

Exploring a range of Self-Supervised Learning applications for Time Series Cryptocurrency Forecasting

Calum Murphy

School of Computing Science, Newcastle University, UK

Abstract

This paper aims to explore a range of Self-Supervised Learning applications specifically applied to Time Series Bitcoin market data. Presented within are several novel solutions that aim to highlight vastly different approaches to forecasting time-series financial data. Through this exploration, the paper aims to highlight the current abilities of several model paradigms, how their results compare to one another, and how they might be combined to form an ensemble model for superior performance.

Keywords: Self-supervised learning, Time Series Data, Forecasting, Deep Learning, Finance, Cryptocurrency, Bitcoin,

1 Introduction

This section aims to give an overview of the paper's structure, familiarise the reader with the context of the research, discuss obstacles that impeded the research, and finally summarise the objectives the research attempts to meet.

1.1 Context

Self-supervised Learning(SSL) is one of the fastest growing Machine Learning Paradigms, with an expected compound annual growth rate of 33.4% until at least 2030, being valued at \$1.8 Billion in 2021[1]. In spite of this rapid growth, however, it has seen a relative lack of application in forecasting Financial Asset Data - namely cryptocurrency. Indeed, there has been research in applying SSL to cryptocurrency markets, but a large proportion relates to anomaly detection[2] [3][4], rather than forecasting tasks. As a result, this paper seeks to explore what is currently an underdeveloped area.

¹ Email: c0036993@ncl.ac.uk

1.2 Obstacles

Generally in Machine Learning problems, a key issue is the sourcing of satisfactory amounts of high-quality data. However, in the case of Bitcoin (or any other stock, cryptocurrency or other security) price data, this is extremely simple to obtain, due to the asset being publicly traded. In spite of this, there was a major obstacle in this particular research area - ensuring the research is novel.

Utilising simple machine learning models to obtain predictions of cryptocurrency price data has been a popular topic even on social media for many years, garnering hundreds of thousands of views[5][6]. In order to ensure this research is unique, it must therefore employ a variety of different methods, contrasting and combining these methods to obtain high-quality results.

1.3 Objectives

Keeping in mind the context and obstacles present in this field of study, the objectives of the project were therefore:

- Explore a variety of machine learning methods applied to the problem of cryptocurrency forecasting
- Combine these methods to form a more efficient "ensemble models"
- Analyse and interpret the results obtained through rigorous experimentation

2 Planning

This section gives an overview of the Planning phase. This encompasses development methodologies chosen (and how they differed from previous projects), and the final plan itself.

2.1 Development Model

This project had an unusually short time-frame of four weeks enforced upon it. As such, every second was exceedingly valuable, and any wasted time was potentially detrimental to the entire project. While many development methodologies could be considered, there was a very clear choice.

In the previous project, Waterfall was chosen as a result of unfamiliarity with the subject matter and time allowed for the project. With these constraints, waterfall allowed for a steady stream of gradual progress. However, in this case, the opposite was true in both cases. As such, Agile, the favoured tool of the modern software project manager[7] was necessary.

Agile consists of several "sprints", favouring iterative progress on all aspects of development, rather than explicitly defined phases of development, in the case of waterfall[8]. Additionally, as the project only had two hard deadlines, both on the same day, the singular main benefit of Waterfall's rigid deadlines would have been largely pointless. With the project's small timescale and the alacrity they would need to be completed with, development phases would naturally overlap, and as such, the flexibility offered by Agile was effectively a necessity, and these overlaps could be contained effectively within each individual sprint.

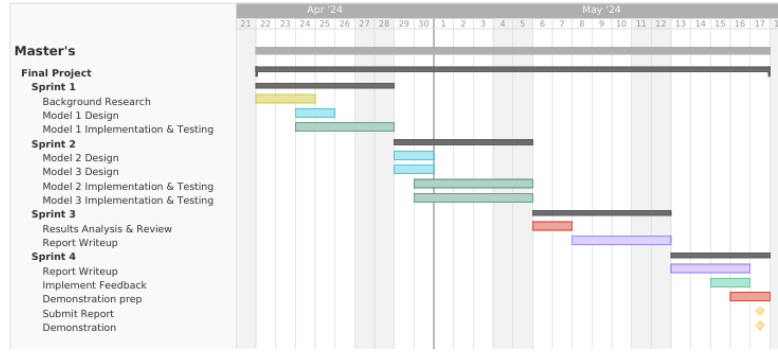


Fig. 1. Project Gantt Chart

2.2 Project Plan

With Agile confirmed as the development methodology of choice, a Gantt chart was produced, detailing one sprint for each week of the project, and the requisite tasks in each week. In addition to this, the end of the project span on the chart has markers indicating the hard deadlines.

3 Technical Background

3.1 Supervised Learning

Supervised Learning (SL) is a machine learning paradigm wherein explicitly labelled data is used in order to produce the training signals that allow the model to learn useful representations[9]. It is sometimes viewed as limited when compared to Self-Supervised Learning, as its representations can be seen as somewhat rigid[10]. However, due to its inherent ubiquity as the standard paradigm, it still sees widespread use.

For this project, as a range of machine learning methods were to be investigated, a standard supervised model was needed. Not only would this serve as a natural starting point in the actual design and implementation of the other, more complex models, but it would also serve as act as the basis for the ensemble model, in the case that the additional models produce wildly different results.

3.2 Self-Supervised Learning

Self-Supervised Learning is a machine learning paradigm differing from the standard SL, in that it enables useful representations to be learned by the machine without the need for explicit labelling. Rather than rigid relationships, LeCun states this is - at least at a conceptual level - far more akin to the way a human child learns so-called "common-sense"[10]. Where Supervised models learn the representations directly, Self Supervised models learn the more generalised representations through "pretext tasks", then predict the outcomes in "downstream tasks".

In this project, the majority of the research will be conducted on Self-Supervised Learning models. Wen et al[11] produced a summary of varying Self-Supervised Time Series methods that was integral in the research phase of this paper. In this work, a taxonomy of Self-Supervised Learning methods for Time Series data was

proposed that highlights three separate categories of methods: Generative-based methods, Contrastive-based methods, and Adversarial-based methods.

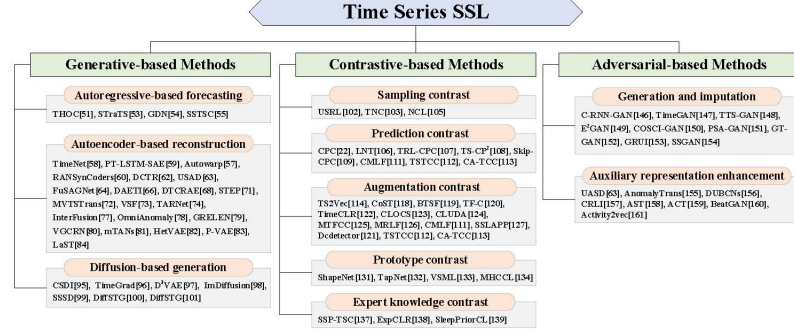


Fig. 2. Taxonomy of SSL for Time Series Methods (Image credit: Wen et al[11])

Below follows a summary of Wen’s proposed taxonomy and definitions:

3.2.1 Generative-based methods

Generative-based methods, which form the basis of the project’s research, consist of generating new data based on a given selection of the data. Common pretext tasks is group are:

- Autoregressive-based Forecasting
- Autoencoder-based Reconstruction
- Diffusion-based Generation

3.2.2 Contrastive-based Methods

Contrastive-based Methods rely on contrasting ”positive” samples with ”negative” samples, where the former are representations that are similar to the input, and the latter are representations that are different. Common pretext tasks is group are:

- Sampling contrast
- Prediction contrast
- Augmentation contrast
- Prototype contrast
- Expert knowledge contrast

3.2.3 Adversarial-based Methods

These methods utilise a Generative Adversarial Network, consisting of a Generator and a Discriminator. The Generator generates synthetic data, akin to real data, and the Discriminator distinguishes whether or not the synthetic data is real data or not. Common pretext tasks is group are:

- Time series generation and imputation
- Auxiliary representation enhancement

3.2.4 *Model Taxonomy Analysis*

Through an analysis of the literature, the Generative-based methods appeared to be the most applicable to a forecasting task. Whilst there was some literature for Generative and Adversarial models applicable to forecasting, they appeared to be far more commonly utilised for alternative tasks, such as anomaly detection, and so it was decided early on that they would not be considered for this particular research.

3.3 *Time Series Data*

A set of data that has datapoints arranged in time. Time series data is often the principle kind of data used in forecasting tasks (such as weather, astronomical, or financial).

3.4 *Financial Market*

Any marketplace where financial securities can be traded. A security is a fungible(indistinct from any other unit of the same security), and trade-able. Stock, bonds, currency, and cryptocurrency are examples of such.

3.5 *Cryptocurrency*

A digital currency, often traded in financial markets. Digital Currencies were long proposed, but not truly possible due to the double spending problem. This was solved in 2009 by Nakamoto with through the introduction of the Blockchain in the Bitcoin Whitepaper[12].

Bitcoin is the original cryptocurrency, created by Nakamoto after the publishing of the white-paper. The ubiquity of Bitcoin made it the ideal choice for training, as it has the most amount of available historical data of any available token. Ethereum is the second most popular token to Bitcoin but fluxuates at a different price range, and so is useful for testing the general representations of trained models.

3.6 *Close Price*

Financial Markets are often measured using a so-called "Candlestick" or "Kline" Chart, alternatively called an "Open High Low Close" (OHLC) chart. In a given "candle" at a given time, the "open" represents the first price in that time interval, the "high" represents the highest price in that time interval, the "low" represents the lowest price in that time interval, and the "close" represents the final price in that time interval. For ease of implementation, the close price was used as the predicted feature in the various machine learning methods.

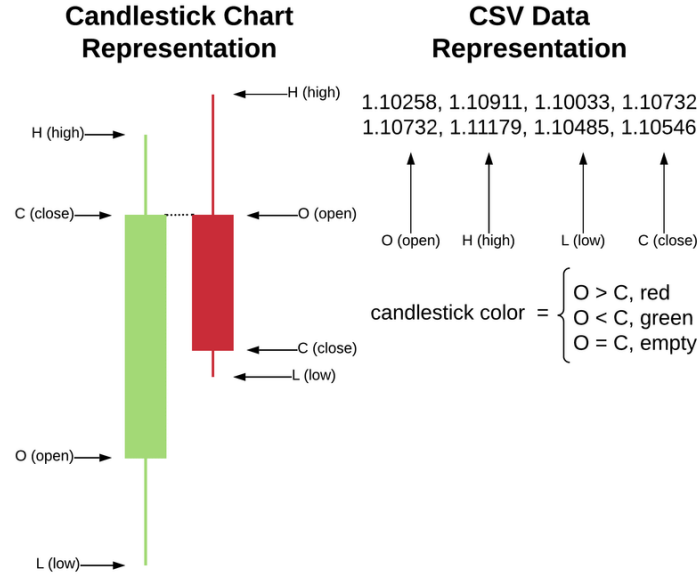


Fig. 3. OHLC Breakdown - Image credit: Nikolay Ivanov, Qiben Yan (Image used under Creative Commons 4.0)[13]

3.7 Dataset

As applying Self-Supervised Learning to Financial Markets Data is a non-standard academic application of Self-Supervised Learning, there is no standardised dataset to be used. However, as highlighted by Peng et al[14], as a result of the nature of Blockchain currencies, a wealth of historical data is available for Bitcoin and Ethereum. This data is available from a multitude of sources, and consisting of a variety of timespans.

4 Design

This section comprises the Design phase of the project.

4.1 Google Colab

To aid in addressing the need for both an environment conducive to high-quality models, and the ability to test the models rigorously and scientifically, Google Colab was chosen as the development environment.

Thanks to its ability to execute code fragments independently from one another, models could be implemented in a modular format, allowing for quicker development time.

4.2 Experimental Design

In order to ensure a wide range of results, the models should be designed in such a way as to be repeatable on larger timescales. To this end, each model should be repeated three times, with one trained to 100 epochs, one trained to 500 epochs, and one trained to 1000 epochs.

Furthermore, in order to ensure there is no bleed between training instances, the Google Colab runtime should be deleted after each successful execution, and a new runtime requested. The results will remain in the notebook, but any data stored in local variables will be flushed, returning the experimental conditions to their initial state.

4.3 Dataset

For the project, two approaches were considered for the dataset - to explore the capabilities of models at a short timespan, with exceptionally small time increments - such as one week with one minute increments, or a longer timespan with larger time increments - such as the entire lifetime of Bitcoin, with daily intervals. While the former application would have some incredibly specific uses, the latter better showcase the ability of SSL to learn more general representations, as the model would learn implicit patterns across the entire lifetime of the data.

To abide by this design requirement for the input data, a dataset spanning from 2023-04-30 to 2024-04-30 was selected. This span of time in particular is somewhat arbitrary, however during this period, Bitcoin experienced periods of stasis and large scale change (insert graph)

Additionally, a secondary dataset detailing the price of Ethereum during the period of 2022-11-05 to 2023-11-05 (roughly six months prior to the training dataset). This would be utilised in order to employ a test of the model using data in which the learned representations may still be relevant, but where they would not be identical.

4.4 Recurrent Neural Network

A Recurrent Neural Network (RNN) is a category of neural network where, unlike uni-directional neural networks, information is able to flow bi-directionally, allowing the network a "memory". Generally, RNNs utilise Sequential or Time-series data[15]. Long Short-Term Memory (LSTM) networks are a form of RNN.

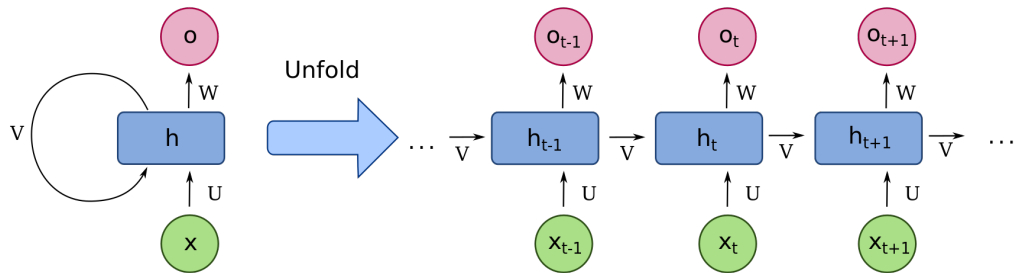


Fig. 4. Breakdown of an RNN, compressed (left) and unfolded (right) (Image credit: fdeloche - Own work, CC BY-SA 4.0[16], <https://commons.wikimedia.org/w/index.php?curid=60109157>)

4.5 Long Short-Term Memory Network

In Machine Learning, training can occur through a gradient-based training method. As the network is trained, the weights present in the network are updated based on the loss of that particular training step. However, this caused the potential to encounter the *vanishing gradient problem*, whereby the magnitude of the gradient decreases or increases exponentially[17] (in cases of increase, the *exploding gradient problem* is another eventuality).

In order to combat this, Long-Short Term Memory or LSTM networks were devised[18]. Their solution to the problem was to construct the network cells using three gates - an "input", an "output", and a "forget". This enabled certain aspects of the input that were found to be repeated regularly to be excluded, allowing the network to retain access to data further in the past[15].

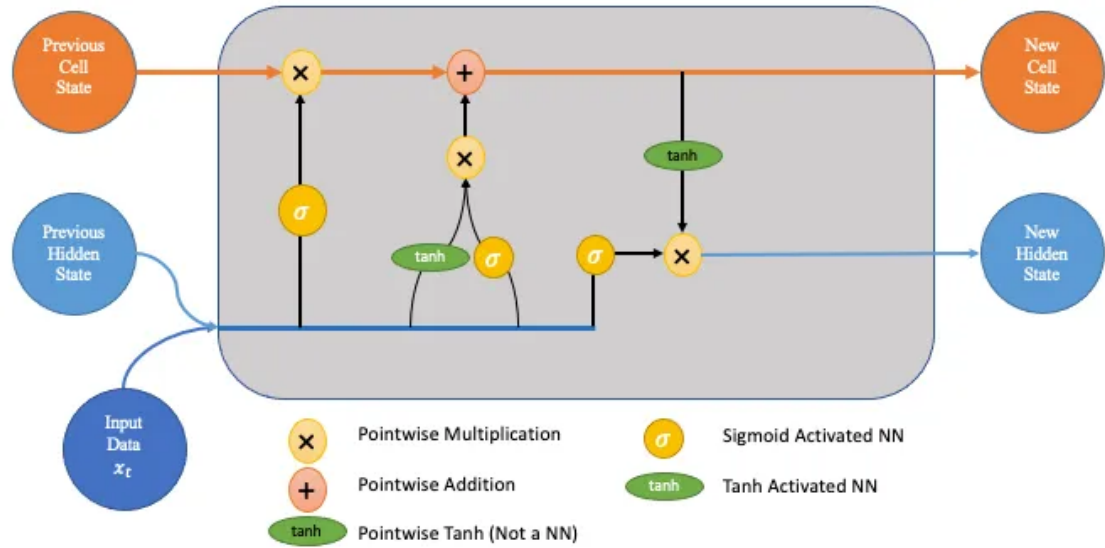


Fig. 5. An breakdown of a LSTM (Image credit: R. Dolphin, Towards Data Science [19])

4.6 Models

As alluded to in Section 2, the models chosen for this project are those that fall under the category of "generative", as a result of a far greater amount of material to utilise in the research. Additionally, observations undertaken during the research phase, especially in Wen et al's[11] survey, imply that Generative models are generally favoured for forecasting tasks, as this category had at least 10 explicit mentions of forecasting tasks, as opposed to Contrastive and GAN which each only had one.

- Autoregressive-based Forecasting - using prior data to forecast future data. Where in traditional regression models, the a given independent variable is used to predict a target[15], the independent and dependent are the same in autoregression.
- Autoencoder-based Reconstruction - reconstructing the input using an autoencoder in order to learn useful representations.
- Diffusion-based Generation - adding "noise" or some other form of obfuscation

to the data in order to force the model to learn more useful generalised representations.

4.6.1 Autoregressive-based Forecasting

Autoregressive-based Forecasting models leverage Self-Supervised Learning in order to develop a useful representation of the past time series up to that point, and utilise that in order to make predictions about the future of the time series.

Autoregression differs to standard applications of regression. Where in a traditional regression problem, the independent variable is used in order to predict the dependent variable [20], in Autoregression, the independent variable also serves as the dependent variable, also called the "target" value or the "ground truth". Autoregressive models can be defined at a high level by the below equation:

$$(1) \quad X_t = \sum_{i=1}^p \varphi_i X_{t-i} + \epsilon_t$$

Fig. 6. The equation defining Autoregression of order p , where $\varphi_1, \dots, \varphi_p$ represents the parameters in the model, and ϵ_t is white noise[21].

In the case of this project, the variable to undergo autoregression would be a sequence of close prices, extracted from the dataset. The model should then learn useful generalisations, understanding a given sequence in the context of the whole rest of the sequence, rather than traditional regression that have no such wider understanding.

The design of the model should utilise a Neural Network adapted for use with Time Series Data specifically. A LSTM-based Recurrent Neural Network (RNN) was chosen specifically due to its popularity in use with Time Series Data[22][23].

4.6.2 Diffusion-based Generation

In this context, Diffusion-based Generation refers to a model where the pretext task involves obfuscating a portion of the time series, and then having the model attempt to predict this unseen portion. In this type of application, the term used is "masking" (the "mask" being applied to part of the time series)[20].

In the case of this project, the model should seek to "mask" a random region of the data, forcing the model to learn a more general representation of the inherent patterns present in the data. For the purposes of the project, the implementation of this model was to be adapted from the Autoregressive model, with the additional requirements added. This similarity allowed for continuity between the model results, as well as having aided in the project's overall development time.

4.6.3 Autoencoder-based Reconstruction

This method relies on an autoencoder that processes the initial data through an encoder, and then attempts to reconstruct it in its original form using the decoder. Similarly to the autoregressive model using the sequence as both the input and the ground truth, the Autoencoder model utilises the original representation as the ground truth, and then compares this to the attempted reconstruction - attempting to minimize this error.

Deviation from standard Autoencoders

Autoencoders have long been researched in the context of deep learning, with innumerable research papers published over the last decade[24][25][26][27]. In order to ensure the research in this project remained novel, the philosophy of the autoencoder-based reconstruction method was applied to an approach that it shares characteristics with, but is by far becoming the new paradigm of machine learning architecture - the Transformer model.

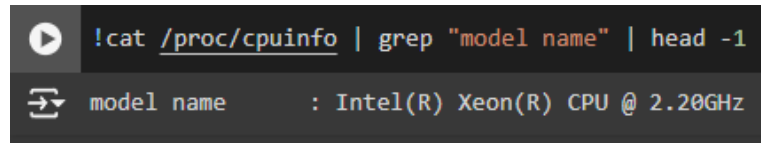
Initially proposed in the 2017 paper by Vaswani et al[28], transformers process input sequences in parallel to one another, allowing for far greater efficiency when compared to Convolutional Neural Networks and Long Short Term Memory Networks. As a result of this, transformers are rapidly becoming a major factor in the development of deep learning models - such as ChatGPT.

For this project, a transformer would be utilised for input reconstruction, rather than a standard encoder-decoder pair. The transformer used was adapted[29] by Keras from the initial version proposed by Vaswani, in order to enable use with time-series data.

5 Implementation

5.1 Environment

As stated in the Design, all models were build in Google Colab. A standard Intel(R) Xeon(R) Platinum 8259CL CPU @ 2.50GHz was used, as for the models in question, this was more than enough processing power to build and train.



```
!cat /proc/cpuinfo | grep "model name" | head -1
model name      : Intel(R) Xeon(R) CPU @ 2.20GHz
```

Fig. 7. CPU information pulled from the Google Colab instance used for the models

5.2 Machine Learning Models

5.2.1 Common Features

Each of the three models share some common features. These allowed for both an easier and more modular implementation, as well as a greater standardisation of results

- **Pandas** - All models utilise Pandas dataframes in order to load the dataset, convert the date column to datetime, and purge unwanted features, leaving only the close column.
- **create_sequence** - All models utilise a function that allows for the creation of a sequence of datapoints within the dataset. This is vital as all three implementations are fitted with a downstream forecasting task.
- **matplotlib** - All models have both their loss output per epoch and their actual prediction results plotted using matplotlib. Furthermore, each model also

plots predictions from a secondary Ethereum dataset in order to demonstrate the effectiveness of the model when given a different financial asset.

As a result of these common features, each of the three models follows a similar structure

- (i) Initialisation - The dataset is loaded in and pre-processed
- (ii) Sequence Creation - The requisite sequences are created
- (iii) Unique section - This section differs per model, and each subsection below will deal with this section
- (iv) Outputs - The model outputs a saved *.h5* file containing its weights, and graphs both the loss result and the predictions

5.2.2 Model Hyperparameters

All three models utilised the Adam optimizer with the following hyperparameters:

- *learning_rate* = 0.001,
- *beta_1* = 0.9,
- *beta_2* = 0.999,
- *epsilon* = 1e-07

All three models shared the same hyperparameters in order to ensure a level of fairness in the experimental results.

5.2.3 Autoregressive-based Forecasting model

The autoregression aspect of this model is implemented as described in the design portion.

This model utilises a small neural network consisting of two LSTM layers of 50 cells each, followed by a dense output layer. Due to this being the simplest of three tasks, there were no additional steps after this.

5.2.4 Diffusion-based Generation model

This model utilises a near-identical neural network as the first model, as it was developed from it (as per the design specification).

Masking Layer

This model differs however in that the first layer of the Neural Network is a "Masking" layer. This layer contains a parameter denoted as *mask_value*; should a value in the input tensor be equal to *mask_value*, the timestep is skipped and the network will not process it, giving the effect of it being "masked" [30]. In this implementation, the *mask_value* was set at 0.

mask_input()

This function takes the current sequence as an input. An array of identical shape to the input tensor is then created, with values distributed uniformly between 0 and 1. This is then evaluated against *mask_ratio*, set at 0.1. An array consisting of *True*

and *False* is then created as a result. Finally, any value evaluated as "True" is set to 0. This is then passed to the masking layer, where these zeroed values are skipped.

5.2.5 Transformer Reconstruction Model

This model, due to being based on a standard transformer structure, differs somewhat from the previous two. In order to allow for as much continuity between results, however, the hyperparameters have been kept the same.

transformer_encoder()

The base element of this model is the *transformer_encoder*. This begins with a *MultiHeadAttention* layer, that computes attentions scores through the use of multiple "heads" - these "heads" allow different parts of the input sequence to be processed simultaneously.

This sequence is then passed through a *dropout* layer and a *normalization* layer, that randomly drop units during the sequence to prevent overfitting, and normalize sequences (respectively).

The input is then passed through the feed-forward part of the network, that consists of a *convolutional* layer with a *ReLU* activation function, followed by another *dropout* layer, another *convolutional* layer, and then a final *normalization* layer.

build_model()

From there, the model is built. Due to the nature of transformer models, the previous section can now be stacked multiple times, to construct a more robust model.

Decoder

This is then passed through a decoder, made up of two convolutional layers, the first layer possessing a *ReLU* activation function.

Parameters

As a result of this structure, the model has several parameters. The key parameters are as follows:

- *head_size* = 256 - the amount of data that the attention heads can process
- *num_heads* = 4 - the number of attention heads
- *num_transformer_blocks* = 4 - the number of transformer blocks to be "stacked" in the model

5.3 Ensemble

The ensemble, whilst not being a "true" model, showcases the mean absolute error of the three models combined, across their various training epochs. This serves as a theoretical demonstration for the potential of a true combined model. Three different ensemble results were obtained for three different epoch lengths. The Mean Absolute Error (MAE) of each model at each epoch length is then taken, with a mean average of the three being produced.

$$(2) \quad MAE = \frac{\sum_{i=1}^n |y_i - x_i|}{n}$$

Fig. 8. The equation defining Mean Absolute Error

$$(3) \quad MAE_{100} = \frac{R_{100} + M_{100} + T_{100}}{3}$$

$$(4) \quad MAE_{500} = \frac{R_{500} + M_{500} + T_{500}}{3}$$

$$(5) \quad MAE_{1000} = \frac{R_{1000} + M_{1000} + T_{1000}}{3}$$

Fig. 9. The equations defining the Ensemble outputs, where R_n represents the MAE of the Autoregressive model at n epochs, M_n represents the MAE of the Masking model at n epochs, and T_n represents the MAE of the Transformer model at n epochs.

6 Results and Evaluation

The models were subject to execution at 100, 500, and 1000 epochs each.

6.1 Autoregressive Model

6.1.1 Experimental Results

Epoch 100

- Loss at Epoch 100: 0.0142,
- Validation Loss at Epoch 100: 0.0306

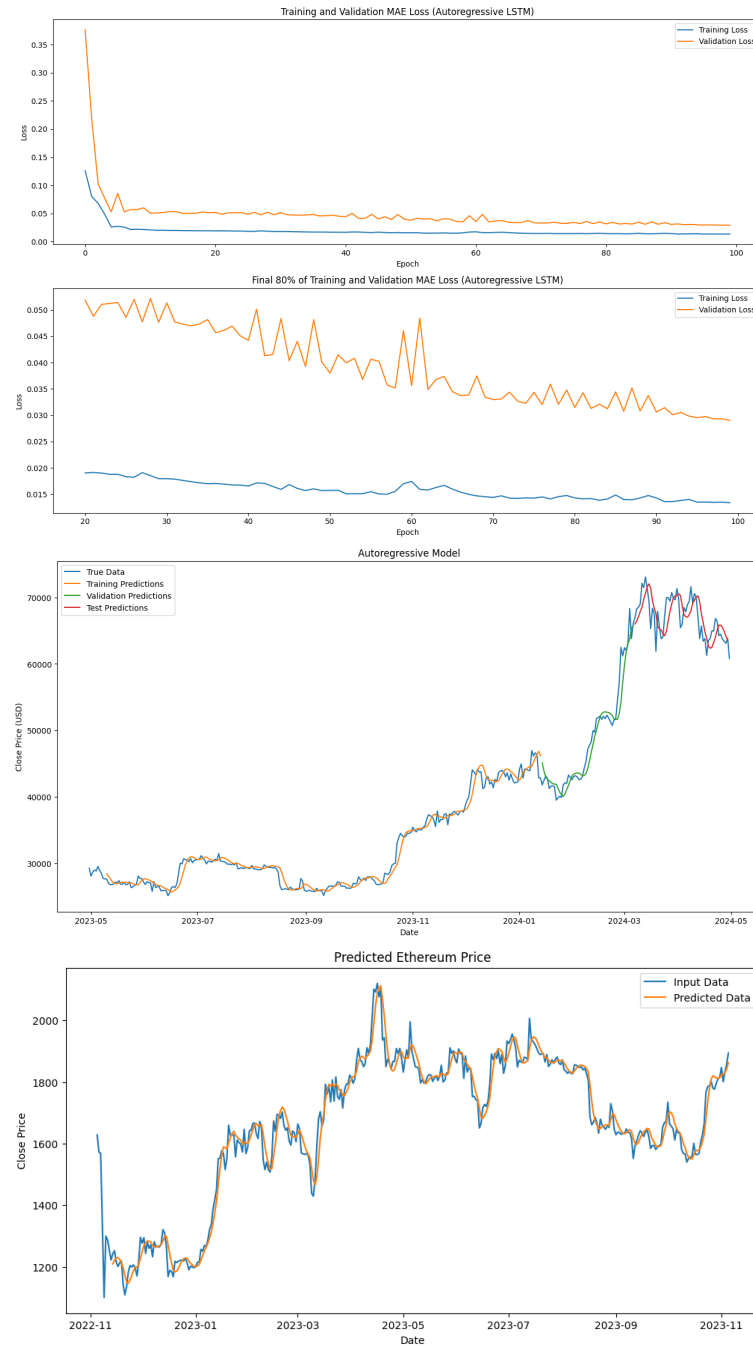


Fig. 10. Autoregressive Model at 100 Epochs

Epoch 500

- Loss at Epoch 500: 0.0096
- Validation Loss at Epoch 500: 0.0407

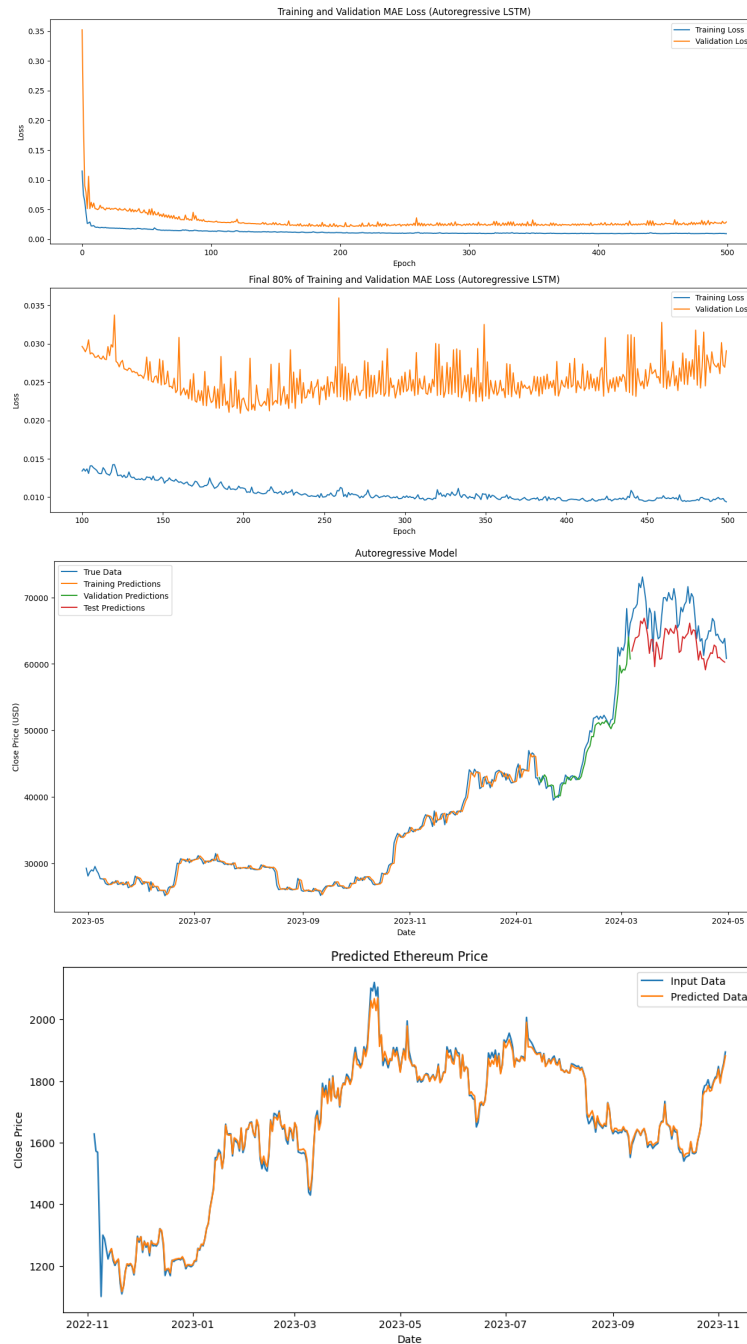


Fig. 11. Autoregressive Model at 500 Epochs

Epoch 1000

- Loss at Epoch 1000: 0.0095
- Validation Loss at Epoch 1000: 0.0497

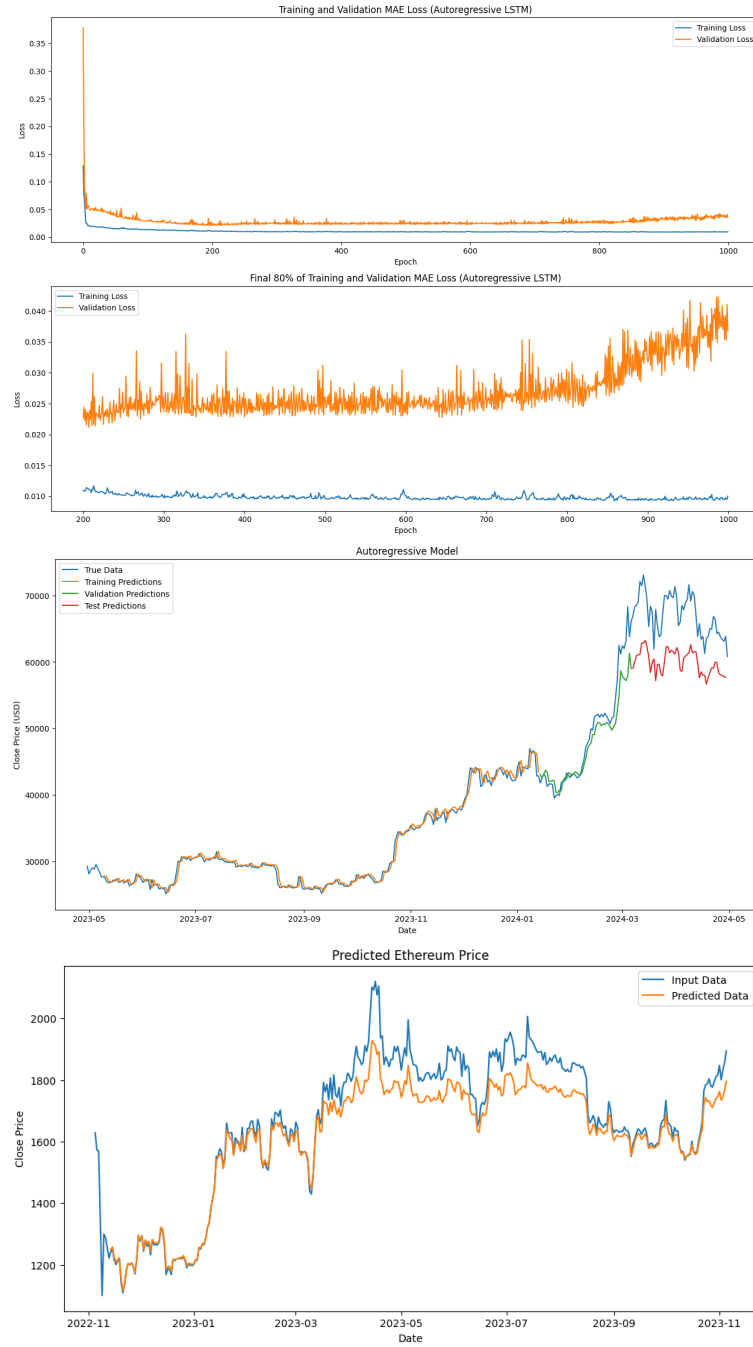


Fig. 12. Autoregressive Model at 1000 Epochs

6.1.2 Interpretation

The Autoregression model shows a satisfactory ability to predict the Bitcoin close price with a high degree of accuracy base. From Epoch 1 to 100, a clear reduction in Loss is present. The predicted results follow a curve along the ground truth results, though these predictions are clearly more generalised, as indicated by the prediction curve having a lack of rigid peaks and troughs - this is satisfactory though as this is only the first set of epochs.

Epoch 1-500 presents a somewhat interesting dichotomy. The prediction curve has increased in precision, matching the shape of the ground truth values far more closely. However, the Test predictions have taken on a slight skew to the average of the results, being "pulled down" to the horizontal centre-point of the graph. This seems to be an early indicator of overfitting and will be noted in several other higher-epoch results.

Epoch 1-1000 appears to support this speculation, as whilst once again, the pattern has become clearer, the predictions are even closer to the average. This is further supported by the quantitative data, in the form of the MAE loss values. For Epochs 1-500, the loss and validation loss are both *greater* than that of 1-1000. Finally, this is cemented by the Ethereum data, which shows a similar improvement from Epoch 100 - 500, as with the Bitcoin data, and a similar degradation from Epoch 500 - 1000.

6.2 Masked Model

6.2.1 Experimental Results

Epoch 100

- Loss at Epoch 100: 0.0130
- Validation Loss at Epoch 100: 0.0281

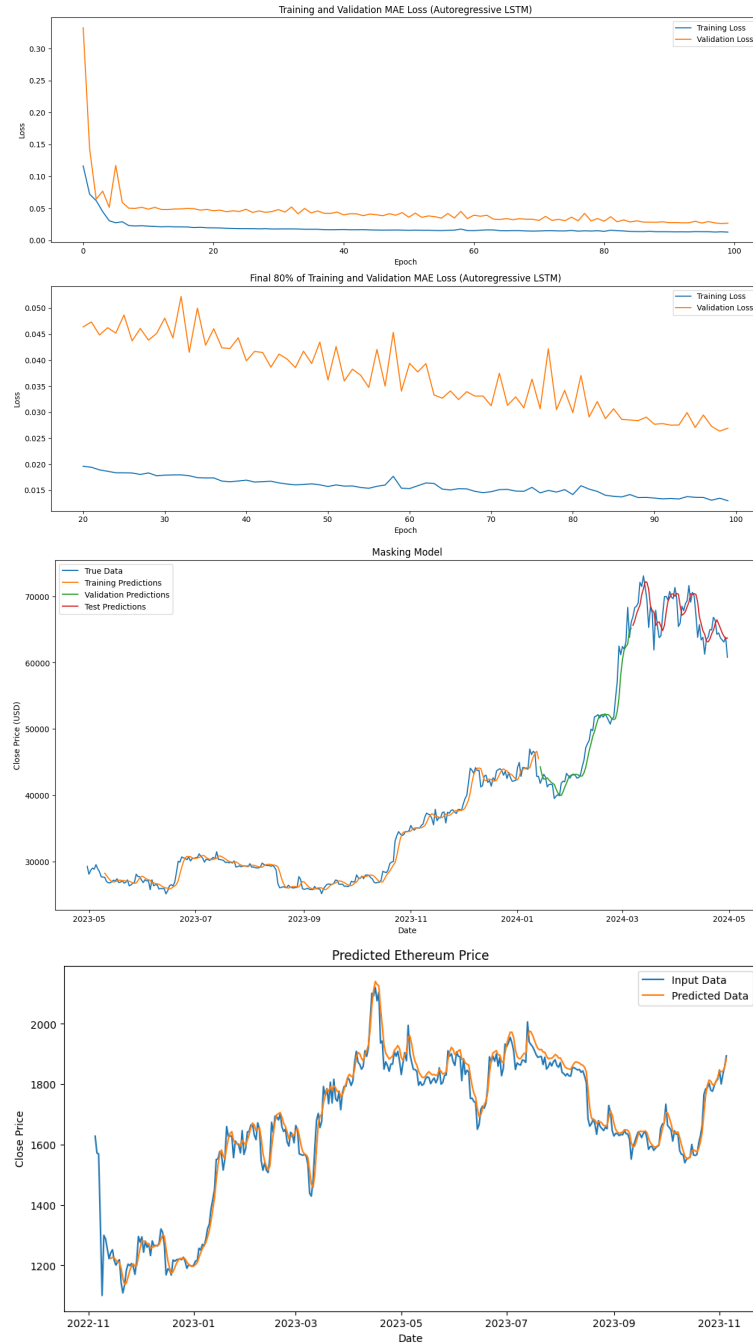


Fig. 13. Masked Model at 100 Epochs

Epoch 500

- Loss at Epoch 500: 0.0097
- Validation Loss at Epoch 500: 0.0231

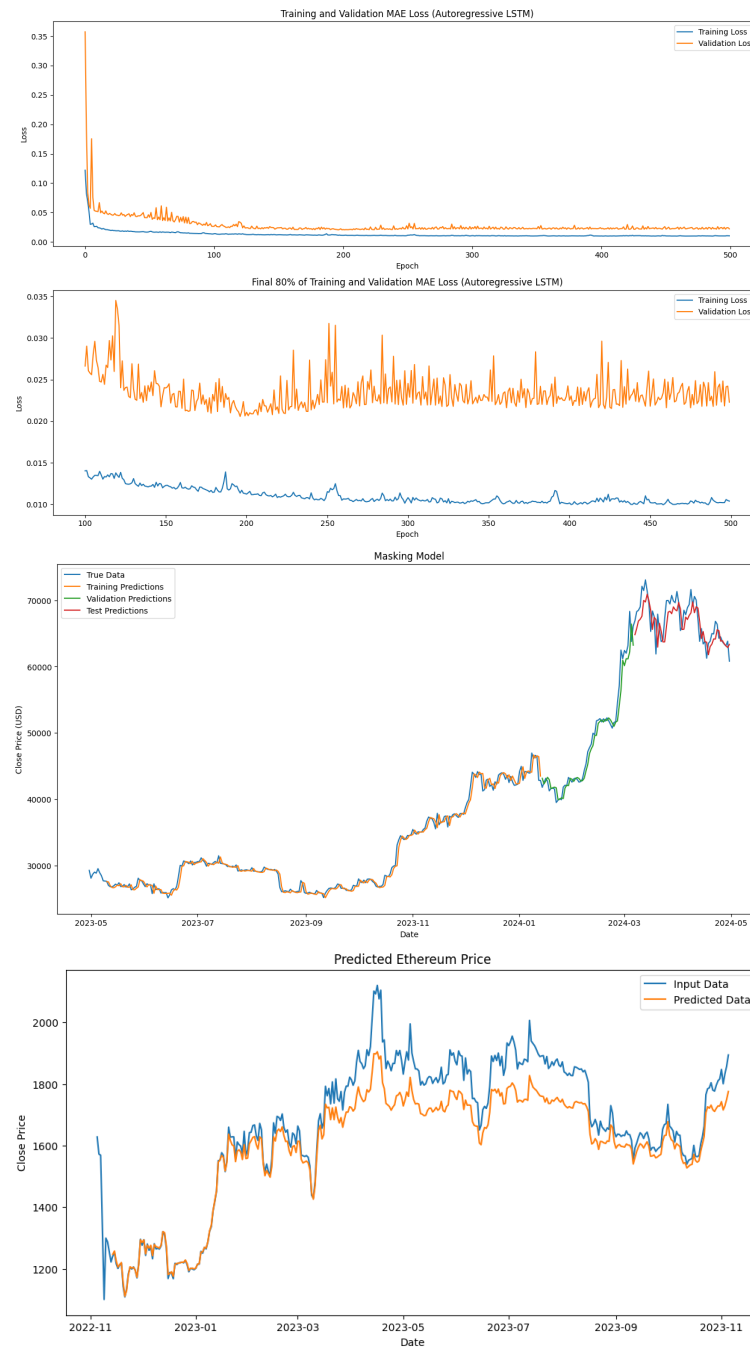


Fig. 14. Masked Model at 500 Epochs

Epoch 1000

- Loss at Epoch 1000: 0.0099
- Validation Loss at Epoch 1000: 0.0562

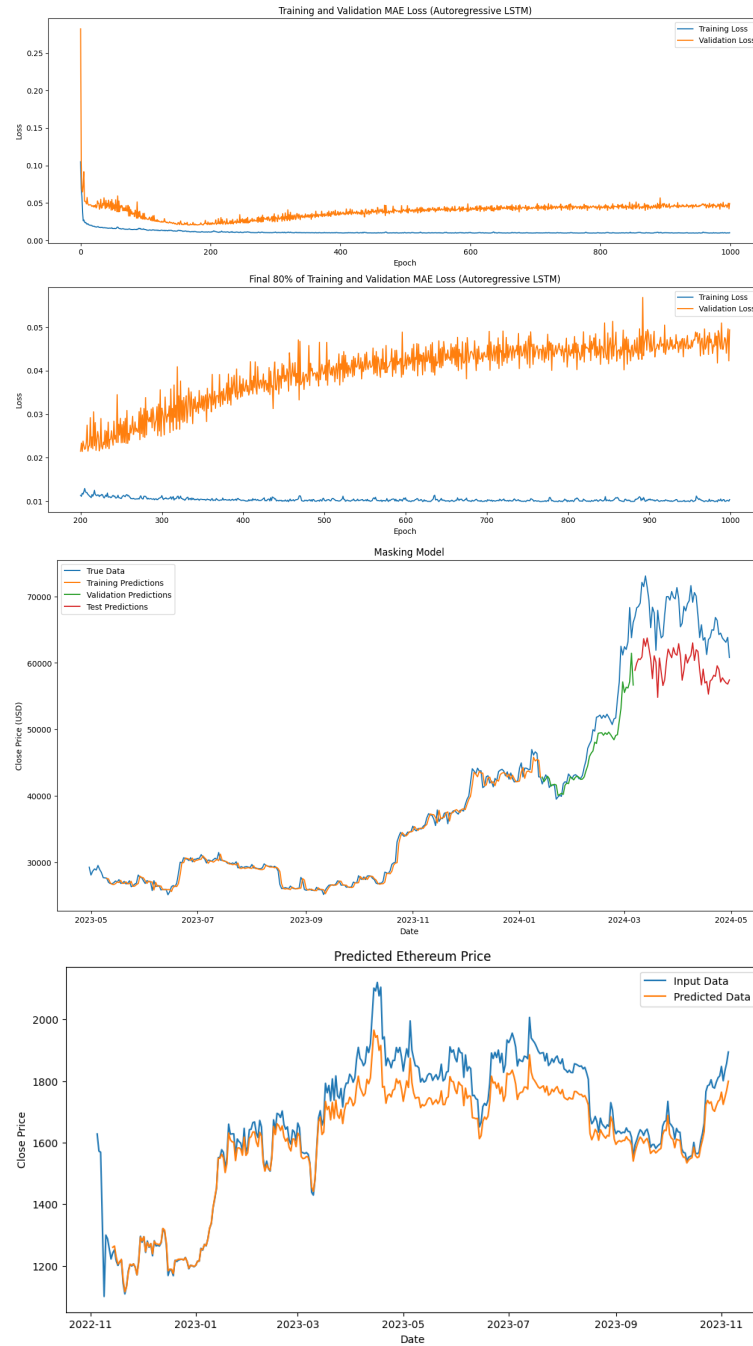


Fig. 15. Masked Model at 1000 Epochs

6.2.2 Interpretation

The results for this model overall are somewhat similar to those of the previous model, with the loss decreasing as expected up to 500 epochs, before showing a slight increase by the time of reaching 1000 epochs, as shown in the main results graphs. However, through careful observation of the Ethereum graph, it is clear that the 1000 epoch model does indeed perform marginally better than the 500 epoch predictions, being closer to the ground truth line at most points, in spite of a higher validation loss. This may suggest that the Masking method allows for a more effective learned representation. It should be noted that the 100 epoch Ethereum predictions appear overall closer, however much as the case with the Autoregressive results, this curve is far more general.

6.3 Transformer Model

6.3.1 Experimental Results

Epoch 100

- Loss at Epoch 100: 0.0363
- Validation Loss at Epoch 100: 0.1373

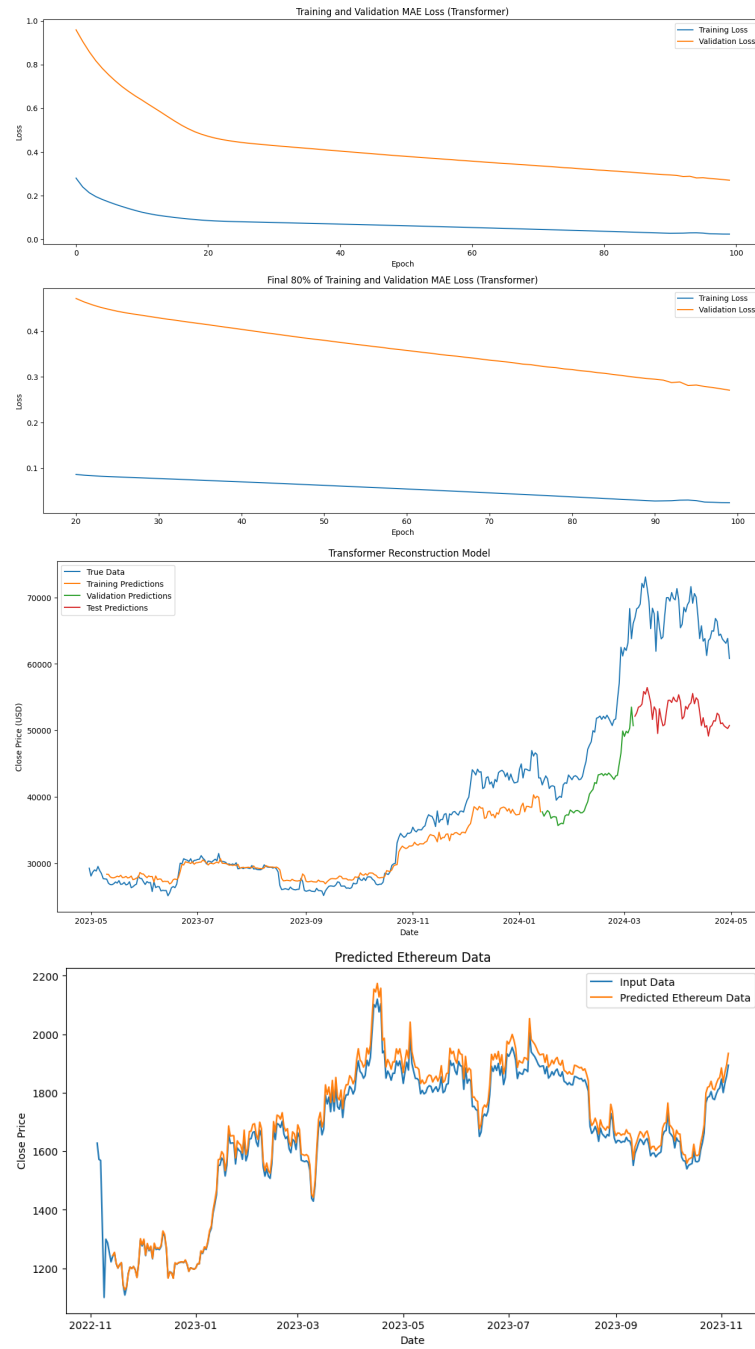


Fig. 16. Transformer at 100 Epochs

Epoch 500

- Loss at Epoch 500: 0.0012
- Validation Loss at Epoch 500: 0.0105

Epoch 1000

- Loss at Epoch 1000: 6.9697×10^{-4}

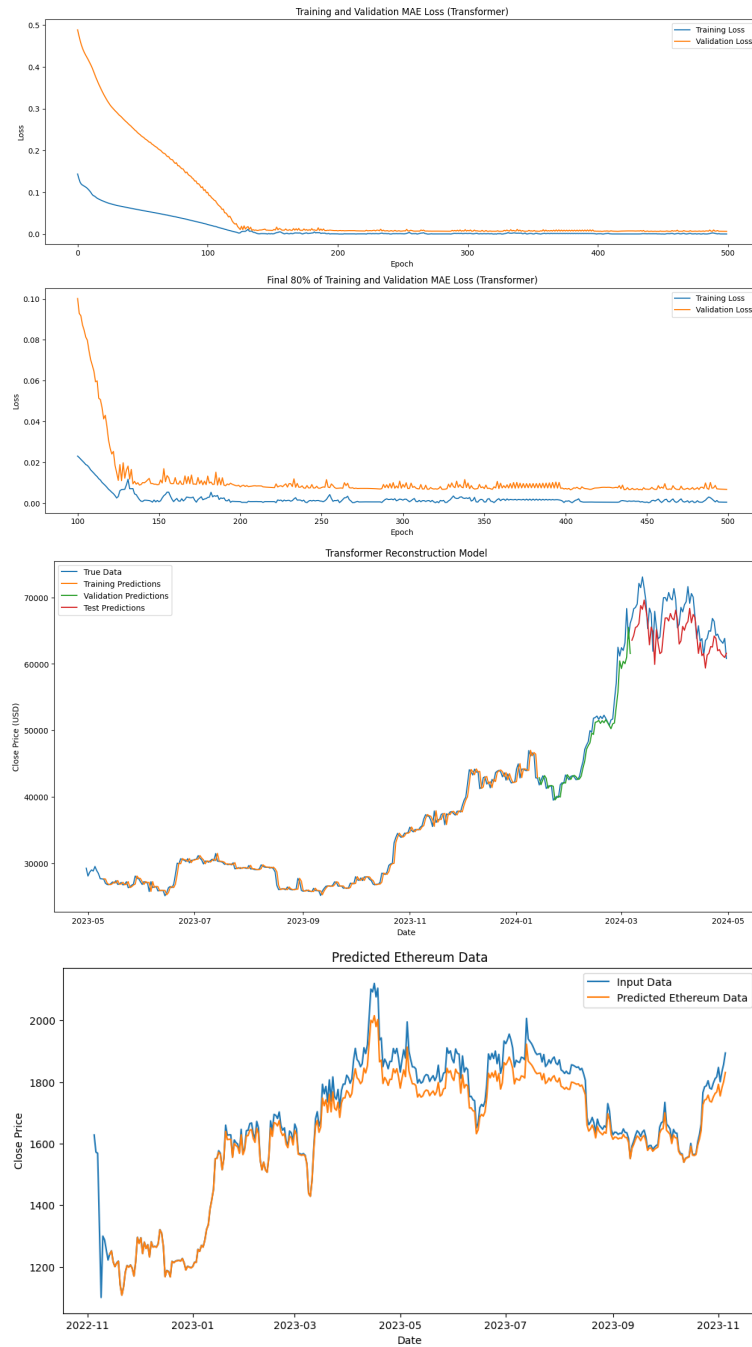


Fig. 17. Transformer at 500 Epochs

- Validation Loss at Epoch 1000: 0.0407

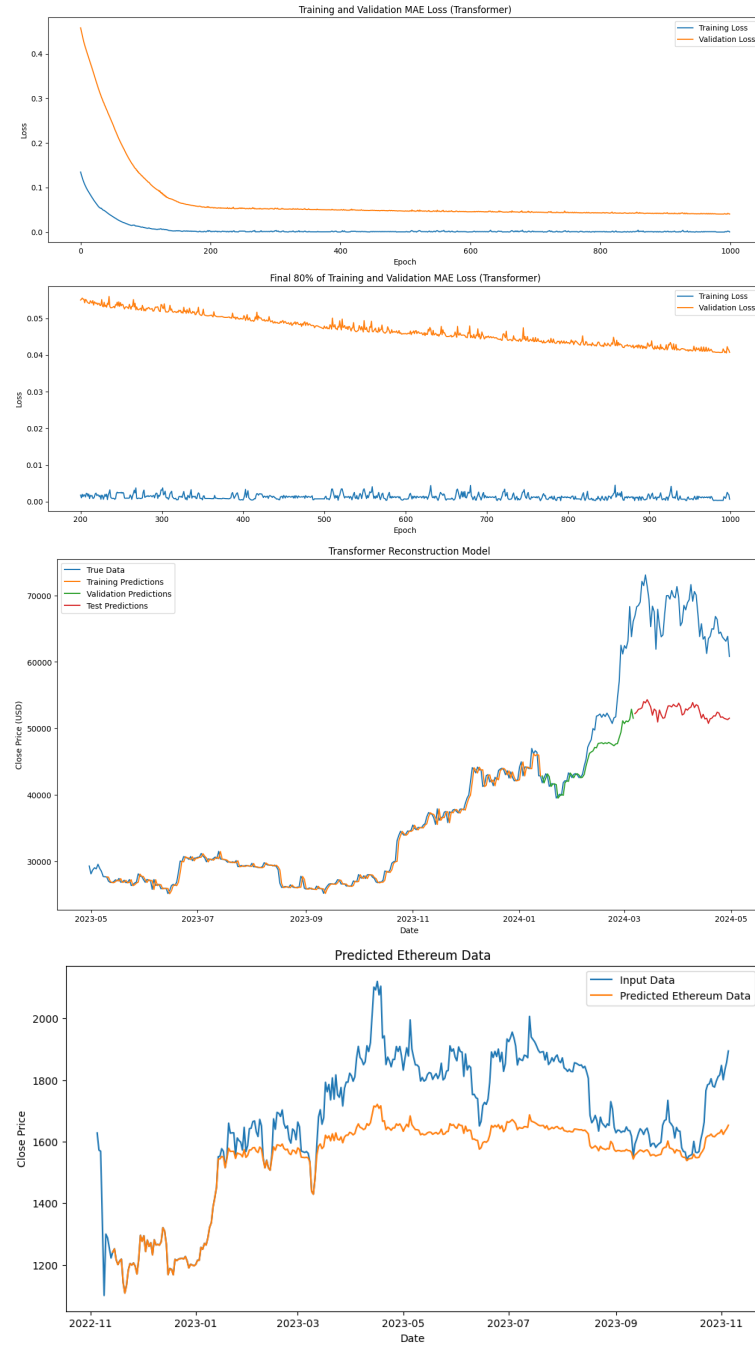


Fig. 18. Transformer at 1000 Epochs

6.3.2 Interpretation

Unlike the previous two models, that show a more generalised curve in the first 100 epochs, the Transformer model (which implements a form of autoencoder-based reconstruction) predicts the shape of the graph with remarkable precision, with the caveat being that the graph itself is plotted closer to the horizontal average (as in the later epochs of the previous two models).

By Epoch 500, this model shows significant accuracy, with the precision having been maintained.

By the Epoch 1000 however, significant overfitting has occurred and once again, the model has been pulled back towards the average. The jump in validation loss, in spite of the decrease in training loss, appears to support this. As expected, therefore, the 500 epoch variant performs the best on the Ethereum dataset, being the closest to the ground truth values.

6.4 Ensemble Model

6.4.1 Experimental Results

- Average Loss at 100 Epochs: 2.1167×10^{-2}
- Average Loss at 500 Epochs: 6.8333×10^{-3}
- Average Loss at 1000 Epochs: 6.6990×10^{-3}

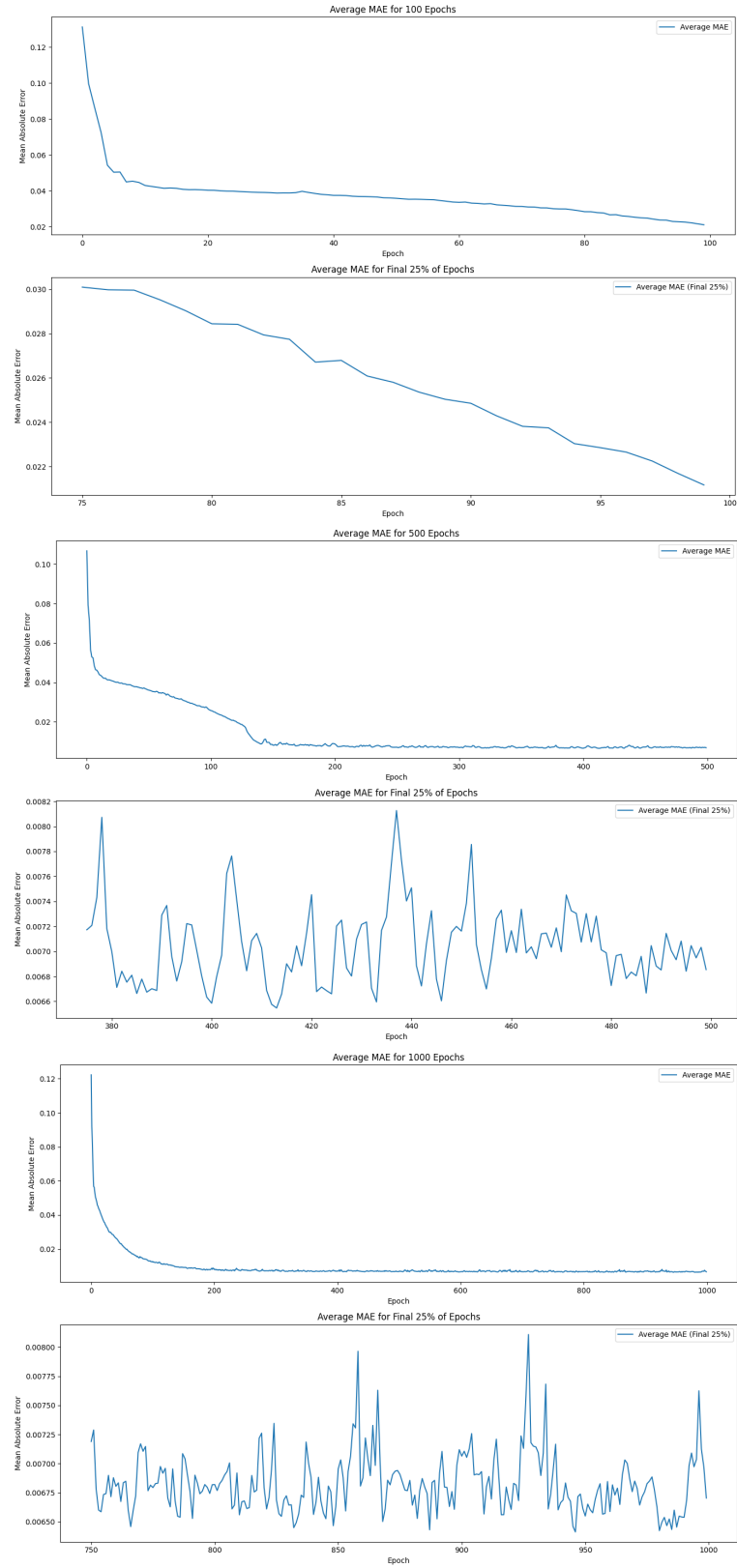


Fig. 19. Results for the Ensemble model

6.4.2 Interpretation

The ensemble model appears to confirm some of the observations made, particularly relating to the second training phase up to Epoch 500 far outperforming the initial training stage. Additionally however, it also confirms that in spite of the apparently lackluster performance of the third training phase, in actual fact, the third phase categorically performed better - though evidently by a small margin. This upholds some of the observations made in particular about the Ethereum test dataset, wherein the performance of the third training phase in some instances appeared better there than in the Bitcoin test data

6.5 Evaluation

The results ultimately highlight that, under experimental conditions, self-supervised learning methods are effective at predicting Bitcoin and Ethereum prices through Time-Series Data. In particular, the performance of the transformer model was the most impressive, boasting a minimum loss value of 6.6990×10^{-3} . Furthermore, the ensemble model suggests that, through a more thorough combination of these methods, a far more precise and robust model could be constructed.

6.5.1 Critique

In spite of the above, which were obtained through rigorous experimental methods, the construction of the models themselves is a point of major critique. While they do perform to a satisfactory degree, the lack of hyperparameter optimization marks a clear unknown, particularly as it could have impacted the model performances drastically. In addition, the observations on the similarities between the results of the Autoregressive and Masked models cannot be discussed in a vacuum, without mention of the fact that they were constructed from the same base model.

6.5.2 Defence

While considering these criticisms, it must be kept in mind the exceptionally short timescale of this project, showcased in the Planning section. The models produce directly address the requirements outlined, regardless of criticism relating to their robustness; high-quality, quantitative and qualitative results were produced, that were comprehensively evaluated.

Furthermore, the lack of specific hyperparameter optimization, or similarities between the Autoregressive and Masking models do not detract from the results themselves. Whilst this is a point that could be improved upon, the results themselves were obtained in a rigorous, scientific manner.

7 Conclusions

7.1 Project Achievements

The core motivation for this project was to undertake research in an area that currently appears to be under-examined. As such, the goals of this project were to explore a variety of self-supervised machine learning models, to create an ensemble using these models, and obtain high-quality experimental results. These goals

were met categorically, with a wealth of results being produced for the various machine learning models. Furthermore, rather than basing the entire research on the categories established in Wen et al's[11] taxonomy, efforts were made to explore alternative novel solutions, such as the case with the transformer model.

7.2 *Shifting of Goals*

The goals of the project did not shift during the course of the development, largely due to the short timescale. In order to ensure results were delivered, the goals had to be assured from the earliest phase, and so as a result, there was little change from the initial plan.

7.3 *Future Research*

Despite the headway made by this project, there is still a plethora of research to be done in this field. A potential avenue for development is combining established anomaly detection, sentiment analysis, and forecasting models to create a multi-variate "master-model" capable of making forecasting predictions based on a wide range of factors. Alternatively, due to the graphed nature of financial markets, they could be approached as a computer vision task, applying solutions more common in fields such as Autonomous Vehicles to make probabilistic judgements about the position of candles on a OHLC graph.

7.4 *Closing Remarks*

Despite the inherent limitation of the timescale, the work done in this project clearly demonstrates its worth. From the short time in development, the potential of the ideas within the project has been demonstrated, and the possibilities for future research in this area is near infinite.

8 Acknowledgements

I would like to thank my supervisor, Dr Varun Ojha, for his guidance throughout this project.

References

- [1] Grand View Research. Self-supervised learning market size & share report, <https://www.grandviewresearch.com/industry-analysis/self-supervised-learning-market-report>, 2021.
- [2] Wai Weng Lo, Gayan K. Kulatilleke, Mohanad Sarhan, Siamak Layeghy, and Marius Portmann. Inspection-l: self-supervised gnn node embeddings for money laundering detection in bitcoin. *Applied Intelligence*, 53, Mar 2023.
- [3] Shucheng Li, Runchuan Wang, Hao Wu, Sheng Zhong, and Fengyuan Xu. Siege: Self-supervised incremental deep graph learning for ethereum phishing scam detection. *31st ACM International Conference on Multimedia*, Oct 2023.
- [4] Shiyu Ouyang, Qianlan Bai, Hui Feng, and Bo Hu. Bitcoin money laundering detection via subgraph contrastive learning. *Entropy*, 26(3):211–211, Feb 2024.
- [5] Greg Hogg. Stock price prediction & forecasting with lstm neural networks in python - youtube, <https://web.archive.org/web/20240522024219/https://www.youtube.com/watch?v=CbTU92pbDKw>, accessed 2024-05-22, Mar 2022.

- [6] NeuralNine. Predicting crypto prices in python - youtube, <https://web.archive.org/web/20240522024333/https://www.youtube.com/watch?v=GFSiL6zEZF0>, accessed 2024-05-22, Apr 2021.
- [7] Forbes Council. "13 Tech Leaders Share Their Preferred Software Development Methodologies And Strategies", Forbes, <https://www.forbes.com/sites/forbestechcouncil/2022/08/15/13-tech-leaders-share-their-preferred-software-development-methodologies-and-strategies/> (accessed April. 17, 2023).
- [8] R. Sherman. "Waterfall Methodology - an overview", Science Direct, <https://www.sciencedirect.com/topics/computer-science/waterfall-methodology> (accessed April. 5, 2023), 2015.
- [9] IBM. What is supervised learning? - ibm, <https://www.ibm.com/topics/supervised-learning>, accessed 2024-05-02, 2023.
- [10] Y. LeCun. "Self-supervised learning: The dark matter of intelligence", Facebook, <https://ai.facebook.com/blog/self-supervised-learning-the-dark-matter-of-intelligence> (accessed 2024-04-05).
- [11] Kexin Zhang, Qingsong Wen, Chaoli Zhang, Rongyao Cai, Ming Jin, Yong Liu, James Zhang, Yuxuan Liang, Guansong Pang, Dongjin Song, and Shirui Pan. Self-supervised learning for time series analysis: Taxonomy, progress, and prospects, 2023.
- [12] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2009.
- [13] Nikolay Ivanov and Qiben Yan. Constraint-based inference of heuristics for foreign exchange trade model optimization, https://www.researchgate.net/figure/e-OHLC-candlestick-data-in-chart-and-comma-separated-value-CSV-representations_fig1_352017050, accessed 2024-05-01, 05 2021.
- [14] Peng Peng, Yuehong Chen, Weiwei Lin, and James Wang. Attention-based cnn-lstm for high-frequency multiple cryptocurrency trend prediction. *Expert Systems with Applications*, 237:121520–121520, Mar 2024.
- [15] IBM. What are recurrent neural networks? - ibm, <https://www.ibm.com/topics/recurrent-neural-networks>, accessed 2024-05-02, 2023.
- [16] fdeloche. Compressed and unfolded basic recurrent neural network. <https://commons.wikimedia.org/w/index.php?curid=60109157>, accessed 2024-05-23, 2017.
- [17] Sunitha Basodi, Chunyan Ji, Haiping Zhang, and Yi Pan. Gradient amplification: An efficient way to train deep neural networks, 2020.
- [18] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.
- [19] Rian Dolphin. Lstm networks — a detailed explanation, <https://towardsdatascience.com/lstm-networks-a-detailed-explanation-8fae6aefc7f9>, accessed 2024-04-29, Mar 2021.
- [20] IBM. What is self-supervised learning? - ibm, <https://www.ibm.com/topics/self-supervised-learning>, accessed 2024-05-02, 2023.
- [21] Et Box, George E P Al. *Time series analysis : forecasting and control*. John Wiley and Sons, Hoboken, New Jersey, 1994.
- [22] Marco Del Pra. Time series forecasting with deep learning and attention mechanism, <https://towardsdatascience.com/time-series-forecasting-with-deep-learning-and-attention-mechanism-2d001fc871fc>, accessed 2024-05-02, Nov 2020.
- [23] Time series forecasting. Time series forecasting tensorflow core, https://www.tensorflow.org/tutorials/structured_data/time_series, accessed 2024-05-10, Mar 2024.
- [24] Chris Häusler, Alex Susemihl, Martin P Nawrot, and Manfred Opper. Temporal autoencoding improves generative models of time series, 2013.
- [25] Nikolaos Gianniotis, Dennis Kügler, Peter Tino, Kai Polsterer, and Ranjeev Misra. Autoencoding time series for visualisation, 2015.
- [26] Pankaj Malhotra, Vishnu TV, Lovekesh Vig, Puneet Agarwal, and Gautam Shroff. Timenet: Pre-trained deep recurrent neural network for time series classification, 2017.
- [27] Abubakar Abid and James Zou. Autowarp: learning a warping distance from unlabeled time series using sequence autoencoders, 2018.
- [28] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need (v1), 2017.
- [29] Theodoros Ntakouris. Keras documentation: Timeseries classification with a transformer model, https://keras.io/examples/timeseries/timeseries_classification_transformer/, accessed 2024-05-02, Aug 2021.
- [30] Keras Team. Keras documentation: Masking layer, https://keras.io/api/layers/core_layers/masking/, accessed 2024-05-01, 2023.