COMP0162 Advanced Machine Learning in Finance
Deep Q-Learning in Cryptocurrency Trading: A Comparative Performance Analysis of "Meme Coins" Versus Bitcoin

Abstract

This study is focused on evaluating the performance of Deep Q-Networks (DQN) in trading Bitcoin versus 'memecoins' (MEME, DOGE, PEPE) against the backdrop of the volatile cryptocurrency market. While memecoins are subject to significant fluctuations and Bitcoin is perceived as relatively stable, both offer unique challenges and opportunities for trading strategies.

The primary aim of this research is to compare the effectiveness of a DQN trading agent against two baseline models: Q-Learning trading agent and a double moving average crossover strategy. Through rigorous backtesting on historical data, the study evaluates the profitability, risk, and operational dynamics of DQN in contrast to these baseline models across the diverse volatility profiles of Bitcoin and memecoins. The comparison aims to highlight the advantages and potential limitations of using DQN in cryptocurrency trading, offering insights into its performance relative to more traditional and simplistic trading models. This paper contributes to the broader discourse on financial technology and algorithmic trading by providing empirical evidence and strategic analysis of advanced machine learning techniques in the context of volatile financial markets.

Our results demonstrate that the DQN trading agent not only outperforms the baseline models in terms of profitability and risk management, particularly in the erratic memecoin market, but also exhibits adaptable and effective decision-making in the relatively stable Bitcoin market, reinforcing the potential of DQN as a robust trading strategy in diverse cryptocurrency environments.

1 Introduction

Blockchain technology has ushered in a new era of financial instruments, with cryptocurrencies emerging as a pivotal innovation. Among these, Bitcoin has established itself as the forerunner, demonstrating significant growth and market stability. Concurrently, a new breed of cryptocurrencies known as meme coins, such as DOGE, MEME, and PEPE, have gained popularity, characterized by their volatile market behavior and community-driven value.

This paper delves into the application of Deep Q-Learning, a reinforcement learning technique, to the domain of cryptocurrency trading. While existing literature extensively explores the efficacy of machine learning models in predicting stock prices and traditional financial instruments, limited research is available on their application to the distinct dynamics of Bitcoin and meme coins.

Li and Yang analysis on the interconnectedness between leading cryptocurrencies and memecoins reveals that memecoins often act as net receivers of spillovers, influencing the volatility and price dynamics of the broader cryptocurrency market.[4] This interplay underscores the importance of examining how Deep Q-Learning algorithms can navigate and capitalize on the unique market conditions presented by meme coins compared to Bitcoin.

Further, the effectiveness of Deep Q-Learning in the realm of stock market trading is confirmed by the findings presented by Chakole et al. Their research follows the development of optimal dynamic trading strategies using a Reinforcement Learning framework, specifically employing the Q-learning algorithm. This approach was aimed at addressing the challenge of selecting a suitable trading strategy for a given stock at a specific time, a significant issue in stock market trading. By proposing two novel methods to represent the discrete states of the trading environment, the study demonstrates how these representations can impact the performance of the trading agent.[2]

The practical application of these models was tested on real data from the Indian and American stock markets. The results showed that the strategies devised using the Q-learning algorithm outperformed traditional Buy-and-Hold and Decision-Tree

based strategies in terms of profitability. This reinforces the potential of using advanced reinforcement learning techniques, like Deep Q-Learning, in developing trading strategies that can adapt to changing market conditions and yield higher returns compared to more conventional methods.

Moreover, Zhang et al. illustrate the proficiency of Deep Reinforcement Learning in the domain of trading, particularly with the application of Deep Q-Learning. Their study on trading continuous futures contracts using reinforcement learning techniques such as Deep Q-Learning, Policy Gradients, and Advantage Actor-Critic methods, showcases the potential for these algorithms to outperform classical strategies and adapt dynamically to market conditions, providing a strong foundation for applying these methodologies to the cryptocurrency markets.[8]

By conducting a comparative analysis of the performance of Deep Q-Learning in trading these cryptocurrencies, this study aims to uncover insights into the adaptability and efficiency of reinforcement learning algorithms in varied market conditions. The choice of machine learning, particularly Deep Q-Learning, is predicated on its ability to make sequential decisions and learn optimal policies in the unpredictable realm of cryptocurrency markets, offering a nuanced approach to trading strategies.

2 Methodology

For the purposes of this research, it is important to highlight the main difference between a typical Q-Learning and a Deep Q-Learning model lies in the way they handle the state-action value function (Q-value). Traditional Q-Learning utilizes a Q-table, which is a simple data structure (like a matrix) to store and update the Q-values for each state-action pair. However, this approach is infeasible in environments with large or continuous state spaces because the Q-table would become too large to handle efficiently.[3]

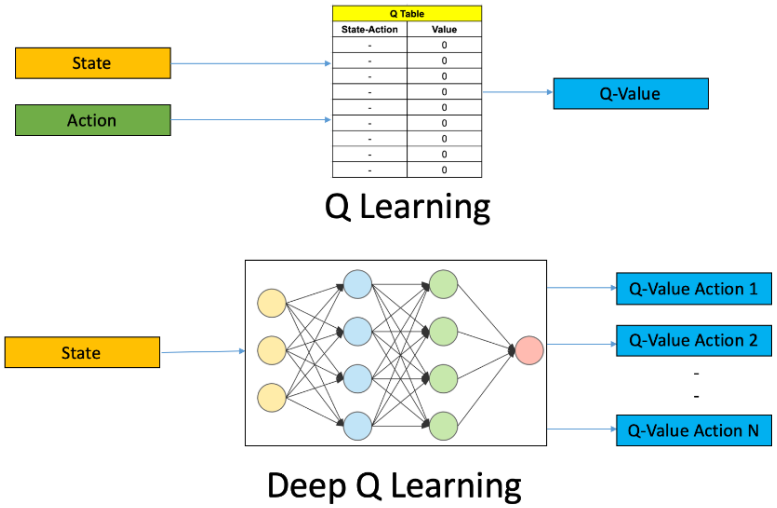


Figure 1: Q-Learning vs. Deep Q-Learning structure

Deep Q-Learning, on the other hand, uses a neural network to approximate the Q-value function. This network takes the state as input and outputs Q-values for all possible actions. The main advantage is the ability to generalize over similar states, thus allowing Deep Q-Learning to work effectively in environments with large or infinite state spaces, which is often the case in real-world scenarios such as trading. The neural network, known as the Q-network, is trained to minimize the difference between the predicted Q-values and the target Q-values, which are computed based on the Bellman equation. This process allows the model to learn complex patterns and make decisions in complex environments.

2.1 Summary of Optimal Action-Value Function

The goal of a typical reinforcement learning agent is to maximize cumulative future rewards through actions taken in an environment. The agent operates under the assumption that future rewards are discounted by a factor of γ per time-step. The discounted return at time t is given by:

$$R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'} \tag{1}$$

where $r_{t'}$ is the reward at time t', and T is the terminal time-step.

The optimal action-value function, denoted as $Q^*(s, a)$, represents the maximum expected return obtainable after observing some state s and then taking an action a. It is defined as:

$$Q^*(s, a) = \max_{\pi} \mathbb{E}\left[R_t | s_t = s, a_t = a, \pi\right] \tag{2}$$

where π is a policy mapping states to actions.

The core of this function is the Bellman equation, which provides a recursive definition for Q^*:

$$Q^*(s, a) = \mathbb{E}_{s' \sim \varepsilon}\left[r + \gamma \max_{a'} Q^*(s', a') | s, a\right] \tag{3}$$

where s' is the state reached after taking action a in state s, and the expectation is over the possible next states s'. This equation acts as the foundation for finding the policy that maximizes the expected return, by iteratively updating the Q-values towards the optimal Q-values.[6]

2.2 Differences Between DQN and Traditional Q-Learning

Deep Q-Networks (DQN) extend the traditional Q-learning by approximating the Q-function using deep neural networks. The fundamental approach in reinforcement learning is to estimate the action-value function with the Bellman equation. The iterative update in traditional Q-learning is expressed as:

$$Q_{i+1}(s, a) = \mathbb{E}\left[r + \gamma \max_{a'} Q_i(s', a') | s, a\right], \tag{4}$$

where $Q_i(s, a)$ is expected to converge to $Q^*(s, a)$ as i approaches infinity.

However, in DQN, the Q-function, parameterized by weights θ, is learned through neural networks, and the update rule is defined by a sequence of loss functions $L_i(\theta_i)$ that changes at each iteration i. The loss function is given by:

$$L_i(\theta_i) = \mathbb{E}_{s, a \sim \rho(\cdot)}\left[\left(y_i - Q(s, a; \theta_i)\right)^2\right] \tag{5}$$

where $y_i = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) | s, a\right]$ is the target for iteration i, and $\rho(s, a)$ is a probability distribution over sequences s and actions a known as the behavior distribution.

In DQN, unlike in traditional methods, the targets are dependent on the network weights and are held fixed during the optimization of $L_i(\theta_i)$. This can be understood as a form of bootstrapping, where we bootstrap the targets from the previous iteration's network weights.

Differentiating the loss function with respect to the weights, we obtain the gradient:

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s, a \sim \rho(\cdot); s' \sim \varepsilon}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i)\right) \nabla_{\theta_i} Q(s, a; \theta_i)\right] \tag{6}$$

DQN is considered a model-free approach as it does not model the environment, but learns from the strategy through sampling and optimizes the action-value function directly off-policy. This allows the network to learn a policy that can generalize across states, contrasting with the tabular methods where a value is learned for each state-action pair without generalization.[6]

2.3 Data Collection

2.3.1 Data Collection for DQN Trading Algorithm

Our DQN trading algorithm requires a robust dataset of price movements to train and test its decision-making capabilities. To facilitate this, we utilize the 'ccxt' library, which connects to the Kucoin exchange, allowing us to fetch historical Open-High-Low-Close-Volume (OHLCV) data for various cryptocurrencies.[1]

We have defined specific time frames for the training and testing periods. The dataset spans from January 1st, 2024, to February 20th 2024. Where 80% is used for training and the remaining 20% are used for testing purposes.

The data collection process is conducted through a function that fetches hourly price data for a specified symbol within a given period. This function operates by incrementally querying the exchange for "hourly candles" until the entire period's data is retrieved. It then compiles the fetched data into a pandas DataFrame, ensuring the information is structured and time-indexed for further processing.

This method is employed to gather data for multiple cryptocurrencies, such as Bitcoin (BTC), Meme (MEME), Dogecoin (DOGE), and Pepe (PEPE) against the US Dollar Tether (USDT). By collecting diverse datasets, we aim to train the DQN on different market conditions and asset behaviors, testing its generalization capabilities in real-world applications.

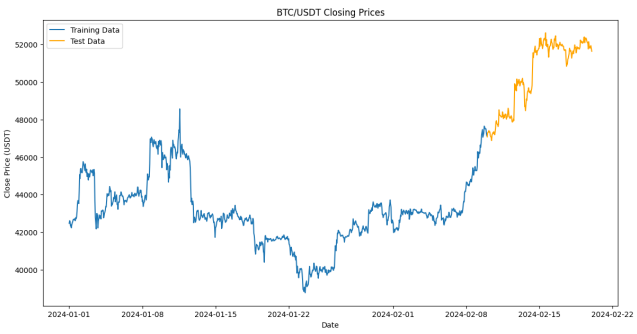
2.3.2 Collected Price Data Analysis

The data presented in Table 1 illustrates a pronounced difference in average volatility between established cryptocurrencies like BTC/USDT and the memecoins selected for our research. Specifically, the volatility of MEME/USDT and PEPE/USDT in both the training and test sets is significantly higher than that of BTC/USDT, a trend that holds true albeit to a lesser extent for DOGE/USDT. This heightened volatility is characteristic of memecoins and suggests a greater degree of price fluctuation over time. Such variability can be advantageous for our comparative analysis when assessing the performance of different trading strategies like the DQN agent, Q-learning, and the double moving average crossover strategy. The pronounced volatility in memecoins provides a more challenging and dynamic environment to test the adaptability and effectiveness of these strategies, which is essential for a thorough evaluation of their potential in real-world trading scenarios. Below we analyse these findings further through visualisation.

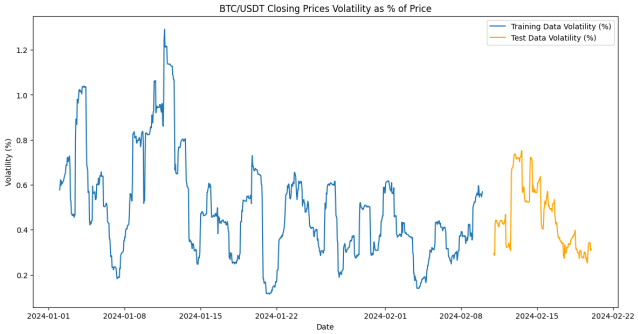
Cryptocurrency	Train Set Volatility (%)	Test Set Volatility (%)
BTC/USDT	0.49	0.46
MEME/USDT	1.16	0.92
DOGE/USDT	0.62	0.52
PEPE/USDT	1.01	1.17

Table 1: Average Volatility for Different Cryptocurrencies

Our analysis extends to the price trends of four selected cryptocurrencies: Bitcoin (BTC), MEME, Dogecoin (DOGE), and PEPE. Despite their varied market positions and community perceptions, one of the observations from our collected dataset is the similarity in overall price trends among these assets as demonstrated in Figure 2a,3a,4a,5a. This parallel movement is indicative of the interconnected nature of the cryptocurrency market, where major market events or shifts in investor sentiment often produce correlated effects across different cryptocurrencies.

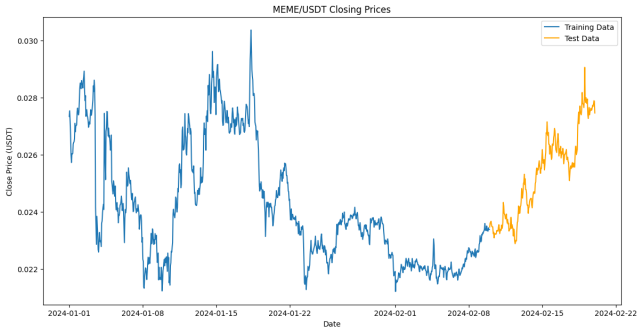


(a) BTC Prices

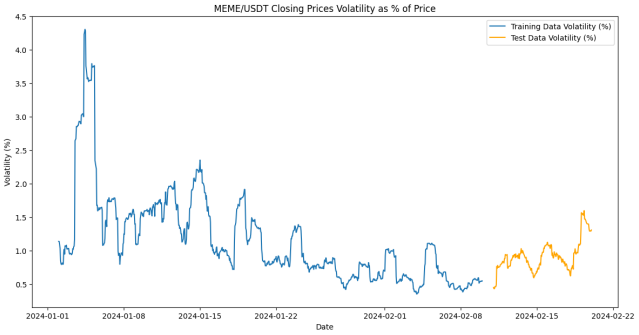


(b) BTC Volatility

Figure 2

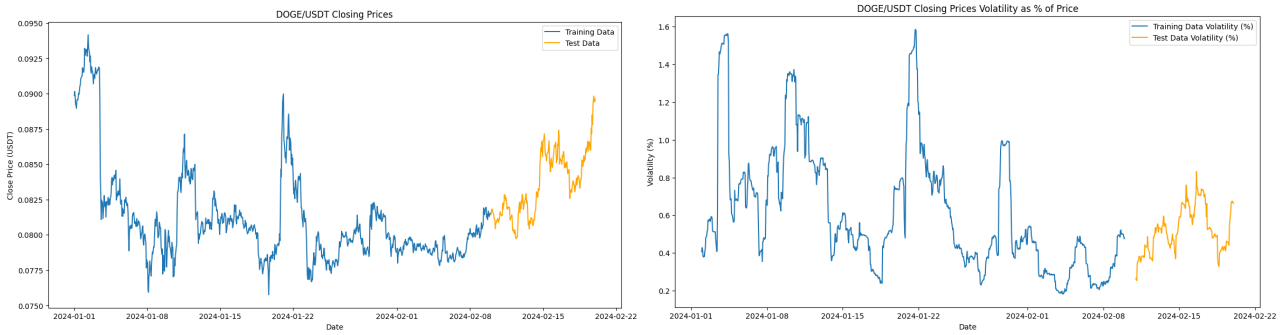


(a) MEME Prices



(b) MEME Volatility

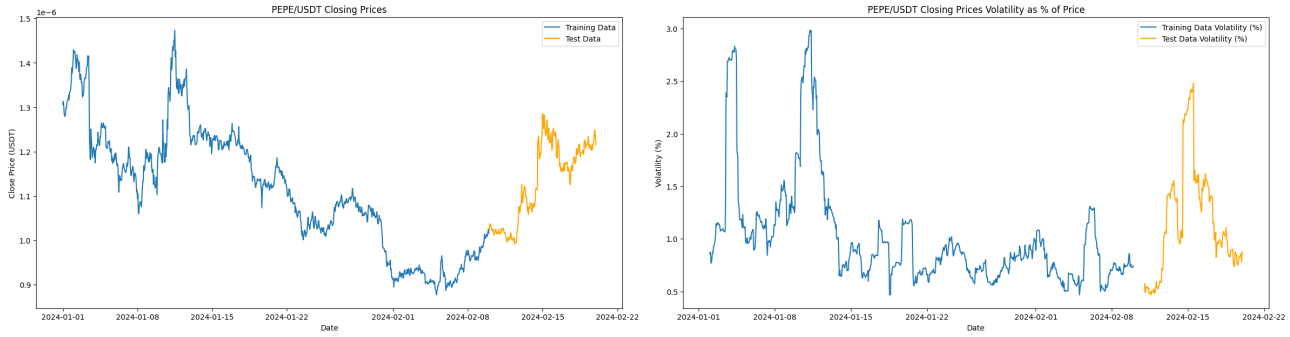
Figure 3



(a) DOGE Prices

(b) DOGE Volatility

Figure 4



(a) PEPE Prices

(b) PEPE Volatility

Figure 5

However, a deeper dive into the price time series plots reveals a distinguishing feature between Bitcoin and the memecoins (MEME, DOGE, and PEPE). While all cryptocurrencies exhibit volatility—a common characteristic of digital assets—the memecoins show a heightened level of price fluctuations compared to Bitcoin. This increased volatility is visually apparent in the time series plots, where MEME, DOGE, and PEPE display more pronounced peaks and troughs compared to the relatively steadier trend observed in Bitcoin's price movement.

To quantify this observation, we calculated the volatility metrics for each cryptocurrency as shown in Table 1. This numerical confirmation underscores the inherent risk and potential for rapid value changes associated with memecoins, contrasting with Bitcoin's relatively more stable investment profile.

2.4 Baseline Models

For our trading agent comparison, we have chosen a traditional **Q-learning algorithm** as the baseline model. This model is pivotal in the reinforcement learning domain for tasks requiring discrete decision-making. The Q-learning agent determines trading actions—buy, hold, or sell—based on a binary state representation, which signifies whether a stock is held. It updates a Q-table, mapping state-action pairs to rewards, guiding the agent towards profit-maximizing strategies.

Q-learning is an intuitive baseline within value-based reinforcement learning, focusing on approximating the optimal action-value function, or Q-function. This function estimates the expected utility of undertaking specific actions from different states, aiding the agent's decision-making process.

The `QRL_Agent` class is defined with parameters such as state size, action size, and learning parameters (ϵ, ϵ_decay, γ, and α). The agent decides on actions based on an epsilon-greedy strategy and updates its Q-table using the temporal-difference learning method as shown in equation (4) above. The `TradingEnvironment` class simulates a trading environment where the agent can execute buy and sell actions based on the observed market prices.

To demonstrate the practical application of this model, we train the QRL agent over multiple epochs (200 in in our case), monitoring its performance and profitability. The training process involves the agent interacting with the trading environment, making decisions, and updating its knowledge base (Q-table) to enhance its trading strategy over time.

In addition to the Q-learning agent, we incorporate a non-ML trading strategy, the **double moving average crossover strategy**, as a second baseline.[7] In our implementation of the double moving average (MA) crossover strategy, we define a function `moving_average_strategy` that computes short-term and long-term moving averages using a specified window size of 10 hours. The short-term moving average is calculated over 5 hours (half the window size), and the long-term moving average is calculated over the full 10 hours. Buy signals are generated when the short-term MA crosses above the long-term MA, and sell signals are initiated when the short-term MA crosses below. The strategy is executed on all 4 cryptocurrency datasets.

This baseline framework allows us to compare the DQN agent against traditional reinforcement learning methods but also contrasts it with conventional trading strategies, thereby showcasing its potential and advanced capabilities in dynamic trading environments.

2.5 Utilised DQN Trading Agent Structure

Algorithm 1 DQN Trader Algorithm

Require: State size, window size, trend, skip, batch size
Initialize state size, window size, action size, memory, etc.
Initialize neural network with TensorFlow
function ACT($state$)
  if random $<$ epsilon **then**
    return random action
  else
    return action with max Q-value from the network
  end if
end function
function GETSTATE(t)
  Compute and return the state based on trend data
end function
function REPLAY($batch\_size$)
  for each sample in mini-batch **do**
    Update Q-values based on the network's predictions
  end for
  Perform one step of the optimization
end function
function TRAIN($iterations$, $checkpoint$, $initial\_money$)
  for $i = 1$ to $iterations$ **do**
    Execute trading actions (buy, sell, hold) based on the policy
    Update the network by replaying experiences
  end for
end function
function TEST($initial\_money$, $trend$)
  Test the trading strategy on new data
end function

The neural network architecture of our DQN Trading Agent is designed to process financial data and make informed trading decisions:

- **Input Layer:** Accepts a state vector $\mathbf{s}$ of size *state_size* (as specified in the agent's initialization), incorporating features from historical price data. This size is critical as it determines how much of the market's past information the agent considers to inform its trading decisions, reflecting the agent's "memory" of the market conditions.
- **Hidden Layers:** Comprises three hidden layers with 512, 256, and 128 neurons, respectively. Each layer l performs a transformation $f_l(\mathbf{x}) = \max(0, \mathbf{W}_l\mathbf{x} + \mathbf{b}_l)$, utilizing ReLU as the activation function. This configuration allows the agent to learn complex non-linear relationships in the data, essential for identifying nuanced trading strategies based on historical price movements.
- **Batch Normalization:** After the second hidden layer, a batch normalization step is applied to normalize the activations, thereby stabilizing and expediting the training process. This technique is particularly beneficial in dynamic environments like financial markets, as it helps the agent adapt more swiftly to new patterns and trends.
- **Output Layer:** Generates Q-value estimates for each possible action a in the action space, resulting in a vector $\mathbf{Q}(s, \cdot)$ of size *action_size* (3 in this case, representing Hold, Buy, or Sell). This layer is critical for the agent's decision-making, as it provides a quantitative evaluation of the expected rewards from each possible action, guiding the agent towards the most profitable decisions.

Strategic aspects of the DQN Trading Agent include:

- **Action Space:** Contains three actions: Hold (0), Buy (1), or Sell (2). This action space dictates the agent's potential decisions at every step, influencing its trading strategy and ability to navigate the market.
- **Experience Replay:** The agent maintains a memory buffer to store and revisit past experiences, a method that reinforces learning from historical data. By re-experiencing past actions and their outcomes, the agent refines its strategy, improving its future decisions.

- **Epsilon-Greedy Strategy:** Adopts a strategy where actions are selected randomly with a probability ϵ (starting at 0.5 and decaying towards a minimum of 0.01) and according to the policy derived from Q with probability $1 - \epsilon$. This approach ensures a balance between exploring new actions and exploiting known strategies, essential for adapting to evolving market conditions.
- **Discount Factor γ:** Employs a discount factor $\gamma = 0.95$ to prioritize immediate over future rewards. This focus is vital in trading, where immediate gains are often more predictable and tangible than distant future rewards, given the market's inherent volatility.
- **Optimizer:** Uses the Adam optimizer with a learning rate of $1e-4$ to minimize the loss function $\mathcal{L} = \mathbb{E}[(y - Q(s,a))^2]$, where y is the target Q-value. This continuous optimization is crucial for enhancing the agent's decision-making capabilities, aiming to maximize trading performance by adapting its Q-value predictions based on observed market outcomes.

By integrating these elements, the DQN Trading Agent is equipped to analyze market data, learn from its interactions with the market, and incrementally improve its trading decisions, all while adapting to new information and market conditions.

2.6 Evaluation Metrics

To rigorously assess the performance of our trading agents, we employ a comprehensive set of evaluation metrics that capture various aspects of trading efficacy. These metrics enable us to quantify the agents' decision-making quality and economic value, providing a holistic view of their performance.

Total Profit (PnL): The primary measure of success for any trading strategy is the total profit it generates. This metric calculates the net profit over the entire trading period, providing a straightforward assessment of the agent's financial performance.

In addition to the total profit, we also present the **Profit and Loss (PnL) over individual trades made**, visualized through graphs. These PnL graphs offer valuable insights into the performance consistency and risk profile of our trading strategies. By analyzing the PnL trends over trades, we can identify periods of significant gains or losses, assess the strategy's ability to capitalize on market opportunities, and evaluate its resilience during market downturns. This graphical representation aids in understanding the strategy's profit-generating mechanisms and its potential for sustainable trading success.

Sharpe Ratio: To evaluate the risk-adjusted return of the trading agents, we utilize the Sharpe ratio. This metric offers insight into the profitability of the agent relative to the risk incurred, allowing us to understand the returns' consistency and stability.

$$\text{Sharpe Ratio} = \frac{\bar{R} - r_f}{\sigma}, \tag{7}$$

where

- $\bar{R}$ is the average return calculated as np.mean(R), with $R = \frac{\text{np.diff(pnl\_data)}}{\text{pnl\_data[:-1]}}$ representing the returns computed from the profit and loss data (`pnl_data`).
- r_f is the risk-free rate, corresponding to the `risk_free_rate` in the code.
- σ is the standard deviation of the returns, calculated as np.std(R).
- If $\sigma = 0$, the Sharpe Ratio is defined to be 0 to avoid division by zero, as handled in the code.

The calculated Sharpe Ratio provides a measure of the risk-adjusted return of an investment.

Maximum Drawdown: Understanding the risk associated with a trading strategy is crucial, and the maximum drawdown helps in this regard. It measures the largest single drop from peak to bottom in the strategy PnL value, indicating the potential risk of loss the strategy entails.

In our analysis, the maximum drawdown is calculated using the cumulative profit and loss (PnL) data for each cryptocurrency as follows:

1. The cumulative PnL data is first transformed into a numpy array for efficient numerical computation.
2. We then compute a running maximum of this array, which tracks the highest value the cumulative PnL has reached at any point in the data series.
3. Subsequently, the drawdown is determined at each time point as the absolute drop from the running maximum, computed by subtracting the cumulative PnL from the running maximum.
4. To circumvent division by zero in the case where the running maximum is zero, we replace such values with the smallest positive number greater than zero that is representable in the data type of our array.

5. The drawdown is then expressed in percentage terms as the ratio of the absolute drawdown to the running maximum, multiplied by 100.
6. Finally, we identify the maximum drawdown percentage as the largest value obtained from these drawdown percentages, representing the most substantial relative decline over the trading period.

The maximum drawdown is presented as a percentage, which illustrates the severity of the drawdown relative to the strategy's PnL at its peak before the drawdown occurred. This percentage is a unit-less measure, allowing for comparison across different scales and values of PnL data.

This methodology ensures an accurate representation of the maximum drawdown even in datasets where the PnL does not surpass the initial value, thereby providing a comprehensive metric of the strategy's downside risk.

By employing these diverse metrics, we aim to capture a comprehensive and nuanced picture of the trading agents' performance. This evaluation allows us to discern not only which agent is the most profitable but also which one offers a favorable balance between return and risk, ensuring a robust comparison and selection of the optimal trading strategy.

3 Results Discussion and Analysis

3.1 Calculated Metrics

The performance metrics presented in Table 2 offer a thorough assessment of three distinct trading strategies across the selected cryptocurrency assets. All 3 strategies were trained and tested on the same 4 specific cryptocurrency datasets for fairness.

Strategy	Maximum Drawdown (%)	Final Cumulative PnL (USDT)	Sharpe Ratio
DQN Trading Agent	BTC: -19.20	2268	0.449
	MEME: -31.78	0.000737	0.411
	DOGE: -51.12	0.00139	0.701
	PEPE: -40.47	4.32e-08	0.583
Double Moving Average Crossover	BTC: -70.78	787.60	-0.067
	MEME: -491.01	-0.4625	0.193
	DOGE: -220.2	-0.4722	0.305
	PEPE: -875.46	-4.660	0.082
Q Learning Trading Agent	BTC: -106.29	4511.30	-0.097
	MEME: -692.26	-0.002487	0.005
	DOGE: -768.29	0.00337	0.097
	PEPE: -160.29	1.6515e-07	0.103

Table 2: Comparison of Trading Strategies

The DQN Trading Agent showcases commendable risk management with Maximum Drawdowns ranging from **19.20%** for BTC to **51.12%** for DOGE, which are substantially lower than those of the other strategies. This conservative approach is especially notable when contrasted with the Double Moving Average Crossover strategy, which reports an excessive Maximum Drawdown of **875.46%** for PEPE. The Q Learning Trading Agent also exhibits significant risk exposure, with drawdowns as high as **768.293%** for DOGE, which is not present in DQN.

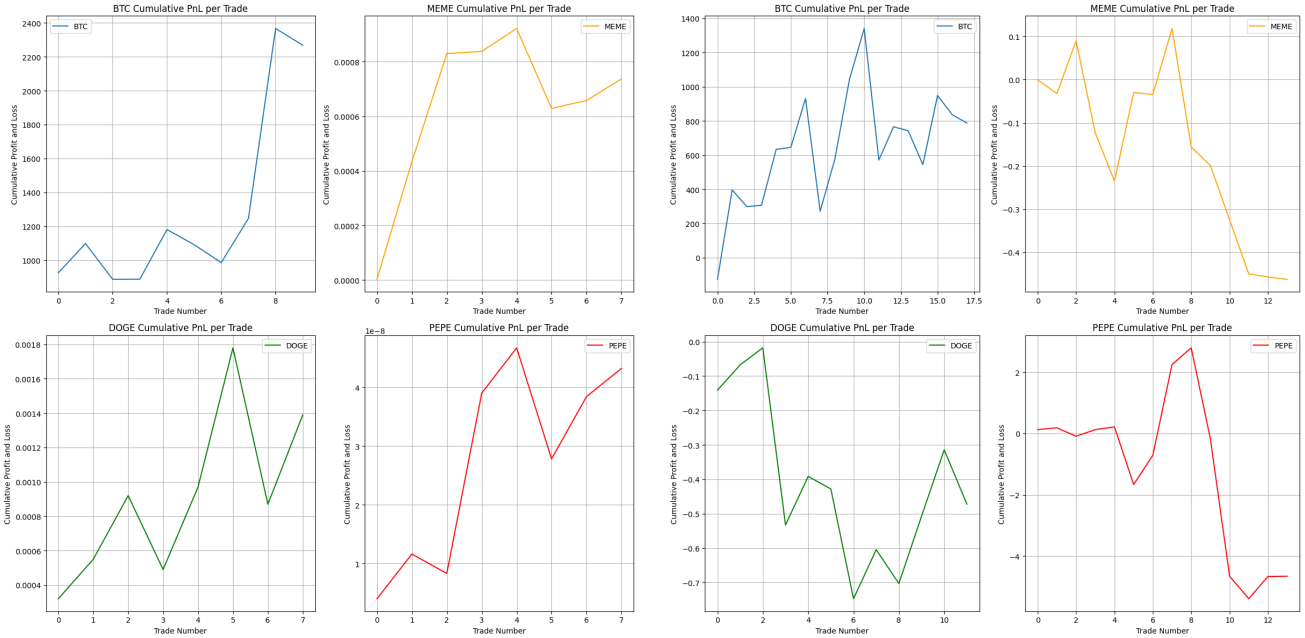
In terms of Final Cumulative PnL, the DQN Trading Agent outperforms with positive returns across all assets, particularly a significant **2268** for BTC, indicating its consistent profitability and adeptness in leveraging market movements. On the contrary, the Double Moving Average Crossover strategy shows a negative Final Cumulative PnL for MEME, DOGE, and PEPE, and the Q Learning Trading Agent, while achieving a high return for BTC, displays variability with negative outcomes for MEME. Showcasing the inconsistency in performance in our two selected baseline strategies.

The Sharpe Ratios paint a similar picture; the DQN Trading Agent achieves ratios from **0.411** for MEME to **0.701** for DOGE, suggesting superior risk-adjusted returns. Meanwhile, the Double Moving Average Crossover and Q Learning Trading Agent strategies often settle with negative or marginal Sharpe Ratios, underscoring a less favorable return per unit of risk.

Overall, the DQN Trading Agent proves to be the most effective and stable strategy, striking an optimal balance between managing risks and securing rewards. Its performance positions it as a promising tool for navigating the complexities of cryptocurrency trading.

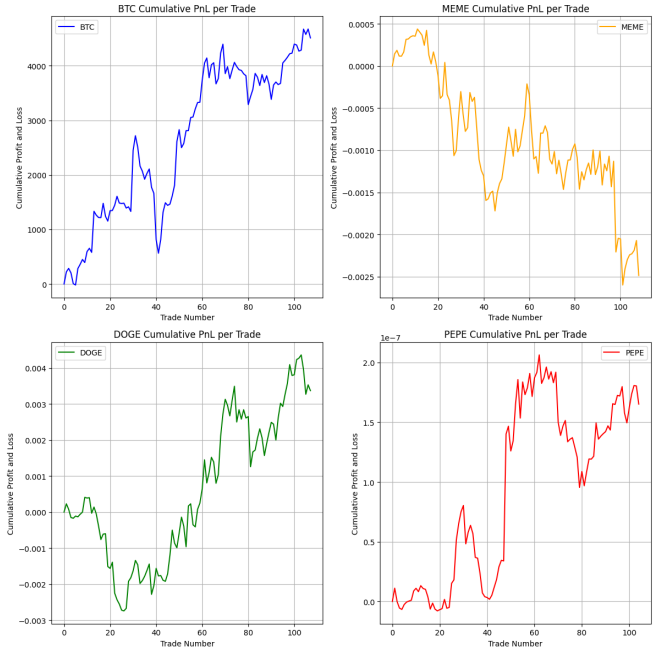
3.2 Profit and Loss per Trade Made - visualisation

From the graphs generated during the testing phase of the utilised trading agents and the MA crossover trading strategy we can see that the DQN strategy was more conservative in terms of carrying out new trades and has only completed 9 trades in Bitcoin compared to 17 of double MA crossover and over 100 for the Q Learning agent.



(a) DQN Trading Agent PnL Results

(b) Double MA Crossover PnL Results



(c) Q Learning Trading Agent PnL Results

Figure 6: Combined PnL Results

In the testing phase, the DQN Trading Agent's actions led to 9 Bitcoin trades, fewer than the 17 of the Double Moving Average Crossover and significantly less than the 100+ trades by the Q Learning Trading Agent in the same time span. This difference in trading frequency is crucial for understanding each strategy's efficiency and as evidenced by the performance data in Table 2, the more conservative and cautious trading strategy of the DQN strategy has led to a better generalised performance over the testing period.

The DQN Agent's lower trade frequency is indicative of a more selective approach to trade execution, which is supported by the evidence of its superior risk metrics (Maximum Drawdown) and efficiency (Sharpe Ratio). This evidences that a lesser number of trades, if well-chosen, can lead to better performance outcomes compared to strategies that trade higher volume of

the same asset, underscoring the importance of quality in trade selection. The DQN Agent's strategy thus emerges as more effective in optimizing the risk-return trade-off, a key objective in financial trading strategies.

Conclusion and Outlook for Future Innovations

The findings detailed in our report provide evidence of the efficacy of utilizing neural networks, specifically through the lens of the DQN Trading Agent, in algorithmic trading. The DQN's robust performance in both the relatively stable Bitcoin market and the notably more volatile memecoin environments highlights its adaptability and potential in diverse market conditions. These outcomes serve as a signal for the need for further investigation and application of neural network-based models, like DQN, in various facets of algorithmic trading to harness their full potential.

The evolving landscape of algorithmic trading, increasingly dominated by machine learning and AI, calls for continuous innovation and exploration of new methodologies. One such promising avenue is the integration of ensemble models and optimization techniques, as detailed in the study by Loh et al. [5] Their research, focusing on Forex trading, leverages an ensembling architecture that incorporates State of the Art (SOTA) machine learning strategies, optimized using a Genetic Algorithm (GA) to maximize profits. This approach not only enhances predictive accuracy but also introduces a robust framework for managing risk and evaluating system performance under varying market conditions.

Inspired by these advancements, future research could explore the synergy between deep reinforcement learning, as utilized in our DQN Trading Agent, the ensemble models and genetic algorithms coupled with optimization techniques highlighted by Loh et al. Such integrative approaches could further refine trading strategies, enhance adaptability to market dynamics, and improve overall profitability and risk management. Moreover, extending these methodologies to different financial markets and instruments can provide a broader understanding of their effectiveness and scalability.

In conclusion, the momentum behind machine learning algorithms in trading, especially in the face of increased market volatility and changing market environments, is poised to grow stronger. Our report adds to the body of evidence supporting this trajectory, advocating for a deeper dive into the neural network-based models and their potential to revolutionize algorithmic trading.

References

[1] *CCXT: CryptoCurrency eXchange Trading Library* [2024], `https://github.com/ccxt/ccxt`. Accessed: 02/03/24.

[2] Chakole, J. B., Kolhe, M. S., Mahapurush, G. D., Yadav, A. and Kurhekar, M. P. [2021], 'A q-learning agent for automated trading in equity stock markets', *Expert Systems with Applications* **163**, 113761.
URL: *https://www.sciencedirect.com/science/article/pii/S0957417420305856*

[3] Choudhary, A. [2023], 'A Hands-On Introduction to Deep Q-Learning using OpenAI Gym in Python', `https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/`. Accessed: 28/02/24.

[4] Li, C. and Yang, H. [2022], 'Will memecoins' surge trigger a crypto crash? evidence from the connectedness between leading cryptocurrencies and memecoins', *Finance Research Letters* **50**, 103191.
URL: *https://www.sciencedirect.com/science/article/pii/S154461232200397X*

[5] Loh, L. K. Y., Kueh, H. K., Parikh, N. J., Chan, H., Ho, N. J. H. and Chua, M. C. H. [2022], 'An ensembling architecture incorporating machine learning models and genetic algorithm optimization for forex trading', *FinTech* .
URL: *https://api.semanticscholar.org/CorpusID:247808050*

[6] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. and Riedmiller, M. A. [2013], 'Playing atari with deep reinforcement learning', *ArXiv* **abs/1312.5602**.
URL: *https://api.semanticscholar.org/CorpusID:15238391*

[7] TrendSpider [n.d.], 'Moving average crossover strategies', `https://trendspider.com/learning-center/moving-average-crossover-strategies/`. Accessed: 10/03/24.

[8] Zhang, Z., Zohren, S. and Roberts, S. J. [2019], Deep reinforcement learning for trading, *in* 'The Journal of Financial Data Science'.
URL: *https://api.semanticscholar.org/CorpusID:208248040*