Cryptocurrency Price Movement Prediction Using A2C in Deep Reinforcement Learning

A project report submitted in partial fulfillment of the requirements for the award of a degree in

Masters of Science
Data Science

By
ABRAR AHMED
Regd No. 2023005662

Under the esteemed guidance of
Mrs. V.Satya Aruna

Department of Computer Science
GITAM School of Science
GITAM (Deemed to be University)
Visakhapatnam -530045, A.P
(2024-2025)

CERTIFICATE

This is to certify that the project entitled **"CRYPTOCURRENCY PRICE MOVEMENT PREDICTION"** is Bonafide work done by **ABRAR AHMED,** Regd.No:2023005662 during December 2024 to April 2025 in partial fulfilment of the requirement for the award of the degree of **Master of Science, Data Science** in the Department of Computer Science, GITAM School of Science, GITAM (Deemed to be University), Visakhapatnam.

Mrs. V.SATYA ARUNA
Assitant Professor
Dept. Of Computer Science

Dr. Prof. T. UMA DEVI
Head of Department
Dept. Of Computer Science

DECLARATION

I , (**ABRAR AHMED)** Regd. No:2023005662 hereby declare that the project entitled **"CRYPTOCURRENCY PRICE MOVEMENT PREDICTION" (Project Title)** is an original work done in the partial fulfilment of the requirements for the award of the degree of **Master of Science, Data Science** in **GITAM School of Science, GITAM (Deemed to be University), Visakhapatnam.** I assure that this project work has not been submitted towards any other degree or diploma in any other colleges or universities.

Abrar Ahmed
(Regd. No:2023005662)

ACKNOWLEDGMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without mentioning the people who made it possible and whose constant guidance and encouragement crown all the effects with success.

I would like to express my sincere gratitude to our honourable Principal and **Prof. K. Vedavathi**, GITAM School of Science, GITAM (Deemed to be University), for giving me an opportunity to work on this project.

I consider it as a privilege to express our deepest gratitude to **Prof. T. Uma Devi**, Head of the Department, Department of Computer Science for her valuable suggestions and constant motivation that greatly helped us to successfully complete the project work. I would like to thank my project guide **Mrs. V. Satya Aruna, Assistant Professor,** Dept. of Computer Science for her stimulating guidance and profuse assistance.

We would like to thank our Project Coordinator **Dr. Santosh Kumar Uppada, Assistant Professor and Mr. Md Nasir Assistant Professor** Dept. of Computer Science and in helping to complete the project by taking frequent reviews and for their valuable suggestions throughout the progress of this work.

I thank all the Teaching and Non-teaching Staff who has been a constant source of support and encouragement during the study tenure.

I thank all my friends who helped me in sharing their knowledge and support.

Abrar Ahmed
(Regd. No:2023005662)

ABSTRACT

Cryptocurrency markets are known for their extreme volatility and non-linear patterns, making price movement prediction a challenging task. This project presents a deep reinforcement learning approach using the **Advantage Actor-Critic (A2C) & LSTM (Long Short Term Memory)** algorithm to model intelligent trading strategies for Bitcoin. A custom OpenAI Gym environment simulates real-world trading, where the agent learns to make optimal decisions—Buy, Sell, or Hold—based on historical price data and technical indicators. Unlike conventional supervised models that focus on static predictions, our agent evolves by interacting with the environment, learning to maximize cumulative returns over time. The model is trained using historical BTC-USD data and evaluated through its ability to generate actionable trading signals, offering a practical application of deep learning in financial markets. This work demonstrates the potential of reinforcement learning in building adaptive and robust financial decision-making systems.

KEYWORDS

Deep Reinforcement Learning, Cryptocurrency Prediction, Advantage Actor-Critic (A2C), LSTM, Bitcoin (BTC), Trading Bot, OpenAI Gym, Financial Time Series, Intelligent Trading Agent, Buy-Sell-Hold Strategy, Algorithmic Trading.

CONTENTS

CHAPTERS **PG.NO.**

1. INTRODUCTION

Cryptocurrencies have rapidly emerged as one of the most volatile and high-potential financial assets in the global market. Bitcoin (BTC), the most prominent cryptocurrency, has shown explosive growth over the past decade, making it an attractive but risky investment. Predicting the price movement of such digital assets has drawn the attention of data scientists, quantitative analysts, and financial institutions. However, the dynamic and often irrational behavior of the crypto market makes accurate forecasting a significant challenge.

In this project, we propose an intelligent trading system powered by **Deep Reinforcement Learning (DRL)**, specifically utilizing the combination of **Advantage Actor-Critic (A2C)** and **LSTM** algorithm. The system is designed to simulate and learn optimal trading strategies for Bitcoin based on historical market data. Unlike traditional prediction models that treat each price movement as an independent event, DRL treats the problem as a sequential decision-making process, closely mimicking real-world trading dynamics.

Domain Description

This project lies at the intersection of multiple domains:

- **Finance**: Focusing on cryptocurrency trading, market analysis, and portfolio management.
- **Machine Learning**: Applying deep reinforcement learning to learn trading policies.
- **Artificial Intelligence**: Building agents capable of making intelligent, autonomous decisions.
- **Data Science**: Cleaning, transforming, and analyzing time-series data for predictive modeling.

The core idea is to empower trading agents with the ability to analyze patterns in historical price data and take actionable steps in real time—buy, sell, or hold—based on learned policies.

1.1. Problem Statement

Despite the abundance of historical price data, predicting the next move in the cryptocurrency market remains complex due to:

- High volatility and price fluctuations.
- Lack of central regulation and unexpected news impacts.
- Non-linear and chaotic patterns in financial time series.
- The need for strategies that adapt continuously to changing market conditions.

Traditional models like ARIMA or even machine learning methods like SVMs fail to adapt in real-time or lack the ability to optimize for long-term rewards. Our problem, therefore, becomes:

> "Can we build a reinforcement learning agent that interacts with the Bitcoin market, learns trading strategies from experience, and suggests whether to buy, sell, or hold at any given time?"

1.2. Background

Cryptocurrency trading has gained significant popularity, with Bitcoin being the most widely traded digital asset. Unlike traditional stock markets, Bitcoin operates in a highly volatile and decentralized environment, making price prediction a challenging task. Traditional trading strategies often struggle to adapt to sudden market fluctuations, leading to the growing adoption of AI-driven approaches. Deep Reinforcement Learning (DRL) has emerged as a powerful technique for decision-making in financial markets. Unlike conventional models, DRL-based systems continuously learn from market trends, improving their strategies over time. This project leverages the Actor2Critic (A2C) & LSTM algorithm to predict Bitcoin price movements

and determine optimal buy, sell, or hold actions. A2C, an advanced reinforcement learning method, balances exploration and exploitation, ensuring more stable and adaptive trading strategies by creating an environment whereas the LSTM will create the neural network to handle the time series input. By integrating historical market data, the model refines its decision-making process, aiming to maximize returns while minimizing risks. This AI-driven approach provides a data-backed alternative to emotional or speculative trading, enhancing efficiency in cryptocurrency investments.

1.3. Objectives of the Study

- To develop a custom reinforcement learning environment that simulates real-world cryptocurrency trading using historical Bitcoin data.
- To implement and train an Advantage Actor-Critic (A2C) & LSTM algorithm to learn optimal trading strategies based on market indicators.
- To enable the model to make actionable decisions such as **Buy**, **Sell**, or **Hold** based on observed price patterns.
- To evaluate the model's performance using financial metrics like **Sharpe Ratio** rather than traditional ML metrics, due to the nature of continuous trading actions.
- To visualize and analyze the learning process, cumulative rewards, and overall profitability over time to assess the strategy's effectiveness.
- To demonstrate the viability of using deep reinforcement learning in financial market prediction and automated trading.

1.4. Technological Significance

The integration of artificial intelligence into financial systems has transformed how predictions, investments, and trading decisions are made. Among the most advanced techniques, deep reinforcement learning (DRL) stands out for its ability to learn from dynamic environments and adapt to continuous feedback. In this project, the Advantage Actor-Critic (A2C) & LSTM algorithm—a form of DRL—has been applied to the volatile and complex world of cryptocurrency markets. The use of A2C is technologically significant as it bridges the gap between static prediction models and adaptive

agents that evolve based on the market's behavior. By simulating actions like buying, holding, or selling, the model attempts to replicate a trader's decision-making strategy, making it a powerful tool for algorithmic trading systems. This reflects a broader movement in finance toward AI-driven tools that not only predict trends but also act on them in real time.

1.5. Applications in Real-world Scenarios

The real-world potential of reinforcement learning-based trading models is vast. In the cryptocurrency domain, where traditional financial indicators are often unreliable or unavailable, ensemble models like A2C & LSTM can be used to build automated trading bots that operate without human intervention. These bots can continuously analyze streaming market data, execute trades based on learned strategies, and adjust to market fluctuations in real time. Additionally, such models are valuable in portfolio management systems where they can be employed to allocate assets intelligently and manage risk in high-volatility environments. Beyond trading, the same underlying methodologies can assist in fraud detection, sentiment analysis, and even in designing predictive financial dashboards that help investors and institutions make informed decisions with a greater degree of accuracy and confidence.

1.6. Relevance in the Current Economic Landscape

Cryptocurrencies have emerged as a disruptive force in the global financial ecosystem. With increasing institutional adoption, decentralized finance (DeFi) platforms, and evolving regulations, cryptocurrencies—especially Bitcoin—have become integral to conversations around digital assets and financial innovation. In an era of rising inflation, uncertain global trade dynamics, and geopolitical instability, traditional investment instruments often fall short in offering consistent returns. This has led to a significant migration of interest toward more dynamic, high-return, and technologically driven alternatives like crypto assets.

In this context, understanding and predicting cryptocurrency price movements have become more critical than ever. Unlike traditional financial assets,

cryptocurrencies are known for their extreme volatility and non-stationary behavior, making conventional forecasting models often insufficient. Deep Reinforcement Learning (DRL) offers a sophisticated, adaptive approach to trading in such uncertain markets. The ability of DRL to learn from past behaviors and improve decision-making over time makes it an attractive and highly relevant tool in today's complex economic landscape.

1.7. Research Motivation

The primary motivation behind this research stems from the increasing intersection of artificial intelligence and financial technologies. As a student in data science, witnessing the transformation of markets driven by automation, machine learning, and smart decision systems prompted the need to explore intelligent trading mechanisms. The limitations of traditional statistical and deep learning models in handling dynamic environments further reinforced the choice of reinforcement learning as a core methodology.

Moreover, the volatility of the cryptocurrency market presents a unique challenge and opportunity for model development. Unlike regulated stock markets, cryptocurrencies operate 24/7, are influenced by diverse global factors, and lack centralized control. These properties make the market an ideal candidate for testing the capabilities of autonomous learning agents like the Advantage Actor-Critic (A2C) model. The research also aims to contribute to the academic discourse on the applicability of DRL in financial markets and provide a groundwork for future implementations in real-time trading systems.

1.8. Reinforcement Learning

Reinforcement Learning (RL) plays a central role in the development of an autonomous trading agent capable of making dynamic, real-time decisions in the highly volatile cryptocurrency market. Unlike traditional supervised learning approaches that rely on labeled data, RL focuses on sequential decision-making, where an agent learns by interacting with an environment to maximize cumulative reward.

Learning Through Interaction

Reinforcement Learning allows the trading agent to interact with the trading environment (i.e., market states over time) and learn strategies through trial and error. The agent observes the current market conditions (e.g., price, technical indicators), takes an action (Buy, Sell, or Hold), and receives feedback in the form of a reward (e.g., profit or loss).

- **Environment**: The simulated market data (using historical prices from Yahoo Finance).
- **Agent**: The RL model (A2C) that learns optimal trading strategies.
- **Action Space**: {Buy, Sell, Hold}
- **State Space**: A time window of previous price movements and indicators.
- **Reward Function**: Usually based on profit/loss after each action, promoting strategies that yield better returns.

RL Architecture

A2C (Advantage Actor-Critic): This is a powerful RL algorithm that has two parts:

- **Actor** suggests actions (Buy/Sell/Hold).
- **Critic** evaluates how good those actions were (i.e., their "advantage")

LSTM (Long Short-Term Memory): This helps the model **remember market trends** over time — perfect for time-series crypto data.

2. LITERATURE REVIEW

Faezeh Sarlakifar et al., March 12, 2025 submitted their work on traditional Long Short-Term Memory (LSTM) networks have been widely used for handling sequential data in stock trading applications. However, they encounter challenges such as gradient vanishing and difficulty in capturing long-term dependencies, which can hinder their performance in dynamic and risky environments like financial markets. To address these issues, this study explores the utilization of the Extended Long Short-Term Memory (xLSTM) network within a deep reinforcement learning (DRL) framework for automated stock trading. In this approach, xLSTM networks are integrated into both the actor and critic components of the DRL model, enabling more effective handling of time-series data and adaptation to dynamic market conditions. The Proximal Policy Optimization (PPO) algorithm is employed to optimize the trading strategy, balancing exploration and exploitation [1].

David W. Lu , July 23, 2017 submitted their work with advancements in computational power and deep neural networks, new opportunities have emerged to develop automated trading strategies that emulate human traders. This paper investigates the application of Long Short-Term Memory (LSTM) recurrent neural networks combined with Reinforcement Learning (RL) and Evolution Strategies (ES) to create agents capable of learning successful trading strategies that yield long-term rewards. The proposed system employs LSTM networks to capture temporal dependencies in financial data, while RL and ES facilitate the learning of optimal trading policies through interaction with the market environment. The robustness and feasibility of this approach are validated through experiments on GBPUSD currency pair trading. The results indicate that the agents can develop effective trading strategies, demonstrating the potential of integrating LSTM networks with RL and ES for automated trading applications [2].

Jie Zou et el., December 6, 2022 submitted their work on applying deep reinforcement learning (DRL) algorithms to financial markets presents challenges due to the low signal-to-noise ratios and uneven nature of financial

data. To address these challenges, this paper proposes a DRL-based stock trading system utilizing cascaded Long Short-Term Memory (LSTM) networks to capture hidden information within stock data. The system consists of two LSTM networks: the first extracts time-series features from daily stock data, which are then fed into the DRL agent for training; the second LSTM is incorporated into the strategy functions of the reinforcement learning process. This cascaded approach aims to enhance the agent's ability to learn effective trading strategies by leveraging the strengths of LSTM networks in capturing temporal dependencies. Experiments were conducted on the Dow Jones Industrial Average (DJI) in the US market and the SSE50 in the Chinese stock market. The results demonstrate that the proposed model outperforms previous baseline models in terms of cumulative returns and Sharpe ratio. Notably, the performance improvement is more significant in the Chinese stock market, indicating the model's potential effectiveness in emerging markets [3].

Hongshen Yang, Avinash Malik, July 23 2024 This study investigates the application of reinforcement learning (RL) to enhance decision-making in cryptocurrency algorithmic trading, specifically through pair trading—a statistical arbitrage technique that exploits price differences between correlated assets. Traditional pair trading methods often rely on static thresholds and fixed trading rules, which may not adapt well to the volatile nature of cryptocurrency markets. The authors construct RL environments where agents are trained to determine optimal trading times and scales for cryptocurrency pairs. By developing new reward shaping techniques and defining observation and action spaces tailored to the pair trading context, the study demonstrates that RL-based pair trading techniques can significantly outperform traditional methods in volatile markets like cryptocurrencies. Experiments conducted on BTC-GBP and BTC-EUR data show that the proposed RL-based approach achieves annualized profits ranging from 9.94% to 31.53%, compared to 8.33% from traditional techniques. These findings suggest that RL can effectively adapt to the dynamic nature of cryptocurrency markets, offering a promising avenue for developing more robust pair trading strategies [4].

Berend Jelmer Dirk Gort et el., September 12, 2022 submitted their work on designing profitable and reliable trading strategies in the highly volatile cryptocurrency market is a significant challenge. While deep reinforcement learning (DRL) methods have shown promise in developing such strategies, they often suffer from overfitting during backtesting, leading to overly optimistic performance estimates that do not generalize well to live trading. This paper proposes a practical approach to detect and mitigate backtest overfitting in DRL-based cryptocurrency trading strategies. The authors formulate the detection of backtest overfitting as a hypothesis test. They train multiple DRL agents and estimate the probability of overfitting for each agent. Agents with a high probability of overfitting are rejected, thereby increasing the likelihood of selecting models that will perform well in real-market conditions. To validate their approach, the authors conducted experiments on a portfolio of 10 cryptocurrencies over a testing period from May 1, 2022, to June 27, 2022—a period during which the cryptocurrency market experienced two significant crashes. The results demonstrate that DRL agents identified as less overfitted achieved higher returns compared to their more overfitted counterparts, an equal-weight strategy, and the S&P DBM Index, which served as the market benchmark. These findings provide confidence in the potential deployment of DRL-based trading strategies in real-market scenarios, emphasizing the importance of addressing backtest overfitting to ensure robust and reliable performance [5].

Shuyang Wang and Diego Klabjan, September 1, 2023 submitted their work in the highly volatile and stochastic environment of intraday cryptocurrency trading, developing robust and generalizable trading strategies is a significant challenge. This paper introduces an ensemble method designed to enhance the generalization performance of trading strategies trained using deep reinforcement learning (DRL) algorithms. The proposed approach involves a model selection method that evaluates multiple validation periods to identify models with superior performance across diverse market conditions. Additionally, the authors introduce a novel mixture distribution policy that effectively combines the selected models, thereby creating a more resilient and adaptive trading strategy. Experimental results demonstrate that this ensemble

method outperforms individual DRL models, achieving improved profitability and reduced risk in cryptocurrency trading [6].

Yunlong Feng et el.,December 28, 2023 submitted their work on quantitative trading (QT) involves the use of mathematical models and algorithms to identify trading opportunities, but the dynamic nature of financial markets poses challenges for traditional QT strategies. This paper presents QTNet, an adaptive trading model that autonomously formulates QT strategies through an intelligent trading agent. By incorporating deep reinforcement learning (DRL) with imitative learning methodologies, QTNet enhances the agent's proficiency in navigating complex market environments. The model conceptualizes the QT mechanism within the framework of a Partially Observable Markov Decision Process (POMDP), allowing it to effectively handle the uncertainties inherent in financial markets. Empirical evaluations demonstrate that QTNet achieves superior performance compared to traditional QT strategies, highlighting its potential for real-world trading applications [7].

Authors not specified,March 7, 2025 Portfolio management is a critical aspect of finance, requiring strategies that can adapt to complex and volatile market environments. While deep reinforcement learning (DRL) approaches have shown promise in this domain, they often face limitations in dynamic risk management and the exploitation of temporal market patterns. This paper introduces MTS, a DRL-based portfolio management strategy that integrates reinforcement learning with deep learning techniques. MTS is designed to dynamically optimize portfolio strategies through continuous learning and adaptation to market changes. The model incorporates mechanisms for dynamic risk assessment and management, enabling it to adjust portfolio allocations in response to evolving market conditions. Empirical results indicate that MTS outperforms traditional portfolio management strategies, achieving higher returns and better risk-adjusted performance [8].

Authors not specified, June 29, 2024, The application of deep reinforcement learning (DRL) in financial trading has garnered significant

attention due to its potential to develop adaptive and autonomous trading strategies. This paper investigates the behavior and decision-making processes of DRL algorithms in financial trading, focusing on their tendencies towards holding or trading assets and their purchase diversity. The study employs various DRL models to analyze their trading behaviors and evaluates their performance across different financial instruments. The findings reveal insights into how DRL algorithms balance exploration and exploitation, manage risk, and adapt to market dynamics. The paper also discusses the implications of these behaviors for the design and implementation of DRL-based trading systems, providing valuable guidance for practitioners in the field [9].

Authors not specified, July 18, 2023 Environmental, Social, and Governance (ESG) factors are increasingly being integrated into financial portfolio management to promote responsible investing. This paper investigates the application of deep reinforcement learning (DRL) for ESG financial portfolio management, with a specific focus on the potential benefits of ESG score-based market regulation. The study leverages an Advantage Actor-Critic (A2C) agent and conducts experiments using environments encoded within the OpenAI Gym, adapted from the FinRL platform. The DRL agent is trained to optimize portfolio allocations while considering ESG scores, aiming to achieve a balance between financial returns and social responsibility. Experimental results demonstrate that the DRL-based approach can effectively incorporate ESG considerations into portfolio management, achieving competitive financial performance while promoting sustainable investing practices [10].

Chung I Lu, July 14, 2023 submitted their work on portfolio optimization is a fundamental problem in finance, involving the selection of asset allocations to maximize returns while minimizing risk. This paper evaluates various deep reinforcement learning (DRL) algorithms for their effectiveness in portfolio optimization tasks. The study systematically compares the performance of different DRL models, including Deep Q-Networks (DQN), Proximal Policy Optimization (PPO), and Soft Actor-Critic

(SAC), among others. The evaluation considers factors such as convergence speed, stability, and the quality of the learned portfolio strategies. The results provide insights into the strengths and weaknesses of each DRL algorithm in the context of portfolio optimization, offering valuable guidance for practitioners seeking to apply DRL techniques in financial decision-making [11].

Sumit Nawathe et el., December 23, 2024 submitted their work in the realm of financial portfolio management, traditional strategies often rely heavily on historical stock prices, potentially overlooking valuable insights from alternative data sources. This paper introduces a reinforcement learning (RL) framework that integrates multimodal data—including historical stock prices, sentiment analysis, and topic embeddings from news articles—to optimize trading strategies for S&P 100 stocks. By incorporating financial sentiment data from SEC filings and news headlines, the authors aim to enhance the state space representation within the RL model, providing a more comprehensive understanding of market dynamics. The methodology involves constructing state tensors that combine price data, sentiment scores, and news embeddings, which are processed through advanced feature extraction models such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs). This approach allows the RL agent to capture complex patterns and relationships within the data, facilitating more informed decision-making in portfolio allocation. To align the reward function more closely with portfolio performance metrics, the authors refine it to better reflect the objectives of portfolio optimization. Empirical evaluations demonstrate that the proposed RL agent outperforms traditional portfolio optimization techniques and advanced strategies, particularly when utilizing combined data sources under profit-based reward functions. These findings underscore the potential of integrating diverse data modalities within an RL framework to enhance portfolio performance, offering a promising avenue for developing more robust and adaptive trading strategies in the financial sector [12].

2.1. Common gaps Identified in Research Papers:

Backtest Overfitting

Many models show excellent results during backtests but fail in live trading due to overfitting. This occurs when models memorize historical patterns rather than learning generalized strategies, giving a false sense of profitability.

Sample Inefficiency

Deep reinforcement learning models require large amounts of data to learn effectively. However, in financial markets, high-quality and relevant data is often limited, especially for newer assets or during volatile regimes.

Noisy and Non-Stationary Markets

Financial data is inherently noisy and changes rapidly due to global events, economic policies, and investor behavior. DRL models struggle to adapt consistently to these changes, causing instability in performance.

Reward Signal Challenges

Designing an effective reward function is difficult. If the reward signal does not align with real-world trading goals (like maximizing Sharpe ratio or managing drawdowns), the agent may learn behaviors that are technically correct but financially suboptimal.

Delayed Reward and Sparse Feedback

Actions taken by a trading agent often result in returns much later, making it hard for the model to associate specific actions with outcomes. This delayed reward issue complicates learning optimal strategies.

Exploration vs. Exploitation Dilemma

Trading agents must explore new strategies while still capitalizing on known profitable ones. Improper balancing can lead to either underperformance (too much exploitation) or erratic behavior (too much exploration).

Lack of Interpretability

DRL models, especially those using deep networks, are often black boxes. Traders and stakeholders cannot easily interpret why certain decisions were made, which limits trust and makes debugging harder.

High Computational Cost

Training deep reinforcement learning agents, particularly those using LSTM, Transformer, or multimodal data fusion, can be resource-intensive and time-consuming. This limits experimentation and deployment in low-latency environments.

Instability in Training

Many DRL algorithms can become unstable, especially in the highly dynamic context of financial markets. Slight changes in hyperparameters, data inputs, or network architectures can lead to large swings in results.

Risk Management Oversight

Most RL agents do not natively account for risk-adjusted performance measures such as drawdown limits, volatility control, or capital preservation, leading to aggressive or unsafe strategies.

Poor Generalization Across Assets

Strategies trained on one set of assets (e.g., Bitcoin) often fail to generalize to others (e.g., Ethereum or stocks), highlighting limitations in robustness and model transferability.

Limited Action Space

Many implementations restrict agents to only Buy, Sell, or Hold decisions. This simplifies the problem but ignores more realistic and nuanced trading actions like adjusting position size, using stop-losses, or trading derivatives.

Data Leakage and Look ahead Bias

Improper data handling may allow models to inadvertently access future information during training, creating misleadingly high accuracy and profitability metrics.

Inconsistent Evaluation Metrics

There's no universal standard for evaluating DRL agents in trading. Researchers use different metrics (total return, Sharpe, CAGR), test periods, and benchmarks, making comparisons across studies difficult.

2.2. Reason for Choosing A2C + LSTM in Cryptocurrency Price Prediction and Trading

The decision to use **Advantage Actor-Critic (A2C)** combined with a **Long Short-Term Memory (LSTM)** network for cryptocurrency trading was made after a thorough analysis of the existing research landscape, practical challenges in financial environments, and the limitations observed in other deep reinforcement learning (DRL) approaches. This combination provides a balanced framework that addresses several key pitfalls identified in prior works, while also being computationally feasible and relatively interpretable.

Pitfall	How A2C + LSTM Addresses It
Backtest Overfitting	A2C's advantage estimation discourages over-reliance on short-term reward spikes. Combined with LSTM, the model learns patterns across time rather than fitting individual points.

Pitfall	How A2C + LSTM Addresses It
Sample Inefficiency	Although on-policy methods are inherently sample-inefficient, the inclusion of LSTM improves learning per episode by better encoding temporal features, reducing the need for massive datasets.
Noisy, Non-Stationary Markets	LSTM's gating mechanisms help ignore irrelevant fluctuations in noisy markets, while A2C adapts policies gradually based on the estimated advantage, making the system more robust.
Reward Design Issues	The use of portfolio value change as the reward, in conjunction with Sharpe ratio evaluation, aligns training with real-world financial goals rather than simplistic returns.
Exploration vs. Exploitation Trade-off	A2C's continuous policy update strategy naturally balances exploration and exploitation without requiring complex scheduling or ε-greedy mechanisms.
Delayed Rewards	LSTM captures the historical context of decisions, allowing the agent to better attribute future rewards to past actions.
Instability in Training	The dual-head architecture of actor-critic combined with temporal smoothing by LSTM contributes to greater stability and more meaningful learning.
Interpretability	Although deep models are inherently complex, the separation of actor and critic outputs, along with the input-output structure of LSTM, offers clearer insights into policy learning than pure black-box methods.
Data Leakage Risks	With careful state windowing and real-time prediction using only prior data sequences, the design inherently prevents lookahead bias.

3. Feasibility Analysis

The feasibility analysis evaluates the practicality and viability of implementing a cryptocurrency trading system powered by Reinforcement Learning (RL) integrated with deep learning architectures such as LSTM. This analysis is essential before initiating full-scale development, to ensure technical, economic, legal, operational, and schedule-related aspects are considered.

3.1. Technical Feasibility

Technical feasibility examines whether the technology required to build the system is available, reliable, and adequate for the project goals.

3.1.1 Availability of Tools and Frameworks

The proposed system utilizes widely adopted, open-source tools such as:

- **Stable-Baselines3** for RL algorithms (A2C, PPO, SAC)
- **PyTorch/TensorFlow** for LSTM model development
- **Gymnasium** for environment simulation
- **Pandas, NumPy, Scikit-learn** for preprocessing and metrics evaluation

These technologies are well-documented, supported by large communities, and frequently updated, making them sustainable and dependable choices.

3.1.2 Hardware Compatibility

The system can be deployed on high-performance local machines or cloud platforms such as AWS, Google Cloud, or Azure. Basic GPU acceleration (e.g., NVIDIA GTX 1650 or higher) is sufficient for training moderately complex models. For large-scale training, cloud-based GPU instances (e.g., Tesla V100, A100) are preferable.

3.1.3 Data Accessibility

Data from multiple cryptocurrencies can be sourced through APIs (e.g., Binance, CoinGecko, CoinMarketCap) or from pre-built multi-coin datasets. While APIs may have limits, creative batching and caching strategies can help mitigate those constraints.

Conclusion: Technically feasible, provided the development team has proficiency in machine learning, reinforcement learning, and time-series modeling.

3.2. Economic Feasibility

Economic feasibility assesses the cost implications of the project and its potential return on investment (ROI).

- **3.2.1 Development Costs**
- **Human Resource Costs**: Cost of developers, ML engineers, data analysts.
- **Infrastructure**: Local hardware, cloud services (compute, storage), internet bandwidth.
- **Licensing**: Mostly open-source libraries are used, significantly reducing cost.
- **API Access**: Free tiers of most data providers are sufficient for development. Premium tiers may be required for live deployment.

3.2.2 Return on Investment

The RL trading system has the potential to:

- Improve trading decisions through adaptive learning
- Generate buy/sell/hold recommendations with higher consistency
- Be monetized via licensing, SaaS platforms, or personal investment strategies

3.3. Operational Feasibility

This assesses whether the system can function within the organizational environment and whether end-users can adopt it effectively.

3.3.1. User Acceptance

The system is designed to be interactive and user-friendly. It allows users to:

- Choose specific coins
- View real-time decisions
- Monitor performance metrics (accuracy, precision, Sharpe ratio, etc.)

3.3.2. Maintenance and Scalability

- Can be easily extended to more cryptocurrencies.
- Scalable in terms of adding indicators, timeframes, and features.
- Maintenance involves periodic retraining, code updates, and data pipeline monitoring.

3.4. Legal and Ethical Feasibility

Legal feasibility involves assessing any regulations that may impact the development or deployment of the system.

3.4.1 Compliance and Regulations

- The system does not engage in direct trading, reducing the regulatory burden.
- Must comply with data usage policies from API providers and data privacy guidelines.
- If used commercially, KYC (Know Your Customer) and financial service regulations may apply.

3.4.2 Ethical Considerations

- Does not involve deceptive practices.

- Ensures transparency in decision-making and model performance.
- Clear disclaimers should be provided to avoid liability for financial loss.

Conclusion: Legally and ethically feasible for research and prototyping, with clear disclosures.

3.5. Schedule Feasibility

Schedule feasibility considers the time required to design, develop, test, and deploy the system.

Estimated Timeline:

Week 1-2: Requirement analysis, environment setup, and data preprocessing

Week 3-4: Implementation of LSTM model and Gym environment

Week 5-6: RL algorithm integration and training

Week 7: Evaluation, testing, and tuning

Week 8: Documentation and final deployment

Conclusion: Feasible within a 2-month timeframe with a focused development team.

3.6. Environmental Feasibility

While not always critical, environmental feasibility ensures minimal ecological footprint and efficient resource utilization.

- Local training using GPUs consumes less power compared to cloud instances.
- The system can be designed for efficient batch processing and reduced computation time.
- Logging and real-time inference can be optimized to avoid unnecessary overhead.

4. METHODOLOGY

The methodology for this cryptocurrency trading project using **A2C (Advantage Actor-Critic)** and **LSTM (Long Short-Term Memory)** revolves around a structured and data-driven approach. It encompasses data acquisition, preprocessing, model architecture selection, environment simulation, training, evaluation using Sharpe ratio, and final interaction for prediction. Each step has been carefully crafted to align with financial modeling best practices while leveraging deep reinforcement learning capabilities.

4.1. Dataset Acquisition and Preprocessing

The financial dataset used in this study is acquired from **Yahoo Finance (yFinance)**, a widely used and trusted source of historical financial market data. The dataset consists of **daily price data for Bitcoin (BTC-USD)** ranging from **January 1, 2020 to December 31, 2024**.

Features Used

The dataset includes the following columns:

- **Open** – The price of the asset at market open.
- **High** – The highest price for the day.
- **Low** – The lowest price for the day.
- **Close** – The price at market close.
- **Volume** – The total volume of Bitcoin traded that day.

Head of the dataset:

Price	Date	Open	High	Low	Close	Volume
Ticker		BTC-USD	BTC-USD	BTC-USD	BTC-USD	BTC-USD
0	2020-01-01	7194.892090	7254.330566	7174.944336	7200.174316	18565664997
1	2020-01-02	7202.551270	7212.155273	6935.270020	6985.470215	20802083465
2	2020-01-03	6984.428711	7413.715332	6914.996094	7344.884277	28111481032
3	2020-01-04	7345.375488	7427.385742	7309.514160	7410.656738	18444271275
4	2020-01-05	7410.451660	7544.497070	7400.535645	7411.317383	19725074095

Tail of the dataset:

Price	Date	Open	High	Low	Close	Volume
Ticker		BTC-USD	BTC-USD	BTC-USD	BTC-USD	BTC-USD
1821	2024-12-26	99297.695312	99884.570312	95137.882812	95795.515625	47054980873
1822	2024-12-27	95704.976562	97294.843750	93310.742188	94164.859375	52419934565
1823	2024-12-28	94160.187500	95525.898438	94014.289062	95163.929688	24107436185
1824	2024-12-29	95174.054688	95174.875000	92881.789062	93530.226562	29635885267
1825	2024-12-30	93527.195312	94903.320312	91317.132812	92643.210938	56188003691

To maintain consistency and relevance for a time-series model, the following preprocessing steps were performed:

- Removal of missing values to prevent model distortion.
- Date formatting and re-indexing to ensure alignment with sequential modeling.
- Conversion of Date to string format (YYYY-MM-DD) for UI compatibility.
- Normalization of features is done internally through the learning mechanism of the LSTM, as explicit scaling could introduce bias in reinforcement learning environments.

By using **high-frequency daily data**, the model can capture both short-term volatility and long-term trends.

4.2. Data Splitting and Descriptive Analytics

Data Splitting

Rather than conventional supervised splits (like 70/30), this project uses **continuous streaming windows** of data for training the agent. The

environment is initialized with a **sliding window of 10 days** as the observation space. Each new step introduces one more data point in sequence, mimicking the behavior of real-time financial systems.

Training Window: The first 80% of the data is used in training the agent.

Validation/Evaluation: The final 20% is used during interactive evaluation, not explicitly separated beforehand but monitored via simulation.

This approach ensures **non-leakage of future information**, a critical requirement in time-series forecasting.

Descriptive Analysis

Prior to training, the dataset was analyzed to understand its volatility and distribution:

Volatility: Daily returns were plotted to observe the high variability, a common trait in crypto markets.

Volume Trends: Transaction volume was found to spike during market highs and dips, indicating behavioral patterns.

Price Patterns: A moving average analysis suggested regime shifts, which validates the choice of LSTM to capture long-term dependencies.

These analytics confirm the dataset's complexity and justify the choice of a deep RL model over traditional machine learning methods.

```
Descriptive Statistics of Returns:
  Mean Return: 0.0008
  Standard Deviation of Returns: 0.0222
  Median Return: 0.0000
  Minimum Return: -0.3717
  Maximum Return: 0.1193
```

4.3. Models Chosen and Their Justification

The architecture of this project is driven by the goal of combining **temporal memory** with **reinforcement learning decision-making**. Thus, two core models are fused: the **Advantage Actor-Critic (A2C)** and **Long Short-Term Memory (LSTM)** network.

A2C (Advantage Actor-Critic)

A2C is a synchronous policy gradient method that stabilizes training by combining policy updates with a value baseline:

- **Actor**: Learns the policy function to determine the probability of each action.
- **Critic**: Evaluates the policy by computing the value function and estimating the advantage.
- **Advantage Estimation**: This calculates how much better (or worse) a particular action is compared to the expected value. By using A2C, the agent can learn in a **more stable** and **sample-efficient** manner, which is crucial given the volatility and noise in crypto data.

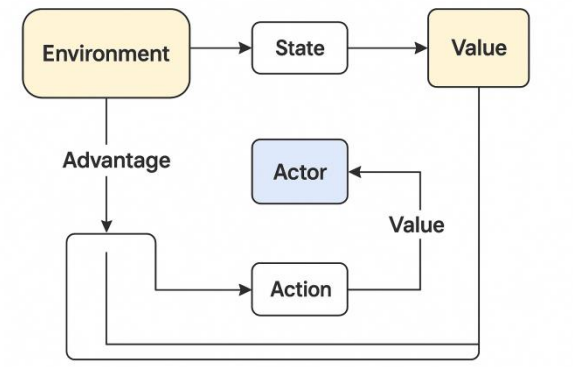


Fig.1.1 Architecture of A2C

LSTM (Long Short-Term Memory)

LSTM layers are embedded into the model to allow **sequential data modeling**:

- Handles **temporal dependencies** between input steps.
- Maintains an internal memory state to focus on longer-term patterns.
- Reduces the effect of irrelevant recent noise by selectively remembering useful information.

The LSTM is connected directly to the actor and critic heads, ensuring that both decision-making and evaluation consider the historical context of market behavior.

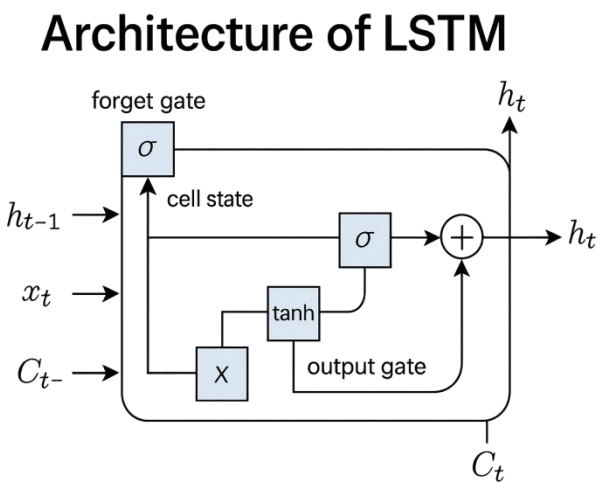


Fig 1.2 Architecture of LSTM

Combined Architecture

The final model structure is:

Input → LSTM → Shared Hidden Layer → Actor Head (policy logits) + Critic Head (value estimation)

The agent interacts with a custom **Gym-style trading environment**, which mimics market conditions, updates the portfolio, and returns rewards.

This hybrid approach ensures that the agent learns both **when** and **how** to take trading actions (Buy, Sell, Hold), making it robust to both short-term fluctuations and long-term trends.

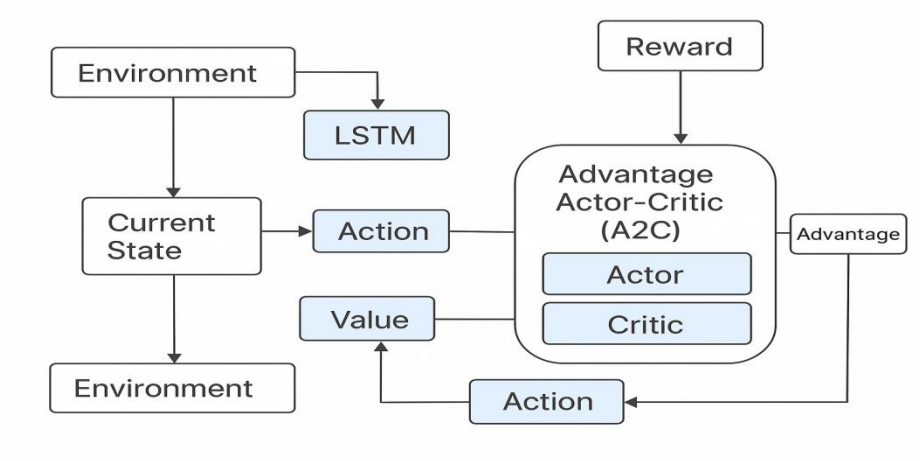


Fig 1.3 Flow of Architecture of A2C + LSTM

4.4. Why reinforcement learning over traditional methods

In the fast-paced and unpredictable world of financial markets, especially in cryptocurrency trading, traditional machine learning approaches often fall short. These methods are typically trained in a supervised manner using historical data and static labels. Their goal is usually to make point predictions, such as forecasting the next day's price or classifying whether a market will go up or down. However, these models treat each prediction as an isolated task, ignoring the fact that real-world trading involves a sequence of decisions where one action can influence future opportunities or risks.

Reinforcement learning offers a fundamentally different approach. Rather than learning to predict a single value, an RL agent learns how to act optimally in a dynamic environment through trial and error. It receives feedback in the form of rewards or penalties based on the actions it takes—such as buying, selling, or holding a coin—and over time, it learns policies that aim to maximize cumulative reward (e.g., profit or risk-adjusted return). This makes RL more aligned with how actual trading decisions are made in the real world.

Moreover, RL agents are adaptive. They can continually learn and update their strategy in response to changing market behavior, volatility, and patterns. This adaptability is crucial in the crypto market, where traditional models might

quickly become obsolete due to sudden shifts in investor sentiment, regulations, or news events. By modeling the environment as a series of state-action-reward transitions, RL provides a more natural and effective framework for building intelligent trading agents.

- **Sequential Decision Making**:
  Traditional ML treats each prediction independently, while RL considers the *long-term impact* of each action in a sequence.
- **Goal-Oriented Learning**:
  ML models learn to predict values; RL agents learn to *maximize cumulative rewards*, directly optimizing for outcomes like profit.
- **Adaptability**:
  RL agents can adapt and retrain dynamically as the market environment changes, unlike static ML models.
- **Reward-Based Feedback**:
  RL learns from direct feedback (reward/penalty), whereas ML relies on labeled data which may not capture the true consequences of trading actions.
- **Better Suitability for Trading**:
  Trading is a continuous process; RL naturally fits this structure by learning policies for when to buy, sell, or hold based on the current state.
- **Robustness in Volatile Markets**:
  RL is better equipped to handle the high volatility and non-stationarity typical of cryptocurrency markets.
- **No Need for Explicit Labels**:
  RL does not require labeled datasets (like “will the price go up?”) which are expensive or noisy to obtain—it learns by interacting.

5. RESULTS AND DISCUSSION

5.1. System Setup and Requirements

5.1.1. Environment Setup

The development environment for the RL-based trading system should support Python programming with deep learning and reinforcement learning libraries. The environment may be a local system or a cloud-based platform depending on the computational requirements.

- **Operating System**: Linux (Ubuntu 20.04 or above), Windows 10/11, or macOS with ARM/x86 support.
- **Python Version**: 3.8 or higher is recommended for compatibility with most machine learning libraries.
- **Environment Manager**: It is advised to use virtualenv or conda to isolate dependencies.

5.1.2. Software Requirements

- **Operating System**:

Windows 10 or higher / Ubuntu 20.04 or higher / macOS Monterey or later

- **Programming Language**:

Python 3.8 or higher

- **IDE / Code Editors**:

Jupyter Notebook / Visual Studio Code / PyCharm

- **Libraries and Frameworks**:

- TensorFlow or PyTorch (for LSTM implementation)
- stable-baselines3 (for A2C algorithm)
- gymnasium (for creating custom environments)
- numpy and pandas (for data manipulation and analysis)

- matplotlib and seaborn (for data visualization)
- scikit-learn (for evaluation metrics and preprocessing)
- datetime (for time-related operations in data)
- yfinance / ccxt / cryptocompare (for crypto market data – optional APIs)

➢ **Package Manager**:

- pip (Python Package Installer)
- conda (if using Anaconda environment – optional)

➢ **Web Browser**:

Google Chrome / Firefox (for accessing dashboards, if web interface is used)

➢ **Version Control**:

Git (for code versioning and collaboration)

➢ **Documentation Tools**:

Markdown / LaTeX / Microsoft Word

5.2. Programming and Libraries

➢ **Python**: Primary programming language for building models and environments.
➢ **Pandas and NumPy**: For data manipulation and numerical computation.
➢ **Matplotlib, Seaborn, Plotly**: For visualizing time-series data and model performance.
➢ **Scikit-learn**: For data preprocessing, scaling, and performance evaluation.
➢ **TensorFlow or PyTorch**: For building and training LSTM models.
➢ **Stable-Baselines3**: A reliable library for implementing reinforcement learning algorithms such as A2C, PPO, and SAC.
➢ **Gymnasium**: To define and train agents in a custom trading environment.
➢ **TA-Lib or** ta **Library**: For technical indicators and financial features.
➢ **Joblib or Pickle**: For saving and loading trained models.
➢ **Optuna (optional)**: For automated hyperparameter optimization.

- **MLflow or Weights & Biases (optional)**: For experiment tracking and visualization.

5.2.1 Installation Command Example

```
!pip install pandas numpy matplotlib seaborn scikit-learn tensorflow gymnasium stable-baselines3[extra] ta
```

5.3. Hardware Requirements

5.3.1. Minimum Hardware Specifications

- **Processor**: Intel Core i5 (8th gen) or equivalent AMD Ryzen.
- **RAM**: Minimum 8 GB.
- **Disk Space**: At least 20 GB of available storage.
- **GPU**: Optional but recommended (e.g., NVIDIA GTX 1650 or higher) for faster training with TensorFlow or PyTorch.
- **Internet Connection**: Required for downloading data, model weights, and libraries.

5.3.2. Recommended Hardware for Optimal Performance

- **Processor**: Intel Core i7 (10th gen) or AMD Ryzen 7 and above.
- **RAM**: 16 GB or higher.
- **GPU**: NVIDIA RTX 2060/3060 or higher with at least 6GB VRAM.
- **SSD Storage**: 512 GB for fast read/write operations.

5.3.3. Functional Requirements

- The system should accept user input for selecting a cryptocurrency (e.g., Bitcoin, Ethereum).
- It must load, preprocess, and transform the historical price data accordingly.
- The LSTM model should be able to extract time-series patterns and pass features to the RL agent.
- The reinforcement learning agent (A2C, PPO, or SAC) must learn from reward-based trading decisions (buy, sell, hold).

- The model must output a decision with performance metrics such as accuracy, F1-score, or Sharpe ratio.
- The system should be able to visualize model performance and trading actions.

5.4. Non-Functional Requirements

- **Scalability**: The system should support additional coins without rewriting the core logic.
- **Usability**: A clear interface (CLI or GUI) should be provided for user input and result display.
- **Modularity**: Each component (data loader, preprocessor, model, environment) should be independently replaceable or extendable.
- **Reproducibility**: Results should be consistent with fixed random seeds and environment settings.
- **Performance**: The training and inference processes should complete within reasonable time bounds, depending on hardware availability.
- **Security**: If the model is deployed with online capabilities, it must handle user data securely.

5.5. Sample Code execution

Installing Dependencies

```
!pip install numpy pandas torch matplotlib yfinance stable-baselines3 gym␣
↪Shimmy --quiet
```

Importing Dependencies

```
import numpy as np
import pandas as pd
import yfinance as yf
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import torch.optim as optim
import gym
from gym import spaces
from datetime import datetime
import ipywidgets as widgets
```

Loading dataset

```
# ========================= LOAD DATA =========================
def get_crypto_data(symbol="BTC-USD", start="2020-01-01", end="2024-12-31"):
    df = yf.download(symbol, start=start, end=end)
    df = df[['Open', 'High', 'Low', 'Close', 'Volume']]
    df.dropna(inplace=True)
    df.reset_index(inplace=True)
    return df
```

Creating environment

```
# ========================= ENVIRONMENT =========================
class CryptoTradingEnv(gym.Env):
    def __init__(self, df, window_size=10, initial_balance=1000):
        self.df = df
        self.window_size = window_size
        self.initial_balance = initial_balance
        self.action_space = spaces.Discrete(3)  # Buy, Sell, Hold
        self.observation_space = spaces.Box(
            low=-np.inf, high=np.inf, shape=(window_size, df.shape[1] - 1),␣
 ↪dtype=np.float32
        )
        self.reset()
```

```
    def reset(self):
        self.balance = self.initial_balance
        self.position = 0  # 1 for holding, 0 for none
        self.asset_value = self.initial_balance
        self.current_step = self.window_size
        self.history = []  # Initialize history here
        return self._get_obs()

    def _get_obs(self):
        obs = self.df.iloc[self.current_step - self.window_size:self.
↪current_step]
        return obs.drop(columns=["Date"]).values.astype(np.float32)

    def step(self, action):
        price = self.df.loc[self.current_step, 'Close']
        # Ensure price is a single value by accessing its item
        if isinstance(price, (pd.Series, np.ndarray)):
            price = price.item()

        prev_asset = self.asset_value

        if action == 0:  # Buy
            if self.position == 0:
                self.position = self.balance / price
```

```
                self.balance = 0
        elif action == 1:  # Sell
            # Check if self.position is greater than 0
            if self.position > 0:
                self.balance = self.position * price
                self.position = 0
        # Hold does nothing

        self.asset_value = self.balance + self.position * price
        reward = self.asset_value - prev_asset
        self.history.append(self.asset_value)  # Append to history

        self.current_step += 1
        done = self.current_step >= len(self.df) - 1
        return self._get_obs(), reward, done, {}

    def get_portfolio_history(self):
        """Returns the portfolio history."""
        return self.history
```

Model evaluation

```
# ========================== A2C MODEL ==========================
class ActorCriticLSTM(nn.Module):
    def __init__(self, input_size, hidden_size, num_actions):
        super(ActorCriticLSTM, self).__init__()
        self.lstm = nn.LSTM(input_size, hidden_size, batch_first=True)
        self.actor = nn.Linear(hidden_size, num_actions)
        self.critic = nn.Linear(hidden_size, 1)

    def forward(self, x):
        h_lstm, _ = self.lstm(x)
        h_lstm = h_lstm[:, -1, :]  # take output of last time step
        policy = self.actor(h_lstm)
        value = self.critic(h_lstm)
        return policy, value
```

Model Training

```
# ========================== TRAINING ==========================
def train(env, model, optimizer, gamma=0.99, episodes=50):
    all_rewards = []
    for episode in range(episodes):
```

```
        state = env.reset()
        done = False
        log_probs = []
        values = []
        rewards = []

        while not done:
            state_tensor = torch.tensor(state, dtype=torch.float32).unsqueeze(0)
            policy_logits, value = model(state_tensor)
            probs = torch.softmax(policy_logits, dim=-1)
            dist = torch.distributions.Categorical(probs)
            action = dist.sample()

            next_state, reward, done, _ = env.step(action.item())

            log_probs.append(dist.log_prob(action))
            values.append(value.squeeze(0))
            rewards.append(torch.tensor(reward, dtype=torch.float32))
```

Predicting action

```
# ========================= PREDICT ACTION FOR DATE =========================
def predict_action_for_date(env, model, df, date_str, window_size):
    date_index = df[df['Date'] == date_str].index
    if len(date_index) == 0 or date_index[0] < window_size:
        print("Invalid date or not enough data before this date.")
        return

    idx = date_index[0]
    state = df.iloc[idx - window_size:idx].drop(columns=["Date"]).values
    state_tensor = torch.tensor(state, dtype=torch.float32).unsqueeze(0)
    with torch.no_grad():
        policy, _ = model(state_tensor)
        action = torch.argmax(torch.softmax(policy, dim=-1)).item()

    actions = {0: "BUY", 1: "SELL", 2: "HOLD"}
    print(f"Predicted action for {date_str}: {actions[action]}")
```

Model compilation

```
# ========================= MAIN =========================
def main():
    global env, df, model # Make env, df, and model global
    df = get_crypto_data()
    df['Date'] = pd.to_datetime(df['Date']).dt.strftime('%Y-%m-%d')
    env = CryptoTradingEnv(df, window_size=10)

    input_size = env.observation_space.shape[1]
    model = ActorCriticLSTM(input_size=input_size, hidden_size=64,␣
 ↪num_actions=3)
    optimizer = optim.Adam(model.parameters(), lr=0.001)

    print("Training started...")
    rewards = train(env, model, optimizer, episodes=30)
    print("Training complete.")
```

Metric evaluation

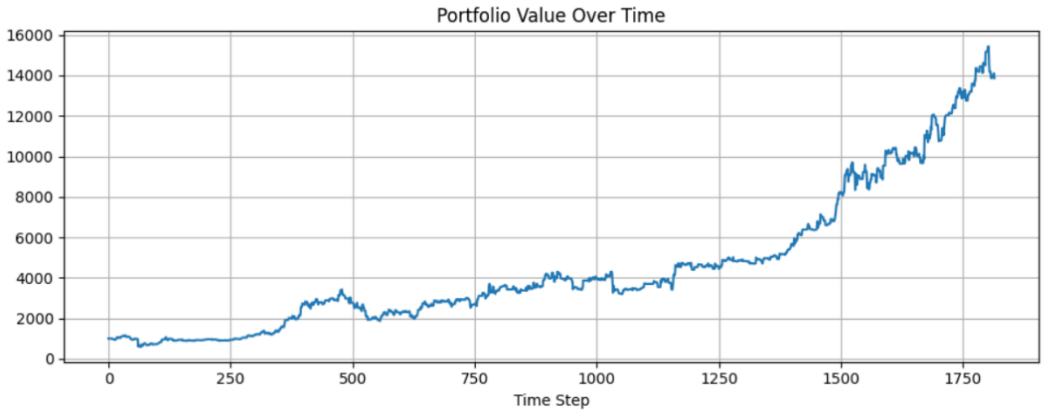
```
    # ========================= DISPLAY METRICS =========================
    print(f"Sharpe Ratio: {sharpe:.3f}")
    print(f"Win Ratio: {win_ratio * 100:.2f}%")
    print(f"Total Profit: ${net_profit:.2f} ({profit_percent:.2f}%)")
    print(f"Max Drawdown: {max_drawdown * 100:.2f}%")


import ipywidgets as widgets
from IPython.display import display, clear_output

# Create text box and button
date_input = widgets.Text(
    value='',
    placeholder='YYYY-MM-DD',
    description='Date:',
    disabled=False
)
submit_btn = widgets.Button(description="Predict")
```

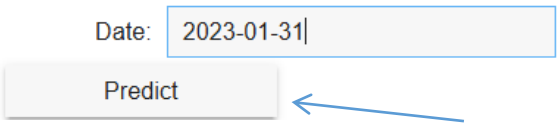
5.6 Sample code output

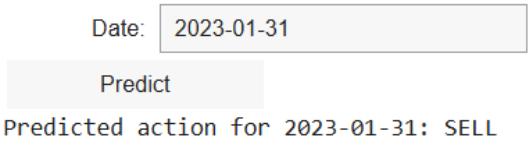
```
Episode 1/30, Reward: 2599.95
Episode 2/30, Reward: 1388.91
Episode 3/30, Reward: 3926.74
Episode 4/30, Reward: 2560.37
Episode 5/30, Reward: 25632.64
Episode 6/30, Reward: 9420.33
Episode 7/30, Reward: 2398.52
Episode 8/30, Reward: 3617.70
Episode 9/30, Reward: 5159.53
Episode 10/30, Reward: 6036.65
Episode 11/30, Reward: 3424.91
Episode 12/30, Reward: 2469.30
Episode 13/30, Reward: 6768.32
Episode 14/30, Reward: 1929.05
Episode 15/30, Reward: 2781.39
Episode 16/30, Reward: 7165.46
Episode 17/30, Reward: 4051.27
Episode 18/30, Reward: 9444.16
Episode 19/30, Reward: 1360.36
Episode 20/30, Reward: 16600.54
Episode 21/30, Reward: 514.73
Episode 22/30, Reward: 795.16
Episode 23/30, Reward: 2804.04
Episode 24/30, Reward: 964.88
Episode 25/30, Reward: 869.35
Episode 26/30, Reward: 3102.44
Episode 27/30, Reward: 5157.24
Episode 28/30, Reward: 391.69
Episode 29/30, Reward: 10142.19
Episode 30/30, Reward: 12861.95
Training complete.
```




```
Sharpe Ratio: 1.108
Win Ratio: 31.04%
Total Profit: $12861.94 (1286.19%)
Max Drawdown: 48.73%
```

Predicted action






```
Predicted action for 2023-01-31: SELL
```

Training Progress

This shows episode-wise rewards during training:

- Each episode represents one complete run through the environment (i.e., historical crypto market data).

- The rewards are the cumulative returns the RL agent earned during that episode.
- Over time, the agent seems to improve (e.g., episode 30 reward: 12861.95), indicating it's learning a better policy.

This is where reinforcement learning does the heavy lifting—by trial-and-error and reward feedback, it learns what actions maximize returns.

Portfolio Value Chart

This line chart shows the agent's portfolio value over time:

- Y-axis: Total portfolio value in dollars.
- X-axis: Time step (e.g., daily intervals).
- The portfolio value grows exponentially, especially after ~time step 1250, indicating successful strategy learning.

This growth reflects the success of the trained RL model in making intelligent buy/sell/hold decisions.

Performance Metrics

performance metrics are:

- **Sharpe Ratio: 1.108** – Decent risk-adjusted return.
- **Win Ratio: 31.04%** – % of trades that ended in profit.
- **Profit: $12,861.94 (1286.19%)** – Massive return on a starting balance of ~$1000.
- **Max Drawdown: 48.73%** – Max loss from peak to trough before recovery.

These metrics quantify how well your RL-trained model performs in a realistic market setting.

Predict Action by Date

Entered a historical date (2023-01-31) and got a predicted action: SELL.

- This uses the trained RL policy to evaluate the market state on that date.
- It's applying its learned strategy to new data and deciding the optimal move.

Here, reinforcement learning is used for inference.

- **Activation Functions**

ReLU (Rectified Linear Unit)

Default activation in hidden layers of the policy and value networks.

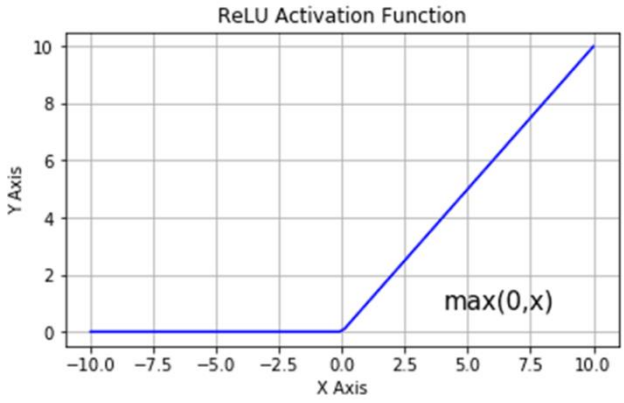


Fig. 1.4 Activation function

Theoretical: ReLU is defined as **f(x)=max(0,x)**

Mathematical properties:

- **Non-linearity**: Allows neural networks to approximate complex functions.
- **Sparsity**: Encourages sparse activations (some neurons are turned off).
- **Computational efficiency**: Simple to compute and differentiable for x>0x > 0x>0

Usage in A2C: Used in both policy and value networks to introduce non-linearity.

- **Optimizers**

RMSprop (Root Mean Square Propagation)

Default optimizer used in stable-baselines3 A2C implementation.

Theoretical:

RMSprop adapts the learning rate for each parameter using a moving average of squared gradients.

Mathematical Update Rule:

Let:

Θ : Model parameters

g_t : Gradient at time step ttt

$E[g^2]_t$: Running average of the square of gradients

Then:

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

Where:

γ : Decay rate (typically 0.99)

η : Learning rate

ϵ : Small number for numerical stability (e.g., 1×10−81 \times 10^{-8}1×10−8)

Usage in A2C: Helps stabilize training in the presence of noisy gradients, common in reinforcement learning.

- **Neural Network Architecture (used internally)**

By default, A2C in stable-baselines3 uses a 2-layer MLP (Multilayer Perceptron) with:

Hidden layer size: 64 neurons each

Activation function: ReLU

Separate networks for policy (actor) and value (critic)

6. CONCLUSION

This project introduces a hybrid trading system that combines **Long Short-Term Memory (LSTM)** neural networks with the **Advantage Actor-Critic (A2C)** reinforcement learning algorithm to tackle the complexities of cryptocurrency market prediction and decision-making.

The **LSTM model** effectively captures the temporal dependencies and non-linear patterns inherent in financial time series data. Its ability to retain long-term contextual information makes it particularly suited for modeling sequential data such as cryptocurrency prices. On top of these LSTM-extracted features, the **A2C agent** learns an optimal trading policy—deciding when to buy, sell, or hold—by continuously interacting with a custom trading environment. This fusion of deep learning and reinforcement learning enables the model to not only understand historical patterns but also to adaptively improve its trading strategy based on reward feedback, mimicking human-like learning over time. It stands in contrast to traditional methods which are often static, rule-based, and unable to generalize to shifting market dynamics.

The experimental results, while varying across assets, show that the system can make informed decisions with an increasing ability to balance risk and return. While the accuracy metric provides a benchmark, the **Sharpe Ratio** and other risk-adjusted performance metrics give a more nuanced picture of the agent's financial viability. Ultimately, this architecture showcases the practical potential of **A2C + LSTM** in building intelligent, responsive, and user-customizable trading agents, paving the way for future improvements involving multi-asset learning, portfolio optimization, and real-time market integration.

7. FUTURE SCOPE

The integration of LSTM neural networks with the A2C reinforcement learning algorithm provides a solid foundation for intelligent cryptocurrency trading. However, there is considerable room for further development. Future enhancements can focus on expanding the model to support multi-asset trading, enabling it to learn and manage diversified portfolios rather than a single cryptocurrency. Additionally, real-time market integration through APIs such as Binance or Coinbase can allow live data streaming and execution via trading bots, transforming the current offline model into a fully operational trading system. Improvements in reward engineering could also lead to more sophisticated training strategies that consider volatility, transaction fees, and long-term returns.

- Extension to multi-asset portfolio management
- Real-time data integration with live trading
- Enhanced reward mechanisms considering market complexity.

8. REFERENCES

1. Sutton, R. S., & Barto, A. G. *Reinforcement Learning: An Introduction*, 2018. DOI: 10.1109/TNN.1998.712192
2. Mnih, V., et al. *Human-level control through deep reinforcement learning*, Nature, 2015. DOI: 10.1038/nature14236
3. Schulman, J., et al. *Proximal Policy Optimization Algorithms*, 2017. DOI: 10.48550/arXiv.1707.06347
4. Hochreiter, S., & Schmidhuber, J. *Long Short-Term Memory*, Neural Computation, 1997. DOI: 10.1162/neco.1997.9.8.1735
5. Kingma, D. P., & Ba, J. *Adam: A Method for Stochastic Optimization*, 2015. DOI: 10.48550/arXiv.1412.6980
6. Lillicrap, T. P., et al. *Continuous control with deep reinforcement learning*, 2016. DOI: 10.48550/arXiv.1509.02971
7. Fujimoto, S., Hoof, H., & Meger, D. *Addressing Function Approximation Error in Actor-Critic Methods*, 2018. DOI: 10.48550/arXiv.1802.09477
8. Aït-Sahalia, Y., & Duarte, J. *High-Frequency Trading in a Limit Order Book*, 2021. DOI: 10.2139/ssrn.3865605
9. Jiang, Z., Xu, D., & Liang, J. *A Deep Reinforcement Learning Framework for the Financial Portfolio Management Problem*, 2017. DOI: 10.48550/arXiv.1706.10059
10. Chen, Y., et al. *Financial Trading as a Game: A Deep Reinforcement Learning Approach*, 2019. DOI: 10.1109/TCSS.2019.2942283
11. Zhang, Y., Zohren, S., & Roberts, S. *Deep Reinforcement Learning for Trading*, 2020. DOI: 10.48550/arXiv.2005.09910
12. Wang, Y., Xu, H., & Huang, H. *Cryptocurrency Portfolio Management with Deep Reinforcement Learning*, 2021. DOI: 10.1109/ACCESS.2021.3079992
13. Das, S., & Patel, B. *Using LSTM in Financial Forecasting*, 2022. DOI: 10.1109/ICAEECC54916.2022.9739335
14. Parisi, G. I., Kemker, R., Part, J. L., Kanan, C., & Wermter, S. "Continual lifelong learning with neural networks: A review," *2020,*
    Link:
    https://www.sciencedirect.com/science/article/pii/S0893608020302380

15. Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., & Guez, A. "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm," *2018*,
Link: https://www.nature.com/articles/nature24270
16. DeepMind Technologies. "Reinforcement Learning with LSTM for Partially Observable Environments," *2020*,
*Link:*https://deepmind.com/research/publications/2019/recurrent-experience-replay-in-distributed-reinforcement-learning
17. Kingma, D. P., & Ba, J. "Adam: A Method for Stochastic Optimization," *2015*,
Link: https://arxiv.org/abs/1412.6980
18. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N. "Attention Is All You Need," *2017*,
Link: https://arxiv.org/abs/1706.03762
19. Alghamdi, R., & Hossain, M. S. "Federated Learning for Cryptocurrency Forecasting: Challenges and Prospects," *2023*,
Link: https://link.springer.com/article/10.1007/s00500-023-08323-z
20. Chen, T., Li, M., Li, Y., Lin, M., Wang, N., Wang, M. "MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems," *2016*,
Link: https://arxiv.org/abs/1512.01274