Lebanese American University

Computer Science and Mathematics Department

CSC461 Introduction to Machine Learning

Project

Gold, Silver and Canadian Dollar Prices Predictor

Instructor: Dr. Danielle Azar

Rita Merhi
Thomas Naouchi

December 03, 2025

Contents

List of Figures

List of Tables

1 Introduction

For the final project of the machine learning course, we have to apply our accumulated knowledge to build a model that predicts the future price of gold, silver and Canadian dollars for multiple horizons over a user specified range. This will acquaint us with the practical implementation of time series. We will also have to figure out the best way to work with an impoverished dataset.

2 Libraries and Packages Used

- yfinance
- pandas
- sklearn
- matplotlib
- seaborn
- numpy
- tensorflow
- keras
- hashlib
- datetime

3 Data Analysis and Preparation

For this project, we are required to get our data strictly from yahoo finance through the yfinance API [1]. The used tickers to download our data are: "GC=F", "SI=F", "CADUSD=X" for gold, silver and Canadian dollars respectively. Each asset has the following columns:

- **Adj Close:** the adjusted closing price. It is adjusted for splits, dividends. Commodities like all three of our assets here rarely have dividends, so Adj Close is the same as Close.
- **Close:** the official closing price of the trading transaction for each interval.
- **High:** the highest trading price of the asset in an interval.
- **Low:** the lowest trading price of the asset in an interval.
- **Open:** the price proposed at the beginning of an interval.
- **Volume:** the number of transactions made during each interval

Here, our target features are the closing prices of Gold, Silver and Canadian Dollars at each timestep. The date and time column is treated as the index of the dataframe.

3.1 Data Constraints and Challenges

The yfinance API restricts us to downloading only 60 days worth of data for a 30min interval, and 730 days for a 1h interval. Regarding the 1 day interval, there are no such restrictions but the data available goes back to 30 August 2000 for gold and silver, and 18 September 2003 for Canadian dollars. Hence, we ended up with 1974 rows for 30min, 12536 rows for 1h and 5792 rows for 1d.

3.2 Dealing with missing values

The data downloaded from yahoo finance has missing values for the days when the market closes so no trading occurs. We assumed that the price will then stay the same until the next update. Hence, we filled the missing attribute values using backward fill for missing values at the beginning of the dataset, and forward fill for the rest. That way, if for example a price is missing at 10:00, we use the last known value at 09:30.

3.3 Dropping Columns

First things first, we generated heatmaps for each asset to better understand their individual features and how these correlate with the target closing price.

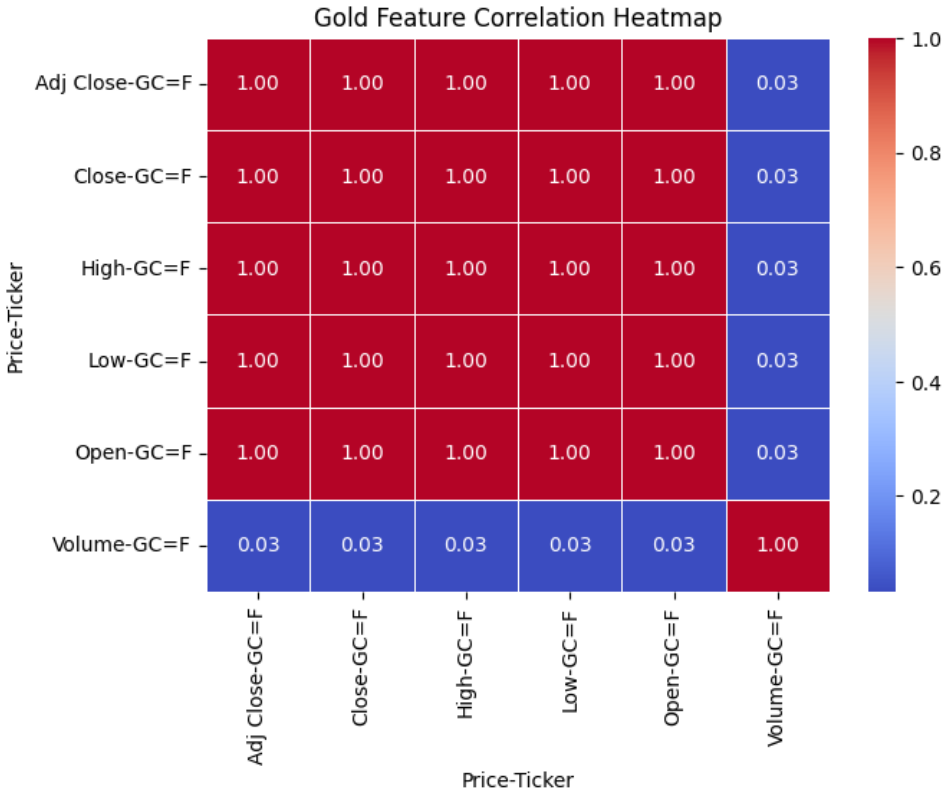

Figure 1: Gold Feature Correlation Heatmap

Figure 2: Silver Feature Correlation Heatmap

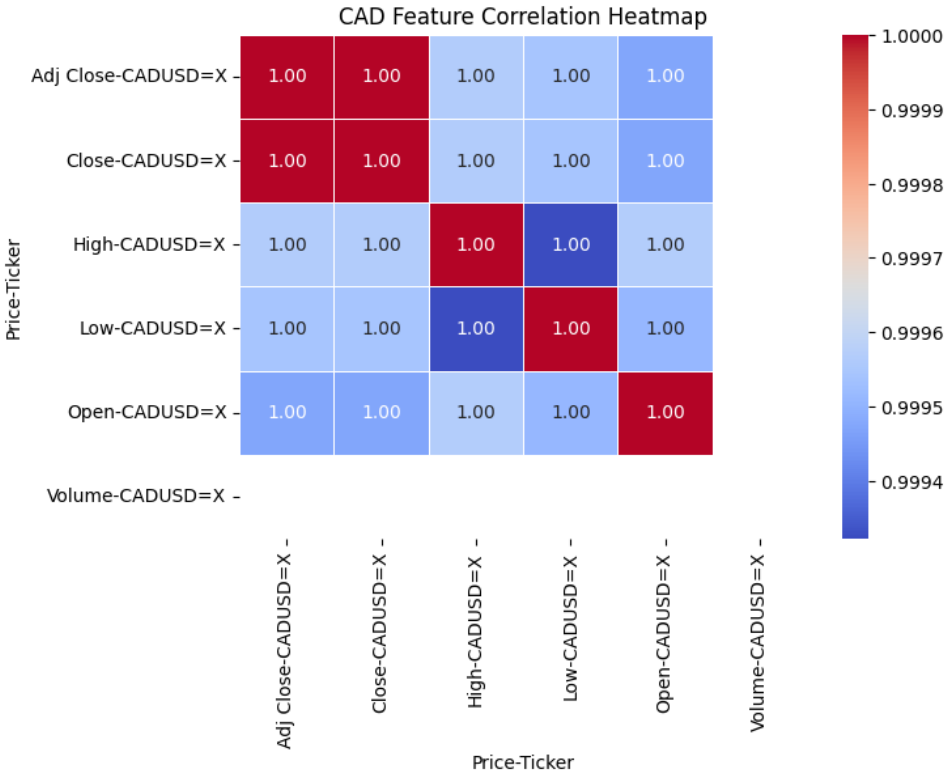

Figure 3: CAD Feature Correlation Heatmap

From Figures 1, 2 and 3 we notice that the features are perfectly correlated with each

other except for Volume. This tells us that these features are redundant, as they carry almost exactly the same information. Including them all adds massive multicollinearity which hurts machine learning models. If they are kept, the model will waste capacity learning the same thing 5 times which reduces efficiency and often lowers accuracy. Also, in linear models, coefficients will become unstable and explode. Hence, we decided to keep the Close column and drop the rest. Volume is also dropped since it has an extremely low correlation of 0.03 which means it does not help predict future asset price.

3.4 Feature Engineering

3.4.1 Lag Feature

For the models that don't usually take timesteps into consideration, namely linear regression and XGBoost in our project, we implemented a lag feature. It consists of taking the value at a previous timestep, and adding it to the instance at a subsequent timestep, thus giving it knowledge of the past. The actual price is also shifted one step into the future, so the model's target is the next timestep's price.

3.4.2 Derived Features

We tried deriving features from our data, namely percentage returns (percentage change between consecutive prices) and moving averages (calculates the 5-period moving average for each asset). While these two features helped XGBoost, they actually hurt the performance of RNN models so they were not used while training them. We think this was the case because XGBoost is a tree-based model, while RNNs are sequence models that learn temporal patterns directly from the raw time series. Because of this difference, the same engineered features that simplify the learning problem for XGBoost can remove or distort the temporal information that RNNs rely on.

3.5 Scaling the data

We used the MinMaxScaler from scikitlearn to scale our data before training the models on it, transforming each variable into the range [0,1]. Normalization is crucial when working with RNN, GRU and LSTM since they compute internal gates using sigmoid and tanh activations: both operate best when inputs are in a small numeric range.

3.6 Splitting the data

One crucial thing that we kept in mind while splitting into a training, validation and testing set is to not ruin the time series. This means we performed the splits without shuffling. We divided our data into:

- 72.25% as training set for building the model (85% further split for 15% validation from it)
- 12.75% as validation set for hyperparameter tuning
- 15% as testing set that is kept till the end for testing the final obtained model

4 Models Built

We tested many models to see which one performs best for this price prediction task. Of course, we repeated our experiments 10 times for not fully deterministic models to be sure that we weren't just lucky with our results.

4.1 Linear Regression

The answer to "have you first tried a simple linear regression?" is yes, we did! We implemented it using a lag feature in order to be able to capture the time dependency since on its own LR does not account for it. The results are the following (1 run only because it is deterministic):

Table 1: Linear Regression Performance Results

Horizon	RMSE	MAE	MAPE
30m	6.1938	2.3254	0.1698
1h	6.4008	2.4587	0.1770
1d	17.2318	6.7456	0.8289

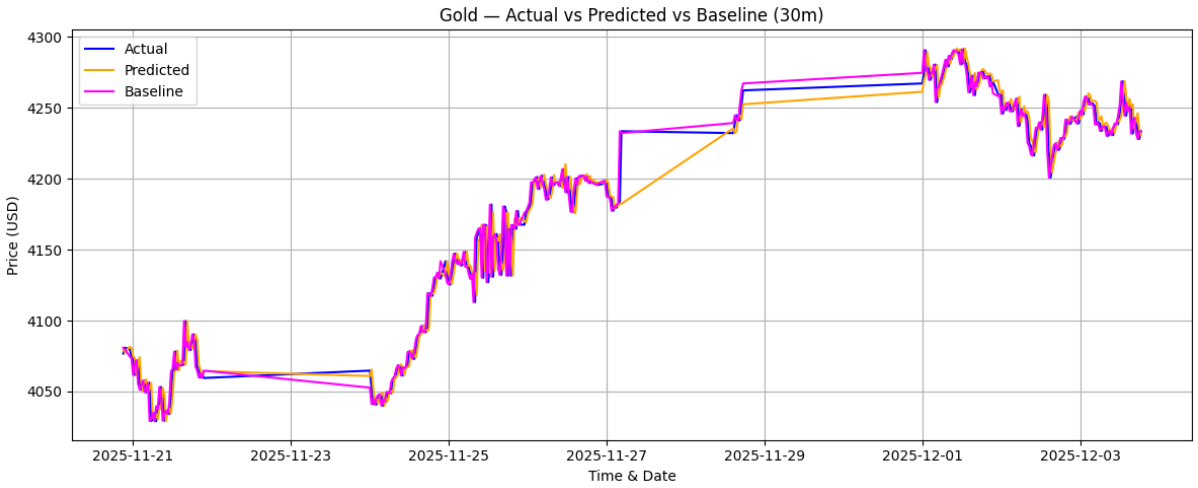

Figure 4: 30m Gold Predictions with LR Over Timesteps

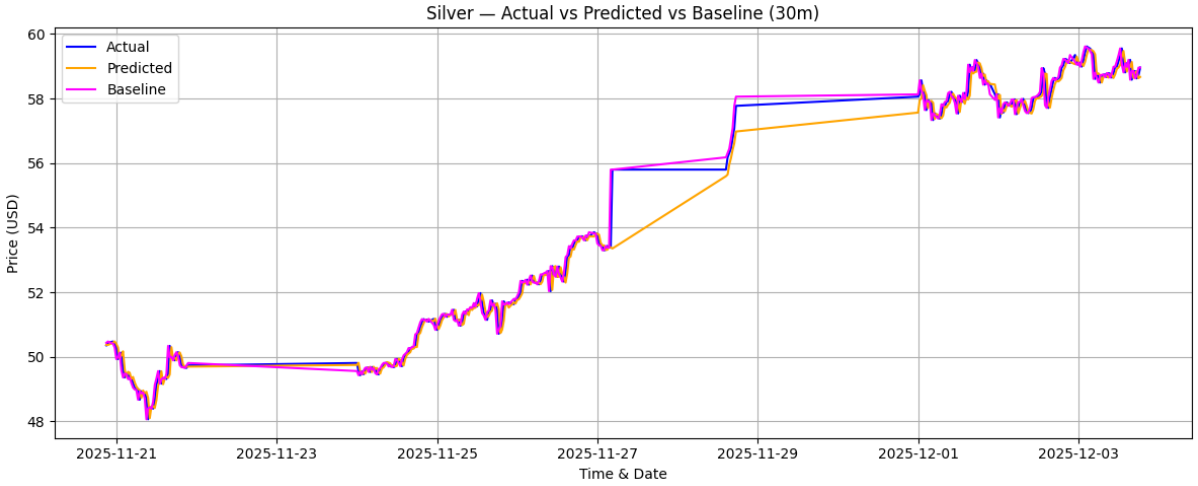

Figure 5: 30m Silver Predictions with LR Over Timesteps

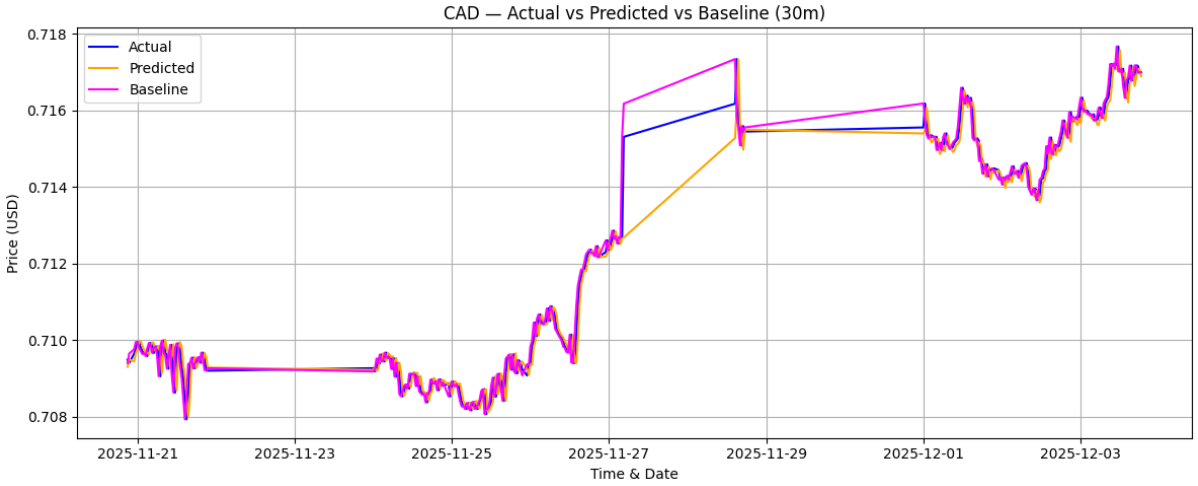

Figure 6: 30m CAD Predictions with LR Over Timesteps

Table 2: Linear Regression Metrics per Asset for 30m

Asset	**RMSE**	**MAE**	**MAPE**
Gold	10.7246	6.8052	0.1632
Silver	0.2703	0.1708	0.3140
CAD	0.0003	0.0002	0.0323

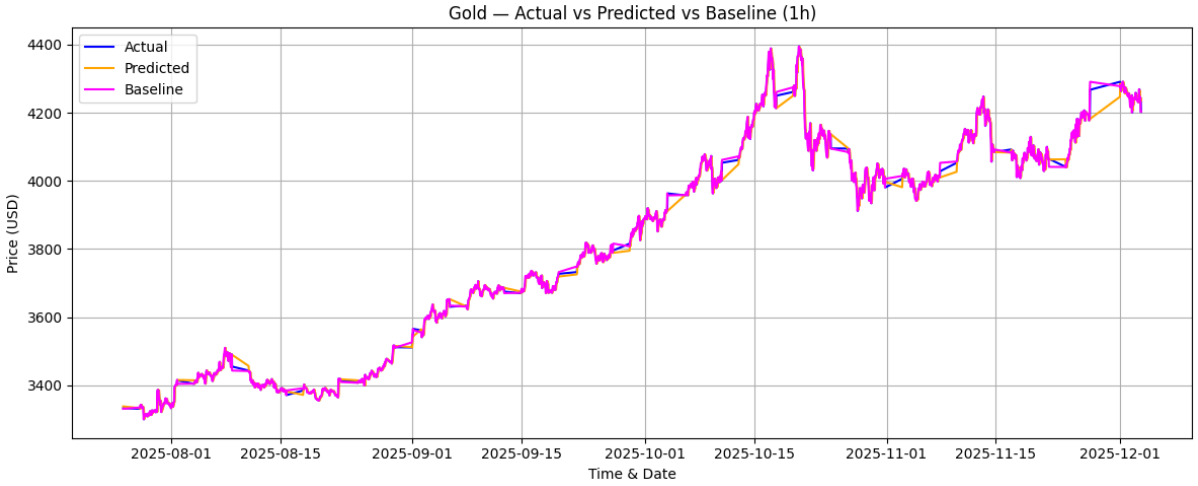

Figure 7: 1h Gold Predictions with LR Over Timesteps

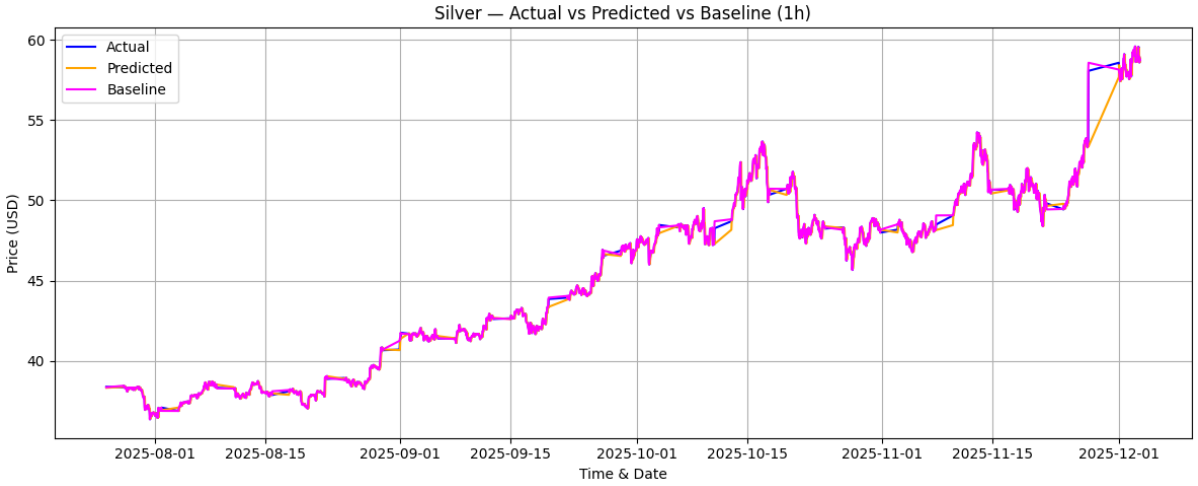

Figure 8: 1h Silver Predictions with LR Over Timesteps

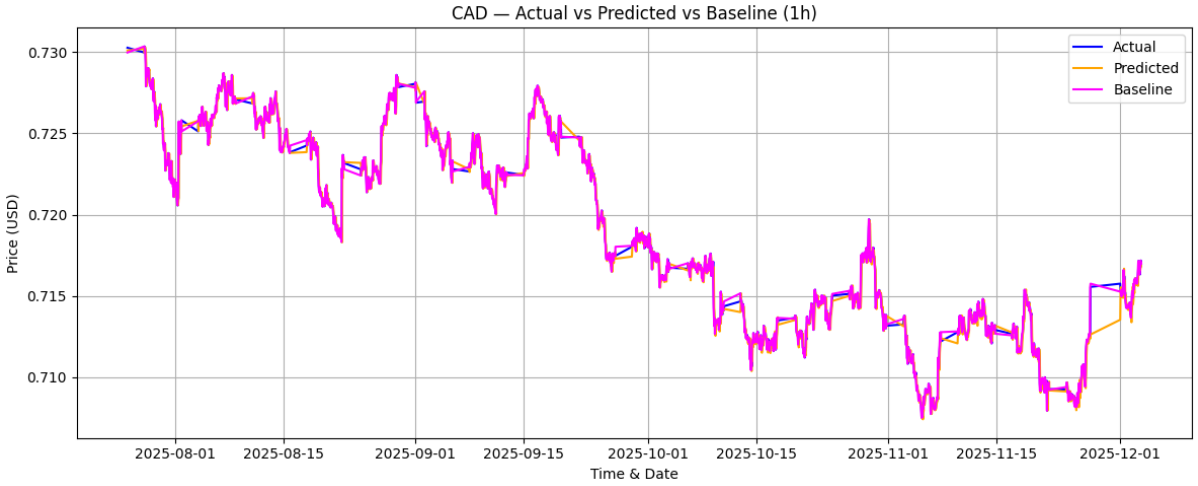

Figure 9: 1h CAD Predictions with LR Over Timesteps

Table 3: Linear Regression Metrics per Asset for 1h

Asset	**RMSE**	**MAE**	**MAPE (%)**
Gold	11.0838	7.2323	0.1854
Silver	0.2418	0.1435	0.3052
CAD	0.0004	0.0003	0.0405

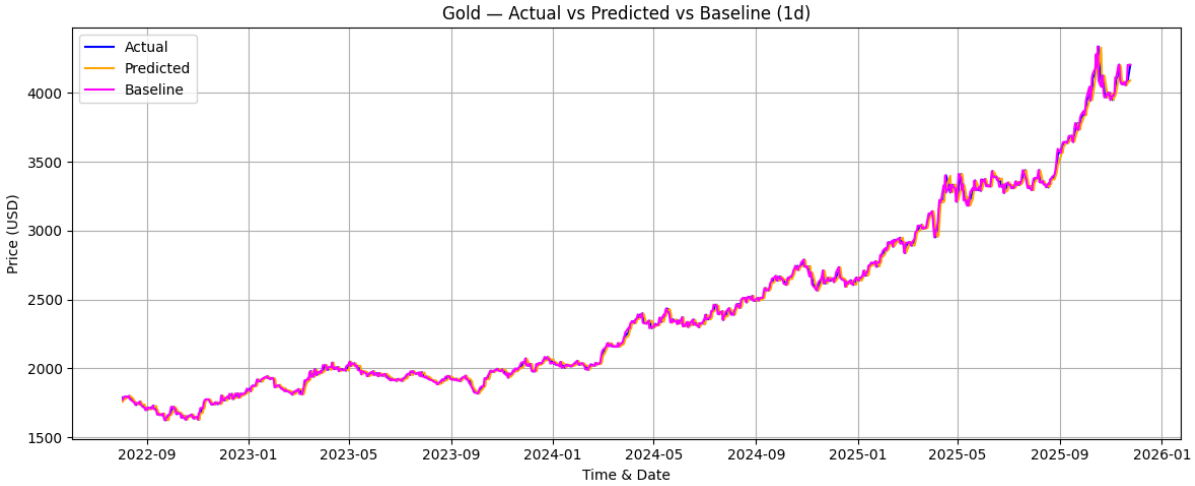

Figure 10: 1 day Gold Predictions with LR Over Timesteps

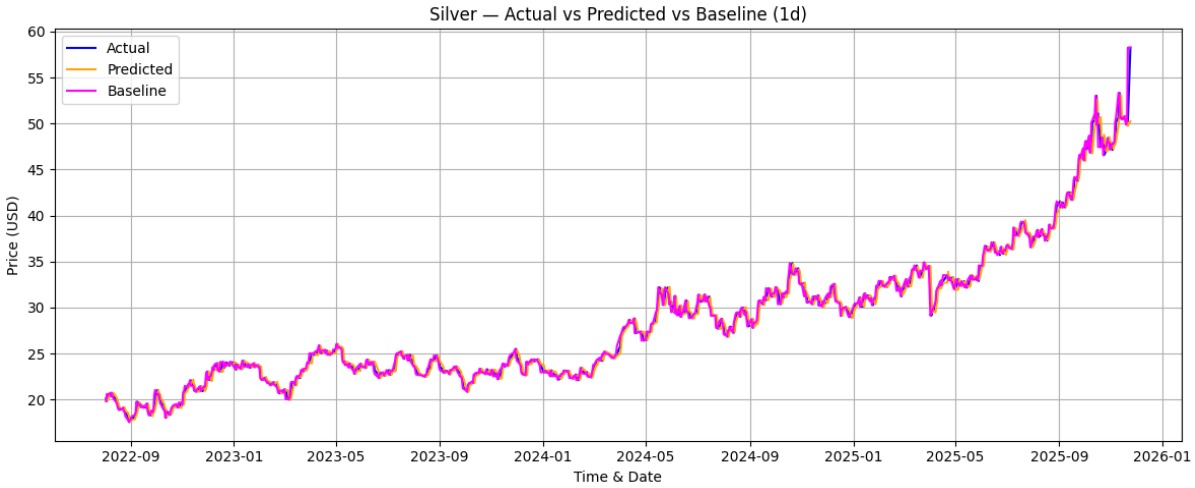

Figure 11: 1 day Silver Predictions with LR Over Timesteps

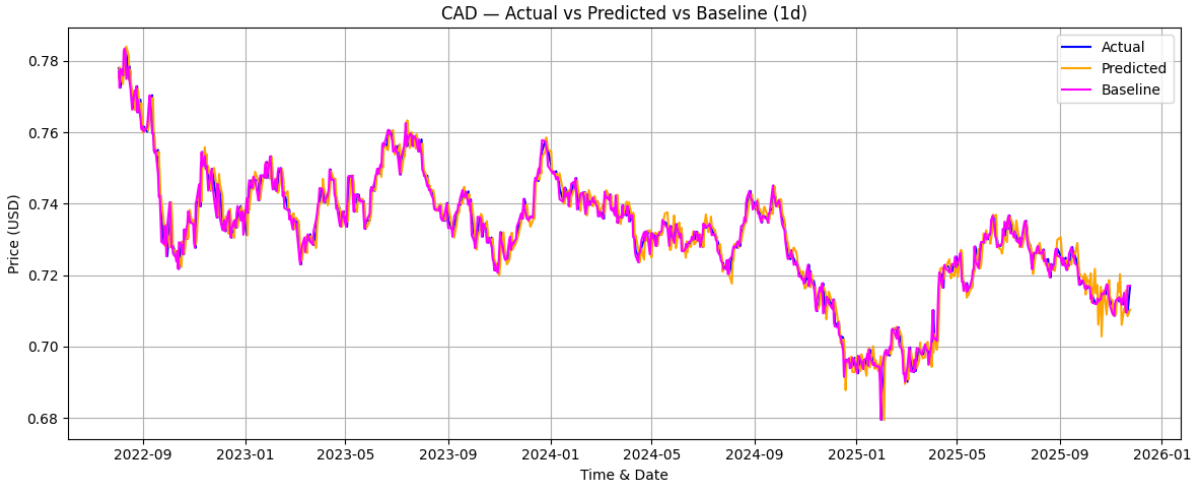

Figure 12: 1 day CAD Predictions with LR Over Timesteps

Table 4: Linear Regression Metrics per Asset for 1 day

Asset	RMSE	MAE	MAPE (%)
Gold	29.8392	19.8199	0.7698
Silver	0.6561	0.4147	1.4217
CAD	0.0029	0.0022	0.2952

However, we are skeptical results will be consistent on the long term as more noise is added to the data and spikes occur, since it Linear Regression assumes linear relationships.

4.2 XGBoost

XGBoost stands for Extreme Gradient Boosting. It is a high-performance implementation of gradient-boosted decision trees designed for speed, scalability, and strong predictive accuracy. It builds trees sequentially, with each new tree correcting the errors of the previous ensemble, while using regularization to prevent overfitting. We thought of trying it because it performs well with small datasets. We added the lag feature for the time series to be taken into account, and our engineered features for more information for XGBoost to be guided by.

Table 5: XGBoost Performance Results

Horizon	RMSE	MAE	MAPE
30m	6.19739	2.44175	0.0033675
1h	271.6373	133.893	0.0777902
1d	459.2777	168.7803	0.0633306

Predictions performance per asset are as follows:

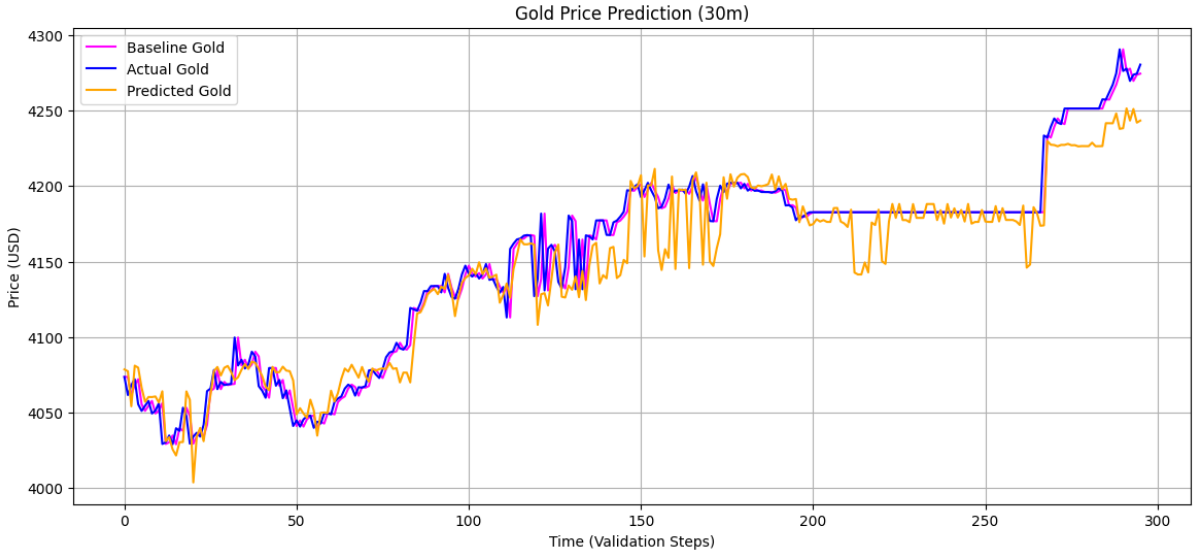

Figure 13: 30m Gold Predictions with XGBoost Over Timesteps

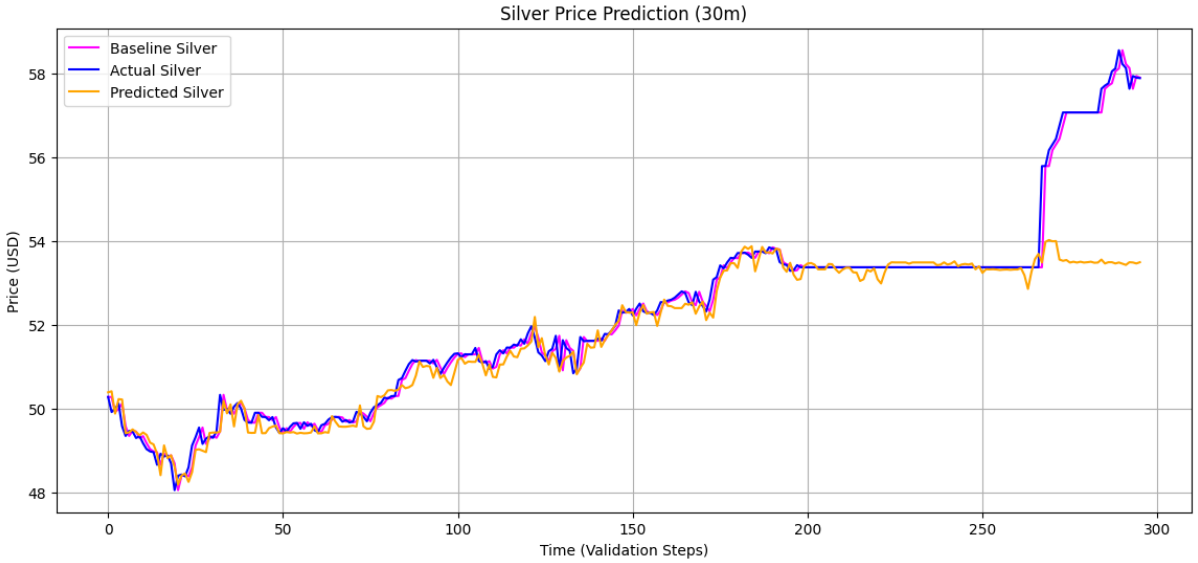

Figure 14: 30m Silver Predictions with XGBoost Over Timesteps

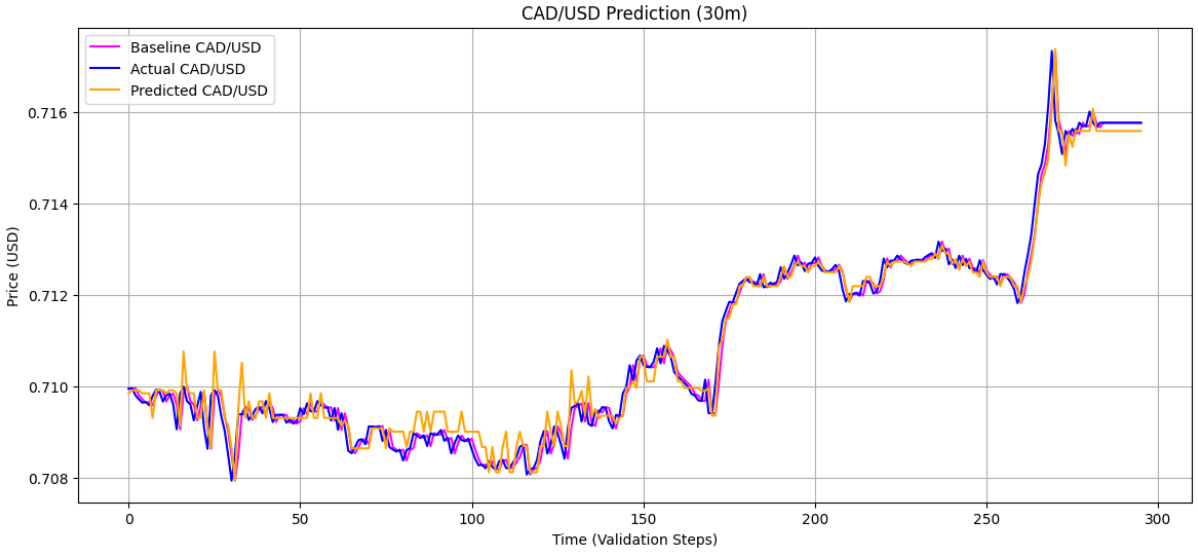

Figure 15: 30m CAD Predictions with XGBoost Over Timesteps

Table 6: XGBoost Metrics per Asset for 30m

Asset	**MAE**	**RMSE**	**MAPE**
Gold	6.8716	10.6758	0.001656
Silver	0.4535	1.1182	0.008147
CAD	0.000213	0.000301	0.000299

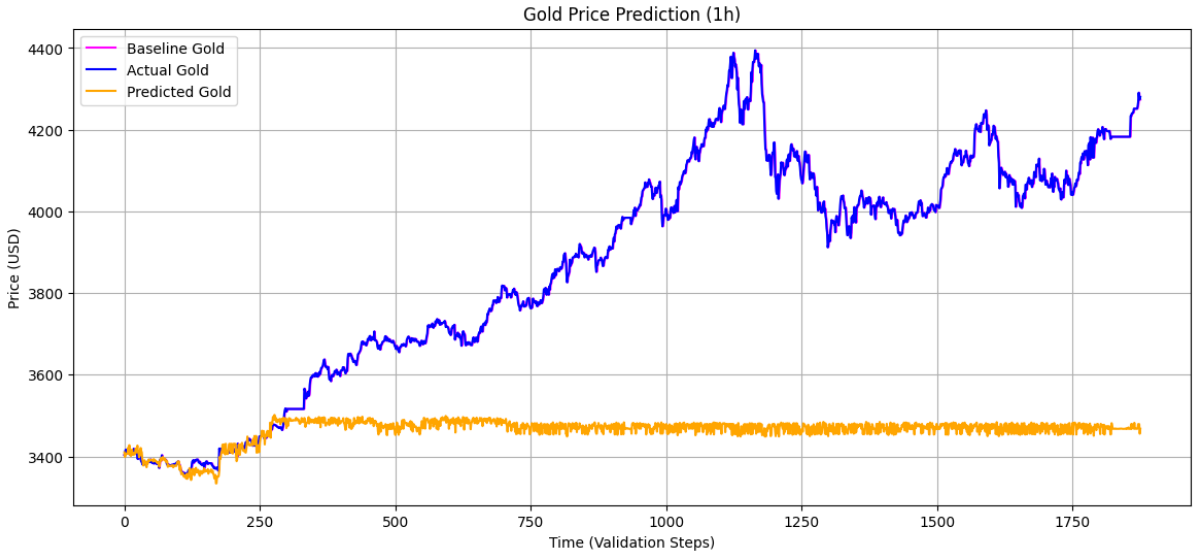

Figure 16: 1h Gold Predictions with XGBoost Over Timesteps

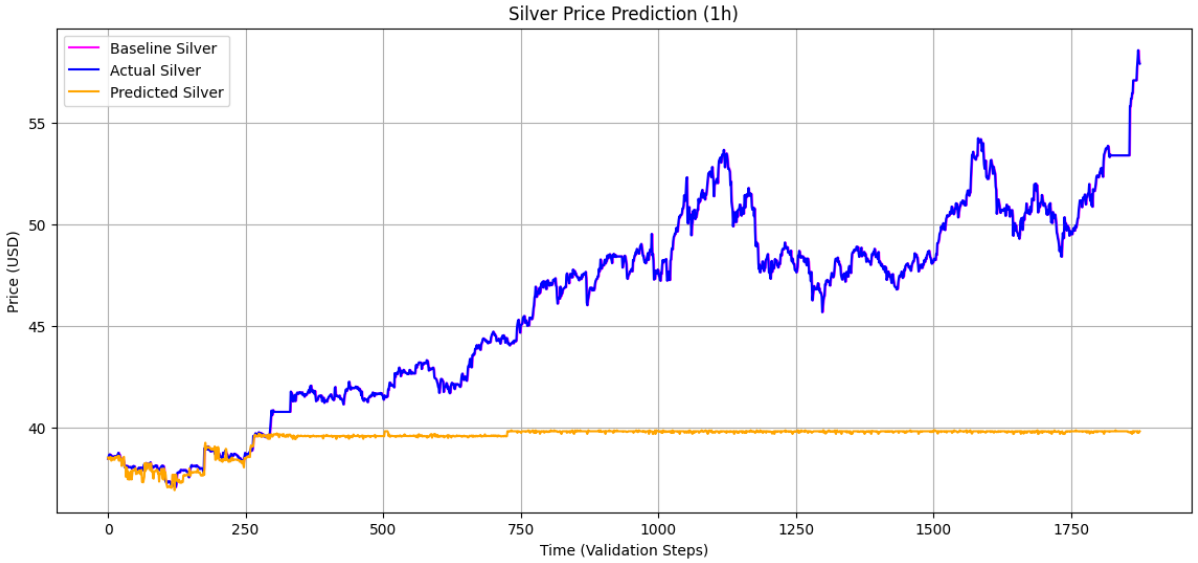

Figure 17: 1h Silver Predictions with XGBoost Over Timesteps

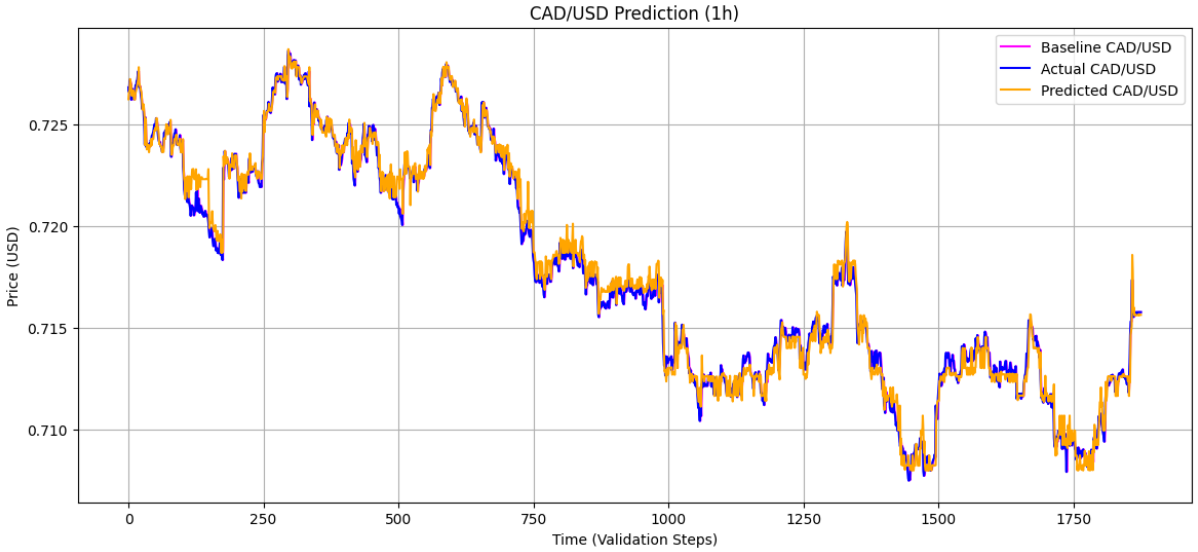

Figure 18: 1h CAD Predictions with XGBoost Over Timesteps

Table 7: XGBoost Metrics per Asset for 1h

Asset	**MAE**	**RMSE**	**MAPE**
Gold	395.0701	470.4223	0.09783
Silver	6.6085	7.9766	0.13483
CAD	0.000508	0.000681	0.000709

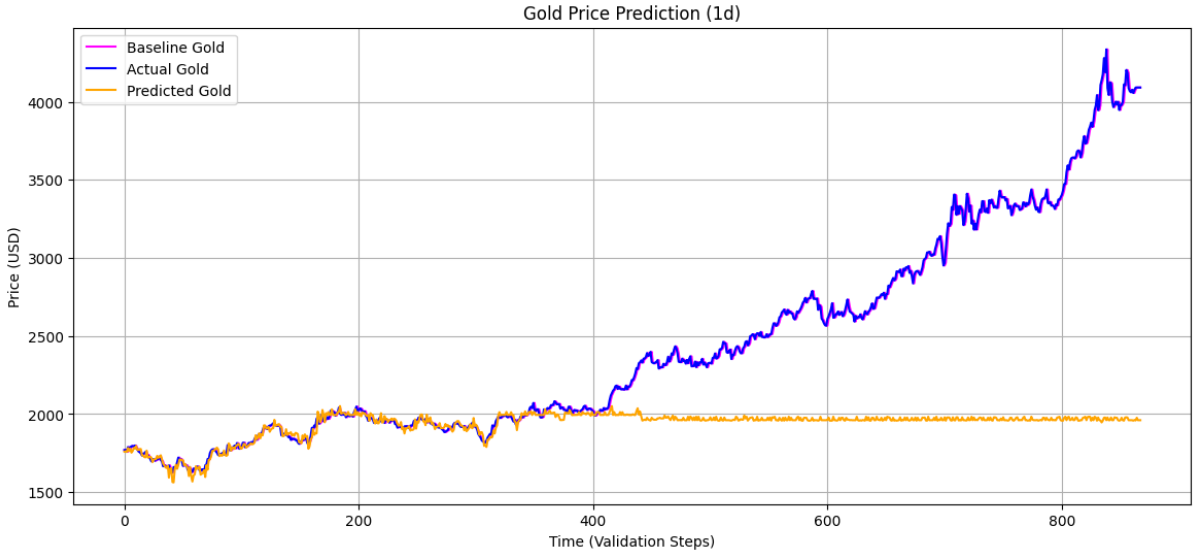

Figure 19: 1 day Gold Predictions with XGBoost Over Timesteps

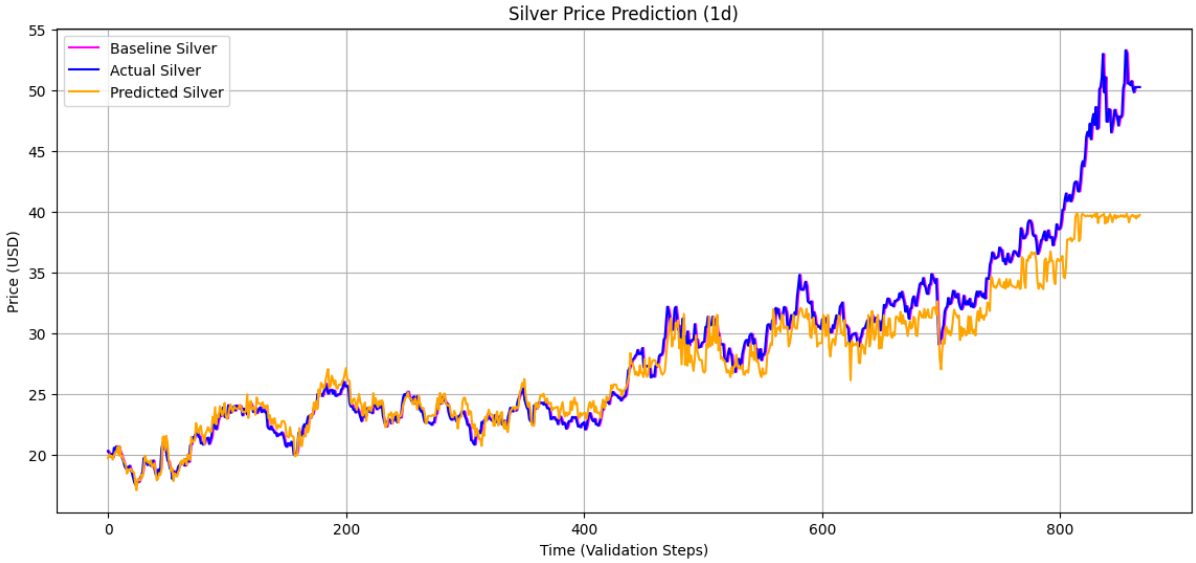

Figure 20: 1 day Silver Predictions with XGBoost Over Timesteps

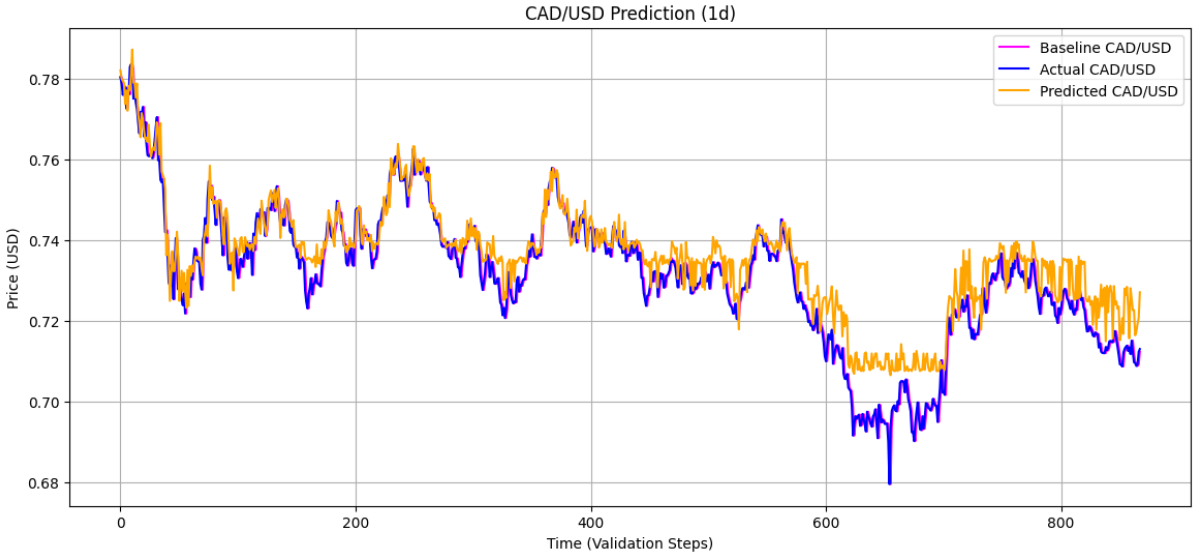

Figure 21: 1 day CAD Predictions with XGBoost Over Timesteps

Table 8: XGBoost Metrics per Asset for 1 day

Asset	MAE	RMSE	MAPE
Gold	505.5450	795.4908	0.16215
Silver	0.7931	1.5922	0.02386
CAD	0.002906	0.003682	0.003980

We can see that as time goes on, the performance of XGBoost deteriorates.

4.3 LSTM and GRU

4.3.1 Hyperparameter Tuning Including Model Type

To choose the best RNN type, we tried a random search hyperparameter tuning method that includes:

- **RNN type:** LSTM or GRU
- **The window size:** between 10 and 100, with a step size of 10
- **The number of units in the first RNN layer:** between 16 and 128, with a step size of 16
- **The number of units in the second RNN layer:** between 8 and 64, with a step size of 8
- **The number of units in the dense layer:** between 8 and 64, with a step size of 8
- **The dropout percentage between layers:** between 1% and 5%, with a step size of 1%
- **The L2 regularization value:** between 1e-6, 1e-5, 1e-4

- **The learning rate value:** between 1e-2, 5e-3, 1e-3, 5e-4, 1e-4

It executed 30 trials, and the objective was to minimize the Root Mean Square Error. The best parameters were found to be the following for each horizon:

- **30min:** window: 20, RNN type: GRU, units for layer 1: 112, L2 reg layer 1: 1e-06, dropout 1: 0.1, units for layer 2: 40, L2 reg layer 2: 0.0001, dropout 2: 0.1, dense units: 32, learning rate: 0.005
- **1 hour:** window: 40, RNN type: GRU, units for layer 1: 48, L2 reg layer 1: 1e-06, dropout 1: 0.2, units for layer 2: 64, L2 reg layer 2: 1e-05, dropout 2: 0.2, dense units: 64, learning rate: 0.001
- **1 day:** window: 70, RNN type: GRU, units for layer 1: 48, L2 reg layer 1: 0.0001, dropout 1: 0.4, units for layer 2: 16, L2 reg layer 2: 0.0001, dropout 2: 0.1, dense units: 8, learning rate: 0.0005

Since GRU performed best in all cases, we decided to further perform a more rigorous hyperparameter tuning on GRU. We hypothesise that GRU performed better because it has fewer parameters than LSTM, thus needing less data for good results without overfitting.

4.3.2 GRU with Hyperparameter Tuning

Hyperparameter tuning for GRU was performed using hyperband. We chose it because we found that it is more efficient than random search. It uses an early-stopping strategy to eliminate poor configurations quickly. It begins by sampling many random hyperparameter sets and training each for only a few epochs. Based on their performance, only the most promising configurations are allowed to continue for more epochs, while the weaker ones are discarded early. This allows hyperband to explore a wider hyperparameter space while also reducing training time.

The best parameters were found to be the following for each horizon:

- **30min:** window: 80, units for layer 1: 112, L2 reg layer 1: 1e-05, dropout 1: 0.1, units for layer 2: 64, L2 reg layer 2: 1e-06, dropout 2: 0.25, learning rate: 0.005
- **1 hour:** window: 40, units for layer 1: 128, L2 reg layer 1: 1e-06, dropout 1: 0.1, units for layer 2: 64, L2 reg layer 2: 1e-06, dropout 2: 0.15, learning rate: 0.001
- **1 day:** window: 50, units for layer 1: 112, L2 reg layer 1: 1e-05, dropout 1: 0.2, units for layer 2: 48, L2 reg layer 2: 1e-06, dropout 2: 0.1, learning rate: 0.01

The GRU metrics over 10 runs, using the best parameters found, are the following:

Table 9: GRU Performance Results

Horizon	RMSE	MAE	MAPE
30m	7.71995 (± 1.04743)	3.54573 (± 0.70682)	0.0022149 (± 0.00046768)
1h	40.96897 (± 15.52618)	21.06752 (± 9.02719)	0.010016 (± 0.0044966)
1d	122.95768 (± 39.36922)	51.275520 (± 21.25859)	0.036102802 (± 0.0096851)

The high standard deviation that can be seen in Table 9, especially for gold, indicate that some days are much harder to predict than others, which is normal for such volatile assets.

The following figures illustrate how our GRU model predictions perform for each asset compared to the actual price and a baseline which takes the next price to be equal to the current actual one:

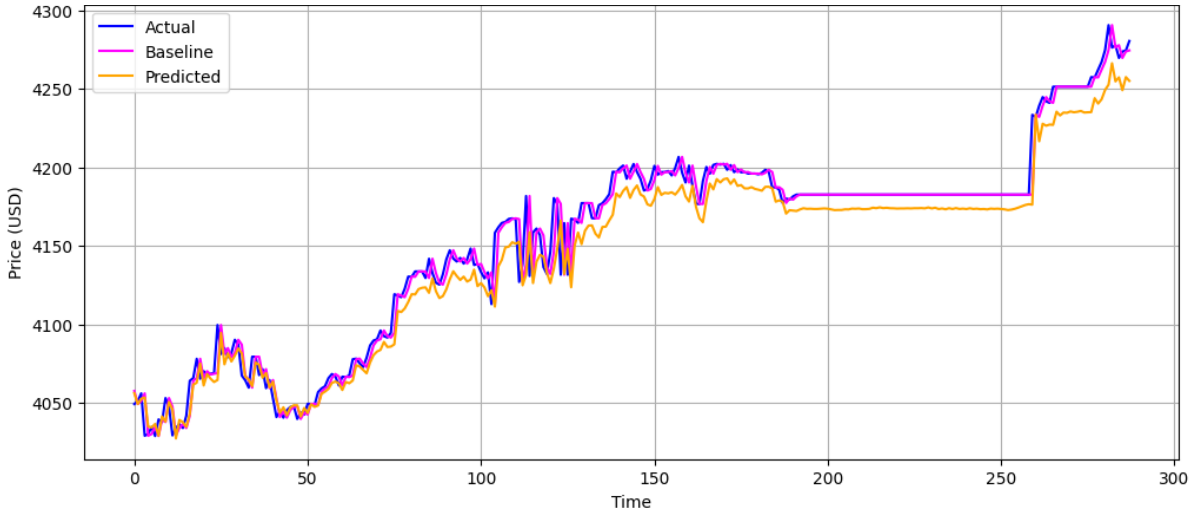

Figure 22: 30m Gold Predictions with GRU Over Timesteps

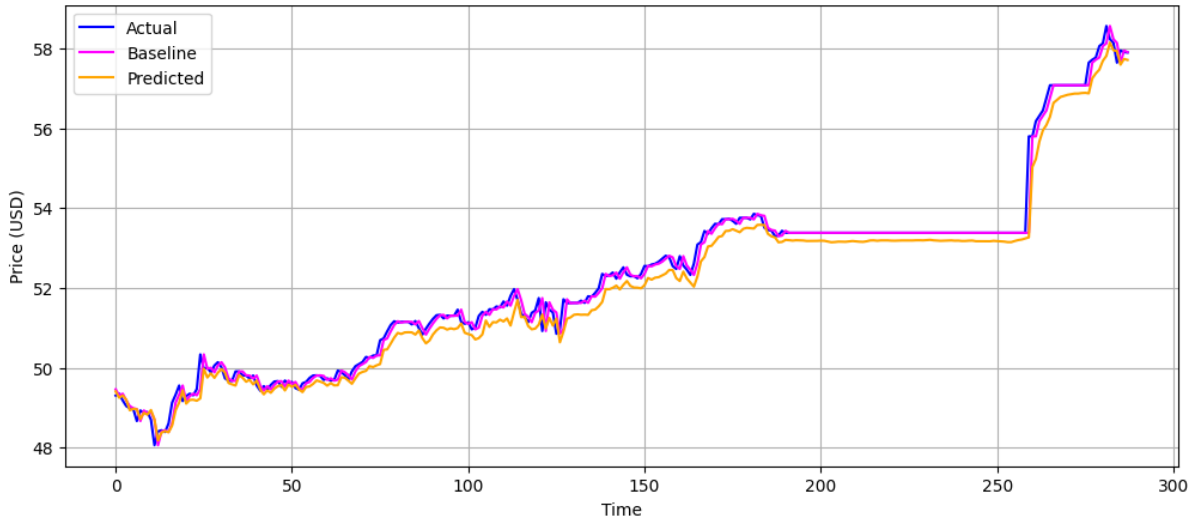

Figure 23: 30m Silver Predictions with GRU Over Timesteps

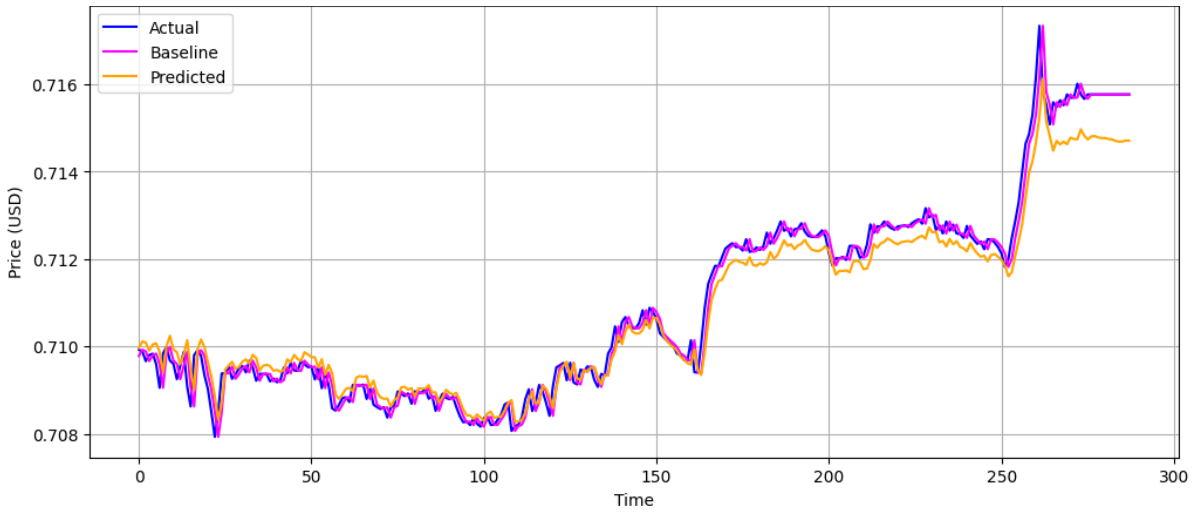

Figure 24: 30m CAD Predictions with GRU Over Timesteps

Table 10: GRU Metrics per Asset for 30m

Asset	**MAE**	**RMSE**	**MAPE**
Gold	11.7734	14.4789	0.2824
Silver	0.2799	0.3551	0.5319
CAD	0.000377	0.000495	0.0530

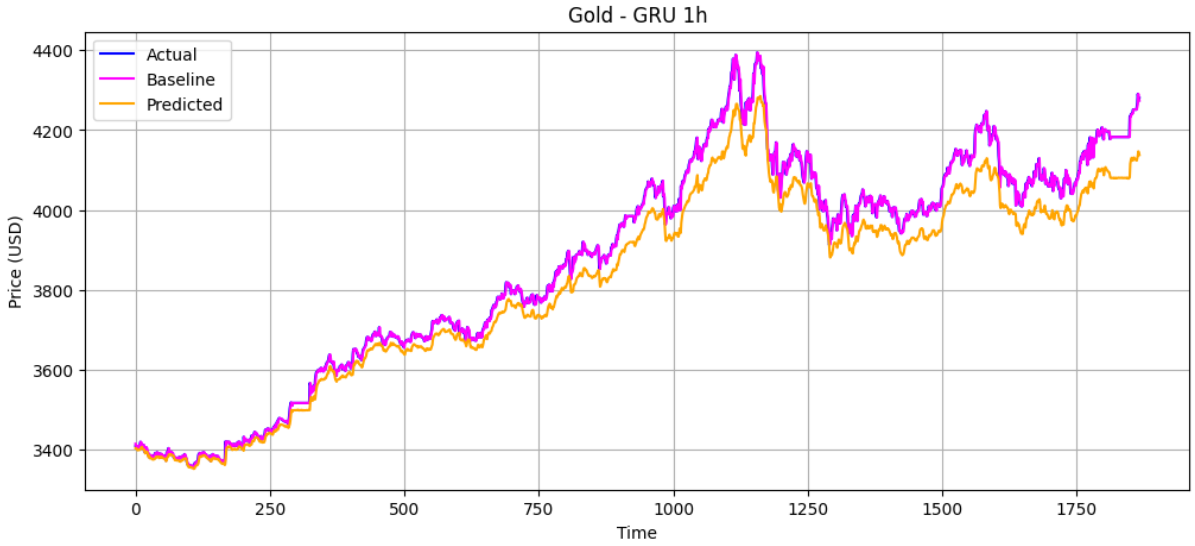

Figure 25: 1h Gold Predictions with GRU Over Timesteps

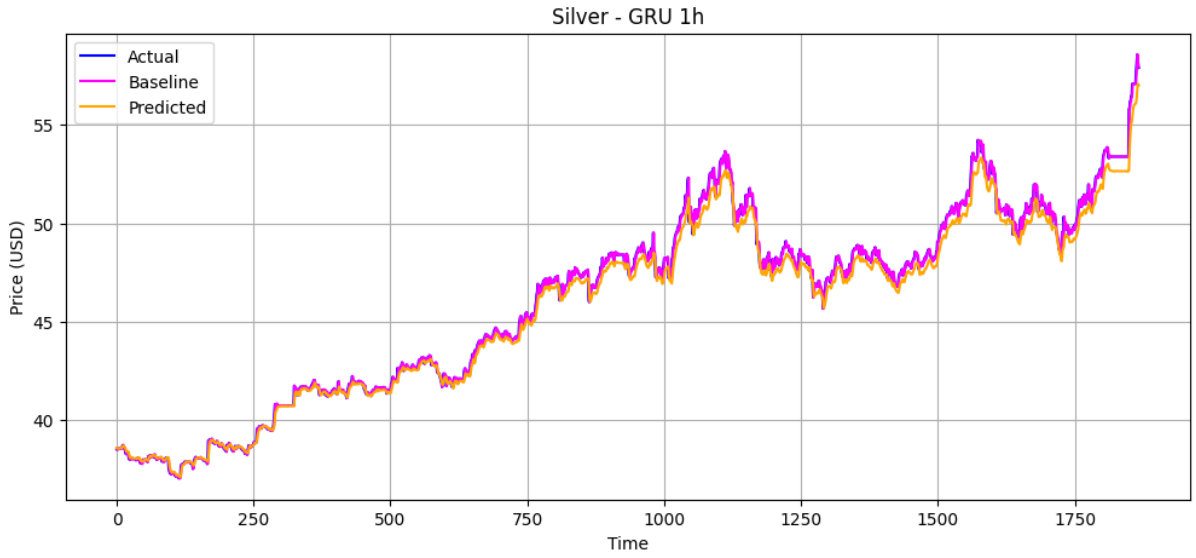

Figure 26: 1h Silver Predictions with GRU Over Timesteps

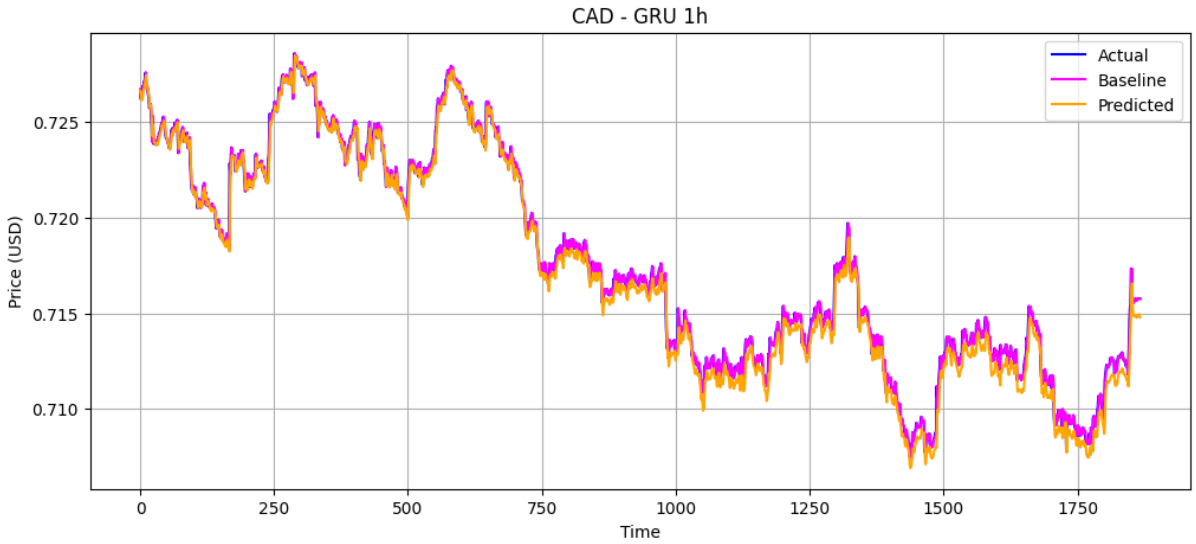

Figure 27: 1h CAD Predictions with GRU Over Timesteps

Table 11: GRU Metrics per Asset for 1h

Asset	**MAE**	**RMSE**	**MAPE**
Gold	53.4608	62.6727	1.3327
Silver	0.3959	0.5147	0.8132
CAD	0.000496	0.000624	0.0692

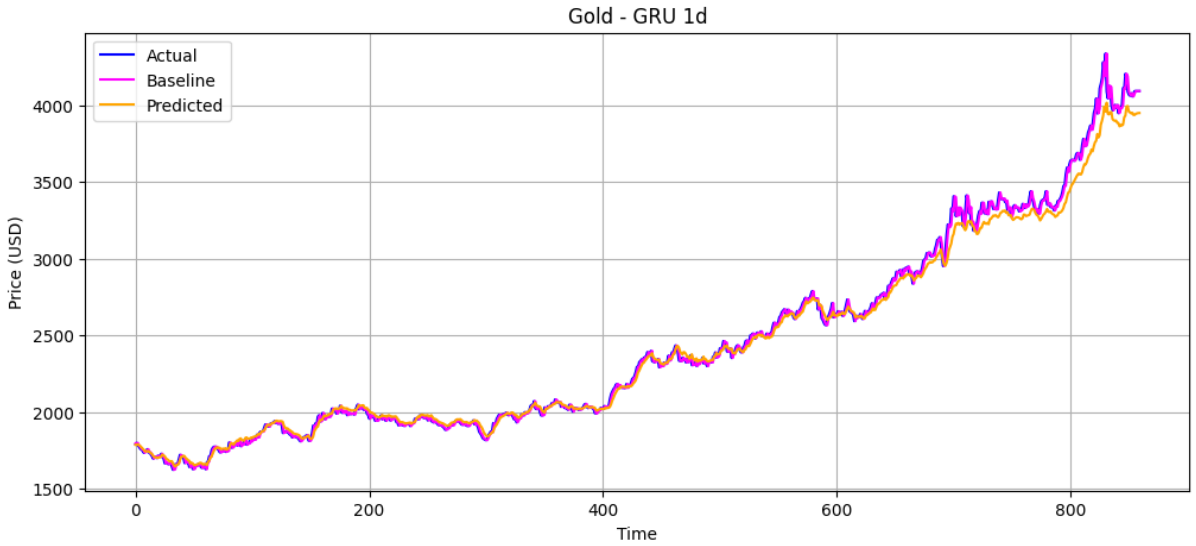

Figure 28: 1 day Gold Predictions with GRU Over Timesteps

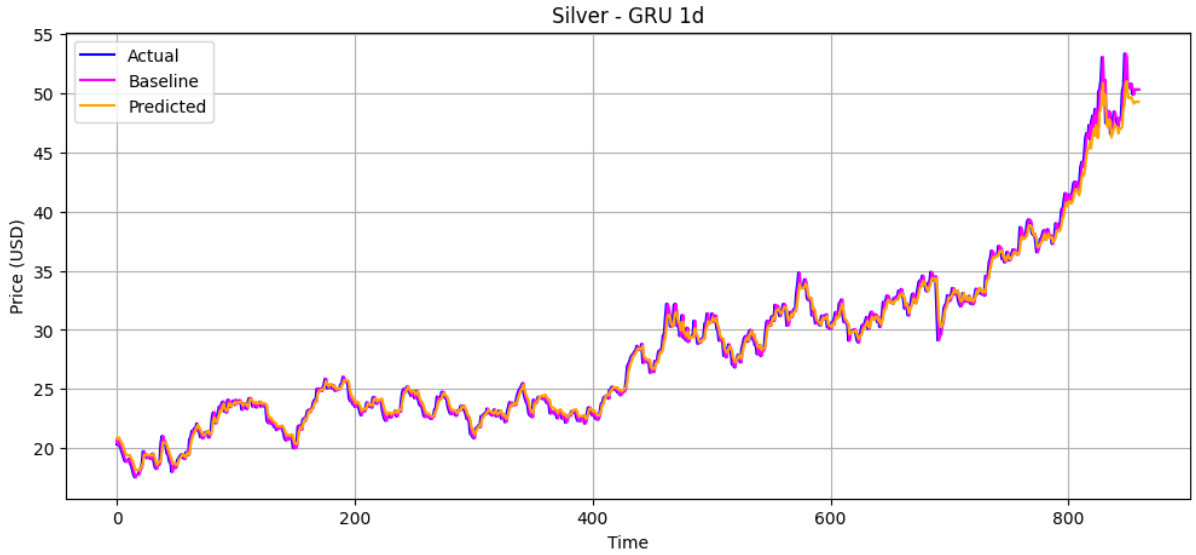

Figure 29: 1 day Silver Predictions with GRU Over Timesteps

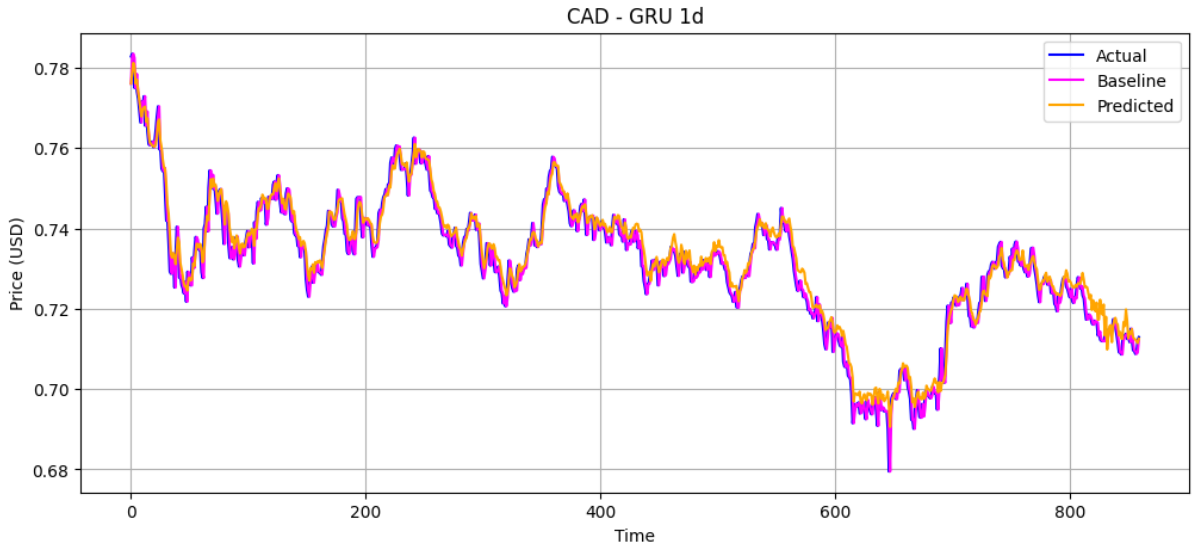

Figure 30: 1 day CAD Predictions with GRU Over Timesteps

Table 12: GRU Metrics per Asset for 1 day

Asset	MAE	RMSE	MAPE
Gold	40.3682	63.6832	1.4360
Silver	0.4899	0.6973	1.6884
CAD	0.002662	0.003363	0.3653

4.4 What about a basic, simple RNN model?

Initially we did not consider a simple RNN because of the gradient problem and instead directly began trying the modified types. However, we then realized that a basic RNN might work perfectly well here because of the small dataset and small window size. So we experimented in the same way as we did with GRUs, using hyperband for hyperparameter tuning.

The best parameters were found to be the following for each horizon:

- **30min:** window: 50, units for layer 1: 32, L2 reg layer 1: 1e-06, dropout 1: 0.15, units for layer 2: 48, L2 reg layer 2: 1e-06, dropout 2: 0.15, learning rate: 0.005
- **1 hour:** window: 70, units for layer 1: 48, L2 reg layer 1: 0.0001, dropout 1: 0.1, units for layer 2: 56, L2 reg layer 2: 1e-06, dropout 2: 0.1, learning rate: 0.0005
- **1 day:** window: 50, units for layer 1: 112, L2 reg layer 1: 1e-05, dropout 1: 0.25, units for layer 2: 56, L2 reg layer 2: 1e-06, dropout 2: 0.15, learning rate: 0.001

The RNN metrics over 10 runs, using the best parameters found, are the following:

Table 13: RNN Performance Results

Horizon	RMSE	MAE	MAPE
30m	7.41175(± 1.4389)	3.36195 (± 0.913459)	0.0022963 (± 0.00035218)
1h	20.78361 (± 7.95785)	10.29797 (± 4.25616)	0.0065735 (± 0.0020245)
1d	49.84613 (± 19.28924)	20.12864 (± 9.68006)	0.017496 (± 0.0051285)

The following figures illustrate how our GRU model predictions perform for each asset compared to the actual price and a baseline which takes the next price to be equal to the current actual one:

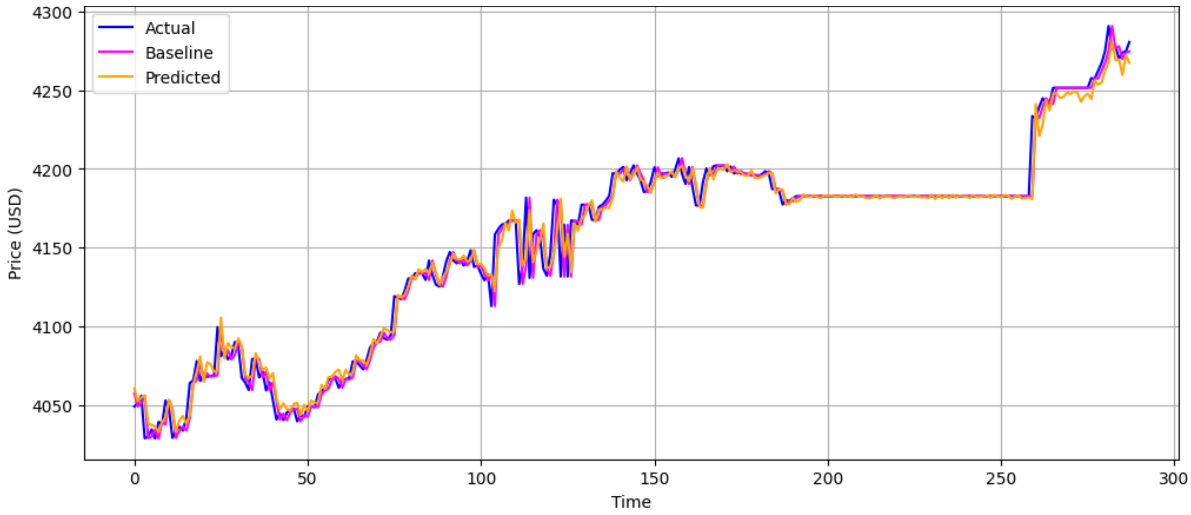

Figure 31: 30m Gold Predictions with RNN Over Timesteps

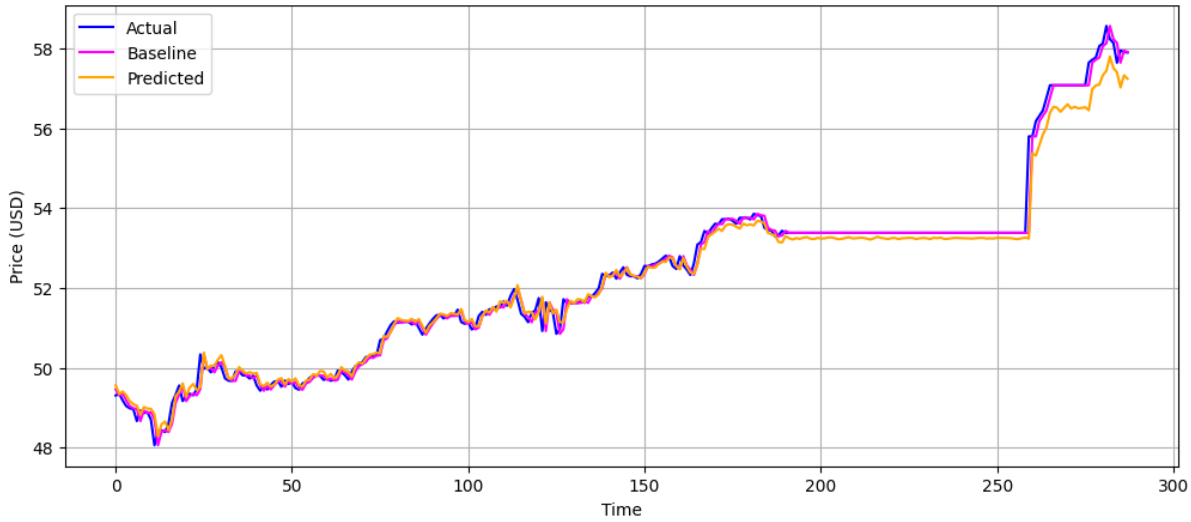

Figure 32: 30m Silver Predictions with RNN Over Timesteps

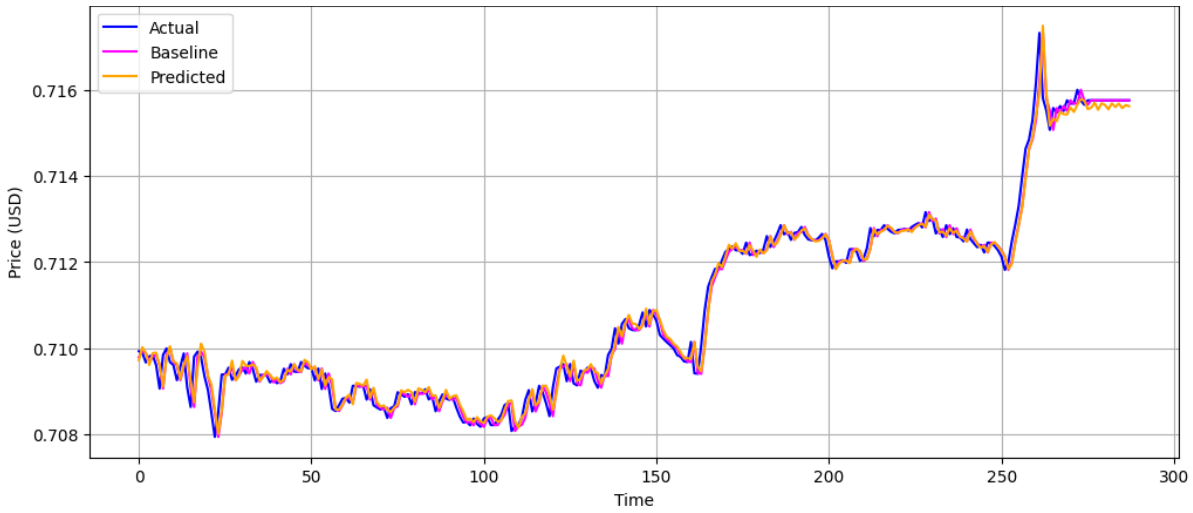

Figure 33: 30m CAD Predictions with RNN Over Timesteps

Table 14: RNN Metrics per Asset for 30m

Asset	**MAE**	**RMSE**	**MAPE**
Gold	6.0244	10.1856	0.1453
Silver	0.2061	0.3213	0.3857
CAD	0.000217	0.000303	0.0306

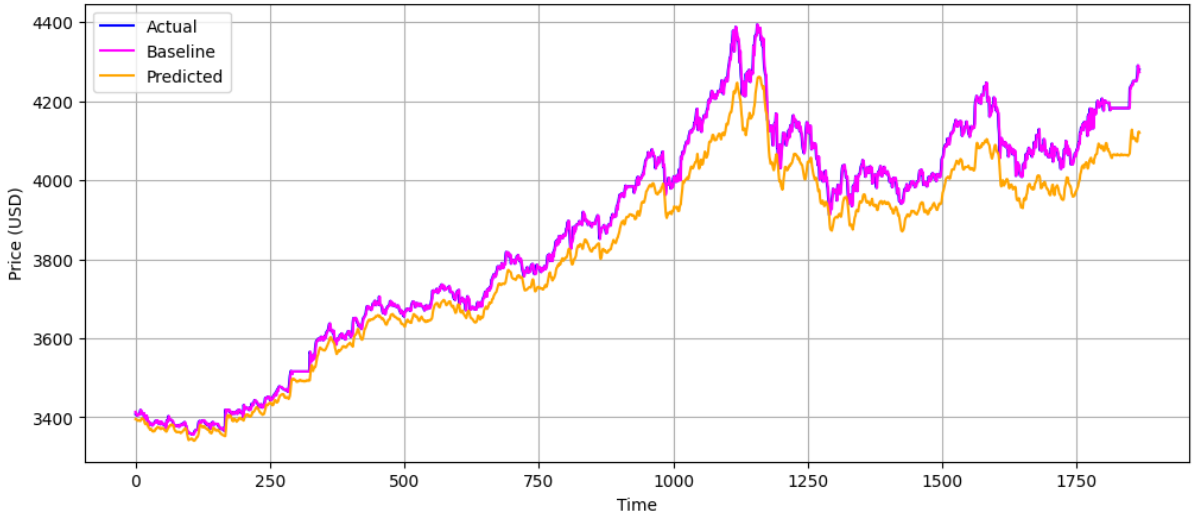

Figure 34: 1h Gold Predictions with RNN Over Timesteps

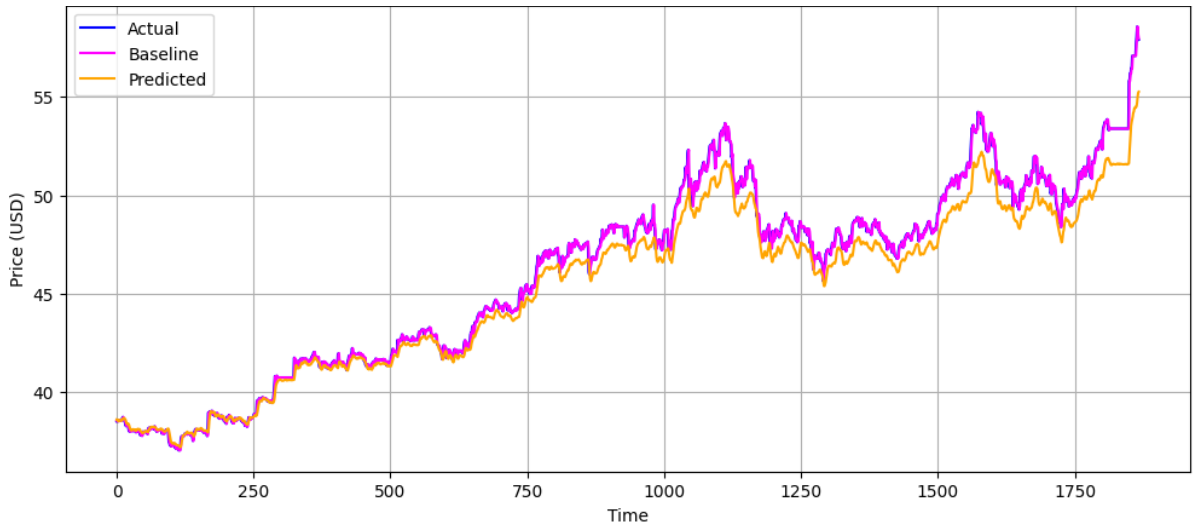

Figure 35: 1h Silver Predictions with RNN Over Timesteps

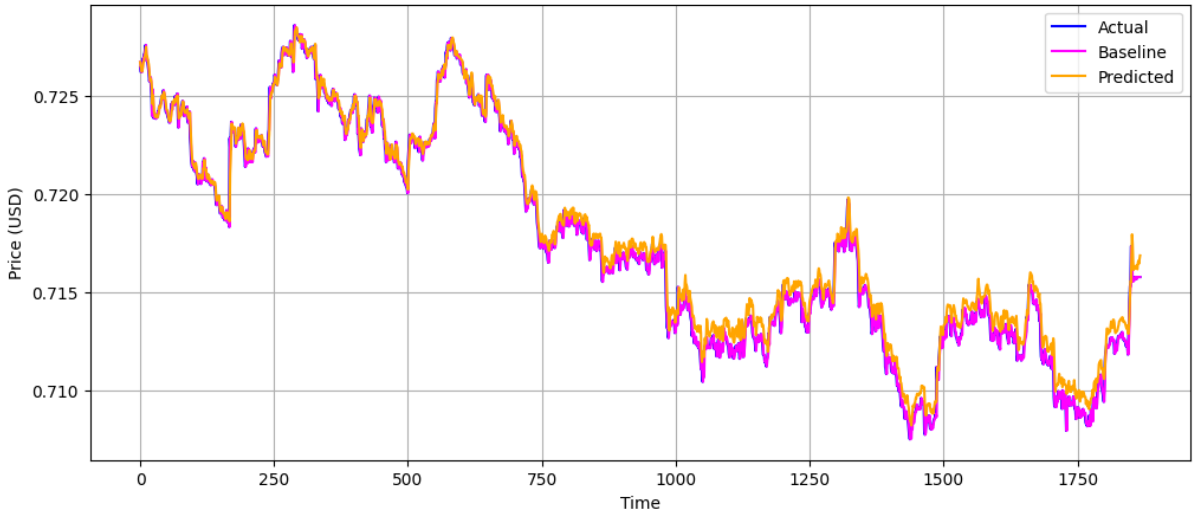

Figure 36: 1h CAD Predictions with RNN Over Timesteps

Table 15: RNN Metrics per Asset for 1h

Asset	**MAE**	**RMSE**	**MAPE**
Gold	63.3800	73.2060	1.5839
Silver	0.7886	1.0048	1.6064
CAD	0.000536	0.000674	0.0749

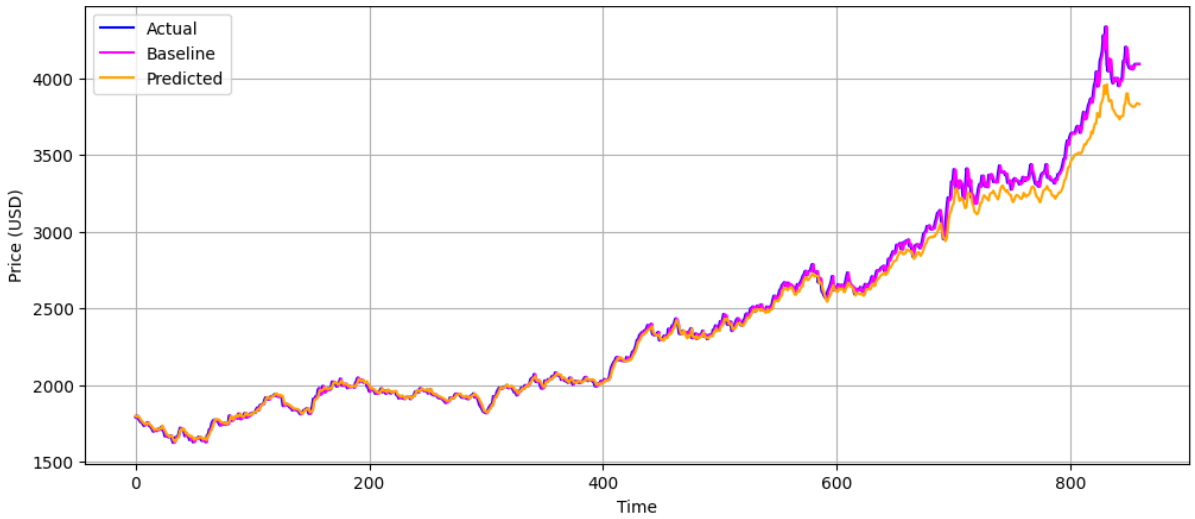

Figure 37: 1 day Gold Predictions with RNN Over Timesteps

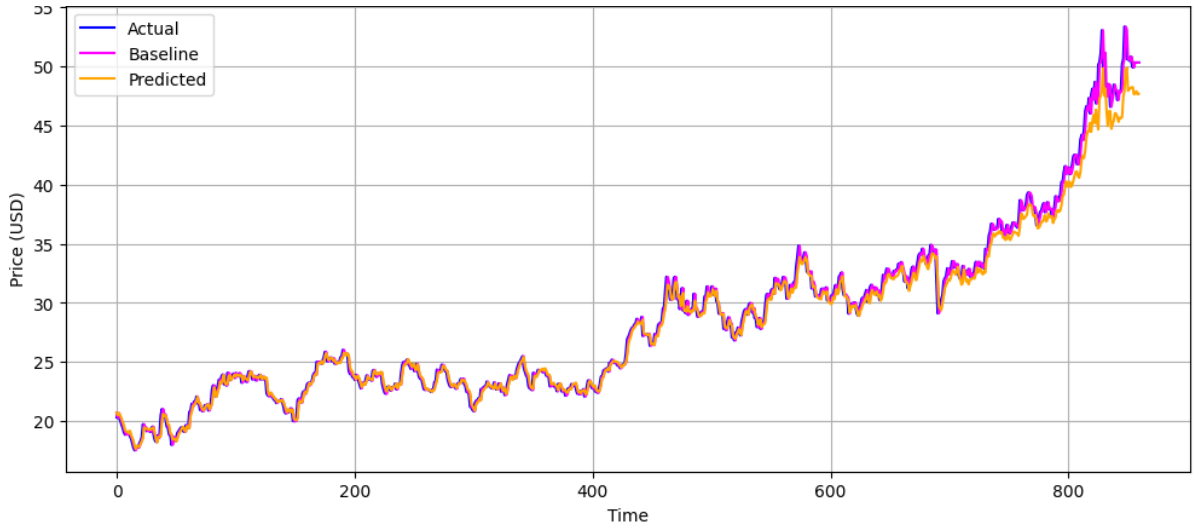

Figure 38: 1 day Silver Predictions with RNN Over Timesteps

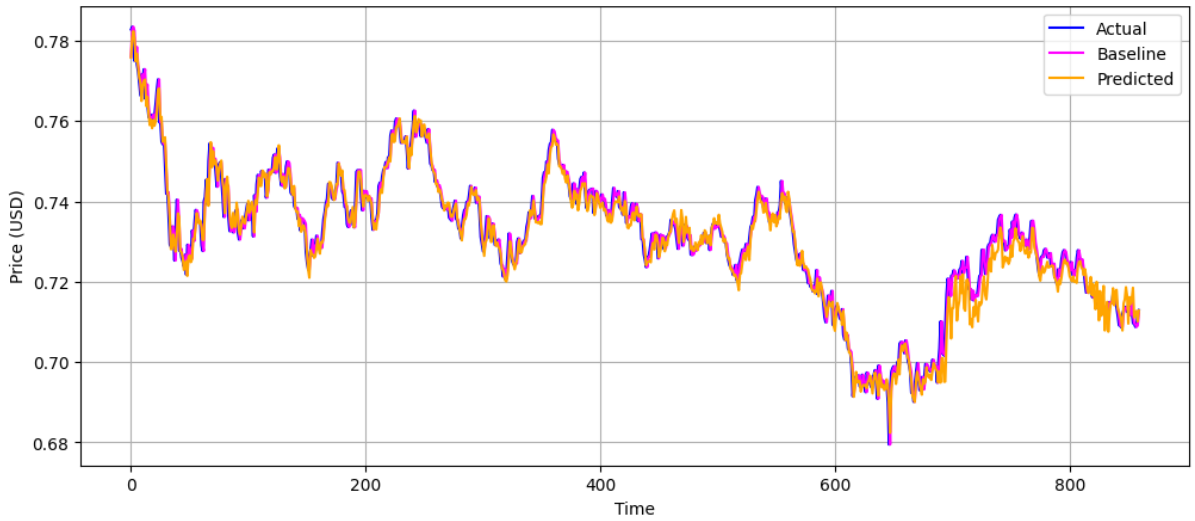

Figure 39: 1 day CAD Predictions with RNN Over Timesteps

Table 16: RNN Metrics per Asset for 1 day

Asset	**MAE**	**RMSE**	**MAPE**
Gold	48.3792	82.4580	1.6080
Silver	0.5786	0.9009	1.8308
CAD	0.002582	0.003459	0.3535

4.5 Ruling Out Transformers

After doing some research about the Transformer architecture, we found that it requires too much data for pretraining and training [2]. We then considered using pretrained models but realized they will still require a lot of data for training to avoid the risk of overfitting.

4.6 Best Models

According to the obtained results both overall and per asset, RNN gives the most reliable predictions for all horizons, so it was selected as our final model to use.

5 Easiest Horizon To Predict

The easiest horizon to predict turned out to be 30min, followed by 1h and finally 1 day. We think this is because in a shorter time interval, the future price is very close to the current price unless something external disrupts the economy. Financial data is noisy, and the more time passes the more noise accumulates between data points which makes predictions more difficult.

6 Conclusion

In conclusion, we are ready to be rich. Less importantly, we also learned how to choose potential models for a certain task, experiment with them and with the features of the dataset, and finally select the best one. We also realized how much the dataset size affects the model choice.

6.1 Future Work

The models obtained here are still far from the best performance that can be achieved. Future models can be improved in many ways, namely:

- using a much bigger dataset for training from a source other than yahoo finance.
- accounting for seasonal patterns which requires a bigger dataset for a meaningful improvement.
- adding a sentiment analysis component for political events that might affect buyer behavior.
- using datasets with other economic factors that may affect gold, silver and CAD prices.

References

[1] R. Aroussi, "yfinance: Download market data from yahoo! finance's api," https://ranaroussi.github.io/yfinance/, 2025.

[2] R. Shao and X.-J. Bi, "Transformers meet small datasets," *IEEE Access*, vol. 10, pp. 118 454–118 464, 2022.

A Appendix

A.1 Code

The full implementation is in the submitted folder on Blackboard. This is the link to the Google Colab testing notebook if you want to test the final models: `https://drive.google.com/file/d/1vuEFOMFz_ostlsL2zZWCEJLh6mCUNH4G/view?usp=sharing`

A.2 Regarding figures

- Graphs and figures were generated using matplotlib and seaborn in Jupyter Notebook.