POE: A General Portfolio Optimization Environment for FinRL

Caio de Souza Barbosa Costa[1], **Anna Helena Reali Costa**[1]

[1]Escola Politécnica
Universidade de São Paulo (USP) – São Paulo, SP – Brazil

{caio326,anna.reali}@usp.br

Abstract. *Portfolio optimization is a common task in financial markets in which a manager rebalances the invested assets in the portfolio periodically aiming to make a profit, minimize losses and maximize long-term returns. Due to their great adaptability, Reinforcement Learning (RL) techniques are considered convenient for this task but, despite RL's great results, there is a lack of standardization related to simulation environments. In this paper, we present an RL environment for the portfolio optimization problem based on state-of-the-art mathematical formulations. The environment aims to be easy-to-use, very customizable, and have integrations with modern RL frameworks.*
Keywords*: Portfolio optimization. Reinforcement learning. Simulation environment. Quantitative finance.*

1. Introduction

In order to deal with market volatility, investors constantly apply strategies to reduce losses, increase gains and maximize their long-term returns. For this, it is opportune for these strategies to be able to predict the future price of assets so that buying and selling actions are carried out properly. Therefore, machine learning techniques have been increasingly applied to financial markets, achieving excellent results [Khadjeh Nassirtoussi et al. 2014, Hu et al. 2015, Henrique et al. 2019].

One of the most prominent approaches is the use of reinforcement learning (RL) algorithms, as the main objective of these algorithms is to maximize the long-term return of a profit-related reward function, just like human investors. Furthermore, RL is able to handle brokerage fees (something supervised machine learning approaches have a hard time dealing with [Deng et al. 2017]), is very adaptable to unusual situations and can achieve effective behaviors even in the early stages of learning [Sutton and Barto 2018]. However, to effectively use RL, it is necessary to model the dynamics of interactions between the agent and a task-specific environment. In the financial market context, the task of an agent is to trade assets from time to time and make a profit, therefore, the task-specific environment must be able to simulate the market behavior over time, perform the agent's buying and selling actions and calculate the resulting gains and losses.

Due to the need to develop a task-specific environment to train an agent, one of the major problems in RL research is the lack of standardization in environments, tasks and interactions, which hinders the ability to make comparisons between the performances of different solutions. To deal with that, frameworks like *OpenAI Gym* [Brockman et al. 2016] and *FinRL* [Liu et al. 2022b] aim to provide a number of environments to be used by researchers as common benchmarks, the latter one focused on

financial markets applications. However, despite having quite a success and wide adoption by the scientific community, none of those frameworks provide an easy-to-use environment compatible with the modern portfolio optimization agents, especially after the introduction of convolutional architectures like in [Jiang et al. 2017], which make use of multiple price time series as state-space.

That said, the objective of this paper is to fill that gap by introducing *PortfolioOptimizationEnv* (POE), an open-source portfolio optimization environment whose state representation is compatible with state-of-the-art approaches. Developed as part of *FinRL*, this environment benefits from the features of that framework, follows the *OpenAI Gym*'s environment standards, and is compatible with modern repositories of RL algorithms such as *stable baselines 3* [Raffin et al. 2021]. Finally, the environment is also able to calculate several performance metrics to be used to compare different solutions to the portfolio optimization problem.

This paper is organized as follows: Section 2 formalizes the portfolio optimization task and introduces the main metrics and hypotheses, Section 3 introduces the RL approach to solve the task, Section 4 describes the proposed portfolio optimization environment, its guidelines and how it was implemented and, finally, Section 5 concludes the paper.

2. Portfolio optimization

Portfolio optimization is a task in which a trading system is responsible for periodically defining the allocation of resources in a portfolio. The system starts the task with a specific amount of cash and, seeking to make a profit, it constantly rebalances all the investments after analyzing the market for an interval of time: the small periods of time between market analysis intervals in which the system is able to rebalance the portfolio are called *reallocation periods* (as shown in figure 1). An optimal portfolio optimization framework must be able to act in these reallocation periods to increase profits and mitigate losses.

2.1. Mathematical definition

Given a portfolio with n assets, we can define, at each time step, a vector called *weights vector* ($\mathbf{W_t}$) that contains the percentage (or weights) of all the assets in the portfolio and of the remaining capital. Therefore, the size of $\mathbf{W_t}$ is $|\mathbf{W_t}| = n + 1$ and the sum of their elements must be 1. Mathematically speaking, being $\mathbf{W_t}(i)$ the i-th element of $\mathbf{W_t}$, the *weights vector* must be constrained by

$$\sum_{i=0}^{n} \mathbf{W_t}(i) = 1. \tag{1}$$

At each time step, we can also define a price vector $\mathbf{P_t}$ with the following form $[1, \mathbf{P_t}(1), \mathbf{P_t}(2), ..., \mathbf{P_t}(n)]$. The first element of the price vector is always 1 because it's related to the remaining capital and it is invariable. The first element can also be seen as a reference price: all other prices are calculated in relation to this one.

By knowing the last value of the portfolio ($\mathbf{V_{t-1}}$), the current price vector ($\mathbf{P_t}$), the last price vector ($\mathbf{P_{t-1}}$) and the last weights vector ($\mathbf{W_{t-1}}$), it's possible to calculate

the current portfolio value V_t by using the recursive formula:

$$V_t = V_{t-1}\Big(\mathbf{W_{t-1}} \cdot (\mathbf{P_t} \oslash \mathbf{P_{t-1}})\Big), \tag{2}$$

where $\cdot$ is the dot product of two vectors, and $\oslash$ is the element-wise division. Note that this method assumes that every asset will have positive prices since a division by zero in the element-wise operation will cause errors.

The value of the portfolio has changed from V_{t-1} to V_t because the prices of the assets have changed, but that change also affects the weights vector, which, in the end of the last time-step, assumes a different value given by

$$W^f_{t-1} = \frac{(\mathbf{P_t} \oslash \mathbf{P_{t-1}}) \odot W_{t-1}}{(\mathbf{P_t} \oslash \mathbf{P_{t-1}}) \cdot W_{t-1}}, \tag{3}$$

where $\odot$ is the element-wise multiplication. The intuition of W^f_{t-1} is that it represents the weights matrix at the final moment of time step $t-1$, information that is necessary when trading fees are imposed to the system (section 2.2). Figure 1 represents the portfolio optimization process over time, note that W_t is determined during the reallocation period.

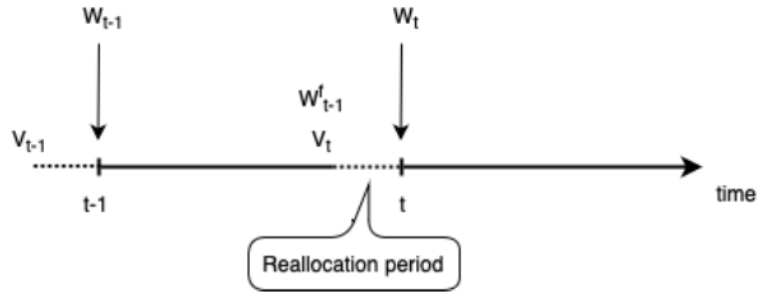

Figure 1. Diagram representing the portfolio optimization process over time.

It's common to model the profit at time-step t as a logarithmic rate of return (r_t):

$$r_t = ln\Big(\frac{V_t}{V_{t-1}}\Big). \tag{4}$$

This way, the final portfolio value can be expressed as:

$$V_{T+1} = V_0 \; exp\Big(\sum_{t=1}^{T+1} r_t\Big), \tag{5}$$

where T is the last time step in which a portfolio allocation occurs, and r_{T+1} is the profit of this final allocation.

The portfolio optimization problem consists of finding the optimal sequence of portfolio weights vector ($[\mathbf{W_0}, \mathbf{W_1}, ..., \mathbf{W_T}]$) that maximizes the final portfolio value V_{T+1}.

2.2. Trading fees

The formulation in section 2.1 does not consider brokerage fees, which are usually charged when assets are bought or sold. In the real world, though, trading fees are a common practice and can greatly influence the system's performance since the cost of trading increases as the number of orders increases. Usually, in many brokers, a trading fee is given by a percentage of the overall value of the transaction, and during the portfolio reallocation period, the portfolio value decreases due to costs related to it. Because of that, it is necessary to differentiate the portfolio value at the end of a time step (V_{t-1}^{f}) from the portfolio value at the beginning of the next step (V_t) (after fees are applied).

There are two ways of modeling brokerage fees in portfolio optimization problems:

Weights vector modifier: In this approach, at the beginning of time step t, the agent defines an action in the form of a weights vector W_t. The environment then calculates the trading fees by summing the absolute variation of cash invested in each asset and applying the percentage fee. The brokerage fee is, then, taken from the remaining cash, modifying the weights vector W_t to a new one W_t' and also changing the portfolio value from V_{t-1}^{f} to V_t. This method has one main exception: if the trading fee is bigger than the remaining cash, the portfolio reallocation step can not be performed and the last vector W_{t-1}^{f} is considered as W_t'. Figure 2 illustrates how the weights vector modifier is applied to the trading process.

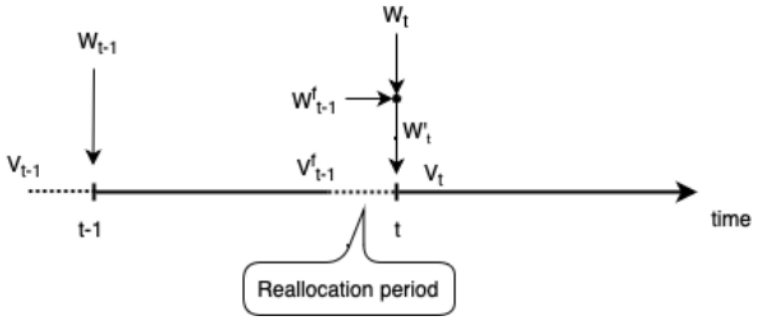

Figure 2. Portfolio optimization process with weights vector modifier trading fee.

Transaction remainder factor: Another way to simulate transaction costs is to define a remainder factor μ_t that modifies the portfolio value during the reallocation:

$$V_t = \mu_t V_{t-1}^{f}. \tag{6}$$

μ_t has a different value depending on the time step since each reallocation may have a different number of trading orders performed. The formula to calculate μ_t and its demonstration is available in [Jiang et al. 2017]. Figure 3 shows the influence of the transaction remainder factor on the portfolio value.

Since the transaction remainder factor model does not suffer with the exception described in the weights vector modifier model, it is preferred the majority of the time.

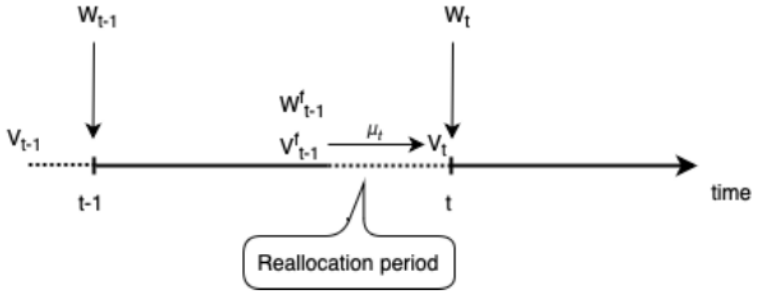

Figure 3. Portfolio optimization process with transaction remainder factor trading fee.

2.3. Performance metrics

In order to compare different portfolio optimization approaches, it is important to apply some performance metrics. Like in [Jiang et al. 2017], the metrics adopted here are final accumulative portfolio value (fAPV), maximum drawdown (MDD), and Sharpe ratio (SR), described below.

Final accumulative portfolio value (fAPV): This performance metric is responsible for calculating the profit generated by the portfolio during a specific time interval. The bigger the fAPV, the bigger the profit generated by the implemented approach. Being V_0 the initial portfolio value and V_{T+1} the final portfolio value, it is possible to calculate the final accumulative portfolio value by doing:

$$fAPV = \frac{V_{T+1}}{V_0}. \tag{7}$$

Maximum DrawDown (MDD): The fAPV is a great measure of profit but it ignores the risk necessary to achieve that profit. One of the ways to model the risk is to analyze the maximum drawdown of a portfolio value time series. The maximum drawdown is defined as the biggest downward movement of that time series [Magdon-Ismail et al. 2004]:

$$MDD = \max\left(\max_{t<\tau} \frac{V_t - V_\tau}{V_t}\right). \tag{8}$$

Sharpe Ratio (SR): Another risk-related metric is the Sharpe Ratio [Sharpe 1994]. This metric is a risk-adjusted mean return, defined as the average of the risk-free return by its deviation:

$$SR = \frac{\mathbb{E}_t[\rho_t - \rho_F]}{\sqrt{var_t[\rho_t - \rho_F]}}, \tag{9}$$

where $\rho_t = V_t/V_{t-1}$ is the rate of return, ρ_F is the rate of return of risk-free asset (which is usually assumed to be zero) and the denominator is basically a standard deviation of the numerator.

Considering these metrics, a good portfolio should have the highest possible $fAPV$ and SR, as these are measures of return. It should also have a lower MDD to ensure it is as risk-free as possible.

2.4. Common hypotheses

In many portfolio optimization solutions, especially those based on RL, it is necessary for the system to pretend to be back in time at a point in the history of the market and experience a simulation of the passage of time [Jiang et al. 2017]. To achieve this, two hypotheses are commonly imposed:

No slippage: The market is assumed to have considerably high liquidity so orders placed are immediately completed at the last price.

No market impact: The trading actions done by the system don't impact the market dynamics because it's assumed that the amount of assets traded is small.

These hypotheses simplify the task of portfolio optimization, as in the real world a large amount of trades can affect market prices, and there is also no guarantee that a trader will be able to complete an order without delay.

3. Reinforcement learning in the portfolio optimization problem

A way to solve the portfolio optimization problem is to train an RL agent. In this approach, an agent learns a policy (π) by interacting directly with an environment. At each time step, the environment is responsible for providing observations (O_t) to the agent that defines a state (S_t) of the system, which is used by the agent to decide an action (A_t) to be performed. After an action is performed, the environment also provides the agent a reward (R_t) that is used by the agent to know if the action performed was well chosen. The goal of the RL agent is to define an actuation policy that maximizes the expected sum of received rewards. If the observations match the states of the system, i.e. $O_t = S_t$, then the policy can be defined as $\pi : S \rightarrow A$. Figure 4 represents a training cycle of an RL agent interacting with the environment.

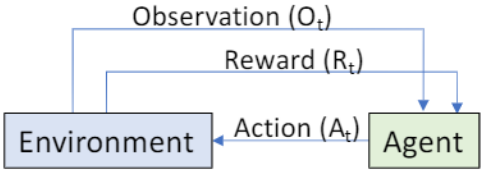

Figure 4. Reinforcement learning interacting cycle.

However, to be properly used in portfolio optimization tasks, the agent must be able to use complex state spaces, as a portfolio is composed of multiple assets and each asset has its own price series. In their book, [Sutton and Barto 2018] introduce ways to deal with complex and continuous state space by using function approximations like neural networks, which achieve formidable results in many tasks. Therefore, the most convenient algorithms are *Deep Reinforcement Learning* (DRL) ones.

Before modeling the state space, action space, and reward function, it is important to explain the assumptions made by the RL approach described in this section. To deal with the hypotheses introduced in section 2.4, RL is modeled as a *contextual bandit* problem instead of a *full RL* one. The main difference is that in (full) RL, an action A_t in state S_t not only affects the reward R_t that the agent will get but will also affect the next state S_{t+1} the agent will end up in, while, in contextual bandits, an action A_t in state S_t only

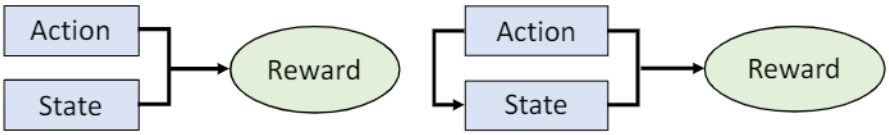

Figure 5. Comparing Contextual Bandit (left) with Reinforcement Learning (right).

affects the reward R_t that the agent will get. Contextual bandit can be seen as a simpler form of RL, as illustrated in Figure 5.

Another assumption made is that the problem can be modeled as a Markov decision process (MDP) [Sutton and Barto 2018] and therefore has the Markov property, which states that every information necessary by the agent to achieve the optimal policy is observable by the agent in the state space. In the real world, considering the complexity of the trading market, the Markov property might not be the best assumption, which would force the modeling of a partially observable Markov decision process (POMDP), which makes the solution much more complex. However, in this version of the simulator that we present here, the problem is considered an MDP and the observations are considered to be the states, i.e., $O_t = S_t$.

State representation: The agent needs information about the time series of each asset of the portfolio. So, if a portfolio has n assets with history information of t steps in the past (time series with t time window), a matrix if size (n, t) is necessary to represent all the information. However, this bi-dimensional representation only encompasses one feature when, in fact, multiple features might be necessary for the agent to capture market trends. [Weng et al. 2020] shows that the most important features to use in the portfolio optimization problem are closing, low and high price but features like volume, open price and even indicators can be used in the state representation. Therefore, the state space is better represented as a multi-dimensional matrix of shape (f, n, t), where f is the number of features, n is the number of assets and t is the time window of the times series. Figure 6 represents the state space for portfolio optimization tasks.

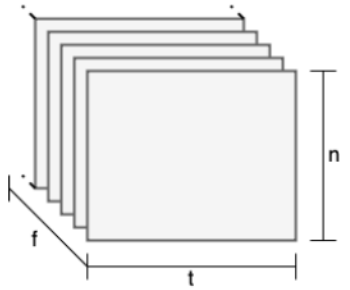

Figure 6. State space for portfolio optimization.

Action space: Considering the mathematical formulation introduced in section 2.1, the action space for RL agents trained for the portfolio optimization problem is identical to the *weights vector* $\mathbf{W}_t$ and follows the same restrictions. Note that, in the context of DRL agents, these restrictions are easily implemented using a softmax activation function in the policy output layer.

Reward function: Many functions can be used to model the environment reward, but the most common one is the *logarithmic rate of return* r_t introduced in Equation 4, since it is available naturally at each time step. Some papers [Almahdi and Yang 2017, Betancourt and Chen 2021] have also sought to incorporate some indicators (mainly the SR index) for the reward function so that the agent learns to deal with market risk.

4. Proposed Environment

With the advent of DRL in financial markets, many frameworks and environments have been developed to allow the research in the area. Many of those environments, like *gym-anytrading* [Haghpanah 2023], *gym-mtsim* [Haghpanah 2021] and the ones available in *FinRL-Meta* [Liu 2022], follow *OpenAI Gym*'s architecture and are integrated with many RL libraries but do not provide the Markov representation presented in Section 3. Others, like [Amrouni 2022], are developed with the mathematical formulations presented in previous sections but are not well integrated with RL libraries.

To improve this scenario, we propose an open-source environment for RL agents that is able to simulate the main aspects of the portfolio optimization task considering the definitions presented in Sections 2 and 3 and is integrated with modern RL libraries. This environment[1] is a contribution to *finRL-Meta* [Liu 2022], a repository containing multiple open-source environments of *finRL* [Liu et al. 2022b], one of the main RL frameworks for finance.

4.1. Guidelines

The proposed environment has the following guidelines:

Complete integration with *FinRL*: *FinRL* provides a full pipeline to build, evaluate and deploy DRL agents that operate on the financial market, the reason why the proposed environment was designed to be a part of it. To have full integration with *FinRL* allows this project to be considerably "plug-and-play", increasing the convenience for researchers.

Generality: As seen in previous sections, the portfolio optimization task has many parameters to be considered: portfolio size, time window size, trading fee percentage, etc. For that reason, the proposed environment follows the guideline of being as general as possible so that users do not need to modify its code the majority of the time.

***OpenAI Gym* structure:** Since *OpenAI Gym* [Brockman et al. 2016] is currently the most famous framework for standardizing RL environments, this project aims to follow its structure, allowing it to be used outside of *FinRL*.

Integration with reliable DRL libraries: One of the main reasons to use *OpenAI Gym* structure is to allow the environment to be used with reliable DRL libraries that have integrations with *Gym*'s structure. The ability to use this environment with powerful tools such as *stable-baselines-3* [Raffin et al. 2021] or *ElegantRL* can improve research considerably.

[1]The implementation code is available in `https://github.com/C4i0kun/FinRL-Meta/tree/portfolio_allocation/meta/env_portfolio_optimization`, but it will be eventually merged to FinRL-Meta (`https://github.com/AI4Finance-Foundation/FinRL-Meta`) when the merge request is accepted.

Open contribution: This project aims to be open-source and contributions from the community over time will let new features to be added to it. The main idea of this environment is to be eventually merged to *FinRL-Meta* when approved by the maintainers so that a larger community is able to use it and contribute to it.

4.2. Implementation

The environment was implemented using Python programming language and, as stated in section 4.1, it is an *OpenAI Gym* environment, which has `reset`, `step` and `render` interface. As any Python object, to use that interface, it's necessary to instantiate the `PortfolioOptimizationEnv` and determine a *Pandas* [Team 2023] dataframe that contains historical data and an initial amount to invest. Following the generality guideline, many simulation parameters can be determined, such as:

- A normalization method, which can be one of the included normalization methods or can be created by the user;
- A time window to represent the state (see Section 3);
- Commission fee model (weights vector modifier or transaction remainder factor) and a commission fee rate to be considered in the models;
- The list of columns from the dataframe to be considered as features by the environment (unlisted columns are ignored). It is also possible to modify the feature that is used to calculate the portfolio value (by default, the asset's closing price);
- Options to deal with dataframe's column names and time format if the user is not using *FinRL*'s default ones;
- A repository in which graphs must be saved.

After determining all the parameters, it's finally possible to interact with the environment. Since it follows an *OpenAI Gym*'s structure, any agent that receives the state and determines an action (a weights vector) at each time step can be used, being an RL agent or not. The agent must interact through a `reset`, `step`, and `render` interface.

`reset`: This method is responsible for resetting the environment by returning the simulation date to the first date of the dataframe. It also modifies the environment attributes to their default value, allowing the simulation to execute again. This method returns the current state after resetting.

`step`: As the name suggests, this method is responsible for executing a simulation step. As seen in section 2, a step modifies the portfolio considering price fluctuation during periods. The amount of time a step represents depends on the dataframe used by the environment: if the dataframe contains daily time series, then it is assumed that a step represents a period of a full day. This method needs to have an action parameter (that is the new portfolio weights the agent wants to allocate) and returns the common necessary information for an RL algorithm: a new state, a reward considering the action performed in the previous state, and a boolean variable defining if the next state is terminal.

`render`: Renders the environment so that a human user can visualize what is happening. Currently, this method only returns the current state's numerical values.

It's important to understand what is defined as an episode in this environment: considering a dataframe ordered by time and being t the size of the time window and T

the last time step of the dataframe, the episode will start in the t-th time step and end when the environment tries to access an unexistent $T + 1$ time step. At that moment, the `step` method will stop simulating the passage of time and will print and plot metrics of the portfolio related to the full episode. To start a new episode, the agent must use the `reset` method.

4.3. Visualizing performance

After an episode, it's important to assess the agent's performance using the performance metrics presented in section 2.3. Therefore, the proposed environment makes use of *QuantStats* [Aroussi 2023] library to provide not only reliable metric calculations but also graphs so that researchers can understand the behavior of the trading system.

Figures 7 and 8 show two graphs generated by the environment: the portfolio summary and rewards over time. As it can be seen, even though the portfolio had great results according to the cumulative return of figure 7, the rewards shown in figure 8 indicates that the agent receives erratic rewards during the episodes and is not learning with the experience at all. In fact, the agent used in this example performs a simple buy-and-hold strategy in a portfolio of 19 assets of the Brazilian market from 2010 and 2022, so it does not adjust according to the market's dynamics and only makes a profit due to the growth of Brazilian's market in the period.

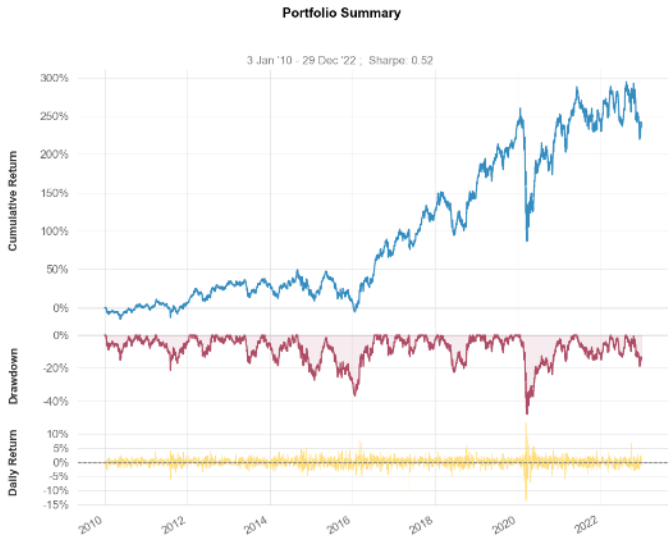

Figure 7. Visualization of portfolio summary.

5. Conclusions

This article presented an open-source portfolio optimization environment for RL agents to improve the standardization of benchmarks and comparisons between researchers in the area. Being developed as part of *FinRL*, the proposed environment can benefit from its features and be used in its ecosystem as part of an automatic financial pipeline. This project was also developed following *OpenAI Gym*'s structure and, because of that, has great integrations with many RL algorithm repositories, such as *stable-baselines-3* [Raffin et al. 2021] and *ElegantRL* [Liu et al. 2022a]. All of those design decisions can

Figure 8. Visualization of portfolio reward.

considerably impact the area of research positively since the environment is *plug-and-play*, which allows the researchers to focus on the agent's architecture and feature extractions.

Future works can focus on turning the environment even more realistic by considering the agent's impact on the market if big trades or trades related to a low-volume asset are done, transforming the problem into a full RL problem. Another area of improvement is the fact that this environment is modeled as a Markov decision process, something not very realistic as stated in Section 3, so the addition of a partially observable Markov decision process model of the problem would be necessary. Finally, the state representation forces all the assets of the portfolio to be analyzed in the same time window, which may not be possible for all assets. A solution to this problem can be addressed in future works, making the environment even more general.

Acknowledgments

The authors would like to thank the *Programa de Bolsas Itaú* (PBI) of the *Centro de Ciência de Dados* (C^2D) at Escola Politécnica at USP, supported by *Itaú Unibanco S.A.*, and the Brazilian National Council for Scientific and Technological Development (CNPq Grant N. 310085/2020-9).

References

Almahdi, S. and Yang, S. Y. (2017). An adaptive portfolio trading system: A risk-return portfolio optimization using recurrent reinforcement learning with expected maximum drawdown. *Expert Systems with Applications*, 87:267–279.

Amrouni, S. (2022). Selimamrouni/Deep-Portfolio-Management-Reinforcement-Learning: V2.0. Zenodo.

Aroussi, R. (2023). Ranaroussi/quantstats.

Betancourt, C. and Chen, W.-H. (2021). Deep reinforcement learning for portfolio management of markets with a dynamic number of assets. *Expert Systems with Applications*, 164:114002.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). OpenAI Gym.

Deng, Y., Bao, F., Kong, Y., Ren, Z., and Dai, Q. (2017). Deep Direct Reinforcement Learning for Financial Signal Representation and Trading. *IEEE Transactions on Neural Networks and Learning Systems*, 28(3):653–664.

Haghpanah, M. A. (2021). Gym-mtsim.

Haghpanah, M. A. (2023). Gym-anytrading.

Henrique, B. M., Sobreiro, V. A., and Kimura, H. (2019). Literature review: Machine learning techniques applied to financial market prediction. *Expert Systems with Applications*, 124:226–251.

Hu, Y., Liu, K., Zhang, X., Su, L., Ngai, E. W. T., and Liu, M. (2015). Application of evolutionary computation for rule discovery in stock algorithmic trading: A literature review. *Applied Soft Computing*, 36:534–551.

Jiang, Z., Xu, D., and Liang, J. (2017). A Deep Reinforcement Learning Framework for the Financial Portfolio Management Problem.

Khadjeh Nassirtoussi, A., Aghabozorgi, S., Ying Wah, T., and Ngo, D. C. L. (2014). Text mining for market prediction: A systematic review. *Expert Systems with Applications*, 41(16):7653–7670.

Liu, X.-Y. (2022). FinRL-Meta: Market Environments and Benchmarks for Data-Driven Financial Reinforcement Learning.

Liu, X.-Y., Li, Z., Yang, Z., Zheng, J., Wang, Z., Walid, A., Guo, J., and Jordan, M. I. (2022a). ElegantRL-Podracer: Scalable and Elastic Library for Cloud-Native Deep Reinforcement Learning.

Liu, X.-Y., Yang, H., Gao, J., and Wang, C. D. (2022b). FinRL: Deep reinforcement learning framework to automate trading in quantitative finance. In *Proceedings of the Second ACM International Conference on AI in Finance*, ICAIF '21, pages 1–9, New York, NY, USA. Association for Computing Machinery.

Magdon-Ismail, M., Atiya, A. F., Pratap, A., and Abu-Mostafa, Y. S. (2004). On the maximum drawdown of a Brownian motion. *Journal of Applied Probability*, 41(1):147–161.

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N. (2021). Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research*, 22(268):1–8.

Sharpe, W. F. (1994). The Sharpe Ratio. *The Journal of Portfolio Management*, 21(1):49–58.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA.

Team, T. P. D. (2023). Pandas-dev/pandas: Pandas. Zenodo.

Weng, L., Sun, X., Xia, M., Liu, J., and Xu, Y. (2020). Portfolio trading system of digital currencies: A deep reinforcement learning with multidimensional attention gating mechanism. *Neurocomputing*, 402:171–182.