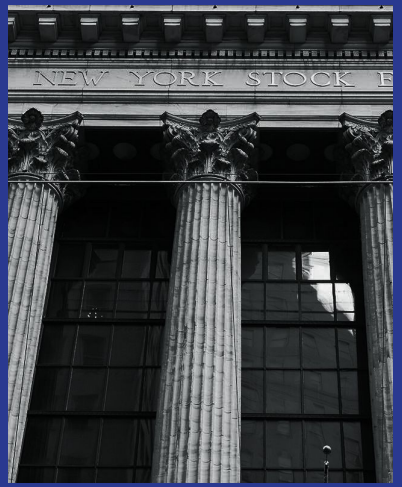
# P. Optimizers: RL Stock Trading




Adeet Patel

Anthony Chen

Iktae Kim

Akram Reshad

# Agenda

- Background of stock trading
- Overview of our problem
- Methods
- Agents used
- Our twist: Day Trading, Ensemble Strategy
- Other attempts/failures: Decision Transformers
- Evaluation of our results
- Other Considerations

# Background Knowledge (1 of 2)

- **Stock Market:** A public market where shares of companies are bought and sold, enabling businesses to raise capital and investors to potentially earn returns.
- **Day Trading:** The practice of buying and selling financial instruments within a single trading day, aiming to capitalize on short-term price fluctuations.
- **Technical Analysis:** A method of analyzing historical price and volume data to predict future price movements using charts, patterns, and indicators.

# Background Knowledge (2 of 2)

- **Trading Bot:** An algorithmic system that autonomously executes trades based on predefined rules or learned strategies, often using technical analysis to inform decisions.

- A stock trading's dynamic environment provides ample opportunities to apply reinforcement learning techniques when it comes to deciding what actionable risk to take, depending on the fluctuations of closing prices, to generate a high value return.

# Our Project Goal:

Can we use Reinforcement Learning methods to understand technical analysis features and generate day trading signals?


BUY
AND
HOLD
DAY
TRADING

# Problem Definition

- State space: {all 2D real-valued arrays with shape (window size, # of features)}
  - Window size = # of past ticks for which we look at features

- Action space: [sell 1 share, hold existing shares, buy 1 share]

- Reward: the profit or loss achieved from executing that trade
  - If buy or hold: 0
  - If sell: current price minus average price at which existing shares were bought

# How we obtained our dataset

We used the yfinance package to webscrape 2 years worth of hourly stock data from **S&P 500 (SPY) ETF***

Time period: 04-14-2021 until 04-14-2023

Δ Price over time period:  0.15%

Constraints preventing us from using more data:

- Scrape limits
- Cost skyrockets for longer and more granular data

We also used the technical analysis indicators library to engineer new features to use later on

```
pip install ta
```

These include momentum, volume, volatility, and trend indicators

*Exchange traded fund, mutual fund that can be bought and sold like a regular stock

# Custom Environment

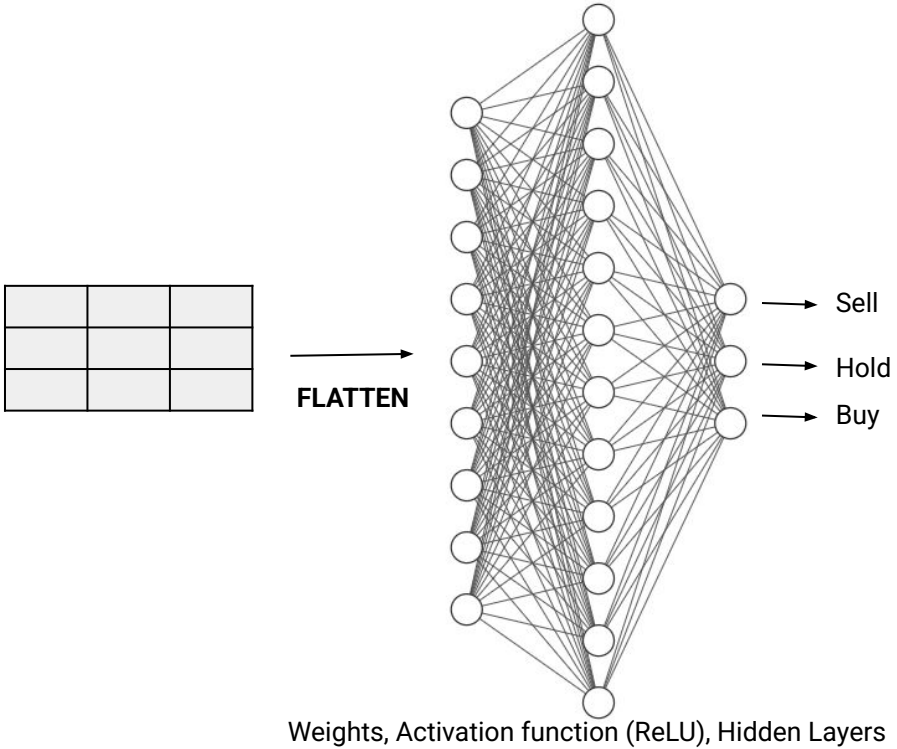
- We built upon the gym-anytrading StocksEnv environment and customized it to suit our goal, for the following reasons:
  - Feeding specific features to the agent (i.e. defining our own state space)
  - Keeping track of both realized and unrealized gains
    - Realized gain/loss = money actually earned or lost after selling a stock
    - Unrealized gain/loss = money that would be earned or lost after selling a stock
  - Only buy or sell 1 share at a time
- We keep track of the total number of shares currently held by the agent
- We reward / penalize the agent for unrealized gains / losses

# Methods and Algorithms We Decided to Use

- Policy gradient methods
  - Customized policy neural network
  - Stable baselines agents: PPO, DQN, A2C
  - Ensemble of the above methods

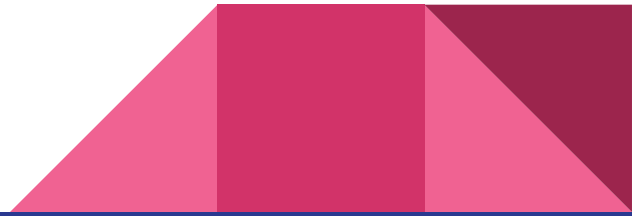
# Our Custom Agent

- We implemented this policy gradient method from scratch in PyTorch
- We approximate the policy using a function (a neural network) trained over episodes
    - **Input:** state array of shape (window size, # of features)
    - **Output:** a log-probability distribution of actions that the agent can sample from → allows for better exploration



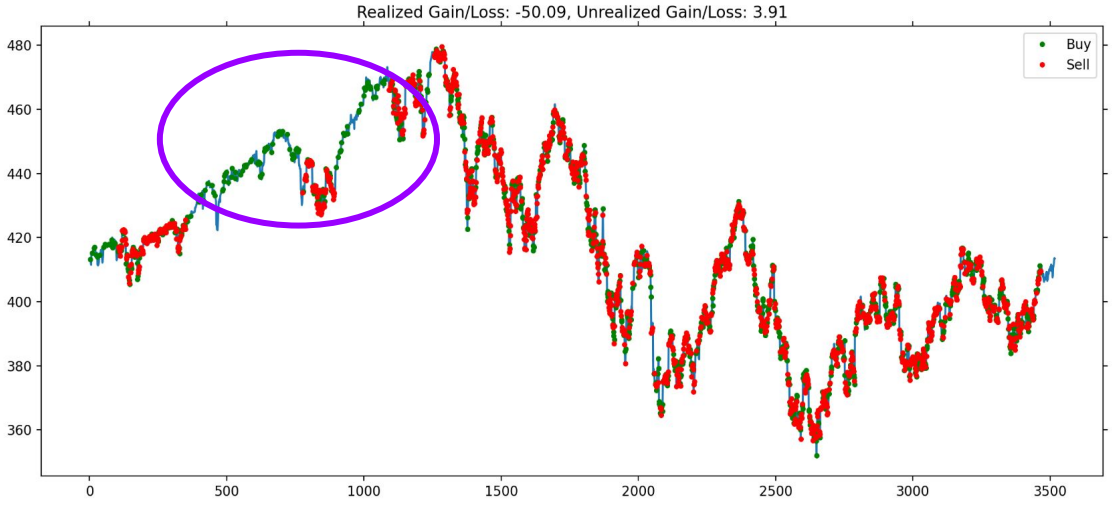
# Evaluation Hyperparameters

- Window size = 50
- Epochs = 100
- Hidden nodes = 200
- Max shares = 100
- Learning rate = 1e-3
- Learning rate scheduler = True



# Evaluations of Our Custom Agent

- We see that our agent bought some shares initially, and sold after some holding
- The agent also shows that it made some mistakes early on by buying too many times in a row when the stocks prices were rising, and sold when they dropped.
- It then bought many shares again and then sold before the prices dropped too low.
- Soon it learned to buy and sell with small holding periods to achieve an overall loss. Because our model chooses to hold, we also calculated an unrealized gain since we can't immediately see the continuing period.
- Thus, some of the model's purchases (net loss) was intended as a future potential sell.



# Deep dive into data and our “twist” on this RL problem

# Engineered Features We Used In Our dataset

These are the features we selected using the technical analysis indicators library: **'volatility_atr', 'volume_obv', 'momentum_stoch_rsi', 'trend_vortex_ind_pos', 'trend_macd_diff'**

**'Volatility_atr': Volatility Average True Range** indicator shows the degree of price volatility.

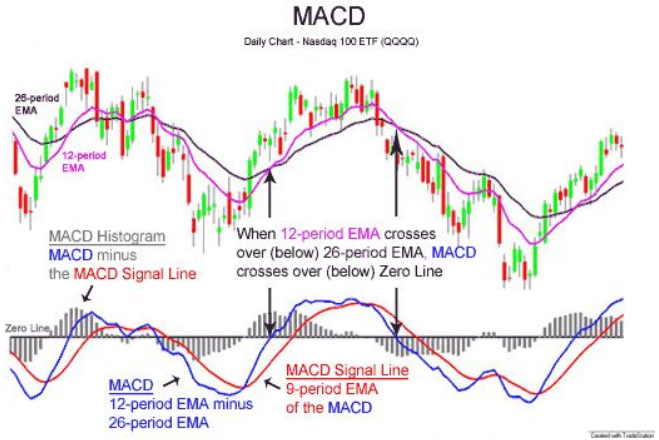
**'Volume_obv'**: **On-balance volume** relates price and volume in the stock market, and is based on cumulative total volume.

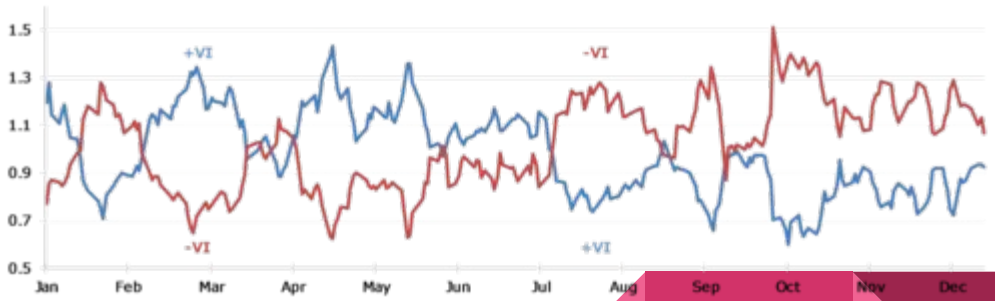
**Momentum_stoch_rsi'**: **Momentum Stochastic RSI** compares the magnitude of recent gains and losses over a specified time period to measure speed and change of price movements of a security. It is used to identify overbought or oversold conditions in the trading of an asset.

# Engineered Features We Used In Our dataset

**'Trend_vortex_ind_pos'**: **Trend vortex** consists of 2 oscillators that capture positive and negative trend movement. Bullish signal occurs when the positive trend indicator crosses above the negative trend indicator or a key level.

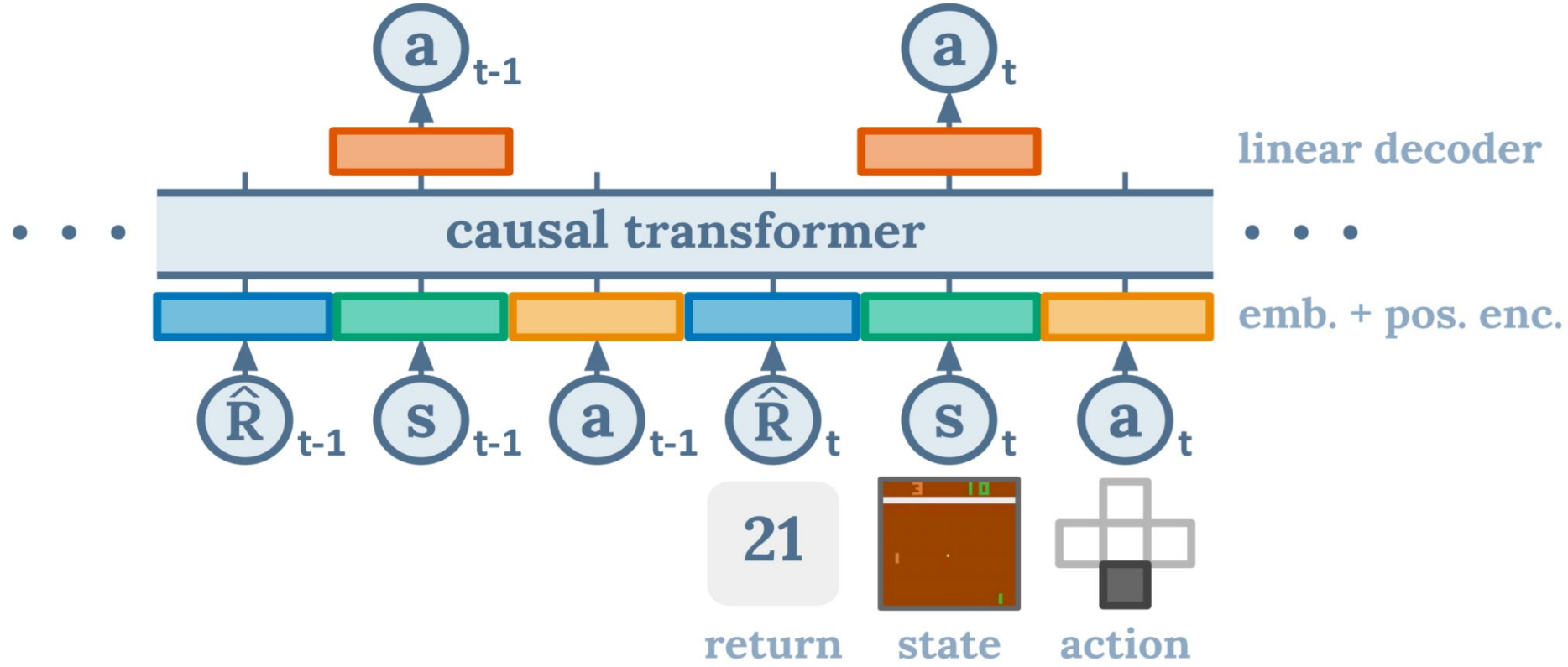
**'Trend_macd_diff'**: **Moving Average Convergence Divergence difference**. Shows the relationship between MACD and MACD Signal. Where MACD is a trend-following momentum indicator that shows the relationship between two moving averages of prices. The signal line helps identify trend changes in the price of security and to confirm the strength of a trend.



# Decision Transformer

- Attention is All You Need - Google Brain, University of Toronto
  - Revolutionized NLP
- GPT model -  OpenAI
  - First Major LLM - Optimized Word Embeddings
- Decision Transformer - Facebook AI Research, Google Brain
  - Innovation of using Transformer in RL
  - Offline Learning
- Online Decision Transformer - Facebook AI
  - Offline and Online Learning
  - HardCoded For Mujuco

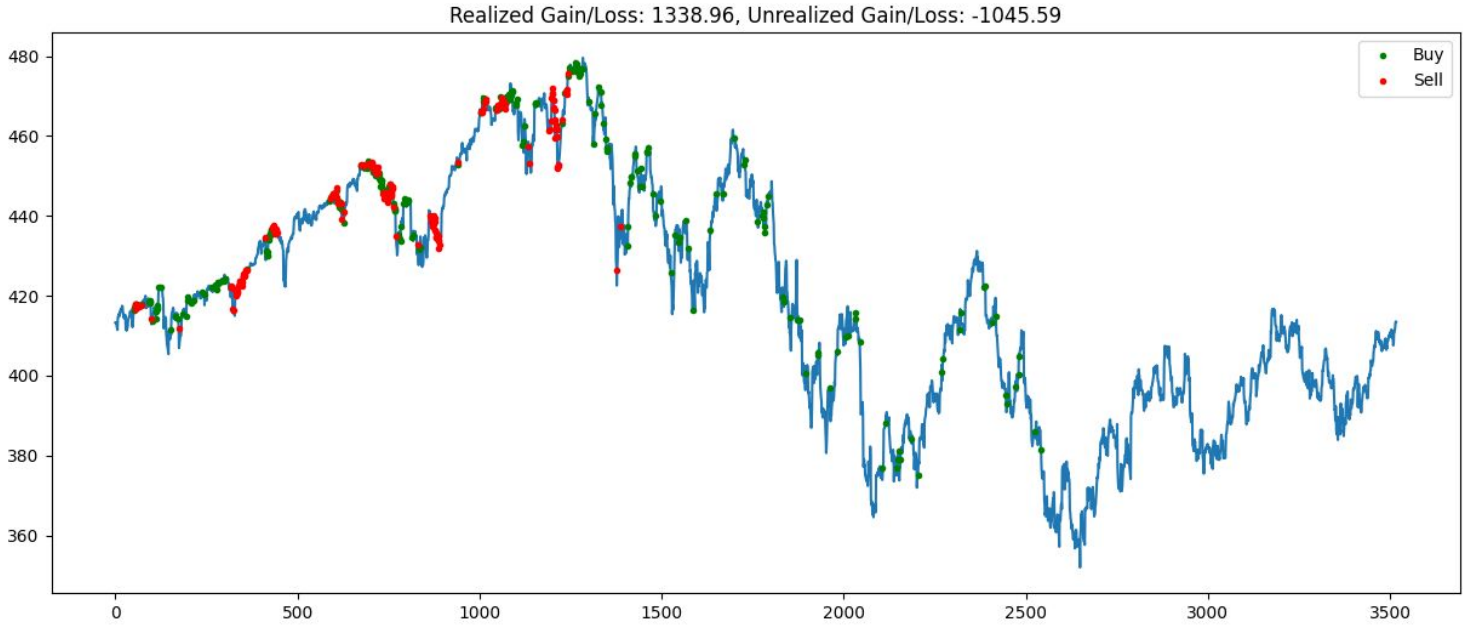

a t-1
a t
linear decoder
causal transformer
emb. + pos. enc.
R̂ t-1
s t-1
a t-1
R̂ t
s t
a t
21
return
state
action

# Ensemble Strategy

- Train each Agent (PPO, A2C, DQN, OurCustomAgent) separately, on a certain number of timesteps each
- At each step we determine what would be the majority action between the four agents, and take that action

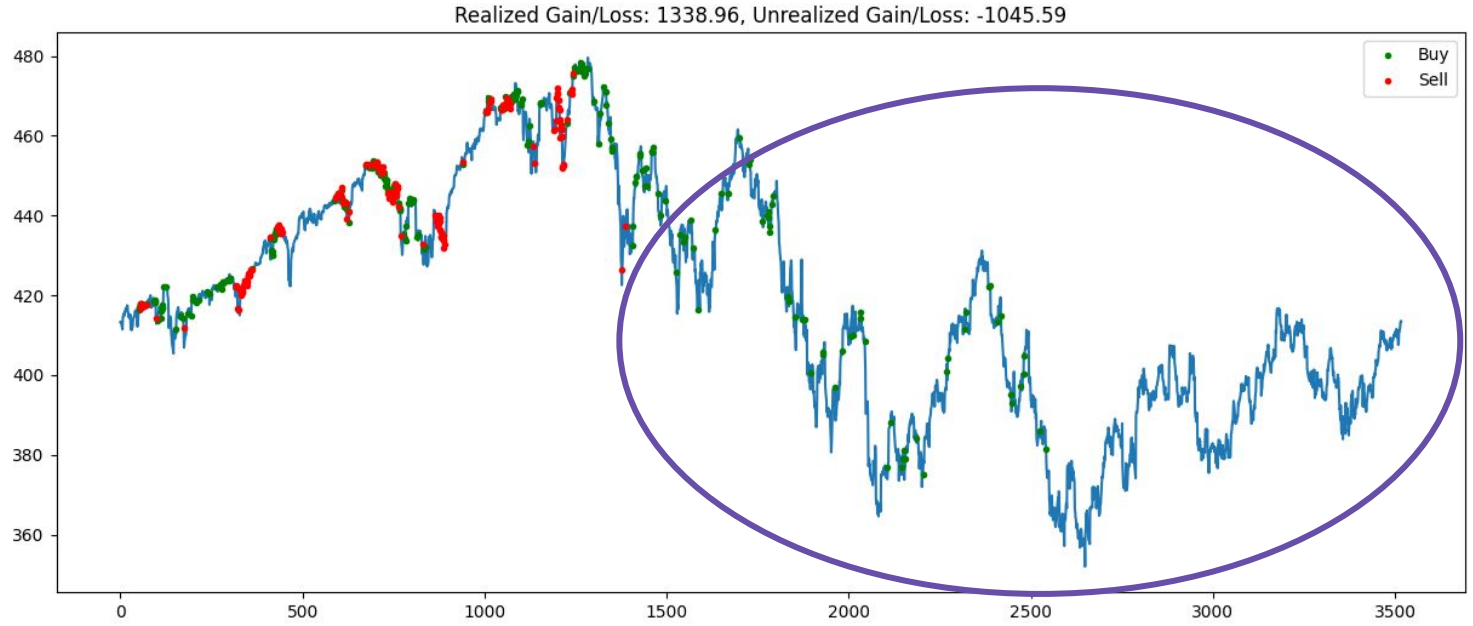
```
actions = [int(ppo_action), int(a2c_action), int(dqn_action), int(custom_action)]
      # If we have a tie (no majority):
      if check_tie(actions):
          majority_action = Actions.Buy.value
      else:
          majority_action = Counter(actions).most_common(1)[0][0]
      obs, reward, done, info = env.step(majority_action)
```

# Visualization of Results With **PPO** Method (timestep=100,000)



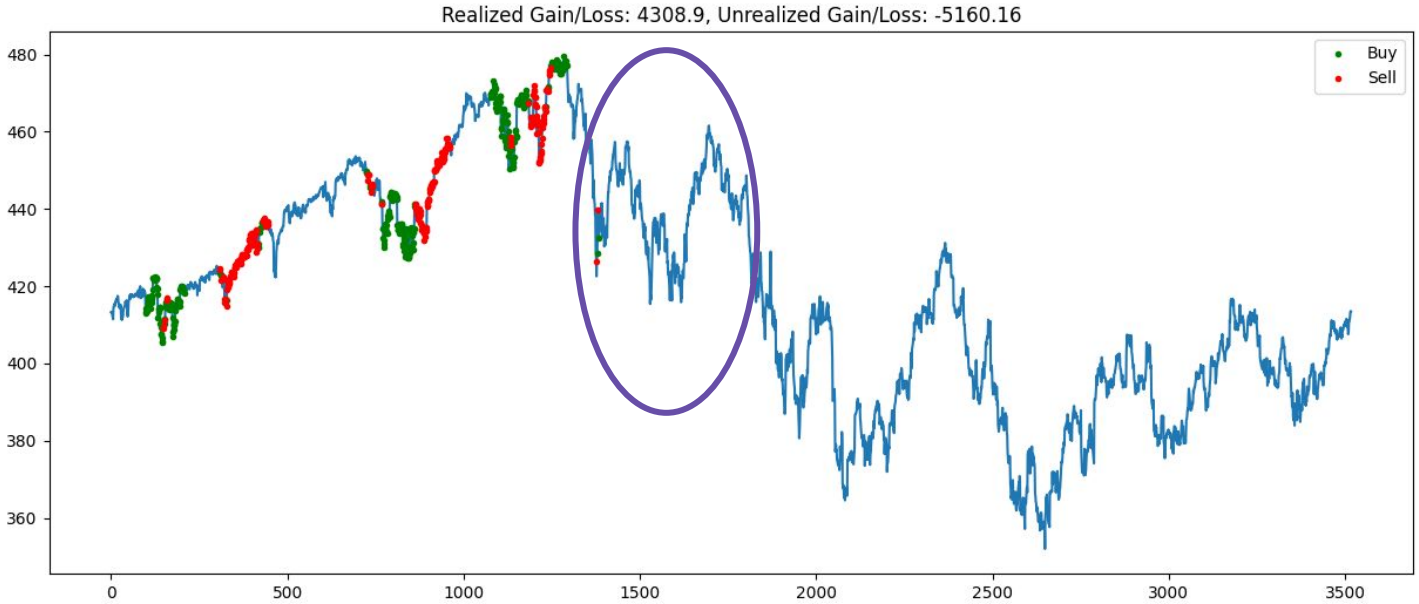

The PPO Agent seems to learn to day trade

# Visualization of Results With **PPO** Method (timestep=100,000)




While PPO is the State of the art in Reinforcement Learning, it still needs more data / PPO could be holding stocks at the end in anticipation of another uptrend.

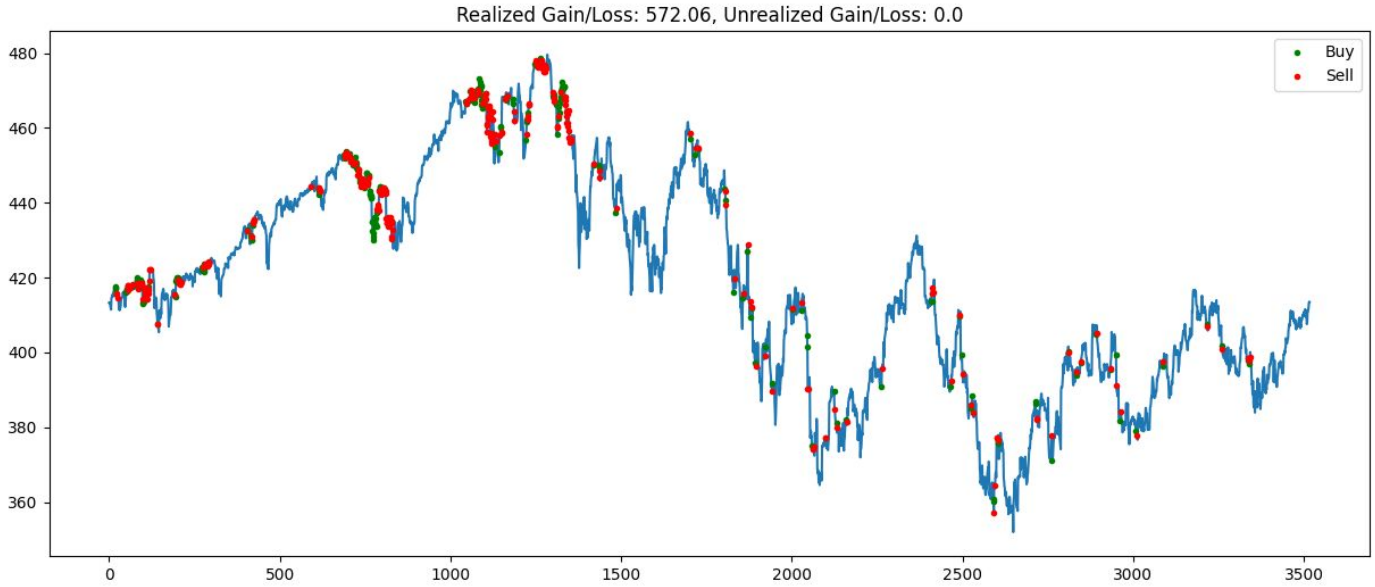
# Visualization of Results With **A2C** Method (timestep=100,000)




A2C finds the local minima of uptrends and learns only this, but never evaluates downtrends so it holds for long periods.

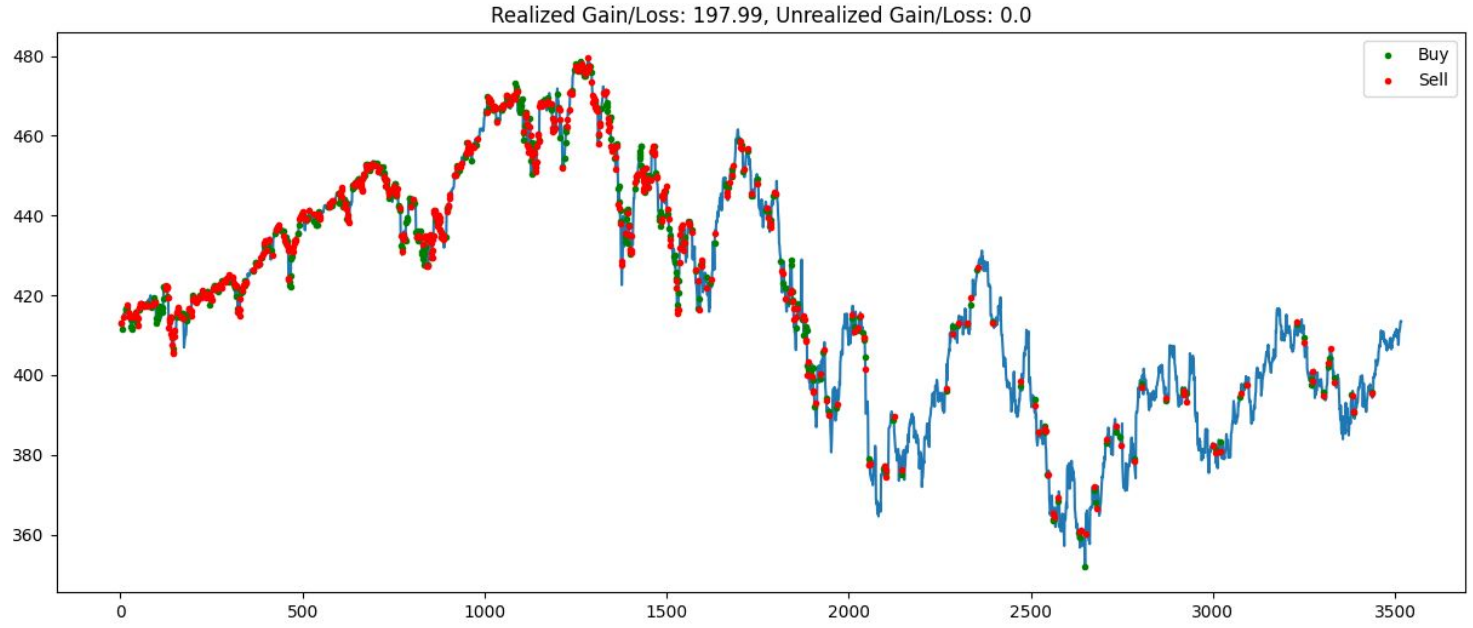
So it decides not to sell when it gets stuck. It is unwilling to make some losses to earn back later.

# Visualization of Results With **A2C** Method (timestep=500,000)

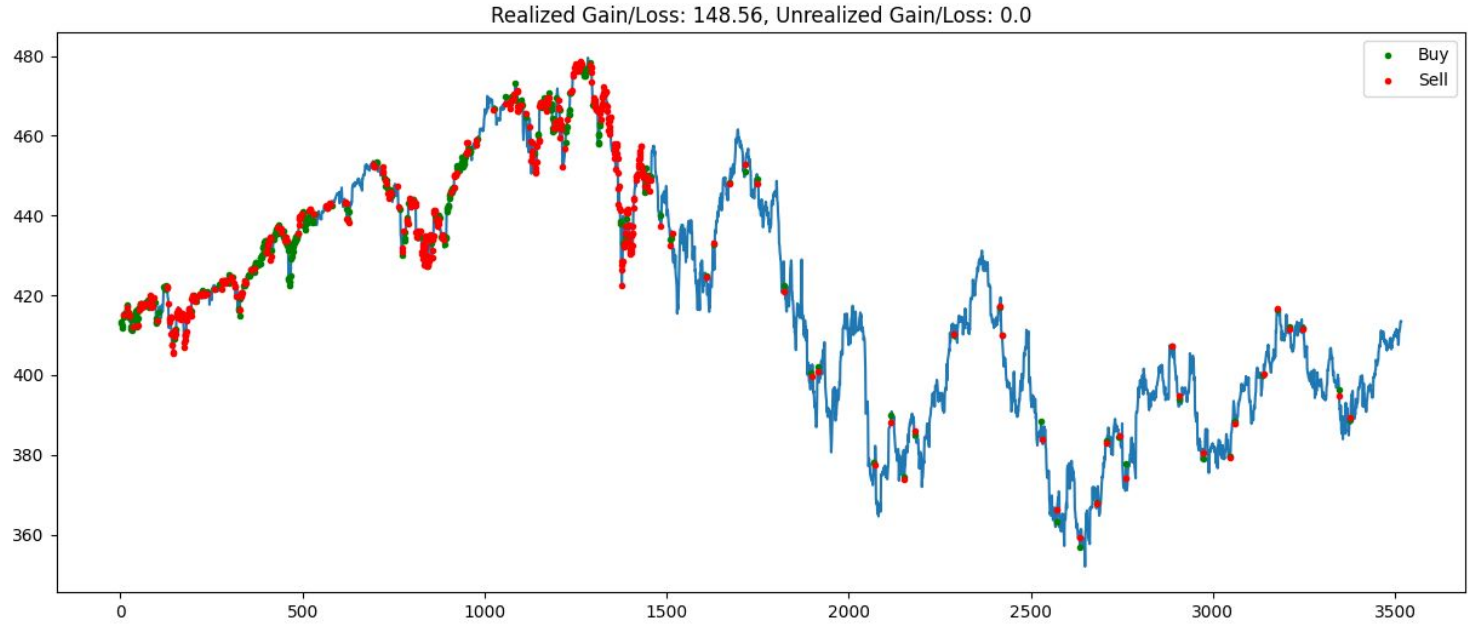



We believe for the 500,000 timesteps, the model overfits, and is learning there is a specific endpoint to the dataset, which in reality does not exist

# Visualization of Results With **DQN** Method (timestep=100,000)

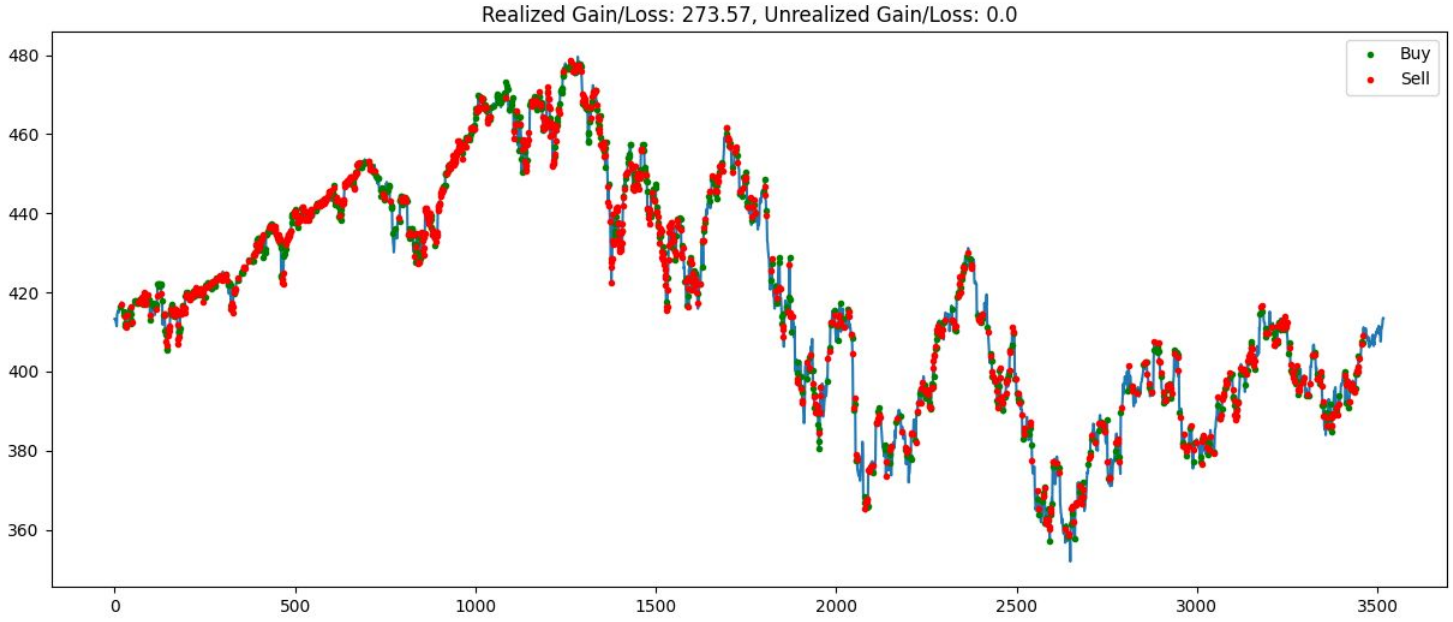


# Visualization of Results With **DQN** Method (timestep=500,000)



# Visualization of Results With **Ensemble** Method

(CustomAgent epoch = 100, timesteps = 100,000)




Buys and sells quickly throughout

# Future considerations/ Improvements

- Hyperparameter tuning: window size, epochs, hidden nodes,  max shares,  learning rate, and learning rate scheduler to optimize performance (maximizing profit)
- Experimentation with different features; Requires more Technical Analysis expertise
- Incorporating language data from financial statements, news, or alternative data (X Corp.) using NLP vectorized-word embeddings
- A different ensemble strategy, using Sharpe ratio* to select actions; utilizing timestamps to calculate historical turbulence or volatility
- Multi-agent approach: Particle Swarm Optimization

*ratio to compare against a risk-free asset such as a U.S. Treasury Bill for potential excess return

# Questions?

# Appendix

# Why we chose PPO

- PPO is an on-policy algorithm, meaning it can directly optimize the policy that the agent uses to interact with the environment, which can result in faster convergence to an optimal policy
- On-policy algorithms may suffer from sample inefficiency, but for PPO, TRPO and Clipping mechanism allows it to be more sample efficient and stable
- PPO has shown to provide a favorable balance between sample complexity, simplicity in implementation and wall time
- *PPO is able to utilize continuous or discrete action spaces

# Why we chose A2C

- A2C maintains both an actor (policy, trading strategy) and a critic (value function, Profit_Return(market movement, trade decision)), which can help the agent balance exploration and exploitation more effectively in a trading environment
- It is deterministic and waits for each actor's experience to end before updating, averaging over all the actors
- A2C is an on-policy algorithm, meaning it learns from its current policy while interacting with the environment. This can be advantageous in trading applications where the environment is non-stationary, and the agent needs to adapt to changing market conditions
- A2C also has the potential to be extended to an asynchronous version (A3C), where multiple parallel agents learn simultaneously
- *A2C is able to utilize continuous or discrete action spaces

# Why we chose DQN

- DQN is an off-policy algorithm, meaning the agent to store and reuse past experiences in a replay buffer, improving sample efficiency and enabling more stable learning
- DQN’s deep learning techniques, allows the agent to learn complex, high-dimensional representations of the environment, which can be useful for capturing intricate patterns in financial data
- *DQN is well-suited for problems with discrete action spaces

# Visualization of Results With **PPO** Method (timestep=500,000)