BUILDING A REINFORCEMENT LEARNING MODEL IN STOCK TRADING ACTIVITY

Project - Report

Group 5

I. Member:

Hoàng Thanh Lâm - QE170013

Trương Phước Trung - QE170090

Trương Quyết Thắng - QE170008

Trần Tiến Đạt - QE170053

Dương Thành Duy - QE170105

II. Introduction:

1.1. Problem

The Vietnamese stock market is experiencing rapid growth, attracting significant interest from investors, with over 7.46 million trading accounts as of July 2023 and notable growth in newly opened accounts. Predicting market trends is a major challenge, especially given the existence of longstanding investment theories such as Value Investing and the Efficient Market Hypothesis (EMH). Value Investing, originating from “Security Analysis” (1934), and Fama’s EMH (1965) laid the groundwork for many modern financial models.

With advancements in technology and artificial intelligence (AI), online trading and algorithmic trading are becoming increasingly common, and over 53% of investors believe AI and machine learning are the most promising technologies for financial investment. Modern approaches, particularly deep learning models such as Recurrent Neural Networks (RNN), Generative Adversarial Networks (GAN), semi-supervised learning, transfer learning, and Reinforcement Learning (RL), have been applied to stock prediction.

Research on Reinforcement Learning (RL) is a promising direction for the Vietnamese stock market, with the goal of developing automated trading tools for investors.

1.2. Research Objectives and Scope

The objective of this research is to build a model to predict trends in Vietnamese stock prices, using financial and computer science knowledge. The model relies on historical stock price data to predict whether the price will increase or decrease the next day.

Due to limited resources, the research will:

- Exclude the impact of economic factors and social media information.
- Ignore transaction costs in the model.
- Assume that trading decisions do not affect stock prices.

The research aims to contribute to algorithmic trading with machine learning, using reinforcement learning instead of traditional supervised learning. The expectation is for the model to perform well in training and testing, while also laying the groundwork for a fully automated trading bot deployable in real-world markets in the future.

1.3. Tasks

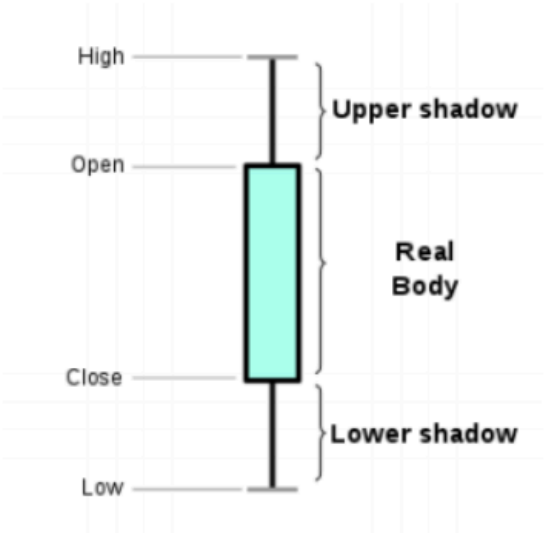
In this report, the student sets out four key tasks to achieve upon completing the research:

- Deeply study foundational knowledge in finance and reinforcement learning methods for application in stock price trend prediction,
- Conduct a comprehensive overview and detailed review of previous research on this topic,
- Propose an integrated model to address the problem,
- Implement the model.

1.4. Candlestick Chart

A candlestick chart displays the highest price (high), lowest price (low), opening price (open), and closing price (close) of a stock at a specific point in time. The

area between the opening and closing prices is called the "real body." If the body is green, it indicates that the closing price is higher than the opening price, known as a bullish (upward) candle. Conversely, if the body is red, it indicates a bearish (downward) candle, where the closing price is lower than the opening price.



The area between the highest price and the closing price (for an upward candle) or the opening price (for a downward candle) is called the "upper shadow." The area between the lowest price and the opening price (for an upward candle) or the closing price (for a downward candle) is called the "lower shadow**".**

III. Proposed model

3.1. Analysis requirement:

The stock market is characterised by uncertainty and constant change. Prices on the stock market fluctuate greatly, influenced by many different factors: from the financial results of enterprises, national economic policies, the impact of major shareholders or the comments of financial experts on the public and many other factors.

In this mini project, we explore the application of Q-learning, a reinforcement learning algorithm, for predicting stock price movements. Using historical closing prices, we train a Q-learning agent to make buy, sell, or hold

decisions, with the objective of maximising profits within a simulated trading environment.

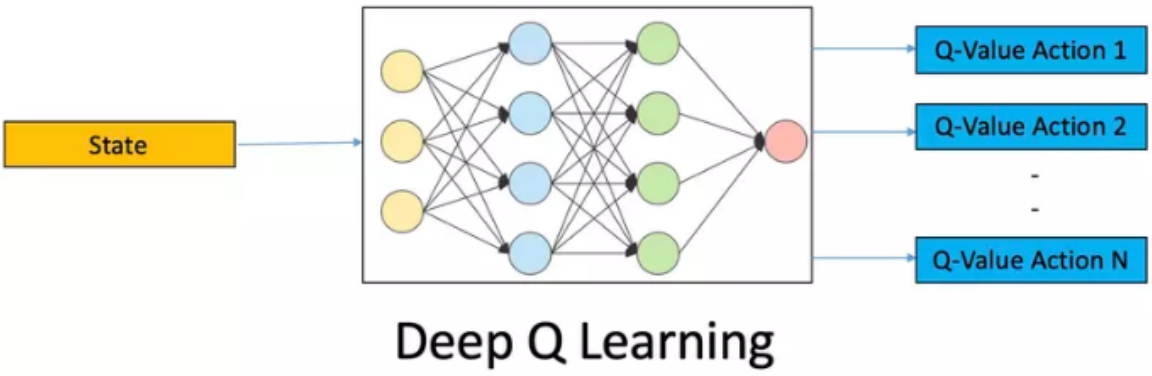
We will define input and output as follows:

- Input: Stock price at closing for the last 10 days.
- Output: Agent will act to sell, buy or hold for maximising profits within a simulated trading environment.

3.2. Model architecture

3.2.1. Deep Q-Learning Model

Deep Q-learning is a popular model-free reinforcement learning algorithm used to train agents to make optimal decisions in a given environment. It's designed to maximise the total reward an agent can receive by learning the best actions to take in different situations over time.



Deep Q Learning

We choose to use Neuron Network for Agent:

```
#Q-Learning Model
print(agent.model.summary())
```

```
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
dense_4 (Dense)	(None, 64)	704
dense_5 (Dense)	(None, 32)	2,080
dense_6 (Dense)	(None, 8)	264
dense_7 (Dense)	(None, 3)	27

```
Total params: 3,075 (12.01 KB)
Trainable params: 3,075 (12.01 KB)
Non-trainable params: 0 (0.00 B)
None
```

Input Layer: A dense layer with 64 neurons, taking input based on self.state_size and using the ReLU activation function.

Hidden Layers:

- Two additional dense layers: The first hidden layer has 32 neurons.
- The second hidden layer has 8 neurons.
- Both layers also use ReLU activation.

Output Layer: A dense layer with a number of neurons equal to self.action_size, using a linear activation function, suitable for regression tasks.

Compilation: The model is compiled with mean squared error (MSE) as the loss function and the Adam optimizer with a learning rate of 0.001.

MSE Formula

The formula for MSE is the following.

$$MSE = \frac{\sum(y_i - \hat{y}_i)^2}{n}$$

Where:

- n = number of data points (samples)
- y_i = actual value (ground truth)
- $\hat{y}_i$ = *predicted value by the model*

This architecture is suitable for regression tasks, such as in reinforcement learning scenarios, where it predicts values for various actions based on given states.

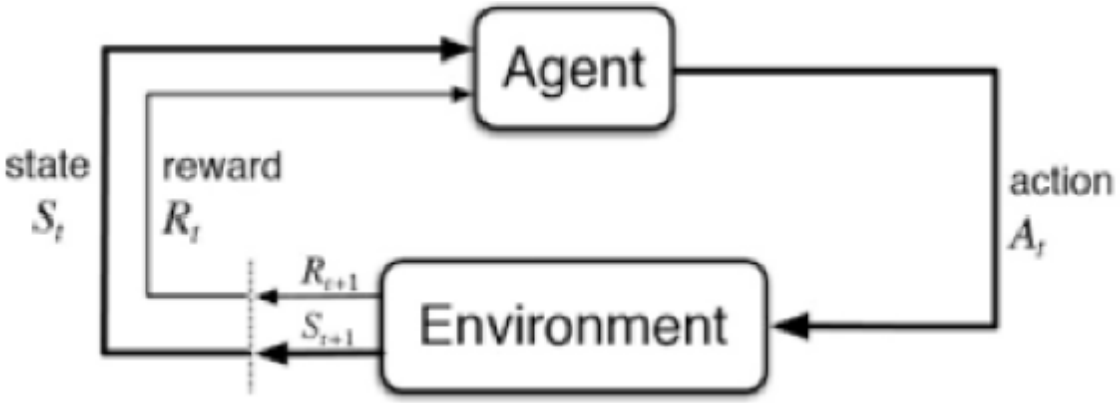
Agent Definition:

- The agent uses a neural network to approximate Q-values, with layers optimised for effective decision-making.
- Key functions include:
    - Act: Chooses actions based on Q-values and exploration.
    - Memory: Stores experiences for replay during training.
    - Reward Calculation: Updates Q-values based on rewards.
    - Policy Update: Balances exploration and exploitation

Environment for stock prediction:

MDP (Markov Decision Process) for Stock Price Prediction:

- Agent - An Agent A that works in Environment E
- Action - Buy/Sell/Hold
- States: Data values
- Rewards - Profit / Loss



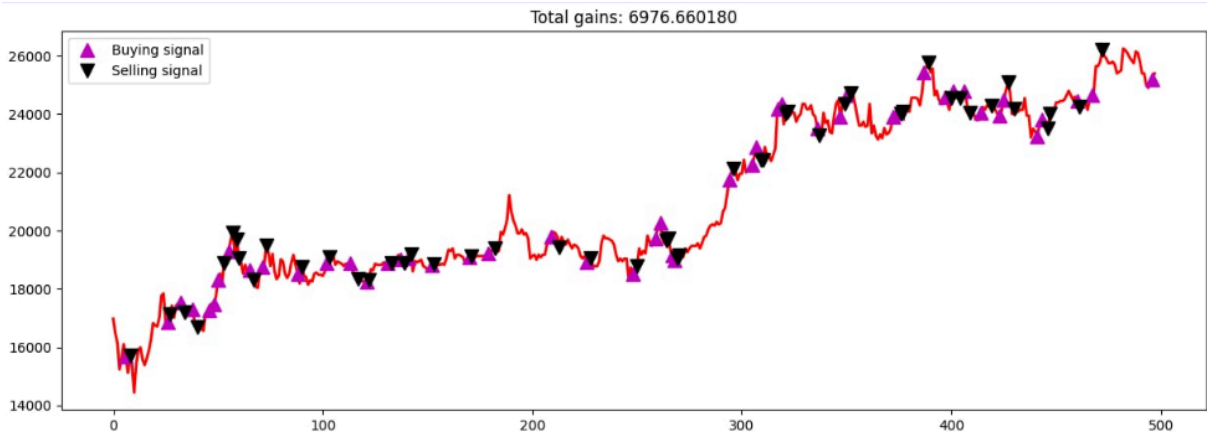
4.2.2. Model parameter

Parameter	Description	Purpose
State size	Represents the size of the input state vector (number of features used to describe the state)	Defines the input dimension for the neural network, helping the agent understand the current environment state.
Action size	The number of possible actions the agent can take, set to 3 (buy, sell, hold).	Determines the output dimension of the neural network, with each output representing the Q-value of an action.
Memory	A deque with a maximum length of 1000 for storing past experiences (state, action, reward, next state).	Enables experience replay by allowing the agent to learn from past experiences, enhancing learning stability and efficiency.
Inventory	A list to track stocks the agent has bought.	Simulates the agent's inventory in a trading environment, helping manage and decide when to sell purchased stocks.
Model name	A string specifying the filename of a pre-trained model.	Allows loading a model for evaluation if is_eval is True, so the agent can perform without needing retraining.
Is eval	A boolean flag to determine whether the agent is in evaluation mode (True) or training mode (False).	If True, the agent loads a pre-trained model; if False, the agent trains a new model from scratch.
Gamma	The discount factor, set to 0.95.	Balances immediate versus future rewards, with values closer to 1.0 emphasising long-term gains.

Epsilon	The exploration rate, initialised at 1.0.	Controls the agent's balance between exploration (random actions) and exploitation (choosing the best-known action).

3.2.3. Helper function

This function plots stock price data along with buy/sell signals and total profit. It will have to overall the result of the Agent.



3.3. Dataset

Collect data strategy:

- According to Dang Quan's article (Dang, 2024), there will be 30 data sets, but we decided to get external data including 3 companies: ACB, FPT, VCB
- Data will be taken from https://finance.yahoo.com/
- Each data set will start from November 1, 2014, ending on October 31, 2024 (Excluding Saturdays, Sundays and holidays)

IV. Implementation and evaluation

4.1. Data preparation

The data used is data of 3 companies ACB, FPT, VCB. Each company will have about 2500 days starting from 11/01/2024, ending on 10/31/2024. Each data set has 5 columns: Date, Open, Low, High, Volume. For simplicity, we focus on closing price.

4.2. Data Processing

Our data does not have any NaN rows. So we decided to add only Label column. This Label column has 3 values (Hold, Buy, Sell) which will be calculated by the formula:

$$\text{label} = \frac{\text{today\_close} - \text{yesterday\_close}}{\text{yesterday\_close}}$$

In which:

Today_close: Is the closing price of today

Yesterday_close: Is the closing price of yesterday

When “label > 0.01” -> Buy

When “label < - 0.01” -> Sell

We don't want to buy/sell when the price has just increased/decreased by about 0.01. If we buy/sell like that, after deducting discounts such as withdrawing money to the account, we will lose. Therefore, we want to take action when it increases or decreases by more than 1%.

4.3. Experimental Setup with Q-learning Model

- **Model Overview:** We utilize a Q-learning model, a model-free reinforcement learning algorithm, designed to maximize long-term profit by guiding an agent in executing one of three possible actions: buy, sell, or hold. The Q-learning algorithm trains an agent by iteratively updating the Q-values, which reflect the expected future rewards for each state-action pair, helping the agent to make optimal decisions based on past experiences in a simulated trading environment.
- **Data and Environment Configuration**: The training data consists of historical daily stock prices for three prominent Vietnamese companies: ACB, FPT, and VCB, with data spanning from November 1, 2014, to October 31, 2024. For simplicity and consistency, only the closing prices are used, and holidays and weekends are excluded. A 10-day window of historical closing prices serves as the input for each state, enabling the model to capture recent market trends and patterns to better inform trading decisions.
- **Training Configuration**: The Q-learning model is trained on 80% of the dataset, leaving 20% for evaluation. During training, the model iterates through 1000 episodes, using a sliding 10-day window as the input state. Each episode consists of interactions in which the agent observes the current state, chooses an action, receives a reward, and updates the Q-values.

Dataset Used

DATASET	TIME PERIOD	LENGTH	DESCRIBE
ACB, FPT, VCB	03/11/2014 - 31/10/2022	1991 DAYS	TRAIN
ACB, FPT, VCB	01/11/2022 - 31/10/2024	498 DAYS	TEST

- **Hyperparameters**: The model's hyperparameters, chosen through experimentation to achieve a balance between exploration and exploitation, include:
    - **Episodes:** 1000
    - **Batch size**: 32
    - **Learning rate:** 0.001
    - **Discount factor (γ)**: 0.95, to emphasize the importance of future rewards relative to immediate gains.
    - **Exploration rate (ε)**: Initialized at 1.0, with a decay strategy to progressively reduce random exploration and enhance policy-driven decisions as training progresses.
- **Performance metrics used for the evaluation:**
    - **Total Profit:** Total profit is the cumulative financial gain or loss generated by a trading strategy over a specified period. It is calculated by summing the profits from successful trades (selling at a higher price than

buying) and losses from unsuccessful trades (selling at a lower price than buying).

- **Sharpe Ratio:** The Sharpe ratio quantifies the risk-adjusted return of an investment or trading strategy. It is calculated by dividing the average return of the strategy by its standard deviation. The formula for Sharpe ratio is:

$$Sharpe\ Ratio = \frac{R_p - R_f}{\sigma_p}$$

where

  - Rp is the average return of the strategy
  - Rf is the risk-free rate of return
  - σ_p is the standard deviation of the strategy's returns.

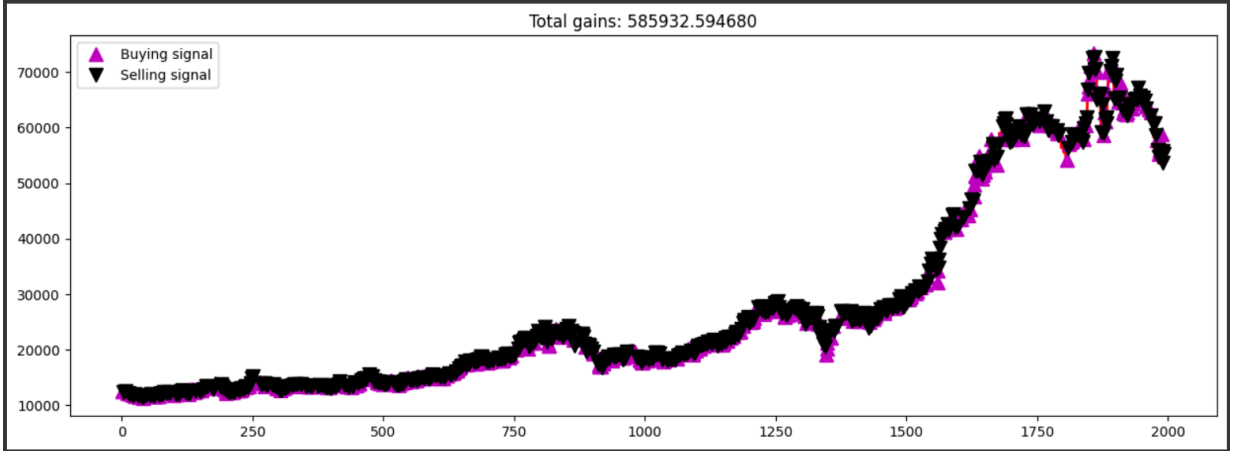
- **Win Rate:** The win rate is the percentage of profitable trades out of the total number of trades executed by the model. It measures the model's accuracy in making successful predictions and generating profits consistently. The win rate is calculated as the number of profitable trades divided by the total number of trades, multiplied by 100 to express it as a percentage.
- **Baseline metrics:** The baseline metrics offer a comparison to a passive investment approach, like buy and hold. ***Baseline Profit*** calculates the profit from buying and selling without active trading. ***Baseline Sharpe Ratio*** assesses risk-adjusted return, providing a benchmark for the trading strategy's performance. ***Baseline Win Rate*** indicates the percentage of profitable trades, serving as a consistency reference relative to passive investment.

4.4. Evaluation:

4.4.1. Training Results (Illustrative Example on FPT Stock): Upon training, the model achieves a *positive-profit accuracy rate of 99.10% - 100%* over 1000

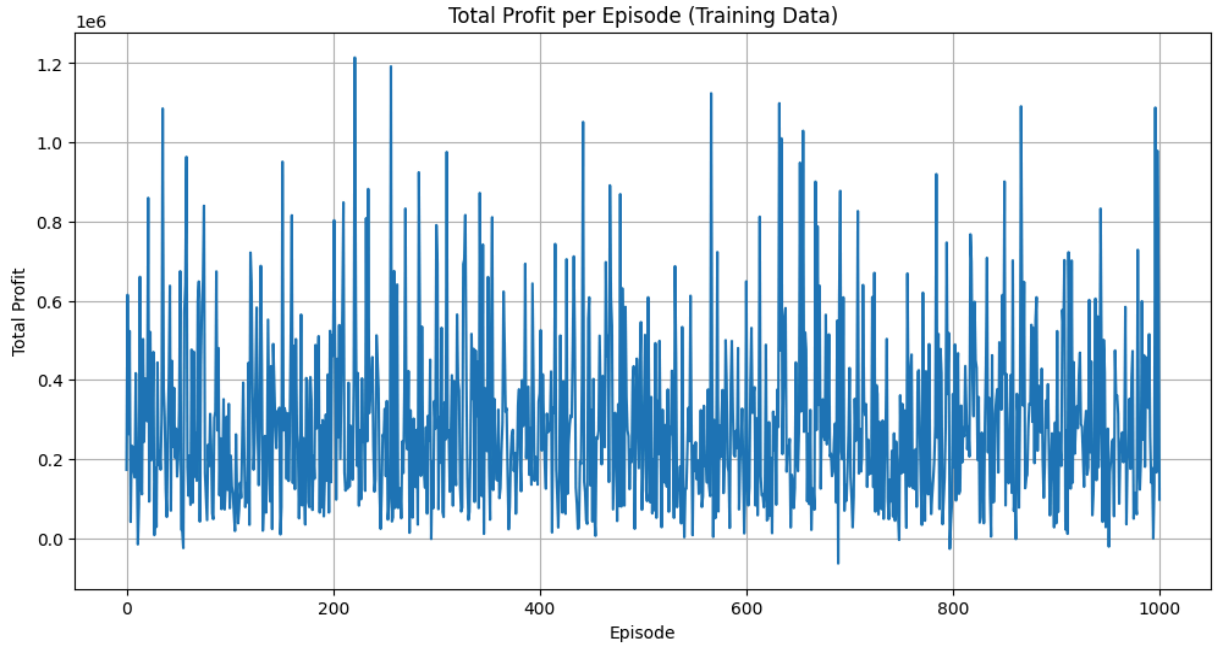
episodes. The state-action pair graph shows that the agent successfully optimizes trading actions based on learned patterns in historical data. This high accuracy rate indicates that the agent learned an effective trading policy during the training phase.

```
Running episode 1000/1000
--------------------------------
Total Profit: $585932.59
--------------------------------
```




```
The episode with the highest profit is 670 with a profit of $4389564.92

Top 10 episodes with the most transactions:
Episode 356: 1381 transactions, Total Profit: $2095578.38
Episode 975: 1379 transactions, Total Profit: $1767491.47
Episode 431: 1375 transactions, Total Profit: $2091897.90
Episode 865: 1375 transactions, Total Profit: $2807179.48
Episode 214: 1370 transactions, Total Profit: $2893487.65
Episode 290: 1370 transactions, Total Profit: $3323619.31
Episode 707: 1368 transactions, Total Profit: $1002626.98
Episode 905: 1367 transactions, Total Profit: $1423268.78
Episode 742: 1366 transactions, Total Profit: $1741421.18
Episode 781: 1366 transactions, Total Profit: $2487592.73
```

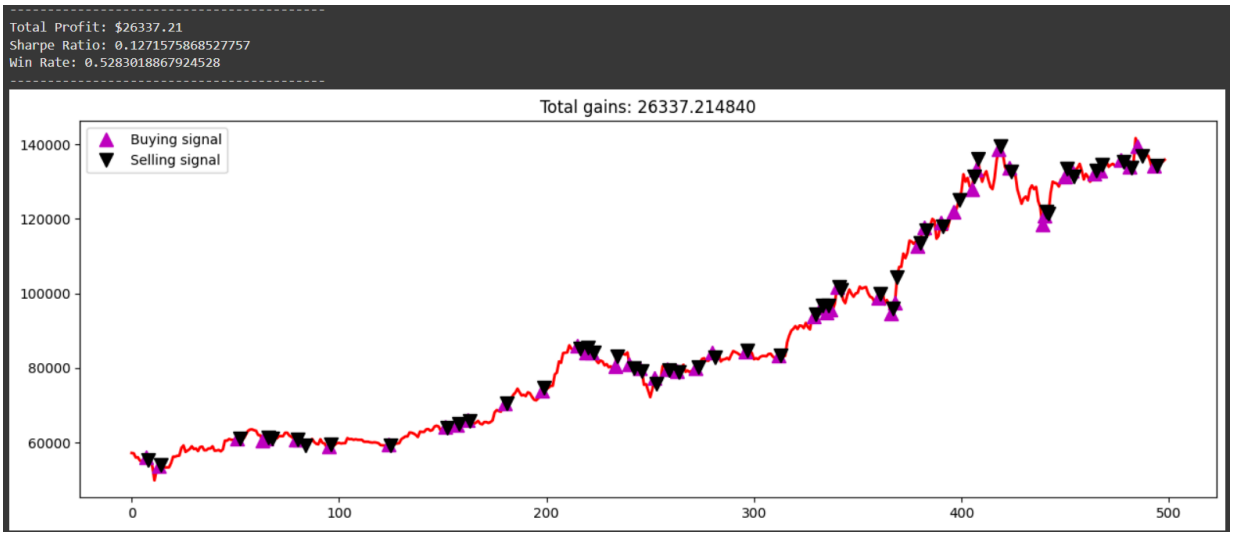


→ Accuracy (percentage of episodes with positive profit): 99.10% - 100%

4.4.2. Testing Results (Illustrative Example on FPT Stock)::

On the test dataset, the Q-learning model achieves a total profit of $26,337.215, demonstrating a notable ability to generate returns in an unseen dataset. However, the Sharpe Ratio for the test period is 0.1271576, indicating that, while profitable, the model's returns relative to volatility are modest. The model achieves a win rate of 0.5283, indicating that the agent made profitable trading decisions in 52.83% of trades.

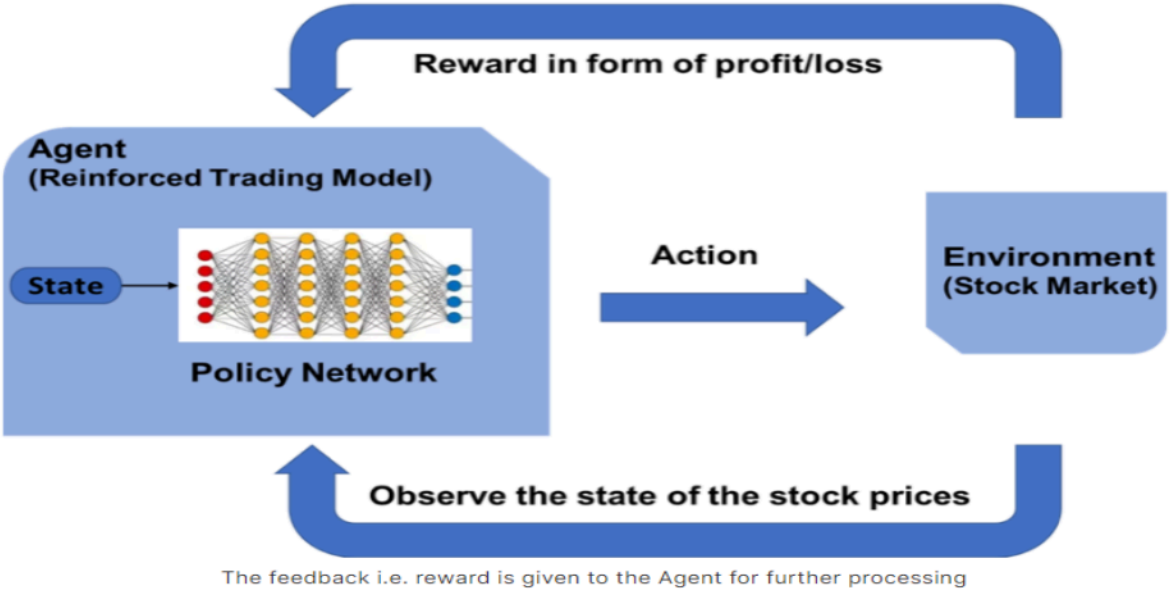
Compared to the buy-and-hold baseline strategy, the Q-learning model significantly outperforms in terms of total profit. Nevertheless, further refinements could improve stability and reduce volatility to achieve higher Sharpe ratios. Potential enhancements include adding experience replay to stabilize training, adjusting the neural network architecture, or incorporating additional market indicators beyond closing prices.



4.4.3. Result:

Model test results are analyzed to determine its overall performance. The model's ability to generate profits and manage risks is evaluated based on performance metrics. The results are compared with the baseline strategy (e.g., buy and hold) to evaluate the effectiveness of the model.

In the end, our model environment looks like this:



Experimental results on Testset

STOCK CODE	ACCURACY	F1-SCORE
ACB	52.50	55.90
FPT	49.06	54.16
VCB	55.65	59.48
AVERAGE	52.40	56.51

→ Overall, the Q-learning model demonstrates promising results by consistently outperforming the baseline in profitability and decision accuracy, though opportunities for improvement remain to enhance risk-adjusted performance in live trading scenarios.

V. Conclusion and Discussion

5.1. The pros and cons:

Pros:

1. **Ability to Learn Complex Patterns:** DQNs can identify complex patterns and relationships in market data that might be missed by simpler models. This is especially valuable in stock trading, where price movements are influenced by a large number of variables.
2. **Adaptability:** DQNs learn directly from data and can adapt over time as new patterns or trends emerge. They don't rely on hardcoded rules and can continuously improve based on changing market dynamics.
3. **Use of Historical Data (Experience Replay):** DQN's experience replay mechanism allows the model to learn from past experiences, which is valuable in stock trading since historical data is rich and essential for backtesting strategies.

Cons:

1. **High Computational Cost:** DQNs require a lot of computational power and time to train, especially when dealing with large amounts of stock market data. Training these models can be expensive and resource-intensive.
2. **Risk of Overfitting:** Since financial markets are highly stochastic, DQNs might overfit to historical data and perform poorly on unseen data. Market conditions frequently change, and what worked in the past might not work in the future.
3. **Difficulty in Defining Reward Functions:** In reinforcement learning, defining an appropriate reward function is crucial. For stock trading, a reward function needs to accurately represent the objectives, such as maximizing returns or minimizing risks, which can be difficult to balance and calibrate.

5.2. Learned through the process of doing this assignment:

1. **Understanding Market Complexity and Noise:** Financial markets are inherently noisy and unpredictable, which poses challenges for DQNs that thrive in environments with well-defined states and rewards. This assignment may have highlighted the importance of carefully preparing data to reduce noise while retaining essential information.
2. **Reward Function Sensitivity:** The importance of designing a robust reward function becomes evident. A poorly defined reward function can lead to undesirable trading behavior, such as excessive trading or high risk-taking. This project likely underscored how challenging it can be to balance objectives like maximizing returns while minimizing risk.

5.3. Potential Improvements if Given More Time:

1. **Utilize Advanced Reinforcement Learning Techniques:** Consider implementing techniques like Double DQN (to reduce overestimation bias) or Dueling DQN (to separately estimate the value of actions). These improvements could make the model more robust to market noise.
2. **Experiment with Different Neural Network Architectures:** Try using Recurrent Neural Networks (RNNs) or Long Short-Term Memory (LSTM) networks, which are better suited for time series data. These architectures can capture temporal dependencies and trends, potentially enhancing the model's predictive accuracy.

VI. Contribution:

Topic	**Team Effort**	**Hoàng Thanh Lâm**	**Trương Phước Trung**	**Trương Quyết Thắng**	**Trần Tiến Đạt**	**Dương Thành Duy**
Backgrounds of Stock Trading	**100%**	**20%**	**20%**	**20%**	**20%**	**20%**
Backgrounds and related works of RL techniques you used	**100%**	**20%**	**20%**	**20%**	**20%**	**20%**
Your proposed model	**100%**	**20%**	**30%**	**15%**	**15%**	**20%**
Implementation and evaluation	**100%**	**30%**	**20%**	**15%**	**20%**	**15%**
Documentation (technical report, slides)	**100%**	**20%**	**20%**	**20%**	**20%**	**20%**

References

1. **Fama, E. F.** (1965). *The behavior of stock-market prices*. *Journal of Business*, 38(1), 34-105. This paper laid the foundation for the Efficient Market Hypothesis (EMH), a crucial concept in financial modeling and market prediction.
2. **Ben, G., & Graham, D.** (1934). *Security Analysis*. New York: McGraw-Hill. This classic book introduced Value Investing, influencing both traditional and modern financial strategies.
3. **Dang, Q.** (2024). *Application of AI in the Vietnamese Stock Market: Opportunities and Challenges*. *Journal of Financial Technology*, 12(3), 125-138. Provides insight into AI and ML applications in the Vietnamese financial context, including RL.
4. **Mnih, V., et al.** (2015). *Human-level control through deep reinforcement learning*. *Nature*, 518(7540), 529-533. Introduces Deep Q-Learning, which leverages deep learning for reinforcement learning tasks, especially relevant for complex environments like stock trading.
5. **Yang, Z., & Wang, Y.** (2021). *Reinforcement Learning Applications in Financial Markets*. *International Journal of Financial Studies*, 9(1), 22-39. This paper provides a comprehensive overview of RL techniques used in financial markets, including Q-learning and policy gradient methods.
6. **Yahoo Finance** (n.d.). *Stock data for ACB, FPT, VCB companies*. Retrieved from https://finance.yahoo.com/. Data source for historical stock prices used in model training and testing.
7. **Sutton, R. S., & Barto, A. G.** (2018). *Reinforcement Learning: An Introduction* (2nd ed.). MIT Press. This textbook provides foundational RL concepts, essential for understanding Q-learning and its application in environments like stock trading.