Stock Price Prediction using Double DQN – Team Report

# Team Members: -

BT21CSE166 – Harsh Verma

BT21CSE167 – Daulat Ojha

BT21CSE193 – Vasu Kapasiya

BT21CSE201 – Mudit Loya

BT21CSE202 – Yash Patidhar

# Team Member Contributions

## Harsh Verma – Core Double DQN Implementation and Training

Harsh led the core architecture and model development. He implemented both the DQN (neural network) class and the DQNAgent class, handling essential functions such as experience replay, epsilon-greedy strategy, and target network updates. The DQN model had 3 fully connected layers:  
- Input layer corresponding to the state size  
- Two hidden layers with 64 and 32 neurons respectively, using ReLU activation  
- Output layer with size equal to action space (3: Buy, Hold, Sell)  
He orchestrated the training pipeline:  
- Iteration 1: Trained the model over 10 episodes; initial results were unsatisfactory (e.g., only 4 trades over 3 years).  
- Iteration 2: After feedback from Mudit, he tuned hyperparameters (learning rate, gamma, epsilon decay) and extended training to 50 episodes.  
Harsh also coordinated workflow alignment between members due to the centrality of this module.

## Daulat Ojha – Trading Environment Design

Daulat developed the StockTradingEnv class which defines the agent’s interaction with the environment. He encoded the logic for:  
- State representation: Combining current stock price, technical indicators, portfolio state, and cash balance.  
- Action space: [0 - Hold, 1 - Buy, 2 - Sell].  
- Reward function: Based on net profit/loss, penalizing idle behavior, and encouraging successful trades.  
- Done condition: If all stock data is processed or cash balance becomes negative.  
This environment acts like the simulated market where the agent observes, acts, and learns.

## Mudit Loya – Model Evaluation and Validation

Mudit was responsible for testing and validating the model using:  
- Cumulative moving average of rewards  
- Total return, daily return, and Sharpe ratio  
- Trade frequency and net gains  
He also validated Harsh’s training iterations by identifying underfitting and advised retraining with tuned parameters.

## Vasu Kapasiya – Data Preparation and Feature Engineering

Vasu gathered historical stock data using the yfinance library and cleaned it for missing values. He also computed technical indicators using the `ta` library:  
- Bollinger Bands  
- Moving Averages (SMA/EMA)  
- RSI and MACD  
The processed data became the input features for the agent’s environment.

## Yash Patidhar – Visualization and Result Representation

Yash developed the plotting utilities and `Visualizer` class to show model performance. He plotted:  
- Buy/Sell points on stock price timeline  
- Equity curve and reward trends  
- Overlays of indicators with agent decisions  
These visuals helped make the model behavior interpretable and presentation-ready.

# End-to-End Process Flow

## 1. Data Gathering

Historical stock data (e.g., TSLA) was downloaded using the yfinance API with OHLCV format.

## 2. Data Cleaning and Feature Engineering

Handled missing values using forward-fill and generated technical indicators using `ta` library:  
- Bollinger Bands, MACD, RSI  
- SMA, EMA, Volatility indicators  
Final dataset was scaled and formatted for environment ingestion.

## 3. Environment Design (Trading Simulator)

The `StockTradingEnv` acts as a custom OpenAI Gym environment. It simulates a market environment with the following logic:  
- Observation (state): vector of price window, indicators, portfolio status.  
- Action Space: Buy (1), Sell (2), Hold (0).  
- Rewards: Computed as net profit/loss per trade, encouraging profitable trading and penalizing bad moves.  
- Terminal Condition: End of data or depleted capital.

## 4. Double DQN Architecture and Intuition

Double DQN solves the overestimation bias in standard DQN by decoupling action selection and evaluation:  
- The agent uses one network (online) to choose an action.  
- A second network (target) is used to evaluate the value of that action.  
Model Details:  
- Input Layer: state vector  
- Two Hidden Layers: 64 and 32 neurons, ReLU activation  
- Output Layer: Q-values for each action  
- Optimizer: Adam | Loss: MSE | Epsilon Decay for exploration

## 5. Model Training Process

Harsh trained the model in episodes, each simulating full stock price history.  
- Initial run: 10 episodes, poor trade frequency  
- Post-tuning: 50 episodes, improved decision-making  
- Replay memory was used to stabilize learning and reduce correlation.  
- Target network was updated every few steps to align with Double DQN logic.

## 6. Validation and Evaluation Metrics

Mudit validated the model via:  
- Cumulative profits and Sharpe ratio  
- Trade accuracy and reward trajectory  
- Buy-and-hold vs Agent strategy performance comparison

## 7. Visualization and Interpretation

Yash visualized:  
- Trade markers on price charts  
- Reward trends per episode  
- Final equity curve to summarize gains/losses visually