The University of Jordan

School of Engineering
Department of Computer Engineering

Real-Time and AI-Based Automated Trading App

Supervisor:
Prof. Iyad Jafar

Author(s):

Adnan Khrais	0203000
Abdallah Alshebailat	0205000
Tariq Shawabkeh	0201313

May 20th, 2025

Submitted in partial fulfillment of the requirements of B.Sc. Degree in Computer Engineering

This page is intentionally left blank

ETHICAL STATEMENT

We, the undersigned students, certify and confirm that the work submitted in this project report is entirely our own and has not been copied from any other source. Any material that has been used from other sources has been properly cited and acknowledged in the report.

We are fully aware that any copying or improper citation of references / sources used in this report will be considered plagiarism, which is a clear violation of the Code of Ethics of the University of Jordan.

We further certify and confirm that we had no external help without the approval of our supervisor and proper acknowledgment when it is due. We certify and affirm that we never at any point commissioned a 3rd. party to do the work or any part of it on our behalf regardless of the amount charged or lack thereof. We also acknowledge that if suspected and thereafter proven that we commissioned a 3rd. party to do any part of this project that we risk failing the entire project.

We certify and confirm that all results presented in this project are true with no manipulation of data or fraud, that any statistics done, or surveys collected are conducted with the highest degree of scientific fidelity and integrity, and that if proven otherwise, we risk failing the entire project. We acknowledge that for any data collected, we have taken all the steps necessary in applying for proper authorizations if deemed necessary, and that all user data collected is subject to the utmost degrees of privacy and anonymity.

Adnan Khrais	Abdallah Alshebailat	Tariq Shawabkeh
20 May 2025	20 May 2025	20 May 2025
Signature:	Signature:	Signature:

This page is intentionally left blank

SUPERVISOR CERTIFICATION

I hereby certify that the students in this project have successfully finished their senior year project and by submitting this report they have fulfilled the partial requirements of B.Sc. Degree in Computer Engineering. ☐

I hereby certify that the students in this project have not completed their senior year graduation project and I do not approve that they proceed to the discussion. ☐

I suspect that the students have violated one or more of the clauses in the ethical statement and I suggest that an investigation committee look into the matter. ☐

Prof. Iyad Jafar

Signature:

This page is intentionally left blank

SYMBOLS, ABBREVIATIONS, AND ACRONYMS

AI	Artificial Intelligence
ANN	Artificial Neural Network
API	Application Programming Interface
ARIMA	Autoregressive Integrated Moving Average
ATR	Average True Range
BPTT	Backpropagation Through Time
DL	Deep Learning
EDA	Exploratory Data Analysis
EMA	Exponential Moving Average
FA	Fundamental Analysis
GA	Genetic Algorithm
GRU	Gated Recurrent Unit
HFT	High-Frequency Trading
LR	Learning Rate
LSTM	Long Short-Term Memory
MACD	Moving Average Convergence Divergence
ML	Machine Learning
MSE	Mean Squared Error
RMSE	Root Mean Squared Error
RNN	Recurrent Neural Network
ROI	Return on Investment
RSI	Relative Strength Index
SGD	Stochastic Gradient Descent
SMA	Simple Moving Average

ABSTRACT

This project introduces Kryseos, a real-time, AI-based trading application designed for cryptocurrency markets, with emphasis on Bitcoin and Ethereum. The name Kryseos, derived from the Greek word for “gold” or “golden,” reflects the system’s objective to deliver intelligent, value-driven automation in the financial domain.

Kryseos allows users to automate trading strategies using technical indicators and machine learning models, enhancing decision-making in volatile, continuously active markets. The platform comprises a Flutter-based cross-platform mobile front-end and a Python FastAPI back-end server, integrated with exchange APIs like Alpaca for real-time execution.

The server architecture supports modular strategies, including SMA, MACD, SMAADX, and an ML-based predictive model. User data—API keys, strategy preferences, and performance metrics—is securely stored and managed. Real-time synchronization between the app and server is achieved using WebSocket communication, and sensitive data is protected via Fernet encryption.

The mobile app offers intuitive controls for configuring strategies, monitoring live performance, visualizing equity curves, and reviewing trade history. Strategy effectiveness was evaluated statistically under simulated market conditions.

Kryseos provides a full-stack, secure, and user-centric solution for algorithmic trading, serving as a foundation for future innovations in decentralized finance and autonomous trading systems.

Table of Contents

ETHICAL STATEMENT II
SUPERVISOR CERTIFICATION IV
SYMBOLS, ABBREVIATIONS, AND ACRONYMS VI
ABSTRACT VII
LIST OF FIGURES XI
LIST OF TABLES XII
CHAPTER 1 INTRODUCTION 1
1.1. Problem Definition 1
1.1.1 Target Audience 1
1.2. Why Cryptocurrency? 1
1.3. Project Deliverables 2
1.4. Project Impact 2
1.5. Report Structure Overview 3
CHAPTER 2 RELATED WORK 4
2.1. Technical Indicator-Based Strategies 4
2.2. Machine Learning in Trading 4
2.3. Commercial Trading Platforms 5
2.4. Broker APIs and Execution Engines 5
2.5. Summary 5
CHAPTER 3 SYSTEM OVERVIEW 6
3.1. High-Level Architecture 6
3.2. Core System Components 6
3.2.1 Trading Strategies 6
3.2.2 Backend Server (FastAPI) 8
3.2.3 Broker APIs (Alpaca, Binance) 8
3.2.4 Database (SQLite) 8
3.2.5 Mobile Application (Flutter) 9
3.3. Data Flow Between Components 9

3.4. System Requirements 11
3.5. Summary 11
CHAPTER 4 Trading Strategies 12
4.1. Indicators 12
4.1.1 Simple Moving Average (SMA) Cross-Over 12
4.1.2 Moving Average Convergence Divergence (MACD) 13
4.1.3 Relative Strength Index (RSI) 15
4.1.4 Hull Moving Average (HMA) 16
4.1.5 Average Directional Index (ADX) 17
4.2. Strategies 19
4.2.1 ML Strategy 20
4.2.2 SMA Cross-Over Strategy 41
4.2.3 MACD Strategy 43
4.2.4 RSI Strategy 43
4.2.5 HMA Strategy 45
4.2.6 ADX and SMA Cross-Over 46
4.3. Performance Benchmark: Model vs. Buy-and-Hold Strategy 48
4.3.1 BTC Buy-and-Hold Strategy 48
4.3.2 ETH Buy-and-Hold Strategy 49
CHAPTER 5 Server-Side Implementation 52
5.1. Introduction 52
5.1.1 Core Technologies 52
5.1.2 Responsibilities 52
5.2. System Initialization 52
5.3. Database Management 53
5.4. User Authentication and API Key Encryption 53
5.5. In-Memory User State and Thread Model 54
5.6. Trading Session Control 54
5.7. Strategy Execution and Trade Flow 55
5.8. Performance Feedback System 56
5.9. Real-Time WebSocket Communication 57
5.10. Security Measures 57
5.11. Summary 58
CHAPTER 6 Mobile App Design and Implementation 59

6.1. Overview 59
6.2. Architecture Overview 59
6.3. UI and UX Design 60
6.3.1 Login Page (`login.dart`) 60
6.3.2 Trade Page (`trade.dart`) 61
6.3.3 Dashboard Page (`dashboard.dart`) 63
6.3.4 Market Page (`market.dart`) 63
6.4. State Management and Data Flow 64
6.5. Real-Time Communication 65
6.5.1 WebSocket Connection 65
6.5.2 Message Handling 66
6.6. Conclusion 66
CHAPTER 7 RESULTS AND DISCUSSION 67
7.1. Project Outcomes 67
7.2. System Performance and Integration 67
7.3. Discussion 68
CHAPTER 8 CONCLUSIONS AND FUTURE WORK 69
8.1. Conclusions 69
8.2. Future Work 69
REFERENCES 71

LIST OF FIGURES

FIGURE 1 - KRYSEOS HIGH-LEVEL OVERVIEW 6
FIGURE 2 - KRYSEOS DATA FLOW SEQUENCE DIAGRAM 10
FIGURE 3 - SMA CROSS-OVER: BLUE (9-PERIOD SMA), ORANGE (21-PERIOD SMA) 13
FIGURE 4 - MACD CROSS-OVER: BLUE (MACD LINE), ORANGE (SIGNAL LINE) 15
FIGURE 5 - RSI BEHAVIOR RELATIVE TO THRESHOLD LEVELS 30 AND 70. 16
FIGURE 6 - HULL MOVING AVERAGE (HMA) WITH TREND-BASED BUY AND SELL SIGNALS 17
FIGURE 7 - ADX BEHAVIOR ACROSS DIFFERENT TREND STRENGTHS 19
FIGURE 8 - UNROLLED STRUCTURE OF A RECURRENT NEURAL NETWORK (RNN). 21
FIGURE 9 - INTERNAL ARCHITECTURE OF AN LSTM CELL. 22
FIGURE 10 -ACTUAL VS. PREDICTED CLOSING PRICES ON UNSEEN DATA. 24
FIGURE 11 -ZOOMED-IN VIEWS OF ACTUAL VS. PREDICTED PRICES. 24
FIGURE 12 -THE PERFORMANCE OF THE FIRST MODEL. 26
FIGURE 13 -THE PERFORMANCE OF THE SECOND MODEL. 28
FIGURE 14 -THE PERFORMANCE OF THE THIRD MODEL. 31
FIGURE 15 -THE PERFORMANCE OF THE FOURTH MODEL. 34
FIGURE 16 -THE PERFORMANCE OF THE FIFTH MODEL. 37
FIGURE 17 -THE PERFORMANCE OF THE OPTIMIZED FIFTH MODEL. 39
FIGURE 18 -THE PERFORMANCE OF THE ETH MODEL. 41
FIGURE 19 -THE PERFORMANCE OF THE SMA STRATEGY 42
FIGURE 20 -THE PERFORMANCE OF THE MACD STRATEGY 44
FIGURE 21 -THE PERFORMANCE OF THE RSIANDSMA50 STRATEGY 45
FIGURE 22 -THE PERFORMANCE OF THE HMA STRATEGY 46
FIGURE 23 -THE PERFORMANCE OF THE ADX AND SMA CROSS-OVER STRATEGY 47
FIGURE 24 -THE PERFORMANCE OF THE BTC BUYHOLD STRATEGY 48
FIGURE 25 -THE PERFORMANCE OF THE ETH BUYHOLD STRATEGY 50
FIGURE 26 -KRYSEOS APP LOGIN PAGE. 61
FIGURE 27 -KRYSEOS APP TRADE TAB. 61
FIGURE 28 -SYMBOL SELECTION. 62
FIGURE 29 -STRATEGY SELECTION. 62
FIGURE 30 -(20%) RISK RATIO. 62
FIGURE 31 -(50%) RISK RATIO. 62
FIGURE 32 -KRYSEOS APP DASHBOARD TAB. 63
FIGURE 33 -KRYSEOS APP DASHBOARD TAB. 63
FIGURE 34 -KRYSEOS APP MARKET TAB. 64

LIST OF TABLES

TABLE 1 - MARKET COMPARISON: STOCK, FOREX, AND CRYPTOCURRENCY 2
TABLE 2 - COMPARISON OF TECHNICAL INDICATOR STRATEGIES 4
TABLE 3 - COMPARISON OF ML METHODS IN TRADING 4
TABLE 4 - COMPARISON OF COMMERCIAL TRADING PLATFORMS 5
TABLE 5 - COMPARISON OF FASTAPI AND FLASK 8
TABLE 6 - COMPARISON OF SQLITE, FIREBASE, AND POSTGRESQL 9
TABLE 7 - DIFFERENCE BETWEEN EMA AND SMA 14
TABLE 8 - PERFORMANCE METRICS OF THE FIRST MODEL 25
TABLE 9 - PERFORMANCE METRICS OF THE SECOND MODEL 28
TABLE 10 - PERFORMANCE METRICS OF THE THIRD MODEL 30
TABLE 11 - PERFORMANCE METRICS OF THE FOURTH MODEL 33
TABLE 12 - PERFORMANCE METRICS OF THE FIFTH MODEL 36
TABLE 13 - PERFORMANCE METRICS OF THE OPTIMIZED FIFTH MODEL 38
TABLE 14 - PERFORMANCE METRICS OF THE ETH MODEL 40
TABLE 15 - PERFORMANCE METRICS OF THE SMA STRATEGY 42
TABLE 16 - PERFORMANCE METRICS OF THE MACD STRATEGY 43
TABLE 17 - PERFORMANCE METRICS OF THE RSIANDSMA50 STRATEGY 44
TABLE 18 - PERFORMANCE METRICS OF THE HMA STRATEGY 45
TABLE 19 - PERFORMANCE METRICS OF THE ADX AND SMA CROSS-OVER STRATEGY 47
TABLE 20 - PERFORMANCE METRICS OF THE BEST MODEL AND BUYHOLD ON BTC 49
TABLE 21 - PERFORMANCE METRICS OF THE BEST MODEL AND BUYHOLD ON ETH 51

This page is intentionally left blank

CHAPTER 1
INTRODUCTION

In recent years, the cryptocurrency market—especially Bitcoin and Ethereum—has experienced exponential growth, attracting a diverse range of investors and traders. However, the extreme volatility and continuous 24/7 nature of these markets present significant challenges for individuals attempting to trade manually. To address these challenges, this project introduces Kryseos—a real-time, AI-powered automated trading system specifically tailored for cryptocurrency trading, with a primary focus on Bitcoin and Ethereum.

Kryseos integrates a Flutter-based mobile application with a Python-powered backend server to enable users to configure trading strategies, monitor performance, and interact with live crypto exchanges through secure API connections. By combining technical indicators, machine learning models, and real-time data processing, Kryseos offers an intelligent and automated trading solution that enhances efficiency, responsiveness, and decision-making, while significantly reducing reliance on manual intervention.

1.1. Problem Definition

Cryptocurrency markets—particularly those involving Bitcoin and Ethereum—are highly volatile and operate continuously, 24/7, making it challenging for individual traders to respond effectively to rapid market movements. Manual trading in such fast-paced environments often results in delayed decisions, missed opportunities, and susceptibility to emotional bias. Additionally, many existing crypto trading platforms either lack intelligent automation or are not user-friendly for non-expert traders.

To address these challenges, this project proposes Kryseos—a real-time, AI-based trading application that automates trade execution in Bitcoin and Ethereum markets using algorithmic strategies and machine learning models. Kryseos provides users with a responsive, data-driven trading experience that enhances performance, consistency, and accessibility.

1.1.1. Target Audience

The application primarily targets cryptocurrency traders in regions where digital asset trading is permitted or practiced with minimal restrictions. Countries like the UAE, Bahrain, and Turkey offer favorable or active regulatory environments. In Jordan, while financial institutions are restricted from participating, individuals are not explicitly banned from trading and continue to engage in crypto activities at their own risk.

1.2. Why Cryptocurrency?

Cryptocurrency markets offer several distinct advantages that make them an ideal domain for developing automated trading systems. One of the key benefits is high liquidity, particularly in assets like Bitcoin and Ethereum, which ensures that buy and sell orders are executed with minimal slippage and without significant delays—an essential factor for real-time algorithmic trading.

Unlike traditional stock or forex markets, which operate within limited hours, cryptocurrency markets are open 24/7, allowing continuous trading and eliminating gaps caused by overnight or weekend closures.

Additionally, many crypto exchanges offer high leverage, often up to 1:100, enabling traders to control larger positions with relatively small capital. These characteristics make cryptocurrency trading highly dynamic and well-suited for automated, AI-driven systems that can monitor and respond to market movements without human intervention. Table 1 summarizes the key differences between cryptocurrency, forex, and stock markets.

Table 1 – Market Comparison: Stock, Forex, and Cryptocurrency

Market	**Liquidity**	**Trading Hours**	**Leverage**
Stock	High (varies by stock)	Weekdays, limited hours	Up to 2:1 (regulated)
Forex	Very high (global)	24/5 (weekdays)	Up to 50:1 (regulated)
Crypto	High (major coins)	24/7 (continuous)	Up to 100:1 (unregulated)

1.3. Project Deliverables

The **Kryseos** project deliverables consist of two main components: a cross-platform mobile application and a robust backend server.

The **mobile application**, developed using Flutter, serves as the user interface. It enables users to securely enter their API credentials, select and configure automated trading strategies, and monitor real-time market data and performance feedback.

The **backend server**, implemented using Python's FastAPI framework, is responsible for executing trading strategies, managing live market data streams, maintaining user sessions, and securely interfacing with external platforms such as Alpaca (for trade execution) and Binance (for real-time price data).

These components operate in coordination to provide a complete end-to-end automated trading system. Users initiate and configure trades from the mobile app, while the server performs continuous market analysis and executes trades based on the selected strategy logic in real time.

Disclaimer: Users of this application trade at their own risk. The developers and maintainers of Kryseos are not liable for any financial losses incurred through the use of this system. This disclaimer aligns with standard practices followed by legitimate trading applications.

1.4. Project Impact

The development of an AI-based automated cryptocurrency trading application has meaningful implications across global, economic, environmental, and societal dimensions. On a global scale, it contributes to the democratization of access to financial technologies, allowing individuals from various regions to participate in digital asset markets with greater efficiency and lower barriers to entry.

Economically, it enhances trading performance by reducing emotional bias and enabling data-driven decision-making, potentially leading to more efficient market participation and improved capital utilization.

Societally, the system promotes financial inclusion by offering individuals a tool to engage with advanced algorithmic trading without requiring deep technical expertise, fostering greater engagement with emerging financial ecosystems and encouraging responsible personal investment practices.

1.5. Report Structure Overview

The remaining chapters of this report provide a comprehensive breakdown of the system design, implementation, and evaluation:

- **Chapter 2 – Related Work:** Reviews existing trading strategies, machine learning applications in finance, and an overview of commercial trading platforms and broker APIs.
- **Chapter 3 – System Overview:** Outlines the system architecture and components, including the backend server, trading strategies, broker integration, database structure, and mobile frontend.
- **Chapter 4 – Trading Strategies:** Details both traditional indicator-based and machine learning strategies, along with their logic, implementation, and comparative performance analysis.
- **Chapter 5 – Server-Side Implementation:** Describes the backend infrastructure built with FastAPI, covering threading, strategy execution, API security, and real-time communication.
- **Chapter 6 – Mobile App Design:** Explains the Flutter app's architecture, UI/UX design, and how it integrates with the backend to support live trading and user interaction.
- **Chapter 7 – Results and Discussion:** Summarizes the system's implementation outcomes. It also discusses architectural strengths, and observed limitations.
- **Chapter 8 – Conclusions and Future Work:** Summarizes the project achievements and outlines directions for expanding to live trading, platform support, and enhanced strategy features.

CHAPTER 2
RELATED WORK

Algorithmic and automated trading systems have received significant attention in both academia and industry. Research has explored strategies based on technical indicators, machine learning (ML), and reinforcement learning (RL), while commercial platforms have implemented various models for practical trading automation. This chapter discusses these works, compares their advantages and limitations, and positions Kryseos relative to existing approaches.

2.1. Technical Indicator-Based Strategies

Technical indicators such as SMA, MACD, and RSI have been widely used in automated trading strategies. Researchers have shown that using a single indicator can be effective in some conditions, but combining multiple indicators often yields better performance. For instance, Hadi and Fernández showed the effectiveness of MACD in certain markets [1], while others leveraged genetic algorithms to optimize combinations of indicators [2].

Kryseos includes a modular set of technical strategies such as SMA, MACD, SMAADX, and RSISMA50. Unlike static rule-based systems, it allows flexible configuration. Table 2 compares common indicator strategies used in previous research versus Kryseos.

Table 2 – Comparison of Technical Indicator Strategies

Study/System	**Indicators Used**	**Customizability**
Sezer et al. (2020)	RSI, MACD, Williams %R	Fixed features [2]
Hadi & Fernández (2016)	MACD	Low [1]
Kryseos	SMA, MACD, RSI, etc.	High (modular configuration)

2.2. Machine Learning in Trading

Machine learning has been employed to model market patterns using features such as price history and indicators. Sezer et al. trained ANN models on indicators [2], while Mello et al. tested 30 different neural networks, showing that only a few were profitable [3]. Reinforcement learning has recently been used in crypto markets to optimize profits through reward-based learning [4].

Kryseos uses ML as one of several modular strategies. It enhances static indicators by learning optimal weights and combinations. Table 3 summarizes major ML methods applied to trading.

Table 3 – Comparison of ML Methods in Trading

Study	**ML Model**	**Market Tested**
Sezer et al.	ANN	Stock
Mello et al.	Deep Neural Networks	Stock
Deng et al.	Deep Q-Network (RL)	Bitcoin
Kryseos	Supervised Learning (RNN - LSTM)	Bitcoin, Ethereum

2.3. Commercial Trading Platforms

Real-world trading platforms like Pionex, 3Commas, and HaasOnline provide practical automation. Pionex offers preset bots with simple configuration, but limited customizability [5]. 3Commas offers advanced bot design with DCA and SmartTrade features [6]. HaasOnline enables full scripting of bots with HaasScript [7].

Unlike these platforms, Kryseos is fully self-hosted, prioritizing user control and API key security through encryption. It supports modular strategy logic, including ML and indicator-based approaches. Table 4 compares selected commercial platforms to Kryseos.

Table 4 – Comparison of Commercial Trading Platforms

Platform	**Customization**	**Security Model**	**UI Type**
Pionex	Low	Cloud-stored keys	Web-based
3Commas	Medium-High	Cloud + Exchange API	Web/mobile
HaasOnline	Very High	Local or Cloud	Desktop/Web
Kryseos	High	Encrypted, Self-hosted	Flutter mobile app

2.4. Broker APIs and Execution Engines

Modern algorithmic systems rely on broker APIs for real-time data and order execution. Alpaca, used in Kryseos, supports WebSocket streams, REST endpoints, and paper trading [8]. Similar systems, like QuantConnect, also provide execution frameworks [9].

The server component of Kryseos uses FastAPI to handle trading signals and integrates with Alpaca's APIs to execute trades and stream data. The real-time feedback loop is maintained via WebSocket between the server and the Flutter client.

2.5. Summary

In summary, prior works have contributed numerous strategies, platforms, and frameworks for algorithmic trading. Kryseos integrates technical indicators, supervised ML, secure API handling, and modular deployment in a mobile-first trading app tailored to cryptocurrency markets. The project draws inspiration from academic and industrial solutions while offering a fully customizable and secure platform for real-time trading.

CHAPTER 3
SYSTEM OVERVIEW

This chapter provides a comprehensive overview of the Kryseos system, describing its high-level architecture, core components, and internal data flow. It serves as a foundation for understanding how Kryseos operates as a real-time, AI-powered cryptocurrency trading platform.

3.1. High-Level Architecture

Kryseos is built on a modular full-stack architecture for real-time cryptocurrency trading. This section outlines the system's main components—the mobile app, backend server, database, and broker APIs—and how they interact to enable automated strategy execution and user feedback. The overall structure and data flow are illustrated in Fig. 1.

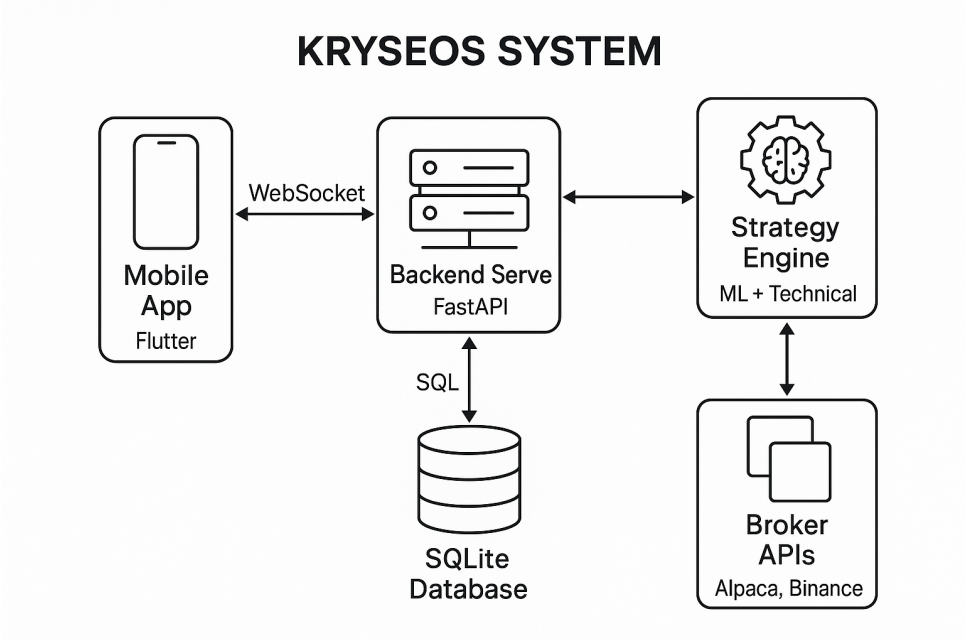

Figure 1 – Kryseos High-level Overview.

3.2. Core System Components

The Kryseos system is composed of several interconnected components, each responsible for a specific aspect of the trading workflow. This section describes the functionality and role of each core module in the overall architecture.

3.2.1. Trading Strategies

The process begins with fetching real-time market data, which includes open, high, low, close prices. This data is then passed to the strategy module where it is processed using either technical indicators or machine learning models to interpret market conditions. Based on this analysis, the system generates trading signals such as BUY or SELL. Finally, these signals are executed by the backend, which places the corresponding orders through the broker's API. The loop is then repeated continuously, enabling real-time and automated trading.

The Kryseos system supports a set of modular trading strategies, each based on a widely recognized technical analysis indicator or a machine learning approach. The following strategies are available in the current implementation:

- **SMA Crossover Strategy**: This strategy computes two simple moving averages (SMA) over the asset's closing price: a short-term (9-day) and a long-term (21-day) average. A BUY signal is generated when the short-term SMA crosses above the long-term SMA (bullish crossover), while a SELL signal is triggered when the opposite occurs.
- **MACD Strategy**: Based on the Moving Average Convergence Divergence indicator, this strategy uses the difference between the 12-period and 26-period exponential moving averages (EMA) to form the MACD line, and a 9-period EMA of the MACD as the Signal line. A BUY signal is generated when the MACD line crosses above the Signal line, and a SELL signal when it crosses below.
- **HMA Strategy (Hull Moving Average)**: The Hull Moving Average is designed to reduce lag in trend-following strategies. This strategy calculates the HMA and issues a BUY signal when the most recent HMA value is higher than its value two days ago, and a SELL signal when the opposite is true.
- **RSI with SMA50 Filter Strategy**: This strategy combines the Relative Strength Index (RSI) with a 50-period SMA filter to enhance signal reliability. RSI thresholds are dynamically adjusted based on whether the current price is above or below the SMA50. A BUY signal is triggered when RSI crosses below a threshold during a downtrend, and a SELL signal when it crosses above a threshold during an uptrend.
- **ADX with SMA Crossover Strategy**: This hybrid strategy enhances a standard SMA crossover (9-day and 21-day) by incorporating the Average Directional Index (ADX) to confirm trend strength. A BUY or SELL signal is only issued if the crossover occurs and the ADX value exceeds 20, indicating a strong trend.
- **Machine Learning Model-Based Strategy**: This strategy leverages a trained neural network model to predict market movements based on recent price data, MACD, and MACD signal values. The model uses preprocessed and standardized features over a fixed time window (e.g., 10 or 25 days) to forecast the next price movement. The RSI value is also computed separately to assist with final decision logic. This model was trained and exported using joblib, and inference is performed in real-time using recent Binance data.

Each strategy is implemented as a modular Python module and can be dynamically selected by the user through the mobile application interface. This modularity allows for future expansion and customization of trading logic.

3.2.2. Backend Server (FastAPI)

The **Backend Server** functions as the central controller and core logic coordinator of the system. It is implemented using Python's `FastAPI` framework, chosen for its high performance, native asynchronous support, and excellent scalability characteristics (as detailed later in Table 5).

The server handles all incoming requests from the mobile application, including user login, configuration updates, and start/stop trading signals. It manages user session states, enforces secure communication protocols, and orchestrates the operation of the trading strategies.

Furthermore, the backend coordinates closely with the Strategy Engine and broker APIs (such as Alpaca and Binance), ensuring a seamless and real-time trading experience across devices and components.

Table 5 – Comparison of FastAPI and Flask

Aspect	**FastAPI**	**Flask**
Performance	High throughput; optimized for speed with async support.	Moderate performance; synchronous by default.
Asynchronous Support	Built-in support using ASGI; ideal for concurrent operations.	Limited support; requires additional configurations.
Scalability	Excellent; designed for high-concurrency environments.	Moderate; relies on WSGI and external tools for scaling.

3.2.3. Broker APIs (Alpaca, Binance)

The Kryseos system integrates with two major broker APIs: **Alpaca** and **Binance**, each serving a distinct purpose within the architecture.

- **Alpaca API**: The Alpaca API is used for executing trades. It was chosen due to its simple and well-documented REST interface, as well as its support for paper (simulated) trading. This feature makes it ideal for developing and testing trading strategies without the risk of financial loss, a capability not offered by many other platforms.
- **Binance API**: The Binance API is used exclusively for retrieving real-time market data such as open, close, high, and low prices. This data is essential for driving the decision-making logic of the strategy engine, enabling accurate and up-to-date market analysis.

By separating the concerns of data retrieval and trade execution between two specialized APIs, the system ensures flexibility and reliability in both simulation and real-world trading scenarios.

3.2.4. Database (SQLite)

A lightweight local database is employed by the backend to securely store persistent data, including encrypted user credentials (API keys and secrets) and trading equity history for performance monitoring. `SQLite` was chosen for its simplicity and efficiency, suitable for the system's moderate data volumes (see Table 6 for a comparison with alternative databases). The database ensures that sensitive information is never stored in plaintext and maintains a reliable record of trading performance over time.

Table 6 – Comparison of SQLite, Firebase, and PostgreSQL

Aspect	**SQLite**	**Firebase**	**PostgreSQL**
Deployment Complexity	Minimal; embedded within application.	Moderate; requires cloud setup and configuration.	High; necessitates server setup and maintenance.
Performance	High for local, low to moderate data volumes.	Optimized for real-time synchronization; may introduce latency.	Excellent for complex queries and large datasets.
Scalability	Limited; suitable for single-user or low-concurrency applications.	High; designed for real-time, multi-user applications.	High; supports extensive concurrent operations and large-scale deployments.

3.2.5. Mobile Application (Flutter)

The Kryseos mobile application, built using the Flutter framework, serves as the primary user interface. It allows users to log in, configure trading strategies, control trading sessions, and monitor live performance. Designed to be cross-platform, it offers a consistent experience across Android and iOS.

The app is structured around reusable widgets (stateless and stateful) for building responsive UI elements like dropdowns and sliders. Navigation is handled through a named route system that separates pages such as login, dashboard, and trading configuration.

Communication with the backend uses HTTP for authentication and setup, and WebSocket for streaming real-time updates like equity changes and trade signals. A unified ‘ThemeData‘ ensures visual consistency across the app.

Main pages include:

- **Login Page** – Authenticates users and stores encrypted credentials.
- **Main Page** – Root container with navigation across tabs.
- **Trade Tab** – Configure strategy parameters and start/stop trading.
- **Dashboard Tab** – Visualize real-time performance metrics.
- **Market Tab** – View live market charts and price data.

3.3. Data Flow Between Components

Kryseos operates through a coordinated data flow between its core components: the mobile application, backend server, strategy engine, and broker APIs. Upon login, users submit their API credentials and strategy preferences, which are securely transmitted and stored by the backend. Starting a session triggers the backend to launch the selected strategy, continuously fetch market data, evaluate trading signals, and execute orders via the broker API. Performance metrics

are computed after each cycle and pushed to the mobile app in real time using WebSocket communication. The session ends cleanly when the user stops trading. The complete interaction flow is illustrated in the sequence diagram shown in Fig. 2.

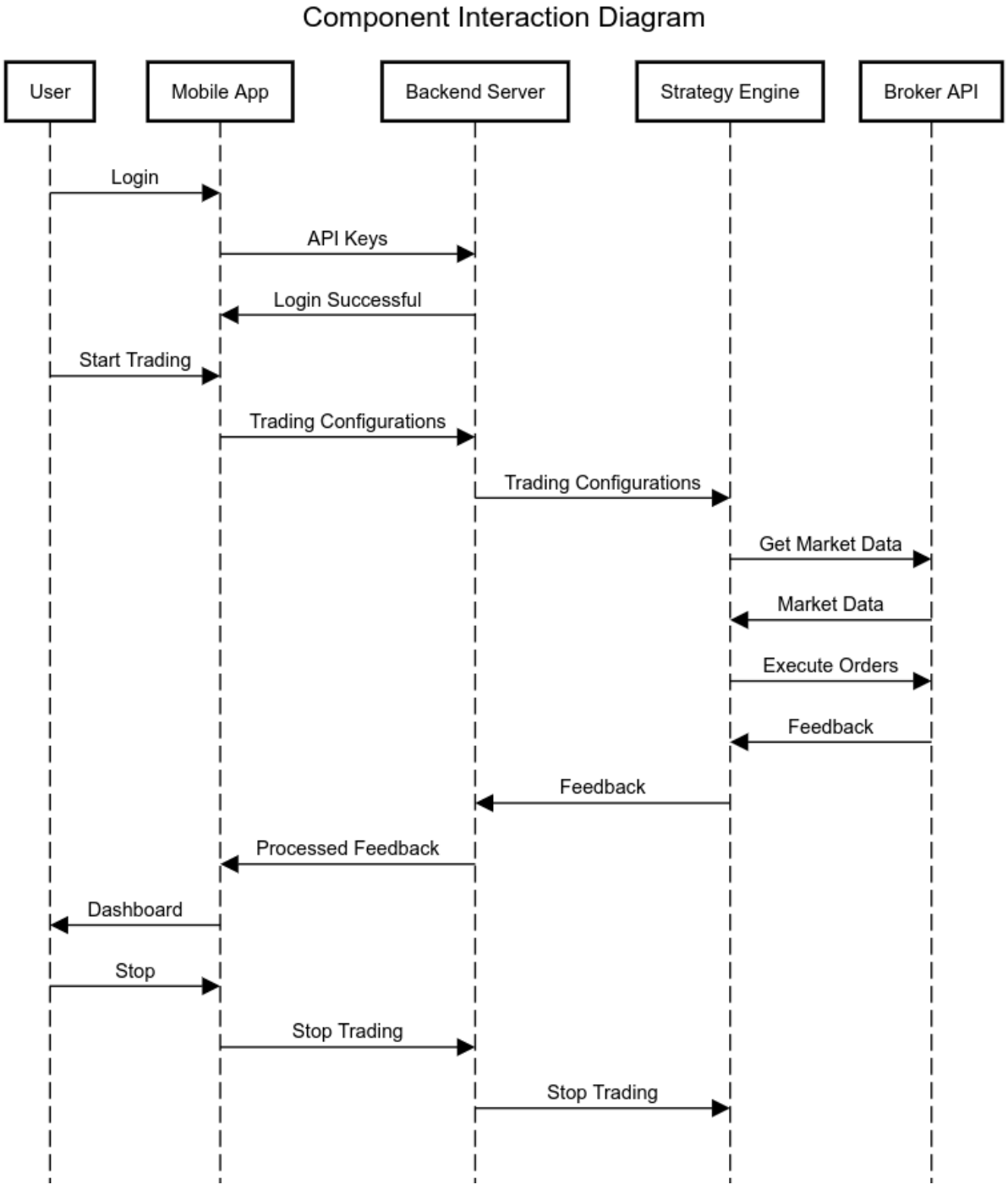

Figure 2 – Kryseos Data Flow Sequence Diagram.

3.4. System Requirements

Kryseos consists of a cross-platform mobile application and a Python-based backend server, each with specific software dependencies.

Mobile Application: Developed using Flutter (v3.10.5+) and Dart (v3.0+), the app targets Android 8.0+ and iOS 12.0+. Development is supported via Android Studio or VS Code. Core packages include `provider` for state management, `http` and `web_socket_channel` for communication, `shared_preferences` for local storage, and `fl_chart` for visualization.

Backend Server: Built with Python 3.10+, using `FastAPI` and `uvicorn` for asynchronous web handling. It leverages `SQLAlchemy (async)` for database operations, `cryptography` for secure API key storage, and integrates with broker APIs via `alpaca-trade-api` and `ccxt` or `requests`.

API Access: Users must provide valid API keys and secrets from Alpaca (for trading) and Binance (for market data) to enable real or paper trading functionality.

3.5. Summary

This chapter presented a comprehensive overview of the Kryseos system architecture, detailing its major components and the interaction between them. The system is designed as a modular, full-stack solution for real-time cryptocurrency trading, integrating a Flutter-based mobile application, a FastAPI-powered backend server, a local SQLite database, and third-party broker APIs such as Alpaca and Binance.

Each component plays a distinct role: the mobile app serves as the user interface, the backend server manages trading logic and secure communication, and the strategy engine executes technical and machine learning–based strategies. The system's architecture supports secure API key storage, concurrent user sessions, and live feedback delivery via WebSocket.

The described data flow ensures that user actions, market data processing, and trade execution are seamlessly integrated into a responsive and efficient automated trading experience. This architectural foundation enables Kryseos to be scalable, secure, and extensible for future enhancements.

CHAPTER 4
Trading Strategies

This chapter presents the trading strategies implemented in the system. Each strategy is described in terms of its rationale, parameters, logic flow, and performance characteristics.

4.1. Indicators

Indicators refer to mathematical calculations based on historical price, volume, or open interest data that traders use to analyze market trends and identify potential trading signals. They help in assessing momentum, trend direction, volatility, and market strength, providing a data-driven basis for making buy or sell decisions. Common examples include moving averages, oscillators, and trend-following tools. In this section, we will explain some of them and the most commonly used.

4.1.1. Simple Moving Average (SMA) Cross-Over

The Simple Moving Average (SMA) is a fundamental technical indicator that smooths out price data over a specified time window to highlight trends.

We compute two moving averages:

- A short-term SMA over the last 9 candles
- A long-term SMA over the last 21 candles

A **BUY** signal is generated when the 9-period SMA crosses above the 21-period SMA, indicating upward momentum in price. Conversely, a **SELL** signal is triggered when the 9-period SMA falls below the 21-period SMA, reflecting a potential trend reversal.

SMA Equation

The Simple Moving Average (SMA) is calculated as the average of the last N closing prices. The SMA formula [10] is given by:

$$\text{SMA}_t = \frac{1}{N} \sum_{i=0}^{N-1} P_{t-i} \tag{1}$$

Where:

- SMA_t is the Simple Moving Average at time t
- P_{t-i} is the closing price i periods before time t
- N is the number of periods (e.g., 9 or 21)

Illustration

Fig. 3 illustrates the core idea behind the SMA crossover. The blue line represents the 9-period Simple Moving Average (SMA), while the orange line represents the 21-period SMA. A **BUY** signal is triggered when the short-term SMA crosses above the long-term SMA. Similarly, a **SELL** signal is generated when the short-term SMA crosses below the long-term SMA.

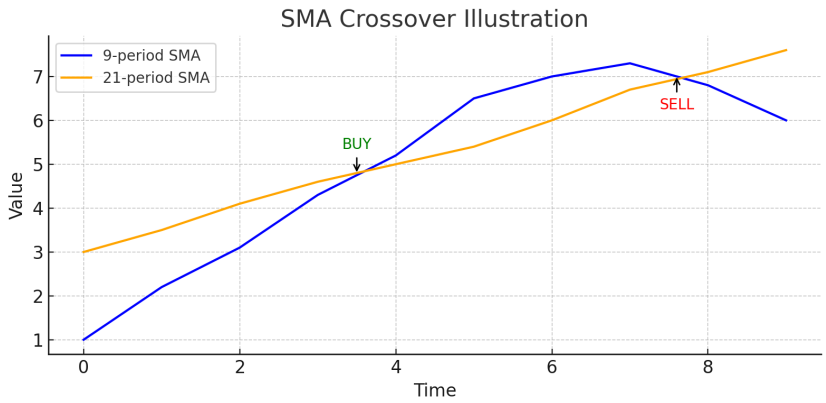

Figure 3 – SMA Cross-Over: Blue (9-period SMA), Orange (21-period SMA)

4.1.2. Moving Average Convergence Divergence (MACD)

The Moving Average Convergence Divergence (MACD) is a popular momentum indicator used in technical analysis to track changes in the strength, direction, and duration of a trend. It is derived from the relationship between two Exponential Moving Averages (EMAs) of the asset's closing price.

MACD is composed of the following elements:

- A 12-period EMA (fast EMA)
- A 26-period EMA (slow EMA)
- The MACD line, which is the difference between the 12-period and 26-period EMAs
- The Signal line, which is a 9-period EMA of the MACD line

A **BUY** signal occurs when the MACD line crosses above the Signal line, indicating potential upward momentum. Conversely, a **SELL** signal occurs when the MACD line crosses below the Signal line, suggesting potential downward movement. The equations below shows the the calculations for the MACD signal [11].

MACD Equation

$$\text{MACD}_t = \text{EMA}_{12}(P_t) - \text{EMA}_{26}(P_t) \tag{2}$$

$$\text{Signal}_t = \text{EMA}_9(\text{MACD}_t) \tag{3}$$

Where:

- P_t: the closing price at time t
- EMA_n: the exponential moving average over n periods
- MACD_t: MACD line value at time t
- Signal_t: Signal line value at time t

How EMA is Calculated

The Exponential Moving Average (EMA) gives more weight to recent prices, making it more responsive to recent market movements compared to the Simple Moving Average (SMA). The recursive formula for calculating the EMA is [12]:

$$\text{EMA}_t = \alpha \cdot P_t + (1 - \alpha) \cdot \text{EMA}_{t-1} \tag{4}$$

Where:

- EMA_t: EMA at time t
- P_t: current price
- $\alpha = \frac{2}{N+1}$: smoothing factor
- N: number of periods
- EMA_{t-1}: EMA of the previous time step

Table 7 shows the difference between EMA and SMA regarding weight distribution, responsiveness, calculation complexity, and use case.

Table 7 – Difference Between EMA and SMA

Feature	**SMA**	**EMA**
Weight Distribution	Equal weight to all data points	More weight to recent data
Responsiveness	Slower to respond to recent changes	Faster and more sensitive to price shifts
Calculation Complexity	Simple average	Recursive weighted formula
Use Case	Trend smoothing	Momentum and signal generation

Illustration

Fig. 4 illustrates the MACD crossover. The blue line represents the MACD line, and the orange line represents the Signal line. A **BUY** signal is generated when the MACD crosses above the Signal line. Similarly, a **SELL** signal is generated when it crosses below.

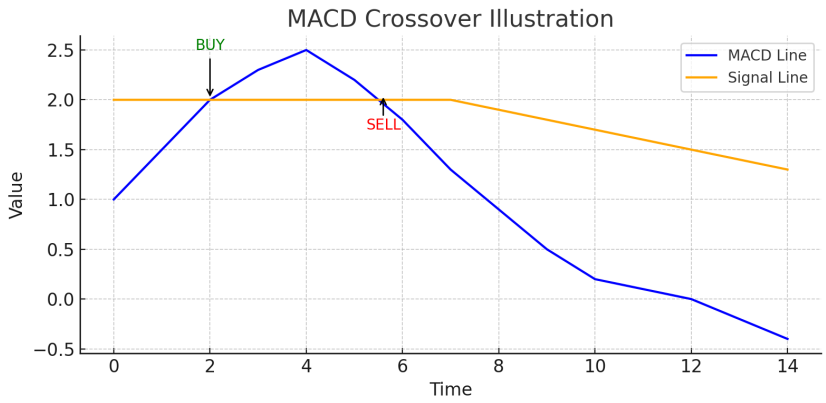

Figure 4 – MACD Cross-Over: Blue (MACD line), Orange (Signal line)

4.1.3. Relative Strength Index (RSI)

The Relative Strength Index (RSI) is a widely used momentum oscillator that measures the speed and magnitude of recent price changes to evaluate overbought or oversold conditions in the price of a financial asset. The RSI oscillates between 0 and 100.

Typically:

- RSI values above 70 suggest that an asset may be **overbought**
- RSI values below 30 suggest that an asset may be **oversold**

These thresholds are commonly used to anticipate potential reversal points in price direction.

Overbought means the price has risen quickly and may be too high, suggesting that buying momentum could slow down and a drop might follow.
Oversold means the price has fallen sharply, often due to heavy selling, and may be ready to bounce back as selling pressure eases. The following equation shows the calculation for RSI [13].

RSI Equation

$$\text{RSI}_t = 100 - \left(\frac{100}{1 + RS_t} \right) \tag{5}$$

Where:

- $RS_t = \dfrac{\text{Average Gain over } N \text{ periods}}{\text{Average Loss over } N \text{ periods}}$
- Gain is the positive price change between periods
- Loss is the absolute value of negative price change between periods
- The default period N is typically 14

Illustration

Fig. 5 illustrates how the RSI behaves relative to its common threshold levels. A **BUY** signal is generated when RSI crosses below the oversold level (30) and then rises, while a **SELL** signal is generated when RSI crosses above the overbought level (70).

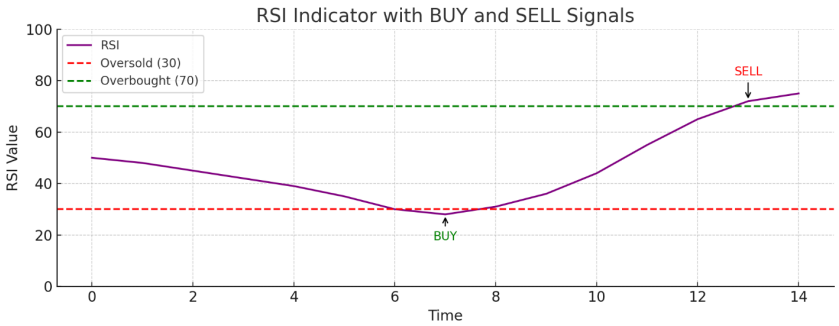

Figure 5 – RSI behavior relative to threshold levels 30 (oversold) and 70 (overbought)

4.1.4. Hull Moving Average (HMA)

The Hull Moving Average (HMA) is a trend-following indicator designed to provide a smoother and more responsive moving average by reducing lag without sacrificing accuracy. It is particularly useful for identifying the current trend direction with minimal noise.

HMA Calculation

The Hull Moving Average is calculated in the following steps:

1. Compute the WMA of the price with a half-length window
2. Compute the WMA with a full-length window
3. Subtract the full-length WMA from twice the half-length WMA
4. Apply a final WMA on the result using the square root of the original length as the period

The formula for HMA is [14]:

$$\text{HMA} = \text{WMA}\left(2 \cdot \text{WMA}\left(\frac{n}{2}\right) - \text{WMA}(n), \sqrt{n}\right) \tag{6}$$

Where:

- WMA is the Weighted Moving Average
- $\sqrt{n}$ is the smoothing period for the final step

How WMA is Calculated

The Weighted Moving Average (WMA) assigns more weight to recent prices, giving them greater influence on the average than older prices. This makes the WMA more responsive to recent market activity compared to a Simple Moving Average (SMA), which treats all periods equally [15].

The formula for the WMA over n periods is:

$$\text{WMA}_t = \frac{\sum_{i=1}^{n} i \cdot P_{t-n+i}}{\sum_{i=1}^{n} i} \tag{7}$$

Where:

- P_{t-n+i} is the price i periods into the n-length window ending at time t
- i is the weight assigned to each price, ranging from 1 to n

If the HMA is rising, it suggests upward momentum (**BUY** signal). If the HMA is falling, it indicates downward momentum (**SELL** signal).

Illustration

Fig. 6 shows the Hull Moving Average (HMA) line over a simulated price series. **BUY** and **SELL** signals are annotated based on changes in the direction of the HMA.

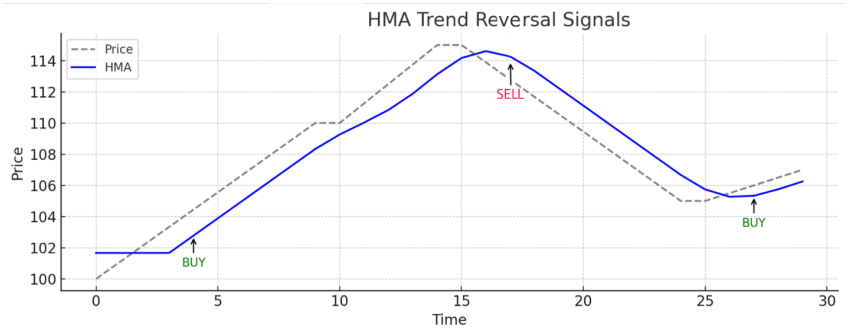

Figure 6 – Hull Moving Average (HMA) with trend-based BUY and SELL signals

4.1.5. Average Directional Index (ADX)

The Average Directional Index (ADX) is a technical indicator used to measure the *strength* of a trend. It does not indicate the direction of the trend (whether upward or downward), but simply how strong the current trend is. The ADX ranges from 0 to 100, where low values indicate a weak or sideways market, and high values indicate a strong trend.

How the ADX is Calculated

The ADX is calculated over a typical period of 14 days using the following five steps [16]:

Step 1: Calculate the True Range (TR)

$$\text{TR}_t = \max\left(\text{High}_t - \text{Low}_t, |\text{High}_t - \text{Close}_{t-1}|, |\text{Low}_t - \text{Close}_{t-1}|\right) \tag{8}$$

Where:

- High_t: today's high price
- Low_t: today's low price
- Close_{t-1}: previous day's close
- TR_t: largest movement today, accounting for price gaps

Step 2: Calculate +DM and −DM (Directional Movements)

$$+DM_t = \begin{cases} \text{High}_t - \text{High}_{t-1}, & \text{if greater than } (\text{Low}_{t-1} - \text{Low}_t) \text{ and positive} \\ 0, & \text{otherwise} \end{cases}$$

$$-DM_t = \begin{cases} \text{Low}_{t-1} - \text{Low}_t, & \text{if greater than } (\text{High}_t - \text{High}_{t-1}) \text{ and positive} \\ 0, & \text{otherwise} \end{cases}$$

Where:

- $+DM_t$: today's upward movement
- $-DM_t$: today's downward movement

Step 3: Smooth TR, +DM, and −DM using a 14-period average

- ATR_t = 14-period moving average of TR (Average True Range)
- $+DI_t = 100 \times \dfrac{\text{14-period average of } +DM}{\text{ATR}}$
- $-DI_t = 100 \times \dfrac{\text{14-period average of } -DM}{\text{ATR}}$

Where:

- $+DI_t$: strength of uptrend
- $-DI_t$: strength of downtrend

Step 4: Calculate the Directional Index (DX)

$$DX_t = 100 \times \frac{|+DI_t - -DI_t|}{+DI_t + -DI_t} \tag{9}$$

Where:

- DX_t: difference in directional strength
- Close to 0: balanced movement
- Close to 100: one direction dominates

Step 5: Calculate ADX

$$\text{ADX}_t = \text{14-period moving average of } DX_t \tag{10}$$

The resulting ADX value summarizes the trend strength:

- ADX < 20: weak or sideways trend
- ADX > 20: strong trend (can be up or down)

How ADX Helps in Trading

The ADX helps traders determine whether the market is trending strongly or moving sideways. A high ADX value (typically above 20) indicates a strong trend, either bullish or bearish. A low ADX value (below 20) suggests a weak market with choppy price action, where signals are more likely to fail.

Illustration

Fig. 7 shows an example of the ADX line. From time 0 to 20, ADX remains below 20, indicating a weak trend. After time 20, ADX rises above 20, signaling the development of a strong trend.

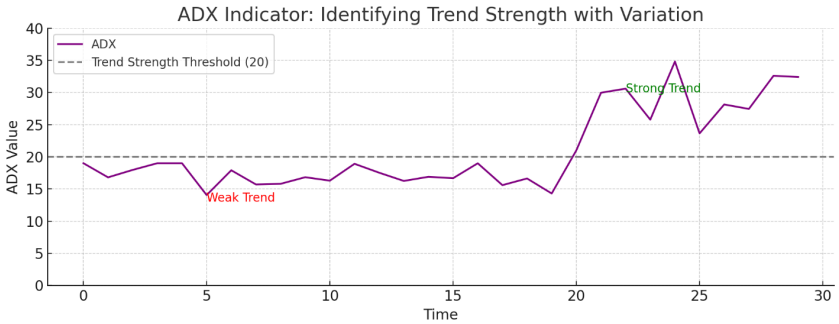

Figure 7 – ADX behavior across different trend strengths

4.2. Strategies

In this section, we present the trading strategies implemented and tested throughout this project. Each strategy is designed based on specific technical indicators, rules, or machine learning models to generate buy and sell signals. The goal is to evaluate their performance in different market conditions and compare them to a benchmark like SPY. We also highlight how each strategy balances return potential with risk management, offering insight into its practical effectiveness.

4.2.1. ML Strategy

The Machine Learning (ML) strategy represents the most advanced and important component of the trading system. Unlike rule-based strategies that rely on fixed technical indicators, the ML strategy dynamically learns patterns from historical data to predict future market behavior.

However, predicting price movement is not a straightforward machine learning task — it is fundamentally more complex than many standard problems due to the multifactorial and volatile nature of financial markets.

Price movement is influenced by a wide range of unpredictable and often unquantifiable factors, including:

- Economic indicators
- Political developments
- Global events
- Social sentiment

Moreover, financial data is inherently sequential and temporal, meaning that meaningful predictions require the model to correctly interpret the order and timing of past events. Successfully capturing such dependencies is critical when attempting to forecast the market's direction in the upcoming hours or days.

Developing this model required a rigorous and iterative process of experimentation, evaluation, and refinement to achieve optimal performance in live trading environments.

Why We Chose Recurrent Neural Networks (RNNs)

Given the sequential nature of market data and the complexity of predicting future price movements, we chose Recurrent Neural Networks (RNNs) as the foundational architecture for our machine learning model. RNNs are specifically designed to handle time-series data by maintaining a hidden state that captures information about previous inputs, making them particularly suitable for problems where temporal context and data dependencies across time are critical.

In contrast to traditional feed-forward networks, RNNs process input sequences element by element while preserving memory through internal loops, enabling them to model patterns over time. This makes them ideal for applications such as stock price prediction, language modeling, and speech recognition — where past inputs directly influence future outputs [17]. In our case, this allows the model to "remember" the recent behavior of the market (e.g., price trends, volume spikes, indicator movements) when making a decision about the future.

RNNs are advantageous because they:

- Can learn dependencies in time-series data,
- Are able to model non-linear temporal relationships,
- Use the same weights across time steps, which reduces the number of parameters and improves generalization.

Fig. 8 illustrates the structure of a Recurrent Neural Network (RNN), highlighting how the hidden state propagates through successive time steps to process sequential inputs. Each unit in the unrolled network shares the same set of parameters, enabling consistent learning across the temporal dimension.

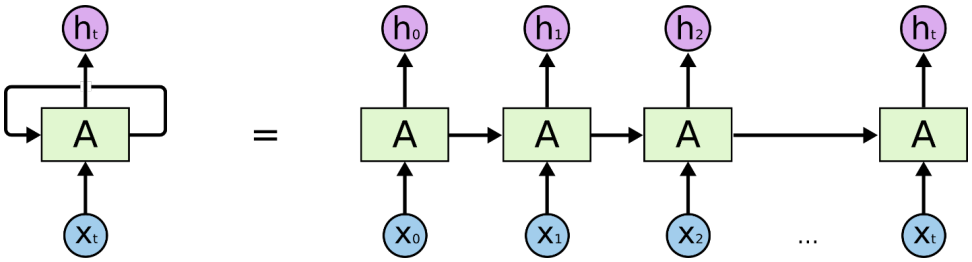

Figure 8 – Unrolled structure of a Recurrent Neural Network (RNN).

However, while RNNs are theoretically well-suited for sequential prediction, they suffer from well-documented limitations — most notably the vanishing and exploding gradient problems during training. These issues occur during Backpropagation Through Time (BPTT), when gradients are propagated over many time steps. If the gradients shrink exponentially (vanishing), the network fails to learn long-range dependencies. Conversely, if gradients grow uncontrollably (exploding), the training becomes unstable [18].

To address these limitations, we adopted the Long Short-Term Memory (LSTM) model, a specialized type of RNN designed to overcome the vanishing and exploding gradient problems while preserving the temporal modeling capability.

Using LSTM to Solve RNN's Limitations

Long Short-Term Memory (LSTM) networks extend traditional RNNs by introducing a memory cell and gating mechanisms that regulate the flow of information over time. These gates — the input gate, forget gate, and output gate — allow the network to selectively retain or discard information as needed, enabling it to learn both short-term and long-term dependencies more effectively.

The key innovation of LSTM lies in its ability to maintain constant error flow through time via its memory cell, which mitigates the vanishing gradient problem. Instead of letting gradients diminish exponentially, the cell state can propagate unchanged (or with minor linear transformations) across many time steps, allowing gradients to remain stable during backpropagation [19].

In particular:

- The **forget gate** controls whether information in the cell state should be retained or discarded.
- The **input gate** determines how much new information should be written into the memory.
- The **output gate** decides what part of the memory should influence the output and the next hidden state.

By learning to balance these three gates through training, LSTM networks can retain relevant information over long periods and discard irrelevant noise, a crucial advantage in financial modeling where patterns may span across many time steps and may be buried under noisy fluctuations [20].

Thanks to these properties, LSTMs have become a standard architecture for sequential modeling tasks in domains such as natural language processing, speech recognition, and financial time-series prediction.

Fig. 9: Internal architecture of an LSTM cell, showing the forget gate, input gate, and output gate, which regulate information flow and enable the network to learn long-term dependencies without suffering from vanishing or exploding gradients.

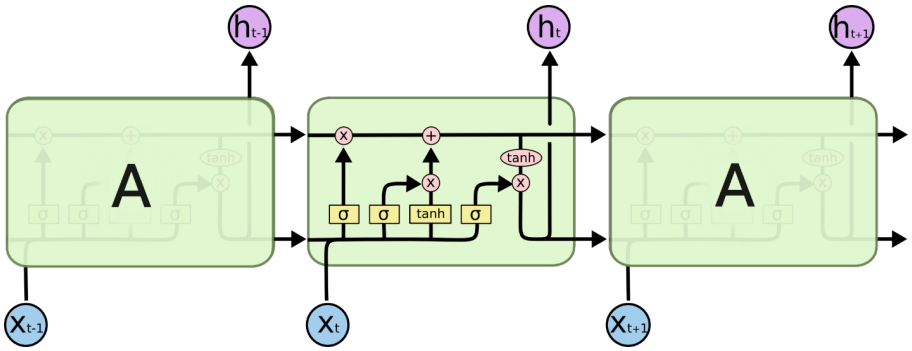

Figure 9 – Internal architecture of an LSTM cell.

Dataset Description

The dataset used for training the machine learning models was obtained from local CSV files containing historical cryptocurrency market data (Bitcoin and Ethereum). The Bitcoin data spans a time range from November 28, 2014, to May 8, 2025, covering more than ten years of trading activity, and the Ethereum data spans a time range from January 1, 2018, to September 26, 2024. Each file includes the essential market features required for time-series prediction: Open, High, Low, Close, and Volume prices.

This raw data was left unprocessed at the global level. Instead, each model independently handled its own data preprocessing steps—such as normalization, slicing, or indicator calculation—based on its specific requirements. This allowed for maximum flexibility and experimental variety when designing different model strategies throughout the project.

First Model: Predicting Next-Day Price Using Minmax-Scaled Closing Prices (BTC)

The first model aimed to learn a simple forecasting pattern: given the previous 10 days of closing prices, it would predict the closing price of the next day. If the predicted price was higher than the current day's price, the strategy would issue a Buy signal; otherwise, it would issue a Sell signal.

To prepare the data for training, input sequences were created using a sliding window of 10 consecutive closing prices. The target output was the closing price on the following day. Before being passed into the model, all values were scaled using Min-Max normalization to fall within the range [0, 1], ensuring stable gradient descent during training.

Each training sample followed this format: Input (X) = $[P_{t-10}, P_{t-9}, ..., P_{t-1}]$, Target (y) = P_t

The model was implemented in Keras using sequential architecture. It consisted of:

- Two stacked LSTM layers with 50 units each, the first returning sequences
- Dropout layers (20%) between them to reduce overfitting
- A final Dense output layer with a linear activation to perform regression on the next day's price

The model architecture is summarized as follows:

- Input Shape: (10, 1)
- Layer 1: LSTM (50 units, return_sequences=True)
- Dropout: 0.2
- Layer 2: LSTM (50 units, return_sequences=False)
- Dropout: 0.2
- Output Layer: Dense (1 unit, linear activation)

The model was compiled with the Adam optimizer, using Mean Squared Error (MSE) as the loss function, and Mean Absolute Error (MAE) as a performance metric. It was trained for 20 epochs using a data generator to feed batches of time-windowed sequences.

Trading Logic: After training, the model was used in a simple rule-based strategy. If the predicted next day's closing price was greater than the current day's closing price, the strategy executed a Buy; if it was lower, it executed a Sell.

Observation: To evaluate the model's performance, we tested it on a portion of data it had never seen before, specifically from April 1, 2022, to May 8, 2025. As shown in the prediction vs. actual price plot (see Fig. 10), the two lines appear closely aligned at first glance. This visual alignment gives the impression that the model is accurately forecasting future prices and performing well.

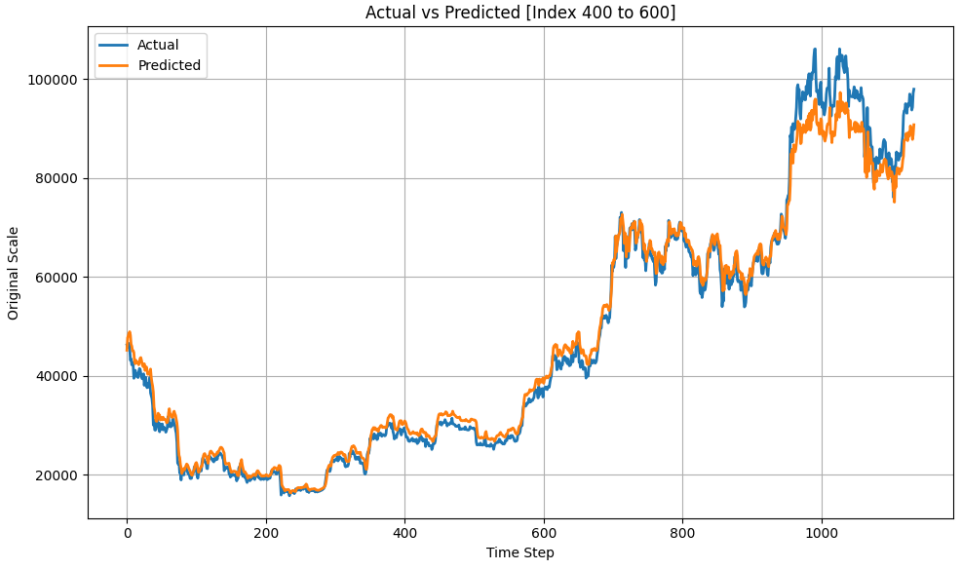

Figure 10 – Actual vs. predicted closing prices on unseen data.

However, this initial impression is misleading. When examining zoomed-in portions of the same plot (see Fig. 11), a consistent pattern emerges: the predicted line is not actually anticipating future prices. Instead, it is slightly lagging the actual line — often appearing as a shifted version of the ground truth. At times, it also appears vertically offset, either above or below the real price curve.

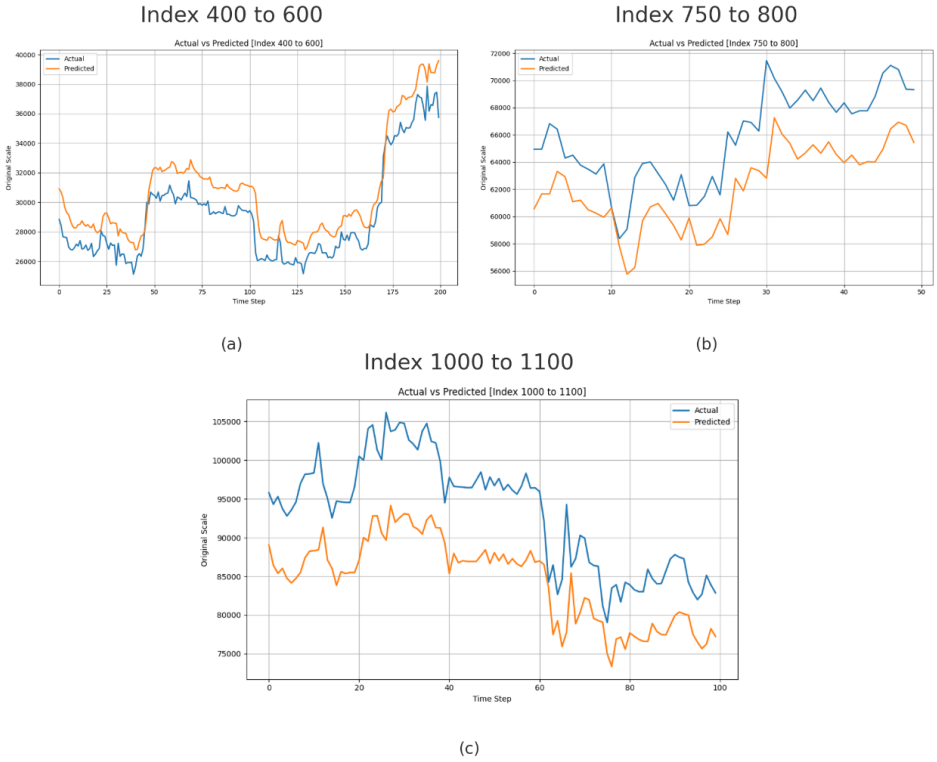

Figure 11 – Zoomed-in views of actual vs. predicted prices.

This behavior can be explained by the model's reliance on short-term continuity in the input data. Since prices — especially after Minmax scaling — typically change only slightly from day to day, the model learns that predicting something close to today's price is a strategy that minimizes the loss (e.g., RMSE or MSE). While this reduces numerical error, it does not lead to real predictive power. In effect, the model learns to 'copy' the input and shift it, rather than discovering meaningful patterns that drive actual price movements.

This revealed a key flaw in using raw prices for direct regression in trading strategies: although the model achieves low error metrics, it lacks true decision-making value. This observation motivated the development of alternative models based on movement encoding, richer features, and redefined targets in future iterations.

Model Strategy Back Test Results (Using LUMIBOT)

To evaluate the performance of the model in a realistic trading setup, we conducted a back test using the LUMIBOT framework. The strategy was tested over the period from April 10, 2023, to February 28, 2025. In each trade, the bot invested 50% of the total account balance, and it never held more than one open position at a time. A 0.25% transaction fee was applied only when buying the asset, based on the total amount purchased.

The backtest includes the SPY index as a benchmark by default, which serves as a standard reference for comparing the model's performance against a passive market investment. SPY is commonly used in trading benchmarks because it represents the overall performance of the U.S. stock market — specifically tracking the largest 500 publicly traded companies in the United States.

Performance Metrics The following Table 8 summarizes key risks and performance metrics observed during the back test:

Table 8 – Performance Metrics of the First Model

Metric	**Value**
Total Return	$\approx$ 55%
Maximum Drawdown	$\approx$ -16.56%
RoMaD (Return over Max Drawdown)	$\approx$ 3.32
Sharpe Ratio	$\approx$ 1.45
Sortino Ratio	$\approx$ 2.15

Metric Definitions

- **Sharpe Ratio:** Measures return relative to overall volatility. Higher is better; a value above 1.0 is generally considered strong.
- **Sortino Ratio:** Like Sharpe but only considers downside risk. It is more forgiving of positive volatility and better suited to strategies that aim to minimize losses.
- **RoMaD (Return over Maximum Drawdown):** Represents how much return was earned for each unit of drawdown. Higher RoMaD values indicate more efficient risk-return balance.

Back Test Performance Overview

The following visualizations illustrate the performance of the strategy in terms of cumulative returns, risk-adjusted performance, and month-by-month profitability. These charts provide a complete overview of how the model performed over time, both against the benchmark and in absolute terms (see Fig. 12).

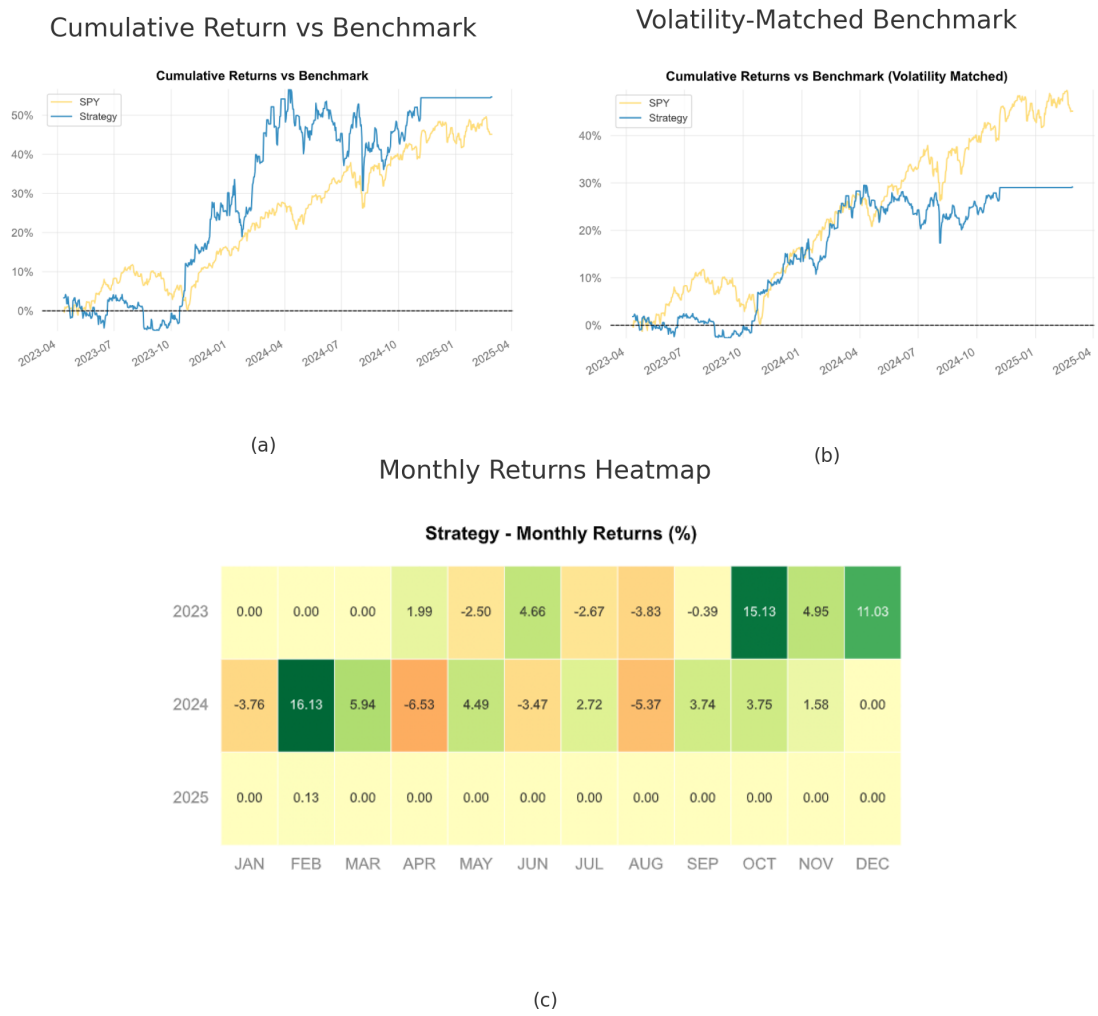

Figure 12 – Visual illustration shows the performance of the first model.

From Figure 12:

- **(a) Cumulative Return vs Benchmark** shows that the model generated a higher total return compared to SPY over the back test period.
- **(b) Volatility-Matched Benchmark** indicates that although the strategy made more profit than SPY, once adjusted for risk, SPY demonstrated more consistent and stable performance, suggesting it is preferable from a risk-adjusted perspective.
- **(c) Monthly Returns Heatmap** reveals periods of strong gains (e.g., February 2024 with +16.13%) alongside months of decline, reflecting a performance pattern that is profitable but includes volatility and drawdowns.

Second Model: Predicting Price Movement from Binary Encoded Trends (BTC)

After evaluating the first model, it became evident that using raw prices — even when scaled — as both input and prediction targets, introduces significant limitations. Although the model occasionally generated profits in back tests, its predictions were often lagging actual market movements and tended to mimic short-term price continuity rather than anticipate directional shifts. This behavior made the strategy unreliable and highly volatile, with sharp drawdowns and inconsistent signal quality.

To address the limitations observed in the first model, we shifted our modeling approach from price-based regression to movement-based classification. Instead of training the model to predict exact price values, we encoded the price action into a binary form. Each day was assigned a label: 1 if the closing price was higher than the previous day, and 0 otherwise. This transformed the forecasting task into a binary classification problem aimed at predicting the direction of the next day's movement, rather than its magnitude.

For the input, the model used a binary sequence representing the last 20 days of price movement, where each element was either 0 or 1 depending on whether the market went down or up. This abstraction reduced the influence of noise and scale, allowing the model to focus on recognizing directional patterns.

The model was implemented using stacked LSTM architecture in Keras. Its structure consisted of:

- **Input Shape:** (20, 1)
- **Layer 1:** LSTM with 50 units and `return_sequences=True`
- **Dropout:** 0.2
- **Layer 2:** LSTM with 50 units and `return_sequences=False`
- **Dropout:** 0.2
- **Output Layer:** Dense (1-unit, sigmoid activation) for binary classification

The model was compiled using the Adam optimizer with the binary cross-entropy loss function, and accuracy as the evaluation metric. It was trained for 20 epochs using a sequence generator that supplied batches of input-target pairs. The target was simply the movement label for the day immediately following the 20-day input sequence.

This formulation allowed the model to produce a probability value between 0 and 1. A prediction above 0.5 was interpreted as a signal that the market would rise the next day (BUY), and below 0.5 as a signal that it would fall or stay flat (SELL). This probability was then used to drive real-time trading decisions during back testing.

Model Strategy Back Test Results (Using LUMIBOT)

We conducted a back test using the LUMIBOT framework, following the same configuration and testing period used in the first model. This includes the same risk allocation, position constraints, and transaction fee structure to ensure consistent evaluation across all models.

Performance Metrics The following Table 9 summarizes key risk and performance metrics observed during the back test:

Table 9 – Performance Metrics of the second model

Metric	Value
Total Return	≈ 29%
Maximum Drawdown	≈ -12.87%
RoMaD (Return over Max Drawdown)	≈ 2.25
Sharpe Ratio	≈ 0.88
Sortino Ratio	≈ 1.40

Back Test Performance Overview

The following visualizations illustrate the performance of the strategy in terms of cumulative returns, risk-adjusted comparisons, and monthly profitability. These charts provide a complete overview of how the model performed throughout the testing period (see Fig. 13).

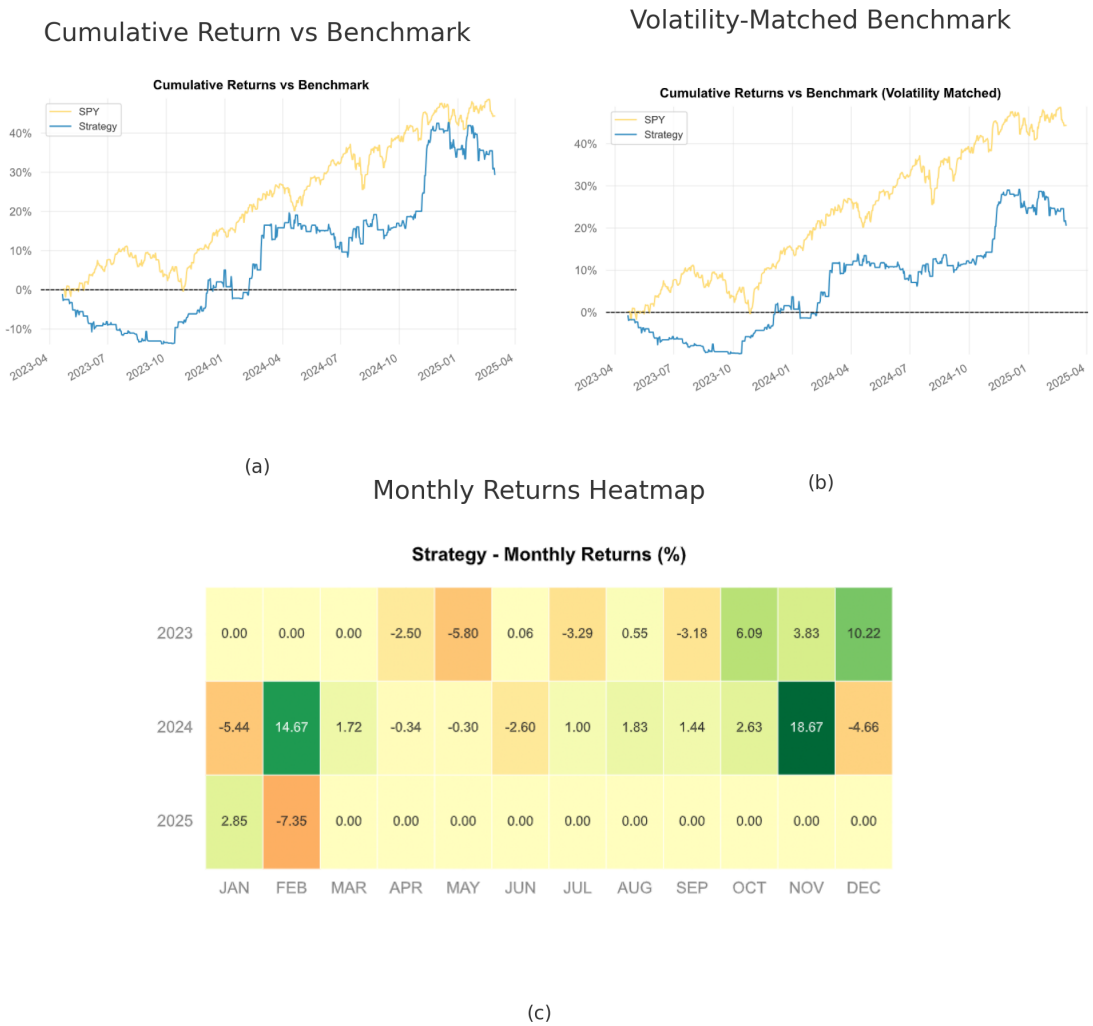

Figure 13 – Visual illustration shows the performance of the second model.

From Figure 13:

- **(a) Cumulative Return vs Benchmark** shows that the strategy underperformed the SPY benchmark in terms of total return.
- **(b) Volatility-Matched Benchmark** comparison indicates that although the strategy made some profit, the benchmark offered better stability and performance per unit of risk.
- **(c) Monthly Returns Heatmap** shows high variability in month-to-month results, with strong gains in November 2024 (+18.67%) and losses in February 2025 (-7.35%), suggesting a strategy that can capture trends but with noticeable volatility.

Third Model: Predicting Price Movement Using MACD, RSI, and Binary Labels (BTC)

While the second model introduced a valuable shift by treating the prediction task as binary classification based on price direction (up or down), its performance remained limited. The model relied solely on a sequence of binary values (0s and 1s) to represent past movements — a simplification that, although noise-resistant, lacked depth. The results indicated that such minimal representation may not be sufficient to capture more nuanced or emerging trends in the market. Its predictive signals were often too simplistic, leading to inconsistent trade quality and moderate volatility.

Building upon the limitations observed in the previous models, the third model introduces a more feature-rich approach to market prediction. Instead of relying solely on binary sequences or raw prices, this model incorporates a set of widely used technical indicators that are known to capture momentum and trend behavior. The goal was to help the model identify more meaningful and generalizable patterns in the data.

For each input sample, the model receives a 2D sequence of shape (10, 4), representing the last 10 days of the following features:

- **Target:** A binary label indicating if the price moved up from the previous day
- **MACD:** Measures momentum by comparing short- and long-term EMAs
- **MACD Signal Line:** A smoothed version of MACD to signal crossovers
- **RSI (Relative Strength Index):** Captures overbought or oversold market conditions

These features were standardized or min-max scaled as appropriate to ensure numerical stability during training. The model was trained as a binary classifier, with the objective of predicting whether the market would go up (label = 1) or not (label = 0) the next day.

The model architecture was implemented using a stacked LSTM design in Keras. Its structure consisted of:

- **Input Shape:** (10, 4)
- **Layer 1:** LSTM with 50 units, return_sequences=True
- **Dropout:** 0.2 (regularization)
- **Layer 2:** LSTM with 50 units, return_sequences=False
- **Dropout:** 0.2
- **Output Layer:** Dense (1-unit, sigmoid activation)

The model was compiled using the Adam optimizer, the binary cross-entropy loss function, and accuracy as the evaluation metric. It was trained for 20 epochs using a sequence generator that structured the time-series input for classification.

This architecture is better aligned with realistic market behavior, as it integrates both directional movement and trend-based signals in a format well-suited for sequence modeling.

Same as the previous model, it will produce a probability value between 0 and 1. A prediction above 0.5 was interpreted as a signal that the market would rise the next day, and if it's below 0.5 it predicts the market will go down the next day.

Model Strategy Back test Results (Using LUMIBOT)

Performance Metrics
The following Table 10 summarizes key risk and performance metrics observed during the back test:

Table 10 – Performance Metrics of the third model

Metric	Value
Total Return	≈ 30%
Maximum Drawdown	≈ -15.52%
RoMaD	≈ 1.9
Sharpe Ratio	≈ 0.81
Sortino Ratio	≈ 1.2

Back test Performance Overview

The following visualizations illustrate the performance of the strategy in terms of cumulative returns, risk-adjusted comparisons, and monthly profitability. These charts provide a complete overview of how the model performed throughout the testing period. (See Fig. 14)

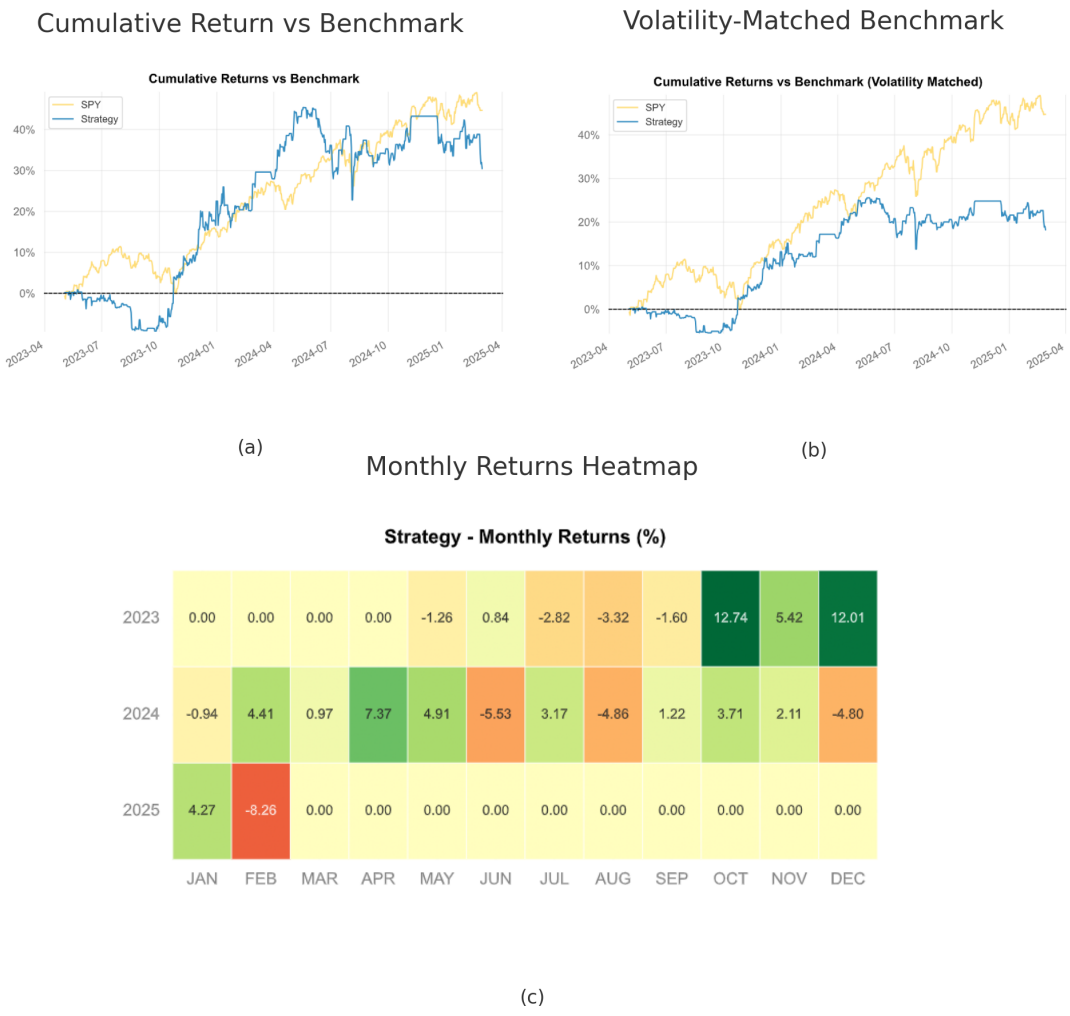

Figure 14 – Visual illustration shows the performance of the third model.

From Figure 14: (a) **Cumulative Return vs Benchmark** shows that the strategy kept pace with SPY for most of the test period and ended with a slightly lower total return. (b) **Volatility-Matched Benchmark** confirms that SPY outperformed the strategy on a risk-adjusted basis. (c) **Monthly Returns Heatmap** highlights periods of strong growth (e.g., October and December 2023 and April 2024) as well as setbacks like February 2025 (-8.26%), indicating a moderate drawdown.

Fourth Model: Predicting Daily Return Using MACD, RSI and return percentage (BTC)

The third model introduced technical indicators like MACD and RSI alongside a binary direction label, with the goal of helping the model detect trends more reliably than using movement sequences alone. However, in practice, it did not yield significant improvements over the second model. The binary representation of the target (up or down) still lacked nuance. It treated all upward or downward movements as equal, without capturing the strength or magnitude of price change — a limitation when distinguishing between weak signals and strong trends.

The fourth model is designed to predict the percentage change in closing price (daily return) based on recent sequences of technical indicators. Unlike the third model, which treated the target as a binary classification problem (up or down), this model reframes the task as a regression problem to estimate the actual magnitude of price movement. The goal is to produce a richer, more informative prediction that captures both direction and strength.

Each input to the model is a time series of shape (10, 4), representing the past 10 days of the following features:

- **Target:** The daily return percentage (used as the label)
- **MACD:** Measures short-term vs long-term momentum
- **MACD Signal:** A smoothed version of MACD
- **RSI:** Indicates overbought/oversold conditions

These features were scaled using a combination of `StandardScaler` (for MACD and MACD Signal) and `MinMaxScaler` (for RSI) prior to training. The input retains the same sequential format used in earlier models but with a more expressive target.

The model architecture follows a stacked LSTM design implemented in Keras:

- **Input Shape:** (10, 4)
- **Layer 1:** LSTM with 50 units, return_sequences=True
- **Dropout:** 0.2 (regularization)
- **Layer 2:** LSTM with 50 units, return_sequences=False
- **Dropout:** 0.2
- **Output Layer:** Dense (1-unit, linear activation)

The model was compiled using the Adam optimizer, with the Mean Squared Error (MSE) loss function, and Mean Absolute Error (MAE) as the evaluation metric. It was trained for 20 epochs using a data generator that prepared time-series windows from the indicator dataset.

By outputting a continuous value representing predicted return, the model offers more flexibility in generating trading signals (e.g., only buying if the return is significantly positive), and lays the foundation for more nuanced risk-control strategies.

The trading logic is straightforward and based directly on the model's output value, which represents the predicted percentage return for the next day. If the model predicts a positive return, the strategy executes a BUY signal and opens a position using 50% of the available balance. If the predicted return is negative, the strategy issues a SELL signal and exit any open position. This allows the bot to act on the direction and strength of the anticipated move, adapting trading decisions to the market's expected behavior.

Model Strategy Back Test Results (Using LUMIBOT)

In this model we conducted a back test using the LUMIBOT framework, following the same configuration but a different period from May 1, 2023, to February 28, 2025.

Performance Metrics

The following Table 11 summarizes key risk and performance metrics observed during the back test:

Table 11 – Performance Metrics of the fourth model

Metric	**Value**
Total Return	$\approx 61\%$
Maximum Drawdown	$\approx -20.76\%$
RoMaD	≈ 2.94
Sharpe Ratio	≈ 1.43
Sortino Ratio	≈ 2.21

Backtest Performance Overview

The following visualizations illustrate the performance of the strategy in terms of cumulative returns, risk-adjusted comparisons, and monthly profitability. These charts provide a complete overview of how the model performed throughout the testing period. (See Fig. 15)

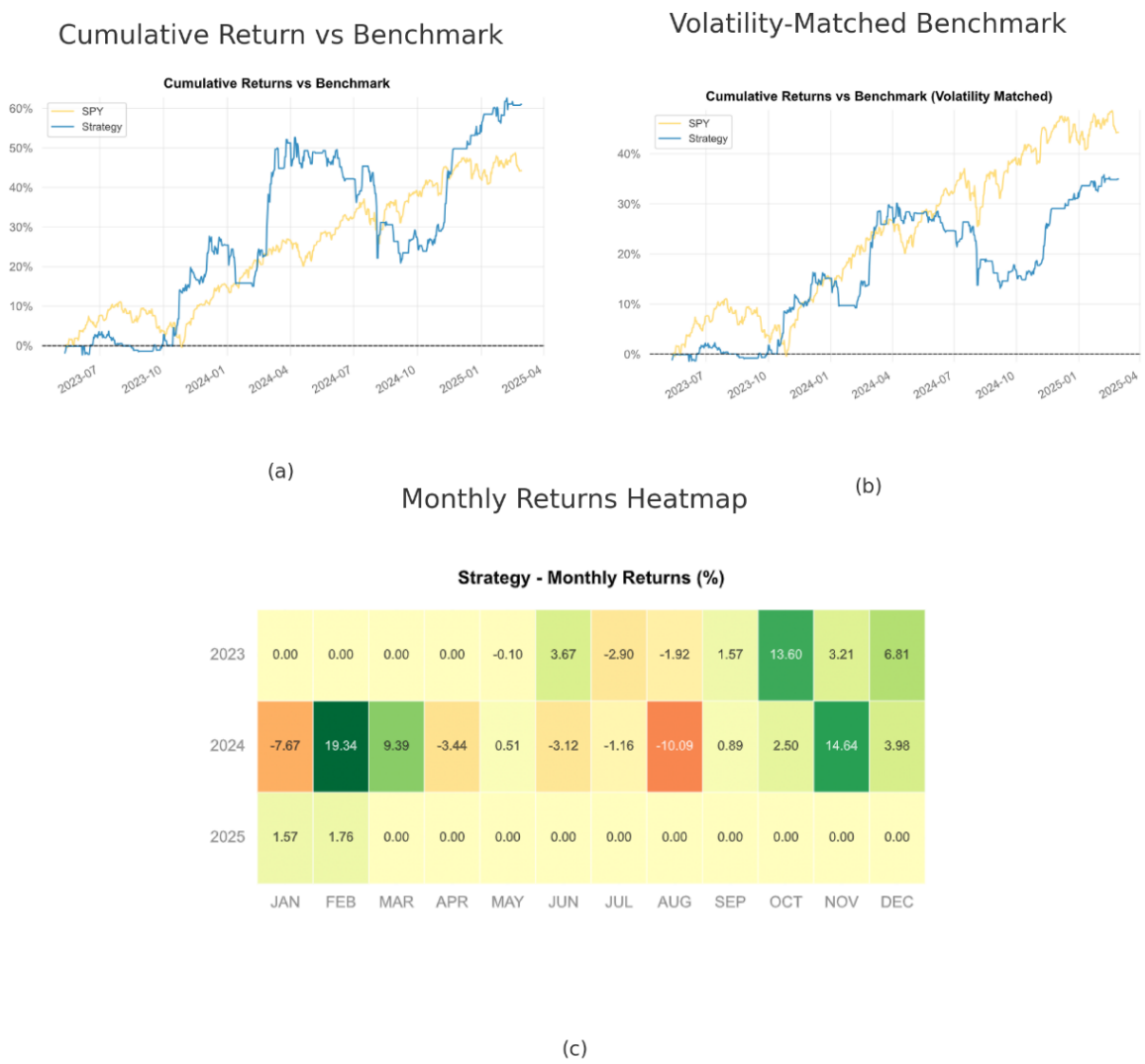

Figure 15 – Visual illustration shows the performance of the fourth model.

From Figure 15:

- (a) **Cumulative Return vs Benchmark** shows that the strategy outperformed SPY, achieving over 60% return versus SPY's 45%. However, it exhibited more volatile price swings and sharper drawdowns.
- (b) **Volatility-Matched Benchmark** confirms that SPY had a smoother trajectory and, when normalized for risk, was occasionally more consistent in returns.
- (c) **Monthly Returns Heatmap** displays strong growth in February 2024 (+19.34%) and November 2024 (+14.64%), as well as losses in August 2024 (-10.09%) and January 2024 (-7.67%). The beginning of 2025 shows stable, positive growth.

Fifth Model: Predicting Return Using Indicators with RSI-Guided Filtering (Hybrid-BTC)

The fifth model builds on the fourth model's regression-based design, which predicts the daily percentage return using technical indicators. While the previous model relied solely on its predicted return to generate trading signals, this model introduces a hybrid approach: it combines the model's output with a classic technical rule to improve selectivity and reduce false signals.

The model is trained using three input features over a lookback window of N days:

- **Target:** The daily return percentage (used as the label)
- **MACD:** Measures short-term vs long-term momentum
- **MACD Signal:** A smoothed version of MACD

N: number of days

Unlike the previous model, RSI (Relative Strength Index) is not part of the training data. Instead, it is calculated dynamically during trading and used as a filtering condition. The strategy executes trades based on the following logic:

- If the model predicts a positive return and the RSI is below a certain value, a **BUY** signal is triggered.
- If the model predicts a negative return and the RSI is above a certain value, a **SELL** signal is triggered.

This design allows for a dynamic decision-making process where the model forecasts return strength, and RSI serves as a secondary filter to confirm whether market momentum justifies taking a position. Different RSI thresholds and lookback windows were tested to optimize this behavior, with the final configuration selected based on backtest performance.

Model Architecture and Training

The model architecture follows a stacked LSTM design implemented in Keras:

- **Input Shape:** (N, 3)
- **Layer 1:** LSTM with 50 units, `return_sequences=True`
- **Dropout:** 0.2
- **Layer 2:** LSTM with 50 units, `return_sequences=False`
- **Dropout:** 0.2
- **Output Layer:** Dense (1-unit, linear activation)

The model was compiled using the Adam optimizer, with the Mean Squared Error (MSE) loss function and Mean Absolute Error (MAE) as a secondary evaluation metric. It was trained for 20 epochs using a time-series generator that provided rolling N-day windows of standardized MACD data and price return targets.

This design allows the model to forecast the strength and direction of price movement while letting RSI control trade execution. The result is a hybrid strategy that combines learned trend forecasting with classic overbought/oversold filtering — producing more reliable and selective trading signals.

Model Strategy Back test Results (Using LUMIBOT)

In this model, we conducted a backtest using the LUMIBOT framework, following the same testing period as the previous model. The RSI thresholds used were 25 as the lower bound (for oversold conditions) and 75 as the upper bound (for overbought conditions). The model was trained on a 10-day lookback window ($N = 10$).

Performance Metrics

The following Table 12 summarizes key risk and performance metrics observed during the back test:

Table 12 – Performance Metrics of the Fifth Model

Metric	**Value**
Total Return	$\approx$ 150%
Maximum Drawdown	$\approx$ -12.12%
RoMaD	$\approx$ 12.37
Sharpe Ratio	$\approx$ 2.43
Sortino Ratio	$\approx$ 4.3

Back test Performance Overview

The following visualizations illustrate the performance of the strategy in terms of cumulative returns, risk-adjusted comparisons, and monthly profitability. These charts provide a complete overview of how the model performed throughout the testing period (See Fig. 16).

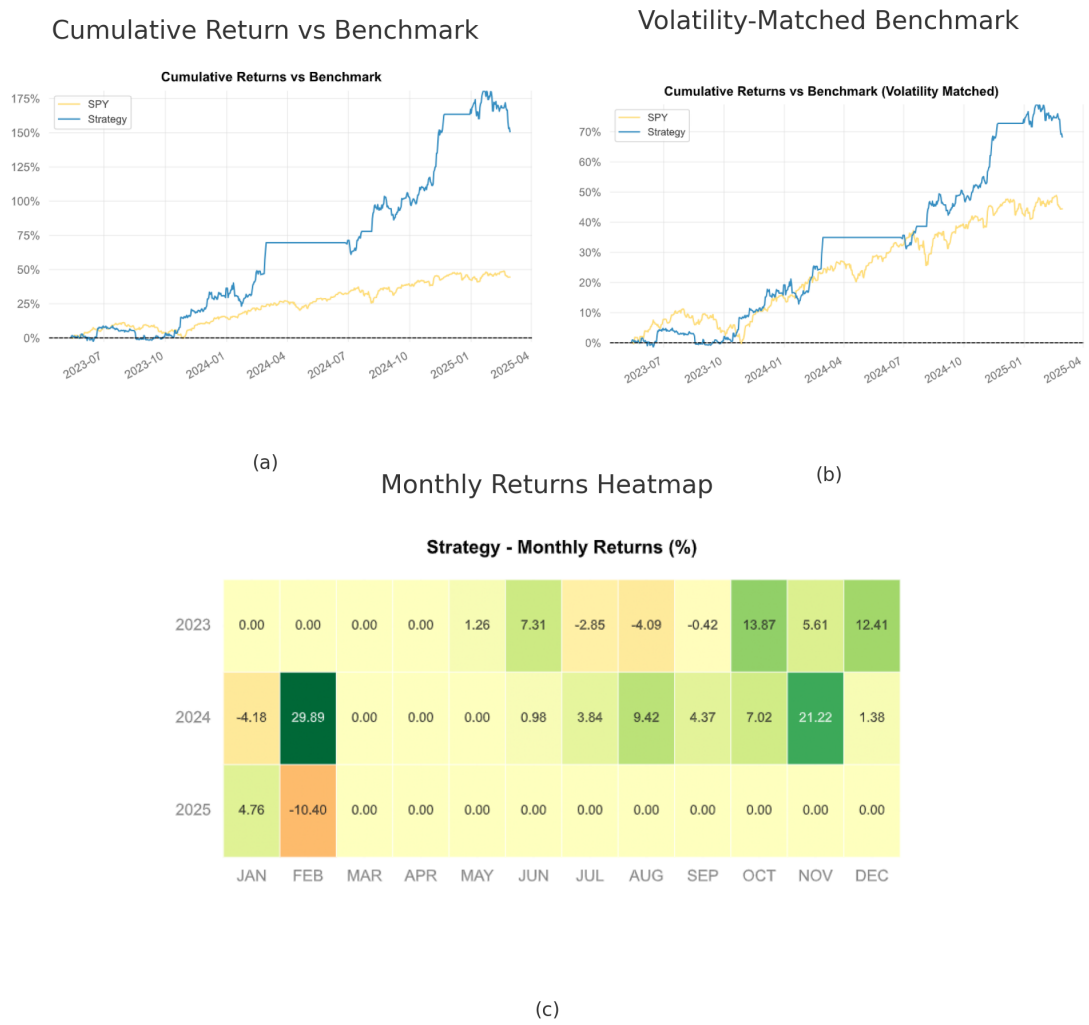

Figure 16 – Visual illustration shows the performance of the fifth model.

From Figure 16:

- **(a) Cumulative Return vs Benchmark:** The strategy achieved significantly higher returns than SPY, ending with 150% versus SPY's 44%.
- **(b) Volatility-Matched Benchmark:** Confirms that while the strategy is more volatile, it generated superior risk-adjusted performance.
- **(c) Monthly Returns Heatmap:** Highlights exceptional returns in February 2024 (+29.89%) and steady gains in early 2025, with occasional dips like February 2025 (-10.40%).

Given the strong performance demonstrated by the fifth model — both in terms of total return and risk-adjusted metrics — we identified it as the most promising approach among all the models tested. Its hybrid structure, which combines model-driven return predictions with RSI-based filtering, produced reliable and selective signals that translated into consistent trading profitability.

Based on these results, we decided to continue developing and refining this model. Our focus shifted toward experimenting with different parameters such as RSI thresholds and lookback window sizes in order to further optimize performance and identify the most effective variant of this strategy.

Optimized Fifth Model Back test Results (Best Parameters-BTC)

To further improve the fifth model, we experimented with a range of configurations by adjusting key parameters such as the RSI thresholds and the lookback window size. Multiple versions of the model were tested and evaluated based on their trading performance. After careful comparison, we selected the best-performing variant, which used an RSI threshold of 25 for oversold and 83 for overbought, along with a lookback window size of 25 days. The following section presents the back test results of this optimized model configuration.

Back test Performance Overview

The following Table 13 summarizes key risk and performance metrics observed during the back test:

Table 13 – Performance Metrics of the Optimized Fifth Model

Metric	**Value**
Total Return	$\approx$ 162%
Maximum Drawdown	$\approx$ -8.44%
RoMaD	$\approx$ 19.19
Sharpe Ratio	$\approx$ 3.65
Sortino Ratio	$\approx$ 8.75

The following visualizations illustrate the performance of the best-performing configuration of the fifth model, including total returns, risk-adjusted comparisons, and monthly profitability (See Fig. 17).

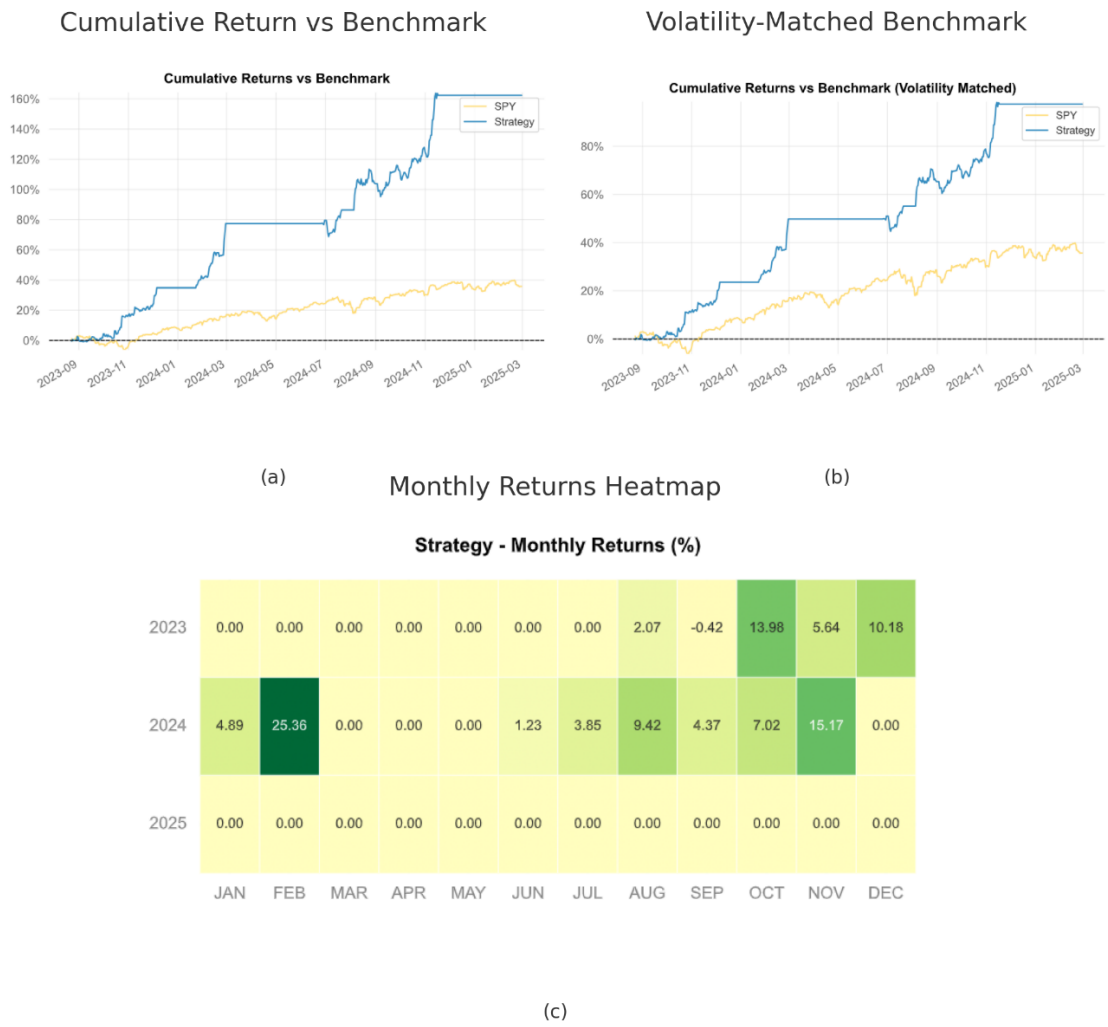

Figure 17 – Visual illustration shows the performance of the optimized fifth model.

From Figure 17:

- **(a) Cumulative Return vs Benchmark:** Shows strong outperformance of the strategy relative to SPY.
- **(b) Volatility-Matched Benchmark:** Confirms that even when adjusting for risk, the strategy maintained superior returns.
- **(c) Monthly Returns Heatmap:** Demonstrates consistent gains across multiple months, with occasional dips managed well within acceptable drawdown levels.

The performance metrics of this optimized strategy clearly demonstrate its effectiveness. With a total return of 162%, it outperformed all previously tested models and benchmarks by a significant margin. More impressively, it achieved this result while maintaining a maximum drawdown of only 8.44%, indicating strong risk control and stable performance even during volatile periods. The high Sharpe ratio and Sortino ratio reflect not just profitability but consistent risk-adjusted gains, and the exceptionally high RoMaD value shows that the strategy was able to generate substantial returns with relatively minimal downside.

These results confirm that the hybrid structure — combining learned return forecasts with RSI-based filtering — not only boosts return but also helps avoid high-risk entries, making this the most balanced and reliable strategy in the entire suite.

Extension to Ethereum (ETH): Model Retraining and Results

After optimizing the fifth model for Bitcoin (BTC), we sought to evaluate its generalizability by applying it to Ethereum (ETH). Initially, we tested the same BTC-trained model directly on ETH data. While the results were acceptable and showed decent performance, they fell short of the strategy's full potential. This outcome was expected, as ETH exhibits different price dynamics, volatility behavior, and market structure compared to BTC.

To address this, we retrained the model specifically on ETH historical data, using the same architecture and hybrid logic — combining MACD-based trend prediction with RSI filtering. As anticipated, the ETH-specific model delivered markedly better performance, confirming that adapting the training data to the asset's unique characteristics is essential for maximizing strategy effectiveness.

To fine-tune the strategy for Ethereum, we experimented with multiple parameter configurations, including different RSI thresholds and lookback window sizes. After evaluating the results, the best-performing configuration used a 10-day lookback window and RSI thresholds of 15 for oversold and 80 for overbought. We conducted the back test using the LUMIBOT framework over the period from June 26, 2023 to February 28, 2025. This configuration demonstrated a strong balance between profitability and risk control, further validating the adaptability of the model to different assets when appropriately retrained.

Back test Performance Overview

The following Table 14 summarizes key risk and performance metrics observed during the back test:

Table 14 – Performance Metrics of the ETH model

Metric	**Value**
Total Return	≈ 80%
Maximum Drawdown	≈ -9.43%
RoMaD	≈ 8.48
Sharpe Ratio	≈ 3.0
Sortino Ratio	≈ 6.03

The following visualizations present the ETH model's performance, including cumulative returns, risk-adjusted comparisons, and monthly return distribution. (See Fig. 18)

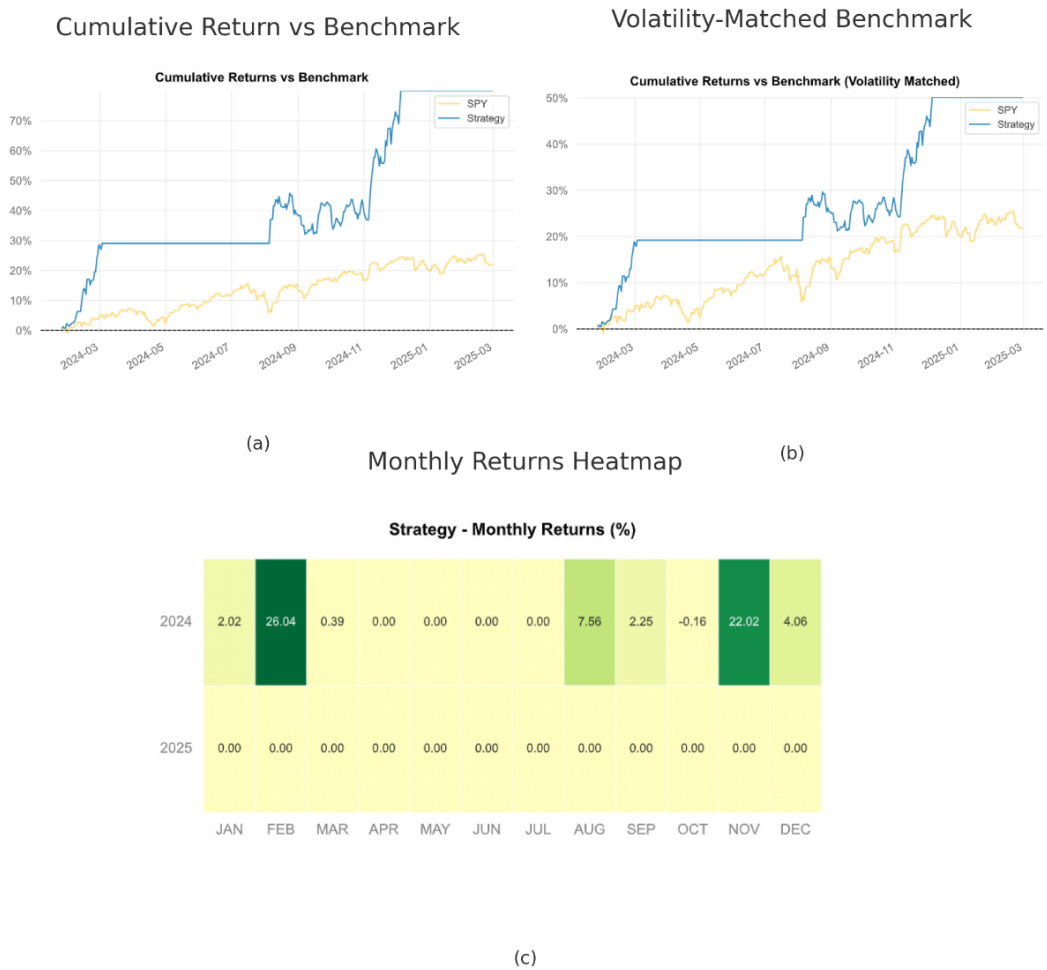

Figure 18 – Visual illustration shows the performance of the ETH model.

From Figure 18: (a) Cumulative Return vs Benchmark shows that the ETH-trained strategy consistently outperformed SPY, especially in Q4 2024 and early 2025. (b) Volatility-Matched Benchmark confirms that the strategy retained superior returns even after adjusting for risk. (c) Monthly Returns Heatmap shows strong performance in February (26.04%) and November (22.02%), with periods of no trading or low volatility being avoided — a sign of the model's selectivity and precision.

4.2.2. SMA Cross-Over Strategy

This strategy applies the crossover method using two SMAs: a 9-period and a 21-period, each based on 16-hour candles. The strategy continuously monitors these averages and generates a BUY signal when the short-term (9-period) SMA crosses above the long-term (21-period) SMA, signaling a potential upward trend. A SELL signal is triggered when the short-term SMA crosses below the long-term SMA, indicating a possible reversal.

Back Test Performance Overview

The following Table 15 summarizes key risk and performance metrics observed during the back test:

Table 15 – Performance Metrics of the SMA Strategy

Metric	Value
Total Return	≈ 94%
Maximum Drawdown	≈ -12.97%
RoMaD	≈ 7.2
Sharpe Ratio	≈ 1.98
Sortino Ratio	≈ 3.33

The following visualizations present the SMA strategy performance, including cumulative returns, risk-adjusted comparisons, and monthly return distribution. (See Fig. 19)

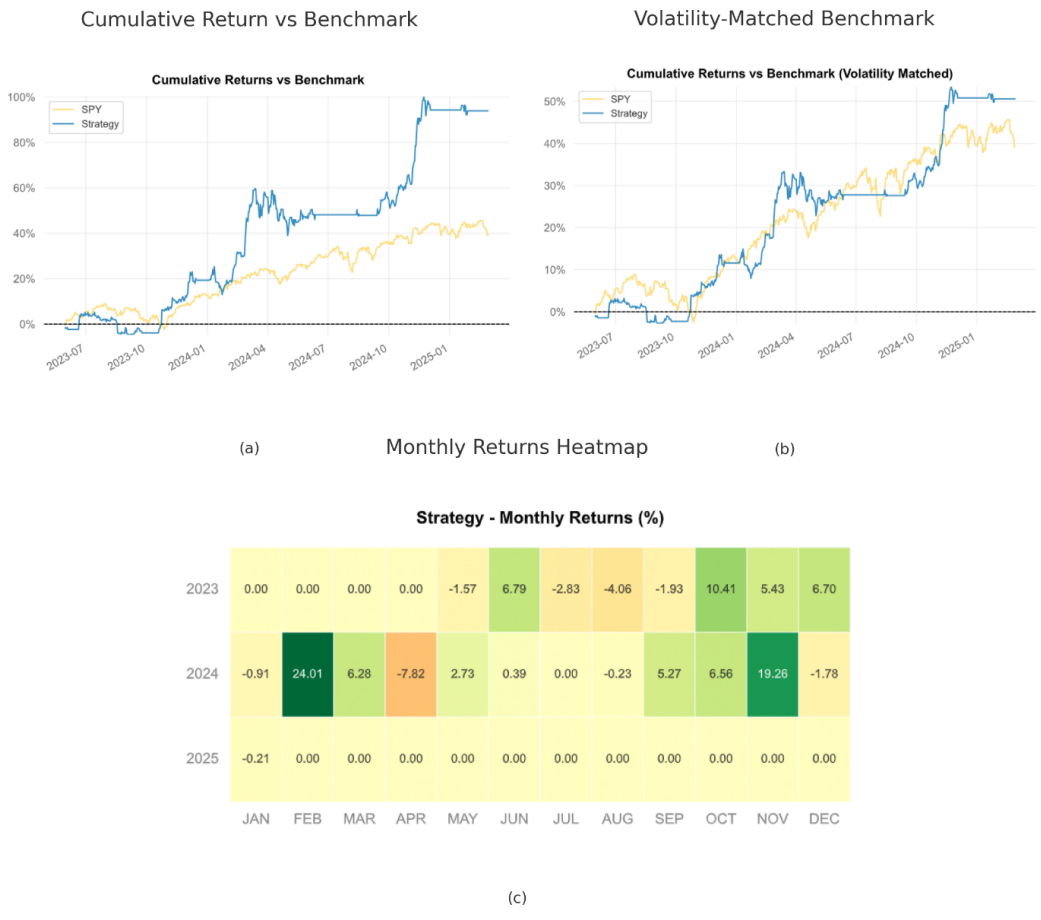

Figure 19 – Visual illustration shows the performance of the SMA strategy.

4.2.3. MACD Strategy

This strategy applies the crossover method using two exponential moving averages: a 12-period EMA and a 26-period EMA, each computed from 480-minute candles. The MACD line is calculated as the difference between these two EMAs, and a 9-period EMA of the MACD line is used as the Signal line. A BUY signal is generated when the MACD line crosses above the Signal line, indicating potential bullish momentum. A SELL signal is triggered when the MACD line crosses below the Signal line, suggesting a possible trend reversal.

Back Test Performance Overview

The following Table 16 summarizes key risk and performance metrics observed during the back test:

Table 16 – Performance Metrics of the MACD Strategy

Metric	**Value**
Total Return	≈ 77%
Maximum Drawdown	≈ -13.32%
RoMaD	≈ 5.7
Sharpe Ratio	≈ 1.65
Sortino Ratio	≈ 2.81

The following visualizations present the MACD strategy performance, including cumulative returns, risk-adjusted comparisons, and monthly return distribution. (See Fig. 20)

4.2.4. RSI Strategy

This strategy combines the Relative Strength Index (RSI) with a 50-period Simple Moving Average (SMA) to enhance signal reliability and reduce noise. It uses 10-hour candles to compute a 14-period RSI and overlays a 50-period SMA as a trend filter. A BUY signal is generated when the RSI drops below 40 in an uptrend or below 30 in a downtrend, indicating a potential reversal from an oversold condition. Conversely, a SELL signal is triggered when the RSI rises above 80 in an uptrend or above 70 in a downtrend, suggesting possible exhaustion of buying momentum. This hybrid momentum-trend strategy seeks to capture profitable entries while filtering out false signals using dynamic RSI thresholds based on market context.

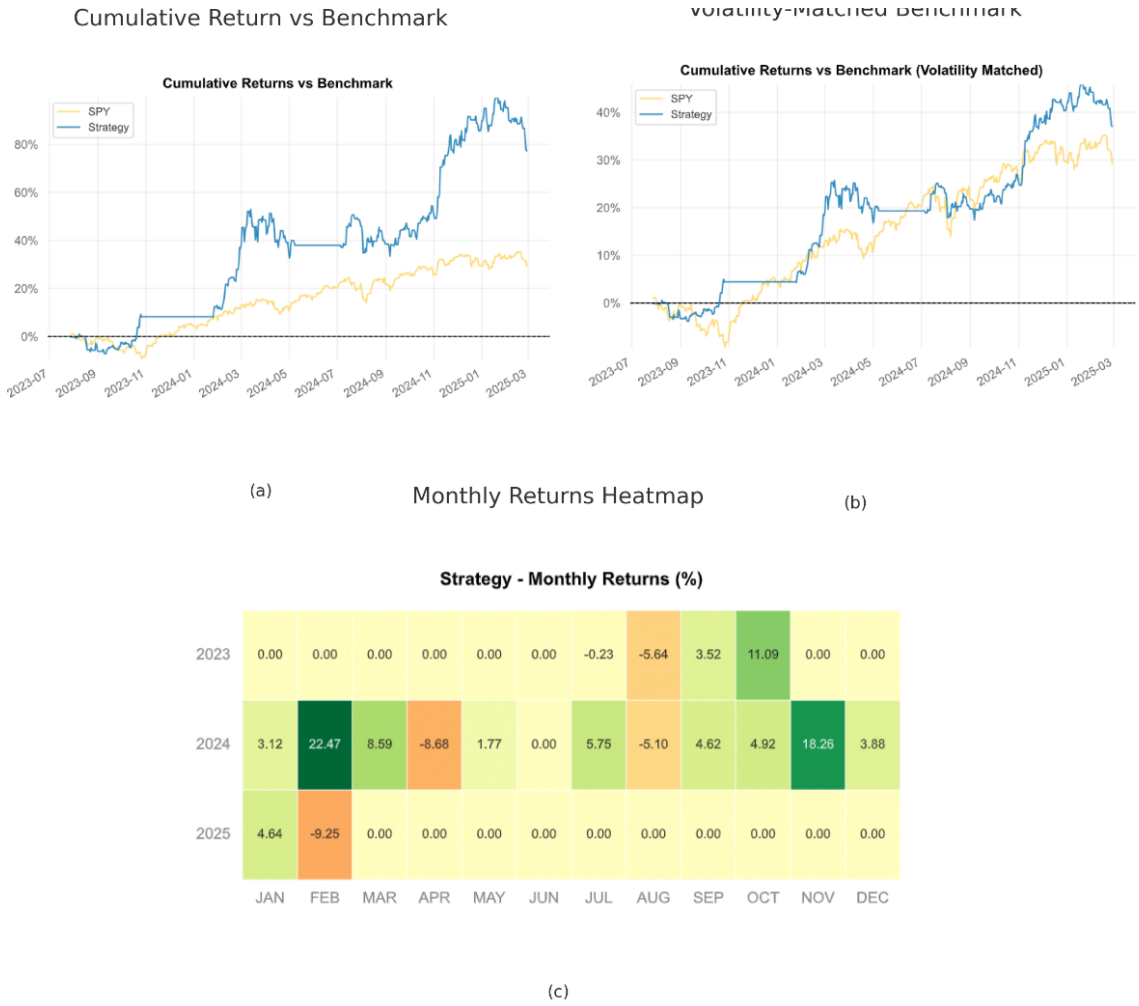

Figure 20 – Visual illustration shows the performance of the MACD strategy.

Back Test Performance Overview

The following Table 17 summarizes key risk and performance metrics observed during the back test:

Table 17 – Performance Metrics of the RSIandSMA50 Strategy

Metric	Value
Total Return	≈ 86%
Maximum Drawdown	≈ -10.93%
RoMaD	≈ 7.88
Sharpe Ratio	≈ 1.85
Sortino Ratio	≈ 3.24

The following visualizations present the RSIandSMA50 strategy performance, including cumulative returns, risk-adjusted comparisons, and monthly return distribution. (See Fig. 21)

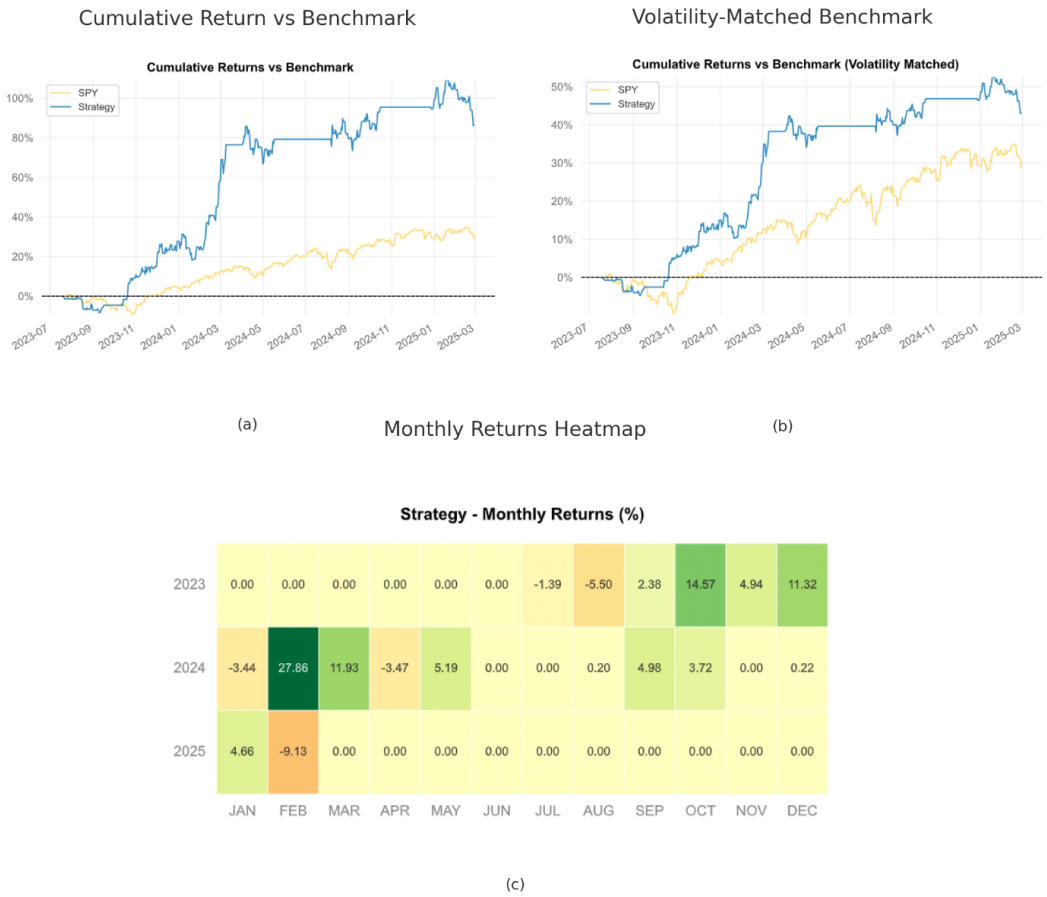

Figure 21 – Visual illustration shows the performance of the RSIandSMA50 strategy.

4.2.5. HMA Strategy

This strategy applies a directional crossover logic based on the Hull Moving Average (HMA), a smoothed and responsive trend-following indicator. The strategy computes the HMA using 1-day candles and evaluates price direction by comparing the most recent HMA value to its value two bars earlier. A BUY signal is generated when the HMA is rising, suggesting growing upward momentum. Conversely, a SELL signal is triggered when the HMA is falling, indicating potential downward pressure.

Back Test Performance Overview

The following Table 18 summarizes key risk and performance metrics observed during the back test:

Table 18 – Performance Metrics of the HMA strategy

Metric	Value
Total Return	≈ 80%
Maximum Drawdown	≈ -27.03%
RoMaD	≈ 2.96
Sharpe Ratio	≈ 1.46
Sortino Ratio	≈ 2.22

The following visualizations present the HMA strategy performance, including cumulative returns, risk-adjusted comparisons, and monthly return distribution. (See Fig. 22)

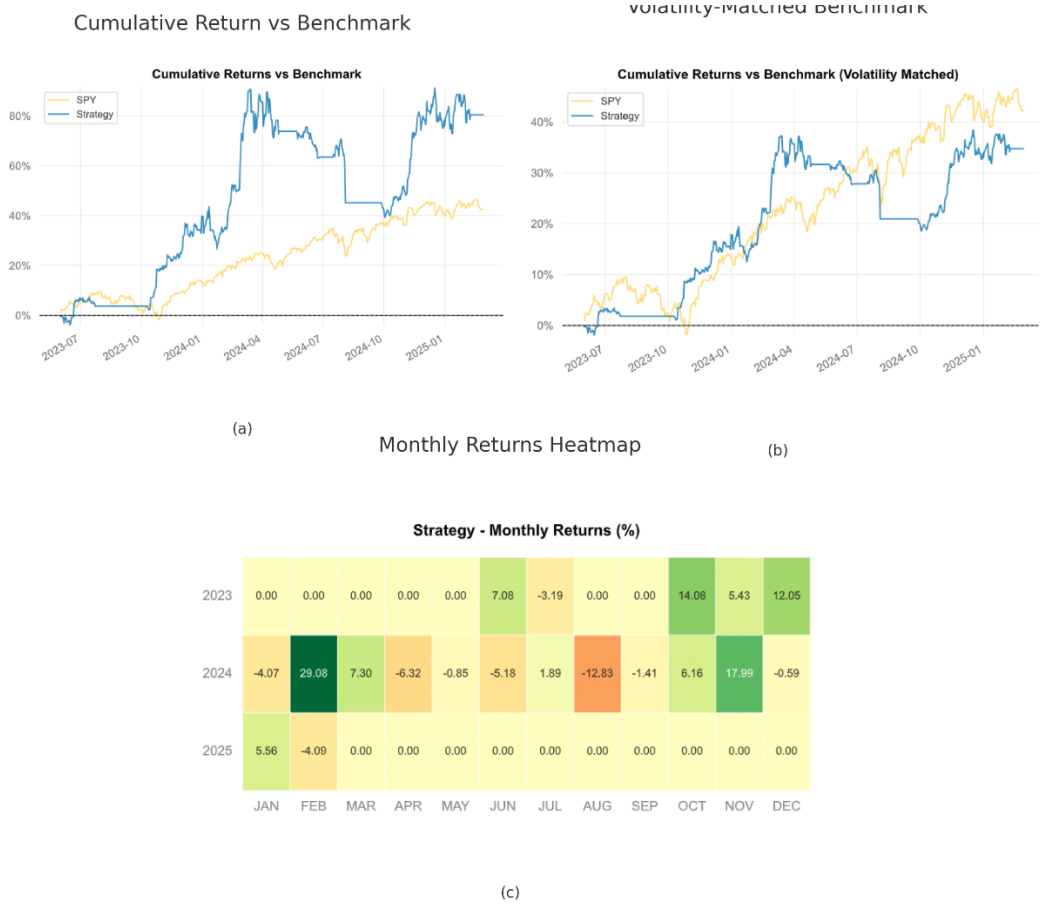

Figure 22 – Visual illustration shows the performance of the HMA strategy.

4.2.6. ADX and SMA Cross-Over

This strategy combines a traditional SMA crossover system with a trend strength filter using the Average Directional Index (ADX). It uses 1-day candles to compute a 9-period and a 21-period Simple Moving Average, generating a crossover signal when the short-term SMA crosses above or below the long-term SMA. A BUY signal is generated when the 9-period SMA crosses above the 21-period SMA, but only if the ADX value is greater than 20, indicating a strong trend. A SELL signal is triggered when the 9-period SMA crosses below the 21-period SMA, again filtered by an ADX value above 20. By enforcing this trend-strength condition, the strategy aims to enter positions only during meaningful market movements, avoiding trades during sideways or weak markets.

Back Test Performance Overview

The following Table 19 summarizes key risk and performance metrics observed during the back test:

Table 19 – Performance Metrics of the ADX and SMA cross-over strategy

Metric	Value
Total Return	≈ 101%
Maximum Drawdown	≈ -22.52%
RoMaD	≈ 4.48
Sharpe Ratio	≈ 1.52
Sortino Ratio	≈ 2.25

The following visualizations present the ADX and SMA cross-over strategy performance, including cumulative returns, risk-adjusted comparisons, and monthly return distribution. (See Fig. 23)

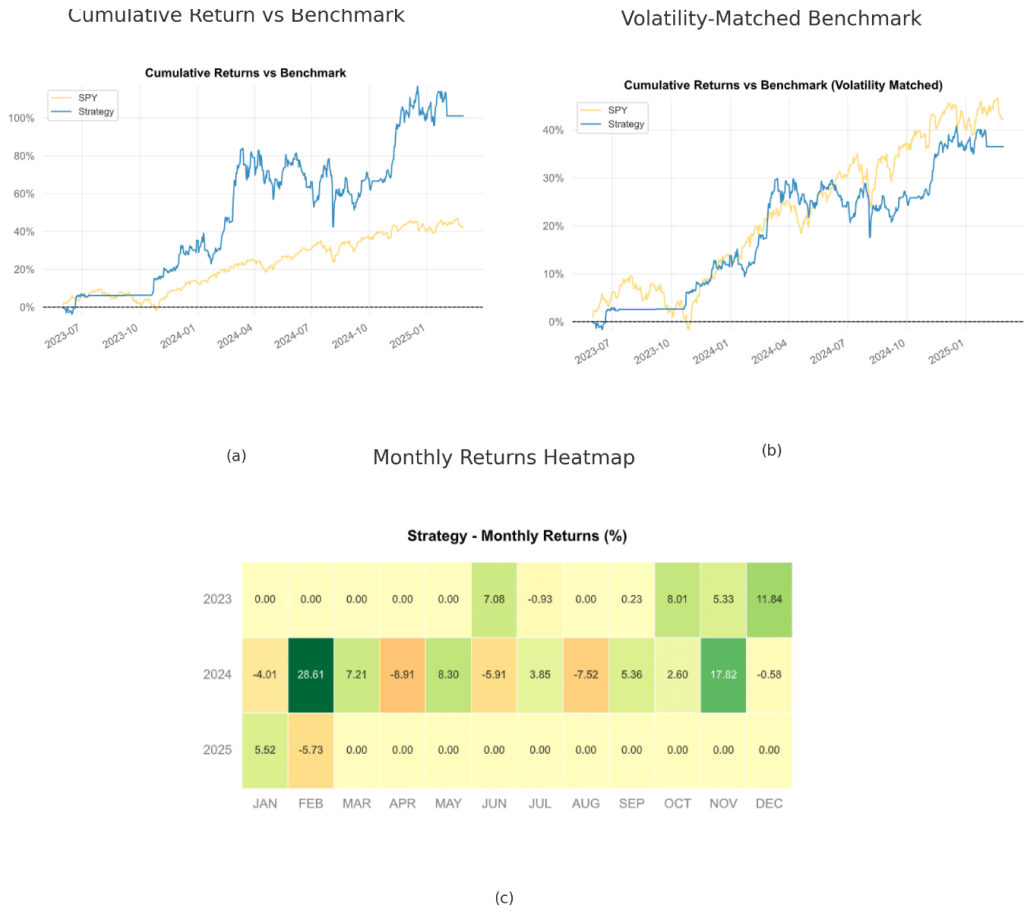

Figure 23 – Visual illustration shows the performance of the ADX and SMA cross-over strategy.

4.3. Performance Benchmark: Model vs. Buy-and-Hold Strategy

To contextualize the strength of our trading strategies, we conducted a comparative performance evaluation between our best-performing models for BTC and ETH and a traditional Buy-and-Hold strategy on the same assets. The back tests were executed using the LUMIBOT framework.

In the Buy-and-Hold approach, we assumed the entire capital was invested in the asset at the beginning of the period (i.e., BTC or ETH bought in full at the start) and held without any trades until the end of the test. This comparison provides a meaningful benchmark to assess whether our active strategies can consistently outperform a passive investment strategy over the same timeframe.

4.3.1. BTC Buy-and-Hold Strategy

The following visualizations present the performance of the Buy-and-Hold strategy on BTC over the period from May 15, 2023 to February 28, 2025, using SPY as a benchmark. These charts include cumulative returns, volatility-adjusted comparisons, and monthly return distribution. They allow for a clear evaluation of how a passive investment in BTC performs relative to a market benchmark. (See Fig. 24)

Figure 24 – Visual illustration shows the performance of the BTC BuyHold strategy.

From Figure 24:

- **(a) Cumulative Return vs Benchmark:** The Buy-and-Hold strategy on BTC significantly outperformed SPY in total return, reaching over 200% compared to SPY's less than 50%.
- **(b) Volatility-Matched Benchmark:** Despite the high return, the strategy came with substantial volatility and risk, as evidenced by sharp fluctuations.
- **(c) Monthly Returns Heatmap:** Strong gains occurred in February and November 2024, alongside setbacks such as February 2025 (-20.40%), indicating a moderate drawdown.

Table 20 provides a side-by-side comparison of performance metrics between the Buy-and-Hold strategy and our best-performing BTC model. It highlights key indicators such as return, risk, and risk-adjusted performance.

Table 20 – Performance Metrics of the best model and BuyHold on BTC.

Metric	**Best Model**	**BuyHold**
Total Return	≈ 162%	≈ 204%
Maximum Drawdown	≈ -8.44%	≈ -31.67%
RoMaD	≈ 19.19	≈ 6.44
Sharpe Ratio	≈ 3.65	≈ 1.51
Sortino Ratio	≈ 8.75	≈ 2.3

Despite the Buy-and-Hold strategy achieving a higher total return of approximately 204% compared to 162% for the best model, the performance metrics clearly highlight the superior risk management and consistency of the model. The Buy-and-Hold approach suffered a maximum drawdown of -31.67%, significantly worse than the model's modest -8.44%.

As a result, the Return over Maximum Drawdown (RoMaD) for the model was nearly three times higher at 19.19 versus 6.44. Similarly, the model outperformed both Sharpe Ratio (3.65 vs 1.51) and Sortino Ratio (8.75 vs 2.3), indicating far better risk-adjusted returns and downside protection. These results suggest that while Buy-and-Hold yielded a higher return, it came with substantially greater risk and volatility, making the model a more attractive choice for stable and reliable performance.

4.3.2. ETH Buy-and-Hold Strategy

The following visualizations present the performance of the Buy-and-Hold strategy on ETH over the period from June 26, 2023 to February 28, 2025, using SPY as a benchmark. These charts include cumulative returns, volatility-adjusted comparisons, and monthly return distribution. They allow for a clear evaluation of how a passive investment in ETH performs relative to a market benchmark. (See Fig. 25)

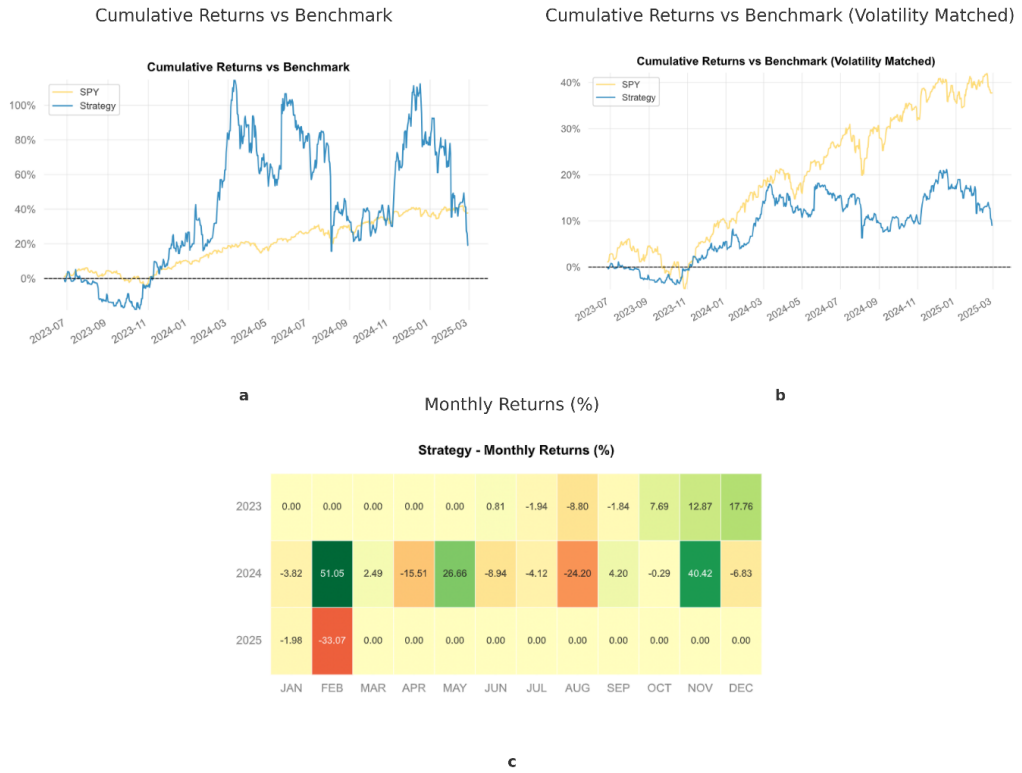

Figure 25 – Visual illustration shows the performance of the ETH BuyHold strategy.

From Figure 25:

- **(a) Cumulative Return vs Benchmark:** Although the Buy-and-Hold strategy on ETH briefly peaked above 100% return, it experienced multiple sharp crashes throughout the test period, ultimately ending with a return of less than 19%, significantly underperforming SPY's steady and consistent growth.
- **(b) Volatility-Matched Benchmark:** Despite the limited return, the ETH strategy experienced substantial volatility and inconsistency, underperforming SPY on a risk-adjusted basis.
- **(c) Monthly Returns Heatmap:** Highlights the unstable nature of the strategy, showing sharp gains in February and November 2024, but also severe losses such as February 2025 (-33.07%) and August 2024 (-24.20%), indicating a highly turbulent performance profile.

Table 21 presents a side-by-side comparison of performance metrics between the Buy-and-Hold strategy and our best-performing model on ETH. This summary highlights key indicators such as return, risk, and risk-adjusted performance.

Table 21 – Performance Metrics of the best model and BuyHold on ETH.

Metric	**Best Model**	**BuyHold**
Total Return	≈ 80%	≈ 19%
Maximum Drawdown	≈ -9.43%	≈ -46.08%
RoMaD	≈ 8.48	≈ 0.41
Sharpe Ratio	≈ 3.0	≈ 0.48
Sortino Ratio	≈ 6.03	≈ 0.67

Despite the Buy-and-Hold strategy yielding a positive return of approximately 19%, it was vastly outperformed by our best model, which achieved a return of about 80%. More importantly, the Buy-and-Hold approach was accompanied by severe risk, suffering a maximum drawdown of -46.08%, compared to the model's much smaller drawdown of -9.43%.

This stark contrast is further reflected in the RoMaD, where the model scored 8.48 versus just 0.41 for Buy-and-Hold—indicating far better capital preservation. Additionally, the model showed superior risk-adjusted performance with a Sharpe Ratio of 3.0 compared to 0.48, and a Sortino Ratio of 6.03 compared to 0.67. These figures underscore the model's robustness and reliability, positioning it as a far more effective strategy for long-term ETH exposure than passive holding.

CHAPTER 5
Server-Side Implementation

This chapter presents the server-side architecture that powers the automated trading system, ensuring real-time execution, scalability, and secure user management. It details the system's modular components, data flow, and communication layers that coordinate strategy execution and user interaction seamlessly.

5.1. Introduction

The server acts as the central coordinator of the Kryseos trading system. It manages user authentication, executes trading strategies, and streams real-time feedback to the client. Its architecture is designed for concurrency, reliability, and secure handling of credentials.

5.1.1. Core Technologies

The backend is built using:

- **FastAPI** – RESTful APIs and WebSockets.
- **SQLite + SQLAlchemy** – Persistent, lightweight data storage.
- **Threading** – Concurrent strategy execution per user.
- **Alpaca API** – Broker access for trading and data.
- **TensorFlow** – ML-based trading model support.

5.1.2. Responsibilities

The system is responsible for securely managing user authentication and encryption of API credentials to ensure data privacy. It enables users to start and stop trading strategies on a session basis, allowing personalized and controlled execution. Additionally, it performs real-time trade execution and integrates machine learning inference for decision-making. To enhance user experience and situational awareness, the system streams live feedback through WebSocket connections, providing timely updates on strategy status, performance metrics, and market signals.

5.2. System Initialization

At startup, the server prepares its environment, security keys, and user session state to support real-time trading. Environment variables are loaded from a `.env` file using `python-dotenv`, including the `FERNET_KEY` used with `cryptography.Fernet` for encrypting API credentials. If the key is missing, a new one is generated and saved.

The FastAPI app is initialized in `app.py`, where endpoints are registered, logging is enabled, and `startup_event()` is triggered. This function performs asynchronous database initialization via `init_db()` and restores user records from the SQLite database using `load_existing_users()`. For each user, decrypted credentials are loaded into memory and associated dictionaries such as `user_configs`, `user_feedback`, and `user_locks` are initialized.

Each user gets a dedicated thread using `threading.Thread` running `user_thread_function()`, which stays idle until trading is activated. Once active, it fetches market data, generates trade signals, executes orders, and updates metrics. This design ensures encrypted persistence, recovery after restarts, and concurrent strategy execution.

5.3. Database Management

The Kryseos backend uses SQLite for persistent user data storage, interfaced via SQLAlchemy with asynchronous support. The schema is defined in `models.py`, with the primary entity being the `User` table. Each user record includes a unique `user_id` (UUID), encrypted `api_key` and `api_secret` (secured using Fernet encryption with a key from `.env`), account type (paper/live), and a mutable `equity` list that tracks trading performance over time.

The database is initialized on startup using the `init_db()` function, which ensures all required tables are created using an asynchronous engine configured with the `sqlite+aiosqlite` dialect. A session factory named `AsyncSessionLocal` is defined via `sessionmaker()` to enable efficient, non-blocking database operations throughout the server.

The `load_existing_users()` function retrieves users asynchronously, decrypts credentials, and restores their in-memory state. This ensures session continuity and real-time readiness. Error handling is in place for invalid encryption, preserving integrity across restarts.

5.4. User Authentication and API Key Encryption

The Kryseos backend implements secure login and logout mechanisms with encrypted credential storage and runtime session management.

Login Process (/login)

- **Form Input:** Requires `api_key`, `api_secret`, and `account` (paper/live).
- **Key Validation:** Verifies credentials by sending a GET request to `/v2/account` on the Alpaca API using the submitted keys.
- **Encryption:** Validated keys are encrypted using Fernet (key loaded from `.env`) and stored securely in the database.
- **User Handling:**
  - If the user exists, session is resumed.
  - If new, a UUID is generated and credentials are stored encrypted.
- **Session Setup:** Initializes:
  - `user_credentials`, `user_configs`, `user_feedback`, `user_locks`, `user_running_flags`, `user_sockets`
  - Spawns a dedicated background thread using `user_thread_function()`.

Logout Process (/logout)

- Gracefully closes all WebSocket connections for the user.
- The trading thread continues even if the user logs out.
- Keeps all user-related in-memory dictionaries as well as the database entry.

Note: Persistent trading history is retained across sessions, and all communication is encrypted and thread-safe.

5.5. In-Memory User State and Thread Model

To manage real-time trading sessions efficiently, the backend maintains several global in-memory dictionaries, each tracking a different aspect of user-specific state. This design allows isolated and concurrent execution for every active user.

At login or server startup, the system initializes the following structures:

- `user_credentials` – Maps `user_id` to decrypted API keys and account type, used for API client setup.
- `user_configs` – Stores strategy parameters such as symbol, risk, and trading mode.
- `user_feedback` – Holds live metrics (e.g., equity, win rate) for WebSocket updates.
- `user_running_flags` – Mutable flags signaling active trading status.
- `user_threads` – References to background threads executing user strategies.
- `user_locks` – Threading locks for safe concurrent updates to shared dictionaries.
- `user_sockets` – Tracks active WebSocket connections for push-based feedback.

Each user operates on a dedicated thread created using `threading.Thread`, which runs in a continuous loop to handle strategy execution and real-time data independently. The thread first waits for a valid configuration in `user_configs`. Once available, it initializes the corresponding trading logic and begins monitoring the market. If a change in strategy or symbol is detected, the thread reinitializes to reflect the new configuration. When the user logs out or issues a stop command, the thread clears the configuration and sets the run flag to `False`, effectively terminating its execution.

This structure allows multiple trading sessions to run independently, ensuring responsiveness, thread safety, and persistent user state throughout a session's lifecycle.

5.6. Trading Session Control

The trading server controls the lifecycle of each user's trading session through dedicated API endpoints. When a session is started, a background thread assigned to that user begins executing the selected strategy using live market data. When stopped, the thread halts trading logic while preserving the session context.

Starting a Session (`/start-trading`)

To start trading, the client sends a POST request to `/start-trading` with:

- `user_id` – the user's unique identifier
- `symbol` – trading asset (e.g., BTC/USD)
- `strategy` – the strategy ID to execute
- `risk` – risk percentage per trade
- `account` – Alpaca endpoint (paper/live)

The server responds by updating `user_configs[user_id]` with the new configuration and setting `user_running_flags[user_id][0]` to `True`. A "Running" status is immediately sent to the client via WebSocket.

Each user has a thread that monitors their configuration. When a valid config is detected, the thread:

- Initializes the Alpaca API client using stored credentials.
- Starts a loop that continuously fetches market data.
- Executes the strategy and pushes performance updates via WebSocket.

Stopping a Session (`/stop-trading`)

To stop trading, the client calls `/stop-trading`. The server:

- Clears the user's config by setting `user_configs[user_id]` to `None`.
- Sets `user_running_flags[user_id][0]` to `False` to halt trading.
- Sends a "Stopped" message to the client over WebSocket.

This halts trade execution while keeping the user session active for future reactivation.

Conclusion: This modular control flow ensures trading sessions are started and stopped cleanly, with real-time feedback and minimal disruption to the user experience.

5.7. Strategy Execution and Trade Flow

Each user's trading thread dynamically selects and executes a strategy based on their current configuration. The selection logic is implemented in `services.py`, where integer identifiers map to specific trading strategies.

Strategy IDs:

- 1 – Machine Learning
- 2 – Simple Moving Average (SMA)
- 3 – MACD

- 4 – Hull Moving Average (HMA)
- 5 – SMA + ADX
- 6 – RSI + SMA50

These IDs guide the trading loop to dispatch the appropriate signal logic, whether it's based on a rule-based strategy or a machine learning model prediction.

Trading flow:

- Signals: `BUY`, `SELL`, or `NONE`.
- `BUY` triggers only if no open position exists.
- `SELL` closes open positions via `api.close_all_positions()`.

The amount to trade is determined using the user's configured risk and current asset price:

$$quantity = \frac{cash \times risk}{asset\ price} \tag{11}$$

This ensures controlled risk exposure. All trading decisions and executions follow a modular structure, enabling real-time response and safe operation aligned with user-defined settings.

5.8. Performance Feedback System

To keep users informed in real-time, the server continuously evaluates trading performance and broadcasts feedback using WebSockets. This system includes two major components: analysis and broadcasting.

Performance Analysis:

- The function `analyze_trading_performance()` fetches the user's equity via Alpaca's API.
- If the value differs from the last saved entry, it is appended to the equity history.
- Filled orders are retrieved and matched as BUY/SELL pairs to compute per-trade profit:

$$profit\ ratio = \frac{sell\ price - buy\ price}{buy\ price} \tag{12}$$

Key metrics derived:

- Total and last trade profit ratios
- Number of completed trades
- Win rate based on profitable trades
- Equity history for time-based performance tracking

Result Broadcasting:

- `broadcast_feedback_to_user()` sends performance data to all active WebSocket connections.
- `send_bot_status()` updates client UI with bot status (e.g., "Running" or "Stopped").

Together, this system enables accurate, responsive feedback and empowers users to monitor performance across sessions in real time.

5.9. Real-Time WebSocket Communication

The trading server uses WebSockets to deliver low-latency updates to clients, including trading performance metrics and bot status. This ensures real-time synchronization between the server and all connected devices.

WebSocket Endpoint: `/ws/{user_id}` Each user connects through a dedicated WebSocket route. Upon connection:

- The server accepts the connection using `await websocket.accept()`.
- The connection is saved to `user_sockets[user_id]`, allowing multi-device support.

Initial Sync on Connect: Right after establishing a connection, the server pushes:

- A `performance_update` message if feedback exists in `user_feedback`.
- A `bot_status` message showing whether the bot is "Running" or "Stopped".

Ongoing Usage and Broadcasts:

- Connections are reused for pushing updates with `broadcast_feedback_to_user()`.
- Bot status is pushed via `send_bot_status()` to reflect session changes.

Disconnection Handling: When a client disconnects, the WebSocket is removed using:

```
user_sockets[user_id].discard(websocket)
```

This avoids memory leaks and keeps the active socket pool clean.

Conclusion: WebSockets provide the backbone for user-facing reactivity in the system, enabling live feedback, session control, and seamless state awareness across all connected clients.

5.10. Security Measures

The server incorporates multiple layers of security to protect user credentials, prevent misuse, isolate execution, and manage resources safely across concurrent trading sessions.

Encrypted API Credentials: User API keys and secrets are never stored in plaintext. Instead, they are encrypted using the Fernet symmetric encryption scheme from the `cryptography` library:

- A Fernet key is generated on first launch and stored in a `.env` file.
- Encrypted credentials are stored in the database as part of the `User` table.
- Decryption is done only at runtime and kept in memory.

Account Endpoint Validation: To ensure proper API usage and block malicious configurations:

- Only official Alpaca endpoints are allowed:
  - `https://paper-api.alpaca.markets`
  - `https://api.alpaca.markets`
- Any other endpoint returns a 400 Bad Request.
- Validity is confirmed by making a test call to `/v2/account`.

Thread Isolation and Safety: Each user runs in a dedicated thread with isolated runtime state, ensuring that no user can access or interfere with another's session. Shared structures such as `user_configs` and `user_feedback` are protected using locks to prevent data corruption. This design allows for safe concurrency while avoiding race conditions.

Graceful Cleanup on Logout: When a user logs out:

- WebSockets are closed and removed from `user_sockets`.
- The trading thread continues running even in logout.

5.11. Summary

The trading server is designed for reliability, scalability, and real-time responsiveness in a multi-user environment. It manages encrypted user credentials, executes personalized trading strategies in isolated threads, and provides live performance updates through WebSocket channels. The system ensures thread-safe memory access, supports modular strategy integration, and persists user equity history for accurate analytics.

Key Features:

- Encrypted API credentials and persistent SQLite storage
- Isolated per-user threads for safe parallel strategy execution
- Real-time feedback and bot status updates via WebSocket

CHAPTER 6
Mobile App Design and Implementation

This chapter presents the design, functionality, and architecture of the mobile application developed to interface with the automated trading system. It details how the app facilitates user interaction, strategy control, and real-time monitoring through seamless communication with the backend server.

6.1. Overview

The mobile application acts as the main interface between the user and the automated trading system. It is designed for ease of use, allowing users to configure strategies, control trading activity, and monitor live performance, all from a portable device.

Purpose and Communication

The app's primary goal is to provide users with a convenient, real-time control panel for their trading bot. It achieves this through two main communication channels with the backend server:

- **REST API** – Used to send user credentials, strategy selections, and trading parameters. It also receives session and configuration responses from the server.
- **WebSocket** – Maintains a persistent connection to receive real-time updates, including trading performance metrics and bot status.

Core Responsibilities

The mobile app handles several key functions:

- **User Configuration** – Users can input their API keys, choose trading strategies, and set trading symbols and risk levels.
- **Strategy Control** – Offers buttons to start and stop trading sessions. These actions are relayed to the server to trigger backend trading threads.
- **Live Monitoring** – Real-time data such as equity changes, trade count, and profit ratio are displayed dynamically using the WebSocket channel.

Design Philosophy

By offloading all computation and trading execution to the backend, the mobile app remains lightweight and responsive. Its role is focused on user experience, configuration, and feedback delivery, ensuring that users can interact with the trading system efficiently and securely from anywhere.

6.2. Architecture Overview

The mobile application adopts a modular and reactive architecture, ensuring maintainability, clear data flow, and responsiveness. Key components are split across Dart files, each responsible for specific UI and logic functionality.

Core Components

- **main.dart:** Initializes the app and registers the global `AppData` provider using `MultiProvider`.
- **login.dart:** Manages authentication and sends credentials to the server.
- **mainPage.dart:** Hosts the tab-based layout for core pages.
- **trade.dart:** Allows users to configure strategies and control trading sessions.
- **dashboard.dart:** Displays real-time performance metrics streamed from the backend.
- **market.dart:** Visualizes market price movements for selected symbols.
- **appData.dart:** Central state container using `ChangeNotifier`; handles all user and trading-related data.

Data Flow

The app follows a unidirectional flow:

$$\texttt{Server} \rightarrow \texttt{WebSocket/REST} \rightarrow \texttt{AppData} \rightarrow \texttt{UI Widgets}$$

Widgets receive backend updates through network channels, modify `AppData`, which in turn notifies the UI.

State Management

State is globally managed via the `Provider` package. `AppData` handles:

- API credentials and login state
- Real-time trading feedback (e.g., equity, win rate)
- Bot status and strategy parameters

This design enables real-time synchronization between backend logic and user interface while keeping the app lightweight and modular.

6.3. UI and UX Design

The user interface of the mobile application is designed to provide an intuitive, responsive, and visually informative experience. This section describes the design considerations and interactive elements of each key page, as implemented in the app.

6.3.1. Login Page (`login.dart`)

As shown in Fig. 26, the login page is responsible for securely capturing the user's API credentials and ensuring they are valid before allowing access to the system:

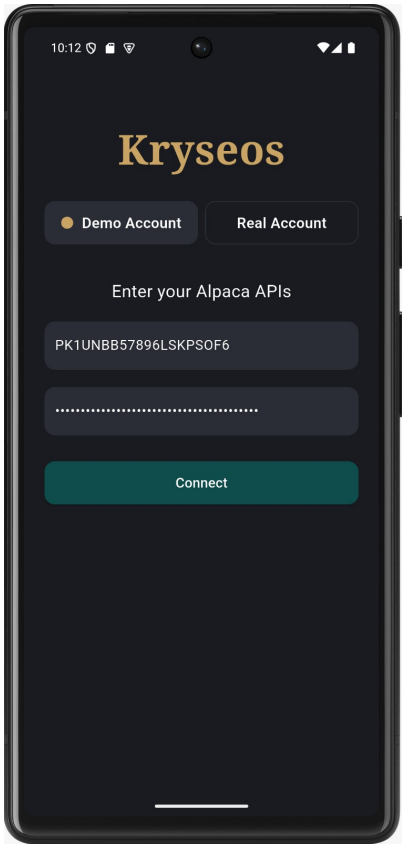

Figure 26 – Kryseos App Login Page.

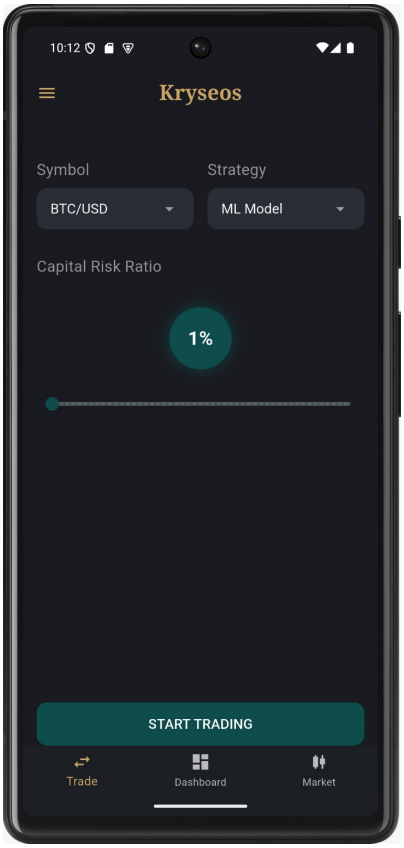

Figure 27 – Kryseos App Trade Tab.

6.3.2. Trade Page (`trade.dart`)

As shown in Fig. 27, the trade page allows users to configure and manage trading sessions. The design prioritizes both flexibility and safety in execution:

- **Symbol Selection:** As shown in Fig. 28, the page provides a dropdown menu for selecting the trading symbol (e.g., BTC/USD, ETH/USD). This allows the user to target specific assets for strategy execution and ensures that all performance feedback and market data correspond to the selected symbol.
- **Strategy Selection:** As shown in Fig. 29, the page includes dropdown menus for selecting from multiple trading strategies (e.g., SMA, MACD, ML Model).
- **Risk Configuration:** As illustrated in Fig. 30 and Fig. 31, users can adjust parameters such as risk percentage using interactive widgets like sliders and text inputs.
- **Session Control Buttons:** Clear and accessible buttons are provided to start and stop trading sessions. These trigger backend commands to initiate or terminate trading threads.
- **UX Considerations:**
    - Input validation prevents submission of incomplete or illogical configurations.
    - The active strategy and trading status are clearly indicated, reducing the risk of user confusion or accidental trades.

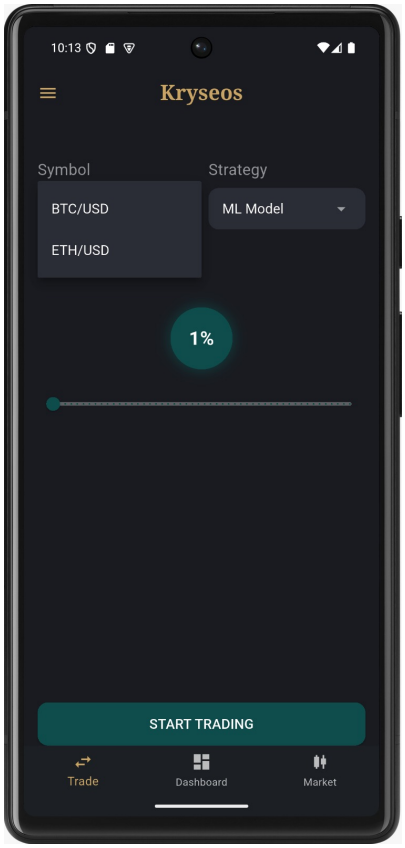

Figure 28 – Symbol Selection.

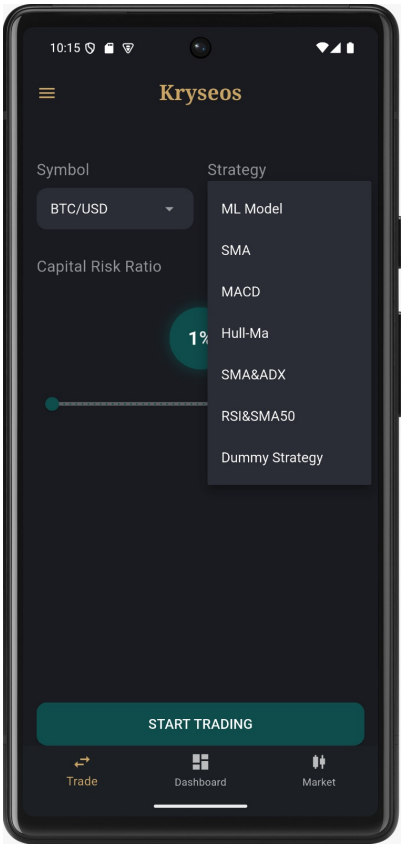

Figure 29 – Strategy Selection.

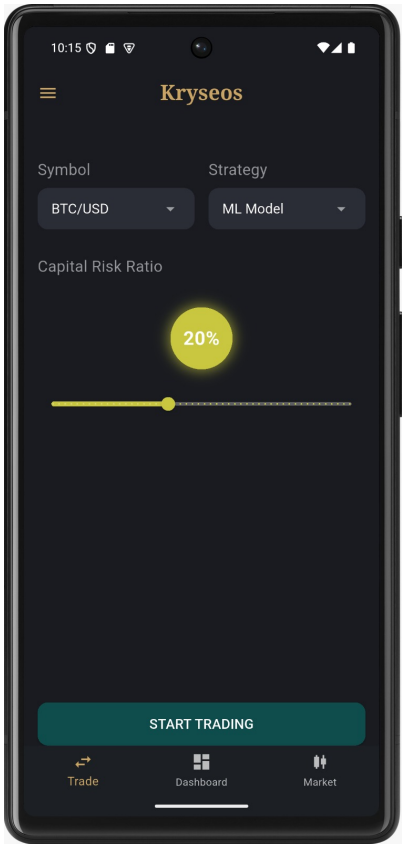

Figure 30 – (20%) Risk Ratio.

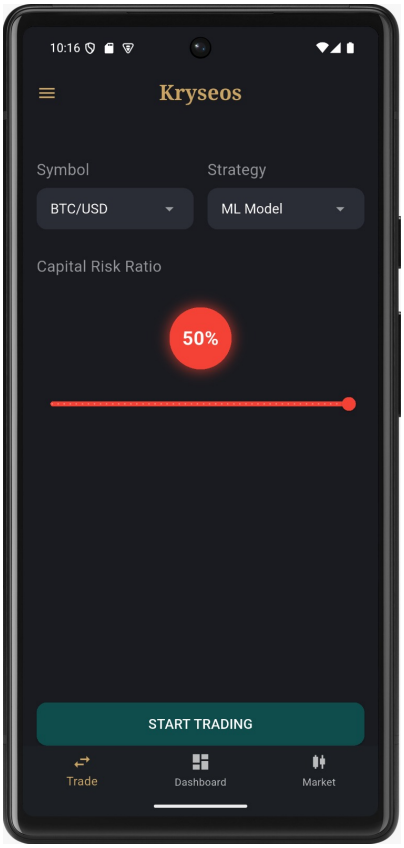

Figure 31 – (50%) Risk Ratio.

6.3.3. Dashboard Page (`dashboard.dart`)

The dashboard page presents performance metrics and real-time feedback in a visually digestible format, as shown in Fig. 32 and Fig. 33.

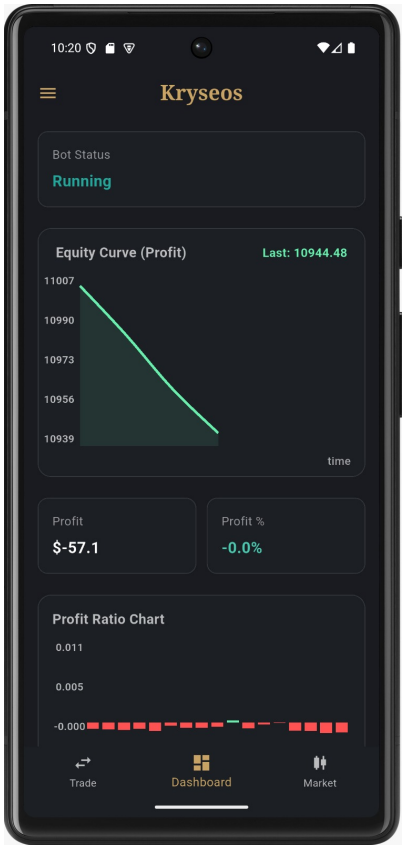

Figure 32 – Kryseos App Dashboard Tab.

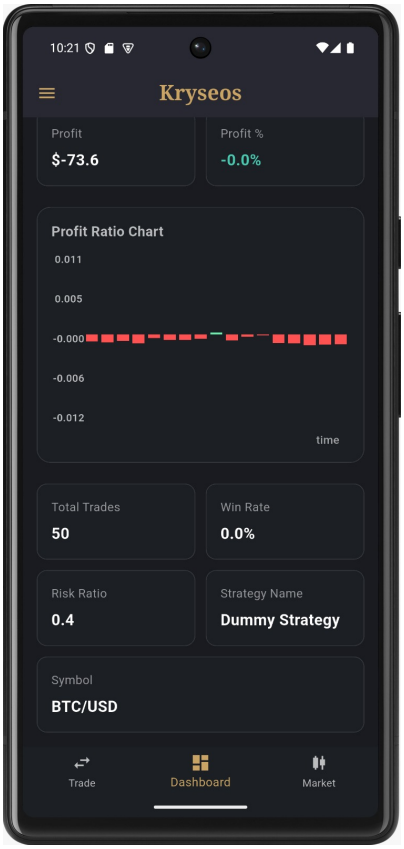

Figure 33 – Kryseos App Dashboard Tab.

- **Live Charts:** Performance is shown using dynamic charts including equity curves and profit/loss progression.
- **Bar Charts and Key Metrics:** Important feedback variables such as win rate, number of trades, and profit ratios are displayed as labeled bars and numeric indicators.
- **UX Focus:** The layout emphasizes readability and real-time responsiveness. Metrics update via WebSocket without user interaction, and data is grouped logically for quick interpretation.

6.3.4. Market Page (`market.dart`)

The market page offers a snapshot of live trading data from selected exchanges, as shown in Fig. 34.

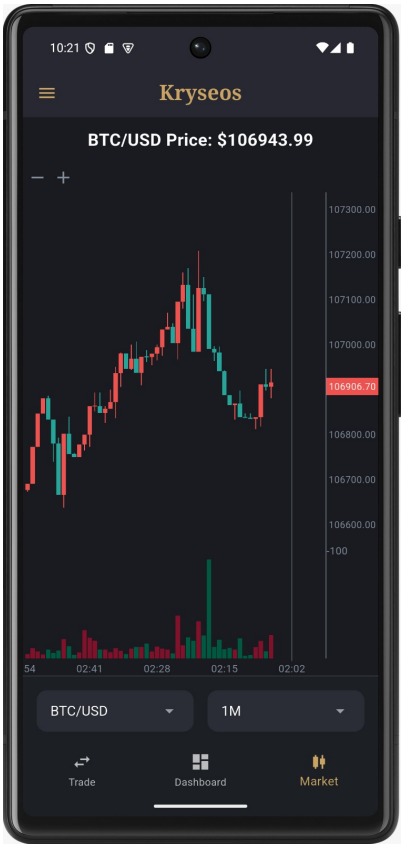

Figure 34 – Kryseos App Market Tab.

- **Live Prices:** Fetched from Binance based on trading mode.
- **Candlestick View:** Visualize recent market trends.
- **User-Friendly:** Designed for quick insights and fast strategy decisions.

All app pages follow a responsive, minimal, and intuitive design to ensure usability across experience levels.

6.4. State Management and Data Flow

The Kryseos mobile app manages its runtime state using a centralized model built with the `Provider` package. The `appData.dart` file defines an `AppData` class that extends `ChangeNotifier`, serving as a global state container injected across the widget tree via `MultiProvider`.

Key Responsibilities of `AppData`:

- Stores API key, trading symbol, strategy, risk settings, and live metrics (e.g., equity, win rate).
- Handles real-time updates via the `updateFromServer` method, which parses WebSocket data and triggers UI refresh with `notifyListeners()`.
- Provides safe update methods consumed by UI pages like `trade.dart` and `dashboard.dart`.

This design ensures consistent, reactive state across the entire application.

For persistent login, the app uses Flutter's `SharedPreferences` to locally store the `user_id` upon successful authentication. This ID allows:

- Auto-login checks at startup—if valid, the app skips the login screen.
- Re-authentication fallback if the session is expired or invalid.
- Secure session management—API credentials are never stored locally.

By combining centralized state and minimal persistent data, the app delivers a smooth user experience while maintaining session integrity and security.

6.5. Real-Time Communication

To enable live updates and reactive interfaces, the mobile application establishes a persistent WebSocket connection with the backend server. This section describes the setup, management, and message handling mechanisms for real-time communication.

6.5.1. WebSocket Connection

The WebSocket connection is initiated and managed primarily through a coordinated interaction between `main.dart` and `appData.dart`:

- **Connection Setup:** The function `startWebSocket()` defined in `main.dart` is responsible for opening the WebSocket channel to the server. Once connected, it continuously listens for incoming messages and routes them to the appropriate handler.
- **Injection into AppData:** The message stream is passed into the `AppData` provider via the `handleWebSocketMessage()` method. This function parses the incoming JSON data and updates relevant application state fields.
- **Session Lifecycle Management:** On logout or when trading sessions are stopped, the app ensures that WebSocket resources are released. The socket is closed cleanly, and any active listeners are removed. This prevents stale data propagation or memory leaks.
- **Reconnection Logic:** The WebSocket client automatically handles reconnection scenarios using built-in error detection. If the connection drops during a session, the app attempts to re-establish the socket and restore feedback streaming.

The WebSocket connection provides the backbone for all real-time feedback mechanisms in the app, especially for the `dashboard` and `trade` pages.

6.5.2. Message Handling

The mobile client processes multiple types of messages sent by the backend, with each corresponding to a specific UI or logic update:

- **Message Types:**
  - `performance`: Contains metrics such as equity, number of trades, profit ratio, and win rate.
  - `status`: Describes the current bot session state (e.g., active, idle, or stopped).
- **Message Parsing:** Incoming messages are received in JSON format and parsed based on their `type` field. Each message is routed to appropriate update methods within `AppData`, such as `updatePerformanceData()` for performance metrics or `setBotStatus()` for session state changes.
- **UI Synchronization:** Once `AppData` is updated, all listening widgets (e.g., `dashboard.dart`, `trade.dart`) are automatically rebuilt to reflect the latest state. This enables live feedback graphs, bot status indicators, and real-time strategy control.
- **Robustness:** The message handler includes safety checks for malformed or unexpected data types to ensure application stability even under erratic network conditions.

This real-time communication layer is essential to the system's responsiveness, ensuring that the user interface accurately mirrors backend state with minimal latency.

6.6. Conclusion

The mobile application constitutes the interactive frontend of the trading system, providing users with a responsive and intuitive interface to control trading strategies, observe live performance metrics, and manage their sessions securely. Built using the Flutter framework, the application follows a structured tab-based navigation system comprising the `Dashboard`, `Market`, and `Trade` pages, with a dedicated `Login` page handling secure user authentication and local storage of credentials.

From a user experience standpoint, the application offers configurable trading parameters including strategy selection, risk control sliders, and session management controls. The interface emphasizes clarity and feedback, with visual indicators and error handling integrated into every critical interaction.

In summary, the mobile app effectively bridges the user and the backend trading engine, encapsulating all essential features—secure configuration, live feedback monitoring, and strategic control—within a modern, modular, and extensible Flutter-based system. It stands as a reliable and practical component of the larger automated trading infrastructure, built to accommodate future enhancements in both functionality and user interface design.

CHAPTER 7
RESULTS AND DISCUSSION

This chapter presents the final outcomes and critical analysis of the Kryseos trading application, developed as a real-time, AI-assisted cryptocurrency trading platform. The results span the successful implementation of both server-side and mobile-side architectures, integration with financial APIs, support for multiple trading strategies, and the performance of the strategies over historical market data. The system was designed to be modular, multi-user, and production-ready, ensuring reliability and extensibility.

7.1. Project Outcomes

The final product consists of two main components:

- **FastAPI-based Trading Server:** This backend handles strategy execution, user management, encryption of sensitive credentials, real-time communication with clients via WebSocket, and logging of trades and performance metrics.
- **Flutter Mobile Application:** Designed with usability in mind, the app allows users to log in, configure strategies, monitor trades in real time, and visualize performance data through interactive plots and dashboards.

The system was tested with both simulated and live data using Alpaca and Binance APIs. A series of machine learning and rule-based strategies were implemented in a modular format, including moving averages (SMA, MACD, HMA), RSI/ADX-based filters, and a neural network-based predictor.

7.2. System Performance and Integration

- **Concurrency and Stability:** The server supports concurrent users, where each session operates in a separate asynchronous context with isolated state and real-time feedback.
- **Data Flow and State Management:** The WebSocket connection ensures low-latency updates to the app, while the Provider-based Flutter architecture ensures consistent UI state with real-time feedback values and performance metrics.
- **Security:** API keys are encrypted using Fernet (AES-128) and stored securely in SQLite. No keys are stored on the device; instead, a persistent user ID enables automatic login.
- **Visualization:** The app supports candlestick charts, equity curves, performance bars, and strategy control panels for transparency and control.

7.3. Discussion

The project successfully demonstrates the feasibility of deploying a real-time, mobile-connected trading system that combines the strengths of AI prediction and technical indicators. The results of the best-performing strategies showed superior returns compared to simple Buy-and-Hold strategies, particularly in terms of drawdown minimization and risk-adjusted performance.

Limitations and Future Work:

- Strategies were tested on historical data only; live deployment was limited due to market volatility and simulation constraints.
- Some model overfitting was observed in early tests—future versions could include walk-forward validation and ensemble methods.
- More advanced portfolio management (multi-asset support, position sizing) is recommended as a next step.

Overall, Kryseos provides a robust foundation for intelligent, mobile-first algorithmic trading and serves as a viable prototype for both personal use and further commercial development.

CHAPTER 8
CONCLUSIONS AND FUTURE WORK

8.1. Conclusions

This project successfully delivered a complete end-to-end cryptocurrency trading platform—Kryseos—that integrates a real-time backend server with a responsive Flutter-based mobile application. The system enables users to select and deploy various trading strategies, monitor their performance in real time, and manage their trading sessions securely.

Key achievements include:

- Development of a multi-user FastAPI backend capable of asynchronous execution and secure credential storage using AES-based encryption.
- Integration with Alpaca and Binance APIs to fetch live market data and execute trades reliably.
- Implementation of multiple rule-based and AI-based trading strategies in a modular fashion, allowing for flexible strategy deployment.
- Design of an intuitive mobile application that supports real-time communication via WebSockets, visual performance dashboards, and state management with auto-login support.

These outcomes demonstrate the feasibility of deploying an AI-assisted, mobile-first trading platform capable of real-time decision-making and user interaction, with results showing favorable performance relative to passive investment strategies.

8.2. Future Work

While the project successfully achieves its initial objectives, several directions offer potential for future enhancement and expansion:

- **Live Market Deployment:** Extending the system from historical backtesting to continuous live trading would validate its effectiveness under real market conditions. Furthermore, adding support for multiple trading platforms—such as Binance, Interactive Brokers, or MetaTrader—in addition to Alpaca, would improve flexibility and broaden adoption.
- **Short Selling Capabilities:** Enabling the system to take short positions would allow it to capitalize on downward market trends, unlocking profit opportunities that are currently missed in the long-only framework.
- **Strategy Optimization:** Performance could be further enhanced through hyperparameter tuning, ensemble techniques, and reinforcement learning approaches, which may lead to more adaptive and profitable trading behavior.
- **User Experience Enhancements:** Improvements to the mobile interface—such as real-time performance alerts, strategy comparison dashboards, and dark mode—would enhance usability and engagement.

- **Scalability and Deployment:** Migrating the backend infrastructure to a cloud-native architecture (e.g., using Docker and Kubernetes) would enhance system reliability and scalability, supporting a larger number of concurrent users.

These enhancements would elevate Kryseos from a functional prototype to a production-grade financial trading assistant ready for market integration and further research.

REFERENCES

[1] S. Hadi, "A study on macd trading strategy," *Journal of Financial Indicators*, vol. 12, no. 3, pp. 45–53, 2016.

[2] O. B. Sezer, M. U. Gudelek, and A. M. Ozbayoglu, "Financial time series forecasting with deep learning: A systematic literature review: 2005–2019," *Applied Soft Computing*, vol. 90, p. 106 181, 2020. doi: 10.1016/j.asoc.2020.106181.

[3] P. A. Mello, "Qualitative comparative analysis: An introduction to research design and application," *Patrick A. Mello*, 2018. [Online]. Available: https://patrickmello.com/qca-research-design-application/.

[4] R. O. Gonzalez Aviles, *Deeptrader ai ea mt4 + dll (works on build 1425+)*, https://thetradelovers.gumroad.com/l/DeeptraderAIEAMT4, Forex Expert Advisor integrating AI with traditional technical analysis, 2024.

[5] Pionex, *Pionex documentation: Onboarding pionex bots with ease*, https://www.reddit.com/r/Pionex/comments/lncp8f/pionex_documentation_onboarding_pionex_bots_with/, Community-shared documentation for Pionex bots, 2021.

[6] 3Commas, *3commas review: Automated trading terminal & crypto trading bots*, https://goodcrypto.app/3commas-review-automated-trading-terminal-crypto-trading-bots/, Overview and features of 3Commas trading platform, 2022.

[7] HaasOnline, *Haasonline review and best alternatives*, https://cryptolisty.com/automated-trading/haasonline-review-and-best-alternatives/, Analysis of HaasOnline trading bots and alternatives, 2022.

[8] Alpaca Markets. "Alpaca api documentation." Accessed: May 28, 2025. (n.d.), [Online]. Available: https://alpaca.markets/docs/api-references/.

[9] D. T. Christiansen, *Functional programming in lean*, https://leanprover.github.io/functional_programming_in_lean/, Comprehensive guide on functional programming using Lean, 2023.

[10] Investopedia. "Simple moving average (sma)." Accessed: May 28, 2025. (n.d.), [Online]. Available: https://www.investopedia.com/terms/s/sma.asp.

[11] Investopedia. "Moving average convergence divergence (macd)." Accessed: May 28, 2025. (n.d.), [Online]. Available: https://www.investopedia.com/terms/m/macd.asp.

[12] Investopedia. "Exponential moving average (ema)." Accessed: May 28, 2025. (n.d.), [Online]. Available: https://www.investopedia.com/terms/e/ema.asp.

[13] Investopedia. "Relative strength index (rsi)." Accessed: May 28, 2025. (n.d.), [Online]. Available: https://www.investopedia.com/terms/r/rsi.asp.

[14] A. Hull. "The hull moving average." Accessed: May 28, 2025. (n.d.), [Online]. Available: https://alanhull.com/hull-moving-average.

[15] Corporate Finance Institute. "Weighted moving average (wma)." Accessed: May 28, 2025. (n.d.), [Online]. Available: https://corporatefinanceinstitute.com/resources/career-map/sell-side/capital-markets/weighted-moving-average-wma/.

[16] Investopedia. "Average directional index (adx)." Accessed: May 28, 2025. (n.d.), [Online]. Available: https://www.investopedia.com/terms/a/adx.asp.

[17] A. Graves, *Supervised Sequence Labelling with Recurrent Neural Networks*. Springer, 2012.

[18] Y. Bengio, P. Simard, and P. Frasconi, “Learning long-term dependencies with gradient descent is difficult,” *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.

[19] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[20] C. Olah. “Understanding lstm networks.” Accessed: May 28, 2025. (2015), [Online]. Available: `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`.

APPENDICES

The following materials are considered essential components of this project but are submitted separately from this documentation in digital format. These files are provided independently to maintain clarity, reduce file size, and preserve the reproducibility of our work. All files are organized and clearly labeled for ease of access.

- **Backtesting Code**: Python scripts used to simulate trading strategies on historical data.
- **Backtesting Results**: Performance metrics and visualizations from strategy evaluation.
- **Raw Market Data**: Historical price data used in both training and testing phases.
- **Training Models**: Serialized versions of trained machine learning models and related scripts.
- **Server Source Code (Python)**: Full implementation of the backend server, including strategy execution, real-time user handling, API integration, and encrypted key management.
- **Mobile App Source Code (Flutter)**: The complete Flutter codebase for the mobile client, enabling user interaction with the trading system and live performance tracking.
- **Gantt Chart**: Project timeline illustrating task distribution and milestones.
- **Presentation Slides**: The final project presentation submitted to the examination committee.

All files are included in the submitted digital package as standalone files.