A Deep Reinforcement Learning and Predictive Architecture for Stock Portfolio Management

Ali Elhor
University of Waterloo
aaelhor@uwaterloo.ca

Raghav Vasudeva
University of Waterloo
r2vasude@uwaterloo.ca

Amin Ambike
University of Waterloo
asambike@uwaterloo.ca

Aditya Ajay
University of Waterloo
a2ajay@uwaterloo.ca

Bryan Deng
University of Waterloo
b33deng@uwaterloo.ca

Jesse Xia
University of Waterloo
jesse.xia@uwaterloo.ca

***Abstract*—This paper presents a deep reinforcement learning framework for stock portfolio management that integrates time-series forecasting with advanced graph representations. We employ a DeepAR module to provide predictive signals on future price movements and a Temporal Portfolio Graph (TPG) to capture inter-asset correlations. These enriched features are fed into a Proximal Policy Optimization (PPO) agent, enabling robust portfolio reallocation across diverse market conditions. Experimental evaluations from 2012 to 2024 demonstrate that our approach outperforms vanilla PPO and traditional market benchmarks, delivering higher returns and favorable risk-adjusted performance. The results underscore the effectiveness of combining predictive modeling and graph-based state representations for more informed, adaptable trading strategies.**

I. Introduction

Portfolio management is a method for retail and institutional investors alike to track and optimize their investments. Conventionally, portfolio management requires a high degree of human intervention and expertise, which favors those who have an inherent understanding of market patterns. As a result, investors without said expertise often fail to capture key market movements and adapt to changing conditions.

Despite RL succeeding in various sequential decision-making tasks, its application to financial markets remains challenging due to their stochastic and non-stationary nature. Additionally, RL models have traditionally failed to incorporate predictive models to guide decision-making, limiting their effectiveness in volatile market conditions. Existing literature that explores algorithms such as Deep Q Networks (DQN) and Proximal Policy Optimization (PPO) has struggled with addressing risk management and does not leverage time-series forecasting to improve decision-making.

This paper investigates how Deep Reinforcement Learning can be combined with predictive modeling to improve portfolio management strategies. We propose a portfolio management framework that integrates *DeepAR*, a time-series forecasting model, to guide a trading agent in making informed decisions. Our approach utilizes a PPO-based Reinforcement Learning agent that dynamically adjusts portfolio allocations using historical stock price data, with a DeepAR forecasting module and a *Temporal Portfolio Graph* (TPG) incorporated into its state representation to improve decision-making. By integrating predictive modeling with deep Reinforcement Learning, our approach aims to reduce volatility and maximize risk-adjusted returns despite changing market conditions.

A. Motivation

There is a growing need for automated portfolio management tools that can support investors with their decision-making abilities, minimizing human intervention. In recent years, Reinforcement Learning (RL) has emerged as a powerful tool for tackling decision-making problems. Its adaptability and robustness make it well-suited for problems involving temporal data. By training on historical market data, RL can potentially outperform conventional investment strategies by learning optimal trading strategies.

B. Related Works

Previous approaches to the application of RL for portfolio management have explored different RL algorithms and their varying effectiveness given the large state space and uncertainty of financial markets. However, recent work has instead placed emphasis on providing more diverse and contextual representations of high-dimensional financial data, along with clever mechanisms and slight modifications to vanilla RL algorithms to aid them in their learning.

One of the earlier attempts in 2019 by Yu et al. [1] integrated a prediction module and generative adversarial data augmentation into a model-based RL algorithm, aiming to mitigate data scarcity through synthesized time-series. In 2022, Yue et al. [2] expanded on risk mitigation by coupling a denoising autoencoder with an actor–critic RL framework, thereby enhancing stability in noisier markets. Building on these ideas, a 2023 study by Zou et al. [3] introduced a cascaded LSTM architecture, feeding extracted temporal features into a Proximal Policy Optimization (PPO) agent to capture richer dynamics. Later that same year, Yang et al. [4] proposed the TC-MAC (Task-Context Mutual Actor–Critic) method, which encodes both local asset features and global portfolio context; by maximizing mutual information between them, it emphasized inter-asset relationships to improve policy

robustness. Most recently, Li and Hai in 2024 presented a multi-agent deep RL system that fuses not only standard market quotes but also additional stock indices, highlighting the growing trend to incorporate diverse data sources [5].

C. Problem Definition

1) Reinforcement Learning: Reinforcement Learning (RL) is formulated as a Markov Decision Process (MDP), defined by the tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, where:

- $\mathcal{S}$ is the state space.
- $\mathcal{A}$ is the action space.
- $P(s'|s,a)$ is the transition probability function, representing the probability of transitioning to state $s' \in \mathcal{S}$ given the current state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$.
- $R(s,a)$ is the reward function, mapping state-action pairs to a real-valued reward.
- $\gamma \in [0,1]$ is the discount factor, determining the importance of future rewards.

The objective of the agent is to learn a policy $\pi : \mathcal{S} \to \mathcal{A}$ that maximizes the expected cumulative discounted reward:

$$J(\pi) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \mid s_0 = s\right]. \tag{1}$$

The state-value function under policy π is defined as:

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \mid s_0 = s\right]. \tag{2}$$

Similarly, the action-value function (Q-function) is:

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \mid s_0 = s, a_0 = a\right]. \tag{3}$$

The optimal policy π^* maximizes these functions, leading to the Bellman optimality equation:

$$Q^*(s,a) = \mathbb{E}\left[R(s,a) + \gamma \max_{a'} Q^*(s',a') \mid s,a\right]. \tag{4}$$

This framework serves as the foundation for various RL algorithms, including value-based, policy-based, and actor-critic methods.

2) Portfolio Management: Within this framework, portfolio management is performed using the continuous reallocation of funds among a fixed number of assets within the portfolio. The portfolio consists of a risk-free cash balance and m stocks such that a portfolio vector for a given time can be defined as:

$$\boldsymbol{w_t} = [w_{c,t}, w_{1,t}, ..., w_{m,t}], \tag{5}$$

where $w_{i,t}$ represents the percentage or weight of the portfolio's total value allocated to asset i for time t, and $w_{c,t}$ represents the percentage remaining as cash. This vector $\boldsymbol{w_t}$ corresponds directly to the action of the agent a_t, since it is re-determined by the agent at every timestep and can be adjusted to values between 0 (no funds allocated to this asset) and 1 (all funds allocated to this asset), given that $w_{c,t} + \sum_{i=1}^{m} w_{i,t} = 1$ for all t.

By the end of a trading day, price fluctuations then cause the weights to shift according to

$$\boldsymbol{w'_t} = \frac{\boldsymbol{u_t} \odot \boldsymbol{w_{t-1}}}{\boldsymbol{u_t} \cdot \boldsymbol{w_{t-1}}}, \tag{6}$$

where $\boldsymbol{u_t} = \frac{\boldsymbol{p_t}}{\boldsymbol{p_{t-1}}} = (1, \frac{p_{1,t}}{p_{1,t-1}}, \frac{p_{2,t}}{p_{2,t-1}}, ..., \frac{p_{m,t}}{p_{m,t-1}})$, and $\boldsymbol{p_{t-1}}$, $\boldsymbol{p_t}$ represent the closing prices of the stocks on the previous and current day respectively. The agent must then optimally reallocate the weights in the portfolio to the updated vector $\boldsymbol{w_t}$, accounting for transaction fees that shrink the portfolio by factor μ_t. This factor is referred to as the *transaction remainder factor* and is determined recursively using the method introduced in [6] and extended in [7].

We denote the value of the portfolio at the beginning of trading day t as v_{t-1} and its value at the end of the trading day as v'_t. The actual value of the portfolio at the end of the trading day, after reallocating funds and accounting for transaction fees, then becomes $v_t = \mu_t v'_t$.

We then define the logarithmic rate of return as

$$r_t = ln\frac{v_t}{v_{t-1}} = ln(\mu_t \boldsymbol{u_t} \cdot \boldsymbol{w_{t-1}}), \tag{7}$$

which allows the final portfolio value to be represented as a continuous reallocation problem defined by the equation

$$v_f = v_0 exp(\sum_{t=1}^{t_f+1} r_t) = v_0 \prod_{t=1}^{t_f+1} \mu_t \boldsymbol{u_t} \cdot \boldsymbol{w_{t-1}}. \tag{8}$$

3) Assumptions: Key assumptions are made to idealize the trading environment and make the approach to this problem more feasible.

1) **No Slippage:** We assume that even during after-hour markets (when the agent does the reallocation), all assets are liquid enough such that a trade can be carried out immediately and at the last price when an order was placed.
2) **No Market Impact:** The capital invested or liquidated by the agent is insignificant compared to the total volume of any traded asset and does not affect the market in any way.
3) **Fractional Shares:** All assets in the portfolio can be traded with fractional shares, making the portfolio vector representation feasible. This assumption is redundant in cryptocurrency markets but does not always hold true with traditional stocks, hence the importance in listing it here.
4) **No After-Hours Movement:** The prices of assets do not fluctuate during after-hours, which allows the closing prices of the previous day p_{t-1} to be treated as the opening prices of the current day.

II. Methodology

The architecture we propose aims to employ predictive models and advanced data representations to enrich the state information of the RL algorithm. We incorporate a prediction

module which leverages the *DeepAR* model for time-series forecasting, providing the RL agent with an independent evaluator and a predictive ability separate from that which it implicitly learns. The motivation behind this is that by offloading the learning needed for predictive decision-making, the agent can dedicate more resources towards optimizing asset re-allocations to maximize returns. We also introduce a *Temporal Portfolio Graph* (TPG) to provide a more comprehensive representation of the assets in the portfolio and their correlations. Finally, we use a popular RL algorithm known as *Proximal Policy Optimization* (PPO) as the agent outputting actions (updated portfolio weights) given a state, and a reward to maximize cumulatively.

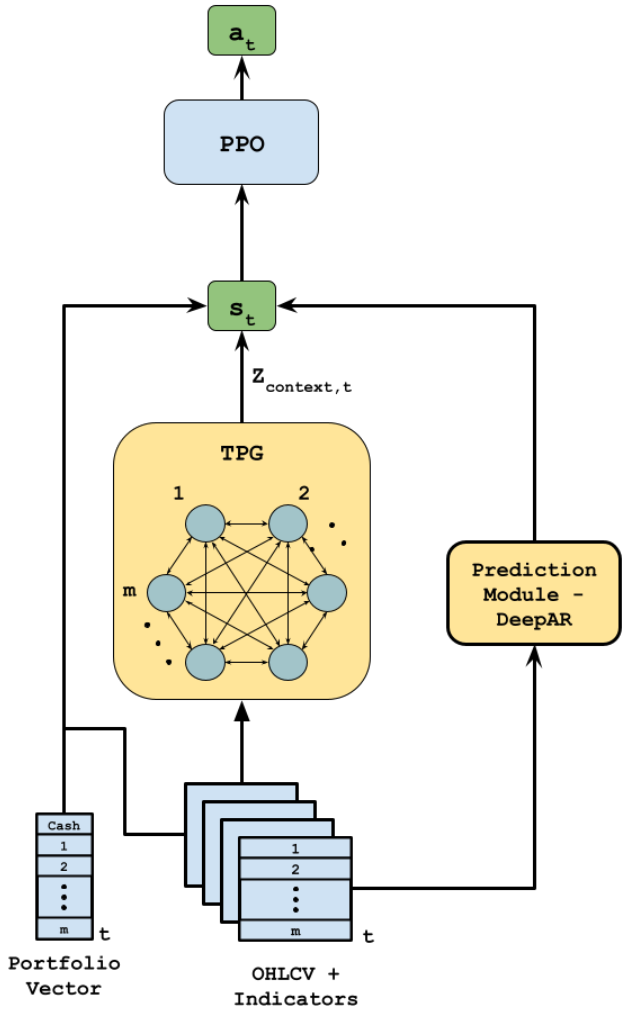

Fig. 1. Simplified model architecture, showing inputs to the TPG, DeepAR and the construction of the state used by the PPO algorithm

A. DeepAR Prediction Module

The DeepAR prediction module originates from research at Amazon Web Services (AWS) in 2017 [8]. The incorporation of the prediction module in this project serves a key purpose: the stock forecasting predictions are added to the state of the PPO RL model, adding more dimensionality and contextual information, which is useful in generating more accurate actions for the agent.

As its own module, the DeepAR model takes as input normalized OHLCV data (Open, High, Low, Close, Volume) for any number of stocks, obtained from YFinance, and then pre-processed. It returns μ and σ, corresponding to predictions with the Gaussian distribution.

The model architecture consists of probabilistic forecasting with a recurrent neural network (RNN), specifically a multi-layered LSTM network to learn the sequential dependencies in the stock data. The LSTM transforms the input into hidden representations, applying dropout between the layers. The LSTM outputs pass through a fully-connected sub-network consisting of: a linear layer that maintains the hidden size dimension, a ReLU activation function for non-linearity, and another linear layer that produces the final feature representation. The transformed features are then processed by two parallel output layers:

- The μ-layer projects features to generate mean predictions for each time step.
- The σ-layer generates unconstrained values that are passed through an exponential function to ensure positive standard deviations.

The predicted mean (μ) and standard deviation (σ) are used to compute the Negative Log-Likelihood (NLL) loss. Mathematically, the loss function is given by:

$$\mathcal{L}_{NLL} = \frac{1}{2}\log(2\pi\sigma^2) + \frac{(y-\mu)^2}{2\sigma^2} \tag{9}$$

The model implements data normalization using the mean and standard deviation of the training data, and performs denormalization when generating the final predictions. Training uses an Adam optimizer with early stopping based on validation loss improvement.

The model training process occurs for a specific time period designated by the user: $t_t - t_2$. Then, using the trained model, it predicts prices for the time period $t_2 - t_3$, where t_3 is designated by the user. Predictions are performed with a sliding-window mechanism.

We divide the total prediction time-period into a series of windows with a fixed length (window length = 7). For each window, $windowlength - 1$ days are used as input, and the model predicts the price for the next day. This continues for the entire prediction time-period, ultimately allowing us to predict stock prices for one day beyond the designated prediction time-period.

To incorporate these predictions into the PPO agent's state, the model was trained on 6 years' of data from 2012-2018, and its predictions from 2018-2024 served as an additional element in the state representation when training the PPO.

B. Temporal Portfolio Graph with Graph Attention Networks

In addition to the DeepAR module, our framework incorporates a Temporal Portfolio Graph (TPG) similar to Yang et al. [4] to capture the evolving relationships among all

assets in the portfolio at each time step. By encoding each asset as a node and forming edges based on similarity or correlation measures, the TPG provides a graph-based view of the portfolio's internal structure. Unlike prior works that leveraged Graph Convolutional Networks (GCN) followed by attention-based pooling, we adopt a **Graph Attention Network (GAT)** to learn both node embeddings and attention scores end-to-end. Below is a concise overview of how this TPG module operates:

a) Node Features.: At each time t, each asset i is associated with a feature vector that includes recent price movements and technical indicators. Concatenating these yields $\mathbf{x}_{i,t}$ as the node feature for asset i.

b) Edges and Attention.: In a standard GCN-based approach, one would compute an adjacency matrix $\mathbf{B}^t$ via a heat-kernel or thresholded correlation. In *GAT*, by contrast, we start with a fully connected graph with self-loops, and then *learn* attention coefficients α_{ij}:

$$\alpha_{ij} \;=\; \mathrm{softmax}\Big(\sigma\big(\mathbf{a}^\top[\,\mathbf{W}\mathbf{x}_{i,t} \,\|\, \mathbf{W}\mathbf{x}_{j,t}\,]\big)\Big),$$

where $\mathbf{a}$ and $\mathbf{W}$ are learnable parameters, $\|$ denotes concatenation, and σ is a suitable activation (e.g. LeakyReLU). These attention coefficients effectively dictate how much information each neighbor j contributes to node i's updated embedding.

c) Propagation.: Within each GAT layer, node i's next embedding is then aggregated using the attention scores:

$$\mathbf{z}_{i,t} \;=\; \sigma\Big(\sum_{j\in\mathcal{N}(i)} \alpha_{ij}\,\mathbf{W}\,\mathbf{x}_{j,t}\Big).$$

Stacking multiple GAT layers yields increasingly higher-level representations. We omit certain details here for brevity; one can refer to the original GAT paper for full derivations.

d) Global Pooling.: Finally, to obtain a single "global context" vector for the *entire* portfolio, we use an attention-pool over the node embeddings:

$$Z_{\text{context},t} \;=\; \sum_{i=1}^{m} \beta_{i,t}\,\mathbf{z}_{i,t}, \quad \beta_{i,t} \;=\; \mathrm{softmax}\big(\omega_{i,t}\big),$$

where $\omega_{i,t}$ is an attention score for node i. This global vector $Z_{\text{context},t}$ is concatenated to other features in the RL agent's state representation.

e) Motivation for GAT vs. GCN.: Our decision to replace the GCN+attention-based scoring with a *single GAT module* stems from several considerations:

- **Learned Adjacency:** Rather than using a fixed or thresholded similarity matrix, GAT dynamically infers how strongly each asset should attend to every other asset.
- **End-to-End Training:** By combining graph convolutions and attention into one framework, the network can directly optimize relevant attention scores for downstream tasks (e.g. RL policy learning).
- **Reduced Complexity:** Collapsing the two-step pipeline (GCN + separate global attention) into a unified GAT architecture can simplify hyperparameter tuning and code maintenance while retaining strong expressive power.

In practice, GAT layers handle the portfolio's evolving relationships effectively, as each node embedding focuses more on those neighbor assets that *matter most* at time t. This synergy between graph attention and RL ultimately improves policy robustness by highlighting the dependencies among assets in a data-driven manner. Detailed equations for multi-head attention and skip connections in GAT can be found in the original paper [9], which we do not reproduce here for brevity.

C. Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) is a prominent policy-gradient algorithm proposed by Schulman et al. [10] as a simplified, yet robust, alternative to Trust Region Policy Optimization (TRPO). PPO optimizes policies through an objective function with a clipping mechanism that constrains policy updates, preventing drastic deviations from the previous policy, ensuring stability and efficiency during training. This approach strikes an ideal balance between computational simplicity, sample complexity, and empirical performance in diverse reinforcement learning tasks. The core of PPO is its clipped surrogate objective function:

$$L_{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t \left[\min\left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right], \tag{10}$$

where the probability ratio is given by

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}, \tag{11}$$

and $\hat{A}_t$ denotes the advantage function at time step t. This function explicitly constrains how significantly the policy can update, addressing the instability associated with large policy updates in conventional policy gradient methods. The clipping parameter ϵ critically controls the magnitude of allowable policy updates. A larger ϵ value permits a larger trust region with more volatile adjustments in policy updates, potentially leading to drastic portfolio reallocation and increased portfolio volatility. Conversely, a smaller ϵ stabilizes policy adjustments, promoting consistent but potentially overly cautious updates. Optimizing this hyperparameter is crucial to balancing responsiveness and stability.

1) Application to Portfolio Management: For our stock portfolio management problem, we leveraged PPO's capacity to handle high-dimensional continuous state and action spaces effectively. We constructed a sophisticated state representation incorporating multiple dimensions for each asset and the risk-free cash component. Specifically, the state included:

- Historical normalized OHLCV data with a 20-day look-back window for each asset.
- A non-risky cash asset as part of the portfolio.
- Technical indicators, specifically the Relative Strength Index (RSI), due to empirical performance gains in preliminary experiments.
- Current portfolio allocation weights.
- Predictions from the DeepAR forecasting module for next-day closing prices, when the prediction module was incorporated.

- The context vector from the TPG representing the correlation between assets in the portfolio.

For our experiments, we chose the five stocks: AMZN, TSLA, AAPL, MSFT, GOOG. This resulted in a comprehensive state representation with dimensionality between 606 and 611, dependent on the inclusion of predictive forecasts.

2) Training Nuances: Our training process included several specific considerations to ensure robust generalization:

- **Data Period**: Without predictions, our PPO model trained over a period of 12 years (2012-2024). When integrating DeepAR, the forecasting model was trained from 2012-2018 and PPO utilized predictions from 2018-2024.
- **Randomized Batches**: Training occurred in batches of 252 trading days (1 year), with each batch initiated from a random date within the specified time period. This randomization prevented the model from overfitting historical price sequences, thereby promoting a generalized strategy.
- **Episodes and Timesteps**: PPO training lasted 1 million timesteps, ensuring sufficient exposure to diverse market conditions.

3) Observed Behavior and Insights: A notable observation during testing was the model's tendency to identify and adhere to an "optimal" allocation distribution for assets. After initial convergence, portfolio weights typically remained within ±5% of their starting allocation throughout the testing period. This behavior indicated the model's learned stable investment strategy, focusing on controlled risk exposure rather than aggressive daily trading. The application of PPO in conjunction with predictive modeling (DeepAR) and advanced state representation (TPG) significantly improved the risk-adjusted returns by providing enhanced state information. Predictive insights allowed the agent to anticipate market trends and allocate resources accordingly, thus yielding higher returns with reduced volatility compared to traditional RL methods and standard market indices. Our observations validate the effectiveness of PPO in dynamic environments like stock markets, particularly highlighting its adaptability, efficiency, and practical suitability for real-world financial decision-making.

III. Results

The primary metric used to evaluate the performance of the model is the portfolio's value over time, which can also be defined as the cumulative returns. The model was tested in bullish and bearish markets, with the bullish market occurring from 01-01-2024 to 01-01-2025, and the bearish market occurring from 01-19-2020 to 04-30-2020. As can be seen in 2 and 3, by the end of the time period the model outperformed the equivalent of investing all capital into the S&P 500 index, and a vanilla PPO implementation in both market conditions. However, the model seems to be more capable of capitalizing on periods of upturn than precisely managing funds for minimal losses during periods of downturn. This is demonstrated by the model considerably outperforming the S&P 500 portfolio when market conditions begin to rise, while only performing similar when conditions begin to decline.

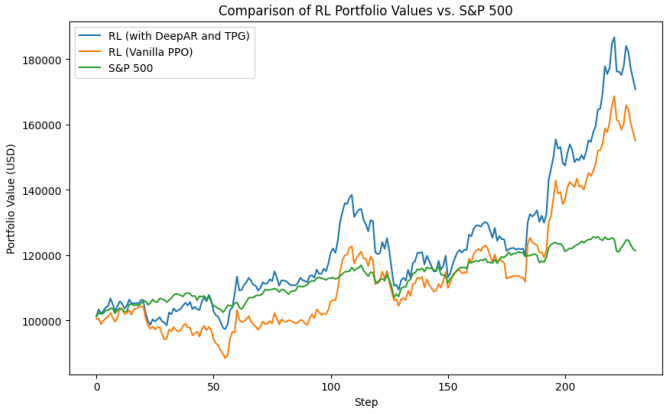

Fig. 2. Comparison of RL portfolio values vs. S&P 500 index during bullish market

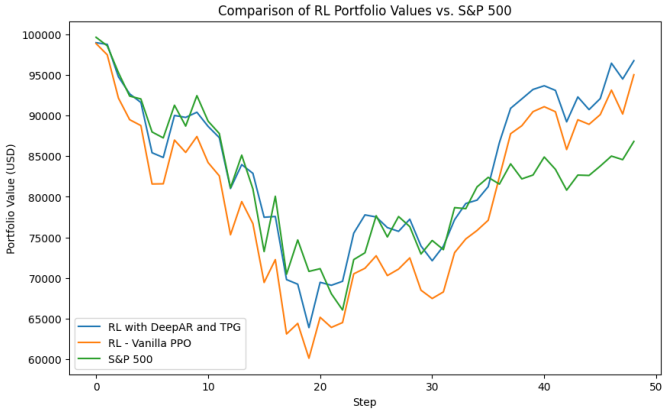

Fig. 3. Comparison of RL portfolio values vs. S&P 500 index during bearish market

We also consider the average Sharpe ratio (SR) for all three portfolios over their given periods. The Sharpe ratio is a widely-used financial metric that measures the excess return being received for the amount of volatility or risk taken on by an investment. It can be calculated as follows:

$$SR = \frac{R_p - R_f}{\sigma_p}, \tag{12}$$

where R_p is the portfolio return, R_f is the risk-free rate, and σ_p is the standard deviation of portfolio returns. Generally, a higher SR indicates better risk-adjusted performance, meaning a greater return per unit of risk. From the tables below, it can be observed that our model has a lower SR than the S&P 500 portfolio in the bullish market but a much higher SR in the bearish market. This reflects the behavior of the model, as it trades more aggressively with less concern for risk during upturns while trading more conservatively during

downturns. This would conventionally represent the ideal behavior of a trading agent. That is, to maximize profits while simultaneously minimizing losses.

TABLE I
FINAL VALUES OF RL VS. S&P 500 PORTFOLIOS DURING BULLISH MARKET

Portfolios	**Final Value ($)**	**Average SR**
S&P 500	120,470	2.635
Vanilla PPO	155,055	1.683
PPO (DeepAR & TPG)	170,742	2.039

TABLE II
FINAL VALUES OF RL VS. S&P 500 PORTFOLIOS DURING BEARISH MARKET

Portfolios	**Final Value ($)**	**Average SR**
S&P 500	87,285	0.461
Vanilla PPO	95,009	1.681
PPO (DeepAR & TPG)	96,740	1.466

IV. CONCLUSION

There are numerous ways this research can be extended to produce even more sophisticated and profitable trading agents. As markets become increasingly captured by the interconnectedness of the media and global macroeconomics, this problem cannot be formulated or solved traditionally. The inclusion of live sentiment analysis of ongoing public emotions, news releases, and financial statements has become crucial to predicting future market trends and day-to-day swings, as passing statements from influential figures can determine the trajectory of certain market assets for the foreseeable future. Incorporating sentiment analysis into future iterations would be a software engineering problem that would involve the retrieval of relevant sources, the use of NLP, the integration of multi-modal data, and a way to incorporate this scattered data into the training pipeline of a sequential RL model.

Further improvements can also be made to the prediction module itself, as it is meant to act as an independent evaluator for the RL agent's use. By employing additional predictive models capable of modeling posterior distributions, such as Variational Inference (VI) and Temporal Fusion Transformers (TFTs), the prediction module's robustness and overall predictive ability could increase.

Finally, the TPG's representative ability can potentially be improved by replacing the GAT with a Temporal Graph Network (TGN). GCNs and GATs assume static graphs, with no inherit notion of time or event sequences. Conversely, TGNs capture continuous graph changes with nodes that are updated through memory vectors. This would allow the nodes to more accurately model the time-series nature of assets in the portfolio and their time-varying correlations.

REFERENCES

[1] P. Yu, J. S. Lee, I. Kulyatin, Z. Shi, and S. Dasgupta, "Model-based Deep Reinforcement Learning for Dynamic Portfolio Optimization," 2019.

[2] H. Yue, J. Liu, D. Tian, and Q. Zhang, "A Novel Anti-Risk Method for Portfolio Trading Using Deep Reinforcement Learning," 2022.

[3] J. Zou, J. Lou, B. Wang, and S. Liu, "A Novel Deep Reinforcement Learning Based Automated Stock Trading System Using Cascaded LSTM Networks," 2023.

[4] S. Yang, "Deep Reinforcement Learning for Portfolio Management," 2023.

[5] H. Li and M. Hai, "Deep Reinforcement Learning Model for Stock Portfolio Management Based on Data Fusion," 2024.

[6] Ormos, Mihály and Urbán, András, "Performance Analysis of Log-optimal Portfolio Strategies with Transaction Costs," 2011.

[7] Z. Jiang, D. Xu, and J. Liang, "Deep Portfolio Management: A Deep Reinforcement Learning Framework for the Financial Portfolio Management Problem," 2017.

[8] David Salinas, Valentin Flunkert, Jan Gasthaus, "DeepAR: Probabilistic Forecasting with Autoregressive Recurrent Networks," 2019.

[9] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph Attention Networks," 2018.

[10] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," 2017.