DYNAMIC PORTFOLIO MANAGEMENT WITH DEEP REINFORCEMENT LEARNING

WARAMETH NUIPIAN

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF MASTER OF SCIENCE IN
INFORMATION AND DATA SCIENCE
DEPARTMENT OF INFORMATION TECHNOLOGY
GRADUATE COLLEGE
KING MONGKUT'S UNIVERSITY OF TECHNOLOGY NORTH BANGKOK
ACADEMIC YEAR 2024

DYNAMIC PORTFOLIO MANAGEMENT WITH DEEP REINFORCEMENT LEARNING

WARAMETH NUIPIAN

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF MASTER OF SCIENCE IN
INFORMATION AND DATA SCIENCE
DEPARTMENT OF INFORMATION TECHNOLOGY
GRADUATE COLLEGE
KING MONGKUT'S UNIVERSITY OF TECHNOLOGY NORTH BANGKOK
ACADEMIC YEAR 2024

Thesis Certificate

The Graduate College, King Mongkut's University of Technology North Bangkok

Title	Dynamic Portfolio Management with Deep Reinforcement Learning
By	WARAMETH NUIPIAN

Accepted by the FACULTY OF INFORMATION TECHNOLOGY AND DIGITAL INNOVATION, King Mongkut's University of Technology North Bangkok in Partial Fulfillment of the Requirements for the Master of Science in Information Technology

Dean / Head of Department

Thesis Examination Committee

.... Chairperson

(Dr. CHOOCHART HARUECHAIYASAK)

.... Advisor

(Associate Professor Dr. PHAYUNG MEESAD)

.... Co-Advisor

(Assistant Professor Dr. MALEERAT MALIYAM)

.... Committee

(Dr. KANCHANA VIRIYAPANT)

Name	: WARAMETH NUIPIAN
Thesis Title	: Dynamic Portfolio Management with Deep Reinforcement Learning
Major Field	: Information and Data Science King Mongkut's University of Technology North Bangkok
Thesis Advisor	: Associate Professor Dr. PHAYUNG MEESAD :
Co-Advisor	: Assistant Professor Dr. MALEERAT MALIYAM
Academic Year	: 2024

ABSTRACT

This thesis confronts the complex challenges of the Stock Exchange of Thailand (SET) by leveraging Deep Reinforcement Learning (DRL) to develop optimized trading strategies. The thesis employs a rigorous methodology encompassing data collection, preprocessing, and the development of sophisticated algorithms for trade management and portfolio allocation. A Proximal Policy Optimization (PPO) agent integrates both short-term and long-term signals to enhance the strategy's adaptability and resilience in a volatile market environment, providing a sense of reassurance about its performance. Empirical testing revealed a significant outperformance of 2.67% compared to baseline strategies for the portfolio. A comprehensive comparative analysis demonstrates the DRL-based strategy's robustness across diverse market conditions unique to the SET. The study's contributions are threefold: It provides advanced analytical tools for investors and policymakers, offering data-driven insights and unbiased recommendations. It introduces a novel diversified portfolio management model tailored to the SET. It develops customized DRL algorithms that balance return maximization with risk mitigation. This research advances financial technology and decision-making in dynamic market settings by providing sophisticated mechanisms for addressing SET-specific challenges. The findings not only exemplify DRL's potential in optimizing SET trading strategies but also pave the way for future research and applications of artificial intelligence in emerging market environments.

(Total 97 pages)

Keywords : Reinforcement Learning (RL), Deep Q-Network (DQN), Stock Portfolio Optimization, Proximal Policy Optimization (PPO), Machine Learning in Finance

_______________________________________ Advisor

ACKNOWLEDGEMENTS

Completing this thesis could not have been possible without the participation and assistance of so many people. First, I would like to thank my parents for always supporting and encouraging me.

I would also like to thank my advisors, Associate Professor Dr. Phayung Meesad and Assistant Professor Dr. Maleerat Maliyaem, for their guidance and assistance throughout my research and the writing of this thesis.

Furthermore, I am deeply thankful to King Mongkut's University of Technology North Bangkok for providing me with the opportunity, resources, and support necessary to pursue this research.

WARAMETH NUIPIAN

TABLE OF CONTENTS

		Page
ABSTRACT		iv
ACKNOWLEDGEMENTS		v
TABLE OF CONTENTS		vi
LIST OF TABLES		vii
LIST OF FIGURES		viii
LIST OF ABBREVIATIONS		ix
Chapter 1 Introduction		1
1.1	Toward Reinforcement Learning in Portfolio	1
1.2	Objective	3
1.3	Scope	3
1.4	Outline of Thesis	3
Chapter 2 Related Works		5
2.1	Artificial Intelligence Methodologies in Stock Market Analysis	5
2.2	Stocks prediction	6
2.3	Deep Learning	8
2.4	The application of Deep Learning	11
2.5	Fundamentals of Reinforcement Learning	17
2.6	Stock trading	20
2.7	Deep Reinforcement Learning for Stock Trading	22
2.8	Dynamic Allocation	29
2.9	Economic Analysis	30
Chapter 3 Methodology And Experiment		33
3.1	Overview of Model Architecture	33
3.2	Data Preparation	34
3.3	Data Preprocessing	37
3.4	Hyperparameter Analysis	40
3.5	Actor and Critic Network	42
3.6	Deep Q-Network (DQN) Implementation	44
3.7	Model Training	45
3.8	Testing and Evaluation	47
Chapter 4 Results		49
4.1	Proximal Policy Optimization	49
4.2	Advantage Actor-Critic	53
4.3	Deep Q-Network	57
CHAPTER 5 CONCLUSION, DISCUSSION, AND FUTURE WORK		63
5.1	Conclusion	63
5.2	Discussion	63
5.3	Future Work	64
References		67
Appendix A		73
Appendix B		89
Appendix C		93
VITA		97

LIST OF TABLES

Table		Page
4-1	The portfolio (test model) of PPO	49
4-2	The portfolio (test model) of A2C	53
4-3	The portfolio (test model) of DQN	57

LIST OF FIGURES

Figure		Page
2-1	AI technique for Stock Analysis	5
2-2	The ARIMA model	7
2-3	Generative Adversarial Networks	10
2-4	LSTM model for price prediction	12
2-5	Random Forest	13
2-6	The Support Vector Machine	14
2-7	The AdaBoost model with decision trees for price prediction	15
2-8	Illustration of general reinforcement learning	17
2-9	DQN with a replay buffer and a target network	19
2-10	The Actor-Critic Process	25
2-11	Illustration of general actor-critic models	26
2-12	Illustration of the PPO model	27
3-1	Overview of Model Architecture	34
3-2	Illustration data after Feature Engineering	38
3-3	Illustration data after Normalization	39
3-4	Visualizing data split of Advanced Info Service PCL (AIS)	40
4-1	The price trend of the ADVANC and INTUCH of PPO	50
4-2	The price trend of the PTTEP and BDMS of PPO	51
4-3	The price trend of the MINT and CPN of PPO	51
4-4	The price trend of the AOT and TISCO of PPO	52
4-5	The price trend of the SCC and IVL of PPO	53
4-6	The price trend of the INTUCH and PTTEP of A2C	55
4-7	The price trend of the BDMS and CPN of A2C	55
4-8	The price trend of the IVL of A2C	56
4-9	The price trend of ADVANC, AOT, TISCO, and SCC of A2C	57
4-10	The price trend of the ADVANC and INTOUCH of DQN	59
4-11	The price trend of the PTTEP and BDMS of DQN	59
4-12	The price trend of the MINT and CPN of DQN	60
4-13	The price trend of the AOT and TISCO of DQN	61
4-14	The price trend of the SCC and IVL of DQN	61

LIST OF ABBREVIATIONS

A2C	Advantage Actor-Critic
AdaBoost	Adaptive Boosting
AI	Artificial intelligence
ANNs	Artificial Neural Networks
ARIMA	Autoregressive Integrated Moving Average
BH	Buy-and-Hold
CPU	Central processing unit
CNN	Convolutional Neural Networks
DDPG	Deep Deterministic Policy Gradient
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
EMH	Efficient Market Hypothesis
EVA	Economic Value Added
ETFs	Exchange-traded funds
EWMA	Exponentially weighted moving average
GAE	Generalized Advantage Estimation
GANs	Generative Adversarial Networks
GARCH	Generalized Autoregressive Conditional Heteroskedasticity
GRU	Gated Recurrent Unit
GPUs	Graphics Processing Units
HAN	hybrid attention network
LSTM	Long-Short Term Memory
ML	Machine Learning
MLP	Multilayer Perceptrons
MSE	Mean squared error
PPO	Proximal Policy Optimization
RBF	Radial basis function
RL	Reinforcement learning
RMSE	Root mean squared error
RMSprop	Root Mean Squared Propagation
RNN	Recurrent Neural Networks
ROI	Return on Investment
ROIC	Return on Invested Capital
RRL	Recurrent Reinforcement Learning
SET	Stock Exchange of Thailand
SVM	Support Vector Machines
TD	Temporal Difference
TDQN	Trading Deep Q-Network
TPUs	Tensor Processing Units
VAEs	variational autoencoders
VWAP	Volume Weighted Average Price
XAI	Explainable AI

CHAPTER 1
INTRODUCTION

1.1 Toward Reinforcement Learning in Portfolio

In recent years, the financial industry has shifted toward algorithmic and data-driven trading strategies, driven by the vast expansion of market data and the increasing power of computational technologies. While effective in some contexts (Hendershott et al., 2011), traditional trading strategies often need help adapting to rapid market dynamics changes. This thesis investigates the application of Deep Reinforcement Learning (DRL) for dynamic portfolio management, specifically targeting the Stock Exchange of Thailand (SET). By exploring advanced DRL algorithms, this research aims to design a robust and adaptive trading model capable of balancing returns with risk.

Rule-based and machine learning-based approaches are the two primary groups into which trading strategies are usually separated. Using traditional trading strategies or statistical models that may concentrate on price momentum, liquidity, or market sentiment, rule-based trading is predicated on predetermined rules created by human specialists. On the other hand, trading that relies on machine learning trains models with past data so that they can make trades on their own. Prominent methods in this field include Recurrent Reinforcement Learning (RRL) and Trading Deep Q-Network (TDQN) (Théate & Ernst, 2021).

By gleaning patterns and insights from past data, deep learning-driven algorithmic trading techniques seek to maximize investment decisions, helping investors make better judgments and reap higher returns. The use of deep reinforcement learning algorithms in algorithmic trading has been made possible by developments in high-performance computing and deep learning algorithms (Liu et al., 2020), which have produced automated trading techniques, particularly for situations involving a single asset. Through interactions with dynamic surroundings, these algorithms learn and hone their methods to identify profitable patterns and create superior long-term solutions. The Q function represents the expected cumulative reward an agent can receive in deep reinforcement learning when it performs a specific action in a particular state and then proceeds to follow a particular policy. Q networks are designed to learn the optimal action-selection strategy by estimating the Q value (expected cumulative reward) for various state-action pairs. The design of Q-Networks incorporating neural networks like Multilayer Perceptrons (MLP) (Li et al., 2019), Recurrent Neural Networks (RNN), and Convolutional Neural Networks (CNN) (Dang, 2020) significantly impacts the model's performance. Additionally, algorithms such as Proximal Policy Optimization (PPO) offer more excellent stability and efficiency compared to traditional DQN, making them particularly suitable for complex trading environments. Q networks are made to estimate the Q value, or expected cumulative reward, for different state-action combinations to learn the best action-selection strategy. The architecture of Q-Networks, which incorporates neural networks, dramatically influences the model's performance. Furthermore, compared to classic

DQN, algorithms like Proximal Policy Optimization (PPO) are more stable and efficient, which makes them especially useful in complicated trading situations.

A custom trading environment, StockTradingEnv, is developed using OpenAI Gym to simulate the dynamics of the Stock Exchange of Thailand (SET). The environment defines a discrete action space representing buy and sell decisions and an observation space comprising key financial indicators such as open, high, low, close prices, volume, and Volume-Weighted Average Price (VWAP). The reward function is meticulously crafted to reflect portfolio performance, considering net profit, volatility, and trade ratios. This tailored environment allows the Deep Reinforcement Learning (DRL) agent to interact realistically with market data, facilitating the development of effective trading strategies.

Existing financial time series feature extraction methods, like linear models and traditional neural networks, often need help capturing complex and nonlinear relationships in the data, leading to poor prediction performance due to overfitting or underfitting. State-of-the-art time series feature extraction networks have been designed to address these challenges using advanced techniques and perspectives.

However, existing trading strategies often rely on single models or algorithms that may perform well under certain market conditions but fail when conditions change or unexpected events occur (Cheng et al., 2021; Nan et al., 2022; Taghian et al., 2022). This can lead to suboptimal trading decisions and lost profits. It is crucial to develop adaptive trading strategies that can be learned from and adjusted to different market conditions to tackle this issue.

Financial markets are complex systems characterized by high volatility and non-linearity, making traditional stock trading strategies often insufficient. Machine learning, particularly reinforcement learning (RL), has shown promise in addressing these challenges by enabling algorithms to learn and adapt to dynamic market conditions. Among the RL approaches, Advantage Actor-Critic (A2C) has emerged as a powerful method that combines the benefits of policy-based and value-based strategies. A2C employs an actor, which proposes actions, and a critic, which evaluates them, facilitating more stable and efficient learning than standalone methods like DQN. The actor-network in A2C is responsible for selecting actions based on the current policy, while the critic network estimates the value function, providing feedback that helps refine the policy. This dual-network architecture allows A2C to reduce the variance in policy updates, leading to more reliable convergence during training.

Despite its advantages, A2C can face limitations in terms of sample efficiency and scalability, especially in highly volatile environments such as financial markets where rapid and significant changes are commonplace. These challenges necessitate the exploration of more advanced RL algorithms that can offer enhanced stability and efficiency. Proximal Policy Optimization, a more recent reinforcement learning algorithm, addresses these limitations by introducing a clipped surrogate objective function that prevents large, destabilizing updates to the policy. This clipping mechanism ensures that policy updates remain within a trust region, maintaining a balance between exploration and exploitation while safeguarding against drastic changes that could undermine the learning process.

PPO has been successfully applied to various domains, including gaming and robotics, demonstrating its robustness and adaptability. Its improved stability and efficiency over traditional methods like DQN and even A2C make it well-suited for complex trading environments where maintaining a reliable and adaptable strategy is crucial. This paper investigates the efficacy of PPO in stock portfolio optimization on the Stock Exchange of Thailand. By leveraging historical stock data, the author develops a PPO agent to make trading decisions, optimizing the portfolio's return. This paper details the model architecture, training process, data preprocessing techniques, and performance evaluation of the PPO agent, providing a comprehensive case on its application in financial markets.

1.2 Objective

The main objectives of this study are described below:

1.2.1 To develop and refine deep reinforcement learning algorithms customized for the unique characteristics and dynamics of the Stock Exchange of Thailand.

1.2.2 To optimize trading strategies for portfolio construction on maximizing returns.

1.2.3 To evaluate the developed DRL models and trading strategies through extensive backtesting on historical data.

1.3 Scope

A portfolio of 10 stocks from the SET is chosen based on their market capitalization, liquidity, and sector diversity to ensure representative coverage of the market.

1.4 Outline of Thesis

Chapter 2 provides a comprehensive review of existing research in stock prediction and algorithmic trading, focusing on the application of DRL. The chapter discusses traditional stock prediction models like Autoregressive Integrated Moving Average (ARIMA) and Generalized Autoregressive Conditional Heteroskedasticity (GARCH), highlighting their limitations in dynamic market conditions. It then explores how Machine Learning and Deep Learning techniques, such as Support Vector Machine (SVM), Random Forests, and Long-Short Term Memory (LSTM) networks, have enhanced prediction accuracy but still face challenges like overfitting.

The fundamentals of Reinforcement Learning are introduced, explaining how RL agents interact with their environment to make trading decisions. Advanced DRL algorithms like DQN and PPO are then discussed, showcasing their superiority in adapting to volatile markets. The chapter also delves into algorithmic trading systems, outlining their components and the importance of performance metrics like Return on Investment (ROI) and Sharpe Ratio.

Through this extensive review, the chapter identifies significant research gaps, particularly the limited application of DRL in emerging markets like Thailand. This sets the stage for the subsequent chapters, where your research aims to develop a DRL-based automated trading system tailored to the unique challenges of the Stock Exchange of Thailand, thereby contributing to both academic knowledge and practical trading strategies.

Chapter 3 is dedicated to the development and practical application of the DRL model for the automation of portfolio optimization in the Stock Exchange of Thailand.

The DRL agent's basic architecture and state space's proper setting is defined at the start of this chapter. Furthermore, the traded volumes of stocks also determine the pricing of them. The trader can have other insights into the stock price based on the stocks. This is achieved using technical indicators for the targeted stocks or the VWAP. Thus, the development of the DRL agent is tackled with the construction of the reward function’s part that seeks to suggest the yields. The straightforward approach to how the Proximal Policy Optimization (PPO) method is implemented to practice reinforcement learning and how the agent is then trained to make appropriate trading decisions are covered in this part. The part also includes parts on model architecture, training process, and additional strategies to improve its performance, such as hyperparameter tuning.

Chapter 4 presents the experimental results and analysis of the developed DRL trading strategies, specifically Proximal Policy Optimization (PPO), Advantage Actor-Critic (A2C), and Deep Q-Network (DQN), evaluated using historical data from the SET. The chapter begins by outlining the experimental setup and objectives, including the testing methodology with a separate dataset not seen during training. The evaluation metrics used ROI, Sharpe Ratio, Maximum Drawdown, and Calmar Ratio to assess profitability and risk management capabilities. It then delves into the performance evaluation of each agent, providing detailed portfolio results and analyses of their trading behaviors, supported by tables summarizing key performance metrics and figures illustrating the agents' buy and sell signals alongside stock price movements. The comparative analysis highlights the strengths and weaknesses of each agent in terms of profitability and risk management, discussing their consistency across different stocks and market conditions and identifying areas where strategies could be refined for improved performance. The chapter concludes with a summary of the key findings, assessing the agents' effectiveness in trading within the SET market, reflecting on the implications for real-world trading strategies, and suggesting future work to enhance the agents' performance and consistency across diverse market environments.

Chapter 5 summarizes the key findings of the research, emphasizing the contributions of the DRL-based portfolio optimization framework. The section revisits the research objectives, demonstrating how they were met through the development of the customized DRL algorithm and its application in the Stock Exchange of Thailand. The chapter also highlights the limitations of the current approach, suggesting areas for further improvement, such as incorporating fundamental analysis and expanding the model to other market environments. Finally, the potential for real-world application of DRL strategies in portfolio management is discussed, along with future research directions to refine and enhance the methodology.

To design a robust DRL-based model for dynamic portfolio management, it is essential first to understand the current landscape of algorithmic trading methodologies and their limitations. Chapter 2 provides a comprehensive review of relevant literature, from traditional statistical models to recent advancements in machine learning and DRL, setting the stage for this research's novel contributions.

CHAPTER 2
RELATED WORKS

Building on the introduction to DRL in financial markets, in Chapter 2, we explore the evolution of algorithmic trading, beginning with traditional stock prediction models and advancing deep reinforcement learning (DRL) applications, starting with foundational stock prediction models and extending through machine learning and DRL techniques, this chapter traces the evolution of automated trading systems (SET, 2024). Section 2.1 introduces AI techniques in stock analysis, setting the foundation for DRL's adaptability in complex environments. Section 2.2 reviews econometric models, such as ARIMA, followed by deep learning methods like CNN and LSTM in Section 2.3. Section 2.4 presents reinforcement learning fundamentals, while Section 2.5 examines standard algorithmic trading strategies. We then discuss DRL's role in automating stock trading decisions in Section 2.6, followed by an analysis of advanced DRL algorithms in Section 2.7. Lastly, Section 2.8 addresses challenges and future directions, highlighting DRL's potential for further innovation in financial markets.

2.1 Artificial Intelligence Methodologies in Stock Market Analysis

The integration of artificial intelligence methodologies illustrated in Figure 2-1, with particular emphasis on Deep Reinforcement Learning (DRL) agents, represents a paradigm shift in quantitative trading systems. DRL frameworks demonstrate superior adaptability in navigating the inherent complexities of financial markets, characterized by non-stationary distributions and multifaceted dependencies (Li et al., 2020; Liu et al., 2022). Empirical evidence indicates that DRL-based trading strategies consistently achieve higher Sharpe ratios and cumulative returns compared to conventional portfolio allocation methods, with documented annualized returns of 22.24% against traditional benchmarks (Li et al., 2020). This performance differential underscores the efficacy of AI-driven approaches in synthesizing complex market signals and executing dynamic trading decisions across diverse market regimes.

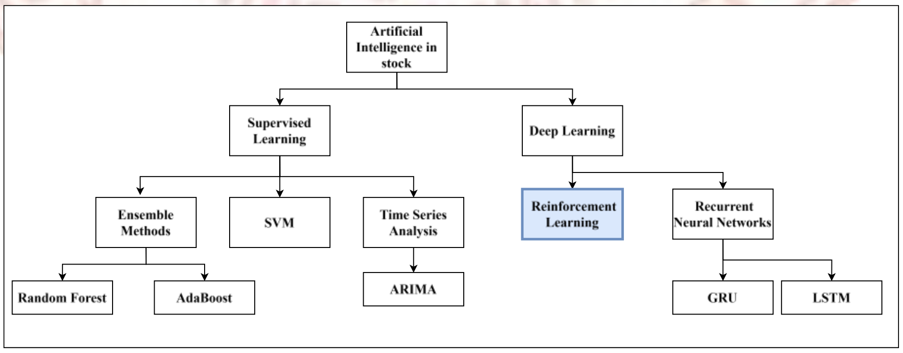


FIGURE 2-1 AI technique for Stock Analysis

The hierarchical framework presented illustrates the sophisticated integration of artificial intelligence methodologies in stock market analysis, bifurcating into

Supervised Learning and Deep Learning paradigms. This taxonomic organization demonstrates the systematic application of machine learning algorithms for financial market prediction and trading optimization.

2.1.1 Supervised Learning Framework

2.1.1.1 Ensemble Learning Architectures leverage the power of multiple learning algorithms to enhance predictive accuracy. The Random Forest algorithm implements a bootstrap aggregating approach, constructing multiple decision trees to generate robust predictions for stock movement patterns. Complementing this, AdaBoost employs an iterative boosting mechanism, systematically adjusting weights on misclassified instances to optimize predictive performance.

2.1.1.2 Support Vector Machine Implementation employs kernel-based optimization to construct optimal hyperplanes in high-dimensional feature spaces. This mathematical foundation enables precise classification of stock price trajectories through nonlinear mapping of market indicators.

2.1.2 Time Series Analytics

The ARIMA methodology provides a rigorous statistical framework for analyzing temporal dependencies in stock price movements. This approach incorporates autoregressive components and moving averages to model complex market dynamics and seasonal patterns.

2.1.3 Deep Learning Architecture

2.1.3.1 Reinforcement Learning Paradigm implements a Markov Decision Process to optimize trading strategies through interaction with market environments. This approach enables the development of adaptive trading agents that optimize decision policies based on historical market states and reward signals.

2.1.3.2 Recurrent Neural Network Implementation The architecture incorporates two sophisticated RNN variants.

a) GRU (Gated Recurrent Unit): Implements an efficient gating mechanism for processing sequential market data.

b) LSTM (Long Short-Term Memory): Employs advanced memory cells to capture long-term dependencies in financial time series.

2.1.4 Methodological Integration

This comprehensive framework facilitates the synthesis of multiple analytical approaches, enabling researchers and practitioners to implement hybrid strategies that leverage the complementary strengths of different algorithmic paradigms. The integration of supervised and deep learning methodologies provides a robust foundation for developing sophisticated trading systems that can adapt to dynamic market conditions.

The systematic organization of these methodologies demonstrates the evolution from traditional statistical approaches to advanced neural architectures, reflecting the increasing sophistication of quantitative finance and algorithmic trading strategies.

2.2 Stocks prediction

stock price movements based on historical data, market conditions, and other external factors. The ability to predict stock prices accurately is crucial for investors, as it directly influences buy, sell, hold decisions, ultimately determining the profitability of a portfolio. Over the years, numerous methods have been developed for stock prediction, ranging from traditional statistical models to advanced machine learning and reinforcement learning algorithms. Each approach has its advantages and limitations, with varying degrees of success in different market conditions.

Prediction is a key area of research in the financial domain, aimed at forecasting future stock price movements based on historical data, market conditions, and other external factors. The ability to predict stock prices accurately is crucial for investors, as it directly influences buy, sell, hold decisions, ultimately determining the profitability of a portfolio. Over the years, numerous methods have been developed for stock prediction, ranging from traditional statistical models to advanced machine learning and reinforcement learning algorithms. Each approach has its advantages and limitations, with varying degrees of success in different market conditions.

Similarly to frequent global difficulties of the economy with the depreciation of the national currency and the existence of the non-performing stock market in many countries around the world, the rapid change of the price of assets is often named as one of the problems (Nassirtoussi et al., 2014). The trends obtained for the asset price prediction were quite clearly visible in the initial stages, where more traditional econometric models, such as the use of algorithms, were first noticed, and ARIMA and GARCH were given the names, respectively (Kara et al., 2011). The baseline models of ARIMA are shown in Figure 2-1. By the side of the corresponding patterns, these were most inappropriately picked by the people, as the weaker analysis was innovative and was the focal point of the attack of the viruses to the hosts. Yet with the worsening of the malaise and the necessity of a strong virus as opposed to a weak one, immunity was hit, and the old virus was zapped due to the changes which were in the air at that time. The key point is that trends in the market that relate to asset prices are usually transient and can either come about in times of short durations with the coming and going of seasons, which may be periodic or abrupt. Ultimately, they can also remain for long periods and change with time, as shown in the biannual studies undertaken about stock prices in Vietnam in the recent period (Yadav et al., 2020).

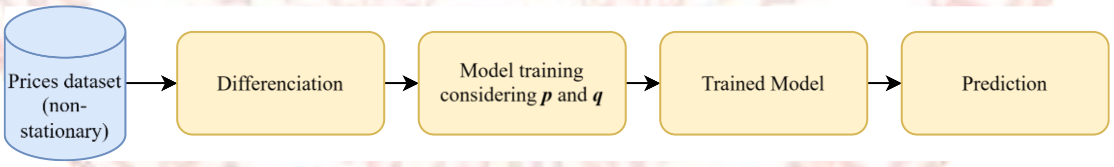

Prices dataset (non-stationary)
Differenciation
Model training considering *p* and *q*
Trained Model
Prediction

FIGURE 2-2 The ARIMA model

The first one to postulate the Efficient Market Hypothesis (EMH) was that person in 1965, using the theory of a short financial market price change pattern. But for initiating this theory at the beginning of the project, he uses the random walk model to illustrate that the change of the asset price will be random, which results in a different model and would be unpredictable. The fact that the theory stuck in the life of the discussions promoted soon after different investigations as a continuation with the outcomes being that the hypothesis is valid only if there is efficiency; thus, the information produced will be asymmetric. The ongoing debate was one in which many studies found no strong EMH in many places worldwide, e.g., in Southeast Asian countries that never had EMO (Li et al., 2020). An article mapping the effect of the news on stock prices was published, which concluded that news could whipsaw stock prices, e.g., a stock group may be affected if they see some common stock. A joint stock with a similar buying and selling price on the trading day will be more likely to be trained in short-term investments during its term, in this case, the first week in Hong Kong. In the short run, a correlational market can be very efficient because of independent processes going on if trading occurs where a slight chance is for the market to be out of balance. A news standpoint, at least, could only be seen by the market in sources like news outlets and social media.

More than ever, the use of ML and DL has strengthened the stock prediction area. They have realized this by detecting non-linear and complex data. During 2010, stocks were commonly analyzed through SVM, Random Forest, and Artificial Neural Networks (ANNs) algorithms since they can discover stock patterns through the processing of massive data (Jordan & Mitchell, 2015; Nelson et al., 2017). However, they also need to work on facing problems of overfitting and real-time decision-making. Due to this, scientists in this field have been thinking about newer examination methods.

The time after 2010 was the age when many scientists chose to delve into the study of RL and DRL, which would make trading systems adaptable to the unstable market. These came in the form of a model being fed with its learning experiences with the market over time, which resulted in the model being able to adapt its strategies through past applications and feedback from the market environment. Traditional models that wholly depended on historical data from a single source were the opposite of these recent RL and DRL models. These new models could fine-tune decision-making techniques by using data from real-time markets; hence, they could be more precise and profitable in developed and emerging markets (Hu et al., 2018).

Until 2020, the top solutions were the types of integrated learning that had advanced to sentiment analysis and reinforcement learning. Research (Bollen et al., 2011) on finding the most suitable kind of cancer therapy for patients by considering the sentiments received from several datasets, such as company reports, news headlines, and text data mining, together with social media, clearly highlights the exponential role of bridging ML/DL algorithms with various data sources, which led technology to catch up with asynchronous data reporting and social media.

A year later, the research tried to integrate these models into the decision-making process by examining the processing power of the procedure and the model scale. A mix of sentiment analysis, technical indicators, and learning models keeps ongoing research. At the same time, the focus is increasingly on the application of new sources of data and technology (Weng et al., 2018).

2.3 Deep Learning

Deep Learning is an advanced subset of machine learning that utilizes artificial neural networks with multiple layers to model and interpret intricate patterns within large and complex datasets (LeCun et al., 2015). Unlike traditional machine learning algorithms, which often rely on manual feature engineering, deep learning autonomously discovers hierarchical data representations. This capability facilitates processing high-dimensional and unstructured information such as images, audio, and text, enabling DL models to excel in tasks that require understanding complex data structures and relationships.

2.3.1 Foundations of Deep Learning

At the core of deep learning are ANNs, which are computational models inspired by the biological neural networks of the human brain. An ANN consists of layers of interconnected neurons, each performing computations through weighted inputs and activation functions. The architecture typically includes an input layer that receives the initial data, multiple hidden layers that perform feature transformations and abstractions, and an output layer that produces the final prediction or classification. Each neuron in a layer applies a weighted sum of its inputs followed by a non-linear activation function, such as ReLU (Rectified Linear Unit), sigmoid, or tanh, introducing non-linearity into the model and enabling the network to capture complex patterns.

2.3.2 Key Architectures in Deep Learning

Deep learning encompasses a variety of neural network architectures, each tailored to specific types of data and tasks. CNNs are primarily used for spatial data analysis, such as image and video recognition. They utilize convolutional layers with filters that automatically detect spatial hierarchies and features like edges, textures, and shapes, making them highly effective in capturing local patterns while reducing the number of parameters through weight sharing and pooling operations.

Recurrent Neural Networks, including variants like LSTM and Gated Recurrent Unit (GRU) networks, are designed for sequential data processing, such as time series analysis, natural language processing, and speech recognition. RNNs feature recurrent connections that allow information to persist across time steps, enabling the network to maintain context and learn long-term dependencies. This is crucial for tasks involving temporal dynamics (Hochreiter & Schmidhuber, 1997).

Autoencoders are primarily used for unsupervised learning tasks such as dimensionality reduction, feature learning, and anomaly detection. They consist of an encoder that compresses the input into a latent-space representation and a decoder that reconstructs the input from it. Variants like denoising autoencoders, variational autoencoders (VAEs), and sparse autoencoders enhance specific aspects of the learning process, improving robustness and efficiency in feature extraction.

By offering a fresh method for producing synthetic data, Generative Adversarial Networks, or GANs show in Figure 2-3, have completely transformed the field of generative modeling. The basic idea is that two neural networks, the discriminator and the generator, are always competing. Essentially "fooling" the discriminator, the generator aims to generate data that closely mimics real-world data. Conversely, the discriminator assesses both produced and real data, becoming increasingly accurate in differentiating between the two. Over time, extremely realistic synthetic data is produced by the generator because of this adversarial training, which forces iterative output improvement from the generator.

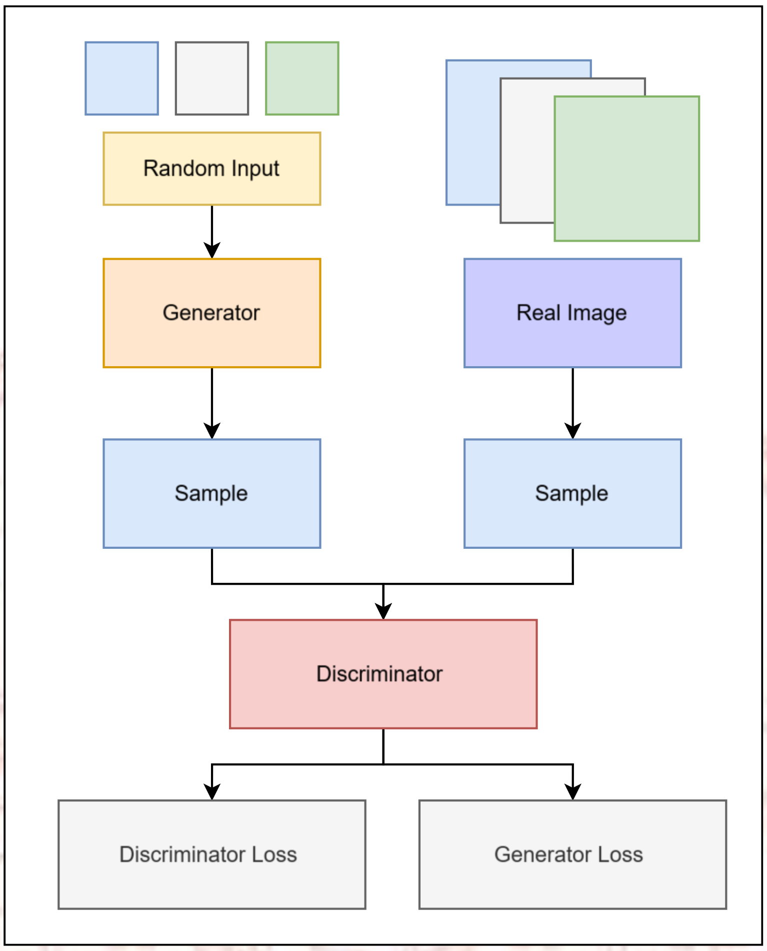


FIGURE 2-3 Generative Adversarial Networks

GANs have an impact on many different applications. GANs can produce lifelike images of scenes, objects, and even human faces that are not real through image synthesis. For sectors like virtual reality and entertainment, this capability is priceless. GANs aid in the expansion and diversity of datasets for data augmentation, which is especially helpful when data is complex to come by or prohibitively expensive to acquire. By applying the stylistic components of one image to another, style transfer allows GANs to alter images. For example, it can be used to change a photograph into the look of a well-known artwork. In machine learning and artificial intelligence research, GANs have opened new vistas, making creating synthetic datasets almost identical to real ones easier.

2.3.3 Training Deep Neural Networks

Training deep neural networks involves systematically adjusting the network's parameters, including weights and biases, to minimize a loss function quantifying the difference between the predicted outputs and targets. This process begins with forward propagation, passing input data through each network layer to generate predictions based on the current parameter values. The loss function then calculates the prediction error, providing a metric for optimization. To reduce this error, backward propagation is performed, where gradients of the loss concerning each parameter are computed using the chain rule, enabling precise adjustments to the weights and biases.

Optimization algorithms such as Stochastic Gradient Descent, Adam, and Root Mean Squared Propagation (RMSprop) are employed to update the network's parameters efficiently, ensuring a balance between the speed of convergence and the stability of the training process. Additionally, regularization techniques like dropout, L2 regularization, and batch normalization are applied to prevent overfitting and enhance the model's ability to generalize to new data by reducing complexity and improving training dynamics. This comprehensive training framework enables deep neural networks to learn complex patterns and make accurate predictions across a wide range of applications.

2.3.4 Advancements Enabling Deep Learning

The rapid advancement and widespread adoption of deep learning can be attributed to several key factors. The increase in computational power, particularly using Graphics Processing Units (GPUs) and specialized hardware like Tensor Processing Units (TPUs), has significantly accelerated the training of deep neural networks, enabling the handling of large-scale models and datasets. The availability of large datasets across various domains provides the vast amounts of labeled and unlabeled data necessary for training deep learning models effectively.

Improvements in algorithms and architectures have also played a crucial role. Innovations in neural network architectures, optimization techniques, and training methodologies have enhanced the performance and scalability of deep learning models, allowing them to achieve state-of-the-art results in numerous applications. Additionally, the development of open-source frameworks such as TensorFlow, PyTorch, and Keras has democratized access to deep learning technologies, facilitating research and application development by providing user-friendly tools and extensive libraries.

2.4 The application of Deep Learning

Machine learning and combinatorial data algorithms for learning analytics in technology spaces are all part of the scientific proof. One of the basic ideas is the interaction of algorithm diversification and combination and the use of artificial intelligence to provide a new type of learning for those where discipline is absent by testing with courses.

The rise of DL models showed a move away from these traditional methods. Research revealed that DL models such as LSTM networks convincingly outperformed ARIMA and SVM in stock trend prediction. This leap was ascribed to the DL models' ability to capture the temporal patterns of financial data and simultaneously handle the high-dimensionality properties with the complex dynamics, making them more suitable in financial data featuring vibrancy, noise, and non-linearity.

Deep learning models like LSTM are now utilized primarily in the financial industry for stock price prediction. LSTM is a RNN designed to learn data sequences over time (Hochreiter & Schmidhuber, 1997). Its characteristic of long-run dependency in the time series data allows it to be used both for financial forecasting and long-time experience with the data, which is crucial for making accurate predictions.

An LSTM network maintains a memory cell with lengthy retention times. The forget gate, input gate, and output gate are the three primary gates shown in Figure 2-4 that control the information flow in this memory cell. The forget gate determines which data from the memory cell should be erased. It generates values between 0 and 1 by applying a sigmoid function to the current input and the prior hidden state. These numbers serve as weights, indicating how easily knowledge is lost.

The input gate decides what fresh data is added to the memory cell. Additionally, a sigmoid function is used to build a filter to determine which input values need updating. Furthermore, a tanh function produces a vector of new candidate values that could be added to the state. Combining these two functions updates the cell state with relevant new information. Lastly, the output gate decides what information from the cell state is sent to the next hidden state, ultimately influencing the output. It filters the cell state through a sigmoid function and then multiplies it by the tanh of the updated cell state to produce the final output. This gating mechanism allows LSTMs to learn and remember essential patterns over long sequences, making them ideal for time-dependent tasks like stock price prediction.

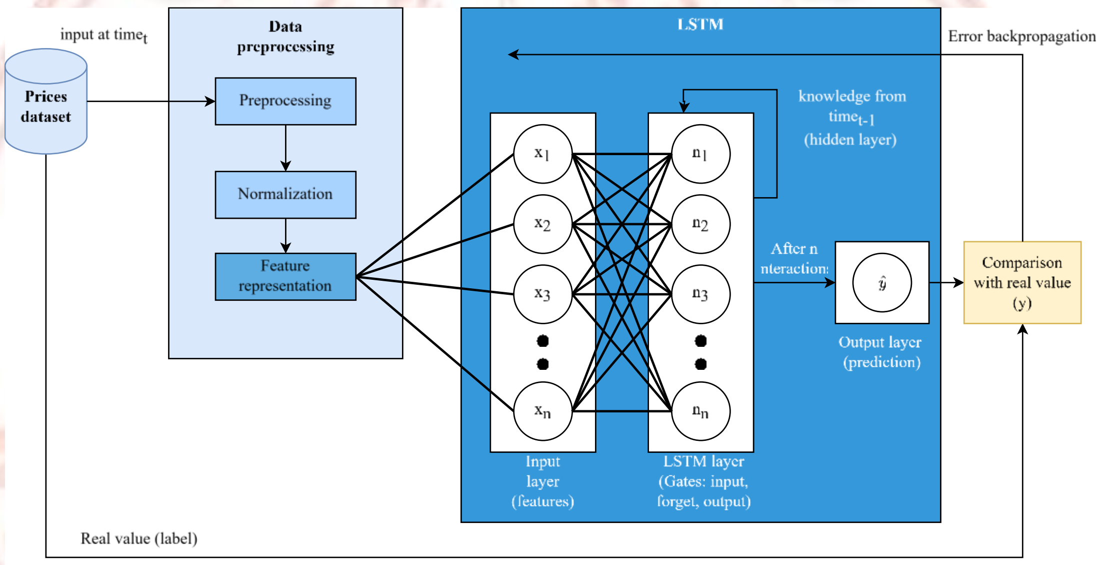


FIGURE 2-4 LSTM model for price prediction

Besides its potential for processing complex temporal data, DL provides another significant advantage. This is the ability to feature automatic extraction. Traditional ML models like SVM or decision trees require careful and predefined feature engineering, which is a process consisting of the manual selection of those features known to be relevant concerning the domain. On the other hand, DL models, especially CNNs and LSTM networks, can learn high-level abstract features directly from raw data (LeCun et al., 2015; Sülo et al., 2019). The capacity to sidestep the need for manual feature engineering has been one of the factors leading to the general acceptance of DL models in finance and other relevant domains.

Recent research has tackled using DL models in more challenging prediction tasks in financial markets. A case in point is the study (Avramelou et al., 2023), in which univariate and multivariate LSTM and CNN models were set side by side to predict the opening price of stocks listed on the Indian stock markets. Thus, the authors attested that LSTM was more accurate than CNN when applied to multivariate time series, which use several input features, such as volume and price. However, CNN models still perform reasonably, revealing the possibility of using DL architectures apart from LSTM.

Ensemble techniques have also been used in combination with prediction models to enhance the modeling accuracy. The authors of the article (Ballings et al., 2015) employed ensemble techniques to predict stock market trends, employing numerous different models like random forest, SVM, and Adaptive Boosting (AdaBoost). They discovered that ensemble models could outperform individual models to some extent since they might find different angles to the data, reducing the chances of overfitting or prejudice in predictions. In finance, ensemble methods have proved particularly useful due to their capacity to amalgamate various insights from diverse data sources due to the markets' unpredictable nature.

The Random Forest ensemble learning technique, shown in Figure 2-5, is applied to tasks including regression and classification. During training, it builds many decision trees and outputs the average prediction made by each tree. To provide variation among the trees, a random selection of characteristics and a random subset of data are used to construct each decision tree in the forest. When a model learns the training data too well and performs poorly on fresh, unseen data, it is said to be overfitting. This randomization helps to reduce overfitting. It is a powerful tool because Random Forest can handle big datasets with increased dimensionality and maintain accuracy even when a sizable amount of data is missing. The model produces more accurate and dependable predictions by averaging biases and reducing volatility by merging the forecasts of numerous trees. Because Random Forest can capture intricate interactions among many variables impacting stock values, it is conducive in financial modeling.

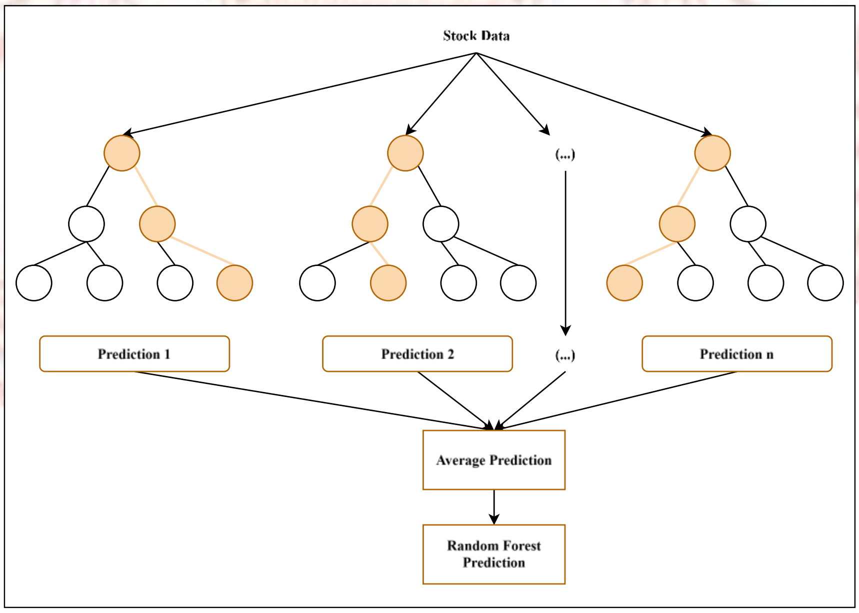


FIGURE 2-5 Random Forest

Strong supervised learning models for regression and classification applications are support vector machines. SVM's basic idea is to identify the ideal hyperplane in a feature space for dividing data points into distinct groups. The margin of the distance between the hyperplane and the closest data points from each class, or support vectors, is maximized to determine the location of this hyperplane. SVM seeks to increase the model's capacity to generalize to fresh, untested data by optimizing this margin, therefore lowering the possibility of misclassification, as shown in Figure 2-6.

Many real-world scenarios do not allow for a straight line to precisely divide data. SVM uses slack variables and soft margins to address this, enabling some data points to be incorrectly classified or fall inside the boundary. Using kernel functions to manage non-linear interactions in the data is another fundamental idea of SVM. Kernels transform the input features into a higher-dimensional space that may contain a linear separator. The linear, polynomial, sigmoid, and radial basis function (RBF) kernels are examples of common kernel functions. The computation is more efficient because of the kernel trick, which enables SVM to carry out this transformation without explicitly calculating the coordinates in the higher-dimensional space. SVM can handle a wide range of data distributions by selecting the right kernel, which makes it flexible enough to capture intricate patterns needed for jobs like stock market prediction.

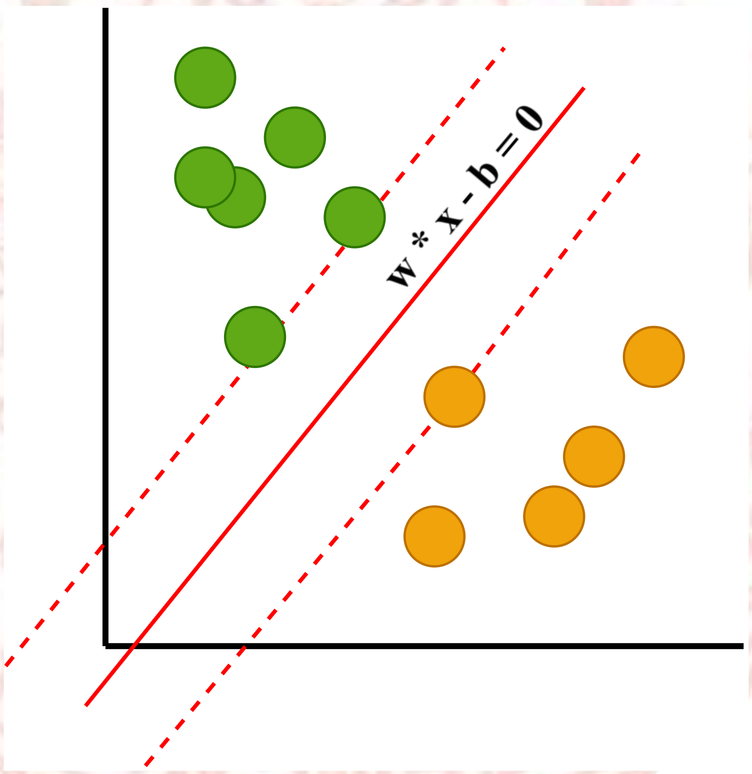


FIGURE 2-6 The Support Vector Machine

Adaptive Boosting, shown in Figure 2-7, is an ensemble learning method that builds a powerful predictive model by combining several weak learners. A model that does only marginally better than random guessing is called a weak learner. AdaBoost trains these poor learners stepwise, giving more attention to data instances that earlier learners misclassify as they progress. At the beginning, equal weights are assigned to each training data point. Each iteration ends with a weight adjustment: instances that were incorrectly classified have their weights reduced, and instances that were correctly classified have their weights increased. This adaptive weighting procedure contains the principle. AdaBoost encourages second and subsequent learners to focus on these difficult scenarios by emphasizing them more. Based on their accuracy, each weak learner adds to the final model; learners with higher accuracy significantly impact the final forecast. A robust model is produced by adding together all the weak learners, and this model frequently outperforms any single learner in terms of accuracy.

However, AdaBoost may give undue weight to misclassified occurrences that are anomalies rather than typical of the underlying data distribution, which makes it susceptible to noisy data and outliers. Because of this sensitivity, the data may need to be carefully adjusted and occasionally preprocessed to lessen the influence of outliers. AdaBoost's capacity to blend distinct weak learners aids in acquiring various market signals and patterns in financial modeling. It can enhance the accuracy of stock price predictions by utilizing several viewpoints, which makes it an invaluable instrument in the ensemble learning toolbox.

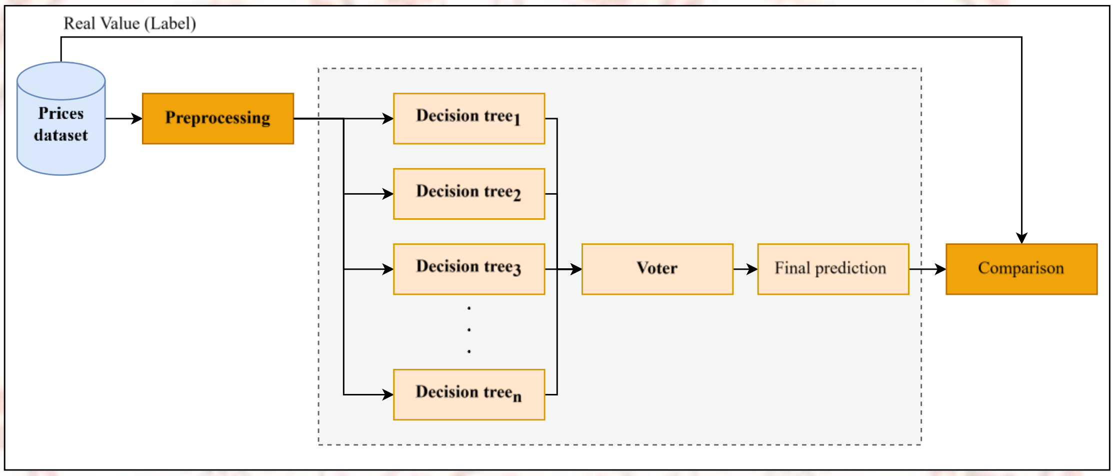


FIGURE 2-7 The AdaBoost model with decision trees for price prediction

Recent advancements in the development of deep reinforcement learning have also resulted in new prospects for financial prediction. Unlike traditional supervised learning methods fed through labeled data, RL methods are taught through interacting with their surroundings and receiving responses as rewards or penalties. DRL is an RL method resulting from merging RL with deep neural networks, making it feasible for models to acquire complicated decision strategies over time. In trading, for instance, DRL has been used to design strategies that adjust to market conditions. For example, DRL models can learn asset allocation based on historical market data and reward, leading to increased financial return.

Despite the claim of DL and DRL models, many things could still be improved. The biggest one is the availability of high-quality, marked financial data. While large datasets are necessary for training DL models, financial data often needs fixing, such as noise, missing values, and other inconsistencies. Furthermore, the temporal nature of economic data introduces autocorrelation, making model training even more difficult. Some studies have addressed these challenges by applying pre-processing techniques such as normalization and feature scaling, which help DL models converge faster and perform better (Hu & Lin, 2019). However, further research is essential to develop applicable methods for handling noisy, high-dimensional financial data.

Besides the issues related to data, there is also the problem of interpretability, which holds a significant place in applying DL models to finance. DL models, on the one hand, successfully capture the various intricate patterns in data. Still, on the other hand, they are often criticized for being called "black boxes" because their internal decision-making processes are not easily understood. This lack of interpretability may hinder, especially in finance, the need of the decision-makers to know how the predictions come, and which factors drive specific results. Some researchers (Lakshmanarao et al., 2023) point out that interpretable models are crucial in high-stakes applications like financial forecasting, where errors can lead to substantial monetary losses.

Not long ago, scholars had already commenced the solution of making DL learning more transparent and interpretable. For example, attending to the features in a dataset, which are needed mainly by attention mechanisms, is now integrated into DL models to enable them to make more understandable predictions. Further methods, such as Explainable AI (XAI), intend to explain the decision-making process of DL models in detail. These new achievements align with the actual scenario in the field because professionals should be informed about model interpretability and responsibility when dealing with it, without it being just a bonus.

Another new aspect in the field is the use of hybrid models that mix DL with traditional ML models to take advantage of the good aspects of both methods. The use of the hybrid models improves stock price prediction with the help of the DL models, as they capture complex patterns effortlessly and make it simple for users to read. For instance, a hybrid model might be used with an LSTM network to process the old stock prices and then a random forest model for final prediction. Such models have been found to have a greater success rate than those with classical statistical methods. It was observed mainly in cases where the data in question was noisy and non-linear.

Transfer learning, a technique that allows models to benefit from one task to another, has gained momentum in financial ML. This approach is especially relevant in finance, where the common features are shared between different markets or assets.

However, despite these advances, there is still much work to be done in evaluating deep learning models in financial forecasting. Many studies use loss metrics such as mean squared error (MSE) or root mean squared error (RMSE) as model performance benchmarks. These metrics have an adverse influence, as they may not fully represent the real-world implications of a model's predictions, particularly in the trading environment where transaction costs and risk are relevant. Integrating evaluation (Kumar et al., 2023) metrics to the model, such as the Sharpe ratio or maximum drawdown, accounts for a trading strategy's risk-adjusted returns. These metrics provide a more accurate picture of a model's performance in practical trading scenarios.

In summary, machine learning has come a very long way in the last ten years. This was particularly visible in the financial domain. Traditional models like SVM and AdaBoost have been thrown aside in the finance industry, and now the deck is stacked with more intricate deep learning frameworks, LSTM and CNN, that can handle financial data, which is highly dimensional and packed with noise. The emergence of deep reinforcement learning, together with colleagues, has taken financial prediction to another height, making time-variant models capable of working with shifting market conditions. Nevertheless, the model's data availability, interpretability, and accuracy are still issues to be conquered. Through research that tackles these ways, the vision of ML in finance promises to have future applications like automated trading systems, portfolio optimization, and more.

2.5 Fundamentals of Reinforcement Learning

Rather than restructuring through a decision-making process, a sequential course also enables agents to learn a wide range of decision-making processes in situations characterized by uncertainty, often financial markets. In reinforcement learning, the agent learns the optimal way to approach the environment by getting rewards and penalties with time. Thus, this method is an appropriate way of teaching trading as the market is a game against an agent, and the agent's strategy focuses on decisions of when to buy and sell or hold the stock depending on market price, volatility, and other indicators. The following stages characterize the RL model: the agent (the decision-maker), the environment (the system the agent interacts with), states (the environment representation at different periods), actions (possible movements that the agent can choose), and rewards (feedback from the environment). The agent's problem is finding a policy π(a|s) that best maps states to actions and such that the expected average steps are more than the most that can be got. This process is formalized via Markov Decision Processes (MDPs) Illustrate in Figure 2-8, and these are characterized by states S, actions A, transition probabilities P(s′|s, a), and rewards R(s, a) (Puterman, 2014).

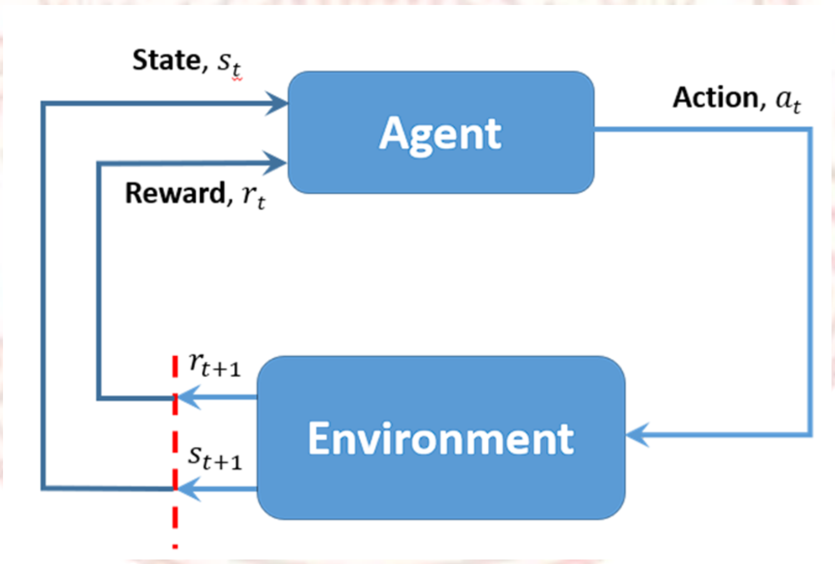


FIGURE 2-8 Illustration of general reinforcement learning

Rewards derive from value functions in the form of function estimates that embody the best outcomes of a given state or decision. The state-value function V(s) stands for the expected return that starts from state s, while the action-value function Q(s,a) signifies the probable return for taking action a in state s. The Bellman equation provides a recursive decomposition of these values:

$$V(s) = \sum_a \pi(a|s) \sum_s{}' P(s'|s,a)[R(s,a,s') + \gamma V(s')] \qquad (2\text{-}1)$$

where γ is the discount factor that determines how much the agent values future rewards. The Bellman equation lays the foundation for many RL algorithms by breaking down complex, long-term decisions into simpler subproblems.

One of the essential methods for updating value estimates in RL is Temporal Difference (TD) learning, which updates value functions based on the difference between successive estimates of future rewards. This method can be formalized as:

$$V(s) \leftarrow V(s) + \infty[r + \gamma V(s') - V(s)] \tag{2-2}$$

where α represents the learning rate. TD learning is the basis for more advanced algorithms such as Q-learning, an off-policy algorithm that directly learns the optimal action-value function Q(s, a)* (Watkins & Dayan, 1992). The Q-learning update rule is given by:

$$Q(s,a) \leftarrow Q(s,a) + \infty[r + \gamma max_{a'} Q(s',a') - Q(s,a)] \tag{2-3}$$

Building upon the foundational principles of Q-Learning, DQN, illustrated in Figure 2-9, leverages the power of deep neural networks to effectively approximate the optimal action-value function Q∗(s, a) in environments with vast and complex state spaces, such as financial markets. Unlike traditional Q-Learning, which relies on a discrete table to store Q-values for each state-action pair, DQN employs a convolutional neural network (or other suitable architectures) to generalize learning across similar states, significantly reducing memory requirements and enhancing scalability. This neural approximation allows the agent to discern intricate patterns and relationships within high-dimensional financial data, such as price movements, trading volumes, and various technical indicators, essential for making informed trading decisions. Additionally, DQN incorporates experience replay, a mechanism that stores past experiences in a replay buffer and samples mini batches of these experiences randomly during training. This approach mitigates the issue of correlated data samples, leading to more stable and efficient learning by breaking the temporal dependencies inherent in sequential trading data. Furthermore, the introduction of target networks, a separate network with periodically updated weights used to compute target Q-values, helps prevent harmful feedback loops and reduces the risk of divergence during the training process. These innovations collectively enable DQN to learn robust trading strategies that can adapt to financial markets' stochastic and non-stationary nature. In practical applications, DQN has demonstrated its capability to outperform traditional trading algorithms by continuously refining its policy through interactions with the market environment, optimizing buying, selling, and holding actions to maximize cumulative returns while managing risk. Despite its strengths, implementing DQN in financial trading must carefully address challenges such as ensuring sufficient exploration, avoiding overfitting historical data, and maintaining computational efficiency to operate in real-time trading scenarios. Ongoing research continues to enhance DQN's efficacy in trading by integrating advanced techniques like Double DQN, which further reduces overestimation bias, and Dueling DQN, which separately estimates state values and advantages, thereby improving the agent's ability to prioritize beneficial actions under varying market conditions.

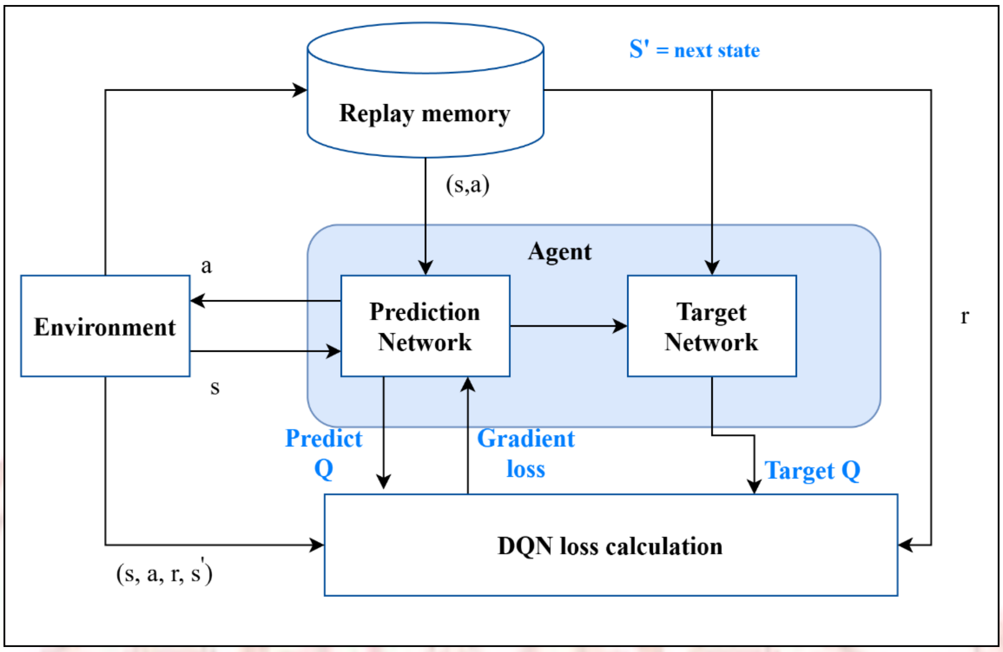


FIGURE 2-9 DQN with a replay buffer and a target network

Deep neural networks handle large and complex state spaces. They use experience replay, a technique where past transitions (states, actions, rewards, and next states) are stored and randomly sampled to train the network. This reduces the correlation between consecutive samples, stabilizing the learning process. Additionally, target networks are employed in DQN to provide more consistent Q-value targets by updating the network's parameters at a slower rate (Mnih et al., 2015).

Another crucial aspect of RL is the exploration vs exploitation dilemma. Essentially, the agent must balance exploration (experimenting with new actions to discover rewarding strategies further) and exploitation (capitalizing on the knowledge of actions that previously resulted in high rewards). The trade-off situation is particularly relevant in stock trading, where market conditions vary. Overutilization may lead the agent to a missed opportunity to explore new concepts or strategies, which could have brought a more significant increase in the portfolio. Algorithms built similarly to reinforcement learning, like scenarios to exploit the error found within the datasets to learn and make better estimates. For example, the agent sometimes selects a random action (exploration) to leave suboptimal solutions (Serrano, 2022).

Among all the different trading reinforcement learning technologies, DQN is the most basic one, and researchers have come up with additional technologies relying on reinforcement learning. The PPO has been considered to equalize the risk of moving away from the optimal policy in the update. A2C is the process through which the system trains itself to make the most effective decisions by allowing it to select from numerous alternatives while remembering the solution to the sample problem. The input avoids the necessity of a graduating student lest it may take much time to provide instruction for the output. Then, the associated teacher may have many other procedures to do so; that is the issue.

Despite being highly of the possibilities, RL has found itself in many difficulties, mostly encountered when employed in applications like automated trading. An important among these is overfitting, in which the model may perform exceptionally well based on historical data but cannot be generalized based on unseen market conditions (Moody & Saffell, 2001). Consistency issues within the training phase are additional difficulties that the environment probably confronts; a slight change in market conditions can lead to the agent’s performance fluctuating from good to bad. Moreover, RL often needs long training periods and computational resources, especially when the environment has a sparse reward and thus is not fully observable. An example of that can be found in a stock trading environment where the ultimate reward comes after only one of the possible decisions, in which the agent must try and win the most profits.

Recent advances in RL, including DRL, have addressed some of these obstacles through deep learning models designed to approximate the value function and policy more efficiently. DRL deployed in algorithmic trading has exhibited a remarkable performance, allowing computerized agents to regulate their behaviors in volatile market conditions, optimize trading operations, and allocate resources more efficiently. The most classic one is AlphaGo, which Google DeepMind created. DRL made a giant leap in complex decision-making environments, which resembled one of the cases encountered in the financial markets (Silver et al., 2016).

Rather than simply using neural networks, experience replay, and other innovations, RL-based trading systems have moved on to the stage where they meet the requirements of solving quantitative issues, such as managing vast amounts of financial data, making real-time decisions, and detecting systematic errors. These progressive developments are being rolled out to the edge of technological and economic possibilities in the trading industry, making RL a key technology for the years ahead.

2.6 Stock trading

Their number one concern has always been to predict future returns and quantify the risks of their investment strategies, not the accuracy or validity of the predicted prices or trends. However, these concerns have been broader risk management over time, with returns management alongside the process of moving to the center. Research studies in the last year are about how the current financial world is very complex and, therefore, is not a matter of trading statistics alone, issued warnings, but risks also (Lei et al., 2020). The primary and proficient trading strategy evaluation tools include annualized and cumulative returns, the Sharpe ratio, annual volatility, and maximum drawdown.

Annualized returns are gains or income from investments over one year, while cumulative returns are the total return acquired during a specific period. These two are critical factors in finding the roots of the deficiency of a particular strategy. However, contemplating the possible risks and the fact that the strategy is not winnable is also extremely important. The Sharpe ratio is a mathematical measure that reflects the relationship between risk and reward. It is computed by subtracting the risk-free rate from the annual returns and dividing the result by the annual volatility. This means that a higher Sharpe ratio is the reward for a lower trade risk.

Annual volatility, calculated as the standard deviation of the portfolio returns over a year, is the third risk measure. Alongside volatility, measured by the R-squared of the linear fit of cumulative log returns, the two indicators account for risks and performance. Drawdown, which indicates the total percentage loss before partial recovery, and maximum drawdown, which points to the most severe negative impact achieved within a given time frame, are two tools the investor uses to evaluate the riskiness of the investment strategy.

Algorithmic trading systems are generally based on a plan that covers the following phases: pre-trade analysis, signal generation, trade execution, post-trade analysis, risk management, and asset. In the last period, the trend has increasingly leaned towards involving artificial intelligence (AI) tools in the system, including ML, DL, and RL models. Automating tasks like feature extraction, price prediction, and trade execution is an example of the technology.

The "Buy-and-Hold" (BH) strategy is when there is a specific time duration when stocks are bought at first, and they are then held for the duration to understand trading models (Dantas & Silva, 2018). BH is a trade that merely demands one buy/hold trade to make a fast profit. This downwardly simple approach BH even proved to be the heavy benchmark for many REPLNIQX addresses, which could not displace it constantly. On the contrary, the new DRL models have demonstrated the potential to perform better under good market conditions.

Two major approaches have been recognized for algorithmic trading: knowledge-based and data-driven. The knowledge-based strategy uses expert features. In contrast, the data-driven strategy, which has become the primary method in recent years, employs machine learning to facilitate decision-making. Additional data sets are used to enhance forecasts.

For instance, a trading strategy combined stock prices and sentiment analysis using a support vector machine (SVM), thus facilitating decision-making, especially in the information technology and retail sectors. Nonetheless, the writers considered deep learning models might be more efficient for sentiment analysis and price prediction tasks.

Moreover, newer models, like a hybrid attention network (HAN), have been developed to predict stock price trends using news sentiment and price data (Hu et al., 2018). This model gained an annualized return of 0.611 for a portfolio of 40 stocks, leaving the BH strategy far behind, as it returned only 0.04. However, the trading strategy needed to be more complex. Therefore, it could be enhanced by incorporating reinforcement learning that allows the system to learn from the environment and adapt over time.

The first deep learning applications in the Brazilian stock market trading systems used MLP models to predict stock prices and make trades. These developed systems have been observed to have a drastic reduction in error (measured by MAPE) and got higher returns than the traditional models like moving averages.

Exchange-traded funds (ETFs) are instruments that transmit the underlying indexes or simplify the operation of a pool of stocks by reducing the complexity of the overall trading process. They are the most preferred asset class in the world for algorithmic trading, though, on an individual level, a trader has the same ability to execute a trade. The argument being put forward here is not about being capable of how many models a human trader can trade in an algorithm; instead, it is more about which model is being used (Johnman et al., 2018). Agencies and institutional ink any other regulated business like Stockbrokers, Real Estate Agents, and Auto dealers as these persons are licensed personnel authorized to carry or sell the business's products and services. Stockbrokers always have IPOs of new stocks, and they are certified by the stock exchange to trade. To trade several stocks, robots usually contribute to complexity since the need to verify and monitor multiple is increased.

Dealing with execution costs and sequencing remains a significant hurdle in algorithmic trading. Most transaction cost models take only the volume of a given trade as a base point for measuring trading costs and neglect other primary factors like intraday price volatility and transparency. The algorithms being developed to execute the trade should be fair. This means they should be designed to consume the sparse data in a particular fashion and not over-consume it. Consumer finance is another sector that is being majorly disrupted by two main trends; in most cases, it's the new virtual currency that is responsible for the changes. Credit card companies that withdraw a 1.75% fee on particular transactions and do not charge the same transaction on others, with lower volatility experience, are likely candidates for such pricing models. Sequential decision-making models such as the RL and DRL are able to solve this problem by allowing systems to learn from their past mistakes and in the future, they can optimize their decisions (Dantas & Silva, 2018).

The latest advancements in RL have paved the way for the deployment of models. These models have greatly assisted in dealing with continuous action spaces, which is crucial in stock trading. DDPG is a learning technique that makes use of deterministic policies, whereas PPO is the methodology that brings about practical solutions. Both are considered the most advanced applications for trading strategy optimization in real business environments around the world. One example is the Palm device, which can act as a modem, and at the same time, users can enjoy the hardware add-on functions. Another potential application is the use of cellular telephones as satellite models. For stock trading, this is an insight that will hopefully help to change the game.

To sum up, the evolution of algorithmic trading has not only brought AI models to the next level. The synergy of sentiment analysis, reinforcement learning, and more advanced ways of dealing with risk and transaction costs have profoundly increased the capabilities of these systems, although traditional benchmarks like the BH strategy continue to be.

2.7 Deep Reinforcement Learning for Stock Trading

Reinforcement Learning and its advanced extension, Deep Reinforcement Learning, have become pivotal in the financial industry, particularly in stock trading, where the automation of decision-making processes is highly desirable. Traditional ML methods in stock trading typically focus on making price or trend predictions, which are then implemented into rule-based systems to guide trading actions. However, this approach is limited as it often needs to pay more attention to several critical aspects of market behavior, such as liquidity and transaction costs. The introduction of DRL marks a significant improvement, as it automates decision-making by identifying the essential rules that define effective trading strategies in a data-driven way.

One key advantage of DRL over traditional ML models is its ability to operate without predefined rules for executing trades. Instead, DRL models learn optimal strategies directly from the data by experimenting with different actions and adjusting to new market conditions. This is particularly useful in volatile financial markets, where price fluctuations, liquidity issues, and other market conditions evolve rapidly. As a result, DRL systems can automate trading by incorporating market predictions and decision-making processes. In contrast, conventional ML systems rely on rule-based mechanisms for executing trades.

Another notable advantage of using DRL for stock trading is its ability to optimize reward functions. Unlike traditional models that may focus on maximizing returns without considering other factors, DRL models can be designed to include essential market aspects such as transaction costs, market liquidity, and risk aversion (Fischer & Krauss, 2018). This flexibility allows traders to tailor their reward functions to align with their specific trading goals, such as maximizing risk-adjusted returns or minimizing drawdowns. In doing so, DRL models offer a holistic approach to trading by accounting for the interplay between short-term gains and long-term risks.

The foundations of RL and DRL lie in their ability to make sequential decisions by interacting with an environment. RL models rely on agents interacting with their environment, gathering information, and selecting actions to maximize cumulative rewards over time. Unlike supervised learning models, RL agents are not trained on labeled data but instead learn by trial and error. This approach is well-suited for stock trading, where agents must continually update their decisions based on ever-changing market conditions. Agents gradually learn which actions yield the best outcomes by experimenting with different strategies.

In RL, several key components must be considered: the agent, the environment, the state space, the action space, the reward function, and the policy. The agent interacts with the environment, which, in the context of stock trading, represents the financial markets. The environment’s dynamics constantly change, with factors such as news events, political developments, and economic indicators affecting market behavior. The state space represents the agent's possible conditions, such as stock prices, trading volumes, and other relevant indicators. The action space refers to the potential actions the agent can take, such as buying, selling, or holding a stock (Vázquez-Canteli & Nagy, 2019). Each action leads to a new state, and the agent receives a reward based on the success of that action in maximizing profit or other objectives. The reward function, therefore, plays a critical role in guiding the agent's behavior, as it defines the goals the agent is trying to achieve.

The design of the reward function is crucial in financial markets, where actions can have immediate and long-term consequences. For example, an agent might receive an immediate reward from a profitable trade but suffer a long-term penalty if that trade leads to increased risk or future losses. Thus, DRL models are typically designed to optimize short-term and long-term rewards, considering each decision's impact on future outcomes.

The concept of policy is also central to RL models. The policy determines the agent's behavior by mapping states to actions. Policies can be deterministic, where the agent always takes the same action in each state, or stochastic, where the agent selects actions based on a probability distribution. Stochastic policies are often preferred in stock trading, allowing the agent to explore different strategies and avoid becoming overly reliant on a single approach. Over time, as the agent learns which actions lead to the best outcomes, the policy is updated to reflect the agent's improved understanding of the environment.

The evolution of RL into DRL, facilitated using deep neural networks, has significantly expanded the capabilities of these models. While traditional RL models were limited by their reliance on handcrafted features and small state and action spaces, DRL models can handle much more complex environments, making them particularly suitable for financial markets. Deep neural networks enable DRL agents to automatically extract relevant features from raw market data, eliminating the need for manual feature engineering. This capability is critical in stock trading, where various factors influence markets, including historical prices, technical indicators, and sentiment data.

One of the seminal contributions to DRL was the development of the DQN, which extended Q-learning to continuous state spaces. DQN introduced several key innovations, including experience replay and target networks, which helped stabilize the learning process and improved performance in real-world applications (Mnih et al., 2015). Experience replay allows the agent to store and reuse past experiences, reducing the correlation between consecutive updates and making learning more efficient. On the other hand, target networks help prevent the agent from overfitting to recent experiences by maintaining a separate network that provides more stable estimates of the value function.

DQN has been successfully applied to stock trading, where it has demonstrated superior performance compared to traditional strategies such as the buy-and-hold (BH) strategy. For example, in experiments conducted on stock market indices like the Hong Kong HSI and the U.S. SP500, DQN-based agents outperformed both the BH and RRL agents regarding cumulative returns and Sharpe ratios. However, despite its success, DQN has limitations. One of the key challenges is that it relies solely on daily closing prices, which may only capture some relevant market information. Additionally, DQN assumes that actions are discrete, which can be limiting in financial markets where actions, such as trade sizes, are often continuous.

Advantage Actor-Critic is a prominent actor-critic algorithm that enhances policy gradient methods by incorporating an advantage function to stabilize and improve learning efficiency. In stock trading, A2C is a crucial component in ensemble strategies by simultaneously training separate actor and critic networks. Based on the current market state, the actor-network is responsible for selecting the optimal trading actions—such as buying, selling, or holding a stock. At the same time, the critic network evaluates these actions by estimating the advantage function. The advantage function measures how much better a chosen action is compared to the average action in each state, reducing the variance in policy updates and leading to more reliable and robust trading strategies. Figure 2-10 shows the process of the Actor-Critic.

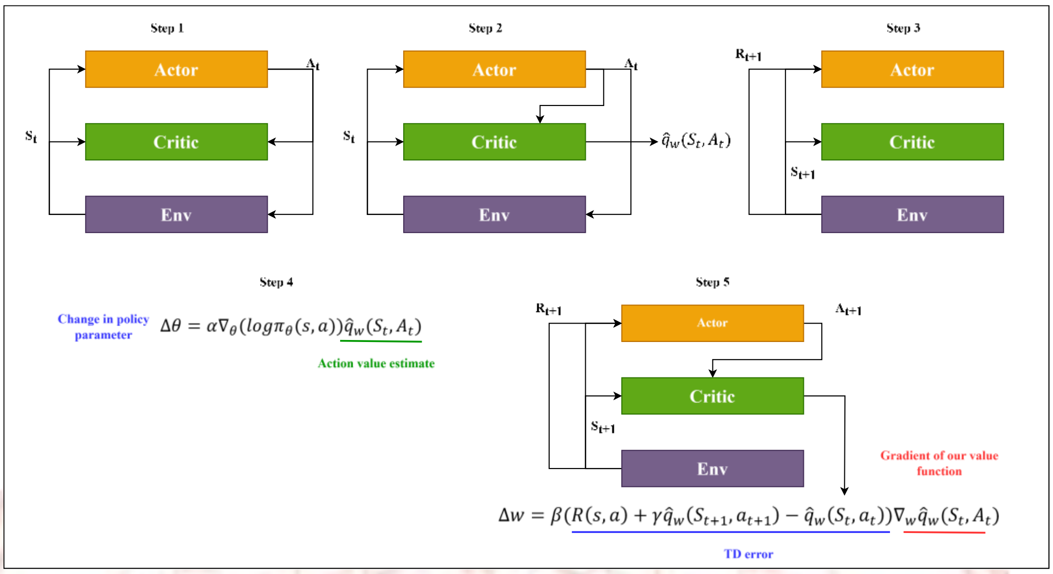

Step 1
Actor
Critic
Env
S_t
A_t
Step 2
$\hat{q}_w(S_t, A_t)$
Step 3
R_{t+1}
S_{t+1}
Step 4
Change in policy parameter
$\Delta\theta = \alpha\nabla_\theta(log\pi_\theta(s,a))\hat{q}_w(S_t, A_t)$
Action value estimate
Step 5
A_{t+1}
Gradient of our value function
$\Delta w = \beta(R(s,a) + \gamma\hat{q}_w(S_{t+1}, a_{t+1}) - \hat{q}_w(S_t, a_t))\nabla_w\hat{q}_w(S_t, A_t)$
TD error

FIGURE 2-10 The Actor-Critic Process

One of the significant strengths of A2C lies in its ability to leverage multiple parallel agents that interact with the trading environment independently. Each agent explores different parts of the state and action spaces, gathering diverse experiences that contribute to the learning process. After a set of interactions, the gradients computed by each agent are aggregated and used to update the global actor and critic networks. This parallelism not only accelerates the training process but also enhances the diversity of the training data, making the model more adaptable to varying market conditions. Additionally, by using synchronous updates, A2C ensures that the learning process remains stable and efficient, even when dealing with large batches of data typical in financial markets.

The objective function for A2C is:

$$\nabla J_\theta(\theta) = \mathbb{E}[\textstyle\sum_{t=1}^{T} \nabla_\theta log\pi_\theta(a_t|s_t)A(s_t|a_t)] \qquad (2\text{-}4)$$

Where $\pi_\theta(a_t|s_t)$ is the policy network, $A(s_t|a_t)$ is the advantage function can be written as:

$$A(s_t, a_t) = Q(s_t, a_t) - V(s_t) \qquad (2\text{-}5)$$

Or

$$A(s_t, a_t) = r(s_t,\ a_t, s_{t+1}) + \gamma V(s_{t+1}) - V(s_t) \qquad (2\text{-}6)$$

In practical applications, A2C has demonstrated its effectiveness in developing sophisticated trading strategies that account for both short-term gains and long-term risks. By optimizing the reward function to include factors such as transaction costs, market liquidity, and risk aversion, A2C enables the creation of trading models that are profitable and resilient to market volatility. The actor-critic architecture allows the model to continuously refine its policy based on real-time feedback from the market, ensuring that the trading decisions remain aligned with the desired financial objectives. This adaptability and robustness make A2C an excellent choice for stock trading, where the ability to respond to rapidly changing market dynamics is essential for sustained success.

Recent studies have demonstrated the practical applicability of A2C in stock trading environments. For instance, a conceptually simple, lightweight framework for deep reinforcement learning (Mnih et al, 2016) uses asynchronous gradient descent to optimize deep neural network controllers. Researchers present asynchronous variants of four standard reinforcement learning algorithms and show that parallel actor-learners stabilize training, allowing all four methods to train neural network controllers successfully. The best-performing method, an asynchronous variant of actor-critic, surpasses the current state-of-the-art on the Atari domain while training for half the time on a single multi-core Central processing unit (CPU) instead of a GPU. Furthermore, the asynchronous actor-critic succeeds on various continuous motor control problems and on a new task of navigating random 3D mazes using a visual input.

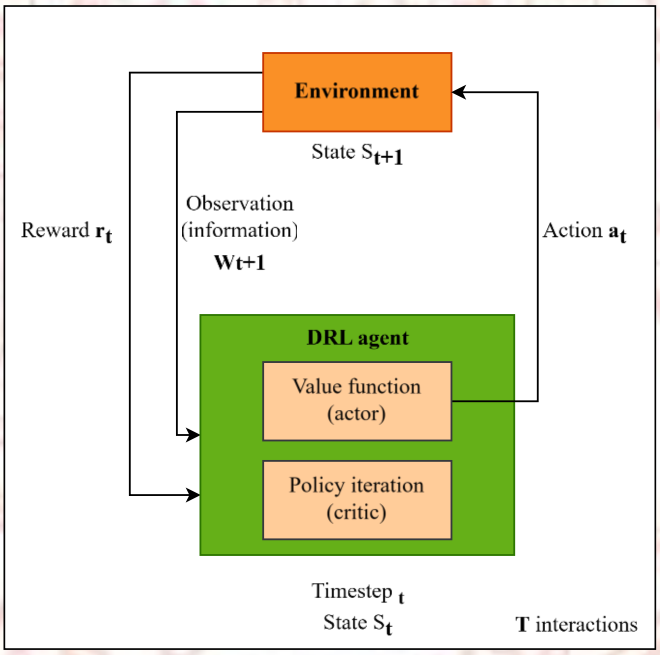


FIGURE 2-11 Illustration of general actor-critic models

To address the limitations of DQN, researchers developed the DDPG model, which extends the Q-learning framework to continuous action spaces. DDPG uses an actor-critic architecture, where the actor-network determines the best action to take, and the critic network evaluates the quality of that action by estimating the expected return. The illustrates the DRL models' general components, describing specific components of the actor-critic methods in Figure 2-11. This approach allows DDPG to handle the continuous nature of trading actions, such as varying trade sizes or adjusting portfolio allocations. The use of deep neural networks in actor and critic networks enables DDPG to learn complex trading strategies previously out of reach for traditional RL models.

DDPG has demonstrated superior performance in stock trading applications compared to traditional strategies and earlier DRL models. However, it does come with certain limitations. One of the main challenges is that DDPG can overfit, especially when dealing with noisy or highly volatile markets. To address this, techniques like experience replay and target networks are employed. These methods help stabilize the learning process, preventing the model from becoming overly reliant on recent experiences and making it more robust.

Building on the strengths of DDPG, PPO, shown in Figure 2-12, was developed, which introduced improvements to the actor-critic framework. PPO uses a more stable policy gradient approach, ensuring that the policy is updated gradually to avoid drastic changes, which can be problematic in volatile market conditions. It also includes a surrogate objective function that discourages extensive policy updates, promoting more stable exploration and strategy development. This makes PPO particularly suitable for financial markets, where sudden shifts in trading strategies can lead to significant risks or losses.

The policy gradient theorem is foundational for algorithms shown in (2-7):

$$\nabla_\theta J(\theta) = \mathrm{E}[\nabla_\theta \log \pi_\theta\, (a/s) Q^\pi(s/a)] \tag{2-7}$$

where:

θ are the parameters of the policy $\pi_\theta(a/s)$

$J(\theta)$ is the expected reward objective function.

$Q^\pi(s,a)$ is the action-value function under policy π.

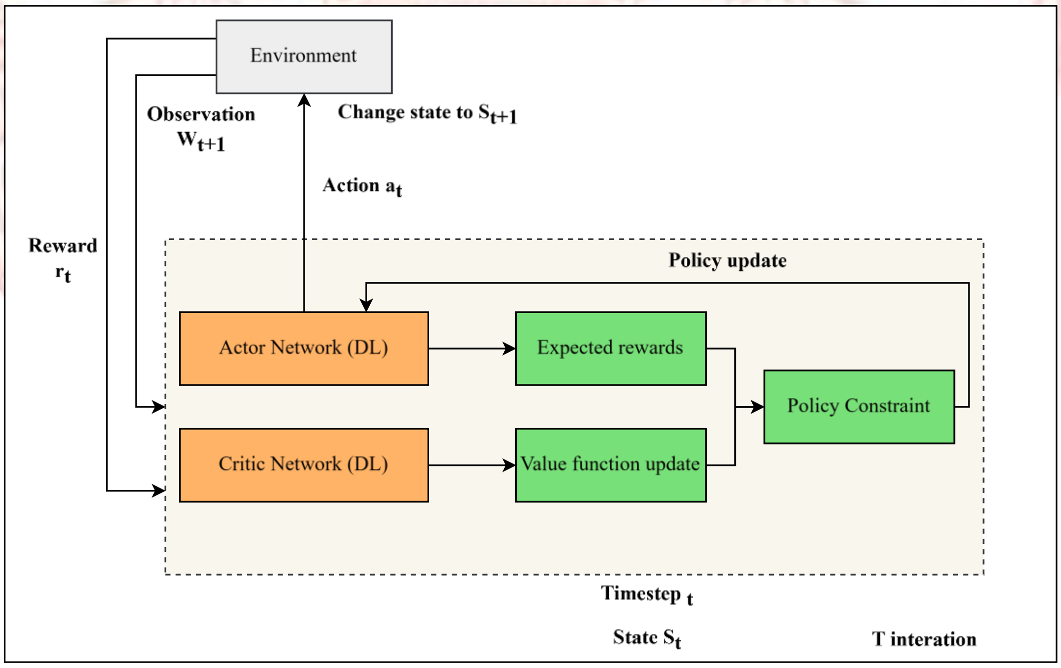


FIGURE 2-12 Illustration of the PPO model

PPO has been applied successfully in stock trading, demonstrating faster convergence and more stable performance than DDPG. In a study on 30 liquid stocks from the Dow Jones Industrial Average, a PPO-based trading system achieved a cumulative return of 70.4%, compared to 38.6% for the BH strategy (Yang et al., 2020). Additionally, PPO was found to have lower volatility and a higher Sharpe ratio than traditional strategies, making it an attractive option for traders looking to balance returns with risk. Despite its advantages, PPO still needs help in highly volatile markets, where the need for frequent policy updates can limit its effectiveness.

As DRL models evolve, researchers have begun exploring ensemble methods combining multiple agents to create more robust trading systems. Ensemble methods leverage the strengths of different DRL models, such as the fast convergence of PPO and the ability of DDPG to handle continuous action spaces, to improve overall performance. In stock trading, ensemble methods have been shown to enhance cumulative returns, reduce volatility, and increase the Sharpe ratio compared to single-agent systems. Furthermore, ensemble methods can help mitigate the risk of overfitting by allowing agents to explore different strategies simultaneously and select the best-performing ones.

Another active research area in DRL for stock trading is integrating risk management into the reward function. Traditional DRL models often focus solely on maximizing returns, but this approach can lead to excessive risk-taking, particularly in volatile markets. Researchers have begun incorporating risk-aware objectives, such as minimizing drawdowns or maximizing risk-adjusted returns, into the reward function to address this. By doing so, DRL agents can learn to balance the trade-off between risk and reward, making them more suitable for real-world trading applications where risk management is crucial.

Despite the significant advancements in DRL for stock trading, several challenges remain. One of the main issues is the need for more interpretability in DRL models, which makes it difficult for human traders to understand and trust the decisions made by these agents. This is particularly problematic in financial markets, where transparency and accountability are essential. Another challenge is that many DRL models are tested in simulated environments, which may not accurately reflect the complexities of real-world markets. As a result, DRL agents may perform well in simulations but struggle when exposed to live trading conditions.

To address these challenges, future research should focus on improving the interpretability of DRL models and developing more realistic market simulations. Additionally, there is a growing need for DRL models that can adapt to changing market conditions in real time and incorporate external factors such as news sentiment or macroeconomic indicators into their decision-making process. By addressing these challenges, DRL can potentially revolutionize stock trading and other areas of finance, offering traders more sophisticated and adaptive tools for navigating complex markets.

In conclusion, the development of RL and DRL for stock trading has made significant strides over the years, evolving from early rule-based systems to advanced models capable of learning directly from raw market data. While there are still challenges to overcome, such as model interpretability and the inclusion of real-world factors, the potential benefits of DRL in the financial domain are undeniable. Future research should continue exploring the integration of risk management into DRL models, the use of ensemble methods, and the development of more realistic market simulations to improve the applicability of these models in live trading environments.

2.8 Dynamic Allocation

A key component of investing strategy is asset allocation, which involves allocating an investor's capital across several asset classes, such as cash, bonds, and stocks, to balance risk and return in accordance with their investment horizon, risk tolerance, and financial objectives. The main goal is to attain optimal returns while minimizing potential losses from any asset class through diversification of investments.

Building on this basis, investors can use a customized strategy called Volatility-Based Allocation. In this investment strategy, investors adjust how much money they put into risky assets based on how much the market is fluctuating. When the market is calm and not very volatile, investors allocate more funds to risky investments because the conditions seem stable and favorable. Conversely, when the market is shaky and highly volatile, they reduce their investments in risky assets to minimize potential losses. This approach aims to balance the chances of making good returns while controlling the risks by responding to changing market conditions.

A study explored how ANNs, which are advanced computer programs, can predict market volatility more accurately. Their goal was to improve an asset allocation strategy that maintains a specific level of risk, known as target volatility. The strategy works by dynamically shifting investments between a risky asset, like stocks, and a risk-free asset, such as cash. One challenge they faced was the limited availability of high-volatility data because extreme market swings, like those during financial crises, don't happen often. To overcome this, they compared current high-volatility periods with past low-volatility data to have more information for their models. They tested their ANN-based approach against traditional methods like the volatility index, historical volatility, exponentially weighted moving average (EWMA), and the GARCH model. Their results showed that the ANN method performed better in forecasting volatility, which helped in making more informed investment decisions. They conducted their study using data from the Korea Composite Stock Price Index 200 (KOSPI 200) and certificate of deposit interest rates from January 2006 to February 2016 (Kim & Enke, 2016). In this study, the weight of equity in the portfolio is calculated by Eq. (2-8), where $\hat{\sigma}_t^{equity}$ is an estimate of the volatility of equity returns, and σ^{target} is the target volatility, applied between time *t* and the next rebalancing time, *t + 1*.

$$w_t^{equity} = \min\left(\frac{\sigma^{target}}{\hat{\sigma}_t^{equity}}, 100\%\right) \tag{2-8}$$

Some parameters are needed to implement the target volatility strategy. These include the volatility target, the computation of current volatility, the maximum leverage amount, and the rebalancing frequency. However, the maximum weight of equity is restricted to a constraint (i.e., no leverage) since the objective of this study is to use ANNs for volatility forecasting to enhance the ability of an asset allocation strategy based on the target volatility.

Another important study reviewed various ML models used in the financial sector, focusing mainly on predicting stock prices and managing portfolios. They found that traditional models like ARIMA (Auto Regressive Integrated Moving Average) and exponential smoothing didn't perform as well as DL models when it came to predicting stock prices and volatility. This is because financial data is often complex, has high dimensions, and changes dynamically over time. Among the DL models, LSTM networks were particularly effective, outperforming other methods like SVM and MLP. However, DRL was used less frequently than these models. Conclude that DL models offer significant advantages for financial predictions due to their ability to handle intricate data patterns.

These studies highlight the growing importance of using advanced computer models, such as neural networks and deep learning, to improve volatility forecasting and asset allocation strategies. By better predicting how volatile the market will be, investors can make more informed decisions about where to allocate their money, aiming to achieve higher returns while managing risks effectively. As technology and data analysis techniques advance, these methods will become even more integral to successful investment strategies.

2.9 Economic Analysis

2.9.1 Return on Investment (ROI)

ROI is a fundamental financial metric used to evaluate the efficiency or profitability of an investment. It measures the return relative to the investment's cost, providing a straightforward way to assess the effectiveness of various investments. A study discusses ROI's simplicity and versatility, demonstrating its applicability beyond traditional financial investments to marketing campaigns, project management, and human resources. Their work emphasizes how ROI facilitates decision-making by providing a clear metric for comparing the profitability of different initiatives. Over the years, enhancements to ROI calculations have been introduced to incorporate the time value of money, leading to more sophisticated metrics like Return on Invested Capital (ROIC) and Economic Value Added (EVA). These advancements have allowed businesses to better understand their investment returns by accounting for factors such as capital costs and economic profit. Additionally, the integration of data analytics has enabled more dynamic and real-time ROI assessments, allowing for more accurate and timely investment evaluations (Brigham & Ehrhardt, 2013). This foundational understanding underscores ROI's enduring relevance as a critical strategic planning and operational management tool. The ROI is calculated by (2-9):

$$ROI = \left(\frac{Net\ Profit}{Cost\ of\ Investment}\right) \times 100\% \tag{2-9}$$

A higher ROI indicates a more profitable investment, making it a valuable tool for comparing investment opportunities.

2.9.2 Drawdown

Drawdown is a risk metric that quantifies the decline from a portfolio's peak value to its lowest point over a specific period. It provides insight into the potential loss an investment might experience, helping investors understand the risk involved. The drawdown is calculated using the following equation:

$$Drawdown = \left(\frac{Peak\ Value - Trough\ Value}{Peak\ Value}\right) \times 100\% \tag{2-10}$$

Drawdown (Magdon-Ismail & Atiya, 2004) is crucial for assessing the downside risk and the emotional resilience investors require to withstand market volatility. It has been extensively studied as a critical measure of investment risk, particularly in portfolio management and behavioral finance. A study integrates drawdown metrics into portfolio optimization, highlighting the importance of minimizing potential losses to enhance long-term investment stability. Bailey and colleagues demonstrate that by incorporating drawdown considerations, investors can develop trading strategies that aim for high returns and effectively manage and mitigate significant losses during market downturns. Their research underscores drawdown's role in understanding and controlling investment risks, thereby improving portfolio resilience. Additionally, advancements in computational methods have allowed for more precise drawdown analyses, such as calculating maximum and average drawdown over rolling periods. This comprehensive approach to drawdown assessment has reinforced its significance in constructing robust investment portfolios, ensuring that investors are better prepared to handle adverse market conditions.

While drawdown measures any decline from a portfolio's peak to its trough, maximum drawdown focuses on the largest such decline over a specific period. It represents the most significant loss an investment could incur, providing critical insight into its risk profile. The maximum drawdown is calculated by identifying the most considerable drawdown among all peak-to-trough declines during the period:

$$Maximum\ Drawdown = \max\left\{\left(\frac{Peak\ Value - Trough\ Value}{Peak\ Value}\right) \times 100\%\right\} \qquad (2\text{-}11)$$

For instance, if a portfolio experiences several drawdowns 5%, 8%, 15%, and 7% over a year, the maximum drawdown is 15%. This figure is crucial for investors as it indicates the worst possible loss they might face, enabling them to gauge their risk tolerance and adjust their investment strategies accordingly.

2.9.3 Sharpe Ratio

The Sharpe Ratio is a widely used measure of risk-adjusted return, developed by Nobel laureate William F. Sharpe. It assesses how much excess return an investment generates per unit of risk, allowing investors to compare the performance of different portfolios or assets. The Sharpe Ratio is calculated using the following equation:

$$Sharpe\ Ratio = \frac{R_p - R_f}{\sigma_p} \qquad (2\text{-}12)$$

where:

R_p is the return of the portfolio.

R_f is the risk-free rate.

σ_p is the standard deviation of the portfolio's excess return.

A higher Sharpe Ratio indicates a more favorable risk-adjusted return, making it a crucial tool for portfolio optimization and performance evaluation. In this influential work, Sharpe presented the ratio to evaluate the performance of mutual funds by comparing their excess returns to the volatility of those returns. This metric became essential in the rise of modern portfolio theory and quantitative finance, providing a standardized method for assessing risk-adjusted returns across diverse investment strategies. Sharpe's analysis demonstrated that the ratio effectively captures the trade-off between risk and return, enabling investors to make more informed decisions when selecting and managing portfolios. Over the decades, the Sharpe Ratio has been extensively validated and refined, addressing its initial limitations and expanding its applicability to various asset classes, including equities, bonds, and alternative investments like cryptocurrencies and ESG-focused funds (Sharpe, 1966) Its enduring relevance is evident in academic research and practical investment management, where it continues to guide portfolio optimization and comparative performance analysis.

2.9.4 Calmar Ratio

A performance metric that evaluates the return of an investment relative to its maximum drawdown, providing insight into the risk-adjusted return over a specified period. It is beneficial for assessing the performance of hedge funds and managed portfolios. The Calmar Ratio is calculated using the following equation:

$$Calmar\ Ratio = \frac{\text{Annualized Return}}{Maximum\ Drawdown} \tag{2-13}$$

where:

Annualized Return is the compounded annual growth rate of the investment.

Maximum Drawdown is the largest peak-to-trough decline during the investment period.

A higher Calmar Ratio indicates a more favorable balance between return and downside risk, making it a valuable tool for investors focused on capital preservation and consistent performance. Some work emphasized the ratio's ability to capture the balance between an investment's return and its worst-case scenario loss, making it especially relevant during market instability. The Calmar Ratio effectively differentiates between high-performing funds with manageable drawdowns and those with similar returns but greater risk exposure, providing investors with a clear metric for assessing growth and risk (Magdon-Ismail & Atiya, 2004). Research demonstrated that incorporating the Calmar Ratio into portfolio evaluation frameworks enhances the ability to construct portfolios that maximize returns while minimizing potential losses. Additionally, the metric has been widely adopted in evaluating algorithmic trading strategies and various asset classes, including equities, commodities, and cryptocurrencies, reflecting its versatility in modern investment landscapes. Advancements in financial modeling have further improved the accuracy and real-time applicability of the Calmar Ratio, reinforcing its role as a vital tool for investors prioritizing both growth and the mitigation of significant losses.

The review of related works reveals a clear opportunity to leverage advanced DRL models in portfolio management, particularly within emerging markets such as the SET. Chapter 3 introduces the methodology for developing a customized DRL trading environment and optimizing agent performance to meet these unique challenges.

CHAPTER 3
METHODOLOGY AND EXPERIMENT

Having established the theoretical foundation and identified the research gap, Chapter 3 outlines the methodology for developing a deep reinforcement learning (DRL) model for dynamic portfolio management on the Stock Exchange of Thailand (SET). The chapter begins with an Overview of Model Architecture (Section 3.1), outlining the frameworks of Proximal Policy Optimization (PPO), Advantage Actor-Critic (A2C), and Deep Q-Network (DQN). Data Preparation (Section 3.2) and Data Preprocessing (Section 3.3) detail the selection and transformation of stock data, followed by Hyperparameter Analysis (Section 3.4), which fine-tunes key parameters to enhance model stability. Sections 3.5 and 3.6 cover the Actor-Critic network structures and DQN implementation, respectively, explaining the technical foundations of each model. Model Training (Section 3.7) describes the learning process of these agents. In contrast, Testing and Evaluation (Section 3.8) assesses performance using metrics like ROI and Sharpe Ratio, providing insights into the models' profitability and risk management capabilities. This methodology framework sets the stage for a robust DRL-based trading model tailored to SET’s market conditions.

3.1 Overview of Model Architecture

The model architecture for developing a deep reinforcement learning (DRL)-based trading system, shown in FIGURE 3-1, is designed in three main stages: Data Collection, Data Preprocessing, and Deep Reinforcement Learning Model Implementation. These stages work together to facilitate the learning and execution of trading strategies optimized for profitability and risk management on a selected portfolio of stocks in the Stock Exchange of Thailand (SET).

Stage 1: Data Collection

In the first stage, a portfolio of 10 representative stocks is selected based on market significance, liquidity, and sector diversity criteria. Each stock is represented by its historical data, including daily price movements, which provide the foundational input for the model. These data points serve as raw inputs, capturing trends, volatility, and individual stock behavior, which is essential for building a robust and representative dataset.

Stage 2: Data Preprocessing

Data preprocessing transforms the raw stock data into formats suitable for training the DRL model. This step includes two primary processes: Normalization and Feature Engineering. Normalization involves converting price changes into percentage changes, making the data comparable across stocks. A Volume-Weighted Average Price (VWAP) feature is also calculated to capture short-term price trends and enhance the model's understanding of price movements. After feature engineering, a Standard Scaler is applied to standardize the dataset, ensuring that features are scaled consistently, which is crucial for stable and practical model training.

Stage 3: Deep Reinforcement Learning Model Implementation

The core of the architecture is the DRL model, where Trading Agents interact with a simulated environment that mimics the stock market. The trading agents—implemented using algorithms such as TDQN (Target Deep Q-Network), A2C (Advantage Actor-Critic), and PPO (Proximal Policy Optimization)—are responsible for making buy, sell, or hold decisions based on the observations from the environment. The environment provides real-time data on raw prices, engineered features (like VWAP), and close prices and calculates rewards based on profit or loss after each action the agent takes.

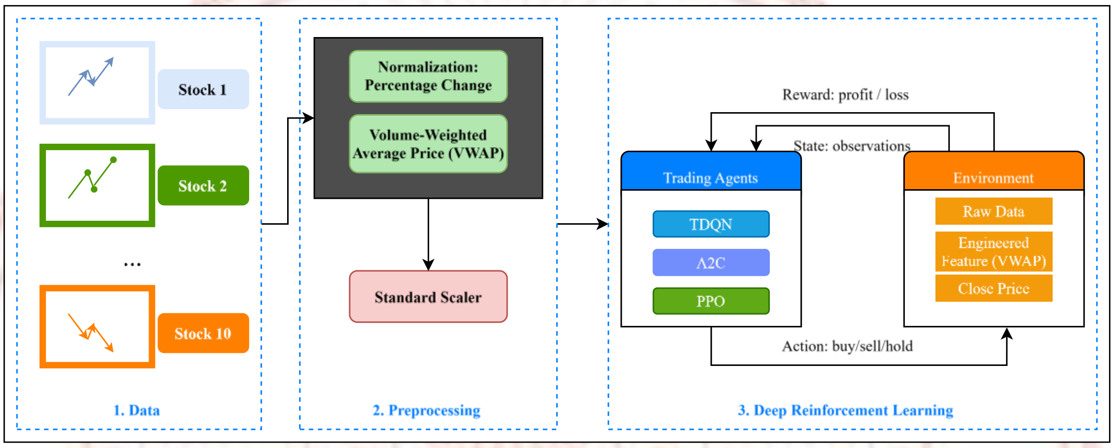


FIGURE 3-1 Overview of Model Architecture

Each trading agent operates in a feedback loop with the environment. The agent observes the current state, performs an action (e.g., buying or selling a stock), and receives a reward based on the outcome of that action. This reward feedback guides the agent’s learning, reinforcing profitable strategies while discouraging unprofitable ones. Over time, the agents learn to optimize their trading policies through continuous interaction with the environment, aiming to maximize cumulative rewards in alignment with market conditions.

This multi-stage architecture, combining data processing with DRL agents, allows for an adaptive and data-driven trading approach, leveraging historical trends and real-time decision-making capabilities. Through this setup, the model aims to produce trading strategies that can dynamically respond to SET’s unique market conditions, providing a balance between profitability and risk management.

This study utilizes a dataset of ten stocks: ADVANC, AOT, BDMS, CPN, INTUCH, IVL, MINT, PTTEP, TISCO, and SCC Exploration. These stocks were selected due to their significant market presence, high trading volumes, and potential for substantial returns. The dataset spans from January 1, 2017, to June 1, 2023, providing ample historical data for the Model Overview shown in Figure 3-1.

3.2 Data Preparation

The preprocessing step in the methodology involves handling missing values and then transforming raw stock data into percentage changes, which is a part of feature engineering—selecting a diverse portfolio of stocks spanning several vital economic sectors, including Telecommunications, Aviation, Healthcare, Real Estate, Energy/Oil

& Gas, Financial Services, Hospitality/Retail, Industrials/Cement, and Chemical/Textiles. This strategic diversification across different industries is designed to optimize the portfolio’s performance and enhance its robustness. By investing in various sectors, the portfolio can better withstand sector-specific downturns, capitalize on growth opportunities in multiple areas, and achieve a balanced risk-return profile. This approach mitigates risks associated with market volatility and leverages each sector's unique strengths and growth potentials, ensuring sustained and stable investment growth.

3.2.1 Advanced Info Service PCL (AIS)

Listed under the ticker symbol ADVANC, AIS is Thailand's largest mobile phone operator. It provides various telecommunications services, including mobile networks, broadband internet, and digital solutions. As a leading company in the SET50, AIS holds a significant market share in the Thai telecommunications industry, a critical sector for the country's economic infrastructure.

The stock is a critical component of the SET50 index, representing a considerable portion of the market capitalization within the telecommunications sector. AIS's consistent performance is reflected in its vital financial metrics, including a high market capitalization, robust earnings growth, and attractive dividend yield. The company's stock is known for its liquidity and is frequently traded by domestic and international investors.

Strategically, AIS is focused on expanding its 5G network and enhancing digital services, positioning itself to capitalize on Thailand's growing demand for connectivity. With a forward-looking approach, AIS will likely maintain its industry leadership.

3.2.2 Airports of Thailand PCL (AOT)

AOT is the foremost airport operator in Thailand, overseeing key international airports such as Suvarnabhumi and Don Mueang. AOT plays an indispensable role in the Thai economy, particularly within the tourism and transportation sectors, making it a pivotal entity in driving the country’s economic activities. The company’s efficient airport operations management significantly boosts Thailand’s tourism industry, a significant contributor to the national GDP.

It is a significant constituent of the SET50 index, reflecting its large market capitalization and the critical nature of its services. The company's financial performance is robust, supported by steady growth in passenger numbers and aircraft movements, contributing to its revenue stability. The stock is favored for its defensive qualities, offering resilience in market volatility and attracting domestic and foreign investors.

Looking ahead, AOT aims to expand airport infrastructure to accommodate future passenger growth. The company's investment plans include upgrading existing facilities and enhancing service efficiency, which will help solidify its market position and maintain its substantial influence.

3.2.3 Bangkok Dusit Medical Services PCL (BDMS)

Thailand's leading healthcare provider operates a network of hospitals and medical facilities nationwide. BDMS offers various medical services, including specialized treatments, diagnostics, and wellness programs. The company benefits from Thailand's growing healthcare demand, driven by local and international patients seeking high-quality medical care. BDMS exhibits robust financial performance with consistent revenue growth, strong profit margins, and a solid dividend history. The stock is attractive to investors due to the essential nature of healthcare services and the company's reputation for excellence. BDMS is focused on expanding its service offerings, enhancing patient care technologies, and exploring international markets to sustain its growth trajectory.

3.2.4 Central Pattana PCL (CPN)

CPN is a leading real estate developer specializing in developing and managing retail properties, office buildings, and residential projects. CPN is renowned for its high-quality shopping malls, such as CentralWorld and CentralPlaza, major commercial hubs attracting millions of visitors annually. The company boasts strong financial metrics, including substantial revenue from property sales and rentals, healthy profit margins, and consistent dividend payouts. Investors favor CPN's stock for its stable income and growth potential driven by Thailand's expanding urbanization and consumer spending. The company is committed to sustainable development, innovative property solutions, and strategic acquisitions to enhance its portfolio and market presence.

3.2.5 Intouch Holdings PCL (INTUCH)

A major player in Thailand's telecommunications and digital services landscape. The company provides a wide range of services, including mobile communications, digital media, and fintech solutions. Intouch has established a strong market presence through continuous innovation and strategic partnerships. Financially, the company showcases solid revenue growth, healthy profit margins, and a reliable dividend payout, making it an attractive option for investors seeking stability and development in the tech sector. Intouch is committed to advancing its digital infrastructure and expanding its service offerings, positioning itself well to capitalize on Thailand's increasing demand for digital transformation.

3.2.6 Indorama Ventures Ltd (IVL)

Specializing in the production of polyester, petrochemicals, and related products. IVL operates an extensive network of manufacturing facilities and distribution channels across multiple continents, making it a significant player in the global textiles and chemical industries. The company exhibits financial solid performance with robust revenue growth, efficient production processes, and healthy profit margins. IVL's stock is attractive to investors due to its global reach, diversified product portfolio, and consistent dividend payouts. The company is committed to expanding its production capacities, investing in sustainable and environmentally friendly technologies, and exploring new markets to drive future growth and maintain its competitive edge.

3.2.7 Minor International PCL (MINT)

Operating a diverse portfolio of hotels, resorts, restaurants, and retail stores. MINT owns well-known brands such as Minor Hotels, The Pizza Company, and Sushi Samba. The company's extensive global presence allows it to tap into international markets, driving revenue growth and brand recognition. MINT demonstrates solid financial performance with steady revenue streams from its hospitality and retail operations, healthy profit margins, and a solid dividend policy. The stock is attractive to investors due to the company's resilience and growth potential in the dynamic hospitality and consumer sectors. MINT is committed to expanding its global footprint, enhancing customer experiences, and diversifying its brand portfolio to capitalize on emerging market trends and consumer preferences.

3.2.8 PTT Exploration and Production PCL (PTTEP)

PTTEP is a crucial player in Thailand’s energy sector, specializing in oil and natural gas exploration and production. PTTEP's operations are vital for securing Thailand’s energy supply, and its activities span both domestic and international territories. The company’s contributions are essential to the nation’s energy security and economic stability, providing a significant portion of its energy needs.

It is an essential component of the SET50 index, representing the energy sector's substantial contribution to the Thai economy. The company’s financial performance is characterized by its strong revenue streams and profitability, supported by successful exploration projects and production efficiency. Stock is highly regarded for its steady dividends and potential for capital appreciation, making it a preferred choice for investors seeking exposure to the energy sector.

PTTEP is committed to expanding its exploration and production activities, particularly in high-potential areas, while investing in sustainability initiatives. The company’s focus is on operational efficiency and environmental responsibility positions.

3.2.9 TISCO Financial Group PCL (TISCO)

A prominent player in Thailand's financial services sector. The company offers various financial products and services, including banking, asset management, and securities brokerage. TISCO is known for its strong market presence, innovative financial solutions, and customer-centric approach. The company exhibits solid financial health, consistent revenue growth, strong asset quality, and attractive profitability ratios. TISCO's stock is appealing to investors seeking exposure to the financial sector, offering both growth potential and dividend income. The company is focused on expanding its digital banking services, enhancing customer experience, and exploring new financial technologies to stay competitive in a rapidly evolving market.

3.2.10 Siam Cement Group (SCC)

It is one of Thailand’s largest and most diversified industrial conglomerates, with operations spanning cement production, building materials, chemicals, and packaging. SCC plays a critical role in Thailand's infrastructure development and industrial growth. The company boasts strong financials, including substantial revenue from its diverse business segments, solid profit margins, and consistent dividend payments. Investors favor SCC's stock for its stability, diversified revenue streams, and growth prospects driven by ongoing infrastructure projects and industrial demand. The company is focused on innovation, sustainable practices, and strategic acquisitions to enhance its product offerings and expand its domestic and international market presence.

3.3 Data Preprocessing

Effective data preprocessing is crucial for the success of machine learning models, particularly in the context of financial data, where noise and variability are prevalent. This section outlines the comprehensive preprocessing pipeline applied to the raw stock data to prepare it for modeling with the RL agent.

3.3.1 Data Retrieval

The initial step involves acquiring historical stock data using the Yahoo Finance API via the yfinance library. For each stock symbol, daily data encompassing the following attributes is retrieved:

Open: The price at which the stock opened at the beginning of the trading day.

High: The highest price reached during the trading day.
Low: The lowest price reached during the trading day.
Close: The price at which the stock closed at the end of the trading day.
Volume: The number of shares traded during the trading day.

3.3.2 Missing Value Handling

Financial datasets often need more value due to various reasons, such as market holidays or data transmission issues. To ensure data integrity and model reliability, missing data points are handled through the following approaches:

Removal of Missing Data: Rows with missing values are dropped to prevent the introduction of bias or errors during model training.

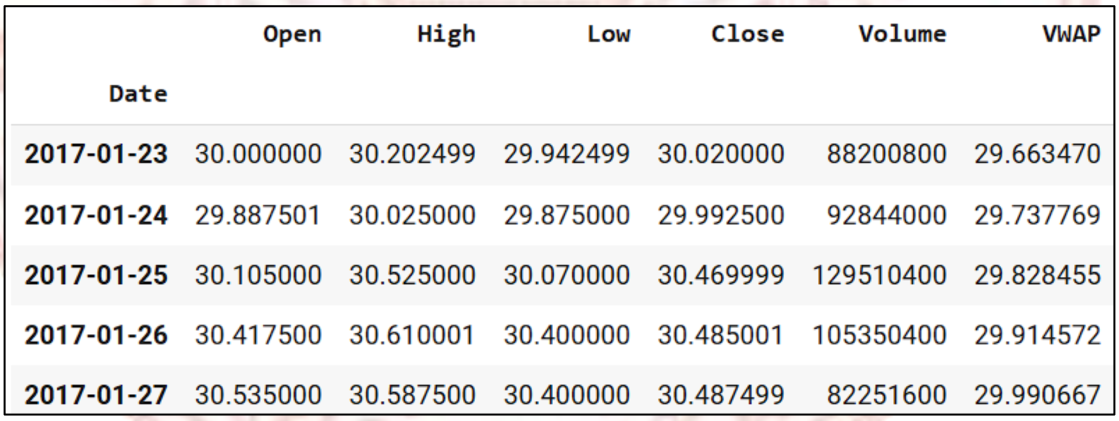
3.3.3 Feature Engineering:

A key feature engineered in this process is the Volume-Weighted Average Price, calculated over a 14-day window to capture short-term price trends.

Percentage Change Calculation: The raw stock data is transformed into percentage changes to normalize the values and focus on relative price movements rather than absolute prices. The formula used was:

$$Percentage\ Change = \left(\frac{Close_t - Close_{t-1}}{Close_{t-1}}\right) \times 100\% \tag{3-1}$$

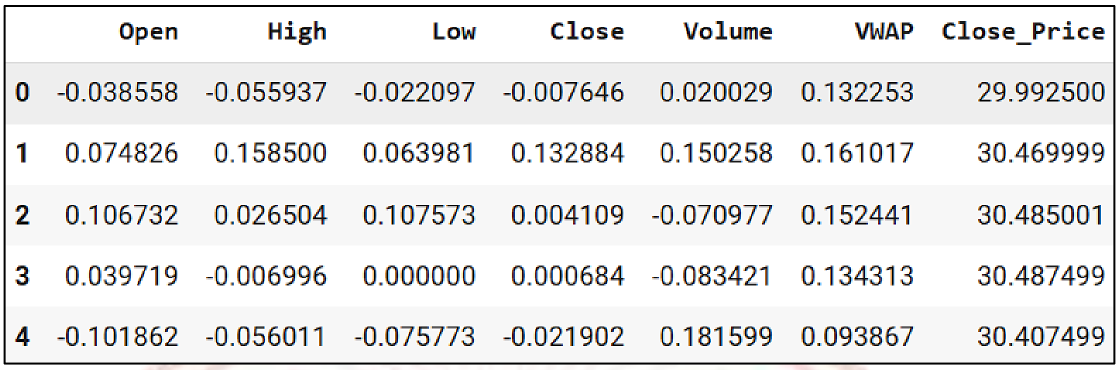
The result after calculation show in Figure 3-2



Date	Open	High	Low	Close	Volume	VWAP
2017-01-23	30.000000	30.202499	29.942499	30.020000	88200800	29.663470
2017-01-24	29.887501	30.025000	29.875000	29.992500	92844000	29.737769
2017-01-25	30.105000	30.525000	30.070000	30.469999	129510400	29.828455
2017-01-26	30.417500	30.610001	30.400000	30.485001	105350400	29.914572
2017-01-27	30.535000	30.587500	30.400000	30.487499	82251600	29.990667

FIGURE 3-2 Illustration data after Feature Engineering

Normalization: The percentage changes are further normalized by dividing by the maximum value to standardize the data across different stocks show in Figure 3-3.



	Open	High	Low	Close	Volume	VWAP	Close_Price
0	-0.038558	-0.055937	-0.022097	-0.007646	0.020029	0.132253	29.992500
1	0.074826	0.158500	0.063981	0.132884	0.150258	0.161017	30.469999
2	0.106732	0.026504	0.107573	0.004109	-0.070977	0.152441	30.485001
3	0.039719	-0.006996	0.000000	0.000684	-0.083421	0.134313	30.487499
4	-0.101862	-0.056011	-0.075773	-0.021902	0.181599	0.093867	30.407499

FIGURE 3-3 Illustration data after Normalization

After that, the StandardScaler was created and applied to these features using the fit transform method. This method calculates each feature's mean and standard deviation and scales them to have a mean of zero and a standard deviation of one. The scaled features are converted back into a pandas Data Frame, ensuring that the original feature names are retained for clarity. The unscaled 'Close_Price' column is added to this Data Frame to preserve the actual closing prices for plotting or future reference. By standardizing the features, the code ensures that all variables contribute equally to the model training process, particularly important for algorithms sensitive to input data's scale. This step enhances the performance and convergence speed of the machine learning models used later in the pipeline.

3.3.4 Training and Testing Split

The preprocessed dataset is divided into training and testing sets to evaluate the model's performance and generalization capability. A common practice is employed, allocating 70% of the data for training and the remaining 30% for testing. This split ensures that the model is trained on a substantial portion of the data while retaining enough data to assess its performance in unseen market conditions.

3.3.5 Data Visualization

Visualizing the processed data aids in understanding the underlying patterns and verifying the effectiveness of preprocessing steps. Specifically, plotting the Close_Price for training and testing sets provides insights into price trends and volatility shown in Figure 3-4.

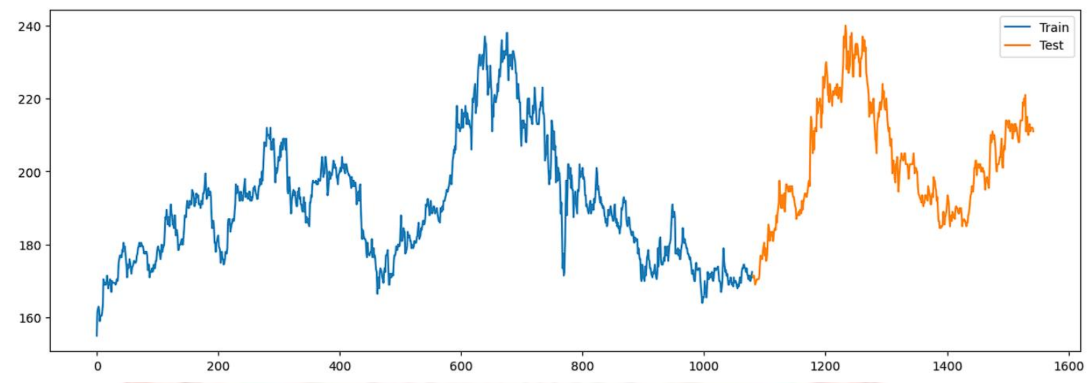


FIGURE 3-4 Visualizing data split of Advanced Info Service PCL (AIS)

3.4 Hyperparameter Analysis

Hyperparameters play a pivotal role in the performance and efficiency of machine learning models. Unlike model parameters learned during training, hyperparameters are predefined settings that govern the training process. In reinforcement learning, particularly with the PPO, A2C, and DQN algorithms, selecting appropriate hyperparameters is essential for achieving optimal policy performance, ensuring stability during training, and enhancing the agent's ability to generalize across different market conditions. This section outlines the hyperparameters considered in this study, the strategies employed for their selection, and the rationale behind the chosen configurations.

The PPO and A2C algorithms encompass several hyperparameters that influence their behavior and performance. The key hyperparameters examined in this study include

3.4.1 Learning Rate (alpha)

Determines the step size at each iteration while moving toward a minimum of the loss function. It affects how quickly the Actor and Critic networks update their weights. A higher learning rate can accelerate training but may lead to instability, while a lower rate ensures more stable convergence but may prolong training time. Set at 0.0003, and the learning rate dictates the magnitude of updates to the neural network weights during training. A moderate learning rate ensures the agent learns efficiently without overshooting optimal policy parameters, maintaining stability throughout training.

3.4.2 Batch Size

The number of samples processed before the model is updated influences the granularity of updates. Smaller batch sizes can lead to more frequent updates and potentially faster learning, whereas larger batches provide more stable gradient estimates. With a batch size of 64, the agent processes 64 experiences during each training iteration. This size balances computational efficiency and the stability of gradient estimates, facilitating effective learning without incurring excessive computational costs.

3.4.3 Number of Epochs

Number of times the entire training dataset is passed through the model during training. Determines the extent of learning from each batch of data. More epochs can enhance learning but risk overfitting, especially in dynamic environments like financial markets. Ten epochs per training cycle allow the agent to iteratively refine its policy and value estimates based on the sampled experiences. This number ensures sufficient learning from each batch while preventing overfitting to specific subsets of data.

3.4.4 Gamma Discount factor (γ) for future rewards

Balances the importance of immediate versus future rewards. A higher gamma emphasizes long-term rewards, which is critical for trading strategies aimed at long-term profitability. The discount factor is set to 0.99, prioritizing long-term rewards and encouraging the agent to develop strategies that yield sustained profitability.

3.4.5 GAE Lambda (λ)

The weighting factor for Generalized Advantage Estimation (GAE). Controls the bias-variance trade-off in advantage estimation. It affects how much future rewards are considered in the advantage calculation. The Generalized Advantage Estimation parameter is set to 0.95, balancing the trade-off between bias and variance in the advantage calculations. Together, these hyperparameters ensure that the agent effectively evaluates the long-term benefits of its actions while maintaining robust learning dynamics.

3.4.6 Policy Clip

A clipping parameter 0.2 restricts the policy updates within a predefined range, preventing drastic changes that could destabilize the learning process. This mechanism ensures that the agent's policy evolves smoothly, maintaining consistency and reliability in its trading strategies.

3.4.7 Memory Buffer

The memory buffer stores up to 2000 transitions, allowing the agent to sample diverse experiences and break the correlation between consecutive data points. This approach enhances the generalization capabilities of the agent, ensuring that it can adapt to varying market conditions without being biased by specific patterns in the training data.

The Deep Q-Network algorithm relies on a different set of hyperparameters tailored to value-based learning. The key hyperparameters for DQN in this study include:

3.4.8 Epsilon (ε)

Epsilon represents the initial exploration rate in the ε-greedy policy employed by the DQN agent. This parameter dictates the probability with which the agent will choose a random action (exploration) as opposed to selecting the best-known action (exploitation) based on its current Q-value estimates. Setting ε to 1.0 at the outset ensures that the agent engages in full exploration. This high exploration rate is crucial during the initial training phases, allowing the agent to gather a diverse set of experiences across the state-action space. By exploring extensively, the agent can discover a wide range of potential strategies, which is essential for effective learning in complex environments where optimal actions are not immediately apparent.

3.4.9 Epsilon End

Epsilon End defines the minimum exploration rate that ε will decay to over the course of training. This parameter ensures that the agent retains a residual probability of exploration even after extensive training. By setting 0.01, the agent maintains a small but non-negligible level of exploration throughout its learning process. This residual exploration is vital for preventing the agent from becoming trapped in local optima or suboptimal policies. It allows the agent to occasionally explore new actions that may yield better long-term rewards, thereby enhancing the robustness and adaptability of the learned policy.

3.4.10 Memory Size

Memory Size specifies the maximum number of experiences (transitions) the replay buffer can store. The replay buffer is a critical component in DQN, enabling the agent to sample past experiences for training. A substantial memory size of 100,000 transitions ensures that the agent can access a comprehensive and diverse set of experiences. This diversity is essential for breaking the temporal correlations between consecutive data points, which can otherwise lead to inefficient learning and instability in the training process. By maintaining a large and varied replay buffer, the agent can generalize better across different states and actions, enhancing its ability to perform effectively in diverse and dynamic market conditions.

3.4.11 Epsilon Decay

Epsilon Decay determines how the exploration rate ε decreases over time. This parameter controls how quickly the agent transitions from exploration to exploitation. A gradual decay rate of 1e-5 ensures a smooth and controlled reduction in ε. This slow decay allows the agent to continue exploring sufficiently during the early and middle stages of training while progressively shifting towards exploitation as it gains more experience and refines its policy. Such a controlled decay helps in balancing the exploration-exploitation trade-off, enabling the agent to exploit learned strategies effectively without prematurely abandoning the exploration of potentially better actions.

3.4.12 Target Network Update Frequency (replace)

Target Network Update Frequency dictates the number of training steps after which the target network is synchronized with the main Q-network. In DQN, a separate target network is used to stabilize training by providing consistent Q-value targets. Updating the target network every 1,000 steps strike an optimal balance between stability and responsiveness. Frequent updates can lead to oscillations and instability in Q-value estimates, as the target network rapidly changes in response to the main network's updates. Conversely, infrequent updates may slow down the learning process, causing the agent to rely on outdated targets. By setting the update frequency to 1000 steps, the training process benefits from stable and consistent target values, which facilitates more reliable convergence of the Q-network while still allowing it to adapt to new information at a reasonable pace.

3.5 Actor and Critic Network

The reinforcement learning frameworks employed in this study, Proximal Policy Optimization (PPO) and Advantage Actor-Critic (A2C), are meticulously designed to navigate the high-dimensional state space characteristic of financial markets. PPO is a state-of-the-art reinforcement learning algorithm that balances performance and computational efficiency, making it well-suited for complex trading environments. This section delves into the architecture of the Actor and Critic networks, detailing their components and the rationale behind their design choices.:

3.5.1 Input Layer:

The state input includes historical stock prices (Open, High, Low, Close), VWAP, Close Price, and other features.

Seven input features represent each step in the environment at each time.

3.5.2 Hidden Layers (Actor and Critic):

The Actor and Critic networks are constructed with two fully connected hidden layers, each comprising 256 neurons. These hidden layers utilize the Rectified Linear Unit (ReLU) activation function, which introduces non-linearity into the model, enabling it to capture complex patterns and dependencies within the financial data. The depth and width of the hidden layers provide sufficient capacity to model the intricate relationships inherent in stock market dynamics without leading to overfitting. By employing ReLU activations, the networks benefit from faster convergence during training and mitigate issues such as vanishing gradients, thereby enhancing the overall stability and performance of the PPO algorithm in optimizing trading strategies.

3.5.3 Output Layer (Actor):

The Actor network concludes with an output layer consisting of n_actions neurons, where n_actions represents the number of possible actions the agent can take (e.g., Buy, Sell). This output layer employs the Softmax activation function, which transforms the raw outputs into a probability distribution over the available actions. By generating a probability distribution, the Actor-network facilitates stochastic policy updates, allowing the agent to explore various trading actions probabilistically. This probabilistic approach promotes exploration, enabling the agent to discover potentially profitable strategies that may not be immediately apparent. The Softmax activation ensures that the probabilities are normalized and sum to one. It is crucial for the policy gradient methods used in PPO to update the policy effectively based on the expected rewards.

3.5.4 Output Layer (Critic):

In contrast to the Actor-network, the Critic network's output layer is designed to produce a single scalar value, representing the estimated value of the current state. This output is achieved through a linear activation function, allowing the Critic to provide an unbounded estimate of the expected cumulative reward from the current state onward. The Critic network plays a pivotal role in the PPO framework by offering a baseline for advantage estimation, essential for reducing the variance of policy gradient updates. By accurately estimating the state value, the Critic helps the Actor-network discern which actions are genuinely advantageous, guiding the policy updates toward actions that maximize long-term profitability while maintaining stability in the learning process.

3.5.5 Integration within PPO and A2C Frameworks

Both PPO and A2C utilize the Actor and Critic architectures but differ in their optimization and training methodologies.

3.5.5.1 PPO Framework: employs a clipped surrogate objective to ensure that policy updates do not deviate excessively from the current policy, enhancing training stability. the Actor and Critic networks are updated simultaneously using collected experiences, with the Critic providing value estimates that inform the Actor's policy adjustments.

3.5.5.2 A2C Framework: A2C typically operates synchronously, where multiple agents interact with the environment in parallel, and gradients from these agents are aggregated to update the networks. it uses advantage estimates (the difference between the observed rewards and the Critic's value estimates) to update the Actor, encouraging actions that lead to higher-than-expected rewards.

This architecture corresponds to the Actor and Critic networks, which are meticulously architected to function cohesively within the PPO framework and A2C framework. With its two hidden layers and Softmax output, the Actor-network generates a probability distribution over possible trading actions, promoting exploration and strategic decision-making. Conversely, the Critic network, comprising two hidden layers but with a linear output, provides accurate state value estimations that serve as a baseline for advantage calculations. Together, these networks enable the PPO algorithm to refine the trading policy iteratively, balancing the pursuit of high returns with risk management and ensuring robust performance in dynamic market conditions.

3.6 Deep Q-Network (DQN) Implementation

The DQN model implemented in this study serves as a foundational reinforcement learning algorithm. This section provides a comprehensive overview of the DQN architecture, elucidating its core components, including the Replay Buffer, Neural Network architecture, and the DQN Agent, along with the rationale behind each design choice.

3.6.1 Replay Buffer

The Replay Buffer is a critical component in DQN architecture. It facilitates the storage and sampling of experiences to stabilize and improve the learning process.

It maintains a fixed-size memory to store tuples of experiences, each consisting of the current state, action taken, reward received, next state, and a terminal flag indicating the end of an episode. By employing a circular buffer mechanism, the buffer efficiently manages memory usage, overwriting the oldest experiences when the buffer is full. The Random Sampling method enables the agent to randomly sample a batch of experiences, breaking the temporal correlations between consecutive samples and promoting more stable and diverse training data.

3.6.2 Deep Q-Network Architecture

3.6.2.1 Input Layer receives the state representation, which includes historical stock prices (Open, High, Low, Close), VWAP, and other relevant features. Each state is represented by a vector with a dimensionality corresponding to the number of input features.

3.6.2.2 Hidden Layers The network comprises two fully connected (dense) hidden layers, each with 256 neurons. These layers employ the ReLU activation function to introduce non-linearity, enabling the network to capture complex patterns and relationships within the financial data. The depth and width of the hidden layers are chosen to provide sufficient capacity for modeling intricate market dynamics without incurring excessive computational costs or risking overfitting.

3.6.2.3 Action-Value Outputs consist of n_actions neurons, each corresponding to the Q-value of a possible action (e.g., buy, sell, hold). A linear activation function produces unbounded Q-value estimates, which are essential for accurately assessing the expected cumulative rewards of actions.

3.7 Model Training

3.7.1 Proximal Policy Optimization

The PPO model is trained within a trading environment defined by several key parameters and constraints. The initial account balance is 2,000 Baht, which the agent uses to trade over the simulation period. The agent can invest 10% of its capital per trade to limit risk and manage capital allocation. A trading cost rate of 0.001 is applied, simulating real-world transaction fees that reduce profitability if not appropriately handled. The agent can take a maximum of 2,000 trades throughout the simulation, restricting one open position at a time and forcing the agent to carefully decide whether to buy, sell, or hold in each step.

The batch size for training is 64, meaning that the model is updated based on mini batches of 64 experiences sampled from the experience replay buffer, which stores the last 2000 transitions. This buffer helps break the correlation between consecutive experiences and stabilizes learning.

The agent is trained in over 100 episodes (n_games = 100), each representing a complete trading period based on the historical dataset. A termination threshold (KILL_THRESH) is set, where the environment ends if the agent's balance falls below 40% of the initial account balance, adding a layer of risk management. The agent also faces a lag of 20-time steps, simulating a delay in market responses, while the market volatility is initialized to 1 and dynamically updated based on observed price movements.

3.7.2 Advantage Actor-Critic

The A2C (Advantage Actor-Critic) model is trained within the same trading environment as the PPO model, adhering to the predefined parameters and constraints to simulate realistic trading conditions. The initial account balance is 2,000 Baht, and the agent can invest up to 10% of its capital per trade. A trading cost rate of 0.001 is applied to each transaction, mimicking real-world fees that can impact profitability if not managed effectively. The agent is limited to a maximum of 2,000 trades throughout the simulation and is restricted to one open position at a time, necessitating strategic decision-making in buying, selling, or holding positions.

The agent is trained over 100 episodes (n_games = 100), with each episode representing a complete trading period derived from the historical dataset. A termination threshold (KILL_THRESH) is set, where the environment concludes if the agent's balance falls below 40% of the initial account balance, reinforcing risk management practices. The agent experiences a lag of 20-time steps to simulate delays in market responses, and the market volatility is initialized to 1, dynamically updating based on observed price movements.

In the A2C training process, learning updates occur every N = 10 steps within an episode, meaning the agent updates its neural networks after every ten actions. The batch size for training is set to 5, and the agent undergoes ten epochs of learning during each update phase. The learning rate (alpha) is initialized at 0.0003, influencing the step size during optimization.

The A2C agent utilizes both an actor-network, which is responsible for determining the optimal actions, and a critic network, which evaluates the value of the current state. The agent stores experiences, including states, actions, probabilities, values, rewards, and done flags, in a memory buffer throughout each episode. This memory enables the agent to calculate the GAE for more stable and efficient learning. At the end of each episode, the agent performs an additional learning phase to update the networks based on the accumulated experiences, refining its policy and value estimations.

3.7.3 The Deep Q-Network

The model is trained in the same trading environment, maintaining consistency in simulation parameters to ensure comparability across different algorithms. The initial account balance is 2,000 Baht, with the agent allowed to invest up to 10% of its capital per trade. A trading cost rate of 0.001 is implemented, and the agent is limited to a maximum of 2,000 trades, holding only one open position at any given time to enforce disciplined trading behavior.

The DQN agent is trained in over 250 episodes (n_games = 250), each representing an entire trading period based on historical data. A termination threshold (KILL_THRESH) is applied, ending the episode if the agent's balance drops below 40% of the initial amount, thus embedding risk management into the training process. The agent also encounters a lag of 20-time steps to simulate realistic market response delays, with market volatility initialized at one and adjusted dynamically according to price fluctuations.

In the DQN training framework, the agent employs an experienced replay buffer with a capacity of 100,000 transitions. This buffer allows the agent to learn from a diverse set of past experiences, breaking the correlation between sequential data and enhancing learning stability. The batch size for training is set to 64, and the learning rate is 0.0003, which dictates the speed at which the agent updates its Q-network.

The agent begins with an epsilon value of 1.0 for its epsilon-greedy policy, promoting exploration of the action space. This epsilon value decays over time (eps_dec = 1e-5) to a minimum threshold, gradually shifting the agent's focus from exploration to exploitation of learned strategies. The target network is updated every 1,000 steps (replace = 1,000) to provide a stable target for Q-value predictions, enhancing the convergence of the learning process.

During training, the agent selects actions based on either exploration or exploitation and stores each transition—including the current state, action taken, reward received, next state, and done flag in the replay buffer. The learning process involves sampling mini-batches from the replay buffer to update the Q network. The agent minimizes the loss between the predicted and target Q-values, calculated using the Bellman equation through backpropagation and gradient descent optimization techniques.

3.8 Testing and Evaluation

The model was tested on a separate, unseen dataset to evaluate its performance in a realistic trading environment. The testing phase is the final step in determining how well the model generalizes to new market conditions and stock price movements that were not part of the training data. The testing period consists of the final 30% of the entire dataset, corresponding to the most recent market activity.

3.8.1 Testing Process

During testing, the agent made trading decisions based on the stock's features, including Open, High, Low, Close, Volume, and the engineered VWAP. These decisions were guided by the learned policy from the training phase but with no further updates to the model's parameters (i.e., no learning occurs during testing). The agent aimed to maximize the ROI based on time-series market conditions observed during this period.

The model was initialized for each stock with a starting capital of 2,000 Baht, and 10% of the capital was allocated per trade. The agent could open only one position at a time, either long or short, and was subject to a small trading cost of 0.1% per transaction. Throughout the testing period, the agent interacted with the market data and executed trades based on its learned strategy.

For executing trades, models made decisions at each time step using the test data. The A2C agent mentions how action probabilities were computed, and actions were selected. The DQN agent clarifies how actions were chosen based on Q-values.

3.8.2 Evaluation

In the evaluation phase, the trading models' performance was rigorously assessed in a realistic trading environment using two primary metrics for each stock: Return on Investment (ROI) and additional indicators such as the Sharpe Ratio, Maximum Drawdown, Sortino Ratio, and Calmar Ratio. ROI measures the profitability of the agent's trading decisions relative to the initial capital, accounting for all executed buy and sell actions during the testing period. The supplementary metrics provide insights into risk-adjusted returns, volatility, and potential losses, ensuring a comprehensive evaluation of the models' robustness and reliability. To contextualize their effectiveness, the models' performance was compared with existing literature, positioning the findings within the broader field of algorithmic trading research and highlighting relative strengths and areas for improvement.

With the methodology in place, including data preprocessing, environment configuration, and model optimization, the DRL agents are now ready for testing in simulated trading scenarios. Chapter 4 presents the results of these experiments, providing insights into the comparative performance of each model and their practical implications for portfolio management in volatile markets.

CHAPTER 4
RESULTS

Chapter 4 presents and analyzes the outcomes of the DRL models introduced in the previous chapter, focusing on their effectiveness in managing portfolios within the SET. This chapter compares the three algorithms (PPO, A2C, and DQN), using metrics such as ROI, Sharpe Ratio, and maximum drawdown to assess performance across different market conditions. By evaluating both profitability and risk management, explore the strengths and weaknesses of each model, highlighting specific scenarios where certain algorithms outperform others. The insights derived from this analysis validate the methodology and offer practical implications for DRL-based portfolio management in volatile financial environments.

4.1 Proximal Policy Optimization

4.1.1 Portfolio result

To evaluate the effectiveness of the PPO trading agent, the performance across a diversified portfolio of ten different stocks. Table 4-1 summarizes the key performance metrics for each stock during the testing period. This analysis provides insights into how well the agent managed risk and generated returns in various market conditions

TABLE 4-1 The portfolio (test model) of PPO

Stock	ROI (%)	Sharpe Ratio (%)	Maximum Drawdown (%)	Calmar Ratio (%)
ADVANC	1.47	0.42	-1.82	0.44
INTUCH	1.80	0.44	-2.68	0.36
PTTEP	3.21	0.86	-1.63	1.06
BDMS	1.76	0.73	-1.29	0.74
MINT	1.91	0.40	-3.88	0.27
CPN	-3.18	**-0.73**	**-5.24**	**-0.33**
AOT	1.45	0.53	**-1.30**	0.60
TISCO	1.01	0.39	-1.84	0.30
SCC	0.11	0.06	-1.11	0.05
IVL	**5.92**	**1.27**	-1.91	**1.67**
Cumulative Return	**15.46**			

Table 4-1 highlights the diverse performance of the PPO trading agent across ten different stocks during the testing period. The agent achieved notable success with IVL and PTTEP, recording the highest ROIs of 5.92% and 3.21%, respectively, accompanied by strong Sharpe and Calmar Ratios, which indicate adequate risk-adjusted returns and robust risk management. BDMS and MINT also delivered positive ROIs of 1.76% and 1.91%, though MINT faced a higher maximum drawdown, suggesting increased risk exposure. In contrast, CPN significantly underperformed with a negative ROI of -3.18%, reflecting poor risk-adjusted performance and substantial losses. AOT, INTUCH, ADVANC, TISCO, and SCC exhibited modest to low ROIs, ranging from 0.11% to 1.80%, which may indicate conservative trading strategies or missed profitable opportunities. While the cumulative return across all stocks was 15.46%, the varying results underscore the agent's strengths in certain stocks like IVL and PTTEP, highlighting areas for improvement in managing risk and enhancing performance for underperforming stocks such as CPN and SCC. Overall, the PPO agent demonstrates potential with selective stocks, but consistency and strategy refinement are necessary to optimize its performance across the entire portfolio.

4.1.2 Agent's Trading Behavior

To better understand the agent's decision-making process, we plotted the agent's trading behavior during the testing period. The charts below show the agent's buy and sell signals, along with the stock's price movements.

Figure 4-1 presents the price trends of ADVANC and INTUCH stocks during both testing periods and the agent's buy and sell signals. The agent's trading behavior for these telecommunications companies reflects its responsiveness to market fluctuations. For ADVANC, the agent capitalized on short-term price movements, contributing to a modest ROI of 1.47%. Similarly, for INTUCH, the agent's timely trades resulted in a slightly higher ROI of 1.80%. The frequent adjustments in trading positions indicate an adaptive strategy to optimize returns in a relatively stable market sector.

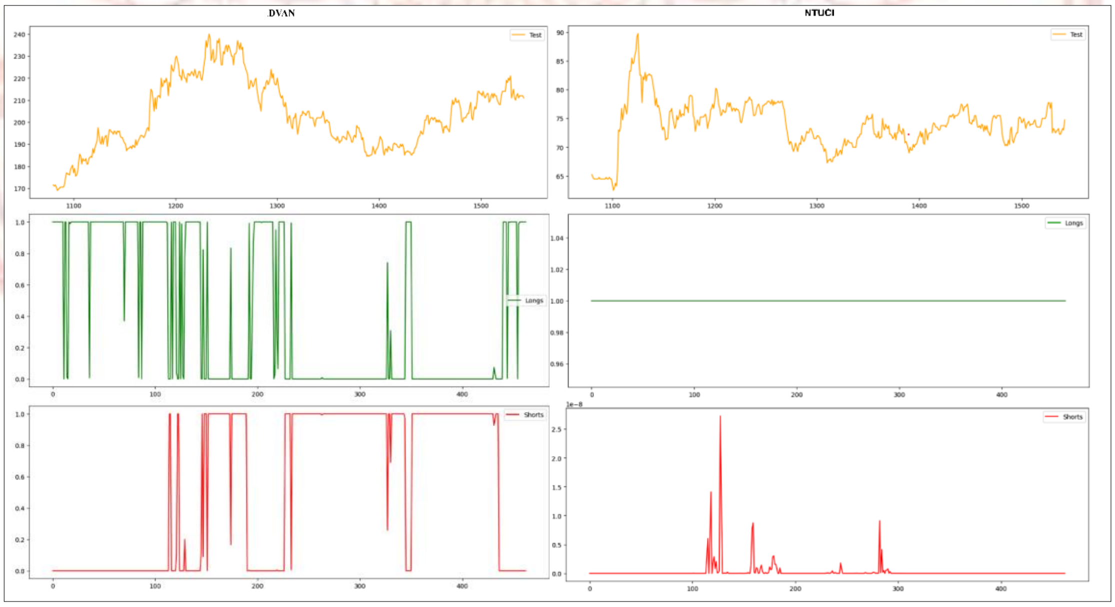


FIGURE 4-1 The price trend of the ADVANC and INTUCH of PPO

Figure 4-2 showcases the price trends of PTTEP and BDMS stocks, including the agent's buy and sell signals throughout the testing periods. PTTEP, an energy sector stock, yielded a significant ROI of 3.21%, suggesting that the agent effectively leveraged price volatility in the energy market. The trading signals for PTTEP show well-timed entries and exits that maximized gains. For BDMS, a healthcare stock, the agent achieved a positive ROI of 1.76%. The agent's trading decisions for BDMS reflect a balance between capturing growth opportunities and managing risk in a defensive sector.

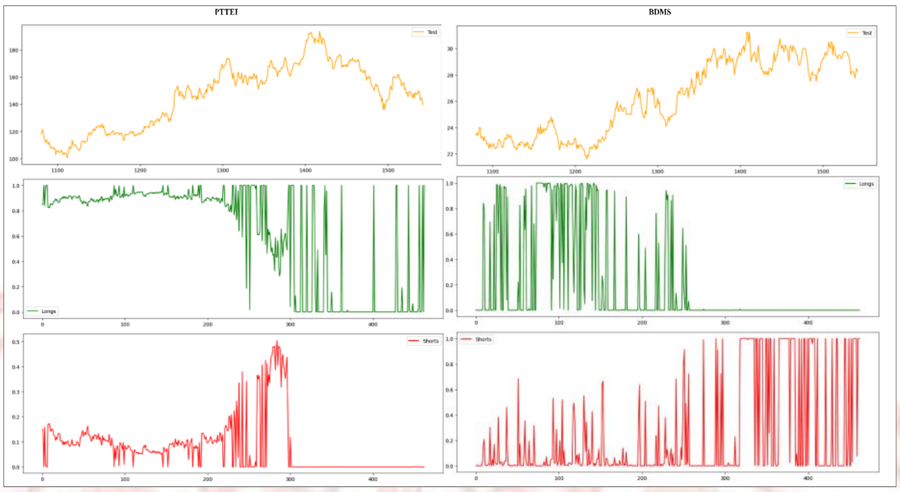


FIGURE 4-2 The price trend of the PTTEP and BDMS of PPO

Figure 4-3 illustrates the price trends of MINT and CPN stocks and the agent's buy and sell signals during the testing periods. MINT, operating in the hospitality industry, provided an ROI of 1.91% despite higher market volatility, as indicated by a maximum drawdown of -3.88%. The agent's trading behavior for MINT shows attempts to exploit short-term uptrends while mitigating losses. In contrast, CPN, a retail property development stock, underperformed with a negative ROI of -3.18%. The agent's trading signals for CPN reveal challenges in adjusting to adverse market conditions, highlighting an area for strategy improvement.

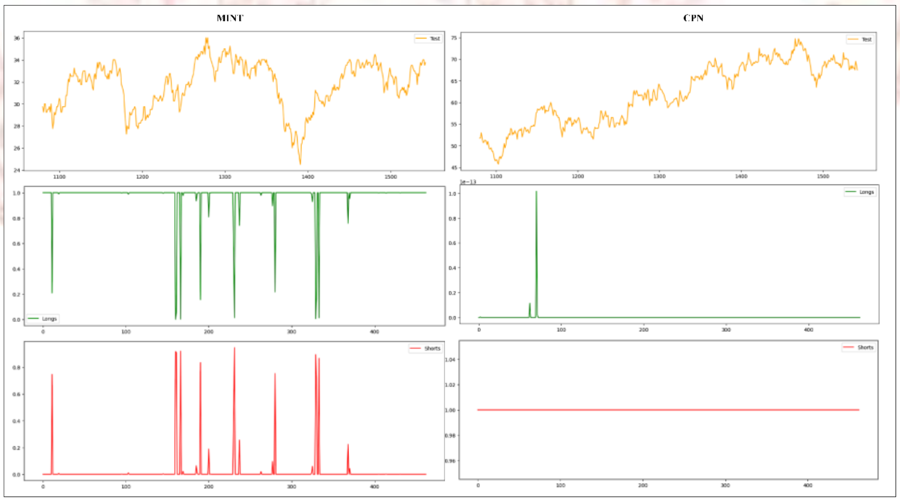


FIGURE 4-3 The price trend of the MINT and CPN of PPO

Figure 4-4 displays the price trends of AOT and TISCO stocks, accompanied by the agent's buy and sell signals during testing periods. The agent secured a modest ROI of 1.45% for AOT, an airport services company, indicating cautious trading in a sector sensitive to global travel trends. The trading signals suggest the agent made conservative decisions to protect against downside risks. TISCO, a financial services firm, yielded an ROI of 1.01%. The agent's trading behavior for TISCO reflects a careful approach in a sector often influenced by economic indicators and regulatory changes.

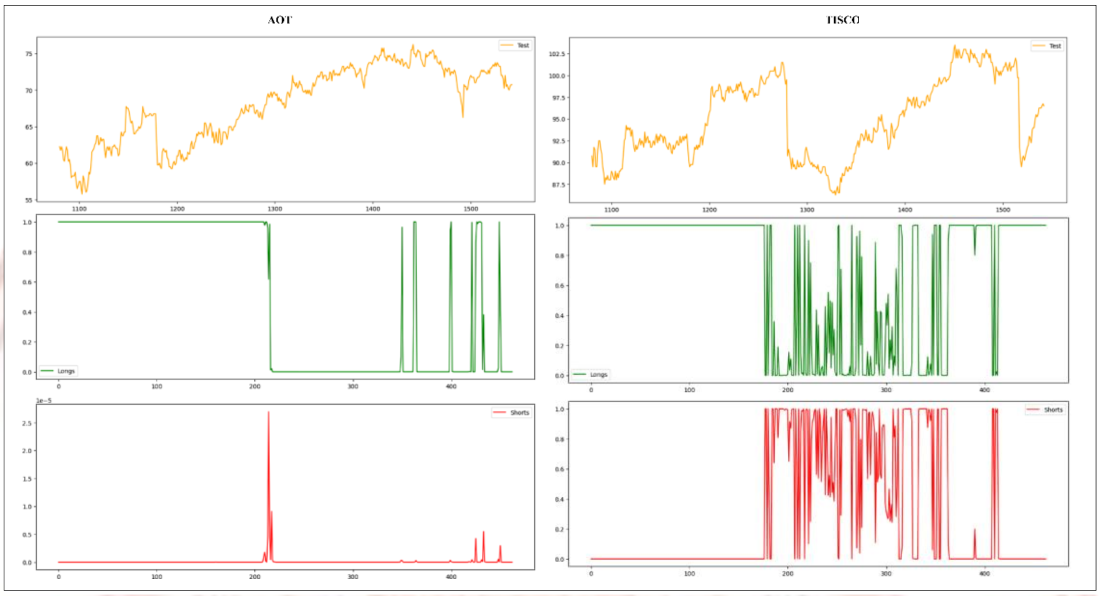


FIGURE 4-4 The price trend of the AOT and TISCO of PPO

Figure 4-5 presents the price trends of SCC and IVL stocks and the agent's buy and sell signals during the testing periods. IVL, a chemical production company, delivered the highest ROI of 5.92%, demonstrating the agent's proficiency in capitalizing on favorable market conditions within the materials sector. The well-timed trades for IVL indicate strong market trend identification and execution by the agent. Conversely, SCC showed a negligible ROI of 0.11%, suggesting that the agent's strategy was less effective for this stock. The trading signals for SCC point to a more conservative approach, potentially missing out on profitable opportunities.

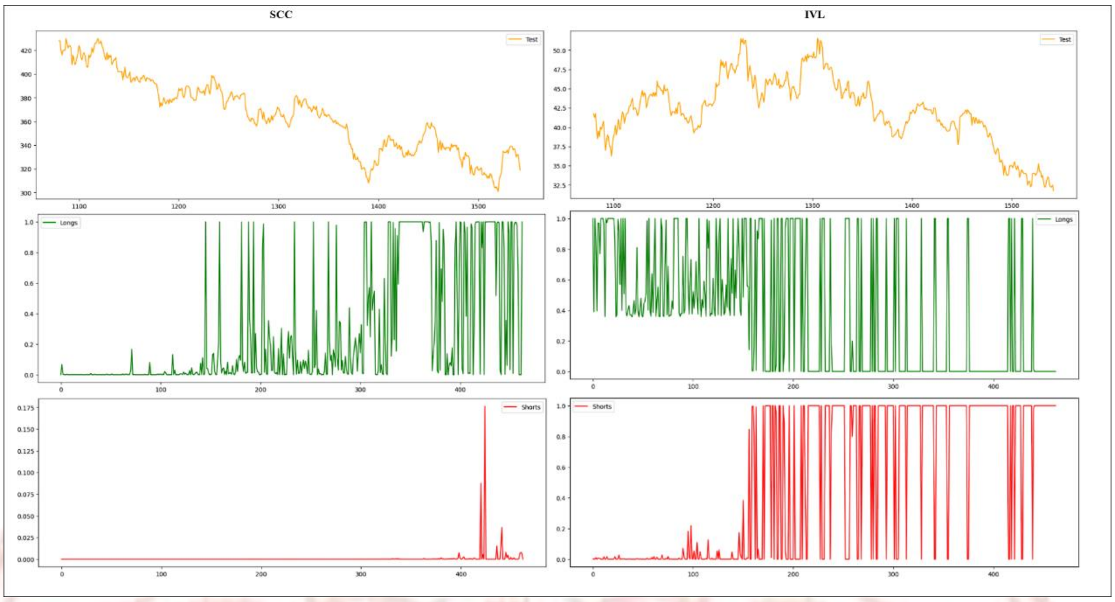


FIGURE 4-5 The price trend of the SCC and IVL of PPO

4.2 Advantage Actor-Critic

4.2.1 Portfolio result

To comprehensively evaluate the effectiveness of the Advantage Actor-Critic trading agent, tested its performance across a diversified portfolio of ten different stocks. Table 4-2 summarizes the key performance metrics for each stock during the testing period. This detailed analysis provides insights into how well the agent managed risk and generated returns under various market conditions.

TABLE 4-2 The portfolio (test model) of A2C

Stock	ROI (%)	Sharpe Ratio (%)	Maximum Drawdown (%)	Calmar Ratio (%)
ADVANC	0	0	0	0
INTUCH	1.80	0.44	-2.68	0.36
PTTEP	2.32	0.47	-3.38	0.37
BDMS	-0.01	-1.04	**-0.02**	-0.22
MINT	1.85	0.39	-3.63	0.28
CPN	**3.17**	**0.72**	-1.60	**1.07**
AOT	0	0	0	0
TISCO	0	0	0	0
SCC	0	0	0	0
IVL	**-2.04**	-0.38	-4.46	**-0.25**
Cumulative Return	**7.09**			

Table 4-2 highlights the performance of the A2C trading agent across ten different stocks during the testing period. The agent demonstrated varying degrees of success, with notable achievements and areas needing improvement. CPN achieved the highest ROI of 3.17%, accompanied by a robust Sharpe Ratio of 0.72 and a Calmar Ratio of 1.07. These figures indicate that the agent generated substantial returns and managed risk effectively, resulting in robust risk-adjusted performance. The lower Maximum Drawdown of -1.60% suggests that the agent successfully limited losses during unfavorable market movements. PTTEP and MINT also performed well. PTTEP recorded an ROI of 2.32% with a Sharpe Ratio of 0.47, while MINT had an ROI of 1.85% and a Sharpe Ratio of 0.39. These Sharpe Ratios suggest moderate risk-adjusted returns, indicating that the agent managed to balance risk and reward adequately for these stocks. However, their Maximum Drawdowns of -3.38% and -3.63%, respectively, imply that there were periods of significant decline, which the agent navigated to still achieve positive returns. INTUCH delivered a positive ROI of 1.80% with a Sharpe Ratio of 0.44.

While the ROI reflects a profitable outcome, the moderate Sharpe Ratio indicates that there is room for improvement in risk management to enhance risk-adjusted returns. Stocks like ADVANC AOT, TISCO, and SCC showed no change in ROI, each recording 0%. Their Sharpe Ratios and Calmar Ratios were also 0%, suggesting minimal trading activity or that the agent maintained a neutral position throughout the testing period. This lack of activity could be due to the agent not identifying profitable trading opportunities or choosing to hold positions without executing trades. On the downside, BDMS and IVL underperformed. BDMS had a negligible negative ROI of -0.01%, with a Sharpe Ratio of -1.04 and a Calmar Ratio of -0.22. These negative ratios indicate poor risk-adjusted returns and suggest that the agent's strategy did not align well with this stock's market behavior. IVL experienced a more significant negative ROI of -2.04%, with a Sharpe Ratio of -0.38 and a Calmar Ratio of -0.25, highlighting challenges in risk management and loss mitigation for this stock.

The cumulative return across all stocks was 7.09%, reflecting the overall performance of the A2C agent. While this indicates a positive return on the portfolio level, the disparities among individual stocks suggest that the agent excelled in certain areas but struggled with consistency across the portfolio. The success with stocks like CPN, PTTEP, and MINT showcases the agent's potential, whereas the lackluster performance with other stocks points to areas where the agent's strategy could be refined.

4.2.2 Agent's Trading Behavior

Figures 4-6 to 4-9 present the price trends of selected stocks during the testing periods, along with the agent's buy and sell signals. These visual representations provide deeper insights into the agent's decision-making process and its responsiveness to market dynamics.

Figure 4-6 illustrates the price trends and trading signals for INTUCH and PTTEP. The agent's buy and sell decisions corresponded with favorable market movements, enabling it to capture gains during upward trends. The timing of these trades suggests that the agent effectively identified trading opportunities in these stocks.

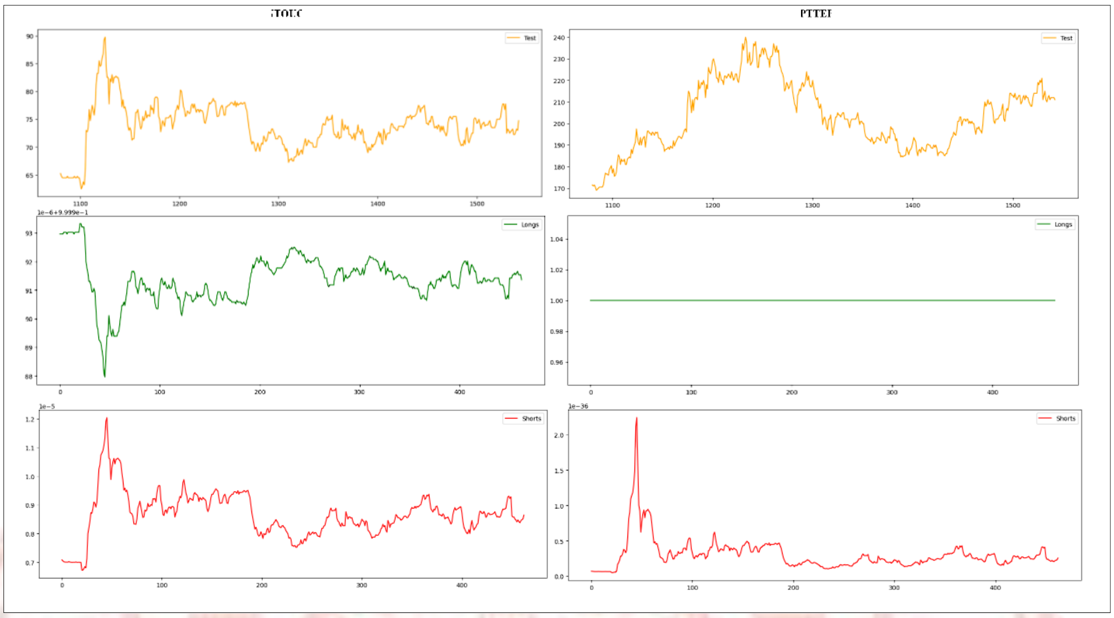


FIGURE 4-6 The price trend of the INTUCH and PTTEP of A2C

Figure 4-7 displays the price trends and agent signals for BDMS and CPN. While the agent achieved significant success with CPN, reflected in its high ROI and risk-adjusted metrics, its performance with BDMS was suboptimal. The negative ROI for BDMS indicates that the agent's trading signals did not align well with the stock's price movements, leading to poor outcomes.

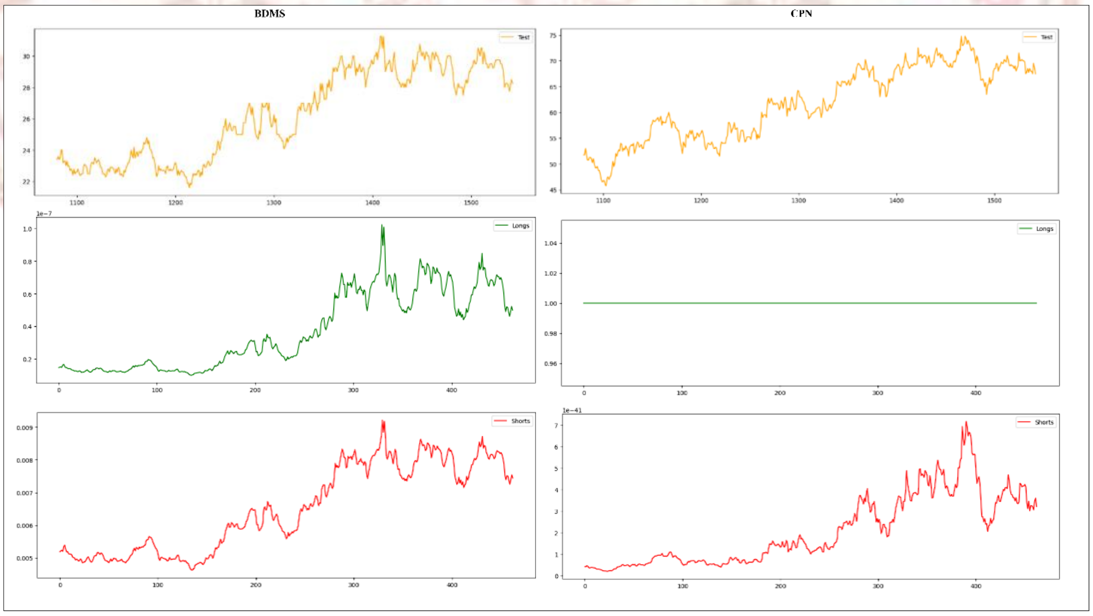


FIGURE 4-7 The price trend of the BDMS and CPN of A2C

Figure 4-8 focuses on IVL, where the agent's trading decisions resulted in a negative ROI. The buy and sell signals suggest that the agent may have misinterpreted market indicators or failed to adapt to this stock's volatility, highlighting a need for strategy refinement.

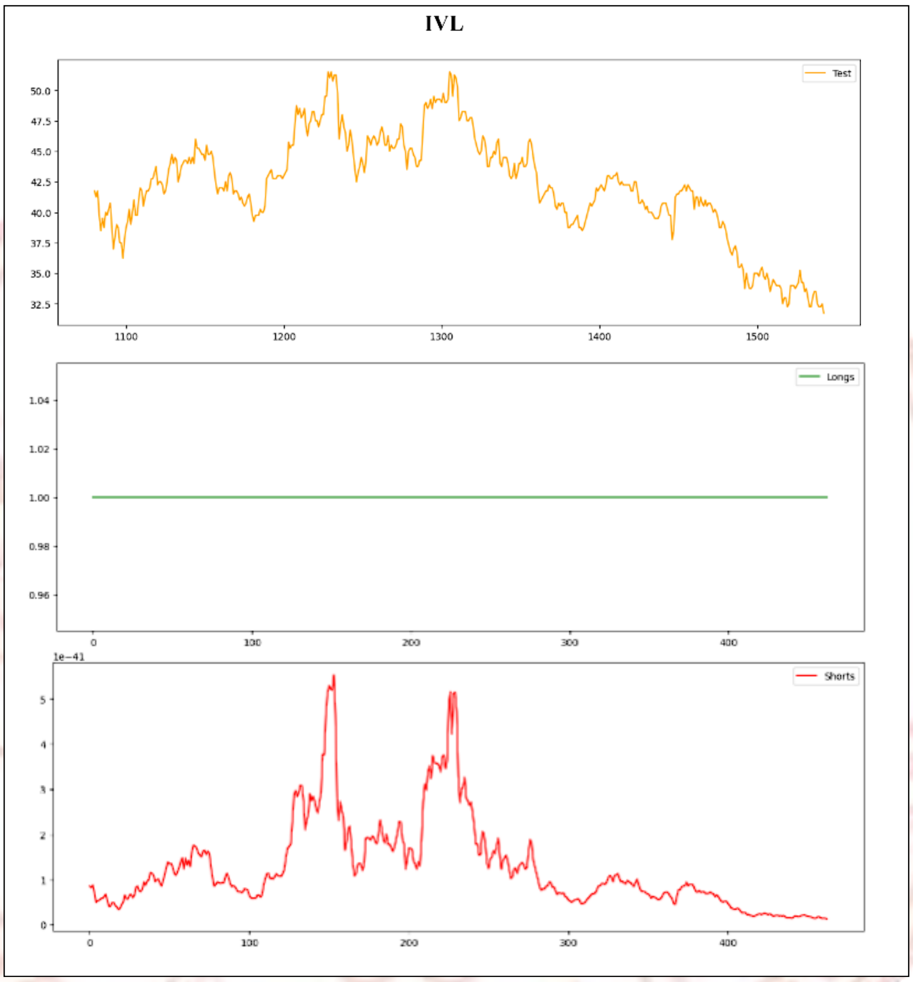


FIGURE 4-8 The price trend of the IVL of A2C

Figure 4-9 presents the price trends of the other stocks ADVANC, AOT, TISCO, and SCC along with the agent's buy and sell signals. The ROI for these stocks remained at 0%, indicating that the agent either did not execute any trades or consistently held positions without realizing gains or losses. This inactivity might be due to the agent not detecting sufficient trading opportunities or choosing to maintain a neutral stance in uncertain market conditions.

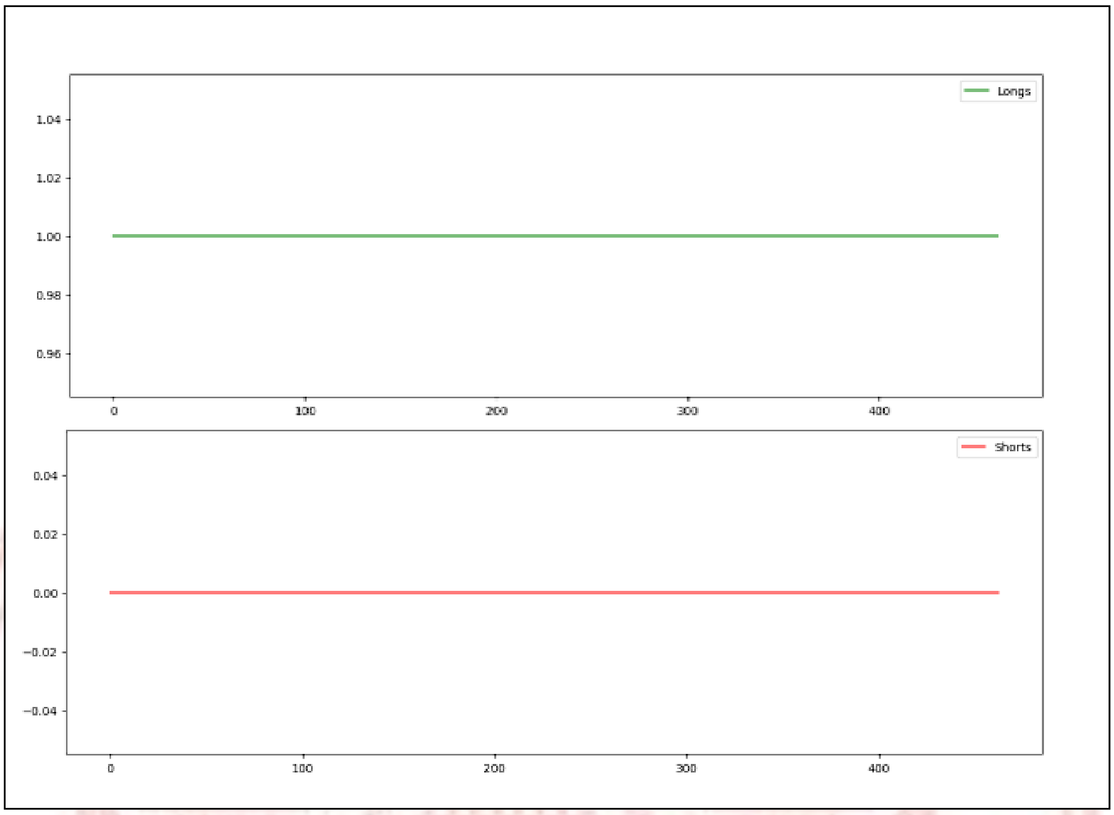


FIGURE 4-9 The price trend of ADVANC, AOT, TISCO, and SCC of A2C

4.3 Deep Q-Network

4.3.1 Portfolio result

To comprehensively evaluate the effectiveness of the DQN trading agent. Table 4-3 highlights the performance of ten different stocks during the testing period.

TABLE 4-3 The portfolio (test model) of DQN

Stock	ROI (%)	Sharpe Ratio (%)	Maximum Drawdown (%)	Calmar Ratio (%)
ADVANC	**4.14**	1.22	-1.45	1.69
INTUCH	-0.68	-0.17	-3.31	-0.12
PTTEP	-0.31	-0.06	**-5.45**	-0.03
BDMS	0.08	0.03	-2.39	0.02
MINT	0.42	0.11	-4.44	0.06
CPN	2.32	0.59	-2.88	0.48
AOT	1.12	0.37	-2.01	0.33
TISCO	3.55	**1.53**	**-0.72**	**2.91**
SCC	**-1.28**	**-0.50**	-2.71	-0.28
IVL	3.43	0.76	-2.17	0.94
Cumulative Return	12.79			

The agent demonstrated varying degrees of success, with notable achievements and areas needing improvement. This detailed analysis provides insights into how well the agent managed risk and generated returns under various market conditions.

ADVANC achieved the highest ROI of 4.14%, accompanied by a robust Sharpe Ratio of 1.22 and a Calmar Ratio of 1.69. These figures indicate that the agent generated substantial returns and managed risk effectively, resulting in robust risk-adjusted performance. The low Maximum Drawdown of -1.45% suggests effective loss limitation during unfavorable market movements. also delivered impressive results, recording an ROI of 3.55%, the highest Sharpe Ratio of 1.53, and a Calmar Ratio of 2.91. The minimal Maximum Drawdown of -0.72% reflects exceptional risk management and stability throughout the trading period. IVL showed significant gains with an ROI of 3.43%, a Sharpe Ratio of 0.76, and a Calmar Ratio of 0.94. Despite a Maximum Drawdown of -2.17%, the agent effectively navigated market fluctuations to achieve positive returns. CPN achieved an ROI of 2.32%, with a Sharpe Ratio of 0.59 and a Calmar Ratio of 0.48. These metrics suggest moderate risk-adjusted returns, indicating a balanced approach to risk and a reward for this stock.

AOT and MINT also contributed positively to the portfolio. AOT recorded an ROI of 1.12% with a Sharpe Ratio of 0.37, while MINT had an ROI of 0.42% and a Sharpe Ratio of 0.11. Their Maximum Drawdowns of -2.01% and -4.44%, respectively, imply periods of significant decline that the agent managed to overcome for overall gains. On the downside, SCC underperformed with a negative ROI of -1.28%, a Sharpe Ratio of -0.50, and a Calmar Ratio of -0.28. These negative ratios indicate poor risk-adjusted returns and suggest that the agent's strategy did not align well with this stock's market behavior. INTUCH and PTTEP also recorded negative ROIs of -0.68% and -0.31%, respectively, with corresponding negative Sharpe and Calmar Ratios, highlighting challenges in risk management and loss mitigation for these stocks.

BDMS (Bangkok Dusit Medical Services PCL) had a negligible positive ROI of 0.08%, with a Sharpe Ratio of 0.03 and a Calmar Ratio of 0.02, suggesting minimal trading activity or limited profitability during the testing period. The cumulative return across all stocks was 12.79%, reflecting the overall performance of the DQN agent. While this indicates a positive return at the portfolio level, the disparities among individual stocks suggest that the agent excelled in certain areas but struggled with consistency across the portfolio. The success with stocks like ADVANC, TISCO, and IVL showcases the agent's potential, whereas the underperformance with stocks like SCC, INTUCH, and PTTEP points to areas where the agent's strategy could be refined.

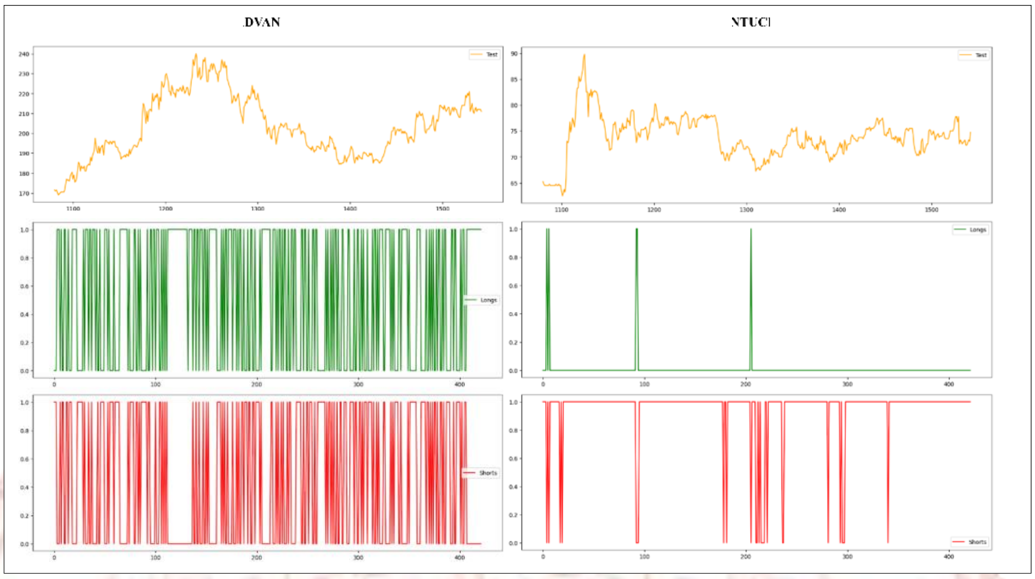


FIGURE 4-10 The price trend of the ADVANC and INTOUCH of DQN

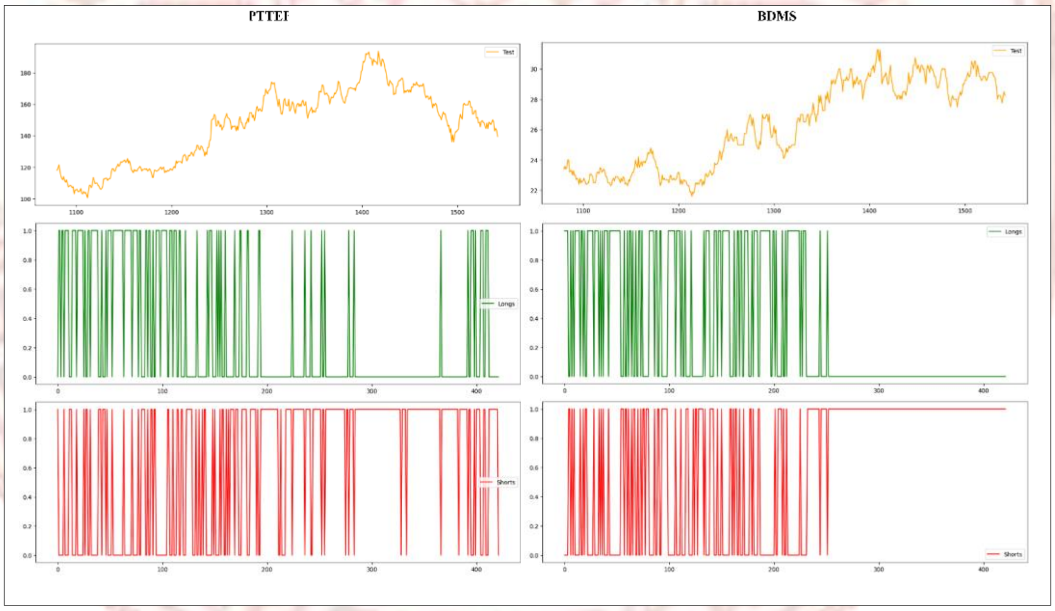


FIGURE 4-11 The price trend of the PTTEP and BDMS of DQN

Figure 4-11 displays the price trends of PTTEP and BDMS stocks during the testing period, accompanied by the agent's buy and sell signals. The agent's trading actions for these stocks demonstrate its strategy in navigating market movements. For PTTEP, the agent's cautious approach resulted in minimal losses, showing its tendency to avoid high-risk scenarios. With BDMS, limited trading signals suggest the agent struggled to identify clear market trends, leading to negligible returns.

Figure 4-12 shows the price trends of MINT and CPN stocks during the testing period, along with the agent's buy and sell signals. The agent's decisions for these stocks highlight its approach to capitalizing on market trends. The agent effectively leveraged upward trends in CPN, timing its trades to maximize profits. In contrast, trading with MINT was more conservative, possibly due to higher market volatility, resulting in modest gains.

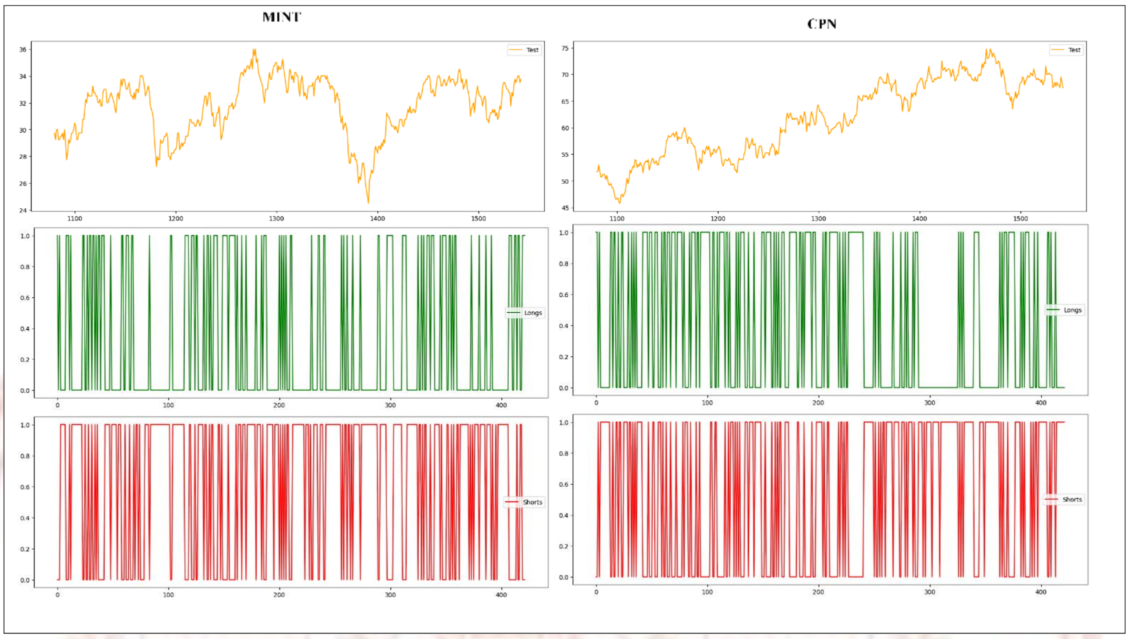


FIGURE 4-12 The price trend of the MINT and CPN of DQN

Figure 4-13 illustrates the price trends of AOT and TISCO stocks during the testing period, with the agent's buy and sell signals overlaid. The agent's trading behavior for these companies reflects its responsiveness to market dynamics. The agent demonstrated strong performance with TISCO, making timely trades that led to substantial returns and low drawdowns. The agent maintained a steady trading pattern for AOT, achieving consistent, albeit smaller, profits.

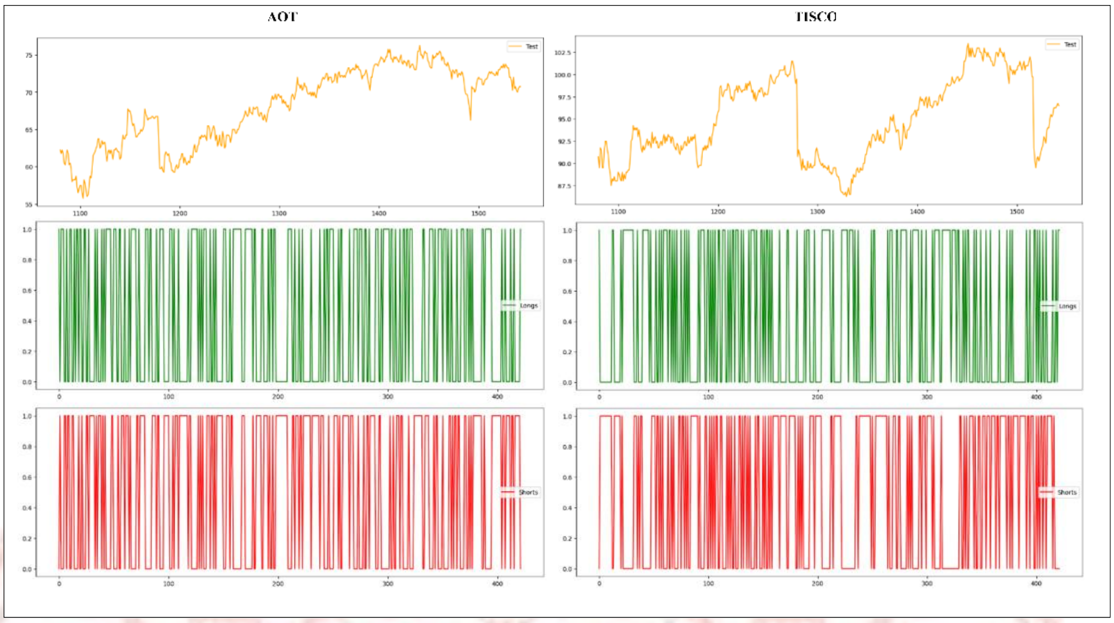


FIGURE 4-13 The price trend of the AOT and TISCO of DQN

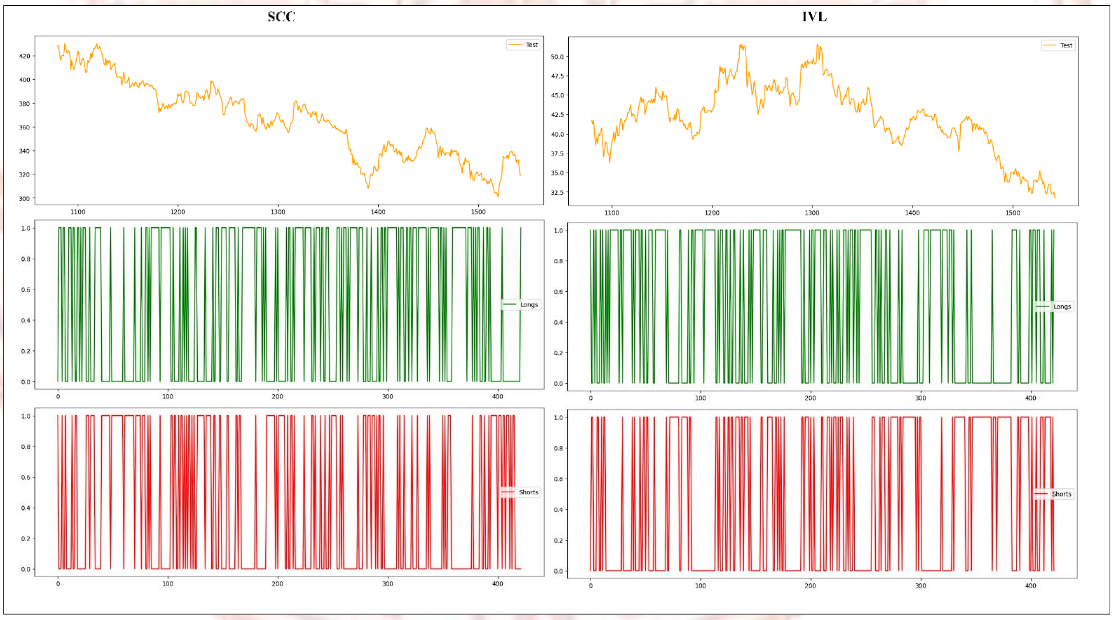


FIGURE 4-14 The price trend of the SCC and IVL of DQN

Figure 4-14 presents the price trends of SCC and IVL stocks during the testing period, along with the agent's buy and sell signals. The agent's trading actions for these stocks showcase its strategy for adapting to market conditions. While the agent struggled with SCC, failing to mitigate losses during declining markets, it adapted its strategy with IVL to capture gains despite mid-period volatility.

It comprehensively evaluated three reinforcement learning trading agents, PPO, A2C, and DQN, across a diversified portfolio of ten stocks. To gauge profitability and risk management capabilities, each agent was assessed using key performance metrics, including ROI, Sharpe Ratio, Maximum Drawdown, and Calmar Ratio. The PPO agent achieved a cumulative return of 15.46%, demonstrating notable success with stocks like IVL and PTTEP, but faced inconsistencies with underperforming stocks such as CPN. The A2C agent attained a cumulative return of 7.09%, showing strengths in stocks like CPN but recorded zero ROI in several others, indicating areas for strategy refinement. The DQN agent yielded a cumulative return of 12.79%, excelling in stocks like ADVANC and TISCO, but encountered challenges with stocks like SCC. Overall, while each agent exhibited potential in generating returns and managing risk under certain market conditions, the varying performances underscore the necessity for improving consistency and optimizing trading strategies across the entire portfolio to enhance overall effectiveness.

The results highlight the strengths and limitations of DRL-based trading strategies, offering valuable insights into their adaptability and performance under varying market conditions. Considering these findings, Chapter 5 reflects on this research's overall contributions, addresses its limitations, and proposes further directions for future work to advance DRL in financial markets.

CHAPTER 5
CONCLUSION, DISCUSSION, AND FUTURE WORK

This chapter summarizes the key findings of this study, discusses their implications in the broader context of financial markets and reinforcement learning, and identifies potential directions for future research and practical applications. Through this chapter, we provide a reflective assessment of the deep reinforcement learning DRL-based trading model developed for the Stock Exchange of Thailand (SET) and explore avenues for further refinement and extension of this work.

5.1 Conclusion

The thesis presented in this thesis demonstrates the potential of deep reinforcement learning (DRL) models to effectively handle the complexities of stock trading on the Stock Exchange of Thailand (SET). By implementing three DRL algorithms—Target Deep Q-Network (TDQN), Advantage Actor-Critic (A2C), and Proximal Policy Optimization (PPO), this study has shown that AI-driven trading agents can develop adaptive strategies that maximize returns while managing risk. Key conclusions from the research include:

Performance Comparison: Among the three models, PPO demonstrated superior adaptability in the volatile SET environment, achieving higher returns and a favorable risk-to-reward ratio. A2C and TDQN also showed promise but needed more consistency in handling rapid market fluctuations.

Data Preprocessing and Feature Engineering: Integrating data normalization techniques and engineered features, such as the Volume-Weighted Average Price (VWAP), played a critical role in enhancing model performance. These features enabled the models to capture market trends better, thus improving decision accuracy.

Applicability to Emerging Markets: This study emphasizes the effectiveness of DRL in emerging markets like SET, where traditional models often need to catch up. The DRL model’s capacity for continuous learning and adaptation shows significant promise in navigating the unique challenges of emerging financial markets.

Overall, the study validates the effectiveness of DRL-based trading strategies, providing a solid foundation for integrating these models into real-world trading systems that require adaptability, robustness, and a balance between profit and risk management.

5.2 Discussion

This study's findings contribute to a growing body of research on AI-driven trading strategies and underscore the advantages of DRL in complex, volatile markets. This section discusses the study's findings' implications, limitations, and practical deployment considerations in financial markets.

Advantages of DRL in Financial Trading: DRL offers a flexible framework that allows trading agents to learn optimal strategies without the need for predefined rules. This adaptability is particularly advantageous in financial markets, where conditions constantly change, and traditional rule-based approaches often need help keeping up.

The results show that DRL models, particularly PPO, can learn complex trading strategies to yield higher returns while managing risk.

Limitations and Constraints: Despite the promising results, the study also encountered limitations that affected the generalizability of the findings. The DRL models were trained on historical data, which may not fully represent future market conditions. Furthermore, the model's performance may vary based on hyperparameter settings and the specific selection of stocks. Additionally, the computational demands of DRL training can be a limiting factor in practical applications, as real-time adaptation requires substantial processing power.

Implications for Financial Practitioners: This study's successful implementation of DRL suggests that AI-driven models can enhance traditional financial strategies. However, deploying DRL models in real trading environments demands careful consideration of operational factors, such as data latency, transaction costs, and regulatory constraints. Financial practitioners must also consider the interpretability of AI models as regulatory bodies increasingly prioritize transparency in automated decision-making systems.

This discussion highlights the transformative potential and the practical challenges associated with integrating DRL into stock trading, underscoring the need for further research and refinement before full-scale deployment.

5.3 Future Work

While this research has made significant strides in applying DRL to stock trading in emerging markets, several promising directions remain for future exploration and improvement. Expanding upon this foundation could enhance the practical performance and theoretical robustness of DRL-based trading models.

Real-Time Learning and Online Adaptation: Future work could incorporate online learning techniques, enabling the model to adapt to incoming market data quickly. This could improve the model's responsiveness to sudden market changes and enhance its robustness in volatile trading environments.

Integration of Additional Data Sources: Including alternative data sources, such as news sentiment, social media signals, and macroeconomic indicators, could provide a richer context for trading decisions. By integrating these external factors, future models could achieve a more holistic understanding of market conditions, potentially improving prediction accuracy and adaptability.

Model Interpretability and Explainability: As DRL models are increasingly considered for deployment in regulated financial environments, enhancing their interpretability becomes crucial. Future studies could explore explainable AI (XAI) techniques, such as Shapley values or feature importance analysis, to increase transparency in the model’s decision-making process, making it easier for financial analysts to understand and trust AI-driven strategies.

Hybrid Models and Ensemble Techniques: Combining DRL with other machine learning methods, such as supervised learning or ensemble techniques, may create hybrid models that leverage the strengths of multiple approaches. For example, integrating supervised learning for market prediction with DRL for decision-making could yield a more robust trading system.

Risk Management and Reward Shaping: Future research could focus on refining the reward functions to incorporate more sophisticated risk management strategies. By shaping the reward structure to account for factors like volatility, maximum drawdown, and transaction costs, models can learn trading strategies that are not only profitable but also more risk sensitive.

Application to Different Markets and Asset Classes: Expanding the application of DRL models to other markets, such as commodities, bonds, or forex, could provide insights into their adaptability across different asset classes. Additionally, applying these models to high-frequency trading or multi-asset portfolios could test the scalability and flexibility of DRL in diverse trading environments.

This future work aims to build upon the foundations of the current study, addressing its limitations and exploring innovative approaches to enhance DRL-based trading models' adaptability, interpretability, and applicability in real-world financial markets.

REFERENCES

Avramelou, L.,et al. “Cryptosentiment: A Dataset and Baseline for Sentiment-Aware Deep Reinforcement Learning for Financial Trading.” *In Proceedings of 2023 IEEE International Conference on Acoustics, Speech, and Signal Processing Workshops (ICASSPW)*. 1-5 (4-10 June 2023).

Ballings, M.,et al. "Evaluating multiple classifiers for stock price direction prediction." *Expert Systems with Applications*, vol.42 no.20, (2015): 7046-7056. https://doi.org/https://doi.org/10.1016/j.eswa.2015.05.013

Bollen, J., Mao, H., & Zeng, X. "Twitter mood predicts the stock market." *Journal of Computational Science*, vol.2 no.1, (2011): 1-8. https://doi.org/https://doi.org/10.1016/j.jocs.2010.12.007

Brigham, E. F., & Ehrhardt, M. C. (2013). *Financial Management: Theory & Practice*. Cengage Learning.

Cheng, L. C.,et al. (2021). A novel trading strategy framework based on reinforcement deep learning for financial market predictions. *Mathematics*, *9*(23), 3094. https://doi.org/10.3390/math9233094

Dang, Q.-V. “Reinforcement Learning in Stock Trading.” *In Proceedings of Advanced Computational Methods for Knowledge Engineering*. 311-322 (2020).

Dantas, S. G., & Silva, D. G. “Equity Trading at the Brazilian Stock Market Using a Q-Learning Based System.” *In Proceedings of 2018 7th Brazilian Conference on Intelligent Systems (BRACIS)*. 133-138 (22-25 Oct. 2018).

Fischer, T., & Krauss, C. "Deep learning with long short-term memory networks for financial market predictions." *European Journal of Operational Research*, vol.270 no.2, (2018): 654-669. https://doi.org/https://doi.org/10.1016/j.ejor.2017.11.054

Hendershott, T., Jones, C. M., & Menkveld, A. J. "Does Algorithmic Trading Improve Liquidity?" *The Journal of Finance*, vol.66 no.1, (2011): 1-33. https://doi.org/https://doi.org/10.1111/j.1540-6261.2010.01624.x

Hochreiter, S., & Schmidhuber, J. "Long Short-Term Memory." *Neural Comput.*, vol.9 no.8, (1997): 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Hu, Y. J., & Lin, S. J. "Deep Reinforcement Learning for Optimizing Finance Portfolio Management." *In Proceedings of 2019 Amity International Conference on Artificial Intelligence (AICAI)*. 14-20 (4-6 Feb. 2019).

Hu, Z.,et al. (2018). Listening to Chaotic Whispers: A Deep Learning Framework for News-oriented Stock Trend Prediction. *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, 261–269. https://doi.org/10.1145/3159652.3159690

Johnman, M., Vanstone, B. J., & Gepp, A. "Predicting FTSE 100 returns and volatility using sentiment analysis." *Accounting & Finance*, vol.58 no.S1, (2018): 253-274. https://doi.org/https://doi.org/10.1111/acfi.12373

Jordan, M. I., & Mitchell, T. M. "Machine learning: Trends, perspectives, and prospects." *Science*, vol.349 no.6245, (2015): 255-260. https://doi.org/10.1126/science.aaa8415

Kara, Y., Acar Boyacioglu, M., & Baykan, Ö. K. "Predicting direction of stock price index movement using artificial neural networks and support vector machines: The sample of the Istanbul Stock Exchange." *Expert Systems with Applications*, vol.38 no.5, (2011): 5311-5319. https://doi.org/https://doi.org/10.1016/j.eswa.2010.10.027

Kim, Y., & Enke, D. (2016). Using neural networks to forecast volatility for an asset allocation strategy based on the target volatility. *Procedia Computer Science, 95*, 281–286.

Kumar, B. R. A.,et al. "Evaluating the Performance of Diverse Machine Learning Approaches in Stock Market Forecasting." *In Proceedings of Multi-*

disciplinary Trends in Artificial Intelligence: 16th International Conference, MIWAI 2023, Hyderabad, India, July 21–22, 2023, Proceedings. 255–264.

Lakshmanarao, A.,et al. “Loan Default Prediction Using Machine Learning Techniques and Deep Learning ANN Model.” *In Proceedings of 2023 Annual International Conference on Emerging Research Areas: International Conference on Intelligent Systems (AICERA/ICIS)*. 1-5 (16-18 Nov. 2023).

LeCun, Y., Bengio, Y., & Hinton, G. "Deep learning." *Nature*, vol.521 no.7553, (2015): 436-444. https://doi.org/10.1038/nature14539

Lei, K.,et al. "Time-driven feature-aware jointly deep reinforcement learning for financial signal representation and algorithmic trading." *Expert Systems with Applications*, vol.140, (2020): 112872. https://doi.org/https://doi.org/10.1016/j.eswa.2019.112872

Li, Y., Zheng, W., & Zheng, Z. "Deep Robust Reinforcement Learning for Practical Algorithmic Trading." *IEEE Access*, vol.7, (2019): 108014-108022. https://doi.org/10.1109/ACCESS.2019.2932789

Li, Y., Ni, P., & Chang, V. (2020). Application of deep reinforcement learning in stock trading strategies and stock forecasting. *Computing*, *102*(6), 1305–1322. https://doi.org/10.1007/s00607-019-00773-w

Liu et al. (2022). Practical deep reinforcement learning approach for stock trading. *arXiv*. https://doi.org/10.48550/arXiv.1811.07522

Liu et al. “FinRL: deep reinforcement learning framework to automate trading in quantitative finance.” *In Proceedings of The Second ACM International Conference on AI in Finance*. Article 1.

Magdon-Ismail, M., & Atiya, A. F. (2004). Maximum drawdown. *Risk Magazine*, *17*(10), 99–102.

Mnih, V.,et al. “Asynchronous Methods for Deep Reinforcement Learning.” *In Proceedings of The 33rd International Conference on Machine Learning*. 1928--1937.

Mnih, V.,et al. "Human-level control through deep reinforcement learning." *Nature*, vol.518 no.7540, (2015): 529-533. https://doi.org/10.1038/nature14236

Moody, J., & Saffell, M. "Learning to trade via direct reinforcement." *IEEE Transactions on Neural Networks*, vol.12 no.4, (2001): 875-889. https://doi.org/10.1109/72.935097

Nan, A., Perumal, A., & Zaiane, O. R. "Sentiment and Knowledge Based Algorithmic Trading with Deep Reinforcement Learning." *In Proceedings of Database and Expert Systems Applications*. 167-180 (2022//).

Nassirtoussi, A. K., et al. "Text mining for market prediction: A systematic review." *Expert Systems with Applications*, vol.41 no.16, (2014): 7653-7670. https://doi.org/https://doi.org/10.1016/j.eswa.2014.06.009

Nelson, D. M. Q., Pereira, A. C. M., & Oliveira, R. A. d. "Stock market's price movement prediction with LSTM neural networks." *In Proceedings of 2017 International Joint Conference on Neural Networks (IJCNN)*. 1419-1426 (14-19 May 2017).

Puterman, M. L. (2014). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons.

Serrano, W. "Deep Reinforcement Learning with the Random Neural Network." *Engineering Applications of Artificial Intelligence*, vol.110, (2022): 104751. https://doi.org/https://doi.org/10.1016/j.engappai.2022.104751

Sharpe, W. F. (1966). Mutual fund performance. *The Journal of Business*, *39*(1), 119–138.

Silver, D.,et al. "Mastering the game of Go with deep neural networks and tree search." *Nature*, vol.529 no.7587, (2016): 484-489. https://doi.org/10.1038/nature16961

Stock Exchange of Thailand. 12 November, 2024, Available from: https://www.set.or.th/th/about/overview/journey

Sülo, I.,et al. “Energy Efficient Smart Buildings: LSTM Neural Networks for Time Series Prediction.” *In Proceedings of 2019 International Conference on Deep Learning and Machine Learning in Emerging Applications (Deep-ML)*. 18-22 (26-28 Aug. 2019).

Taghian, M., Asadi, A., & Safabakhsh, R. "Learning financial asset-specific trading rules via deep reinforcement learning." *Expert Systems with Applications*, vol.195, (2022): 116523. https://doi.org/https://doi.org/10.1016/j.eswa.2022.116523

Théate, T., & Ernst, D. "An application of deep reinforcement learning to algorithmic trading." *Expert Systems with Applications*, vol.173, (2021): 114632. https://doi.org/https://doi.org/10.1016/j.eswa.2021.114632

Vázquez-Canteli, J. R., & Nagy, Z. "Reinforcement learning for demand response: A review of algorithms and modeling techniques." *Applied Energy*, vol.235, (2019): 1072-1089. https://doi.org/https://doi.org/10.1016/j.apenergy.2018.11.002

Watkins, C. J. C. H., & Dayan, P. "Q-learning." *Machine Learning*, vol.8 no.3, (1992): 279-292. https://doi.org/10.1007/BF00992698

Weng, B.,et al. "Predicting short-term stock prices using ensemble methods and online data sources." *Expert Systems with Applications*, vol.112, (2018): 258-273. https://doi.org/https://doi.org/10.1016/j.eswa.2018.06.016

Yadav, A., Jha, C. K., & Sharan, A. (2020). Optimizing LSTM for time series prediction in Indian stock market. *Procedia Computer Science*, *167*, 2091–2100. https://doi.org/10.1016/j.procs.2020.03.257

Yang, H.et al. “Deep reinforcement learning for automated stock trading: an ensemble strategy.” *In Proceedings of The First ACM International Conference on AI in Finance*. Article 31.

APPENDIX A

The code of Proximal Policy Optimization (PPO) model

```
"""
Below is the complete code for the PPO agent, organized into logical sections for
clarity in Google Colab.
requir:
python 3.10.12
ta 0.11.0
create folder: tmp1
"""
#!pip install ta
# Import necessary libraries for data handling and processing
import pandas as pd
from pandas_datareader import DataReader
import ta
from ta.volume import VolumeWeightedAveragePrice
import yfinance as yf
from sklearn.preprocessing import StandardScaler

# Env
import gym
from gym import spaces
import numpy as np
import random
import torch

# Pytorch
import os
import numpy as np
import torch as T
import torch.nn as nn
import torch.optim as optim
from torch.distributions import Categorical
# Outputs
import matplotlib.pyplot as plt

# 1. Data Preprocessing
def process_stock(symbol):
    print(f"Processing stock: {symbol}")

    # Download data
    df = yf.download(symbol, start='2017-01-01', end='2023-06-01')
    df.drop('Adj Close', axis=1, inplace=True)
    print(df)

    # Flatten MultiIndex columns if they exist
    if isinstance(df.columns, pd.MultiIndex):
        df.columns = df.columns.get_level_values(0)
```

```
print(df.head())  # For debugging

# Calculate VWAP and preprocess data
print('Calculate VWAP')


# Calculate VWAP and preprocess data
vwap = VolumeWeightedAveragePrice(
    high=df['High'],
    low=df['Low'],
    close=df['Close'],
    volume=df['Volume'],
    window=14,
    fillna=False
)
df['VWAP'] = vwap.volume_weighted_average_price()
df.dropna(inplace=True)

# Select features
features = ['Open', 'High', 'Low', 'Close', 'Volume', 'VWAP']
scaler = StandardScaler()
scaled_features = scaler.fit_transform(df[features])

# Prepare the dataframe
df_mod = pd.DataFrame(scaled_features, columns=features)
close_price_values = df["Close"].values

# Assign 'Close_Price' to df_mod
df_mod["Close_Price"] = close_price_values  # Potential source of error

# Reset index
df_mod = df_mod.reset_index(drop=True)
print(f"df_mod shape after reset_index: {df_mod.shape}")

# Split data into training and testing sets
df_train = df_mod.iloc[:int(len(df_mod) * 0.7)]
df_test = df_mod.iloc[int(len(df_mod) * 0.7):]

# Plot close data
plt.rcParams['figure.figsize'] = [15, 5]
df_train['Close_Price'].plot(label='Train')
df_test['Close_Price'].plot(label='Test')
plt.legend()
plt.show()

return df_train, df_test, df, df_mod, scaler  # Return scaler for later use
```

```
# 2. Environment Definition
INITIAL_ACCOUNT_BALANCE = 2000

# Structure env
class StockTradingEnv(gym.Env):
    """A stock trading environment for OpenAI gym"""
    metadata = {'render.modes': ['human']}

    def __init__(self, df):
        super(StockTradingEnv, self).__init__()

        # Generic variables
        self.df = df

        # Account variables
        self.available_balance = INITIAL_ACCOUNT_BALANCE
        self.net_profit = 0

        # Position variables
        self.num_trades_long = 0
        self.num_trades_short = 0
        self.long_short_ratio = 0

        # Current Step
        self.current_step = 0
        self.lag = 20
        self.volatility = 1
        self.max_steps = len(df)

        # Actions: Long (0), Short (1), Hold (2)
        self.action_space = spaces.Discrete(3)

        # Observation space
        self.observation_space = spaces.Box(low=-np.inf, high=np.inf, shape=(7,), 
dtype=np.float32)

        # Parameters for dynamic allocation
        self.max_percent = 0.2  # Maximum 20% allocation
        self.min_percent = 0.05 # Minimum 5% allocation

    def _calculate_dynamic_allocation(self):
        """
        Calculates dynamic position sizing based on recent volatility.
        """
        recent_volatility = self.df.loc[self.current_step - self.lag:self.current_step, 
"Close_Price"].std()
```

```
        allocation = self.min_percent + (self.max_percent - self.min_percent) * (1 / (1 +
recent_volatility))
        allocation = np.clip(allocation, self.min_percent, self.max_percent)
        return allocation

    # Reward function now returns immediate profit or loss
    def _calculate_reward(self, profit_loss):
        """
        Calculates the immediate reward based on profit or loss.
        """
        return profit_loss

    # Structure observation data
    def _next_observation(self):
        """
        Retrieves the next observation from the environment.
        """
        item_0_T0 = self.df.loc[self.current_step, "Open"].item()
        item_1_T0 = self.df.loc[self.current_step, "High"].item()
        item_2_T0 = self.df.loc[self.current_step, "Low"].item()
        item_3_T0 = self.df.loc[self.current_step, "Close"].item()
        item_4_T0 = self.df.loc[self.current_step, "Volume"].item()
        item_5_T0 = self.df.loc[self.current_step, "VWAP"].item()

        env_4 = 1 if self.long_short_ratio else 0
        obs = np.array([item_0_T0, item_1_T0, item_2_T0, item_3_T0, item_4_T0,
item_5_T0, env_4], dtype=np.float32)
        return obs

    # Update the action handling and reward calculation
    def _take_action(self, action):
        """
        Executes the given action and updates the environment state.
        """
        current_price = self.df.loc[self.current_step, "Close_Price"].item()
        next_price = self.df.loc[self.current_step + 1, "Close_Price"].item()
        next_return = next_price / current_price - 1

        allocation = self._calculate_dynamic_allocation()

        if action == 0:  # Long
            profit_loss = self.available_balance * allocation * next_return
            self.available_balance += profit_loss
            self.net_profit += profit_loss
            self.num_trades_long += 1

        elif action == 1:  # Short
```

```
            profit_loss = self.available_balance * allocation * -next_return
            self.available_balance += profit_loss
            self.net_profit += profit_loss
            self.num_trades_short += 1

        elif action == 2:  # Hold
            # No position taken; no profit or loss
            profit_loss = 0

        # Calculate reward based on immediate profit or loss
        self.reward = self._calculate_reward(profit_loss)

        # Update metrics
        self.long_short_ratio = self.num_trades_long / (self.num_trades_short +
self.num_trades_long + 1e-5)
        self.volatility = self.df.loc[self.current_step - self.lag:self.current_step,
"Close_Price"].std()

    # Execute one time step within the env
    def step(self, action):
        """
        Executes one time step within the environment.
        """
        self._take_action(action)

        reward = self.reward  # Use the immediate reward calculated in _take_action

        self.current_step += 1

        is_max_step_taken = self.current_step >= self.max_steps - self.lag - 1
        done = is_max_step_taken

        obs = self._next_observation()

        return obs, reward, done, {}

    # Reset the state of the env to an initial state
    def reset(self):
        """
        Resets the environment to an initial state.
        """
        self.available_balance = INITIAL_ACCOUNT_BALANCE
        self.net_profit = 0
        self.current_step = self.lag
        self.num_trades_long = 0
        self.num_trades_short = 0
        self.long_short_ratio = 0  # Corrected variable name
        return self._next_observation()

    # Render the env to the console
    def render(self, mode='human', close=False):
        """
        Renders the environment.
        """
        pass

# 3. Neural Network Architectures
# Actor Neural Net
class ActorNetwork(nn.Module):
    """
    Neural network for the actor in PPO.
    """
    def __init__(self, n_actions, input_dims, alpha,symbol, fc1_dims=256,
fc2_dims=256, fc3_dims=256, chkpt_dir='/content/tmp1/'):
        super(ActorNetwork, self).__init__()
        self.checkpoint_file = os.path.join(chkpt_dir, f'actor_torch_ppo_{symbol}')
        self.actor = nn.Sequential(
            nn.Linear(*input_dims, fc1_dims),
            nn.ReLU(),
            nn.Linear(fc1_dims, fc2_dims),
            nn.ReLU(),
            nn.Linear(fc2_dims, fc3_dims),  # New layer
            nn.ReLU(),
            nn.Linear(fc3_dims, n_actions),
            nn.Softmax(dim=-1)
        )


        self.optimizer = optim.AdamW(self.parameters(), lr=alpha)
        self.device = T.device('cuda:0' if T.cuda.is_available() else 'cpu') # GPU if have
        self.to(self.device)

    def forward(self, state):
        dist = self.actor(state)
        dist = Categorical(dist)

        return dist

    def save_checkpoint(self):
        T.save(self.state_dict(), self.checkpoint_file)

    def load_checkpoint(self):
        self.load_state_dict(T.load(self.checkpoint_file))
```

```
# Critic Neural Net
class CriticNetwork(nn.Module):
    """
    Neural network for the critic in PPO.
    """
    def __init__(self, input_dims, alpha,symbol, fc1_dims=256, fc2_dims=256,
chkpt_dir='/content/tmp1/'):
        super(CriticNetwork, self).__init__()
        self.checkpoint_file = os.path.join(chkpt_dir, f'critic_torch_ppo_{symbol}')
        self.critic = nn.Sequential(
            nn.Linear(*input_dims, fc1_dims),
            nn.ReLU(),
            nn.Linear(fc1_dims, fc2_dims),
            nn.ReLU(),
            nn.Linear(fc2_dims, 1)
        )

        self.optimizer = optim.AdamW(self.parameters(), lr=alpha)
        self.device = T.device('cuda:0' if T.cuda.is_available() else 'cpu') # GPU if have
        self.to(self.device)

    def forward(self, state):
        value = self.critic(state)

        return value

    def save_checkpoint(self):
        T.save(self.state_dict(), self.checkpoint_file)

    def load_checkpoint(self):
        self.load_state_dict(T.load(self.checkpoint_file))

# 4. Agent Definition
class PPOmemory:
    """
    Memory buffer for storing experiences during training.
    """
    def __init__(self, batch_size):
        self.states = []
        self.probs = []
        self.vals = []
        self.actions = []
        self.rewards = []
        self.dones = []

        self.batch_size = batch_size
    def generate_batches(self):
        n_states = len(self.states)
        batch_start = np.arange(0, n_states, self.batch_size)
        indices = np.arange(n_states, dtype=np.int64)
        np.random.shuffle(indices)
        batches = [indices[i:i+self.batch_size] for i in batch_start]

        return np.array(self.states),\
                np.array(self.actions),\
                np.array(self.probs),\
                np.array(self.vals),\
                np.array(self.rewards),\
                np.array(self.dones),\
                batches

    def store_memory(self, state, action, probs, vals, reward, done):
        self.states.append(state)
        self.actions.append(action)
        self.probs.append(probs)
        self.vals.append(vals)
        self.rewards.append(reward)
        self.dones.append(done)

    def clear_memory(self):
        self.states = []
        self.probs = []
        self.actions = []
        self.rewards = []
        self.dones = []
        self.vals = []

class Agent:
  """
    PPO Agent that interacts with the environment and learns from experiences.
    """
  def __init__(self, n_actions, input_dims,symbol, gamma=0.99, alpha=0.0003,
gae_lambda=0.95, policy_clip=0.2, batch_size=64, n_epochs=10):
      self.gamma = gamma
      self.policy_clip = policy_clip
      self.n_epochs = n_epochs
      self.gae_lambda = gae_lambda
      self.actor = ActorNetwork(n_actions, input_dims, alpha,symbol)
      self.critic = CriticNetwork(input_dims, alpha,symbol)
      self.memory = PPOmemory(batch_size)

  def remember(self, state, action, probs, vals, reward, done):
      """
        Stores experiences in memory.
        """
      self.memory.store_memory(state, action, probs, vals, reward, done)

  def save_models(self):
      """
        Saves the actor and critic models.
        """
      print('... saving models ...')
      self.actor.save_checkpoint()
      self.critic.save_checkpoint()

  def load_models(self):
      print('... loading models ...')
      self.actor.load_checkpoint()
      self.critic.load_checkpoint()

  # Old: Something incorrect
  def choose_action(self, observation):
      """
        Chooses an action based on the current policy.
        """
      state = T.tensor([observation], dtype=T.float).to(self.actor.device)
      state = state.flatten(0) # new

      dist = self.actor(state)
      value = self.critic(state)
      action = dist.sample()

      probs = T.squeeze(dist.log_prob(action)).item()
      action = T.squeeze(action).item()
      value = T.squeeze(value).item()

      return action, probs, value

  def learn(self):
      """
        Updates the actor and critic networks based on collected experiences.
        """
      for _ in range(self.n_epochs):
        state_arr, action_arr, old_prob_arr, vals_arr, reward_arr,\
        dones_arr, batches = self.memory.generate_batches()

        values = vals_arr
        advantage = np.zeros(len(reward_arr), dtype=np.float32)
```

```
        for t in range(len(reward_arr)-1):
          discount = 1
          a_t = 0
          for k in range(t, len(reward_arr)-1):
            a_t += discount*(reward_arr[k] + self.gamma*values[k+1]*(1-
int(dones_arr[k])) - values[k])
            discount *= self.gamma*self.gae_lambda
          advantage[t] = a_t
        advantage = T.tensor(advantage).to(self.actor.device)

        values = T.tensor(values).to(self.actor.device)
        for batch in batches:
          states = T.tensor(state_arr[batch], dtype=T.float).to(self.actor.device)
          old_probs = T.tensor(old_prob_arr[batch]).to(self.actor.device)
          actions = T.tensor(action_arr[batch]).to(self.actor.device)

          dist = self.actor(states)
          critic_value = self.critic(states)

          critic_value = T.squeeze(critic_value)

          new_probs = dist.log_prob(actions)
          prob_ratio = new_probs.exp() / old_probs.exp()

          weighted_probs = advantage[batch] * prob_ratio
          weighted_clipped_probs = T.clamp(prob_ratio, 1-self.policy_clip,
1+self.policy_clip)*advantage[batch]

          actor_loss = -T.min(weighted_probs, weighted_clipped_probs).mean()

          returns = advantage[batch] + values[batch]
          critic_loss = (returns - critic_value)**2
          critic_loss = critic_loss.mean()

          total_loss = actor_loss + 0.5*critic_loss
          self.actor.optimizer.zero_grad()
          self.critic.optimizer.zero_grad()
          total_loss.backward() # new
          self.actor.optimizer.step()
          self.critic.optimizer.step()

      self.memory.clear_memory()

# 5. Training and Evaluation Functions
def plot_learning_curve(x, scores, figure_file):
    """
    Plots the learning curve of the agent.
    """
    running_avg = np.zeros(len(scores))
    for i in range(len(running_avg)):
        running_avg[i] = np.mean(scores[max(0, i-100):(i+1)]) # this can change to 50
scores
    plt.plot(x, running_avg)
    plt.title('Running average of previous 100 scores')
    plt.savefig(figure_file)

def plot_signals_and_equity(df_res, original_df, symbol):
    plt.figure(figsize=(15,5))
    plt.plot(original_df["Close"], label='Close Price', color='blue')
    # Plot Actions (Buy/Sell Signals)
    plt.rcParams["figure.figsize"] = (15, 5)
    df_res[["Longs"]].plot(color="green")
    df_res[["Shorts"]].plot(color="red")
    plt.show()


def calculate_roi(df_res, initial_capital):
    final_equity = df_res["Equity"].iloc[-1]
    roi = (final_equity / initial_capital - 1) * 100
    return roi

def calculate_sharpe_ratio(df_res, risk_free_rate=0.0001):
    daily_returns = df_res["Equity"].pct_change().dropna()
    excess_returns = daily_returns - risk_free_rate / 252
    sharpe_ratio = np.mean(excess_returns) / np.std(excess_returns) * np.sqrt(252)
    return sharpe_ratio

def calculate_max_drawdown(df_res):
    equity = df_res["Equity"]
    running_max = equity.cummax()
    drawdown = (equity - running_max) / running_max
    max_drawdown = drawdown.min() * 100  # Expressed as a percentage
    return max_drawdown

def calculate_sortino_ratio(df_res, risk_free_rate=0.0001):
    daily_returns = df_res["Equity"].pct_change().dropna()
    excess_returns = daily_returns - risk_free_rate / 252
    downside_returns = excess_returns[excess_returns < 0]
    downside_deviation = np.sqrt(np.mean(downside_returns**2)) * np.sqrt(252)
    if downside_deviation == 0:
        return np.nan  # Avoid division by zero
    sortino_ratio = np.mean(excess_returns) / downside_deviation
    return sortino_ratio
```

```
def plot_equity_drawdown(df_res, symbol):
    """
    Plots the equity curve and drawdown over time.
    """
    plt.figure(figsize=(13, 8))

    # Equity Curve
    plt.subplot(2, 1, 1)
    plt.plot(df_res["Equity"], label='Equity Curve', color='blue')
    plt.title(f'Equity Curve for {symbol}')
    plt.xlabel('Time')
    plt.ylabel('Equity')
    plt.legend()

    # Drawdown
    plt.subplot(2, 1, 2)
    equity = df_res["Equity"]
    running_max = equity.cummax()
    drawdown = (equity - running_max) / running_max
    plt.plot(drawdown, label='Drawdown', color='red')
    plt.title('Drawdown')
    plt.xlabel('Time')
    plt.ylabel('Drawdown (%)')
    plt.legend()

    plt.tight_layout()
    plt.show()

def calculate_metrics(df_res, initial_capital, perc_invest, symbol):
    # Calculate equity
    equities = [initial_capital]
    for i in range(1, len(df_res)):
        direction = df_res["Longs"].iloc[i-1] if df_res["Longs"].iloc[i-1] >= 0.5 else -
df_res["Shorts"].iloc[i-1]
        equity = equities[i-1] + equities[i-1] * direction * df_res["Returns"].iloc[i] *
perc_invest
        equities.append(equity)

    df_res["Equity"] = equities

    # Calculate Metrics
    roi = calculate_roi(df_res, initial_capital)
    sharpe_ratio = calculate_sharpe_ratio(df_res)
    max_drawdown = calculate_max_drawdown(df_res)
    sortino_ratio = calculate_sortino_ratio(df_res)
    # Estimate the number of years from the data
    total_days = len(df_res)
    years = total_days / 252  # Assuming 252 trading days in a year
    calmar_ratio = calculate_calmar_ratio(df_res, initial_capital, years)
    Benchmark_Perc = (df_res["Close_Price"].iloc[-1] / df_res["Close_Price"].iloc[0] -
1) * 100

    # Print Metrics
    print(f"=== Metrics for {symbol} ===")
    print(f"Benchmark Return for {symbol}: {round(Benchmark_Perc, 2)}%")
    print(f"ROI for {symbol}: {roi:.2f}%")
    print(f"Sharpe Ratio: {sharpe_ratio:.2f}")
    print(f"Maximum Drawdown: {max_drawdown:.2f}%")
    print(f"Sortino Ratio: {sortino_ratio:.2f}")
    print(f"Calmar Ratio: {calmar_ratio:.2f}")
    print("==============================\n")

    # Plot Equity and Drawdown
    plot_equity_drawdown(df_res, symbol)


def train_and_test_agent(symbol, df_train, df_test, df_mod, original_df, scaler):
    # Create the StockTradingEnv using the training data
    env = StockTradingEnv(df_train)

    # Initialize the agent
    N = 20
    batch_size = 5
    # n_epochs = 3
    n_epochs = 10
    alpha = 0.0003
    agent = Agent(n_actions=env.action_space.n,
input_dims=env.observation_space.shape, alpha=alpha, n_epochs=n_epochs,
batch_size=batch_size, symbol=symbol)

     # File to save the model training plot
    figure_file = f'{symbol}_stock_training.png'

    # Initialize tracking variables for training performance
    best_score = env.reward_range[0]
    score_history = []
    avg_score = 0
    n_steps = 0

    print(f"... start training for {symbol} ...")

    # Train the agent
    """
    The n_games value selected from the learning rate value in the log and improved
value of the model.
    recommend: n_games >= 100
    """
    n_games = 100
    score_history = []
    for i in range(n_games):
        observation = env.reset()
        done = False
        score = 0
        while not done:
            action, prob, val = agent.choose_action(observation)
            observation_, reward, done, info = env.step(action)
            agent.remember(observation, action, prob, val, reward, done)
            n_steps += 1
            score += reward
            if n_steps % N == 0:
              agent.learn()
            observation = observation_

        # Save score history and calculate the average score
        # if score > -5000 and score < 5000:
        score_history.append(score)
        avg_score = np.mean(score_history[-50:])  # Average over the last 50 games

        # Save model if the performance has improved
        if avg_score > best_score and i > 5:
            best_score = avg_score
            agent.save_models()

        print(f"Episode {i} | Score: {score} | Avg Score: {avg_score} | Best Score:
{best_score}")


    # After training, get the model results on test data
    reporting_df = df_test.copy().reset_index(drop=True)

    # Apply the same scaling to the test data
    features = ['Open', 'High', 'Low', 'Close', 'Volume', 'VWAP']
    scaled_features = scaler.transform(reporting_df[features])
    reporting_df[features] = scaled_features

    long_probs = []
    short_probs = []

    for step in range(len(reporting_df)):
        item_0_T0 = df_mod.loc[step - 0, "Open"].item()
        item_1_T0 = df_mod.loc[step - 0, "High"].item()
        item_2_T0 = df_mod.loc[step - 0, "Low"].item()
        item_3_T0 = df_mod.loc[step - 0, "Close"].item()
        item_4_T0 = df_mod.loc[step - 0, "Volume"].item()
        item_5_T0 = df_mod.loc[step - 0, "VWAP"].item()

        obs = np.array([item_0_T0, item_1_T0, item_2_T0, item_3_T0, item_4_T0,
item_5_T0, 0.5])

        state = T.tensor(obs).float()
        # Load Model
        n_actions = env.action_space.n
        input_dims = env.observation_space.shape
        alpha = 0.0003
        model = ActorNetwork(n_actions, input_dims, alpha, symbol)
        model.load_state_dict(T.load(f'/content/tmp1/actor_torch_ppo_{symbol}',
weights_only=True))
        model.eval()
        dist = model(state)
        probs = dist.probs.detach().numpy()

        action = np.argmax(probs)
        long_probs.append(probs[0])
        short_probs.append(probs[1])

    # Add the buy/sell signal to df
    df_res = reporting_df[["Open", "Close_Price"]].copy()
    df_res["Returns"] = df_res["Close_Price"].pct_change()
    df_res["Longs"] = long_probs
    df_res["Shorts"] = short_probs
    df_res.loc[df_res["Longs"] >= 0.5, "Action"] = "Buy"
    df_res.loc[df_res["Longs"] < 0.5, "Action"] = "Sell"

    # Plot Buy/Sell Signals and Equity
    plot_signals_and_equity(df_res, original_df, symbol)

    # Call calculate_metrics to evaluate the model performance
    initial_capital = 2000  # e.g., $100,000
    perc_invest = 0.1  # 10% of capital per trade
    calculate_metrics(df_res, initial_capital, perc_invest, symbol)

stock_symbols =
["ADVANC.BK","INTUCH.BK","PTTEP.BK","BDMS.BK","MINT.BK","CPN.BK
","AOT.BK","TISCO.BK","SCC.BK","IVL.BK"]
for symbol in stock_symbols:
    df_train, df_test, df, df_mod, scaler = process_stock(symbol)
    train_and_test_agent(symbol, df_train, df_test, df_mod, df, scaler)
```

APPENDIX B

The code of Advantage Actor Critic (A2C) model

```
"""
1. Memory Management
Memory Class (Memory): The A2C agent employs a simpler memory class without batch processing. It collects experiences and clears them after each learning update.
"""
class Memory:
    def __init__(self):
        self.states = []
        self.probs = []
        self.vals = []
        self.actions = []
        self.rewards = []
        self.dones = []

"""
2. Learning Update Mechanism
No Policy Clipping: A2C does not use policy clipping. The updates are more straightforward and occur more frequently.
"""
actor_loss = - (new_probs * advantages.detach()).mean()
critic_loss = F.mse_loss(critic_value, returns)
total_loss = actor_loss + 0.5 * critic_loss

"""

3. Advantage Calculation
Temporal Difference (TD) Error: A2C typically uses the TD error for advantage estimation, which is simpler and aligns with the actor-critic framework.
"""
for i in reversed(range(len(rewards))):
    if i == len(rewards) - 1:
        next_value = 0
    else:
        next_value = values[i + 1]
    delta = rewards[i] + self.gamma * next_value * (1 - dones[i]) - values[i]
    gae = delta + self.gamma * self.gae_lambda * (1 - dones[i]) * gae
    returns.insert(0, gae + values[i])
    advantages.insert(0, gae)



"""
4. Agent Class Differences
Simpler Initialization: Does not include policy_clip, reflecting a simpler update mechanism.
"""
```

```
class Agent:
    def __init__(self, ..., batch_size=64, ...):
        # No policy_clip parameter

"""
5. Neural Network Architecture
Consistency in Networks: Both PPO and A2C implementations use similar neural
network architectures for the actor and critic networks.
"""
# Actor Network
class ActorNetwork(nn.Module):
    def __init__(self, n_actions, input_dims, alpha, symbol, ...):
        # Network layers

# Critic Network
class CriticNetwork(nn.Module):
    def __init__(self, input_dims, alpha, symbol, ...):
        # Network layers

"""
6. Hyperparameter Adjustments
Frequent Updates: May perform updates more frequently (e.g., after every step or
episode) without batching.
N = 10
Immediate Learning: The agent may learn after each episode or after a smaller number
of steps.
"""

if n_steps % N == 0:
    agent.learn()
    n_steps = 0  # Reset steps after each update
# Single Update per Learning Call: The agent performs a single update without iterating
# over multiple epochs.
```

By focusing on these aspects, we can appreciate how the A2C algorithm balances efficiency and simplicity in policy optimization, impacting both the performance and practicality of reinforcement learning solutions.

APPENDIX C

The code of Deep Q-Network (DQN) model

```
"""
1. Value-Based vs. Policy-Based Methods
Single Network Outputting Q-Values in DQN:
"""
class DeepQNetwork(nn.Module):
    def __init__(self, lr, n_actions, input_dims, ...):
        # Initialize network layers
        # ...

    def forward(self, state):
        # Compute Q-values for all possible actions
        actions = self.fc3(x)
        return actions

"""
2. Experience Replay Buffer
Replay Buffer (ReplayBuffer Class): Stores transitions (state, action, reward, next state,
done) to decorrelate data and improve sample efficiency.
"""

class ReplayBuffer:
    def __init__(self, max_size, input_shape):
        self.mem_size = max_size
        self.mem_cntr = 0

        self.state_memory = np.zeros((self.mem_size, *input_shape), dtype=np.float32)
        self.new_state_memory = np.zeros((self.mem_size, *input_shape),
dtype=np.float32)
        self.action_memory = np.zeros(self.mem_size, dtype=np.int64)
        self.reward_memory = np.zeros(self.mem_size, dtype=np.float32)
        self.terminal_memory = np.zeros(self.mem_size, dtype=np.bool_)

    def store_transition(self, state, action, reward, state_, done):
        index = self.mem_cntr % self.mem_size

        self.state_memory[index] = state
        self.new_state_memory[index] = state_
        self.reward_memory[index] = reward
        self.action_memory[index] = action
        self.terminal_memory[index] = done

        self.mem_cntr += 1

    def sample_buffer(self, batch_size):
        max_mem = min(self.mem_cntr, self.mem_size)

        batch = np.random.choice(max_mem, batch_size, replace=False)
        states = self.state_memory[batch]
        states_ = self.new_state_memory[batch]
        rewards = self.reward_memory[batch]
        actions = self.action_memory[batch]
        dones = self.terminal_memory[batch]

        return states, actions, rewards, states_, dones

""
```

3. Target Network and Network Updates
Purpose: Stabilizes training by keeping a separate network (q_next) for calculating target Q-values, which is updated less frequently.

```
""
class DQNAgent:
    def __init__(self, ...):
        self.q_eval = DeepQNetwork(...)
        self.q_next = DeepQNetwork(...)
        # ...

    def replace_target_network(self):
        if self.learn_step_counter % self.replace_target_cnt == 0:
            self.q_next.load_state_dict(self.q_eval.state_dict())

""
```

4. Action Selection Mechanism
Exploration vs. Exploitation: Balances exploration and exploitation by selecting random actions with probability ϵ and the best action according to the Q-network otherwise.

```
""
class DQNAgent:
    def choose_action(self, observation):
        if np.random.random() > self.epsilon:
            # Exploit: Choose action with highest Q-value
            actions = self.q_eval.forward(state)
            action = T.argmax(actions).item()
        else:
            # Explore: Choose random action
            action = np.random.choice(self.action_space)
        return action
```

# 5. Loss Function and Optimization
DQN Loss Function (Mean Squared Error)

- Calculating Target Q-Values

q_target = rewards + self.gamma * q_next

- Computing Loss Between Predicted and Target Q-Values

loss = self.q_eval.loss(q_pred, q_target).to(self.q_eval.device)

```
# 6. Handling of Action Probabilities
# For plotting purposes, set probability of selected action to 1
probs = np.zeros(env.action_space.n)
probs[action] = 1
long_probs.append(probs[0])
short_probs.append(probs[1])

# 7. Exploration Strategy
class DQNAgent:
    def decrement_epsilon(self):
        if self.epsilon > self.epsilon_min:
            self.epsilon -= self.eps_dec
        else:
            self.epsilon = self.epsilon_min

# 8. Agent Initialization Parameters
class DQNAgent:
    def __init__(self, gamma, epsilon, lr, n_actions, input_dims,
                 batch_size, epsilon_end=0.01, mem_size=100000,
                 eps_dec=1e-5, replace=1000, ...):
        # Initialize DQN-specific parameters

# 9. Training Loop Differences
while not done:
    action = agent.choose_action(observation)
    observation_, reward, done, info = env.step(action)
    agent.store_transition(observation, action, reward, observation_, done)
    agent.learn()
    observation = observation_
```

VITA

Name	Mr. Warameth Nuipian
Thesis Title	Dynamic Portfolio Management with Deep Reinforcement Learning
Major Field	Information and Data Science (International Program)
Biography	House No. 14/268 Buathong Thani Park Ville 2, Moo2 Phimonrat Bang Bua Thong Nonthaburi 11110 I graduated with a Bachelor's degree in Computer Engineering from Suranaree University of Technology academic year 2022. I practice co-op at SCG as a full-stack developer. After graduating, I applied for a new company and worked in a front-end development position. I was responsible for developing systems such as the web e-commerce platform and LINE's e-commerce solution. I was also involved in creative AI-related projects, working specifically on fine-tuned Large Language Models (LLMs) for targeted business applications.