1.TITLE AND GROUP DETAILS:

Title: ALGORITHMIC TRADING USING REINFORCEMEMENT LEARNING
Group ID:25SM03

Team Members	Roll numbers	Roles and Responsibilities
Sakshi Saxena	2301CS45	Collected and preprocessed financial data using Yahoo Finance API; implemented the reward function and experience replay logic for training; wrote the *Introduction* and *Related Work* sections of the report.
Ishika Gupta	2301AI10	Designed the DQN model architecture and implemented the training loop (forward, backward propagation, and optimization); contributed to the *Methodology* section, including *Dataset Description* and *Block Diagram*.
Shiwani Shrivastava	2301CS67	Developed the *Mathematical Model* and *Algorithm* sections, defining key equations, Bellman updates, and pseudocode.
Manshi Prajapati	2302CS08	Conducted *testing and evaluation*, analyzed performance metrics and results, and compiled the final *References* and report formatting.

2.INTRODUCTION:

Overview of the problem domain:
The financial market is dynamic and complex, with prices affected by economic, political, and psychological factors. Traditional strategies relying on fixed rules or human judgment often fail to adapt in real time. Algorithmic trading overcomes this by using computer programs to analyze data, identify opportunities, and execute trades automatically, combining finance, data science, and AI to make informed decisions in volatile markets.

Importance and Relevance of the Topic**:**
Algorithmic Trading using Reinforcement Learning (RL) integrates AI with automated decision-making to improve trading efficiency and adaptability. Unlike traditional models, RL agents learn optimal strategies from market interactions, uncover hidden patterns, and balance risk–reward—driving smarter FinTech systems.

Project Objectives and Scope:
This project develops and compares Deep Q-Network (DQN) and Proximal Policy Optimization (PPO) for automated trading to maximize returns and minimize risk.
Objectives:

- Implement and evaluate DQN and PPO on real financial data.
- Measure performance using cumulative return, Sharpe ratio, and drawdown.
- Assess stability, adaptability, and generalization for real-time trading.

Scope:

- Develop an RL-based trading agent (DQN) that interacts with historical stock data to learn optimal Buy, Sell, or Hold strategies in a simulated environment.
- Measure the agent's effectiveness using key financial metrics such as cumulative return, Sharpe ratio, and drawdown, and compare its decisions across multiple tickers.
- Provide a Python-based framework that can be extended to new tickers, indicators, or alternative RL models (like PPO) for future experimentation in algorithmic trading.

3.RELATED WORK:

-Research Papers:-

a) Financial Trading as a Game: A Deep Reinforcement Learning Approach
The paper by Chien Yi Huang (2018) presents a Deep Recurrent Q-Network (DRQN) framework for financial trading, modeling it as a Markov Decision Process (MDP). It introduces a small replay memory, an action augmentation technique for continuous greedy policy training, and longer RNN training sequences to improve efficiency.

b) Improving financial trading decisions using deep Q-learning
The paper by G. Jeong (2019) applies deep Q-learning to enhance trading strategies. It introduces three methods to maximize profits and handle market volatility, showing through experiments that these approaches outperform traditional strategies in profitability and adaptability.

c) Algorithmic Trading and Short-term Forecast for Financial Time Series with Machine Learning Models: State of the Art and Perspectives
The paper reviews ML and DL methods for short-term financial forecasting, including LSTM, CNN, and ensemble models using technical and sentiment data. LSTMs often perform best, but accuracy is moderate due to noisy, non-stationary data. Hybrid models and standardized datasets, explainable AI, and reinforcement learning are highlighted as promising directions for future research.

- Previously used Algorithms and Approaches:-

Before Deep Q-Networks (DQN), algorithmic trading relied on models like Linear Regression, ARIMA, and SVM, which worked for short-term trends but couldn't adapt to dynamic markets. Early reinforcement learning methods such as Q-Learning and SARSA introduced adaptability but were limited to small, discrete state spaces. Policy Gradient and Actor-Critic methods improved stability yet relied on handcrafted features and struggled with non-linear patterns. DQN overcame these challenges by combining deep neural networks with reinforcement learning, enabling agents to learn directly from raw market data, capture complex patterns, and make adaptive trading decisions-marking a major advancement in intelligent, scalable algorithmic trading.

4.METHODOLOGY:

1. Dataset Description:-

The dataset includes historical stock market data for multiple companies, with 602,963 rows representing daily trading information such as date, open, high, low, close, volume, dividends, stock splits, and company symbols. Preprocessing involved handling missing values, Min-Max

normalization, and feature engineering with indicators like RSI and MACD. The data was time-ordered per company and split 70-30 for training and testing, providing a rich input for evaluating DQN and PPO trading agents.

2. Block Diagram or Workflow:-

3. Proposed Model:-

We use a Deep Q-Learning (DQN) framework with a custom Gymnasium trading environment to learn an optimal policy (Buy, Sell, Hold) from recent OHLCV data. The neural network outputs Q-values for each action, and the agent maximizes cumulative reward (net profit) using Q-learning updates with an ε-greedy strategy.

Justification: DQN suits discrete action spaces and can generalize across stocks by learning patterns like momentum and volatility. Unlike supervised models, RL treats trading as a sequential decision problem with delayed rewards, enabling the agent to optimize entry/exit timing, holding periods, and risk management holistically.

4.Mathematical Model:-

The model uses a Deep Q-Network (DQN) in a Reinforcement Learning (RL) setting, where the trading agent observes the market state (s_t), takes an action (a_t: Buy, Sell, or Hold), and receives a reward (r_t). The goal is to learn an optimal policy that maximizes cumulative discounted rewards.

1. Objective Function: Maximize $Q^*(s, a) = \max E[\Sigma (\gamma^t * r_t)]$, where γ is the discount factor controlling future reward importance.
2. Q-Learning Update: $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_t + \gamma * \max Q(s_{t+1}, a') - Q(s_t, a_t)]$, where α is the learning rate.

3. Neural Network Approximation: A deep neural network approximates $Q(s, a; \theta)$, minimizing $L(\theta) = (y_t - Q(s_t, a_t; \theta))^2$, where $y_t = r_t + \gamma * \max Q(s_{t+1}, a_{t+1}; \theta^-)$.
4. Reward Function: $r_t = (P_{t+1} - P_t) / P_t$, encouraging profitable actions.
5. Evaluation Metrics: Performance is measured using Total Return, Sharpe Ratio, Win Rate, Maximum Drawdown, and Cumulative Profit.
This formulation enables the DQN agent to learn profitable and risk-aware trading strategies.

Algorithm 1: DQN-based Multi-Ticker Trading Framework

Require:
Tickers list $T = \{t_1, t_2, ..., t_k\}$, date range [s, e], window size W, embedding size E, replay buffer capacity C, batch size B, learning rate α, discount factor γ, target update interval U, total training steps S, epsilon parameters ε_start, ε_end, τ

Ensure:
Trained policy network Q_θ

1. Download OHLCV data for all tickers $t \in T$ over period [s, e].
2. For each ticker t, compute derived features:
   $x_t = \{Close, ret, log\_ret, ma5, ma10, std5, vol, rsi\}$.
3. Fit a StandardScaler on each ticker's feature matrix and store the scalers.
4. Initialize policy network Q_θ with company embedding of size E; copy weights to target network $Q\_\theta' \leftarrow Q\_\theta$.
5. Initialize optimizer (Adam) and replay buffer D with capacity C.
6. Prefill replay buffer with random policy until $|D| \geq P$: randomly select ticker t_i, simulate environment, and store transitions (s, a, r, s′, done) into D.
7. Set step counter steps_done = 0.
8. for step = 1 to S do
   11. Compute exploration rate:
   $\varepsilon = \varepsilon\_end + (\varepsilon\_start - \varepsilon\_end) \times e^{(-steps\_done / \tau)}$
   12. Sample random ticker t_i, build environment using its features and scaler.
   13. Select action a using ε-greedy policy:
   with probability $(1 - \varepsilon)$, choose $a = \text{argmax}_{a'} Q\_\theta(s, t_i)$; otherwise, select a random action from {0, 1, 2}.
   14. Execute action a in the environment and observe (s′, r, done).
   15. Store transition (s, a, r, s′, done) in replay buffer D.
   16. If $|D| \geq B$, sample a minibatch of size B:
   – Compute current Q-values: $Q\_\theta(s, a)$
   – Compute target values:
   $y = r + \gamma \times \max_{a'} Q\_\theta'(s', t_i)$ for non-terminal transitions; else $y = r$
   – Minimize loss:
   $L = (1/B) \Sigma (Q\_\theta(s, a) - y)^2$
   – Perform backpropagation and optimizer update.
   17. Every U steps, update target network: $Q\_\theta' \leftarrow Q\_\theta$.
   18. Increment steps_done.
9. end for
10. Save trained model parameters and scalers to *dqn_trading_checkpoint.pth*.
11. return trained policy network Q_θ

5.REFERENCES:-

1. C.-Y. Huang, "Financial trading as a game: A deep reinforcement learning approach," *arXiv preprint arXiv:1807.02787*, 2018. [Online]. Available: https://arxiv.org/pdf/1807.02787

2. G. Jeong and H. Y. Kim, "Improving financial trading decisions using deep Q-learning: Predicting the number of shares, action strategies, and transfer learning," *Expert Systems with Applications*, vol. 133, pp. 1–11, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417418306134

3. T.-V. Pricope, "Deep reinforcement learning in quantitative algorithmic trading: A review," *IEEE Access*, vol. 7, pp. 108014–108022, 2019. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8786132