UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

School of Informatics

On the exploration of Deep Reinforcement Learning in Stock Trading through the framework FinRL

End of Degree Project

Bachelor's Degree in Informatics Engineering

AUTHOR: Marugán Rubio, Carolina Alba

Tutor: Onaindia de la Rivaherrera, Eva

Cotutor: Aso Mollar, Ángel

ACADEMIC YEAR: 2023/2024

Resum

Prendre decisions financeres informades requereix temps i coneixement. Recursos que no tothom té. A més, les persones, les famílies i les empreses són unitats de despesa que necessiten mantenir l'estabilitat financera per superar els temps d'emergència. Això vol dir que part de la població pot necessitar recursos monetaris addicionals però no té la capacitat d'obtenir-los. Aquest treball ofereix una solució a aquest problema: una eina per assessorar les unitats de despesa sobre com operar a curt termini en el mercat financer (*trading*). El paradigma proposat per fer-ho és l'aprenentatge de reforç profund. En primer lloc, es realitza una anàlisi de diferents algorismes i modelització d'aprenentatge per reforç. Això es fa per tal d'identificar si existeix una diferència significativa en l'ús d'uns indicadors financers o d'altres, i si l'agent té un millor rendiment utilitzant més o menys indicadors financers. A partir dels resultats de l'anàlisi, s'escull la millor combinació d'indicadors financers per construir un sistema que incorpori les aportacions de tres algorismes diferents, oferint una solució que sintetitza els resultats de tots ells. Aquest projecte proposa una aplicació que simula un robot d'aprenentatge de reforç que pren com a entrada la quantitat de diners que té l'usuari i està disponible per negociar, el nombre d'accions que ja posseeix, els preus de les accions al mercat financer i els indicadors financers. Aleshores, el robot emet les operacions recomanades que donen a l'usuari la millor rendibilitat. Aquesta aplicació es pot personalitzar segons les circumstàncies de l'usuari. La biblioteca *FinRL* dóna suport al desenvolupament d'aquesta solució.

Paraules clau: Aprenentatge de reforç, *FinRL*, agent, entorn, finances, estoc, indicadors tècnics, algorisme

---

Resumen

Tomar decisiones financieras informadas requiere tiempo y conocimiento. Recursos que no todos poseen. Además, los individuos, las familias y las empresas son unidades de gasto que necesitan mantener la estabilidad financiera para superar tiempos de emergencia. Esto significa que parte de la población podría necesitar recursos monetarios adicionales pero no tiene la capacidad de obtenerlos. Este trabajo ofrece una solución a este problema: una herramienta para asesorar a las unidades de gasto sobre cómo operar a corto plazo en el mercado financiero (*trading*). El paradigma propuesto para hacerlo es el aprendizaje por refuerzo profundo. En primer lugar, se realiza un análisis de diferentes algoritmos y modelados de aprendizaje por refuerzo. Esto se hace con el fin de identificar si existe una diferencia significativa en el uso de unos indicadores financieros u otros, y si el agente se desempeña mejor usando más o menos indicadores financieros. Con base en los resultados del análisis, se elige la mejor combinación de indicadores financieros para construir un sistema que incorpora las contribuciones de tres algoritmos diferentes, ofreciendo una solución que sintetiza los resultados de todos ellos. Este proyecto propone una aplicación que simula un robot de aprendizaje por refuerzo que toma como entrada la cantidad de dinero que posee el usuario y está disponible para negociar, la cantidad de acciones que ya posee, los precios de las acciones en el mercado financiero e indicadores financieros. Luego, el robot genera las operaciones recomendadas que brindan al usuario la mejor rentabilidad. Esta aplicación se puede personalizar según las circunstancias del usuario. La biblioteca *FinRL* respalda el desarrollo de esta solución.

Palabras clave: Aprendizaje por refuerzo, *FinRL*, agente, entorno, finanzas, acciones, indicadores técnicos, algoritmo

---

Abstract

Making informed financial decisions takes time and knowledge. Resources that not everyone possesses. In addition, individuals, families, and firms are expense units that need to maintain financial stability to overcome emergency times. This means that part of the population might need additional monetary resources but does not have the ability to obtain them. This work offers a solution for this problem: a tool to advise expense units on how to trade. The proposed paradigm to do so is Deep Reinforcement Learning. First, an analysis of different Reinforcement Learning algorithms and modeling is carried out. This is done in order to identify whether it exists a significant difference in using some financial indicators or others, and if the Agent performs better using more or less financial indicators. Based on the analysis results, the best combination of financial indicators is chosen to build a system that incorporates the contributions of three different algorithms, offering a solution that synthesizes the outputs of all of them. This project proposes an app that simulates a Reinforcement Learning robot that takes as input the amount of money the user owns and is available for trading, the number of shares they already own, the stock prices in the financial market, and financial indicators. The robot then outputs the recommended trades that give the user the best profitability. This app is customizable to the user circumstances. The FinRL library supports the development of this solution.

Key words: Reinforcement Learning, FinRL, Agent, Environment, Finance, stock, technical indicators, algorithm

Contents

List of Figures

List of Tables

CHAPTER 1
Introduction

Stock trading is a financial resource to get profitability from monetary sources. In other words: allocating resources where they are most effective [43]. There is a positive correlation between the risk the investors assume and the return they obtain. This means that even though stock trading has a higher return than other financial operations, it also has a higher risk. Careful analysis of each transaction is needed. Prediction is used for decision making on where and how much to invest, and it is normally based on information such as stock price historical data, accounting information of the firm, macroeconomic and statistical indicators, the news, and other technical information. When all the information is gathered, the decision is made and an action is taken.

In this work, we focus on the use of stock price historical data, financial indicators and customizable trading parameters to aid short term investors in finding an optimal trading strategy.

Historically, difficult tasks have been assigned to computers to reduce error margin and increase precision. Why not assigning this task to an algorithm and decide where and when to trade? Several years ago, humans were replaced by computers when quick trading decisions and quantitative analysis were needed. A computer could perform calculations faster than a human.

This work proposes Deep Reinforcement Learning (DRL) as the method to train an intelligent agent that will learn to trade based on historical trading data. By the end of the training, the agent should be capable of advising the user in which stocks to invest, given the personal economic circumstances of the user and personal preferences. In order to do this, an analysis of different modeling and different DRL algorithms will be carried out. We propose an automated tool to manage stock portfolio and optimise financial investing to obtain the best return. If the goal for the spending units is getting profitability from their current money, this tool would save them time, and could also be used by those who did not have an advanced financial or programming knowledge level.

All the code used for the development of this project can be found at Codecamaru [13] on GitHub.

1.1 Motivation

1.1.1. Personal Motivation

Pursuing a dual degree in Informatics Engineering and Business Management and Administration, I have been keen to dive deeper into this intersection of these two fields.

When doing a low level implementation of the software classes that define the environment of a Reinforcement Learning (RL) problem, every class needs to be adapted to the specific characteristics of the scenario. With such a complex environment as the stock market, I was really curious to understand the object model design that could make the system work.

Moreover, I was intrigued by the possibility of developing an intelligent agent within this framework. My goal was to explore whether an agent could learn to operate successfully in the stock market, thereby integrating my dual interests in technology and finance to address real-world challenges.

1.1.2. Professional Motivation

Spending units can save their money by placing it in a bank account and obtain a low rate return in the best case as well as convert the money into another financial asset that allows for a higher rate return in the best case. In the past six years, bank deposit interest rates have not reached the one percent [14]. This does not allow the spending units for a sufficient amount of savings to meet their financial needs. Most of the companies and 12.33% of the families invest in stock trading [12]. Family spending units can benefit from financial investing to cushion emergency times such as a decrease or loss of income or an increase in debts [30], and most of the firms manage their financial structure in order to obtain the maximum possible profitability [11]. Therefore, it seems reasonable to provide them with a suitable tool that has the right strategy to do so.

Quantitative analysts, also known as *Quants*, develop mathematical models that predict market movements. They use statistical techniques to analyze historical data and identify trading opportunities, and apply quantitative methods to manage and mitigate financial risks. The algorithms they design execute trades based on predefined criteria. These algorithms can process a vast amount of market data at high speeds to make trading decisions faster than human traders. These models are essential for making informed trading decisions and for the valuation of financial instruments. Nonetheless, most of the quantitative analysis techniques are not based on RL. We want to know if RL can play a part in this process and we propose a similar study in the efficiency of different algorithms with an RL methodology to do so.

The RL paradigm comprises controlling a system so as to maximize a numerical performance measure [37]. Similarly, effective investing in stock markets, as well as in other financial assets, aims to maximize profits. It is promising to conduct research into this application of DRL methods in Finance. This project will focus on the utilization of DRL in stock exchange. DRL applied to Finance has gained attention in the past couple years, but it is not one of the most explored fields. It seems only reasonable and interesting to dive deeper into this spectrum. Stock markets are increasingly complex and have dynamic properties. Financial data contains noise and has a non-stationary condition, so inflexible trading strategies will not succeed. Instead, the dynamical nature of Reinforcement Learning seems suitable to meet this demand [43].

Furthermore, every technical indicator serves a different trading objective, but there are many similar ones. A technical indicator is a mathematical formula applied to the pricing information of an asset. There are many formulas to measure different information on the data. We want to understand how different combinations of technical indicators on the RL modeling impact on the performance of the trading agent.

1.2 Main Goals

The main objectives of this work can be summarized as follows:

- To train several RL agents with different modeling of technical indicators and determine if they can learn how to trade.
- To analyse the differences in training and testing among the different modeling.
- To detect if there is a specific combination of technical indicators with which the agent has a better performance.
- To build a basic program that gives trading advice to users without programming or financial knowledge.

1.3 Expected Impact

In this project, we recommend DRL as a solution for the creation of trading strategies. By automating the process, a considerable amount of time could be saved in the decision making process.

There are two potential users: an individual who is willing to trade, and a company that needs to manage its portfolio through short term investments.

This tool would make financial advise available for many users. Democratising investments in this way and contributing to the eighth Sustainable Development Goal: “Decent Work and Economic Growth” [29]. According to Global Goals [20]: "Economic growth should be a positive force for the whole planet. This is why we must make sure that financial progress creates decent and fulfilling jobs while not harming the environment. [...] If we promote job creation with expanded access to banking and financial services, we can make sure that everybody gets the benefits of entrepreneurship and innovation". Our tool could enhance family economies and ease their access to finance.

1.4 Structure of this work

The content of each chapter is described as follows:

- Chapter 2 provides a comprehensive overview of the essential concepts necessary to understand the project, covering both financial aspects and reinforcement learning background. It also reviews the state of the art and technologies similar to the tool we employ.
- Chapter 3 details our problem statement more explicitly. It presents a thorough description of the proposed tool, FinRL, and outlines our solution design.
- Chapter 4 explains the training and testing processes we undertake, as well as the conceptualization and development of the application we create.
- Chapter 5 presents the conclusions we draw regarding the use of Deep Reinforcement Learning in finance. It also discusses the challenges we face and potential future lines of work.

CHAPTER 2
Background

This chapter presents a Reinforcement Learning (RL) background that is important to understand the subsequent content of this work and will also discuss the current Deep Reinforcement Learning in Finance state of the art. Additionally, after the RL background section, another one is dedicated to explain financial concepts such as technical indicators in order for the reader to understand the rest of this work. The reader, though, does not need to understand all the formulas to be able to follow the rest of the work.

2.1 Reinforcement Learning

RL aims to make an agent learn by evaluating its behaviour. The agent starts the training at a certain instant t and keeps taking actions that lead to different scenarios. It differs from other types of learning because it does not make the agent learn by giving instructions [36]. Rather, the agent learns by trial-and-error, balancing exploration and exploitation. The former is about discovering potentially better solutions by randomly picking the next action. The latter is about utilizing the current knowledge to choose the action that maximises the reward at that time step.

An RL problem is defined by a sequence of random variables representing observations $O_t \in \{O_1, O_2, ..., O_n\}$, actions $A_t \in \{A_1, A_2, ..., A_k\}$, and rewards R_t, which are generally defined by a function. There are two entities: agent and environment. They both interact at every time step. At each step t the agent executes an action A_t that the environment receives. For this action, the environment produces an observation O_{t+1} and a reward R_{t+1} that the agent will receive at the next step [33]. In other words, the environment assigns a score (how good the performance was) to the chosen action given the current environment state. Every agent-environment iteration is considered a step. Along the steps, the agent will try to maximize the reward.

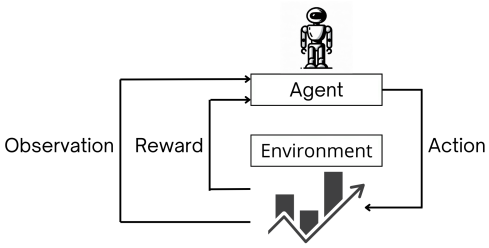


Figure 2.1: Basic schema of RL

Markov Decision Process

The formal definition of the aforementioned RL problem is generally described as a Markov Decision Process (MDP), which is a tuple $\langle S, A, P, R, \gamma \rangle$ described as follows:

- $S = \{s_1, ..., s_j\}$ is a set of states
- $A = \{a_1, ..., a_k\}$ is a set of actions
- P is a state transition probability function that given a state S and an action A to apply to that state S, it returns a value that belongs to the real numbers.
- R is a reward function that follows the equation: $R_s^a = \mathbf{E}[R_{t+1}|S_t = s, A_t = a]$
- A discount factor $\gamma \in [0, 1]$ that weights the value of future rewards

Gamma γ is used to define the total discounted reward G_t:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

The ending goal of RL is to maximise the expected total discounted reward G_t. Depending on the value assigned to γ, little or great importance will be given to future rewards; leading to myopic or far-sighted evaluation when near zero or one, respectively.

The **policy** π defines the behaviour of the agent by a distribution over actions given states [33]: $\pi(a|s) = \mathbf{P}[A_t = a|S_t = s]$. It maps the action of a state. It can be deterministic $a = \pi(s)$ or stochastic $\pi(a|s) = \mathbf{P}[A_t = a|S_t = s]$.

The **state value function** $v(s)$ is the expected return G_t of a state. When following a policy π, the state value function $v_\pi(s) = \mathbf{E}_\pi[G_t|S_t = s]$ can be defined by the Bellman equation as follows [37]:

$$V^\pi(x) = r(x, \pi(x)) + \gamma \sum_{y \in S} \mathbf{P}(x, \pi(x), y) V^\pi(y); \quad where \quad x \in S$$

The optimal state value function satisfies the Bellman Optimality equation for $V^*(x)$:

$$V^*(x) = \max_{a \in A} \left\{ r(x, a) + \gamma \sum_{y \in S} \mathbf{P}(x, a, y) V^*(y) \right\}; \quad where \quad x \in S$$

The **action value function** $q_\pi(s, a)$ is the expected return G_t of taking an action, from a state, and following a policy: $q_\pi(s, a) = \mathbf{E}_\pi[G_t|S_t = s, A_t = a]$. The optimal action value function is therefore defined as the maximum $q_*(s, a)$ over all policies.

The training of an RL agent usually starts with a high rate of exploration that will decay along the way.

We talk about **evaluating** a policy when the expected total discounted reward is given from a state and following a policy: $v_\pi(s) = \mathbf{P}[G_t|S_t = s]$. On the other hand, **improving** the policy means updating the policy towards the optimal one π^*, which is following a greedy strategy with respect to v_π. The process of alternating between policy evaluation and policy improving is known as **policy iteration** [37][33].

RL methods

There are three types of methods: critic-only, actor-only, and actor-critic.

Critic-only methods involve learning a value function which estimates how good it is to be in a given state. **Actor-only** methods directly learn the policy that maps states to actions without explicitly modeling the value of each state-action pair. **Actor-critic** methods implement policy iteration and combine critic-only and actor-only [37]. The actor updates policy parameters in direction suggested by the critic and the critic updates the action value function parameters [33].

Therefore, to find the optimal policy, there are two methods: value-based and policy-based [21]. **Policy-based** methods do not use a value function, but rather directly train the policy to find the optimal one. With so, starting from a state, the policy $\pi(s)$ will give an action (or a probability distribution of the actions) from that state. **Value-based** methods train a value function. Given a state s, we compute $q_\pi(s, a)$ for every possible action from that state. This gives state-action pairs. Since we do not train the policy, we need to specify the behavior. If maximizing reward is preferred, the highest value pair will be the chosen action.

Temporal Difference (TD) learning updates the value function or policy after an episode. **Monte Carlo** learning waits for the end of the episode to do so [40].

Algorithms

A2C stands for **Advantage Actor Critic** [38]. It uses the actor-critic method, combining value-based and policy-Based methods, and a function called the Advantage function as Critic, instead of the action value function. This helps stabilizing the training by reducing the variance that arises when using Monte-Carlo sampling to estimate return. This is due to the many possible and different expected returns given a state. The Advantage function is defined as: $A(s, a) = q(s, a) - v(s)$. This is the relative advantage of an action with respect to all the other possible ones from that state. Furthermore: if $A(s, a) > 0$, the gradient will be adjusted so that a is favorable; if $A(s, a) < 0$ the gradient will be adjusted so that a is less favorable.

It is empirically known that to converge to an optimal policy, smaller policy updates are better [39]. In addition, having policy updates in steps that are too big can lead to worse policies, enlarging the time to recover. **Proximal Policy Optimization** (PPO) improves training stability by avoiding policy updates that are too large. We want to limit the change in the policy at each training epoch. The idea is to clip the ratio of change from the old policy to the current policy. The ratio function is:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$$

With this, the ratio $r_t(\theta)$ is clipped in $[1 - \epsilon, 1 + \epsilon]$, where ϵ is a constant. Note that the full set of equations was not provided in order to simplify the concept behind the algorithm.

Deep Deterministic Policy Gradient (DDPG) is an algorithm which concurrently learns a Q-function (action value function) and a policy [31]. It uses off-policy data (information about the environment that is collected from a different policy than the one currently being learned) that can only be used for environments with continuous action spaces. It uses an experience replay buffer. This is the set D of previous experiences so that the agent has stable behavior [31] (we randomize samples to avoid correlation). In Deep Q-Networks (DQN), two neural networks are used. One to approximate the Q value of the current pairs of state and action in t, and another network with fixed param-

eters for the Q values of the pairs in $t+1$ (in order to estimate the Temporal Difference target). The second network can be updated every certain number of steps, but it is just copied over from the main network. We minimize the loss function:

$$L_i(w_i) = \mathbf{E}_{s,a,s'}[(r + \gamma \times max_{a'} Q(s', a'; w_i^-) - Q(s, a; w_i))^2]$$

The second network previously mentioned is the term $Q(s', a'; w_i^-)$ and the term $r + \gamma \times max_{a'} Q(s', a'; w_i^-)$ is the target. But in DDPG, the target network is updated as follows (which is known as polyak averaging):

$$\phi_{\text{targ}} \leftarrow \rho \phi_{\text{targ}} + (1 - \rho)\phi; \quad where \quad \rho \in [0, 1]$$

TD3 means **Twin Delayed DDPG**. It is an off-policy algorithm and can only be used for environments with continuous action spaces, as does DDPG. It adds three main features. It learns two Q-functions (action value functions) instead of one (hence "twin") [42], updates the policy less frequently than the Q-function, and adds noise to the target action. The latter is done to make it harder for the policy to exploit errors, and it is called target policy smoothing.

Soft Actor Critic (SAC) is an off-policy algorithm that optimizes a stochastic policy. It incorporates the double action value function learning. SAC adds to the state value function a term that measures the entropy (which indicates the information or learning that the agent has). In this way, it evaluates the randomness in the policy (the balence of exploration-exploitation). The entropy is inversely proportional to the information, and directly proportional to the exploration, in our case. With this, the policy is trained to maximize a trade-off between expected return and entropy [41].

2.2 Financial Background

Often, when deciding to make an investment, a cost of opportunity is measured. As an example, the reader can imagine an individual with 100 euros. This money can be invested with an interest rate of return i that can vary depending on the asset (whether it is a bond, stock, a derivative...). The interest rate determines the value of an initial amount of money at the end of a period of time. Its value depends on the length of the period of time taken as reference. With an annual interest rate i_a, after a year, the initial amount will be worth $V = 100 \times (1 + i_a)$. Therefore, the cost of opportunity that the individual needs to measure when deciding whether to invest is the offered interest rate in the market.

Similarly, to calculate the current value at time t of future cash flows (money that will be received in the future), we adjust the future value based on the current offered interest rate in the market, so that the adjustment truly reflects the actual cost of opportunity. For a yearly future cash flow of 100 euros during four years, the current value would be:

$$V = \frac{100}{(1 + i_a)} + \frac{100}{(1 + i_a^2)} + \frac{100}{(1 + i_a^3)} + \frac{100}{(1 + i_a^4)}$$

This adjustment has a discount effect on the future cash flows, similar to the total discounted reward equation G_t.

The interest rate is often bounded with the concept of profitability, because once the asset's future value V' is known, profitability can be computed as follows:

$$Profitability = \frac{V' - V}{V} * 100$$

Technical Indicators

A technical indicator consists of data points generated by applying a specific formula to the pricing information of a security. This pricing information may comprise any combination of the opening price (open), highest price (high), lowest price (low), or closing price (close) over a designated time frame [35]. The following is the explanation of the technical indicators that we use for the comparative model analysis.

A **Moving Average** (MA) is a time series formed by computing the averages of successive segments from another time series [23]. A **Simple Moving Average (SMA)** is given by:

$$SMA = \frac{A_1 + A_2 + ... + A_n}{n}$$

where A_n is the price of an asset at period n and n is the number of total periods. The **Exponential Moving Average (EMA)** is computed as:

$$EMA = Price(t) \times k + EMA(t-1) \times (1-k)$$

being $k = \frac{2}{(n+1)}$ and n the number of days in EMA. A variation is the **Triple Exponential Moving Average (TEMA)**, given by:

$$TEMA = (3 \times EMA_t) - (3 \times EMA_{t-1}) + EMA_{t-2}$$

The **Volume-Weighted Average Price (VWAP)** is calculated by dividing the total traded dollars for every transaction in the day by the total shares traded:

$$VWAP = \frac{\sum(TP \times volume)}{\sum volume}$$

where typical price (TP) is $TP = \frac{high+low+close}{3}$ of the session.

Bollinger Bands consist of three lines: middle, upper and lower. The middle band is typically the simple moving average (SMA) of the closing prices over a set number of periods. The upper band is typically calculated by adding two standard deviations to the middle band. The lower band is calculated by subtracting two standard deviations from the middle band.

The **Relative Strength Index (RSI)** is a momentum oscillator that quantifies the velocity and magnitude of price changes [16]. The formula is:

$$RSI = 100 - \frac{100}{1 + \frac{Avg.UpwardPriceChange}{Avg.DownwardPriceChange}}$$

Stochastic RSI (StochRSI) gives traders an idea of whether the current RSI value is overbought (the price of an asset is considered too low relative to its usual levels, usually when StockRSI is 70) or oversold (the price of an asset is considered too low relative to its usual levels, usually 30).

$$StochRSI = \frac{RSI - min[RSI]}{max[RSI] - min[RSI]}$$

True Range (TR) formula is given by:

$$TR = \max[(high - low), |close_{t-1} - high|, |close_{t-1} - low|]$$

The Average True Range (ATR) simply consists of computing a moving average of the already computed TR values over a period of time.

The **Average Directional index (ADX)** is used to determine the strength of a trend.

$$ADX = \frac{ADX_{t-1} \times (n-1) + DX_t}{n}$$

where $DX = \frac{|(+DI)-(-DI)|}{(+DI)+(-DI)}$, $+DI = \frac{smoothed(high_t - high_{t-1})}{ATR}$, $-DI = \frac{smoothed(low_t - low_{t-1})}{ATR}$, and n is the number of periods to consider.

The **Z-Score (Z)** indicator can be computed as $z = \frac{X-\mu}{\sigma}$.

The **Awesome Oscillator (AO)** is used to anticipate trends:

$$AO = SMA(MedianPrice, 5) - SMA(MedianPrice, 34)$$

being 5 and 34 the number of periods in SMA, and $MedianPrice = \frac{High+Low}{2}$.

Williams Overbought (WO) measures overbought and oversold levels $WO \in [0, -100]$. It tries to find entry and exit points in the market over the fixed period.

$$WO = \frac{Highest - Close_t}{Highest - Lowest}$$

The **Percentage Price Oscillator (PPO)** helps in comparing asset performance and volatility, and in confirming trend direction. It is usually calculated as:

$$PPO = \frac{EMA_{12period} - EMA_{26period}}{EMA_{26period}}$$

Supertrend indicator determines the trend direction of a market asset. It combines the concepts of volatility and price movements to provide a comprehensive view of the market conditions. It is composed by two bands:

$$upper = \frac{high + low}{2 + (k \times ATR)}$$
$$lower = \frac{high + low}{2 - (k \times ATR)}$$

where k is a sensitivity parameter.

Choppiness Index (chop) measures market stability.

$$chop = \frac{\log_{10}(\frac{\sum_{i=0}^{n} ATR}{High_{nPeriods} - Low_{nPeriods}})}{\log_{10}(N)}$$

The **Commodity Channel Index (CCI)** determines whether an asset is reaching the condition of overbought or oversold. It helps traders decide on next action step: exit or exit a trade.

$$CCI = \frac{TP - MA}{0.15 \times MeanDeviation}$$

The **Triple Exponential Average (trix)** shows the percentage change in a three times exponentially smoothed MA [24].

Our study turns around a specific **stock market index**, which is a statistical measurement system that shows the evolution of stock prices over time relative to a baseline

date. The index that we use for our analysis is the **Dow Jones Industrial Average (DJIA)**, commonly known as the Dow Jones 30 (or Dow-30). It is a price-weighted index that represents the stock performance of thirty large, publicly-owned companies based in the United States. This means that companies with higher stock prices have more influence on the index's performance, unlike most major indices which are market-capitalization weighted.

The **ticker** is commonly known as the abbreviation of the names of companies. They are are unique identifiers made up of letters, numbers, or a combination of both, assigned to publicly traded shares of a particular stock on a particular stock exchange. These symbols are used to facilitate the clear and quick identification of companies on stock tickers, trading platforms, and financial reporting. For example: AAPL for Apple Inc., GOOGL for Alphabet Inc. (Google), or MSFT for Microsoft Corporation.

2.3 Deep Reinforcement Learning in Finance

Deep Reinforcement Learning (DRL) applied to Finance has proven good results in previous studies. It has been used for stock price prediction or return prediction and portfolio optimization.

As demonstrated by Yu and Li [48], Long Short Term Memory (LSTM) networks can effectively forecast stock price index volatility. Similarly, Huynh et al. [22] have proposed a novel model for predicting stock price movements using deep neural networks. By including DQN and Policy Gradient methods to forecast stock price movements and execute trades that maximize financial returns, Li [27] results show that DRL models can outperform traditional quantitative and algorithmic strategies, providing a novel toolset for financial analysts and traders. Yang et al. [47] combine three actor-critic based algorithms to form an ensemble strategy aimed at maximizing investment returns. The ensemble approach is designed to leverage the strengths of each algorithm, enhancing the robustness and adaptability of the trading strategy to various market conditions. Jiang et al. [25] introduce a DRL model that addresses the portfolio management problem by trading multiple assets in a financial market. The model uses a convolutional neural network (CNN) to capture the temporal patterns among different assets. Wu and Fujita [43] proposes Gated Recurrent Unit (GRU) to summarize the market conditions from raw data and technical indicators of stock markets, and critic-only Gated DQN and actor-critic Gated Deterministic Policy Gradient (GDPG) to make trading decisions.

The inconvenience of using DRL models in finance is the practical implications of deploying such RL models in real-world trading scenarios that are the need for continuous learning due to market and data evolution, which is discussed by Jiang et al. [26].

Especially when combined with other methods from Deep Learning such as feature extraction from articles or sentiment analysis from documents, it can lead to additional information and better accuracy with combined results. The latter combination is useful to analyse the impact the press has in the stock prices when reporting about new economic and company policies. Recurrent neural networks output a sequence of vectors $h_t = (h_1, h_2, ..., h_n)$ based on an input sequence of vectors x_t, and a hidden state (the previous one–hence it's name 'recurrent') h_{t-1}, computing the hidden state as $h_t = g(Wx_t + Uh_{t-1} + b)$. They can be used for text recognition or sentiment classification, but they frequently have exploding or vanishing gradients when dealing with great amount of information, which hinders the training. To solve this, LSTM, which are a type of recurrent network, incorporate a cell with three gates. The Forget gate will indicate which information of the previous memory state is forgotten. The Input gate will determine which new information to add to the current state. From all the information

in the current state, the Output gate will compute which will be the output sequence. GRU networks merge the forget and the input gates into a single one and are often faster. Huynh et al. [22] introduced Bidirectional Gated Recurrent Unit (BGRU) networks, which have two recurrent nets connected to the same output layer, to represent the left and the right context of a sentence. They achieved higher accuracy with the BGRU than with LSTM or GRU, training with Reuters and Bloomberg news to predict stock prices. Their system performed better than the one that depended on historical prices only.

2.4 Technological Context

Gym-mtsim [10] is similar tool that generates RL simulations on stock markets. It is specifically tailored to simulate the MetaTrader 5 trading platform [28]. It does not provide a general approach suitable for various trading platforms and data sources.

The nature of gym-mtsim extends beyond simple simulation; it is engineered to be a multifunctional toolkit facilitating the entire trading process—from initial strategy testing to detailed performance analysis with advanced visualization tools. The architecture of gym-mtsim is composed of the following components: a simulation class that handles the operational aspects of MetaTrader, and a custom Gym environment built on top of MtSimulator. The latter is tailored for conducting reinforcement learning experiments where trading decisions are modeled as actions within a defined state space. This tool focuses on the ease of use and high code readability, aiming to democratize access to sophisticated trading simulation tools.

AnyTrading [9] offers a suite of OpenAI Gym [15] environments designed for developing and testing RL algorithms aimed at trading. These trading algorithms are predominantly used in the FOREX and stock markets. The primary goal of AnyTrading is to enhance and simplify the process of creating and evaluating RL-driven trading strategies through the provision of specific Gym environments. This objective is achieved by introducing three types of environments: TradingEnv (a generic environment), ForexEnv, and StocksEnv (which inherit and extend TradingEnv).

Gym-anytrading primarily offers environments without extensive additional tools. It serves as a foundational tool that can be expanded through user implementations and extensions. It is only an environment provider; it does not include a full suite of tools for data processing, feature engineering, trading strategy simulation, backtesting, and deployment.

As for general purposes and applications other than finance, there are two well-known open source libraries to simulate RL environments: Gymnasium [15] and Stable-Baselines3 [34].

Stable Baselines is a popular library for RL that provides implementations of various state-of-the-art RL algorithms, optimized for ease of use and efficiency. It is built on top of machine learning frameworks like TensorFlow (Stable Baselines) and PyTorch (Stable Baselines3). Its modular design allows users to customize and extend existing algorithms. It has multiprocessing support, being able to train agents in parallel, speeding up the training process. Users can save and load trained models.

CHAPTER 3
Proposed Solution: FinRL

3.1 Problem Statement

There exists a necessity for individuals to invest; however, making informed financial decisions is both time-consuming and complex. Automating this process appears to be a desired solution. As previously discussed, RL seems like a promising paradigm for this matter. Consequently, the goal is to train RL agents to automate trading strategies, which primarily focus on making decisions about where to trade and in what quantities. Essentially, this involves determining whether to buy, hold, or sell stocks.

The ultimate objective is not to achieve a specific profitability or return, but rather to enable the agent to learn trading strategies by identifying profitable opportunities and acting accordingly. This involves managing a portfolio over time, using days as the basic unit of time.

Much like a human trader, the agent begins with a set amount of capital (money and/or owned stock). It must invest this capital wisely in the market by purchasing securities (assets) and determining the optimal time to sell (liquidate) them, based on data received from the environment. The trading objective is framed as a maximization problem, where the effectiveness of the DRL trader is measured by the attainment of positive rewards.

3.2 FinRL

We proceed to explain the framework we aim to evaluate as a potentially appropriate tool for evaluating trading strategies and training agents that could automate trading.

AI4Finance [2] is a nonprofit organization that promotes AI in the financial industry. It develops open source code libraries in which every student or professional can collaborate. Their three main open source projects are FinRL, FinGPT, and FinRobot. It can be found at AI4Finance [2]. It is an educational resource.

Our focus remains in FinRL, which stands out by its accessibility and reproducibility. FinRL [4] is a framework for financial RL. When it comes to modeling trading strategies with DRL, it has two major advantages: portfolio scalability and market model independence. The former means that the DRL framework can manage portfolios of various sizes and complexities. For instance, it can manage different numbers of assets as inputs (from small portfolios to larger ones), which makes it interesting for real-world applications, where portfolio sizes can vary greatly. Market model independence means that the framework learns directly from the market data rather than making assumptions about

market behavior such as normal distributions in the data. This potentially makes it more flexible and robust to real market conditions. The library is developed in Python.

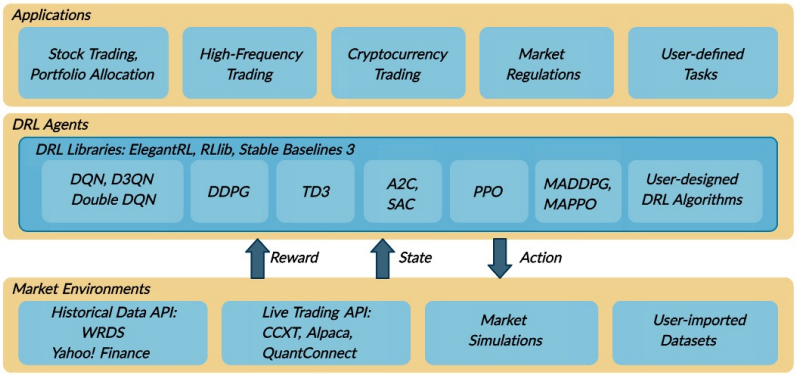


Figure 3.1: AI4Finance Foundation [6] FinRL layered architecture

The FinRL library is structured into three distinct layers: market environments, DRL agents, and applications. The foundational layer offers APIs to the higher layers, ensuring transparency and seamless integration. The agent layer engages with the environment layer through a balance of exploration and exploitation strategies, as shown in Figure 3.5. As can be seen in Figure 3.1, the layers are separated in modules that serve for different purposes, allowing for better adaptability by programmers.

The specific market environment development tries to accurately mimic live stock markets, accounting for transaction costs, market liquidity (whether the stock is able to sell), and the investor's level of risk aversion (turbulence index), which are vital in determining net returns when investing in short term and with medium frequency. The market environments are based on OpenAI [32] Gym and use time-driven simulations.

The training methods allow for a structured pipeline that comprises training, validation, and testing stages. Initially, the DRL agent is trained within a controlled environment, followed by validation in a separate environment for additional refinements. Subsequently, the refined agent undergoes testing using separated historical datasets. Upon successful testing, the result is a trained agent (or `stable_baselines3` trained model) that can be deployed for live trading–it can be used to predict new data and make real trades.

This pipeline (Figure 3.2) effectively addresses the issue of information leakage, as trading data is strictly segregated during the agent's adjustment phase. Furthermore, this standardized pipeline facilitates equitable comparisons among various algorithms and trading strategies.

The Agent layer provides algorithms developed by three DRL libraries: ElegantRL, RLlib and Stable Baselines 3. Our project utilizes the latter. As for the applications layer, this project focuses on the stock trading application.

To provide the reader with a deeper understanding of FinRL, the main classes among its file architecture (Figure 3.3) will be explained.

The `stock_trading.py` script inside the applications folder (Figure 3.3) is designated to be run as a standalone script, containing a single function that defines training and trading periods and toggles specific RL algorithms on or off to run a whole simulation of stock trading. It makes use of all the other classes that develop this application (the ones that are below in Figure 3.3). However, this script does not allow for a flexible and

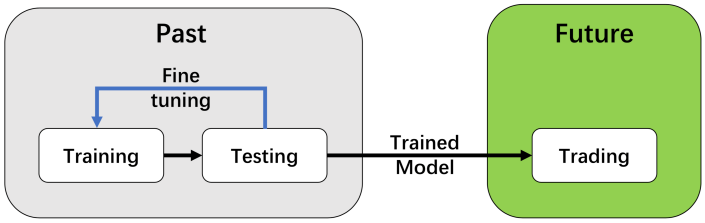


Figure 3.2: AI4Finance Foundation [6] Fine tuning training process

```
FinRL
├── finrl (the main folder)
│   ├── applications
│           ├── cryptocurrency_trading
│           ├── high_frequency_trading
│           ├── portfolio_allocation
│           └── stock_trading
│   ├── agents
│           ├── elegantrl
│           ├── rllib
│           └── stablebaseline3
│   ├── meta
│           ├── data_processors
│           ├── env_cryptocurrency_trading
│           ├── env_portfolio_allocation
│           ├── env_stock_trading
│           ├── preprocessor
│           ├── data_processor.py
│           └── finrl_meta_config.py
│   ├── config.py
│   ├── config_tickers.py
│   ├── main.py
│   ├── train.py
│   ├── test.py
│   ├── trade.py
└──└── plot.py
```

Figure 3.3: AI4Finance Foundation [6] FinRL file architecture

customisable implementation of stock trading applications. We don't make use of it in our solution. We rather implement our own notebook and program structure.

The agents folder contains classes subdivided in three folders: stablebaselines3, elegantrl, and rllib. As noted earlier, these directories correspond to maintained DRL libraries. The classes within the "agents" folder import these specific libraries and manage the instances created from them, functioning essentially as API wrappers for the libraries.

Inside the stablebaseline3 folder (the one we use in our implementation), there is one script and two classes: `models.py`, `tune_sb3.py` and `hyperparams_opt.py`. The class `DRLAgent` inside `models.py` is the main class to train the agent. It implements the getter model method, which initializes and returns a stablebaseline model based on the specified algorithm. This class also contains the training and prediction method of the agent. The `train_model` method has a callback to plot additional values in TensorBoard (a visualization toolkit from TensoFlow). The Learning Rate and other parameters can be modified through the dictionary `model_kwards`, which is passed by parameter to the `get_model` method. The agent is optimized through online learning. The `DRL_prediction` method returns the actions taken by the agent in a $d \times n$ size `DataFrame` (where n is the number of stocks and d is the number of desired days to make a prediction), as well as the daily value of the account. `DRL_prediction` internally calls the `predict` method from Stable Baselines and feeds it with the environment testing observations as input.

The files `tune_sb3.py` and `hyperparams_opt.py` are in charge of handling the fine tuning process shown in Figure 3.2. The former sets up with the required environments for training and testing, and the type of model to tune. It automatically manages the process of finding the best hyperparameters using advanced sampling and pruning techniques, and evaluates the best model creating a `DRLAgent` instance. The latter integrates Optuna (an automatic hyperparameter optimization software framework) to explore a hyperparameter space. In our solution, we do not implement the fine tuning process, but rather run one training process and then perform testing.

The files that fetch and preprocess our data are inside the preprocessor folder of meta (Figure 3.3). The preprocessors script implements dataset loading, splitting and datetime conversion functions, as well as the `FeatureEngineer` class. The latter implements functions for automating preprocess (adding technical indicators, volatility and turbulence indexes), and cleaning the data (dealing with missing values).

Thanks to the internal implementation, the user can easily fetch data from Yahoo Finance [44] and perform initial analysis and processing on the financial data. This is implemented in the `Yahoodownloader` class, making use of the `yfinance` preexisting Python library and transparent to the user. Efficient computation of technical indicators is feasible due to the integration of preexisting Python libraries in the framework. To compute the technical indicators, the `stockstats` Python library is imported. It supports automated calculation with standard `DataFrame` formats. The use of the `DataFrame` structure (from Python *pandas*) is fundamental to store all the relevant stock information in each step. Since the open-high-low-close price of every index is fetched from Yahoo Finance [44], many technical indicators can be calculated.

This serves well in accessing, cleaning, and extracting features from various data sources with both high quality and efficiency, and from diverse formats and sources into a cohesive framework. The class allows customisation of the following parameters when preprocessing the data: list of used indicators, boolean variables for the use of the volatility and turbulence index, and in case the user wants to define a feature. All of this is part of the preprocessor folder.

The env_stock_trading folder, located within the meta folder, is where the agent's actions are processed and transformed into a new State and Reward for each step. The environment we use ouf solution is `StockTradingEnv` from `env_stocktrading.py`. Occasionally, the agent may propose actions that are not feasible in actual scenarios. The `StockTradingEnv` class ensures that these actions are processed and modified into actions that can realistically be executed. The buy and sell stock methods are developed accordingly. This means that cases like whether there is enough cash to buy, balance updating, or checking if the stock is available to sell (market liquidity) are handled. As usual, the environment class has the three main methods: init, reset, and step. The init method is where all the used parameters and structures are initialized. The step method always returns the new state and reward. The reset method is used when an episode reaches the end, and its purpose is to bring the environment to an original or intermediate parameter configuration. The step method computes the step reward as `end_total_asset - begin_total_asset`. Both values take into account the available cash and the value of held stocks. To compute both, the cash is added to the total market value, and the latter is calculated by multiplying quantities and prices. All of this is done through `numpy.array` structures. If another parameter needs to be used when computing the reward, the user must create another customized environment. The returned state from the step method is a $[[1 + 2n + (mn)]]$ size `numpy` matrix (as explained in the previous section). In addition, this class acts as a dictionary updater. It keeps track of the asset, action, and state progress and stores them in `DataFrame` objects. The class allows for customisation and implementation of the following parameters: initial amount of money to invest with, ini-

tial number of stock shares that the agent owns, desired maximum number of trades per day, the cost of each transaction (due to possible commissions that some trading agencies might charge to investors), and list of used indicators.

When creating the environment instance for either training or trading, the class `StockTradingEnv` is converted into a Vectorized Environment from Stable Baselines (a technique for consolidating several independent environments into a single unified environment). This is done when calling the `get_sb_env` from `StockTradingEnv`.

In addition, FinRL provides additional mechanisms to detect when the variance of the prices is too high, and progressively starts selling when possible. This is particularly useful to prevent investors from situations like the Great Recession.

The library gives the possibility to train agents and experiment with the following stock indexes: Dow Jones Industrial Average, NASDAQ-100, Standard & Poor's 500, Hang Seng China 50, Shanghai Stock Exchange 50, Shanghai Shenzhen CSI 300, *Cotation Assistée en Continu* 40, *Deutscher Aktienindex* 30, Technology DAX, MidDAX 50, SmallDAX 50, Jakarta Stock Exchange Liquid 45, Sustainable and Responsible Investment *KEHATI*, and Foreign Exchange. All the components of these are listed in the `config_tickers.py` file, which is a configuration file that defines the ticker (abbreviation of the company name) parameters that are called from other functions. We use this file in our solution to get the ticker list of the Dow-30 constituents. A similar file is located in the meta folder as `finrl_meta_config.py`.

The `config.py` is another configuration file used to define model training parameters such as the learning rate, batch size and entropy coefficient, among others, for the RL algorithms. To train the algorithms in our solution, we make use of this file but we modify some parameters.

The train, test and trade scripts separately carry out the corresponding training, testing and trading processes. The train script calls the `train` function with the parameters in the `config.py`. The test script calls the `DRL_prediction` method from the chosen DRL library. Finally, the trade script conducts real trading using the `AlpacaPaperTrading` class, which implements a paper trading (trade without real money) environment in Alpaca [8] (API that allows developers to automate trading strategies and offers brokerage services for trading financial assets). These are FinRL pre-built scripts. We do not use them in our solution.

To sum up, we have designed a rough schema of the basic interactions that take place in FinRL when we run our solution using the elements shown in Figure 3.4.

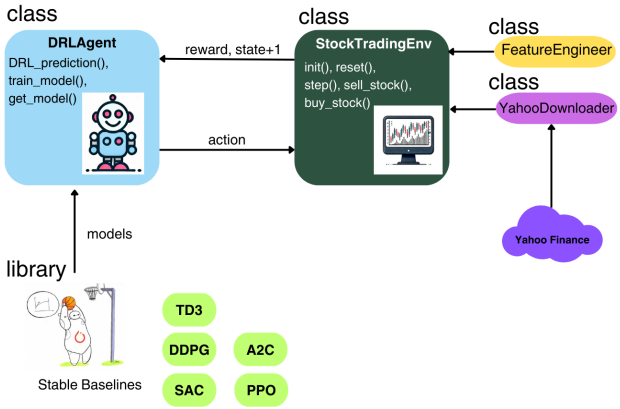


Figure 3.4: Main used methods and classes from FinRL that interact in our solution (environment and robot pictures generated with GPT)

3.3 Solution Design

FinRL bases its algorithm development on the following MDP approach:

- $S \in \{\text{balance, close price, shares, technical indicators}\}$.
  - The agent gathers all the following information at the end of the day (when the market is closed), and is able to make a prediction for the next day.
  - Balance is the amount of money left in the bank account at the current time step t and it is computed as: $Balance(t) = Balance(t-1)+$ (amount of money received for selling shares in t) - (amount of money payed to buy shares in (t)). This is the available money (cash) that the agent had the previous day, adding the money the agent spent or gained on the current day by buying or selling shares, respectively.
  - The close price is the price the stock had when the market was closed in (t). There is one closing price for each stock.
  - Shares is the number of shares the agent owns at t for every stock. There is one value for each asset.
  - There can be as many technical indicators for the input as one desires. Each stock is measured by all the technical indicators.
  - Therefore, S is a vector of size $[1 + 2 \times n + (m \times n)]$, where m is the number of indicators and n is the number of stocks.
- $A \in \{-k, ..., -1, 0, 1, ..., k\}$ where k is the number of shares to buy and $-k$ the number of shares to sell. For n stocks, the entire action space is size $(2k+1)^n$. It is useful to set a parameter of maximum allowed trades per day.
- $R(s, a, s') = v' - v$ where v' and v represent the Account Values at state s' and s, respectively. The step reward is therefore $R = Value(t) - Value(t-1)$. The account value takes into account both the cash and the portfolio value. This means: cash in the bank account at t and value of all the shares the agent owns at t.

According to the official documentation [5], the agent derives the strategy to maximize the long-term accumulated rewards, which is the Q-value.

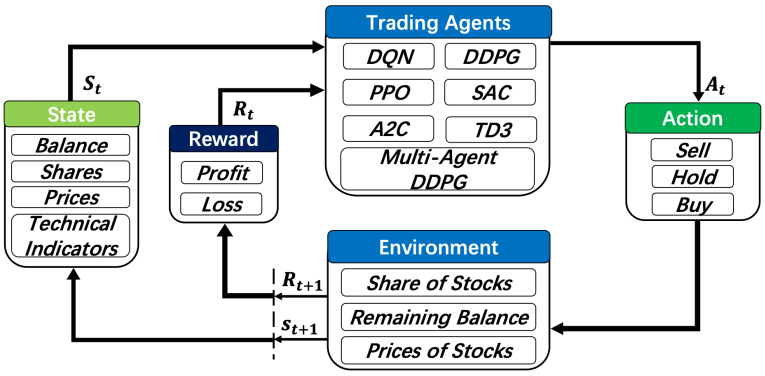


Figure 3.5: AI4Finance Foundation [6] problem construction in FinRL

In Figure 3.5 there is an illustration of the MDP in FinRL. We do not use all the trading agents shown in the picture, we train: PPO, SAC, A2C, TD3, and DDPG.

Briefly, our solution design consists of:

1. Understanding how FinRL implements software market environments, RL trader agents, and integrates both in a RL software system
2. Modeling seven MDPs differing in the used technical indicators that partially form the state space.
3. Training five RL algorithms with every modeling using FinRL.
4. Analysing the differences among them in training and testing.
5. Using the best configurations and corresponding trained agents to advise the final user on where, when, and how much to trade.
6. Developing a basic program that advises the user based on its personal and customisable circumstances.

To analyse the differences among the different RL models, we use different technical indicators. The technical indicators are part of the states in the MDP, and are explained in Chapter 2, Section 2.2. They are not predictions; they are calculated values based on historical price data after applying a formula. We vary their use in quantity over the models, using only one, four, and six technical indicators. This is to test whether the algorithm performance increases with more or less information in the state space (input). To differentiate each model, we name each of them as 'Set', because all of them contain a different sets of technical indicators. The sets with only one technical indicator have the sign '(-)' on their ID. The sets with six technical indicators have the sign '(+)' on their ID. In other words, we try to experiment with the size in number of the technical indicators to be imputed to the algorithms. We divide the modeling in the sets that are described in Table 3.1.

Table 3.1: Abbreviation of used technical indicators for every set in the training

Set	Used Technical Indicators in this set
1(-)	Simple Moving Average of the five last periods using the highest prices
1	Simple Moving Average of the five last periods using the highest prices, stochastic Relative Strength Index, Average Directional Movement
1(+)	Simple Moving Average of the five last periods using the highest prices, stochastic Relative Strength Index, Average Directional Movement, Bollinger Bands, Z-Score for the last 75 close prices, Awesome Oscillator
2	Williams Overbought, Triple Exponential Moving Average, True Range, Percentage Price Oscillator
3	Supertrend with the Upper Band and Lower Band, Simple Moving Average of the five last periods using the highest prices, Triple Exponential Moving Average, Choppiness Index
4(+)	Commodity Channel Index, Supertrend with the Upper Band and Lower Band, Average True Range, Triple Exponential Average, Volume Weighted Moving Average, Relative Strength Index
5(-)	True Range

To give the reader an example, the specific MDP for Set 1 (described at Table 3.1) is the following:

- $S \in$ {balance, close price, shares, Simple Moving Average of the five last periods using the highest prices, stochastic Relative Strength Index, Average Directional Movement}

- $A \in \{-k, ..., -1, 0, 1, ..., k\}$
- $R(s, a, s') = v' - v$

We use the Dow Jones Industrial Average (a price market index) for our experiments. Having its constituents as possible stocks to invest in (30 USA companies).

For each model from the sets in Table 3.1, we run all five RL algorithms. In total, we train 35 RL agents. A2C, PPO, DDPG, TD3, SAC agents for Set 1(-), for Set 1, and so on and so forth.

To analyse the performance and the learning progress, we gather training statistics, and the evolution of the Account Values of every algorithm (which takes into account both the money in the back and the value of the stocks). We do this in order to analyse differences in the results.

The proposed solution sets the day as the granularity measure; and the prediction depends on the previous state. This means that we cannot predict where to invest with a week in advance. For this, we need to wait until the previous day in which we wish to make the prediction in order to gather all the needed information. For instance, if the user desires to trade on a specific day, they can get the agent's recommendation at the end of the previous day, when the market has already closed. The agent will gather the data that forms the state space the previous day, and make a prediction for the desired day based on it. The stock closing prices that are part of that input can only be known when the market closes the previous day.

Finally, we developed the proposed app, which offers users a financial advisory service.

CHAPTER 4
Analysis and User App

4.1 Analysis

As previously mentioned, the market index we use is the Dow Jones Industrial Average (DJIA). Therefore, we train the algorithms with the historical stock prices from 30 companies. The training date range we use goes from January 1, 2010, to October 1, 2021. The testing date range used for this analysis goes from October 1, 2021, to March 1, 2023.

Before the training, the Preprocessing and dataset construction consist on the following steps. We fetch the data from YahooFinance [44] with the `fetch_data` method from the `YahooDownloader` class. We check that for correct downloaded amount of stocks and shape the `DataFrame` object to construct open, high, low, close, volume and ticker columns.

To build the combinations of technical indicators in each set, we randomly sample an array from a list, and we do it with different array sizes. Then, we use the `preprocess_data` method from `FeatureEngineer` class and obtain all the values of the technical indicators and turbulence for each stock and for each day. We use the `data_range` method from `Pandas` to assign the same date to every stock and assure the same granularity, to later remove the generated non-trading days (the market closes). We check for infinity or `NaN` values. Next, we split in train and test.

We assign the state space variable as $1 + 2 \times stock\_dimension + length(array_{indicators}) \times stock\_dimension$, as explained in the previous chapter, where the stock dimension is the number of unique tickers.

It is worth noting that after preprocessing stock prices with `FeatureEngineer` class, we notice that some technical indicators are not supported by the FinRL library. We rise an Issue [7] on GitHub to inform of this problem. Additionally, after some testing, we also notice that depending on the used market index, when fetching the stock data from `YahooDownloader` class, some companies might not even be available due to a `No timezone found` exception. Because of this, some tickers, or stocks, might be delisted and therefore, not used in the training. This is why we should always check the size of the returned arrays.

4.1.1. Training

We are ready to set the agent and environment parameters to create the class instances. The parameters we use in the training for the agent are described in Table 4.1. The trading parameters we use for the environment in the training are described in Table 4.2.

Table 4.1: Parameters for the Agent in the training

Parameter	A2C	PPO	DDPG	TD3	SAC
gamma	0.0001	0.0001	0.0001	0.0001	0.0001
entropy coefficient	0.01	0.01	-	-	0.1
learning rate	0.0007	0.00025	0.001	0.001	0.0001
batch size	-	128	128	100	128

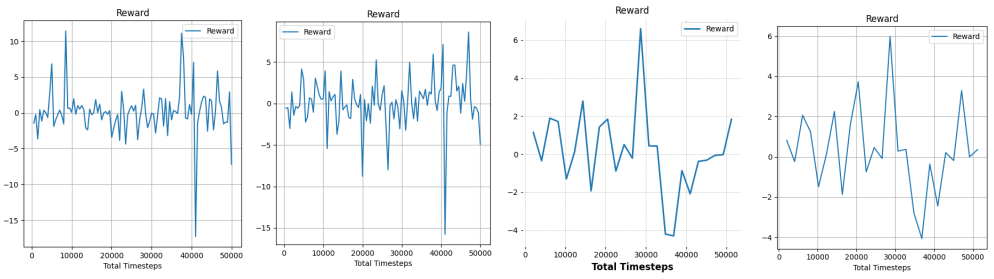
Table 4.2: Environment trading parameters in the training

Parameter	Value
transaction cost	0.001
number of owned shares	0
maximum amount of allowed trades in a day	80
initial amount of dollars to trade with	50000
Use turbulence	True

We create two different object instances of `StockTradingEnv` to simulate the training and the testing environment. For every algorithm, we create `DRLAgent` instances to get the corresponding model from `stable_baselines3` and be able to save and later load the models. Additionally, we set a logger to gather training statistics and be able to track the learning process.

As for the reward curve, since it is computed as $R = AV(t) - AV(t-1)$, where AV is the Account Value, it is strongly linked to the share price and, therefore, has a natural instability. After training all the algorithms in every set, we do not appreciate a clear increase and convergence of the curve on any set. The values seem to have more stability between $[-5, 5]$ for A2C and between $[-6, 6]$ for PPO in all of the sets, though. In the case of TD3, SAC and DDPG, the logger stores the reward at increased intervals, being harder and less precise to analyse it.

We notice that in PPO and A2C there is a drop in the reward towards the end of the training for most of the sets. This occurs approximately at the same time than the COVID crisis, which is a rare event that alters the stock prices. Even at the end of the training it could be reasonable that the agent does not know how to act in response to a rare event like this one. There are a few examples in Figure 4.1, where the curves in A2C have a reasonable stability until they reach timestep 40000 and have a sudden drop. Similarly, PPO reaches more regative rewards between timesteps 35000 and 40000.

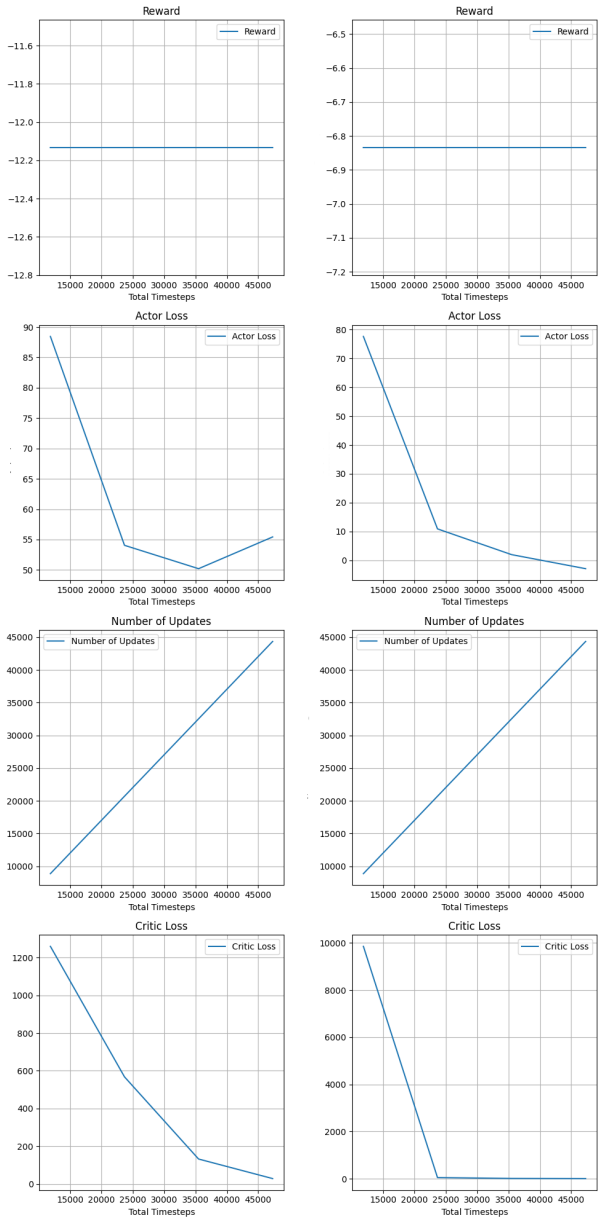


(a) A2C from Set 1(+) **(b)** A2C from Set 3 **(c)** PPO from Set 1 **(d)** PPO from Set 5

Figure 4.1: A2C and PPO Reward curve drop

After trying with different values for the learning rate, we always obtain a flat reward curve for the DDPG and TD3 algorithms. They do change in value among the sets, always

obtaining negative values ranging between -12 and -6, but always obtaining a negative curve (Figure 4.2).

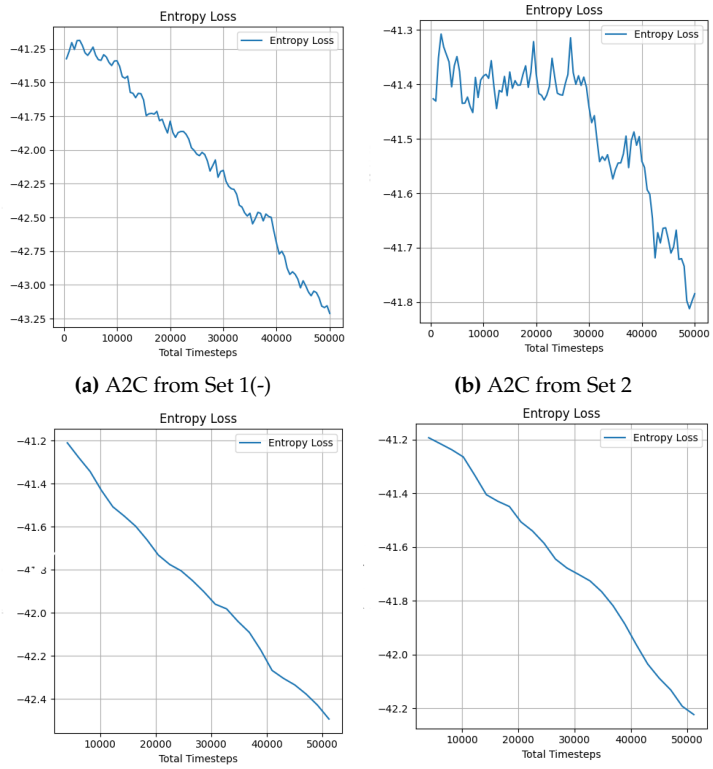


(a) TD3 from Set 4(+) **(b)** DDPG from Set 4(+)

Figure 4.2: TD3 and DDPG training stats: flat reward, Actor and Critic loss training statistics

The entropy loss in A2C and PPO always decreases progressively. This means that the agent does not become overly greedy too quickly. We show some of the cases in Figure 4.3.

The specifications of the machine in which the code was run are the following ones: macOS (operative system), 8-core GPU, Apple M1 (chipset model), 8-core CPU.



(a) A2C from Set 1(-)

(b) A2C from Set 2

(c) PPO from Set 1(-)

(d) PPO from Set 2

Figure 4.3: A2C and PPO entropy loss progressive decay

4.1.2. Testing

To compare the performance of each model, we first fetch the price evolution of the index directly from YahooFinance [44]:

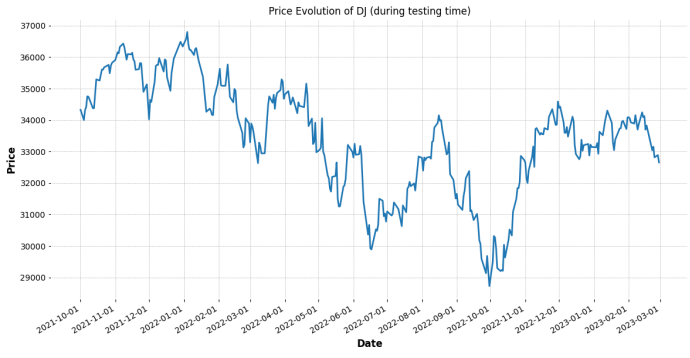


Figure 4.4: DJIA price evolution for testing dates

Then, we compute the portfolio value someone would get when investing the same initial amount of money on the starting test day and not moving it (daily index profitability multiplied by the same initial available amount of money we used for the training).

We use the portfolio or account value shown in Figure 4.5 as a baseline. This means, of course, that the comparative baseline is the same one for all of the sets.

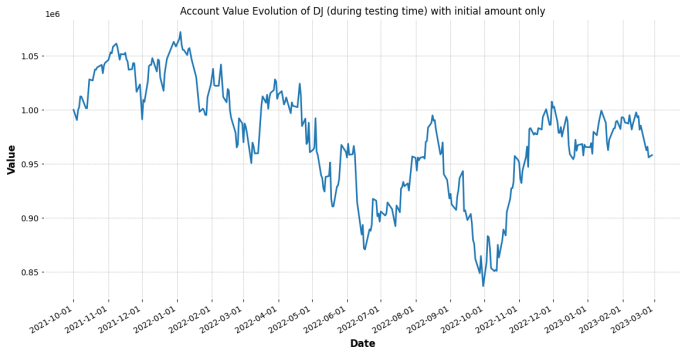


Figure 4.5: DJIA Account Value evolution for testing dates

We do this because one of the outputs of `DRL_prediction` method from `DRLAgent` class is a `Dataframe` object with the daily account value.

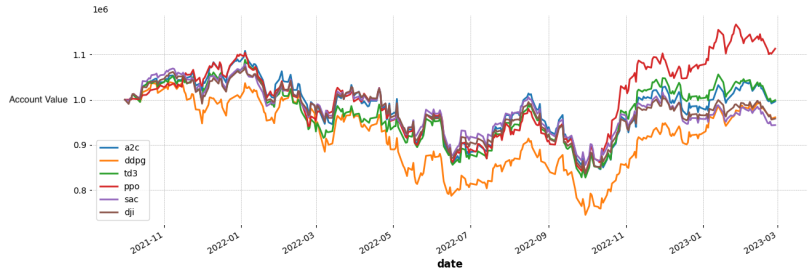
(a) Set 3

(b) Set 4(+)

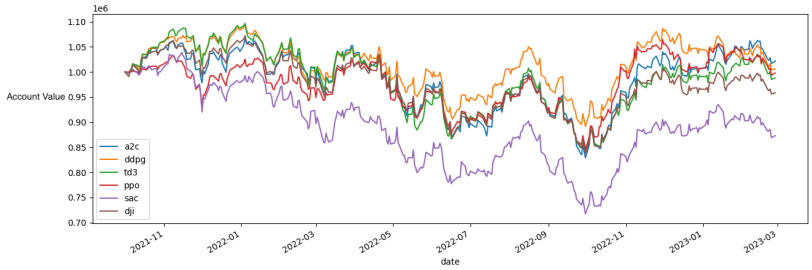
(c) Set 5(-)

Figure 4.6: Account value evolution throughout Test period in each set and each algorithm

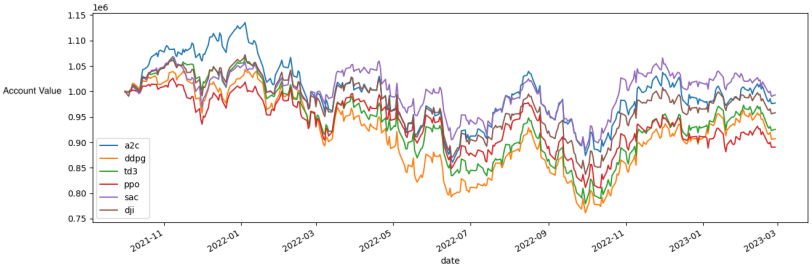
(a) Set 1(-)



(b) Set 1



(c) Set 1(+)



(d) Set 2

Figure 4.7: Account value evolution throughout Test period in each set and each algorithm

In Figures 4.7 and 4.6, we show the testing results for all the sets with the index performance (named 'dji') as a baseline. The unit for Account Value is measured in millions of dollars. The initial value for all the curves is 1.000.000 dollars. We remind the reader that the account value takes into account both the value of the owned stocks and the cash the individual has in their account. Therefore, we can affirm that the agent has learned

or knows how to trade obtaining good profitability when the account value of that algorithm is higher or equal than the baseline (dji).

Overall, the agents appear to have successfully learned how to maintain a stable trading strategy that allows for account profitability, as demonstrated by the consistent performance of multiple algorithms surpassing the baseline in each set. Particularly, Set 3 emerges as the most effective, with all RL algorithms outperforming the benchmark and achieving the highest account values compared to other sets. This suggests that incorporating technical indicators such as Supertrend, SMA of the highest last five prices, TEMA, and Choppiness Index in our model may offer a marginal improvement over other tested configurations. Similarly, set 4(+) and 1(+), have only one algorithm with a worse performance than the baseline, and the rest have a considerably better performance.

On the other hand, there is also one model in every set that underperforms the baseline. With this, we conclude that no single model uniformly excels across all scenarios; rather, it is the combination of technical indicators within each set that guides our selection of one set over another.

Moving forward, we utilize the trained and saved model from Set 3 for making predictions and providing recommendations to users, given that this set is the one with the best performance. Specifically, for the implementation of our app, we used the trained PPO, TD3, and A2C with the technical indicators: supertrend, SMA of the highest last five prices, TEMA, and Choppiness Index. We believe that using more than one agent provides a more accurate result.

4.2 User App

A visualization of the main page is presented in Figure 4.8. As depicted, Robot Advisor operates as an RL-based recommender system. Users have the capability to input their personal preferences manually. Shortly thereafter, the agent provides trading recommendations. We remind the reader that this application is designed for short-term trading purposes only. It does not facilitate transactions or conduct live trading; rather, it is a tool designed to offer advisory services to users.

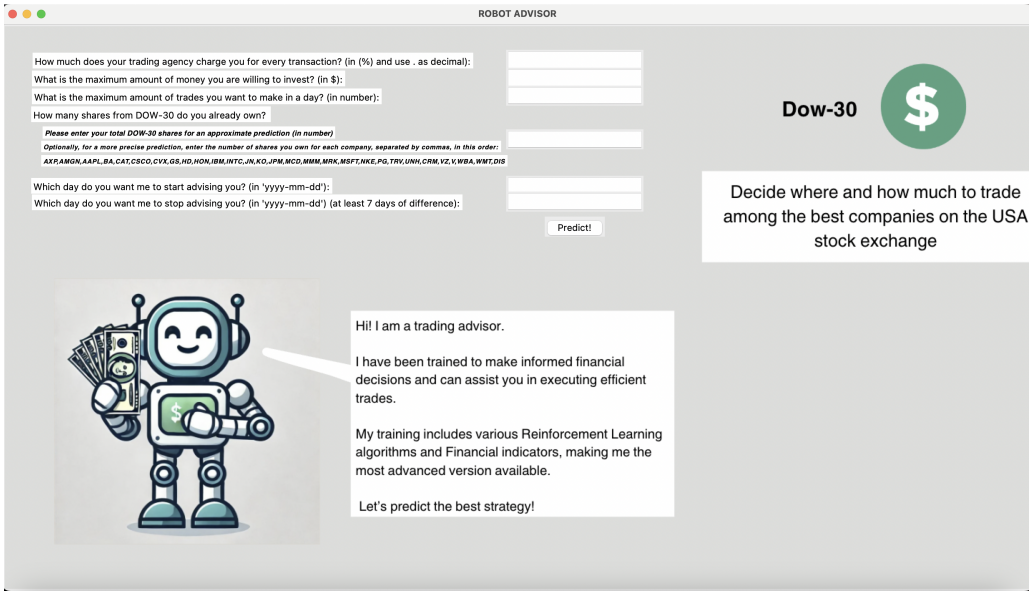
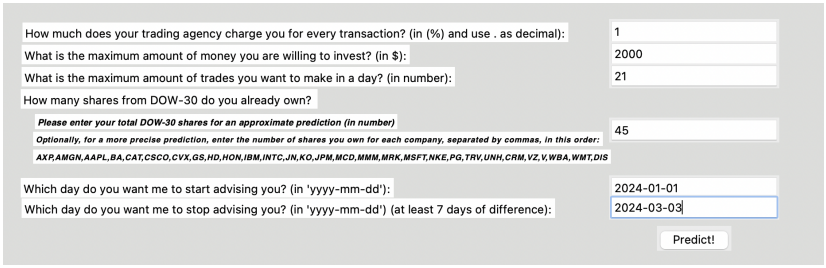


Figure 4.8: Robot App Main Page

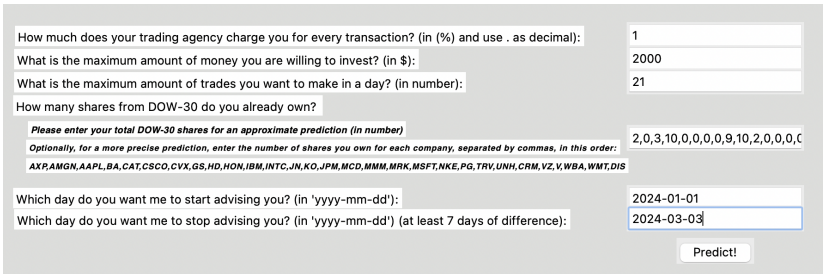
Since we trained the models with data from Dow-30, the app can only give advise within this market index. Nevertheless, further training could be implemented and give the user the possibility of choosing among a wider range of market indexes. Even the possibility of specifically choosing the desired stocks the user wants to get recommendations for could be given. But the same problem arises: different agents would need to be trained. This is an inconvenience we detected during the implementation of this app, and that was already warned in the state of the art chapter. The models are data dependant. The same input size as in the training must be provided when testing or predicting. If the size of utilized companies change, the model changes. This means that in order to adapt to the user's preferences and provide them with more personalized features, more training would be needed.

As illustrated in Figure 4.9, the app prompts the user to enter the trading fee charged by their brokerage, which is considered within the `StockTradingEnv` class of FinRL, as well as the other asked parameters. Additionally, the user is asked to specify the initial amount of money they are prepared to invest. The app also sets a limit on the maximum number of trades per day to ensure the agent does not exceed this threshold, accommodating potential restrictions imposed by the user's trading agency or personal preferences for limiting trade activity.

The user may already possess shares from one or more companies featured in the Dow-30 index. For this, we offer two possibilities. If the user does not want to explicitly indicate how many shares they already own from each stock, they can instead provide a total count of shares they own among companies that pertain to the index (Figure 4.9). In



(a) Total number of stocks held in the Dow 30



(b) Specific number of stocks held in the Dow 30

Figure 4.9: User individual parameters

the opposite case, they can manually specify them in the same entry box as a list (Figure 4.9).

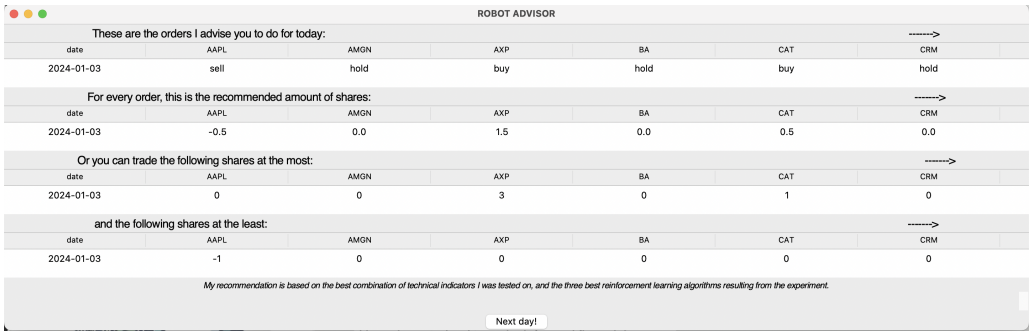
For the advising period, the user must enter a minimum of seven days difference between the start and end date. This is partially due to the fact that the market is closed on the weekends.

The app fetches the data that includes the recommendation period from YahooFinance, as well as historical prices to compute the technical indicators. We remind the reader that the robot takes the information of today (when market has closed) to make a prediction for tomorrow. This means that the user would need to wait another day to get the recommendation for the day after tomorrow, so that tomorrow the agent can fetch again the data. In any case, there is no maximum limit in length for the recommendation period. Moreover, the agent takes into account the change in the conditions as soon as it starts giving recommendations along the desired period. Nevertheless, if any of the initially introduced conditions from the user would change, the use needs to manually modify them and make the prediction again. For instance, if the trading agency rises the transaction fee, the user must insert this data again.

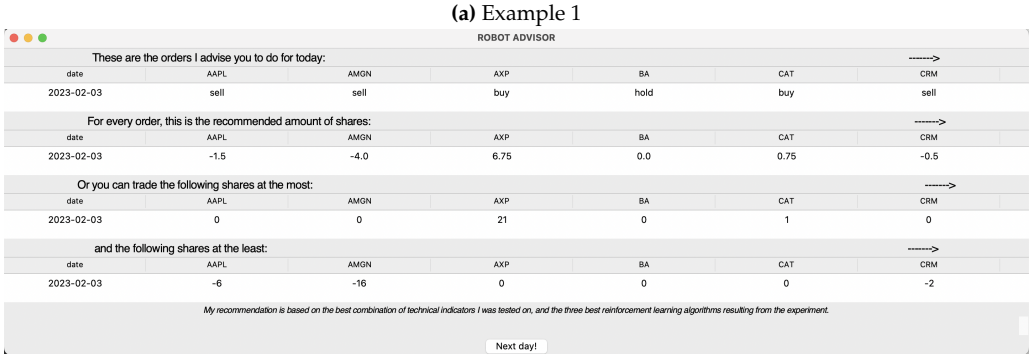
Finally, the agent gives its recommendations. It specifies the trading order for every stock: sell, hold, buy. For each of them, it provides the estimation from combined TD3, PPO and A2C. At first, it gives the mean recommendation for the number of shares to trade. In case the user disagrees, it provides a maximum recommended amount, and a minimum. This is, the maximum and minimum amount of shares any agent recommended. In this way, the user is free to act differently but still has a warning range.

The 'Next day!' button provides the same information for the consequent market date. The user can get recommendations by pressing it until it reaches the end recommendation date initially provided. There is a scrollable element to visualize all the stocks.

As for the user input validation, we offer warning error boxes that allow the user to correct their mistakes without breaking the execution. The error cases covered are: Format (error when introducing negative numbers for the owned shares, start date is



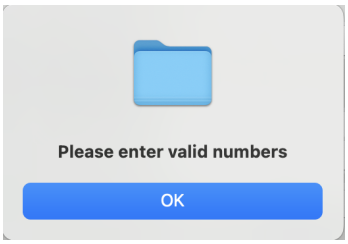
(a) Example 1



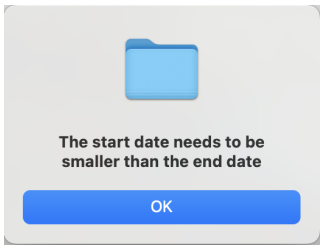
(b) Example 2

Figure 4.10: Robot Advisor Recommendation examples

greater than the end date) and Input error (when introducing letters as numbers in any box). This is shown in Figure 4.11.



(a) Input Error



(b) Date Format Error

Figure 4.11: Robot Advisor Warning Error Boxes

The libraries `pillow` [1], `tkinter` [19] and `FinRL` were used for the implementation of the user app. The whole process of finance data fetching and preprocessing (done with `FeatureEngineer` and `YahooDownloader` classes) needs to be done every time the user hits the predict Button. Then, the user parameters are set to the model and the prediction environment is created. We load he models from the folders in which we saved them. When the predictions are done, the average, maximum and minimum recommended shares to trade are computed. The structure used for displaying the recommendations is a `Treeview` from `tkinter` library.

CHAPTER 5
Conclusions and Future work

During the training and app development we became aware of the inconveniences of the practical implications when using DRL models in finance, as already mentioned in Section 2.3 and discussed by Jiang et al. [26]. We realized that in order to offer more functionalities to the user, more and different model training was needed. We built an app that only gave advice among the companies belonging to Dow-30, but in order to offer the user a personalized selection of firms on which to get advice, a different is needed (with the information about those exact firms as input, as well as individual user information). The same goes for additional features, such as offering advice to invest in other market indexes, or continuing to offer advice on indices after index composition change. In other words, any variation in the definition of states in the MDP needs a new training process, which takes time and energy consumption. Moreover, the financial market is a dynamic environment that evolves with time. Additional training with certain periodicity could improve performance. To sum up, these characteristics of DRL modeling in finance account for flexibility, being able to adapt to varying conditions and dynamics of the stock market. However, the data dependency characteristic makes the models lack in reusability, having to retrain or restructure the model to suit the new conditions and consuming resources. This is especially inconvenient given the fact that these additional offered features (advice on other indexes and stocks) may usually be requested by users.

Nevertheless, for short term investments, which are our study case, the DRL models outperform the natural index performance in our experiments, which are already promising results. Any financial advisor tool has a margin of error and carries a certain risk. Given the beneficial impact that this tool could have on personal or family economies through asset management, we would recommend it as a valid tool.

We have trained several RL agents with different modeling of technical indicators and shown that they can learn how to trade on normal circumstances. We did not prove whether they can predict rare events. We have analysed differences in training parameters such as the reward curve or entropy loss, and differences in performance during the testing. We did not detect significant differences in the performance of different technical indicators, but we detected marginal differences. We used the technical indicators that produced marginally better results to build our user app that is able to give financial advice to users without any programming or financial knowledge.

As for new acquired knowledge during the development of this work, I have learned theoretical RL concepts that I did not know before such as the differences among some RL algorithms like PPO, A2C, DDPG, TD3 and SAC. I have also understood the importance of making RL algorithms parallelizable and the utility of CUDA and TensorFlow for this purpose. The idea of someone implementing an RL environment to simulate a stock market seemed almost impossible to me at the beginning. But I had the opportunity to

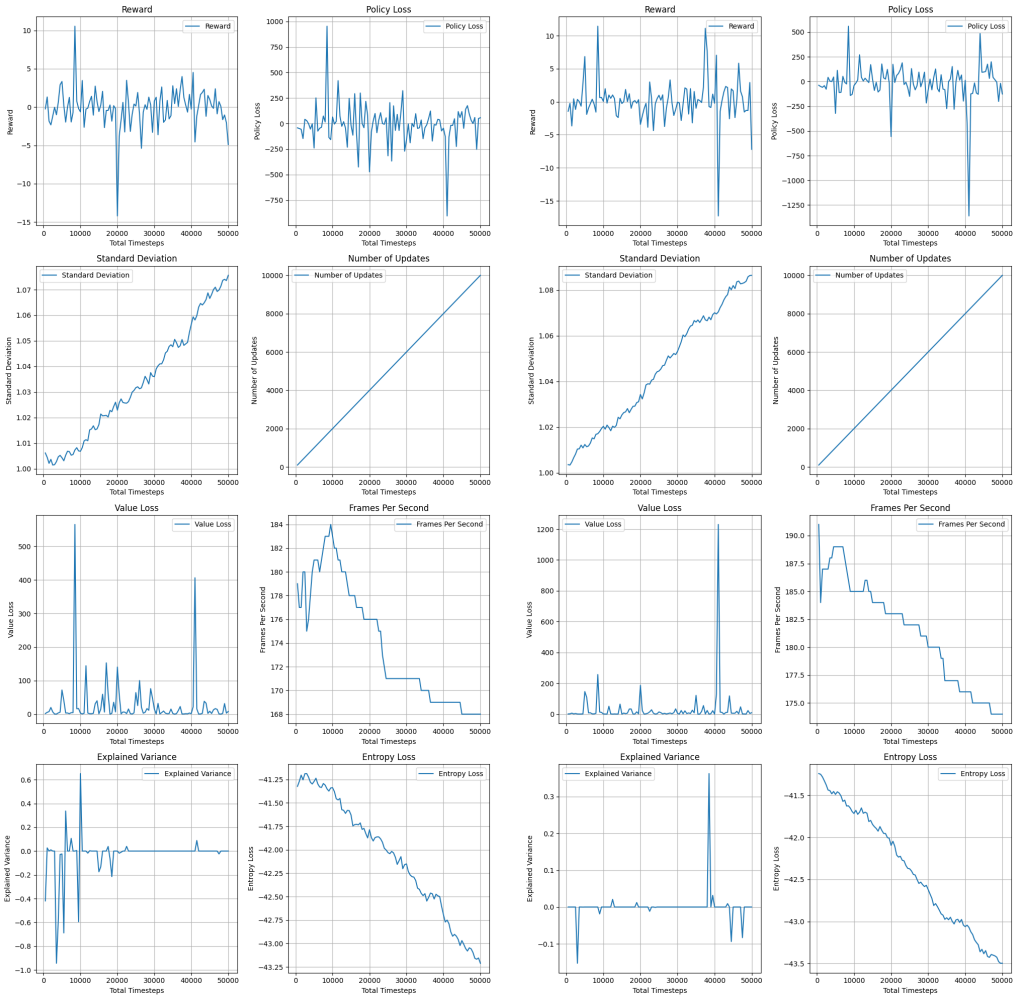
study the low level implementation of FinRL and understood that the use of some data structures make this possible. Many people do not support the use of AI techniques in finance because of its low predictability, but I believe that conducting experiments like this one are beneficial for obtaining conclusions. Therefore, I am glad I could go through this process.

Some encountered problems were due to FinRL poor maintenance. Since it is an open source library, I believe the use of GitHub Issues was key to communicate with different people who developed it or with people that already encountered the same problem. I raised an Issue [3] to inform that some technical indicators were not supported, for which I got an answer from Hongyang (Bruce) Yang, Founder and President of AI4Finance Foundation, and author of the original papers of FinRL. He confirmed its poor maintenance. This also involves poor documentation, as I had to understand the code by diving deeper into the class development rather than by reading the documentation, which took more time.

Future work could include training additional models to provide the previously mentioned features, such as advice across multiple market indexes and customizable firm selection. A second line of future work could be exploring the use of DRL in long term investment. The model could take into account other variables that are measured with a wider granularity than the day. For instance, the dividends a company gives to the stakeholders (which are normally taken into account when deciding where to invest long term, and only distributed monthly at the least). The periodicity between two steps could be yearly instead of daily. A third line for future research (either for long or short term) could be combining the predictions that are input price-based, with predictions that are fed with the information on the news. As mentioned in the 2.3 Section, sentiment analysis on the news is carried out. This information can detect whether the expert's opinions about a company are optimistic or pessimistic. Therefore, aiding on the decision making on where to invest.

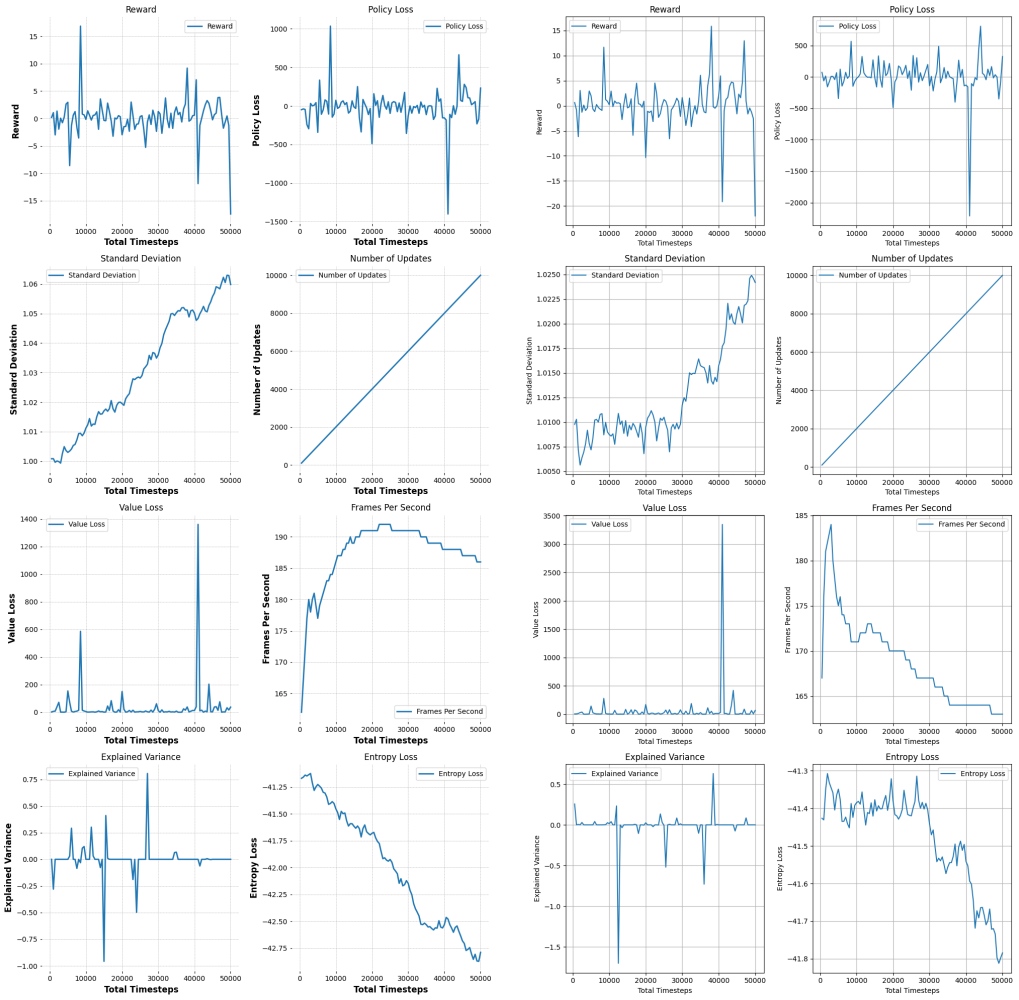
As for other interesting tools that would have been interesting to dive deeper in, AI4Finance has developed other projects. FinGPT is a Large Language Model (LLM) that focuses on providing financial insights and democratizing information, as well as providing researchers resources to develop their financial LLMs [45]. The model can be found at FinGPT [17]. FinRobot is an AI powered platform designed to serve many financial applications. It integrates diverse technologies within its four layers: Financial AI Agents, Financial LLMs Algorithms, LLMOps and DataOps, Multi-source LLM Foundation Models [46]. It is designed to highlitght its adaptability. It can be found at Foundation [18]. These tools started being available when we had already set the definition of this project, so it was not possible to explore them more deeply due to time and complexity constraints.

APPENDIX A
Training Results Statistics



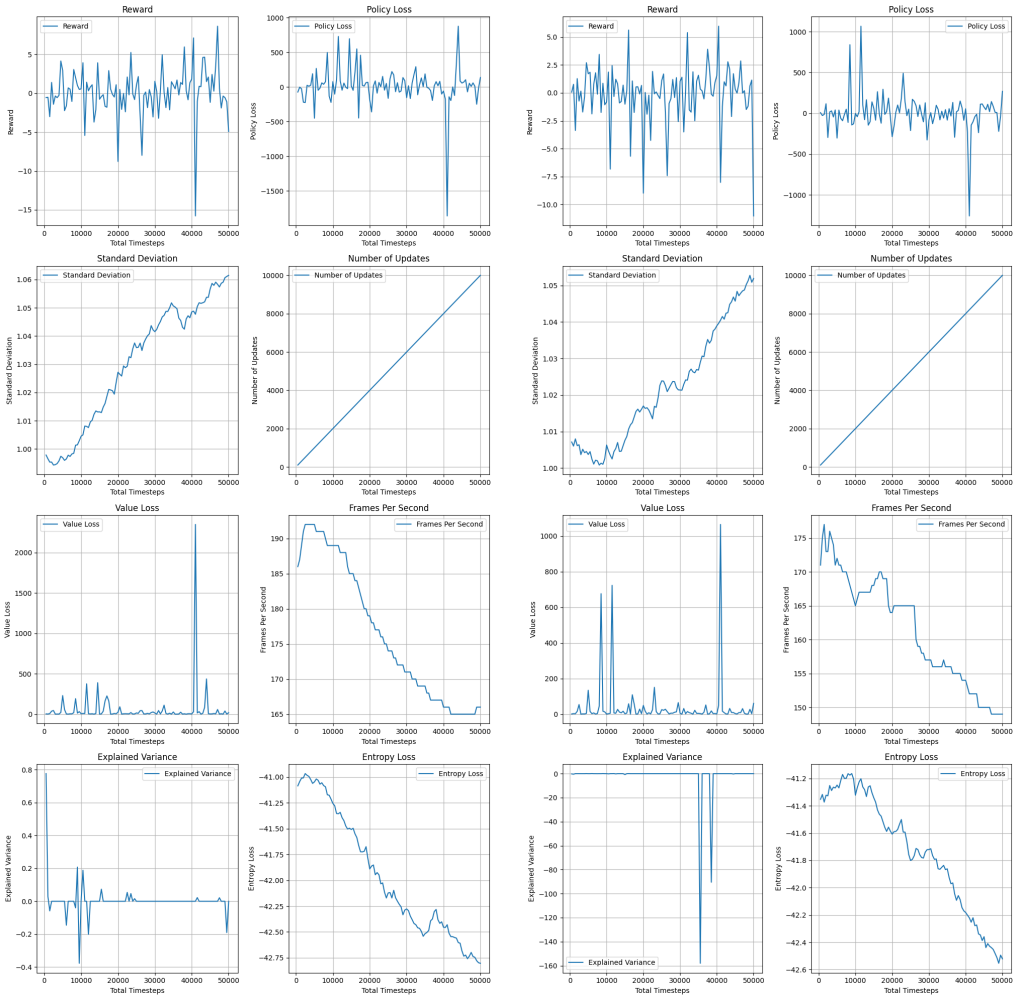
(a) A2C Set 1(-) **(b)** A2C Set 1(+)

Figure A.1: A2C Training Results



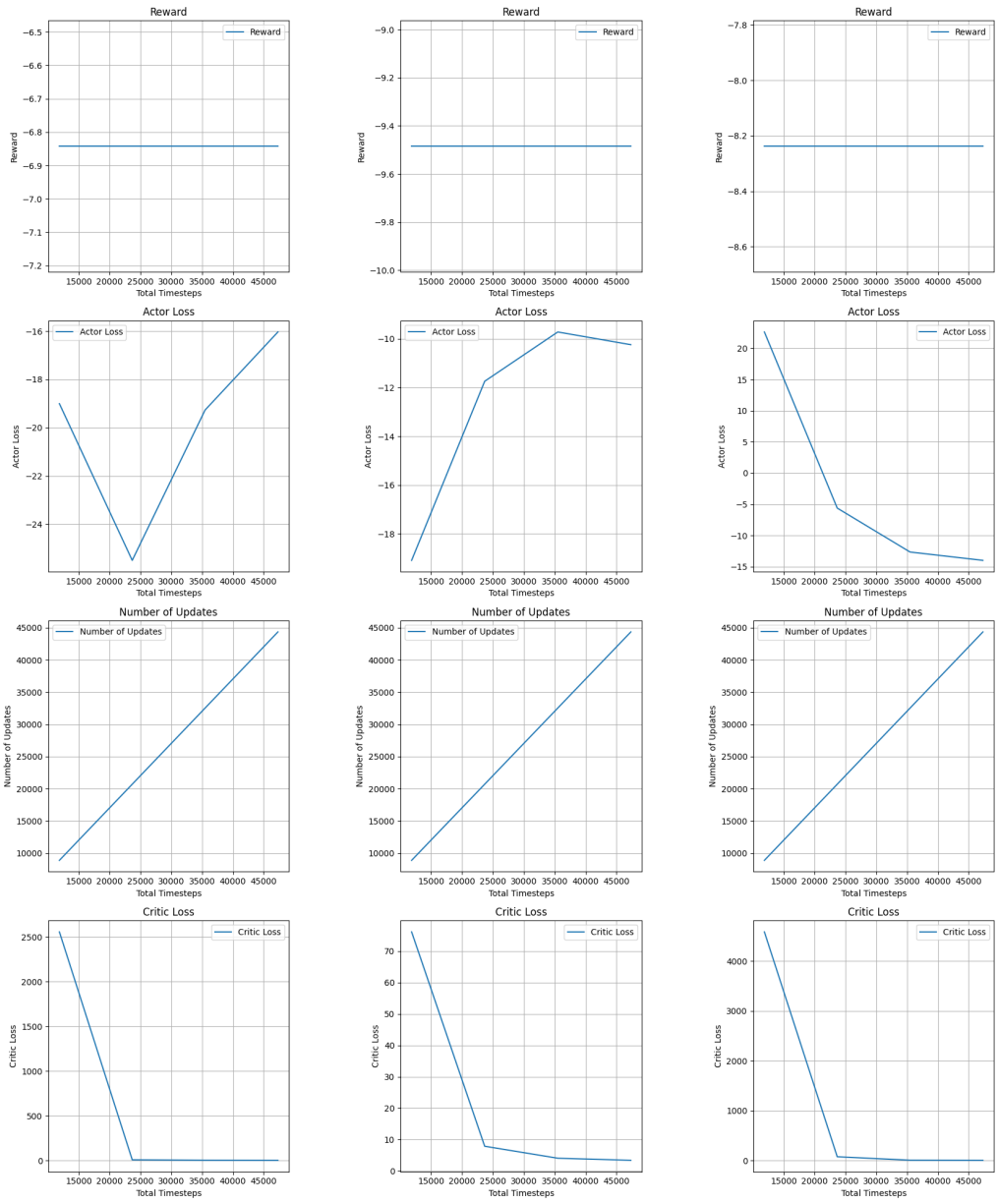
(a) A2C Set 1 **(b)** A2C Set 2

Figure A.2: A2C Training Results



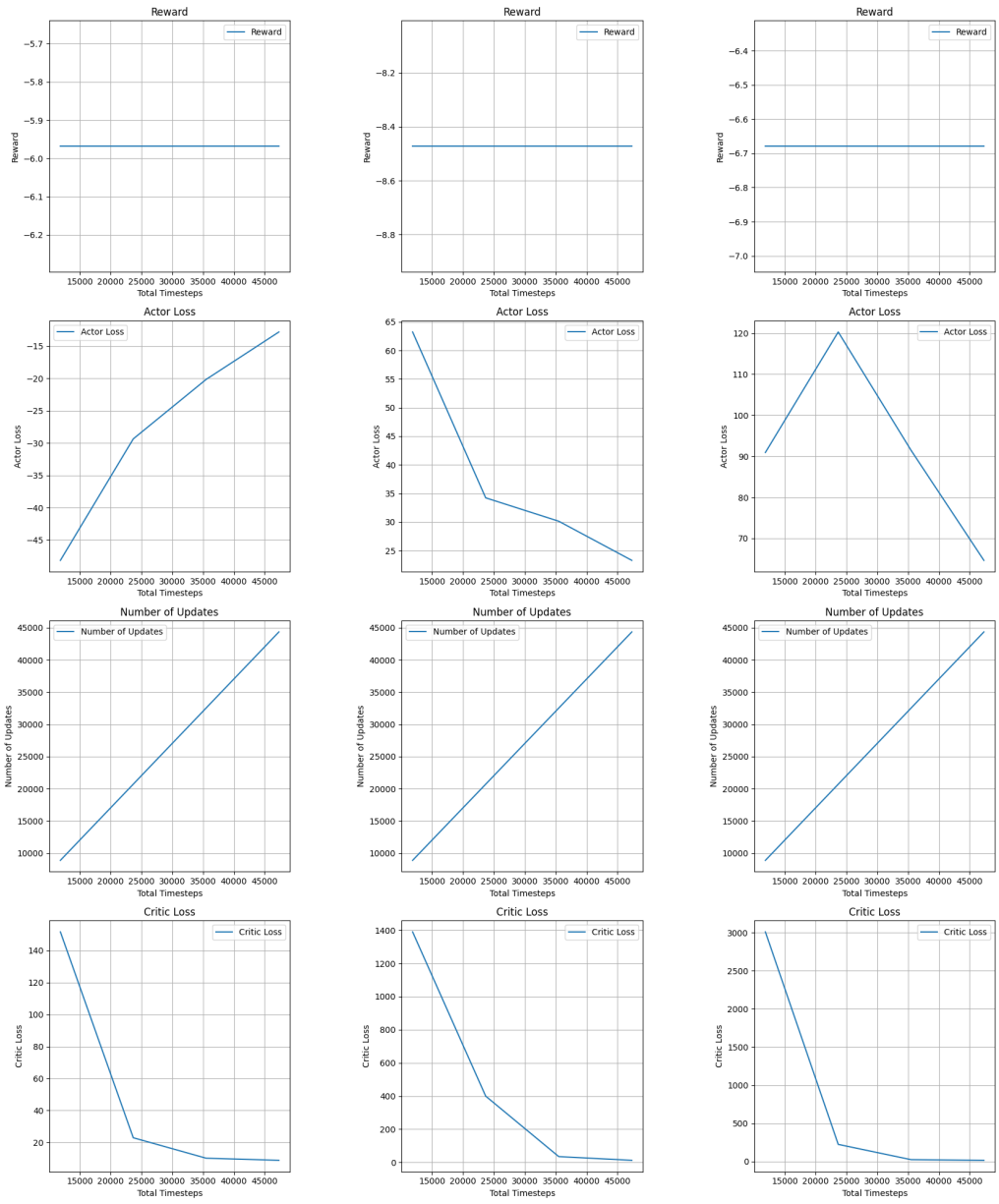
(a) A2C Set 3 **(b)** A2C Set 4

Figure A.3: A2C Training Results



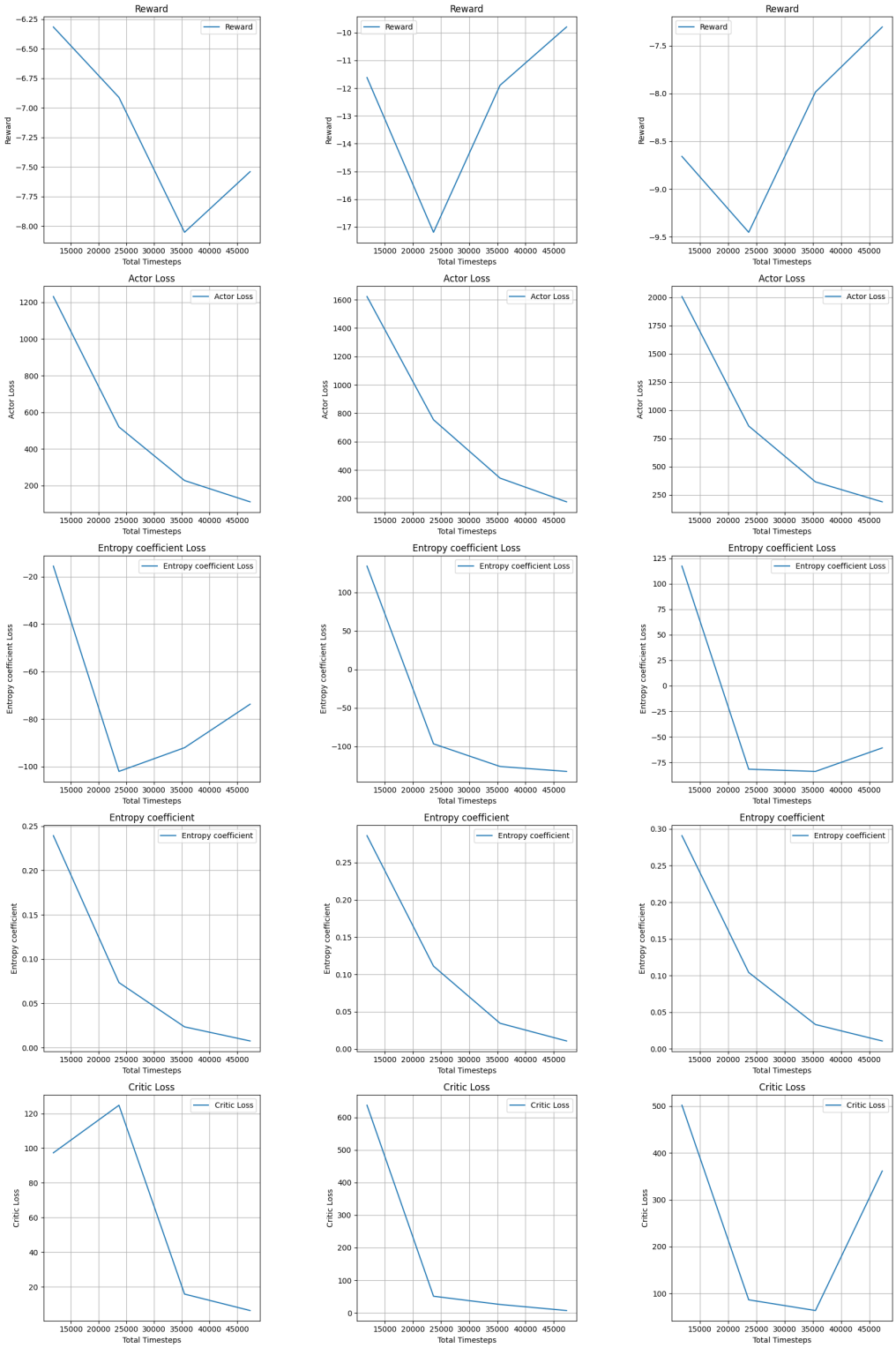
(a) DDPG Set 3 **(b)** DDPG Set 2 **(c)** DDPG Set 5

Figure A.4: DDPG Training Results



(a) TD3 Set 3 **(b)** TD3 Set 2 **(c)** TD3 Set 5

Figure A.5: TD3 Training Results



(a) SAC Set 3 **(b)** SAC Set 2 **(c)** SAC Set 5

Figure A.6: SAC Training Results

(a) PPO Set 1(-) **(b)** PPO Set 1(+)

Figure A.7: PPO Training Results

(a) PPO Set 1 **(b)** PPO Set 2

Figure A.8: PPO Training Results

(a) PPO Set 3 **(b)** PPO Set 4

Figure A.9: PPO Training Results

Bibliography

[1] Pillow: The friendly PIL fork. https://pypi.org/project/pillow/, 2023. Accessed: 2023-06-24.

[2] AI4Finance. Ai4finance, 2024. URL https://ai4finance.org/.

[3] AI4Finance Foundation. Issue 1245: Discussion about Specific Functionality in FinRL. https://github.com/AI4Finance-Foundation/FinRL/issues/1245, 2023.

[4] AI4Finance Foundation. Finrl documentation. https://finrl.readthedocs.io/en/latest/index.html, 2024.

[5] AI4Finance Foundation. Finrl environments documentation. https://finrl.readthedocs.io/en/latest/start/three_layer/environments.html, 2024. Accessed on: June 24, 2024.

[6] AI4Finance Foundation. Finrl three-layer architecture. https://finrl.readthedocs.io/en/latest/start/three_layer.html, 2024.

[7] AI4Finance Foundation. Issue 1245: [describe the issue title here]. https://github.com/AI4Finance-Foundation/FinRL/issues/1245, 2024.

[8] Alpaca. Data scientists' approach to algorithmic trading using deep reinforcement learning. https://alpaca.markets/learn/data-scientists-approach-algorithmic-trading-using-deep-reinforcement-learning/, 2024. Accessed on: June 24, 2024.

[9] AminHP. gym-anytrading: Openai gym environments for trading. https://github.com/AminHP/gym-anytrading, 2024. Accessed on: June 22, 2024.

[10] AminHP. gym-mtsim: Openai gym - metatrader 5 simulator. https://github.com/AminHP/gym-mtsim, 2024.

[11] A. M. Arroyo and M. Prat. *Dirección financiera*. Deusto, 3rd edition, 1996.

[12] Bolsas y Mercados Españoles. Informe de propiedad de las acciones españolas cotizadas, 2023. URL https://www.bolsasymercados.es/docs/BME/docsSubidos/Estudios-Articulos/BME-Informe-Propiedad-Acciones-espanolas-cotizadas.pdf. Accessed: 2023-05-18.

[13] Codecamaru. RobotAdvisor. https://github.com/codecamaru/RobotAdvisor/tree/main, 2024.

[14] Banco de España. Tipos de interés, n.d. URL https://www.bde.es/webbe/es/estadisticas/temas/tipos-interes.html. Retrieved May 15, 2024.

[15] Farama Foundation. Gymnasium: A collection of openai gym environments. https://gymnasium.farama.org, 2024.

[16] Fidelity Investments. Rsi (relative strength index) - technical indicator guide. https://www.fidelity.com/learning-center/trading-investing/technical-analysis/technical-indicator-guide/RSI, 2024.

[17] FinGPT. Fingpt on hugging face, 2024. URL https://huggingface.co/FinGPT. Accessed: 2024-06-18.

[18] AI4Finance Foundation. Finrobot: An open-source ai agent platform for financial applications using large language models. https://github.com/AI4Finance-Foundation/FinRobot, 2024.

[19] Python Software Foundation. Tkinter — python interface to tcl/tk, 2023. URL https://docs.python.org/es/3/library/tkinter.html. Accessed: 2023-06-24.

[20] Global Goals. Goal 8: Decent work and economic growth. https://www.globalgoals.org/goals/8-decent-work-and-economic-growth/, 2024.

[21] Hugging Face Team. Two types of value-based methods. Hugging Face's Deep RL Course, 2023. URL https://huggingface.co/learn/deep-rl-course/unit2/two-types-value-based-methods. Accessed: 2024-06-01.

[22] H.D. Huynh, L.M. Dang, and D. Duong. A new model for stock price movements prediction using deep neural network. In *Proceedings of the Eighth International Symposium on Information and Communication Technology*, pages 57–62. ACM, 2017.

[23] Rob J. Hyndman. Moving averages, November 2009.

[24] Investopedia. Investopedia: Sharper insight, better investing. https://www.investopedia.com, 2024.

[25] Zhengyao Jiang, Dixing Xu, and Jinjun Liang. A deep reinforcement learning framework for the financial portfolio management problem. *arXiv:1706.10059 [q-fin.CP]*, 2017. Available at arXiv: https://arxiv.org/abs/1706.10059.

[26] Zhiyuan Jiang, Dexian Zhang, and Jun Luo. Using deep reinforcement learning for algorithmic trading. *Journal of Financial Markets*, 48:107–128, 2020.

[27] Yuxi Li. Financial trading as a game: A deep reinforcement learning approach. *Journal of Financial Data Science*, 1(1):42–59, 2019.

[28] MetaQuotes Software Corp. Metatrader 5 trading platform. https://www.metatrader5.com/es/trading-platform, 2024.

[29] United Nations. Economic growth. United Nations website, n.d. Available online: https://www.un.org/sustainabledevelopment/economic-growth/ [Accessed on: 10/06/2024].

[30] North Dakota State University. The North Dakota Horizons of Focus: A Comprehensive Statewide Plan for the Retention and Recruitment of Qualified Individuals with Disabilities, 2009. URL https://library.ndsu.edu/ir/bitstream/handle/10365/9322/fe258_2009.pdf?sequence=1. Retrieved [Insert Retrieval Date Here].

[31] OpenAI. Deep deterministic policy gradient (ddpg). OpenAI Spinning Up Documentation, 2023. URL https://spinningup.openai.com/en/latest/algorithms/ddpg.html. Accessed: 2024-06-01.

[32] OpenAI. Gym: A toolkit for developing and comparing reinforcement learning algorithms. https://github.com/openai/gym, 2024.

[33] David Silver. Introduction to reinforcement learning, 2020. URL https://www.davidsilver.uk/wp-content/uploads/2020/03/intro_RL.pdf. Accessed: 2023-09-05.

[34] Stable Baselines3 Team. Stable baselines3 documentation. https://stable-baselines3.readthedocs.io/en/master/, 2024. Accessed on: June 24, 2024.

[35] StockCharts.com. Introduction to technical indicators and oscillators. https://school.stockcharts.com/doku.php?id=technical_indicators:introduction_to_technical_indicators_and_oscillators, Accessed: 2024.

[36] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018.

[37] Csaba Szepesvári. *Algorithms for Reinforcement Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan and Claypool Publishers, San Rafael, California, 2010.

[38] Hugging Face Team. Deep rl: A2c, 2021. URL https://huggingface.co/blog/deep-rl-a2c. Accessed: 2024-06-01.

[39] Hugging Face Team. Intuition behind ppo. Hugging Face's Deep RL Course, 2023. URL https://huggingface.co/learn/deep-rl-course/unit8/intuition-behind-ppo. Accessed: 2024-06-01.

[40] Hugging Face Team. Mc vs td: Understanding the differences. Hugging Face's Deep RL Course, 2023. URL https://huggingface.co/learn/deep-rl-course/unit2/mc-vs-td. Accessed: 2024-06-01.

[41] OpenAI Team. Soft actor-critic (sac). OpenAI Spinning Up Documentation, 2023. URL https://spinningup.openai.com/en/latest/algorithms/sac.html. Accessed: 2024-06-01.

[42] OpenAI Team. Twin delayed ddpg (td3). OpenAI Spinning Up Documentation, 2023. URL https://spinningup.openai.com/en/latest/algorithms/td3.html. Accessed: 2024-06-01.

[43] Chen Hao Wang Jianye Troiano Luigi Loia Vincenzo Wu, Xue and Hamido Fujita. Adaptive stock trading strategies with deep reinforcement learning methods. *Information Sciences*, 538:142–158, 2020.

[44] Yahoo Finance. Yahoo finance, 2024. URL https://finance.yahoo.com/?guccounter=1&guce_referrer=aHR0cHM6Ly93d3cuZ29vZ2xlLmNvbS8&guce_referrer_sig=AQAAAIiCbhvZzsaol08M-zlec231DC3Z5DKCCxH7xCZYCCu9FkZ7euwl7JWmOE5xCvHxvhPYV61YeJ62IM4x7lHBuBf-p0ZcLT3RY157XIElgzHwu2w7HAwSrWW-3QHkzMdMDuabcgnRcMtzsOSNIOQ4gfRruhfVfZ9yCDomxpN. Accessed: 2024-02-31.

[45] Hongyang (Bruce) Yang, Xiao-Yang Liu, and Christina Dan Wang. Fingpt: Open-source financial large language models, 2023. URL https://arxiv.org/pdf/2306.06031. Columbia University and New York University (Shanghai).

[46] Hongyang (Bruce) Yang, Boyu Zhang, Neng Wang, Cheng Guo, Xiaoli Zhang, Likun Lin, Junlin (Jason) Wang, Tianyu Zhou, Mao Guan, Runjia (Luna) Zhang, and Christina Dan Wang. Finrobot: An open-source ai agent platform for financial applications using large language models. https://huggingface.co/FinGPT, 2024. URL https://huggingface.co/FinGPT. AI4Finance Foundation, Columbia University, NYU Shanghai, Shanghai Frontiers Science Center of Artificial Intelligence and Deep Learning, Business Division, Shanghai AI-Finance School ECNU.

[47] Yuxi Yang, Yinchuan Liu, Ziyi Wang, and Hongyang Zhang. Deep reinforcement learning for automated stock trading: An ensemble strategy. *SSRN Electronic Journal*, 2020. doi: 10.2139/ssrn.3690996. Available at SSRN: https://ssrn.com/abstract=3690996.

[48] S. Yu and Z. Li. *Forecasting Stock Price Index Volatility with LSTM Deep Neural Network*, pages 265–272. Springer, 2018.

APPENDIX: Sustainable Development Goals

This work is connected to several aspects of the United Nations Sustainable Development Goals (SDGs), particularly those related to reducing inequality, promoting sustainable economic growth, and fostering innovation.

Degree of relationship of the work with the SDGs:

Sustainable Development Goals	**High**	**Medium**	**Low**	**Unrelated**
SDG 1. **No poverty.**			X	
SDG 2. **Zero hunger.**				X
SDG 3. **Good health and well-being.**				X
SDG 4. **Quality education.**				X
SDG 5. **Gender equality.**			X	
SDG 6. **Clean water and sanitation.**				X
SDG 7. **Affordable and clean energy.**				X
SDG 8. **Decent work and economic growth.**	X			
SDG 9. **Industry, innovation and infrastructure.**			X	
SDG 10. **Reduced inequalities.**	X			
SDG 11. **Sustainable cities and communities.**				X
SDG 12. **Responsible consumption and production.**				X
SDG 13. **Climate action.**				X
SDG 14. **Life below water.**				X
SDG 15. **Life on land.**				X
SDG 16. **Peace, justice and strong institutions.**				X
SDG 17. **Partnership for the goals.**				X

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

ETS Enginyeria Informàtica
Camí de Vera, s/n, 46022, València
T +34 963 877 210
F +34 963 877 219
etsinf@upvnet.upv.es - www.inf.upv.es

Reflection on the relationship of the Degree Final Work with the SDGs:

The SDG that is related the most with this work is **Goal 8**: 'Decent Work and Economic Growth'. The primary aim of SDG 8 is to promote sustained, inclusive, and sustainable economic growth, full and productive employment, and decent work for all. The financial advisor agent that we developed in this project, directly contributes to this goal by democratizing access to financial markets. By providing individuals, families, and firms with the capabilities to make informed trading decisions without any prior knowledge, this tool increases their economic opportunities. It facilitates greater participation in financial activities. This democratization of financial services can lead to more equitable economic growth, where more people have the opportunity to improve their economic status, thereby contributing to the overall health of the global economy.

SDG 10 aims to reduce inequality within and among countries. By offering access to the proposed tool to the whole population, we can help to contribute to the national economies, and reducing not only national financial access inequality, but also reducing the economic inequalities among countries. This access is particularly crucial for people in developing countries or marginalized communities in developed nations, where access to financial services and literacy is often limited. By improving financial inclusivity, the tool aids in reducing economic disparities and enhancing the financial autonomy of individuals across different socio-economic backgrounds. According to the United Nations, one way of aiding in this goal and ensure equal opportunities is by eliminating discriminatory practices. Our tool does not discriminate any gender or nationality, which indirectly contributes to **SDG number 5**.

SDG 9 focuses on building resilient infrastructure, promoting inclusive and sustainable industrialization, and fostering innovation. Without a doubt, our app represents an innovation in financial technology, even though it is not the only app to serve this purpose, DRL applied to stock trading is definitely an untapped field. This tool enhances the financial services infrastructure by integrating cutting-edge AI into everyday financial activities. The project contributes to the infrastructure of digital financial services, making them more robust and capable of handling complex, data-driven decision-making processes.

According to the official website of the united nations, one of the ways of aiding in the eradication of poverty (**first SDG**) is fostering innovation and critical thinking in all the ages. It is true that in order to invest, someone initially needs money. But for short term investing, the needed quantities are not huge, and with the right agency, getting profitability is possible. So, in this way, we would also be contributing to this SDG.

ETS Enginyeria Informàtica
Camí de Vera, s/n, 46022, València
T +34 963 877 210
F +34 963 877 219
etsinf@upvnet.upv.es - www.inf.upv.es