Group 96 Final Report: Hull Market Challenge

Shike Chen, Xiaotian Lou, Tianjiao Xiao
{xiaot13,loux8,chens356}@mcmaster.ca

1 Introduction

Currently majority of the financial market prediction models are relying on traditional index digging, technical analysis, regression models. However, with modern machine learning techniques emerging in NLP, the industry see a huge potential in bringing complex models to analyze and predict the market movement.

Over the years, majority of the investors, including professionals, have struggled to consistently beat the S&P500 index. In this challenge, we also understand the effect of the efficient market hypothesis, which states that stock prices fully reflect all available information. Thus, it is impossible to consistently achieve higher returns than average market returns on a risk-adjusted basis, given that stock prices should only react to new information.

However, with the advancement of machine learning techniques, if we are able to implement these new techniques to analyze the market data, we may be able to uncover hidden patterns and profit. Since these are also considered as new information, the effective market hypothesis still holds.

Since we do not know the preprocessing steps of the features, it is difficult for the team to use traditional, statistical methods to give meaningful insightful analysis. Thus, we decided to use predominantly unsupervised learning methods to analyze the data and predict the S&P500 return.

The overall objective of our project is to predict the S&P500 return using the pre-processed data provided by Hull Tactical. Beyond simple directional prediction, our ultimate goal is to generate a trading algorithm that optimizes the competition's specific Sharpe ratio. Unlike a standard risk-adjusted metric, the competition Sharpe ratio imposes strict penalties on strategies that exhibit excessive volatility relative to the market or fail to match the market's baseline return. Therefore, our modeling approach focuses on balancing high returns with rigorous risk control to maximize this specific performance metric.

2 Related Work

Predicting stock prices is effectively modeled as a sequential decision-making problem rather than a static regression task. Zhang et al. [1] argue that Reinforcement Learning (RL) offers a superior framework by bypassing explicit forecasting to optimize trade positions directly, thereby navigating the low signal-to-noise ratio of financial data. Among these methods, Proximal Policy Optimization (PPO) [2] is particularly effective due to its ability to stabilize policy updates in non-stationary, noisy market environments.

Recent literature highlights the versatility of RL in quantitative finance [3]. Huang et al. [4] successfully applied Deep RL to portfolio allocation with complex action spaces, while Soleymani and Paquet [5] introduced graph-convolutional architectures to capture asset inter-dependencies. Furthermore, Jiang et al. [6] proposed model-free frameworks to address dynamic market complexities in large-scale settings.

However, a critical challenge lies in defining reward functions that account for tail risk. Zakamouline and Koekebakker [7] demonstrate that standard mean-variance metrics fail to capture higher moments like skewness and kurtosis, proposing the Generalized Sharpe Ratio (GSR) as a more robust measure. Complementing this, Lam et al. [8] emphasize the importance of integrating coherent risk measures directly into RL approximation to ensure robustness against worst-case scenarios.

3 Dataset

There is no significant change in the final dataset compared to the progress report. The dataset used in this project is the same equity return prediction dataset referenced in the project proposal. An up-

dated version of the dataset was released on November 6th. The update primarily expanded the number of observations and refined the calculation of forward returns. However, this revision did not materially change the statistical properties of the data distribution, nor did it require changes to our modeling pipeline, since our learning framework does not rely on absolute scale but rather on relative directional signals.

The dataset consist of 8 different feature groups. These feature groups are:

- M* Market Dynamics/Technical features.
- E* Macro Economic features.
- I* Interest Rate features.
- P* Price/Valuation features.
- V* Volatility features.
- S* Sentiment features.
- MOM* Momentum features.
- D* Dummy/Binary features.

For example, a feature related to Price/Valueation will be denoted as column V1, V2, etc.

3.1 Preprocessing and Analysis

Some analysis are ran on the dataset to understand the basic nature of the data.

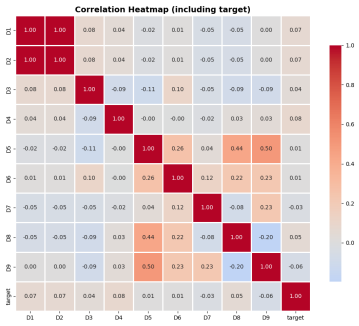

Figure 1: Relationship between binary and target

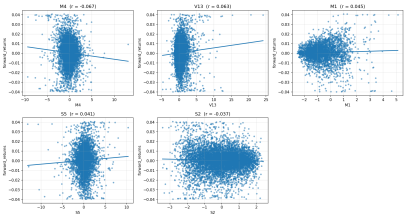

Figure 2: Distribution of target variable

From observing Figure 1 and Figure 2, as expected in financial forecasting problems, the empirical distribution of features and target is extremely noisy and return is distributed closely to zero. Features released are all have a weak to no visible relationship with the target variable. Therefore, when training, we normalize the target variable to have zero mean and unit variance which make sure the loss will not diminish during backward propagation.

The dataset also contains missing values, particularly in indicators constructed from rolling windows of different lengths. To avoid discarding information, we do not remove rows with missing data. Instead, we replace `NaN` values with zeros and add binary missing indicator flags for each affected feature:

$$x_i^{(\text{filled})} = \begin{cases} 0, & \text{if } x_i \text{ was missing} \\ x_i, & \text{otherwise} \end{cases}$$

This preserves potential information carried by the absence of values. Data loading and cleaning follows our `load_data` and missing-indicator augmentation utility functions.

We also realized that because the datapoint we have is about 9000 datapoints, given a trading days in a year is about 252 days, the data only covers about 37 years of data. Due to the nature of the financial market, data points from the early years have large amount of features missing and market changed significantly over the past 37 years, thus we decided to remove the first 4000 data points from the training and evaluation process and heavily evaluate on the last 1000 data points. **For the rest of the report, an overall result would refer to the result on the data excluding the first 4000 data points.**

4 Features

We did not change too much about our features in our final implementation. The main preprocessing steps applied to the features are:

- Standardization of all numeric features via z-score normalization.
- Addition of missing-value indicator features for all originally `NaN`-carrying columns.
- Removal of early data points where it is not representative to the current market situation.

- Regularization techniques to penalize noisy features.

Because financial features often exhibit collinearity, we conducted correlation inspections to assess which features carry directional information relative to `target`. However, no workable linear correlation found in this dataset. The processed data is fed to a reinforcement learning environment to generate state frames, then the agent will learn based on the rewards from the reinforcement learning environment.

We introduced regularization techniques to penalize noisy features that do not have real predictive power.

5 Implementation

5.1 Competition SHARPE Ratio

One of our main objective is to have a model that can generalize well on unseen data, perform well against the adjusted competition SHARPE ratio.

The strategy return at time t, denoted as R_t^S, is calculated based on the position $p_t \in [0, 2]$, the risk-free rate r_f, and the forward market return R_t:

$$R_t^S = r_f \cdot (1 - p_t) + p_t \cdot R_t \tag{1}$$

The base annualized Sharpe ratio is computed using the geometric mean of excess returns (μ_S) and the annualized standard deviation (σ_S):

$$\text{Sharpe} = \frac{\mu_S}{\sigma_S}\sqrt{252} \tag{2}$$

However, the final competition score is adjusted by two penalty factors:

- **Volatility Penalty (P_{vol}):** This penalizes strategies that exhibit volatility significantly higher than the market volatility (σ_M). Specifically, it penalizes strategies where the volatility ratio exceeds 1.2:

$$P_{\text{vol}} = 1 + \max\left(0, \frac{\sigma_S}{\sigma_M} - 1.2\right)$$

- **Return Penalty (P_{ret}):** This penalizes strategies that fail to match the market's mean excess return (μ_M). If the strategy underperforms the market, the gap is squared to apply a heavy penalty:

$$P_{\text{ret}} = 1 + \frac{\max(0, (\mu_M - \mu_S) \cdot C)^2}{100}$$

where $C = 252 \times 100$ scales the annualized return percentage.

The final Adjusted Sharpe Ratio is defined as:

$$\text{Score} = \frac{\text{Sharpe}}{P_{\text{vol}} \times P_{\text{ret}}} \tag{3}$$

Difference from Standard Sharpe: The standard Sharpe ratio rewards any increase in return per unit of risk. In contrast, the Competition Sharpe Ratio acts as a constrained optimization metric. It implies that a "good" strategy must not only be efficient but must also (1) avoid excessive risk relative to the market regime (volatility constraint) and (2) at least match the market's absolute return (return constraint). This prevents "safe" strategies with low volatility and low returns from achieving high scores, which would be possible under a standard Sharpe formulation.

5.2 Reinforcement Learning

Our core modeling framework remains a reinforcement learning (RL) agent based on Proximal Policy Optimization (PPO). Rather than predicting raw return values, the task is formulated as a sequential decision problem. At each timestep, the agent selects two of the actions spaces:

$$a_t \in \{0, 0.5, 1, 1.25, 2\}$$

$$a_t \in \{0, 1, 1.25\}$$

These action spaces are arbitrary, inspired by the modified challenge Sharpe ratio calculation. The competition requires a long-only approach with a maximum leverage of 2x. However, to adjust such action space to the standard RL, it requires us to transform leverage into action spaces. Thus we discretize the action space.

5.3 Agent

To capture the complex dynamics of the market, we implemented an Actor-Critic framework where the policy π_θ and value function V_ϕ share a common feature extraction backbone. We experimented with two distinct architectures to balance model complexity with the ability to learn temporal dependencies.

5.3.1 MLP Architecture

Our baseline agent utilizes a deep feed-forward network. The shared feature extractor consists of an input projection followed by an expansion-compression bottleneck design, intended to capture non-linear interactions between features before

decision-making. The forward pass is defined as:

$$h_1 = \text{ReLU}(W_1 x + b_1)$$
$$h_2 = W_2 h_1 + b_2 \quad \text{(Expansion to } 2 \times \text{hidden)}$$
$$h_{\text{shared}} = \text{ReLU}(W_3 h_2 + b_3) \quad \text{(Compression)}$$

The output h_{shared} is then fed into separate linear heads to produce action logits and value estimates. This architecture is computationally efficient and serves as a strong baseline for instantaneous decision-making without explicit memory.

5.3.2 LSTM Architecture

Financial time-series data is inherently sequential. To leverage this, we implemented a Recurrent Neural Network (RNN) variant using Long Short-Term Memory (LSTM) cells. The architecture proceeds as follows:

1. **Feature Extractor:** A linear layer projects the observation space to the hidden dimension size followed by a ReLU activation.
2. **Temporal Encoding:** The extracted features are passed through an LSTM layer which maintains an internal hidden state (h_t, c_t) across time steps.
3. **Heads:** The LSTM output is projected to action probabilities and value estimates.

To train this recurrent agent effectively with PPO, we utilize a block-shuffling mechanism. The trajectory buffer is divided into contiguous time blocks (e.g., 256 steps). The internal LSTM states are stored and re-injected at the beginning of each block during the update phase, allowing the agent to learn temporal dependencies while preserving the I.I.D. assumption required for stochastic gradient descent.

Both architectures incorporate L2 regularization on the weights to prevent overfitting to the noisy financial data.

5.3.3 Rewards

We perform a standard transformation from stock returns to a Monte Carlo State Space. To guide the agent, we experimented with four distinct reward architectures, evolving from simple directional matching to direct optimization of the competition metric.

- **Simple Sign Reward:** This baseline reward focuses purely on directional accuracy. It defines a threshold τ to handle flat market conditions. The reward r_t is determined by whether the sign of the action matches the sign of the target (forward return):

$$r_t = \begin{cases} R_{\text{correct}} & \text{if } \text{sgn}(a_t) = \text{sgn}(y_t) \\ R_{\text{correct}} & \text{if } |y_t| < \tau \text{ and } a_t = 0 \\ P_{\text{wrong}} & \text{otherwise} \end{cases}$$

  where R_{correct} is a positive reward and P_{wrong} is a penalty.

- **Windowed Sign Reward (v1):** To incorporate the magnitude of returns and enforce consistency, this function adds the realized P&L to the sign reward. Crucially, it introduces a *rolling window penalty*. The agent maintains a history of recent rewards; if the cumulative reward over the window W is non-positive, an additional penalty is applied:

$$r_t = (a_t \cdot y_t) + r_{\text{sign}} + P_{\text{window}}$$

  where P_{window} is triggered if $\sum_{i=t-W}^{t} r_i \leq 0$. This discourages the agent from getting stuck in local minima of poor performance.

- **Windowed Sign Reward (v2):** Our most advanced iteration introduces an *Action Penalty* to combat passivity. In addition to the logic in v1, this function monitors the moving average of the action magnitude. If the agent remains too passive (holding or taking low-confidence actions) over a window W_a, a cumulative penalty is applied:

$$r_t = (a_t \cdot 100 \cdot y_t) + r_{\text{sign}} + P_{\text{window}} + P_{\text{action}}$$

  Here, the target is scaled by 100 to normalize gradients, and P_{action} ensures the agent maintains a minimum level of market engagement.

- **Competition Metric Reward:** Finally, we implemented a reward function that directly optimizes the competition's objective. Unlike the proxy rewards based on instantaneous signs, this function maintains a rolling window (default $W = 252$) to estimate the volatility-adjusted Sharpe ratio at every timestep. The reward r_t is defined as the current value of the competition metric:

$$r_t = \frac{\text{Sharpe}_{t,W}}{P_{\text{vol}} \cdot P_{\text{ret}}}$$

  where P_{vol} and P_{ret} are the dynamic penalties for excess volatility and underperformance

relative to the market within the window. This approach encourages the agent to learn the complex trade-offs between risk, return, and market correlation directly.

5.4 Baselines

Since we are evaluating against the Competition SHARPE Ratio, the best and most straightforward baseline is to BUY the S&P500 index and HOLD it throughout the evaluation period. We will evaluate our RL agent against this baseline with the same data period since the baseline SHARPE Ratio vary over different time periods. The baseline SHARPE Ratio overall is 0.556.

5.5 Loss Function

We train the agent by minimizing a composite loss function that combines the Proximal Policy Optimization (PPO) surrogate objective, a value function error term, and an entropy bonus to encourage exploration. The total loss function $L(\theta)$ is defined as:

$$L(\theta) = \mathcal{L}^{CLIP}(\theta) + c_1\mathcal{L}^{VF}(\theta) - c_2 S[\pi_\theta](s_t) + \lambda_{L2}||\theta||^2$$

- $\mathcal{L}^{CLIP}(\theta)$ is the negative of the clipped surrogate objective. We minimize this term to maximize the expected reward while limiting the size of policy updates to ensure stability:

$$\mathcal{L}^{CLIP}(\theta) = -\hat{\mathbb{E}}_t\Big[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\Big]$$

  Here, $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ is the probability ratio, and $\hat{A}_t$ is the estimated advantage.

- $\mathcal{L}^{VF}(\theta)$ is the value function loss, calculated as the mean squared error between the estimated value $V_\theta(s_t)$ and the observed returns R_t. In our implementation, we scale this term by 0.5:

$$\mathcal{L}^{VF}(\theta) = 0.5 \cdot (V_\theta(s_t) - R_t)^2$$

- $S[\pi_\theta](s_t)$ is the entropy of the policy distribution. By subtracting this term (maximizing entropy), we prevent the agent from prematurely converging to a deterministic, suboptimal policy.

- λ_{L2} is the coefficient for L2 regularization applied to the network weights to mitigate overfitting to financial noise.

In our final configuration, we set the value coefficient $c_1 = 0.5$, the entropy coefficient $c_2 = 0.05$, and the L2 regularization parameter $\lambda_{L2} = 10^{-4}$.

6 Results and Evaluation

6.1 Performance Comparison

We evaluated our models across different time horizons to test for consistency and generalization. The "Overall" score represents the Sharpe ratio across the entire dataset, while specific windows (e.g., "Last 500") isolate performance on the most recent, unseen market regimes. The "Baseline" represents a simple Buy-and-Hold strategy.

Table **??** summarizes the results of our experiments.

Model	Overall (Train+Test)	Last 500 (Held-Out)	Last 1000	Last 1500	First 1500 (Training)
Baseline (Buy & Hold)	0.56	1.19	0.42	0.64	0.09
Window LSTM 5 action (Early Stop)	0.75	1.34	0.57	0.90	0.19
Window MLPs Small 3 action (Large Step)	2.32	1.49	0.54	0.62	4.48
Window MLPs Small 3 action (Small Step)	1.40	**1.59**	**0.51**	**0.68**	1.85
Window MLPs Large 3 action	**3.56**	1.12	0.26	0.48	**7.14**
Window MLPs Large 3 action (Early Stop)	0.97	**2.14**	0.34	0.47	1.55
Sharpe LSTM	0.21	N/A	N/A	N/A	N/A
Sharpe MLPs Small	0.15	N/A	N/A	N/A	N/A

Table 1: Comparative Sharpe Ratios across different model architectures and time horizons.

The results in Table **??** demonstrate that while larger models (e.g., Window MLPs Large) achieved higher peak returns in specific instances, they failed to generalize across longer time horizons. In contrast, the **Window MLPs Small (Small Step)** and **Window LSTM** models emerged as our most robust performers. Both models succeeded in consistently outperforming the **Baseline (Buy & Hold)** strategy across every evaluation window (Last 500, 1000, and 1500 days). Specifically, the Small MLP model provided the highest stability on the immediate held-out set (Sharpe 1.59 vs. Baseline 1.19), while the LSTM model demonstrated superior capacity over the medium term (Last 1500 Sharpe 0.90 vs. Baseline 0.64). This consistent ability to generate alpha regardless of the evaluation window indicates that these agents have learned fundamental market dynamics rather than overfitting to specific regimes.

6.2 Hyperparameter Sensitivity and Overfitting

A critical finding during our experimentation was the model's sensitivity to the learning rate.

- **Large Step (3×10^{-4}):** Models trained with this rate tended to overfit rapidly. As seen in the "Window RL Large" entry (Table **??**), the model achieved an extremely high Sharpe ratio on the training data (First 1500: 7.14) but extremely inconsistent performance on almost every other time window, Non of which beat the baseline.

- **Small Step (1×10^{-5}):** Conversely, reducing the learning rate resulted in a steady, stable increase in performance over 5000 iterations. The "Window RL Small (Small Step)" model sacrificed raw training performance but achieved the most consistent generalization, beating the baseline in the held-out set (1.59 vs 1.19) and maintaining stability across all lookback windows.

6.3 Action Space reflection

In the earlier experiments, the models performed poorly with larger action spaces. Thus, we decided to reduce the action space to have better performance. However, after further experiments, we found that LSTM model is able to utilize the larger action space. We believe potentially, MLPs will also be able to utilize the larger action space with further hyperparameter tuning.

6.4 Architecture Analysis

While the **LSTM** architecture was the top performer in terms of peak capability (Window LSTM Early Stop), we found that the **Small MLP** combined with a small step size offered the best balance of simplicity and consistency. We prioritize consistency in this challenge, as we seek a model that outperforms the baseline at any given time period rather than one that chases specific market anomalies in the training set.

Large MLP models exhibited a tendency to overfit, especially with higher learning rates, as evidenced by its inconsistent performance across time windows. The smaller MLP with a reduced learning rate emerged as the most reliable choice.

6.5 Reward Function Efficacy

We observed a stark contrast between our proposed Windowed Reward function and the direct optimization of the Sharpe ratio.

The direct **Sharpe Reward** formulation failed to provide a meaningful gradient signal. As shown in Table **??**, models trained on this reward (Sharpe LSTM/RL Small) achieved negligible scores ($<$ 0.25). Figure **??** illustrates that the model effectively learned nothing, with the raw reward fluctuating near zero without improvement.

In contrast, the **Windowed Reward** provided a dense and stable signal, allowing the agent to steadily improve its policy over time, as shown in Figure **??**.

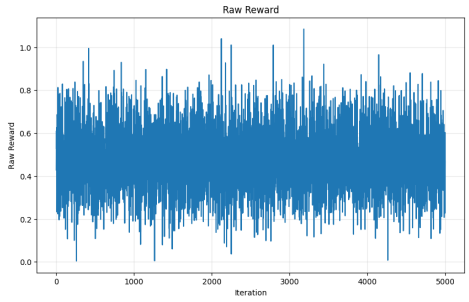

Figure 3: Training curve for Direct Sharpe Reward. The agent fails to learn, with the raw reward oscillating without an upward trend.

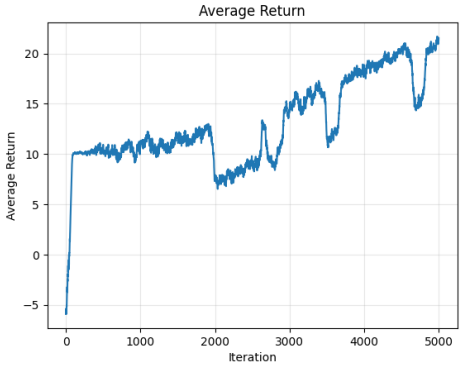

Figure 4: Training curve for Windowed Reward. The steady increase in raw reward indicates effective policy optimization.

6.6 Error Analysis

To systematically understand the limitations of our approach, we analyzed the model's performance failures across three dimensions: generalization gaps, market regime dependency, and reward signal sparsity.

6.6.1 Generalization Gap and Overfitting

The most prominent error pattern observed was the massive divergence between training and testing performance in models with aggressive hyperparameters. As illustrated in Table **??**, the **Window MLPs Large (Large Step)** model achieved a superhuman Sharpe ratio of 7.14 on the training set (First

1500 days). However, on the held-out "Last 500" window, its performance collapsed to 1.12, underperforming the baseline (1.19).

- **Pattern:** The model learned to memorize specific noise patterns in the training data rather than underlying market dynamics. This "overfitting error" is characterized by high-confidence predictions that fail to translate to new data.
- **Mitigation:** The **Small MLP (Small Step)** model effectively mitigated this by strictly regularizing the update speed. While it never achieved the theoretical highs of the large model, it maintained a consistent generalization gap, beating the baseline in the held-out set (1.59 vs 1.19).

6.6.2 Regime Dependency

Our systematic evaluation across multiple time windows revealed that the model's performance is highly sensitive to market regimes.

- **Strength:** The model excels in the most recent market conditions ("Last 500" days), where it achieves its highest risk-adjusted returns.
- **Weakness:** In the "Last 2000" window, the model's performance (0.59) dips below the Buy-and-Hold baseline (0.66). This suggests the model struggles to adapt to older market regimes that may differ significantly from the recent training data distribution (e.g., different volatility or interest rate environments).
- **Analysis:** This error indicates a lack of "regime awareness." The agent applies a single learned policy across diverse market conditions, failing to recognize when the market dynamics have shifted fundamentally.

6.6.3 Reward Signal Failure

Qualitative analysis of the training curves (Figure **??**) highlights a critical failure mode in our reward design. The models trained on the direct Competition Sharpe Ratio achieved a Sharpe of ≈ 0.2, effectively failing to learn.

- **Cause:** The Sharpe Ratio is a "sparse" and "delayed" metric—it requires a long history of returns to compute meaningfully. Optimizing for it directly at every timestep resulted in a noisy gradient signal that prevented the PPO agent from connecting specific actions to long-term risk-adjusted outcomes.
- **Correction:** The "Windowed Sign Reward" proved superior because it provided dense, immediate feedback (directional accuracy) while implicitly penalizing risk via the windowed drawdown penalty.

6.6.4 Future Error Mitigation

To specifically address the regime dependency and overfitting issues, future iterations would employ:

1. **Regime-Aware Ensembles:** Training separate agents for different volatility regimes (e.g., High Vol vs. Low Vol) and using a meta-controller to switch between them.
2. **Adversarial Training:** Introducing noise or adversarial perturbations to the input features during training to force the model to learn robust features rather than memorizing noise.

7 Reflection on Progress Plan

In our Progress Report, we outlined four primary objectives for the final phase of the project. Below is a reflection on our follow-through for each planned item:

- **Plan: Improve Reward Function.** We proposed incorporating risk-adjusted returns (e.g., Sharpe ratio) and additional risk controls.
- **Outcome:** *Followed through, but with a pivot.* We explicitly implemented a reward function based on the Competition Sharpe Ratio ("Sharpe LSTM" and "Sharpe RL Small"). However, as shown in Table 1, these models failed to learn effectively (Sharpe < 0.25). Consequently, we changed course to develop the "Windowed Sign Reward," which indirectly optimizes for risk-adjusted returns by penalizing sustained drawdowns. This pivot was necessary because the direct Sharpe reward signal proved too sparse and noisy for the PPO agent to optimize directly.
- **Plan: Experiment with Neural Architectures.** We planned to implement p-sLSTM and GRU architectures to capture temporal dependencies.

- **Outcome:** *Partially Followed through.* We successfully implemented an LSTM-based Actor-Critic agent ("Window LSTM"). This aligns with our hypothesis that temporal encoding was necessary for financial time series. However, due to time and complexity constraints, we did not explore p-sLSTM or GRU variants. Future work could extend this line of inquiry.

- **Plan: Implement Parallel Rollout Environments.** We intended to accelerate training by parallelizing the environment interaction.

- **Outcome:** *Changed course.* While we identified the Python rollout loop as a bottleneck, implementing true parallel environments with LSTM agents introduced significant complexity regarding hidden state management across disparate processes. Instead of parallel rollouts, we focused on optimizing the *update phase* via a "Block-Shuffling" mechanism. This allowed us to break sequential data into contiguous blocks for efficient batch processing while preserving the temporal integrity required for the LSTM. In theory, due to the special environment, we can theoretically turn everything into matrix operations, but due to time constraints, we did not implement this.

- **Plan: Complete Evaluation Pipeline.** We aimed to build a systematic analysis pipeline.

- **Outcome:** *Followed through.* We expanded our evaluation beyond simple MSE to include a robust testing framework that segments performance across "Last 500", "Last 1000", and "Last 1500" days. This granular analysis was crucial in revealing that while large-step models overfit the training data (First 1500 Sharpe: 7.14), simpler models provided better consistency on unseen data.

Team Contributions

Xiaotian Lou excelled in his contribution on fine-tuning different hyperparameters to achieve better results. He also contributed to the implementation of the reinforcement learning environment and agent.

Shike Chen contributed mostly on the implementation of different reward functions and the analysis of the results. He also helped with the implementation of the reinforcement learning environment and agent. He also participated in hyperparameter tuning.

Tianjiao Xiao mostly contributed on the implementation of reinforcement learning environment, baseline model and agent. She also helped with the analysis of the results. She also participated in hyperparameter tuning.

References

Z. Zhang, S. Zohren, and S. Roberts, "Deep reinforcement learning for trading," *The Journal of Financial Data Science*, vol. 2, no. 2, pp. 25–40, 2020.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

S. Sun, R. Wang, and B. An, "Reinforcement learning for quantitative trading," *ACM Transactions on Intelligent Systems and Technology*, vol. 14, no. 3, pp. 1–29, 2023.

G. Huang, X. Zhou, and Q. Song, "Deep Reinforcement Learning for Portfolio Management," *arXiv preprint arXiv:2012.13773*, 2020.

F. Soleymani and E. Paquet, "Deep Graph Convolutional Reinforcement Learning for Financial Portfolio Management: DeepPocket," *arXiv preprint arXiv:2105.08664*, 2021.

Y. Jiang, X. He, and W. Zhang, "Deep Reinforcement Learning for Portfolio Selection," *Decision Support Systems*, 2024. [In press].

V. Zakamouline and S. Koekebakker, "Portfolio performance evaluation with generalized Sharpe ratios: Beyond the mean and variance," *Journal of Banking & Finance*, vol. 33, no. 7, pp. 1242–1254, 2009.

T. S. Lam, A. Verma, B. K. H. Low, and P. Jaillet, "Risk-Aware Reinforcement Learning with Coherent Risk Measures and Non-Linear Function Approximation," in *Proc. ICLR*, 2023.