Capstone Project Report

Group D2—-Using Reinforcement Learning and Multi-Agent Simulator to Improve the Profitability and Efficiency of High-Frequency Trading

Team Member: Charlotte Jin, Ethan Choukroun,

Jingqi Ma, Haoyang Wang

Berkeley
UNIVERSITY OF CALIFORNIA

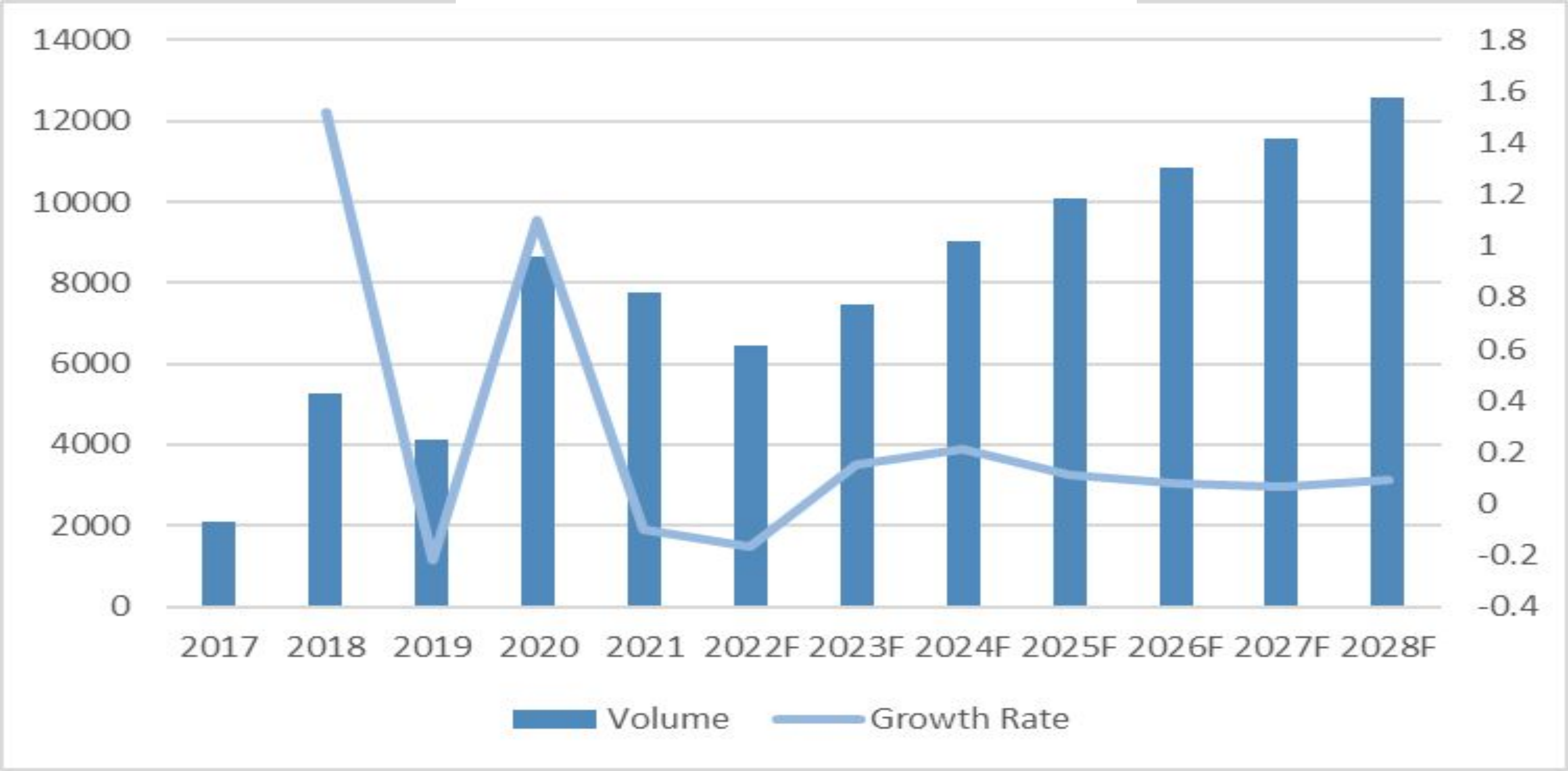
1. Introduction

- The total trading volume in the stock market has been increasing dramatically since 2012

1. Introduction

- The volume of High Frequency Trading(HFT) has been increasing during the past few years

Growth of HFT volume in the stock market



1.Introduction

- Aim:

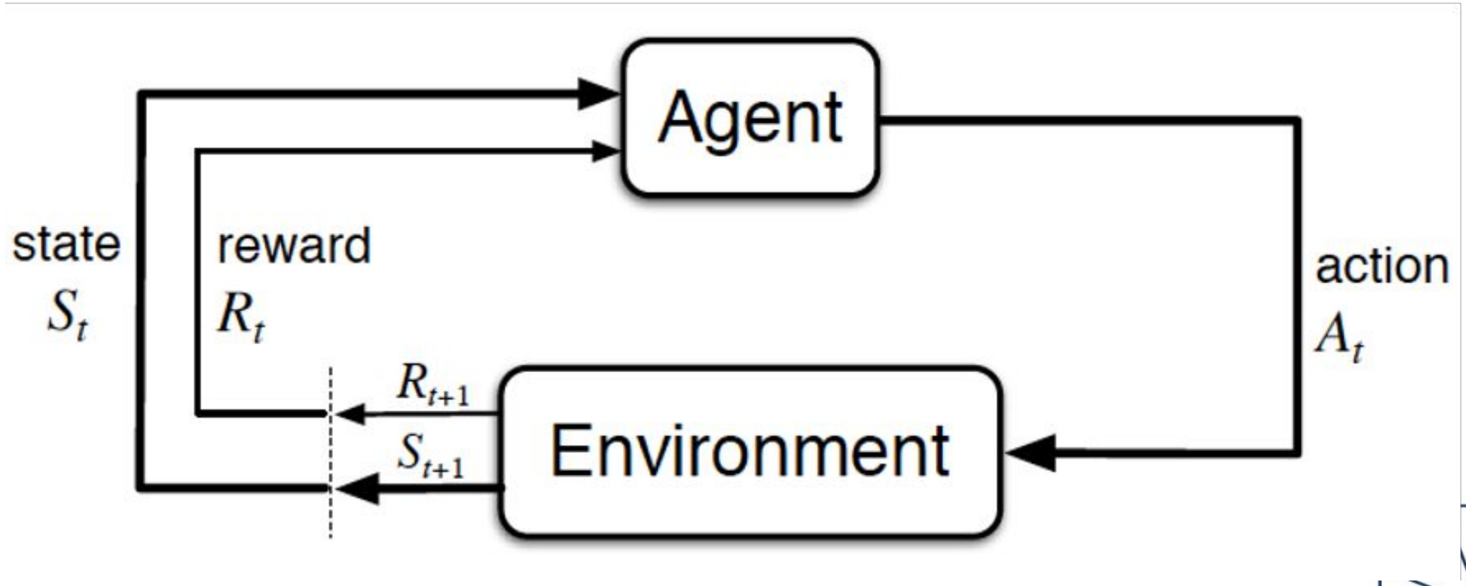
  In order to meet the high frequency trading demand, we plan to using the state-of-the-art machine learning method, **Reinforcement Learning**.

  By coming up with **optimal high frequency trading strategy**, we can help investors to improve the profitability.

2. Related Work

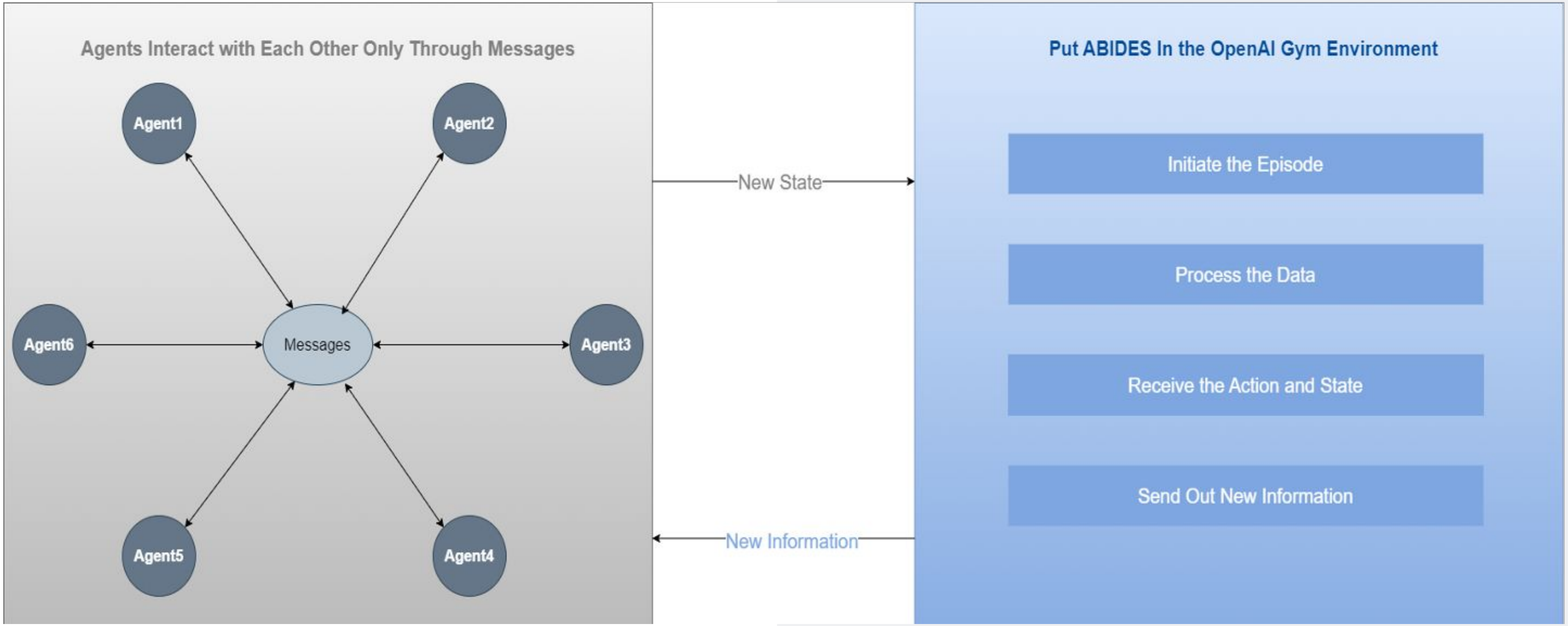
- Reinforcement Learning

In Reinforcement learning (RL), agents are trained on a reward and punishment mechanism. The agent is rewarded for correct moves and punish for the wrong ones. It can train agents to choose the optimal trading strategy.

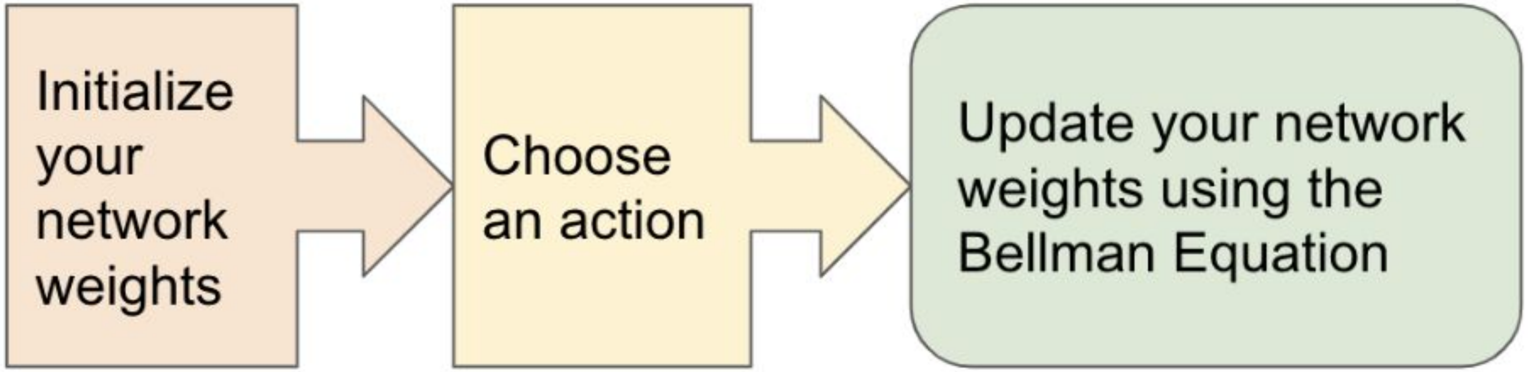


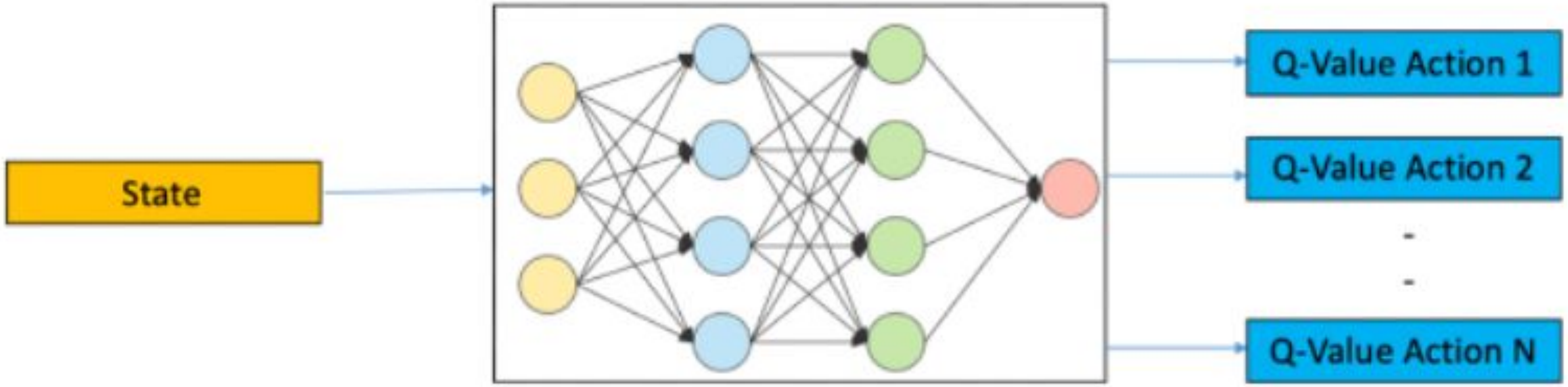
2. Related Work

- ABIDES–Market Simulator
- OpenAI Gym–Training and Testing Env



3.Baseline Model





Deep Q Learning

3.Baseline Model

```
tune.run("DQN",
         name=name_xp,
         resume=False,
         stop={'training_iteration':1000},
         checkpoint_at_end=True,
         checkpoint_freq=10,
         config={'env':"markets-daily_investor-v0",
                 'env_config':{'background_config':'rmsc04',
                               'timestep_duration':"10S",
                               'mkt_close':"16:00:00",
                               'timestep_duration':"60s",
                               'starting_cash': 1000000, #original 1_000_000
                               'order_fixed_size': 10,
                               'state_history_length':4,
                               'market_data_buffer_length': 5,
                               'first_interval': "00:05:00",
                               'reward_mode': "dense",
                               'done_ratio': 0.3,
                               'debug_mode': True,
                               #'execution_window':'04:00:00',
                               #.... Here we just use the default values
                               },
                 'seed':tune.grid_search([1,2,3]),
         #'seed':tune.grid_search([1,2,3]),
                 'num_gpus':0,
                 'num_workers':0,
                 'hiddens':[50,20],
                 'gamma':1,
                 'lr':tune.grid_search([0.001,0.0001,0.01]),
         #'lr':tune.grid_search([0.001,0.0001,0.01]),
                 'framework':'torch',
                 'observation_filter':'MeanStdFilter',
         },
         )
```

MDP Formulation

- **State space:**

- ***holdings(t)***: number of shares of the stock held by the experiment agent at time step t
- ***imbalance(t)*** = bids volume / (bids volume + asks volume)
- ***spread(t)*** = best Ask(t) - best Bid(t)
- ***directionfeature(t)*** = midPrice(t) - lasTransactionPrice(t)
- ***R^k(t)*** = (r(t),...,r(t-k+1))
  where r(t-i) = mid(t-i) - mid(t-i-1)

- **Actions Space:**

- “Buy”, “Hold”, “Sell”

4.Proposed Techniques

Proximal Policy Optimization (PPO)

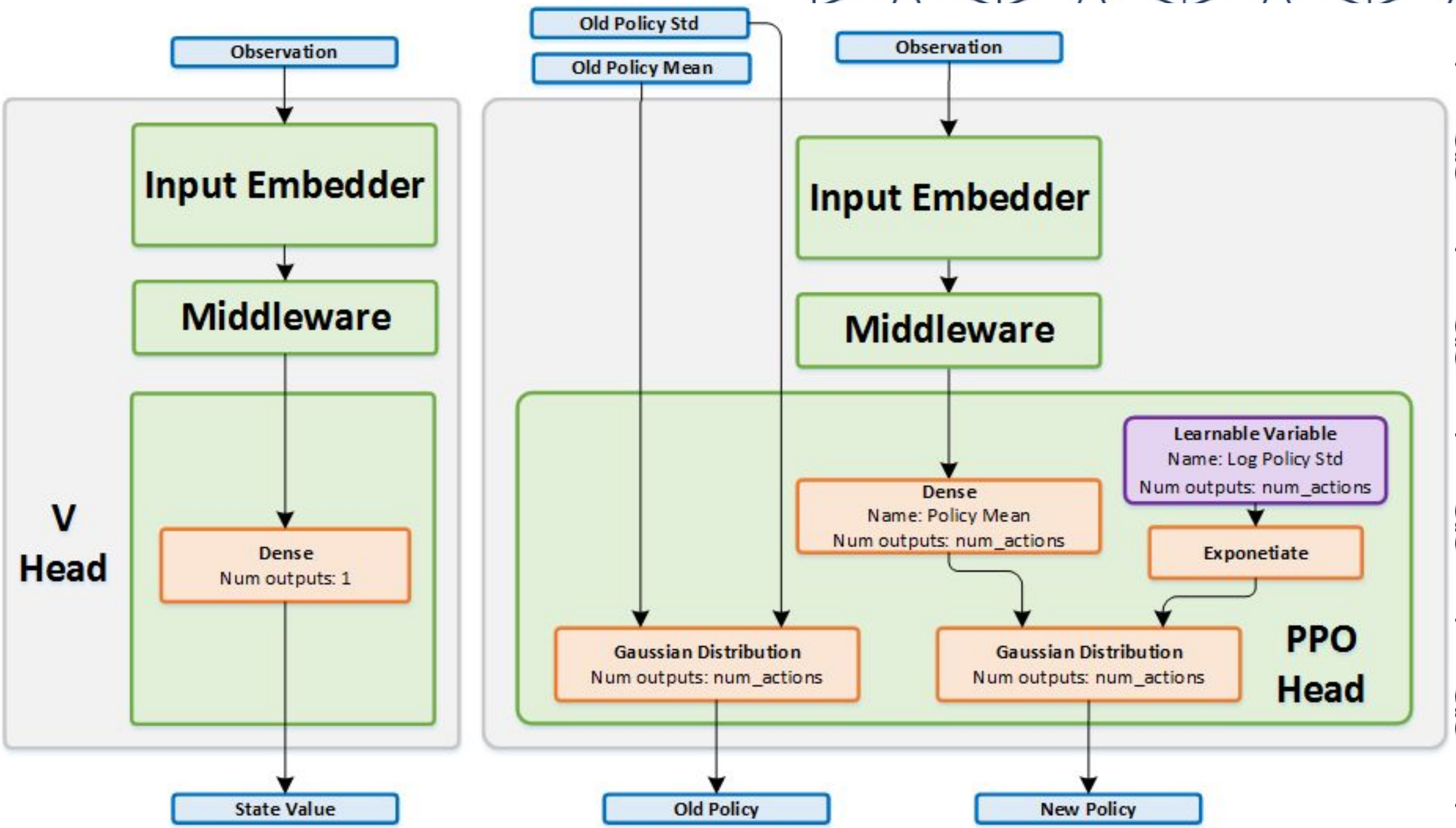
Advantage of PPO over DQN

- On-policy algorithm

  More sample-efficient and less prone to overfitting

- Use surrogate objective function (Not Bellman Equation)

  More stable and robust to changes in the environment



4.Proposed Techniques

Add LSTM layers in PPO

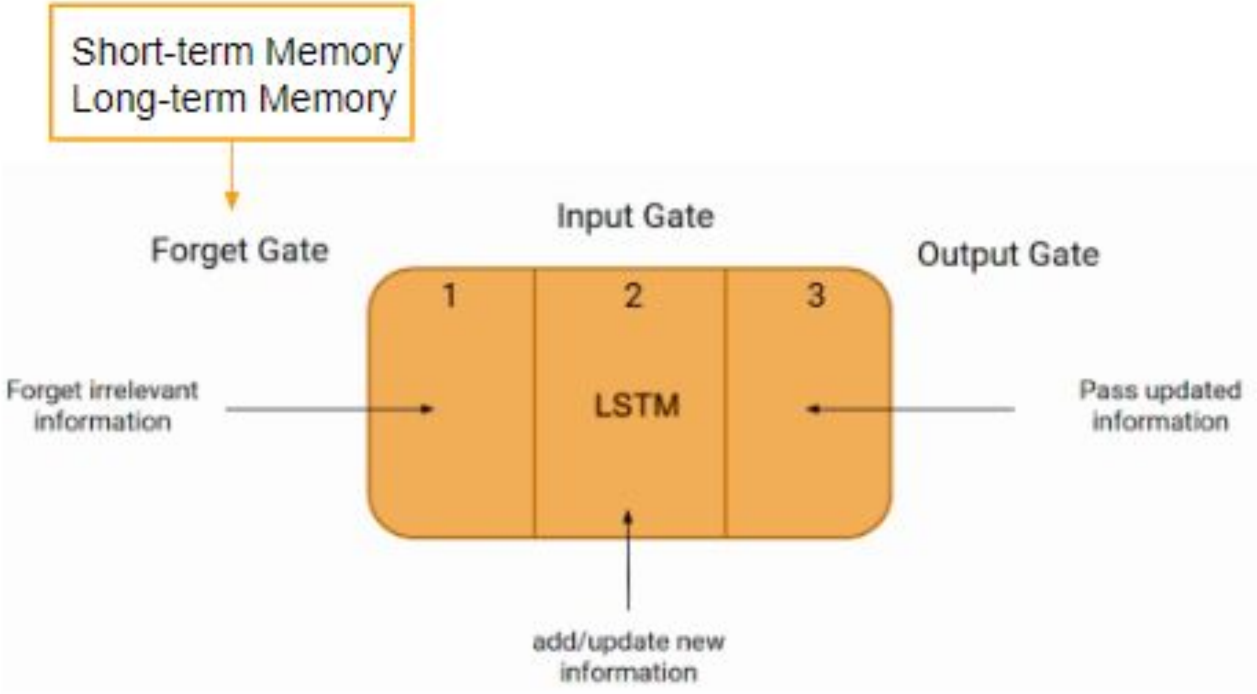
The future stock prices are dependent on both current and recent market states in short term.

LSTM provides a natural representation to utilize short-term market state information.

4.Proposed Techniques

Add LSTM layers in PPO

- Well-Suited for Processing Time Series Data;
- Take Advantage of Agents' Past Experience and Historic Data;
- Can Memorize Previous Information for A Long Time;
- Can Automatically Store Only the Important Historic Information;



4.Proposed Techniques

```
tune.run("PPO",
        name=name_xp,
        resume=True,
        stop={'training_iteration':1000},
        checkpoint_at_end=True,
        checkpoint_freq=10,
        config={'env':"markets-daily_investor-v0",
                'env_config':{'background_config':'rmsc04',
                              'timestep_duration':"10S",
                              'mkt_close':"16:00:00",
                              'timestep_duration':"60s",
                              'starting_cash': 1_000_000,
                              'order_fixed_size': 10,
                              'state_history_length':4,
                              'market_data_buffer_length': 5,
                              'first_interval': "00:05:00",
                              'reward_mode': "dense",
                              'done_ratio': 0.3,
                              'debug_mode': True,
                              #'execution_window':'04:00:00',
                              #.... Here we just use the default values
                              },
                'seed':tune.grid_search([1,2,3]),
#'seed':tune.grid_search([1,2,3]),
                #'num_gpus':0,
                #'num_workers':0,
                #'hiddens':[50,20],
                'gamma':1,
                'lr':tune.grid_search([0.001,0.0001,0.01]),
#'lr':tune.grid_search([0.001,0.0001,0.01]),
                'framework':'torch',
                #'observation_filter':'MeanStdFilter',
},
)
```

PPO
Model

4.Proposed Techniques

```python
from ray.rllib.models.torch.torch_modelv2 import TorchModelV2
```

```python
# The custom model that will be wrapped by an LSTM.
class MyCustomModel(TorchModelV2):
    def __init__(self, obs_space, action_space, num_outputs, model_config, name):
        super().__init__(obs_space, action_space, num_outputs, model_config, name)
        self.num_outputs = int(np.product(self.obs_space.shape))
        self._last_batch_size = None

    # Implement your own forward logic, whose output will then be sent
    # through an LSTM.
    def forward(self, input_dict, state, seq_lens):
        obs = input_dict["obs_flat"]
        # Store last batch size for value_function output.
        self._last_batch_size = obs.shape[0]
        # Return 2x the obs (and empty states).
        # This will further be sent through an automatically provided
        # LSTM head (b/c we are setting use_lstm=True below).
        return obs * 2.0, []

    def value_function(self):
        return torch.from_numpy(np.zeros(shape=(self._last_batch_size,)))
```

LSTM
Layer

```python
tune.run("PPO",#DQN
         name=name_xp,
         local_dir="/global/scratch/users/irisma/ppo_lstm_market_run1000_1",
         resume=False,
         stop={'training_iteration':1000},
         checkpoint_at_end=True,
         checkpoint_freq=10,
```

```python
config={'env':"markets-daily_investor-v0",
        'env_config':{'background_config':'rmsc04',
                      'timestep_duration':"10S",
                      'mkt_close':"16:00:00",
                      'timestep_duration':"60s",
                      'starting_cash': 1_000_000,
                      'order_fixed_size': 10,
                      'state_history_length':4,
                      'market_data_buffer_length': 5,
                      'first_interval': "00:05:00",
                      'reward_mode': "dense",
                      'done_ratio': 0.3,
                      'debug_mode': True,
                      #'execution_window':'04:00:00',
                      #.... Here we just use the default values
                      },
        'model':{'use_lstm':True,
                 'lstm_cell_size':64,
                 "custom_model": "my_torch_model",
                 "custom_model_config": {}
                 },
        'seed':tune.grid_search([1]),
#'seed':tune.grid_search([1,2,3]),
        #'num_gpus':0,
        #'num_workers':0,
        #'hiddens':[50,20],
        'gamma':1,
        'lr':tune.grid_search([0.001]),
#'lr':tune.grid_search([0.001,0.0001,0.01]),
        'framework':'torch',
        #'observation_filter':'MeanStdFilter',
},
)
```

5.Simulated Stock Market Analysis

We use the daily investor environment, with 1 Exchange Agent, 2 Adaptive Market Maker Agents, 1000 Noise Agents, 102 Value Agents, and 12 Momentum Agents.

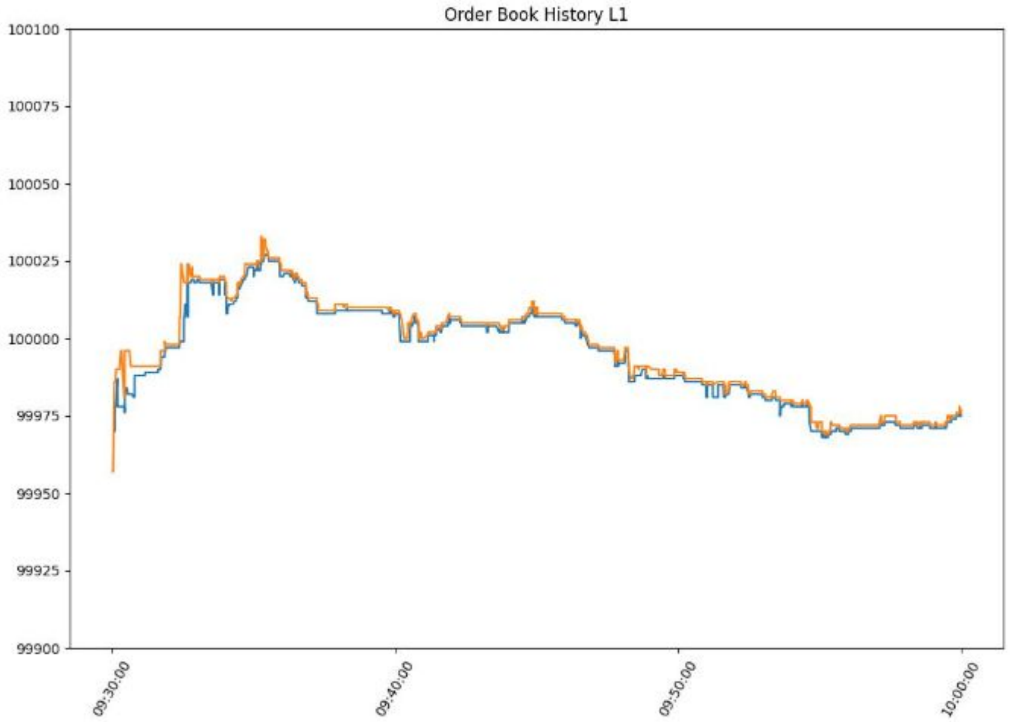


Fig8. Simulated Best Bid Price and Best Ask Price Movements

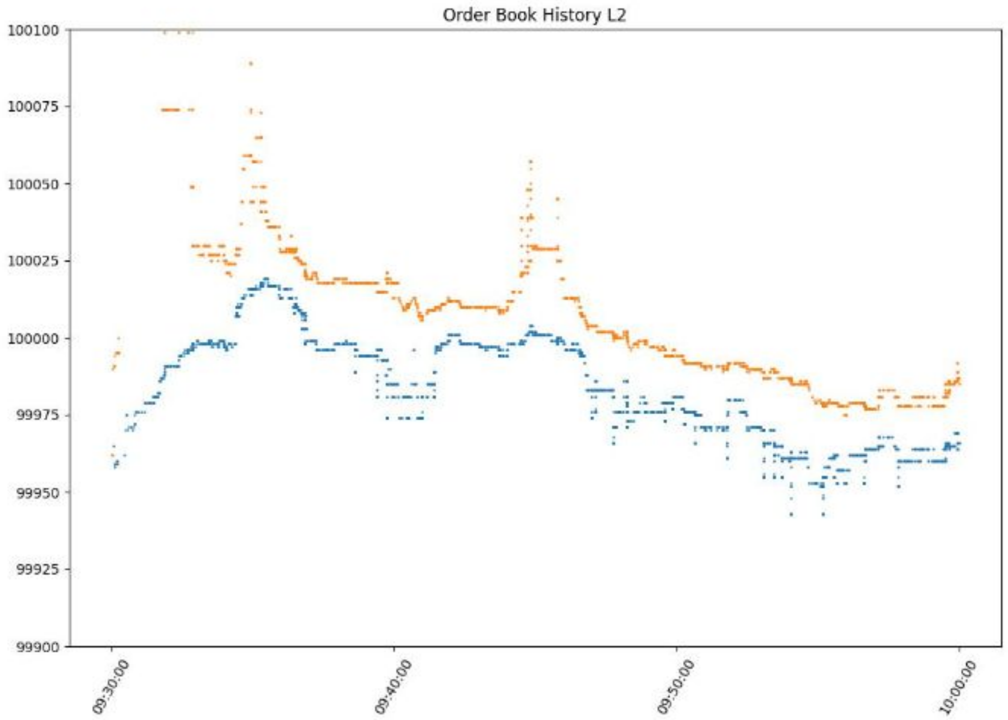


Fig9. Simulated Bid Price and Ask Price at Level 5

5.Simulated Stock Market Analysis

Property of the simulated market

- There are more fluctuations when the market just open (the first 10 minutes) ;

- The numbers of order submissions overall are spread out across the half hour;

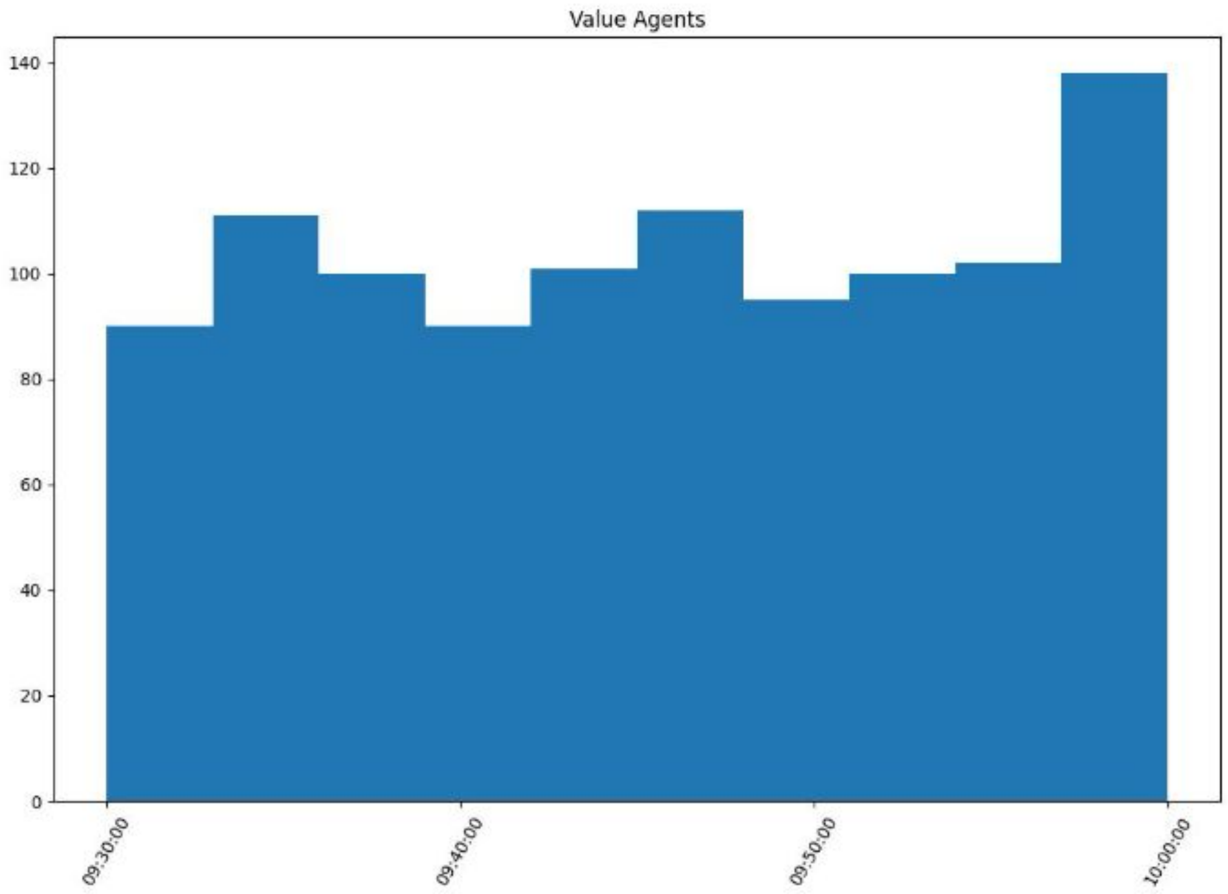
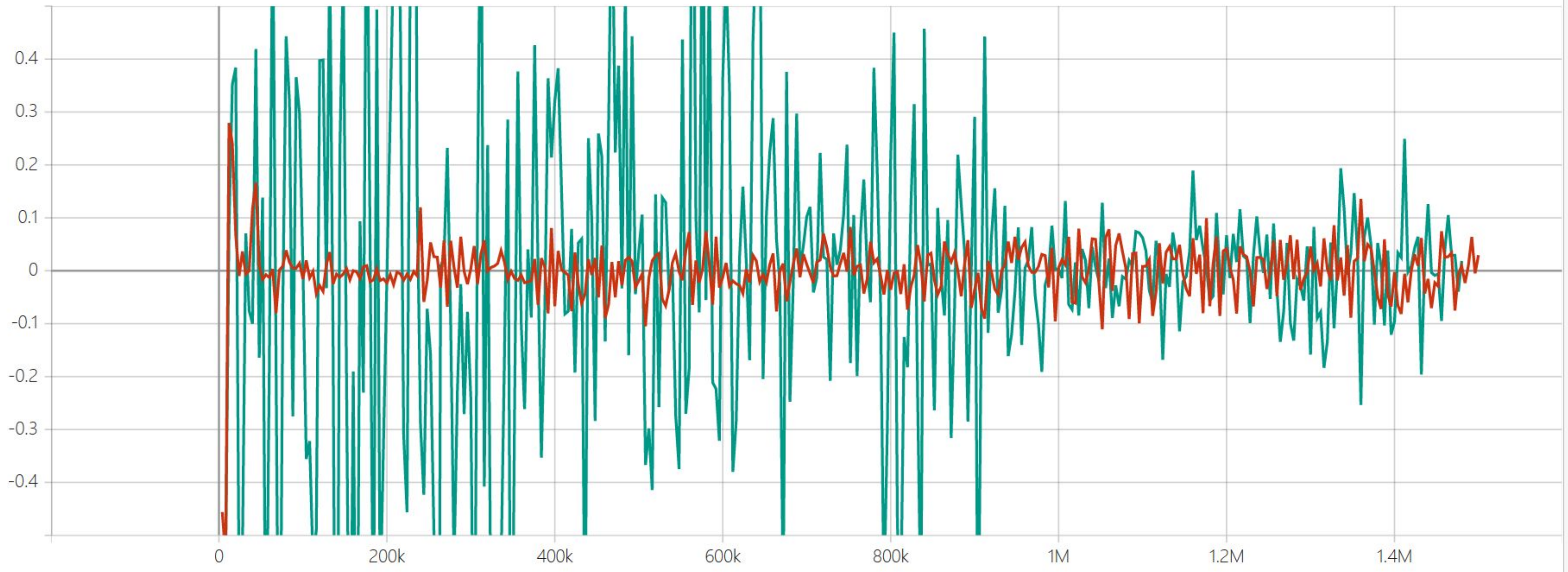


Fig10. Histograms of Order Submission Times of Value Agents

6.Trading Policy Results

PPO Episode Reward (Mean) per Training Episode



Conclusion

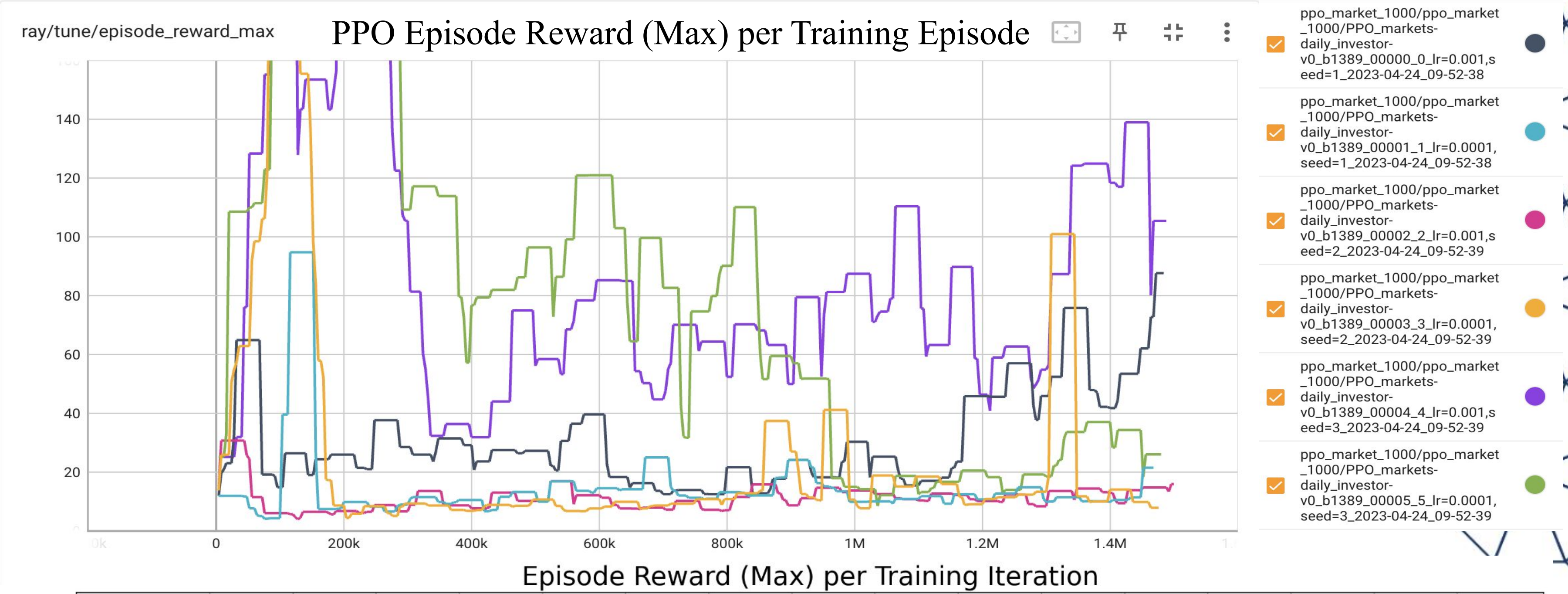
After 1.4 million steps, the PPO model demonstrates reduction in variance, and convergence towards average, resulting in a more robust policy.

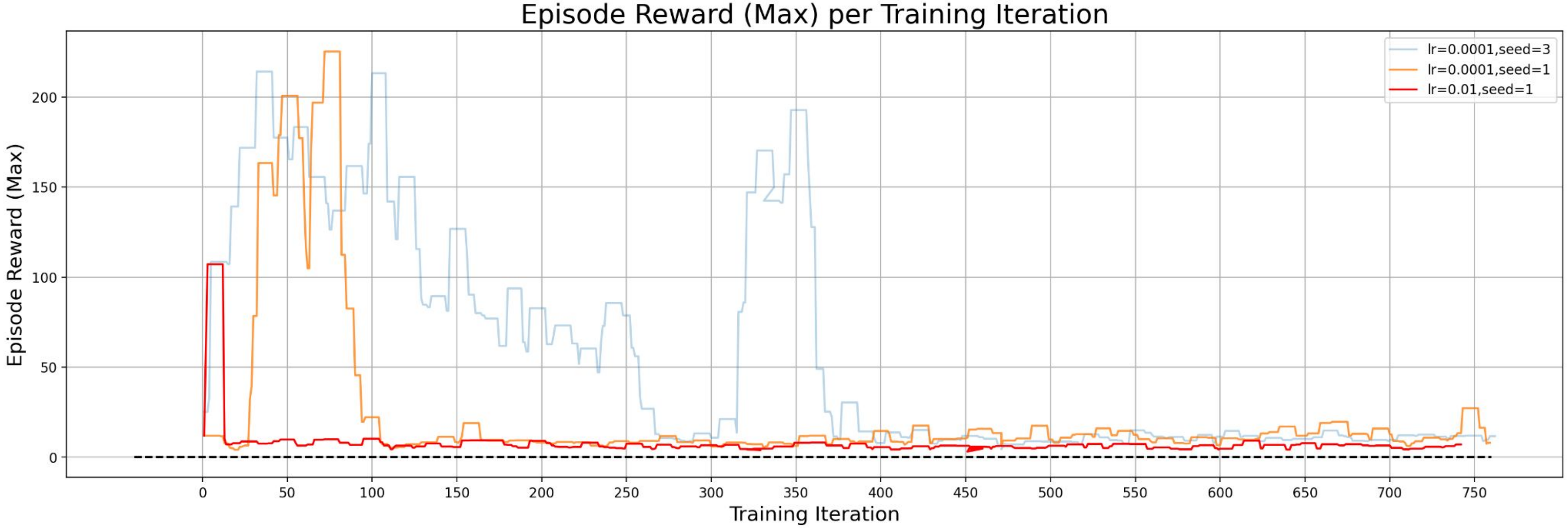
6.Trading Policy Results

Conclusion

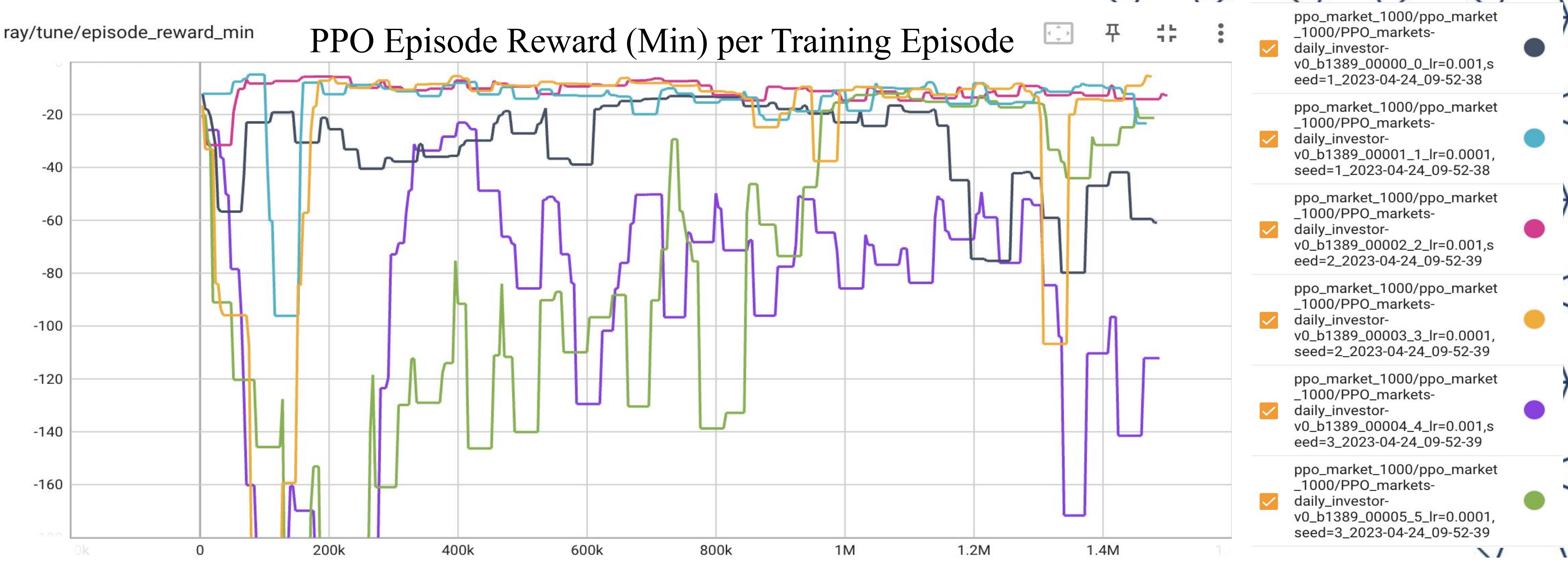
After 3 million steps, the PPO model with learning rate 0.01 can quickly converge from negative rewards to slightly positive rewards, and performs more stable.

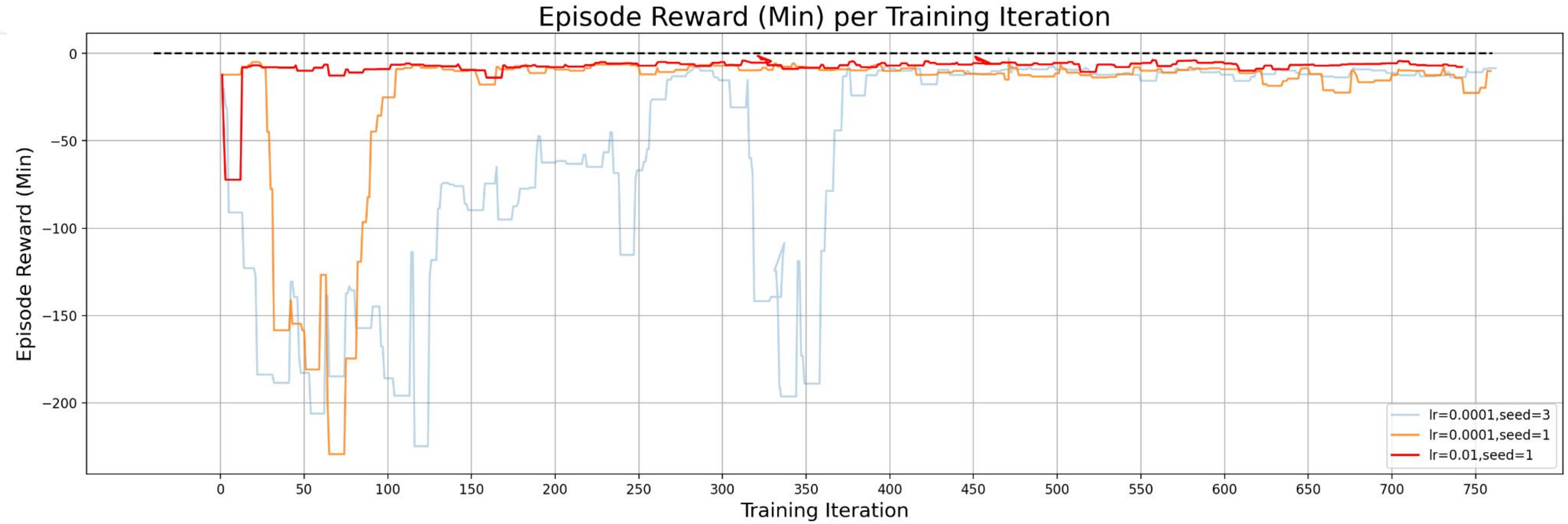
6.Trading Policy Results





6.Trading Policy Results



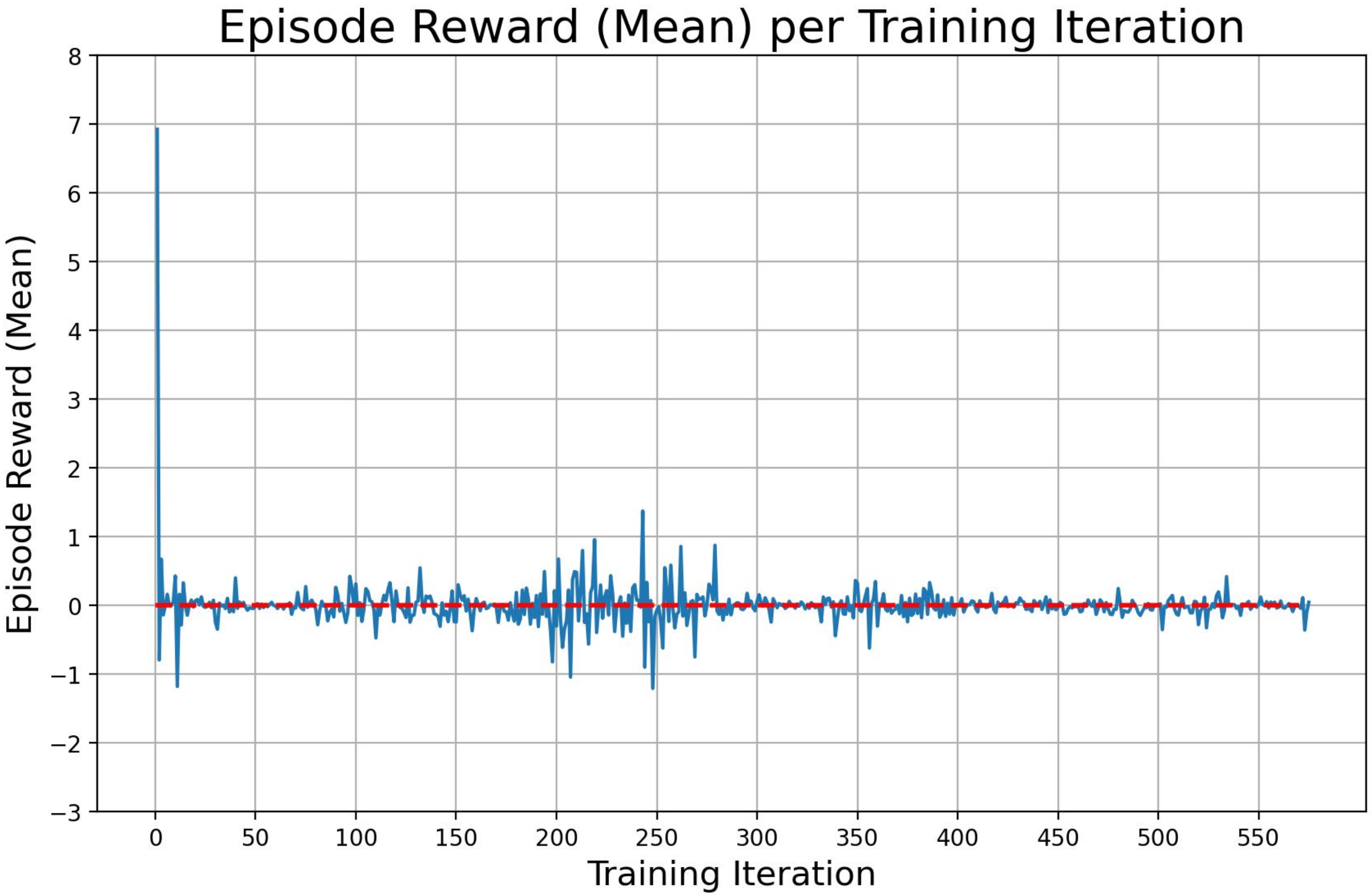


6.Trading Policy Results

Conclusion

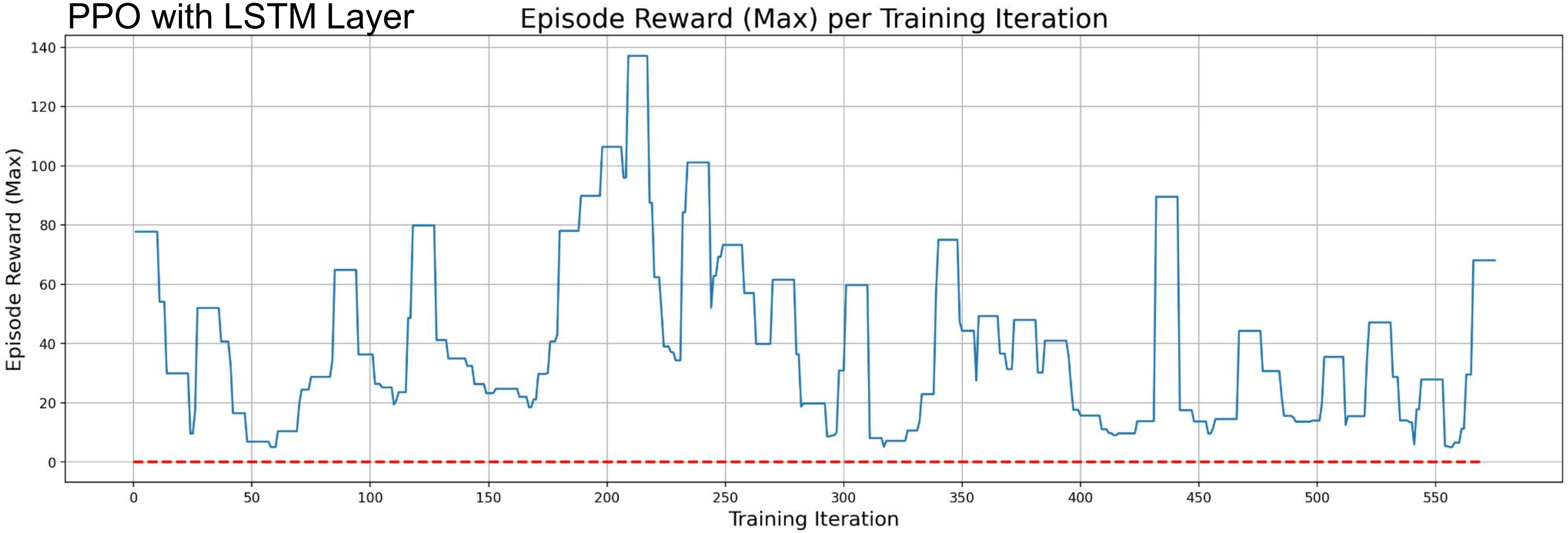
After 570 training iterations, the PPO model with LSTM layers demonstrates reduction in variance, and convergence towards average, resulting in a more robust policy.

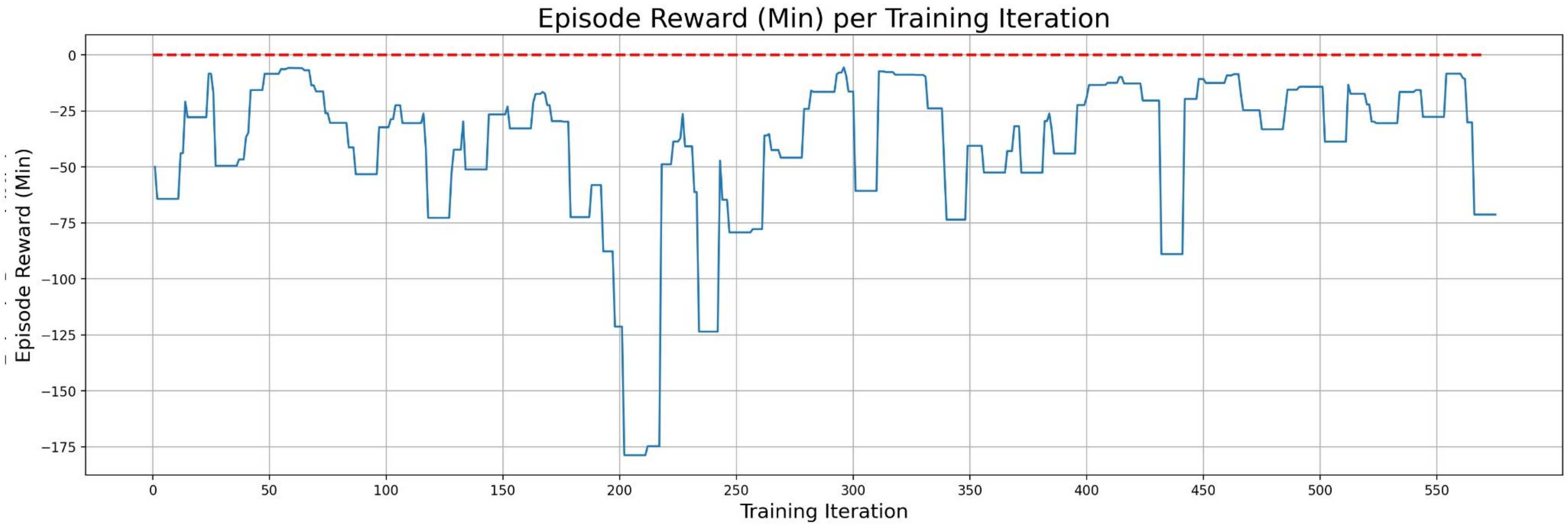
PPO with LSTM Layer



6.Trading Policy Results

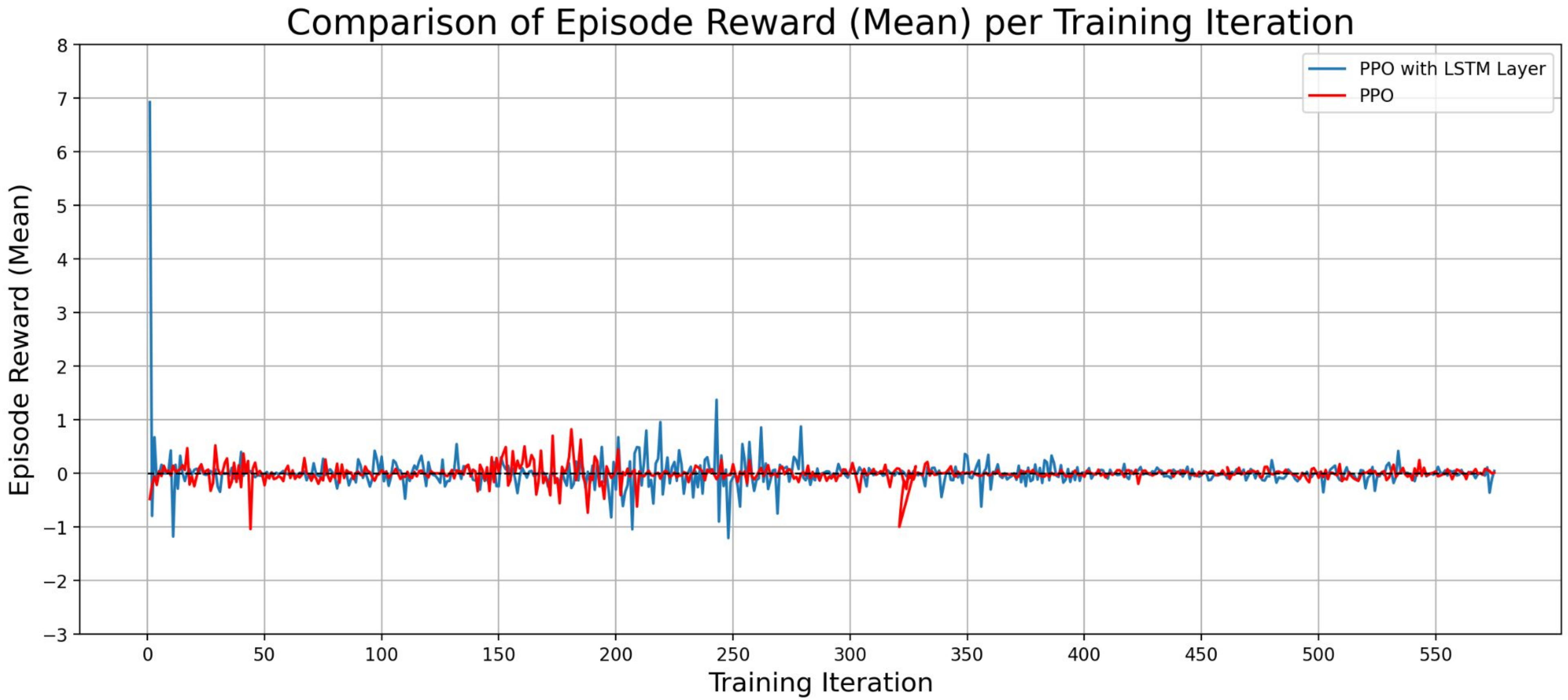
PPO with LSTM Layer





6.Trading Policy Results

Comparison of PPO and PPO with LSTM Layer under same Hyper-Parameters



6.Trading Policy Results

```python
class policyPassive:
    def __init__(self):
        self.name = 'passive'

    def get_action(self, state):
        return 1

class policyAggressive:
    def __init__(self):
        self.name = 'aggressive'

    def get_action(self, state):
        return 0

class policyRandom:
    def __init__(self):
        self.name = 'random'

    def get_action(self, state):
        return np.random.choice([0,1])

class policyRandomWithNoAction:
    def __init__(self):
        self.name = 'random_no_action'

    def get_action(self, state):
        return np.random.choice([0,1, 2])
```

Other Baseline Trading Strategies

- Passive Policy: No transaction at all
- Aggressive Policy: Always buying stocks
- Random Policy: Actions of whether buy or sell are chosen randomly
- Random Policy including holding: Actions of whether buy, sell or hold are chosen randomly

6.Trading Policy Results

Rewards and Profits for Different Trading Policies

Metrics of success

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

Compared to other policies, the PPO model achieves the best episode reward and the highest profit, while the DQN model achieves the third best performance with a small loss.

6.Trading Policy Results

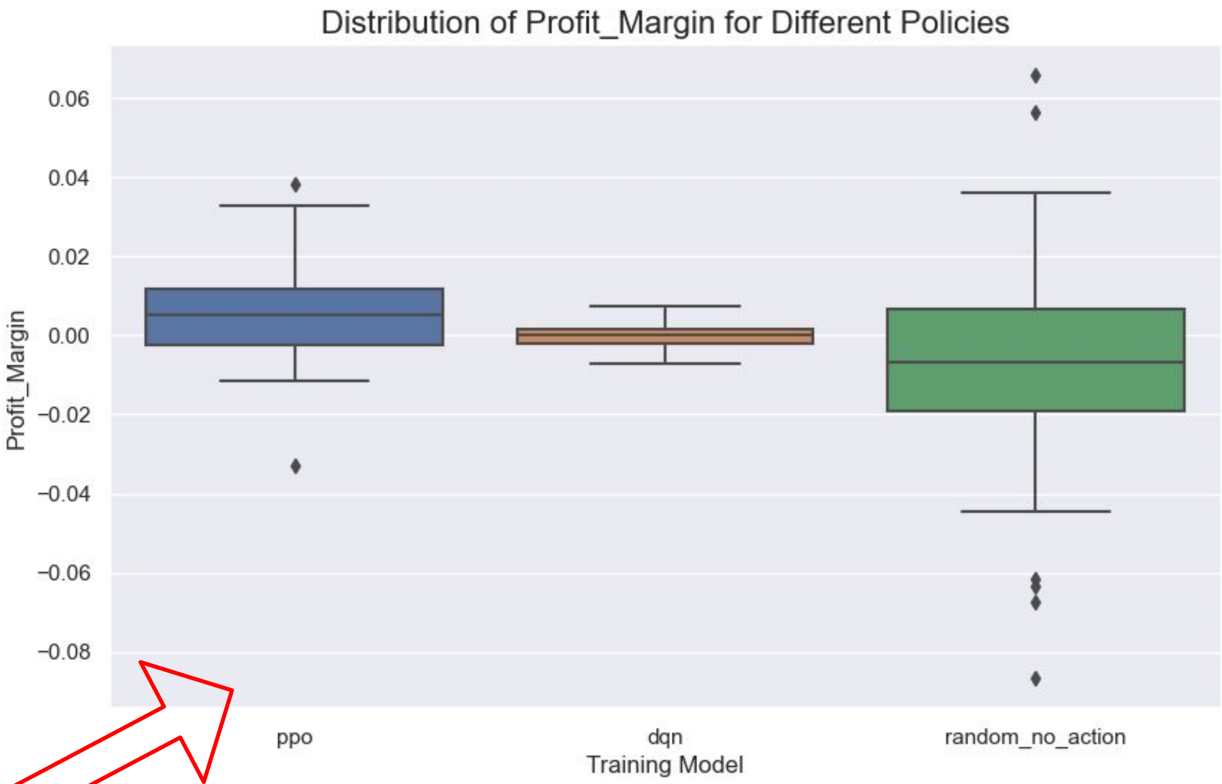
Positive Profits

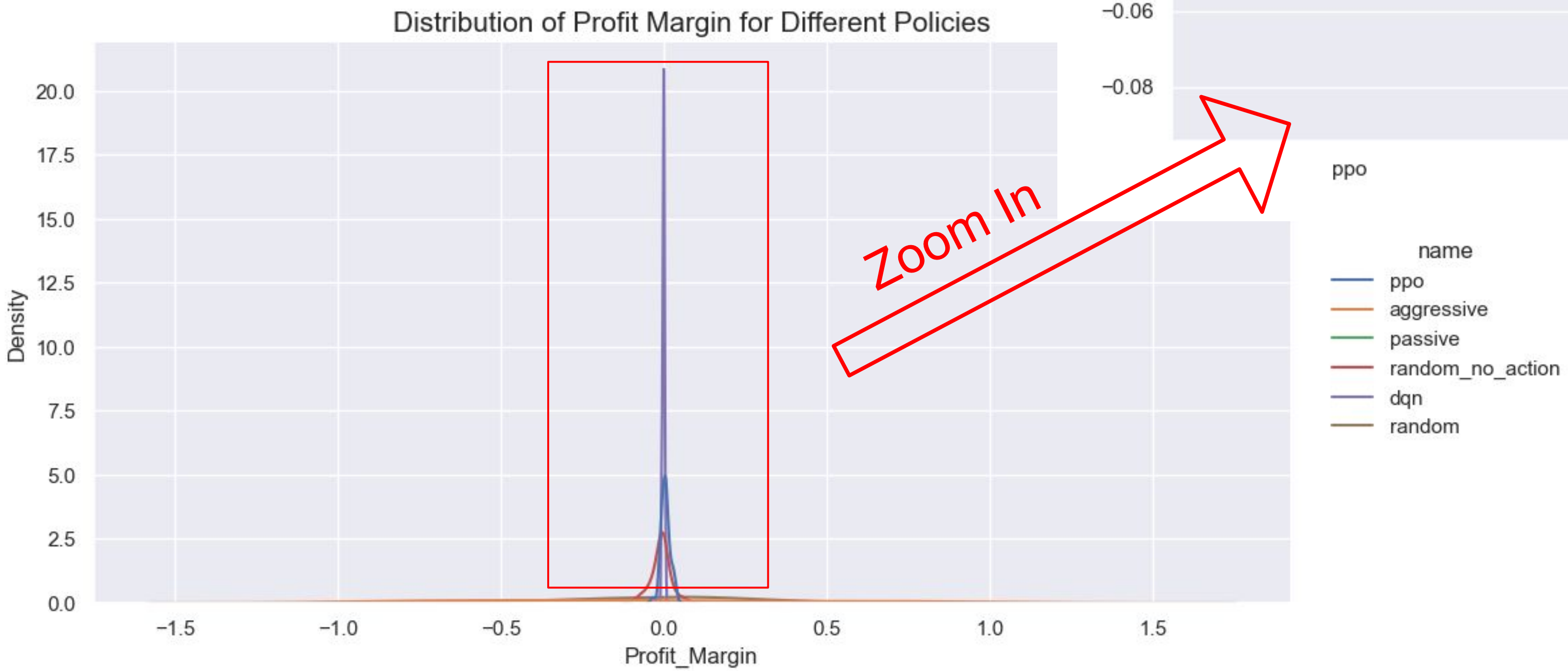
Rewards and Profits for PPO with Different Hyper-Parameters

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

6.Trading Policy Results

Notes: Detailed information of profit margin distribution of different trading policies for 50 testing rounds.





7.Conclusion

- By applying reinforcement learning, we can achieve a trading policy with better performance compared with other baseline policies, such as aggressive and random investment decisions.

- Compared with DQN model, PPO performs better, achieving positive and robust rewards, which can lead to higher investment profits.

- PPO with LSTM layers is more suitable for time series data and can lead to faster convergence.

Thanks for Listening

April 28 2023

Berkeley
UNIVERSITY OF CALIFORNIA