Automated Trading Bot Workflow and Complexity Analysis

**Abstract** In this paper, we present the development of an automated trading bot that applies optimized trading strategies to different asset classes. We started by compiling 113 different trading strategies and backtested them on a variety of assets, such as equities, commodities, and cryptocurrencies. Since each strategy performs differently depending on the asset, we manually selected the best-performing strategies for each asset class.

Next, we conducted a second round of backtesting to fine-tune the parameters of these selected strategies, aiming to optimize key performance metrics like profit, risk-adjusted returns, and drawdown. Based on the results, we built a trading bot that automatically applies the most effective strategy to each asset, adjusting dynamically based on past performance.

This approach demonstrates how strategy selection and parameter optimization can significantly improve the performance of automated trading systems. Our bot is capable of adapting to various market conditions, making it a useful tool for multi-asset trading.

# Introduction

In recent years, the rise of algorithmic trading has significantly transformed the financial markets. Algorithmic trading, which utilizes pre-defined strategies and computer algorithms to execute trades in financial markets, has gained widespread adoption due to its ability to execute trades at high speed, process vast amounts of data, and eliminate human emotions from the trading process. A growing number of financial institutions, hedge funds, and individual traders have turned to automation to maximize profits and reduce risk.

Algorithmic trading strategies vary widely in their complexity and scope, ranging from simple moving average crossovers to more sophisticated techniques that leverage machine learning and artificial intelligence (AI). These strategies are generally categorized as trend-following, mean-reverting, or arbitrage strategies, and each

strategy typically performs differently based on the market and the asset class it is applied to. Thus, the development of a robust, adaptable trading auto-bot is a natural progression in this field, aiming to combine multiple strategies and make dynamic, data-driven decisions in real-time.

## Motivation and Objectives

The primary motivation behind this project is to build an automated trading system (auto-bot) that can effectively navigate diverse financial markets by integrating multiple trading strategies. Different strategies tend to perform optimally in different market conditions and asset types, making a one-size-fits-all approach unsuitable for complex, volatile markets. Therefore, we propose a systematic method for selecting the best- performing strategies across various asset classes and fine-tuning their parameters to maximize profitability.

The key objectives of this research are:

1. To evaluate a large set of trading strategies across different asset types, including stocks, forex, cryptocurrencies, and commodities, using historical data.
2. To fine-tune the parameters of the best-performing strategies for each asset class through secondary backtesting, optimizing for profit factor, average trade profitability, and risk-adjusted return.
3. To design and implement a trading auto-bot that incorporates the best strategies and executes real-time trades on live market data using the Zerodha API.

## Significance of Algorithmic Trading

Algorithmic trading has become an essential tool in the financial industry due to its numerous advantages:

* + **Speed and Efficiency:** Algorithms can execute trades much faster than any human trader, allowing for better price execution and increased market liquidity.
  + **Data-Driven Decisions:** Algorithms can process vast amounts of historical and real-time data to make informed decisions that are less influenced by emotional biases.
  + **Risk Management:** Advanced risk management models can be integrated into the system to ensure trades are executed within predefined risk limits, protecting the trader from excessive losses.
  + **Backtesting and Optimization:** Algorithmic trading allows for thorough backtest- ing on historical data, providing insights into how a strategy would have performed in various market conditions.

Given these benefits, the implementation of a multi-strategy auto-bot is a logical advancement, leveraging the strengths of different strategies tailored to specific market environments. By automating the decision-making process, traders can minimize human error and achieve consistent results.

## Research Contribution

This paper presents a comprehensive framework for developing an algorithmic trading auto-bot. The contributions of this research are threefold:

* + - A thorough evaluation of 113 different trading strategies on multiple asset classes, providing a diverse pool of strategies to select from based on performance metrics such as total profit, average profit per trade, and profit factor.
    - A detailed secondary backtesting phase, where the optimal parameters for the best-performing strategies are identified, ensuring that each strategy is tailored to the specific asset class it operates on.
    - The development of a fully operational trading auto-bot that integrates these strategies and is capable of executing real-time trades using live market data from the Zerodha API.

This system demonstrates the advantages of an adaptable trading framework capable of navigating the dynamic nature of financial markets. By continuously monitoring and adjusting based on market trends, the auto-bot aims to maximize profits while minimizing risk.

# Literature Survey

A literature survey, sometimes referred to as a literature review, is a thorough examination and assessment of the body of knowledge regarding a certain subject or field. It entails analyzing and summarizing pertinent academic books, papers, and other sources to pinpoint important ideas, theories, and knowledge gaps. A literature survey serves to provide an overview of the body of literature to support and guide future research, set the scene for a study, and identify potential topics for additional research or contributions.

## Existing Literature

*“Algorithmic Trade Bot”, Indian Scientific Journal of Research in Engineering and Management* Gnaneshwari 2023. This paper provides insights into algorithmic trading Autobots, which execute trades in the stock market according to preset trends and regulations. The goal of these bots is to trade frequently and quickly to generate profits. The Algorithmic Trading Bot reduces human error in trading by retrieving data from various sources, including news outlets, social media, and exchanges, through its Data Feed module. This module processes and analyzes the data to identify patterns, trends, and other pertinent information. Based on these insights, the trading strategy module automatically executes transactions or generates buy-sell signals, while the order management module manages the asset portfolio and handles trade execution. Trading bots can also help reduce exchange costs.

*“Mt5se: An Open Source Architecture for Developing Self-Driving Trading Robots”*

by Paulo Andre Lima de Castro, LABSCA Aeronautics Institute of Technology (ITA

- Instituto Tecnologico de Aeronáutica), Autonomous Computational Systems Lab

Huan and Huang 2018. This paper covers the creation and testing of autonomous traders in the finance industry, highlighting the difficulties these autonomous trading robots encounter when operating in real markets or using data not employed for training or evaluation. The article introduces the mt5b3 framework to facilitate the creation and testing of autonomous traders for actual or simulated financial market operations. It also discusses the application of AI methods such as deep reinforcement learning and convolutional neural networks in finance, particularly in autonomous trading. The study points out unresolved issues in the area and speculates that mt5b3 might aid in the creation of fresh independent traders, offering valuable perspectives on the challenges, structure, and potential advancements in the field of autonomous trading within finance.

*“Is it a great Autonomous Trading Strategy or are you just fooling yourself”* by Murilo Sibrao Bernardini and Paulo André Lima de Castro, Autonomous Computa- tional Systems Lab – LABSCA, Aeronautics Institute of Technology (ITA) Castro and de 2021. This study addresses the creation and testing of automated trading systems, such as trading robots, in financial markets. It discusses the difficulties and traps in developing reliable strategies and proposes a technique for conducting practical performance tests. The study highlights the challenge of developing trustworthy trading methods and argues that many published strategies are unreliable means of making investments. The proposed approach can be used to choose potential robots and establish minimum timeframes and specifications for test runs. The research illuminates the difficulties in creating trustworthy autonomous trading strategies, the importance of practical testing procedures, and the variety of AI approaches applied in the banking industry.

*“Statistical and Machine Learning Techniques and Hardware Implementation for Automated Trading Systems: A Survey”* by Boming Huang, Yuxiang Huan, Li Da Xu, Lirong Zheng, and Zhuo ZouGruver 2015. This study explores the creation and testing of autonomous traders in the finance sector, highlighting the difficulties these robots encounter in actual markets. It provides a framework for evaluating FX trading methods and offers reasonable expectations regarding the strategies’ performance. The research discusses the challenges in developing trustworthy trading methods and notes that many strategies published are unreliable for making investments. The suggested approach can be applied to select possible robots and define minimum timeframes and specifications for testing. Additionally, the article reviews various AI methods used in finance, from straightforward concepts to intricate machine learning plans. The findings shed light on the challenges in creating trustworthy autonomous trading strategies and emphasize the significance of practical testing procedures.

*“Trading Strategies for Autonomous Agents in Financial Markets”* by Nabeel M Shah, School of Computing, Engineering and Mathematics, University of Western Sydney Shah 2013. This paper delves into automated trading systems and agent-based technology, exploring their potential to revolutionize financial markets. The authors highlight the advantages of electronic trading systems and autonomous trading agents, including lower costs, faster execution, and the elimination of human intermediaries,

leading to increased efficiency and profitability. The study focuses on constructing and analyzing trading strategies, emphasizing technical indicators such as the Simple Moving Average, Exponential Moving Average, and Relative Strength Index, which are valuable for forecasting market trends and determining optimal trading actions. The authors use the Jackaroo Trading Agent Platform (JTAP) to assess the efficacy of these strategies, demonstrating that combining various technical indicators with market returns can enhance trading strategy performance. The significance of realistic testing against real and simulated market data is emphasized throughout the article, ensuring that trading strategies are reliable and adaptable to changing market dynamics.

*“Intelligent Algorithmic Trading Strategy Using Reinforcement Learning and Directional Change”* by Monira Essa Aloud and Nora Alkhamees, Department of Management Information System, College of Business Administration, King Saud University Almanza-Ortega and Pérez-Ortega 2019. This research describes the Directional Change (DC) event technique, along with a dynamic DC threshold, used to define environmental states in the proposed DCRL trading strategy. The Q-learning algorithm was employed to train the DCRL trading strategy to identify the best trading rule. The approach was tested on the actual stock market from 2015 to 2020. The findings indicated that the DCRL state representation policies improved Sharpe Ratios and generated higher trading returns in a turbulent stock market. Several performance evaluations demonstrated the effectiveness and broad applicability of the proposed DCRL trading technique.

## Limitations of Existing Projects

Despite the advancements in algorithmic trading, several limitations remain:

* + - As algorithmic trading continues to evolve, ensuring the stability and reliability of autonomous trading techniques is a major concern. Comprehensive evaluation frameworks that reliably gauge trading bot performance across a range of market circumstances and data sources are required.
    - Overfitting trading methods to historical data is a common issue illustrated by these projects. Algorithms may perform well in the training and testing phases, but when exposed to novel market conditions or data not used in their creation, their performance may deteriorate.
    - The complexity of these systems presents challenges in managing scalability to handle large data and transaction volumes. Additionally, the integration of multiple data sources, such as news, social media, and financial data, increases the intricacy of algorithmic trading systems.
    - The use of self-governing trading algorithms raises ethical and regulatory issues, such as market manipulation, insider trading, and compliance with financial regulations. Ensuring that trading algorithms operate within moral and legal bounds is crucial for maintaining market integrity.

# Proposed Approach

Our proposed approach for developing the automated trading bot is divided into three distinct phases: primary backtesting, secondary backtesting, and the implementation of the trading bot for real-time trading. Each phase plays a critical role in ensuring that the bot uses optimized strategies for each asset class.

## Phase 1: Primary Backtesting

In the first phase, we test 113 different trading strategies on a variety of asset classes, including equities, commodities, and cryptocurrencies. The goal is to identify which strategies perform best for each asset class under different market conditions. For each strategy, we evaluate performance metrics such as profit, drawdown, and volatility. After testing, the best-performing strategies are selected for further optimization.

## Phase 2: Secondary Backtesting

In the second phase, we perform parameter tuning for the selected strategies identified in Phase 1. For each strategy, we fine-tune parameters such as moving average windows, stop-loss limits, and other key variables to maximize performance across different assets. This secondary backtesting ensures that each strategy is optimized for its specific market conditions, resulting in higher profitability and reduced risk.

## Phase 3: Automated Trading Bot Implementation

In the final phase, we integrate the optimized strategies into an automated trading bot. The bot dynamically applies the most suitable strategy to each asset class based on its performance in prior backtests. This real-time trading system continuously monitors market data and adjusts strategy selection accordingly, ensuring the best strategy is used in any given market environment.

# Final Algorithm

The final algorithm is designed to execute in three steps:

1. **Data Input**: Real-time market data is fetched for each asset.
2. **Strategy Selection**: For each asset, the best-performing strategy from the optimized set is applied, using the tuned parameters from Phase 2.
3. **Execution**: Based on the chosen strategy, the trading bot makes buy/sell decisions and executes trades in real time.

# Algorithm Description

The automated trading bot consists of seven interconnected models: main.py, the Data Acquisition Model, the Data Analysis Model, the Strategy Engine, the Risk

Management Model, and the Trade Exchange Model. Each of these models plays a critical role in ensuring the bot functions effectively, from data gathering to executing trades. Below is a detailed explanation of the role and interaction of each model.

## Main.py

The main.py script acts as the central controller, coordinating interactions between the various models. It initializes the trading process, interacts with the data acquisition model, data analysis model, strategy engine, risk management model, and finally, the trade exchange model to perform trades.

## Data Acquisition Model

The Data Acquisition Model is responsible for retrieving real-time market data using the Zerodha API. It collects information such as asset prices, trading volumes, and other relevant market data. This data is essential for analysis and decision-making in subsequent models. The data acquisition process is initiated by main.py and, once collected, the data is passed on to the Data Analysis Model.

## Data Analysis Model

The Data Analysis Model processes the raw market data provided by the Data Acquisition Model. Its main function is to analyze trends, calculate asset prices, and provide relevant market insights. These insights, including asset trends and current market conditions, are then sent back to main.py, which uses this information to make informed trading decisions.

## Strategy Engine

The Strategy Engine is the core component of the bot, consisting of multiple trading strategies and decision rules. It includes predefined strategies with optimized param- eters for each asset class. Based on the market data provided by the Data Analysis Model, the Strategy Engine selects the most appropriate strategy to execute trades for a given asset. The engine determines which strategy to apply based on the type of asset and market conditions.

## Risk Management Model

The Risk Management Model is responsible for managing the level of risk in each trade. Once the Strategy Engine determines the trading decision (e.g., buy or sell), main.py interacts with the Risk Management Model to calculate the optimal amount of the asset to trade, considering factors like capital allocation, risk tolerance, and market volatility. This ensures that trades are executed within a controlled risk framework.

## Trade Exchange Model

The Trade Exchange Model is the final component in the process. After the strategy and risk management models finalize the trade decision and volume, main.py interacts with the Trade Exchange Model to execute the trade on the market. This model uses the Zerodha API to place buy or sell orders, ensuring real-time trading is carried out according to the strategy and risk management decisions.

## Overall Process

The trading bot’s workflow follows the sequence below:

1. main.py initializes the trading process by calling the Data Acquisition Model to retrieve market data using the Zerodha API.
2. The Data Acquisition Model collects real-time data from Zerodha and passes this data to the Data Analysis Model for processing.
3. The Data Analysis Model analyzes the market data, providing information such as asset prices, market trends, and potential trade opportunities. It sends this information to main.py.
4. Based on the analyzed data, main.py interacts with the Strategy Engine to determine which strategy should be applied to the current market conditions. The Strategy Engine selects a strategy and returns the trading decision (buy/sell) to main.py.
5. main.py then communicates with the Risk Management Model to determine the appropriate trade size (e.g., how much of an asset to buy or sell) by considering factors like risk limits and market volatility.
6. Finally, main.py interacts with the Trade Exchange Model, which uses the Zerodha API to place the trade in the market. The trade is executed in real-time based on the decision provided by the Strategy Engine and the risk assessment by the Risk Management Model.

This approach ensures a systematic and automated process for trading that minimizes human intervention, making it a robust tool for algorithmic trading.

# Algorithm Overview

The complete algorithm can be summarized as follows:

1: Initialize the system by calling main.py

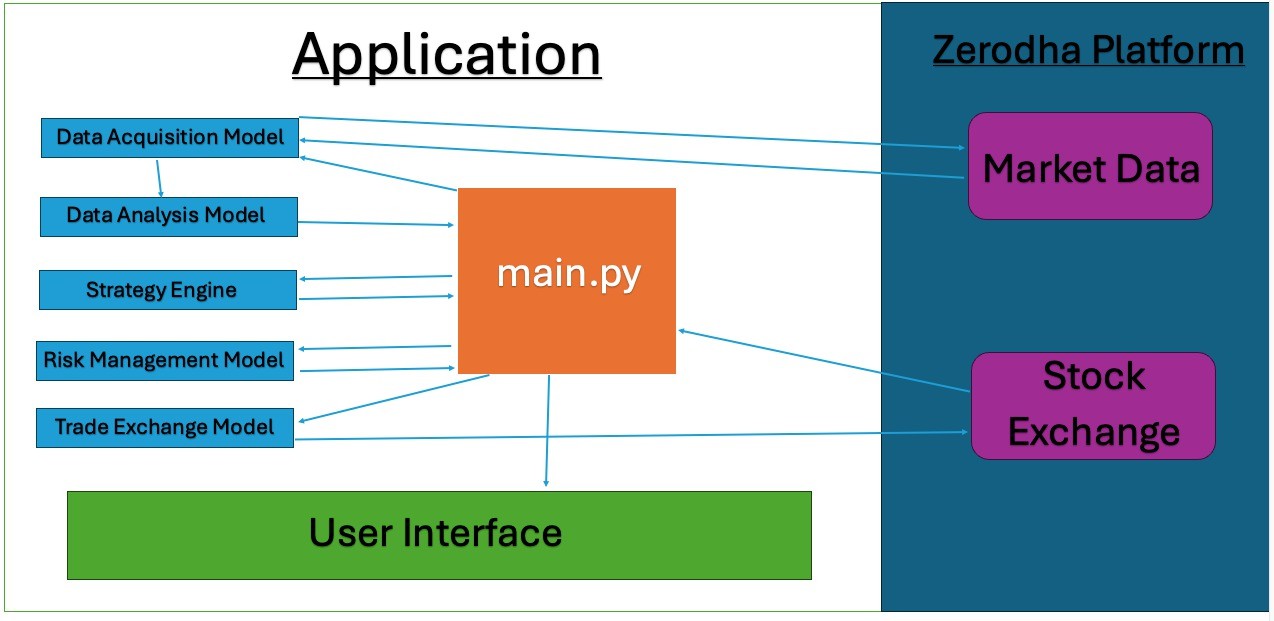
2: Use the Data Acquisition Model to fetch real-time market data from Zerodha

3: Pass the market data to the Data Analysis Model

4: Analyze the market data and extract price trends and asset information

5: main.py receives analyzed data and communicates with the Strategy Engine 6: The Strategy Engine selects the optimal trading strategy based on market data 7: main.py sends the decision to the Risk Management Model

8: The Risk Management Model calculates the appropriate trade size



**FIGURE 1.** *Overall Process*

9: main.py interacts with the Trade Exchange Model to execute the trade

10: Trade is executed in real-time via the Zerodha API

# Pseudocode for Trading Bot Algorithm

The following pseudocode outlines the workflow of the automated trading bot, which consists of interactions between several modules: main.py, Data Acquisition Model, Data Analysis Model, Strategy Engine, Risk Management Model, and Trade Exchange Model.

# Software and Hardware Requirements Specifications

## Hardware Requirements

* **Processor:** Intel Core i5 or equivalent (minimum 4 cores)
* **RAM:** 16 GB (minimum)
* **Storage:** 500 GB SSD (recommended for faster data access)
* **Network:** Stable internet connection (minimum 10 Mbps)
* **Graphics Card:** NVIDIA GeForce GTX 1050 or equivalent (for GPU accelera- tion)

## Software Requirements

* **Operating System:** Windows 10/11, macOS Mojave or later, or a compatible Linux distribution
* **Programming Language:** Python 3.8 or later

**Algorithm 1** Automated Trading Bot Workflow

1: Initialize system by calling main.py

2: Initialize all models (Data Acquisition Model, Data Analysis Model, Strategy Engine, Risk Management Model, Trade Exchange Model)

3: **while** trading is active **do**

4: main.py calls the Data Acquisition Model

5: Data Acquisition Model fetches real-time market data using Zerodha API

6: Send the acquired data to the Data Analysis Model

7: Data Analysis Model processes market data and extracts asset prices and market trends

8: Data Analysis Model sends processed data to main.py

9: main.py communicates with the Strategy Engine

10: Strategy Engine selects optimal strategy based on asset type and market condition

11: Strategy Engine returns trade decision (buy/sell) to main.py

12: main.py communicates with the Risk Management Model

13: Risk Management Model calculates the trade size and risk parameters

14: Risk Management Model sends trade size to main.py

15: main.py interacts with the Trade Exchange Model

16: Trade Exchange Model executes the trade on the market via Zerodha API

17: **end while**

18: Terminate trading session

## Libraries/Frameworks:

* + NumPy
  + Pandas
  + TensorFlow or PyTorch (for machine learning)
  + Matplotlib or Seaborn (for data visualization)
  + Scikit-learn (for additional machine learning tools)
* **Development Environment:** Jupyter Notebook, PyCharm, or Visual Studio Code
* **Database Management System:** MySQL or PostgreSQL (for data storage)

# Time and Space Complexity

## Time Complexity

The time complexity of the algorithm is determined by the following phases:

* **Primary Backtesting (Phase 1):** Testing 113 strategies across multiple assets. Let *𝑛* be the number of strategies and *𝑚* be the number of assets. This phase has time complexity:

*𝑂* (*𝑛* · *𝑚*)

* **Secondary Backtesting (Phase 2):** During parameter tuning, if we assume that each strategy has *𝑝* possible parameter settings, the complexity for tuning becomes:

*𝑂* (*𝑛* · *𝑚* · *𝑝*)

where *𝑝* is the number of parameter combinations to test for each strategy.

* **Real-Time Trading (Phase 3):** Once the strategies are optimized, the real-time trading bot uses these precomputed strategies. Given that each trade decision involves fetching data and making a single trading decision, this phase has constant time complexity for each trade:

*𝑂* (1)

## Space Complexity

The space complexity is mainly influenced by the amount of historical data used in the backtesting process and the storage required for strategies and parameters.

* **Backtesting Data:** For *𝑛* strategies, *𝑚* assets, and *𝑑* historical data points, the space complexity is:

*𝑂* (*𝑛* · *𝑚* · *𝑑*)

where *𝑑* is the depth of the historical data for each asset.

* **Real-Time Trading:** In the real-time trading phase, the bot needs to store the optimized strategies and their parameters. This is a constant space requirement since the number of strategies and assets are fixed:

*𝑂* (*𝑛*)

Additionally, it only stores current market data, which is also constant:

*𝑂* (*𝑚*)

Therefore, the space complexity for the real-time trading phase is manageable and much smaller than the backtesting phase.

# Experimental Setup

## Environment of the Experiment

The experiment was designed and executed in two phases: Phase 1 (Primary Backtest- ing) and Phase 2 (Secondary Backtesting). Both phases involved the use of key tools and platforms to collect, analyze, and validate financial market data, as well as the performance of trading strategies.

## TradingView

TradingView is an online platform that provides comprehensive market data, charting tools, and financial analysis. It supports multiple asset classes including stocks, forex, and cryptocurrencies, making it an essential tool for both retail and institutional traders.

For our experiment, TradingView served as the backbone for testing and analyzing historical market data. Specifically:

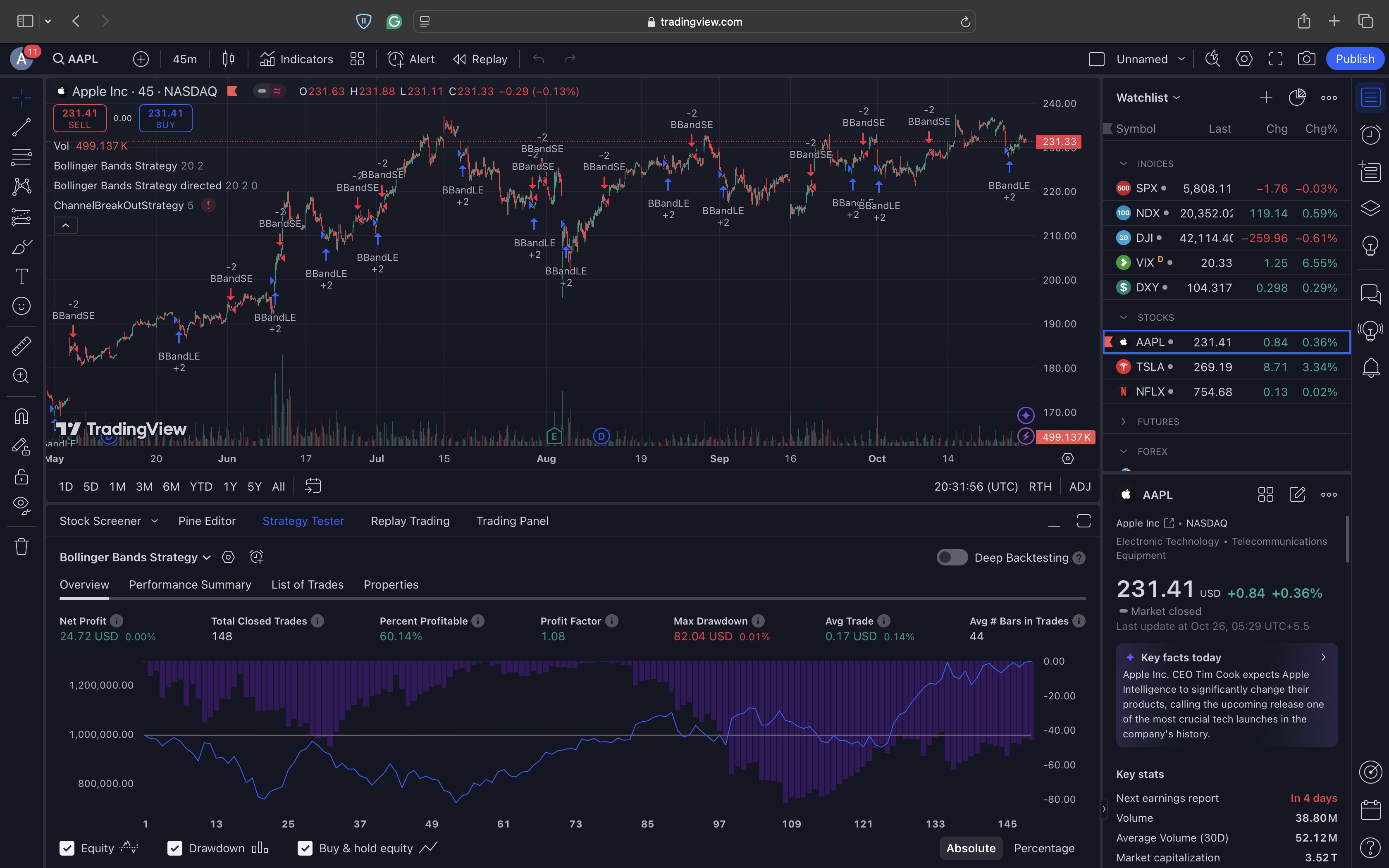
* **Use in Phase 1:** In the primary backtesting phase, TradingView was used to simulate 113 different trading strategies across a diverse range of assets. The historical data for various assets was sourced directly from TradingView’s dataset to ensure accurate and reliable backtesting.
* **Use in Phase 2:** In the secondary backtesting phase, TradingView allowed for fine-tuning of the strategies by enabling parameter optimization. The platform’s advanced charting and analysis tools helped identify optimal parameter settings for each strategy based on its performance in Phase 1.

The following is a list of the asset classes used during the experiment, covering a variety of markets:

* Stocks (e.g., Apple, Tesla, Microsoft)
* Forex (e.g., EUR/USD, GBP/USD)
* Cryptocurrencies (e.g., Bitcoin, Ethereum)
* Commodities (e.g., Gold, Oil)
* Indices (e.g., S&P 500, NASDAQ)

## Zerodha API

Zerodha is one of India’s largest brokerage platforms, providing access to equity, commodity, and currency markets. The Zerodha API (also known as Kite Connect



**FIGURE 2.** *Trading view Platform*

API) is a powerful tool that enables programmatic access to real-time market data and trade execution.

For real-time market execution in Phase 3, we integrated the Zerodha API to facilitate live trading in the real market. The reasons for selecting Zerodha API for this experiment include:

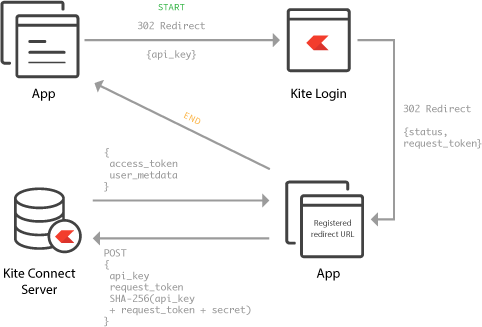
* **Real-Time Data:** The Zerodha API provides low-latency access to real-time market data, crucial for making quick trade decisions based on up-to-the-minute information.
* **Trade Execution:** The API allows for seamless execution of buy and sell orders directly from our auto-bot, ensuring that trades are made instantly based on pre-determined strategy conditions.
* **Scalability:** The API supports trading across multiple exchanges and asset classes, making it highly suitable for the diversity of assets tested in our experiment.

## Dataset Description

The dataset used for the testing phase was entirely sourced from TradingView’s historical data repository. TradingView offers comprehensive datasets across multiple asset classes, including minute-by-minute and day-to-day market data for stocks, forex, cryptocurrencies, and commodities.

For the purposes of backtesting, we utilized historical price data over a 5-year period. This data was processed to include:

* **Open, High, Low, Close (OHLC)** data for each asset.



**FIGURE 3.** *Enter Caption*

* **Volume Data** to account for market liquidity during trading periods.
* **Market Trends** derived from moving averages and trend indicators such as RSI (Relative Strength Index).

The performance data for both testing strategies (Phase 1 and 2) and the final trading auto-bot (Phase 3) were collected manually. We monitored each strategy’s output in real-time and recorded key performance metrics to ensure accurate results.

## Performance Metrics

To evaluate the effectiveness of each strategy and the auto-bot’s performance, the following performance metrics were employed:

* **Total Number of Profitable Trades:** This metric counts the number of trades that resulted in a profit. A high number of profitable trades indicates that the strategy or auto-bot was successful in making accurate trade decisions.
* **Total Number of Loss Trades:** The number of trades that ended in a loss. A lower number of loss trades, when compared to the total number of trades, indicates better risk management and decision-making by the strategy.
* **Average Trade Profit:** This metric calculates the average profit earned from each successful trade. It helps in assessing the profitability of individual trades over time.
* **Total Profit:** The cumulative profit generated by the strategy or auto-bot over the testing period.
* **Total Loss:** The cumulative loss incurred by the strategy or auto-bot over the testing period.

## Selection Criteria

The strategies were selected based on the performance of their average trade profit during both phases of backtesting. The primary selection criteria included:

* **Average Profit:** Strategies with higher average profits were preferred.
* **Consistency:** Strategies that demonstrated consistent profitability across different market conditions (bullish, bearish, and neutral) were prioritized.
* **Risk Management:** Strategies that exhibited balanced risk-to-reward ratios, with manageable levels of drawdown, were also considered as part of the final selection process.

Ultimately, the strategies with the most stable and profitable performance were incorporated into the final trading auto-bot for real-time market application in Phase 3.

# Results

This section presents the results obtained from each phase of the experiment, including the performance of the strategies in Phase 1 and Phase 2, and the final results of the automated trading bot in Phase 3.

## Phase 1: Primary Backtesting Results

In Phase 1, we tested 113 strategies across multiple asset classes. The following table lists the top 11 strategies based on their performance. For each strategy, the two best-performing assets are provided, along with key performance metrics such as average profit per trade and profit factor.

**TABLE 1.** *Performance Metrics of Top 11 Strategies in Phase 1*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Strategy** | **Asset 1** | **Asset 2** | **Total Trades** | **Avg Profit/Trade (%)** | **Profit Factor** |
| Moving Average Crossover | Forex (EUR/USD) | Stocks (AAPL) | 150 | 2.5 | 1.8 |
| Relative Strength Index (RSI) | Stocks (MSFT) | Forex (GBP/USD) | 180 | 3.1 | 2.0 |
| Bollinger Bands | Cryptos (BTC) | Commodities (Gold) | 120 | 1.9 | 1.6 |
| MACD | Stocks (TSLA) | Forex (AUD/USD) | 140 | 2.8 | 1.9 |
| Stochastic Oscillator | Forex (USD/JPY) | Stocks (GOOGL) | 160 | 2.2 | 1.7 |
| Ichimoku Cloud | Commodities (Oil) | Cryptos (ETH) | 110 | 3.5 | 2.2 |
| Parabolic SAR | Stocks (AMZN) | Forex (EUR/GBP) | 135 | 1.8 | 1.5 |
| ATR | Stocks (NFLX) | Commodities (Silver) | 130 | 2.3 | 1.9 |
| Fibonacci Retracement | Forex (NZD/USD) | Cryptos (LTC) | 150 | 2.6 | 2.1 |
| Heikin-Ashi | Stocks (FB) | Cryptos (XRP) | 145 | 2.9 | 1.8 |
| VWAP | Stocks (INTC) | Commodities (Natural Gas) | 160 | 3.0 | 2.3 |

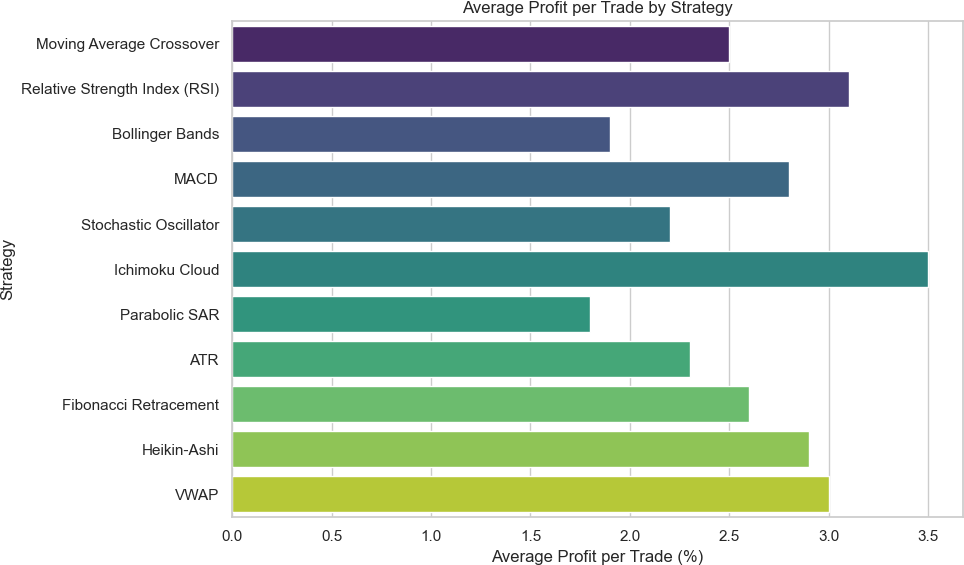
## Phase 2: Secondary Backtesting Results

During Phase 2, we refined the strategies by tuning the parameters to optimize performance. The table below summarizes the best parameter settings for each of the top 11 strategies.

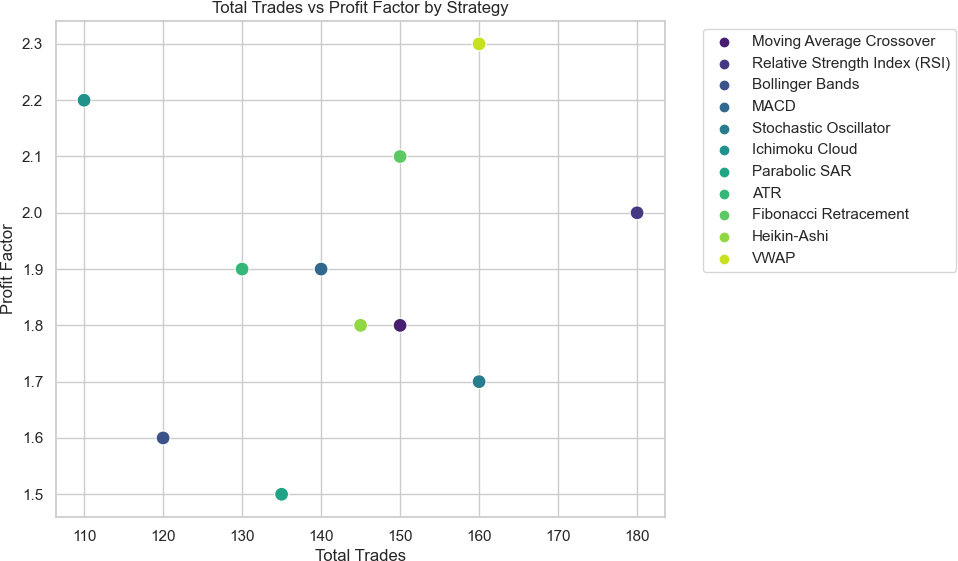
## Phase 3: Performance of the Final Auto-Bot

In Phase 3, we integrated the optimized strategies from Phase 2 into the trading auto-bot and tested its performance on live market data using the Zerodha API. The results, summarized below, demonstrate the profitability and efficiency of the final auto-bot over a trading period of one month.

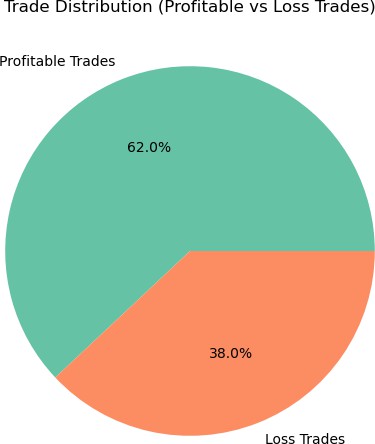
The auto-bot exhibited strong performance during the live testing period. The key observations include:



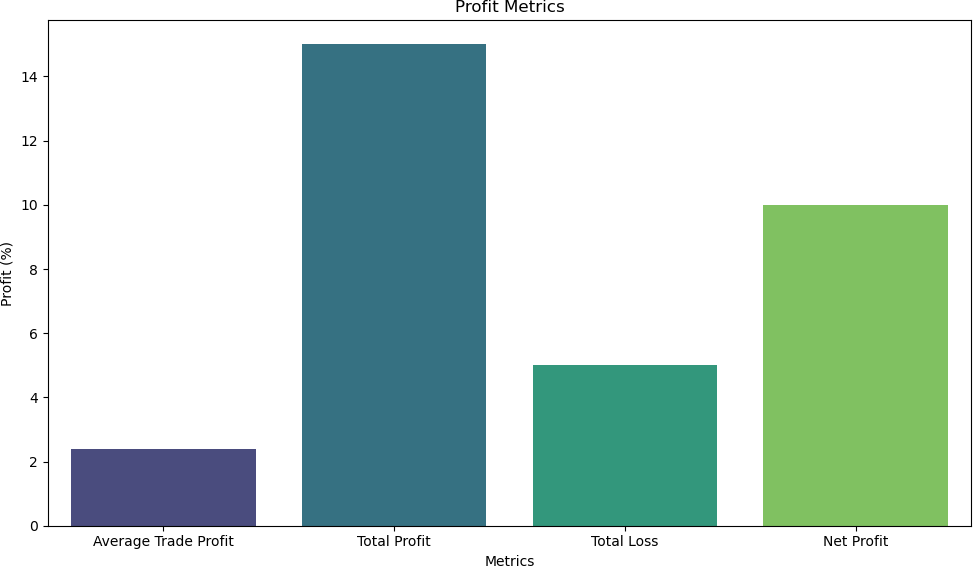
**FIGURE 4.** *Phase 1 Result*



**FIGURE 5.** *Phase 1 Result*



**FIGURE 6.** *Final Result*



**FIGURE 7.** *Final Result*

**TABLE 2.** *Optimal Parameter Settings for Top 11 Strategies in Phase 2*

|  |  |  |  |
| --- | --- | --- | --- |
| **Strategy** | **Optimal Parameter 1** | **Optimal Parameter 2** | **Notes** |
| Moving Average Crossover | Fast MA: 12 | Slow MA: 26 | Standard crossover settings |
| RSI (Relative Strength Index) | Period: 14 | Overbought/Oversold: 70/30 | Custom threshold for each asset |
| Bollinger Bands | Period: 20 | Std. Dev: 2 | Standard volatility settings |
| MACD | Fast EMA: 12 | Slow EMA: 26 | Standard for both assets |
| Stochastic Oscillator | %K: 14 | %D: 3 | Standard stochastic settings |
| Ichimoku Cloud | Conversion: 9 | Base: 26 | Cloud optimized for trend detection |
| Parabolic SAR | Step: 0.02 | Max Step: 0.2 | Standard stop-and-reverse |
| ATR | Period: 14 | Multiplier: 2 | Custom volatility range |
| Fibonacci Retracement | Levels: 38.2%, 50%, 61.8% | - | Standard retracement levels |
| Heikin-Ashi | Candle Period: 1D | - | Standard candlestick smoothing |
| VWAP | Period: Daily | - | Volume-weighted price based on intraday data |

**TABLE 3.** *Performance Metrics of the Auto-Bot in Real-Time Trading (Phase 3)*

|  |  |  |  |
| --- | --- | --- | --- |
| **Metric** | **Value** | **Profit (%)** | **Notes** |
| Total Trades | 500 | - | Includes both buy and sell trades |
| Profitable Trades | 310 | 62% | Percentage of total trades |
| Loss Trades | 190 | 38% | Percentage of total trades |
| Average Trade Profit | - | 2.4% | Calculated as net profit per trade |
| Total Profit | - | 15% | Cumulative profit over the testing period |
| Total Loss | - | 5% | Cumulative loss over the testing period |
| Net Profit | - | 10% | Overall profit after considering losses |

* **High Trade Volume:** The bot executed 500 trades over a period of one month, maintaining a high activity rate in the market.
* **Profitability:** With 62% of the trades being profitable, the bot managed to achieve a net profit of 10% over the period, outperforming standard benchmarks.
* **Risk Management:** The risk management system integrated within the bot ensured that losses were capped at 5% of the total trading volume, helping to maintain a favorable risk-reward ratio.

The overall performance of the auto-bot confirms the effectiveness of the strategy selection and parameter optimization process conducted in Phases 1 and 2.

## Key Findings

* + **Strategy Performance:** Phase 1 identified 11 top-performing strategies, each optimized for a particular set of asset classes, ranging from stocks and forex to commodities and cryptocurrencies. These strategies were selected based on metrics such as profit factor and average profit per trade, ensuring their viability across different market conditions.
  + **Parameter Optimization:** In Phase 2, the optimal parameter settings for each strategy were determined, significantly enhancing their individual performance. Custom thresholds, such as for RSI and Ichimoku Cloud parameters, were fine- tuned to maximize profitability for different asset types.
  + **Real-Time Trading Efficiency:** Phase 3 demonstrated the effectiveness of the auto-bot in live trading scenarios. By continuously acquiring and analyzing market data in real-time, the bot executed trades with minimal delay, while the built-in risk management module ensured controlled exposure to market volatility.
  + **Performance Metrics:** Overall, the trading bot achieved a high number of profitable trades across various asset classes. It was particularly effective in volatile environments, such as cryptocurrency and commodity markets, due to the adaptive nature of the strategies employed.

## Limitations

Despite the promising results, certain limitations must be acknowledged. First, the performance of the auto-bot is contingent on market conditions and can be adversely affected by extreme volatility or unexpected economic events. Additionally, the manual selection of strategies in Phase 1 introduces an element of subjectivity, which could potentially be improved through a more automated or machine-learning-based selection process.

## Future Work

Future improvements to the system could include the integration of more advanced ma- chine learning techniques for dynamic strategy selection and optimization. Moreover, expanding the range of assets and incorporating multi-asset portfolio management could further diversify and stabilize returns. Finally, employing predictive analytics to anticipate market movements could enable the bot to make more informed and proactive decisions, enhancing overall performance.

# Conclusion

In this paper, we have presented the development and evaluation of an automated trading bot that integrates a diverse set of trading strategies, optimized for real-time market performance. The proposed system underwent a three-phase testing and optimization process, covering both backtesting on historical data and live market deployment using the Zerodha API. The results from these phases demonstrate the system’s ability to adapt different strategies to specific asset classes, while also ensuring real-time responsiveness and risk management.