Learning to Trade: Reinforcement Learning versus Rule-Based Strategies on Long-Horizon High-Frequency Financial Data

Chirag Mahaveer Chivate

Abstract

This project investigates whether a reinforcement learning (RL) agent can outperform traditional technical analysis-based strategies on long-horizon, high-frequency financial data. Using 20 years of minute-level stock data from Apple, Reliance, and the S&P 500 Index, we construct a custom OpenAI Gym-style trading environment and evaluate both rule-based strategies and deep Q-networks (DQN) with LSTM memory. The agent observes technical indicators and makes discrete trading decisions — Buy, Hold, or Sell — with transaction fees and delayed feedback. Hardcoded strategies are built using trend, momentum, mean-reversion, and volume indicators, tested under various stop-loss settings. In contrast, the DQN agent is trained end-to-end using sequential state-action-reward data. Empirical results show the RL agent consistently outperforms hardcoded strategies in both absolute ROI and robustness, while generalizing well across multiple financial instruments. The project also discusses key challenges such as partial observability, delayed rewards, data efficiency, and the limitations of 1-step TD learning in complex domains like trading.

Keywords: Reinforcement Learning, DQN, Trading Agent, Technical Indicators, LSTM, Financial Markets, Rule-Based Strategies, High-Frequency Data, Deep Q-Learning, Stochastic Policies, Generalization, Market Efficiency

1 Introduction

The rise of algorithmic trading has dramatically transformed financial markets, with increasingly sophisticated strategies automating decision-making at ever-higher frequencies. Traditionally, rule-based systems relying on technical indicators like moving averages or momentum oscillators have dominated retail and institutional strategies alike. However, such systems are limited in flexibility, often overfitted to specific market conditions, and incapable of adapting to changing environments without manual tuning.

This project explores whether reinforcement learning (RL)—a paradigm that enables agents to learn optimal actions through interaction with an environment—can outperform static indicator-based trading strategies and the classical buy-and-hold approach. Specifically, we evaluate RL agents trained on 20 years of minute-by-minute data from three distinct financial instruments: Apple Inc. (AAPL), the S&P 500 Index, and Reliance Industries (RELIANCE.NS).

We test two reinforcement learning algorithms: Proximal Policy Optimization (PPO), a widely-used policy-gradient method, and a custom Deep Q-Network (DQN) enhanced with Long Short-Term Memory (LSTM) units to model temporal dependencies in price movements.

Our evaluation is twofold: we assess raw profitability and generalization ability across assets, comparing the RL agents to (1) a naive buy-and-hold baseline and (2) eight hardcoded strategies based on well-known technical indicators and their combinations:

- Trend: EMA Crossover (50-200)
- Mean Reversion: Bollinger Bands
- Momentum: Stochastics, MACD
- Volume: On-Balance Volume (OBV)
- Hybrid 2-way: EMA + MACD
- Hybrid 2-way: EMA + Bollinger Bands
- Hybrid 3-way: EMA + MACD + OBV

Each strategy is evaluated with and without stop-losses, and all are tested on the exact same unseen data as the reinforcement learning agents to ensure fair comparison. Ultimately, the project seeks to understand if learning-based methods can outperform fixed-rule approaches—and whether they generalize across diverse financial instruments.

2 Problem Formulation

This project casts the problem of financial trading as a Reinforcement Learning (RL) task, where an agent interacts with a custom environment to learn an optimal trading policy. The environment simulates a simplified trading scenario with the following key components:

Action Space

The agent selects from a discrete set of three possible actions:

- **Hold** ($a = 0$): Take no action.
- **Buy** ($a = 1$): Buy $100 worth of the asset (adjusted for transaction fees).
- **Sell All** ($a = 2$): Sell all currently held shares.

Observation Space

At each timestep, the agent receives an observation vector consisting of 11 features:

- 10 normalized technical indicators: `EMA_50`, `EMA_200`, `SMA_20`, `Upper_Band`, `Lower_Band`, `%K`, `%D`, `MACD_Line`, `Signal_Line`, `OBV`.
- Number of shares currently held.

Reward Function

The agent receives a reward only when it performs a **Sell** action:

$$r_t = \text{net proceeds from sale} - \text{total invested in position}$$

All other actions yield zero reward. At the end of each episode, any unsold shares are automatically liquidated.

Objective

The agent's goal is to learn a policy that maximizes cumulative realized profit across each episode. The trading horizon spans thousands of timesteps from minute-level financial data, making this a long-horizon, sparse-reward problem.

Transaction Fees

Each Buy or Sell action incurs a 0.25% transaction fee, mimicking real-world slippage and friction. These fees are deducted from the investment or sale proceeds accordingly.

3 Dataset and Preprocessing

We used minute-level price data from three distinct financial assets to train and evaluate our trading agents:

- **Apple Inc. (AAPL)** – A major US technology stock.
- **S&P 500 Index** – A broad market index used as a benchmark.
- **Reliance Industries** – A large-cap Indian stock with similar size and structure to Apple's dataset.

Each dataset consists of 20 years of minute-by-minute trading data, originally containing standard price metrics: `Open`, `High`, `Low`, `Close`, `Adj Close`, `Volume`, and `Date`. After removing null rows and converting timestamps, we engineered the following technical indicators:

- **Trend Indicators:** EMA_{50}, EMA_{200}, SMA_{20}
- **Mean Reversion:** Upper and Lower Bollinger Bands
- **Momentum Indicators:** Stochastic Oscillator (%K, %D), MACD Line, Signal Line
- **Volume-Based:** On-Balance Volume (OBV)

This produced a total of 10 technical features. We appended an additional feature for the number of `shares_held` by the agent, creating an 11-dimensional observation space for each timestep.

Normalization: All features were scaled between 0 and 1 using `MinMaxScaler`.

Data Splitting: We split each dataset based on the `Year` column:

- **Training Set:** All data before 2016
- **Test Set:** All data from 2016 onwards

Dataset Sizes:

- Apple: ˜8.6 million rows
- S&P 500: ˜2.1 million rows (smaller coverage)
- Reliance: Nearly identical in size and time range as Apple

These processed datasets were used to evaluate both the rule-based strategies and reinforcement learning agents under consistent conditions.

4 Methods / Algorithms

This project explored two core reinforcement learning methods for financial trading: **Proximal Policy Optimization (PPO)** and **Deep Q-Networks (DQN) with LSTM**. Each method was evaluated against a custom trading environment using minute-level stock data and a fixed action space of {Hold, Buy($100), Sell All}.

4.1 PPO (Policy Gradient)

PPO was implemented as an actor-critic method using discrete categorical actions. The agent received a normalized observation vector of 11 features: 10 technical indicators (EMA, SMA, Bollinger Bands, Stochastic Oscillator, MACD, OBV) and `shares_held`.

While multiple PPO variations were explored—including clipped surrogate objectives, entropy regularization, and extended training windows—the agent consistently failed to make trades, even after many episodes. Extensive tuning (batch size, gamma, entropy coefficient, learning rate) did not resolve this. PPO was ultimately ruled out due to its inability to effectively learn in this sparse-reward trading setup.

Why Might PPO Be Failing Here? After analyzing the results and environment characteristics, several potential causes emerged:

1. **Sparse and Delayed Rewards**: PPO typically performs well in environments with frequent rewards. In this case, rewards are sparse—only occurring during `Sell` actions—and are often negative during early training, which discourages exploration.

2. **On-Policy Limitation**: PPO learns solely from data gathered in the current policy rollout. Unlike off-policy algorithms (like DQN), it cannot leverage successful past experiences, making recovery from early suboptimal policies much harder.

3. **Flat Exploration Dynamics**: The lack of early positive rewards leads PPO to collapse into a conservative `Hold`-only policy, as it learns that most actions incur fees or losses without seeing potential upsides.

4. **Sensitivity to Action Granularity**: In this discrete financial setting—with hard action boundaries and transaction costs—PPO struggles to estimate useful gradients, making it poorly suited for optimizing precise, event-triggered rewards.

4.2 DQN + LSTM

The primary agent used was a **Deep Q-Network with an LSTM core**, enabling it to learn temporal dependencies over sequences of technical indicator values. This design allowed the agent to recognize short-term patterns or trends across time that static feedforward networks might miss.

Architecture: The Q-network consisted of:

- An LSTM layer with hidden size 64, which maintains internal memory over 8 timesteps,
- A fully connected linear head projecting the LSTM output to 3 Q-values, corresponding to the three discrete actions: `Hold`, `Buy ($100)`, and `Sell All`.

Why LSTM? The inclusion of an LSTM allows the agent to capture short-term temporal dependencies in stock movement — for example, understanding that a sharp rise in indicators over several minutes might suggest a temporary overbought condition. Standard DQNs, lacking memory, cannot account for this sequential information.

Exploration and Epsilon Decay Strategy: The agent follows an ϵ-greedy exploration strategy:

- Initially, $\epsilon = 1.0$ (fully random actions),
- Linearly decayed over millions of steps to $\epsilon = 0.1$,
- This encourages exploration early and stable exploitation later.

Hyperparameters and Their Roles:

- **Input Dimension (11)**: 10 technical indicators + 1 feature for current `shares_held`.
- **Hidden Size (64)**: Size of the LSTM hidden state. Higher values allow more expressive modeling but increase training complexity.
- **Batch Size (8)**: Number of sequences sampled per training step.
- **Sequence Length (8)**: Each sampled sequence spans 8 consecutive timesteps — enough to model short-term trends.
- **Learning Rate (1e-3)**: Controls step size during gradient descent. Balanced for stable updates without divergence.
- **Gamma (0.99)**: Discount factor prioritizing long-term reward while still valuing immediate profits.
- **Replay Buffer Size (100,000)**: Stores past experiences to enable stable off-policy learning.

- **Epsilon Decay Steps (e.g. 4.2M)**: Length of decay period for exploration — typically set to roughly match the number of training steps.
- **Target Update Frequency (1000)**: Frequency (in steps) to sync weights from the online Q-network to the target network.

ReplayBuffer: A circular buffer that stores transitions of the form $(s, a, r, s', done)$. During training, random sequences of 8 steps are sampled to break temporal correlations and stabilize learning.

Action Selection (select_action): At each timestep:

- With probability ϵ, a random action is chosen,
- Otherwise, the Q-network predicts Q-values, and the action with the highest value is selected,
- LSTM hidden states are carried over across timesteps to maintain temporal context.

Training (train_one_episode): Each training episode consists of:

- Rolling through the environment from start to end,
- Storing each transition in the replay buffer,
- Performing updates after each step using batches of sequences,
- Decaying ϵ for the exploration-exploitation trade-off.

Evaluation (evaluate_agent): During evaluation:

- ϵ is set to 0.0 to enforce greedy (fully exploitative) behavior,
- The environment runs from start to finish with no exploration noise,
- The agent's performance is measured in terms of final realized profit and total investment made,
- The environment automatically liquidates positions at the end if any are still held.

Environment Details: The `TradingEnv`:

- Applies a 0.25% transaction fee on all Buy/Sell actions,
- Tracks realized profit on selling only (no reward on holding or buying),
- Forces final liquidation of shares at episode end,
- Includes total buys and cumulative profit as tracked metrics.

5 Baseline Strategies

To rigorously evaluate the RL agents' performance, we compared them against a variety of baseline strategies. These included both passive and rule-based approaches commonly used in algorithmic trading.

5.1 Buy-and-Hold

The most basic benchmark is a buy-and-hold strategy: purchase at the start of the test window and never sell. It captures the underlying market trend and sets a strong baseline for assessing if the RL agents can outperform simple passive investing.

5.2 Hardcoded Technical Strategies

We implemented and tested 9 hardcoded rule-based strategies derived from classical technical analysis:

- **Trend Following (EMA Crossover)**: Buy when short-term EMA crosses above long-term EMA.
- **Mean Reversion (Bollinger Bands)**: Buy below lower band, sell above upper band.
- **Relative Strength (RSI)**: Buy when RSI is below 30, sell when above 70.
- **Momentum (MACD)**: Buy when MACD crosses above signal line.
- **Volume (OBV)**: Buy when OBV rises significantly.
- **EMA + MACD**: Combines trend and momentum indicators.
- **EMA + Bollinger Bands**: Combines trend and mean-reversion signals.
- **EMA + MACD + OBV**: A three-indicator fusion strategy.
- **Stop Loss Variants**: Each of the above was tested with stop-loss thresholds (e.g., 0.03%, 1%, 3%, ..., 20%).

5.3 Evaluation Protocol

Each baseline strategy was evaluated on the **exact same test sets** as the RL agents and buy-and-hold, ensuring a fair head-to-head comparison. Strategies were tested on both S&P 500 and Reliance datasets.

All 9 strategies were evaluated with and without stop-loss triggers and on overall and test datasets, resulting in **54 baseline comparisons**. The key observations were:

- Most hardcoded strategies either made **very few trades** or invested large sums with negligible profits.
- Typical ROI hovered around −1% to +1%, with many configurations incurring consistent losses.
- Only one strategy (EMA+MACD+OBV) on S&P 500 showed modest gains, but with trivial capital involved.
- Compared to these, both the RL agents and buy-and-hold exhibited **significantly superior performance**.

These results underscore the importance of learning adaptive trading policies over relying solely on static rule-based systems.

6 Experimental Results

We now present the core experimental results from evaluating our RL agent across three major assets: Apple, S&P 500 Index, and Reliance Industries. We compare the agent's performance to traditional Buy-and-Hold and a suite of hardcoded technical strategies.

6.1 Apple (AAPL)

- **RL Agent (DQN + LSTM)** achieved an **ROI of 587.88%**.
- **Buy and Hold** over the same period yielded **578.28%** ROI.
- **Hardcoded strategies**, including EMA crossovers, MACD, Bollinger Bands, RSI, and others, performed worse with most delivering minor profits or flat returns.
- The RL agent made significantly more trades and capitalized on local trends that static strategies missed.

6.2 S&P 500 (Index)

- **RL Agent ROI: -4.73%** despite learning an active trading policy.
- **Buy and Hold ROI: -16.76%**, indicating a downward trend over the selected test window.
- **Hardcoded strategies** mostly avoided trading or made minimal losses (0–1% range), suggesting the dataset was more difficult to trade profitably.
- Notably, the RL agent still engaged in more trades than the static methods but could not outperform the negative trend.

6.3 Reliance Industries (RELIANCE)

- **RL Agent ROI: 459.23%**
- **Buy and Hold ROI: 465.99%**
- **Hardcoded methods** once again failed to capitalize on the price movements, with most delivering near-zero or negative ROI.
- RL agent achieved comparable performance to buy-and-hold but via shorter-term trades.

6.4 Summary Table

Asset	RL Agent ROI (%)	Buy and Hold ROI (%)	Amount Traded ($)
Apple	587.88	578.28	751,500
S&P 500	-4.73	-16.76	79,668,300
Reliance	459.23	465.99	23,900

Table 1: Performance Summary Across Assets

Note: While not shown for brevity, **all agents improved significantly with more training data**. Apple and Reliance in particular benefited from the full dataset, with noticeable increases in trading activity and profitability.

7 Analysis & Insights

After extensive experimentation across three real-world assets—**Apple**, **S&P 500**, and **Reliance**—several key insights emerged regarding the relative performance of RL agents, traditional baselines, and rule-based strategies.

DQN + LSTM vs. PPO: A Clear Winner

While PPO is a powerful policy-gradient algorithm in many domains, it consistently struggled in this sparse-reward trading setup. Despite multiple architectural variations, hyperparameter sweeps, and extended training windows, PPO failed to learn meaningful trade policies and often collapsed to a "Hold forever" strategy.

In contrast, the **DQN + LSTM agent demonstrated reliable and improving performance** across experiments:

- Successfully executed trades in varied market regimes.
- Captured temporal dependencies through its recurrent structure.

- Frequently matched or outperformed the buy-and-hold baseline.

Conclusion: The DQN + LSTM framework is retained as the foundation for future experimentation and refinement.

Hardcoded Strategies Underperform

Despite incorporating a variety of technical indicators (EMA, MACD, RSI, Bollinger Bands, OBV, etc.) with and without stop-loss logic, **none of the rule-based strategies delivered robust returns**.

- Most returned negligible or negative ROI, even with significant investment.
- More complex indicator combinations often led to zero trades due to conflicting conditions.
- Across the board, RL agents and even simple buy-and-hold policies outperformed.

Conclusion: Hardcoded logic lacks the adaptability required for dynamic market environments.

Case-by-Case Observations

- **Apple**: Agent ROI (587.88%) closely tracked and slightly outperformed Buy-and-Hold (578.28%), with $751,500 traded.
- **S&P 500**: Agent ROI was -4.73% compared to Buy-and-Hold's -16.76%, with heavy capital deployment ($79M+), suggesting risk-aware behavior.
- **Reliance**: Agent ROI of 459.23% nearly matched Buy-and-Hold's 465.99%, albeit with very few trades ($23,900), indicating minimal confidence in active trading.

Note: While not shown for every stage, all agents **consistently improved with larger training sets**, emphasizing the value of long historical data.

Key Insights and Implications

- **Agent behavior varied across assets** despite using the same model structure, highlighting market-specific dynamics learned during training.
- **Outperformance is not guaranteed**: RL agents closely tracked or lagged behind buy-and-hold in some cases, supporting ideas from the Efficient Market Hypothesis that markets may already encode all public information.

- **Risk awareness emerged organically**: Particularly in the S&P 500 case, the agent reduced downside exposure without any explicit risk constraint.
- **Generalization is limited**: The best Apple-trained agent transferred modestly to the S&P 500, but failed entirely on Reliance, suggesting retraining is necessary for new assets.

This phase demonstrates that learning-based agents can meaningfully compete with traditional strategies, while remaining adaptive and responsive to structural differences in market behavior. It lays the groundwork for more advanced reasoning, representation learning, and trading intelligence in future work.

8 Future Work

This project demonstrated a full pipeline for applying deep reinforcement learning (DQN + LSTM) to algorithmic trading using a realistic trading simulator and a diverse set of technical indicators. While the results were promising—especially compared to hard-coded baselines—there remain several impactful directions for future work to improve performance, generalization, and interpretability.

Enhanced DQN Variants (e.g., Double DQN, Dueling DQN)

Double DQN: The current agent uses the standard Q-learning TD target, which often overestimates action values:

$$\text{TD target} = r + \gamma \max_{a'} Q_{\text{target}}(s', a')$$

Double DQN decouples action selection and evaluation by using the online network to choose the best action and the target network to evaluate it:

$$\text{TD target} = r + \gamma Q_{\text{target}}(s', \arg\max_{a} Q_{\text{online}}(s', a))$$

This approach can yield **more stable and accurate learning**, especially in noisy environments like markets.

Dueling DQN: This architecture separates the estimation of state-value and advantage functions, allowing the network to **identify valuable states even when action choices have little impact**—which is common in sideways or low-volatility markets.

n-Step Returns Instead of 1-Step TD Learning

The current agent relies on 1-step temporal difference learning. However, rewards in trading often realize several steps after an action. Using **n-step returns** (e.g., 5-step, 10-step) could:

- Capture medium-term action consequences.
- Improve reward signal smoothing and credit assignment.
- Accelerate learning in trending or delayed-reward scenarios.

Actor-Critic and Parallelized Learning Agents

DQN is inherently off-policy and value-based. Actor-critic agents introduce more flexibility:

- **A3C (Asynchronous Advantage Actor-Critic):** Parallel agents explore simultaneously and update a shared policy—improving diversity and convergence speed.
- **SAC (Soft Actor-Critic):** Entropy-regularized learning encourages robust exploration in noisy domains.
- **PPO (with better implementation):** Though PPO underperformed in this project, it remains a strong candidate with continuous actions, better reward shaping, or hybrid state encodings.

Richer Feature Sets and Meta Information

The agent currently observes 10 normalized indicators and its own share count. However, markets respond to broader context:

- News sentiment, macroeconomic indicators.
- Cross-asset or sectoral signals.
- Intraday seasonalities (e.g., time of day).

Future agents could benefit from **attention-based encoders**, multi-modal inputs, or learned embeddings to dynamically focus on informative signals.

Synthetic Data Generation for Simulated Backtesting

One of the project's largest bottlenecks was limited access to rich, clean, and long-span intraday stock data. Generating synthetic environments would allow:

- Testing in rare or extreme market scenarios (e.g., 2008 crisis, COVID crash).
- Reproducible experimentation across different asset types.
- Robust stress-testing and evaluation of generalization.

Approaches could include GANs, autoregressive models, or domain-specific time series simulators.

Position Sizing and Risk Management Logic

Currently, the agent buys \$100 per action and sells all shares at once. More realistic agents could learn:

- Variable buy/sell amounts based on confidence or volatility.
- Risk-aware metrics like Sharpe ratio or drawdown.
- Portfolio-level decisions for multi-asset strategies.

Market Impact, Slippage & Realism Enhancements

The trading simulator assumes:

- Immediate order execution at the closing price.
- No liquidity or volume impact from trades.

To better reflect real-world constraints, future environments should include:

- Slippage models.
- Execution latency and order queues.
- Spread-aware price formation and limited order book depth.

Transfer Learning Across Assets

The Apple-trained agent showed some generalization to S&P 500, but failed on Reliance. Future work could explore:

- Fine-tuning agents across new stocks.
- Meta-learning techniques to build generalist agents.
- Domain adaptation between similar assets (e.g., tech stocks).

Counterfactual and Causal Reinforcement Learning

While TD learning propagates outcomes back to earlier states, it does not directly address the question: *"What should I have done instead?"*

To explore this, one could investigate:

- **Counterfactual learning**, to evaluate alternate trajectories.
- **Causal reinforcement learning**, which learns structural relationships and intervention effects.

In trading, this is especially important—because opportunity cost and missed timing can drastically alter profit potential.

Overall, these avenues open the door to more intelligent, adaptive, and interpretable trading agents that go beyond reactive learning to become truly strategic market participants.

References

1. Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (2nd ed.). MIT Press.
2. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal Policy Optimization Algorithms. *arXiv preprint arXiv:1707.06347.*
3. Fujimoto, S., van Hoof, H., & Meger, D. (2018). Addressing function approximation error in actor-critic methods. *Proceedings of the 35th International Conference on Machine Learning (ICML).*
4. Arjona, R., Blanco, A., & Curto, J. D. (2021). Deep reinforcement learning for trading with technical indicators. *Expert Systems with Applications*, 177, 114924.
5. Fama, E. F. (1970). Efficient Capital Markets: A Review of Theory and Empirical Work. *The Journal of Finance*, 25(2), 383–417.
6. Lillicrap, T. P., Hunt, J. J., Pritzel, A., et al. (2016). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971.*
7. Murphy, J. J. (1999). *Technical Analysis of the Financial Markets: A Comprehensive Guide to Trading Methods and Applications.* New York Institute of Finance.