41040 AT2 AI Application Showcase

GROUP:

AUTHORS: Felix Colin Lianto (25615838), Pemapol Sripratipbundit (25779253), Yeamin Ahmed (25649740)

DATE: 27 October 2025

SEMESTER: Spring

YEAR: 2025

UNIVERSITY OF TECHNOLOGY SYDNEY

LINK TO SOLUTION NOTEBOOK:

https://colab.research.google.com/drive/14wuVEpnddeJG8M18ZF4XRiyW8Hd885aR?usp=sharing (SimpleRNN)

https://colab.research.google.com/drive/1XHOFUnCYrV8JwKqTTeKHDKAkxuIKAEYr?usp=sharing (LSTM)

https://colab.research.google.com/drive/1bDp_yCAcXyT1_iVA3lwMxUiR0rJhSPAL?usp=sharing (GRU)

Bachelor of AI

Faculty of Engineering & IT

UTS

UTS CRICOS 00099F

Table of Contents

Summary

This project explored how artificial intelligence can be used to predict stock prices, focusing on three main types of Recurrent Neural Networks (RNNs): **SimpleRNN**, **Long Short-Term Memory (LSTM)**, and **Gated Recurrent Unit (GRU)**. The aim was to find out which model could best handle the unpredictable and rapidly changing nature of stock market data.

We used stock price data from **Yahoo Finance (2018–2023)**, which included open, close, high, low, and volume values. All experiments were run using Python in Google Colab, supported by tools such as TensorFlow, Keras, and scikit-learn. Before training, the dataset was cleaned and standardized so that the models could learn more efficiently from patterns in the data.

Each model was tested under various hyperparameter settings, and performance was measured using Root Mean Square Error (RMSE), where a lower value means better accuracy. Across 72 experiments per model, the GRU consistently performed the best, achieving the lowest testing RMSE of 343.83. This makes GRU the most effective model among the three, as it not only learned faster but also provided more reliable predictions.

The GRU's success is mainly due to its simpler design and two gating mechanisms (reset and update gates), which allow it to remember only the most relevant past data and ignore noise. In comparison, SimpleRNN struggled with gradient vanishing problems, while LSTM required more computational power without showing better accuracy.

An interesting discovery was that increasing the look-back value (the amount of past data the model looks at) did not necessarily improve predictions. In fact, using just one previous day's data worked best. This suggests that stock price changes are highly situational and often depend more on short-term fluctuations than long-term trends.

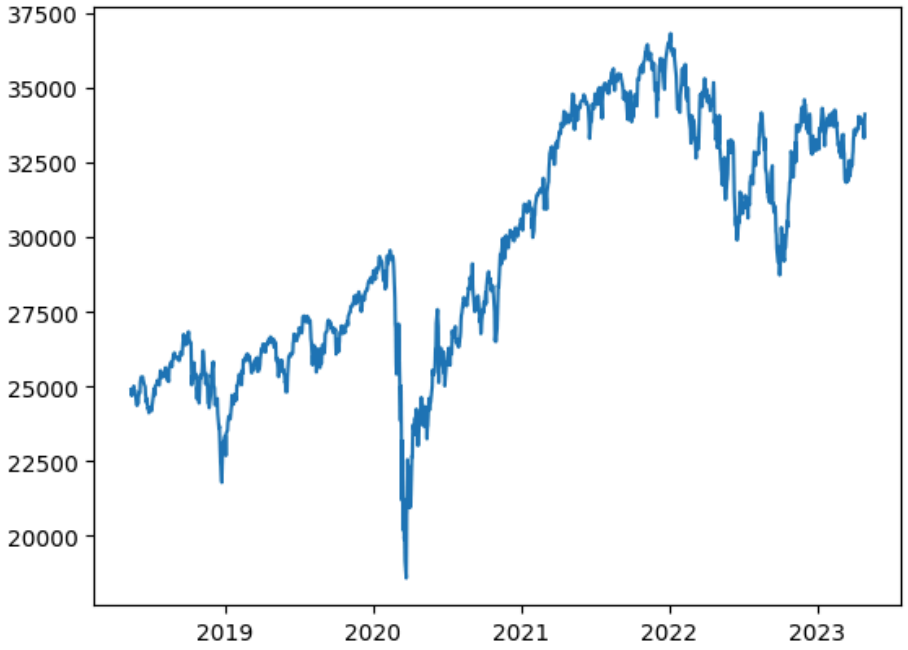
Overall, the GRU model proved to be the most balanced option with simple, efficient, and accurate performance for forecasting stock prices in volatile environments.

Introduction

Stock price prediction still remains a huge challenge due to its complex and non-linear nature with many factors influencing its trajectory. An accurate prediction is extremely advantageous for investors and institutions seeking to optimize their portfolios and risks. Traditional statistics are unable to capture the intricacies and patterns of historical data, hence the need for more advanced deep learning techniques such as Recurrent Neural Networks (RNNs), which were designed specifically for sequential time series data. (Bao et al., 2025)

The RNN model is necessary as it is able to capture the temporal dynamics of sequential data, allowing it to learn from past data and better forecast future trends. We aim that by leveraging the model's capabilities, we will be able to enhance stock price prediction accuracy beyond what other simpler models are able to achieve.

The dataset used in this project is the Yahoo Finance dataset which includes monthly historical stock market data from 2018 to 2023. The dataset includes features such as Open, Close, High, Low prices, and Volume, as well as Adjusted Close which accounts for splits and dividends. This particular dataset is suitable as it exactly shows the historical stock trends over a five year period.



Yahoo Finance Stock Close from 2018 to 2023 (Arora, 2023)

Report Body

Methods/AI techniques used

For all RNN model techniques, Google Colab is the main integrated Development Environment supported with cloud computing power to train the models effectively. Python libraries pandas, numpy and scikkit-learn are also installed into the environment to integrate functions and operations that may support the data preprocessing phase (McKinney, 2010; Harris et al., 2020; Pedregosa et al., 2011). The scikit-learn library is also used for preprocessing and evaluation of machine learning models (Pedregosa et al., 2011). Frameworks such as TensorFlow and Keras are used to import off-the-shelf models that can be fine-tuned according to the stock price data (Abadi et al., 2015). Finally, matplotlib library supports the interpretation of metrics and visualization of results for each RNN technique model (Hunter, 2007).

Our stock prediction model uses the RNN model from the tensorflow library. We decided on using this specific model due to its ability to predict future patterns by looking back from past patterns. The StandardScaler function (also the same as z-score normalization) from the sklearn library is also used as a technique of data normalization to tone down the volatility of the stock data since it is very unstable and this technique being very friendly towards outliers, which in stock graphs are really commonly found (Souza et al., 2023). There are 3 main techniques we are doing experiments on, simpleRNN, LSTM, and GRU.

The simpleRNN model involves 3 main layers in their neural network, the input layer, hidden layer, and the output layer. Nodes are connected to each other so the model may learn current information by calculating the output of the neural network through time *t* to the input of the network with time *t* + 1. The input data should be an array of vectors that changes over time, which are connected to the hidden layer. Hidden layers use hidden unit connections and small non-zero elements to improve the performance and stability of the network (Utevayeva et al., 2022).

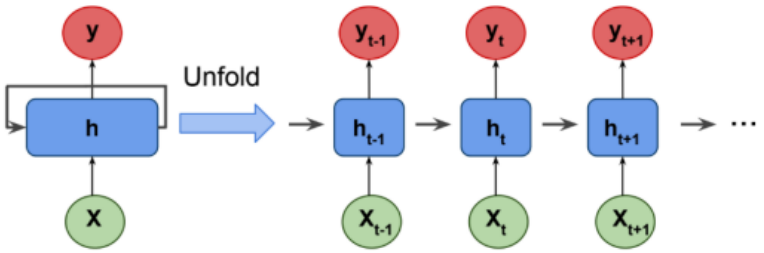


Illustration of simpleRNN structure (Utevayeva et al., 2022)

However, the simpleRNN network has drawbacks being gradient vanishing and explosion problems, which complicates the learning process to not be able to handle long sequences if tanh is the main activation function, and extremely unsteady when ReLU is the main activation function (Utevayeva et al., 2022). The vanishing gradient problem is where the gradient to change weights during RNN backpropagation reaches 0 in an unexpectedly short amount of time, not being able to update the weights any longer to search for the model's most optimized weights resulting in the best quality for predictions on unseen data (Kashyap, 2024). On the other hand, exploding gradients is the total opposite, which is the way gradients get too big during backpropagation resulting in unstable training and divergence when optimizing the model (Yogesh, 2024).

LSTM is another type of RNN that uses long-term memory cell units instead of standard layers to prevent long-term dependency. This architecture includes 4 layers, the cell state, input gate, output gate, and a forget gate. This model always combines the feature extracted data with the output of the previous cell, which then goes through the forget and input gate that has sigmoid activation functions to output values either 0 or 1. The forget gate decides which data to throw out of the cell, while the input gate keeps values from the input to update. The combined data also goes through the tanh activation function layer, then connected again to the old cell state to start an effective recurrence loop with the input data. The forget gate controls the recurrence loop reduce risk of gradient vanishing rather than explosion. The system then positions the cell state in the tanh function to output values between -1 and 1 to multiply again by the sigmoid gate's output, obtaining what should be the next value(s) from the array of vectors showing past data (Utevayeva, 2022).

$$f_t = \sigma(\omega_f[h_{t-1}, x_t] + b_f)$$

$$i_t = \sigma(\omega_i[h_{t-1}, x_t] + b_i)$$

$$\hat{C}_t = \tanh(\omega_c[h_{t-1}, x_t] + b_c)$$

$$C_t = f_t \times C_{t-1} + i_t \times \hat{C}_t$$

$$O_t = \sigma(\omega_O[h_{t-1}, x_t] + b_O)$$

$$h_t = O_t \times \tanh(C_t)$$

LSTM Mathematical Processes to obtain predictions (Utevayeva et al., 2022)

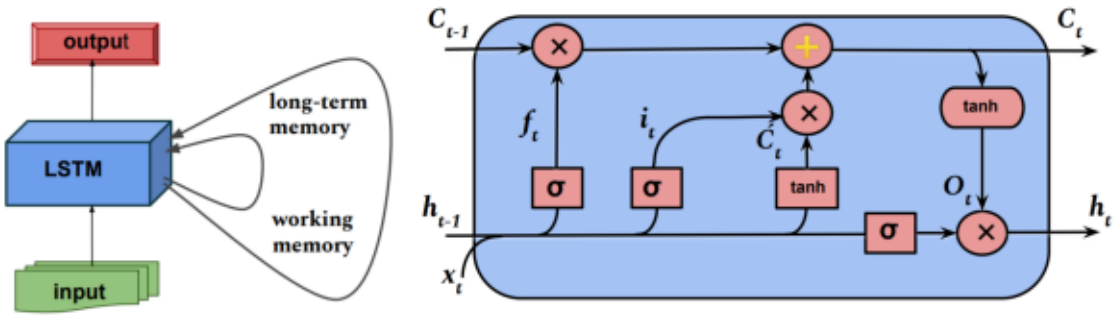


Illustration of LSTM structure (Utevayeva et al., 2022)

LSTM's main drawbacks are its memory requirements and computational complexity compared to simpleRNN because of the memory slots, differing from usual RNNs through long-term dependency and advantages on the overcoming on gradient vanishing and explosions, and long-term time dependency issues on the cell's memory (Utevayeva et al., 2022).

Compared to LSTM, the GRU network has less parameters, resulting in faster training while still maintaining high accuracy. GRU also does not have output gates. During the whole process, two main input functions are involved, which are vector h_{t-1} and input vector x_t. Each gate output is obtained through a logical operation and the input's nonlinear transformation. A reset gate determines how much of the previous hidden state should be ignored, filtering irrelevant past information. An update gate then decides how much past information gets carried forward to the current hidden state, preserving long-term dependencies. The combination of the filtered previous state and the current input produces a new hidden state that is used for predictions. Each hidden layer captures dependencies in different time frames, focusing on either short term or long term patterns, which provides a compact and efficient sequential data modelling (Utevayeva et al., 2022).

$$r_t = \sigma_g(\omega_r x_t + U_r h_{t-1} + b_r)$$

$$z_t = \sigma_g(\omega_z x_t + U_z h_{t-1} + b_z)$$

$$h_t = (1 - z_t)h_{t-1} + z_t \hat{h}_t$$

$$\hat{h}_t = \sigma_h(\omega_h x_t + U_h(r_t h_{t-1}) + b_h)$$

GRU Mathematical Processes to obtain predictions (Utevayeva et al., 2022)

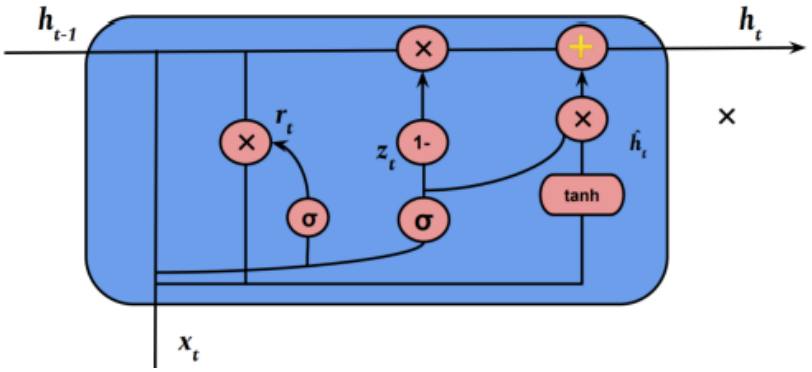


Illustration of GRU structure (Utevayeva et al., 2022)

Implementation of AI techniques

The dataset used for all of our experimental RNN techniques is the Yahoo Finance dataset containing stock price data ranging from 2018 to 2023, which was retrieved from the open-source dataset platform Kaggle (Arora, 2023). Our dataset included several features, including the date of stock prices recorded, open and close stock prices for that day, the highest and lowest cost for that day, as well as the volume of bought stocks for that day.

Date	Open	High	Low	Close*	Adj Close**	Volume
Apr 28, 2023	33797.43	34104.56	33728.40	34098.16	34098.16	354310000
Apr 27, 2023	33381.66	33859.75	33374.65	33826.16	33826.16	343240000
Apr 26, 2023	33596.34	33645.83	33235.85	33301.87	33301.87	321170000
Apr 25, 2023	33828.34	33875.49	33525.39	33530.83	33530.83	297880000
Apr 24, 2023	33805.04	33891.15	33726.09	33875.40	33875.40	252020000
...	...	...	...	...	...	...
May 07, 2018	24317.66	24479.45	24263.42	24357.32	24357.32	307670000
May 04, 2018	23865.22	24333.35	23778.87	24262.51	24262.51	329480000
May 03, 2018	23836.23	23996.15	23531.31	23930.15	23930.15	389240000
May 02, 2018	24097.63	24185.52	23886.30	23924.98	23924.98	385350000
May 01, 2018	24117.29	24117.29	23808.19	24099.05	24099.05	380070000

Sample of Stock Price Dataset Dataframe in Google Colab

Before the data is used, preprocessing is needed to ensure the data is ready for the model to learn patterns effectively. We decided to follow the closing stock price for each day, since starting and closing prices were different every day, we wanted to focus on how the stock ended rather than started since the closing price will not change after the time window finishes. We also removed every other feature to allocate the model's focus on our desired feature, before moving on with experimentation. Since the dataset is abnormal in pattern with the stock market being naturally unpredictable, we also standardized the dataset with the standardscaler() function to scale down the abnormal patterns in the dataset so that weights would not be disproportionately biased towards very high or low prices. We also reshaped the data into 3D arrays where the RNN models will learn from the shape being a list of previous records the model will look back to before creating a prediction.

Feature Removal

	Date	Close
0	2018-05-01	24099.05
1	2018-05-02	23924.98
2	2018-05-03	23930.15
3	2018-05-04	24262.51
4	2018-05-07	24357.32
...	...	...
1253	2023-04-24	33875.40
1254	2023-04-25	33530.83
1255	2023-04-26	33301.87
1256	2023-04-27	33826.16
1257	2023-04-28	34098.16

Standardization

	Date	Close
0	2018-05-01	-1.373061
1	2018-05-02	-1.416515
2	2018-05-03	-1.415224
3	2018-05-04	-1.332256
4	2018-05-07	-1.308588
...	...	...
1253	2023-04-24	1.067442
1254	2023-04-25	0.981425
1255	2023-04-26	0.924269
1256	2023-04-27	1.055150
1257	2023-04-28	1.123050

3D Array Reshape

```
[[[-1.37306099e+00]]

 [[-1.41651466e+00]]

 [[-1.41522406e+00]]

 [[-1.33225593e+00]]

 [[-1.30858820e+00]]

 [[-1.30786676e+00]]

 [[-1.26235112e+00]]

 [[-1.21317585e+00]]

 [[-1.19029946e+00]]

 [[-1.17326448e+00]]

 [[-1.22144371e+00]]
```

There are a few parameter values we can fine tune to discover the best settings for the model to have the most optimized performance and quality. For example, the look back parameter selects how many past entries the model will look at before creating the prediction for the next number in the pattern. So far we have only used look back 1 and we can change it in a way that the model will look from different data coming from a variative number of past days. The number of hidden units in recursive neural networks is also important to find balance of being able to generalize towards unseen data well while not overfitting during training. Different techniques also may provide higher accuracy and confidence with the tradeoff of higher complexity to be experimented, which includes SimpleRNN, LSTM, and GRU. Different ratios of training and testing data is also important to find the balance between how much the model should learn in order to perform well enough in unseen testing data. Our main metric to observe model performance is with the Root Mean Square Error (RMSE). The higher it is, the more it shows how the model's predictions deviates from the true data, so the main goal is to minimize it as low as possible (Hodson, 2022). Documentations of different RNN technique experimentations are shown in the tables below.

RNN Technique	**Random Seed**	**Train-Test Set Ratio**	**Batch Size**	**Epochs**	**Hidden Units**	**Look Back**	**RMSE Results**
SimpleRNN	42	60% Train 40% Validation & Testing	1	21	16	1	Train: 397.5177 Test: 350.6779
		60% Train 40% Validation & Testing	1		8	1	Train: 406.765 Test: 355.3625
		50% Train 50% Validation & Testing	1		16	3	Train: 376.5464 Test: 361.5408
		50% Train 50% Validation & Testing	1		8	3	Train: 376.6168 Test: 364.448
		60% Train 40% Validation & Testing	2		8	3	Train: 382.825 Test: 366.6946

After several experiments through 72 different combinations, the table above shows the top 5 results of best combinations of settings for the SimpleRNN technique outputting the best results sorted based on the least testing error. Having less testing error means the model has predictions accurately made on unseen data from the test set, meaning it would also perform well if it were to be deployed in real environments

with unseen data as well. The best configurations for the SimpleRNN model are 60% training with 40% validation and testing ratio, along with batch size = 1, through 21 epochs of iterations, with 16 hidden units and 1 look back value. The best results obtained were training with 397.5177 RMSE and testing with 350.6779 RMSE. Although the error on both training and testing is very high, it is reasonable with the nature of the stock price dataset being very abnormal. However, obtaining testing error that is lower than the training error shows how the model generalized well in learning patterns from the training set to create accurate predictions even on unseen data in the test set The consistency of these results were also guaranteed as a random seed of 42 is set for the hyperparameter search grid to ensure reproducibility of accurate prediction performance even in real situations.

RNN Technique	Random Seed	Train-Test Set Ratio	Batch Size	Epochs	Hidden Units	Look Back	RMSE Results
LSTM	42	70% Train 30% Validation & Testing	1	21	16	1	Train: 363.894193 Test: 380.331987
		80% Train 20% Validation & Testing	1		16	7	Train: 377.692903 Test: 382.177707
		80% Train 20% Validation & Testing	2		16	7	Train: 358.975149 Test: 382.498516
		70% Train 30% Validation & Testing	1		8	1	Train: 361.063063 Test: 384.111828
		80% Train 20% Validation & Testing	2		8	3	Train: 347.593028 Test: 384.538197

The hyperparameter search grid for simpleRNN is done similarly to the LSTM model. After experimenting between 72 different combinations of settings for the LSTM model, the table above shows the top 5 results of all combinations outputting the least error in training and testing. The best combination of configurations involves 70 % training to 30% validation and testing set ratio, batch size = 1, through 21 epochs of iterations, 16 hidden units in 1 layer, and 1 look back value. The result is 363.894193 RMSE in training error, and 380.331987 RMSE in testing error.

RNN Technique	Random Seed	Train-Test Set Ratio	Batch Size	Epochs	Hidden Units	Look Back	RMSE Results
GRU	42	60% Train 40% Validation & Testing	1	21	16	1	Train: 367.9856 Test: 343.8319
		60% Train 40% Validation & Testing	2		16	1	Train: 367.3431 Test: 356.9823
		60% Train 40% Validation & Testing	1		4	1	Train: 365.4521 Test: 365.2162
		70% Train 30% Validation & Testing	2		16	1	Train: 365.5222 Test: 372.5447

		60% Train 40% Validation & Testing	1		16	3	Train: 366.5758 Test: 374.7797

The above hyperparameter search grid for GRU is done similarly to the previous two models. After experimenting between 72 different combinations of settings for the GRU model, the table above shows the top 5 results of all combinations outputting the least error in training and testing. The best combination of configurations involves 60% training to 40% validation and testing set ratio, batch size = 1, through 21 epochs of iterations, 16 hidden units in 1 layer, and 1 look back value. The result is 367.9856 RMSE in training error, and 343.8319 RMSE in testing error.

Comparisons of different AI techniques

RNN Technique	**Random Seed**	**Train-Test Set Ratio**	**Batch Size**	**Epochs**	**Hidden Units**	**Look Back**	**RMSE Results**
GRU	42	60% Train 40% Validation & Testing	1	21	16	1	Train: 367.9856 Test: **343.8319**
SimpleRNN		60% Train 40% Validation & Testing					Train: 397.5177 Test: **350.6779**
LSTM		70% Train 30% Validation & Testing					Train: 363.894193 Test: **380.331987**

Comparing the best results from all 3 RNN techniques, the GRU model seems to have the best performance in having the least testing RMSE. With most optimized hyperparameter settings being the same in different models outputting the best results, it all comes down to which model's algorithm mechanics work to learn patterns through the volatile fluctuating stock prices best.

Compared to simpleRNN which is problematic in terms of vanishing or exploding gradient problems hindering in learning long-term patterns, and LSTM which requires higher computational resources and more parameters, the GRU technique offers simpler architecture with less gates and parameters used while also maintaining high prediction accuracy. The GRU technique's reset and update gates retain relevant past data and filter noise in parallel, making it the best option to handle highly volatile unpredictable stock price data. This also allows GRU to generalize better on unseen data in capturing short and long term fluctuation trends in the stock market, resulting in lower testing error and higher reliability for predictions in real market conditions. This is also reflected in the GRU technique's training result, showing the least amount of training and testing error compared to the 2 other techniques.

Conclusion

This project aimed to identify the most effective recurrent neural network (RNN) architecture for stock price prediction using historical Yahoo Finance data (2018–2023). Three models—**SimpleRNN**, **LSTM**, and **GRU**—were developed and tested under multiple hyperparameter combinations to evaluate their performance.

Among all models, the **GRU (Gated Recurrent Unit)** achieved the **best results**, with the lowest testing RMSE of **343.83**, indicating superior accuracy and generalization. Its design, which combines simplicity with strong sequential learning ability, allowed it to capture short-term fluctuations effectively while avoiding common RNN issues like vanishing or exploding gradients. Compared with LSTM, the GRU required fewer computational resources but still maintained high accuracy, making it both efficient and practical.

Another key finding was that increasing the look-back value—the number of previous data points the model considers—did not always enhance predictive accuracy. Surprisingly, a look-back of just one previous day produced strong results, suggesting that stock movements are heavily influenced by recent trends rather than long-term patterns. This reinforces the idea that financial time-series forecasting benefits from models that can adapt quickly to sudden market shifts rather than rely heavily on extended historical data.

Overall, the project demonstrates that the GRU model provides a balanced and efficient approach for time-series prediction tasks, particularly in volatile domains such as the stock market, where capturing short-term dependencies can be more meaningful than modeling long-term ones.

Recommendations

Although the GRU model performed best in this study, there are several opportunities to enhance both model accuracy and practical applicability.

First, future research should focus on advanced hyperparameter tuning. Adjusting learning rates (e.g., 0.001, 0.1), experimenting with different activation functions (such as ReLU, Leaky ReLU, or Sigmoid), and testing alternative optimization algorithms (like Adam or RMSProp) may reveal more optimal configurations.

Second, exploring hybrid or alternative deep-learning models could further improve performance. Integrating GRU with Convolutional Neural Networks (CNNs) to extract spatial features, or comparing results with Feedforward Neural Networks (FNNs), could help determine whether combining different architectures strengthens forecasting accuracy.

Third, for real-world deployment, the system could be improved by extending training epochs and including larger, more diverse datasets to enhance model robustness. Developing a user-friendly interface would also make the predictor accessible to investors or analysts by visualizing real-time predictions and performance metrics interactively.

Finally, to boost reliability in real trading scenarios, it is recommended to incorporate external market-driving factors such as political events, economic indicators, global news, or investor sentiment. These contextual variables could be integrated alongside price data to provide a more comprehensive and realistic prediction framework, enabling more informed decision-making in dynamic market environments.

List of References

Arora, S. (2023). *Yahoo Finance Dataset (2018-2023)*. Kaggle. https://www.kaggle.com/datasets/suruchiarora/yahoo-finance-dataset-2018-2023/data.

Bao, W., Cao, Y., Yang, Y., Che, H., Huang, J., & Wen, S. (2025). *Data-driven stock forecasting models based on neural networks: A review*. Information Fusion, 113, 102616. https://doi.org/10.1016/J.INFFUS.2024.102616

Hunter, J.D. (2007). *Matplotlib: A 2D Graphics Environment* (Online). IEEE Explore. https://ieeexplore.ieee.org/document/4160265/authors

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (2011). *Scikit-learn: Machine learning in Python* (Online). *Journal of Machine Learning Research, 12*(85), 2825–2830. https://jmlr.org/papers/v12/pedregosa11a.html

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, Ł., Kudlur, M., Levenberg, J., … Zheng, X. (2015). *TensorFlow: Large-scale machine learning on heterogeneous distributed systems* [White paper]. Arxiv. https://arxiv.org/pdf/1603.04467

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., ... & Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Souza, V.M.A., Lima, F.T. (2023). A *Large Comparison of Normalization Methods on Time Series* (Online). Science Direct. https://www.sciencedirect.com/science/article/pii/S2214579623000400

Hodson, T. (2022, July). *Root-mean-square error (RMSE) or mean absolute error (MAE): when to use them or not* (Online). Research Gate. https://www.researchgate.net/publication/362110985_Root-mean-square_error_RMSE_or_mean_absolute_error_MAE_when_to_use_them_or_not

Utabayeva, D., Ilipbayeva, L., Matson, E.T. (2022, December). *Practical Study of Recurrent Neural Networks for Efficient Real-Time Drone Sound Detection: A Review* (Online). Research Gate. https://www.researchgate.net/publication/366805579_Practical_Study_of_Recurrent_Neural_Networks_for_Efficient_Real-Time_Drone_Sound_Detection_A_Review

Kashyap, P. (2024, October 30). *Understanding the Vanishing Gradient Problem in Deep Learning* (Online). Medium. https://medium.com/@piyushkashyap045/understanding-the-vanishing-gradient-problem-in-deep-learning-840a9da6567f

Yogesh. (2024, April 25). *Understanding Vanishing and Exploding Gradients in Deep Learning* (Online). Medium. https://medium.com/@venkatyogesh003/understanding-vanishing-and-exploding-gradients-in-deep-learning-61a5f38d1605

Appendices

Full Hyperparameter Search Grid Results for SimpleRNN

Train Ratio	Batch Size	Neurons	Look Back	Train RMSE	Test RMSE
0.6	1	16	1	397.5177	350.6779
0.6	1	8	1	406.765	355.2625
0.5	1	16	3	376.5464	361.5408
0.5	1	8	3	376.6168	364.448
0.6	2	8	3	382.825	366.6946
0.6	2	16	3	371.8364	366.97
0.5	2	16	3	391.9699	369.8459
0.6	1	8	3	366.6425	370.6301
0.6	1	16	3	371.5118	371.135
0.6	2	8	1	425.9657	371.646
0.5	1	8	7	379.8785	373.9455
0.5	2	8	3	395.1276	374.7439
0.6	2	16	1	445.3905	376.4765
0.8	1	16	1	369.0951	377.6247
0.8	1	8	1	358.3717	379.4577
0.5	2	8	1	420.8639	379.9297
0.8	2	16	1	351.2631	381.7775
0.8	1	8	7	357.627	387.7705
0.8	2	8	1	350.1991	388.4855
0.6	1	16	7	381.0837	389.2248
0.8	2	16	3	382.4735	389.8185
0.8	1	4	1	353.113	390.3004
0.8	2	8	3	366.5424	390.4649
0.8	1	4	7	360.5225	393.8544
0.8	1	16	7	361.4625	393.869
0.8	1	8	3	361.724	393.989
0.5	2	8	7	382.6716	395.1183
0.8	1	16	3	357.3525	395.7058

0.7	2	8	3	351.0658	396.552
0.7	1	8	7	388.0283	397.5125
0.7	1	16	7	400.8694	399.5237
0.8	2	4	1	354.9426	402.4666
0.8	1	4	3	366.8778	403.4188
0.7	2	16	3	353.1412	404.594
0.6	2	16	7	399.0478	405.1678
0.5	2	4	7	407.6654	405.2366
0.5	1	4	7	388.179	406.1284
0.6	1	8	7	381.1138	406.1727
0.5	2	16	1	434.5522	410.1038
0.5	1	16	7	398.3365	414.2503
0.5	2	16	7	390.4837	414.4811
0.6	2	8	7	384.319	415.2327
0.7	1	8	3	357.4302	418
0.7	1	4	3	361.3831	423.2572
0.7	2	4	1	372.9182	424.059
0.7	1	16	3	360.3679	424.3179
0.7	1	4	7	363.424	424.7169
0.6	1	4	1	407.9853	426.2534
0.8	2	4	3	370.8066	428.7835
0.5	1	8	1	393.481	433.12
0.7	2	4	3	373.4173	436.7455
0.8	2	16	7	416.3869	440.2126
0.5	1	4	3	394.8903	442.1296
0.7	2	8	1	380.7152	445.6037
0.7	2	4	7	371.2466	447.0934
0.7	1	4	1	371.8789	451.1957
0.6	2	4	1	399.4937	453.2953
0.8	2	8	7	432.3614	454.7028
0.7	2	16	7	376.7313	460.9905

0.6	2	4	3	409.9626	462.3189
0.7	1	16	1	379.0273	462.8304
0.7	2	8	7	379.0029	463.8347
0.7	1	8	1	375.6666	463.8358
0.6	1	4	3	372.8418	464.1076
0.7	2	16	1	386.028	466.2087
0.5	2	4	3	438.641	467.2831
0.6	2	4	7	382.1411	476.4263
0.5	1	16	1	390.0829	477.4864
0.6	1	4	7	386.196	488.715
0.8	2	4	7	471.3927	513.6244
0.5	2	4	1	402.8616	623.8136
0.5	1	4	1	396.8196	695.4127

Full Hyperparameter Search Grid Results for LSTM

Train Ratio	Batch Size	Neurons	Look Back	Train RMSE	Test RMSE
0.7	1	16	1	363.8942	380.332
0.8	1	16	7	377.6929	382.1777
0.8	2	16	7	358.9751	382.4985
0.7	1	8	1	361.0631	384.1118
0.8	2	8	3	347.593	384.5382
0.8	1	8	3	350.899	385.5587
0.8	2	16	3	350.1726	385.9479
0.8	2	16	1	360.4591	387.2404
0.8	1	16	3	355.8334	387.6254
0.8	1	8	1	352.4127	388.9355
0.8	1	16	1	353.2155	390.6794
0.8	2	8	1	365.6295	391.9518
0.8	1	8	7	381.122	392.6453
0.8	1	4	3	365.052	395.5522
0.7	2	16	1	368.8327	395.8779
0.8	1	4	1	366.48	396.0726

0.8	1	4	7	389.0219	397.1069
0.8	2	4	1	380.8003	403.0313
0.7	2	8	3	354.1499	411.5637
0.7	2	8	1	369.4403	412.3916
0.8	2	4	7	372.3793	412.437
0.7	1	8	3	363.7059	414.8981
0.7	2	16	7	360.0421	416.4078
0.7	2	16	3	356.9753	416.4673
0.8	2	8	7	368.5094	416.7982
0.7	1	16	3	370.6891	418.1123
0.7	1	4	1	375.9118	419.4295
0.6	1	8	1	365.68	427.8963
0.7	1	16	7	343.0989	428.7189
0.6	1	16	1	366.0598	428.7549
0.7	1	4	3	373.7069	437.0255
0.8	2	4	3	378.0044	444.4349
0.7	2	4	1	385.1403	449.0939
0.7	1	8	7	352.2907	473.6701
0.7	1	4	7	349.9909	478.1305
0.7	2	4	3	384.1658	492.7571
0.6	1	16	3	358.1977	510.4814
0.7	2	4	7	366.5476	513.438
0.7	2	8	7	368.1427	514.1354
0.6	2	16	7	353.4922	525.8402
0.6	2	16	3	361.4342	526.4595
0.6	2	16	1	370.567	529.7767
0.6	1	8	3	357.2542	540.4809
0.6	2	8	3	358.0332	560.6154
0.6	1	16	7	349.3697	590.1386
0.6	2	8	1	384.4882	697.1036
0.6	1	8	7	356.0673	752.3279

0.6	1	4	7	362.1309	818.4404
0.6	1	4	1	390.4445	822.7413
0.6	1	4	3	370.4321	833.6976
0.6	2	8	7	367.006	860.4791
0.6	2	4	1	401.5503	972.0777
0.6	2	4	7	372.2668	1003.253
0.6	2	4	3	401.1754	1085.653
0.5	1	16	1	381.6402	2108.127
0.5	2	16	1	379.6005	2191.289
0.5	1	16	3	368.6652	2951.492
0.5	2	4	1	384.562	3010.855
0.5	2	16	3	377.6028	3010.907
0.5	1	4	1	383.5516	3017.31
0.5	2	16	7	382.4321	3565.665
0.5	1	16	7	394.4411	3567.625
0.5	1	8	1	385.4351	3567.725
0.5	2	8	1	386.6823	3663.783
0.5	1	4	3	380.6855	3885.567
0.5	1	8	3	367.854	3979.788
0.5	2	4	3	403.1894	4028.44
0.5	2	8	3	378.8862	4286.483
0.5	1	4	7	399.7993	4339.348
0.5	2	4	7	388.0875	4548.341
0.5	1	8	7	397.113	4626.62
0.5	2	8	7	389.3044	4697.65

GRU Hyperparameter tuning table:

Train Ratio	Batch Size	Epochs	Neurons	Look Back	Train RMSE	Test RMSE
0.6	1	21	16	1	367.9856148	343.8318739
0.6	2	21	16	1	367.3430847	356.9822507
0.6	1	21	4	1	365.452117	365.2161591
0.7	2	21	16	1	365.5222143	372.5447472

0.6	1	21	16	3	366.5757539	374.7796785
0.6	2	21	16	3	382.5618352	377.1554563
0.7	1	21	4	1	360.941348	377.5020245
0.6	1	21	16	7	420.141918	377.5238027
0.7	2	21	4	1	365.022225	378.1408645
0.5	1	21	16	3	431.8640807	378.9070592
0.7	1	21	8	1	361.9264048	379.1841729
0.8	2	21	8	1	355.6261464	379.45955
0.8	2	21	4	1	355.4344412	379.7734297
0.8	2	21	16	1	350.2597449	380.3289316
0.7	2	21	8	1	365.5888612	380.5445352
0.6	1	21	8	3	363.9643346	382.6733579
0.6	1	21	8	7	396.7258653	384.2392984
0.6	2	21	4	1	367.8460304	385.238914
0.8	1	21	16	1	351.4977791	385.6968085
0.6	1	21	8	1	365.8550256	386.5307992
0.8	1	21	8	1	350.2922481	386.607201
0.8	1	21	4	1	350.6137533	386.9053763
0.7	1	21	16	1	366.8228039	387.6511887
0.8	1	21	8	3	369.882745	389.8566015
0.8	1	21	4	7	382.617922	391.3972274
0.8	2	21	8	3	356.1786005	392.2026078
0.8	1	21	8	7	376.584202	394.6018154
0.7	1	21	16	3	376.7521859	394.9812338
0.6	2	21	16	7	372.8461944	395.3533474
0.8	2	21	16	3	360.6551706	395.9063954
0.8	1	21	4	3	365.7006937	396.1293312
0.7	1	21	8	3	377.2072312	396.2578328
0.8	1	21	16	3	373.7872657	397.5743556
0.8	2	21	4	7	408.018276	401.1261831
0.8	1	21	16	7	380.3102234	403.1430291

0.7	1	21	8	7	386.3345261	404.7173587
0.7	2	21	16	7	360.4130054	405.2666148
0.7	2	21	8	7	357.2569572	406.5061896
0.7	2	21	8	3	367.4080943	407.0269495
0.8	2	21	4	3	368.4166346	407.7465396
0.7	1	21	4	7	367.3770475	408.6886678
0.7	2	21	16	3	379.6747107	411.0102548
0.8	2	21	8	7	419.3586802	413.7849506
0.7	1	21	4	3	379.488631	416.6253027
0.6	2	21	8	3	376.5660723	417.3031001
0.7	1	21	16	7	434.4564913	418.1398572
0.7	2	21	4	7	366.2418516	427.9456224
0.6	2	21	8	7	370.741649	428.4281628
0.5	1	21	16	1	384.2526664	432.8166054
0.6	2	21	8	1	369.1966692	433.3950782
0.6	1	21	4	7	378.7413728	436.1393661
0.7	2	21	4	3	378.9654433	436.7412958
0.8	2	21	16	7	452.2543682	454.7519301
0.5	2	21	16	3	392.4934932	471.917474
0.5	2	21	16	1	388.6361012	478.5895474
0.5	1	21	8	3	396.3895377	529.1718566
0.6	2	21	4	7	384.6156862	532.2475573
0.5	1	21	4	1	383.8003573	534.8710139
0.6	1	21	4	3	375.5041653	547.6141358
0.5	2	21	4	1	386.2347814	602.1187863
0.6	2	21	4	3	395.4939367	678.6176204
0.5	2	21	8	3	379.4981447	692.7606856
0.5	1	21	16	7	428.2246127	760.5403735
0.5	2	21	16	7	418.9164595	771.5962499
0.5	1	21	8	1	383.1793364	836.0759455
0.5	2	21	8	1	387.4248513	903.9189451

0.5	1	21	8	7	421.840414	1020.988604
0.5	2	21	8	7	420.4617145	1163.468637
0.5	1	21	4	7	419.3137008	1422.414532
0.5	1	21	4	3	386.5349596	1586.03656
0.5	2	21	4	7	428.7968503	1754.720204
0.5	2	21	4	3	407.6448896	1828.845191