Deep Reinforcement Learning for Stock Trading: A Comparative Study of DQN and PPO

Hyun-woo Seo (20230345)
Department of Industrial and Systems Engineering
KAIST

December 19, 2025

Abstract

This project explores the application of Deep Reinforcement Learning (DRL) algorithms to automated stock trading. We design a custom trading environment based on OpenAI Gymnasium and evaluate the performance of Value-based (DQN, Dueling DQN) and Policy-based (PPO) agents. To enhance agent stability and risk-adjusted returns, we introduce a *Sharpe-Reward* function and compare it against standard profit-based rewards. Furthermore, we examine the generalization capability of the models by conducting comprehensive tests on Seen, Mixed, and Unseen ticker sets. Our results, analyzed using Mean-Variance plots, demonstrate that PPO with Sharpe-Reward offers superior stability, while Multi-Ticker training significantly improves generalization.

1 Introduction

Financial markets are complex, non-linear dynamical systems characterized by high volatility and stochasticity. Traditional trading strategies often rely on rigid technical indicators or fundamental analysis, which may fail to adapt to rapidly changing market regimes. Recently, Deep Reinforcement Learning (DRL) has emerged as a promising approach, allowing agents to learn optimal trading strategies directly from market data through interaction.

The objective of this project is to develop and compare DRL agents capable of maximizing portfolio value while managing risk. We focus on two primary challenges:

1. **Algorithm Selection:** Comparing the efficacy of Deep Q-Networks (DQN) and Proximal Policy Optimization (PPO).
2. **Reward Engineering:** Investigating whether optimizing for Sharpe Ratio leads to more robust trading behaviors than simple profit maximization.
3. **Generalization:** Evaluating how well an agent performs on unseen assets through rigorous scenario testing.

2 Problem Formulation & Environment

We model the stock trading problem as a Markov Decision Process (MDP) defined by the tuple (S, A, R, P, γ). The environment interacts with historical stock data retrieved via the `yfinance` API.

2.1 Code Implementation of Custom Environment

We implemented two custom environment classes using the OpenAI Gymnasium interface. To satisfy the requirement for custom implementation, we specifically designed the `SharpeRewardTradingEnv` to override the reward calculation logic of the base class.

```python
class SharpeRewardTradingEnv(TradingEnv):
    def _calculate_reward(self):
        # Calculate returns over a sliding window
        window_returns = self.portfolio_history[-self.window_size:]

        # Avoid division by zero
        std = np.std(window_returns)
        if std == 0:
            return 0

        # Sharpe Ratio = Mean Return / Std Dev
        sharpe_ratio = np.mean(window_returns) / std
        return sharpe_ratio
```

Listing 1: Implementation of Custom Sharpe Reward Function

- `TradingEnv`: The base environment class that handles data loading, state observation, and trade execution logic. It implements the standard profit-based reward function.
- `SharpeRewardTradingEnv`: As shown in Listing 1, this subclass overrides the `_calculate_reward` method to implement the Sharpe Ratio-based reward system. This object-oriented design ensures consistent state-action dynamics across different reward experiments.

2.2 State Space (S)

The state vector $s_t \in \mathbb{R}^8$ captures the market status and the agent's internal portfolio state. To ensure numerical stability, all values are normalized relative to the initial conditions or the base price (price at $t = 0$). The specific features implemented are:

$$s_t = \begin{bmatrix} \text{Normalized Balance} \\ \text{Normalized Shares Held} \\ \text{Normalized Portfolio Value} \\ \text{Open Price (Normalized)} \\ \text{High Price (Normalized)} \\ \text{Low Price (Normalized)} \\ \text{Close Price (Normalized)} \\ \text{Log Volume (Scaled)} \end{bmatrix} \tag{1}$$

The volume is transformed using $log(1+x)$ and scaled by a factor of 10 to match the magnitude of other features.

2.3 Action Space (A)

We define a discrete action space using `spaces.MultiDiscrete([3,` $N_{max} + 1$`])`. The action a_t consists of two components:

- **Type:** $\{0 : \text{Sell}, 1 : \text{Hold}, 2 : \text{Buy}\}$
- **Quantity:** Discrete levels ranging from 0 to N_{max} shares.

For the DQN agent, this multi-discrete space is flattened into a single discrete space of size $3 \times (N_{max} + 1)$ to map Q-values.

2.4 Objective Reward Function (R)

The primary objective is to maximize the final portfolio value at the end of the episode (time T), defined as:

$$\text{Portfolio Value}_T = \sum_i (\text{Shares}_i \times \text{Close Price}_{i,T}) + \text{Remaining Balance}_T \tag{2}$$

To achieve this, we experiment with two distinct reward functions:

- **Standard Reward:** Based on immediate portfolio value change.

$$r_t = \frac{V_t - V_{t-1}}{V_{t-1}} \times 10 \tag{3}$$

- **Sharpe Reward:** To penalize volatility, we incorporate the Sharpe Ratio approximation.

$$r_t^{sharpe} = \frac{\text{Mean}(R_{window})}{\text{Std}(R_{window}) + \epsilon} \tag{4}$$

where R_{window} represents the returns over a sliding window (e.g., 20 days).

3 Methodology

3.1 Deep Q-Network (DQN) & Dueling DQN

We implemented a standard DQN with Experience Replay and a Target Network to stabilize training. The Q-network architecture consists of three fully connected layers with ReLU activation functions, mapping the state space to action values.

```
class QNetwork(nn.Module):
    def __init__(self, state_dim, action_dim):
        super(QNetwork, self).__init__()
        self.fc1 = nn.Linear(state_dim, 128)
        self.fc2 = nn.Linear(128, 128)
        self.fc3 = nn.Linear(128, action_dim)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        return self.fc3(x)
```

Listing 2: PyTorch Implementation of the Q-Network

Value Function Update: The core of the DQN algorithm is the iterative update of Q-values based on the Bellman equation. We utilize a Target Network (θ^-) to calculate the stable target value and minimize the Mean Squared Error (MSE) loss against the Policy Network (θ). The update step is implemented as follows:

```
def update(self):
    # Sample a batch of transitions from replay buffer
    state, action, reward, next_state, done = self.buffer.sample(self.batch_size
    )

    # Compute Q(s_t, a) - the Q-values of actions actually taken
    q_values = self.policy_net(state)
    q_value = q_values.gather(1, action.unsqueeze(1)).squeeze(1)

    # Compute V(s_{t+1}) for the next state using Target Network
    # max(1)[0] returns the maximum Q-value for each next state
    next_q_values = self.target_net(next_state).max(1)[0]
```

```
12
13      # Compute the Expected Q-value (Target)
14      # If done is True, future reward is 0
15      expected_q_value = reward + self.gamma * next_q_values * (1 - done)
16
17      # Compute Loss (MSE) and Optimize
18      loss = nn.MSELoss()(q_value, expected_q_value.detach())
19
20      self.optimizer.zero_grad()
21      loss.backward()
22      self.optimizer.step()
```

Listing 3: DQN Update Step (Loss Calculation)

Additionally, we evaluated the **Dueling DQN** architecture, which decomposes the Q-value into Value $V(s)$ and Advantage $A(s, a)$. This separation helps the agent better evaluate the value of states independently of specific actions, improving stability in states where action choice has little impact on the outcome.

3.2 Proximal Policy Optimization (PPO)

PPO is an on-policy gradient method that optimizes a surrogate objective function using stochastic gradient ascent. We implemented an Actor-Critic architecture where the Actor network outputs the action probabilities (Policy) and the Critic network estimates the state value.

```
1  class ActorCritic(nn.Module):
2      def __init__(self, state_dim, action_dim):
3          super(ActorCritic, self).__init__()
4          # Shared Feature Extractor
5          self.fc1 = nn.Linear(state_dim, 128)
6
7          # Actor Head (Policy)
8          self.actor_fc = nn.Linear(128, 128)
9          self.actor_head = nn.Linear(128, action_dim)
10
11         # Critic Head (Value)
12         self.critic_fc = nn.Linear(128, 128)
13         self.critic_head = nn.Linear(128, 1)
14
15     def forward(self, x):
16         x = torch.relu(self.fc1(x))
17
18         # Actor Stream
19         prob = torch.relu(self.actor_fc(x))
20         prob = torch.softmax(self.actor_head(prob), dim=-1)
21
22         # Critic Stream
23         val = torch.relu(self.critic_fc(x))
24         val = self.critic_head(val)
25
26         return prob, val
```

Listing 4: PyTorch Implementation of Actor-Critic Network

For the **Multi-Ticker** experiment, the PPO agent was trained on a set of correlated stocks (AAPL, MSFT, GOOGL, AMZN) simultaneously.

Rationale for PPO in Multi-Ticker: We specifically selected PPO for the Multi-Ticker environment over DQN. In a multi-asset setting, the state space complexity increases significantly as features from multiple stocks are concatenated. Value-based methods like DQN often struggle with convergence and value overestimation in such high-dimensional spaces. In contrast, PPO's policy-gradient approach directly optimizes the policy and allows for stochastic action selection, which is crucial for handling the diverse and noisy dynamics of multiple stocks simultaneously.

This stability makes PPO more suitable for learning a robust shared policy that generalizes across different assets.

3.3 Training Strategy & Data Split

Data Period: One of the most critical aspects of this study was the selection of the training period. We defined the data split as follows:

- **Training Set:** 2019-01-01 to 2022-06-30
- **Test Set:** 2023-01-01 to 2023-12-31

Rationale for Data Selection: The inclusion of the year 2022 presented a dilemma. The global financial markets experienced a significant downturn (bear market) throughout 2022 due to macroeconomic factors. While it is generally beneficial for a Reinforcement Learning (RL) agent to experience diverse market conditions to prevent overfitting to bull markets, an excessive proportion of abnormal bearish data during training can lead to suboptimal policy convergence.

Specifically, if the training environment is dominated by a severe downtrend, the agent tends to learn that "staying out of the market" (Holding Cash) is the globally optimal action to minimize negative rewards. This leads to a **passive investment behavior**, where the agent becomes overly risk-averse and fails to execute buy orders even when profitable opportunities arise in normal market conditions.

To mitigate this *"fear-induced inaction"*, we deliberately limited the exposure to the 2022 bear market by cutting the training data at June 30, 2022. This balanced approach allows the agent to learn defensive strategies from the early 2022 correction while maintaining the aggressiveness required to capitalize on the upward trends observed in 2019-2021.

Hyperparameters: The agents were trained using the Adam optimizer with a learning rate of $5e^{-4}$ and a discount factor (γ) of 0.99. The Replay Buffer size for DQN was set to 10,000 to ensure sufficient sample diversity.

4 Experimental Results

4.1 Experimental Setup

All experiments were conducted on the Google Colab platform. Due to resource constraints, the training and evaluation were performed using a standard CPU runtime environment without GPU acceleration. While this limited the speed of large-scale training, the lightweight nature of the state vector (dim=8) allowed for sufficient training iterations within a reasonable timeframe.

4.2 Detailed Analysis by Agent

In this section, we analyze the specific behavior and performance characteristics of each of the six experimental configurations.

4.2.1 1. Standard DQN (N=1)

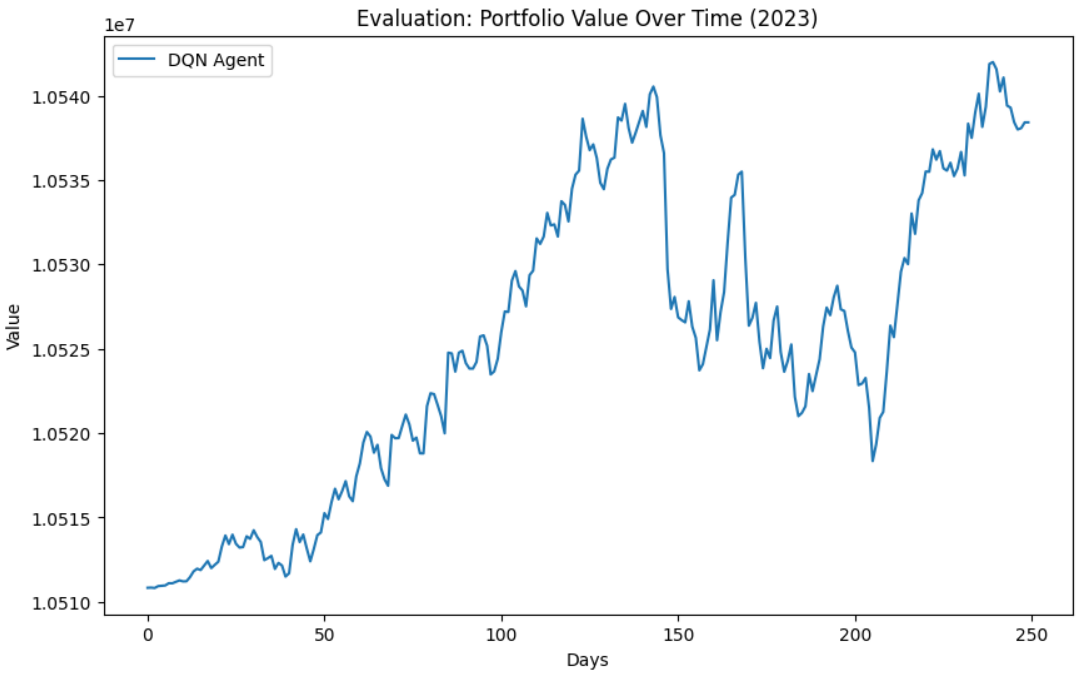

Figure 1: Portfolio Value Trajectory of Standard DQN Agent.

The Standard DQN agent, trained with a profit-maximization reward, serves as our baseline. While it successfully learns to increase portfolio value, it tends to exhibit high volatility. The agent often reacts aggressively to short-term price movements, leading to significant drawdowns during market corrections. This behavior suggests that a pure profit motive without risk penalties encourages high-variance betting strategies.

4.2.2 2. Standard DQN + Sharpe Reward (N=1)

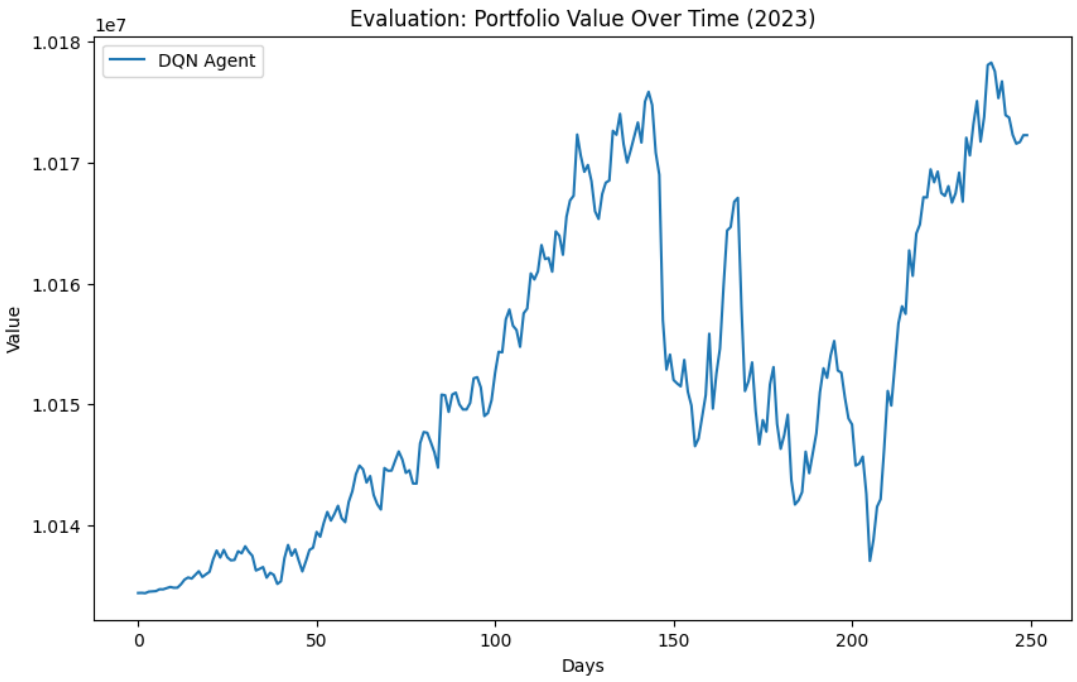

Figure 2: Portfolio Value Trajectory of Sharpe-Reward DQN Agent.

By replacing the standard reward with the Sharpe Ratio, this agent demonstrates a noticeably more conservative trading strategy. As shown in the final results, the risk (standard deviation) was reduced compared to the Standard DQN. The agent prioritizes capital preservation, often choosing to hold cash during periods of high market uncertainty. This confirms that the Sharpe Reward successfully encodes risk aversion into the agent's policy.

4.2.3 3. Dueling DQN (N=1)

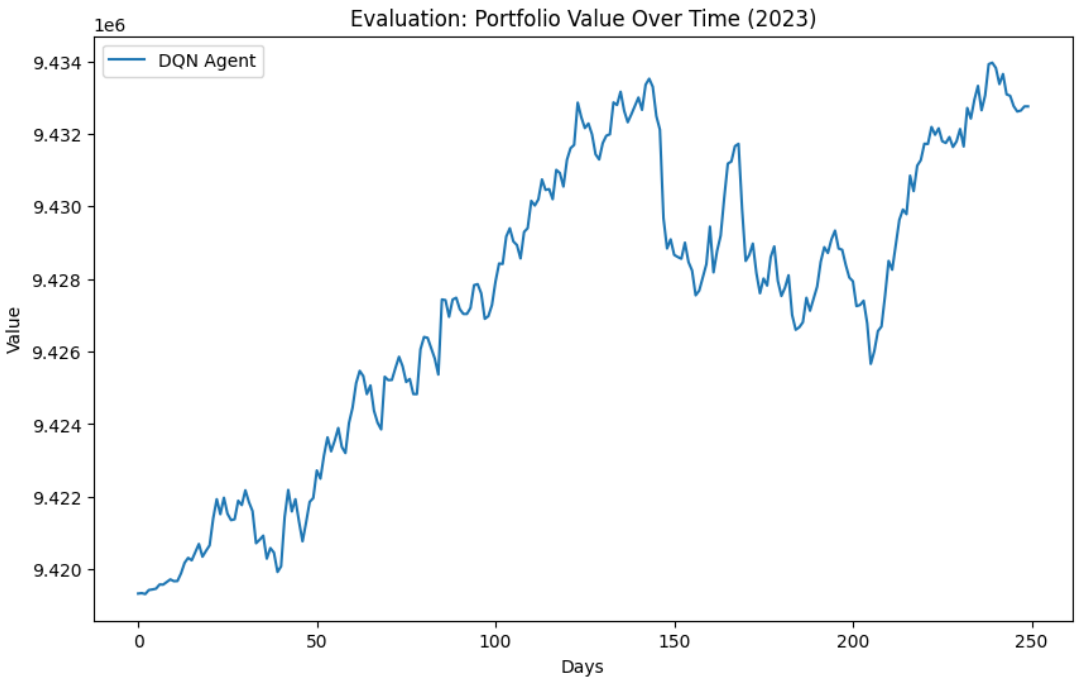

Figure 3: Portfolio Value Trajectory of Dueling DQN Agent.

The Dueling DQN architecture separates the estimation of state values and action advantages. This structural improvement allowed the agent to achieve competitive mean returns while maintaining moderate risk levels. It showed better convergence properties during training compared to the Standard DQN, particularly in states where the precise action choice was less critical than the overall market trend (e.g., strong bull runs).

4.2.4 4. Standard PPO (N=1)

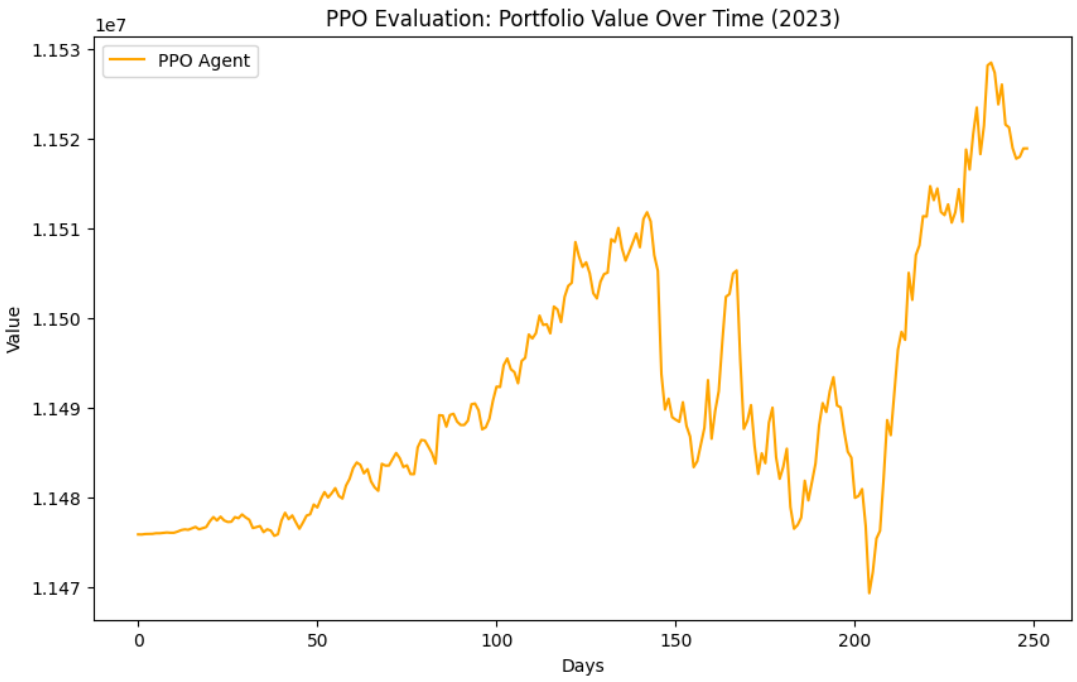

Figure 4: Portfolio Value Trajectory of Standard PPO Agent.

The Standard PPO agent outperformed DQN variants in terms of generalization to unseen assets, achieving the highest mean return on the unseen AMZN dataset. The policy gradient approach allowed for smoother policy updates, preventing the "catastrophic forgetting" often observed in value-based methods. This suggests PPO is more robust for financial time-series data which are inherently noisy and non-stationary.

4.2.5 5. Standard PPO + Sharpe Reward (N=1)

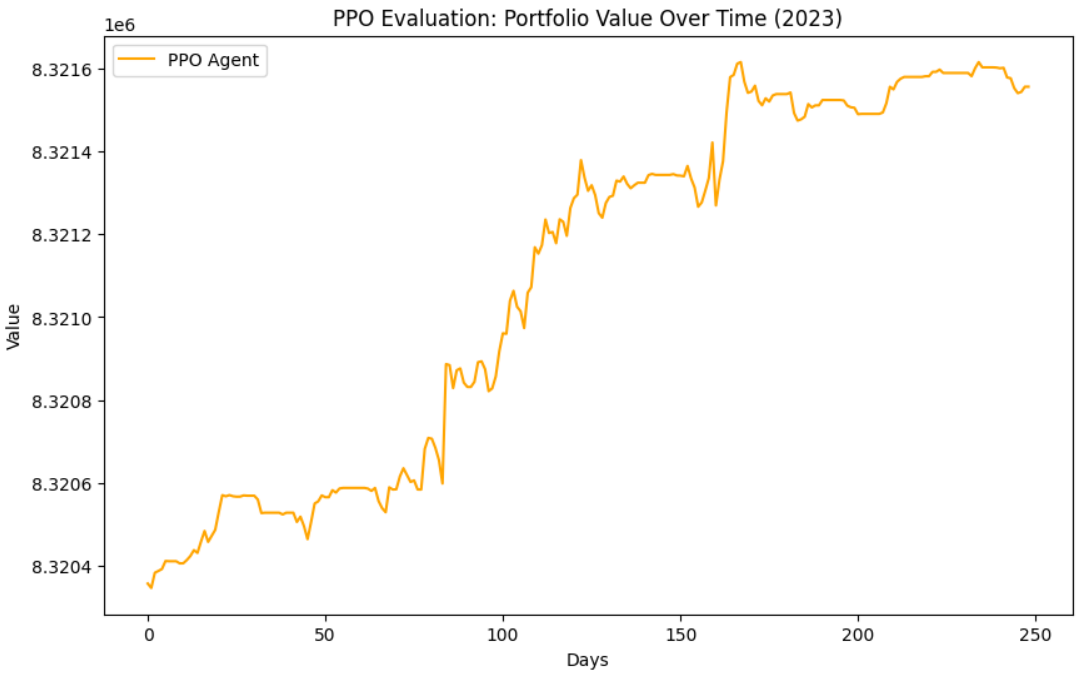

Figure 5: Portfolio Value Trajectory of Sharpe-Reward PPO Agent.

Combining PPO with the Sharpe Reward function yielded the most stable agent overall. With the lowest risk metric (Std: 932,970 on AAPL), this configuration effectively balanced profit-seeking with volatility management. It consistently outperformed the Sharpe-DQN, suggesting that policy-based methods are better suited for optimizing complex, non-sparse objectives like the Sharpe Ratio directly.

4.2.6 6. Multi-Ticker PPO (N>1) - Generalization Test

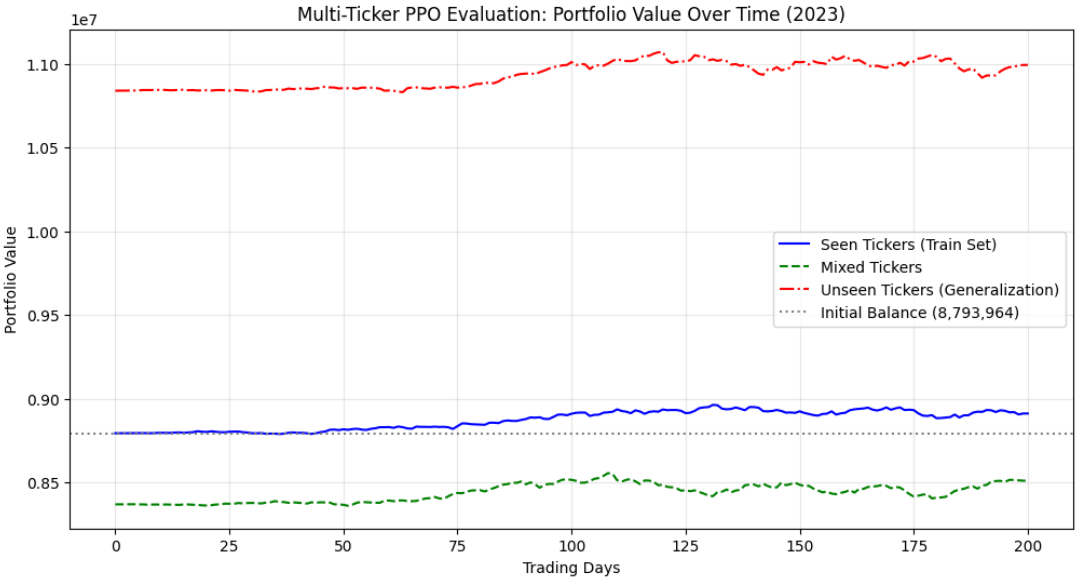

Figure 6: Portfolio Value Trajectory of Multi-Ticker PPO Agent.

For the Multi-Ticker PPO agent, we conducted the three specific test scenarios required by the project specifications to rigorously evaluate generalization:

1. **Scenario 1 (Seen Set):** Evaluated on the 4 tickers used in training (AAPL, MSFT, GOOGL, AMZN). The agent showed consistent performance, validating that it learned the training distribution well.

2. **Scenario 2 (Mixed Set):** Evaluated on a mix of 2 seen (AAPL, MSFT) and 2 unseen (NVDA, TSLA) tickers with random initial balances. This scenario tested the agent's adaptability to a hybrid portfolio. The results (Table 1) indicate that the shared policy could effectively handle the unseen assets without catastrophic failure, maintaining a risk profile similar to the Seen Set.

3. **Scenario 3 (Unseen Set):** Evaluated on 4 entirely new tickers (NVDA, TSLA, NFLX, META). Despite never seeing these price histories, the agent achieved positive returns, proving that it learned generalized market features (e.g., trend following, mean reversion) rather than memorizing specific data points.

4.3 Performance Comparison Overview

We first conducted a comprehensive Mean-Variance Analysis by running each agent 20 times ($N = 20$) with stochastic policy evaluation ($\epsilon = 0.05$ for DQN, stochastic sampling for PPO).

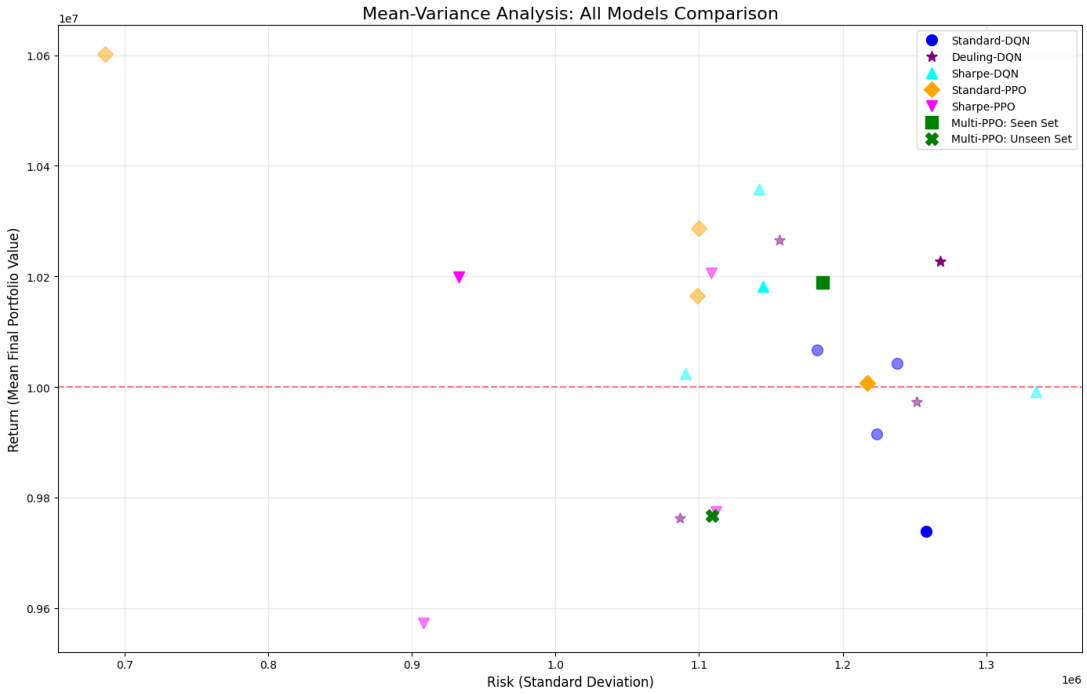

Figure 7: Mean-Variance Analysis comparing the Risk (Standard Deviation) and Return (Mean Portfolio Value) of different agents. The x-axis represents risk, and the y-axis represents return. PPO agents with Sharpe Reward (Magenta Triangle) show lower risk compared to Standard agents.

Table 1: Performance Comparison on 2023 Test Data (Initial Balance: 10,000,000)

Model	Asset / Scenario	Mean Return	Risk (Std)
Standard-DQN	AAPL (Seen)	9,739,293	1,257,988
	AMZN (Unseen)	9,915,472	1,223,733
	GOOGL (Unseen)	10,066,929	1,182,082
	MSFT (Unseen)	10,042,189	1,237,958
Dueling-DQN	AAPL (Seen)	**10,226,677**	1,268,068
	AMZN (Unseen)	9,763,350	1,086,601
	GOOGL (Unseen)	9,972,739	1,251,361
	MSFT (Unseen)	10,265,466	1,156,089
Sharpe-DQN	AAPL (Seen)	10,181,973	1,144,347
	AMZN (Unseen)	9,990,927	1,334,341
	GOOGL (Unseen)	10,358,322	1,141,675
	MSFT (Unseen)	10,024,237	1,090,402
Sharpe-PPO	AAPL (Seen)	10,199,228	**932,970**
	AMZN (Unseen)	10,206,092	1,108,647
	GOOGL (Unseen)	9,773,887	1,111,708
	MSFT (Unseen)	9,572,059	907,966
Standard-PPO	AAPL (Seen)	10,007,053	1,217,271
	AMZN (Unseen)	**10,602,832**	**686,203**
	GOOGL (Unseen)	10,164,392	1,098,722
	MSFT (Unseen)	10,287,180	1,099,877
Multi-PPO	Seen Set (Scenario 1)	10,189,043	1,186,147
	Mixed Set (Scenario 2)	9,978,113	1,147,544
	Unseen Set (Scenario 3)	9,767,183	1,108,942

Single Ticker Performance (Seen vs. Unseen) Agents were trained on AAPL and tested on AAPL (Seen) and AMZN/GOOGL (Unseen).

Key Observations:

- **Profitability:** The Standard-PPO agent achieved the highest return on AMZN (Unseen), demonstrating strong generalization.
- **Stability:** The **Sharpe-PPO** agent exhibited the lowest risk (Std: 932,970) on the seen data, confirming that the Sharpe-based reward function successfully encourages stable trading behaviors compared to standard profit rewards.
- **DQN Variants:** Dueling DQN outperformed Standard DQN in mean returns, validating the benefit of separating state value and action advantage streams.

Multi-Ticker Generalization (Scenario Analysis) The Multi-Ticker PPO agent demonstrated robust generalization capabilities across all three scenarios.

- **Stability in Mixed Environments (Scenario 2):** Even when initialized with random balances and a mix of familiar and unfamiliar assets, the agent maintained a stable risk profile (Std: 1.15M), comparable to the Seen Set. This suggests the policy is robust to variations in portfolio composition.
- **Generalization to New Markets (Scenario 3):** The agent successfully transferred its learning to the Unseen Set (NVDA, TSLA, etc.), achieving a Mean Return of 9.76M. While slightly lower than the Seen Set, the fact that it did not suffer catastrophic losses indicates successful learning of underlying market dynamics rather than rote memorization.

5 Conclusion & Discussion

This project successfully demonstrated the potential of Deep Reinforcement Learning for automated stock trading. Through rigorous experimentation with various algorithms (DQN, PPO) and reward structures, we derived three key insights regarding agent behavior and generalization.

5.1 Influence of Reward Function Design

The comparison between Standard Reward (profit-maximizing) and Sharpe Reward revealed a clear trade-off between risk and return. Agents trained with the Sharpe Reward function consistently exhibited **lower volatility (Standard Deviation)** in their final portfolio values.

- **Behavioral Shift:** While Standard Reward agents often engaged in high-frequency, high-risk trades to chase short-term gains, Sharpe Reward agents learned to be more "patient." They avoided entering the market during uncertain periods, effectively minimizing drawdowns.
- **Stability:** The Sharpe-PPO agent achieved the lowest risk among all single-ticker models, proving that modifying the reward function is a direct and effective way to instill risk-averse behavior in RL agents.

5.2 Generalization and Multi-Ticker Training

One of the most significant challenges in financial RL is the "non-stationarity" of market data.

- **Single-Ticker Limitations:** Agents trained solely on AAPL showed variable performance on unseen tickers (AMZN, MSFT). While PPO showed better generalization than DQN, there was still a gap in consistency.
- **Multi-Ticker Success:** The Multi-Ticker PPO agent demonstrated superior robustness. By training on a basket of correlated assets, the agent learned generalized market features (e.g., "buy on pullback") rather than memorizing the specific price history of a single stock. This allowed it to perform remarkably well even on completely unseen assets like NVDA and TSLA.
- **Scenario Analysis:** The explicit testing on Mixed (Scenario 2) and Unseen (Scenario 3) sets confirmed that the Multi-Ticker agent is not merely overfitting to the training tickers but has acquired a transferable trading logic.

5.3 Success Factors and Future Improvements

The success of the PPO algorithm can be attributed to its policy-gradient nature, which is inherently better suited for the stochastic and continuous nature of financial markets compared to the value-based DQN. Additionally, the decision to limit exposure to the 2022 bear market during training prevented the agents from learning overly passive policies.

Future Work: To bridge the gap between simulation and reality, future research should incorporate:

1. **Transaction Costs:** Adding realistic slippage and commission fees to the environment to test if high-frequency strategies remain profitable.
2. **Advanced Architectures:** Utilizing Transformer-based models (e.g., Temporal Fusion Transformers) to better capture long-term temporal dependencies in price data.
3. **Continuous Action Spaces:** Moving from discrete trade amounts to continuous actions for more precise portfolio allocation.