Time is Mining: Applying Statistical and Neural Networks Models to Forecast Bitcoin Price

Masters in Data Science

João Filipe Rodrigues Peixoto

Leiria, September 2024

Time is Mining: Applying Statistical and Neural Networks Models to Forecast Bitcoin Price

Masters in data science

João Filipe Rodrigues Peixoto

Dissertation carried out under the guidance of

Professor Carlos Fernando de Almeida Grilo and Professor José Maria Gouveia Martins

Leiria, September 2024

Originality and Copyrights

This dissertation is original, prepared solely for this purpose, and all authors whose studies and publications contributed to its preparation have been duly cited.

Dedicatory

Lipa, thank you for giving me all the strength and courage to chase my dreams. After all these years, my love and admiration are still the same.

José, thank you for teaching me the importance of enthusiasm and joy.

Tomás, you still need to teach me how to break every heart with a smile.

Every day, I'm thankful to have you in my life.

Love you.

Acknowledgements

I would like to express my deepest gratitude to Professor Carlos Fernando de Almeida Grilo and Professor José Maria Gouveia Martins for their unwavering commitment in guiding and supporting me throughout this dissertation. Your ability to keep me focused and offer the right words at the right time gave me the confidence to believe in my work.

It has been an honor to work with you.

Thank you.

Abstract

The volatile nature of Bitcoin, the most prominent cryptocurrency, presents a significant challenge for accurate price forecasting. Given this volatility, finding a reliable forecasting model is crucial for making informed decisions in trading, investing, and risk management. This study explores various methodologies for predicting Bitcoin prices, focusing on both traditional statistical models and state-of-the-art neural networks. Two primary approaches were examined: directly modelling the price and predicting daily returns, which were later converted into price predictions. Both methods underwent an extensive hyperparameter optimization process to ensure optimal performance, evaluated using metrics such as mean absolute error (MAE), mean squared error (MSE), root mean squared error (RMSE), mean absolute percentage error (MAPE), and R-squared (R^2).

The return-to-price transformation approach led to significant improvements in older models like LSTM and GRU, enhancing accuracy and reducing the inherent randomness within neural networks, thereby increasing stability. PatchTST, when modelling the price directly, achieved the best results with an MAE of $709.91 and a MAPE of 1.702%, although all models produced similar outcomes. Despite low error rates, the models struggled to capture daily price spikes, particularly when predicting returns, suggesting limitations in forecasting significant price movements based solely on historical prices.

This work concludes by identifying limitations and providing recommendations for future research, particularly in integrating external variables and exploring statistical models designed to predict volatility, such as GARCH, which could potentially offer deeper insights into sharp price fluctuations. Additionally, investigating the performance of the tested models in multi-step forecasting scenarios and the impact of adjusting the time series' granularity to smooth out price volatility warrants further exploration.

Keywords: Time Series, Bitcoin Forecast, Statistical Forecasting Models, Neural Networks Forecasting Models, N-HITS, PatchTST

Resumo

A volatilidade do preço da Bitcoin, a criptomoeda mais proeminente, representa um desafio significativo para uma previsão correta dos preços. Dada esta volatilidade, é crucial encontrar um modelo de previsão fiável para tomar decisões informadas em matéria de *trading*, investimento e gestão de riscos. Este estudo explora várias metodologias para prever os preços da Bitcoin, centrando-se tanto nos modelos estatísticos tradicionais como nas redes neurais de última geração. Foram examinadas duas abordagens principais: modelação direta do preço e previsão de retornos diários, que foram posteriormente convertidos em previsões de preços. Ambos os métodos foram submetidos a um extenso processo de otimização de hiperparâmetros para garantir um desempenho ótimo, avaliado através de métricas como o erro absoluto médio (MAE), o erro quadrático médio (MSE), a raiz do erro quadrático médio (RMSE), o erro percentual absoluto médio (MAPE) e o R-quadrado (R^2).

A abordagem de transformação do retorno para o preço conduziu a melhorias significativas em modelos mais antigos como LSTM e GRU, aumentando a precisão e reduzindo a aleatoriedade inerente às redes neuronais, aumentando assim a estabilidade. O PatchTST, ao modelar o preço diretamente, obteve os melhores resultados com um MAE de $709,91 e um MAPE de 1,702%, embora todos os modelos tenham produzido resultados semelhantes. Apesar das baixas taxas de erro, os modelos tiveram dificuldade em captar os picos de preços diários, em particular quando previram retornos, sugerindo limitações na previsão de movimentos de preços significativos com base apenas em preços históricos.

Este trabalho conclui identificando as limitações e fornecendo recomendações para investigação futura, particularmente na integração de variáveis externas e na exploração de modelos estatísticos concebidos para prever a volatilidade, como o GARCH, que poderiam potencialmente oferecer uma visão mais profunda das flutuações acentuadas dos preços. Além disso, a investigação do desempenho dos modelos testados em cenários de previsão em várias etapas e o impacto do ajustamento da granularidade das séries cronológicas para suavizar a volatilidade dos preços merecem ser explorados mais aprofundadamente.

Palavras-chave: Séries temporais, previsão de Bitcoin, modelos de previsão estatística, modelos de previsão de redes neurais, N-HITS, PatchTST

Table of Contents

List of Figures

List of Tables

List of abbreviations

ADF	Augmented Dickey Fuller Test
ANN	Artificial Neural Network
AR	Auto Regressive
ARCH	Auto Regressive Conditional Heteroskedasticity
ARIMA	Auto Regressive Integrated Moving Average
BTC	Bitcoin
CRISP-DM	Cross-Industry Standard Process for Data Mining
GARCH	Generalized Autoregressive Conditional Heteroskedasticity
GRU	Gated Recurrent Unit
LSTM	Long Short-Term Memory
MA	Moving Average
MAE	Mean Absolute Error
MAPE	Mean Absolute Percentage Error
MASE	Mean Absolute Scaled Error
MLP	Multilayer Perceptron
MQLoss	Multi Quartile Loss
MSE	Mean Squared Error
NBEATS	Neural Basis Expansion Analysis Time Series
N-HiTS	Neural Hierarchical Interpolation for Time Series
PatchTST	Patch Time Series Transformer
ReLU	Rectified Linear Unit

RMSE	Root Mean Squared Error
RNN	Recurrent Neural Networks
SARIMA	Seasonal AutoRegressive Integrated Moving Average
sMAPE	Symmetric Mean Absolute Percentage Error
TiDe	Time-series Dense Encoder

1 Introduction

Since its creation in 2009, *bitcoin* has revolutionized the financial landscape, which started as an experimental cryptocurrency and has become into a major asset class. Initially adopted by tech enthusiasts and libertarians, bitcoin has since attracted institutional investors, hedge funds like BlackRock, and multinational corporations such as Tesla and MicroStrategy. Its role in the digital financial revolution has been significant, offering a decentralized, transparent, and secure system for currency and transactions. As bitcoin's influence on the global financial system continues to grow, its impact is felt across multiple sectors.

Bitcoin is often described both as a store of value, comparable to gold, and as a medium of exchange, similar to traditional currencies like the US dollar or the euro. Its limited supply and price volatility have made it a popular speculative asset. Various factors influence its price, including market demand, technological advancements, regulatory changes, and shifts in public perception.

Given bitcoin's growing prominence and highly volatile nature, accurately forecasting its price has become a key area of research. Predicting its future price movements is crucial for investors and stakeholders looking to mitigate risk and capitalize on market opportunities.

1.1 Objective

The main objective of this work is to predict Bitcoin's price for the following day. This presents a typical time series forecasting challenge, where the goal is to identify the model that can consistently deliver accurate predictions over an extended period. Given bitcoin's volatile nature, finding a reliable forecasting model is crucial for making informed decisions in trading, investing, and risk management. The study aims to evaluate various statistical and machine learning models to determine which approach yields the best results for bitcoin price forecasting.

1.2 Methodology

There were multiple methodologies considered for this study, including data science frameworks like *OSEMN* (*O*btain, *S*crub, *E*xplore, *M*odel, i*N*terpret), *KDD* [1], and *CRISP-DM* [2], as well as time series-specific approaches like the basic steps in forecasting suggested by Rob J. Hyndman et al. [3]. However, after analysing these methods, it becomes clear that they all rest on the same foundational principles. First, they require a deep understanding of the business question at hand, followed by gathering and preparing the appropriate data. Next, different models and approaches are explored to address the problem, and finally, these approaches are compared to select the best solution.

This methodology seeks to incorporate the most common steps from all the approaches mentioned while emphasizing a key shared characteristic: flexibility. Built on four pillars, *Business*, *Data*, *Model* and *Result*, the framework maintains a logical flow, starting with the business problem, followed by data management, model development, and results evaluation. However, these steps are not rigid; the process is adaptable, allowing for iterations and reordering as needed during the project. This flexibility is crucial for addressing unforeseen challenges or insights that arise during development. These four pillars and their interconnection are illustrated in Figure 1.

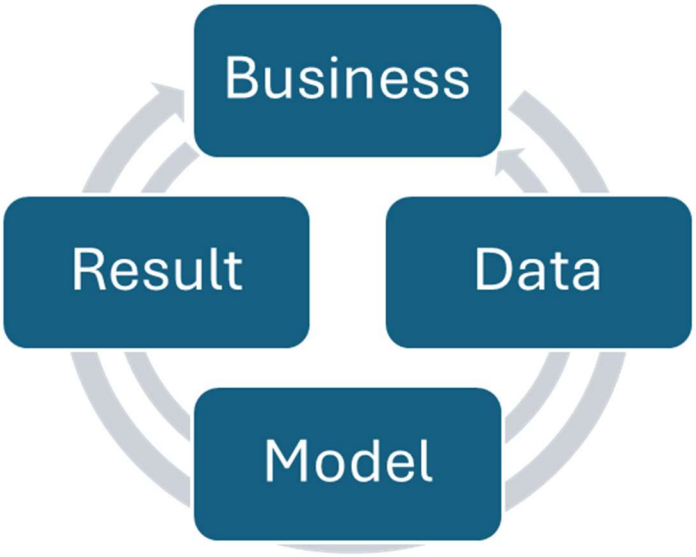

Figure 1: Methodology Pillars

Business is the starting point of any project. In this pillar, the business problem is clearly defined, along with the goals and objectives of the analysis. It involves understanding the requirements, challenges, and the broader context of the problem. For this study, the focus is on predicting Bitcoin prices, so the business question revolves around how to create a reliable forecast that investors, traders, and analysts can use for decision-making.

Once the business problem is understood, the next step is to gather the relevant data. This involves identifying the necessary data sources, collecting the data, and performing exploratory data analysis. In this study, the data is centred around Bitcoin prices, and analysis will be conducted to ensure it is clean, relevant, and suitable for modelling. This pillar also involves any necessary transformations to prepare the data for the modelling phase.

After preparing the data, the focus shifts to selecting and building models. This pillar encompasses the design, configuration, and execution of various forecasting models, including both statistical and machine learning approaches. The process involves training the models, tuning hyperparameters, and validating performance on test datasets. In this study, multiple models will be explored, such as ARIMA, LSTM, and others, to determine the best-performing approach for predicting Bitcoin prices.

The final pillar focuses on evaluating and interpreting the results of the models. In this stage, error metrics are used to compare the performance of the different models. Based on the findings, insights are drawn, and the best solution is selected. This pillar also includes presenting the results in a way that answers the initial business question and supports decision-making, alongside identifying possible next steps or improvements for future work.

These pillars, though distinct, are interrelated and can loop back to earlier stages when necessary, ensuring the methodology is iterative and adaptable to the project's needs.

1.3 Structure

The structure of the study was designed to guide the reader through the process as if they were conducting their own project. Each chapter builds logically upon the previous one, providing a clear and comprehensive understanding of the subject. The intention is to not only present the findings but also offer a step-by-step journey through the entire forecasting process, from understanding the background concepts to applying advanced models, analysing results, and drawing conclusions.

Chapter 2, Background: This chapter explains the most important concepts necessary to understand the subsequent chapters. It provides an overview of time series forecasting, statistical and machine learning models, and key concepts related to Bitcoin and its pricing dynamics.

Chapter 3, Literature Review: This chapter evaluates the evolution of time series forecasting and Bitcoin price forecasting by covering the most important literature. It highlights historical models, breakthroughs, and trends in both time series analysis and Bitcoin forecasting.

Chapter 4, Data: This chapter details the data needed for the study, describing how it was gathered, explored, and transformed to meet the project’s requirements. It covers preprocessing techniques, the dataset used, and key variables analysed.

Chapter 5, Modelling: This chapter explains how the models were configured and executed. It outlines the training and validation processes, the choice of hyperparameters, and the rationale behind selecting specific models for this study.

Chapter 6, Results: This chapter presents a comparison of the results from each model, using error metrics to evaluate their performance.

Chapter 7, Conclusions: The final chapter draws conclusions about the study and suggests directions for further research and future improvements in Bitcoin price forecasting.

2 Background

Bitcoin is a decentralized digital currency, operating without a central authority or regulatory body, and its value is known for its high volatility. This unpredictability has made forecasting Bitcoin's price a major area of interest for investors, traders, and researchers, as accurate predictions can lead to significant financial gains or improved risk management strategies.

One way to tackle the challenge of predicting Bitcoin prices is through time series forecasting, which involves analysing historical data to make informed predictions about future values. Time series forecasting can be approached using both statistical and machine learning methods. Traditional statistical models are designed to capture linear patterns and are often effective in scenarios where the relationships between data points remain stable over time. These models are particularly useful for identifying trends and seasonality in short-term forecasts.

On the other hand, machine learning methods can capture complex, nonlinear relationships within the data. These models are well-suited for time series with intricate patterns and can adapt to changing dynamics in the data, making them potentially more accurate for long-term or volatile datasets like Bitcoin prices.

This chapter introduces the fundamental concepts behind Bitcoin and the methodologies of time series forecasting, setting the stage for a comparison of statistical and neural networks based models in predicting Bitcoin's future price.

2.1 Bitcoin

Bitcoin is a decentralized digital currency, introduced in 2008 by an anonymous person or group under the pseudonym Satoshi Nakamoto. It was created to enable direct transactions between parties without needing a third party like a bank. To guarantee that, Bitcoin could not be duplicated or falsified, therefore, Nakamoto proposed a system called blockchain that uses cryptographic proof rather than trust [4].

Blockchain is a distributed ledger that records all transactions in a transparent and tamper-proof way. Transactions are grouped into blocks and linked sequentially, forming a chain. Each block contains a cryptographic hash of the previous block, ensuring that the data cannot be altered without changing all subsequent blocks, which would require consensus from the majority of the network. The blockchain is maintained by a network of nodes, known as miners, who validate transactions through a process called proof of work. This consensus mechanism requires miners to solve complex mathematical problems to add a new block to the chain. The first miner to solve the problem is rewarded with new bitcoins, which is known as the block reward [5].

Bitcoin has a fixed supply of twenty-one million coins, which introduces a deflationary element to the system. Every 210,000 blocks (approximately every four years), the reward

for mining new blocks is halved in an event known as the Bitcoin halving. The halving ensures that the total supply of Bitcoin will not exceed the limit, making it scarce [6].

Mining is energy-intensive, as it requires specialized hardware and significant computational power to solve the cryptographic puzzles. The increasing difficulty of mining over time has also concentrated mining activities in regions with cheaper energy costs [7].

Bitcoin has been described as both a store of value and a medium of exchange. Some view Bitcoin as "digital gold," using it as a hedge against inflation and economic instability, others see its potential as a global currency for frictionless international payments. Due to its price volatility and limited supply, it has gained popularity as a speculative asset [8].

Bitcoin was launched in January 2009, and the first-ever block of Bitcoin, called the Genesis Block, was mined by Nakamoto.

2.2 Time Series

A *time series* is a sequence of data points collected at regular time intervals over time, such as annual birth rates, monthly sales figures, daily stock prices, or hourly rainfall amounts. The subject of the data is less important than its time-dependent nature, as the goal is to capture the temporal evolution of a variable or set of variables [9]. When a time series involves the observation of a single variable, it is called *univariate*. If it includes multiple variables, then it is called *multivariate*. In Figure 2 we have an example of a univariate time series with the number of passengers from an US Airline[1].

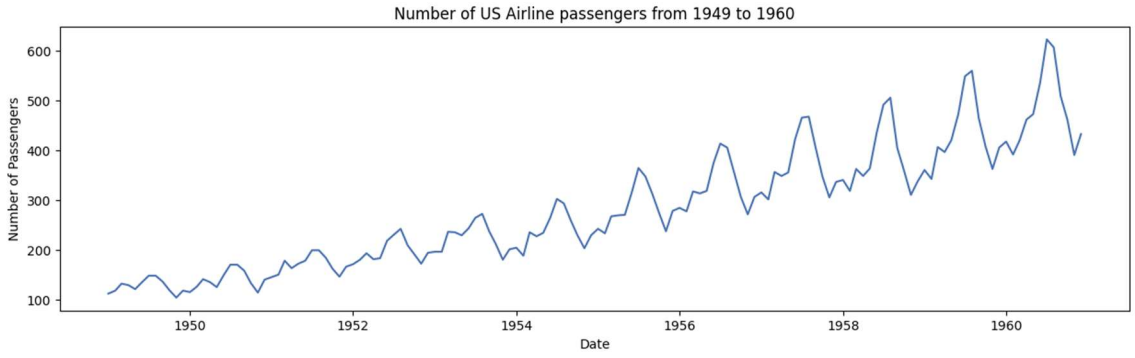

Figure 2: Time series example

Time series data typically consist of several key components that reflect different patterns within the data [3]. These components include:

- *Trend*: The long-term movement in the data, representing an overall upward or downward direction;
- *Seasonality*: Regular patterns or cycles in data that repeat at consistent intervals;
- *Cyclical Patterns*: Fluctuations in the data that occur over irregular periods due to broader economic or natural cycles;

[1] https://www.kaggle.com/datasets/chirag19/air-passengers

- *Irregular/Residual Component*: Random variations or noise that cannot be assigned to trend, seasonality, or cyclical components.

In Figure 3, we present a classic decomposition of our example above. The time series exhibits a clear upward trend alongside a twelve-month seasonal pattern, with no visible cyclical fluctuations.

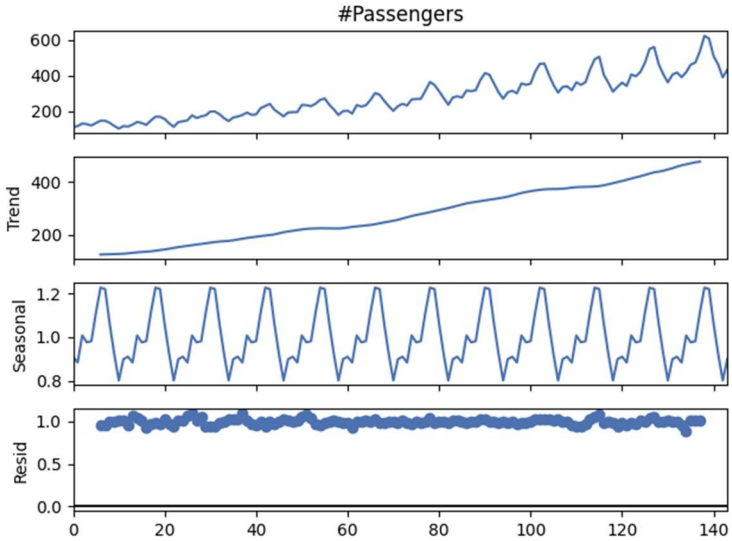

Figure 3: Time Series decomposition

Time series analysis aims to predict future values based on historical data, a process known as *time series forecasting*. Forecasting is based on the premise that current and past knowledge can be used to make predictions about the future, and we can see examples of forecasting every day. Meteorologists, for instance, analyse time series data on temperature, rainfall, and wind speed to predict future weather conditions. Governments develop budgets by predicting future expenses, while investors forecast stock prices, interest rates, and other financial indicators to make informed decisions. In healthcare, tracking time series data such as heart rate, blood pressure, or glucose levels can assist in the early diagnosis of potential health issues [10].

A crucial concept in time series analysis is *stationarity*. A time series is considered stationary if its statistical properties (mean, variance, autocorrelation) do not change over time [11]. Stationarity simplifies the modelling process and makes it easier to predict future values, because the characteristics of the series remain consistent over time [9].

2.3 Statistical models

Statistical methods are central to time series forecasting, providing robust models for understanding trends, seasonality, and cyclical behaviour. These methods, such as *ARIMA*, Auto Regressive Integrated Moving Average, and its seasonal extension *SARIMA*, help capture the linear patterns in the data and are widely used due to their simplicity and interpretability. By focusing on past values and error terms, these models effectively handle

time series components like trend and seasonality, making them valuable tools for generating accurate forecasts, particularly for stable and structured time series.

2.3.1 ARIMA – Auto Regressive Integrated Moving Average

The ARIMA is a widely used statistical model for forecasting. As the name suggests, it is built on three components: Auto Regressive (AR), Integrated (I), and Moving Average (MA). The Auto Regressive component refers to the use of past values in the regression equation for the series, the Integrated component involves differencing the data to make it stationary by removing trends, and the Moving Average component models the error term as a linear combination of past errors.

ARIMA is typically denoted as ARIMA(p, d, q), where:

- p is the number of lag observations included in the autoregressive component model;
- d is the number of times the raw observations are differenced to make the time series stationary;
- q represents the size of the moving average window.

ARIMA parameters can be selected manually using autocorrelation and partial autocorrelation plots, which help identifying the appropriate values for the AR and MA components. Alternatively, optimization techniques that minimize criteria such as the Akaike Information Criterion (AIC) or Bayesian Information Criterion (BIC) can be used to automate parameter selection, ensuring a more efficient and accurate model fit [3].

2.3.2 SARIMA - Seasonal Auto Regressive Integrated Moving Average

When a time series exhibits seasonality, such as monthly sales figures or quarterly economic data, a Seasonal ARIMA (SARIMA) model can be applied. SARIMA extends ARIMA by incorporating seasonal components to capture repeating patterns. It is specified as SARIMA(p, d, q)(P, D, Q, m), where:

- P: Seasonal autoregressive terms;
- D: Seasonal differencing;
- Q: Seasonal moving average terms;
- m: Seasonal period.

The modelling procedure for SARIMA is like a non-seasonal ARIMA, with the additional need to select seasonal AR and MA terms alongside the non-seasonal components. In simple terms, SARIMA combines seasonal and non-seasonal elements to better model time series data with regular seasonal patterns. [3].

2.4 Machine learning

Machine learning has revolutionized how we approach complex tasks by enabling systems to learn from data and improve their performance over time [12]. Unlike traditional programming, which relies on explicit rules and instructions coded by humans, machine learning models develop patterns and insights automatically by training on large datasets. This capability makes machine learning particularly useful in fields like predictive analytics, natural language processing, image recognition, and autonomous systems.

The objective of *machine learning algorithms* is to generate models able to generalize and infer from the data and not to learn the data itself, so they are generally categorized based on their type of learning. When an algorithm is trained on labelled data it is categorized as *Supervised Learning*. This means that each input is paired with the correct output and the model learns by comparing its predictions to the actual outcomes and adjusting accordingly. Algorithms such as *linear regression*, *decision trees*, and *neural networks* are widely used in supervised learning. This type of models can be applied on spam detection, credit scoring, and image classification [13].

In *Unsupervised Learning* the algorithms operate on unlabelled data, trying to discover hidden patterns or structures. Since no predefined outcomes exist, the model explores the data to identify clusters, associations, or anomalies. Clustering algorithms like *k-means* and *hierarchical clustering* are often used in customer segmentation and anomaly detection [13].

Reinforcement Learning involves interacting with an environment and receiving feedback in the form of rewards or penalties. The algorithm iteratively improves its decision-making process by learning which actions yield the best long-term rewards. Reinforcement learning has applications in game playing, autonomous driving, and robotic control [13, 14].

According to Domingos, the machine learning process consists of combinations of just three components: *Representation*, *Evaluation* and *Optimization*. Representation is the spectrum of algorithms that can learn from the data available and answer the hypothesis being tested. Each generated model is then evaluated based on an internal evaluation or objective function. The optimization step is the method to search for the best model based on metrics that allows to measure the error rate of each model [15].

Machine learning's impact spans across industries, from improving healthcare diagnoses to optimizing financial predictions, personalizing marketing, and enabling autonomous systems. Its importance continues to grow, underpinning advances in artificial intelligence and driving innovation across diverse sectors [16].

Deep learning is a subset of machine learning methods based mainly on neural networks and distinguishes itself by its ability to automatically extract features from raw data without the need for manual feature engineering. This is particularly beneficial in complex tasks, such as speech recognition, image classification, and financial forecasting, where the relationships between variables are not easily discernible.

The multilayered architecture of neural networks – "Deep" comes from neural networks multilayer architecture [12] – allows each layer to learn progressively more abstract representations of the data. For example, in image recognition, lower layers might learn to detect edges and textures, while deeper layers recognize more complex structures, such as shapes or objects.

Additionally, deep learning models benefit from the vast amounts of data available in modern applications, as more data tends to improve model accuracy. Advances in hardware, such as GPUs, have also made it possible to train deep networks efficiently.

Their ability to learn from large amounts of sequential data and adapt to non-linear trends and irregularities makes them highly suitable for forecasting in dynamic environments like stock prices, energy demand, or cryptocurrency markets [12].

To apply supervised learning methods to time series data, the data must be labelled, meaning it needs to be restructured into input-output pairs, or samples. This restructuring is accomplished using a technique called *sliding window*. A sliding window moves across the time series, creating consecutive samples that the model can learn from. Each sample consists of a fixed window of time steps, *input size,* used to predict the output, called *step.* When the model predicts the next time step is called *One-Step Forecast*, if the model predicts two or more future time steps, then it is called *Multi-step Forecast* [17].

2.5 Neural networks based models

Financial markets are influenced by complex non-linear relationships and neural networks are able to capture non-linear patterns in the data and, theoretically, are capable to approximate any measurable function to any desired degree of accuracy [18]. *Artificial Neural Networks*, ANNs, are nonlinear data-driven models capable of performing nonlinear modelling without a priori knowledge about the relationships between input and output variables, so they are a general and flexible modelling tool for forecasting [19]. ANNs are being tested for forecasting since 1964 when a network was tested for weather forecasting [20]. Nowadays, they are used in almost every forecasting scenario, from demand forecasting [21] and online sales [22] in retail, to predicting next word in a sentence [23].

2.5.1 Multilayer Perceptron

The *Perceptron*, exemplified in Figure 4, was introduced by Frank Rosenblatt in 1958 [24] for binary classification tasks and it is the simplest type of an artificial neural network model. To compute the output, the perceptron multiplies the inputs by their weights and sums everything creating a weighted sum of inputs. The weighted sum is then applied to a step function to determine the binary output. To helping shift the step function and prevent the neuron from always passing through the origin, an extra parameter called *bias* is added to the weighted sum [25].

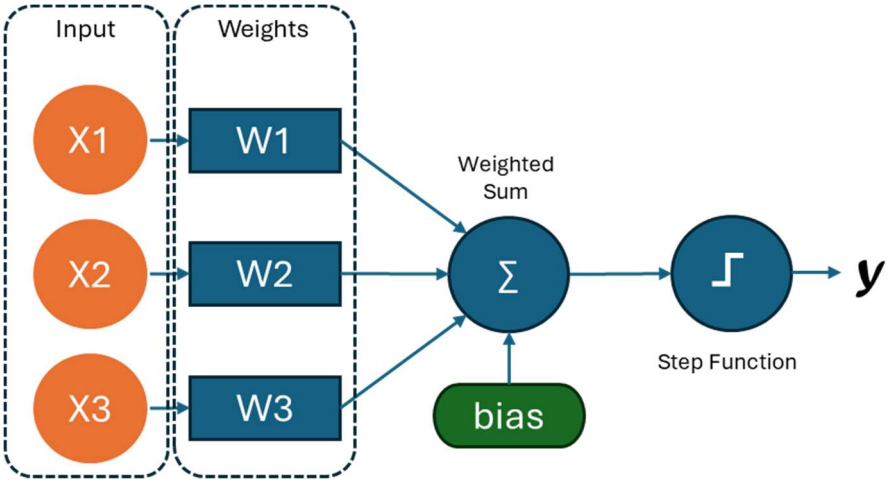

Figure 4: Perceptron

A *Multilayer Perceptron*, MLP, is an evolution of a Perceptron and consists of at least three layers of nodes, or neurons, and each node is connected to every node in the next layer, represented in Figure 5. The first layer is the *input layer* where the model receives input data, or signals. In the next layer, called *hidden layer*, an activation function is applied to the weighted sum of the previous layer outputs. The activation function introduces non-linearity to the model and the most common functions are *sigmoid*, *tanh*, and *ReLU* (Rectified Linear Unit). An MLP can have more than one hidden layer. The last layer, or *output layer* also applies an activation function and computes the predicted values [12, 26].

These types of networks are called *feedforward neural networks* since the information in the model flows in only one direction, from the input layer, through the hidden layers to the output nodes, without any cycles or loops.

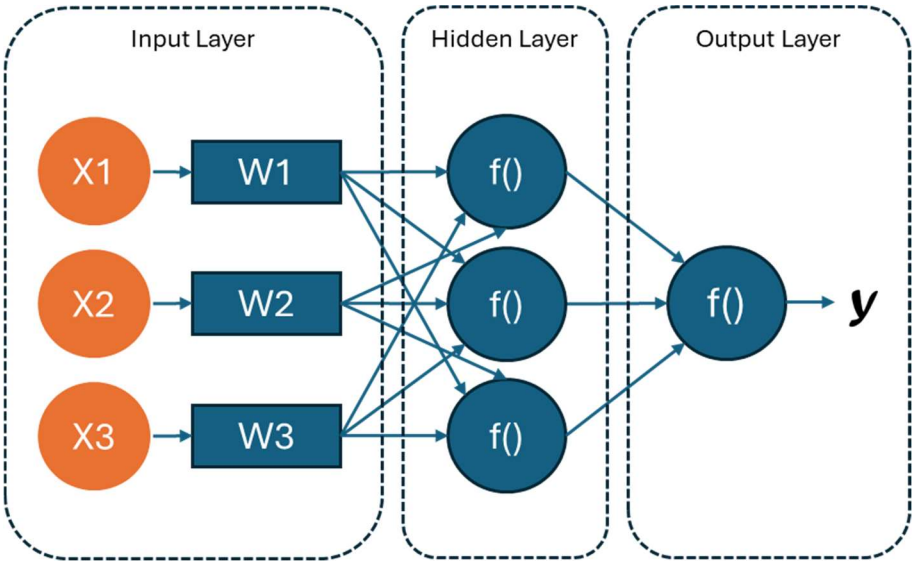

Figure 5: Multilayer Perceptron

Training MLPs had a major breakthrough in 1986 when Rumelhart et al. published a paper introducing the concept of *Backpropagation* to neural networks. Backpropagation "repeatedly adjusts weights of the connection to minimize a measure of difference between

actual output vector and desired output vector" [27]. In other words, Backpropagation computes gradients of a loss function updating the weights accordingly to minimize the loss. Figure 6 represents the flow of information through an MLP.

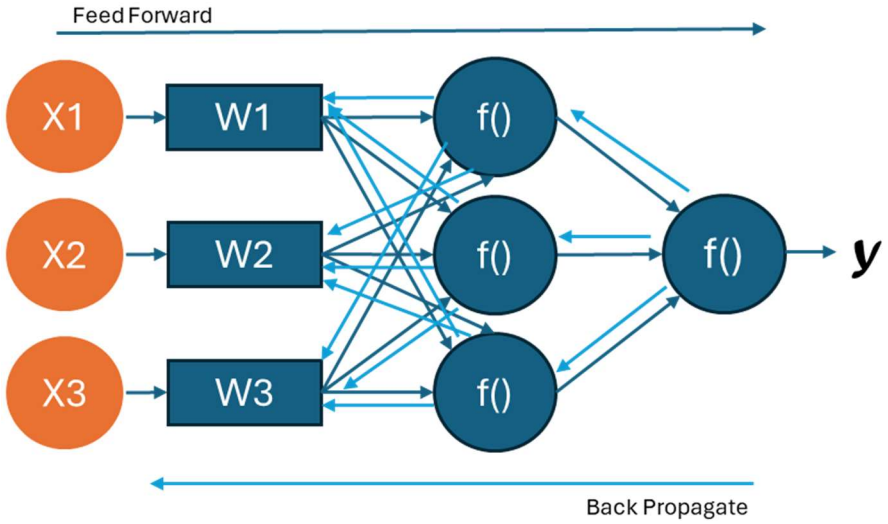

Figure 6: Multilayer Perceptron information flow

2.5.2 Recurrent Neural Networks

Recurrent Neural Networks (RNN) are different from feedforward networks, like MLPs, because they contain feedback loops, as shown in Figure 7. This feature allows nodes to keep an internal state, memory, and process a sequence of inputs. They were developed to handle sequential data like time series, speech or text [12, 28].

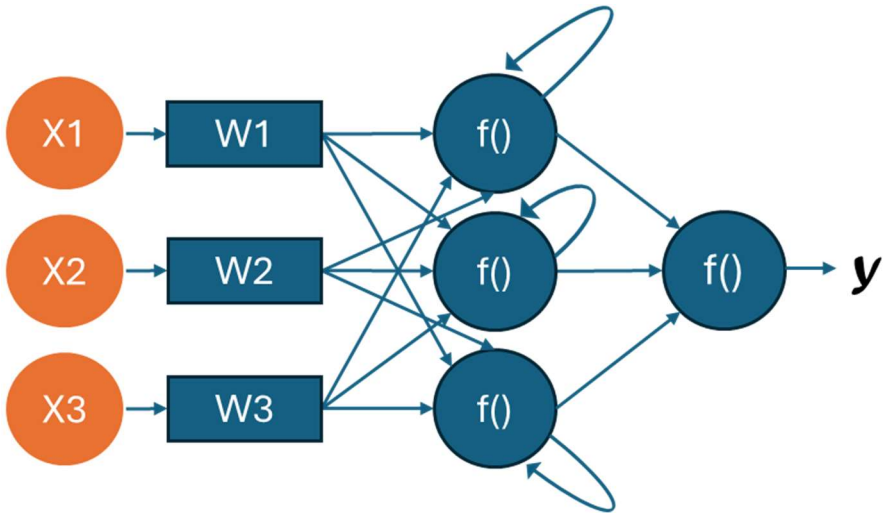

Figure 7: Recurrent Neural Network

An RNN cell, represented in Figure 8, consists of neurons with recurrent connections. Each cell receives input from the current time step and the previous hidden state. The hidden state is updated at each time step, allowing the network to maintain a memory of previous inputs [12, 28].

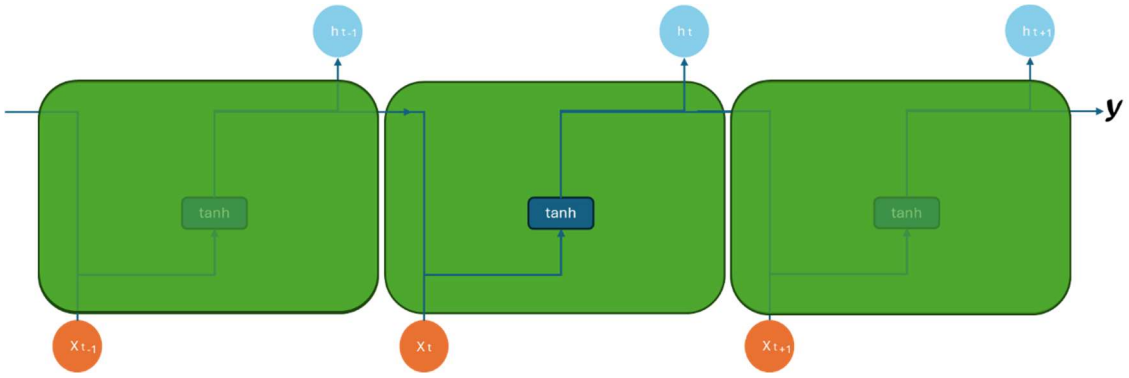

Figure 8: Recurrent Neural Network unfolded

2.5.3 Long Short-Term Memory

Long Short-Term Memory (LSTM) networks are a type of RNN introduced in 1997 to address the RNN's vanishing gradient problem [29]. In traditional RNNs, gradients can become very small, leading to slow learning and difficulty in capturing long-range dependencies. LSTMs introduced gating mechanisms to control information flow, effectively managing long-term dependencies and improving performance on sequential tasks.

LSTMs have two pillars, gates to regulate the flow of information and cell state to carry information across time steps [30]. When processing an input, LSTMs units combine the input with the hidden state of the previous step and then takes it through four gates. The *Forget Gate* decides what information to discard and what information to keep from the cell state. Then, the *Store Gate* decides which values from the input will be stored. The *Update Gate* updates the cell state with the relevant components of prior information and the current input. Finally, the *Output Gate* decides what part of the cell state to output. Figure 9 represents the LSTM architecture.

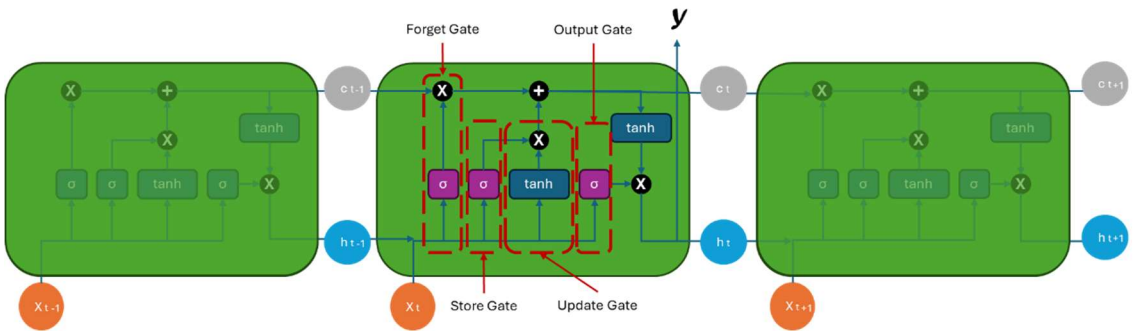

Figure 9: Long Short-Term Memory unfolded

Like MLPs, LSTMs are trained using a variant of Backpropagation adapted for sequence data, Backpropagation through time. It involves unfolding the network through time and calculating gradients for each time step [31].

2.5.4 Gated Recurrent Units

The *Gated Recurrent Unit* (GRU) is a type of RNN architecture introduced by Cho et al. in 2014 [32]. It was designed as a simplified version of the LSTM network, with fewer parameters but comparable performance, particularly in capturing dependencies in sequential data [33].

The main difference between GRU and LSTM architecture is the number of gates. While LSTMs have four gates, GRUs only have two: the *Reset Gate* and the *Update Gate,* represented in Figure 10.

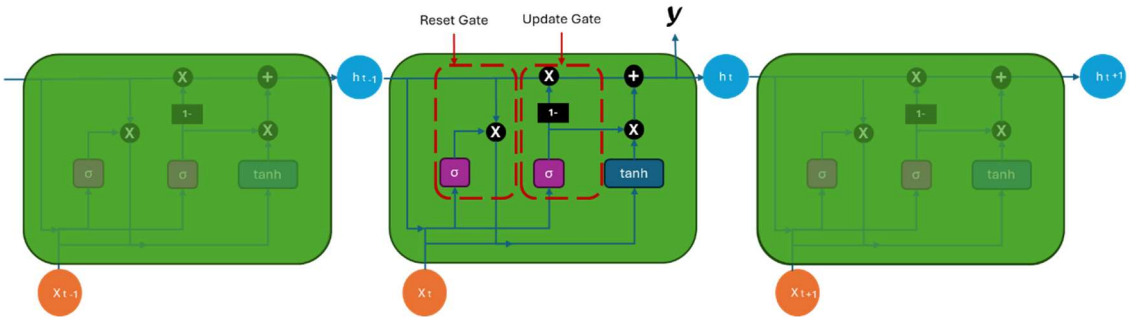

Figure 10: Gated Recurrent Unite unfolded

The Reset Gate controls how much of the previous hidden state to forget, and then the Update Gate determines how much of the past information to pass along to the future.

2.5.5 DeepAR

Introduced by Amazon researchers in 2017, *DeepAR* is a probabilistic forecasts model based on an auto-regressive recurrent network trained on a large number of related time series [34]. This concept brought two main differences to other RNN models available at the time. First, DeepAR predicts the entire probability distribution of future time series values instead of predicting a single point value. This allows the model to capture uncertainty in the forecasts, providing prediction intervals rather than just point estimates. Secondly, DeepAR leverages shared information across multiple time series to improve forecasting accuracy. By sharing parameters across series, the model can generalize better and make more accurate predictions, even for series with limited historical data.

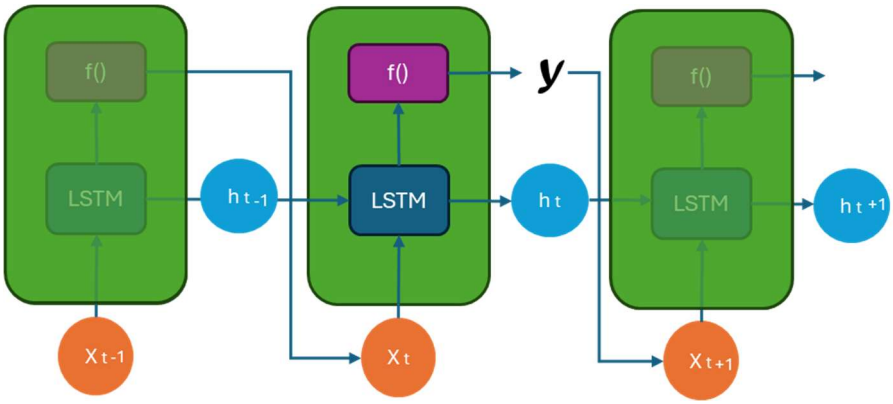

Figure 11: DeepAR unfolded

Figure 9 illustrates how DeepAR is able to predict a probability distribution instead of a deterministic value. DeepAR applies a likelihood function on top of an LSTM. The LSTM cell outputs its hidden state to the next time step, allowing the model to capture temporal dependencies, and the outputs are used as parameters of a likelihood function to predict a probability distribution. This likelihood function depends on the nature of the time series data, and it is typically based on Gaussian or negative binomial distributions. As an input, LSTM cells take the input variables from the current time step and hidden state and target variable from previous time step.

2.5.6 N-BEATS

Neural Basis Expansion Analysis Time Series Forecasting (N-BEATS), introduced by Boris Oreshkin et al. in 2019, was designed based on key principles of a simple and generic architecture that should not rely on domain-specific feature engineering or input scaling with the intent of making its outputs human interpretable [35]. This means that N-BEATS was originally designed for univariate time series forecasting able to produce outputs interpretable regarding trend and seasonality, so the authors developed two model versions, the generic version, and the interpretable version. The N-BEATS architecture is represented in Figure 12.

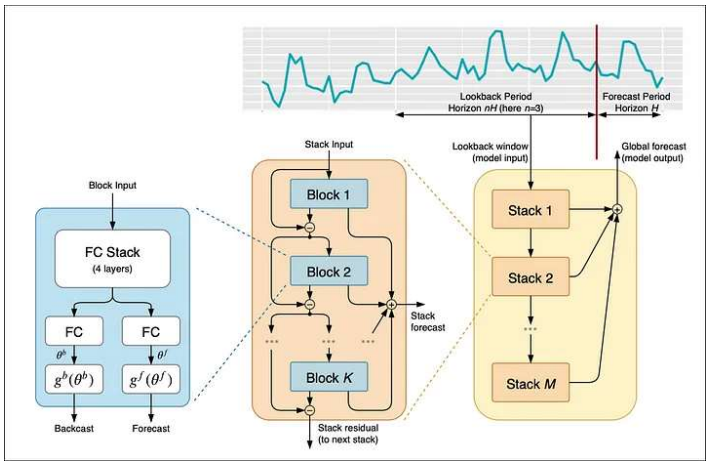

Figure 12: N-BEATS architecture **[35]**

A *block* is the basic processing unit of N-BEATS, consisting of four dense, fully connected layers with ReLU non-linearity. The output of these layers is directed to two separate fully connected layers that calculate the expansion coefficients. These coefficients are then projected to a set of *basis* functions, which are used to compute the *backcast* and *forecast* outputs. The forecast represents the block's forward prediction, while the backcast is the block's best estimate of the past, constrained by the functional space available to approximate signals. This setup enables N-BEATS to apply a doubly residual stacking architecture with two residual branches, one over the backcast prediction and another over the forecast.

A *stack* is a collection of blocks where the first block receives the original input while other blocks receive the backcast signal from the previous block. In each block, the model removes the previous backcast signal from the input, so each block only models a portion of the input signal, and the final forecast is the sum of all forecast signals from all blocks. So, the N-BEATS model is a collection of stacks and stacks are collections of blocks.

To make the model interpretable the authors created an interpretable architecture with two stacks containing three blocks each. The first stack models the typical behaviour of a trend by applying a polynomial of small degree and then the second stack mimics the regular and cyclical behaviour of seasonality by applying the Fourier series. Since each stack has its own output, trend and seasonality are interpretable outputs.

To incorporate exogenous factors and make N-BEATS architecture able to deal with multivariate time series, N-BEATSx was introduced by K.G. Olivares et al. [36].

2.5.7 N-HITS

Neural Hierarchical Interpolation for Time Series Forecasting (N-HiTS) is a deep learning model introduced by Challu et al., in 2022, as an extension of the N-BEATS architecture [37]. This model generates predictions based on forecasts at different time scales combining short-term and long-term effects in the series.

N-HITS's architecture, shown in Figure 13, is very similar to N-BEATS's architecture, it is a collection of stacks, and each stack is a collection of blocks and blocks have a forecast and a backcast. The main difference is the structure of the blocks. Instead of fully connected layers, there is a *MaxPool* layer and an MLP to produce coefficients for the backcast and forecast outputs. The MaxPool layer creates the multi-rate signal sampling that helps the block focus on a specific signal scale.

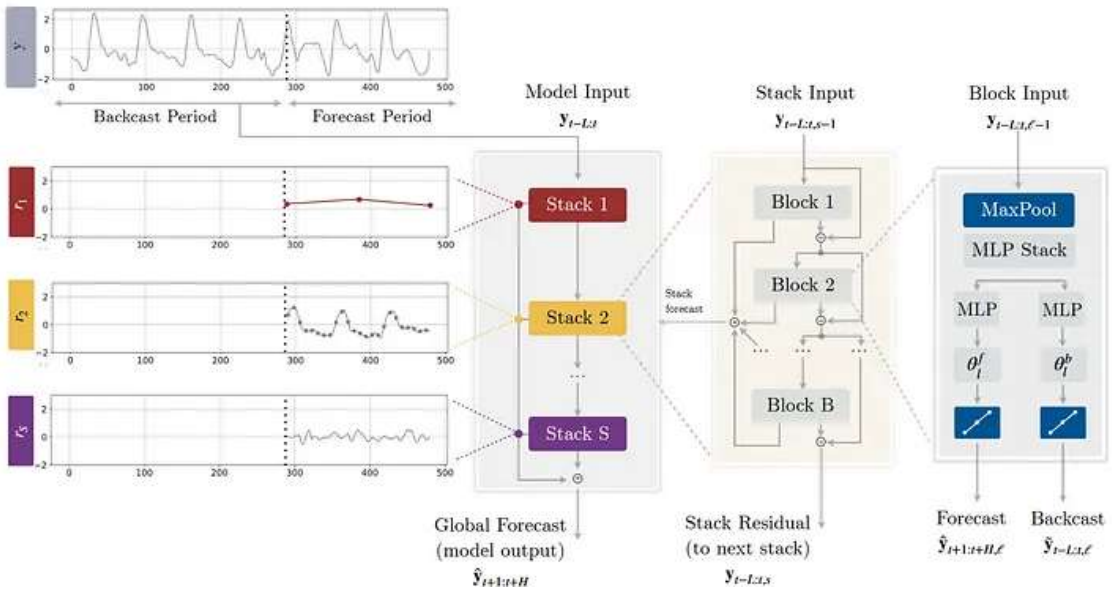

Figure 13: N-HiTS architecture **[37]**

Looking at different time scales helps the model to apply hierarchical interpolation. Each block has an expressiveness ratio, number of predictions per unit of time, that is decreasing from block to block. Example: First block can make 24 predictions per day, one for every hour and the next block make a prediction for every 2 hours, so 12 predictions per day. This interpolation makes the model lighter because it reduces the number of predictions per block.

2.5.8 PatchTST

In 2022, inspired by the success of transformers on various fields, such as natural language processing, computer vision and speech recognition, Nie et al. introduced *Patch Time Series Transformer* (PatchTST). PatchTST applies patching to extract local semantic information and reduce the number of tokens and therefore reduce memory usage and computational complexity [38].

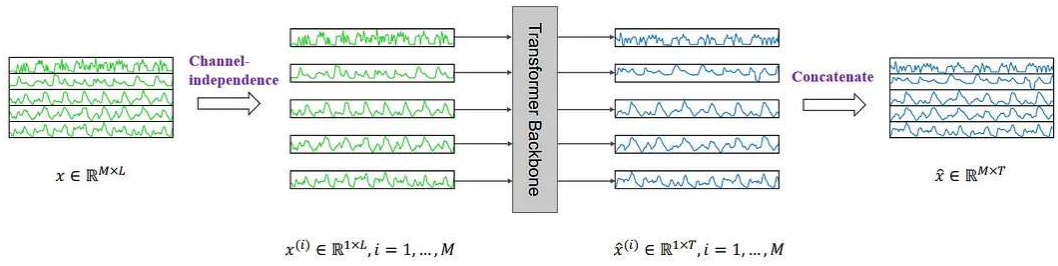

Figure 14: PatchTST architecture **[38]**

As shown in Figure 14, PatchTST is designed to handle multivariate time series. It divides the different series into different channels creating a channel-independence. All channels share the same *Transformer* backbone, illustrated in Figure 15, but the forward processes are independent, so predictions are made for each series and the results are concatenated for the final predictions.

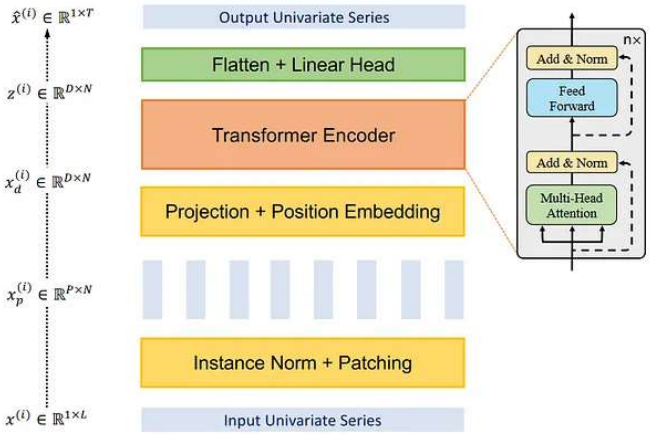

Figure 15: PatchTST Transformer Backbone Supervised **[38]**

On the transformer backbone, inputs are normalized with zero mean and unitary standard deviation and then patching is applied. Patches can be overlapping, or non-overlapping, and the number of patches depends on the length of the patch and the stride. These patches are tokenized by a linear projection and a position encoding and then used as input for the transformer. The transformer encoder has a traditional structure with a multi-head attention block and a feed forward network with residual connections [39]. Finally, a flatten layer with linear head is used to obtain the prediction result.

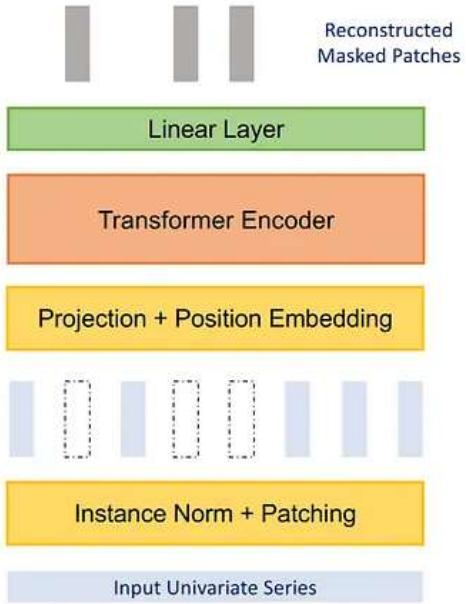

Figure 16: PatchTST Transformer Backbone Self-Supervised **[38]**

PatchTST also has a self-supervised transformer backbone, represented in Figure 16, where random patches are masked, and the model is trained to recreate the original patches. In this case, patches cannot be overlapped to ensure the transformer never gets information of the masked patches. The output layer is also different to reduce overfitting, and the prediction head is removed and replaced by a linear layer.

2.5.9 TiDe

In 2023, a team of researchers at Google lead by A. Das, introduced the *Time-series Dense Encoder* (TiDe). Similar to PatchTST, TiDe is applied in a channel independent manner but applies an encoder-decoder model built with MLPs instead of transformers. MLPs help the model achieve faster training and inference times [40].

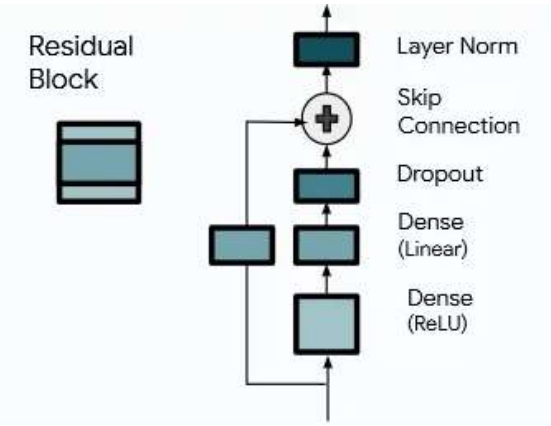

Figure 17: TiDe residual block **[40]**

As represented in Figure 17, TiDe uses residual blocks, which are used across the network. It is an MLP with one hidden layer with a ReLU activation followed by a dropout layer, a skip connection, and a normalization layer at the end.

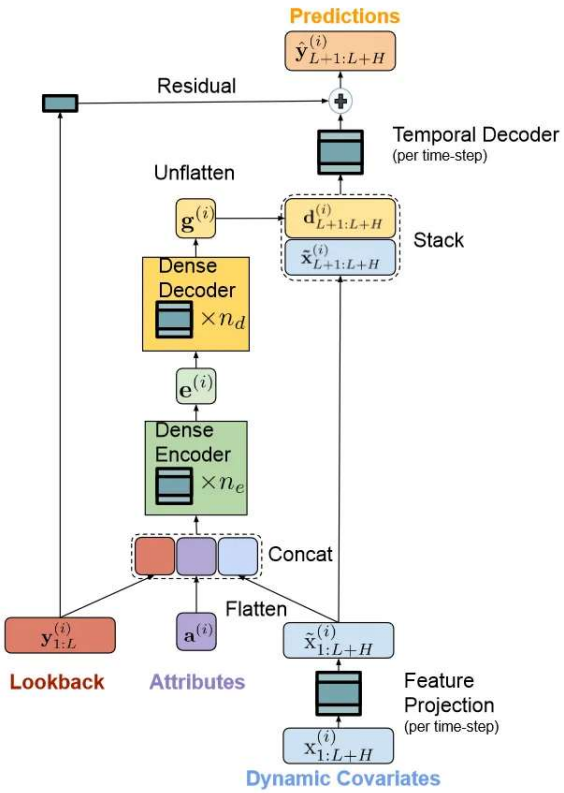

Figure 18: TiDe architecture **[40]**

The input of TiDe's architecture, illustrated in Figure 18, is a concatenation of lookback, attributes and exogenous variables. These variables, or covariates, go through a feature projection where the residual block maps them into a lower dimensional projection to reduce the size of the input vector. Then, a dense encode and a dense decoder, both built with stacks of residual blocks, are responsible for learning a representation of the inputs and output a matrix to be fed into a second decoder, the temporal decoder.

The temporal decoder maps a concatenation of the decoded output with projected features to an output size of 1. This step is seen as a "highway" for future covariates since some can have important impacts on the prediction.

2.6 Evaluation Measures

In time series forecasting, the accuracy of a model is measured by the difference between predicted values and actual values. This difference is called *error*, and it is easily confused with residuals. Residuals are calculated on the training set and based on one-step forecasts while errors are calculated on the test set and can involve multi-step forecasts [3].

The choice of error measure can significantly impact model evaluation, particularly when comparing models or optimizing them during training.

Measures like the mean absolute error (MAE) and the root mean squared error (RMSE), represented in equation (1) and (2) respectively, are scale-dependent measures since they are in the same scale as the data. They are easy to interpret but cannot be used to make comparisons between series that involve different units.

$$MAE = \frac{1}{n}\sum_{t=1}^{n}|y_t - \hat{y}_t|\,; \tag{1}$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{t=1}^{n}(y_t - \hat{y}_t)^2}; \tag{2}$$

$\boldsymbol{n}$: Number of observations, $\boldsymbol{y_t}$: Actual value at time t, $\boldsymbol{\hat{y}_t}$: Predicted value at time t

Percentage error measures, like mean absolute percentage error (MAPE) and symmetric mean absolute error (sMAPE), represented in equations (3) and (4) respectively, are more appropriate to compare forecast performances between series with different units since they are unit-free. The disadvantage of percentage errors is when the actual value is relatively small compared to the forecast error. The resulting percentage error becomes extremely large, which distorts the results of further analyses [41].

$$MAPE = \frac{1}{n}\sum_{t=1}^{n}\left|\frac{y_t - \hat{y}_t}{y_t}\right| \times 100 \tag{3}$$

$$sMAPE = \frac{100}{n} \sum_{t=1}^{n} \frac{|\hat{y}_t - y_t|}{|y_t| + |\hat{y}_t|} \times 100 \quad (4)$$

$\boldsymbol{n}$: Number of observations, $\boldsymbol{y_t}$: Actual value at time t, $\boldsymbol{\hat{y}_t}$: Predicted value at time t

Scaled errors, like mean absolute scaled error (MASE), represented in Equation (5), are an alternative to percentage errors. To be scale free, these measures remove scale by comparing the forecasts with a benchmark forecast method like the naïve method [42].

$$MASE = \frac{\frac{1}{n} \sum_{t=1}^{n} |y_t - \hat{y}_t|}{\frac{1}{n} \sum_{t=2}^{n} |y_t - y_{t-1}|}; \quad (5)$$

$\boldsymbol{n}$: Number of observations, $\boldsymbol{y_t}$: Actual value at time t, $\boldsymbol{\hat{y}_t}$: Predicted value at time t

The accuracy of a forecasting model can also be evaluated using the R^2 metric, also known as the coefficient of determination. R^2, represented in Equation (6), is a statistical measure that indicates the proportion of variance in the dependent variable explained by the independent variables in a regression model. It reflects how well the model fits the data, with values typically ranging from 0 to 1, where a higher R^2 value suggests a better fit and a more accurate model. However, when applied to nonlinear models, R^2 values can sometimes fall below 0 [43], indicating that the model's performance is poor [44].

$$R^2 = 1 - \frac{\sum (y_t - \hat{y}_t)^2}{\sum (y_t - \bar{y})^2} \quad (6)$$

$\boldsymbol{y_t}$: Actual value at time t, $\boldsymbol{\hat{y}_t}$: Predicted value at time t, $\boldsymbol{\bar{y}}$**:** Mean actual value

In summary, each error measure has strengths and weaknesses, and the selection of measures is depending on the specific context of the forecasting task. In this case, since only one unit is being forecasted, MAE, RMSE, MAPE and R^2 are used in this work.

3 Literature Review

Forecasting is a vast field with a rich history, where numerous models and approaches have been developed and tested across various domains over the years. Covering every aspect of forecasting is challenging, as it has evolved significantly, with advancements in both statistical and machine learning methodologies. To provide a structured overview, this literature review first examines some of the early, widely-used models that laid the foundation for modern forecasting. It then traces the key developments in the field, using a well-recognized competition in time series forecasting to highlight the main breakthroughs and conclusions reached over time.

The second part of this chapter shifts focus specifically to Bitcoin price forecasting, reviewing the methodologies that have been applied to this highly volatile market. This analysis aims to identify the trends, challenges, and future directions in the study of Bitcoin price prediction, offering insights into what has been accomplished so far and how the field is expected to evolve.

3.1 Time Series Forecasting

Time series forecasting had its first breakthrough, in 1927, when Udny Yule introduced the notion of stochasticity in time series [45]. This idea led to the development of methods adjusted for seasonality and trends with the autoregressive (AR) and moving average (MA) concepts, known as ARMA models. While statisticians were focused on ARMA models, business and industry were essentially using exponential smoothing methods due to their simplicity and effectiveness in short-term forecasting [46].

In 1970, George Box and Gwilym Jenkins [9] integrated the existing knowledge creating the Box–Jenkins approach and popularized the use of autoregressive integrated moving average (ARIMA) models [46].

To model volatility in econometrics and finance, in 1982, Robert Engle introduced the Autoregressive Conditional Heteroskedasticity (ARCH) model [47], which was later extended to the Generalized ARCH (GARCH) model by Tim Bollerslev in 1986 [48]. A GARCH model has an ARMA-type representation, so that the models share many properties [46].

With the emergence of different models, Spyros Makridakis created the Makridakis Competitions. Also known as the M Competition, the intent was to benchmark different forecasting methods, providing insights into their relative performance and encouraging the development of new techniques. In its first edition, in 1982, traditional statistical methods such as Exponential Smoothing and ARIMA were compared. The main finding was that simple methods often performed as well as or better than more complex methods. At that time results were criticized which led to subsequent M Competitions [49].

In the second edition, M2, the number of series was reduced and data from private companies was included to simulate real-world forecasting. Participants worked with the data providers and the results were improved by the combination of judgmental and statistical forecasts [50].

In 2000, the M3 competition included a larger number of time series and forecasting methods, including neural networks. The results showed that combining forecasts models often led to better accuracy [51].

After a long time without any competition, in 2018 the M4 competition was launched to test forecasting methods on a large scale, with 100,000 time series from finance, economics, demographics, and other areas. The best performing method was a combination of statistical and machine learning techniques, demonstrating the effectiveness of hybrid models [52].

The latest competition, in 2020, focused on practical applications with retail sales, hierarchical and grouped time series, and had multiple prize moneys to best performing methods. The competition highlighted the effectiveness of sophisticated machine learning models, such as LightGBM and confirmed that the accuracy can be improved by combining different methods [53].

With the latest success of machine learning methods in the M competition and the latest success of transformers in other areas such as natural language processing, speech recognition and computer vision, a new wave of models based in transformers have emerged since 2022, but none of them was able to be significantly more accurate than previous models and some researchers are questing the effectiveness of transformers in time series [54].

3.2 Bitcoin Price Forecast

Forecasting financial markets has been a subject of extensive research and some models, like ARCH, were developed specifically to solve the complexity involved in predicting market behaviour. Bitcoin is the most prominent cryptocurrency and, given its volatile nature, forecasting Bitcoin prices has become a highly active area of research.

The first studies on bitcoin price forecast were made in 2014 [55, 56], but the number of relevant studies increased rapidly after 2018 [57]. Since then, multiple studies have been made using statistical and machine learning methods to predict bitcoin prices [58].

Recently, however, research has shifted focus. Current studies emphasize feature engineering, incorporating social media [59] and on-chain data [60] to improve predictions, while less attention is being given to newer forecasting models like N-HITS and transformer-based approaches [61]. In fact, in 2023, LSTMs were the most used models for bitcoin price forecasting [62].

Creating a robust benchmark of metrics is challenging due to the limited research published in studies similar to this one. While metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Squared Error (RMSE) are often used to evaluate model performance, they are scale-dependent and therefore heavily influenced by the specific time periods and datasets used in each study. To enable better comparisons across studies, scale-free metrics are preferred.

Among the scale-free metrics, the most commonly used is the Mean Absolute Percentage Error (MAPE), as it normalizes the error relative to the actual values. In this study, MAPE is used as a key metric to evaluate the benchmark performance of each model and compare results with similar studies. Table 5 summarizes findings from other Bitcoin price forecasting studies, providing a comparative context for this research.

Table 1: Mape benchmark

Study	**Model**	**MAPE**
Bitcoin time series forecasting with exogenous factors: deep learning approach [63]	LSTM	3.1
	GRU	1.94
	CNN - BiLSTM	1.82
Forecasting Cryptocurrency Prices using Contextual ES-adRNN with Exogenous Variables [64]	ARIMA	3.38
	MLP	3.38
	DeepAR	3.30
	N-BEATS	3.27
Forecasting Bitcoin Prices Using N-BEATS Deep Learning Architecture [65]	N-BEATS	2.26
	LSTM	2.97
	ARIMA	2.28
A New Forecasting Framework for Bitcoin Price with LSTM [66]	LSTM	2.7
	LSTM + AR(2)	2.6
N-BEATS Perceiver: A Novel Approach for Robust Cryptocurrency Portfolio Forecasting [67]	N-HiTS	2.56
	N-BEATS	2.75

4 Data

Everything starts with data. In a time series study, like this one, data is the foundation, the raw material, and the heart of the entire process. The success of time series forecasting hinges on the quality of the data since it directly affects the performance and reliability of the models. Data is crucial because, without clean, consistent, and accurate data, even the best algorithms will fail to produce meaningful results.

The first step in any data-driven project is data collection. It involves gathering historical data related to the problem, ensuring it covers a broad enough range to capture patterns like seasonality, trends, and any anomalies that may occur over time.

Once the data is collected, it must be explored. Exploration helps to gain a better understanding of the dataset. Through exploratory data analysis, patterns, trends, and irregularities are identified, and potential relationships between variables can be spotted. Visualization tools like line plots, histograms, and box plots help to gain insights and recognize outliers or anomalies that might skew the results.

The next step is data preparation. Cleaning and transforming the raw data to make it usable by models. This process can include dealing with missing values, normalizing or scaling the data, and transforming it if needed. Proper data preparation ensures that the models receive well-structured, noise-free inputs, which is essential for reliable forecasting [17].

4.1 Collection

The core dataset needed to include a time series of Bitcoin (BTC) prices. After evaluating different data sources, *yfinance*[2] was selected. yfinance is an open-source tool that retrieves market data from the Yahoo! Finance API, commonly used for research and educational purposes.

The dataset was extracted using the ticker BTC-USD for the period between December 31, 2014, and July 4, 2024. Although the focus of the study is on data from 2015 onwards, data from the day before was included to calculate the price return for January 1, 2015, if needed.

The dataset returned by yfinance consisted of 3,474 rows and 7 columns specified in Table 2. The number of rows correspond exactly to the number of days requested, meaning we do not have missing days. Each column was analysed individually, confirming there were no missing values or duplicates. A small sample of data points was manually cross-referenced with other sources to ensure data accuracy.

[2] https://pypi.org/project/yfinance/

Table 2: BTC-USD data description

Colum Name	Data Type	Description
Date	Datetime64	Time step of the data. Since we are looking at daily data, we have one
Open	Float64	BTC opening price in US Dollars
High	Float64	Highest BTC price
Low	Float64	Lowest BTC price
Close	Float64	BTC closing price
Adj Close	Float64	Closing price after adjustments for all applicable splits and dividend distributions
Volume	Float64	Total volume of BTC transacted

4.2 Exploration

During the data exploration phase, basic statistics were calculated for all numeric variables to gain a better understanding of the available data. This metrics are presented in Table 3.

Table 3: BTC-USD data summary

Name	Mean	Std	Min	Max
Open	$ 17,415.45	$ 18,860.48	$ 176.90	$ 73,079.38
High	$ 17,807.74	$ 19,285.62	$ 211.73	$ 73,750.07
Low	$ 16,985.20	$ 18,389.26	$ 171.51	$ 71,334.09
Adj Close	$ 17,425.83	$ 18,868.03	$ 178.10	$ 73,083.50
Close	$ 17,425.83	$ 18,868.03	$ 178.10	$ 73,083.50
Volume	17,846,587,998	19,222,946,339	7,860,650	350,967,941,479

This study focuses on Bitcoin's price, specifically using the "Close" column for analysis, modelling, and prediction. To gain a deeper understanding of this key variable, a complete analysis was conducted.

Figure 19 presents a histogram that illustrates the distribution of Bitcoin's closing prices, helping to visualize the range of values and how frequently they occur within those ranges. This diagram shows a positively skewed distribution of the BTC prices, as the low prices are more frequent than the large ones.

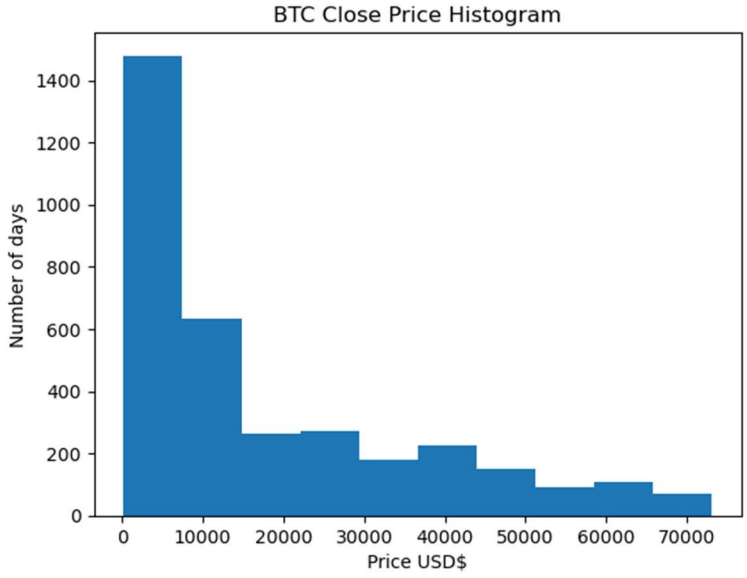

Figure 19: Bitcoin close price histogram

The box plot from Figure 20 reveals that Bitcoin's price distribution is heavily skewed, with a large portion of time spent at lower price levels, but with a noticeable presence of outliers at higher values. The median price is significantly below the upper whiskers and the outliers, indicating that the distribution is not symmetric. After closely examining these outliers, it was determined that they are not errors or anomalies, but instead reflect important market events. Consequently, these outliers were retained in the dataset without any transformations.

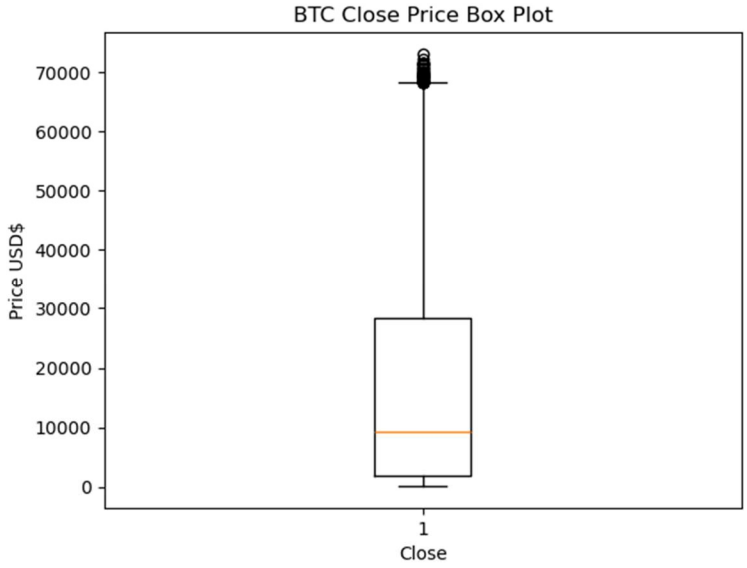

Figure 20: Bitcoin close price box plot

To better understand the typical price change from one day to the next, a new column called *Return* was calculated. In financial forecasting, returns are often used instead of raw prices because they are scale-free, allowing for easier comparisons between different types of investments [68]. Returns capture the percentage change in price, providing a more

standardized view of price movements across varying investment scales. The expression used to calculate returns is shown in Formula (7).

$$Return_{\ t} = \frac{y_t - y_{t-1}}{y_{t-1}}. \tag{7}$$

$\boldsymbol{y_t}$: Actual value at time t

The calculated returns, showed in Figure 21, reveal a significant level of volatility throughout the analysed period, with sharp spikes in both positive and negative directions. These rapid changes indicate the inherent volatility of Bitcoin.

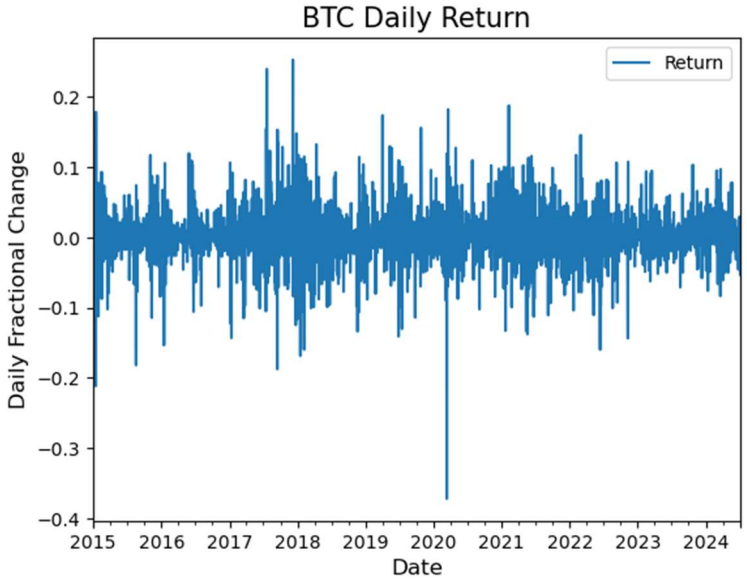

Figure 21: BTC daily return

Looking at the histogram of returns in Figure 22, the distribution is highly concentrated around 0, displaying a slight negative skew. This means that there are more days with negative returns than positive ones. The majority of daily returns are clustered between -0.05 and 0.05, indicating that most daily price movements are relatively small. However, there are a few extreme negative outliers beyond this range, highlighting occasional sharp swings in Bitcoin's value, which is consistent with its volatile nature.

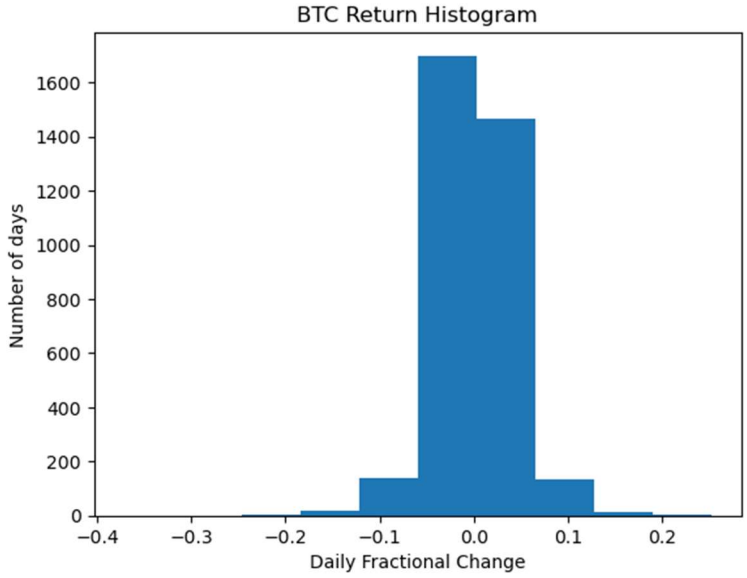

Figure 22: BTC daily return histogram

Analysing Figure 23, it is evident that the data is relatively symmetric, with the median return close to zero, indicating a balanced distribution of daily returns. However, the presence of several outliers suggests that Bitcoin has experienced a few days with extreme positive or negative returns, even though most daily changes are moderate. This pattern aligns with what was observed in the price data, reinforcing the decision to keep these outliers as they provide crucial information for accurate modelling and prediction.

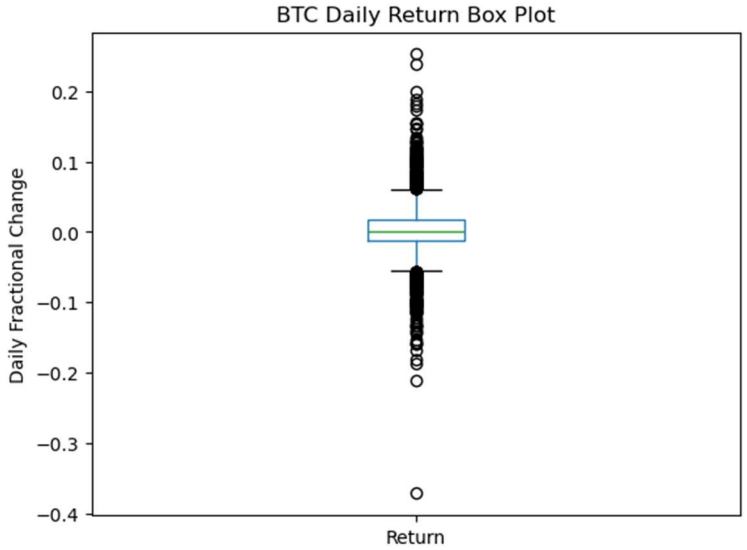

Figure 23: BTC daily return box plot

4.3 Preparation

The *train-validation-test* split is a widely adopted methodology in machine learning to evaluate a model's performance. It involves dividing the dataset into three distinct sets: training, validation, and test. This approach ensures that the model generalizes well to unseen data, preventing overfitting and providing a more accurate representation of how the model will perform in real-world scenarios.

Although there is no fixed rule for the ideal split ratio, common guidelines suggest allocating 60-80% of the data for training, 10-20% for validation, and another 10-20% for testing. The purpose of these splits is to train the model and the training set to learn patterns, relationships, and trend from the data. The performance of the model is then evaluated on the validation set during training to fine-tune hyperparameters. The test is never seen by the model during training and validation, and it is used to assess the model's final performance and ability to generalize to new data.

In this study, instead of following percentage-based splits, the data was split based on calendar years and it is represented in Figure 24: Train, validation and test sets split. The first 6 years, from the start of 2015 until the end of 2020 and representing 63% of the dataset, were used for training. The next 2 years, 2021 and 2022, accounting for 21%, were designated for validation and from 2023 onwards, the remaining 1 year, 7 months, and 4 days were allocated for testing, representing 16% of the data. This method ensures a realistic time-based division, which is crucial for time series forecasting, as it maintains the chronological integrity of the data, making predictions more reliable and reflective of real-world conditions.

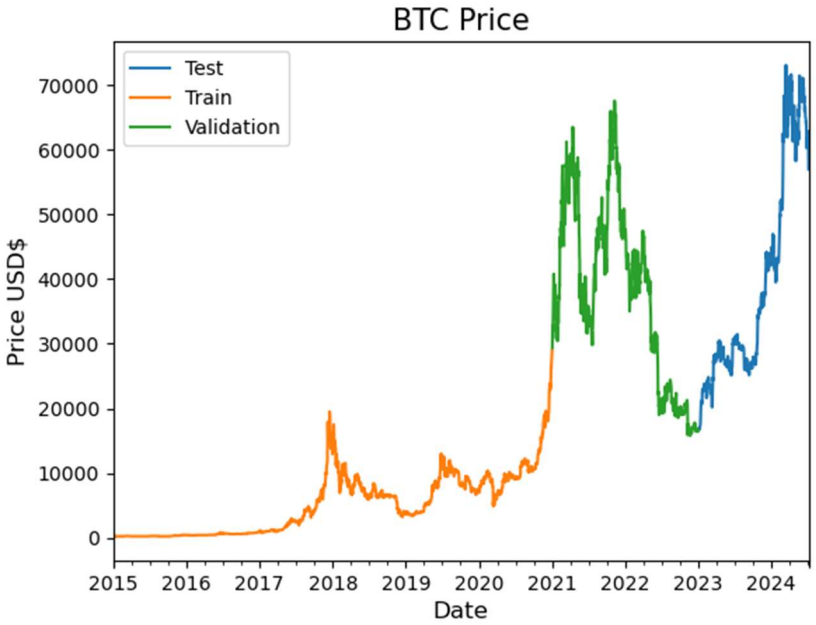

Figure 24: Train, validation and test sets split

5 Modelling

The modelling phase followed a similar structured approach for all models to ensure thorough exploration and comparison. Initially, different implementations of each model were tested using various Python libraries with default values. This helped to assess the tools provided by each library, compare results, and gain a deeper understanding of the model's mechanics. Once a suitable library was identified for each model, further testing was carried out by adjusting hyperparameters manually to observe their impact on performance. This process provided insights into key factors that could be optimized in the hyperparameter tuning phase.

5.1 Forecasting with Statistical models

Auto-ARIMA, an object from the *pmdarima*[3] library, was used to automatically select the best parameters for the ARIMA/SARIMA models. Auto-ARIMA begins by performing differencing tests to determine the appropriate level of differencing, and then fits models with a range of values for the AR (Auto Regressive) and MA (Moving Average) terms. The selected model is the model that minimizes the chosen criterion, AIC in this case.

Using training and validation subsets, the obtained model selected by Auto-Arima to predict bitcoin price for the next day was (2,1,2) (0,0,0) [0]. The full detail of the models tested by Auto-Arima are presented in Table 4.

[3] https://alkaline-ml.com/pmdarima/

Table 4: Auto-Arima models

Model	AIC	Time (sec)
(0,1,0) (0,0,0) [0]	47,412.568	0.07
(1,1,0) (0,0,0) [0]	47,412.655	0.13
(0,1,1) (0,0,0) [0]	47,412.674	0.11
(0,1,0) (0,0,0) [0]	47,410.706	0.02
(1,1,2) (0,0,0) [0]	47,415.095	0.58
(2,1,1) (0,0,0) [0]	47,416.612	0.32
(3,1,2) (0,0,0) [0]	47,396.054	3.79
(3,1,1) (0,0,0) [0]	47,415.624	0.94
(4,1,2) (0,0,0) [0]	47,418.157	2.32
(3,1,3) (0,0,0) [0]	47,397.774	3.41
(2,1,3) (0,0,0) [0]	47,396.114	3.32
(4,1,1) (0,0,0) [0]	47,416.129	1.00
(4,1,3) (0,0,0) [0]	47,400.117	4.98
(3,1,2) (0,0,0) [0]	47,394.376	2.88
(2,1,2) (0,0,0) [0]	**47,388.830**	**2.19**
(1,1,2) (0,0,0) [0]	47,412.366	0.20
(2,1,1) (0,0,0) [0]	47,414.755	0.17
(2,1,3) (0,0,0) [0]	47,394.281	1.54
(1,1,1) (0,0,0) [0]	47,412.787	0.09
(1,1,3) (0,0,0) [0]	47,413.272	0.75
(3,1,1) (0,0,0) [0]	47,413.387	0.65
(3,1,3) (0,0,0) [0]	47,393.913	2.08

The model (2,1,2) (0,0,0) [0] means the time series does not present seasonality. The $d = 1$ in the ARIMA model indicates that the series was differenced once to make it stationary. To confirm the stationarity of the series after differing by 1, an Augmented Dickey-Fuller (ADF) Test was performed. The ADF test identifies the presence of a unit root, as a unit root indicates that the time series is non-stationary and follows a stochastic trend. The null hypothesis of the ADF test is that the series contains a unit root (i.e., it is non-stationary), while the alternative hypothesis is that the series is stationary. In this case, the p-value of the test was 0, meaning that, regardless of the significance level, the null hypothesis is rejected, providing statistical evidence that the differentiated series is indeed stationary.

The $p = 2$ and $q = 2$ values in the ARIMA model indicate that the autoregressive (AR) component uses two lagged values of the time series, and the moving average (MA) component relies on two lagged forecast errors to predict the current value.

With the optimal parameters defined, ARIMA (2,1,2) (0,0,0) [0] was fitted using the class SARIMAX from the *statsmodels* [4] library. This library allows for the application of ARIMA

[4] https://www.statsmodels.org/

and SARIMA models with exogenous variables, which could be useful for future iterations of this study.

To evaluate the model in a real-world scenario, the test subset was assessed using a technique called *Expanding Window*, illustrated in Figure 25. In this approach, once the best model is selected, it is applied to the data available up to the point of the step we want to predict. With each iteration, more data points from the time series are added to the training set, effectively expanding the training window and increasing the amount of historical data available to the model, this method mirrors a realistic forecasting situation where more data becomes available over time.

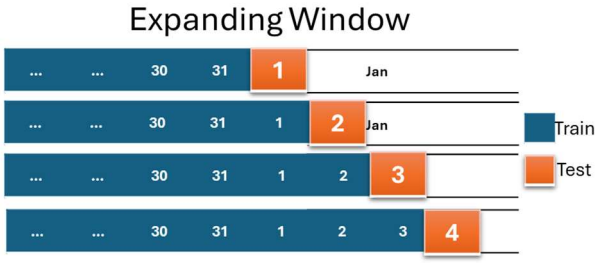

Figure 25: Expanding window

5.2 Forecasting with Neural networks based models

All neural networks based models were implemented using *NeuralForecast*[5], a library developed by Nixtla[6] . NeuralForecast provides scalable and user-friendly neural forecasting algorithms specifically designed for time series data. It was chosen for this study because it supports all the selected methods and simplifies implementation by allowing all models to be trained under consistent conditions.

The *cross_validation* method from the NeuralForecast library was used to perform the training and validation processes. This method defines the sliding windows, fits the model to the training dataset and then predicts the validation dataset. The step between each window was set to 1, ensuring that the entire dataset was covered. The window size is model specific and can be seen as a hyperparameter.

To look for optimal performance for each model, hyperparameters were fine-tuned. In this study, the neural networks based models' hyperparameters were optimized using *Optuna*[7], an open source hyperparameter optimization framework. Optuna offers an easy-to-use, scalable solution that integrates seamlessly with NeuralForecast.

Each model was initially trained individually with manual hyperparameter tuning to better understand its performance on the time series and to determine the number of training steps, or epochs, required for the training loss to stabilize. After this analysis, all models were fine-tuned using Optuna, with a maximum of 2,000 epochs and an early stopping patience of 10

[5] https://github.com/Nixtla/neuralforecast
[6] https://nixtlaverse.nixtla.io
[7] https://optuna.org/

steps which was verified every 50 epochs. The early stopping mechanism halted training if the accuracy did not improve over 10 consecutive steps, reducing the number of steps and optimizing resource usage during training. MAE was defined as the metric to be optimized by Optuna since it is the default loss applied by the NeuralForecast library. DeepAR is the only exception since it predicts the parameters of a probability distribution, so the default loss is the Multi Quartile Loss (MQLoss) [69].

Another approach was tested with the neural network models, where instead of directly predicting Bitcoin prices, these models were used to predict bitcoin's daily returns. Once the return was predicted by the model, it is then applied to the previous day's price to calculate the accuracy metrics. This transformation aligns the error metrics of both scenarios, making the comparison of these them more straightforward and consistent.

Neural networks based models involve randomness on the weight initialization, so using the same random number generators seeds ensures that results are replicable. This is crucial for debugging, validation, and comparing models [70]. For the hyperparameters optimization process, seed 1 was used for all models. After the optimization process, the best hyperparameters were used to fit the model 30 times with sequential seeds to assess how each model performs under different conditions. The performance metrics of these models are the mean value from these 30 executions. While this approach does not guarantee perfect hyperparameter selection due to vast search space, other methods would be computationally prohibitive for this study [71].

Next, we use N-BEATS as an example to present how the hyperparameters were selected and how the process was conducted. The full details of each model are provided in Appendix A.

From all parameters available to N-BEATS, available on NeuralForecast[8] official documentation, the *scaler type*, *input size*, *learning rate*, and *batch size* were fine tuned. For input normalization, models could use either an *identity*, *standard*, or *MinMax* scaler. The identity scaler acts as a placeholder, applying no transformation. The standard scaler normalizes inputs to have a zero mean and unit variance, while the MinMax scaler standardizes data by scaling values between 0 and 1. The input size, which represents the number of time steps used for each prediction, was tested with 2, 7, 15, or 30. These values were chosen to determine whether the models would perform better with weekly, biweekly, or monthly data windows. The learning rate was sampled logarithmically from a range between 1e-6 and 1e-2, while batch size options included 64, 128, 256, 512, or 1024. The best hyperparameters selected are available on Figure 26.

One of the reasons to choose Optuna to automatically do the hyperparameter optimization was its robust visualization tools, which provide insights into the optimization process. For example, in Figure 26, the optimization history is represented, showcasing how the objective values evolve over the course of the optimization.

[8] https://nixtlaverse.nixtla.io/neuralforecast/models.nbeats.html

In this particular case, most of the trials show objective values that closely align with the best-performing value, indicating a stable and consistent optimization process. This suggests that the search space and hyperparameter ranges were well-defined, leading to efficient exploration of potential solutions. The visualization not only helps track the progress but also provides a clearer picture of how many trials reached near-optimal results, reinforcing confidence in the robustness of the selected hyperparameters.

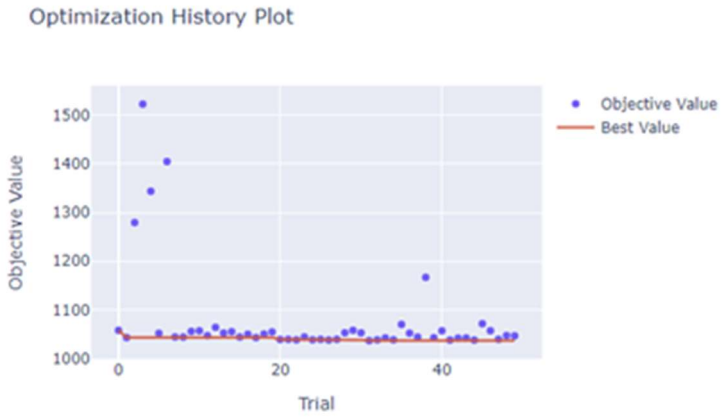

Figure 26: N-BEATS optimization history.

In Figure 27 the importance of each hyperparameter for the N-BEATS model is represented. From this visualization, it is clear that the learning rate stands out as the most influential hyperparameter for the model's performance.

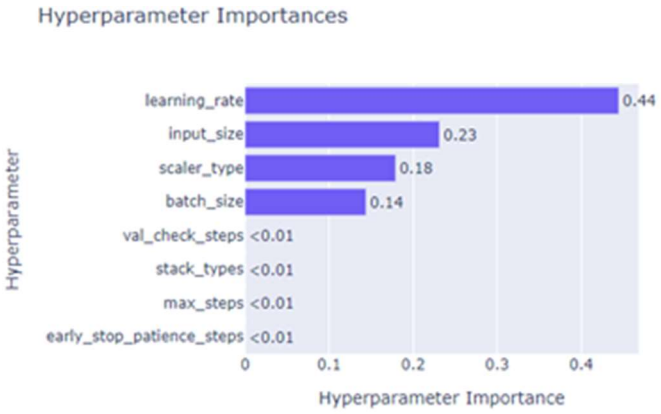

Figure 27: N-BEATS hyperparameter importance

After the optimization process, N-BEATS was trained with the best-found combination of hyperparameters. As shown in Figure 28, the training loss exhibited a small decrease in the initial epochs before stabilizing within a narrow range of values. Meanwhile, the validation loss stabilized after around 50 epochs and then maintained stability throughout the remainder of the training process. It is also visible that the training process triggered the early stopping after 950 epochs.

This suggests that the model quickly learned the key patterns in the data, and further training yielded diminishing returns, which is why early stopping was activated to prevent overfitting and save computational resources.

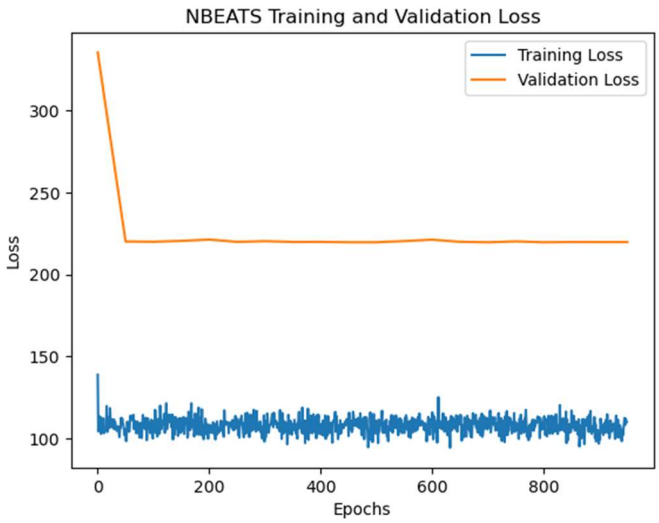

Figure 28: N-BEATS training and validation losses

N-BEATS was subsequently tested to model returns, using the same search space for hyperparameters as in the previous experiment. The selected hyperparameters from this search are summarized in Table 5. This experiment aimed to investigate how N-BEATS would handle predicting daily returns instead of prices, providing a different perspective on bitcoin's volatility and price fluctuations.

Table 5: N-BEATS hyperparameters to model price

Parameter	Search Range	Price Model	Return Model
Scaler Type	Identity; Standard; Minmax	Identity	Identity
Input Size	2; 7; 15; 30	2	2
Learning Rate	From 1e-6 to 1e-2	0.000057187	0.000182030
Batch Size	64; 128; 256, 512, 1024	128	1024

The scaler type selected for both models was identity, meaning neither model applied any data scaling. This choice ensures that the models worked directly with the raw data without altering the scale of Bitcoin prices or returns. For both models, the input size was kept minimal at 2, indicating a reliance on very short-term historical data to make predictions.

However, the models diverged when it came to learning rate and batch size. The model predicting returns used a faster learning rate and a significantly larger batch size, suggesting that it required a more aggressive yet stable learning process to handle the inherent volatility and quick shifts in bitcoin's daily returns.

Both models were trained 30 times, each time with a sequential random seed to account for variability in the training process. After training, the models predicted the test subset, and the results of each model represent the statistical mean of those 30 runs. This method ensures

that the model's performance is not skewed by the randomness inherent in neural network training, providing a more reliable and robust estimate of its predictive capabilities. By averaging the results across multiple runs, we mitigate the impact of outliers or unusually high or low performance from any single run.

6 Results

The evaluation of the models was conducted by testing the model predictions on the test subset of the data, using MAE, RMSE, MAPE, and R^2 as performance metrics. The results presented for neural networks models are a statistical mean of 30 models ran with sequential random seeds to ensure performance is not skewed by the randomness inherent in neural networks.

All models tested were trained to predict bitcoin prices for the next day and the summarized results are presented in Table 6.

Table 6: Results for price models

Model	MAE	MSE	RMSE	MAPE	R^2
ARIMA	712.59	1.3 e+06	1,146.29	1.708	0.99512
MLP	728.15	1.3 e+06	1,149.98	1.758	0.99508
LSTM	1,329.56	3.7 e+06	1,868.20	3.190	0.98638
GRU	2,243.15	10.0 e+06	2,831.00	5.506	0.96297
DeepAR	1,244.26	4.0 e+06	1,785.30	3.087	0.98545
N-BEATS	713.98	1.3 e+06	1,140.90	1.717	0.99517
N-HITS	715.48	1.3 e+06	1,141.56	1.721	0.99516
PatchTST	**709.91**	1.3 e+06	1,138.37	**1.702**	**0.99519**
TiDE	712.80	**1.3** e+06	**1,137.73**	1.711	**0.99519**

PatchTST and TiDE have the lowest error rates and highest R^2 scores, indicating the most accurate predictions. ARIMA, N-BEATS, and N-HITS perform comparably well, with only marginally higher error rates. LSTM, GRU, and DeepAR have higher errors and lower R^2, indicating lower predictive accuracy compared to the other models.

Since the results for neural network models are the statistical mean of 30 individual runs, the skewness of each model's performance should also be considered to provide a more complete evaluation. The variability or inconsistency in model performance can be measured using standard deviation, which helps to understand how much the results deviate from the mean across different runs.

Table 7 presents the standard deviation of each model's performance, giving insight into the reliability and stability of the models. A lower standard deviation indicates that the model consistently produces similar results across different runs, whereas a higher standard deviation suggests more variability and potential sensitivity to random initialization.

Table 7: Neural networks price model metrics standard deviation

Model	MAE	MSE	RMSE	MAPE	R^2
MLP	18.7	2.6 e+04	11.2	0.054	0.000
LSTM	359.6	17 e+06	428.5	0.937	0.006
GRU	1,280.7	1.1 e+07	1,423.1	3.156	0.041
DeepAR	713.6	4,1 e+06	870.4	1.814	0.015
N-BEATS	5.3	5.4 e+03	2.4	0.016	0.000
N-HITS	7.9	8.7 e+03	3.8	0.023	0.000
PatchTST	1.4	6.2 e+03	2.7	0.004	0.000
TiDE	1.8	2,1 e+03	0.9	0.004	0.000

LSTM, GRU, and DeepAR models exhibit higher standard deviation in their performance metrics, indicating that they are more sensitive to the inherent randomness of their internal processes, such as weight initialization and dropout layers. This suggests that these models' predictions can vary significantly across different runs, relying heavily on the random seed used during training. As a result, their performance may be less stable compared to models with lower standard deviation, which could impact their reliability in real-world forecasting tasks.

To address the instability observed in price predictions, neural network models were also tested on predicting daily returns instead of prices. This approach aims to reduce the sensitivity to randomness by focusing on the relative change rather than the absolute price values. Once the returns were predicted, they were then applied to the previous day's price to derive the predicted price. The results of this approach are summarized in Table 8.

Table 8: Results for return models

Model	MAE	MSE	RMSE	MAPE	R^2
MLP	716.03	1.31 e+06	1,146.04	1.719	0.99512
LSTM	724.10	1.33 e+06	1,152.84	1.738	0.99506
GRU	715.16	1.31 e+06	1,146.56	1.714	0.99511
DeepAR	745.17	1.37 e+06	1,170.50	1.796	0.99491
N-BEATS	713.51	1.31 e+06	1,145.68	1.717	0.99512
N-HITS	713.97	1.29 e+06	1,134.28	1.721	0.99522
PatchTST	738.71	1.35 e+06	1,160.03	1.785	0.99500
TiDE	714.55	1.30 e+06	1,141.90	1.717	0.99516

The overall performance of the models when predicting returns is very similar between them. Figure 29 and Figure 30 provide a comparative view of the models' performance when predicting returns versus direct price forecasting. Significant improvements can be observed in models like LSTM, GRU, and DeepAR. These models, which initially exhibited higher variance and sensitivity to randomness in direct price prediction, demonstrated more stability and improved results when tasked with forecasting returns.

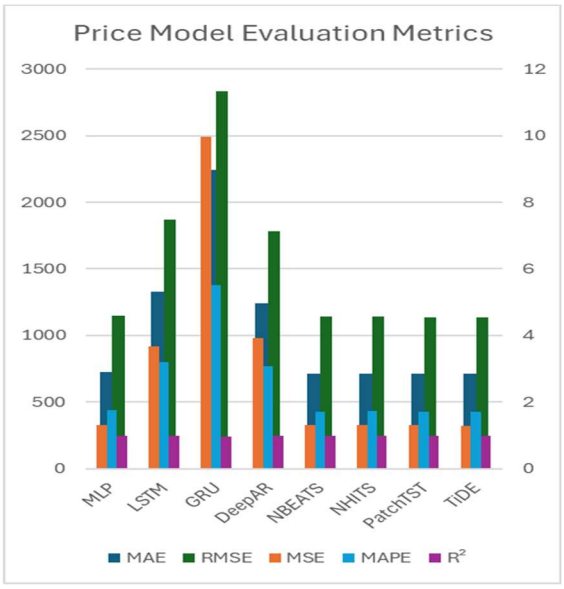

Figure 29: Price model evaluation metrics

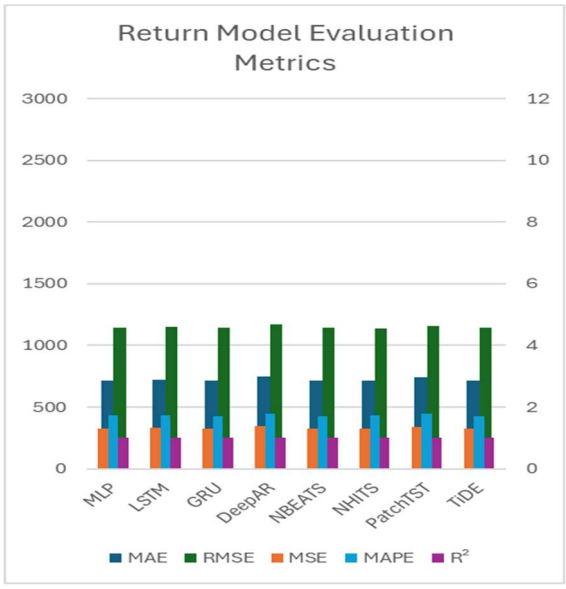

Figure 30: Return model evaluation metrics

Table 9 shows the standard deviation of the return model metrics, and the improvement is clear across the board. While LSTM still exhibits more variability than other models, the scale of this variability is significantly lower compared to the direct price prediction scenario. This demonstrates that predicting returns, rather than prices, helps reduce the inherent randomness and volatility in models like LSTM.

Table 9: Neural networks return model metrics standard deviation

Model	MAE	MSE	RMSE	MAPE	R^2
MLP	2.8	6.3 e+03	2.8	0.005	0.0000
LSTM	12.3	1.5 e+04	6.7	0.036	0.0001
GRU	2.2	4.8 e+03	2.1	0.005	0.0000
DeepAR	5.8	1.7 e+04	7.4	0.013	0.0001
N-BEATS	8.6	1.1 e+04	4.7	0.024	0.0000
N-HITS	2.9	4.7 e+03	2.1	0.008	0.0000
PatchTST	4.4	1.1 e+03	4.8	0.009	0.0000
TiDE	2.6	2.4 e+03	1.1	0.007	0.0000

Looking closely at the output of each model and using DeepAR as an example, Figure 31 illustrates the actual and predicted outputs for return forecasting. The predicted returns generally hover around 0, showing a tendency for the model to anticipate small or no changes in returns. However, the actual data exhibits numerous sharp spikes, both positive and negative, indicating significant volatility.

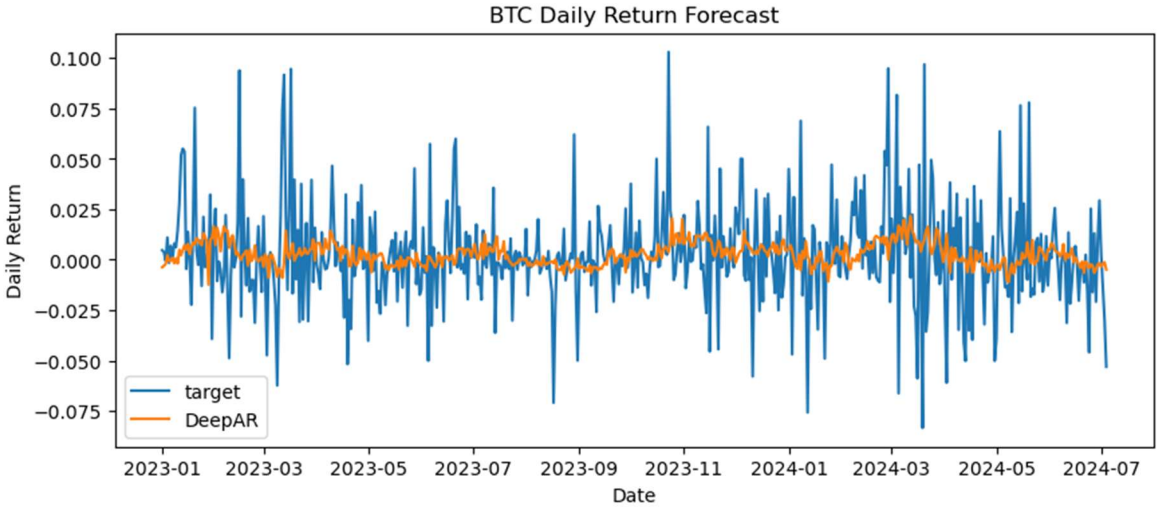

Figure 31: DeepAR returns actuals vs predicted

This behaviour might initially suggest that predicting returns is less effective than predicting prices due to the model's difficulty in capturing extreme fluctuations. However, once these predicted returns are transformed into prices, as illustrated in Figure 32, and compared with the direct price prediction model, as shown in Figure 33, the improvement becomes noticeable. The transformation from returns to prices smooths out some of the volatility and provides more accurate forecasts, indicating that while the return model struggles with sharp spikes, it still manages to capture the overall price trend more effectively than the direct price prediction model.

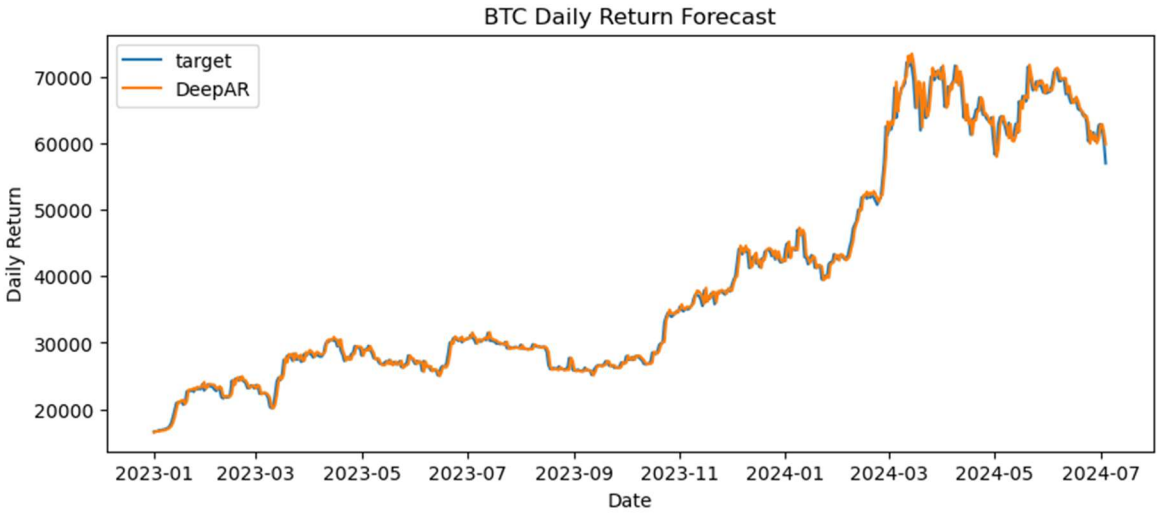

Figure 32: DeepAR return model forecast

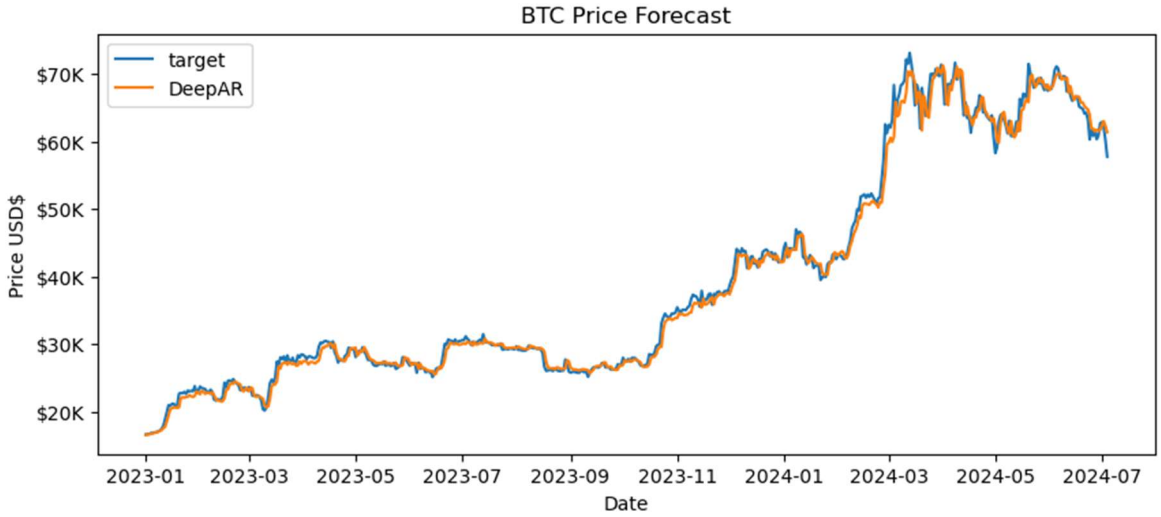

Figure 33: DeepAR price model forecast

An overall analysis of the best results for each model, as shown in Table 10, reveals that all models performed similarly in terms of accuracy. MAPE ranged from 1.7% to 1.8%, and the R^2 scores, which indicate how well the predicted values match the actual data, ranged from 0.995 to 0.996. These results demonstrate that despite differences in model architecture and approach, all models delivered a high performance level, with only marginal differences in performance across the board.

Table 10: Results by model

Model	Method	MAE	MSE	RMSE	MAPE	R^2
ARIMA	Price	712.59	1.31 e+06	1,146.29	1.708	0.99512
MLP	Return	716.03	1.31 e+06	1,146.04	1.719	0.99512
LSTM	Return	724.1	1.33 e+06	1,152.84	1.738	0.99506
GRU	Return	715.16	1.31 e+06	1,146.56	1.714	0.99511
DeepAR	Return	745.17	1.37 e+06	1,170.5	1.796	0.99491
N-BEATS	Price	713.98	1.3 e+06	1,140.9	1.717	0.99517
N-HITS	Return	713.97	**1.29 e+06**	**1,134.28**	1.721	**0.99522**
PatchTST	Price	**709.91**	1.3 e+06	1,138.37	**1.702**	0.99519
TiDE	Price	712.8	**1.29 e+06**	1,137.73	1.711	0.99519

N-HITS, PatchTST, and TiDE achieved the lowest errors and highest R^2 scores in the evaluation. Notably, both transformer-based models, PatchTST and TiDE, showed the best performance when predicting bitcoin prices, while N-HITS performed better when predicting returns.

Other models, including ARIMA, MLP, GRU, and N-BEATS, also performed well, with only slight differences in their accuracy metrics. However, DeepAR exhibits higher errors, making it less competitive compared to the other models in both price and return predictions.

Although accuracy metrics such as MAPE show low values, around 1.7%, and suggest that the model can effectively follow the overall price trend, a deeper examination of the returns

reveals a critical limitation. These models struggle to predict short-term spikes in Bitcoin's price. This limitation becomes evident when zooming into the last 60 days, as shown in Figure 34 and Figure 35: N-HITS return forecast last 60 days. In these figures, we can observe how the models tend to smooth out the sharp price fluctuations, failing to capture the sudden and significant price movements. This pattern is observed across all models tested, suggesting a common challenge in accurately forecasting the high volatility inherent in bitcoin's behaviour.

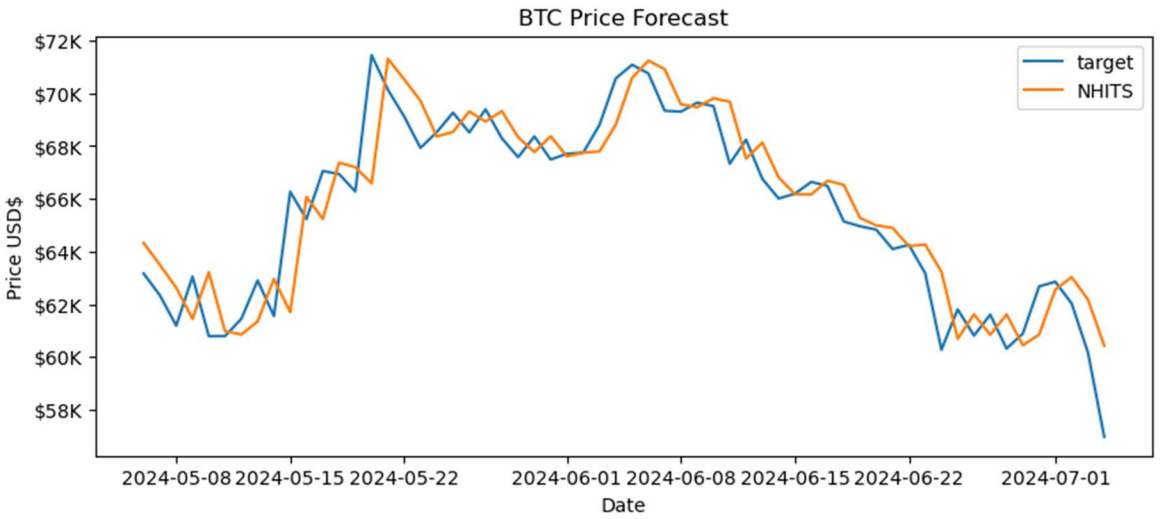

Figure 34: N-HITS price forecast last 60 days

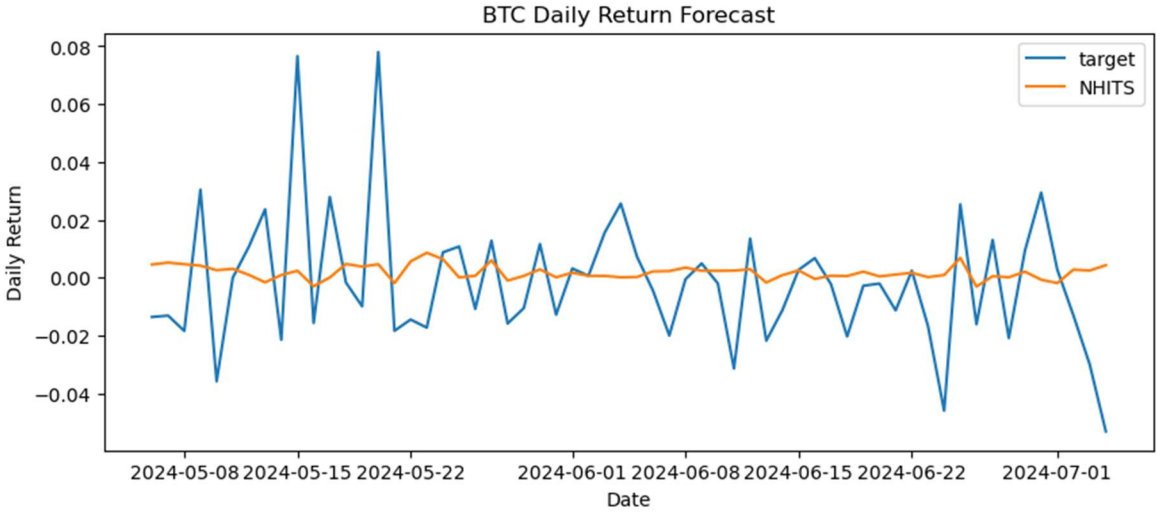

Figure 35: N-HITS return forecast last 60 days

7 Conclusion

The intent of this study was to predict Bitcoin's next-day price. The methodology began by framing the problem and identifying methods to address it, supported by a comprehensive literature review. This was followed by a data-driven step that included data gathering, exploration, and preparation for the modelling stage.

In the modelling stage, a variety of time series models were tested, ranging from statistical models to state-of-the-art neural networks. These models were evaluated both by predicting prices directly and by predicting daily returns, which were later transformed into price to ensure comparability between the two approaches. All models underwent an optimization process to ensure the best possible performance. The performance was measured using commonly applied performance metrics: mean absolute error (MAE), mean squared error (MSE), root mean squared error (RMSE), mean absolute percentage error (MAPE), and R-squared (R^2).

The approach of modelling returns and then converting them to price led to significant improvements in older models like LSTM and GRU, not only in terms of performance but also by increasing model stability and reducing the impact of randomness inherent in neural networks.

PatchTST, when modelling price directly, delivered the best results, with a MAE of $709.91 and a MAPE of 1.702%, although all models produced similar results.

While the results align with the reviewed literature, modelling returns revealed the difficulty models faced in predicting daily price swings. They struggled to forecast significant price movements based solely on Bitcoin’s price history.

This highlights the importance of ongoing research into exogenous variables and feature engineering, as incorporating additional data could help models capture Bitcoin's sudden and significant price movements.

This study also raised new questions. Since models have difficulty with sharp daily movements, how would they perform in a multi-step forecast? Would lowering the granularity of the time series help smooth out price fluctuations and improve predictions?

Additionally, exploring models designed to predict volatility, such as GARCH (Generalized Autoregressive Conditional Heteroskedasticity), could offer valuable insights. Since GARCH models are tailored for volatility forecasting, they might perform better in capturing the sharp fluctuations in Bitcoin's daily returns and price dynamics.

8 References

[1] W. J. Frawley, G. Piatetsky-Shapiro and C. J. Matheus, “Knowledge Discovery in Databases: An Overview,” *AI Magazine,* vol. 13, pp. 57-70, 1992.

[2] P. Chapman, J. Clinton, R. Kerber, T. Khabaza, T. Reinartz, C. Shearer and R. Wirth, “CRISP-DM 1.0 Step-by-step data mining guide,” SPSS Inc., 2000.

[3] R. J. Hyndman and G. Athanasopoulos, Forecasting: principles and practice, 3rd edition, Melbourne, Australia: OTexts, 2021.

[4] S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System,” 2008.

[5] J. Yli-Huumo, D. Ko, S. Choi, S. Park and K. Smolander, “Where Is Current Research on Blockchain Technology?,” *A Systematic Review. PLoS ONE,* vol. 11, no. 10, 2016.

[6] A. M. Antonopoulos, Mastering Bitcoin, O'Reilly Media, Inc., 2014.

[7] N. Houy, “The Bitcoin Mining Game,” *Ledger,* vol. 1, pp. 53-68, 2016.

[8] D. G. Baur, K. Hong and A. D. Lee, “Bitcoin: Medium of exchange or speculative assets?,” *Journal of International Financial Markets, Institutions and Money,* vol. 54, pp. 177-189, 2018.

[9] G. E. P. Box and G. M. Jenkins, Time Series Analysis: Forecasting and Control, Holden-Day, 1970.

[10] F. Petropoulos, D. Apiletti, V. Assimakopoulos, M. Z. Babai, D. K. Barrow, S. B. Taieb, C. Bergmeir, R. J. Bessa, J. Bijak, J. E. Boylan, J. Browell, C. Carnevale, J. L. Castle and P. Ciri, “Forecasting: theory and practice,” *International Journal of Forecasting,* vol. 38, no. 3, pp. 705-871, 2022.

[11] R. H. Shumway and D. S. Stoffer, Time Series Analysis and Its Applications, Springer Cham, 2017.

[12] I. Goodfellow, Y. Bengio and A. Courville, Deep Learning, MIT Press, 2016.

[13] K. P. Murphy, Machine Learning: A Probabilistic Perspective, The MIT Press, 2012.

[14] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, 2nd Edition, The MIT Press, 2018.

[15] P. Domingos, "A few useful things to know about machine learning," *Communications of the ACM,* vol. 55, no. 10, pp. 78-87, 01 October 2012.

[16] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science,* vol. 349, no. 6245, pp. 255-260, 2015.

[17] J. Brownlee, Introduction to Time Series Forecasting With Python: How to Prepare Data and Develop Models to Predict the Future, Machine Learning Mastery, 2017.

[18] K. Hornik, M. Stinchcombe and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks,* vol. 2, no. 5, pp. 359-366, 1989.

[19] G. Zhang, B. E. Patuwo and M. Y. Hu, "Forecasting with artificial neural networks: The state of the art," *International Journal of Forecasting,* vol. 14, no. 1, pp. 35-62, 1998.

[20] M. J. C. Hu and H. E. Root, "An Adaptive Data Processing System for Weather Forecasting," *Journal of Applied Meteorology and Climatology,* vol. 3, no. 5, p. 513–523, 1964.

[21] J. M. Oliveira and P. Ramos, "Evaluating the Effectiveness of Time Series Transformers for Demand Forecasting in Retail," *Mathematics,* vol. 12, no. 17, 2024.

[22] Y. Ahmadov and P. Helo, "Deep learning-based approach for forecasting intermittent online sales," *Discov Artif Intell,* vol. 3, no. 45, 2023.

[23] M. Soam and S. Thakur, "Next Word Prediction Using Deep Learning: A Comparative Study," in *12th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, Noida, India, 2022.

[24] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review,* vol. 65, no. 6, p. 386–408, 1958.

[25] S. Haykin, Neural Networks: A Comprehensive Foundation, Prentice Hall, 1999.

[26] S. R. Das, Data science: Theories, models, algorithms, and analytics, 2016.

[27] D. E. Rumelhart, G. E. Hinton and R. J. Williams, "Learning representations by back-propagating errors," *Nature,* vol. 323, no. 6088, pp. 533-536, 1986.

[28] J. Schmidhuber, "Deep learning in neural networks: An overview.," *Neural Networks,* vol. 61, pp. 85-117, 2015.

[29] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation,* vol. 9, no. 8, p. 1735–1780, 1997.

[30] F. A. Gers, J. Schmidhuber and F. Cummins, "Learning to forget: Continual prediction with LSTM," *Neural Computation,* vol. 12, no. 10, pp. 2451-2471, 2020.

[31] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional LSTM and other neural network architectures.," *Neural networks,* vol. 18, no. 5-6, pp. 602-610, 2005.

[32] K. Cho, B. v. Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk and Y. Bengio, "Learning Phrase Representations using RNN Encoder–Decoder," in *Conference on Empirical Methods in Natural Language Processing*, Doha, Qata, 2014.

[33] J. Chung, C. Gulcehre, K. Cho and Y. Bengio, "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling," in *Presented in NIPS 2014 Deep Learning and Representation Learning Workshop*, San Diego, CA, 2014.

[34] D. Salinas, V. Flunkert, J. Gasthaus and T. Januschowski, "DeepAR: Probabilistic Forecasting with Autoregressive Recurrent Networks," *International Journal of Forecasting,* vol. 36, no. 3, pp. 1181-1191, 2020.

[35] B. N. Oreshkin, D. Carpov, N. Chapados and Y. Bengio, "N-BEATS: Neural basis expansion analysis for interpretable time series forecasting," in *International Conference on Learning Representations*, 2020.

[36] K. G. Olivares, C. Challu, G. Marcjasz, R. Weron and A. Dubrawski, "Neural basis expansion analysis with exogenous variables: Forecasting electricity prices with NBEATSx," *International Journal of Forecasting,* vol. 39, no. 2, pp. 884-900, 2023.

[37] C. Cristian, K. G. Olivares, B. N. Oreshkin, F. Garza, M. M. Canseco and A. Dubrawski, "NHITS: Neural Hierarchical Interpolation for Time Series Forecasting," *AAAI,* vol. 37, no. 6, pp. 6989-6997, 2023.

[38] Y. Nie, N. H. Nguyen, P. Sinthong and J. Kalagnanam, "A Time Series is Worth 64 Words: Long-term Forecasting with Transformers," in *International Conference on Learning Representations*, 2023.

[39] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser and I. Polosukhin, "Attention is All you Need," in *Advances in Neural Information Processing Systems*, 2017.

[40] A. Das, W. Kong, A. Leach, S. K. Mathur, R. Sen and R. Yu, "Long-term Forecasting with TiDE: Time-series Dense Encoder," *Transactions on Machine Learning Research,* pp. 2835-8856, 2023.

[41] A. Davydenko and R. Fildes, "Measuring forecasting accuracy: The case of judgmental adjustments to SKU-level demand forecasts," *International Journal of Forecasting,* vol. 29, no. 3, pp. 510-522, 2013.

[42] R. J. Hyndman and A. B. Koehler, "Another look at measures of forecast accuracy," *International Journal of Forecasting,* vol. 22, no. 4, pp. 679-688, 2006.

[43] A. C. Cameron and F. A. Windmeijer, "An R-squared measure of goodness of fit for some common nonlinear regression models," *Journal of Econometrics,* vol. 77, no. 2, pp. 329-342, 1997.

[44] D. Chicco, M. J. Warrens and G. Jurman, "The coefficient of determination R-squared is more informative than SMAPE, MAE, MAPE, MSE and RMSE in regression analysis evaluation," *PeerJ Computer Science,* vol. 7, p. 623, 2021.

[45] G. U. Yule, "On a method of investigating periodicities disturbed series, with special reference to Wolfer's sunspot numbers," *Philosophical Transactions of the Royal Society of London, Series A,* pp. 267-298, 01 January 1927.

[46] J. G. De Gooijer and R. J. Hyndman, "25 years of time series forecasting," *International Journal of Forecasting,* vol. 22, no. 3, pp. 443-473, 2006.

[47] R. F. Engle, "Autoregressive Conditional Heteroscedasticity with Estimates of the Variance of United Kingdom Inflation," *Econometrica,* pp. 987-1007, 1982.

[48] T. Bollerslev, "Generalized autoregressive conditional heteroskedasticity," *Journal of Econometrics,* pp. 307-327, 1986.

[49] S. Makridakis, A. Andersen, R. Carbone, R. Fildes, M. Hibon, R. Lewandowski, J. Newton, E. Parzen and R. Winkler, "The accuracy of extrapolation (time series) methods: Results of a forecasting competition," *Journal of Forecasting,* vol. 1, no. 2, pp. 111-153, 1982.

[50] S. Makridakis, C. Chatfield, M. Hibon, M. Lawrence, T. Mills, K. Ord and L. F. Simmons, "The M2-competition: A real-time judgmentally based forecasting study," *International Journal of Forecasting,* vol. 9, no. 1, pp. 5-22, 1993.

[51] S. Makridakis and M. Hibon, "The M3-Competition: results, conclusions and implications.," *International Journal of Forecasting,* vol. 16, no. 4, pp. 451-476, 2000.

[52] S. Makridakis, E. Spiliotis and V. Assimakopoulos, "The M4 Competition: Results, findings, conclusion and way forward," *International Journal of Forecasting,* vol. 34, no. 4, pp. 802-808, 2018.

[53] S. Makridakis, E. Spiliotis and V. Assimakopoulos, "M5 accuracy competition: Results, findings, and conclusions," *International Journal of Forecasting,* vol. 38, no. 4, pp. 1346-1364, 2022.

[54] A. Zeng, M. Chen, L. Zhang and Q. Xu, "Are Transformers Effective for Time Series Forecasting?," *AAAI,* vol. 37, no. 9, pp. 11121-11128, 2023.

[55] D. Shah and K. Zhang, "Bayesian regression and Bitcoin," in *Fifty-second Annual Allerton Conference*, Illinois, USA, 2014.

[56] I. Madan, S. Saluja and A. Zhao, "Automated Bitcoin Trading via Machine Learning Algorithms," 2014.

[57] Y.-S. Ren, C.-Q. Ma, X.-L. Kong, K. Baltas and Q. Zureigat, "Past, present, and future of the application of machine learning in cryptocurrency research," *Research in International Business and Finance,* vol. 63, 2022.

[58] I. Kervancı and M. Akay, "Review on Bitcoin Price Prediction Using Machine Learning and Statistical Methods," *SAKARYA UNIVERSITY JOURNAL OF COMPUTER AND INFORMATION SCIENCES,* vol. 3, no. 3, 2020.

[59] A. Y. Nouira, M. Bouchakwa and M. Amara, "Role of social networks and machine learning techniques in cryptocurrency price prediction: a survey," *Social Network Analysis and Mining,* vol. 14, no. 152, pp. 1869-5469, 2024.

[60] O. Omole and D. Enke, "Deep learning for Bitcoin price direction prediction: models and trading strategies empirically compared," *Financial Innovation,* vol. 10, no. 117, 2024.

[61] D. L. John, S. Binnewies and B. Stantic, "Cryptocurrency Price Prediction Algorithms: A Survey and Future Directions," *Forecasting,* vol. 6, pp. 637-671, 2024.

[62] S. Bistarelli, F. Santini and L. M. Tutino, "A Short Survey on Bitcoin Price Prediction," 2024.

[63] A. A. Catarino, "Bitcoin time series forecasting with exogenous factors : deep learning approach," 2023.

[64] S. Smyl, G. Dudek and P. Pełka , "Forecasting Cryptocurrency Prices Using Contextual ES-adRNN with Exogenous Variables," in *2023*, Computational Science – ICCS.

[65] A. Bulatov, "Forecasting Bitcoin Prices Using N-BEATS Deep Learning Architecture," 2020.

[66] C.-H. Wu, C.-C. Lu, Y.-F. Ma and Ruei-Sha, “A New Forecasting Framework for Bitcoin Price with LSTM,” in *2018*, IEEE International Conference on Data Mining Workshops.

[67] A. Sbrana and P. A. Lima de Castro , “N-BEATS Perceiver: A Novel Approach for Robust Cryptocurrency Portfolio Forecasting,” *Computational Economics,* vol. 64, p. 1047–1081, 2024.

[68] R. S. Tsay, Analysis of Financial Time Series, 2nd Edition, Wiley, 2005.

[69] [Online]. Available: https://nixtlaverse.nixtla.io/utilsforecast/losses.html#multi-quantile-loss. [Accessed 26 09 2024].

[70] A. Géron, Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition, O'Reilly Media, Inc., 2019.

[71] S. Bethard, “We need to talk about random seeds,” in *ACL ARR 2021*, 2021.

Appendix A

In this appendix, the detailed specifications of each forecasting model used in the study are presented. This includes an overview of the hyperparameters for each model, the optimization process undertaken to enhance their performance, and a comprehensive summary of the results obtained for both the price modelling and return forecasting approaches. Each section aims to provide clarity on the methodologies employed and the outcomes achieved, facilitating a deeper understanding of the models' performance and effectiveness in predicting Bitcoin prices.

MLP

Implemented using class MLP[9] from NeuralForecast which uses ReLU as the activation function and trains with ADAM stochastic gradient.

Table 11: MLP hyperparameters

Parameter	Search Range	Price Model	Return Model
Scaler Type	Identity; Standard; MinMax	Identity	Identity
Input Size	2; 7; 15; 30	2	2
Learning Rate	0,000001 to 0,01	0.002811	0.000414
Batch Size	64; 128; 256; 512; 1024	512	128
Hidden Size	2; 4; 8; 16; 32	32	2

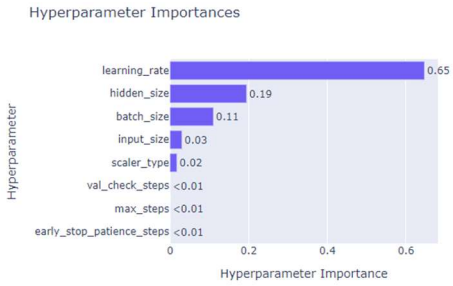

Figure 36: MLP price model hyperparameter importance

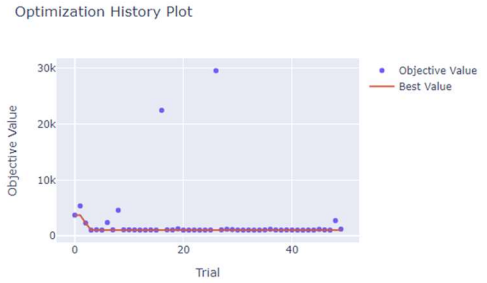

Figure 37: MLP price model optimization history

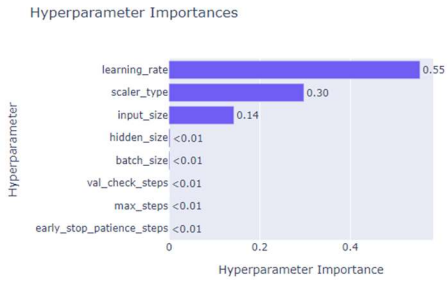

Figure 38: MLP return model hyperparameter importance

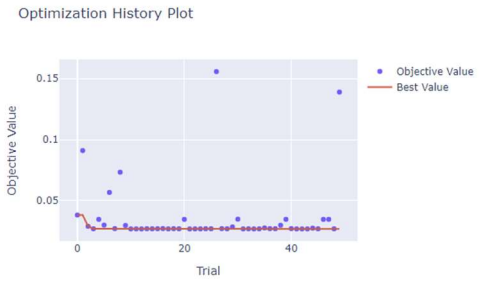

Figure 39: MLP return model optimization history

Table 12: MLP performance metrics

MODEL	MAE	MSE	RMSE	MAPE	R^2
Price	728.15	1.32 e+06	1,149.98	1.758	0.99508
Return	**716.03**	**1.31 e+06**	**1,146.04**	**1.719**	**0.99512**

[9] https://nixtlaverse.nixtla.io/neuralforecast/models.mlp.html

LSTM

Implemented using class LSTM[10] from NeuralForecast which implements a multi layer network with LSTMs as encoders and MLPs, with ReLU as the activation function, as decoders. Trained with ADAM stochastic gradient.

Table 13: LSTM hyperparameters

Parameter	Search Range	Price Model	Return Model
Scaler Type	Identity; Standard; MinMax	MinMax	Standard
Input Size	2; 7; 15; 30	15	7
Learning Rate	0.000001 to 0,01	0.00070668	0.00000895
Batch Size	64; 128; 256; 512; 1024	128	128
Encoder Layers	1 to 4	1	3
Encoder Dropout[11]	0 to 0.2	0.0211	0.0637
Encoder Hidden Size	2; 4; 8; 16; 32	16	8
Decoder Layers	1 to 4	3	3
Decoder Hidden Size	2; 4; 8; 16; 32	32	32
Context Size	2; 4; 8; 10; 12	12	12

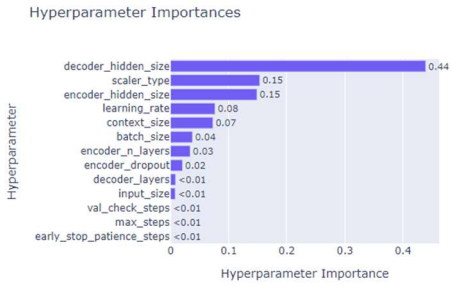

Figure 40: LSTM price model hyperparameter importance

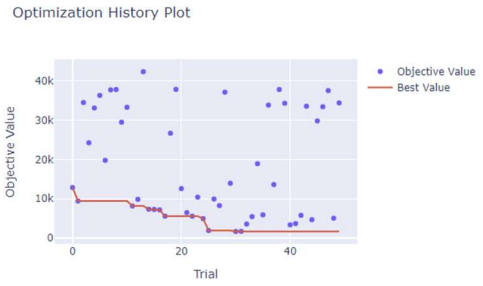

Figure 41: LSTM price model optimization history

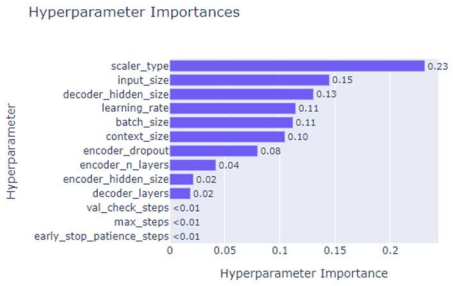

Figure 42: LSTM return model hyperparameter importance

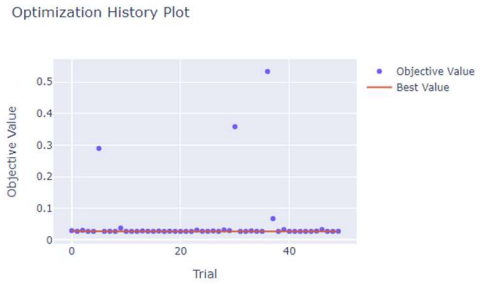

Figure 43: LSTM return model optimization history

Table 14: LSTM performance metrics

MODEL	MAE	MSE	RMSE	MAPE	R^2
Price	1,329.56	3.67 e+06	1,868.20	3.190	0.98638
Return	**724.10**	**1.33 e+06**	**1,152.84**	**1.738**	**0.99506**

[10] https://nixtlaverse.nixtla.io/neuralforecast/models.lstm.html
[11] If the number of layers is only 1 then the dropout is not applied

GRU

Implemented using class GRU[12] from NeuralForecast which implements a multi layer network with GRUs as encoders and MLPs, with ReLU as the activation function, as decoders. Trained with ADAM stochastic gradient.

Table 15: GRU hyperparameters

Parameter	Search Range	Price Model	Return Model
Scaler Type	Identity; Standard; MinMax	MinMax	Standard
Input Size	2; 7; 15; 30	2	30
Learning Rate	0.000001 to 0,01	0.004499907	0.000390041
Batch Size	64; 128; 256; 512; 1024	256	1024
Encoder Layers	1 to 4	1	2
Encoder Dropout[13]	0 to 0.2	0.1068	0.0095
Encoder Hidden Size	2; 4; 8; 16; 32	8	4
Decoder Layers	1 to 4	2	3
Decoder Hidden Size	2; 4; 8; 16; 32	32	16
Context Size	2; 4; 8; 10; 12	8	10

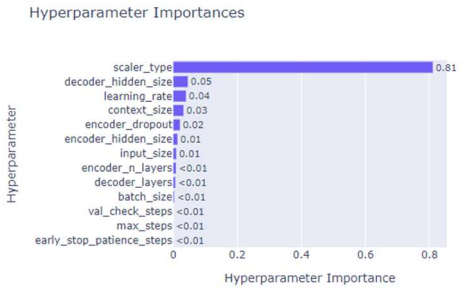

Figure 44: GRU price model hyperparameter importance

Figure 45: GRU price model optimization history

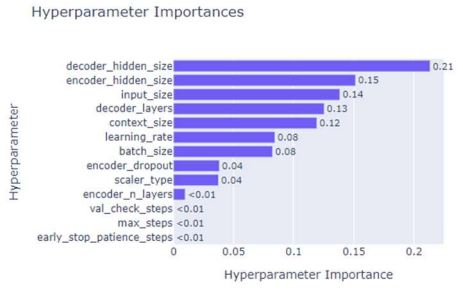

Figure 46: GRU return model hyperparameter importance

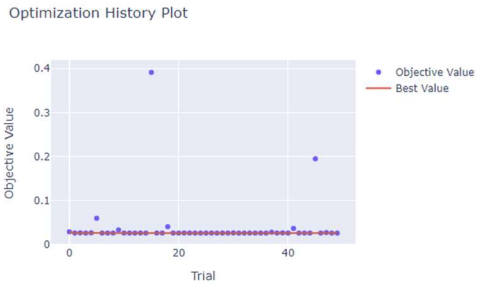

Figure 47: GRU return model optimization history

Table 16: GRU performance metrics

MODEL	MAE	MSE	RMSE	MAPE	R^2
Price	2,243.15	9.97 e+06	2,831.00	5.506	0.96297
Return	**715.16**	**1.31 e+06**	**1,146.56**	**1.714**	**0.99511**

[12] https://nixtlaverse.nixtla.io/neuralforecast/models.gru.html
[13] If the number of layers is only 1 then the dropout is not applied

DeepAR

Implemented using class DeepAR[14] from NeuralForecast.

Table 17: DeepAR hyperparameters

Parameter	Search Range	Price Model	Return Model
Scaler Type	Identity; Standard; MinMax	MinMax	Standard
Input Size	2; 7; 15; 30	30	30
Learning Rate	0.000001 to 0,01	0.003660815	0.000200222
Batch Size	64; 128; 256; 512; 1024	128	256
LSTM Layers	2; 4; 8; 12	4	12
LSTM Hidden Size	2; 4; 8; 16; 32	2	4

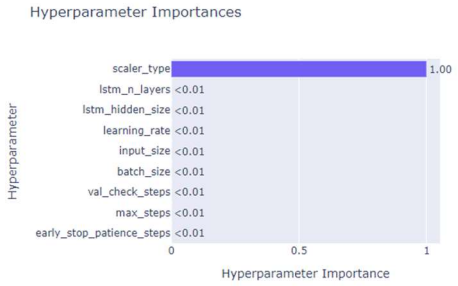

Figure 48: DeepAR price model hyperparameter importance

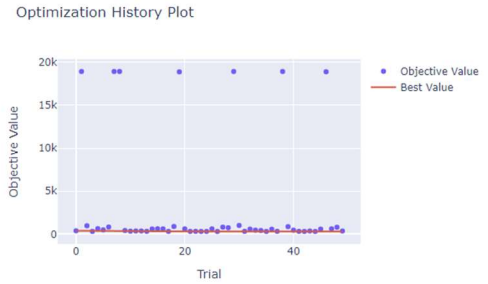

Figure 49: DeepAR price model optimization history

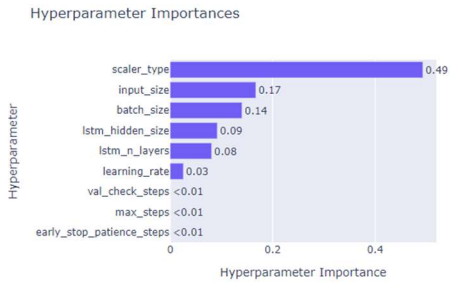

Figure 50: DeepAR return model hyperparameter importance

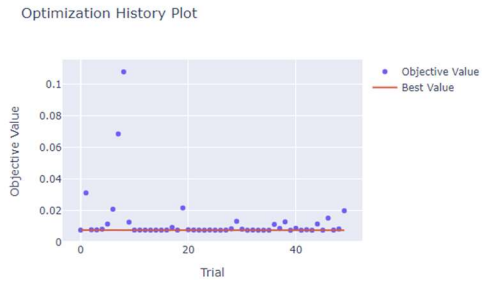

Figure 51: DeepAR return model optimization history

Table 18: DeepAR performance metrics

MODEL	MAE	MSE	RMSE	MAPE	R^2
Price	1,244.26	3.92 e+06	1,785.30	3.087	0.98545
Return	**745.17**	**1.37 e+06**	**1,170.50**	**1.796**	**0.99491**

[14] https://nixtlaverse.nixtla.io/neuralforecast/models.deepar.html

N-BEATS

Implemented using class N-BEATS[15] from NeuralForecast.

Table 19: N-BEATS hyperparameters

Parameter	Search Range	Price Model	Return Model
Scaler Type	Identity; Standard; MinMax	Identity	Identity
Input Size	2; 7; 15; 30	2	2
Learning Rate	0.000001 to 0,01	0.00005719	0.000182030
Batch Size	64; 128; 256; 512; 1024	128	1024

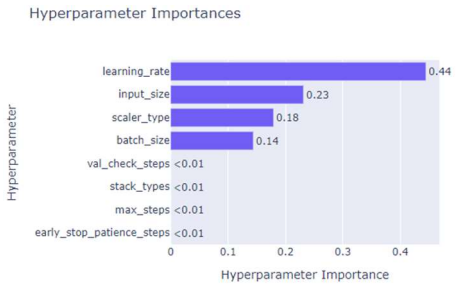

Figure 52: N-BEATS price model hyperparameter importance

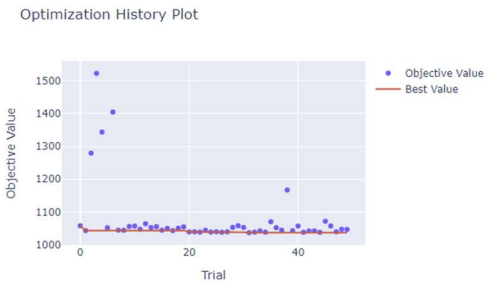

Figure 53: N-BEATS price model optimization history

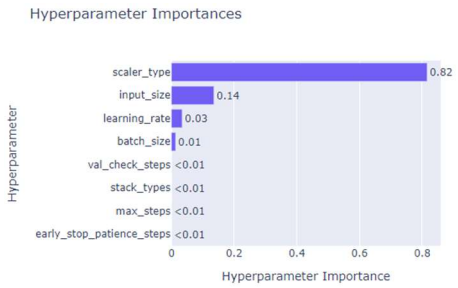

Figure 54: N-BEATS return model hyperparameter importance

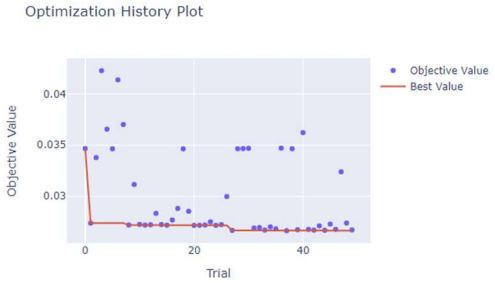

Figure 55: N-BEATS return model optimization history

Table 20: N-BEATS performance metrics

MODEL	MAE	MSE	RMSE	MAPE	R^2
Price	713.98	**1.30 e+06**	**1,140.90**	**1.717**	**0.99517**
Return	**713.51**	1.31 e+06	1,145.68	1.717	0.99512

[15] https://nixtlaverse.nixtla.io/neuralforecast/models.nbeats.html

N-HITS

Implemented using class N-BEATS[16] from NeuralForecast.

Table 21: N-HITS hyperparameters

Parameter	Search Range	Price Model	Return Model
Scaler Type	Identity; Standard; MinMax	Identity	Identity
Input Size	2; 7; 15; 30	2	7
Learning Rate	0.000001 to 0,01	0.000197852	0.000006857
Batch Size	64; 128; 256; 512; 1024	512	1024

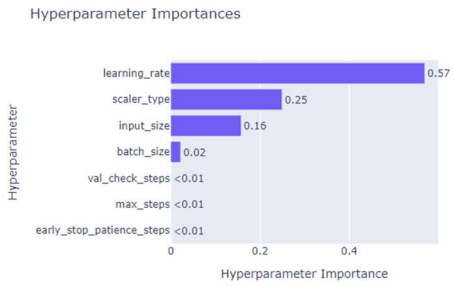

Figure 56: N-HITS price model hyperparameter importance

Figure 57: N-HITS price model optimization history

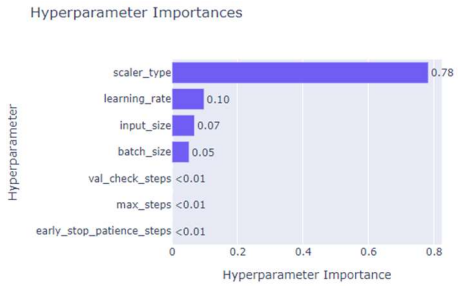

Figure 58: N-HITS return model hyperparameter importance

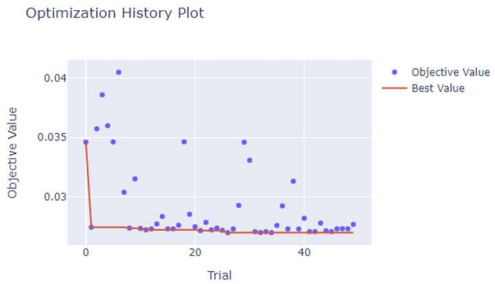

Figure 59: N-HITS return model optimization history

Table 22: N-HITS performance metrics

MODEL	MAE	MSE	RMSE	MAPE	R^2
Price	715.48	1.30 e+06	1,141.56	1.721	0.99516
Return	**713.97**	**1.29 e+06**	**1,134.28**	**1.721**	**0.99522**

[16] https://nixtlaverse.nixtla.io/neuralforecast/models.nbeats.html

PatchTST

Implemented using class N-BEATS[17] from NeuralForecast.

Table 23: PatchTST hyperparameters

Parameter	Search Range	Price Model	Return Model
Scaler Type	Identity; Standard; MinMax	Standard	Standard
Input Size	2; 7; 15; 30	7	30
Learning Rate	0.000001 to 0,01	0.00000305	0.000002047
Batch Size	64; 128; 256; 512; 1024	128	64
Heads	2, 4, 8, 16	16	2
Patch Length	2, 4, 8, 16, 24	8	24

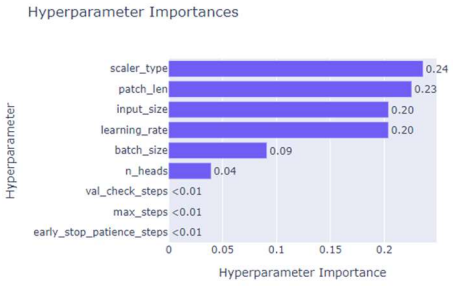

Figure 60: PatchTST price model hyperparameter importance

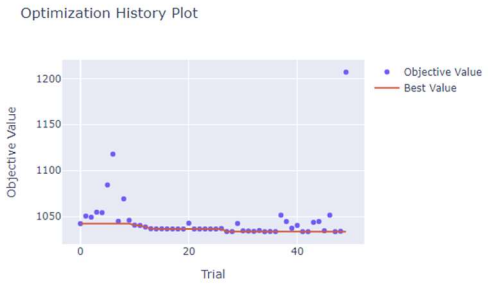

Figure 61: PatchTST price model optimization history

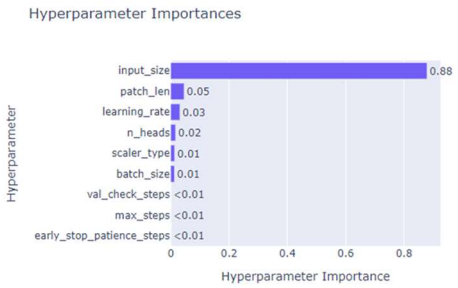

Figure 62: PatchTST return model hyperparameter importance

Figure 63: PatchTST return model optimization history

Table 24: PatchTST performance metrics

MODEL	MAE	MSE	RMSE	MAPE	R^2
Price	**709.91**	**1.30 e+06**	**1,138.37**	**1.702**	**0.99519**
Return	738.71	1.35 e+06	1,160.03	1.785	0.99500

[17] https://nixtlaverse.nixtla.io/neuralforecast/models.nbeats.html

TiDE

Implemented using class N-BEATS[18] from NeuralForecast.

Table 25: TiDE hyperparameters

Parameter	Search Range	Price Model	Return Model
Scaler Type	Identity; Standard; MinMax	Standard	Identity
Input Size	2; 7; 15; 30	7	7
Learning Rate	0.000001 to 0,01	0.00632421	0.003080501
Batch Size	64; 128; 256; 512; 1024	512	1024

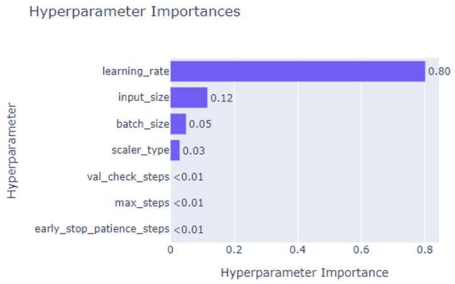

Figure 64: TiDE price model hyperparameter importance

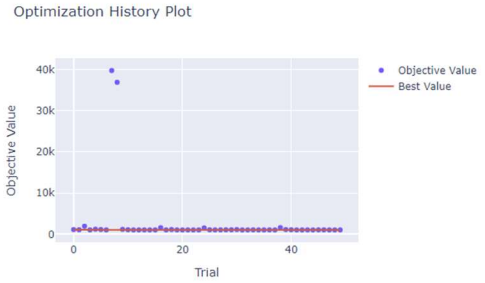

Figure 65: TiDE price model optimization history

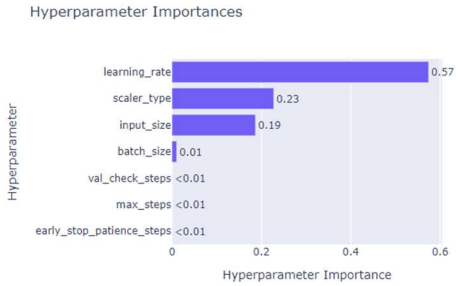

Figure 66: TiDE return model hyperparameter importance

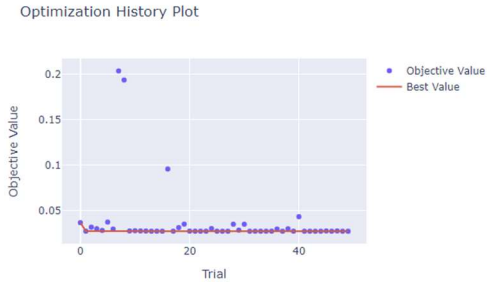

Figure 67: TiDE return model optimization history

Table 26: TiDE performance metrics

MODEL	MAE	MSE	RMSE	MAPE	R^2
Price	**712.80**	**1.29 e+06**	**1,137.73**	**1.711**	**0.99519**
Return	714.55	1.30 e+06	1,141.90	1.717	0.99516

[18] https://nixtlaverse.nixtla.io/neuralforecast/models.nbeats.html