The University of Hong Kong
Department of Computer Science, Faculty of Engineering

FINDING WINNING STRATEGIES IN
ALGORITHMIC TRADING USING REINFORCEMENT LEARNING

Supervisor: Dr. Ting, Hing Fung

Buntoro, Reynard Verill  (3035663461)
Bhargava, Anishka  (3035660782)
Andrianto, Reno  (3035663473)
ACKNOWLEDGEMENT

First and foremost, we want to express our profound gratitude to Prof. Ting Hing Fung, our supervisor, for his ongoing support of our project and for his patience and in-depth expertise throughout the project. We are also thankful for the hard work, dedication, and continuous effort of our team members, namely Reynard Verill Buntoro, Reno Andrianto and Anishka Bhargava. This project would not be where it is now without the teamwork and cooperation of each one of them. Finally, we want to express my gratitude to each of our family for their unwavering support and role as our inspiration.
According to a study by seekingalpha, the average investor has generated less annualized returns as compared to the compounded S&P market index returns over the past 30 years. The average person with actively managed portfolio achieved about 4% less return than the S&P500 market index across different time horizons. This finding suggests that the average investor would be better off by just passively investing in the overall S&P500 index fund. To aid investors in making more rationale financial decisions through active portfolio management, this project aims to develop a fully automated reinforcement-learning (RL)-based model for financial forecasting that can help the average investor obtain excess profits compared to the market. The RL model covered in this project are developed upon the best performing RL-based algorithms from past research conducted. The algorithms discussed in this report were developed using a combination of notable open-source libraries such as FinRL and OpenAI that were integrated into the functionality of Alpaca Markets as a brokerage account and data collection source using the Google Colab environment. The RL agents were trained using the historical Dow Jones data along with 8 technical indicators, VIX short-term ETF, and a turbulence index in the market. These agents were trained with different environment factors and different data to accommodate different trading strategies such as daily trading and a higher frequency, minutely trading. The results are displayed in the form of historical backtesting and live trading performance on the Dow Jones Index that have taken financial overheads and transaction costs of a trade into account. A graphical user interface (UI) as a data exploratory and backtesting platform for assessing the performance of the RL-based algorithms was also made to aid individual investors in examining the relevant data, tracking the portfolio history, testing the RL agents’ performances on past data, and managing the brokerage account. Current limitations identified such as the ability for users to only interact and trade with the data of Dow Jones 30 Index on the highest frequency of minutely trading. This report highlights the development schedule, analysis of feasibility and implementation of the technologies, the results of this project in the form of Colab Notebook and a full-stack UI dashboard, the difficulties and limitations encountered, and the future plans for this project. Each stage of this project will be discussed in detail in the subsequent sections of this report.
# Table of Contents

**Acknowledgement** .................................................................................................................. 2  
**Abstract** ....................................................................................................................................... 3  
**Table of Contents** ....................................................................................................................... 4  
**List of Abbreviations** .................................................................................................................. 10  

## 1. Introduction

1.1. Reinforcement Learning ........................................................................................................ 11  
1.2. Introduction to Important Financial Jargons ......................................................................... 12  
   1.2.1. Dow Jones Industrial Average Index (DJI) ..................................................................... 12  
   1.2.2. Alpha ............................................................................................................................. 13  
   1.2.3. Technical Indicators ......................................................................................................... 14  
   1.2.4. Turbulence Index ........................................................................................................... 17  
   1.2.5. VIX (VIXY) Short-Term Futures ETF ........................................................................... 18  
1.3. Project Objectives .................................................................................................................. 18  
1.4. Project Outline ....................................................................................................................... 19  

## 2. Literature Review

2.1. Deep Neural Network (DNN) Strategy .................................................................................... 20  
2.2. Long Short-Term Memory (LSTM) Strategy .......................................................................... 21  
2.3. Deep Reinforcement Learning (DRL) Strategies .................................................................. 21  
   2.3.1. Critic-only model ............................................................................................................ 21  
   2.3.2. Actor-only model ........................................................................................................... 22  
   2.3.3. Actor-critic model ......................................................................................................... 22  
   2.3.4. A3C/A2C Algorithm ....................................................................................................... 23  
   2.3.5. PPO Algorithm ............................................................................................................. 23  
   2.3.6. Deep Deterministic Policy Gradient (DDPG) Algorithm .............................................. 24  
   2.3.7. Twin Delayed Deep Deterministic (TD3) Algorithm ..................................................... 25  
2.4. FINRL ...................................................................................................................................... 26
3. METHODOLOGY ........................................................................................................................................... 29

3.1. ENVIRONMENT SETUP ......................................................................................................................... 29

3.1.1. Alpaca Markets Setup ....................................................................................................................... 29

3.1.2. FinRL Setup ......................................................................................................................................... 30

3.1.3. Amazon EC2 Setup ........................................................................................................................... 30

3.1.4. React.JS and Chakra UI ..................................................................................................................... 31

3.2. DATA COLLECTION AND PROCESSING ............................................................................................... 32

3.2.1. Data Collection .................................................................................................................................. 32

3.2.2. Data Splitting for Training and Testing ............................................................................................. 35

3.3. RL ALGORITHM DEVELOPMENT ......................................................................................................... 36

3.4. LIVE TRADING SIMULATION ............................................................................................................... 37

3.5. USER INTERFACE (UI) DASHBOARD DEVELOPMENT ........................................................................... 38

3.5.1. Development of Front-end ................................................................................................................ 40

3.5.2. Development of Back-end .................................................................................................................. 40

3.6. DEPLOYMENT OF PROJECT TO EC2 INSTANCE .................................................................................. 41

4. RESULTS ....................................................................................................................................................... 42

4.1. LITERATURE REVIEW ........................................................................................................................... 42

4.2. DATA COLLECTION PIPELINE ............................................................................................................. 42

4.3. ALGORITHM PERFORMANCE ................................................................................................................. 44

4.3.1. A2C Results ......................................................................................................................................... 44

4.3.2. PPO Results ......................................................................................................................................... 45

4.3.3. DDPG Results ..................................................................................................................................... 46

4.3.4. TD3 Results ......................................................................................................................................... 48

4.3.5. Analysis on the Abrupt Strong Sell Sentiment ................................................................................. 49
List of Tables

Table 1 Dow Jones Industrial Average Components .................................................................13

Table 2: Data Download Pseudocode ......................................................................................32

Table 3: Data Splitting Pseudocode .........................................................................................35

Table 4: Pseudocode of Live Trading Simulation Run Method .................................................38

Table 5: Pseudocode of Bash Script ..........................................................................................41
List of Figures

Figure 1 Average Investor vs S&P500 Returns ................................................................. 11
Figure 2: The diagram of actor-critic architecture ......................................................... 23
Figure 3: Finrl Architecture ....................................................................................... 27
Figure 4: Comparison of RL Algorithms ...................................................................... 28
Figure 5: Raw DJI Data .............................................................................................. 33
Figure 6: DJI Data with Technical Indicators, VIX Index, and Turbulence Index ........ 34
Figure 7: Observation of Missing and Invalid Data on DJI Dataframe ......................... 34
Figure 8: Cleaned and Complete DJI Data .................................................................... 34
Figure 9: Observation of Missing and Invalid Data on Cleaned DJI Dataframe .......... 35
Figure 10: Resulting Shape for Train and Test Dataframes ........................................... 36
Figure 11: Definition for State Space ............................................................................. 36
Figure 12: Environment Arguments ............................................................................. 37
Figure 13: UI Dashboard Architecture ........................................................................ 39
Figure 14: Back-end Structure Diagram ....................................................................... 40
Figure 15: Data Collection Pipeline ............................................................................ 43
Figure 16: The expected Resulting Dataframe ............................................................. 44
Figure 17: Actions Taken by A2c During Backtesting ................................................... 44
Figure 18: A2C Actions on the First 100 Minutes of Backtesting ................................. 45
Figure 19: Actions Taken by PPO During Backtesting .................................................. 45
Figure 20: PPO Actions on the First 100 Minutes of Backtesting ................................. 46
Figure 21: Actions Taken by DDPG During Backtesting .............................................. 47
Figure 22: DDPG Actions on the First 100 Minutes of Backtesting ............................... 47
Figure 23: Actions Taken by TD3 During Backtesting .................................................................48
Figure 24: TD3 Actions on the First 100 Minutes of Backtesting ........................................48
Figure 25: Comparison of the Performances of the Algorithms and DJI ..........................49
Figure 26: Visualization of the RL-agents’ Account Value compared to DJI ................50
Figure 27: Implications of the Performance of the Best Performing Algorithm ............50
Figure 28: Live Trading Simulation ..........................................................................................51
Figure 29: Trade Logs Made During Live Trading Simulation ........................................52
Figure 30: Live Trading Simulation While the Market is Not Open .................................52
Figure 31: Main Landing Page of UI Dashboard .................................................................53
Figure 32: Account Information ............................................................................................53
Figure 33: Daily and Monthly View of Equity History .....................................................54
Figure 34: Table of Positions Held .........................................................................................54
Figure 35: Table of Orders Made ...........................................................................................55
Figure 36: "Stopped" Trading Status .....................................................................................55
Figure 37: "Running" Trading Status .....................................................................................55
Figure 38: Backtesting Results ...............................................................................................56
Figure 39: Embedded Notebook in the Dashboard ............................................................56
LIST OF ABBREVIATIONS

1. Algorithmic Trading: AT
2. Artificial Intelligence: AI
3. Deep Neural Networks: DNN
4. Deep Reinforcement Learning: DRL
5. Dow Jones Industrial Average Index: DJI
7. Reinforcement Learning: RL
8. Application Programming Interface: API
9. Long Short Term Memory: LSTM
10. Advantage Actor Critic: A2C
11. Proximal Policy Optimization: PPO
12. Deep Deterministic Policy Gradient: DDPG
13. Twin-Delayed DDPG: TD3
14. User Interface: UI
1. **INTRODUCTION**

The average investor investing in the American stock market has generated significantly less annualized returns when compared to the compounded S&P500 stock market index over the past 30 years in different time frames [1].

![Investor Returns For Equity Funds vs. S&P 500](image)

**FIGURE 1 AVERAGE INVESTOR VS S&P500 RETURNS**

This suggests that the average investor might lack the financial aptitude and emotional resilience to rationally make profitable decisions in the stock market. It implied that the average investor would attain more satisfactory returns by just investing in the broad stock market index passively over a long period of time. Additionally, the progression of technology over the past decades has also led to the growing adoption of algorithmic trading within the financial markets to improve trading efficiency and speed [2]. Algorithmic trading is generally defined as the utilization of computer programs and algorithms to automate the process of trade execution is utilized to automatically execute the trades using a proposed strategy in an efficient manner [3]. Hence, an RL-based agent can be utilised as a technology in executing trades algorithmically to aid the average investor facing the aforementioned issues and attain excessive returns compared to the market indexes.

1.1. **REINFORCEMENT LEARNING**
Reinforcement learning (RL) is a form of artificial intelligence that incorporates a learning agent that learns an optimal policy by trial and error for sequential decision-making problems [3]. Over time, an RL agent engages with its surroundings. The agent receives a state at each time step \( t \) in a state space \( S \) and chooses an action from an action space \( A \), adhering to a policy that represents the agent’s behavior. In an episodic problem, the agent goes through this process until it reaches a terminal condition, at which point it restarts. With the recent major advancements in deep learning, the combination of deep learning and RL has accomplished new accolades in various fields such as the AlphaGo agent in the Go game that managed to defeat the best professional Go players in the world [3]. Taking this proven track record in mind, RL has not been applied prevalently in the field of finance as it is still a new tool to be explored in the domain [3].

1.2. **INTRODUCTION TO IMPORTANT FINANCIAL JARGONS**

As this project is interconnected with financial concepts, it is important to understand some underlying financial terms used in this project. This project is primarily related to the use of Dow Jones 30 Index data, technical indicators, and other relevant metrics in parallel with an RL agent to predict the relevant actions to be taken given a particular state. The following sections 1.2.1 until 1.2.5 will discuss the terms and definitions of Dow Jones Industrial Average Index (DJI), alpha, technical indicators, turbulence, and VIXY index, and their relevance to the development of this project.

1.2.1. **Dow Jones Industrial Average Index (DJI)**

The Dow Jones Industrial Average Index (DJI) or commonly known as Dow 30 is a second oldest stock market index in the United States that was initially defined to keep track of the 12 most important companies in the United States at the time [4]. The index was first created to simplify financial information of public companies as a benchmark for the public using the stock price average method as its average price [4]. It is now used to compile the information of the 30 most traded stocks in the New York Stock Exchange (NYSE) which includes prominent companies such as Apple. The list of companies in the DJI market index is always changing depending on the respective company’s prominence in the economy over time.
### Dow Jones Industrial Average Components

<table>
<thead>
<tr>
<th>Company</th>
<th>Symbol</th>
<th>Year Added</th>
</tr>
</thead>
<tbody>
<tr>
<td>American Express Co</td>
<td>AXP</td>
<td>1982</td>
</tr>
<tr>
<td>Amgen</td>
<td>AMGN</td>
<td>2020</td>
</tr>
<tr>
<td>Apple Inc</td>
<td>AAPL</td>
<td>2015</td>
</tr>
<tr>
<td>Boeing Co</td>
<td>BA</td>
<td>1987</td>
</tr>
<tr>
<td>Caterpillar Inc</td>
<td>CAT</td>
<td>1991</td>
</tr>
<tr>
<td>Cisco Systems</td>
<td>CSCO</td>
<td>2009</td>
</tr>
<tr>
<td>Chevron Corp</td>
<td>CVX</td>
<td>2008</td>
</tr>
<tr>
<td>Goldman Sachs Group</td>
<td>GS</td>
<td>2013</td>
</tr>
<tr>
<td>Home Depot Inc</td>
<td>HD</td>
<td>1999</td>
</tr>
<tr>
<td>Honeywell International Inc</td>
<td>HON</td>
<td>2020</td>
</tr>
<tr>
<td>International Business Mach</td>
<td>IBM</td>
<td>1979</td>
</tr>
<tr>
<td>Intel Corp</td>
<td>INTC</td>
<td>1999</td>
</tr>
<tr>
<td>Johnson &amp; Johnson</td>
<td>JNJ</td>
<td>1997</td>
</tr>
<tr>
<td>Coca-Cola Co</td>
<td>KO</td>
<td>1987</td>
</tr>
<tr>
<td>JP Morgan Chase &amp; Co</td>
<td>JPM</td>
<td>1991</td>
</tr>
<tr>
<td>McDonald's Corp</td>
<td>MCD</td>
<td>1985</td>
</tr>
<tr>
<td>3M Co</td>
<td>MMM</td>
<td>1976</td>
</tr>
<tr>
<td>Merck &amp; Co Inc</td>
<td>MRK</td>
<td>1979</td>
</tr>
<tr>
<td>Microsoft Corp</td>
<td>MSFT</td>
<td>1999</td>
</tr>
<tr>
<td>Nike Inc</td>
<td>NKE</td>
<td>2013</td>
</tr>
<tr>
<td>Procter &amp; Gamble Co</td>
<td>PG</td>
<td>1932</td>
</tr>
<tr>
<td>Travelers Companies Inc</td>
<td>TRV</td>
<td>2009</td>
</tr>
<tr>
<td>UnitedHealth Group Inc</td>
<td>UNH</td>
<td>2012</td>
</tr>
<tr>
<td>Salesforce Inc</td>
<td>CRM</td>
<td>2020</td>
</tr>
<tr>
<td>Verizon Communications Inc</td>
<td>VZ</td>
<td>2004</td>
</tr>
<tr>
<td>Visa Inc</td>
<td>V</td>
<td>2013</td>
</tr>
<tr>
<td>Walgreens Boots Alliance Inc</td>
<td>WBA</td>
<td>2018</td>
</tr>
<tr>
<td>Walmart</td>
<td>WMT</td>
<td>1997</td>
</tr>
<tr>
<td>Walt Disney Co</td>
<td>DIS</td>
<td>1991</td>
</tr>
<tr>
<td>Dow Inc</td>
<td>DOW</td>
<td>2019</td>
</tr>
</tbody>
</table>

1.2.2. Alpha

Alpha, or commonly referred to as $\alpha$, is a term used to illustrate the capability of an investment strategy to beat a certain market benchmark. It is also known as the excess return as compared to the market or abnormal rate of return in efficient market theory which hypothesizes that markets are efficient, and there is no systematic way to beat their return. In the scope of this project, Alpha
will be used as a metric to determine the successfulness of an agent in performing algorithmic trading. The alpha used in this project will be the excess return against the DJI stock index.

1.2.3. Technical Indicators

Technical indicators are heuristics that are derived from implicit information about the stock price, volume, and open interest of a security of a contract [5]. These indicators are often utilised to predict the future movement of the stock price and are also used as the determining factors in buying or selling a security by the RL agents in this project. The technical indicators involved for the scope of this project are MACD, BOLU, BOLD, RSI 30, CCI 30, DMI 30, SMA 30, and SMA 60.

**MACD (MOVING AVERAGE CONVERGENCE/DIVERGENCE INDICATOR)**

The Moving Average Convergence/Divergence Indicator (MACD) is a tool used by traders to identify potential trend reversals and momentum shifts in financial markets [6]. The MACD is calculated by subtracting the 26-day exponential moving average (EMA) from the 12-day EMA. A nine-day EMA of the MACD, known as the "signal line," is usually plotted on top of the MACD line by traders to act as a trigger for buy and sell signals. The formula for calculating the MACD indicator is as follows:

\[ MACD = 12\text{-Period EMA} - 26\text{-Period EMA} \]

When the MACD line crosses above the signal line, it generates a buy signal, indicating that the momentum is shifting upwards [6]. Conversely, when the MACD line crosses below the signal line, it generates a sell signal, indicating that the momentum is shifting downwards. This indicator is included as one of the determining factors for the RL agent on whether to buy or sell a stock as it can provide useful suggestion when complemented with the other indicators used for this project.

**BOLU (UPPER BOLLINGER BAND)**

The Upper Bollinger Band (BOLU) is a technical analysis indicator that is used to identify potential price resistance levels in financial markets [7]. The BOLU is calculated using the Bollinger Bands formula, which involves adding a specified number of standard deviations to the moving average. The Bollinger Bands formula uses a 20-day simple moving average (SMA) as the default setting, along with two standard deviations. The BOLU is calculated by adding two standard deviations to the 20-day SMA. The formula for calculating BOLU is as follows:
\[ BOLU = MA(TP,n) + m \times \sigma[TP,n] \]

Where

MA = Moving average

TP (typical price) = \((\text{High} + \text{Low} + \text{Close}) \div 3\)

\(n\) = Number of days in smoothing period (typically 20)

\(m\) = Number of standard deviations (typically 2)

\(\sigma[TP,n]\) = Standard Deviation over last \(n\) periods of TP

**BOLD (LOWER BOLLINGER BAND)**

The underlying concept of Lower Bollinger Band (BOLD) also stems from the Bollinger Band indicator. While the BOLU indicator indicates the upper limits on the trend of the price movement of an asset class, the BOLD indicator acts as the lower limits. Its calculation is slightly different from that of BOLU where the plus sign is changed to a subtract sign as defined on the following equation:

\[ BOLD = MA(TP,n) - m \times \sigma[TP,n] \]

Together with BOLU, they synergise to become the upper limit and lower limit of a price movement trend to determine whether the market is overbought or oversold at a particular time [7]. The distance apart in which the bands move suggest the overall volatility level of the market [7]. The farther the bands move, a decrease in volatility level is more likely, and otherwise.

**RSI 30 (RELATIVE STRENGTH INDEX)**

RSI is a momentum oscillator that measures the speed and magnitude of a security’s recent changes in price to evaluate whether it is in an overvalued or undervalued condition [8]. The RSI displays on the oscillator a value that ranges from 0 to 100. The idea is to generally identify whether a particular security is overbought or oversold at a given moment. A value below 30 indicates an oversold condition and that the security may be primed for a trend reversal, whereas a value above 70 implies an overbought situation. The RSI line is often used by traders as a signal to buy or sell a security. It works best in more volatile market where a security is trading in ranges as opposed to a trending market.
\[ RSI_{step \ one} = 100 - \left( \frac{100}{1 + \left( \frac{Avg.\ gain}{Avg.\ loss} \right)} \right) \]

The second calculation can be done after a collection of data from 30 different periods is obtained. The second formula’s aim is to smoothen the results so that RSI only nears the extreme values of 100 or 0 in a strongly trending market [8].

\[ RSI_{step \ two} = 100 - \left( \frac{100}{1 + \left( \frac{Prev\ Avg.\ Gain \times 13 + Current\ Gain}{Prev\ Avg.\ Loss \times 13 + Current\ Loss} \right)} \right) \]

By adding this metric as a consideration for the agent in making trades decision, it is hoped that the agent would be able to consider the cases when the security falls into the overbought and oversold region and identify a proper trading decision in coordination with other information.

CCI 30 (Commodity Channel Index)

CCI was initially developed to sport long-term changes in trends but had been adopted by traders to provide information for trading in different timeframes [9]. Despite its name being tied with commodity, this indicator has been adopted for different asset classes which can be traded in the market [9]. The underlying formula of this indicator is as follows:

\[ CCI = \frac{(Typical\ Price - Simple\ Moving\ Average)}{(0.015 \times Mean\ Deviation)} \]

The indicator typically involves movement across its value in the range of -100 and 100. A movement of value outside of this normal range indicates a relative strength or weakness in price movement and can be used as a signal to buy or sell a particular asset. The CCI 30 that is used for this project uses the data of the 30 most recent data entries for its calculation and inform the RL agent in its decision-making process [10].

DMI 30 (Directional Movement Index)

DMI is an indicator used to identify where the direction of the price of a security is moving. The indicator performs this task by applying a drawing line between prior highs and prior lows [11]. The lines drawn are called the positive directional movement line (+DI) and the negative directional movement line (-DI). When the +DI is above -DI, it is implied that there is a stronger upward pressure in price than a downward pressure [11]. The indicator is often used by traders in assessing the trend direction and crossover between the two lines can be used as a signal to buy or sell. The formulas used to calculate the indicator is as follows:
\[ +DI = \left( \frac{\text{Smoothed} + DM}{ATR} \right) \times 100 \]

\[ -DI = \left( \frac{\text{Smoothed} - DM}{ATR} \right) \times 100 \]

\[ DX = \left( \frac{+DI - -DI}{+DI + -DI} \right) \times 100 \]

where:

\[ +DM \text{ (Directional Movement)} = \text{Current High} - PH \]

\[ PH = \text{Previous high} \]

\[ -DM = \text{Previous Low} - \text{Current Low} \]

\[ \text{Smoothed} + / - DM = \sum_{t=1}^{30} DM - \left( \frac{\sum_{t=1}^{30} DM}{30} \right) + CDM \]

\[ CDM = \text{Current DM} \]

\[ ATR = \text{Average True Range} \]

**SMA (Simple Moving Average)**

SMA is simply the average of a range of prices within a period. SMA is an indicator that can aid in determining if an asset price will continue or if it will reverse a bull or bear trend. For this project, the SMA indicators used are SMA-30 and SMA-60 which are simply the average price of a security over the previous 30 entries and 60 entries periods within the data collected. The formula for SMA is as follows:

\[ SMA_n = \frac{P1 + P2 + \cdots + Pn}{n} \]

where:

\[ Pn = \text{the price of an asset at period } n \]

\[ N = \text{the number of total periods} \]

A trading signal is usually generated when a SMA of shorter time-period crosses a SMA of longer time-period, indicating a trend of movement in security prices. For this project, SMA 30 and SMA 60 are chosen for aiding the agent in the decision-making process of whether to trade a security.

1.2.4. Turbulence Index
The turbulence index is often used for measuring financial risk. The index is anticipated to quantify the level of unusual behaviour among a universe of assets [12]. The intuition behind this is that the more asset returns, volatilities, and correlations deviate from their historical trends, the more likely it is that these differences arise from a significant market event instead of unsystematic risks imposed by securities or random noise [12]. Increased volatility and a decline in asset correlations are the typical characteristics of financial market crisis periods. The turbulence index is calculated as the following as computed in the Python portfolio optimizer library:

\[ dt = \frac{1}{n} \left( r_t - \mu_t \right)^t \sum_{t-1}^{t} \left( r_t - \mu_t \right) \]

The turbulence index is used to determine whether the agent should exit the positions of the total portfolio holdings completely. The rationale is because it can be very risky to trade in especially volatile periods of the market that might lead to disastrous outcome.

1.2.5. VIX (VIXY) Short-Term Futures ETF

The final indicator will be considered by the agent is the VIX short-term futures ETF. VIXY ETF or commonly known as the “fear index” essentially gauges how the consensus are about the future of the market for the next 30 days [13]. It has been used as a hedging strategy by investors as the price has historically been negatively correlated to the S&P500 returns [14]. The VIXY ETF mirrors the VIX through using S&P500 VIX Short-Term Futures index through the examination of a bundle of future contracts with a weighted average of expiration to one month [13]. Therefore, it is expected that this index can assist the agent in evaluating the outlook for trading stocks at a particular moment.

1.3. PROJECT OBJECTIVES

Taking the aforementioned profit maximization problem and the potential of reinforcement learning into account, this project aims to develop a RL-based strategy in algorithmic trading to aid the average investor achieve excess profits when trading in the market. The problem identified can be reduced to a multi-stock trading problem in the Dow 30 Index using RL. This project aims to develop a working RL-based learning agent that can take rational buying or selling decisions on multiple stocks in DJI based on technical indicator. In order to achieve this goal, a review on relevant literatures was done to understand the subject on RL and basic financial terms better. After having understood the subject matter and compiled the relevant resources for developing the project, the data collection process up until the development of RL agent will be
done. Subsequently, a UI dashboard will be made for the users to interact with the RL agents developed and to assess the performance of the algorithms on historical market data over different time periods. The dashboard is also expected to serve a purpose to allow the users in interacting with the data gathered and experience the trading simulation with RL directly. Then, the final results of this project will be shown in the form of adjusted annualized return on historical stock prices after taking into account trading and overhead costs.

1.4. PROJECT OUTLINE

This report will discuss the current completion status of the project in detail including the literature review stage, engineering choices and justifications made, data collection process, exploratory data analysis (EDA), algorithm development phase and the readiness of a UI dashboard that allows users to interact directly with the various aspects of this project. The project was finished on track according to the project timeline defined earlier. At the final project timeline, all stages excluding the potential expansion of the project which comprises of literature review about related topics such as AI-based strategies in algorithmic trading, data collection, exploratory data analysis (EDA) to feed the appropriate data to the model, various RL agents training and testing, development of UI dashboard, live trading simulation, and evaluation of the algorithms compared to the market had been done. The challenges faced when curating the end-to-end algorithm development phase and trying to expand the scope of this project will also be discussed in detail. This report will discuss about the findings made and literature gap problems found to be overcome from the sources gathered surrounding the topic of AI-based algorithmic trading strategies in section 2 about literature review. It will then communicate the methodology of the works that had done in section 3 followed by the results made about the project in section 4. It will then discuss the limitations encountered in this project on section 5, and conclude the report with conclusion and future plans on section 6.
2. Literature Review

Existing deep neural network (DNN), long short-term memory (LSTM), deep reinforcement learning (DRL) based algorithms that are implemented on financial markets have demonstrated satisfactory annualized returns with excess returns to markets. The DNN-based strategy discussed in section 2.1 of the report managed to achieve around 81.7% success probability rate when trading in live market condition but does not take trading costs into account. The LSTM-based strategy examined in section 2.2 accomplished profitable results on 6 different stock market indices such as Hang Seng Index (HSI), S&P500 Index, CSI 300 Index, Nifty 500 Index, Nikkei 225 Index, and DJIA Index over a 6-year testing period while taking transaction costs into account. However, the study suggested that the LSTM algorithm might not be optimized enough in terms of the hyperparameter selections. The DRL based strategies which include critic-only DRL, actor-only DRL, and critic-actor DRL in section 2.3 demonstrated promising results but each has their own respective drawbacks in their algorithm. The following sections will discuss the advantages and drawbacks of these algorithms in more detail and explain how they will be used as the foundation of the algorithm to be developed in this project.

2.1. Deep Neural Network (DNN) Strategy

Deep Neural Network (DNN) has been increasing in popularity due to its versatility in handling a wide array of challenging problems. It is a more sophisticated version of neural network that frequently consists of numerous hidden layers between the input layer and the output layer. DNN processes input using sophisticated mathematical modelling. In DNN, the input data are changed nonlinearly at each hidden layer to enable the construction of features. These features are then honed at each subsequent layer until they are high-level features with fewer dimensions. This process continues until the system can learn complex patterns. Through intermediate processing and refinement that takes place in each hidden layer, this sort of neural network can learn high-level abstractions from vast quantities of raw input [15].

A study conducted by Arevalo et al. that leveraged deep neural network (DNN) techniques for forecasting the market showcased satisfactory results with a 66.15% directional accuracy and 2333 successful trades out of 2853 trades without taking trading costs into account [15]. Their study demonstrated that DNN can achieve satisfactory performance on in high volatility time through DNN’s ability to deliver high-performance results through complex set of inputs [15]. This opens opportunities for further development using the existing model variations of DNN.
with RL such as deep recurrent neural networks, Deep Beliefs Networks, Deep Q Learning, etc. when dealing with higher frequency trading problems later in this project.

2.2. **LONG SHORT-TERM MEMORY (LSTM) STRATEGY**

Long Short-Term Memory (LSTM) as a form of Recurrent Neural Network (RNN) has been commonly used in the financial technology sector given its ability to process data at an enormous scale efficiently [16]. It is also well known for its usefulness on predicting time sensitive data. When compared to the implementation of traditional neural network that keeps all the input vectors’ units being independent of each other, the LSTM model adds hidden layers that control the interaction between neighbouring memory cells [15]. Given this added feature, one of the major additions of leveraging LSTM for price forecasting is its capability in selecting information to remember for making predictions [16]. This feature allows LSTM to filter the necessary information for improving the model’s forecasting ability. However, the underlying nature of LSTM also makes it prone to change in relevancy of data for making predictions considering the dynamic nature of financial markets.

Despite of the LSTM model’s nature that is prone to change in relevancy of data, a strategy proposed by Bao et al. integrating wavelet transforms, stacked autoencoders, and long short-term memory (LSTM) managed to gain around 63.98% of yearly earnings on average in the Dow Jones Industrial (DJI) Index during its 6 years of the testing period [16] [17]. This result suggests that the modification to the traditional LSTM model can overcome its drawback on handling dynamic data, and can be combined with the RL agent developed for this project in later experimentation.

2.3. **DEEP REINFORCEMENT LEARNING (DRL) STRATEGIES**

Deep reinforcement learning (DRL) is a subcategory of machine learning that combines deep learning and reinforcement learning, resulting in a brain-like machine where it can learn through a set of finite actions [17]. Like how a normal RL operates, the machine keeps on learning through a series of trial and errors which makes this model suitable for everchanging dynamic environments such as the stock market. The DRL algorithm has an impressive track record in solving complicated games such as the notable DeepMind’s AlphaGo bot that managed to beat a Go grandmaster [17]. Given the performances of this algorithm on complex games in the past, it is expected that this algorithm will also perform well within the stock market space.

2.3.1. Critic-only model
The first form of DRL strategy examined in the study conducted by Pricope is the critic-only DRL model [18]. This model aims to solve a discrete action space problem. Therefore, certain mechanisms are needed to trick the model into thinking that there is only a finite set of actions in a discrete space at a given time in the stock market, despite the stock market having a continuous value [18]. A study conducted by Chen et al. managed to return a total of 22-23% on the S&P500 stock index annually with the limitation that the agent is only constrained to trade this particular index [18] [19]. Additionally, another implementation of this algorithm using a total of 16 features with normalized returns as training dataset successfully returned 10% on average on 12 different currencies under the foreign exchange (FX) market, with the agent returning a total of 60% on some currencies [18]. These results indicates that these algorithms can be used as references for the initial development of algorithm for this project.

2.3.2. Actor-only model
Secondly, another DRL strategy evaluated was the actor-only DRL model. This learning agent is different from the critic-only DRL model as it learns a direct mapping of the actions to be performed in a particular state, causing the action space to be possibly interpreted as continuous [18]. However, the overall annualized returns could not be assessed as the writers forgot to include the initial capital used for their study.

2.3.3. Actor-critic model
The final DRL strategy assessed was the actor-critic DRL model. It is the least explored out of the other 2 RL despite of its success [18]. This third type of RL combines the two forms of previously mentioned RL model and trains them simultaneously [18]. One of the major advantages of actor-critic DRL model over the actor-only and critic-only model is that the training data distributions over observations and rewards are constantly changing as the agent learns which is a major cause of instability. One of the experiments on this topic managed to return around 15% annually on the DJIA 30 US stock index over a 4 year time horizon when incorporating trading fees into account [18]. The approach taken in this study is however more conservative as it employed a financial turbulence index that measures extreme price movements in the market that leads the algorithm to exit the market completely in times of market instability. This can potentially harm the expected results as previous studies have shown that a volatile market can instead be leveraged to generate supernormal profits.
2.3.4. A3C/A2C Algorithm

Further improvements of the actor-critic were discussed in the research paper named Asynchronous Methods for Deep Learning [20]. The algorithm, namely the Asynchronous Advantage Actor-Critic (A3C), was developed by Google DeepMind, and was initially published in 2016 [20]. This algorithm works by introducing a new value called the advantage value to the traditional actor-critic method, which intuitively serves as a metric of how much better it is to take a specific action compared to the average, general action at a given state. It can be simply obtained by subtracting the Q value of a specific state and action pair by the V value (calculated using state value function).

\[ A(s_t, a_t) = Q_w(s_t, a_t) - V_v(s_t) \]

As for the asynchronous element of A3C, the team members will not be researching much further into this because it was discovered from an OpenAI blog post that the asynchrony did not evidently led to a better performance compared to a synchronous version of A3C, namely Advantage Actor-Critic (A2C) [21]. Furthermore, the post also mentioned that A2C implementation is said to be much more cost efficient in single GPU machines and is much simpler in implementation [21]. Hence, the team members will focus on the implementation using A2C instead for this particular project.

2.3.5. PPO Algorithm

Proximal Policy Optimization (PPO) algorithm trains agents to carry out tasks in environments via on-policy reinforcement learning. An on-policy algorithm updates the policy while it is interacting with the environment.
PPO was introduced by OpenAI in 2017. It has been effectively used for a variety of applications, including robotics control, game playing, and natural language processing [22]. The fundamental tenet of PPO is to keep the policy update as close to the original policy as possible while still using a surrogate objective function to approximate the performance of the policy. This is achieved by constraining the policy update to be within a certain "trust region" around the current policy. The PPO algorithm has two main components, namely, a sampling step and an optimization step [22]. In the sampling step, the agent collects a batch of trajectories by interacting with the environment using the current policy [22]. In the optimization step, the surrogate objective function is optimized using the collected data to update the policy [22].

The sampling step is important to correct for the fact that the distribution of actions collected by the agent may be different from the distribution of actions that would be generated by the updated policy. This helps to improve the stability of the algorithm and prevent it from making large updates that could lead to policy oscillations or divergence [22]. The surrogate objective function on the other hand, is a combination of the policy's objective function from the original reinforcement learning problem and a "clipped" objective function that limits how far the updated policy can deviate from the original policy [22]. The clipping helps to ensure that the updated policy is not too different from the original policy, which can prevent it from diverging or becoming unstable.

One of the major benefits of PPO is that it is relatively simple to implement and can be applied to a wide range of tasks and environments. It has been shown to perform well on both continuous and discrete action spaces, and can be used with a variety of neural network architectures. Despite of the aforementioned advantages, PPO has some shortcomings such as high sensitivity to hyperparameters and its difficulty to optimize for policies with complex It can be sensitive to the choice of hyperparameters and the design of the reward function, and it may require a large amount of data to achieve good performance. Additionally, it can be difficult to optimize policies for complex tasks with high-dimensional state and action spaces.

Overall, PPO is a powerful and widely-used algorithm in reinforcement learning that has contributed to many recent advances in the field. As this algorithm has not been explored extensively within the algorithmic trading field, this project finds it suitable to develop an agent based on this algorithm and examine its performance relative to other algorithms and the market.

2.3.6. Deep Deterministic Policy Gradient (DDPG) Algorithm
Contrary to PPO algorithm, DDPG is an off-policy RL algorithm that is primarily used to train agents to perform continuous control tasks in environments [23]. It was introduced by Lillicrap et al. in 2016, and has been successfully applied to a wide range of tasks, including robotic manipulation, locomotion, and game playing [23].

The key idea behind DDPG is to use a deterministic policy, indicating that the agent directly outputs an action rather than sampling from a probability distribution [23]. This allows for more efficient exploration of the action space and can lead to better performance on tasks with continuous action spaces. In the context of algorithmic trading, the actions represent the buy/sell decisions for a given security.

Like other RL algorithms, DDPG uses a reward signal to guide the learning process. In the context of algorithmic trading, the reward is typically based on the profits or returns generated by the trading strategy. The agent learns to maximize the expected reward by adjusting its policy over time. It uses a replay buffer to store past experiences and a target network to estimate the Q-values to improve the stability and efficiency of the algorithm [23]. The replay buffer allows the agent to learn from a diverse set of experiences, while the target network helps to stabilize the learning process by reducing the variance of the updates. Given these implied advantages and features, DDPG is therefore an interesting field to be explored in the context of algorithmic trading.

2.3.7. Twin Delayed Deep Deterministic (TD3) Algorithm

TD3 is an off-policy reinforcement learning algorithm that is an extension of the Deep Deterministic Policy Gradient (DDPG) algorithm. It was introduced by Fujimoto et al. in 2018 and has been used to learn optimal policies for a wide range of tasks, including robotic manipulation, locomotion, and game playing [24]. The key idea behind TD3 is to use multiple critics to estimate the Q-value, which is the expected return given a state and action. This helps to reduce the variance of the Q-value estimates and improve the stability of the algorithm [24]. Additionally, TD3 uses delayed policy updates, which means that the policy is updated less frequently than the critics. This can help to prevent overfitting and improve the generalization of the learned policy.

TD3 uses a neural network to represent the policy and the critics. The policy is updated using the deterministic policy gradient, while the critics are updated using the mean squared error loss. The updates are also constrained to be within a certain range, which helps to prevent the policy from deviating too far from the current policy and improve stability. To improve the efficiency
of the learning process, TD3 uses a replay buffer to store past experiences and a mini-batch sampling method to update the policy and critics in batches. The replay buffer allows the agent to learn from a diverse set of experiences, while the mini-batch sampling method helps to reduce the variance of the updates and improve convergence [24]. As an extension of DDPG, TD3 is a compelling algorithm to be tried within the field of algorithmic trading.

2.4. FinRL

FinRL (Finance Reinforcement Learning) is a Python library for deep reinforcement learning in finance. It is designed to help researchers and practitioners develop and test trading algorithms using reinforcement learning techniques. According to the developers of FinRL, the library provides a flexible and customizable framework for creating trading strategies. It proved to be helpful in developing and implementing the algorithms in this project as will be discussed in section 3, and subsections 4.3 and 4.4 as it provides an ecosystem for integrating the end-to-end data pipelining with model training along with live trading simulation.

2.4.1. Three-layer architecture

The FinRL architecture is composed of three modular layers: FinRL-Meta for market environments, DRL agents, and applications [25]. The bottom layer offers APIs for the top layer, ensuring that the top tier does not interact with the lower layer directly. The agent layer uses an exploration-exploitation approach to interact with the environment layer, choosing between repeating successful past decisions or making new decisions with the expectation of achieving higher cumulative rewards.
In the market environments layer, FinRL focuses mostly on pre-processing of market data, which then are used to build stock market environments [25]. The environments observe the change in prices and other features, which prompt the agent to act and receive reward from the same environment so that it can suitably adjust its strategy to maximize the long-term rewards, also known as Q-values. The trading environments that are available to use in FinRL are all based on OpenAI gym, and they use real time financial data to simulate the markets [25]. On top of that, the market environments can access data across different stock exchanges, which makes FinRL applicable to many different stock exchanges.

The middle layer is where the DRL Agents are contained. The module provides standard fine-tuned DRL algorithms included in ElegantRL, Stable Baselines 3 (SB3), and DRLlib [25]. Although the developers of FinRL claimed that ElegantRL is a faster and more stable implementation of both SB3 and RLLib, our team will focus on implementing SB3 in the project instead, due to its complete documentation, cleaner codebase, and much wider support due to its popularity. Within these libraries, the supported algorithms include A2C, PPO, DDPG, Multi-Agent DDPG, SAC, and TD3 [25]. These algorithms are highly adaptable, which makes them possible to be designed further to different uses cases (in this case, stock market environments).

The final layer focuses on applications and variety of use cases that can be performed directly in the trade market, which includes single stock trading, multi stock trading, portfolio allocation, high frequency trading, and even cryptocurrency trading. However, this project is specifically
focused on automated and multi-stock trading, and therefore, only those usage scenarios will be considered.

2.5. PROJECT AIM ON THE DEVELOPMENT OF THE ALGORITHM

Given the lack of extensive research and the potential of these algorithms on the applications of algorithmic trading, the A2C, PPO, DDPG, and TD3 as multi-stock trading algorithms will be developed for this project. The critic-only and actor-critic will not be examined given the respective limitations of these algorithms. This project will incorporate FinRL as a development environment, connecting the data gathering pipelines with the agents and their respective trading environment and back to the market layer for live trading simulation on real time data based on these trained models. LSTM will also be excluded from the scope of this project given the lack of implementation of the algorithm within the algorithmic trading field and insufficient sample of good track records on historical data.

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Input</th>
<th>Output</th>
<th>Type</th>
<th>State-action spaces support</th>
<th>Finance use cases support</th>
<th>Features and Improvements</th>
<th>Advantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>DQN</td>
<td>States</td>
<td>Q-value</td>
<td>Value based</td>
<td>Discrete only</td>
<td>Single stock trading</td>
<td>Target network, experience replay</td>
<td>Simple and easy to use</td>
</tr>
<tr>
<td>Double DQN</td>
<td>States</td>
<td>Q-value</td>
<td>Value based</td>
<td>Discrete only</td>
<td>Single stock trading</td>
<td>Use two identical neural network models to learn</td>
<td>Reduce overestimations</td>
</tr>
<tr>
<td>Dueling DQN</td>
<td>States</td>
<td>Q-value</td>
<td>Value based</td>
<td>Discrete only</td>
<td>Single stock trading</td>
<td>Add a specialized dueling Q head</td>
<td>Better differentiate actions, improves the learning</td>
</tr>
<tr>
<td>DDPG</td>
<td>State action pair</td>
<td>Q-value</td>
<td>Actor-critic based</td>
<td>Continuous only</td>
<td>Multiple stock trading, portfolio allocation,</td>
<td>Being deep Q-learning for continuous action spaces</td>
<td>Better at handling high-dimensional continuous action spaces</td>
</tr>
<tr>
<td>A2C</td>
<td>State action pair</td>
<td>Q-value</td>
<td>Actor-critic based</td>
<td>Discrete and continuous</td>
<td>All use cases</td>
<td>Advantage function, parallel gradients updating</td>
<td>Stable, cost-effective, faster and works better with large batch sizes</td>
</tr>
<tr>
<td>PPO</td>
<td>State action pair</td>
<td>Q-value</td>
<td>Actor-critic based</td>
<td>Discrete and continuous</td>
<td>All use cases</td>
<td>Clipped surrogate objective function</td>
<td>Improve stability, less variance, simply to implement</td>
</tr>
<tr>
<td>SAC</td>
<td>State action pair</td>
<td>Q-value</td>
<td>Actor-critic based</td>
<td>Continuous only</td>
<td>Multiple stock trading, portfolio allocation</td>
<td>Entropy regularization, exploration-exploitation trade-off</td>
<td>Improve stability</td>
</tr>
<tr>
<td>TD3</td>
<td>State action pair</td>
<td>Q-value</td>
<td>Actor-critic based</td>
<td>Continuous only</td>
<td>Multiple stock trading, portfolio allocation</td>
<td>Clipped double Q-Learning, delayed policy update, target policy smoothing</td>
<td>Improve DDPG performance</td>
</tr>
<tr>
<td>MADDPG</td>
<td>State action pair</td>
<td>Q-value</td>
<td>Actor-critic based</td>
<td>Continuous only</td>
<td>Multiple stock trading, portfolio allocation</td>
<td>Handle multi-agent RL problem</td>
<td>Improve stability and performance</td>
</tr>
</tbody>
</table>

FIGURE 4: COMPARISON OF RL ALGORITHMS

As can be seen on Figure 4 as gathered from FinRL’s paper about the table compiled by the developers of FinRL in discussing the various advantages and unique features of each algorithm, only DDPG, A2C, PPO, SAC, TD3, and MADDPG are suitable for multi stock trading. From these algorithms, A2c, PPO, DDPG, and TD3 are selected for this project due to their respective advantages, unique features, and more available information.
3. **Methodology**

There are five phases necessary in this project to achieve a fully working RL-based learning agent that can aid the individual investor in making educated decision on their investments. In phase 1, the initial environment setup was performed as the foundation of this project. In phase 2, the data collection and processing for model training was done. In phase 3, the algorithms development as a final deliverable and backtesting on historical data will be discussed. In phase 4, live trading demo will be performed to test the models in real world situation. In phase 5, the UI dashboard development will be done for users to interact with the models.

3.1. **Environment Setup**

Python is chosen as the primary language for this research due to the availability of various libraries and tools needed for this research such as Pandas and NumPy, and it is the language that the team members are most comfortable with. Google Colab platform is selected as the development environment for this project due to the convenience it provides by allowing users to run Python scripts and libraries on the cloud with CPU allocation for free. Moreover, it also allows multiple users to edit the notebook and there is no additional configuration required for running a Colab notebook apart from importing the necessary libraries to run the program.

After setting up the programming environment, an account set up for collecting the data from a financial data provider is created. Initially, an account setup on Marketwatch as a financial data provider was performed as it is reliable, easy to use, and flexible in terms of the number of symbols and requests that can be made per day. However, as further research on this project progressed, a pivot to using Alpaca Markets as a reliable data provider was made due to their integrated nature within FinRL that can also directly connect with stable baselines as reliable RL algorithms implementations in PyTorch [25]. The advantages of Alpaca Markets will also be further elaborated briefly in section 3.1.1 about Alpaca Markets setup. Brief justifications for choosing FinRL and EC2 will also be made before explaining further the steps taken to set FinRL as an ecosystem and EC2 as a cloud hosting platform respectively.

3.1.1. **Alpaca Markets Setup**

Alpaca Markets is considered to be more practical and reliable than Marketwatch as it provides a commission-free trading platform and infrastructure for algorithmic trading with its development APIs. Due to their commission-free and accommodating nature for developers, Alpaca Markets was chosen as the suitable brokerage for this project. One of the key features which allow developers to devise trading strategies and build custom trading applications that can be
connected to the platform is aligned with the goals of this project in building an end-to-end algorithmic trading application with RL. Additionally, their ability to offer live trading data with paid subscription services makes them the unequivocal choice for the feasibility of live trading simulation in this project.

For the development of this project, a paper trading account was first set up along with its corresponding API key to be used for function calls. The API key can then be used to obtain relevant information about the market for development purposes. Alpaca’s documentation as provided in https://alpaca.markets/docs/api-references/market-data-api/stock-pricing-data/ gives a clear explanation of how to connect an Alpaca account with a project through the broker API.

3.1.2. FinRL Setup

As previously explained in section 2.4, FinRL provides a robust ecosystem for the integration of the three layers needed for this project, namely the application layer, DRL agent, and market environment [25]. Additionally, the library provides a flexible and customizable framework for creating trading strategies. It proved to be helpful in developing and implementing the algorithms in this project as will be discussed in sections 3.3 and 4.4 as it provides an ecosystem for integrating the end-to-end data pipelining with model training along with the live trading simulation. Given that FinRL is built on top of the OpenAI Gym framework, it can also work with a range of reinforcement learning algorithms with adaptable market scenarios to experiment with in algorithmic trading.

For the initial setup of FinRL for this project in the project directory, a setup documentation as provided by Yang et al. at https://byfintech.medium.com/finrl-for-quantitative-finance-install-and-setup-tutorial-for-beginners-1db80ad39159 was followed. OpenAI was first installed on the local machine, followed by the installation of finrl library using pip, and finished by cloning the FinRL repository into the working directory where the application involving FinRL will be at.

3.1.3. Amazon EC2 Setup

Scalable computing resources are offered by the popular cloud computing service Amazon EC2. Amazon EC2 is particularly well-known to be cost-effective, especially for scientific computing applications that demand high-performance computing resources. Scalability, cost-efficiency, and adaptability make Amazon EC2 the best option for this project. Scalability is crucial because algorithmic trading demands real-time trade execution and processing of massive volumes of data. Since users only pay for the computing resources consumed, the pay-as-you-go pricing
model of EC2 makes it incredibly cost-effective, making it a far more affordable option than buying and operating your own physical server.

Additionally, a variety of instance types that are tailored to different use cases are available through Amazon EC2. These instance types give customers the freedom to choose the best resource combination for their applications because they are made up of different combinations of CPU, memory, storage, and networking capacity. The Amazon EC2 T2.medium instance was chosen to give all the required resources for this project. The trading program was initially attempted to be hosted on the free tier T2.micro. However, it was found that the Python script, which required extensive data processing, could not be performed on the T2.micro instance due to a lack of memory. The choice was made as a result to switch to the T2.medium instance, which offered more memory (4GB) and processing capacity (2vCPUS). To ease the development process, the deployed instance was configured with Linux-based Ubuntu 22.04 platform. The region was set to **us-east-1b (N. Virginia)** by considering that the program is designed to make trades in the US market. To access the instance, a secure shell (SSH) can be used to connect directly using a private Amazon EC2 key pair, which has been configured previously.

### 3.1.4. React.JS and Chakra UI

React.js is a popular JavaScript library for building user interfaces, and it has gained a lot of traction in recent years. React.js follows a component-based architecture, which allows developers to build reusable UI components. This approach results in a more modular and maintainable codebase, which is especially important in a medium to large scale projects.

React.js also utilises a virtual DOM to optimize the performance of UI updates. Instead of updating the actual DOM for every change, React.js updates the virtual DOM and then applies the changes to the actual DOM only when necessary. This approach results in faster and more efficient rendering of complex UI components, which is important in building dashboards that deal with large amounts of financial data.

On top of React.js, Chakra UI library is used in order to provide a set of accessible and reusable components for a faster development, allowing the team to focus on core functionality and features. Its extensive documentation, easy-to-use APIs, and responsive design make it an ideal choice for building financial dashboards that work across multiple devices and platforms. Furthermore, Chakra UI's support for theming and customization ensures that the dashboard can be easily branded and uniquely customized, should the team later choose to continue to launch the product commercially.
3.2. DATA COLLECTION AND PROCESSING

3.2.1. Data Collection

Data collection is needed for the training and validation datasets of the models to be developed. The whole data collection process is aided by the functionality of FinRL’s alpaca data processor class. The class provides the relevant API function calls to accurately and reliably download the necessary data, clean the data, add relevant technical indicators, add VIX index, and add turbulence index from a specific time-period. For the scope of this project, a list of stock tickers within the Dow 30 index are imported from the finrl library as the input for data collection process. Along the development process, multiple API providers have been experimented with and evaluated, including Tiingo API, Marketwatch, and Alpaca. Although initially Marketwatch was chosen for its precise historical market data, the team ultimately decided to use Alpaca due to its direct integration with the FinRL package, which also employs Alpaca Trading API as its live trading platform provider.

Table 2: Data Download Pseudocode

<table>
<thead>
<tr>
<th>Algorithm 1: Download Stock Tick Data and Technical Indicators with 1-Minute Time Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data</strong>: List of stock ticker symbols from Dow 30</td>
</tr>
<tr>
<td><strong>Results</strong>: Pickle file with columns (date, open, high, low, close, volume, tic, macd, boll_ub, boll_lb, rsi_30, cci_30, dx_30, close_30_sma, close_60_sma, vix, turbulence)</td>
</tr>
</tbody>
</table>

1. Import Pandas and Numpy
2. Import necessary libraries from FinRL
3. Import config_tickers from FinRL
4. From finrl.config import INDICATORS
5. DOW_30_tickers <- config_tickers.DOW_30_TICKER
6. Define a FinRL data processor class as DP with API information as input
7. Define starting date as 31 days ago and ending date as today’s date
8. data <- DP.download_data(start_date = starting_date, end_date = ending_date, ticker_list = DOW_30_tickers, time_interval = ‘1min’)
9. data <- DP.clean_data(data)
10. data <- DP.add_technical_indicator(data, INDICATORS)
11. data <- DP.add_vix(data)
data <- DP.add_turbulence(data)
Fill missing data within the dataframe with ffill and bfill methods and replace
infinite values with 0
end

Table 2 explains the pseudocode for the program to download the 1-minute interval data of the Dow 30 stock tickers for this project. It can be seen that the data generated are in pickled format. The program leverages the embedded Alpaca Trading API inside the FinRL’s DataProcessor class to gather the data needed for this project, which includes date and time, closing price, highest price, lowest price, opening price, and volume in the time interval.

<table>
<thead>
<tr>
<th>timestamp</th>
<th>open</th>
<th>high</th>
<th>low</th>
<th>close</th>
<th>volume</th>
<th>trade_count</th>
<th>vwap</th>
<th>tic</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>155.3300</td>
<td>155.5197</td>
<td>155.3200</td>
<td>155.4500</td>
<td>211739</td>
<td>1917</td>
<td>155.444620</td>
<td>AAPL</td>
</tr>
<tr>
<td>1</td>
<td>155.4500</td>
<td>155.5814</td>
<td>155.4500</td>
<td>155.5650</td>
<td>155836</td>
<td>1307</td>
<td>155.515260</td>
<td>AAPL</td>
</tr>
<tr>
<td>2</td>
<td>155.5700</td>
<td>155.5900</td>
<td>155.2400</td>
<td>155.2800</td>
<td>203539</td>
<td>1707</td>
<td>155.414785</td>
<td>AAPL</td>
</tr>
<tr>
<td>3</td>
<td>155.2895</td>
<td>155.4400</td>
<td>155.2350</td>
<td>155.4350</td>
<td>106889</td>
<td>1042</td>
<td>155.330476</td>
<td>AAPL</td>
</tr>
<tr>
<td>4</td>
<td>155.4350</td>
<td>155.5200</td>
<td>155.3700</td>
<td>155.4350</td>
<td>109511</td>
<td>1087</td>
<td>155.450038</td>
<td>AAPL</td>
</tr>
<tr>
<td>205349</td>
<td>149.5200</td>
<td>149.5200</td>
<td>149.5200</td>
<td>149.5200</td>
<td>1197</td>
<td>1</td>
<td>149.520000</td>
<td>WMT</td>
</tr>
<tr>
<td>205350</td>
<td>149.7600</td>
<td>149.7600</td>
<td>149.7600</td>
<td>149.7600</td>
<td>452</td>
<td>2</td>
<td>149.757876</td>
<td>WMT</td>
</tr>
<tr>
<td>205351</td>
<td>149.7999</td>
<td>149.7999</td>
<td>149.7999</td>
<td>149.7999</td>
<td>220</td>
<td>1</td>
<td>149.799900</td>
<td>WMT</td>
</tr>
<tr>
<td>205352</td>
<td>149.5300</td>
<td>149.5300</td>
<td>149.5300</td>
<td>149.5300</td>
<td>150</td>
<td>3</td>
<td>149.526667</td>
<td>WMT</td>
</tr>
<tr>
<td>205353</td>
<td>149.7900</td>
<td>149.7900</td>
<td>149.7900</td>
<td>149.7900</td>
<td>160</td>
<td>4</td>
<td>149.784125</td>
<td>WMT</td>
</tr>
</tbody>
</table>

**FIGURE 5: RAW DJI DATA**

Figure 5 shows how the raw downloaded data from Alpaca Markets API looks like. The data gathered will always have 9 columns, but the number of rows varies on each rundown of the notebook, depending on the time period for data collection. The data was then cleaned, technical indicators were added, and the VIX index price and turbulence index were added to each entry using the API function call from the same DataProcessor object previously defined to gather the data.
Since the cleaning part of the data does not include cleaning and filling the technical indicators columns as can be seen on the observations made in figure 6 and 7, further data cleaning was needed to fill the null values and invalid values. Therefore, further data cleaning operations were performed to handle these problems within the data.

```python
data.isna().sum()
timestamp 0
open 0
high 0
low 0
close 0
volume 0
tic 0
macd 0
boll_ub 30
boll_lb 30
rsi_30 1831
cci_30 5207
dx_30 1800
close_30_sma 0
close_60_sma 0
VIX 0
turbulence 0
dtype: int64
numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
checkdf = data.select_dtypes(include=numerics)
np.isinf(checkdf).sum() + np.isnan(checkdf).sum()
timestamp 0
open 0
high 0
low 0
close 0
volume 0
tic 0
macd 0
boll_ub 0
boll_lb 0
rsi_30 0
cci_30 0
dx_30 0
close_30_sma 0
close_60_sma 0
VIX 0
turbulence 0
dtype: int64
```
Figure 8 and 9 shows the cleaned data needed for the algorithm development of this project and its analysis on missing and infinite values. The cleaned data was then exported as a pickle file to the local machine. The storage of data using pickle files is preferred over the traditional usage of comma-separated value (CSV) files because pickle files take less memory as opposed to CSV files as the data entries are stored in byte values.

3.2.2. Data Splitting for Training and Testing

The cleaned data can then be split into 2 dataframes for training and backtesting. The following table will showcase the pseudocode for performing the data splitting operation using the module function provided by FinRL’s pre-processors library.

**TABLE 3: DATA SPLITTING PSEUDOCODE**

<table>
<thead>
<tr>
<th>Algorithm 2: Splitting Cleaned Dataframe into Training and Testing Dataframes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data:</strong> Cleaned Dataframe</td>
</tr>
<tr>
<td><strong>Results:</strong> Training and Testing Dataframes</td>
</tr>
<tr>
<td>1 from finrl.meta.preprocessor.preprocessors import data_split</td>
</tr>
<tr>
<td>2 data &lt;- data.rename(columns={“timestamp”:”date”})</td>
</tr>
<tr>
<td>3 define START_DATE_TRAIN as the earliest date of the cleaned data</td>
</tr>
<tr>
<td>4 define END_DATE_TRAIN as START_DATE_TRAIN + 24 days</td>
</tr>
<tr>
<td>5 define START_DATE_TEST as END_DATE_TRAIN</td>
</tr>
<tr>
<td>6 define END_DATE_TEST as the latest date of the cleaned data</td>
</tr>
<tr>
<td>7 Define starting date as 30 days ago and ending date as today’s date</td>
</tr>
<tr>
<td>8 Train_df &lt;- data_split(data, START_DATE_TRAIN, END_DATE_TRAIN)</td>
</tr>
</tbody>
</table>
As can be seen from the pseudocode for splitting the dataframe into training and testing dataframes, the training dataset results in cleaned data ranging from the starting time until 24 days after the first datetime entry. The remaining entries will then be allocated for the testing dataset.

![Train and Test Dataframes](image)

**FIGURE 10: RESULTING SHAPE FOR TRAIN AND TEST DATAFRAMES**

### 3.3. RL ALGORITHM DEVELOPMENT

The algorithm development is needed as the final deliverable of this project expects a working RL-based algorithmic trading strategy. The environment StockTradingEnv from FinRL was chosen as the underlying environment for the models with the reward function remained unchanged. This environment only defines two set of actions, namely buying and selling a stock at a given time. Despite of its nature to only consider buying and selling as actions and not selling, it does not affect the rewarding system of the RL in a detrimental way as the trading fee in Alpaca markets is virtually zero.

```
stock_dimension = len(train_df.tic.unique())
state_space = 1 + 2*stock_dimension + len(INdicATORS)*stock_dimension
print(f"Stock Dimension: {stock_dimension}, State Space: {state_space}")
```

Stock Dimension: 30, State Space: 301

**FIGURE 11: DEFINITION FOR STATE SPACE**

The stock dimension on figure 11 is simply defined as the length of unique stock tickers within the dataframe. Since the number of stock tickers in training and testing datasets are always the same, it is simply defined as the unique number of stock tickers in the training dataset. The number assigned to the state_space variable can be interpreted as 1 number of initial amount, 30 entries for closing price of the stocks, 30 entries the previous closing price, and finally 240 entries for 8 technical indicators on the 30 stocks in DJI. The sum of these numbers amount to 301 as defined in the state space.
The environment arguments were set according to figure 12. For stock training environment, a buy cost of 0.001 was set to make the models more conservative when making trading decisions despite the very low to non-existent fees when trading with Alpaca. Hmax argument was set to 100 to restrict the agent from buying more than 100 shares of a stock at a given time to minimize risks involved in the trading mechanism.

This stage was then continued with the training of A2C, DDPG, PPO, and TD3 models. The models were trained using the stable-baselines3 library that is embedded within the DRLAgent class defined in the FinRL ecosystem. The A2C agent was trained with a total of 50,000 timesteps. The PPO agent was trained with the parameters of{"n_steps": 2048, "ent_coef": 0.01, "learning_rate": 0.00025, "batch_size": 128} and a total timesteps of 50,000. The DDPG agent was trained with the default parameters of{ 'batch_size': 128, 'buffer_size': 50000, 'learning_rate': 0.001} and a total timesteps of 50,000. Finally, the TD3 agent was trained with the parameters of{ 'batch_size': 100, 'buffer_size': 1000000, 'learning_rate': 0.001} and a total timesteps of 30,000.

3.4. Live Trading Simulation

The live trading simulation platform was developed on top Alpaca Trade API library. A class AlpacaProcessor was defined along with key functionalities as its class methods. The key functionalities include latency test, trade execution, state retrieval, order submission, and running the live trading simulation server. The implementation on the live trading server makes it only

```python
buy_cost_list = sell_cost_list = [0.001] * stock_dimension
num_stock_shares = [0] * stock_dimension

env_kwars = {
    "hmax": 100,
    "initial_amount": 1000000,
    "num_stock_shares": num_stock_shares,
    "buy_cost_pct": buy_cost_list,
    "sell_cost_pct": sell_cost_list,
    "state_space": state_space,
    "stock_dim": stock_dimension,
    "tech_indicator_list": INDICATORS,
    "action_space": stock_dimension,
    "turbulence_threshold": 30,
    "reward_scaling": 1e-4
}
```

FIGURE 12: ENVIRONMENT ARGUMENTS
runnable whenever the market is open or otherwise will prompt message on the console stating that the desired action is not available.

**TABLE 4: PSEUDOCODE OF LIVE TRADING SIMULATION RUN METHOD**

| Algorithm 3: Running the Live Trading Simulation Server on the Run Method |
|---|---|
| **Inputs:** Self (AlpacaPaperTrading object) |
| 1 | Def run (Self): |
| 2 | Cancel all open order in Alpaca |
| 3 | Wait until the market opens |
| 4 | While True: |
| 5 | If time to market closing is less than 1 minute, close all positions |
| 6 | Else, keep trading and sleep for the time interval |

Although the actual implementation is much more complex than the pseudocode on table 4, it serves as a simplified logic about the underlying mechanism of how the system works.

3.5. **User Interface (UI) Dashboard Development**
After having constructed the models and learning agents, a user interface (UI) dashboard was developed using React.js with a Flask backend for users to interact with the underlying data. The architecture design for the web application is according to Figure 13. A data download script was first made based on the codes for google colab to automatically download new data for the past month. The completion of the download script results in pickled market data for the past month and trigger the model training script to run. These scripts will be managed by the EC2 host to run every week to ensure updated agents that can perform well during live trading based on recent trends in the market. The resulting agents are then utilized for the purpose of user interaction with the live trading simulation and backtesting platform to assess the performance of the algorithms.

The backend server was implemented using Flask in Python as it is easier to be integrated with the Python scripts used for the algorithm development and exploratory data analysis. The backend server was developed with currently 2 major API function calls to serve data for the front-end and will be still maintained in the future. Together with the direct brokerage API, the Flask server serves the necessary data to be displayed on the UI dashboard by the React.js server. The users
will then be able to directly interact with the front-end and communicate with the data used for training and testing, along with RL agents developed and live trading simulation.

3.5.1. Development of Front-end
An open-source admin dashboard template for Chakra UI & React, called Horizon UI, was used to speed up development of the front-end. With highly detailed guidance on how to utilize each one, Horizon UI offers prebuilt, ready-to-use React components. Horizon UI library was used extensively throughout the development of the dashboard of this project. Components such as table, line charts, and others were developed on top of Horizon UI in the dashboard. The resulting dashboard with all the connected components and features can be seen on subsection 4.5.1.

3.5.2. Development of Back-end
The development of the back-end was done according to the implementations of the notebook.

FIGURE 14: BACK-END STRUCTURE DIAGRAM

As can be seen from Figure 14, the back-end hierarchy structure was divided into 6 different major segments. The datasets folder includes the relevant data downloaded in the development of the RL algorithms used for the training of agents. FinRL libraries folders simply store the necessary FinRL libraries adhering to their guidelines on how to incorporate their service in projects by cloning their source repository. The Python script project_script.py contains the main logic of the script which runs every end of the week to re-train the RL agents using more relevant data from the past 31 days. This script imports modules from the project helpers folder to run the functions for collecting data, training, and testing the different RL agents. The Trained Models folder simply stores all the relevant RL agents that had been developed for a week of live trading
and performance assessment through backtesting. Finally, the server.py is simply the backbone of the Flask server that feeds relevant data for the front-end to display.

3.6. DEPLOYMENT OF PROJECT TO EC2 INSTANCE

In the context of setting up the cloud instance for this project, a service that runs the trading needs to be configured to start whenever the instance is booted. For this purpose, systemctl is used to control how the service starts on the Ubuntu platform. This enabled the instance to run a custom script that executes the Python program automatically without further manual intervention.

TABLE 5: PSEUDO CODE OF BASH SCRIPT

<table>
<thead>
<tr>
<th>Algorithm 4: Bash Script to Run Live Trading Server</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Inputs:</strong> -</td>
</tr>
<tr>
<td>1 source to the path of conda inside the instance ending with conda.sh</td>
</tr>
<tr>
<td>2 conda activate environment</td>
</tr>
<tr>
<td>3 run the python script for live trading server on the background</td>
</tr>
</tbody>
</table>

The bash script is written according to the pseudocode as provided in algorithm 4 in table 5. It simply activates an anaconda environment in which all the required packages are installed before executing the live trading Python script. To view the status of the program, systemctl status command can be called to show information about the running service, or the service’s failed attempt.
4. RESULTS

This section will discuss the relevant findings and results that have been made. The subsection 4.1 will elaborate the findings that had been made through literature review and how it had informed the works on this project. The subsection 4.2 articulates the data collection pipeline that had been made for this project, followed by the performance of the RL algorithm compared to the DJI index as a benchmark in subsection 4.3. It will then discuss the live trading implementation results along with its performance on subsection 4.4. Ultimately, the section will be concluded with the presentation of the UI dashboard on subsection 4.5. The subsections 4.1 until 4.5 follow a sequential workflow.

4.1. LITERATURE REVIEW

The review of A2C, PPO, DDPG, and TD3–based algorithms in section 2.5 led to the decision that these 4 algorithms were to be for the algorithm development stage. Despite the promising results of the other algorithms, the renditions of RL agents would need to be able to perform comparatively well in a multi-stock trading environment. Additionally, the literature review on FinRL as discussed in subsection 2.4 led to the adoption of FinRL as the ecosystem for this project. The three-layered architecture of FinRL provides a simple way of developing an end-to-end model training pipeline to be implemented in a live trading simulation. When the RL agents are given exposure to the FinRL ecosystem that provides stability and easiness in accessing reliable trade environments, these RL agents are also expected to perform reasonably well compared to the market.

4.2. DATA COLLECTION PIPELINE

Adhering to the subsections 3.2 and 3.3 on data collection and data splitting, the resulting cleaned data collection pipeline can be seen in the form of a Python module which can be run automatically on the cloud. The data collection and processing processes are well-integrated into and handled by the FinRL library.
As can be seen from Figure 15, the resulting data collection pipeline in the form of a Python module defined for this project is simply a sequence of atomic actions. A failure in one of the action will force the module to re-run until the whole process can be completed. A successful run of the module results in a saved pickled file format and the trigger of the module for model training to run.
As can be seen from Figure 16, a successful completion of the data collection process results in a dataframe with 17 columns and no missing or invalid values. This dataframe is then stored on the local machine in a pickled format.

4.3. ALGORITHM PERFORMANCE

This subsection will evaluate the performance of each respective algorithm and compare them with the DJI market data as the benchmark. Each agent’s performance will be evaluated comprehensively and the general trend among the actions will be observed. These actions analysis will be done on the last 7 days of the stock data previously gathered which spans from the 11th of April 2022 up until the 17th of April 2022. The profitability of each algorithm will then be compared in a consolidated table showcasing the returns of each agent during the backtesting period.

4.3.1. A2C Results

As can be seen from Figure 16, a successful completion of the data collection process results in a dataframe with 17 columns and no missing or invalid values. This dataframe is then stored on the local machine in a pickled format.
As can be seen from Figures 17 and 18, it seems that this agent favours using the stock ticker DOW as its trading commodity. The algorithm started with a strong buying sentiment during the first opening minutes of the market and encountered a strong selling sentiment of the stocks at around time 10:30. It then continued its buying and selling cycle based on each individual company’s indicators as can be seen in the last 2 hills and troughs.

4.3.2. PPO Results

**FIGURE 19: ACTIONS TAKEN BY PPO DURING BACKTESTING**
As can be seen from Figures 19 and 20, it seems that this agent favours using the stock ticker AMGN as its trading commodity. The algorithm started with a considerable amount of buying sentiment but still less than A2C during the first minutes of market opening and encountered a strong selling sentiment of the stocks at around time 10:30, which is about the same time as when the A2C changed its sentiment. It then continued its buying and selling cycle based on each individual company’s indicators as can be seen in the last 2 hills and troughs.

4.3.3. DDPG Results
As can be seen from Figures 21 and 22, it seems that this agent favours using the stock ticker AMGN as its trading commodity, which is similar to what PPO did. The algorithm started with a considerable amount of buying sentiment but still less than A2C during the first minutes of market opening and encountered a strong selling sentiment of the stocks at around time 10:30, which now seems to be a recurring pattern among the agents. It then continued its buying and selling cycle based on each individual company’s indicators as can be seen in the last 2 hills and troughs.
4.3.4. TD3 Results

Similar to how the previous 3 algorithms performed, Figures 23 and 24 showcased that the TD3 agent also started with a considerable amount of buying sentiment after the market first opened and later faced a strong sell signal around the time 10:30. As can be seen from Figures 21 and 22, it seems that this agent favours using the stock ticker AMGN as its trading commodity, which is similar to what PPO did. It then also continued its buying and selling cycle similar to how the previous three agents did as can be seen in the last 2 hills and troughs.
4.3.5. Analysis on the Abrupt Strong Sell Sentiment

When taking a look at one of the stocks that was abruptly sold by the PPO and DDPG agents, namely AMGN, at time 10:30, it can be seen that there was a very drastic change in the values of its indicators during the time leading up to 10:30. The reason behind this unusual anomaly is unknown and the implication of this issue as a limitation for this project will be discussed in section 5 about limitations.

4.3.6. Overall Performance of the Algorithms Compared to DJI

![Figure 25: Comparison of the Performances of the Algorithms and DJI](image)

FIGURE 25: COMPARISON OF THE PERFORMANCES OF THE ALGORITHMS AND DJI
As can be seen from Figures 25 and 26, 2 agents, namely DDPG and TD3 managed to beat the DJI performance by a considerably significant margin over the backtesting time period. The 2 leading algorithms which are TD3 and DDPG managed to beat the DJI by about 55%. On the other hand, A2C and PPO agents’ performances were also not that far off when compared to the benchmark DJI data. A2C and PPO trails DJI in performance as their returns were merely 0.56% and 0.3% each when compared to the 1% return of DJI during the time period. It can be concluded that each of the algorithm managed to perform sufficiently during the period as they were able to generate some returns over the period. Out of the 4 RL agents trained, only 2 managed to produce alpha against the market during the testing period.

<table>
<thead>
<tr>
<th>Backtest</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Annual return</td>
<td>116.577%</td>
</tr>
<tr>
<td>Cumulative returns</td>
<td>1.545%</td>
</tr>
<tr>
<td>Annual volatility</td>
<td>2.558%</td>
</tr>
<tr>
<td>Sharpe ratio</td>
<td>37.84</td>
</tr>
<tr>
<td>Calmar ratio</td>
<td>NaN</td>
</tr>
<tr>
<td>Stability</td>
<td>0.99</td>
</tr>
<tr>
<td>Max drawdown</td>
<td>0.0%</td>
</tr>
<tr>
<td>Omega ratio</td>
<td>NaN</td>
</tr>
<tr>
<td>Sortino ratio</td>
<td>inf</td>
</tr>
<tr>
<td>Skew</td>
<td>NaN</td>
</tr>
<tr>
<td>Kurtosis</td>
<td>NaN</td>
</tr>
<tr>
<td>Tail ratio</td>
<td>2.59</td>
</tr>
<tr>
<td>Daily value at risk</td>
<td>0.062%</td>
</tr>
<tr>
<td>Alpha</td>
<td>NaN</td>
</tr>
<tr>
<td>Beta</td>
<td>NaN</td>
</tr>
</tbody>
</table>

FIGURE 26: VISUALIZATION OF THE RL-AGENTS’ ACCOUNT VALUE COMPARED TO DJI

FIGURE 27: IMPLICATIONS OF THE PERFORMANCE OF THE BEST PERFORMING ALGORITHM
As can be seen in Figure 27, the implied annual return of the TD3 algorithm should it be able to maintain its performance throughout the whole year is 116.577%. The cumulative one-week return is about 1.55% with annual volatility estimated at about 2.56%. The Sharpe ratio, which is used for measuring risk-adjusted return, is estimated at about 37.84 which is considered a highly lucrative investment with very low downside risks. However, the other measurement metrics are outside of the scope of this project and only serve as a reference for the results.

4.4. **LIVE TRADING SIMULATION**

```
paper_trading_erl.run()
```

Waiting for market to open...
Market opened.
stocks[3] = 100.0
action[3] = 100
Market order of 100 BA sell | completed.
stocks[5] = 58.0
action[5] = 58
Market order of 58 CRM sell | completed.
stocks[8] = 1.0
action[8] = 1
Market order of 1 DIS sell | completed.
stocks[11] = 11.0
Market order of 11 HD sell | completed.
stocks[12] = 89.0
action[12] = 7
Market order of 7 HON sell | completed.
stocks[19] = 227.0
action[19] = 100
Market order of 100 MMM sell | completed.
stocks[25] = 49.0
action[25] = 49
Market order of 49 UNH sell | completed.
stocks[29] = 200.0
action[29] = 100
Market order of 100 WMT sell | completed.

Figure 28 shows a sample run of the live trading simulation which was done before during live market hours. The console would then prompt relevant texts to demonstrate the actions that the RL had taken given a specific set of states collected from the market in real-time. The paper trading object will first wait until the market and only start making trades when after it is open. The possibility of invalid operations such as division by zero that might happen due to a fault in the API data service was also handled as can be seen from the prompt in Figure 28. The resulting...
decisions made by the RL agents will be in the form of live trade execution on the brokerage account.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Qty</th>
<th>Filled Qty</th>
<th>Side</th>
<th>Type</th>
<th>Time in Force</th>
<th>Limit Price</th>
<th>Stop Price</th>
<th>Filled Avg Price</th>
<th>Notional</th>
<th>Amount</th>
<th>Status</th>
<th>Fee</th>
<th>Source</th>
<th>Submitted at</th>
<th>Filled at</th>
</tr>
</thead>
<tbody>
<tr>
<td>WMT</td>
<td>85</td>
<td>85</td>
<td>buy</td>
<td>market</td>
<td>day</td>
<td>-</td>
<td>-</td>
<td>$148.50</td>
<td>-</td>
<td>-</td>
<td>filled</td>
<td>0.00</td>
<td>access_key</td>
<td>2023-04-14 00:11:32</td>
<td>2023-04-14 00:11:33</td>
</tr>
<tr>
<td>JPM</td>
<td>1</td>
<td>1</td>
<td>buy</td>
<td>market</td>
<td>day</td>
<td>-</td>
<td>-</td>
<td>$128.72</td>
<td>-</td>
<td>-</td>
<td>filled</td>
<td>0.00</td>
<td>access_key</td>
<td>2023-04-14 00:11:32</td>
<td>2023-04-14 00:11:32</td>
</tr>
<tr>
<td>V</td>
<td>24</td>
<td>24</td>
<td>sell</td>
<td>market</td>
<td>day</td>
<td>-</td>
<td>-</td>
<td>$231.33</td>
<td>-</td>
<td>-</td>
<td>filled</td>
<td>0.00</td>
<td>access_key</td>
<td>2023-04-14 00:11:32</td>
<td>2023-04-14 00:11:32</td>
</tr>
<tr>
<td>HON</td>
<td>33</td>
<td>33</td>
<td>sell</td>
<td>market</td>
<td>day</td>
<td>-</td>
<td>-</td>
<td>$194.22</td>
<td>-</td>
<td>-</td>
<td>filled</td>
<td>0.00</td>
<td>access_key</td>
<td>2023-04-14 00:11:32</td>
<td>2023-04-14 00:11:32</td>
</tr>
<tr>
<td>AMGN</td>
<td>1</td>
<td>1</td>
<td>sell</td>
<td>market</td>
<td>day</td>
<td>-</td>
<td>-</td>
<td>$250.66</td>
<td>-</td>
<td>-</td>
<td>filled</td>
<td>0.00</td>
<td>access_key</td>
<td>2023-04-14 00:11:32</td>
<td>2023-04-14 00:11:32</td>
</tr>
<tr>
<td>WMT</td>
<td>100</td>
<td>100</td>
<td>buy</td>
<td>market</td>
<td>day</td>
<td>-</td>
<td>-</td>
<td>$148.44</td>
<td>-</td>
<td>-</td>
<td>filled</td>
<td>0.00</td>
<td>access_key</td>
<td>2023-04-14 00:11:32</td>
<td>2023-04-14 00:11:32</td>
</tr>
<tr>
<td>INTC</td>
<td>1</td>
<td>1</td>
<td>buy</td>
<td>market</td>
<td>day</td>
<td>-</td>
<td>-</td>
<td>$32.14</td>
<td>-</td>
<td>-</td>
<td>filled</td>
<td>0.00</td>
<td>access_key</td>
<td>2023-04-14 00:11:32</td>
<td>2023-04-14 00:11:32</td>
</tr>
<tr>
<td>IBM</td>
<td>68</td>
<td>68</td>
<td>buy</td>
<td>market</td>
<td>day</td>
<td>-</td>
<td>-</td>
<td>$126.65</td>
<td>-</td>
<td>-</td>
<td>filled</td>
<td>0.00</td>
<td>access_key</td>
<td>2023-04-14 00:11:32</td>
<td>2023-04-14 00:11:32</td>
</tr>
<tr>
<td>HD</td>
<td>100</td>
<td>100</td>
<td>buy</td>
<td>market</td>
<td>day</td>
<td>-</td>
<td>-</td>
<td>$289.97</td>
<td>-</td>
<td>-</td>
<td>filled</td>
<td>0.00</td>
<td>access_key</td>
<td>2023-04-14 00:11:32</td>
<td>2023-04-14 00:11:32</td>
</tr>
<tr>
<td>WBA</td>
<td>2</td>
<td>2</td>
<td>sell</td>
<td>market</td>
<td>day</td>
<td>-</td>
<td>-</td>
<td>$25.55</td>
<td>-</td>
<td>-</td>
<td>filled</td>
<td>0.00</td>
<td>access_key</td>
<td>2023-04-14 00:11:32</td>
<td>2023-04-14 00:11:32</td>
</tr>
</tbody>
</table>

**FIGURE 29: TRADE LOGS MADE DURING LIVE TRADING SIMULATION**

Figure 29 shows the table for order logs on the brokerage account. It can be seen that many of these orders were submitted at the same time simultaneously. These results are due to the multiple stock orders made by the agent at a time when performing live trading.

**FIGURE 30: LIVE TRADING SIMULATION WHILE THE MARKET IS NOT OPEN**

In the case when the user might be tempted to run the live trading simulation when the market is not open, a text waiting for the market to open will be prompted out in the console as can be seen in Figure 30.

4.5. **User Interface Dashboard**

To aid the users in interacting with the data, progress, results, and agents developed in this project, a UI dashboard was developed. The UI dashboard follows the architecture defined in subsection 3.6 and is made using the combination of React.js and Python Flask, along with the help of Python scripts.

4.5.1. **Front-End Development**
Several crucial features of the application's front-end will be covered in-depth in this part.

![Main Dashboard](image)

**FIGURE 31: MAIN LANDING PAGE OF UI DASHBOARD**

The user will see the primary dashboard page as shown in Figure 31 when they first enter the landing page.

![Account Information](image)

**FIGURE 32: ACCOUNT INFORMATION**

Users can view the general statistics of project's trading account at the top as shown in Figure 32. The four main data presented are the equity, purchasing power, profit/loss change, and total cash. Equity or otherwise defined as the worth of a trader's trading account, including their balance and any gains or losses from open positions, is shown to be $103,895.31 as can be seen in Figure 32. This total equity larger than the initial $100,000 indicates that the live trading simulation had been profitable. As further shown in Figure 32, Profit/loss change determines the daily profit or loss within the account, buying power reflects the surplus equity, which is the sum of money available for the customer to acquire assets in a leveraged or margin account.
The line chart in Figure 33 serves as a way to visualize the equity value history using a single continuous line. A feature of changing between daily, monthly, and yearly enables users to change the way they view the chart. This serves to help in perceiving how their portfolio changes across different timeframes. The users can alter the mode of viewing by simply pressing the buttons on the top left of the chart.

<table>
<thead>
<tr>
<th>Positions</th>
<th>ASSET</th>
<th>PRICE</th>
<th>QUANTITY</th>
<th>MARKET VALUE</th>
<th>PROFIT LOSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMZN</td>
<td>$250.00</td>
<td>1</td>
<td>250</td>
<td>189.99</td>
<td></td>
</tr>
<tr>
<td>HD</td>
<td>$292.19</td>
<td>100</td>
<td>29219</td>
<td>222</td>
<td></td>
</tr>
<tr>
<td>HDN</td>
<td>$195.05</td>
<td>49</td>
<td>96030.95</td>
<td>-28.75</td>
<td></td>
</tr>
<tr>
<td>IBM</td>
<td>$226.30</td>
<td>68</td>
<td>15713.6</td>
<td>105.4</td>
<td></td>
</tr>
<tr>
<td>INTC</td>
<td>$191.00</td>
<td>1</td>
<td>191</td>
<td>-0.24</td>
<td></td>
</tr>
<tr>
<td>JPM</td>
<td>$188.75</td>
<td>1</td>
<td>188.75</td>
<td>1003</td>
<td></td>
</tr>
<tr>
<td>MMM</td>
<td>$215.77</td>
<td>27</td>
<td>58012.79</td>
<td>952.48</td>
<td></td>
</tr>
<tr>
<td>WMT</td>
<td>$145.48</td>
<td>285</td>
<td>42386.8</td>
<td>489.3</td>
<td></td>
</tr>
</tbody>
</table>
Another integral component of the dashboard is the table which lists the current positions held by the account provided as can be seen in Figure 34. In the table, users can view the stock ticker of each position held with the prices, quantity bought, market value, and gains/losses from the time the asset was acquired. These positions dynamically change whenever the trading algorithm is running.

Another table is placed below the positions table as can be seen on Figure 35. The table lists all the orders that were made by the trading algorithm. Each row includes the ticker symbol of the asset, type of order (buy/sell), quantity, average cost, total amount, status, and order date.

As can be seen in Figures 36 and 37, an indicator displaying the state of the trading instance is provided in the navigation bar at the top of the page. The indicator's function is to indicate whether or not the trading algorithm is active. The indicator should ideally say "running" during U.S. trading hours to show that the algorithm is actively placing orders in the market. The trading program's EC2 instance should be put on hold after trading hours, leading the indicator to change to "stopped."
As can be seen in Figure 38, the users also can view the Back-Testing results of each respective algorithm researched in this project. The results are shown in the form of a multi-line graph that compares the performance of each algorithm to the DJI 30 index.

A portion specifically designed to present the working notebook in html format has also been included on a separate page as can be seen in Figure 39. Users will be able to evaluate the
algorithm's inner workings in this area, including setup, data preprocessing, feature engineering, model training, and backtesting outcomes. To ensure that consumers thoroughly comprehend how the pipeline works, each stage of the procedure should be explained to them through the attachment of this notebook.

4.5.2. Back-End Development

To complement the presentation of data on the front-end a back-end using Flask, the deployment of scripts on Amazon EC2 was done. The Flask server only serves as a messenger between the brokerage account and the front-end as there is no involvement of any database. The scripts, on the other hand, serve as automated components in EC2 that can automatically collect data and train agents needed for the project. A script to automatically run the live server trading was also made to accommodate the user-controlled button of running the server on the UI dashboard.
5. LIMITATIONS

After having worked on the project for almost a year and facing many challenges throughout the process of developing this project, some key drawbacks and potential limitations will be discussed with detail in the following subsections.

5.1. THE POTENTIAL OF OCCASIONAL UNRELIABLE DATA

As previously discussed in subsection 4.3.5 about the potential of the anomaly in data that leads to extreme agents’ behaviours being an error, this abrupt change in data seems to arise from the data vendor, thus leaving no room for the contributors of this project to control. Although this was the first time such an anomaly was encountered, the potential rendition of this error in the future might lead to extreme action taken by the RL agents. Since there is no guarantee over the reliability of the data provider and FinRL in giving perfect data entry all the time, the only solution that can be offered is to keep the agents as conservative as possible during training and prepare them for extreme cases such as this.

5.2. POTENTIAL BIAS IN DATA AND RESULTS PRESENTATION

As human bias might be involved in presenting good results to the audience, it is imperative to note that results presented might be susceptible to bias by presenting good numbers at the expense of objective evaluation based on methodology. Although it is to be made aware that the contributors of this project tried to hinder any kind of bias from affecting any phase of this project, this tendency might still affect some minor pieces of this project. It is then to be made aware that the contributors of this project tried to provide objective evaluations on each part of this project by constantly examining each other’s works.

5.3. ERROR IN INTERPRETING FINANCIAL DATA OR JARGONS

Given the team members of this project are from computer science background, a considerable amount of time was spent in trying to understand the complex financial jargons and financial terms. The team members leveraged the contents on financial education websites such as corporate finance institute, Investopedia to better understand the financial terminologies better and suitably incorporate relevant concepts into the development process of this project. Apart from that, the team members also read articles found from financial communities such as seekingalpha to understand the issue problematized in section 1 more thoroughly.

5.4. SHIFT TO PANDAS 2.0
As with the recent release of Pandas, some of the codes within the codebase of FinRL libraries were deprecated. The development for this project was therefore still done based on Pandas of version 1.5.3. The codes for this project might need an update in the future should the adoption of Pandas 2.0 is becoming more prevalent and previous versions are no longer supported.
6. **CONCLUSION AND FUTURE PLANS**

To summarize, the average individual equity investor has generated significantly less annualized returns when compared to the compounded S&P500 stock market index over the last decades. This finding suggests that the average investor might not be well-educated enough in trading and that there might be emotional bias influencing their decisions. Therefore, this project aims to aid the average equity investor in generating excess profits as compared to the market returns by developing RL-based algorithmic trading strategies that will be trained on historical data with the mitigation of emotional bias on the American stock market. Furthermore, the completion of the project also aims to provide a UI dashboard that is integrated with a backtesting framework to allow users to interact directly with the model and data.

This report presented the findings on DNN, LSTM, and RL–based algorithmic trading strategies, information regarding FinRL as the primary ecosystem of this project, a methodology on the end-to-end project development starting from data collection up until algorithm development, the results of the algorithm development in the form of working RL agents, a live trading simulation framework using the captioned RL agents, and a UI dashboard made for users to interact with. The results of backtesting on how TD3 and DDPG successfully managed to generate alpha over DJI indicate that these algorithms can potentially be lucrative for live trading in the market and helpful for investors. This argument is further reinforced by the current equity value of the brokerage account which showcases an equity of 103,895.31, implying a positive return over the initial $100,000 account balance through live trading simulations performed.

The limitations faced on this project mostly revolved around the reliability of data, potential biases, and potential misinterpretation of data. The nature of information and data is very integral for this project since the speed at which a trader can convey data accurately is what decides the potential alpha generated by a trade.

Should there be more time in the future on developing this project, the potential on expanding the scope of the project to include the combination of technical and fundamental analysis for day trading seems promising given the slower nature of changes in a company’s fundamental data. Additionally, improving the frequency into the realms of high-frequency trading (HFT) might also be an interesting topic to be researched in the future rendition of this project.
REFERENCES


