Trading platform for comparing configurations of RL-based trading strategies.

Student ID: 20281155

I hereby declare that this dissertation is all my own work, except as indicated in the text.

Supervisor: Dr. Kai Xu

psyrf4@nottingham.ac.uk
University of Nottingham
18th April 2024
Signature: RF

1 Abstract

This project explores the design and development of a long-only, day-trading strategy based on reinforcement learning agents trained using the Maskable PPO [2] [8] algorithm on a custom environment. To pursue this goal, the project explores the development of a high-performance and extensible trading platform that facilitates both historical and live market data acquisition and storage. The functionalities of this platform are extensively used to train the trading agents in the custom environment and later evaluate their profitability through experiments. The results of all experiments are carefully collected and analyzed.

Contents

2 Introduction

The value of assets such as cryptocurrencies and stocks exhibit continuous fluctuations due to market forces. These variations in value are primarily a result of alterations in supply and demand. When more people purchase a particular stock (indicating higher demand) compared to those interested to sell it (indicating lower supply), the price of the stock tends to rise. Conversely, in situations where more individuals wish to sell a stock than there are buyers (higher supply than demand), the stock's price declines. [1].

Investors and traders capitalize on the continuous fluctuation of asset values by buying or selling them with varying frequency, a practice known as trading, with the aim of securing a profit. In the past, these trades were executed by traders physically visiting exchanges and spending hours on the trading floor. However, nowadays, the majority of trades are conducted electronically through advanced trading platforms and automated computer algorithms. In the US stock markets, quantitative investment using algorithmic trading has accounted for over 85% of transactions since 2012 [6]. This transformation has facilitated the execution of numerous trades within fractions of a second, lowered transaction costs, and expanded opportunities for a broader range of market participants, including retail traders.

When formulating trading strategies, traders use a variety of tools and methodologies to analyse data and try to anticipate the market trend of an asset or a portfolio. Some of the most prevalent techniques include technical analysis, sentiment analysis, and more recently, machine learning. Technical analysis involves the evaluation and interpretation of historical price and volume data to identify patterns and trends using technical indicators (such as moving averages). Sentiment analysis, conversely, involves the examination of news sentiment to assess the overall sentiment of investors towards an asset. In particular, over recent years, there have been significant advances in this field due to the adoption Large Language Models (LLMs) that significantly outperform previous techniques when performing sentiment analysis tasks, which ultimately translate in superior economic performance of trading strategies[7]. Finally, machine learning and reinforcement learning have been adopted in conjunction with other techniques to create models capable of accurately forecasting asset price trends and even providing insights into the optimal time to purchase or sell a given asset.

Each of the methodologies present several instances, offering flexibility and customization to suit specific needs. In technical analysis, a wide range of indicators exist, each with unique characteristic, objectives and adjustable parameters that can be tailored to various asset types and market conditions. When it comes to sentiment analysis using LLMs, there is a wide variety of options available, ranging form quantized general-purpose LLMs such as LLAMA 2 and Mistral, which can run on consumer-grade computers, to cutting-edge closed-source models like GPT-3 and GPT-4, accessible only on the cloud. Additionally, machine learning techniques offer a wide spectrum of choices, ranging from reinforcement learning to deep learning, making the development of trading strategies and the feature selection process an extremely complex endeavour that requires iterative experimentation with large amounts of data. Furthermore, once a strategy has been clearly defined, it is crucial to assess its performance via backtesting on historical data prior to adopting it in a live-trading setting. Periodic re-assessments and fine-tuning of the strategies is often required as market trends evolve over time[6].

This project aims to explore the formulation and development of trading strategies using existing research in the areas of reinforcement learning for trading signal generation, technical analysis, and even LLM-based sentiment analysis of financial news. Through the integration of these components, the objective is to develop sophisticated models capable of generating profitable trading signals for equity day-trading.

To reach the objectives of the project it is crucial to first design and develop a framework capable of handling the various stages of trading. This involves the acquisition of large amounts of real-time and historical market data from a broker, its efficient storage, and subsequent distribution for training reinforcement learning models and performing backtesting.

Therefore, the first objective that is tackled as part of this project is the development of an open-source, modular, high-performance, distributed trading platform. This platform is designed to reliably handle large volumes of data from various sources and includes several components that facilitate all the data acquisition/preprocessing steps, including a sentiment-analysis component that makes use of LLMs to extract the sentiment from news headlines.

The development of the trading platform is followed by the investigation and development of trading strategies that make use of reinforcement learning agents trained on historical market data and technical indicators, all provided through a custom-made environment that accurately simulates real day-to-day trading scenarios.

Finally, a series of experiments are designed and executed to assess the profitability of the adopted RL-based agents when trading various equity assets in different scenarios. Throughout both the training and testing phases, features such as technical indicators are incorporated into the environment in which the agent operates. Further experiments are then conducted to examine the profitability impact when introducing news sentiment as a feature in combination with existing indicators. The results are then compared with the buy-and-hold strategy, which is used as a baseline.

3 Related Work

3.1 Challenges of Developing Trading Strategies, a Focus on Factor Selection

During the initial stages of formulating and developing trading strategies, one of the most challenging components of the process is selecting which data and market features to use in order to optimize the strategy's ability to predict market trends and make profitable trading decisions. The process of using financial variables, such as traded volume and company growth measures, to build predictive models is also known as factor investing. Recent global surveys reveal that this technique is employed by 90% of institutional investors to manage part of their portfolios[6].

Currently, the general approach that quantitative traders use involves manually selecting factors by comparing their corresponding portfolio returns in backtesting. This method is mainly a trial-and-error approach that can become extremely challenging due to the large combinatorial factor space and the initial factor selection process, which is often the result of speculation. Furthermore, in typical quantitative investment scenarios, portfolios and features need to be adjusted periodically at the end of trading cycles that can last between a day and several months. This process is crucial given the ever-changing market conditions, but it also presents significant challenges since traders need to identify which assets and factors are linked to the portfolio performance decrease and make appropriate adjustments to allow the model to consistently outperform market returns.

The research of iQuant [6] focuses mainly on the design and implementation of an interactive quantitative investment system that combines the use of algorithmic models for initial feature selection with a manual refinement process aided by interactive interfaces. While the visualisation component of the paper is not strictly related to this project, the steps taken by the authors to produce iQuant yielded a crucial feature selection framework, crafted thanks to the help of extensive feedback from expert traders, that can be, at least in part, integrated into this project to conduct the initial asset and feature selection. The framework presents the following tasks:

1. Selection of an initial pool of stocks and factors
2. Evaluation of factors (using historical data)
3. Factor refinement and portfolio creation
4. Evaluation and comparison of portfolio returns

3.2 Reinforcement Learning-Based Trading Strategies

Given the ample range of factors and their interactions that drive asset prices on the market, developing trading strategies based solely on a discrete set of static conditions, which can consistently outperform market indices, can be extremely challenging. The main difficulties of this endeavor stem from the need for a deep understanding of market conditions and the connection between the factors that influence it, combined with a continuous need for strategy fine-tuning due to changes in market conditions. This becomes particularly challenging when working with strategies that have thousands of discrete and continuous parameters that need to be continuously updated at the end of each trading cycle.

Furthermore, the psychological and emotional aspect of the risk of losing significant amounts of money, paired with the inability to instantly notice and react to market changes every time, can often lead traders to act in greedy and irrational ways, further contributing to the increase of their losses when trading manually. As such, replacing the human trader with a computer program that acts according to predefined logic that is reliable even in the most sensitive scenarios where humans might let their emotional state cloud their judgement, seems to be the most appropriate choice. [5]

Recent years have seen a significant increase in the adoption of reinforcement learning algorithms in algorithmic trading. This surge in popularity is driven by the potential of these algorithms to adapt and learn from market dynamics in real-time, offering a promising avenue for developing more adaptive and responsive trading strategies. Unlike traditional strategies reliant on static conditions[5], reinforcement learning-based approaches allow trading systems to dynamically adjust and optimize their actions based on ongoing market feedback. An open-source example of a trading and finance-oriented reinforcement learning framework is FinRL [9], which despite the significant challenges of simulating market conditions, manages to produce several market environments based on real-world market data.

Furthermore, research conducted using FinRL [9] and deep reinforcement learning, to develop an ensemble strategy that combines multiple actor-critic-based algorithms [4], shows that this approach yields significant results, outperforming each individual algorithm comprising it, the Dow Jones Industrial Average, and the min-variance portfolio allocation method in terms of the Sharpe ratio, a risk-adjusted return measure of an investment.

More recent research further explores the use of deep reinforcement learning by treating stock trading as a Markov decision process represented by states, actions, and rewards in a reinforcement learning algorithm [10]. In the paper, the authors adopt a actor-critic approach such as PPO (Proximal Policy Optimization)[2], an algorithm that has seen great success with games such as Go, Chess, and even Dota 2 [3], and that is seeing strong adoption in the financial sector. The PPO agent is trained on various types of data, including technical indicators, that are first passed through cascaded LSTM networks to extract time-series dependencies between data points in a given window of time. Results of the paper indicate that the proposed model has strong profit-taking ability in the US and Chinese market, further emphasizing the potential of using PPO-based trading agents, especially when combined with ML networks such as LSTM [10].

3.3 The Use of LLMs for Sentiment Analysis on Financial News

One of the methods that traders often adopt in their strategies to aid the decision process when it comes to performing trades is sentiment analysis, a technique that involves understating the sentiment of the market and investors towards a particular asset, typically by analysing news articles, social media posts, and other forms of publicly available information.

Traditional approaches for sentiment analysis generally involve a representation step, where techniques such as "bag of words" are used to represent the text in vector form, and an econometric model step, where the output of the previous step is used to determine a sentiment score of the text provided as input. While this technique has shown acceptable results, the representation step sacrifices a lot of information that is generally conveyed through word ordering and contextual relationship between words, which could prove extremely valuable in the econometric model step. [7].

A modern alternative to the traditional sentiment analysis approaches is the adoption of Large Language Models (LLMs). LLMs are models trained on extensive text datasets encompassing a wide range of sources and topics. Due to their nature, LLMs provide much better text representation than more traditional techniques, making them a more suitable candidate to be integrated in the first step of the pipeline. Empirical results have shown that the heightened accuracy in capturing contextual information through LLMs translates into superior economic performance of trading strategies. [7].

When adopting LLMs for sentiment analysis and making trading decisions based on their generated signals, it’s crucial to consider various factors, including model choice, model configuration, news sources, type of news/post, the asset being traded, and even trading frequency. For instance, in some trading strategies, adopting ChatGPT to analyse the sentiment of news headlines can yield significant returns [14]. However, it may not be as suitable for certain real-time high-frequency trading scenarios where the time required for ChatGPT to produce a response is significantly slower than the time available for making a trading decision, rendering it impractical [7].

4 Project Motivation

The project aims to address significant challenges in quantitative trading by developing innovative solutions and tools. Key components of this initiative include:

4.1 Trading Platform Development

The first step in developing a trading strategy that utilizes reinforcement learning models to generate trading signals involves identifying and acquiring the necessary data. This data is crucial for training the models, conducting back-testing on the resulting strategies, and ultimately running the strategy in a live-trading scenario. This step results to be non-trivial given that, despite the wide range of brokers available, few of them provide both live/historical market data and the ability to perform trades at a reasonable cost.

After extensive research, Alpaca.markets was identified as the most ideal broker to be adopted for acquiring the market and news data required for the project, as well as for performing both paper and live trades. Despite the benefits of using this broker, which include no transaction fees and free historical/live market data, it also presents some drawbacks associated with the free tier it provides. One of the most impactful drawbacks for the project is the one-socket connection restriction, which only allows one live stream of maximum 30 symbols to run at any one time. With this restriction in place, each developed strategy cannot individually subscribe to the live feed of the symbol it operates over. Instead, all the data would need to flow in a single feed from the broker and later be redistributed to the strategies as needed.

Restrictions like these are quite common with most brokers that provide market data and trading capabilities at a low cost, limiting the ability to run and experiment with multiple strategies on live data. Therefore, a highly extensible, open-source trading platform will be implemented, capable of dynamically providing both historical and live market data and redistributing it to all interested strategies. The platform will initially support the Alpaca.markets broker, but its structure will allow for easy extension to support multiple brokers as needed.

The platform stores each data point, facilitating in-depth analysis of market data and sentiment-labeled news headlines. Additionally, the platform will be published as an open-source product, with the intention of facilitating future research by diminishing and even removing some of the complexities of managing large amounts of data, which is often associated with algorithmic trading. This will enable researchers to focus more of their time on research, and the implementation of models and trading strategies.

4.2 Reinforcement Learning-Based Strategies

The project explores the integration of reinforcement learning and deep reinforcement learning algorithms trained on market data, technical analysis indicators, and even LLM-based sentiment. This initiative advances existing research [9, 4] by introducing new environments that are used to train RL agents. Furthermore, the project aims to experiment and assess the usability of the Maskable PPO (Proximal Policy optimization) algorithm in a trading context.

This algorithm is a variant of PPO [2] that uses an action mask to alter the calculation of action probability distribution to prevent the agent from performing invalid actions. This approach shows promising results when compared to other common techniques, such as negative rewarding on invalid action, which often lead the agent to struggle to find positive rewards in the environment [8].

4.3 LLM-Based Sentiment Analysis

LLMs are adopted for this project to perform sentiment analysis tasks on news headlines. Experiments are designed to assess the impact of adopting the sentiment score as feature among other technical indicators. The use of LLMs for sentiment analysis of financial news is still in its infancy, and this project aims to contribute to existing research [7] and advance this field by exploring the use of quantized LLMs such as Llama 2 (4bit), Mistral, and Orca 2 (4bit), which have not yet been explored for sentiment analysis tasks in trading purposes. Quantized LLMs have a smaller memory footprint, allowing them to be run on consumer-grade computers. By assessing the performance of these specific models in the context of financial sentiment analysis, this project aims to provide valuable insights into their suitability for real-world trading applications, potentially broadening the range of accessible tools for algorithmic traders.

4.4 Overall Project Motivation

In summary, the project aims to contribute to quantitative trading research by introducing novel methodologies and tools. Through the development of a sophisticated trading platform, integration of LLM-based sentiment analysis, and exploration of reinforcement learning-based algorithms, it seeks to address existing challenges and facilitate future research in the field.

5 Description of Work

This project aims to design and develop the following components:

- A trading platform with the following capabilities
  - Live and historical data acquisition from a broker
  - Data distribution
  - Data storage
  - Use LLMs to perform sentiment analysis on news headlines collected from the broker
    - Allow users to specify which LLM to use for a particular news headline
- A library that allows for interaction with the trading platform

- A library that allows to develop and utilize data pipelines to preprocess market data before using it to train agents and run backtesting scenarios
- An environment where the agent, trained using the Maskable PPO algorithm, can learn to perform trades in order to maximize profits
- A backtesting environment that mimics closely live-trading environment characteristics
- Experiments pipeline that uses the backtesting environment to assess the performance of agents trained to operate in various trading conditions

5.1 Trading Platform Functional Specifications

To meet the requirements of the project, the trading platform should allow for the following functionalities:

- On start, it should connect to the API of a broker (e.g. Alpaca.markets)
- On demand from user, it should request historical data to the broker and provide it as a response
  - It should be able handle large amounts of data, up to all data points for a given symbol that the broker provides
  - While working to provide large amounts of data, the platform must still be responsive to other requests in parallel (i.e. requests must be non-blocking)
- On demand, users should be able to request that a live feed of market data for a symbol is started/stopped
  - The live data received from the broker should be distributed in separate feeds on a symbol-by-symbol basis
  - The platform should be able to provide information to the users about active symbol feeds
- All data received from a broker should be stored in a local database for fast access
- The platform should provide a unified interface for users to request sentiment labels for a given news headline
  - Users should be able to select the LLM to be used for each sentiment analysis task
- All market news and sentiment analysis data should be stored in a local database
  - Requesting sentiment analysis for the same news with the same LLM as done previously should provide a quick response from the database (i.e. identical sentiment analysis tasks will be performed only once)

5.2 Trading Platform Client Functional Specifications

The trading platform client should be fully integrated with the trading platform and provide the following functionalities:

- Request asynchronously all types of historical data that the platform supports
- Request that the trading platform subscribes to a new symbol feed to receive live data from the broker
- Asynchronously subscribe to the live feed of market data that the trading platform re-distributes and allow for the user to make use of it
- Request that the trading platform performs sentiment analysis of the news for a given symbol and returns them to the client asynchronously

5.3 Market Data Preprocessing and RL Strategy Building Library

Once the data is collected from the trading platform, it should undergo several preprocessing steps before being utilized in a RL environment to train the agent. Both the data preprocessing pipeline components and the RL-based strategy building tools should be provided as a library with the following functionalities:

- Allow the user to define multi-step pipelines that perform transformations on the input data in order to prepare it for training/testing RL trading agents. Some transformations include:
  - Adding technical indicators based on basic market data features
  - Handling any gaps in the data
  - Resampling data to different timeframes
  - Applying sentiment scores as an indicator on stock candle data
- Provide a trading environment that follows the structure described by the Markov decision process to be used to train RL agents on historical market data
- Provide a backtesting environment to assess the performance of the RL trading agents that resulted from training on historical data

5.4 Experiments Pipeline

Given the stochastic nature of the training process of RL agents such as Maskable PPO, the resulting policy can differ significantly between two training sessions on the same data and under the same conditions. As such, to be able to determine the real accuracy of the resulting agents it is required to train and evaluate the model multiple times and consider the ensemble results when comparing with a baseline. Additionally, to gain a better understanding of the extent of the performance of the resulting RL agents, it is important to experiment with different market conditions and environment parameters.

A clearly defined pipeline with appropriate tracking tools should be developed and adopted in order to run the experiments systematically and collect the results. It is imperative that the experiments are defined in such a way that allows for easy reproducibility and extendibility under the same conditions as described in this project.

6 Methodology

To ensure the project meets its requirements and specifications, specific methodologies were carefully selected for each component.

6.1 Trading Platform

To ensure the correct functioning of trading strategies both during model training and backtesting, the trading platform, responsible of providing required data, should meet the following criteria:

- Scalability and reliability
- Performance

6.1.1 A Distributed Modular Trading Platform

Trading signals generated by trading strategies are highly dependent on the input market data that the strategy receives. As such, any gaps or inconsistencies in the data used to make trading decisions can have a significant negative impact on trading outcomes. To ensure that trading strategies perform at their best capacity, the trading platform should always provide reliable data acquisition and trading functionalities.

While ensuring 100% reliability is not always possible due to a wide range of potential issues, there are choices that can be made during the design stages to enhance the trading platform's dependability. One such choice involves designing the trading platform as a modular distributed system. The division into modules is based on functional responsibility, and each module's runtime is separate from the others, ensuring that if one crashes, the others are only marginally affected. The following modules have been identified for the trading platform:

- Data provider: responsible of fetching historical and live data from a broker.
- Data storage: responsible of storing data.
- Sentiment Analyzer: responsible of performing sentiment analysis tasks.

Communication between the various modules is facilitated through a distributed medium that supports the publisher-subscriber paradigm. This allows for interest-based communication, where each module subscribes only to the feeds of the components it needs to interact with (e.g., the Sentiment Analyzer only subscribes to news feeds provided by the Data Provider, while the Data Storage subscribes to all feeds to store all types of data).

The distributed nature of the platform also provides significant scalability benefits (example can be seen in figure 1), as each module can be placed on a separate host, and multiple instances of the same type of module (e.g., data provider) can be launched in parallel to provide load balancing capabilities when handling large amounts of data.

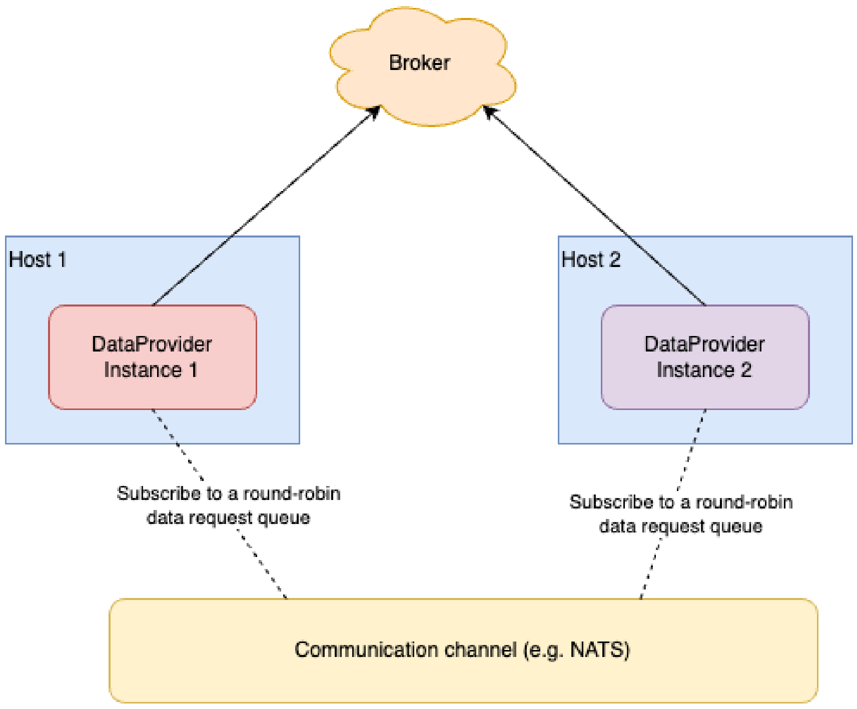

Broker
Host 1
DataProvider Instance 1
Host 2
DataProvider Instance 2
Subscribe to a round-robin data request queue
Subscribe to a round-robin data request queue
Communication channel (e.g. NATS)

Figure 1: Example of how the components of the trading platform can scale and provide load balancing capabilities. The communication channel provides a distributed queue of requests, each instance of the DataProvider pops a request from the queue as soon as it can. Each instance of the DataProvider only responds to the requests it has received.

6.1.2 A Parallel Processing Model

Each of the modules described in the previous section needs to handle multiple user requests and data streams simultaneously without any slowdowns. To address this requirement, the trading platform employs a robust parallel processing model (described in figure 2) for each of its components. This enables processes such as handling data requests and providing live data streams to work in parallel without blocking each other during execution. This approach enhances overall system performance and responsiveness, enabling the platform to scale seamlessly as user demands increase. Careful considerations need to be made during the implementation stage. An appropriate programming language that supports efficient parallelism needs to be chosen, and parallelism needs to be handled carefully to avoid possible deadlocks or race conditions.

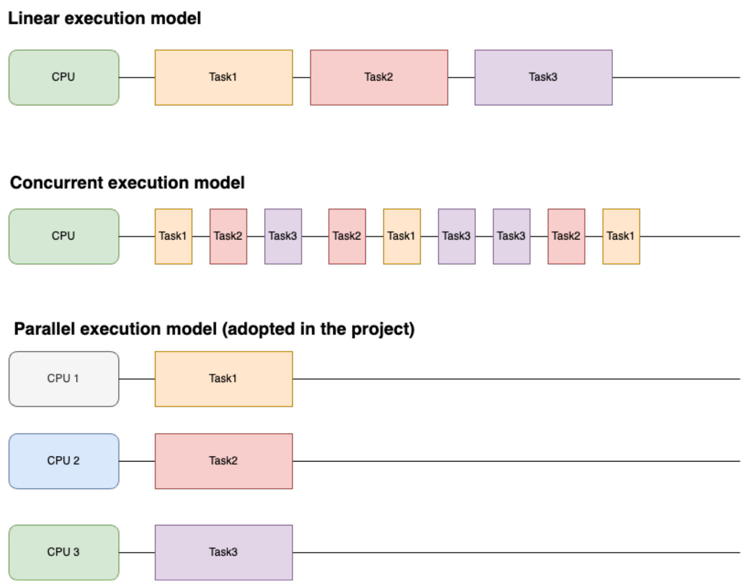


Figure 2: Execution models (linear, concurrent, parallel). Note: this diagram is an over-simplification aimed to provide a general idea on how various execution models work. It might not match the behaviour of the execution models in all scenarios.

6.2 Trading Strategy

Trading strategies can have drastically different characteristics depending on the market conditions, traded asset classes, trading costs and frequency.

This project aims to experiment with RL-based strategies that trade equities within a short period of time at a medium-high frequency. In particular, the goal is to develop strategies that only hold positions during the day and that ensure all positions are closed before market close time, an approach also known as day trading.

There are several benefits and drawbacks to this trading approach compared to other strategies that have more long-term horizons. In particular, while day trading allows for high profits when correctly exploiting short-term market movements, it also exposes traders to higher risk for substantial losses as some market movements can be very unpredictable. Additionally, frequent buying and selling, a defining factor of day trading, can lead to significant commissions and fees which need to be carefully considered before each trade in order to avoid reaching a loss simply due to having the additional costs being greater than the obtained profits. Finally, the high liquidity, often associated with the assets chosen for day trading, allows for easy position entry and exit which can be crucial at minimizing loss potential, but it can also lead to impulsively entering and exiting uncertain positions more easily, especially when performing manual trades.

Both benefits and drawbacks need to be carefully taken under consideration when developing the trading strategies. Extensive market research, considerate asset selection and continuous performance supervision of the adopted strategies can help reduce the risks that are associated with day trading while maximizing the benefits of better resilience to long-term market downtrends and reduced exposure to sudden downturns due to unforeseen overnight events.

6.2.1 Long vs Short Positions

When sending orders to exchanges to open positions, traders generally have the options of buying or selling an asset, which is associated with opening a long or short position respectively.

Opening and closing a long position on an equity involves the following steps:

1. Buying a quantity of a given stock (which opens the position).
2. Partially or fully selling the stock at a loss/profit.

Both buying and selling the assets can incur in fees and commissions. The profit or loss from the two trades is defined by the following formula (without considering commissions):

$$pnl = q \cdot (b - s) \tag{1}$$

- *pnl* represents the Profit and Loss
- *b* represents the buy price
- *q* represents the quantity (e.g. number of stocks)
- *s* represents the sell price

Opening and closing a short position on an equity involves the following steps:

1. Borrowing a given quantity of a stock from a borrowing entity
2. Selling the stock at a given price
3. Buying the owed stock back
4. Returning the borrowed stock to the borrowing entity

In addition to commissions on buying and selling the asset on the market, fees may incur to borrow the assets. The profit or loss from the two trades is defined by the following formula (without considering commissions):

$$pnl = q \cdot (s - b) \tag{2}$$

6.2.2 Long-Only Strategy

The borrowing aspect of short positions introduces additional risks and requirements to the trading process. Unlike buying a stock, where your loss is capped at your investment, short positions have the potential for massive losses associated with the obligation to return the borrowed asset regardless of how much the price has increased compared to the initial sell price. Furthermore, the requirement of margin accounts for short positions and the borrowing interest, make the worth of short positions harder to estimate prior to opening them.

Given the additional complexities and uncertainties associated with managing the risk of short positions, the strategies developed as part of this project are only allowed to open long positions.

6.2.3 Overall Strategy Characteristics

This project focuses on the development and experimentation of RL trading agents for equity trading that act within the following bounds:

- The agent is only allowed to open long positions.
- The agent only create orders during regular day-market-hours (no pre-queued orders for next day open hours).
- The agent must close all positions before market close on the same day as when the position was opened.

6.3 RL Agent

The strategies developed as part of this project are based on reinforcement learning agents trained using Maskable Proximal Policy Optimization (MPPO), an algorithm known for its effectiveness and balance between performance and stability.

6.3.1 Proximal Policy Optimization

PPO is an actor-critic reinforcement learning algorithm used to train a policy network that maps states, or observations of a given environment, to actions for the agent to take with the goal of producing a policy that maximizes the agent's long-term reward. [2]

PPO is an on-policy algorithm, where data collected from the current policy being evaluated is used to update the policy itself. An approach that can be more efficient than off-policy methods such as DQN.

One of the key features of the PPO algorithm is policy clipping. During policy update, the probability ratio between the new and old policies is clipped to a certain range, an approach that helps prevent the policy from changing too drastically in a single update, which can lead to instability.

Finally, PPO incorporates importance sampling that introduces a weight to each sum of rewards in the experience data which is meant to adjust for the difference between the old policy (used to collect data) and the new policy (being optimized). This approach allows PPO to learn from past experiences even if they were collected under a different policy and helps create a *"trust region"* to further constrain policy updates.

6.3.2 Maskable PPO

Maskable PPO builds upon the core principles of the PPO algorithm by integrating a mechanism for handling environments where not all actions are available in every state (invalid actions). [8]

Maskable PPO prevents the agent from performing invalid actions through the adoption of a masking function. During policy execution, the agent interacts with the environment using the policy network, however, before taking an action, MPPO applies the action mask to the predicted action probabilities effectively setting the invalid ones to zero. As the agent interacts with the environment, the algorithm collects state-action-reward sequences in the same way as the original PPO, however the collected actions are guaranteed to be valid due to the masking step. During policy update, MPPO only considers the masked actions when calculating policy loss, helping the policy learn to make good choices only within the available options.

This approach has proven to be quite powerful as it allows the agent to learn only from valid actions, thus making it align with the actual constraints of the environment it is trained on. In the context of the trading strategies developed as part of this project, MPPO provides an alternative to negative rewarding when the agents attempts to perform invalid trading actions, such as buying the same asset two times in a row.

6.3.3 Hyperparameter Tuning

Like many other RL algorithms PPO has a series of hyperparameters that can have significant impact on the policy learning process during training, from determining the learning rate of the policy network, to promoting exploration by preventing early convergence to a local minima.

While PPO is generally considered a more robust algorithm in terms of hyperparameter initialization, and default values provided by most implementations can work on a wide range of problems, performing appropriate hyperparameter search can greatly improve overall results. During the hyperparameter tuning process, ranges for values were chosen according to recommendations from this article [11] and the original PPO paper [2].

6.4 Training and Testing Data

The primary type of data, that is used both for training the Maskable PPO agent and for testing, is historical candle market data with a one-minute time-frame. Candle data generally includes the open, close, high, and low price for a given minute, and in some cases also volume, number of trades and even VWAP (volume weighted average price). In addition to these basic features that are provided by the broker, technical indicators are also added to further enrich the data.

During the initial experimentation with training the RL agents on the absolute values of these features, significant performance loss was seen on the testing data, especially for stocks that had seen drastic scale changes in the testing data compared to the training data. One such example is the Nvidia stock (NVDA). If the agent is trained on data between 2019 and 2022, where the minimum stock price is around 30$ and the maximum is around 270$, and the the agent is tested on data from 2023 to 2024, where the minimum stock price is around 134$ and the maximum is around 815$, the difference in scale between the two datasets greatly impacts the agent performance, even when data is normalized. Additionally, given that most big stocks follow a general upwards trend, in line with the continuously growing economy, the difficulties of training agents on older data and testing on new data can become quite significant.

To overcome these issues, this project only utilizes scale invariant features, where the scale of the features does not change over time as the price of the stock increases or decreases. For instance, instead of using the closing price of an asset, the close price percentage difference compared to the previous minute is used. Instead of using the absolute value of a simple moving average, the difference between the close price and the simple moving average is used instead. Adopting this approach allows the model to focus on learning the price movement and not the absolute value of the price, making the training process more efficient.

7 Design

This section explores the design of the project. It focuses on the trading platform, RL environment design, and experiments design.

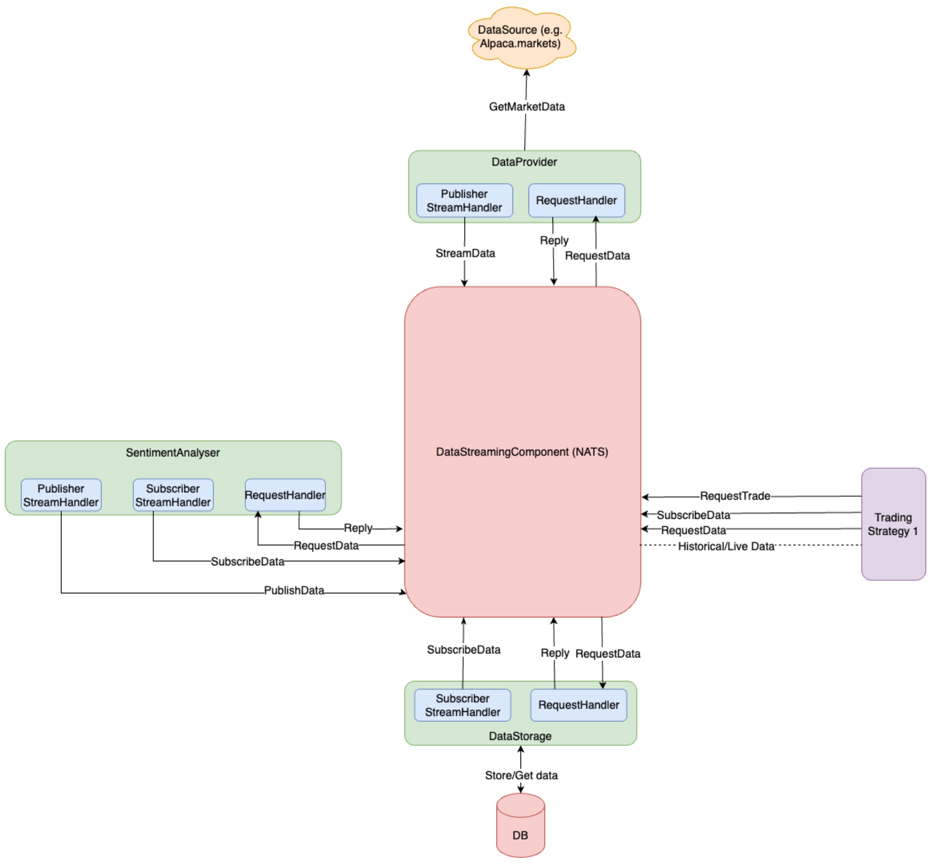
7.1 Trading Platform Design

Ensuring the reliable and prompt reception of both historical and live market data, is crucial for maintaining consistent positive trading performance with trading strategies. As such, the trading platform, responsible providing data, was designed with reliability, scalability, and performance in mind. To ensure minimal downtime and low impact in case of a crash, the platform was designed to be modular and distributed, with each of its modules running as an independent program. This approach enables significant scalability where needed, as each component can be deployed on a separate host and replicated to allow for load balancing and overall better performance.

The trading platform was also designed with the broker in mind. Many online trading brokers impose strict limitations on low-cost/free plans, restricting users' ability to request data, particularly concerning live-data feeds. Platforms like Alpaca.markets, for instance, impose a one-connection-per-account restriction on users' ability to connect to their live data websockets. Therefore, the trading platform was developed to connect to the websocket, request live data for the desired symbols, and redistribute the data from a single feed from the broker into multiple symbol-based feeds. This approach allows strategies to subscribe only to the desired data, effectively circumventing the broker's restrictions.

Requesting large amounts of historical data from the broker can prove to be extremely slow, especially when all historical data for a given symbol is required for model training purposes. Initial experiments show that it takes, on average, around 1 minute and 20 seconds to retrieve all 1.5 million candle records for a symbol such as AAPL. While this might not seem like a long time, the waiting period increases significantly when multiple strategies request the same historical data, slowing down the operation of the trading strategy. To overcome this limitation, the trading platform was designed to incorporate a long-term storage system that saves each record requested from the broker, effectively acting as a cache. Retrieving data from a local database is much faster than obtaining it from the broker, and with this approach, the broker will only be asked for the same piece of data once, while all subsequent identical requests will be fulfilled by the database.

Figure 3: Trading Platform Design



Each component in figure 3 has its distinct responsibility. To ensure as little code duplication as possible, each component utilizes the same communication paradigm based on the following sub-components:

- PublisherStreamHandler: responsible of publishing data.
- SubscriberStreamHandler: responsible of subscribing to data feeds.
- RequestHandler: responsible of handling requests from the users.

The role of each component in figure 3 is briefly described as follows:

- Data Provider
  - Interfaces with the broker to request data.
  - Handles historical data requests from the user.
  - Handles live data streams requested by the user.
- Data Storage
  - Interfaces with the long-term-storage database .

  - Stores live/historical market data, as well as trading signals generated by strategies.
  - Handles data requests from the user and forwards the request to the database.

- Sentiment Analyzer
  - Interfaces with LLM models
  - Handles sentiment analysis requests

- Data Streaming Component
  - Provides communication channels between all other components
  - Does not cache or store any information, its focus is quick data delivery

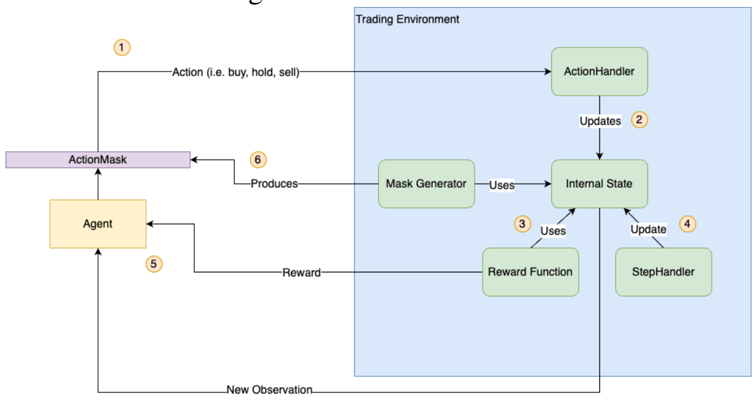
7.2 RL Environment

The Maskable PPO agent, that is adopted as part of this project, is trained in an environment that closely simulates real-world trading scenarios. The environment is designed to only support market data for one symbol, which means that each agent that is trained is specialized on one single symbol. During environment setup, all the historical market data (1-minute candle data and technical indicators) is inserted into the environment which subsequently splits it into trading days. Each trading day represents a trading episode, and once the environment cycles through all timesteps (generally minutes) in a given episode, the environment resets and moves to another episode. To increase learning diversity, the environment does not present the training episodes (days) to the agent in chronological order (i.e. 1st March first, then 2nd March, etc...), instead it randomly picks one from a pool of unseen episodes. Once all episodes have been seen by the agent, training stops.

When created, the environment accepts the following primary parameters:

- Training data.
- The money balance available to the agent during simulated trading (note that this balance resets after each episode). The default is 100_000 USD.
- Trading commission (computed as percentage of traded value that is applied both when buying, and selling equity).
- A time cooldown to delay action from the agent once a position is opened. This is to prevent trading at a frequency that is too high. The default is 2 minutes.

Figure 4: RL Environment



The diagram in Figure 4 describes the individual steps and components of the RL training process and how it relates to the environment. The following steps occur for each timestep (minute) of each episode (trading day):

1. The agent produces an action among buy, hold, sell that is handled by the ActionHandler component of the environment

2. The ActionHandler performs the action by opening/closing the position at the current minute closing price or by holding. This process updates the internal state of the environment.

   - If the action is "buy", the environment starts keeping track of opened position and pnl (Profit and loss). Commissions are applied when the equity is bought.
   - If the action is "sell", the position is closed and the available cash balance is updated accounting for any profit or loss. Commissions are applied when the equity is sold.
   - If the action is "hold" no action is performed regardless of whether a position is opened or not.

3. Based on the current internal state, a reward is generated

4. The environment moves to the next timestep if there is any in the current episode, otherwise it moves to the first timestep of the next episode.

5. A new observation is generated based on the current internal state of the environment. This includes:

   - Position-related state such as a flag that indicates whether the agent is currently holding a position and current PnL
   - Technical indicator and market data values

6. Both reward and the new observation are fed to the agent which will use them to produce a new action.

7. The environment produces a new action mask based on the current internal state of the environment which is used to mask (or disable) invalid actions from the agent.

This process repeats in a continuous loop until all training data has been used to generate the trained policy.

7.2.1 Reward Function

The training process of the MaskablePPO-based Reinforcement Learning agent involves the use of rewards/penalities to "teach" the agent when the performed action had a good/negative result in the long term.

The reward function adopted in this environment has the following characteristics:

- No reward is given when the agent performs a "buy" or "hold" action
- A reward is given when the agent performs a "sell" action that closes an opened position
    - The reward is equivalent to the percentage difference in cash balance between the time before the position was opened and after the position was closed.
    - Example: Balance: 180 USD; Buy 1 share of AAPL @ 170 USD; Wait for n amount of minutes; Sell 1 share of AAPL @ 172 USD; Final balance: 182 USD; Reward: 1.11 (or 1.11%)

7.2.2 Action Masking Logic

The following action masking logic is adopted as part of the environment to prevent the agent from performing invalid actions:

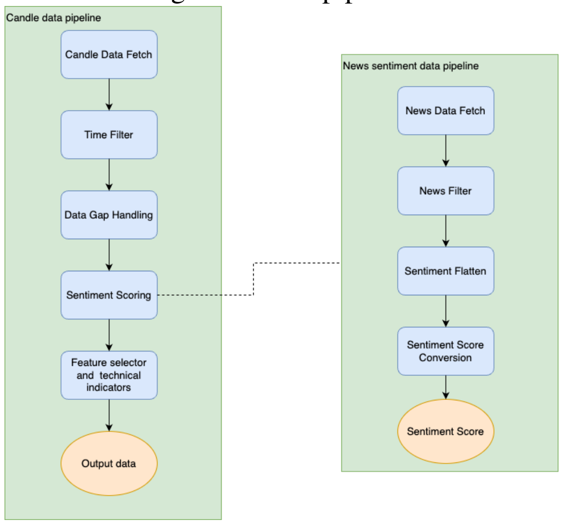
- Scenario: There is an opened position
    - Scenario: Trading day is about to end (5 more minutes till market close)
        - Available actions: Sell
    - Scenario: The position was just opened (less than 2 minutes ago)
        - Available actions: Hold
    - Scenario: Position has been opened for more than 2 minutes and we are not approaching end of day
        - Available actions: Hold, Sell
- Scenario: There no opened position
    - Scenario: Trading day is about to end (5 more minutes till market close)
        - Available actions: Hold (do not buy at the end of day)

  - Scenario: Trading day is not about to end
    * Available actions: Buy, Hold

7.3 Data Pipeline

To increase reproducibility of the experiments performed as part of this project, and ensure that both the training and testing data is pre-processed in the same way, a data pipeline was defined.

Figure 5: Data pipeline



The following steps are included as part of the Candle data pipeline in Figure 5:

- **Candle Data Fetch**: Makes use of the Trading Platform and Trading Platform client to fetch historical market data for a given symbol
- **Time Filter**: Filters the data based on time by removing all data points that are not within US market hours or on US market open days (excludes weekends and holidays).
- **Data Gap Handling**: Fills the gaps in the data using various approaches. Default approach is linear interpolation.
- **Sentiment Scoring**: A separate pipeline that fetches News data with sentiment scores and applies them as a technical indicator to the candle data
- **Feature selector and technical indicators**: This final steps enriches the input market data by adding technical indicators and and processing other features.

The sentiment scoring step in the "Candle data pipeline" is a nested data pipeline with the following components:

- **News Data Fetch**: Fetches News data with sentiment scores from the Trading Platform

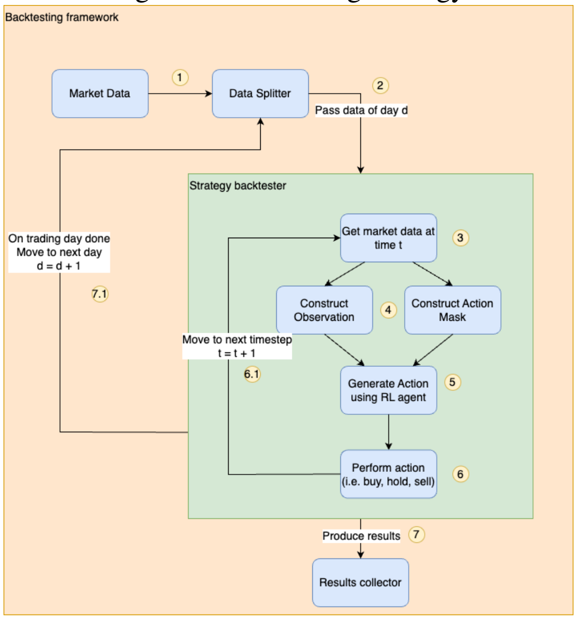
- **News Filter**: Filters news by removing all headlines that are not related to a given symbol
- **Sentiment Flatten**: Each news might have multiple sentiment scores that were generated with different LLM prompts or using completely different LLMs. This process allows to filter sentiment scores from only one LLM and prompt. The end result is that each news headline has one sentiment score.
- **Sentiment Score Conversion**: This process converts the sentiment score from labels such as "positive", "negative" and, "neutral" to respective numeric values 1, -1, and 0.

The pipelines described above should be highly configurable and parameterized allowing to easily modify each step, from the symbol to fetch data for to which parameters to choose for each technical indicator.

7.4 Backtesting Strategy

Once an agent is trained on historical market data, thorough back-testing is required to assess the ability of the policy to make decisions that lead to profitable trades.

Figure 6: Backtesting Strategy



The diagram in Figure 6 shows the various design components of the backtesting framework that encompasses the following steps:

1. Historical market data is split into multiple datasets, where each dataset has the historical data of a specific day.
2. The dataset of a day is passed to the strategy backtester component.

3. The backtester component iterates over each timestep (minute) of the trading day
4. The data at a given timestep t is used to construct the observation and the action mask needed by the policy to produce an action
5. The RL agent produces an action
6. The backtester component performs the action by opening/closing position
   (a) The backtester proceeds to the next timestep $t + 1$ and repeats the process starting from step 3
7. The backtester is done once all timesteps for a given day have been used. Performance metrics such as end-of-day balance are produced and collected.
   (a) A new backtester instance is started with the dataset of the next day $d + 1$. The process repeats from step 2 until all datasets (days) have been used for testing.

It is important to note that, compared to the training process, the cash balance during testing is not reset each time the day ends, instead the cash balance at the end of the previous day is used as the starting balance of the next day. This is done to better simulate a realistic trading scenario, where a lump sum of cash is invested through a strategy over a long period of time and all profits from the strategy are re-invested after each day of trading.

7.5 Experimentation

Given the stochastic nature of the training process of the Maskable PPO-based trading agents, even when training on the same data under the same conditions performance results may vary significantly. As such, to allow for a better estimation of the actual performance of the agents, they are trained and tested multiple times for each symbol. Furthermore, training and testing the agent with different environment conditions, such as with different commission percentages, allows for a better understanding of how the strategy would perform in more complex/harder trading conditions.

To ensure that all experiments are conducted under the same conditions, a well defined experiment framework is defined. Each experiment conducted follows this standardized structure.

Figure 7: Experiments Design

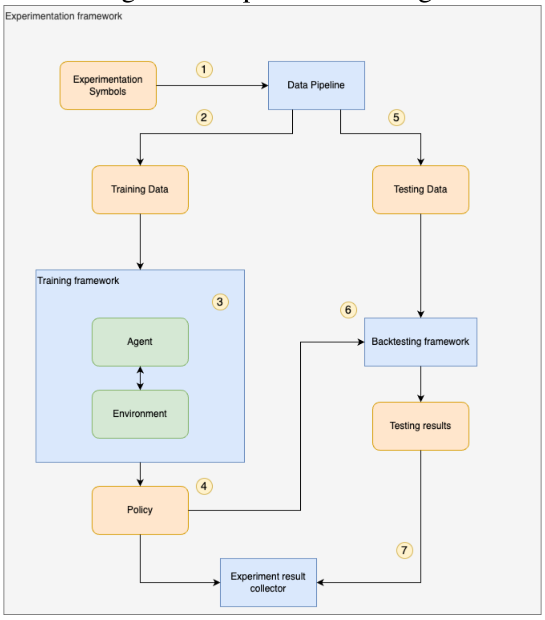


Figure 7 shows the design of the experiments framework which includes the following steps:

1. For each symbol that is used in the current experiment, use the *Data Pipeline* to fetch and pre-process the data.
2. For the current symbol, the training data is selected.
3. Training data is passed to the RL environment to train the Maskable PPO agent.
4. The resulting policy from the training step is saved for later steps.
5. For the current symbol, testing data is selected.
6. Using the policy generated during the training stage, the backtesting framework is adopted to evaluate the performance of the agent.
7. Both the policy and testing results are collected and saved for analysis.

8 Implementation

This section explores the implementation of the project. It focuses on the trading platform, RL environment, Maskable PPO-based agent, and experiments implementation.

8.1 Trading Platform

The programming language chosen for implementing the trading platform is Go, an open-source language developed by Google. Designed with a focus on simplicity, efficiency, and concurrency, Go is statically typed and compiled, delivering the performance of a compiled language while maintaining the simplicity and readability of a dynamically typed one. Go was selected for this project primarily because of its concurrency and parallel programming model based on coroutines (or Goroutines), enabling the efficient distribution of parallel work among multiple CPU threads. This capability is crucial as each component of the platform needs to handle multiple data streams and user requests simultaneously. Additionally, its statically typed nature allows for the development of a more robust platform, that is less prone to crashes due to typing-related bugs introduced during development.

Communication between the various components of the platform is done through the adoption of NATS, a lightweight and high-performance open-source messaging system designed for building scalable and distributed systems. It follows a publish-subscribe (pub/sub) and point-to-point messaging paradigm, providing a simple and efficient way for applications to communicate with each other. While a more modern version of NATS, called JetStream, allows for more robust functionalities that include temporary storage of streaming buffers, the core version is more appropriate for this project given that temporary storage of large data buffers will be handled directly by the individual components of the platform, leaving to NATS only the responsibility of promptly delivering messages.

In the addoption of NATS, communication is segmented based on the data that is being sent through the use of topics, allowing for subscribers to only receive the specific data they need. Examples of topics include:

- dataprovider.command
  - Topic where commands to the Data Provider, such as the ones used to request historical data or to start a live data stream, are sent.
- dataprovider.stream.alpaca.stock.bar.AAPL
  - Topic where live candle data for AAPL (apple) provided by the Data Provider (from alpaca.markets broker) component is streamed.
- dataprovider.data.alpaca.crypto.bar.1min.ETH/USD.3893jfe344jsf.1456629
  - Topic where historical 1 minute candle data for ETH/USD (etherium) provided by the Data Provider (from alpaca.markets broker) component is streamed.
  - The last two components of the topic identify the request ID (3893jfe344jsf) and the number of resulting records (1456629).
  - The user, after making a request, uses the response's request ID to subscribe to the response topic. The number of resulting records is used by the user to know when to stop listening for records (e.g. after all 1.4 million records have been received)

Similar topics are used by each of the components of the trading platform, with slight variations based on individual requirements.

The Sentiment Analyzer component is responsible of processing user requests to perform sentiment analysis tasks on news headlines. This process is made possible by the execution of LLMs locally through tools like Ollama (supported on MacOS and Linux), and GPT4ALL (supported on Windows). The Sentiment Analyzer component communicates with these tools through a RESTful API and sends sentiment analysis prompts which include the headline of the news that is being analyzed. The resulting sentiment, included in the response from these tools, is processed by the component and streamed both to the user who requested it and to the database. Users can select the desired LLM to use for the sentiment analysis process, as long as it is installed locally through either Ollama or GPT4ALL.

Storing news headlines with associated sentiment in the database allows the Sentiment Analyzer component to avoid performing the same analysis task multiple times, which is often slow and computationally expensive. In particular, as long as the sentiment analysis prompt and the LLM are the same, the analysis task is performed only once. All subsequent requests are fulfilled from the value stored in the database, a much faster process.

To store data received from the Data Provider and Sentiment Analyzer, PostgreSQL, an open-source relational database management system (RDBMS), is adopted. PostgreSQL is known for its advanced features and extensibility. Given the well-defined static structure of data received from brokers and generated by strategies, a relational database is the most suitable choice for this project. To streamline communication with the database and mitigate the complexities often associated with working directly with SQL queries, ORM (Object-Relational Mapping) libraries, such as Gorm in Go, are adopted for interacting with the database. The primary purpose of ORMs is to bridge the gap between the object-oriented model used in programming languages and the relational model used in databases.

The Data Storage component of the platform listens to all available topics where data is being streamed over NATS and stores it in the appropriate tables of the PostgreSQL database.

8.2 Trading Platform Client

While the Trading Platform was developed in Go, a language that allows for true parallelism and faster execution due to its compiled nature, all subsequent components of this project, from the RL environment to the evaluation experiments, are implemented in Python.

Python was chosen for the remaining components of the project due to its ample range of machine learning and data science libraries and frameworks. Furthermore, while speed is crucial during the data acquisition and storage stages, the subsequent stages do not have the same performance requirements in this project, making Python the perfect choice.

To allow Python scripts and programs to communicate with the Trading Platform, a Client library was developed. This library interacts with the platform by sending requests through NATS using a pre-defined structure in JSON format, and then handles the incoming data, which comes in Protobuf format, by mapping it to instances of Python classes.

Given that this client can be used in live-trading scenarios too, where providing a prompt response is crucial, both historical and live data feeds should not block the execution of any Python code in the strategy. To ensure that, the Async.io Python library was adopted in the client implementation, which allows for all requests made to the platform to be asynchronous. This is crucial especially when subscribing to live market-data feeds where it is not always known how long is needed to wait before the next data point arrives.

8.3 RL Agent

The Maskable PPO algorithm, used to train trading policies in this project, is implemented as part of the Stable Baselines 3 Contrib Python library [12]. This implementation builds on the Stable Baseline 3 PPO implementation [13] by allowing developers to provide action masks in the form of boolean lists (True for allowed action and False otherwise). This is done by simply implementing an *action_masks* function as part of the RL environment that produces the action mask before each step of the environment is executed.

8.3.1 Hyperparameter Tuning

The Maskable PPO algorithm presents many hyperparameters that need to be carefully tuned in order to improve overall results. For the Hyperparameter tuning process, the Optuna framework was adopted due to its simplicity paired with high parallelism and performance capabilities. In particular, the TPE (Tree-structured Parzen Estimator algorithm) sampler was adopted to sample combinations of parameters. Each hyperparameter tuning trial executed through Optuna trains a policy with the Maskable PPO algorithm on historical market data and then it evaluates the policy on data that was never seen before during training. The parameters that were tuned include network learning rate, number of steps, batch size, entropy coefficient and policy network architecture.

The following values were found to bring the best performance for the problem currently being tackled:

- **learning_rate**: 0.0001
- **n_steps**: 1600 (equivalent to roughly one week of 1-minute market data)

- **batch_size**: 1024
- **fully_connected_network_architecture**: 4 hidden layers of 64 nodes each

8.4 RL Environment

To train the RL trading Agent, a custom environment was developed using the Gymnasium Python library. The environment was designed to only allow long-positions (which requires no borrowing) and day-trading (all positions are closed at the end of day) as described in the earlier design sections. When instantiated, the environment takes the training data as a Pandas Dataframe input, together with the initial cash balance and trading commission percentage.

The environment produces observations in a dictionary format (key-value pairs) and the following features and technical indicators are included:

- 9, 21, and 50 minute exponential moving averages subtracted from the closing price
- 9 minute RSI scaled in the range of -1 and 1
- Bolinger Bands with length 20 and std 2
  - Difference between high bolinger band and low bolinger band
  - Difference between closing price and upper bolinger band
- PSAR indicator subtracted from the closing price
- High, Low, Open, Close percentage difference at time t compared to t-1
- Ratio between current close and open price
- Ratio between current high and low price
- A flag indicating whether a position is currently opened in the environment
- The PnL (profit and loss) of the current position (zero if there is no position)

All observations and rewards produced by the environment during training are normalized using the VecNormalize environment wrapper provided by Stable Baselines 3.

8.5 Backtesting

To evaluate the performance of the RL-based strategies, the backtesting.py library was adopted. This library can be used to develop simulation strategies that closely mimic a realistic trading environment. Each backtesting strategy implements two primary functions, *init*, which is used to define all the market data and technical indicators required by the strategy, and *next*, where all trading-related actions, such as buying and selling, are performed. One of the primary characteristics of the *next* function is that accessing market data from its scope only reveals the datapoints associated with the current timestep *t* and previous timesteps *t-1, t-2, ..., t-n*, while all data related to future timesteps is hidden. This approach is crucial to ensure that no data that would not normally be available in a live-trading environment is used (i.e. data from the future).

To backtest a RL policy, the weights of the network are first loaded in a new instance of the MaskablePPO agent. Next, for each call of the *next* function, an observation, containing all required features extracted from the strategy at time *t*, is generated (all data from the observation is normalized). Next, the action mask is generated. Finally, the *predict* method is called on the agent which produces the next action to execute. Once the action is executed as part of the backtesting strategy, the *next* function is called again for timestep *t+1*.

Once the backtesting strategy finishes execution for all timesteps in the testing dataset, the backtesting.py library automatically produces valuable information and performance metrics such as trades information and final cash balance which can be collected to be analyzed in later stages.

8.6 Experimentation

The experimentation framework design, described in section 7.5, is implemented as a set of Python scripts, where each script contains a single experiment to be executed. This clear-cut separation allows to easily identify and re-run each experiment, a practice that greatly improves reproducibility.

Each experiment script performs only one run, which encompasses both training the policy and running backtesting for each symbol that is provided. If we want to perform multiple runs of the same experiment in order to produce ensemble results from several evaluations, it is sufficient to run the same experiment script multiple times, even in parallel, if computing power allows.
Each experiment is fully integrated with MLFlow, a library that is used to collect both the parameters and the resulting metrics from each experiment. Furthermore, after the execution of each experiment, additional artifacts, such as performed trades information, policy network weights, and day-by-day trading performance breakdowns are also saved for further analysis.

9 Evaluation

In this section we explore how the RL-based strategy, described in the previous section, is evaluated and how it performs compared to the buy and hold strategy.

9.1 General Evaluation Conditions

All experiments conducted to evaluate the profitability of the RL-based strategies are based on the design and implementation of the experimentation sections (7.5, 8.6). Each experiment is carefully designed to evaluate specific trading conditions and environment parameters for multiple symbols. This is done to gain a better understanding of the usability of the RL-based strategy under different market conditions, given that each equity asset follows its own trend.

For each equity symbol that is considered for evaluation as part of an experiment, a policy is trained using the Maskable PPO algorithm and the environment described in section 7.2. Furthermore, the profitability of the policy is assessed through the backtesting approach described in section 8.5. This process is be repeated for a total of four times for each symbol and each experiment, and the average of the profit metrics is used. This is done to increase the statistical significance of the results and to ensure consistent performance evaluation.

During the training process of the trading policy using the Maskable PPO algorithm, the same hyperparameters are used across all runs of each experiment. The exact values chosen and how they were chosen are described in section 8.3.1.

When commission rates are set to non-zero values, those values indicate the percentage of traded value. For instance, if a "buy" occurs of 100 shares at 100 USD each for a total value of 10000 USD, and a 0.01% commission is used, the commission paid for that trade would be 1 USD. Commissions apply both on "buy" and "sell" orders.

9.2 Evaluation on SNP 500

The first experiment focuses on assessing the performance of the proposed RL-based strategy on the top 10 constituents of the SNP 500 index by weight using 4 different commission rates for trades. As of 17th April 2024 these are: MSFT, AAPL, NVDA, AMZN, META, GOOGL, BRK.B, LLY, AVGO, JPM. The commission rates that are used for these experiments are 0%, 0.01%, 0.02%, and 0.03%. The data used for training each policy is comprised of historical data from 2016-01-01 to 2023-01-14, while the data used for backtesting ranges from 2023-01-15 to 2024-02-15. All agents start with an initial cash balance of 100_000 USD.

An initial performance comparison of the strategy, on the ten symbols with the four different commission rates (see Appendix Figure 1 and 2), shows that in a 0% commission scenario the strategy brings significant profits ranging from 29.9% profit on JPM to 308% on AVGO. It is crucial to note that the profitability of the strategy does not rely just on the ability of the policy to take the appropriate actions in any given scenario, but also the price trend of the stock. In general, stocks that see a big increase in price over the testing period seem to allow the RL-based strategy to produce higher profits. As the commission rate increases, we can clearly see a significant decrease in profitability with the RL-strategy with AMZN seeing the lowest performance at -7.9% profit with a 0.03% commission rate and AVGO at +65.2% profit with a 0.01% commission rate.

It is important to note that, despite having some symbols where the RL-strategy causes an overall loss, when the ten symbols are taken as a portfolio where each of them receives an equal investment, the average profit for the three non-zero commission rates is always positive. Results show a 5.6% portfolio profit with 0.03% commission rate, 10.5% with 0.02% commission rate, and 12.7% with 0.01% commission rate. The profits of the winning symbols significantly outweigh the losses of the losing ones.

When these results are compared with the profitability of the buy and hold strategy with different commission rates (see Appendix Figure 3 and 4), a distinct pattern can be seen. The different commission rates do not have any significant impact on the profitability of the buy and hold approach. This difference is expected, given that while the buy and hold strategy only involves two orders (a "buy" and a "sell"), the proposed RL-strategy performs multiple trades each day accumulating significantly more total commission. Overall, this shows that the suggested approach using the RL-based policies work best in zero or very low commission trading scenarios.

When the RL-based strategy is compared with the buy and hold at different commission rates (see Appendix Figure 5, 6, 7, and 8), the proposed strategy is able to outperform the buy and hold approach only with a 0% commission approach for 5 out of the 10 considered symbols (AAPL, AVGO, BRK.B, GOOGL, and JPM). This suggests that careful consideration needs to be put into which symbols are picked to be used with the RL-based strategy. With all the other commission rates, the buy and hold approach outperforms the RL-based strategy.

Finally, when analysing the percentage win-rate (number of trades that made a profit) for each symbol at different commission rates (see Appendix Figure 9 and 10), an interesting pattern emerges. For all ten symbols the win-rate increases with higher commission rates, indicating that the policy is capable, at least to some extent, to understand that with higher costs of trading, each trade needs to have a higher probability of bringing a profit, thus reducing the number of risky actions taken. This is an invaluable information that can lead to interesting research into how the RL training environment can be further improved to have a better control over the risk that the resulting policy is willing to take with each trade.

9.3 Evaluation on SNP 500 in a Downtrend Period

This experiment was conducted under the same conditions as the previous one (section 9.2) with a slight variation on the training and testing data that was used to train and evaluate the policy. To train the various policies on each symbol, historical data from 2016-01-01 to 2022-

01-14 was used, while data from 2022-01-15 to 2023-01-15 was used for backtesting. This was done to evaluate the profitability of the RL-based strategy during a period, 2022-2023 in this case, when the SNP500 saw a decline compared to the previous year. All agents start with an initial cash balance of 100_000 USD.

Initial performance comparison of the RL-based strategy with three different commission rates (see Appendix Figure 11 and 12) shows outstanding performance for the 0% commission rate experiment. In particular, GOOGLE is the most profitable at 623.2% profit and JPM is the least profitable at 36.4%. When considering the other commission rates, the lowest profitability is reached by NVDA at -19.7% loss with 0.01% commission, and the highest is GOOGL at +31.5% profit with 0.01% commission. As in the previous experiment, commissions have a powerful negative impact on the overall profits. In spite of that, if the 10 symbols are taken as a portfolio, where the same budget is allocated to each symbol, an overall profit can still be reached at 8.6% profit with the 0.01% commission rate and a smaller 0.3% with the 0.03% commission rate.

When these results are compared with the profitability of the buy and hold strategy with different commission rates (see Appendix Figure 15, 16, and 17), an interesting result can be observed. In the 0% commission rate trading scenario, the proposed RL-based strategy outperforms the buy and hold strategy for each of the ten symbols considered. This is a strong indicator that the proposed strategy is overall more robust and that it is capable of making a profit even when the market is not performing as well. Furthermore, the results suggest that the long-term adoption of this strategy is a strong method to be more resilient in downtrend markets at the cost of possibly not having profits as big as the ones generated by the buy and hold approach during uptrend markets.

As the commission rate increases to 0.01% and 0.03%, a similar pattern to the one seen in the previous experiment can be seen where the the buy and hold approach greatly outperforms the proposed RL-strategy. One minor difference is that, with the 0.01% commission rate, the strategy is still capable of outperforming the buy and hold approach for 3 out of 10 symbols, namely AMZN, GOOGL, and META.

Finally, when analyzing the results related to the win-rate (see Appendix Figure 18 and 19), a similar pattern compared to the previous experiment emerges, where a higher commission rate leads to increased win-rate among most symbols (with the exception of BRK.B in this case).

9.4 Evaluation on Russell 2000

This experiment focuses on assessing the performance of the proposed RL-based strategy on the top 9 constituents of the Russle 2000 index by weight using 3 different commission rates for trades. As of 17th April 2024 these are: BRBR, CYTK, ELF, LNW, ONTO, RMBS, SMCI, SSD, UFPI. The commission rates that will be used for this experiments are 0%, 0.01%, and 0.03%. The data used for training each policy is comprised of historical data from 2016-01-01 to 2023-01-14, while the data used for backtesting ranges from 2023-01-15 to 2024-02-15. All agents start with an initial cash balance of 100_000 USD.

An initial performance comparison of the strategy, on the nine symbols with the three different commission rates (see Appendix Figure 20 and 21), shows that in a 0% commission scenario the strategy brings exceptional profits ranging from 12.8% profit on BRBR, to 9814.6% on ONTO. As the commission rate increases, we can clearly see a significant decrease in profitability with the RL-strategy with BRBR seeing the lowest performance at -26.4% loss with 0.01% commission rate and ONTO at +3029.15% profit with 0.01% commission rate. These results present significant differences compared to the initial experiment on the ten SNP 500 stocks, especially when non-zero commissions are applied. These differences are further accentuated when considering the 9 stocks in the context of a equal-weight portfolio where each symbol receives the same initial cash balance for trading. In this scenario, the average profit of the portfolio is 792.78% when the commission rate is 0.01%, and 58.4% when the commission rate is 0.03%.

When the RL-based strategy is compared with the buy and hold at different commission rates (see Appendix Figure 24, 25, and 26), the proposed strategy is able to outperform the buy and hold approach only with a 0% commission approach for 9 out of the 10 considered symbols (CYTK, ELF, LNW, ONTO, RMBS, SMCI, SSD, and UFPI). Furthermore, with a 0.01% commission, 3 out of 10 symbols (ONTO, SSD, UFPI) outperform the buy and hold approach, while only 2 out of 10 symbols (ONTO and SSD) outperform it when a 0.03% commission rate is adopted. These results are significantly better compared to the ones produced through the evaluation on the SNP500 stocks, further accentuating the importance of thorough experimentation and market analysis when selecting the assets to trade.

Finally, when analysing the percentage win-rate (number of trades that made a profit) for each symbol at different commission rates (see Appendix Figure 27 and 28), the same pattern that was seen in previous experiments emerges where higher commission rates lead the policy to produce higher win-rates.

9.5 A Short Note on Sentiment Analysis

Finally, a smaller-scale experiment was conducted to evaluate the importance of the sentiment score feature, obtained using LLM-based sentiment analysis, on the MSFT stock. For this experiment, historical data between 2016-01-01 and 2023-01-14 was used to train the policies, while data between 2023-01-15 and 2024-02-15 was used for backtesting. The LLM used to generate the sentiment labels among positive, negative, and neutral, was Orca2.

The news headlines used for sentiment analysis come from the Benzinga website, and they were provided through the Trading Platform developed as part of this project. The prompt used for the sentiment analysis task is the following: *"You are a markets expert. Analyze the sentiment of this financial news related to the given symbol and respond with one of the following words about the sentiment [positive, negative, neutral]. Respond with only one word."*.

Two primary scenarios where explored, one where the sentiment score was included among the features and one where it was not. Both the training of the policy and the evaluation step were performed for a total of 40 times for each scenario, to ensure that the results had higher statistical significance, given the stochastic nature of the training process.

Results from this experiment show little to no difference in profitability between having and not having the sentiment score as a feature. When sentiment is present, the average percent profit is 10.17%, which is slightly lower compared to the 10.68% profit when the sentiment score is not included.

Furthermore, this experiment was repeated again with the introduction of a sentiment decay rate of 40%, where the positive or negative sentiment (1 or -1 respectively) would decay towards 0 by 40% after each minute that passed since the news was released. With this approach, a bigger profit gap can be observed between having the sentiment score as a feature and not having it. Results show that, with the sentiment score feature, the average profit is only 8.52%, a significantly lower value compared to the 10.68% that was previously seen when the sentiment score is not included.

Overall, with the RL training environment developed as part of this project, the integration of the sentiment score does not seem to bring any profitability increase for the selected symbol. Further research is needed with more data and symbols to gain a better understanding of whether this is true in a wider range of scenarios, as well as how the training environment can be modified for the policy to learn to better take advantage of this additional feature.

10 Project Management

For both the research and development stages of the project, a Kanban-style Agile process was adopted to facilitate planning and progress tracking.

Starting with an initial list of high-level tasks estimated to be completed within a specific timeframe, the tasks for the following weeks are subdivided into smaller, manageable units of work. This Agile approach offers flexibility in task organization, particularly when task sizes may differ from initial expectations due to unforeseen complexity changes.

Initially, there was be a big focus on researching the best approaches and technologies for the implementation of the platform, followed by the implementation stage. Once the core platform was implemented, the focus shifted to experimenting with the development of trading strategies. One of the first goals that were tackled, was learning about and identifying the best approach to implement and experiment with RL-based trading strategies. This research stage resulted in the development of a custom RL trading environment, that was later used to train the trading policies using the Maskable PPO algorithm. Finally, evaluation experiments were designed and executed to assess the performance of the trained policies in different scenarios.

At the beginning of each week a meeting is scheduled with the supervisor to discuss the progress of the work done in the previous week and any issues that might have risen in the mean time.

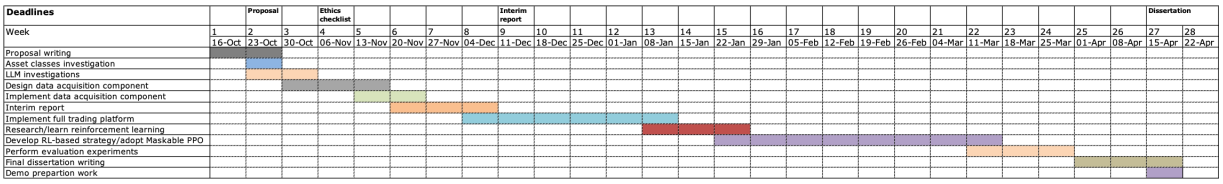
The following macro tasks were tackled as part of the project:

1. Proposal writing (Completed)
2. Asset classes investigation (Completed)
    - Investigate which asset class is best to start with between crypto and stocks
    - Investigate commonly used trading strategies for each asset class
    - Identify best trading platform and data sources
3. LLM investigation(Completed)
    - For each language model adopted identify what is the best way to integrate it into the trading platform (consider cost implications)
4. Design the data acquisition component of the platform (Completed)
    - Identify most appropriate data sources and technologies to use
    - Define high-level and detailed structure of the data acquisition component for both news and asset prices
5. Interim report (Completed)
6. Implementation of data acquisition component (Completed)
7. Full implementation of Trading Platform (Completed)
8. Research and learn about Reinforcement Learning environments and agents (Completed)
9. Develop a custom long-only day-trading RL environment used for the policy training (Completed)

10. Learn about the Maskable PPO algorithm for policy training and adopt it (Completed)
11. Perform evaluation experiments and collect profitability metrics (Completed)
12. Final dissertation writing (Completed)
13. Demo video preparation (Completed)

Compared to the list of tasks provided in the interim report, several changes were made to ensure that the goal of developing and experimenting with reinforcement learning to train profitable trading policies would be reached. As part of these changes, experimentation with simpler non-RL-based strategies and performing paper-trades through the Alpaca platform were replaced with tasks that were more targeted on the RL-based strategies. While this change lead to a slight change in the project trajectory by removing the real-time paper-trading component, it also allowed more time to focus on finding better trading approaches and performing thorough evaluation experiments.

The final version of the Gantt chart is presented here:



Deadlines		Proposal		Ethics checklist					Interim report					
Week	1 16-Oct	2 23-Oct	3 30-Oct	4 06-Nov	5 13-Nov	6 20-Nov	7 27-Nov	8 04-Dec	9 11-Dec	10 18-Dec	11 25-Dec	12 01-Jan	13 08-Jan	14 15-Jan
Proposal writing	■	■												
Asset classes investigation		■												
LLM investigations		■	■											
Design data acquisition component			■	■	■									
Implement data acquisition component					■	■								
Interim report						■	■	■						
Implement full trading platform								■	■	■	■	■	■	
Research/learn reinforcement learning													■	■
Develop RL-based strategy/adopt Maskable PPO														
Perform evaluation experiments														
Final dissertation writing														
Demo prepartion work														

Deadlines													Dissertation	
Week	15 22-Jan	16 29-Jan	17 05-Feb	18 12-Feb	19 19-Feb	20 26-Feb	21 04-Mar	22 11-Mar	23 18-Mar	24 25-Mar	25 01-Apr	26 08-Apr	27 15-Apr	28 22-Apr
Proposal writing														
Asset classes investigation														
LLM investigations														
Design data acquisition component														
Implement data acquisition component														
Interim report														
Implement full trading platform														
Research/learn reinforcement learning	■													
Develop RL-based strategy/adopt Maskable PPO	■	■	■	■	■	■	■	■						
Perform evaluation experiments								■	■	■				
Final dissertation writing											■	■	■	
Demo prepartion work													■	

Figure 8: Gantt chart

11 Contributions and Reflections

Overall, the project progressed well throughout the year, and I have successfully managed to stay on top of my tasks, implementing high-quality software solutions for the development of the Trading Platform, reinforcement learning environment, and evaluation utilities. The initial planning, researching, and designing stages have proven to be extremely valuable as the implementation stage began, providing a clear understanding of what needed to be done to achieve the goals of the project. Taking fewer credits in the first semester has allowed me to concentrate more of my time on the more challenging software development tasks of the project, such as the development of the trading platform, which was completed by the beginning of the second semester. Despite taking more credits in the second semester, the tasks to research, explore, and develop the trading strategy using reinforcement learning agents were completed successfully, primarily through careful planning and prioritization.

Throughout the first semester, I have deepened my understanding of Large Language Models (LLMs) and Algorithmic Trading, gaining insights into the software requirements necessary for the development of trading strategies. Moreover, working on the Trading Platform has improved my skills in software design and development. Specifically, I have enhanced my knowledge of concurrency and parallelism models, distributed computing, large-scale data storage, and, overall, I have strengthened my proficiency in Go programming. I firmly believe that the thoughtful and creative design choices made have had a big impact when the platform was adopted for the development of the trading strategy. Additionally, these choices will benefit other researchers seeking to explore trading-related topics, facilitating their work on data acquisition and management due to the capabilities of this platform.

During the second semester, as my focus shifted on the development of the trading strategy, I deepened my knowledge in machine learning, reinforcement learning, the importance of feature selection, data preprocessing, and hyperparameter tuning. All this newly acquired knowledge allowed me to produce a strategy that is profitable and that, when adopted on appropriate symbols, can significantly outperform other approaches such as the *Buy and Hold* strategy. Furthermore, this project provides new exploration paths for research to be conducted in order to further improve the profitability of the methodology adopted as part of this project. The following extensions of the project could be pursued:

- The reinforcement learning environment could be extended to support taking short positions. The policies trained on this updated environment could be evaluated to assess profitability compared with the current environment.
- Research could be conducted to find better ways to integrate sentiment analysis as part of the strategy, possibly through the adoption of different prompts or different sentiment scoring systems.
- Using a hierarchical reinforcement learning approach, adopt the strategy presented in this project in the context of a portfolio-based reinforcement learning environment that allocates different amounts of cash to each traded symbol.
- Research and develop an automated stock screening mechanism capable of suggesting the best symbols to trade using the strategy presented in this project.

Software products, such as the trading platform and trading strategy, produced as part of this project, are considered intellectual property. Given that I am not employed in any way by the University and that this project is not sponsored in any way through grants that imply that the grantor owns the IP, I am the owner of the IP I have produced as part of this project. The Trading Platform and all the source code related to the trading strategy is distributed as an open-source product under the Apache-2 license to facilitate future research in the field of algorithmic trading.

The project does not involve any human participants, and thus, none of the ethical requirements involving participants apply. Additionally, the work is not affected by any other legislation.

When considering the broader ethical and social aspects of the work, it is clear that both traders and researchers interested in further advancing the field of algorithmic trading, and LLM-based sentiment analysis, would benefit both from the software products and the research results of this project. No unintended consequences, such as military or criminal applications, of the results of this project have been identified for the project.

The overall intention is to make this work accessible to all. Providing more investment tools, such as the ones described in this project, can be beneficial to aid, at least partially, people with limited financial knowledge to learn more about the world of trading and to become conscientious market participants. It is crucial to highlight that none of the components of this dissertation project constitute financial advice. Trading can be a risky activity, and it is known that the trading approach described in this dissertation is generally considered to have higher risks, due to the sheer trading frequency, compared to other investment strategies such as *Buy and Hold*. Nevertheless, this dissertation can be used as a starting point to learn more about developing trading strategies using modern technology.

I strongly believe that through open projects that provide free-to-use open-source financial/investment tools such as the Trading Platform developed in this project, we can democratize access to financial markets and empower individuals to make informed investment decisions regardless of their background or financial means, which can promote inclusive economic growth and contribute, at least in part, to reduce financial inequalities.

References

[1] Carl Hopman. “Do supply and demand drive stock prices?” In: *Quantitative Finance* 7.1 (2007), pp. 37–53. DOI: 10.1080/14697680600987216.

[2] John Schulman et al. *Proximal Policy Optimization Algorithms*. arXiv:1707.06347 [cs]. Aug. 2017. DOI: 10.48550/arXiv.1707.06347. URL: http://arxiv.org/abs/1707.06347 (visited on 04/10/2024).

[3] OpenAI et al. *Dota 2 with Large Scale Deep Reinforcement Learning*. arXiv:1912.06680 [cs, stat]. Dec. 2019. DOI: 10.48550/arXiv.1912.06680. URL: http://arxiv.org/abs/1912.06680 (visited on 04/10/2024).

[4] Hongyang Yang et al. *Deep Reinforcement Learning for Automated Stock Trading: An Ensemble Strategy*. en. SSRN Scholarly Paper. Rochester, NY, Sept. 2020. DOI: 10.2139/ssrn.3690996. URL: https://papers.ssrn.com/abstract=3690996.

[5] Jagdish Bhagwan Chakole et al. “A Q-learning agent for automated trading in equity stock markets”. In: *Expert Systems with Applications* 163 (Jan. 2021), p. 113761. ISSN: 0957-4174. DOI: 10.1016/j.eswa.2020.113761. URL: https://www.sciencedirect.com/science/article/pii/S0957417420305856 (visited on 04/10/2024).

[6] Xuanwu Yue et al. “iQUANT: Interactive Quantitative Investment Using Sparse Regression Factors”. en. In: *Computer Graphics Forum* 40.3 (2021), pp. 189–200. ISSN: 1467-8659. DOI: 10.1111/cgf.14299. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.14299.

[7] Yifei Chen, Bryan T. Kelly, and Dacheng Xiu. “Expected Returns and Large Language Models”. In: (2022).

[8] Shengyi Huang and Santiago Ontañón. “A Closer Look at Invalid Action Masking in Policy Gradient Algorithms”. In: *The International FLAIRS Conference Proceedings* 35 (May 2022). arXiv:2006.14171 [cs, stat]. ISSN: 2334-0762. DOI: 10.32473/flairs.v35i.130584. URL: http://arxiv.org/abs/2006.14171 (visited on 04/11/2024).

[9] Xiao-Yang Liu et al. *FinRL-Meta: Market Environments and Benchmarks for Data-Driven Financial Reinforcement Learning*. arXiv:2211.03107 [q-fin]. Nov. 2022. DOI: 10.48550/arXiv.2211.03107. URL: http://arxiv.org/abs/2211.03107.

[10] Jie Zou et al. “A novel Deep Reinforcement Learning based automated stock trading system using cascaded LSTM networks”. In: *Expert Systems with Applications* 242 (May 2024), p. 122801. ISSN: 0957-4174. DOI: 10.1016/j.eswa.2023.122801. URL: https://www.sciencedirect.com/science/article/pii/S0957417423033031 (visited on 04/10/2024).

[11] *PPO Hyperparameters and Ranges*. `https://medium.com/aureliantactics/ppo-hyperparameters-and-ranges-6fc2d29bccbe`.

[12] *SB3 contrib Maskable PPO*. `https://sb3-contrib.readthedocs.io/en/master/modules/ppo_mask.html`.

[13] *SB3 PPO*. `https://stable-baselines3.readthedocs.io/en/master/modules/ppo.html`.

[14] Alejandro Lopez-Lira and Yuehua Tang. "Can ChatGPT Forecast Stock Price Movements? Return Predictability and Large Language Models". In: (April 6, 2023).