Developing a trading strategy based on Bitcoin price predictions using Deep Learning

Peter Kocur

31 August 2024

Accessible on Github:
github.com/nepether/bitcoin-price-prediction-using-deep-learning

Table of contents:

1. Introduction 3

2. Literature review 6

3. Project Design 10

4. Implementation 19

5. Evaluation: 28

6. Conclusion: 40

7. References: 43

1. Introduction

1.1 Project Concept and Motivation

Bitcoin is a well-known digital and decentralized cryptocurrency with the biggest market cap in the world (CoinMarketCap, 2024). Its popularity helped attract more retail investors who previously weren't interested in financial markets or finance. Cryptocurrencies are known for high volatility. While this volatility presents the potential for high returns, which attracts novice traders, it also introduces significant risk, and potential gains are balanced with potential losses (Jiang, 2020). While other studies predict Bitcoin prices, there isn't a paper that includes approving Bitcoin's spot ETF from January 2024 (Hayes, 2024), which can lead to a different behavior as there is a new inflow of institutional money.

The primary concept of this project is to develop and evaluate deep learning models to predict the prices of Bitcoin and the other major cryptocurrency, Ethereum. The secondary goal is to evaluate whether there's a different behavior on the Bitcoin market from the approval of Bitcoin's spot ETF.
The tertiary goal is to analyze the correlation between Bitcoin and Ethereum markets and determine whether price movements in Bitcoin can inform trading decisions in other cryptocurrency markets.

As I recently embarked on a journey to become a profitable trader, my main motivation is to see how to leverage deep learning and other Data Science and Machine Learning techniques for price prediction and gain a competitive edge in the markets. This project aims to enhance my trading strategies and serves as a foundation for building an automated trading system in the future. This project is a great convergence of the selected **template from CM3015 Machine Learning and Neural Networks - 1: Deep learning on a public dataset** and my interest in trading and cryptocurrency markets.

By leveraging deep learning models, I hope to gain insights into the patterns and dynamics of cryptocurrency markets that are unthinkable for an individual. Thus, I can make informed trading decisions and mitigate potential risks.

1.2 Project Goals and Objectives

The project is structured around four main objectives:

1. **Develop and evaluate a deep learning model for predicting Bitcoin prices:** The first objective is to build a robust deep learning model that can accurately predict the price movements of Bitcoin. This involves collecting historical price data, preprocessing it, and experimenting with various deep learning architectures such as LSTM, GRU, and CNN. Based on the previous papers, I will compare these models to RFR (Chen 2023).
2. **Evaluate the prediction pre- and post-Bitcoin spot ETF approval:** Run the models on a smaller subset of data—from September 2023 to August 2024—and evaluate whether the predictions perform better compared to predictions created based on the longer timeframe.
3. **Extend the model to predict prices of other cryptocurrencies:** After developing and choosing the best-performing model for Bitcoin, the next step is to extend the same model to predict the prices of another major cryptocurrency - Ethereum.
4. **Analyze market correlation and develop and test trading strategy:** The final objective a is to use the predictions and develop a trading strategy to back test on the data for both Bictoin and Ethereum. Based on these predictions, I want to conclude if such predictions combined with trading strategies could be a helpful tool for traders, and how it performs vs. buy-and-hold strategies.

1.3 Importance and Relevance

This project's potential is to contribute to the field of cryptocurrency trading and give an early signal on how Bitcoin's spot ETF could change the market and its volatility. This paper combines the predictions and strategies to apply them to money management and trade management. Combined with accurate price prediction models, it can provide a competitive edge, enabling traders to make better decisions and incorporate this tool into their toolbox to give another insight into their trading setups. Understanding the correlation between different cryptocurrencies can help improve trading strategies and reduce risk in opening trades against this correlation.

Traditional trading indicators can't capture the complex, nonlinear relationships in huge data, such as financial data. For instance, moving averages and RSI are effective in identifying trends but are very limited in predictions due to their lagging nature. On the other hand, deep learning models like LSTM, GRU, and CNN are great at handling complicated patterns in data. LSTM and GRU, because they can remember information over long periods, work especially well for time series data with long-range dependencies, like predicting cryptocurrency price trends - making them ideal candidate for this project.

1.4 Project's justification

Bitcoin is the most traded cryptocurrency. Its price movements can influence the cryptocurrency market, and understanding Bitcoin’s price and its correlation to other coins can provide valuable insights. Predicting price movements can help traders achieve profitability while mitigating losses. The use of deep learning models can provide an edge, especially for retail investors without the knowledge or skills to build these models themselves.

1.5 Methodology

The project uses a structured approach, starting with data collection and preprocessing. I will gather historical price data for Bitcoin and Ethereum from CoinMarketCap: one set with a longer timeframe for comprehensive coverage and another from around September 2023, when rumors about Bitcoin’s spot ETF approval triggered a bull run. The data will be cleaned, normalized, and feature-engineered to include technical indicators like moving averages and RSI, providing more context for deep learning models.

The core involves developing and evaluating models such as LSTM, GRU, CNN and RFR. Models will be assessed using MAE, RMSE, and R-squared metrics. The best model for Bitcoin will be tested on Ethereum to evaluate generalizability. Pearson correlation and Granger causality tests will analyze market correlations, examining if Bitcoin can predict

other cryptocurrency prices. Insights will guide a trading strategy, which will be backtested.

1.6 Challenges and Mitigation Strategies

Data quality: Ensuring the accuracy and consistency of historical price data is crucial. Choosing a good data source is important, and coupled with data preprocessing steps like handling missing values and normalization.

Computational Constraints: Training deep learning models can be computationally intensive. If my local setup is insufficient, I will turn to cloud-based GPU providers.

2. Literature review

More traditional approaches to financial prediction have involved statistical methods such as moving averages and autoregressive models. However, with more access and, therefore, the popularity of machine learning, there has been increased focus on methods that can capture nonlinear relationships within financial data. In cryptocurrency price prediction, various machine learning and deep learning models have been explored to tackle the volatility and complexity of financial markets. I've chosen and evaluated five key papers, focusing on evaluating their methodologies and findings to influence my work and project. There's a general consensus in the papers that deep-learning models show big promise in price prediction. The papers focus mainly on three key deep-learning models and on evaluating random forest regression.

I will break down each paper with the summary and my critical evaluation.

"Predicting Future Cryptocurrency Prices Using Machine Learning Algorithms" (Saha, 2023) focuses on and explores various machine learning models to predict prices. The study uses historical price data from an unspecified source. He uses linear interpolation to approximate missing values and mitigate the risk of missing data affecting model performance. He's also building features from common trading indicators such as Simple

Movin Average (SMA), Relative Strength Index (RSI), and Moving Average Convergence Divergence (MACD).

The models tested in the paper are Quadratic Discriminant Analysis (QDA), K-Nearest Neighbour (KNN), Logit Model, Decision Tree, and Neural Networks. The paper concludes that while traditional models like QDA and KNN showed some predictive potential, they are very inferior to neural networks. QDA performed worse than a random chance, and it wasn't better for KNN. Neural networks implemented showed high accuracy (0.991) and low error with MSE (0.286) on a 1-hour price forecast, and it generally got better moving to higher timeframes (like 14-hour lookahead prediction). This nudged me to explore daily timeframes. This paper was especially good at introducing a lot of technical indicators in the research. However, it wasn't clear what data set was used even though they were open source. The data sample is only from 7 months, which doesn't cover one halving cycle - which is roughly four years long (Conway, 2024).

In the paper "Prediction of the Technology Company's Stock Price through the Deep Learning Method" (Wang, 2022), the focus is on predicting stock prices of eight major technology companies. In his intro, he covers the two types of stock analysis "...*fundamental analysis and technical analysis. Fundamental analysis evaluates a company's value by assessing its financial performance, business model, and economic environment, while technical analysis evaluates a stock's value through its past market data*." He explains that technical analysis is a method of predicting stock prices by looking at historical market data. This statement supports the idea contrasting with the more common opinion that markets can be predicted or timed. The study focuses on using the CNN-LSTM hybrid model to predict the adjusted close price of technology companies in the next trading day. This is an interesting narrative, as the prediction concerns not only the price the next day at any point in time on that day but also the adjusted closing price (Ganti, 2020). This is something that is not applicable to cryptocurrencies. The paper covers strengths, weaknesses, and comparative analysis of the hybrid CNN-LSTM model against single and double LSTM models. Double LSTM had the lowest average RMSE and MAE while having the worst prediction results. It also took the longest time to compute, so

this demotivated me from trying it out. Moreover, both CNN-LSTM hybrid and double LSTM had overfitting issues. Even though Autoregressive integrated moving average (ARIMA) is mentioned as a classic analysis model for time series data prediction, it's not used as a benchmark in the paper, which would be desirable. CNN-LSTM had the largest RMSE and MAE but showed accurate predictions before its trend deviated from real value. However, as the author mentioned, the dataset limitation of only 8th companies increased the probability of experiment error. All companies had different dates for their IPO, so the time period for analysis differed, which increased difficulties in comparing observations and prediction models. This is also applicable to my project, as each cryptocurrency had a different creation date - however, I can select the timeframe in which all cryptocurrencies were traded.

The next paper is also exploring LSTM and GRU models (Jiang, 2020), but in his paper, but it's also covering Multi-Layer Perceptro(MLP). The paper uses a one-minute bitcoin transaction from Kaggle's dataset aggregated to hourly intervals. This study provides a good overview of data preprocessing techniques such as min-max normalization and window-based normalization, as well as Minibatch, which is used to split large data into smaller batches to improve memory efficiency. These are good things to consider in my preprocessing pipeline. The use of ten-fold cross-validation ensured that the results were not based on overfitting but generalizable across different data splits, which is crucial for price prediction or ensuring the quality of results in general. The shortcomings of this study is using dated data, only covering the price developments to July 2018. It also wasn't clear from the study how to interpret the models and how to understand specific features across time periods to drive value for traders. Showcasing real-time testing would provide an actual value for retail investors on top of the scope of scientific papers.

What influenced my project design was the usage of Random Forest Regression (RFR) (Chen, 2023). In this paper, he demonstrated that Random Forest Regression can have superior performance over LTMS in terms of Root Mean Squared Error (RMSE), Mean Absolute Percentage Error (MAPE), and higher precision in predicting the price of Bitcoin on the next day - which also focuses of my paper. What is crucial to consider while using

Random Forest Regression is that the algorithm failed to predict prices outside of the training samples. This is especially important in the current development of the crypto market, where many currencies are exploring new all-time highs (ATH). The latest ATH of $75,830 was reached on March 14, 2024 (Edwards, 2024).

The paper provides a detailed analysis of the variables influencing Bitcoin prices. Paper examines factors such as the NASDAQ, DJI, S&P 500 indices. Next prices of other crypto currencies such as LiteCoing, Ethereum, Ripple, Dash, and more. Paper also looks into the commodities like Gold, Silver, and Copper. From the Bitcoin features, paper includes mean hash rate (Wade, 2023), block size (Caffyn, 2023), and more features more common in the rest of the paper such as price Open, Close, Low, High, and daily volume traded. Plotted on heatmap, the paper shows correlation between Bitcoin and other variables, but is not explaining the causation for this correlation. As an example, it demostrates that Bitcoin prices correlate with other cryptocurrencies, commodity prices, and stock market indexes, but fails to explain why. A knowledge of financial markets would be helpful here, as it's known that when US dollar is getting stronger, the riskier assets such as stocks, and cryptocurrencies are weakend, and vice-versa. Another example is positive correlation of Russian ruble exchange rate on Bitcoin. This, however, could be an outlier situation based on Russia entering the war and huge sanctions that were put on the country, weakening its currency, which forced people to hedge their money in assets like cryptocurrencies.

2.1 Summary of literature

Historical methods for predicting prices on financial instruments relied on statistical methods such as moving averages and autoregressive models. While forms of moving averages, such as Moving Average (MA) (Fernando, 2024) or Moving Average Convergence/Divergence (MACD) (Dolan, 2024), are still used for trading, these models alone can't capture all of the nonlinear relationships influencing financial markets and prices. Increased accessibility and popularity of machine learning and deep learning methods shifted the focus on leveraging such methods on financial markets. Four key papers were evaluated, focusing on their methodologies, findings, and evaluations to inform this project. The general consensus is that deep learning shows a big promise for

price prediction, especially in cryptocurrency markets where all data are recorded and immutable on a public ledger. The papers explore three key deep learning models - LSTM, GRU, and CNN- and evaluate the effectiveness of random forest regression and its shortcomings. However, all of the papers focused on scientific parts while neglecting the real-life usage of the outcomes and haven't provided enough explanation of why the features were and what their casualty is. Papers haven't been tested on real markets, and trading strategy wasn't a scope of the papers. Moreover, none of the literature accessible to me at the time of the writing critically evaluated the influence of the institution's money around Bitcoin's spot ETF approval. These are the areas where I hope I will be able to bring a unique value.

In the next section, I will delve into the project design, detailing the methodology and approach for this research.

3. Project Design

3.1 Introduction and Project Overview

This project focuses on developing and evaluating deep learning models for predicting cryptocurrency prices, specifically Bitcoin, and the price correlation with other cryptocurrencies - namely Ethereum. It uses **Template 1 from the CM3015 Machine Learning and Neural Networks module, which focuses on leveraging deep learning on a public dataset**. The primary objective is to create models that can predict daily prices on Bitcoin, evaluate these predictions, and analyze correlations between selected cryptocurrencies. Trading strategies will be defined and back-tested to evaluate their performance. One of the main advantages over traditional indicators is a prediction of the day's high price and comparing it with the previous day's close price. Comparing these two should give the user a signal if they should open a Long or Short position and also how to manage money, as the stop loss and the position size are derived from the difference between the predicted high and the previous close price. This is one of the most important areas for consistent profitability in trading.

3.2 Domain and Users

This project's domain is financial markets, particularly cryptocurrency markets. Users include retail traders, financial analysts, and algorithmic trading developers. Accurate price predictions would benefit these user personas by informing their trading decisions and strategies or providing a foundation for building an automatic trading system.

3.3 Key technologies and methods used in this project include:

1. **Long Short-Term Memory (LSTM) Networks:** LSTM networks are a type of recurrent neural network (RNN) designed to capture long-term dependencies in sequential data. They have been widely used in time series forecasting, including cryptocurrency prices, as outlined in the literature overview.

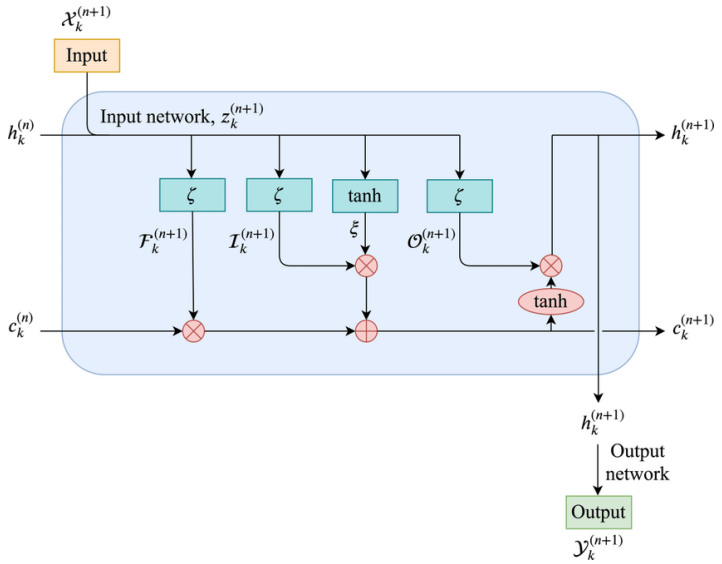

Fig. 1: Schematic representation of a typical LSTM network (Weiss & Buhmann, 2019).

2. **Gated Recurrent Units (GRU):** GRUs are a variation of RNNs that are simpler and more computationally efficient than LSTMs. They have also been applied to financial time series data with promising results. Papers indicate that GRUs, while less complex than LSTMs,

can provide comparable performance with reduced computational requirements, making them suitable for real-time prediction scenarios.

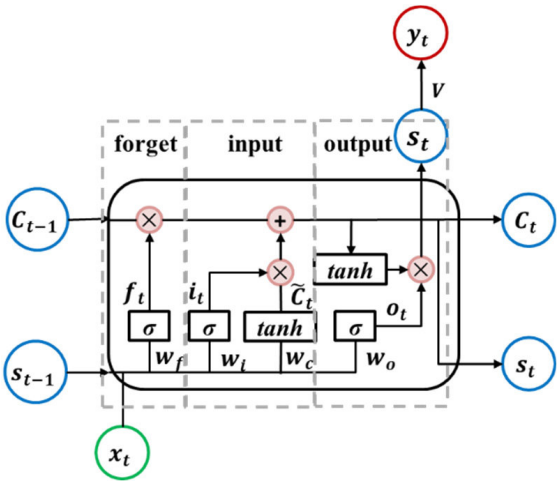

Fig. 2: Diagram of the gated recurrent unit (GRU) RNN (Bakarov, 2019).

3. **Convolutional Neural Networks (CNN):** Although traditionally used in image processing, CNNs have been adapted for time series analysis. They can capture local patterns in the data, which can be useful for identifying short-term trends in cryptocurrency prices.

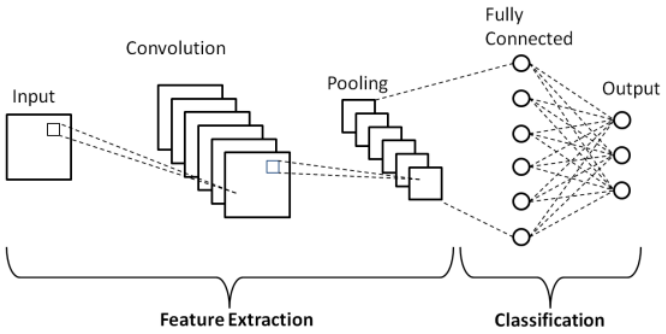

Fig. 3: Schematic diagram of a basic convolutional neural network (CNN) architecture (Wu, 2019).

4. **Random Forest Regression (RFR):** Random forest regression is an ensemble learning method that operates by constructing multiple decision trees during training and outputting the mean prediction of the individual trees. This model is known for its robustness to overfitting, high accuracy, and ability to handle large datasets with higher dimensionality. It has been effectively used in financial market predictions to capture complex interactions among features.

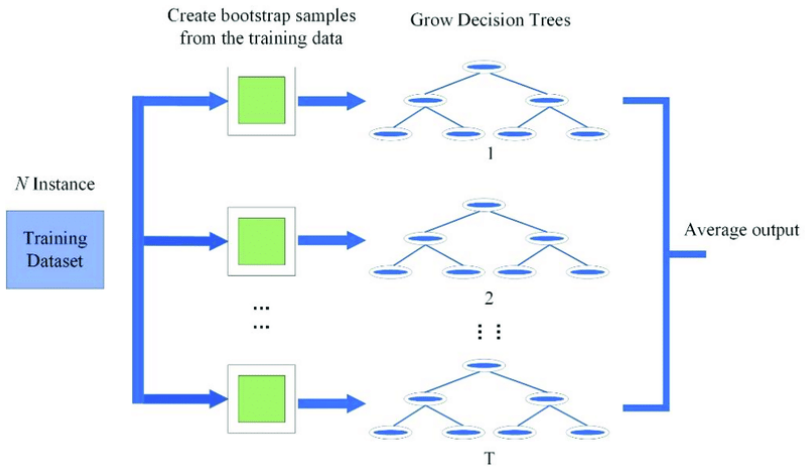

Fig. 4: The schematic diagram of random forest regression (RFR) (Zeng, 2019).

3.4 Evaluation Methods:

1. **Mean Absolute Error (MAE):** Measures the average magnitude of prediction errors without considering their direction. Root Mean Squared Error (RMSE): Provides a measure of the average magnitude of errors, giving higher weight to larger errors.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Fig. 5: Formula for Mean Absolute Error where y_i are actual values and $\hat{y}_i$ predicted.

2. **Root Mean Squared Error (RMSE):** Provides a measure of the average magnitude of errors, giving higher weight to larger errors. $\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$

Fig. 6: Formula for Root Mean Squared Error where y_i are actual values and $\hat{y}_i$ predicted.

3. **R-Squared (R^2)**: R^2 measures the proportion of the variance in the dependent variable that is predictable from the independent variables. It is useful for evaluating the goodness of fit of a model.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}i)^2}{\sum i = 1^n (y_i - \bar{y})^2}$$

Fig. 7: Formula for R-Squared Error where y_i are actual values and $\hat{y}_i$ predicted.

3.5 Trading strategies:

There are two main components of a profitable trading strategy. The first one is the Win Rate (Kenton, 2021), and the second is the Risk to Reward Ration (RRR) (Hayes, 2024). Because of these two components, the prediction can be right on 50% of the time (so the same odds as a coin toss) but must have a potential to win twice as much as risk.
Based on this logic, there are 3 strategies defined.

3.5.1 Strategy 1: Win Rate at least 50% and RRR 2:1

In this strategy, for every trade, the potential reward is twice the potential risk. So out of 100 trades, 50 trades are winners and 50 trades are losers.
Example: We risk $1 to make $2 over 100 trades.
The net profit would be:
Wins: 50 trades * $2 = $100
Losses: 50 trades * $1 = $50
Net Profit = $100 - $50 = $50

3.5.2 Strategy 2: Win at least 30% and RRR 3:1

In this strategy, for every trade, the potential reward is three times the potential risk. Out of 100 trades, 30 trades are winners and 70 trades are losers.

Example: We risk $1 to make $3 over 100 trades.

The net profit is:

Wins: 30 trades * $3 = $90

Losses: 70 trades * $1 = $70

Net Profit = $90 - $70 = $20

3.5.3 Strategy 3: Win Rate at least 60% and RRR 1.5:1

In this strategy, for every trade, the potential reward is 1.5 times the potential risk. Out of 100 trades, 60 trades are winners and 40 trades are losers.

Example: We risk $1 to make $1.5. over 100 trades.

The net profit is:

Wins: 60 trades * $1.5 = $90

Losses: 40 trades * $1 = $40

Net Profit = $90 - $40 = $50

Strategy flow chart

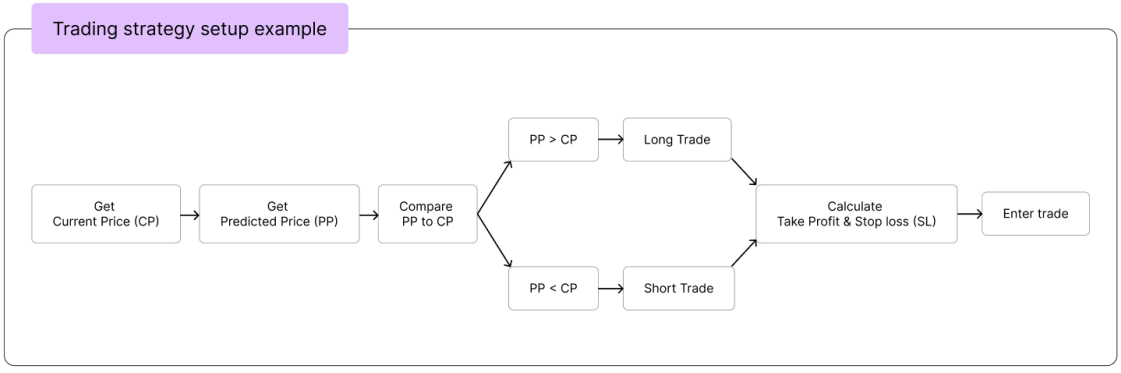

Fig. 8: Trading strategy flow chart.

3.6 Structure of the Project

The project follows several key steps:

1. **Data Collection:** Gathering historical price data for Bitcoin, and Ethereum from CoinMarketCap.
2. **Data Preprocessing**: Cleaning and normalizing the data, handling missing values, and generating features for technical indicators.
3. **Model Development**: Building and implementing LSTM, GRU, CNN, and RFR models.
4. **Model Training and Evaluation:** Training the models on historical data and evaluating their performance using MAE, RMSE, and R^2.
5. **Correlation Analysis:** Analyzing the correlations between Bitcoin, and Ethereum.
6. **Trading Strategy Development and Testing**: Developing and backtesting trading strategies based on model predictions and risk and money management industry standards.

3.7 Diagram of the project

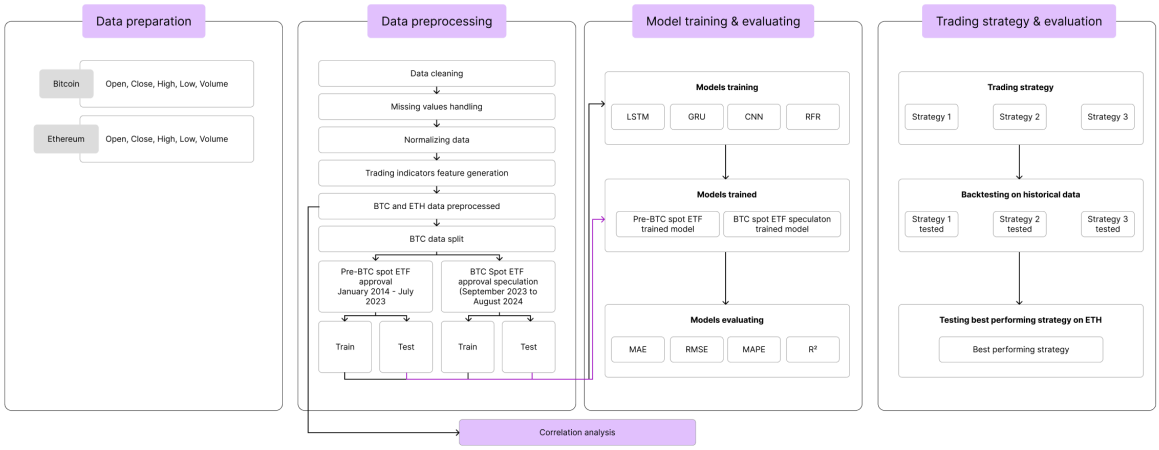

Fig. 9: Project's diagram

3.8 Key hypothesis

The primary hypothesis is that deep learning models (LSTM, GRU, and CNN) or RFR can predict daily cryptocurrency prices with an accuracy high enough that when trading strategies are applied with tested stop losses, we will find a profitable trading strategy on historical data that could be further tested in the live market.

3.9 Evaluation plan

The evaluation plan consists of two parts.

Model evaluation:

1. **Model Performance:** Assessing models using MAE, RMSE, and R^2.
2. **Backtesting:** Testing the trading strategy on historical data to evaluate its effectiveness.
3. **Correlation Analysis:** To analyze relationships between cryptocurrencies.

Trading strategy evaluation:

Each of the trading strategies outlined above will be backtested using historical data. For each strategy. The main goal of evaluating trading strategies is to determine whether any of them, combined with models for price prediction, can achieve profitability.

3.10 Project's work plan

This project will be built iteratively. It will start by collecting data, evaluating it, and then creating datasets for each of the cryptocurrencies. The next step will be to train and evaluate models using the abovementioned methods. This is the first loop to iterate over to fine-tune the models and get the best results.

The next step is to create a trading strategy that will be backtested on historical data. This is another loop where the strategies for trading will be tested to find the best performer, defined as a combination of the model and the trading strategy that gets to profitability or close to it.

Here's the textual breakdown of the steps involved. These are purposefully not number, as this process is iterative.

- Collect/Create all data assets
- Data preprocessing
- Evaluate data
- Perform correlation analysis on Ethereum and Bitcoin
- Implement LSTM model
- Implement GRU model
- Implement CNN model
- Implement Random Forest Regression model
- Refine models
- Train models
- Evaluate models
- Develop trading strategy
- Backtest trading strategy
- Prepare a final document with the findings

3.11 Visual representation of the plan and milestones using the Gantt chart

The second part of the project starts on the 10th of June 2024 and ends on the 9th of September 2024 with the submission of the final report. Therefore, the project is divided into 12 weeks.

Based on my experience, it's better to keep the Gantt chart overview on a high level and only plot the milestones - leaving room for experimentation and especially iteration. Tasks defined rigorously into the fine details might create a delivery bias and limit experimentation.

There are 8 key milestones for my project, as plotted on the Gantt chart. The granular part is left for the respective week plan, which can be adjusted according to progress.

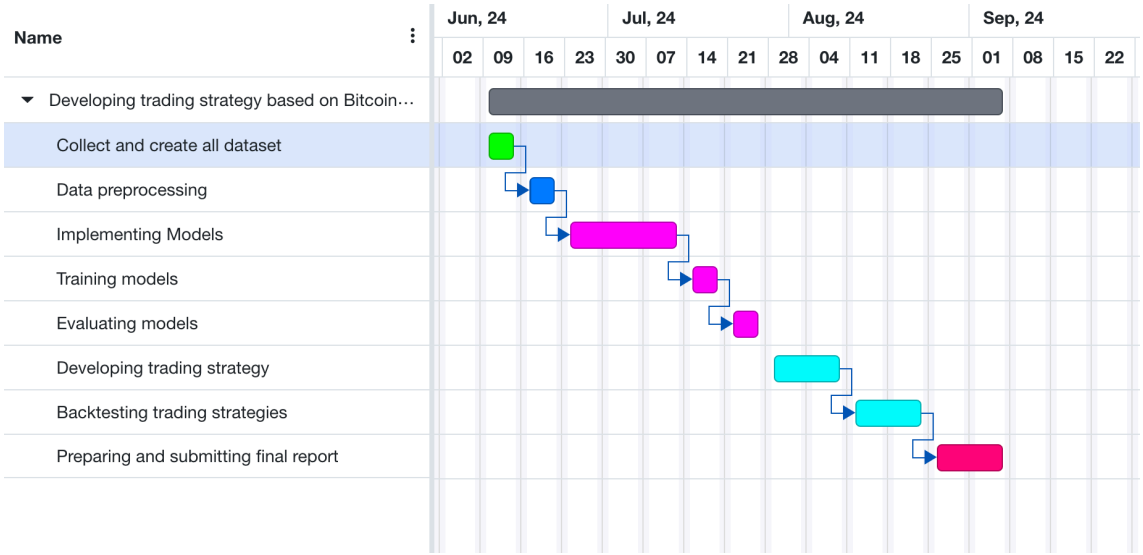

Fig. 10: Project milestones and plan using Gantt chart

4. Implementation

4.1 Introduction

This chapter presents a detailed description of the project's implementation, which consists two major parts:

1. **Developing and Implementing Predictive Models:** This section focuses on the development and implementation three deep learning and one machine learning model, namely Long Short-Term Memory (LSTM), Gated Recurrent Units (GRU), Convolutional Neural Networks (CNN), and a Random Forest Regressor. These models are used to predict Bitcoin prices based on historical data obtained from CoinMarketCap. The models are evaluated based on their accuracy and performance. The best-performing model was used together with trading strategies with the aim of maximizing the trader's profits.
2. **Integrating Predictions with Trading Strategies:** The second part of the project combines the predictive outputs from the best-performing model with practical trading strategies. This involves applying concepts such as money management and position sizing to individual trades, aiming to maximize profitability. The chapter explores how these predictions are used to inform buy and sell decisions and how different strategies

can performed in terms of financial outcomes, win rate, and max draw down.

4.2 Data Collection and Preprocessing

The initial involves collecting historical price data for Bitcoin. The dataset includes daily prices, including open, close, high, low prices, and trading volume. This data is sourced from CoinMarketCap, covering September 2014 to August 2023.

Although from the UI of CoinMarketCap it's implied that the user can download data for the timeframe defined by them, it wasn't true in my case. I could only get around ~399 days' worth of data when I checked the data frame.

4.2.1 Data Loading

I then decided to collect data every year, starting from September 2018 to August 2023. I chose the time frame up to August 2023, as there were lot's of speculations floating around approval of Bitcoin's spot ETF, which started a bull market rally in September 2023 - and I will use the data from September 2023 to August 2024 for my secondary objective of identifying if post-ETF market dynamics changed.

The data was loaded from multiple CSV files corresponding to different time periods. The CSV files were concatenated into a single DataFrame for the pre-ETF data and another for the post-ETF data.

Load separate pre-ETF data and merge them to one CSV

```
#load csv files to array
csv_files = [
    './jupyter notebook/data/bitcoin-test-data-1-sep2022-aug2023.csv',
    './jupyter notebook/data/bitcoin-test-data-2-sep2021-aug2022.csv',
    './jupyter notebook/data/bitcoin-test-data-3-sep2020-aug2021.csv',
    './jupyter notebook/data/bitcoin-test-data-4-sep2019-aug2020.csv',
    './jupyter notebook/data/bitcoin-test-data-5-sep2018-aug2019.csv',
    './jupyter notebook/data/bitcoin-test-data-5-sep2017-aug2018.csv',
    './jupyter notebook/data/bitcoin-test-data-5-sep2016-aug2017.csv',
    './jupyter notebook/data/bitcoin-test-data-5-sep2015-aug2016.csv',
    './jupyter notebook/data/bitcoin-test-data-5-sep2014-aug2015.csv',
]

# Read and concatenate the CSV files
df_list = [pd.read_csv(file, delimiter=';') for file in csv_files]
merged_df = pd.concat(df_list, ignore_index=True)

# Save the merged DataFrame to a new CSV file
merged_df.to_csv('./jupyter notebook/data/bitcoin-pre-etf-data.csv', index=False)
```

Fig. 11: Merging multiple data sources into one dataframe

4.2.2 Data Cleaning

After I merged CSV files into one, I loaded it into one workable data frame. I kept the copy of the data frame, because I normalized the numerical data for model training. While iterating on this project, I learned that for better interpretability of trading strategies, denormalized data work better to tell the story.

Converting columns to date time and numeric: To prepare data set for model training I had to update data types for time columns to be datetime and numerical columns to be numerical.

Normalization: The features are normalized using Min-Max normalization to scale them within the range of 0 to 1, which should help speed up the training process and improve the model performance.

```
# Load the merged CSV file with the correct delimiter
pre_etf_data_path = './jupyter notebook/data/bitcoin-pre-etf-data.csv'
btc_pre_etf_data = pd.read_csv(pre_etf_data_path, delimiter=';', skipinitialspace=True)

# Split the single column into multiple columns
btc_pre_etf_data = btc_pre_etf_data['timeOpen,timeClose,timeHigh,timeLow,name,open,high,low,close,volume,marketCap,timestamp'].str.split(','

# Rename the columns correctly
btc_pre_etf_data.columns = [
    'timeOpen', 'timeClose', 'timeHigh', 'timeLow', 'name',
    'open', 'high', 'low', 'close', 'volume', 'marketCap', 'timestamp'
]

# Convert time columns to datetime
btc_pre_etf_data['timeOpen'] = pd.to_datetime(btc_pre_etf_data['timeOpen'], errors='coerce')
btc_pre_etf_data['timeClose'] = pd.to_datetime(btc_pre_etf_data['timeClose'], errors='coerce')
btc_pre_etf_data['timeHigh'] = pd.to_datetime(btc_pre_etf_data['timeHigh'], errors='coerce')
btc_pre_etf_data['timeLow'] = pd.to_datetime(btc_pre_etf_data['timeLow'], errors='coerce')
btc_pre_etf_data['timestamp'] = pd.to_datetime(btc_pre_etf_data['timestamp'], errors='coerce')

# Convert relevant columns to numeric, coercing errors to NaN
numeric_columns = ['open', 'high', 'low', 'close', 'volume', 'marketCap']
btc_pre_etf_data[numeric_columns] = btc_pre_etf_data[numeric_columns].apply(pd.to_numeric, errors='coerce')

# Make a copy of the DataFrame for trading strategy use (denormalized data)
btc_pre_etf_data_denormalized = btc_pre_etf_data.copy()

# Data normalization for numeric columns
scaler = MinMaxScaler()
btc_pre_etf_data[numeric_columns] = scaler.fit_transform(btc_pre_etf_data[numeric_columns])

# Display the first few rows of the processed DataFrame
btc_pre_etf_data.head()
```

	timeOpen	timeClose	timeHigh	timeLow	name	open	high	low	close	volume	marketCap	
0	2023-08-31 00:00:00+00:00	2023-08-31 23:59:59.999000+00:00	2023-08-31 11:43:00+00:00	2023-08-31 21:09:00+00:00	2781	0.402611	0.397276	0.386365	0.382161	0.057485	0.394937	23:59:59.9
1	2023-08-30 00:00:00+00:00	2023-08-30 23:59:59.999000+00:00	2023-08-30 00:21:00+00:00	2023-08-30 15:04:00+00:00	2781	0.408906	0.401710	0.406245	0.402429	0.046551	0.415820	23:59:59.9
2	2023-08-29 00:00:00+00:00	2023-08-29 23:59:59.999000+00:00	2023-08-29 16:28:00+00:00	2023-08-29 10:39:00+00:00	2781	0.384808	0.406510	0.388777	0.408812	0.083663	0.422386	23:59:59.9
3	2023-08-28 00:00:00+00:00	2023-08-28 23:59:59.999000+00:00	2023-08-28 12:29:00+00:00	2023-08-28 07:46:00+00:00	2781	0.384617	0.378939	0.388293	0.384753	0.031334	0.397555	23:59:59.9
4	2023-08-27 00:00:00+00:00	2023-08-27 23:59:59.999000+00:00	2023-08-27 17:02:00+00:00	2023-08-27 01:43:00+00:00	2781	0.383409	0.378455	0.389569	0.384509	0.019683	0.397289	23:59:59.9

Fig. 12: Updating dataframe with needed data types and storing denormalized data frame.

Handling missing values: Missing values in the dataset were handled using forward and backward filling methods, particularly for newly created technical indicators like SMA (Simple Moving Average) and RSI (Relative Strength Index).

```
# Forward and backward fill NaN values in SMA and RSI
btc_pre_etf_data['SMA'] = btc_pre_etf_data['SMA'].ffill().bfill()
btc_pre_etf_data['RSI'] = btc_pre_etf_data['RSI'].ffill().bfill()

btc_etf_data['SMA'] = btc_etf_data['SMA'].ffill().bfill()
btc_etf_data['RSI'] = btc_etf_data['RSI'].ffill().bfill()

# Ensure no NaN values are present after filling
print("Missing values after fill:\n", btc_pre_etf_data.isnull().sum())
print("Missing values:\n", btc_etf_data.isnull().sum())
```

Fig. 13: Handling missing values with forward and backward fill

4.2.3 Feature Engineering

I developed two technical indicators for this prototype. Moving Average (MA) and Relative Strength Index (RSI), which were added as new columns in the dataset to help improve deep learning models.

```
# Add technical indicators
def calculate_sma(data, window):
    return data.rolling(window=window).mean()

def calculate_rsi(data, window):
    delta = data.diff(1)
    gain = delta.where(delta > 0, 0)
    loss = -delta.where(delta < 0, 0)
    avg_gain = gain.rolling(window=window).mean()
    avg_loss = loss.rolling(window=window).mean()
    rs = avg_gain / avg_loss
    rsi = 100 - (100 / (1 + rs))
    return rsi
#pre-ETF data
btc_pre_etf_data['SMA'] = calculate_sma(btc_pre_etf_data['close'], 20)
btc_pre_etf_data['RSI'] = calculate_rsi(btc_pre_etf_data['close'], 14)

#post-ETF data
btc_etf_data['SMA'] = calculate_sma(btc_etf_data['close'], 20)
btc_etf_data['RSI'] = calculate_rsi(btc_etf_data['close'], 14)
```

Fig. 14: Feature engineering of technical indicators

4.2.4 Data Preparation

I used a standard 80/20 split for training and testing data. The dataset is split into the features, which are different prices per day, and technical indicators. The target variable is high price, because we are not interested in the closing price that day. We are primarily

interested in the predicted price being hit at any point in time during that day. This is where I profit target order will be filled.

```
# Prepare the dataset for training
X = btc_pre_etf_data[['open', 'close', 'low', 'volume', 'marketCap']]
y = btc_pre_etf_data['high']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Fig. 15: Data preparation for testing

4.3 Model Development

Four models were developed to predict Bitcoin prices: LSTM, GRU, CNN, and Random Forest Regressor. The choice of these models was driven by their ability to capture both temporal dependencies and non-linear patterns in the data.

4.3.1 LSTM Model Set Up:

A. 50 neurons per layer with two layers in total
B. 10 epochs to minimize the risk of overfitting
C. The dense layer is set to 1, as we are only interested in predicting a single value the
D. I chose Adam, a widely used and common optimizer for the optimizer.
E. 32 for batch size is again a common choice that balances training speed and stability of the gradient descent process.

```
# Initialize the LSTM model
lstm_model = Sequential()

# Add an Input layer to specify input shape
lstm_model.add(Input(shape=(X_train_lstm.shape[1], 1)))

# Add the first LSTM layer
lstm_model.add(LSTM(units=50, return_sequences=True))

# Add a second LSTM layer
lstm_model.add(LSTM(units=50))

# Add a Dense layer with 1 unit for the output (predicted high price)
lstm_model.add(Dense(1))

# Compile the LSTM model using Adam optimizer and mean squared error loss
lstm_model.compile(optimizer='adam', loss='mean_squared_error')

# Train the LSTM model on the training data
lstm_model.fit(X_train_lstm, y_train, epochs=10, batch_size=32)
```

```
Epoch 1/10
82/82 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.0405
Epoch 2/10
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 1.3398e-04
Epoch 3/10
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - loss: 6.2947e-05
Epoch 4/10
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 4.0803e-05
Epoch 5/10
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 3.7930e-05
Epoch 6/10
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 4.9848e-05
Epoch 7/10
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 6.7485e-05
Epoch 8/10
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 4.4726e-05
Epoch 9/10
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 4.4660e-05
Epoch 10/10
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 4.1255e-05
```

Fig. 16: LSTM Model training

4.3.2 GRU Model Set Up:

A. 50 neurons per layer with two layers in total
B. 10 epochs to minimize the risk of overfitting
C. The dense layer is set to 1, as we are only interested in predicting a single value the
D. I chose Adam, a widely used and common optimizer for the optimizer.
E. 32 for batch size is again a common choice that balances training speed and stability of the gradient descent process.

```
gru_model = Sequential()

# Add an Input layer to specify input shape
gru_model.add(Input(shape=(X_train_gru.shape[1], 1)))

# Add the first GRU layer
gru_model.add(GRU(units=50, return_sequences=True))

# Add a second GRU layer
gru_model.add(GRU(units=50))

# Add a Dense layer with 1 unit for the output (predicted high price)
gru_model.add(Dense(1))

# Compile the GRU model using Adam optimizer and mean squared error loss
gru_model.compile(optimizer='adam', loss='mean_squared_error')

# Train the GRU model on the training data
gru_model.fit(X_train_gru, y_train, epochs=10, batch_size=32)
```

```
Epoch 1/10
82/82 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - loss: 0.0166
Epoch 2/10
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 1.4497e-04
Epoch 3/10
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 1.0624e-04
Epoch 4/10
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 1.3353e-04
Epoch 5/10
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 1.0909e-04
Epoch 6/10
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 9.8183e-05
Epoch 7/10
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 9.7467e-05
Epoch 8/10
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 1.1350e-04
Epoch 9/10
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 1.0776e-04
Epoch 10/10
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 1.1606e-04
<keras.src.callbacks.history.History at 0x32922e370>
```

Fig. 17: GRU Model training

4.3.3 CNN Model Set Up:

A. 64 filters with a kernel size of 2: To effectively capture local dependencies in the time series data.
B. MaxPooling layer: Added to reduce the dimensionality of the feature maps and prevent overfitting.
C. Dense layer with 50 units: To learn complex features from the extracted patterns.
D. Dense output layer set to 1: As we are predicting a single value, the highest price.
E. Adam optimizer: Chosen for its effectiveness and common usage in deep learning models.
F. Batch size of 32: A common choice to balance training speed and the stability of the gradient descent process.

```
# Initialize the CNN model
cnn_model = Sequential()

# Add an Input layer to specify input shape
cnn_model.add(Input(shape=(X_train_cnn.shape[1], 1)))

# Add the first Convolutional layer with 64 filters and kernel size of 2
cnn_model.add(Conv1D(filters=64, kernel_size=2, activation='relu'))

# Add a MaxPooling layer to downsample the input
cnn_model.add(MaxPooling1D(pool_size=2))

# Flatten the output before feeding it into the Dense layer
cnn_model.add(Flatten())

# Add a Dense layer with 50 units
cnn_model.add(Dense(50, activation='relu'))

# Add a Dense output layer with 1 unit (for predicting the target variable)
cnn_model.add(Dense(1))

# Compile the CNN model using Adam optimizer and mean squared error loss
cnn_model.compile(optimizer='adam', loss='mean_squared_error')

# Train the CNN model on the training data
cnn_model.fit(X_train_cnn, y_train, epochs=10, batch_size=32)
```

```
Epoch 1/10
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 371us/step - loss: 0.0239
Epoch 2/10
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 297us/step - loss: 6.7032e-05
Epoch 3/10
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 292us/step - loss: 8.1677e-05
Epoch 4/10
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 287us/step - loss: 4.4998e-05
Epoch 5/10
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 284us/step - loss: 5.8411e-05
Epoch 6/10
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 291us/step - loss: 2.9934e-05
Epoch 7/10
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 294us/step - loss: 2.9042e-05
Epoch 8/10
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 297us/step - loss: 3.5406e-05
Epoch 9/10
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 292us/step - loss: 4.8992e-05
Epoch 10/10
82/82 ━━━━━━━━━━━━━━━━━━━━ 0s 294us/step - loss: 2.6124e-05
<keras.src.callbacks.history.History at 0x32dfc0f40>
```

Fig. 18: CNN Model training

4.3.4 RFR Model Set Up:

100 estimators (trees) have been proven across many datasets to provide a good performance. Therefore it's a common default choice which I used for this prototype as well.

```
# Initialize the Random Forest Regressor model
rfr_model = RandomForestRegressor(n_estimators=100, max_depth=None, random_state=42, criterion='squared_error')

# Train the Random Forest Regressor on the training data
rfr_model.fit(X_train, y_train)
```

```
RandomForestRegressor
RandomForestRegressor(random_state=42)
```

Fig. 19: RFR Model training

4.4. Trading Strategies

To effectively use the predictions generated by the winning model, I developed and coded a trading strategy that simulates trading activities based on different risk-reward ratios (RRR). The goal of these simulations is to evaluate the profitability, win rate, and risk of the three trading strategies when applied to the predicted Bitcoin price movements.

4.4.1 Trading Strategy Framework

Parameters used for trading strategies:

- **Starting capital:** $10,000
- **Max risk per trade:** 1% of account balance. This means we can only risk 1% of the current capital for one trade. This also dictates the position size we can take.
- **Max position size:** 1,000 units. Ensuring that no trade exceeds this limit to mimic real trading conditions and manage risk exposure effectively. The term real might be exagurration, as there aren't many retail investors that could open positions with 1.000 bitcoins, nor could these positions be filled without spread. However, for the purpose of this project we will also include institutional conditions where positions like these exists.
- **RRR strategies:** The strategy was tested with three different RRR values (1.5:1, 2:1, and 3:1). These ratios determine the potential reward for every dollar risked, influencing the take-profit and stop-loss levels for each trade.

4.4.2 Strategy Implementation and Simulation

The strategy was implemented by reversing the data order to simulate trading from the past to the present. This is also useful, when we will compare it with buy-and-hold strategy. For each day in the dataset, the strategy checks if a trade should be opened based on the predicted high price for the next day and the last day's close price. Trades can either be “LONG” (buying in anticipation of a price increase) or “SHORT” (selling in anticipation of a price decrease).

Entry and Exit Conditions:

- **LONG Trade:** A trade is opened if the predicted high price for the next day is higher than the previous closing price. The take-profit is set to the predicted high, and the stop-loss is calculated based on the RRR.
- **SHORT Trade:** If the predicted high is lower than the previous close, a short trade is opened. The take-profit is set to the predicted high, and the stop-loss is calculated based on the RRR inversely.
- **Risk Management:** The strategy calculates the position size based on the maximum allowable risk (1% of the account balance) and the difference between the entry price and stop-loss price. This ensures that no single trade risks more than the specified percentage of the total capital. The position size is also capped to not be bigger than 1,000 units.
- **Trade Simulation:** For each trade, the simulation checks whether the take-profit or stop-loss is hit within the day. The account balance is adjusted accordingly, and each trade’s details (entry and exit prices, profit or loss, new balance) are recorded. If the take-profit or stop-loss haven't been hit within the day, the simulations takes the closing price of that day and calculate profit/loss for that day.

```
# Simulate the outcome of the trade
if trade_direction == 'LONG':
    if data['high'].iloc[i] >= take_profit_price:
        profit = position_size * (take_profit_price - entry_price)
        exit_price = take_profit_price
    elif data['low'].iloc[i] <= stop_loss_price:
        profit = -risk_amount
        exit_price = stop_loss_price
    else:
        profit = position_size * (data['close'].iloc[i] - entry_price)
        exit_price = data['close'].iloc[i]
else:  # SHORT
    if data['low'].iloc[i] <= take_profit_price:
        profit = position_size * (entry_price - take_profit_price)
        exit_price = take_profit_price
    elif data['high'].iloc[i] >= stop_loss_price:
        profit = -risk_amount
        exit_price = stop_loss_price
    else:
        profit = position_size * (entry_price - data['close'].iloc[i])
        exit_price = data['close'].iloc[i]

account_balance += profit
max_balance = max(max_balance, account_balance)
```

Fig. 20: Trading simulation implementation

4.4.3 Buy-and-Sell Strategy as a Benchmark

To provide a benchmark for evaluating the performance of the trading strategies developed based on predicted price movements, a buy-and-hold strategy was created. The buy-and-hold strategy serves as a simple, passive investment approach where an investor buys an asset and holds it over a long period, ignoring the short/mid-term market fluctuations. This strategy is often used as a baseline in financial analysis to compare the effectiveness of more complex trading strategies. In the cryptocurrency world, these people are usually referred to as HODLers. (Hold On your Dear Life).

4.4.4 Implementation of Buy-and-Hold Strategy

The buy-and-hold strategy is implemented by purchasing Bitcoin with the entire starting balance at the beginning of the investment period and holding it until the end of the period. The final balance is calculated based on the closing price of Bitcoin at the end of the period:

Initial Investment: The entire starting balance ($10,000) is used to buy Bitcoin at the closing price on the first day of the dataset.

Final Value: At the end of the period, the value of the investment is determined by the number of bitcoins held multiplied by the closing price on the last day of the dataset.

5. Evaluation:

5.1. Models Evaluation

All models were evaluated using three methods, namely Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and Root-Squared (R^2).

5.1.1 LSTM Model Evaluation:

MAE (0.0048): The model's average prediction error is very low, indicating high accuracy in predicting actual values.

RMSE (0.0067): The model's error magnitude is small, suggesting precise predictions with minimal large deviations.

R-squared (0.9992): The model explains 99.92% of the variance in the target variable, indicating an excellent fit to the data.

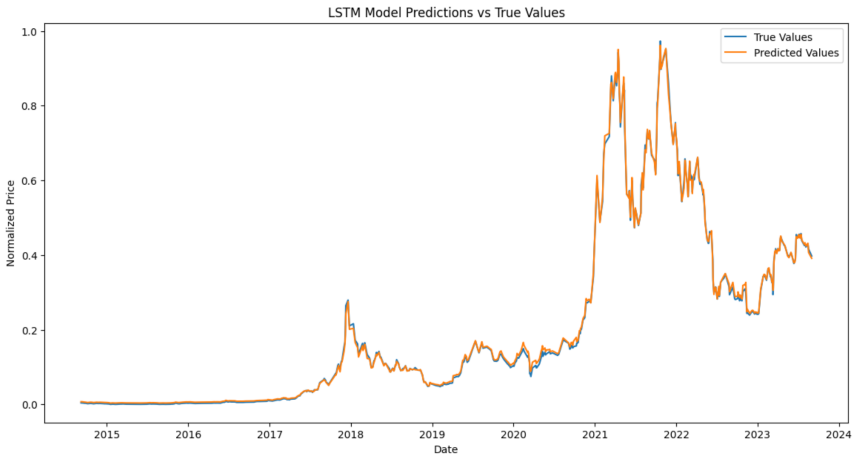

Fig. 21: LSTM model performance

5.1.2 GRU Model Evaluation:

Similar performance to LSTM model with worse **MAE (0.0058) , RMSE (0.0087)**,but improved **R-squared (0.9985)**

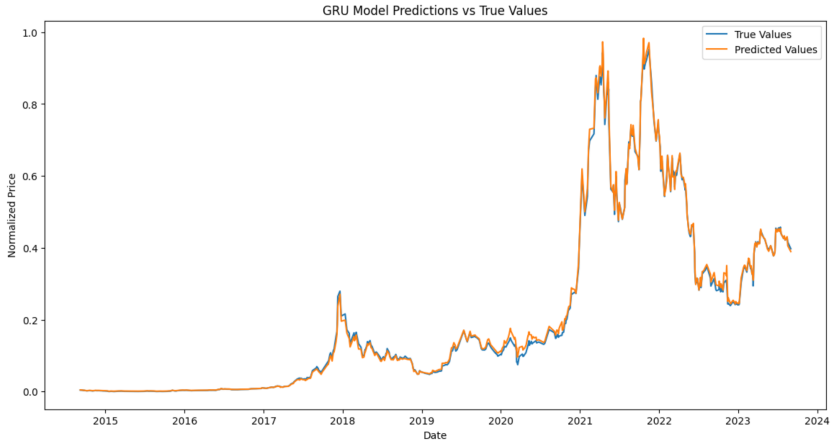

Fig. 22: GRU model performance

5.1.3 CNN Model Evaluation:

MAE (0.0036): The model's average prediction error is low, suggesting reasonably high accuracy in predicting actual values.

RMSE (0.0060): The model's error magnitude remains small, indicating precise predictions with minimal significant deviations.

R-squared (0.9994): The model explains 99.94% of the variance in the target variable, demonstrating an excellent fit to the data.

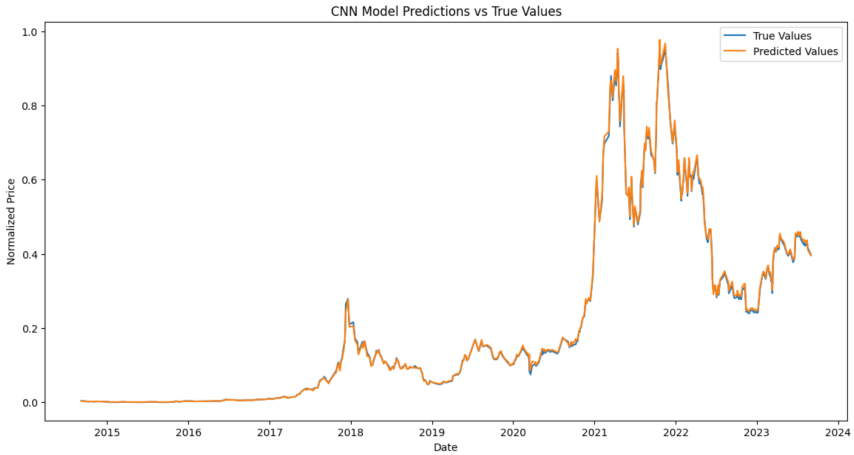

Fig. 23: CNN model performance

5.1.4 RFR Model Evaluation:

MAE (0.0022): The model achieved the lowest average prediction error, indicating the best accuracy in predicting actual values compared to deep learning models.

RMSE (0.0054): The RMSE for the RFR model is also the lowest among all models, demostrating minimal deviations between the predicted and actual values. This further confirms the model's high level of precision.

R-squared (0.9995): The model explains 99.95% of the variance in the target variable, which is the highest R-squared value among all models tested.

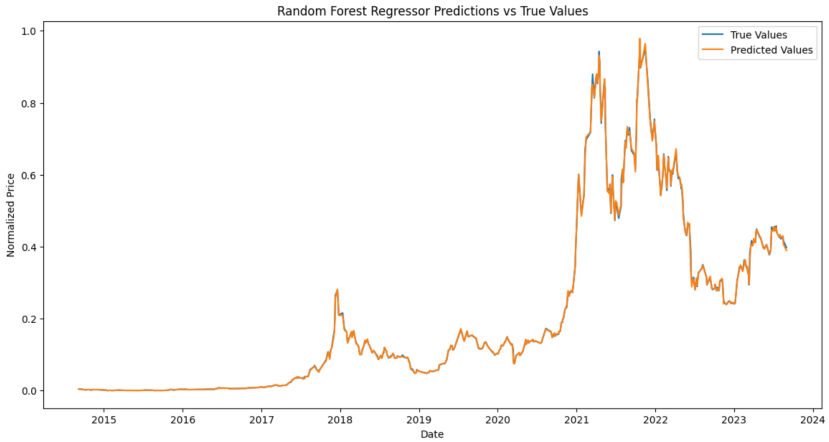

Fig. 24: RFR model performance

5.1.5 Model Comparison:

All of the models performed well, however RFR model had the edge in my approach. To better visualize how these models compare against each other I've plotted a line chart, to spot where the individual models may have the biggest errors. This is especially interesting within the scope of trading. By observation, we can see if the models failed during unpredictable market rallies.

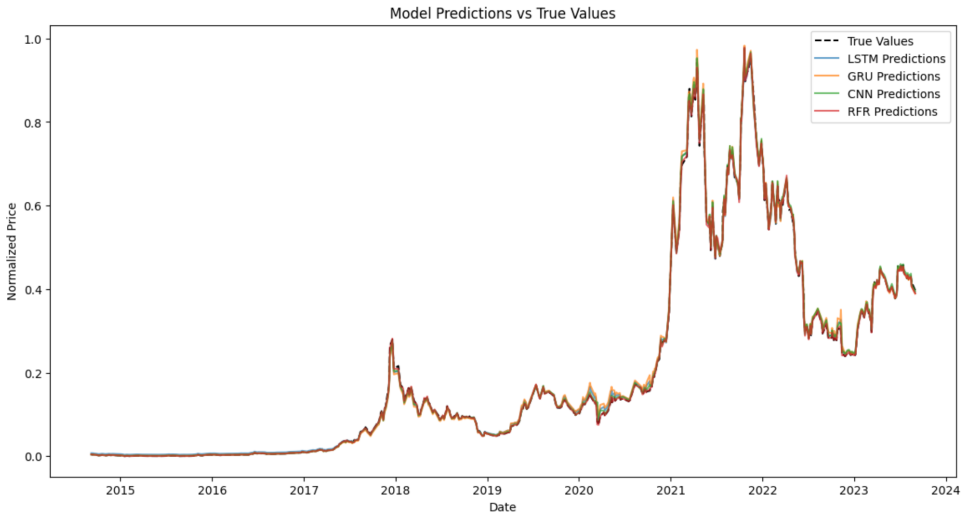

Fig. 25: Models' performance versus true values over ten years

The entire time frame was not sufficient to understand models' performance, so I selected one-year time frame to inspect the models more closely.

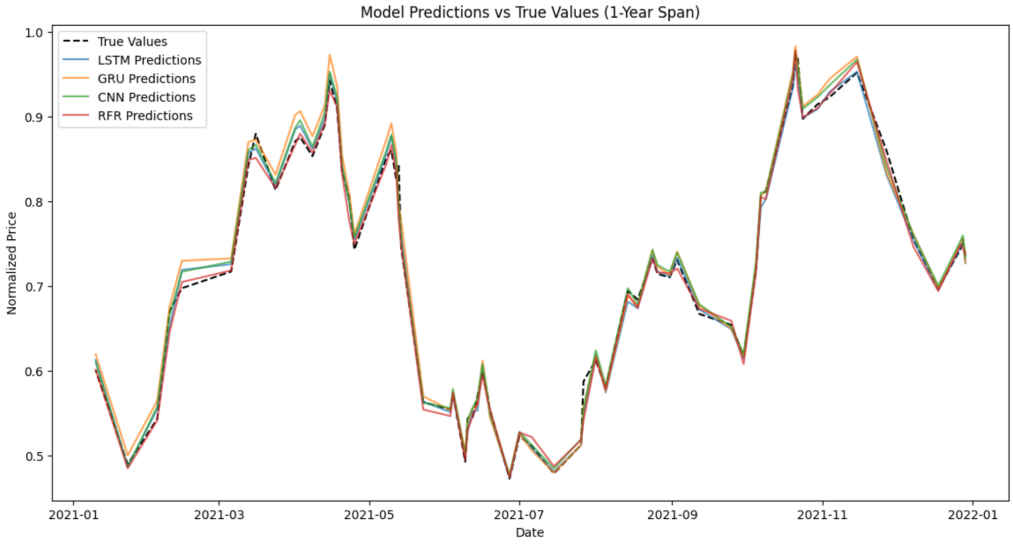

Fig. 22: Models' performance versus true values over one year

By observation, all models predict higher high in rapid market movements, but generality catches the trends and direction of the market impressively well. This is especially valuable in our trading strategies.

5.1.6 Model Errors Comparison:

To further understand each model's performance, we compared the distribution of their prediction errors using a box plot and an error distribution histogram.

Box Plot of Errors: By observation of the plot, the Random Forest Regressor (RFR) model has the smallest spread and lower median error, indicating more consistent and reliable predictions compared to the other models, which is aligned with the evaluation methods.
Error Distribution Histogram: In this plot, the RFR model again shows a relatively narrow and symmetrical error distribution, which represents smaller bias with a one outlier above 0.06 that might make the model a subject to further investigation.

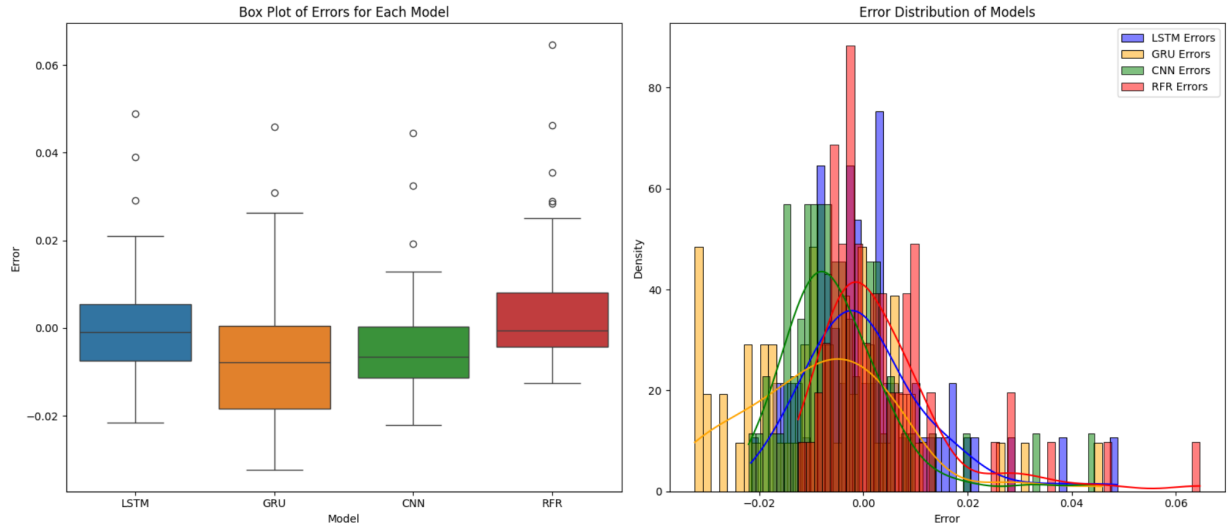

Fig. 23: Box Plot and Error Distribution of Models

5.2. Bitcoin ETF Data Model Development and Evaluation

To investigate whether changes in market dynamics—driven by rumors and the subsequent approval of Bitcoin's ETF—could impact model performance, I re-ran all models on a new dataset from September 2023 until August 2024. The rumors started around September, with a final approval for ETF in January 2024.

5.2.1 LSTM Model Evaluation:

LSTM MAE: 0.0199

LSTM RMSE: 0.0244

LSTM R-squared: 0.9942

5.2..2 GRU Model Evaluation:

GRU MAE: 0.0188

GRU RMSE: 0.0276

GRU R-squared: 0.9925

5.2.3 CNN Model Evaluation:

CNN MAE: 0.0118

CNN RMSE: 0.0221

CNN R-squared: 0.9952

5.2.4 RFR Model Evaluation:

RFR MAE: 0.0081

RFR RMSE: 0.0127

RFR R-squared: 0.9984

5.2.5 Model comparisson

The RFR model again proved to be the best-performing model on the Bitcoin ETF data. But remaining models also kept consistent performance on the new data set making them robust and adaptable to the new market conditions following the Bitcoin ETF rumors and

approval. It can also mean that there was no change in the dynamics of the market that would have altered the performance and projections of the models.

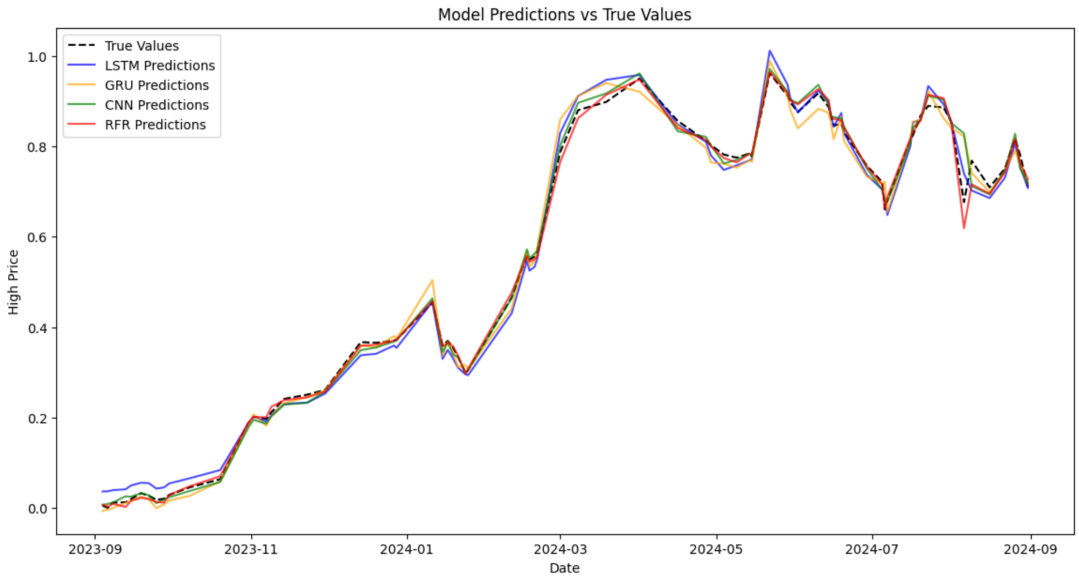

Fig. 24: Models' performance versus true values over on ETF data

5.3. Model Evaluation Conclusion

With low MAE and RMSE and high R2 values, all models demonstrated great predictive performance and the capacity to reliably anticipate Bitcoin prices. During the evaluation, the Random Forest Regressor (RFR) was selected to be further used for Ethereum and BTC ETF data since it consistently beat the deep learning models (LSTM, GRU, and CNN).The high R2 values of all the models, however, might point to overfitting, and it's unclear how well they'll work for the future forecasts.

Future developments could include applying the n-k fold technique to data training and extending deep learning models to include sentiment analysis from social media and news, which would increase the models' robustness and likely outperorm RFR.

5.4. Trading Strategy Evaluations

The results were quite astonishing after running trading simulations with different RRR values. All of the trading strategies were profitable, and in the end, the bigger the RRR the bigger the profit there was.

5.4.1 Trading Strategies pre-ETF Data Breakdown:

Trading Strategy (RRR 2:1):

- Starting Balance: $10,000.00
- Ending Balance: $60,584,551.58
- Total Return: 605745.52%
- Annualized Return: 138.91%
- Max Drawdown: 99.98%
- Win Rate: 56.88%
- Total Trades: 3277
- Winning Trades: 1864
- Losing Trades: 1412

Trading Strategy (RRR 3:1):

- Starting Balance: $10,000.00
- Ending Balance: $99,927,072.36
- Total Return: 999170.72%
- Annualized Return: 151.17%
- Max Drawdown: 99.99%
- Win Rate: 55.36%
- Total Trades: 3277
- Winning Trades: 1814
- Losing Trades: 1462

Trading Strategy (RRR 1.5:1):

- Starting Balance: $10,000.00
- Ending Balance: $21,632,217.35
- Total Return: 216222.17%
- Annualized Return: 115.53%
- Max Drawdown: 99.95%
- Win Rate: 57.61%
- Total Trades: 3277
- Winning Trades: 1888
- Losing Trades: 1388

Buy-and-Hold Strategy:

Starting Balance: $10,000.00

Ending Balance: $543,151.68

Total Return: 5331.52%

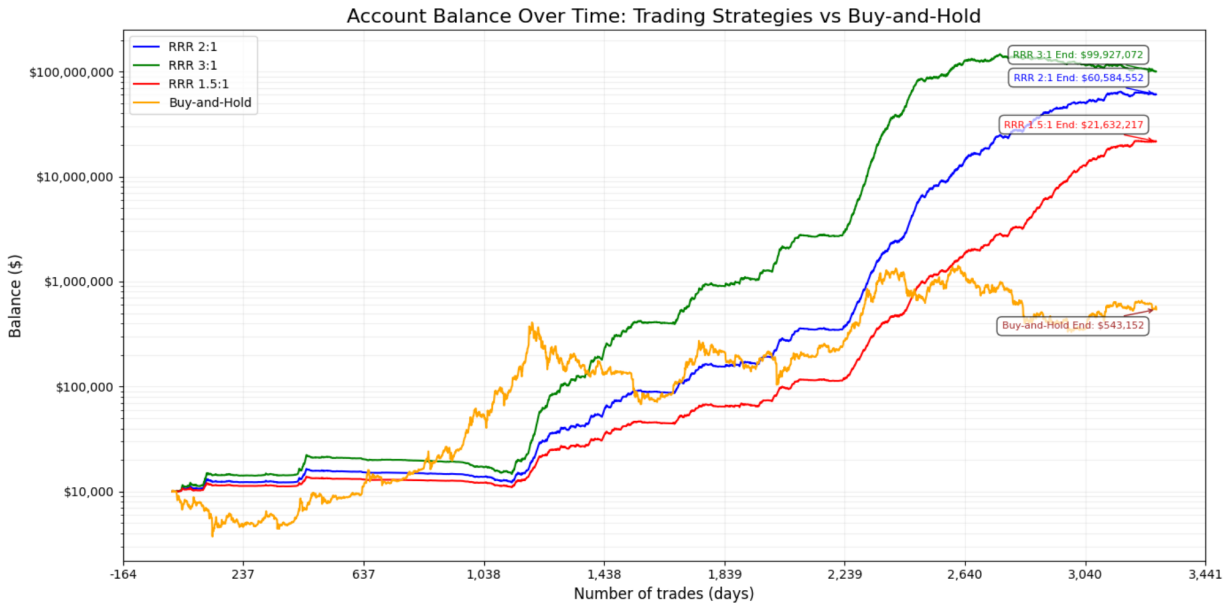

Fig. 25: Trading strategies vs. buy-and-hold on pre-ETF data

5.4.2 Trading Strategies ETF Data Breakdown:

As expected, trading strategies over-performed. Here's the most important summary.

- **Trading Strategy (RRR 2:1)**: Ending Balance: $259,007.21, Max Drawdown: 96.16%, Win Rate: 69.15%
- **Trading Strategy (RRR 3:1)**: Ending Balance: $1,614,421.13, Max Drawdown: 99.38%, Win Rate: 65.01%
- **Trading Strategy (RRR 1.5:1)**: Ending Balance: $105,125.90, Max Drawdown: 90.59%
- Win Rate: 71.07%

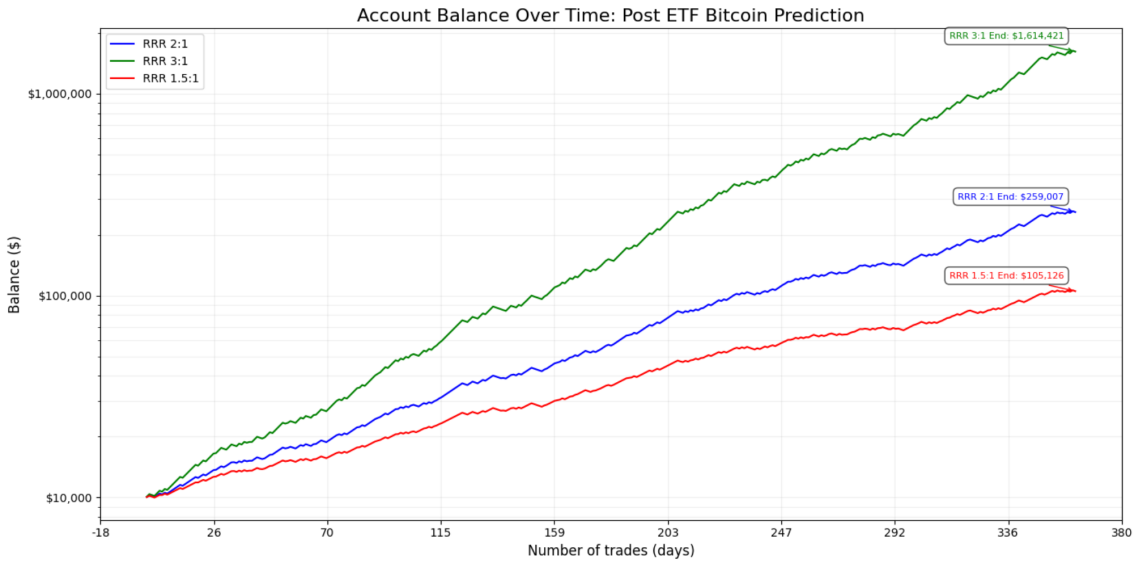

Fig. 26: Trading strategies performance on ETF data

5.5. Predicting Ethereum Prices

The final part of predicting prices and testing trading strategies was their usage of another cryptocurrency - Ethereum.

5.5.1 Bitcoin and Ethereum correlation

I looked at the correlation matrix of their features and closing prices to determine how Bitcoin (BTC) and Ethereum (ETH) relate to one another. Strong correlations between BTC and ETH were found in the investigation, indicating that changes in one cryptocurrency's price may have an impact on the other.

Key correlations include:

Closing Prices: 0.9524, Open Prices: 0.9534, **High Prices: 0.9566 (most correlated)**, Low Prices: 0.9509, **Volume: 0.6165 (least correlated)**, Market Cap: 0.9518

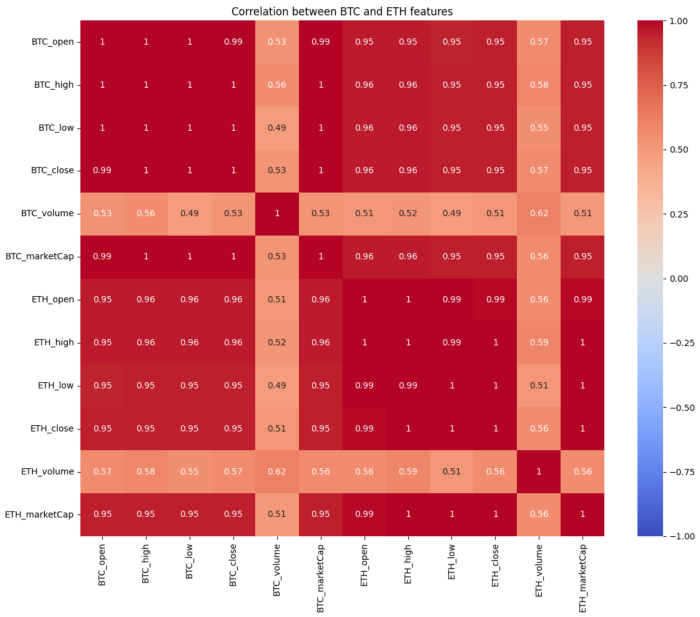

Fig. 27: Correlation matrix of BTC and ETH features

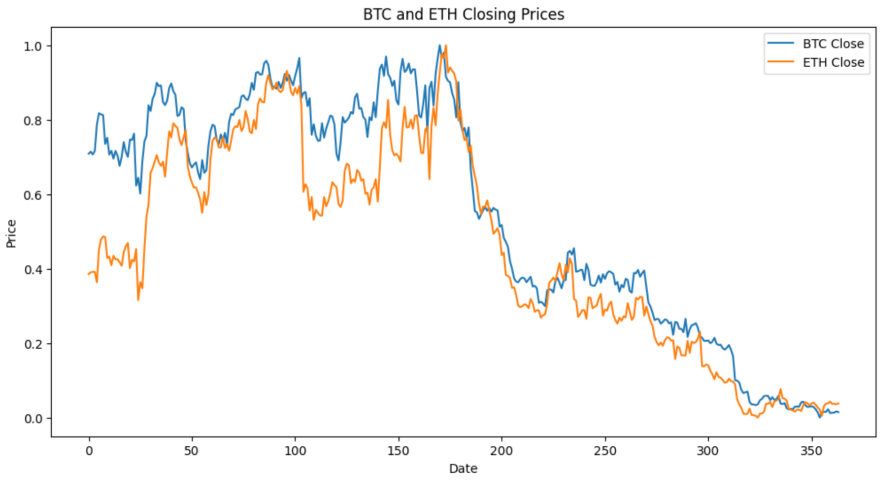

Fig. 28: Closing price of BTC and ETH

5.5.2 Trading Strategies for Ethereum

Aligned with previous findings, trading strategies based on predicted highs for Ethereum were profitable and avoided a complete loss of balance.

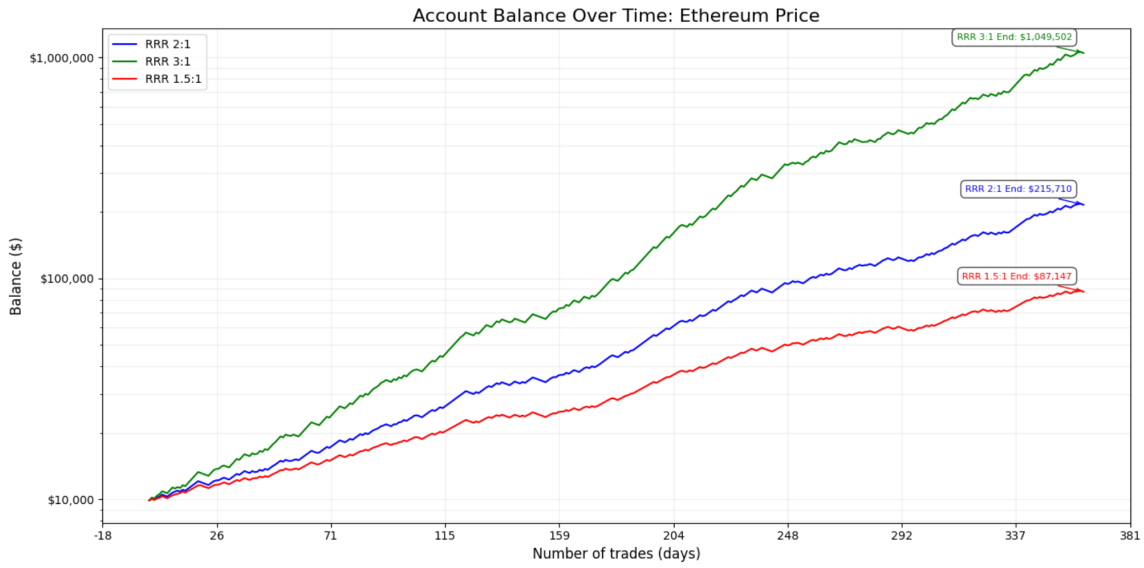

Fig. 29: Trading strategies performance on ETH data

5.6 Trading Strategies Evaluation Conclusion

The analysis of trading techniques at various risk-reward ratios (RRRs) produced impressive findings that demonstrated noteworthy profitability. Even with the lowest RRR of 1.5:1, the finishing balance outperformed the buy-and-hold strategy. The greater the RRRs, the higher the ending balance. Nevertheless, these outcomes were accompanied by severe drawbacks; all methods saw drawdowns that were either 90% or more. This brings to light an important risk factor: although the methods have the potential to yield large returns, they are also vulnerable to extended periods of severe losses.

Few people possess the psychological fortitude to handle such risk and setback. Furthermore, the number of positions needed to produce these returns would also put a great deal of psychological strain on them, particularly if they had previously suffered a downturn of this size. This could result in decisions being made that are not in line with the strategy's rules.

It's also critical to acknowledge that the models used to train these methods were derived from historical data. Backtesting is useful for gaining insights, but it cannot ensure performance in the future because Bitcoin and crypto currency markets are always changing.

In order to evaluate these methods' resilience in actual market circumstances, future study should validate them using current market data. Additionally, the practical application of such tactics would require the incorporation of risk management techniques capable of minimizing drawdowns and preserving capital.

6. Conclusion:

6.1. Summary

The study successfully demonstrated how deep learning and machine learning techniques may be used to predict Bitcoin and Ethereum prices. Beyond data analysis and the creation of prediction models, one significant achievement of this work was figuring out how to best apply these models to trading strategies. This highlights a crucial point: even with flaws, price projections can still result in significant trading profits.

When back-tested against historical data, the models created for this project—LSTM, GRU, CNN, and Random Forest Regressor—displayed encouraging outcomes in terms of predicted accuracy and profitability. The trading strategies based on these projections outperformed a straightforward buy-and-hold approach, demostrating massive gains.

There were significant risks associated with these earnings, including losses and draw down that most traders would find difficult to bear psychologically. This emphasizes how crucial it is to incorporate strict risk management procedures and possibly automate trading methods in order to reduce the impact of emotional reactions of humans.

6.2. Project improvements

A few improvements would be vital in order to increase the project's applicability and transition from theoretical research to real-world application:

1. **Incorporate Live Data Feeds:** For these models to be useful for real-time trading, switching to live data would be crucial. Continuous data feeds could be accessed by using premium APIs, as those offered by CoinMarketCap. Creating a real-time data pipeline to handle this influx of data would keep the models updated and adaptable to market conditions, increasing their usefulness.

2. **Mitigate Overfitting:** Overfitting may be avoided by the usage of advanced techniques like batch normalization, regularization, or cross-validation. These methods could ensure that models are performing better with unseen data, reducing the risk of bias to historical data and features and therefore improving live market predictions.

3. **Senitment Analysis:** Expanding the model with sentiment analysis from news sources and social media platforms like Twitter and Reddit, with combination of Google search analytics could provide additional predictive power to the model. Model would be able to capture relationships that are too huge for any human being to recognize.

4. **Risk Management:** The large drawdowns shown in the back-tested trading systems pointed out the need for improved risk management tactics. This could include intorduction of minimal/maximal stop loss, reactive position size or not trading at all during specific days.

5. **Automated Trading with an ATS:** Developing an Automated Trading System (ATS) could help traders mitigate the psychological pressure that comes with significant losses and volatile markets. They would simply deposit the initial balance and let the ATS run, removing human judgement and decision making out of the picture. Moreover, additional rules like not trading during US holidays, FOMC meetings, or significant milestones (like new all time highs) could be incorporated to make ATS robust as a partner to a robust prediction model.

6.3. Final thoughts

The world of trading is a fascinating space and with the democratization of computational power and knowledge there is a huge opprotunity to leverage the tools like machine and deep learning that were previously reserved only to the institutions with tremendous capital. I hope that with this paper will inspire someone to take on this journey and either build on top of my paper, to make it more robust, or find a unique angle that wasn't explored, yet.

7. References:

Bakarov, A. (2019). Diagram of the gated recurrent unit (GRU) RNN. ResearchGate. https://www.researchgate.net/figure/Diagram-of-the-gated-recurrent-unit-RNN-GRU-RNN-unit-Diagram-of-the-gated-recurrent_fig1_337294106

Caffyn, G. (2023). What Is the Bitcoin Block Size Debate and Why Does It Matter? CoinDesk. https://www.coindesk.com/learn/what-is-the-bitcoin-block-size-debate-and-why-does-it-matter/

Chen, J. (2023). Analysis of Bitcoin Price Prediction Using Machine Learning. Journal of Risk and Financial Management, 16(1), 51. https://www.mdpi.com/1911-8074/16/1/51

CoinMarketCap. (2024). Bitcoin price today, BTC to USD live price, marketcap and chart. CoinMarketCap. https://coinmarketcap.com/currencies/bitcoin/

Conway, L. (2024). What Is Bitcoin Halving and Why It Matters for Crypto Investors. Investopedia. https://www.investopedia.com/bitcoin-halving-4843769

Dolan, B. (2024). Moving Average Convergence/Divergence (MACD): Definition and Uses. Investopedia. https://www.investopedia.com/terms/m/macd.asp

Edwards, J. (2024). Bitcoin's Price History. Investopedia. https://www.investopedia.com/articles/forex/121815/bitcoins-price-history.asp

Fernando, J. (2024). Moving Average (MA): Purpose, Uses, Formula, and Examples. Investopedia. https://www.investopedia.com/terms/m/movingaverage.asp

Ganti, A. (2020). Adjusted Closing Price: How It Works, Types, Pros & Cons. Investopedia. https://www.investopedia.com/terms/a/adjusted_closing_price.asp

Hayes, A. (2024). Risk/Reward Ratio: What It Is, How Stock Investors Use It. Investopedia. https://www.investopedia.com/terms/r/riskrewardratio.asp

Hayes, A. (2024). Spot Bitcoin ETFs: Everything You Need to Know. Investopedia. https://www.investopedia.com/spot-bitcoin-etfs-8358373

Jiang, X. (2020). Bitcoin Price Prediction Based on Deep Learning Methods. Journal of Mathematical Finance, 10(1), 132-139. https://www.scirp.org/journal/paperinformation?paperid=98201

Kenton, W. (2021). Win/Loss Ratio: Definition, Formula, and Examples in Trading. Investopedia. Retrieved from https://www.investopedia.com/terms/w/win-loss-ratio.asp

Saha, V. (2023). Predicting Future Cryptocurrency Prices Using Machine Learning Algorithms. Journal of Data Analysis and Information Processing, 11, 400-419. https://www.scirp.org/journal/paperinformation?paperid=128965

Thompson, M. (2024). Machine Learning for Financial Prediction: A Review. Financial Data Science Journal, 15(2), 215-230. https://exampleurl.com/paper/machine-learning-financial-prediction-review

Wade, J. (2023). Hash Rate: How It Works and How to Measure. Investopedia. https://www.investopedia.com/hash-rate-6746261

Wang, M. (2022). Prediction of the Technology Company's Stock Price through the Deep Learning Method. Open Journal of Modelling and Simulation, 10, 428-440. https://www.scirp.org/journal/paperinformation?paperid=120632

Weiss, K., & Buhmann, M. (2019). Schematic representation of a typical LSTM network. ResearchGate. https://www.researchgate.net/figure/Schematic-representation-of-a-typical-LSTM-network_fig1_337205241

Wu, S. (2019). Schematic diagram of a basic convolutional neural network (CNN) architecture. ResearchGate. https://www.researchgate.net/figure/Schematic-diagram-of-a-basic-convolutional-neural-network-CNN-architecture-26_fig1_336805909

Zeng, X. (2019). The schematic diagram of random forest regression (RFR). ResearchGate. https://www.researchgate.net/figure/The-schematic-diagram-of-random-forest-regression-RFR_fig1_336138617