parcIT

FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Bachelor's Thesis

in the Bachelor's Program 'Applied Mathematics and Computer Science'

Backtesting and Live-Testing of Classic and AI-Powered Trading Strategies

by

Jason Becker

Enrollment Number: 3567285

Supervisor:	Prof. Dr. Bodo Kraft
Co-Supervisor:	Hendrik Karwanni, M. Sc.
Submitted on:	30th July, 2025

Declaration of Authorship

This bachelor thesis has been written by:

Jason Becker
Wachtelstraße 24
40789 Monheim am Rhein

I hereby declare that this bachelor thesis has been written independently and without unauthorized assistance. I confirm that I have used only the sources and aids indicated in the bibliography. Furthermore, I declare that I have marked all passages that are either verbatim or paraphrased versions of sources.

I am aware that any form of plagiarism or dishonesty in academic work constitutes a serious offense and may lead to disciplinary measures.

Monheim am Rhein, 30th July, 2025

.....
Jason Becker

Abstract

Contents

List of Figures

List of Tables

Code Listing

List of Abbreviations

ETH	Ethereum
BTC	Bitcoin
USDC	USD Coin
M1	Minute-Level
PnL	Profit and Loss
TA	Technical Analysis
CFD	Contract For Difference
EMA	Exponential Moving Average
SMA	Simple Moving Average
MACD	Moving Average Convergence/Divergence
ATR	Average True Range
TR	True Range
RSI	Relative Strength Index
RRR	Risk-Reward Ratio
DD	Drawdown
MDD	Maximum Drawdown
PCA	Principal Component Analysis
AI	Artificial Intelligence
MLP	Multilayer Perceptron
FNN	Feedforward Neural Network
RNN	Recurrent Neural Network
GRU	Gated Recurrent Unit
LSTM	Long Short-Term Memory
CNN	Convolutional Neural Network
ReLU	Rectified Linear Unit
1D	One Dimensional
SPI	Service Provider Interface
JAR	Java Archive
API	Application Programming Interface
Min	Minimum
Max	Maximum
No	Number
UTC	Coordinated Universal Time
PM	Post Meridiem

1 Introduction

1.1 Motivation

In recent years, the cryptocurrency market has emerged as a highly dynamic and rapidly evolving financial ecosystem. Trading pairs such as ETH/USDC M1 on minute-level (M1) intervals provide vast amounts of high-frequency data, reflecting extreme volatility and market shifts. This creates challenges and opportunities for traders and researchers.

The availability of detailed tick and minute data, combined with direct API access from crypto brokers, creates new opportunities for developing data-driven trading systems. Advances in machine learning and deep learning offer promising tools to identify patterns in market movements. This motivates an exploration of AI-based trading systems specifically designed for the ETH/USDC M1 pair, aiming to leverage technical features and advanced models to improve performance in the volatile cryptocurrency environment.

On the other hand, classical trading strategies also could perform well in this environment, especially if the strategies are adapted to specific market regimes. Techniques such as moving averages, Bollinger Bands, or momentum indicators have the advantage of simplicity and transparency, allowing traders to understand and trust their decision-making process.

The ETH/USDC trading pair at the minute timescale was deliberately chosen because it offers several key advantages for analyzing and modeling algorithmic trading strategies. Ethereum is one of the largest and most liquid cryptocurrencies in the world, while USDC, as a stablecoin, is subject to lower price fluctuations. This combination allows for a clear separation between the volatile component (ETH) and a stable reference currency (USDC), simplifying analysis. The high liquidity of this pair also ensures a reliable data basis with low slippage and tight spreads. In particular, the M1 timescale provides sufficiently granular information to identify short-term market movements and micro-patterns without completely relying on high-frequency tick data. This creates a practical scenario for the development and evaluation of trading algorithms that are both data-driven and operationally realistic.

1.2 Aim of this Paper

The primary goal of this paper is to develop and evaluate AI-powered and classical trading strategies focused on the ETH/USDC M1 cryptocurrency pair. This paper will:

1. Retrieve historical ETH/USDC M1 data via a broker API and process it for analysis.

2. Perform exploratory data analysis and feature engineering, including trend, volatility, and momentum indicators.

3. Identify and classify distinct market regimes to provide adaptive trading decisions.

4. Apply risk and money management techniques to realistically simulate trading performance.

5. Design, train, and benchmark multiple deep learning architectures such as LSTM, CNN, and hybrid models for forecasting the price and classify trading decisions.

6. Develop algorithmic trading strategies based on AI model predictions and compare them with classical technical trading approaches.

7. Build a modular trading engine capable of backtesting trading strategies and live execution using real broker connections.

8. Execution of the final best trading strategy in live operation on a demo account with a real broker.

By focusing on the ETH/USDC M1 pair, this paper aims to provide insights into the effectiveness of deep learning and classical strategies in cryptocurrency trading and contribute practical tools for automated, adaptive trading in this challenging asset class.

1.3 Structure of this Paper

section 2 first lays the foundation for technical integration with financial markets. This includes selecting a suitable broker and setting up an automated interface for data retrieval. Without this foundation, real-time access to market data and the subsequent live execution of strategies are not possible.

section 3 and section 4 introduce essential concepts such as market regimes and money and risk management. These theoretical foundations serve to analyze the markets in a structured manner while ensuring that trading decisions are made within the framework of sound risk management. Classification approaches for market phases as well as rules for position sizing and risk limitation are covered.

Building on this, section 6 provides a comprehensive exploratory data analysis (EDA) approach, describing key steps for processing and transforming historical price data. In addition to generating technical indicators, modern methods such as feature scaling and principal component analysis (PCA) are also discussed as a foundation for machine learning.

Subsequently, in section 7, various deep learning models are developed and trained to enable predictions of future market movements. Both regression and classification models are examined and evaluated for their suitability for use in trading.

section 8 is devoted to the development of concrete trading strategies. These are based either on classic technical indicators or on signals derived from the previously developed ML models. Each strategy is methodically described and prepared for later evaluation.

To execute and evaluate these strategies, section 9 introduces a modular trading engine that enables both backtesting and live trading. The structure follows a plug-in architecture to flexibly integrate different strategies.

section 10 verifies the performance of the developed strategies using backtests. Various metrics such as profit curves, drawdowns, and hit rates are used. The impact of fees is also considered to ensure a realistic evaluation.

section 11 describes the transition to practice by conducting a live test. This section is intended to demonstrate the extent to which the results observed in the backtest can be confirmed in the real market environment. The results obtained are then critically discussed.

section 12 concludes, summarizing the key findings and drawing conclusions. section 13 provides suggestions for further work, for example, regarding improvements to the models, strategies, or engine extensions.

2 Data Source and Broker Selection

Cryptocurrency brokers (also called crypto brokers) play an important role in cryptocurrency trading. Among other things, they act as intermediaries between different market participants. Their key tasks include:

1. **Providing access:** Individuals can participate in the market through a broker and thereby trade various cryptocurrencies. This includes executing orders such as buying cryptocurrencies at the lowest available price or selling them at the highest available price.

2. **Security, and Compliance:** They also provide customers with a secure platform for executing transactions and adhere to the financial regulations established by authorities.

3. **Leveraging:** Brokers offer customers the opportunity to borrow money, and thus trade with more capital than they actually have in their account.

This has the advantage that trading with cryptocurrencies is much easier and safer, but one of the biggest disadvantages is the fees that are incurred when using [1].

2.1 Broker Selection

For this paper, one broker must be selected for data retrieval and live testing. Since the process is fully automated in short time-frames, the broker must meet certain requirements.

The API must be able to stream market data, request historical data, the current account balance, closed trades, and currently open positions, placing orders, and positions, as well as canceling unfilled orders.

Apart from the API, the broker must support leveraged long/short products like CFDs or margin trading, with the lowest possible fees. They also must provide data in high quality as well as a demo depot.

Table 1 summarizes the required features for some potential brokers. All listed there meet the API functionality requirements.[1]

Broker	Tradable assets	Fees		Leverage
		Maker	Taker	
ByBit	Spot, Spot with Leverage, Futures, Options	0.02%	0.055%	10:1
IG	CFDs, Knock-out-Options	Spread (approx. $1.30)		2:1
Capital.com	CFDs	Spread (approx. $1.75)		2:1

Table 1: Broker Comparison

[1] Sources: [2], [3], [4], [5], [6], [7]

Although ByBit's fees are calculated as a percentage of the transaction volume, they are lower overall than those charged by IG or Capital.com. The average price of ETH is approximately 2280 USDC [2]. Based on the average ETH price, ByBit's fees are approximately 1.25 USDC for a taker order and 0.45 USDC for a maker order per ETH bought or sold. This means it is possible to save up to 1.3 USDC compared to IG and Capital.

Taking into account Table 1, ByBit is the best broker because it has the lowest fees, high quality data, as well as the highest possible leverage.

2.2 API Connection and Data Retrieval Process

Before starting with the Machine Learning process, and the backtests, the first step is to download historical ETH/USDC M1 data via the ByBit API. The request was executed on the `/v5/market/kline` API-Endpoint [8] with the category `linear`, symbol `ETHPERP`, and interval `1` at 17th June, 2025, 11:30 UTC. Since ByBit only returns 1000 candlesticks per request, the same request with different start-, and end-times was executed until the ByBit API does no longer return older candlestick data. This resulted in a candlestick data pool with data on a minute basis from 5th August, 2022, 10:00 UTC to 17th June, 2025, 11:30 UTC. Chapter 6 will go into more detail about the data.

[2] The average price was calculated using all data collected in subsection 2.2

3 Market Regimes

Cryptocurrency markets are subject to constant change, which is reflected not only in price movements but also in the underlying structures and dynamics. In quantitative analysis and algorithmic trading, understanding these changes is essential for developing and adapting robust trading strategies. A central concept in this context is market regimes.

3.1 Introduction to Market Regimes

Market regimes describe phases with distinct statistical and economic characteristics, such as volatility, and trend behavior. They can be understood as different "states" of the market in which certain trading patterns dominate. Distinguishing between, for example, upward, sideways, and downward trends or high and low volatility enables more targeted strategy selection and adaption. Accordingly, the identification and classification of market regimes plays an increasingly important role in modern trading analysis. For example, a strategy that performs well in a stable uptrend may fail in a sideways movement or in periods of high volatility [9].

Understanding market regimes leverages traders and analysts to design strategies that are adaptive, and therefore more robust. Through targeted adaptation of the parameters or the selection of different models, the performance can be increased, and the risk can be reduced.

3.2 How to Categorize the Market?

Categorizing different market characteristics is a common step in identifying market regimes. In literature and practical applications, there exist different approaches to classifying markets. A fundamental categorization is often made by the following dimensions:

1. **Trend behavior:** Markets can be categorized trend following (bullish/ bearish) or trendless (sideways) [10].
2. **Volatility:** The volatility of a market is often an indicator for insecurity or stability [11]. High volatility can indicate periods of stress, while low volatility indicates calm markets.
3. **Liquidity:** In illiquid markets, pricing processes can be different compared to liquid markets which has effect on strategies [12].

According to the analysis goal, the categorization can be binary (e.g., bullish vs. bearish) or granular (e.g., a combination of trend behavior, and volatility). While binary classification may be sufficient for high-level market orientation, more granular categorization enables a finer distinction of market conditions.

In the current context, the market is categorized by trend behavior, and volatility. This results in six categories:

1. Downtrend + Low Volatility
2. Downtrend + High Volatility
3. Sideways Trend + Low Volatility

4. Sideways Trend + High Volatility
5. Uptrend + Low Volatility
6. Uptrend + High Volatility

This takes into account the two most central aspects that lead traders to different trading decisions.

3.3 Recognizing Market Regimes

As described in subsection 3.2, the market will be divided into six categories which are the result of combinations of two individual categories. This makes it possible to categorize the two individual categories individually, and finally merge them.

3.3.1 Recognizing Trend Behavior

The first step is to categorize the market into uptrends, downtrends, and sideways trends. Commonly, a combination of a short-term simple moving average (e.g., SMA(50)), and a long-term simple moving average (e.g., SMA(200)) is used to identify superior trends. The SMA(200) is considered the classic boundary between bullish and bearish market phases. If the short-term simple moving average is above the long-term simple moving average, the market is considered bullish. Vice versa, if the short-term simple moving average is below the long-term simple moving average, the market is considered bearish [13].

For the purpose of the current context, a modified combination of SMA(50), and SMA(100) was chosen. This decision is based on two considerations:

1. **Faster reaction:** The SMA(100) is intended to achieve faster reaction to medium-term trend changes without heavily weighting short-term volatility.
2. **Inertia of trend definition:** A shorter trend window, compared to the SMA(200) reduces the inertia of trend definition, which can be particularly advantageous for more refined classification into uptrends, downtrends, and sideways trend.

Additionally, a minimum slope threshold over the last 15 minutes was integrated for the SMA(50) to avoid that minimal direction changes are mistakenly interpreted as a meaningful trend. This short time span ensures that current market movements are adequately incorporated into the trend classification without being dominated by short-term noise (e.g., individual volatility peaks), and therefore increases the robustness of the trend recognition, and addresses the weaknesses of simple moving averages in sideways phases. A slope above +0.05 signals a significant short-term uptrend, while a slope below -0.05 suggests a clear downtrend. Values in between are interpreted as ambiguous, and are included in the sideways classification accordingly.

The market can therefore be divided into three trend phases based on the following criteria:

1. **Uptrend:** The SMA(50) is above the SMA(100), and the slope of the SMA(50) in the last 15 minutes is greater than 0.05.

2. **Downtrend:** The SMA(50) is below the SMA(100), and the slope of the SMA(50) in the last 15 minutes is less than -0.05.

3. **Sideways trend:** The market currently does not meet condition 1 or 2.

Figure 1 shows an example of trend classification of ETH/USDC M1 for 700 minutes.

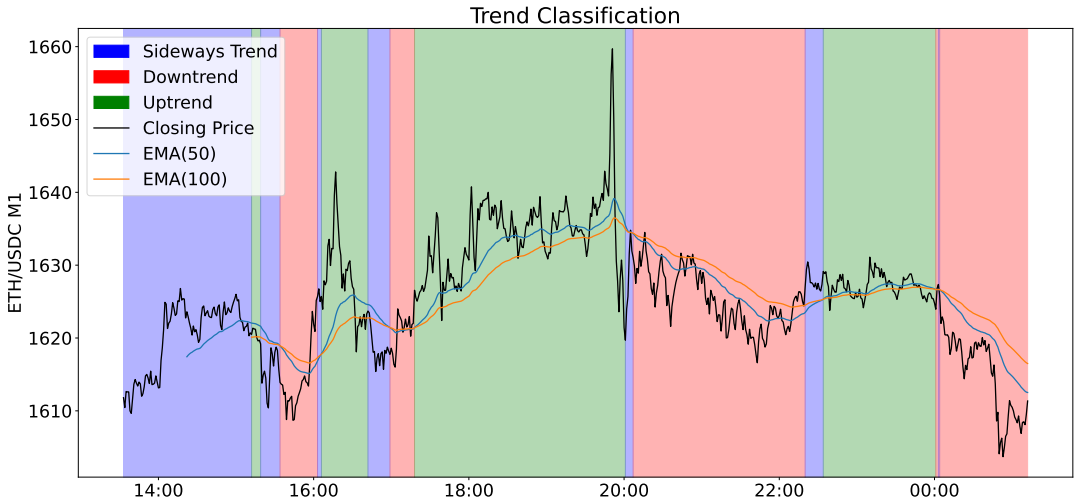

Figure 1: Trend Classification

3.3.2 Recognizing Volatility

The second step is to categorize the market into phases with high and low volatility. The volatility is calculated as the standard deviation of the logarithmic returns over the last 30 minutes [14]. This locally calculated volatility depicts short-term fluctuation intensity, and enables a context-dependent assessment of current market behavior.

To classify this local volatility, a comparison is made with the median of all available volatilities in the training dataset which was used for fitting the volatility classification indicator. If the current volatility is greater than the median, the market is classified as highly volatile. Otherwise, the market is classified as low volatility.

This threshold definition is deliberately based on a dynamic, data-dependent approach rather than using a fixed absolute threshold. This automatically adapts the volatility classification to each specific market, and can therefore theoretically be applied to other financial markets.

Figure 2 shows the volatility classification on the same base data used in Figure 1.

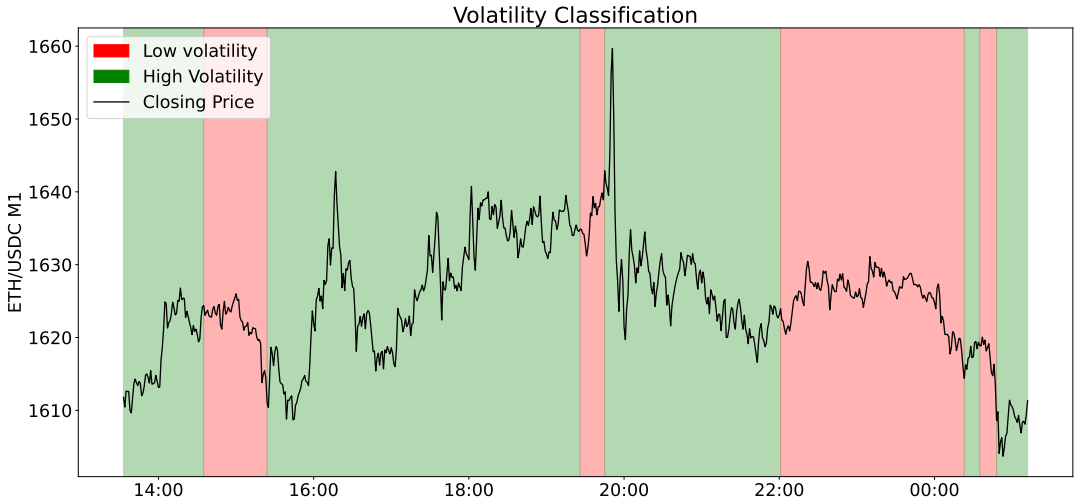

Figure 2: Volatility Classification

3.3.3 Combining Trend and Volatility Classification

After the two separate classifications in subsubsection 3.3.1 and subsubsection 3.3.2 the results can be combined. The results of the combination can be seen in Figure 3.

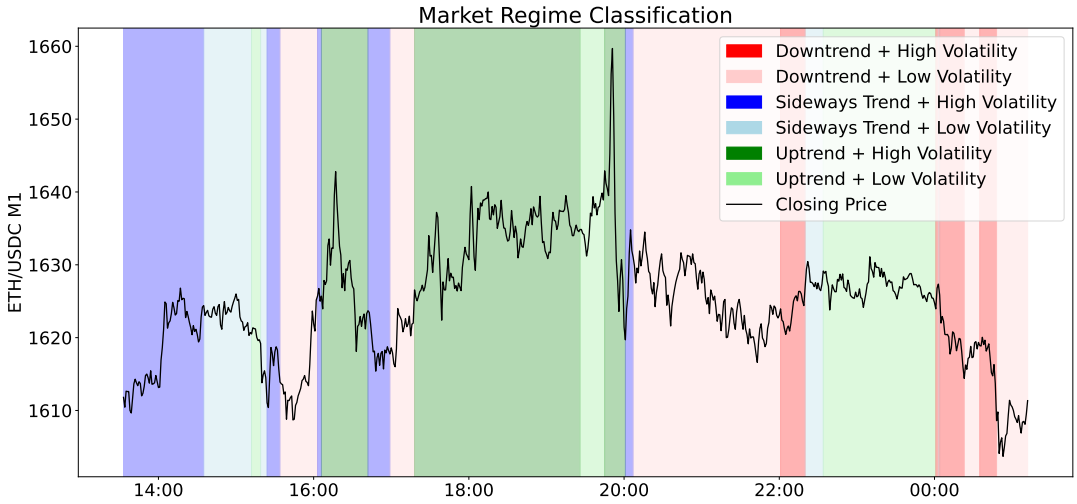

Figure 3: Regime Classification

While classification of the market regimes some discrepancies may occur in individual cases between the algorithmically determined category and the visually perceived category. Such misclassifications are particularly possible during transition phases or in the

case of short-term outliers. For example, the downtrend, highly volatile market regime after 10:00 PM, the algorithm detected a downtrend, but visually, an uptrend is perceived. At this time, the market is in a transition, which causes this misclassification.

Crucially, the classification distinguishes consistently and meaningfully, which is the case when viewed as a whole. The robustness and usefulness of the approach does not arise from absolute freedom from errors, but rather from the systematic reduction of uncertainty compared to a purely visual or subjective assessment.

3.4 Dividing Market Regimes in Durations

Based on the combination of trend direction and volatility level, the duration of each market regime is also taken into account to enable a more differentiated classification. The regime duration is divided into three quantiles (33%, 66%, 100%) based on its distribution, allowing for classification into short-term, medium-term, and long-term phases. The notation in the current work for the respective regime durations is $Q_{0.33}$, $Q_{0.66}$ and Q_1.

The combination of these three dimensions results in a total of 18 different market regimes. This fine-grained segmentation makes it possible to more precisely capture and differentiate different market conditions. For example, a short-term, volatile uptrend from a long-term, calm downtrend.

Even through this makes the market appear more fragmented, and regimes can change more frequently, this finer subdivision is analytically useful. It allows for context-sensitive market behavior to be examined, and differences in the dynamics of effectiveness of trading strategies in specific regime types to be systematically analyzed.

Instead of smoothing out reality with overly broad categories, a more nuanced picture is deliberately drawn that takes into account both the direction, intensity, and stability of market behavior. The resulting increased complexity is not a disadvantage, but rather a prerequisite for more meaningful, context-based analyses.

4 Money- and Risk-Management

Successful trading is not only based on a good strategy, but also on a disciplined management of capital and risk. Even the best prediction is useless if losses grow uncontrolled or a majority of the capital is risked by a few wrong decisions. This is where money and risk management come into play. They define clear rules regarding how much to invest per trade, what level of risk is acceptable, and how losses can be limited. The goal is to protect capital over the long term, minimizing drawdowns, and profit from positive expected values in a controlled manner. This chapter introduces fundamental concepts, metrics, and methods helping to make rational and sustainable decisions.

4.1 Calculating the Position Size

An important part of risk management is calculating the position size. It is helpful to maximize the potential returns, as well as minimizing the financial risk. Many traders are only willing to risk 1% or 2% of the available capital per trade, to prevent a series of losing trades from decimating the available capital too much.

The distance between the estimated entry price and the estimated stop-loss price represents the maximum distance a market can move in an unprofitable direction before a position is automatically closed.

This allows the calculation of the position size to be carried out in three steps:

1. **Determining the risk per trade:** First, it must be determined how much of the available capital should be risked. If 1% of \$10,000 is to be risked, the maximum risk is \$100. It is important to note that the fraction of 1% does not have to be fixed. Thus, it is possible to risk more if the entry signal is very clear. If the entry signal is less clear, it can also be risked less. Only a fixed upper limit should be defined.

2. **Calculating the risk per unit:** To calculate the risk per share the absolute distance between the estimated entry price, and the estimated stop-loss price must be calculated. This represents the risk per unit.

3. **Calculate the position size:** Dividing the risked capital by the risk per unit represents the number of units to buy or sell.

In total, these three steps can be combined into one formula [15]:

$$PositionSize = \frac{AvailableBalance * RiskPerTrade}{RiskPerUnit}$$

So if the available account balance is \$10,000.00, the risk per trade is 1%, and the distance from the estimated entry price to the stop-loss is \$5. The position size is calculated by:

$$PositionSize = \frac{\$10.000 * 1\%}{\$5} = \frac{\$100}{\$5} = 20[Units]$$

It is important to note that numbers can result with many decimal places, and many brokers only allow positions with a certain number of decimal places. If this is the case, the position size must be subsequently rounded to the maximum number of decimal places supported.

4.2 Validating an Entry Signal

Not every entry signal generated by a trading strategy is necessarily profitable. To be profitable in the long term, it is important to ensure that entry signals that are too risky or unrealistic in advance are filtered out, thus preventing positions from being opened. This chapter presents some techniques that can be used to validate entry signals.

4.2.1 Risk-Reward Ratio

The risk-reward ratio (RRR) is a fundamental key figure in trading. It describes the ratio between the potential profit (reward), and the potential loss (risk) of a single trade. The RRR helps in deciding whether an entry signal is too risky or not. It ensures that not only the win rate determines the success of a strategy, but also the ratio of profit to loss in each individual trade.

The RRR is calculated by:

$$RRR = \frac{PossibleProfit}{PossibleLoss} = \frac{|OpenPrice - TakeProfitPrice|}{|OpenPrice - StopLossPrice|}$$

For example, if a long trade is opened at \$100, with a take-profit at \$110, and a stop-loss at \$95, the result is:

$$RRR = \frac{|\$100 - \$110|}{|\$100 - 95\$|} = \frac{\$10}{\$5} = 2$$

This means that for every dollar risked, a potential profit of two dollars is targeted.

A RRR greater 1 is generally considered positive because the expected profit is higher than the potential loss. However, the RRR should not be viewed in isolation. The essential factor is the combination of RRR, and win rate:

1. **High RRR, low win rate:** e.g., $RRR = 3$ with only a 30% probability of winning $\Rightarrow$ potentially profitable.
2. **Low RRR, high win rate:** e.g., $RRR = 0.5$ with an 80% win rate $\Rightarrow$ also potentially profitable.

The following rule of thumb clarifies when a strategy has a positive expected value in the long term:

$$ExpectedValue = PossibleProfit * HitRatio - PossibleLoss * (1 - HitRatio)$$

Only when this expected value is above zero, a trading strategy is statistically profitable.

The RRR is not only a mathematical metric, but a central component of risk management. It helps traders systematically plan how much they are willing to lose per trade, relative to the expected profit. By consistently applying a minimum RRR (e.g., ≥ 1.5), many inefficient setups can be eliminated in advance [16].

4.2.2 Maximum Account Risk

A central aspect of risk management in algorithmic trading systems is limiting overall risk at the portfolio level. In this work, a fixed risk threshold of a maximum of 10% of the total portfolio value has been defined. This threshold serves as an overarching safety limit to prevent excessive risk accumulating through individual positions or simultaneous strategies.

In this context, overall risk refers to the maximum potential loss of all open positions, weighted by their respective stop-loss levels and position size. The 10% limit means that, aggregated across all positions, the potential loss in the event of a simultaneous stop-out may not exceed one-tenth of the current portfolio value.

The risk limitation serves as a protective mechanism against systemic error in the trading system, as even in the case of erroneous trades or sudden market turmoil, the total loss remains within a calculable maximum risk. Therefore, it increases the portfolio resilience against extreme scenarios.

4.2.3 Minimum Take-Profit

In every trading strategy, transaction fees play an important role. Especially in short-term trading, transaction fees can turn seemingly profitable trades negative if they are not adequately considered. A common mistake is setting the take-profit level too narrowly, resulting in a profit that is smaller than the costs incurred. To trade profitably and sustainably, it is therefore essential that the take-profit at least covers the fees incurred, but ideally, significantly higher.

4.3 Dealing with Trading Fees

As shown in Table 1 ByBit charges two different fees named maker-, and taker-fee. The maker fee is charged when a limit order is placed in the order book, thereby creating liquidity. In contrast, the taker fee is charged when a market order is executed. This removes liquidity from the market. It is better for a broker if a market is as liquid as possible. Therefore, maker orders incur lower fees than taker orders.

Especially in higher-frequency trading, it is better to charge as few fees as possible. Therefore, it is better for a trader to execute as many limit orders as possible. Two orders are required for a complete trade: one for entry, and one for exit. But typically, three orders are placed (entry, stop-loss, take-profit), with either the take-profit or stop-loss order being executed.

If a market order is to be executed as an entry order, it is possible to convert it into a limit order by setting the order price slightly below (for long positions) or slightly above the current price (for short positions). The same procedure can be followed for the take-profit order.

However, this procedure should not be used for a stop-loss order. If this is converted to a limit position, there is no longer any guarantee that the stop order will be executed at all, as there is no longer any guarantee that the price will rise above or below the order level. This means that a position may remain open significantly longer than intended. If the price then continues to move in an unprofitable direction, the worst-case scenario is that the position is automatically closed by the broker because there is no longer any capital in the account.

The same problem can theoretically occur with a take-profit order. The difference here is that the set stop market order still defines the maximum risk. Thus, while this particular trade may not be profitable, it is impossible to lose all the capital. It is similar with the opening order. If it is not executed, the entire trade is not going to be executed. This means a missed potential profit, but there is no risk of losing capital.

If all orders are executed as planned, and the price moves in a profitable direction, the conversion of the orders will result in a reduction of fees by a factor of 2.75.

5 Performance Comparison

The performance of trading strategies can be compared in many ways. To reduce complexity, only the three most important metrics are used to compare the result of trading strategies.

Namely, these are [17]:

1. **Equity Curve**
2. **Maximum Drawdown**
3. **Win Ratio**

5.1 Equity Curve

The equity curve is one of the simplest and fastest ways to compare trading strategies. It is a visual or graphical representation of the account equity over the backtested period. A gradually upward sloping equity curve is more preferred than a curve which is very volatile [17].

Figure 4 shows two different equity curves. Even if the more volatile one (orange) is better in the beginning, there are many resets, which causes are worse result compared to the more stable one (blue).

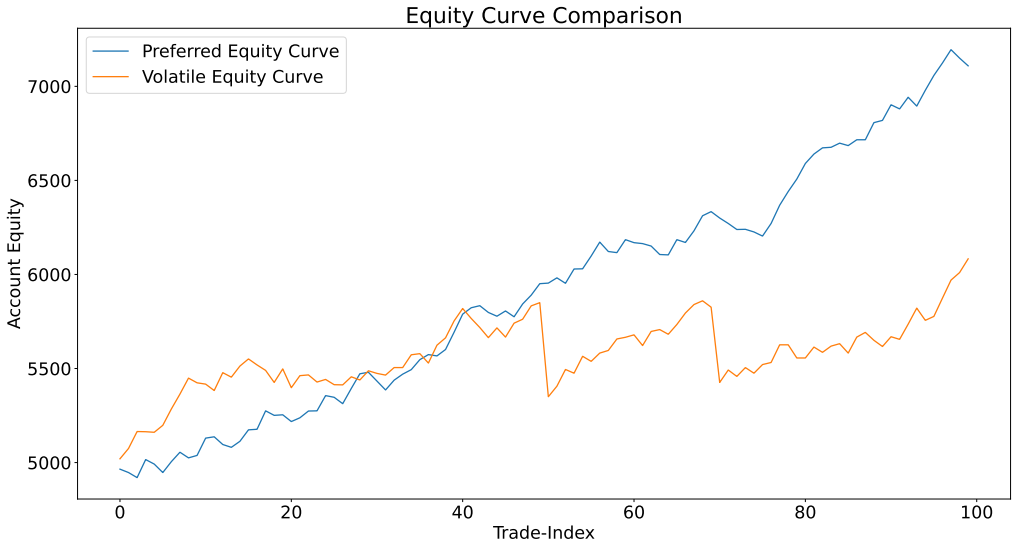

Figure 4: Equity Curve Comparison

5.2 Maximum Drawdown

The maximum drawdown (MDD) measures the largest percentage loss of an investment from its highest point (peak) to its lowest point (trough) within a specific period. It shows the maximum amount of a trading strategy lost before the investment recovers. The MDD is an important risk measure because it highlights the potential losses in times of crisis. A high drawdown indicates high volatility or poor risk management [18].

For each time-point in the equity curve, the drawdown (DD) at time t can be calculated by [19]:

$$DD_t = \frac{Equity_t - PeakValue_{Before}}{PeakValue_{Before}}$$

The maximum drawdown is then the minimum of all drawdowns.

Figure 5 shows the drawdown curves for both equity curves from subsection 5.1. It is noticeable that the maximum drawdown for the more volatile equity curve is much bigger, which also indicates a not optimal trading strategy.

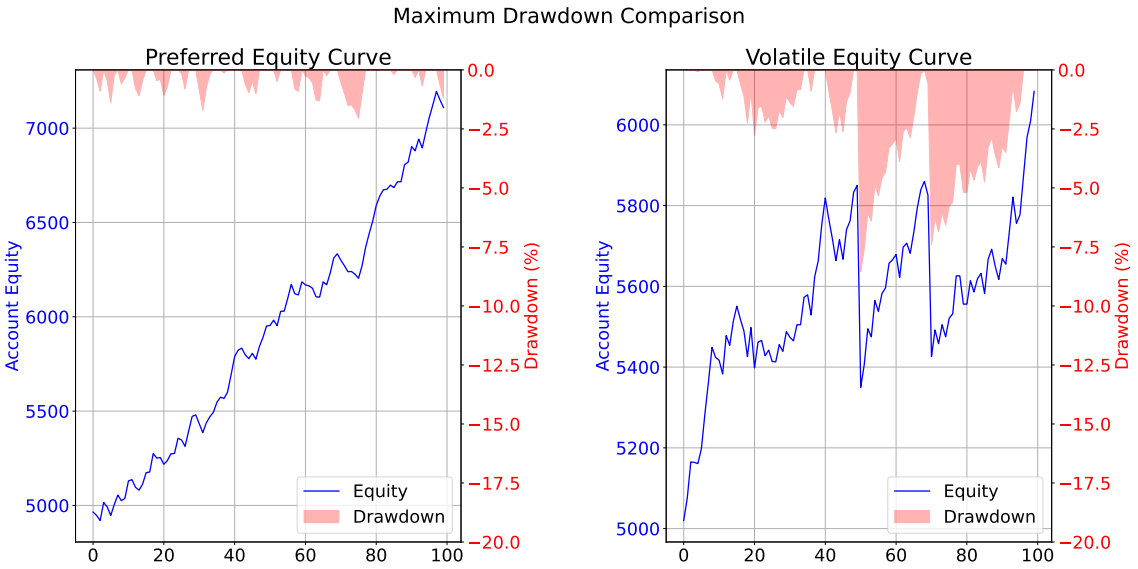

Figure 5: Drawdown Comparison

5.3 Win Rate

Another interesting metric for trading strategy comparison is the win rate (or win ratio). It is the percentage of profitable trades in relation to the total number of trades and is calculated as follows:

$$WinRate = \frac{NumberOfWinningTrades}{TotalNumberOfTrades} * 100$$

For example, if for a total of 20 trades, 16 have been profitable, the win rate is $\frac{16}{20} * 100 = 80\%$.

The greatest limitation of the win rate is that a high win rate does not always indicate a profitable trading strategy [20]. For example, if the win rate is 80% with an average gain of 20$ per profitable trade and an average loss of 90$ per unprofitable trade, the weighted average is $\frac{80*20-(100-80)*90}{100} = -2\$$. Assuming a different strategy with 40% win rate, but with an average gain of 70$ per profitable trade and an average loss of 15$ per unprofitable trade, its weighted average is $\frac{40*70-(100-40)*15}{100} = 19\$$. The second strategy has a much lower win rate, but its long-term outcome is much higher, compared to the first one.

6 Exploratory Data Analysis

Exploratory data analysis uses statistical metrics to gain an understanding of the raw data. In this work, exploratory data analysis also includes data partitioning for training, validation, and testing machine learning models and trading strategies, eliminating data drift, adding additional features, as well as scaling and dimension reduction.

It should be noted that apart from the data splitting, which is a general step, the following steps are only relevant for the machine learning models and not for the classic trading strategies introduced later.

6.1 Dividing the Data

Unlike classic machine learning processes, where data is split in three subsets, named train-, validation-, and test-set, here the data is split in four subsets. The fourth data-set is used for backtesting the real trading strategy, and is therefore not part of the machine learning process but plays an important role in developing the final trading strategy. In contrast, the test-set is only part of the machine learning process and is not used for evaluating or backtesting trading strategies. This distinction was made so that the critical phases (testing machine learning models and testing trading strategies) cannot influence each other.

Set	From	To	No. of Datapoints	% of All Data
Complete	08/05/2022 10:00 UTC+2	06/17/2025 11:30 UTC+2	1,507,598	
Train	08/05/2022 10:00 UTC+2	04/30/2024 23:59 UTC+2	913,684	60.6%
Validation	05/01/2024 00:00 UTC+2	09/30/2024 23:59 UTC+2	220,320	14.6%
Test	10/01/2024 00:00 UTC+2	12/31/2024 23:59 UTC+2	132,482	8.8%
Backtest	01/01/2025 00:00 UTC+2	06/17/2025 11:30 UTC+2	241,111	15.9%

Table 2: Data Split

After the splitting, the four subsets have the following summaries:

	Open	High	Low	Close	Volume
	Open	High	Low	Close	Volume
count	913684.0	913684.0	913684.0	913684.0	913684.0
mean	1933.4	1933.95	1932.85	1933.4	7.58
std	619.6	619.92	619.28	619.6	105.09
min	1074.35	1077.45	1064.05	1074.35	0.0
25%	1578.44	1578.89	1577.99	1578.44	0.0
50%	1801.52	1801.81	1801.13	1801.52	0.05
75%	2083.4	2083.84	2083.05	2083.4	2.02
max	4098.5	4099.48	4096.35	4098.5	31350.65

Table 3: Train Data

	Open	High	Low	Close	Volume
count	220320.0	220320.0	220320.0	220320.0	220320.0
mean	3050.36	3051.3	3049.4	3050.36	2.89
std	473.4	473.45	473.34	473.4	21.45
min	2111.8	2160.4	2088.13	2111.8	0.0
25%	2617.31	2618.04	2616.71	2617.31	0.0
50%	3063.4	3064.31	3062.43	3063.4	0.12
75%	3462.22	3463.29	3461.21	3462.22	1.32
max	3974.68	3976.96	3969.94	3974.68	4972.18

Table 4: Validation Data

	Open	High	Low	Close	Volume
count	132482.0	132482.0	132482.0	132482.0	132482.0
mean	3093.93	3095.27	3092.58	3093.94	4.42
std	531.85	532.29	531.4	531.85	18.42
min	2309.01	2311.73	2307.73	2309.01	0.0
25%	2541.8	2542.65	2540.94	2541.8	0.08
50%	3143.69	3145.38	3142.02	3143.7	0.68
75%	3487.9	3489.46	3486.33	3487.9	2.92
max	4107.28	4112.68	4102.6	4107.28	1341.05

Table 5: Test Data

	Open	High	Low	Close	Volume
count	241111.0	241111.0	241111.0	241111.0	241111.0
mean	2432.57	2433.77	2431.35	2432.57	7.0
std	574.72	574.95	574.49	574.72	32.11
min	1386.6	1395.8	1382.99	1386.6	0.0
25%	1887.22	1888.28	1886.1	1887.22	0.18
50%	2510.29	2511.4	2509.1	2510.29	1.26
75%	2732.0	2733.21	2730.7	2732.0	4.94
max	3742.33	3745.13	3739.65	3742.33	2534.66

Table 6: Backtest Data

6.2 Using Log-Returns

In the summary statistics of the subsets (Table 3, Table 4, Table 5, Table 6) it is noticeable that the mean values change over time. This becomes also clear when visualizing the data (Figure 6).

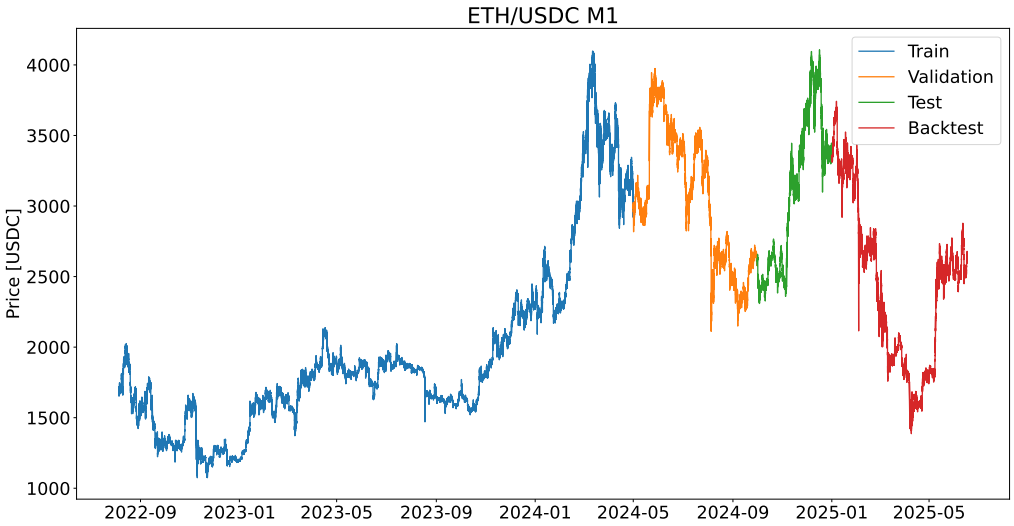

Figure 6: Price Fluctuation of ETH in USDC

To avoid this data drift, the price is transformed to its logarithmic returns (also called log-returns). These are calculated as follows:

$$LogReturn_t = ln(\frac{Price_t}{Price_{t-1}})$$

After the transformation, the means and standard deviations in the subsets are very similar, and the data does no longer drift over time. Table 7 shows the means and standard deviations after the logarithmic return transformation. Additionally, Figure 7 shows visually that the data drift is eliminated.

Data Set	Mean	Standard Deviation
Train	6.5E-7	8.7E-4
Validation	-6.1E-7	9.4E-4
Test	1.8E-6	9.4E-4
Backtest	-9.5E-7	1.1E-3

Table 7: Statistics after Logarithmic Returns Transformation

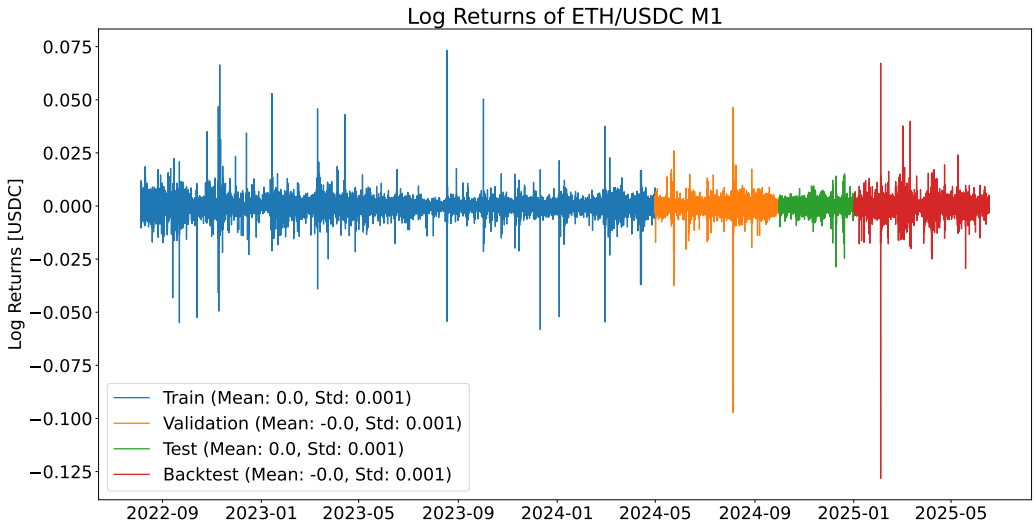

Figure 7: Log Returns of ETH in USDC

If an exact price is to be restored from the logarithmic returns, this can be done with the previous price using the following formula:

$$Price_t = Price_{t-1} * e^{LogReturn_t}$$

If a series of logarithmic returns is present, any price in the series can also be calculated using the following formula, where $Price_0$ is the price at the start of the series and t is the distance to the start of the series:

$$Price_t = Price_0 * \prod_{i=0}^{t} e^{LogReturn_i} = Price_0 * e^{\sum_{i=0}^{t} LogReturn_i}$$

The transformation is applied to the open, high, low, and close prices, and the original prices are replaced by the logarithmic returns, so that all subsequent actions are carried out with the prices on the logarithmic returns.

6.3 Additional Features

To provide the machine learning models with more context about the price, additional features from different categories were added to the raw data.

6.3.1 Trend Following Indicators

In financial analysis trend following indicators play an essential role while modeling a predicting future price movements. This occurs because markets move in trends that are repeatedly interrupted by outliers. This results in a zigzag movement that nevertheless moves in one direction. Trend-following indicators can be used to filter out these outliers [21].

The exponential moving average (EMA) is one type of trend following indicator. It is a variation of the classic simple moving average (SMA), placing more emphasis on newer prices. It is often used by traders in length of 10-, 50-, and 200-period.

An important aspect of distinguishing between EMA and SMA is that many traders assume that new data better reflects the current trend, while others argue that giving more weight to recent prices can lead to distortion [22].

Since the aim of this paper is short-term predictions and the EMA reacts faster to price changes than the SMA, the EMA could provide more relevant context for the model. The EMA was added in 5, 10, 20, 30, 50, and 200 period to the data.

Another trend following indicator is the moving average convergence/divergence (MACD) which does not only help to identify price trends, but also helps to measure the trend momentum. It shows the relationship between two exponential moving averages. To calculate the MACD line, an EMA(12) is subtracted from an EMA(26). Additionally, a signal line is calculated as an EMA(9) of the MACD line.

Although the MACD can signal possible reversals, it is also known for generating a significant number of false positives. This tends to occur particularly in sideways or ranging markets, where price movements lack strong momentum or clear directional trends. In such environments, the MACD lines frequently cross without a meaningful shift in market sentiment, leading traders to act on misleading signals. Consequently, relying solely on MACD in these conditions can result in wrong entries or exits [23]. To mitigate this, many traders combine MACD with additional indicators, such as support and resistance levels or trend confirmation tools, to improve decision-making and reduce noise.

Despite its susceptibility to false signals in sideways markets, the MACD can still be useful as a feature in a machine learning model. This is because the indicator provides valuable information about momentum and potential trend changes, especially during periods of clear price movements. In a machine learning context, the MACD is not considered in isolation but processed in combination with many other features. This allows the model to learn under which market conditions the MACD is more reliable and when it should be ignored. The nonlinear relationships and interactions with other features allow the model to identify context-dependent patterns from the MACD signals, which can be useful for predicting price movements.

6.3.2 Volatility Indicators

To measure volatility, there are other indicators and techniques to measure the volatility, in addition to those described in subsection 3.2.

One indicator is the average true range (ATR) which decomposes the entire range of an asset price for a period. It is calculated by determining the so-called true range (TR) for each candlestick - the maximum of: current high minus low; distance from the previous closing price up, and down. The ATR is then the moving average of these TR values, usually over 14 periods.

The ATR has two main limitations. The first is that an ATR value must always be set into comparison to previous ATR values, because one single value is not enough to tell if a trend is going to reverse. The second limitation is that the ATR does not tell anything about the direction of the price [24]. The ATR was added in 5, 7, 10, 14, and 18 periods to the data.

Another volatility indicator are Bollinger Bands, which consist of three lines. The middle line is a SMA of the closing prices, the lower line is calculated by subtracting a certain number of standard deviations from the middle line, and the upper line is calculated by adding a certain number of standard deviations to the middle line. Usually the double of the standard deviation is added, and subtracted from the middle line.

The higher the volatility of the market is in the last closing prices, the wider the band gets. If the price of the market rises near the upper band, traders see the market as overbought. Similarly, if the market falls near the lower band, the market could be oversold. This allows to generate possible entry and exit signals [25]. The three Lines where added to the data with a 15, 20, and 25 period SMA.

6.3.3 Momentum Indicators

Momentum measures the strength and direction of a price movement over a certain period of time. Momentum indicators are useful because they give insights into the strength of trending prices. Therefore, they can indicate possible reversals in the trend direction [26].

A common momentum indicator is the relative strength index (RSI). It measures the speed and magnitude of an asset price by comparing the average gains and losses of the asset, and can be used to detect overbought and oversold conditions. The RSI ranges between zero and 100. Usually an RSI over 70 indicates an overbought, and an RSI below 30 indicates an oversold market. Commonly the default RSI period to compare the average gains, and losses is 14 [27]. The RSI was added in periods 7, 14, and 20 to the data.

To depict relative trend strength, a sophisticated momentum indicator was constructed that compares the log returns of two different time frames. This feature allows the model to distinguish phases of accelerating price movements from stable or declining trends. The use of logarithmic returns simultaneously achieves scale independence, and improved comparability, which is particularly advantageous for modeling financial market-related time series. This indicator was added for time frames M2, M3, M6, M9, and M12.

6.3.4 Price Transformation Indicators

Apart from the mentioned indicators shifted logarithmic returns for the last six minutes have been added to provide additional context about the last price movements in a compact form. This could help the models to recognize trend reversals, volatility changes or short-term patterns.

Lastly, the logarithmic returns of other time frames (M2, M3, M6, M9, and M12) have been added to the data providing another more stable trend context which helps to correctly classify short-term price movements. This creates a balanced feature set that takes into account both rapid reactions and long-term patterns.

6.4 Further Transformations

Due to the previous steps, the data contains 56 features (price data and additional indicators). Two further transformations are then performed on the features. This includes scaling and dimensionality reduction. It is important to note that before these steps, the data is clustered into the respective regimes using the algorithm described in section 3, and both scaling and dimensionality reduction are performed in isolation for each regime. This brings advantages in both steps.

When scaling, the value range of the scaler can be better utilized. This is evident, for example, with volatility indicators such as the ATR. This indicator generates higher values in volatile phases than in less volatile ones. If the scaler is fitted to the entire data and then a differentiation between market regimes is made, the entire value range of the scaler cannot be utilized in the respective regimes. However, once the differentiation into regimes is made, the entire value range is utilized for each regime.

Dimensionality reduction also offers an advantage. Here, different features may be more dominant than others in different regimes. This allows to examine which features are most important for each regime individually, preventing dilution.

6.4.1 Scaling the Data

Scaling data is an important step in exploratory data analysis. This is because, on the one hand, machine learning models perform better with scaled data [28], and, on the other hand, principal component analysis (subsubsection 6.4.2) is sensitive to unscaled data [29].

The use of the scikit-learn `MinMaxScaler` [30], which scales data in a fixed range of $[0; 1]$ is useful and widely used. Neural networks, especially those with activation functions such as ReLU, which are introduced in subsection 7.2 benefits greatly from inputs with a uniform range of values. The scaling stabilizes the gradient flow, shortens training time and reduces vanishing gradients [31].

For these reasons, all features were scaled with the `MinMaxScaler` and are used exclusively in the scaled form in all steps related to machine learning.

6.4.2 Principal Component Analysis

The principal component analysis (PCA) is a process for dimensional reduction, by linearly transforming high dimensional datasets to a small number of uncorrelated principal components (directions of the new coordinate system). During the transformation, it can be specified how much variance in the data can be eliminated. After a transformation using PCA, it is ensured that at least the specified variance is retained [32].

Especially when processing numerous technical indicators or derived features in financial data, the high dimensionality can become problematic - a phenomenon known as the course of dimensionality. This term describes the increasing challenges in modeling as the number of dimensions or features increases. Data points become increasingly sparsely disturbed, computational costs increase, and many models lose their ability to generalize. Applying PCA allows redundant or correlated information to be condensed, making the model more robust, faster, and easier to interpret. At the same time, the risk of overfitting is reduced because the model focuses on the most important structures in the dataset [33].

Figure 8 shows the cumulative explained variance for the 56 added features in subsection 6.3 for each quantile market regime. It shows that the cumulative variance increases rapidly at the beginning. In this case, the reduction of the project to just 3 to 6 principal components already explains at least 80% of the variance. This means that the majority of the statistically relevant structures in the dataset are retained, even though the number of features has been massively reduced. Even if 20% of the variance is lost, the benefit outweighs this: The remaining principal components capture the statistical essence of the original feature space in a significantly more compact and robust form, which is particularly well-suited for machine learning.

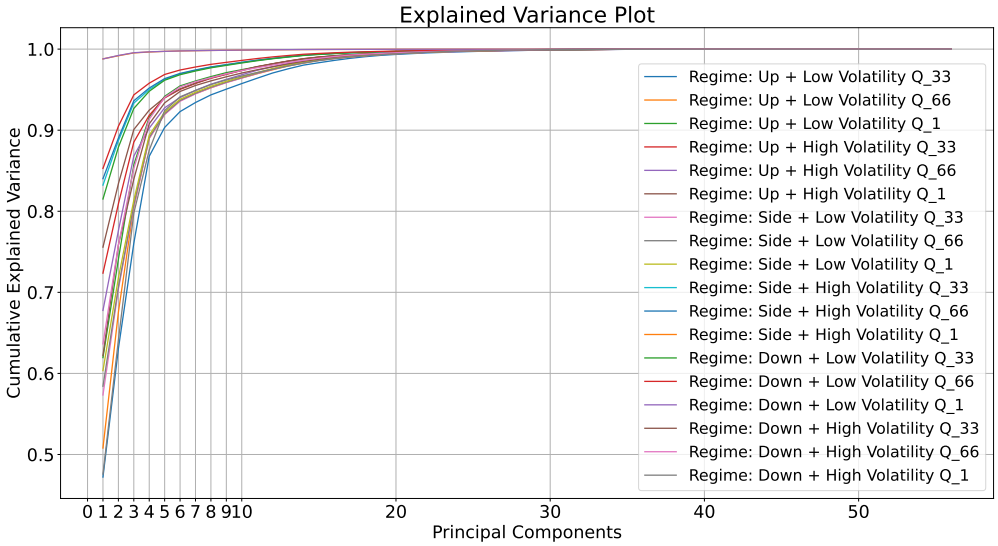

Figure 8: Cumulative Explained Variance

7 Deep Learning Models

This chapter presents the key components for developing and evaluating deep learning models. This includes both the selection of suitable models and the criteria for assessing their performance.

For this work, Keras version 3.10.0 was used as a high-level API for modeling and training neural networks [34]. Instead of the standard TensorFlow backend, PyTorch version 2.7.1 was used as the backend. A key reason for this decision was its better support on Windows, compared to TensorFlow [35], particularly with regard to installation and compatibility with existing CUDA drivers. By using Keras in combination with the PyTorch backend, user-friendly modeling could be combined with stable and well-supported execution on Windows systems.

The model architectures used in this work are not based on specific, citable publications, but were designed as part of an experimental and iterative development process. The goal was to construct powerful models well-suited to the given problem. Particularly with regard to processing sequential data of limited length and high variability.

Established neural building blocks such as 1D convolutional layers, GRUs, and attention mechanisms were used, the fundamentals of which are comprehensively documented in the literature. However, the specific design of the model architectures, such as the combination of multiple parallel CNN paths, the integration of GRUs after convolution steps, or the use of Global Average Pooling instead of flattening, represents a creative and pragmatic composition of its own.

Additionally, all models were automatedly optimized using Optuna, so that architectural decisions were also influenced by the hyperparameter search. In many cases, initial ideas originate from non-scientific sources such as blog posts, online tutorials, or community code snippets, which are not scientifically citable.

In summary, the developed models are original variations, inspired by familiar architectural elements, but not directly adopted or reconstructed from specific publications. This allows for greater flexibility and reinforces the experimental nature of the work.

For both regression and classification models, 13 different architectures were used, which can be divided into neural networks, convolutional neural networks, and long short-term memories.

7.1 Optuna

In modern machine learning methods, the selection of hyperparameters plays a central role in model performance. Hyperparameters such as learning rate, the number of layers in a neural network, or regularization strengths directly influence the behavior and generalization of the model. The search for optimal values for these parameters, the so-called hyperparameter optimization, is often a time-consuming and computationally intensive process [36].

Optuna is a modern framework for automated hyperparameter optimization designed for efficiency, flexibility, and ease of use. It was developed to enable easy integration into existing machine learning pipelines while providing powerful, goal-oriented optimization.

The central abstraction in Optuna is the concept of a study, which defines the optimization task. It encapsulates the objective function and maintains a history of evaluated hyperparameter configurations. Within a study, multiple trials are executed. Each trial

represents a single evaluation of the objective function with a specific set of hyperparameters [37].

The objective function is a user-defined Python function that receives a set of hyperparameters as input and returns a scalar value, which is typically the validation loss or accuracy, that reflects the performance of the model with those parameters. Optuna uses this value to decide how to sample the next set of hyperparameters. Thus, the objective function is the core of the optimization process, as it defines what is being optimized and how success is measured [37].

The goal of Optuna is to automatically find those hyperparameter combinations that satisfy a specific optimization criterion (e.g., minimum validation error rate or maximum accuracy). The search process should be as efficient as possible, requiring as few model training sessions as possible [38].

In this work, Optuna was used to automatically run multiple training models with different hyperparameters to find the best hyperparameters. In the relevant sections of the next chapters, the range of the hyperparameters for each model will be mentioned.

7.2 Neural Networks

Neural networks are among the central methods of machine learning and form the basis of many modern deep learning models. The simplest and most widely used form is the feedforward neural network (FNN), also known as a multilayer perceptron (MLP). Such networks are universally applicable and, with appropriate structuring, can be used for classification, regression, and pattern recognition.

A classic neural network consists of several layers of artificial neurons that forward information in a fixed order from input to output. The structure can be divided as follows [39]:

1. **Input Layer:** Accepts the raw input data, e.g., a vector form of a time series or preprocessed features.
2. **Hidden Layers:** One or more layers of artificial neurons, each of which calculates a weighted sum of the inputs and further processes it using an activation function. Typical activation functions are ReLU, sigmoid, or tanh.
3. **Output Layer:** Provides the final result, e.g., class membership (for classification) or continuous value (for regression).

In this work, three different neural network models have been tested. The tested models are different in their complexity, depth, and methodical approach to data processing.

7.2.1 Base Model with Flatten-Architecture

The first model follows a classic feedforward approach, with one to three dense layers, each with 32 to 128 neurons and a rectified linear unit (ReLU) activation function. The ReLU activation function was used because it provides an efficient calculation and a good gradient propagation with fewer vanishing gradients [40]. Either before or after the dense layers, the input data are flattened [41]. This decision influences whether the model processes all timepoints as a vector early on or treats each time step separately. This architecture was used as a simple baseline model to obtain a quick training and a first baseline model.

The learning rate of the Adam optimizer ranges between 10^{-5} and 10^{-2}, while the input layer has a length of 5 to 150 All hyperparameters are managed by Optuna.

```
num_layers = trial.suggest_int('num_layers', 1, 3)
num_units = trial.suggest_int('num_units', 32, 128)
learning_rate = trial.suggest_float('learning_rate', 1e-5, 1e-2, log=
    True)
input_length = trial.suggest_int('input_length', 5, 150)
flatten_before = trial.suggest_categorical("flatten_before", [True,
    False])

model = Sequential()

model.add(InputLayer(shape=(input_length, input_dimension)))

if flatten_before:
    model.add(Flatten())

model.add(Dense(num_units, activation='relu'))
for _ in range(num_layers - 1):
    model.add(Dense(num_units, activation='relu'))

if not flatten_before:
    model.add(Flatten())

model.add(Dense(...))

model.compile(optimizer=Adam(learning_rate=learning_rate), ...)
```

Listing A: Base FNN

7.2.2 Dropout Neural Network

In the second model, the data is first converted into a one-dimensional vector using a flatten layer. This is followed by one to three dense layers using ReLU activation. Batch normalization supports stable and rapid convergence [42], while dropout between 5% and 35% of the data is used as a regularization method to prevent overfitting [43]. All other parameters are identical to those in the simple neural network.

The parameters tuned via Optuna control the model depth, the number of nodes per layer, the training dynamics, and the number of historical time points used as input.

```
num_layers = trial.suggest_int('num_layers', 1, 3)
num_units_1 = trial.suggest_int('num_units_1', 32, 256)
num_units_2 = trial.suggest_int('num_units_1', 32, 256)
dropout = trial.suggest_float('dropout', 0.05, 0.35)
learning_rate = trial.suggest_float('learning_rate', 1e-5, 1e-2, log=
    True)
input_length = trial.suggest_int('input_length', 5, 150)

model = Sequential()

model.add(InputLayer(shape=(input_length, input_dimension)))

model.add(Flatten())

model.add(Dense(num_units_1, activation='relu'))
model.add(BatchNormalization())
```

```
model.add(Dropout(dropout))
for _ in range(num_layers - 1):
    model.add(Dense(num_units_2, activation='relu'))
model.add(Dropout(dropout))
model.add(Dense(...))

model.compile(optimizer=Adam(learning_rate=learning_rate), ...)
```

Listing B: Dropout NN

7.2.3 Residual Neural Network

The third model integrates a residual concept, originally known from image processing but also increasingly used in time series contexts. The architecture consists of a linear shortcut branch connected to the main path via an additive connection [44]. This construction facilitates the training of deeper networks by improving gradient flow and preventing information loss through multiple layers.

In addition to the previous parameters, two additional variables controlled by Optuna are added: `num_units_res_prep` and `num_units_res`. These determine the complexity of the residual branch. The distinction whether flattening is done before or after the hidden layers is also included here to flexibly adapt data processing to the underlying data structure.

This architecture is particularly suitable for nonlinear and complex relationships, such as those frequently encountered in ETH price forecasting. The residual path allows the model to pass on basic information directly, while deeper layers learn abstract features.

```
num_layers = trial.suggest_int('num_layers', 1, 3)
num_units = trial.suggest_int('num_units', 32, 128)
num_units_res_prep = trial.suggest_int('num_units_res_prep', 32, 128)
num_units_res = trial.suggest_int('num_units_res', 32, 128)
learning_rate = trial.suggest_float('learning_rate', 1e-5, 1e-2, log=
    True)
input_length = trial.suggest_int('input_length', 5, 150)
flatten_before = trial.suggest_categorical("flatten_before", [True,
    False])

input_layer = Input(shape=(input_length, input_dimension))

x = Flatten()(input_layer)

res = Dense(num_units_res_prep, activation="relu")(x)
res = Dense(num_units_res, activation="linear")(res)

x = Dense(num_units_res, activation="relu")(x)

x = Add()([x, res])

x = Dense(num_units, activation="relu")(x)

output = Dense(...)(x)

model = Model(input_layer, output)

model.compile(optimizer=Adam(learning_rate=learning_rate), ...)
```

Listing C: Residual NN

7.3 Long Short-Term Memory Models

In addition to traditional dense neural networks, recurrent neural networks (RNNs) based on the Long Short-Term Memory (LSTM) architecture were also used. LSTMs are specifically designed to capture temporal dependencies in sequential data and retain long-term information across multiple time steps [45]. This is particularly important for financial data such as ETH prices, as current prices are often influenced by past developments.

LSTM cells have an internal memory structure and three control mechanisms (input, forget, and output gates) that allow the network to decide which information should be stored, overwritten, or passed on [46]. This effectively mitigates the so-called vanishing gradient problem of traditional RNNs [47].

7.3.1 Base LSTM

This model consists of one or two stacked LSTM layers followed by a dense output layer. If only one LSTM layer is added, it returns the result directly. With multiple layers, the temporal sequence information is passed to the second LSTM layer.

Similar to feedforward neural networks, the number of layers, the number of neurons per layer, the learning rate, and the length of the input window used are managed via Optuna.

This model represents the basic form of a sequential predictor and is well suited for simple time series patterns.

```
num_layers = trial.suggest_int('num_layers', 1, 2)
num_units_input = trial.suggest_int('num_units_input', 32, 128)
num_units = trial.suggest_int('num_units', 32, 128)
learning_rate = trial.suggest_float('learning_rate', 1e-5, 1e-2, log=
    True)
input_length = trial.suggest_int('input_length', 30, 150)

model = Sequential()
model.add(InputLayer(shape=(input_length, input_dimension)))
if num_layers == 1:
    model.add(LSTM(num_units_input, return_sequences=False))
else:
    model.add(LSTM(num_units_input, return_sequences=True))

for i in range(num_layers - 1):
    if i == num_layers - 2:
        model.add(LSTM(num_units, return_sequences=False))
    else:
        model.add(LSTM(num_units, return_sequences=True))

model.add(Dense(...))

model.compile(optimizer=Adam(learning_rate=learning_rate), ...)
```

Listing D: Base LSTM

7.3.2 Dropout LSTM

This variant extends the standard model with dropout layers inserted between the LSTM units, discarding between 5% and 50% of the data.

The additional parameter is also determined by Optuna and allows fine-tuned control over the regularization strength. This model is particularly useful when working with noisy or volatile price trends, such as cryptocurrencies.

```
num_layers = trial.suggest_int('num_layers', 1, 2)
num_units_input = trial.suggest_int('num_units_input', 32, 128)
num_units = trial.suggest_int('num_units', 32, 128)
learning_rate = trial.suggest_float('learning_rate', 1e-5, 1e-2, log=
    True)
input_length = trial.suggest_int('input_length', 30, 150)
dropout = trial.suggest_float('dropout', 0.05, 0.5)

model = Sequential()
model.add(InputLayer(shape=(input_length, input_dimension)))
if num_layers == 1:
    model.add(LSTM(num_units_input, return_sequences=False))
else:
    model.add(LSTM(num_units_input, return_sequences=True))

model.add(Dropout(dropout))
for i in range(num_layers - 1):
    if i == num_layers - 2:
        model.add(LSTM(num_units, return_sequences=False))
    else:
        model.add(LSTM(num_units, return_sequences=True))
    model.add(Dropout(dropout))

model.add(Dense(...))

model.compile(optimizer=Adam(learning_rate=learning_rate), ...)
```

Listing E: Dropout LSTM

7.3.3 Bidirectional LSTM (BiLSTM)

The third model uses bidirectional LSTM layers. These process the input sequence forward and backward simultaneously, allowing both earlier and later information to be incorporated into the internal state [48].

This architecture can be particularly valuable in the context of ETH predictions based on historical data, as it allows for better detection of symmetric patterns or turning points in price trends.

The hyperparameter structure is similar to the previous models, but the modeling is performed using double LSTM paths.

```
num_layers = trial.suggest_int('num_layers', 1, 2)
num_units_input = trial.suggest_int('num_units_input', 32, 128)
num_units = trial.suggest_int('num_units', 32, 128)
learning_rate = trial.suggest_float('learning_rate', 1e-5, 1e-2, log=
    True)
input_length = trial.suggest_int('input_length', 30, 150)

model = Sequential()
```

```
model.add(InputLayer(shape=(input_length, input_dimension)))

if num_layers == 1:
    model.add(Bidirectional(LSTM(num_units_input, return_sequences=
    False)))
else:
    model.add(Bidirectional(LSTM(num_units_input, return_sequences=True
    )))

for i in range(num_layers - 1):
    if i == num_layers - 2:
        model.add(Bidirectional(LSTM(num_units, return_sequences=False)
    ))
    else:
        model.add(Bidirectional(LSTM(num_units, return_sequences=True))
    )

model.add(Dense(...))

model.compile(optimizer=Adam(learning_rate=learning_rate), ...)
```

Listing F: Bidirectional LSTM

7.3.4 Encode-Decode Model with Repeat-Vector

The fourth model is based on an encoder-decoder approach, as used in machine translation. The input sequence is first converted into a fixed context vector by an LSTM layer, which is then duplicated multiple times using RepeatVector [49]. A decoder-LSTM interprets this vector sequentially to generate a temporally structured output.

This architecture is particularly suitable for multistep forecasting, as it can map not only the next value but an entire sequence of future values. The final preprocessing is performed using `TimeDistributed(Dense(...))` to calculate a separate output for each time step [50].

```
num_units_input = trial.suggest_int('num_units_input', 32, 128)
num_units = trial.suggest_int('num_units', 32, 128)
learning_rate = trial.suggest_float('learning_rate', 1e-5, 1e-2, log=
    True)
input_length = trial.suggest_int('input_length', 30, 150)

model = Sequential()
model.add(InputLayer(shape=(input_length, input_dimension)))

model.add(LSTM(num_units_input, return_sequences=False))

model.add(RepeatVector(30))

model.add(LSTM(num_units, activation="relu", return_sequences=True))
model.add(TimeDistributed(Dense(...)))

model.compile(optimizer=Adam(learning_rate=learning_rate), ...)
```

Listing G: Encode-Decode LSTM

7.4 Convolutional Neural Networks

Convolutional neural networks (CNNs) represent a special architecture within deep learning that was developed specifically to process data with spatial structures, such as images. Compared to traditional feedforward networks, CNNs are significantly more efficient in handling high-dimensional inputs and have established themselves as standard methods in numerous application areas such as object detection, natural language processing, and time series analysis [51].

The key difference between CNNs and classical neural networks lies in the use of so-called convolutional layers, which exploit the local structures in the input data. Instead of connecting every neuron to all input values (as with a fully connected layer), local filters (kernels) are used that respond only to a small portion of the input space. These filters are automatically learned during the training process [51].

Time series typically exhibit temporal dependencies, repetitions, seasonal patterns, or short-term fluctuations. Classic methods for processing this data, such as LSTM networks, explicitly model such dependencies using recursive states. CNNs, on the other hand, uses an alternative approach, called local convolutions, which also serve time series to detect relevant patterns or changes over short or medium periods of time [52].

The advantages of using CNNs for time series include:

1. **Parallel Processing:** CNN does not require recursive loops, e.g., compared to LSTMs.

2. **Local Pattern Detection:** CNNs can detect local patterns, such as peaks, fluctuations, or trend changes.

3. **Computational Complexity:** CNNs requires lower computational power compared to recursive architectures.

7.4.1 Base CNN

The simplest variant consists of two stacked one-dimensional convolutional layers, followed by maxpooling, and two dense layers for output preparation. The position of the flattening layer affects the processing order between the dense and the sequence structure.

The hyperparameters tuned by Optuna are the number of units in the convolutional layers, the length of the filters (Kernel size), the size of the maxpooling region, the position of the flattening layer, and the number of neurons in the dense layers.

```
num_units_cnn = trial.suggest_int('num_units_cnn', 32, 128)
num_units = trial.suggest_int('num_units', 32, 128)
learning_rate = trial.suggest_float('learning_rate', 1e-5, 1e-2, log=
    True)
input_length = trial.suggest_int('input_length', 5, 150)
flatten_before = trial.suggest_categorical("flatten_before", [True,
    False])
kernel_size = trial.suggest_int("kernel_size", 2, 5)
pool_size = trial.suggest_int("pool_size", 1, 5)

model = Sequential()
model.add(InputLayer(shape=(input_length, input_dimension)))
model.add(Conv1D(num_units_cnn, kernel_size=kernel_size, activation="
    relu"))
model.add(MaxPooling1D(pool_size=pool_size))
```

```
if flatten_before:
    model.add(Flatten())
model.add(Dense(num_units, activation="relu"))
if not flatten_before:
    model.add(Flatten())
model.add(Dense(...))

model.compile(optimizer=Adam(learning_rate=learning_rate), ...)
```

Listing H: Base CNN

7.4.2 Deep CNN

The model shown here represents an extension of a simple CNN model (Base CNN). While the simple model uses only a single 1D convolutional layer followed by a pooling and dense layer, this Deep CNN model uses two consecutive Conv1D layers. This additional depth allows the network to learn more complex and hierarchical feature representations from the input data [53]. This is particularly useful for timeseries with multi-level dependencies or subtle patterns. The model is complemented by a MaxPooling layer, which reduces the temporal resolution and counteracts overfitting. The optional placement of the Flatten layer allows for the evaluation of different configurations of feature identification, which in turn increases model flexibility. Overall, this more deeply structured CNN enables more powerful modeling than the flattened variant.

```
num_units_cnn = trial.suggest_int('num_units_cnn', 32, 128)
num_units = trial.suggest_int('num_units', 32, 128)
learning_rate = trial.suggest_float('learning_rate', 1e-5, 1e-2, log=
    True)
input_length = trial.suggest_int('input_length', 5, 150)
flatten_before = trial.suggest_categorical("flatten_before", [True,
    False])
kernel_size = trial.suggest_int("kernel_size", 2, 5)
pool_size = trial.suggest_int("pool_size", 1, 5)

model = Sequential()
model.add(InputLayer(shape=(input_length, input_dimension)))
model.add(Conv1D(num_units_cnn, kernel_size=kernel_size, activation="
    relu", padding="same"))
model.add(Conv1D(num_units_cnn, kernel_size=kernel_size, activation="
    relu", padding="same"))
model.add(MaxPooling1D(pool_size=pool_size))
if flatten_before:
    model.add(Flatten())
model.add(Dense(num_units, activation="relu"))
if not flatten_before:
    model.add(Flatten())
model.add(Dense(...))

model.compile(optimizer=Adam(learning_rate=learning_rate), ...)
```

Listing I: Deep CNN

7.4.3 Attention CNN

An alternative architecture uses multiple parallel convolutional layers with different kernel sizes. This allows the model to detect both short-term and medium-term patterns simultaneously. The resulting feature maps are combined using an attention layer to highlight important sequence segments [54].

This is followed by global average pooling and a dense layer for output generation.

```
num_units_cnn = trial.suggest_int('num_units_cnn', 32, 128)
num_units = trial.suggest_int('num_units', 32, 128)
learning_rate = trial.suggest_float('learning_rate', 1e-5, 1e-2, log=
    True)
input_length = trial.suggest_int('input_length', 5, 150)
kernel_size_1 = trial.suggest_int("kernel_size_1", 2, 9)
kernel_size_2 = trial.suggest_int("kernel_size_2", 2, 9)
kernel_size_3 = trial.suggest_int("kernel_size_3", 2, 9)

input_layer = Input(shape=(input_length, input_dimension))

conv_1 = Conv1D(num_units_cnn, kernel_size=kernel_size_1, padding="same
    ", activation="relu")(input_layer)
conv_2 = Conv1D(num_units_cnn, kernel_size=kernel_size_2, padding="same
    ", activation="relu")(input_layer)
conv_3 = Conv1D(num_units_cnn, kernel_size=kernel_size_3, padding="same
    ", activation="relu")(input_layer)

concat = Attention()([conv_1, conv_2, conv_3])

gap = GlobalAveragePooling1D()(concat)
dense1 = Dense(num_units, activation="relu")(gap)

output_layer = Dense(...)(dense1)

model = Model(input_layer, output_layer)

model.compile(optimizer=Adam(learning_rate=learning_rate), ...)
```

Listing J: Attention CNN

7.4.4 Concatenation CNN

A slightly modified version replaces the attention module with a simple concatenation of the feature maps. This allows the model to pass the extracted features directly without performing weighting [55].

This variant is computationally less expensive than the attention-based one and is well-suited when all filter outputs should be treated equally.

```
num_units_cnn = trial.suggest_int('num_units_cnn', 32, 128)
num_units = trial.suggest_int('num_units', 32, 128)
learning_rate = trial.suggest_float('learning_rate', 1e-5, 1e-2, log=
    True)
input_length = trial.suggest_int('input_length', 5, 150)
kernel_size_1 = trial.suggest_int("kernel_size_1", 2, 9)
kernel_size_2 = trial.suggest_int("kernel_size_2", 2, 9)
kernel_size_3 = trial.suggest_int("kernel_size_3", 2, 9)
pool_size = trial.suggest_int("pool_size", 1, 5)

```

```
input_layer = Input(shape=(input_length, input_dimension))
conv_1 = Conv1D(num_units_cnn, kernel_size=kernel_size_1, padding="same", activation="relu")(input_layer)
conv_2 = Conv1D(num_units_cnn, kernel_size=kernel_size_2, padding="same", activation="relu")(input_layer)
conv_3 = Conv1D(num_units_cnn, kernel_size=kernel_size_3, padding="same", activation="relu")(input_layer)

concat = Concatenate()([conv_1, conv_2, conv_3])

gap = GlobalAveragePooling1D()(concat)
dense1 = Dense(num_units, activation="relu")(gap)

output_layer = Dense(...)(dense1)

model = Model(input_layer, output_layer)

model.compile(optimizer=Adam(learning_rate=learning_rate), ...)
```

Listing K: Concatenation CNN

7.4.5 CNN-GRU Hybrid Model

The Gated Recurrent Unit (GRU) model is a simplified variant of the LSTM and was developed to reduce computational effort and model complexity. Compared to LSTM, GRU has only two gates, an update gate and a reset gate, instead of three (input, forget, and output gates in LSTM). This makes GRU faster to train, requires less memory, and delivers comparable results in many tasks [56].

To combine the advantages of both architectures, a hybrid CNN-GRU model was also implemented. A Conv1D layer first extracts local features from the sequence, which are then sequentially processed using a GRU module to capture long-term dependencies and trends [57].

This model is particularly well-suited for linking local patterns (using CNN) with temporal dependencies (using GRU). The final dense layers perform the transformation to the target variable as usual.

```
num_units_cnn = trial.suggest_int('num_units_cnn', 32, 128)
num_units = trial.suggest_int('num_units', 32, 128)
num_units_gru = trial.suggest_int('num_units_gru', 32, 128)
learning_rate = trial.suggest_float('learning_rate', 1e-5, 1e-2, log=True)
input_length = trial.suggest_int('input_length', 5, 150)
kernel_size = trial.suggest_int("kernel_size", 2, 5)
pool_size = trial.suggest_int("pool_size", 1, 5)

model = Sequential()
model.add(InputLayer(shape=(input_length, input_dimension)))
model.add(Conv1D(num_units_cnn, kernel_size=kernel_size, activation="relu"))
model.add(MaxPooling1D(pool_size=pool_size))
model.add(GRU(num_units_gru, return_sequences=False))
model.add(Dense(num_units, activation="relu"))
model.add(Dense(...))

model.compile(optimizer=Adam(learning_rate=learning_rate), ...)
```

Listing L: CNN + GRU

7.5 Model Training

In the following, all models mentioned in subsection 7.2, subsection 7.3, and subsection 7.4 are trained on the training data (Table 3) for 20 Optuna trials, each with 30 epochs. After each epoch, the models are evaluated on the validation data (Table 4).

As already mentioned, the model architectures mentioned are used for regression and classification. For this purpose, a different final dense layer is used for each architecture, which is described in the respective chapters.

Similar to scaling and dimensionality reduction, the models are trained individually for each regime. This results in 10,800 [3] trained models per architecture. After each epoch, the evaluation metrics and the model are saved.

Another aspect that is determined by the regimes is how the input-output sequences can be divided. There are four possibilities here:

1. **Complete sequence within a regime:** In this case, the entire sequence, both the input and output data, would be within a single regime. This has the advantage that each model is truly assigned to only one regime and operates exclusively with data within that regime. However, this approach is impractical in live operation, as it is unknown whether the market will remain in this regime in the future in which the prediction is made. This would constitute data snooping, and the models would be useless.

2. **Only the output data within a regime:** The same problem arises here as in point 1. In live operation, it is unknown whether the market will remain in this regime in the future. Thus, data snooping occurs again, and the models become unusable.

3. **Only the input data within a regime:** It must be ensured that all input data lies within a regime. The number of regimes in the output data is irrelevant. This ensures that the model always receives data that lies within a regime as input, but no data snooping occurs.

4. **The last N percent of the input data must be within a regime:** This has the advantage of eliminating data snooping and providing more test data compared to point 3, since more input sequences meet the requirements. However, it requires additional effort because the percentage N must be determined. This would require either heuristics or a complex process to determine N. Furthermore, compared to point 3, the model cannot guarantee that 100% of the input data is in the same regime and would therefore have to learn more patterns in the data to cope with regime changes within the input data.

For the reasons listed above, it was decided in this work to divide the data as described in point 3, since the distinction as to which model the data belongs is easy to implement, data snooping is excluded and no additional heuristics or processes have to be carried out to determine what percentage of the input data must belong to the respective regime.

[3] $18 Regimes * 20 Trials * 30 Epochs$

7.6 Regression Models

Regression models were used to predict the next 30 minutes of logarithmic returns as a continuous sequence. To achieve a continuous sequence as output, the activation function in the last dense layer of the above-mentioned models was set to `linear` and the number of neurons to 30.

7.6.1 Loss-Function

In order to incorporate the trading context during model training and evaluation, traditional loss functions such as the root mean squared error have been avoided. Instead, a proprietary loss function was implemented that calculates realized profit and loss. The value of the loss function decreases the more profit is generated.

As mentioned above, the output of the regression models are either a sequence of the next expected logarithmic returns by minute. The first step of the loss function is to decide whether the opened position should be a long or short position. This is decided by cumulating the predictions. The absolute maximum is then determined for the cumulative values greater than and less than zero. If the absolute maximum of the values greater than zero is greater than the absolute maximum of the values less than zero, a long position is opened. Otherwise, a short position is opened. Figure 9 illustrates the decision using a long position as an example.

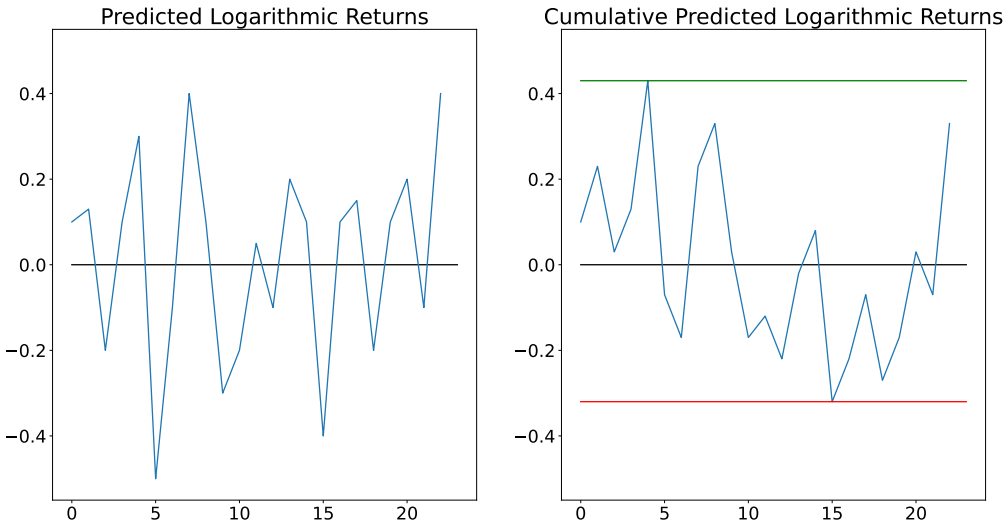

Figure 9: Long Position Decision

After the direction decision, the stop-loss and take-profit levels must be determined. For long and short positions, there are two possibilities at which specific level the stop-loss and take-profit are set, based on the global high and low of the cumulative predicted logarithmic returns. If a long position was previously decided and the global high comes after the global low, the stop-loss is set at the global low and the take-profit at the global high. However, if the global high comes before the global low, the stop-loss is set at the lowest low before the global high. For short positions, the decision is exactly the opposite.

Figure 10 shows the stop-loss and take-profit levels for all four cases. The red areas mark the predicted unprofitable zones and the green areas mark the predicted profitable zones.

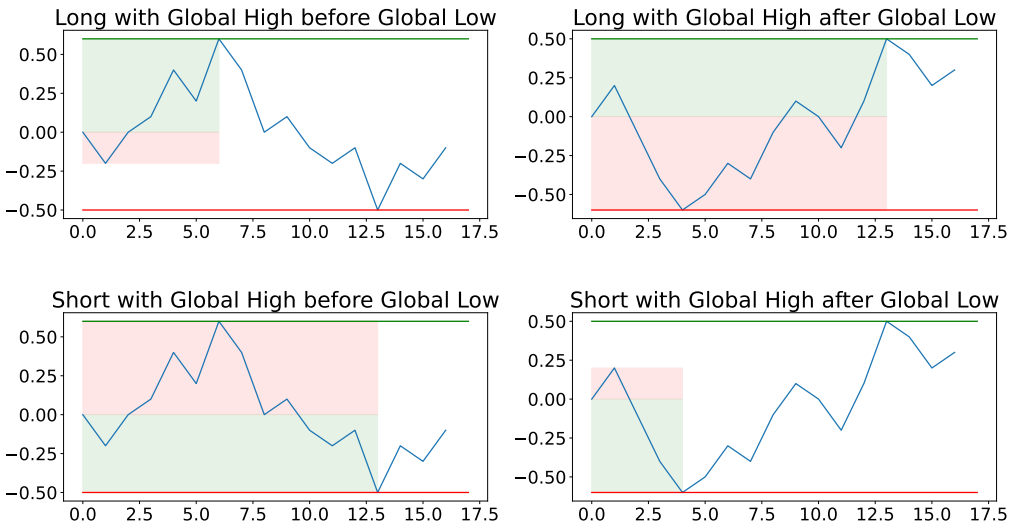

Figure 10: Long Position Decision

After the stop-loss and take-profit levels are determined, the actual logarithmic returns are used to calculate the profit or loss for each position. If neither the stop-loss nor the take-profit is reached, the profit calculation will by default return the loss when the stop-loss is reached to ensure that the model predicts the correct take-profit more often and is penalized if neither the take-profit nor the stop-loss is reached.

7.6.2 Metrics

Additionally, additional metrics in the trading context were added to the model during training. These include:

1. **Profit Hit-Rate:** Percentage of trades that would have been profitable based on the prediction.
2. **Loss Hit-Rate:** Percentage of trades that would have been unprofitable based on the prediction.
3. **None Hit-Rate:** Percentage of trades that would have reached neither the stop loss nor the take profit based on the prediction.

When the three parts are added together, the result must be 1, since one of the events must occur in each prediction.

7.6.3 Model Evaluation

Despite the implementation and training of various deep learning models for accurate price prediction, combined with various technical indicators and a specially developed loss function for directly simulating trading decisions, none of the models achieved satisfactory predictive performance. The presented loss hit-rate metric was always exactly 1 for every model and in every market regime on both the validation and test data, meaning that every trade decision made based on the prediction was incorrect and closed with a loss.

The inadequate suitability of the models is also evident in an exemplary prediction. In one specific case, future prices were reconstructed based on the predicted logarithmic returns and the price known at the time. The calculation was carried out using the formula presented in subsection 6.2.

In reality, the ETH/USDC price was around USDC 1,800 at the given time. However, the model predicted such a negative return that the resulting price fell to almost USDC 0 within just 30 minutes. Such behavior is not only extremely unlikely under realistic market conditions, but economically virtually impossible, as it would be tantamount to a complete market collapse.

This incorrect forecast is not an isolated case, but rather symptomatic of the weaknesses of the regression models tested. It shows that the models are not only inaccurate but also tend to sometimes produce unrealistic extreme values, which would entail considerable risks in a trading context.

This chapter is dedicated to a critical analysis of possible causes.

A key reason for the poor model performance lies in the low predictability of short-term price movements. Minute-by-minute logarithmic returns are subject to a strong stochastic nature. They resemble a random walk and rarely exhibit robust, persistently usable patterns [58]. Thus, the models lack the structural foundation to make reliable predictions. However, many machine learning methods implicitly assume the existence of certain stationary or repeatable relationships, a circumstance that is severely limited with M1 returns.

A central methodological element of this work was the introduction of a modified loss function based on the simulation of trades. This function was explicitly designed to consider not only the statistical error but also the practical utility of a model within a trading strategy.

In practice, however, it became apparent that the model often "learned" to remain as inactive as possible to minimize losses, or developed strategies with high trading frequency in the presence of negative performance because it was unable to identify stable patterns of advantage. This behavior is due to a lack of positive feedback in the learning process: If no or only random winning patterns are present in the data, even the best loss function cannot enable meaningful optimization.

The poor model performance should not be interpreted as a failure of the method or the data selection, but rather as a relevant empirical result. It shows that applying deep learning to high-frequency cryptocurrency data does not automatically lead to profitable predictive models, even when modern model architectures, additional features, and specialized loss functions are used.

For the reasons mentioned above, the regression models will not be pursued further in the course of this work.

7.7 Classification-Models

Despite the poor results of the regression models, classification models were trained that predict an action to be performed. The action can be either buy, sell, or do nothing. For that, the model predicts a probability for each of the three classes. To achieve a direct comparison with the regression models, the same models as in subsection 7.6 were used. However, with the difference that the last dense layer has only three neurons, each representing an action, and the activation function `softmax` was used. The identical training and validation period was also used, and the training took place over 20 Optuna trials with 30 epochs each.

7.7.1 Loss-Function

For classification models, the categorical cross-entropy loss function was used. This choice is common and well-suited for classification problems where the model output represents a probability distribution across multiple classes. Mathematically, it is defined as follows:

$$L_{CCE} = -\sum_{i=1}^{C} y_i * log(\hat{y}_i)$$

Where C denotes the number of classes, y_i the actual value (usually encoded as a one-hot vector), and $\hat{y}_i$ the probability predicted by the model for class i. This function penalizes the model particularly severely if it assigns a high probability to an incorrect class and rewards correct predictions with high confidence.

Categorical cross-entropy measures the difference between the actual target distribution and the probability distribution predicted by the model [40].

7.7.2 Model Evaluation

Table 8 shows the best performing classification models on the validation data, as well as their associated loss and accuracy on the validation and test data for each market regime.

Market Regime			Model	Loss		Accuracy	
				Validation	Test	Validation	Test
Down	Low	$Q_{0.33}$	Concatenation CNN	0.69	0.70	0.53	0.47
		$Q_{0.66}$	Concatenation CNN	0.70	0.70	0.55	0.52
		Q_1	Attention CNN	0.89	0.86	0.58	0.43
	High	$Q_{0.33}$	Attention CNN	0.78	0.89	0.53	0.51
		$Q_{0.66}$	Attention CNN	0.84	0.68	0.54	0.48
		Q_1	LSTM	0.81	0.79	0.55	0.49
Sideways	Low	$Q_{0.33}$	Attention CNN	0.75	0.63	0.52	0.49
		$Q_{0.66}$	Concatenation CNN	0.72	0.67	0.52	0.51
		Q_1	Attention CNN	0.95	0.66	0.55	0.50
	High	$Q_{0.33}$	Attention CNN	0.73	0.69	0.54	0.49
		$Q_{0.66}$	Attention CNN	0.99	0.76	0.58	0.51
		Q_1	Concatenation CNN	0.66	0.71	0.54	0.52
Up	Low	$Q_{0.33}$	Concatenation CNN	0.69	0.69	0.54	0.54
		$Q_{0.66}$	Concatenation CNN	0.69	0.69	0.54	0.51
		Q_1	Base CNN	0.75	0.65	0.53	0.49
	High	$Q_{0.33}$	Concatenation CNN	0.70	0.70	0.52	0.48
		$Q_{0.66}$	Attention CNN	0.81	0.95	0.53	0.48
		Q_1	BiLSTM	0.69	0.69	0.56	0.51

Table 8: Validation and Test Metrics for Classification Models

It is striking that some models achieve very good results on the validation data. On the test data, the model performance of all models is worse than on the validation data, but the metrics still show that a profitable trading strategy is possible.

As explained in subsection 5.3, a high win rate is not the only criterion that determines whether a trading strategy is successful.

It is also striking that identical model architectures perform significantly better in classification than in regression. This once again highlights that not only the model architecture, but also other factors, such as the loss function or the task, determine whether a model can learn from the training data.

8 Trading Strategies

In the context of automated trading, systematic trading strategies play a central role. They enable decisions to be made based on clearly defined rules or mathematical models, rather than on subjective assessments or human intuition.

A trading strategy defines a methodical approach by which financial instruments are bought or sold to achieve a specific goal. Typically, this is to maximize profit while limiting risk. Such strategies are often based on indicators from technical analysis or fundamental market information [59].

Essential components of a strategy are signal generation (entry/exit criteria), risk management (e.g., stop-loss, position sizes), and performance evaluation based on metrics such as cumulative profit, volatility, or maximum drawdown. The development process of a successful strategy typically comprises several phases, ranging from idea generation and backtesting to optimization and validation on previously unseen market data [60].

Similar to deep learning hyperparameters, trading strategies have also parameters, which can be defined before executing. In this work, these parameters are defined in a range, consisting of a minimal value, a maximum value, and a step size.

This chapter provides an overview of the trading strategies compared in this paper. Both the models trained in section 7 and classic trading strategies are presented. An overview of the metrics used to compare the trading strategies is also provided. The presented strategies also have their parameters defined, which will be tested later. It should be noted that each strategy also contains a parameter that defines whether positions are opened with a fixed stop level or a trailing stop. Since this parameter is included in all strategies, it is not listed every time.

8.1 AI Trading Strategies

AI trading strategies treat the model outputs differently. Here, too, a distinction is made between regression and classification models.

8.1.1 Regression AI Strategy

As described in subsubsection 7.6.3, the trained regression models are not suitable for developing a trading strategy. Nevertheless, this chapter describes a trading strategy that could be implemented using a predicted series of logarithmic returns. However, it is not evaluated or tested.

From the predicted series of logarithmic returns, the expected price in t minutes can be calculated by the formula mentioned in subsection 6.2.

Based on the predicted price, the stop-loss and take-profit can be determined identically to the loss-function of the regression models (subsubsection 7.6.1). This creates an entry signal with fixed stop-loss and take-profit level.

However, since it is possible that the price could exceed the stop level or the take-profit level might not be reached, additional parameters are introduced into the strategy that shift the two levels by a specific number of points. A third parameter is also introduced, which specifies a fixed distance to the stop-loss if no stop-loss has been predicted. This can happen, for example, if the initial prediction is positive and the price in the prediction does not fall below the current price.

Parameter Name	Min Value	Max Value	Step Size
Take-Profit Delta	-20.0	20.0	2.0
Stop-Loss Delta	-20.0	20.0	2.0
Stop-Loss not Predicted Delta	1.0	20.0	20.0

Table 9: AI Regression Model Strategy Parameters

8.1.2 Classification AI Strategy

As described in subsection 7.7 the classification models predict a probability for an action, which can be either buy, sell or do nothing. This strategy executes a buy or sell action if the predicted probability is greater than a predefined minimum probability. If the buy or sell probability is less than the minimum probability or the do nothing probability is the greatest, the strategy does nothing.

The stop-loss and take-profit levels are also predefined parameters.

Parameter Name	Min Value	Max Value	Step Size
Take-Profit Distance	5.0	100.0	5.0
Stop-Loss Distance	5.0	100.0	5.0
Min. Probability for Entry	0.3	0.9	0.1

Table 10: AI Classification Model Strategy Parameters

8.2 Classic Trading Strategies

Technical analysis (TA) is based on the assumption that market movements do not develop randomly, but follow certain patterns that have repeated themselves in the past. In contrast to fundamental analysis, which deals with the intrinsic value of a financial instrument, technical analysis focuses exclusively on past price and volume movements to draw conclusions about future price developments.

Technical analysis focuses on visual and computational methods for identifying trend, support and resistence levels, and reversal points. These analysis are used to develop specific trading strategies that specifically respond to specific market behaviours. These strategies are often rule-based and can be implemented both manually and algorithmically [61]. Therefore, technical analysis strategies are more likely suitable for algorithmic trading.

This chapter describes three widely used technical analysis trading strategies. The strategies introduced are not the only classic strategies tested. They are the three that performed best in the tests. The other tested strategies are also briefly listed in Chapter X. However, some of them have been slightly modified in this work compared to the most widely used ones.

8.2.1 Dual Simple Moving Average Strategy

A common trading strategy is the simple moving average crossover strategy. It consists of two SMA with different periods. If the short-term SMA crosses the long-term SMA above, a long position is opened. Otherwise, if the short-term SMA crosses the long-term SMA below, a short position is opened [62]. Figure 11 shows exemplary two moving averages for a synthetic price, with blue arrows marking buy entry signals and red arrows marking sell entry signals.

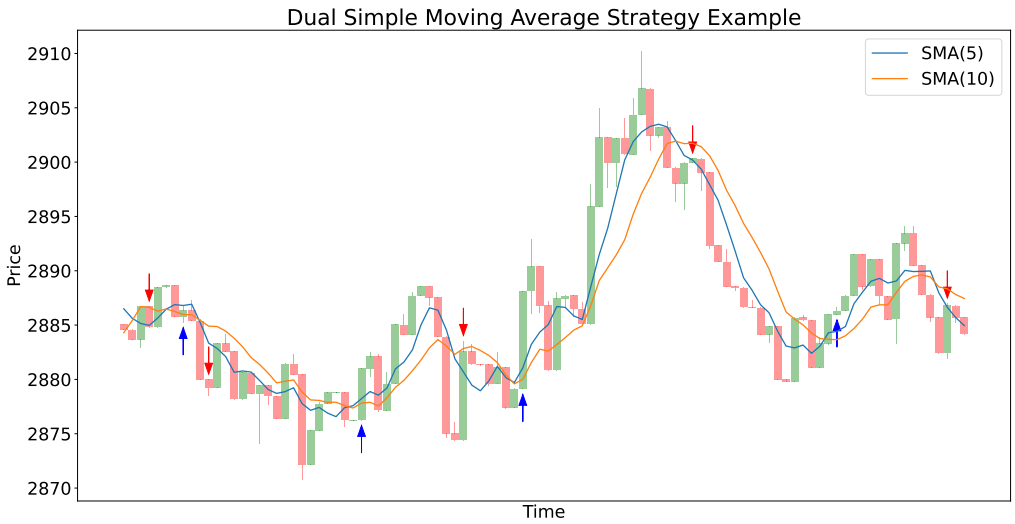

Figure 11: Dual Simple Moving Average Strategy Example

This strategy is very easy to understand and implement and can help to identify possible trend changes which generate entry and exit signals. One of the greatest disadvantages is that the SMA is a lagging indicator and in quickly changing market conditions, the signals can be delayed. Especially in sideways market regimes, the SMA's can cross often, which can lead to false signals [63]. Here, the market regime recognition plays an important role. Through this, the strategy can be disabled in sideways regimes and enabled if the market has a clear trend.

Often the criteria for stop-loss and take-profit levels are percentage and risk-reward ratio based, which means that the stop-loss has a fixed distance, e.g., 2% and the take-profit is e.g. double the distance of the stop-loss distance (relative to the current price) [64]. To achieve a more adaptive stop-loss and take-profit level determination, a self-developed mechanism called swing detection was added.

This detection finds swings in the price movements differs between swing high and swing low points. A swing high point at time t is a point where the closing price is greater than all closing prices in the intervall $[t - N; t + N]$. Similarly, a swing low point is a point where the closing price is lower than all closing prices in the same intervall. This allows detecting simple support and resistence zones, on which the stop-loss can be set. Therefore, the strategy adapts to the current market behaviour.

The strategy has six parameters, consisting of the long-term and short-term SMA period, the order N of the swing detection, which determines how significant the swing

has to be, the maximum age of the last swing point, as well as a delta for the stop-loss and take-profit levels (identically to those in subsubsection 8.1.1). All parameter combinations where the short-term SMA period is greater than the long-term SMA period, have been filtered out and are not tested.

Parameter Name	Min Value	Max Value	Step Size
Short-Term SMA Period	1	10	1
Long-Term SMA Period	2	20	1
Swing Order N	1	10	1
Take-Profit Delta	0.0	200.0	10.0
Stop-Loss Delta	0.0	100.0	10.0

Table 11: Dual Simple Moving Average Strategy Parameters

8.2.2 Triple Exponential Moving Average Strategy

This strategy consists of three exponential moving averages (EMA) with different periods. The shortest EMA identifies short-term trends, while the medium EMA identifies medium-term trends, and the long EMA identifies long-term trend.

The common application of the triple exponential moving average strategy generates a buy signal if the short-term EMA crosses above the medium-term EMA. Additionally, the medium-term EMA must already be above the long-term EMA to validate the buy signal. For sell signals, the short-term EMA must cross below the medium-term EMA and the medium-term EMA has to be bellow the long-term EMA [65].

Therefore, the long-term EMA acts as a trend filter to reduce false positives mentioned in subsubsection 8.2.1. In this work, the long-term EMA is also used as a trend filter but not by its position relative to two other EMA's. During the backtesting of the classic variant, it was noticed that many false entry signals were still generated, especially in sideways markets. Therefore, the trend was filtered using a minimal slope of the long-term EMA. The slope of the EMA is calculated by the slope of the regression line which is defined by the current EMA value (at time t) and the EMA value i minutes in the past.

$$Slope = \frac{EMA_t - EMA_{t-i}}{i}$$

If the slope is greater than a predefined threshold, only long positions can be opened. On the other hand, if the slope is less than the negated threshold, only short positions can be opened. Otherwise, the strategy cannot open any new positions.

Figure 12 shows the same synthetic price as in Figure 11. Additionally, a third EMA with period 20 is added, which is used to calculate the slope. The red-filled areas indicate zones where no positions can be opened. So the entry signals in these areas are ignored. In the green-filled areas, the entry signals are executed.

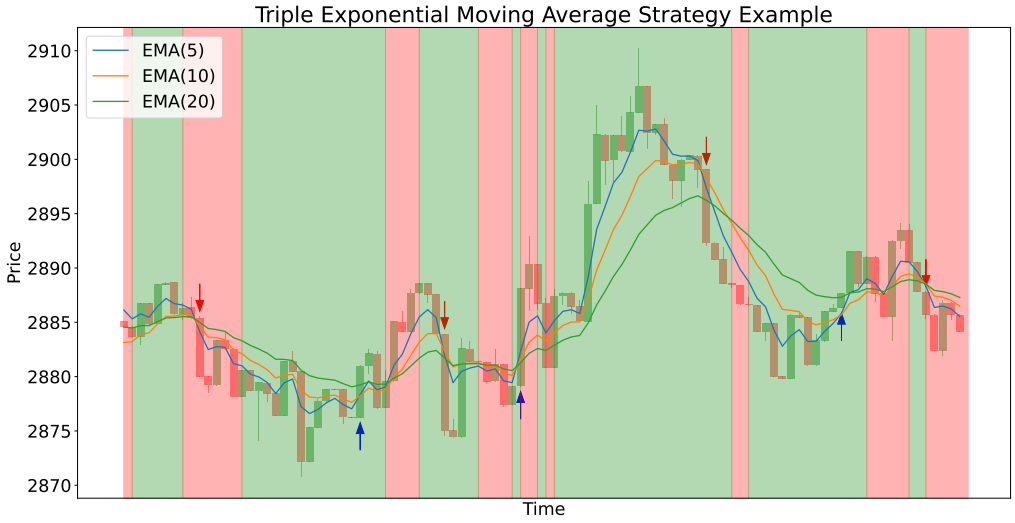

Figure 12: Triple Exponential Moving Average Strategy Example

In this strategy, the stop-loss and take-profit distance have been added as fixed parameters, which are also permuted. All parameter combinations, where the short-term EMA period is greater than the medium-term EMA period, or the medium-term EMA period is greater than the long-term EMA period, are filtered out.

Parameter Name	Min Value	Max Value	Step Size
Short-Term EMA Period	3	10	2
Medium-Term EMA Period	5	30	3
Long-Term EMA Period	10	50	5
Minimum EMA Slope	0.2	1.2	0.2
EMA Slope Window Length i	10	40	10
Stop-Loss Distance	10	100	10
Take-Profit Distance	10	150	10

Table 12: Triple Exponential Moving Average Strategy Parameters

8.2.3 Bollinger Bands Strategy

The bollinger bands strategy is a mean reversion strategy, designed to exploit short-term price fluctuations. This strategy assumes that after approaching or breaking through the outer bands, the price reverts to the central line (the moving average). A buy entry signal is generated if a candle opens below the lower bollinger band and closes above the lower

bollinger band. This indicates a price reversal from an oversold condition. On the other hand, a sell entry signal is generated if the candle opens above the upper bollinger band and closes below the upper bollinger band [66].

Figure 13 shows the synthetic price, identically to Figure 11 and Figure 12. The blue-filled area shows the bollinger band, with the central line.

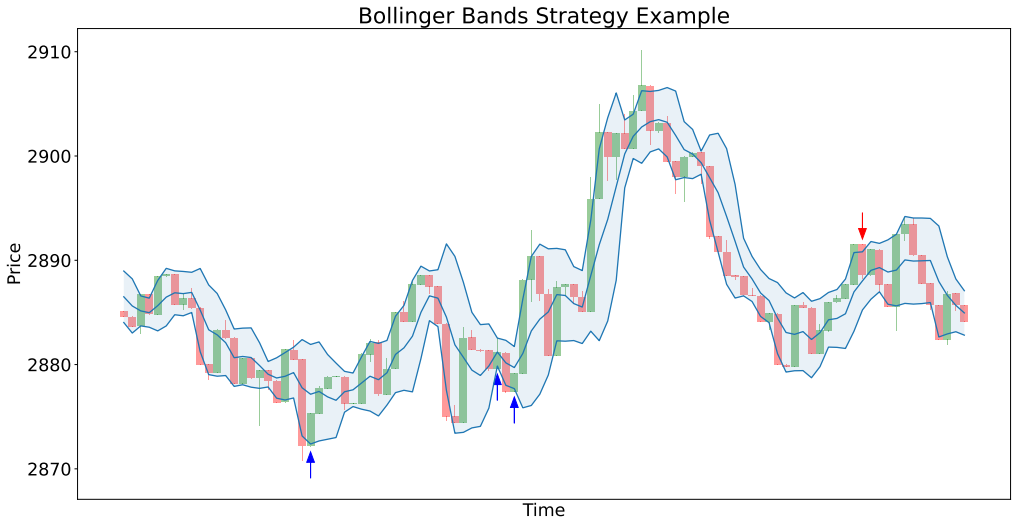

Figure 13: Bollinger Bands Strategy Example

The stop-loss level is set in a predefined distance relative to the current price. The take-profit level is set at the level of the central line , with a certain delta being added (for buy signals) or subtracted (for sell signals) to the value of the middle line.

Parameter Name	Min Value	Max Value	Step Size
Bollinger Band Period	10	25	1
No. of Standard Deviations	1.5	3.0	0.5
Stop-Loss Distance	10.0	100.0	20.0
Take-Profit Delta	-5.0	20.0	2.0

Table 13: Bollinger Band Strategy Parameters

9 Trading-Engine

For backtesting and live execution of trading strategies, a Java framework was built. This chapter describes the use cases as well as the most important components of the trading engine.

9.1 Use-Cases of the Trading-Engine

The developed trading engine represents a flexible and modular software platform that covers various use cases in the field of algorithmic trading. The focus is on combining usability, adaptability, and performance. The following describes the most important use cases and features:

1. **Backtesting trading strategies:** The trading engine enables the simulated execution of trading strategies on historical market data. This allows strategies to be tested under realistic conditions, and evaluate their performance, robustness, and risk characteristics. By replicating historical market conditions, incorrect decisions can be identified early, and the trading strategy can be adjusted without real capital.

2. **Live Execution in Real-Time Operation:** In addition to backtesting, the engine also supports real-time execution of strategies in the market. By an interface to brokers, the engine can receive market data in real time, make trading decisions, and execute orders directly. This enables the engine to be used as the basis for automated trading in production environments.

3. **Using money, and risk management strategies:** An important use case is the implementation of different money and risk management approaches. Users can integrate various strategies for position sizing, stop-loss setting, or profit-taking, and examine their impact on strategies performance in backtests or in live operation. This supports the development of more stable and profitable trading approaches.

4. **Connecting to any broker:** The trading engine is designed to connect to various brokers via interchangeable interfaces. This allows the engine to be combined with different trading systems and platforms. This flexibility enables its use in a wide variety of markets and infrastructures.

5. **Development, and integration of custom trading strategies:** A key feature of the engine is the ability for users to implement their own trading strategies. A clear separation between core functionality and strategy logic allows for flexible integration and testing of individual algorithms.

Most components of the trading engine are interchangeable, and can be implemented by the user. Only the core of the framework, the basic process, and control logic are fixed. This allows users to tailor the engine to their individual needs, develop their own modules for data feeds, order management, strategy, or risk control, and thus ensure a high degree of flexibility in the use, and further development of the platform.

9.2 Architecture

The trading engine consists of many modules which can be clustered in three main categories:

1. Core Modules (Trading Engine)
2. Applications
3. Adapters

Figure 14 shows the components of the trading engine.

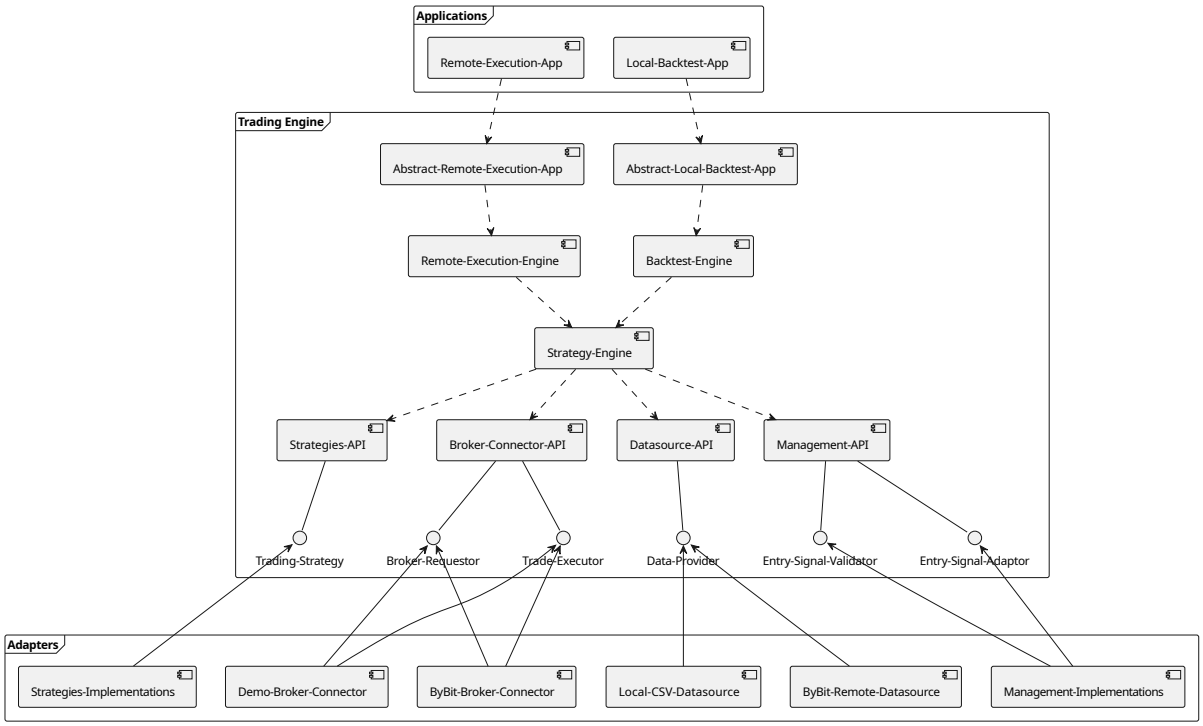

Figure 14: Trading-Engine Components

9.2.1 Plugin Architecture

To achieve the flexibility mentioned in subsection 9.1, a plugin architecture was used. Within the core, interfaces have been defined that define which external functionalities are required (Service provider interface, or SPI). To enable easy interchangeability by users, the implementations of the interfaces in the external modules (Adapters) are loaded by the Java `java.util.ServiceLoader`.

The `ServiceLoader` loads classes at runtime that implement a specific interface or abstract class. The classes to be loaded are specified via a configuration file in the `resources/META-INF/services` directory, which the `ServiceLoader` searches for in every JAR file in the classpath. If a corresponding configuration file is found, the ServiceLoader can create an instance of the respective class using the default constructor [67].

As an example, a service provider interface for read access can be defined.

```java
// Contained in module spi
package com.example.spi.read;

public interface ReadRepository{
    public DomainObject findAll();
}
```

Listing M: SPI Definition

This interface can be implemented in multiple modules. One implementation can be used to read data from files:

```java
// Contained in module file-adapter
package com.example.adapter.file.read;

public class FileReadRepository implements ReadRepository {

    @Override
    public DomainObject findAll(){
        // Logic to read a file
    }
}
```

Listing N: File-Repository Implementation

Here, a configuration file named `com.example.spi.read.ReadRepository` is located in the directory `resources/META-INF/services`. This file contains the fully qualified class name of the implementation class (`com.example.adapter.file.read.FileReadRepository`).

Another implementation can be used to read data from databases:

```java
// Contained in module db-adapter
package com.example.adapter.database.read;

public class DatabaseReadRepository implements ReadRepository {

    @Override
    public DomainObject findAll(){
        // Logic to read a database
    }
}
```

Listing O: Database-Repository Implementation

Similar to the `FileReadRepository` file named `com.example.spi.read.ReadRepository` is located in the directory `resources/META-INF/services`. However, this file contains `com.example.adapter.database.read.DatabaseReadRepository`.

The `ServiceLoader` searched each JAR in the classpath for files under `resources/META-INF/servic` with the name `com.example.spi.read.ReadRepository` that contains the implementation of this interface, and creates them.

Since an external application has the ability to define the classpath, it can add or remove specific JARs. This allows it to determine which implementations to use. For example, it can decide to add only the `file-adapter` module or, if necessary, the `db-adapter` module if loading from a file, and a database simultaneously.

9.2.2 Core Modules

The core of the framework consists of several loosely coupled components that communicate with each other through well-defined APIs. The most central unit is the `Strategy-Engine` which contains the main logic for execution of trading strategies, and controls the cooperation of the other subsystems.

The following components form the core:

1. **Data-Provider:** Obtains market data by a configurable `Datasource-API`. The specific data source (local or remote) is integrated via adapters.

2. **Trading-Strategy:** The actual trading logic is integrated via the `Strategies-API`. This API is separate from the framework, and can be extended as required.

3. **Broker-Requestor & Trade-Executor:** Both communicate with brokers via a generic `Broker-Connector-API`, and enable both order placement, and broker status queries, such as the current account balance or all open positions.

4. **Entry-Signal-Adaptor & Entry-Signal-Validator:** These two components process, and validate generated entry signals. They are linked to the `Management-API` and enable additional risk checks, such as capital requirements or position limits. Currently the implemented management includes all techniques introduced in section 4 and is executed every time a strategy creates an entry signal.

Before the `Strategy-Engine` there are two other engines, named `Remote-Backtest-Engine`, and `Backtest-Engine`, which configure the `Strategy-Engine`. The difference between the engines is that the `Remote-Backtest-Engine` defines the `Strategy-Engine` asynchronously. This means that a remote data source can stream market data via an API. When new data arrives, the `Strategy-Engine` is notified by the data source, and the `Strategy-Engine` starts executing the trading strategy. For a local backtest, the `Strategy-Engine` is configured so that data synchronization is handled by the backtest engine, not by the data source.

Abstract apps are modules that accept and parse user configurations. For example, the task of the `Abstract-Local-Backtest-App` is to ask the user via the console which strategy should be used for the backtest. With the `Abstract-Remote-Execution-App`, the configuration is not done via the console, but primarily via environment variables. The parsed configurations are then passed to the respective engines.

9.2.3 Applications

The application layer defines specific execution environments by defining the classpath with adapters.

1. **Local-Backtest-App:** Executes trading strategies offline by using the `Local-CSV-Datasource` and the `Demo-Broker-Connector`.

2. **Remote-Execution-App:** Executes strategies in real time, and communicates with live broker interfaces.

Each application contains its own `main` method which constructs and starts the respective implementation.

9.3 Demo Broker

For local backtests, the connection to a real broker must be mocked. Therefore, the `Demo-Broker-Connector` is used to replace a real broker. He takes over the main tasks including:

1. **Order execution and position management:** Executing market, limit, take-profit, and stop-loss orders, including execution fee calculation.
2. **Adapt trailing stop positions:** Monitor trailing stop positions, and adapt the stop-level according to the most recent price.
3. **Account balance management:** Manage the available account balance as well as the margin balance.
4. **Store executed trades:** Storing the executed trades is essential for risk management and later analysis.

9.3.1 Order Execution and Position Management

In the context of trading order execution and position management are closely connected, because positions can be translated in three orders where only two of them are actually executed. As described in subsection 4.3 it is necessary that the trading engine supports market and limit orders. In real life environments, most orders are not executed at the expected price, due to slippage [68]. To simulate slippage, the trading engine can either use historical data of the underlying cryptocurrency, which is available in the lowest available timeframe (usually seconds or tick data), or simulate slippage randomly. Two problems exist with simulations using historical data. The first is that data in the seconds or tick range over a long period of time is difficult for private individuals to obtain. Furthermore, a type of latency must be simulated, which simulates the processing time and network traffic in live operation. With random simulations, the problem arises that the backtest results are no longer deterministic, thus a subsequent comparison of different strategies is subject to error. For these reasons, the slippage factor is not taken into account during order execution. This means that market orders are always executed at the most current available price and limit orders are always executed at the specified order level.

Figure 15 shows the process of opening a single position. The margin in euro is calculated using the following formula. $EuroConversion$ is the current price for converting the current counter currency in euro. For example, for ETH/USDC the $EuroConversion$ is the current price of EUR/USDC:

$$Margin = \frac{PositionQuantity}{Leverage * EuroConversion} = \frac{PositionSize * OpenPrice}{Leverage * EuroConversion}$$

The conversion is done in euros because for different currency pairs with different counter currencies, the margin would be calculated in the respective counter currency. For ETH/USDC, the margin is calculated in USDC according to the formula. However, when trading ETH/BTC, the margin is calculated in BTC. To obtain a result that is always in the same unit, or in this case, the same currency, an additional conversion is implemented. It is important to note that it is also possible to implement conversion into

any currency. This way, the margin could also be calculated in BTC, for example. A conversion to euros was preferred here, as the data provider ByBit does not offer direct USD/USDC rates. Since USDC/EUR is offered, it was easier in this case to obtain historical data directly from ByBit and obtain the EUR/USDC rate through a conversion $EUR/USDC = \frac{1}{USDC/EUR}$.

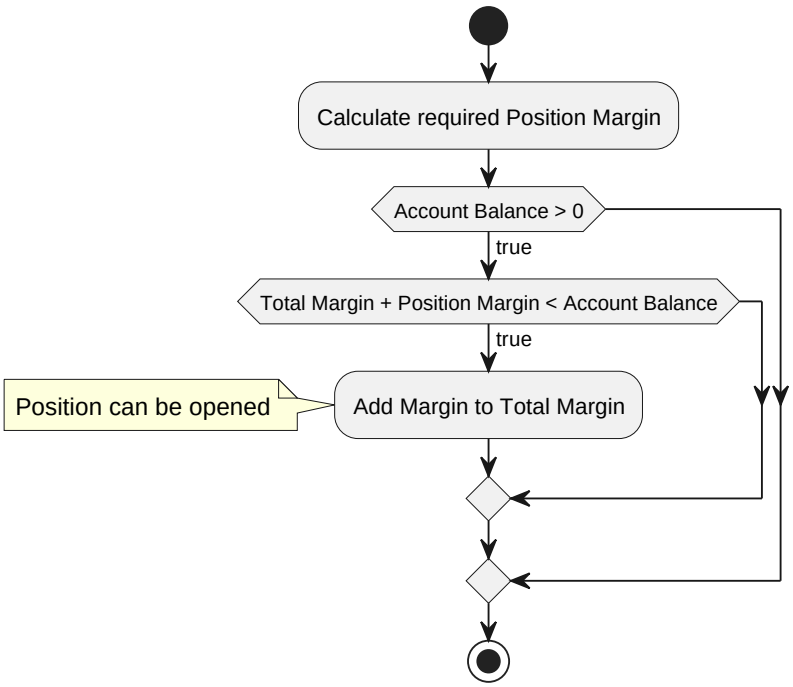

Figure 15: Opening a Single Position

Figure 16 shows the process of closing a single position. The profit in euro is calculated using the following formula:

$$Profit = \frac{(ClosePrice - OpenPrice) * PositionSize}{EuroConversion} * \begin{cases} -1, & \text{if Sell-Position} \\ 1, & \text{otherwise} \end{cases}$$

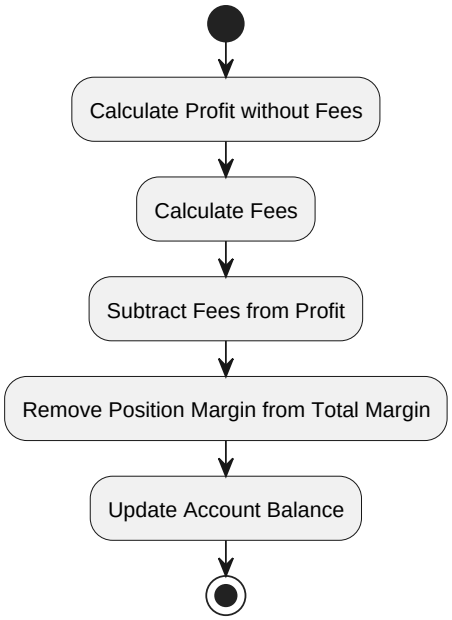

Figure 16: Closing a Position

Figure 17 shows the process of opening a position taking into account that other open positions can already exist.

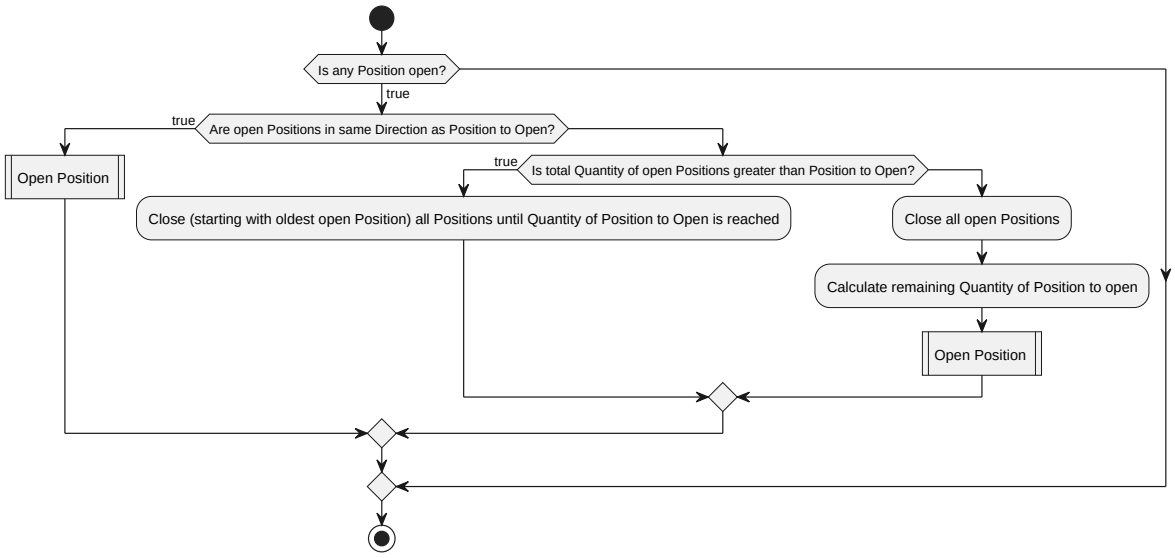

Figure 17: Opening a Position

9.3.2 Adapting Trailing Stop Positions

Trailing stop positions are positions with a dynamic stop order. The stop level moves automatically with the price as soon as it moves in the desired direction. For a buy position, the stop level moves if the price rises. If the price falls, the stop remains unchanged [69].

Figure 18 shows a synthetic closing price (black) and a trailing-stop order with 60 points distance (red) for a buy position. In the green-filled areas, the price moves in the desired direction, so the trailing-stop level follows the price. In the red-filled areas, the prices do not move in the desired direction, so the trailing-stop level does not move. An exception is the red-filled area after $t = 3$, where the price moves in the desired direction, but the trailing stop does not. This is because there is not yet a 60-point gap between the price and the stop level. Therefore, the stop only moves again once the 60-point gap is restored.

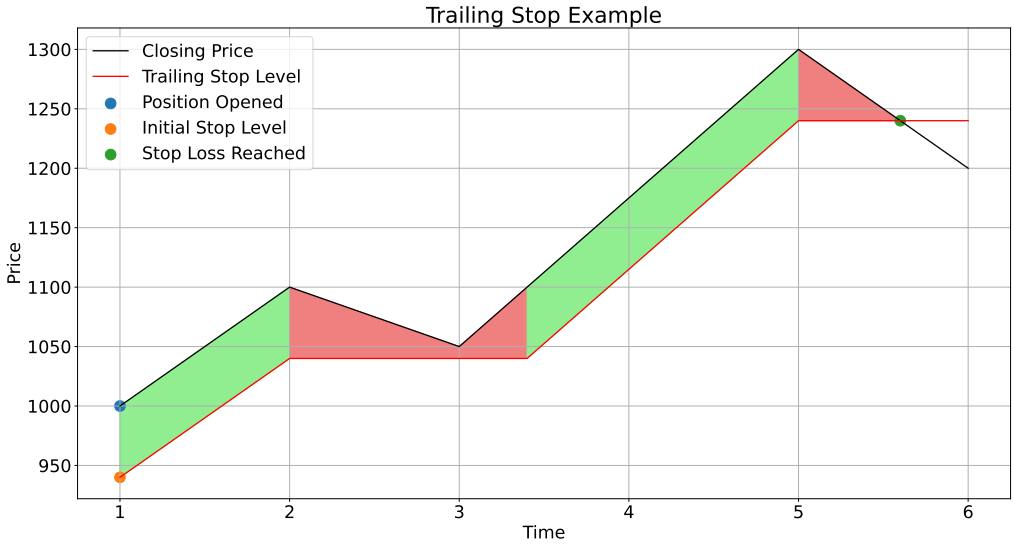

Figure 18: Trailing Stop Example

10 Backtesting Trading Strategies

The quantitative evaluation of trading strategies requires a structured and comprehensible testing environment which was introduced in section 9. This chapter describes the executed backtests, which aim to verify the performance of the develop trading strategies in section 8 under realistic conditions.

Backtesting describes a performance simulation of a trading strategy based on historical data. This involves investigating how a specific strategy would have performed in the past if it had been executed under the given market conditions. The goal is to gain initial insights into the robustness, profitability, and risk profile of the approach before it is used in live trading [70].

10.1 Trading Strategy Parameter Selection

In contrast to the classic machine learning process, the parameter selection for the ultimately executed strategies is performed without the training set. This also applies to strategies that use deep learning (subsection 8.1). Unlike the training of deep learning models, which learn from the training data and adapt their internal states based on the training data, the trading strategies are initialized with fixed parameters. This corresponds to the fitting of deep learning models. However, the subsequent process is identical. The strategies are validated with their previously defined parameters on the validation set to find the best possible parameters.

10.2 Executing Backtests

Due to the large number of parameter combinations shown in Table 14, backtesting all combinations is not possible due to the execution time. Therefore, only 1000 combinations per strategy were tested on the validation set. In order to cover the greatest possible variance of the parameters, the combinations were not sampled randomly from the set of all possible combinations, but the combinations were sorted in lexicographic order and sampled at constant intervals.

Strategy Name	Number of Parameters
Regression AI Strategy	8,820
Classification AI Strategy	5,600
Dual Simple Moving Average Strategy	98,000
Triple Exponential Moving Average Strategy	1,555,200
Bollinger Bands Strategy	8,320

Table 14: Number of Parameters per Strategy

10.3 Evaluating Backtests

Due to the length, the detailed results of the backtests [4], including the parameters [5] that led to the respective results, can be found in the appendix. Here, only the results of the

[4] Detailed results in subsection 15.1

[5] Detailed parameters per strategy and regime in subsection 15.3

strategies with the highest cumulative last equity combined across all market regimes are presented.

Strategy		Classification AI	Dual SMA	Triple EMA	Bollinger Bands
Number of Trades		267	3473	870	275
Maximum Gain		956	357	875	191
Maximum Loss		-245	-160	-135	-64
Profits Std.		280	37	105	41
	Minimum	-2404	-23	-75	49
Equity	**Maximum**	57908	15510	10717	2510
	Last	55567	15159	9502	2510
Maximum Drawdown		4	149	126	51
Win Ratio		64	49	33	42

Table 15: Combined Strategy Results

As seen in Table 15 the Classification AI strategy achieves by far the best cumulative results on the validation set, with the lowest maximum drawdown and the highest win ratio. But on the other hand, it also has the greatest loss.

To compare the out-of-sample performance [6], all strategies have also been tested with the best parameters on the validation set on the backtest set.

Strategy		Classification AI	Dual SMA	Triple EMA	Bollinger Bands
Number of Trades		173	5494	553	444
Maximum Gain		424	268	937	163
Maximum Loss		-144	-63	-207	-88
Profits Std.		92	9	179	32
	Minimum	-2630	-4694	-948	-4378
Equity	**Maximum**	2630	0	4048	0
	Last	-2630	-4694	560	-4365
Maximum Drawdown		200	898314	131	87
Win Ratio		23	26	19	25

Table 16: Combined Out-of-Sample Strytegy Results

Table 16 shows that the strategies have a poor out-of-sample performance, resulting in either high losses or very volatile equity curves, which are not useful for real live trading. This shows that there is also a kind of overfitting in the parameter selection. While the strategies work well on the validation data, they fail in the out-of-sample tests.

10.4 Fee-Impacts

A key component of the realistic evaluation of algorithmic trading strategies is the consideration of transaction costs. These costs can have a significant impact on performance depending on trading frequency, broker model, and market conditions.

[6] Detailed out-of-sample results in subsection 15.2

In this study, trading fees were integrated directly into the trading engine so that they are automatically considered with every order execution in backtesting. The chosen model is based on the ByBit fee structure based on the traded volume (0.02% for maker orders and 0.055% for taker orders). This enables realistic backtests of trading strategies. However, things like slippage were not taken into account but can be added in later improvements.

subsection 10.4 shows the sum of all fees incurred with the best parameters on the validation data (In-Sample) and on the test data (Out-of-Sample).

Strategy	In-Sample		Out-of-Sample
	Total Fees	Relative to PnL	Total Fees
Classification AI	3570	6.4%	1937
Dual SMA	4219	27.8%	2874
Triple EMA	3858	40.6%	4652
Bollinger Bands	1127	44.9%	1858

Table 17: Total Trading Fees

It becomes clear that a considerable amount of fees is incurred, which amounts to up to approximately 40% of the net profit (Triple EMA). However, when comparing the number of trades across strategies, an interesting insight emerges. As shown in Table 15, the Dual SMA strategy executes by far the most trades (3473 in total) on the validation set, but despite this high trading frequency, its relative fee impact is noticeable lower compared to the Triple EMA strategy, which executed only 870 trades. This indicates that not only the number of trades but also entry and exit timing, holding duration, and trade efficiency play a crucial role in determining fee sensitivity.

This highlights that fee efficiency is not strictly a function of trading frequency alone. While one might intuitively expect more trades to always lead to higher costs, the Dual SMA strategy demonstrated that a high-frequency system can still be cost-efficient if implemented with consistent execution logic and stable market conditions.

Most importantly, these findings emphasize the critical importance of incorporating fees in every phase of strategy evaluation. Conducting backtests without fees would draw an overly optimistic and misleading picture of strategy performance. For instance, a strategy that appears profitable on a gross level may become unprofitable once realistic costs are accounted for.

For this reason, no backtests or performance evaluation in this work was conducted without accounting for trading fees. All strategies were tested under the assumption of realistic maker and taker fees, according to the selected broker ByBit. This ensures that every performance metric presented reflects realistic scenarios and therefore increases the practial relevance and robustness of the results.

11 Live-Testing

Despite the poor results from section 10, a strategy will be executed live on a demo account at ByBit for further work. The primary purpose of this live test is not to achieve profitability, but rather to validate the end-to-end functionality of the trading infrastructure under realistic market conditions.

11.1 Broker Setup

To connect to ByBit the `ByBit-Broker-Connector` and `ByBit-Remote-DataSource` components have been used.

The `ByBit-Remote-DataSource` is able to:

1. Request historical market data by the ByBit `Get Kline` API Endpoint [71].
2. Stream market data by the ByBit `Kline` websocket endpoint [72].

The `ByBit-Broker-Connector` is able to execute following actions via the ByBit API:

1. Request the current wallet balance as well as the current margin [73].
2. Placing Orders in the market, which includes open, take-profit, and stop-loss orders. Each Order is supported as limit and market order [74].
3. Request all open positions for one or multiple pairs [75]. In the current paper this only includes positions on ETH/USDC M1.
4. Request closed trades in a specified time period [76].

The connection to ByBit is established by an API key and an API secret which can be created in the ByBit web interface. The distinction between a demo account and a real money account is regulated by ByBit via the base URL, which is different in each case. To use the demo depot, for REST endpoints, the base url `https://api-testnet.bybit.com` has to be used. For the real money account, the base url `https://api.bybit.com` has to be used [77]. To stream the market data, no distinction is required because the base URL `wss://stream.bybit.com/v5/public/linear` [78] for streaming USDC market data is a public endpoint, which can be used without API Key. In the trading engine, the base URLs, API key, and API secret are set by environment variables to achieve an independent implementation, which is configured by the user.

11.2 Live-Test Results

Using the trading engine configuration described above, a live execution of the triple exponential moving average strategy was performed from 20th July, 2025, 05:00 UTCto 28th July, 2025, 05:00 UTC. To keep the environment as similar as possible to the local backtest environment, the balance of the demo account was set to 5000 USDC at startup.

Strategy		Triple EMA
Number of Trades		66
Maximum Gain		304
Maximum Loss		-228
Profits Std.		82
Equity	**Minimum**	3640
	Maximum	5000
	Last	3913
Maximum Drawdown		27
Win Ratio		27

Table 18: Live-Test Statistics

Figure 19: Live-Test Results

As Table 18 and Figure 19 show, the strategy does not deliver profitable results in the live test either.

The logs [7] show that the connection (data streaming) was interrupted several times during execution by the broker or due to network problems. In these cases, the trading engine attempts to reconnect every ten seconds. As soon as the connection is re-established, the engine continues to run normally. The only limitation is that the internal state of the currently executing strategies is not reset, as this function does not yet exist. This is particularly relevant if a connection interruption causes several candles to be skipped, making it appear to the strategies as if there was a jump in the data. A mechanism must therefore be built in that detects gaps in the data and resets the strategies. In the test carried out, however, this did not pose a problem, as the connection interruptions never lasted longer than a few seconds and no candles have been missed.

[7] The logs are not published in this work due to their length.

12 Conclusion

This paper investigates the performance of AI-based and classic trading strategies for the ETH/USDC market on a minute-by-minute basis. The aim was to compare both neural networks (NN, LSTM, CNN) and rule-based strategies (moving averages, Bollinger Bands) with regard to their suitability for high-frequency crypto trading.

Based on historical market data obtained from the broker ByBit from 5th August, 2022, 10:00 UTCto 17th June, 2025, 11:30 UTC, an exploratory data analysis was initially conducted. The raw prices were transformed into logarithmic returns to eliminate data drift and supplemented with technical indicators such as trend, volatility, and momentum indicators.

Subsequently, scaling using MinMaxScaler and a PCA reduction to the principal components, which explain 80% of the variance, were performed for each market regime. This resulted in a manageable feature set for the subsequent models.

To dynamically adapt to different market regimes, a regime detection algorithm was developed that uses two moving averages (SMAs) and the rolling standard deviation to classify the market environment into trend and volatility phases.

Additionally, various money and risk management techniques were evaluated. These include:

1. Position sizing based on the stop-loss distance.
2. Minimum risk-reward ratio.
3. Maximum allowable account risk.
4. Minimum take-profit to cover trading fees.

To further minimize costs, market orders were automatically converted into limit orders.

Hyperparameter optimization with Optuna played an important role in model training. However, regression models designed to predict future logarithmic returns over the next 30 minutes exhibited severe overfitting and produced economically implausible results. Classification models designed to predict buy, sell, or no action achieved on the test-set win rates minimally better than a random model and resulted in losses in simulations. In parallel, classic strategies such as dual SMA, triple EMA, and Bollinger Bands were implemented and their performance compared.

The developed trading strategies were implemented in a modular Java-based trading engine that supports backtests and live trading. The modular structure makes it possible to implement specific trading strategies, adapters to different brokers, and money and risk management mechanisms without the need to adapt the core logic.

While all trading strategy approaches showed excellent results in the in-sample backtest, their performance collapsed significantly in the out-of-sample test, leading to losses or highly volatile equity curves.

A subsequent live test primarily served to verify the technical stability of the trading engine. The engine ran without interruption, even in the event of connection interruption by the broker. The executed triple EMA strategy did not generate any significantly profitable signals.

Overall, the work illustrates that high-frequency ETH/USDC data contains hardly any predictable patterns for the model architectures examined. Both AI models and classical approaches suffer from overfitting, which requires robust optimization methods

and careful validation. While the PCA transformation facilitates model creation, it can dilute important indicator signals. Despite the lack of profitability, the work provides valuable insights into data preprocessing, regime detection, risk management, and the development of a stable trading infrastructure.

Overall, the work demonstrates that high-frequency ETH/USDC data contains hardly any predictable pattern for the model architectures examined. Both AI models and classical approaches suffer from overfitting, which requires robust optimization methods and careful validation. While PCA transformation facilitates model building, it can misrepresent important indicator signals. Despite its lack of profitability, the work provides valuable insights into data preprocessing, regime detection, risk management, and the development of a stable trading infrastructure.

13 Aim of further Works

The challenges and limitations identified in this work open up many concrete and promising starting points for further research and development projects. Particularly with regard to alternative data sources, model architectures and improved validation strategies, opportunities arise to specifically address existing weaknesses while simultaneously exploiting the full potential of data-driven trading strategies in the cryptocurrency market. The following concrete approaches can form the basis for future work:

1. **Expanding the feature set:** The integration of order book data (e.g., depth snapshots or order flow imbalance), sentiment indices from social media and news, and macroeconomic indicators can provide uncorrelated information and improve forecasting capabilities.

2. **Alternative model architectures:** Reinforcement learning agents (e.g., DQN, PPO) could learn to dynamically allocate between long, short, and cash positions. Transformer models with attention mechanisms offer the possibility of specifically weighting relevant market phases. Ensemble methods from multiple model types promise to reduce overfitting.

3. **Meta-strategies and target size adjustment:** Instead of exact return predictions, models could only forecast the trading direction or volatility classes to increase stability. A meta-strategy that switches daily between different sub-strategies based on their current performance expectations can create additional robustness.

4. **Robust optimization and validation:**Monte Carlo cross-validation with random parameter starts and scenario variation, combined with restrictive regularization (ElasticNet, DropConnect) and data augmentation, can further mitigate overfitting.

5. **Diversification of markets and time frames:** Checking trading signals on different timeframes (5-minute, 15-minute, hourly) and in other currency pairs (e.g., BTC/USDC, altcoins) enables the analysis of correlation structures and low-risk diversification.

6. **Composite strategies:** Entry confirmations should only occur when multiple algorithms issue a signal simultaneously. Portfolio-level management that weights strategies according to risk budget and other risk metrics can stabilize overall returns.

7. **Stress tests and scenario analyses:** Simulating extreme market events such as flash crashes or liquidity shortages, including slippage and order execution models, provides important insights into strategy robustness.

With these extensions on the stable technical infrastructure developed in this work, it could be possible to design a significantly more robust and better generalizing trading approach. The insights gained in modeling, regime detection, risk management, and strategy integration compared with the provided suggestions for further researches form a solid foundation for the development of practice-relevant algorithms that can deliver consistent results not only in backtests but also in live operation.

14 References

[1] Investing.com - Die besten Krypto Broker – Juni 2025. 2025. URL: https://de.investing.com/brokers/cryptocurrency-brokers/#:~:text=Zu%20den%20wichtigsten%20Aufgaben%20eines,an%20Kryptow%C3%A4hrungen%20als%20einzelne%20B%C3%B6rsen. (visited on 06/18/2025).

[2] ByBit.com. 2025. URL: https://www.bybit.com/en/ (visited on 06/18/2025).

[3] ByBit API Doc. 2025. URL: https://www.bybit.com/en/%20https://bybit-exchange.github.io/docs/v5/intro (visited on 06/18/2025).

[4] IG.com. 2025. URL: https://www.ig.com/de/kryptowaehrungshandel?_gl=1*15gtia3*_up*MQ..*_gs*MQ..&gclid=CjwKCAjwpMTCBhA-EiwA_-MsmST2-xxZ7Kxhnw1PDrrdvL7kdRSPmknOM0R2x2ShxY2-5F5G5_mIFBoCUikQAvD_BwE&gclsrc=aw.ds&gbraid=0AAAADzDEsR-vDC7urfY1ODmU7i3x7bQU (visited on 06/18/2025).

[5] IG API Doc. 2025. URL: https://labs.ig.com/getting-started.html (visited on 06/18/2025).

[6] Capital.com. 2025. URL: https://capital.com/de-de (visited on 06/18/2025).

[7] Capital API Doc. 2025. URL: https://open-api.capital.com/#section/General-information (visited on 06/18/2025).

[8] ByBit API Doc - Get Kline. 2025. URL: https://bybit-exchange.github.io/docs/v5/market/kline (visited on 06/18/2025).

[9] Macrosynergy - Classifying market regimes. 2025. URL: https://macrosynergy.com/research/classifying-market-regimes/ (visited on 06/24/2025).

[10] Felix Haase and Matthias Neuenkirch. "Predictability of bull and bear markets: A new look at forecasting stock market regimes (and returns) in the US". In: *International Journal of Forecasting* 39.2 (2023), pp. 587–605. DOI: https://doi.org/10.1016/j.ijforecast.2022.01.004.

[11] Arjun Prakash et al. "Structural Clustering of Volatility Regimes for Dynamic Trading Strategies". In: *Applied Mathematical Finance* 28.3 (May 2021), pp. 236–274. DOI: 10.1080/1350486x.2021.2007146.

[12] Szabolcs Mike and J. Doyne Farmer. An empirical behavioral model of liquidity and volatility. 2007. arXiv: 0709.0159 [q-fin.ST].

[13] IG.com - Moving averages: a guide to trend trading. 2025. URL: https://www.ig.com/en/trading-strategies/moving-averages--a-guide-to-trend-trading-241031 (visited on 06/25/2025).

[14] wikipedia.org - Volatility (finance). 2025. URL: https://en.wikipedia.org/w/index.php?title=Volatility_(finance)&oldid=1291878143 (visited on 06/25/2025).

[15] BritannicaMoney - Calculating position size in trading: The key to risk management. 2024. URL: https://www.britannica.com/money/calculating-position-size (visited on 06/23/2025).

[16] bitpanda - Risk reward ratio: what is it & why is it important? 2024. URL: https://www.bitpanda.com/academy/en/lessons/risk-reward-ratio-what-is-it-and-why-is-it-important/ (visited on 06/23/2025).

[17] quantifiedstrategies.com - Trading Performance: Strategy Metrics, Risk-Adjusted Metrics, And Backtest. 2025. URL: https://www.quantifiedstrategies.com/trading-performance/ (visited on 07/15/2025).

[18] investopia.com - Maximum Drawdown (MDD): Definition and Formula. 2025. URL: https://www.investopedia.com/terms/m/maximum-drawdown-mdd.asp (visited on 07/15/2025).

[19] investopia.com - Drawdown: What It Is, Risks, and Examples. 2025. URL: https://www.investopedia.com/terms/d/drawdown.asp (visited on 07/15/2025).

[20] binance.com - Win Rate. 2025. URL: https://academy.binance.com/en/glossary/win-rate (visited on 07/16/2025).

[21] Investiopia.com - Trend Trading: The 4 Most Common Indicators. 2025. URL: https://www.investopedia.com/articles/active-trading/041814/four-most-commonlyused-indicators-trend-trading.asp (visited on 06/19/2025).

[22] Investiopia.com - What is EMA? How to Use Exponential Moving Average With Formula. 2024. URL: https://www.investopedia.com/terms/e/ema.asp (visited on 06/19/2025).

[23] Investiopia.com - What Is MACD? 2024. URL: https://www.investopedia.com/terms/m/macd.asp (visited on 06/19/2025).

[24] Investiopia.com - Average True Range (ATR) Formula, What It Means, and How to Use It. 2025. URL: https://www.investopedia.com/terms/a/atr.asp (visited on 06/19/2025).

[25] Investiopia.com - Bollinger Bands: What They Are, and What They Tell Investors. 2024. URL: https://www.investopedia.com/terms/b/bollingerbands.asp (visited on 06/19/2025).

[26] Investiopia.com - What Is a Momentum Indicator? Definition and Common Indicators. 2025. URL: https://www.investopedia.com/investing/momentum-and-relative-strength-index/ (visited on 06/19/2025).

[27] Investiopia.com - Relative Strength Index (RSI) Indicator Explained With Formula. 2024. URL: https://www.investopedia.com/terms/r/rsi.asp (visited on 06/19/2025).

[28] Bharathi Deepa and K. Ramesh. "Epileptic seizure detection using deep learning through min max scaler normalization". In: *International journal of health sciences* (2022).

[29] scikit-learn.org - Importance of Feature Scaling. 2025. URL: https://scikit-learn.org/stable/auto_examples/preprocessing/plot_scaling_importance.html (visited on 06/20/2025).

[30] scikit-learn.org - MinMaxScaler. 2025. URL: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html (visited on 06/20/2025).

[31] Yoshua Bengio. "Practical recommendations for gradient-based training of deep architectures". In: *CoRR* abs/1206.5533 (2012). arXiv: 1206.5533.

[32] Wikipedia.org - Principal component analysis. 2024. URL: https://en.wikipedia.org/w/index.php?title=Principal_component_analysis&oldid=1296008542 (visited on 06/20/2025).

[33] Wikipedia.org - Curse of dimensionality. 2024. URL: https://en.wikipedia.org/w/index.php?title=Curse_of_dimensionality&oldid=1296421397 (visited on 06/20/2025).

[34] keras.io. 2025. URL: https://keras.io/ (visited on 07/06/2025).

[35] tensorflow.org - Windows Install. 2025. URL: https://www.tensorflow.org/install/pip#windows-native (visited on 07/06/2025).

[36] Jasman Pardede and Khairul Rijal. "The Effect of Hyperparameters on Faster R-CNN in Face Recognition Systems". In: *Jurnal RESTI (Rekayasa Sistem dan Teknologi Informasi)* 9 (May 2025), pp. 436–448. DOI: 10.29207/resti.v9i3.6405.

[37] optuna.io - Basic Concepts. 2025. URL: https://optuna.readthedocs.io/en/stable/index.html (visited on 07/06/2025).

[38] Yosua Harmoni, Kartika Maulida Hindrayani, and Dwi Prasetya. "Optimizing Categorical Boosting Model with Optuna for Anti-Tuberculosis Drugs Classification". In: *Indonesian Journal of Electronics, Electromedical Engineering, and Medical Informatics* 7 (May 2025), pp. 401–414. DOI: 10.35882/ijeeemi.v7i2.92.

[39] Saleem Abdullah, Ihsan Ullah, and Fazal Ghani. "Heterogeneous wireless network selection using feed forward double hierarchy linguistic neural network". In: *Artificial Intelligence Review* 57 (July 2024). DOI: 10.1007/s10462-024-10826-y.

[40] Maria Vakalopoulou et al. "Deep Learning: Basics and Convolutional Neural Networks (CNNs)". In: *Machine Learning for Brain Disorders*. Ed. by Olivier Colliot. New York, NY: Springer US, 2023, pp. 77–115. DOI: 10.1007/978-1-0716-3195-9_3.

[41] keras.io/flatten. 2025. URL: https://keras.io/api/layers/reshaping_layers/flatten/ (visited on 07/08/2025).

[42] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *CoRR* abs/1502.03167 (2015). arXiv: 1502.03167.

[43] keras.io/dropout. 2025. URL: https://keras.io/api/layers/regularization_layers/dropout/ (visited on 07/08/2025).

[44] Daoping Du et al. "Underwater image restoration based on a dual-branch super-resolution residual network". In: *Multimedia Systems* 31 (June 2025). DOI: 10.1007/s00530-025-01861-y.

[45] Thomas Fischer and Christopher Krauss. "Deep learning with long short-term memory networks for financial market predictions". In: *European Journal of Operational Research* 270.2 (2018), pp. 654–669. DOI: https://doi.org/10.1016/j.ejor.2017.11.054.

[46] Thorir Mar Ingolfsson. 2021. URL: https://thorirmar.com/post/insight_into_lstm/ (visited on 07/08/2025).

[47] Greg Van Houdt, Carlos Mosquera, and Gonzalo Nápoles. "A review on the long short-term memory model". In: *Artificial Intelligence Review* 53.8 (2020), pp. 5929–5955. DOI: 10.1007/s10462-020-09838-1.

[48] Yuxin Wang. "Advanced Network Traffic Prediction Using Deep Learning Techniques: A Comparative Study of SVR, LSTM, GRU, and Bidirectional LSTM Models". In: *ITM Web of Conferences* 70 (Jan. 2025). DOI: 10.1051/itmconf/20257003021.

[49] keras.io/repeat$_v$*ector*. 2025. URL: https://keras.io/api/layers/reshaping_layers/repeat_vector/ (visited on 07/08/2025).

[50] keras.io/time$_d$*istributed*. 2025. URL: https://keras.io/api/layers/recurrent_layers/time_distributed/ (visited on 07/08/2025).

[51] Pedro Lara-Benítez, Manuel Carranza-García, and José C. Riquelme. "An Experimental Review on Deep Learning Architectures for Time Series Forecasting". In: *CoRR* abs/2103.12057 (2021). arXiv: 2103.12057.

[52] Hans Weytjens and Jochen De Weerdt. "Process Outcome Prediction: CNN vs. LSTM (with Attention)". In: *CoRR* abs/2104.06934 (2021). arXiv: 2104.06934.

[53] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. 2015. arXiv: 1409.1556 [cs.CV].

[54] Ashish Vaswani et al. "Attention Is All You Need". In: *CoRR* abs/1706.03762 (2017). arXiv: 1706.03762.

[55] keras.io/concatenate. 2025. URL: https://keras.io/api/layers/merging_layers/concatenate/ (visited on 07/08/2025).

[56] Junyoung Chung et al. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. 2014. arXiv: 1412.3555 [cs.NE].

[57] Yinghan Li et al. A Deep Learning Algorithm Based on CNN-LSTM Framework for Predicting Cancer Drug Sales Volume. 2025. arXiv: 2506.21927 [cs.CE].

[58] R. Cont. "Empirical properties of asset returns: stylized facts and statistical issues". In: *Quantitative Finance* 1.2 (2001), pp. 223–236.

[59] Investiopia.com - What Is a Trading Strategy? How to Develop One. 2024. URL: https://www.investopedia.com/terms/t/trading-strategy.asp (visited on 07/12/2025).

[60] Investiopia.com - 10 Steps to Building a Winning Trading Plan. 2024. URL: https://www.investopedia.com/articles/trading/04/042104.asp (visited on 07/12/2025).

[61] Yunus Emre Akdoğan. "WHEN MACHINES LEARN TECHNICAL ANALYSIS: AN APPLICATION ON TECHNICAL ANALYSIS WITH MACHINE LEARNING IN BORSA ISTANBUL". In: *Trakya Üniversitesi Sosyal Bilimler Dergisi* 27 (Mar. 2025), pp. 275–302. DOI: 10.26468/trakyasobed.1514346.

[62] Sabyasachi Mazumder, Sayan Neogy, and Sahana Das. "Simple Moving Average (SMA) Crossover Strategy with Buy Sell Indicator". In: *Asia-Pacific Journal of Management and Technology* 03 (Apr. 2023), pp. 26–40. DOI: 10.46977/apjmt.2023.v03i04.004.

[63] medium.com - SMA Dual Moving Average Trading Strategy. 2024. URL: https://medium.com/@redsword_23261/sma-dual-moving-average-trading-strategy-c1b764fe809c (visited on 07/14/2025).

[64] investopia.com - How to Use a Moving Average to Buy Stocks. 2025. URL: https://www.investopedia.com/articles/active-trading/052014/how-use-moving-average-buy-stocks.asp (visited on 07/14/2025).

[65] howtotrade.com - The Triple Moving Average Crossover Strategy (With Examples). 2025. URL: https://howtotrade.com/trading-strategies/triple-moving-average-crossover/ (visited on 07/15/2025).

[66] Chun-Hao Chen et al. "An Advanced Optimization Approach for Long-Short Pairs Trading Strategy Based on Correlation Coefficients and Bollinger Bands". In: *Applied Sciences* 12 (Jan. 2022), p. 1052. DOI: 10.3390/app12031052.

[67] oracle.com - ServiceLoader (Java SE 17 & JDK 17). 2025. URL: https://docs.oracle.com/javase/8/docs/api/java/util/ServiceLoader.html (visited on 06/26/2025).

[68] ig.com - Slippage definition. 2025. URL: https://www.ig.com/uk/glossary-trading-terms/slippage-definition (visited on 06/26/2025).

[69] ig.com - Trailing stop orders definition. 2025. URL: https://www.ig.com/uk/glossary-trading-terms/trailing-stops-definition (visited on 06/26/2025).

[70] Jiarui Ni and Chengqi Zhang. "An Efficient Implementation of the Backtesting of Trading Strategies". In: vol. 3758. Nov. 2005, pp. 126–131. DOI: 10.1007/11576235_17.

[71] bybit-exchange.github.io - Get Kline. 2025. URL: https://bybit-exchange.github.io/docs/v5/market/kline (visited on 07/19/2025).

[72] bybit-exchange.github.io - Kline. 2025. URL: https://bybit-exchange.github.io/docs/v5/websocket/public/kline (visited on 07/19/2025).

[73] bybit-exchange.github.io - Get Wallet Balance. 2025. URL: https://bybit-exchange.github.io/docs/v5/account/wallet-balance (visited on 07/19/2025).

[74] bybit-exchange.github.io - Place Order. 2025. URL: https://bybit-exchange.github.io/docs/v5/order/create-order (visited on 07/19/2025).

[75] bybit-exchange.github.io - Get Position Info. 2025. URL: https://bybit-exchange.github.io/docs/v5/position (visited on 07/19/2025).

[76] bybit-exchange.github.io - Get Closed PnL. 2025. URL: https://bybit-exchange.github.io/docs/v5/position/close-pnl (visited on 07/19/2025).

[77] bybit-exchange.github.io - Integration Guidance. 2025. URL: https://bybit-exchange.github.io/docs/v5/guide (visited on 07/19/2025).

[78] bybit-exchange.github.io - Connect. 2025. URL: https://bybit-exchange.github.io/docs/v5/ws/connect (visited on 07/19/2025).

15 Appendix

15.1 Strategy Results on Validation Data

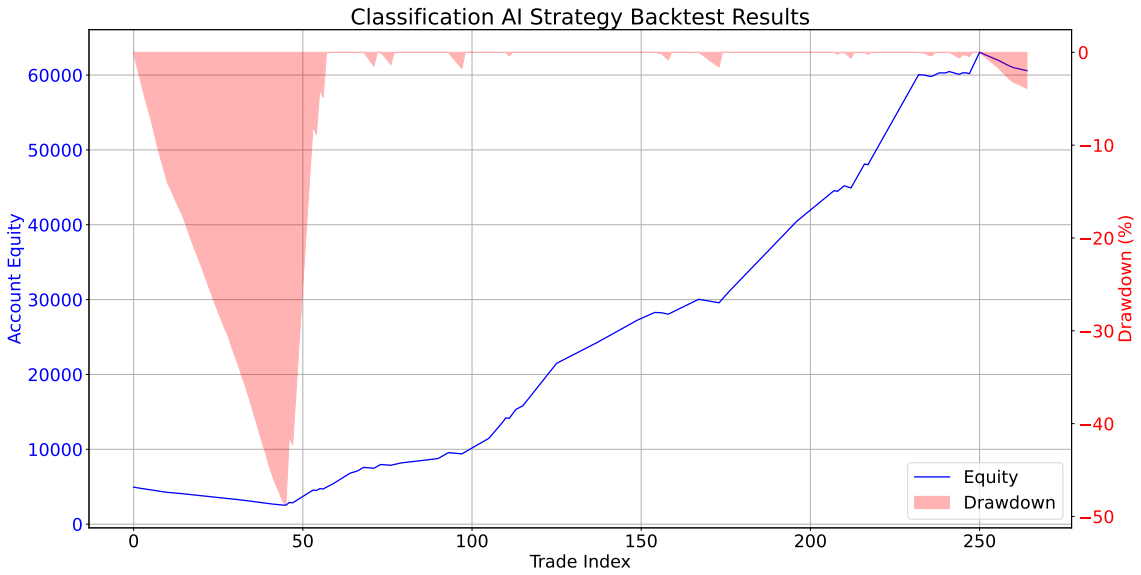

Figure 20: Classification AI Strategy Results

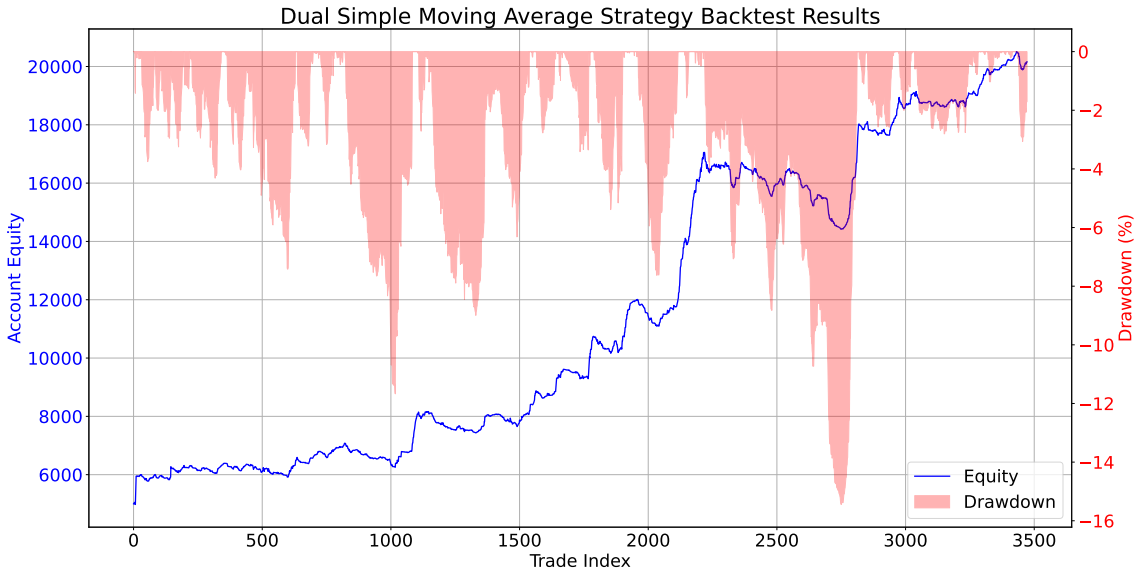

Figure 21: Dual Simple Moving Average Strategy Results

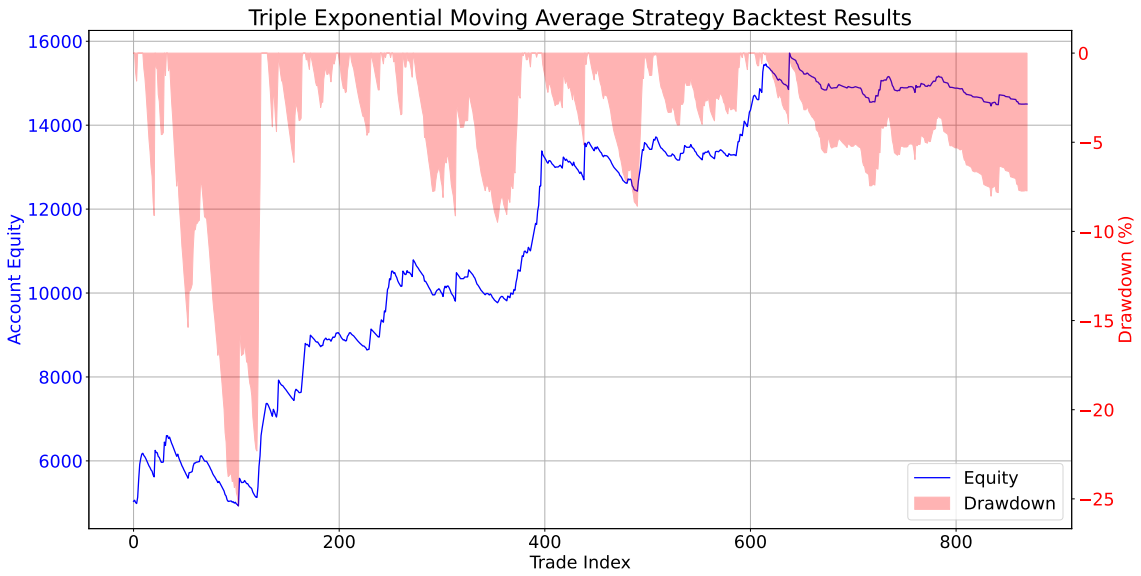

Figure 22: Triple Exponential Moving Average Strategy Results

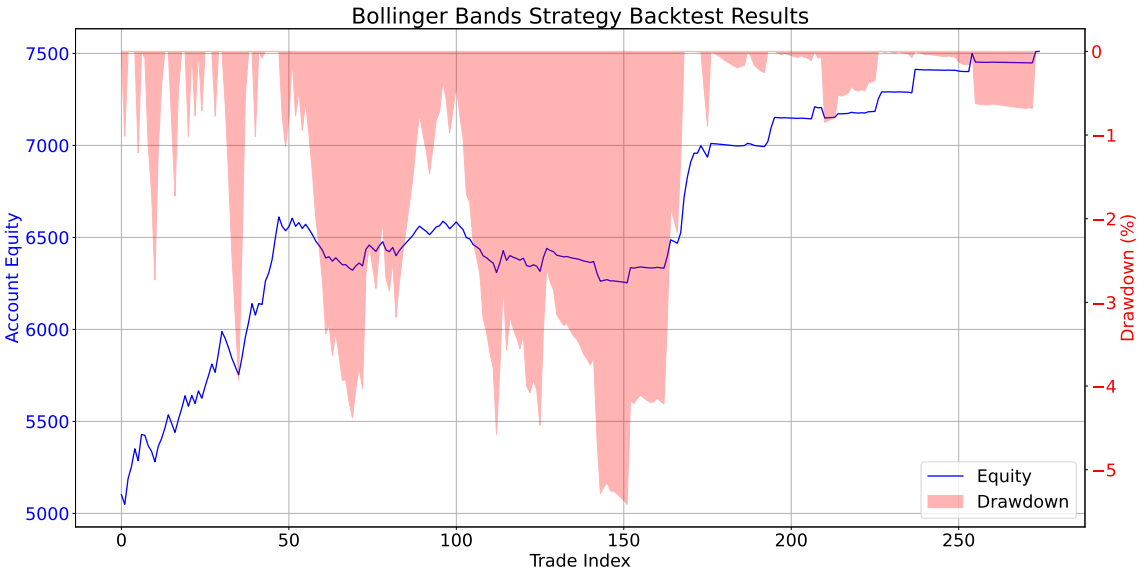

Figure 23: Bollinger Bands Strategy Results

15.2 Out-of-Sample Strategy Results

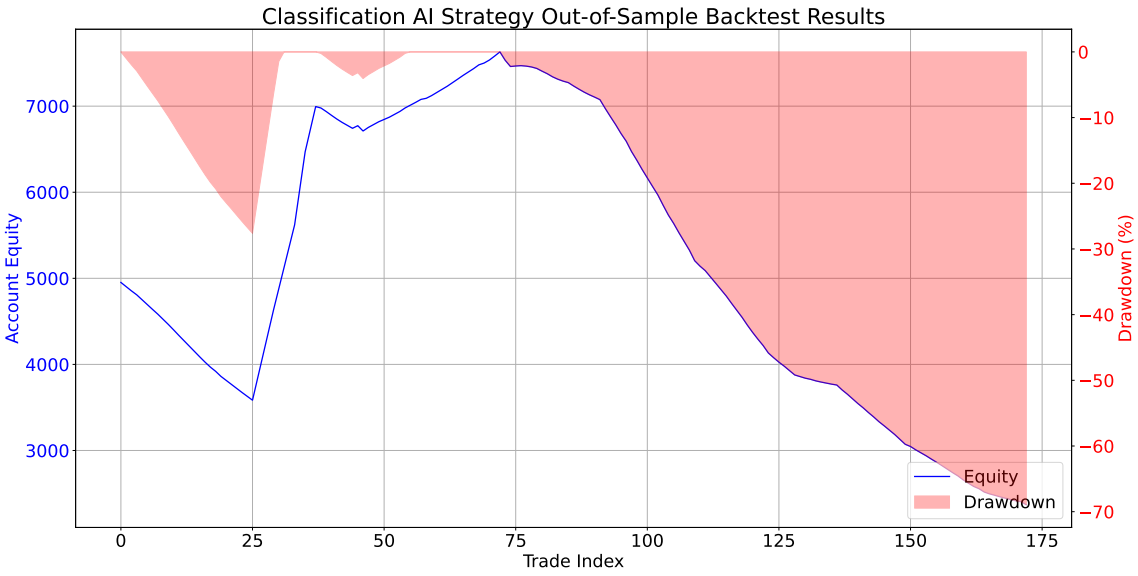

Figure 24: Out-of-Sample Classification AI Strategy Results

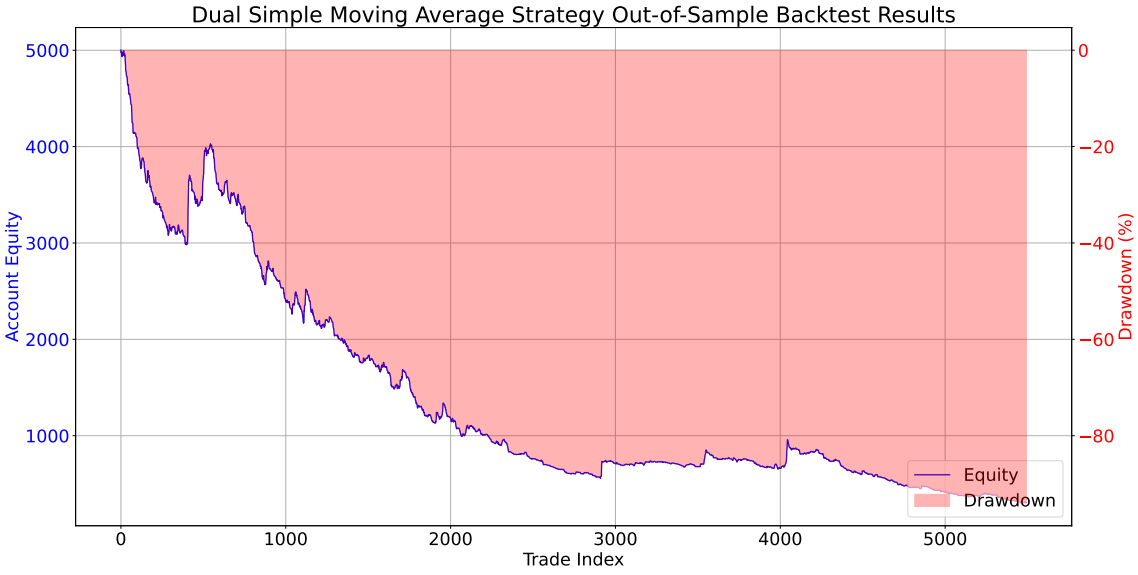

Figure 25: Dual Simple Out-of-Sample Moving Average Strategy Results

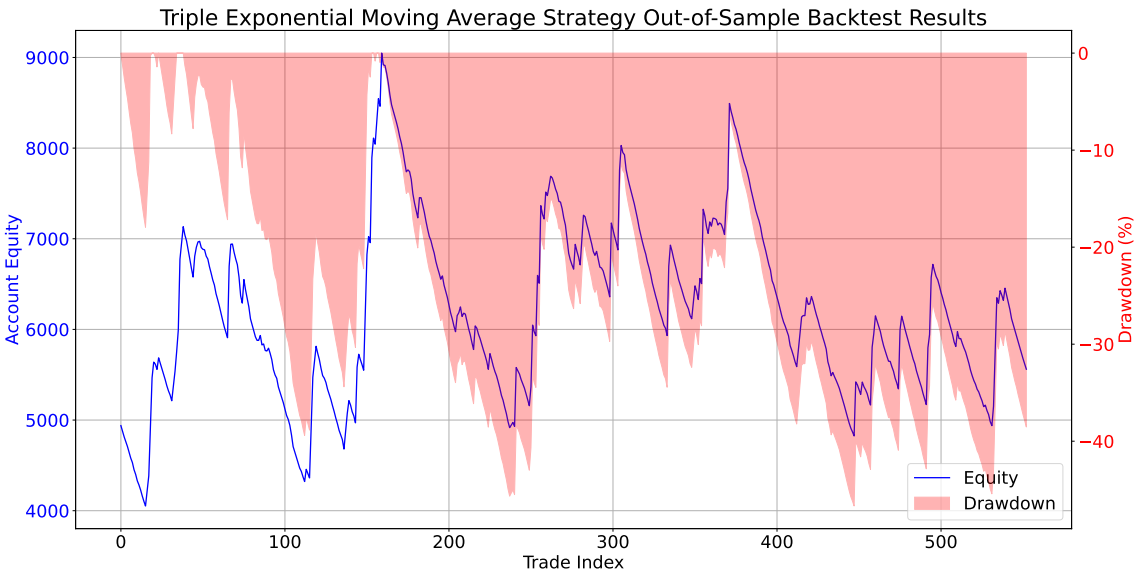

Figure 26: Triple Exponential Out-of-Sample Moving Average Strategy Results

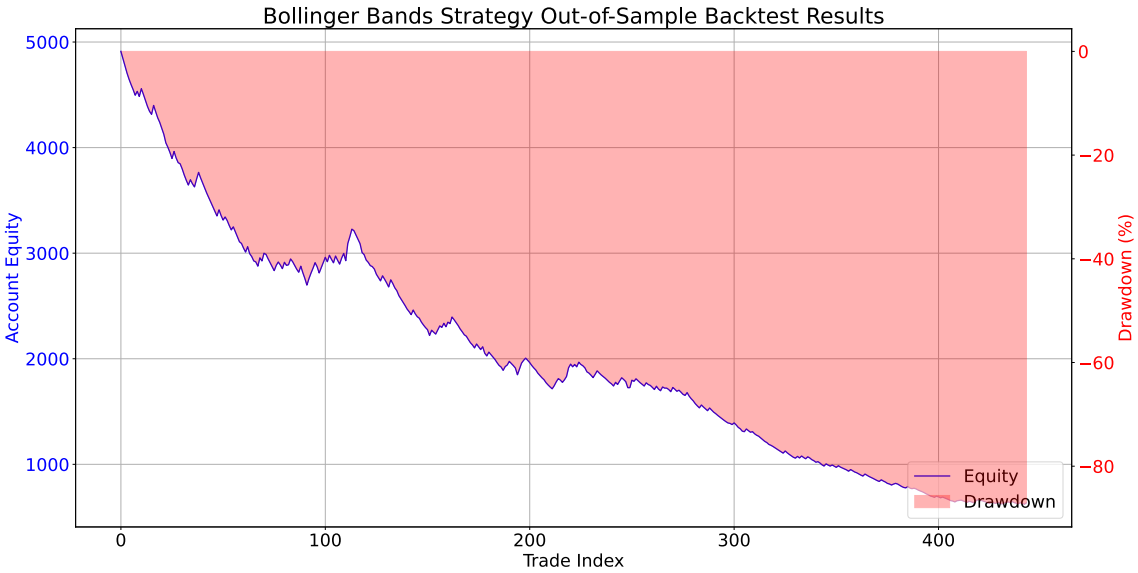

Figure 27: Out-of-Sample Bollinger Bands Strategy Results

15.3 Best Strategy Parameters

15.3.1 Classification AI Strategy Backtest Results

Regime		Downtrend + Low Volatility			Downtrend + High Volatility		
		$Q_{0.33}$	$Q_{0.66}$	Q_1	$Q_{0.33}$	$Q_{0.66}$	Q_1
Min. Probability for Entry		0.3	0.6	0.3	-	-	-
Trailing Stop Orders		false	false	false	-	-	-
Stop-Loss Delta		11	16	11	-	-	-
Take-Profit Delta		96	61	86	-	-	-
Number of Trades		29	30	20	-	-	-
Maximum Gain		518	371	803	-	-	-
Maximum Loss		-111	-95	-112	-	-	-
Profits Std.		212	108	199	-	-	-
Equity	**Minimum**	-47	0	0	-	-	-
	Maximum	3833	7978	15144	-	-	-
	Last	3532	7978	15144	-	-	-
Maximum Drawdown		23	1	3	-	-	-
Win Ratio		48	96	95	-	-	-

Table 19: Classification AI Strategy Results I

Regime	Sideways Trend + Low Volatility			Sideways Trend + High Volatility		
	$Q_{0.33}$	$Q_{0.66}$	Q_1	$Q_{0.33}$	$Q_{0.66}$	Q_1
Min. Probability for Entry	0.3	0.3	0.4	-	-	-
Trailing Stop Orders	false	false	false	-	-	-
Stop-Loss Delta	11	11	6	-	-	-
Take-Profit Delta	81	81	76	-	-	-
Number of Trades	4	13	32	-	-	-
Maximum Gain	530	530	956	-	-	-
Maximum Loss	-109	-152	-139	-	-	-
Profits Std.	257	255	333	-	-	-
Equity Minimum	-65	-301	-463	-	-	-
Equity Maximum	701	1999	4583	-	-	-
Equity Last	592	1705	4583	-	-	-
Maximum Drawdown	15	14	18	-	-	-
Win Ratio	50	46	46	-	-	-

Table 20: Classification AI Strategy Results II

Regime	Uptrend + Low Volatility			Uptrend + High Volatility		
	$Q_{0.33}$	$Q_{0.66}$	Q_1	$Q_{0.33}$	$Q_{0.66}$	Q_1
Min. Probability for Entry	0.4	0.3	0.8	-	-	-
Trailing Stop Orders	false	true	false	-	-	-
Stop-Profit Delta	11	26	11	-	-	-
Take-Profit Delta	86	76	86	-	-	-
Number of Trades	23	58	56	-	-	-
Maximum Gain	568	250	469	-	-	-
Maximum Loss	-66	-53	-245	-	-	-
Profits Std.	279	125	268	-	-	-
Equity Minimum	-193	-819	-619	-	-	-
Equity Maximum	6275	6348	11467	-	-	-
Equity Last	6275	6348	9427	-	-	-
Maximum Drawdown	36	0	100	-	-	-
Win Ratio	60	68	55	-	-	-

Table 21: Classification AI Strategy Results III

15.3.2 Dual Simple Moving Average Strategy Backtest Results

Regime		Downtrend + Low Volatility			Downtrend + High Volatility		
		$Q_{0.33}$	$Q_{0.66}$	Q_1	$Q_{0.33}$	$Q_{0.66}$	Q_1
SMA Period	**Short-Term**	11	3	3	3	-	-
	Long-Term	13	19	5	5	-	-
Swing Order N		5	7	9	7	-	-
Swing Max Age		10	20	10	40	-	-
Trailing Stop Order		false	false	false	true	-	-
Take-Profit Delta		140	155	185	95	-	-
Stop-Loss Delta		65	50	35	65	-	-
Number of Trades		267	501	327	2	-	-
Maximum Gain		135	211	162	218	-	-
Maximum Loss		-81	-160	- 81	-41	-	-
Profits Std.		37	34	56	130	-	-
Equity	**Minimum**	-11	-122	- 644	-41	-	-
	Maximum	1146	2250	2123	177	-	-
	Last	1098	2044	2123	177	-	-
Maximum Drawdown		108	259	836	0	-	-
Win Ratio		52	49	49.5	50	-	-

Table 22: Dual Simple Moving Average Strategy Results I

Table 23: Dual Simple Moving Average Strategy Results II

Regime		Sideways Trend + Low Volatility			Sideways Trend + High Volatility		
		$Q_{0.33}$	$Q_{0.66}$	Q_1	$Q_{0.33}$	$Q_{0.66}$	Q_1
SMA Period	**Short-Term**	3	3	3	3	5	3
	Long-Term	19	19	19	5	11	19
Swing Order N		1	3	5	9	3	1
Swing Max Age		30	10	30	30	10	10
Trailing Stop Order		false	false	false	false	false	true
Take-Profit Delta		155	125	200	65	155	110
Stop-Loss Delta		65	5	80	20	35	35
Number of Trades		94	111	19	1	1	1
Maximum Gain		124	357	54	3	4	25
Maximum Loss		-40	-67	-40	3	4	0
Profits Std.		23	37	20	0	0	0
Equity	**Minimum**	-11	-8	0	0	0	0
	Maximum	585	418	126	3	4	6
	Last	573	385	126	3	4	6
Maximum Drawdown		125	116	77	0	0	0
Win Ratio		45	33	57	100	100	100

Table 24: Dual Simple Moving Average Strategy Results III

	Regime	Uptrend + Low Volatility			Uptrend + High Volatility		
		$Q_{0.33}$	$Q_{0.66}$	Q_1	$Q_{0.33}$	$Q_{0.66}$	Q_1
SMA Period	**Short-Term**	3	5	3	5	11	-
	Long-Term	19	11	19	11	13	-
Swing Order N		5	7	7	7	3	-
Swing Max Age		30	10	40	10	20	-
Trailing Stop Order		false	false	false	false	true	-
Take-Profit Delta		200	185	185	155	140	-
Stop-Loss Delta		80	50	20	50	35	-
Number of Trades		1468	526	152	1	1	-
Maximum Gain		189	234	337	160	83	-
Maximum Loss		-121	-114	-59	160	83	-
Profits Std.		27	52	46	0	0	-
Equity	**Minimum**	-149	-218	0	0	0	-
	Maximum	2860	5120	975	160	83	-
	Last	2490	5025	835	160	83	-
Maximum Drawdown		427	408	14	0	0	-
Win Ratio		48	53	42	100	83	-

15.3.3 Triple Exponential Moving Average Strategy Backtest Results

Regime		Downtrend + Low Volatility			Downtrend + High Volatility		
		$Q_{0.33}$	$Q_{0.66}$	Q_1	$Q_{0.33}$	$Q_{0.66}$	Q_1
EMA Period	**Short-Term**	3	9	7	7	5	-
	Medium-Term	5	14	14	14	23	-
	Long-Term	10	25	40	40	45	-
Minimum EMA Slope		0.2	0.4	0.8	0.8	1.0	-
EMA Slope Window Length i		30	20	20	40	10	-
Trailing Stop Order		false	false	false	false	false	-
Stop-Loss Distance		10	40	30	10	50	-
Take-Profit Distance		140	130	130	100	150	-
Number of Trades		531	159	6	1	3	-
Maximum Gain		475	174	205	460	111	-
Maximum Loss		-90	-78	-60	460	111	-
Profits Std.		70	75	123	0	0	-
Equity	**Minimum**	-568	0	-54	0	0	-
	Maximum	2323	2471	357	460	335	-
	Last	800	2088	191	460	335	-
Maximum Drawdown		151	97	46	0	0	-
Win Ratio		28	44	33	100	100	-

Table 25: Triple Exponential Moving Average Strategy Results I

Regime		Sideways Trend + Low Volatility			Sideways Trend + High Volatility		
		$Q_{0.33}$	$Q_{0.66}$	Q_1	$Q_{0.33}$	$Q_{0.66}$	Q_1
Short-Term		9	5	9	9	3	-
EMA Period	**Medium-Term**	14	23	14	14	5	-
	Long-Term	25	45	25	25	10	-
Minimum EMA Slope		0.4	1.0	0.4	0.4	0.2	-
EMA Slope Window Length i		20	30	40	40	30	-
Trailing Stop Orders		false	false	false	true	true	-
Stop-Loss Distance		20	10	30	10	30	-
Take-Profit Distance		150	150	150	20	150	-
Number of Trades		114	2	4	1	1	-
Maximum Gain		264	717	261	42	164	-
Maximum Loss		-75	-79	0	42	164	-
Profits Std.		79	398	106	0	0	-
Equity	**Minimum**	-627	0	0	0	0	-
	Maximum	916	717	347	42	164	-
	Last	561	637	347	42	164	-
Maximum Drawdown		2552	11	0	0	0	-
Win Ratio		35	50	100	100	100	-

Table 26: Triple Exponential Moving Average Strategy Results II

Table 27: Triple Exponential Moving Average Strategy Results III

Regime		Uptrend + Low Volatility			Uptrend + High Volatility		
		$Q_{0.33}$	$Q_{0.66}$	Q_1	$Q_{0.33}$	$Q_{0.66}$	Q_1
	Short-Term	7	7	7	7	-	-
EMA Period	**Medium-Term**	14	14	14	14	-	-
	Long-Term	40	40	40	40	-	-
Minimum EMA Slope		0.8	0.8	0.8	0.8	-	-
EMA Slope Window Length i		40	20	40	20	-	-
Trailing Stop Orders		false	false	false	false	-	-
Stop-Loss Distance		10	10	30	10	-	-
Take-Profit Distance		100	150	110	90	-	-
Number of Trades		2	43	2	1	-	-
Maximum Gain		484	875	170	488	-	-
Maximum Loss		484	-135	170	488	-	-
Profits Std.		0	290	0	0	-	-
	Minimum	0	-88	0	0	-	-
Equity	**Maximum**	968	2914	341	488	-	-
	Last	968	2075	341	488	-	-
Maximum Drawdown		0	58	0	0	-	-
Win Ratio		100	18	100	100	-	-

15.3.4 Bollinger Bands Strategy Backtest Results

Regime	Downtrend + Low Volatility			Downtrend + High Volatility		
	$Q_{0.33}$	$Q_{0.66}$	Q_1	$Q_{0.33}$	$Q_{0.66}$	Q_1
Bollinger Band Period	14	18	13	20	15	25
No. of Standard Deviations	2.5	3.0	1.5	2	1.5	1.5
Trailing Stop Orders	true	false	true	false	false	false
Stop-Loss Distance	10	10	10	10	10	30
Take-Profit Delta	11	9	15	-1	1	19
Number of Trades	4	15	8	2	1	1
Maximum Gain	84	109	97	191	111	80
Maximum Loss	-27	-63	-56	127	111	80
Profits Std.	44	50	65	32	0	0
Equity Minimum	0	-9	0	0	0	0
Equity Maximum	174	250	226	319	111	80
Equity Last	174	250	179	319	111	80
Maximum Drawdown	33	98	57	0	0	0
Win Ratio	75	53	50	100	100	100

Table 28: Bollinger Bands Strategy Results I

Regime	Sideways Trend + Low Volatility			Sideways Trend + High Volatility		
	$Q_{0.33}$	$Q_{0.66}$	Q_1	$Q_{0.33}$	$Q_{0.66}$	Q_1
Bollinger Band Period	15	23	24	-	23	24
No. of Standard Deviations	2.0	2.5	1.5	-	1.5	1.5
Trailing Stop Orders	false	false	false	-	false	true
Stop-Loss Distance	10	10	10	-	10	10
Take-Profit Delta	-3	15	5	-	1	5
Number of Trades	2	25	7	-	1	1
Maximum Gain	73	142	140	-	75	-31
Maximum Loss	57	-64	-51	-	75	-31
Profits Std.	7	52	67	-	0	0
Minimum	0	-64	0	-	0	-31
Equity Maximum	130	398	340	-	75	0
Last	130	269	340	-	75	-31
Maximum Drawdown	0	52	51	-	0	0
Win Ratio	100	40	71	-	100	0

Table 29: Bollinger Bands Strategy Results II

Regime	Uptrend + Low Volatility			Uptrend + High Volatility		
	$Q_{0.33}$	$Q_{0.66}$	Q_1	$Q_{0.33}$	$Q_{0.66}$	Q_1
Bollinger Band Period	19	15	13	-	16	-
No. of Standard Deviations	1.5	1.5	1.5	-	1.5	-
Trailing Stop Orders	false	false	false	-	false	-
Stop-Loss Distance	30	10	10	-	10	-
Take-Profit Delta	9	1	15	-	9	-
Number of Trades	1	8	196	-	1	-
Maximum Gain	121	114	96	-	107	-
Maximum Loss	121	-57	-62	-	107	-
Profits Std.	0	66	25	-	0	-
Equity Minimum	0	-57	0	-	0	-
Equity Maximum	121	253	521	-	107	-
Equity Last	121	253	128	-	107	-
Maximum Drawdown	0	59	90	-	0	-
Win Ratio	100	62	36	-	100	-

Table 30: Bollinger Bands Strategy Results III