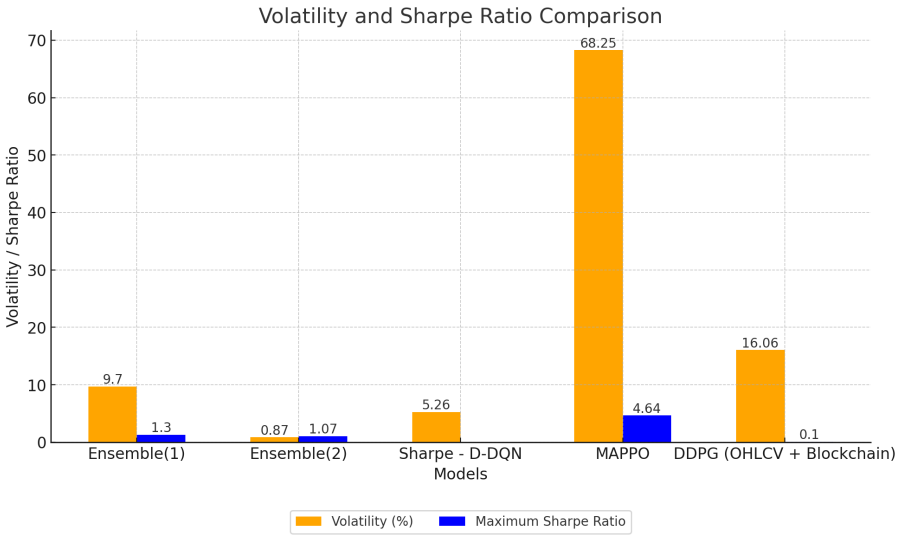


**Figure 5.36** – Volatility and Sharpe Ratio Comparison with other papers




The table 5.10, fig. 5.35, and fig. 5.36 compares the performance of five models across several financial metrics, including Average Profit (%), Cumulative/Annual Return (%), Maximum Return (%), Maximum Sharpe Ratio, and Volatility (%). With the exception of Ensemble(1) which is trained and tested on stock market data, every other model has been tested on cryptocurrency data. The chart in fig. 5.35, shows that DDPG (OHLCV + Blockchain) provides the highest Average Profit of 8.99%, Cumulative Return of 107.95%, and Maximum Return of 51.05%. The numbers and figures suggest that DDPG is highly successful in generating returns, outperforming all other models in these categories. However, the chart in fig. 5.36, shows that its Maximum Sharpe Ratio is low at 0.097, which suggests that while it generates high returns, the risk-adjusted return (as measured by Sharpe Ratio) might not be as favourable as others. Its Volatility is also relatively moderate at 16.06%, indicating that it takes on some level of risk but not excessively high. MAPPO delivers the second-highest Average Profit of 2.36% and Cumulative Return of 28.26%. It excels in risk-adjusted returns with a Maximum Sharpe Ratio of 4.64, which is the highest across all models (Kumlungmak and Vateekul 2023). However, this comes with a high volatility of 68.25%, which implies that although it achieves good Sharpe Ratio performance, the trading strategy is exposed to significant risk. Ensemble (1) has an Average Profit of 1.08% and a Cumulative Return of 13%, which are not particularly high but are stable in comparison. It also has a decent Maximum Sharpe Ratio of 1.30 and Volatility of 9.7%, indicating a good balance between risk and reward (Yang et al. 2020). It is worthwhile to notice that this model was tested exclusively on the stock market i.e relatively stable stocks and bonds. As a result, the risk-reward ratio is inherently a lot better compared to the cryptocurrency market. Ensemble (2), has an Average Profit of 0.93% and a Cumulative Return of 7.93%, and also presents a relatively safe option with the lowest volatility of 0.87%, even though this volatility metric is not clearly defined (S. Wang and Klabjan 2023). However, its Maximum Sharpe Ratio is slightly lower at 1.07, suggesting it has a more conservative approach to risk-taking. Sharpe - D-DQN, while provid-

ing a decent Maximum Return of 26.14%, has the lowest Average Profit of 0.05% and a Cumulative Return of 5.81% (Lucarelli and Borrotti 2019). Its Maximum Sharpe Ratio is not available since Sharpe ratio is used as a reward mechanism in the implementation, making it difficult to assess its risk-adjusted performance directly. However, its Volatility at 5.26% is quite moderate, meaning that the model trades off high returns for more consistent, lower-risk performance.

## 5.6 XAI

### 5.6.1 Integrated Gradient

| Feature | Rank | IG values |
|---|---|---|
| difficulty | 1 | 0.299578 |
| VWAP | 2 | 0.095770 |
| low | 3 | 0.078281 |
| new_addresses | 4 | 0.061004 |
| EMA_7 | 5 | 0.056609 |
| RSI_14 | 6 | 0.047325 |
| SMA_7 | 7 | 0.046645 |
| ATR | 8 | 0.043709 |
| MACD_Hist | 9 | 0.040961 |
| CMF | 10 | 0.036375 |
| block_time | 11 | 0.034053 |
| MACD_Signal | 12 | 0.028709 |
| current_supply | 13 | 0.022076 |
| block_size | 14 | 0.021406 |
| block_height | 15 | 0.019519 |
| SMA_30 | 16 | 0.014936 |
| transaction_count_all_time | 17 | 0.012943 |
| EMA_30 | 18 | 0.012273 |
| large_transaction_count | 19 | 0.008846 |
| open | 20 | 0.008547 |
| average_transaction_value | 21 | 0.006199 |
| MACD | 22 | 0.005741 |
| hashrate | 23 | 0.003936 |
| active_addresses | 24 | 0.003846 |
| volume | 25 | 0.001773 |
| high | 26 | -0.002104 |
| transaction_count | 27 | -0.011800 |
| EMA_14 | 28 | -0.013358 |
| SMA_14 | 29 | -0.016526 |
| unique_addresses_all_time | 30 | -0.016631 |
| id | 31 | -0.020538 |
| close | 32 | -0.027165 |
| zero_balance_addresses_all_time | 33 | -0.039036 |

**Table 5.11** – Integrated Gradient Feature Ranking - BTC

**Figure 5.37** – Integrated Gradient Feature Ranking Barchart- BTC

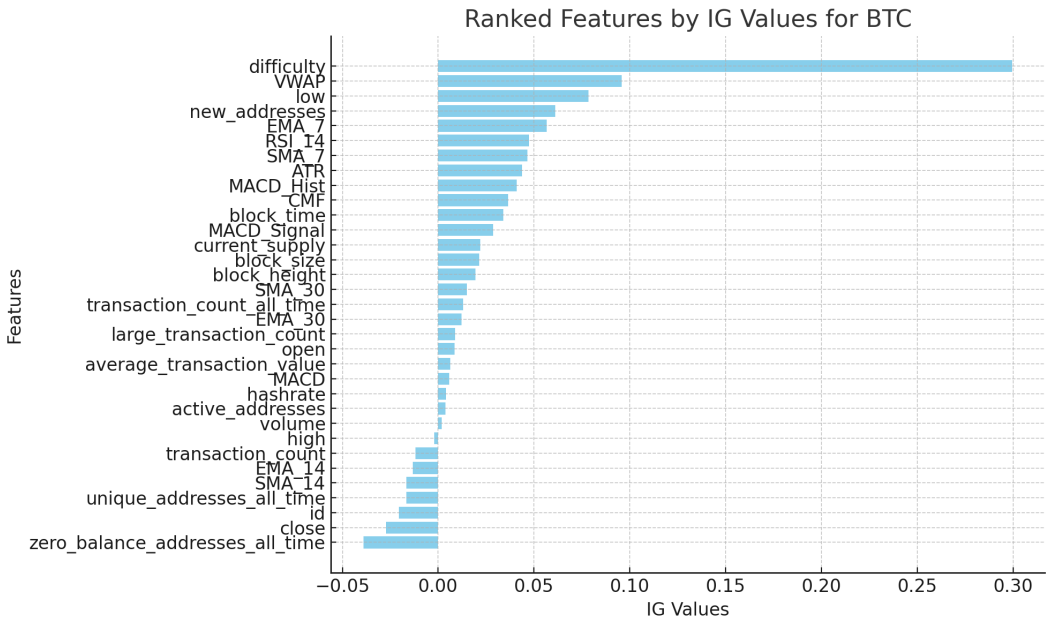
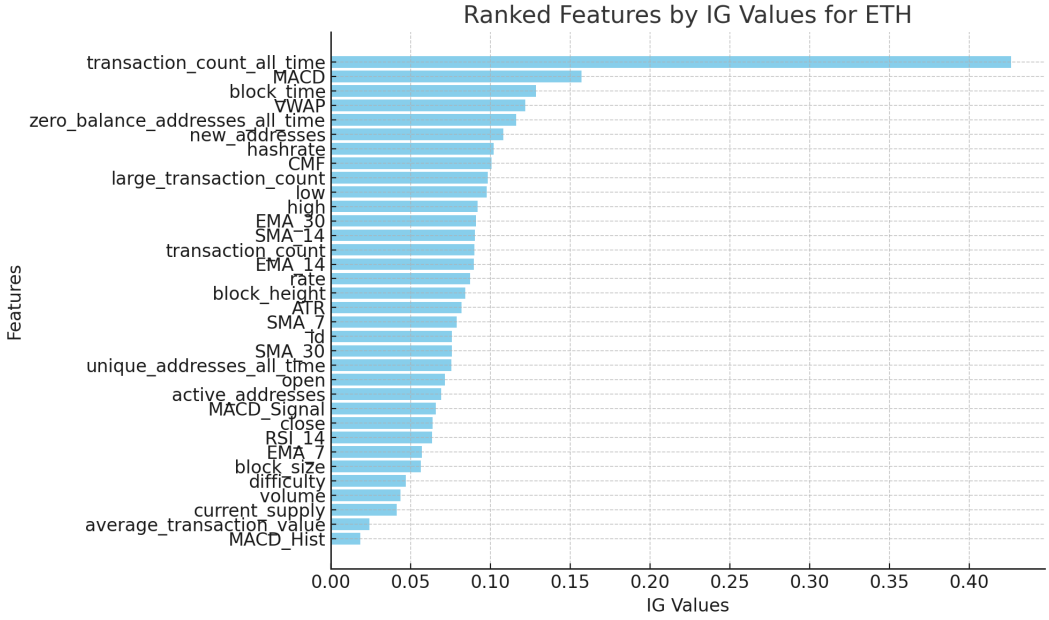



**Figure 5.38** – Integrated Gradient Feature Ranking Barchart- ETH



| Feature | Rank | IG Values |
|---|---|---|
| transaction_count_all_time | 1 | 0.426081 |
| MACD | 2 | 0.157134 |
| block_time | 3 | 0.128626 |
| VWAP | 4 | 0.121895 |
| zero_balance_addresses_all_time | 5 | 0.116045 |
| new_addresses | 6 | 0.108115 |
| hashrate | 7 | 0.101952 |
| CMF | 8 | 0.100623 |
| large_transaction_count | 9 | 0.098190 |
| low | 10 | 0.097585 |
| high | 11 | 0.091810 |
| EMA_30 | 12 | 0.091057 |
| SMA_14 | 13 | 0.090138 |
| transaction_count | 14 | 0.089772 |
| EMA_14 | 15 | 0.089419 |
| rate | 16 | 0.087240 |
| block_height | 17 | 0.084143 |
| ATR | 18 | 0.081760 |
| SMA_7 | 19 | 0.078895 |
| id | 20 | 0.075721 |
| SMA_30 | 21 | 0.075695 |
| unique_addresses_all_time | 22 | 0.075560 |
| open | 23 | 0.071522 |
| active_addresses | 24 | 0.068923 |
| MACD_Signal | 25 | 0.065572 |
| close | 26 | 0.063603 |
| RSI_14 | 27 | 0.063252 |
| EMA_7 | 28 | 0.056993 |
| block_size | 29 | 0.056454 |
| difficulty | 30 | 0.047014 |
| volume | 31 | 0.043619 |
| current_supply | 32 | 0.041306 |
| average_transaction_value | 33 | 0.024234 |
| MACD_Hist | 34 | 0.018496 |

**Table 5.12** – Integrated Gradient Feature Ranking - ETH

The table 5.11 and fig. 5.37 show the ranked feature list for the DDPG model trained on BTC data, calculated based on Integrated Gradient (IG). The fig. 5.37 reveal that difficulty is the most important feature with a significant IG value of 0.299578, indicating its dominant role in predicting outcomes for the BTC model. This is followed by VWAP (Volume Weighted Average Price) with a considerably lower IG value of 0.095770, suggesting that while it has predictive power, it is far less critical than difficulty. The low price ranks third with an IG value of 0.078281, indicating its moderate influence. Features like new_addresses and EMA_7 contribute further with IG values of 0.061004 and 0.056609, respectively. The steep drop in IG values between the first and second features shows a clear distinction in their relevance, highlighting difficulty as a key factor in modelling BTC behavior, with subsequent features contributing diminishing returns. This trend reflects the varying degrees of

importance each feature has in influencing model predictions, with difficulty playing a pivotal role. The table 5.12 and fig. 5.38 show the ranked feature list for the DDPG model trained on ETH data, calculated based on Integrated Gradient (IG). For ETH, the feature transaction_count_all_time dominates the feature ranking with a high IG value of 0.426081, significantly higher than the next-ranked features. This suggests that the total transaction count plays a crucial role in predicting outcomes for ETH. The second most important feature, MACD (Moving Average Convergence Divergence), with an IG value of 0.157134, highlights its moderate influence on ETH price movements but is still less significant compared to transaction count. The block_time and VWAP (Volume Weighted Average Price) features also show moderate relevance with IG values of 0.128626 and 0.121895, respectively. The metric zero_balance_addresses_all_time has an IG value of 0.116045, emphasizing the importance of this on-chain metric in the predictive model. The gap between the first feature and the rest of the ranked features suggests that while several factors contribute to the model, transaction count is especially pivotal in ETH predictions.

### 5.6.2 Saliency Map

**Figure 5.39** – Saliency Map Feature Ranking Barchart- BTC

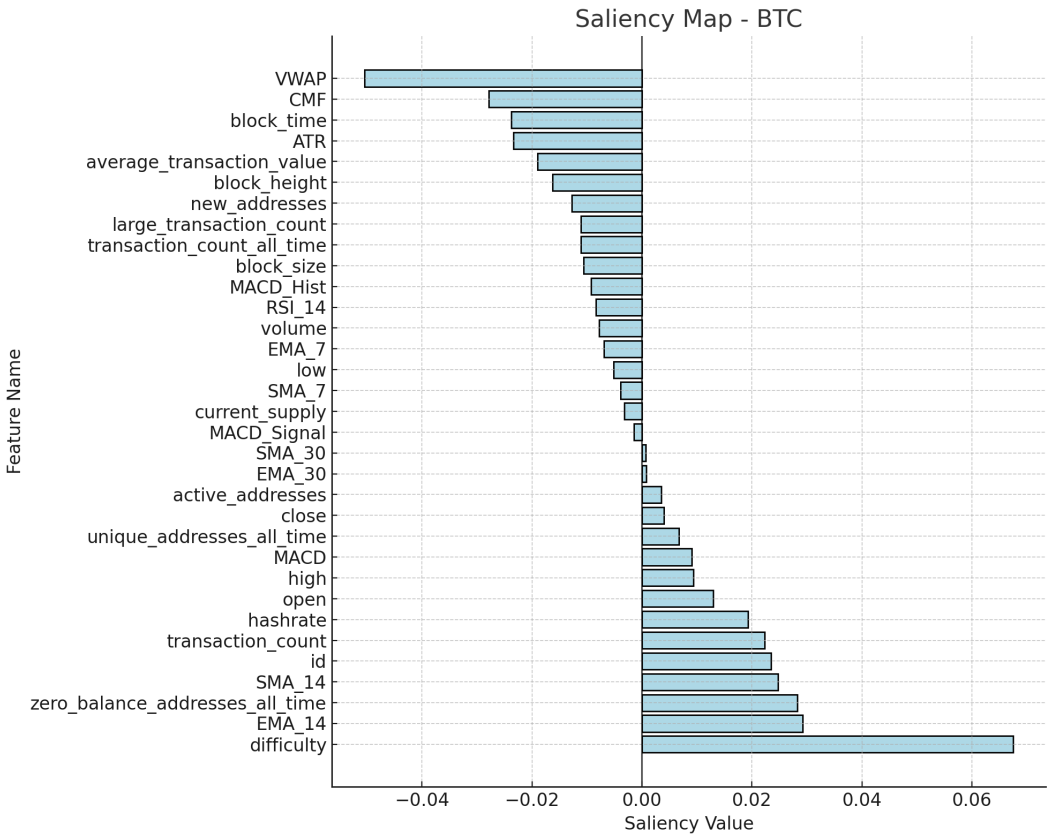
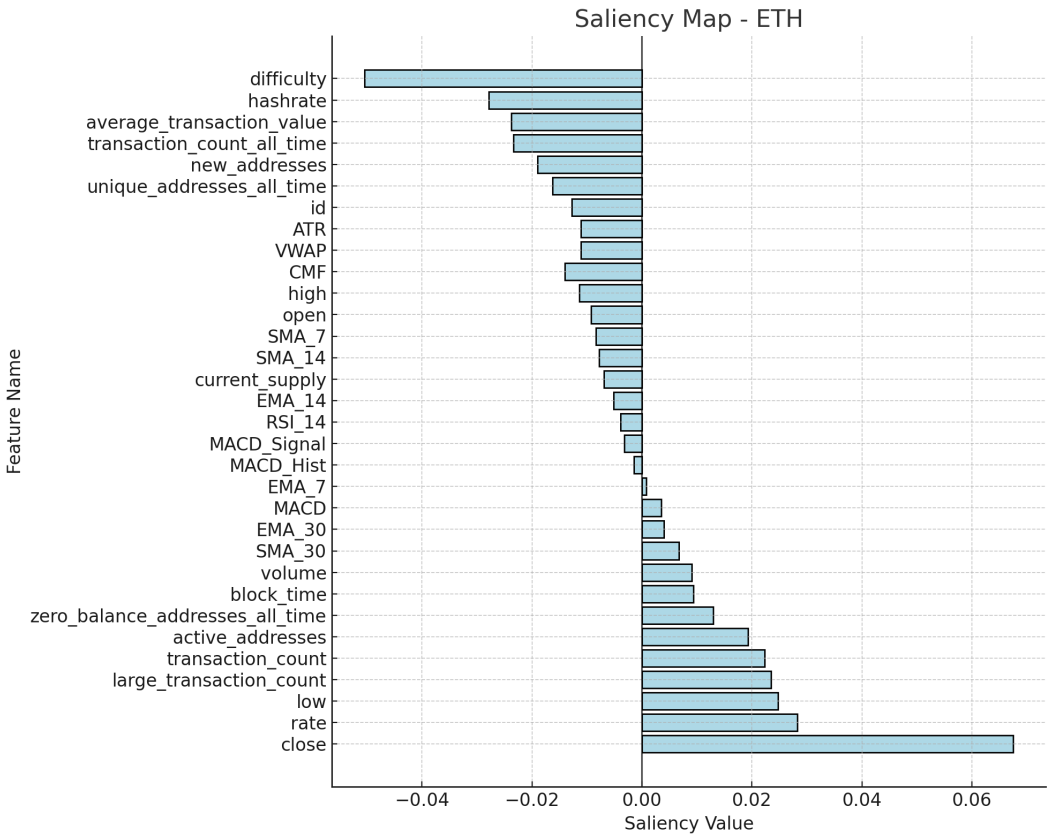


**Figure 5.40** – Saliency Map Feature Ranking Barchart- ETH




The table 5.13 and fig. 5.39 display the results obtained from calculating saliency values of the DDPG model trained on BTC data to find the most relevant features. The saliency map (fig. 5.39) for Bitcoin (BTC) shows that the most influential features for the model are difficulty, hashrate, transaction count, EMA_14, SMA_14, and MACD Signal, all of which are either key blockchain parameters or technical indicators tied to market trends. These features have strong positive saliency, indicating that the model heavily relies on them for predictions. Conversely, features like VWAP, block_time, and new_addresses have significant negative saliency, suggesting they may be less relevant or introduce noise to the predictions. Transaction-related metrics tend to have a lower or negative influence, while technical analysis indicators and blockchain data have a significant effect.

The table 5.14 and fig. 5.40 display the results obtained from calculating saliency values of the DDPG model trained on ETH data to find the most relevant features. The saliency map (fig. 5.40) for Ethereum (ETH) shows that the most influential features for the model are close, rate (staking rate), transaction_count, large_transaction_count, and low which significantly enhance the model's accuracy by capturing essential market trends like price movements, transaction activity and reflects heightened market engagement. Some features negatively impact the model's performance, notably difficulty and hash rate which were kept in the training data set to test the effectiveness of the

XAI algorithms. After Ethereum transitioned to PoS (Proof of Stake), these metrics should not affect the price movement and the DRL model here perfectly reflects that. Additionally, metrics like average_transaction_value and transaction_count_all_time detract from the model's predictions, potentially due to irregular large transactions or cumulative data not reflecting current trends. Technical indicators such as VWAP and CMF also show moderate negative influence, indicating their limited relevance in this context.

| Feature ID | Name | Saliency | Rank |
|---|---|---|---|
| 15 | VWAP | -0.05048716068267822 | 33 |
| 16 | CMF | -0.027880162000656128 | 32 |
| 30 | block_time | -0.023731693625450134 | 31 |
| 17 | ATR | -0.0233375933021307 | 30 |
| 26 | average_transaction_value | -0.018962573260068893 | 29 |
| 27 | block_height | -0.016241516917943954 | 28 |
| 21 | new_addresses | -0.012767801061272621 | 27 |
| 25 | large_transaction_count | -0.011129477992653847 | 26 |
| 24 | transaction_count_all_time | -0.011103508993983269 | 25 |
| 31 | block_size | -0.010666931048035622 | 24 |
| 14 | MACD_Hist | -0.009270478039979935 | 23 |
| 11 | RSI_14 | -0.00835302285850048 | 22 |
| 4 | volume | -0.007769819349050522 | 21 |
| 8 | EMA_7 | -0.006876758765429258 | 20 |
| 2 | low | -0.005157697945833206 | 19 |
| 5 | SMA_7 | -0.0038511829916387796 | 18 |
| 32 | current_supply | -0.003223154228180647 | 17 |
| 13 | MACD_Signal | -0.001471478957682848 | 16 |
| 7 | SMA_30 | 0.0007022533682174981 | 15 |
| 10 | EMA_30 | 0.0007817124715074897 | 14 |
| 22 | active_addresses | 0.0034933267161250114 | 13 |
| 3 | close | 0.003979498520493507 | 12 |
| 20 | unique_addresses_all_time | 0.006701606325805187 | 11 |
| 12 | MACD | 0.009107591584324837 | 10 |
| 1 | high | 0.009385177865624428 | 9 |
| 0 | open | 0.013001052662730217 | 8 |
| 28 | hashrate | 0.019241642206907272 | 7 |
| 23 | transaction_count | 0.022276878356933594 | 6 |
| 18 | id | 0.023457426577806473 | 5 |
| 6 | SMA_14 | 0.024736925959587097 | 4 |
| 19 | zero_balance_addresses_all_time | 0.02827303670346737 | 3 |
| 9 | EMA_14 | 0.029179411008954048 | 2 |
| 29 | difficulty | 0.06755675375461578 | 1 |

**Table 5.13** – Saliency Map Feature Ranking - BTC

| Feature | Name | Saliency | Rank |
|---|---|---|---|
| 31 | difficulty | -0.0504871606826782 | 34 |
| 28 | hashrate | -0.0278801620006561 | 33 |
| 26 | average_transaction_value | -0.0237316936254501 | 32 |
| 24 | transaction_count_all_time | -0.0233375933021307 | 31 |
| 21 | new_addresses | -0.0189625732600689 | 30 |
| 20 | unique_addresses_all_time | -0.016241516917944 | 29 |
| 18 | id | -0.0127678010612726 | 28 |
| 17 | ATR | -0.0111294779926538 | 27 |
| 15 | VWAP | -0.0111035089939833 | 26 |
| 16 | CMF | -0.0140156673345094 | 25 |
| 1 | high | -0.0113578519031526 | 24 |
| 3 | open | -0.00927047803997994 | 23 |
| 5 | SMA_7 | -0.00835302285850048 | 22 |
| 6 | SMA_14 | -0.00776981934905052 | 21 |
| 32 | current_supply | -0.00687675876542926 | 20 |
| 9 | EMA_14 | -0.00515769794583321 | 19 |
| 11 | RSI_14 | -0.00385118299163878 | 18 |
| 13 | MACD_Signal | -0.00322315422818065 | 17 |
| 14 | MACD_Hist | -0.00147147895768285 | 16 |
| 33 | block_size | 0.000702253368217498 | 15 |
| 8 | EMA_7 | 0.00078171247150749 | 14 |
| 12 | MACD | 0.00349332671612501 | 13 |
| 10 | EMA_30 | 0.00397949852049351 | 12 |
| 7 | SMA_30 | 0.00670160632580519 | 11 |
| 4 | volume | 0.00910759158432484 | 10 |
| 2 | block_time | 0.00938517786562443 | 9 |
| 19 | zero_balance_addresses_all_time | 0.0130010526627302 | 8 |
| 22 | active_addresses | 0.0192416422069073 | 7 |
| 23 | transaction_count | 0.0222768783569336 | 6 |
| 25 | large_transaction_count | 0.0234574265778065 | 5 |
| 27 | low | 0.0247369259595871 | 4 |
| 29 | rate | 0.0282730367034674 | 3 |
| 30 | block_size | 0.029179411008954 | 2 |
| 0 | close | 0.0675567537546158 | 1 |

**Table 5.14** – Saliency Map Feature Ranking - ETH

# Chapter 6

# Discussion

## 6.1 Trends and Insights

In the DQN model, while both BTC and ETH show profit growth with higher fractional trades, BTC shows higher annual profits at a 20% fractional trade, whereas ETH shows greater gains at higher levels like 33% and 50%. The Sharpe ratios indicate consistent risk-adjusted returns for both, but BTC maintains slightly better stability overall.
In the A2C model, BTC and ETH show improvements in profitability over time, with BTC exhibiting steadier profit growth. ETH's results are more volatile, as indicated by the increase in maximum profit after 200 episodes, followed by a drop afterwards. Trading activity is higher for ETH at the 200-episode mark, while BTC maintains a more stable trading pattern. When it comes to fractional trading in A2C, Both BTC and ETH show similar trends, with BTC having steadier performance in profits and trades, while ETH shows more volatility, particularly in maximum profits and trades at higher fractional trade percentages. BTC generally outperforms ETH in terms of annual profits and Sharpe ratios.
In the PPO model,both BTC and ETH show consistent improvements in profitability as the number of episodes increases, with BTC generally outperforming ETH again in terms of Annual Profit and Sharpe Ratios. ETH shows more volatility, mainly in Maximum Profit, but the results still reflect positive growth across varying episodes. BTC shows steadier improvements in profitability with increasing fractional trades, while ETH exhibits more volatility but higher peaks in both profits and trades, particularly at lower and higher fractional trade levels.
In the DDPG model, BTC shows consistent profitability across episodes, with relatively stable trading activity, while ETH demonstrates gradual improvements in profit but with a significant reduction in trading activity over time. DDPG is the best-performing model for both BTC and ETH. This can be attributed to it's ability to handle continuous action spaces and the nature of the implementation, which perfectly suits the dynamic nature of cryptocurrency trading. The model shows the ability to make precise decisions on fractional trades to optimize returns while minimizing risk.

The comparative analysis between the DDPG model trained solely on OHLCV data and that using both OHLCV and blockchain data showed a clear advantage in incorporating blockchain data. For both BTC and ETH, blockchain metrics such as difficulty, hashrate, staking rate and transaction count had a significant and noticeable impact on model decisions. These metrics reflect on-chain activity and market sentiment and provide an additional layer of information that enhances the decision-making ability of the model. In conclusion, integrating blockchain data appears to provide a meaningful and significant improvement in trading performance for both BTC and ETH, especially for BTC, where the model not only generates higher profits but also drastically reduces the number of trades.

Overall, the DDPG model significantly outperforms when it comes to raw return metrics, making it the best model for profit generation. The DDPG model achieved an average return of 8.99% and an annual return of 107.95%, significantly outperforming the next-best model, which yielded an average return of 2.36% and an annual return of 28.26%. The relatively low Sharpe Ratio implies that there is a lack of balance between return and risk. However, given the highly volatile nature of the cryptocurrency market, it has the best performance with relatively low volatility in the returns generated compared to the other models.

The results from the XAI layer and two methods suggest that for BTC, the model's performance can be improved, by focusing on refining the top positive features and removing or re-evaluating features with high negative saliency, such as VWAP, CMF and block_time, as they seem to negatively affect the model's performance. The results from the XAI layer and methods suggest that for ETH, model's performance can be improved by focusing on positive influencers like close, rate (staking rate), transaction_count, large_transaction_count, and low thereby increasing the prediction accuracy of the model. The metrics difficulty and hash rate which were kept in the training data set to test the effectiveness of the XAI algorithms. After Ethereum transitioned to PoS (Proof of Stake), these metrics should not affect the price movement and the DDPG model here perfectly reflects that. Negative influencers like difficulty and hashrate, can be removed to increase the model's performance. Furthermore, features with moderate saliency, such as 'volume' and 'block_time', can be fine-tuned to optimize their contribution to the model's performance.

## 6.2 Practical Implication

The results of this thesis have significant implications for the field of automated cryptocurrency trading. The use of DRL algorithms, particularly DDPG, combined with blockchain metrics and explainability, presents a powerful framework for developing more robust and transparent trading strategies. The integration of XAI is especially relevant for traders and financial institutions looking to deploy AI-driven systems with a high degree of transparency and interpretability. Moreover, the ability to perform fractional trades provides an additional layer of risk management that is vital in volatile markets. This functionality will allow traders to manage exposure more effectively and

implement strategies that are better suited to real-time market conditions.

## 6.3 Limitation and Future Work

There are a few directions for future research. Firstly, expanding the scope of assets to include a wider range of altcoin cryptocurrencies, as well as testing it on traditional financial instruments, to provide more comprehensive insights and comparisons. Additionally, exploring multi-agent reinforcement learning (MARL) approaches could enhance decision-making in more complex environments, where multiple agents interact with the market simultaneously. Further testing and the addition of more XAI techniques could potentially lead to even more interpretable models, enabling deeper insights into the decision-making processes of DRL algorithms. Lastly, all the DRL algorithms implemented in this study focused on maximising profits due to the nature of the trading environment and the reward structure. Additionally, a more sophisticated reward structure can be designed and implemented which finds the optimal balance between maximizing profit and minimising risk.

# Chapter 7

# Conclusion

This study explored the application of several Deep Reinforcement Learning (DRL) algorithms, namely DQN, A2C, PPO, and DDPG, for cryptocurrency trading, with an additional layer of explainability through Explainable AI (XAI) techniques. The results indicate that DRL models, particularly DDPG, outperformed others in terms of maximizing returns when both OHLCV and blockchain metrics were incorporated into the decision-making process. The models were tested across 100, 200, and 500 episodes on datasets containing Bitcoin (BTC) and Ethereum (ETH) data. Metrics such as average profit (%), annual profit (%), Sharpe ratios, and average trades per episode were evaluated. The integration of fractional trading, where trades were executed at percentages of available funds, allowed for more nuanced risk management. The results demonstrated that varying fractional trading percentages impacted the performance of the model, with optimized percentages resulting in a balanced risk-reward trade-off. DDPG, trained with both OHLCV and blockchain data, demonstrated the highest overall performance in terms of both profits and Sharpe ratios, showing significantly better performance than the existing next-best model. The DDPG model achieved an average return of 8.99% and an annual return of 107.95%, significantly outperforming the existing next-best model, which yielded an average return of 2.36% and an annual return of 28.26%. The XAI layer provided key insights into the most influential features for both BTC and ETH which helped better understand and interpret the models decisions.

## Acknowledgement

**In preparing this thesis, ChatGPT was utilized solely for correcting grammatical errors and enhancing readability through rephrasing; it was not used to generate any original content. I would also like to express my sincere gratitude to my thesis supervisor, Mostafa Chegenizadeh, for his unwavering support and guidance throughout the course of this research.**

# List Of Abbreviations

**API**: Application Programming Interface

**OHLCV**: Open High Low Close Volume

**ML**: Machine learning

**AI**: Artificial Intelligence

**XAI** Explaianable Artificial Intelligence

**RL**: Reinforcement Learning

**DRL**: Deep Reinforcement Learning

**DQN** : Deep Q-Network

**A2C** : Advantage Actor Critic

**PPO** : Proximal Policy Optmization

**DDPG** : Deep Deterministic Policy Gradient

**IG** : Integrated Gradient

**BTC** : Bitcoin

**ETH** : Ethereum

# Bibliography


Arrieta, Alejandro Barredo, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador García, Sergio Gil-López, Daniel Molina, Richard Benjamins, Raja Chatila, and Francisco Herrera (2019). *Explainable Artificial Intelligence (XAI): Concepts, Taxonomies, Opportunities and Challenges toward Responsible AI*. arXiv: 1910.10045 [cs.AI].

Böhme, Rainer, Nicolas Christin, Benjamin Edelman, and Tyler Moore (May 2015). "Bitcoin: Economics, Technology, and Governance". In: *Journal of Economic Perspectives* 29.2, pp. 213–38. DOI: 10.1257/jep.29.2.213.

Buterin, Vitalik (2013). *Ethereum White Paper: A Next Generation Smart Contract & Decentralized Application Platform*.

Chen, Yu Fu and Szu-Hao Huang (Nov. 2021). "Sentiment-influenced trading system based on multimodal deep reinforcement learning". American English. In: *Applied Soft Computing* 112. Funding Information: This work was supported in part by the Ministry of Science and Technology, Taiwan , under Contract MOST 110-2221-E-A49 -101 and Contract MOST 110-2622-8-009 -014 -TM1 ; and in part by the Financial Technology (FinTech) Innovation Research Center , National Yang Ming Chiao Tung University . Publisher Copyright: © 2021 Elsevier B.V., pp. 1–13. ISSN: 1568-4946. DOI: 10.1016/j.asoc.2021.107788.

Das, Arun and Paul Rad (2020). *Opportunities and Challenges in Explainable Artificial Intelligence (XAI): A Survey*. arXiv: 2006.11371 [cs.CV].

Deng, Yue, Feng Bao, Youyong Kong, Zhiquan Ren, and Qionghai Dai (2017). "Deep Direct Reinforcement Learning for Financial Signal Representation and Trading". In: *IEEE Transactions on Neural Networks and Learning Systems* 28.3, pp. 653–664. DOI: 10.1109/TNNLS.2016.2522401.

François-Lavet, Vincent, Peter Henderson, Riashat Islam, Marc G. Bellemare, and Joelle Pineau (2018). "An Introduction to Deep Reinforcement Learning". In: *Foundations and Trends® in Machine Learning* 11.3–4, pp. 219–354. ISSN: 1935-8245. DOI: 10.1561/2200000071.

Gort, Berend Jelmer Dirk, Xiao-Yang Liu, Xinghang Sun, Jiechao Gao, Shuaiyu Chen, and Christina Dan Wang (2022). *Deep Reinforcement Learning for Cryptocurrency Trading: Practical Approach to Address Backtest Overfitting*. DOI: 10.48550/ARXIV.2209.05559.

Guan, Mao and Xiao-Yang Liu (2021). *Explainable Deep Reinforcement Learning for Portfolio Management: An Empirical Approach*. arXiv: 2111.03995 [q-fin.PM].

Heuillet, Alexandre, Fabien Couthouis, and Natalia Díaz-Rodríguez (2020). *Explainability in Deep Reinforcement Learning*. arXiv: 2008.06693 [cs.AI].

Jiang, Zhengyao and Jinjun Liang (2017). *Cryptocurrency Portfolio Management with Deep Reinforcement Learning*. arXiv: 1612.01277 [cs.LG].

Kapengut, Elie and Bruce Mizrach (2023). "An Event Study of the Ethereum Transition to Proof-of-Stake". In: *Commodities* 2.2, pp. 96–110. ISSN: 2813-2432. DOI: 10.3390/commodities2020006.

Kochliaridis, Vasileios, Eleftherios Kouloumpris, and Ioannis Vlahavas (Apr. 2023). "Combining deep reinforcement learning with technical analysis and trend monitoring on cryptocurrency markets". In: *Neural Computing and Applications* 35, pp. 1–18. DOI: 10.1007/s00521-023-08516-x.

Koratamaddi, Prahlad, Karan Wadhwani, Mridul Gupta, and Sriram Sanjeevi (Mar. 2021). "Market sentiment-aware deep reinforcement learning approach for stock portfolio allocation". In: *Engineering Science and Technology, an International Journal* 24. DOI: 10.1016/j.jestch.2021.01.007.

Kumlungmak, Kittiwin and Peerapon Vateekul (2023). "Multi-Agent Deep Reinforcement Learning With Progressive Negative Reward for Cryptocurrency Trading". In: *IEEE Access* 11, pp. 66440–66455. DOI: 10.1109/ACCESS.2023.3289844.

Li, Yang, Wanshan Zheng, and Zibin Zheng (2019). "Deep Robust Reinforcement Learning for Practical Algorithmic Trading". In: *IEEE Access* 7, pp. 108014–108022. DOI: 10.1109/ACCESS.2019.2932789.

Li, Yuming, Pin Ni, and Victor Chang (June 2020). "Application of deep reinforcement learning in stock trading strategies and stock forecasting". In: *Computing* 102. DOI: 10.1007/s00607-019-00773-w.

Li, Yuxi (2018). *Deep Reinforcement Learning: An Overview*. arXiv: 1701.07274 [cs.LG].

Lillicrap, Timothy P., Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra (2019). *Continuous control with deep reinforcement learning*. arXiv: 1509.02971 [cs.LG].

Liu, Xiao-Yang, Hongyang Yang, Jiechao Gao, and Christina Dan Wang (Nov. 2021). "FinRL: deep reinforcement learning framework to automate trading in quantitative finance". In: *Proceedings of the Second ACM International Conference on AI in Finance*. ICAIF'21. ACM. DOI: 10.1145/3490354.3494366.

Lucarelli, Giorgio and Matteo Borrotti (2019). "A Deep Reinforcement Learning Approach for Automated Cryptocurrency Trading". In: *Artificial Intelligence Applications and Innovations*. Ed. by John MacIntyre, Ilias Maglogiannis, Lazaros Iliadis, and Elias Pimenidis. Cham: Springer International Publishing, pp. 247–258. ISBN: 978-3-030-19823-7.

Mnih, Volodymyr, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu (2016). *Asynchronous Methods for Deep Reinforcement Learning*. arXiv: 1602.01783 [cs.LG].

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller (2013). *Playing Atari with Deep Reinforcement Learning*. arXiv: 1312.5602 [cs.LG].

Nakamoto, Satoshi (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System*. DOI: http://dx.doi.org/10.2139/ssrn.3440802.

Nan, Abhishek, Anandh Perumal, and Osmar R. Zaiane (2020). *Sentiment and Knowledge Based Algorithmic Trading with Deep Reinforcement Learning*. arXiv: 2001.09403 [cs.AI].

Padial, Darío López (Apr. 2018). *Bukosabino/TA: Technical Analysis Library using pandas and Numpy*.

Sattarov, Otabek, Azamjon Muminov, Cheol Won Lee, Hyun Kyu Kang, Ryumduck Oh, Junho Ahn, Hyung Jun Oh, and Heung Seok Jeon (2020). "Recommending Cryptocurrency Trading Points with Deep Reinforcement Learning Approach". In: *Applied Sciences* 10.4. ISSN: 2076-3417. DOI: 10.3390/app10041506.

Schnaubelt, Matthias (2022). "Deep reinforcement learning for the optimal placement of cryptocurrency limit orders". In: *European Journal of Operational Research* 296.3, pp. 993–1006. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2021.04.050.

Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov (2017). *Proximal Policy Optimization Algorithms*. arXiv: 1707.06347 [cs.LG].

Schwager, Jack D (1995). *Technical analysis*. Vol. 43. John Wiley & Sons.

Sun, Shuo, Rundong Wang, and Bo An (Mar. 2023). "Reinforcement Learning for Quantitative Trading". In: *ACM Trans. Intell. Syst. Technol.* 14.3. ISSN: 2157-6904. DOI: 10.1145/3582560.

Théate, Thibaut and Damien Ernst (2021). "An application of deep reinforcement learning to algorithmic trading". In: *Expert Systems with Applications* 173, p. 114632. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2021.114632.

Wang, Shuyang and Diego Klabjan (2023). *An Ensemble Method of Deep Reinforcement Learning for Automated Cryptocurrency Trading*. arXiv: 2309.00626 [q-fin.TR].

Wu, Xing, Haolei Chen, Jianjia Wang, Luigi Troiano, Vincenzo Loia, and Hamido Fujita (2020). "Adaptive stock trading strategies with deep reinforcement learning methods". In: *Information Sciences* 538, pp. 142–158. ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2020.05.066.

Yang, Hongyang, Xiao-Yang Liu, Shanli Zhong, and Anwar Walid (2020). "Deep reinforcement learning for automated stock trading: an ensemble strategy". In: *Proceedings of the First ACM International Conference on AI in Finance*.

Yuan, Yong and Fei-Yue Wang (2018). "Blockchain and Cryptocurrencies: Model, Techniques, and Applications". In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 48.9, pp. 1421–1428. DOI: 10.1109/TSMC.2018.2854904.

Zhang, Zihao, Stefan Zohren, and Stephen Roberts (2019). *Deep Reinforcement Learning for Trading*. arXiv: 1911.10107 [q-fin.CP].

## Eidesstattliche Erklärung

Der/Die Verfasser/in erklärt an Eides statt, dass er/sie die vorliegende Arbeit selbständig, ohne fremde Hilfe und ohne Benutzung anderer als die angegebenen Hilfsmittel angefertigt hat. Die aus fremden Quellen (einschliesslich elektronischer Quellen) direkt oder indirekt übernommenen Gedanken sind ausnahmslos als solche kenntlich gemacht. Die Arbeit ist in gleicher oder ähnlicher Form oder auszugsweise im Rahmen einer anderen Prüfung noch nicht vorgelegt worden.

Zurich, 27/10/2024 ..... Ort, Datum

Manaemi Mondol ..... Unterschrift des/der Verfassers/in