Reinforcement Learning

At a high level, our RL should enhance the predictive accuracy and adaptability of our stock prediction model using a reinforcement learning layer. This layer will dynamically assign weights to the outputs of our NLP, XGBoost, and LSTM models based on real-time market feedback. By adjusting the influence of each model according to market conditions and recent performance, the RL layer should optimize the combined prediction and improve profitability and accuracy.

State

- We need to include each model's prediction for the price or probability that it increases in the state. We should also include the overall sentiment score from NLP.
- We should add multiple technical indicators and see how they impact performance because they give the RL agent context of the market environment.
    - These indicators can help show under what circumstances each model performs best and use that information to weight each model accordingly.
- We need to do some feature engineering to decide which technical indicators are most relevant.
    - For example, long-term indicators like interest rates or broad market indices that aren't explicitly used in LSTM and XGBoost could add valuable context.
- Keeping the state space simple with only the most relevant features is important to reduce overfitting.
- **Example indicators**: Volatility and a trend indicator like the slope of a moving average could provide the context needed without overwhelming the model.

Here's an example of how the state vector might be structured in code:
To implement it like this we will have to do more data processing

```
state = {
    'xgboost_prediction': xgboost_model.predict(),
    'lstm_prediction': lstm_model.predict(),
    'nlp_sentiment': nlp_model.get_sentiment_score(),
    'volatility': calculate_volatility(),
    'trend_slope': calculate_moving_average_slope(),
    'rolling_accuracy_xgboost': calculate_rolling_accuracy(xgboost_model),
    'rolling_accuracy_lstm': calculate_rolling_accuracy(lstm_model),
    'rolling_accuracy_nlp': calculate_rolling_accuracy(nlp_model)
}
```

Action

- Weighting each model's prediction: The action space will allow the agent to assign continuous weights to each model's prediction. It should be dynamic as well.

Reward

- The reward will be based on the profitability and or accuracy of the weighted prediction relative to the actual price movement.
- If the weighted prediction performs well, the agent should receive a positive reward to reinforce the chosen weighting strategy.

```python
def calculate_reward(actual_price, combined_prediction, prev_price):
    price_change = actual_price - prev_price
    predicted_change = combined_prediction - prev_price
    direction_reward = 1 if (predicted_change * price_change > 0) else -1
    magnitude_reward = abs(predicted_change - price_change)
    return direction_reward - magnitude_reward
```

Reinforcement Learning Algorithm

- Proximal Policy Optimization seems like the best choice based on my research. PPO is a policy gradient method and is well-suited for this application because of it's
    - Ability to handle continuous action spaces, which is ideal for weighting models with fractional values.
    - Stability, which is essential for applications in volatile markets.
    - Capability to optimize weights based on both market conditions and model performance.
    - Direct optimization of continuous values for model weights, which fits our needs perfectly.
- The Stable Baselines3 library has strong support for PPO, which should make implementation somewhat straightforward.

Implementation Plan

Training Phase

- We will conduct backtesting using historical data, split into training and validation sets.
- The agent will learn optimal weighting strategies based on each model's past performance and market conditions.
- Feature engineering and experimentation with technical indicators will help us refine the state representation
- https://spinningup.openai.com/en/latest/algorithms/ppo.html

Evaluation Phase

- The agent's performance will be evaluated on validation data to assess its ability to generalize.
- Testing on out-of-sample data will allow us to see how well the agent is doing and ensure it can adapt to unseen market conditions