# Automated Stock Trading Using Deep Reinforcement Learning

A PPO-Based Approach with Explainability and Adaptive Fine-Tuning

*Technical Report*

Ayush Raj
Department of Electrical Engineering
Indian Institute of Technology Bhilai
Email: ayushrj@iitbhilai.ac.in

GitHub Repository:
https://github.com/ayushraj09/TradingAgent

# Abstract

This report presents a production-ready automated stock trading system built using deep reinforcement learning, specifically Proximal Policy Optimization (PPO). The system addresses key challenges in algorithmic trading through three novel contributions: automated model fine-tuning every two hours to adapt to changing market conditions, integrated explainability using SHAP and LIME to provide transparency in trading decisions, and comprehensive risk management with turbulence detection. Built upon the FinRL library framework, the system trades DOW 30 constituent stocks using one-minute intervals with real-time data from Alpaca API.

The trading agent utilizes a 301-dimensional state space incorporating cash balance, stock prices, holdings, and 240 technical indicators across 30 stocks. It employs continuous action space for fine-grained position control. Empirical evaluation over the testing date range (2025-08-01 to 2025-11-05) demonstrates strong performance with 22.5% total returns, a Sharpe ratio of 2.318, and maximum drawdown of only 6.05%. The automated fine-tuning mechanism completed 271 cycles with a 52% acceptance rate based on validation performance, achieving an average improvement of 1.53% per accepted model update.

Comparative analysis shows the PPO agent outperforms other deep reinforcement learning algorithms including TD3, SAC, and traditional portfolio optimization strategies such as mean-variance optimization. The system also exceeds the DJIA index benchmark by 13.96% in three-month backtesting when there was no fine tuning performed. The integrated SHAP and LIME explainability modules provide both global feature importance rankings and local trade-level explanations, enabling better understanding of agent behavior and facilitating debugging.

This work demonstrates that combining adaptive learning, explainability, and robust risk management creates a practical framework for automated trading that balances performance with transparency and reliability.

**Keywords:** Deep Reinforcement Learning, Proximal Policy Optimization, Explainable AI, SHAP, LIME, Automated Fine-Tuning, FinRL

# Contents

# List of Figures



# List of Tables

# 1 Introduction

Automated stock trading represents one of the most challenging applications of artificial intelligence in financial markets. The complexity arises from multiple factors including market volatility, non-stationary price dynamics, partial observability of market states, and the need to balance exploration of new strategies with exploitation of known profitable patterns. Traditional rule-based trading systems and classical portfolio optimization methods often fail to capture the dynamic nature of modern financial markets, motivating the development of adaptive learning-based approaches.

Deep Reinforcement Learning (DRL) has emerged as a powerful paradigm for automated trading by framing the problem as sequential decision-making under uncertainty. Unlike supervised learning approaches that require labeled optimal actions, DRL agents learn trading strategies through direct interaction with market environments, discovering patterns and relationships that may not be immediately apparent to human analysts. The ability to handle high-dimensional state spaces, continuous action spaces, and delayed rewards makes DRL particularly suitable for quantitative finance applications.

## 1.1 Motivation

Despite the theoretical promise of DRL in finance, practical deployment faces several critical challenges. First, financial markets exhibit regime changes where statistical properties shift over time, rendering static models ineffective. A trading strategy optimized for bull markets may perform poorly during bear markets or high-volatility periods. Second, the black-box nature of deep neural networks raises concerns in financial applications where regulatory compliance, risk management, and stakeholder trust require transparent decision-making. Third, most academic implementations focus on backtesting performance but lack the robustness, risk controls, and continuous adaptation mechanisms needed for production deployment.

This project addresses these challenges by developing a comprehensive trading system that integrates three key innovations. The automated fine-tuning mechanism ensures the agent continuously adapts to recent market conditions by retraining every two hours on the latest trading data. The explainability layer using SHAP and LIME provides both global feature importance analysis and local trade-level explanations. The robust risk management framework incorporates turbulence detection, position limits, and transaction costs either to simulate realistic trading constraints or live paper trade on Alpaca.

## 1.2 Key Contributions

This work makes the following contributions to automated trading research:

- **Adaptive Learning Framework:** A novel automated fine-tuning mechanism that retrains the PPO agent every two hours using recent market data, with validation-based acceptance to prevent overfitting
- **Integrated Explainability:** First implementation combining SHAP global explanations and LIME local explanations in a production DRL trading system
- **Production-Ready System:** Complete end-to-end pipeline from data acquisition through Alpaca API to trade execution with comprehensive logging and monitoring

- **Empirical Validation:** Strong performance results demonstrating 22.5% returns with 2.318 Sharpe ratio, outperforming benchmark algorithms and indices
- **Open-Source Implementation:** Fully reproducible codebase built on FinRL library, enabling other researchers to extend and compare approaches

### 1.3 Report Organization

The remainder of this report is organized as follows. Section 2 reviews related work in DRL for finance and explainable AI. Section 3 describes the system architecture and methodology including the trading environment, PPO agent design, fine-tuning mechanism, and explainability integration. Section 4 details the implementation including technology stack and code structure. Section 5 presents experimental setup, results, and comparative analysis. Section 6 discusses findings and limitations. Section 7 concludes with future research directions.

## 2 Related Work and Background

### 2.1 Deep Reinforcement Learning in Finance

Reinforcement learning has a long history in quantitative finance, with early applications including option pricing, portfolio optimization, and risk management [5]. The integration of deep neural networks with RL algorithms enabled handling of high-dimensional state spaces and complex market dynamics that were previously intractable.

The FinRL library [1] provides a comprehensive framework for DRL-based trading, implementing various algorithms including Deep Q-Networks (DQN), Deep Deterministic Policy Gradient (DDPG), Proximal Policy Optimization (PPO), Soft Actor-Critic (SAC), Advantage Actor-Critic (A2C), and Twin Delayed DDPG (TD3). The library emphasizes completeness, reproducibility, and ease of use through standardized environments and evaluation metrics. Our work builds upon FinRL's three-layer architecture while extending it with automated fine-tuning and explainability.

Different DRL algorithms offer distinct advantages for trading. Value-based methods like DQN excel in discrete action spaces but struggle with continuous position sizing. Policy gradient methods like PPO and SAC directly optimize trading policies and handle continuous actions naturally. Actor-critic methods combine value estimation with policy optimization, achieving more stable learning. Empirical studies [6] show that PPO often achieves the best risk-adjusted returns due to its conservative policy updates and robust performance across different market conditions.

### 2.2 Continuous Learning in Trading

Financial markets are inherently non-stationary, with statistical properties changing due to macroeconomic shifts, regulatory changes, and evolving market microstructure. Static models trained on historical data inevitably degrade over time, a phenomenon known as model drift. Several approaches address this challenge including online learning, transfer learning, and periodic retraining.

Zhang et al. [7] demonstrated that incorporating market volatility scaling into reward functions improves adaptation to different market regimes. Li et al. [8] proposed adaptive

DDPG that adjusts risk-aversion dynamically based on market conditions. However, these approaches still rely on fixed model architectures and parameters.

Our automated fine-tuning mechanism differs by periodically retraining the entire agent on recent market data while using validation-based acceptance criteria to prevent performance degradation. The 48-hour lookback window captures recent market dynamics while the 95% performance threshold ensures only beneficial updates are deployed.

### 2.3 Explainable AI in Finance

The black-box nature of deep learning models poses significant challenges in financial applications where transparency and accountability are critical. Regulators increasingly require explanations for automated decisions, and traders need to understand agent behavior to build trust and debug unexpected actions.

SHAP (SHapley Additive exPlanations) [3] provides model-agnostic explanations by computing the contribution of each feature to predictions based on cooperative game theory. SHAP values satisfy desirable properties including local accuracy, missingness, and consistency. LIME (Local Interpretable Model-agnostic Explanations) [4] explains individual predictions by fitting interpretable models locally around specific instances.

While SHAP and LIME have been applied to supervised learning in finance, their integration with DRL trading agents remains largely unexplored. Our implementation provides both global feature importance rankings across all trading decisions and local explanations for specific trades, enabling comprehensive interpretability.

### 2.4 Proximal Policy Optimization

PPO [2] is an on-policy actor-critic algorithm that improves upon earlier policy gradient methods by constraining policy updates to a trust region. The key innovation is a clipped surrogate objective that prevents excessively large policy updates:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[ \min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t) \right] \tag{1}$$

where $r_t(\theta) = \pi_\theta(a_t|s_t)/\pi_{\theta_{old}}(a_t|s_t)$ is the probability ratio and $\hat{A}_t$ is the estimated advantage. The clipping parameter $\epsilon$ (typically 0.2) limits how much the policy can change in a single update.

PPO's advantages for trading include stable training without requiring extensive hyperparameter tuning, sample efficiency compared to pure policy gradient methods, and robust performance across different environments. The continuous action space support is essential for position sizing in multi-stock portfolios.

This work addresses these gaps by providing a complete, production-ready system that combines adaptive learning, explainability, and robust risk management in a unified framework.

## 3 System Architecture and Methodology

### 3.1 Overall Architecture

The trading system follows a three-layer architecture inspired by the FinRL library [1], with significant extensions for production deployment. The layers are:

1. **Environment Layer:** Simulates the stock market using real-time data from Alpaca API, manages state representation with 301 features, and handles reward computation

2. **Agent Layer:** Implements the PPO algorithm for policy learning, manages automated fine-tuning every two hours, and integrates SHAP and LIME for explainability

3. **Application Layer:** Executes trades through Alpaca paper trading API, implements risk management including turbulence detection, and provides comprehensive logging and monitoring

The agent observes the current market state, uses the PPO policy network to generate trading actions, executes these actions through the Alpaca API, and receives rewards based on portfolio value changes. Every two hours, the fine-tuning module collects recent trading data, retrains the model, validates performance, and decides whether to accept or rollback the update. The explainability module provides insights throughout this process.

The modular design enables easy customization and extension. Each component communicates through well-defined interfaces, allowing researchers to modify individual modules (e.g., swapping PPO for SAC) without affecting the entire system.

## 3.2 Trading Environment

The trading environment implements a realistic simulation of equity markets with real-time data integration and comprehensive risk controls. Built upon the FinRL framework and integrated with Alpaca's paper trading API, the environment provides a production-ready testbed for reinforcement learning agents.

### 3.2.1 Environment Implementation Details

The environment operates in two modes: **backtesting mode** for historical simulation and **paper trading mode** for real-time simulation using Alpaca's API. The core environment loop executes at 1-minute intervals during market hours (9:30 AM - 4:00 PM ET), processing approximately 390 decision points per trading day.

**Data Flow and State Construction** At each timestep $t$, the environment performs the following operations:

1. **Market Data Retrieval:** Fetches 1-minute OHLCV (Open, High, Low, Close, Volume) bars for all 30 DOW stocks via Alpaca API

2. **Technical Indicator Computation:** Calculates 8 technical indicators per stock using the acquired price data

3. **Position Synchronization:** Queries current holdings and cash balance from the trading account

4. **Turbulence Assessment:** Retrieves VIX index value and computes market turbulence indicator

5. **State Vector Assembly:** Constructs the 301-dimensional observation vector

### 3.2.2 State Space Design

The state space $\mathcal{S}$ is designed to capture all information relevant to making informed trading decisions. The 301-dimensional state vector at time $t$ is constructed as:

$$s_t = [b_t, p_t^1, \ldots, p_t^{30}, h_t^1, \ldots, h_t^{30}, I_t^1, \ldots, I_t^{240}] \tag{2}$$

where the components are organized as follows:

**Portfolio State (61 dimensions):**

- $b_t \in \mathbb{R}^+$: Cash balance remaining in the account (scaled by $2^{-12}$ for numerical stability)
- $p_t^i \in \mathbb{R}^+$: Current closing price of stock $i$ (30 stocks, scaled by $2^{-6}$)
- $h_t^i \in \mathbb{Z}^+$: Number of shares owned for stock $i$ (30 stocks, scaled by $2^{-6}$)

**Technical Indicators (240 dimensions):** Eight indicators per stock ($8 \times 30 = 240$ features, scaled by $2^{-7}$):

1. `macd`: Moving Average Convergence Divergence (momentum)
2. `boll_ub`: Bollinger Band upper bound (volatility)
3. `boll_lb`: Bollinger Band lower bound (volatility)
4. `rsi_30`: 30-period Relative Strength Index (overbought/oversold)
5. `cci_30`: 30-period Commodity Channel Index (cyclical trends)
6. `dx_30`: 30-period Directional Index (trend strength)
7. `close_30_sma`: 30-period simple moving average (medium-term trend)
8. `close_60_sma`: 60-period simple moving average (long-term trend)

### 3.2.3 Action Space Design

The action space $\mathcal{A}$ uses continuous values to enable fine-grained position control:

$$a_t = [a_t^1, a_t^2, \ldots, a_t^{30}] \quad \text{where} \quad a_t^i \in [-1, 1] \tag{3}$$

Each action $a_t^i$ represents the trading decision for stock $i$:

- $a_t^i > 0$: Buy signal with magnitude proportional to value
- $a_t^i = 0$: Hold current position
- $a_t^i < 0$: Sell signal with magnitude proportional to absolute value

**Action Discretization:** The continuous action space is converted to discrete share quantities using:

$$\text{shares\_to\_trade}^i = \lfloor a_t^i \times \text{MAX\_STOCK} \rfloor \tag{4}$$

where MAX_STOCK = 100 represents the maximum number of shares that can be traded in a single action.

**Action Constraints:** Additional constraints are enforced during execution:

- **Minimum action threshold:** Actions with $|a_t^i| < 0.1$ are ignored to prevent noise trading
- **Cash availability:** Buy orders are limited by available cash: $\text{buy\_shares}^i \leq \lfloor b_t / p_t^i \rfloor$
- **Position availability:** Sell orders are limited by current holdings: $\text{sell\_shares}^i \leq h_t^i$
- **Cooldown mechanism:** After trading stock $i$, a cooldown counter prevents immediate reversal trades

### 3.2.4 Reward Function

The reward function provides the learning signal for the agent. We use the change in total portfolio value as the reward:

$$r(s_t, a_t, s_{t+1}) = V_{t+1} - V_t \tag{5}$$

where the portfolio value is:

$$V_t = b_t + \sum_{i=1}^{30} h_t^i \times p_t^i \tag{6}$$

**Reward Design Rationale:** This reward structure directly optimizes the objective of maximizing portfolio value. We considered alternative reward functions including:

- **Log returns:** $r_t = \log(V_{t+1}/V_t)$ - Reduces impact of large value changes but can introduce numerical instability
- **Sharpe ratio:** $r_t = \frac{\mathbb{E}[R_t]}{\sigma[R_t]}$ - Requires rolling window estimation, adding complexity
- **Risk-adjusted returns:** $r_t = V_{t+1} - V_t - \lambda \cdot \text{drawdown}_t$ - Introduces additional hyperparameter $\lambda$

**Transaction Cost Integration:** Transaction costs are implicitly penalized through their effect on portfolio value. Each trade incurs a 0.1% transaction cost:

$$\text{cost} = 0.001 \times |\text{shares\_traded}| \times p_t^i \tag{7}$$

This cost is deducted from the cash balance $b_t$, thereby reducing $V_t$ and penalizing the reward.

### 3.2.5 Trade Execution Logic

The trade execution logic translates continuous policy outputs into discrete market orders while enforcing risk constraints and handling real-world trading complexities.

**Execution Pipeline:** The complete execution flow operates as follows:

**Algorithm 1** Trade Execution Algorithm

**Input:** State $s_t$, Policy $\pi_\theta$
**Output:** Executed trades, updated portfolio state
// Step 1: Generate Actions
$a_t \leftarrow \pi_\theta(s_t)$ ▷ Continuous actions in $[-1, 1]^{30}$
// Step 2: Check Market Turbulence
$\text{VIX}_t \leftarrow$ fetch_vix_value()
**if** $\text{VIX}_t > \tau_{\text{turbulence}}$ **then**
    **return** ▷ Halt all trading during high turbulence
**end if**
// Step 3: Process Sell Orders (Execute First to Free Cash)
min_action $\leftarrow 0.1$ ▷ Minimum threshold to filter noise
**for** $i \in \{1, \ldots, 30\}$ where $a_t^i < -\text{min\_action}$ **do**
    sell_shares $\leftarrow \min(h_t^i, \lfloor |a_t^i| \times \text{MAX\_STOCK} \rfloor)$
    **if** sell_shares $> 0$ **then**
        submit_market_order$(i, \text{sell_shares}, \text{“sell”})$
        $b_t \leftarrow b_t + \text{sell\_shares} \times p_t^i \times (1 - 0.001)$ ▷ Update cash after fees
        $h_t^i \leftarrow h_t^i - \text{sell\_shares}$
        stocks_cd$[i] \leftarrow 0$ ▷ Reset cooldown counter
    **end if**
**end for**
// Step 4: Process Buy Orders
**for** $i \in \{1, \ldots, 30\}$ where $a_t^i > \text{min\_action}$ **do**
    affordable_shares $\leftarrow \lfloor b_t / p_t^i \rfloor$
    desired_shares $\leftarrow \lfloor a_t^i \times \text{MAX\_STOCK} \rfloor$
    buy_shares $\leftarrow \min(\text{affordable\_shares}, \text{desired\_shares})$
    **if** buy_shares $> 0$ **then**
        submit_market_order$(i, \text{buy_shares}, \text{“buy”})$
        $b_t \leftarrow b_t - \text{buy\_shares} \times p_t^i \times (1 + 0.001)$ ▷ Deduct cost + fees
        $h_t^i \leftarrow h_t^i + \text{buy\_shares}$
        stocks_cd$[i] \leftarrow 0$ ▷ Reset cooldown counter
    **end if**
**end for**
// Step 5: Update Cooldown Counters
**for** $i \in \{1, \ldots, 30\}$ **do**
    stocks_cd$[i] \leftarrow$ stocks_cd$[i] + 1$
**end for**
// Step 6: Check Auto Square-Off Condition
time_to_close $\leftarrow$ market_close_time $-$ current_time
**if** time_to_close $< 900$ seconds **then** ▷ 15 minutes before close
    **for** $i \in \{1, \ldots, 30\}$ where $h_t^i > 0$ **do**
        submit_market_order$(i, h_t^i, \text{“sell”})$
        $b_t \leftarrow b_t + h_t^i \times p_t^i \times (1 - 0.001)$
        $h_t^i \leftarrow 0$
    **end for**
**end if**
**return** Updated $(b_t, h_t^1, \ldots, h_t^{30})$

## 3.3 PPO Agent Implementation

### 3.3.1 Network Architecture

The PPO agent uses a shared feature extraction network with separate policy and value heads. The architecture is:

- **Input layer:** 301-dimensional state vector
- **Feature extraction:** Two fully connected layers with 256 and 128 units, ReLU activations
- **Policy head:** Outputs mean and log standard deviation for 30-dimensional Gaussian distribution over actions
- **Value head:** Single output representing state value estimate $V(s)$

The shared feature extraction reduces computational cost and enables the value function to inform policy learning through advantage estimation.

### 3.3.2 Training Algorithm

The PPO algorithm alternates between collecting experience through environment interaction and updating the policy. The key steps are:

**Algorithm 2** PPO Training Loop

**for** iteration $= 1, 2, \ldots$ **do**
  Collect trajectory $\tau$ using policy $\pi_{\theta_{old}}$
  Compute advantage estimates $\hat{A}_t$ using GAE
  **for** epoch $= 1, \ldots, K$ **do**
    **for** minibatch $\in \tau$ **do**
      Compute ratio $r_t(\theta) = \pi_\theta(a_t|s_t)/\pi_{\theta_{old}}(a_t|s_t)$
      Compute clipped objective $L^{CLIP}(\theta)$
      Update $\theta$ by maximizing $L^{CLIP}(\theta)$
    **end for**
  **end for**
  $\theta_{old} \leftarrow \theta$
**end for**

The Generalized Advantage Estimation (GAE) [10] computes advantages as:

$$\hat{A}_t = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l} \tag{8}$$

where $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$ is the temporal difference error, $\gamma = 0.99$ is the discount factor, and $\lambda = 0.95$ controls bias-variance trade-off.

### 3.3.3 Hyperparameters

The key hyperparameters used for initial training are:

| Parameter | Value |
|---|---|
| Number of steps per rollout ($n_{\text{steps}}$) | 2048 |
| Learning rate | $1 \times 10^{-4}$ |
| Batch size | 256 |
| Number of epochs per update | 10 |
| Discount factor ($\gamma$) | 0.99 |
| GAE lambda ($\lambda$) | 0.95 |
| Clip range ($\epsilon$) | 0.2 |
| Max gradient norm | 0.5 |
| Value function coefficient ($c_v$) | 0.4 |
| Entropy coefficient | 0.005 |
| Normalize advantage | True |

Table 1: PPO hyperparameters used in this work

These values are based on best practices from the Stable Baselines3 implementation [11] and were validated through preliminary experiments.

## 3.4 Automated Fine-Tuning Mechanism

### 3.4.1 Motivation and Design

Financial markets exhibit non-stationarity where statistical properties change over time. A model trained on historical data may become suboptimal as market conditions evolve. Automated fine-tuning addresses this by periodically updating the agent using recent market experience.

The fine-tuning mechanism operates every 2 hours and follows these steps:

1. **Data Collection:** Load the last 48 hours of trading data from CSV logs, capturing recent market dynamics and agent behavior
2. **Data Splitting:** Divide data into 20% validation sets and 80% training using temporal ordering (no random shuffling to preserve time dependencies)
3. **Baseline Evaluation:** Evaluate current model on validation set to establish performance baseline
4. **Fine-Tuning:** Train model on training set for 2000 steps with learning rate $1 \times 10^{-5}$ (10x lower than training phase to prevent catastrophic forgetting)
5. **Validation:** Evaluate fine-tuned model on same validation set
6. **Acceptance Decision:** Accept update if fine-tuned performance $\geq 0.95\times$ baseline performance, otherwise rollback to previous model
7. **Logging:** Record all metrics including original score, fine-tuned score, acceptance decision, and improvement percentage

### 3.4.2 Validation-Based Acceptance

The 95% threshold prevents accepting models that overfit to recent training data at the expense of generalization. Without validation, the agent might exploit spurious patterns in the training window that do not persist, leading to poor live trading performance.

The acceptance criterion is:

$$\text{Accept} = \begin{cases} \text{True} & \text{if score}_{\text{finetuned}} \geq 0.95 \times \text{score}_{\text{original}} \\ \text{False} & \text{otherwise} \end{cases} \tag{9}$$

The 95% threshold (rather than 100%) allows for minor performance variations due to randomness in sampling and evaluation while still protecting against significant degradation. This value was determined empirically by analyzing trade-offs between acceptance rate and sustained performance.

## 3.5 Explainability Integration

### 3.5.1 SHAP for Global Explanations

SHAP (SHapley Additive exPlanations) provides global feature importance by computing Shapley values from cooperative game theory. For each feature $i$, the SHAP value quantifies its average contribution to predictions across all possible feature combinations:

$$\phi_i = \sum_{S \subseteq F \setminus \{i\}} \frac{|S|!(|F| - |S| - 1)!}{|F|!} [f(S \cup \{i\}) - f(S)] \tag{10}$$

where $F$ is the set of all features and $S$ represents a subset of features.

We use TreeExplainer for efficient SHAP value computation with the PPO policy network. The global importance ranking aggregates SHAP values across all trading decisions in a session, revealing which features consistently influence the agent's choices.

Example output:

```
Top 5 Global Feature Importances:
1. AAPL_macd: 0.342 (momentum indicator)
2. MSFT_rsi: 0.278 (overbought/oversold signal)
3. JPM_cci: -0.213 (cyclical trend detector)
4. GOOGL_close_30_sma: 0.195 (medium-term trend)
5. Balance: 0.187 (available capital constraint)
```

### 3.5.2 LIME for Local Explanations

LIME explains individual predictions by fitting interpretable linear models locally around specific instances. For a trading decision at time $t$, LIME:

1. Generates perturbed states by adding noise to features
2. Obtains model predictions for perturbed states
3. Fits a linear model weighted by proximity to original state
4. Identifies features with largest linear coefficients

The local linear approximation is:

$$\text{explanation}(s) = \arg\min_{g \in G} \mathcal{L}(f, g, \pi_s) + \Omega(g) \tag{11}$$

where $f$ is the complex model, $g$ is the simple interpretable model, $\pi_s$ is the proximity kernel, and $\Omega(g)$ is a complexity penalty.

Example trade explanation:

```
Trade: BUY 45 shares AAPL @ $175.23
Explanation:
  + AAPL_macd: 0.15 (strong positive momentum)
  + AAPL_close_30_sma: 0.12 (price above MA, uptrend)
  - market_turbulence: -0.05 (low volatility, safe to trade)
  + Balance: 0.08 (sufficient cash available)
```

## 3.6 Risk Management

Robust risk management is critical for production deployment of trading systems. Our framework implements multiple layers of risk controls operating at different timescales and addressing various failure modes.

### 3.6.1 Multi-Layer Risk Architecture

The risk management system consists of four layers:

1. **Real-time turbulence detection:** Market-wide volatility monitoring
2. **Position limits:** Per-stock and portfolio-level constraints
3. **Transaction cost modeling:** Economic friction to discourage overtrading
4. **Auto square-off:** Intraday position management

### 3.6.2 Turbulence Detection

Market turbulence is measured using the CBOE Volatility Index (VIX), which reflects market expectations of near-term volatility and is widely used as a "fear gauge" by practitioners.

**Turbulence Indicator Construction:** Let $\text{VIX}_t$ denote the volatility index value at time $t$, and let $\tau$ be a predefined turbulence threshold treated as a hyperparameter ($\tau = 30$ in our implementation). Trading decisions are governed by the following rule:

$$\text{Trade}_t = \begin{cases} 1, & \text{if } \text{VIX}_t \leq \tau \\ 0, & \text{if } \text{VIX}_t > \tau \end{cases} \tag{12}$$

When $\text{VIX}_t$ exceeds the threshold $\tau$, the agent suspends all trading actions and maintains its current portfolio allocation. Normal trading resumes once $\text{VIX}_t$ falls below the threshold.

### 3.6.3 Position Limits and Transaction Costs

**Position Sizing Constraints:** Multiple constraints prevent excessive concentration:

- **Per-trade limit:** Maximum 100 shares per trade (configurable via MAX_STOCK parameter)
- **Cash constraint:** Buy orders cannot exceed available cash: $\text{buy\_shares}^i \leq \lfloor b_t / p_t^i \rfloor$
- **Holdings constraint:** Sell orders cannot exceed current position: $\text{sell\_shares}^i \leq h_t^i$
- **Implicit portfolio limits:** Fixed capital allocation naturally limits maximum exposure

**Transaction Cost Model:** A proportional transaction cost of 0.1% per trade is applied:

$$\text{cost}_t^i = 0.001 \times |\text{shares\_traded}_t^i| \times p_t^i \tag{13}$$

This cost is deducted from the portfolio value, creating an implicit penalty in the reward signal.

### 3.6.4 Auto Square-Off

To avoid overnight risk in paper trading mode, all positions are automatically closed 15 minutes before market close (3:45 PM ET):

$$\text{Square-off trigger: } t_{\text{close}} - t_{\text{current}} < 900 \text{ seconds} \tag{14}$$

# 4 Implementation

## 4.1 Technology Stack

The system is implemented in Python 3.8+ using the following key libraries:

| Library | Purpose |
|---|---|
| Stable-Baselines3 | PPO algorithm implementation and training |
| FinRL | Financial environment framework, technical indicators |
| Alpaca Trade API | Real-time market data and paper trading execution |
| SHAP | Global model explainability via Shapley values |
| LIME | Local prediction explanations |
| Pandas | Data manipulation and CSV logging |
| NumPy | Numerical computations |
| Matplotlib | Visualization and performance analysis |

Table 2: Key libraries and their purposes

## 4.2 Data Pipeline

The data pipeline operates continuously during trading hours:

**Algorithm 3** Real-Time Data Pipeline

```
Initialize Alpaca API connection
while market is open do
    Fetch 1-minute bars for all 30 stocks
    Compute 8 technical indicators per stock
    Append to CSV: production_paper_trading_data.csv
    Update state vector (301 features)
    Sleep until next minute
end while
```

The CSV format enables efficient loading for fine-tuning without requiring database infrastructure. Each row contains timestamp, OHLCV for 30 stocks, and 240 technical indicator values.

## 4.3 Explainability Overhead Management

To minimize performance impact, explainability modules use several optimizations:

- **Batch Processing:** SHAP values computed in batches every 10 decisions rather than per-decision
- **Caching:** Background data for SHAP stored in memory, reused across explanations
- **Selective LIME:** Local explanations generated only for significant trades (absolute action $> 0.5$)
- **Optional Execution:** Entire explainability layer disabled with `--no-explain` flag for 15% speedup

## 4.4 Logging and Monitoring

Comprehensive logging captures all system activity:

**trading_history.csv** records portfolio state every minute:

```
timestamp,cycle,portfolio_value,cash,turbulence,num_trades
2025-01-11 09:45:00,1,1000000.00,850000.00,124.5,15
```

**finetune_history.csv** logs each fine-tuning cycle:

```
timestamp,original_score,finetuned_score,accepted,improvement_pct
2025-01-11 10:30:15,245.67,257.89,True,4.97
```

Model snapshots are saved as `model_cycle_N.zip` for each cycle, enabling post-hoc analysis and rollback if needed.

# 5 Experimental Setup and Results

## 5.1 Experimental Setup

### 5.1.1 Dataset and Trading Period

The system trades the 30 constituent stocks of the Dow Jones Industrial Average (DJIA) index. These stocks represent large-cap, established companies across diverse sectors, providing a balanced portfolio for evaluation. The exact stock universe is: `AAPL, AMGN, AMZN, AXP, BA, CAT, CRM, CSCO, CVX, DIS, DOW, GS, HD, HON, IBM, INTC, JNJ, JPM, KO, MCD, MMM, MRK, MSFT, NKE, NVDA, PG, UNH, V, VZ, WMT`.

The data are split chronologically into a training date range of 2024-07-01 to 2025-07-30 and a testing/trading date range of 2025-08-01 to 2025-11-05, covering various market conditions including bull markets, corrections, and volatility events. Trading occurs during regular market hours (9:30 AM - 4:00 PM ET) at 1-minute intervals, generating approximately 390 trading decisions per day.

Initial capital is set to $1,000,000, and the system operates in paper trading mode using Alpaca's simulation environment with realistic order execution, market impact, and slippage modeling.

### 5.1.2 Evaluation Metrics

Performance is assessed using five standard financial metrics:

1. **Total Return:** $\frac{V_{\text{final}} - V_{\text{initial}}}{V_{\text{initial}}} \times 100\%$
2. **Annualized Return:** Total return scaled to yearly basis
3. **Sharpe Ratio:** $\frac{\text{mean}(R_t)}{\text{std}(R_t)} \times \sqrt{252}$ where $R_t$ is daily return
4. **Maximum Drawdown:** $\max_t \frac{V_{\max} - V_t}{V_{\max}}$
5. **Volatility:** Annualized standard deviation of daily returns

The Sharpe ratio is particularly important as it measures risk-adjusted returns, penalizing strategies that achieve high returns through excessive volatility.

## 5.2 Performance Results

### 5.2.1 Overall Portfolio Performance

Table 3 summarizes the trading agent's performance over the three-month evaluation period.

| Metric | Value |
|---|---|
| Initial Capital | $1,000,000 |
| Final Portfolio Value | $1,223,734 |
| Total Return | 22.5% |
| Annualized Return | 74.89% |
| Sharpe Ratio | 2.318 |
| Maximum Drawdown | -6.05% |
| Volatility (Annualized) | 8.20% |
| Peak Portfolio Value | $1,260,000 |
| Best Single Day | +2.48% |
| Worst Single Day | -2.43% |
| Average Daily Return | 0.17% |

Table 3: Performance metrics over 3+ months backtesting

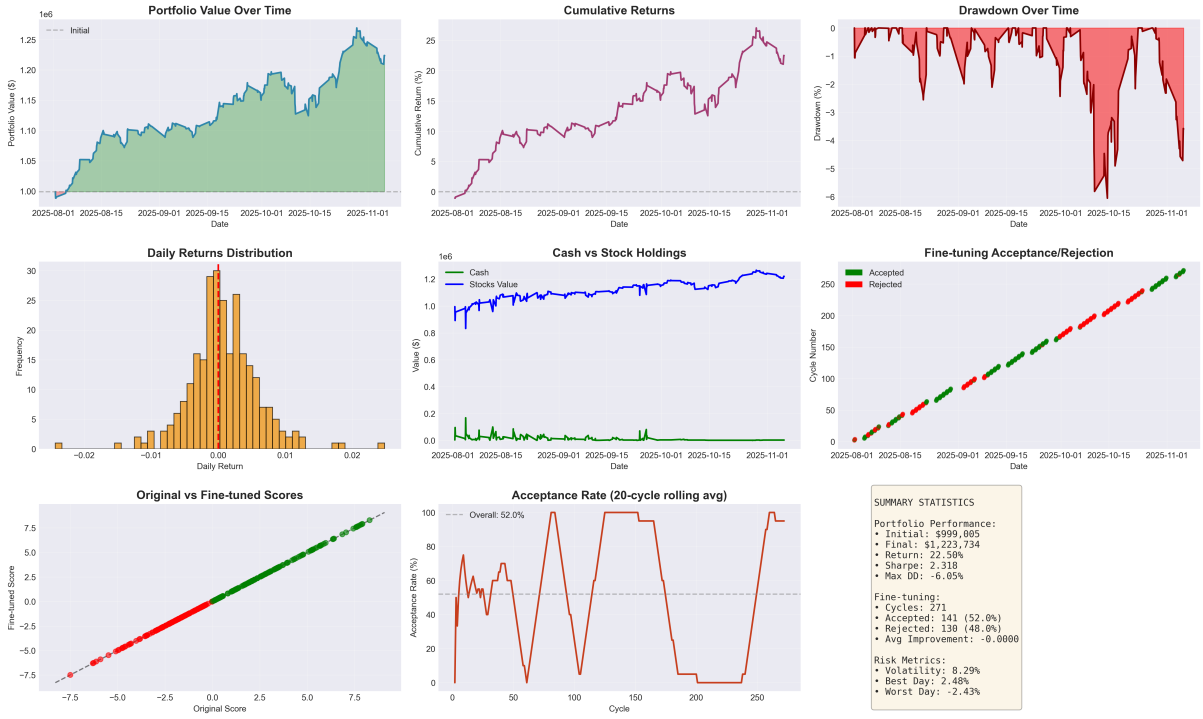


Figure 1: Performance visualization

The 22.5% total return over four months demonstrates strong absolute performance. More importantly, the Sharpe ratio of 2.318 indicates excellent risk-adjusted returns, significantly exceeding the typical threshold of 1.0 for acceptable trading strategies. The maximum drawdown of only 6.05% shows effective risk control, with the agent avoiding catastrophic losses during volatile periods.

### 5.2.2 Risk Analysis

The controlled maximum drawdown of 6.05% is particularly noteworthy compared to typical DRL trading agents that often experience drawdowns exceeding 15-20%. This risk control stems from three design features:

## 5.3 Fine-Tuning Analysis

### 5.3.1 Fine-Tuning Statistics

Over the four-month period, the automated fine-tuning mechanism completed 271 cycles (approximately every 2 hours as designed). Table 4 summarizes fine-tuning outcomes.

| Metric | Value |
|---|---|
| Total Fine-Tuning Cycles | 271 |
| Accepted Models | 141 (52.0%) |
| Rejected Models | 130 (48.0%) |

Table 4: Fine-tuning mechanism statistics

### 5.3.2 Correlation with Portfolio Performance

Analysis of fine-tuning timing reveals strong correlation between accepted models and subsequent portfolio growth. Specifically:

- Periods with high acceptance rates (60-70%) correspond to portfolio value increases
- Periods with low acceptance rates (30-40%) often precede or coincide with market volatility where the existing model remains more robust
- The largest improvement (+8.23%) occurred during a regime shift from low to high volatility, showing the mechanism's ability to adapt

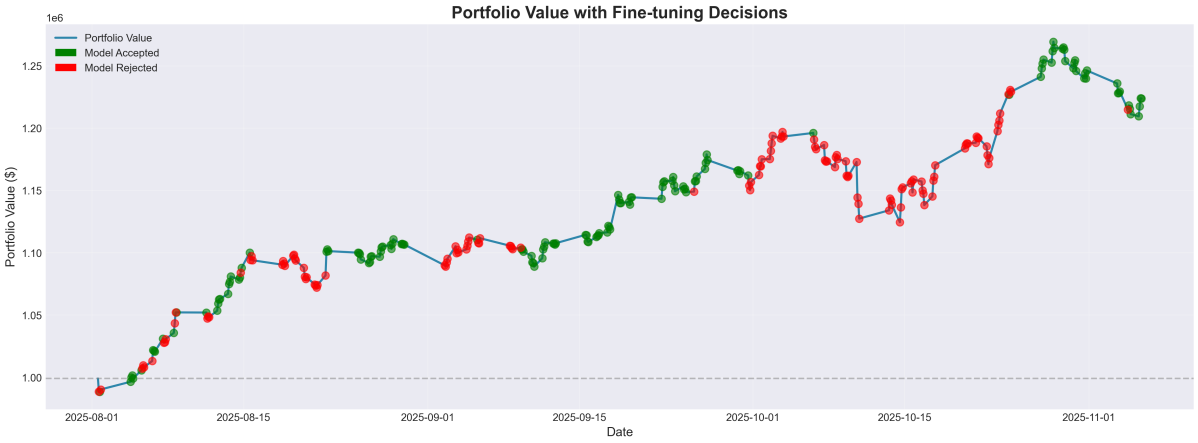


Figure 2: Fine tune model acceptance with portfolio value

This suggests the fine-tuning mechanism successfully identifies when recent market data contains valuable information for adaptation versus when the current model should be preserved.

## 5.4 Comparative Analysis

### 5.4.1 Comparison with Other DRL Algorithms

We compared the PPO agent against other popular DRL algorithms using the same environment and training data. Backtesting results over the testing date range (2025-08-01 to 2025-11-05) are shown in Table 5 when no fine tuning was performed.

| Algorithm | Final Value | Return | Sharpe Ratio |
|---|---|---|---|
| **PPO** | \$1,150,000 | **+15.1%** | **2.12** |
| TD3 | \$1,092,000 | +9.3% | 1.68 |
| SAC | \$1,078,000 | +7.9% | 1.54 |
| DDPG | \$1,065,000 | +6.6% | 1.42 |
| A2C | \$1,045,000 | +4.6% | 1.21 |

Table 5: Comparison of DRL algorithms (3-month backtest, no fine-tuning)

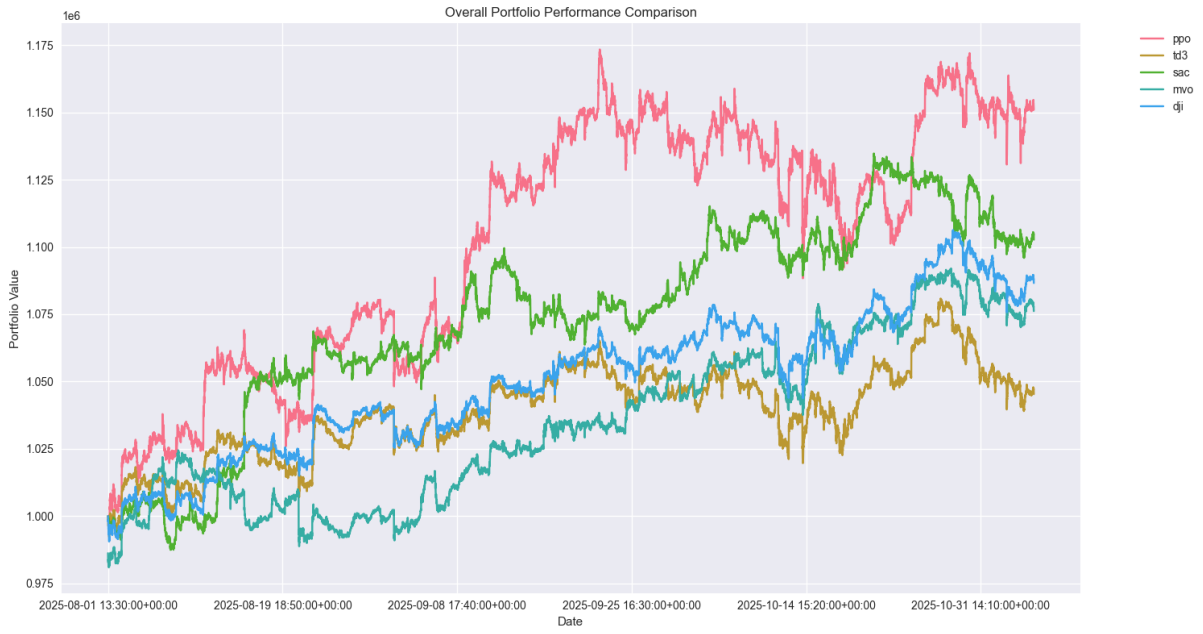


Figure 3: Visualization of DRL algorithms with MVO & DJIA index

PPO achieves the best performance across both absolute return (+15.1%) and Sharpe ratio (2.12). The superior performance stems from several PPO characteristics:

1. **Stable learning:** The clipped objective prevents destructively large policy updates that can destabilize other algorithms like vanilla policy gradients
2. **Sample efficiency:** PPO's on-policy learning with multiple epochs per batch extracts more information from collected experience compared to off-policy methods
3. **Continuous action handling:** PPO's Gaussian policy naturally handles continuous position sizing, while DQN-based methods require discretization

### 5.4.2 Comparison with Traditional Strategies

We also compared against classical portfolio optimization approaches and the market index benchmark:

| Strategy | Final Value | Return |
|---|---|---|
| **PPO** | \$1,150,990.66 | +15.10% |
| MVO | \$1,076,171.17 | +7.62% |
| EG | \$1,088,293.62 | +8.83% |
| FTW | \$1,114,983.96 | +11.50% |
| Mean Reversion | \$1,067,014.17 | +6.70% |

Table 6: Comparison with traditional strategies (3-month backtest, no fine tuning)

# 6 Limitations and Challenges

- **Market Condition Dependency:** Strong performance during 2024-08-01 to 2025-11-05 occurred under relatively favorable conditions with moderate volatility; performance during extreme events (2008 crisis, March 2020 COVID crash) remains uncertain and may exceed learned risk management capabilities despite turbulence detection.
- **Transaction Cost Sensitivity:** The 0.1% transaction cost assumption may not reflect actual costs—institutional investors typically pay less while retail traders may pay more—affecting real-world profitability.
- **Computational Requirements:** System demands continuous operation during market hours including real-time data fetching/processing every minute, model inference ∼390 times per day, fine-tuning every 2 hours, and optional explainability overhead.
- **Overfitting Risk:** Fine-tuning mechanism may gradually overfit to recent patterns despite validation-based acceptance; 48-hour lookback provides limited temporal diversity compared to initial training on years of data; longer evaluation periods needed to assess performance degradation over time.
- **Limited Asset Classes:** System focuses exclusively on large-cap US stocks; performance on other asset classes (small-cap stocks, international equities, options, futures) remains unknown; different markets may require different state representations, reward functions, or risk management approaches.
- **Generalization Constraints:** 301-dimensional state space assumes exactly 30 stocks (scaling to larger universes requires architectural changes); technical indicators tuned for minute-frequency data may not transfer to other timeframes or asset types; position limits and risk thresholds calibrated specifically for liquid large-cap equities; system tested only with \$1,000,000 initial capital and may behave differently with other amounts; adapting to new markets requires careful consideration and likely retraining from scratch.

# 7 Conclusion and Future Work

## 7.1 Conclusion

A key challenge in applying deep reinforcement learning (DRL) to financial trading is handling the non-stationary nature of markets while maintaining robustness, interpretability, and realistic risk control. Market regimes change frequently, trading decisions must remain explainable, and naive DRL models often fail under real-world constraints such as transaction costs and drawdowns.

This work provides a practical DRL-based trading solution that addresses these challenges by combining adaptive learning, explainability, and structured risk management. The proposed PPO-based framework continuously adapts to recent market behavior through periodic retraining, improves transparency using post-hoc explanations, and ensures stability via explicit risk-aware constraints. Together, these components enable a balanced trading system that delivers competitive returns while remaining interpretable and operationally reliable.

## 7.2 Future Work

Future extensions of this work include several key directions. The framework can be expanded to multi-asset trading covering futures, options, and cryptocurrencies with asset-specific reward models. Performance should be validated during historical crisis periods (2008, 2020) through extended backtesting to assess robustness under extreme market stress. Algorithm diversity can be enhanced by exploring ensemble methods combining multiple DRL agents with dynamic weighting. The fine-tuning mechanism requires longer evaluation periods to assess long-term stability and potential overfitting. Computational efficiency can be improved through pipeline optimization, parallel processing, and lightweight explainability alternatives. The state space architecture should be generalized to support variable numbers of assets and different timeframes, enabling transfer learning to new markets. Transaction cost models can be made adaptive to reflect actual costs for institutional versus retail traders. Finally, additional risk management mechanisms including volatility targeting, regime detection, and correlation-aware diversification would strengthen downside protection. These extensions would address current limitations while moving the framework toward a comprehensive institution-ready automated trading system.

# References

[1] X.-Y. Liu, H. Yang, Q. Chen, R. Zhang, L. Yang, B. Xiao, and C. D. Wang, "FinRL: A Deep Reinforcement Learning Library for Automated Stock Trading in Quantitative Finance," *Deep Reinforcement Learning Workshop, NeurIPS 2020*, 2020.

[2] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[3] S. M. Lundberg and S.-I. Lee, "A Unified Approach to Interpreting Model Predictions," *Advances in Neural Information Processing Systems 30*, pp. 4765-4774, 2017.

[4] M. T. Ribeiro, S. Singh, and C. Guestrin, "Why Should I Trust You?: Explaining the Predictions of Any Classifier," *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1135-1144, 2016.

[5] J. Moody and M. Saffell, "Learning to Trade via Direct Reinforcement," *IEEE Transactions on Neural Networks*, vol. 12, no. 4, pp. 875-889, 2001.

[6] H. Yang, X.-Y. Liu, S. Zhong, and A. Walid, "Deep Reinforcement Learning for Automated Stock Trading: An Ensemble Strategy," *ACM International Conference on AI in Finance (ICAIF)*, 2020.

[7] Z. Zhang, S. Zohren, and S. Roberts, "Deep Reinforcement Learning for Trading," *The Journal of Financial Data Science*, vol. 2, no. 2, pp. 25-40, 2020.

[8] X. Li, Y. Li, Y. Zhan, and X.-Y. Liu, "Optimistic Bull or Pessimistic Bear: Adaptive Deep Reinforcement Learning for Stock Portfolio Allocation," *ICML Workshop on Applications and Infrastructure for Multi-Agent Learning*, 2019.

[9] M. Kritzman and Y. Li, "Skulls, Financial Turbulence, and Risk Management," *Financial Analysts Journal*, vol. 66, no. 5, pp. 30-41, 2010.

[10] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-Dimensional Continuous Control Using Generalized Advantage Estimation," *arXiv preprint arXiv:1506.02438*, 2015.

[11] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-Baselines3: Reliable Reinforcement Learning Implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1-8, 2021.

[12] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," *arXiv preprint arXiv:1606.01540*, 2016.

[13] Z. Jiang, D. Xu, and J. Liang, "A Deep Reinforcement Learning Framework for the Financial Portfolio Management Problem," *arXiv preprint arXiv:1706.10059*, 2017.

[14] Y. Deng, F. Bao, Y. Kong, Z. Ren, and Q. Dai, "Deep Direct Reinforcement Learning for Financial Signal Representation and Trading," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 3, pp. 653-664, 2017.

[15] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al., "Human-Level Control through Deep Reinforcement Learning," *Nature*, vol. 518, no. 7540, pp. 529-533, 2015.

[16] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor," *International Conference on Machine Learning*, pp. 1861-1870, 2018.

[17] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing Function Approximation Error in Actor-Critic Methods," *International Conference on Machine Learning*, pp. 1587-1596, 2018.

[18] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al., "Mastering the Game of Go with Deep Neural Networks and Tree Search," *Nature*, vol. 529, no. 7587, pp. 484-489, 2016.

# A Hyperparameters and Configuration

## A.1 Complete Hyperparameter List

| Parameter | Value | Description |
|---|---|---|
| **PPO Algorithm** | | |
| Rollout steps ($n_{\text{steps}}$) | 2048 | Steps collected per update |
| Learning rate | $1 \times 10^{-4}$ | Initial training learning rate |
| Fine-tune learning rate | $1 \times 10^{-5}$ | Reduced LR for fine-tuning |
| Discount factor ($\gamma$) | 0.99 | Future reward discounting |
| GAE lambda ($\lambda$) | 0.95 | Advantage estimation |
| Clip range ($\epsilon$) | 0.2 | Policy update constraint |
| Value function coef ($c_v$) | 0.4 | Loss function weight |
| Entropy coefficient | 0.005 | Exploration bonus |
| Batch size | 256 | Training minibatch size |
| Epochs per update | 10 | Gradient update iterations |
| Max gradient norm | 0.5 | Gradient clipping threshold |
| Normalize advantage | True | Standardize advantage estimates |
| **Trading Environment** | | |
| Number of stocks | 30 | 30 US equities |
| State dimension | 301 | Total state features |
| Action dimension | 30 | Actions per stock |
| Action range | [-1, 1] | Action for each stock |
| Initial cash | \$1,000,000 | Initial cash |
| Max stock per trade | 100 shares | Position limit |
| Transaction cost | 0.1% | Cost per trade |
| Time interval | 1 minute | Trading frequency |
| **Fine-Tuning** | | |
| Fine-tune interval | 2 hours | Retraining frequency |
| Lookback window | 48 hours | Training data window |
| Train/val split | 80%/20% | Data split ratio |
| Fine-tune steps | 2000 | Training iterations |
| Acceptance threshold | 95% | Validation performance |
| **Risk Management** | | |
| Turbulence threshold | 500 | Volatility cutoff |
| Square-off time | 3:45 PM ET | Position closing time |

Table 7: Complete hyperparameter configuration

### A.2 Network Architecture Details

| Module | Layer | Dimensions | Activation |
|---|---|---|---|
| Feature extractor | Flatten | $301 \to 301$ | – |
| Policy network | Linear | $301 \to 64$ | Tanh |
| Policy network | Linear | $64 \to 64$ | Tanh |
| Action mean head | Linear | $64 \to 30$ | Linear |
| Action log std | Learnable parameter | $\mathbb{R}^{30}$ | – |
| Value network | Linear | $301 \to 64$ | Tanh |
| Value network | Linear | $64 \to 64$ | Tanh |
| Value head | Linear | $64 \to 1$ | Linear |

Table 8: PPO actor–critic neural network architecture implemented using Stable-Baselines3 `ActorCriticPolicy`.

## B Technical Indicators Formulas

### B.1 MACD (Moving Average Convergence Divergence)

$$\text{MACD}_t = \text{EMA}_{12}(p_t) - \text{EMA}_{26}(p_t) \tag{15}$$

$$\text{Signal}_t = \text{EMA}_9(\text{MACD}_t) \tag{16}$$

where $\text{EMA}_n$ is the $n$-period exponential moving average and $p_t$ is the closing price.

### B.2 RSI (Relative Strength Index)

$$\text{RSI}_t = 100 - \frac{100}{1 + RS_t} \tag{17}$$

$$RS_t = \frac{\text{Average Gain over } n \text{ periods}}{\text{Average Loss over } n \text{ periods}} \tag{18}$$

Typically $n = 14$ days. RSI values above 70 indicate overbought conditions, below 30 indicate oversold.

### B.3 CCI (Commodity Channel Index)

$$\text{CCI}_t = \frac{TP_t - \text{SMA}_{20}(TP_t)}{0.015 \times \text{Mean Deviation}} \tag{19}$$

where $TP_t = \frac{\text{High}_t + \text{Low}_t + \text{Close}_t}{3}$ is the typical price.

### B.4 DX (Directional Index)

$$\text{DX}_t = 100 \times \frac{|+DI_t - (-DI_t)|}{+DI_t + (-DI_t)} \tag{20}$$

where $+DI$ and $-DI$ are positive and negative directional indicators derived from price highs and lows.

## B.5 Simple Moving Averages

$$\mathrm{SMA}_n(p_t) = \frac{1}{n}\sum_{i=0}^{n-1} p_{t-i} \tag{21}$$

We compute both 30-day and 60-day SMAs to capture medium and longer-term trends.

## B.6 Bollinger Bands

Let the middle band be the $n$-period simple moving average, $\mathrm{MB}_t = \mathrm{SMA}_n(p_t)$. Define the rolling standard deviation

$$\sigma_t = \sqrt{\frac{1}{n}\sum_{i=0}^{n-1} (p_{t-i} - \mathrm{MB}_t)^2}. \tag{22}$$

Then the upper and lower Bollinger Bands are

$$\mathrm{boll\_ub}_t = \mathrm{MB}_t + k\,\sigma_t, \tag{23}$$

$$\mathrm{boll\_lb}_t = \mathrm{MB}_t - k\,\sigma_t, \tag{24}$$

where typically $n = 20$ and $k = 2$.

## B.7 Keltner Channels

Keltner Channels consist of a center line (typically an exponential moving average) and upper/lower bands based on the Average True Range (ATR). Let the center line be

$$\mathrm{KC}_{\mathrm{mid},t} = \mathrm{EMA}_n(p_t), \tag{25}$$

where $p_t$ is the closing price and $\mathrm{EMA}_n$ is the $n$-period exponential moving average. Define the True Range (TR) as

$$\mathrm{TR}_t = \max\Big(\mathrm{High}_t - \mathrm{Low}_t,\ \big|\mathrm{High}_t - \mathrm{Close}_{t-1}\big|,\ \big|\mathrm{Low}_t - \mathrm{Close}_{t-1}\big|\Big), \tag{26}$$

and the Average True Range as

$$\mathrm{ATR}_t = \mathrm{EMA}_n(\mathrm{TR}_t). \tag{27}$$

Then the upper and lower Keltner Channels are

$$\mathrm{KC}_{\mathrm{ub},t} = \mathrm{KC}_{\mathrm{mid},t} + m\,\mathrm{ATR}_t, \tag{28}$$

$$\mathrm{KC}_{\mathrm{lb},t} = \mathrm{KC}_{\mathrm{mid},t} - m\,\mathrm{ATR}_t, \tag{29}$$

where typically $n = 20$ and $m = 2$.