CALIFORNIA STATE UNIVERSITY
FULLERTON

Department of Computer Science

This project has been satisfactorily demonstrated and is of suitable form.

This project report is acceptable in partial completion of the requirements for the Master of Science degree in Computer Science.

Predicting Future Prices of ETH with LSTM

Project Title (type)

John Tu

Student Name (type)

Rong Jin

Advisor's Name (type)

Advisor's signature Date 05/10/2023

Kanika Sood

Reviewer's name

Reviewer's signature Date

CPSC 597 Project Report: Predicting Future Prices of ETH with LSTM

Created by John Tu

Advisor: Dr. Rong Jin

Reviewer: Dr. Kanika Sood

Abstract

Price prediction remains the most popular topic to explore in machine learning and neural networks. The purpose of conducting price prediction experiments is to better understand the change of trends going on for any stock price as well as noting any future patterns that may occur based on a specific time frame. Inspired by this background domain, this project focuses on implementation of recurrent neural networks to predict the future prices of Ethereum, a type of cryptocurrency that earned its popularity, second only to Bitcoin. The procedures and methodologies for this experiment involves 2 parts of the experiment: the first one involves creating 4 separate datasets based on the training-test length ratio and the second one is about the implementation of various neural network models to compare the training losses and the predicted results correlating with the actual ones for each dataset.

Keywords: machine learning, neural networks, recurrent neural networks (RNN), stock price prediction, Ethereum, long short-term memory (LSTM), cryptocurrencies, blockchain technologies

Table of Contents

1 Introduction

1.1 Definition of the Problem

What makes cryptocurrencies distinct from other currencies that exists in today's world? A unique trait that they have is decentralization, which, based on Gervais et al.'s definition [1], indicates that rather than being managed by a central organization, it is the users that oversee the transactions and activities of those digital currencies. Since the advent of Bitcoin in 2009, cryptocurrencies have dominated the financial marketplace and other successors have risen in popularity in the past decade such as Ethereum, Dogecoin, Litecoin, and Bitcoin Cash. While they enjoy an ever-growing popularity in letting users obtain their own digital money via trading exchange services and using them as alternate methods of payments in online stores, cryptocurrencies have still yet to capture worldwide attention in the news and media when it comes to predicting the future prices in the market. There are a wide variety of data science and machine learning methods to effectively analyze the behaviors and predict the future trends of cryptocurrency pricing, yet given the insurmountable size of the data to analyze, such existing methods will be nearly impossible to handle. In order to address those shortcomings, deep learning techniques are deployed to handle huge amounts of real-world data and process them to provide accurate results, whether it is to make predictions or describe the patterns. One example of deep learning is artificial neural networks (or ANNs for short), which consists of multiple nodes linked together that passes along data throughout the entire network (much akin to simulating a human brain in the nervous system). Similar to machine learning, ANNs are designed to teach computer systems the ability to process and evaluate arbitrarily large amounts of input data. The most common types and methods of ANNs along with their associated examples in applications are recurrent neural networks (RNN) in Chinese character recognition from Zhang et al. [2], convolutional neural networks (CNN) in facial expression recognition from Meng et al. [3], perceptron networks in phoneme recognition from Sivaram and Hermansk [4], and autoencoder networks in speech emotion recognition from Deng et al. [5].

1.2 Objective of the Study

The goal of this research project is to predict next year's prices of Ethereum by implementing RNN-LSTM models. This shall be accomplished first constructing a baseline LSTM model of a single layer and with default parameters, followed by adding one or more features in the baseline model, such as dropout rates and regularizers. Furthermore, I will also do a side-by-side comparison of LSTM with other deep learning models such as multilayer perceptron and

convolutional neural network (or CNN), as well as creating hybrid neural network models by combining existing ones such as CNN and LSTM networks. The questions for this project are defined as followed: How much relevance does LSTM hold in being able to accurately predict future prices of Ethereum? Will the addition of features such as dropout rates and regularizers improve or worsen the ability to minimize training losses and errors? In what cases will LSTM outperform and be outperformed by other neural network models, such as MLP, CNN, and GRU?

1.3 Review of Related Research

Neural network methods aren't limited to any specific domain as there are some techniques that overlap between problem spaces. For example, a conference paper from Selvin et al. [6] mentioned about the implementation of a hybrid neural network where it combines both features from recurrent and convolutional neural networks to predict the stock market prices. Another example is using both ANN model and neuro-fuzzy model (or a model that combines both neural networks and fuzzy logic) to compare predicted gold prices according to a conference paper written by Farahani and Mehralian [7]. These indicate that some neural network models can be combined together to form a more complex method, and it is possible that certain problem spaces can be solved with methods outside of the intended ones, such as text processing and time-series prediction with 2-dimensional convolutional neural network.

In addition, there are related research articles that I found based on the topic for my project.

For example, there is a conference paper from Kumar and Rath [8] where it describes the procedures on how to implement deep learning techniques to predict future prices of Ethereum. The two models that are designed for this experiment are multilayer perceptrons and long short-term memory (or MLP and LSTM for short). The main feature that differentiates between the two is that input in the LSTM model depends on memory from the previous output, whereas MLP does not. Based on the results obtained for the losses of both models, it is LSTM that does better in predicting future prices than MLP, given with the lower loss metrics like mean squared error and mean absolute error. Also, an online article from Kwon et al. [9] discusses about applying LSTM in predicting cryptocurrency price trends, and the experiment the authors conducted involves processing the time series data through a LSTM model followed by fine-tuning via adjustment of the parameter values. Following that, the LSTM model is evaluated along with the gradient boosting (or GB)

model, and both of them are compared in terms of f1-score values for benchmark purposes. Based on the results, the LSTM model performed much better than GB.

Another related research article is about predicting cryptocurrency prices with stochastic neural networks from Jay et al. [10]. In general, stochastic neural networks rely on randomization of certain values within the system, such as the weights of each node in the layer. By randomly assigning arbitrary values within the neural network, stochastic models aim to not only provide accurate forecasting in future prices of cryptocurrencies but also remedy potential issues where the price lists may not easily adapt to sudden real-time changes to the market.

A third research article is from Zhang et al. [11], and it talks about predicting stock market prices with generative adversarial networks (or GAN for short). GANs consist of two models: a generator that produces counterfeit copies that are almost similar to the real data and a discriminator that tries to determine whether the data produced by the generator is real or fake. This concept of using GANs to identify potential false copies of the real data is based on a scholarly article from Creswell et al. [12], where it demonstrates the effectiveness of generating samples of the image that appears to be almost imperceptible from the real one and following up with discerning features where the copies may appear dissimilar to the actual image. Similar to what this article discussed about, GANs can also be applied in recurrent sequential datasets like stock market prices, where generators can produce prices that appears almost similar to the actual ones and discriminators to distinguish between generated counterfeits and real values.

The fourth research article discusses about implementation of Twitter sentiments along with RNN models to predict future prices of Bitcoin, according to a conference paper from Pant et al. [13]. In general, Twitter sentiments are a key factor in determining the outlook of specific topics based on user opinion, and the possible outcomes can be positive, neutral, or negative. As stated in the article, Twitter sentiments can also be deployed along with RNN models to provide an effective forecasting method in predicting future prices of Bitcoin based on the current subjective outlook from the list of tweets. Besides using RNNs along with Twitter sentiments to predict future prices of Bitcoin, there are other ways to approach this problem with methods other than neural networks themselves. One way is to use big data techniques like KryptoOracle, a software platform built with Apache environment that specializes processing large amounts of tweets and pricing lists, according to a conference paper by

Mohapatra et al. [14]. Another way is using machine learning techniques to predict the future prices of cryptocurrencies, as noted from Valencia et al. [15]. Overall, sentiment analysis via social media is an approach that I may not be familiar with, yet an interesting way to approach the prediction problem.

2 Part 1: LSTM Experiment

This project will be conducted in Google Colab with the entire source code written in Jupyter notebook format. The software libraries that shall be used for this project are Keras [16], TensorFlow [17], Pandas [18], NumPy [19], Matplotlib [20], and Scikit-Learn [21]. For the first two, Keras and TensorFlow are essential in building and processing neural network models. For the other libraries, they are useful in handling a variety of operations, such as reading and reshaping the dataframe of the input data (Pandas), plotting the data with graphs (Matplotlib), and reshaping and transforming the data (NumPy and Scikit-Learn).

The baseline model that I shall construct is a long short-term memory network with a couple of LSTM layers and with default parameters. The improved models will use one of the following features for each model: dropout rate and regularizers. (These features shall be detailed in the following sections below.) The strategy is to start with the implementation of the baseline model, then gradually add additional features to the existing model, starting from the simplest improvements and move towards more complex methods like combining features together. For Part 1 of the experiment, there will be 4 datasets of the ETH price list, each of them with their predefined training and test lengths, which shall be defined in Section 2.2 below.

For the assumptions, the Ethereum price dataset obtained from Yahoo Finance does not and should not contain any missing data or values (that is, not a single row contains N/A or NULL) that may affect the quality and efficiency of the training process. For limitations that may be encountered for this project, it will be the usability of GPU mode to efficiently execute all experiments for the research project. Given that there may be resource limits on how long I can run the entire notebook in GPU mode, it is possible that I will need to shorten the models that I used, lower the number of units for each layer, or decrease the number of epochs for the training process.

For both Parts 1 and 2 of the experiment, unless specified otherwise, all models will be implemented using Adam as optimizer and mean squared error (or MSE) for loss, the input shape for the first layer will use (num_steps, 1), where num_steps is the number of time steps considered (which is 30 by default), and the first layer will use ReLU as activation function. All models will be trained for 100 epochs with the test set used for validation. While fitting the model, the parameter shuffle will always be set to true to ensure that the training data gets shuffled for each epoch.

2.1 Obtaining the Dataset

I begin the experiment by importing the csv file containing the ETH-USD historical price list obtained from Yahoo Finance [22] with pandas, a Python library that can handle dataset manipulation operations. I use head() to display the first number of rows for the dataset.

	Date	Open	High	Low	Close	Adj Close	Volume
0	2017-11-09	308.644989	329.451996	307.056000	320.884003	320.884003	893249984
1	2017-11-10	320.670990	324.717987	294.541992	299.252991	299.252991	885985984
2	2017-11-11	298.585999	319.453003	298.191986	314.681000	314.681000	842300992
3	2017-11-12	314.690002	319.153015	298.513000	307.907990	307.907990	1613479936
4	2017-11-13	307.024994	328.415009	307.024994	316.716003	316.716003	1041889984
5	2017-11-14	316.763000	340.177002	316.763000	337.631012	337.631012	1069680000
6	2017-11-15	337.963989	340.911987	329.812988	333.356995	333.356995	722665984
7	2017-11-16	333.442993	336.158997	323.605988	330.924011	330.924011	797254016
8	2017-11-17	330.166992	334.963989	327.523010	332.394012	332.394012	621732992
9	2017-11-18	331.980011	349.615997	327.687012	347.612000	347.612000	649638976
10	2017-11-19	347.401001	371.290985	344.739990	354.385986	354.385986	1181529984
11	2017-11-20	354.093994	372.136993	353.289001	366.730011	366.730011	807027008
12	2017-11-21	367.442993	372.470001	350.692993	360.401001	360.401001	949912000
13	2017-11-22	360.312012	381.420013	360.147003	380.652008	380.652008	800819008
14	2017-11-23	381.438995	425.548004	376.088013	410.165985	410.165985	1845680000

Figure 1: Screenshot of the first 15 rows of the ETH-USD price list.

To ensure that every data values are consistent throughout, isna().sum() and isna().any() are used to count how many missing values are there and verify if there are any instances of missing values. If there are any missing values present, then I'll need to follow up by dropping rows or columns that contains missing values.

```
Date          0
Open          0
High          0
Low           0
Close         0
Adj Close     0
Volume        0
dtype: int64
```

```
Date         False
Open         False
High         False
Low          False
Close        False
Adj Close    False
Volume       False
dtype: bool
```

Figure 2: The total number of missing values in the dataset after using isna().sum() (left) and the results of whether N/A is present in the dataset after using isna().any() (right).

Based on the result as shown in the screenshots above, there is not a single missing value present in the dataset, so no further data tidying needs to be performed. To illustrate how the dataset appears visually, a pyplot graph showing the closing price of ETH is created below.

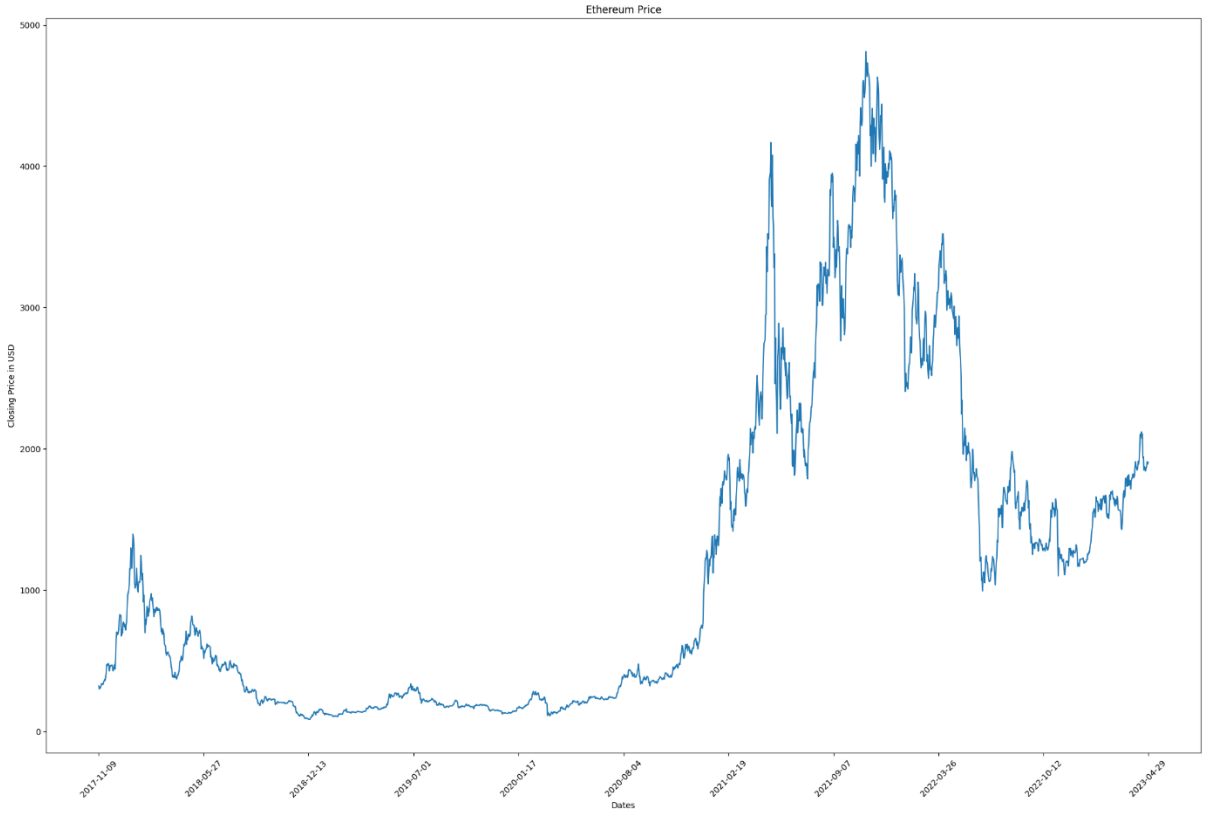


Figure 3: Plot of ETH's closing price over the years in USD.

Two plots are created with the same procedure as above: one with all prices within the year 2021 and the other within the year 2022.

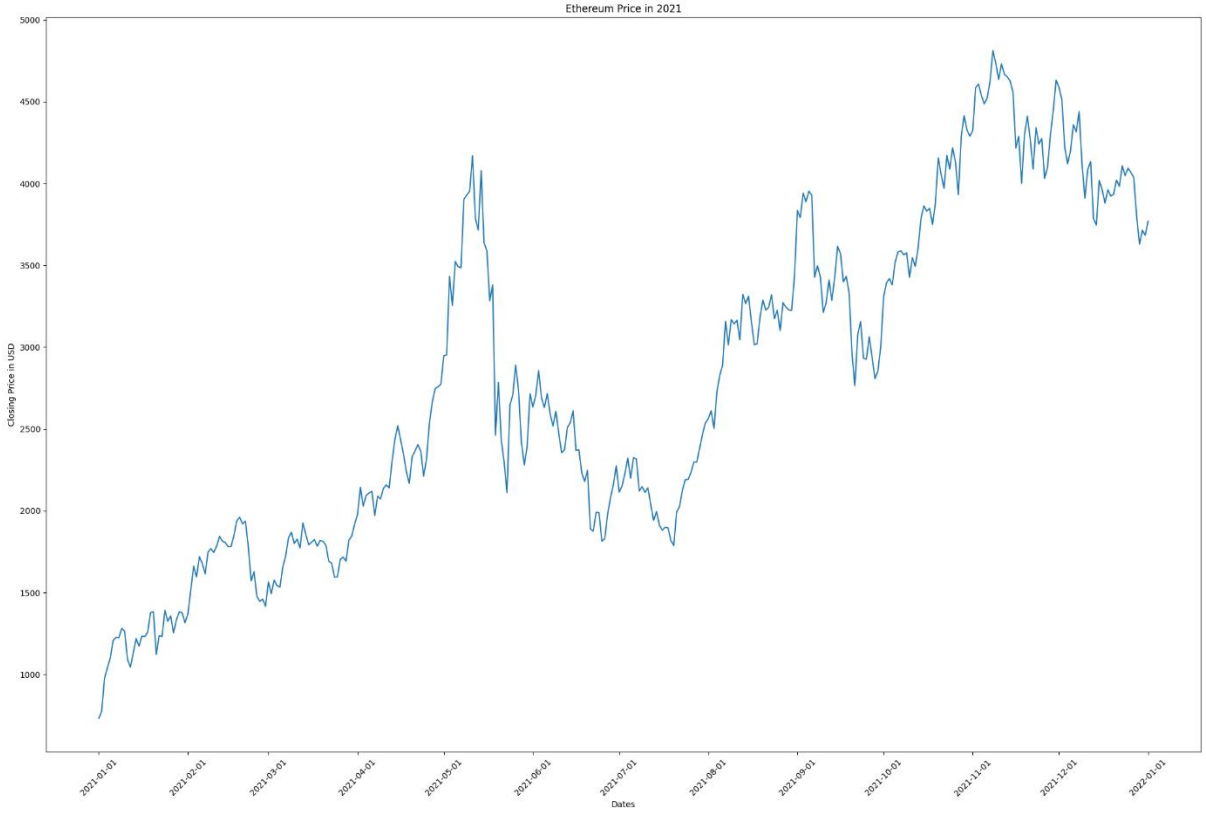


Figure 4: Plot of ETH's closing price in USD for year 2021.

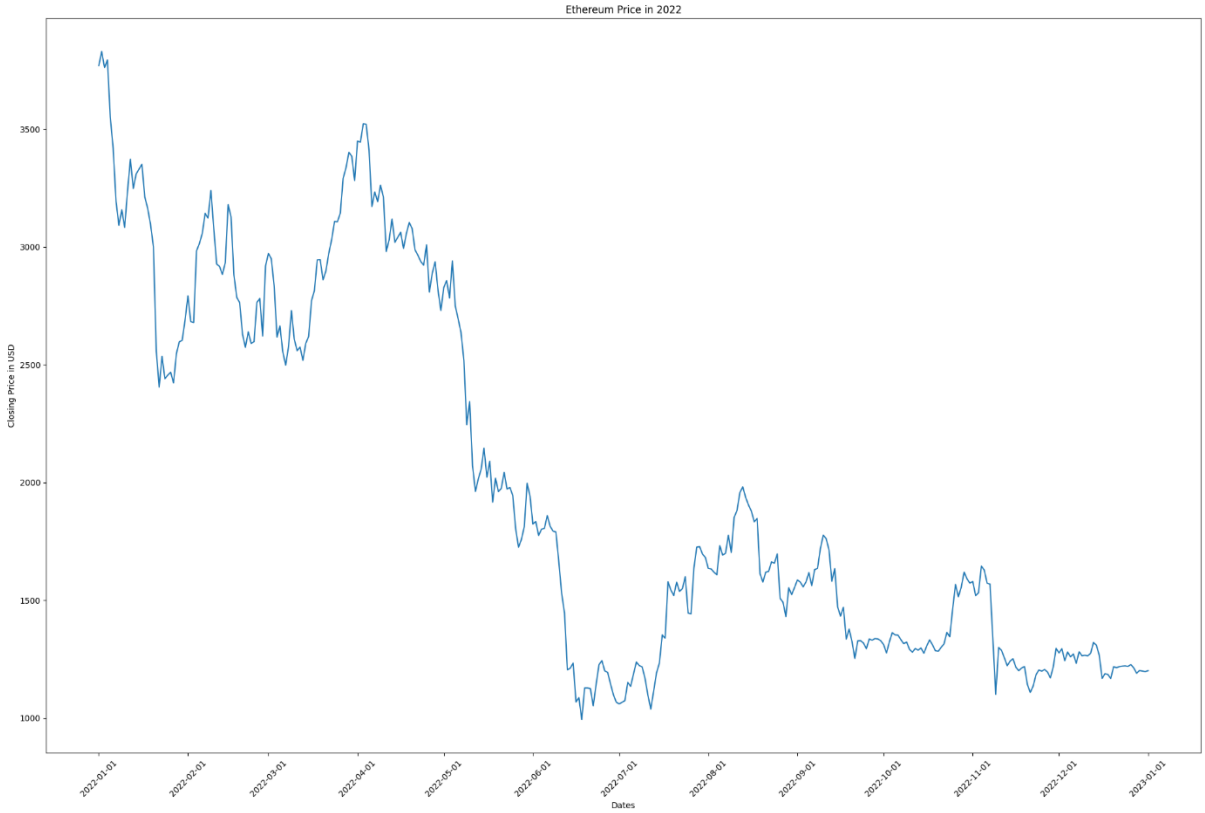


Figure 5: Plot of ETH's closing price in USD for year 2022.

2.2 Data Preprocessing

In this step, I create three different datasets by splitting them into two separate parts based on the training-test ratio: the 70-30 model, the 80-20 model, and the 75-25 model. I also created a fourth dataset of custom length setting aside the price list of Ethereum up to the March 31, 2022 as the training set and the price list of Ethereum starting at April 1, 2022 as the test set. As the length of the training/test data cannot have decimals, the resulting length is converted into an integer. For this experiment, the closing price of Ethereum will be used.

```
Shape of training and test data: (1604, 2) and (394, 2)
```

Figure 6: The shape of training and test data. The size of training data needs to be sufficiently large enough to minimize training loss and prevent overfitting.

The next step is to perform batch normalization of both the training and test data, and it is done by using min-max scalar from Scikit-learn. The formula for min-max scalar is defined as followed from a scientific article by Eesa and Arabo [23]:

$$nv = f(v) = \frac{v - \min(v)}{\max(v) - \min(v)}$$

Figure 7: The equation of the min-max scalar used for normalization. All values are normalized by subtracting each value from the minimum value followed by dividing over the difference between the maximum value and minimum value.

Once the values of training and test datasets are normalized, the features and labels for each dataset are established with time steps set to 30 (30 days are approximately equivalent to 1 month). Following that, the features and labels are converted into NumPy arrays.

```
Features of training data:  (1574, 30, 1)
Labels of training data:  (1574,)
Features of test data:  (338, 30, 1)
Labels of test data:  (338,)
```

Figure 8: The resulting shapes of the features and labels for training and test data. Note that an LSTM model requires a 3-dimensional input shape with the following parameters: (batch size, time steps, number of features).

2.3 Baseline LSTM Model

Layer Type	Properties
LSTM	Units=128, Activation=ReLU, Input_shape=(num_steps, 1)
LSTM	Units=128
LSTM	Units=128
LSTM	Units=128
LSTM	Units=128
Dense	Units=1

Table 1: The architecture summary of the LSTM model.

For all models, early stopping will be implemented to ensure that the performance metrics does not worsen after a certain number of epochs. In other words, training will halt after a number of epochs passed if the validation loss begins to increase at some point. Early stopping is a callback regularization that monitors any metric within the neural network model during the training process, and as soon as that metric cannot be improved further (such as loss beginning to increase at some point), the training process will halt. An example of implementation of early stopping is cross-validation, as shown in the figure below from a

paper by Gencay and Qi [24] analyzing the trends of training and validation errors over a certain number of iterations.

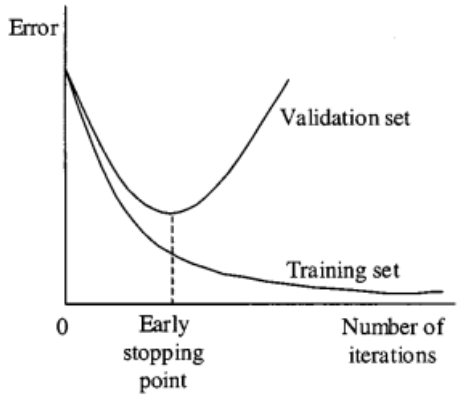


Figure 9: The graph showing the change of the training and validation errors over the number of iterations during the training process. Here, the early stopping point is marked across the x-axis to indicate that at that after a certain number of iterations, the validation error will increase after that point, so training will halt early if that occurs.

For the baseline model, it will be used to compare with the improved models, as described in the later sections. In other words, the question for this experiment is proposed as followed: Will adding one or more features to the baseline model minimize the validation and test loss after training and evaluating? Also, will any of the features listed below help improve the likelihood of predictions corresponding well with the actual dataset?

2.4 Improved LSTM model with features

Now that the baseline model is constructed, additional neural network features will be built on top of the existing one. The dropout rates and regularizers will be included separately in each model, followed by combining the two features together.

2.4.1 Dropout Rates

The dropout rate is introduced as a way to not only alleviate overfitting, but to also minimize the training and validation loss. According to Pham et al. [25], dropout involves freezing a percentage of hidden units in the layer during the training process, meaning a certain number of nodes will remain unchanged when training updates take place. This is a useful technique as it helps reduce overfitting while the model is training, which is a common challenge for most neural networks. Overfitting occurs when a neural network model trains well on the given training data but trains poorly on the data that isn't seen yet (like validation or test data). In other words, overfitting can occur if the training loss remains low, yet the validation

loss skyrockets to an abnormally high value. As noted from a scholarly article by Qian et al. [26], where it discusses about the techniques and methods that helps remedy overfitting, it is important that overfitting should be minimized as much as possible to ensure that the validation loss does not end up increasing much in a way that will affect the performance metrics of the neural networks like accuracy and loss. For this model below, a dropout rate of 20% is assigned to every LSTM layer.

Layer Type	Properties
LSTM	Units=128, Activation=ReLU, Input_shape=(num_steps, 1), Dropout=0.2
LSTM	Units=128, Dropout=0.2
LSTM	Units=128, Dropout=0.2
LSTM	Units=128, Dropout=0.2
LSTM	Units=128, Dropout=0.2
Dense	Units=1

Table 2: The architecture summary of the LSTM model with dropout.

2.4.2 Regularizers

Regularizers can also be implemented to reduce the likelihood of overfitting during training. There are two types of regularizers that are commonly used: the L1 regularizer and the L2 regularizer (or Lasso and Ridge. While L1 regularizer computes the loss by taking in the absolute values of each layer weights, the L2 regularizer rely on squared values of each layer weights to compute the loss. However, both L1 and L2 regularizers can be combined in conjunction to form the elastic net regularizer (or L1-L2 regularizer). Based on a journal from Pei et al. [27], elastic net regularizer can yield both benefits of what L1 and L2 regularizers can provide while also simultaneously reducing the resulting error after training. The model architecture will include an L1-L2 regularizer for the first LSTM layer.

Layer Type	Properties
LSTM	Units=128, Activation=ReLU, Input_shape=(num_steps, 1), Regularizer=L1_L2
LSTM	Units=128
LSTM	Units=128
LSTM	Units=128
LSTM	Units=128
Dense	Units=1

Table 3: The architecture summary of the LSTM model with regularizers.

2.4.3 Dropout Rates and Regularizers

For this model architecture, both the dropout rates and regularizers will be combined together, with the first layer using L1-L2 regularizer and all layers except the Dense layer using a dropout rate of 20%.

Layer Type	Properties
LSTM	Units=128, Activation=ReLU, Input_shape=(num_steps, 1), Dropout=0.2, Regularizer=L1_L2
LSTM	Units=128, Dropout=0.2
LSTM	Units=128, Dropout=0.2
LSTM	Units=128, Dropout=0.2
LSTM	Units=128, Dropout=0.2
Dense	Units=1

Table 4: The architecture summary of the LSTM model with dropout and regularizers.

3 Part 2: Comparison with Other Models

For Part 2 of the experiment, the procedures will be the same as Part 1, except in addition to using LSTM, other neural network models will be used for price prediction such as the multilayer perceptron (MLP), the convolutional neural network (CNN), and gated recurrent unit (GRU). Furthermore, more advanced hybrid models such as bidirectional LSTM and 1-dimensional CNN GRU networks will be built as well. Each model shall be explained and defined below. The only difference for Part 2 is that the training dataset will be all price lists before April 1, 2022, and the test dataset will use all price lists since April 1, 2022.

3.1 Multilayer Perceptron (MLP)

A multilayer perceptron is the simplest form of an artificial neural network, and it consists of multiple hidden layers of arbitrary counts located between input and output layers. Based on an article from Gao et al. [28], multilayer perceptrons can also be interpreted as a directed graph, where there are edges with arrows pointing from one node to another. In the case of multilayer perceptron models, the input data information will be processed starting from the input layer to multiple hidden layers, and ending at the output layer. Figure 10 below shows an example of how a multilayer perceptron can be interpreted visually.

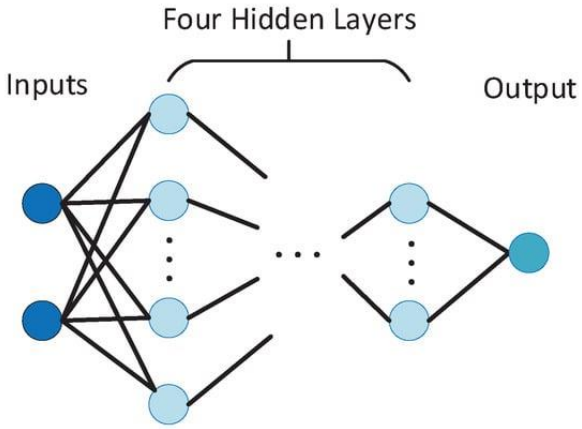


Figure 10: An example of a multilayer perceptron network. Here, there are four hidden layers added between the input and output layers.

One feature that multilayer perceptrons rely on is backpropagation. From Werbos [29], backpropagation relies on chain rules for all functions, which involves finding a derivative of one function for each term and then adding the resulting derivatives together. While chain rules remain a fundamental technique in the study of calculus, the same can be said in the realm of neural networks. In that case, the purpose of backpropagation in multilayer perceptrons is to obtain the gradient descent of the updates and adjust the weights of each

node when needed. That way, backpropagation can assist in helping MLP networks reach the optimal value during the training process.

For implementation of MLP models, the architecture will be entirely Dense layers, and all except the very last Dense layer will use tanh activation function.

3.2 Convolutional Neural Network (CNN)

Similar to an RNN model, a convolutional neural network (CNN) is a type of neural network that depends on spatial data and processes it as input as it feeds through the convolutional layers. Based on a paper from Li et al. [30], there are a wide variety of CNN models that can be implemented for any application domain, yet the common trait that most of the CNN models share are the three layers: convolutional, pooling, and fully connected. Each of those layers are described below based on their purposes:

- Convolutional layers read in the data as a matrix of arbitrary dimensional size and applies a kernel filter to it.
- Pooling layers are responsible for compressing the sizes of the spatial data obtained from the convolutional layers.
- A fully connected layer serves as the endmost portion of the CNN model. It obtains the spatial input and flattens it into a one-dimensional vector of data.

Figure 11 below shows an example of how a CNN model may resemble is seen below in an article from Liu et al. [31]:

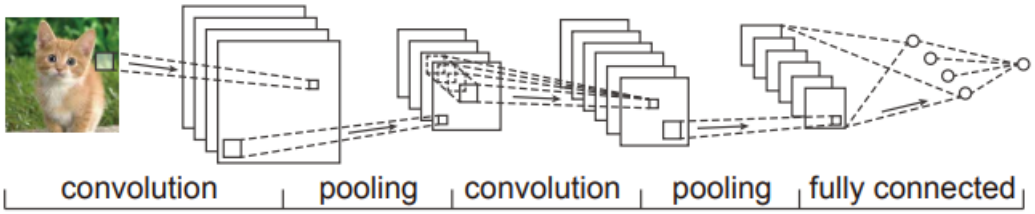


Figure 11: An overview of how the architecture of a CNN (convolutional neural network) model appears. Notice that while there can be any number of convolution and pooling layers in the model, the fully connected layer where the flattening operation takes place is always implemented at the very end of the model.

While CNN models are commonly implemented for 2-dimensional applications such as image classification, facial recognition, and object detection, CNNs can also be purposed in other dimensionalities, and examples include 1-dimension CNN (time-series price lists) and 3-dimension CNN (MRT scans of a human organ).

For all models using CNN layers, the kernel size and filter values will be 30 and 3. Following that, average pooling will be applied, and the flatten operation is applied right before the Dense layer with a unit of 1.

3.3 Gated Recurrent Unit (GRU)

Like LSTM, GRU is a type of RNN model that reads and analyzes temporal data fed through as input. Unlike LSTM, GRU is a simplified form of LSTM because it contains the update and reset gates in place of the input, forget, and output gates found in LSTM. As noted in the stock price prediction report by Sethia and Raut [32], GRU may outperform LSTM in terms of performance metrics given that the predicted results may correspond well with the actual data in some cases, if not all the time. Thus, GRU may behave more similar to LSTM, except that it is simpler and takes less time to train the entire dataset when obtaining results. Figure 12 below shows how the LSTM and GRU cells are defined according to their architecture according to Fu et al. [33]:

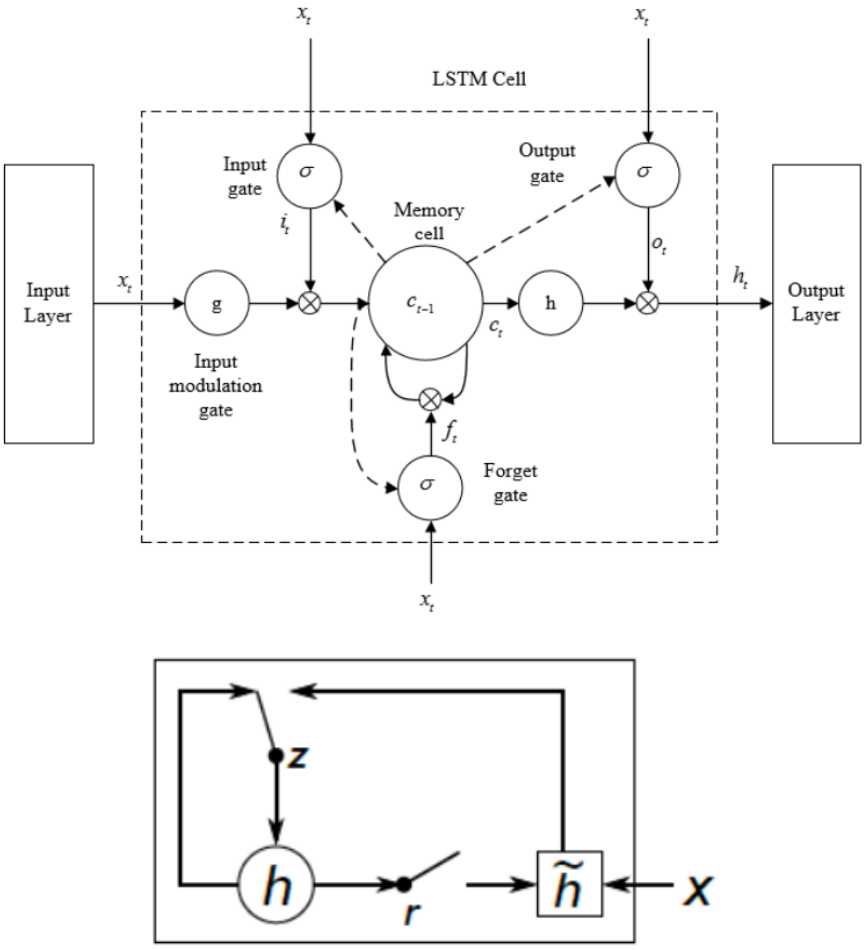


Figure 12: The cell diagrams of the LSTM (top) and GRU (bottom) models. Notice that the GRU cell architecture is more simplified than LSTM.

3.4 Bidirectional LSTM (BiLSTM)

If standard LSTM is a type of recurrent neural network where the time series only proceeds forward in one direction, bidirectional LSTM involves two-way time series where time steps can move forward and backward simultaneously. Figure 13 below shows an example of how a bidirectional LSTM model can be defined based on an IEEE workshop conference paper by Graves et al. where it analyzes speech recognition [34]:

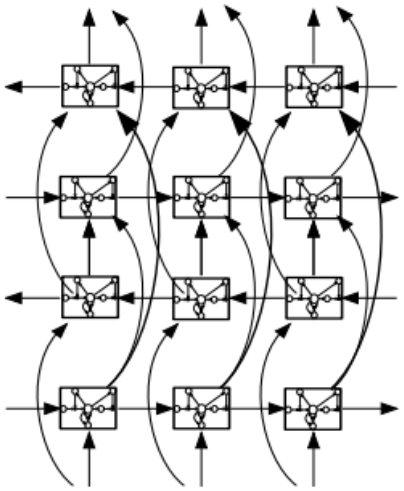


Figure 13: The architecture of how a bidirectional LSTM model appears. Notice that the sequential flow is moving in two directions: forward (which is the default direction in standard LSTM models) and backward.

3.5 Bidirectional GRU (BiGRU)

Similar to the case of LSTM, GRU can also be implemented as bidirectional, where the sequence flows through the entire model both in forward and backward direction. Aside from the differences between the two architecutres, the operations for bidirectional GRU is similar to bidirectional LSTM. One example where a bidirectional GRU model is implemented is an experiment on recurrent neural networks for electrocardiogram classification from a journal article by Lynn et al. [35].

3.6 Hybrid Models

As well as all of the models listed above, hybrid models will also be implemented by combining neural network architectures together. The following examples of hybrid neural network models can be listed based on the related works from other authors as followed: a hybrid CNN-LSTM model that analyzes human activity and behavior from an online journal by Zhu et al. [36], a hybrid CNN-GRU model that measures and predicts the rates where electric energy is consumed by residential households from Sajjid et al. [37], and a GRU-LSTM hybrid network that forecasts the foreign exchange rates of two currencies from Islam and Hossain [38]. The hybrid models that will be considered for this experiment are 1D-CNN

LSTM, 1D-CNN GRU, 1D-CNN BiLSTM, 1D-CNN BiGRU, LSTM-GRU, and 1D-CNN LSTM-GRU. Each of the hybrid models are described below as followed:

- 1D-CNN LSTM, 1D-CNN GRU, 1D-CNN BiLSTM, and 1D-CNN BiGRU: Three Conv1D layers will be added first, followed by AveragePooling1D layer. After that, 3 LSTM layers will be added, followed by the Flatten operation and the single-unit Dense layer. (The same design applies for GRU, BiLSTM and BiGRU.)
- LSTM-GRU: This architecture consists of 3 LSTM layers followed by 2 GRU layers.
- 1D-CNN LSTM-GRU: The model architecture is the same as LSTM-GRU, except 3 Conv1D layers and AveragePooling1D are implemented before LSTM layers, and Flatten is called after the very last GRU layer.

3.7 Model Implementations and Architectures
(For the sake of brevity and less redundance, only the baseline model architectures of all neural networks except LSTM are shown.)

Layer Type	Properties
Dense	Units=128, Activation=tanh, Input_shape=(num_steps,)
Dense	Units=128, Activation=tanh
Dense	Units=128, Activation=tanh
Dense	Units=128, Activation=tanh
Dense	Units=128, Activation=tanh
Dense	Units=1

Table 5: The architecture summary of the baseline MLP model.

Layer Type	Properties
Conv1D	Filters=30, Kernel_size=3, Activation=ReLU, input_shape=(num_steps, 1)
Conv1D	Filters=30, Kernel_size=3
Conv1D	Filters=30, Kernel_size=3
Conv1D	Filters=30, Kernel_size=3
Conv1D	Filters=30, Kernel_size=3
AveragePooling1D	N/A
Flatten	N/A
Dense	Units=1

Table 6: The architecture summary of the baseline CNN model.

Layer Type	Properties
GRU	Units=128, Activation=ReLU, Input_shape=(num_steps, 1)
GRU	Units=128
GRU	Units=128
GRU	Units=128
GRU	Units=128
Dense	Units=1

Table 7: The architecture of the baseline GRU model.

Layer Type	Properties
LSTM (Bidirectional)	Units=128, Activation=ReLU, Input_shape=(num_steps, 1)
LSTM (Bidirectional)	Units=128
LSTM (Bidirectional)	Units=128
LSTM (Bidirectional)	Units=128
LSTM (Bidirectional)	Units=128,
Dense	Units=1

Table 8: The architecture of the baseline bidirectional LSTM model.

Layer Type	Properties
GRU (Bidirectional)	Units=128, Activation=ReLU, Input_shape=(num_steps, 1)
GRU (Bidirectional)	Units=128
GRU (Bidirectional)	Units=128
GRU (Bidirectional)	Units=128
GRU (Bidirectional)	Units=128
Dense	Units=1

Table 9: The architecture of the baseline bidirectional GRU model.

Layer Type	Properties
Conv1D	Filters=30, Kernel_size=3, Activation=ReLU, input_shape=(num_steps, 1)
Conv1D	Filters=30, Kernel_size=3
Conv1D	Filters=30, Kernel_size=3
AveragePooling1D	N/A
LSTM	Units=128, Activation=ReLU
LSTM	Units=128
LSTM	Units=128
Flatten	N/A
Dense	Units=1

Table 10: The architecture of the baseline 1-D CNN LSTM hybrid model.

Layer Type	Properties
Conv1D	Filters=30, Kernel_size=3, Activation=ReLU, input_shape=(num_steps, 1)
Conv1D	Filters=30, Kernel_size=3
Conv1D	Filters=30, Kernel_size=3
AveragePooling1D	N/A
GRU	Units=128, Activation=ReLU
GRU	Units=128
GRU	Units=128
Flatten	N/A
Dense	Units=1

Table 11: The architecture of the baseline 1-D CNN GRU hybrid model.

Layer Type	Properties
Conv1D	Filters=30, Kernel_size=3, Activation=ReLU, input_shape=(num_steps, 1)
Conv1D	Filters=30, Kernel_size=3
Conv1D	Filters=30, Kernel_size=3
AveragePooling1D	N/A
LSTM (Bidirectional)	Units=128, Activation=ReLU
LSTM (Bidirectional)	Units=128
LSTM (Bidirectional)	Units=128
Flatten	N/A
Dense	Units=1

Table 12: The architecture of the baseline 1-D CNN BiLSTM hybrid model.

Layer Type	Properties
Conv1D	Filters=30, Kernel_size=3, Activation=ReLU, input_shape=(num_steps, 1)
Conv1D	Filters=30, Kernel_size=3
Conv1D	Filters=30, Kernel_size=3
AveragePooling1D	N/A
GRU (Bidirectional)	Units=128, Activation=ReLU
GRU (Bidirectional)	Units=128
GRU (Bidirectional)	Units=128
Flatten	N/A
Dense	Units=1

Table 13: The architecture of the baseline 1-D CNN BiGRU hybrid model.

Layer Type	Properties
LSTM	Units=128, Activation=ReLU, Input_shape=(num_steps, 1)
LSTM	Units=128
LSTM	Units=128
GRU	Units=128, Activation=ReLU
GRU	Units=128
Dense	Units=1

Table 14: The architecture of the baseline LSTM GRU hybrid model.

Layer Type	Properties
Conv1D	Filters=30, Kernel_size=3, Activation=ReLU, input_shape=(num_steps, 1)
Conv1D	Filters=30, Kernel_size=3
Conv1D	Filters=30, Kernel_size=3
AveragePooling1D	N/A
LSTM	Units=128, Activation=ReLU
LSTM	Units=128
LSTM	Units=128
GRU	Units=128, Activation=ReLU
GRU	Units=128
Flatten	N/A
Dense	Units=1

Table 15: The architecture of the baseline 1-D CNN LSTM GRU hybrid model.

4 Research Results and Analysis

4.1 Part 1 Results

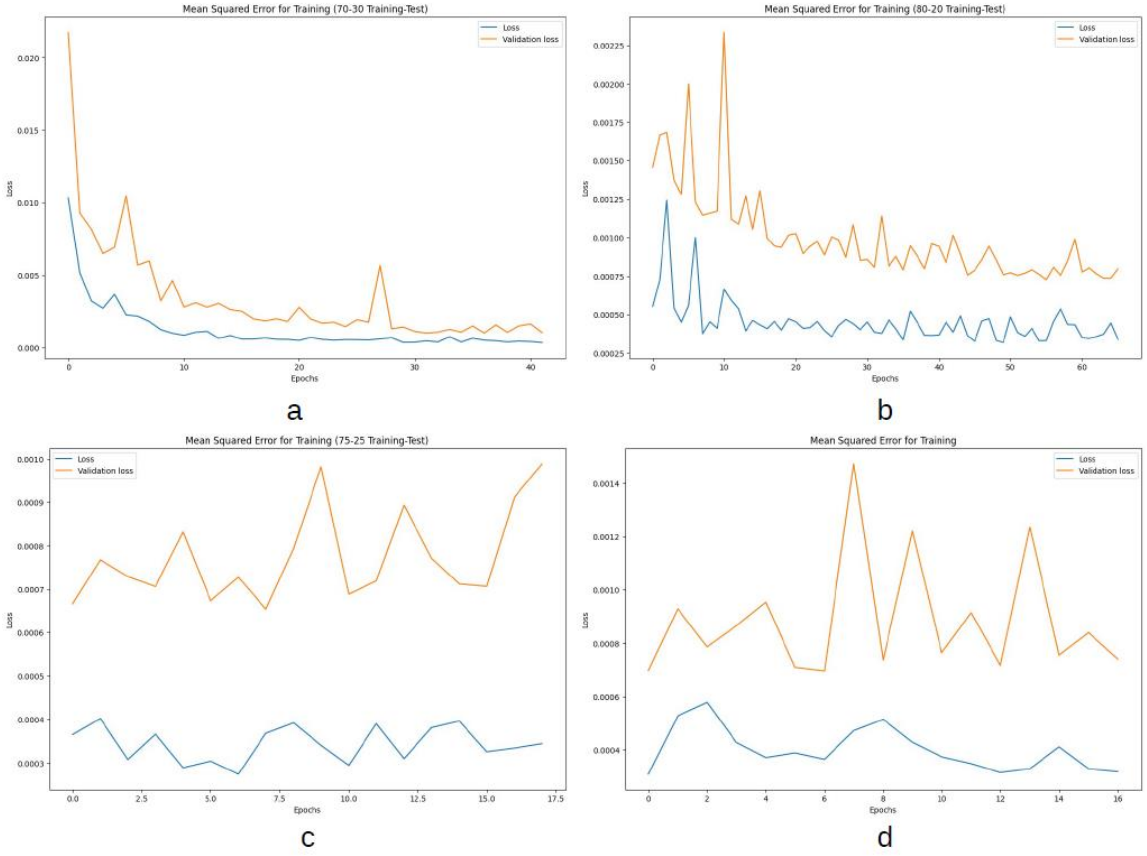


Figure 14: Training losses for baseline LSTM model (a: 70-30 training-test, b: 80-20 training-test, c: 75-25 training-test, d: custom training-test).

In Figure 14b, the 80-20 training-test dataset has shown a sharp spike in the training and validation losses within the first 10 epochs.

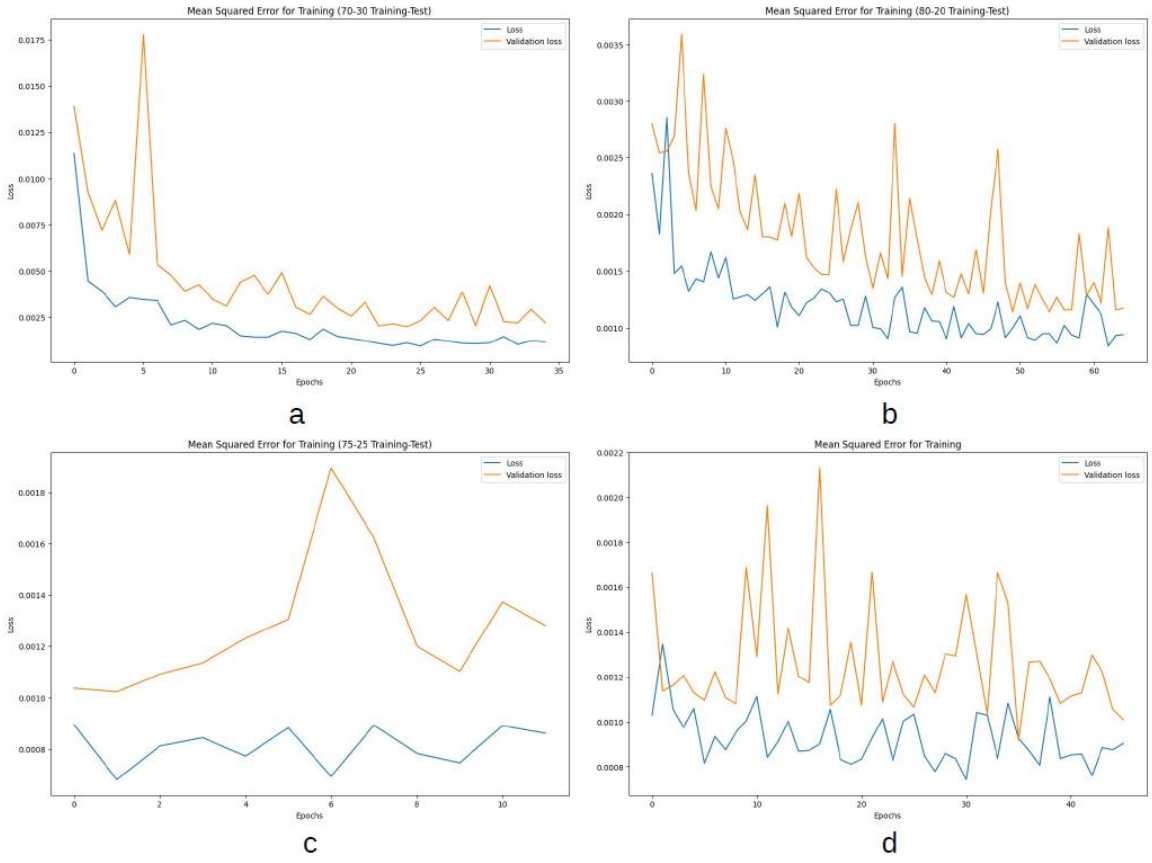


Figure 15: Training loss for LSTM model with dropout rates (a: 70-30 training-test, b: 80-20 training-test, c: 75-25 training-test, d: custom training-test).

Figures 15b and 15d has shown massive variance in the training losses fluctuating between the extreme high and low values, as well as the validation losses.

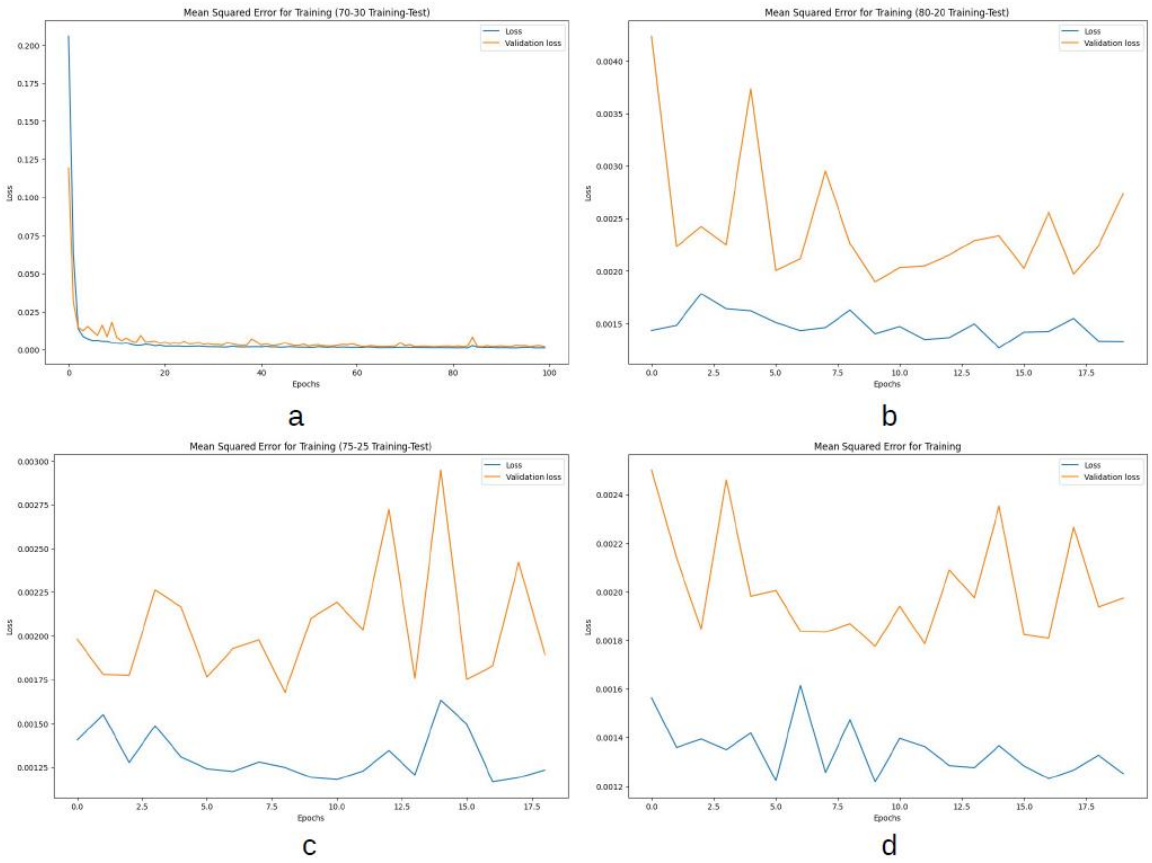


Figure 16: Training loss for LSTM model with regularizers (a: 70-30 training-test, b: 80-20 training-test, c: 75-25 training-test, d: custom training-test).

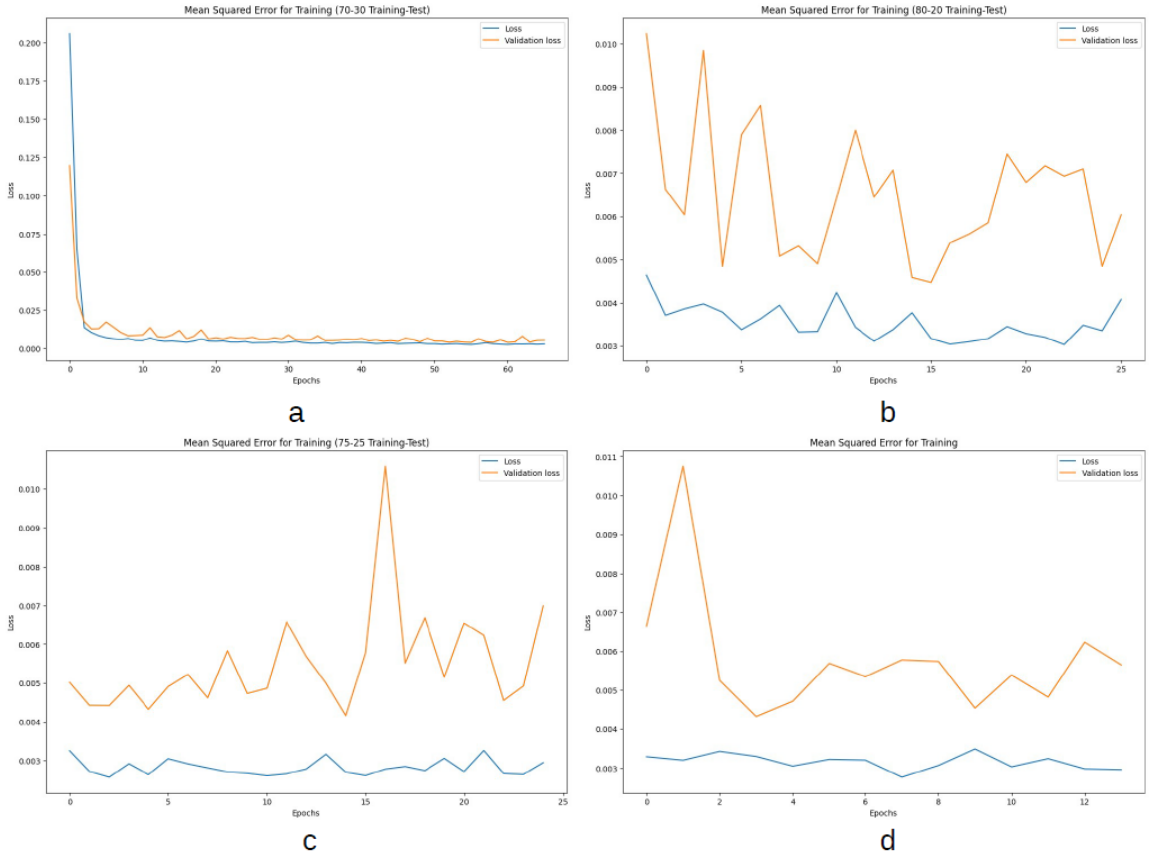


Figure 17: Training loss for LSTM model with dropout rates and regularizers (a: 70-30 training-test, b: 80-20 training-test, c: 75-25 training-test, d: custom training-test).

Figures 16 and 17 shows that the 70-30 training-test dataset maintains a downward curve for both the training and validation losses. All other datasets do not necessarily follow the trend demonstrated by the the 70-30 dataset.

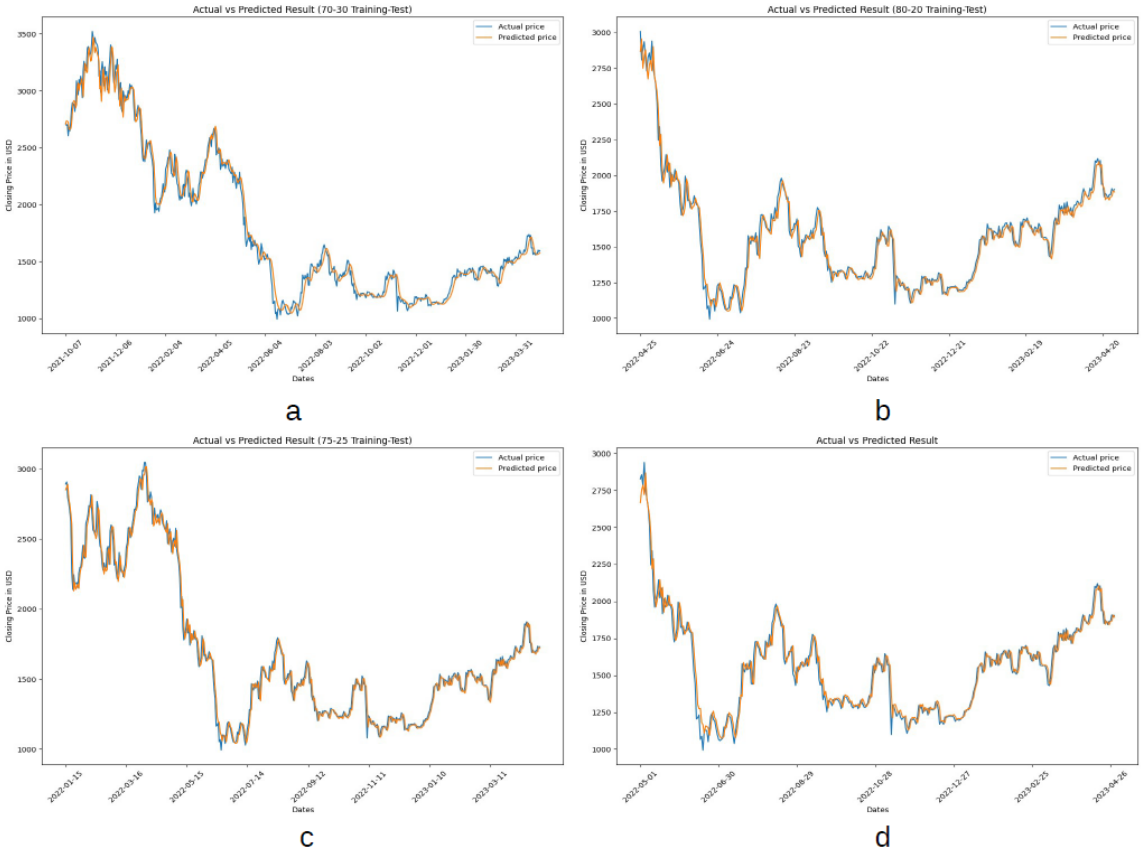


Figure 18: Plots of test set for predicted values of closing price compared with actual values for baseline LSTM model (a: 70-30 training-test, b: 80-20 training-test, c: 75-25 training-test, d: custom training-test).

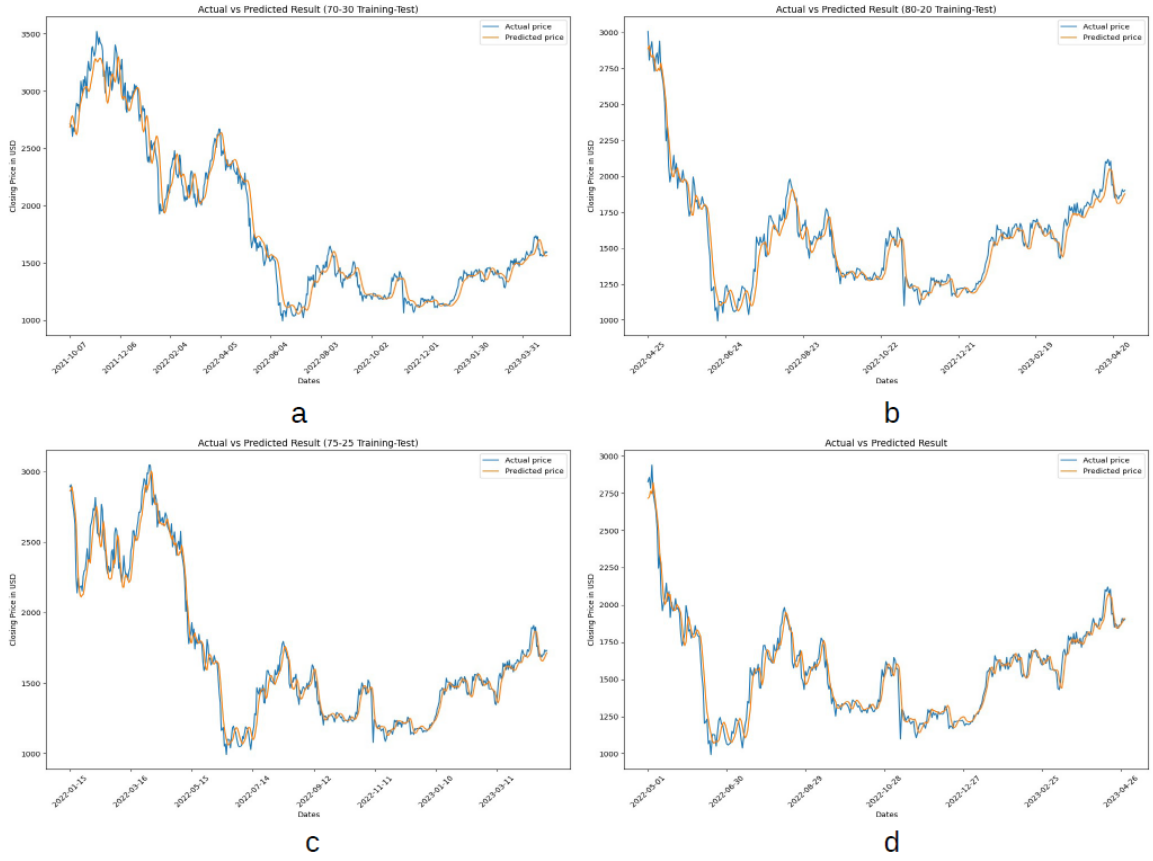


Figure 19: Plots of test set for predicted values of closing price compared with actual values for LSTM model with dropout rates (a: 70-30 training-test, b: 80-20 training-test, c: 75-25 training-test, d: custom training-test).

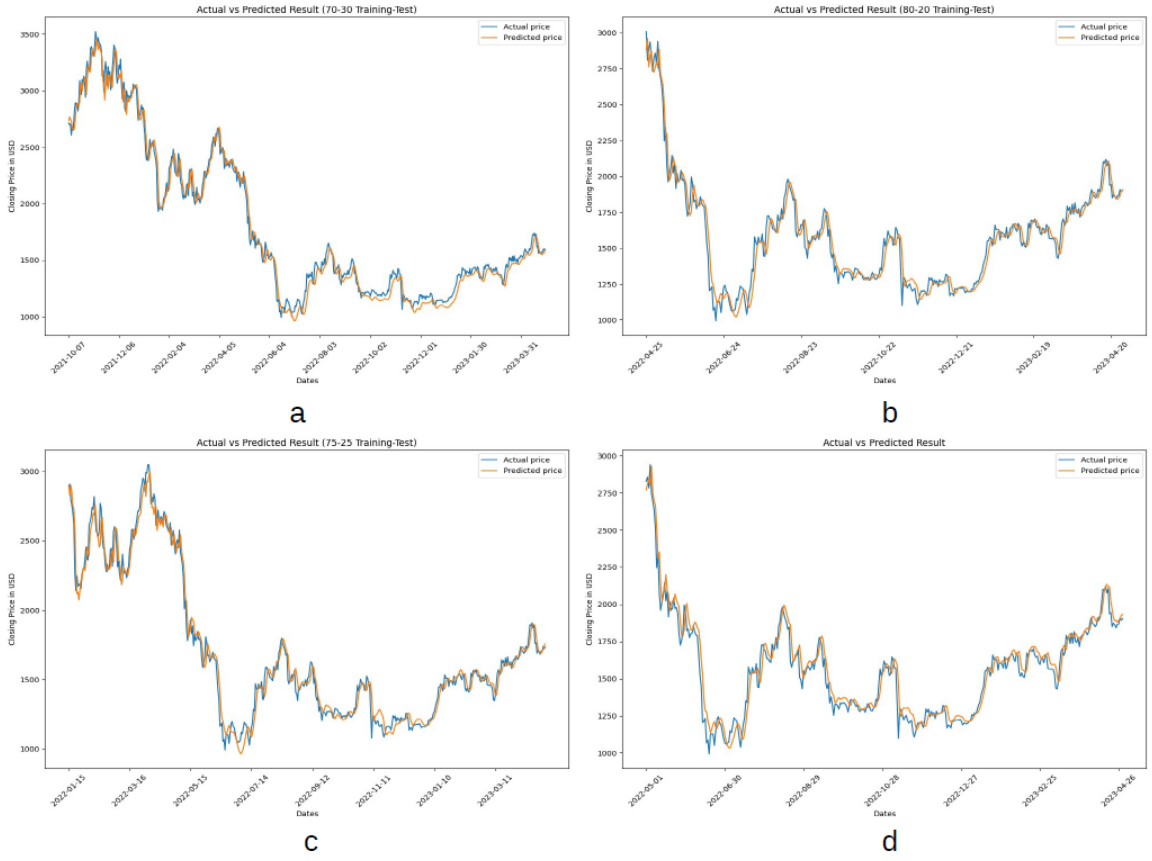


Figure 20: Plots of test set for predicted values of closing price compared with actual values for LSTM model with regularizers (a: 70-30 training-test, b: 80-20 training-test, c: 75-25 training-test, d: custom training-test).

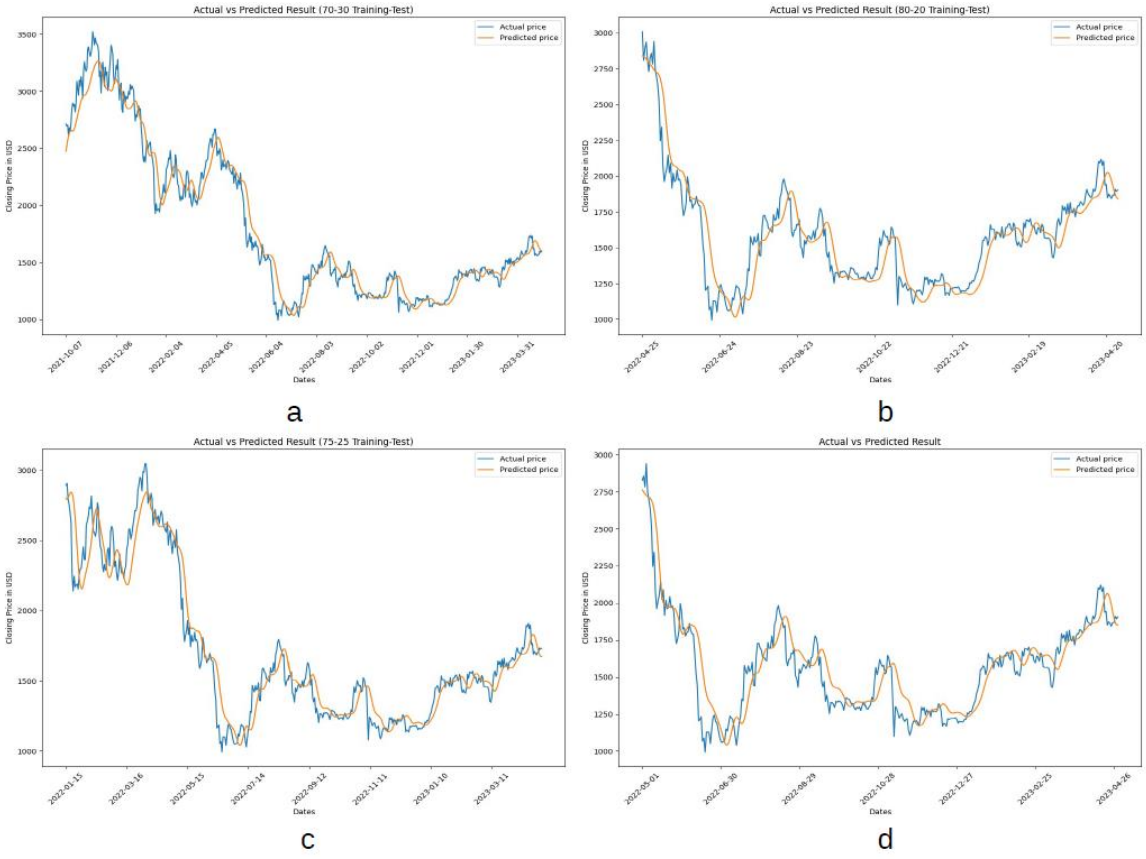


Figure 21: Plots of test set for predicted values of closing price compared with actual values for LSTM model with dropout rates and regularizers (a: 70-30 training-test, b: 80-20 training-test, c: 75-25 training-test, d: custom training-test).

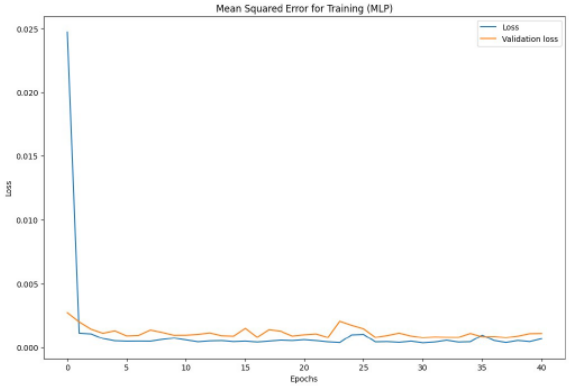
Figures 18 to 21 shows the predicted plots of ETH compared with the actual closing prices based on the features added. The predicted plots are more likely to be misaligned and not correlate well with the actual plots in LSTM models using dropout rates and regularizers than the baseline models. Meanwhile, LSTM models using either the dropout rates or regularizers tend to fall in between the baseline LSTM and the LSTM with dropout rates and regularizers in terms of how well the predictions align well with the actual results.

Features	Training-Test	MSE	MAE	RMSE
Baseline	70-30	0.00101	0.02319	0.03186
	80-20	0.00072	0.01890	0.02695
	75-25	0.00065	0.01777	0.02557
	custom	0.00069	0.01821	0.02638
Dropout	70-30	0.00200	0.03259	0.04479
	80-20	0.00113	0.02430	0.03375
	75-25	0.00102	0.02286	0.03198
	custom	0.00091	0.02102	0.03030
Regularizers	70-30	0.00099	0.02440	0.03155
	80-20	0.00095	0.02189	0.03092
	75-25	0.00089	0.02196	0.02984
	custom	0.00101	0.02264	0.03184
Both	70-30	0.00271	0.03846	0.05214
	80-20	0.00319	0.04059	0.05650
	75-25	0.00300	0.03882	0.05483
	custom	0.00306	0.03969	0.05536

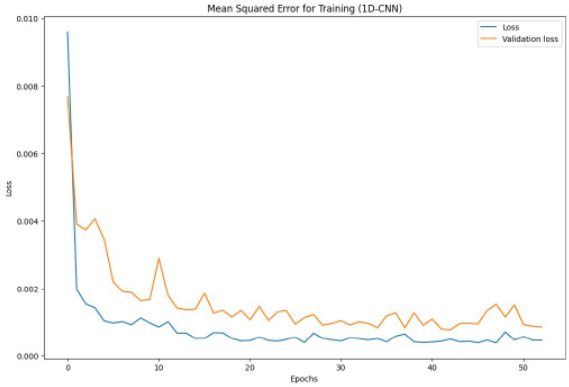
Table 16: The comparison of LSTM models' MSE (mean squared error), MAE (mean absolute error), and RMSE (root mean squared error) based on model feature and training-test ratio.

Upon closer look of the loss values obtained for each model, adding either the dropout rates or regularizers for each model have caused the loss values to increase. (One noted exception is the 70-30 model using the regularizers, where the MSE and RMSE values have slightly decreased.) Adding both the dropout rates and regularizers have caused the loss values to increase as well. Thus, LSTM models with dropout rates and regularizers tend to fare worse than all other LSTM models.

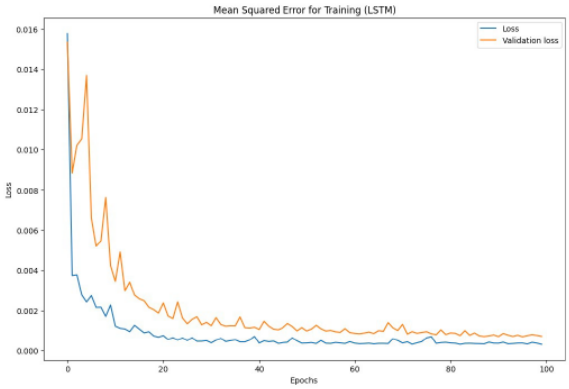
4.2 Part 2 Results



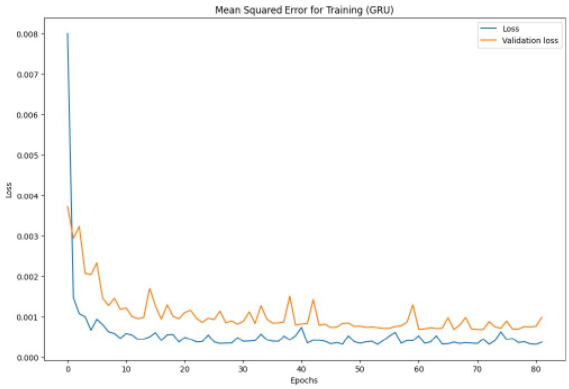
a



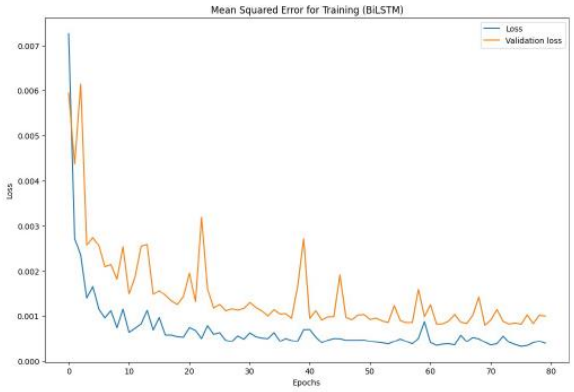
b



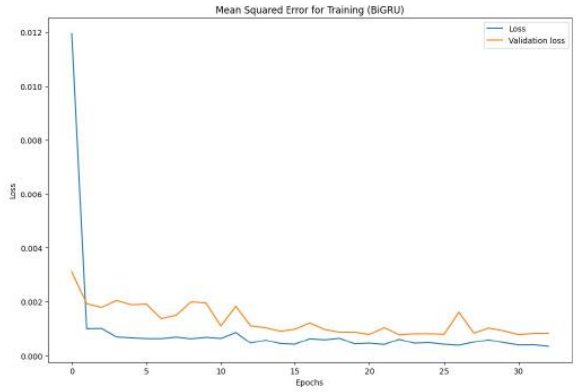
c



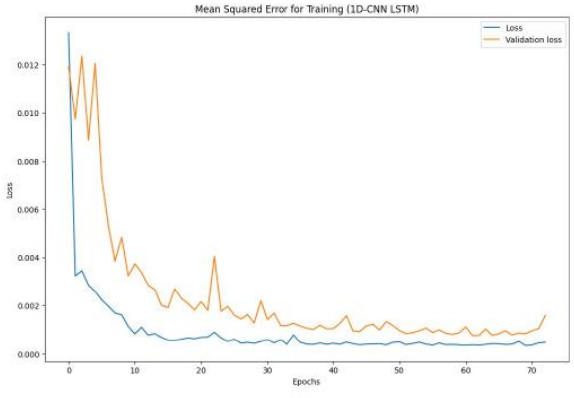
d



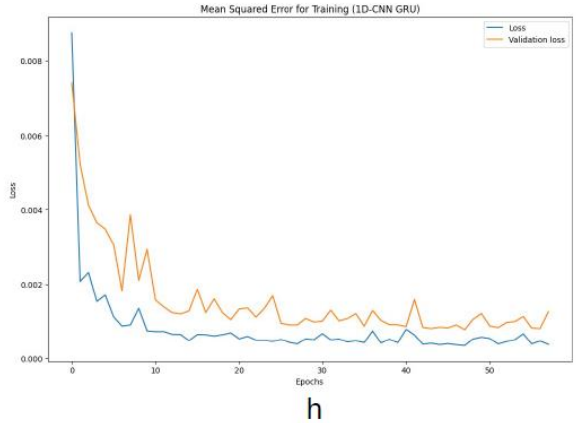
e



f



g



h

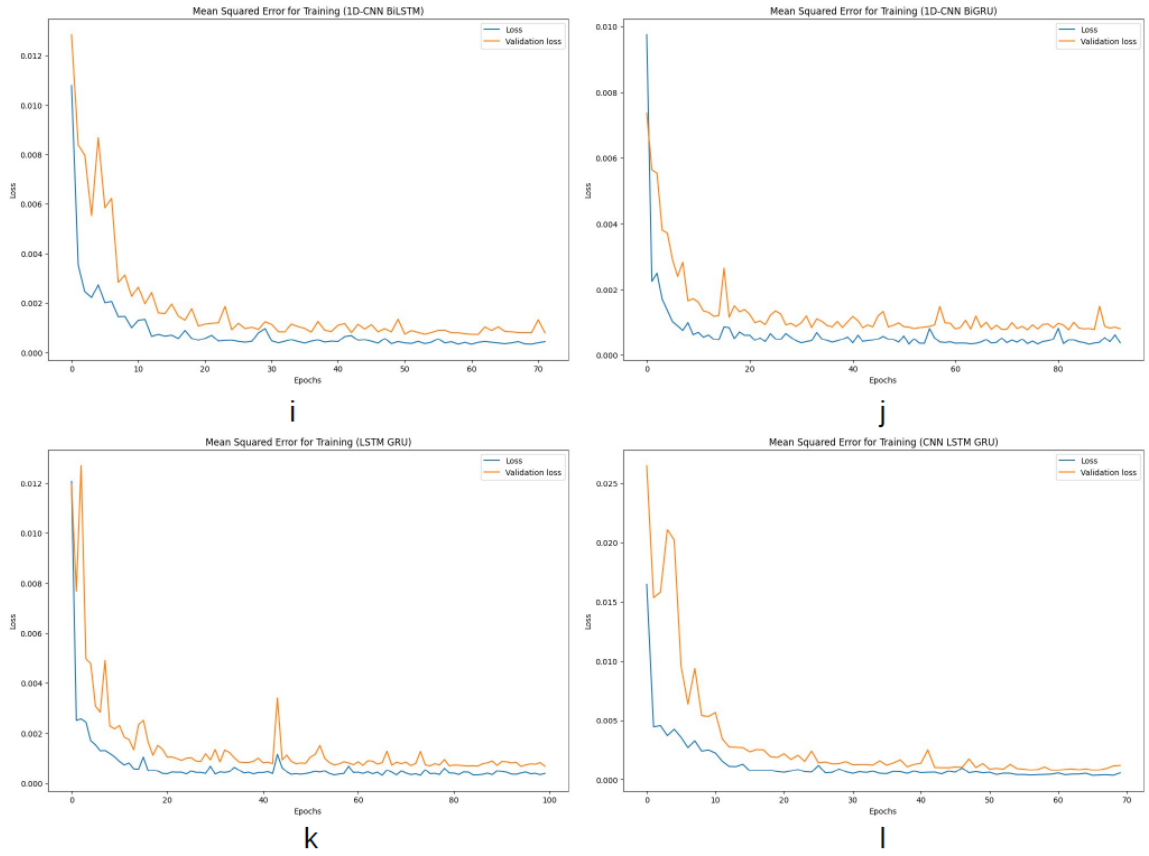
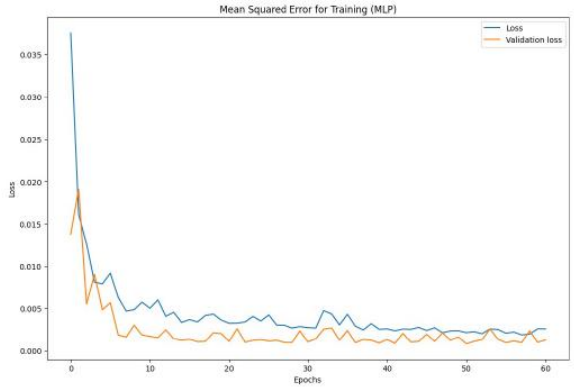
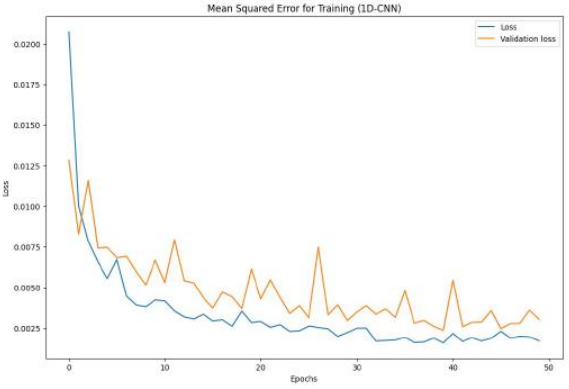


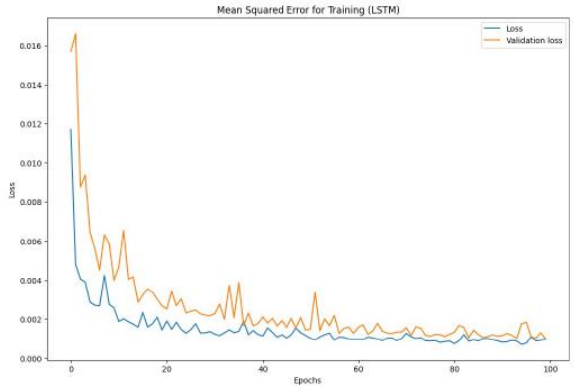
Figure 22: Training loss for baseline models (a: multilayer perceptron (MLP), b: convolutional neural network (CNN), c: long short-term memory (LSTM), d: gated recurrent unit (GRU), e: bidirectional LSTM (BiLSTM), f: bidirectional GRU (BiGRU), g: CNN-LSTM, h: CNN-GRU, i: CNN-BiLSTM, j: CNN-BiGRU, k: LSTM-GRU, l: CNN-LSTM-GRU).


Mean Squared Error for Training (MLP)
Loss
Validation loss
Loss
Epochs

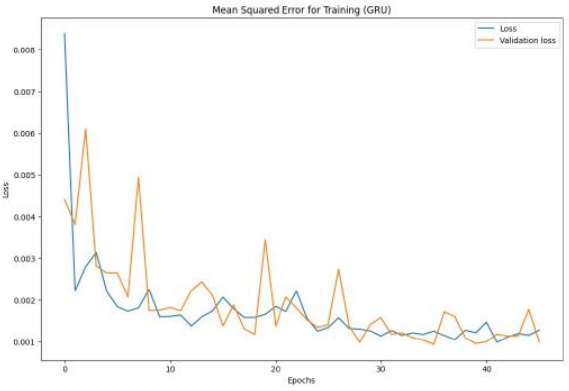
a


Mean Squared Error for Training (1D-CNN)
Loss
Validation loss
Loss
Epochs

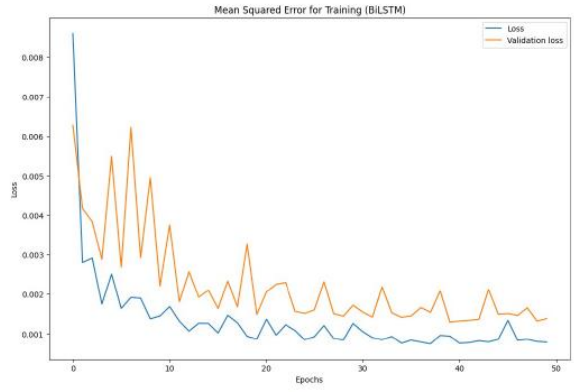
b


Mean Squared Error for Training (LSTM)
Loss
Validation loss
Loss
Epochs

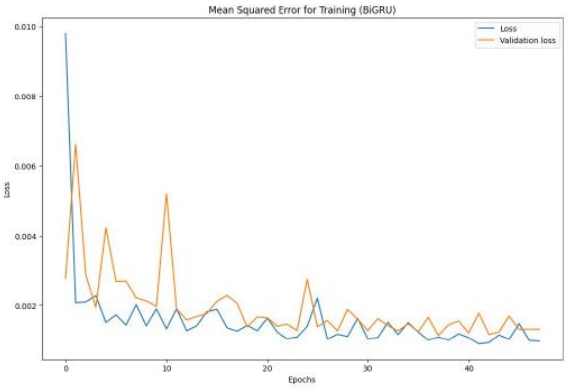
c


Mean Squared Error for Training (GRU)
Loss
Validation loss
Loss
Epochs

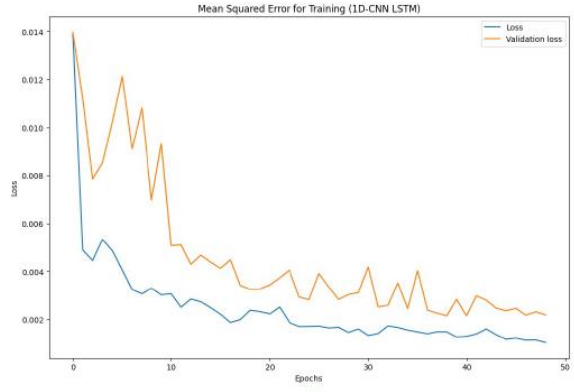
d


Mean Squared Error for Training (BiLSTM)
Loss
Validation loss
Loss
Epochs

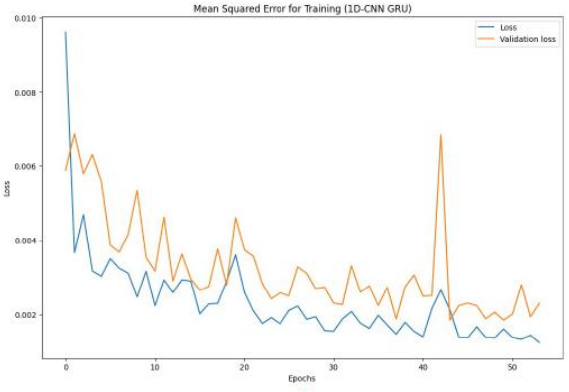
e


Mean Squared Error for Training (BiGRU)
Loss
Validation loss
Loss
Epochs

f


Mean Squared Error for Training (1D-CNN LSTM)
Loss
Validation loss
Loss
Epochs

g


Mean Squared Error for Training (1D-CNN GRU)
Loss
Validation loss
Loss
Epochs

h

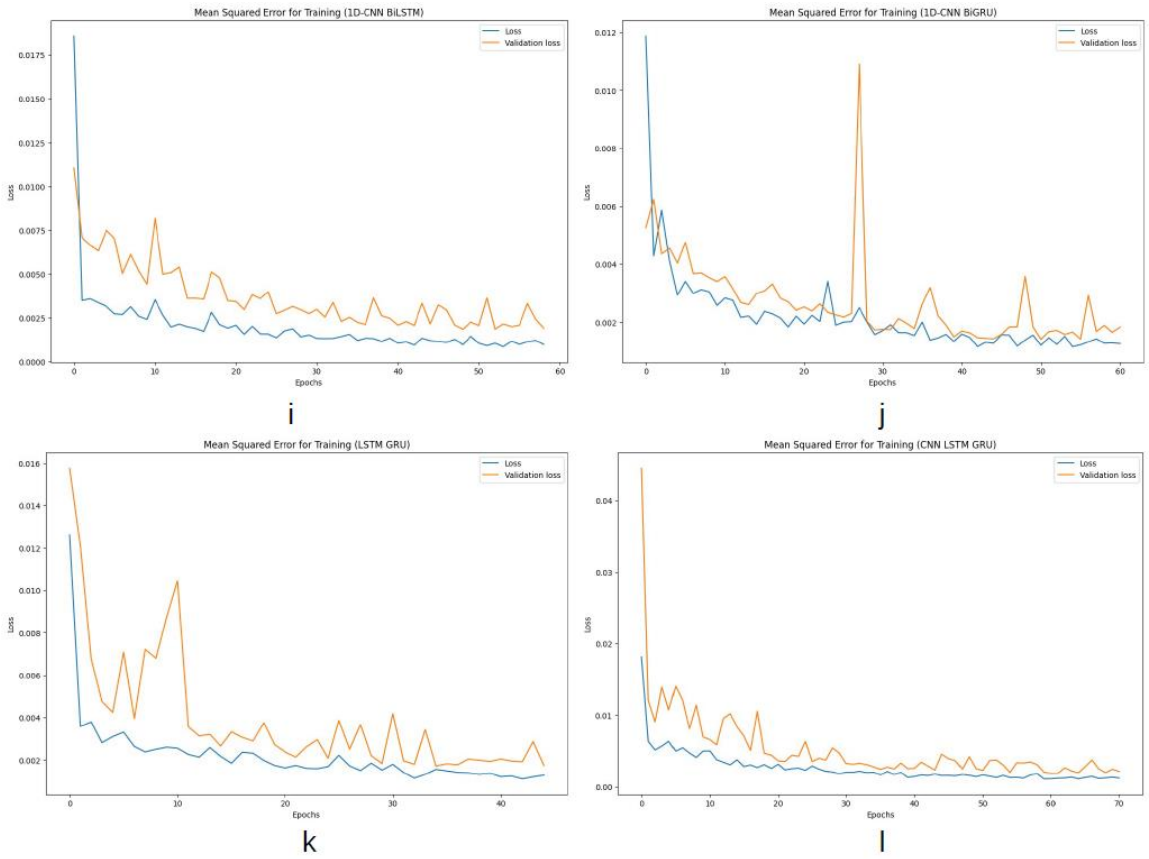
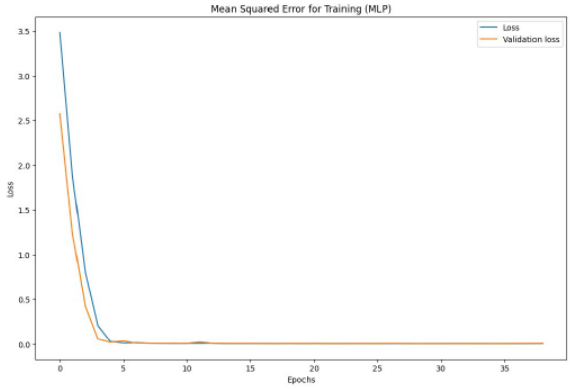
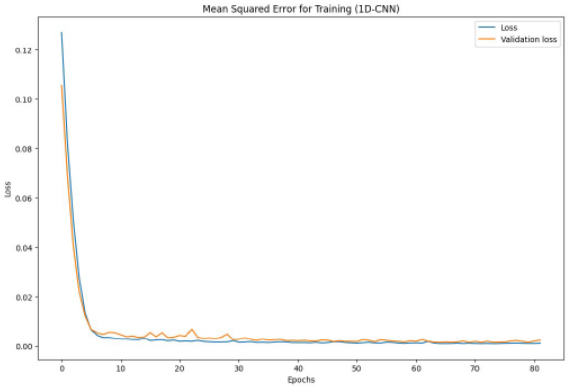


Figure 23: Training loss for models with dropout rates (a: multilayer perceptron (MLP), b: convolutional neural network (CNN), c: long short-term memory (LSTM), d: gated recurrent unit (GRU), e: bidirectional LSTM (BiLSTM), f: bidirectional GRU (BiGRU), g: CNN-LSTM, h: CNN-GRU, i: CNN-BiLSTM, j: CNN-BiGRU, k: LSTM-GRU, l: CNN-LSTM-GRU).

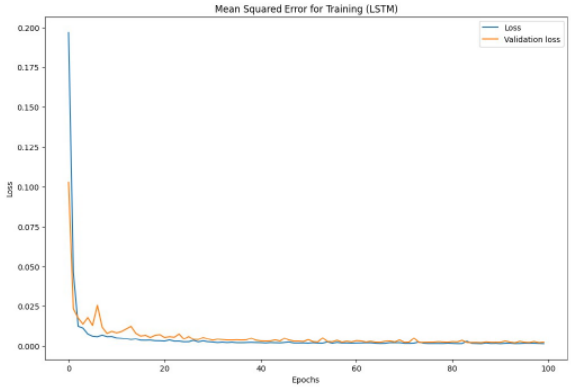
As noted in Figure 23a, the validation loss in the MLP ends up below the training loss, which is an unusual pattern out of all other models where validation loss hovers above the training loss. Most of the figures have shown varying levels of variance in the training losses, and the most notable methodologies that experienced significant spikes in the losses are GRU, BiLSTM, CNN-GRU, and CNN-BiGRU.


Mean Squared Error for Training (MLP)
Loss
Validation loss
Loss
Epochs

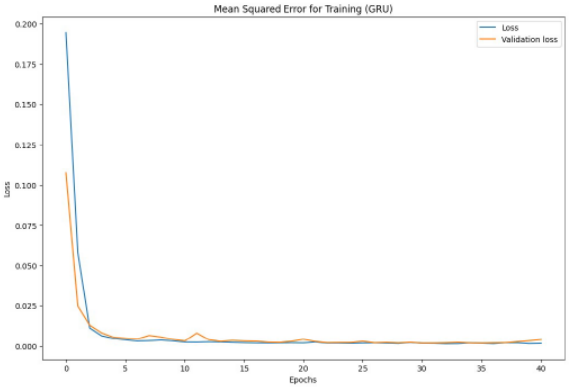
a


Mean Squared Error for Training (1D-CNN)
Loss
Validation loss
Loss
Epochs

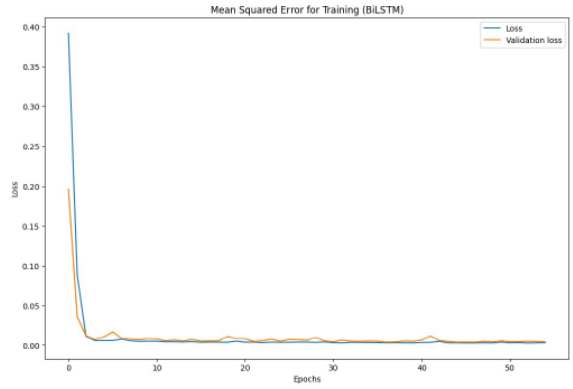
b


Mean Squared Error for Training (LSTM)
Loss
Validation loss
Loss
Epochs

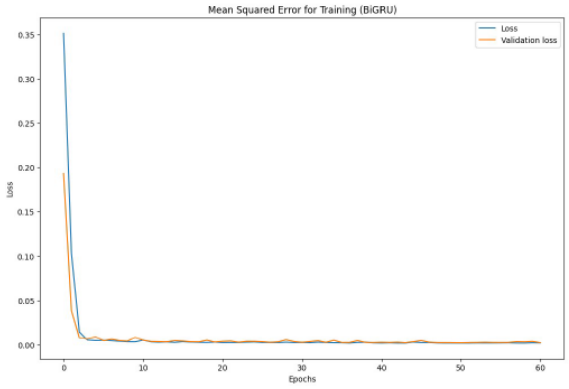
c


Mean Squared Error for Training (GRU)
Loss
Validation loss
Loss
Epochs

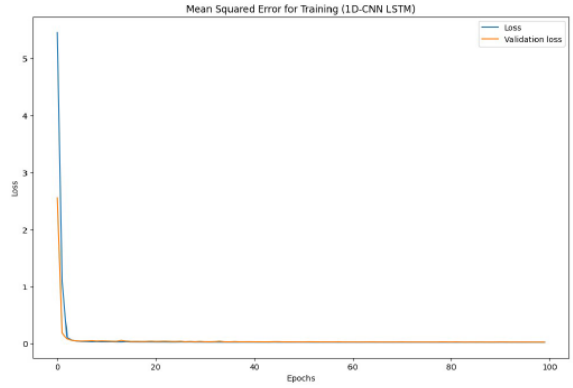
d


Mean Squared Error for Training (BiLSTM)
Loss
Validation loss
Loss
Epochs

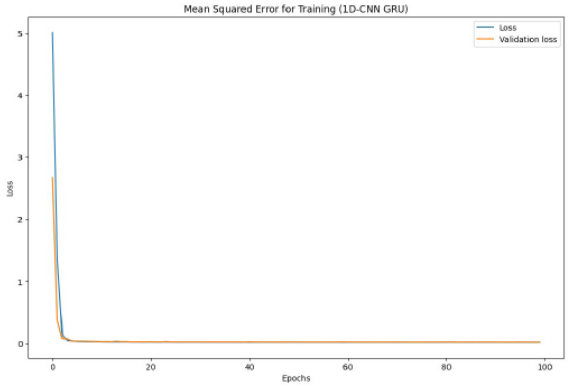
e


Mean Squared Error for Training (BiGRU)
Loss
Validation loss
Loss
Epochs

f


Mean Squared Error for Training (1D-CNN LSTM)
Loss
Validation loss
Loss
Epochs

g


Mean Squared Error for Training (1D-CNN GRU)
Loss
Validation loss
Loss
Epochs

h

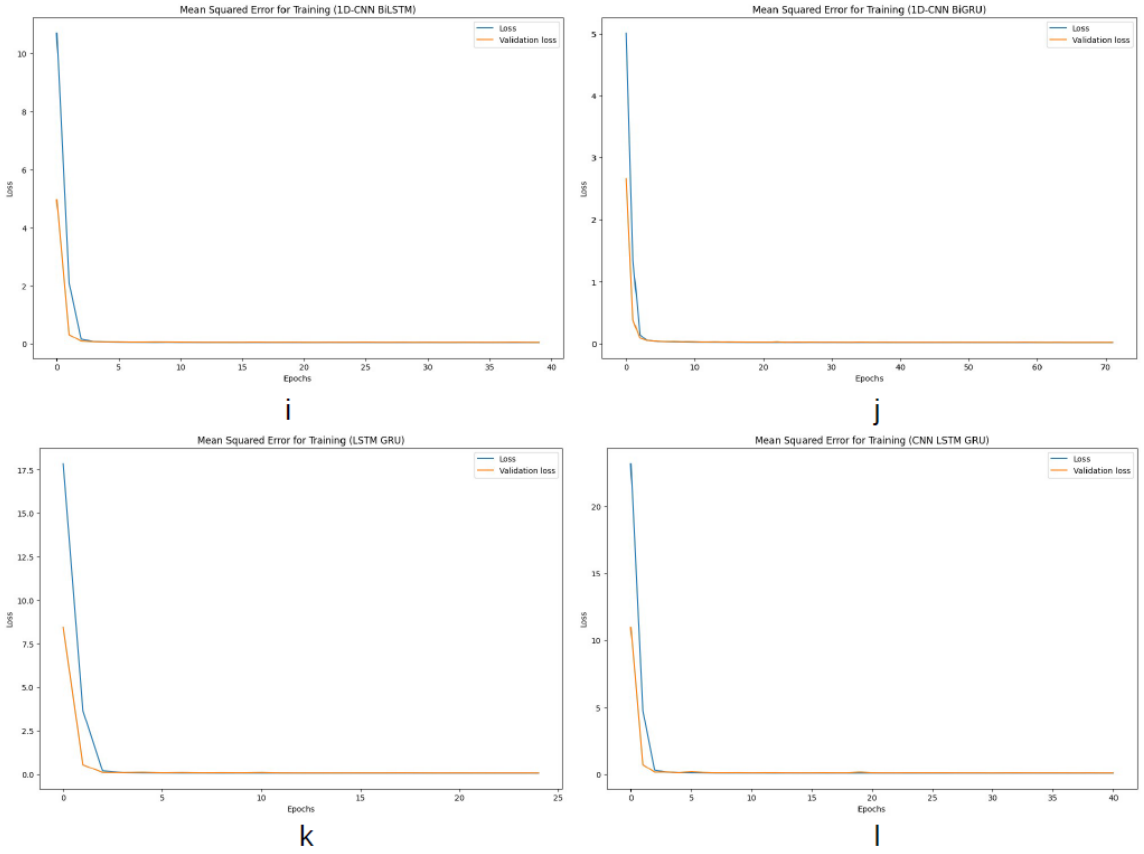
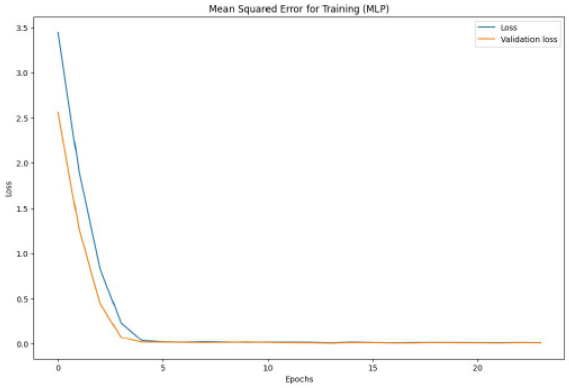
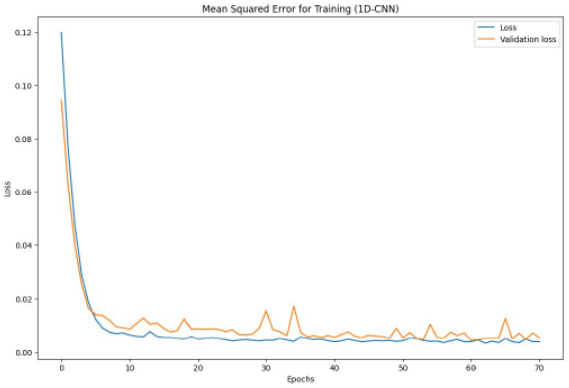


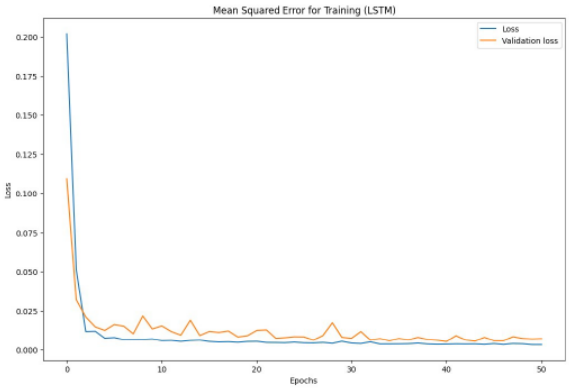
Figure 24: Training loss for models with regularizers (a: multilayer perceptron (MLP), b: convolutional neural network (CNN), c: long short-term memory (LSTM), d: gated recurrent unit (GRU), e: bidirectional LSTM (BiLSTM), f: bidirectional GRU (BiGRU), g: CNN-LSTM, h: CNN-GRU, i: CNN-BiLSTM, j: CNN-BiGRU, k: LSTM-GRU, l: CNN-LSTM-GRU).


Mean Squared Error for Training (MLP)
Loss
Validation loss
Loss
Epochs

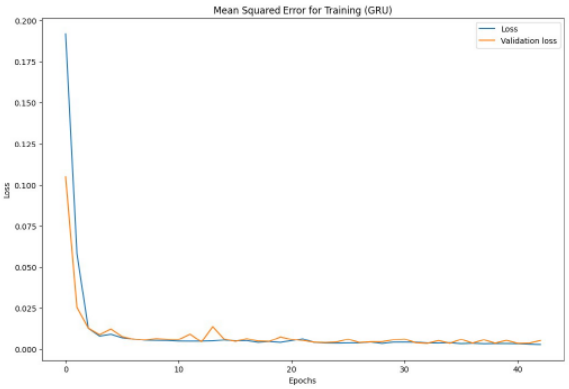
a


Mean Squared Error for Training (1D-CNN)
Loss
Validation loss
Loss
Epochs

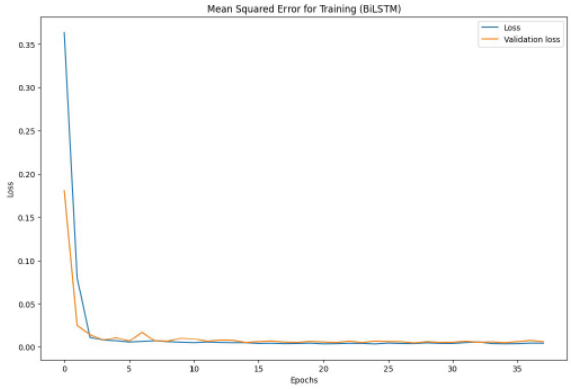
b


Mean Squared Error for Training (LSTM)
Loss
Validation loss
Loss
Epochs

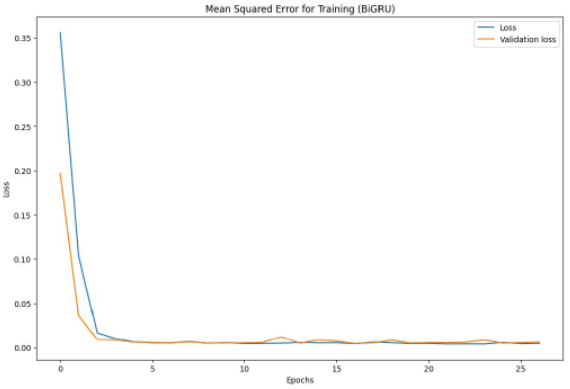
c


Mean Squared Error for Training (GRU)
Loss
Validation loss
Loss
Epochs

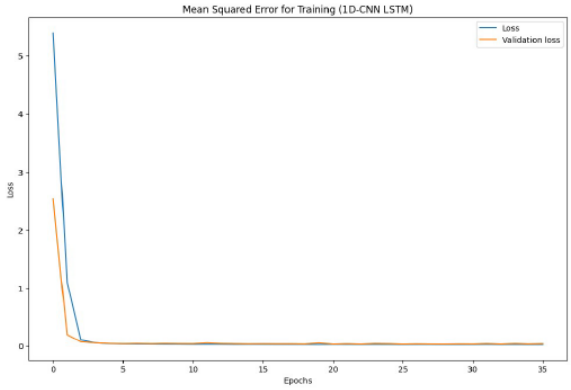
d


Mean Squared Error for Training (BiLSTM)
Loss
Validation loss
Loss
Epochs

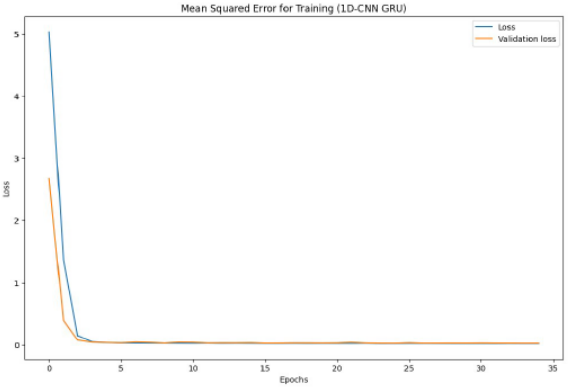
e


Mean Squared Error for Training (BiGRU)
Loss
Validation loss
Loss
Epochs

f


Mean Squared Error for Training (1D-CNN LSTM)
Loss
Validation loss
Loss
Epochs

g


Mean Squared Error for Training (1D-CNN GRU)
Loss
Validation loss
Loss
Epochs

h

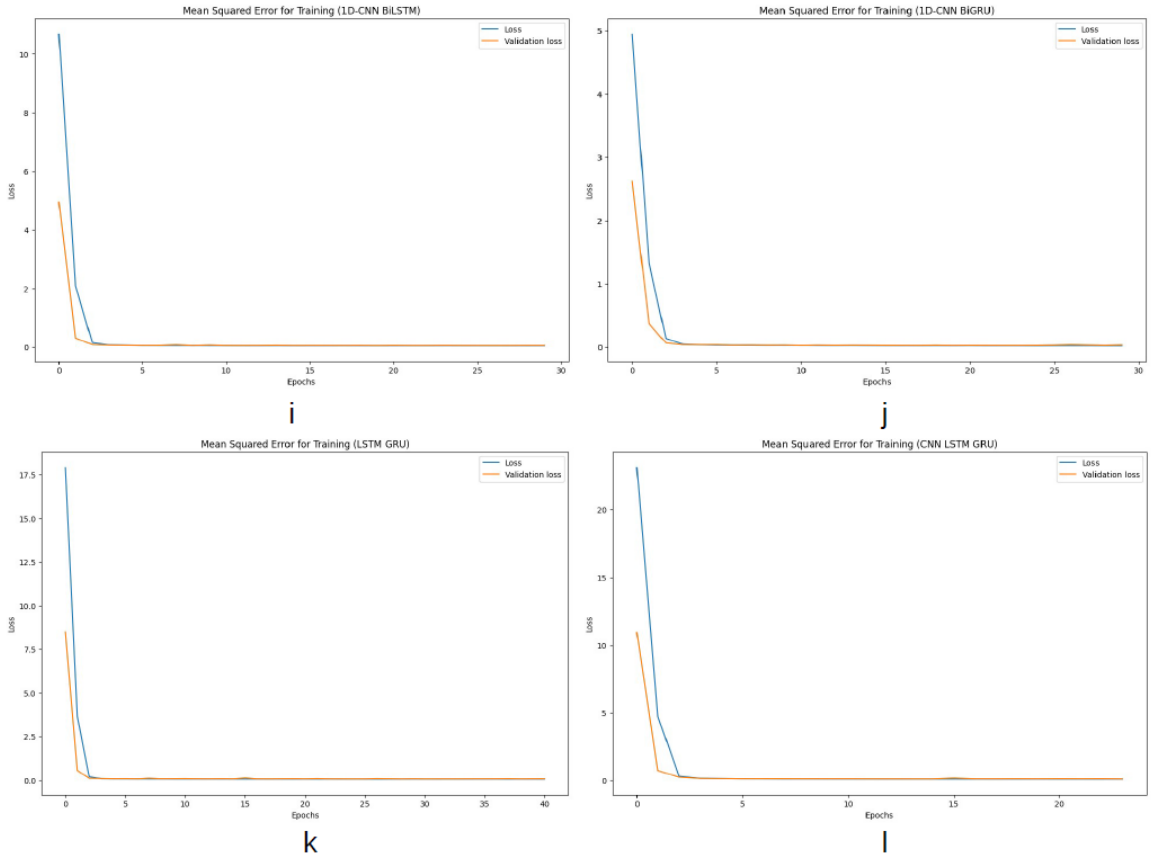
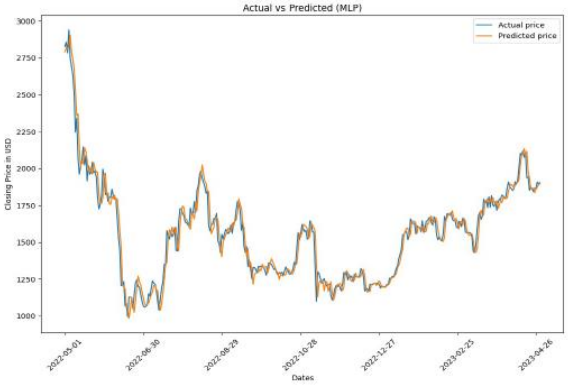
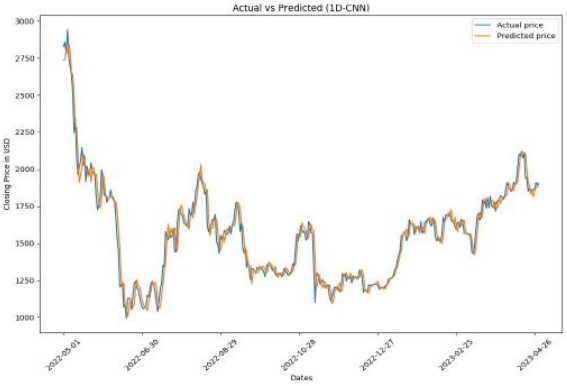


Figure 25: Training loss for models with dropout rates and regularizers (a: multilayer perceptron (MLP), b: convolutional neural network (CNN), c: long short-term memory (LSTM), d: gated recurrent unit (GRU), e: bidirectional LSTM (BiLSTM), f: bidirectional GRU (BiGRU), g: CNN-LSTM, h: CNN-GRU, i: CNN-BiLSTM, j: CNN-BiGRU, k: LSTM-GRU, l: CNN-LSTM-GRU).

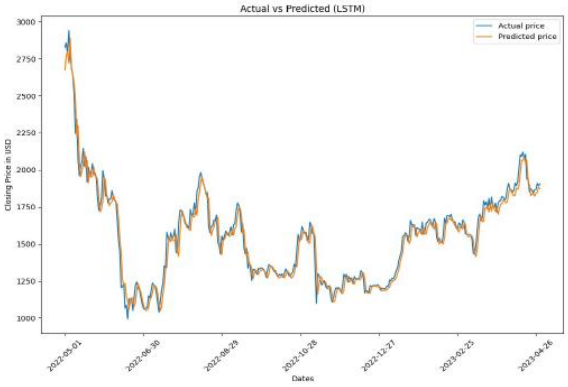
In regards of the initial training loss value, Figures 24 and 25 shows that all models except CNN, LSTM, GRU, BiLSTM, and BiGRU have noticed a very high training loss during the first 5 epochs of training, with values being greater than 1.


Actual vs Predicted (MLP)
Actual price
Predicted price
Closing Price in USD
Dates

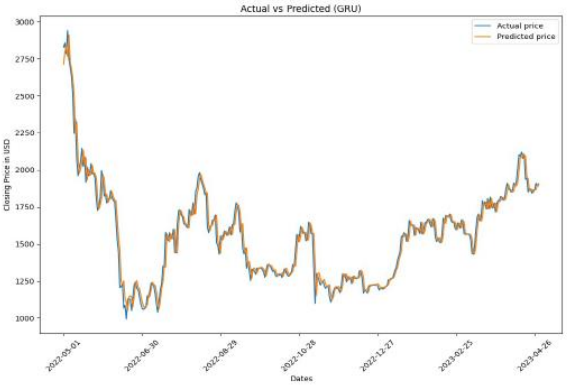
a


Actual vs Predicted (1D-CNN)
Actual price
Predicted price
Closing Price in USD
Dates

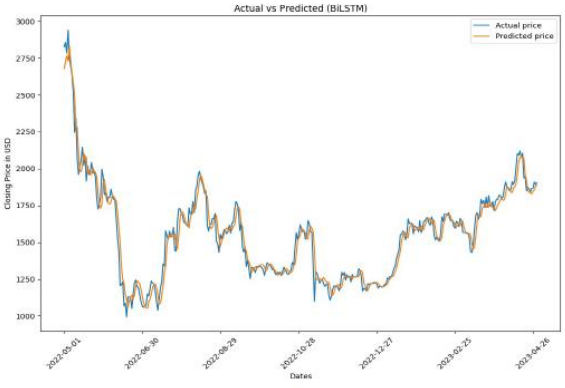
b


Actual vs Predicted (LSTM)
Actual price
Predicted price
Closing Price in USD
Dates

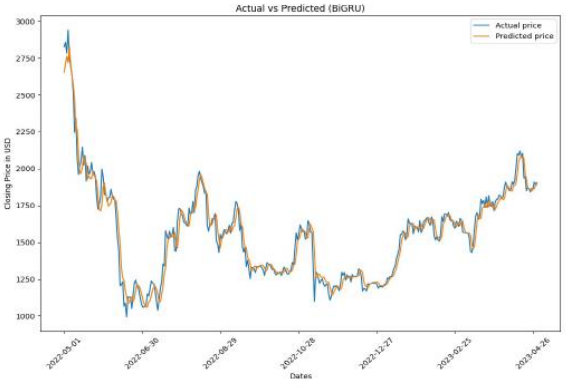
c


Actual vs Predicted (GRU)
Actual price
Predicted price
Closing Price in USD
Dates

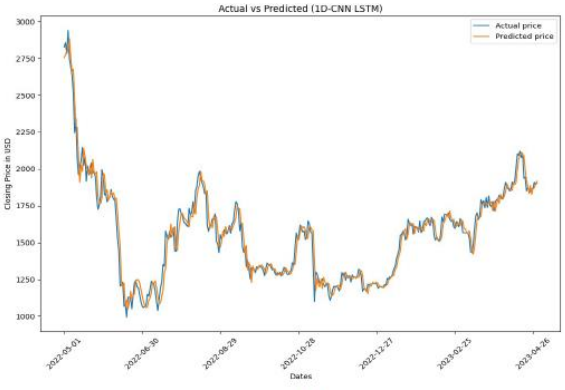
d


Actual vs Predicted (BiLSTM)
Actual price
Predicted price
Closing Price in USD
Dates

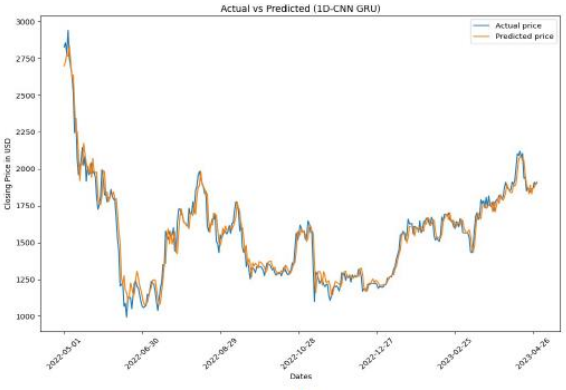
e


Actual vs Predicted (BiGRU)
Actual price
Predicted price
Closing Price in USD
Dates

f


Actual vs Predicted (1D-CNN LSTM)
Actual price
Predicted price
Closing Price in USD
Dates

g


Actual vs Predicted (1D-CNN GRU)
Actual price
Predicted price
Closing Price in USD
Dates

h

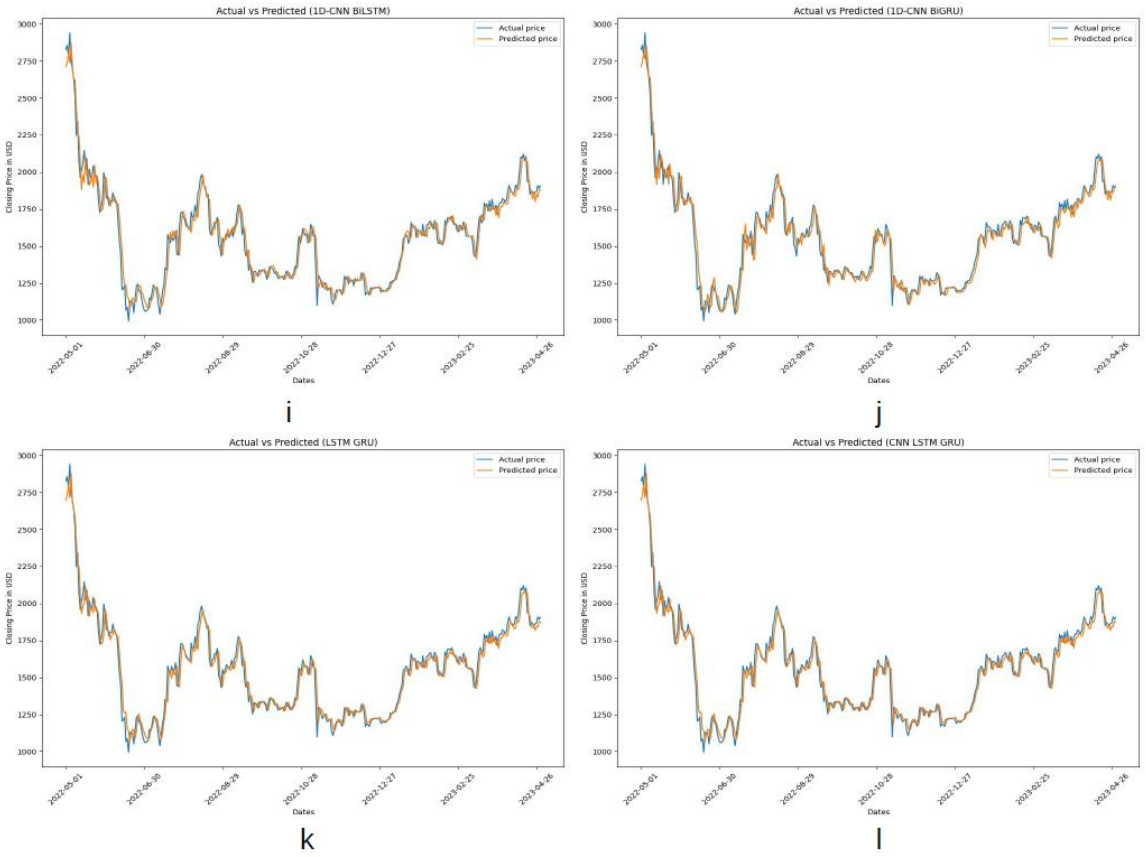
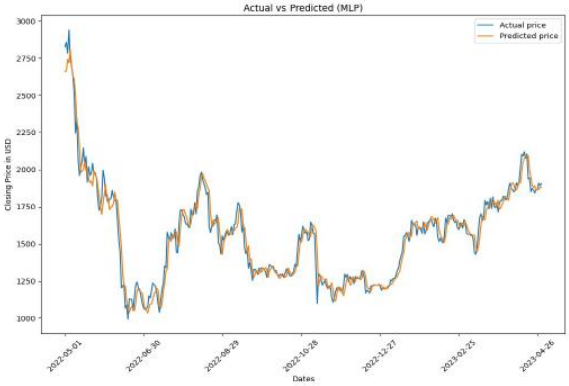
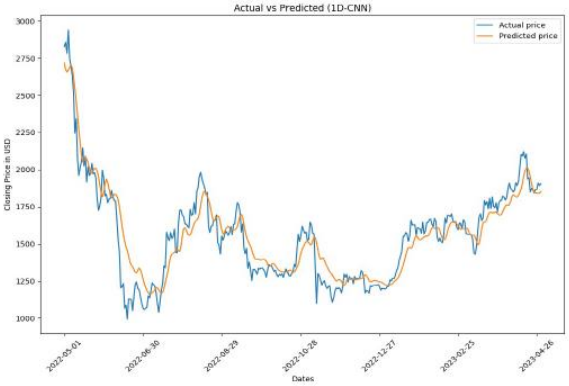


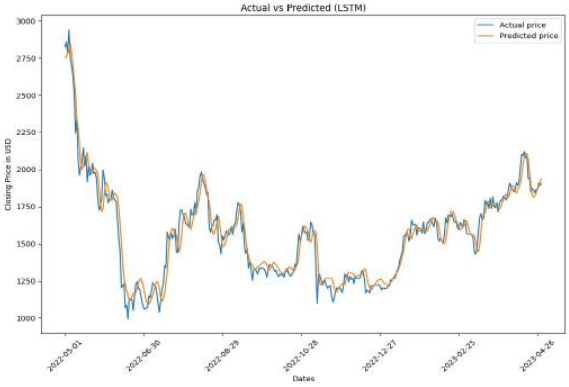
Figure 26: Plots of test set for predicted values of closing price compared with actual values for baseline models (a: multilayer perceptron (MLP), b: convolutional neural network (CNN), c: long short-term memory (LSTM), d: gated recurrent unit (GRU), e: bidirectional LSTM (BiLSTM), f: bidirectional GRU (BiGRU), g: CNN-LSTM, h: CNN-GRU, i: CNN-BiLSTM, j: CNN-BiGRU, k: LSTM-GRU, l: CNN-LSTM-GRU).


Actual vs Predicted (MLP)
Actual price
Predicted price
Closing Price in USD
Dates

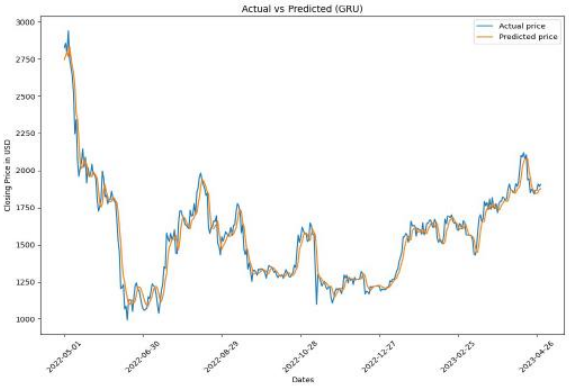
a


Actual vs Predicted (1D-CNN)
Actual price
Predicted price
Closing Price in USD
Dates

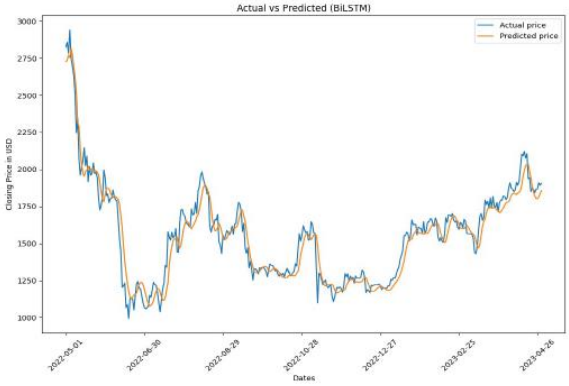
b


Actual vs Predicted (LSTM)
Actual price
Predicted price
Closing Price in USD
Dates

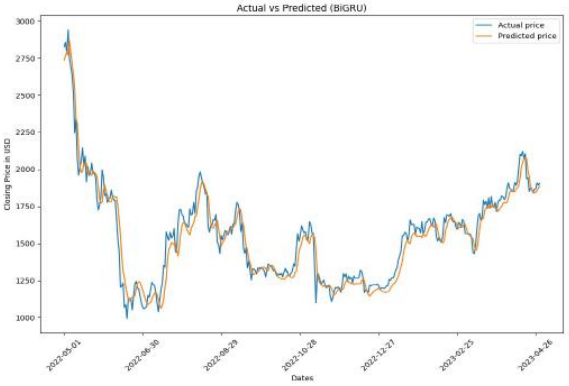
c


Actual vs Predicted (GRU)
Actual price
Predicted price
Closing Price in USD
Dates

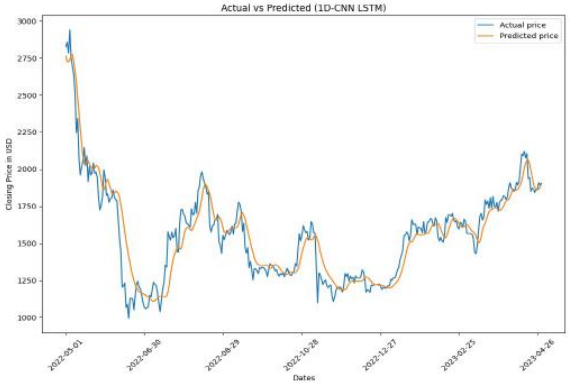
d


Actual vs Predicted (BiLSTM)
Actual price
Predicted price
Closing Price in USD
Dates

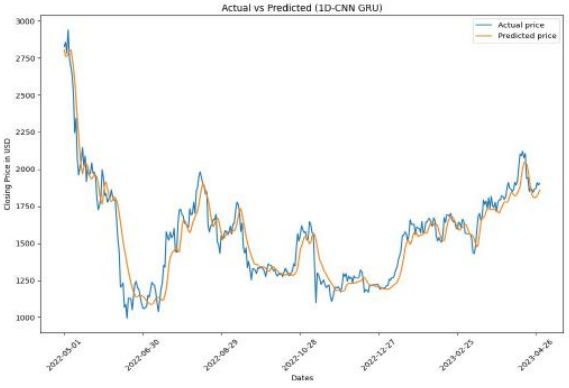
e


Actual vs Predicted (BiGRU)
Actual price
Predicted price
Closing Price in USD
Dates

f


Actual vs Predicted (1D-CNN LSTM)
Actual price
Predicted price
Closing Price in USD
Dates

g


Actual vs Predicted (1D-CNN GRU)
Actual price
Predicted price
Closing Price in USD
Dates

h

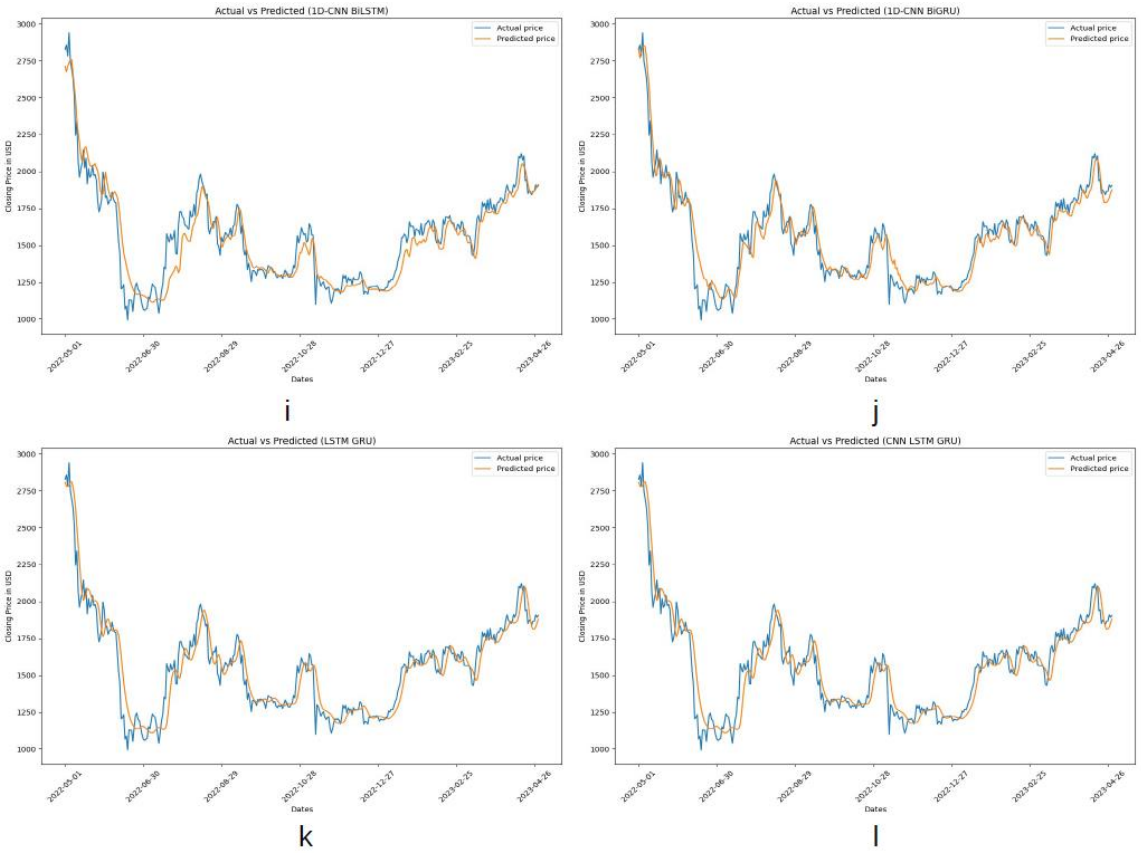
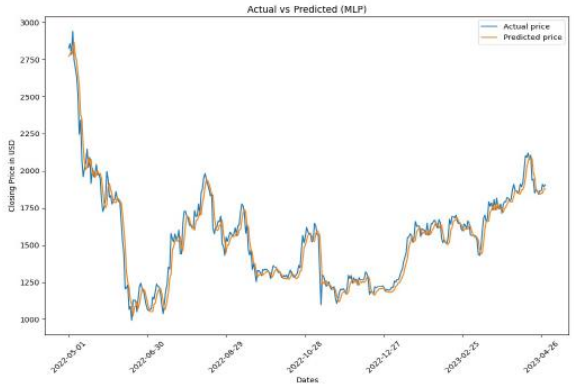
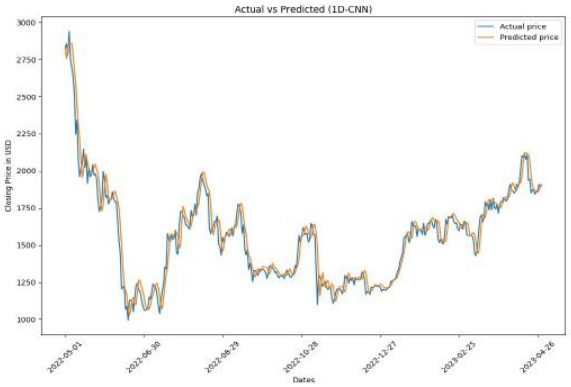


Figure 27: Plots of test set for predicted values of closing price compared with actual values for models with dropout rates (a: multilayer perceptron (MLP), b: convolutional neural network (CNN), c: long short-term memory (LSTM), d: gated recurrent unit (GRU), e: bidirectional LSTM (BiLSTM), f: bidirectional GRU (BiGRU), g: CNN-LSTM, h: CNN-GRU, i: CNN-BiLSTM, j: CNN-BiGRU, k: LSTM-GRU, l: CNN-LSTM-GRU).

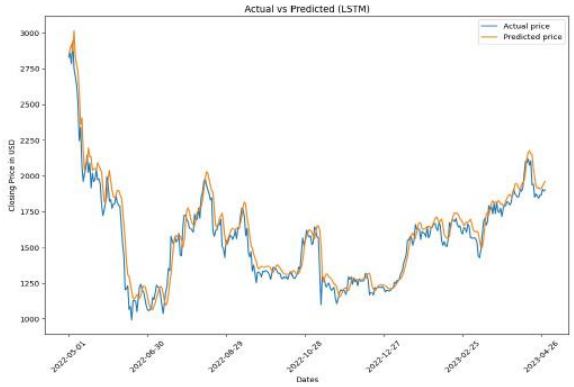
For models using dropout rates, the MLP, LSTM, and GRU models have shown predicted plots that are stable and consistent with the actual trends for the ETH prices. Predictions obtained for all other models have shown somewhat moderate levels of volatility.


Actual vs Predicted (MLP)
Actual price
Predicted price
Closing Price in USD
Dates

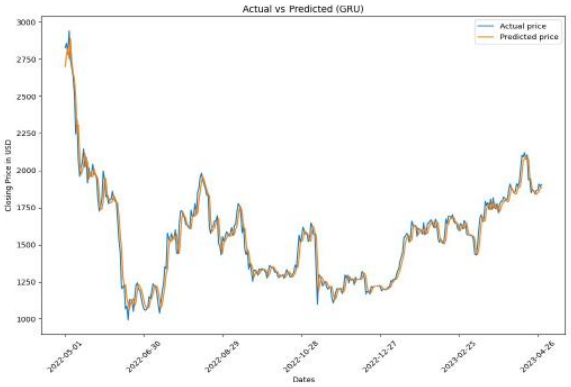
a


Actual vs Predicted (1D-CNN)
Actual price
Predicted price
Closing Price in USD
Dates

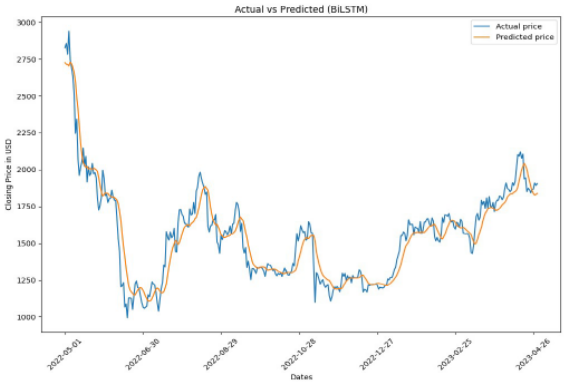
b


Actual vs Predicted (LSTM)
Actual price
Predicted price
Closing Price in USD
Dates

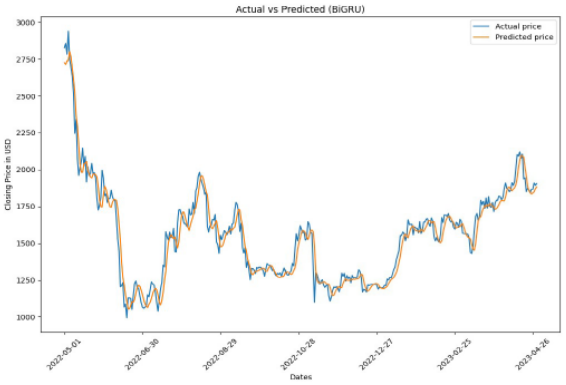
c


Actual vs Predicted (GRU)
Actual price
Predicted price
Closing Price in USD
Dates

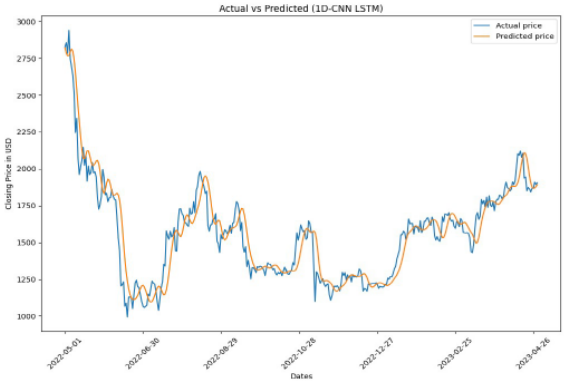
d


Actual vs Predicted (BiLSTM)
Actual price
Predicted price
Closing Price in USD
Dates

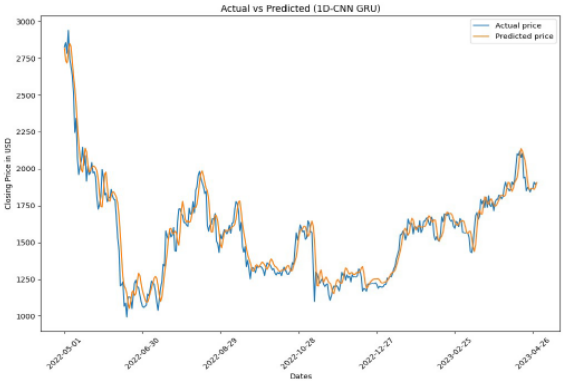
e


Actual vs Predicted (BiGRU)
Actual price
Predicted price
Closing Price in USD
Dates

f


Actual vs Predicted (1D-CNN LSTM)
Actual price
Predicted price
Closing Price in USD
Dates

g


Actual vs Predicted (1D-CNN GRU)
Actual price
Predicted price
Closing Price in USD
Dates

h

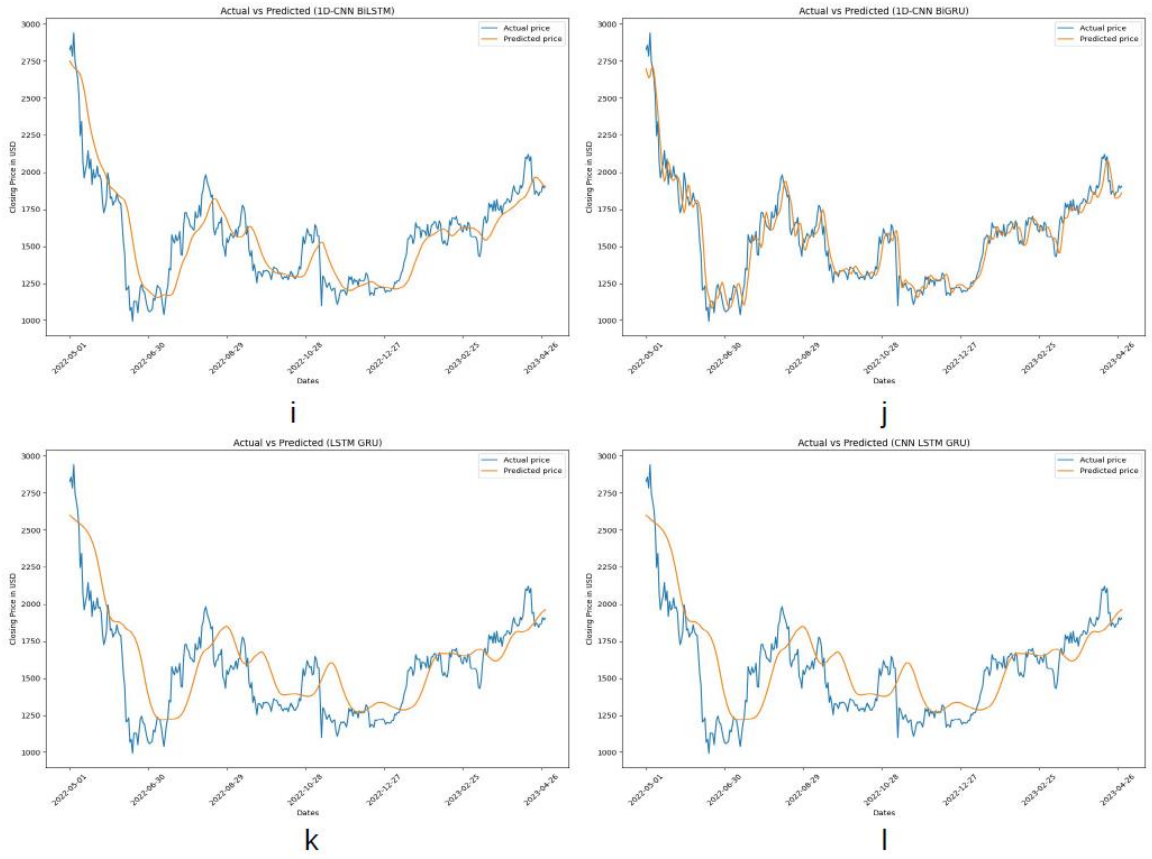
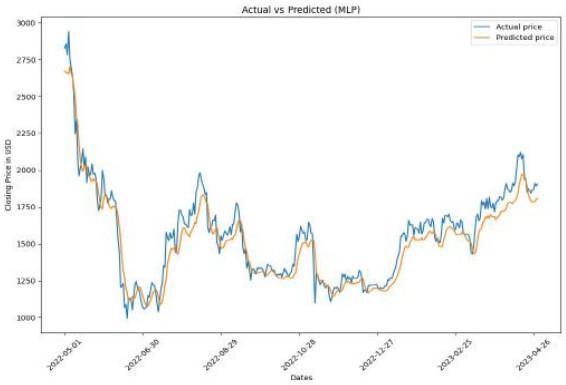
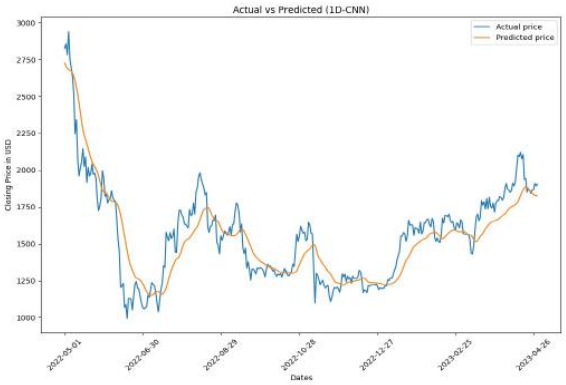


Figure 28: Plots of test set for predicted values of closing price compared with actual values for models with regularizers (a: multilayer perceptron (MLP), b: convolutional neural network (CNN), c: long short-term memory (LSTM), d: gated recurrent unit (GRU), e: bidirectional LSTM (BiLSTM), f: bidirectional GRU (BiGRU), g: CNN-LSTM, h: CNN-GRU, i: CNN-BiLSTM, j: CNN-BiGRU, k: LSTM-GRU, l: CNN-LSTM-GRU).

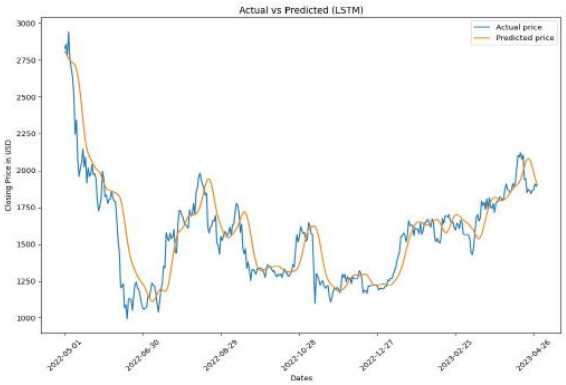
For the following models using regularizers, CNN-BiLSTM, LSTM-GRU, and CNN-LSTM-GRU hybrid models have predicted plots that do not follow well with the actual price trends, as indicated with the misalignment of the two lines.


Actual vs Predicted (MLP)
Actual price
Predicted price
Closing Price in USD
Dates

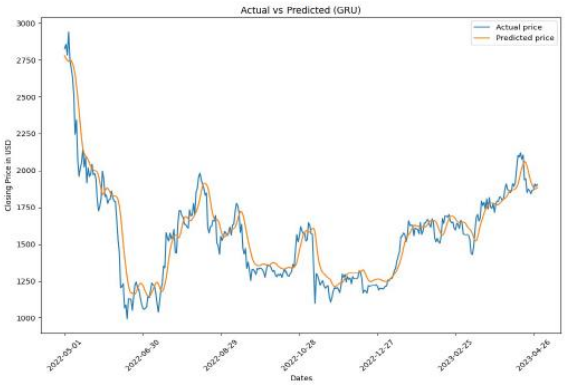
a


Actual vs Predicted (1D-CNN)
Actual price
Predicted price
Closing Price in USD
Dates

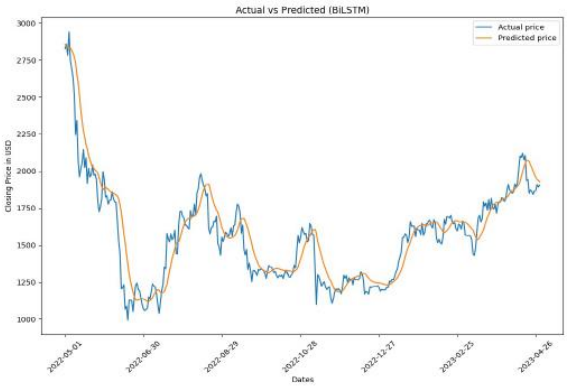
b


Actual vs Predicted (LSTM)
Actual price
Predicted price
Closing Price in USD
Dates

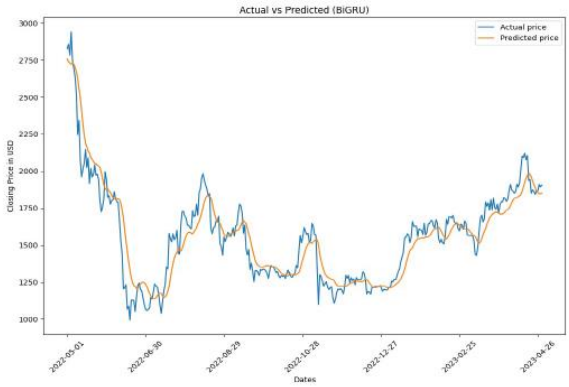
c


Actual vs Predicted (GRU)
Actual price
Predicted price
Closing Price in USD
Dates

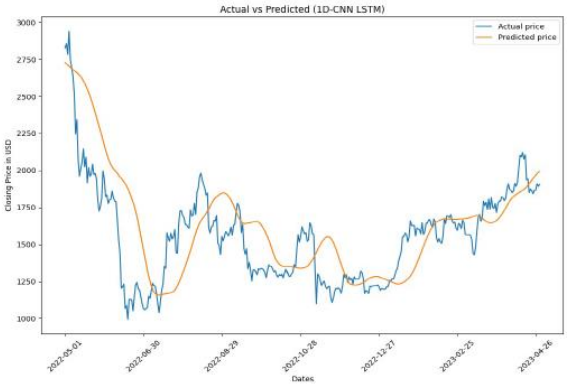
d


Actual vs Predicted (BiLSTM)
Actual price
Predicted price
Closing Price in USD
Dates

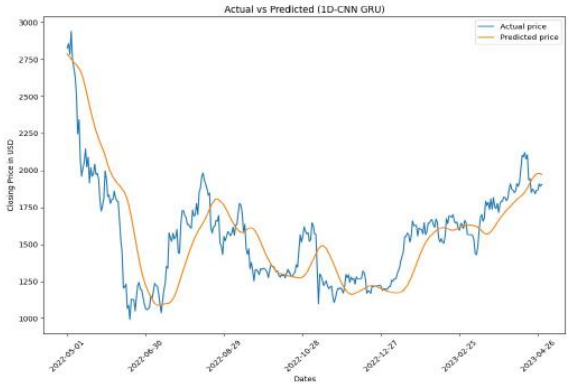
e


Actual vs Predicted (BiGRU)
Actual price
Predicted price
Closing Price in USD
Dates

f


Actual vs Predicted (1D-CNN LSTM)
Actual price
Predicted price
Closing Price in USD
Dates

g


Actual vs Predicted (1D-CNN GRU)
Actual price
Predicted price
Closing Price in USD
Dates

h

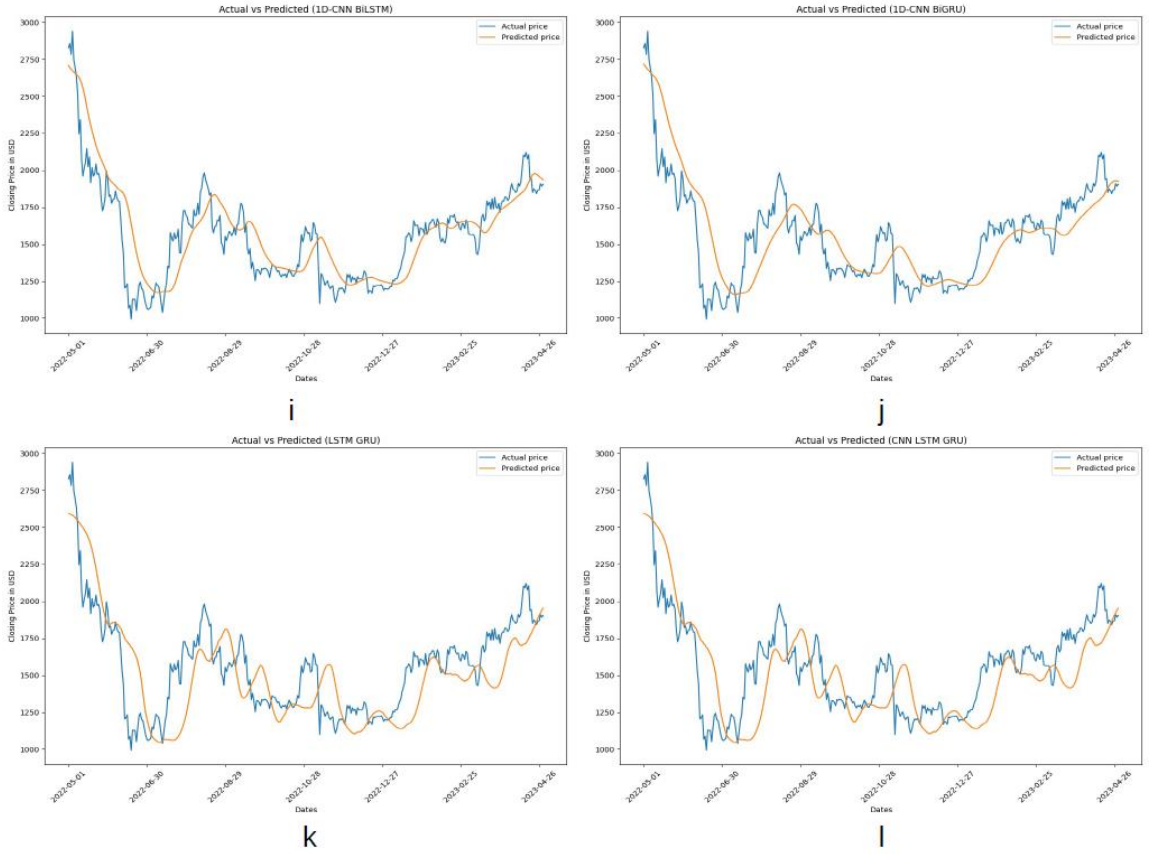


Figure 29: Plots of test set for predicted values of closing price compared with actual values for models with dropout rates and regularizers (a: multilayer perceptron (MLP), b: convolutional neural network (CNN), c: long short-term memory (LSTM), d: gated recurrent unit (GRU), e: bidirectional LSTM (BiLSTM), f: bidirectional GRU (BiGRU), g: CNN-LSTM, h: CNN-GRU, i: CNN-BiLSTM, j: CNN-BiGRU, k: LSTM-GRU, l: CNN-LSTM-GRU).

For the model using dropout rates and regularizers, all models have noticed significant levels of variance and volatility in the predicted trends correlating with the actual plots, except the GRU and BiGRU models.

Model Type	MSE	MAE	RMSE
MLP	0.00077	0.01922	0.02775
CNN	0.00077	0.01981	0.02790
LSTM	0.00072	0.01899	0.02696
GRU	0.00069	0.01808	0.02627
BiLSTM	0.00079	0.01955	0.02825
BiGRU	0.00078	0.01933	0.02805
CNN LSTM	0.00076	0.01924	0.02772
CNN GRU	0.00077	0.01949	0.02774
CNN BiLSTM	0.00074	0.01912	0.02722
CNN BiGRU	0.00077	0.01941	0.02786
LSTM GRU	0.00069	0.01837	0.02634
CNN LSTM GRU	0.00081	0.02055	0.02855

Table 17: The comparison of all baseline models' MSE (mean squared error), MAE (mean absolute error), and RMSE (root mean squared error) based on their architecture types.

Model Type	MSE	MAE	RMSE
MLP	0.00081	0.02005	0.02853
CNN	0.00236	0.03621	0.04866
LSTM	0.00100	0.02264	0.03167
GRU	0.00093	0.02108	0.03063
BiLSTM	0.00129	0.02595	0.03597
BiGRU	0.00114	0.02443	0.03384
CNN LSTM	0.00215	0.03267	0.04639
CNN GRU	0.00184	0.03061	0.04300
CNN BiLSTM	0.00186	0.03077	0.04316
CNN BiGRU	0.00141	0.02663	0.03763
LSTM GRU	0.00171	0.02911	0.04137
CNN LSTM GRU	0.00187	0.02969	0.04327

Table 18: The comparison of all dropout rate models' MSE (mean squared error), MAE (mean absolute error), and RMSE (root mean squared error) based on their architecture types.

Model Type	MSE	MAE	RMSE
MLP	0.00088	0.02101	0.02981
CNN	0.00102	0.02243	0.03199
LSTM	0.00135	0.02664	0.03679
GRU	0.00074	0.01872	0.02726
BiLSTM	0.00180	0.03038	0.04245
BiGRU	0.00099	0.02234	0.03151
CNN LSTM	0.00221	0.03309	0.04703
CNN GRU	0.00118	0.02478	0.03442
CNN BiLSTM	0.00431	0.04836	0.06570
CNN BiGRU	0.00143	0.02706	0.03782
LSTM GRU	0.00840	0.07099	0.09167
CNN LSTM GRU	0.01289	0.09028	0.11356

Table 19: The comparison of all regularizer models' MSE (mean squared error), MAE (mean absolute error), and RMSE (root mean squared error) on their architecture types.

Model Type	MSE	MAE	RMSE
MLP	0.00153	0.03049	0.03922
CNN	0.00349	0.04515	0.05912
LSTM	0.00399	0.04363	0.06321
GRU	0.00201	0.03179	0.04493
BiLSTM	0.00270	0.03632	0.05204
BiGRU	0.00226	0.03453	0.04756
CNN LSTM	0.01098	0.08009	0.10480
CNN GRU	0.00763	0.06662	0.08737
CNN BiLSTM	0.00516	0.05334	0.07185
CNN BiGRU	0.00692	0.06420	0.08319
LSTM GRU	0.00905	0.07588	0.09518
CNN LSTM GRU	0.01561	0.09848	0.12496

Table 20: The comparison of all dropout rate and regularizer models' MSE (mean squared error), MAE (mean absolute error), and RMSE (root mean squared error) based on their architecture types.

In the baseline model, the LSTM-GRU hybrid model ties with the GRU model with the lowest MSE loss value, yet in most cases, GRU outperformed most other neural network models, with LSTM-GRU and LSTM not too far behind. The same case can be said for GRU models using regularizers. MLP models using dropout rates only and both dropout rates and regularizers did well in minimizing the loss values, even though other models could do better.

In terms of LSTM models, based on the numerical results in Tables 16 to 20, baseline LSTM models tend to do better than most other models only outperformed by GRU, LSTM models using either dropout rates or regularizers did slightly worse than all others though not much as hybrid models, and LSTM models using both dropout rates and regularizers fared poorly compare to others.

5 Conclusion

After I finished the project, what I learned so far is that I am able to apply deep learning methodologies to solve real world problems, and here, I would like to explore how the future behaviors of Ethereum's price trend will adjust over time by building a forecasting model with LSTM. I also took an interest in seeing how well LSTM can perform in terms of loss metrics like MSE, MAE, and RMSE by comparing it with other neural network models like convolutional models and hybrid neural network models. Predicting future prices with neural networks is an inspiring topic that I want to explore, and by using neural networks to obtain a bigger picture on comparing the actual trends with the predicted results, I can better understand how the prices' behavior will change over time in long term.

If there are difficulties that I noticed while doing the project, then it is resource limitations that I have to deal with. This is likely because I am using GPU mode in Colab so that it would speed up the current experiment for all models. However, it is unlikely that I am able to run the experiments as long as I want due to limits on maximum use time in GPU mode. Furthermore, it is possible that even if using standard runtime to conduct all of the experiments, the process may halt unexpectedly, even if the entire task takes about 30 to 45 minutes to complete. This means that I decided to simplify and shorten all the models that I created, as well as lowering the number of epochs for the training process. Another issue that I faced is where I prepare the dataset by dividing them into training and test sets, and assigning the features and labels among them. Originally, my project will have all 4 features as part of the training process (the opening, highest, lowest, and closing prices from the ETH price list), but because I encountered issues where the expected shapes of the resulting values must match the expected ones, I decided to consider only the closing price for the entirety of the experiment.

For other things that I have found interesting or unusual for the research project, upon closer inspection of the predicted plots for the ETH future prices, I noticed that for some models, the predictions may fail to correlate with the actual price trends, even if the loss metrics are kept as low as possible. This is likely because the dataset I obtained may have trends that suddenly spikes upward or plunge downward in certain time areas. Based on what Sethia and Raut [32] noted in the conference paper, those instances may indicate that there are events that occur on a global scale that may dramatically affect the future prices of any stock markets, such as an economic recession, a military conflict, or a humanitarian disaster going on. Also, what I noticed in the results section is that some neural network models may outperform others, as

the results generated from the experiment may be different for each run. For example, on rare occasions, certain neural network models can outperform each other like a hybrid LSTM-GRU baseline model doing as well as a baseline GRU model in terms of MSE loss. This is likely the case of stochastic property applied to all models, meaning that randomization can alter the results for each new experiment, which is why no two results will be the same. From what I can conclude, LSTM models may not necessarily yield the optimal results in predicting future prices of Ethereum as certain neural network models like GRU could outperform it in terms of performance metrics, but nonetheless, LSTM does remain useful in time-series forecasting as it can reduce long-term dependence problems (where vanishing gradients can cause the results to not update at all during training) and lower the likelihood of overfitting. Therefore, LSTM remains relevant in being able to forecast future prices of Ethereum, even if it remains overshadowed by more successful competitors.

The improvements that I can consider for future work of my research project are but not limited to the following: implement additional models like support vector machines and generative adversarial networks, increase the number of epochs for training and value of patience (additional epochs to train if no further improvement is made for any metric) for early stopping, use different optimizers for the neural network models like Adamax and RMSProp, try out more complex methodologies like transfer learning (which involves building a new neural network model on top of an existing one), try different lengths of training and test datasets, and adjust the values of dropout rates and regularizers.

Bibliography

[1] A. Gervais, G. O. Karame, V. Capkun and S. Capkun, "Is Bitcoin a Decentralized Currency?," in *IEEE Security & Privacy*, vol. 12, no. 3, pp. 54-60, May-June 2014, doi: 10.1109/MSP.2014.49.

[2] X.-Y. Zhang, F. Yin, Y.-M. Zhang, C.-L. Liu and Y. Bengio, "Drawing and Recognizing Chinese Characters with Recurrent Neural Network," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 4, pp. 849-862, 1 April 2018, doi: 10.1109/TPAMI.2017.2695539.

[3] Z. Meng, P. Liu, J. Cai, S. Han and Y. Tong, "Identity-Aware Convolutional Neural Network for Facial Expression Recognition," *2017 12th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2017)*, 2017, pp. 558-565, doi: 10.1109/FG.2017.140.

[4] G. S. V. S. Sivaram and H. Hermansky, "Sparse Multilayer Perceptron for Phoneme Recognition," in *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 20, no. 1, pp. 23-29, Jan. 2012, doi: 10.1109/TASL.2011.2129510.

[5] J. Deng, Z. Zhang, E. Marchi and B. Schuller, "Sparse Autoencoder-Based Feature Transfer Learning for Speech Emotion Recognition," *2013 Humaine Association Conference on Affective Computing and Intelligent Interaction*, 2013, pp. 511-516, doi: 10.1109/ACII.2013.90.

[6] S. Selvin, R. Vinayakumar, E. A. Gopalakrishnan, V. K. Menon and K. P. Soman, "Stock price prediction using LSTM, RNN and CNN-sliding window model," *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, Udupi, India, 2017, pp. 1643-1647, doi: 10.1109/ICACCI.2017.8126078.

[7] M. K. Farahani and S. Mehralian, "Comparison between Artificial Neural Network and neuro-fuzzy for gold price prediction," *2013 13th Iranian Conference on Fuzzy Systems (IFSC)*, Qazvin, Iran, 2013, pp. 1-5, doi: 10.1109/IFSC.2013.6675635.

[8] D. Kumar and S. K. Rath, “Predicting the Trends of Price for Ethereum Using Deep Learning Techniques,” *Advances in Intelligent Systems and Computing*. Springer Singapore, pp. 103–114, 2020. doi: 10.1007/978-981-15-0199-9_9.

[9] D. Kwon, J. Kim, J. Heo, C. Kim and Y. Han, “Time Series Classification of Cryptocurrency Price Trend Based on a Recurrent LSTM Neural Network,” *Journal of Information Processing Systems*, vol. 15, no. 3, pp. 694-706, 2019. doi: 10.3745/JIPS.03.0120.

[10] P. Jay, V. Kalariya, P. Parmar, S. Tanwar, N. Kumar and M. Alazab, "Stochastic Neural Networks for Cryptocurrency Price Prediction," in *IEEE Access*, vol. 8, pp. 82804-82818, 2020, doi: 10.1109/ACCESS.2020.2990659.

[11] K. Zhang, G. Zhong, J. Dong, S. Wang, and Y. Wang, “Stock Market Prediction Based on Generative Adversarial Network,” *Procedia Computer Science*, vol. 147. Elsevier BV, pp. 400–406, 2019. doi: 10.1016/j.procs.2019.01.256.

[12] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta and A. A. Bharath, "Generative Adversarial Networks: An Overview," in *IEEE Signal Processing Magazine*, vol. 35, no. 1, pp. 53-65, Jan. 2018, doi: 10.1109/MSP.2017.2765202.

[13] D. R. Pant, P. Neupane, A. Poudel, A. K. Pokhrel and B. K. Lama, "Recurrent Neural Network Based Bitcoin Price Prediction by Twitter Sentiment Analysis," *2018 IEEE 3rd International Conference on Computing, Communication and Security (ICCCS)*, Kathmandu, Nepal, 2018, pp. 128-132, doi: 10.1109/CCCS.2018.8586824.

[14] S. Mohapatra, N. Ahmed and P. Alencar, "KryptoOracle: A Real-Time Cryptocurrency Price Prediction Platform Using Twitter Sentiments," *2019 IEEE International Conference on Big Data (Big Data)*, Los Angeles, CA, USA, 2019, pp. 5544-5551, doi: 10.1109/BigData47090.2019.9006554.

[15] F. Valencia, A. Gómez-Espinosa, and B. Valdés-Aguirre, “Price Movement Prediction of Cryptocurrencies Using Sentiment Analysis and Machine Learning,” *Entropy*, vol. 21, no. 6, p. 589, Jun. 2019, doi: 10.3390/e21060589.

[16] K. Team, “Keras: Deep Learning for humans,” Keras, https://keras.io/ (accessed May 11, 2023).

[17] TensorFlow, https://www.tensorflow.org/ (accessed May 11, 2023).

[18] “Pandas - Python Data Analysis Library,” pandas, https://pandas.pydata.org/ (accessed May 11, 2023).

[19] “NumPy: The fundamental package for scientific computing with Python,” NumPy, https://numpy.org/ (accessed May 11, 2023).

[20] “Matplotlib: Visualization with Python,” Matplotlib, https://matplotlib.org/ (accessed May 11, 2023).

[21] “Scikit-Learn: Machine Learning in Python,” scikit-learn, https://scikit-learn.org/stable/ (accessed May 11, 2023).

[22] “Ethereum USD (ETH-USD) price, value, news & history,” Yahoo! Finance, https://finance.yahoo.com/quote/ETH-USD/ (accessed May 11, 2023).

[23] A. Eesa and W. Arabo, “A Normalization Methods for Backpropagation: A Comparative Study”, *Science Journal of University of Zakho*, vol. 5, no. 4, pp. 319-323, Dec. 2017, doi: https://doi.org/10.25271/2017.5.4.381.

[24] R. Gencay and M. Qi, "Pricing and hedging derivative securities with neural networks: Bayesian regularization, early stopping, and bagging," in *IEEE Transactions on Neural Networks*, vol. 12, no. 4, pp. 726-734, July 2001, doi: 10.1109/72.935086.

[25] V. Pham, T. Bluche, C. Kermorvant and J. Louradour, "Dropout Improves Recurrent Neural Networks for Handwriting Recognition," *2014 14th International Conference on Frontiers in Handwriting Recognition*, Hersonissos, Greece, 2014, pp. 285-290, doi: 10.1109/ICFHR.2014.55.

[26] L. Qian, L. Hu, L. Zhao, T. Wang and R. Jiang, "Sequence-Dropout Block for Reducing Overfitting Problem in Image Classification," in *IEEE Access*, vol. 8, pp. 62830-62840, 2020, doi: 10.1109/ACCESS.2020.2983774.

[27] G. Pei, Y. Wang, Y. Cheng and L. Zhang, "Joint Label-Density-Margin Space and Extreme Elastic Net for Label-Specific Features," in *IEEE Access*, vol. 7, pp. 112304-112317, 2019, doi: 10.1109/ACCESS.2019.2934742.

[28] P. Gao, R. Zhang, and X. Yang, “The Application of Stock Index Price Prediction with Neural Network,” *Mathematical and Computational Applications*, vol. 25, no. 3, p. 53, Aug. 2020, doi: 10.3390/mca25030053.

[29] P. J. Werbos, "Backpropagation through time: what it does and how to do it," in *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550-1560, Oct. 1990, doi: 10.1109/5.58337.

[30] Z. Li, F. Liu, W. Yang, S. Peng and J. Zhou, "A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects," in *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 12, pp. 6999-7019, Dec. 2022, doi: 10.1109/TNNLS.2021.3084827.

[31] M. Liu, J. Shi, Z. Li, C. Li, J. Zhu and S. Liu, "Towards Better Analysis of Deep Convolutional Neural Networks," in *IEEE Transactions on Visualization and Computer Graphics*, vol. 23, no. 1, pp. 91-100, Jan. 2017, doi: 10.1109/TVCG.2016.2598831.

[32] A. Sethia and P. Raut, "Application of LSTM, GRU and ICA for Stock Price Prediction," *Information and Communication Technology for Intelligent Systems*. Springer Singapore, pp. 479–487, Dec. 15, 2018. doi: 10.1007/978-981-13-1747-7_46.

[33] R. Fu, Z. Zhang and L. Li, "Using LSTM and GRU neural network methods for traffic flow prediction," *2016 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC)*, Wuhan, China, 2016, pp. 324-328, doi: 10.1109/YAC.2016.7804912

[34] A. Graves, N. Jaitly and A. -r. Mohamed, "Hybrid speech recognition with Deep Bidirectional LSTM," *2013 IEEE Workshop on Automatic Speech Recognition and Understanding*, Olomouc, Czech Republic, 2013, pp. 273-278, doi: 10.1109/ASRU.2013.6707742.

[35] H. M. Lynn, S. B. Pan and P. Kim, "A Deep Bidirectional GRU Network Model for Biometric Electrocardiogram Classification Based on Recurrent Neural Networks," in *IEEE Access*, vol. 7, pp. 145395-145405, 2019, doi: 10.1109/ACCESS.2019.2939947.

[36] J. Zhu, H. Chen and W. Ye, "A Hybrid CNN–LSTM Network for the Classification of Human Activities Based on Micro-Doppler Radar," in *IEEE Access*, vol. 8, pp. 24713-24720, 2020, doi: 10.1109/ACCESS.2020.2971064.

[37] M. Sajjad *et al*., "A Novel CNN-GRU-Based Hybrid Approach for Short-Term Residential Load Forecasting," in *IEEE Access*, vol. 8, pp. 143759-143768, 2020, doi: 10.1109/ACCESS.2020.3009537.

[38] M. S. Islam and E. Hossain, “Foreign exchange currency rate prediction using a GRU-LSTM hybrid network,” *Soft Computing Letters*, vol. 3. Elsevier BV, p. 100009, Dec. 2021. doi: 10.1016/j.socl.2020.100009.