The DRL trading system combines:

* **LSTM Price Predictor**: Pre-trained model forecasting next day's opening price
* **Reinforcement Learning Agent**: PPO algorithm making trading decisions
* **Custom Trading Environment**: Simulates market interactions with realistic constraints

**2. Core Components**

**A. PortfolioTradingEnv (Custom Gym Environment)**

**Observation Space**:

* **Market Features**: Technical indicators (MACD, Bollinger Bands, etc.) + Sentiment scores
* **LSTM Prediction**: Next day's predicted open price
* **Portfolio State**: Current cash balance and position size

**Action Space** (Continuous):

* action[0]: Trade direction [-1 (sell) to 1 (buy)]
* action[1]: Position size [0 (none) to 1 (100% of available funds)]

**Reward Function**:

python

Copy

Sharpe Ratio = (Mean of Portfolio Returns) / (Standard Deviation of Returns)

* Encourages risk-adjusted returns
* Penalizes volatile strategies

**Key Methods**:

* reset(): Initializes new trading episode
* step(): Executes trades and updates portfolio
* \_get\_observation(): Constructs state vector with market data + predictions

**B. PPO Agent Configuration**

python

Copy

model = PPO(

policy="MlpPolicy",

env=env,

learning\_rate=3e-4,

n\_steps=2048,

batch\_size=128,

n\_epochs=10,

gamma=0.99,

gae\_lambda=0.95,

clip\_range=0.2

)

* **Policy Network**: 2-layer MLP (256 units each)
* **Training Parameters**:
  + Discount factor (γ=0.99): Values future rewards
  + GAE (λ=0.95): Reduces variance in policy updates
  + Clip range (0.2): Prevents drastic policy changes

**3. Workflow Sequence**

1. **Initialization**:
   * Load pre-trained LSTM model
   * Create trading environment with historical data
   * Initialize PPO agent
2. **Training Phase**:

python

Copy

for episode in range(total\_timesteps):

obs = env.reset()

while not done:

# 1. Get action from policy

action, \_ = model.predict(obs)

# 2. Execute trade

next\_obs, reward, done, info = env.step(action)

# 3. Store experience

replay\_buffer.add(obs, action, reward, next\_obs, done)

# 4. Periodic policy updates

if time\_to\_update:

model.learn(replay\_buffer)

1. **Trading Logic**:
   * For each timestep:
     + Generate LSTM prediction
     + Construct observation vector
     + Agent decides trade (direction + size)
     + Environment executes order with 0.1% transaction cost
     + Portfolio value updates
2. **Evaluation Metrics**:
   * **Portfolio Value**: Total assets (cash + positions)
   * **Sharpe Ratio**: Risk-adjusted performance
   * **Max Drawdown**: Worst peak-to-trough decline

**4. Key Financial Mechanisms**

**Position Management**:

python

Copy

if action[0] > 0: # Buy

shares = (action[1] \* balance) / current\_price

balance -= (shares \* current\_price \* 1.001) # Include 0.1% fee

elif action[0] < 0: # Sell

shares = min(action[1] \* total\_shares, total\_shares)

balance += (shares \* current\_price \* 0.999) # Deduct 0.1% fee

**Risk Controls**:

* Automatic position sizing
* Transaction cost modeling
* Portfolio value tracking

**5. Integration with LSTM**

1. **Prediction Incorporation**:

python

Copy

lstm\_input = window\_data.reshape(1, window\_size, -1)

predicted\_price = lstm\_model.predict(lstm\_input)[0][0]

1. **State Enhancement**:
   * Original features + LSTM prediction + portfolio state
   * Provides complete market context to agent