POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Fintech - Machine Learning Project

Computer Science and Engineering - EIT Digital Master in Fintech

Giacomo Falcone, 271005

Abstract: This project developed an intelligent algorithmic trading agent for volatile cryptocurrency markets (Bitcoin, Ether) using a Reinforcement Learning (RL) approach with the Proximal Policy Optimization (PPO) algorithm in a custom-simulated environment. The agent's decision-making was informed by historical market data, engineered technical indicators, and, crucially in its final iteration, its own portfolio status for an improved state representation. Iterative refinements to the action space (evolving to support long, short, and hold actions), observation features, and a reward function centered on optimizing log returns of net worth, culminated in a model demonstrating enhanced trading performance and adaptability. Experiments showed these improvements, particularly the inclusion of portfolio state, were effective across both Bitcoin and Ethereum datasets compared to initial configurations. This work underscores PPO's potential for cryptocurrency trading and the importance of iterative design in developing robust financial RL agents.

Key-words: Reinforcement Learning, Algorithmic Trading, Proximal Policy Optimization (PPO), Cryptocurrency, Bitcoin, Ethereum, Financial Time Series, Machine Learning

1. Introduction

The financial markets present a complex and dynamic environment where participants continuously seek strategies to maximize returns while managing inherent risks. Algorithmic trading, the use of computer programs to execute trading strategies, has become increasingly prevalent, offering the potential for speed, efficiency, and the systematic exploitation of market opportunities. However, developing consistently profitable and risk-aware trading algorithms remains a significant challenge due to market volatility, noise, and the non-stationary nature of financial data. Furthermore, many sophisticated machine learning models, while potentially powerful, suffer from a lack of transparency, often referred to as the "black box" problem, making their decision-making processes difficult to interpret and trust.

This project addresses the challenge of developing an intelligent trading agent capable of making optimal decisions in the cryptocurrency market. The primary focus will be on Bitcoin, with subsequent validation of the model's robustness and generalizability on Ether. The core problem lies in navigating the trade-offs between potential profit and exposure to risk under the highly volatile conditions characteristic of crypto assets.

To tackle this, we propose the development of a reinforcement learning (RL) model. Reinforcement learning offers a powerful paradigm where an agent learns to make a sequence of decisions by interacting with an environment and receiving feedback in the form of rewards or penalties. Specifically, this project will explore the application of a Proximal Policy Optimization (PPO) algorithm, or a similar advanced policy-based RL technique. The agent will be trained using historical market data, including price, volume, and various technical indicators, to learn a robust trading policy.

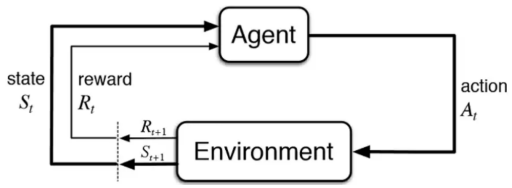
The primary objective is to train an RL agent that can autonomously decide whether to buy, sell, or hold a financial asset. The agent's performance will be evaluated based on its ability to generate profit. Key goals include achieving consistent profitability over various market conditions. This report will detail the methodology, implementation, training process, experimental results, and conclusions drawn from this investigation.

2. Background

This section provides a brief overview of the core concepts essential for understanding the methodologies and context of this project. The discussion encompasses the fundamentals of reinforcement learning, the Markov Decision Process framework, the Proximal Policy Optimization algorithm, and key financial terms relevant to automated trading systems.

2.1. Reinforcement Learning

Reinforcement learning focuses on determining an optimal strategy for making a sequence of decisions rather than a single one. The environment is typically dynamic and unpredictable, requiring the agent to adapt its actions over time. The **agent** explore the environment by following the **policy**. The goal is to modify the policy in order to **maximize the reward** (the reward can change, it doesn't have to be always the same).



The figure shows a typical agent-environment loop. A key aspect is balancing **exploration** (discovering new actions) with **exploitation** (using known good actions). RL is suitable when labeled data isn't available, as the agent learns from experience.

In reinforcement learning, environments are typically modeled as **Markov Decision Processes** (MDPs). An MDP assumes that the future evolution of the environment depends only on the current state and the action taken,not on the full history of prior states or actions. This assumption, known as the **Markov property**, is essential for guaranteeing theoretical convergence of RL algorithms and learning an optimal policy.

2.2. Proximal Policy Optimization

Proximal Policy Optimization (PPO) is an advanced RL algorithm used in this project. It's known for its stability and strong performance by iteratively improving the agent's policy without making excessively large changes, making it effective for complex tasks like financial trading.

2.3. Key Financial Concepts

- **Long** position (buy): buying an asset expecting its price to rise.
- **Short** position (short sell): selling a "borrowed" asset expecting its price to fall, then buying it back cheaper
- **Hold**: maintaining the current position or no position.
- **Return**: the profit or loss from trading activities.
- **Risk**: the potential for financial loss, often associated with market volatility.
- **Technical indicators**: Calculations derived from historical price and volume data, designed to help identify trading opportunities. Indicators used:
  - *Simple Moving Averages (SMA)*: To smooth price data and identify trend directions over various timeframes.
  - *Relative Strength Index (RSI)*: A momentum indicator used to gauge overbought or oversold conditions.
  - *Volatility Measures*: To quantify the degree of recent price fluctuations and assess market risk.
  - *Price Positional Metrics*: Features that describe the current price relative to historical highs, lows, or moving averages, providing context to its current level.

2.4. Cryptocurrency Overview

The primary cryptocurrencies examined in this project are Bitcoin (BTC) and Ether (ETH). As two of the largest digital assets by market capitalization, they garner significant attention within financial markets. A key characteristic of both BTC and ETH is their notable price volatility. While this volatility introduces inherent trading risks, it concurrently creates frequent opportunities, making them compelling and challenging subjects for the development and evaluation of the algorithmic trading strategies explored in this work.

3. Related Work

Reinforcement learning has shown strong potential in cryptocurrency trading, particularly with Proximal Policy Optimization (PPO) due to its stability and efficiency. Liu et al. employed PPO for high-frequency Bitcoin trading, demonstrating its ability to adapt to market dynamics and outperform traditional strategies [9].

Mahayana et al. enhanced PPO-based agents by integrating technical indicators and trend analysis, achieving competitive results on crypto markets [11]. Zhang et al. emphasized the importance of volatility-aware rewards in RL-based trading strategies across multiple asset classes, including cryptocurrencies [19].

Building on these studies, the current project implements a PPO agent in a custom Gym [17] environment designed for Bitcoin trading, supporting both long and short positions and using historical and technical market data as state input [8].

4. Method

In this section there is a description of the method adopted for this project.

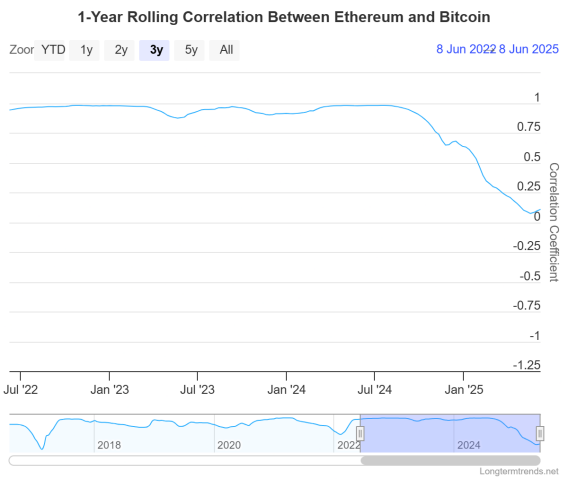
4.1. Data Collection

For this project, historical cryptocurrency data spanning from June 1, 2022, to April 30, 2025, were utilized. The data sources were CoinGecko, for volume data [3], and MarketWatch, for daily price data [12].

Feature	Description
Date	The specific date for which the data is recorded.
Open	The opening price of the asset on that date.
High	The highest price the asset reached on that date.
Low	The lowest price the asset reached on that date.
Close	The closing price of the asset on that date.
Volume	The total quantity of the asset traded on that date.

Bitcoin (BTC) data were principally used for training and validating the trading model. This dataset was divided chronologically: the initial 80% of the period was allocated for training, while the remaining 20% was used for validation.

To assess the model's generalization capabilities on a different but correlated asset, and to avoid testing on periods that might overlap with Bitcoin's training phase, Ethereum (ETH) data [4] [13] were employed for the testing phase. The ETH data used for testing corresponded to the same chronological period as the Bitcoin validation set, ensuring a distinct dataset for final performance evaluation, because (as shown in the graph below) ether and bitcoin are highly correlated (especially in the training period).



EHT-BTC correlation (source: [10])

4.2. Data Inspection

Initial pre-processing of the raw OHLCV (Open, High, Low, Close, Volume) data focused on ensuring its quality and integrity for subsequent analysis. A key step involved carefully concatenating multiple yearly data extracts, primarily to form the complete dataset spanning the required period. During this manual assembly, meticulous checks were performed to ensure chronological continuity and to remove any potential duplicate entries that might arise from overlapping yearly segments. A single missing day's data, an artifact of this stitching process, was identified and this was rectified by ensuring correct alignment and complete data merging from the source files. Beyond confirming this seamless concatenation, the datasets were found to be largely clean and did not exhibit other significant issues regarding missing values (NaNs) or erroneous (e.g., zero) entries in price or volume fields, thus providing a solid foundation for feature engineering.

4.3. Data Pre-Processing

To enhance the information available to the reinforcement learning agent beyond raw price and volume data, a set of technical indicators and price-derived features was engineered. These features were calculated using the historical OHLCV data and are intended to provide the agent with insights into market trends, momentum, volatility, and relative price positioning. The features included are:

- **Simple Moving Averages (SMA):** Both a short-term (12-period) SMA and a long-term (200-period) SMA were calculated. These are used to help the agent identify the direction and strength of short-term versus long-term trends.
- **Relative Strength Index (RSI):** A 14-period RSI was incorporated as a momentum oscillator. Its purpose is to indicate potential overbought or oversold conditions by measuring the speed and change of price movements.
- **Distance from SMA 200:** This feature was calculated as

$$\frac{\text{close\_price} - \text{sma\_200}}{\text{sma\_200}}$$

  It explicitly quantifies the current price's normalized deviation from its long-term average, helping the agent assess how extended the price might be.
- **Volatility (100-period):** Calculated as the standard deviation of daily returns over the preceding 100 periods. This provides the agent with a measure of recent market risk and price fluctuation.
- **Position in 200-Range:** This feature, calculated as

$$\frac{\text{close\_price} - \text{min\_200\_periods}}{\text{max\_200\_periods} - \text{min\_200\_periods}}$$

  normalizes the current closing price relative to its highest and lowest values over the past 200 periods. It helps the agent understand if the current price is near recent highs or lows.

The calculation of these features, particularly those requiring longer look-back periods such as the 200-period SMA and the range-based metrics, inherently results in Not-a-Number (NaN) values for the initial segment of the dataset. Therefore, to ensure that the agent is trained on a complete and valid feature set, the first 200 data points of the processed data, corresponding to the longest indicator warm-up period, were subsequently removed. This step provides a clean dataset where all engineered features have valid numerical values for model training and evaluation.

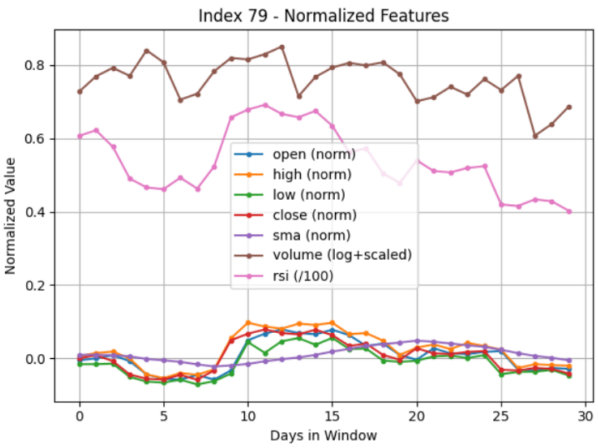
4.4. Feature Analysis

Further examination of the engineered features included an analysis of their variance. Features exhibiting **zero variance** (i.e., holding a single constant value across all data points) were checked for and confirmed to be absent, as such features provide no informational value. Consideration was also given to **near-zero variance predictors**: features with minimal variation that may not significantly contribute to model performance. Applying a variance threshold of 0.01, the volatility_100 feature was identified as having near-zero variance in the processed dataset for the observed period. While the specific factors contributing to this low variance were not explored in depth, the volatility_100 feature was retained for the first part of experimentation and was removed in the final set without showing big improvements. The reinforcement learning agent, particularly with a neural network architecture, could learn to assign appropriate importance to features, potentially still deriving value from it in specific contexts or learning to ignore it if non-informative.

An analysis of the engineered features in the dataset revealed, as is common for financial time series, high correlations among OHLC (Open, High, Low, Close) prices and many of their derived indicators. While strong multicollinearity can be problematic for certain statistical models, Reinforcement Learning agents like PPO, particularly those using neural network function approximators, are generally more robust to such inter-feature correlations. These models can learn to assign appropriate weights, effectively discerning unique signals or downplaying redundant information. The primary approach in this project was therefore to provide the agent with this comprehensive, albeit correlated, feature set, allowing it to learn from a rich representation of the market state. Supporting this decision, an experiment conducted towards the end of the project, aimed at reducing multicollinearity by removing the 'open' price and expressing 'high' and 'low' as percentage differences from the 'close' price, did not yield improvements in the agent's performance. This outcome suggests that the PPO agent was capable of effectively managing the information within the original, more correlated feature set for the majority of the development process.

4.5. Observation Space Design

A well-defined and **normalized observation** space is critical for the stable and effective training of the reinforcement learning agent. The observation provided to the agent at each timestep is a numerical vector constructed from recent market data (using a **30-day window**) and current long-term indicators. A comprehensive normalization strategy was applied to all features: price-based windowed features (Open, High, Low, Close, SMA) were scaled relative to the first `close` price within their window to represent relative changes. Windowed trading `volume` was log-transformed to manage skewness and then Min-Max scaled to a [0,1] range using a `MinMaxScaler`. The windowed `RSI` was also scaled to [0,1] by dividing its values by 100.



Finally, current day long-term features (`volatility_100`, `distance_from_sma_200`, and `position_in_200_range`) were standardized using `StandardScaler`s. Crucially, all scalers (`MinMaxScaler` for volume, `StandardScaler`s for long-term features) were **fitted exclusively on the training dataset** to prevent data leakage. These individually normalized components are then concatenated to form the final flat feature vector serving as the agent's input.

4.6. State Representation and MDPs

The observation provided to the agent consists of a normalized feature vector derived from a 30-day window of recent market data (Open, High, Low, Close, SMA, Volume, RSI) and current-day long-term indicators (`distance_from_sma_200`, `volatility_100`, `position_in_200_range`), as detailed previously. This was designed to summarize the market's recent dynamics.

However, the observation excludes key portfolio variables such as current asset holdings (e.g., amount of BTC held or shorted), entry prices of open positions and available cash or capital. While the full environment state includes this information, the agent's decision is based solely on market data.

This omission may introduce some limitations. In a realistic trading scenario, the optimal action often depends not only on the external market context but also

on the agent's internal portfolio state. For example, whether to sell may depend on whether the agent is currently holding an asset or not. Without access to this information, the agent sees identical market observations regardless of whether it holds a long position, has an open short, or is entirely flat, leading to potential **partial observability**. From a formal standpoint, this means that the agent's observation space does not fully satisfy the Markov property (as explained in *section 2.1*). While deep reinforcement learning algorithms like PPO can still perform reasonably well in partially observable settings, this design choice may limit the agent's ability to learn an optimal policy, especially in situations where action selection depends critically on unobserved internal state variables.

This problem was not directly addressed in the first phase of the project, but the final model address it (by adding the portfolio status to the observable state), and this showed an improvement in the evaluation.

4.7. Trading Environment Design

To train the reinforcement learning agent, a custom trading environment was developed, adhering to the OpenAI Gym interface [2]. Initial explorations involved assessing pre-existing trading environments such as `gym-anytrading` [1] and `Gym Trading Env` [16]. However, to achieve greater control over feature engineering, state representation, action space definition, and the specifics of the reward function tailored to this project's objectives, the decision was made to develop a custom environment. This custom environment simulates the interactions of an agent within a cryptocurrency market, using historical data to provide market feedback. The objective is to make the agent learn effective trading strategies. This involves defining how the agent perceives the market, what actions it can take, how its performance is measured at each step, and the conditions under which a trading episode concludes.

State Representation

The state representation provided to the agent at each timestep is the normalized observation vector detailed in the preceding "Observation Space Design" section. This vector, comprising a 30-day window of market features and current long-term indicators, serves as the agent's complete view of the market for decision-making. Later was added also the portfolio status.

Action Space

The set of actions available to the agent evolved during the project to explore different trading strategy complexities. The primary configurations included:

- **Long-only** (buy, sell): Initially, the agent operated with a discrete action space consisting of two actions:
  - *Action 0 (buy):* If the agent has a cash balance, this action attempts to invest the entire available balance to purchase the asset (e.g., Bitcoin) at the current market price, establishing a long position. If already long (the balance is 0, then just hold the long position).
  - *Action 1 (sell):* If the agent is in a long position, this action attempts to sell all holdings at the current market price. If he is not in open long position, then this action does nothing.
- **Long and short** (buy, sell): Same discrete action space as before (two actions):
  - *Action 0 (buy):* If the agent has a cash balance, this action attempts to invest the entire available balance to purchase the asset (e.g., Bitcoin) at the current market price, establishing a long position. If already in a short position, this action attempts to cover (buy back) the shorted asset.
  - *Action 1 (sell):* If the agent is holding the asset (in a long position), this action attempts to sell all holdings at the current market price. If the agent is not in a long position and not already short, this action attempts to open a new short position, effectively selling an amount of the asset equivalent to a portion (e.g., half) of the current cash balance.
- **Adding** explicit **hold** action (buy, sell, hold): In later experiments (to be detailed in *section 5*), the action space was expanded to three discrete actions: buy, sell (with similar logic to above for initiating/closing long/short positions), and an explicit *action 2 (hold)*, where the agent consciously decides to maintain its current position (long, short, or flat) without executing a trade.

The environment logic ensures that actions are executed based on the agent's current portfolio status (e.g., preventing buying without sufficient balance or selling an asset not held, unless initiating a short).

Reward Function

The reward function is critical for guiding the agent's learning process. In this environment, the primary reward signal provided to the agent at each step (t) is based on the percentage change in its net worth relative to the initial balance:

$$\text{Reward}_t = \frac{\text{NetWorth}_t - \text{NetWorth}_{t-1}}{\text{InitialBalance}} \times 100$$

This provides a continuous, per-step feedback signal reflecting the immediate financial outcome of the agent's decisions and market movements. Additionally, smaller, event-based rewards and penalties were incorporated to further shape behavior:

- A small positive reward (**action_reward**) is given whenever a trade (buy or sell leading to a change in position) is executed, encouraging activity.
- When closing a long or short position, the realized **profit or loss** percentage (amplified) is added to the reward.
- A small penalty (**holding_penalty_pct**) is applied if the agent is holding a position (long or short) during periods identified as having high price volatility, discouraging passive holding during potentially risky (but also potentially profitable) market conditions.

The agent's objective is to learn a policy that maximizes the cumulative sum of these rewards over an episode. Transaction costs, while an important real-world factor, were initially set to zero (transaction_cost_pct=0.0) for the primary development phase to simplify the learning problem, with their inclusion considered for future work.

Episode Definition

Each trading episode is defined by the following conditions:

- **Start:** An episode begins with the agent having an initial cash balance (e.g., 10,000 USD) and no asset holdings or short positions. The simulation starts after the initial window_size period of the dataset to ensure a full historical window is available for the first observation.
- **End Conditions:** An episode terminates if either of the following occurs:
  - The agent reaches the end of the available historical data for the current dataset (BTC or ETH).
  - The agent's net worth falls below a predefined threshold, specifically 10% of the initial balance, indicating a significant loss and triggering a "ruin" condition.

4.8. Reinforcement Learning Agent

The core of the trading system is a Reinforcement Learning (RL) agent tasked with learning an optimal trading policy. This section details the choice of RL algorithm and the specific configuration of the agent model.

Algorithm Choice: PPO

For this project, the Proximal Policy Optimization (PPO) algorithm [15] was selected. PPO is an advanced policy gradient method known for its strong empirical performance across a variety of challenging RL tasks, including those with continuous or discrete action spaces. Key advantages that motivated its selection include its relative ease of implementation and tuning, good sample efficiency compared to simpler policy gradient methods, and its mechanism for stabilizing training by limiting the size of policy updates through a clipped surrogate objective function. This makes PPO a robust choice for navigating the complexities and potential instabilities of financial market environments. The implementation from the Stable Baselines3 library [14] was utilized.

Model Architecture

The PPO agent employed an actor-critic architecture using a Multi-Layer Perceptron policy, specified as `"MlpPolicy"` in Stable Baselines3. To tailor the network capacity to the task, a custom network architecture was defined for both the policy (actor) and value (critic) functions. As specified in the `policy_kwargs` parameter, the configuration was:

- **Policy Network (`pi`):** Two hidden layers, each with 64 units.
- **Value Function Network (`vf`):** Two hidden layers, each with 128 units.

Standard activation functions (typically ReLU for hidden layers) are used by default within the `MlpPolicy` structure.

Key Hyperparameters

Several key hyperparameters influencing the PPO agent's learning process were configured:

- **Learning Rate Schedule:** Various learning rate strategies were explored, including a fixed learning rate and a linear decay schedule. Ultimately, a custom quadratically decaying schedule, analogous to the `faster_schedule` function described in the project's codebase, was selected for its effectiveness. This schedule initiated the learning rate at 0.001 (`initial_learning_rate`) and reduced it with a decay power of 2.0 as training progressed, down to a minimum of 1×10^{-6} (`minimum_learning_rate`). This approach facilitates larger updates early in training when the policy is rapidly evolving, transitioning to smaller, more stable updates as the policy nears convergence.
- **Other PPO Parameters:** Standard PPO parameters such as `n_steps` (number of steps to run for each environment per update), `batch_size` (minibatch size for PPO updates), `ent_coef` (entropy coefficient for exploration), `vf_coef` (value function loss coefficient), and `gae_lambda` (GAE lambda for advantage estimation) were initially considered for tuning. However were primarily used default values from the Stable Baselines3 library.
- The agent was trained for a total of 1,000,000 to 3,000,000 (`TOTAL_TIMESTEPS`) interactions with the environment.

5. Experiments and Results

This section outlines the setup used for training and evaluating the PPO trading agent, including implementation details, agent training parameters, and the metrics used for performance assessment. It also describes the specific experimental scenarios designed to test the agent's capabilities.

5.1. Implementation Details

The project was implemented in Python (version 3.11.7). Key libraries included Pandas [18] for data manipulation, NumPy [6] for numerical operations, Matplotlib [7] for plotting, Stable Baselines3 [14] for the PPO algorithm implementation, Gymnasium [17], the maintained successor to OpenAI Gym, for structuring the custom trading environment, and Finta [5] for the calculation of technical indicators.

Initial explorations involved `gym-anytrading` [1] and `Gym Trading Env` [16] to understand basic RL agent interactions in trading contexts. However, to achieve the specific feature engineering, state representation, action space design, and reward function control re-

quired for this project's objectives, a custom environment was subsequently developed and used for all main experiments. The experiments were conducted using this custom trading environment detailed in *Section 4.7*, which simulates cryptocurrency market interactions.

5.2. Training Configuration

This subsection details the training configuration of the PPO agent, which builds upon the foundational agent characteristics described in *Section 4.8*. The agent employed an `MlpPolicy` featuring a custom network architecture composed of two hidden layers with 64 units for the policy network, and two hidden layers with 128 units for the value network. The specific hyperparameters guiding this training process were determined through iterative experimentation.

For instance, several learning rate strategies were explored. A fixed learning rate and a linear decay schedule were tested, but a custom quadratically decaying schedule was ultimately implemented as it demonstrated a better balance of initial convergence speed and later training stability. This preferred schedule started the learning rate at an initial value of 0.001, which then decayed with a power of 2.0 as training progressed, down to a minimum of 1×10^{-6}.

Regarding other core PPO parameters such as `n_steps`, `batch_size`, `ent_coef`, `vf_coef`, and `gae_lambda`, various settings were initially considered. However, systematic tuning of these parameters did not yield consistent performance benefits over the default values provided by the Stable Baselines3 library, which were therefore largely retained for the final models. This allowed experimentation to focus on more impactful aspects like the learning rate and total training duration.

The total number of training timesteps proved to be a critical factor. It was empirically observed that training for at least one million timesteps was generally necessary for the agent to develop meaningful and active trading policies; shorter durations often resulted in agents exhibiting passive behavior or failing to adequately explore the action space. Consequently, all final training runs were conducted for a total of 1,000,000 - 3,000,000 timesteps.

Training was performed on the first 80% of the Bitcoin (BTC) dataset, with subsequent evaluation on the remaining 20% (validation set) and on a separate Ethereum (ETH) dataset for generalization testing, as detailed in *Section 4.1*.

5.3. Evaluation Metrics

The performance of the trained trading agent was primarily evaluated based on its ability to generate returns over the specified trading period. This assessment was chiefly conducted by calculating the **Total Return** (or Cumulative Profit) and through the visual analysis of the agent's net worth progression over time, as illustrated by performance graphs presented in the Results section.

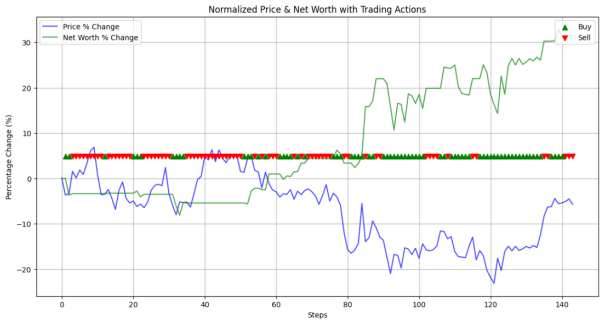
Total Return quantifies the overall percentage gain/loss generated by the agent and is calculated as (where NW stays for Net Worth):

$$\text{Total Return} = \left(\frac{\text{Final NW} - \text{Initial NW}}{\text{Initial NW}} \right) \times 100\%$$

5.4. Experiment 1: Long-Only

An initial baseline experiment was conducted using the PPO agent restricted to long-only trading. Within the custom environment, the agent's action space consisted of Buy, Sell, and Hold actions, and it received the standard normalized observation vector (detailed in *Section 4.5*). The agent was trained for 300,000 timesteps on the Bitcoin (BTC) training dataset, primarily guided by a reward function emphasizing realized trade profits and per-step changes in net worth, alongside minor incentives for trading and penalties for holding during high volatility.

Performance evaluation on the BTC validation set indicated that the agent could learn effective trading strategies for this asset. However, when subsequently tested on an Ethereum (ETH) dataset for generalization (using an ETH test period distinct from the BTC training data), the agent's effectiveness was significantly diminished, highlighting challenges with cross-asset applicability for this initial, simpler model configuration and training duration.



BTC evaluation set

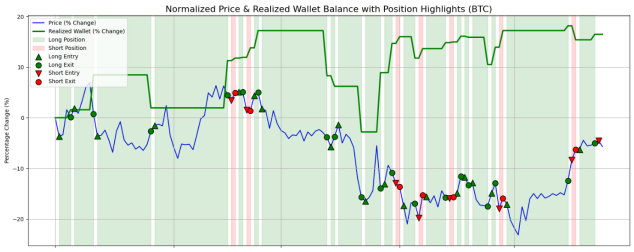
ETH evaluation set

5.5. Experiment 2: Adding Short

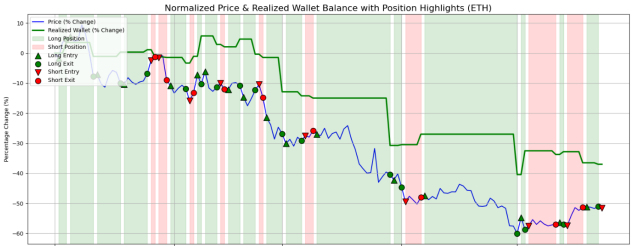
This experiment extended the long-only model by incorporating short-selling capabilities, allowing the

agent to profit from declining markets. The agent retained a two-action discrete space: Action 0 (Buy) closes any short position or opens a long position, while Action 1 (Sell) closes a long position or initiates a short position using 50% of available capital. The reward function was updated to reflect gains or losses from short trades and to penalize holding shorts during high volatility.

On the Bitcoin (BTC) validation set, performance was comparable to the long-only model. However, on an Ethereum (ETH) test set, the agent underperformed, indicating that improvements in market signal integration and dynamic position sizing are needed for effective short-selling.



BTC validation set

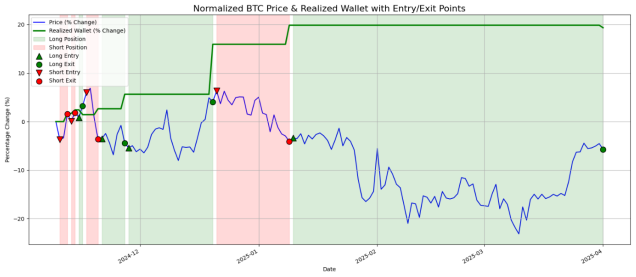


ETH testing set

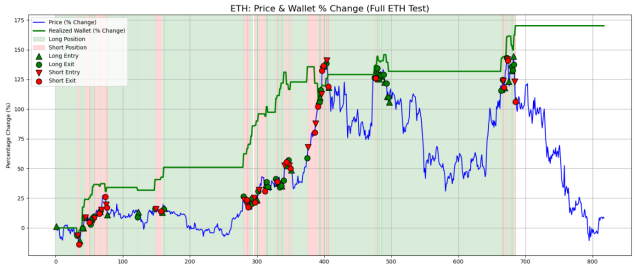
5.6. Experiment 3: Final Model

The final agent configuration emerged from an iterative development process that involved adjustments to the action space, reward function, and the set of features provided to the agent. This refined model aimed to incorporate insights and address limitations identified in earlier experiments.

Key characteristics differentiating this final setup included an enhanced observation space. Notably, normalized portfolio status (current holdings, balance, and market position) was added to provide the agent with more complete state information. Price feature representation was also modified, for instance, by expressing `high` and `low` prices relative to the `close` price, and a new intra-window feature correlation metric was introduced. Concurrently, the `volatility_100` feature was excluded from the long-term indicators. The agent operated with a three-action space (Buy, Sell, Hold) designed to support both long and short trading strategies. The reward function was updated to primarily utilize the log returns of the portfolio's net worth, supplemented by a minor, consistent incentive for any action taken. This final model was trained for 2,000,000 timesteps on the Bitcoin (BTC) training data, employing the PPO algorithm, network architecture, and learning rate schedule as detailed in the general training configuration (*Section 5.2*), to evaluate the cumulative impact of these design improvements. Evaluation on both Bitcoin and Ethereum datasets demonstrated that these enhancements allowed the agent to better adapt to varying market conditions and improved its overall trading performance.



BTC validation set



The figure shows evaluation on ETH also on the period used for training (on BTC). The performance are good, but also the correlation with BTC was high in that period. At the end of the graph there is visible the need for a stop loss (difficult to see it in the graph, but there is a long position open).

6. Conclusions

This project successfully developed an adaptive PPO-based cryptocurrency trading agent for Bitcoin and Ether through iterative custom environment design. The final model's improved performance and adaptability critically hinged on incorporating real-time portfolio status into its observation—addressing partial observability, supported by refined action spaces (long/short/hold) and a reward function based on log returns of net worth. While this work validates PPO's efficacy and the iterative design approach for this domain, the study's use of simplified market simulations (e.g., regarding transaction costs and risk tools like stop-losses) highlights essential future work focused on enhancing realism and integrating more robust risk controls. Ultimately, this project underscores that comprehensive state representation and thoughtful environment design are paramount for developing effective Reinforcement Learning trading agents.

References

[1] AminHP. gym-anytrading: An OpenAI Gym trading environment for any trading source - Forex, Stocks, Crypto, etc. https://github.com/AminHP/gym-anytrading, 2023. Accessed: 2025-05-07.

[2] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016. URL: https://arxiv.org/abs/1606.01540.

[3] CoinGecko. Bitcoin Historical Data (Volume). https://www.coingecko.com/it/monete/bitcoin/historical_data?start=2025-01-01&end=2025-05-26, 2025. Accessed: 2025-05-03.

[4] CoinGecko. Ethereum Historical Data (Volume). https://www.coingecko.com/it/monete/ethereum/historical_data?start=2025-01-01&end=2025-05-30, 2025. Accessed: 2025-05-12.

[5] EMP FinTech. finta: Common financial technical indicators implemented in Pandas. https://github.com/peerchemist/finta, 2023. Accessed: 2025-05-10.

[6] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tiziano Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, 2020. URL: https://doi.org/10.1038/s41586-020-2649-2.

[7] John D. Hunter. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007. URL: https://doi.org/10.1109/MCSE.2007.55.

[8] Legfi. Reinforcement learning in stock market. https://github.com/Legfi/Reinforcement-learning-in-stock-market, 2022. Accessed: 2025-05-14.

[9] Yukun Liu, Fan Wang, and Zhiqiang Zhang. Bitcoin transaction strategy construction based on deep reinforcement learning. *Applied Soft Computing*, 110:107576, 2021. URL: https://www.sciencedirect.com/science/article/abs/pii/S1568494621008747.

[10] LongTermTrends. Ethereum vs Bitcoin - Which is a better investment? https://www.longtermtrends.net/ethereum-vs-bitcoin/, 2024. Accessed: 2025-06-01.

[11] Agung Mahayana et al. Combining deep reinforcement learning with technical analysis and trend monitoring on cryptocurrency markets. *Neural Computing and Applications*, 2023. URL: https://link.springer.com/article/10.1007/s00521-023-08516-x.

[12] MarketWatch. Bitcoin USD (BTCUSD) Historical Price Data. https://www.marketwatch.com/investing/cryptocurrency/btcusd/download-data?mod=mw_quote_tab, 2025. Accessed: 2025-05-03.

[13] MarketWatch. Ethereum USD (ETHUSD) Historical Price Data. https://www.marketwatch.com/investing/cryptocurrency/ethusd/download-data?startDate=4/28/2021&endDate=4/28/2022, 2025. Accessed: 2025-05-12.

[14] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL: http://jmlr.org/papers/v22/20-1364.html.

[15] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. URL: https://arxiv.org/abs/1707.06347.

[16] teddyqq. Gym Trading Env: A stock trading environment for OpenAI gym. https://github.com/teddyqq/gym-trading-env, 2023. Accessed: 2025-05-10.

[17] The Farama Foundation. Gymnasium: A standard API for reinforcement learning environments (a maintained fork of OpenAI Gym). https://gymnasium.farama.org/, 2024. Accessed: 2025-05-10.

[18] The Pandas Development Team. pandas-dev/pandas: Pandas. https://doi.org/10.5281/zenodo.3509134, March 2024. Accessed: 2025-05-07.

[19] Yuhang Zhang, Stefan Zohren, and Stephen Roberts. Deep reinforcement learning for trading. *arXiv preprint arXiv:1911.10107*, 2019. URL: https://arxiv.org/abs/1911.10107.