Building a Reward based effective Portfolio Management strategy using a Twin Delayed Deep Deterministic Policy Gradient (TD3)

Chintan Shah

Nomenclature

Symbol	Meaning	Defined
d_{model}	Transformer embedding dimension	4.1.1
$\mathbf{h}_t^i$	Hidden embedding of token i at transformer layer t	4.1.1
$\text{PE}_{p,2k}$, $\text{PE}_{p,2k+1}$	Sinusoidal positional encoding at position p, dims $2k/2k+1$	4.1.1
m	Momentum in dynamic gating	4.1.2
β	Transformer gate-smoothing weight	4.1.2
α	Transformer gating weight (also Sharpe-bonus weight)	4.1.2; 4.6.2(c)
h	Number of attention heads	4.1.3; 4.1.4
W_h^Q, W_h^K, W_h^V, W^O	Projection matrices for multi-head attention	4.1.3; 4.1.4
$\text{softmax}(\cdot)$	Softmax activation used in attention	4.1.3
L_{recon}	Transformer reconstruction loss (MSE)	4.1.7
L_{ctr}	Contrastive (InfoNCE) loss	4.1.7
CTR	Contrastive loss weight λ_{ctr}	4.1.7
π	Policy mapping states to actions	4.2
$J(\pi)$	Expected discounted return under π	4.2
γ	Discount factor for future rewards	4.2; 4.2.11
a_t	Action at time t (weights, multipliers, etc.)	4.2.6
s_t	State vector at time t (market + portfolio obs.)	4.2.6
r_t	Reward at time t from environment	4.2.6
$V^\pi(s)$	State-value function	4.2.7
$Q^\pi(s,a)$	Action-value function under π	4.2.8

(continued on next page)

(Table continued)

Symbol	**Meaning**	**Defined**
L_{Q_i}	Critic loss: $\frac{1}{B}\sum(Q_i - y_i)^2$	4.2.13
L_μ	Actor loss: $-\frac{1}{B}\sum Q_{\phi_1}$	4.2.13
τ	Soft-update coefficient	4.2.13
ϵ	Exploration noise	4.2.13
D	Replay buffer	4.2.13
δ	Small constant to avoid div-zero	4.2.13
$J_k(\pi)$	Return over horizon k (weekly, monthly, yearly)	4.5
$J_{\text{global}}(\pi)$	Global objective before constraints	4.5
$J_{\text{local}}(\pi)$	Local objective within pacing window	4.5
k	Index over pacing horizons	4.5
λ_k	Lagrange multiplier for constraint k	4.5
τ_k	Target threshold for return on horizon k	4.5
$\mathcal{L}(\pi, \lambda)$	Lagrangian: $J_{\text{global}} - \sum_k \lambda_k[\tau_k - J_k]$	4.5
(π^*, λ^*)	Saddle-point of Lagrangian	4.5
y_i	TD target: $r_i + \gamma Q_{\phi'}$	4.6.1
$P_{t,i}$	Price of asset i at time t	4.6.2(a)
$r_{t,i}$	Return of asset i: $(P_{t,i} - P_{t-1,i})/P_{t-1,i}$	4.6.2(a)
R_t	Portfolio return: $\sum_i w_{t,i} r_{t,i}$	4.6.2(b)
r_t^{env}	Env. reward: $R_t - \text{pen}_t + \alpha S_t$	4.6.2(c)
pen_t	Lagrangian penalty: $\sum_k \lambda_k[\tau_k - J_k]$	4.6.2(c)
S_t	Sharpe ratio at time t	4.6.2(c)
e_{weekly}	Weekly return for penalty calc.	4.6.2(c)
η_{weekly}	Scaling factor for weekly penalty	4.6.2(c)

1 Abstract

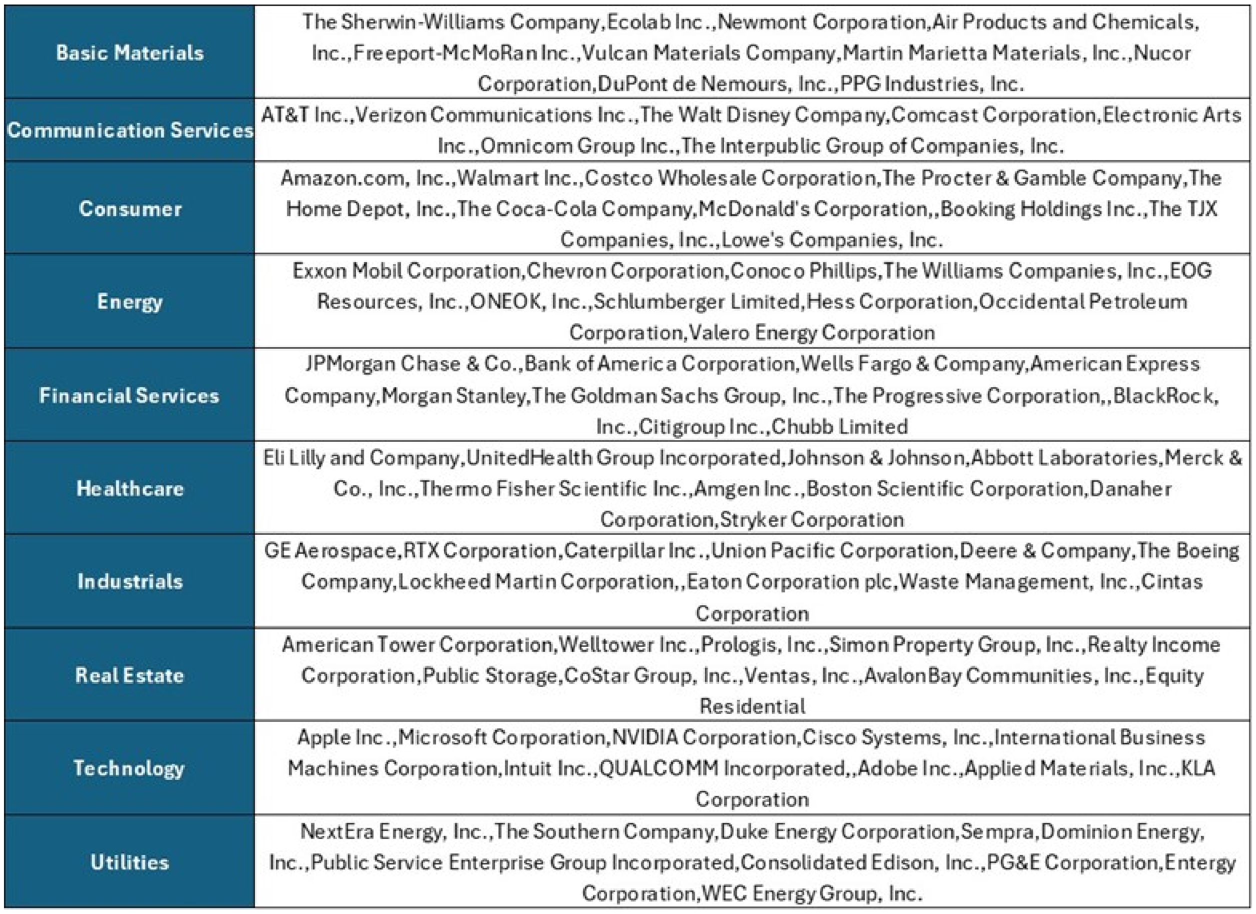
Financial Portfolio Management is seeing yet another evolution in terms of different models being introduced. We have classical models, quantitative models, and now we have a new wave of AI driven models. Deep Reinforcement learning models have taken huge strides in advancing portfolio management techniques. However, these methods have their own limitations and are primarily focused on risk mitigation strategies. While risk mitigation is paramount to portfolio management, there is hardly any research done on exploring rewards. Traditional use of Lagrangian in RL framework also has been to mitigate risk. While we implement the Lagrangian to mitigate risk, we also introduce the usage of Lagrangian methods to explore higher rewards when we have exceeded expectations. We have built a TD3 framework to explore the constraints of Lagrangian method. We introduce a Reward – Driven – Risk framework where we have an Actor – Critic algorithm which encourages our agent to explore the positive rewards spaces when the agent exceeds expectations. The framework helps to maximize rewards while maximizing returns for per unit amount of extra risk.

2 Introduction

Portfolio and Investment management has been traditionally lagging in adoption of Machine Learning and AI generated strategies for a long time. Quatitative or ML driven strategies are often perceived as blackboxes. AI driven strategies are outperforming humans. They manage risk better and generated higher risk adjusted returns despite having more concentrated portfolio as per Chen, R. (2022). Classical Machine Learning methods like Regression (Ridge and Lasso), Tree base methods (like Random Forests and Gradient Boosting), classification tasks using SVM and many other methods have taken giant stride in solving different aspects of financial portfolio optimization. In Mirete-Ferrer et al. (2022), they discussed how unsupervised algorithms like k-means and clusterting algorithms like PCA and Hierarichal Clustering reduce the overfitting problem caused by CAPM style regression models. Recent advances in portfolio management include implementation of Naïve Bayes and Kernel Based Methods to generate higher alpha compared to benchmarks. Methods like Bagging and Boosting have been found to improve forecasting robustness and reduce overfitting. However, we are seeing even more complex and advanced methods making their foray into Active financial portfolio management. In Guidolin, 2024 et.al., they discuss how techniques like Natural Language Processing (NLP) have been heavily implemented in the financial sector. There have even been efforts to merge the traditional concepts of investing with modern AI strategies. In Gang Hu, 2024, et. al., they have explored combining traditional strategy like Markowitz Portfolio Optimization with Reinforcement Learning framework. There have been many advancements in NLP and Reinforcement Learning techniques which have found unique applications in the financial domain. NLP has been a great tool to help aid other portfolio management methods. The output of NLP has augmented the data quality of inputs which have enriched other classical as well as modern Quant and AI driven portfolio management strategies. In Malandri et al. 2018, they discuss how public mood led to an increase in profitability by 5 % - 10 %. Reinforcement Learning is another area which has been gaining traction in financial portfolio management. The attraction for RL lies in the fact that the concept of RL can help simulate portfolio management exercises in a very realistic but innovative way. RL comes close to mimicking an actual portfolio manager. RL's functionality to simulate realistic scenarios and agents acting as portfolio managers are the perfect aid to a portfolio manager. We propose an innovative Transformer based TD3 method to actively manage portfolio over a long horizon. We are taking the advanced capabilities of Transformers to enhance the input data quality before feeding the enhanced data into a TD3 algorithm for a multi-target portfolio management experiment. As with any RL algorithm, we set an objective function which is defined in the reward function. The reward function is comprised of daily returns, penalties, and Sharpe ratios. Reward function determines the 'success' for the agent. It determines whether learning converges and whether the problem is solved. There are various aspects to rewards and various objectives of reward designing. Rewards determine objective specification and help in credit assignments. Our objective is constrained bound. Our objective is Reward – Driven – Risk. In finance and at large, risk is an important aspect to determine any course of action. Risk driven policies are a cornerstone of modern portfolio management. In Bielecki et al. 2023 they explore Dynamic Risk Convex measures and Distortion functions, with focus on risk measures generated by distortion functions in a dynamic discrete setup. In Moresco et al. 2023, they explore dynamic uncertainty sets for multistage portfolio optimization. In Righi, 2024 formulate penalty functions for worst case risk scenarios using Wasserstein-ball uncertainty sets, providing closed-form dual representations that facilitate real-time portfolio stress testing. Most of the research in Financial RL (and otherwise) focuses on incorporating risk to drive strategies. This Risk driven reward structure has been heavily explored and implemented. Conventional wisdom along with a plethora of research states that minimizing risk must be an integral part of reward seeking. Our approach is different in the sense that instead of constantly mitigating risk we encourage risky behaviour (under certain constraints). We have devised a Reward-driven-risk approach where we encourage the agent to pursue more controlled risky actions when we exceed certain expected targets.

3 Data Preparation

We are working with SP 500 stocks data. We took the top 10 companies (by market cap) in each sector and created a stock universe. We got the daily closing prices, the daily volume, and 10-day, 30-day, and 90-days volatilities data. The time frame for the data is from 1st January, 2000 to 31st December,



Basic Materials	The Sherwin-Williams Company,Ecolab Inc.,Newmont Corporation,Air Products and Chemicals, Inc.,Freeport-McMoRan Inc.,Vulcan Materials Company,Martin Marietta Materials, Inc.,Nucor Corporation,DuPont de Nemours, Inc.,PPG Industries, Inc.
Communication Services	AT&T Inc.,Verizon Communications Inc.,The Walt Disney Company,Comcast Corporation,Electronic Arts Inc.,Omnicom Group Inc.,The Interpublic Group of Companies, Inc.
Consumer	Amazon.com, Inc.,Walmart Inc.,Costco Wholesale Corporation,The Procter & Gamble Company,The Home Depot, Inc.,The Coca-Cola Company,McDonald's Corporation,,Booking Holdings Inc.,The TJX Companies, Inc.,Lowe's Companies, Inc.
Energy	Exxon Mobil Corporation,Chevron Corporation,Conoco Phillips,The Williams Companies, Inc.,EOG Resources, Inc.,ONEOK, Inc.,Schlumberger Limited,Hess Corporation,Occidental Petroleum Corporation,Valero Energy Corporation
Financial Services	JPMorgan Chase & Co.,Bank of America Corporation,Wells Fargo & Company,American Express Company,Morgan Stanley,The Goldman Sachs Group, Inc.,The Progressive Corporation,,BlackRock, Inc.,Citigroup Inc.,Chubb Limited
Healthcare	Eli Lilly and Company,UnitedHealth Group Incorporated,Johnson & Johnson,Abbott Laboratories,Merck & Co., Inc.,Thermo Fisher Scientific Inc.,Amgen Inc.,Boston Scientific Corporation,Danaher Corporation,Stryker Corporation
Industrials	GE Aerospace,RTX Corporation,Caterpillar Inc.,Union Pacific Corporation,Deere & Company,The Boeing Company,Lockheed Martin Corporation,,Eaton Corporation plc,Waste Management, Inc.,Cintas Corporation
Real Estate	American Tower Corporation,Welltower Inc.,Prologis, Inc.,Simon Property Group, Inc.,Realty Income Corporation,Public Storage,CoStar Group, Inc.,Ventas, Inc.,AvalonBay Communities, Inc.,Equity Residential
Technology	Apple Inc.,Microsoft Corporation,NVIDIA Corporation,Cisco Systems, Inc.,International Business Machines Corporation,Intuit Inc.,QUALCOMM Incorporated,,Adobe Inc.,Applied Materials, Inc.,KLA Corporation
Utilities	NextEra Energy, Inc.,The Southern Company,Duke Energy Corporation,Sempra,Dominion Energy, Inc.,Public Service Enterprise Group Incorporated,Consolidated Edison, Inc.,PG&E Corporation,Entergy Corporation,WEC Energy Group, Inc.

Figure 1: Stock Universe

2024. We split the data into two parts, with data from 1st January, 2000 to 31st December, 2014 for the pre-training the transformer. We then took the data from 2015 to 2024 for our training, validation, and testing of the RL algorithm.

3.1 Log Returns

We performed Log of the Returns as part of the data preparation process. We need to perform standardization on the data otherwise we would have the following issues within our data.

3.1.1 Additivity across periods

Simple (arithmetic) returns compound multiplicatively. This cannot decompose multi-period growth as sum of simple period returns. This breaks down the portfolio attribution and harder to fit to time-series models. In Knox, et al. (2024) they show that log returns have preferable numerical-stability and additivity properties.

3.1.2 Skewness and heavy-tails in price heavy distribution

Raw price return exhibit 'fat-tails' and leptokurtosis. This is far from Gaussian distribution which is preferable for time series modeling as it leads to unstable parameter estimates and poor generalization.

3.1.3 Heteroskedasticity

Variance of price grows with level of prices, making raw – returns having non-constant variance. This violates the assumption of homoscedasticity. This leads to biased volatility forecasts.
We solve all of the above problems by taking the log of the prices, volume, and volatilities.

3.2 Feature Standardization

We performed Standardization as part of the data preparation process. We need to perform standardization on the data otherwise we would have the following issues within our data.

3.2.1 Disparate feature scales

When one or more features have a large numeric scale, they dominate the gradients. Models over attend large scale features and they dominate the computations, ignoring smaller scaled features.

3.2.2 Hessians gradients

Non-standardized inputs create ill-conditioned Hessians gradients and converge slowly. Training Neural Networks can take a lot more epochs and make it more sensitive to learning rates.

3.2.3 Distorted Regularization

L1/L2 regularization has assumptions of comparable parameter scales, where they penalize large-scale features more than the small-scale features. The nearest neighbors become skewed towards high variance dimensions. In our data, the daily stock trading volume of our securities would shoot up in millions causing a large dominance in the data. Without standardization all our results would be dominated by Volume.
We first perform the log transformation and standardization on the data we use to pre-train the transformer and save the standardization weights. It is imperative that our data used in the RL algorithm be standardized on the same scale as the transformer pre-training data.

4 Methodology

In our portfolio management strategy, we have an agent who trades daily (per time step actions). In real world scenarios, investors often want maximum returns in lieu of least possible risk. We set expected annual targets for the agent, and then the agent is encouraged to take extra risks when they exceed expectations. Our aim is to devise a strategy where we want to have a profit buffer (we can take rewards). Once we have excess profit/rewards, we would like to encourage the agent to take extra risk to explore for more profit. We have broken down the long-term target (Global target) into smaller targets (Local targets). This results in a Bi-Level Optimization problem which we solve using Lagrangian Method.

$$\max_{\pi} J_{\text{global}}(\pi)$$
$$\text{subject to } J_{\text{local}}(\pi) \;\geq\; \tau_{\text{local}},$$

or equivalently, the Lagrangian relaxation

$$\max_{\pi} \; J_{\text{global}}(\pi) \;-\; \lambda_L \max\bigl\{0,\; \tau_{\text{local}} - J_{\text{local}}(\pi)\bigr\}.$$

The Lagrangian method would be used to make sure that we follow the local constraints and there would be penalties for breaking the constraints. We have different parameters to manage different aspects of our framework.

Our method is comprised of two phases. In the first phase we build a custom transformer which acts as multi-headed transformer. We have a Positional Encoder, Temporal Attention head, Spatial Attention head, and a Gating Mechanism. This way we ensure we get temporal, spatial, and time-dimensional information. We then use this custom transformer to pre-train our asset universe. We then use the output obtained i.e. embedding weights to train our input data for the RL algorithm. We obtain market embedding which duly captures the inter asset and intra asset market relationships. We create an embedding laden universe of assets. We capture time based and cross asset-based relationships using the pre-trained transformer. We then join that data with the standardized log transformed data for the observation space for the TD3 algorithm.

4.1 Transformers

Before 2017, translation tasks relied on sequence-to-sequence modeling. They relied on Recurrent (RNN/LSTM) networks or Convolution Neural Networks (CNN) within an encoder-decoder configuration while using attention mechanism only to bridge the encoder and decoder representations. However, due to the sequential nature of Recurrent networks, it limited parallel computations. Recurrent networks also cause bottlenecks as the layers go deep. Convolution networks need to be stacked deeply to capture context beyond local space. Vaswani et al. (2017) brought attention to the forefront with their Transformer model. They dispensed Recurrent and Convolution networks all together and built a model with attention mechanisms. Their model formed the basis of the modern plethora of Language models like GPT. A Transformer consists of an Encoder stack and Decoder stack. Encoder maps the input of tokens to a continuous sequence representation. Decoder generates the output sequence one token at a time, attending to the encoder's output and the previously generated tokens.

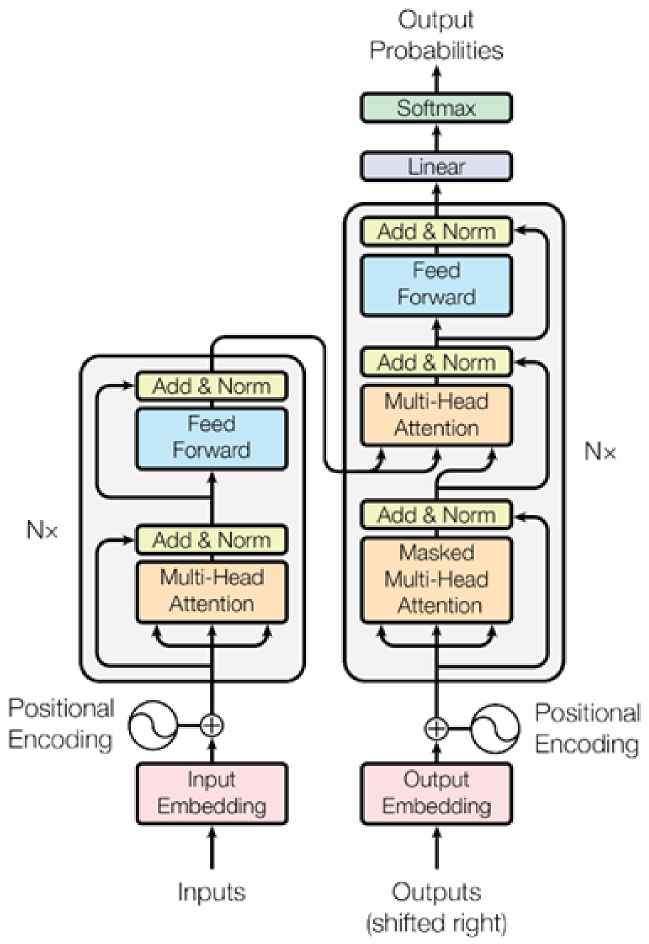


Figure 2: Transformer Architecture

We have built a custom transformer which encodes our data into market embeddings. Transformers were introduced to understand the complexity of

language modeling. Transformers have come a long way from being used for Language Modeling (Vaswani, et al. 2017), to Vision domain with ViT (Dosovitskiy et al. 2021), to Decision Transformer for RL Sequence Modeling (Chen et al. 2021). However, there has been no large scale pre-transformer trained specifically on financial stock market data. We built a custom transformer based on the MASTER transformer model developed by Li et al. (2023). Unlike traditional transformer architectures that primarily focus on sequential inputs like text data, MASTER is designed to capture inter-stock spatial relationships and capture intra-stock temporal relationships. This dual attention structure helps capturing complex financial market structures which are layered with quantitative model based on decisions and human driven actions.
Our TD3-based portfolio management model comprises the following key stages:

- **(a)** **Positional encoding:** Representation stage to incorporate temporal information.
- **(b)** **Input representation and feature gating:** Input feature selection based on gating recent market conditions.
- **(c)** **Multi-head temporal attention:** Intra-stock (security) pattern capture.
- **(d)** **Multi-head spatial attention:** Inter-stock (security) pattern capture.
- **(e)** **Position-wise feed-forward networks:** Sequence information capturing stage.
- **(f)** **Final temporal aggregation:** Compresses sequential information into aggregate embeddings.

4.1.1 Positional Encoding

Positional encodings follow the original architecture of Vaswani et al. (2017). They are added to the original input to add in the positional information in a sinusoidal space and get the time position of the input. For position 'pos' and dimension 'i', the positional encodings are given by

$$\mathrm{PE}_{p,2k} = \sin\!\left(\frac{p}{10000^{2k/d_m}}\right), \qquad \mathrm{PE}_{p,2k+1} = \cos\!\left(\frac{p}{10000^{2k/d_m}}\right).$$

where d_m represents embedding dimension. Transformers are time agnostic, so to better represent the time space, we inject the sinusoids for each projected feature. These sinusoids help build a unique positional pattern for each feature and for each time step. The model learns to interpret these sinusoids, helping it understand the relative and absolute positions in the sequence. In Ding et al. (2020), they demonstrated that these sinusoidal encodings helped the model to extrapolate the sequence lengths which is helpful in time series where time horizons vary.

4.1.2 Dynamic Gating Mechanism

We improved upon the gating mechanism of MASTER by introducing an innovative momentum-based gating mechanism that dynamically weighs input features based on the current market context. This addresses a key challenge in financial modeling; the importance of different features varies with market conditions. With our structure we capture the importance of features based on time. We start with a concatenation of our feature inputs and form a gate input vector, G. Using that we compute a current gate.

$$\begin{aligned}
W_n^{\text{curr}} &= \text{softmax}\Big(\frac{W_g\, g_n + b_g}{T_\beta}\Big), \\
W_n^{\text{dyn}} &\leftarrow m\, W_n^{\text{dyn}} + (1-m)\, W_n^{\text{curr}}, \\
W_n^{\text{out}} &= \alpha\, W_n^{\text{curr}} + (1-\alpha)\, W_n^{\text{dyn}}, \\
\widetilde{x}_{n,t,i} &= x_{n,t,i}\, W_{n,i}^{\text{out}}\, .
\end{aligned}$$

By smoothening the gate, we avoid jumpy feature weighting and give the network a little temporal inertia in how it handles attention in different input dimensions.

4.1.3 Multi-head Temporal Attention

This is the module where parallelism kicks in. Instead of performing a single attention function with dmodel dimensional keys, values, and queries, they are projected into h-dimensions (parallel) learned into dk, dv, dq respectively. The attention function is performed in parallel on each of these dimensions, producing dv dimensional output values. For each head H we i) Project input X to queries, keys, and values:

$$\begin{aligned}
Q_h &= X\, W_h^{Q}, \\
K_h &= X\, W_h^{K}, \\
V_h &= X\, W_h^{V}
\end{aligned}$$

ii) Next, compute scaled dot production

$$\text{Attention}(Q_h, K_h, V_h) = \text{softmax}\Big(\frac{Q_h\, K_h^{T}}{\sqrt{d_k}}\Big)\, V_h$$

iii) Now we concatenate the output from all heads

$$\text{MultiHead}(X) = \text{Concat}\big(\text{head}_1, \ldots, \text{head}_h\big)\, W^O$$

iv) Applying residual connection and layer normalization

$$X' = \text{LayerNorm}\big(X + \text{MultiHead}(X)\big)$$

v) Feed-forward network

$$\begin{aligned} \text{FFN}(x) &= \max\big(0,\, xW_1 + b_1\big)\, W_2 + b_2, \\ \text{Output} &= X' + \text{FFN}(X') \end{aligned}$$

This TAttention module captures temporal dependicies within each securities time series using multi-head self attention. This structure follows the structure of Vaswani et al. (2017).

4.1.4 Multi-head Self Attention

This module capture the cross-sectional relationship between the securities and/or features. It captures the market-wide interactions between securities and the sector dependencies.

(i) Transpose for cross-sectional attention:

$$X^T \; = \; \text{transpose}(X, 0, 1)$$

(ii) Project transposed input to queries, keys, and values:

$$Q_h \; = \; X^T W_h^Q, \quad K_h \; = \; X^T W_h^K, \quad V_h \; = \; X^T W_h^V$$

(iii) Dot-product attention with custom temperature:

$$\text{Attention}(Q_h, K_h, V_h) \;=\; \text{softmax}\Big(\frac{Q_h K_h^T}{\text{temperature}}\Big)\, V_h$$

(iv) Define the temperature scaling:

$$\text{temperature} \;=\; \sqrt{\frac{d_{\text{model}}}{\text{nhead}}}$$

(v) Concatenate the final heads and project out:

$$\text{MultiHead}(X) \;=\; \text{Concat}\big(\text{head}_1, \ldots, \text{head}_h\big)\; W^O$$

There is a primary difference between the Temporal attention and Cross-Spatial attention head. The axis of attention in the Temporal Attention head within each position t in time series attends to other time points in the same feature. In the spatial attention head each security attends to other security in the same time step. To achieve this, we took a transpose of the input so that 'time' becomes the dimension over which queries and keys are computed. We perform the following: $X^T = \text{transpose}(X, 0, 1)$.
Standard attention uses a temperature of $\sqrt{d_k}$. Spatial attention instead uses $\sqrt{\frac{d_m}{h}}$ since each head spans d_m/h features.

4.1.5 Position-Wise Feed Forward

At this point each position is processed independently by the same MLP, expanding representation beyond attention alone.

$$\text{FFN}(z) = W_2\left(W_1\, z + b_1\right) + b_2,$$
$$Z^{(\ell)} = \left(Z'' + \text{FFN}(Z'')\right).$$

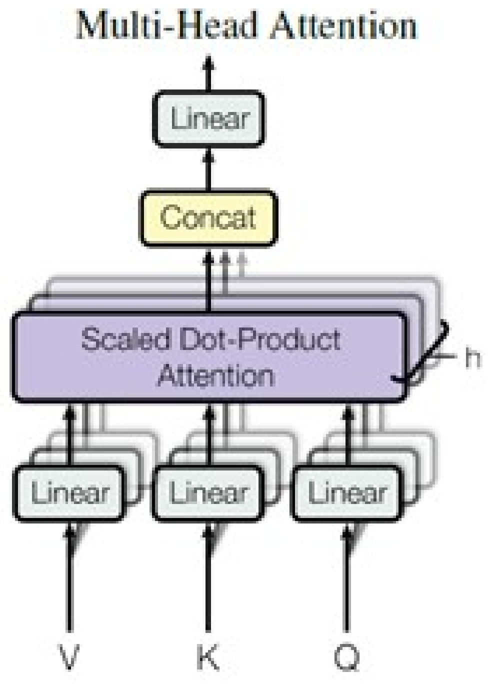


Figure 3: Attention Heads

4.1.6 Temporal Attention Aggregation

This module provides a learned weighted aggregation of the time series, compressing the sequence into a single vector.
i) Project the sequence to an intermediate representation

$$H = XW$$

ii) Use the final time step as a query

$$q = H_T$$

iii) Compute attention weights across time steps

$$\lambda = \mathrm{softmax}\big(H\,q\big)$$

iv) Produce weighted aggregation

$$\text{output} \;=\; \lambda^T X$$

Equivalently, per-instance notation for the attention weights and output is:

$$\alpha_{n,t} = \frac{\exp\bigl(H_{n,t,:} \cdot q_n\bigr)}{\sum_{t'} \exp\bigl(H_{n,t',:} \cdot q_n\bigr)},$$

$$y_n = \sum_{t=1}^{T} \alpha_{n,t} \, Z^{(L)}_{n,t,:} \, .$$

4.1.7 Transformer training

In our custom transformer, we convert raw market data into embeddings that use both, a reconstruction objective and a contrastive objective. Removing Recurrent Networks with self – attention brings two key advantages: parallel modeling over a window of T days and direct dependency modeling between two days. Because our Sequence Model processes sliding windows of length H (e.g. 50 trading days inputs) across all the 100 securities in parallel, the transformer's ability to compute all the attention weights in O(T2) time rather than step by step processing. Moreover, self-attention directly connects each time step to every other step within the window. In particular, for query Q, key K, and value V, we have

$$\text{Attention}(Q, K, V) = \text{softmax}\Bigl(\frac{QK^\top}{\sqrt{d_k}}\Bigr)\, V,$$

where d_k is the dimensionality of the keys. This mechanism lets the model learn intra-feature relationships—e.g. how volatility over the last 50 days influences current volatility trends—without having to propagate information through 50 sequential RNN states. This direct transmission of information mitigates the vanishing-gradient issues common in recurrent networks (RNNs/LSTMs), for example when modeling monthly earnings.

During training, the Transformer's output is

$$\text{output} = \text{Transformer}\big(\text{batch}\big) \;\in\; \mathbb{R}^{N\times T\times d_m},$$

where N is the batch size, T the time-window length, and d_m the embedding dimension. This output is obtained by training the Transformer with two objectives. First, we project the original inputs to form a reconstruction target:

$$\text{target} = (\text{src}) \;\in\; \mathbb{R}^{N\times T\times d_m}\,.$$

Then we minimize the mean-squared error:

$$\mathcal{L}_{\text{recon}} = \big\|\text{output} - \text{target}\big\|_2^2\,.$$

Second, we apply a temporal contrastive loss on each window. Define

$$z_{\text{anchor}} = \text{output}_{:,\,-2,:}\,, \qquad z_{\text{pos}} = \text{output}_{:,\,-1,:}\,.$$

Finally, the contrastive loss is

$$\mathcal{L}_{\text{ctr}} = -\log \frac{\exp\big(z_{\text{anchor}} \cdot z_{\text{pos}}/\tau\big)}{\displaystyle\sum_j \exp\big(z_{\text{anchor}} \cdot z_{\text{pos}}^{(j)}/\tau\big)}\,.$$

The contrastive component is the InfoNCE loss, which tunes the model's sensitivity to temporal changes by reinforcing representations that distinguish one window's trajectory from others. We combine the two objectives into a single loss:

$$\mathcal{L} = \mathcal{L}_{\text{recon}} \;+\; \lambda_{\text{ctr}}\,\mathcal{L}_{\text{ctr}}\,,$$

thereby retaining both spatial details (important for portfolio allocation) and temporal trends (critical for time-sensitive trading decisions). We optimize

this loss using the Adam optimizer, training both the Transformer and a small input-projection head, and then deploy the learned representations in our Robo-Advisor downstream tasks.
Our custom gated features in our transformer's input level ensure that depending on the market context, the model can emphasize which features are important. The momentum weighted smooth gates prevent any erratic shifts in feature weighting.It acts like an inertia term on the signal, so the robo-advisor doesn't over-react to erratic daily or short-term volatility, yielding stable, context-aware embeddings for downstream tasks.

$$\begin{aligned} W_n^{\text{curr}} &= \text{softmax}\Big(\frac{W_g\, g_n + b_g}{T_\beta}\Big), \\ W_n^{\text{dyn}} &\leftarrow m\, W_n^{\text{dyn}} + (1-m)\, W_n^{\text{curr}}, \\ W_n^{\text{out}} &= \alpha\, W_n^{\text{curr}} + (1-\alpha)\, W_n^{\text{dyn}}, \\ \widetilde{x}_{n,t,i} &= x_{n,t,i}\, W_{n,i}^{\text{out}} \,. \end{aligned}$$

Here,
W_n^{curr} is the "current" gate computed via softmax;
W_n^{dyn} is its exponential moving average (momentum);
W_n^{out} is the weighted combination of current and dynamic gates; and
$\widetilde{x}_{n,t,i}$ is the resulting weighted feature.

4.2 Reinforcement Learning

Reinforcement Learning is a novel approach to learning where we have agents (one or more) which interact with the environment and take learning actions based on those. It is a reward-based algorithm where the agents are incentivized to maximize the rewards, with the objective to find optimum policies which maximize the rewards over time. Reward structures can take many forms in this framework. Agents in reinforcement learning (RL) learn value functions to estimate expected returns from states $V^\pi(s)$, state–action pairs $Q^\pi(s,a)$, or policies directly via methods such as Q-learning, SARSA, policy gradients, and actor–critic algorithms. The goal is to learn a policy π that maximizes the discounted expected return:

$$\mathbb{E}\Bigg[\sum_{t=0}^{\infty} \gamma^t r_t\Bigg],$$

where r_t is the reward at time t and $\gamma \in [0,1)$ is the discount factor. Value-based methods approximate either $V^\pi(s)$ or $Q^\pi(s,a)$, while policy-gradient and actor–critic methods adjust policy parameters θ directly by performing gradient ascent on the expected return.

4.2.1 Applications in Financial Domain

In portfolio management, the state space is enriched by market features like daily returns, technical indicators, volume, spreads, volatilities, etc. The agents' actions can either be discrete or continuous and the sum of portfolio weights sums to 1. Depending on short-selling conditions, the weights can be between [0,1] or beyond that range. The reward is generally defined based on the task at hand. It could either be maximizing daily returns or minimizing risk like minimizing the maximum drawdown. Gu et al. (2025) proposed a Time-Aware and Short Selling (MTS) framework that augment deep RL with temporal encoding and explicit risk controls, achieving higher returns and lower volatility on large equity universes than traditional benchmarks. In Chen (2023), they propose a Task – Context Mutual Actor Critic (TC – MAC) architecture that trains value networks on market regimes, yielding a significant improvement in returns over prior deep-RL portfolio strategies.

4.2.2 TD3 Reinforcement Learning Framework

TD3 is part of a deterministic policy – gradient methods where an actor network

$$\mu\big(s \mid \theta^{\mu}\big)$$

comes up actions and a critic network

$$Q\big(s, a \mid \theta^{Q}\big)$$

evaluates them. Deterministic policies enable low variance gradient estimates in continuous action spaces. This is an improvement over stochastic policy methods. TD3 is an improvement on DDPG. DDPG, despite of great performances, has problems with hyperparameters and other kinds of tunings. Another problem with DDPG is that once it learns its Q-function it begins to dramatically overestimate its Q-values. This leads to the breaking of policies as it exploits the error term in Q-values. TD3 algorithm addresses these problems.

4.2.3 Clipped Double Q learning

TD3 learns two twin Q-functions. It then takes the lesser of the Q-values of the two to get the targets for the error loss functions for Bellman.

4.2.4 Delayed Policy Updates

The policy and the target networks are updated less frequently than the Q-functions. In Fujimoto et al.(2018), they recommend updating the policy once for every two updates of the Q-functions.

4.2.5 Target Policy Smoothening

TD3 combats the exploitation of Q-function errors by adding noise to the target action. This smooths the Q-function along with the changes in action.

These updates significantly improved upon the baseline DDPG policies.
Once we get the data preparation done from the input embeddings after running the 01st January, 2015 to 31st December, 2024 through the pre-trained transformer, we are ready to run our TD3 algorithm. We formulate our portfolio management framework as a continuous – action Markov Decision Process (MDP). In this, for each time step 't', the agent (Portfolio Manager) observes the state st, takes and action at (portfolio weights and other parameters) and receive a reward rt and get the observations for the next state (transitions from st to st+1. We solve this MDP using a Twin – Delayed DDPG (TD3) Actor – Critic algorithm.
Our Agent implements:
a) A Deterministic actor policy

$$\mu(s \,|\, \theta^{\mu})$$

b) Twin Critics

$$Q_i\big(s, a \mid \theta^{Q_i}\big)$$

to tackle over-estimation bias
c) Corresponding Target Networks

$$\mu' \quad \text{and} \quad Q_i'$$

for stable bootstrapping.
d) A replay buffer to store transitions and tackle temporal correlations, using different sampling techniques.
e) Learning rate schedulers for actor and critics (e.g. Cosine – annealing)

4.2.6 Markov Decision Process (MDP)

MDP is a stochastic dynamic decision control problem. It is a model for sequential decision making when we have uncertainty in outcomes. This

following is our MDP: the set of states (market + portfolio observations), actions (allocations + Lagrange multipliers +expected targets), the unknown market transition, the reward function (risk-adjusted PL), and the discount factor. ($\mathcal{S}$, $\mathcal{A}$, P, r, γ)
where: $\mathcal{S} \subset \mathbb{R}^d$ is the state space (observation vectors from PortfolioEnv). $\mathcal{A} \subset \mathbb{R}^m$ is the continuous action space, the set of allocations weights. It is a constrained space of actions where the weights are non-negative and sum up to 1.

$$P\big(s_{t+1} \mid s_t, a_t\big)$$

is the transition kernel giving the probability density of next state s after taking action a in state s. It is the probability distribution of next state given the current state and actions (allocations). This is induced by the market in our simulated Portfolio Environment.

$$r(s_t, a_t) = r_t^{\text{env}}$$

is the scalar reward function (risk-adjusted PL). Current time step's portfolio return minus the pacing penalties. $\gamma \in [0, 1)$ is the discount factor. This determines the amount we care for future awards vs current/immediate reward.

4.2.7 State Value function

$$V^{\pi}(s) = \mathbb{E}\Big[\sum_{t=0}^{\infty} \gamma^t \, r_t^{\text{env}} \;\Big|\; s_0 = s,\ a_t = \pi(s_t)\Big].$$

Here $V^{\pi}(s)$ is the expected risk-adjusted profit & loss (P/L) starting from portfolio state s when actions are chosen according to policy π.

4.2.8 Action-Value Function

The action–value function under policy π is

$$Q^{\pi}(s,a) = \mathbb{E}\Big[\sum_{t=0}^{\infty} \gamma^t\, r_t \;\Big|\; s_0 = s,\ a_0 = a,\ a_t = \pi(s_t)\Big].$$

Here, the action–value $Q^{\pi}(s,a)$ represents the expected return if, starting in state s, we take action a at time 0 and thereafter follow policy π.

4.2.9 Bellman Value Expectation Equation

In general, the state-value function under policy π satisfies the Bellman expectation equation:

$$V^{\pi}(s) = r\big(s, \pi(s)\big) + \gamma\, \mathbb{E}_{s' \sim P(\cdot|s,\pi(s))}\big[V^{\pi}(s')\big].$$

Here, - $r\big(s,\pi(s)\big)$ is the immediate environment reward from taking action $\pi(s)$ in state s (in our case this is r_t^{env} when we apply the current step's allocation). - The second term $\gamma\, \mathbb{E}_{s'}[V^{\pi}(s')]$ is the discounted expected value of the next portfolio state.

4.2.10 Bellman Expectation Equation for the Q-Value

The action-value function under policy π satisfies:

$$Q^{\pi}(s,a) = r(s,a) + \gamma\, \mathbb{E}_{s' \sim P(\cdot|s,a)}\big[\, Q^{\pi}\big(s', \pi(s')\big)\big].$$

Here, the Q-value $Q^{\pi}(s,a)$ is the immediate reward $r(s,a)$ plus the discounted expected Q-value of the next state when actions thereafter follow policy π.

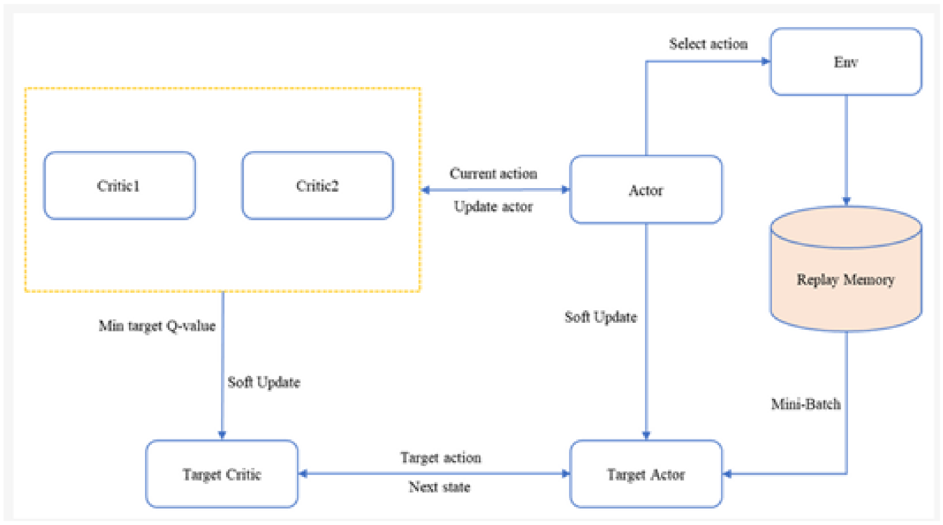


Figure 4: Flowchart of our TD3 framework

4.2.11 Deterministic Policy Gradient

The gradient of the expected return $J(\theta^\pi)$ under a deterministic policy $\pi(s;\theta^\pi)$ is given by the Deterministic Policy Gradient theorem:

$$\nabla_{\theta^\pi} J(\theta^\pi) = \mathbb{E}_{s\sim d^\pi}\Big[\nabla_a Q^\pi(s,a)\big|_{a=\pi(s;\theta^\pi)}\ \nabla_{\theta^\pi}\pi(s;\theta^\pi)\Big],$$

where - $d^\pi(s)$ is the on-policy state distribution under π, - $\nabla_a Q^\pi(s,a)|_{a=\pi(s)}$ measures how a small change in action affects the critic's value

In practice, we approximate this expectation by sampling states s from the replay buffer, computing the two gradient terms for each sample, and performing gradient ascent on θ^π to improve the policy's expected return.

4.2.12 Actor Network

The actor network is a neural network that takes the input state (s) and outputs a deterministic action

$$a = \mu_\theta(s).$$

. The actor maximizes the Q1 by using gradient ascent (or minimizing negative).

$$L_\mu = -\frac{1}{B}\sum_{i=1}^{B} Q_1\big(s_i, \mu(s_i)\big).$$

4.2.13 Critic 1 and Critic 2

We have two separate Q-networks that estimate the action value

$$Q_1(s,a) \quad \text{and} \quad Q_2(s,a)$$

. We take the minimum of Q1 and Q2. This mitigates that positive bias that a single critic introduces (like DDPG structure). The architectures of Q1 and Q2 are identical but they are initialized at different weights. Critics learn on this

$$L_{Q_i} = \frac{1}{B}\sum_{i=1}^{B}\big(Q_i(s_i,a_i) - y_i\big)^2.$$

The actor is updated less frequently than the critics. Specifically, for every n critic updates, we update the actor by minimizing the actor loss. Delaying the actor updates separates the policy improvement step from the rapidly fluctuating critic estimates, preventing destabilizing feedback loops.
After each update, we perform "soft" updates of the target networks with a small factor $\tau \ll 1$:

$$\theta' \;\leftarrow\; \tau\,\theta \;+\; (1-\tau)\,\theta',$$

where θ are the current network parameters and θ' are the target network parameters. This smooth interpolation further stabilizes training by slowly tracking the learned networks.

4.3 The Actor Network Structure

State Representation The input state vector $s \in \mathbb{R}^{S_{\text{dim}}}$ for each time step consists of a history window of length H containing, for each security n:

- Standardized log-returns over the window,
- Standardized log-daily trading volume,
- Rolling volatilities at 10-day, 30-day, and 90-day horizons,
- Historical portfolio values over H days,
- Previous action parameters: two weight vectors of length #securities, plus Λ (risky-capital fraction) and Θ (risk-degree).

Hence the total input dimension is

$$S_{\text{dim}} = H\big(\#\text{securities} \times (1_{\log r} + 1_{\log v} + 3_{\text{vol}} + 1_{\text{pv}} + 2_{\text{w}}) + d_{\text{emb}}\big) + 2 + 2$$

which we abbreviate as

$$S = H\left(\#\text{securities} + d_{\text{emb}}\right) + H\left(\text{portfolio value} + \text{action dims}\right) + \Lambda + \Theta.$$

Network Architecture

- **Hidden layers:** Two fully-connected layers with 400 and 300 units, each followed by ReLU (and optional LayerNorm).
- **Output heads (4):**
  - **(a)** *Conservative weight head:* $\mathbf{w}_{\text{cons}}$, portfolio weights targeting expected returns.
  - **(b)** *Risky weight head:* $\mathbf{w}_{\text{risk}}$, weights in "profit-buffer" mode once returns exceed targets.
  - **(c)** $\Lambda \in [0, 1]$: fraction of capital allocated to risky assets (learned).
  - **(d)** $\Theta \in \mathbb{R}^+$: risk-aversion coefficient for the risky allocation (learned).

The actor outputs unconstrained vectors in $\mathbb{R}^{2n+2}$. We slice into:

- raw base logits $\in \mathbb{R}^n \rightarrow$ softmax $\rightarrow w_{\text{base}}$.
- raw risky logits $\in \mathbb{R}^n \rightarrow$ softmax $\rightarrow w_{\text{risky}}$.
- raw $\Lambda \in \mathbb{R} \rightarrow$ sigmoid $\rightarrow \Lambda \in (0, 1)$.

- raw $\Theta \in \mathbb{R} \to$ sigmoid $\to \Theta \in (0,1)$.

Final portfolio weights:

$$w = (1-\Lambda)\, w_{\text{base}} \;+\; \Lambda\, w_{\text{risky}}.$$

We clamp the noisy actions between [min, max] to ensure valid allocations. At each time step, the actor obtains the current state st and outputs an action at. To encourage exploration, we add a little noise to the action training.

$$a_t = \mu(s_t) \;+\; \varepsilon_t, \quad \varepsilon_t \sim \mathcal{N}(0, \sigma^2 I).$$

The environment takes in the values of the actions and produces the values for the next time step of the state space.
Each of these transitions

$$(s_t, a_t, r_t, s_{t+1})$$

is stored in the memory buffer. We maintain a cyclic buffer D of size M.

4.4 Critic Network

While the actor network follows its update rule, the two critics proceed as follows:

(i) Sample mini-batches. The critic draws N transitions

$$\{(s_i, a_i, r_i, s_{i+1})\}_{i=1}^{N}.$$

(ii) Compute the noise-smoothed target action.

$$\begin{aligned} a'_{i+1} &= \mu_{\theta'}(s_{i+1}), \\ \tilde{a}_{i+1} &= \text{clip}\big(a'_{i+1} + \epsilon,\, -c,\, c\big), \quad \epsilon \sim \mathcal{N}(0, \sigma^2). \end{aligned}$$

(iii) Form the TD-target with twin critics.

$$y_i = r_i + \gamma \min_{j \in \{1,2\}} Q_{\phi'_j}\big(s_{i+1}, \tilde{a}_{i+1}\big).$$

(iv) Compute critic losses. For each online critic $j \in \{1, 2\}$:

$$L(\phi_j) = \frac{1}{N} \sum_{i=1}^{N} \big(Q_{\phi_j}(s_i, a_i) - y_i\big)^2.$$

(v) Gradient-descent updates. Update each critic by backpropagating $\nabla_{\phi_j} L(\phi_j)$ and taking an optimizer step.

(vi) Soft-update target critics. For $j = 1, 2$:

$$\phi'_j \;\leftarrow\; \tau\,\phi_j \;+\; (1-\tau)\,\phi'_j, \quad \tau \ll 1.$$

We then sample random observations for learning. This would help break the temporal correlations between the time series data.

4.5 Lagrange Multipliers for Constrained Optimization

When we are optimizing a function, f(x) with given constraints g(x) = 0, the lagrange multipliers introduces dual variables λ_i which forms the Lagrangian

$$L(x, \lambda) \;=\; f(x) \;-\; \sum_i \lambda_i\, g_i(x)\,.$$

At optimal points we have solution to the KKT stationarity condition,

$$\nabla_x L(x^*, \lambda^*) = 0$$

Along with primal feasibility $g_i(x^*) = 0$ and dual feasibility λ_i^*, unconstrained in the equality case, and complementary slackness when inequalities are present. Intuitively, each λ_i measures the sensitivity of the optimum to the uncertainty in the corresponding constraint. Our robo-advisor framework is a Constraint Markov Decision Process (CMDP). Our agent chooses portfolio allocations (actions) to maximize the overall expected returns, while ensuring that our returns are over our local targets (i.e. weekly, monthly, and annual). We wish to maximize the global objective

$$\pi_{\max} J_{\text{global}}(\pi)$$

$$\text{s.t.} \quad J_k(\pi) \;\geq\; \tau_k, \quad k \in \{\text{week},\ \text{month},\ \text{year}\}.$$

We introduced non-negative Lagrange multipliers $\lambda_k \geq 0$ as a single unconstrained 'Lagrangian' objective, which enforces each constraint.

$$L(\pi, \lambda) \;=\; J_{\text{global}}(\pi) \;-\; \sum_k \lambda_k \big[\tau_k - J_k(\pi)\big], \quad \lambda_k \geq 0.$$

The term $-\lambda_k[\tau_k - J_k(\pi)]$ is for penalizing any shortfall $J_k(\pi) < \tau_k$.

4.5.1 Saddle Points

We consider the Lagrangian $L(\theta, \lambda)$, which is concave in θ and convex in λ. The saddle-point conditions read

$$L(\theta^*, \lambda) \;\leq\; L(\theta^*, \lambda^*) \;\leq\; L(\theta, \lambda^*) \quad \forall\, \theta, \lambda \geq 0.$$

λ-descent: λ^* imposes the minimal necessary penalty to enforce $J_k(\pi) \geq \tau_k$.

θ-ascent: $L(\theta^*, \lambda^*) \geq L(\theta, \lambda^*)$ for all θ.

Solving the KKT conditions directly is intractable for neural policies. Instead, we update the multipliers by stochastic gradient:

$$\lambda_{k,t+1} = \big[\lambda_{k,t} + \beta_t\big(\tau_k\, J^k(\theta_t)\big)\big]_+, \quad \forall\, k,$$

where $[\cdot]_+ = \max\{0, \cdot\}$.
The actor–critic algorithm then treats

$$\sum_k \lambda_k \left[\tau_k - r_k\right]$$

as an additional instantaneous reward term in the policy update.

4.5.2 Dual Ascent Algorithm

Since solving KKT analytically is unfeasible for Neural Net policies, we use alternate methods.
1) Policy update (Primal): We update the policy via the policy-gradient on the Lagrangian reward.

$$r_t^{\text{Lagrangian}} \;=\; r_t^{\text{base}} \;-\; \sum_k \lambda_k \,\max\bigl\{0,\; \tau_k^{\text{cum}}(t) - r_{k,t}\bigr\}\,.$$

where $r_{k,t}$ is k-horizon (of time t)'s cumulative returns. 2) Multiplier update (Dual): We update the dual by stochastic gradient ascent on the Lagrangian

$$\lambda_k \;\leftarrow\; \lambda_k \;+\; \eta\bigl(\tau_k - J_k(\pi)\bigr)$$

where $J^k(\pi)$ is the sample estimate of k-horizon returns at each period.
The alternating process (between Policy Updates and Multiplier Updates) optimizes for the saddle point. Policy updates push the Lagrangian up in π and the Multiplier updates push the Lagrangian down in λ.

4.5.3 Karush – Kush – Tucker Conditions

As analytically solving KKT for Neural Net policies is not feasible, we check if we still enforce the KKT conditions.

1) Primal feasibility ($J_k \geq \tau_k$) By increasing the λ_k whenever the k-horizon returns fall below the τ_k, the dual ascent pushes the policy to satisfy $J_k \geq \tau_k$. Our learned policy empirically meets the pacing constraints, as training increases, the constraints are pushed to zero.
2) Dual feasibility ($lambda_k \geq$) We project each lagrange multiplier with a dual ascent multiplier and we never allow negative values, so λ_k being non-negative is maintained by construction.
3) Complementary slackness ($\lambda_k\,[\tau_k - J_k(\pi)] = 0$) When our policy comfortably exceeds the target, our updates set the λ_k at zero (no penalties). When our policy under performs, λ_k grows until we tighten the constraints. Thus, in steady state $\lambda_k = 0$ (slack) or $J_k = \tau_k$ (tight), establishing complementary slackness.
4) Stationarity ($\nabla_\theta L = 0$) Our actor-critic gradient step uses the Lagrangian form of the reward function, so our learning policy parameters satisfies the $\nabla_\theta\big[J_{\text{base}}(\theta) - \sum_k \lambda_k\, J_k(\theta)\big] \approx 0$ at convergence. This satisfies the stationarity conditions of KKT.

4.6 Reward Structure

Reward structure is the agent's main objective in RL. It maps the actions and states to a feedback loop that the agent seeks to maximize. While reward structure's goals are to maximize the expected rewards, the scale, shape, and the density are very important in reward structure. These factors determine whether the learning would be efficient or whether it converges with it all. Poorly designed rewards can lead to gamification (agents converging to sub-optimal strategies which lead to falsified amplification of rewards), or vanishing gradients. Real world scenarios are often filled with sparse signals for the fail/success structure. This makes credit assignment poor. Reward shaping done via adding auxiliary tasks increases the speed of learning. Bootstrapped reward shaping methods adaptively design shaping potentials from the values estimates to densify feedback without optimality biases.

4.6.1 Reward Integration with TD3

As mentioned earlier, TD3 is an off-policy actor-critic algorithm with continuous actions. It solves the function approximation bias of DDPG with twin-critics, delayed policy updates, and target-action smoothening. At each step the critics are updated towards the Bellman target.

$$y_t \; = \; r_t \; + \; \gamma \, \min_{j=1,2} Q_{\phi'_j}\big(s_{t+1}, \tilde{a}_{t+1}\big)$$

Where r_t is the current rewards, γ is the discount factor and $\tilde{a}_{t+1}$ is a noise induced target action. The actor works towards maximizes the critic's Q – estimates by performing $\nabla_\theta J \approx \mathbb{E}[\nabla_a Q_\phi(s,a)|_{a=\pi_\theta(s)} \nabla_\theta \pi_\theta(s)]$. Thus every aspect of the reward function directly influences the critic's value estimation and actor's ascent direction. TD3 reward function is sensitive to scale (too large will cause instability and too small will cause vanishing gradient).
We simulate daily trading over a portfolio of n assets. Our framework's base reward structure is:

(a) Asset Returns:

$$r_{t,i} = \frac{P_{t,i} - P_{t-1,i}}{P_{t-1,i}}, \quad P_{t,i} \text{ clipped to avoid division by zero.}$$

(b) Portfolio Returns:

$$R_t = \sum_{i=1}^{n} w_{t,i}\, r_{t,i}, \quad \sum_{i=1}^{n} w_{t,i} = 1, \quad w_{t,i} \in [0,1].$$

(c) Penalties: We penalize any under-performance via Lagrange mult:

$$\begin{aligned} p_t = {} & \eta_{\text{weekly}} \max\big(0,\, e_{\text{weekly}} - R_t\big) \\ & + \eta_{\text{monthly}} \max\big(0,\, e_{\text{monthly}} - R_t\big) \\ & + \eta_{\text{yearly}} \max\big(0,\, e_{\text{yearly}} - R_t\big). \end{aligned}$$

Our environment reward then becomes

$$r_{\mathrm{env}} = J_{\mathrm{base}}(\theta) \; - \sum_{k \in \{\mathrm{weekly}, \mathrm{monthly}, \mathrm{yearly}\}} \lambda_k \left[\tau_k - J_k(\theta)\right].$$

4.6.2 Sharpe Ratio

Sharpe Ratio is one of the most common metrics to evaluate the financial performance in portfolio management. Sharpe ratio is a risk measure that is defined as volatility of the excess returns.

$$S \; = \; \sigma_p \, \mathbb{E}\big[R_p - R_f\big]$$

where σ_p is the standard deviation of the excess returns. In Hachaïchi et al. (2024), they demonstrate that adding Sharpe ratio to the rewards structure leads to better risk adjusted returns.
We add in the Sharpe ratio to the rewards structure (as part of the Critic's learning network) and our final reward becomes. As per Li et al. (2024) we have

$$r_t^{\mathrm{env}} = R_t - \mathrm{pen}_t + \alpha \, S_t.$$

5 Algorithm for our framework

Algorithm 1 TD3 with Sharpe-Based Reward and Pacing Dual-Ascent

```
Ensure: learned actor policy $\mu_\theta$
 1: Initialize actor $\mu_\theta$ and critics $Q_{\phi_1}, Q_{\phi_2}$
 2: Initialize target networks: $\mu_{\theta'} \leftarrow \mu_\theta, \quad Q_{\phi'_j} \leftarrow Q_{\phi_j}$
 3: Initialize replay buffer $\mathcal{D}$ of capacity $N$
 4: for episode = 1 to $M$ do
 5:   $s_0 \leftarrow$ Env.reset()
 6:   for $t = 0$ to $T - 1$ do
 7:     Sample noise $\epsilon_t \sim \mathcal{N}(0, \sigma^2)$, clipped to $[-c, c]$
 8:     $a_t \leftarrow \mu_\theta(s_t) + \epsilon_t$
 9:     $(s_{t+1}, r_t^{\text{env}}) \leftarrow$ Env.step($a_t$)
10:     Compute Sharpe bonus inside Env
11:     $r_t \leftarrow r_t^{\text{env}} +$ Sharpe bonus
12:     Store $(s_t, a_t, r_t, s_{t+1})$ in $\mathcal{D}$
13:     Critic update:
14:     Sample minibatch $\{(s_i, a_i, r_i, s'_i)\}$ from $\mathcal{D}$
15:     $a'_i \leftarrow \mu_{\theta'}(s'_i) + \tilde{\epsilon}_i,$
16: $\tilde{\epsilon}_i \sim \text{clip}(\mathcal{N}(0, \sigma^2), -c, c)$
17:     $y_i \leftarrow r_i + \gamma \min_{j \in \{1,2\}} Q_{\phi'_j}(s'_i, a'_i)$
18:     for $j = 1, 2$ do
19:       $L_j \leftarrow \frac{1}{B} \sum_i (Q_{\phi_j}(s_i, a_i) - y_i)^2$
20:       $\phi_j \leftarrow \phi_j - \beta \nabla_{\phi_j} L_j$
21:       $\phi'_j \leftarrow \tau \phi_j + (1 - \tau) \phi'_j$
22:     end for
23:     if $t$ mod $d = 0$ then
24:       Actor update:
25:       $L_\mu \leftarrow -\frac{1}{B} \sum_i Q_{\phi_1}(s_i, \mu_\theta(s_i))$
26:       $\theta \leftarrow \theta - \alpha \nabla_\theta L_\mu$
27:       $\theta' \leftarrow \tau \theta + (1 - \tau) \theta'$
28:     end if
29:     Pacing dual-ascent:
30:     for each period $k \in \{\text{weekly}, \text{monthly}, \text{yearly}\}$ do
31:       $\lambda_k \leftarrow [\lambda_k + \eta(e_k - R_k)]_+$
32:       Env.update_multipliers($\{\lambda_k\}$)
33:     end for
34:     $s_t \leftarrow s_{t+1}$
35:   end for
36: end for
```

6 Experiment Setup and Results

As discussed above we have created a stock universe from top 100 stocks (top stocks by market cap from each of the sectors), to pre-train the transformer. We used the pre-trained transformer to get the embeddings for our experiment. When training the TD3 algorithm, we take two different portfolio and run the training (separately) on both. Portfolio 1: [SHW, ECL, T, VZ, XOM, CVX, JPM, BAC, LLY, UNH] Portfolio 2: [FCX, DD, DIS, TJX, COP, AXP, AMGN]

We will test the performance of the two portfolios with the base reward and Sharpe laden reward and evaluate the performances. We run each of the four scenarios for 50 episodes each and track the rewards for the validation phase. We have set our target structure to generate 3 % returns (annually). We ran our experiments on a RTX 4070 Ti Super GPU. Our training time for 1 run (50 episodes) would be around 90 minutes.

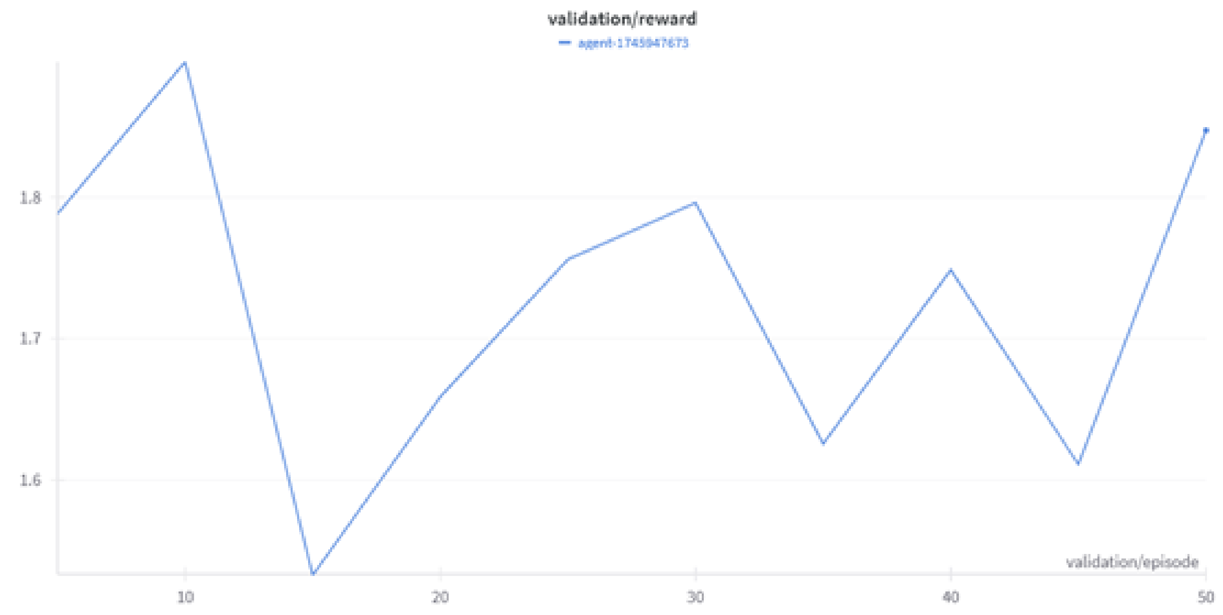


Figure 5: Portfolio 1: [SHW, ECL, T, VZ, XOM, CVX, JPM, BAC, LLY, UNH], Base Rewards

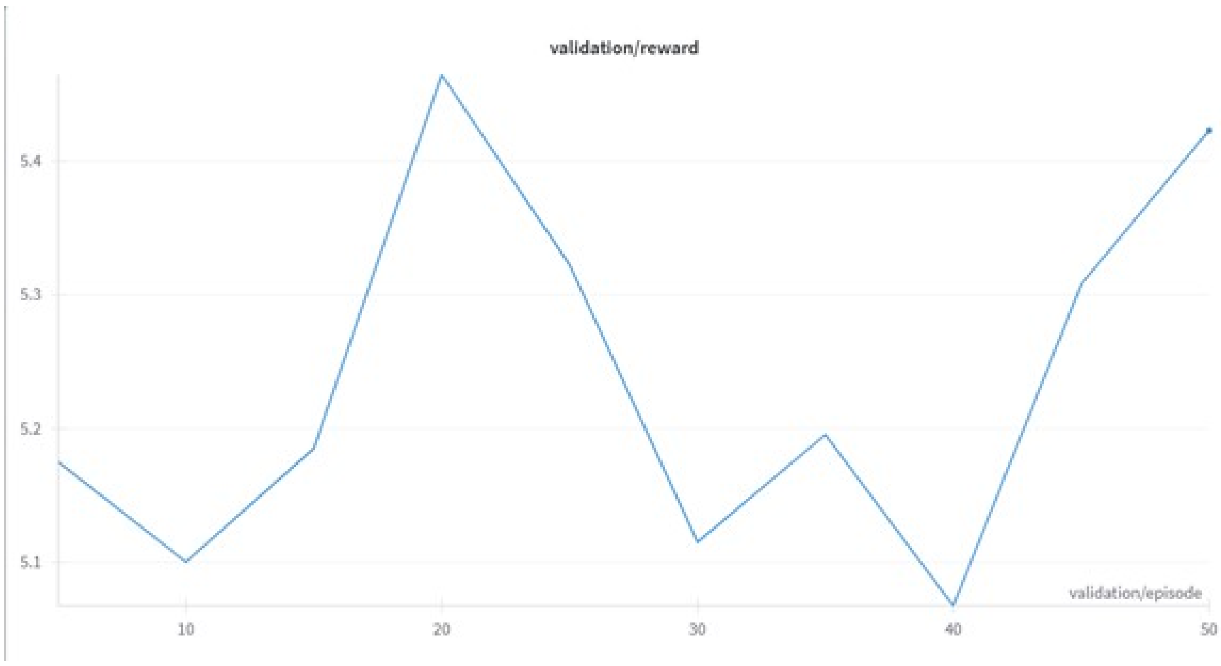


Figure 6: Portfolio 2: [FCX, DD, DIS, TJX, COP, AXP, AMGN], Base Rewards

The above is the reward structure for portfolio 1 and 2 with the reward structure without Sharpe ratio

$$r_{\text{env}} = J_{\text{base}}(\theta) - \sum_k \lambda_k \left[\tau_k - J_k(\theta)\right]$$

We find that if we don't have Sharpe laden in the reward structure our returns are less than the expected rewards. Our test rewards for portfolio 1 were around 0.8 and for portfolio 2 were 0.33. Possible reasons for the lower test scores would be overfitting to the validation data. As we ran the validation test on every 5 episodes, we fine tuned the hyperparameters to get the best possible combinations. This sometimes leads to overfitting the validation data and can give some what diminished results in the test dataset. Possible solutions are to increase the training data or introduce early stopping.

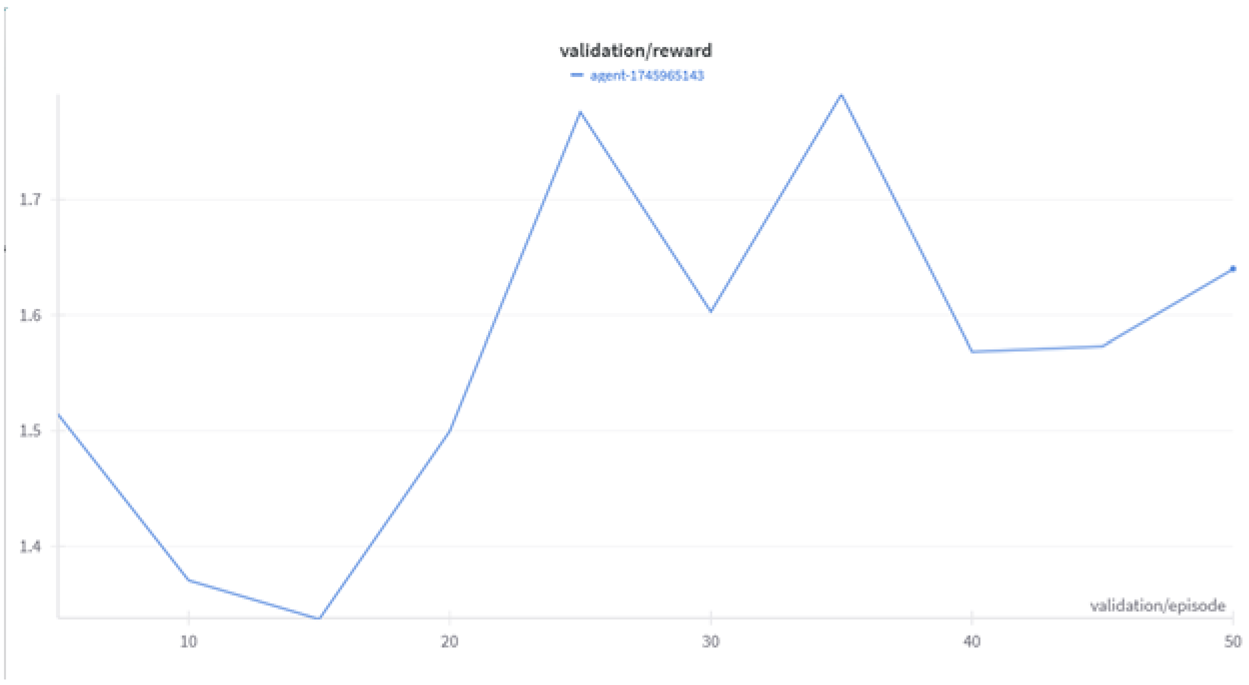


Figure 7: Portfolio 1: [SHW, ECL, T, VZ, XOM, CVX, JPM, BAC, LLY, UNH], Sharpe Reward

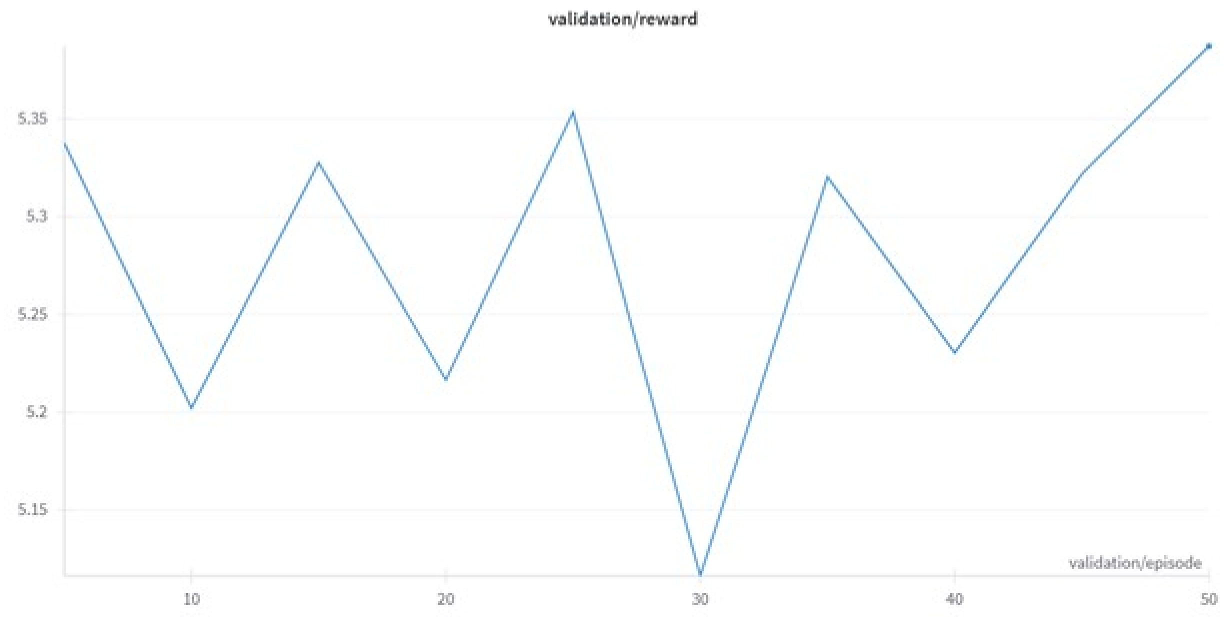


Figure 8: Portfolio 2: [FCX, DD, DIS, TJX, COP, AXP, AMGN], Sharpe Reward

The above are the results of the second reward function; with the Sharpe Ratio baked into the reward function. As we can see, the rewards have significantly improved with the addition of Sharpe Ratio. This is because the critic now understands that the Sharpe Ratio is part of the evaluation. The policy gradient now optimizes for the sharp ratio and optimizes the volatility of the asset returns. This leads to better weight allocation and better returns.

7 Conclusions

Financial portfolio management has evolved a lot over the decades. Classical machine learning models like SVM and Boosting algorithms have had good success in financial portfolio management. However, time series forecasting has their own problems, especially forecasting. Also, practitioners are looking for ways to take advantage of alternative data and get contextual information which can help us understand the market dynamics. Deep Reinforcement Learning has been increasingly used in financial portfolio management. Due to the nature of the RL algorithms, they are very well suited to capture the market dynamics. Algorithms like TD3 are well suited to form effective PM policies. We designed a two-stage framework where we extract market embeddings from a custom pre-trained transformer, and then we use those embeddings as a data augmentation to the raw market data. We have a Constrained MDP setup with the constraints solved using the Lagrangian multipliers. We have set up a TD3 policy for learning. We have trained different sets of portfolios and got better rewards when we included the risk-adjusted Sharpe ratio measure as part of the Actor-Critic's learning domain. Further work could include establishing the relationship between our risk amount and degree of risk to assess our Marginal Rate of Returns. Overall, we have a robust framework set up to leverage the market dynamics and establish a strong portfolio management strategy.

8 References

1. Chen, R., & Ren, J. (2022). Do AI-powered mutual funds perform better? *Finance Research Letters, 47*, 102616.

2. Mirete-Ferrer, P. M., Garcia-Garcia, A., Baixauli-Soler, J. S., & Prats, M. A. (2022). A Review on Machine Learning for Asset Management. *Risks, 10*(4), 84. `https://doi.org/10.3390/risks10040084`

3. Guidolin, M., Panzeri, G., & Pedio, M. (2024). Machine Learning in Portfolio Decisions. BAFFI CAREFIN Centre Research Paper No. 233.

4. Gang Hu & Ming Gu (2024). "Markowitz Meets Bellman: Knowledge-distilled Reinforcement Learning for Portfolio Management," arXiv:2405.05449.

5. Malandri, L., Xing, F., Orsenigo, C., Vercellis, C., & Cambria, E. (2018). Public Mood–Driven Asset Allocation: the Importance of Financial Sentiment in Portfolio Management. *Cognitive Computation.* `https://doi.org/10.1007/s12559-018-9609-2`

6. Bielecki, T. R., Cialenco, I., & Liu, H. (2023). Time consistency of dynamic risk measures and dynamic performance measures generated by distortion functions. arXiv:2309.02570.

7. Coache, A., & Jaimungal, S. (2021; rev. 2022). Reinforcement Learning with Dynamic Convex Risk Measures. arXiv:2112.13414.

8. Moresco, M., Mailhot, M., & Pesenti, S. M. (2023; rev. 2024). Uncertainty Propagation and Dynamic Robust Risk Measures. arXiv:2308.12856.

9. Righi, M. (2024). A note on robust convex risk measures. arXiv:2406.12999.

10. Dosovitskiy, A., Beyer, L., Kolesnikov, A., et al. (2021). An image is worth 16×16 words: Transformers for image recognition at scale. arXiv:2010.11929.

11. Chen, L., Lu, K., Rajeswaran, A., et al. (2021). Decision Transformer: Reinforcement Learning via Sequence Modeling. arXiv:2106.01345.

12. Li, T., Liu, Z., Shen, Y., Wang, X., Chen, H., & Huang, S. (2023). MASTER: Market-guided Stock Transformer for Stock Price Forecasting. arXiv:2312.15235.

13. Ding, X., Zhang, Y., Liu, T., & Duan, J. (2020). Deep learning for event-driven stock prediction. In *IJCAI* (pp. 2327–2333).

14. Zhang, L., Aggarwal, C., & Qi, G. J. (2021). Adaptive feature selection for financial time series forecasting. In *AAAI* (pp. 10961–10969).

15. Fujimoto, S., van Hoof, H., & Meger, D. (2018). Addressing function approximation error in actor-critic methods. arXiv:1802.09477.

16. Gu, F., Jiang, Z., García-Fernández, Á. F., et al. (2025). MTS: A deep reinforcement learning portfolio management framework with time-awareness and short-selling. arXiv:2503.04143.

17. Yang, S. (2023). Deep Reinforcement Learning for Portfolio Management. ScienceDirect.

18. Jiang, C., & Wang, J. (2022). A portfolio model with risk control policy based on Deep Reinforcement Learning. *MDPI*.

19. Hachaïchi, Y., & Lanwer, A. (2024). Benchmarking Reinforcement Learning (RL) Algorithms for Portfolio Optimization.

20. Li, H., & Hai, M. (2024). Deep Reinforcement Learning Model for Stock Portfolio Management Based on Data Fusion. *Neural Processing Letters*, 37, 123–145.