Journal of Electrical and Computer Engineering Innovations
(JECEI)
Journal homepage: http://www.jecei.sru.ac.ir

Research Paper

A Robust Concurrent Multi-Agent Deep Reinforcement Learning based Stock Recommender System

S. Khonsha [1], *M. A. Sarram* [2], *R. Sheikhpour* [3,*]

[1] *Computer Engineering Department, Zarghan Branch, Islamic Azad University, Zarghan, Iran.*

[2] *Computer Engineering Department, Yazd University, Yazd, Iran.*

[3] *Department of Computer Engineering, Faculty of Engineering, Ardakan University, P.O. Box 184, Ardakan, Iran.*

Article Info

Article History:
Received 21 August 2024
Reviewed 06 October 2024
Revised 04 November 2024
Accepted 17 November 2024

Keywords:
Multi-agent
Concurrent learning
Deep reinforcement learning stock recommender system

*Corresponding Author's Email Address:
rsheikhpour@ardakan.ac.ir

Abstract

Background and Objectives: Stock recommender system (SRS) based on deep reinforcement learning (DRL) has garnered significant attention within the financial research community. A robust DRL agent aims to consistently allocate some amount of cash to the combination of high-risk and low-risk stocks with the ultimate objective of maximizing returns and balancing risk. However, existing DRL-based SRSs focus on one or, at most, two sequential trading agents that operate within the same or shared environment, and often make mistakes in volatile or variable market conditions. In this paper, a robust Concurrent Multiagent Deep Reinforcement Learning-based Stock Recommender System (CMSRS) is proposed.

Methods: The proposed system introduces a multi-layered architecture that includes feature extraction at the data layer to construct multiple trading environments, so that different feed DRL agents would robustly recommend assets for trading layer. The proposed CMSRS uses a variety of data sources, including Google stock trends, fundamental data and technical indicators along with historical price data, for the selection and recommendation suitable stocks to buy or sell concurrently by multiple agents. To optimize hyperparameters during the validation phase, we employ Sharpe ratio as a risk adjusted return measure. Additionally, we address liquidity requirements by defining a precise reward function that dynamically manages cash reserves. We also penalize the model for failing to maintain a reserve of cash.

Results: The empirical results on the real U.S. stock market data show the superiority of our CMSRS, especially in volatile markets and out-of-sample data.

Conclusion: The proposed CMSRS demonstrates significant advancements in stock recommendation by effectively leveraging multiple trading agents and diverse data sources. The empirical results underscore its robustness and superior performance, particularly in volatile market conditions. This multi-layered approach not only optimizes returns but also efficiently manages risks and liquidity, offering a compelling solution for dynamic and uncertain financial environments. Future work could further refine the model's adaptability to other market conditions and explore its applicability across different asset classes.

Introduction

Choosing the appropriate share for investment and accurately identifying the time to buy and sell shares is considered a challenging task. To become a professional stock trader and make successful transactions, investor must have significant experience and be able to recognize the trend of share price changes. Prediction tools for detecting market trends and forecasting stock

movements have been popular for several years. Various techniques and models are used to predict stock prices efficiently.

Linear regression [1] is introduced to predict continuous price values. Time series models such as the ARIMA (AutoRegressive Integrated Moving Average) [2] have also been proposed to model historical stock price data. Machine learning algorithms like LSTM (Long Short-Term Memory) [3], [4], RNN (Recurrent Neural Networks) [5] are also provided for stock price prediction and market trend detection. The variable, non-linear and fluctuating nature of the stock market has prevented the proposed models and algorithms from predicting well and being able to perform well in highly volatile markets and crashes. However, another class of techniques called reinforcement learning, which have worked well in computer games [6], [7] and performed as well as or better than humans, can be used to predict stock prices. Reinforcement learning in finance and stock trading involves training intelligent algorithms to make trading decisions by interacting with financial markets. These algorithms learn optimal strategies through trial and error, adapting to market dynamics to maximize returns and manage risks and the reinforcement agent is encouraged with any action that makes reaching the goal with more points, and is punished on the contrary. Accordingly, some researchers have designed trading strategies based on deep reinforcement algorithms [8]-[12], leveraging the power of neural networks to capture intricate market patterns and make informed decisions. These approaches aim to enhance portfolio management, risk assessment, and adaptive trading in the dynamic and complex landscape of financial markets.

Because the financial market is constantly changing and very complex, it is not convenient to learn the optimal trading policy using only an DRL agent [13]. So, in recent researches for automatic trading strategies [13]-[18], different multiagent deep reinforcement learning models have been used to extract features in order to display the environment observations of the reinforcement agent. One of the challenges that these systems face is how to accurately represent the agent's environment, which can give the agents a correct perspective for correct action. All the Multiagent SRS studies that have been done, trading agents use shared data source for learning. In the complex and volatile stock market environment, various distributed and decentralized data sources reflect market changes from different perspectives. A key challenge is obtaining the temporal characteristics of these data types and feeding them into agents differently to provide a deeper understanding of the stock market environment. In order to mitigate this challenge, we use various data sources such as Google stock trends and fundamental data and technical indicators along with historical price data to select and recommend suitable stocks to buy or sell by multiple agents concurrently.

Besides that, due to the different behavior of distinct stocks in financial markets, the presented approaches face unsolved challenges yet. Considerable, all parts of the RL environment only reproduce common historical price data to train trading agents for all assets which makes the efficiency of the algorithm not acceptable for some out-of-sample stocks. While some stocks have fundamental behavior, some are price driven and some of them follow the overall movement of the market. To solve the presented challenge, nonidentical from the research done, we hypothesize that the treatment of stock selection for buy or sell trade specially in the time of volatile market in the form of the stock-based feature selection and learning the trading behavior of each stock independently and the cooperation of agents in choosing the final decision is useful to make robust profitable trading decisions.

Another challenge that automated trading systems face is that in highly volatile markets, they face a lack of liquidity to reduce the average share purchase price. Therefore, we define the novel reward function in such a way that the agent always has adequate liquidity in order to avoid excessive losses in fluctuating markets. The summarized contributions of this paper are as follows:

- We proposed a Concurrent Multiagent Stock Recommender System (CMSRS) to generate collaborative recommendations.
- We used diverse data to co-train multiple concurrent DRLs to robustly detect market trends.
- For different stocks, the Concurrent RL trading agents have a custom-built environment for training. More precisely, effective features are extracted for different stocks using the dvlw state formation, and RL agents are trained using these features as states separately and update shared policy.
- To mitigate losses in bear markets, we defined a novel liquidity-based reward function. This reward function gives points to the reinforcement agent based on the current amount of cash so that the agent can always maintain cash at a suitable level.

The organization of this article is divided as follows: Next section provides a bibliometric-based review of previous studies on DRL and Multiagent RL trading which was extracted on August 20, 2024. Then we present the preliminaries and background concepts needed to define the RL framework for recommender systems as well as the classification of different reinforcement learning algorithms. After that, we describe the proposed method to solve the raised challenges, including the complete Multiagent framework based on multi-layer

recommender structure optimization and the trading agent training process.

Final Section shows the experimental results, including the parameters optimization results for different stocks and the results of the tests and methods applied to compare the performance of the algorithms and presents the trading performance of a trained CMSRS in a real environment. Finally, we concludes the paper with conclusions from this study and provides directions for future research.

Related Work

In recent years, recommender systems [19]-[21] and reinforcement learning (RL) methods have experienced significant advancements, leading to widespread adoption across various complex problem domains. RL, in particular, has led to an increase in the adoption of its algorithms to solve many problems, even those that seemed difficult to solve in the past. In the field of stock trading, these methods have recently received more attention.

Fig. 1 demonstrates annual scientific production trend in this field.

Fig. 1 shows since 2018, researchers have paid more attention to the use of reinforcement learning algorithms in stock trading.

Especially, the attention of individual and institutional investors and financial researchers has also been drawn to DRL algorithms. Many investors are looking for algorithms that can provide reliable investment recommendations by taking into account the turbulent, changing and dynamic conditions of financial markets and considering all aspects affecting these markets. Table 1 shows a systematic comparative review using bibliometrics on the research done in the field of stock trading using RL techniques and Multiagent RL (MARL). According to Table 1, the analysis from 1998-2024 shows that 60 out of 264 RL-related documents contain the keyword Multiagent (from 2002-2024). These documents cover various applications, including but not limited to Stock Recommender Systems.

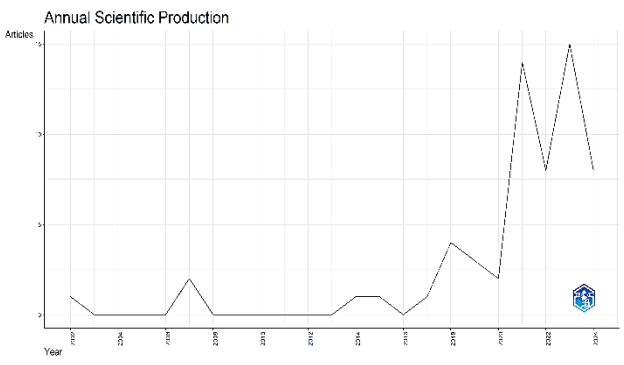

Fig. 1: Annual scientific production.

Table 1 indicates that no book chapters on Multiagent topics have been published. Additionally, only one conference review and one journal review on techniques related to Multiagent Reinforcement Learning (MARL) have been published.

Table 2 summarizes recent research related to 'reinforcement learning in stock trading' using specific keywords.

Table 1: Descriptive statistics of the studies conducted in 1998 to 2024: Reinforcement learning stock trading systematic review

Description	RL Results	MARL Results
MAIN INFORMATION ABOUT DATA		
Timespan	1998:2024	2002:2024
Sources (Journals, Books, etc)	148	44
Documents	264	60
Annual Growth Rate %	9.19	9.91
Document Average Age	5.02	4.06
Average citations per doc	13.7	14.53
References	6781	1132
DOCUMENT CONTENTS		
Keywords Plus (ID)	1293	268
Author's Keywords (DE)	470	102
DOCUMENT TYPES		
Article	99	14
Book chapter	8	0
Conference paper	125	20
Conference review	7	1
Review	5	1

In Fig. 2, the network between the top research sources, researcher countries and keywords are presented, the left side is cited sources, the middle is the country names and on the left side the keywords are specified.

Table 2: Very recent research with keywords “reinforcement learning stock trading”

Reference	Journal		Remarks
[22]	Expert systems with applications (2024)	Contributions	A Cascaded LSTM (CLSTM-PPO) model is utilized. Initially, LSTM is applied to extract time-series features from daily stock data. Additionally, another LSTM model is employed within the RL strategy functions for further training.
		Disadvantages	Instabilities during training.
[18]	Information Sciences (2023)	Contributions	An RRL algorithmic trading model by using self-attention to extract hidden temporal representation of series with hybrid loss is introduced.
		Disadvantages	High computational complexity due to sequential training of model
[22]	Knowledge based systems (2023)	Contributions	DRL-UTrans model is proposed that uses architecture of U-Net and transformer layers combined to RL for trading of single stock.
		Disadvantages	It does not support multi-stock trading and portfolio construction
[24]	Applied soft computing (2023)	Contributions	A multi-agent model is introduced that multiple generative adversarial networks cooperate to regenerate historical price of stocks to resolve generalization issue in stock trading
		Disadvantages	It does not support multi-stock trading and portfolio construction
[13]	Neural Computing and Applications (2023)	Contributions	A Multi-agent DRL is proposed that formulate trend consistency factor into reward function as a regularization term for portfolio construction
		Disadvantages	Accurate trend consistency/inconsistency calculation is a challenge
[25]	Advances in Transdisciplinary Engineering (2022)	Contributions	Three actor critic RL models (SAC, TD3, A2C) is employed to construct an ensemble strategy to automate stock trading
		Disadvantages	Unstable result to choose an agent with best Sharpe ratio
[26]	Expert systems with applications (2022)	Contributions	A deep reinforcement learning model for asset-specific trading rules is investigated that uses different feature extraction modules
		Disadvantages	It does not support multi-stock trading and portfolio construction
[27]	Expert systems with applications (2022)	Contributions	The ResNet-LSTM actor model for crypto currency trading rules investigated that uses ResNet architecture
		Disadvantages	It does not use reinforcement learning but use classification approach

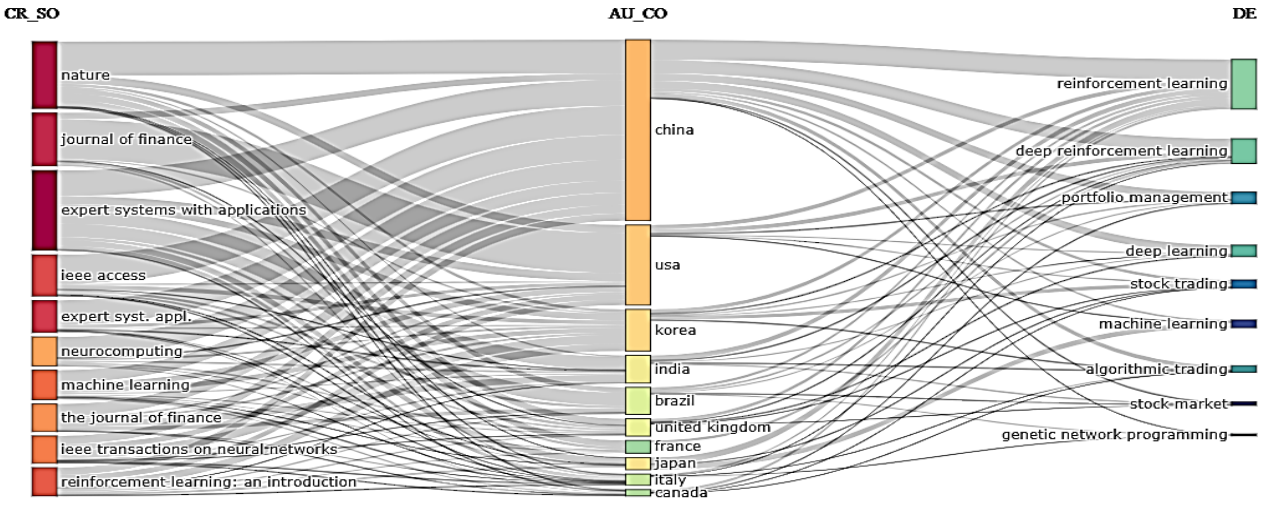

Fig. 2: Cited Sources (CR_SO), Countries (AU_CO) and keywords (DE).

Fig. 3 and Table 3 show word cloud of the most relevant words used in documents.

Fig. 3: World cloud.

Table 3: Most frequent words

Terms	Frequency
reinforcement learning	200
Commerce	164
financial markets	140
electronic trading	115
deep learning	97
Investments	84
learning systems	65
learning algorithms	56
reinforcement learnings	50
Profitability	45
trading strategies	42
stock trading	40
decision making	35
algorithmic trading	24
portfolio managements	23

Preliminaries and Backgrounds

A. Single Agent Reinforcement Learning

In decision-making problems, the Markov decision process (MDP) serves as a framework where outcomes are partially random and partially influenced by the decision maker. MDPs are commonly used to describe the environment in RL, with RL models being a type of state-based model that leverages MDPs. In essence, RL involves training an agent through a system of rewards and punishments. The RL agent observes the current state, performs an action in the environment, receives a reward for that action, and this action transitions the environment to the next state. Fig. 4 schematically shows MDP process in RL.

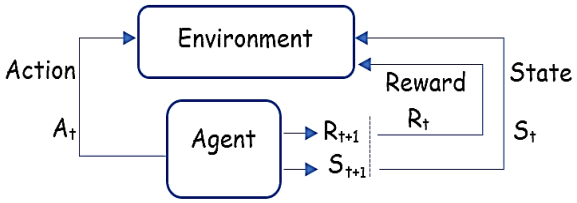

Fig. 4: MDP process in RL [28].

In a trading Reinforcement Learning algorithm:

- At time t, the agent (reinforcement algorithm) assesses the current state (s_t) of the environment, which encompasses various factors such as cash balance, stock prices in the portfolio, the quantity of each share, the time since a share was purchased, technical indicators, fundamental parameters of the share, the share board details (including the number of individual and institutional buyers and sellers), the volume of shares bought by individual and institutional traders, and other relevant features that define the current state of the environment..
- The agent selects the optimal action (a_t) from the available options (buy/sell/hold).
- The environment transitions to a new state (s_{t+1}).
- The environment generates a reward (r_t), which reflects the change in the portfolio's value (increase/decrease/no change).

The process of selecting an action based on the current state is governed by the policy function ($\pi(s_t) = a_t$), which maps states to actions. In reinforcement learning, the system always consists of an environment with a set of states, actions, a policy function (which guides transitions between states), and rewards expressed as numerical values. The reinforcement learning agent continuously observes the current state, uses the policy function to determine the best action, and receives a reward for that action.

This cycle repeats until an end state is reached. The agent's goal is to maximize the reward, with an optimal policy being one that achieves the highest possible reward. Reinforcement learning algorithms vary widely, and their classification in Fig. 5 is based on the specific components they employ to construct the workflow outlined in Fig. 4, ultimately aiming to achieve the optimal policy. In recent years, multi-agent models [13]-[17], [24], ensemble models [8], [25], [29], and models incorporating autoencoders [30] have also been introduced for portfolio optimization.

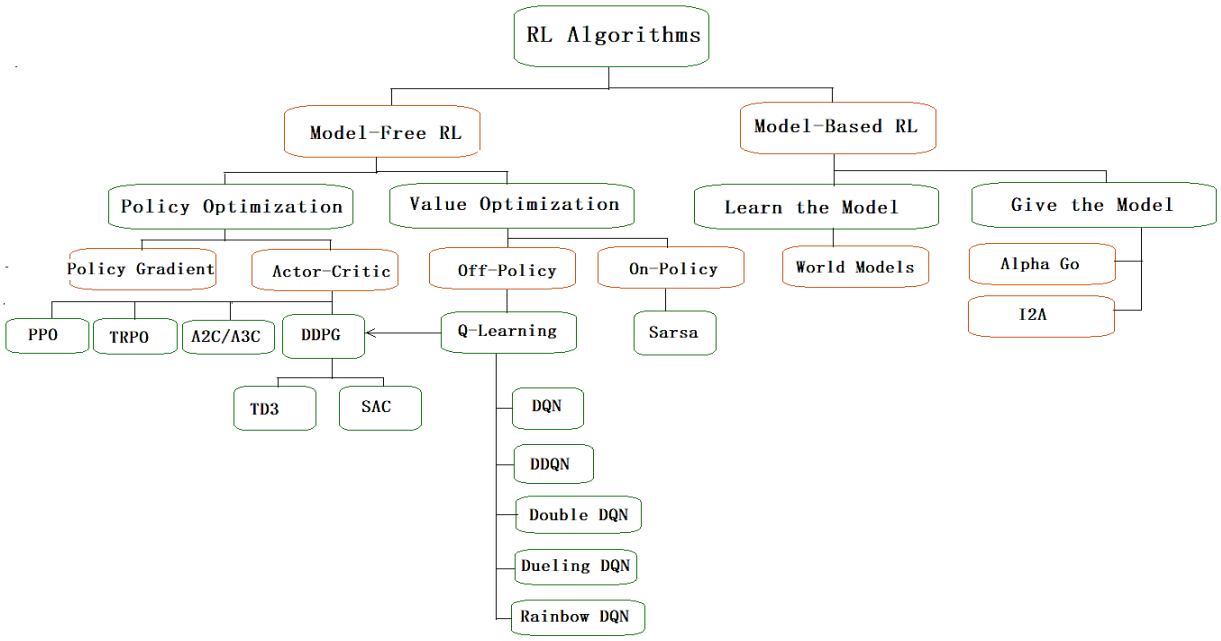

Fig. 5: RL Algorithms [34].

The workflow of all reinforcement learning algorithms typically includes the following steps:

1. Initialize the policy (π) with random parameters.
2. Using the current policy, select the action (a) with the highest probability and store the obtained reward (r) along with the states before (s) and after (s_(t+1)) the action in the experience memory (D).
3. Choose a model to refine the policy.
4. Repeat step 2 to gather more experience with the improved policy and continue refining the policy.

In other words, a common approach to finding an optimal policy that maximizes the expected cumulative discounted reward for each state is policy iteration. This method is particularly useful when faced with multiple options, each with its own distinct rewards and risks. Policy iteration involves a two-stage process that alternates between policy evaluation and policy improvement.

In the policy evaluation stage, we intend to find the exact value function for our current policy. To achieve this goal, we iteratively apply the Bellman equation defined as (1) until we reach convergence.

$$\mathcal{V}_{\pi}(\mathcal{s}) = \sum_{\mathcal{s}',r} \mathrm{p}\,(\mathcal{s}', \mathrm{r} | \mathcal{s}, \pi(\mathcal{s}))[\,\mathrm{r} + \gamma \mathcal{V}_{\pi}(\mathcal{s}')] \qquad (1)$$

where $\mathcal{s}'$ represents the next state, and $\pi(\mathcal{s})$ is the action taken from state $\mathcal{s}$ under policy π. The transition probability, denoted as p, is the likelihood of moving from state $\mathcal{s}$ to the next state $\mathcal{s}'$ when performing action $\pi(\mathcal{s})$ and receiving reward r. The discount factor $\gamma \in [0,1]$ accounts for the time value of rewards.

In essence, rewards may not be immediately received by the agent. Early rewards are generally more predictable and likely, so they are prioritized over potential long-term rewards. In sequences, even larger rewards are discounted if they are further in the future, as the agent is uncertain about receiving them. The discount factor (γ) is used to adjust the value of future rewards. A higher γ means that the agent places more importance on long-term rewards, while a lower γ indicates a greater focus on short-term rewards.

In the policy improvement stage, as shown in (2), the process involves repeatedly applying the Bellman optimality operator.

$$\pi'(\mathcal{s}) = \mathrm{argmax}_{a} \sum_{\mathcal{s}',r} \mathrm{p}\,(\mathcal{s}', \mathrm{r} \mid \mathcal{s}, a)[\,\mathrm{r} + \gamma \mathcal{V}_{\pi}(\mathcal{s}')] \qquad (2)$$

Similarly, the value of choosing action a in state $\mathcal{s}$ under policy π is denoted as Q_{π}(s,a). This represents the expected cumulative reward of taking action a in state $\mathcal{s}$, with all subsequent actions being determined by the policy π, as expressed in (3).

$$\mathrm{Q}_{\pi}(\mathcal{s}, a) = \sum_{\mathcal{s}',r} p\,(\mathcal{s}', r | \mathcal{s}, a)[\,r + \gamma \mathrm{Q}_{\pi}(\mathcal{s}', a')] \qquad (3)$$

The Q_{π} is called the value-action function for the policy π. The value function of $\mathcal{V}_{\pi}$ and Q_{π} can be estimated with repetitive experiments. For example, if an agent follows a policy and averages the amount it receives from experiences for each situation, after an infinite number of repetitions, the value of $\mathcal{V}_{\pi}(\mathcal{s})$ will converge to the real value. Now, if this average is kept for each state-action pair separately, then $\mathrm{Q}_{\pi}(\mathcal{s}, a)$ will be estimated and stored in the table. Such estimation methods are called Monte Carlo methods, which include averaging over a large number of random samples of the real return reward.

For complex and dynamic problems like stock trading and portfolio optimization, which involve high-

dimensional and continuous state-action spaces, finding the exact optimal solution using lookup table-based methods is often impractical. Instead, rough approximators such as neural networks are used. A neural network comprises several layers, where the input layer receives the state vector s, and the output layer determines the action a.

Fig. 1 illustrates the training process of an agent based on a Q-Network with experience replay memory. This architecture consists of three main components:

- Q-network Q(s, a;θ) where θ determines the agent's behavioral policy,
- Q-target network Q(s', a' ;θ′), which is used to obtain the Q values for the error part of the Deep Q-Network (DQN) and
- Experience Replay Memory, which the agent uses to randomly transfer samples to train the Q network.

The replay memory is used to address the issue of high correlation between consecutive examples in the problem, which can slow down convergence when used for training a neural network. To mitigate this, transitions—comprising the state, action, resulting next state, and associated reward—are stored in a replay memory. These transitions are then randomly sampled from the memory for training the network. By doing so, the network can learn from a more diverse set of experiences, reducing the impact of correlation and improving the stability of the learning process. Additionally, because these experiences are valuable, the replay memory allows them to be reused multiple times for more efficient training. The target network, which shares the same structure as the main network, is periodically updated by copying the weights from the main network to the target network θ' after a fixed number of steps. This approach helps to reduce the negative effects of network fluctuations, leading to more stable training and faster convergence.

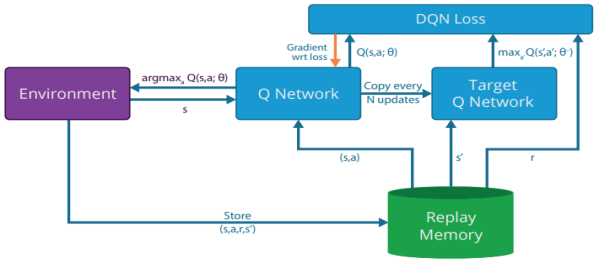

Fig. 6: DQN architecture [31].

B. Multi Agent Reinforcement Learning

In MARL, multiple agents interact and learn from each other in order to better coordinate their actions in the environment to maximize target long-term reward [30]. This coordination is achieved through a process known as cooperative learning, where agents share their experiences with each other and learn from each other's experiences. This allows agents to learn from each other and improve their policies. Specially in the case of the RL agent that uses the neural network approximator, the direct use of single-agent methods in multi-agent frameworks violates the Markov assumptions required for convergence because other agents are considered as part of the environment, and the environment from the perspective of each agent seems to be non-stationary [32]. As described in [33], if we know the actions that are taken according to the local observations of each agent, even with changing policies, the environment has the property of being stationary.

Algorithm 1 Multiagent RL with N **Agent**

Create experience memory D with M size
Create Q function by θ random weights
Create $\hat{Q}$ target function by $\theta' = \theta$ weights
for episode from 1 to MaxEpisode do
  Initialize a random process N for action exploration
  Receive sequence s_1 and preprocessed sequence $\phi_1 = \phi(s_1)$
  for t from 1 to MaxTimeStep do
    $\forall\, agent\ i \in \{1, \dots, N\}\ a^i_t = \mu^i_\theta(o^i) + N_t$
    Execute all N actions a_t and observe reward r_t and state s_{t+1}
    Set $s_{t+1} = s_t$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    for agent i = 1 to N do
      Sample random mini-batch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$from D
      $y_j = r^i_j + \gamma Q(\phi(s_t), a'; \theta')$
      Set $y_j = \begin{cases} r_j, & for\ terminal\ \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi(s_t), a'; \theta'), & for\ non-terminal\ \phi_{j+1} \end{cases}$
      Do a gradient descent step on $(y_j - Q(\phi_j), a_j; \theta))^2$ respect to the θ parameters
      Each C timesteps assign $\hat{Q}$ = Q
    end for
end for

Mathematically, MARL is an MDPs generalization for multi-agent reinforcement learning and can be defined as $(N, S, A_{1:N}, T, R_{1:N}, \gamma)$ tuple, where N is the number of RL agents, S denotes states set, $A_{1:N}$ indicates actions of N agents, T describe probability transition function from states and actions to [0,1] and $R_{1:N}$ is average reward received by N agents, depending on the type of multi-agent MDP, the reward function of the agents can be the same or different. In the multi-agent case, each agent i, in the t-th iteration, only updates the value of $Q(s_t,i,a_t,i)$ and leaves the other entries to the Q function unchanged. Algorithm 1 shows the multi-agent learning process in details.

Proposed Concurrent MultiAgent Stock Recommender System

We propose a multi-layer multi-agent stock recommender system based on deep reinforcement algorithm. In the proposed multi-layer CMSRS, we propose centralized value function estimator and decentralized policy networks of RL agents to diminish explained non-stationary issue and stabilize RL agent training in the DRL layer. The CMSRS architecture consists of four distinct layers, each serving a specific purpose in generating stock recommendations for users. These layers typically include:

Data Layer: This layer involves gathering and aggregating various data sources and extracting relevant features from the preprocessed data.

Environment Layer: In this layer, the new risk aversion reward function is proposed to reduce asset variance and decrease maximum percentage loss.

DRL Layer: This layer includes Multi agent DRL model that concurrently train on the environment to reach optimal policy.

Trading Layer: Finally, the contributions made in the above three layers will lead to more robust recommendations to profitable stock trading in the trading layer.

In the data layer, preprocessing and feature extraction is done on the various data sources such as Google stock trends [34], [35] and fundamental data and technical indicators along with historical price data to construct multiple trading environments, so that different feed DRL agents in the DRL layer can have different observations. Fig. 7 shows the proposed multi-layer CMSRS system. The details of each layer are explained in the following subsections.

A. Data Layer

The data layer in a stock recommender system plays a crucial role in gathering and preparing the various data sources required to make informed recommendations. In the proposed architecture, various data sources including Google Trends, fundamental data, and historical daily stock price data (OHLCV) augmented with technical indicators (MACD, RSI, CCI and IDX) is used to form each agent observation separately.

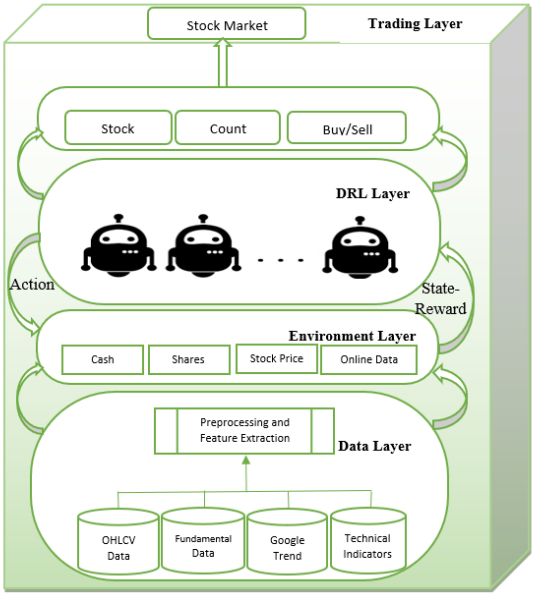

Fig. 7: Proposed CMSRS architecture.

Google trends as a proxy for market sentiment analysis, as analyzed in [35] can improve the Sharpe ratio of trading. Therefore, this feature has been used to feed agents. A normalization preprocess and missing data dealing along with feature extraction is done on the raw data in the data layer. The data layer includes the following tasks in details:

OHLCV Data: This refers to historical price and trading volume data for stocks. It includes the opening, highest, lowest, and closing prices of a stock on a specific day, as well as the trading volume. This data provides insights into price trends, volatility, and trading activity over time. The data layer collects and preprocesses this information, ensuring it is clean, consistent, and ready for analysis.

Fundamental Data: Fundamental indicators are key financial metrics that provide insights into a company's financial health and performance. A fundamental risk aversion indicator could be derived from metrics such as debt-to-equity ratio, earnings per share (EPS), and other relevant financial ratios. This indicator helps assess the financial stability and risk profile of a company. The data layer gathers these fundamental indicators for the stocks under consideration.

Google Trends Data: Google Trends provides information about the popularity of search terms over time. In the context of a stock recommender system, Google Trends data can be used to gauge public interest and sentiment towards specific stocks or sectors. The data layer collects Google Trends data related to search terms relevant to the stocks being analyzed.

The data collected from these sources is often in disparate formats and may require preprocessing to align timestamps, handle missing values, and normalize the data. Once prepared, the data can be integrated into a unified dataset for further analysis.

Overall, the role of data layer in a stock recommender system involves collecting, preprocessing, and integrating diverse data sources to create a comprehensive dataset that captures both historical market trends and external factors affecting stock performance. This integrated dataset serves as the foundation for building CMSRS models that take into account various dimensions of stock behavior and market sentiment.

B. Environment Layer

As mentioned in the preliminaries section, in RL-based learning, the trader agent comes to gain experience by interacting with the market environment through trial-and-error procedure to maximize the reward function. The data on which the agent's observations rely the sensors that provide input to the deep reinforcement algorithm. We assume that the quality and quantity of this data is effective on the amount of reward that the agent can achieve. In addition, the way of rewarding the reinforcement agent is very effective in the convergence of the model. In this layer, the precise definition of the state construction and reward function is proposed, which is explained in detail below.

State Formulation

Defining the state structure in complex environments such as stock trading needs some expertise information. According to our latest information, other researches have used the simple structure of the time window to construct the state. We use novel multi-source n-dimensional vector to represent state. First, we define the *difference vector* in the *t*th timestep for *f*th feature of data, dv_t^f, as the element-wise subtraction of d_{t-1}^f from d_t^f: $dv_t^f := d_t^f \ominus d_{t-1}^f = (d_t^f - d_{t-1}^f)$. Then, *dv* is calculated for a rolling lookback window (*w*) that is selected to the current time and automatically shifts forward with the timesteps, Fig. 8 shows $state_t$ formation for one window. This valuable information is used to construct observations of agents. In experiments this process called difference vector lookback window (*dvlw*) and compared versus simple lookback window (slw).

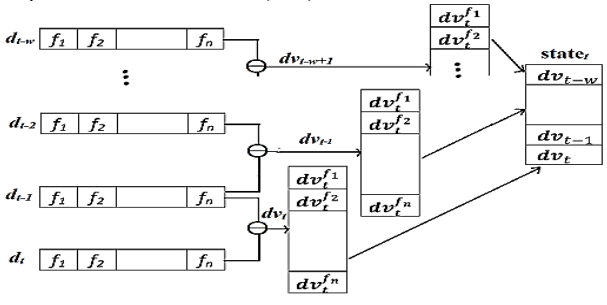

Fig. 8: State formation.

Risk Aversion Reward Function

Designing a reward function for a multi-agent RL that aims to maximize returns, avoid risks, and reduce maximum drawdowns is a complex task that requires careful consideration of various factors. The common definition of the reward in trading agents is the amount of change in the value of the portfolio after the execution of the action that is not risk averse. But in practice, a trader does not prefer his capital balance to be unstable. In other words, high profit along with high loss is not desirable for investors. To model this risk aversion and comprise liquidity requirement, we define precise reward function to dynamically manage cash reserves and penalize the model for not maintaining a reserve of cash. We add a penalty term to the reward function, which aims to reduce the capital variance. This new function enables the model to execute transactions with high confidence and manage cash reserves. Accordingly, we propose the following reward function and compare its effect empirically in real experiments. The immediate reward of ith agent at timestep t, after executing action a_t^i (Sell/Buy shares from jth stock) in state s_t^i and transition to state $s_{t+1}^{'i}$ defined by:

$$r_t^i(s_t^i, a_t^i, s_{t+1}^i) = \left(\mathcal{C}_{t+1}^i + h_{t+1,j}^i * p_{t+1,j}^i\right) - \left(\mathcal{C}_t^i + h_{t,j}^i * p_{t,j}^i\right) - \mathcal{P}_{t+1}^i - \mathcal{C}_0^i \quad (4)$$

where $\mathcal{C}$ denotes cash value and $\mathcal{P}_{t+1}^i$ is cash penalty term as:

$$\mathcal{P}_{t+1}^i = MAX\left(0, \sum\left(\mathcal{C}_t^i + h_{t,j}^i * p_{t,j}^i\right) * \mathcal{P} - \mathcal{C}_t^i\right) \quad (5)$$

where $\mathcal{P}$ is a hyperparameter which determines the liquidity percentage of the portfolio. The cooperative goal of the agents in CMSRS is to maximize the team average cumulative discounted reward obtained by all agents:

$$Q_\pi(s_t, a_t) = \mathbb{E}_{s_{t+1}}[r_t(s_t, a_t, s_{t+1}) + \gamma \mathbb{E}_{a_{t+1\sim\pi(s_{t+1})}}[Q_\pi(s_{t+1}, a_{t+1})] \quad (6)$$

C. DRL Layer

In the DRL layer, we use concurrent multiprocessing training via various observations of the local environment to improve the performance of DRL trading agents. Each trading agent *i* interacts with a market environment to produce transitions independently in the form of $\{s_i, a_i, r_i, s'_i\}$ that respectively are state, action, reward and next state. Then, collection of experience transitions from all the RL agents are stored in a shared replay memory to update a learner. Fig. 9 demonstrates the policy optimization process of the proposed CMSRS. the optimal policy of DRL model is learned by using gradient descent on the loss function: $\mathcal{L}_\theta = \mathbb{E}[(\mathcal{Y}_i - Q(\phi_i, a_i; \theta))^2]$, where θ is policy network`s parameter, ϕ_i is the preprocessed state and $\mathcal{Y}_i = r_i + \gamma \max_{a'} \widehat{Q}(\phi(s_i), a'; \theta')$ is target value.

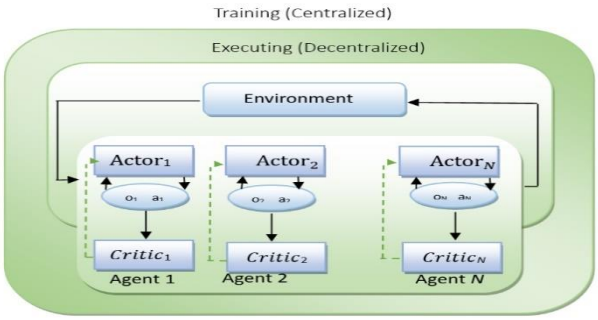

Fig. 9: Multiagent architecture of DRL layer.

So far, several methods have been proposed for RL with neural network approximators, including those based on policy gradients. Proximal Policy Optimization (PPO) [36] has been shown to provide better stability among other RL algorithms. PPO, as the name suggests, seeks to find a proximal policy that uses advantage function ($\mathcal{A}$) as the difference between the future discounted sum of rewards on a certain state and action, and the value function of that policy and thus avoids large policies update. Let the ratio $\mathcal{R}(\theta) = \frac{\pi_\theta'(a_t|s_t)}{\pi_\theta(a_t|s_t)}$, loss function of PPO is:

$$\mathcal{L}_\theta^c = \mathbb{E}[(\min(\mathcal{R}(\theta)\,\mathcal{A}, c\,(\mathcal{R}(\theta),1-\epsilon,1+\epsilon)\,\mathcal{A})] \quad (7)$$

where c denotes clipping operator and ϵ is the bound threshold hyperparameter.

We investigate the concurrent Multiagent PPO algorithm to learn a shared decentralized policy by leveraging team experience from all the PPO agents. At the first of training phase, the parameters of policy of all agents are set to an initial value. Then, for each episode, *N* agents in each timestep sample an action (Buy/Sell) using own deep neural network. After that, the agent executes the action in the trading environment and observes the reward and transfers to the next state. All of team experiences store in the experience replay memory. After collecting samples for an episode, *M* epochs of updating are performed with a small batch of transitions sampled from memory *D* on the loss function of (7) using SGD (stochastic gradient descent). In this architecture, all agents work cooperatively as a team to maximize the team-average cumulative discounted reward.

D. Trading *Layer*

The trading layer in a stock recommender system plays a pivotal role in executing the recommendations provided by a DRL agent. This layer bridges the gap between the recommendations generated by the DRL agent and the actual execution of trades in the financial market. The motivation to create a very robust trading system is achieved by cooperating with several robust models to maximize the cumulative reward and let them trade based on the DRL layer output. Here is an explanation of the key functions of the trading layer:

1. Trade Execution: Once the DRL agent generates stock recommendations based on its learned policy, the trading layer is responsible for executing these recommendations. It converts the agent recommendations into actionable buy or sell orders in the market.

2. Risk Management: The trading layer incorporates risk management strategies to control potential losses. This involves setting limits on the size of trades, diversification across different stocks or asset classes, and implementing stop-loss and take-profit mechanisms to manage trade outcomes.

3. Transaction Costs: The trading layer takes transaction costs into account, including brokerage fees, taxes, and spreads. It aims to optimize trade execution to minimize these costs and enhance overall trading performance.

Experiments

All the experiments are carried out on a computer having a 16 GB RAM with CPU Intel Core i7-10750H and GPU Nvidia GeForce GTX 1080 8 GB dedicated memory, 80 GB of virtual memory has been used to optimize the parameters. Data collection consists of three parts. Historical price data, fundamental data and historical data of Google Trends.

In all experiments, the initial amount of cash balance is 10,000$. We incorporate the transaction cost to reflect market friction, e.g., 0.1% of each buy or sell trade. To control risk during market crash situations the volatility index (VIX) is used that is a real-time U.S. stock market index representing the market's expectations for volatility over the coming 30 days.

In our experiments, we select seven most active stocks from United States stock market due to the high market liquidity, including TSLA, AAPL, AMZN, MSFT, GOOG, META and IBM to evaluate the proposed CMSRS. The first six (META, GOOG, TSLA, MSFT, AAPL, AMZN) have been used for training and evaluation and generalization testing, the last one (IBM) not utilized in training, used to test the robustness of the model and its efficiency. The time period used is from January 1, 2013 to July 1, 2023. One last year, from August 1, 2022 to August 1, 2023, has been used for the trading phase Fig. 10, shows the price plot of six train datasets. We use the following widely used metrics in both research and practice to evaluate the proposed CMSRS:

- Cumulative Return (CR): reflects the overall effect of the trading strategy in a certain period of time
- Sharpe Ratio (SR): returns the earned per unit of volatility, which is a widely used measure of an investment performance.
- Maximum DrawDown (MDD): shows the maximum percentage loss during the trading period.

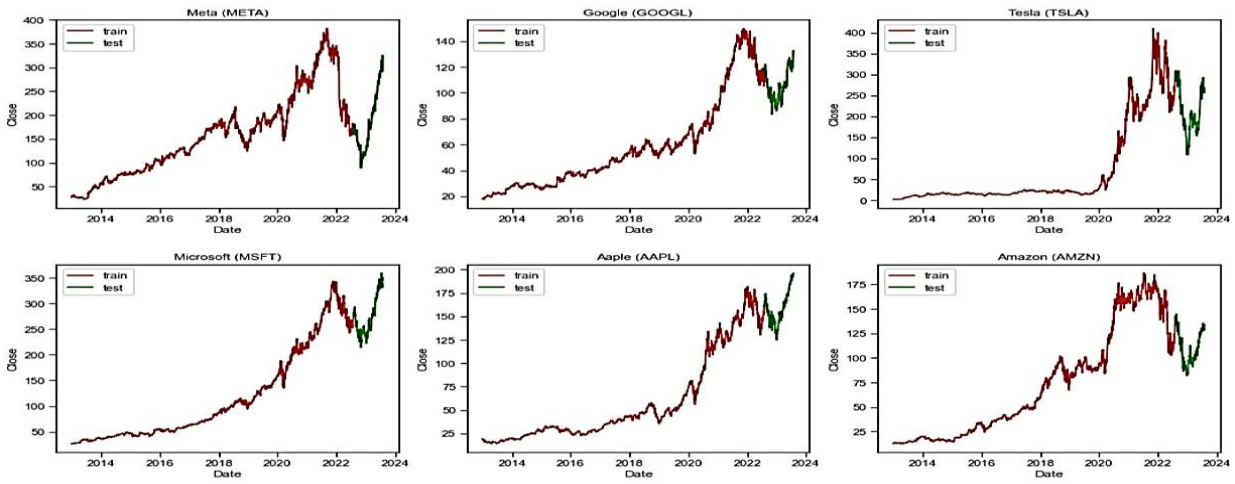

Fig. 10: Stock prices, the training set (red color): January 1, 2013 to August 1, 2022 and trading set (green color): from July 1, 2022 to August 1, 2023- From top left to bottom right: META, GOOGL, TSLA, MSFT, AAPL and AMZN.

Profit and Loss (P&L): presents the amount of profit or loss of the algorithm in the desired time period.

Reinforcement learning algorithms are very sensitive to hyperparameter values, and one of the most time-consuming processes of reinforcement learning is the optimization phase of hyperparameters. To optimize the parameters, we must define a search space in which the valid values of the hyperparameters are specified. Values can be sampled in two ways: normal distribution and uniform distribution.

The selection of hyperparameters can be done both randomly and in a grid manner. We perform Bayesian optimization algorithm for search and use Sharpe ratio as risk adjusted return measure for hyperparameter optimization in validation phase. Fig. 11 shows the effect of the number of episodes on the convergence of the reinforcement agent in the training phase. The convergence of the algorithm is clearly seen in episode 10,000, but in episode 500, the output of the agent's reward fluctuates a lot.

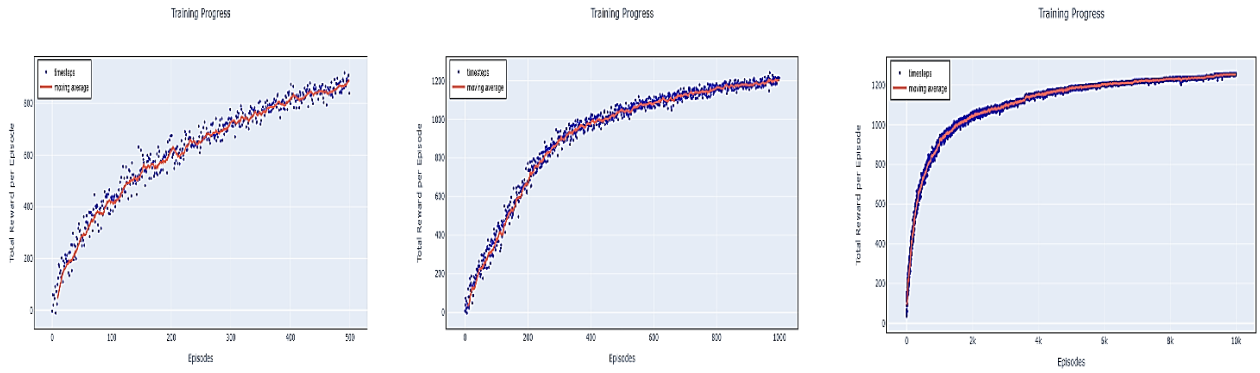

Fig. 11: Effect of the number of episodes on the convergence of the reinforcement agents in the training phase (Left to right: 500, 1000 and 10,000 episodes).

Results and Discussion

A. Results of the Proposed State Formulation

To evaluate the impact of proposed state formation on the convergence speed and the returned reward, the experiment setup involving three agents with random stocks is implemented.

The result as shown in Fig. 12 confirms that the proposed state construction process significantly converges to higher rewards in less time.

Fig. 12: Training reward using proposed difference vector lookback window (dvlw) and simple lookback window(slw), w=32.

B. Results of the Risk Aversion Reward Function

Fig. 13 illustrates the trading actions and outcomes achieved through the utilization of the proposed risk aversion reward function, specifically concerning the concept of Maximum DrawDown. By maintaining a predetermined cash reserve level and implementing a trading reward function that penalizes the RL agent when the cash level falls below a specified threshold, the potential for enhancing the Maximum DrawDown metric becomes apparent. Maximum DrawDown (MDD) represents a widely used risk assessment tool within the realms of trading and investment. It gauges the utmost loss suffered by an investment or trading strategy from its peak to its lowest point before reaching a new peak (ovals in Fig. 13(a) In this context, smoothed increasing gained reward (Fig. 13(b)) versus fluctuated reward curve (Fig. 13(a)) demonstrates that by imposing penalties on the RL agent for sustaining a cash level beneath a designated threshold, a proactive encouragement is established for the agent to uphold a more substantial cash balance. Furthermore, the findings in Fig. 13(c) validate the results of the trade action distribution chart, showcasing a reduced number of long positions and an increased occurrence of short positions, accompanied by instances of holding flat positions. This configuration can be interpreted as a strategy for managing risks. This approach contributes to the mitigation of drawdown severity by deterring the agent from assuming overly risky positions that might otherwise lead to substantial losses. The tabular CR results of training using risk aversion cash penalty (RAPW) and no limit on cash (NLOC) is given in Table 4.

(a) (b) (c)

Fig. 13: a) Trading simulation by no limit on cash reserving and high volatile return. b) Trading simulation by using risk aversion cash penalty. c) Action distribution of RAPW of stocks.

Table 4: CR of stock in training process using risk aversion cash penalty (RAPW) and no limit on cash (NLOC)

Stock / Reward function	MSFT	AAPL	GOOGL	AMZN	META	TSLA
RAPW	2193	997	1063	1788	2755	1762
NLOC	2030	976	950	1487	2402	1695

C. Results of the Multiagent vs. Single agent

This experiment kicked off with the development of a comprehensive set of trading strategies for both the multiagent and single agent systems, taking into account various technical, fundamental, and sentiment-based factors. Prior to launching the experiment, extensive backtesting was carried out to fine-tune the parameters of each trading system, ensuring optimal strategy execution and reducing the potential for overfitting. Fig. 14 shows the result of this experiment in terms of gained total reward per episode.

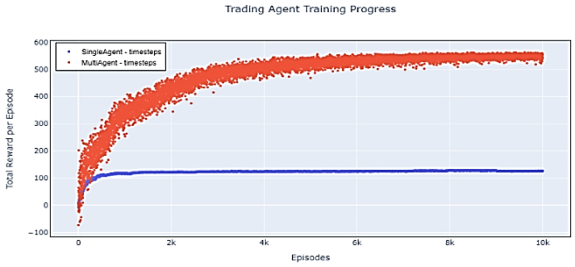

Fig. 14: Total reward per Episode of proposed MultiAgent system and single agent trading system, w=64.

In summary, the empirical evidence provides compelling support for the superiority of the multiagent trading system over the single-agent system in terms of returns. Through dedicating additional time to the model convergence process and refining policies, more favorable outcomes can be realized. This undertaking will not only enhance the model quality and precision, but also markedly elevate final efficiency. Thus, the accurate fusion of patience and focused parameter adjustments and policy optimizations has ultimately culminated in attaining superior performance and greater value compared to the invested time and efforts.

D. Results of the Robustness

To thoroughly challenge the CMSRS adaptability and robustness, the experiment ventured into uncharted territory by subjecting it to non-trained stock [IBM]. Analysis of trading performance metrics such as Sharpe ratio (SR), maximum drawdown (MDD), and average trade profit and loss (P&L) is shown in Table 5. The initial investment has set to 10,000$.

Table 5: Trading performance of proposed MultiAgent system on non-trained IBM stock

Period	January 1, 2013 - July 1, 2023		August 1, 2022 - August 1, 2023	
Measures	Proposed model	Buy and Hold	Proposed model	Buy and Hold
SR	0.1276	-1.3430	0.0057	-1.5821
MDD	-0.3592	-0.4372	-0.101	-0.1761
P&L	26456	-38.58	2512.04	1345.44

E. Comparison with Baselines

In this section, we provide the performance comparison results of our proposed CMSRS against other approaches during both the training and backtesting phases.

We compare our CMSRS against Buy and Hold baseline. Besides that, we employ state of the art Multi Agent DQN (MADQN) RL algorithm (Table 6).

Table 6: CMSRS backtesting results

Stock	Period	January 1, 2013 - July 1, 2023			August 1, 2022 - August 1, 2023		
	Measures	CMSRS	MADQN	B&H	CMSRS	MADQN	B&H
META	SR	**0.4325**	0.087	-0.8224	**0.0326**	0.0076	-0.4849
	MDD	**-0.24**	-0.432	-0.5922	**0.05831**	-0.311	-0.5085
	P&L	**53754.3**	50213	46798.70	**15467**	11098	10179.76
GOOG	SR	**0.3708**	0.0097	-1.1431	**0.05831**	0.0102	-0.8647
	MDD	**-0.21**	-0.298	-0.3087	**-0.09**	-0.203	-0.3166
	P&L	**73217.9**	57605.9	54208.64	**7521**	2314	1534.46
TSLA	SR	**0.1570**	0.0145	-0.4834	**0.1203**	0.0021	-0.5223
	MDD	**-0.3203**	-0.4509	-0.6063	**-0.34**	-0.43	-0.6505
	P&L	**2134012**	1348905	1250510	**3210**	23	-1207.48
MSFT	SR	**0.3773**	0.05	-1.1231	**0.0778**	0.0101	-0.9483
	MDD	**-0.1983**	-0.2809	-0.2908	**-0.12**	-0.311	-0.2684
	P&L	**123709**	113900	112621	**7525**	3715	2181.32
AAPL	SR	**0.3597**	0.1348	-1.0466	**0.09381**	0.0176	-1.0458
	MDD	**-0.2521**	-0.3	-0.3852	**-0.092**	-0.211	-0.2826
	P&L	**97654**	91212	85957.96	**4614.9**	1441	2141.46
AMZN	SR	**0.3464**	0.09	-0.9348	**0.06134**	0.0076	-0.7986
	MDD	**-0.2709**	-0.398	-0.4516	**-0.1689**	-0.271	-0.4349
	P&L	**108306**	97201	94856.15	**3251.65**	1258	-348.67

Fig. 15 shows learning curve of the proposed multiagent RL and baseline multi DQN. As evident from Fig. 15, the learning process of DQN exhibits notable variance. In contrast, the presented model not only outperforms DQN in terms of achieving superior rewards but also excels in learning speed and training efficiency, requiring significantly less time-approximately one-tenth of the time invested by DQN.

Fig. 15: Learning curve of the proposed multiagent RL and baseline multi DQN.

Conclusion

This study has introduced a significant advancement in the realm of stock recommender systems through the development of a Concurrent Multiagent Deep Reinforcement Learning-based Stock Recommender System (CMSRS).

While previous systems focused on a limited number of sequential trading agents within the same environment, often leading to errors in volatile market conditions, the proposed CMSRS represents a robust solution by leveraging concurrent multi-layer architecture. The CMSRS framework is designed with meticulous consideration, encompassing feature extraction in the data layer to construct diverse trading environments.

This innovative approach enables multiple feed Deep Reinforcement Learning (DRL) agents to make recommendations robustly within the trading layer. The system effectively integrates various data sources, incorporating Google stock trends, fundamental data, technical indicators, and historical price data. This comprehensive dataset empowers the concurrent agents to collaboratively select and recommend stocks for buying or selling.

To further enhance the effectiveness of the system, the Sharpe ratio is employed as a risk-adjusted return measure, facilitating the optimization of hyperparameters during the validation phase. Additionally, the introduced reward function ensures dynamic management of cash reserves, thereby addressing liquidity requirements and penalizing deviations from maintaining an adequate cash reserve. Empirical results obtained from real U.S. stock market data corroborate the supremacy of the Concurrent Multiagent SRS (CMSRS), particularly evident in volatile market conditions and out-of-sample scenarios. The CMSRS not only demonstrates its ability to navigate challenging market dynamics but also exhibits robustness and superior performance in comparison to prior systems. By advancing the capabilities of stock recommender systems in the domain of deep reinforcement learning, this research contributes significantly to the field of financial technology and investment strategies.

Author Contributions

The authors declare no potential conflict of interest regarding the publication of this work. In addition, the ethical issues including plagiarism, informed consent, misconduct, data fabrication and, or falsification, double publication and, or submission, and redundancy have been completely witnessed by the authors.

Conflict of Interest

The authors declare no potential conflict of interest regarding the publication of this work. In addition, the ethical issues including plagiarism, informed consent, misconduct, data fabrication and, or falsification, double publication and, or submission, and redundancy have been completely witnessed by the authors.

References

[1] M. Z. Asghar, F. Rahman, F. M. Kundi, S. Ahmad, "Development of stock market trend prediction system using multiple regression," Comput. Math. Organ. Theory, 25(2019): 271-301, 2019.

[2] A. A. Ariyo, A. O. Adewumi, C. K. Ayo, "Stock price prediction using the ARIMA model," in Proc. 2014 UKSim-AMSS 16th International Conference on Computer Modelling and Simulation: 106-112, 2014.

[3] Y. Wang, Y. Liu, M. Wang, R. Liu, "LSTM model optimization on stock price forecasting," in Proc. 2018 17th International Symposium on Distributed Computing and Applications for Business Engineering and Science (DCABES): 173-177, 2018.

[4] S. Banik, N. Sharma, M. Mangla, S. N. Mohanty, S. Shitharth, "LSTM based decision support system for swing trading in stock market," Knowl. Based Syst., 239: 107994, 2022.

[5] S. Selvin, R. Vinayakumar, E. A. Gopalakrishnan, V. K. Menon, K. P. Soman, "Stock price prediction using LSTM, RNN and CNN-sliding window model," in Proc. 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI): 1643-1647, 2017.

[6] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness et al., "Human-level control through deep reinforcement learning," Nature, 518(7540): 529-533, 2015.

[7] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifreet al., "Mastering the game of Go with deep neural networks and tree search," Nature, 529 (7587): 484-489, 2016.

[8] A. R. Azhikodan, A. G. Bhat, M. V. Jadhav, "Stock trading bot using deep reinforcement learning," in Innovations in Computer Science and Engineering, Proc. the Fifth ICICSE 2017: 41-49, Springer, 2019.

[9] X. Wu, H. Chen, J. Wang, L. Troiano, V. Loia, H. Fujita, " Adaptive stock trading strategies with deep reinforcement learning methods," Inf. Sci., 538: 142-158, 2020.

[10] S. Carta, A. Corriga, A. Ferreira, A. S. Podda, D. R. Recupero, "A multi-layer and multi-ensemble stock trader using deep learning and deep reinforcement learning," Appl. Intell., 51: 889-905, 2021.

[11] X. Y. Liu, H. Yang, Q. Chen, R. Zhang, L. Yang, B. Xiao, C. D. Wang, "FinRL: A deep reinforcement learning library for automated stock trading in quantitative finance," arXiv preprint arXiv:2011.09607, 2020.

[12] S. Yang, "Deep reinforcement learning for portfolio management," Knowl. Based Syst., 278: 110905, 2023.

[13] C. Ma, J. Zhang, Z. Li, S. Xu, "Multi-agent deep reinforcement learning algorithm with trend consistency regularization for portfolio management," Neural Comput. Appl., 35(9): 6589-6601, 2023.

[14] Z. Huang, F. Tanaka, "MSPM: A modularized and scalable multi-agent reinforcement learning-based system for financial portfolio management," Plos one, 17(2): e0263689, 2022.

[15] J. Lussange, I. Lazarevich, S. Bourgeois-Gironde, S. Palminteri, B. Gutkin, "Modelling stock markets by multi-agent reinforcement learning," Comput. Econ., 57: 113-147, 2021.

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

[17] P. Koratamaddi, K. Wadhwani, M. Gupta, D. S. G. Sanjeevi, "A multi-agent reinforcement learning approach for stock portfolio allocation," in Proc. the 3rd ACM India Joint International Conference on Data Science & Management of Data (8th ACM IKDD CODS & 26th COMAD): 410-410, 2021.

[18] D. Kwak, S. Choi, W. Chang, "Self-attention based deep direct recurrent reinforcement learning with hybrid loss for trading signal generation," Inf. Sci., 623: 592-606, 2023.

[19] S. Forouzandeh, K. Berahmand, R. Sheikhpour, Y. Li, "A new method for recommendation based on embedding spectral clustering in heterogeneous networks (RESCHet)," Expert Syst. Appl., 231: 120699, 2023.

[20] S. Forouzandeh, M. Rostami, K. Berahmand, R. Sheikhpour, "Health-aware food recommendation system with dual attention in heterogeneous graphs," Comput. Biol. Med., 169: 107882, 2024.

[21] M. Nourahmadi, A. Rahimi, H. Sadeqi, "Designing a stock recommender system using the collaborative filtering algorithm for the Tehran stock exchange," Financ. Res. J., 26(2): 302-330, 2024.

[22] B. Yang, T. Liang, J. Xiong, C. Zhong, "Deep reinforcement learning based on transformer and U-Net framework for stock trading," Knowl. Based Syst., 262: 110211, 2023.

[23] J. Zou, J. Lou, B. Wang, S. Liu, "A novel deep reinforcement learning based automated stock trading system using cascaded lstm networks," Expert Syst. Appl., 242: 122801, 2024.

[24] F. F. He, C. T. Chen, S. H. Huang, "A multi-agent virtual market model for generalization in reinforcement learning based trading strategies," Appl. Soft Comput., 134, 109985, 2023.

[25] S. Singh, V. Goyal, S. Goel, H. C. Taneja, "Deep reinforcement learning models for automated stock trading," in Advanced Production and Industrial Engineering, 27: 175, 2022.

[26] M. Taghian, A. Asadi, R. Safabakhsh, "Learning financial asset-specific trading rules via deep reinforcement learning," Expert Syst. Appl., 195: 116523, 2022.

[27] L. K. Felizardo, F. C. L. Paiva, C. de Vita Graves, E. Y. Matsumoto, A. H. R. Costa, E. Del-Moral-Hernandez, P. Brandimarte, "Outperforming algorithmic trading reinforcement learning systems: A supervised approach to the cryptocurrency market," Expert Syst. Appl., 202: 117259, 2022.

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

[29] T. Faturohman, T. Nugraha, "Islamic stock portfolio optimization using deep reinforcement learning," J. Islamic Monetary Econ. Finance, 8(2): 181-200, 2022.

[30] H. Yue, J. Liu, D. Tian, Q. Zhang, "A novel anti-risk method for portfolio trading using deep reinforcement learning," Electronics, 11(9): 1506, 2022.

[31] A. Nair, P. Srinivasan, S. Blackwell, C. Alcicek, R. Fearon, A. De Maria et al., "Massively parallel methods for deep reinforcement learning," arXiv preprint arXiv:1507.04296, 2015.

[32] Y. Shoham, K. Leyton-Brown, "Multiagent systems: Algorithmic, game-theoretic, and logical foundations," Cambridge University Press, 2008.

[33] R. Lowe, Y. I. Wu, A. Tamar, J. Harb, O. Pieter Abbeel, I. Mordatch, "Multi-agent actor-critic for mixed cooperative-ompetitive environments," Adv. Neural Inf. Processing Syst., 30, 2017.

[34] S. Khonsha, M. A. Sarram, R. Sheikhpour, "A profitable portfolio allocation strategy based on money net-flow adjusted deep reinforcement learning," Iran. J. Finance, 7(4): 59-89, 2023.

[35] H. Hu, L. Tang, S. Zhang, H. Wang, "Predicting the direction of stock markets using optimized neural networks with Google Trends," Neurocomput., 285: 188-195, 2018.

[36] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, "Proximal policy optimization algorithms," arXiv preprint arXiv:1707.06347, 2018.

Biographies

Samira Khonsha is a faculty member of the department of computer engineering, Zarghan branch, Islamic Azad University. She holds a Bachelor's degree in Software Engineering from the Shiraz University. She also has a Master's degree in in Software Engineering from Shiraz University and a Ph.D. in Software Engineering from the Yazd University. Her areas of expertise include reinforcement learning, and financial markets.

- Email: khonsha.samira@gmail.com
- ORCID: 0000-0001-9301-760X
- Web of Science Researcher ID: NA
- Scopus Author ID: NA
- Homepage: https://scholar.google.com/citations?user=4wEfZaAAAAAJ&hl=en&oi=ao

Mehdi Agha Sarram is an Associate Professor at the department of Computer Engineering in Yazd University, Yazd, Iran. He received his Ph.D. degree from University of Wales, Cardiff, U.K. in 1979. He is Member of Australian Institute of Control and Instrumentation and also Member of Steering Committee on IT standards (ISIRI-ITTC). He has been Casual Lecturer in Australian Universities such as SIBT Macquarie University, University of Western Sydney Macarthur and SWIC University of Western Sydney from 2000 to 2003. His research interests include Machine learning, Data mining, Network coding and Wireless sensor networks.

- Email: mehdi.sarram@yazd.ac.ir
- ORCID: 0000-0002-1872-6155
- Web of Science Researcher ID: NA
- Scopus Author ID: NA
- Homepage: https://scholar.google.com/citations?user=Hx1_SDYAAAAJ&hl=en

Razieh Sheikhpour received her Ph.D. in Computer Engineering from Yazd University, Yazd, Iran, in 2017. Currently, she is an Associate Professor at the department of Computer Engineering at Ardakan University, Ardakan, Iran. Her research interests include machine learning, semi-supervised feature selection and bioinformatics.

- Email: rsheikhpour@ardakan.ac.ir
- ORCID: 0000-0002-3119-3349
- Web of Science Researcher ID: N-3816-2017
- Scopus Author ID: 55321804800
- Homepage: https://scholar.google.com/citations?user=SyIdF_4AAAAJ&hl=en

How to cite this paper:
S. Khonsha, M. A. Sarram, R. Sheikhpour, "A robust concurrent multi-agent deep reinforcement learning based stock recommender system," J. Electr. Comput. Eng. Innovations, 13(1): 225-240, 2025.

DOI: 10.22061/jecei.2024.11193.775

URL: https://jecei.sru.ac.ir/article_2229.html