University of Amsterdam

Masters Thesis

The stability of stablecoins: Are these coins stable or is the stability just a flip of the coin?

Author:
Kwan Sing Lie

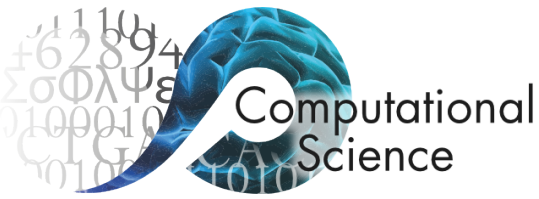
Examiner:
Dr. Eric Pauwels
Supervisor:
Dr. Ronald Heijmans
Assessor:
Prof.dr. R.D. van der Mei

A thesis submitted in partial fulfilment of the requirements for the degree of Master of Science in Computational Science

in the

Computational Science Lab
Informatics Institute

January 2025

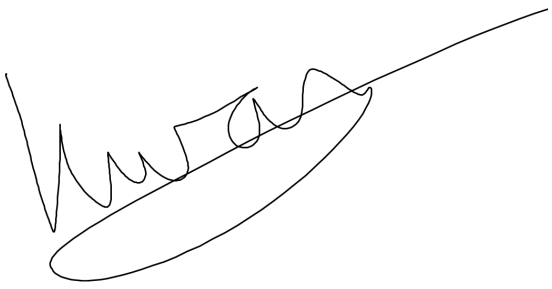


Declaration of Authorship

I, Kwan Sing Lie, declare that this thesis, entitled 'The stability of stablecoins: Are these coins stable or is the stability just a flip of the coin?' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at the University of Amsterdam.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:



Date: January 2025

"What I cannot create, I do not understand."

Richard P. Feynman

UNIVERSITY OF AMSTERDAM

Abstract

Faculty of Science
Informatics Institute

Master of Science in Computational Science

The stability of stablecoins: Are these coins stable or is the stability just a flip of the coin?

by Kwan Sing Lie

This paper focuses on the volatility of the stablecoins USDT and USDC, utilizing both econometric models (ARCH and GARCH) and machine learning models (LSTM and GRU). Both models are optimized for processed daily data of USDT and USDC. Machine learning models perform to be performing better than traditional econometric models. However, these models seem to be very unstable, producing different results with the same inputs. Additionally, features are added within the dataset and some exploratory data analysis was conducted to investigate which factors influence the volatility. The exploratory data analysis includes heatmaps, principal component analysis, permutation feature importance, and the drop-column method. Due to the low movement of stablecoins, identifying correlated features remains quite difficult. Furthermore, the permutation feature importance and drop-column methods, which utilize machine learning models to assess feature correlations, were unable to produce consistent results. This lack of alignment between the two techniques highlights the difficulty in determining which features are truly correlated with the stablecoins, making it difficult to identify significant features that are correlated to stablecoins.

Acknowledgements

I would like to thank 'De Nederlandsche Bank' for being able to do my graduation project at such a prestigious company. Throughout my internship, I have been given the freedom to focus on my thesis while also being actively involved in meetings and workshops. This has provided me with valuable hands-on experience and a genuine insight into the professional work environment.

I want to thank Ronald Heijmans supervising me throughout my thesis. Our weekly meetings allowed me to maintain steady progress and receive insightful feedback, helping me keep my work focused and concise. Also thanks to the Payments Canada for providing us with the data used for this project. With our bi-weekly meetings, I have been given the unique opportunity to collaborate with researchers from diverse areas of expertise.

Finally, I am especially grateful to my other colleagues at the 'Marktinfrastructure and Innovation' department, whose kindness and support made my internship an extremely enjoyable experience. Even with my short termed position as intern, I feel like I have been truly included as part of the team, both inside and outside of work. The past 7 months have been super learnful to me and I will definitely carry the lessons I learned with me into my future career.

Contents

List of Figures

List of Tables

Abbreviations

CSL	**C**omputational **S**cience **L**ab
UvA	**U**niversiteit **v**an **A**msterdam
BTC	**B**i**TC**oin
ARCH	**A**uto **R**egressive **C**onditional **H**eteroskedasticity
GARCH	**Generalized A**uto **R**egressive **C**onditional **H**eteroskedasticity
PACF	**P**artial **A**uto **C**orrelation **F**unction
LSTM	**L**ong **S**hort **T**erm **M**emory
GRU	**G**ated **R**ecurrent **U**nit **M**emory

Chapter 1

Introduction

Cryptocurrencies are digital or virtual currencies that make use of cryptography for secure and decentralized transactions. Cryptocurrencies operate on decentralized networks based on blockchain technology. They have grown in popularity over the last decade. Not only has its popularity risen, alongside with it has its price. In the past 5 years, Bitcoin's value has risen by more than 1500% and Ethereum's by more than 2400%, with a total market cap of cryptocurrency of 2.52 trillion dollars at the time of writing. However, cryptocurrencies hold a large volatility due to many factors such as market sentiment, lack of regulation, thin liquidity and many more. In addition to that, even price differences can be found between crypto exchanges, which leads to arbitrage opportunities [3]. Both, the high volatility and price differences, have attracted traders looking for (arbitrage) profits.

Originally, Bitcoin was invented as a purely peer-to-peer version of electronic cash that would allow online payments to be sent directly from one party to another without going through a financial institution like a bank or government. This would result in much faster payments along with being quite secure because of the use of blockchain. Although the use of Bitcoin or other cryptocurrencies might sound useful, due to their extreme price volatility, banks and government are still hesitant to use this for its practical use.

To overcome this issue, stablecoins have been introduced to the market. As the name suggest, stablecoins attempt to provide a stable value relative to other crypto assets by pegging their value to a real-world asset or by using algorithms that keep the supply of coins proper to its valuation. This is done with the aim of providing an alternative to the high volatility in order to make them more suitable as a medium of exchange.

However, it turned out to be that some stable coins were not as stable as we thought. In May 2022, the stablecoin TerraClassicUSD broke its peg with USD and in a matter of weeks the coins lost nearly 90% of its value.

The use of stablecoins would significantly speed up the process of transactions, since the transfer time of stablecoins between two accounts is almost instantaneously. One could buy stablecoins, send them to someone, for example to the USA, and that person could change those stablecoins back to their own currency USD. Governments and banks are increasingly becoming interested in the use and functionalities of stablecoins; however, they also come with a lot of complications. From a theoretical perspective, since payments do not go through a third party, governments lose control of transactions between users, due to the anonymity of each transaction.

And from a practical standpoint, central banks currently ensure the value of the fiat currency that you are using. If, for example, a bank defaults, then a central bank is still able to give you a certain amount of money back that you would have lost. However, when many people are going to use stablecoins as a form of payment, central banks would lose control and would be unable to ensure the value of money to the user. If a stablecoin suddenly lost its value significantly or even worse, would default, it would have a major impact on the value of many other payments and cryptocurrencies [4]. Therefore, the research to its volatility is essential before the use is widely implemented.

The focus of this paper is to investigate the volatility of stablecoins, in particular the two largest stablecoins in circulation at the time of writing: USDT and USDC. Given the fact that the price should remain stable, the analysis will focus on the stability by examining the volatility. Both econometric and machine learning models are implemented to attempt to predict the volatility, as well as exploratory data analysis to identify which factors like other cryptocurrencies and indices that may influence the stability of these stablecoins. Additionally, the impact of these factors on the performance of machine learning models is evaluated.

This paper begins with presenting related work and provides background on the key topics about cryptocurrencies. Then, the dataset used in the research is described in detail. The theory behind the models employed are discussed. Subsequently, the results are presented, followed by an seperate discussion section. And the paper concludes with a brief summary, and a small final section which addresses the ethical considerations associated with the research.

Chapter 2

Literature review and Background

2.1 Related work

Volatility of cryptocurrencies Given the rise of cryptocurrencies and their influence on the financial market, in the early stages, most of the research has been done on the volatilities of cryptocurrency, which have proven to be quite difficult [5] [6]. In an empirical study [7], it is found that behavior of cryptocurrencies have heavy tails, meaning large price jumps are more common than normal stock markets. The correlation between past and future returns fades quickly. The volatility tends to cluster where high volatility tend to follow one other, and there seems to be a power law correlation between price and volume. In addition, the price of cryptocurrencies can be treated as asset bubbles, where the price can rise rapidly, but also fall rapidly. All these characteristics make cryptocurrency price forecasting challenging and investments in cryptocurrencies much more risky than traditional financial assets [8] [9].

Econometric approaches to crypty price forecasting The majority of studies on bitcoin price forecasting has used traditional methodologies. Catania et al. used a number of univariate and multivariate vector autoregression (VAR) models for predicting four major cryptocurrencies: Bitcoin, Ripple, Litecoin and Ethereum. Their reportings were that there were significant improvements in forecasting accuracy with the use of combinations of various univariate forecasting models. Many investigations using traditional econometric models such as Conrad et al., which uses GARCH-MIDAS, have found a highly significant correlation between S%P 500 and Bitcoins volatility. In another research where GARCH-MIDAS is applied [6], which researched the prediction

of the volatilities of Bitcoin, Ethereum, Litecoin, Ripple and Stellar, the effect of Global Real Economic Activity is a major driver of long-term volatility of cryptocurrency.

Applying machine learning to crypto price forecasting In recent years, the focus of cryptocurrency price forecasting has shifted to machine learning methods. Zahid et al. made use of the combination of econometric GARCH and machine learning models such as LSTM, GRU, and BiLSTM algorithms with single, double, or triple layer architectures to forecast Bitcoin's realized price volatility and has been shown to generate accurate results. In particular, GRU has shown excellent forecasting performance for four major cryptocurrency prices [13] [14], outperforming not only traditional methods but also LSTM-based models.

All these quantitative researches have mostly been done on large cryptocurrencies, however very little if not any research have been done quantitatively on stablecoins, only some general research/papers about stablecoins [15] or a data overview of stablecoins [16]. Likely because of the assumption that the value itself is stable. The price stability of a stablecoin is highly important; therefore, it is also necessary to conduct quantitative research on this topic, which is also the motivation behind our study.

2.2 Blockchain

The main component that enables the use of cryptocurrencies is the blockchain. Blockchain is a shared, immutable ledger that facilitates the process of recording transactions and tracking assets in a business network. As a transaction between two parties happen, the data about this transaction gets stored within a ledger. This data can the record the information of choice: who, what, when, where, how much etc.

A common use case, in order to add security to a transaction, is the use of a public and secret key. Both keys represent a string of bits with only 1s or 0s. The difference is that the secret key should only be available for the owner, whereas a public key is openly known, hence the name. For each transaction, a digital signature is needed to confirm the transaction. This signature is generated by taking the message (the information of the transaction) and private key.

Note that when the message is even altered slightly, the signature changes along with it. So the secret key ensures that only the user can verify the transaction and the message ensures that the transaction can not be altered. After a signature is created, another verification step is needed, which takes the message, signature and public key, to check whether this signature was produced by the private key.

Each transaction tracked on a ledger, a collection of transactions, and as more transactions happen, the ledger gets larger. Once the ledger becomes a certain size, it stops growing in size and a hash value is calculated using the contents of the ledger, just like a signature is created with a transaction. A hash value is created using a cryptographic hash function. A cryptographic hash function has 3 important features:

- They are preimage resistant. It is computationally infeasible to compute the correct input given some output value. e.g. given y, find x such that hash(x) = y.
- They are second preimage resistant. It is computationally infeasible to find a second input which produces the same output. e.g. given x, find y such that hash(x) = hash(y).
- They are collision resistant. It is computationally infeasible to find any two inputs that produce the same output. e.g. find an x and y which hash(x) = hash(y).

A commonly used cryptographic hash function in many blockchain implementations, like bitcoin [17], is the Secure Hash Algorithm with an output size of 256 bits (SHA-256) and the idea behind such function is that it is should be completely unfeasible to find a valid signature if you do not know the secret key.

Once a hash value is established, the ledger with all its values, which we can consider a block, will be added directly after the last recent constructed block. Blocks are chained together through each block containing a created hash value of the previous block, therefore forming a blockchain. If a previously published block has been altered, it would also have a different hash value. Consequently, it would cause all subsequent blocks to have different hashes, since they also include the hash of the previous block. Depending on the chosen technology, the rules of adding a block to the blockchain can differ, for example permissions, priority, or extra computational calculations.

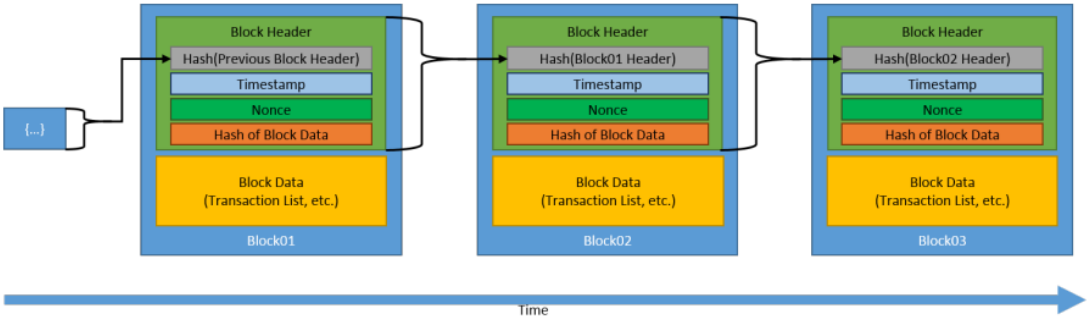


FIGURE 2.1: Generic Chain of Blocks [1]

Decentralization plays a crucial role for cryptocurrencies for the security of the system. Unlike traditional currencies, cryptocurrencies are not controlled by any central authority like a government or institution. Instead, they rely on a distributed network of nodes (computers) to validate and record transactions of the blockchain. This means that instead of storing a blockchain on one server, many copies of the blockchain exist that operate simultaneously. Whenever a transaction happens or a block is added to the blockchain, this occurrence is broadcasted to other copies of the blockchain. With the use of the various copies, we can validate a specific blockchain. For example, when some blockchain is altered by someone, we can check the validity of this blockchain by comparing it with other copies and if this altered block is different from the rest, we can name this copy of blockchain invalid.

There are also various rules surrounding the decision of which blockchain is the correct one. When two blockchains have different lengths, the blockchain with the longest chain will be chosen. Hypothetically, a node with enormous amounts of computing power could start a whole new longer chain than the currently existing chain, thereby wiping the entire blockchain history. However, this does not happen in practice due to the enormous amount of resources this would require. It would take basically 51% of the total computing power, which is in practice highly unlikely. This is also mentioned in the original bitcoin paper. Additionally, some blockchain implementations lock specific older blocks by creating checkpoints to prevent this from happening.

There are many other limitations, methods and intricacies surrounding blockchain, however this section is written to give a concise explanation about the overview of the blockchain and will likely make it easier for the reader to understand conceptually about the project [1].

2.3 Cryptocurrencies

2.3.1 Valuation of fiat currency

Fiat currency is a currency which value is not backed by a certain asset like gold or silver, but rather holds its value by the trust or belief of the people. The biggest problem with fiat currency is that, theoretically, banks and the government can print as much money as they want. The distribution of fiat currency is centralized, meaning it is controlled by one main system or authority. National currencies such as the euro or dollar have not been backed by the gold standard since 1971, which is what many people still believe to this day. The great upside of this is that the government has control of the economy and exchange rate fluctuations, but the downside is that the users/citizens need to pay high

prices, which we call in the form of inflation and reduced purchasing power. For example, $1 in 2000 is equivalent in purchasing power to about $1.80 today, an increase of $0.80 over 24 years. The dollar had an average inflation rate of 2.48% per year between 2000 and today, producing a cumulative price increase of 80.21% [18].

The trust in value of fiat currency plays a very important role and is the reason why we are able to use fiat currency. The goal of the government is to ensure the stability of the price of fiat currency and prevent any big fluctuations. This trust is based mainly on the following three things:

- It is legally set that everyone needs to accept this means of payment.
- Fiat currency is acknowledged as a tool to pay taxes.
- The government ensures that the rate of exchange remains relatively stable.

So, it comes down to the fact that the government holds a monopoly on fiat currency. They decide that it needs to be accepted and that when and how much money can be added to circulation. This leads to trust, however, some are against this idea and this one of the underlying reasons of the existence of cryptocurrencies [19].

The biggest difference between fiat currencies and cryptocurrencies is that cryptocurrencies are not controlled by a central government or bank, they are decentralized by using the blockchain and usually have a fixed total supply to avoid devaluation through inflation.

2.3.2 Valuation of cryptocurrencies

In order to understand the valuation of cryptocurrencies, it is essential to understand the concept of liquidity pools. When buying or selling cryptocurrencies, the transaction is a bit different compared to the traditional stock market. The traditional stock market uses an order book model, where buyers and sellers write down and submit their orders of how much of a stock they want to buy and at what price they want to buy it. Whenever a buyer and a seller meet at the same price, a trade is transacted, where the buyer gets the stock and the seller receives the money. However, this does mean that whenever you want to sell a stock, you would need to put the stock at a price which people want to buy it, which can be time-consuming especially when you want to sell your stock immediately.

Liquidity pools remove the inconvenience of the dependence for someone else to buy your asset, and allows a user to always buy or sell an asset no matter the price and time

of the day [20]. These transactions within a liquidity pool are processed using smart contracts. Smart contracts are self-executing contracts with the terms of agreement directly written into code. They automatically enforce and execute the terms of the contract when predefined conditions are met; therefore, there is no need for an issuer or custodian to fulfill the terms [21].

A liquidity pool starts off a 50:50 ratio of two coins, for example 50% Bitcoin and 50% USDT. So if someone wants to start a liquidity pool with a value of $1000, then the pool starts with $500 worth of BTC and %500 worth of USDT. Most liquidity pools make use of an algorithm called Automated Market Maker (AMM). An AMM allows traders to buy and sell assets using an algorithm that dictates how expensive something should be based on how much of something there is within the liquidity pool. A user is only able to trade coins that are within the current pool they are trading with. In this example, the user can either trade BTC, by putting some BTC in the pool, and receive USDT out of it or vise versa, trading, or rather adding USDT to the pool and receiving BTC from the pool. As one buys one asset, like BTC, then the value rises since less BTC remain within the pool. And since they are giving USDT to the pool for it, that asset becomes cheaper since there is more of it. Basically, supply and demand, but by using an algorithm to calculate the price.

To be specific, the most commonly used formula is called Constant Product Automated Market Maker which uses an inverse formula X * Y = k, where X and Y are the quantities of the assets like BTC and USDT. And k being a constant value. So, if a user were to send X, like BTC, they would receive Y, USDT, from the pool, and since the goal is to keep k constant, the amount of USDT that the user would receive would be calculated by dividing k by the new total supply of BTC and calculating the difference between this value and the quantity of Y of USDT. However, when someone decides to also add the same amount of BTC to the pool, they would receive less USDT in return, since the total supply of BTC within the pool is higher than before and has less value in USDT in return. Hence, contributing to the concept of supply and demand [22].

People also have the option to add liquidity to the liquidity pool by adding assets to the liquidity pool itself. They are adding in this case BTC and USDT to the pool and allowing others to trade with them within the pool. These people are called liquidity providers (LP). Every trade within a liquidity pool takes a small fee; this fee goes back to the liquidity provider proportional to how much the investor has put into the pool compared to the total supply. These returns can go up quite high, even going up to over 200% APR. Gaining APR from investing in liquidity pools is called yield farming. Having a large liquidity pool is important, since if the total volume is large, when an

asset within the liquidity pool is bought, the price of the other asset will not be affected as much.

However, investing in liquidity pools comes with its own risk. Things such as impermanent loss, where an investor would lose money compared to the assets that they would have had, when they would just hold them in their wallet, which happens when other liquidity pools with the same pairs have different values. Or rug pulls, when the developer of a coin runs away with the investor's funds by, for example, taking out all the liquidity within a liquidity pool [23].

Cryptocurrency networks, also known as blockchain networks, form the foundational structure supporting various digital currencies and their transactions. Each network operates independently, adhering to its own set of rules and protocols. This independence often means that cryptocurrencies and tokens are generally confined to their respective networks, making it challenging to trade assets across different blockchain ecosystems.

This means that tokens are only supported within their own ecosystem. So trading two tokens within a Solana network would be no problem; however, when trading across two different networks, for example the Solana network and Binance Smart Chain, you would need a coin that supports both networks that would work as an intermediate trading step between the two tokens.

The price per coin which you can see on various crypto or exchange websites like CryptoCompare, or CoinMarketCap. Although this calculation for the prices is complicated, the prices displayed are basically calculated by weighted averages considering more than 250 exchanges based on their volume, recency of trades, and trust. The calculation for CryptoCompare for example makes use of the Crypto Coin Comparison Aggregated Index (CCCAGG) which refers to the methodology of calculating the real-time index [24]. However, it is important to note that cryptocurrency markets are highly volatile, and determining the exact price of cryptocurrencies can be challenging due to the multitude of factors at play.

Bitcoin was originally created as a store of value. However, since it is not widely adopted and since there are not many regulations on it yet, the price tends to fluctuate a lot to a point where it is considered a speculative investment – or "when an investor hopes to profit from a rapid change in the value of an asset" according to SoFi [25].

These factors include not only the inherent supply and demand dynamics within the crypto ecosystem, but also external influences such as fluctuations in interest rates, movement in the stock market, regulatory developments, and even social media trends. All these different dependencies on factors emphasize the difficulty in estimating the

value in cryptocurrencies and how important it is to have a nuanced understanding of the market dynamics.

2.4 Stablecoins

The objective of stablecoins is to maintain a stable value relative to a specified peg. Stablecoins aim to address the price volatility that other cryptoassets face, which could enhance their appeal as a payment or storage option. This peg can be one asset specifically or a multitude of assets. As such, the issuers of the stablecoin claim that the value of stablecoins can be redeemed at par with the value of the relevant peg. Most stablecoins to date have been fixed to a single asset, most of them typically sovereign currencies, such as the US dollar or euro, but also other commodities are used, such as gold or other crypto assets.

A stablecoin's strategy to maintain its target price is referred to as its 'stabilization mechanism'. Stabilization mechanisms differ according to whether a stablecoin is collateralized by some type of asset or is uncollateralized.

- Off-chain collateralized stablecoins are backed by bank deposits or other cash-like assets traded in the traditional financial system. These stablecoins are called 'off-chain' because traditional assets are not represented by tokens in the blockchain. Until the user redeems the stablecoins, a custodian is required to keep the collateral assets safe. Most commonly, off-chain stablecoins are typically fully collateralized by dollar-denominated assets. Examples include Tether (USDT) and USD Coin (USDC)
- On-chain collateralized stablecoins are backed by assets that can be represented by tokens on a blockchain, so the collateral can be held in smart contracts. On-chain stablecoins are collateralized by crypto assets. Examples include Liquity USD (LUSD) and Dollar on Chain (DoC).
- Uncollateralized stablecoins, also known as algorithmic stablecoins, use a set of rules and strategies to maintain exchange rate stability to dynamically match the supply of the stablecoin with user demand. In contrast to collateralized stablecoins, few or no assets are held in reserve with the aim of supporting the stablecoin's exchange rate. Examples include Terra UST and DAI [26].

The biggest stablecoin by market cap and the most commonly used is USDT, created by the company Tether. Tether tokens are pegged to a fiat currency at a 1-to-1 ratio,

meaning in theory, 1 token equals 1 unit of that currency. Theoretically, if you want 1 USDT, you would pay Tether 1 USD, and they would give you 1 USDT back. They would then hold your dollar forever until you decide to trade back your 1 USDT to 1 USD. In addition to that, they would destroy your purchased USDT token, so the amount of USDT in circulation would remain the same. In this way, the amount of USDT that they have given out is always equal to the USD in their reserve, and you would be able to trade these seemingly equal assets back and forth.

The second largest stablecoin USDC was launched in September 2018 by the Centre consortium, a collaboration between Circle and Coinbase. Just like USDT, the value is pegged with a 1-to-1 ratio to USD. In addition to the difference to the issuer, USDC provides regular transparency reports and is committed to adhering to U.S. money transmission laws, making it one of the most regulated stablecoins in the market. Hence, due to its increased transparency, companies and institutions prefer USDC slightly over USDT. [27] [28]

Although in theory stablecoins should remain stable in price, practical issues do exist which need to be considered as well. For uncollateralized stablecoins, it is a more technical issue, where it is easy to expand the supply in circulation, but decreasing the supply can be more challenging. The issue for collateralized stablecoins is in order for the stablecoin to hold its value. The amount of money in its reserve of the issuer should always enough in case the holders of the coin would want to trade their stablecoin in to back this up. Take for example Tether, where it is not known how much USD they hold in their reserve. They claim that their coin is fully backed in USD, but due to the lack of regulation, this has not been verified by governments or other parties. The value of USDT is very important for the whole crypto market, given the fact that the value of many cryptoassets are linked to USDT. As happened in the bank run in 1930, many people wanted to withdraw their money at the same time, however banks did not have enough cash to back this up, which resulted in huge inflation. People suspect that this could also be happening with Tether. And if the value of USDT drops, many prices of cryptocurrencies will drop along with it.

Chapter 3

Data

3.1 Crypto and stablecoin data

For this project, regarding the crypto and stablecoin data, we made use of the CryptoCompare API. This choice is made because of its comprehensive data offerings, despite the general difficulties in pre-processing data from various providers. Our goal in using CryptoCompare was to obtain a large amount of reliable data to back up our prediction. In the official CryptoCompare API documentation, there are various calls that can be used to obtain data like historical, socials, order book, futures, etc. From a DNB standpoint, we do not have access to the API key since we did not pay for the license. Instead, the data is provided by the Bank of Canada. Additionally, the amount of calls per day is also limited, therefore we can't fully make use of all the data sources the API provides.

For the CryptoCompare API, there are basically two main data sources that are used for training our models. Note that both data sources are on a minute-by-minute basis, meaning that each data point represents information collected or recorded at one-minute intervals. In this research, a minute frequency will not be used since this takes in this would take too much time to process and analyse, however, having this much data, does allow us to have the flexibility to pick whatever interval we want to analyze now and for future research. First, is data called the Crypto Coin Comparison Aggregated Index (CCCAGG). The price conversion per cryptocurrency can differ per exchange, due to the mechanisms of buying/trading cryptocurrencies 2.3.2, so much so that even huge arbitrage opportunities arise if trading across different exchanges instantly was possible [3]. So in order to show the best price estimation for cryptocurrency traders and investors to value their portfolio at any time, the CCCAGG is created that refers to real-time index calculation methodology. CCCAGG is CryptoCompare's own index

calculation technique for digital assets, is based on an outlier approach, a time-penalty factor, and a 24-hour volume weighted average computation. Using a volume-weighted average calculated over a 24-hour period, it compiles transaction data from over 250 exchanges [24]. The dataset for the CCCAGG has the following columns, alongside some description of the value of the data. In the CCCAGG data, the conversion rate is available to many fiat and stable coin currencies, however since in this research stablecoin data is analyzed and that stablecoins are pegged to the dollar, to obtain the valuation of a coin, we will only look at the conversion rate towards USD. This also avoids other factors, such as the exchange rate changes between fiat currencies.

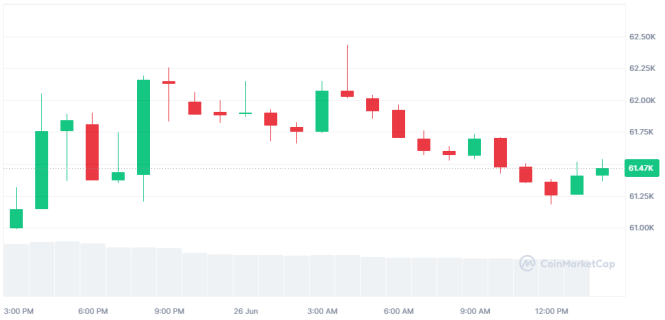


FIGURE 3.1: Example candle data image

The data consists of minute-by-minute candlestick information 3.1, meaning that for each minute, it records the open value, close value, minimum value, and maximum value for each currency pair. Additionally, it includes the volume of the value converted from one currency to another. For the CCCAGG data, the exchange_site value is always set to CCCAGG. Each row has the following columns:

In total, the top 10 biggest cryptocurrencies will be looked at based on CoinMarketCap by the time of writing september 9th 2024. These cryptocurrencies are the following:

In the first column, the volatility of either USDT or USDC is included as target variable. Volatility refers to the degree of variation in the price of a financial asset over time. It measures the extent of how much the price fluctuates, where high volatility means that the price changes considerably in a short period of time, indicating high risk, while lower volatility suggests more stable prices.

The volatility is calculated using the rolling-window method. The current value of volatility is calculated as the standard deviation of returns (percentage change) over a specified number of previous time steps. Since cryptocurrencies tend to be quite unstable, a time window of the past week (7 days) is taken, to analyze the shorter-term price movements.

Column name	Description of value
Exchange time in seconds (float64)	The exchange time converted in seconds, t=0 starts at UNIX time, 1st January 1970 (UTC)
exchange_site (string)	Exchange site where the exchange/trade has occurred
from_currency (string)	The currency that has been traded (sold)
to_currency (string)	The currency that has been converted to (buy)
open_value (float64)	Value of the currency in the bought currency that has been traded at the start of exchange
close_value (float64)	Value of the currency in the bought currency that has been traded at the end of exchange
highest_value (float64)	Highest value within the minute of trading
lowest_value (float64)	Lowest value within the minute of trading
converted_value_from (float64)	Volume of currency that has been traded
converted_value_to (float64)	Volume of currency that has been bought
exchange_time (datetime64)	Timestamp of when the exchange has happened in datetime64 format. YY/MM/DD HH/MM/SS

Table 3.1: Features of the minute candle data

1. Bitcoin (BTC)
2. Ethereum (ETH)
3. Tether (USDT)
4. BNB (BNB)
5. Solana (SOL)
6. USDC (USDC)
7. XRP (XRP)
8. Dogecoin (DOGE)
9. TRON (TRX)
10. Cardano (ADA)

Table 3.2: Crypto currencies to be included

3.2 Additional sections

Next to the crypto data, we will also take a look at additional features, specifically stock index prices and broad real-market indices. Unlike sector-specific stock indices, these real-market indices provide a broader representation of the overall market. They serve as key benchmarks for gauging the performance of the broader economy. These indices are given on a daily basis and have some gaps within the data because the stock market is closed on certain holidays or weekends. However, since the crypto data are on a minute-to-minute basis, we will need to find a way to concatenate and align these data sources nicely.

To resolve this issue, the data is duplicated. When data is missing over the weekends, the gaps are filled with the last recorded value, ensuring that every day of the year is complete. Subsequently, to transition from daily to minute data, we set the value to remain consistent throughout the entire day. The indices that we are using are the following:

- CAC40 Index: Index of the top 40 companies in the French exchange.
- DAX Index: Index of the most important companies in the German exchange.
- DowJones Index: Price-weighted stock market index of 30 major traded companies in the US.
- FTSE100 Index: Index of the most top 100 companies listed on the London Stock Exchange.
- S&P 500 Corporate Bond Index: Index of the performance of the corporate debt issued by companies in the S&P 500 Index.
- S&P 500 Index: Index of the 500 largest-cap U.S. companies.
- S&P Eurozone Sovereign Bond Index: Index of the performance of sovereign debt issued by countries within the Eurozone.
- DXY Curncy (R1): Index that measures the value of the US dollar against a weighted basket of six major currencies (EUR, JPY, GBP, CAD, SEK, CHF)
- VIX Index (L1): Real-time market index that captures the anticipation of volatility in the S&P500 index.
- ITRX EUR CDSI GEN 5Y Corp (L1): iTraxx Europe Credit Default Swap Index for investment-grade European corporate debt, with 5-year maturity denominated in Euros which basically captures the overall market stability.
- CDX IG CDSI GEN 5Y Corp (L1): Credit default swap index that tracks investment-grade corporate credit in North America.

Chapter 4

Methods

4.1 Ecometric Models

In this section, we will discuss the econometric models that will be used to predict the volatility of an asset. In particular, the econometric models ARCH and GARCH will be looked at. The reason for the selection of these models is that these models are very suited for financial time series data due to their ability to model volatility clustering, changing variances, and leverage effects. Also, these models have been frequently used for predicting other cryptocurrencies like Bitcoin, Ethereum and other cryptocurrencies [29] [30]. The econometric models will be used as a baseline model, providing benchmark performance score that we aim to improve using machine learning methods.

4.1.1 ARCH

ARCH, which stands for Autoregressive Conditional Heteroskedasticity, is a statistical model in econometrics and finance to analyze time-series data with varying levels of volatility. Heteroskedasticity basically means volatility, or any kind of deviation. Conditional stands for the idea that the volatility of the time series is not fixed over time, but it is based on where you are in the timeline. And auto regressive means that the current volatility is based on previous days or events. The key idea behind ARCH is to model the conditional variance of a time series as a function of its past observations. Variance is the spread of a data set around its mean value.

So in other words, it could be useful to use an ARCH model whenever the volatility changes depending on the time. Take, for example, the stock market over the years, where it has shown very high volatility during the COVID-19 period. The ARCH models assume that high and low volatility come in batches, so if today's volatility is high, then

the volatility upcoming day will also be high, and vise versa. The hypothesis is that the volatility holds a certain pattern and can therefore also be modelled.

Let r_t be a time series of returns, and let ϵ_t be the error term. The ARCH model assumes that the error term can be expressed as:

$$r_t = \mu + \epsilon_t, \qquad \epsilon_t = \sigma_t z_t \tag{4.1}$$

- μ: Mean of the series (can be constant or modeled as an AR process).
- σ_t: Time-varying standard deviation.
- z_t: A sequence of i.i.d. random variables with mean zero and variance one, often assumed normal distributed.

The key idea is that the volatility σ_t^2 dpeends on the past values of ϵ_t^2. Specially, for an ARCH(q) model:

$$\sigma_t^2 = \alpha_0 + \alpha_1 \epsilon_{t-1}^2 + \cdots + \alpha_q \epsilon_{t-q}^2 = \alpha_0 + \sum_{i=1}^{q} \alpha_i \epsilon_{t-i}^2 \tag{4.2}$$

where $\alpha_0 > 0$ and $\alpha_i \geq 0, i > 0$ [31].

An arch model is generally denoted as ARCH(p), where 'p' represents the number of lagged square errors included in the model. So the equation 4.3 represents the basic ARCH(1) model, where:

$$\sigma_t^2 = \alpha_0 + \alpha_1 \epsilon_{t-1}^2 \tag{4.3}$$

- σ_t^2 represents the volatility squared.
- α_0 and α_1 are parameters that need to be estimated from the data and are constants. This can be done using statistical methods such as maximum likelihood estimation (MLE).
- ϵ_{t-1}^2 represents the error term of a time step before the current one.

To explain it in words, if we are looking at daily data, then the volatility of today is going to be a function of the volatility of yesterday. So, if the volatility was high yesterday, then the volatility will probably be high today as well.

It is important to note that ARCH models does have its limitations and assumptions. For instance, they assume stationary of time series, meaning that its statistical properties such as mean and variance do not change over time. However, this is practically rarely the case, due to factors such as trends or seasonal factors.

Moreover, the basic ARCH model only captures volatility clustering and may not adequately address other features like leverage effects or long memory in volatility.

4.1.2 GARCH

The problem with the regular ARCH model is that the predicted volatility seems 'bursty'. Where it shows big spikes or jumps in volatility and goes back quite quickly to its regular state it was at. Thus, it lacks the ability to model persistent volatility, when the volatility tends to stay high or low for a period of time.

In order to capture volatility spikes that stay high or low for longer periods of time, we will also investigate GARCH models, which are an extension of the ARCH model. The big difference being that it also takes account for the volatility of previous time series in addition to the value of previous time series. The addition of GARCH is the incorporation of lagged conditional variances in addition to lagged squared errors, enabling the model to capture more complex patterns in volatility. The conditional variance refers to the expected value of the squared deviation of a random variable from its conditional mean, given some data.

The equation for the error σ_t is still the same, but GARCH(p, q) comes with two numerical terms, p for the number of conditional variances and q for the number of lagged squared errors. The volatility squared or σ_t^2 is now denoted as:

$$\sigma_t^2 = \alpha_0 + \alpha_1\epsilon_{t-1}^2 + \beta\sigma_{t-1} + \cdots + \alpha_q\epsilon_{t-q}^2 + \beta\sigma_{t-q} = \alpha_0 + \sum_{i=1}^{p}\alpha_i\epsilon_{t-i}^2 + \sum_{j=1}^{q}\beta\sigma_{t-j}^2 \quad (4.4)$$

- ϵ_{t-1}^2 represents the squared error term at time t - i.
- α_t is the baseline variance (unconditional variance).
- α_i is the impact of past squared returns on current volatility
- β_j is the impact of past conditional variances on current volatility.
- σ_t^2 is the volatility squared at time step t.

So, as an example, take a model GARCH(1,1) with daily data. So the volatility of today is affected by the value of the time series yesterday, but also the volatility of yesterday. By taking the volatility of yesterday into account, the model will show much less 'bursty' predictions. So, the volatility gets propagated over time instead of getting rid of it after one or two time periods.

4.1.3 PACF

In order to determine how many time steps need to be taken account with in the ARCH and GARCH models, we can make use of the Partial Autocorrelation Function (PACF). In time series analysis, it is often interesting to understand the relationship between a data point and its past values. The Autocorrelation Function (ACF) does this by measuring how correlated each data point is with its past values at different lags.

The partial autocorrelation function (PACF) takes this to a step, where it also calculates the correlation between a data point and its past values, but ignores the influence of intermediate data points. This means it measures the direct relationship between two specific points in time, while holding other time points constant.

PCAF(k) is calculated as the correlation between the data point at time t and the data point at time t-k, after removing the effects of intermediate data points, where k is the delay. The formula of PACF(k) is as follows [32]:

$$\text{PACF}(k) = \frac{\text{Cov}\left(y_t, y_{t-k} \mid y_{t-1}, y_{t-2}, \ldots, y_{t-k+1}\right)}{\sqrt{\text{Var}\left(y_t \mid y_{t-1}, y_{t-2}, \ldots, y_{t-k+1}\right) \cdot \text{Var}\left(y_{t-k} \mid y_{t-1}, y_{t-2}, \ldots, y_{t-k+1}\right)}} \tag{4.5}$$

- y_t represents the data point at time t.
- $\text{Cov}\left(y_t, y_{t-k} \mid y_{t-1}, y_{t-2}, \ldots, y_{t-k+1}\right)$ is the covariance between y_t and y_{k-k} after controlling for the influence of intermediate data points
- $\text{Var}\left(y_t \mid y_{t-1}, y_{t-2}, \ldots, y_{t-k+1}\right)$ is the conditional variance of y_t given $y_{t-1}, y_{t-2}, \ldots, y_{t-k+1}$
- $\text{Var}\left(y_{t-k} \mid y_{t-1}, y_{t-2}, \ldots, y_{t-k+1}\right)$ is the conditional variance of y_{t-k} given $y_{t-1}, y_{t-2}, \ldots, y_{t-k+1}$

4.2 Machine Learning Models

This section will go over the recurrent neural networks Long Short Term Memory (LSTM) and Gated Recurrent Unit (GRU) models. This section covers the mechanisms of these models, how these models compute their outputs, and the differences between them. The text on LSTM is based on the blogs post 'Understanding LSTM Networks' by Christopher Olah [33] and 'What is LSTM – Long Short Term Memory?' [34]. The text on GRUs is based on the sources [35] [36] [37].

4.2.1 Neural Networks

In order to understand the concepts of machine learning models such as Gated Recurrent Unit (GRU) and Long Short Term Models (LSTM), it is important first to understand the concept of Neural Networks. Neural networks are a class of machine learning algorithms inspired by the function and structure of the human brain. These networks interpret data using layers of artificial neurons that are connected to recognize patterns, make decisions, and solve complex problems by processing data through layers and interconnected artificial neurons.

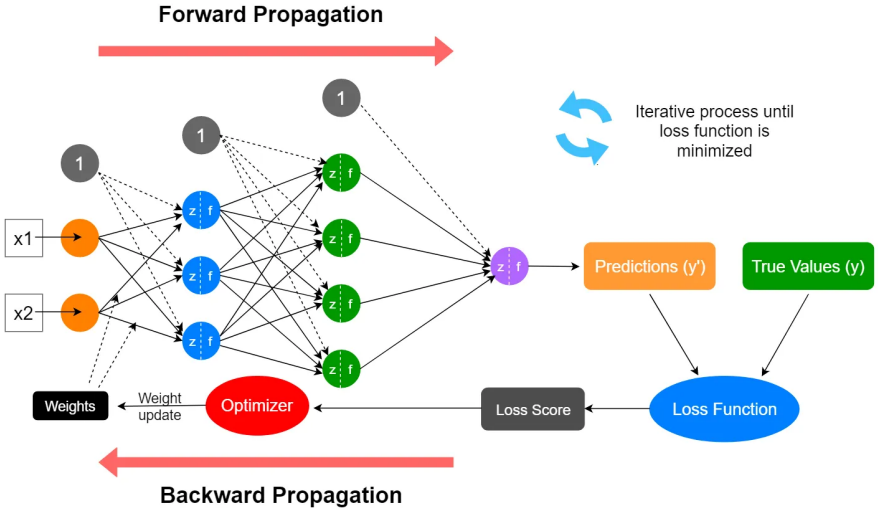


FIGURE 4.1: Schematic of a Neural Network [2]

The function of a neural network comes from artificial neurons. Each artificial neuron receives one or more inputs, processes these inputs, and produces an output. The transformation from inputs to outputs is calculated using an activation function, which determines the neuron's output based on the weighted sum of its inputs plus some bias variable. There are various activation functions such as sigmoid, tanh, and rectified linear unit (ReLU), each serving different purposes in helping the network learn from the data. These functions usually output a value between 0 and 1, where 1 means that

a neuron or vector unit is activated and 0 means that it is not activated. This helps the model to determine which parts of the input is relevant for the prediction.

Neural networks are organized into layers. The input layer is the first layer that receives the initial data, where each neuron represents a feature from the input data. After the input layer are one or more hidden layers, which perform various transformations on the input data. These hidden layers very important since they enable the network to learn complex representations and extract meaningful patterns. Finally, the output layer produces the result of the neural network, with the number of neurons in the layer, corresponding to the amount of possible outputs.

The training of a neural network is done using a process called forward propagation, where input data is passed through the network to obtain an output. The performance of the network is then evaluated using a loss function, which measures how far off the predicted output is compared to the actual target value.

To improve the network's performance, a process called backpropagation is used to update the weights and biases of the neurons. By applying the chain rule, it is calculated how much each weight contributes to the overall error and is then adjusted accordingly, working backwards from output layer to input layer. There exist several optimization algorithms such as gradient descent, stochastic gradient descent, or Adam [38].

4.2.2 Recurrent Neural Networks

Traditional neural networks have a major shortcoming, which is that they are unable to persist information but instead recalculates the output every time at every input. To put it into context, traditional neural networks are unable to identify what would happen next in a movie based on previous events.

Recurrent neural networks address this issue. They are networks with loops in them that allow information to persist.

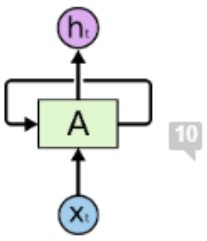
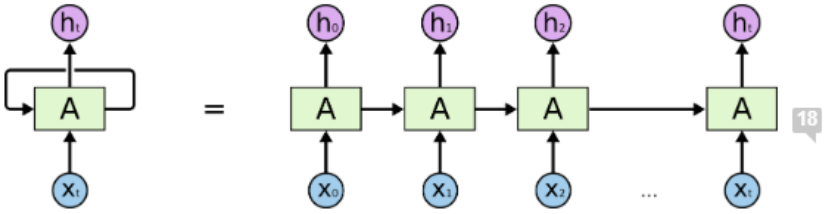


FIGURE 4.2: Schematic Recurrent Neural Network

In figure 4.2, a piece of neural network A, looks at some input x_t and produces a value h_t. The loop allows information to be passed from one step of the network to the next.

Although this loop may seem mysterious, it is not much different compared than a normal neural network. A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor, which look something like this when we unroll the loop:



An unrolled recurrent neural network.

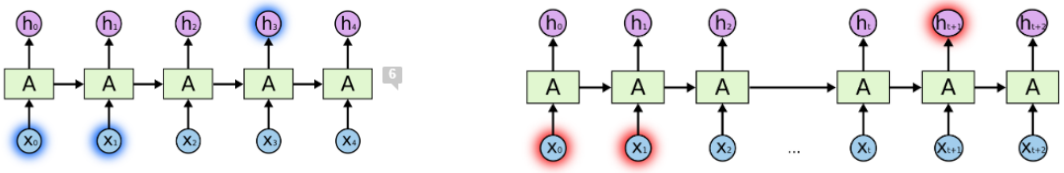
FIGURE 4.3: Unrolled RNN

This chain-like nature reveals that recurrent neural networks are inherently suited for handling sequences and lists, making them the natural neural network architecture for such data. In the last few years, RNNs more used for a variety of problems such as speech recognition, language modelling, translation etc. It is important to understand the structure of recurrent neural networks since they are the foundation to make use of Long Short Term Memory Networks (LSTMs).

One of the appeals of RNNs is the idea that they are might be able to connect previous information to the present task. Unfortunately, this is not always possible with RNNs. Sometimes, it is only necessary to look at recent information to perform the present task. Take, for example, a language model that tries to predict the next word based on the previous ones. When trying to predict the last word in the sentence, "The clouds are in the *sky*", no further context is needed since it is pretty clear that clouds are located in the sky.

However, in some other cases, more context is needed. Consider predicting the last word in the text, "I grew up in France ... I speak fluent *French*." Recent information suggest that the next word is probably the name of a language, but what language that exactly is, is not exactly clear and needs to be found in the context further back. And it is entirely possible for the gap between relevant information and the point where it is needed to become very large.

To visualize, look at figure 4.4. Both figures show the difference between the last meaningful input X and the relevant output for the two sentences whose words are to be predicted. Figure 4.4a represents when the desired context that is needed is close to



(A) Short context (B) Long context

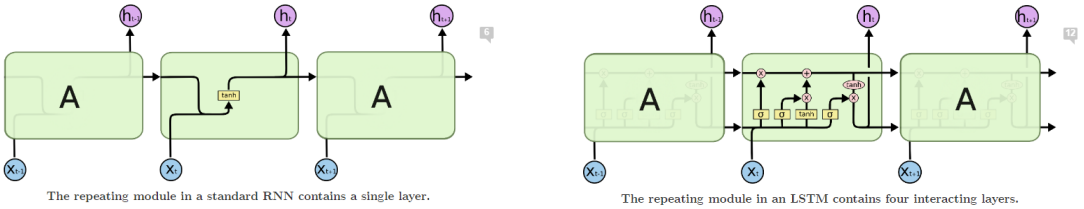
FIGURE 4.4: Visualization short and long context

the point for prediction and the chain stays relatively short. However, when the context is far away, like figure 4.4b, the distance between is very long, and depending on the context, the amount of unnecessary information in between can get very long which leads to a lot of unnecessary computation. This issue is called the vanishing gradient problem, where an RNN is unable to find the minimum of the loss function since there is too much information to process [39].

In theory, RNNs are able of handling 'long-term dependencies'. A human could carefully pick parameters for them to toy problems in this form. Sadly, in practice, RNNs do not seem to be able to learn them and why it is difficult, which was researched in the paper [33].

4.2.3 LSTM

Long Short Term memory networks, usually called LSTMs, are a special kind of RNN capable of learning long-term dependencies and have been improving ever since their introduction in 1997. LSTMs are explicitly designed to avoid the long-term dependency problem. Recalling information is actually their default behavior and not something they struggle to learn. Both LSTM and GRU 4.2.4 are designed to preserve gradients over long time sequences, addressing the vanishing gradient problem.



(A) Repeating module RNN (B) Repeating module LSTM

FIGURE 4.5: Comparison module RNN and LSTM

All recurrent neural networks have a form of a chain of repeating modules. In a standard RNN, this repeating module will have a very simple structure, such as a single tanh layer as seen in figure 5.20b. LSTMs also have this chain like structure, but the repeating

module is a bit more complicated. Instead of having only a single neural network layer, there are four, interacting in a special way seen in figure 5.20a. LSTMs use three gates: input, forget, and output gates, along with a cell state that carries information across long time intervals.

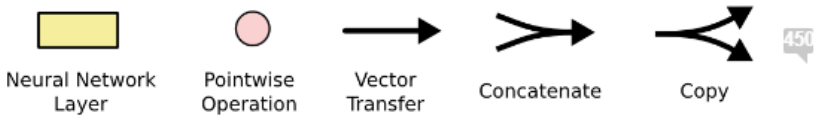


FIGURE 4.6: Icons LSTM

In the diagram 4.6, each arrow represents an entire vector, from the output of one node to the input of others. The pink circles represent pointwise operations, like vector addition, while the yellow boxes are layers of learned neural networks. Arrows merging denote concatenation and arrows splitting denote its content being copied and copies going to different locations.

The key to LSTMs is the cell state, the horizontal line running through the top of the diagram. The cell state is similar to a conveyor belt. It is running straight down the entire chain, with only having minor linear interactions, and is a vector used to store information from longer periods ago. Due to minor interactions, it makes it easy for information to flow unchanged and helps to preserve gradients during backpropagation and allows the model to remember data over a long term.(Figure 4.7).

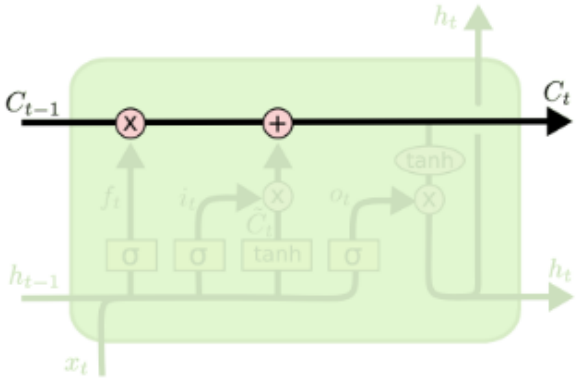


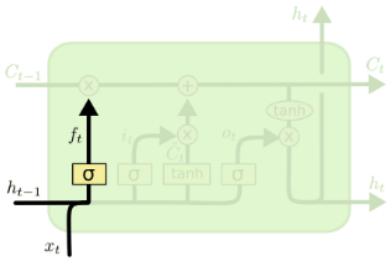
FIGURE 4.7: Highlight horizontal LSTM

The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates. Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation, where they output how much information each component should

let through. An LSTM has three gates, to protect and control the cell state. The hidden state is a filtered version of the cell state that combines short-term and long-term information.

A LSTM goes through several steps to make its prediction. The first step is to decide what information is going to be thrown away from the cell state. This decision is made by a tanh layer called the 'forget layer'. It takes the two inputs h_{t-1} (previous cell output) and x_t (input at a particular time t) and multiplies with weight matrices followed by the addition of a bias value. This gets passed through an activation function which outputs a number between 0 and 1 for each cell state C_{t-1}, with 1 meaning 'completely keep the data' and 0 meaning 'forget all the data'.

Take, for example, the language model that attempts to predict a next word in a sentence based on the previous ones. In a problem like this, a cell state might want to remember the gender of the subject in order to refer to the correct pronouns. But when a new subject is seen, we want to forget the gender of the old subject.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

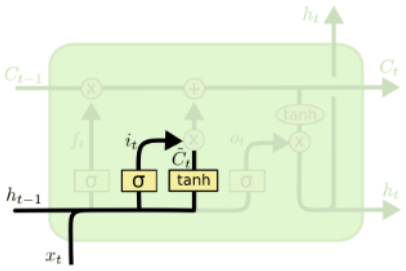
FIGURE 4.8: 'Forget gate' LSTM

Next, it is decided what information to store in the cell state. This has two parts. First, a sigmoid function called the 'input layer' decides which values we will update. Then a vector is created using a tanh function with new candidate values, $\tilde{C}_t$, that could be added to the state. Both calculations of regulation of information is similar to the calculation by the 'forget layer'.

In the language model example, this would be where we want to add gender to the new subject to the cell state, to replace the old one that we are forgetting.

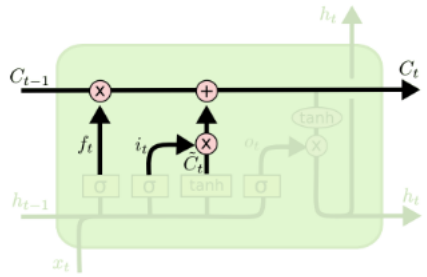
Now it is time to update the cell state. This is done by multiplying the old state by f_t, forgetting the things we decided to forget earlier. Then adding these values $i_t * \tilde{C}_t$. These are the new candidate values, scaled by how much it was decided to update each state value.

In context of the language example, here we drop the information about the old subject's gender and add the new information, as we decided in the previous steps.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

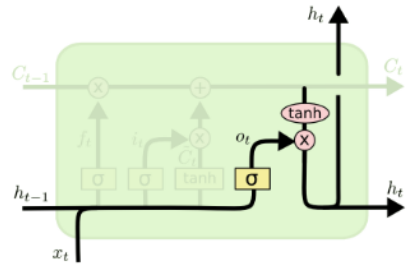
FIGURE 4.9: 'Input gate' LSTM



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

FIGURE 4.10: 'Input gate' calculation LSTM

Finally, the last step is to decide what the output is going to be. This is done in the layer called the 'output layer'. First, an output vector o_t is constructed by filtering the values to be remembered using inputs h_{t-1} and x_t through a sigmoid function. Next, a vector is generated by applying the tanh function to the cell state. At last, both vector are multiplied and are sent as an output and input for the next cell.



$$o_t = \sigma\left(W_o \; [\, h_{t-1}, x_t] \; + \; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

FIGURE 4.11: 'Output gate' calculation LSTM

LSTMs are particularly useful in scenarios where the order of the data and long-term dependencies are crucial for making accurate predictions or understanding the sequence. Aside various applicances such as from Natural Language Processing, image processing, or speech recognition, LSTMs have proven to be very useful with Time Series Prediction, such as weather, sales, and stock, or rather in this case, crypto market forecasting. Hence, the use of this model for research.

4.2.4 GRU

Just like LSTM, Gated Recurrent Unit (GRU) aims to solve the vanishing gradient problem which comes with a standard recurrent neural network. GRU can be considered, introduced in 2014 by Cho et al., to be a variation on the LSTM, since they are both designed similarly and produce equally excellent results.

Unlike LSTM, GRU only uses two gates, which are the update and reset gate. The GRU does not make use of a cell state, but instead updates the hidden state of the network accordingly using the prementioned gates. These gates are trained to control the flow of information in and out of the network, knowing when to keep information, without washing it through time or removing information which is irrelevant for the prediction.

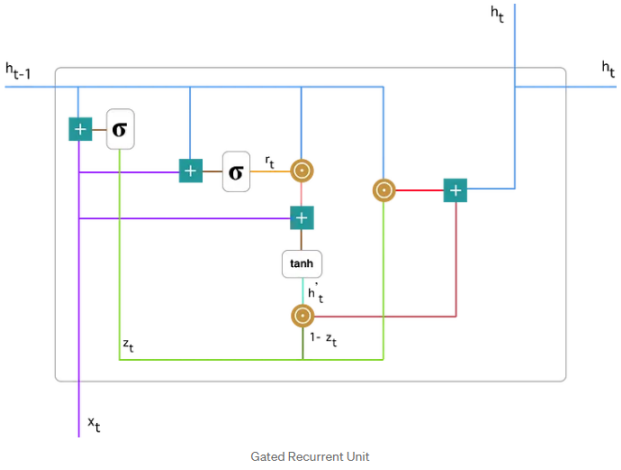


FIGURE 4.12: Single GRU unit

Just like LSTM, a GRU is constructed of a sequence of single GRU units and follows a few steps in order to make its prediction. First, the update gate z_t for time step t is calculated using the formula

$$z_t = \sigma(W^{(z)}x_t + U^{(z)}h_{t-1}) \tag{4.6}$$

The calculation starts with the 'update gate'. The update gate helps the model determine how much of the past information (from previous time steps) needs to be passed along to the future. This gate is similar to the 'output gate' in a LSTM recurrent unit. When input x_t enters into the network unit, it is multiplied by its own weight $W(z)$. The same holds for $h_{(t-1)}$, which holds the information for the previous t-1 units and is multiplied by its own weight U(z). Both results are added together and put into a sigmoid function, which outputs a value between 0 and 1.

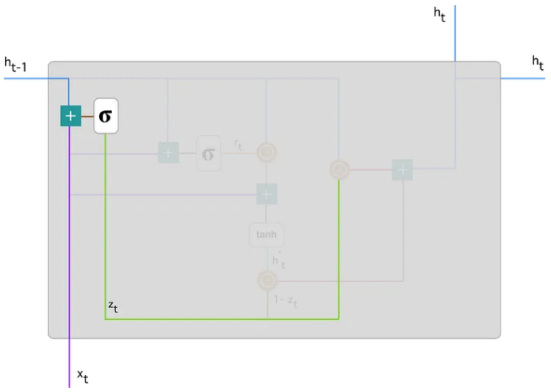


FIGURE 4.13: 'update gate' GRU

The next gate that is used is the 'reset gate'. The reset gate is used by the model to decide how much of the past information to forget. This gate is similar to the combination of the 'input gate' and 'forget gate' in an LSTM recurrent unit. The formula is similar to the update gate. The difference being that the weights and the gate are used differently. To calculate it, we use the following formula:

$$r_t = \sigma(W^{(r)}x_t + U^{(r)}h_{t-1} \tag{4.7}$$

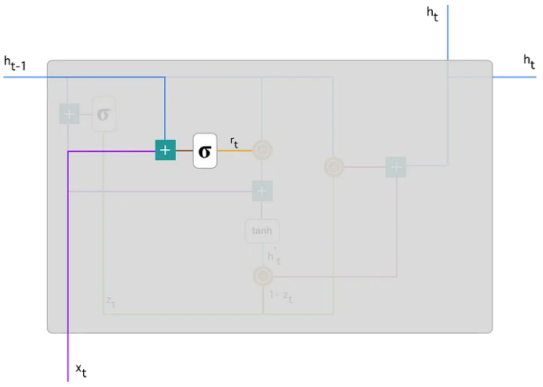


FIGURE 4.14: 'reset gate' GRU

Now we will use both gates to affect the final output. First, we start with the usage of the reset gate. A new memory content is introduced which is used to store relevant information from the past using the reset gate. The formula holds the following structure:

$$h'_t = tanh(Wx_t + r_t \odot Uh_{t-1}) \tag{4.8}$$

The input x_t is multiplied by a weight W and h_{t-1} with a weight U. Then the Hadamard (element-wise) product between the reset gate r_t and Uh_{t-1}. That will determine what

to remove from the previous step. The results are added up and applied to a tanh function. If we take the same example from the LSTM, then when we need to replace the gender of the subject we are talking about, the change of information will be done in this step.

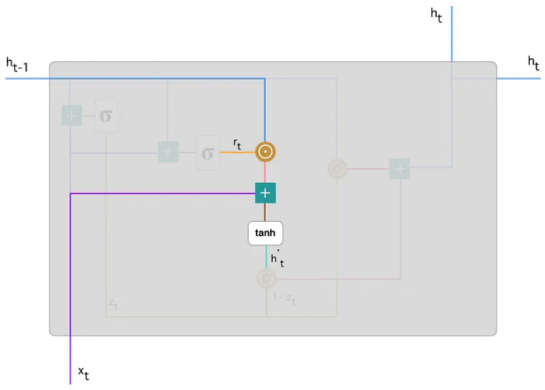


FIGURE 4.15: current memory content GRU

In the last step, the network needs to calculate the vector h_t, which holds information for the current unit and passes it to the network. IN order to do so, the update gate is used. This determines what to collect from the current memory content h'_t and what from the previous steps $h_(t-1)$. Apply element-wise multiplication to the update gate z_t and h_{t-1}, as well as $1 - z_t$ and h'_t. And sum both of the results. The formula will look like the following equation:

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot h'_t \tag{4.9}$$

The model will be able to learn how to adjust the vector z_t, where if the value z_t is close to 1, then it would mean that information from a long time ago is relevant for prediction and vise versa, when the value is close to 0, then the old information would be wiped from the memory.

So, GRUs are able to store and filter the information using their update and reset gates. That eliminates the vanishing gradient problem since the model is not washing out the new input every single time but keeps relevant information and passes it down to the next time steps of the network.

While both LSTM and GRU are designed to address the vanishing gradient problem and capture long-term dependencies in sequential data, they hold some differences between each other. To name a few:

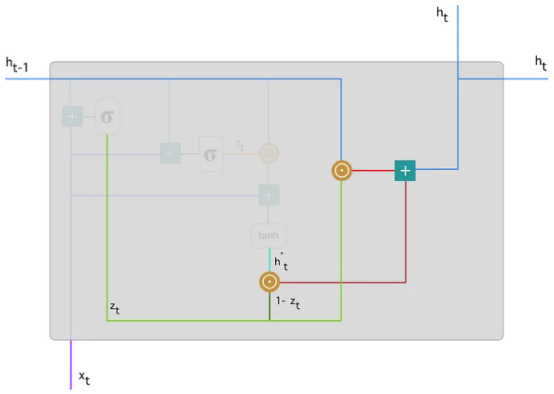


FIGURE 4.16: Final memory step GRU

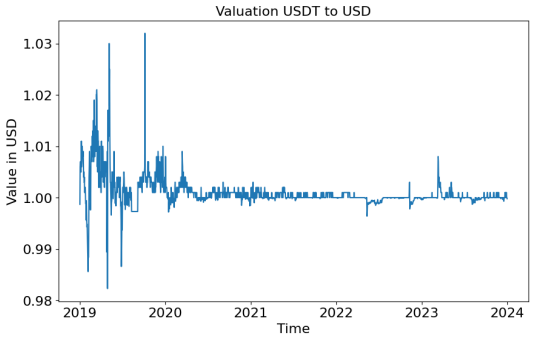
- GRUs have a simpler architecture with only 2 gates: reset gate and update gate, without having a cell state
- GRUs have fewer parameters compared to LSTMs, making them more computationally efficient and easier to train, especially on smaller datasets
- GRUs do not have a separate cell state, but instead directly update the hidden state. This makes them more compact, but may result in a less explicit control over long-term dependencies.
- GRUs are often preferred in scenarios where computational resources are limited or where faster training times are desired, such as real-time applications or working with smaller datasets.

Chapter 5

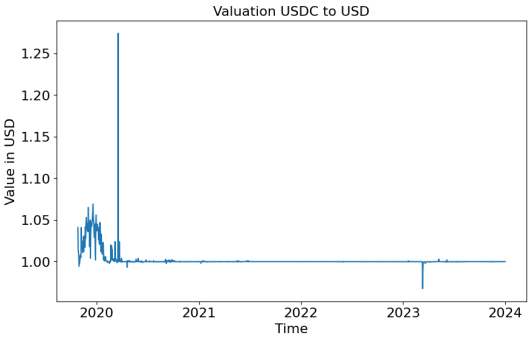
Experiments and results

5.1 Data for testing

For the results, the frequency of the data will be on a daily basis. In order to convert the minute data into daily data, only the first datapoint of an asset of the minute data will be inspected. In order to do this, simply the whole minute dataset can be filtered by taking each 1440th datapoint. This means that only the first minute of the day will be taken into the first dataset. So, this means that the 'exchange_time' of each day will be '00:00:00 +00:00 (UTC)'. The average volatility of USDT is 1.1042108655073055, where USDC holds an average volatility of 1.9972634153633628.



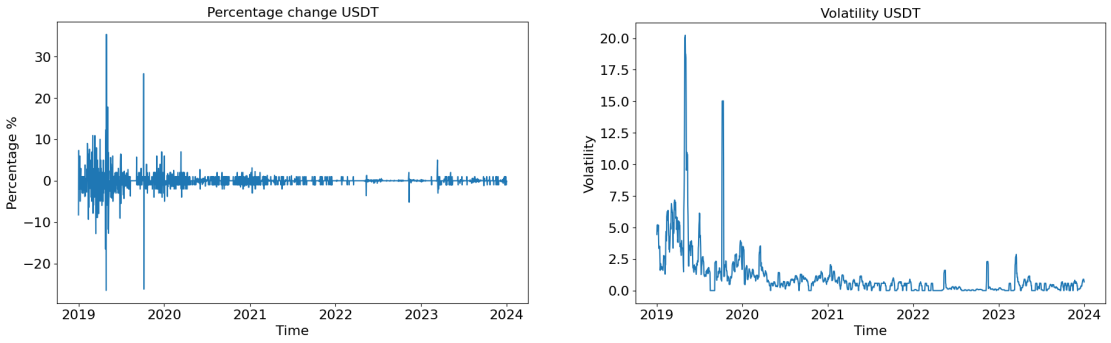
(A) Value USDT



(B) Value USDC

As mentioned, the current value of volatility is calculated as the standard deviation of returns over a span of the previous week. The return is the percentage change per datapoint, which is in this case the daily percentage change. Note that the y-values are upscaled by 1000 since ARCH and GARCH would not work otherwise.

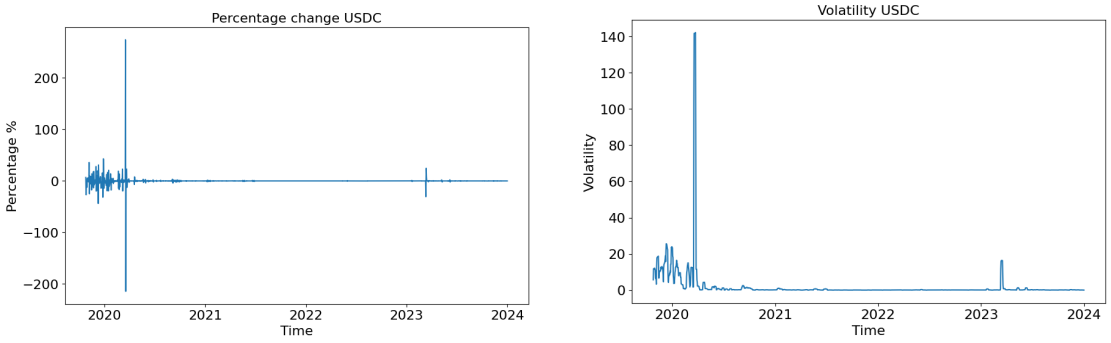
On the left side, the percentage changes of USDT is shown from 2019 to 2024. On the right side, the volatility based on USDT on the left-hand percentage change. It is visible



(A) Percentage changes USDT (B) Volatility USDT

FIGURE 5.2: Return and volatility USDT

that the volatility of USDT is quite high in the beginning 2018-2019, with heaving big spikes of volatility. However, the size of the spikes, as well as the overall volatility, decrease over time.



(A) Percentage changes USDC (B) Volatility USDC

FIGURE 5.3: Return and volatility USDC

In figure 5.16, the percentage change and volatility respectively can be seen of USDC. The CCCAGG data does not contain any data of USDC before October 19, 2019, where the valuation in the data from USDC to USD is a value of 0. Since volatility is calculated by looking back at a 7-day window, the data for these plots begin on October 26, 2019.

Similar to USDT, the volatility at the start is relatively high, with also a very big peak on March 18, 2020, where the price of USDC goes up in one day to 1.27 USD, but goes down in relative normal value the next day. However, these high volatility levels tend to decrease with time.

The data will be split into 80% training and 20% test. So this means that the training data for USDT will be from 2018-2023 and the test data will be 2023-2024. The training data for USDT holds 1461 samples and test data 365 samples. For USDC, the training data will be 1224 samples and the test data will be 305 samples, being the last 305 days of year 2023.

5.2 Code

The code for running econometric as well as the machine learning models are implemented in Python 3.11.7 within the Anaconda distribution environment. Python was chosen for its rich and easy to use ecosystem of machine learning libraries. Anaconda provides a convenient package management system that ensures compatibility and reproducibility across different computing environments. The results are run on a Ryzen 7800x3D processor with 32 GB of RAM.

The key libraries and packages used in the implementation are the following:

- **NumPy**: For numerical computations and array operations, used in model training and evaluation.
- **Pandas**: Used to format the data in proper data frames for data manipulation and preprocessing tasks, including feature engineering and dataset splitting.
- **Matplotlib**: Used for data visualization, like plotting performance of models and other plots
- **statsmodels**: Used for pacf plotting to determine the parameters of ARCH/-GARCH.
- **Arch**: Used for implementing the ARCH and GARCH models, for fitting the models as well as making a prediction.
- **Scikit-learn**: Used for scaling and other calculation operations.
- **TensorFlow**: Used for easy to use, building, training and deploying machine learning models.

Since the value of stablecoins are quite stable around 1 USD, the volatility will not fluctuate as much, and it might be difficult to verify whether the model works correctly or not. Therefore, in order to measure the robustness of the code and the model, some results regarding Bitcoin and Ethereum will be reported.

5.2.1 Root Mean Square Error

For measuring the performance, the Root Mean Squared Error (RMSE) is used since it provides a clear indication of average differences of errors and is very commonly used in other research comparing problems. The goal is to have a RMSE as small as possible,

where a RMSE of 0 means that the model provides a perfect prediction. The formula for the RMSE is the following:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{5.1}$$

Where:

- n is the number of observations.
- y_i is the actual value.
- $\hat{y}_i$ is the predicted value.

5.2.2 Sensitivity Analysis

In this section, the adjustable parameters for model construction and sensitivity analysis will be examined. Machine learning models rely on several key parameters that can be adjusted accordingly, each of which playing a crucial role in determining the accuracy and reliability of the predictions. Below, a brief overview of these parameters is provided, which are the following.

5.2.3 Parameter 1: Batch size (n)

Description: The batch size refers to the number of training examples used in one iteration of the model' optimization during training. E.g the dataset for USDT has 1402 samples, and if the batch size is set to 32, then the model will process 32 samples each iteration and updates its weights based on these 32 examples. This would result in $1402/32 = 43.8$ batches per epoch. Test ranges: `1, 8, 16, 32, 64`

Assumptions: Smaller batch sizes are slower but have more stable updates. Whereas bigger batch sizes are faster and are better at generalization.

5.2.4 Parameter 2: Number of Epochs

Description: The number of epochs refers to the number of times the model passes through the entire training set during the training process. Each epoch allows the model to adjust its weights to minimize the error between the predicted and actual outcomes. Test ranges: `1 to 20`

Assumptions: More epochs generally improves the model's performance due to the increased iterations of learning, but too many epochs could induce overfitting, where the model performs well on the training data but poorly on the unseen/test data.

5.2.5 Parameter 3: Window size (n)

Description: Since we are working with sequential data, how LSTM and GRU works is that it looks n data elements back to predict the following value. For example, when working with daily data and n=7, then we are using past weeks data to predict the following value. Test ranges: `1 to 60, with increments of 5`

Assumptions: A big window size would take in a lot of previous information, however by taking in more information, more input noise would be included which might have a negative impact on the performance. Inversely, a smaller window size might be more straightforward but might lack information for the prediction.

5.2.6 Parameter 4: Learning Rate (α)

Description: The learning rate controls the size of the steps during the optimization process of the gradient descent. It determines how quickly the model updates its parameters in response to the calculated error. Test ranges: `0.0001, 0.001, 0.01`

Assumptions: A smaller learning rate may lead to more accurate results but requires more iterations, while a larger learning rate can speed up convergence but risks overshooting the optimal solution.

5.2.7 Parameter 5: Number of LSTM or GRU units

Description: The number of units in the LSTM or GRU layers defines the dimensionality of the output space, or the number of hidden units in each layer. These units control how much information the model can capture from the input sequence. Test ranges: `16, 32, 64, 128`

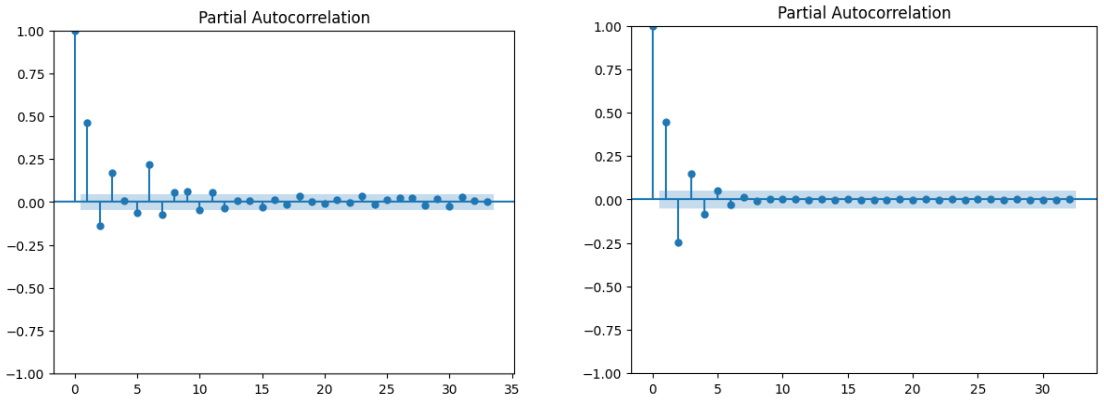
Assumptions: A higher number of units allows the model to learn more complex patterns, but may increase the risk of overfitting and require more computational resources. Conversely, a smaller number of units may lead to faster training but could limit the model's capacity to capture detailed information from the data.

5.3 Results

5.4 ARCH/GARCH

In this section, the results of the analysis using Autoregressive Conditional Heteroskedasticity (ARCH) and Generalized Autoregressive Conditional Heteroskedasticity (GARCH) are presented. These models were employed to capture and forecast the volatility in our dataset.

To start, we will need to determine what the parameters for ARCH(q) and GARCH(p, q) will be, or rather how many lags the model will have to look back at. In order to do so, we can make use of the Partial Autocorrelation Function (PACF). The returns of USDT will be used as input. The return is squared to normalize the values and inputted using the statsmodels `plot_pacf` function. This will return the following plot:



(A) PACF of USDT (B) Volatility USDC

FIGURE 5.4: Return and volatility USDC

The number of parameters for the ARCH/GARCH is determined by observing the number of dots following the first one, that are gradually decreasing while still remaining outside the significance area. In the PACF for USDT, many dots fall outside the significance area highlighted in blue. This might suggest that taking into account many time delays might help when setting up a model. In the PACF of USDC only 4/5 dots fall outside the significance area after the first one.

	coef	std err	t	P>|t|	95.0% Conf. Int.
omega	6.3350e-03	5.265e-03	1.203	0.229	[-3.984e-03,1.665e-02]
alpha[1]	0.0739	2.650e-02	2.789	5.291e-03	[2.197e-02, 0.126]
beta[1]	0.9261	3.469e-02	26.692	5.760e-157	[0.858, 0.994]

FIGURE 5.5: Summary of GARCH (1, 1) USDT

This is further emphasized when a GARCH(1,1) model is fitted. Both alpha[1] and beta[1], standing for the p and q parameter respectively, show to have significant values, seen in the column $P > |t|$. However, when a GARCH(3, 3) is fitted the parameters no longer show any significant value.

In order to make a proper plot for the prediction of volatility, a rolling forecast origin is used. For the final 365 days of the data, a ARCH(p) or GARCH(p, q) model is built for each forecasting period. After each prediction, the new day is corporated into the dataset, an ARCH/GARCH model is rebuilt and a new 1-day forward prediction is made.

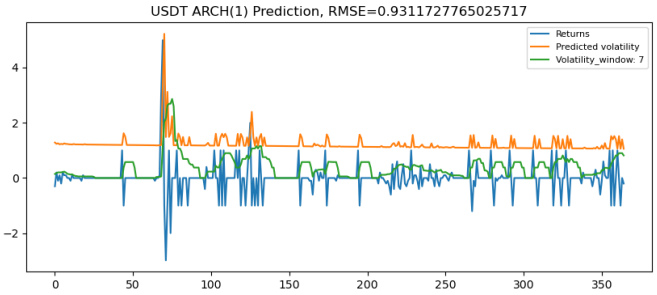


FIGURE 5.6: ARCH (1) USDT

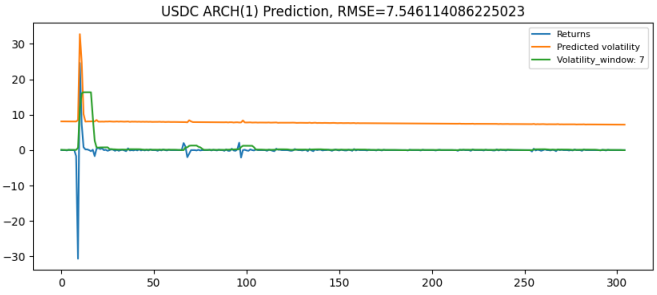
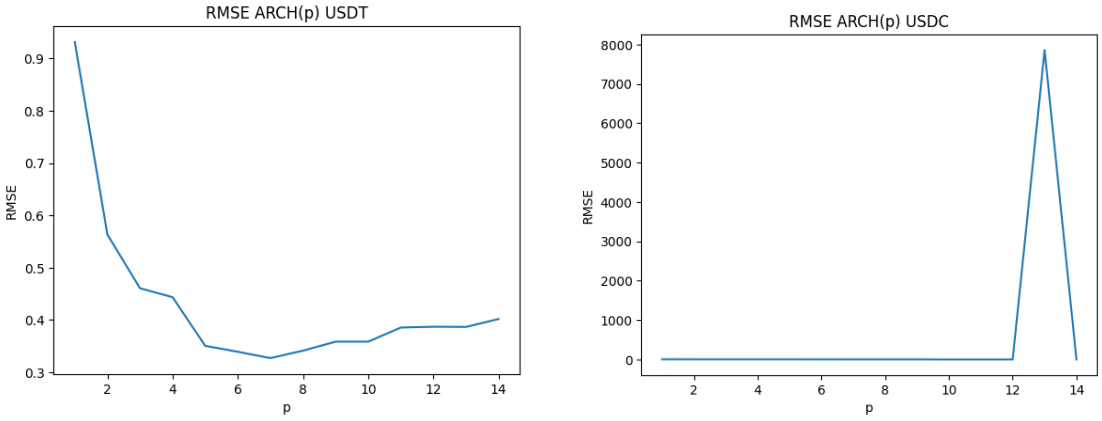


FIGURE 5.7: ARCH (1) USDC

First, an analysis of ARCH models will be examined. The plot shows the predictions created by the associated ARCH models. The blue line represents the return, orange the predicted volatility created by the ARCH model and the green line is the volatility with a window of 7 days. It can be seen that the predicted volatility spikes when the returns spike as well. However, it is noticeable that the predicted volatility seems to be shifted upward compared to the actual volatility.

After testing, for USDT, it seems that adding more time lags to the ARCH models lowers its overall average, therefore better fitting the actual volatility (see 9.22 & 9.2). For USDC, it is less obvious what happens with the shape of the prediction, but it is important to test which parameters would fit the model the best. Therefore, to identify the best parameter settings, various ARCH models are fitted to the data with different time-lag lookbacks.



(A) PACF of USDT (B) Volatility USDC

FIGURE 5.8: Return and volatility USDC

The optimal setting of the ARCH parameter for USDT seems to be when 7 time lags are used for the model with a RMSE of 0.3273111629275738 9.1. For USDC, where the optimal number of time lags is 10, which results in a RMSE of 5.845639783221598 9.2. The sudden high spike in USDC data is probably the reason why the RMSE is so high compared to USDT. Especially the big spike on p=13, shows the inconsistency and how much the ARCH model struggles to make a correct prediction.

When creating a GARCH(p, q) fitted on the data, it will give a prediction as the following. As the theory suggests, instead of assuming bursty volatility, the volatility stays high for a prolonged period.

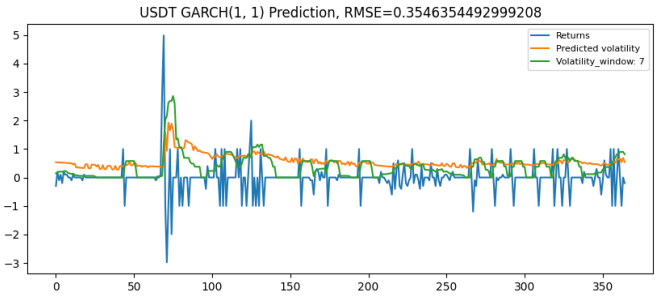


FIGURE 5.9: GARCH (1, 1) USDT

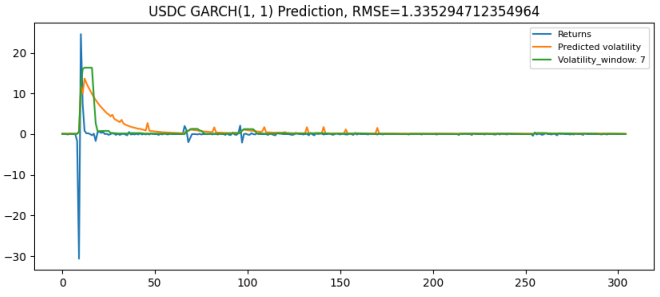
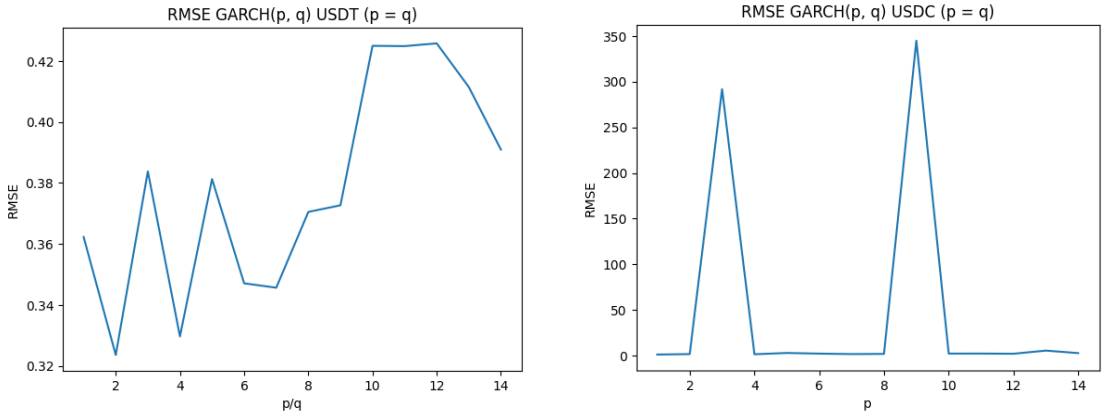


FIGURE 5.10: GARCH (1, 1) USDC

However, compared to the actual volatility, the model's performance is quite poor. The actual volatility shows sharper peaks and deeper dips, whereas the GARCH model remains relatively high and fails to capture the daily intricacies of the volatility.



(A) SA GARCH(p, q) USDT

(B) SA GARCH(p, q) USDC

FIGURE 5.11: Sensitivity analysis GARCH(p, q)

The performance of a GARCH(2, 2) model performs best in USDT data with an RMSE of 0.3235505361672609 9.3, and a GARCH(1, 1) model performs best in USDC data with an RMSE of 1.335294712354964 9.4. The RMSE is better than the optimized ARCH model. This is likely because the results are nicely averaged out to the actual values. However, the shape of the plot does not provide sufficient insight into the specific daily price volatility.

5.5 Machine Learning models

5.5.1 Implementation

The implementation of machine learning models is an important step in processing the available data into producible results. This section outlines the process followed to develop, train, and evaluate the LSTM and GRU models to address the problem at hand. The machine learning workflow includes data preprocessing, feature engineering, model selection, and model evaluation.

The first key aspect to consider is that the machine learning models that are used are designed to handle sequential data. This means that the model uses past observations to predict future values, specifically with the aim of predicting the value at time step $t + 1$ by using the values from time steps $t - n$ to t as input. Note that the values of the data are scaled from 0 to 1 using a MinMaxScaler to increase the performance by reducing the difference between features.

The parameters of the LSTM or GRU models are defined, such as the amount of LSTM/GRU units, the mount of dense neurons, and the optimizer with a loss function. Then a model can be trained by fitting it onto the training set with a defined batch size and epochs. Once a model is fitted, predictions can be made based on the test input data and an RMSE score can be calculated by looking at the predictions and actual values.

There are various parameters that can significantly impact the performance of the models, making it difficult to determine the optimal settings. To address this problem and to identify the best parameter configuration, the structured approach will be followed as outlined in the following flowchart:

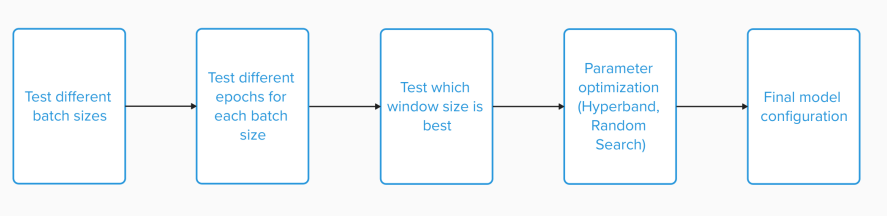


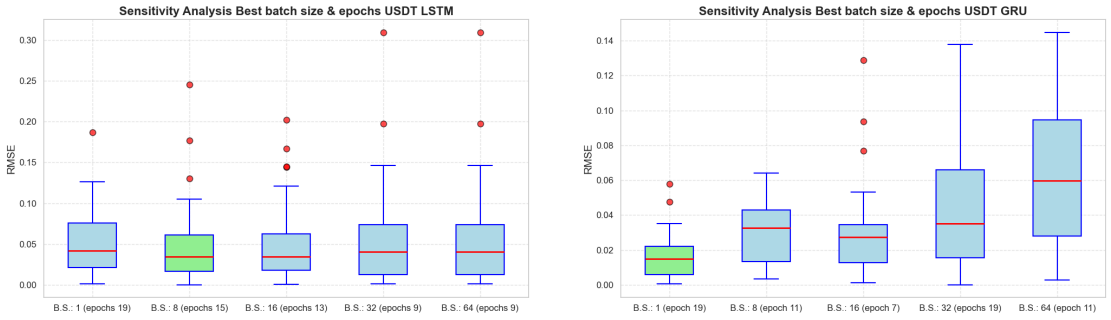
FIGURE 5.12: Flowchart workflow model testing

The optimization process starts by testing various combinations of batch sizes and epoch amounts to determine which configuration gives the best performance. The batch size that produces the highest performance, along with its corresponding optimal number of epochs, is selected for further optimization. The optimal window size is then determined during data preprocessing to enhance performance. After identifying these values, a more in-depth parameter optimization is conducted to fine-tune the model's internal parameters and maximize its overall performance. Following the finding of these values, a more detailed parameter optimization is performed to fine-tune the model's internal parameters and maximize its overall performance. The order of these actions are based on the impact of the model, where the batch sizes and epoch range matter the most, therefore they are done first.

The results produced by machine learning models are inconsistent, even when the same input is used. Therefore, to ensure robust evaluation of the model's performance, the model is trained 30 times with the same inputs, which will also be visible in the plots in the following section. In this research, single-layer LSTM and GRU models are implemented with the argument that due to the limited size of the data, a simple model would be preferred. Based on other papers, the amount of units per model varies quite a lot depending on the data used. Usually the number of units is a power of two ranging from (16 to 128) [41] [42] [43] [44]. Therefore, before hyperparameter optimization, the base machine learning model of LSTM and GRU will have 64 units with a dense layer of 8 neurons. The learning rate is set to 0.0001 with a window size of 60.

5.5.2 Setting up a single variate model

As indicated in the flowchart, the first step is to experiment with various batch sizes and evaluate the performance across different epochs, ranging from 1 to 20. To visualize the results, the performance values from these multiple simulations are plotted using boxplots, which highlights the distribution and spread of the data. The y-axis represents the RMSE , while the x-axis displays the corresponding model for the box plot. In this section, only the best performing parameters per machine learning model for each stablecoin are shown. The boxplot with the best performing setting is highlighted in green. The results per epoch per batch size can be found in the appendix 9.

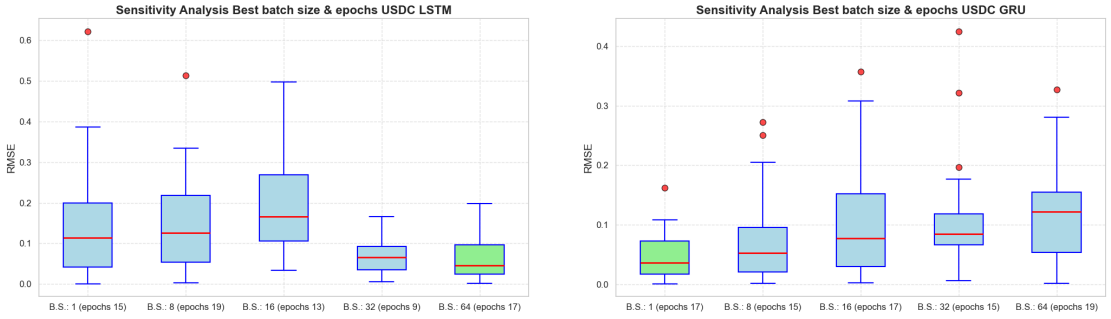


(A) Best parameters batch size & epochs on LSTM

(B) Best parameters batch size & epochs on GRU

FIGURE 5.13: Best parameters batch size & epochs USDT

For the USDT dataset, the LSTM model shows an optimal performance with a batch size of 8 and 15 epochs. When comparing different batch sizes, the LSTM performance does not seem to perform better or consistently when using different batch sizes in comparison to the GRU model. However, in the case of the GRU model, it is clearly visible that optimal results are achieved with a smaller batch size and 19 epochs, while performance becomes increasingly inconsistent as the batch size increases.

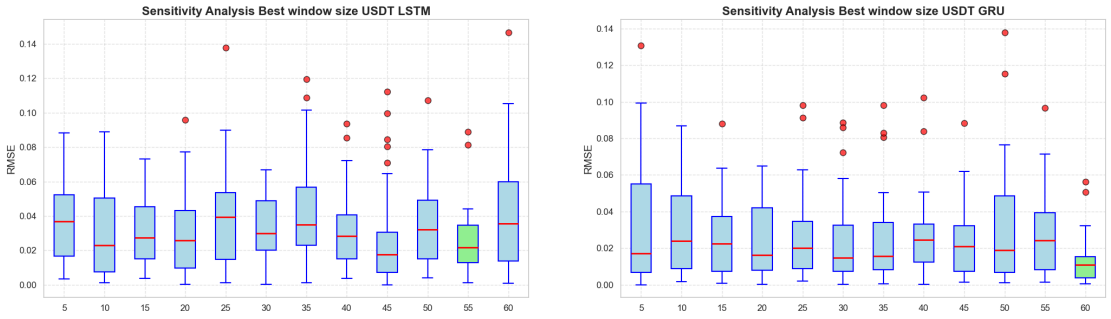


(A) Best parameters batch size & epochs on LSTM

(B) Best parameters batch size & epochs on GRU

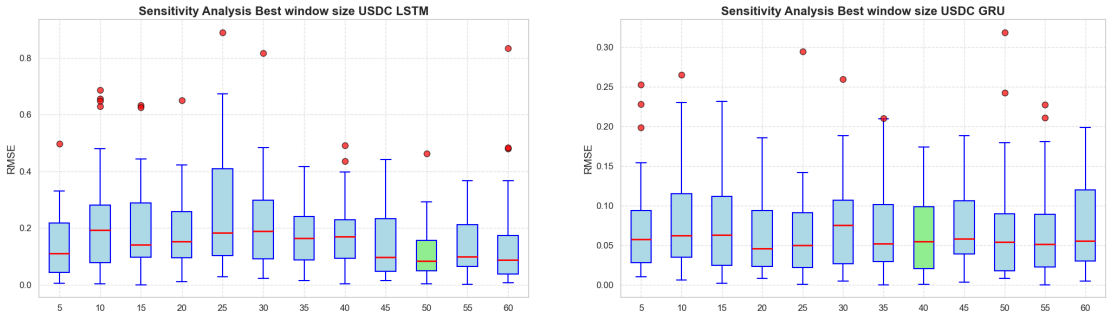
FIGURE 5.14: Best parameters batch size & epochs USDC

For the USDC dataset, the LSTM model shows the best performance with a batch size of 64 and 17 epochs, and this time it seems to perform differently when using different batch sizes. Similarly, the GRU model achieves comparable results, as with the USDT dataset, where it performs optimally with a smaller batch size of 1 and epochs of 17. The best performing models are continued to be optimized by test different window sizes that are used to predict the next value in the sequential data.



(A) Best window size on LSTM (B) Best window size on GRU

FIGURE 5.15: Best parameters window sizes USDT



(A) Best window size on LSTM (B) Best window size on GRU

FIGURE 5.16: Best parameters window sizes USDC

For calibrating the window sizes, there seems to be no visible pattern, as the models do no necessarily perform better with a larger window size, nor is there a distinguishable best window size compared to the other sizes. Nevertheless, the best-fitting window size parameter for each model will be considered in subsequent hyperparameter optimization.

For hyperparameter optimization, we will make use of Hyperband [45]. Hyperband is an optimization algorithm designed to efficiently tune hyperparameters by allocating computational resources in a more efficient way compared to grid or random search.

The Hyperband starts with generating multiple random hyperparameter configurations, these can be any combination described in the sensitivity analysis section 5.2.2. Each of these configurations are exploited using a limited amount of epochs. Hyperband trains all configurations for a few epochs to get a sense of their performance.

Parameter	LSTM	GRU
LSTM/GRU units	64	64
Dense units	16	64
Learning rate	0.0001	0.01

TABLE 5.1: Hyperband results USDT

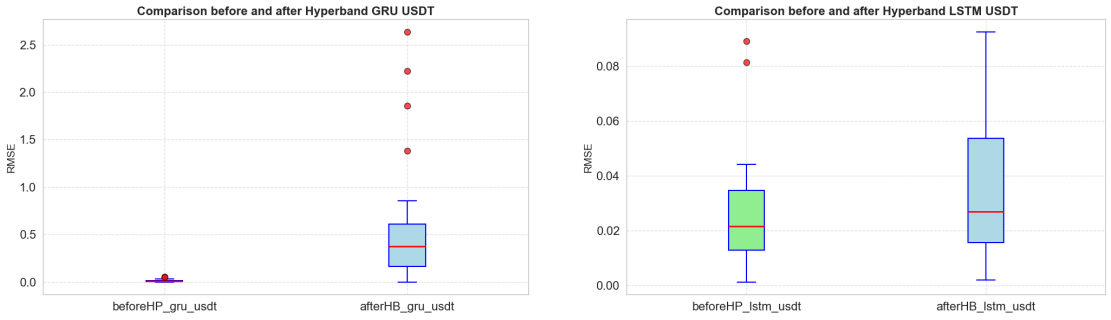
Parameter	LSTM *	GRU
LSTM/GRU units	16	32
Dense units	64	32
Learning rate	0.001	0.001

TABLE 5.2: Hyperband results USDC

After initial evaluation, the worst performing configurations are pruned, while the top-performing models are continued to be evaluated, receiving additional computational resources. This step gets repeated, where the worst models get discarded, and the best models continue to get more epochs.

What Hyperband makes great is that it balances exploration (trying many different hyperparameter configurations) with exploitation (focusing more resources on the best-performing configurations) instead of completely checking each configuration, which is too computationally heavy. It uses a factor parameter to control how aggressively the model prunes the models at each stage. This parameter is set to 3, which means that one-third of the models will continue each round.

Once Hyperband finishes its search, it returns the best hyperparameter configuration based on the objective metric. This is the set of hyperparameters that resulted in the best performance of the model. Each combination will be tested 15 times and the average result will be used to evaluate the performance of that specific combination.

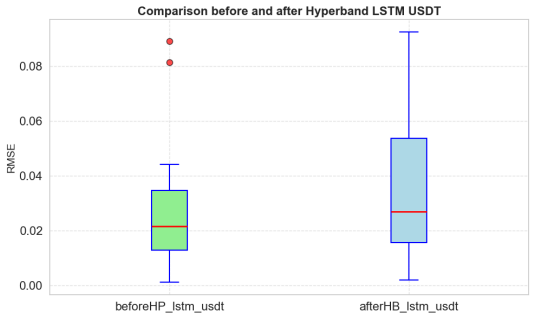


(A) Before and after Hyperband GRU USDT

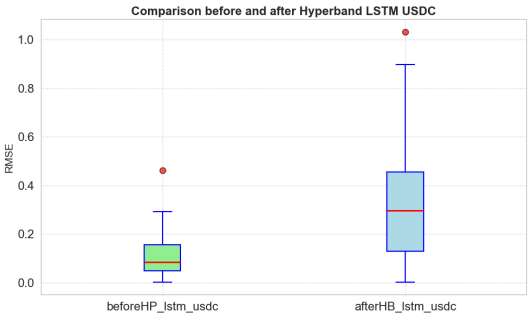
(B) Before and after Hyperband LSTM USDT

FIGURE 5.17: Hyperband effects USDT

Despite the implementation of the Hyperband optimization algorithm, it can be observed that the suggested parameter combinations consistently underperformed compared to



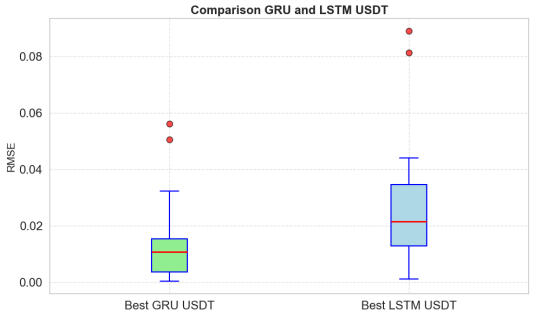
(A) Before and after Hyperband GRU USDC



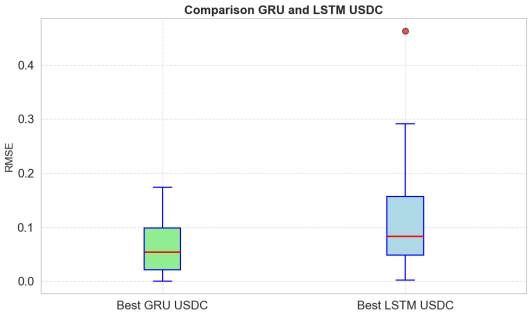
(B) Before and after Hyperband LSTM USDC

FIGURE 5.18: Hyperband effects USDC

the baseline model. In each iteration, the proposed configurations resulted in models with poorer performance, failing to improve the overall predictive accuracy. As a result, the hyperparameter suggestions generated by Hyperband were disregarded, as they did not lead to meaningful improvements. Consequently, the initial configuration consisting of 64 units, 8 neurons, and a learning rate of 0.0001 was retained for further analysis.



(A) Final comparison GRU vs LSTM USDT



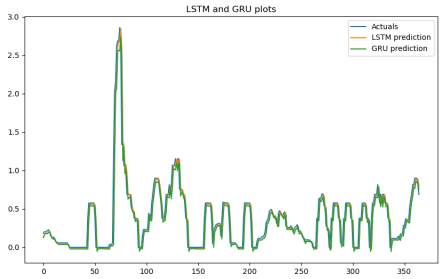
(B) Final comparison GRU vs LSTM USDC

FIGURE 5.19: Final comparisons

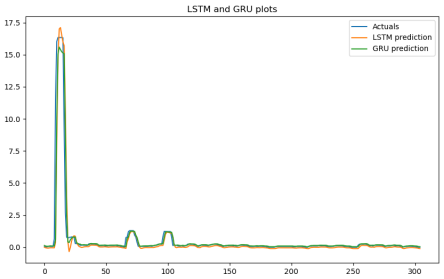
Finally, the performance of the best fitted LSTM and GRU models are compared to each other. In both datasets, USDT and USDC, GRU consistently outperforms LSTM. Both having an average lower RMSE and having more consistent results with less spread. Since, GRU consistently outperforms LSTM, for future experiments, only the results of GRU will be looked at. When comparing the predicted values with the actual values, both models are able to capture the volatility quite effectively seen in figure 5.20

5.6 Exploratory work

In this section, various methods of exploratory data analysis are implemented to gain a better understanding into the features and the correlations between them. Heatmaps,



(A) Plotting GRU and LSTM (USDT)



(B) Plotting GRU and LSTM (USDC)

FIGURE 5.20: Plot graph USDT and USDC

principal component analysis (PCA), permutation importance and drop-column method are discussed. These techniques also aim to identify potential features that could enhance the performance of the existing model.

5.6.1 Heatmap

A heat map is a visualization tool that uses color to represent the values of a matrix. This is most often used when trying to visualize large amounts of data and to identify patterns, anomalies or in this case which features have high correlations with each other. Each feature of the dataset is represented on both the x and y axes. The heat map shows which stocks or cryptocurrencies are correlated to each other based on the perceptual change over time. In each cell in the matrix, a value is visible ranging from 0 to 1, where 0 means low correlation, and 1 means high correlation. Additionally, colors are added in each cell to enhance the heat map's clarity.

When an asset is compared with itself, the price movement is naturally identical, resulting in a correlation value of 1. It is noticeable that cryptocurrencies show strong correlations with each other, as do stocks within their market, aside from a few exceptions. For cryptocurrencies, aside from the stablecoins, all other cryptocurrencies show high positive relations with each other. Whereas with stocks, aside from the DXY, S&P500 Investment grade and S&P Eurozone, all the other stocks have either a high or low correlation with each other. Stock prices seem to be high correlated with each other, and have a high negative correlation with real-market indices. When comparing correlations between cryptocurrencies and the stocks, there seem to be very little correlation.

However, when comparing the volatility of USDC and USDT, no other feature seem to have a correlation with these coins. Relatively speaking, both volatilities are correlated

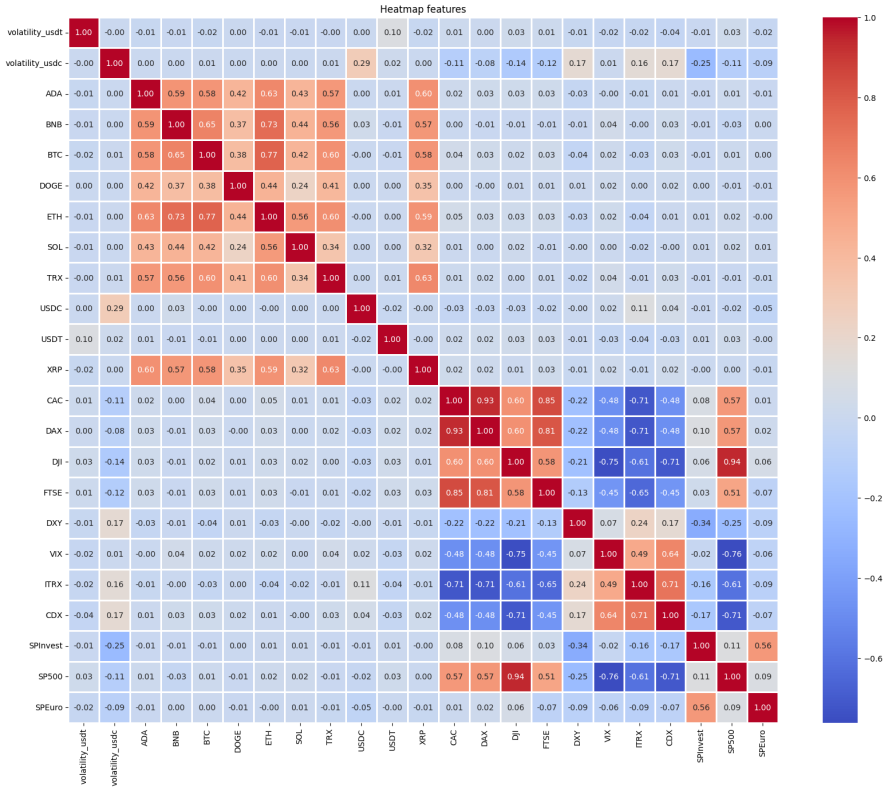


FIGURE 5.21: Heatmap stocks and cryptocurrencies

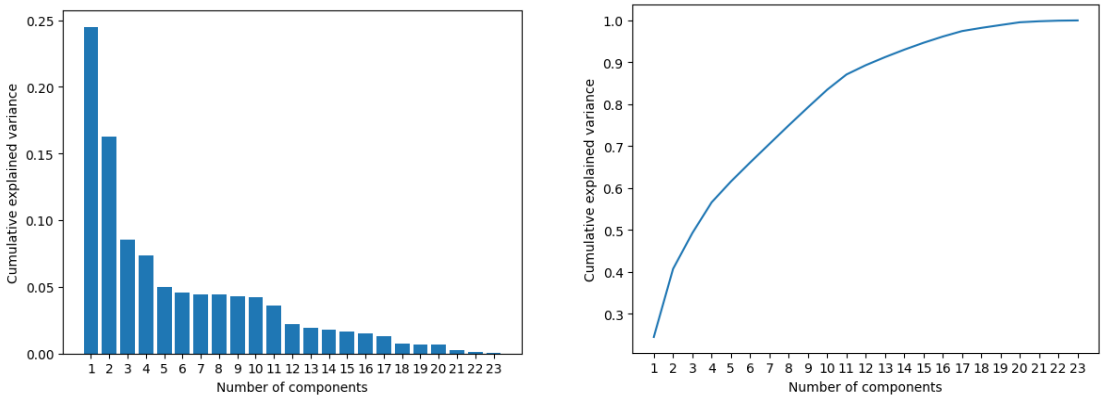
with each other as well as the price of the corresponding volatility. USDC seem to be relatively more correlated with general stocks rather than cryptocurrencies.

Visualization tools can improve our understanding of what features are important for training models. Although heat maps display the correlation of assets based on price changes, the question remains whether these combinations of assets have the same impact on model predictions. In practice, this does not always appear to be the case.

5.6.2 Principal Component Analysis

Principal Component Analysis (PCA) is a statistical technique to reduce the dimensionality of a large dataset while maintaining as much variability as feasible in the data. PCA transforms the variables into a new set of uncorrelated linear variables known as principal components. Each principal component captures a portion of the total variance in the data. For calculating the principal components, the covariance matrix of returns is first calculated and after that eigenvalue decomposition is performed to obtain the values.

In figure 6.2, the variance distribution of the 24 principal components can be seen. In these plots, the eigenvalues are scaled by their sum, so they can be understood as a



(A) Principal Components (B) Cumulative principal components

FIGURE 5.22: Principal component analysis

proportion of the data's variability explained by its corresponding principal component vector. As expected from the theory, the first principal component has the largest variance, and this gradually decreases after each component. The cumulative proportion of the variance explained is the percentage of total variance explained by the first n principal components, where n is the number of maintained components.

PC	Eigenvalue	% variance	Cum. variance
1	5.6385332	24.50%	24.50%
2	3.73763617	16.24%	40.74%
3	1.96357338	8.53%	49.27%
4	1.6856768	7.32%	56.59%
5	1.14335131	4.97%	61.56%
6	1.04516368	4.54%	66.1%
7	1.01942383	4.43%	70.53%
8	1.01642809	4.42%	74.95%
9	0.99170596	4.31%	79.26%
10	0.96920017	4.21%	83.47%

TABLE 5.3: First 10 principal components

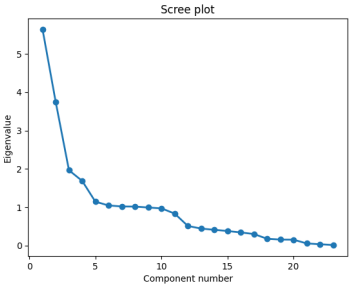


FIGURE 5.23: Scree plot of PCs

There are various methods to determine how many principal components should be taken into account. First, the Kaiser rule, which suggests taking principal components that have an eigenvalue over 1, with the idea that any component should account for at least as much as a single variable. There is a practical approach to take as many components so it will account for an X amount of total variance (commonly 70-95%). And there is a Scree Plot approach which looks for the elbow of the curve, where the elbow is the point after which the remaining eigenvalues start decreasing linearly and only the components above the elbow are retained. Using the scree plot method is difficult as there is no clear sharp drop-off. However, slight declines are noticeable when retaining 3, 5, and 11 components. The Kaiser rule recommends retaining 8 principal components, which

account for roughly 75% of the total variance. This aligns closely with the results of the practical approach.

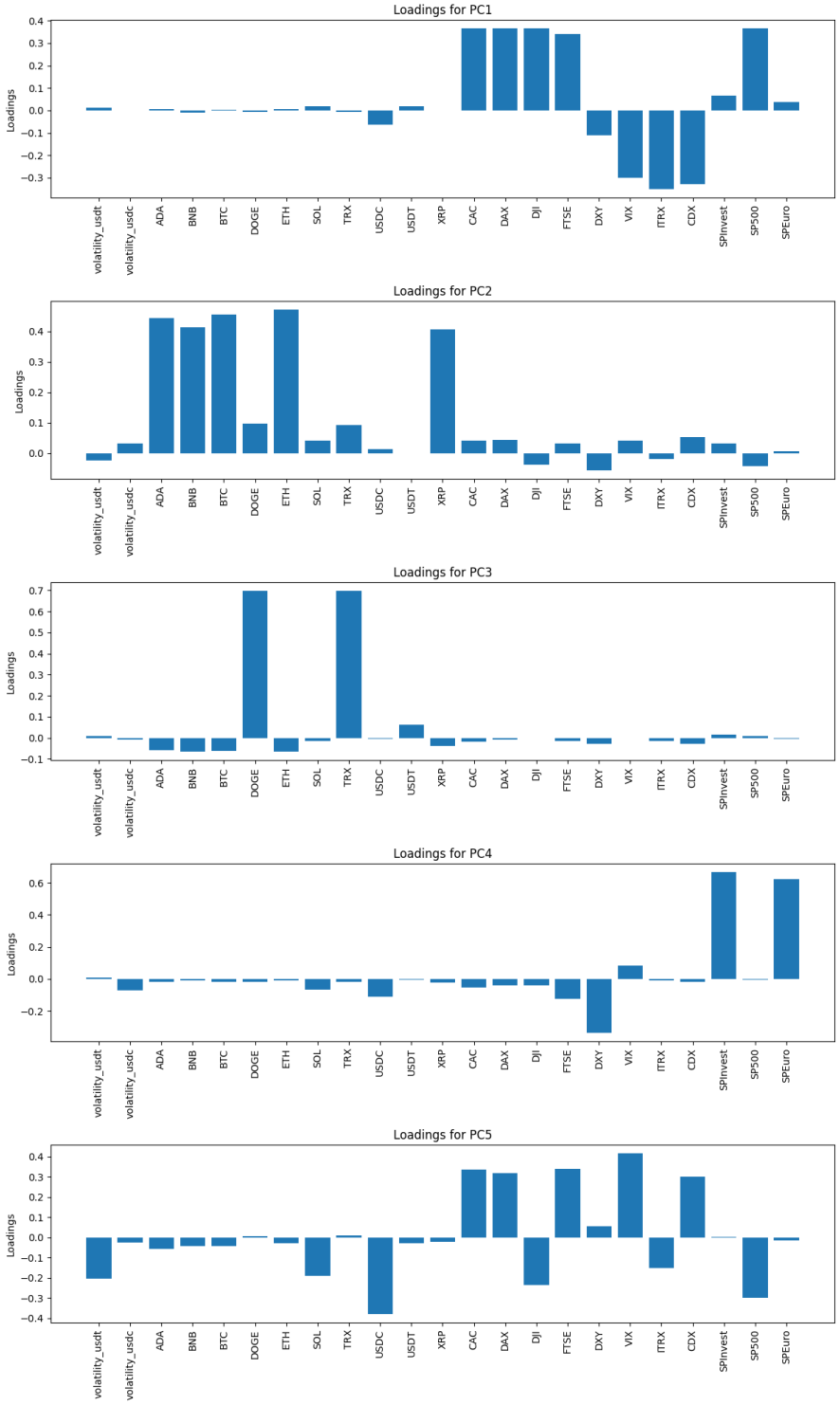


FIGURE 5.24: Principal components 1 to 5

The x-axis displays all the features of the dataset, and the y-axis displays the loadings corresponding to its feature. Each principal component captures a portion of the total variance in the data. Variables with high positive or negative loadings on a component

are the most important for that PC. In this case, general indices show strong positive and negative loadings on the first principal component (PC1), while cryptocurrencies exhibit strong positive and negative loadings on the second principal component (PC2). The idea behind this plot is that features with similar loadings in the principal components are moving similarly, and thus are correlated with each other. However, when examining the first and second principal components related to the volatility of USDT and USDC, the loadings remain very low. This is likely due to the lack of price changes within these stablecoins, therefore, contributing very little to the overall variance of the dataset.

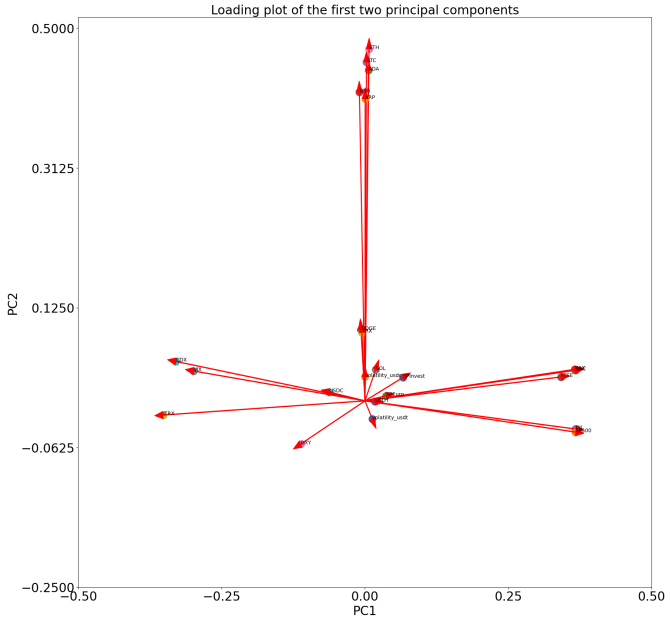


FIGURE 5.25: Scatter plot of the 1st and 2nd principal component

This is clearly illustrated in a scatter plot 5.25, where the first principal component is plotted on the x-axis and the second principal component on the y-axis. Using a scatter plot like this makes the correlations between variables more visible and easier to understand. This type of plotting captures approximately 40% of the variance. The correlations between the variables can be understood by examining the angles between the vectors. When the vectors form a 90-degree angle, the variables are likely to be uncorrelated. In contrast, a 180-degree angle indicates that the variables are negatively correlated.

The plot shows a similar pattern to the heatmap, with cryptocurrencies clustering together, index stocks forming their own group, and real-time market indices grouped separately. This plot further illustrates that, in addition to the clustering, stocks and cryptocurrencies exhibit little correlation.

In the middle, some features which have a low loadings are visible and these are also features that don't really show that much movement in value. The volatility of USDT

and USDC stands quite isolated, with no other features clustered around them, like the other clusters.

5.6.3 Permutation Feature Importance

Another method for improving the understandability of a model is called Permutation Feature Importance. Permutation Feature Importance measures the increase/decrease in the prediction error of the model after permuting the feature's values, which breaks the relationship between the feature and the true outcome. This idea comes from the paper Fisher et al..

The importance of a feature is measured by calculating the increase in the model's prediction error after permuting the feature. A feature is considered 'important' if shuffling its values increases the model error. And if shuffling its values leaves the model error unchanged, the feature would not have much effect on the model and can therefore be left out.

Permutation importance starts by calculating the baseline error of the original model. Then, for each feature in the dataset, the values within a specific column are shuffled a total of 30 times to generate modified datasets. These modified datasets are then used by the model and a RMSE is computed for each iteration. The difference between the average RMSE from the shuffled datasets and the baseline RMSE is then used to quantify the importance of the feature. Note that the importance of the permutation feature is applied only to the test set to prevent any overfitting or training a wrong model.

The goal of reshuffling the values multiple times is to reduce the impact of stochasticity and to improve the robustness of the analysis. However, despite reshuffling, the permutation scores vary considerably. To address this issue, 20 models are trained independently. The standard deviations and averages of all feature importance scores are collected. If a feature consistently ranks important across different runs with a small standard deviation, then it can be concluded that the feature has a strong and stable impact on the model's performance. However, features with high variability may have weaker influences on the model

The y-axis represents the average change in RMSE compared to the baseline in blue, and the red bar represents the standard deviation over those 20 runs. The x-axis displays the corresponding column that is shuffled during those simulations. So if the bar is below 0, shuffling the values would have a positive effect on the model. In the permutation importance analysis of USDT, the average scores, as well as the standard deviation of

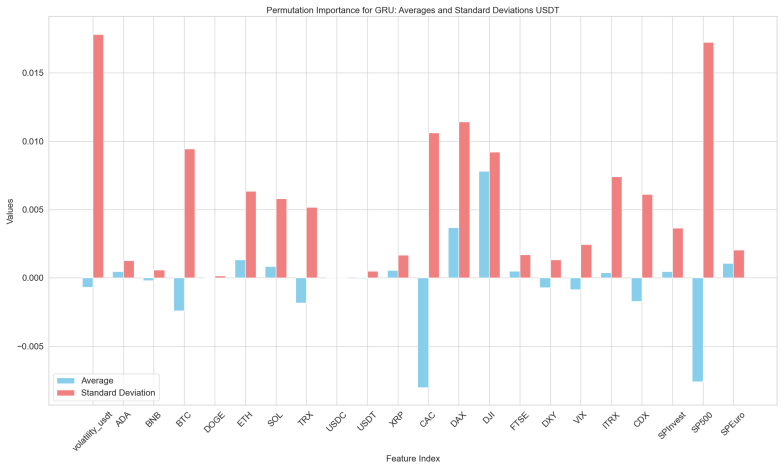


FIGURE 5.26: Permutation Importance USDT

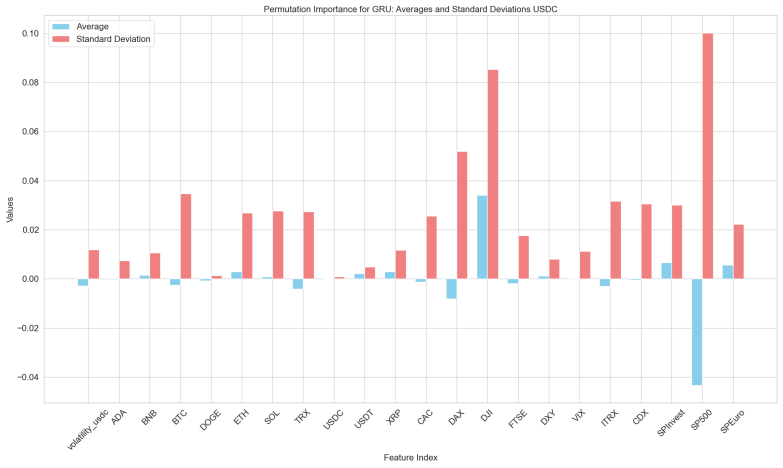


FIGURE 5.27: Permutation Importance USDC

the stablecoins, are notably low. The features that have the greatest influence on the model are CAC, DJI and SP500, while the volatility of USDT holds the highest standard deviation. However, there is no feature present that has a combination of a high average score and low standard deviation.

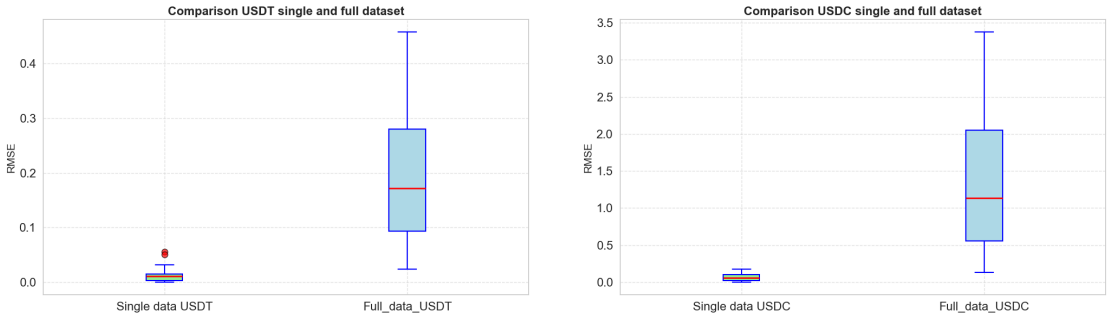
For USDC, the SP500 Index again has by far the biggest impact on the model, significantly outpacing other features in terms of performance. Similarly, the Dow Jones Industrial Average (DJI) also has strong influence on the model. Both features having also very high standard deviations. However, the rest of the features seem to not show high impact on the model.

5.6.4 Drop-column importance

An alternative method to measure the importance of individual characteristics is the drop column importance approach. This method begins by training a model on the full

dataset, which includes all features. Next, a feature is excluded, and the model is re-trained using the modified dataset. The performance of the model is then re-evaluated. If the removal of this feature leads to a significant change in performance, it suggests that the feature is likely important.

Given the high computational cost of the drop-column method, feature importance is generally preferred. However, this method is a nice way to validate the permutation scores as the importance values could slightly differ, but the general shape and order of the feature importance should be roughly the same. In these experiments, for each excluded column, a model is trained 20 times.



(A) GRU USDT single and full comparison

(B) GRU USDC single and full comparison

FIGURE 5.28: Comparison single and full dataset

Initially, the performance of the optimized GRU model is evaluated by comparing two scenarios: one where only the target variable is used as input and the other using the full dataset, which consists of 22 features mentioned previously. The results show that the model's performance is significantly worse when the full dataset is used. Despite the performance being so poor, it can still serve as a baseline for further analysis. With the drop-column method the relative performance differences can be analyzed and be compared with the permutation importance scores.

In the drop-column method, the average performance is first calculated over 20 runs for each feature. The performance of the model uses the full dataset as the baseline, with the value set to 0. For each feature that is removed, the difference in performance between the model with the dropped column and the baseline is computed and plotted. The y-axis represents the performance difference between the baseline and the dropped column in blue and the red bar stands again for the standard deviation, while the x-axis corresponds to the column that was removed. So, if the average score goes below 0, then it would suggest that removing the column from the dataset improves the model's performance.

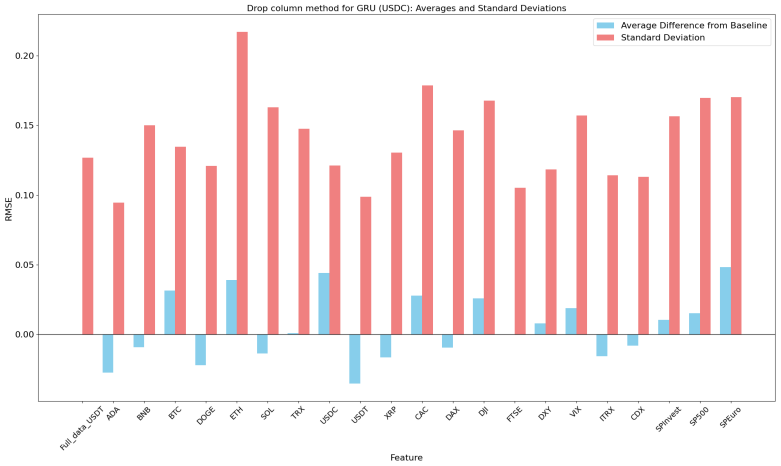


FIGURE 5.29: Drop column method USDT

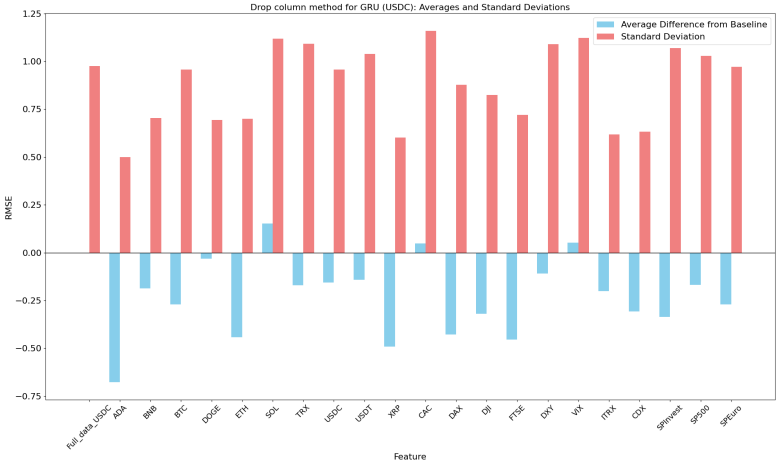


FIGURE 5.30: Drop colum method USDC

When comparing the drop-column method and the permutation feature importance, the results seem to be different. For example, in the USDT dataset for the drop-column method, there SP500 and DJI do not have big influence on the model compared to the other features. Similar with USDC, with the fact that very little similarities can be seen compared to the permutation feature importance. However, it is noticeable that when removing columns in the dataset for the USDC prediction, more often than not, the model seems to perform better.

Chapter 6

Discussion

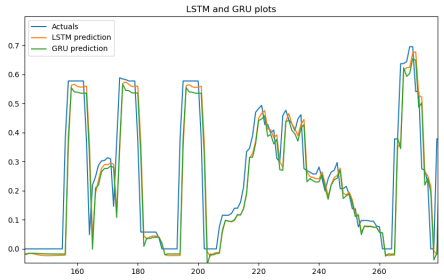
This paper looks into stablecoin volatility, specifically with regard to the two biggest stablecoins available at the moment: USDT and USDC. Both econometric and machine learning models have been implemented to predict volatility, alongside exploratory data analysis to identify factors such as other cryptocurrencies and indices that may influence the stability of these stablecoins. The analysis shows that the ARCH and GARCH models perform worse than the machine learning models, where GRU outperforms LSTM in the daily dataset used.

The reason for the poor performance of ARCH and GARCH could be due to the interpretation of volatility. Although the standard deviation of returns is widely used to calculate volatility, there exists numerous other ways to calculate volatility, each with its own formula and resulting values. Similarly, the interpretation of volatility within ARCH and GARCH are also different. Therefore, measuring the performance of ARCH and GARCH with this target volatility may not provide a fair or accurate comparison.

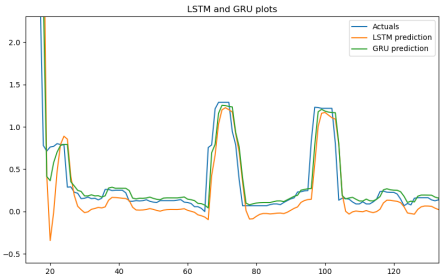
Next, the machine learning models LSTM and GRU were optimized for the daily datasets of USDT and USDC. The optimization process starts off by experimenting with various combinations of batch sizes and epochs, followed by adjustments of the window sizes, and finally hyperparameter optimization was implemented to identify the best parameters. The first two steps seem to provide notable improvements of performance. However, Hyperband was unable to identify better parameter settings.

This could be explained due to the stochastic nature of the model's predictions. During the exploratory phase of Hyperband, the algorithm evaluates the performance based on a limited amount of epochs. If these initial results are promising, the algorithm continues to evaluate these parameter settings with additional epochs. However, the performance

outcomes for each setting can fluctuate significantly with a small number of epochs, making it difficult for the algorithm to accurately rank the parameter settings.



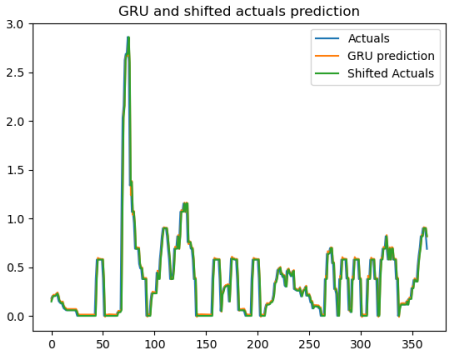
(A) Zoom USDT LSTM and GRU results



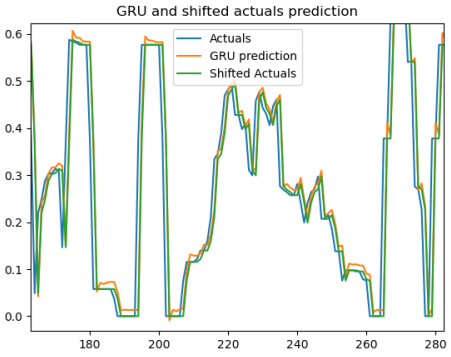
(B) Zoom USDT LSTM and GRU results

FIGURE 6.1: Zoom of ML models

Despite the low RMSE values observed in both machine learning models, when zooming in on the predictions of the model, it might suggest that the models are not accurately forecasting future values. Rather, the predictions appear to closely replicate previous values, as it seems that the plots are slightly shifted to the right. So an argument would be that it tends to mirror previous values rather than actually predicting future values.



(A) GRU and shifted values prediction



(B) Zoom USDT LSTM and GRU results

FIGURE 6.2: Comparison GRU and shifted values prediction

When comparing the GRU model's predictions to the actual values, a noticeable pattern emerges, where the predictions seem to be consistently shifted one timestep forward. The RMSE between the actual values and the test set shifted by one timestep is 0.0014807, which is very low comparing this to the measured GRU and LSTM performances. This suggests that the GRU model may not effectively forecast future values; instead, it appears to be closely mirroring or replicating previously observed volatility patterns, rather than making genuine predictions.

Next, some exploratory data analysis is conducted to improve the understanding of the dataset's features and to identify potential features that could enhance the performance of the existing model. The heatmap shows that cryptocurrencies and stock indices are correlated separately. When comparing stock index prices and real-market indices, it seems to be that there is a negative correlation between these two types of stocks. This relationship can be theoretically justified. Real-market indices measure the volatility of the current market and provide an indication of the likelihood of default, where a high value reflects either increased volatility or greater chance of default. Conversely, when the market performs well, which can be reflected by the increase of stock index prices, there is generally less panic, where investors feel more secure reducing the likelihood of selling off stocks. And with more money earned, the likelihood of defaulting also goes down.

The volatility and value of USDT and USDC show very minimal correlation with other features in the dataset. Only USDC shows a slight correlation with stock indices, rather than with other cryptocurrencies. The low correlation is likely due to the low rate of change in the values of these features. This idea is supported by the results of the principal component analysis (PCA), which was done to reduce the dimensionality of the dataset by transforming the variables into principal components. In the PCA, cryptocurrencies had similar loadings among themselves, same for stock indices.

The scatterplot from the PCA analysis illustrates groupings of features that are highly correlated with each other, while it also highlights the negative correlation between real-market indices and stock index prices. Some features are clustered in the center of the plot because of their low loadings. This is likely because these features have relatively low price movements. For example, stablecoins are known to show very small price fluctuations. Similarly, S&P Eurozone Sovereign Bond Index (SPEuro) and the S&P 500 Corporate Bond Index (SPInvest) show price changes of no more than 20% over the past five years. Additionally, cryptocurrencies such as SOL, TRX, and DOGE, although they have shown big price changes like other cryptocurrencies, were only introduced to the dataset at a later stage, resulting in values of zero in early data. In contrast to other features, all their data almost complete in the data and easily exceed changes over the 100% past 5 years.

First, using more features does not improve the model compared when only the target variable is used as input. However, the dynamics within this full dataset can be analyzed to gain more insight per individual feature. Both permutation feature importance and drop-column methods are implemented based on the performance of GRU. The theory suggests that both these methods should produce similar results; however, this does not seem to be the case. This is likely do to the high stochasticity of the model, where, for

example, removing one column out of the 22 features or shuffling the values does not have a big enough effect on the model to see significant differences.

Chapter 7

Conclusion and future work

This research investigated both econometric models and machine learning models to predict the volatility movement of the stablecoins: USDT and USDC. Based on the RMSE of the implemented models, it can be concluded that the econometric models perform worse than the machine learning models.

Econometric models demonstrate a reasonable level of prediction, where it shows the volatility peaking when returns are high. However, when compared to the target volatility, which is calculated by the standard deviation of returns over the past 7 time steps, these models struggle to closely replicate the precise fluctuations of the observed volatility. Despite the attempts to improve the performance through different parameter configurations for ARCH and GARCH, the models' performance remain quite poor compared to LSTM and GRU.

The interpretation and consequently the values of volatility can vary largely. Therefore, comparing the volatility derived from this method with that obtained from ARCH and GARCH models may not yield a fair or meaningful comparison. To address this issue, it can be advised to reconsider the volatility calculation formula or solve algebraically what the differences exactly are. Machine learning models are not affected by this, since they are designed to optimize performance based on their specific target variable.

Both the machine learning models LSTM and GRU are optimized for the volatility prediction of USDT and USDC. Initially, different combinations of batch sizes and epochs were tested for each model. The GRU model demonstrated improved performance with smaller batch sizes and epochs, but that trend was less apparent in the LSTM model. After that, the optimal window size was determined, though no clear pattern was visible with the effect of smaller or larger window sizes. Finally, hyperparameter optimization using Hyperband was applied to further improve the models, but this did not result

in significant improvements. When both models are compared to each other, GRU consistently outperforms LSTM in both datasets.

Next, some exploratory data analysis is implemented to improve the understanding of the features within the dataset and aiming to identify potential features that could enhance the performance of the existing model.

The heatmap reveals a clear distinction in correlations between cryptocurrencies and stocks. Cryptocurrencies exhibit strong correlations among themselves, and similarly, stocks are highly correlated with each other. However, when comparing the correlations between cryptocurrencies and stocks, there is very low correlation. When looking at either the volatility or value of USDT and USDC, there seems to be low correlation with all other features, only USDC seeming to have a larger correlation with general stocks compared to cryptocurrencies.

Next, principal component analysis is implemented to further reduce the dimensionality of the data set by transforming variables into principal components and analyzing the loadings of these. Similar to the heatmap, cryptocurrencies exhibit strong correlations among themselves by having similar loadings, similarly where stocks are highly correlated with each other, but between cryptocurrencies and stock there seems to be little similarities. Also, noticeable is that some features like USDT

Future research directions could include using more data for analysis. Currently, the dataset operates on a daily basis, meaning looking at data every 24 hours. Investigating the model performance with smaller intervals, such as 12 hours or smaller, could drastically large the training set. Together with the fact that CryptoCompare API has many other valaube available data such as greed indices and social media influences. By looking at more data, the results might reveal changes in the boxplot results as well as impact the other methodologies such as PCA and permutation importance.

Moreover, increasing the number of simulations used in boxplot analyses and other statistical tests are recommended. In this study, only 20 to 30 simulations were conducted per box plot due to the limited computational resources. However, a higher number of simulations could show more reliable result. This consideration also extends to the hyperparameter optimization part, where implementing and exploring alternative parameter search may find parameter combinations that would improve the model unlike Hyperband.

Finally, exploring different models is advisable. This research only looked at a single layered LSTM or GRU. However, in recent studies, promising results have been shown when multilayered models are implemented as well as hybrid approaches where LSTM and GRU are combined. And also recently, transformers, which are basically successors

of LSTM and GRU and is the main idea behind GPT, might also provide interesting predictions.

Chapter 8

Ethics and Data Management

This paper focuses on the prediction of volatility of stablecoins, aiming to provide more understanding in the volatility of stable coins. One could theoretically change the target variable to something else like stocks or other cryptocurrencies and repeat the research steps for investment purposes. However, it is crucial that this paper should in any means not be taken as financial advice. As seen in the discussion, the prediction of the models might not align with real-world outcomes. Therefore, reliance on these models should be approached with caution. No one should fully trust a model, but only use them as tools to aid your own decision-making. As George Box said in 1976: "All models are wrong, but some are useful".

I affirm that this thesis adheres to the ethical code (https://ivi.uva.nl/research/ethical-code/ethical-code.html) and research data management policies (https://rdm.uva.nl/en) of the University of Amsterdam and the Informatics Institute, and respects the principles of academic integrity.

Chapter 9

Appendix

9.1 ARCH/GARCH plots

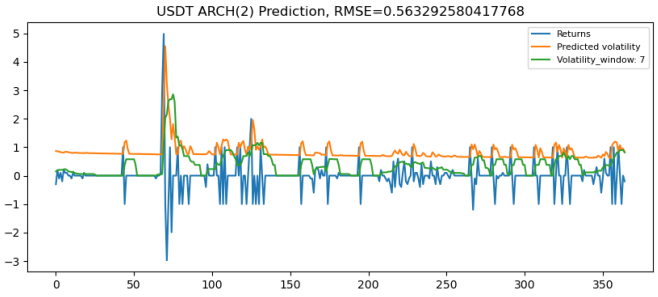


FIGURE 9.1: Summary of ARCH(2) USDT

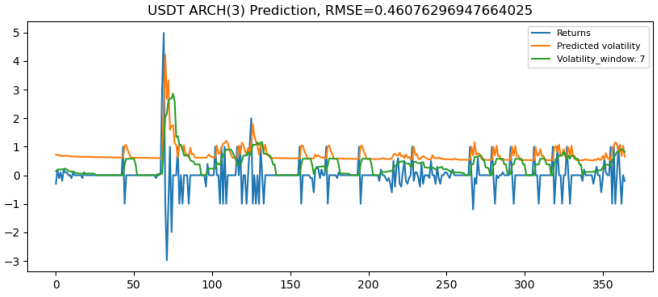


FIGURE 9.2: Summary of ARCH(3) USDT

The overall average drops when more time lags are taken into account in the ARCH models for USDT.

ARCH (p) USDT	RMSE
1	0.9311727765025717
2	0.563292580417768
3	0.46076296947664025
4	0.44363065709582056
5	0.3506649733478931
6	0.33934290512361115
7	0.3273111629275738
8	0.34150775313063564
9	0.35867486840758817
10	0.3586712104851082
11	0.3857032095966633
12	0.3871817290956796
13	0.3868907382840634
14	0.40194087176030674

Table 9.1: Sensitivity analysis results ARCH(p) USDT

ARCH (p) USDC	RMSE
1	7.546114086225023
2	7.314951398474967
3	6.943395974488551
4	6.936911820844786
5	6.93688939400027
6	6.505236564082667
7	5.845639783221598
8	6.10566604016857
9	6.033253429786215
10	2.2514700686123494
11	2.6471705042328884
12	2.7626499712342483
13	7857.592644498782
14	2.3744449299059323

Table 9.2: Sensitivity analysis results ARCH(p) USDC

GARCH (p, q) USDT	RMSE
1, 1	0.36225115245007616
2, 2	0.3235505361672609
3, 3	0.3837861372751236
4, 4	0.3296492258115549
5, 5	0.38126488577036904
6, 6	0.3470733713796538
7, 7	0.3456248942856468
8, 8	0.3704862785262342
9, 9	0.37266380976752034
10, 10	0.4249808850067281
11, 11	0.4248860273784137
12, 12	0.4257874194941187
13, 13	0.4114591453245924
14, 14	0.39094365476984605

TABLE 9.3: Sensitivity analysis results GARCH(p, q) USDT

GARCH (p, q) USDC	RMSE
1, 1	1.335294712354964
2, 2	1.8379467936099765
3, 3	291.8071517404907
4, 4	1.670688193760798
5, 5	3.060326176905211
6, 6	2.3512641950349726
7, 7	1.8462440648512586
8, 8	2.052659378320812
9, 9	344.79996863957246
10, 10	2.3810171471241164
11, 11	2.413098947328761
12, 12	2.2174964046550003
13, 13	5.650739247114772
14, 14	2.912234597127772

TABLE 9.4: Sensitivity analysis results GARCH(p, q) USDC

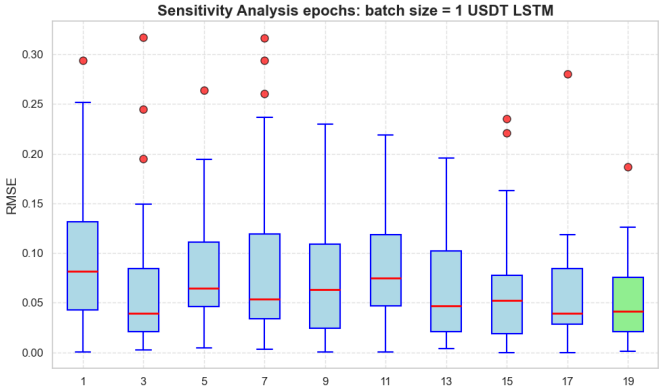


FIGURE 9.3: LSTM batch size 1 USDT

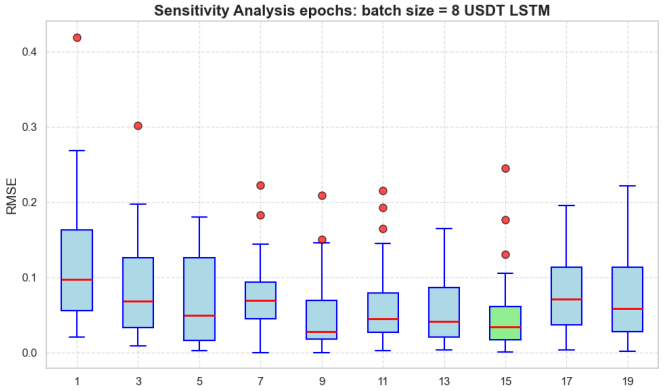


FIGURE 9.4: LSTM batch size 1 USDT

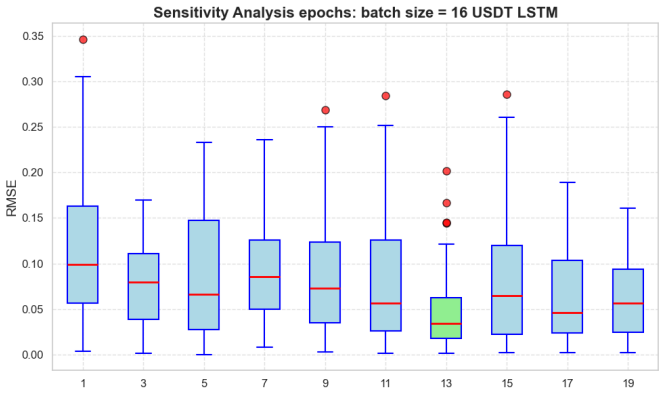


FIGURE 9.5: LSTM batch size 1 USDT

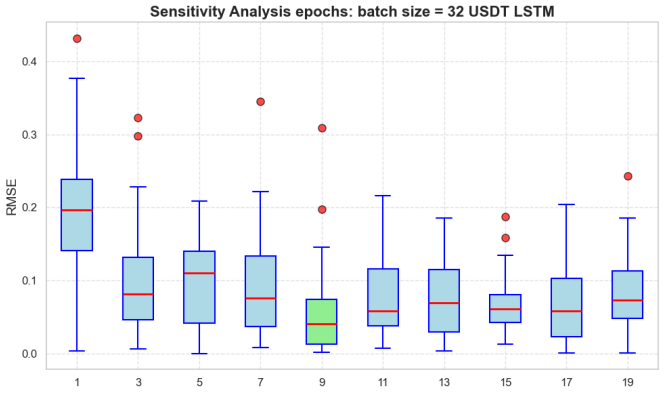


FIGURE 9.6: LSTM batch size 1 USDT

9.2 Sensitivity analysis epochs

9.2.1 USDT

9.2.1.1 LSTM

9.2.1.2 GRU

9.2.2 USDC

9.2.2.1 LSTM

9.2.2.2 GRU

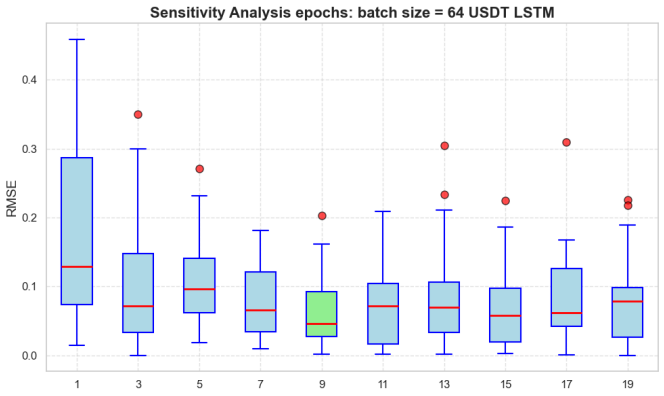


FIGURE 9.7: LSTM batch size 1 USDT

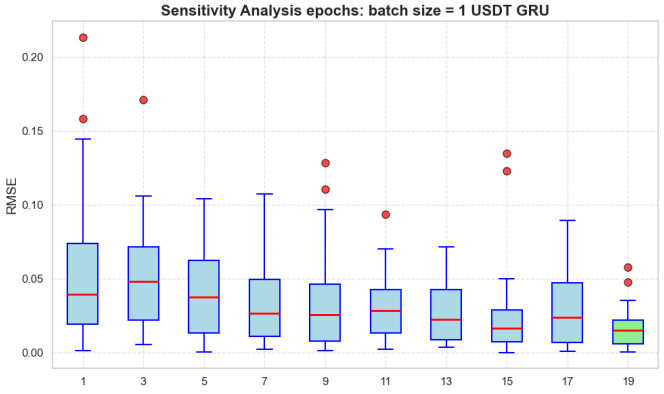


FIGURE 9.8: GRU batch size 1 USDT

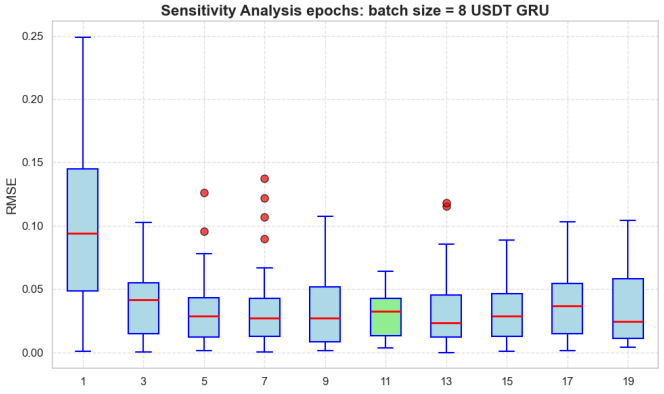


FIGURE 9.9: GRU batch size 8 USDT

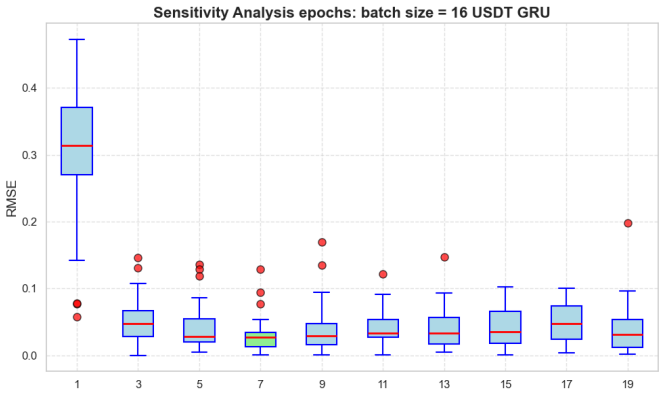


FIGURE 9.10: GRU batch size 16 USDT

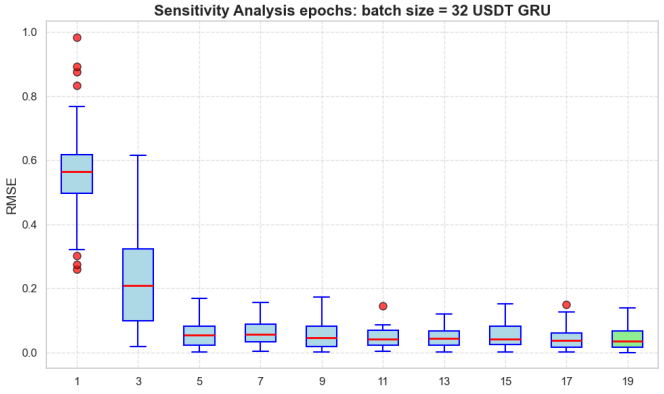


FIGURE 9.11: GRU batch size 32 USDT

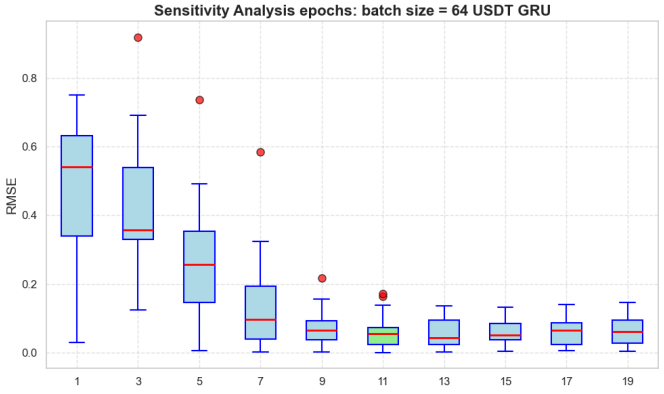


FIGURE 9.12: GRU batch size 64 USDT

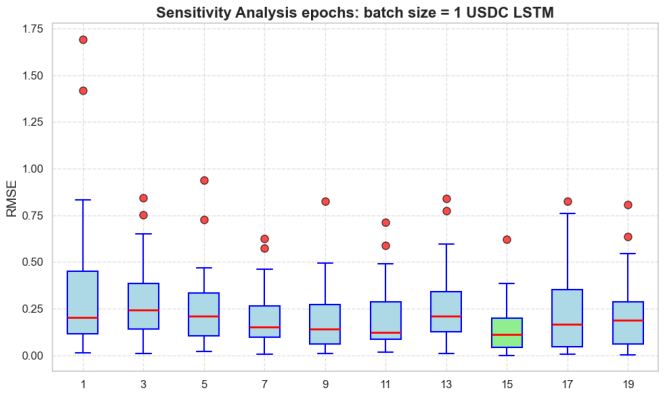


FIGURE 9.13: LSTM batch size 1 USDT

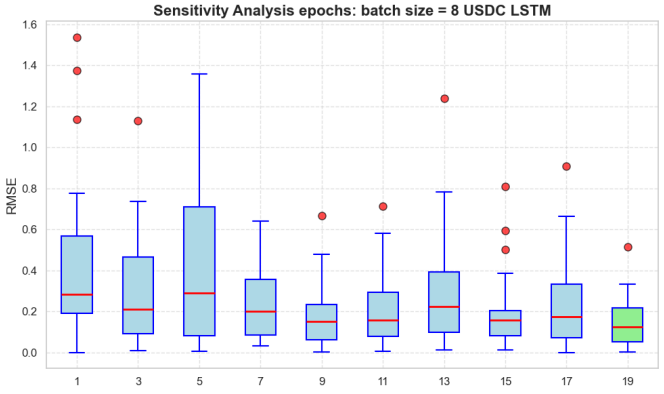


FIGURE 9.14: LSTM batch size 1 USDT

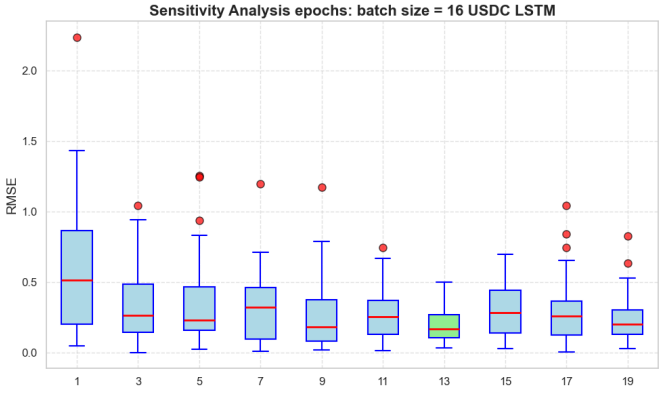


FIGURE 9.15: LSTM batch size 1 USDT

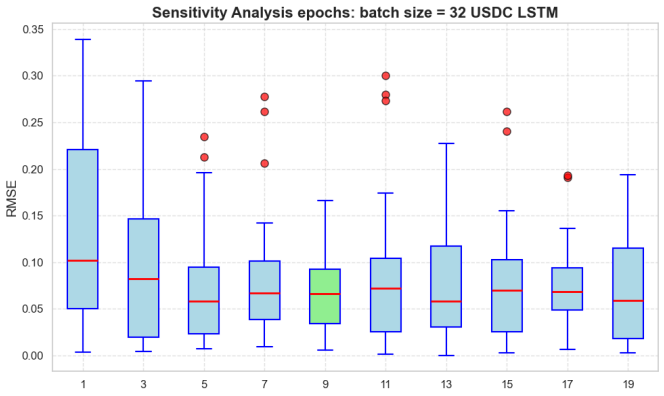


FIGURE 9.16: LSTM batch size 1 USDT

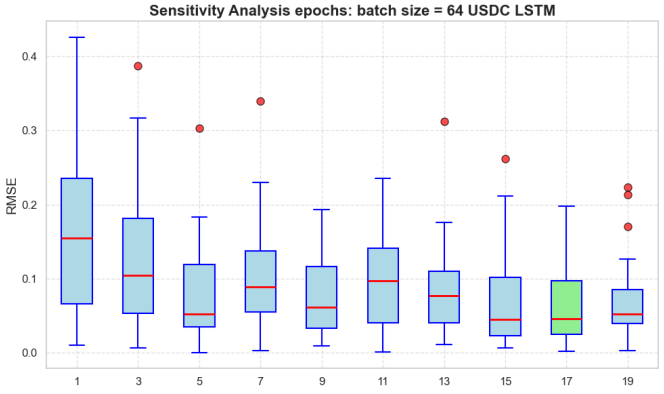


FIGURE 9.17: LSTM batch size 1 USDT

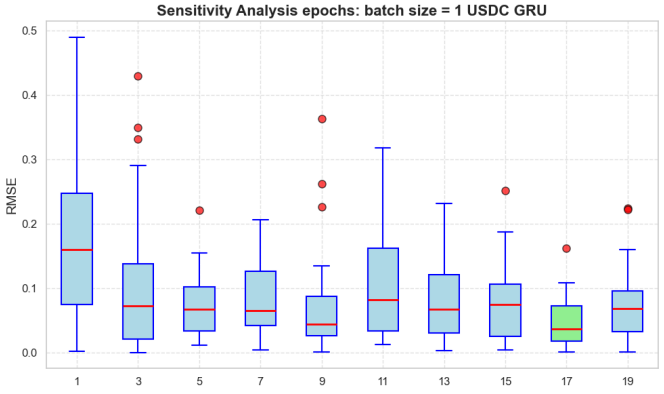


FIGURE 9.18: GRU batch size 1 USDT

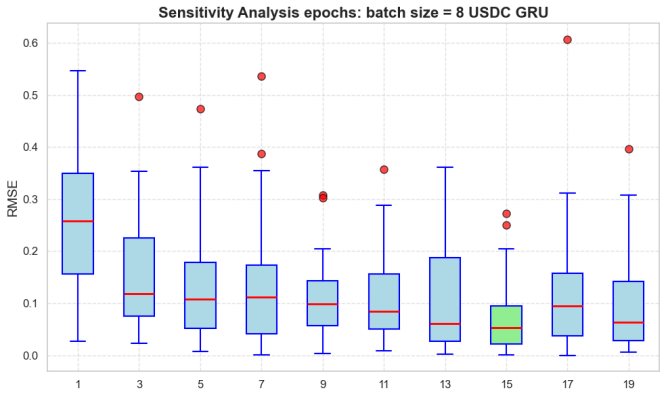


FIGURE 9.19: GRU batch size 8 USDT

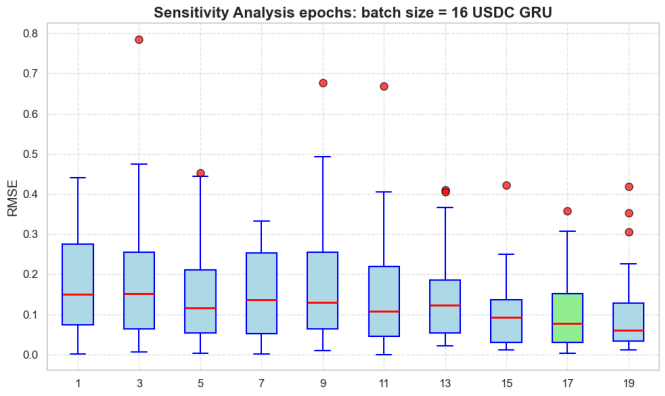


FIGURE 9.20: GRU batch size 16 USDT

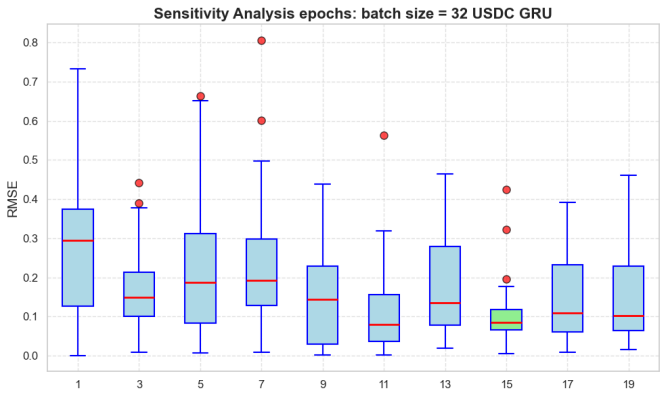


FIGURE 9.21: GRU batch size 32 USDT

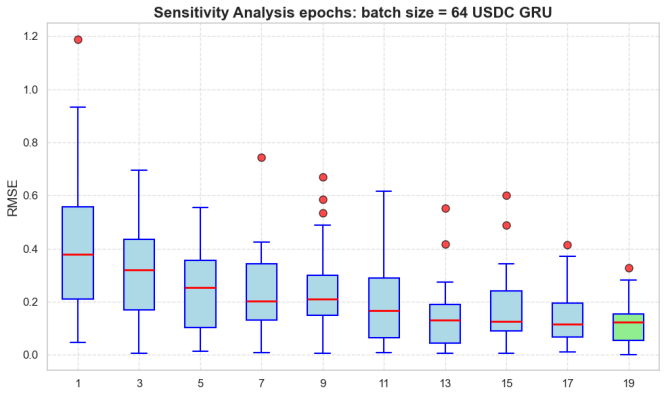


FIGURE 9.22: GRU batch size 64 USDT

Bibliography

[1] National Institute of Standards and Technology. NIST interagency/internal report (nistir) 8202: Blockchain technology overview. Technical report, National Institute of Standards and Technology, 2018. URL https://nvlpubs.nist.gov/nistpubs/ir/2018/nist.ir.8202.pdf.

[2] Rukshan Pramoditha. Neural network schematic. https://medium.com/data-science-365/overview-of-a-neural-networks-learning-process-61690a502fa, 1 Feb 2022. Accessed: May 23, 2024.

[3] Oluwasegun Bewaji, Sania Hamid, Timothy Aerts, Ronald Heijmans Shaun Byck, and Ellen van der Woerd. Are cryptocurrencies cryptic or a source of arbitrage? a genetic algorithm approach. *Payments Canada*, 2024. doi: 10.21314/JFMI.2023.006.

[4] De Nederlandsche Bank. Crypto-assets: evolution and policy response. 2021. URL https://www.dnb.nl/media/o10fpkev/dnb-occasional-study-crypto-s.pdf.

[5] Tong Fang, Zhi Su, and Libo Yin. Economic fundamentals or investor perceptions? the role of uncertainty in predicting long-term cryptocurrency volatility. *International Review of Financial Analysis*, 71:101566, 2020. ISSN 1057-5219. doi: https://doi.org/10.1016/j.irfa.2020.101566. URL https://www.sciencedirect.com/science/article/pii/S1057521920302106.

[6] Thomas Walther, Tony Klein, and Elie Bouri. Exogenous drivers of bitcoin and cryptocurrency volatility – a mixed data sampling approach to forecasting. *Journal of International Financial Markets, Institutions and Money*, 63:101133, 2019. ISSN 1042-4431. doi: https://doi.org/10.1016/j.intfin.2019.101133. URL https://www.sciencedirect.com/science/article/pii/S1042443119302446.

[7] Xiao Li Wei Zhang, Pengfei Wang and Dehua Shen. Some stylized facts of the cryptocurrency market. *Applied Economics*, 50(55):5950–5965, 2018. doi: 10.1080/00036846.2018.1488076. URL https://doi.org/10.1080/00036846.2018.1488076.

[8] Feng Dong, Zhiwei Xu, and Yu Zhang. Bubbly bitcoin. *Economic Theory*, 74(3): 973–1015, 2022.

[9] Nikolaos Kyriazis, Stephanos Papadamou, and Shaen Corbet. A systematic review of the bubble dynamics of cryptocurrency prices. *Research in International Business and Finance*, 54:101254, 2020. ISSN 0275-5319. doi: https://doi.org/10.1016/j.ribaf.2020.101254. URL https://www.sciencedirect.com/science/article/pii/S0275531919310037.

[10] Leopoldo Catania, Stefano Grassi, and Francesco Ravazzolo. Forecasting cryptocurrencies under model and parameter instability. *International Journal of Forecasting*, 35(2):485–501, 2019. ISSN 0169-2070. doi: https://doi.org/10.1016/j.ijforecast.2018.09.005. URL https://www.sciencedirect.com/science/article/pii/S0169207018301584.

[11] Christian Conrad, Anessa Custovic, and Eric Ghysels. Long- and short-term cryptocurrency volatility components: A garch-midas analysis. *Journal of Risk and Financial Management*, 11(2), 2018. ISSN 1911-8074. doi: 10.3390/jrfm11020023. URL https://www.mdpi.com/1911-8074/11/2/23.

[12] Mamoona Zahid, Farhat Iqbal, and Dimitrios Koutmos. Forecasting bitcoin volatility using hybrid garch models with machine learning. *Risks*, 10(12), 2022. ISSN 2227-9091. doi: 10.3390/risks10120237. URL https://www.mdpi.com/2227-9091/10/12/237.

[13] Zhuorui Zhang, Hong-Ning Dai, Junhao Zhou, Subrota Kumar Mondal, Miguel Martínez García, and Hao Wang. Forecasting cryptocurrency price using convolutional neural networks with weighted and attentive memory channels. *Expert Systems with Applications*, 183:115378, 2021. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2021.115378. URL https://www.sciencedirect.com/science/article/pii/S0957417421008046.

[14] Ahmed Bouteska, Mohammad Zoynul Abedin, Petr Hajek, and Kunpeng Yuan. Cryptocurrency price forecasting – a comparative analysis of ensemble learning and deep learning methods. *International Review of Financial Analysis*, 92:103055, 2024. ISSN 1057-5219. doi: https://doi.org/10.1016/j.irfa.2023.103055. URL https://www.sciencedirect.com/science/article/pii/S1057521923005719.

[15] Douglas Arner, Raphael Auer, and Jon Frost. Stablecoins: risks, potential and regulation. BIS Working Papers 905, Monetary and Economic Department, Bank for International Settlements, November 2020. URL https://www.bis.org/publ/work905.htm.

[16] Anneke Kosse, Marc Glowka, Ilaria Mattei, and Tara Rice. Will the real stablecoin please stand up? BIS Papers 141, Monetary and Economic Department, Bank for International Settlements, November 2023. URL https://www.bis.org/publ/bispap141.htm. JEL classification: E42, E58, G23, G28, O33.

[17] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *https://bitcoin.org/bitcoin.pdf*, 2008.

[18] Cpicalc. https://www.in2013dollars.com/us/inflation. Accessed: April 10, 2024.

[19] fiatcur. https://allesovercrypto.nl/blog/fiatgeld-eenvoudige-uitleg. Accessed: April 10, 2024.

[20] Ruchi Gupta, Mandeep Gupta, and Deepanshu Gupta. Role of liquidity pool in stabilizing value of token. *Scientific Journal of Metaverse and Blockchain Technologies*, 1(1):9–17, Dec. 2023. doi: 10.36676/sjmbt.v1i1.02. URL https://sjmbt.com/index.php/j/article/view/2.

[21] Weiqin Zou, David Lo, Pavneet Singh Kochhar, Xuan-Bach Dinh Le, Xin Xia, Yang Feng, Zhenyu Chen, and Baowen Xu. Smart contract development: Challenges and opportunities. *IEEE Transactions on Software Engineering*, 47(10):2084–2106, 2021. doi: 10.1109/TSE.2019.2942301.

[22] Vijay Mohan. Automated market makers and decentralized exchanges: a defi primer. *Financial Innovation*, 8(1):20, 2022. ISSN 2199-4730. doi: 10.1186/s40854-021-00314-5. URL https://doi.org/10.1186/s40854-021-00314-5.

[23] Jiahua Xu and Yebo Feng. Reap the harvest on blockchain: A survey of yield farming protocols. *IEEE Transactions on Network and Service Management*, 20 (1):858–869, 2023. doi: 10.1109/TNSM.2022.3222815.

[24] CryptoCompare. Cccagg index methodology. *CC Data Limited*, 2021.

[25] Guide to speculative investments. https://www.sofi.com/learn/content/speculative-investment/.

[26] Garth Baughman, Francesca Carapella, Jacob Gerszten, and David Mills. The stable in stablecoins. FEDS Notes, December 16 2022. URL https://doi.org/10.17016/2380-7172.3224.

[27] Mural. Usdc vs usdt: What is the difference?, 2024. URL https://www.muralpay.com/blog/usdc-vs-usdt-what-is-the-difference. Accessed: 2024-07-17.

[28] Moonpay. Usdt vs usdc: A stablecoin comparison, 2024. URL https://www.moonpay.com/learn/cryptocurrency/what-is-the-difference-between-usdt-and-usdc. Accessed: 2024-07-17.

[29] Changlin Wang. Different garch models analysis of returns and volatility in bitcoin. *Data Science in Finance and Economics*, 1(1):37–59, 2021.

[30] Bashar Yaser Almansour, Muneer M Alshater, and Ammar Yaser Almansour. Performance of arch and garch models in forecasting cryptocurrency market volatility. *Industrial Engineering & Management Systems*, 20(2):130–139, 2021.

[31] Robert F. Engle. Autoregressive conditional heteroscedasticity with estimates of the variance of united kingdom inflation. *Econometrica*, 50(4):987–1007, 1982. ISSN 00129682, 14680262. URL http://www.jstor.org/stable/1912773.

[32] Robert H. Shumway and David S. Stoffer. *Time Series Analysis and Its Applications: With R Examples*. Springer, 2017. ISBN 978-3319524511.

[33] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994. doi: 10.1109/72.279181.

[34] Deep learning: Introduction to long short-term memory, 2024. URL https://www.geeksforgeeks.org/deep-learning-introduction-to-long-short-term-memory/. Accessed: 2024-07-30.

[35] Simeon Kostadinov. Understanding gru networks, 2024. URL https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be. Accessed: 2024-07-30.

[36] Unlocking the power of gated recurrent unit (gru): Understanding the innovative architecture, 2024. URL https://medium.com/@sachinsoni600517/unlocking-the-power-of-gated-recurrent-unit-gru-understanding-the-innovative-arch Accessed: 2024-07-30.

[37] Gated recurrent unit networks, 2024. URL https://www.geeksforgeeks.org/gated-recurrent-unit-networks/. Accessed: 2024-07-30.

[38] AD Dongare, RR Kharde, Amit D Kachare, et al. Introduction to artificial neural network. *International Journal of Engineering and Innovative Technology (IJEIT)*, 2(1):189–194, 2012.

[39] Christopher Olah. Understanding lstm networks. https://colah.github.io/posts/2015-08-Understanding-LSTMs/, August 27, 2015. Accessed: May 20, 2024.

[40] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014. URL https://arxiv.org/abs/1406.1078.

[41] I Sibel Kervanci, M Fatih Akay, and Eren Özceylan. Bitcoin price prediction using lstm, gru and hybrid lstm-gru with bayesian optimization, random search, and grid search for the next days. *Journal of Industrial and Management Optimization*, 2023.

[42] Phumudzo Lloyd Seabe, Claude Rodrigue Bambe Moutsinga, and Edson Pindza. Forecasting cryptocurrency prices using lstm, gru, and bi-directional lstm: a deep learning approach. *Fractal and Fractional*, 7(2):203, 2023.

[43] Nurhayati Buslim, Imam Lutfi Rahmatullah, Bayu Aji Setyawan, and Aryajaya Alamsyah. Comparing bitcoin's prediction model using gru, rnn, and lstm by hyperparameter optimization grid search and random search. In *2021 9th International Conference on Cyber and IT Service Management (CITSM)*, pages 1–6. IEEE, 2021.

[44] Bilal Hassan Ahmed Khattak, Imran Shafi, Chaudhary Hamza Rashid, Mejdl Safran, Sultan Alfarhood, and Imran Ashraf. Profitability trend prediction in crypto financial markets using fibonacci technical indicator and hybrid cnn model. *Journal of Big Data*, 11(1):58, 2024.

[45] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(185):1–52, 2018.

[46] Aaron Fisher, Cynthia Rudin, and Francesca Dominici. All models are wrong, but many are useful: Learning a variable's importance by studying an entire class of prediction models simultaneously, 2019. URL https://arxiv.org/abs/1801.01489.