**Trading Bot 34-01 - Project Document**Project: 8-25-2024\_MM01  
Project Manager: Michael Masters  
Client: TradeFloor Trader034

**Trading Bot Documentation**

**Overview**

This trading bot uses a reinforcement learning model (Proximal Policy Optimization - PPO) to make trading decisions based on real-time stock data. The bot logs its actions, along with relevant market data and predictions, to a MySQL database for further analysis.

**Prerequisites**

**Software Requirements**

* Python 3.x
* MySQL Server
* Required Python packages:
  + yfinance
  + numpy
  + torch
  + mysql-connector-python
  + gym

**Hardware Requirements**

* A machine capable of running Python scripts and MySQL server.

**Installation**

1. **Install Python and MySQL:**
   * Install Python from [python.org](https://www.python.org/).
   * Install MySQL from [mysql.com](https://www.mysql.com/).
2. **Install Python Packages:**

1. pip install yfinance numpy torch mysql-connector-python gym

1. **Set up MySQL Database:**
   * Create a MySQL database named trading\_bot.
   * Create a table named action\_logs with the following schema:

sql

1. CREATE TABLE action\_logs (

2. id INT AUTO\_INCREMENT PRIMARY KEY,

3. episode INT,

4. step INT,

5. state TEXT,

6. action INT,

7. log\_prob FLOAT,

8. reward FLOAT,

9. open\_price FLOAT,

10. close\_price FLOAT,

11. estimated\_price\_5min FLOAT,

12. timestamp DATETIME

13. );

**Configuration**

**Database Configuration**

Modify the database connection details in main.py:

python

1. db\_config = {

2. 'user': 'your\_username',

3. 'password': 'your\_password',

4. 'host': 'localhost',

5. 'database': 'trading\_bot'

6. }

**Trading Parameters**

Set the ticker symbol and window size for the trading environment:

python

8. ticker = 'AAPL' # Example ticker

9. window\_size = 30 # Example window size for the state representation

**Code Structure**

**Main Components**

1. **main.py**
   * The main script that runs the trading bot, fetches real-time data, makes trading decisions, and logs actions to MySQL.
2. **models/ppo\_agent.py**
   * Defines the PPO agent and its methods for selecting actions and optimizing the model.
3. **env/trading\_env.py**
   * Defines the trading environment, including methods for resetting the environment, taking steps, and calculating rewards.

**Logging Function**

python

10. def log\_action\_to\_mysql(episode, step, state, action, log\_prob, reward, open\_price, close\_price, estimated\_price\_5min):

Logs the details of each action taken by the agent into a MySQL database.

Parameters:

- episode: The current episode number.

- step: The current step number within the episode.

- state: The current state representation.

- action: The action taken by the agent.

- log\_prob: The log probability of the action.

- reward: The reward received for the action.

- open\_price: The open price of the stock at the time of the action.

- close\_price: The close price of the stock at the time of the action.

- estimated\_price\_5min: The estimated stock price 5 minutes after the action.

11. cursor.execute(

12. """

13. INSERT INTO action\_logs (episode, step, state, action, log\_prob, reward, open\_price, close\_price, estimated\_price\_5min, timestamp)

14. VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s)

15. """,

16. (episode, step, str(state.tolist()), action, float(log\_prob), float(reward), open\_price, close\_price, estimated\_price\_5min, datetime.now())

17. )

18. conn.commit()

**Main Trading Loop**

python

Copy code

19. for episode in range(total\_episodes):

20. state = env.reset()

21. done = False

22. step = 0

23. total\_reward = 0

24.

25. while not done:

26. # Fetch real-time data from yfinance (adjust as needed)

27. data = yf.download(ticker, period='1d', interval='1m')

28. current\_data = data.iloc[-1] # Use the most recent data

29. open\_price = current\_data['Open']

30. close\_price = current\_data['Close']

31. state = np.array(current\_data[['Open', 'High', 'Low', 'Close', 'Volume']].values)

32.

33. # Make a decision based on the current state

34. action, log\_prob = agent.select\_action(state)

35.

36. # Estimate the price in 5 minutes (this is hypothetical and should be based on your model's prediction mechanism)

37. estimated\_price\_5min = close\_price \* 1.01 # Placeholder for the estimated price

38.

39. # Apply the action to the environment and get the next state and reward

40. next\_state, reward, done, \_ = env.step(action)

41. total\_reward += reward

42.

43. # Log the action and state to MySQL

44. log\_action\_to\_mysql(episode, step, state, action, log\_prob, reward, open\_price, close\_price, estimated\_price\_5min)

45.

46. # Move to the next state

47. state = next\_state

48. step += 1

49.

50. print(f"End of episode {episode}, total reward: {total\_reward}")

**Usage**

1. **Run the Script:**
   * Execute main.py to start the trading bot.

python main.py

1. **Monitor Logs:**
   * Check the action\_logs table in your MySQL database to monitor the bot's actions and decisions.
2. **Analyze Results:**
   * Use the logged data to analyze the performance of the trading bot and make necessary adjustments to improve its performance.

**Further Enhancements**

* **Model Improvements:** Enhance the PPO model with additional features or more sophisticated architectures.
* **Data Handling:** Improve data fetching and preprocessing to handle various market conditions and anomalies.
* **Risk Management:** Integrate risk management strategies to minimize potential losses.
* **Backtesting:** Implement backtesting capabilities to evaluate the bot's performance on historical data before deploying it in a live environment.

By following this documentation, you should be able to set up, run, and monitor the trading bot, as well as make enhancements based on the logged data and analysis.