**Applied Industry Project : Stock Trading Bot**

**Group A3 – [Br@in]**

Benny, Abin

Birru, Shivaganesh

Bhatt, Jaldhi Himanshu

Duarte Junior, Fabio

Patel, Manali Shaileshkumar

Applied Industry Project – 2024, Loyalist College in Toronto

AIP Project Advisor: Mr. Usman Ahmad

AIP Coordinator: Mr. Stanley Chor

April 08, 2024

# Abstract

Stock trading involves the movement of trillions of dollars being invested daily with millions of decisions made by both humans and automation systems. The competitive landscape is extremely high, and traditional automation tools serve merely as triggers for user-predefined decisions. Thus, based on this scenario, we were challenged by the college to build a Stock Trading Bot based on Deep Reinforcement Learning techniques (referred to in this document as DRL), a solution aimed at modernizing existing technology. It would be capable of automating high-frequency trading using state-of-the-art technology, reducing risks, and increasing returns, with the goal of surpassing current technologies. In conducting a study on market pain points and how AI has been applied in other sectors, parallel discussions also raised questions such as "Are AI solutions, like trading automations, increasing the gap between the rich and the poor?". Therefore, we decided that our solution should not only modernize current solutions but also be accessible by reducing the complexity of the trading process to democratize access to our platform. Our project was based, for the most part, on the pre-existing work of the AI4Finance-Foundation through its research, papers, and Python libraries made available in its open repository. This details our approach, focusing on technological innovation and the democratization of access to AI, based on the FinRL, FinRL-Meta, and FinGPT libraries to create not just a stock trading bot but a more comprehensive trading platform based on AI and cloud technologies.

Table of Contents

[Abstract 2](#_Toc163552816)

[Introduction 4](#_Toc163552817)

[Literature Review 6](#_Toc163552818)

[Stock Market domain and available technologies. 6](#_Toc163552819)

[Regarding insights and compliance. 9](#_Toc163552820)

[Methods 10](#_Toc163552821)

[FinRL and Deep Reinforcement Learning (DRL) 10](#_Toc163552822)

[Exploring Libraries 11](#_Toc163552823)

[FinRL and FinRL-Meta 11](#_Toc163552824)

[ElegantRL 12](#_Toc163552825)

[Deep Q-learning and other approaches from scratch 12](#_Toc163552826)

[FinGPT 13](#_Toc163552827)

[Deep Reinforcement Learning Algorithms 14](#_Toc163552828)

[Available algorithms and their characteristics 14](#_Toc163552829)

[Available Datasets 15](#_Toc163552830)

[Testing FinRL 16](#_Toc163552831)

[Training and testing the agents: 17](#_Toc163552832)

[A2C (Advantage Actor-Critic) 18](#_Toc163552833)

[DDPG (Deep Deterministic Policy Gradient) 18](#_Toc163552834)

[PPO (Proximal Policy Optimization) 19](#_Toc163552835)

[TD3 (Twin Delayed Deep Deterministic policy gradient) 19](#_Toc163552836)

[SAC (Soft Actor-Critic) 20](#_Toc163552837)

[Trading (testing) 20](#_Toc163552838)

[Architecture 22](#_Toc163552839)

[BrainLib 24](#_Toc163552840)

[Batch Process 25](#_Toc163552841)

[Findings 28](#_Toc163552842)

[Hidden Overfitting 28](#_Toc163552843)

[FinRL Installation and Alternatives for CUDA 29](#_Toc163552844)

[Creation of our own version of FinRL : FinRL-Mini 31](#_Toc163552845)

[Discussion: 34](#_Toc163552846)

[Conclusion: 34](#_Toc163552847)

[Recommendation: 35](#_Toc163552848)

[References: 35](#_Toc163552849)

[Appendices: 35](#_Toc163552850)

# Introduction

The stock market is highly sought after as an option for asset generation, especially in the current economic situation. For example, research conducted by the website Finder shows that 26% of Canadians turn to the stock market for long-term savings purposes between 2022 and 2023. (King, Romana (2023). Finder. Statistics and Facts about the Stock Market. Retrieved from https://www.finder.com/ca/stock-trading/stock-trading-statistics#source) On the other hand, the same research indicates that 37% of Canadians lack confidence in the trading market, which is understandable given the volatility and inherent risk of this niche, as well as the amount of financial education needed, an overload of information, and the necessity to closely follow every change happening through news related to target companies or via information providers. In summary, there are many opportunities for expansion in this market if the trading process can be somehow simplified and/or made more accessible to the general public, thus also easing life.

To achieve this goal, technology plays an essential role, especially when we talk about optimizing strategies in the complex and dynamic context of the Stock Market. Thus, guided by our Project Advisor, we explored the use of AI techniques based on the Deep Reinforcement Learning approach, inspired by the work of the AI4Finance-Foundation, to create an automation capable of performing stock trading, or as the Project Charter defines, “high-frequency trading.”

Nevertheless, as already mentioned, our objective is not only to automate these operations but also to democratize the use of these technologies and the stock trading market. Therefore, we attempted to wrap around our solution in a platform with an easy-to-use interface and features that assist and instruct the user to optimize their experience and results when using our solution.

To achieve our goals, we utilized the following technologies:

* **Markov Decision Process** (MDP) through Deep Reinforcement Learning. (For the stock trading bot)
* **Natural Language Processing** (NLP), through the RAG (Retrieval Augmented Generation) technique, which combines simpler NLP techniques to identify user intent with APIs of Large Language Models (LLM) for user-friendly responses (For the Chatbot Aka AiKnowledgeHub)
* **FinRL & FinRL Meta**, a Python library made available by the AI4 finance team, used to train, test, and as a basis for creating simulated trading environments.

We also used some free APIs and external data as follows:

* **Alpaca API** – For news and real-time stock prices and Exchanges.
* **Finnhub API** – For obtaining news data and financial indicators of symbols for the batch process that populates the database.
* **Cohere API** – API and library for accessing the LLM model “Command,” to be used by the chatbot.

The following cloud technologies were also used to execute our solution:

* **Firebase** – Part of the Google Cloud Platform, responsible for database authentication (Firestore).
* **AWS Cloud** – We used a free tier EC2 server to run our chatbot and our simulation feature based on historical data.
* **Vercel** – We used Vercel, in the free tier, to execute our front-end. Vercel is a cloud platform with deployment integrated with GitHub, which greatly facilitated our work.
* **GitHub** – Used as a repository for our project with deployment integrated with the Vercel platform.
* **Ngrok** – This is an HTTPS “tunnel” used to convert our HTTP application to HTTPS for free and with few restrictions. This tool provides a fixed domain and an HTTPS tunnel application that can link any flow from a local server to the HTTPS port of your domain.
* **Yahoo! Finance** – We used data extracted from this platform to train our DRL model and to execute simulation functionalities based on historical data and trading simulation. The data used are the symbols of the Dow30, a “portfolio” of the top 30 symbols selected as a market reference by Dow Jones.

# Literature Review

## Stock Market domain and available technologies.

As previously mentioned, our project was inspired by the work of the AI4Finance team, primarily through the FinRL, FinRL-meta, and FinGPT libraries. However, before initiating any development or solution design, we needed to deepen our understanding of the market. For this purpose, we utilized the Investopedia platform. Investopedia serves as a database of articles about the financial market at large and is respected by numerous entities such as the Gramercy Institute, SABEW, and the Financial Communications Society, having received awards from these entities in 2020, 2021, and 2022. Its articles are also cited in the financial sections of major news portals like NBC News, CNBC, and Yahoo Finance.

Given its undeniable credibility, editorial standards, fact-checking, and many other mechanisms to keep information as reliable as possible, we used it for research on understanding the business, market volatility, and specific terms of this domain. We observed that the information obtained from the mentioned sources and the way technology is being approached are in resonance with what is currently being used in the market. For example, the use of the volatility indicator VIX as a tool to measure market sentiment and future volatility. We note that the implementation of AI4Finance papers uses this type of approach for paper-trading model training. According to our interpretation, these data demonstrate that one of the market's major concerns is volatility, and various strategies and indices have been created, such as the VIX, on how to address this issue, which is reinforced by the technical papers researched. It's worth mentioning that we also spent extensive time conducting tests and following the various tutorials available on GitHub and the official page of the FinRL library.

For a broader view, we also consulted research sources that specifically demonstrated how people who are not considered professional traders view the market. Our research team discovered some data on this in the research conducted by the award-winning personal finance writer and real estate expert, Romana King, where she summarizes data from other research conducted between 2022 and 2023, stating relevant points like:

- "41% — Of [Canadian] investors do not invest in the stock market".

- "1 in 4 — Canadians were not confident in the stock market in 2022".

To shape our technical design approach, we studied the AI4Finance team's papers from 2020, 2022, and 2023 (they mention 2024, but the publication date is in 2023). They explain the need for more advanced techniques to optimize trading strategies even in a dynamic environment. Besides discussions on the subject, they present a solution, actually a set of solutions encapsulated in the form of two libraries with complete tools for historical data manipulation, DRL agent training, and various simulated trading environments, some with integrations to external brokers' APIs like Alpaca. To date, they maintain and update these libraries through a community with the principle of bringing state-of-the-art AI solutions to the financial market.

In summary, the researched data seem to indicate that trading market professionals are always looking for ways to mitigate the problems caused by market volatility. Investors, both professional and Canadian respondents in the surveys, are fearful or somewhat reluctant to invest in stock trading. On the other hand, there are studies, research, and a community leveraging modern technology for financial purposes addressing the same problems mentioned and maintaining an open-source community. Unfortunately, within the time constraints we had for research, we did not find a viewpoint in the literature that could contrast with what we found during this phase to have a more sober view on the topic. However, given that the use of AI4 Finance solutions was a project requirement and the research aligns with what they publish in their papers, we understood this to be sufficient to validate and clarify the directions and forms the project should take.

Thus, we combined the knowledge from these market researches with the available technical knowledge on state-of-the-art (SOTA) AI technology to design a system that was simple to use, yet powerful, but could be executed within the constraints of our course, such as not having financial or technological support, having to use our own resources and freely available resources.

## Regarding insights and compliance.

Research was conducted, primarily based on the IIROC (Investment Industry Regulatory Organization of Canada), focused on specific regulations for the use of AI and automation in Canada.

Here are the most important points produced by the research

**Automated Order System Definition**: System generating or transmitting orders automatically on a pre-determined basis. Encompasses hardware and software, including smart order routers and trading algorithms.

**Market Impact Risks**: High-speed order transmission can quickly impact markets if issues arise. Marketplace participants are accountable for automated order system use, requiring prevention of interference with fair and orderly markets.

**Regulatory Mandates**: Marketplace participants must take reasonable steps to prevent interference. General understanding and annual testing of automated order systems are required.

**Confidentiality Considerations**: Acknowledgment of confidential client information but emphasizes the need for a sufficient level of knowledge to manage risks.

**Fair and Orderly Markets Support**: Provisions aim to support fair and orderly market functioning upon deploying smart order routers, trading algorithms, or other automated order system aspects.

**Risk Mitigation Controls**: Requirement for controls, including a "kill switch," to disable malfunctioning automated order systems promptly.

**Essential Market Functioning**: Recognition of the necessity of controls in mitigating risks posed by automated order systems to market functioning.

# Methods

## FinRL and Deep Reinforcement Learning (DRL)

The cornerstone of our project is the use of FinRL and its other components for the training, testing, and trading process of the DRL agents that are responsible for the trading transactions.

For this purpose, we based our research primarily on the paper by Liu, X.-Y., et al. (2022), "FinRL-Meta: Market environments and benchmarks for data-driven financial reinforcement learning," presented at NeurIPS, and on the official web documentation of the respective software/libraries.

Through this, we evaluated potential Agents, environments, and libraries.

## Exploring Libraries

### FinRL and FinRL-Meta

These paired open-source libraries provide both a pipeline for creating Reinforcement Learning models and allow the creation of a diverse training environment with a variety of data sources and integration with other libraries. Although the documentation and the library seem a bit outdated, with broken links in the 2021 documentation (see the references section) and version conflicts during installation, we were able to install and test their capabilities. Currently, they seem to be unified, which reduces implementation complexity. After testing and reviewing the papers and documentation, we can conclude that it is a very powerful and flexible library, suitable for our purposes. We will refer to them as one entity throughout the rest of the document. The documentation for the library includes guidelines on how to conduct the installation on AWS Cloud environment and docker image.

It is divided into three layers, transparent to the implementer, abstracting much of the complex training process.

* **Data Layer:** It essentially provides functions to load the data it is integrated with, as well as for data cleaning and feature engineering.
* **Environment Layer**: It creates market environments in the standard used by OpenAI Gym, incorporating specific techniques to simulate real-world markets.
* **Agent Layer**: This is where the DRL algorithms are available, provided through an internal implementation of the ElegantRL library. The algorithms cater to various types of applications, such as stock trading and portfolio management, each with its characteristics, advantages, and disadvantages. However, we tested some using the tutorials provided in the documentation and saw promising results.

ElegantRL

Offers open-source DRL algorithms suitable for both continuous and discrete actions, as well as for multi-agent reinforcement learning scenarios. We indirectly tested some agents from this library via FinRL; however, ElegantRL stands out as a promising option should we choose an environment other than FinRL, such as OpenAI Gym. One notable advantage is that, unlike FinRL which is TensorFlow-based, ElegantRL operates on PyTorch, compatible with our available GPU hardware. Moreover, ElegantRL has minimal dependencies, requiring only Python version higher than 3.6 and PyTorch above 1.0.2, making it more manageable. Therefore, if compatibility issues arise with FinRL, we can utilize ElegantRL independently.

### Deep Q-learning and other approaches from scratch

We came across projects like the trading-bot by Prabhsimran Singh (refer to the reference section), which implements the Q-learning algorithm from the ground up. Although it's not a library by itself, it acts as a benchmark for building from scratch. Nevertheless, we think that replicating such an algorithm would be redundant since ElegantRL already offers an optimized DQN agent with support from the open-source community. Opting for an already established algorithm appears to be more beneficial than developing our own within this constrained timeframe. We also found another intriguing example, built from scratch, named stock-trading-bot by Aditya Oberai, which employs a web scraper to fetch data from Yahoo and integrates a news API connection to perform sentiment analysis for making buy and sell decisions. While this approach is interesting, it does not use DRL, which may not align well with our project's expectations.

### FinGPT

FinGPT is an open-source framework designed for Financial Forecasting and Advice, leveraging Natural Language Processing (NLP), specifically Large Language Models (LLMs). Within the scope of our project, it could serve as a Robo-Advisor, aid in Portfolio Optimization, and perform Financial Sentiment Analysis. The tool can be trained out-of-the-box using LLaMA or ChatGLM algorithms, but it also offers the flexibility to integrate with other LLMs, like ChatGPT, via APIs. It utilizes data from news, trends, and social media to create a repository for sentiment analysis. We tested the Robo-Advisor implementation and, although the results were impressive (it searches for news about a company or ticker, then makes a forecast and provides investment advice based on sentiment analysis), the process is time-consuming. This leads us to believe that we might lack the computational resources required to run these more advanced technologies efficiently. We also began reviewing the related paper, Yang, H. (Bruce), et al. (2023). FinGPT: Open-Source Financial Large Language Models. https://doi.org/10.48550/arXiv.2306.06031, but have not yet completed our research and understanding of its features. There are still questions regarding the computational resources needed to achieve satisfactory results using this library and if we have pre-trained models available (or if we need to train and generate the “data lake” all from scratch).

## Deep Reinforcement Learning Algorithms

### Available algorithms and their characteristics

A table with text on it

Description automatically generated

*FinRL. (2021). DRL Agents. Retrieved January 28, 2024, from* [*https://finrl.readthedocs.io/en/latest/start/three\_layer/agents.html#elegantrl-drl-library*](https://finrl.readthedocs.io/en/latest/start/three_layer/agents.html#elegantrl-drl-library)

Within the FinRL documentation itself, we found definitions and advice on top state-of-the-art DRL Agents, and to our surprise, the library mentioned already has an integrated implementation of the ElegantRL library and its algorithms, which can be easily implemented. We chose three models to test, using the default configurations from the documentation (refer to the "references" section) with an initial budget of $1000000.00.

The training process took six minutes (on average) for 10 “episodes”. Using this basic configuration, we achieve a very positive result in the training PPO, DDPG, and SAC algorithms.  
With no tunning and a few “episodes” of training over a ten-year time frame on Yahoo Financial data, we get a risk ratio, observed here as Sharpe ratio, within what would be considered "good" in terms of performance.

## Available Datasets

Guided by the FinRL paper, we reviewed the available datasets and chose to use Yahoo Finance for its ease of use and availability. We also encountered the possibility of using an API service called Alpaca, which promises to provide financial data for free for educational purposes and solution development, though we haven't thoroughly explored this option yet. Here is the list of dataset options available within the FinRL environment for reference, but we will primarily be working with Yahoo Finance.

A table of data

Description automatically generated

*Liu, X.-Y., et al. (2022), "FinRL-Meta: Market environments and benchmarks for data-driven financial reinforcement learning,"*

## Testing FinRL

As mentioned, we conducted tests on three different types of agents using a training window of ten years of historical data from Yahoo Finance (Dow Jones top 30 tickers) and the standard parameters from the tutorials provided in the official FinRL documentation. To measure the results, FinRL provides some metrics for comparing the models, as follows:

* Cumulative return: R = V- V0/ V0

V = final portfolio value

V0 = original capital

* Annualized return: r = ((1+R) ^ 365/t) -1

t= trading days

* Annualized volatility 

ri = annualized return in year i

ř = avg annualized return

n = number of years

* Sharpe ratio:  (The Sharpe ratio estimates investment returns versus risk.)

rf = risk-free rate

* Max. drawdown: The maximal percentage loss in portfolio value.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Annual return | Cumulative returns | Annual volatility | Sharpe ratio | Calmar ratio | Stability | Max drawdown | Daily value at risk |
| SAC | 0.041 | 0.125 | 0.221 | 0.295 | 0.111 | 0.019 | -0.372 | -0.028 |
| PPO | 0.092 | 0.291 | 0.233 | 0.495 | 0.263 | 0.243 | -0.350 | -0.029 |
| DDPG | 0.139 | 0.457 | 0.217 | 0.708 | 0.449 | 0.739 | -0.309 | -0.027 |

With an initial budget of $1,000,000.00 and a runtime between 5 and 6 minutes, we achieved good results both in terms of increasing the value of the assets and with a risk vs. investment ratio very close to the margin between "good" and "high."

For instance, by running the SAC-type Agent after training in an environment simulating a real exchange with a 2-year timeframe (2019-2020), we achieved an estimated return of approximately $3,258,353.00.

A screenshot of a cell phone

Description automatically generated

A number and text on a white background

Description automatically generated

## Training and testing the agents:

**Tickers.**: Top 30 Dow Jones.

**Start**.: 2010-01-01.

**End**.: 2023-12-01 (testing data from 2022 to 2023)

We tested the following agents by varying some hyperparameters and increasing the amount of data compared to the previous version. We used the following agents, utilizing the stable baseline 3 library (initially, we used the elegantRL library but encountered compatibility issues and decided to switch): a2c, ddpg, td3, ppo, sac. The a2c and ddpg agents (which we used in the first prototype) do not have many hyperparameters. In contrast, the td3, ppo, and sac models allow adjustments to the batch size, buffer size, and learning rate.

During training, all of them performed well, with varying execution times for similar configurations. One notable aspect was the SAC model, which initially showed extremely high profits with a Sharpe ratio above 1.0, considered very good. However, as training progressed, these metrics decreased, with the Sharpe ratio seeming to stabilize around 0.8. This could indicate that the initial results were due to overfitting. We plan to explore this characteristic further and measure it more accurately to understand what is happening.

Here are the training results:

### A2C (Advantage Actor-Critic)

Combines value-based and policy-based approaches, using the advantage function to reduce variance.

Execution time : 14.94 minutes

A close-up of numbers

Description automatically generated

### DDPG (Deep Deterministic Policy Gradient)

Learns policies in high-dimensional, continuous action spaces using off-policy data and the Bellman equation.

Execution time :25.58 minutes

A black numbers and a white background

Description automatically generated with medium confidence

### PPO (Proximal Policy Optimization)

Optimizes policy gradients by maintaining a balance between exploration and exploitation, with simplified updates.

Execution time : 12.35 minutes

Hyper parameters:

"n\_steps": 2048,

"ent\_coef": 0.01,

"learning\_rate": 0.00025,

"batch\_size": 128,

"device":"mps"



A number and numbers on a white background

Description automatically generated

### TD3 (Twin Delayed Deep Deterministic policy gradient)

Improves DDPG by using two value functions to reduce overestimation and delayed policy updates.

Execution time : 25.1 minutes

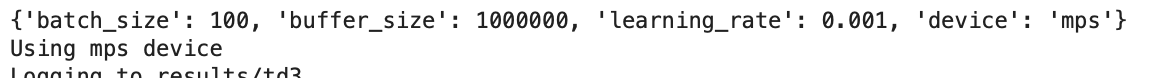
Hyper parameters:

"batch\_size": 100,

"buffer\_size": 1000000,

"learning\_rate": 0.001,

"device":"mps"



A close-up of numbers

Description automatically generated

### SAC (Soft Actor-Critic)

Focuses on maximizing a trade-off between expected return and entropy, encouraging exploration.

Execution time : 47.47 minutes

Hyper parameters:

"batch\_size": 128,

"buffer\_size": 100000,

"learning\_rate": 0.0001,

"learning\_starts": 100,

"ent\_coef": "auto\_0.1",

"device":"mps"



A close up of numbers

Description automatically generatedA screenshot of a computer code

Description automatically generated

## Trading (testing)

During testing, we also tested with data from 2021-10-01 to 2023-12-01. An interesting observation is that all models, including the average and the Dow Jones Industrial Average (DJI), had a negative peak on 2021-11-10, likely due to some atypical event on that date. In the long term, the PPO model surpassed the SAC agent, but it exhibited much instability; during training, PPO had the highest risk index of all.

A close-up of a chart

Description automatically generatedA graph showing a line of different colored lines

Description automatically generated with medium confidence

The SAC agent showed the least variation, but its financial gains were not as good as those of A2C and TD3.

The a2c model performed very well, but in the long term, the td3 model showed the best result, with moderate variation and achieving profit most of the time.

**A graph with lines and numbers

Description automatically generated**

## Architecture

To make all our solutions working together without costs, our application was divided into smaller services, each running in a different environment, but all are interconnected at some point. We are using different cloud solutions in integration with local batch process.

A diagram of a server

Description automatically generated

As for the AI Knowledge Hub, part of the Chatbot API, is using RAG architectures by using Coherer’s API for the LLM and Finnhub to generate financial information in the batch process.

**A diagram of a computer system

Description automatically generated**

## BrainLib

We created this library to facilitate development, the complexity of the code, and the number of libraries we use because of FinRL (for example). We encapsulated the main functions by organizing our library as follows:

A screenshot of a computer

Description automatically generated

**BrainDB.py** -> Contains simplified functions for accessing data in local and cloud databases already with connection settings. As in the example above, to check the data of the bots in the cloud, it is only necessary to create an instance of the brainDB.FireStoreDB class and then call the list\_bots() function. It's that simple.

**CustomEnv.py** -> This is the class containing the most recent version of our customized FinRL-meta environment.

**BrainTrader.py** -> This is the class that controls the integration between data, agent, and environment. It encapsulates complex tasks and libraries necessary for the bot's execution.

In the code snippet below, for example, we are using the brainTrader.GenericTrader class. The GenericTrader class was made to work with customized agents and data. This generalization makes it easier to use different agents and different data files.

A screenshot of a computer code

Description automatically generated

With the GenericTrader class, we just pass the data and call the “start\_simulation()” function.

## Batch Process

We are using the Apscheduler library to execute the “run\_session” function at a specific frequency. So, every x minutes, the batch process will run the trading based on the information registered by the interface and the data generated in the previous session.

A screen shot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generatedA screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generatedA screenshot of a computer

Description automatically generated

The run\_session function loads the list of information for all buttons that have not yet expired. Subsequently, the function checks whether it deals with multi-trading or single trading, and then, based on the value and type of trading, it selects a model that has been refined for that situation. For example, if the user chooses the ticker "AAPL" and allocates a high amount, the function will use the aapl\_td3\_high agent. After that, the trading is executed, with the session log saved in a CSV file. The function then concludes by reading the log file, updating the cash balance, and adding to the database the record of the number of assets purchased in the session and their values. Example of a record of the user's bot assets from a session for a bot running for all the tickers of the Dow 30:

A screenshot of a computer

Description automatically generated

Outcome of the batch execution for the example we registered in the user interface (demonstrated at the beginning of this document):

A screenshot of a computer

Description automatically generated

The batch process outcome displays the state of the previous session:

AAPL, $10,000 available for the bot to trade, the ticker's price on that day, and the number of shares the user held on that day, followed by the next row representing the trading day. In the second row, one can observe that the money now is the remainder from the day's transactions. We also have the share amount, the total asset value (number of shares \* ticker price), and the total fees paid in the transactions for that session.

# Findings

## Hidden Overfitting

It's very easy to fall into overfitting, although it's hard to identify clearly.

As noted in previous sections, when we tested the models, the results were quite impressive. We were able to increase our portfolio value by more than fourfold using models that were trained for just a few minutes over a 10-year period of historical data.

A number and numbers on a white background

Description automatically generated

When we apply these types of models to more recent test data, they end up performing well, though not as promisingly as in the training phase. However, it's noticeable that they become "biased" towards certain symbols, keeping them in the portfolio even when their market value is declining. It's as if they learn something along the lines of "holding Apple stocks is always good; they will always perform well."

In summary, great care must be taken when tuning these models, as it's possible to fall into the "illusion" that they can perform very well when, in fact, they are being "lazy" or "biased" towards a point of just buying more and more of a stock, even when it's no longer performing well in the present. The solution we found to balance this was to execute several rounds of tuning, carefully observing the behavior and distribution of the portfolio chosen by the model and adjusting the hyperparameters and training iterations.

## FinRL Installation and Alternatives for CUDA

We encountered numerous issues with the installation and operation of FinRL-Meta. Apparently, in the last three months, there has been a consolidation of all repositories by the AI4Finance team, resulting in broken or moved links to examples and sources, parts of the source codes being outdated or having conflicting versions of the same classes, bugs, among other issues. Consequently, many of the code examples were also broken, outdated, or incomplete, and it wasn't even possible to run them on Google Colab without making adaptations.

Additionally, the official version of the library is completely outdated. If you try to install it via pip, you will need to downgrade your Python version and many of the main libraries.

We found an installation alternative in the documentation, which was to install directly from the GitHub repository using the pip package manager, which was a pleasant surprise. Thus, by using the command `pip install git+<repository path>`, it's possible to install a library directly from its most recent version on GitHub, without needing any authentication or additional steps. With this information, we managed to install it in the Google Colab environment for testing, and even though we used GPU and TPU, we understood that the training process took a long time to complete.

Among the equipment available to our team were two gaming laptops and two Apple laptops with Apple Silicon (ARM64) architecture. However, it was an impossible task to install the library on Windows due to dependencies on development tools like C++ compilers among other requirements that seemed "strange" to us. Despite much effort, it was not possible, making running the system using CUDA unfeasible since the machines with Nvidia GPUs ran on Windows and even virtual machines couldn't install the entire environment.

Parallelly, based on past experiences in college, we identified that the libraries we were trying to use ran PyTorch behind the scenes, which allows the use of MPS or Metal Performance Shaders, which is Apple's equivalent to CUDA.

Thus, we cloned the FinRL repository and added a command so that the library could now be compatible with the use of MPS, allowing us to run the model training using the GPU of Apple laptops. Then, we used the direct installation feature from GitHub to install the modified library.

The MPS-compatible version of FinRL is available at https://github.com/FabioD-Junior/FinRL and involves a simple line of instruction that was replaced in the paper trading module.

A screen shot of a computer program

Description automatically generated

In summary, running a simple basic example of the library for the first time was very challenging, but once we had a complete understanding of what we were doing, we were able to overcome the problem. This made us look at an improvement point that is ignored nowadays and is also not widely disclosed, which is the use of MPS for AI purposes. Most people developing AI solutions with Python are not even aware of this option to replace NVIDIA's CUDA, and due to its ARM64 architecture nature, it consumes much less energy than an approach using regular GPUs. However, we not only applied the use of MPS in our project but also enabled the FinRL library so that other people from the open community can also be aware of and enjoy this alternative.

## Creation of our own version of FinRL : FinRL-Mini

The processes of training and executing DRL agents, especially when there's a large amount of data, require a lot of resources, burdening any basic system. It can present bottlenecks in processing, demanding a long time to finalize processes that seem simple to the user, as well as being prone to memory overflow, disk, or in the case of using cloud technologies, a very high charge if the automatic provisioning of resources is not being well managed and monitored. One point that we are proud of in this project was the ability to use our creativity to circumvent some of these problems and still achieve an excellent result, for example, executing the simulation in a cloud environment (AWS) within the free tier.

After a great effort analyzing the source codes of FinRL, which is available as open source on GitHub, we had the idea to perform "reverse engineering" to understand how this library works and to create a new version containing only what is necessary to specifically execute our purpose. As explained in the previous section, the FinRL library tries to bring the most varied and useful tools for the development of DRL solutions for finance, making it generalist and more specific for training the model and exploration than for the process of executing the created solution. Thus, we cloned the entire library repository and downloaded it locally, removing the components and classes that were used only for training or that were not related to our solution, such as the environment module used to connect with Chinese exchanges, the environment and dependencies used to access the Alpaca environment, methods of downloading data that were not from Yahoo Finance among others. We discovered in this way that this library does not use the common dependency mapping process that we learned in college and are used to, such as the use of requirements.txt, instead, it uses a pair of files to map what must be installed for the library to function. These files are:

* **Pyproject.toml** – Contains the description and metadata of the library, such as the author's name, license, etc., the version of Python compatible with the library, and the main libraries that must be installed first.
* **Poetry.lock** – Responsible for the dependencies necessary to install the main libraries related in the **pyproject.toml**, as well as any other extra dependency (such as examples, templates, etc.).

A screenshot of a computer program

Description automatically generatedA computer screen with text and numbers

Description automatically generated

We also made adaptations in these files with the intent of installing only the relevant dependencies and excluding extra dependencies that are never used, for example, one of the libraries has as an extra dependency a gaming environment for Atari, which probably should be used as an example for studying and training agents capable of playing video games.

Our library, FinRL-Mini, derived from the FinRL library is available on our GitHub. We maintained all the original credits and information of FINRL, adding the information that this is a lighter version adapted to run using fewer resources and only for execution, not training models.

It can be installed using the command:

pip install git+https://github.com/FabioD-Junior/finrl\_mini/

Once again, through our creativity and problem-solving ability, we managed to contribute to the open-source community while solving a complex problem.

A screenshot of a computer

Description automatically generated

# Discussion:

a. In-depth discussion about your product.

# Conclusion:

a. Any discussions above that need resolution.

b. Wrap up your topic of discussion.

c. Wrap up the overall project.

# Recommendation:

a. Features that can be added or improved to your project.

b. Addition or removal of features.

# References:

a. APA Style Citations.

b. Proper convention of Reference List.

c. Must triple check.

# Appendices:

a. Any data that are mentioned in the findings or discussions but is obstructing the flow of

information can be included in this section.

b. Notation is (See Appendix A).

* AIP Group A3 [Brain] (2024), Research: Official guidelines on trading regulations in Canada.