Hierarchical Multi-Agent Reinforcement Learning Approaches to Algorithmic Trading

Shomit Basu
University College London
shomit.basu.16@ucl.ac.uk

Oli Bridge
University College London
oliver.bridge.23@ucl.ac.uk

Isaac Watson
University College London
isaac.watson.23@ucl.ac.uk

Abstract

This paper explores multi-agent reinforcement learning (MARL) for algorithmic trading, addressing limitations of single-agent approaches. Extending a hierarchical MARL system to longer timeframes, this work evaluates the performance of cooperating agents, which are each experts on their own respective timeframe, across a range of assets. Our MARL framework is compared to individual single-agent reinforcement learning (RL) traders, as well as a series of rule-based benchmark strategies. For a 3-agent system, it was found that single- and multi-agent RL approaches significantly outperform rule-based methods, and the MARL framework performs better on stock data with clearer trends in the trading horizon considered. For Forex data, the MARL framework underperforms compared to a single-agent implementation, which can be attributed to market capitilisation of the asset class and its impact on trend development.

1 Introduction

Trading in financial markets continues to receive notable attention from researchers, with diverse methodologies emerging to navigate their complexities. Recent years have witnessed a shift towards algorithmic trading, where financial institutions leverage pre-defined rules and strategies to automate trades. This approach offers advantages such as faster execution and reduced human bias in decision-making. However, these methods remain reliant on hand-crafted features and algorithms, requiring extensive domain knowledge for effective implementation.

The dynamic nature of financial markets necessitates a framework that can adapt to evolving conditions. Single-agent reinforcement learning (RL) provides a strong foundation for algorithmic trading due to the following characteristics:

- **Sequential Decision Alignment:** Financial markets involve a series of buy, sell, or hold decisions. RL excels in optimizing sequential decision-making processes where an agent interacts with its environment (the market) and maximizes a reward signal (profit) based on learned experience.
- **Comprehensive State Representation:** RL effectively accommodates the complexities of market data, facilitating rich state representation that may include historical prices, technical indicators, and market sentiment.
- **Discrete Action Mapping:** The well-defined actions of algorithmic trading (buy, sell, hold) directly map to the discrete action spaces readily handled by RL frameworks.

- **Iterative Learning:** Single-agent RL mirrors the learning mechanisms of human traders. RL agents can analyze past trading interactions, continually refining their decision-making strategies over time.

While single-agent RL offers a useful foundation, it has limitations for financial market applications. Its focus on a single agent prevents it from fully considering the complexities of a market driven by many actors with differing strategies. This limits potential diversification of trading approaches and increases the risk of overfitting. The agent might learn to perform well within a limited dataset but fail to adapt quickly to the dynamic, ever-shifting nature of real-world financial markets.

Multi-agent systems (MAS) provide a promising framework to address the limitations of single-agent RL in algorithmic trading. MAS allow for multiple autonomous agents to interact, mirroring the diverse participants in real-world markets [1]. This approach has shown potential in fields like traffic simulation [2], smart grids [3], and robotics [4]. Applying multi-agent reinforcement learning (MARL) to finance can lead to agents that learn and adapt through these interactions. This offers potential for collective intelligence, incorporating feedback loops between agent actions and market dynamics, and facilitating risk diversification through multiple trading strategies. An example of a strong MARL framework proposed for trading is that of Shavandi and Khedmati (2022) [5], which involves a system of agents trained non-episodically to be experts on a different timeframe, which pass information from high to low timeframes.

This study investigates the potential of multi-agent reinforcement learning (MARL) to address the limitations of traditional single-agent frameworks in algorithmic trading. By building upon the hierarchical MARL framework in [5], we develop an N-agent system[1], where each agent is trained with an episodic DQN algorithm as described by [6]; such an episodic framework allows for the use of longer-term timeframes (daily, weekly etc.) rather than being restricted to high-latency timeframes with a very large number of data points.

Our key contributions include:

- **Hierarchical Cooperation:** We examine the effectiveness of employing hierarchical agent cooperation in the context of low-latency stock data for a 3-agent system[2].
- **Performance in Dynamic Environments:** We assess the MARL framework's performance against individual single-agents trained individually on their respective timeframes, across a series of financial assets.

Our objective is to advance the understanding of MARL for algorithmic trading, highlighting its potential advantages and domain-specific challenges. By extending to N-agents and longer time frames, we bridge the gap between theoretical MARL models and real-world financial trading strategies employed by practitioners.

This paper is organised as follows. Section 2 provides a brief literature review of related work in the area of single- and multi-agent RL approaches to algorithmic trading. Section 3 provides an explanation of our methods and experimental setup, and Section 4 includes a discussion of our findings. Section 5 concludes our work and gives potential directions for further research.

2 Literature Review

Seminal work by Moody *et al.* (1998) [7] laid the foundation for single agent RL in financial trading by employing Recurrent Reinforcement Learning (RRL) to directly optimize utility functions and evaluate the performance of financial investments. This was followed up by a Q-learning [8] approach to trading systems on the S&P 500 [9]; the study found that Q-learning significantly underperformed RRL and suffered from the curse of dimensionality. Applications of RRL have been further investigated for high frequency trading in currency markets [10].

In parallel, researchers were also investigating value-based RL in financial trading. Gao *et al.* (2000) [11] introduced one of these applications by optimizing the Sharpe ratio with Q-learning. A majority of RL-based financial trading research studies have focused on signalling and only a few studies have

[1]Details of our code repository are provided in the Appendix.

[2]For computational reasons, $N = 3$ was chosen for our experiments.

addressed the execution of the trading decisions. One of these studies belongs to Nevmyvaka *et al.* (2006) [12], who proposed a value-based RL algorithm to execute the trading orders. The input data for their algorithm was taken from NASDAQ order books, which contained millisecond order data for one and a half years.

The advent of deep reinforcement learning (DRL) has significantly accelerated the exploration of value-based RL in financial trading. Liu *et al.* (2022) [13] showcased DRL, and in particular the DDPG algorithm [14], as an effective solution for portfolio allocation. Taghian *et al.* (2020) [6] and Kabbani *et al.* (2022) [15] established impressive performance in single asset trading, implementing the DQN [16] and TD3 [17] algorithms, respectively. However, as mentioned above, single-agent RL, deep or not, has it's limitations when applied to financial markets.

To address portfolio management limitations, researchers have explored Multi-Agent Reinforcement Learning (MARL). Lee *et al.* (2020) [18] proposed a cooperative system where agents prioritize diversification alongside maximizing returns. Huang *et al.* (2024) [19] introduced a novel MARL algorithm with sub-agents leveraging time-series features and a final agent utilizing collective intelligence to make informed decisions. A similar framework is implemented by Shavandi and Khedmati (2022) [5], that utilises a hierarchy of sub-agents, each trained on a specific timeframe. The idea is inspired by the fractal market hypothesis which states the statistical distributions of price variations are similar over different scales in time [20]. This hierarchical agent framework is one we look to develop in our work.

3 Methods

3.1 Deep Reinforcement Learning

In reinforcement learning, the action value function $Q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ for a given policy π is defined as the expected return for taking action a in state s and following policy π thereafter [21]:

$$Q^\pi(s,a) = \mathbb{E}\left[G_t | S_t = s, A_t = a, \pi\right] \tag{1}$$

where the return G_t is the sum of discounted rewards R from time t:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2}$$

The action value function satisfies the Bellman equation:

$$Q^\pi(s,a) = \mathbb{E}\left[R_{t+1} + \gamma Q^\pi(S_{t+1}, A_{t+1}) \middle| S_t = s, A_t = a, \pi\right] \tag{3}$$

The optimal action value function $Q^* : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the maximum action value function over all possible policies; it satisfies the Bellman optimality equation, and for a deterministic environment (where each state/action pair (s, a) maps to exactly one new state s') it can be written as

$$Q^*(s,a) = r(s,a) + \gamma \max_{a'} Q^*(s', a') \tag{4}$$

where $r(s, a)$ is the reward for taking action a in state s. In Deep Q-Learning (DQN) [16], a single agent learns an estimate of the optimal action value function using a neural network $Q(s, a; \theta) \approx Q^*(s, a)$, called a *Q-network*. A Q-network can be trained by minimising the temporal difference error:

$$\delta = \underbrace{r + \gamma \max_{a'} Q(s', a'; \theta^-)}_{y} - Q(s, a; \theta) \tag{5}$$

where y is the target value, θ are the weights of the Q-network, and θ^- are the weights of a *target network* which are updated every C timesteps by copying the weights of the Q-network to stabilise learning. The loss for the DQN algorithm can be computed by taking the expectation of some loss function $\mathcal{L} : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ on the temporal difference error over a batch of experiences drawn from a *replay buffer* $\mathcal{D}$ [22]:

$$\begin{aligned} L(\theta) &= \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\left[\mathcal{L}(\delta)\right] \\ &= \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\left[\mathcal{L}\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i)\right)\right] \end{aligned} \tag{6}$$

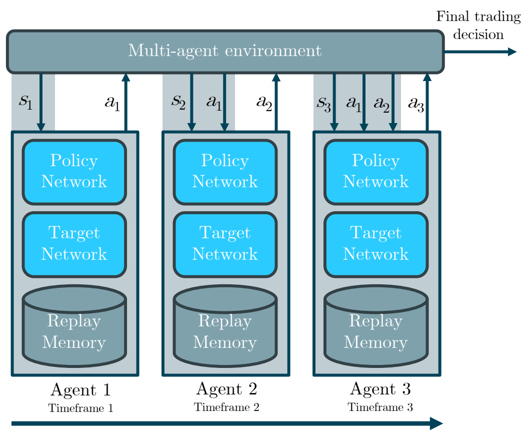


Figure 1: Schematic diagram of the multi-agent framework used for single-stock trading with 3 agents. Actions from each agent are passed down to lower timeframe agents in a hierarchical fashion.

In the original DQN paper [16], a squared error loss function was used, but in this work a Huber (smooth L_1) loss function [23] was implemented, which penalises outliers less strongly:

$$\mathcal{L}(\delta) = \begin{cases} \frac{1}{2}\delta^2 & \text{if } |\delta| \leq 1 \\ |\delta| - \frac{1}{2} & \text{otherwise} \end{cases} \tag{7}$$

Performing stochastic gradient descent on this loss function with respect to the Q-network weights θ allows the agent to learn an optimal policy. This procedure is combined with an ϵ-greedy policy[3] to form the popular DQN algorithm. The Q-network architectures used in this work are based on that of Taghian *et al.* (2020) [6], which use multilayer perceptrons (MLPs) with batch normalisation as a regulariser to prevent overfitting to the training data.

3.2 Single-Agent Framework

A Markov Decision Process (MDP) [21] is defined as the tuple $\langle \mathcal{S}, \mathcal{A}, \mathbb{P}, R, \gamma \rangle$ where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathbb{P} : \mathcal{S} \times \mathcal{A} \to [0, 1]$ are the transition probabilities, $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function, and γ is the discount factor. A single agent acting within an environment described by an MDP wishes to maximise its discounted return G_t. This problem formulation has been used extensively in the past to model an agent trading on a financial market [6][10][12]. In this work, we are interested in single-stock trading, so we formulate the MDP as follows:

- State space $\mathcal{S}$: the agent will observe the open, high, low and close prices of the chosen asset for its given timeframe i.e. $s = (O, H, L, C) \in \mathbb{R}^4$.
- Action space $\mathcal{A}$: the agent will be able to sell, hold or buy stock i.e. $\mathcal{A} = \{-1, 0, +1\}$. If an agent issues a buy signal, it buys up as much stock as it can with its available balance, and a sell signal will fully liquidate the agent's position and sell all the stock it owns. This regime was chosen to simplify the action space.
- Transition probabilities $\mathbb{P}$: these are determined by the market dynamics i.e. the next state in the time-series data.
- Rewards R: the reward function was based on that used in [6], and computes the percentage change in the close price after an n-step horizon. The reward at timestep t is given by

$$R_t = \begin{cases} \frac{C_t - C_{t+n}}{C_{t+n}} \times 100 & a_t = -1 \text{ (sell)} \\ 0 & a_t = 0 \text{ (hold)} \\ \frac{C_{t+n} - C_t}{C_t} \times 100 & a_t = +1 \text{ (buy)} \end{cases} \tag{8}$$

For this single-agent framework, we employ the DQN algorithm of [16] and [6], described in Section 3.1, to approximately learn the optimal value function for the trading problem.

[3]Details of the agents' ϵ-greedy policy are given in the Appendix.

Algorithm 1 Multi-Agent Hierarchical DQN

```
for agent $i = 1, N$ do
    Initialise replay memory $\mathcal{D}_i$ to capacity $K$
    Initialise policy action-value function $Q_i$ with random weights $\theta_i$
    Initialise target action-value function $\hat{Q}_i$ with weights $\theta_i^- = \theta_i$
end for
for episode $= 1, M$ do
    Initialise sequence $s_1^i = (O_1, H_1, L_1, C_1, \mathcal{A}_1^{<i})_{\text{tf}_i}$ for each agent $i$
    for $t = 1, T$ do
        for agent $i = 1, N$ do
            with probability $\epsilon(t)$ select a random action $a_t^i$
            otherwise select $a_t^i = \arg\max_a Q_i(s_t^i, a; \theta_i)$
        end for
        Lowest timeframe agent executes action $a_t^N$ in environment
        for agent $i = 1, N$ do
            Observe reward $r_t^i$ and next state $s_{t+1}^i = (O_{t+1}, H_{t+1}, L_{t+1}, C_{t+1}, \mathcal{A}_{t+1}^{<i})_{\text{tf}_i}$
            Store transition $(s_t^i, a_t^i, r_t^i, s_{t+1}^i)$ in $\mathcal{D}_i$
            Sample random minibatch of transitions $(s_j^i, a_j^i, r_j^i, s_{j+1}^i)$ from $\mathcal{D}_i$
            Set $y_j^i = \begin{cases} r_j^i & \text{if episode terminates at step } j+1 \\ r_j^i + \gamma \max_{a'} \hat{Q}_i(s_{j+1}^i, a'; \theta_i^-) & \text{otherwise} \end{cases}$
            Perform a gradient descent step on $\mathcal{L}(y_j^i - Q_i(s_j^i, a_j^i; \theta_i))$ with respect to parameters $\theta_i$
            Every $C$ steps reset $\theta_i^- = \theta_i$
        end for
    end for
end for
```

3.3 Multi-Agent Framework

A multi-agent extension of an MDP is known as a Markov Game [24], which is defined by the tuple $\langle N, \mathcal{S}, \mathcal{A}, \mathbb{P}, R, \gamma \rangle$. In this case, we have N agents, where each agent i has its own state space $\mathcal{S}_i$ and action space $\mathcal{A}_i$. The transition probabilities and rewards depend on all agents' actions, i.e. $\mathbb{P} : \mathcal{S} \times \mathcal{A} \to [0, 1]$ and $R_i : \mathcal{S} \times \mathcal{A} \times \mathcal{S}$, where $\mathcal{S} = \mathcal{S}_1 \times \ldots \times \mathcal{S}_N$ and $\mathcal{A} = \mathcal{A}_1 \times \ldots \times \mathcal{A}_N$. In this work, our multi-agent framework is similar to the single-agent case, with some modifications. The most significant change is the state space of each agent, which facilitates the exposure to different timeframes and the hierarchical passing of information from one agent to the next.

Consider a system of N agents and N different timeframes $\{\text{tf}_1, \ldots, \text{tf}_N\}$. At timestep t, agent i receives a state s_t^i which consists of an open, high, low and close price for timeframe tf$_i$, and the actions chosen by all agents in *higher* timeframes. The state vector for agent i is given by

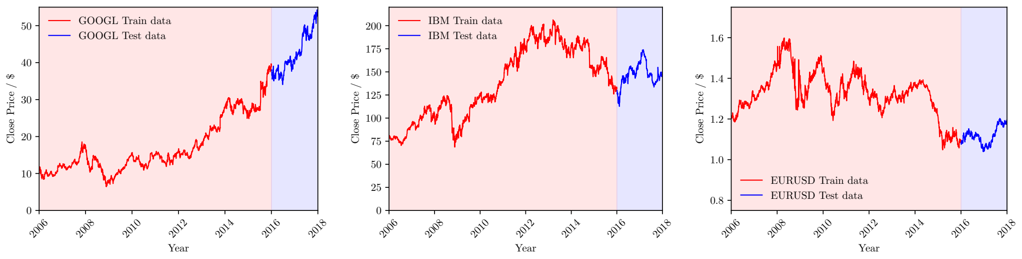
$$s_t^i = (O_t, H_t, L_t, C_t, \mathcal{A}_t^{<i})_{\text{tf}_i} \tag{9}$$

where $\mathcal{A}_t^{<i}$ is the set of actions at timestep t from all higher timeframe agents:

$$\mathcal{A}_t^{<i} = \{a_t^j | j < i\} \tag{10}$$

For example, consider timeframe tf$_i$ being a 5-day period; the OHLC prices at timestep (day) t are calculated as the open, high, low and close prices for the last 5 trading days. The lowest timeframe agent, which we choose to be a daily agent, sees all actions from the other agents and makes the final trading decision, which is fed into the trading environment. Each agent is then issued a reward based on its own action, using Equation 8. A schematic diagram of an example 3-agent system is shown in Figure 1. Each agent has its own policy network, target network and replay memory. Based on the rewards they receive, using a multi-agent version of the DQN algorithm shown in Algorithm 1, the agents become experts of their respective timeframes, and learn to cooperate with each other to enhance their return.

This hierarchical approach has some key differences to that of Shavandi and Khedmati [5]. Firstly, a single set of OHLC prices at each timestep was chosen as the input features for each agent, rather than a series of consecutive values. This is because it lowers the input dimensionality of the Q-networks,



(a) Google (GOOGL) (b) IBM (IBM) (c) Euro - US Dollar (EURUSD)

Figure 2: Train and test sets for the single-stock trading experiments.

and it has also been found that a single OHLC is a sufficient set of features for a DQN agent to learn good trading decisions [6]. Secondly, our approach involves an episodic training procedure, which means one isn't constricted to timeframes that have a very large number of data points (high-latency timeframes).

3.4 Experiments

In order to test our hierarchical multi-agent episodic framework, experiments were conducted on three different single assets; Google, IBM, and the Euro/US Dollar exchange rate. For the train/test split, a 10 year training period from 01/01/2006 - 31/12/2015, and a 2 year testing period from 01/01/2016 - 31/12/2017 was chosen. Close prices for these assets over this time period are displayed in Figure 2. These assets were chosen in order to test the framework on a wide range of asset behaviour, ranging from stocks with more clear long-term trends (Google and IBM) to assets that see a lot of fluctuation over time (EUR/USD).

For each asset, the multi-agent framework was trained with 3 agents and 5-day, 3-day, and 1-day timeframes. Results are compared to the 3 individual agents acting on their own, receiving features from their respective timeframe. All RL-based methods were trained for 100 episodes. The single- and multi-agent deep RL frameworks were also compared to two rule-based strategies:

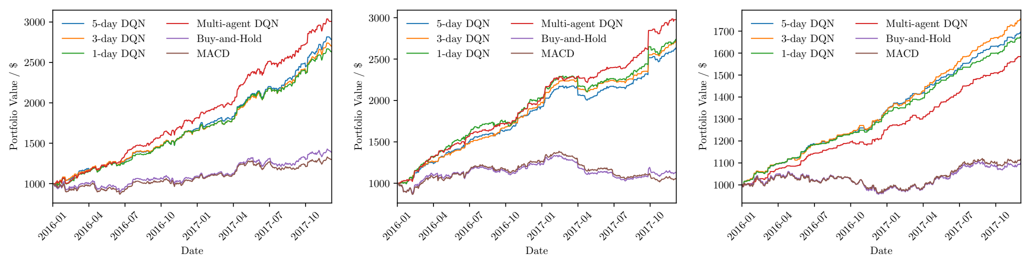
- Buy-and-Hold: Trading agent buys up the stock at the beginning of the testing period and holds all the way to the end, where it liquidates.
- Moving Average Convergence Divergence (MACD): Trading agent issues buy and sell signals based on the relationship between a series of exponential moving averages (EMA) from the price data. Default settings were used [25].

In all cases, the trading agents were given a starting balance of \$1000 at the beginning of the testing period. Model hyperparameters were chosen empirically based on those deemed to be effective for the single-agent DQN trading framework of [6]. We tried to keep as true to the literature as possible to avoid overfitting. Details of the model hyperparameters are provided in the Appendix.

4 Results and Discussion

Plots of portfolio value for experiments on the three different assets are displayed in Figure 3. Additionally, the different methods' performances are summarised by a series of metrics in Table 1. The cumulative return provides an overview of a strategy's profitability over the testing period, the Sharpe ratio measures the performance of the strategy in terms of risk-adjusted return, and the maximum drawdown is an indicator of downside risk over the testing period. Details of how these metrics are calculated are provided in the Appendix.

It is clear from all experiments in Figure 3 that the DRL trading approaches vastly outperform our selection of standard rule-based alternatives. Such rule-based benchmarks, particularly MACD, are typically used as indicators to assist a human trader in making their own decisions on the market, therefore they appear to struggle when used as a sole trading strategy. By contrast, the DRL methods show strong performance by effectively learning how to make good trading decisions based on their input features. Figure 4 shows the trading actions taken by the multi-agent framework for the first



(a) Google (GOOGL) (b) IBM (IBM) (c) Euro - US Dollar (EURUSD)

Figure 3: Portfolio values for the single-stock trading experiments.

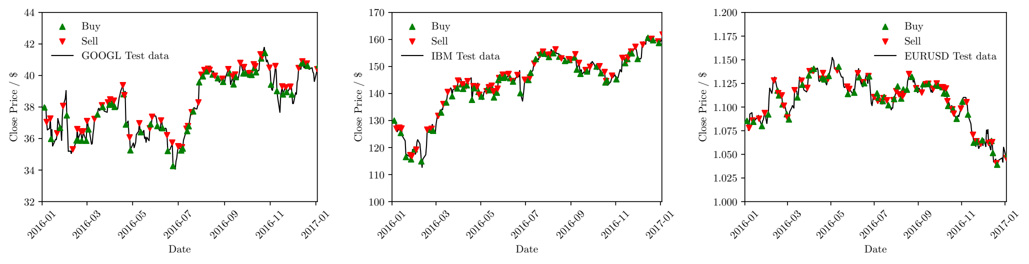
		5-day DQN	3-day DQN	1-day DQN	Multi DQN	B&H	MACD
GOOGL	Cum. Return	178.1%	169.9%	162.8%	**200.8%**	38.7%	29.7%
	Sharpe Ratio	4.96	4.67	4.61	**5.56**	1.00	0.80
	Max. Drawdown	**5.01%**	**5.01%**	6.45%	6.45%	13.53%	13.92%
IBM	Cum. Return	163.4%	171.7%	173.8%	**197.6%**	12.9%	5.4%
	Sharpe Ratio	4.97	5.53	4.79	**6.03**	0.35	0.16
	Max. Drawdown	7.99%	6.51%	8.43%	**3.46%**	23.22%	26.75%
EURUSD	Cum. Return	69.5%	**75.6%**	67.5%	58.4%	10.02%	11.9%
	Sharpe Ratio	6.40	**6.61**	6.50	5.70	0.61	0.73
	Max. Drawdown	**1.05%**	1.64%	0.75%	1.65%	9.84%	8.94%

Table 1: Performance metrics for the single-stock trading experiments.

year of the testing period; it is clear that in all cases the framework has learned to make good trades from its input features, which allows it to be so effective compared to rule-based approaches.

For the stock data, Google and IBM, the multi-agent framework achieved a higher cumulative return and Sharpe ratio than the corresponding individual single-agent traders. For IBM, the multi-agent framework also produced the lowest maximum drawdown, indicating a lower downside risk in its investing strategy. From Figure 2, these assets exhibit clearer trends on the horizon considered for our agents, as well as daily fluctuations; the multi-agent framework's superior performance here supports the hypothesis that collective intelligence is more effective, and that more detailed market dynamics are being captured resulting in better trading decisions. The lower Sharpe ratios for the multi-agent trader also shows that it is not only obtaining higher returns, but at lower risk as well.

By contrast, for the experiment performed on Forex data, EUR/USD, the single-agent traders outperformed our multi-agent trader. From Figure 2, we can see that this asset is more sporadically fluctuating, with less of a noticeable trend on the timeframes that our agents are operating on. This asset's high volatility creates opportunities for single agents to capitalize on short-term market



(a) Google (GOOGL) (b) IBM (IBM) (c) Euro - US Dollar (EURUSD)

Figure 4: Visualisation of the trading decisions made by the multi-agent framework for each of the experiments in the first year of the testing period.

fluctuations, potentially explaining its outperformance against the multi-agent framework. Shavandi and Khedmati's hierarchical study [5] found that their multi-agent framework was superior for high-latency timeframes on Forex data, which offers another perspective.

The Forex market's massive liquidity ($7.5 trillion daily volume), as highlighted by the Bank for International Settlements' Triennial Central Bank Survey in April 2022, can lead to slower trend development compared to equities [26]. In future work, our hierarchical model could better capture these dynamics by including both very short-term agents to capitalize on EUR/USD volatility and long-term agents to align with FX trends. This customization demonstrates a key strength of our flexible N-agent framework.

Limitations of our framework mainly lie in the simplicity of the RL environment considered. We have imposed simplifications in our experiments compared to real-world markets, such as an action space where the agent may only buy or sell with its entire available balance, and no transaction costs which would penalise an agent making lots of unnecessary trades.

5 Conclusions and Further Work

This work has extended a MARL framework for algorithmic trading to low-latency timeframes, and demonstrated its performance against individual single-agent RL traders, as well as rule-based benchmark strategies. The framework outperformed single-agent traders on assets with clearer long-term trends across our trading horizon, but fell short on Forex data which is more highly fluctuating.

Our framework's robustness can be further evaluated by testing it on a broader range of assets and with even longer timeframes. Additionally, expanding the number of cooperating agents could reveal interesting dynamics. In real-world trading, transaction costs significantly impact decision-making. While data limitations prevented their incorporation in this study, a crucial next step would be to compare agent behavior with and without transaction costs. This comparison would provide valuable insights into the agent's ability to adapt and learn optimal strategies in a more realistic, market-like environment.

References

[1] A. Dorri, S. S. Kanhere, and R. Jurdak, "Multi-agent systems: a survey", IEEE Access **6**, 28573–28593 (2018).

[2] M. Kolat, T. Tettamanti, T. Bécsi, and D. Esztergár-Kiss, "On the relationship between the activity at point of interests and road traffic", Communications in Transportation Research **3**, 100102 (2023).

[3] M. Roesch, C. Linder, R. Zimmermann, A. Rudolf, A. Hohmann, and G. Reinhart, "Smart grid for industry using multi-agent reinforcement learning", Applied Sciences **10**, 6900 (2020).

[4] Z. Yan, N. Jouandeau, and A. A. Cherif, "A survey and analysis of multi-robot coordination", International Journal of Advanced Robotic Systems **10**, 399 (2013).

[5] A. Shavandi and M. Khedmati, "A multi-agent deep reinforcement learning framework for algorithmic trading in financial markets", Expert Systems with Applications **208**, 118124 (2022).

[6] M. Taghian, A. Asadi, and R. Safabakhsh, *Learning financial asset-specific trading rules via deep reinforcement learning*, 2020.

[7] J. Moody and M. Saffell, "Reinforcement learning for trading systems and portfolios", (1998).

[8] C. Watkins and P. Dayan, "Q-learning", Machine Learning **8**, 279–292 (1992).

[9] J. E. Moody and M. Saffell, "Reinforcement learning for trading", (1998).

[10] C. Gold, "FX trading via recurrent reinforcement learning", in 2003 IEEE international conference on computational intelligence for financial engineering, 2003. proceedings. (2003), pp. 363–370.

[11] X. X. Gao and L.-W. Chan, "An algorithm for trading and portfolio management using q-learning and sharpe ratio maximization", in (2000).

[12] Y. Nevmyvaka, Y. Feng, and M. Kearns, "Reinforcement learning for optimized trade execution", in Proceedings of the 23rd international conference on machine learning - ICML '06 (2006), pp. 673–680.

[13] X.-Y. Liu, Z. Xiong, S. Zhong, H. Yang, and A. Walid, *Practical deep reinforcement learning approach for stock trading*, 2022.

[14] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, *Continuous control with deep reinforcement learning*, 2016.

[15] T. Kabbani and E. Duman, “Deep reinforcement learning approach for trading automation in the stock market”, IEEE Access **10**, 93564–93574 (2022).

[16] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, *Playing atari with deep reinforcement learning*, 2013.

[17] S. Fujimoto, H. van Hoof, and D. Meger, *Addressing function approximation error in actor-critic methods*, 2018.

[18] J. Lee, R. Kim, S.-W. Yi, and J. Kang, “MAPS: multi-agent reinforcement learning-based portfolio management system”, Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, 4520–4526 (2020).

[19] Y. Huang, C. Zhou, K. Cui, and X. Lu, “A multi-agent reinforcement learning framework for optimizing financial trading strategies based on TimesNet”, Expert Systems with Applications **237**, 121502 (2024).

[20] J. Blackledge and M. Lamphiere, “A review of the fractal market hypothesis for trading and market price prediction”, Mathematics **10**, 117 (2021).

[21] R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction*, Adaptive computation and machine learning (MIT Press, Cambridge, Mass, 1998), 322 pp.

[22] L.-J. Lin, “Reinforcement learning for robots using neural networks”, (1993).

[23] P. J. Huber, “Robust Estimation of a Location Parameter”, The Annals of Mathematical Statistics **35**, 73–101 (1964).

[24] M. L. Littman, “Markov games as a framework for multi-agent reinforcement learning”, in *Machine learning proceedings 1994* (Elsevier, 1994), pp. 157–163.

[25] D. L. Joshi, “Use of moving average convergence divergence for predicting price movements”, Int. Res. Jnl. MMC **3**, 21–25 (2022).

[26] B. I. S, *Triennial central bank survey*, 2022.

Appendix

Model Hyperparameters

Hyperparameters that were used for the DRL agent experiments are given in Table 2. The ϵ-greedy policy employed by the DQN agents was decayed from an initial value to a final value as follows:

$$\epsilon_t = \epsilon_f + (\epsilon_i - \epsilon_f)\exp\left\{-\frac{t}{d}\right\} \tag{11}$$

where ϵ_i is the initial value, ϵ_f is the final value, t is the number of steps completed by the agent, and d is the ϵ decay parameter measured in number of steps. This ensures that the ϵ parameter never decays fully to zero to maintain a low level of exploration later on in training; this prevents the agent from getting stuck on a suboptimal policy.

Hyperparameter	Value
Replay buffer size	20
Batch size	10
Optimiser	Adam
Q-network hidden dims	[128,128]
Learning rate α	0.01
Discount factor γ	0.7
Initial ϵ_i	0.9
Final ϵ_f	0.05
ϵ decay d (steps)	500
Target update C (episodes)	5
Reward horizon n (steps)	10

Table 2: Hyperparameters used for DRL agents in the single- and multi-agent frameworks.

Performance Metrics

The *Cumulative Return* CR is given by

$$CR = \frac{V_f - V_i}{V_i} \times 100 \tag{12}$$

where V_i and V_f are the initial and final values of the portfolio respectively.

The *Sharpe Ratio* S is defined as

$$S = \frac{R_p - R_f}{\sigma_p} \tag{13}$$

where R_p is the annualised return of the portfolio, R_f is the risk-free rate of return (which is assumed to be 0 in this work), and σ_p is the annualised standard deviation (risk) of the portfolio's returns. R_p and σ_p can be further defined in terms of the daily returns as

$$R_p = \left(1 + \frac{V_f}{V_i} - 1\right)^{\frac{N}{T}} - 1 \tag{14}$$

$$\sigma_p = \sigma_d \sqrt{N} \tag{15}$$

where N is the number of trading days in a year (252), T is the number of days the agent has traded over, and σ_d is the standard deviation of daily returns.

The *Maximum Drawdown* MDD is defined as follows. Consider the set of peaks $\{P_i\}$ along an investment's portfolio value, and a corresponding set of troughs $\{T_i\}$, where T_i is the trough following peak P_i. The maximum drawdown is given by the maximum drop between a peak and a trough, measured as a percentage of the peak value:

$$MDD = \max_i \left\{ \frac{P_i - T_i}{P_i} \times 100 \right\} \tag{16}$$

Code

Code for the the single- and multi-agent frameworks, along with implementations of the experiments can be found at `https://github.com/olibridge01/MultiAgentTrading`.