VIETNAM NATIONAL UNIVERSITY – HO CHI MINH CITY
INTERNATIONAL UNIVERSITY
SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

Developing a Deep Q-Learning Agent for Vietnam Stock Trading

By

CHE TRUNG NGUYEN

Advisor: Dr. Mai Hoang Bao An

A thesis submitted to the School of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
Bachelor of Information Technology

HCMC, Vietnam – 2024

Developing a Deep Q-Learning Agent for Vietnam Stock Trading

APPROVED BY:

_____

(*Type Committee names beneath lines*)

_____

(*Typed Committee name here*)

_____

(*Typed Committee name here*)

_____

(*Typed Committee name here*)

_____

(*Typed Committee name here*)

THESIS COMMITTEE

(Whichever applies)

"A thesis submitted to the School of Computer Science and Engineering

in partial fulfillment of the requirements

for the degree of Bachelor of Computer Science "

HCMC – 06/2024

ACKNOWLEDGEMENT

First of all, I'd want to express my gratitude to my primary supervisor, Dr. Mai Hoang Bao An, for his ongoing assistance and guidance in building this Deep Q-Learning trading agent for the Vietnamese stock market. His passion, depth of knowledge and abilities were important in launching this idea and his encouragement and constructive feedback assisted personal growth. Because of his belief in my abilities, that inspired my enthusiasm and helped me overcome several challenges.

I'm grateful to all of my friends and classmates for their companionship and encouragement, which created an advantageous environment for me to develop and learn. Their companionship and encouragement taught me the value of support and teamwork. I learned that having a strong network of friends and classmates can boost confidence, provide different perspectives and foster personal growth. Their presence motivated me to push myself and strive for success in all aspects of life

Furthermore, I want to thank everyone who assisted with this project in a more indirect way. The knowledge and skills I have gained along the road are the result of the combined efforts of all of these incredible people, not just my own.

Finally, this Deep Q-Learning trading agent for the Vietnam stock market exemplifies everyone's contribution, support and hard work. The information acquired is just for the purpose of completing this graduation thesis and is not intended for any other use. I am grateful for the opportunity to work with such lovely people. Their advice has improved my career path and I am really excited to apply what I've learned in the future.

Student

Che Trung Nguyen

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

First of all, rapid developments in machine learning and artificial intelligence have changed many fields, especially with financial trading is the most affected. Historically, trading decisions were based on human emotions, also their technical analysis and fundamental analysis. The objective is to develop an intelligent agent (Deep Q-Learning Agent) capable of produces highly profitable trading decisions by using historical stock data and technical indicators.

In order to achieve this, the DQN agent is trained with historical data acquired from the VNDirect API. This data contains essential stock market information such as Open, High, Low and Close (OHLC) values, as well as calculating some technical indicators like moving average convergence divergence (MACD), relative strength index (RSI), commodity channel index (CCI) and average directional index (ADX), in order to teach the agent with a wide market overview, making it possible for making better decisions.

The trading environment has been replicated using an OpenAI Gym interface, providing a realistic and dynamic training experience. This environment continually provides data to the agent via rewards and penalties, allowing DQN-Agent to learn and adjust its tactics over time as needed. Furthermore, the training process consists of several episodes in which the agent interacts with its environment, makes decisions and learns from the results.

The DQN agent's performance is evaluated using both rigorous back-testing of historical data and real-time testing on actual market data. While back-testing helps determine an agent's capacity to generate money while maintaining high correct action rates in previous market situations, real-time testing, on the other hand, evaluates the agent's adaptability and efficiency in a live trading environment.

According to the results of this study, the DQN agent may occasionally deliver significant cumulative returns while retaining high accurate action rates and hopefully higher than traditional trading techniques. The agent's ability to learn from past experiences and might adapt to new data in order to demonstrates the effective use of reinforcement learning in financial trading.

Finally, the outcomes of this project may allow for the improvement of trade strategies and the development of more complex trading agents that provide not only actions but also volumes.

CHAPTER 1. INTRODUCTION

1.1 Background

The stock market is always on the move, with investors and traders looking to buy and sell shares in the hopes of making the biggest gains and maximize profits as possible. Historically, trading choices were made using emotions, technical analysis and fundamental analysis. However, it is an evident to everybody that automated trading strategies are becoming more popular as a consequence of developments in artificial intelligence and machine learning.

Machine learning, specifically reinforcement learning (RL), has showed promise in developing trading agents that can learn and adapt to market situations. RL helps an agent to learn from its surroundings by providing feedback in the form of rewards or punishments. Among RL algorithms Deep Q Learning (DQN) is notable for its capability to handle state spaces using deep neural networks.

In times the Vietnam stock market characterized by rapid economic growth and attractive market prospects, has emerged as a compelling investment option. Despite its potential, the market's volatility and unique characteristics pose significant challenges for traditional trading approaches. This dissertation aims to develop a DQN-powered trading agent tailored specifically for the Vietnam stock market. By leveraging historical data and key technical indicators such as Moving Average Convergence Divergence (MACD), Relative Strength Index (RSI), and Commodity Channel Index (CCI), the agent is designed to refine trading strategies and optimize performance.

The rapid growth of the Vietnam stock market over the past decade has attracted both domestic and international investors. This market's development has been driven by the country's robust economic performance, increased foreign direct investment, and progressive financial reforms. As the market continues to mature, there is a growing need for sophisticated trading strategies that can effectively manage the inherent risks and capitalize on opportunities.

Automated trading systems, particularly those based on machine learning, offer several advantages over traditional trading methods. These systems can process vast amounts of data, identify patterns, and execute trades with precision and speed, often outperforming human traders. Reinforcement learning, in particular, provides a framework for developing adaptive trading strategies that can respond to changing market conditions in real-time.

Additionally, the unique features of the Vietnam stock market, such as its relatively high volatility and liquidity constraints, are taken into account in the design of the trading agent. The DQN algorithm is adapted to handle these challenges, ensuring that the agent can operate effectively in this specific market environment. The ultimate aim is to develop a trading system

that not only maximizes returns but also manages risk effectively, providing a reliable tool for investors looking to navigate the Vietnam stock market.

1.2 Problem Statement

Navigating the stock market is like trying to solve a complex puzzle that's always shifting. It is full of sudden changes, heaps of information and much more outside factors that can affect stock price and human need to be quick analysis for smart investing. Traditional trading methods, which often rely on gut feelings, chart analysis and economic fundamentals, can be shaky ground-especially in emerging markets like Vietnam where local and global events can shake things up.

Stock trading, a cornerstone of the global financial system, presents numerous challenges to investors, regardless of their level of expertise. One of the primary difficulties in stock trading worldwide is market volatility. Stock prices can fluctuate wildly due to a variety of factors, including economic indicators, geopolitical events, and market sentiment. This unpredictability makes it difficult for traders to consistently make profitable decisions.

Another significant challenge is the vast amount of information that traders must process. Market data is generated continuously, encompassing everything from stock prices and trading volumes to financial reports and economic forecasts. The sheer volume and speed of this information can overwhelm even the most experienced traders, leading to analysis paralysis or hasty decisions based on incomplete data.

Market manipulation and insider trading also pose risks to traders. Despite regulatory efforts to maintain fair and transparent markets, there are instances where prices are influenced by unfair practices, making it difficult for retail investors to compete with institutional players who have access to more information and advanced trading tools.

Specifically, in the Vietnam stock market, these challenges are further exacerbated by the market's unique characteristics. The Vietnamese market, although growing rapidly, is still considered an emerging market with relatively high volatility and lower liquidity compared to more developed markets. This means that price swings can be more pronounced, and it can be harder to execute large trades without significantly impacting the stock price.

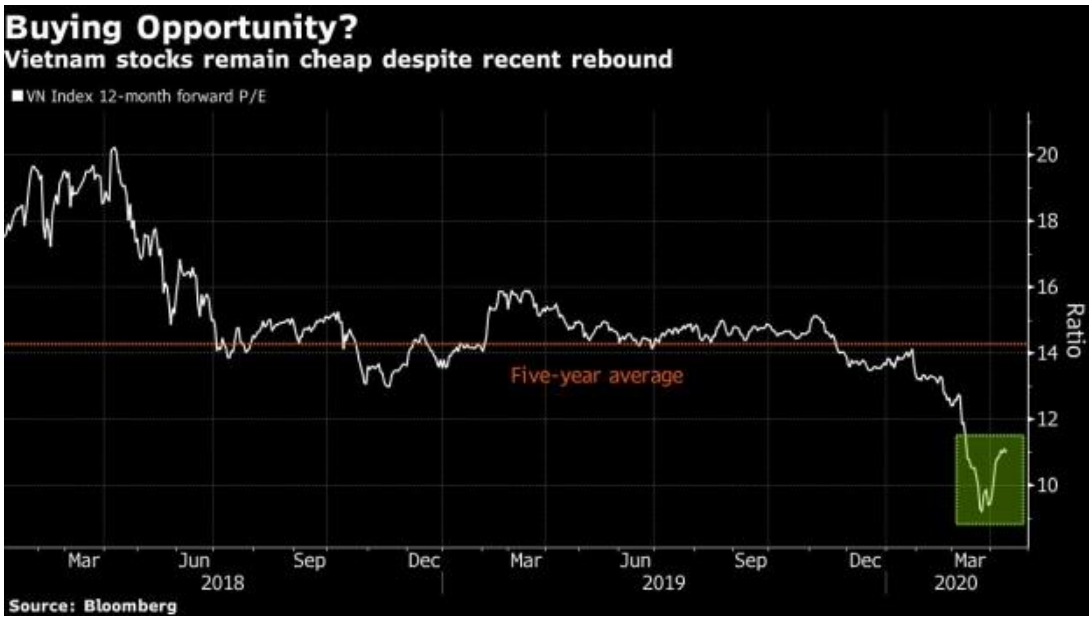

Figure 1: VN Index from 2018 to 2020

For example, Nguyen Kieu Giang from Bloomberg News has claimed that providing strategies for Vietnam stock market are incredibly complex. The Vietnamese stock market experienced extreme turmoil in March 2020, plummeting by 31% due to concerns over the coronavirus impact. However, by April, the market rebounded by 15%, becoming the world's best performer at that time. This recovery was driven by investor confidence in the government's swift and effective measures to manage the virus outbreak, as well as the attractiveness of super cheap valuations. Despite the rebound, the market remained in a bear territory, highlighting the challenges of navigating such a volatile environment. The pandemic's impact on exports, tourism, and income levels further complicates trading strategies in Vietnam.

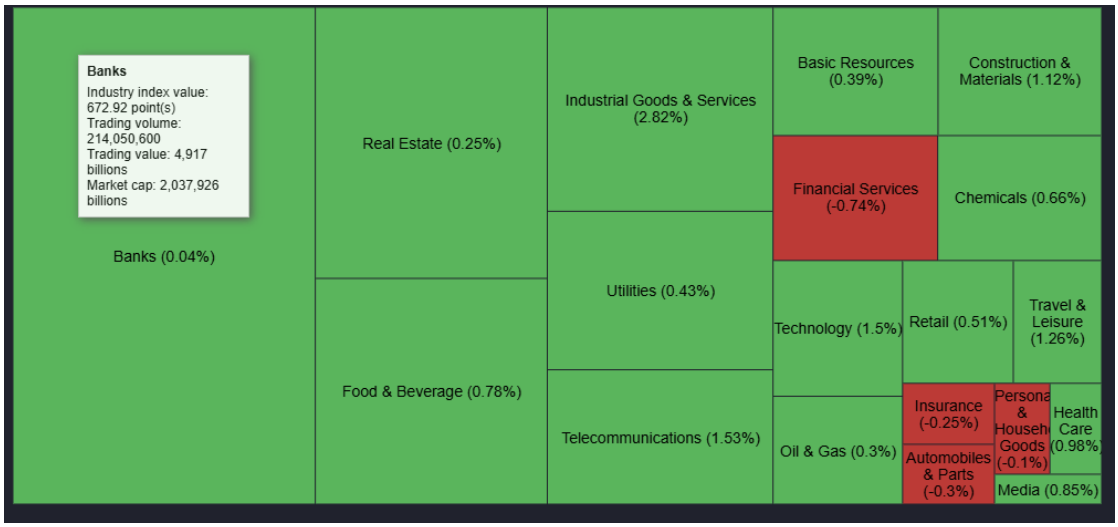

Figure 2: VNDirect Sector Movement Chart

As the heatmap show, up to 21st Jun 2024, the Banks sector has an industry index value of 672.92 points, reflecting its substantial weight in the overall market. It boasts a high trading volume of 214,050,600, which underscores the active participation and interest of investors in banking stocks. The trading value for this sector amounts to 4,917 billion, further emphasizing the significant capital flow within this industry.

Additionally, the market capitalization of the Banks sector is 2,037,926 billion, which highlights its large scale and critical impact on the market. Despite the relatively modest percentage change of 0.04%, the substantial trading volume and market cap indicate that the banking sector remains a focal point for investors. This stability, even with minimal percentage changes, suggests that banks are considered a safe and reliable investment option, particularly in volatile markets like Vietnam's.

Because of the banks' pivotal role in the financial system and their substantial impact on the overall market, they are often viewed as a barometer of economic health. The banking sector's high trading volume and significant market capitalization highlight the continuous interest and confidence of investors in this industry. These factors make banks a particularly attractive focus for trading strategies, as they offer ample opportunities for both stability and growth.

The main goal of this study is to create a clever trading system that can look back at old stock data, find trends and make smart trades that lead to big earnings. This project testing if a Deep Q-Learning Networks (DQN) agent can really pick up and put into action trading strategies that boost profits in Vietnam’s lively stock market. Ultimately, this project aims to make decision-making better, reduce mistakes people make and establish a strong base for automated trading in a market that’s full of surprises.

1.3 Scope and Objectives

This project is all about putting together, testing and checking out a Deep Q-Learning Networks (DQN) agent that's made just for the Vietnam stock market. This through a bunch of important steps: collecting data and calculating indicators, getting the right environment ready, teaching the agent and seeing how well it does. Historical stock data from the VNDirect API, encompassing essential market information and technical indicators, serves as the foundation for training the DQN agent.

The primary objectives of this research are as follows:

1. **Data Collection and Preprocessing:** To establish a thorough dataset for training the DQN agent, gather and preprocess historical stock data from the VNDirect API, including OHLC prices. Indicators like MACD, RSI, CCI and ADX should be also added.
2. **Environment Setup:** Using the OpenAI Gym interface, create a personalized trading environment that faithfully builds up the process of trading stocks. The environment should give the agent realistic feedback in the form of incentives and penalties.
3. **Agent Design and Training:** Use the DQN approach to train the agent on the historical dataset by combining target networks and experience replay. Multiple episodes will be included in the training process to make sure the agent can properly learn and adjust its trading methods.
4. **Performance Evaluation:** To determine whether the agent can make profit and keep up high right action rates, perform extensive back-testing on historical data. Furthermore, do real-time testing using current market data to assess the agent's flexibility and performance under real-world trading circumstances.
5. **Comparison with Traditional Strategies:** To determine wheter the DQN agent's relative efficacy and possible benefits, compare its performance with that of traditional trading techniques.
6. **Optimization and Improvement:** Determine the primary factors that affect of the agent's performance and investigate possible improvements, such as new features for the market, modifications to the reward function and testing of sophisticated reinforcement learning algorithms like Double DQN and Dueling DQN.

This study hopes to contribute to the field of machine learning in finance by demonstrating via these goals the viability and advantages of utilizing a DQN-based trading agent for automated stock trading in the Vietnamese market.

1.4 Target of Project

The primary goal of this project is aims to create an intelligent trading agent capable of making profitable trades on the stock market of Vietnam on its own. In order to create a robust and adaptable trading system that can withstand the ups and downs of an extremely volatile market, this project uses Deep Q-Learning Networks (DQN). These following are some of the project's specific goals:

1. Building a High-Performance Trading Agent: To identify the most beneficial trading opportunities, create a DQN-based agent that can examine technical indicators and historical stock data. Based on past market behavior, the agent should be able to modify its approach to maximize cumulative earnings.
2. Improving the Processes of Making Decisions: Use modern machine learning techniques to improve the agent's decision-making accuracy. The agent should use OHLC prices to provide complete market insights and a variety of technical indicators in order to make knowledgeable trading decisions.
3. Achieving Consistent Profitability: Put the DQN agent through an intensive education and testing procedure to ensure they can consistently deliver strong returns. The agent's performance has to be tested against traditional trading strategies in order to confirm its effectiveness.
4. Optimizing the Reward Function: Develop or refine a rewards system that maintains a balance between short and long-term financial gain. The agent should be encouraged by the reward function to engage in actions that lower risks and further long-term objectives.
5. Validating in Real-Time Trading: Utilizing up-to-date market data supplied from the VNDirect API, assess the agent's performance in a real-time trading scenario. In real-world situations, the agent ought to demonstrate its capacity to adjust to the state of the market and complete deals that are profitable.

1.5 Structure of Thesis

There will be five main components to this thesis project:

- Chapter 1. Introduction: This chapter gives a general overview of the project and focuses especially on developing a DQN agent for stock trading on the Ho Chi Minh Stock Exchange (HSX). It covers the project's goal, background, problem statement, scope and goals. It also gives a summary of the thesis structure.
- Chapter 2. Literature Review: In this chapter, the key technologies that make up the project's core stack are thoroughly examined. It covers using Python, JavaScript, HTML/CSS, MongoDB and reinforcement learning, more especially, Deep Q-Learning Networks and its application in financial trading.
- Chapter 3. Methodology: In this chapter, the system's structural architecture is described, along with the procedures involved in data collecting and preprocessing, trading environment setup, DQN agent design and training and implementation specifics. Moreover, it also contains explanations of the state transition, reward function, observation space, action space and deployed reinforcement learning techniques.
- Chapter 4. Implementation and Output Results: This section gets into the nitty-gritty of how the agent was built, taught and checked the stock trading agent. It lays out how put it all together, how the agent was trained and what happened when tested it, showing off how well the agent did with certain stocks during a set time frame.
- Chapter 5. Conclusions and Future Works: This chapter sums up about how well the DQN agent did in stock trading and really emphasizing how key the reward function is. It also focuses at possibilities for future research, such as enhancing the agent's performance and adaptability and further optimizing the reward function and initial balance.

CHAPTER 2. LITERATURE REVIEW

2.1 Introduction to Reinforcement Learning

Through interaction with its surroundings, an agent that uses reinforcement learning to learn to make judgments. It's all about doing actions that yield benefits, much like knowing how to make the greatest decisions to get desired results. The primary components of RL include:

- **State:** Represents the current situation of the environment.
- **Action:** The decision or move made by the agent.
- **Reward:** Feedback received from the environment based on the action taken.
- **Policy:** The strategy used by the agent to determine actions.

Reinforcement learning has worked out pretty well in different areas like robots, video games and money matters. This setup lets the computer programs learn from what they do and get better at making decisions as they go along. RL has made significant strides across various fields, demonstrating its versatility and potential to solve complex real-world problems. The reviewed documents provide an overview of RL applications and their impact in different domains.

The document "Reinforcement Learning Algorithms: An Overview and Classification" by AlMahamid and Grolinger (2021) offers a comprehensive classification of RL algorithms based on the environment types they are best suited for. The authors classify RL algorithms into three categories: environments with limited states and discrete actions, unlimited states and discrete actions and unlimited states and continuous actions. The study emphasizes the importance of selecting the appropriate RL algorithm for the specific environment type to solve problems efficiently.

The document highlights several widely used RL algorithms:

1. Deep Q-Networks (DQN): Suitable for environments with unlimited states and discrete actions, DQN uses deep neural networks to approximate the Q-value function, providing a robust method for solving problems with large state spaces.
2. Policy Gradient Methods: Including Reinforce, Trust Region Policy Optimization (TRPO) and Proximal Policy Optimization (PPO), these methods are effective for environments with continuous actions, where they optimize a parameterized policy to maximize expected rewards.
3. Actor-Critic Methods: Combining value-based and policy-based approaches, algorithms like A3C (Asynchronous Advantage Actor-Critic) and DDPG (Deep Deterministic Policy Gradient) offer solutions for complex control tasks in continuous action spaces.

These algorithms have been applied successfully in various domains, showcasing RL's capability to handle diverse and challenging tasks. The document "Reinforcement Learning Applications" by Yuxi Li (2019) explores the broad spectrum of RL applications, demonstrating its impact across multiple fields:

- **Healthcare**: RL is used for optimizing dynamic treatment strategies and medical image report generation. For example, Komorowski et al (2018) developed an RL-based policy for sepsis treatment, showing that RL can achieve better patient outcomes than traditional clinician policies by learning optimal treatment strategies from patient data. Furthermore, Devinder Thapa, In-Sung Jung and Gi-Nam Wang from AJOU University, South Korea (2008) came up with an idea in order to help clinicians make quick and accurate diagnosis decisions and choose the best course of therapy, integrate the agent-based decision support system with widely used results to enhance its intelligence.
- **Finance**: RL algorithms are applied to option pricing and order book execution. For instance, in order book execution, RL helps in developing strategies to optimize the trade execution process, reducing trading costs and improving profitability. Kearns and Nevmyvaka (2013) discuss the use of RL for optimizing the execution of large orders in financial markets, highlighting its effectiveness in handling high-dimensional and dynamic trading environments.
- **Robotics**: RL enables the development of advanced robotic systems capable of performing complex tasks autonomously. Applications include dexterous manipulation and legged robot locomotion. These RL-based systems can learn and adapt to new environments, improving their performance over time.
- **Transportation**: RL is employed in optimizing ride-sharing order dispatching and intelligent traffic light control. Hua Wei et al (2018) introduced Intellilight, an RL-based system for traffic light control that adapts to real-time traffic conditions, significantly improving traffic flow and reducing congestion.
- **Computer Systems**: RL is used in optimizing resource management and performance in computer systems. Applications include neural architecture search, device placement and data center cooling. Mao et al (2019) developed an open platform, Park, which supports RL-based optimization for various computer system tasks, demonstrating substantial improvements in system efficiency and performance.

2.2 Deep Q-Learning Networks

Because reinforcement learning (RL) is so good at improving and becoming more intelligent via experience, it has gained a lot of traction in the trading world. In reinforcement learning (RL), as opposed to supervised learning, which depends on labeled data, an agent learns

to make decisions by making mistakes and getting feedback in the form of incentives or punishments.

One well-known RL technique that combines Q-Learning and deep neural networks to enable the agent to manage big and complicated state spaces is called Deep Q-Learning Network (DQN). DQN makes use of a neural network to approximate the Q-value function. The expected reward of performing a certain action in a particular condition is estimated by the Q-value function. Important DQN developments include:

- Experience Replay: An approach to break correlation and enhance learning efficiency by storing and reusing previous experiences.
- Target Networks: are distinct networks that are employed to stabilize training by offering dependable target values.

Deep Q-learning (DQN) has established itself as a pivotal technique in the field of reinforcement learning, particularly for its ability to handle high-dimensional sensory inputs and learn complex control policies. The seminal work by Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, Martin Riedmiller (2013) demonstrated the application of DQN to Atari 2600 games, where a convolutional neural network (CNN) was trained using a variant of Q-learning. This approach allowed the network to learn directly from raw pixel inputs, effectively mapping visual input to actions that maximize future rewards. The use of experience replay and a target network were critical in addressing the challenges of correlated data and non-stationary distributions, which are inherent in reinforcement learning tasks. This method outperformed previous approaches in six out of seven games and even surpassed human expert performance in three, showcasing its robustness and efficacy in various game environments.

Following this breakthrough, research has focused on enhancing the stability and convergence of DQN. Balazs Varga, Balazs Kulcsar, Morteza Haghir Chehreghani (2022) proposed integrating robust control theory with DQN to improve its learning dynamics. They employed the Neural Tangent Kernel (NTK) to model the learning process as an uncertain linear time-invariant (LTI) system. This novel approach allowed the application of well-established control theory techniques to analyze and stabilize the learning process. By formulating robust controllers that inject dynamical rewards, the authors aimed to replace traditional heuristics like target networks and randomized replay buffers. Their findings suggested that robust controlled learning could achieve faster convergence and higher scores compared to standard Double Deep Q-learning (DDQN) in various OpenAI Gym environments. This control-oriented perspective not only improved performance but also provided a more transparent and analytically grounded approach to tuning DQN agents.

The integration of deep learning with reinforcement learning, particularly through DQN, has led to significant advancements in the ability of agents to learn from high-dimensional data.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, Martin Riedmiller (2013) work laid the foundation for using CNNs to handle raw visual inputs, addressing key issues such as the temporal credit assignment problem through techniques like experience replay. This method's success in the Atari 2600 games demonstrated the potential of deep learning to generalize across different tasks without game-specific adjustments.

Building on this foundation, the control-theoretic approach introduced by Balazs Varga, Balazs Kulcsar, Morteza Haghir Chehreghani (2022) represents a significant step towards more stable and explainable reinforcement learning algorithms. By leveraging robust control techniques, the authors provided a framework for understanding and enhancing the convergence properties of DQN. Their approach demonstrated that incorporating control theory into reinforcement learning could lead to more reliable and efficient learning systems, applicable to a wide range of complex environments.

In summary, the advancements in DQN, from its initial application in Atari games to the integration of robust control theory, highlight the evolving landscape of reinforcement learning. The combination of deep learning's representational power with control theory's analytical rigor holds promise for developing robust and efficient learning systems capable of tackling a variety of real-world tasks. These developments underscore the importance of interdisciplinary approaches in advancing the field of artificial intelligence.

2.3 Application of Deep Q-Learning in Stock Trading

Deep Q-Learning (DQN) has emerged as a powerful technique for automating stock trading strategies. The advancements in this field have shown mixed results across various applications and markets.

In the report "Optimizing Stock Trading Strategy With Reinforcement Learning" by Thakur et al (2021), a DQN-based approach was employed to maximize wealth through stock trading. The project aimed to develop a predictive model that integrates data science and web development to forecast future stock prices and make trading decisions. The DQN model was tested against traditional strategies such as buy-and-hold and Moving Average Convergence Divergence (MACD). The results indicated that the DQN-based strategy could outperform traditional methods under certain conditions, showcasing its potential in enhancing trading performance. However, the study also highlighted the challenges associated with model training and the importance of fine-tuning hyperparameters to achieve optimal results.

Similarly, in the paper "Deep Reinforcement Learning Approach for Trading Automation in The Stock Market" by Kabbani and Duman (2022), the authors applied the Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm, an extension of DQN, to automate stock trading. They formulated the trading problem as a Partially Observed Markov Decision Process

(POMDP) and used technical indicators and sentiment analysis to enhance state representation. The TD3 algorithm, which handles continuous action spaces, allowed for dynamic portfolio rebalancing. The back-testing results showed that the DRL approach achieved a high Sharpe ratio of 2.68 on the test dataset, indicating superior performance compared to traditional machine learning methods. This study demonstrated the effectiveness of DRL in generating profitable trades and optimizing investment strategies.

Building on these global advancements, the application of DQN to the Vietnamese stock market involves tailoring the approach to local market conditions and constraints, allowing for comprehensive trading strategies across stocks. This implementation uses technical indicators such as MACD, RSI, CCI and ADX to inform the agent's trading decisions, similar to the approaches discussed in the literature.

The integration of these technical indicators helps capture market trends and momentum, providing the agent with a robust feature set for decision-making. The DQN agent's learning process includes dynamic epsilon decay, prioritized experience replay and a target model for stable updates, ensuring efficient learning and the evolution of trading strategies over time.

Deploying the DQN-based trading system in the Vietnamese stock market allows the model to adapt to local market conditions and regulations, optimizing trading strategies for better returns. The use of real-time stock data through integration with the VNDIRECT API further enhances the model's ability to make timely and informed decisions, making it a valuable tool for traders in this market.

In summary, the advancements in DQN applied to stock trading highlight its potential to transform traditional trading strategies into adaptive, automated systems capable of handling complex market dynamics. The specific application to the Vietnamese stock market demonstrates the versatility of DQN in addressing different market environments and achieving optimal trading performance.

2.4 Python

Many technical developers and financial whizzes choose Python over other languages because of how easy to understand and write code with Python and it also offers a wealth of useful libraries and tools. It works well for everything from creating and executing machine learning models to preparing data to web development using Flask.

For machine learning and deep leaning, PyTorch and TensorFlow are two of the top open-source libraries for deep learning and machine learning. Google Brain's TensorFlow provides stable scaling and production-ready deployment, whereas Facebook's AI Research lab's PyTorch is commended for its accessibility and dynamic computational network.

The Python driver for MongoDB, a NoSQL database renowned for its scalability and flexibility, which is called PyMongo. Because of its dynamic structure, MongoDB is perfect for applications that require rapid development and iteration. Also data is kept as documents with varying formats and a JSON-like appearance. It's perfect for quick assembly and customization data within access deep into storage database.

Additionally, Flask is a Python microweb framework for web development. Because of its modular architecture and lightweight nature, it may be customized to match the unique needs of developers. Moreover, Flask can easily interface with databases such as MongoDB with PyMongo, which makes it possible to construct web apps that can perform CRUD operations on the database.

2.5 JavaScript and HTML/CSS

For this project, JavaScript is required in addition to HTML and CSS to create the front end of web-based financial trading systems. The usage of JavaScript as a high-level programming language is the main feature of web development, which is essential for enabling dynamic and interactive user experiences on the great majority of websites. Because of its foundational capabilities, notably first-class functions and asynchronous event handling, are indispensable for crafting real-time, uninterrupted user interfaces, this can be an important feature that is particularly vital in the context of an AI-driven trading application.

HTML (HyperText Markup Language) and CSS (Cascading Style Sheets) are the two primary technologies used in web development. Web page structure is defined by HTML, which also includes elements like headers, paragraphs and tables needed for trading and market data organization. Nevertheless, CSS is used to create these elements, making the program both aesthetically pleasing and easy to use.

This project seeks to construct a complete and effective trading system by utilizing JavaScript, HTML and CSS for frontend development and Python for backend development and machine learning. By integrating these technologies, the DQN agent is guaranteed to be able to handle and evaluate substantial amounts of financial data, make wise trading choices and provide users an intuitive and engaging interface.

2.6 MongoDB

Imagine MongoDB as a huge, cutting-edge library that excels at arranging a wide variety of books. It is extremely quick to aid consumers in discovering exactly what they're looking for and may extend to accommodate new books as needed. MongoDB, unlike traditional relational databases, stores data as documents that look like JSON. This enables MongoDB to handle a wide range of data types and formats with great flexibility. In settings like financial trading systems, where data formats might differ greatly, this document-oriented method is very helpful.

Because of its features, MongoDB is a great option for a range of financial trading systems use cases:

- Real-time Data Ingestion: MongoDB enables real-time data processing and ingestion by handling high-velocity data streams from market sources.
- Historical Data Storage: Thorough back-testing and strategy analysis are made possible by the database's capacity to hold substantial amounts of historical data.
- Technical Indicator Calculation: MongoDB's aggregation structure streamlines the data processing workflow by enabling the computation of technical indicators directly within the database.
- User Data and Preferences: By storing indicators and trade history, MongoDB's can easily adaptable design increases the level of flexibility offered by trading platforms.

Based on above usages, due of its scalability, flexibility, high performance and strong security features, MongoDB might be an excellent choice for designing financial trading systems. Trading platforms can function because of its capacity to handle a wide range of data types and substantial amounts of real-time data.

CHAPTER 3. METHODOLOGY

3.1 Overview

The methodology section outlines the process of developing and evaluating the DQN-based trading agent. The process involves several key steps:

1. **Data Collection and Preprocessing:** Gathering historical stock data and technical indicators and preparing the data for training.

2. **Environment Setup:** Creating a custom trading environment that simulates stock trading and provides observations, rewards and handles actions.

3. **Agent Design and Training:** Implementing the DQN agent and training it using the historical data within the defined environment.

4. **Evaluation and Performance Metrics:** Assessing the agent's performance using various metrics, including back-testing and real-time testing.

5. **Optimization and Improvement:** Based on the evaluation results, the agent's performance is further optimized through various techniques.

3.2 Data Collection and Preprocessing

The very first step in creating the trading agent is to collect and prepare historical stock market data. This project makes use of data collected from the VNDirect API, which offers detailed stock market information such as Open, High, Low and Close (OHLC) values.

3.2.1 Data Collection

Source: The historical stock data is derived from the VNDirect API, a dependable source of precise market data for many equities in the Vietnamese market. (The information gathered is strictly for the purpose of completing this graduation thesis and is not intended for any other use).

Content: The dataset includes essential information for each trading day, such as:

- Code: The stock ticker symbol (ex: "SHB").
- Date and Time: The specific date and time of the data point (ex: "2024-01-23", "15:01:05").
- Floor: The stock exchange where the stock is listed (ex: "HOSE").
- Type: The type of security (ex: "STOCK").
- Prices: This includes the basic price, ceiling price, floor price, open price, high price, low price, close price and average price.
- Adjusted Prices: These include adjusted open, high, low, close and average prices.

- Volume and Value: This includes normal market volume (nmVolume), normal market value (nmValue), put-through volume (ptVolume) and put-through value (ptValue).
- Changes: This includes the change in price, adjusted change and percentage change.


```
{
    "data": [
        {
            "code": "VCB",
            "date": "2024-01-23",
            "time": "15:01:05",
            "floor": "HOSE",
            "type": "STOCK",
            "basicPrice": 92,
            "ceilingPrice": 98.4,
            "floorPrice": 85.6,
            "open": 92.1,
            "high": 92.4,
            "low": 90.4,
            "close": 91.5,
            "average": 91.1,
            "adOpen": 92.1,
            "adHigh": 92.4,
            "adLow": 90.4,
            "adClose": 91.5,
            "adAverage": 91.1,
            "nmVolume": 1256900,
            "nmValue": 114506670000,
            "ptVolume": 98000,
            "ptValue": 8966400000,
            "change": -0.5,
            "adChange": -0.5,
            "pctChange": -0.5435
        }
    ],
    "currentPage": 1,
    "size": 9990,
    "totalElements": 1,
    "totalPages": 1
}
```


Figure 3: Raw data example

3.2.2 Data Preprocessing

Effective data preprocessing is plays as an important role for training the DQN agent to make informed trading decisions. Because of its important, this step involves several processes, including data cleaning, feature engineering and formatting. On top of that, technical indicators, which give the agent thorough insights into market conditions and trends is one of the main components of feature engineering.

Feature engineering is the process of developing new features from raw data in order to supply the DQN agent with more context and knowledge. These indicators are mathematical computations based on previous price, volume and open value that traders use to estimate future price changes.

The primary financial measures used for this project are:

1. Moving Average Convergence Divergence (MACD): This indicator helps traders detect momentum by comparing the average price over the past 12 days to the average price over the last 26 days. A nine-day EMA of the MACD, called the "signal line" is then plotted on top of the MACD line which can function as a trigger for buy and sell signals. It helps trader identify changes in the strength, direction, momentum and duration of a trend in a stock's price.
2. Relative Strength Index (RSI): is calculated using the formula: "RSI = 100 - (100 / (1 + RS))" where RS is the average of n days' up closes divided by the average of n days' down closes. RSI measures the speed and change of price movements. It is typically used to identify overbought or oversold conditions in a stock.
3. Commodity Channel Index (CCI): is determined by taking the normal price (average of high, low and close), removing the 20-period simple moving average and dividing the result by the typical price's mean absolute deviation. CCI examines cyclical patterns in a stock's price and determines if it is overbought or oversold.
4. Average Directional Index (ADX): is derived from the smoothed averages of the difference between "+DI" and "-DI" (Directional Indicators). It is used in conjunction with the directional movement index (DMI) and quantifies the strength of a trend. A high ADX obtain suggests a strong trend while a low ADX value reveals a weak one.

However, in order to have a faster calculation and less computation, in this project using builtin libraby from Python call 'ta'. For example, we can easily import its library and call RSI values of df just by using "RSIIndicator(df['close'])" and for other indicators just by OHLC values.

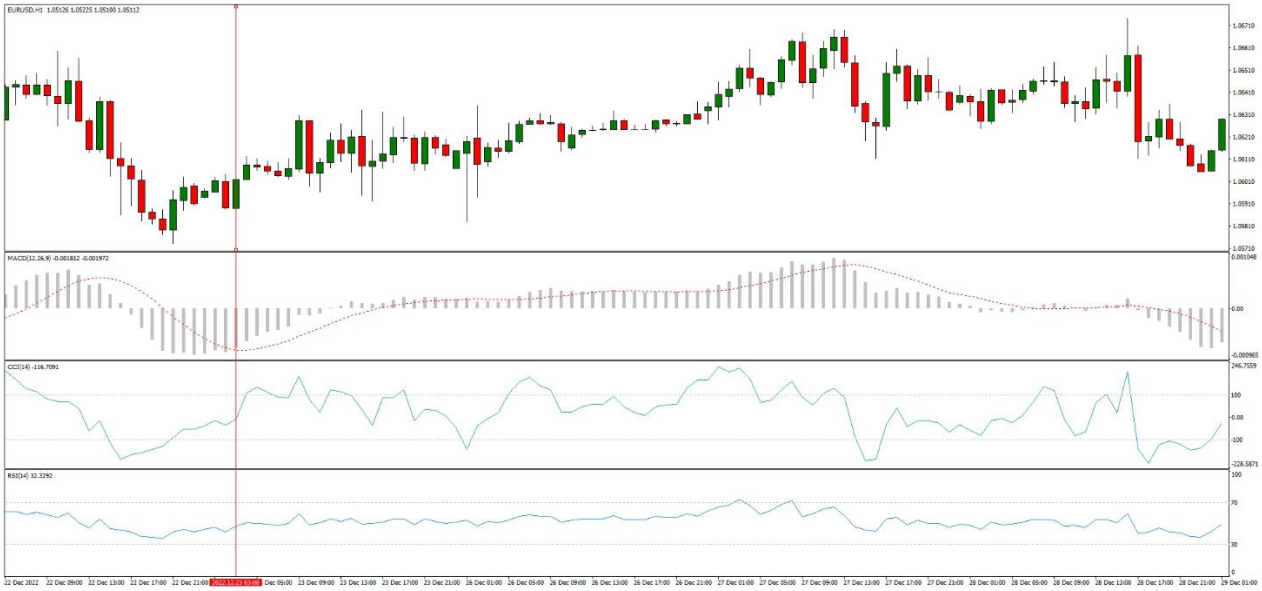

Figure 4: MACD RSI CCI strategy buy signal example

Figure 5: MACD RSI CCI strategy sell signal example

Due to its ability to assist traders in developing their trading system and risk management, the MACD RSI CCI approach is a favorite among traders. It may be combined with other technical indicators or used alone to provide a more thorough trading strategy. In the implementation step, these signals can be used as a part of reward function. For example, buy signal here mean the MACD line has crossed above the signal line, which suggests an uptrend, the RSI has risen above the oversold level (below 30) signaling a potential bullish reversal and the CCI has risen above the oversold level (below -100), indicating a potential bullish reversal.

3.3 Environment Setup

Creating a realistic and dynamic trading environment is essential for training the Deep Q-Learning Networks (DQN) agent effectively. This environment simulates the stock trading process, providing the agent with a platform to interact, learn and make decisions. The environment setup involves defining the observation space, action space, reward function and state transition mechanisms, all of which are critical for the agent's learning process.

The trading environment is designed using the OpenAI Gym interface, a popular framework for developing and testing reinforcement learning algorithms. The custom environment mimics the behavior of a real stock market, allowing the DQN agent to receive observations, take actions and receive rewards based on its performance.

3.3.1 Observation Space

The observation space represents the state of the environment at any given time. In this project, the observation space consists of a fixed-size window of historical data that captures critical market information and technical indicators used by the agent to make judgments. The most significant elements of the observing space are:

- OHLC Prices: Open, High, Low and Close prices for the stock over the observation window.
- Technical Indicators: Moving Average Convergence Divergence (MACD), Relative Strength Index (RSI), Commodity Channel Index (CCI) and Average Directional Index (ADX).

Hence with an observation window_size = 25 days, here is the example table of observation space. This matrix-like structure allows the agent to capture temporal dependencies and patterns in the data.

Day	Open	High	Low	Close	MACD	RSI	CCI	ADX
Day 1	12.15	12.25	12.05	12.10	0.015	45.30	-100.5	20.75
...	...	...	...	...	...	...	...	...
Day 25	12.05	12.20	12.00	12.10	0.010	46.50	-101.0	21.10

Table 1: Example of observation space

3.3.2 Action Space

The action space defines the set of possible actions that the agent can take at any given time. In this project, the action space consists of three discrete actions. These actions represent the fundamental decisions a trader must make and are designed to be simple yet effective in capturing trading strategies.

1. **Buy Action (2):** Buy one unit of the stock.
2. **Sell Action (1):** Sell one unit of the stock if there are shares has been held.
3. **Hold Action (0):** Do nothing and maintain all related values like share_held, last_trade, buy_price, …

3.3.3 Reward Function

The reward function is an important component that directly effects the agent's learning process and behavior by delivering feedback on actions performed. The reward function aimed to design balance between short-term and long-term profitability, while penalizing undesirable behaviors such as overtrading or holding during strong buy/sell signals market conditions.

- Profit-based rewards: agents get bonus points for profitable transactions and negative rewards for unproductive ones.
- Rewards are changed based on technical indications. For example purchasing when RSI is less than 30, showing oversold conditions, or selling when RSI is greater than 70, representing overbought situations, yields significant advantages..
- Holding Rewards/Penalties: Small incentives for holding during low volatility periods (ex: ADX below 20) and penalties for holding during strong buy/sell signals (ex: MACD and CCI signals).

3.3.4 State Transition

State transition is the process of shifting from one state to another based on an agent's activities and a period of time. In this environment:

- Action Execution: The agent's purchase, sell, or hold decision affects the portfolio's value and number of shares held.
- Market Update: New market data is added to the observation area when the environment moves on to the next time step.
- The reward for the activity was determined via changes in portfolio value and executing the reward function.
- State Update: The agent received a new state (observation) reflecting current market events and an updated portfolio.

3.3.5 Implementation Details

The custom environment is implemented using Python and integrated with the OpenAI Gym framework. Implementation steps:

1. Class definition: Define a new environment class that inherits from gym.Env..
2. Initialization (__init__ method): Initialize the environment's parameters, including the observation space, action space and initial portfolio state.
3. Reset function (def reset): Reset the environment to its initial state at the beginning of each episode then returning the initial observation.
4. Step function (def step): Implement the logic for executing actions, updating the state, calculating rewards and checking for episode termination.
5. Observation function (_get_observation): Generate the current state observation based on next step of df or the latest market data retrieved from API.

By providing a realistic and dynamic trading environment as described above, the project assures that the DQN agent can successfully learn and react to market conditions. This configuration lays a strong basis for the agent during the training phase, allowing DQN Agent to develop dependable trading strategies based on both historical and current data.

3.4 Agent Design and Training

The DQN agent is designed and trained to learn optimal trading strategies through interaction with the trading environment. This section details the reinforcement learning (RL) algorithm used, specifically focusing on the DQN algorithm, its architecture and the training process.

3.4.1 Reinforcement Learning Algorithm

3.4.1.1 Overview of Reinforcement Learning

Reinforcement Learning (RL) is a type of machine learning where an agent learns to make decisions by interacting with an environment. The agent aims to maximize cumulative rewards through a trial-and-error process. The key components of an RL algorithm include:

- **Agent:** the decision-maker that interacts with the environment.
- **Environment:** the observation space which the agent interacts and receives rewards or penalties.
- **State (s):** a representation of the current situation of the environment.
- **Action (a):** a set of possible moves the agent can make (buy, hold and sell).
- **Reward (r):** rewards and penalties from the environment that response to an action.
- **Policy (π):** a strategy that the agent uses to determine the next action based on the current state.

The learning process involves the agent taking an action in a given state, receiving a reward and transitioning to a new state. Over time, the agent learns to associate actions with rewards, enabling it to make better decisions that maximize cumulative rewards.

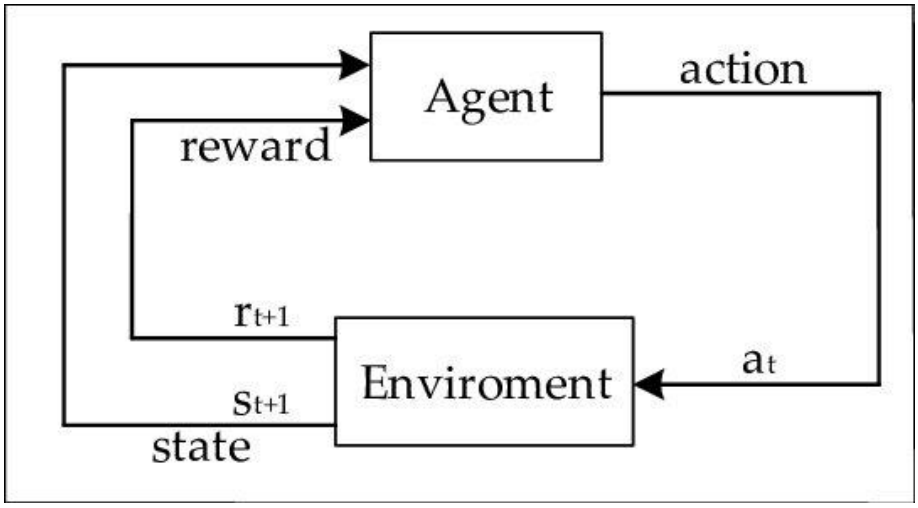

Figure 6: Reinforcement Learning Algorithm (Markov Decision Process)

3.4.1.2 The Q-Learning algorithm

The Q-table is a fundamental concept in reinforcement learning, particularly the Q-Learning algorithm. The Q-table is a matrix with rows representing states and columns indicating actions. The value at the intersection of a state and an action represents the expected cumulative reward (Q-value) for doing that action in that state and subsequently adopting the optimum policy.

In order to creating a trading agent for the Vietnam stock market, the Q-table may be used to describe the agent's grasp of the expected rewards for various trading actions under different market circumstances.

- States indicate market conditions using OHLC prices and technical indicators (e.g. MACD, RSI, CCI and ADX). To simplify, we can discretize these features into bins to create a finite set of states.
- Actions: the possible actions are Buy, Sell and Hold.

Below table is a simplified example where:

- The states are defined by discretized values of the closing price and RSI.
- The actions are Buy, Sell and Hold.

State (Closing Price, RSI)	Buy (Action 2)	Sell (Action 1)	Hold (Action 0)
(80, 30)	10	-5	1
(80, 50)	8	2	0
(100, 70)	-2	5	1
...	...	...	...

Table 2: Example Q-Table

The Q-Learning algorithm updates the Q-table iteratively using the Bellman equation. The goal is to find the optimal policy that maximizes the cumulative reward over time. The update rule for Q-values is given by:

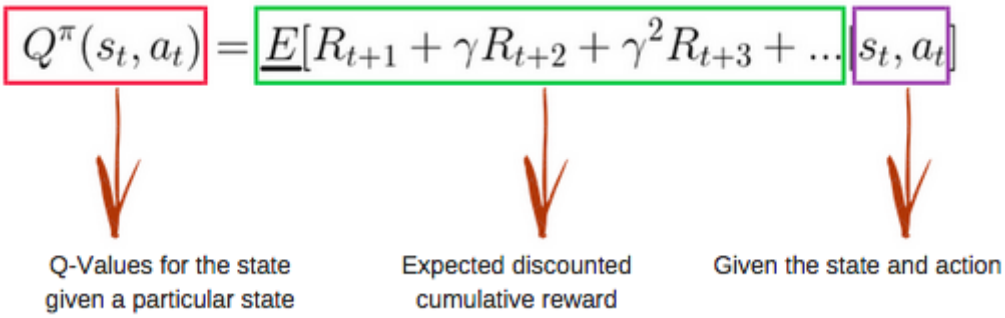

Figure 7: Q-Values function

Where:

- s is the current state.
- a is the action taken.
- r is the reward received after taking action a.
- α is the learning rate.
- γ is the discount factor.

3.4.2 Deep Q-Learning Networks Algorithm

An improved version of the Q-Learning technique called Deep Q-Learning Networks (DQN) uses deep neural networks to approximate the Q-value function. With this improvement, DQN can now handle high-dimensional state spaces, which is especially helpful in complicated settings like financial markets.

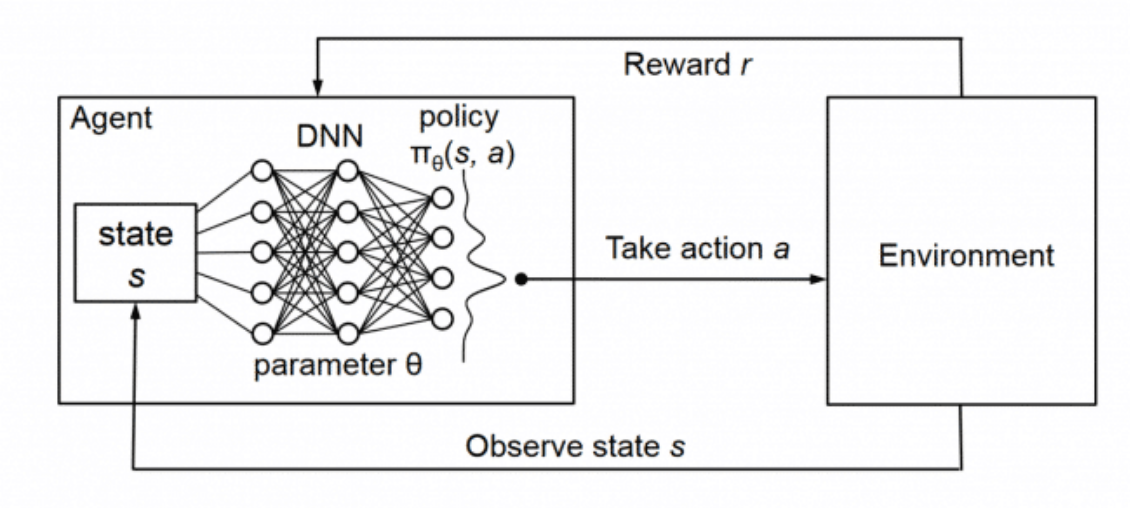

Figure 8: Deep Q-Network Algorithm

In order to approximate the Q-value function, DQN combines Q-Learning with deep neural networks. This enables the technique to scale to situations with huge and complicated state spaces. The essential elements of DQN consist of:

1. Q-Network: this neural network computes Q-values for every conceivable action after taking the state as input. To reduce the discrepancy between the goal and forecast Q-values, the network parameters are adjusted.
2. Experience Replay: this function keeps track of an agent's experiences, including action, reward, state and subsequent state in a replay buffer. Mini-batches of experiences are chosen at random from the buffer during training in order to enhance training stability by severing the link between successive events.
3. Target Network: an alternate neural network with the Q-network's design. Consistent target Q-values are provided during training and oscillations are avoided by frequent parameter adjustments to match the Q-network.

3.4.3 Training Process

The following training process involves the following steps and shown by pseudo-code below:

1. **Initialization:**
   - Initialize the Q-network with random weights.
   - Initialize the target network with the same weights as the Q-network.
   - Initialize the experience replay buffer.
2. **Interaction with the Environment:**
   - For each episode, start from an initial state.
   - For each time step within the episode:
     - Select an action using an epsilon-greedy policy, which balances exploration and exploitation.
     - Execute the action and observe the reward and next state.
     - Store the experience in the replay buffer.
     - Sample a mini-batch of experiences from the replay buffer.
     - For each experience in the mini-batch:
     - Compute the target Q-value using the target network.
     - Update the Q-network by minimizing the loss between predicted Q-values and target Q-values.
3. **Updating the Target Network:**
   - Periodically update the target network to match the Q-network, ensuring stable training targets.

Algorithm 1 Training Process for DQN Agent

Initialize parameters: α, γ, ϵ, ϵ_{min} , ϵ_{decay}, batch_size, memory_size, target_update_frequency
Initialize Q-network Q and target network Q' with random weights
Initialize replay memory M with capacity memory_size
for episode = 1 to num_episodes **do**
    $state \leftarrow environment.reset()$
    $total\_reward \leftarrow 0$
    $done \leftarrow False$
    while not $done$ **do**
        if $random() < \epsilon$ **then**
            $action \leftarrow random.choice(possible\_actions)$
        else
            $action \leftarrow \arg\max Q(state)$
        end if
        $next\_state, reward, done \leftarrow environment.step(action)$
        $total\_reward \leftarrow total\_reward + reward$
        $M.add(state, action, reward, next\_state, done)$
        if $len(M) >$ batch_size **then**
            $mini\_batch \leftarrow M.sample(\text{batch\_size})$
            for each $(state, action, reward, next\_state, done)$ in $mini\_batch$ **do**
                if $done$ **then**
                    $target \leftarrow reward$
                else
                    $target \leftarrow reward + \gamma \max Q'(next\_state)$
                end if
                $target\_f \leftarrow Q(state)$
                $target\_f[action] \leftarrow target$
                $Q.train\_on\_batch(state, target\_f)$
            end for
        end if
        $state \leftarrow next\_state$
        if $time_step\%$target_update_frequency $== 0$ **then**
            $Q'.set\_weights(Q.get\_weights())$
        end if
    end while
    if $\epsilon > \epsilon_{\text{min}}$ **then**
        $\epsilon \leftarrow \epsilon \times \epsilon_{\text{decay}}$
    end if
end for

Figure 9: Pseudo code for Training Process

CHAPTER 4. IMPLEMENT AND OUTPUT RESULTS

4.1 Implementation

Developing the web-based trading platform entails developing the original idea into a functional system that meets the needs of traders. This calls for thorough planning, in-depth examination of any existing systems and creative solution development to implement the intended new system. Before launching, the new website must be extensively tested to make sure everything goes according to plan and the demands of the users are met. Furthermore, extensive testing is required before agent deployment to ensure the agent functions as intended and meets the trade goals.

The DQN agent was trained on four tickers: 'VCB', 'MBB', 'BID' and 'EIB' which have been trading since 2015 and ended in the end of 2022. Due to that rich historical data, agent was able to develop successful trading strategies and learn from a range of market circumstances.

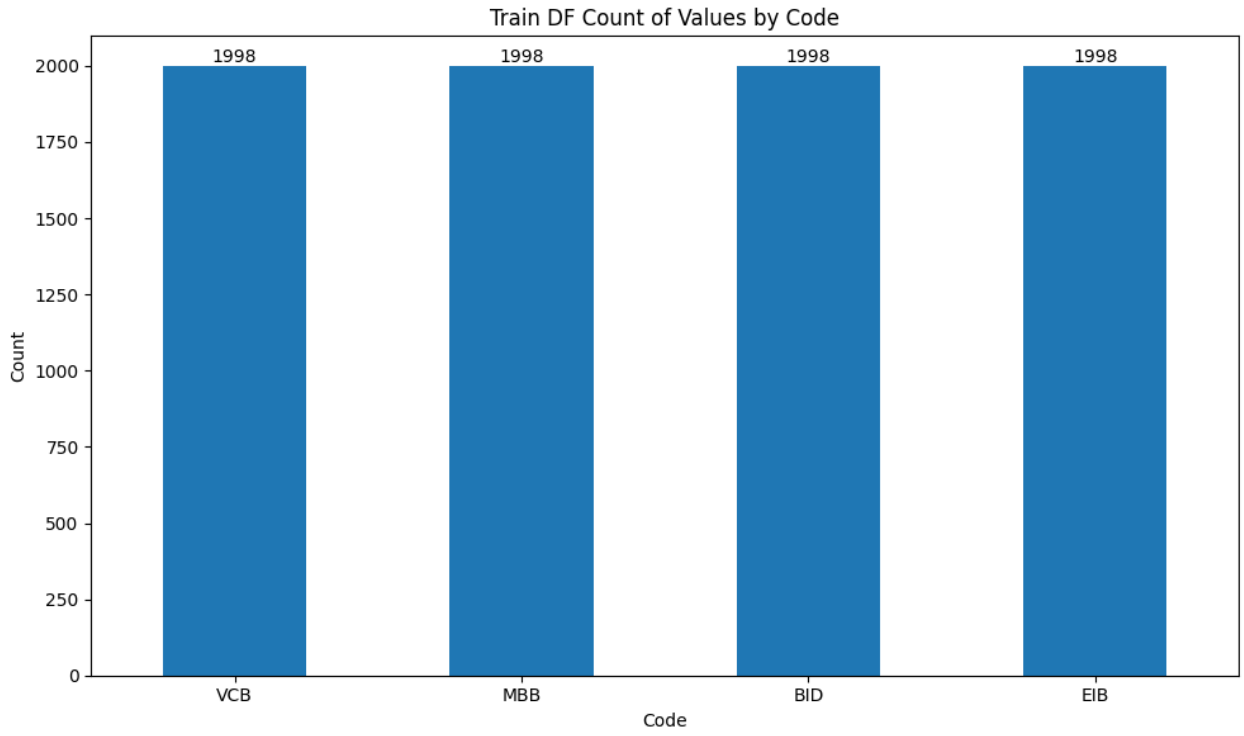

Figure 10: Length of train set

To test the trading strategy, the implementation entails configuring the Flask application, the DQN agent and the trading environment. Technical indicators like MACD, RSI, CCI and ADX are included into the trading environment to mimic stock trading. These indicators give the agent pertinent data to help with trading decisions by dynamically calculating them using past stock values.

PyTorch was used to create the DQN agent, which has a multi-layered neural network architecture to approximate Q-values for various actions. Training process stabilization was largely dependent on the agent's experience replay mechanism and target network updates. Iteratively interacting with the environment, storing experiences and upgrading the neural network using mini-batches sampled from the experience replay buffer were all part of the agent's training process.

The Flask application was utilized to offer an endpoint (/get_action_data) for stock trading action prediction based on the trained model during the assessment phase. The endpoint delivers the expected actions, portfolio value and balance over time after receiving a stock code and beginning balance as inputs. The agent behaves deterministically (with epsilon set to 0) to forecast actions based on the present state. The application's logic makes sure that the trading environment is reset for every evaluation run.

4.2 Exploratory Data Analysis

The very first step of any developing model and agent in machine learning and deep learning is data analysis, a part of preprocessing step. In this project, this step also help developer to examine the data and even understand what indicators should be use to determine efficiency reward function based on these indicators.

Firstly, checking about null values in dataset is essential, when checking on data that take from data that have been calculated indicators, the results will show that:

```
macd    76
rsi     52
cci     76
adx      0
dtype: int64
```

Figure 11: Data null check

This is because when calculating indicators, the algorithm needs at least first few data to compute, until these data points are available, the indicators value is not computable, resulting in NaN values for the initial periods. For example with MACD, an important indicator to capture pattern, using 26-period EMAs, it needs at least 25 data points for each code, and for 4 codes used in this project, it will have 100 of null values, other indicators are the same. Here are example of how MACD values look when it fit in close price:

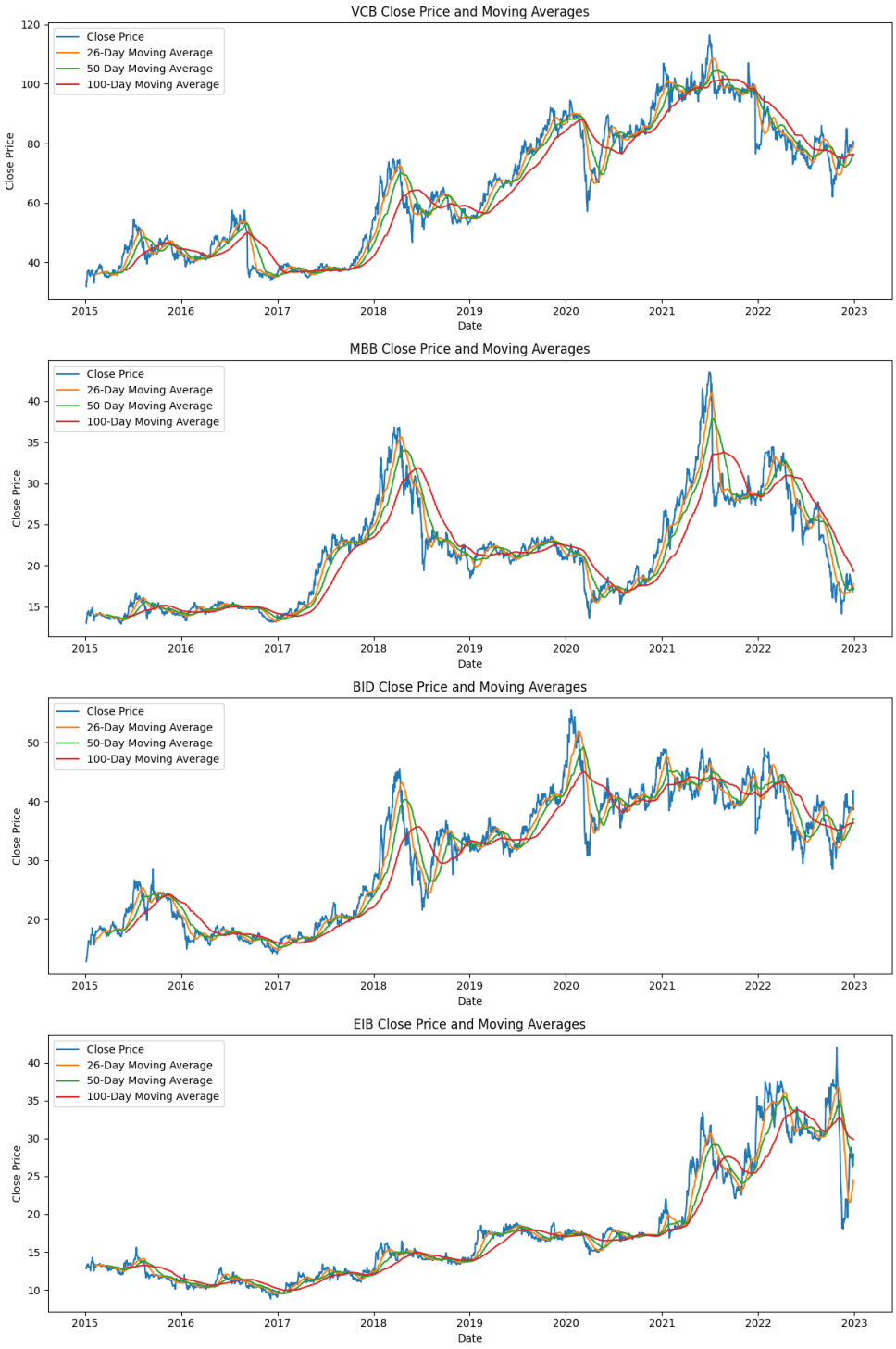

Figure 12: MACD strategy example

When talking about MACD, it can effectively capture pattern of time series data. The higher day of moving average, the longer term can be capture. However, it also filter out short-term pattern as can be seen. So in training process, the agent need to learn and analyze pattern that tracking exacly on moving pattern of closing price, 9-day moving average used as a signal line and MACD = 10-day EMA - 20-day EMA where 10-day EMA as a shorter-term moving average and

20-day EMA as A longer-term moving average so as to help to predict close price for few next days, which is really important for agent to learn as a buy or sell signal.

Next, let take a look at time series data of close price for each code:

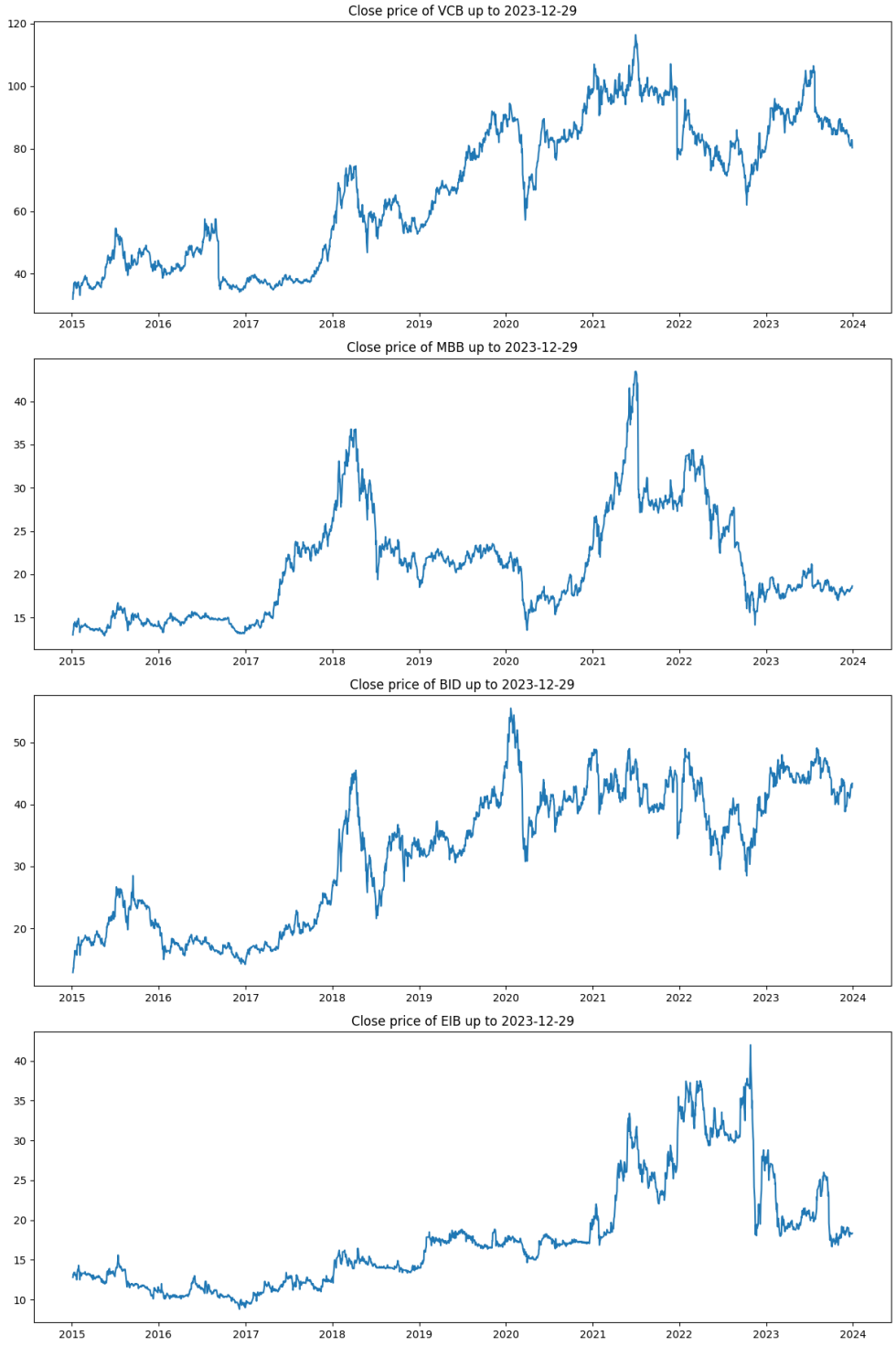

Figure 13: Close price of four codes

From this, it can be archived that the range for each code are different from each others. While VCB range is higher, from ~40 to over 100, others has lower max and min price. This mean when in training phase, the initial balance cannot be the same, it can affact the behavior of agent return invalid action like buy when balance is insufficient.

The data is split in to train and test set, 1998 rows and 249 rows respectively, which mean train_test_split ~= 0.11. The test size mean that agent can be tested over one year (2023) and check if it make profitable trades and analyze pattern of unseen data or not.

After exploratary data analysis, some of parameters and trading strategy are defined. It will be easier through traing process for initalize and optimize key factors of agent.

4.3 Training

For every ticker, the DQN agent was put through several episodes as part of the training process. The agent was configured with a learning_rate = 0.0005 and an epsilon_decay rate = 0.99, with a min_epsilon = 0.1. With a batch_size = 64 and a replay_frequency = 100, the target network was updated every 500 steps. So to guarantee there was enough money for trading, the observation space's window size was set to 25 and the starting balance was selected depending on the stock price range. All the parameters that used during training show as below table:

γ (Gamma)	**ε (Epsilon)**	**ε_{min}**	**ε_{decay}**	**α**	**episode**	**batch_size**	**replay_freq**	**target_upd_freq**
0.95	1.0	0.1	0.99	0.0005	1000	64	100	500

Table 3: Parameters table

Where γ (Gamma) is the discount factor, set to 0.95, which determines the importance of future rewards. A value close to 1 places more emphasis on long-term rewards, encouraging the agent to consider the long-term consequences of its actions. The value replay_freq can be set to higher in order to reduced computational overhead, reduce the number of times the replay function is called, which can decrease the computational burden and speed up training. However, the agent might adapt more slowly to new information since it updates its knowledge less frequently.

Historical stock prices from 2015 through the end of 2022 made up the training data. During this time, the agent received training to acquire efficient trading techniques. Data from 2023 to the present was utilized in the testing phase to assess the agent's performance under actual market circumstances.

The agent engaged with the surroundings in each training episode by acting (buying, selling, holding) then earning rewards according to how these activities turned out. These bonuses were intended to promote successful transactions and each time an agent acted with the appropriate indicator values. Not only rewards, agent can also be penalized when holding too

long or too short (if over 2*window_size or less than 2.5 days due to Vietnam stock market rule), it also penalized when taking action under bad indicators condition. The agent's performance improved over time as it learned to make better trading decisions. Finally in order to monitor and analyze the training progress, TensorBoard was used to log key metrics such as episode rewards, epsilon values, portfolio value and balance. This made it possible to track and visualize the agent's learning process in real time, giving insights into how the agent's performance changed over the course of episodes. In order to make sure that the agent was successfully learning from its interactions with the environment, the TensorBoard logs were also crucial in diagnosing and improving the training process.

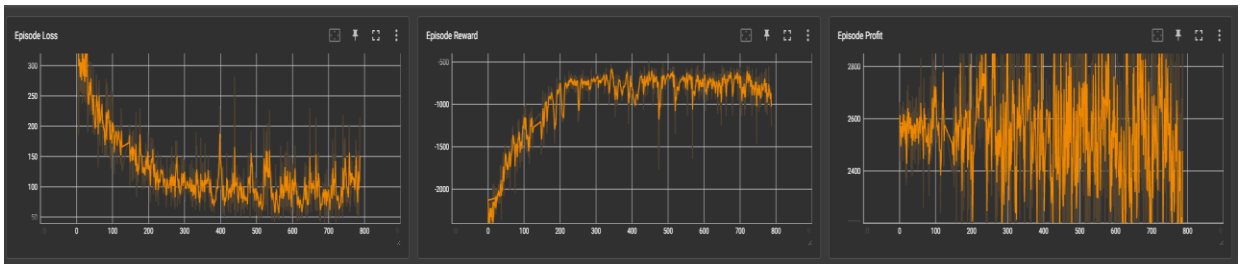

Figure 14: Example of Tensorboard log to track agent during training phase

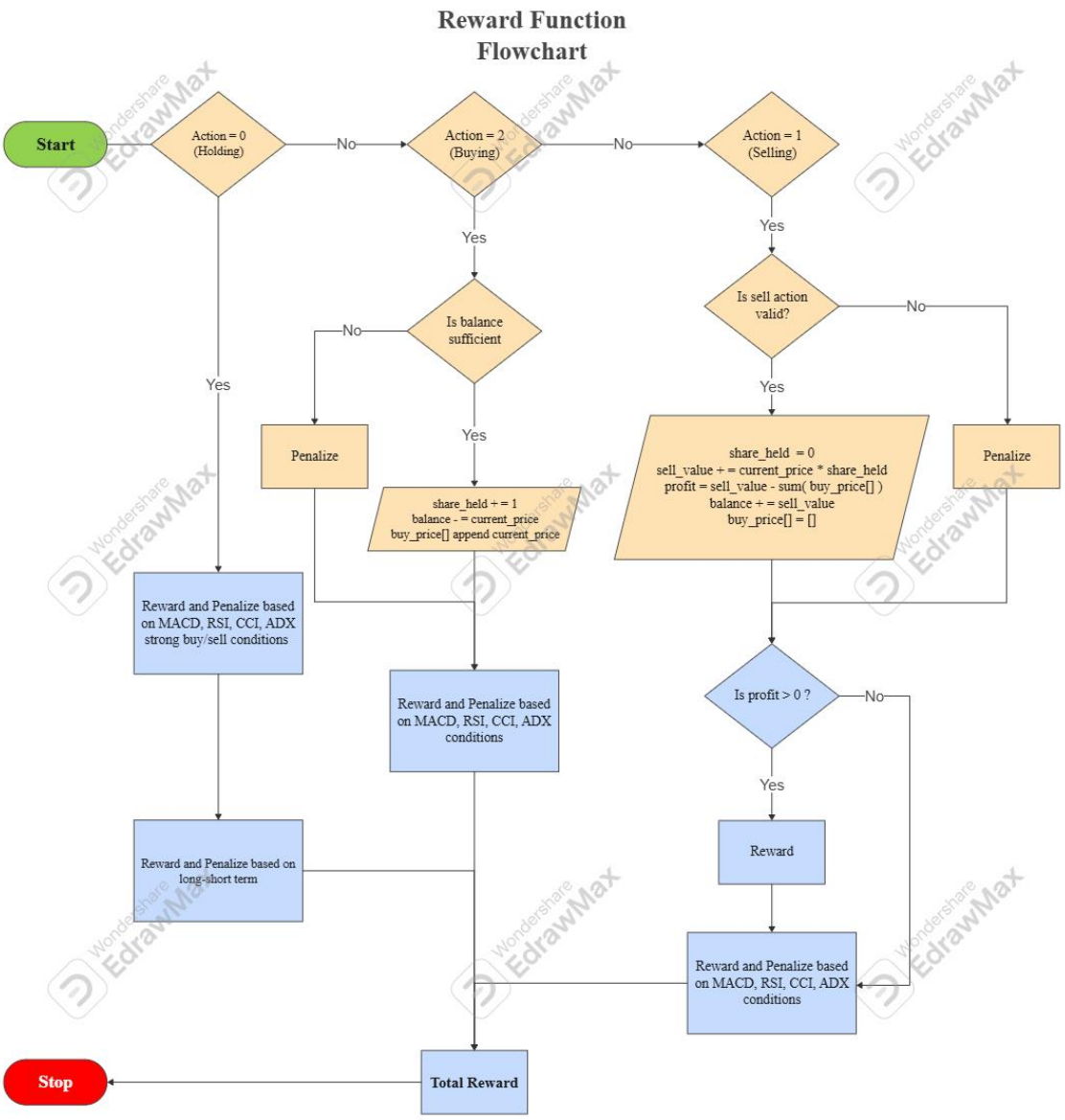

Figure 15: Reward function flowchart

The flowchart above outlines the decision-making process for a trading agent's actions and their corresponding rewards in order to leading the agent's behavior. It begins with the start node, leading to a decision node that checks if the action is holding (action = 0). If true, it calculates rewards based on MACD, RSI, CCI, and ADX conditions, particularly focusing on strong buy/sell signals. Then calculates rewards for holding from 2 days to 1-2 weeks and penalize if holding for too long.

In addition, if the action is buying (action = 2), another decision node checks if the balance is sufficient. If yes, it updates the shares held, decreases the balance by the current price, and appends the current price to the buy price list. Rewards are then calculated based on the technical indicators mentioned above. If the balance is insufficient, a penalty is applied.

For selling (action = 1), the flowchart checks if the sell action is valid (i.e., shares are held). If valid, it calculates the sell value, updates the balance, computes profit, and resets the shares held and buy price list. Rewards are then adjusted based on profit and technical indicators. If the sell action is invalid, a penalty is applied.

4.4 Testing and Output Results

During the testing phase, unseen data was used to assess the trained agent's generalization and performance in various market circumstances. This was made possible by the Flask application, which offered an easy-to-use interface for entering stock codes and starting balances as well as for monitoring the agent's performance.

Plotly charts that were incorporated into the Flask application's front end were used to illustrate the output results. The stock prices, agent activities, portfolio value and balance over time were all shown on the charts. An intuitive comprehension of the agent's trading behavior and its effect on the portfolio was made possible by the visual portrayal.

When assessing an agent's performance, for instance, on a particular stock, the resultant chart displayed the points on the price line, denoted by markers, where the agent chose to purchase or sell shares. The value and balance lines of the portfolio provide information about the trading strategy's financial performance. The trend in the portfolio demonstrated the agent's capacity to manage purchases and sales, limit losses and seize profitable chances.

Figure 16: Flask App User-Interface

The testing phase focused on evaluating the performance of the DQN agent on the four selected tickers ('VCB', 'MBB', 'BID' and 'EIB') over the year 2023. This period was chosen to assess how well the agent's learned strategies generalize to new, unseen data. Each ticker was tested individually and the agent's actions were recorded along with the resulting portfolio value and balance over time. Here are the length of test set and train/test split value.

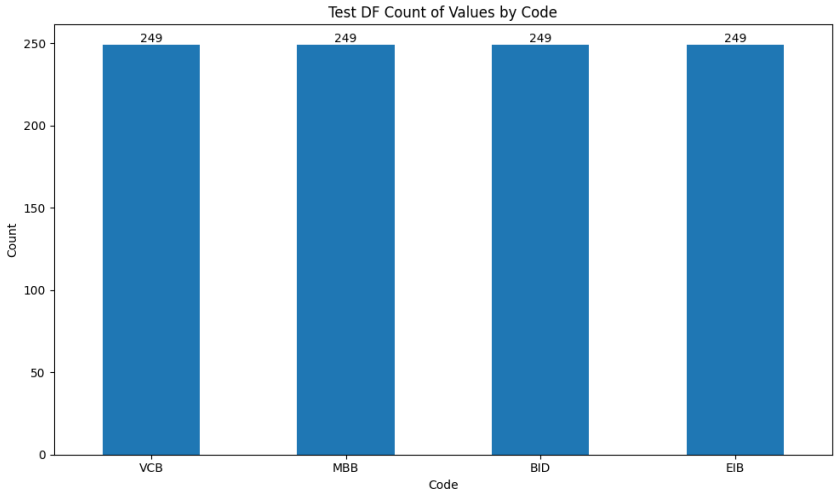

Figure 17: Length of test set

```
Train/Test split = 0.11081441922563418
```

Figure 18: Train/Test split value

Dealing with the testing set, indicators has been used as a buy and sell signal to create a new action column. This column create based on strong sell and buy conditions with MACD, RSI, CCI and ADX strategy only, not by agent. With the new action column, agent can test it accuracy with the strategy base only on information that data given, this can help to track and optimize in the future. Below are the figure of how action return based on strong buy and sell signal with MACD, RSI, CCI and ADX strategy:

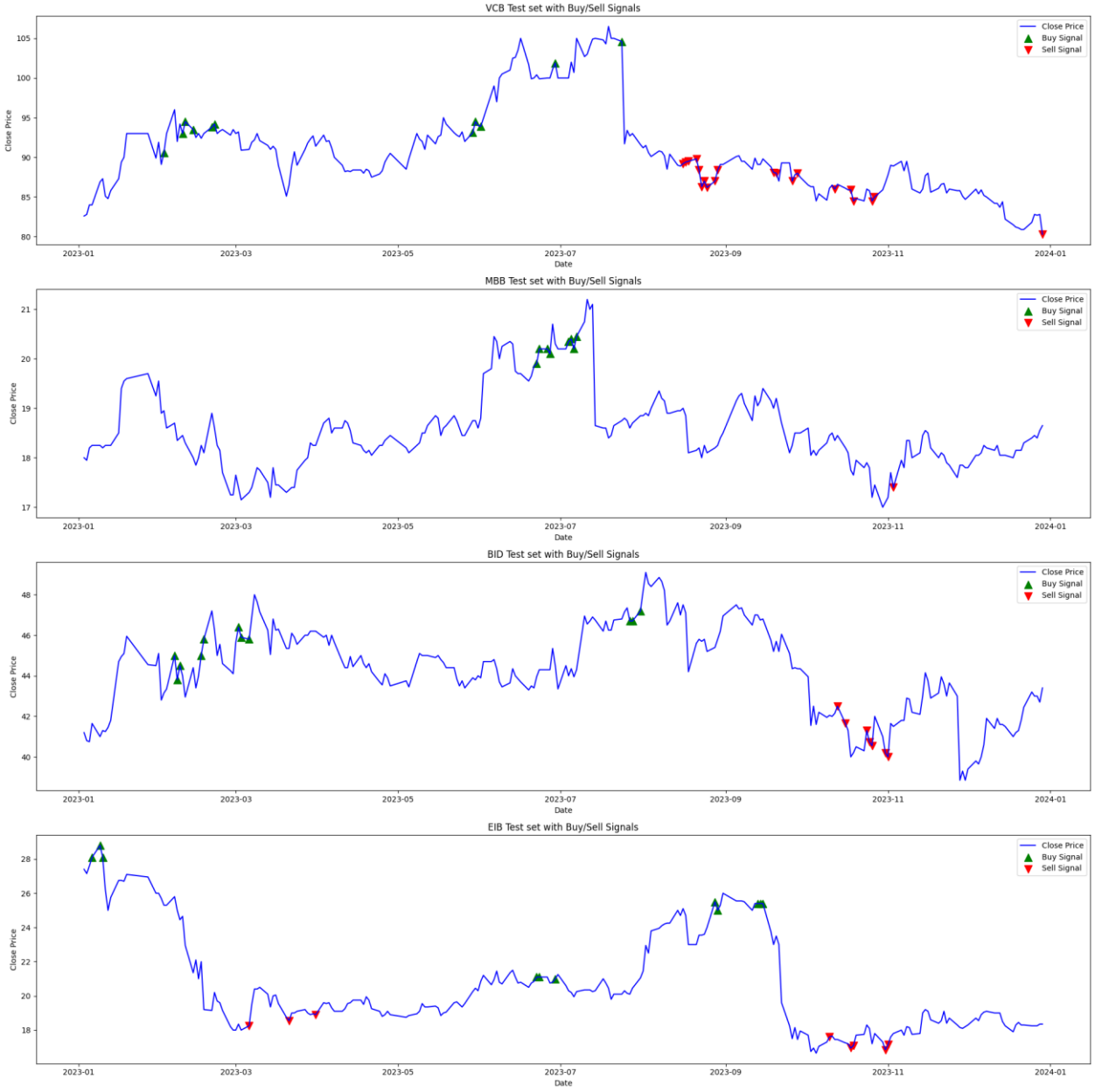

Figure 19: Actions based on strategy condition

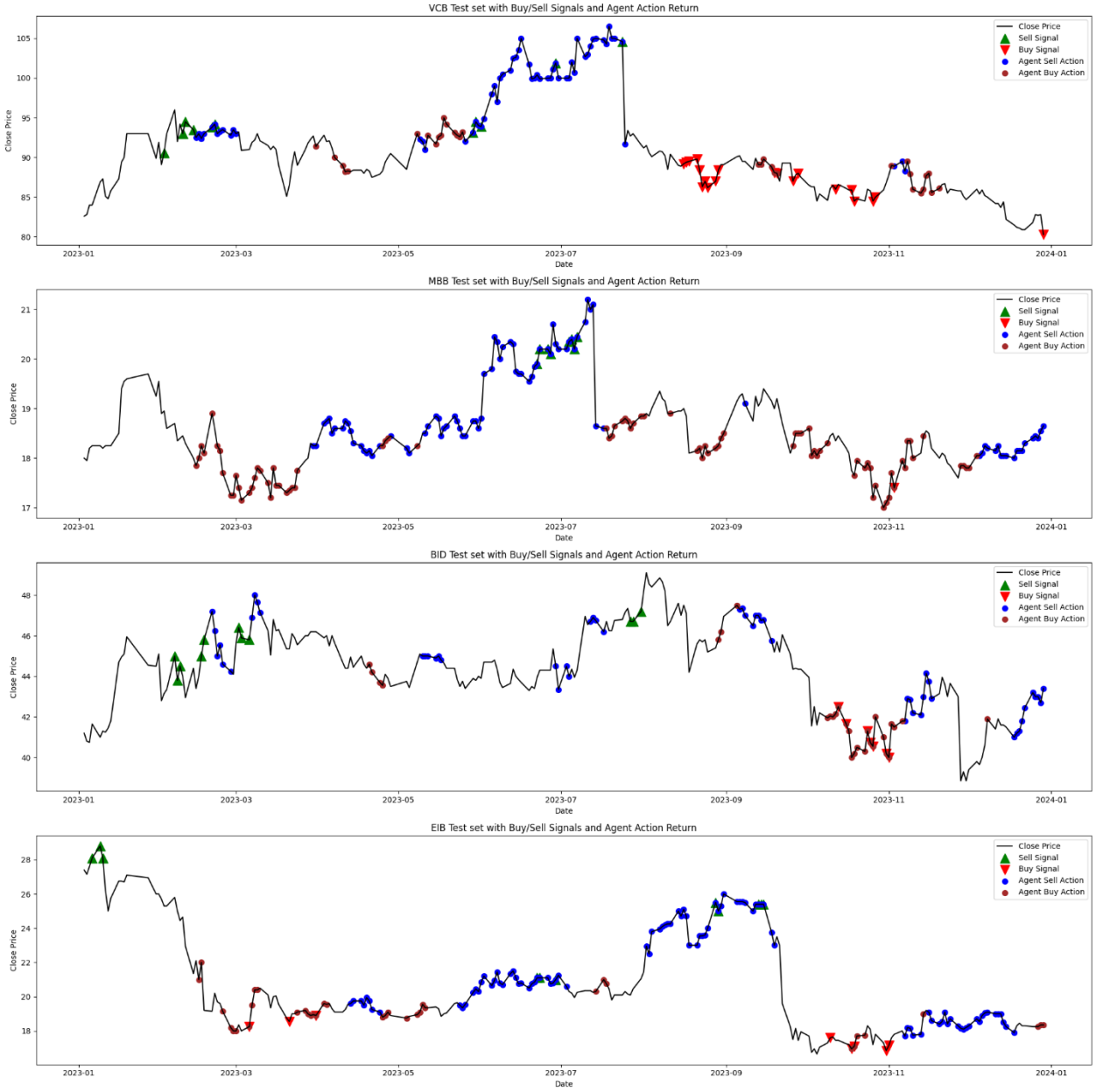

Figure 20: Actions based on strategy condition vs Actions agent return

```
Accuracy Actions of VCB: 29.03225806451613%
Accuracy Actions of MBB: 100.0%
Accuracy Actions of BID: 38.88888888888889%
Accuracy Actions of EIB: 63.1578947368421%
```

Figure 21: Agent's actions accuracy with traditional signals

The above figures illustrate differences between actions based on MACD, RSI, CCI and ADX condition with actions that the agent return on testing set from the beginning of 2023 to the end of 2023. These visualizations highlight the agent's trading decisions in relation to the buy/sell

signals generated by traditional technical indicators. By comparing the agent's actions with the traditional signals, the plots aim to evaluate the agent's ability to make profitable trading decisions, demonstrating its performance over the given timeframe. The accuracy image displays the accuracy of the agent's actions, showing that the agent achieved 100% accuracy for MBB, 63.16% for EIB, 38.89% for BID and 29.03% for VCB. The high accuracy for MBB suggests the agent closely followed the traditional signals, while the lower accuracies for VCB, BID, and EIB indicate a divergence in strategy, potentially highlighting areas for further refinement in the agent's trading algorithm.

Figure 22: VCB testing result

The chart for VCB illustrates the stock price movements alongside the agent's buy/sell actions and the account balance over time. The blue line represents the stock price, and the orange line indicates the balance. Green triangles denote sell actions, while red triangles signify buy actions. Initially, there are a few buy actions followed by sell actions as the price moves upward. The agent seems to sell frequently during price peaks, capturing profits. However, the balance fluctuates significantly, indicating that the agent's strategy might be too aggressive, leading to potential missed opportunities or unnecessary trades. Despite these fluctuations, the portfolio value at the end of the period is 2443.90.

Figure 23: MBB testing result

In the MBB chart, the stock price and agent's actions are depicted similarly to the VCB chart. The agent's buy actions (red triangles) are more concentrated in the early period when the stock price is lower, suggesting an attempt to capitalize on potential upward trends. As the price increases, the agent performs several sell actions (green triangles) to secure profits. However, there are multiple buy actions even when the stock price peaks, which might indicate an overly optimistic strategy. The balance remains relatively stable with occasional spikes, reflecting a cautious but potentially overly confident approach. The portfolio value at the end is 2534.20, indicating a better performance compared to VCB.

Figure 24: BID testing result

The BID chart shows the stock price and trading actions with distinct buy and sell signals. The agent performs buy actions (red triangles) during price reach its low and sell actions (green triangles) during price rises. This strategy aligns well with traditional trading principles, aiming to buy low and sell high. The balance fluctuates but shows significant increases during sell actions, suggesting effective profit-taking. However, frequent buy actions during price drops could indicate a riskier approach, potentially leading to higher volatility. The final portfolio value is 2517.90, demonstrating a relatively successful trading period despite some fluctuations.

Figure 25: EIB testing result

The EIB chart presents a clear depiction of stock price movements and the agent's trading actions. The agent's buy actions (red triangles) are well-distributed during price lows, while sell actions (green triangles) occur at price highs. This pattern indicates a strategic approach to buying low and selling high. The balance line (orange) shows stability with occasional spikes, reflecting successful profit-taking during price peaks. The portfolio value at the end of the period is 2516.50, suggesting a consistent and effective trading strategy with minimal unnecessary trades. This balance of actions indicates a well-optimized approach to trading EIB stock.

These analyses provide insights into the agent's trading strategies across different stocks, highlighting areas of success and potential improvements.

CHAPTER 5. CONCLUSIONS AND FUTURE WORKS

5.1 Conclusions

The implementation and evaluation of the DQN agent for stock trading on the Ho Chi Minh Stock Exchange (HSX) have demonstrated the potential of reinforcement learning in financial markets. By leveraging a well-structured trading environment and a robust neural network model, the agent effectively learned to make profitable trading decisions based on historical data. The results from testing on four key tickers ('VCB', 'MBB', 'BID', 'EIB') in 2023 showcased the agent's ability to navigate various market conditions and maintain a positive portfolio trajectory.

One of the significant achievements of this project was the development and refinement of a dynamic reward function. This reward function successfully balanced the need to encourage profitable trades and discourage unprofitable actions. By incorporating technical indicators such as MACD, RSI and CCI, the reward function provided nuanced feedback that guided the agent towards more informed trading decisions. The use of risk-adjusted return measures like the Sharpe ratio further enhanced the reward mechanism, promoting strategies that optimized returns while managing risk.

This agent is capable of suggesting actions, but not the specific volumes. Additionally, the initial balance is a critical factor that influences the agent's behavior, however, in this project, the agent can only return fixed actions while changing initial balance, this problem seem to be fixed and optimize in future work to adapt the real world cases. Optimizing these values along with the reward function can lead to better trading strategies and improved performance.

5.2 Future Works

While the present project has established a solid basis, there are a number of suggestions that future study and development might take to improve the functionality and practicality of the agent:

1. **Advanced Reward Function:**
   - **Customized Indicators**: To better capture risk-adjusted performance, keep enhancing the reward function by adding more complex financial indicators.
   - **Market Sentiment Research**: Integrating sentiment research from news and social media can give a more comprehensive context for trading choices.
2. **Enhanced Model Architecture:**
   - **Reinforcement Learning Algorithms**: Consider using additional reinforcement learning methods, such as Proximal Policy Optimization (PPO) or Soft Actor-Critic (SAC), to enhance learning efficiency and decision quality.
   - **Model Complexity**: To better capture complicated market trends, experiment with deeper neural network topologies or ensemble approaches.

- **Range of initial balance**: In order to provide more accurate actions with different initial balance, training agent with a range of balance might be an esstial feature of updated agent.

3. **Robustness and Adaptability:**
   - **Multi-Asset Trading**: Expand the model to support portfolios with various assets, allowing more diverse and sophisticated trading methods.
4. **Comprehensive Validation and Testing:**
   - **Long-Term Performance Evaluation**: Evaluate the agent's long-term performance and endurance by doing comprehensive back-testing over a number of years and market cycles.
   - **Stress Testing**: To evaluate the robustness and dependability of the agent, do stress testing in extremely volatile markets.

To sum up, this experiment has effectively shown that employing a DQN agent for HSX stock trading is reasonable. The development of very efficient and adaptable trading agents is possible with the combination of cutting-edge reinforcement learning techniques and ongoing reward function enhancement. To make sure our agent can handle more complicated tasks and be reliable, it need to be taught to be more adaptable, able to grow with the job and quick to respond to whatever the stock market throws their way.

REFERENCES

[1]. Jiang, Z., Xu, D., & Liang, J. (2017). A deep reinforcement learning framework for the financial portfolio management problem. [https://arxiv.org/abs/1706.10059]

[2]. Huang, Y. (2018). Financial Trading as a Game: A Deep Reinforcement Learning Approach. [https://arxiv.org/abs/1807.02787]

[3]. An introduction to Q-Learning: Reinforcement Learning by ADL (2018). [https://www.freecodecamp.org/news/an-introduction-to-q-learning-reinforcement-learning-14ac0b4493cc/]

[4]. Taylan Kabbani, Ekrem Duman (2022). Deep Reinforcement Learning Approach for Trading Automation in The Stock Market. [https://www.researchgate.net/publication/362706338_Deep_Reinforcement_Learning_Approach_for_Trading_Automation_in_The_Stock_Market]

[5]. Amey Thakur (2021). Optimizing Stock Trading Strategy With Reinforcement Learning. [https://www.researchgate.net/publication/358141909_Optimizing_Stock_Trading_Strategy_With_Reinforcement_Learning]

[6]. Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, Martin Riedmiller (2013). Playing Atari with Deep Reinforcement Learning. [https://arxiv.org/abs/1312.5602]

[7]. Balazs Varga, Balazs Kulcsar, Morteza Haghir Chehreghani (2022). Deep Q-learning: a robust control approach. [https://arxiv.org/abs/2201.08610]

[8]. Yuxi Li (2019). Reinforcement Learning Applications. [https://arxiv.org/abs/1908.06973]

[9]. Fadi AlMahamid, Katarina Grolinger (2022). Reinforcement Learning Algorithms: An Overview and Classification. [https://arxiv.org/abs/2209.14940]

[10]. Amrani Amine (2020). Deep Q-Networks: from theory to implementation. [https://towardsdatascience.com/deep-q-networks-theory-and-implementation-37543f60dd67]

[11]. Python Software Foundation. (2023). Python Documentation. [https://docs.python.org/3/]

[12]. PyTorch Documentation. (2023). [https://pytorch.org/docs/stable/index.html]

[13]. Flask Documentation. (2023). [https://flask.palletsprojects.com/en/2.0.x/]

[14]. Salvador Villalon (2018). How to build a web application using Flask and deploy it to the cloud. [https://www.freecodecamp.org/news/how-to-build-a-web-application-using-flask-and-deploy-it-to-the-cloud-3551c985e492/]

[15]. Flask API Documentation. (2023). Flask RESTful Quickstart. [https://flask-restful.readthedocs.io/en/latest/quickstart.html]

[16]. MongoDB Inc. (2023). MongoDB Manual. [https://docs.mongodb.com/manual/]

[17]. PyMongo Documentation. (2023). PyMongo 3.12.0 Documentation. [https://pymongo.readthedocs.io/en/stable/]

[18]. Plotly.js Documentation. (2023). Plotly JavaScript Graphing Library. [https://plotly.com/javascript/]

[19]. Overleaf Documentation. (2023). Overleaf, Online LaTeX Editor. [https://www.overleaf.com/learn]

[20]. LaTeX Project. (2023). LaTeX Documentation. [https://www.latex-project.org/help/documentation/]

[21]. Nguyen Kieu Giang, Bloomberg News (2020). [https://www.bnnbloomberg.ca/vietnam-stocks-become-world-s-best-after-extreme-turmoil-in-march-1.1421125]

[22]. Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, Marco Hutter (2019). Learning agile and dynamic motor skills for legged robots. [https://www.science.org/doi/10.1126/scirobotics.aau5872]

[23]. Devinder Thapa, In-Sung Jung, Gi-Nam Wang (2008). RL Based Decision Support System for u-Healthcare Environment. [https://www.intechopen.com/chapters/690]

[24]. Hua Wei, Guanjie Zheng, Huaxiu Yao, Zhenhui Li (2018). IntelliLight: A Reinforcement Learning Approach for Intelligent Traffic Light Control. [https://www.researchgate.net/publication/326504263_IntelliLight_A_Reinforcement_Learning_Approach_for_Intelligent_Traffic_Light_Control]

[25]. The Forex Geek (2024). MACD RSI and CCI Strategy. [https://theforexgeek.com/macd-rsi-cci-strategy/]

[26]. EdrawMax Online - Free Diagram Maker Powered by AI. [https://www.edrawmax.com/online/en/]