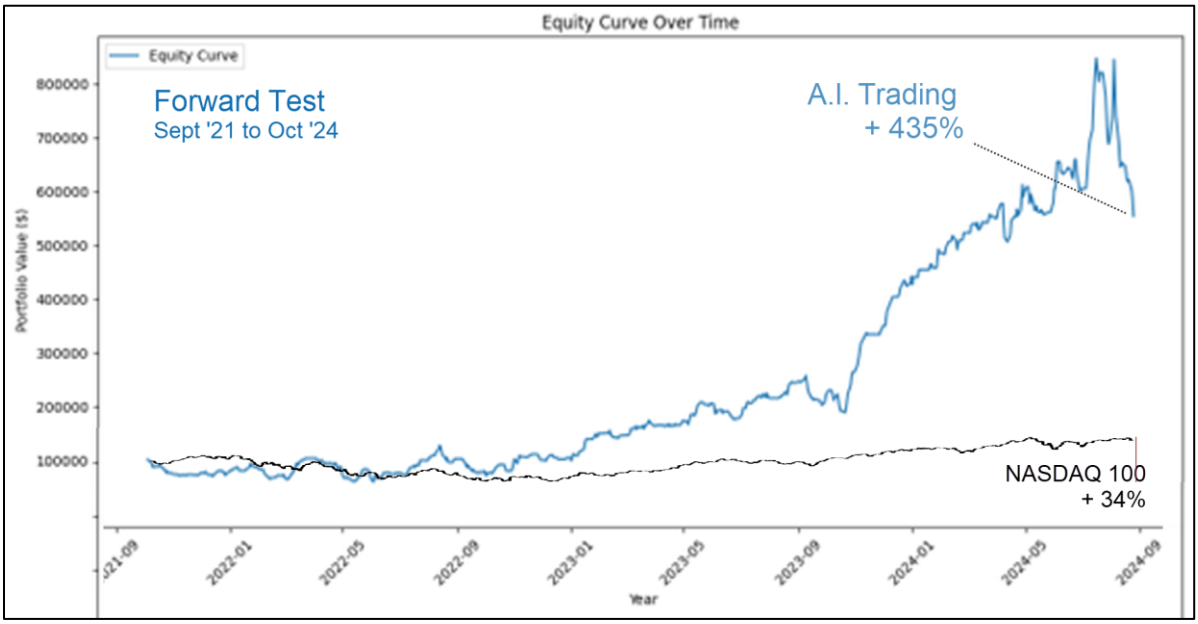
Reinforcement Learning
with Transformers Trading Environment

by Carolina Craus & Ed Downs

October 31, 2024

Summary

The project began with a straightforward idea: Apply a **Transformer-based model** to predict future returns in financial markets and couple it with a custom-built **Reinforcement Learning (RL) environment** for trading decisions. After considerably refining the model, we recently arrived at the following Forward Test trading the symbol QQQ. An overlay of the NASDAQ 100 is shown for reference.



Recent result achieved in Forward Test trading after adjusting the Reinforcement Model

Clearly, the out of sample "Forward Test" run on this model trading QQQ is impressive and exciting. While there is a lot to validate following this test, we believe this is a huge step forward in terms of the Model's ability to learn how to trade.

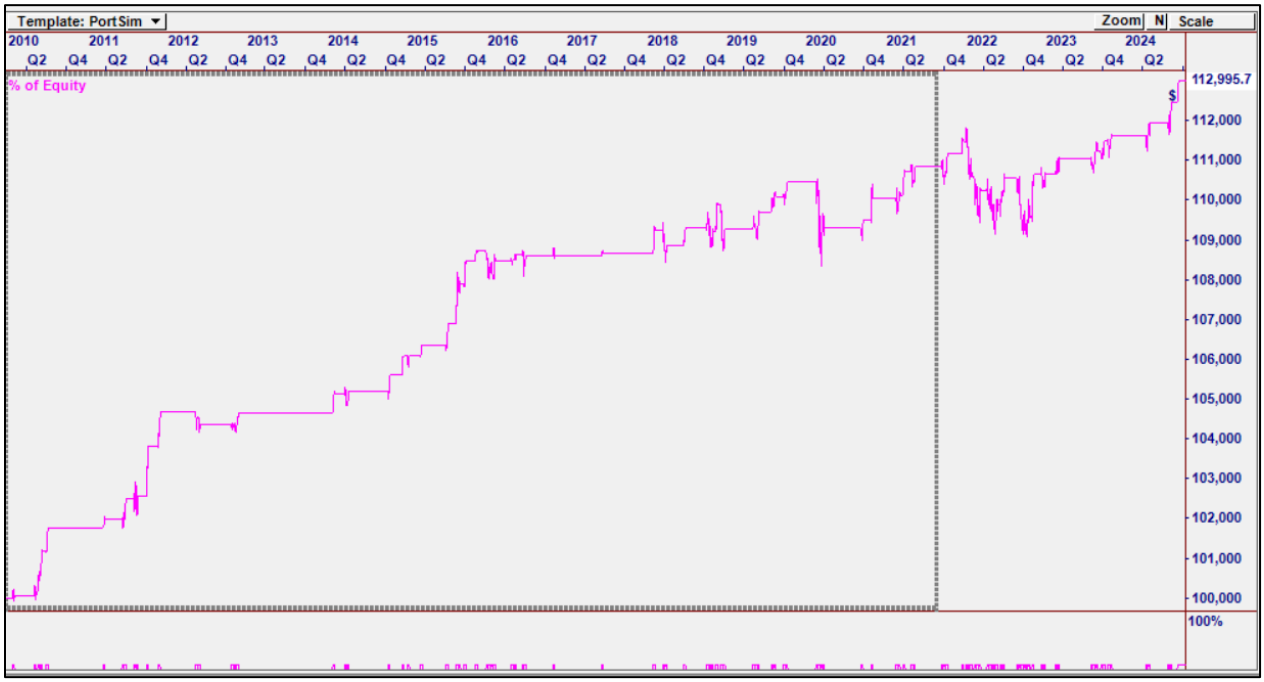
The following sections cover the technology improvements that have led us to this point. We are now working to addressing prediction anomalies that caused the draw down at the end of this experiment and expect to have an improved model soon .

Using the GA to Test Initial Features

The features are the specific factors the A.I. is using to learn from. In our case, we start with Nirvana's Genetic Algorithm, feeding it various technical measurements as the feature set:

Features:

- BOL_MIDDLE(13,2)
- AROSC(14)
- CoralTrend(21,0.4)
- RSI(14)
- TRII(60,30)
- ULT(7,14,28)
- MACD(12,26)
- VTY_PRICE(14,5)
- C>EMA(6)
- C>EMA(14)
- LNREG_SLOPE(14)
- SEACOR (20, 1990, 2019)
- 100*(MACD_HIST (12, 26, 9)-SMA(MACD_HIST (12, 26, 9), 50))/C



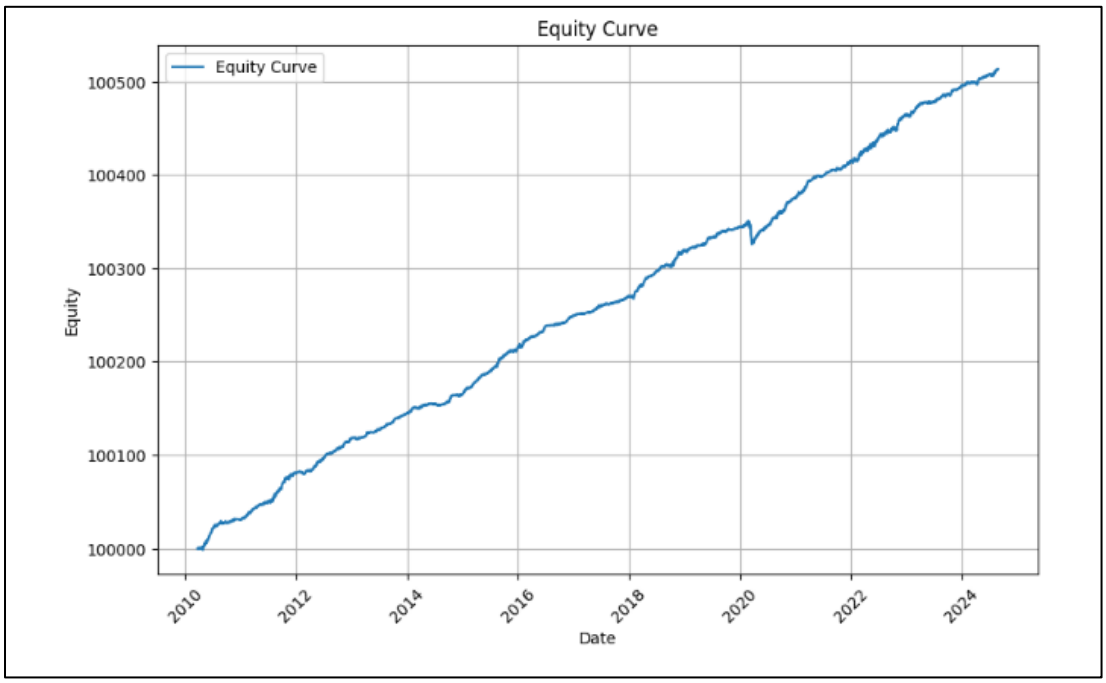
An early GA runs on these inputs. We can see profitability in the forward test, so the goal from here is to see how much this can be improved using our Transformer/Reinforcement Learning models.

Early Transformer Experiments

The Transformer was trained from scratch alongside the RL agent, and the model made predictions directly from historical financial data. The RL agent used these predictions to decide whether to **buy**, **sell**, or **hold** assets based on the projected future returns.

This setup, while functional, had its limitations. Since the Transformer and RL model were trained together, it lacked flexibility—every time we ran an experiment, we had to retrain the entire model, slowing down the process and limiting our ability to test improvements in a modular way. Additionally, we were using a custom RL approach that didn't take full advantage of more robust, pre-built algorithms.

The initial model did not have an out of sample testing period and the buy/sell/trade RL agent logic was incorrect and not interpreting the Target Percent Profit correctly, which is shown in the equity curve below.



One of the early Transformer Training Runs

Refining the Model

Overview

This environment tracks the actions of the agent over time and evaluates them based on their effect on profit/loss over a lag period (e.g., 5 bars).

By comparing the action taken 5 bars ago with the current Target value, the agent is rewarded for making correct buy/sell decisions. The environment is simulated with a starting cash balance of $100,000.

Key Changes and Improvements

1. **Reinforcement Learning Approach**
   - Previous Implementation: Custom Reinforcement Learning (RL) environment based with no built-in RL algorithm support.
   - Current Implementation: Implemented Stable-Baselines3's PPO algorithm, a well-known policy gradient method for RL which is better suited for trading scenarios due to its stability and effectiveness in handling continuous and discrete action spaces.
     - **Evaluation Callback**: Monitors and saves the best-performing PPO model during training.
2. **Transformer Model Integration**
   - Previous Implementation: The Transformer model and RL trading environment were trained together, which limited flexibility.
   - Current Implementation: The Transformer model (for predicting the direction of future returns) is pre-trained and re-used without having to re-train for every fun, enhancing modularity and efficiency of the experimentation process.

Learning Over Iterations

What "Learning over Iterations" Looks Like in Practice:

- At the start of training, the RL agent's policy might be poor, leading to random or suboptimal decisions.
- As the agent goes through more iterations:
  - It **collects more data** from interacting with the environment.
  - It **learns from its mistakes** by adjusting its policy to avoid actions that lead to bad outcomes.

Example:

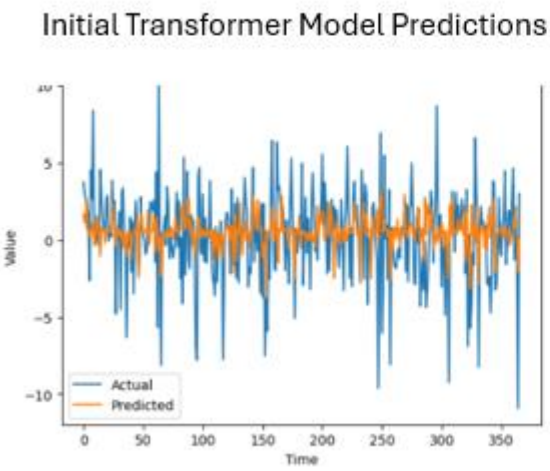
If an RL agent is learning to play a game over 50,000 time steps, it might divide this into 50 iterations where each iteration involves:

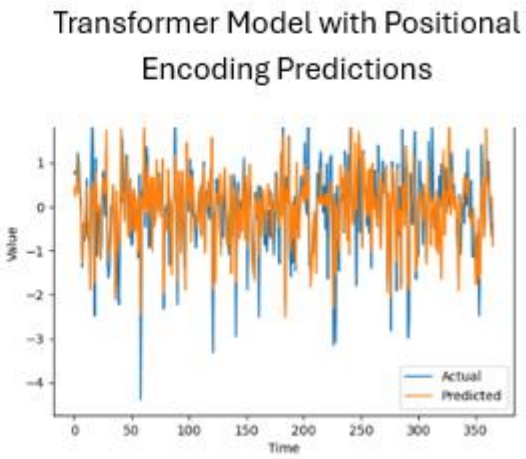
- The agent playing the game for 1,000 steps.
- After those 1,000 steps, the agent updates its policy based on the rewards it received.
- This process repeats for 50 iterations (totaling 50,000 steps), and with each iteration, the agent gets better at playing the game.

Transformer Model Predictions

The initial transformer model had very basic functionality and did not use standardized features. As can be seen on the left, the predictions were not very accurate compared to the actual values.

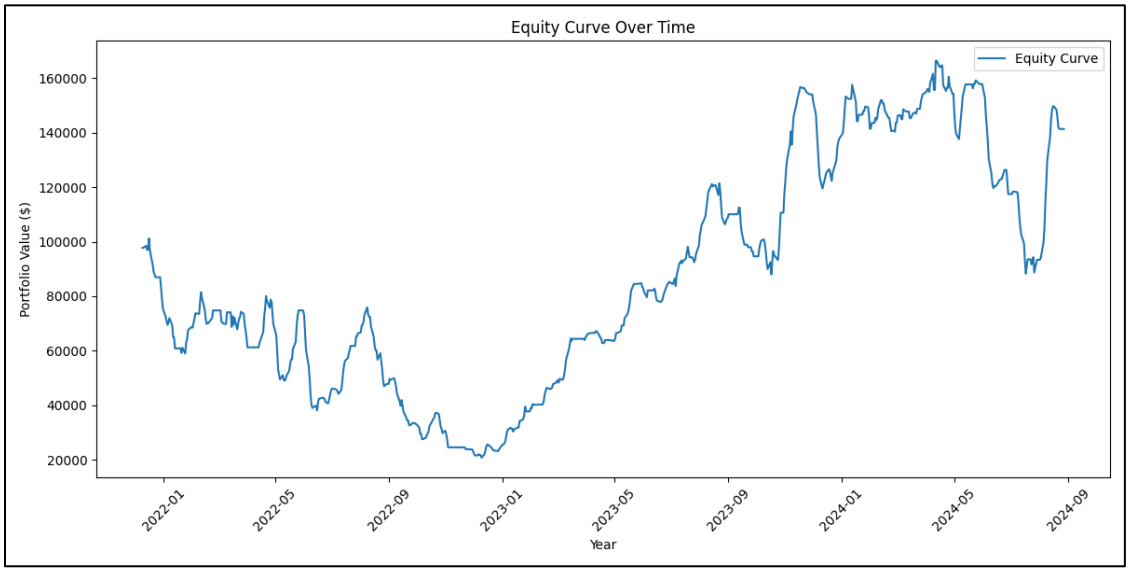
After standardizing the features using Z-Score normalization and implementing Positional Encoding into the transformer model, the predictions were much more aligned with the actual values as can be seen on the right. Positional encoding by giving it the ability to recognize the order or sequence of the data which is crucial to for time series and sequential data and allows the model to capture temporal dependencies. For example,





since in trading a spike in prices followed by a drop conveys different information than the reverse, positional encoding allows the model to interpret these differences based on their temporal order.

After improving the buy/sell/trade logic and incorporating an out-of-sample testing period, the RL agent can be seen making better decisions but with some high drawdowns.

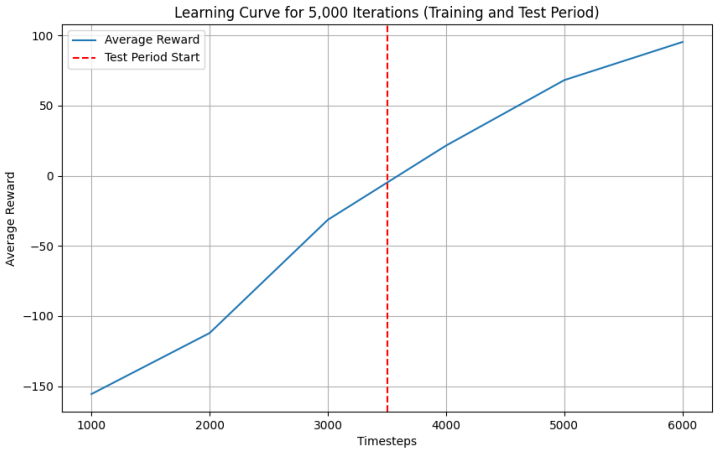


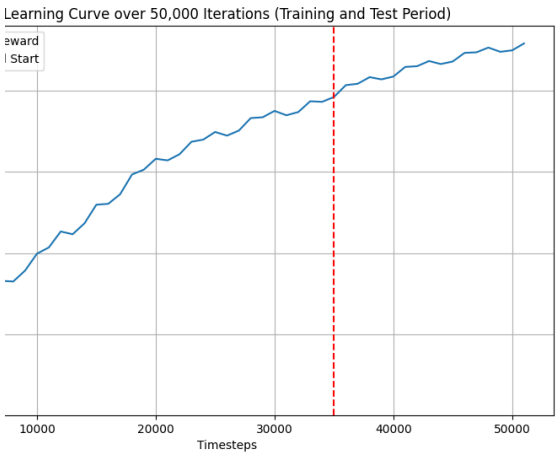
Forward Test output from the Model

Increasing Iterations

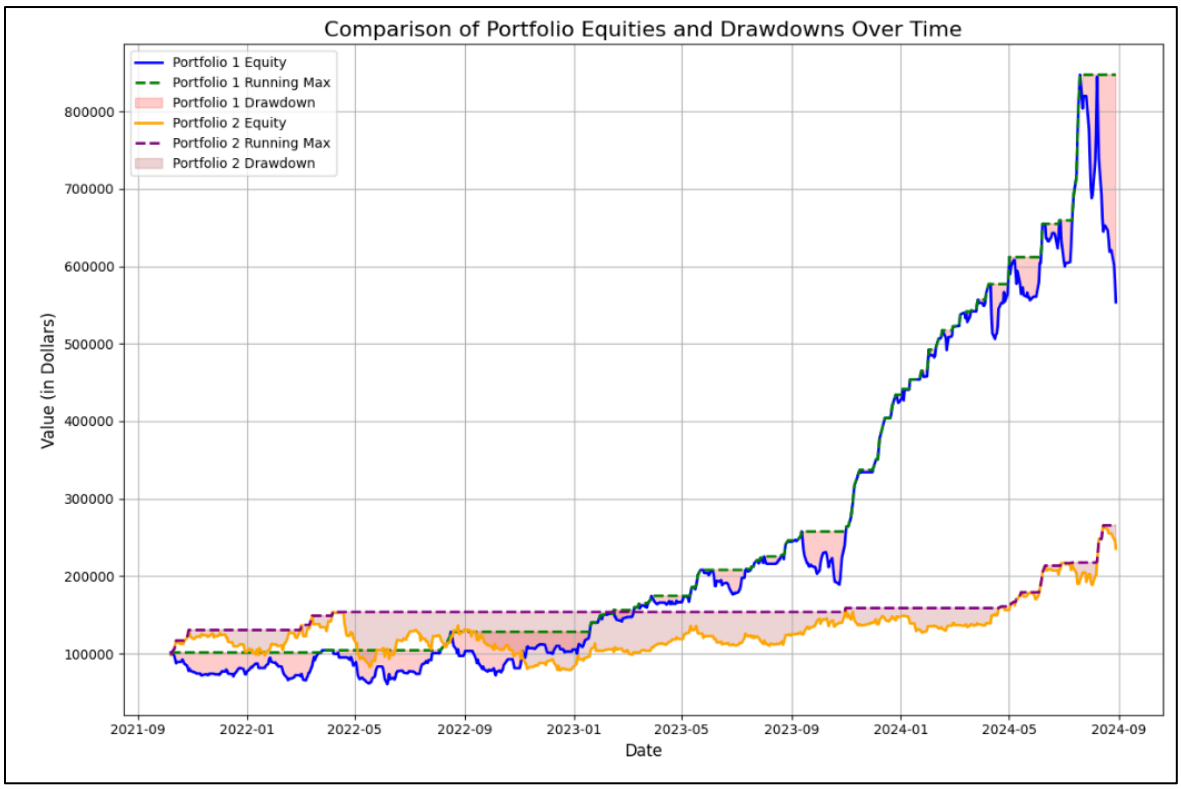
Since the Model appeared to be learning, we let it run over an extended period of time, increasing from 5,000 to 50,000 iterations.

Below, the Learning Curves are shown for 5,000 iterations (left) and 50,000 iterations (right). This plot shows the progression of the average reward over time and highlights the rate at which the model is learning at higher iterations.





Here is an equity curve comparison between the two runs (5,000 vs. 50,000 iterations):



Results for 5,000 Iterations (Portfolio 2) and 50,000 Iterations (Portfolio 1)

The Max Drawdown at the end of the Out-of-Sample testing occurs due to RL agent not having as much previous data to learn from but can be further tweaked and investigated.

Conclusion

From this point, we are investigating the instability seen at the end of the trading curve for QQQ and are also training other symbols, notably AAPL and NVDA for additional validation on the approach. We expect this work to lead to a more robust training model.

Appendix:

Transformer-Based Reinforcement Learning Trading Model

Overview

This report details the implementation of a trading model that combines a Transformer architecture with Reinforcement Learning (RL) techniques. The model is designed to predict and execute trades based on technical indicators for the QQQ stock.

Data Preprocessing

1. **Data Loading**:
2. **Feature Scaling**:
    - StandardScaler is applied to normalize all features, including the target variable. This ensures all input features are on the same scale, which is crucial for the model's performance.
3. **Target Variable**:
    - The 'Target' column represents the percentage profit and is used as the main objective for the model.

Model Architecture

The code defines two main components:

1. **Transformer Model**
    - A custom Transformer encoder is defined, including multi-head attention and feed-forward layers.
    - The model is built using TensorFlow and Keras.
    - A custom Mean Absolute Error (MAE) loss function is defined.
2. **Reinforcement Learning Environment**:
    - A custom `TradingEnv` class is created, implementing the OpenAI Gym interface.
    - The environment manages the trading simulation, including cash balance, positions, and rewards.
    - Integrates the Transformer model's predictions into the trading decision process.

Transformer Model

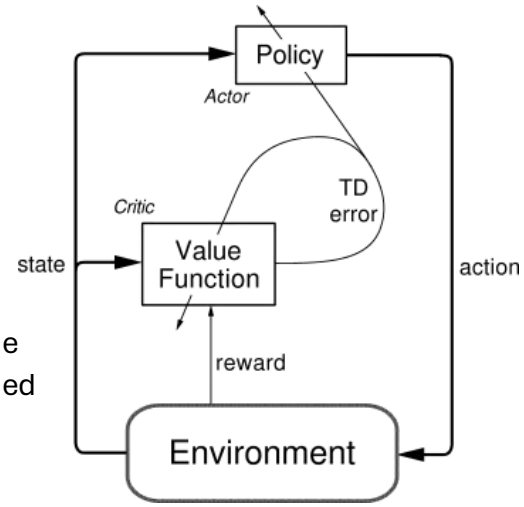
1. **Architecture**:
   - Input shape: (`sequence_length, number_of_features`)
   - Multiple Transformer encoder layers
   - Global Average Pooling
   - Final Dense layer with linear activation
2. **Training Process**:
   - Data is split into training (80%), validation (10%), and test (10%) sets.
   - The model is trained for 50 epochs with a batch size of 64.
   - Adam optimizer is used with the custom MAE loss function.

Reinforcement Learning Setup

1. **State Space:**
   - Includes the last 55 time steps of feature data.
   - Appends the Transformer model's prediction to each state.
2. **Action Space**:
   - Discrete action space with 3 possible actions: hold (0), buy (1), sell (2).
3. **Observation Space**:
   - Continuous observation space representing the last `context_length` time steps of feature data.
4. **Reward Function**:
   - Rewards are calculated based on the profit/loss of each position.
   - The cash balance is updated accordingly.
5. **Agent**:
   - The Proximal Policy Optimization (PPO) algorithm from Stable Baselines3 is used as the RL agent.
6. **Trading Logic**
   - The Transformer model predicts the next target value.
   - Actions are determined by comparing the predicted value to the current value:
     - Buy if predicted > current
     - Sell if predicted < current Hold if predicted = current

PPO Agent with MLP Policy

1. **Proximal Policy Optimization (PPO)**:
   - PPO is an on-policy algorithm that aims to improve the stability of policy updates.
   - Key features:
     - Uses "clipping" to prevent large policy changes in a single update.
     - Balances exploration and exploitation through entropy regularization.
     - Employs a value function to estimate the expected returns, aiding in more informed decision-making.

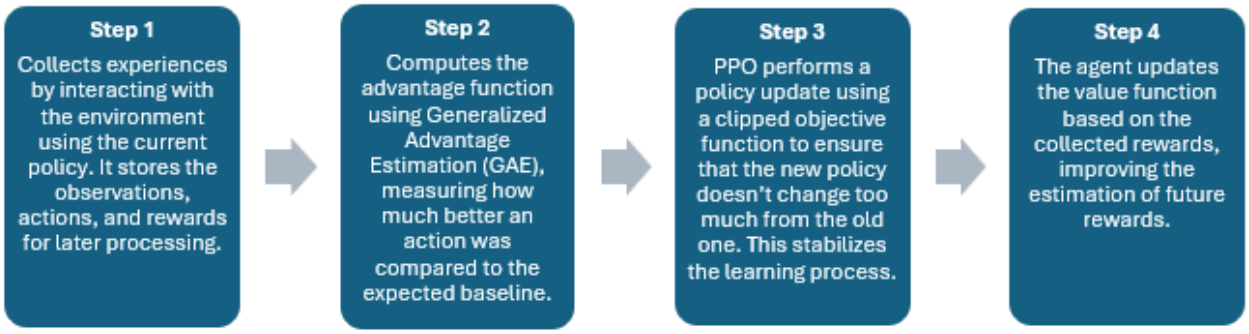


2. **Multi-Layer Perceptron (MLP) Policy**:
   - The MLP policy is a neural network that maps the state (including Transformer predictions) to actions.
   - Capable of handling both continuous and discrete action spaces.
   - Processes multi-dimensional input, making it suitable for the complex state representation in this trading environment.

3. **Training Process**:
   - The PPO agent interacts with the trading environment over multiple episodes.
   - It collects experiences (state, action, reward, next state) and uses them to update its policy and value function.
   - The policy is updated iteratively, with each update aiming to improve the agent's performance while avoiding drastic changes.

4. **Advantages of PPO for Trading**:
   - Stability: The clipping mechanism prevents sudden, large changes in policy, which is crucial in the volatile trading environment.
   - Sample Efficiency: PPO can learn from a relatively small number of interactions with the environment.
   - Continuous Learning: The agent continuously updates its policy based on rewards (profits/losses) from past decisions, adapting to changing market conditions.

RL PPO Agent Process



The following parameters were added to the Reinforcement Learning agent:

1. Gamma (γ): Discount Factor

The **gamma** (γ) parameter is a **discount factor** that determines how much the agent values future rewards compared to immediate rewards. It influences how far into the future the agent should consider rewards when making decisions.

gamma = 0.99

Makes model more far-sighted, considering long-term rewards in decision-making and balances immediate and future rewards, giving the agent foresight into the impact of its current actions.

2. GAE Lambda (λ): Generalized Advantage Estimation Lambda

GAE Lambda (λ) is part of **Generalized Advantage Estimation (GAE)**, which is used to reduce the variance of the policy gradient while still keeping the bias low. It controls how much weight is given to long-term versus short-term advantage estimates.

gae_lambda = 0.95

More reliant on long-term rewards, making reinforcement agent more reliable and strikes a balance between variance and bias, leading to more stable learning while still being responsive to both immediate and future rewards.

3. Entropy Coefficient (ent_coef): Encouraging Exploration

The **entropy coefficient** (ent_coef) controls the amount of **exploration** by encouraging the agent to keep its policy unpredictable. It adds an entropy bonus to the loss function, which pushes the agent to explore more diverse actions, rather than sticking to the actions it currently believes are optimal.

ent_coef = 0.01

Increases the amount of exploration, encouraging the agent to take a wider variety of actions, which helps prevent premature convergence to suboptimal policies.

Training Process

1. The environment is initialized and wrapped in a `DummyVecEnv` for compatibility with Stable Baselines3.
2. A PPO model is instantiated with the '`MlpPolicy`'.
3. The model is trained for a specified number of timesteps (10,000 in this case).

Execution & Evaluation

1. **Data Preparation**:
   - Load and preprocess the data.
   - Create sequences for Transformer model training.
2. **Model Training**:
   - Build and train the Transformer model. ○ Save the trained model for future use.
3. **Trading Simulation**:
   - Initialize the trading environment with the trained Transformer model.
   - Run a single episode, making trading decisions at each step. ○ Record actions, rewards, and equity changes.
4. **Performance Evaluation**:
   - Calculate key metrics: Total Return
     - Max Drawdown
     - Annual Rate of Return
     - Final Equity

Considerations and Potential Improvements

1. **Hyperparameter Tuning** for both the PPO agent and the environment (e.g., context length) could optimize performance.
2. **Extended Training**: Increasing the number of epochs or using early stopping might improve the Transformer's performance.
3. **Additional Features** like transaction costs or more complex trading rules could make the simulation more realistic.
4. **Transaction Costs**: Incorporating trading fees would make the simulation more realistic.
5. **Risk Management**: Implementing stop-loss or position sizing strategies could improve overall performance.