Algorithms to be used in the Final Year Project

Sentiment Analysis:

- **Lexicon-Based Models (e.g., VADER, AFINN)**
    - These models are very straightforward to use because they rely on predefined dictionaries of words associated with positive or negative sentiment. If a word appears in the text, the model assigns a sentiment score to it based on the dictionary.
    - While they're simple and lightweight, they struggle to capture the nuances of financial language, like when a word like "loss" might be used positively ("reduced losses").
    - These models work well for quick setups or when resources and training data are limited, but they can't match the complexity of more advanced methods.
- **Machine Learning Models (e.g., Naive Bayes, SVM, Logistic Regression)**
    - These are traditional models that rely on labeled datasets to learn sentiment patterns. They can handle structured data well and are much faster for inference compared to deep learning models.
    - However, these models don't understand context and tend to struggle when faced with complicated or domain-specific language like finance. They're useful when you have a moderate amount of labeled data and need something that's efficient but not overly complex.
- **Deep Learning Models (e.g., LSTM, CNN)**
    - Deep learning models excel at analyzing sequences and capturing relationships within text. For example, they can understand how a sequence of words like "profit margins are rising despite headwinds" indicates a positive sentiment overall.
    - On the downside, these models require a lot of data to perform well, and they're computationally expensive, making them harder to implement without the right resources. They shine when you have a large dataset and want to dive deeper into the text structure.
- **Transformer Models (e.g., FinBERT, BERT, GPT-based Models)**
    - Transformer models are currently the state-of-the-art for text analysis, and they stand out because they are designed to understand the context and meaning of words in relation to each other. FinBERT, in particular, is trained specifically on financial text, making it an excellent choice for analyzing sentiment in news, reports, and social media about stocks or markets.
    - These models do require significant computational power and may need fine-tuning if your text data is highly specific, but they offer unmatched accuracy and depth when it comes to understanding financial sentiment.
- **Hybrid Models (e.g., Knowledge Graphs + Sentiment Analysis, Ensemble Models)**
    - Hybrid approaches combine multiple methods to provide a broader and more nuanced understanding. For example, they might pair sentiment analysis with technical indicators to capture both market trends and sentiment in a single system.
    - While incredibly powerful, these systems are often complex to implement and require more resources, both in terms of computation and data processing. They are ideal for advanced setups where a deeper level of analysis is essential.

Model	Pros	Cons	Best Use Cases
Lexicon Based	Easy to set up, no training data needed, and highly interpretable.	Limited vocabulary and struggles with financial nuances.	Quick insights, small projects, or real-time systems with minimal data requirements.

Machine Learning	Customizable with labeled data, efficient for simpler tasks, and faster inference.	Requires labeled data, cannot capture complex financial context or relationships.	Suitable for moderate-sized projects with labeled datasets and resource constraints.
Deep Learning	Captures complex patterns and relationships in sequential text data effectively.	Requires large datasets and high computational resources, making it less scalable.	Ideal for projects with large datasets and a need for deep contextual understanding.
Transformer Models	State-of-the-art accuracy, especially FinBERT for financial text. Captures nuanced sentiments.	Computationally expensive and may need fine-tuning for domain-specific datasets.	Perfect for financial sentiment analysis in stocks or index funds with detailed textual data available.
Hybrid Models	Combines multiple strengths (e.g., sentiment with technical indicators) for rich insights.	Complex to implement, integrate, and requires significant computational resources.	Best for advanced projects needing multi-faceted analysis (e.g., sentiment + market trend analysis).

For my project, I've decided to use transformer models, and more specifically, FinBERT, for sentiment analysis. Transformer models are incredible at understanding the context and meaning behind words, which is especially important when analyzing complex financial texts. FinBERT is a standout choice because it has been trained on financial data like earnings reports and market news, which means it's designed to understand the specific language and nuances of the finance world. General sentiment analysis tools often miss the subtle meanings in financial contexts, but FinBERT excels at picking up on these.

By using FinBERT, I can ensure that the sentiment scores I generate are not only accurate but also highly relevant to the stocks and index funds I'm working with. This makes it the perfect fit for my project, especially since I'll be combining sentiment analysis with reinforcement learning. FinBERT's ability to process large amounts of text and provide high-quality insights will help me create more adaptive and informed trading strategies. Overall, it's the ideal tool for bringing the power of financial sentiment into my project.

Reinforcement Learning

- **Value-Based Methods**
  These methods focus on learning a value function that estimates the expected reward for each state or state-action pair. The policy is derived indirectly by selecting the action with the highest value.
    - **Common Algorithms**:
        - Q-Learning: Learns the value of taking an action in a given state (action-value function) without requiring a model of the environment.
        - Deep Q-Networks (DQN): Combines Q-learning with deep neural networks to handle high-dimensional state spaces.
        - Double DQN (DDQN): Addresses overestimation issues in DQN by decoupling action selection and evaluation.
    - **Pros**:
        - Efficient for discrete action spaces.
        - Well-suited for environments with clear state-action-reward dynamics.
    - **Cons**:
        - Struggles with continuous action spaces.
        - Requires discretization of actions if the space is not naturally discrete.

- **Policy-Based Methods**
  These methods directly optimize the policy (a mapping from states to actions) without explicitly learning a value function.
    - **Common Algorithms**:
        - Policy Gradient Methods: Use gradient ascent to optimize the policy by maximizing cumulative rewards.
        - REINFORCE: A simple policy gradient method that updates the policy after every episode.
    - **Pros**:
        - Works well for continuous action spaces.
        - Can learn stochastic policies, which are useful in uncertain environments.
    - **Cons**:
        - High variance in updates can lead to instability.
        - Requires careful tuning of hyperparameters like learning rate.
- **Actor-Critic Methods**
  These combine value-based and policy-based methods by maintaining both a policy function (actor) and a value function (critic). The critic evaluates the policy's actions, reducing variance in the policy updates.
    - **Common Algorithms**:
        - Advantage Actor-Critic (A2C): Simplifies computation by running multiple instances in parallel.
        - Proximal Policy Optimization (PPO): Improves stability and performance by clipping policy updates.
        - Deep Deterministic Policy Gradient (DDPG): Handles continuous action spaces by combining deterministic policy gradients with actor-critic architecture.
    - **Pros**:
        - Reduces instability of policy-based methods.
        - Balances exploration and exploitation effectively.
    - **Cons**:
        - More complex to implement than pure policy or value-based methods.
        - Computationally intensive due to maintaining two separate functions.
- **Model-Based Methods**
  These learn a model of the environment to simulate its dynamics, which can then be used for planning or to generate additional training data for the agent.
    - **Common Algorithms**:
        - Dyna-Q: Combines Q-learning with a learned model of the environment.
        - Model-Predictive Control (MPC): Uses the learned model to plan and predict optimal actions over a time horizon.
    - **Pros**:
        - Can be more sample-efficient since the model can be used to generate data.
        - Suitable for environments where simulating dynamics is computationally cheaper than real-world interactions.
    - **Cons**:
        - Performance heavily depends on the accuracy of the learned model.
        - Complex environments with high variability make modeling difficult.
- **Evolutionary Strategies and Hybrid Methods**
  These are non-gradient-based optimization methods that use evolutionary techniques to evolve policies.
    - **Common Algorithms**:
        - CMA-ES (Covariance Matrix Adaptation Evolution Strategy): Optimizes policies by evolving population-based solutions.
        - Hybrid Approaches: Combine RL with other methods, such as supervised learning or imitation learning.

- **Pros**:
  - Doesn't rely on gradients, so it's useful for environments with non-differentiable rewards.
  - Robust to noisy environments.
- **Cons**:
  - Often requires a large number of simulations or evaluations.
  - May converge more slowly than gradient-based methods.

Method	Key Idea	Best For	Challenges
Value Based	Learn a value function to derive the policy.	Discrete action spaces.	Poor performance in continuous action spaces.
Policy Based	Directly optimize the policy.	Continuous actions and stochastic environments.	High variance in updates; unstable training.
Actor Critic	Combine value and policy methods.	Balancing exploration and exploitation.	Computationally intensive; more complex to set up.
Model Based	Learn a model of the environment.	Sample efficiency and planning-based strategies.	Heavily reliant on accurate models.
Evolutionary / Hybrid	Use evolutionary algorithms or hybrid models.	Non-differentiable rewards and noisy environments.	Slow convergence and high computational demands.

Method	Key Features	Advantages	Challenges	Best Use Cases
Advantage Actor-Critic (A2C)	Uses advantage estimates (difference between action-value and state-value functions) to stabilize policy updates. Runs multiple environments in parallel to gather diverse experiences.	Efficient parallelization improves learning speed. Handles stochastic and continuous environments well. Balances exploration and exploitation using advantage functions.	Parallel environments increase computational overhead. Requires careful tuning of hyperparameters like learning rate and advantage estimation parameters.	Ideal for projects needing stability, efficiency, and adaptability in complex environments (e.g., stock trading with market volatility).
Proximal Policy Optimization (PPO)	Introduces a "clipping" mechanism to restrict policy updates within a trust region, improving stability. Combines the simplicity of policy gradient methods with the efficiency of A2C.	More stable than A2C due to constrained updates. Easy to implement and tune. Works well with both discrete and continuous action spaces.	Computationally more intensive than A2C due to additional constraints. Slightly more complex to implement and fine-tune.	Best for large-scale, high-stakes projects needing stability (e.g., long-term portfolio management or high-frequency trading).

Deep Deterministic Policy Gradient (DDPG)	Designed for continuous action spaces. Combines deterministic policy gradients with an actor-critic framework. Uses replay buffers and target networks to stabilize learning.	Excellent for environments with continuous actions. Learns deterministic policies, which can lead to faster convergence in low-noise settings. Handles complex action spaces better than A2C or PPO.	Sensitive to hyperparameters (e.g., learning rate, target network updates). Struggles in highly stochastic environments. Replay buffer adds memory and computation overhead.	Best for trading scenarios requiring precise, continuous actions, such as deciding optimal trade sizes or proportions.
Twin Delayed DDPG (TD3)	An improvement over DDPG that uses two critics to mitigate overestimation of value functions and introduces noise regularization for exploration.	Reduces overestimation bias, leading to better policy stability. Adds noise for more effective exploration. Retains benefits of DDPG for continuous action spaces.	Computationally expensive due to maintaining two critic networks. Requires more hyperparameter tuning than DDPG.	Ideal for highly complex environments with continuous actions and high volatility (e.g., derivative trading or options strategies).
Soft Actor-Critic (SAC)	Extends DDPG with entropy regularization, encouraging the policy to explore more effectively by maximizing both reward and entropy.	Encourages exploration, making it robust in stochastic or uncertain environments. Handles continuous actions with better stability than DDPG. Offers state-of-the-art performance in many benchmarks.	High computational cost. Requires careful balancing of entropy regularization and reward maximization.	Best for highly dynamic and uncertain environments, such as real-time market simulations or trading with uncertain sentiment data.

For my project, I believe the best method for implementing reinforcement learning is Advantage Actor-Critic (A2C). It offers a great balance of simplicity, efficiency, and adaptability, which makes it an ideal fit for combining sentiment analysis with trading strategies. A2C uses two components: the actor, which decides the actions, and the critic, which evaluates how good those actions are. This setup helps it learn more effectively by stabilizing the updates and ensuring better decision-making in dynamic environments like the stock market. One of the best features of A2C is its ability to run multiple simulations in parallel, which speeds up the training process and provides a diverse set of experiences for the model to learn from. This is particularly useful when dealing with complex data like sentiment scores and market indicators. While there are more advanced methods like PPO, A2C's efficiency and relatively lower computational requirements make it the most practical choice for my project. It's versatile enough to handle the complexities of financial data while being stable and resource-friendly.

Reward Function for ReinforcementLearning

- **Net Profit**
  - **Description**: A straightforward reward function that incentivizes the RL agent to maximize the overall portfolio value.
  - **Formula**: $R_t = P_t - P_{t-1}$

    Where:

    P_t: Portfolio value at time t

    P_{t-1}: Portfolio value at time $t - 1$.
  - **Pros**:
    - Simple to implement.

        - Focuses purely on profitability.
    - **Cons**:
        - Does not account for risk, volatility, or trading costs.
    - **Best Use Case**: Initial experiments or environments where transaction costs and risk are negligible.

- **Risk-Adjusted Reward (Sharpe Ratio)**
    - **Description**: Encourages high returns while minimizing volatility to manage risk effectively.
    - **Formula**: $R_t = \frac{E[R_p - R_f]}{\sigma_p}$

      Where:
      R_p: Portfolio return.
      R_f: Risk-free rate (e.g., government bond yield).
      σ_p: Standard deviation of portfolio returns.
    - **Pros**:
        - Accounts for both returns and risk.
        - Encourages stable strategies over time.
    - **Cons**:
        - Requires rolling window calculations for σp\sigma_p, adding computational complexity.
    - **Best Use Case**: Projects requiring risk-aware trading strategies.

- **Reward with Transaction Cost Penalty**
    - **Description**: Reflects real-world constraints by deducting costs associated with buying and selling.
    - **Formula**: $R_t = (P_t - P_{t-1}) - \alpha \cdot |Shares\ Traded|$

      Where:
      α : Cost per trade (e.g., percentage or flat rate).
    - **Pros**:
        - Captures real-world trading expenses.
        - Discourages excessive trading.
    - **Cons**:
        - Focuses heavily on minimizing trades, which might hinder exploration.
    - **Best Use Case**: Realistic market simulations where transaction costs are significant.

- **Profit and Drawdown Control**
    - **Description**: Penalizes strategies that experience large drawdowns, encouraging steady growth.
    - **Formula**: $R_t = (P_t - P_{t-1}) - \lambda \cdot Drawdown$

      Where:
      λ: Weight for penalizing drawdowns.
    - **Pros**:
        - Encourages strategies with steady growth.
        - Reduces the likelihood of large losses.
    - **Cons**:
        - Computationally intensive to calculate drawdown.
    - **Best Use Case**: Managing portfolios in volatile markets

- **Sentiment-Weighted Reward**
    - **Description**: Incorporates sentiment analysis by amplifying rewards for positive market sentiment and dampening them for negative sentiment.

- **Formula**: $R_t = (P_t - P_{t-1}) \cdot (1 + Sentiment\ Score)$

  Where:

  $Sentiment\ Score$: Sentiment value (e.g., between -1 and 1).
- **Pros**:
  - Integrates sentiment analysis directly into trading decisions.
  - Encourages the agent to align actions with market sentiment.
- **Cons**:
  - Highly dependent on the accuracy of sentiment scores.
- **Best Use Case**: Projects involving financial sentiment analysis as a key factor.

Reward with Combined Factors

- **Description**: Combines profitability, risk adjustment, transaction costs, and sentiment into a single reward function.
- **Formula**: $R_t = (P_t - P_{t-1}) - \alpha \cdot |Shares\ Traded| - \lambda \cdot Drawdown] \cdot (1 + Sentiment\ Score)$

  Where:

  α: Cost per trade.

  λ: Weight for drawdown penalty.
- **Pros**:
  - Balances multiple real-world constraints.
  - Captures both market trends and sentiment.
- **Cons**:
  - More complex to implement and tune.
- **Best Use Case**: Advanced trading systems requiring robust performance across multiple dimensions.

Risk-Seeking Reward

- **Description**: Rewards the agent for achieving high returns, even if it involves higher risk-taking.
- **Formula**: $R_t = (P_t - P_{t-1}) + \beta \cdot Volatility$

  Where:

  β: Weight for encouraging risk-taking.
- **Pros**:
  - Encourages bold strategies for high returns.
  - Useful in exploratory phases.
- **Cons**:
  - May lead to overly aggressive strategies.
- **Best Use Case**: Scenarios where risk tolerance is high, such as testing volatile markets.

Volatility-Aware Reward

- **Description**: Penalizes excessive volatility to encourage steadier returns.
- **Formula**: $R_t = (P_t - P_{t-1}) - \gamma \cdot Volatility$

  Where:

  γ: Weight for penalizing volatility.
- **Pros**:
  - Favors consistent, low-risk strategies.
  - Useful for conservative investors.
- **Cons**:
  - May discourage taking advantage of volatile, profitable opportunities.
- **Best Use Case**: Long-term portfolio management with low-risk tolerance.