Design and develop a real-time automated trading system for cryptocurrencies using Reinforcement Learning combined with Spark Streaming and Kafka

*7/2024

1st Quang-Trung Dang *MSSV: 20522067*
University of Information Technology
HCM City, VietNam
20522067@gm.uit.edu.vn 2nd Huu-Lam Nguyen *MSSV: 20521516*
University of Information Technology
HCM City, VietNam
20521516@gm.uit.edu.vn 3rd Minh-Quang Le *MSSV: 20520718*
University of Information Technology
HCM City, VietNam
20520718@gm.uit.edu.vn 4th Nguyen-Lich Phan *MSSV: 20521526*
University of Information Technology
HCM City, VietNam
20521526@gm.uit.edu.vn

***Abstract*—Binance is a reputable and popular cryptocurrency exchange founded in 2017 by Changpeng Zhao (CZ). The exchange quickly became one of the largest cryptocurrency trading platforms in the world [1], providing users with a safe, efficient, and user-friendly environment to buy, sell, and trade various cryptocurrencies. Designing effective investment strategies is crucial but challenging due to the high volatility of the market, trends, noise, and unexpected fluctuations. This paper presents the design and implementation of a real-time automated cryptocurrency trading system using a Reinforcement Learning model, combined with Spark Streaming and Kafka technology. The main objective of the system is to make effective and timely trading decisions by leveraging continuous market data. The system is capable of real-time trading, adapting to rapid market changes, and capitalizing on profit opportunities.**

Keywords: ***Index Terms*—Machine Learning, Deep Learning, Deep Q Network, Linear, Long Short-Term Memory (LSTM), Big Data, Streaming Data**

I. Introduction

Binance is one of the world's leading reputable and popular cryptocurrency exchanges. It quickly became one of the largest cryptocurrency trading platforms globally, providing users with a safe, efficient, and user-friendly environment to buy, sell, and trade various cryptocurrencies.

Binance offers a wide range of services and products for cryptocurrency trading. It provides competitive trading fees, lower than many other exchanges. The transaction speed can handle millions of trades per second, ensuring fast and efficient trading.

Security is one of Binance's strongest points. Due to the highest levels of security applied to registration, identity verification, and data replication, it is a global example of security systems and technologies applied to an advanced platform.

Automated Trading Systems (ATS) are increasingly popular due to their ability to address challenges in traditional trading. However, building an effective ATS requires extensive knowledge and skills.

Developing a system that can analyze and collect data in real time requires seamless coordination across multiple fields and skills. This not only helps improve trading performance but also better manage risks and optimize profits.

This report presents the construction of a real-time automated cryptocurrency trading system, applying a Reinforcement Learning model to optimize trading efficiency. The system integrates Spark Streaming and Kafka technologies to process continuous market data and make timely trading decisions based on cryptocurrency trading data from the Binance exchange.

Our goal is to provide an efficient and reliable automated cryptocurrency trading solution, promoting the application of machine learning in finance and cryptocurrency sectors. [2]

II. Related works

Although conventional deep neural networks have achieved significant success in online trading, there are still some limitations when applied in practice. These limitations include

the requirement for large labeled datasets and difficulties in making convincing predictions to execute trading actions. Additionally, market instability can reduce the performance of traditional deep learning models.

A promising alternative is the use of reinforcement learning algorithms. In this paper, we apply the Deep Q Network (DQN) algorithm [3] to optimize trading decisions based on market information. The DQN can learn and improve predictions as well as trading strategies over time.

For a dynamic, constantly changing, and unstable financial environment, continuous learning allows the DQN to capture new trends and adapt to market volatility.

Moreover, we use Apache Kafka [4] to monitor and process market data as it occurs, enabling rapid and accurate model state updates. Combining this with Apache Spark [5] during training and deployment helps accelerate computations and handle large and complex datasets easily and reliably. The system is also designed to be easily scalable, enhancing its future scalability.

III. Methodology

A. Overview

This study focuses on applying Reinforcement Learning (RL) algorithms [6] to build an RL model that allows the system to autonomously learn from past trading experiences and improve trading decisions over time. The system utilizes Apache Kafka and Spark Streaming to process data from the Binance WebSocket, performing predictions and model training directly on the Spark platform. The ultimate goal is to create an automated cryptocurrency trading system capable of optimizing profits and rapidly adapting to the volatility of the cryptocurrency market in real time. The report also evaluates the performance and effectiveness of the system through experiments and analysis of the trading results.

B. Deep Q Network Model

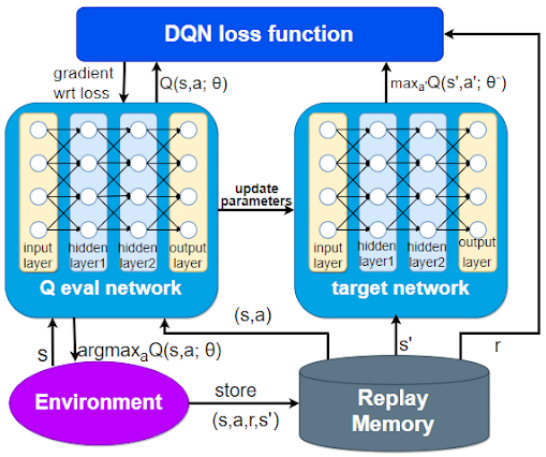

Fig. 1. Deep Q Network Model.

Reinforcement Learning is a machine learning approach applied to optimize decision-making for buying, selling, or holding financial assets to achieve maximum profit.

$$R = \sum_{t=0}^{\infty} r_t$$

Where:

- R is understood as the total reward.

In this section, we use the Q-learning algorithm, a reinforcement learning method based on a value function called the Q-function, used to evaluate the potential of actions in specific states.

First, the profit value is based on the current state (s_t) and the strategy $\pi[a|s]$. From this state, through subsequent state sequences, actions are generated and rewards are obtained. The strategy $\pi[a|s]$, the probability of state transition $Pr(s_{t+1}|s_t, a_t)$ and the probability of reward $Pr(r_{t+1}|s_t, a_t)$ are all random. The formula for the state value based on the strategy $\pi[a|s]$ is:

$$v[s_t|\pi] = \sum_{a_t} \pi[a_t|s_t](r[a_t, s_t]\gamma . \sum_{a}$$

$$+\gamma . \sum_{s_{t+1}} Pr(s_{t+1}|s_t, a_t)v[s_{t+1}|\pi])$$

Similarly, the value of an action based on the strategy $\pi[a|s]$ is the immediate reward $r_{t+1} = r[a_t, s_t]$ obtained from that action plus the value of the future state $v[s_{t+1}]$. Since s_{t+1} is not deterministic, we consider the transition probabilities $Pr(s_{t+1}|s_t, a_t)$ to weigh the value of $v[s_{t+1}]$.

$$q[s_t, a_t|\pi] = r[a_t, s_t] + \gamma . \sum_{s_{t+1}} Pr(s_{t+1}|s_t, a_t)$$

$$(\sum_{a_{t+1}} \pi[a_{t+1}|s_{t+1}]q[s_{t+1}, a_{t+1}|\pi])$$

Updating the policy with the goal of maximizing the value of actions is done as follows:

$$\pi[a_t|s_t] \leftarrow \underset{a_t}{argmax}[Pr(s_{t+1}|s_t, a_t)(r[s_t, a_t] + \gamma . v[s_{t+1}])]$$

In trading, the state-action space is often large and complex. Therefore, the discrete representation $q(s_t, a_t)$ of actions is replaced by a machine learning model $q(s_t, a_t, \phi)$. The loss function is defined using the Huber loss formula:

$$\delta = Q(s, a) - (r + \gamma \underset{a}{max} Q(s', a))$$

$$L(\delta) = \begin{cases} \frac{1}{2}\delta^2 & for|\delta| \leq 1 \\ |\delta| - \frac{1}{2} & otherwise. \end{cases}$$

$$L = \frac{1}{|B|} \sum_{s,a,s',r \in B} L(\delta)$$

The update of the neural network parameters is:

$$\phi \leftarrow \phi + \alpha(r[a_t, s_t] + \gamma . \underset{a_{t+1}}{max}[q[s_{t+1}, a_{t+1}, \phi]])$$

$$-q[s_t, a_t, \phi]) \frac{\theta q[s_t, a_t, \phi]}{\theta \phi}$$

To ensure the algorithm continuously updates with new data and improves its intelligence over time, our team proposes using the Target-Policy architecture for continuous learning or online learning.

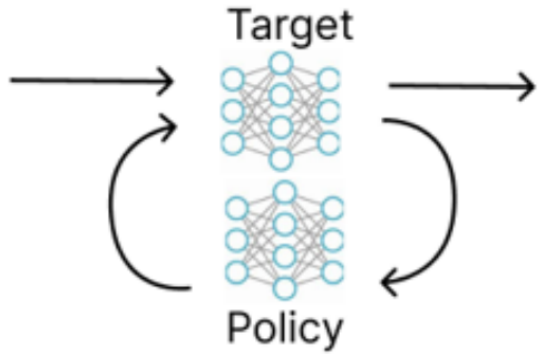

Fig. 2. Target-Policy network architecture.

The Target-Policy architecture is a method designed around two separate neural networks: a Policy network used for training and a Target network used for prediction. Both networks operate concurrently during training and prediction of new data. Periodically, the parameters of the Policy network are assigned to the Target network to synchronize the latest knowledge from the Policy network for future prediction tasks.

With this architecture, as new data is collected, the Policy network continues to be trained to update its knowledge and improve performance, while the Target network is used for predicting action values and estimating optimal values during the data collection process.

C. Linear DQN và LSTM DQN

Here, we are exploring and utilizing two types of networks, namely Linear and LSTM in the Deep Q-Network (DQN) framework to assess their optimization potential when integrated with the DQN model.

1) ***Linear DQN****:* The Linear Network [7] is a type of neural network that is simple, consisting only of linear layers or fully-connected layers without non-linear activation functions. It can be understood as a neural network where the output of each layer is a linear combination of the inputs, without any non-linear activation functions between the layers.

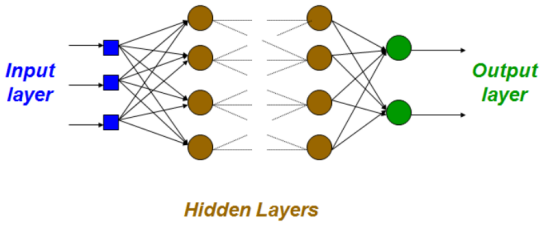

Fig. 3. Linear Model.

A linear network can be represented by the formula:

$$y = Wx + b$$

Where:

- y is the output of the layer.
- W is the weight matrix.
- x is the input to the layer.
- b is the bias vector.

Characteristics of a Linear Network:

- **Simplicity**: The Linear Network has a straightforward structure without non-linear activation functions like ReLU, Sigmoid, or Tanh.
- **Lack of Non-linear Representation**: Because it only uses linear combinations, a Linear Network cannot represent or learn non-linear relationships in data.
- **Speed**: Due to the absence of non-linear functions, computations in a Linear Network are typically fast and efficient.

2) ***LSTM DQN****:* Long Short-Term Memory (LSTM) [8] is a type of Recurrent Neural Network (RNN) designed to handle and predict sequential data, such as time series and natural language data. LSTM was introduced by Hochreiter and Schmidhuber in 1997, and it has become popular in various applications such as speech recognition, natural language processing, and time series forecasting.

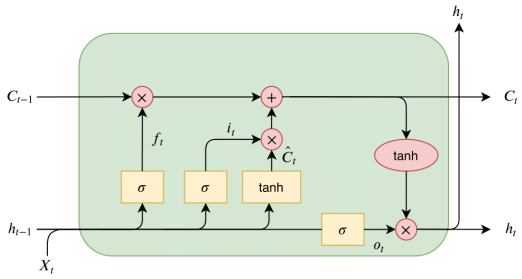

Fig. 4. Long Short-Term Memory Model.

Forget gate:

$$f_t = \sigma(W_f * [h_{t-1}, x_t] + b_f)$$

- f_t: output vector of the Forget Gate at time t.
- W_f: weight matrix for the Forget Gate.
- $[h_{t-1}, x_t]$: concatenated vector of the previous hidden state $t-1$ and current input x_t.
- b_f: bias vector for the Forget Gate.
- σ: sigmoid activation function, outputs range from 0 to 1.

Creating a vector to decide which values will be updated:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

- i_t: output vector of the Input Gate at time t.
- W_i: weight matrix for the Input Gate.
- $[h_{t-1}, x_t]$: concatenated vector of the previous hidden state $t-1$ and current input x_t.
- b_i: bias vector for the Input Gate.

Creating a vector of new values that can be added to the memory cell state:

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

- $\tilde{C}_t$: vector of new memory cell values at time t.

- W_C: weight matrix for the memory cell.
- b_C: bias vector for the memory cell.
- $\tanh$: hyperbolic tangent activation function, outputs range from -1 to 1.

The updated memory cell state C_t is calculated by combining partially forgotten old state and new information from the Input Gate:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

- C_t: memory cell state at time t.
- f_t: output of the Forget Gate.
- C_{t-1}: memory cell state at time $t-1$.
- i_t: output of the Input Gate.
- $\tilde{C}_t$: new memory cell value.

The Output Gate determines which part of the memory cell state will be output and become the next hidden state h_t:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

- o_t: output vector of the Output Gate at time t.
- W_o: weight matrix for the Output Gate.
- $[h_{t-1}, x_t]$: concatenated vector of the previous hidden state $t-1$ and current input x_t.
- b_o: bias vector for the Output Gate.
- σ: sigmoid activation function, outputs range from 0 to 1.

Finally, the new hidden state h_t is determined by combining the Output Gate and the new memory cell state passed through the hyperbolic tangent function:

$$h_t = o_t * \tanh(C_t)$$

- h_t: hidden state at time t.
- o_t: output of the Output Gate.
- C_t: new memory cell state.
- $\tanh$: hyperbolic tangent activation function, outputs range from -1 to 1.

D. Operation

In this section, we describe in detail the operational process of the system. This process includes steps from data collection via Binance WebSocket, data transmission through Kafka, processing and prediction in Spark, to data storage and executing buy/sell orders on Binance. The system is deployed using Docker Compose to ensure flexibility and ease of management.

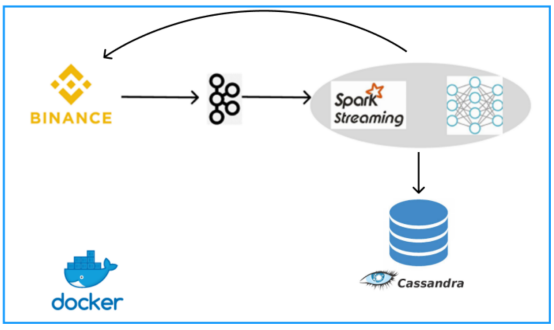

Fig. 5. Operational process.

*1) **Collecting Data from Binance WebSocket**:* Using a Python module, the system connects to Binance's WebSocket to receive data on the prices and trading volumes of cryptocurrencies. This data is then forwarded to Kafka for further processing.

*2) **Transmitting Data through Kafka**:* Apache Kafka is responsible for transmitting data from Binance's WebSocket to Spark. Information on the prices and volumes of cryptocurrencies is sent through Kafka topics so that Spark Streaming can receive and process it.

*3) **Prediction and Model Training in Spark**:* Spark Streaming reads data from Kafka and processes it in real time. The Deep Q-Network (DQN) model is used to predict cryptocurrency prices and is continuously trained with new data from Kafka to improve accuracy.

*4) **Storing Data and Sending Trading Orders**:* After processing and prediction, the data is stored in a Cassandra database. Buy and sell orders are sent directly to Binance's server for execution. The entire system is managed and deployed through Docker Compose, ensuring flexibility and ease of deployment.

IV. Dataset and Experiments Results

A. Dataset

This study uses data from cryptocurrency trading pairs on the Binance exchange, covering the period from January 2021 to January 2023, with an hourly time frame. The year 2021 was chosen because it was a period of significant volatility in the cryptocurrency market, and the data from this year is also used to predict subsequent years.

Fig. 6. Training, Validation, and Trading Process.

The dataset includes the following attributes:

- open time: The opening time of the trading session.
- close: The closing price of the cryptocurrency during that time period.
- volume: The trading volume during that time period.
- count: The number of trades executed during that time period.
- taker buy volume: The trading volume of taker trades (trades executed at the current market price).
- symbol: The name or code of the cryptocurrency being traded.

In this study, data from five cryptocurrencies—BNBUSDT, BTCUSDT, ETHUSDT, LTCUSDT, and XRPUSDT—were used to build and train the model for predicting Bitcoin prices (BTCUSDT). The use of different cryptocurrencies makes the model more diverse and capable of handling various market situations.

B. Experiments results

We present the trading results of the Reinforcement Learning algorithms [9] after being trained with data from 2021 and applied to data from January 2023.

Through experiments, the Linear-DQN algorithm demonstrated better profit optimization compared to LSTM-DQN. The number of buy and sell orders for Linear-DQN was also lower, indicating that although LSTM might have advantages in handling time series data, in a highly volatile financial environment, Linear-DQN can still perform more effectively.

TABLE I
TABLE COMPARING THE RESULTS OF LINEAR-DQN AND LSTM-DQN

	Linear-DQN	**LSTM-DQN**
Initial profit ($)	0.0	0.0
Final profit ($)	15,774.90	11,772.80
Number of Buy Orders	559	577
Number of Sell Orders	556	575

The charts below illustrate the profit results and trades of the two algorithms:

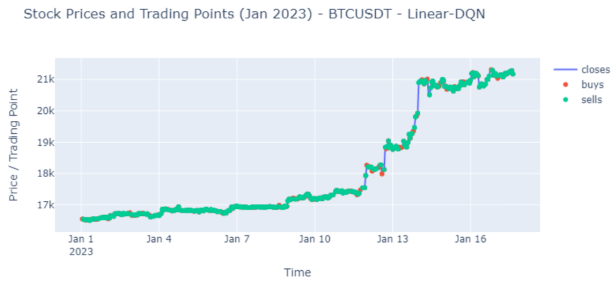

Fig. 7. The trading chart of the Linear-DQN algorithm.

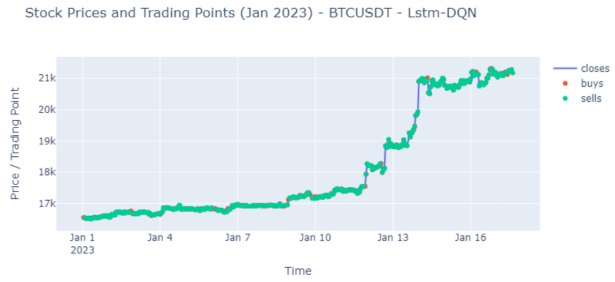

Fig. 8. The trading chart of the LSTM-DQN algorithm.

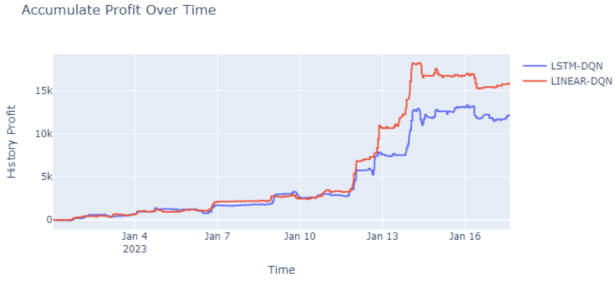

Fig. 9. Profit achieved chart.

On the account usage chart, placing a buy order decreases the account balance, and selling returns funds, creating upward and downward movements on the chart:

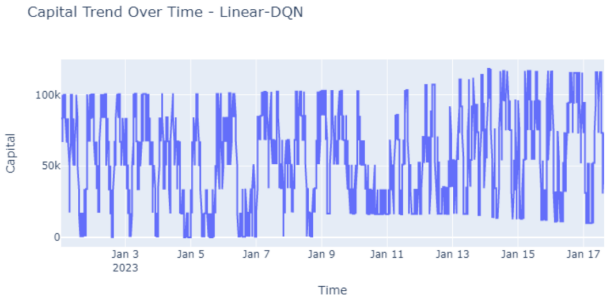

Fig. 10. The account usage chart with the Linear-DQN algorithm.

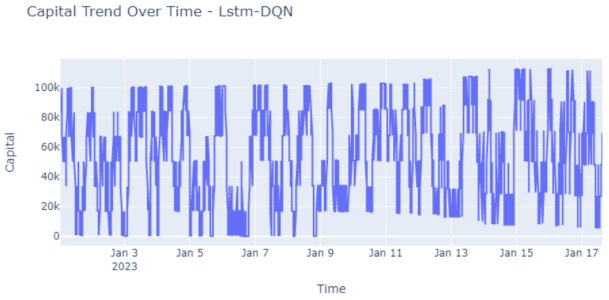

Fig. 11. The account usage chart with the LSTM-DQN algorithm.

V. CONCLUSION

The Deep Q-Network (DQN) method, though fundamental in reinforcement learning, has demonstrated its impressive effectiveness. Integrating and deploying it with Kafka and Spark has improved the algorithm's speed and stability, making it easier to apply in practical applications.

Results show that the Linear-DQN model has optimized profit better. Meanwhile, the LSTM-DQN model can execute more trades, reacting swiftly to cryptocurrency market fluctuations. This underscores that while LSTM excels in time series processing tasks, it isn't always superior to other neural networks, especially in volatile financial environments like cryptocurrency markets.

In summary, the DQN method has proven its impressive effectiveness in automated trading in the cryptocurrency market. Using simple network designs combined with Kafka and Spark integration has enhanced performance and brought the algorithm closer to real-world applications.

REFERENCES

[1] Rishabh Garg. Blockchain for decentralized finance: Impact, challenges and remediation. *International Journal of Computer and Information Engineering*, 17(3):225–237, 2023.

[2] Sushree Das, Ranjan Kumar Behera, Santanu Kumar Rath, et al. Real-time sentiment analysis of twitter streaming data for stock prediction. *Procedia computer science*, 132:956–964, 2018.

[3] Melrose Roderick, James MacGlashan, and Stefanie Tellex. Implementing the deep q-network. *arXiv preprint arXiv:1711.07478*, 2017.

[4] Ted Dunning and Ellen Friedman. *Streaming architecture: new designs using Apache Kafka and MapR streams*. "O'Reilly Media, Inc.", 2016.

[5] Ben Blamey, Andreas Hellander, and Salman Toor. Apache spark streaming, kafka and harmonicio: a performance benchmark and architecture comparison for enterprise and scientific computing. In *International Symposium on Benchmarking, Measuring and Optimization*, pages 335–347. Springer, 2019.

[6] Zihao Zhang, Stefan Zohren, and Roberts Stephen. Deep reinforcement learning for trading. *The Journal of Financial Data Science*, 2020.

[7] *Introduction to Machine Learning* by Ethem Alpaydin.

[8] Hochreiter, Sepp, and Jurgen Schmidhuber. *Long short-term memory* in Neural Computation.

[9] Thanh Thi Nguyen, Ngoc Duy Nguyen, Peter Vamplew, Saeid Nahavandi, Richard Dazeley, and Chee Peng Lim. A multi-objective deep reinforcement learning framework. *Engineering Applications of Artificial Intelligence*, 96:103915, 2020.