MANCHESTER 1824
The University of Manchester

A Novel Approach to Stock Trading with Group-Agent Deep Reinforcement Learning

A thesis submitted to the University of Manchester for the degree of MSc Advanced Computer Science: Artificial Intelligence in the Faculty of Science and Engineering

2024

Author: Clint Johnson
Student ID: 11358578

Supervisors: Dr. Xiao-jun Zeng, Kaiyue Wu

TABLE OF CONTENTS

List of Figures

List of Tables

Abstract

This study investigates the application of Group-Agent Reinforcement Learning (GARL) to stock market trading, addressing a gap in Reinforcement Learning (RL) research and its application on algorithmic trading. It examines the effectiveness of such a GARL based trading strategy against traditional machine learning (ARIMA) based trading strategies and standard RL algorithms (specifically, Deep Q-Networks, Proximal Policy Optimization and Advantage-Actor Critic). It uses historical data from large-cap US companies listed on the Nasdaq-100 index to test the developed strategies in a realistic simulated environment and compares them against various common trading strategy evaluation criteria.

The results reveal that while GARL shows potential in outperforming existing RL algorithms in terms of the profit, it carries additional risk. Surprisingly, a rolling ARIMA forecast based trading strategy demonstrated the most robust performance and risk management capabilities, outperforming the more complex RL strategies developed. Among the RL algorithms, DQN performed well consistently. The various challenges in training an RL algorithm and designing a trading environment are identified.

The research provides insights into the effectiveness of GARL as a RL paradigm by improving the learning of the agents, as well as its effectiveness as a trading algorithm by providing flexibility in capturing various dependencies in the market. The findings show that importance of environment design in RL and also the effectiveness of simple but carefully designed traditional machine learning trading strategies. It opens up possibilities for future research in various GARL formulations and algorithm variations within the GARL framework.

Declaration of originality

I hereby confirm that no portion of the work referred to in the thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright Statement

i. The author of this thesis (including any appendices and/or schedules to this thesis) owns certain copyright or related rights in it (the "Copyright") and s/he has given The University of Manchester certain rights to use such Copyright, including for administrative purposes.

ii. Copies of this thesis, either in full or in extracts and whether in hard or electronic copy, may be made *only* in accordance with the Copyright, Designs and Patents Act 1988 (as amended) and regulations issued under it or, where appropriate, in accordance with licensing agreements which the University has from time to time. This page must form part of any such copies made.

iii. The ownership of certain Copyright, patents, designs, trademarks and other intellectual property (the "Intellectual Property") and any reproductions of copyright works in the thesis, for example graphs and tables ("Reproductions"), which may be described in this thesis, may not be owned by the author and may be owned by third parties. Such Intellectual Property and Reproductions cannot and must not be made available for use without the prior written permission of the owner(s) of the relevant Intellectual Property and/or Reproductions.

iv. Further information on the conditions under which disclosure, publication and commercialisation of this thesis, the Copyright and any Intellectual Property and/or Reproductions described in it may take place is available in the University IP Policy (see http://documents.manchester.ac.uk/DocuInfo.aspx?DocID=24420/), in any relevant Thesis restriction declarations deposited in the University Library, The University Library's regulations (see http://www.library.manchester.ac.uk/about/regulations/) and in The University's policy on Presentation of Theses.

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my supervisor, Dr. Xiao-Jun Zeng, for his invaluable guidance and mentorship. His vast expertise and keen insights have moulded the backbone of this work, his encouragement has been the driving force behind my learning, and his dedication has been the key to the timely completion of this project. I consider myself fortunate to have had the opportunity to work under his supervision.

Additionally, I would like to extend my sincere thanks to my supervisor, Kaiyue Wu, PhD student, for her knowledgeable observations, thoughtful inputs, and constructive feedback, all of which have enriched the quality of this project. Her generous sharing of time and expertise has been a true asset.

Furthermore, I would like to acknowledge my appreciation to my family and my partner, who have been a constant source of love, strength, and emotional support throughout this journey.

Chapter 1

1 INTRODUCTION

1.1 Overview and Motivation

Trading is the practice of buying and selling financial securities such as stocks, forex, or derivatives. Unlike investing, wherein an investor may buy and own financial assets to benefit from dividends and price appreciation, trading is focused on buying and selling based on speculations of short term price movements and profiting from them.

Algorithmic trading refers to using computer algorithms to automatically and systematically execute trading strategies in financial markets. Traditionally, trading was carried out manually by human traders on exchange floors, relying on intuition or data analysis. The emergence of electronic exchanges facilitated the advancement of automated rule-based trading systems and algorithms. Due to their automated nature, these systems have the ability to swiftly execute trades and respond to market fluctuations more rapidly than humans. Additionally, they lack any human fallibility and remain unaffected by human biases and emotions, allowing for impartial decision making.

Algorithmic trading strategies are used by trading platforms to optimise the execution of trade orders and by institutional investors to execute large orders and profit from minute pricing discrepancies in the market (known as high-frequency trading). They eliminate bid-ask spreads (the difference between bid and ask prices for an asset) and have proven to improve asset liquidity in the market [1]. Liquidity here refers to the ease with which an asset can be bought or sold without affecting its price. The algorithmic trading market was valued at USD 14.42 billion in 2024 and is projected to grow at a compound annual growth rate (CAGR) of 8.53% [2]. Figure 1 shows a report by Goldman Sachs from 2018 that reveals that over 70% of the trade value in the US stock market is algorithmic [3].

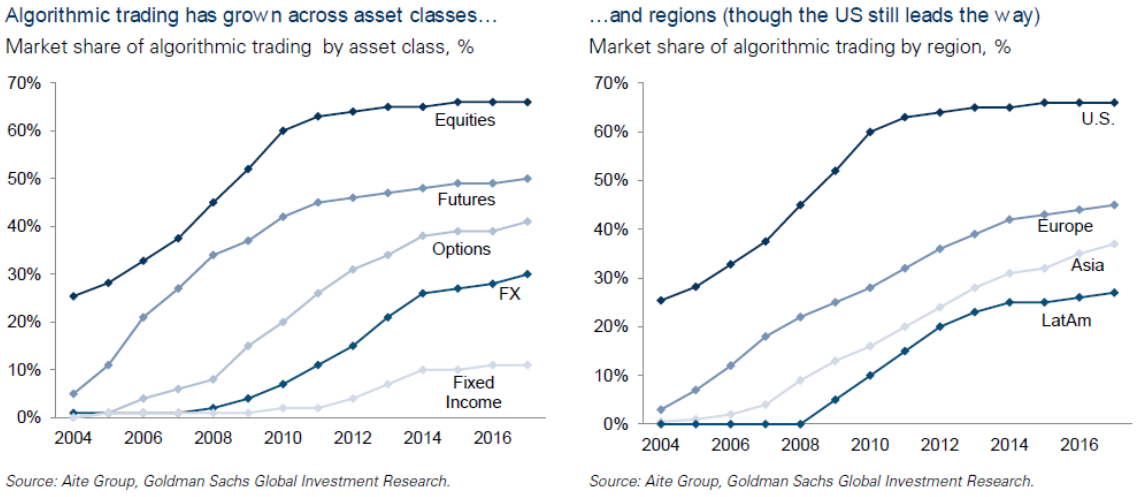

Figure 1: Market share of algorithmic trading, Goldman Sachs Global Investment Research, 2018 [3]

Quantitative trading algorithms are a subset of algorithmic trading algorithms that use mathematical and statistical models to identify trading opportunities and make decisions. Recent advancements in machine learning and deep learning have significantly improved the capabilities of trading systems by enabling them to efficiently examine large volumes of data and enhance decision making processes.

Deep learning techniques, in particular, can extract more complex patterns, automatically extract complex features, and model temporal and non-linear dependencies, leading to increased predictive accuracy [4]. Among these, deep reinforcement learning (RL) has emerged as a potent method for predicting stock prices and maximising returns in recent years [5], [6], [7], [8].

Deep RL techniques are characterised by virtual trader agents that self-learn to trade effectively. Unlike traditional machine learning and other deep learning techniques that focus on producing good trade signals that can then drive trading strategies, RL techniques focus on directly developing an intelligent agent that can perform good trades by automatically learning underlying market patterns implicitly, as well as learning to act on them opportunistically. Depending on the number of agents they contain, they can be divided into single-agent and multi-agent RL trading algorithms. While single-agent reinforcement learning have shown impressive results, outperforming indexes and traditional trading strategies [6], [8], [9], recent studies suggest that appropriate multi-agent reinforcement learning trading strategies working together in a cooperative or ensemble manner have the capacity to further enhance trading performance from single-agent RL strategies [10], [11], [12]. They capitalise on the successes of single-agent deep reinforcement learning agent, collectively performing better than the individual agents. A discussion and critical analysis of the various approaches to multi-agent RL and ensemble RL strategies are provided in Section 2.2.

Group-agent reinforcement learning (GARL) is a novel formulation of reinforcement learning which involves multiple RL agents in a group communicating to share knowledge and learning to perform their tasks [13]. GARL is fundamentally different from multi-agent reinforcement learning and ensemble approaches in its formulation and will be discussed in detail in Section 2.2.

The fractal market hypothesis states that market behaviour is influenced by the collective psychology of investors, who have different investment horizons and variable interpretations of information [14]. The GARL algorithm, by allowing agents to work in their own isolated environments, has the potential to leverage this hypothesis by enabling the exchange of knowledge among agents working on different timeframes of the same stock. Furthermore, similar to popular statistical arbitrage techniques such as pairs trading, which exploit temporary deviations between stocks that are positively or negatively correlated [15], a GARL-based trading strategy might benefit from enabling agents engaged in trading in separate but highly correlated stocks to exchange information. Unlike ensemble RL approaches, where the focus is on the collective performance of the group, GARL emphasises cooperative learning among the agents, potentially improving the performance of individual agents as well as the group as a whole. Although GARL has shown promising results in typical RL scenarios like the CartPole-v0 game [16] and outperformed the individual agents in the group [11], its application in stock trading remains unexplored.

This project aims to apply GARL to stock market data to potentially improve trading performance and maximise returns. GARL allows for the formation of a group of distinct agents employing different reinforcement learning algorithms without imposing restrictions on the algorithms used by individual agents or the type of information that may be exchanged. Hence, there is significant potential for experimentation with diverse algorithms and knowledge-sharing strategies. In this project, a trading strategy based on GARL was devised and implemented following extensive research and experimentation. The strategy was then backtested using historical US stock market data, and its performance was assessed in comparison to benchmarks based on single-agent reinforcement learning strategies and traditional machine learning based strategies. **Error! Reference source not found.** attempts to contextualise GARL-based trading strategies in the landscape of trading strategies.

This research addresses a significant gap in the field of algorithmic trading by applying GARL to stock trading, a novel RL approach that has shown promise in other domains. Unlike other multi-agent RL approaches, GARL enhances both group and individual agent performance, and can model knowledge sharing between agents trading on different or same stocks. The study is expected to improve trading performance and provide insights into effectiveness of GARL in capturing complex market dynamics.

Additionally, this study opens up possibilities of future research in optimizing knowledge-sharing techniques and member agent algorithm combinations within the GARL framework.

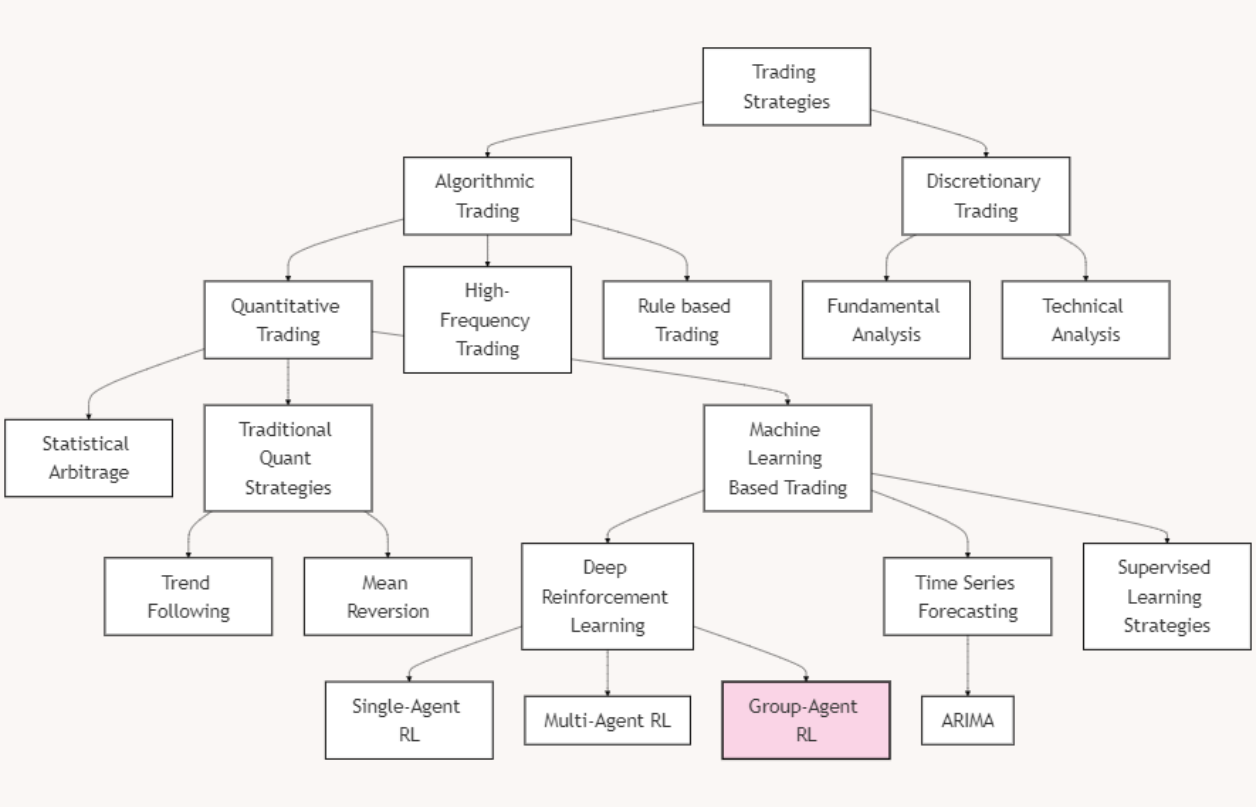

Figure 2: A non-exhaustive graph of the trading strategy landscape

1.2 Aims and Objectives

This project investigates the effectiveness of a group-agent reinforcement learning based trading strategy. The key objectives are:

1. Identify gaps in existing literature on the application of RL in stock trading, and assess effectiveness compared to traditional methods.
2. Investigate the application of GARL in stock trading, identifying the specific use cases and challenges it addresses that existing RL algorithms are unable to resolve.
3. Design, implement, and fine-tune a trading strategy based on the GARL model.
4. Develop two baseline trading strategies, one traditional machine learning and one RL based, to serve as benchmarks for the GARL strategy.
5. Assess the viability of the strategies by testing them in a realistic simulated trading environment with historical stock price data.
6. Analyse, interpret, and evaluate the performance and returns of the GARL approach and compare them to those of the two baseline strategies based on the test results.
7. Reflect on the impact and implications of the analysis, and outline potential directions of future work.

1.3 Project Scope

This project focusses exclusively on the US stock markets, particularly on daily trades aimed at short-term returns. The scope encompasses:

Data and market: The project utilised daily stock price data of large cap US companies listed on the Nasdaq-100 index, including Google, Apple, and Microsoft. Large-cap stocks were chosen due to their stability, low volatility, and high liquidity, enabling easy entry and exit. Similar stocks were chosen for the potential for a fair comparison of performance across stocks.

Trading timeframe: The project aims to develop strategies for short-term gains, with forecasting and trading activities conducted on a daily basis.

Methodology: The trading strategies developed were limited to machine learning and deep reinforcement learning based approaches.

Environment: A custom RL environment for stock trading was designed and set up.

GARL design: Synchronous variants of GARL algorithm were explored, as only improvement in performance of trading is expected, and not that of computation.

Training: The strategies were trained on market data spanning a 20-year period from 2001 to 2020.

Testing: All the strategies were backtested over a three-year trading period from 2021 to 2023 on actual historical stock data.

Evaluation: The performance of all the developed strategies was assessed and compared by calculating the total returns generated, among other evaluation criteria.

1.4 Dissertation Structure

The remainder of this dissertation is organised into the following chapters:

- **Chapter 2: Background and Theory:** This chapter provides a comprehensive overview of the foundational concepts, background, and theories underpinning the project. It reviews existing literature on market hypotheses and algorithmic trading strategies, with a particular focus on the background and application of machine learning and deep reinforcement learning approaches. The chapter dives deep into the concept of GARL, formulating and contrasting it with other multi-agent and ensemble RL algorithms. It contextualises the GARL-based trading strategy, reviews related work, and offers a critical analysis of its effectiveness and relevance.
- **Chapter 3: Research Methodology:** This chapter outlines the methodology employed in developing and testing the trading strategies. It details the processes of data collection and preprocessing, RL environment design and setup, model architecture design, model training, tuning, and testing, as well as the trading simulation and backtesting process. Additionally, the chapter discusses the rationale behind the decisions made at each step of the process.
- **Chapter 4: Results and Discussion:** This chapter discusses the evaluation criteria and examines the results of the trading simulation. It benchmarks and compares the performance of GARL-based strategies against baseline strategies, discussing the factors affecting performance and their implications.
- **Chapter 5: Summary and Conclusion:** The final chapter summarises the key outcomes of the project, reflecting on the successes and limitations encountered. It highlights potential areas for improvement and future research, as well as the broader implications of the project's outputs and contributions to the field of algorithmic trading.

Chapter 2

2 BACKGROUND AND THEORY

2.1 Chapter Overview

The first part of the chapter provides a quick overview of essential market theories and hypotheses that lay the foundational assumptions about how markets behave, serving as the basis for the trading strategies developed in this project. The following sections cover time series forecasting, specifically the ARIMA algorithm, and single-agent reinforcement learning algorithms, both of which are used to create the baseline strategies for benchmarking. Following this, related work using ensemble RL and multi-agent RL based trading strategies are explored and then finally GARL is discussed in detail and contrasted with other RL scenarios. A critical analysis of each of the above techniques is presented in each part.

2.2 Market Theories

The Efficient Market Hypothesis (EMH) proposed by Eugene Fama is the cornerstone of modern financial theory. It claims markets are "informationally efficient", i.e. asset prices in a market reflect all the information at all times, and so always trade at their true value [17]. Burton G. Malkiel in his book further builds on this, by arguing that asset prices follow a random walk, i.e. future price change is independent of the historical price changes [18]. This implies that it is impossible for a trader to consistently "beat" the market.

These theories have been highly controversial and contested over the years. Other than just empirical evidence against it in the form of market bubbles and crashes, insider trading cases and existence of high performing investors like Warren Buffet, there have been various studies showing the existence of various market anomalies and patterns. One such pattern has been the January Effect, which posits that small-cap stock prices tend to rise more in January than other months [19]. Behavioural finance is a field that tries to understand the role investor psychology and market sentiment plays in trading [20], [21]. It challenges the assumption EMH makes that investors always act rationally, and can explain market anomalies like bubbles and crashes. It addresses cognitive biases like herd behaviour and aversion to loss and other emotional influences like fear and greed.

Perhaps the most complete picture of market behaviour is provided by the Adaptive Market Hypothesis (AMH). It challenges EMH's static view of the market, instead proposing that the markets have varying levels of information efficiency, as they evolve over time and with the introduction of new financial instruments, technological advancements, regulatory changes and shifts in investor sentiments and cognitive capabilities [22]. AMH incorporates some insights from behavioural finance, and is able to explain market anomalies and the existence of patterns like the January effect, as well as market bubbles and other phenomenon caused by investor irrationality. In conclusion, AMH and behavioural finance theories suggest that profit opportunities can exist in markets, even if temporary.

Such profit opportunities are identified by traders through two primary forms of analysis: fundamental and technical. Fundamental analysis are methods to quantify the intrinsic value of a company based on its financial data [23]. In contrast, technical analysis focuses on identifying price movement patterns and predicting future trends based on historical data [24]. The algorithms developed in this project can be viewed as technical analyses driven by computer systems.

The Elliot Wave theory is a foundational technical analysis concept that describes the nature of market patterns [24]. It is based on the idea that markets move in predictable cycles (sometimes referred to as market cycles), driven by investor psychology. It further postulates that market prices move with respect to an overall direction (trend), with a wave structure. Elliot waves are fractal in nature over different time frames. The fractal market hypothesis mentioned in Section 1 builds on this same idea [14]. These concepts have formed the basis of market forecasting models. Forecasting based trading strategies, like the time series forecasting based strategy created as a baseline in this project, use these models to predict market movements, identify trading opportunities and capitalise on it by buying or selling at a predicted dip or peak.

2.2.1 Technical Indicators

Technical indicators are an essential component of technical analysis toolkit, and are used by traders as signals to identify patterns in the market and potential entry and exit points for trades. While a wide variety of indicators exist, a few are discussed below.

Trend indicators

Trend indicators are used to identify if the asset prices are trending in a direction or moving randomly.

- Average Directional Index (ADX) [25] identifies trends, their intensity and direction.
- Aroon Indicator [25] identifies trend changes by measuring time between highs and lows.
- Detrended Price Oscillator (DPO) [26] estimate price cycles by measuring time between sequential peaks (or troughs).
- Simple Moving Average (SMA) aggregates the average closing price during a lookback period.
- Exponential Moving Average (EMA) [24] is similar to SMA, except that it is weighted heavily on recent prices.

Volatility indicators

Volatility indicators measure how much and how often the price fluctuates and are used to adjust for risk and identify breakout opportunities.

- Bollinger bands [24] are a set of three lines calculated from the SMA, and the width between the bands indicate the volatility of the asset. Wide bands indicate high volatility and low bands indicate low volatility, and the lower and upper boundaries act as resistance levels.
- Average True Range (ATR) [24], [25] measures the volatility of the asset. It calculates the average of a set of price ranges over the last 14 days.
- Keltner Channel [24] are another set of bands that use ATR to signal trends and breakouts.

Momentum indicators

Momentum indicators measure the speed and strength of price movements, and are used to identify potential turning points of trend directions.

- A simple momentum indicator is the change in price in the past 10 days.
- Commodity Channel Index (CCI) [24], [27] is used for spotting new trends by measuring the difference between current and historical mean prices.
- Relative Strength Index (RSI) [24], [25] provides overbought or oversold signals, indicating a possible trend reversal.
- Moving Average Convergence/Divergence (MACD) [24] measures the relationship between two EMAs to identify possible trend reversals and is usually used with RSI to confirm oversold or overbought conditions.
- Williams percent range, a.k.a Williams %R [24], [28] spots oversold/overbought conditions by locating the current closing price with respect to the price in the past 14 days It is typically used with RSI and Bollinger bands for confirmation.
- Stochastic Oscillator [24] is similar to Williams %R but adds additional smoothing to reduce volatility.
- Rate of Change (ROC) [24] calculates the percentage change between current price and the price 10 days back. It is used to spot divergences and crossovers.

Volume indicators
Volume indicators are important to identify strong and weak price movements.

- On-Balance Volume (OBV) [24] confirm trends by measuring positive and negative volume flow.
- Volume Weighted Average Price [29] is calculated as the average price an asset has traded at in a day weighted by volume, giving a better view of the price changes in the day.

- Accumulation/Distribution (A/D) [29] determines if the asset is being accumulated (bought) or distributed (sold) to signal buying and selling pressure in the market.
- Chaikin Money Flow (CMF) [29] signals the flow of money into or out of an asset in the last 20 days. It uses the A/D line in its calculation.

Performance indicators
Performance indicators signal how well the asset is doing recently.

- Log Return is the log of the return in the past 20 days.

The technical indicators discussed span the breadth of the most important types of technical indicators. The also work well together and are commonly used in conjunction with each other to confirm speculations of trend, momentum and volatility. This project makes use of these 20 indicators to inform the trading strategies with comprehensive market analysis, allowing for informed decision making by using the signals to identify potential opportunities and optimize entry and exit points.

2.3 Time series forecasting and trading

A popular approach to market forecasting is to treat the historical market data as a time series. A time series is sequence of data points (here, stock prices) ordered across time. Time series forecasting is a statistical method to uncover underlying patterns in the data and predict future values. Prior to forecasting, a time series is decomposed into its components –

- Trend (**T**): long-term direction in the data
- Seasonality (**S**): regular patterns in the data at fixed intervals
- Cycles (**C**): long term, non-periodic oscillatory movements
- Noise (**N**): random unpredictable fluctuations

A time series $\boldsymbol{Y}(t)$ can be represented as an additive or multiplicative model of the decomposed components [30].

$$Y(t) = T(t) + S(t) + C(t) + N(t) \text{ [additive model]} \quad \textbf{(2.1)}$$

$$Y(t) = T(t) \times S(t) \times C(t) \times N(t) \text{ [multiplicative model]} \quad \textbf{(2.2)}$$

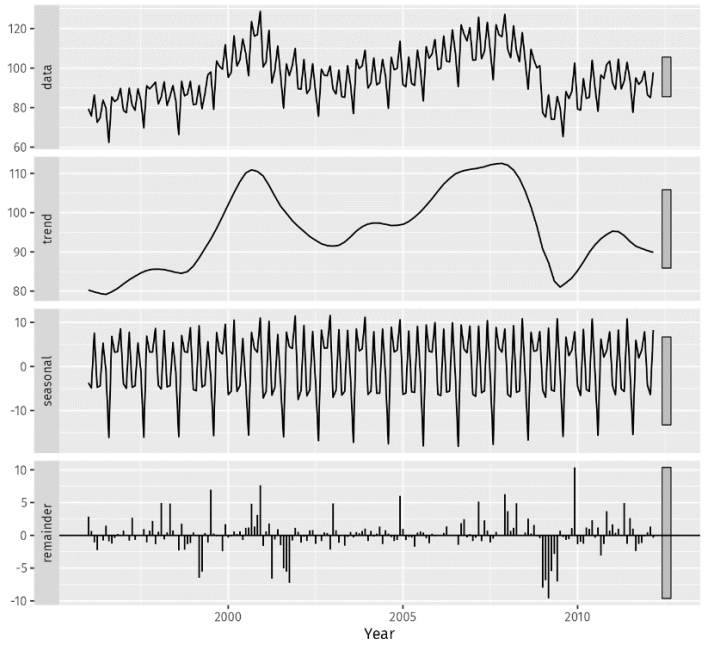

Figure 3: A sample time series split into its three additive components [31]

2.3.1 Auto Regressive Integrated Moving Average (ARIMA)

Auto Regressive Integrated Moving Average (ARIMA), also known as Box-Jenkins model after its creators, is a popular univariate time series forecasting method [30]. It is a linear combination of Autoregressive (AR), Integration (I) and Moving Average (MA) models. Unlike typical time series methods, it does not explicitly decompose the series into its components, instead, it assumes that the time series is weakly stationary or can be transformed into one, and then models the remaining structure. A stationary series is one where the mean, variance and autocovariance are time-invariant.

The Autoregressive model of order *p*, AR(*p*), regresses the series on its own *p* lagged values.

$$Y_t = c + \epsilon_t + \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \cdots + \phi_p Y_{t-p}$$

(2.3)

where,

Y_t is the value at time t

Φ_k are p autoregressive parameters

ϵ_t is the error term at time t

c is a constant

The Integration model of order *d*, I(*d*) attempts to make the time series stationary through differencing the values with past values *d* times. The recursive form of the differencing can be written as -

$$Y'_t = Y_t - Y_{t-1}$$

(2.4)

The time series is differenced until it passes the Augmented Dickey-Fuller test, which is a statistical test used to determine if a time series is stationary [32].

The Moving Average model of order q, MA(q) models the error in the prediction as a linear combination of the previous error terms -

$$Y_t = \mu + \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \cdots + \theta_q \epsilon_{t-q}$$

(2.5)

where,

μ is the mean

θ_k are the q moving average parameters

ARIMA(p,d,q) model, can be written as a linear combination of equations (2.3), (2.4) and (2.5)

$$Y'_t = c + \phi_1 Y'_{t-1} + \cdots + \phi_p Y'_{t-p} + \epsilon_t + \theta_1 \epsilon_{t-1} + \cdots + \theta_q \epsilon_{t-q}$$

(2.6)

Seasonal Auto Regressive Integrated Moving Average [30], denoted as SARIMA(p,d,q)(P,D,Q)$_m$ is an extension of ARIMA that incorporates additional seasonal terms to capture patterns that repeat over a fixed period. It does that by simply introducing additional lags that are multiples of a seasonal period m, and calculate AR(P), I(D) and MA(Q) components. This project uses the extended SARIMA model.

The data is fitted to the model by doing a search over a range of possible parameter values and calculating the Akaike Information Criterion (AIC) [33] value to pick the best combination. AIC is a statistical measure that uses the principle of maximum likelihood estimation to fit the model to the data, producing lower values for better fits.

ARIMA suffers from many limitations due to its simplicity and assumptions it makes –

1. *Stationarity*: ARIMA assumes that the time series is stationary or can be turned into one. This might not be possible for all stock price data.
2. *Constant variance:* ARIMA assumes that the variance of the time series remains constant over time, but this might not hold true. Studies have shown that periods of high volatility are followed by more periods high volatility and vice versa. This is known as volatility clustering and is common enough to be called a "stylized fact" [34].
3. *Linearity*: ARIMA uses linear relationships to model the data, but stock market data can often exhibit non-linear patterns.
4. *Univariate:* Stock markets often involve interactions with other assets and indicators, and these cannot be modelled in the univariate framework of ARIMA.
5. *Short-term accuracy:* The predictive power of ARIMA models are generally seen to be short-term. Techniques such as rolling window ARIMA might be needed for long term predictions.

This project uses a SARIMA based trading strategy as a baseline for this project because of the relative simplicity of the model and prior success in market forecasting [35]. It has shown to outperform even advanced deep learning techniques in cryptocurrency markets [36], and shows promise in returning comparable performance in stock markets.

A trading strategy is designed to use the SARIMA predictions as signals, comparing the current stock prices to forecasted prices, to identify potential trading opportunities when there is a significant discrepancy between them. The technical specifications of the trading strategy developed in this project will be described in Section 3.3.2**Error! Reference source not found.**.

2.4 Reinforcement Learning and Trading

Modern portfolio theory (MPT) [37] describes a utility function, which represents an investor's satisfaction from their portfolio, and helps manage their risk tolerance.

$$U = E(W) - 0.5 \times A \times \sigma^2$$

(2.7)

where,

U is the utility function

A is the risk aversion coefficient

σ is the variance of the portfolio

E(W) is the expected wealth or return of the portfolio

The performance of the final portfolio value depends on the sequence of actions the investor makes. For a time period *T*, the investor must make decisions that balance both immediate and future rewards, and maximize the net reward at the end of time T. If we assume that the investor is risk neutral, the utility function becomes a simple linear cumulation of returns (See equation (2.8))[38].

$$E[U(W_T)] = E\left[U\left(W_0 + \sum_{t=1}^{T} \delta W_t\right)\right]$$

(2.8)

This takes the shape of a classical sequential decision-making process. Reinforcement learning (RL) is a semi-supervised or self-learning algorithm that learns to make optimal decisions from sequential interactions.

A typical RL system consists of *agent* (an autonomous learner) in a well-defined *environment* (the world in which the agent exists). It performs some *action* (from a set of actions/decisions that the agent can take), which causes a change in the *state* (configuration of the environment at a point in time), and is subsequently given some *reward* (positive or negative feedback given to the agent based on how well its doing). The agent's goal is to maximize such rewards over time, and it learns the best strategy to do that (also known as the *policy*) by repeatedly interacting with the environment in a feedback loop (See Figure 4). An *observation* is the part of the environment state that is visible to the agent at a time.

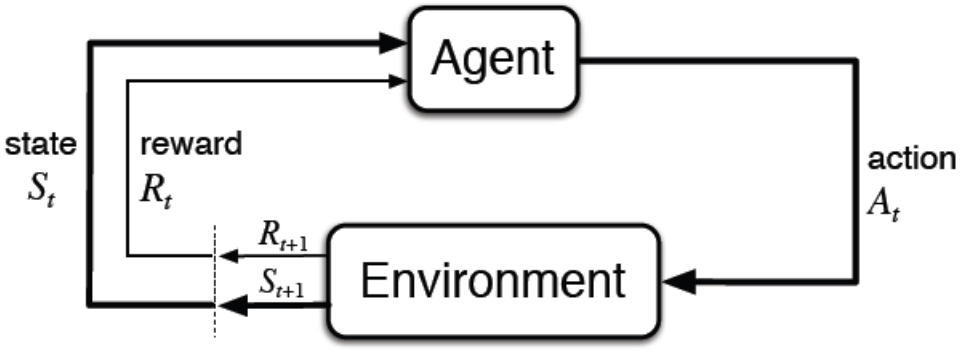

Figure 4: Agent-environment interaction feedback loop in RL [39]

RL is typically modelled as a Markov Decision Process (MDP) [40]. An MDP is a tuple

$$(S, A, P, R, \gamma)$$

(2.9)

where,

S is a finite set of *states*

A is a finite set of *actions*

$P(s'|s,a)$ is the state transition probability from a state to the next when the agent takes an action a

$R(s,a)$ is the reward function

$\gamma \in [0,1)$ is the discount factor, used to determine the importance of future rewards.

The Reward Hypothesis claims that all goals that the agent wants to achieve can be represented as a maximization of the expected cumulative reward over time [5]. Supposing that a RL agent runs for a time T, at a time step *t*, the RL agent tries to maximize the expected discounted cumulative reward G_t (See Equation (2.10)) .

$$G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$$

(2.10)

In the context of trading, it can be seen that optimizing *E(G)* in Equation (2.10) is equivalent to optimizing the expected wealth $E[U(W_T)]$ in Equation (2.8), if we consider the portfolio return or wealth W as the reward R [6]. This is in line with the Reward Hypothesis claim, and shows that reinforcement learning can be used for trading by modelling the goal as a maximization of the portfolio return.

The RL algorithm learns through a trial-and-error approach to identify and capitalise on trading opportunities. The *environment* is the stock market and the prices of the financial securities as they move over time, the *agent* is a virtual trader starting out with a portfolio with some fixed amount of capital, and the *actions* are typically a discrete set of buy, hold or sell orders (though it could also be more complex risk managed actions like stop-loss order). The *agent* executes the order at a timestep *t*, which changes portfolio value at the next timestep. The *rewards* (or penalties) it receives are the profits (or losses) made by the portfolio (See Figure 5). Note that all the actions, including hold may lead to different portfolio values due to the changing stock prices. A detailed technical description of the environment, actions and reward functions used in this project is given in Section 3.

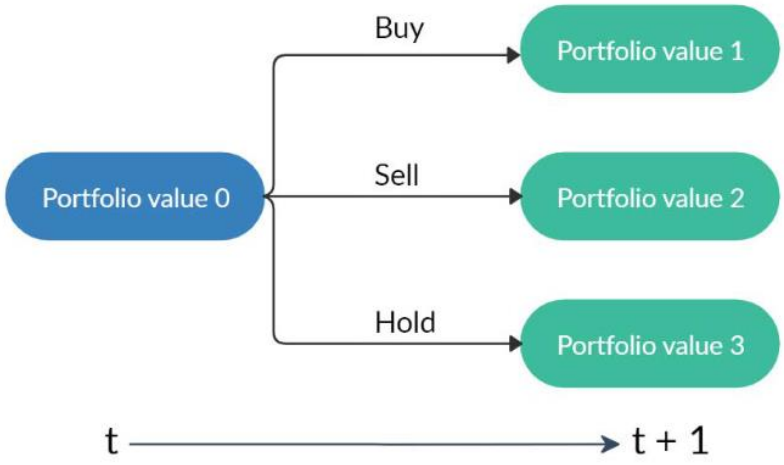

Figure 5: Portfolio value transition with actions ∈ {buy, hold, sell} [10].

2.5 Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) combines the principles of reinforcement learning and deep learning by using deep neural networks to approximate the decision making functions in reinforcement learning [5]. Deep learning provides several advantages over traditional machine learning and time series forecasting such as automatic feature development. One example, is that time series forecasting are capable to predict for a short term (of just a few days), but stock market trends can persist for much longer with some consolidation periods [6], and sometimes market cycles last for years. Such patterns can be modelled better with deep reinforcement learning models, as they can make trade decisions directly instead of making explicit forecasts.

The decision-making functions of a RL algorithm could either be the value functions or the policy. On this basis, RL algorithms are broadly classified into three approaches namely value optimization, policy optimization, and hybrid approaches**.**

2.5.1 Value Optimization Methods

Value optimization algorithms estimate the expected future rewards for being in a state and take actions that maximize this value. Deep Q-Network (DQN) is a value optimization deep reinforcement learning that builds on the traditional Q-learning RL algorithm [41], using deep neural networks instead of a Q-table to approximate the Q-function [42], [43]. The Q-function $Q(s, a)$ represents the expected cumulative reward for taking an action a in state s, and is a mapping from state-action pairs to their expected future rewards.

2.5.2 Policy Optimization Methods

Policy optimization methods learn a policy function, $\pi(a \mid s)$, that maps states to a probability distribution of actions. Proximal Policy Optimization (PPO) is a policy optimization method that uses deep neural networks to approximate the policy function [44]. It uses some optimization techniques such as performing multiple epochs of minibatch updates, to a clipped "surrogate" objective function, rather than performing one gradient update per data sample as in standard policy gradient methods. The surrogate objective function is used to limit the size of the policy updates.

2.5.3 Hybrid Approaches

Advantage- Actor-Critic (A2C) and its asynchronous variant, Asynchronous Advantage Actor Critic (A3C) are hybrid approaches that combine the strengths of actor-critic and advantage learning methods [45]. It uses a policy network (actor) to predict actions and a value network (critic) to estimate state values. The advantage function is the difference between the estimated value of a state-action pair and the average value of all possible actions in that state.

All three algorithms work with discrete action spaces, and so can be used for stock trading. They have shown to outperform time series forecasting based trading strategies in the futures market [6]. While A3C and PPO are more advanced and general methods, DQN shown relatively better performance in the US market [46]. This project will use one of each the above-mentioned approaches, namely DQN, PPO and A2C to model baselines for single-agent RL strategies.

2.6 Multi-Agent Reinforcement Learning

Multi-agent Reinforcement Learning (MARL) is an extension of RL. They can be classified into cooperative, competitive, and hybrid approaches – which describe the nature of interaction between the

asynchronous autonomous agents. The agents themselves typically use any of the single agent RL algorithms.

Like its single-agent counterpart, multi-agent reinforcement learning (MARL) can be modelled as a generalization of MDP tuple [47].

$$(S, A_1, \cdots, A_n, P, R_1, \cdots, R_n, \gamma) \quad \textbf{(2.11)}$$

where,

n is the number of agents

A_i are action sets of n agents

P_i are reward functions of n agents

shared set of states S, state transition probability P and discount factor γ

The agents are therefore necessarily in a common environment.

In the domain of trading, multi-agent trading strategies generally benefit from a cooperative approach, as it is more beneficial for the agents to collaborate to maximize returns.

An ensemble multi-agent reinforcement learning strategy has been proposed that combines three actor-critic algorithms (PPO, A2C and DDPG) to invest in 30 Dow Jones stocks and has proven to outperform the respective individual algorithms (See Figure 6) [10]. Another ensemble method involves training multiple agents using a split feature space approach, where each agent learns from a subset of features [48]. In both of these, however, each individual algorithm generates forecasts for the identical stock. The ensemble mechanism then determines which algorithm is most accurate by applying an evaluation metric (such as the Sharpe ratio) during a validation period that precedes the trading period. Additionally, they lack any form of collaborative learning beyond the mere selection of the optimal alternative. In that sense they are just extensions of single-agent RL and don't share any knowledge.

A MARL setup trading in foreign exchange markets involving collaboration among multiple agents in a hierarchical structure has been proposed, where each agent (running a DQN algorithm) is an expert trader for a specific timeframe and the knowledge flows from the agents trading at higher timeframes to those at lower timeframes [11]. the trades are only executed by the agent at the lowest timeframe from collective knowledge, and the knowledge flow is unidirectional. Moreover, due to the particular set up of the method, knowledge from particular assets across different time frames is shared, and there is no scope for sharing across different stocks or different environments. Therefore, they lack flexibility in terms of the kind of knowledge that is shared and in the kinds of securities that can be selected.

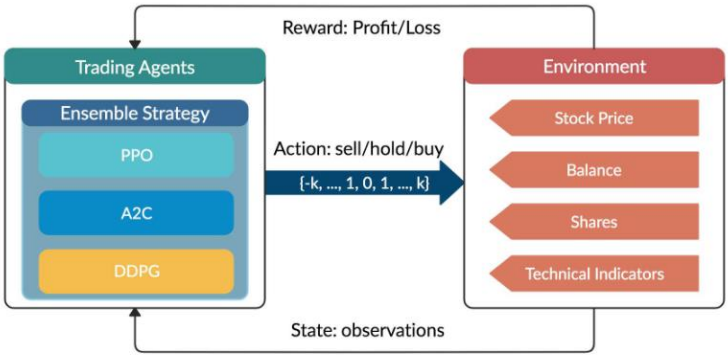

Figure 6: Overview of an ensemble multi-agent RL strategy [10]

2.7 Group-Agent Reinforcement Learning

Group-agent Reinforcement Learning (GARL) was introduced by Kaiyue Wu and Xiao-Jun Zeng in 2023 as a third paradigm of reinforcement learning, distinct from single-agent and multi-agent RL. It proposes multiple geographically distributed agents performing separate tasks in separate environments, cooperating with each other by sharing knowledge [13]. Inspired by social learning theory, the setup is similar to how humans learn from their peers in a social context, like in a classroom or study-group setting, drawing from their diverse experiences to share knowledge [49].

GARL can be seen as a further generalization of the MDP tuple in (2.11)

$$(S_1,\cdots,S_n,A_1,\cdots,A_n,P_1,\cdots,P_n,R_1,\cdots,R_n,\gamma_1,\cdots,\gamma_n,K_1,\cdots,K_n,K-1,\cdots,K-n) \quad \textbf{(2.12)}$$

where,

n is the number of agents

$S_i, A_i, P_i, R_i, \gamma_i, K_i$ are the states, action sets, state transition probability, rewards, discount factors and internal knowledge respectively for the *n* agents

$K_{-i} = \{K_{1,i}, \dots, K_{i-1,i}, K_{i+1,i}, \dots, K_{n,i}\}$ is the set of knowledge shared by an agent to another, and each $K_{i,i'} \subseteq K_i$ [13]

Group-agent reinforcement learning (GARL) is different from MARL in that the environment states, pace, and goals of the agents can be varied, and they learn from each other by sharing knowledge [13]. Unlike MARL, the agents do not directly interact with each other, and the separated environment implies that there is no competition for resources or completion of tasks in a common environment.

The authors propose a framework of the GARL paradigm called Decentralized Distributed Asynchronous Learning (DDAL)[13], that features asynchronous communication of shared knowledge among homogeneous agents. The asynchronous nature of communication has additional benefits of parallelizability, and so can scale well in practical real-world applications. It also allows for flexibility and autonomy of the agents in terms of when and how they share knowledge and learn from other agents. Experimentation with the proposed method. Since the group contain homogeneous agents, i.e. agents of the same RL algorithm with similar neural networks, the gradients can be shared among the agents, and is indeed the form of knowledge that is shared. Specifically, the received gradients are averaged, weighted on the learning experience so far and the relevance of the gradient to the receiving agent (. The weighting allows to mitigate the influence of poor experiences and irrelevant agents.

$$\bar{g} = \frac{1}{2}\left(\sum_{j=1}^{m}\frac{T_j}{\sum_{j=1}^{m}T_j}g_j + \sum_{j=1}^{m}\frac{R_j}{\sum_{j=1}^{m}R_j}g_j\right) \quad \textbf{(2.13)}$$

where,

g_j is the jth piece of shared gradient among *m* gradients

T_j is the learning experience (number of epochs) associated with g_j

R_j is the relevance of g_j to the agent

They further experiment with a group of A2C agents with weighted gradient sharing, called Decentralized Distributed Asynchronous Advantage Actor Critic (DDA3C) and apply the algorithm on

the classic RL task CartPole-v0. The results show that the decentralized asynchronous learning leads to more stable learning among different group sizes, and robust performance even when some agents in the group underperform.

In the context of stock trading, the GARL approach allows the agents to trade on different stocks and options at different timescales (which may or may not be related and so can be modelled as their own environments), and they can transfer knowledge asynchronously that can potentially help each other achieve their respective goals. This also leaves a lot of flexibility in terms of variations in implementation, knowledge sharing and choice of individual agent algorithms. The specific approach taken is detailed in Section 3.5.2.

2.8 Summary

This section introduces the reader to the basics of algorithmic stock trading, the driving theories behind its success and the common machine learning and time series forecasting based trading strategies prevalent in the market. It dived into various RL algorithms (specifically, DQN, PPO and A2C), the MARL paradigm and introduced the GARL paradigm of RL. It examined the numerous studies that apply RL algorithms to stock trading, as well as critically analysing the strengths and limitations.

Single-agent RL algorithms struggle to work with complex action spaces, and so addressing the multidimensional aspect of stock trading, like asset selection, timing, and interaction with external signals might become difficult. MARL algorithms addresses some of these issues, by letting multiple agents work on different tasks and aggregating their efforts together in a cooperative or ensemble techniques. Still, the flexibility of MARL algorithms is severely limited, as they run on the same environment by definition. Therefore, the agents do not have the flexibility to work on different environments, like different stocks, or different trading timescales. When they do work on similar environments, the resulting ensemble setup do not affect individual learning. GARL addresses this gap by providing a flexible framework to model multiple agents in different environments, while still promising learning benefits from cooperation.

In the following chapters, GARL trading strategies are further explored to address some of these gaps.

Chapter 3

3 RESEARCH METHODOLOGY

3.1 Chapter Overview

This chapter details the research methodology and technical details of the project. It will describe the data sources, data preprocessing, system architecture, environment design, and the processes involved in model training, tuning, and testing the baseline models, and then subsequently of the GARL model. Additionally, the ethical, professional and risk considerations are discussed.

The following research methodology has been proposed considering the need for a systematic approach to developing and evaluating a reinforcement learning based trading strategy. Figure 7 provides an overview of the same.

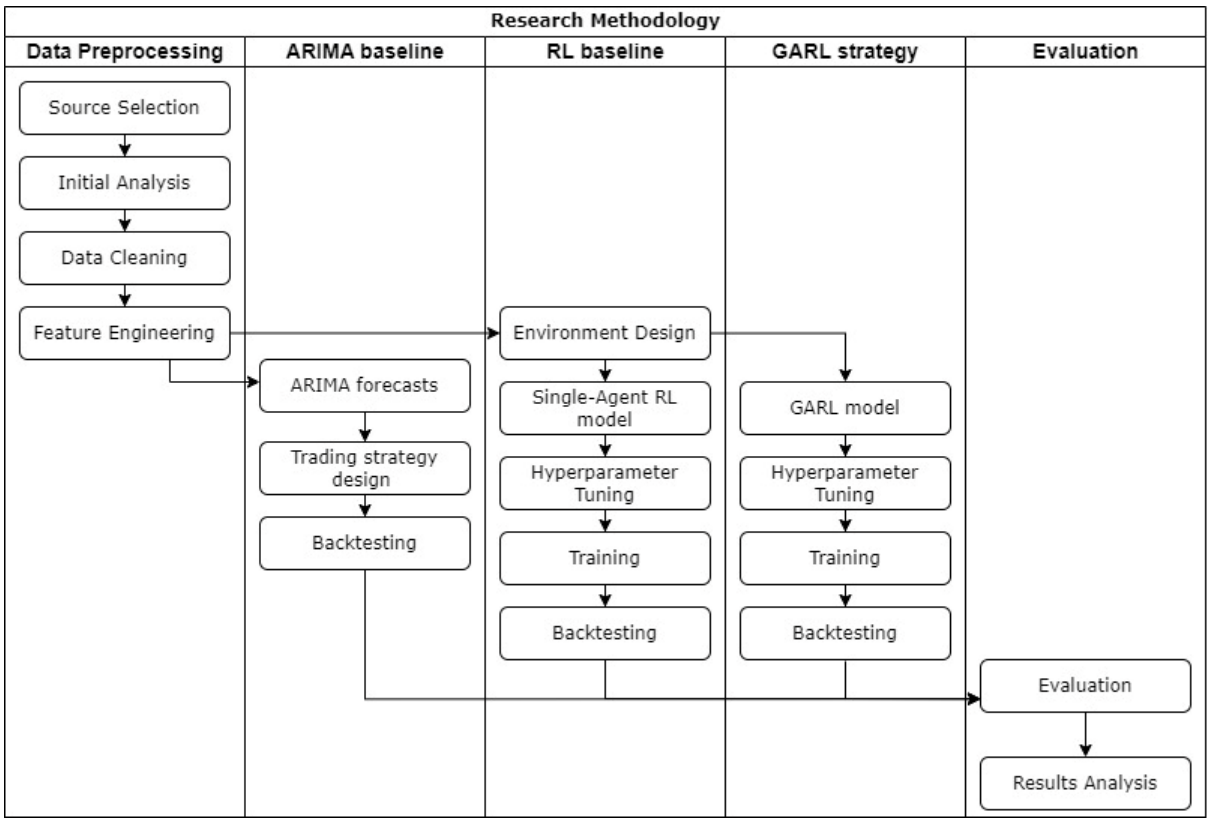

Figure 7: Research Methodology Overview

3.2 Data Preprocessing

The stock market data needs to be collected, cleaned and transformed into an appropriate format before it could be used for training machine learning and reinforcement learning models. This project used

Pandas[1], an open-source library that has rich features for data analysis and manipulation, for creating the data processing pipeline.

3.2.1 Data Source

Historical US stock market data is publicly available via many sources. Yahoo Finance[2] is selected as the source for this project, as it provides quality live data over the past few decades, is publicly and freely available through an API. It contains all the essential features such as Open, High, Low, Close (OHLC) pricing and the trading volumes required for analysis. The project used an open-source library called yfinance[3] to interact with the Yahoo Finance API. Daily stock pricing data between 2001-2023 was collected for Google, Apple and Microsoft stocks.

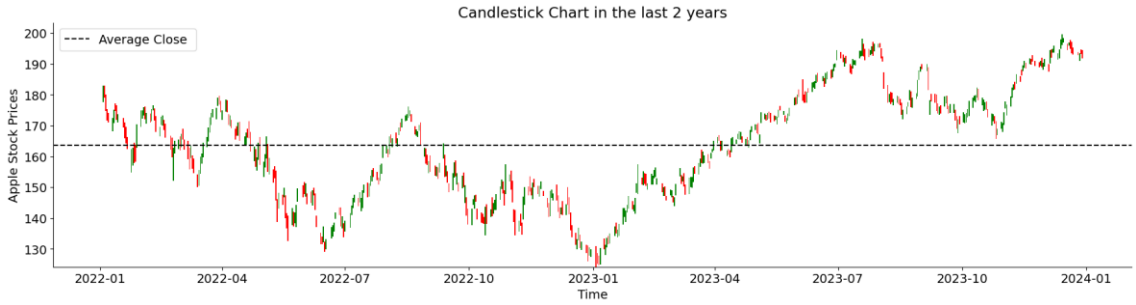

Figure 8: Apple stock prices candlestick chart between 2022-2023

3.2.2 Data Analysis

Some initial data analysis was performed on the data to acquire an intuition for the patterns in the data.

Figure 9 shows the day-wise analysis for the Apple, Microsoft and Google stocks between 2019-2023. The top row shows the day-wise price percentage change statistics, specifically the mean and inter-quartile range (IQR). First, it is obvious that the statistics across the stocks have similar shape, which is as expected considering the similarity of the stocks. It is observed that the daily percentage change in closing prices on Mondays is slightly higher than others for Apple, lower for Google and remains the same for Microsoft. This shows that each stock series can have its own patterns. The bottom row in Figure 9 shows the day-wise average volume of the three stocks and it is observed to have a slight increase regularly on Fridays.

[1] https://pandas.pydata.org/
[2] https://uk.finance.yahoo.com/
[3] https://github.com/ranaroussi/yfinance

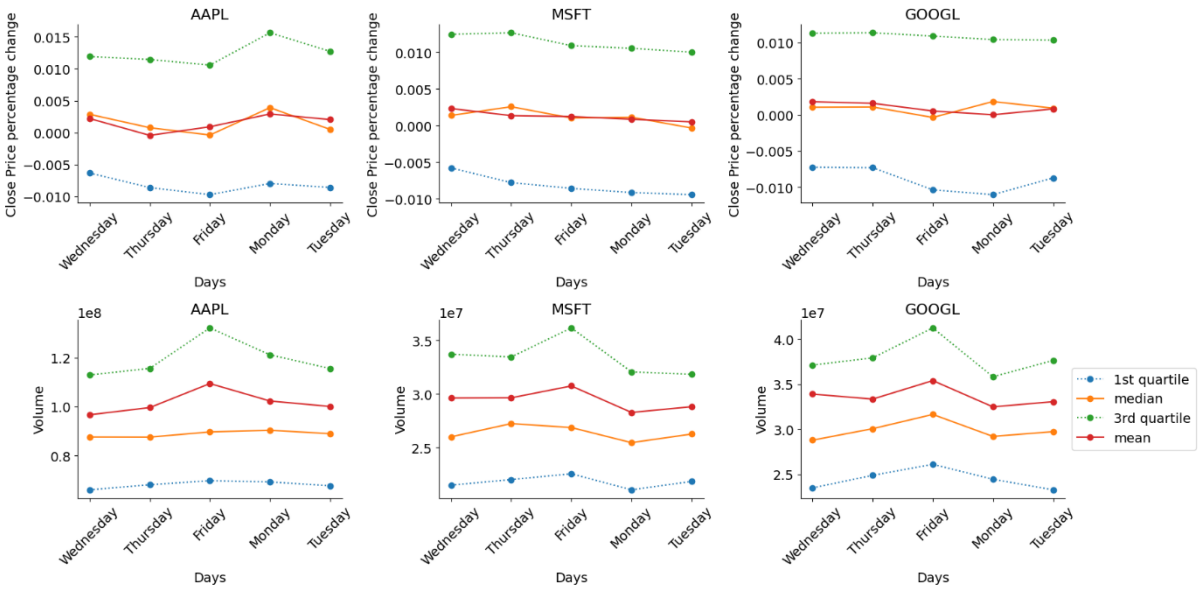

Figure 9: Daily price & volume statistics through 2019-2023 for Apple, Microsoft & Google.

Similarly, the month-wise statistics for the stocks are analysed in Figure 10. The net price movements in every month from 2013-2023 are calculated and plotted. It is observed that the stocks consistently perform well around June-July and take a turn for the worse in August through the years. The narrower the band formed by the four statistics, the more consistent the price movements are in those months through the years.

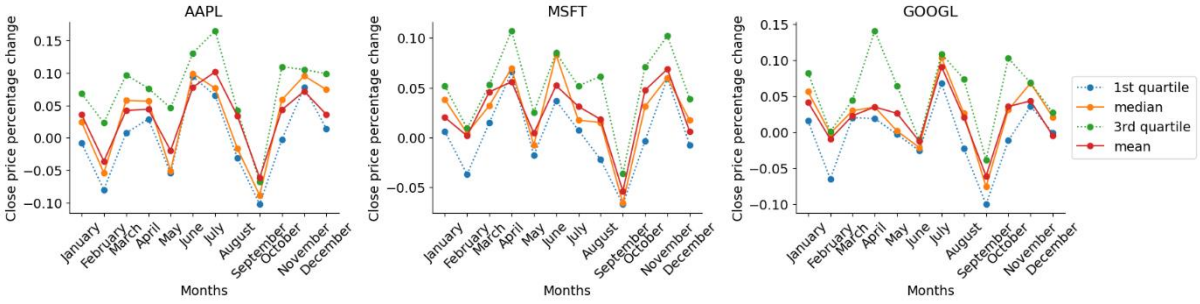

Figure 10: Monthly price statistics through 2013-2023 for Apple, Microsoft & Google.

The analysis presented confirms the presence of market patterns and seasonality in the data. This is exploited by the trading algorithms presented in the project.

3.2.3 Data Cleaning

The data collected from the Yahoo API was already of relatively high quality, with no null values or other such discrepancies. There is no data on the weekends, but that is because the markets are closed. This is unlikely to cause any issues such as sudden spikes or dips, because theories such as the Monday Effect [50] suggest that the trends from Friday prevail through the weekend to Mondays. Besides missing data on the weekends, there were a few additional missing data. Given that stock market data is a time series, these missing values were addressed by linear interpolation. This method involves estimating the missing values based on the known values before and following the missing data point, by fitting a straight line through them.

3.2.4 Feature Engineering

Various feature engineering techniques are commonly applied to multi-variate models to enrich the input data. The raw input data is transformed into new meaningful features, which can potentially improve model performance. The state inputs (or observations) in RL algorithms can be of any shape, and so a 2D matrix formed from a set of features can be used as inputs. Neural networks, in particular work better with large amount of data and features. Since deep RL algorithms use neural networks inherently to learn, and since the stock time series data is quite limited, additional features can be expected to improve model learning.

Technical indicators are essentially calculations based on the price and volume of a stock and so they can be generated from the existing features in the dataset. This project adds 20 of the most popular technical indicators introduced in Section 2.2.1 as additional features for the RL algorithms using an open-source library called pandas-ta[4]. They have been chosen carefully across the different types of indicators to provide the RL algorithm with a diverse set of signals to make more informed trading decisions. The added indicators are ADX, Aroon Indicator, DPO, SMA, EMA, MOM, CCI, RSI, MACD, Stochastic Oscillator, Williams % R, Rate of Change, Bollinger bands, ATR, Keltner Channel, OBV, A/D line, VWAP, CMF and log return (See Section 2.2.1).

The additional numerical features need to be rescaled to achieve consistency across various indicators. As discussed in Section 2.5, RL algorithms use neural networks to estimate their decision-making functions, and since neural networks can form biases towards features with larger scale, feature scaling becomes essential. The indicators added have different units and scales and are assumed to be normally distributed, and so standardization is the preferred scaling technique used. Standardization (a.k.a. Z-Score normalization) transforms the data so that it has a mean of 0 and unit standard deviation (3.1).

$$x'_i = \frac{x_i - \mu}{\sigma} \quad \textbf{(3.1)}$$

where,

x_i is the original data point
μ is the mean of the series
σ is the standard deviation of the series

Lastly, the data is split into training and test datasets, with training period between 2001-2020 and test period between 2021-2023.

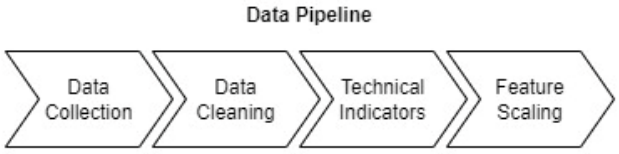

Figure 11: Data pipeline

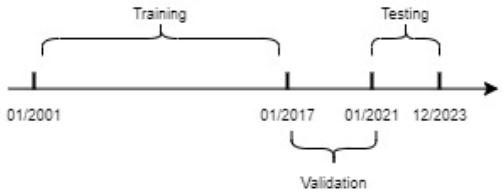

Figure 12: Stock data splitting

[4] https://twopirllc.github.io/pandas-ta/

3.3 Time Series Forecasting Based Trading Strategy

The stock prices are decomposed into trend, seasonality and residuals (noise) components using a multiplicative model provided by the open-source statsmodels[5] library (See Figure 13). A clear seasonality is observed. A trend is also present, which means that the series has to be converted to a stationary one.

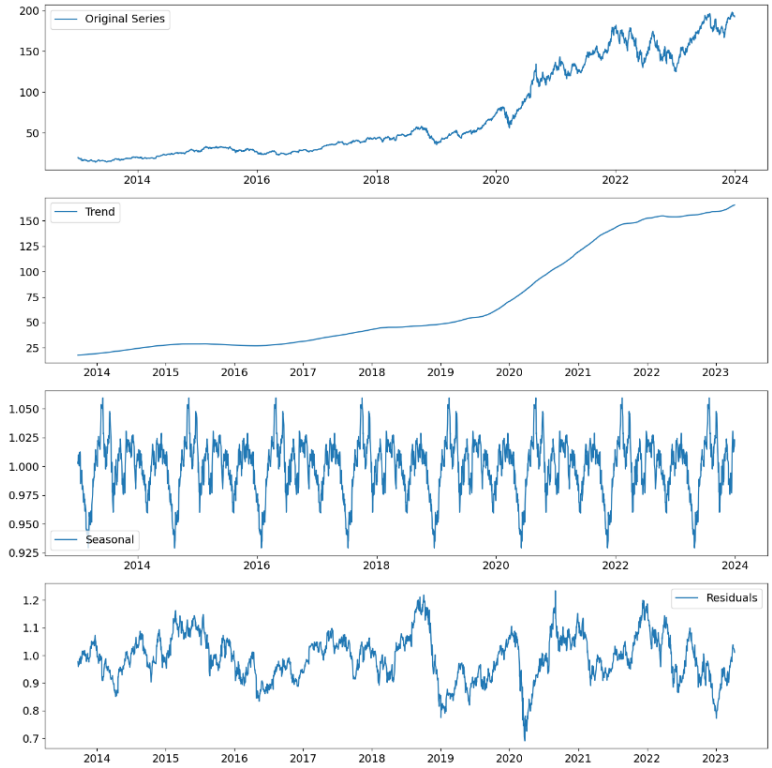

Figure 13: Decomposition of Apple stock prices between 2013-2023

3.3.1 Forecasting using ARIMA

The time series data of the stock prices from 2001-2020 was fitted to a seasonal ARIMA model using pmdarima[6]. Pmdarima is an open-source library that automatically fits the seasonal ARIMA parameters by running a search over its parameter space, performing the Augmented Dickey-Fuller (ADF) test to check for stationarity (returning an error if it cannot be converted into a stationary series) and minimizes the Akaike Information Criterion (AIC) to find the best fit model.

[5] https://www.statsmodels.org/stable/index.html
[6] https://alkaline-ml.com/pmdarima

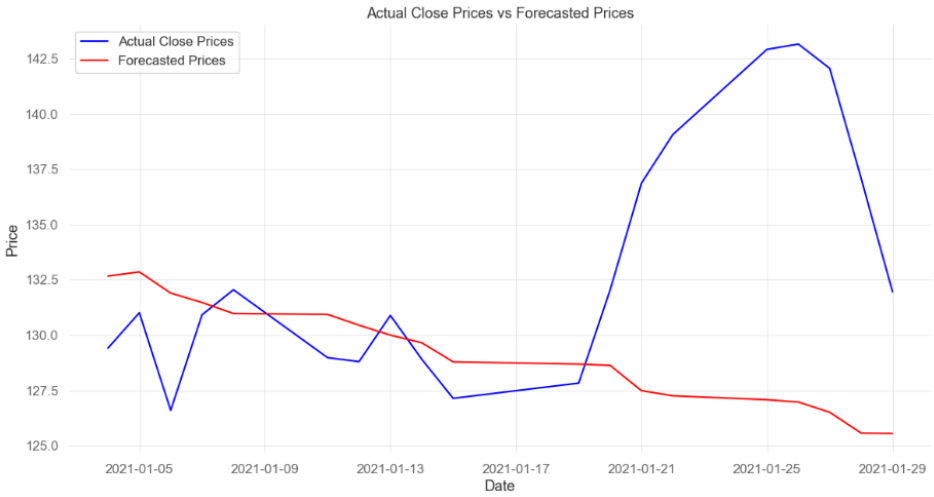

Figure 14: Predicted vs actual Apple stock prices after fitting an offline ARIMA model

Figure 14 shows the model predictions on Apple stock prices after the model is fitted. The predictions deviate significantly after 10 days, showing that the model is only suitable for short-term predictions.

The online ARIMA model (a.k.a. Rolling ARIMA) in contrast, performs very well. This approach involves fitting the ARIMA model to data within a sliding window, continually searching for the best fit model within each window. The model is fitted to a year's worth of data, followed by predicting the prices for the next 10 days. At the end this period, the process is repeated and a new model is fitted again to the last 365 days of data, predictions made again for the next 10 days (See Figure 15). This iterative process allows the model to adapt better to changing data (See Figure 16). The figure shows the predictions after 1 year of learning (2019-2020) compared to actual prices.

```
def forecast_arima(data, lookback_period, forecast_period):
    forecasts = pd.Series()
    for i in range(0, len(data), forecast_period):
        if i + lookback_period < len(data):
            train_data = data[i:i+lookback_period]
            model = auto_arima(train_data, seasonal=False, suppress_warnings=True)
            model.fit(train_data)
            forecast = model.predict(n_periods=forecast_period)
            forecasts = pd.concat([forecasts, pd.Series(forecast.values, index=train_data.index[-forecast_period:])])
    return forecasts
```

Figure 15: The online ARIMA code snippet

Figure 16: Predicted vs actual Apple stock prices after fitting an online ARIMA model

3.3.2 Trading Strategy Design

A trading strategy was developed that utilizes the forecasted prices as a signal. The strategy starts with some fixed amount of capital and executes simple buy, hold or sell orders. A buy order means that the strategy invests all of the capital in the asset. A sell order means it exits the market by liquidating it entirely into cash. Alternatively, it can hold its current position. As discussed previously in Figure 5, the portfolio value will likely change in all three cases. This action set was found to be simple but effective, and is kept uniform for all strategies developed in this project.

The strategy uses the forecasting signal and introduces a margin of opportunity. It executes a buy order when the forecast for the day is 1% higher than the previous day's closing price, and conversely, a sell order when the forecasted price for the day is 1% lower. Otherwise, it simply holds the current position. The 1% margin is meant to provide some robustness and only make a trade when there is significant opportunity for profit.

Figure 17 shows the strategy architecture. The strategy pulls in stock price data and forecasted prices from an online ARIMA model to make buy, hold or sell decisions.

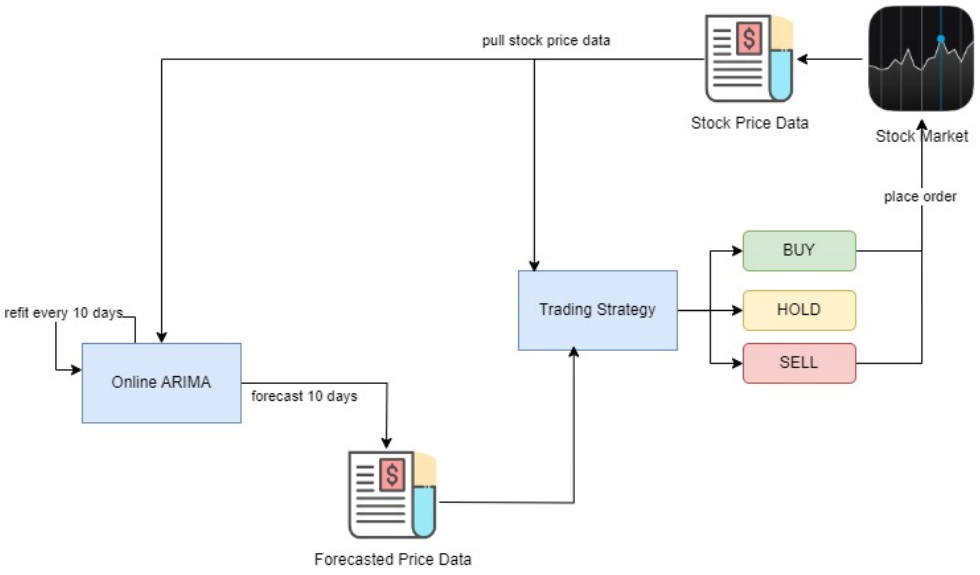

Figure 17: Forecasting based trading strategy architecture

3.3.3 Testing and Evaluation

Backtesting is the process of simulating a trading strategy on historical data in order to assess its viability and effectiveness. Though backtesting cannot predict how well a strategy might perform in the future, it can give insights into the viability, strengths and weaknesses of a strategy by simulating real-world conditions [51]. The open-source library backtesting.py[7] is used to backtest the strategy on the test dataset (See Figure 18). Starting with an initial capital of $100,000, the trading strategy looks at the previous days' close price and the forecasted price of the day and outputs a buy, hold or sell order, which is then simulated using the current day's open price (assuming that the order is placed at the beginning of the day). A 0.3% loss was applied on every trade order, to accommodate for trading fees and possible slippage. The strategy was backtested on all three stocks. The evaluation and results are discussed in Section 4.

[7] https://kernc.github.io/backtesting.py/

```
class ArimaStrat(Strategy):
    margin = 0.01 # % margin

    def init(self):
        self.forecast = self.I(lambda: data['Forecasts'].values, name='forecast')

    def next(self):
        # If forecasted price is higher than current price, buy
        if not np.isnan(self.forecast[-1]):
            if self.forecast[-1] > ((1 + self.margin) * self.data.Close[-1]):
                self.buy()
            # If forecasted price is lower than current price by margin, sell
            elif self.forecast[-1] < ((1 - self.margin) * self.data.Close[-1]):
                self.sell()

bt = Backtest(data, ArimaStrat, cash=CASH, commission=COMMISSION,exclusive_orders=True)
```

Figure 18: ARIMA trading strategy and backtesting snippet

3.4 Reinforcement Learning Environment Design

RL algorithms consist of an environment that facilitate interaction and learning for the agent. The environment maintains an internal state, defines the bounds of the observations and the set of actions that is available to the agent, the state transition logic at each time step, as well as the reward structure given to the agent. The Gymnasium framework is an extensible framework that specifies a standardized interface for defining reinforcement learning environments [52]. The advantage of defining the environment using this framework is that it provides an abstraction such that standard implementations of various deep RL algorithms can work interchangeably. In fact, the RL agent does not need to know anything about the environment, its state or mechanisms, except the observations it makes and the rewards it gets on performing some action at some timestep. Most of the RL algorithm implementation libraries are compatible with the Gymnasium framework.

An appropriate environment that simulates the stock market is required in order to apply RL algorithms to stock trading. Tensortrade[8] is an open-source library that was used to develop and configure an OpenAI Gymnasium compatible trading environment. A trading environment was set up with a simulated stock exchange running on the historical stock market data collected. Similar to the forecasting strategy backtesting conditions, a trading fee of 0.3% was levied all trades and an initial capital of $100,000 assigned to the agent. The environment contains all the logic to simulate trade order execution and track the portfolio value over time. Different environments for training and testing were set up using training and testing datasets. In the training environment, a maximum loss of 10% is configured. If during training the agent loses more than 10% of the portfolio value, the environment terminates that session to avoid wasting time on underperforming agent. The testing environment though, lets the agent play out the strategy until either the 3-year trading period is over or until the entire capital is lost.

3.4.1 Action Space

Like the forecasting strategy, an action set consisting of buy, hold and sell actions was configured. The agent can only trade on a single asset, and it can either go all in on the asset (buy action), or cash out entirely (sell action), or hold the current position. To simplify outputs from the action network in the RL agent, a binary action space is used in combination with an internal agent state maintaining the

[8] https://www.tensortrade.org/en/latest/#

current position the agent is in. The binary actions are therefore positions, with 1 for holding the entire net worth in the asset and 0 for exiting the market (See Figure 19).

State	Action	Meaning
0	0	Keep net worth in `cash` wallet (HOLD)
0	1	Transition net worth from `cash` wallet to `asset` wallet (BUY)
1	0	Transition net worth from `asset` wallet to `cash` wallet. (SELL)
1	1	Keep net worth in `asset` wallet (HOLD)

Figure 19: The binary action space and the internal agent state transition matrix[9]

3.4.2 Observation Space

The observation space consists of the pre-processed stock price and volume data along with the 20 technical indicators described in Section 3.2.4. Few indicators output more than 1 feature, and so the total number of features is 35. It looks back at the last 120 days of data at each timestep, making the observation space a (120×35) matrix. Using a larger observation window seemed to improve performance during experimentation, but computationally expensive, and so 120 was chosen as a good trade-off size.

3.4.3 The Reward Function

This is the feedback signal given to the agent for performing an action. A naïve approach is to use the difference in portfolio value before and after an action, but this is problematic as price fluctuations arising from noise can provide unintended positive or negative feedback to the agent. The result of a good trading decision can show up days after the trade, and is often not immediately obvious.

Additionally, the reward needs to be scaled to avoid imbalance. A trade with a slight profit margin made with larger capital might return a larger profit value than a more worthwhile trade with higher profit margin made with lesser capital. The total capital available to invest will change with time, depending on how successful previous trades were. Therefore, a reward function must be selected that scales well and is resistant to noise. A few different reward functions were experimented with –

Log Return: Logarithm of the change in portfolio value from the previous timestep.

Return percentage: The cumulative percentage change in the portfolio net worth over the last 30 days. Fewer than 20 days would be too tiny a window to capture the result of a good trading decision, as sometimes good bets may not be immediately profitable due to market fluctuations/noise. More days might weaken the signal. Long term profits can be expected to be captured better by the cumulative reward.

Risk Adjusted Return: Similar to return percentage, the last 30 days of portfolio values are analysed and the Sharpe ratio calculated. The Sharpe ratio is a common evaluation metric to assess the risk adjusted performance of a trading strategy [53]. It is the ratio obtained by dividing the excess returns over the risk-free rate of return by a measure of its volatility. A ratio above 1 is generally considered to be good, and a higher ratio is better when comparing strategies.

[9] https://www.tensortrade.org/en/latest/components/action_scheme.html#

$$Sharpe\ Ratio = \frac{R_s - R_f}{\sigma_s}$$

(3.2)

where,

R_s is the return of trading strategy,

R_f is the risk-free rate,

σ_s is the standard deviation of the return.

To test the environment design, a simple predictable sine curve was generated. This was used as the time series data to stimulate a stock price movement. The various reward functions were plugged in and a simple DQN RL algorithm was trained on the environment. The log return function described was too vulnerable to noise. The risk adjusted return with Sharpe ratio was found to be a bit too risk averse, and the agent would not make any trades and simply hold the position if the general trend of the stock is going up. In RL terms, the agent is said to be purely in exploitation state, where it does not explore new actions. The return percentage reward function was chosen for its robust performance following experimentation and testing.

The trading environment is packaged into a factory function to instantiate the environments on-demand, supplying them to the deep RL algorithms as needed, and to keep them isolated from each other to avoid unintentional interference (See Figure 20).

```
def create_env(ticker, config, dataset):
    # set up Exchange
    close_price = Stream.source(list(dataset["Close"]), dtype="float").rename("USD" + "-" + ticker)
    exchange_options = ExchangeOptions(commission=COMMISSION)
    exchange = Exchange("Exchange", service=execute_order, options=exchange_options)(close_price)
    # set up Instruments, Wallets and Portfolio
    USD = Instrument("USD", 4, "US Dollar")
    ASSET = Instrument(ticker, 4, ticker)
    cash = Wallet(exchange, CASH * USD)  # This is the starting cash we are going to use
    asset = Wallet(exchange, 0 * ASSET)
    portfolio = Portfolio(USD, [cash, asset])
    # create Renderer feed
    renderer_feed = DataFeed([
        Stream.source(pd.to_datetime(dataset.index)).rename("date"),
        Stream.source(list(dataset["Open"]), dtype="float").rename("open"),
        Stream.source(list(dataset["High"]), dtype="float").rename("high"),
        Stream.source(list(dataset["Low"]), dtype="float").rename("low"),
        Stream.source(list(dataset["Close"]), dtype="float").rename("close"),
        Stream.source(list(dataset["Volume"]), dtype="float").rename("volume")
    ])
    # create environment feed
    features = [Stream.source(list(dataset[c]), dtype="float").rename(dataset[c].name) for c in dataset.columns if c not in ['Open', 'High', 'Low']]
    feed = DataFeed(features)
    feed.compile()
    # set action scheme
    action_scheme = BSH(cash=cash, asset=asset)
    # create environment
    env = ttenv.create(
            feed=feed,
            portfolio=portfolio,
            action_scheme=action_scheme,
            reward_scheme=REWARD_SCHEME,
            renderer_feed=renderer_feed,
            renderer=PlotlyTradingChart(),
            window_size=config["window_size"],
            max_allowed_loss=config["max_allowed_loss"]
        )
    return env
```

Figure 20: Environment creation function snippet

3.5 Deep Reinforcement Learning Trading Strategies

The deep RL based trading strategies were developed using the Stable-Baselines3 library. Stable-Baselines3 is a popular open-source library that provides reliable standard implementations of deep reinforcement learning algorithms [54]. It uses the popular PyTorch[10] library to build the neural networks approximating policy and value functions of the RL algorithms and leverages quality implementations of learning optimisers such as the Adam Optimiser [55] to train the networks. Stable-baselines3 implementations of RL algorithms are compatible with the OpenAI Gymnasium framework and so work on the trading environment set up in Section 3.4.

This project uses Stable-Baselines3 implementations of DQN, PPO and A2C algorithms (discussed in Section 2.5) to develop single-agent RL strategies as baselines. The GARL based trading strategy was developed by extending the existing single-agent RL algorithm implementations.

Figure 21 shows the strategy architecture for RL algorithms. The RL agent interacts with the trading environment discussed in the previous section in an interaction loop. The specific RL algorithm can be swapped out with any standard RL algorithm or GARL algorithm. The model architecture of the RL algorithm, including the internal state, and one or more neural networks differs according to the algorithm.

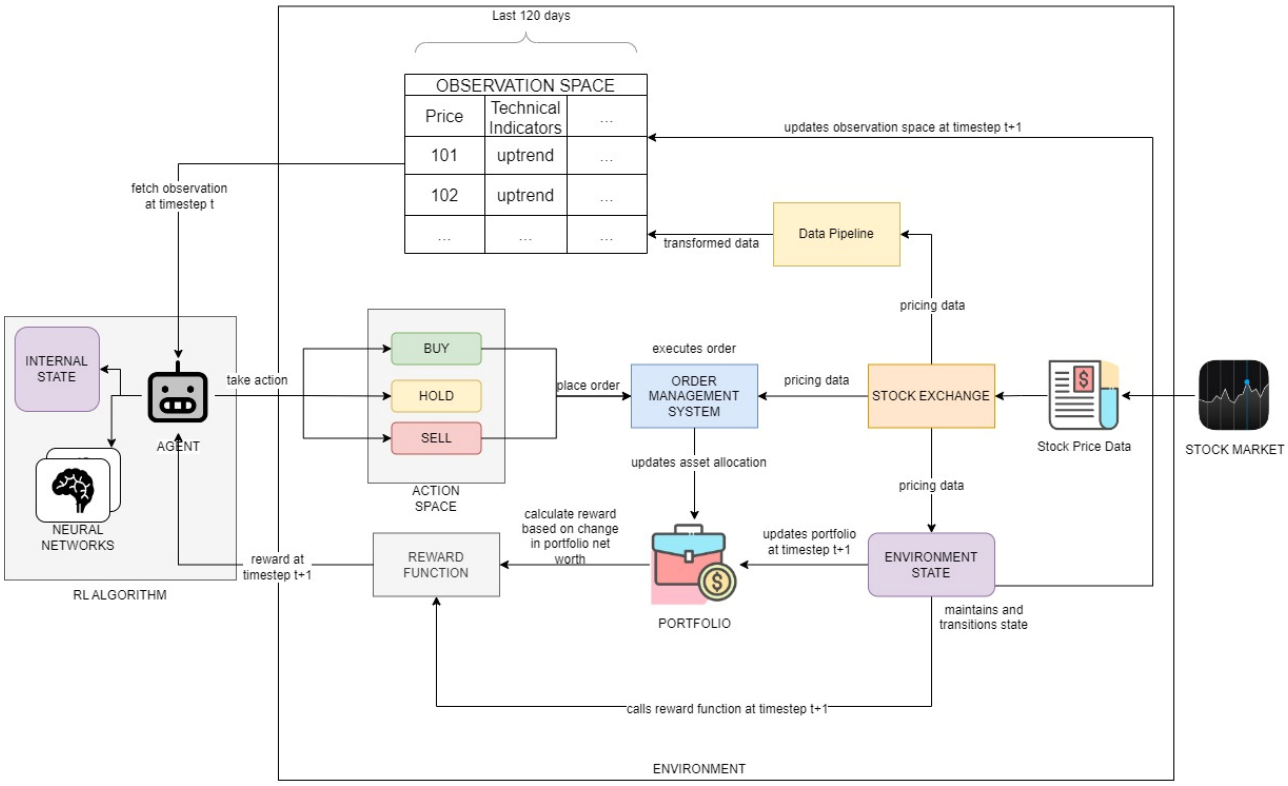

Figure 21: RL Trading Strategy architecture

3.5.1 Single-Agent Reinforcement Learning Model Architecture

The single-agent RL model architectures depends on the specific RL algorithm, and are consistent with the standard architectures described in Section 2.5. Additionally, Stable-baselines3 adds a feature extractor neural network that processes raw observations and transforms them into a feature vector that can be fed into the policy/value networks (See Figure 22). Since the observations are discrete 2D arrays, they are flattened into 4200 input parameters (120 days * 35 features = 4200) and fed into a feature

[10] https://pytorch.org/

extractor consisting of a multilayer perceptron network with 2 fully connected layers. The extracted features are then fed into the algorithm specific fully connected output networks, like the QNetwork in DQN and action/value networks in A2C/PPO.

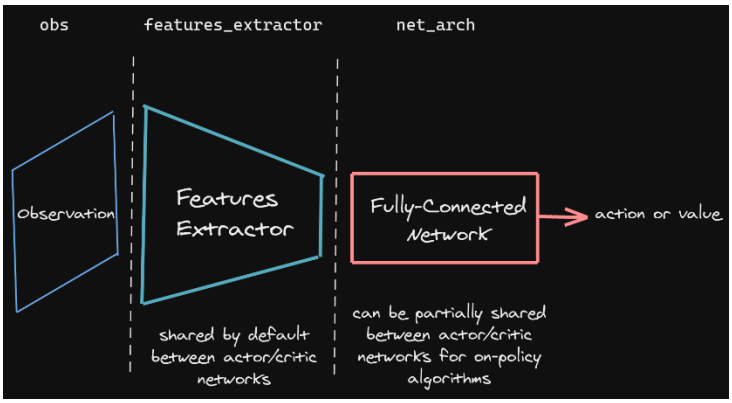

Figure 22: Feature extractor in stable-baselines3[11]

```
ActorCriticPolicy(
  (features_extractor): FlattenExtractor(
    (flatten): Flatten(start_dim=1, end_dim=-1)
  )
  (pi_features_extractor): FlattenExtractor(
    (flatten): Flatten(start_dim=1, end_dim=-1)
  )
  (vf_features_extractor): FlattenExtractor(
    (flatten): Flatten(start_dim=1, end_dim=-1)
  )
  (mlp_extractor): MlpExtractor(
    (policy_net): Sequential(
      (0): Linear(in_features=4200, out_features=64, bias=True)
      (1): Tanh()
      (2): Linear(in_features=64, out_features=64, bias=True)
      (3): Tanh()
    )
    (value_net): Sequential(
      (0): Linear(in_features=4200, out_features=64, bias=True)
      (1): Tanh()
      (2): Linear(in_features=64, out_features=64, bias=True)
      (3): Tanh()
    )
  )
  (action_net): Linear(in_features=64, out_features=2, bias=True)
  (value_net): Linear(in_features=64, out_features=1, bias=True)
)
```

Figure 23: Model structure output for the A2C algorithm.

Figure 23 shows the model architecture output returned by stable-baselines3 for the A2C algorithm. A2C consists of 2 neural networks – the policy/action network and the value network. The feature extractor (pi_features_extractor) flattens the input 2D matrix observations into a 4200 size 1-D vector, and then the MLP extractor network (policy_net) extracts features using an MLP network with 2 fully connected layers with tanh activation, and finally the extracted 64 features are fed into the action network (action_net). The action network outputs the binary action space probabilities. Similar architecture is produced for the value network (vf_features_extractor and value_net), except that the final output of the value_net is a single feature containing the value estimate of the current state. PPO

[11] https://stable-baselines3.readthedocs.io/en/master/guide/custom_policy.html

also consists of action and value networks, and has a similar model architecture. DQN has a slightly different architecture, consisting of a QNetwork and Target network instead.

3.5.2 Group-Agent Reinforcement Learning Model Architecture

The decentralized GARL framework Decentralized Distributed Asynchronous Learning (DDAL) is acknowledged for its numerous benefits such as scaling and learning stability, as detailed in Section 2.7. However, the asynchronous knowledge sharing is deemed too complex. A simpler synchronous version of the algorithm using homogenous DQN agents, dubbed Group Deep-Q Network (GDQN) is proposed for this project. A pair of DQN algorithms with identical neural network architecture for the QNetwork and Target network is chosen to enable gradient sharing as the type of knowledge shared. The selection of DQN is justified by its simple architecture and effectiveness seen in the single-agent RL setup. Both agents work on the same stock, though in different trading environments. After an initial period of learning without knowledge sharing, the gradients are shared as knowledge at regular intervals. As the pair of DQN agents work on the same stock, they are both equally relevant to each other and so the relevance term in the weighted average is omitted. Additionally, the training experience term is also omitted as gradients are synchronously shared at the same level of experience. A simple averaging of the accumulated gradients is performed and subsequently applied to the models.

An extension of the stable-baselines3 library is implemented to facilitate the GARL approach described. This is done through the use of stable-baselines3 callbacks functions, which are leveraged to execute custom code at different stages of the learning process. While several built-in callback functions exist, a custom callback function is developed to meet the specific requirements of the project.

The ‘on_training_start’ callback is used to register a gradient hook function to each of the parameters of the neural network and initialize a shared list data structure to hold the gradients in memory. The gradient hook function gets called whenever gradients are computed during backpropagation, and the gradients generated are accumulated in the shared memory. The learning process is structured in two phases: the initial phase, lasting about a third of the total training period, where the agents are allowed to learn independently without knowledge sharing. This phase is followed by a collaborative learning period marked by synchronous knowledge-sharing along with independent learning. The accumulated gradients are pulled from the shared memory, averaged and applied to every agent every 5000 timesteps. The described model architecture and a code snippet is given in Figure 24 and Figure 25.

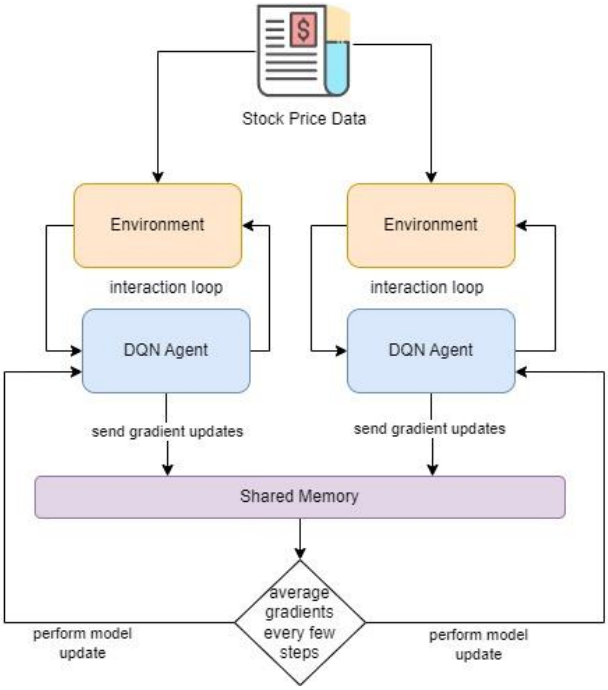

Figure 24: Synchronous GARL setup with twin DQN agents (GDQN) architecture

```
class GARLDQN:
    def __init__(self, envs, n_agents = 2):
        self.envs = envs
        self.n_agents = n_agents
        self.agents = []
        self.callbacks = []

        for i in range(n_agents):
            train_env = envs[i]
            agent = DQN("MlpPolicy", train_env, device='cuda', verbose=0, buffer_size=int(1e4))
            callback = GARLCallback()
            self.agents.append(agent)
            self.callbacks.append(callback)

    def train(self, total_timesteps, sharing_frequency, start_sharing_after):
        steps_per_iteration = sharing_frequency // self.n_agents
        for i in range(int(total_timesteps // sharing_frequency)):
            print(f"Iteration {i+1}")

            # Train agents
            for j, (agent, callback) in enumerate(zip(self.agents, self.callbacks)):
                agent.learn(total_timesteps=steps_per_iteration, callback=callback)
                print(f"Agent {j+1} trained for {steps_per_iteration} steps")

            if i >= start_sharing_after:
                # Share and update gradients
                print("Sharing knowledge...")
                self.share_knowledge()
```

Figure 25: GARL code snippet

3.5.3 Hyperparameter Exploration

Stable-Baselines3 provides a layer of abstraction on the underlying implementation of the RL algorithms and architecture, while still exposing the hyperparameters such as exploration/exploitation trade-off fraction, discount factor, network update frequency etc. Additionally, it also exposes

hyperparameters of the underlying neural network implementation, such as the network architecture (number of layers, activation functions, etc), learning parameters such as learning rate, batch size and epoch, and the learning optimiser and regularisation parameters. These hyperparameters were tuned in the hyperparameter exploration stage.

An evaluation environment is required to validate the effectiveness of the algorithm at this stage. The training data is further split into training and evaluation data in an 80:20 split, these are then used to create separate training and evaluation Gymnasium environments.

The models are tuned using Optuna. Optuna is an open-source hyperparameter optimization library that uses a combination of various advanced search algorithms, automatic pruning and parallelization techniques to automate hyperparameter exploration and model optimization [56]. Along with the set of parameter spaces, an objective function needs to be defined that will be then maximized with optimal parameters. Stable-baselines3 provides evaluation helpers that help validate the agent against the evaluation environment, returning the cumulative reward acquired by the agent. This was supplied as the objective function to Optuna. Figure 26 shows the optimization history plot, with the various objective values plotted against the number of trials. The red line tracks the best objective value.

The three algorithms are trained for 10,000 timesteps in each exploration run (to reduce computation load), then evaluated using the objective function and finally the best parameters of each algorithm were found.

Once the best parameters were found, a new model was trained using the entire training dataset for 1 million timesteps to yield the fine-tuned model (See Figure 27). The process was repeated for all the algorithms.

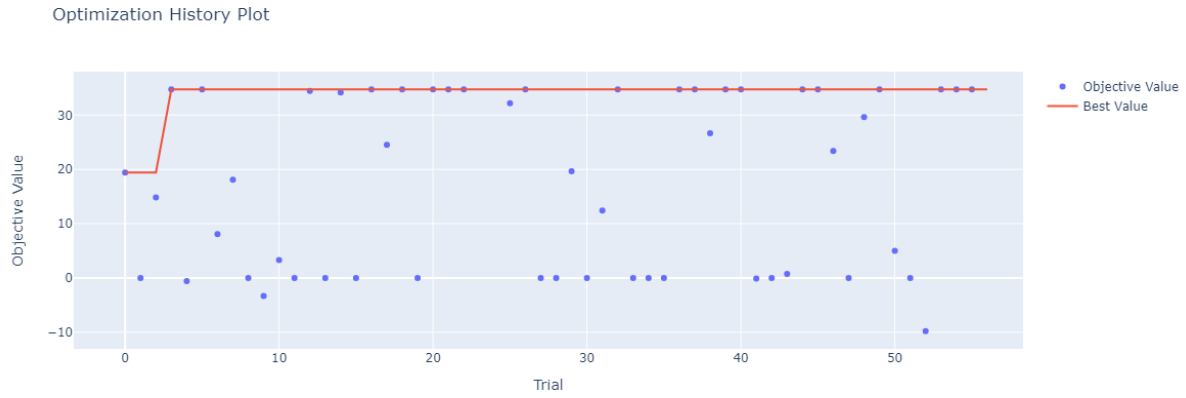

Figure 26: Hyperparameter optimization history

```python
def train_best_model(best_params, df_train_eval):
    config = {
        "window_size": WINDOW_SIZE,
        "max_allowed_loss": MAX_ALLOWED_LOSS
    }

    gradient_steps = max(best_params['train_freq'] //best_params['subsample_steps'], 1)
    train_env = create_env(config, reward_scheme, df_train_eval)
    train_env = DummyVecEnv([lambda: train_env])

    model = DQN(policy="MlpPolicy",
            env=train_env,
            learning_rate=best_params['learning_rate'],
            buffer_size=best_params['buffer_size'],
            learning_starts=best_params['learning_starts'],
            batch_size=best_params['batch_size'],
            gamma=best_params['gamma'],
            train_freq=best_params['train_freq'],
            gradient_steps=gradient_steps,
            target_update_interval=best_params['target_update_interval'],
            exploration_fraction=best_params['exploration_fraction'],
            exploration_final_eps=best_params['exploration_final_eps'],
            policy_kwargs=dict(net_arch=best_params['net_arch']),
            tensorboard_log="./logs/",
            verbose=1,
            device='cuda')

    model.learn(total_timesteps=1e6)
    return model, train_env
```

Figure 27: Training RL model on best parameters

3.5.4 Testing and Evaluation

The testing environment containing data from 2021-2023 is used to backtest the strategy. Figure 28 shows the backtesting snippet and Figure 29 shows the backtesting simulation results. The red and green arrowheads plotted show the entry and exit points of the strategy.

In the offline mode, the trained agent is set up in an interaction loop, predicting the next best possible action based on the current observation. Then the environment steps into the next state, returning the reward for the action taken and the next observation. The predicted actions are recorded, along with the change in portfolio value. The loop runs until the end of testing data or until the session is truncated due to insufficient funds.

In the online mode, the trained agent is set up in a similar interaction loop, except that the reward given to the agent is used to iteratively train the agent as well at each step.

The evaluation and results are discussed in Section 4.

```
test_env = create_env(config,reward_scheme, df_test)
obs, info = test_env.reset()

terminated, truncated = False, False
actions, rewards, portfolio_values = [], [], []

while not terminated and not truncated:
    action, _states = model.predict(obs)
    next_obs, reward, terminated, truncated, info = test_env.step(action)
    actions.append(action)
    rewards.append(reward)
    portfolio_values.append(info['net_worth'])
```

Figure 28: Backtesting RL in test environment

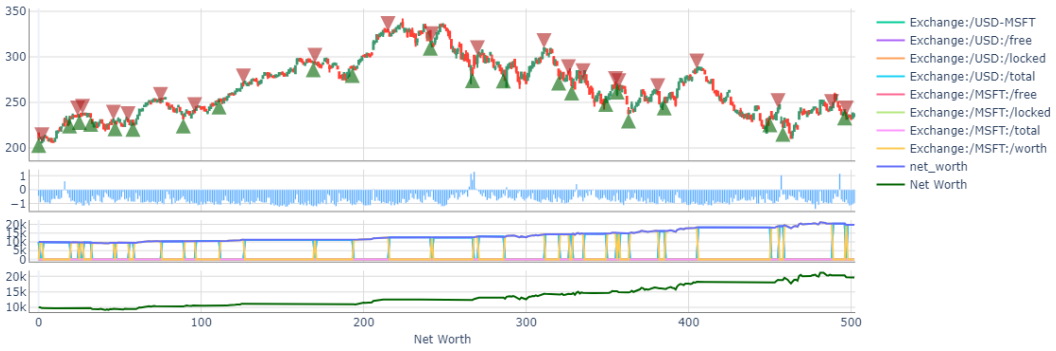

Figure 29: Backtesting DQN on MSFT

3.6 Ethical and Professional Considerations

Data Access & Source: This project will use only publicly available, legally sourced US stock market data. This includes the open, high, low, close and volume of the stocks.

Privacy: It does not use any personally identifiable or otherwise sensitive information. No private company information will be used to avoid insider trading. Even so, the data itself is not shown in the project report, but aggregates of profits and other metrics is shown as graphs.

Intellectual Property: All data, algorithms, libraries, frameworks and software used in the project are open sourced under MIT license and not proprietary.

Market manipulation: Though the strategies developed are not deployed in the real market, care has been given to avoid potential misuse of the strategies to that end, in line with ACM guidelines. The RL algorithms trade orders are monitored under testing for any such signs.

FCA regulations: Financial Conduct Authority (FCA) has released guidelines on algorithmic trading requirements – MAR 7A.3 [57]. These include development of robust systems and controls, record keeping, dealing with system failures, market making and market abuse prevention. These do not apply to this project as it is not deployed in the real market. Regardless, some measures are taken as best practices such development of control systems and risk management, and rigorous testing.

Other professional considerations: All experiments and results are in their original form and are not misrepresented. All ideas from other research work are referenced, and existing libraries and tools will be credited.

3.7 Risk Considerations

There are several risks associated with the successful completion of the project. The likelihood, impact, and mitigation strategies of the risks have been detailed in the Table 1.

Risk	Likelihood	Impact	Mitigation
Low quality of historical market data	Medium	Medium	Appropriate data source selection is done with a focus on quality. Quality analysis of the data is done. Data cleaning methods are employed to deal with small quality issues.
Loss of Data, like code and report	Low	High	Use of regular cloud backups
Accidental code overwriting	Medium	Medium	Use of version control systems (Git)
Device Failure	Low	High	Use of backup devices like the university lab computers
Time management issues	High	High	Tasks are broken down and weekly sprints are created with well-defined goals and outcomes. A Gantt chart of the timeline is created and followed closely.
Unexpected personal commitments or accidents	Medium	High	Sufficient time is kept as buffer between tasks
Development and operational costs, financial risks	Low	Low	The models are trained on personal systems. The strategies are not live, so no transactional costs or other fees are incurred.
Market complexity like high volatility and noise due to a variety of factors	High	Medium	The market is simulated using real historical data to mimic market complexity and the strategies are tested on them. Care has been taken to mimic market inefficiencies such as slippage and brokerage fees.
Reward function design can introduce bias in terms of risk appetite. Effects of decisions might not be apparent, leading to delayed rewards	High	Medium	Sufficient amount of investigation and experimentation is done to design an appropriate reward function.
Data leakage and selection bias	Medium	Medium	Data is split into training and test, and the test data is never used in the training process.

Table 1: Risk Assessment

3.8 Summary

The research methodology, along with ethical, professional and risk considerations were discussed. The methodology detailed the collection of stock market data and creation of a data pipeline to perform various data preprocessing on the data. The processed data was used to create baseline strategies. First, ARIMA is used on the data to generate forecasts on future data, and a trading algorithm uses the forecasts to make trading decisions, forming the first baseline. A custom trading environment is designed for RL algorithms. Then, three single-agent RLs (DQN, PPO and A2C) are trained and fine-tuned on the trading environment created. A GARL strategy is devised, implemented, trained and fine-tuned on the environment. A separate testing environment is created and all three single-agent RLs and GARL strategies are backtested. In the following chapter, the results of the backtesting are discussed and evaluated.

Chapter 4

4 RESULTS AND DISCUSSION

4.1 Chapter Overview

This chapter discusses the evaluation criteria and metrics selected to evaluate the trading strategies developed. It presents the results of backtesting the models and compares the performance of the models with respect to the evaluation criteria selected. Finally, the results are analysed and explained, along with a discussion on the limitations.

4.2 Evaluation Metrics

While evaluating a trading strategy, it is important to consider the strategy's profitability as well as the risk associated. Some of the widely used ones are profit factor, win ratio, maximum drawdown and Sharpe ratio.

1. **Cumulative Return**: Cumulative return is the total returns that the trading strategy has generated over the evaluation period, expressed as a percentage. This is a standard measure of profitability.

$$Cumulative\ Return = \frac{Current\ Price - Original\ Price}{Original\ Price} \quad \textbf{(4.1)}$$

2. **Compound Annual Growth Rate (CAGR):** CAGR is a smoothed-out measurement of the rate of return and provides an idea of how well a strategy does over time. It is the return that an investment needs to have per year to reach its end net value at the end of the duration, assuming any profits were reinvested at the end of each year.

$$CAGR = \left(\frac{Value_{end}}{Value_{begin}}\right)^{\frac{1}{t}} - 1 \quad \textbf{(4.2)}$$

3. **Maximum Drawdown:** A common risk management measure, maximum drawdown of a strategy is the maximum loss from its peak to its trough over the evaluation period. This indicates the strategy's worst-case risk exposure.

$$Maximum\ Drawdown = \frac{Trough\ Value - Peak\ Value}{Peak\ Value} \quad \textbf{(4.3)}$$

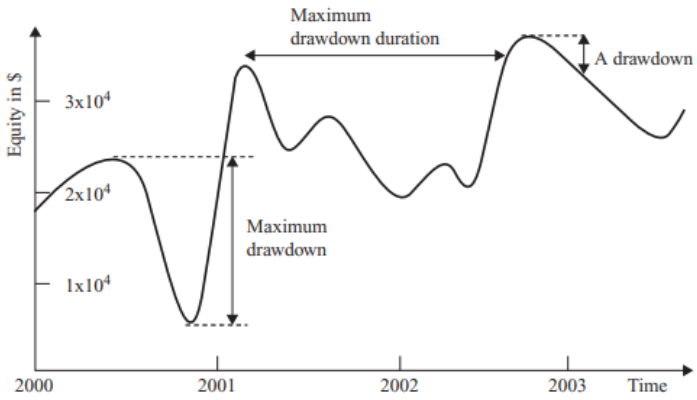

Figure 30: Maximum drawdown for a sample strategy [58]

4. **Sharpe Ratio**: The Sharpe ratio introduced in Section 3.4 is also commonly used to compare reinforcement learning based trading strategies [10], [48]. Equation (3.2) shows the formula for the ratio calculation. Sharpe ratio is an essential metric of risk adjusted returns of a strategy, and a higher ratio is better.

The above four evaluation metrics were selected as they are considered to be key indicators of profitability and risk management.

The evaluation metrics were calculated using Quantstats[12], an open-source analytics library that provides APIs for common portfolio performance and risk metrics. The strategy performance was benchmarked, suitable graphs were plotted and a strategy report tear sheet was generated.

4.3 Results

All the strategies are backtested on Apple, Microsoft and Google stocks between 2021-2023. Most of the strategies developed remained profitable in the testing period.

Time Series forecasting (ARIMA) Trading Strategy

The offline ARIMA model proved ineffective for long term prediction, deviating significantly from the actual prices after the 10-day mark. The online ARIMA model, on the other hand, proved extremely effective and robust (See Figure 31). The blue lines show the portfolio returns and the yellow lines show the closing price graph. The closing price return is equivalent to the strategy of buying and holding the asset during the entire duration. Online ARIMA consistently outperformed the buy and hold strategy on Apple stocks. On Microsoft, it slightly underperforms for a brief duration, but bounces back. The performance metrics of online ARIMA are shown in Table 3.

[12] https://github.com/ranaroussi/quantstats

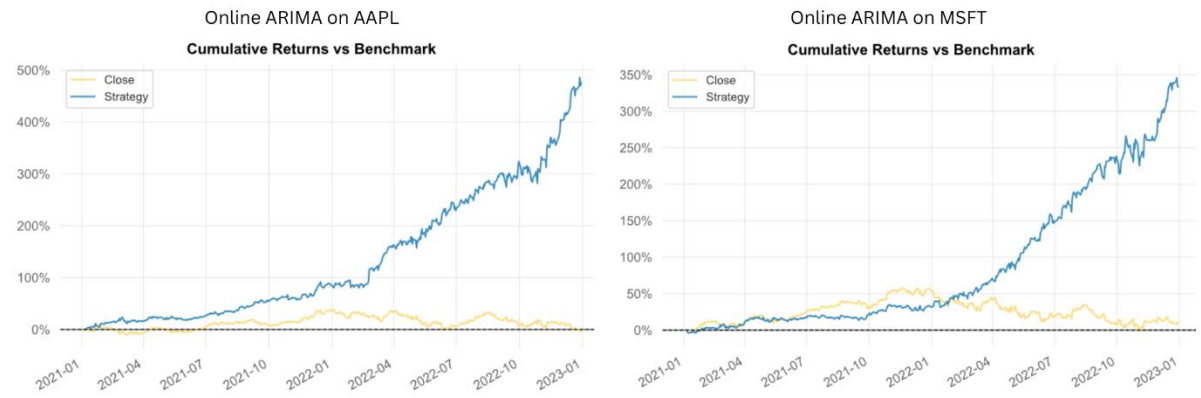

Figure 31: Online ARIMA model returns on Apple and Microsoft stocks

Single-Agent RL Trading Strategy

In the single-agent reinforcement learning case, DQN proves to be most effective one, performing well for all the stocks (See Figure 32). A2C and PPO algorithms on the other hand, struggle to keep up with even the buy and hold strategies. The online single agent algorithms does not show any performance improvement. A2C does not seem to work any better than taking random actions. PPO remains profitable, but still does not beat the buy and hold strategy. DQN initially underperforms the buy and hold strategy (while still remaining profitable), but after a few bad trades, bounces back and beats the buy and hold strategy, consistently outperforming it for the remaining period.

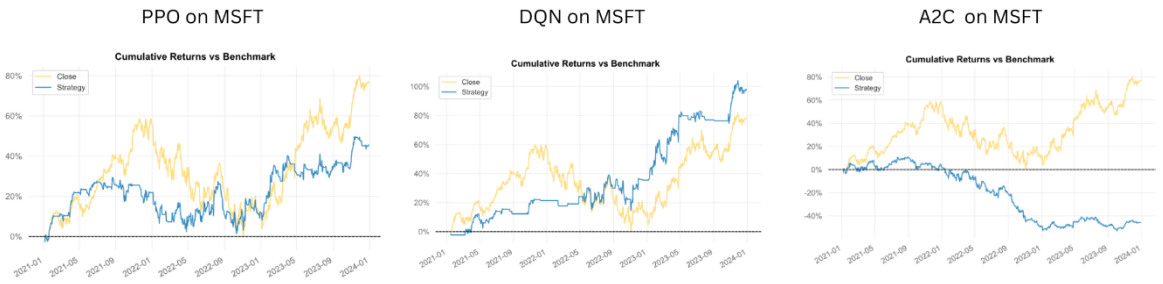

Table 2 compares the evaluation metrics for all three single-agent RL algorithms for Microsoft stocks. DQN shows much better performance across all key metrics.

Figure 32: Single-agent (PPO, DQN and A2C) returns on Microsoft stocks

Metric	PPO	DQN	A2C	Buy and Hold
Cumulative Return	45.66%	98.27%	-45.82%	77.21%
CAGR%	9.11%	17.33%	-13.24%	14.18%
Sharpe Ratio	0.68	1.27	-0.75	0.83
Max Drawdown	-21.59%	-17.7%	-57.4%	-37.15%

Table 2: Single-agent performance metrics- PPO vs DQN vs A2C vs Buy & Hold

Figure 33 takes a closer look at the end of year (EOY) returns for PPO and DQN. The close column shows the buy and hold strategy returns, whereas the strategy column shows PPO and DQN. It is

observed that both strategies performed well during market downturns in 2022, with PPO limiting the losses and DQN even turning a profit.

PPO

EOY Returns vs Benchmark

Year	Close	Strategy	Multiplier	Won
2021	55.64%	21.88%	0.39	-
2022	-28.02%	-8.28%	0.30	+
2023	58.19%	30.30%	0.52	-

DQN

EOY Returns vs Benchmark

Year	Close	Strategy	Multiplier	Won
2021	56.76%	21.41%	0.38	-
2022	-28.02%	11.59%	-0.41	+
2023	58.19%	46.35%	0.80	-

Figure 33: EOY Returns of PPO and DQN vs Buy and hold strategy

GARL Trading Strategy

In the GARL setup, the two individual DQN agents learn differently. While one performs worse off than the single-agent DQN, the other one performs much better, consistently performing better than the buy and hold strategy. Figure 34 shows the returns of the GARL agent against the buy and hold strategy. Table 3 compares the four evaluation metrics of GARL strategy, single-agent DQN strategy, online ARIMA and the buy and hold strategy on Microsoft stocks in the testing period. The GARL strategy shows better returns and CAGR percentage than single-agent DQN, but the Sharpe ratio and max drawdown metrics are lower, indicating that the single agent DQN manages risk better. The online ARIMA proved to be the best performing model across all metrics.

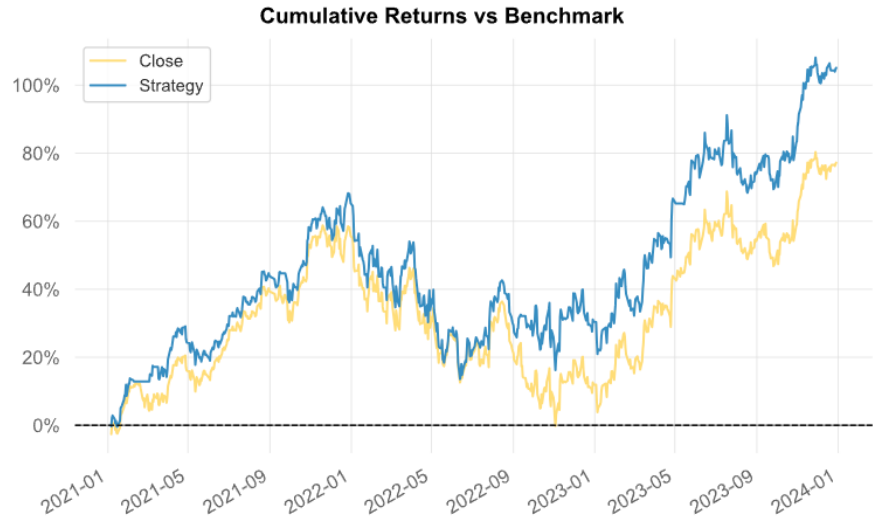

Figure 34: GARL agent returns on Microsoft stocks

Metric	GARL	Single DQN	ARIMA	Buy and Hold
Cumulative Return	105.09%	98.27%	333.35%	77.21%
CAGR%	18.12%	17.33%	66.71%	14.18%
Sharpe Ratio	1.03	1.27	2.89	0.83
Max Drawdown	-32.49%	-17.7%	-11.05%	-37.15%

Table 3: GARL vs DQN vs ARIMA vs Buy and Hold strategy performance metrics

4.4 Discussion

The backtesting process provides valuable insights into the strengths and weaknesses of the various trading strategies developed.

Time Series Forecasting (ARIMA) Trading Strategy

The offline ARIMA model managed to capture the general trend and predict prices for a very short term (~10 days), but performed poorly for long term predictions. It is not considered viable for use in a trading strategy. This is as expected from a simple linear univariate model in financial markets where prices are highly stochastic and volatile, and often exhibit non-linear patterns.

The online ARIMA model, however, overcomes these limitations by repeatedly retraining every 10 days. It exhibited robust performance across all stocks, even during market downturns. The slight underperformance for Microsoft for an interval followed by a rebound indicates that there might still be periods where markets are difficult to predict. The online model uses ARIMA's effectiveness in capturing recent market patterns and capitalizing on short term movements, while also updating it regularly with recent data. The trading strategy designed capitalizes on a 1% margin of difference between the forecasts and recent price, which allows the strategy to pick up small movement opportunities while accounting for noise in prediction, market inefficiencies and trading fees. The success of the model highlights the effectiveness of simple linear models when combined with intelligently engineered strategies, providing a high degree of control. ARIMA manages to outperform much more complex deep RL models, while being much easier to implement and faster to train. This is inline with a few studies, showing its superior performance to complex models in cryptocurrency markets [36]. Another study on deep RL for trading by Pricope, 2021 [59] showed that while deep RL for trading has showed potential in recent studies, there is a lack of realistic backtesting and meaningful comparisons of deep learning strategies with traditional machine learning models.

Single-Agent RL Trading Strategy

The online DQN algorithm outperforms other RL algorithms (PPO and A2C) as well as the buy and hold strategy. Although it starts out underperforming, it appears the model quickly learns from the negative rewards it gets, and adapts to the market soon enough. This shows its ability to learn from the feedback it receives, correct course and make decisions profitable in the long term.

PPO, while remaining profitable, seems to make very conservative bets, underperforming the buy and hold strategy. However, its ability to limit losses in 2022 during market downturn, suggests it might be better suited for strategies focused on capital preservation during bear markets. While PPO seems to be risk averse, DQN strikes a good balance, managing profitability while limiting risk well. In contrast, A2C performs poorly, indicating that it may not be suitable for stock market prediction tasks.

While the online versions of the RL algorithms are presented in Section 4.3, constantly learning after each reward is presented, they do not seem to perform much better than their offline counterparts. Unlike ARIMA, a huge performance improvement is not seen. This might be due to the various gradient smoothing techniques typically used in neural networks, leading to a resistance to drastic policy changes, with just a few negative rewards. RL algorithms that work on policy networks, like A2C and PPO, are particularly affected by this. A possible workaround is to retrain the model for a large number of timesteps regularly on the latest data, identical to online ARIMA.

One major reason for the ineffectiveness of deep RL algorithms compared to traditional machine learning strategies could be the lack of rich, vast stock pricing data. The daily stock price data for any

stock, can be at most a few thousand datapoints over a few decades. This is very small compared to the millions of datapoints neural networks typically work with. Neural networks are characterized by a large number of model parameters, and the lack of data likely overfits the data. Though several additional features are engineered to supplement the data, like the 20 technical indicators, they seem to fall short.

DQN's experience replay mechanism might be the reason for its improved performance from other RL strategies, allowing for improved sample efficiency. It allows the agent to reuse experiences multiple times and improve the learning. This might be the reason DQN manages to cope well even with lesser data. PPO and A2C are less sample efficient compared to DQN, and generally need more interactions to learn. Though lacking experience replay, PPO uses multiple epochs of minibatch updates to cope with sample inefficiencies. A2C, in contrast, does not possess any such techniques.

PPO and A2C both use policy networks, and but PPO uses advanced techniques like gradient clipping avoid high variance in gradient estimates, encouraging conservative updates. This is lacking in A2C, which might explain the unstable learning seen. Another possible issue with A2C could be its myopic optimisation, focusing more on immediate awards and the inability to handle delayed rewards, which is not ideal in stock markets where consequences of actions are often delayed. A sample efficient version of A2C incorporated with experience replay exists, called Actor-Critic with Experience Replay (ACER) [60]. It might mitigate these issues, and can be taken as possible future research work.

GARL Trading Strategy

The individual DQN agents in GARL showed divergent performance, where one performed better and the other worse than the single-agent DQN variant. This suggests that GARL allows for distinct learning pathways for its constituent agents. For example, one might make some bad decisions, but share its learning from the exploration performed to the other agent, which can then focus on exploitation from the knowledge earned. A careful selection of a larger group with varying risk appetites (translating to exploration/exploitation ratio parameter in RL terms) can further test this theory.

However, the Sharpe ratio and the drawdown for GARL showed that it carried more risk than the DQN variant. This shows that the GARL shows better returns at the cost of added risk, which is in line with market hypotheses. The results suggest that while GARL has the potential to outperform existing RL methods in trading, it carries additional risk.

Chapter 5

5 SUMMARY AND CONCLUSIONS

The project addresses gaps in existing literature on the application of RL and GARL algorithms in stock trading. GARL algorithms is proposed to improve trading performance by leveraging the performance of the member RL algorithms in a way unexplored previously. It can provide a very flexible model to use multiple RL agents to possibly model various dependencies between different stocks and market phenomena.

The project successfully develops two baseline strategies, one using traditional time series forecasting technique and the other using standard single-agent RL algorithms. All strategies are tested in a realistic simulated environment using historical data, and most returned profitable results. The traditional ARIMA model proved exceptionally robust. Among RL algorithms, DQN showed the most adaptability and performance. The lack of data in the task of stock trading is identified to be a major barrier to RL algorithms to achieve comparable performance to traditional ML methods, when combined with clever strategy design.

While the existing standard RL algorithms might be exceptionally suited to the well-known RL tasks, they require significant fine-tuning and modifications to work well in real-world conditions. Specifically, it highlights the importance of designing a good RL environment before they can be applied to real-world environments. This proves doubly important in the realm of algorithmic stock trading, as no standard trading environments exist. Trading environments pose additional challenges of the stock market being highly volatile and noisy, varied in the type of asset, market and industry selected, and the nature of the results of trading decisions not being apparent. The designed environment must not only sufficiently simulate realistic conditions, it also requires clever design of the observation space, action sets and the rewards the agent receives for its actions. Such an environment is successfully designed and implemented for this project, and used to train and test the RL algorithms.

A GARL based architecture is proposed (GDQN) and was successfully implemented. It is shown to be able to leverage the cooperative group aspect to investigate various learning paths through its member agents. Further, the effectiveness of GARL strategy is investigated through backtesting. The results show that even though GARL strategy has the potential to outperform existing RL methods, it does that through carrying additional risk. While GARL can improve individual learning of the constituent group members, it is still limited by the effectiveness of its individuals for the task at hand.

Overall, the results confirm that while advanced deep learning models like GARL and RL have potential in automated trading, simpler well-tuned models like ARIMA can still outperform them in certain contexts. The objectives of the project were met, and a strong intuition for future research is developed.

5.1 Future Work

Online ARIMA models can be further improved by using exogenous variables (ARIMAX) models, designing a trading strategy that uses forecasts from multiple forecasting algorithms or technical indicators and devising an adaptive margin at which to make trading decisions.

The impressive performance of the online ARIMA model raises the question of the effectiveness of hybrid models that combine time series forecasting and RL. The forecasts can be used as signals to RL algorithm, which will benefit from the additional data. The ARIMA model too, will benefit from more fine-grained decisioning control from RL, instead of simple margin-based logic used in the project.

More research can be done on improvement of GARL strategies' risk management. Possible directions include configuring constituent agents with different exploration/exploitation ratios, allocating group resources in a way so that lesser capital is available to agents with larger risk appetite. Another possible risk management strategy is designing an adaptive capital allocation technique based on the recent performance of the member agent.

GARL trading strategy still leaves a lot of experimentation desired –

Stock Selection: Group agents can trade on different (corelated or uncorrelated) stocks or the same stock on different timescales.

Portfolio Variations: Different variations in portfolio, including machine learning models to intelligently select stocks and capital allocation.

Indicators: Use of additional indicators from other forecasting or sentiment analysis algorithms. Indicators from human traders can also be incorporated.

Heterogenous agents: While the original paper experiments with homogenous agents in a group [13], GARL is flexible enough to be used with heterogenous agents. So various RL algorithms can be used in a group instead of the same type of agent, or even the same algorithm with variations in neural network architecture can be applied in a group. Care must be taken to chose sample efficient RL algorithms like DQN and ACER.

Shared knowledge: Variations in the type of knowledge shared, like the accumulated reward so far, model parameters, state-action values, etc can be explored. Different weighting techniques to combine knowledge, like weighting on the Sharpe ratio achieved by the agent so far, signalling how well it has been doing, are exciting areas to explore.

6 REFERENCES

[1] T. Hendershott, C. M. Jones, and A. J. Menkveld, "Does Algorithmic Trading Improve Liquidity?".

[2] M. Intelligence, "Algorithmic Trading Market Share, Size & Trading Statistics." Accessed: Aug. 17, 2024. [Online]. Available: https://www.mordorintelligence.com/industry-reports/algorithmic-trading-market

[3] A. Nathan, "Top of mind," *Top of Mind, Goldman Sachs*, Jun. 2018, [Online]. Available: https://www.gsam.com/content/dam/gsam/pdfs/sg/en/commentary/GS_Top%20of%20Mind_June.pdf?sa=n&rd=n

[4] S. K. Sahu, A. Mokhade, and N. D. Bokde, "An Overview of Machine Learning, Deep Learning, and Reinforcement Learning-Based Techniques in Quantitative Finance: Recent Progress and Challenges," *Applied Sciences*, vol. 13, no. 3, p. 1956, Feb. 2023, doi: 10.3390/app13031956.

[5] V. Francois-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, "An Introduction to Deep Reinforcement Learning," *FNT in Machine Learning*, vol. 11, no. 3–4, pp. 219–354, 2018, doi: 10.1561/2200000071.

[6] Z. Zhang, S. Zohren, and S. Roberts, "Deep Reinforcement Learning for Trading," Nov. 22, 2019, *arXiv*: arXiv:1911.10107. Accessed: Feb. 19, 2024. [Online]. Available: http://arxiv.org/abs/1911.10107

[7] P. N. Kolm and G. Ritter, "Modern Perspectives on Reinforcement Learning in Finance," *SSRN Journal*, 2019, doi: 10.2139/ssrn.3449401.

[8] L. Chen and Q. Gao, "Application of Deep Reinforcement Learning on Automated Stock Trading," in *2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS)*, Beijing, China: IEEE, Oct. 2019, pp. 29–33. doi: 10.1109/ICSESS47205.2019.9040728.

[9] X.-Y. Liu, Z. Xiong, S. Zhong, H. Yang, and A. Walid, "Practical Deep Reinforcement Learning Approach for Stock Trading," Jul. 30, 2022, *arXiv*: arXiv:1811.07522. Accessed: Jul. 27, 2024. [Online]. Available: http://arxiv.org/abs/1811.07522

[10] H. Yang, X.-Y. Liu, S. Zhong, and A. Walid, "Deep Reinforcement Learning for Automated Stock Trading: An Ensemble Strategy," *SSRN Journal*, 2020, doi: 10.2139/ssrn.3690996.

[11] A. Shavandi and M. Khedmati, "A multi-agent deep reinforcement learning framework for algorithmic trading in financial markets," *Expert Systems with Applications*, vol. 208, p. 118124, Dec. 2022, doi: 10.1016/j.eswa.2022.118124.

[12] J. Lussange, I. Lazarevich, S. Bourgeois-Gironde, S. Palminteri, and B. Gutkin, "Modelling Stock Markets by Multi-agent Reinforcement Learning," *Comput Econ*, vol. 57, no. 1, pp. 113–147, Jan. 2021, doi: 10.1007/s10614-020-10038-w.

[13] K. Wu and X.-J. Zeng, "Group-Agent Reinforcement Learning," vol. 14259, 2023, pp. 37–48. doi: 10.1007/978-3-031-44223-0_4.

[14] E. E. Peters, *Fractal Market Analysis: Applying Chaos Theory to Investment and Economics*. in Wiley Finance. Wiley, 1994. [Online]. Available: https://books.google.co.uk/books?id=_bkoySKyc_cC

[15] Y.-F. Chen, W.-Y. Shih, H.-C. Lai, H.-C. Chang, and J.-L. Huang, "Pairs Trading Strategy Optimization Using Proximal Policy Optimization Algorithms," in *2023 IEEE International Conference on Big Data and Smart Computing (BigComp)*, Jeju, Korea, Republic of: IEEE, Feb. 2023, pp. 40–47. doi: 10.1109/BigComp57234.2023.00015.

[16] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-13, no. 5, pp. 834–846, Sep. 1983, doi: 10.1109/TSMC.1983.6313077.

[17] E. F. Fama, "Efficient Capital Markets: A Review of Theory and Empirical Work".

[18] Burton. G. Malkiel, *A Random Walk Down Wall Street*. Norton, New York, 1973.

[19] M. S. Rozeff and W. R. Kinney, "Capital market seasonality: The case of stock returns," *Journal of Financial Economics*, vol. 3, no. 4, pp. 379–402, 1976, doi: https://doi.org/10.1016/0304-405X(76)90028-3.

[20] L. F. Ackert and R. Deaves, *Behavioral Finance: Psychology, Decision-making, and Markets*. South-Western Cengage Learning, 2010. [Online]. Available: https://books.google.co.uk/books?id=bazZQAAACAAJ

[21] R. J. Shiller, "From Efficient Markets Theory to Behavioral Finance".

[22] A. W. Lo, “The Adaptive Markets Hypothesis: Market Efficiency from an Evolutionary Perspective”.
[23] H. K. Baker, G. Filbeck, and H. Kiymaz, *Equity Markets, Valuation, and Analysis*. in Wiley Finance. Wiley, 2020. [Online]. Available: https://books.google.co.in/books?id=PMX4DwAAQBAJ
[24] J. J. Murphy, *Technical Analysis of the Financial Markets: A Comprehensive Guide to Trading Methods and Applications*. in New York Institute of Finance Series. Penguin Publishing Group, 1999. [Online]. Available: https://books.google.co.in/books?id=5zhXEqdr_IcC
[25] J. W. Wilder, *New Concepts in Technical Trading Systems*. Trend Research, 1978. [Online]. Available: https://books.google.co.uk/books?id=WesJAQAAMAAJ
[26] “Detrended Price Oscillator (DPO): Overview, Calculations,” Investopedia. Accessed: Sep. 01, 2024. [Online]. Available: https://www.investopedia.com/terms/d/detrended-price-oscillator-dpo.asp
[27] V. Vaghela, R. Gor, and N. Chavda, “Elliott Wave formation using combination of CCI and DEMA”.
[28] “Original Williams %R | Williams Percent R Indicator (%R).” Accessed: Sep. 04, 2024. [Online]. Available: https://williamspercentr.com/the-original-percent-r
[29] C. D. Kirkpatrick and J. R. Dahlquist, *Technical analysis: the complete resource for financial market technicians*, Third edition. Old Tappan, New Jersey: Pearson Education Inc. publishing as FT Press, 2016.
[30] G. Tunnicliffe Wilson, “Time Series Analysis: Forecasting and Control,5th Edition, by George E. P. Box, Gwilym M. Jenkins, Gregory C. Reinsel and Greta M. Ljung, 2015. Published by John Wiley and Sons Inc., Hoboken, New Jersey, pp. 712. ISBN: 978-1-118-67502-1,” *Journal of Time Series Analysis*, vol. 37, p. n/a-n/a, Mar. 2016, doi: 10.1111/jtsa.12194.
[31] R. J. Hyndman and G. Athanasopoulos, *Forecasting: principles and practice*. OTexts, 2018. [Online]. Available: https://books.google.co.uk/books?id=_bBhDwAAQBAJ
[32] D. A. Dickey and W. A. Fuller, “Distribution of the Estimators for Autoregressive Time Series With a Unit Root,” *Journal of the American Statistical Association*, vol. 74, no. 366, pp. 427–431, 1979, doi: 10.2307/2286348.
[33] H. Akaike, “A new look at the statistical model identification,” *IEEE Transactions on Automatic Control*, vol. 19, no. 6, pp. 716–723, 1974, doi: 10.1109/TAC.1974.1100705.
[34] R. Cont, “Empirical properties of asset returns: stylized facts and statistical issues,” *Quantitative Finance*, vol. 1, no. 2, pp. 223–236, 2001, doi: 10.1080/713665670.
[35] A. A. Ariyo, A. O. Adewumi, and C. K. Ayo, “Stock Price Prediction Using the ARIMA Model,” in *2014 UKSim-AMSS 16th International Conference on Computer Modelling and Simulation*, 2014, pp. 106–112. doi: 10.1109/UKSim.2014.67.
[36] S. Hudnurkar, A. S. Rajeevan, S. Choudhary, P. D. Mukherjee, H. Kumar Jaiswal, and K. Kadam, “Forecasting of Cryptocurrency and Stocks Prices using Machine Learning,” in *2023 International Conference on Innovative Computing, Intelligent Communication and Smart Electrical Systems (ICSES)*, Chennai, India: IEEE, Dec. 2023, pp. 1–9. doi: 10.1109/ICSES60034.2023.10465415.
[37] H. Markowitz, “Portfolio Selection,” *The Journal of Finance*, vol. 7, no. 1, pp. 77–91, 1952, doi: 10.2307/2975974.
[38] K. J. Arrow, *Essays in the Theory of Risk-bearing*. in Markham economics series. North-Holland, 1971. [Online]. Available: https://books.google.co.uk/books?id=vivwAAAAMAAJ
[39] R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction*, Second edition. in Adaptive computation and machine learning series. Cambridge, Massachusetts: The MIT Press, 2018.
[40] R. BELLMAN, “A Markovian Decision Process,” *Journal of Mathematics and Mechanics*, vol. 6, no. 5, pp. 679–684, 1957.
[41] C. J. C. H. Watkins and P. Dayan, “Q-learning,” *Machine Learning*, vol. 8, no. 3, pp. 279–292, May 1992, doi: 10.1007/BF00992698.
[42] V. Mnih *et al.*, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015, doi: 10.1038/nature14236.
[43] V. Mnih *et al.*, “Playing Atari with Deep Reinforcement Learning,” Dec. 19, 2013, *arXiv*: arXiv:1312.5602. Accessed: Sep. 03, 2024. [Online]. Available: http://arxiv.org/abs/1312.5602

[44] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal Policy Optimization Algorithms,” Aug. 28, 2017, *arXiv*: arXiv:1707.06347. Accessed: Feb. 17, 2024. [Online]. Available: http://arxiv.org/abs/1707.06347
[45] M. Babaeizadeh, I. Frosio, S. Tyree, J. Clemons, and J. Kautz, “Reinforcement Learning through Asynchronous Advantage Actor-Critic on a GPU,” Mar. 02, 2017, *arXiv*: arXiv:1611.06256. Accessed: Apr. 26, 2024. [Online]. Available: http://arxiv.org/abs/1611.06256
[46] P. K. R *et al.*, “Analyzing Deep Reinforcement Learning Strategies for Enhanced Profit Generation and Risk Mitigation in Algorithm Stock Trading,” in *2023 6th International Conference on Recent Trends in Advance Computing (ICRTAC)*, Chennai, India: IEEE, Dec. 2023, pp. 766–771. doi: 10.1109/ICRTAC59277.2023.10480823.
[47] L. Buşoniu, R. Babuška, and B. De Schutter, “Multi-agent Reinforcement Learning: An Overview,” in *Innovations in Multi-Agent Systems and Applications - 1*, D. Srinivasan and L. C. Jain, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 183–221. doi: 10.1007/978-3-642-14435-6_7.
[48] M. Németh and G. Szűcs, “Split Feature Space Ensemble Method using Deep Reinforcement Learning for Algorithmic Trading,” in *2022 8th International Conference on Computer Technology Applications*, Kapfenberg Austria: ACM, May 2022, pp. 188–194. doi: 10.1145/3543712.3543722.
[49] tim boone, anthony j reilly, and M. Sashkin, “SOCIAL LEARNING THEORY Albert Bandura Englewood Cliffs, N.J.: Prentice-Hall, 1977. 247 pp., paperbound,” *Group & Organization Studies*, vol. 2, no. 3, pp. 384–385, 1977, doi: 10.1177/105960117700200317.
[50] F. Cross, “The Behavior of Stock Prices on Fridays and Mondays,” *Financial Analysts Journal*, vol. 29, no. 6, pp. 67–69, 1973.
[51] P. Treleaven, M. Galas, and V. Lalchand, “Algorithmic trading review,” *Commun. ACM*, vol. 56, no. 11, pp. 76–85, Nov. 2013, doi: 10.1145/2500117.
[52] G. Brockman *et al.*, “OpenAI Gym,” Jun. 05, 2016, *arXiv*: arXiv:1606.01540. Accessed: Apr. 30, 2024. [Online]. Available: http://arxiv.org/abs/1606.01540
[53] W. F. Sharpe, “The Sharpe Ratio,” 1994. [Online]. Available: https://api.semanticscholar.org/CorpusID:55394403
[54] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, “Stable-Baselines3: Reliable Reinforcement Learning Implementations”.
[55] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” Jan. 29, 2017, *arXiv*: arXiv:1412.6980. Accessed: Sep. 03, 2024. [Online]. Available: http://arxiv.org/abs/1412.6980
[56] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, “Optuna: A Next-generation Hyperparameter Optimization Framework,” Jul. 25, 2019, *arXiv*: arXiv:1907.10902. Accessed: Sep. 02, 2024. [Online]. Available: http://arxiv.org/abs/1907.10902
[57] The Financial Conduct Authority, “MAR 7A.3 Requirements for algorithmic trading - FCA Handbook,” Financial Conduct Authority | FCA. Accessed: Sep. 06, 2024. [Online]. Available: https://www.handbook.fca.org.uk/handbook/MAR/7A/3.html
[58] E. Chan, *Quantitative Trading: How to Build Your Own Algorithmic Trading Business*. in Wiley Trading. Wiley, 2009. [Online]. Available: https://books.google.co.uk/books?id=NZlV0M5Ije4C
[59] T.-V. Pricope, “Deep Reinforcement Learning in Quantitative Algorithmic Trading: A Review,” May 31, 2021, *arXiv*: arXiv:2106.00123. Accessed: Feb. 19, 2024. [Online]. Available: http://arxiv.org/abs/2106.00123
[60] Z. Wang *et al.*, “Sample Efficient Actor-Critic with Experience Replay,” Jul. 10, 2017, *arXiv*: arXiv:1611.01224. Accessed: Sep. 06, 2024. [Online]. Available: http://arxiv.org/abs/1611.01224