Master Thesis

Cryptocurrency Prices Prediction - A Comparative Study of Statistical, Machine Learning, Deep Learning and Hybrid Approaches.

Author:

Ng Jia Yan

August 25, 2021

Advisors:

Prof. Dr. Jürgen Jacobs
Institute of Information Systems

Prof. Dr. Peter Niemeyer
Institute of Information Systems

Declaration

I hereby declare that I am the sole author of my work and that I have not used any sources or learning materials other than those indicated. I also declare that I have acknowledged the work of others by providing detailed references. I also hereby confirm that my work has not been prepared for any other examination or assignment has been prepared for any other examination or assignment, either in whole or in part.

Lüneburg, August 25, 2021 *Ng Jia Yan*

[illegible]

Abstract

As an emerging alternative form of financial transactions, cryptocurrency has gradually gained popularity as the new alternative investment. Hence cryptocurrency price prediction can facilitate investors for building strategies to trade and also to support financial researcher to understand their behaviour as digital assets in the market. Several attempts have been made to predict the prices of cryptocurrencies, especially Bitcoin, using machine learning and deep learning methods. This master thesis aims to investigate and compare the typical statistical methods of time series analysis with more advanced approaches such as machine learning, deep learning, and hybrid methods for predicting cryptocurrency prices with a prediction horizon of one hour. In most studies, typical evaluation measures such as the mean square error and the mean absolute error are used to evaluate model accuracy in regression problems. This study also examines other evaluation measures that can be used to assess how well a model predicts large movements. The experimental results show that the ARIMA model outperforms the other advanced models in price prediction, while the LSTM model is better at predicting large price movements.

Contents

List of Figures

List of Tables

1. Introduction

Since Bitcoin's introduction in the wake of the economic recession in 2009, this new form of digital currency has gained tremendous traction over the past decade. Its success has led to the emergence of a large number of other cryptocurrencies. As of April 22, 2020, there are approximately 5392 cryptocurrencies traded with a total market capitalization of $201 billion. Nevertheless, the cryptocurrency market is to a great extent dominated by Bitcoin (BTC), Ethereum (ETH) and Ripple (XRP), which together hold 77.4% of the global cryptocurrency market capitalization. (Bagshaw, 2020). Although many governments and central banks of various countries are reluctant to adopt and accept the technology, access to the cryptocurrency market has become easier and easier over time. These major cryptocurrencies can be purchased with fiat currency on various online exchanges and trading platforms such as Coinbase, Binance, and Voyager. Meanwhile, the other less popular cryptocurrencies can in turn be bought by these major cryptocurrencies.

When it comes to identifying the driving factors behind the price of cryptocurrencies, we have barely scratched the surface. They are considered highly volatile and many researchers suggest that Bitcoin is subject to bubble-like behavior, at least for a period of time (Corbet et al., 2018; Chaim and Laurini, 2019; Härdle et al., 2020). Therefore, predicting the prices of cryptocurrencies is considered one of the most difficult problems in finance and this task can be considered analogous to other financial time series forecasting tasks, such as stock and forex forecasting.

Different attempts have been made to predict cryptocurrency prices in the form of time series analysis using different strategies. Typically, time series problems have been tackled with simple classical methods such as Simple Exponential Smoothing (SES), Error Trend and Seasonality (ETS) and Autoregressive Integrated Moving Average (ARIMA) models. Desev et al. (2019) analyze the cryptocurrency markets using ARIMA models. On the other hand, Malladi and Dheeriya (2021) perform a time series analysis of cryptocurrency returns and volatilities using other statistical modeling techniques such as Autoregressive-moving-average model with exogenous inputs model (ARMAX) and Vector Autoregression (VAR) model.

Others also have attempted to tackle the problem using more sophisticated methods such as machine learning and deep learning techniques. Chen et al. (2020) carefully applies machine learning approaches with an appropriate sampling dimension and thoughtful feature evaluation to predict the bitcoin price. Livieris et al. (2021) proposes a deep multi-input neural network model called MICDL to predict the price and movement of cryptocurrencies.

The focus of this paper is to compare the accuracy of different time series forecasting approaches for predicting cryptocurrency prices. In particular, a univariate analysis of cryptocurrencies is conducted using both statistical and neural network-based approaches to predict prices with a forecast horizon of one hour. This empirical study analyzes data of the three most popular cryptocurrencies, Bitcoin, Ehtereum, and Ripple, over five consecutive years.

The thesis is organized as follows: This chapter introduces the topic and motivation for this thesis. Chapter 2 highlights the methods proposed so far for predicting cryptocurrency prices and reviews the relevant literature on this topic. Chapter 3 explains the theoretical background of the selected models used for one-step prediction and discusses the modeling approaches chosen for this research. Chapter 4 presents the empirical results under different conditions and finally chapter 5 draws the conclusions and discusses future work.

1.1 Motivation

The review of the literature reveals that much of the research on cryptocurrencies earlier focuses on Bitcoin, while less attention is paid to other cryptocurrencies. In recent years, there has been more research focused on applying machine learning or deep learning techniques to predict cryptocurrency prices, involving Ethereum, Ripple, or other cryptocurrencies in their experiments. However, none of these studies use simple statistical methods or naive forecast as a benchmark. Therefore, it is very difficult to know how efficient or accurate these advanced models really are.

In the past, a number of open forecasting competitions, known as M-competitions, have empirically demonstrated that most well-established statistical methods outperform more sophisticated machine learning methods in time series forecasting (Makridakis et al., 2018, 2020). Since the reported results may be related to the specific data set used in their experiments, it is worth conducting my own experiment with cryptocurrency data to confirm this claim. Moreover, as Pintelas et al. (2020) noted, an alternative method for evaluating currency forecasting models is needed because the usual error metrics used for regression problems are limited in evaluating the directional accuracy of the model. Therefore, this study also examines various evaluation metrics that can be used to evaluate the performance of cryptocurrency price prediction models to draw a more comprehensive conclusion.

Similar to the work of Ji et al. (2019), this thesis investigates and compares the model performance of different neural network-based methods under different conditions, but with the inclusion of a classical statistical method. The focus of this study is also not only on the models of deep learning approach, but also some other neural networks from different classes of techniques with different levels of complexity are of interest. The main objective of this study is thus not to develop a novel method for predicting cryptocurrency prices or to investigate suitable predictive variables for the problem. Instead, the main objective is to demonstrate the performance of different neural network-based methods compared to a classical time series prediction method for a prediction problem where cryptocurrency prices are used as univariate time series.

1.2 Research Question

"How accurately can advanced neural network-based models predict cryptocurrency prices compared to a statistical approach?"

2. Literature Review

The ARIMA models have been widely used for time series forecasting, especially in finance. Previously, they are frequently used in studies on stock predictions (Ariyo et al., 2014; Adebiyi et al., 2014). However, there aren't just as many prominent research papers that focus on using ARIMA models for forecasting cryptocurrency prices. This could be due to the fact that more advanced techniques such as machine learning and deep methods have overshadowed the use of statistical approaches in recent years.

With a slightly different focus, the work of Desev et al. (2019) analyze the efficiency of cryptocurrency markets with different short-term forecast horizons using ARIMA models. Specifically, some simple investment strategies are used to compare the returns achieved by the model's forecasts. The authors suggest that variation in model forecast accuracy should also be considered when determining the efficiency of trading strategies.

Machine learning-based forecasting approaches are widely used to predict cryptocurrency prices. For instance, Akyildirim et al. (2021) examines the predictability of 12 cryptocurrencies over four different time scales, at both daily and various minute levels, using machine learning classification algorithms, including logistic regression, random forests, support vector machine, and artificial neural networks with information about past prices and technical indicators as model predictors. The results show that the support vector machine is the most robust model, providing good generalization ability at different time levels.

In addition to machine learning techniques, more advanced deep learning approaches have also gained considerable attention. Livieris et al. (2021) propose a CNN-LSTM model that uses different cryptocurrency data (e.g., Bitcoin, Etherium, and Ripple) The data are preprocessed independently and then fed into the model as multiple inputs to estimate the final prediction. The proposed model is evaluated against two other CNN-LSTM models with slightly different configuration without using combination of data as input. Their experimental analysis shows that the proposed model reduces both computational costs and the risk of overfitting. The authors argue that using data from multiple cryptocurrencies can improve prediction accuracy.

Aside from that, Pintelas et al. (2020) evaluates deep learning algorithms such as LSTM, bidirectional long-term memory (BiLSTM), and convolutional neural network (CNN) for predicting prices of BTC, ETH and XRP. The best deep learning topologies used in this paper are found through extensive experimentation. In addition, the performance of the model is also compared with machine learning models such as the support vector regressor, the 3-nearest neighbor regressor, and the decision tree regressor and the authors conclude that the deep learning models are inferior for predicting the prices. The problem of classical regression evaluation metrics such as mean absolute error (MAE) and root mean square error (RMSE) in evaluating the best model for predicting cryptocurrency prices is also highlighted, as the authors argue that a predictive model could have low values on these metrics but fail to predict the price direction correctly.

Recently, the combination of different strategies has also been explored, either through an ensemble approach or through hybrid modeling. For instance, Dixon and London (2020) propose a general class of an exponential smoothed recurrent neural networks (RNNs) to tackle challenges in forecasting non-stationary series. The author argues that by incorporating the classical statistical method of exponential smoothing in time series analysis into the conventional RNN architecture, the design of the proposed model is more suitable for forecasting time series with numerical data. The paper also investigates the predictability of Bitcoin prices at the minute level using the proposed architecture and two other popular deep learning methods such as the LSTM and gate recurrent units (GRU) models. The results show that the GRU is overfitted and the model performance of LSTM and the proposed α-RNN is comparable. Since the results show that there is little difference in model performance between the GRU and α-RNN models, the authors argue that the additional cellular gates in the LSTM provide no advantage, so the proposed model with fewer parameters to train is preferred.

In the field of comparative study, Ji et al. (2019) compares the prediction performance of different deep learning methods such as deep neural network, long short term memory model, convolution neural network and a combination of convolution neural network and recurrent neural network model called ResNet for predicting Bitcoin price. The prediction problem is defined as a regression problem and also a classification problem, where the first objective is to predict the bitcoin price, and the second is to predict whether the price will rise or fall. Their study uses 18 blockchain features of Bitcoin as predictors for each model, and their results are inconclusive as no model significantly outperforms another.

3. Methodology

3.1 Forecasting Methods

Numerous models from each technique class are available in the existing literature. Autoregressive integrated moving average (ARIMA), multilayer perceptron (MLP) and long short-term (LSTM) models are selected for this empirical study as the representative method of its respective technique class since other researches have shown that these models provide the most accurate result for time series forecasting in other domains Makridakis et al. (2018); Pintelas et al. (2020). However, given the myriads of variations for each considered model proposed in the existing literature, it would be a laborious task to include all possible varieties. Therefore, my strategy is to only consider the basic architecture of each model. On the other hand, although many hybrid models have been proposed for time series forecasting in the recent forecasting competition, the M4 competition, the application of hybrid models in financial time series forecasting is relatively limited. An exponential smoothed recurrent neural networks (α-RNNs) proposed by Dixon and London (2020) recently shows a promising result in forecasting Bitcoin prices, it is thus included in this study as the hybrid approach. The following subsection provides the theoretical description of each model and a brief explanation of the forecasting procedure used in each case.

3.1.1 Autoregressive integrated moving average (ARIMA)

An autoregressive integrated moving average is the general class of statistical models for forecasting a time series. Given time series data X_t, they can be modelled as a linear combination of a constant, its lagged values plus the current and past forecast errors:

$$X_t = \delta + \phi_1 x_{dt-1} + \phi_p x_{dt-p} + \ldots + \theta_1 \epsilon_{t-1} + \theta_q \epsilon_{t-q} + \epsilon_t$$

The three key aspects encapsulated in the acronym of the model is explained with their respective order terms, p,d and q as following:

p: The order that captures the autoregressive (AR) part of the model. It is defined as the number of lagged values (previous values in time) for the forecast variable to be considered in the model.

d: The order that captures the integrated (I) nature of the model. It represents the number of times a non-stationary time series should be differenced in order to achieve stationarity.

q: The order that corresponds to the moving average (MA) part of the model. It is defined as the number of lagged values for the noise term to be considered in the model.

The ARIMA process can then be generalized as following:

$$\left(1-\sum_{i=1}^{p}\phi_i B^i\right)(1-B)^d X_t = \delta + \left(1+\sum_{i=1}^{q}\theta_i B^i\right)\varepsilon_t \tag{3.1}$$

where B is the backshift operator, δ is the constant and ϵ as the error term.

Estimation of the parameters in equation 3.1 relies on the values of p, d and q which will be determined using the following steps which are adopted from the work of Desev et al. (2019):

1. Estimating the order of differencing, d based on augmented Dickey–Fuller (ADF) stationary test.
2. Setting the boundary values for autoregressive (AR) and moving average (MA) order that are based on autocorrelation and partial autocorrelation decay.
3. Performing model selection based on Akaike Information Criteria (AIC) with the range of p, d and q found in preceding steps.
4. Performing one step ahead forecast over the period of test set and computing error metrics described in section 3.5 with the best performing ARIMA model for each time series to access the performance of the respective selected model.

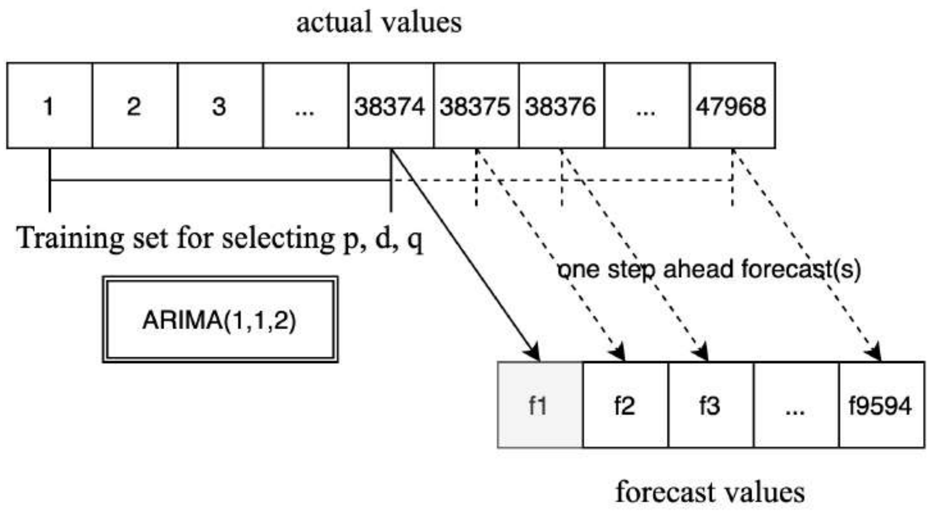

Figure 3.1: An overview of the forecasting procedure of ARIMA for BTC series

After a selection of the best combination of ARIMA order is done using the training dataset, the best fitting model for each series will be used to perform one step ahead forecast iteratively. Each subsequent forecast value is obtained sequentially using the originally fitted model and the next actual observation. The ARIMA modeling describe is done using a Python library called “statsmodels”. All the forecast values are then measure against the actual values using evaluation metrics described in section 3.5. An illustration of how the forecasting procedure works is demonstrated in Figure 3.1.

3.1.2 Multi-Layer Perceptron(MLP)

Multi-layer Perceptron (MLP) is the simplest form of feedforward artificial neural network (ANN). It comprises three key components: an input layer, a hidden layer/several hidden layers and an output layer. The input layer represents a vector of input features, $\mathbf{x}$ and all other layers are made up of nodes that use activation function, h. For time series forecasting, observations from preceding w period $\mathbf{x} = [x_{t-1}, x_{t-2}, \cdots, x_{t-w}]$ are fed to the hidden layer that contains n number of neuron. Each neuron applies a linear transformation and activation function to $\mathbf{x}$ to compute the output of $g_i = h(\mathbf{w_i}\mathbf{x} + b_i)$ where $\mathbf{w_i}$ and b_i are the weights and bias of the linear transformation. These are the model parameters of MLP whereas h is a activation function which is a hyperparameter of the model. The goal of the model is to predict the closing prices which are continuos variables therefore this is a regression problem. For a regression problem, a linear activation function node should be used for the output layer. To implement the basic topology, a single hidden layer neural network is constructed as illustrated in Figure 3.2. The number of input nodes will be determined by how the time series is split by a fixed size window, w as discussed in section 3.2.1. In this study, the MLP models are constructed with $w = 5$, 10 and 30 only, whose performance is then compared to find the best number of input nodes. Adam optimizer is used for training the model. The optimal number of hidden nodes, the type of activation function in the hidden layer and other hyperparameters needed for the training process are searched by using Bayesian Optimization method as describe in section 3.4.3.2. Furthermore, in order to facilitate the convergence of backpropagation algorithm, the data is scaled between 0 and 1 as elaborated in section 3.3.1.3. All the predicted values are then rescaled back to the original scale once all predictions are made. The differences between all forecasts and actual prices are evaluated in original scale using error metrics explained in section 3.5.

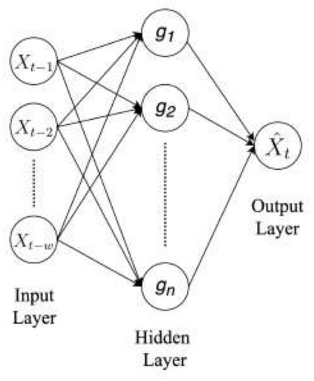

Figure 3.2: A single hidden layer neural network

3.1.3 Long Short-Term Memory(LSTM) Model

A typical feedforward neural network like MLP usually has limitation in predicting time series data because it does not implicitly consider the temporal relationship between the input sequence. Simply put, MLP model has no notion of order in time, it has no memory of the inputs it received and consider them as independent variables, which have no sequential characteristic.

In contrast, Long Short Term Memory (LSTM) model is a special type of recurrent neural network (RNN) capable of "memorizing" long term dependencies. This allows it to better understand the temporal relationship of the time series data. All RNNs consist of a chain of modules of neural network that repeat recursively. A LSTM model also has a similar chain like structure but the structure of the repeating modules are different. The repeating modules of standard RNNs usually consist a single tanh layer but while the LSTM network consists of four neural network layers that interact with each other in specific ways.

The number of repeating modules needed depends on how we split the input-output pairs for the training. In this study, only 5, 10 and 30 LSTM units are considered to construct the model as described in section 3.2.1. Assuming that three days data are used for the price forecast, the following diagram illustrates the operation of the internal mechanism within a LSTM module:

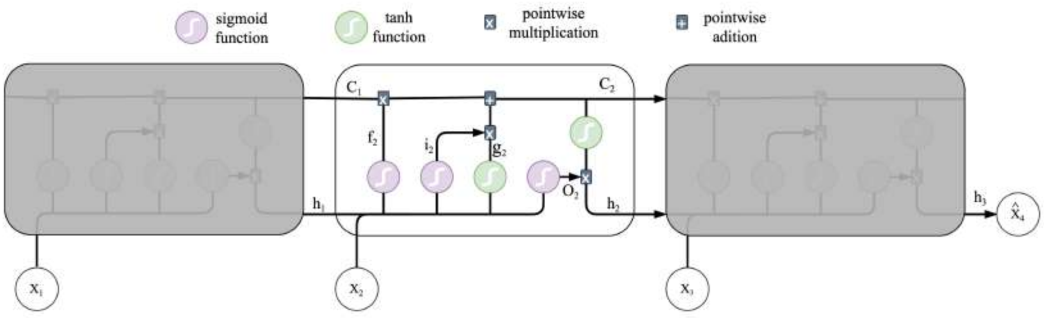

Figure 3.3: A long short term memory (LSTM) model

The key to the internal mechanism of LSTM's repeated module is a structure called the cell state. Thanks to the cell state, LSTM model is capable to remove or retain information from the previous time step. This operation is carefully controlled by other mechanisms called gates. There are three type of gates in LSTM, namely, f_t, i_t and o_t corresponding to the forget, input and output gates respectively.

Forget gate, f_t regulates what information to be removed from the cell state. On the contrary, the input gate, i_t and a tanh layer that creates a vector of new values, g_t controls the fraction of new information to be added to the cell state, c_t. The new cell state is then updated by multiplying the old cell state to the forget gate, which decides to what extent the old information should be removed, and then adding the new candidate value, scaled by the amount of new information to be updated.

Finally, the cell state value passes through a tanh layer and is multiplied by the output gate, which decides how much information from the cell state is output from the module. Through this internal mechanism, LSTM can transfer memory over long periods of time. All operations within each repeating module are formally defined as follows,

$$\begin{aligned}
f_t &= \sigma\left(W_f x_t + U_f h_{t-1} + b_f\right) \\
i_t &= \sigma\left(W_i x_t + U_i h_{t-1} + b_i\right) \\
g_t &= \tanh\left(W_g x_t + U_g h_{t-1} + b_g\right) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
o_t &= \sigma\left(W_o x_t + U_0 h_{t-1} + b_o\right) \\
h_t &= o_t \odot \tanh\left(c_t\right)
\end{aligned}$$

where σ is a sigmoid function and W, U and b are model parameters to be learned from the training process.

Due to high computational time, the basic architecture of LSTM model is implemented using a single layer of LSTM network. The number of input nodes depend on the window size, w used to prepare the input sequences that will be fed to the model. Similar to the MLP model, a single linear node in the output layer is used to build the model, and the model is trained with an Adam optimizer. Section 3.4.3 further explains how the rest of the hyperparameters are tuned in detail. The model is implemented using TensorFlow framework.

3.1.4 Exponential Smoothed Recurrent Neural Network(α-RNN)

Dixon and London (2020) propose a general class of exponentially smoothed recurrent neural networks (α-RNNs), which can be viewed as a hybrid method that integrates the exponential smoothing approach in classical time series modeling and the standard recurrent neural network architecture in deep Learning. Their paper discusses the conditions under which a simple RNN can be made to generalize a nonlinear autoregressive model. The core idea of the proposed architecture is to modify the simple RNN model to include an additional smoothing parameter $\alpha \in [0, 1]$ to produce a smoothed version of the hidden state, which in turn is linearly transformed with model parameters to predict the output. This exponential smoothing operation with the hidden state in a simple RNN structure allows the network to have a ”long memory” without having additional cellular states like LSTM or GRU.

The model architecture implemented in this study is the dynamic version of the proposed α-RNN model, where the smoothing parameter is updated dynamically using a recurrent layer. This model is implemented with the source code provided by the authors on Github. This model has similar hyperparameters as the MLP and LSTM models, namely the hidden units, batch size and the L1 regulizer. These hyperparameters are also tuned using the same approach as the previous two models, as described in Section 3.4.3.2.

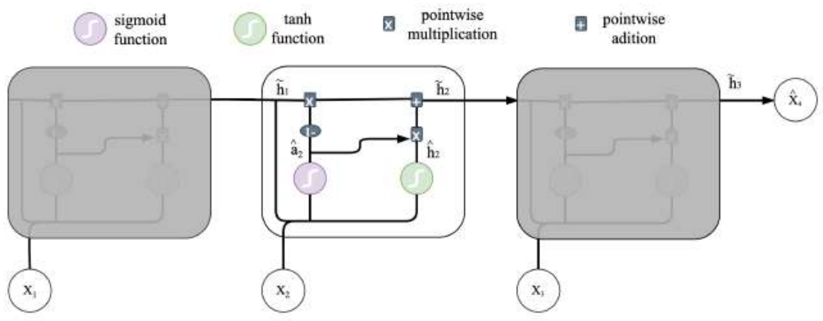

Figure 3.4: Exponential Smoothed Recurrent Neural Network (α-RNN) model

The diagram above illustrates the operation of the internal mechanism within a α-RNN module that contains a single hidden unit with three days data used as an input to forecast the price at next time step. In addition, the structure of the model is defined by the following formula.

$$\tilde{h}_t = \widehat{\alpha}_t \odot \widehat{h}_t + (1 - \widehat{\alpha}_t) \odot \tilde{h}_{t-1}$$

$$\widehat{h}_t = \tanh\left(U_h \tilde{h}_{t-1} + W_h x_t + b_h\right)$$

$$\widehat{\alpha}_t = \sigma\left(U_h \tilde{h}_{t-1} + W_\alpha x_t + b_\alpha\right)$$

$$\widehat{y}_{t+1} = W_y \widehat{h}_t + b_y$$

where σ is a sigmoid function and $W_y, b_y, W_h, U_h, b_h, W_\alpha, U_\alpha$ and b_α are model parameters to be learned from the training process; $\widehat{h}_t$ is the hidden state, $\tilde{h}_t$ is the smoothed hidden state, $\widehat{\alpha}_2$ is the smoothing parameter at t time step

3.2 Data

The univariate time series used in this thesis refer to the hourly closing prices of BTC, ETH and XRP that are collected from cryptocompare.com. The hourly closing prices collected cover a period of five years and 5 months, starting from 2015-08-07 13:00:00 to 2021-01-26 04:00:00 with a total of 47,968 observations. Figure 3.5 shows the closing prices of each time series over the period of time analyzed. Since the price disparity in US dollars between these three cryptocurrencies is huge, each series is plotted in its respective price scale to better identify the patterns and trends that exist between them. It is evident that XRP does not have large variability as compared to BTC and ETH. However, the three cryptocurrencies generally still share similar price movement so it will be interesting to investigate how well the same experimental settings applied can generalize across the three time series.

Each time series contain 7 missing data points and the timing of the missing data is not informative. It happened most likely due to a technical glitch during data transmission from the website's API endpoint, therefore the missing values are not part of the data pattern and an assumption is made to handle these missing values. Although cryptocurrency prices exhibit high fluctuation over the observed period, it is still safe to assume the hourly closing prices remain constant for a few hours occasionally. The value from the previous one time step of the missing observations is then propagated forward to fill the missing value.

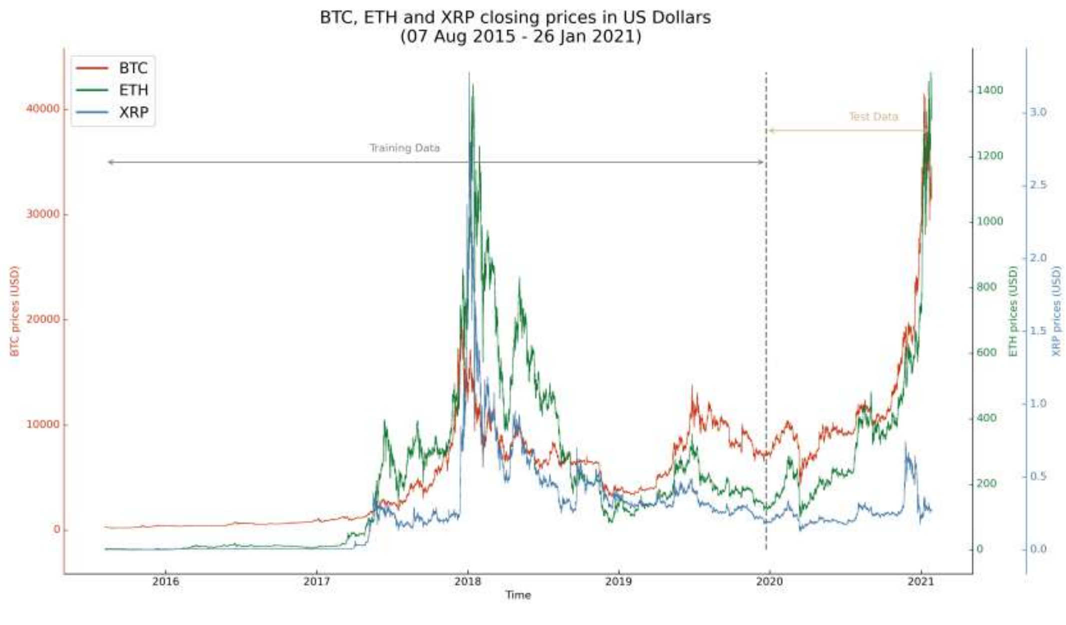

Figure 3.5: Hourly closing prices in US dollars for BTC, ETH, XRP from Aug 2015 to Jan 2021

3.2.1 Data Preparation

As illustrated in figure 3.10, the first 80% of the data are used for training all the models of interest in this study and the rest are used as test data to evaluate their model performance. Table 3.1 illustrates the descriptive statistics of the training dataset and testing dataset including minimum, maximum, mean, standard deviation, skewness and kurtosis for each cryptocurrency. All the datasets are clearly not normally distributed. Different preprocessing methods are then applied to the raw data in order to adjust the statistical properties of the data so that the assumptions of each model used can be satisfied. The summary of the experimental setup as shown in Table 3.3 provides an overview of the methods applied to each model.

Data	**Min**	**Max**	**Mean**	**Std.Dev.**	**Skewness**	**Kurtosis**
BTC(Train Set)	202.940	19784.900	4427.350	3954.510	0.733	-0.184
BTC(Test Set)	4240.690	41490.800	12497.900	7156.310	2.183	4.299
ETH(Train Set)	0.150	1420.770	203.260	241.802	1.783	3.440
ETH(Test Set)	99.230	1457.880	355.958	253.713	2.18903	5.049
XRP(Train Set)	0.001	3.28	0.265	0.335	3.00577	14.693
XRP(Test Set)	0.124	0.744	0.258	0.100	2.430	5.796

Table 3.1: Descriptive Statistics for training set and test set of BTC, ETH and XRP

On top of data transformation, special attention is needed to structure the data in a way so that the neural network models can be trained. To perform one-step ahead forecast with LSTM, MLP and α-RNN models, the series of BTC, ETH and XRP prices are framed as a supervised learning problem. Each series of prices need to be split at a fixed time intervals sequentially using a window size to generate overlapping sequences that will be used to train the neural network models. Since the window size, w is often set using heuristics, window sizes of 5, 10 and 30 are implemented for neural network models so that its effect on model performance can be analyzed.

Given a training dataset of x_t for $t \in \{1, 2, ...T\}$, with each window size, w, a total of $T - w + 1$ sample of input-target pairs, where each output is associated with its previous first w lags will be generated. Using a window size of 5, Table 3.2 illustrate this autoregressive sequence pattern of the input-target pairs used to train the neural network models.

n	**Input**	**Target**
1	$<x_1, x_2, x_3, x_4, x_5>$	x_6
2	$<x_2, x_3, x_4, x_5, x_6>$	x_7
$\vdots$	$\vdots$	$\vdots$
$T - w + 1$	$<x_{T-5}, x_{T-4}, x_{T-3}, x_{T-2}, x_{T-1}>$	x_T

Table 3.2: Input-output pairs with a window size, w=5

3.3 Experimental Setup

In addition to the conventional preprocessing methods that are commonly used in most research papers, fractional differentiation is also applied to ARIMA, MLP, LSTM and α-RNN so that the effectiveness of this preprocessing methods can be analyzed and compared across each model. Fractional differentiation is only implemented to data with window size of 5 for neural network based models due to time constraint. Hence, a comparison will only be done with standard preprocessed models that have the same window size of 5.

Different model selection approaches are also used for ARIMA and neural network-based models to select the respective optimal model. The best model from each technique class will be compared using performance metrics described in section 3.5. The following table summarizes the preprocessing methods and model selection procedures used for each models of interest. The subsequent subsections further elaborate on each of the topics in details.

Model	Standard Preprocessing	Fractional Differentiation	Model Selection
ARIMA	Log Transformation & First Order Differencing: BTC, ETH Original Series: XRP	BTC, ETH	Grid Search for p, d, q using AIC
LSTM	Standardization: BTC, ETH, XRP (window size: 5, 10, 30)	BTC, ETH (window size: 5)	Bayesian Optimization Hyperparameter Search for hidden units, batch size, dropout rate, L1 regularizer using average RMSE of 10 folds cross validation
MLP			
α-RNN			

Table 3.3: Summary of experimental setup for ARIMA, LSTM, MLP and α-RNN models

3.3.1 Standard Preprocessing

This section describes the standard preprocessing applied in this study for ARIMA and the other three neural network based models. These methods are the conventional preprocessing methods that are widely used in most studies when modelling using the respective technique classes. For any statistical forecasting method like ARIMA, an exploratory analysis on seasonality and stationarity before any data transformation is necessary. On the other hand, AI-based forecasting methods like MLP. LSTM and α-RNN require standardized input that facilitate the convergence of the technique used for optimization during training. As a result, the so-called ”standard” preprocessing methods applied to ARIMA and other three neural network-based models in this study are different, as shown in Table 3.3.

3.3.1.1 Check for Seasonality

There is no obvious seasonality can be observed in Figure 3.5, therefore a further analysis on seasonality is done by grouping the closing prices based on week of day and month. These daily and monthly seasonal plots are then inspected visually to determine if an obvious seasonality exists. The former inspection is uninformative as the value appears constant across each day of the week. On the other hand, although there is obvious fluctuations in the monthly plot but no clear and consistent trend is observed as shown in Figure 3.6. A statistical test on the existence of seasonal unit root, the Osborn-Chui-Smith-Birchenhall (OCSB) test as presented in Osborn et al. (1988) is used to verify the presence of the seasonality. The null hypothesis that a seasonal unit root exists cannot be rejected for all three time series, so the data does not require seasonal differencing. As a result a non-seasonal version of ARIMA model will be used for forecasting the prices.

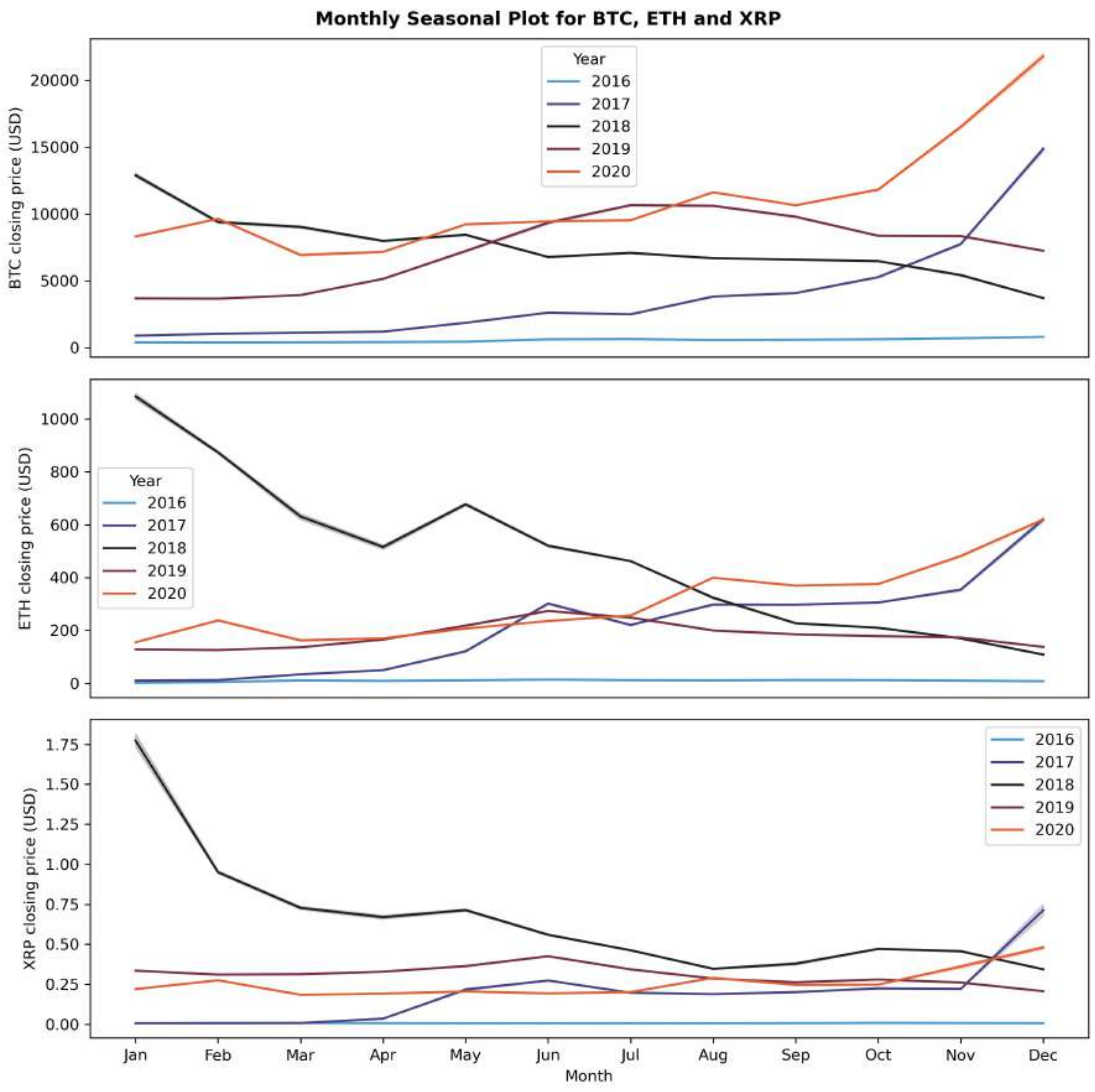

Figure 3.6: Monthly Seasonal Plot of Closing Prices for BTC, ETH and XRP

3.3.1.2 Log Transformation & Differencing

In order to forecast with ARIMA model, the time series must be made stationary. As observed from Table 3.1, the data distribution exhibits a great degree of variability. To help stabilize the variance of the time series, log transformation is then applied to the raw data. A general upward trend is observed after log transformation as shown in the left column of Figure 3.7. First order differencing is then used to reduce the trend of the log prices in order to stablize their mean. The resulting series is essentially equivalent to the log return of the cryptocurrency series (log x_t - log x_{t-1} = $\log(x_t/x_{t-1})$) as plotted on the right column of Figure 3.7. A statistical test for unit root, namely, the augmented Dickey-Fuller (ADF) test is then used to determine stationarity statistically. Stationarity is attained with first order differencing applied to the log transformed series since the null hypothesis of ADF test that a unit root is present can be rejected at 95% confidence level as presented in Table 3.4.

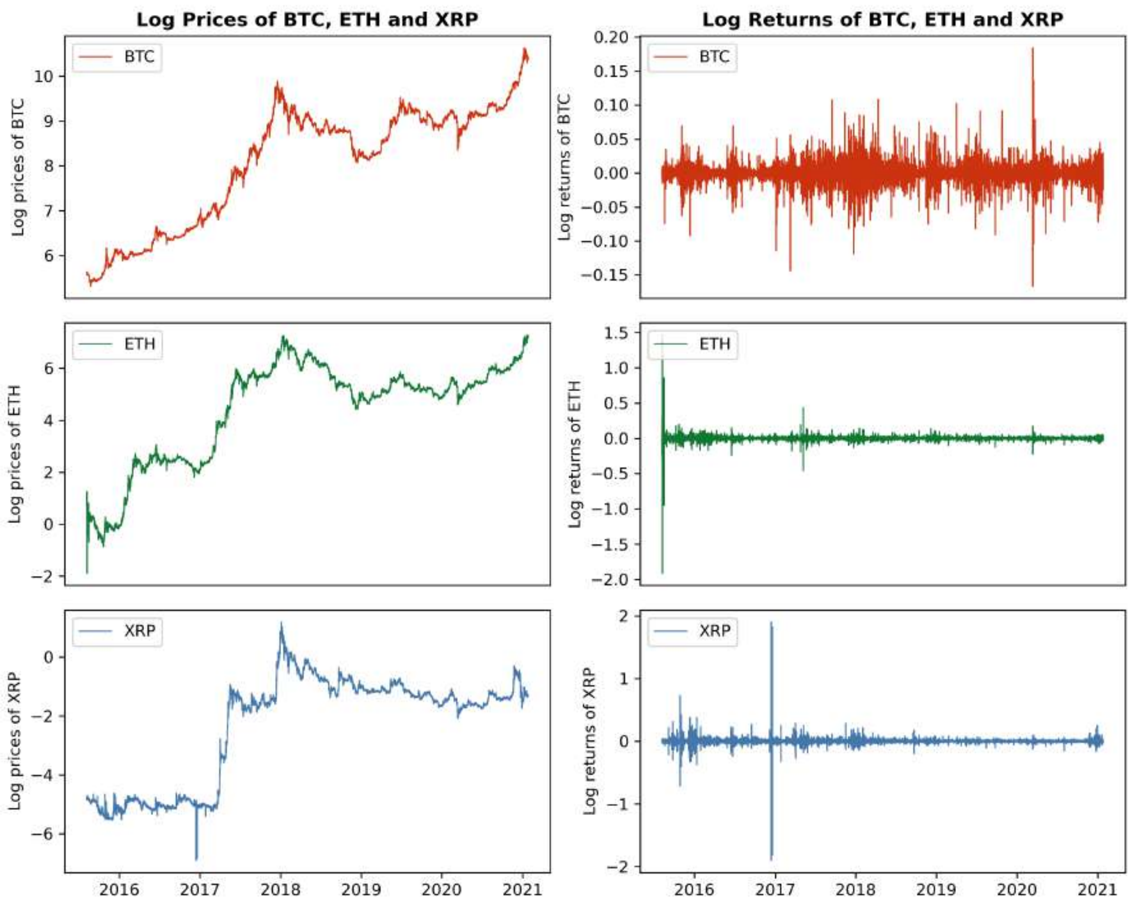

Figure 3.7: All Cryptocurrency Time Series after Log Transformation and First Order Differencing

3.3.1.3 Standardization

According to Chollet (2017), it is risky to feed relatively large inputs to a neural network because doing so could possibly lead to large gradient updates which will in turn prevent the network from converging. The raw data is therefore readjusted so that its distribution has a mean of 0 and a standard deviation of 1. It is also crucial to avoid data leakage in order to achieve better generalization, the training dataset must then be rescaled without any knowledge of the test dataset. Therefore, the training data is standardized using the mean of the deviation of the training data itself instead of the whole time series. In addition, in order to avoid a systemic bias in the test data, the identical standardization is used for the test set.

For a training data where $X = x_t, x_{t+1}, x_{t+2}, ..., x_T$ and a test data where $Y = y_t, y_{t+1}, x_{t+2}, ...y_T$, the standardized data, X' and Y' will be transformed as following:

$$X' = \frac{X - \mu_x}{\sigma_x} \quad ; \quad Y' = \frac{Y - \mu_x}{\sigma_x}$$

where μ_x and σ_x are the mean and standard deviation of training data respectively.

Figure 3.8 shows the data distribution of the training and test data after standardization. This preprocessing step is not necessary for statistical methods and hence it is not applied to ARIMA model. Arguably, the data distribution does not resemble the ideal normal distribution since there are multiple peaks in the distribution and they are not centered exactly at zero. This is expected since the standardization is not done using the whole data but this compromise is unavoidable in order to prevent data leakage during the process.

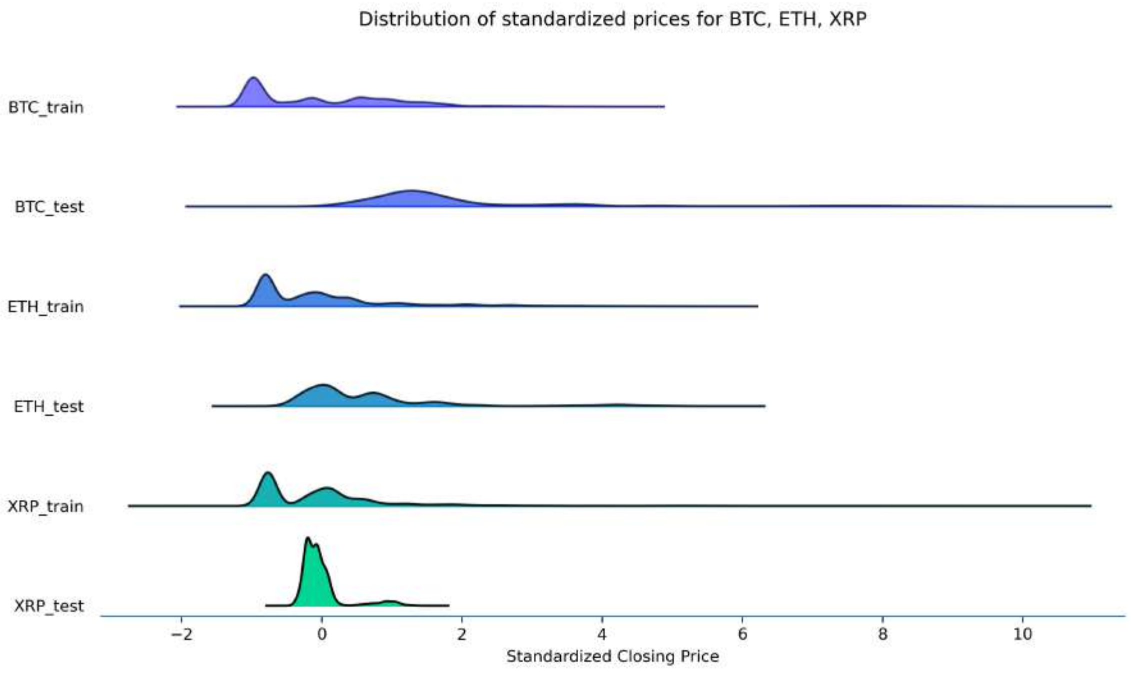

Figure 3.8: Data distribution of training data and test data after standardization

Time Series		*t*-Statistic	*p*-Value
Default	BTC	1.47	0.9974
	ETH	0.1538	0.9695
	XRP	**-3.2154**	**0.0191**
Log Transformed	BTC	-0.888	0.7919
	ETH	-1.6449	0.4597
	XRP	-2.3832	0.1465
Differenced Log Transformed	BTC	-30.1898	0.0000
	ETH	-31.5264	0.0000
	XRP	-28.6095	0.0000

Table 3.4: Augmented Dickey–Fuller (ADF) unit root test of default and transformed cryptocurrency time series at 95% confidence of -2.8616

3.3.2 Fractional Differentiation

As described in previous section, the log return of all cryptocurrency series (first order differenced log prices) exhibits stationarity properties. However, De Prado (2018) argues that integer differentiation or changes in log prices make the time series stationary at the expense of model's predictive power since these data transformations erase all the memory needed to produce good forecast. In order to address the dilemma of either having a memory-less stationary log return series or having a non-stationary prices with memory, the author proposes a transformation method, that is called fractional diffentiation with fixed-width windown fracdiff (FFD). This pre-processing method ensures stationarity but in the meanwhile preserves certain degree of memory in the series. The effectiveness of this pre-processing method will then be compared against the conventional pre-processing method as discussed in earlier section. The following subsection briefly describes the FFD method and how the minimum value of fractional differentiation, d that passes the ADF test is selected. The ADF test result on fractionally differentiated series is included in Table 3.4. It is worth mentioning that XRP series is not included in the table because the series passes the ADF test by default as presented in Table 3.7. The following subsection describes the theoretical background of a fractional differentiation method called fixed-width window fracdiff proposed by De Prado (2018).

3.3.2.1 Fixed-width window Fracdiff (FFD)

Given a time series, X_t where t = 1,..,T, and let B denote the backshift operator, then the previous element in the time series can be defined as $X_{t-1} = BX_t$. The backshift operation can be operated k arbitrary times so we can have a generalized formula of $B^k X_t = X_{t-k}$ for $k \geq 0$.

From the binomial series: $(1+x)^d = \sum_{k=0}^{\infty} \begin{pmatrix} d \\ k \end{pmatrix} x^k$ where d is allowed to be a real number, it is defined as the differencing value in a fractional model. As demonstrated in De Prado (2018), combining the backshift operation and the binomial series expansion, we can generate a sequence of weights at various differencing value, d.

The value of weights determine the amount of memory from the past to be included in estimating the present value:

$$
\begin{aligned}
(1-B)^d = \sum_{k=0}^{\infty} \binom{d}{k} (-B)^k &= \sum_{k=0}^{\infty} (-B)^k \prod_{i=0}^{k-1} \frac{d-i}{k-i} \\
&= 1 - dB + \frac{d(d-1)}{2!}B^2 - \frac{d(d-1)(d-2)}{3!}B^3 + \cdots
\end{aligned}
$$

Theoretically speaking, an element in the time series at current timestep is then can be estimated as the weighted sum of all the past values from previous timestep:

$$\tilde{X}_t = \sum_{k=0}^{\infty} \omega_k X_{t-k} \tag{3.2}$$

where weights w is equivalent to

$$\omega_k = \left\{ 1, -d, \frac{d(d-1)}{2!}, -\frac{d(d-1)(d-2)}{3!}, \ldots, (-1)^k \prod_{i=0}^{k-1} \frac{d-i}{k!}, \ldots \right\} \tag{3.3}$$

Figure 3.9 illustrate the concept of fractional differentiation. Lets consider $d \in [0, 1]$ at the increments of 0.2, using the various differencing values, a sequence of weights is generated. For $d = 0$, all weights other than $w_0 = 1$ is cancelled. This is the case when no differencing is done. For $d = 1$, it is the case when the standard first-order integer differencing is performed. Under this scenario, all weights are cancelled except for $w_0 = 1$ and $w_1 = -1$. Anywhere in between these two cases, all weights after $k = 1$ are not cut-off completely as in $d = 0$ but they lie in between 0 and -1 and hence by finding the optimal d value that lies between zero and one, we can have a sweet spot where the stationarity is obtained but with the maximum amount of memory preserved. Note that d is not necessarily bounded [0,1] as long as it is any positive fractional number (De Prado, 2018). But in my experiment, I narrow down the search for optimal d value that lies in between this boundary.

From equation 3.3, we can compute the weights iteratively as:

$$\omega_k = -\omega_{k-1} \frac{d-k+1}{k}$$

Ideally we can compute an infinite series of weights however in practice, since we only have finite amount of data, we cannot calculate frationally differentiated value $\tilde{X}_t$ as define in equation 3.2. Alternatively, we generate a finite weight sequence using a fixed-width window and fractionally differentiate the series using $\tilde{X}_t = \sum_{k=0}^{l^*} \tilde{\omega}_k X_{t-k}$ where the new variable, $\tilde{\omega_k}$ is defined as

$$\tilde{\omega}_k = \begin{cases} \omega_k & \text{if } k \leq l^* \\ 0 & \text{if } k > l^* \end{cases}$$

The key idea is to control the length of the weight sequence, l^* using a threshold value, τ where the weights are dropped out when the weight modulus, $|\omega_k|$ falls below the threshold τ. Putting everything together, the length of the weight series and hence the resulting differentiated series are both influenced by τ and differencing value, d. It is more complicated to find the best combination of these two variable that yield the optimal result in term of retaining optimal memory and achieving stationary. Hence, I fix $\tau = 1e-4$ to generate different fractionally differentiated series at various differencing value, d to study their corresponding correlation and stationarity. The optimal d value that strikes the best balance will then be selected to generate the fractionally differentiated series. The result is discussed in the subsequent section.

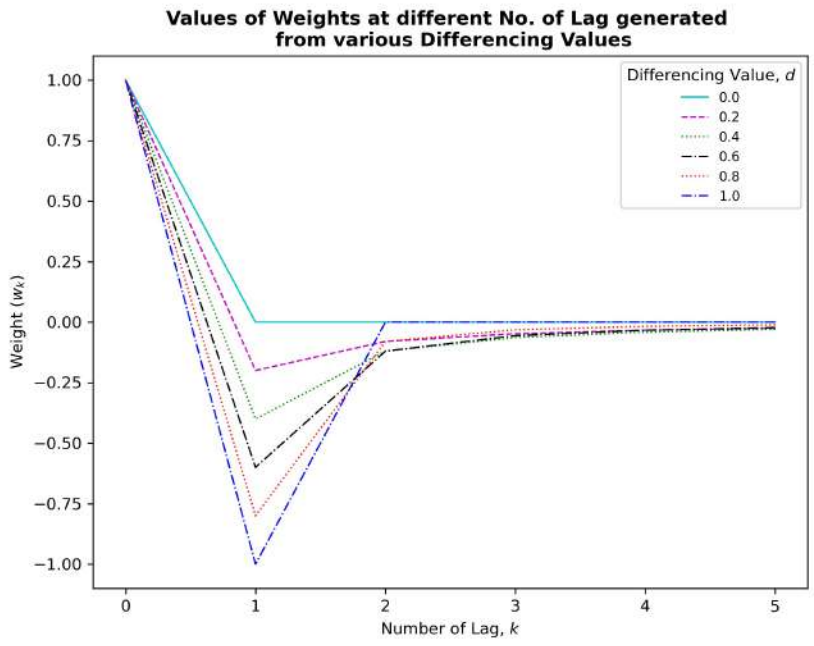

Figure 3.9: Values of weights with each line corresponds to each sequence of weights generated from different differencing value, d in 0.2 increments

3.3.2.2 Finding the optimal differencing value, d

By applying the FFD method as explained in previous section on all cryptocurrency series at various d values, we can compare the resulting correlation (how much memory is preserved) and the ADF test statistic (whether the transformed series achieves stationarity) at each differencing value as shown in Tables 3.5, 3.6 and 3.7. For BTC series, the optimal d value is 0.4 as this is the point when ADF test statistic reaches 95% critical value with p-Value smaller than 0.05. Note that with $d = 0.4$ differencing, the correlation between the fractionally differenced series and the original series is over 95.5% as compared to the 2.03% with first-order differencing. The indicates that not only the FFD series is stationary but also retains considerable amount of memory of the original series. The same analysis is done on ETH and XRP series and the minimum d value for each series is highlighted in their respective table. Note that no differencing is needed for XRP series as the original series is shown stationary. The following tables show the result of correlation and ADF test statistic with $d \in [0,1]$ in 0.1 increments for XRP, ETH and XRP at 95% confidence level of -2.8616:

d	Correlation	p-Values	t-Statistic
0	1	0.9974	1.47
0.1	0.9987	0.969	0.1446
0.2	0.9926	0.6445	-1.2661
0.3	0.9798	0.1049	-2.5451
0.4	**0.9552**	**0.003**	**-3.7937**
0.5	0.907	0.000	-5.1408
0.6	0.8166	0.000	-6.6392
0.7	0.6681	0.000	-8.5175
0.8	0.479	0.000	-10.9123
0.9	0.2807	0.000	-14.8098
1	0.0203	0.000	-29.8917

Table 3.5: ADF test result of fractionally differentiated BTC with different d order

d	Correlation	p-Values	t-Statistic
0	1	0.9695	0.1538
0.1	0.9979	0.8156	-0.812
0.2	0.9882	0.2657	-2.0487
0.3	**0.9682**	**0.0113**	**-3.3896**
0.4	0.9306	0.0001	-4.8081
0.5	0.8612	0	-6.3581
0.6	0.7438	0	-8.0288
0.7	0.5763	0	-10.1098
0.8	0.3937	0	-12.7263
0.9	0.2243	0	-16.7917
1	0.0192	0	-28.9659

Table 3.6: ADF test result of fractionally differentiated ETH with different d order.

d	Correlation	p-Values	t-Statistic
0	**1**	**0.0191**	**-3.2154**
0.1	0.9943	0.0016	-3.9702
0.2	0.9704	0	-5.108
0.3	0.9264	0	-6.5106
0.4	0.8553	0	-8.1978
0.5	0.7458	0	-10.1351
0.6	0.5944	0	-12.3491
0.7	0.4241	0	-14.9855
0.8	0.2732	0	-18.0935
0.9	0.1513	0	-22.1283
1	0.0151	0	-29.1353

Table 3.7: ADF test result of fractionally differentiated XRP with different d order.

3.4 Model Selection

A set of candidate models can be generated using different model configurations for each ARIMA, MLP, LSTM and α-RNN model class. Briefly speaking, model selection is the process of selecting the most appropriate model from these set of candidate models. In this comparative study, the corresponding most appropriate model from each technique class will be compared based on the evaluate metrics defined in section 3.5 after their respective model selection process. Note that the selected most appropriate model may simply be the lucky winner among the few selected candidate models as no all-inclusive candidate models are included in this study. However, methods are carefully employed in this selection process to establish models that can offer the best possible predictive performance.

Existing literature proposes various methods for model selection that are justified under different circumstances (Ding et al., 2018). I thus choose the representative approach that is suitable to the context of my study. An information criteria based on maximum likelihood function is used for model selection of ARIMA modeling and for MLP, LSTM and α-RNN, cross validation is used since it is a popular method of model selection in machine learning practices. Through a brute force search for the appropriate oders of ARIMA model that provides the lowest information criterion, the appropriate model for this class of models is constructed. In the context of ML, DL and hybrid approaches, the appropriate model refers to a model that provides the best mean accuracy over the 10 equal sized subsamples of training data in cross validation which is illustrated in Figure 3.10. Under this circumstance, the subjects of selection are the number of neurons and other hyperparameters needed during the training process. The following sections elaborate the key components of these selection processes in detail.

3.4.1 Akaike's Information Criterion (AIC)

Akaike's Information Criterion (AIC) is useful for selecting the model parameters, p and q of an ARIMA model but it tends to be a poor guide in determining the order of differencing d. This is because after a differencing operation, the data on which the likelihood is computed are altered. As a result, the AIC values of models with differencing order are not comparable (Hyndman and Athanasopoulos, 2018). The information criteria can be defined as following:

$$\text{AIC} = -2\log(L) + 2(p + q + k + 1)$$

where p and q are the order of the model; L is the likelihood of the data; if $c \neq 0$ then $k = 1$ otherwise $k = 0$ if $c = 0$.

According to the analysis done in section 3.3.1, the first order differencing on a log transformed XRP and ETH series will make them stationary but the default XRP series is stationary according to ADF test result as tabulated in Table 3.4. Hence, d will be set as 1 for the log transformed XRP and ETH whereas d will be defined as 0 for original XRP series during the grid search for p and q order of the ARIMA model for each series. Section 3.4.3.1 will further discuss how this grid search for these two model parameters is done.

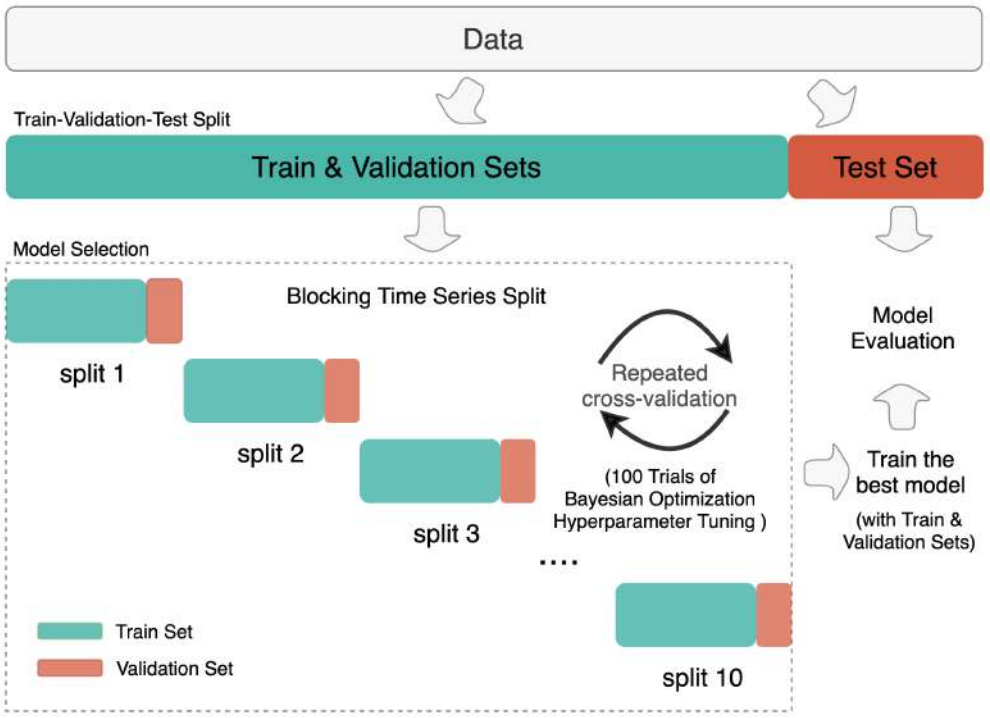
Data
Train-Validation-Test Split
Train & Validation Sets
Test Set
Model Selection
Blocking Time Series Split
split 1
split 2
split 3
....
split 10
Repeated cross-validation
(100 Trials of Bayesian Optimization Hyperparameter Tuning)
Train Set
Validation Set
Model Evaluation
Train the best model
(with Train & Validation Sets)

Figure 3.10: Cross Validation with Blocking Time Series Split

3.4.2 Cross Validation (CV)

An empirical study done by Bergmeir et al. (2018) shows that cross validation (CV) procedures yield more accurate estimate than a simple "hold-out" evaluation approach which is widely used in time series forecasting because CV procedure can adequately minimize overfitting during training. The key concept behind CV procedure is to partition the data once or multiple times into training set(s) and validation set(s). The training part is used to train the algorithm whereas the validation part is used to estimate the predictive performance of the algorithm. The method with the lowest estimated error will be selected in the CV procedure. As Jansen (2020) suggested, the risk of multiple testing bias may arise if many CV iterations result from model selection involving hyperparameter tuning. Consequently, the resulting validation score is then no longer a fair estimate of the generalization error. This study adopts the author's three-way split of the data as illustrated in Figure 3.10. The first 80% of the data is partitioned to be used in cross-validation and the remainder is hold out for model evaluation. The first partition is split into 10 equal sized training and validation sets. How the split is performed will be further explained in the following subsection. As illustrated in Figure 3.10, this CV procedure is simply an approach of generating training and validation sets for the process of hyperparameter tuning. The Bayesian Optimization algorithm is set to search for 100 different combination of hyperparameters and each trial is evaluated based on the average RMSE score of the 10 validation sets partitioned from the CV procedure.

3.4.2.1 Blocking Time Series Split

The most popular way to partition the original data into subset of train and validation data is the K-fold CV. This approach is arguably not suitable for finance time series due to two main reasons. Firstly, the time series in finance cannot be assumed to be drawn from an independent and identically distributed (IID) process. In addition, when we shuffle the training and test set randomly, we will end up using values from the future to predict the values in the past within certain folds. The second reason is that the procedure could lead to selection bias since the test set is used several times in the process of developing a model (De Prado, 2018).

Another common approach is to have multiple folds across different time period, with the training set expanding in each fold as if it is "walking forward" as demonstrated in Hyndman and Athanasopoulos (2018). Although this approach does not violate the temporal dependency of the observations but it does not address the issues discussed earlier. On top of that, since the test set is used multiple times with the overlapping folds, it could lead to multiple testing and selection bias. Instead, I use a "blocking time series split" where 10 folds of non-overlapping train and validation set with equal amount of observations are used to calculate an average model error for tuning the hyperparameters. As implemented by Bergmeir et al. (2018), the suitability of this approach will also be investigated using a residual series constructed from the predictions across all folds. Ljung-Box test is then applied to this residual series to study the existence of remaining autocorrelation. Only models with residual series that pass the Ljung-Box test are considered.

3.4.3 Model Parameter Selection and Hyperparameter Tuning

A non-seasonal ARIMA implemented in this study is a classic parametric modeling method to time series. It contains few model parameters, p, d and q that need to be set before predictions can be done. Conventionally, estimation of these model parameters is done using autocorrelation and partial autocorrelation plots (Hyndman and Athanasopoulos, 2018). However, this heuristic search for optimal p and q is automated using grid search and the process will be elaborated in next subsection.

On the other hand, in machine learning and deep learning, model parameters refer to parameters that can be learned during training process. In this context, hyperparamters are the parameters that need to be defined before model training. Although in principle, the hyperparameters affect only the quality of the learning process, its configuration could still have an impact on the convergence and generalization of the model. As a result, the prediction accuracy of the model could be affected by the hyperparameter settings and the best settings are usually differ for different datasets (Zheng, 2015). Hence, the hyperparameters of each MLP, LSTM and α-RNN need to be tuned for each time series as illustrated in Table 3.3.

As mentioned earlier, the most basic topology of MLP, LSTM and α-RNN is implemented in this study so a single hidden layer is fixed for their respective structure but other hyperparameters related to the network structure such as hidden units and activation functions are still required to be estimated. These hyperparameters control the complexity of the models and poor choices for these values can lead to having overfitted models. An overfitted model is a model that contains higher degree of freedom than can be justified by the available data. Under such circumstance, the model will attempt to fit too closely with the training data and will eventually result in poor generalization with unseen data. An appropriate control of model capacity can avoid overfitting and this can be done through a broad range of regularization techniques such as dropout and L1 regularizer. These regularization hyperparameters and other hyperparameters that come from the training process such as batch size will need to be tuned in MLP, LSTM and α-RNN models. Since there are numerous hyperparameters involved, an efficient tuning approach to search for the optimal settings is required. The method selected is the Bayesian Optimization which will be elaborated further in the subsequent subsection.

3.4.3.1 Grid Search for p and q in ARIMA

The 3 parameters in ARIMA model, p, d and q can be estimated repeatedly by trial and error from reviewing diagnostic plots such as autocorrelation and partial autocorrelation plots but it is an exhaustive effort. Hence, the examination of appropriate model parameters in the ARIMA model is automated by using a grid search procedure. As a rule of thumb, the grid of search is set in a range of 1 to 5 for each model parameter. The ARIMA model is then fitted with the training data and evaluated using AIC value as discussed in section 3.4.1. The hyperparameter combination with the lowest AIC value will then be selected to perform one-step ahead forecasting for the entire period of the test data and to be evaluated using the error metrics as describe in section 3.5 for each BTC, ETH and XRP series.

3.4.3.2 Bayesian Optimization for hyperparameters in MLP, LSTM and α-RNN

Grid search is the simplest but yet the most time consuming way of performing hyperparameter optimization. Since it methodically select a grid of hyperparameter values and evaluates the model for each combination to return the best hyperparameter, this approach is practically inept for model with numerous hyperparameters. In addition, it is relatively inefficient as it does not keep track of the past results. Unlike the aforementioned method which execute each search in isolation, Bayesian Optimization tuning method is a smarter tuning technique that choose a few hyperparameter settings, examine their performance and then decide where to sample next. In another words, this method keep track of the past search results and then narrow down the search region to efficiently look for hyperparameter combinations that yield better model performance. Therefore, the search for optimal hyperparameters of MLP, LSTM and α-RNN models is automated using Bayesian Optimization as illustrated in diagram 3.10. The tuning process is implemented with the python library, HyperOpt that is created by (Bergstra et al., 2013). Using the optimal hyperparameters found from each experiment, the model is refitted with the entire training dataset and the out-of sample performance is evaluated using the test dataset with a set of evaluation metrics presented in the following section.

3.5 Performance Evaluation

The commonly used error metrics for regression problem are Mean Absolute Error (MAE), Root Mean Square Error (RMSE), R^2 score and Mean Absolute Percentage Error (MAPE). Just like the "no free lunch" theorem, there is no one size fits all evaluation measurement. Each one of them has its strengths and limitations, some of the evaluation metrics are context dependent. This means that they cannot be used to make a comparison across the three BTC, ETH and XRP series. Therefore, they are usually used together so that they can complement each other to draw a more comprehensive evaluation.

On top of that, these error metrics cannot differentiate whether the errors are overestimated errors or underestimated errors. This is particularly important in finance where we also want to evaluate the capability of the forecast model in term of directional accuracy. Hence, this study also includes another existing metric, namely, Mean Directional Accuracy (MDA) and a proposed evaluation method called Normalized Weight Movement Indicator (NWMI) to address that limitation. Each of the following subsection explains how each of them is defined, how they should be interpreted and what are their respective limitations.

3.5.1 Mean Absolute Error (MAE)

Mean absolute error is the average of absolute difference between the actual values and the forecasts. It measures how far the predictions are from the actual values on average without evaluating the direction of the error. The smaller the MAE value, the better the predictive power of the model. The key shortcoming of this evaluation metric is that it disregards the relative size of the error. This means that we cannot differentiate the large errors from the small ones. This pitfall can be compensated with other metric like RMSE to check if the errors are larger.

$$\mathrm{MAE}(x, \hat{x}) = \frac{1}{n}\sum_{t=1}^{n} |x_t - \hat{x}_t|$$

where n is equal to the number of test data.

3.5.2 Root Mean Square Error (RMSE)

While mean squared error (MSE) is the average of the square of the difference between actual values and predicted values, RMSE is simply the square root of MSE. This metric is usually more preferrable than MSE because the RMSE value is in the same unit as the predicted value. This makes it more interpretable than MSE and hence this is chosen over MSE. This metric is always positive and the smaller values are better. As compared to MAE, it gives a higher weight for large errors because of the square operation. It can be used with MAE to determine whether the predictions contain infrequent large errors. The bigger the difference between these two metrics, the more inconsistent the error size.

$$\mathrm{RMSE}(x, \hat{x}) = \sqrt{\frac{\sum_{t=1}^{n} (x_t - \hat{x}_t)^2}{n}}$$

where n is equal to the number of test data.

3.5.3 R^2 score

As we have seen how MAE and RMSE are calculated, their measurements depend on the context. Hence the metrics are incomparable between the three time series especially when their price range disparity is huge. The idea of R^2 score is to compare the accuracy of the model against the mean line of the training dataset. In another word, it tells us how good the model as compared to a naive model that always predicts the expected value of the target as predictions. The best possible score is 1.0 and it can be negative when the model is way worse than the constant model.

$$R^2(x, \hat{x}) = 1 - \frac{\sum_{t=1}^{n} (x_t - \hat{x}_t)^2}{\sum_{t=1}^{n} (x_t - \bar{x})^2}$$

where n is equal to the number of test data and $\bar{x} = \frac{1}{n} \sum_{t=1}^{n} x_t$

3.5.4 Mean Absolute Percentage Error (MAPE)

Mean Absolute Percentage Error (MAPE) calculate the MAE in percentage terms. Similar to R^2, by using MAPE, we can compare the predictions of different series in different scales. In this case, the accuracy of the forecast of BTC, ETH and XRP is comparable using this metric, although they are in different price range.

$$\text{MAPE}(x, \hat{x}) = \frac{1}{n} \sum_{t=1}^{n} \frac{|x_t - \hat{x}_t|}{\max(\epsilon, |x_t|)}$$

where n is equal to the number of test data and ϵ is an arbitrary small positive values to prevent undefined results when $x_t = 0$.

3.5.5 Mean Directional Accuracy (MDA)

All of the evaluation metrics discussed so far measure the model performance in term of how well it predicts the actual price. In the context of algorithmic trading, we are often interested more in the directional movement of the price. By comparing the predicted direction to the actual realized direction, MDA calculates to which degree the model accurately predicts the directional changes from one time step to the next time step. The score ranges from 0 to 1. With a score of 0.5 means the model predicts the direction correctly half of the time.

$$\text{MDA}(x, \hat{x}) \frac{1}{n} \sum_{t} \mathbf{1}_{\text{sgn}(x_t - x_{t-1}) = \text{sgn}(\hat{x}_t - x_{t-1})}$$

where n is equal to the number of test data, $\mathbf{1}$ is the indicator function and $\text{sgn}(\cdot)$ is the sign function.

3.5.6 Normalized Weighted Movement Indicator (NWMI)

In trading, it is especially crucial for a model to capture abrupt price movements because these large price movements provide better profit margins. MDA metric does not account for the magnitude of each movement as each correct directional prediction contribute equally to the final score. In another words, MDA does not place a higher weight to a correct prediction when it is a steep upward trend or a steep downward trend. To remedy this drawback, I propose an evaluation metric that is defined as the following:

$$\hat{y}_t = \mathrm{sgn}(\hat{x}_t - \hat{x}_{t-1})$$

$$y_t^* = \mathrm{sgn}(x_t - x_{t-1})$$

$$s(y_t) = \begin{cases} \frac{x_t - x_{t-1}}{x_{t-1}}, & \text{if } y_t = 1 \\ \frac{x_{t-1} - x_t}{x_t}, & \text{if } y_t = -1 \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

$$\mathrm{S} = \sum_{t=1}^{n} s(\hat{y}_t)$$

$$\mathrm{S}^* = \sum_{t=1}^{n} s(y_t^*)$$

$$\mathrm{NWMI} = \frac{\mathrm{S}}{\mathrm{S}^*}$$

where n is the number of test data and $\mathrm{sgn}(\cdot)$ is the sign function; At t time step: x_t is the actual price whereas $\hat{x}_t$ is the predicted price and y_t^* is the actual direction whereas $\hat{y}_t$ is the forecast direction; Given a function $s(y_t)$ that calculates an intermediate score for a direction: S is the summation of all intermediate scores obtained from all predicted directions meanwhile S^* is the ideal sum obtained from all actual directions.

When it is an actual upward trend, $y_t^* = 1$ and when it is an actual downward trend, $y_t^* = -1$ otherwise, $y_t^* = 0$. On the other hand, when it is a predicted upward trend, $\hat{y}_t = 1$ and when it is a predicted downward trend, $\hat{y}_t = -1$ otherwise, $\hat{y}_t = 0$. Hence, as defined in equation 3.4, when the actual direction and the forecast direction do not match, $s(y_t)$ will be negative and this wrong prediction will be penalized according to the magnitude of relative change. In such manner, the directional accuracy is weighted according to the size of upward or downward movement. For instance, a model will obtain a higher score of $s(y_t)$ for predicting correctly a large movement than for predicting correctly a small movement whereas it will get a lesser penalty for a wrong prediction for a small movement than a wrong prediction for a large movement.

Note that a wrong prediction for an actual constant movement will not be penalized so this is the main drawback of this metric but this can easily be compensated by evaluating the model with other metrics like RMSE and MAE. In addition, the main objective of this metric is to evaluate the model capacity in predicting large movement and since there are only a few occasions when the price change is actually constant across over 40,000 observations so the effect is trivial.

Subsequently, the score obtained at every time step of the training data will be summed up altogether to obtain a total score of S. In order to make the score comparable between different datasets, the model score is then measured against the ideal score, S^*. So ideally, when the forecast directions match with the actual directions closely, S score will be very close to S^* and hence NWMI will be close to 1. Otherwise, if the model predicts the directions wrongly especially when it misses out too many steep upward or downward movements, S score could be negative since the model can be arbitrarily worse.

A simple scenario as following is used to illustrate how NWMI provides additional intuition and evaluate how well a model capture the large directional movements. Consider a series, x_t contains the 6 actual predictions in the test set and m_t, n_t, h_t are the one-step ahead forecast for 6 time steps made by three different models.

Let the last price in the training set be $x_0 = 5$, we can then construct the price movement for each case at every time step by extracting the sign of the difference between the value at current time step and the value at next time step as shown below:

Actual Prices: $x_t = [10, 40, 15, 10, 20, 25] \rightarrow (\uparrow, \boldsymbol{\uparrow}, \boldsymbol{\downarrow}, \downarrow, \uparrow, \uparrow)$
Model 1: $m_t = [3, 10, 5, 15, 5, 10] \rightarrow (\downarrow, \uparrow, \downarrow, \uparrow, \downarrow, \uparrow)$
Model 2: $n_t = [2, 15, 20, 18, 15, 20] \rightarrow (\downarrow, \uparrow, \uparrow, \downarrow, \downarrow, \uparrow)$
Model 3: $h_t = [20, 15, 20, 15, 10, 20] \rightarrow (\uparrow, \downarrow, \uparrow, \uparrow, \downarrow, \uparrow)$

	RMSE	MDA	NWMI
Model 1	14.755	0.5	0.483
Model 2	10.876	0.5	0.287
Model 3	11.339	0.5	-0.017

The highlighted arrows are the two respective steepest increment and decrement which carry the highest weight for accurate prediction. As we can see all three models get only half of the directional predictions correctly so all of them have an equal score of 0.5 for the MDA measurement. On the other hand, since RMSE only gives us an idea how the predicted values are closely matched to the actual values regardless of the direction, even though Model 3 missed out the two important price movements, it still has the best score of RMSE. Model 1 and Model 2 both have similar predictions but with Model 2 missed out one of the important price movements and hence it scores lower in NWMI metric as compared to Model 1.

3.5.6.1 The expected NWMI of random models

To take a step further in the analysis with NWMI, an expected NWMI value from models that randomly predict the price movements is calculated so that each model performance can be measured against the expected NWMI score of some random guessing. 100 sets of random directional predictions are generated alongside with each model that uses standard preprocessing methods as described in Table 3.3. A confidence level is then calculated with the total 3000 NWMI scores obtained from the random models and the expected value is compared against the NWMI score obtained from each model.

4. Result

The experiments described in Table 3.3 has four objectives. First, to evaluate which technique is more suitable for predicting prices of Bitcoin, Ethereum and Ripple. Second, to find an optimal value for the window size for MLP, LSTM and α-RNN models. Third, to investigate how well the models are able to accurately predict large movements. Finally, it will be assessed whether fractional differencing improves model performance. In the next subsection, I present the results obtained by the standard preprocessing methods described in Section 3.3.1 and draw conclusions on the first two questions. A proposed evaluation metric, NWMI as explained in Section 3.5.6 is used to assess how well the models predict the large movements. By assessing the results of this measurement along with other conventional accuracy measurements, as described in section 3.5, we can add another layer of complexity to the evaluation of model performance. The last subsection of this chapter presents results obtained with fractionally differentiated series of Bitcoin and Ethereum. The focus of this subsection is to present results that answer the last question above.

4.1 Standard Preprocessing

This section includes results obtained from experiment done with standard preprocessing as described earlier. In addition to comparing the model performance with the aforementioned error metrics, the predictions of each model are first compared visually. Figure 4.1 shows the comparison of the four forecasting methods of interest for their respective the in-sample and out-of-sample predictions for BTC series. It is noticeable that, with the exception of ARIMA, the other three methods cannot predict the prices increase in early 2021, which contributes to the very high RMSE value. We can also notice that predictions of α-RNN show a similar pattern to LSTM model's predictions, with RNN having a slightly better predictions over a slightly longer time period. Among the four models, predictions of MLP model show the largest difference from actual prices.

Comparing the results of different window sizes used for MLP, LSTM and α-RNN, there are no noticeable performance differences between the three window sizes for these visual inspections. The in-sample and out-of-sample predictions for the other two series, ETH and XRP, using other window sizes show a similar pattern, with the ARIMA model able to predict actual prices more accurately than the other three models throughout the test period. Moreover, LSTM, MLP and α-RNN models have better predictions for ETH and XRP compared to BTC. These plots for other experiments can be found in the appendix.

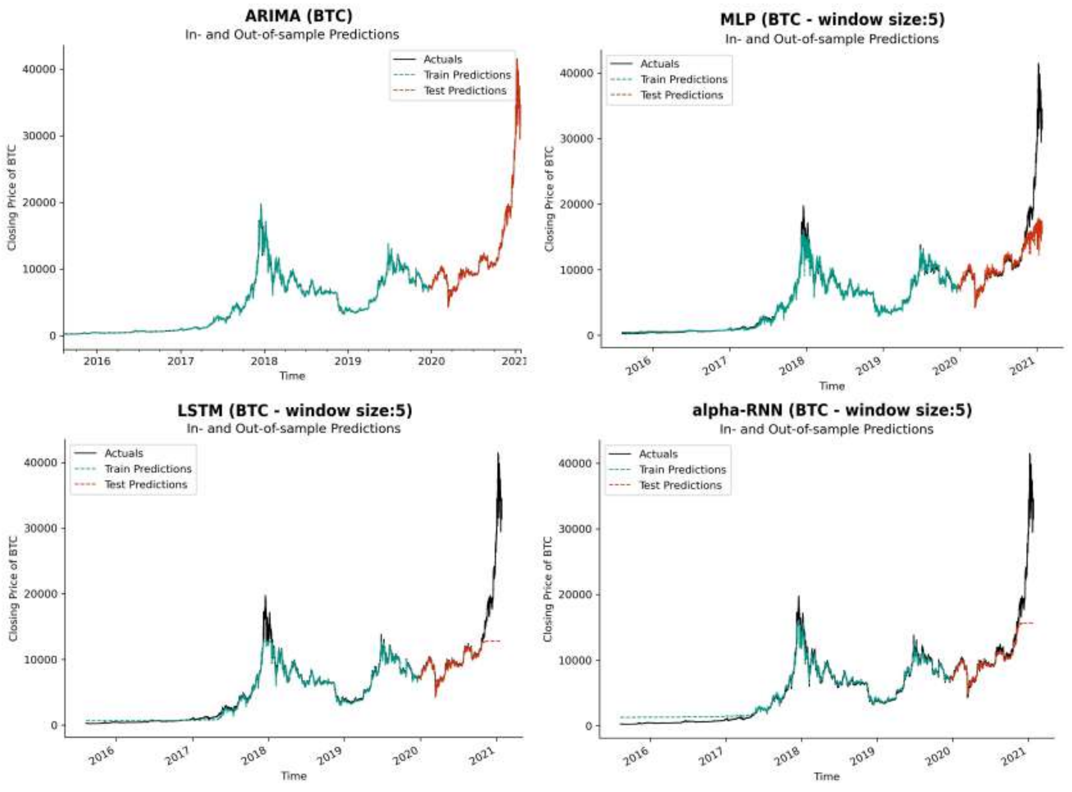

Figure 4.1: In-sample and out-of-sample forecasts of ARIMA(1,1,2), MLP, LSTM and α-RNN (with window size of 5) for BTC Series

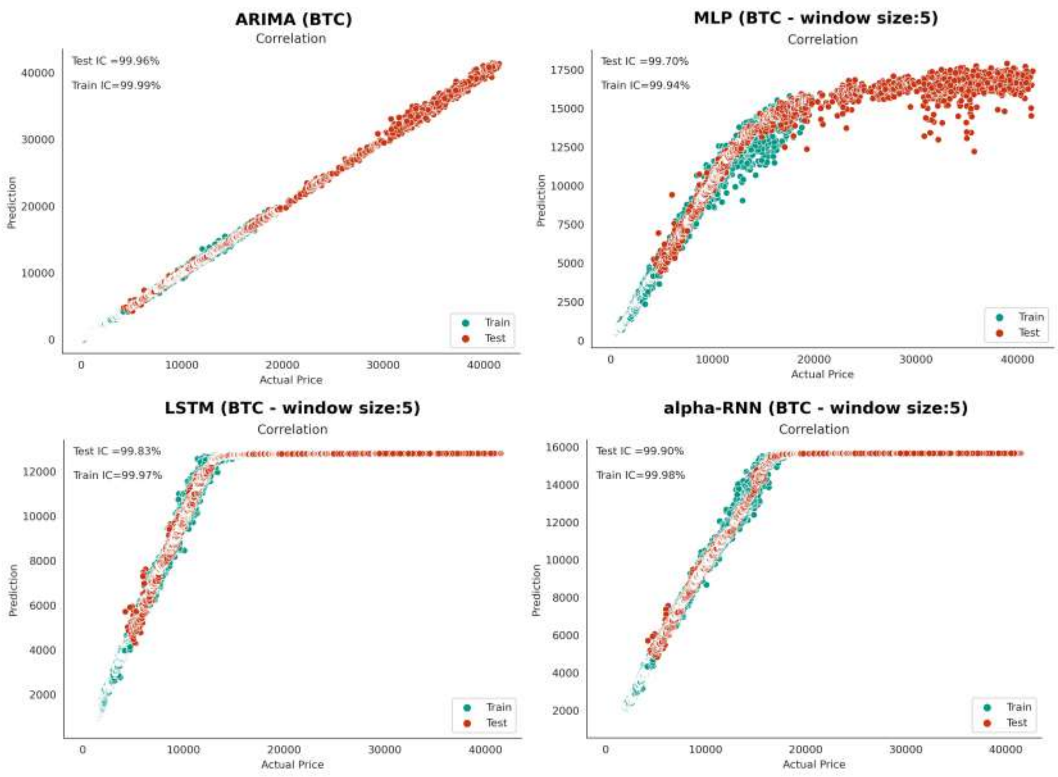

Figure 4.2: Correlations of actual prices and predictions of ARIMA(1,1,2), MLP, LSTM and α-RNN (with window size of 5) for BTC Series

The correlation between the actual prices and predicted prices is plotted to examine the predictive power of each model. The results are compared side by side as shown in Figure 4.2. With a result of 99.99 % of Spearman's rank correlation coefficient, ARIMA slightly outperforms the other three models. MLP model has the lowest correlation values and generally has the worst fit among the four models.

Tables 4.1, 4.2 and 4.3 present the results of ARIMA and other models with a window size of 5 for predicting prices of BTC, ETH and XRP. On the other hand, Tables 4.4, 4.5 and 4.6 present the results of ARIMA and other models with a window size of 10 for predicting prices of BTC, ETH and XRP. Lastly, Tables 4.7, 4.8 and 4.9 present the results of ARIMA and other models with a window size of 30 for predicting prices of BTC, ETH and XRP. Note that a different window size is not used for the ARIMA model. The results of the three ARIMA models for BTC, ETH and XRP are included in these tables repeatedly to allow direct comparison.

In most cases, ARIMA models outperform all other three models, except for XRP series, where LSTM model with window size of 5 and 10 and α-RNN model with window size of 30 achieve the best overall performance in the conventional error metrics. Another thing we can notice is that, the difference between RMSE score and MAE score is generally very large for LSTM, MLP and α-RNN models, except for XRP series. This indicates that the predictions of these models contain infrequent large errors, which is consistent with the observation made earlier in the in-sample and out-of-sample plot 4.1.

It is evident from the tables that there is no singular optimal value of window size for MLP, LSTM and α-RNN models. As can be seen in Figures 4.3 and 4.4, there is no obvious pattern to whether higher or lower values of the window size give better results. When using LSTM, MLP and α-RNN models with window sizes of 10 and 30, worse results are obtained for BTC and ETH series but it is the opposite for the XRP series. The window size of 5 seems to be suitable for the BTC series when using the LSTM model and the α-RNN model, but not for the ETH and XRP series when using the MLP model. A window size of 5 for XRP series when using the MLP model produces the worst results. As a result, it is not possible to find a clear optimal window size based on the results obtained.

Another observation that can be made from the tables in this subsection is that the optimal values of MDA and NWMI do not match the results of the conventional metric scores except for the XRP series when using the LSTM model with window size of 10 as shown in Table 4.6. This shows that the most accurate models, as measured by the accuracy metrics RMSE, R^2, MAE and MAPE, do not necessarily have the best directional accuracy. It is also noticeable that LSTM has the best directional accuracy, if not the second best in most cases. As mentioned earlier, the accuracy values of LSTM, MLP and α-RNN are significantly affected by the infrequent large errors made in the early 2021. To allow for a direct comparison with ARIMA model, further comparison plots that exclude all predictions from 2021 are created as shown in Figures 4.3, 4.4 and 4.5.

Based on Figure 4.3, ARIMA models still have the optimal scores for MAPE metric for all BTC, ETH and XRP. However, from Figures 4.4 and 4.5, it can be concluded that ARIMA models have relatively worse directional accuracy compared to the other three neural network-based models. Based on the NWMI values, the ARIMA models cannot capture large fluctuations as well as the others. On the other hand, we cannot say whether the LSTM, the MLP or the α-RNN model has the best overall performance in predicting large price fluctuations. Each of these three models has poor performance in predicting the large fluctuations under certain experiments. However, LSTM model has a significant better NWMI score than the MLP and α-RNN models in 4 experiments and the two worst results in other experiments are not much worse than MLP and α-RNN models. This suggests that LSTM model has shown promising capabilities in capturing historical large price fluctuation.

Because of the limitation of the MDA metric described in Section 3.5.5, we can hardly draw an firm conclusion about directional accuracy from assessing the model performance using this metric alone as shown in Figure 4.4. Nonetheless, the proposed metric, NWMI can be assessed alone to evaluate the directional accuracy for large movement. As mentioned in subsection 3.5.6.1, a detailed comparison of NWMI score with an average NWMI value obtained from 3000 random prediction models can be performed to evaluate how much better the models are in terms of directional accuracy than models that guessed randomly. Figure 4.6 shows the comparison of model performance in terms of NWMI scores obtained by standard preprocessing for each model of interest using all predictions. It is shown from Figure 4.6 that, most models do not perform significantly better than the averaged random forecasts in terms of directional accuracy. Nevertheless, the LSTM models still have better overall directional accuracy than the other models.

Model Performance for BTC:

Model		RMSE	R^2	MAE	MAPE	MDA	NWMI
LSTM	Train Error	536.082	0.9816	166.698	23.6376	0.4925	
	Test Error	6317.050	0.2210	127.175	9.8388	0.4908	**0.02311**
MLP	Train Error	365.910	0.9914	123.522	12.4007	0.5113	
	Test Error	5105.390	0.4912	318.220	8.4324	**0.4988**	0.01757
α-RNN	Train Error	566.852	0.9795	260.091	70.3864	0.4833	
	Test Error	5303.020	0.4510	229.774	8.1513	0.4771	0.00694
ARIMA	Train Error	66.790	0.9997	6.382	0.4854	0.4525	
	Test Error	**146.634**	**0.9996**	**27.143**	**0.4600**	0.4587	-0.00546

Table 4.1: LSTM, MLP, α-RNN with window size = 5 and ARIMA(1,1,2)

Model Performance for ETH:

Model		RMSE	R^2	MAE	MAPE	MDA	NWMI
LSTM	Train Error	40.4619	0.9720	9.1257	266.0630	0.4908	
	Test Error	85.9359	0.8853	3.4786	3.2465	**0.4837**	-0.00578
MLP	Train Error	42.0201	0.9698	10.2862	94.4098	0.4708	
	Test Error	80.1122	0.9003	21.8692	9.1580	0.4761	0.01220
α-RNN	Train Error	28.0178	0.9866	14.3654	566.886	0.4755	
	Test Error	40.3371	0.9747	7.3582	3.1625	0.4827	**0.03345**
ARIMA	Train Error	4.7659	0.9996	0.3204	0.8954	0.4521	
	Test Error	**6.0771**	**0.9994**	**1.0662**	**0.6348**	0.4598	0.00033

Table 4.2: LSTM, MLP, α-RNN with window size = 5 and ARIMA(0,1,3)

Model Performance for XRP:

Model		RMSE	R^2	MAE	MAPE	MDA	NWMI
LSTM	Train Error	0.0265	0.9937	0.0027	4.1612	0.4704	
	Test Error	**0.0113**	**0.9873**	**0.0059**	**2.8448**	0.4658	0.00566
MLP	Train Error	0.0946	0.9203	0.0235	55.0563	0.4645	
	Test Error	0.0465	0.7853	0.0498	19.1384	0.4669	0.00362
α-RNN	Train Error	0.0519	0.9760	0.0189	370.888	0.4771	
	Test Error	0.0117	0.9864	0.0076	3.6262	**0.4713**	0.00599
ARIMA	Train Error	0.8303	0.4135	0.0290	12.2056	0.4176	
	Test Error	0.0652	0.8053	0.0302	12.2621	0.4493	**0.00601**

Table 4.3: LSTM, MLP, α-RNN with window size = 5 and ARIMA(3,0,3)

Model Performance for BTC:

Model		RMSE	R^2	MAE	MAPE	MDA	NWMI
LSTM	Train Error	296.341	0.9944	89.2519	6.6776	0.4972	
	Test Error	5086.130	0.4952	393.8050	8.6424	**0.4971**	0.01080
MLP	Train Error	392.930	0.9901	176.3460	16.9972	0.4811	
	Test Error	4862.540	0.5386	487.4230	9.1071	0.4838	-0.03016
α-RNN	Train Error	1013.550	0.9343	508.7720	130.2990	0.4862	
	Test Error	5485.000	0.4129	371.9680	9.2294	0.4802	**0.01966**
ARIMA	Train Error	66.790	0.9997	6.3818	0.4854	0.4525	
	Test Error	**146.634**	**0.9996**	**27.1426**	**0.4600**	0.4587	-0.00545

Table 4.4: LSTM, MLP, α-RNN with window size = 10 and ARIMA(1,1,2)

Model Performance for ETH:

Model		RMSE	R^2	MAE	MAPE	MDA	NWMI
LSTM	Train Error	10.9183	0.9980	4.4714	66.722	0.4868	
	Test Error	18.0051	0.9950	5.0258	2.4396	**0.4973**	0.02800
MLP	Train Error	37.4207	0.9761	4.9340	48.2684	0.5034	
	Test Error	70.8152	0.9221	11.9839	6.1310	0.5104	0.00780
α-RNN	Train Error	36.8294	0.9768	12.0587	819.988	0.4731	
	Test Error	39.7564	0.9755	4.4328	2.6217	0.4834	**0.03223**
ARIMA	Train Error	4.7659	0.9996	0.3204	0.8954	0.4521	
	Test Error	**6.0771**	**0.9994**	**1.0662**	**0.6348**	0.4598	0.00033

Table 4.5: LSTM, MLP, α-RNN with window size = 10 and ARIMA(0,1,3)

Model Performance for XRP:

Model		RMSE	R^2	MAE	MAPE	MDA	NWMI
LSTM	Train Error	0.0294	0.9923	0.0092	38.7717	0.4824	
	Test Error	**0.0163**	**0.9736**	**0.0099**	**4.4321**	**0.4851**	**0.02280**
MLP	Train Error	0.0503	0.9774	0.0165	103.9060	0.4833	
	Test Error	0.0241	0.9421	0.0137	6.4170	0.4751	0.00432
α-RNN	Train Error	0.0906	0.9269	0.0252	791.6480	0.4339	
	Test Error	0.0183	0.9665	0.0134	6.7497	0.4535	-0.01718
ARIMA	Train Error	0.8303	0.4135	0.0290	12.2056	0.4176	
	Test Error	0.0652	0.8053	0.0302	12.2621	0.4493	0.00601

Table 4.6: LSTM, MLP, α-RNN with window size = 10 and ARIMA(3,0,3)

Model Performance for BTC:

Model		RMSE	R^2	MAE	MAPE	MDA	NWMI
LSTM	Train Error	461.854	0.9864	264.298	19.1857	0.5039	
	Test Error	5557.320	0.3979	482.134	10.3210	**0.4990**	**0.01939**
MLP	Train Error	712.313	0.9676	266.782	18.4084	0.4976	
	Test Error	6111.890	0.2717	837.308	14.2083	0.4988	0.01861
α-RNN	Train Error	1212.580	0.9060	642.125	147.3240	0.4460	
	Test Error	6229.130	0.2435	226.130	10.5079	0.4599	0.00633
ARIMA	Train Error	66.790	0.9997	6.382	0.4854	0.4525	
	Test Error	**146.634**	**0.9996**	**27.143**	**0.4600**	0.4587	-0.00545

Table 4.7: LSTM, MLP, α-RNN with window size = 30 and ARIMA(1,1,2)

Model Performance for ETH:

Model		RMSE	R^2	MAE	MAPE	MDA	NWMI
LSTM	Train Error	20.8858	0.9925	10.0121	195.9050	0.4960	
	Test Error	36.6887	0.9791	8.4933	3.9788	**0.4983**	**0.04185**
MLP	Train Error	69.2019	0.9181	21.8049	267.673	0.4896	
	Test Error	122.2140	0.7681	41.9184	16.9795	0.4973	0.00708
α-RNN	Train Error	36.2539	0.9775	13.2026	808.896	0.4784	
	Test Error	38.1213	0.9774	4.5931	2.6147	0.4921	0.02899
ARIMA	Train Error	4.7659	0.9996	0.3204	0.8954	0.4521	
	Test Error	**6.0771**	**0.9994**	**1.0662**	**0.6348**	0.4598	0.00033

Table 4.8: LSTM, MLP, α-RNN with window size = 30 and ARIMA(0,1,3)

Model Performance for XRP:

Model		RMSE	R^2	MAE	MAPE	MDA	NWMI
LSTM	Train Error	0.0261	0.9939	0.0049	27.0479	0.4247	
	Test Error	0.0094	0.9913	0.0057	2.5801	0.4524	0.00576
MLP	Train Error	0.0646	0.9628	0.0155	98.4894	0.4825	
	Test Error	0.0248	0.9391	0.0147	7.0152	**0.4738**	-0.00824
α-RNN	Train Error	0.0385	0.9868	0.0148	153.2500	0.4572	
	Test Error	**0.0080**	**0.9936**	**0.0044**	**2.1397**	0.4548	-0.01843
ARIMA	Train Error	0.8303	0.4135	0.0290	12.2056	0.4176	
	Test Error	0.0652	0.8053	0.0302	12.2621	0.4493	**0.00601**

Table 4.9: LSTM, MLP, α-RNN with window size = 30 and ARIMA(3,0,3)

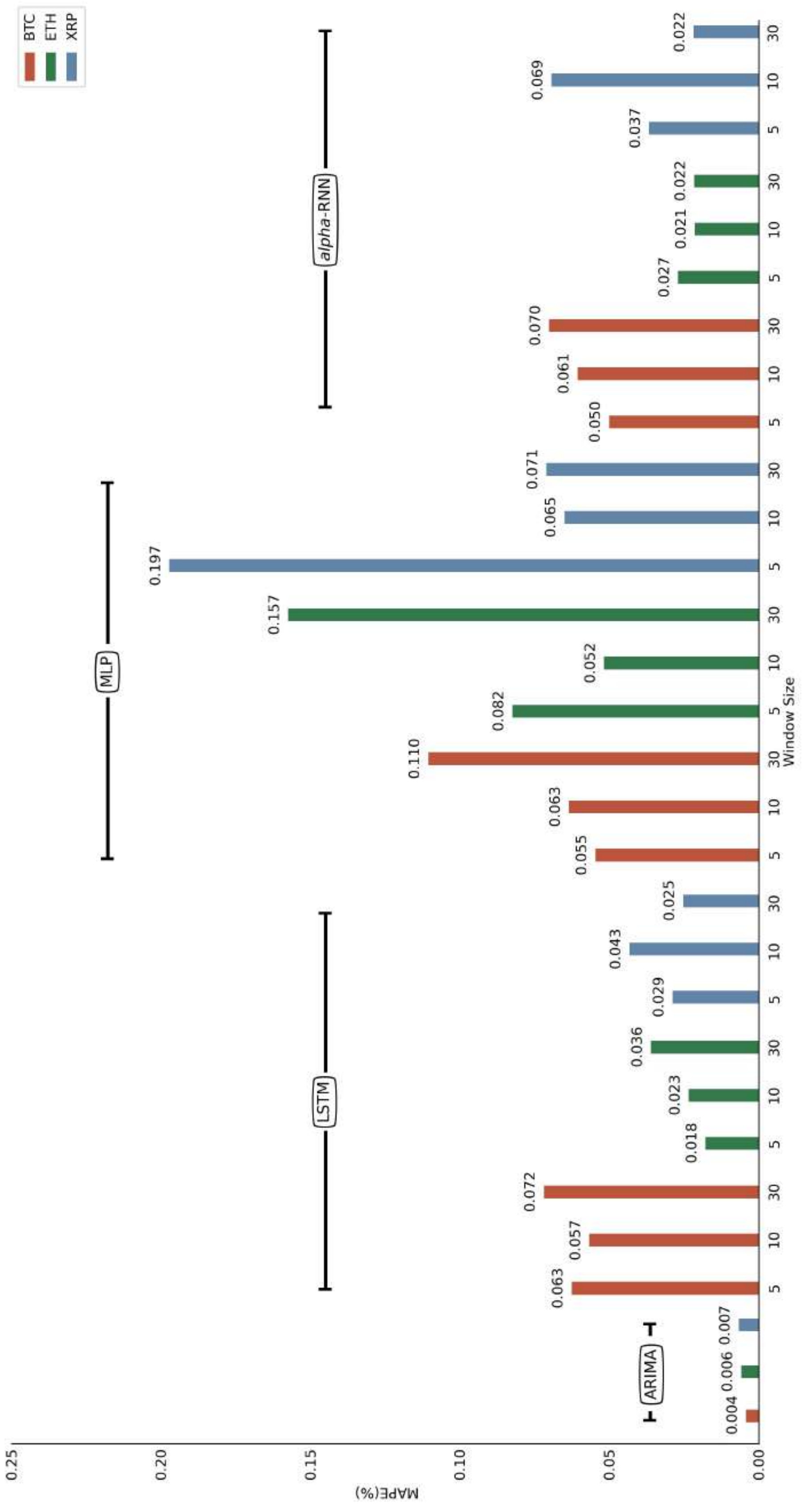

Figure 4.3: Comparison of Model Performance using MAPE metric (excluding the predictions in 2021)

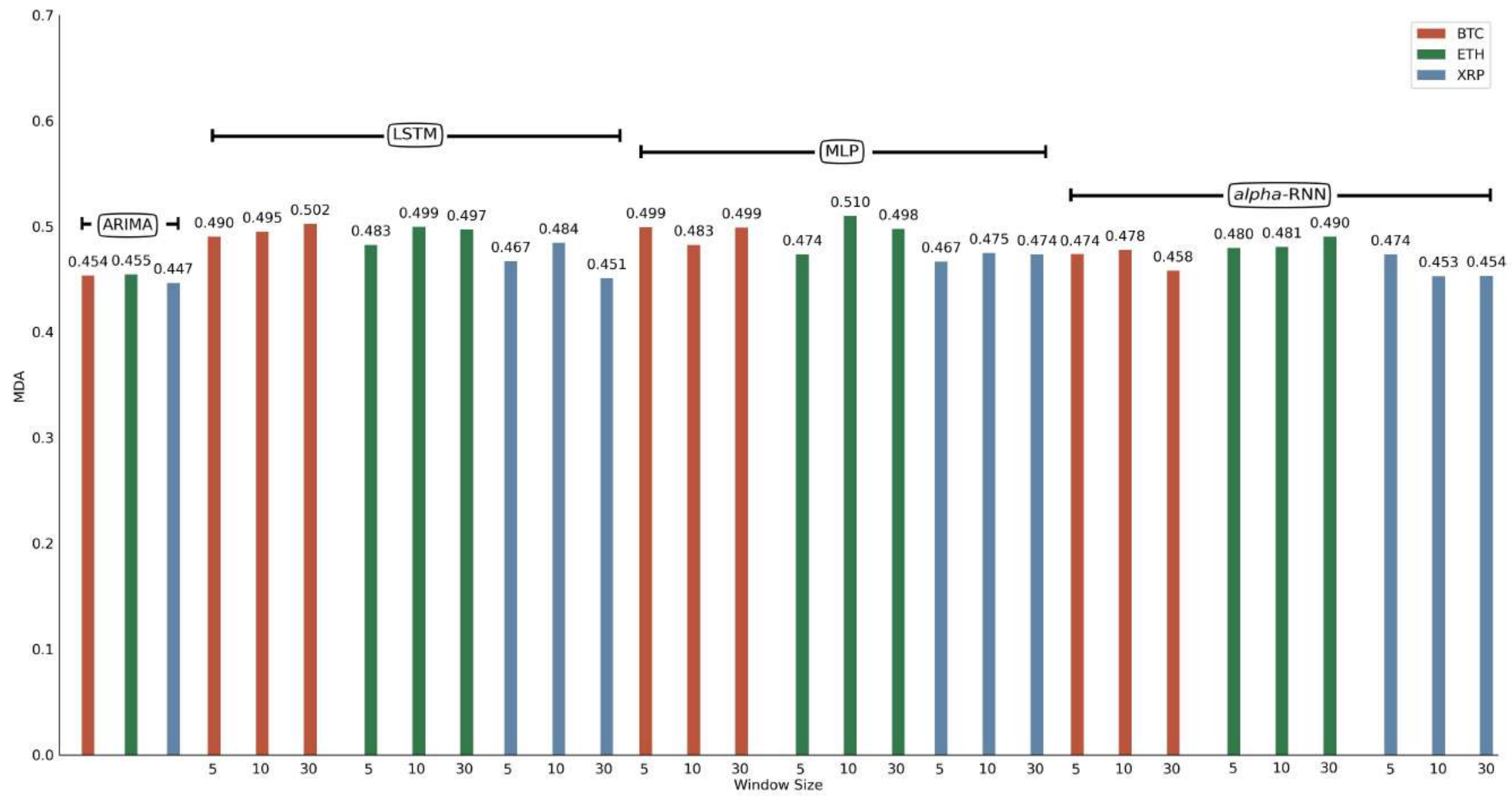

Figure 4.4: Comparison of Model Performance using MDA metric (excluding the predictions in 2021)

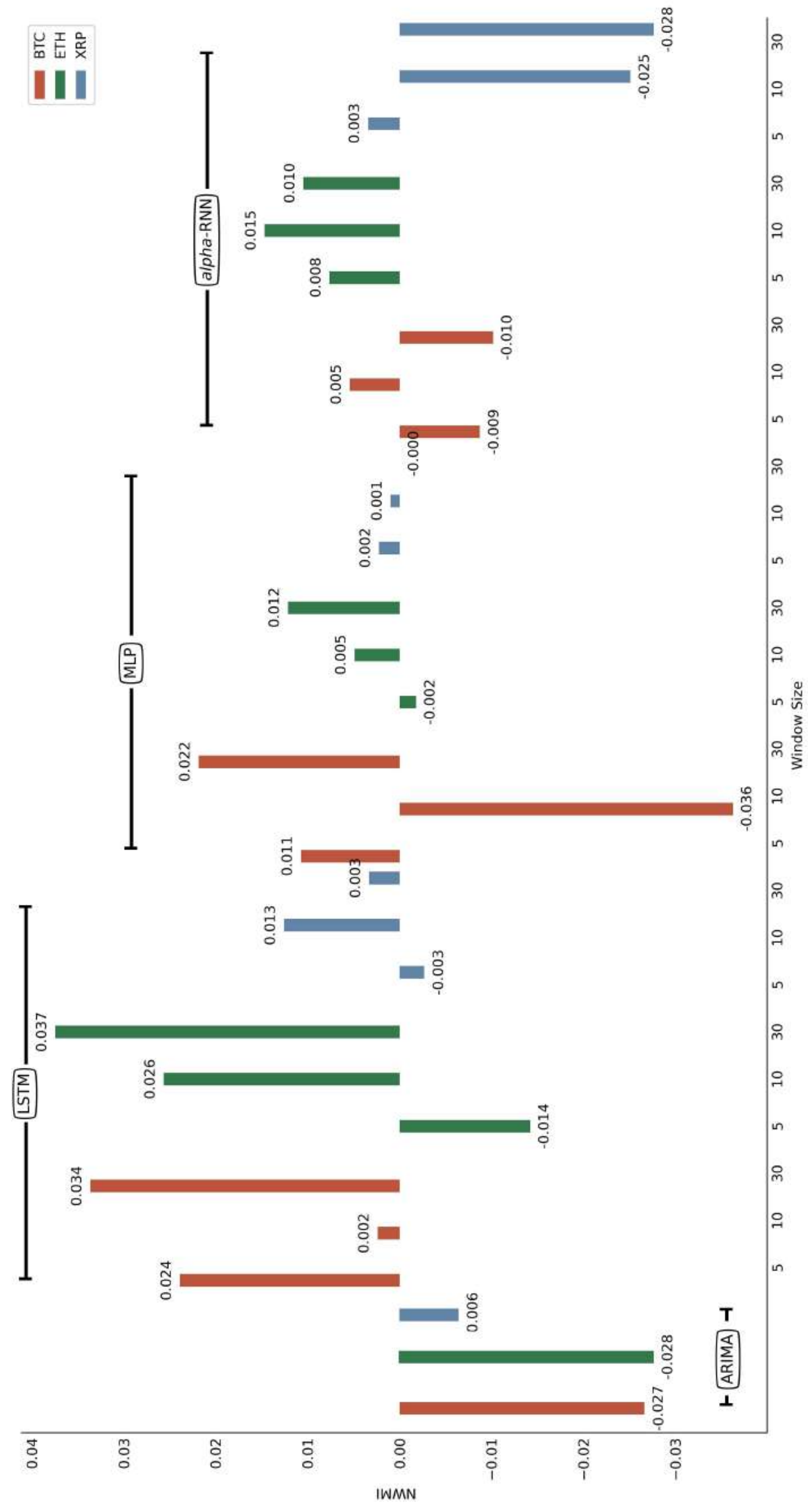

Figure 4.5: Comparison of Model Performance using NWMI metric (excluding the predictions in 2021)

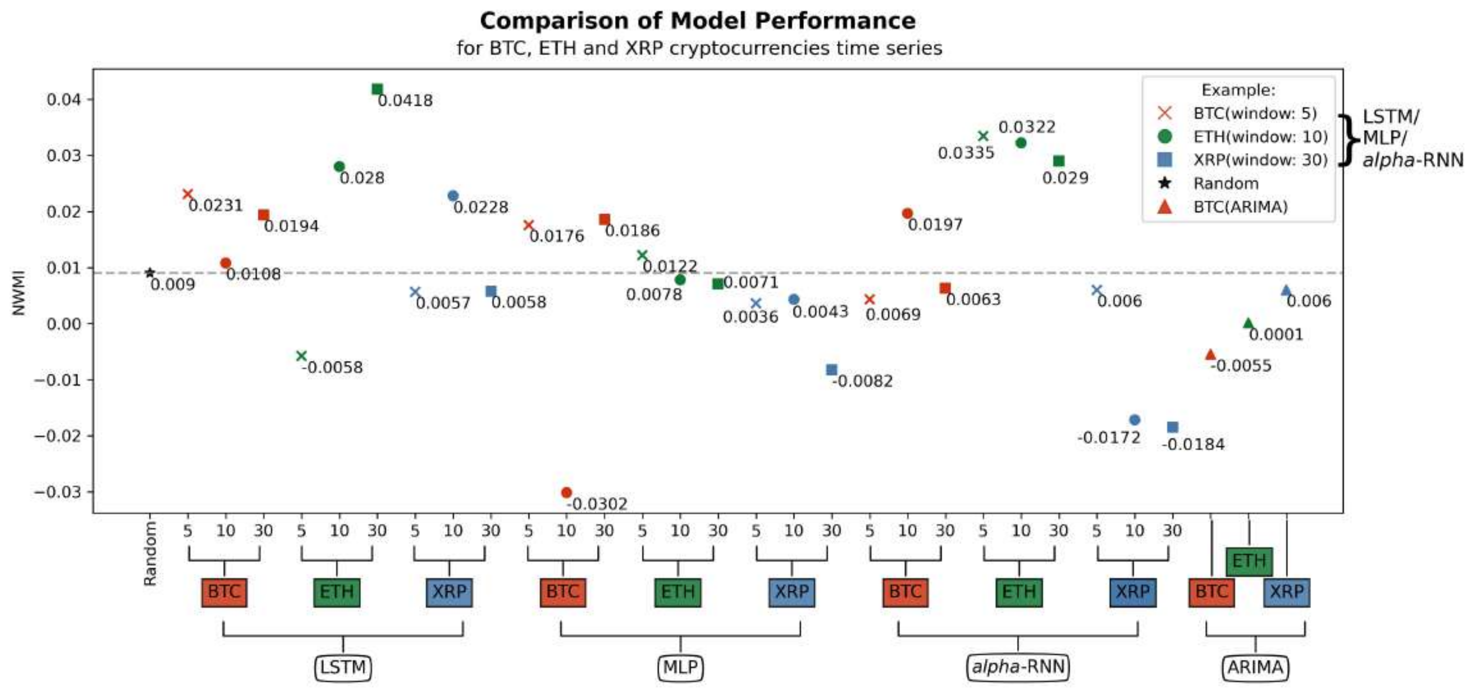

Figure 4.6: Comparison of Model Performance using NWMI score using all predictions.
(Grey dashed line represents the expected NWMI score obtained as described in Section 3.5.6.1)

4.2 FFD Preprocessing

In this subsection, I present the results obtained using the fractional differentiation technique described in Section 3.3.2. As mentioned earlier, the XRP series is not considered in the study of the effectiveness of FFD preprocessing because the optimal differentiation order, as studied in section 3.3.2.2,is zero. Similar to the analysis in the previous subsection, the in-sample and out-of-sample predictions of each model are plotted against actual prices and compared side by side as shown in Figure 4.7. It can be seen that the predictions of all models are less accurate than predictions obtained using standard preprocessing method (refer plot 4.1).

It is also noticeable that the Spearman's rank correlation coefficient as shown in Figure 4.8 also decreases for all models. From the correlation plots, it can be seen that the predictions obtained with the FFD preprocessing approach are worse than the predictions obtained with the standard preprocessing approach (refer to 4.2). Note that the price scale in the 4.7 and 4.8 graphs differs from the two graphs in the standard processing section, as the fractionally differentiated BTC and ETH series are not restored to the original scale.

The results of the evaluation metrics as tabulated in Table 4.10 and 4.11 and are consistent with the observations made previously. Using the context-independent metrics such as R^2 and MAPE, we can see that the model performance has decreased in terms of accuracy. However, the predictions contain less rare errors compared to the results obtained with the standard preprocessing method since the difference between RMSE score and MAE score is not as big as the former cases. Similar to the previous results, ARIMA models have the best accuracy, but not all directional accuracy metrics agree.

As can be seen from the results of the MDA and NWMI values, the overall performance in terms of directional accuracy has decreased significantly for all models. With the FFD preprocessing method, all models correctly predict direction less than half of the time, which is worse than the performance obtained with the standard preprocessing method. The same applies to the capacity of the models to predict large movements, as expressed in the NWMI value.

Similar to the previous analysis, the comparison of performance in terms of MAPE, MDA and NWMI is performed using three corresponding bar charts 4.9, 4.10 and 4.11. The predictions for 2021 are also excluded because α-RNN has large errors for this period. A comparison is also made using all predictions with NWMI score, as shown in Figure 4.12. The performance of all models with FFD preprocessing is worse, except for the α-RNN model with MAPE score for ETH series as shown in Figure 4.9. Comparing the all metric values of the results obtained with the standard preprocessing method, I can conclude that the use of fractional differencing for BTC and ETH series does not improve the model performance.

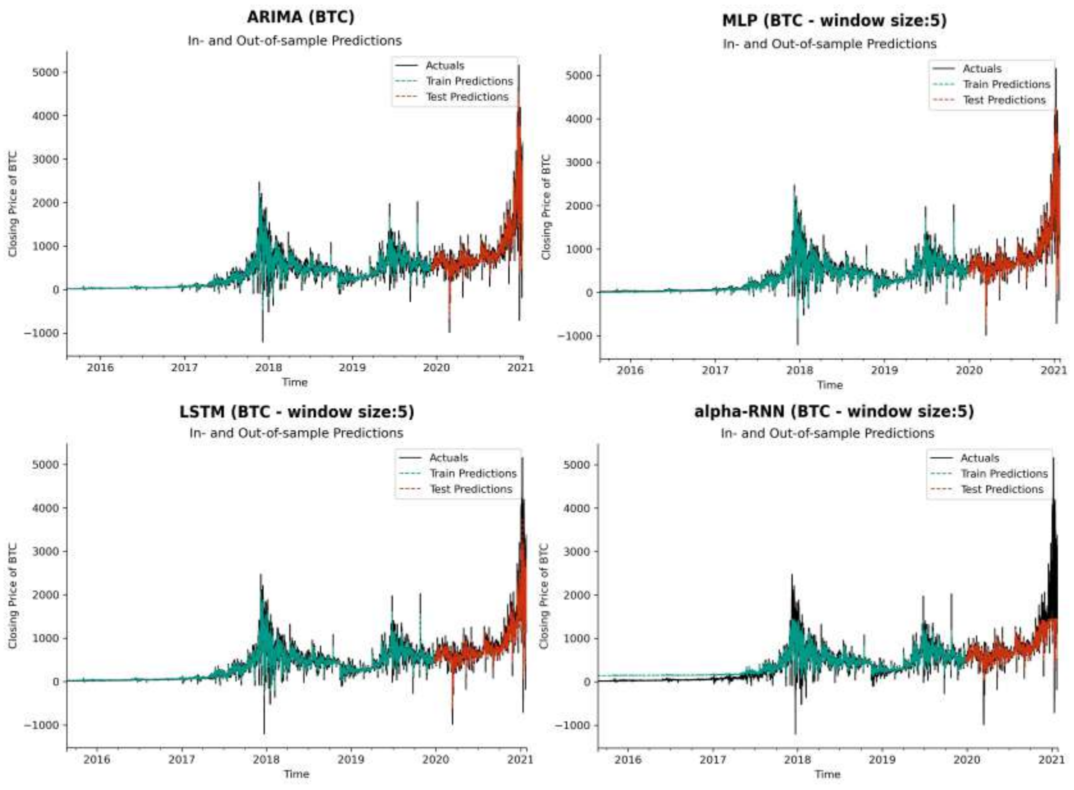

Figure 4.7: In-sample and out-of-sample forecasts of ARIMA(2,0,3), MLP, LSTM and α-RNN (with window size of 5) for BTC Series

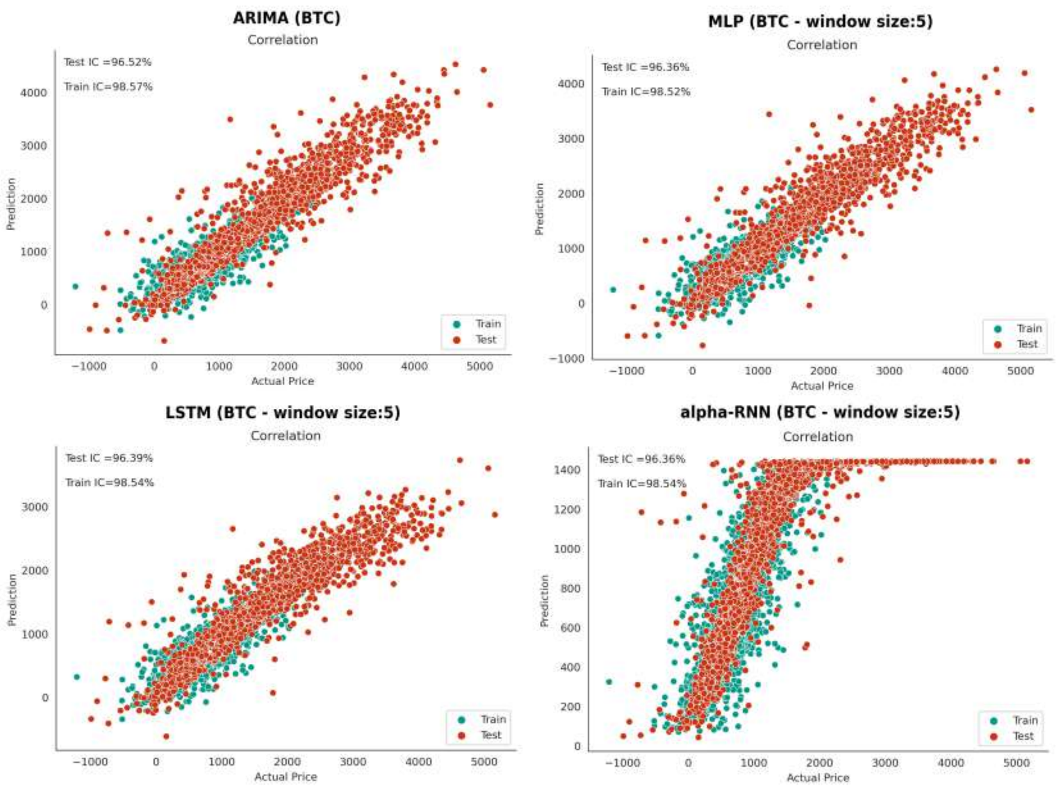

Figure 4.8: Correlations of actual prices and predictions of ARIMA(2,0,3), MLP, LSTM and α-RNN (with window size of 5) for BTC Series

Model Performance for BTC:

Model		RMSE	R^2	MAE	MAPE	MDA	NWMI
LSTM	Train Error	69.132	0.9474	9.0806	20.8081	0.3632	
	Test Error	200.570	0.8889	39.7155	11.0163	0.3743	**-0.0054**
MLP	Train Error	69.698	0.9466	15.2002	26.5620	0.3721	
	Test Error	152.134	0.9361	31.8345	10.0478	0.3754	-0.0188
α-RNN	Train Error	99.224	0.8917	68.4376	149.5970	0.3701	
	Test Error	387.983	0.5841	31.3910	12.3634	**0.3767**	-6.3734
ARIMA	Train Error	67.178	0.9504	6.6414	13.8616	0.3719	
	Test Error	**147.585**	**0.9398**	**27.1755**	**8.7261**	0.3742	-0.0838

Table 4.10: LSTM, MLP, α-RNN with window size = 5 and ARIMA(2,0,3)

Model Performance for ETH:

Model		RMSE	R^2	MAE	MAPE	MDA	NWMI
LSTM	Train Error	4.9053	0.9780	1.4202	198.6080	0.3838	
	Test Error	6.2362	0.9746	1.1468	6.9221	0.3832	**-0.0950**
MLP	Train Error	5.1514	0.9758	0.9695	131.3170	0.3904	
	Test Error	6.8175	0.9696	1.4785	7.3295	0.3884	-0.1199
α-RNN	Train Error	6.2188	0.9647	3.6335	709.1610	0.3777	
	Test Error	8.2899	0.9551	1.3677	7.2806	0.3792	-0.1350
ARIMA	Train Error	4.7572	0.9793	0.3375	8.9213	0.3939	
	Test Error	**6.0631**	**0.9760**	**1.0838**	**6.4088**	**0.3926**	-0.1461

Table 4.11: LSTM, MLP, α-RNN with window size = 5 and ARIMA(2,0,3)

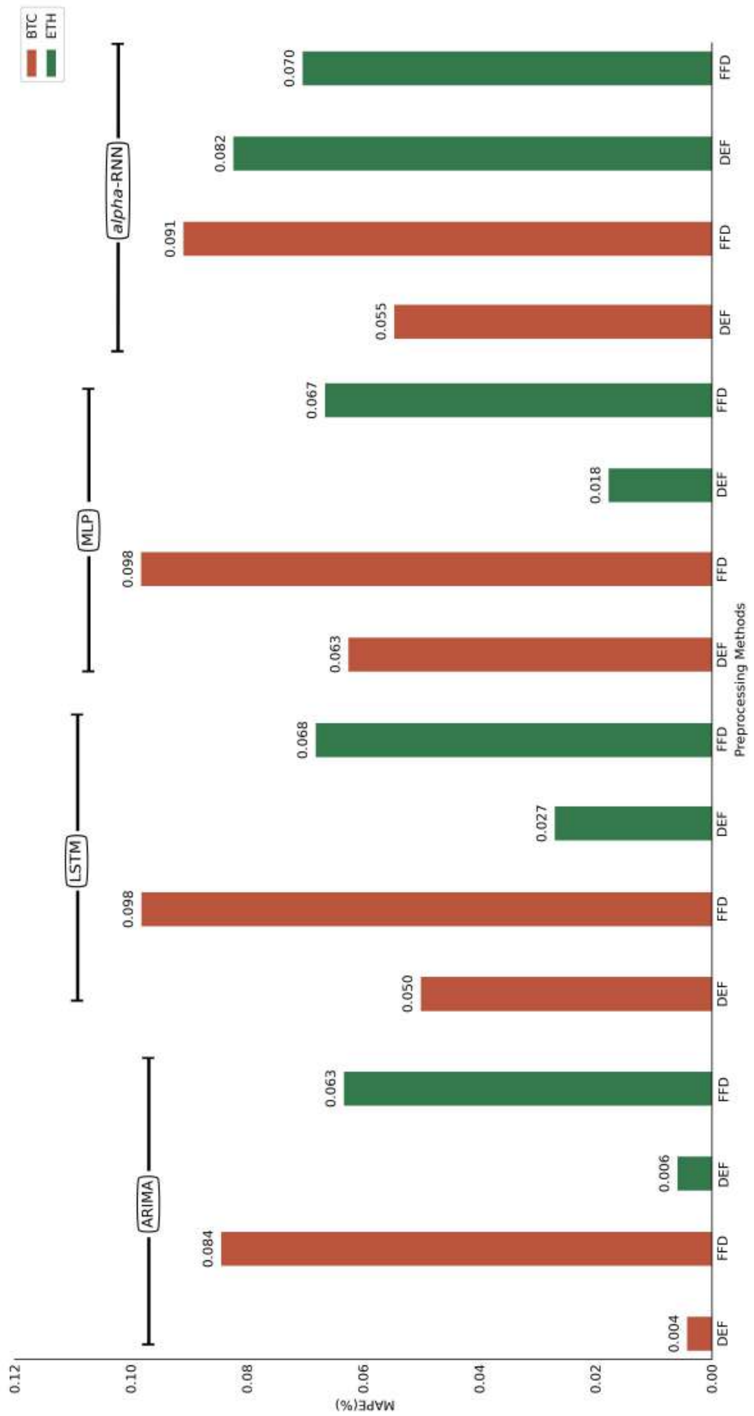

Figure 4.9: Comparison of Model Performance by different preprocessing methods using MAPE metric (excluding the predictions in 2021)

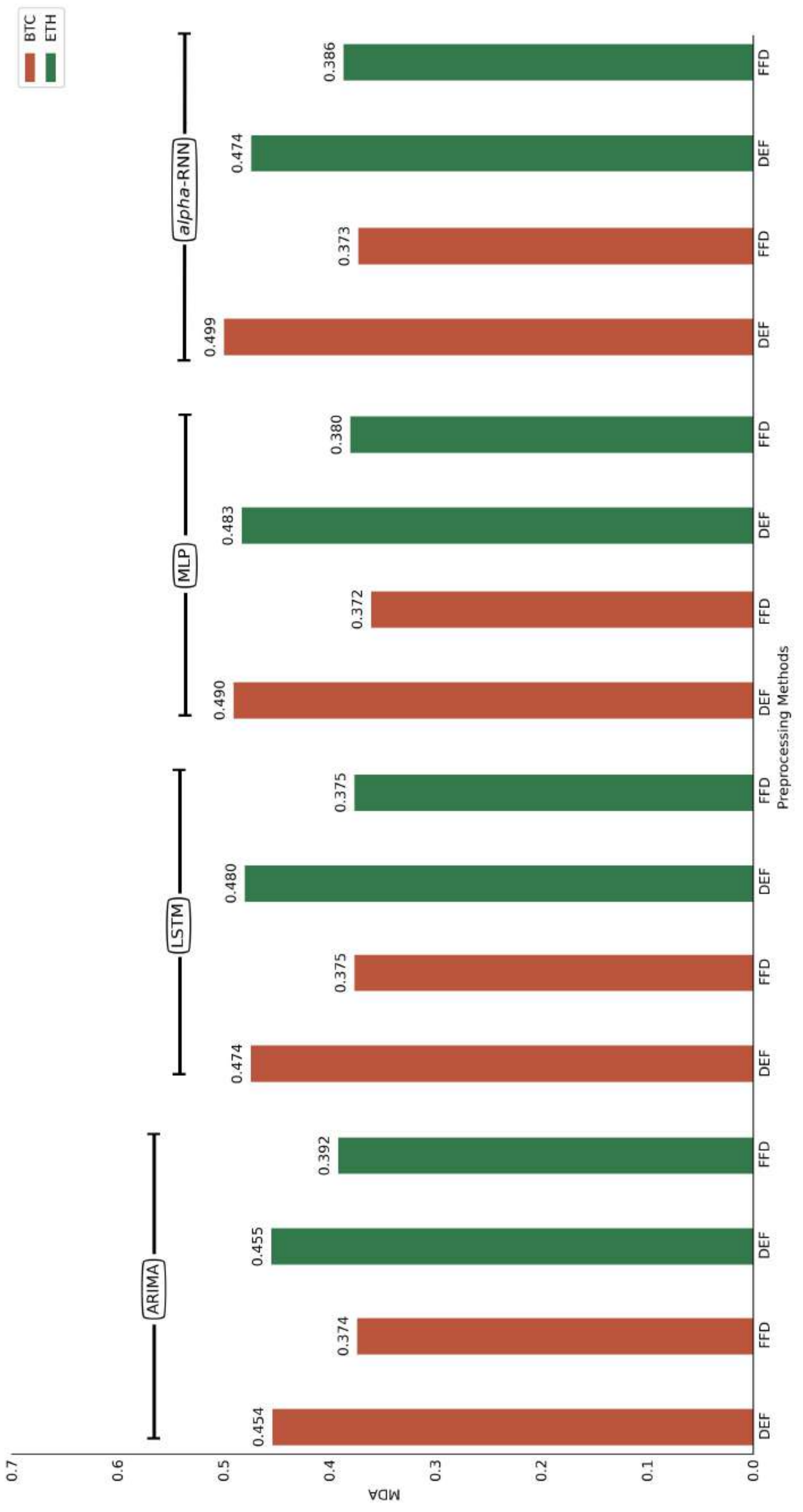

Figure 4.10: Comparison of Model Performance by different preprocesing methods using MDA metric (excluding the predictions in 2021)

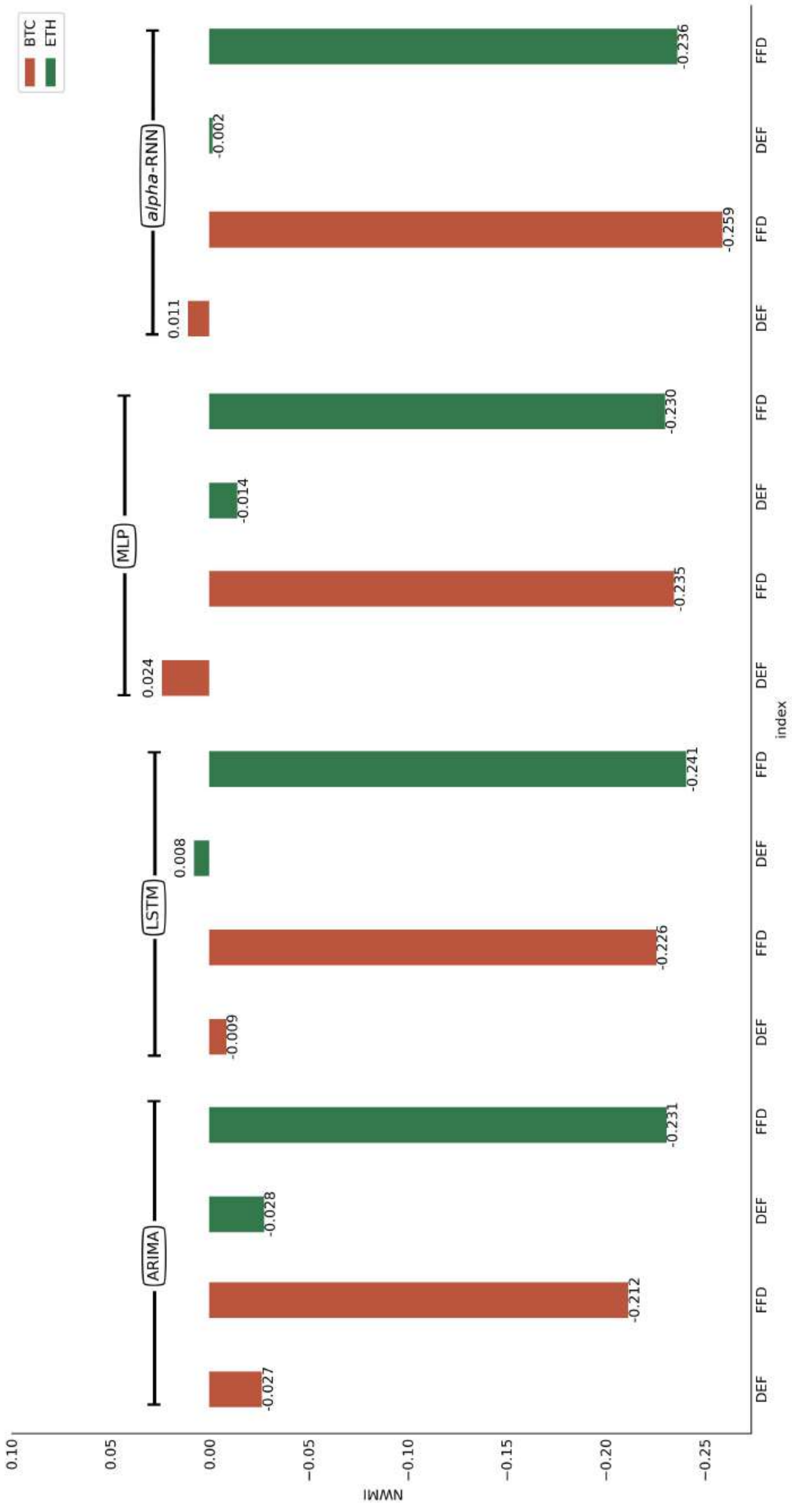

Figure 4.11: Comparison of Model Performance by different preprocesing methods using NWMI metric (excluding the predictions in 2021)

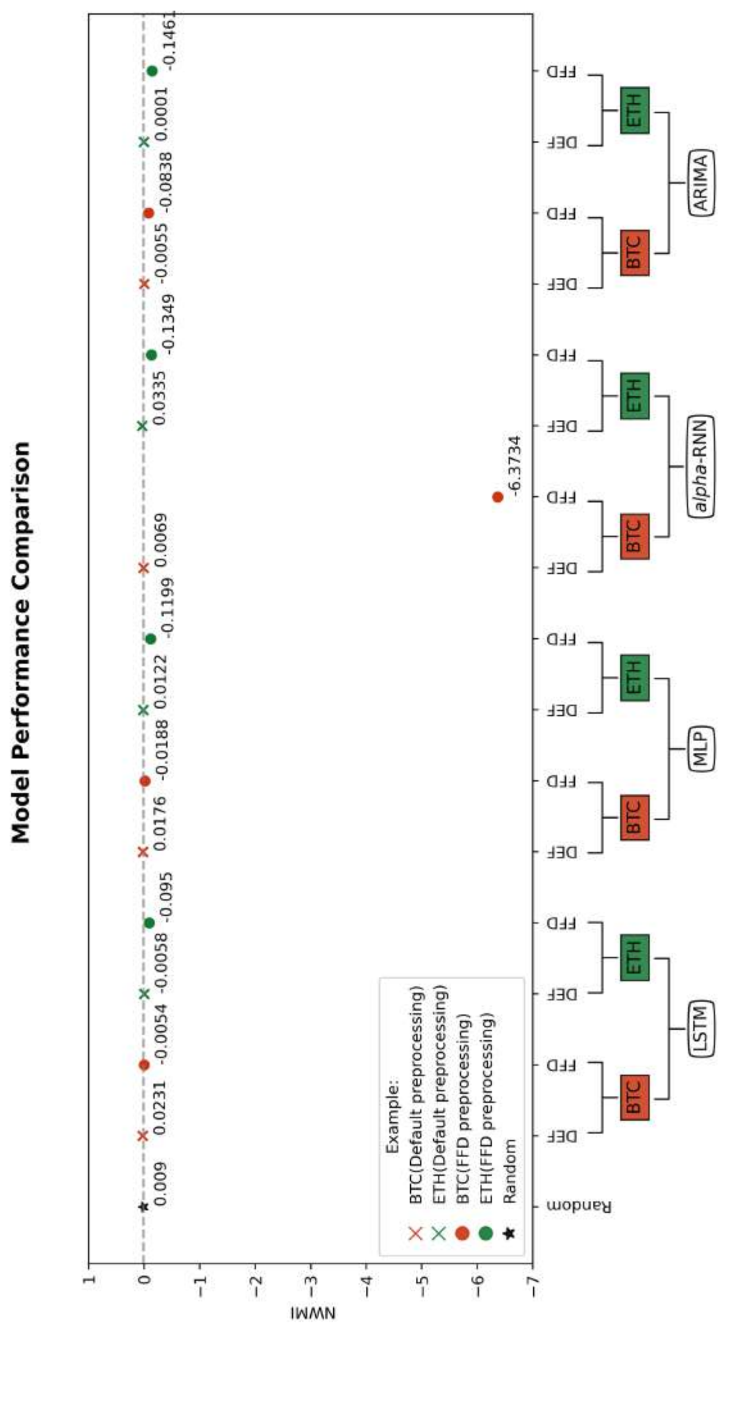

Figure 4.12: Comparison of Model Performance by different preprocessing methods using NWMI score using all predictions. (Grey dashed line represents the expected NWMI score obtained as described in Section 3.5.6.1)

5. Conclusion

This work presents a comparative study of four different forecasting methods with varying degrees of model complexity for predicting the price of three cryptocurrencies for a one hour horizon. The main objective of this study is to investigate whether more advanced models outperform a well-established statistical method. With the typical preprocessing method for each model, the ARIMA model provides better accuracy than MLP, LSTM and α-RNN models in predicting the actual price, but it does not provide the same level of accuracy in predicting the price movement.

Instead, the LSTM model provides better directional accuracy. LSTM models with different window sizes for BTC, ETH and XRP are consistently better than the random model in most cases, which implies that this model is potentially more robust and reliable than the others in terms of predicting the price movement. On the contrary, MLP models perform worst in predicting both actual price and price movement. This result is consistent with the findings of Makridakis et al. (2018) that ML algorithms have little to offer in time series forecasting. On the other hand, there are small differences observed in the performance of LSTM and α-RNN in terms of accuracy in predicting the actual price as concluded by Dixon and London (2020)

The performances of the advanced models are also compared under different experimental conditions. In particular, I would like to determine the best value for the window of consecutive samples from the data to be considered in order to obtain the most accurate predictions. However, there is no clear winner as the performance of models with different window sizes varies under different combinations of experimental settings. On the contrary, neither the performance of the statistical method nor that of the more advanced methods is improved by the fractionally differentiated input series. The decrease in accuracy in predicting the price movement is much larger than the decrease in accuracy in predicting the actual price, with the ARIMA model being somewhat less affected. In summary, although the LSTM model does not outperform the ARIMA model in predicting the actual price, this model seems to learn the historical movement and trend and therefore predict the next movement better than the purely statistical method. Finally, the MLP model provides the worst estimate of both future price and future price movement.

5.1 Future Work

This thesis can easily be extended in several directions. First, it would be interesting to compare the model performance at different multiple forecast horizons. Second, we may consider including other models such as ensemble models of machine learning or deep learning method in the comparative study to make a more general statement about model accuracy in predicting cryptocurrency prices. In addition, we can also consider other cryptocurrencies such as Monero, Dash, Litecoin, and so on, as in the work of Akyildirim et al. (2021). Lastly, the comparative study can focus on the multivariate analysis of cryptocurrencies. One can consider statistical methods such as VAR models that include multiple cryptocurrency price series and compare their performance with the advanced models that use multiple cryptocurrencies as input proposed by **??**. As for the validation method of cryptocurrency price prediction models, the directional metric proposed in this paper is only limited to the use case of the one-step prediction. The formulation of this simple metric is not suitable for multiple forecast horizons. Therefore, we still need a more general version of the directional assessment metric to assess the cryptocurrency price prediction models.

Appendix

A.1 Results of Models using Standard Preprocessing

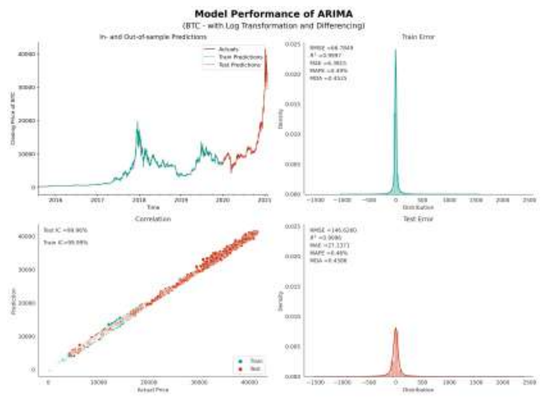

Figure A.1: ARIMA(1,1,2)

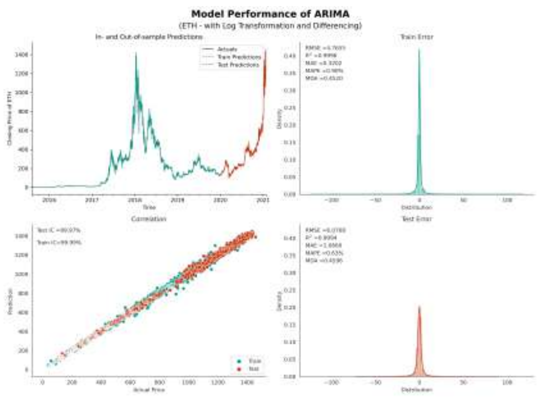

Figure A.2: ARIMA(0,1,3)

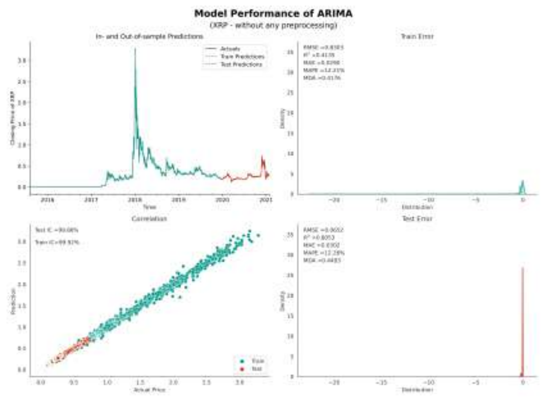

Figure A.3: ARIMA(3,0,3)

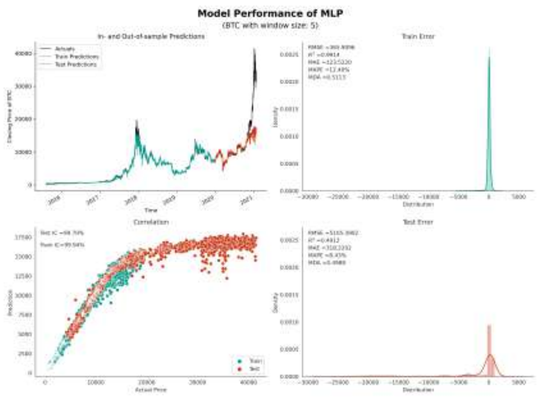

Figure A.4: BTC-MLP:window 5

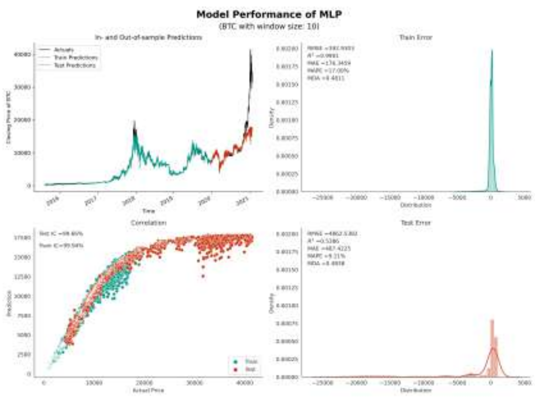
Figure A.5: BTC-MLP:window 10

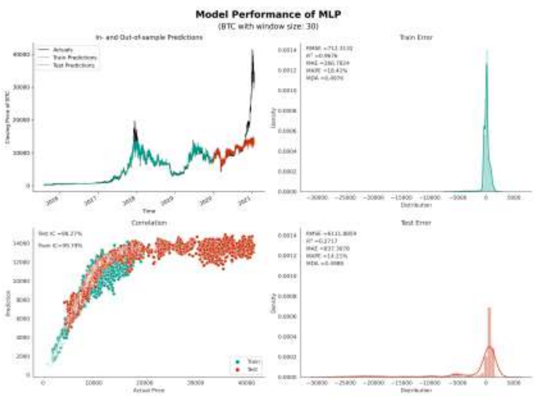
Figure A.6: BTC-MLP:window 30

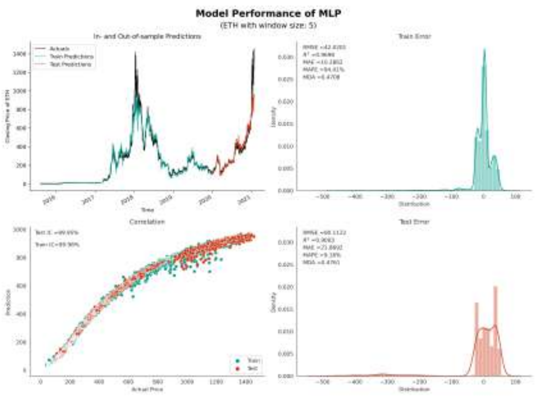
Figure A.7: ETH-MLP:window 5

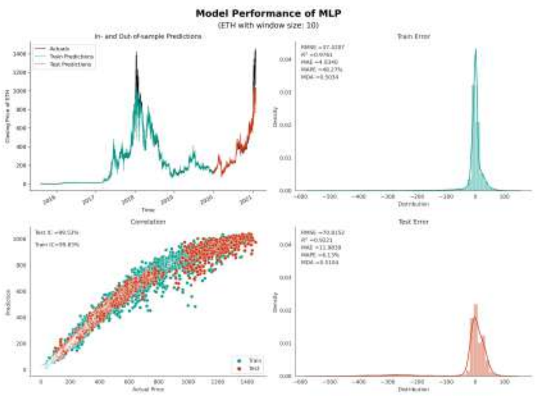
Figure A.8: ETH-MLP:window 10

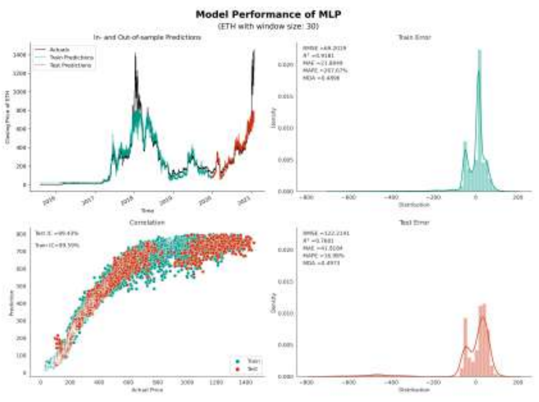
Figure A.9: ETH-MLP:window 30

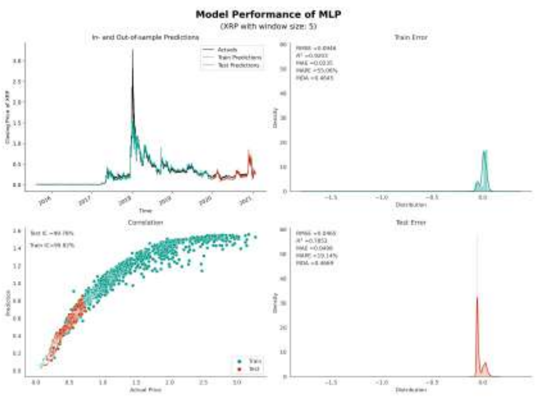
Figure A.10: XRP-MLP:window 5

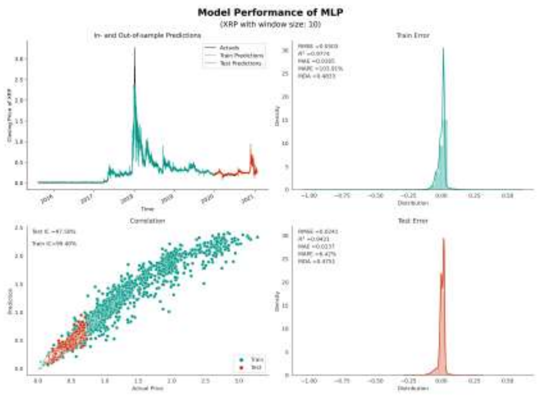
Figure A.11: XRP-MLP:window 10

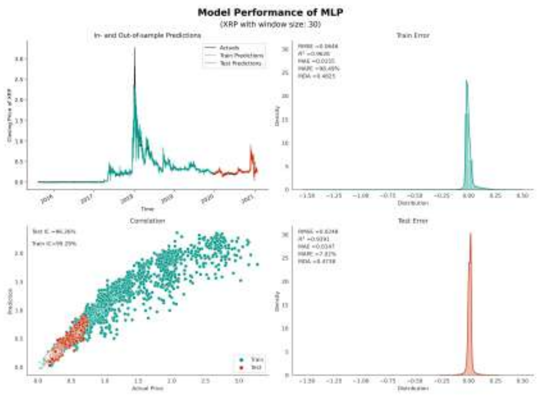
Figure A.12: XRP-MLP:window 30

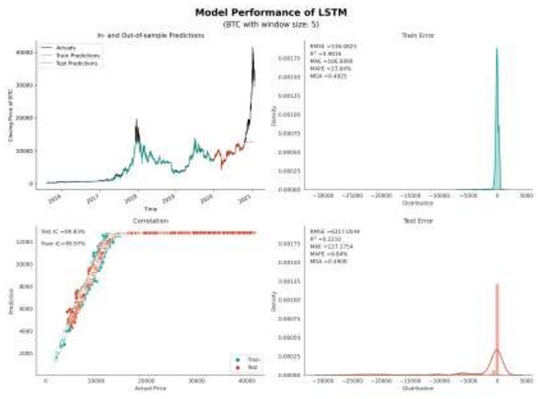
Figure A.13: BTC-LSTM:window 5

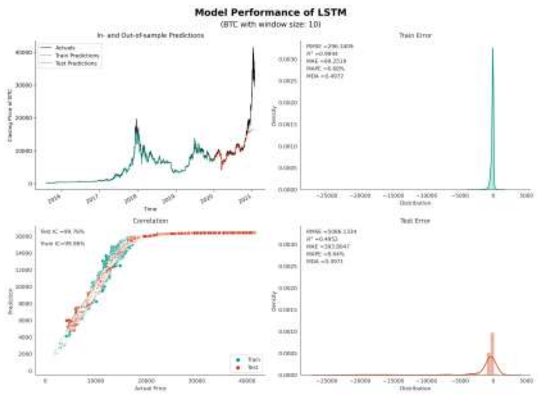
Figure A.14: BTC-LSTM:window 10

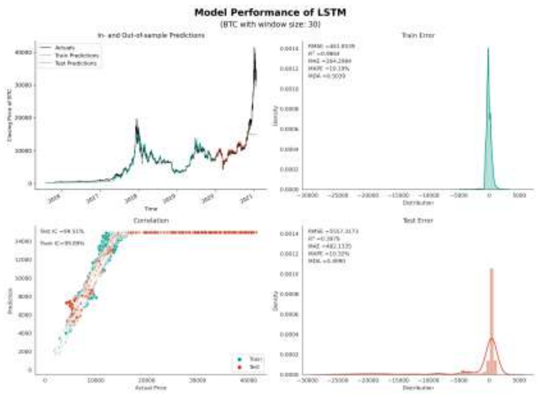
Figure A.15: BTC-LSTM:window 30

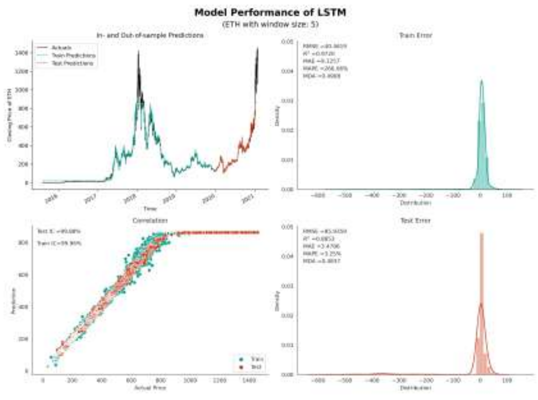
Figure A.16: ETH-LSTM:window 5

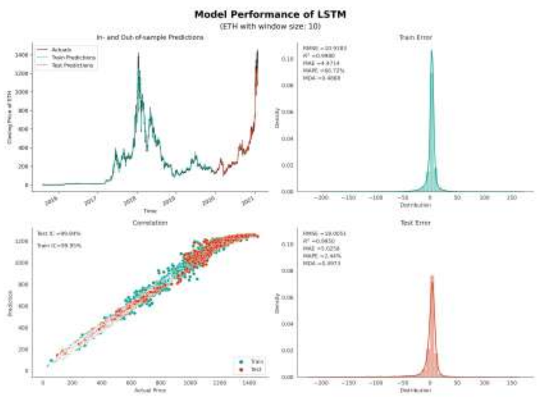

Figure A.17: ETH-LSTM:window 10

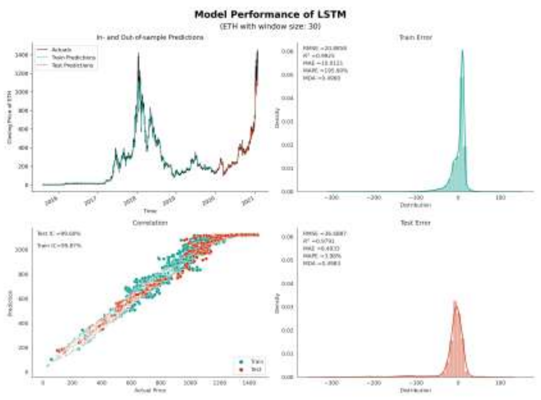

Figure A.18: ETH-LSTM:window 30

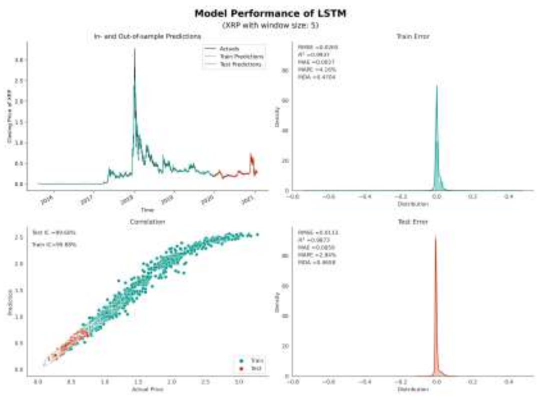

Figure A.19: XRP-LSTM:window 5

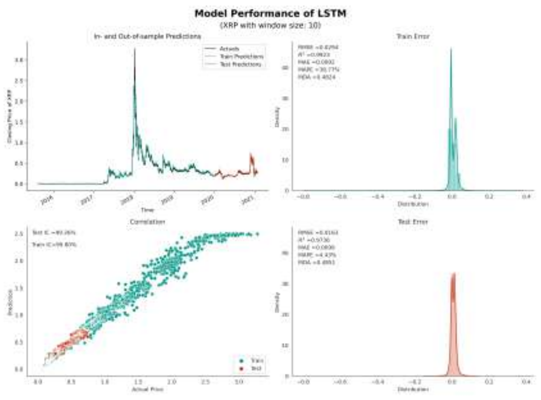

Figure A.20: XRP-LSTM:window 10

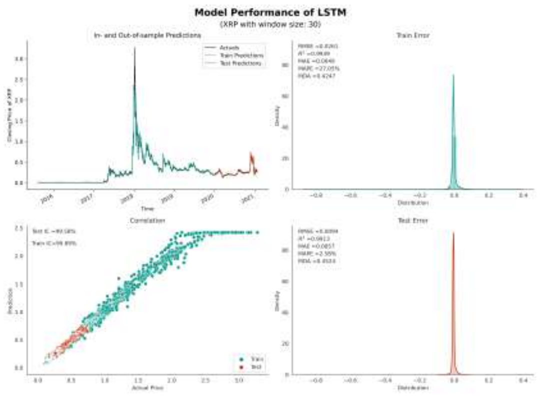

Figure A.21: XRP-LSTM:window 30

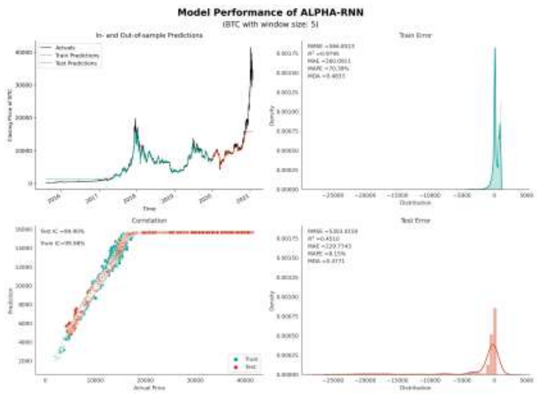

Figure A.22: BTC-RNN:window 5

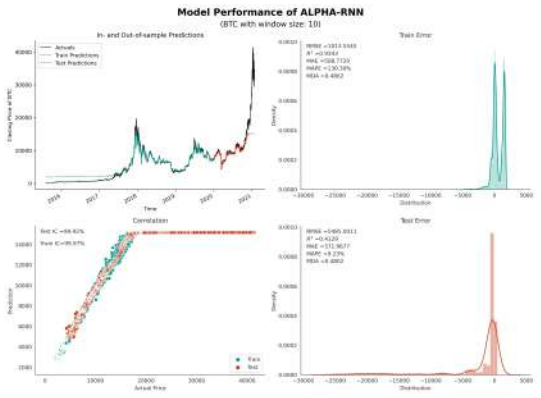

Figure A.23: BTC-α-RNN:window 10

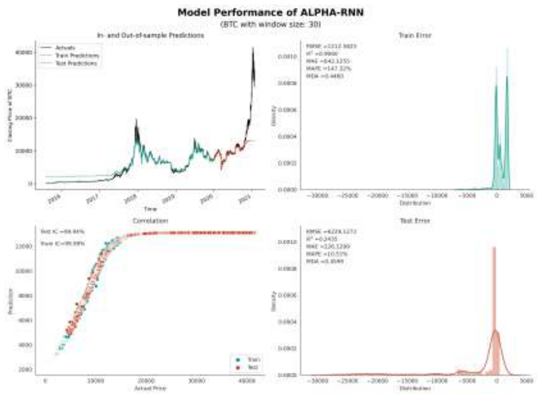

Figure A.24: BTC-α-RNN:window 30

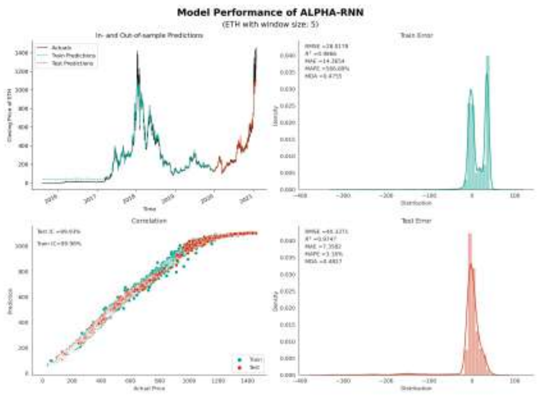

Figure A.25: ETH-α-RNN:window 5

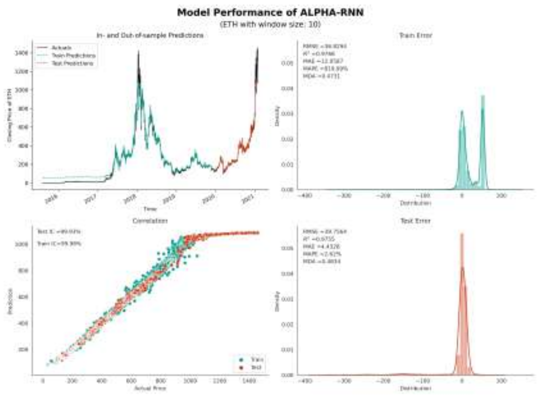

Figure A.26: ETH-α-RNN:window 10

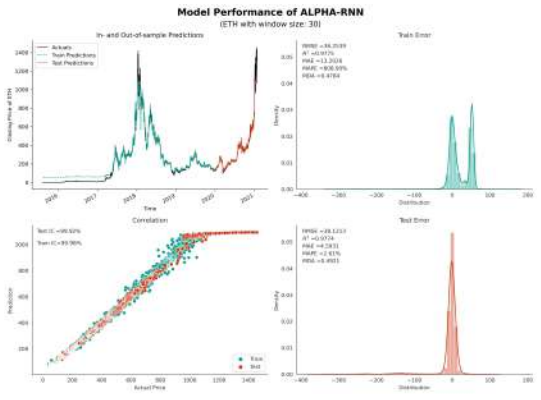

Figure A.27: ETH-α-RNN:window 30

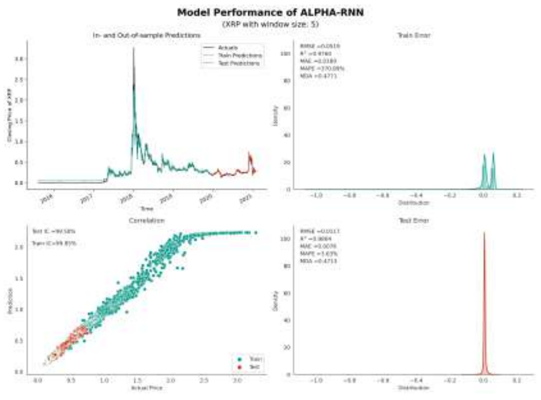

Figure A.28: XRP-α-RNN:window 5

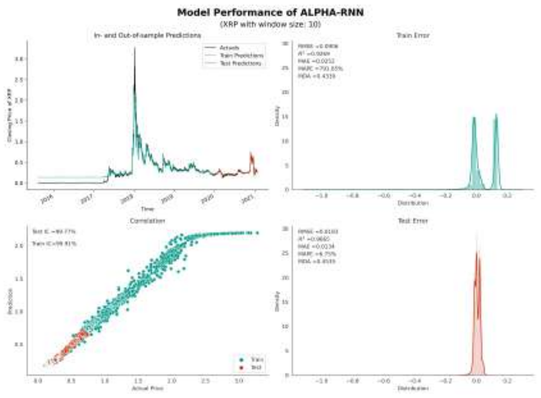

Figure A.29: XRP-α-RNN:window 10

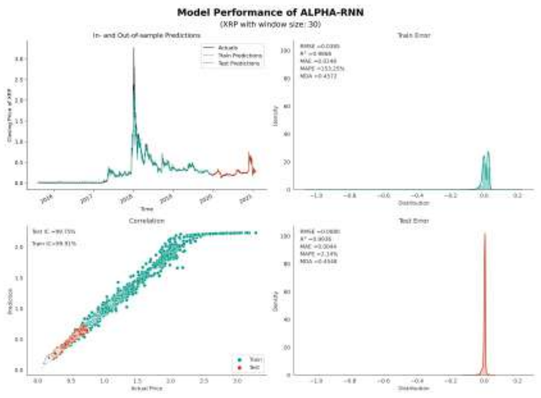

Figure A.30: XRP-α-RNN:window 30

A.2 Results of Models using Fractional Differentiation Preprocessing

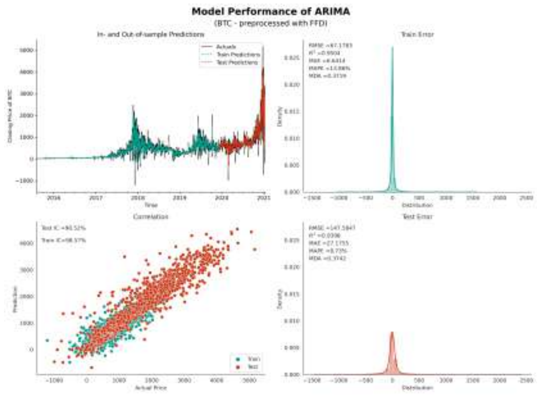

Figure A.31: ARIMA(2,0,3)

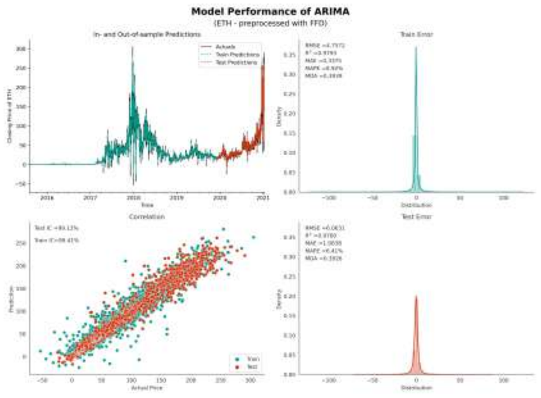

Figure A.32: ARIMA(2,0,3)

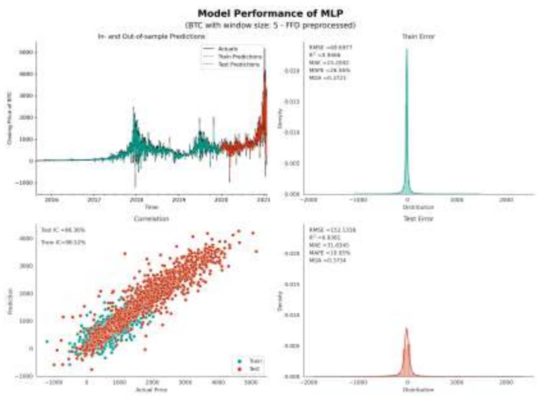

Figure A.33: BTC-MLP:window 5

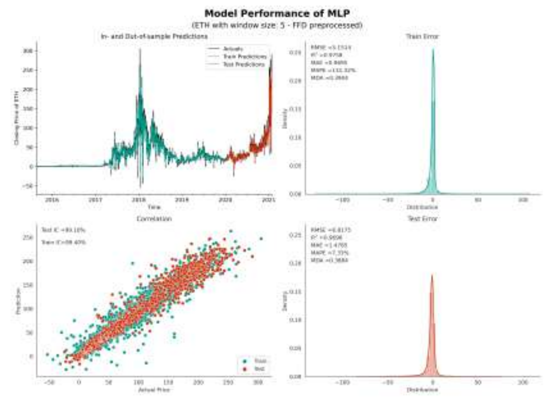

Figure A.34: ETH-MLP:window 5

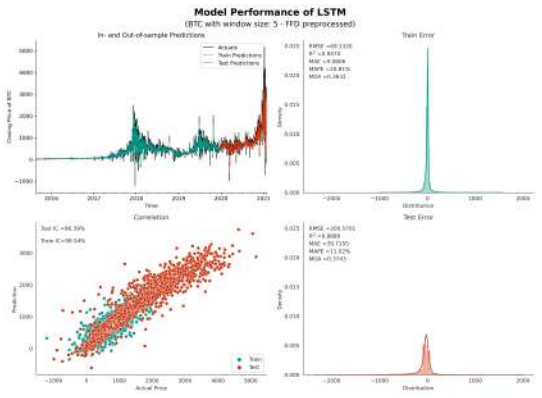

Figure A.35: BTC-LSTM:window 5

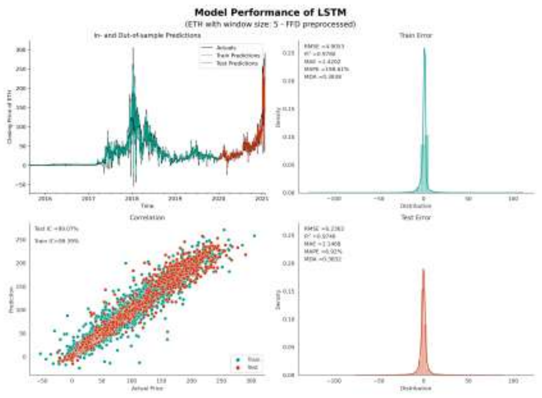

Figure A.36: ETH-LSTM:window 5

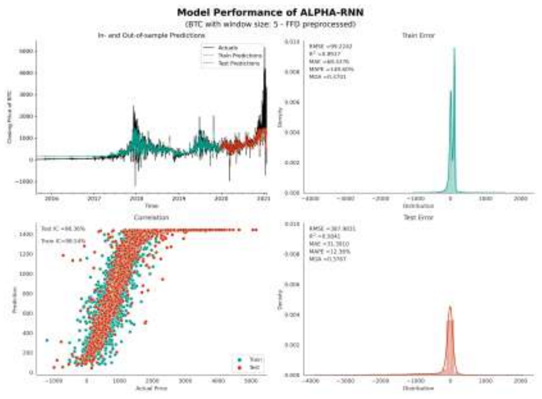

Figure A.37: BTC-α-RNN:window 5

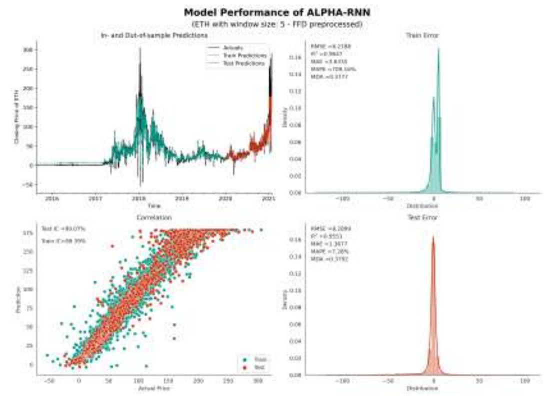

Figure A.38: ETH-α-RNN:window 5

Bibliography

Adebiyi, A. A., Adewumi, A. O., and Ayo, C. K. (2014). Comparison of arima and artificial neural networks models for stock price prediction. *Journal of Applied Mathematics*, 2014.

Akyildirim, E., Goncu, A., and Sensoy, A. (2021). Prediction of cryptocurrency returns using machine learning. *Annals of Operations Research*, 297(1):3–36.

Ariyo, A. A., Adewumi, A. O., and Ayo, C. K. (2014). Stock price prediction using the arima model. In *2014 UKSim-AMSS 16th International Conference on Computer Modelling and Simulation*, pages 106–112. IEEE.

Bagshaw, R. (2020). Top 10 cryptocurrencies by market capitalisation. https://finance.yahoo.com/news/top-10-cryptocurrencies-market-capitalisation-160046487.html. Last accessed on 2021-07-26.

Bergmeir, C., Hyndman, R. J., and Koo, B. (2018). A note on the validity of cross-validation for evaluating autoregressive time series prediction. *Computational Statistics & Data Analysis*, 120:70–83.

Bergstra, J., Yamins, D., and Cox, D. (2013). Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *International conference on machine learning*, pages 115–123. PMLR.

Chaim, P. and Laurini, M. P. (2019). Is bitcoin a bubble? *Physica A: Statistical Mechanics and its Applications*, 517:222–232.

Chen, Z., Li, C., and Sun, W. (2020). Bitcoin price prediction using machine learning: An approach to sample dimension engineering. *Journal of Computational and Applied Mathematics*, 365:112395.

Chollet, F. (2017). *Deep learning with Python*. Simon and Schuster.

Corbet, S., Lucey, B., and Yarovaya, L. (2018). Datestamping the bitcoin and ethereum bubbles. *Finance Research Letters*, 26:81–88.

De Prado, M. L. (2018). *Advances in financial machine learning*. John Wiley & Sons.

Desev, K., Kabaivanov, S., and Desevn, D. (2019). Forecasting cryptocurrency markets through the use of time series models. *Business and Economic Horizons (BEH)*, 15(1232-2020-345):242–253.

Ding, J., Tarokh, V., and Yang, Y. (2018). Model selection techniques: An overview. *IEEE Signal Processing Magazine*, 35(6):16–34.

Dixon, M. F. and London, J. (2020). Financial forecasting with alpha-rnns: A time series modeling approach. *Frontiers in Applied Mathematics and Statistics*, 6:59.

Härdle, W. K., Harvey, C. R., and Reule, R. C. (2020). Understanding cryptocurrencies.

Hyndman, R. J. and Athanasopoulos, G. (2018). *Forecasting: principles and practice.* OTexts.

Jansen, S. (2020). *Machine Learning for Algorithmic Trading: Predictive models to extract signals from market and alternative data for systematic trading strategies with Python.* Packt Publishing Ltd.

Ji, S., Kim, J., and Im, H. (2019). A comparative study of bitcoin price prediction using deep learning. *Mathematics*, 7(10):898.

Livieris, I. E., Kiriakidou, N., Stavroyiannis, S., and Pintelas, P. (2021). An advanced cnn-lstm model for cryptocurrency forecasting. *Electronics*, 10(3):287.

Makridakis, S., Spiliotis, E., and Assimakopoulos, V. (2018). Statistical and machine learning forecasting methods: Concerns and ways forward. *PloS one*, 13(3):e0194889.

Makridakis, S., Spiliotis, E., and Assimakopoulos, V. (2020). The m4 competition: 100,000 time series and 61 forecasting methods. *International Journal of Forecasting*, 36(1):54–74.

Malladi, R. K. and Dheeriya, P. L. (2021). Time series analysis of cryptocurrency returns and volatilities. *Journal of Economics and Finance*, 45(1):75–94.

Osborn, D. R., Chui, A., Smith, J., and Birchenhall, C. (1988). Seasonality and the order of integration for consumption. *Oxford Bulletin of Economics and Statistics*, 50(4):361–377.

Pintelas, E., Livieris, I. E., Stavroyiannis, S., Kotsilieris, T., and Pintelas, P. (2020). Investigating the problem of cryptocurrency price prediction: a deep learning approach. In *IFIP International Conference on Artificial Intelligence Applications and Innovations*, pages 99–110. Springer.

Zheng, A. (2015). *Evaluating machine learning models: a beginner's guide to key concepts and pitfalls.* O'Reilly Media.