Berkeley
UNIVERSITY OF CALIFORNIA

Capstone Project Report

Using Reinforcement Learning and Multi-Agent Simulator to Improve the Profitability and Efficiency of High-Frequency Trading

FinTech Team D2

Team Member:

Charlotte Jin,

Ethan Choukroun,

Jingqi Ma,

Haoyang Wang

April 30, 2023

Report Outline

1. Executive Summary

Reinforcement learning is a machine learning method that finds strategies to maximize the reward. Because reinforcement learning can take advantage of a large amount of data and training itself without human interruption during the process, it is highly adaptable to high-frequency trading, which is widely used in recent years due to the rapid increase of market volume and scale. Our project focuses on the simulator which can create an artificial financial market to train reinforcement learning models. In our capstone, a novel agent-based simulator called Agent-Based Interactive Discrete Event Simulator (ABIDES)[1] was used, which allows agents to interact with each other by affecting the overall market stock price through interactive simulated transactions.

Our primary achievement is the integration of Preferred Provider Organizations (PPOs), Long-Short Term Memory (LSTM)[2] and ABIDES[3]. In Particular, we introduce LSTM into the modeling process because while traditional reinforcement learning follows the Markov decision process, which assumes that the current state is only related to the previous state, financial markets are actually influenced by multiple factors. Besides, the asset prices also exhibit periodic trends. Considering this fact, we believe that LSTM networks will allow our model to remember the most relevant previous and current information while discarding irrelevant data. Additionally, we use neural networks to define the state space[4], removing human intervention and including only the most important market elements to improve accuracy.

Our future research will focus on refining our model. One limitation of our capstone is that the current action space is limited to simple actions such as buy, sell, and hold. We plan to make these actions more complex to better simulate financial agents. Furthermore, we aim to explain how our neural networks select states to uncover the underlying logic of these selections.

1.1 Introduction of the financial market with increasing trading volume and frequency

Ever since 2000, we have witnessed a soaring increase in trading volume. According to data from New York Stock Exchange (NYSE), trading volume grew by over 350% between 2002 and 2008[5]. (Fig 1) Compared to the market before the 21st century, the financial market with larger volumes became more complicated and varied for agents to trade and analyze. As a result, trading strategies must evolve in order to be efficient and accurate in the more complicated environment of today' s market.

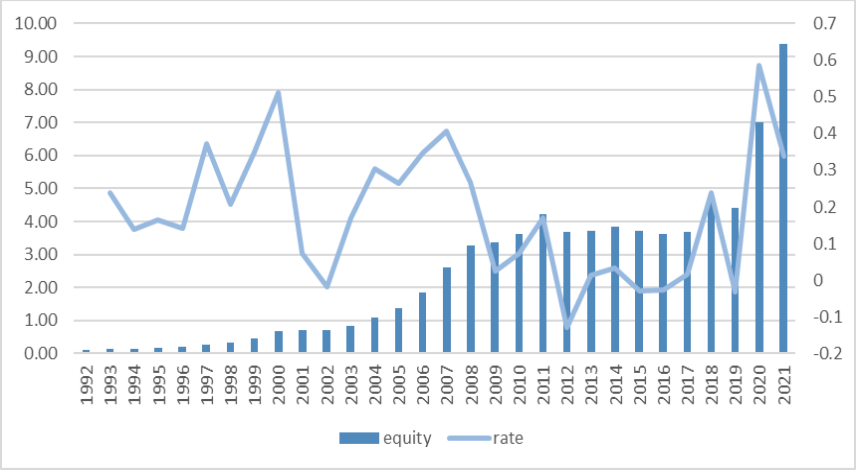

Fig 1. Trading Volume and Growth Rate

At the same time, with the improvement of calculating speed and dimension, computer-based trading methods were introduced where computer science was integrated with financial trading. Among them, high frequency trading (HFT), which is characterized by the high speed of orders, high turnover rates, and high order-to-trade ratio, came to our attention. Around 2004, HFT had an execution time of several seconds, whereas by 2010 this had decreased to milli- and even microseconds, which makes the trade even faster[6].(Fig 2)

Traditional analyzing methods cannot effectively leverage large amounts of data in a highly complex market. Some suggested using machine learning by developing algorithms that can learn from the previous trading data and make predictions or decisions without being explicitly programmed.

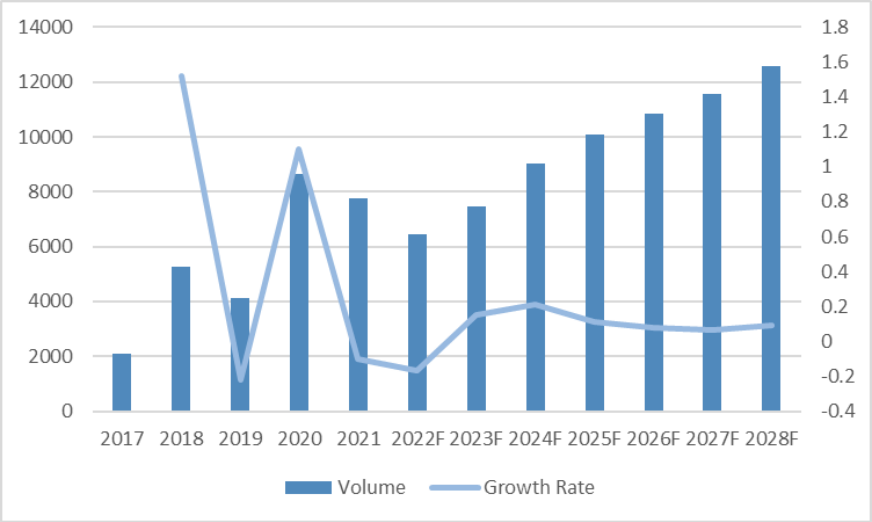

Fig 2. HFT Volume and Growth Rate Prediction

One of the challenges in traditional machine learning is that training these algorithms requires large quantities of labeled data. Additionally, traditional machine learning algorithms are limited in their ability to learn from experience and adapt to changing situations. Reinforcement learning is a subfield of machine learning that addresses these limitations by allowing algorithms to learn from self-exploring experiences and deal with the ever-changing environment. As shown in Fig 3, in reinforcement learning, an agent interacts with an environment and receives feedback in the form of rewards or penalties based on their actions. The goal of the agent is to learn a policy that maximizes its cumulative reward over time[7].

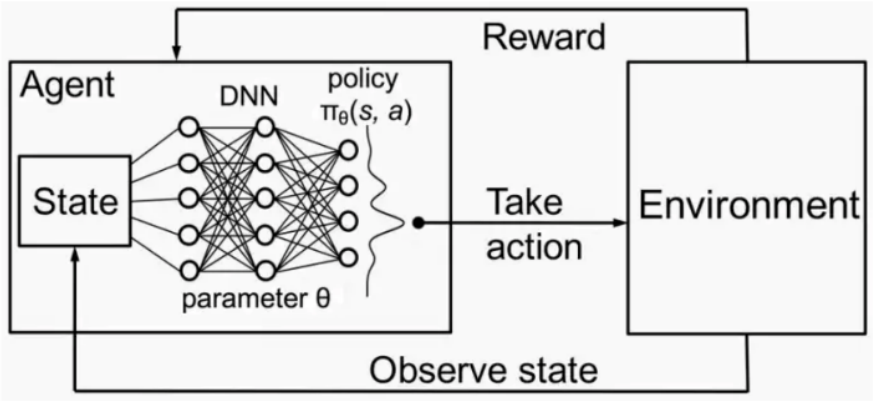

Fig 3. Mechanism of Reinforcement Learning

However, it is expensive to model the states and actions transition steps. Our project focuses on how to successfully simulate the whole financial market which not only guarantees accuracy but also can be easily modeled. We use the OpenAI Gym framework to simulate a direct environment for reinforcement learning. On top of that, what distinguishes our work from others is that we'll also use Agent-Based Interactive Discrete Event Simulator (ABIDES), to separate each agent, and consider this agent's own Markov Decision Process (MDP).

Besides, each agent interacts with others via "messages", which is used to represent actual interactive actions among different agents through the form of changing holdings. Each agent takes actions that will be sent to the environment and returns a reward and new state. In this process, ABIDES is combined with OpenAI Gym [8], in which agents interact individually through a message system. Fig 4 shows the linkage between ABIDES and OpenAI Gym. Each agent only has his or her own state and information about the rest of the world.

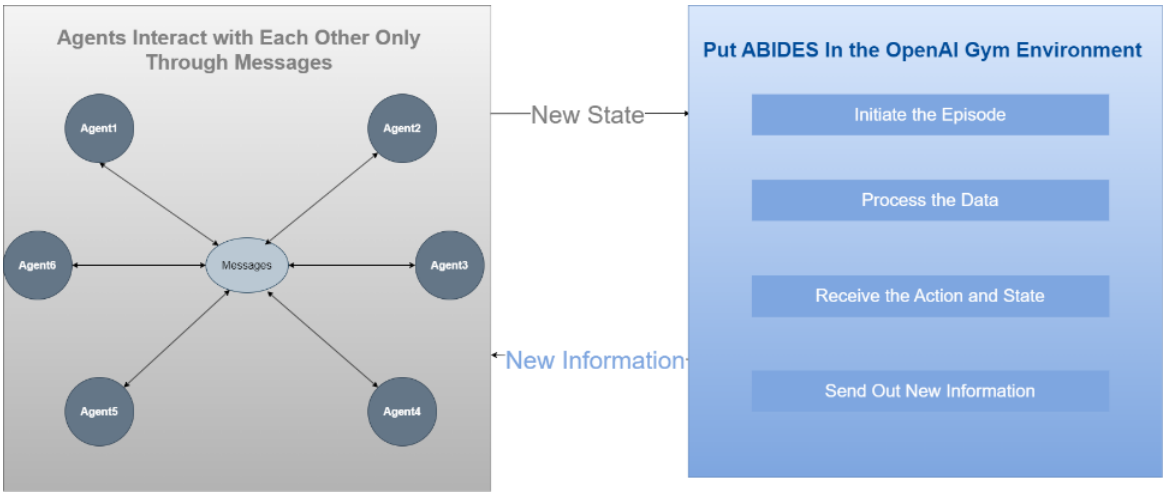

Fig 4. Mechanism of ABIDES

1.2 Capstone Objective: increase trading policy efficiency and profitability

Our project aims to utilize reinforcement learning in the financial market by using ABIDES Gym to simulate the market and agents to train machine learning models. We introduce some new methods, such

as PPO and LSTM to enhance the performance of the reinforcement learning models, with the ultimate goal of improving the efficiency and profitability of trained optimal trading strategies for both the market and financial agents.

Our result showed a significant improvement in the profit and episode reward of the trading policy using ABIDES and Proximal Policy Optimization (PPO) compared to other trading policies. We observed a 2.38% profit margin after 700 iterations of training, which exceeds other policies which have negative profits. We also witnessed that with a higher learning rate, the trading policy we proposed performed better which is abnormal in other training processes.

In section 2, we will introduce some related work about the methods we use in this project to help readers better understand these concepts. In section 3, we will provide a detailed explanation of our baseline model, as well as our reinforcement learning configuration. In section 4, we will explain LSTM model and PPO model algorithm logics and provide reasons why they can increase the model performance. In section 5, we provide some simple analysis of the artificial financial market generated by ABIDES. In section 6, we present and evaluate the performance and profitability of trading strategies obtained from different reinforcement learning models. In section 7, a final conclusion of our findings will be presented.

2. Related work about methods used in the capstone project

2.1 Reinforcement Learning to illustrate how agents choose strategies

Reinforcement learning[9] is a method to illustrate how smart agents choose their strategy to optimize their objectives. In RL, the environment is typically stated in the form of a MDP[10]. There are two types of RL, model-based and model-free. Model-based methods assume the information needed to construct MDP, while model-free methods assume the information is unknown. To gain more information about states, actions, and rewards, agents need to interact with the environment. Simulations are leveraged to avoid the high cost of training a model through direct interactions with real-world environments.

2.2 ABIDES multi-agent simulation environment to create artificial stock market

Among plenty of simulators, ABIDES is utilized because it provides agents with their own message and information about the rest of the world. Agents are divided into the exchange agent, the agent in simulation, and market participants, other agents[1]. They communicate only by messages, which include orders and market data. However, it requires a deep understanding of the relationship between agents and environments in the financial industry. Besides, the framework is unconventional to use in currently existing models, which means that our capstone project is quite innovative.

2.3 OpenAI Gym to develop and test learning agents

OpenAI Gym is an environment for developing and testing learning agents. It is a focused and best-suited reinforcement learning agent [11]. To avoid the drawbacks of ABIDES, we use it through the OpenAI Gym environment framework. Running ABIDES as a black box and leaving the learning and MDP formulation process outside the simulator, we overcome the obstacles we met when using ABIDES alone[1]. Besides, we combine ABIDES-Gym and ABIDES-Markets to build ABIDES-Gym-Markets, an abstract base environment that has the functionality of both platforms. In addition, by leveraging open-sourceRL tools, we demonstrated that an RL agent could easily be trained using ABIDES-Gym-Markets[12].

3. Baseline Model: ABIDES-Gym Environment and Deep Q-Learning

3.1 Introduction about Deep Q Learning: feeding data in a continuous simulated market

To train RL agents to trade in the simulated market environment, we propose using the Deep Q-Learning[13] (DQN) algorithm for the baseline model, as DQN is easy to implement and can easily adapt to a simulated realistic market environment, such as the ABIDES-Gym. Large-scale and complex datasets can be directly fed into DQN for training.

More specifically, we use the simulated data as input to feed into the neural network architecture in the DQN model, where the neural networks will compute the rewards or profits the trading agents obtained for each trading action they make at different market states. To train the neural network toward our objective, we calculate the loss function by comparing the values to the Bellman equation and updating the weights of the neural network via stochastic gradient descent (SGD) and backpropagation (BP). The more training iterations we perform, the better the DQN model is good at outputting optimal values, which leads to an optimal policy in the action space.

To build a DQN model, we use a fully connected feed-forward neural network with 2 layers composed of 50 and 20 neurons. We implement an epsilon-greedy search approach decreasing from 1 to 0.02 in 10k steps to allow the agents to fully interact with the environment. The learning rate is scheduled to linearly decrease from $1 \cdot 10^{-3}$ to 0 in 90k steps.

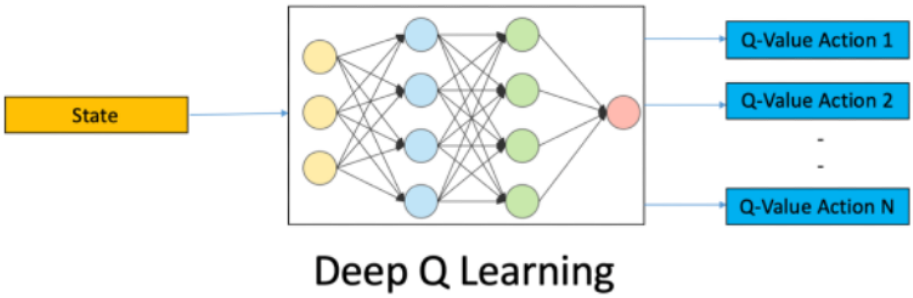

Fig 5. Representation of a Deep Q Learning model

After simulating the synthetic data and choosing a Deep RL model for the trading agents, our next challenge will be to simulate the simultaneous training process of multiple trading agents in the ABIDES environment. ABIDES is designed to support AI agent research in market applications, by enabling the simulation of tens of thousands of trading agents, which interact with an exchange agent to receive information about the market, to facilitate transactions.

3.2 ABIDES-GYM associated with Deep Q Learning

We aim to simulate our agents in the ABIDES-Gym environment which utilizes the OpenAI Gym framework, an open-source Python library used for developing and comparing reinforcement learning algorithms by providing a standard API to communicate between learning algorithms and environments. Unlike the original ABIDES, the MDP formulator is left outside of the simulator and can communicate with the agents. Thus, the user can interrupt the simulation by sending out a request, unlike the original environment in which the whole simulation plays out without interruption. When paused, the system returns the current state of a specified agent. After being processed using Kernel methods, this data can be used to run further computations or be fed into the DQN algorithm to define the next action. To use the ABIDES-Gym environment, the user can then use the runner method with action a as input to resume the simulation until the next interruption.

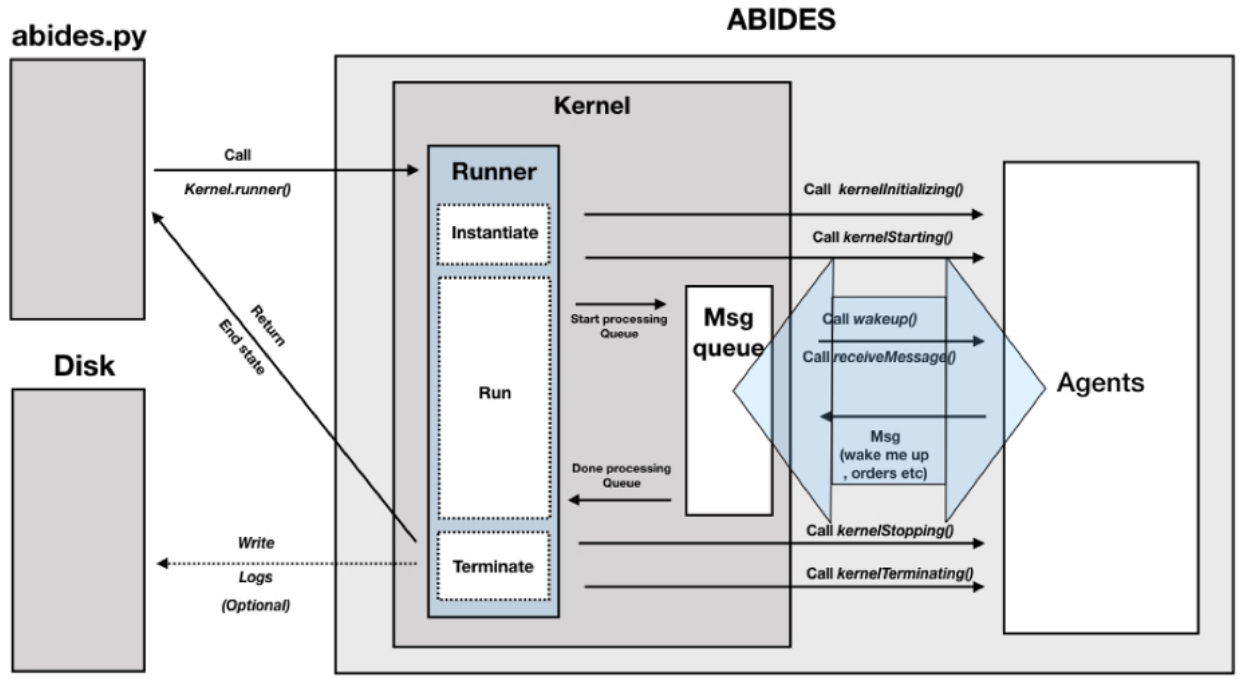

Fig 6. ABIDES-Gym Environment

This environment presents us with a way to accurately simulate our agents and have great control over the data that we use as input for our model. Using it is a great opportunity and can play a major role in the overall success of our project.

3.3 Markov Decision Process: States, Actions and Rewards

Our reinforcement learning agent operates by taking decisions in a continuous environment following a Markov Decision Process (MDP).

A Markov Decision Process (MDP) is a mathematical framework utilized in various fields, including finance and economics, to model decision-making problems. It serves as the foundation for reinforcement learning, a type of machine learning algorithm that allows an agent to learn from its experiences and make improved decisions over time[14]. In a continuous financial market, an MDP can be used to train a reinforcement learning agent to optimize trading strategies.

A Markov Decision Process is defined by a state space, an action space and a reward function which is associated with the decision taken in a given spate (t) as a function of only the last recorded state (t-1).

3.3.1 State Space

The state space comprises all possible states an agent can be in at any given time. In a continuous financial market, this space includes both private states, known only to the agent, and public market states. At each time step, trading agents perceive the market through the following states:

- $remainingTime_t = \frac{T-t}{T}$: the time remaining at time t;
- $remainingQuantity_t = \frac{M-m}{M}$: the number of stock shares remaining at time t, where M is the total amount of shares the agent must trade, and m is the amount of shares the agent has traded up to time t;
- $spread_t = bestAsk_t - bestBid_t$: the difference between the the best bid price and the best ask price at time t;
- $imbalance_t = \frac{totalBidVolume_5}{totalBidVolume_5 + totalAskVolume_5}$: the difference between the order volume on the bid side and the ask side of the first 5 levels;
- $R_t^k = (r_1, \cdots, r_k)$: a series of mid-price returns, where $r_j = log(\frac{midPrice_t}{midPrice_{t-j}})$ and $k = 5$;

- $priceImpact_t = midPrice_t - midPrice_0$: the difference between the mid price at time t and the mid price at time T_0;
- $marketOrderCost_t$: the best bid price at time t if the agent is selling or the best ask price at time$t$ if the agent is buying;
- $signedTransactionVolume_t$: the signed volume (buy orders are positive and sell orders are negative) within the past time period at time t;

3.3.2 Action Space

The action space consists of the possible actions an agent can take at each time step. In our continuous financial market context, the action space includes the following actions:

- 'Buy': The agent buys a certain quantity of shares.
- 'Sell': The agent sells a certain quantity of shares.
- 'Hold': The agent neither buys nor sells any shares.

3.3.3 Reward Function

The reward function measures the success of the agent's actions and guides its learning. In this setting, the reward function is defined as:

$$R(t) = \sum_{o \in O_t} \epsilon(a).\,(transactionPrice_{0,t} - midPrice_0).\,quantity_{o,t}/M$$

Here, O, t represents the set of executed orders at time t, $\epsilon(a)$ is 1 for buy orders and else is -1, $transactionPrice_{0,t}$ is the executed price for order o at time t, $midPrice_0$ is the mid-price at the initial time step, and $quantity_{o,t}$ is the executed volume for order o at time t.

The reward function captures the impact of executed orders on the agent's performance relative to the initial mid-price, normalized by the total number of shares the agent must trade (M).

Furthermore, we add a Reward associated to the episode to place a penalty based on the amount of uncompleted trades:

$$R(episode) = max(penalty\,.\,(M - m_{T_N}),\ 0)$$

Where $penalty = 100$ per shares

In summary, our reinforcement learning agent follows a markov decision process and each reward serves as a feedback to update the model and make it better.

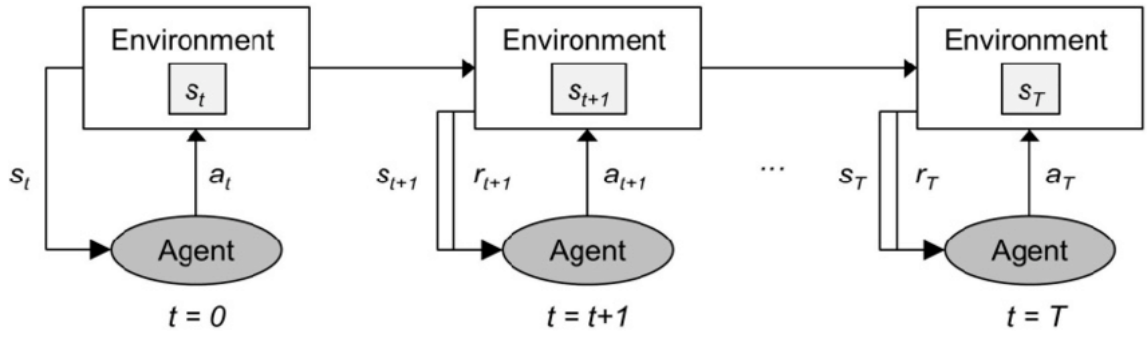

Fig 7. Reinforcement Learning Design

The agent operates in a continuous environment and his goal is to find the most optimal action to take in each state, action that will maximize its reward at time t.

4. Proposed Modeling Techniques with LSTM and PPO

In our proposed model, we first propose to add LSTM (Long Short-Term Memory) layers to train trading agents in the DQN algorithm. One limitation of the current DQN model is that it is only able to utilize the current input information and hence when it is applied to time series data, the algorithm fails to make use of agents' experience learned from previous data.

4.1 LSTM Implementation in Deep Q Learning to process time series data

LSTM is known for its remarkable performance in processing time series data. By incorporating historic information from states far in the past, trading agents can take advantage of previous experience, and therefore the predicted Q-values and the optimized trading policies will be better.

Built on classical recurrent neural networks (RNN) structure, LSTM mildly modifies the structure to incorporate three gates, input gate, output gate and forget gate, to handle the vanishing gradient problem, where RNN begins to forget information far in the past. With these gates, LSTM is able to distinguish between important and less important information. The structure of LSTM can be shown in Figure 8.

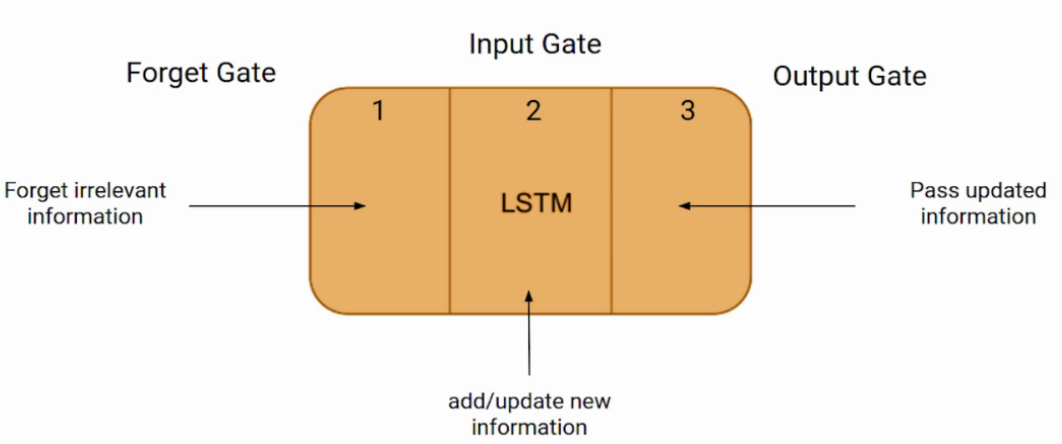

Fig 8. LSTM Architecture

4.2 PPO implementation to update policy based on old and new policies information

Apart from LSTM, we also think about incorporating Proximal Policy Optimization (PPO). PPO is a deep reinforcement learning algorithm commonly used to train agents. It is an on-policy algorithm, meaning that it updates policy based on the data collected in the interactive environment under the current policy. PPO works by optimizing a surrogate objective function that approximates the expected improvement in the policy. The surrogate objective function is designed to be easy to optimize while still being a good approximation of the true objective. The core idea of PPO is to limit the size of the policy updates at each iteration, which can make sure that the policy does not change too much and cause instability. This is done by a clipping mechanism to ensure that the policy update does not deviate too far from the previous policy. Besides, PPO also uses the observations to update from the old policy to the new policy. The architecture of PPO is shown in Figure 9.

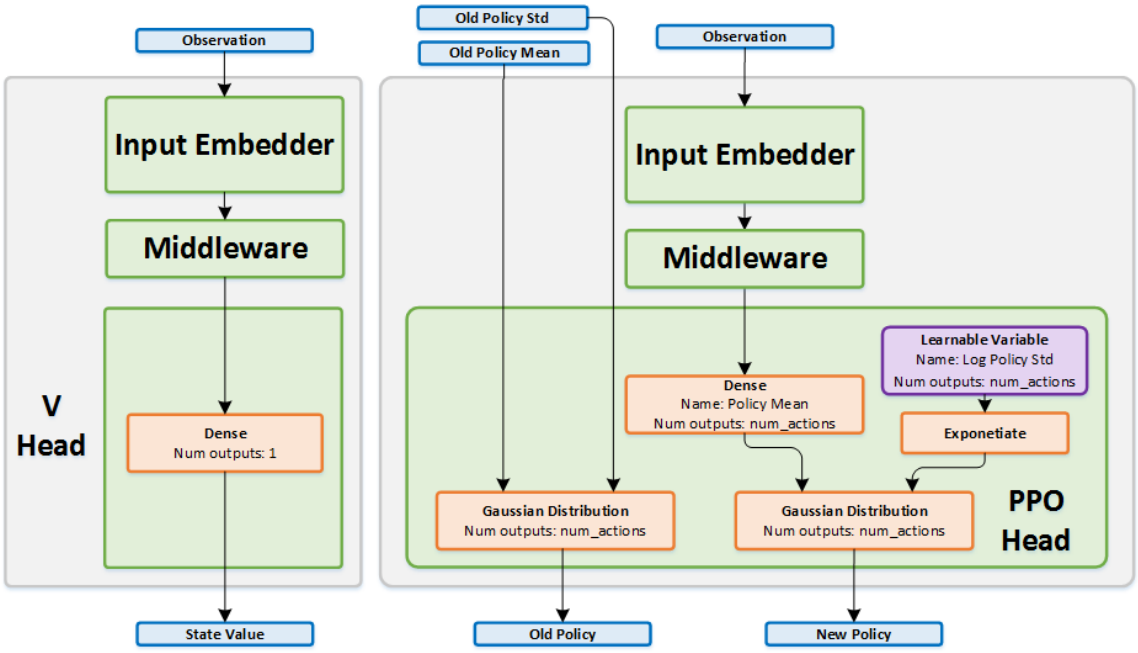

Figure 9. PPO Architecture

PPO has been shown to perform well on a wide range of tasks, including the daily investor problem. One of the advantages of PPO over DQN is that PPO is an on-policy algorithm while DQN is an off-policy algorithm, making PPO more sample-efficient and less prone to overfitting, especially in a complicated and noisy environment such as the financial market. Besides, PPO uses a surrogate objective function, making it more stable and robust to changes in the environment, while DQN uses the Bellman equation to update Q-values, which can lead to instability in the policy and divergence in some cases.

Experiment results have shown that PPO generally performs better than DQN and can achieve convergence in a relatively fewer number of steps. PPO is also more suitable to implement in the ABIDES-Market environment, which requires exploration and exploitation to fully capture the market complexity. On the other hand, DQN relies on the epsilon-greedy exploration strategy, which may not be effective in a complex and high-dimensional environment like the financial market. To better compare the performance of PPO with DQN, we use the same neural network architecture as DQN.

5. Simulated Stock Market Analysis

The simulated daily investor environment includes 1 exchange agent, 2 adaptive market maker agents, 1,000 noise agents, 102 value agents, and 12 momentum agents, where the exchange agent is responsible for the update of the limit order book at different timesteps, the adaptive market maker agent helps ensure the liquidity of the simulated market, the noise agent randomly buys or sells the stock shares, the value agent buys or sells the stock based on the most recent stock price, and the momentum agent makes trading decisions based on the stock price movement in a given period.

5.1 Stock Price and Order Volume: Close to Real-World Financial Market

In this section, we analyze the intraday stock price and order volume trends at different levels of the limit order book. The goal is to verify the stock price and order volume movements are close to the real-world financial market.

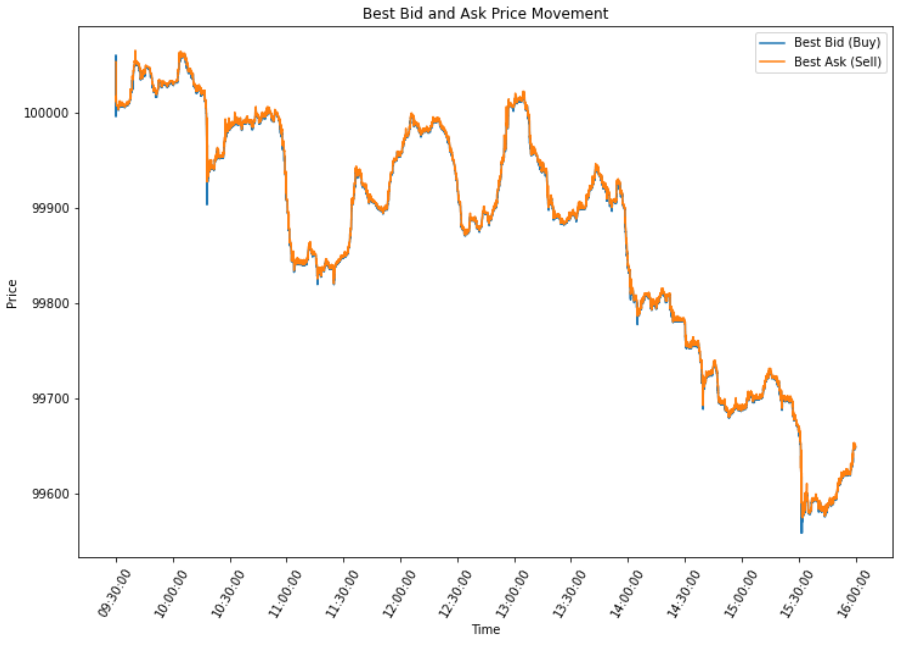

Fig 10. Best Bid Price and Best Ask Price Movement

Figure 10 shows the best bid and best ask price movements from 9:30 a.m. to 4:00 p.m. It shows that the price movement closely resembles the real-world stock price movement with several fluctuations and an

overall downward movement. The overall downward trend represents the one-day behavior of a stock while the sharp changes in the stock price represent macro shocks to this stock.

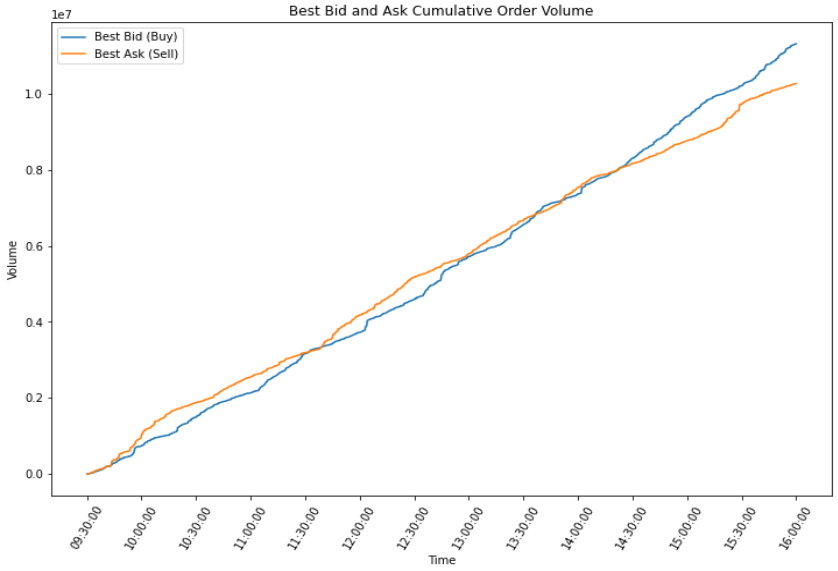

Figure 11. Cumulative Order Volume of Best Bid and Ask

Figure 11 presents the cumulative order volume or order quantity for the whole day. It shows that the order volume is approximately linearly increasing with respect to time and at the end of the day, it will reach about 12 million, indicating that there will be more than 10 million orders being executed in a single day for just one stock.

Figure 12 shows the bid and ask price movement from the 5th level to utilize more information on the stock market. Generally, the stock price movement is the same as the best bid and best ask, while the bid and ask price would have more small fluctuations along the day.

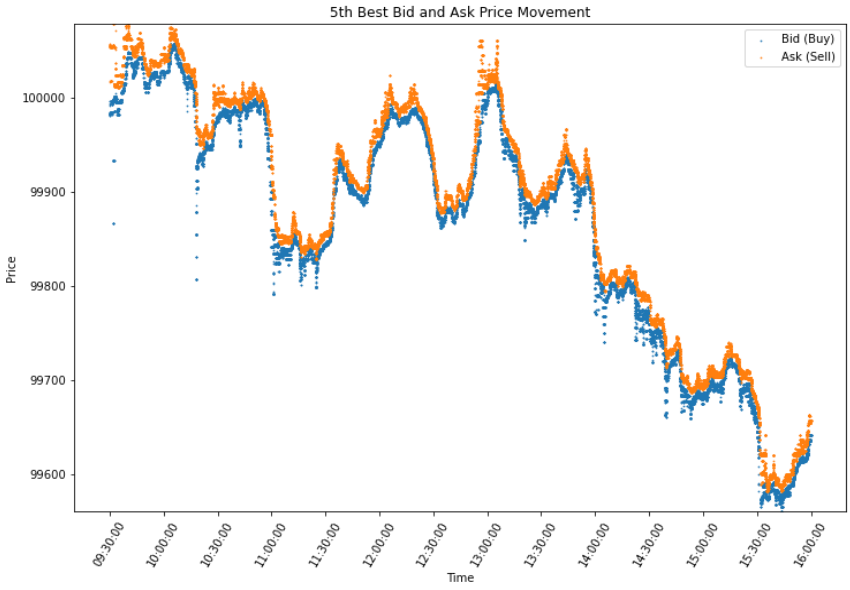

Fig 12. 5th Best Bid and Best Ask Price Movement

5.2 Trading Agent Behavior for different trading agents

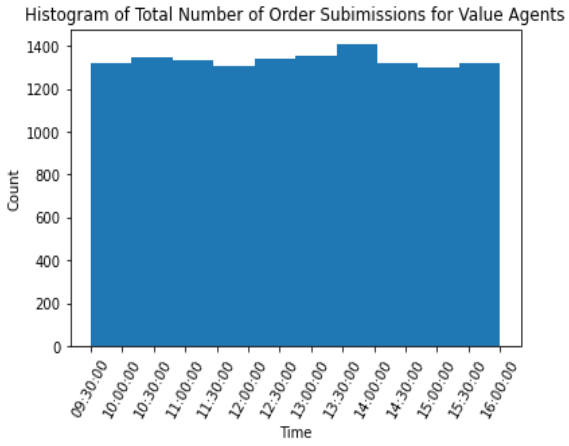

Fig 13. Histogram of Number of Order Submissions for Value Agents

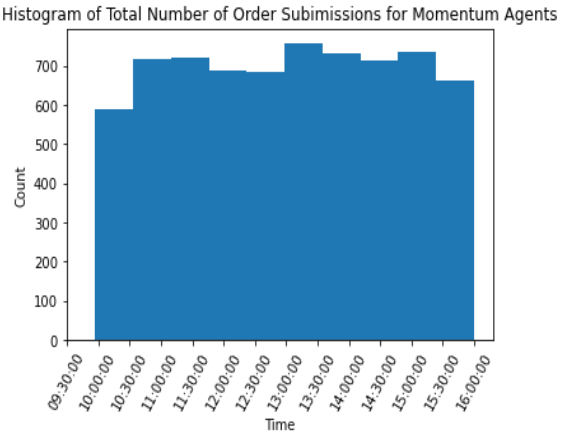

Fig 14. Histogram of Number of Order Submissions for Momentum Agents

Figure 13 shows the number of submitted orders across time for value agents, where the number of orders submitted is similar at different times of the day. Figure 14 shows the number of submitted orders across

time for momentum agents, where the number of orders submitted increases by 100 in the first hour and stays relatively stable throughout the rest of the day. There is also a slight decrease in the last 30 minutes. The behavior of value agents and momentum agents in general agree with the design of these two types of agents, where the value agent makes trading decisions based only on the most recent stock price, which does not change much throughout the day, while the momentum agent makes trading decisions based on a series of recent stock prices. The behavior of the momentum agent also represents a smart trading strategy, i.e., placing fewer orders when the market is unstable.

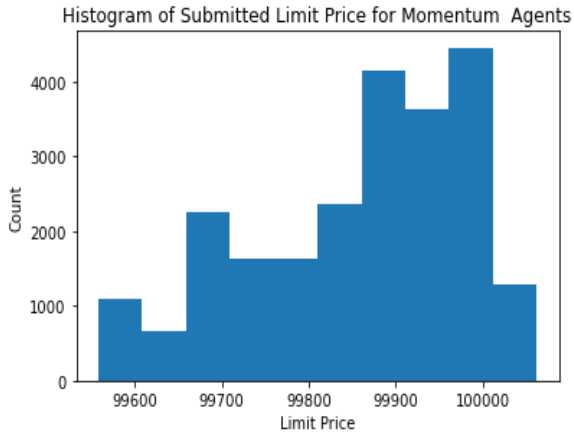

Fig 15. Histogram of Submitted Price for Value Agents

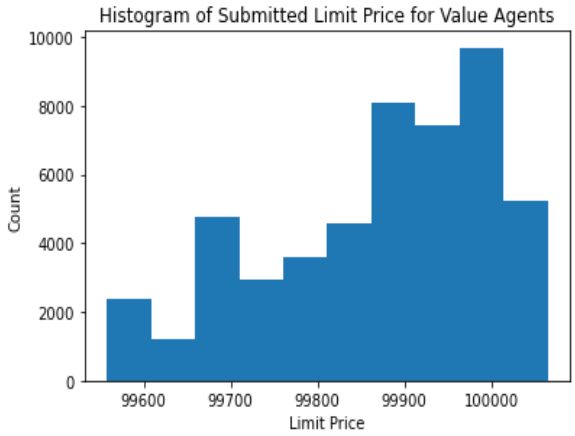

Fig 16. Histogram of Submitted Price for Momentum Agents

Figure 15 and Figure 16 show the histograms of the trading price for value agents and momentum agents. Generally, the two types of agents trade at similar prices, except that the value agents tend to place more orders at the highest price.

6. Trading Policy Results

6.1 Convergence Analysis of the Proposed Models

In this section, we compare the convergence performance of the PPO model and the PPO model with LSTM layers by their maximum, minimum, and average episode rewards under different hyperparameter combinations.

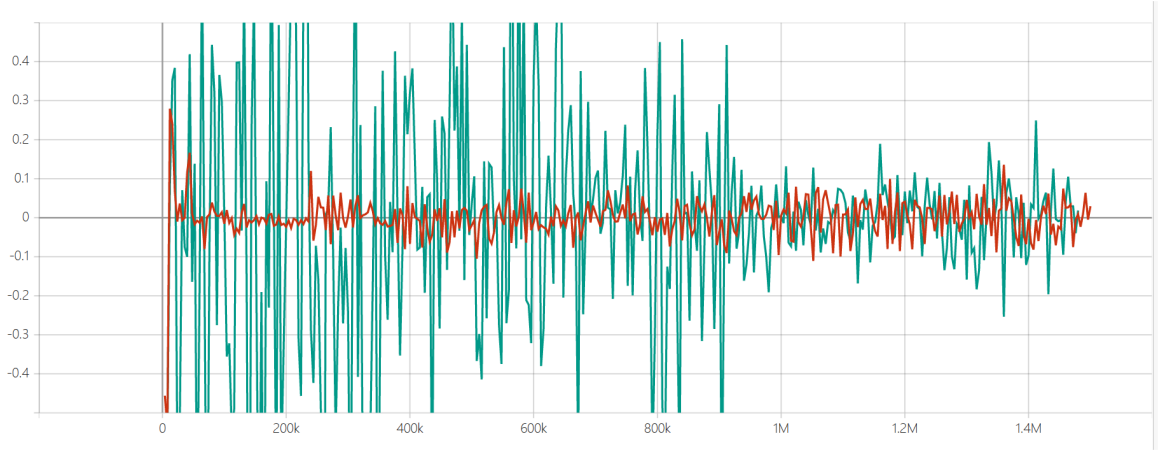

Fig 17. Mean Episode Reward of the PPO Model

Figure 17 shows the mean episode reward of the PPO model per training episode with respect to steps, where the green line and the red line present the PPO model performance under different learning rates and random seeds. We can see that after 1.4 million steps of training, the trading agents are able to reduce variance and reach convergence towards small positive average reward values, thereby achieving more stable and robust trading policies.

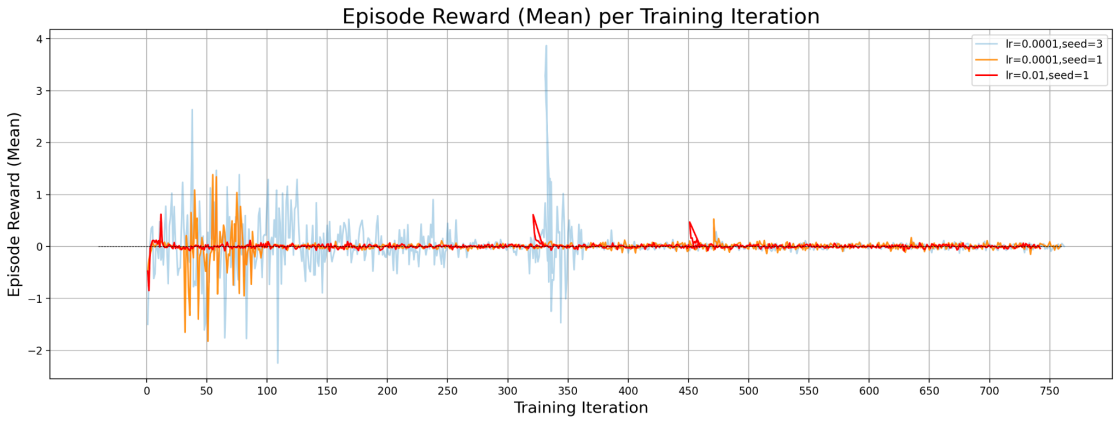

Fig 18. Mean Episode Reward of the PPO Model with Different Hyperparameters

Figure 18 compares the PPO model performance under different hyperparameter combinations. We can see that the model with a learning rate of 0.01 and a fixed random seed 1 quickly achieves convergence in 50 training iterations, while the model with a learning rate of 0.0001 and the same random seed achieves convergence in 100 iterations with a slower speed. One possible explanation for the inferior performance of the model with a smaller learning rate is that each time the model only updates the parameters slightly, thereby taking a much longer time to reach convergence.

We also compare the PPO model performance under different random seeds. We can see that the model performance varies greatly between different random seeds, where the model with the random seed 3 achieves steady performance after 450 iterations.

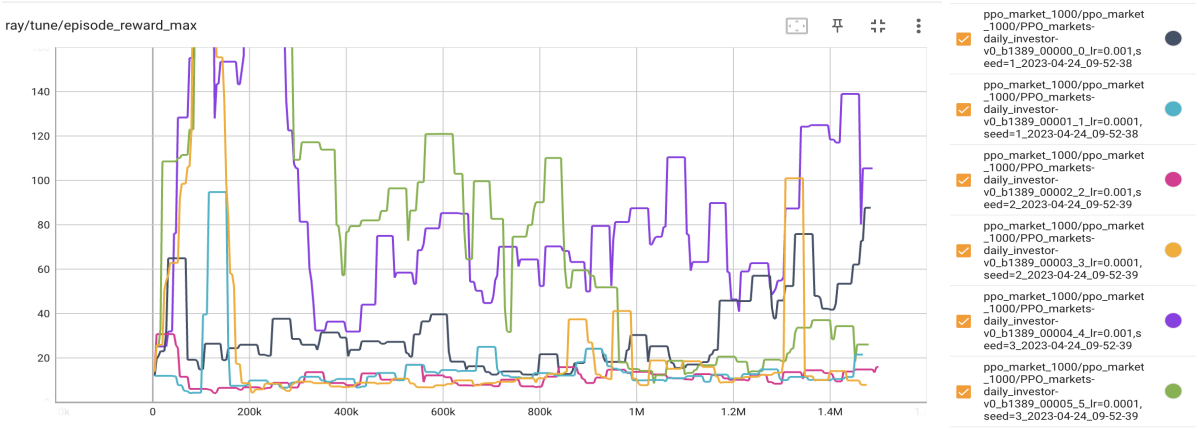

Fig 19. Maximum Episode Reward of the PPO Model with Different Hyperparameters

Figure 19 and Figure 20 compare the maximum episode reward of the PPO model under different hyperparameters. Overall, all models have a positive maximum episode reward, suggesting that all models are able to achieve at least small net profits after training with a positive probability. The performance of the model of the maximum value under different hyperparameters is similar to the mean reward value performance presented in Figure 18.

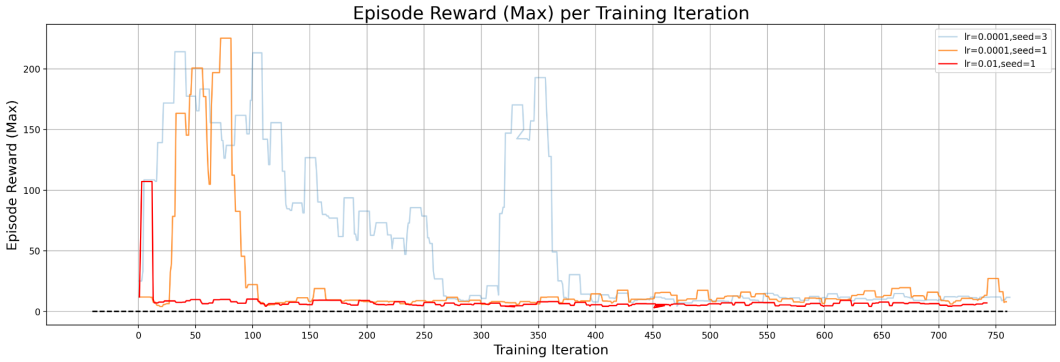

Fig 20. Maximum Episode Reward of the PPO Model with the Best Performance

Similarly, Figure 21 and Figure 22 compare the minimum episode reward of the PPO model under different hyperparameters. Similar to the analysis above, the model with a learning rate of 0.01 and a random seed of 1 is able to achieve a steady performance with a minimum net loss.

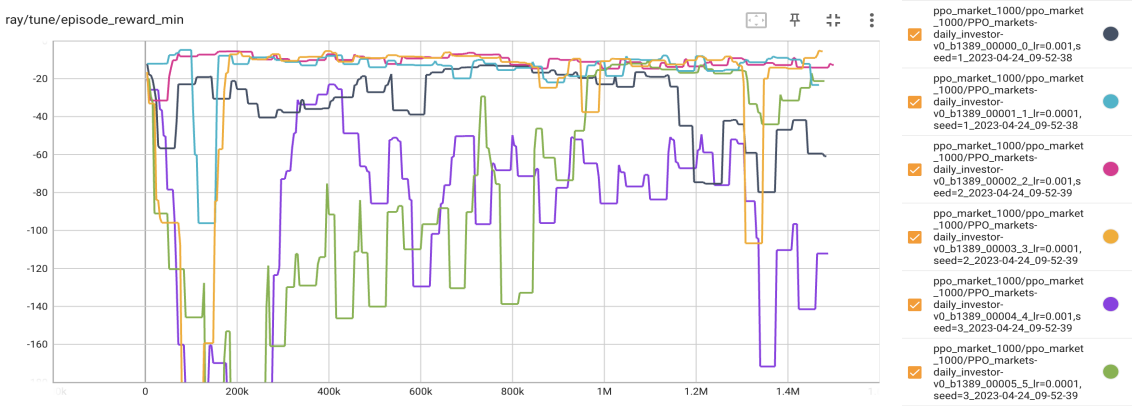

Fig 21. Minimum Episode Reward of the PPO Model with Different Hyperparameters

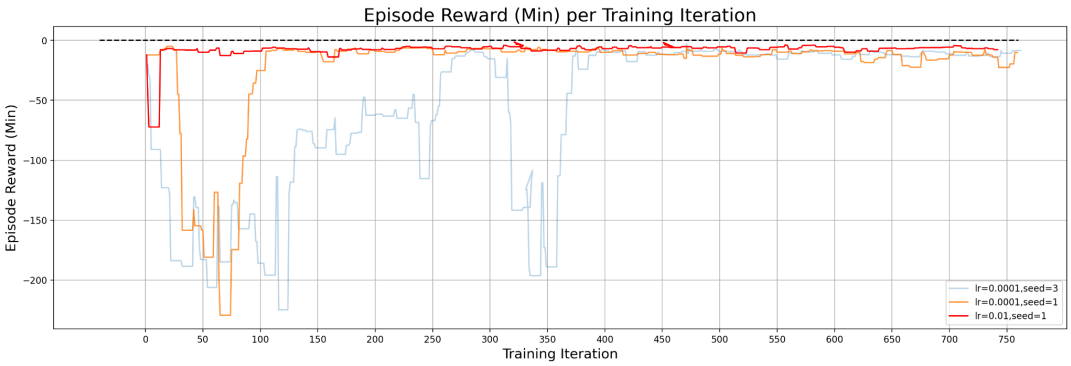

Fig 22. Minimum Episode Reward of the PPO Model with the Best Performance

Figure 23 presents the mean episode reward of the PPO model with LSTM layers with respect to the number of training iterations. The hyperparameters are set to a learning rate of 0.001 and a random seed of 1. We can see that the model is able to achieve stable performance in less than 50 iterations and reach convergence in 300 iterations towards an average value slightly above zero.

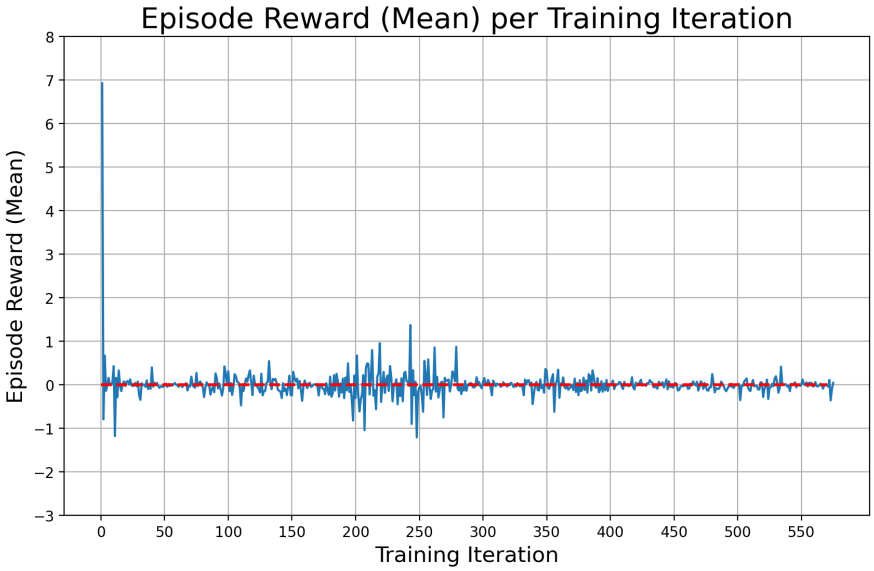

Fig 23. Mean Episode Reward of the PPO Model with LSTM Layers

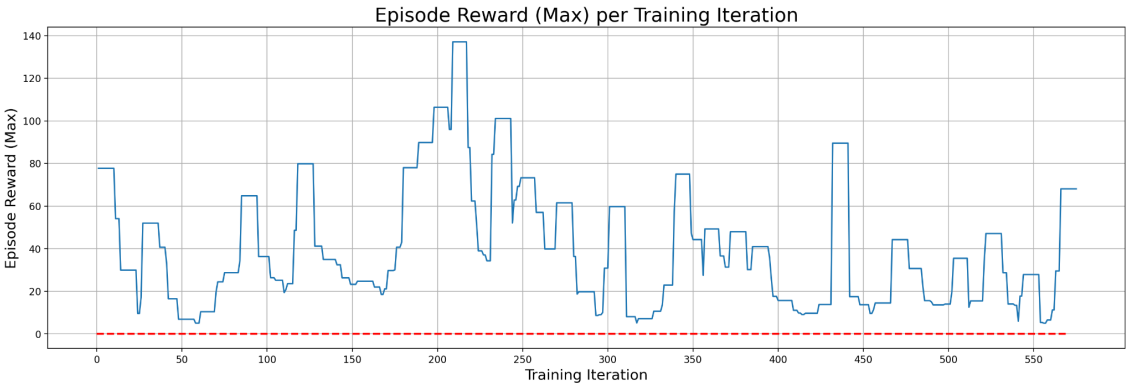

Fig 24. Maximum Episode Reward of the PPO Model with LSTM Layers

Figure 24 and Figure 25 present the maximum and minimum episode reward of the PPO model with LSTM layers. In 550 training iterations, the model always has a positive maximum reward.

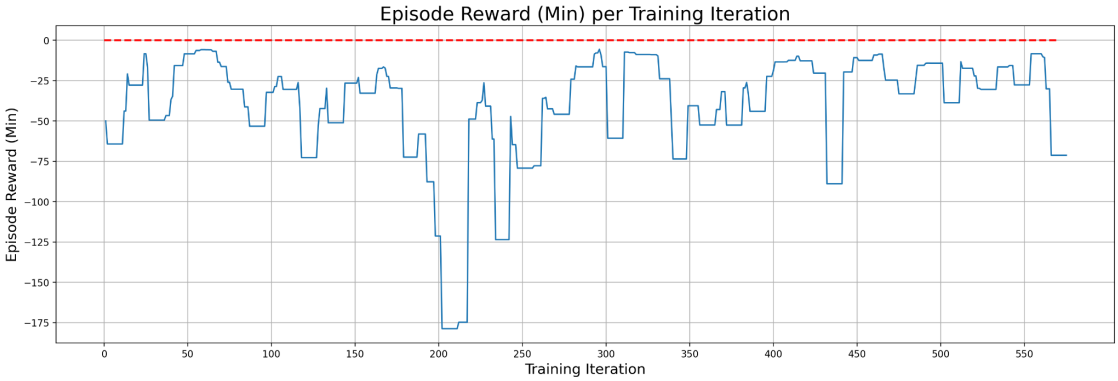

Fig 25. Maximum Episode Reward of the PPO Model with LSTM Layers

Finally we compare the mean episode reward of the PPO model and the PPO model with LSTM layers in Figure 26. By adding LSTM layers, the PPO model is able to reach faster convergence and a smaller variance and fluctuation in 500 iterations. Both models are able to achieve a steady and robust performance with the mean episode reward value fluctuating between -1 and 1.

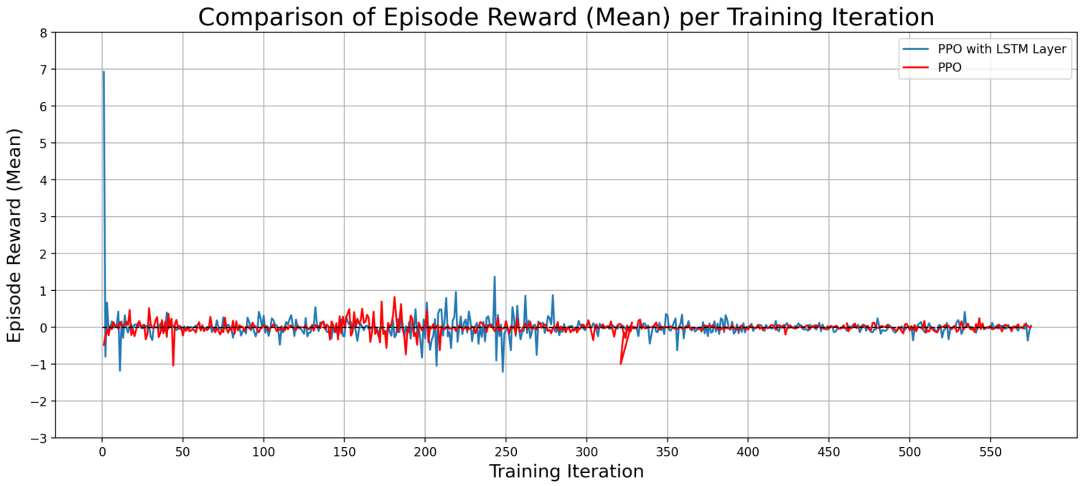

Fig 26. Comparison of the Mean Episode Reward of the Proposed Models

6.2 Profitability of Different Trading Policies

In this section, we compare the performance of different trading policies, including our proposed models, the baseline models, and some other traditional trading policies. Specifically, we investigate the daily investment setting where the trading agent is given 1 million dollars in cash at the start of the day and is allowed to buy or sell one stock. To evaluate model performance, we simulate the intraday trading with 50 random seeds and calculate the average values for the evaluation metrics. Specifically, we compare the cash, the number of shares held, the profit, and the profit margin at the end of the day for different trading policies. We investigate the following trading policies:

- PPO: the proposed PPO model with a learning rate of 0.01 and a random seed of 1;
- DQN: the baseline DQN model;
- Passive: the agent holds on to the cash and does not make any transaction;
- Aggressive: the agent always buys the stock and does not sell them;
- Random: the agent randomly buys or sells a certain amount of stock shares;
- Smart Random: besides buying and selling stocks, the agent is allowed to make no transaction;

Table 1. Comparison of Rewards and Profits for Different Trading Policies

name	episode_reward	Profit	Profit_Margin	cash	holdings	spread	marked_to_market
ppo: lr=0.01,seed=1	6.114810	23847.76	2.3848%	9.830686e+05	0.4	1.58	1023847.76
passive	0.000000	0.00	0.0000%	1.000000e+06	0.0	2.04	1000000.00
dqn	-0.051795	-202.00	-0.0202%	4.204756e+05	5.8	1.70	999798.00
random_no_action	-2.100179	-8190.70	-0.8191%	1.601430e+06	-6.0	1.40	991809.30
random	-6.835390	-26658.02	-2.6658%	-1.894300e+08	1904.2	1.72	973341.98
aggressive	-16.317056	-63636.52	-6.3637%	-3.729324e+08	3738.8	2.18	936363.48

Table 1 presents the comparison of rewards and profits for different trading policies, with our proposed PPO model outperforming all other policies. Specifically, the PPO trading policy is able to achieve a 2.38% profit margin and a positive average episode reward while all other policies experience a net loss at the end of the day. The baseline DQN model is able to achieve the third-best performance with a 0.02% net loss. We also notice that the aggressive policy results in the worst performance, suggesting that simply investing in the stock would not lead to a profitable strategy.

Table 2. Comparison of Rewards and Profits of the PPO Model with Different Hyperparameters

name	episode_reward	Profit	Profit_Margin	cash	holdings	spread	marked_to_market
ppo: lr=0.01,seed=1	6.114810	23847.76	2.3848%	9.830686e+05	0.40	1.58	1023847.76
ppo: lr=0.0001,seed=3	3.504077	13665.90	1.3666%	-3.445695e+06	44.60	1.40	1013665.90
ppo: lr=0.0001,seed=1	3.361390	13109.42	1.3109%	-1.686267e+06	27.00	1.54	1013109.42
ppo: lr=0.001,seed=1	2.035990	7940.36	0.7940%	-3.195727e+06	42.00	1.82	1007940.36
ppo: lr=0.01,seed=3	2.002508	7809.78	0.7810%	-1.900177e+08	1910.00	1.56	1007809.78
ppo: lr=0.0001,seed=2	1.263800	4928.82	0.4929%	-2.273719e+07	237.50	1.66	1004928.82
ppo: lr=0.001,seed=3	0.367159	1431.92	0.1432%	6.513820e+06	-55.20	1.50	1001431.92
ppo: lr=0.001,seed=2	-5.515923	-21512.10	-2.1512%	1.074026e+08	-1063.80	1.82	978487.90
ppo: lr=0.01,seed=2	-7.896462	-30796.20	-3.0796%	-3.577650e+08	3587.22	1.40	969203.80

Table 2 compares the rewards and profits of the PPO model under different hyperparameter combinations. In analogous to the results presented in Figure 18, the model with a learning rate of 0.01 and a random

seed of 1 is able to achieve the highest profits and rewards. Overall the majority of the PPO models are able to achieve positive profits, with only 2 models resulting in a net loss at the end of the day.

We also notice an interesting phenomenon that a better performance is achieved with a higher learning rate, which does not agree with the conclusion for other machine learning methods. A lower learning rate normally means that the model would update the parameters slower, while a higher learning rate would result in a faster convergence model and may fluctuate around the optimal point. One possible explanation is that PPO is an on-policy algorithm where the trading agent uses information from the most up-to-date policy and makes trading decisions based on the most recent policy. Other algorithms including the baseline DQN model are off-policy algorithms, which use information collected at any point. Therefore, the PPO model may require a higher learning rate to reach a good performance, while a lower learning rate may lead to possible overfitting.

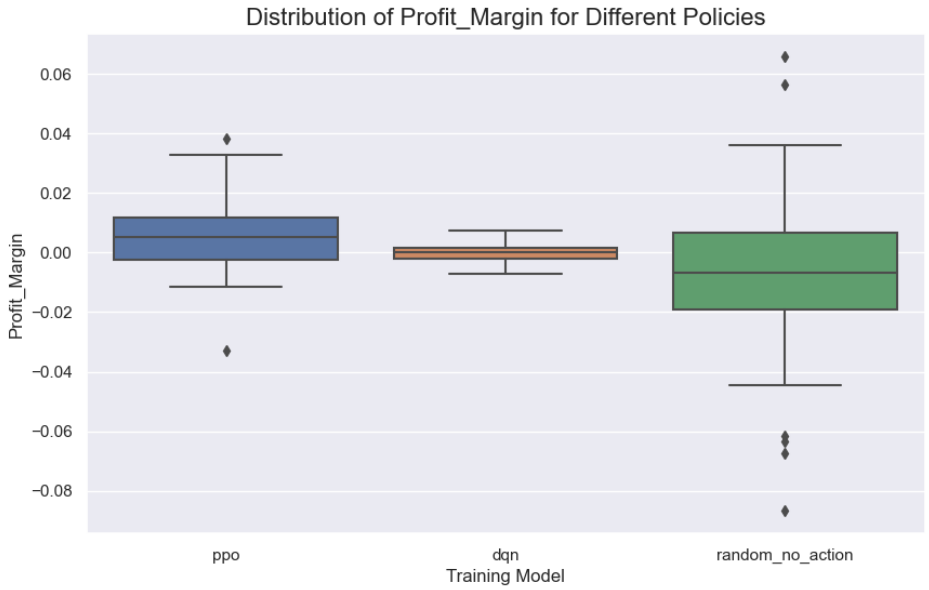

Fig 27. Comparison of Profit Margins for Different Trading Policies

Figure 27 shows the detailed comparison of profit margins for three trading policies, the PPO model, the DQN model, and the SMART RANDOM model. We can see that the PPO model has a profit margin distributed mostly above 0 with a median profit margin of approximately 1%, while the DQN model achieves the second-best performance with a more concentrated distribution of around 0.

7. Conclusion and Future Work

Our project utilizes state-of-the-art reinforcement learning models and a multi-agent simulator named ABIDES in the OpenAI Gym environment to investigate optimal trading strategies with high profitability in a simulated financial stock market. To train trading agents in the ABIDES environment, we employ the advanced PPO model and LSTM layers, allowing the agents to make use of previous market information and update their knowledge throughout each training episode.

We compare the performance of the proposed models with the baseline model and some other traditional trading policies. Our major finding is that by using reinforcement learning models, we are able to develop smarter trading policies, outperforming all other traditional trading strategies, such as the aggressive trading policy. More specifically, our proposed PPO model is able to achieve a robust and steady trading policy with an average of 2% daily profit margins. The implementation of the LSTM layers allows the utilization of previous market information, thereby helping the agents make close-to-real-world trading decisions and achieving faster convergence.

There are multiple areas we can continue to explore in the future. First, we can investigate the model performance under more hyperparameter combinations of different learning rates and random seeds for our proposed models. In addition, we can also explore including more market state variables into the state space, allowing the agents to make trading decisions more thoroughly. We plan to enlarge the action space as well, allowing the agents to buy or sell different amounts of stock shares.

References:

[1] Selim Amrouni, Aymeric Moulin, Jared Vann, Svitlana Vyetrenko, Tucker Balch, Manuela Veloso, ABIDES-Gym: Gym Environments for Multi-Agent Discrete Event Simulation and Application to Financial Markets, https://arxiv.org/abs/2110.14771

[2] Karima Amoura, Patrice Wira, Said Djennoune, A State-space Neural Network for Modeling Dynamical Nonlinear Systems, https://www.scitepress.org/papers/2011/36805/36805.pdf

[3] David Byrd, Maria Hybinette, Tucker Hybinette Balch, ABIDES: Towards High-Fidelity Market Simulation for AI Research, https://arxiv.org/abs/1904.12066

[4] JózefKorbicz, MarcinMrugalski, ThomasParisini, Designing State-space Models With Neural Networks, https://doi.org/10.3182/20020721-6-ES-1901.01630

[5] Trading Volume from 1992 to 2021, According to NYSE, https://www.nyse.com/trading-data

[6] High Frequency Trading Market Report, https://dataintelo.com/report/high-frequency-trading-market/

[7]Olivier Buffet, Olivier Pietquin, Paul Weng, Reinforcement Learning. https://arxiv.org/abs/2005.14419

[8] Svitlana Vyetrenko, David Byrd, Nick Petosa, Mahmoud Mahfouz, Danial Dervovic, Manuela Veloso, Tucker Hybinette Balch, Get Real: Realism Metrics for Robust Limit Order Book Market Simulations, https://arxiv.org/abs/1912.04941

[9] Zihao Zhang, Stefan Zohren, Stephen Roberts, Deep Reinforcement Learning for Trading, https://arxiv.org/abs/1911.10107

[10] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, Martin Riedmiller, Playing Atari with Deep Reinforcement Learning, https://arxiv.org/abs/1312.5602

[11] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, Wojciech Zaremba, OpenAI Gym, https://arxiv.org/abs/1606.01540

[12] Michaël Karpe, Jin Fang, Zhongyao Ma, Chen Wang, Multi-Agent Reinforcement Learning in a Realistic Limit Order Book Market Simulation, https://arxiv.org/abs/2006.05574

[13] Hui Niu, Siyuan Li, Jian Li, MetaTrader: An Reinforcement Learning Approach Integrating Diverse Policies for Portfolio Optimization, https://arxiv.org/abs/2210.01774

[14] Tohid Atashbar, and Rui (Aruhan) Shi, Deep Reinforcement Learning: Emerging Trends in Macroeconomics and Future Prospects, https://www.elibrary.imf.org/downloadpdf/journals/001/2022/

259/001.2022.issue-259-en.pdf