COMP0031 - MARL in Portfolio Management

Cheng Loo, Julia Goh, Neil Badal, Preston Tong, Radu-Bogdan Priboi, Thatchawin Leelawat

***Abstract*—Financial Portfolio Management (PM) is an essential technique investors use to manage risk and maximise long-term returns. While the central concept of portfolio management is straightforward and can be automated, manual strategies are usually used instead. These strategies are complex and generally require extensive intuition, knowledge, and experience. For these reasons, there has been a recent interest in applying multi-agent reinforcement learning (MARL) to this area. Using MARL, we allow the system to autonomously self-learn custom strategies from patterns in data, reducing the dependence on human expertise. However, building these systems requires selecting certain hyperparameters and specific training algorithms. In this paper, we aim to explore the impact of these hyperparameters on the overall solution, specifically on the final return and chosen performance metrics of the resulting portfolio.**

I. Introduction

PM can be defined as the process of continuously distributing and allocating capital into multiple assets with the objective to maximise total return and minimise the risk [1]. There is a correlation between risk and the specific type of strategy used to allocate assets; a higher-risk strategy may lead to higher overall returns but has an increased chance of significant losses. Stock market states, current asset portfolio ratios, and observed market sentiment are some of the main inputs used to determine trading decisions. These decisions refer to the allocating or reallocating of an asset through a buy, sell, or hold action [2]. An investor must manually analyse this data and take the required actions. The main problem with this process is the vast amounts of non-stationary, non-linear, and multidimensional financial data needed to be analysed. This has led to a limited number of statistical and traditional approaches being used [3].

Current baselines and solutions require experts to read financial statements manually, follow news articles, and analyse stock price trends [1]. Others include experts constructing algorithms using domain knowledge and intuition [4]. Such methods tend to be rigid and unscalable for the dynamic and unpredictable nature of financial markets and the task, making them unreliable and unsuitable [1]. Classic machine learning approaches make many assumptions, like zero-transaction costs. Existing reinforcement learning (RL) techniques build agents that are essentially not reusable and ad-hoc trained [1]. While some of these traditional and RL-based strategies [5] perform reasonably well, these systems tend to perform poorly in real-life settings [3]. Due to the vast amounts of information and the dynamic nature of financial markets, PM faces significant challenges and is a complex problem to solve [6]. Nevertheless, these challenges are effectively addressed through MARL techniques, making them extremely valuable.

While single agent systems can solve the problem, we believe MARL systems are more suited due to the nature of the problem. In these systems, an agent represents an investor. Each agent with some capital aims to allocate capital optimally into assets using its past experience. In this model, having multiple (interacting) agents either working together or against each other (collaborative vs. competitive) is more representative of properties seen in real markets. Modelling the problem in this direction is likely to lead agents to learn more meaningful strategies. Additionally, it has the added benefit of being useful for regulators to model environments. A collaborative setting can be compared to investors in a company working together to maximise their total return while each is individually trying to create diverse and optimal portfolios. A competitive setting can be compared to a situation where investors may compete against each other to get the highest return. Both situations cannot be represented using single-agent systems that generally focus on the unrealistic scenario of one investor investing with no challenges. For these reasons, we believe that MARL systems are more appropriate for solving this problem. We explore how modifications to the system's hyperparameters affect return, performance metrics, and the allocation strategies learned.

II. Related Works

A. Single-Agent RL

In single-agent RL, only one agent is interacting with the environment. It is used to decide the actions to take for maximising the total reward. By mapping the states and actions, feedback is received in the form of rewards and penalties.

The environment can be represented in the form of a Markov Decision Process, mathematical framework, that can be expressed as a 5-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, where: $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of all valid actions, $\mathcal{P}$ is a transition probability function, defined as $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$, $\mathcal{R}$ is a reward function, defined as $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ and $\gamma \in$ [0, 1] is a discount factor used to evaluate the importance of the future rewards. This is used in the expected discounted return: $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...$, where R_t is the sum of future rewards.

Most of the single agent research in PM focuses on deep reinforcement learning (DRL), where the agent and planner are implemented with DNN. This is due to its benefit of overcoming the "curse of dimensionality" as the sizes of input data and features grow. In terms of feature engineering, most of them involve using the basic metrics (e.g., opening, closing, high and low prices) as training inputs. On top of

these, [7] seeks to combine features to train the model with financial indexes like Price-to-Earning and Price-to-Book ratios, whereas [8] includes beta index to observe the volatility of the stocks.

Different algorithms are also implemented and investigated in the projects respectively. For instance, in this project [8], the Deep Q-Network algorithm which is model-free, is used with artificial neural network (ANN) layers which can learn the non-linear relationship in data. Adversarial DRL training is also introduced in [7] to experiment three different algorithms: (a) Deep Deterministic Policy Gradient, (b) Proximal Policy Optimization, and (c) Policy Gradient in the PM setting. The Deep Deterministic Policy Gradient algorithm is also used to design a DRL framework for the financial PM problem [9]. To track the movement of the prices, memory vectors such as Portfolio Vector Memory (PVM) and Long Short Term Memory (LSTM) are used in [10]. Furthermore, this literature [11] introduced augmented asset movement as the RL states with the use of LSTM to predict the movement of prices in each state and increase the performance of the portfolio value.

In general, the model output involves its estimated accumulative portfolio value (APV) [10]. In terms of evaluation metrics, most research such as [7], [8] and [10] use the Sharpe Ratio and average or cumulative returns to estimate the model performance. In [7] and [10], they seek to evaluate their model through an additional measure of maximum markdown.

B. RL Algorithms

Advantage Actor-Critic (A2C) is a type of RL algorithm which takes advantage of the combination of value-based and policy-based methods. It consists of two networks, "the actor" that controls the behaviour of the agent, and "the critic" that gives an estimate action quality.

Proximal Policy Optimization (PPO) is a descendent of policy gradient method, similar to A2C. It works by continuously updating the policy in order to maximise the expected reward. In order to ensure stability of the training and to avoid massive changes in the policy, constraints are applied on the updates [12].

Deep Q-Network (DQN) is a Deep RL algorithm which makes use of the learning capacity of a Q-Network. This is used to estimate Q-values - the expected total reward, where it is calculated for all possible actions in a given state. By using a deep neural network (DNN) to estimate these values, DQN can also learn in complex environments [12].

Deep Deterministic Policy Gradient (DDPG) represents an extension of the DQN algorithm. The agent is using two DNNs, one as an actor network that determines the policy, and the other as a critic network that evaluates the quality of the action.

C. MARL

As an extension to the previous approach, MARL relies on the utilisation of multiple agents that are sharing the same environment. It is possible to replicate environments of higher complexity. The transition from single to multi-agent RL implies considering all the interactions between agents in the environment. This requires that each agent takes decisions influenced by the actions of the other agents. The policy of each agent is also influenced by the actions of the other agents. The agents can be cooperative where they work towards a common purpose, or competitive where they try to maximise their own rewards against the other agents. The agents may have a single shared policy, or multiple policies.

Although single agent DRL in PM made advanced contributions, there remain limitations such as the inability to manage diverse portfolios. To overcome the issues, MARL in PM is introduced to allow multiple different agents to each manage a portfolio. As the topic is relatively new, among the few popular literature are MSPM [1] and MAPS [4].

MSPM aims to train different agents independently as modules, where each of them manages different sets of portfolios. There are two types of modules involved, which are the portfolio-managing DQN (a) Evolving Agent Module (EAM), and the decision-making PPO (b) Strategic Agent Module (SAM). This solved the reusability and scalability issues as EAMs can be changed and connected to SAM in different combinations. On the other hand, MAPS is a cooperative MARL system implemented with the DQN algorithm. This project focused on the simulation of maximising both the overall and individual returns of a group of independent investors.

In terms of reward functions, MSPM tuned it for SAM to accommodate for training data biases and exposure of high volatile stocks, which is known as the risk-adjusted rate of return. Next, MAPS introduces a global loss function which adjusts the rewards appropriately by observing the independent rewards of all other agents, which successfully promotes collaboration. As for features, on top of the stock prices, which are commonly used as training data in the single agent DRL models, both MSPM and MAPS added the news sentiment as an additional feature, which predicts the status of the market (i.e., bear or bull market) based on reports associated with a certain time step.

Both MSPM and MAPS measure performances with average or cumulative daily returns. Other than that, MSPM also evaluates models with the Sortino ratio and maximum drawdown, whereas MAPS uses Sharpe ratio. In the case of MSPM, while it is performing better than the single agent ARL [7] in terms of daily returns, MSPM does not necessarily achieve better stability. For MAPS, It is also worth noting that it achieved a better Sharpe ratio when the resulting daily return is similar to other experimented models, for instance, convolution NN (CNN) and dual stage recurrent NN (DA-RNN).

III. Research Hypothesis

A. Problem Framing

The majority of literature in MARL portfolio management does not cover the rationale for making certain experiment design choices, specifically the choice of algorithms used to train agents. While dependent on the research question at hand, RL model design choices generally become more difficult in the multi-agent case compared to the single-agent case as there are more implementation aspects to consider following the MDP framework. Moreover, these model design choices often contribute to the overall tractability of the solution.

From the literature discussed in *Section II*, it is also common for MARL portfolio management models to utilise a cooperative setting for agents, where agents optimise a joint reward function to maximise overall returns. In this case, cooperative MARL settings have been used in portfolio management to model the collaborative interactions of investors working in the same firm where the goal is to maximise the firm's cumulative returns. One area we explore is competitive MARL portfolio management models, which can be analogous to competition between different investment firms.

Our research focuses on two areas that have not been thoroughly discussed or explored in existing literature:

1) evaluating and reviewing the effect of different algorithms in a MARL model
2) analysing the behaviour of MARL models in both competitive and collaborative settings

B. Research Approach and Goal

We take a general approach to construct a simple model where we can evaluate changes in training algorithms and hyperparameters. In this case, performance will be evaluated based on the risk of the agent's portfolio measured by the Sharpe Ratio and the overall returns of the portfolio. An overview of our MARL model takes inspiration from [8] and utilises a simple model with three agents, managing a portfolio containing one high-volatility stock and one low-volatility stock. Since most stock portfolios consist of any combination of high-volatility and low-volatility stocks, these two-stock portfolios would represent a reduced model of an actual portfolio. Further details of our simplified model will be explained in *Section IV*.

Our research investigates whether changing the type of training algorithm affects the results produced in MARL collaborative and competitive cases. We take a structured hierarchical approach involving a series of experiments. We approach this in two parts; first to show the effectiveness of our MARL implementation:

1) Show that benchmarks such as 'buy low and sell high' and 'constant rebalancing' perform well (demonstrated in a single agent setting)
2) Show that agents in our MARL model can learn naive trading strategies such as 'buy low and sell high' based on cosine periodic time series (demonstrated using mock data)
3) Show how agents in our MARL model (that learn using training algorithms) behave (demonstrated using mock and real data)

Second, to show and compare the results obtained by the different training algorithms in both competitive and collaborative cases:

1) In a collaborative MARL setting, we compare the results yielded by the training algorithms (A2C, DDPG and PPO) based on key metrics such as risk, volatility and return.
2) In a competitive MARL setting, we compare results yielded by the training algorithms (A2C, DDPG and PPO) based on key metrics such as risk, volatility and return.

IV. Experiment Design

The following subsections describe the system architecture, development process, and chosen design decisions.

A. Methodology

1) Environment Development:

Before reaching our current solution, different libraries and approaches are experimented in 3 main stages to build the research environment.

Stage 1: Kaggle Tutorial - To get an idea of how RL works in portfolio management, we started by exploring a tutorial available on Kaggle (here) and tried to understand the basic concepts and logic behind it. We then progressed by naively introducing multiple agents of different policies (i.e., A2C, DDPG, and PPO) to make all the agents interact with the same environment with the help of the stable-baseline3 library [13].

However, as the agents were merely observing the environment and their own rewards without any penalty factors, we suspected that the agents are not interacting and affecting each other. Thus, we moved on to research possible different solutions and experimented with the Petting Zoo library [14].

Stage 2: Petting Zoo - By referencing the same code, we ported the code from the single agent Gym [15] environment to the multi agent Petting Zoo environment [14]. This second experiment is done with the AECEnv from Petting Zoo, which models sequential actions. Internally, Python dictionaries are used to define the required attributes and spaces. This is to ensure that each agent has their own copy of spaces and will not cause unnecessary updates to another agent's spaces.

The issue is that while the current agent is now observing the rewards of the other agents in order to cast a penalty factor, they are all updating their respective portfolios through the same set of actions determined by the current agent. This is

not ideal as it is more realistic for each agent to decide their own sets of actions and generate their rewards. Moreover, the current training process requires an agent to complete its learning for *n* timesteps before starting the training of the following agents. This is problematic because the training is sequential and the weights of the other agents are not updated until the learning of the current agent is completed.

Stage 3: Final Custom Gym Environment - We aim to create a custom Gym environment for the research. The Petting Zoo structure is inherited, where the attributes and spaces are recorded in the form of Python dictionaries. This ensures that each agent has their own copies of states and spaces. Improvements are also made to the learning and stepping methods to fit our objectives. To simulate a more realistic system, the default one-off training of n timesteps is replaced with a loop to mimic the parallel training process. At each timestep *t*, the training of an agent will not progress to $t+1$ until all other agents have completed their respective weights update at *t*.

Moreover, for the agents to have their own independent actions and ratio of stocks investment, a set of predictions are collected before the start of a time-step so that the training actions are consistent. For example, let there be 3 agents, all actions $\{a_1,\ a_2,\ a_3\}$ for time-step t is collected first so that a prediction is not accidentally made after an agent has been trained, such as, $\{(a_1)_{t+1},\ (a_2)_t,\ (a_3)_t\}$ which resembles sequential trading instead of parallel. Finally, at the end of each loop, the calculated rewards for all agents are stored in memory. Before returning the final rewards, penalty factors are applied, where the heuristics depend on the type of environments as explained in *Section IV-A2*.

2) Environment Structure:
Types of Environments - We have created two main types of environments: cooperative and competitive. In the cooperative setting, agents work together to increase individual and collective returns. On the other hand, in the competitive setting, agents work against each other to get the highest returns. While both our environments are essentially the same, the rewards function is the critical differentiator between the cooperative and competitive environments. Hence, by changing the rewards function we have changed the method by which agents interact and behave. Further details regarding the rewards functions are described in *Section IV-A3*.

Furthermore, within each environment, we define 3 interacting agents. These agents all learn using the same specified learning algorithm. For both cooperative and competitive settings, we switch the algorithm used by the agents to observe and compare the performance of different algorithms in each setting. Details regarding the type of learning algorithms used are described below. Note each environment has two stocks. A summary of these different environments is shown in *Table I*.

Table I
BASE ENVIRONMENTS

Environment	***Type***	***Algorithms***
0	Cooperative & Competitive	Random Actions
1	Cooperative	A2C
2	Cooperative	DDPG
3	Cooperative	PPO
4	Competitive	A2C
5	Competitive	DDPG
6	Competitive	PPO

State & Action Space - A state can be defined as an agent's observation (and, therefore, the situation or position the agent is in) before deciding. In our environment, the state is the value (and technical indicators) of the two stocks at a daily resolution. An agent observes this state and the rewards provided to take action using its learning algorithm. An action can be defined as the decision taken by the agent. In our environment, agents' actions are defined as a list of values between 0 and 1 (one value for each stock). These represent the percentage amount of each stock the agent holds. Hence, at each step, the agent implicitly buys, sells or holds, changing its allocation and providing the updated list of values. These are then used to update the portfolio value (and calculate the next reward).

Agent Structure & Learning/Training Algorithms - The environment consists of 3 agents that are either working collaboratively or competitively. Each agent aims to learn the best strategy to gain the highest return and Sharpe ratio. To do this, all agents use a single learning algorithm to choose actions based on the previous state and rewards obtained. In our environments the following learning algorithms are used: A2C, DDPG and PPO. We have also created agents that take random actions to use as a baseline.

3) Rewards & Penalty Factors:
Rewards are used to guide agents towards taking good actions and avoiding bad ones in the learning process. They are also used to create implicit interactions between agents. The types of reward functions used in our environments are described below.

Rewards within the cooperative environment - Initially, the reward of each agent is set to the portfolio value it obtains after taking the action it predicted using the learning algorithm. To make the environment cooperative, the reward for each agent is adjusted. Specifically, by comparing the rewards obtained by each agent, the system aims to provide high rewards when agents: individually get differing rewards and collectively get large total rewards. Initial agent rewards are adjusted to account for this. By implementing this, agents are encouraged to implicitly interact and collaborate to try and get the highest overall returns. A significant negative reward for obtaining negative returns is introduced to discourage an agent from taking undesirable actions.

$$Reward = DRR_{curr} - |DRR_{curr} - DRR_{mean}| \quad (1)$$

where DRR_{curr} is the daily rate of return of the current agent, and DRR_{mean} is the average daily rate of return of all agents.

Rewards within the competitive environment - Initially, the reward of each agent is set to the portfolio value it obtains after taking the action it predicted using the learning algorithm. To make the environment competitive, each agent's initial reward is adjusted. Precisely, we adjust each reward by multiplying it with a ratio of rewards calculated using rewards from the other agents. By implementing this, agents are encouraged to interact and compete to try and outperform each other. A similar negative reward for negative returns is also added here.

$$Reward = DRR_{curr} + \sum_{i}(DRR_{curr} - DRR_i), \quad \forall i \in agents, i \neq curr \tag{2}$$

Table II
ASSUMPTIONS AND REASONING

Assumption	***Reasoning/Impact***
Agents have no direct impact on each other	Agents only implicitly interact through the rewards function
Agents are in sync and take actions in parallel	Since the stock data is at a daily rate, we do not consider the order agents trade in
Agents are trading daily and only once a day	In practice they could trade multiple times in a day
Tradings have zero transaction cost	Agents do not consider external costs such as commission fee

4) Model Assumptions:
While these assumptions (in *Table II*) in some sense reduce the system's practicality, they allow us to provide a simple proof of concept that can easily be extended and modified in the future. These future extensions are discussed in *Section VI*.

B. Dataset

1) Volatility: Volatility is the rate at which stock prices increase and decrease over a period of time. A higher volatility means higher risk overall for the investors of that stock, and a lower volatility represents lower risk. This is a useful feature as investors are able to estimate the fluctuations knowing the volatility, and it affects their decision in investing in a particular stock. The calculation of volatility is done by taking the standard deviation of a stock's annualised returns over a given period of time and shows the range of possible fluctuations, as shown here:

$$\sigma_{annually} = \sigma_{daily} \times \sqrt{252} \tag{3}$$

2) Mock Dataset: The experiments started off with mock datasets of different made-up stock combinations. This is to simplify observations and arguments because the trends are tailored manually, allowing greater control. In order to show that the model is able to learn naive trading strategies such as *buy low and sell high*, a set of data based on the upward trend of sine wave movement. For our purpose of generating a portfolio of two stocks, we also created its complement dataset: if the price of the original stock is going up, the price of the complementary stock is going down, and vice versa. Using this, we can also prove that the model can learn to increase the position on one stock and decrease it on the other for two complementary stocks.

With *Numpy*'s built-in function, sine graphs of varying noise and adjusted variances are generated to represent the mock data. In addition to this, using the ratio between the daily price and the monthly average price (computed over the last month from that day), we produced the daily traded volume for the mock dataset. The graph below (on the right) in Fig. 1 shows one of the mock stocks, along with its complement.

3) Real Dataset: On top of that, a dataset with real stocks is prepared for an additional experiment. It is made up of a low volatility (35.19%) Apple Inc. (AAPL) stock and a high volatility (68.13%) Tesla Inc. (TSLA) stock. This promotes simplicity as there are only two different stocks, and diversity as the stocks are of different volatilities.

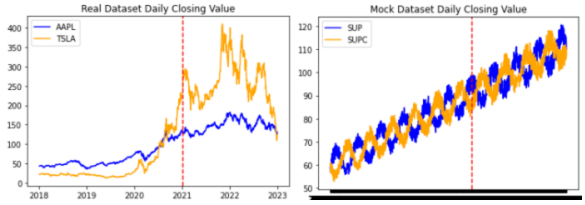

Figure 1. Result of Benchmark algorithm on the mock dataset

4) Train-Test Split: For better model evaluation, the datasets are split into two main portions: Training: 1 Jan 2018 to 31 Dec 2020 (3 years), Testing: 1 Jan 2021 to 31 Dec 2022 (2 years). This helps in determining if the model has been overfitted as non-overlapping dataset is used for the different phases. Due to the limited project duration, a relatively small dataset is used for training for ensuring a reasonable training time whilst avoiding excessive payoff of the training quality.

C. Metrics

Cumulative Rate of Return (CRR) measures the total return on an investment over a specified period of time. It is calculated using the initial investment and adding returns earned over an investment period. This measure is utilized in our experiments as it gives us a comprehensive view of the performance of each investment made by an agent over timesteps.

$$Cumulative\ Rate\ of\ Return = \frac{Closing\ Value}{Beginning\ Value} - 1 \tag{4}$$

The *Sharpe ratio* is a measure of risk-adjusted return on a portfolio and is widely used in finance as a way to evaluate the performance of an investment relative to its risk. In our experiments, the Sharpe ratio is calculated when agents reach the end of their trading period. Generally, a Sharpe ratio of 1 and above is favorable as it indicates that the investment

of a portfolio is generating a sufficient excess returns to compensate for the risk taken.

$$Sharpe = \sqrt{252} \times \frac{R_{portfolio} - R_{riskfree}}{\sigma_{portfolio}} \quad (5)$$

D. Benchmarks

After the algorithm has been created, trained, and evaluated. Some of the standard algorithms should be used to compare the matrix of the portfolio including return, risk-adjusted return, and variance of the portfolio. We are choosing the following algorithm as a benchmark. The first algorithm is buy and hold. The algorithm starts by choosing the initial proportion of the asset and holding the asset without rebalancing the portfolio. The second algorithm is constant rebalancing. This algorithm will set the holding ratio between each asset. After a certain period, the stock will be rebalanced in order to maintain the desired ratio between each asset. In our experiment we chose to rebalance the asset in every 30 days. Figs 2 and 3 illustrate the cumulative return over the certain period of time in both real and mock datasets.

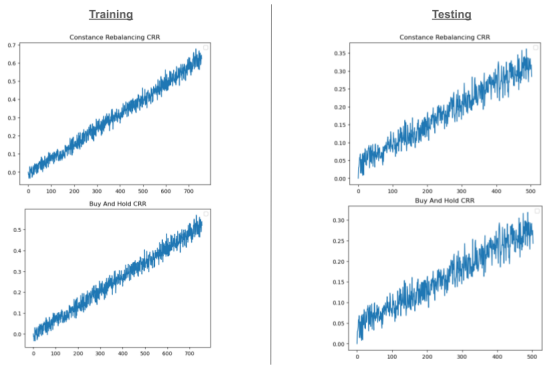

Figure 2. Result of Benchmark algorithm on the mock dataset

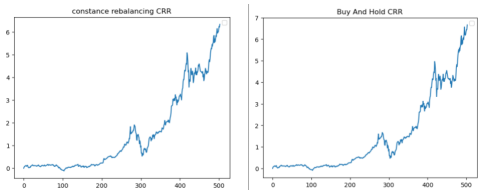

Figure 3. Result of Benchmark algorithm on the real dataset

V. Analysis of Results

To analyse, compare and review the behaviour of agents in our environment, we are using different plots with day number on the x-axis and CRR on the y-axis. These plots demonstrate how the returns of the agents grow over time. Pairs of plots are included:

1) initial performance: agents are executed for a complete step at training iteration = 0 to observe the behaviours before training and allow comparison.
2) trained performance: this is plotted after training iteration = 25 so that the CRR and behaviour after training is observed.

These pairs of plots are used to clearly show the emergence of competitive or cooperative behaviour among agents over training timesteps.

In general, for a cooperative setting, agents should work together and all achieve similar high returns. As for a competitive setting, we expect ordering between agents where each agent performs better than another. These patterns were analysed in the graphs and the best-performing algorithm is chosen based on the level of exhibition of competitive or cooperative behaviour. Then, the performance of trained agents to random ones were compared. Overall, it was noticed that changing the type of training algorithms definitely affects the behaviour of agents in the environment and that certain algorithms may be superior depending on the specific use case. Due to time constraints, we were unable to conduct a full detailed analysis of each algorithm and compare complex statistics for considering the reasons behind these differences (this includes testing for the real data set) - this is something we will definitely like to consider in the future.

A. Mock Data - Cooperative Environment Results

1) Training Data Set Results:

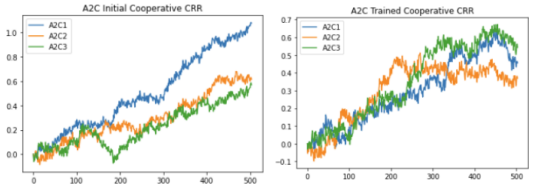

Figure 4. Result of Cooperative MARL with the A2C algorithm on the mock training dataset

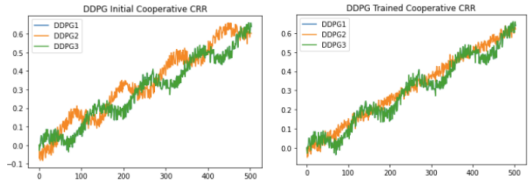

Figure 5. Result of Cooperative MARL with the DDPG algorithm on the mock training dataset

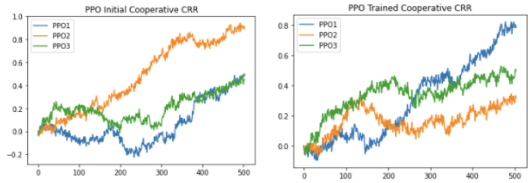

Figure 6. Result of Cooperative MARL with the PPO algorithm on the mock training dataset

2) Test Data Set Results:

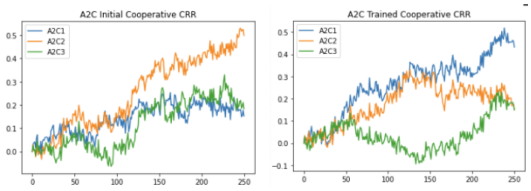

Figure 7. Result of Cooperative MARL with the A2C algorithm on the mock testing dataset

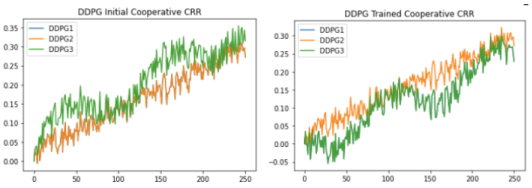

Figure 8. Result of Cooperative MARL with the DDPG algorithm on the mock testing dataset

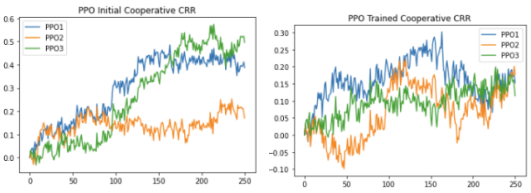

Figure 9. Result of Cooperative MARL with the PPO algorithm on the mock testing dataset

3) Best Performing Algorithm:

To demonstrate cooperative agent behavior, we are looking for a graph pattern that converges at the end, meaning all agents end with similar returns and there is no clear winner or loser. This elucidates the collaborative nature of agents to increase rewards collectively. Here, both A2C and PPO demonstrate a level of cooperation among agents. For A2C, we see a more consistent exhibition of cooperative behavior in both the test and training set as there will always be two agents ending with similar returns as seen through convergence of two coloured lines in Fig. 4 and Fig. 7. However, it is worth noting that PPO only shows strong collaborative behavior in the test set with all agents converging to a Sharpe ratio of approximately 0.3 and a value of 0.15 on the y-axis (CRR) in Fig. 9.

As for DDPG, we believe that it is the worst performing algorithm here, where it can be seen from Fig. 8 that there are multiple agents learning and taking the exact same set of actions. The underlying reason may be due to the fact that it is an off policy algorithm, where instead of learning a policy through the current action, the policy learns from the best available action, which may be different from the current selected action.

Some unexpected behaviour in the model was spotted in agents using the DDPG algorithm. Here two of the agents were performing exactly the same actions. The first reason for this behaviour might be caused by the combination of simple reward functions and the DDPG algorithm. Since the reward function currently did not explicitly encourage agents to behave differently it may cause agents to sometimes take the same actions. The second reason could be the model overfitting. The third reason might be some property of the DDPG algorithm since this behaviour is not observed in the other algorithms that use the same rewards function.

B. Mock Data - Competitive Environment Results

1) Training Data Set Results:

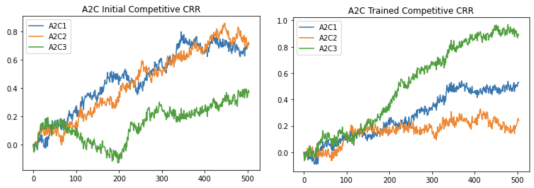

Figure 10. Result of Competitive MARL with the A2C algorithm on the mock training dataset

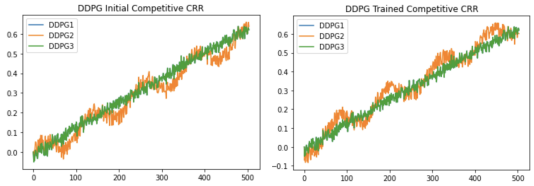

Figure 11. Result of Competitive MARL with the DDPG algorithm on the mock training dataset

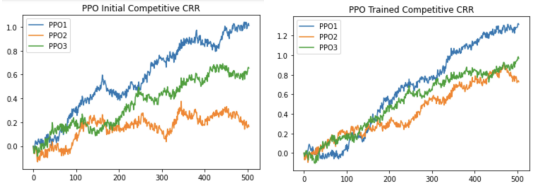

Figure 12. Result of Competitive MARL with the PPO algorithm on the mock training dataset

2) Test Data Set Results:

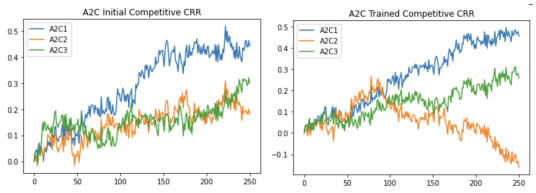

Figure 13. Result of Competitive MARL with the A2C algorithm on the mock testing dataset

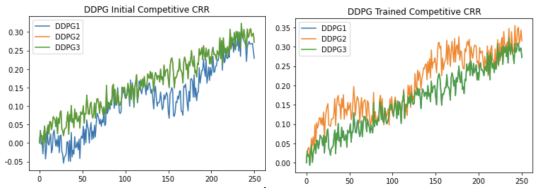

Figure 14. Result of Competitive MARL with the DDPG algorithm on the mock testing dataset

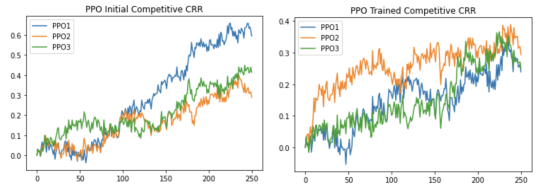

Figure 15. Result of Competitive MARL with the PPO algorithm on the mock testing dataset

3) Best Performing Algorithm:

We believe that A2C is the best-performing algorithm. This is because it learnt the competitive behaviour for both the training and testing datasets as shown in Figs 10 and 13. Moreover, A2C successfully boosted the CRR of the most competitive agent to a value close to 1.0 as shown in Fig. 10, which is a 0.19 gain of the initial CRR of 0.8. On the other hand, it maintained its maximum CRR at approximately 0.5 with the testing dataset as shown in Fig. 13. As for Sharpe Ratio, in the training phase, the post-training value hits 0.92 for the most competitive agent, which is higher than the initial ratio of 0.38. For the testing phase, it also achieved Sharpe ratio of 0.98 if compared to the initial 0.90.

While PPO seems to also perform well in the training phase, it failed to maintain its performance with the testing dataset. From Fig. 12, it is observed that it exhibits slight competitive behaviour. However, when we moved on to testing, the behaviour disappeared and resembled more towards the cooperative behaviour as shown in Fig. 15. Lastly, DDPG is the worst performing algorithm due to similar reasons in Section V-A3, where it is an off policy algorithm, which made it failed to learn a more diverse set of actions.

C. Real Data - Cooperative Environment Results

1) Training Data Set Results:

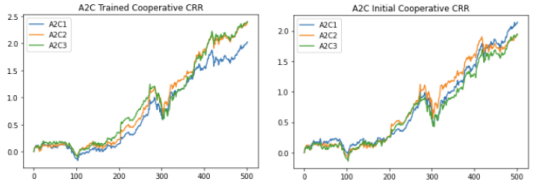

Figure 16. Result of Cooperative MARL with the A2C algorithm on the real training dataset

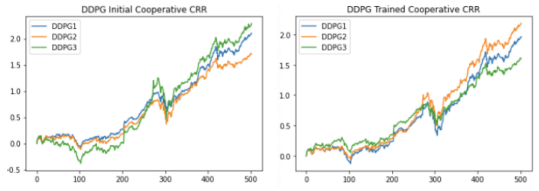

Figure 17. Result of Cooperative MARL with the DDPG algorithm on the real training dataset

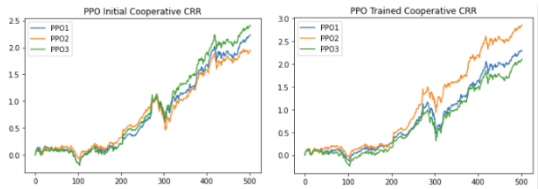

Figure 18. Result of Cooperative MARL with the PPO algorithm on the real training dataset

2) Best Performing Algorithm:

A2C performed the best because it showed the most cooperative behavior as two out of three agents have the same CRR value at the end in Fig. 16. An interesting observation is that A2C yields similar behavior in the mock and real dataset, with two agents consistently matching the CRR of the other. These two collaborating agents also increased their Sharpe ratio for about 0.4, which indicates smaller risk in the returns. However this is not the case for DDPG and PPO, all agents ended with a distinct ordering with different levels of CRR in Figs 17 and 18, where their CRR and Sharpe ratio also remained relatively the same as pre-training.

D. Real Data - Competitive Environment Results

1) Training Data Set Results:

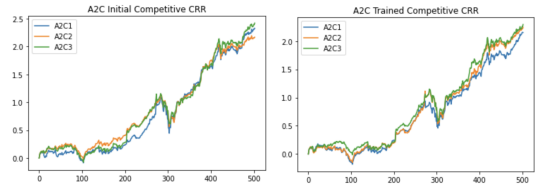

Figure 19. Result of Competitive MARL with the A2C algorithm on the real training dataset

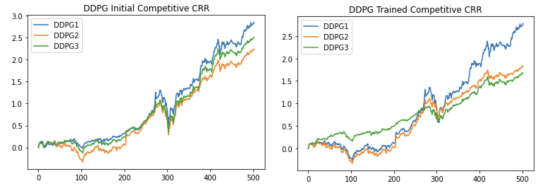

Figure 20. Result of Competitive MARL with the DDPG algorithm on the real training dataset

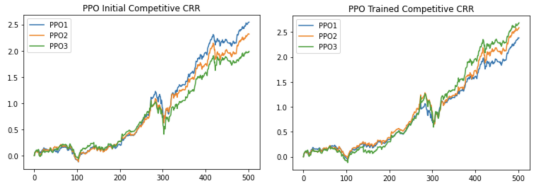

Figure 21. Result of Competitive MARL with the PPO algorithm on the real training dataset

2) Best Performing Algorithm:

Here we believe that DDPG is the best-performing algorithm. This is because it exhibited the competitive behaviour most obviously. It can be observed from Fig. 20 that the agent represented by the blue line is the most competitive as it achieved a marginal improvement in CRR compared to the other agents in the environment. In terms of metrics, all agents managed to maintain their CRR at approximately 2.2 and Sharpe ratio within a range different of 0.19.

While the other algorithms, A2C and PPO, are showing similar performances, they resemble the cooperative behaviour more as the graphs of their CRR are rather close to each other and not showing a competition between the agents (Figs 19 and 21). As per the figures, the CRR remained at a value of about 2.5 for both algorithms.

VI. Discussion & Limitations

This section provides possible areas where the model or the environment can be extended to make it more realistic and practical.

Increase number of stocks - In our environment, the portfolio was built using only two stocks. While this was useful for testing at a proof of concept stage, the model should include more stocks in practice. This implies that each agent will be required to make more decisions and implicitly learn more complex strategies.

Increase complexity of action space (introduce risk probability) - In order to simulate a more realistic scenario, the agents should have the possibility to invest a partial amount of the initial funds, instead of requiring the full amount to be placed in stocks. This would allow the agents to avoid unnecessary risk.

Include market impact - In the experiment report environment, the states were static due to the assumption of zero market impact. However, as the resolution and the amount of trades increased, the impact should have been considered. The augmented asset feature introduced in [11] could be included and investigated further in the future so that the stock price is also forecasted at the same time for improving predictions and rate of returns.

Include transaction costs - By incorporating transaction costs in the model, the agents may change their behaviour significantly. Adding this feature would impose a restriction on the number of actions performed by the agents, in the form of a penalty in the reward function for excessive trading.

Increase complexity of reward functions - In the current environment, a simple and naive rewards function is used to guide the agents on how they interact and implicitly set the environment as cooperative or competitive. a number of potential modifications can be applied to the rewards functions to improve their effectiveness. Some examples of additions include: adding noise, daily Sharpe ratio, historical rewards, or some technical indicators. These allow the agent to get better overall guidance and learn better and more complex strategies.

Increase training time and dataset size - The environment uses a small dataset consisting of 2 stocks and only considers a small time frame with limited training timesteps. In addition, the stock data at a daily resolution being used to conduct the experiment may not give enough information to the agents. The quantities were limited owing to time and resource constraints. However, these quantities can all be increased to improve the complexity of the model.

VII. Conclusion & Future Work

A. Summary

The research collated the performance of using different algorithms: PPO, DDPG, and A2C algorithms in a cooperative setting where the agents maximise rewards collectively, and a competitive setting where the agents are trying to outperform each other by maximising their own rewards. The experiments were conducted on real and mock datasets, where evaluations are performed using the Sharpe ratio and CRR. As the Sharpe ratio of agents in each algorithm did not show significant difference, we evaluated the performance of algorithms based on the emergence of competitive or cooperative behaviour, shown by the graph trends of CRR among agents. With the mock dataset, it was revealed that the A2C and PPO algorithms were the leading algorithms for competitive and cooperative settings respectively. Although the agents in the experiment did not outperform the benchmarks in the noisy real stock data, DDPG and A2C were the best-performing algorithms for competitive and cooperative settings. Overall, the choice of training algorithms and environment settings have an impact on results, and these hyperparameters should be carefully chosen.

B. Future Works

To improve the models we have created, in the future, we would like to develop ideas mentioned in *Section VI*.

C. Code

The link to our GitHub repository is in references [16].

REFERENCES

[1] Z. Huang and F. Tanaka, "Mspm: A modularized and scalable multi-agent reinforcement learning-based system for financial portfolio management," *Plos one*, vol. 17, no. 2, e0263689, 2022.

[2] Y.-J. Hu and S.-J. Lin, "Deep reinforcement learning for optimizing finance portfolio management," in *2019 Amity International Conference on Artificial Intelligence (AICAI)*, IEEE, 2019, pp. 14–20.

[3] Y.-C. Lin, C.-T. Chen, C.-Y. Sang, and S.-H. Huang, "Multiagent-based deep reinforcement learning for risk-shifting portfolio management," *Applied Soft Computing*, vol. 123, p. 108 894, 2022.

[4] J. Lee, R. Kim, S.-W. Yi, and J. Kang, "Maps: Multi-agent reinforcement learning-based portfolio management system," *arXiv preprint arXiv:2007.05402*, 2020.

[5] B. Li and S. C. Hoi, "Online portfolio selection: A survey," *ACM Computing Surveys (CSUR)*, vol. 46, no. 3, pp. 1–36, 2014.

[6] K. Decker, K. Sycara, and D. Zeng, "Designing a multi-agent portfolio management system," in *Proceedings of the AAAI Workshop on Internet Information Systems*, vol. 60, 1996.

[7] Z. Liang, H. Chen, J. Zhu, K. Jiang, and Y. Li, "Adversarial deep reinforcement learning in portfolio management," *arXiv preprint arXiv:1808.09940*, 2018.

[8] O. Jin and H. El-Saawy, "Portfolio management using reinforcement learning," *Stanford University*, 2016.

[9] Z. Jiang, D. Xu, and J. Liang, "A deep reinforcement learning framework for the financial portfolio management problem," *arXiv preprint arXiv:1706.10059*, 2017.

[10] J. Patani, S. Nair, K. Mehta, A. Sankhe, P. Kanani, and D. Sanghvi, "Financial portfolio management using reinforcement learning," *International Journal of Advanced Science and Technology*, vol. 29, pp. 9740–9751, Jan. 2020.

[11] Y. Ye, H. Pei, B. Wang, *et al.*, "Reinforcement-learning based portfolio management with augmented asset movement prediction states," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, 2020, pp. 1112–1119.

[12] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[13] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *The Journal of Machine Learning Research*, vol. 22, no. 1, pp. 12 348–12 355, 2021.

[14] J. Terry, B. Black, N. Grammel, *et al.*, "Pettingzoo: Gym for multi-agent reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 15 032–15 043, 2021.

[15] G. Brockman, V. Cheung, L. Pettersson, *et al.*, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[16] C. G. 02, *Marl in finance*, https : / / github . com / interritus141 / COMP0031 - Group - Research - Project, 2023.