Deep Reinforcement Learning for Portfolio Management: An Extended EIIE Framework with PPO and Technical Indicators

Minh Thuan Nguyen

Faculty of Computer Science and Engineering
Vietnamese-German University
10421057@student.vgu.edu.vn

Dang Nhat Duong Le

Faculty of Computer Science and Engineering
Vietnamese-German University
10421011@student.vgu.edu.vn

***Abstract*— This electronic document is a "live" template. The various components of your paper [title, text, heads, etc.] are already defined on the style sheet, as illustrated by the portions given in this document.**

I. INTRODUCTION

Financial markets exhibit complex, nonlinear behaviors influenced by various economic, political, and psychological factors. Traditional trading strategies, including rule-based approaches and statistical models, often struggle to adapt to dynamic market conditions. With the advancement of deep learning techniques, neural networks have demonstrated significant potential in financial modeling due to their ability to capture intricate patterns and dependencies within time-series data [1].

In recent years, deep reinforcement learning has emerged as a promising approach for algorithmic trading. Unlike supervised learning, which relies on historical labels, reinforcement learning enables an agent to optimize trading strategies through interaction with the market environment [2]. Moreover, the integration of Convolutional Neural Networks (CNNs) has further enhanced trading models by effectively capturing spatial and temporal dependencies in financial data [3].

This study implements a neural network-based trading model designed to optimize portfolio allocation through adaptive learning. By leveraging deep learning architectures and reinforcement learning techniques, the model dynamically adjusts portfolio weights based on market conditions. The proposed approach aims to outperform traditional benchmark strategies by learning optimal trading actions through continuous market interactions. [4, 5]

II. RELATED WORK

Early approaches in financial modeling relied on supervised learning techniques for stock price prediction [6, 7]. Traditional methods trained neural networks to forecast asset prices based on historical data. However, these approaches lacked the ability to make dynamic trading decisions, as they only focused on prediction without integrating trading strategies.

Reinforcement Learning (RL) has been applied to financial portfolio management for several decades [8, 9]. The application of Q-learning and policy gradient methods enabled portfolio optimization by directly learning trading strategies. For instance, direct recurrent policy learning was proposed to handle non-Markovian financial environments [10]. However, these methods struggled with high-dimensional state spaces and failed to generalize well in complex financial markets.

The integration of deep learning and reinforcement learning has led to significant improvements in financial decision-making. Deep RL frameworks, such as Deep Q-Networks (DQN) and Actor-Critic methods, have been explored for asset allocation . These approaches leverage deep neural networks to approximate value functions and policy functions, improving decision-making in financial markets. [5, 11]

Recent studies have also employed deep recurrent neural networks (RNNs) such as Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRUs) to address the challenges of partially observable financial environments [12]. Moreover, risk-adjusted reward functions, such as the Sharpe ratio, have been incorporated to better optimize trading strategies.

III. PROBLEM STATEMENT

Financial portfolio management involves the continuous reallocation of capital among multiple financial assets with the objective of maximizing returns while controlling risk and minimizing transaction costs. In dynamic financial markets, investors face the challenge of making real-time decisions based on incomplete information, where historical price data is assumed to encapsulate all relevant market signals [5]. The problem is further complicated by the non-stationary nature of financial time series, high transaction costs, and the inherent risk associated with market volatility.

A. Trading Period

In this study, we define a trading period as a fixed interval at which the reinforcement learning agent is allowed to adjust its portfolio allocation. Unlike Jiang et al. [5], where the trading period is set at 30-minute intervals, we adopt a 60-minute interval to better capture market trends while reducing the frequency of trades, thereby mitigating transaction costs.

For each trading period t, the state representation provided to the agent consists of historical price movements and a set of technical indicators. The price data includes the four fundamental price points: open, high, low, and close prices

of all assets in the portfolio. However, beyond these raw price features, we extend the input space by incorporating additional financial indicators widely used in technical analysis, such as: Simple Moving Average (SMA), Parabolic Stop and Reverse (PSAR), Trading Volume and many more. By integrating these additional indicators, we aim to enhance the model's ability to capture meaningful market dynamics beyond raw price fluctuations. [3, 1, 2]

B. Transaction Cost

In real-world financial markets, executing buy and sell orders is not free and is subject to transaction costs, typically in the form of brokerage fees, exchange fees, or slippage. These costs can significantly impact net portfolio returns, especially in high-frequency trading strategies where frequent rebalancing occurs. To account for these expenses, we incorporate a transaction cost penalty into the reinforcement learning framework. [5]

Following a simplified yet realistic approach, we assume a constant commission rate λ on every trade executed by the agent. Given a portfolio reallocation at time t, the transaction cost is computed based on the absolute difference between the previous asset holdings and the new allocations. The transaction cost at each step is formulated as:

$$C_t = \lambda \sum_{i=1}^{n} |w_{i,t} - w_{i,t-1}| \, P_{i,t}, \tag{1}$$

where:

- λ is the fixed transaction fee per trade, set to **0.001** (equivalent to a **0.1%** commission fee).
- $w_{i,t}$ represents the agent's portfolio weight allocation for asset i at time t.
- $w_{i,t-1}$ represents the previous portfolio allocation for asset i at time $t-1$.
- $P_{i,t}$ is the market price of asset i at time t.

The transaction cost C_t is deducted from the portfolio's net worth at each trading step.

C. Mathematical Formalism

The mathematical formulation of the portfolio management problem in this study builds upon the reinforcement learning framework introduced by Jiang et al. [5]. While we adopt their mathematically formulation, we revise key aspects of the state representation, reward function, and portfolio update mechanism to incorporate additional financial indicators and account for transaction costs more explicitly.

IV. DATA TREATMENT

The data used for training and testing our trading agent are collected from Trading View, a trading platform that tracks all of the popular assets in trading (Future, stocks, forex, crypto, ...). We choose this platform because it offer a large volume of data dated back serveral years. [13]

We choose 4 different stocks data as assets, which are Pepsi (PEP), Costco (COST), Apple (APPL) and Citigroup Inc. (C). In each of the stocks, we include with undamental price points such as high, low, open and close. As stated before, also add some indicators such as

- **Parobolic Stop and Reverse (PSAR)**: is a trading indicator used in technical analysis to identify trend direction and potential reversals. [14]
- **Simple Moving Average (SMA)**: is an arithmetic moving average calculated by adding recent prices and then dividing that figure by the number of time periods in the calculation average. [15]
- **Relative Strength Index (RSI)**: is a momentum indicator used in technical analysis. It measures the speed and magnitude of a security's recent price changes to detect overbought or oversold conditions in the price of that security. [16]
- **Volume**: is used in trading and investing to measure the liquidity and market activity. It provides insights into buying and selling pressure by measuring the number of shares or contracts traded during a specified period.
- **Bollinger Band (BBH and BBL)**: is a technical analysis tool that highlights how prices are dispersed around an average value. It is composed of an upper band, a lower band, and a middle moving average line. The upper and lower bands expand when market volatility is high, and contract when market volatility is low. [17]
- **Bollinger Band Width (BBW)**: is the distance between the two bands that make up the price channel defined Bollinger Bands. [17]

A. Data preprocessing

The data collected from Trading View is complete and of high quality, eliminating the need for missing value imputation.

To ensure consistency and comparability, the dataset was normalized using Z-score standardization, also known as standardization or zero-mean normalization. It is a widely used data preprocessing technique that transforms numerical variables into a standardized scale with a mean of zero and a standard deviation of one. This method ensures that all features have comparable magnitudes, making it particularly suitable for machine learning models that assume normally distributed data. [18]

The standardization process consists of the following steps:

- **Compute the Mean:** Calculate the mean μ of the feature to determine the central tendency of the data.
- **Compute the Standard Deviation:** Determine the standard deviation σ of the feature, which measures the dispersion of values around the mean.
- **Apply the Standardization Formula:** Subtract the mean from each data point and divide by the standard deviation to obtain the standardized value. This is represented mathematically as:

$$X' = \frac{X - \mu}{\sigma} \tag{2}$$

where X is the original feature value, μ is the mean, and σ is the standard deviation of the feature. [19]

V. METHODOLOGY

A. Environment Design

Financial markets are inherently complex, with vast amounts of information influencing asset prices at any given moment. It is impossible for an agent to obtain a complete and perfect representation of the market state due to the sheer scale of market interactions and external economic factors. However, in line with the philosophy of technical traders [20, 21], all relevant market information is believed to be inherently reflected in asset prices. Consequently, the agent relies solely on **historical price movements** and market-derived indicators to infer trends and make trading decisions.

From this perspective, the **environmental state** can be approximated by the historical price data of all assets up to the current moment. While full order book histories are publicly available in many financial markets, the vast volume of trade-level data makes it impractical for the agent to process them in real time. Instead, a **sub-sampling strategy** is employed, where the agent receives a fixed-size **look-back window** of past market observations. This approach condenses the vast market information into a manageable format while preserving the key patterns needed for decision-making. [22, 23]

The **state representation** at any timestep t, denoted as S_t, is defined as follows:

- **Historical Price Data**: A time-series of closing prices for each asset, capturing short- and long-term market trends.
- **Portfolio Allocation**: The weight distribution of the agent's capital across different assets at the previous timestep, providing context for decision-making.
- **Net Worth**: The total value of the portfolio at the current timestep, serving as a performance metric.

Mathematically, the **state vector** is given by:

$$S_t = \{X_t, w_{t-1}, NW_t\} \tag{3}$$

where:

- X_t represents the **historical price tensor**, encoding past market movements.
- w_{t-1} is the **portfolio weight vector** from the previous step.
- NW_t is the **portfolio's net worth** at time t.

By restricting the state space to **price-derived information**, the agent operates in a **financial-model-free environment**, similar to the approach used in Jiang et al. (2017) [5]. This allows the model to focus on learning profitable patterns without requiring explicit economic or fundamental data inputs.

B. Reward Function

The reward function plays a crucial role in guiding the agent's learning process in reinforcement learning. In this study, the reward function is designed to incentivize long-term portfolio growth while taking into account transaction costs and risk. Inspired by Jiang et al. (2017) [5], we adopt the logarithmic portfolio return as the primary reward signal. The reward at time t is defined as:

$$R_t = \log\left(\frac{NW_t}{NW_{t-1}}\right), \tag{4}$$

where NW_t is the net worth of the portfolio at time t and NW_{t-1} is the net worth at the previous timestep. This logarithmic transformation measures relative growth, provides a smooth learning signal, and naturally aligns with the goal of compounding portfolio returns over time.

$$R_t = \log\left(\frac{NW_t}{NW_{t-1}}\right) - C_t. \tag{5}$$

Furthermore, to encourage stable trading strategies, a risk-adjusted reward based on the Sharpe ratio can be employed:

$$R_t = \frac{E[R_t]}{\sigma_t}, \tag{6}$$

where $E[R_t]$ is the expected return and σ_t is the standard deviation of the portfolio's returns, representing risk. This formulation ensures that the agent prioritizes high risk-adjusted returns over strategies that yield high returns at the cost of excessive volatility.

In summary, the reward function used in this study combines the logarithmic portfolio return with a transaction cost penalty, and can be adjusted to incorporate risk considerations via the Sharpe ratio. This multi-objective reward framework encourages the agent to develop a profitable, stable, and cost-efficient trading strategy that aligns with real-world financial principles. [8, 24]

C. Reinforcement Learning Trading Agent

The reinforcement learning trading agent developed in this study is structured within a deep convolutional neural network (CNN) architecture, specifically designed to optimize portfolio management through adaptive learning. The model follows the **Ensemble of Identical Independent Evaluators (EIIE)** framework, originally proposed by Jiang et al. (2017), which leverages deep learning techniques to process financial time-series data and dynamically adjust asset allocations. This architecture is particularly suited for financial applications due to its ability to extract meaningful patterns from market data while maintaining memory of past portfolio allocations. [5]

Unlike conventional deep reinforcement learning models that rely on fully connected or recurrent network structures, the EIIE model employs a **2D convolutional neural network** to process historical market data. CNNs are particularly effective for financial applications due to their ability to capture both spatial and temporal dependencies in time-series data . The EIIE architecture consists of the following layers: [25]

- **First Convolutional Layer**: A 2D convolution with **2 filters** of size 3×1, followed by a **ReLU activation function**. This layer extracts short-term dependencies in asset price fluctuations.

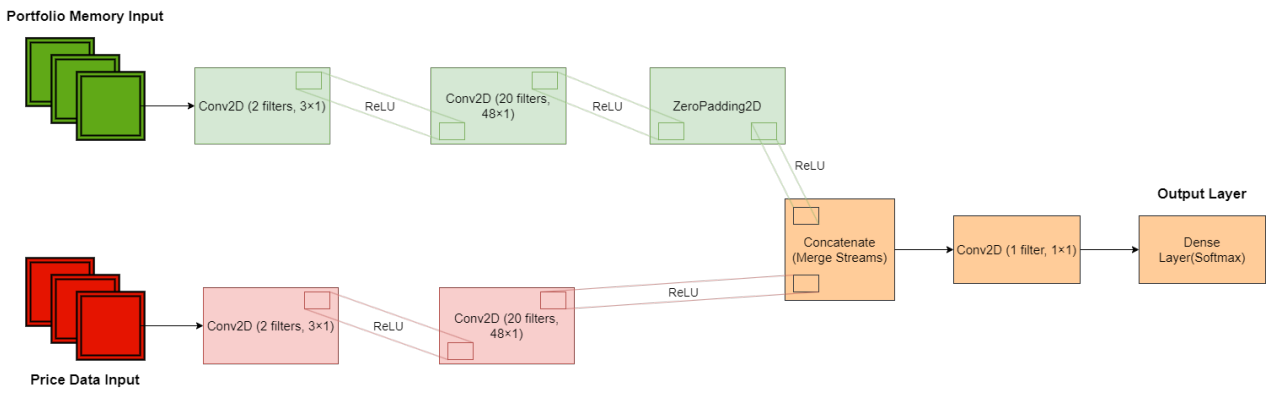

Fig. 1. EIIE-based CNN architecture.

- **Second Convolutional Layer**: Employs **20 filters** with a kernel size of 48×1, capturing long-range temporal correlations across assets.
- **Zero-Padding Layer**: Standardizes feature dimensions by padding the extracted feature maps.
- **Final Convolutional Layer**: Utilizes a **1-filter** kernel of size 1×1 to aggregate feature representations and prepare the output for portfolio decision-making.

The CNN processes input market data in the form of a structured **price tensor**, where each dimension represents historical prices of multiple assets over a defined look-back period. By employing a hierarchical feature extraction process, the network effectively learns complex patterns in financial data, improving its ability to identify profitable asset allocation strategies. [26]

A key innovation of the EIIE model is its incorporation of a **portfolio memory mechanism**, which facilitates **temporal consistency in asset allocation**. Unlike traditional reinforcement learning models, where portfolio decisions are independently computed at each timestep, the EIIE framework retains and updates past allocations, ensuring smoother transitions between successive rebalancing operations. This is implemented by incorporating a secondary input into the CNN, which stores the previous portfolio allocation vector. [18, 27]

At time t, the network receives two inputs:

$$\text{Input}_1 = X_t, \quad \text{Input}_2 = w_{t-1}, \tag{7}$$

where:

- X_t represents the market data tensor at time t, encoding historical price trends.
- w_{t-1} denotes the portfolio weight vector from the previous timestep.

The model then updates the portfolio allocation by computing:

$$w_t = \text{softmax}(F_t + w_{t-1}), \tag{8}$$

where:

- F_t is the CNN-extracted feature representation of the current market state.
- The **softmax function** ensures that portfolio allocations remain normalized, satisfying:

$$\sum_{i=1}^{n} w_{i,t} = 1, \quad w_{i,t} \geq 0. \tag{9}$$

This mechanism enables the agent to **persist previous investment decisions**, mitigating unnecessary portfolio turnover and reducing transaction costs. By maintaining continuity in asset allocations, the model improves capital efficiency and reduces volatility in trading strategies.

D. Proximal Policy Optimization

Proximal Policy Optimization (PPO) is a reinforcement learning algorithm that improves training stability and efficiency by restricting policy updates within a bounded region. Introduced by Schulman et al. (2017) [28], PPO is widely used in deep reinforcement learning due to its balance between performance and computational efficiency. In this study, PPO is applied to optimize the trading policy of the reinforcement learning agent, ensuring stable learning and robust decision-making.

In the implemented trading agent, PPO is used to update the policy network (actor) and the value function network (critic). The actor network predicts the optimal portfolio allocation at each timestep, while the critic network estimates the value function to guide policy updates. The objective of PPO is to maximize the expected advantage while ensuring that policy updates do not deviate excessively from the previous policy. The clipped surrogate objective function used in PPO is defined as:

$$L^{CLIP}(\theta) = E\left[\min\left(r_t A_t, \text{clip}(r_t, 1-\epsilon, 1+\epsilon)A_t\right)\right] - \lambda H(\pi), \tag{10}$$

where:

- r_t is the probability ratio between the new and old policy, defined as:

$$r_t = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \tag{11}$$

- A_t is the advantage estimate, representing the expected improvement of the action a_t.

- ϵ is the clipping parameter that prevents excessively large policy updates.
- $H(\pi)$ is the entropy of the policy, added to encourage exploration.

In our implementation, the actor network is trained to minimize this objective function using the Adam optimizer, ensuring stable policy updates. The critic network is updated using the mean squared error (MSE) loss function:

$$L^{Critic}(\theta) = E\left[(V_\theta(s_t) - V^{\text{target}}(s_t))^2\right], \tag{12}$$

where $V_\theta(s_t)$ is the estimated value function and $V^{\text{target}}(s_t)$ is the target value obtained from bootstrapping future rewards.

The application of PPO in this trading framework provides several key benefits. The clipping mechanism prevents large, destabilizing updates to the policy network, ensuring more stable training. Additionally, the entropy term encourages efficient exploration, preventing premature convergence to suboptimal policies. By leveraging the critic network's value estimation, PPO enables the agent to make more informed trading decisions, leading to improved long-term portfolio growth. Furthermore, PPO offers superior computational efficiency compared to Trust Region Policy Optimization (TRPO), achieving similar performance while reducing computational overhead, making it well-suited for large-scale financial data. Overall, PPO facilitates stable and controlled policy updates, enhancing training efficiency and improving trading performance in dynamic financial markets. [2, 28]

E. Training Scheme and Testing Procedure

The proposed reinforcement learning framework follows a structured training and testing pipeline. The training process involves an episodic learning paradigm where the agent interacts with a simulated market environment, refines its policy using Proximal Policy Optimization (PPO), and iteratively improves portfolio allocation strategies. The testing phase evaluates the learned policy on unseen market data to assess generalization performance.

Training Phase: The training phase consists of multiple episodes, where each episode represents a sequence of trading periods within a predefined market history. The agent starts with an initial portfolio allocation and iteratively adjusts asset weights based on the observed state. The training procedure consists of the following steps:

1) **Environment Initialization:** At the beginning of each episode, the environment stochastically selects a starting index within the historical dataset. The number of steps loaded into the episode is equal to the training batch size, ensuring that the agent learns from varying subsequences of market data. The agent starts with an initial capital allocation.
2) **State Observation:** At each trading step t, the agent receives a state vector s_t consisting of historical price movements, technical indicators, and the previous portfolio allocation w_{t-1}.
3) **Action Selection:** Using the actor-network, the agent generates a probability distribution over possible portfolio allocations. The portfolio weight vector w_t is sampled from this distribution.
4) **Portfolio Update and Execution:** The selected action is executed, and new asset holdings are computed. The net portfolio value is updated based on market price changes and transaction costs.
5) **Reward Calculation:** The reward at each step is computed as the logarithmic return of the portfolio, penalized by transaction costs follow ...
6) **Experience Replay and Policy Update:** A batch of past experiences $\{s_t, a_t, R_t, s_{t+1}\}$ is stored, and PPO is used to update the actor and critic networks. The advantage function is estimated to guide the policy improvement:

$$A_t = R_t + \gamma V(s_{t+1}) - V(s_t), \tag{13}$$

where $V(s)$ is the value function estimated by the critic network, and γ is the discount factor.

A key factor influencing the performance of the reinforcement learning model is the number of training episodes and the batch size. More training episodes allow the agent to explore a broader range of market scenarios, leading to better generalization. Similarly, a larger batch size enables the agent to learn from longer sequences of market fluctuations within each episode, improving its ability to capture long-term dependencies in asset prices. Thus, by training over an extensive number of episodes and using an appropriately large batch size, the model can generalize more effectively and achieve improved trading performance in real-world financial markets.

Testing Phase: Once the training is complete, the final policy is evaluated on a separate test dataset that contains unseen market data. The testing procedure is as follows:

1) The market environment is initialized using historical data from the test dataset.
2) The trained policy is deployed, and the agent selects portfolio allocations without further learning.
3) The agent's performance is returned as total Net worth at the end of a testing episode

By following this structured training and testing pipeline, the reinforcement learning agent learns a robust portfolio allocation strategy that generalizes to new market conditions while accounting for transaction costs and risk factors.

VI. EXPERIMENT

The testing experiments are examined in 1 hour time frame. The testing conditions are changed with these elements: Training batch size, Training episodes, Testing batch size, Testing episode. Moreover, we define 1 episode is 1 hour differences. For example, at the start of episode 0, 1000 USD will be diversified in 3 assets A, B, C and D, which account equally for 25 percent. In the process of training in episode 0, the number of training sessions is equal to Training batch size. Then the results is saved for learning

LR	Training-E	Training-Batch	Test-E	Test-Batch	EIIE	UBAH
0.000001	50	600	15	600	1007	1014
0.000001	100	600	15	600	1010	1005
0.000001	100	600	15	600	1001	995
0.000001	150	600	15	600	1025	1017
0.000001	200	600	15	600	1004	1007
0.00001	50	600	15	600	1005	1010
0.00001	75	600	15	600	1016	1010
0.00001	100	600	15	600	1008	1003
0.00001	150	600	15	600	989	970
0.00001	200	600	15	600	1035	1030
0.00001	200	600	15	600	1001	970
0.0001	100	600	15	600	990	1000
0.0001	150	600	15	600	985	1001
0.001	100	600	15	600	980	999

TABLE I

PERFORMANCE OF EIIE AND UBAH IN TESTING EXPERIMENTS. THERE ARE 5 PARAMETERS WHICH ARE LEARNING RATE (LR), TRAINING EPISODE (TRAINING-E), TRAINING-BATCH (TRAINING BATCH SIZE), TEST-E (TESTING EPISODE), TEST-BATCH (TESTING BATCH SIZE). THE RESULT SHOWS THAT INCREASING LR LEADS TO POOR PERFORMANCE IN EIIE.

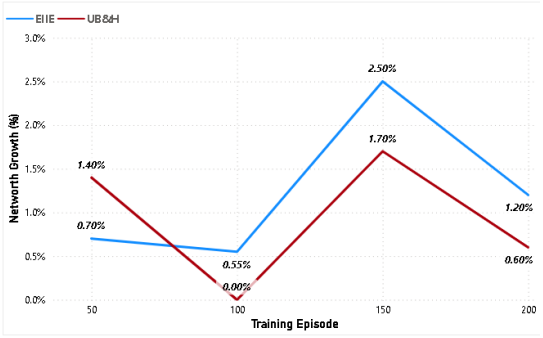

Fig. 2. Comparison the increasing in APV Growth between EIIE and UBAH with Learning rate = 0.000001, Training batch size = 600, Testing episode = 15, Testing batch size = 600

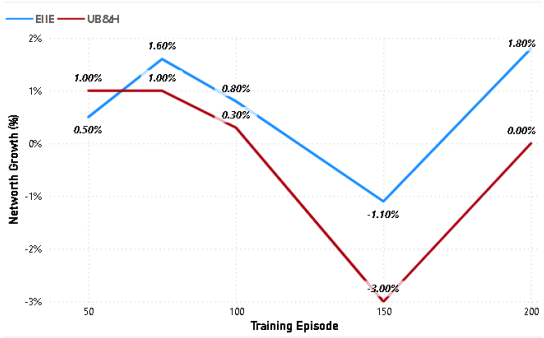

Fig. 3. Comparison the increasing in APV Growth between EIIE and UBAH with Learning rate = 0.00001, Training batch size = 600, Testing episode = 15, Testing batch size = 600

purpose. In the end of episode 0, we will calculate the total money in A, B and C. After 1 episode, the money is reset to 1000 USD.

In the environment, we use 4 different stocks data as assets, which are Pepsi (PEP), Costco (COST), Apple (APPL) and Citigroup Inc. (C). The agent then learn through these 4 assets.

The main compared financial metric is the Uniform Buy and Hold (UBAH) strategy. UBAH is a passive investment strategy in which an investor buys stocks (or other types of securities such as ETFs) and holds them for a long period regardless of fluctuations in the market. [29]

A. Test Ranges

A set of continuous data is chosen when training and testing. At first, the data is divided into 2 parts for training and testing, which is approximately 80 percent and 20 percent. In the training set, it choose a random date and continuously training until it runs all the training batch size. In a training session (1 training batch size), it will base on the history data and history training process (if it is saved in memory) to diversify the money into 4 stocks. After finishing 1 episode, it will choose another random date to entry the training episode.

B. Performance Measures

To measure the result of the model, we calculate the different between the value of stocks in time t compared to time 0. The equation is

$$d_t = (p_t - p_0)/p_0 * 100 \tag{14}$$

which d_t is the difference, p_t is the value of stocks at time t and p_0 is the value of stocks at time 0.

A major disadvantage is that is does not measure the risk factors, since it merely calculate the average of all returns without considering fluctuation in these returns.

C. Results

The performance of the EIIE model is assessed in comparison to the Uniform Buy and Hold (UBAH) strategy. Figures 2 and 3 illustrate the results across different learning rates. Initially, with a limited number of training episodes, EIIE underperforms relative to UBAH due to insufficient training. However, as the number of training episodes increases, EIIE progressively improves and eventually surpasses UBAH. This pattern remains consistent regardless of market conditions.

Table 1 further highlights the impact of learning rates on model performance. At higher learning rates (LR = 0.001, LR = 0.001), EIIE underperforms UBAH due to increased volatility in stock allocation adjustments. This instability suggests that higher learning rates require extended training to achieve optimal performance. Nevertheless, on average, EIIE outperforms UBAH by approximately 50%.

In conclusion, while an excessively high learning rate can negatively affect performance, increasing the number of training episodes substantially enhances the model's effectiveness.

VII. CONCLUSIONS

REFERENCES

[1] Y. Zhang, X. Chen, and J. Li. "Deep Learning for Financial Market Prediction: A Survey". In: *Journal of Financial Technology* 5 (2020), pp. 1–20.

[2] T. Li, P. Zhao, and H. Wang. "Reinforcement Learning in Algorithmic Trading: A Review of Recent Advances". In: *Quantitative Finance* 21 (2021), pp. 1347–1365.

[3] T. Fischer and C. Krauss. "Deep Learning with Long Short-Term Memory Networks for Financial Market Predictions". In: *European Journal of Operational Research* 270 (2018), pp. 654–669.

[4] K. Vasantha Lakshmi and K.N. Udaya Kumara. "A novel randomized weighted fuzzy AHP by using modified normalization with the TOPSIS for optimal stock portfolio selection model integrated with an effective sensitive analysis". In: *Expert Systems with Applications* 243 (2024), p. 122770. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2023.122770. URL: https://www.sciencedirect.com/science/article/pii/S0957417423032724.

[5] Zhengyao Jiang, Dixing Xu, and Jinjun Liang. *A Deep Reinforcement Learning Framework for the Financial Portfolio Management Problem*. 2017. arXiv: 1706.10059 [q-fin.CP]. URL: https://arxiv.org/abs/1706.10059.

[6] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. "Learning Long-Term Dependencies with Gradient Descent Is Difficult". In: *IEEE Transactions on Neural Networks* 5.2 (1994), pp. 157–166. DOI: 10.1109/72.279181.

[7] J. B. Heaton, N. G. Polson, and J. H. Witte. "Deep Learning for Finance: Deep Portfolios". In: *Applied Stochastic Models in Business and Industry* 34.1 (2018), pp. 1–12. DOI: 10.1002/asmb.2283.

[8] John Moody and Matthew Saffell. "Learning to Trade via Direct Reinforcement". In: *IEEE Transactions on Neural Networks* 12.4 (2001), pp. 875–889. DOI: 10.1109/72.935092.

[9] Ralf Neuneier. "Enhancing Q-Learning for Optimal Asset Allocation". In: *Advances in Neural Information Processing Systems*. Vol. 10. 1997, pp. 936–942.

[10] John Moody and Liao Wu. "Performance Functions and Reinforcement Learning for Trading Systems and Portfolios". In: *Journal of Forecasting*. Vol. 17. 5-6. 1998, pp. 441–470. DOI: 10.1002/(SICI)1099-131X(1998090)17:5/6<441::AID-FOR721>3.0.CO;2-4.

[11] Ziqi Liang et al. "Deep Reinforcement Learning in Portfolio Management". In: *arXiv preprint* (2018). eprint: arXiv:1808.09940. URL: https://arxiv.org/abs/1808.09940.

[12] Yifan Hu, Xudong Liu, Yixiang Fang, et al. "Deep Reinforcement Learning for Financial Investment: A Survey". In: *arXiv preprint arXiv:1911.12377* (2019).

[13] TradingView. *TradingView Market Data*. Accessed: 2024-03-03. 2024. URL: https://www.tradingview.com.

[14] Üzeyir Aycel and Yunus Santur. "A New moving average approach to predict the direction of stock movements in algorithmic trading". In: *Journal of New Results in Science* 11.1 (2022), pp. 13–25. DOI: 10.54187/jnrs.979836.

[15] John J. Murphy. "Technical Analysis of the Financial Markets". In: *New York Institute of Finance* (1999).

[16] J. Welles Wilder Jr. *New Concepts in Technical Trading Systems*. Trend Research, 1978.

[17] John Bollinger. *Bollinger on Bollinger Bands*. McGraw-Hill, 2001.

[18] Bin Li, Steven C.H. Hoi, and Vivekanand Gopalkrishnan. "CORN: Correlation-driven nonparametric learning approach for portfolio selection". In: *ACM Trans. Intell. Syst. Technol.* 2.3 (May 2011). ISSN: 2157-6904. DOI: 10.1145/1961189.1961193. URL: https://doi.org/10.1145/1961189.1961193.

[19] Pierre Sermanet, Soumith Chintala, and Yann LeCun. "Convolutional neural networks applied to house numbers digit classification". In: *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*. 2012, pp. 3288–3291.

[20] A. Charles. *Technical Analysis and Stock Market Profits: A Course in Forecasting*. Harriman House, 2006.

[21] Andrew W. Lo. "Adaptive Markets Hypothesis: Market Efficiency from an Evolutionary Perspective". In: *Journal of Portfolio Management* 30.5 (2000), pp. 15–29. DOI: 10.3905/jpm.2004.443819.

[22] Rama Cont, Sasha Stoikov, and Arsen Talreja. "A Stochastic Model for Order Book Dynamics". In: *Operations Research* 58.3 (2010), pp. 549–563. DOI: 10.1287/opre.1090.0741.

[23] Álvaro Cartea, Sebastian Jaimungal, and José Penalva. "Algorithmic and High-Frequency Trading". In: *Cambridge University Press* (2015). DOI: 10.1017/CBO9781139583392.

[24] X. Yang, Y. Zhou, and J. Wang. "Multi-Objective Reinforcement Learning for Financial Portfolio Optimization". In: *Quantitative Finance* 21.5 (2021),

pp. 723–741. DOI: 10.1080/14697688.2021.1903314.

[25] Alessandra Bonfiglioli, Rosario Crinò, and Gino Gancia. "Firms and economic performance: A view from trade". In: *European Economic Review* 172 (2025), p. 104912. ISSN: 0014-2921. DOI: https://doi.org/10.1016/j.euroecorev.2024.104912. URL: https://www.sciencedirect.com/science/article/pii/S0014292124002411.

[26] Gaurav Khemka, Mogens Steffensen, and Geoffrey J. Warren. "A buy-hold-sell pension saving strategy". In: *Insurance: Mathematics and Economics* 119 (2024), pp. 1–16. ISSN: 0167-6687. DOI: https://doi.org/10.1016/j.insmatheco.2024.07.003. URL: https://www.sciencedirect.com/science/article/pii/S0167668724000799.

[27] Puja Das and Arindam Banerjee. "Meta optimization and its application to portfolio selection". In: *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '11. San Diego, California, USA: Association for Computing Machinery, 2011, pp. 1163–1171. ISBN: 9781450308137. DOI: 10.1145/2020408.2020588. URL: https://doi.org/10.1145/2020408.2020588.

[28] John Schulman et al. "Proximal Policy Optimization Algorithms". In: *Proceedings of the 34th International Conference on Machine Learning*. 2017, pp. 1–12. URL: https://arxiv.org/abs/1707.06347.

[29] David Silver et al. "Deterministic Policy Gradient Algorithms". In: *Proceedings of the 31st International Conference on Machine Learning*. Ed. by Eric P. Xing and Tony Jebara. Vol. 32. Proceedings of Machine Learning Research 1. Bejing, China: PMLR, 22–24 Jun 2014, pp. 387–395. URL: https://proceedings.mlr.press/v32/silver14.html.