Determining Entry/Exit Points of a Pairs Trading Strategy Using Online Reinforcement Learning

Carlos Gafa'
carlos.gafa.19@um.edu.mt
University of Malta
Imsida, Malta

Abstract

This research explores the effectiveness of an online reinforcement learning (RL) agent in optimizing entry and exit points for pairs trading, addressing declining arbitrage opportunities due to increased market efficiency. An actor-critic RL framework employing LSTM networks was tested on daily and hourly datasets with varying transaction costs. Results show superior performance of the agent in low-frequency trading scenarios with low transaction costs, whereas agility significantly declined under frequent structural breaks in intraday data. Future work should address model collapse and incorporate more realistic trading conditions.

ACM Reference Format:
Carlos Gafa'. 2025. Determining Entry/Exit Points of a Pairs Trading Strategy Using Online Reinforcement Learning. In *ARI5123 Assignment 2*. ACM, New York, NY, USA, 12 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Pairs trading is a market-neutral, relative-value strategy that first identifies two assets whose prices have historically moved together [11]. Typically, this occurs when two stocks are from the same sector and share common risk factors. The strategy identifies when assets in the pair are over/undervalued by considering the deviation of the price spread from its long term equilibrium value, and bets that this spread will revert to its mean value.

Traditionally, the entry and exit points for this strategy have been determined by a fixed spread threshold between the prices of the stock in the pair. However, as pairs-trading arbitrage drew growing interest from investors and hedge funds, the strategy's returns started to decline, with arbitrage opportunities becoming harder to come by [15]. To address this decline, researchers have devoted considerable effort to refining and upgrading pairs-trading methods. Some considerable advances in this domain, include the introduction of stop-loss rules [17], dynamic thresholds [35] and, more recently, machine-learning and reinforcement-learning enhancements [31] to restore performance.

The motivation of this work is to test the viability of an online RL agent to determine the entry and exit points for such a strategy by optimizing for long term returns. Such an online training approach, enables the agent to update policies "on the go" without the need for multiple epochs of training and validation, improving efficiency and adaptability to changing market regimes. The aim of this work is to build upon the work in [34], by performing hyperparameter tuning to test further input features from technical analysis, to include transaction costs, and to test the model in an intraday trading setting.

The agent was found to perform best in an interday setting, with only the spread as input features. Moreover, even with transaction costs the model displayed comparable performance to other machine learning strategies [15] in this area.

This work will be organized as follows. First, the necessary technical background on pairs trading and RL is given in Sec. 2. This is followed by detailed literature review, Sec. 3, on advances in pairs trading, and the use of machine learning in algorithmic trading, with a focus on pairs trading. The detailed methodology, including the modifications done to the RL algorithm by [34] and the data gathering process is discussed in Sec. 4. Finally, the results are presented and discussed in Secs. 5 and 6.

2 Technical Background

The relevant technical theory on pairs trading, and RL are discussed in this section.

2.1 Pairs Trading

Two assets are said to be co-integrated when their spread is mean reverting. A pairs trading strategy aims to exploit such statistical arbitrage opportunities between two co-integrated stocks, by going long the undervalued one of the pair while simultaneously shorting the overvalued stock [14]. These class of strategies assume that the price movements of the

ARI5123,

ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/XXXXXXX.XXXXXXX

two assets will continue to follow historical trends in the future, such that they will return to their long-term equilibrium value.

The general approach to a pairs trading strategy is as follows:

1. First, a pair of possibly co-integrated stocks, which are expected to remain so in the future, are identified.
2. Regression analysis, through ordinary least squares, total least squares or more advanced error correction models, is used to calculate the spread between the stocks [15].
3. Then, if the spread reaches preset boundaries the overvalued stock is shorted while a long position is taken on the undervalued one.
4. The position is then closed once the spread reverses to its mean value, or falls below some predetermined threshold.

In this work, an ordinary least squares approach to determine the spread between two assets is used. Consider two price series y_t and x_t, which have a unit root, then the spread ϵ_t is given by the error term

$$\epsilon_t = y_t - \beta x_t \,, \tag{1}$$

where β is obtained by regressing y_t on x_t and is known as the hedge ratio. The Z-score of this spread is then used to determine the entry and exit points using predefined thresholds. Using the Z-score of the spread normalizes it over time, allowing for consistent trading signals regardless of magnitude. It accounts for volatility and scale, making entry and exit thresholds more robust across different pairs and market conditions.

For instance, if the Z-score is positive and above a certain threshold, then the strategy would go long on stock y_t, while shorting x_t with relative ratio of β.

To test, for co-integration, the augmented Dickey–Fuller test is used on the spread. This tests for the null hypothesis that the time series has a unit root. Therefore, if the p-value of this test is above some critical size, then it cannot be rejected that there is unit root, implying that the time series is possibly non-stationary. This test is known as the Engle-Granger test.

Another popular test for co-integration is the Johansen test [13], which unlike the two-step Engle-Granger test is able to simultaneously examine multiple variables and determine how many co-integrating relationships exist among them. The test uses two complementary statistics: the trace statistic and the maximum eigenvalue statistic. The trace statistic tests whether there are at most r co-integrating relationships, while the maximum eigenvalue statistic tests whether there are exactly r co-integrating relationships. Both statistics are compared against critical values at different confidence levels to determine statistical significance. When the test statistics exceed their respective critical values, we reject the null hypothesis of no co-integration, indicating that the variables share a long-term equilibrium relationship.

2.2 Reinforcement Learning

In reinforcement learning the model parameters are learned through interaction with the environment. This type of learning is distinguished by two defining attributes: (i) trial and error, as the model discovers optimal strategies by exploring the environment, and (ii) delayed rewards, as immediate actions might have consequences far reaching into the future [25].

An RL model is defined by (i) the state space, $\mathbf{S}$, consisting of all the possible states of the environment, (ii) the action space, $\mathbf{A}$, consisting of all the possible actions that the model can take, (iii) the policy function $\pi : \mathbf{S} \mapsto \mathbf{A}$, which tells us how the model chooses its action at a given state, (iii) the reward function $R : \mathbf{S} \times \mathbf{S} \mapsto \mathbb{R}$ which gives us the return r_t when transitioning from a state S_t to another S_{t+1}, and (iv) the value function $V^\pi : \mathbf{S} \mapsto \mathbb{R}$, which is the expected return of rewards when starting at a state $s \in \mathbf{S}$ under a policy π [34]. Then, the goal of an RL strategy is to find the policy which maximizes the rewards over an episode, $\sum_{k=0}^{\infty} \gamma^k r_k$, where γ is known as the discount factor and is used so that immediate rewards are given greater importance, and to asymptotically bound the cumulative rewards [7].

The policy function can be approximated through an actor network $\pi(a|s, \theta_p)$ which learns the parameters θ_p. In this work, a stochastic policy function will be used to allow exploration as the model will be learning online. The Policy Gradient Theorem, tells us that the gradient of the agent's expected returns with respect to the policy parameters is given by [16]

$$\mathbb{E}[Q^\pi(s, a) \nabla_{\theta_p} \log \pi(s, a)] \,, \tag{2}$$

where $Q^\pi(s, a) = \mathbb{E}_{s'} [r(s, a) + \gamma V(s')]$ is the expected return when starting from a state s and taking action a, under the policy π. That is, the policy gradient which needs to be maximized, is expectation of the gradient of the log probability that an action is taken, scaled by the value of that action. In standard REINFORCE [33] methods, the Q function is estimated by Monte Carlo sampling the environment, that is by playing a number of episodes and calculating the discounted total rewards for each step of each episode. This approach, however, leads to high variance in the policy gradient which may cause convergence issues.

To reduce the variance of the policy gradient one can subtract a baseline from the Q function. In an actor-critic approach, the baseline is taken to be the value of the state itself, which gives us the advantage, $A^\pi(a_t, s_t) = Q^\pi(a_t, s_t) - V^\pi(s_t)$, that taking a particular action rather than following the policy gives us. The advantage function can be estimated using the TD error,

$$\delta^\pi = r_t + \gamma V^\pi(s_{t+1}) - V^\pi(s_t) \,, \tag{3}$$

since [30]

$$\mathbb{E}[\delta^\pi|s_t, a] = \mathbb{E}[r_t + \gamma V^\pi(s_{t+1})|s_t, a] - V^\pi(s_t) \quad (4)$$

$$= Q^\pi(a_t, s_t) - V^\pi(s_t) = A^\pi(a_t, s_t)\,. \quad (5)$$

Hence, the policy gradient to be maximized becomes

$$\nabla_{\theta_p} \log \pi(a_t|s_t; \theta_p)\,(r_t + \gamma V^\pi(s_{t+1}) - V^\pi(s_t))\,. \quad (6)$$

Now, the value function can be approximated using a critic network, $V^\pi(s_t; \theta_v)$, which has a similar structure to the actor network with parameters θ_v. To learn the critic parameters, one needs to minimize the mean squared error loss between the target value function V^π and its current estimate $V^\pi(\theta_v)$, where for the target the TD, $r_t + \gamma V(s_{t+1})$ is used. To avoid chasing a moving target, a copy of the critic network,$V^\pi(s_t; \theta_v^-)$, which is updated at every N steps, is used.

2.3 LSTMs

Data related to a stock price, including the spread between two stocks, is a sequential one. Therefore, recurrent neural networks (RNNs) will be used to build the actor and critic networks, as these are best suited to handle such sequential data.

Unlike feedforward neural networks, RNNs possess a form of memory that allows them to retain information from previous inputs in a sequence, thereby influencing the processing of current and future inputs. This is achieved by introducing loops within the network's architecture, where the output from a layer at a given time step is fed back into the same layer as part of the input for the next time step. This recurrent connection enables the network to capture temporal dependencies and learn from context. The core idea is that the hidden state at time step t, denoted as h_t, is computed as a function of the current input x_t and the hidden state from the previous time step h_{t-1}.

Despite their conceptual power for handling sequential data, traditional RNNs, face significant challenges when learning long-range dependencies. This is primarily due to the problem of vanishing or exploding gradients [4]. During the backpropagation through time (BPTT) algorithm used to train RNNs, gradients can shrink exponentially as they are propagated back through many time steps, making it difficult for the network to learn correlations between temporally distant events. Conversely, gradients can also grow exponentially, leading to unstable training. These issues limit the practical effectiveness of simple RNNs for tasks requiring the memory of events that occurred far back in the input sequence.

To address the limitations of traditional RNNs, particularly the vanishing gradient problem, Long Short-Term Memory (LSTM) networks were introduced by Hochreiter and Schmidhuber [12]. LSTMs are a specialized type of RNN architecture explicitly designed to learn long-term dependencies. The key innovation of LSTMs lies in their memory cell and a series of gating mechanisms that regulate the flow of information into and out of this cell, Fig. 1.

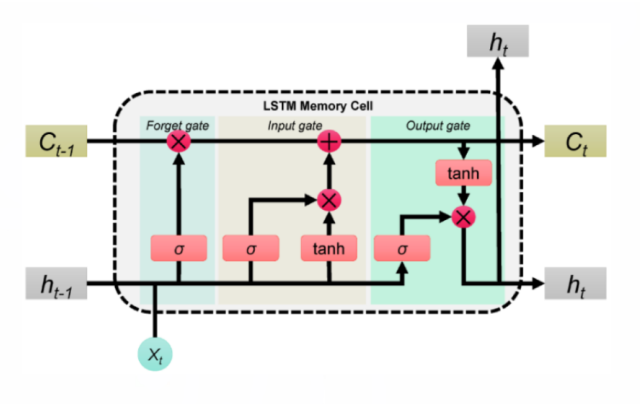

Figure 1. The structure of an LSTM cell. Figure from [9].

The gating mechanism in an LSTM unit is composed of three primary types of gates: the forget gate, the input gate, and the output gate. Each gate is a sigmoid neural network layer followed by a pointwise multiplication operation [28]. The sigmoid layer outputs numbers between zero and one, describing how much of each component of information should be let through. The forget gate, f_t, determines what information should be discarded from the cell state. It looks at the previous hidden state, h_{t-1}, and the current input, x_t, and outputs a number between 0 and 1 for each number in the previous cell state, c_{t-1}. This allows the LSTM to selectively erase irrelevant past information.

The input gate decides which new information will be stored in the cell state. This process involves two parts: first, a sigmoid layer decides which values will be updated. Then, a tanh layer creates a vector of new candidate values, $\tilde{c}_t$, that could be added to the state. The input gate essentially acts as a filter, determining how much of each candidate value should actually influence the cell's memory. These two are then combined to update the cell state c_t. This selective updating mechanism enables the LSTM to focus on relevant new information while preserving important existing memories.

Finally, the output gate determines what the next hidden state, h_t, will be, which is also the output for that time step. The output gate uses a sigmoid layer to decide which parts of the cell state will be outputted. The cell state is then passed through a tanh function and multiplied by the output of the sigmoid gate, so that only the desired parts are outputted. This sophisticated gating mechanism allows LSTMs to maintain and manipulate information over long sequences, effectively mitigating the vanishing gradient problem and capturing complex temporal dynamics.

3 Literature Review

This section provides a comprehensive overview of existing research related to algorithmic trading, specifically focusing on machine learning models and advancements in pairs trading strategies. This exploration highlights the progression from traditional approaches to more dynamic RL methods, ultimately laying the groundwork for the online RL approach employed in this work.

Machine learning models for algorithmic trading have seen an increasing popularity in the literature. These models can be broadly categorized into two methodologies; (i) supervised learning methods and (ii) RL methods [14]. The former are used to predict stock prices and directions. These predictions are then used to inform trading decisions. On the other hand, RL approaches take this a step further, and the model is used to determine the trading strategies itself. This direct optimization approach is generally preferred because it inherently accounts for environmental constraints such as transaction costs and market frictions, allowing for a more robust and adaptive strategy development.

In an RL setting, the model can be trained both offline or online. Online reinforcement learning involves training the agent in real-time environments where data is sequentially acquired, with policies continuously updated based on immediate feedback from market interactions; thus, the agent dynamically adapts to evolving market conditions and temporal dependencies in asset prices [8]. Conversely, offline reinforcement learning utilizes historical market data exclusively, training the policy without direct interaction with the live market environment. This offline approach mitigates the risk of significant financial losses during the exploration phase, yet it may suffer from distributional shift issues where learned policies do not adequately generalize to unseen market conditions due to static datasets and limited representational capacity of historical samples [10]. Given this trade-off, the online approach was specifically selected for this research due to its strength in adjusting to changing market dynamics, essential for successful pairs trading.

Recent advancements in online RL, have explored various approaches to enhance learning efficiency and stability. Some strategies involve synchronous updates, where learning steps are forced to occur within the sampling period, while asynchronous updates allow learning to happen in a separate process, decoupling it from the strict timing of environment interaction [22]. This decoupling is particularly beneficial as it enables actor and critic updates to operate at their own pace, whether slower or faster than real-time execution [26]. The use of an experience replay buffer is a key advancement, storing transition samples from environment interactions. It enables efficient data re-usability by allowing mini-batches, of independently distributed data, to be randomly sampled for updating actor and critic models, discarding older samples when full [27].

Online reinforcement learning approaches in algorithmic trading can be traced back to Moody and Saffel [23], who introduced a "recurrent reinforcement learning" method to feed a simple single layer model a series of price returns (or other trading features) and optimize trading positions to maximize the Sharpe ratio,

$$\frac{\text{Mean Returns} - \text{Risk Free Returns}}{\text{Volatility}}. \tag{7}$$

The policy model is trained in a synchronous manner without a replay memory, by collecting the reward at each timestep and using its gradient for stochastic gradient ascent to maximize the rewards. The approach was found to perform better than standard, offline Q-learning approaches at the time. This work opened the door for other subsequent results which optimized for the Downside Deviation ratio [5], Calmar ratio [2] and Conditional Value at Risk [1].

For instance, Deng et al. [8] introduced a deep direct reinforcement learning framework capable of representing complex financial signals, to optimize for the total returns and Sharpe ratio. Similarly, Li et al. [20] proposed an adaptive reinforcement learning method utilizing online policy gradient algorithms, specifically optimizing the Calmar ratio, thereby achieving enhanced risk-adjusted returns. Furthermore, Spooner et al. [32] employed an actor-critic methodology with risk-aware measures, notably optimizing for Conditional Value at Risk, leading to more resilient trading strategies under volatile market conditions. Collectively, these online methodologies underscore the adaptability and effectiveness of continuous policy updating in dynamic financial environments.

3.1 Advances in Pairs Trading

With the increasing rarity of arbitrage opportunities, researchers have been exploring methods to improve upon the standard pairs trading strategy described in Sec. 2.1. This section reviews these advancements, specifically focusing on techniques that address the inherent challenges of defining precise entry and exit points.

The establishment of appropriate boundaries is crucial for optimizing the pairs-trading strategy, as these serve as criteria for trade execution. A lower boundary may lead to more frequent trades but with diminished profits, whereas a higher boundary could yield greater returns per trade. However, this exposes the investors in a pairs trading strategy to significant losses if the spread does not revert to its mean once a position has been opened. To minimize this downside risk, stop-loss boundaries have been explored in the literature [15]. Stop-loss boundaries are predetermined thresholds set by traders to automatically exit positions once losses reach a specified limit. The application of stop-loss boundaries, thus, aims to prevent severe losses by cutting positions when divergence in the spread becomes excessive. Therefore,

the overall performance of pairs trading is heavily dependent on the judicious setting of these boundaries.

For example, Lin et al. [19] introduced a strategy focused on loss protection in pairs trading by implementing a minimum-profit condition. Their approach involved using an Ordinary Least Squares (OLS) method to create a spread between two co-integrated stocks. Trading actions were then initiated based on various predefined conditions related to the spread's standard deviations, specifically when the spread was above 0.3, 0.5, 0.75, 1.0, and 1.5 standard deviations. The study, which utilized daily closing prices of the Australia New Zealand Bank and the Adelaide Bank from January 2001 to August 2002, demonstrated that a decrease in the open condition value led to an increase in both the number of trades and the overall profits, while maintaining a protected minimum level of profit.

Puspaningrum et al. [24] concentrated on identifying optimal preset boundaries for pairs trading strategies based on co-integration techniques. Their methodology involved estimating parameters for the average trade duration, inter-trade interval, and the number of trades. These estimated parameters were then used to calculate optimal boundaries with the objective of maximizing the minimum total profit. The effectiveness of this approach was evaluated using daily closing price data from seven distinct pairs of stocks on the Australian Stock Exchange, spanning from January 2004 to June 2005. The results indicated that their proposed method was efficient in generating profits through the pairs-trading strategy.

Chen et al. [6] conducted an examination of pairs trading in Chinese commodity futures markets between 2006 and 2016, comparing three distinct methods: classical, closed-loop, and dynamic stop-loss. The classical method incorporated stop-loss boundaries into the closed-loop approach. The closed-loop method, in contrast, focused solely on a stop-profit barrier, executing the strategy without explicit consideration of risk if spreads failed to revert to the mean. The dynamic stop-loss method employed a variety of stop-profit and stop-loss barriers that adapted to the spreads, particularly when the spread exceeded a standard deviation set using historical average criteria. The findings revealed that all three methods achieved annualized returns exceeding 15%, with the closed-loop method demonstrating the highest profitability at 26.94%.

Roa [29] explored various threshold strategies for pairs trading, including fixed optimal thresholds, conditional volatility, percentile-based thresholds, spectral analysis, and neural network thresholds. The study aimed to identify which of these methods yielded superior performance in the context of pairs trading. The authors noted that the neural network threshold approach generally outperformed all other strategies examined, suggesting its potential for more effective boundary determination in pairs trading.

3.2 Reinforcement Learning Approaches to Pairs Trading

Reinforcement learning has emerged as a powerful paradigm for addressing the dynamic challenges inherent in pairs trading, particularly in the automated determination of entry and exit points. In [15], Kim and Kim, used a DQN algorithm to dynamically set the entry and stop loss boundaries. The authors used an action space of six pre-set entry/stop loss pairs, with the stop loss boundary being +2.0 of the entry boundary of the pair. The agent was then given a negative reward if the spread reached the stop loss boundary or did not converge to the mean, which implies that the it would have experienced a loss. The DQN agent was evaluated using various spread formation windows, and was found to outperform strategies in which the boundary pairs were fixed.

Using a similar action space, Lu et al. [21] used a DQN algorithm in an intraday trading environment. The authors noted that co-integration relationships are weaker in such intraday settings, and are often observed to break. Such breaks in the co-integration relationship are known as structural breaks.

In [14], Kim et al. combined such approaches which determine the trading and stop loss boundaries, with those that take trading actions directly on the value of the spread, to remove the dependency of the stop loss boundary on the entry boundary.

In [34], the authors took a different approach by utilizing an online actor-critic agent to determine the trading position based on the spread. The work was limited as no transaction costs were used, and the agent was only compared to buy-and-hold strategy. Building upon this prior work, the current research significantly extends this approach by rigorously testing the actor-critic agent with realistic transaction costs, experimenting with diverse input features, evaluating its performance in an intraday setting, and conducting a more detailed statistical comparison against established benchmark models.

4 Methodology

The aim of this section is to present and discuss the chosen model and training algorithm, including the parameter tuning performed and the comparative models used. The dataset selection and feature engineering is also discussed in this section.

4.1 Model and Algorithm

In the system used in this work, at the end of each time interval (hourly or daily), the agent will observe a sequence of previous interval features, up to the closing of the current interval, and take an action, a_t, at the end of the interval, which is held up the end of the next interval. Here it is assumed that at the end of the interval, the closing price does not change significantly, allowing the agent to observe the

closing price (or metrics derived from it) and take a position using that price. To lessen this assumption the spread cost may be increased by increasing the transaction cost.

Following [34], we use a simplified action space for our agent with the possible actions at the end of each time interval being:

1 Go long stock 1 and short stock 2,
0 Take no position in both stocks,
-1 Go short stock 1 and go long stock 2.

The state of the system is determined using a sequence, s_t, of n previous interval features, which will include the spread between the two stocks and the position taken on each interval of that sequence. The previous position is important for the agent to determine whether or not the next action will incur a transaction cost penalty. Moreover, it allows the agent to see length the time spent in a given position, as it might learn strategies to exit positions after a given amount of intervals to prevent excessive losses if the spread does not revert to its mean in a given time.

The reward of each action would then be the sum of the returns observed for each stock in the pair, with their respective position, minus the transaction cost,

$$\text{cost}_t = t_c \times \text{ABS}(a_t - a_{t-1}), \tag{8}$$

where t_c is taken to be 0.05% [15] throughout.

Within this system, the agent's policy and value functions will be modeled using an LSTM of variable size and depth, which feeds its final output to a standard feedforward layer. For the policy function, the final layer has three outputs and their softmax is taken to get the probability of each action. For the value function, the final layer has a single linear output, representing the value of the input state.

To train the neural networks, a slightly modified algorithm from [34] is used, which is based on [27]. This is given in Algorithm 1.

In this algorithm, the policy and value networks are updated, as discussed in Sec.2.2, after every time step by utilizing a batch of previous experiences stored in a replay buffer which stores the state, action, subsequent state and reward obtained for the previous M time steps. The target network, for the value function, is then updated at every N steps. Alternatively a soft target update,

$$\theta_v^- \leftarrow \tau\theta_v^- + (1-\tau)\theta_v, \tag{9}$$

every step, with $\tau = 0.001$, was tested. However this was found to give worse results then a hard target update. Moreover, clipping the gradients was found to stabilize the variance during training.

Algorithm 1 Actor-Critic Algorithm

Initialize critic and actor networks $V_{\theta_v}(s_t)$ and $\pi(a_t \mid s_t)$;
Initialize target critic $V_{\theta_v^-}(s_t)$ with weights $\theta_v^- \leftarrow \theta_v$;
Initialize replay memory;
while $t <$ End of Dataset **do**
    Get new state s_t;
    Agent picks action $a_t \sim \pi_\theta(a_t \mid s_t, \theta_p)$ and observes reward r_t;
    Append (s_t, a_t, r_t, s_{t+1}) to replay memory;
    Sample a mini-batch of (s, a, r) from replay buffer;
    Compute TD error $\delta_t = \left(r_t + \gamma V_{\theta_v^-}(s_{t+1}) - V_{\theta_v}(s_{t+1})\right)$ for each sample;
    Compute critic loss as MSE(δ_t) and back-propagate gradient;
    Update critic with ADAM;
    Compute actor loss as $\left(-\left(\delta_t \times \pi(a_t \mid s_t, \theta_p)\right)\right)$ and back-propagate gradient;
    Update actor with ADAM;
    if t mod $N = 0$ **then**
        Update target network $\theta_v^- \leftarrow \theta_v$;
    end if
    Update replay memory to keep most recent M states;
end while

4.2 Data and Features

To see in which time horizon the agent performs best, it will be tested on both daily and hourly data.

For each dataset, the spread, as defined in Eq.(1), was calculated using a rolling window of previous prices to avoid any look ahead bias. Both a long and short window size were tested as features during hyperparameter tuning. Moreover, the following technical indicators were computed for each stock in the pair and tested as features; RSI, MACD, OBV.

4.2.1 Daily Data. For the daily data we will make use of the same datasets as in [15]. In that publication, the period from 1990-01-02 till 2008-12-31 was used as the training period, and the period from 2009-01-02 till 2018-07-31 was used as the testing period. The daily closing prices of a list of 25 stocks in the S&P500, during the training period, was used to find a co-integrated pair, by applying the pairwise Johannes test and selecting the pair which rejected the null hypothesis with the largest test statistic. Using this filtering technique the pair XOM and CVX was identified as possibly co-integrated.

In this work, the training period will be utilized as the validation set, while the testing set will be used for online training and evaluation of the agent. Following [15], the window sizes of 30 and 120 days were used to calculate the spread, as these were found to give the best results on the training set.

4.3 Hourly Data

To identify co-integrated pairs in an hourly trading interval, a Yahoo Finance API [3] was first used to download the hourly data of the top 250 stocks in the S&P500 from 2023-05-07 till 2025-05-25. The short date range used is a limitation of Yahoo Finance. The pairwise Johannes test was then applied

Table 1. Table of the Johannes test statistics for the selected stocks.

	DG - DLTR	CPT - EQR
Trace Statistic	35.895	30.943
Trace Critical Value 99%	19.935	19.9349
Max Eigenvalue Statistic	34.596	27.265
Max Eigenvalue Critical Value 99%	18.520	18.520

to all combinations to find the pairs which had a trace and maximum eigenvalue statistic greater than the critical value required for a 99% confidence that the null hypothesis can be rejected. Out of these, two pairs were selected such that their constituent stocks were from the same industry, thereby increasing the likelihood that any co-integration observed during this period is not only due to chance alone. While applying the Johannes test on the test data introduces look ahead bias, this was done (i) to allow comparison between fixed threshold models and the RL agent in a setting where the pairs are likely to be co-integrated, (ii) due to the limited hourly data available. Two pairs, were, however, chosen such that one may be used as the validation set for hyperparameter tuning.

The pairs selected using this method are; (i) DG and DLTR which belong to the consumer defensive sector, and (ii) CPT and EQR which belong to the real estate sector. Their Johannes test statistics can be found in Table 1. Since the DG and DLTR pair had a higher test statistic, this will be used as the test set. A 24 and 72 time interval window sizes were found to give the best results on the validation set. The spread for these windows are given in Fig. 2 and Fig. 3. Here it is noted that the average spread between the different window sizes is almost the same, further indicating that this is stationary.

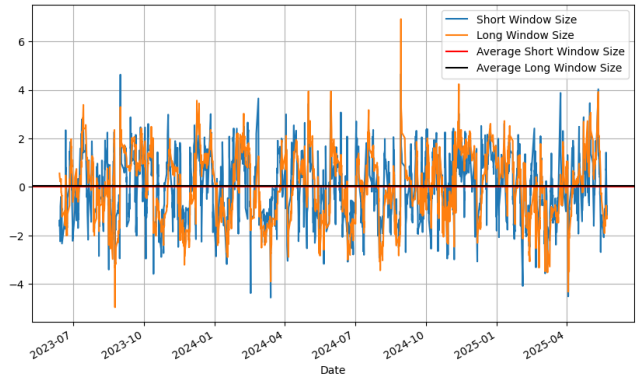

Figure 2. Spread for the DG and DLTR pair.

4.4 Parameter tuning

The hyperparameters selected for tuning are presented in Table 2. These parameters were optimized separately for both

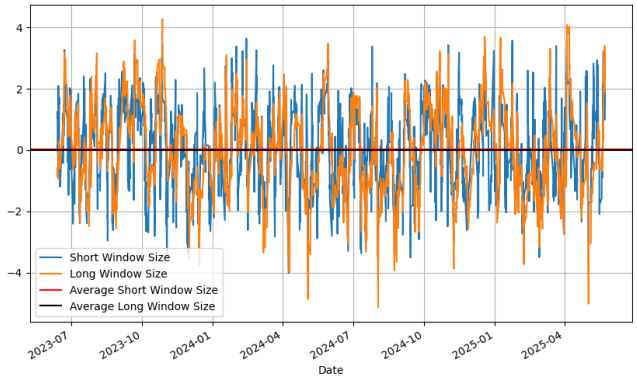

Figure 3. Spread for the CPT and EQR pair.

Table 2. Table of the optimized hyperparameters.

	Daily Data	Hourly Data
Sequence Length	20	10
γ	0.99	0.99
Batch Size	128	64
N	15	20
M	2000	1000
Hidden Size	32	32
Number of Layers	1	1
Features	Spread (Both Windows)	Spread (Both Windows)

daily and hourly datasets, using their respective validation sets. A comprehensive grid search was conducted over the hyperparameter space to identify the most effective configuration. As a result of the stochastic policy, and the inherent randomness of the training procedure, a high variance was observed between successive runs. Therefore, to ensure an optimal choice of parameters, the model was evaluated four times for each parameter set and the average mean returns were taken for that set. For each run, a different random seed was used to introduce variation; however, the same set of seeds was consistently applied across all parameter combinations to maintain fairness in comparison. The final set of hyperparameters was chosen based on the configuration that achieved the highest average mean return across these runs.

The learning rates were fixed for both the actor and the critic to 1×10^{-4} and 1×10^{-3} respectively [7]. By assigning a lower learning rate to the actor, its policy updates are effectively smoothed, allowing it to follow the critic's evaluation with greater stability and less risk of divergence. This approach aligns with common practices in actor–critic architectures, where the critic is allowed to adapt more quickly to changes in the environment, providing more reliable feedback for guiding the actor's policy improvement.

Based on the results of the hyperparameter tuning procedure, as shown in Table 2, it was observed that the agent achieved better performance with a discount factor of $\gamma = 0.99$, and a network architecture consisting of a single hidden layer of size 32, on both the daily and hourly validation sets. Interestingly, the inclusion of technical indicators as input features led to a decline in performance for both data types, with best results being obtained when the spread over both window sizes were used as the primary feature inputs. Furthermore, the agent trained on hourly data performed more effectively when a shorter replay memory, smaller batch size, and reduced sequence length were employed. This suggests that the agent benefits from a more responsive learning process on the hourly timescale, where rapidly shifting market conditions render older experiences less relevant and potentially detrimental to performance.

4.5 Comparison Models

To evaluate the performance of the reinforcement learning agent, three types of benchmark strategies were employed. First, a buy-and-hold strategy was implemented for both assets in the pair, serving as a passive benchmark without any active trading decisions. Second, two simple fixed-threshold pairs trading strategies were used These strategies triggered trades when the Z-score of the price spread exceeded predefined thresholds, specifically set at 0.5 and 1.0, respectively.

Moreover, to verify that the agent's behavior was not merely the result of random chance, a baseline random strategy was implemented. This strategy selected one of the three possible actions uniformly at random at each time step. Both the RL agent and the random strategy were each executed ten times using different random seeds to ensure variability and robustness in the results. Subsequently, a Kolmogorov–Smirnov (KS) test was applied to the aggregated returns from both approaches to assess whether they were drawn from the same underlying distribution.

5 Results

In this section the agent's results on the daily and hourly datasets, both with and without transaction costs, are presented and discussed.

5.1 Daily Data

The agent's cumulative performance on the XOM-CVX pair is presented in Fig. 4. Here it is evident that the agent significantly lacked behind both buy-and-hold strategies, with the exact performance statistics given in Table 3. However, the agent out-performed both fixed threshold models, across the entire date range taken.

A closer look at the distribution of returns for the agent, Fig. 5, reveals that an average daily return of 0.00006 ± 0.000055 was achieved by the agent, compared to the average return of -0.000498 ± 0.000095 attained by the random model.

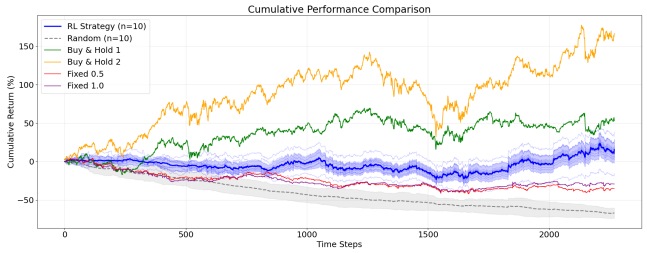

Figure 4. The cumulative returns of the RL agent on the XOM-CVX daily data pair.

In fact, the KS test for these distributions had a p-value of 0.00, rejecting the null hypothesis that the two distributions are identical.

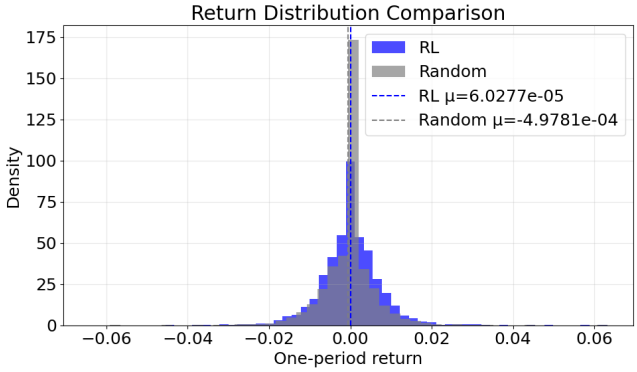

Figure 5. The distribution of daily returns on the XOM-CVX data pair.

This is further evidenced by the distribution of positions that the agent took, Fig. 6. Nonetheless, the high frequency of position shorting stock 1 while going long on stock 2, is indicative of model collapse.

This is additionally demonstrated by 50-day rolling window Sharpe ratio for all ten runs, Fig. 7, where all the different runs are observed to collapse between 1000-1500 daily steps. Such a collapse to this short/long position can be expected when looking at the cumulative returns of the buy-and-hold strategies, Fig. 4, where stock 2 has on average higher returns than stock 1. Therefore, taking a constant short/long position allows the agent to avoid transaction costs while still obtaining a positive return.

This collapse can be detrimental if the market regime changes and the agent ceases to explore alternative strategies. To encourage greater exploration, an epsilon-greedy approach may be employed, in which the agent selects a random position with a small probability of ϵ at each step. Alternatively, an entropy term can be incorporated into the actor's loss function to promote a more uniform action probability distribution, thereby increasing the likelihood of selecting alternative actions. Additionally, a lower discount

Table 3. Performance metrics for the various strategies on the daily dataset. For the RL agent and random model, the average and standard deviation over all the runs is given. Transaction Cost is set to 0.05%. To calculate the yearly metrics, 252 trading days are used. The Sharpe ratio uses a risk free rate of 0, following [15].

	Period Return	Step Return	Ann. Return	Ann. Volatility	Ann. Sharpe
RL Agent	0.1368 ± 0.1241	0.00006 ± 0.000055	0.0152 ± 0.0138	0.1221 ± 0.0006	0.1242 ± 0.1125
Random	−1.1300 ± 0.2152	−0.000498 ± 0.000095	−0.1254 ± 0.0239	0.1032 ± 0.0021	−1.2162 ± 0.2323
Buy&Hold 1	0.4254	0.000187	0.0472	0.1839	0.2567
Buy&Hold 2	0.9798	0.000432	0.1088	0.2102	0.5174
Fixed 0.5	-0.4253	-0.000187	-0.0472	0.1145	-0.4122
Fixed 1.0	-0.3439	-0.000151	-0.0382	0.0979	-0.3898

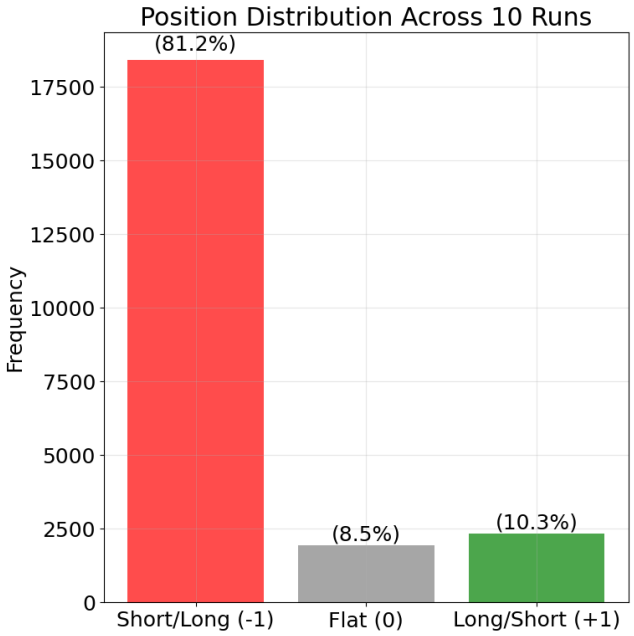

Figure 6. The distribution of the agent's positions on the XOM-CVX data pair.

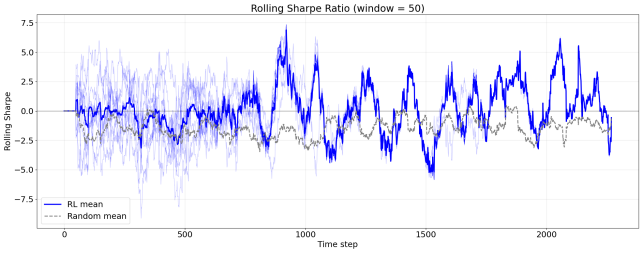

Figure 7. The agent's rolling Sharpe ratio on a 50-day window.

factor γ can be tested to make the model more sensitive to short-term volatility rather than focusing primarily on long-term returns.

In fact, when transaction costs are excluded, as shown in Fig. 8, the variance across different runs of the reinforcement learning agent increases noticeably. Moreover, under these conditions, the fixed-threshold models underperforms even the random model, further underscoring the necessity for more sophisticated approaches to pairs trading strategies.

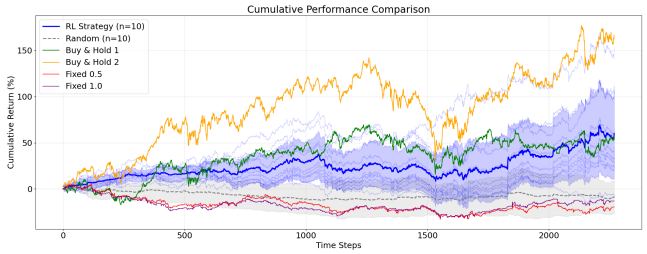

Figure 8. Cumulative returns on the daily dataset without transaction costs, $t_c = 0$.

To compare our model against that proposed in [15], their results from Table 4 ("Results of applying the DQN method to each window size using OLS") with formation window sizes 30 and 120 are used. These were found, by the authors of [15], to have a Sharpe ratio of 0.1197 and 0.1165 respectively. It should be noted, that in [15] other spread formation window sizes were also tested on the test dataset, such as 60 and 90 which had a Sharpe ratio of 0.1409 and 0.1327 respectively. Nonetheless, under the same spread window conditions, it can be concluded that the model proposed in this work had a comparable performance, with a Sharpe ratio of 0.1242 ± 0.1125, on the same dataset and the same transaction cost environment.

5.2 Hourly Data

The agent's cumulative performance on the DG-DLTR pair is presented in Fig. 9. From this figure, three different structural regimes after the initial period can be observed;

- From ∼ 400 to ∼ 1300 hourly steps, during which both buy-and-hold trended upwards while the fixed threshold pairs trading strategies remained relatively flat with a slight downward slope. During this period the

RL agent demonstrated a high variance between the different runs, likely due to it being in the early stages of the training procedure. However, on average the agent followed the same trend of the fixed threshold strategies, with a slight negative performance.

- From ~ 1300 to ~ 2100 hourly steps, during which all buy-and-hold and fixed threshold strategies trended downwards. During this period the variance of the RL agent started to the decrease, and on average the agent outperformed all other comparison strategies. This out performance is likely due to the agent learning from the first regime to take opposite positions from the fixed threshold strategies.
- From ~ 2100 hourly steps till the end of the dataset, during which the fixed threshold strategies outperformed the buy-and-hold strategies, which demonstrated a big drop in performance. The RL strategy remained mostly flat during this period, exhibiting a slight over performance over the pairs trading strategy with a fixed threshold of a Z-score of 1.0 over the whole dataset.

The detailed performance metrics over this dataset can be found in Table 4.

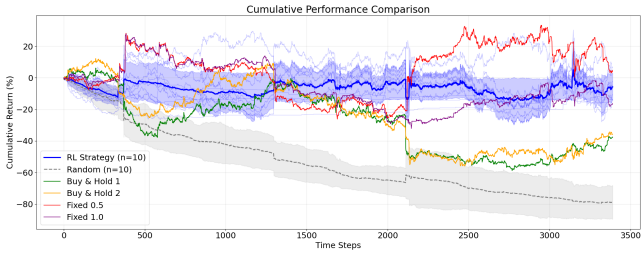

Figure 9. The cumulative returns of the RL agent on the DG-DLTR hourly data pair.

Taking a look at the distribution of the agent's position across all the runs, Fig. 10, it is noted that the agent took more diverse positions than the agent on the daily data. Nonetheless, the KS test gave a p-value of 0.0, indicating that the agent was not simply behaving randomly.

When transaction costs are removed, Fig. 11, the agent's variance across the different runs is observed to increase, indicating that exploration is higher since the model is not penalized for changing positions. In this case, the agent is seen to outperform the fixed threshold strategies, on average, during the market regimes when these are trending downward. However, the fixed threshold model with a threshold of 1.0, when the market changes regimes at ~ 2100 steps, as the agent is still learning this new regime.

6 Conclusion

The motivation for this work stemmed from the need to develop automated pairs trading systems capable of adapting

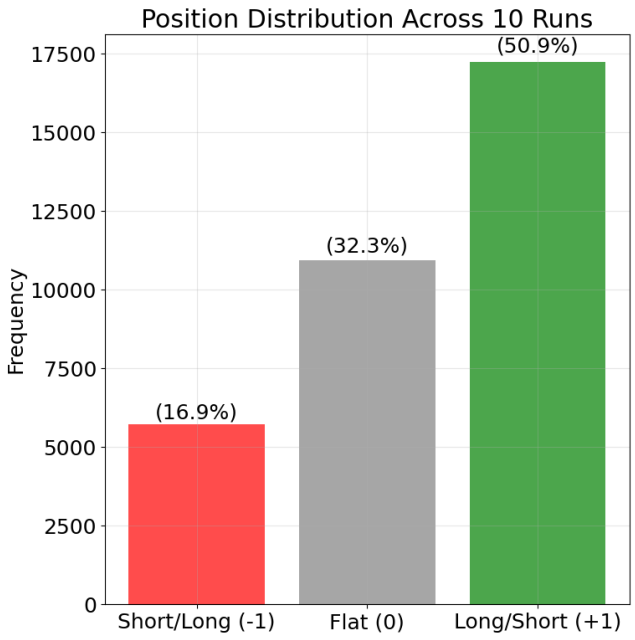

Figure 10. The distribution of the agent's positions on the DL-DLTR data pair.

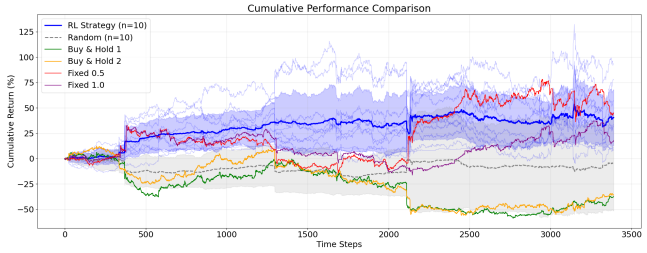

Figure 11. Cumulative returns on the hourly dataset without transaction costs, $t_c = 0$.

to dynamic market conditions while maintaining robust performance across varying temporal resolutions and market environments. The aim of this work, being to improve upon the results obtained by [34].

To this end, an online actor-critic RL agent was developed to take trading decisions by utilizing information on the spread between two stocks. Multiple market environments were simulated to test the agent's adaptability across different temporal frequencies and transaction cost scenarios.

The experimental results demonstrated that the agent achieved best comparative performance when operating on daily market data within low transaction cost environments. When exposed to higher-frequency hourly data, particularly during periods characterized by structural breaks or significant market volatility, the agent displayed notably slower agility in adapting its trading strategy.

Table 4. Performance metrics for the various strategies on the hourly dataset. For the RL agent and random model, the average and standard deviation over all the runs is given. Transaction Cost is set to 0.05%. To calculate the yearly metrics, six hours in a trading day are used, as per the dataset.

	Period Return	Step Return	Ann. Return	Ann. Volatility	Ann. Sharpe
RL Agent	−0.0633 ± 0.0921	−0.000019 ± 0.000027	−0.0282 ± 0.0411	0.2985 ± 0.1257	−0.2130 ± 0.2938
Random	−1.640 ± 0.3898	−0.000485 ± 0.000115	−0.7339 ± 0.1738	0.3136 ± 0.0236	−2.3702 ± 0.6354
Buy&Hold 1	-0.4724	-0.000139	-0.2106	0.3919	-0.5374
Buy&Hold 2	-0.4437	-0.000131	-0.1978	0.3806	-0.5198
Fixed 0.5	0.0463	0.000014	0.0206	0.3479	0.0593
Fixed 1.0	-0.1765	-0.000052	-0.0787	0.2556	-0.3079

However, several limitations were identified during this study. A critical issue observed was model collapse, where the agent consistently favored a particular trading position, limiting exploration and potentially impairing future adaptability in evolving market regimes. Additionally, the relatively short historical data period used for hourly datasets posed a constraint, limiting the generalizability of the results.

Future work should focus on several key enhancements. Implementing a stop-loss boundary could potentially mitigate excessive losses and improve risk management, addressing vulnerabilities observed during market regime shifts. Furthermore, exploring more sophisticated and potentially continuous action spaces [18] for position sizing could provide the agent with greater flexibility and responsiveness, thus improving performance in higher-frequency trading scenarios. Incorporating exploration techniques, such as entropy terms or epsilon-greedy policies, may help prevent model collapse by promoting consistent exploration of alternative strategies. Furthermore, pretraining [32] on a separate training set, possibly with a larger learning rate, before deployment to live real-time environment can serve to reduce the agent's variance and improve its performance in the early stages.

Finally, future research should also consider comprehensive comparative studies across diverse markets and asset classes to validate the robustness and generalizability of RL-based strategies in algorithmic trading. Extending the evaluation to longer and more varied datasets could provide deeper insights into the conditions under which online reinforcement learning strategies are most beneficial, further enriching the existing literature on machine learning applications in financial markets.

References

[1] Ali Al-Ameer and Khaled Alshehri. 2021. Conditional Value-at-Risk for Quantitative Trading: A Direct Reinforcement Learning Approach. arXiv:2109.14438 [q-fin.TR] https://arxiv.org/abs/2109.14438

[2] Saud Almahdi and Steve Yang. 2019. A constrained portfolio trading system using particle swarm algorithm and recurrent reinforcement learning. *Expert Systems with Applications* 130 (04 2019). doi:10.1016/j.eswa.2019.04.013

[3] Ran Aroussi. 2025. yfinance: Yahoo! Finance market data downloader. https://github.com/ranaroussi/yfinance Accessed: 2025-06-01.

[4] Y. Bengio, P. Simard, and P. Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks* 5, 2 (1994), 157–166. doi:10.1109/72.279181

[5] Francesco Bertoluzzo and Marco Corazza. 2007. Making Financial Trading by Recurrent Reinforcement Learning. In *Knowledge-Based Intelligent Information and Engineering Systems*, Bruno Apolloni, Robert J. Howlett, and Lakhmi Jain (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 619–626.

[6] Danni Chen, Jing Cui, Yan Gao, and Leilei Wu. 2017. Pairs trading in Chinese commodity futures markets: an adaptive cointegration approach. *Accounting and Finance* 57 (12 2017), 1237–1264. doi:10.1111/acfi.12335

[7] Nigel Cuschieri. 2022. Deep Reinforcement Learning for Financial Portfolio Optimisation. (2022). University of Malta.

[8] Yue Deng, Feng Bao, Youyong Kong, Zhiquan Ren, and Qionghai Dai. 2017. Deep Direct Reinforcement Learning for Financial Signal Representation and Trading. *IEEE Transactions on Neural Networks and Learning Systems* 28, 3 (2017), 653–664. doi:10.1109/TNNLS.2016.2522401

[9] dida. 2025. What is an LSTM Neural Network? https://dida.do/what-is-an-lstm-neural-network Accessed: 2025-06-01.

[10] Scott Fujimoto, David Meger, and Doina Precup. 2019. Off-Policy Deep Reinforcement Learning without Exploration. arXiv:1812.02900 [cs.LG] https://arxiv.org/abs/1812.02900

[11] Evan Gatev, William N. Goetzmann, and K. Geert Rouwenhorst. 2006. Pairs Trading: Performance of a Relative Value Arbitrage Rule. *The Review of Financial Studies* 19, 3 (2006), 797–827. doi:10.1093/rfs/hhj020

[12] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[13] Søren Johansen. 1988. Statistical analysis of cointegration vectors. *Journal of Economic Dynamics and Control* 12, 2 (1988), 231–254. doi:10.1016/0165-1889(88)90041-3

[14] Sang-Ho Kim, Deog-Yeong Park, and Ki-Hoon Lee. 2022. Hybrid deep reinforcement learning for pairs trading. *Applied Sciences* 12, 3 (2022), 944.

[15] Taewook Kim and Ha Young Kim. 2019. Optimizing the Pairs-Trading Strategy Using Deep Reinforcement Learning with Trading and Stop-Loss Boundaries. *Complexity* 2019, 1 (2019), 3582516. doi:10.1155/2019/3582516 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1155/2019/3582516

[16] M. Lapan. 2024. *Deep Reinforcement Learning Hands-On: A Practical and Easy-To-Follow Guide to RL from Q-Learning and DQNs to PPO and RLHF.* Packt Publishing, Limited. https://books.google.com.mt/books?id=5ZfP0AEACAAJ

[17] Tim Leung and Xin Li. 2015. Optimal Mean Reversion Trading with Transaction Costs and Stop-Loss Exit. arXiv:1411.5062 [q-fin.TR] https://arxiv.org/abs/1411.5062

[18] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2019. Continuous control with deep reinforcement learning. arXiv:1509.02971 [cs.LG] https://arxiv.org/abs/1509.02971

[19] Yan-xia Lin, Mccrae Michael, and Gulati Chandra. 2006. Loss protection in pairs trading through minimum profit bounds: A cointegration approach. *Journal of Applied Mathematics and Decision Sciences* 2006 (08 2006). doi:10.1155/JAMDS/2006/73803

[20] Yang Liu, Qi Liu, Hongke Zhao, Zhen Pan, and Chuanren Liu. 2020. Adaptive Quantitative Trading: An Imitative Deep Reinforcement Learning Approach. *Proceedings of the AAAI Conference on Artificial Intelligence* 34, 02 (Apr. 2020), 2128–2135. doi:10.1609/aaai.v34i02.5587

[21] Jing-You Lu, Hsu-Chao Lai, Wen-Yueh Shih, Yi-Feng Chen, Shen-Hang Huang, Hao-Han Chang, Jun-Zhe Wang, Jiun-Long Huang, and Tian-Shyr Dai. 2022. Structural break-aware pairs trading strategy using deep reinforcement learning. *The Journal of Supercomputing* 78 (2022), 3843–3882. doi:10.1007/s11227-021-04013-x Accessed: 2025-06-01.

[22] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous Methods for Deep Reinforcement Learning. arXiv:1602.01783 [cs.LG] https://arxiv.org/abs/1602.01783

[23] J. Moody and M. Saffell. 2001. Learning to trade via direct reinforcement. *IEEE Transactions on Neural Networks* 12, 4 (2001), 875–889. doi:10.1109/72.935097

[24] Heni Puspaningrum, Yan-xia Lin, and Chandra Gulati. 2010. Finding the Optimal Pre-Set Boundaries for Pairs Trading Strategy Based on Cointegration Technique. *Journal of statistical theory and practice* 4 (09 2010). doi:10.1080/15598608.2010.10411994

[25] A. G. Barton R. S. Sutton. 1999. *Reinforcement Learning: An Introduction.* Trends in Cognitive Science.

[26] Mircea-Bogdan Radac and Darius-Pavel Chirla. 2025. Near real-time online reinforcement learning with synchronous or asynchronous updates. *Scientific Reports* 15, 17158 (2025). doi:10.1038/s41598-025-00492-7 Accessed: 2025-06-01.

[27] Simon Ramstedt and Christopher Pal. 2019. Real-Time Reinforcement Learning. *arXiv preprint arXiv:1911.04448* (2019). https://arxiv.org/abs/1911.04448 Accessed: 2025-06-01.

[28] Sebastian Raschka, Yuxi (Hayden) Liu, and Vahid Mirjalili. 2022. *Machine Learning with PyTorch and Scikit-Learn.* Packt Publishing, Birmingham, UK. https://sebastianraschka.com/books/machine-learning-with-pytorch-and-scikit-learn/ Accessed: 2025-06-01.

[29] A. A. Roa. 2018. *Pairs Trading: Optimal Threshold Strategies.* Master's thesis. Universidad Complutense de Madrid. https://deanstreetlab.github.io/papers/papers/Statistical%20Trading/Pairs%20Trading%20-%20Optimal%20Threshold%20Strategies.pdf Accessed: 2025-06-01.

[30] Russ Salakhutdinov. [n. d.]. 10703 Deep Reinforcement Learning and Control. ([n. d.]). https://www.cs.cmu.edu/~rsalakhu/10703/Lectures/Lecture_PG2.pdf.

[31] John Smith. 2020. Pairs Trading Strategy Using Machine Learning. https://digitalcommons.usu.edu/cgi/viewcontent.cgi?article=2447&context=gradreports Accessed: 2025-06-01.

[32] Thomas Spooner and Rahul Savani. 2020. Robust Market Making via Adversarial Reinforcement Learning. doi:10.24963/ijcai.2020/626

[33] Ronald J. Williams. 1999. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning* (1999).

[34] Yiding Zhao Yichen Shen. 2017. Deep Reinforcement Learning for Pairs Trading Using Actor-critic. (2017). https://github.com/shenyichen105/Deep-Reinforcement-Learning-in-Stock-Trading.

[35] Dong-Mei Zhu, Jia-Wen Gu, Feng-Hui Yu, Tak-Kuen Siu, and Wai-Ki Ching. 2021. Optimal pairs trading with dynamic mean-variance objective. *Mathematical Methods of Operations Research* 94, 1 (2021), 145–168. doi:10.1007/s00186-021-00751-z Accessed: 2025-06-01.