TradeForecast:
A Multi-Horizon Stock Price Forecasting Framework

CS-677 Deep Learning
Section - 005, Fall 2024
New Jersey Institute of Technology

GitHub Repository

Vinay Ram Gazula

March 2, 2025

Contents

1 Abstract

Time series forecasting plays a pivotal role in various domains, particularly in financial markets where accurate stock price prediction is critical for informed decision-making. This project introduces TradeForecast, a production-ready framework for multi-horizon stock price forecasting using advanced deep learning models. The framework allows users to incorporate domain-specific features, including technical indicators and temporal attributes, to enhance model performance. Three distinct neural network architectures were developed: `LSTM`, `ConvLSTM`, and `EncTransformer`, each designed to capture unique dependencies in the data. A learning rate scheduler, ReduceLROnPlateau, was employed to ensure effective convergence during training. The modular nature of the codebase enables seamless extension and customization, empowering users to experiment with various features and architectures for optimal forecasting performance. This project demonstrates a robust, flexible approach to stock price prediction, providing a valuable tool for professional traders and researchers. A demonstration of the framework is executed using Alphabet Inc. (`GOOG`) stocks' historical data, showcasing its practical utility. Grid search is implemented for hyper-parameter tuning to achieve optimal model configurations. Finally, the performance metrics of the three model architectures are compared to identify the best-performing model.

2 Introduction

Time series forecasting presents significant challenges due to long-term dependencies that forecast models must effectively capture to produce accurate predictions. Existing research highlights the application of advanced deep learning algorithms to forecast diverse time series data across sectors such as supply chain management, finance, and healthcare analytics.

This project focuses on developing a production-ready framework for multi-horizon forecasting of individual stock prices using deep learning algorithms. The framework is tailored to assist professional traders in making informed trading decisions and devising innovative strategies. Recognizing the critical role of technical indicators in analyzing a stock's price action, this framework enables users to seamlessly incorporate such features into their analyses, providing a powerful tool for predictive modeling and decision-making in financial markets.

The aim of this project, TradeForecast, is to utilize deep learning models for predicting the closing prices of stocks using a multi-horizon time series prediction approach. This approach provides insights into trends over several future time points, aiding investors and traders in making informed decisions. A modular codebase for feature engineering has been developed, enabling users to incorporate new features, such as technical indicators, into historical stock data based on their expertise. This framework allows for the seamless customization of features to enhance the dataset for analysis. Additionally, three distinct neural network architectures `LSTM`, `ConvLSTM`, and `EncTransformer` have been implemented, providing users with the flexibility to experiment with different models and select the one that performs best for the specific stock data and forecasting task at hand.

2.1 Related Works

Wen et al. [1] proposed a framework for general probabilistic multi-step time series regression using Sequence-to-Sequence neural networks like Recurrent or Convolution neural networks, Quantile Regression and direct multi-horizon forecasting. Lim et al. [2] introduced Temporal Fusion Transformer (TFT) a novel attention-based architecture that combines high-performance multi-horizon forecasting with interpretable insights into temporal dynamics. TFT uses recurrent layers for local processing and interpretable self-attention layers for long-term dependencies. Eisenach et al. [3] developed a novel decoder-encoder attention for context-alignment, improving forecasting accuracy by allowing the network to study its own history based on the context for which it is producing a forecast.

3 Methodology

3.1 Multi-Horizon Forecasting

Multi-horizon forecasting is the process of predicting values for multiple future time steps simultaneously. Unlike single-step-ahead forecasting, which focuses on predicting only the next time step, multi-horizon forecasting offers a more comprehensive view by providing insights over a range of future periods. This capability is particularly valuable in applications such as stock price forecasting, where understanding trends across multiple horizons is critical for decision-making.

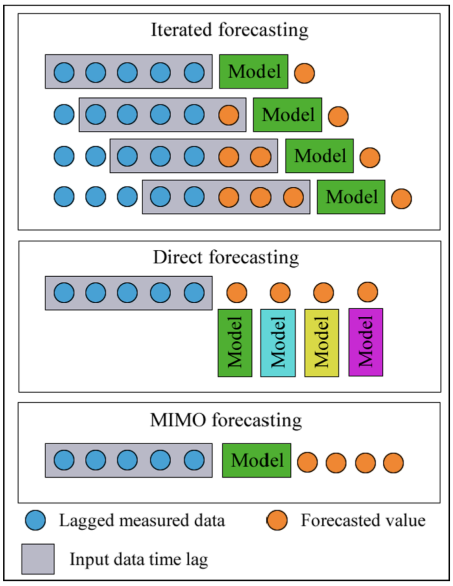


Figure 1: Multi-Horizon forecasting approaches [4]

Approaches to Multi-Horizon Forecasting (illustrated in figure 1):

1. **Recursive/Iterated Approach:** A single model predicts the next time step, and the prediction is fed back into the model as input to forecast subsequent steps. This approach is simpler but can suffer from error propagation, where inaccuracies in early predictions amplify over longer horizons.

2. **Direct Approach:** Separate models are trained for each time horizon (e.g., one model for predicting day 1, another for day 2, etc.). While this approach can provide accurate results for specific horizons, it is computationally expensive and lacks consistency across predictions.

3. **Multi-Output/MIMO Approach:** A single model generates predictions for all desired future time steps simultaneously. This method is efficient and ensures consistency across the horizons by capturing dependencies within the predicted sequence.

This project adopts the multi-output approach, leveraging its ability to efficiently capture dependencies across multiple horizons and streamline the forecasting process.

Look-Back and Forecast Periods

- The look-back period defines the historical data window used as input for forecasting. For example, a look-back period of 60 days uses the past 60 days of data to make predictions.

- The forecast period specifies the number of future time steps to predict. For instance, forecasting the next 5 days involves generating predictions for 5 consecutive days ahead.

For example, consider a look-back period of 60 days and a forecast period of 5 days. The input data matrix will include stock prices and engineered features from the past 60 days, while the target matrix will include closing prices for the subsequent 5 days. This structure ensures the model is trained to map a fixed-length historical window to a multi-step future forecast. Both look-back period and forecast period can be considered as hyper-parameters and can be tuned to achieve better performance.

3.2 Data and Feature Engineering

3.2.1 Raw Data

The historical stock price data for this project is obtained using the yfinance Python package. Each stock has a unique identifier commonly known as a "Ticker", for example the Ticker for Amazon.com Inc is `AMZN`. The user must input the Ticker for the stock they are interested to fetch the historic data. This data includes the standard OHLCV (Open, High, Low, Close, Volume) variables, which form the foundational features for our models. These variables provide insights into daily market movements and form the baseline for predicting future trends.

Feature engineering is a critical step in improving the predictive performance of the models. For this project, we have designed a flexible framework that allows users to extend and customize the feature set based on their expertise.

3.2.2 Temporal Features

Temporal information, such as week of the year, quarter of the year, and hour of the day, provides critical insights into periodic trends and seasonal patterns inherent in time series data. These features are especially valuable in financial forecasting, where stock prices often exhibit predictable behaviors during specific time frames, such as increased trading activity during certain hours of the day or seasonal fluctuations tied to quarterly earnings reports.

To ensure these temporal features are effectively utilized by the model, they are encoded using **Cyclical Encoding**[1] which applies sine and cosine transformations on temporal features. This approach retains the inherent cyclical nature of these variables (e.g., hour 0 and hour 23 are close to each other in time) and allows the model to learn smooth periodic patterns.

By extracting and encoding these temporal features on demand, the framework provides users with the flexibility to incorporate relevant periodic trends into their forecasting models, enhancing their ability to predict stock price movements more accurately.

3.2.3 Technical Indicators

- **Moving Average (MA):** Smooths out price fluctuations to highlight the underlying trend over a defined period. Commonly used by traders to identify potential buying or selling signals.
- **MACD (Moving Average Convergence Divergence):** A momentum indicator that tracks the relationship between two moving averages of a stock's price. Helps identify trend strength and potential reversals.
- **RSI (Relative Strength Index):** Measures the speed and change of price movements to determine overbought or oversold conditions. Aids in assessing market momentum.
- **ATR (Average True Range):** Evaluates market volatility by analyzing recent price ranges. Useful for setting stop-loss levels and identifying breakouts.

These indicators enhance the dataset by providing deeper insights into price movements, enabling the models to learn complex market dynamics.

The feature engineering codebase is designed with modularity and flexibility in mind. Users can:

[1] https://developer.nvidia.com/blog/three-approaches-to-encoding-time-information-as-features-for-ml-models/

- Add new technical indicators: The framework supports the integration of custom indicators, allowing users to experiment with advanced metrics tailored to their specific requirements.
- Incorporate new feature engineering techniques: The modular design facilitates seamless addition of novel methods for enriching the dataset.

This extensible framework empowers users to customize their analysis pipeline and adapt the system to evolving market conditions or specific forecasting needs.

3.3 Forecast Models

The project implements three deep learning architectures optimized for multi-horizon time series forecasting: `LSTM`, `ConvLSTM`, and `EncTransformer`. Each model is designed to leverage different techniques for capturing temporal patterns in stock price data, offering flexibility for various forecasting tasks. Figure 2 gives an overview of the model architectures for these three models.

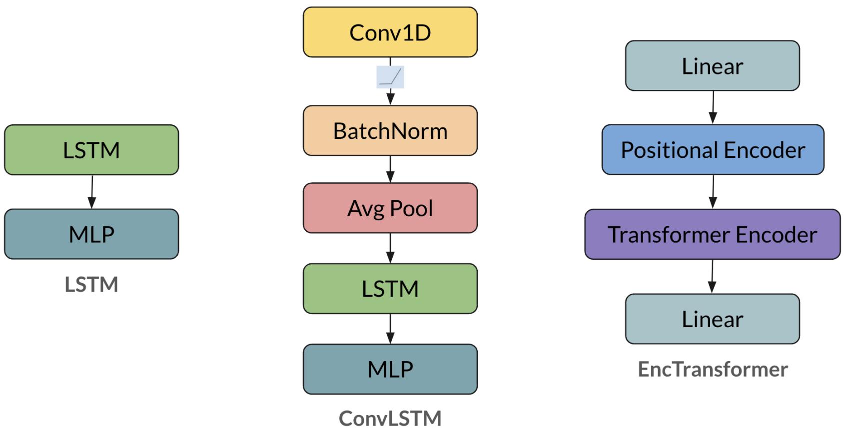


Figure 2: Model Architectures

3.3.1 LSTM Model

This model architecture is based on LSTM (Long Short-Term Memory) [5] which is a Recurrent Neural Network (RNN) based architecture with various gates specifically designed to handle sequential data by learning long-term dependencies, making it an ideal choice for time series forecasting.

1. LSTM Layer:
   - Processes sequential input data (look-back period) to extract temporal dependencies.
   - Maintains a memory cell, updated through input, forget, and output gates, allowing the model to selectively remember or forget information over time.
   - Outputs a fixed-length feature vector summarizing the input sequence.
2. MLP (Multi-Layer Perceptron):
   - A fully connected feed-forward network processes the LSTM output.
   - Maps the sequence representation to the forecast horizon (number of steps in the forecast period).

LSTMs effectively capture long-term dependencies in stock price data. By pairing LSTM with an MLP, the architecture achieves both sequence encoding and regression for multi-horizon forecasting. The hyper-parameters for this model include: number of LSTM layers, unidirectional/bidirectional LSTM, hidden size, dropout between LSTM layers, number of Linear/Dense layers in MLP, input/output dimensions of intermediate MLP layers. The source code for this model architecture can be found in ./tradeforecast/forecast/lstm.py file in the GitHub repository.

3.3.2 ConvLSTM Model

The ConvLSTM architecture combines 1D Convolutional Neural Networks (1D-CNN) [6] with LSTM [5], integrating both spatial and temporal feature extraction. This hybrid approach captures short-term local patterns and long-term dependencies.

1. Conv1D Layer:
    - Applies a 1-dimensional convolution to the input data, sliding a kernel over the sequence to extract short-term patterns (e.g., local trends in stock prices).
    - Captures spatial dependencies across feature dimensions. Each feature is considered as an input channel for this layer.
    - A ReLU activation function is added after this layer to introduce non-linearity and enhance the models ability to learn complex patterns.
2. Batch Normalization:
    - Normalizes the feature maps produced by the Conv1D layer, ensuring stable gradient flow and faster convergence during training.
3. Average Pooling:
    - Down-samples the feature maps by taking the average over specified regions, reducing dimensionality and focusing on prominent features.
4. LSTM Layer:
    - Processes the temporally reduced feature maps from the convolutional block, capturing long-term dependencies.
    - Outputs a fixed-length feature vector summarizing the input sequence.
5. MLP (Multi-Layer Perceptron):
    - Transforms the LSTM's sequential output into forecasts for the specified horizon.

Conv1D layers enhance feature extraction by identifying local dependencies. By feeding these features into the LSTM, the architecture captures a combination of short-term patterns and long-term temporal trends. The hyper-parameters for this model include kernel size for convolution, number of output channels after convolution, and the ones discussed in the `LSTM` Model. The source code for this model architecture can be found in ./tradeforecast/forecast/convlstm.py file in the GitHub repository.

3.3.3 EncTransformer Model

The EncTransformer architecture is built on the Transformer encoder [7], which leverages self-attention mechanisms to model complex relationships across input sequences.

1. Linear:
    - Projects input features into a higher-dimensional embedding space to enhance feature representation for the transformer.
2. Positional Encoder:
    - Adds positional information to the input embeddings using sine and cosine transformations, allowing the model to understand the temporal order of the sequence.
3. Transformer Encoder:
    - Composed of multiple self-attention layers and feed-forward sub-layers.

- Self-Attention Mechanism: Utilizes multi-head attention to calculate weighted interactions between all time steps, enabling the model to identify global dependencies in the input sequence. Multi-head attention allows the model to focus on different parts of the sequence simultaneously, enhancing its ability to capture intricate patterns.
- Feed-forward Sub-layers: Refine the learned representations, ensuring effective extraction of complex patterns.

4. Linear:
   - Maps the transformer's output embeddings to the forecast horizon.

The self-attention mechanism, enhanced by multi-head attention, allows the model to capture both short-term and long-term dependencies globally, overcoming the limitations of recurrence-based models. The positional encoding ensures that temporal information is preserved despite the lack of inherent sequential structure in the transformer. Hyper-parameters for this model include embedding dimension, number of attention heads, number of transformer encoder layers, dropout. The source code for this model architecture can be found in ./tradeforecast/forecast/enctransformer.py file in the GitHub repository.

3.4 Training Strategy

A learning rate scheduler called **ReduceLROnPlateau**[2] was utilized to train these models. This scheduler dynamically adjusts the learning rate during training based on the training loss. Specifically: When the training loss stops improving for a defined number of epochs (referred to as the "patience" parameter), the learning rate is reduced by a predefined factor.

Advantages:

- Helps the model converge effectively by lowering the learning rate when progress stalls.
- Prevents overshooting the optimal point in the loss landscape.
- Improves training stability, especially in scenarios where the loss plateau occurs before reaching the global minimum.

This approach ensures that the training process remains efficient and avoids unnecessary oscillations in the learning trajectory. The default initial learning rate used in training is 0.1 and the minimum limit is set to 0.00001 meaning that the scheduler can not reduce the learning rate if the minimum has reached.

4 Results

We have considered Alphabet Inc. (parent company of Google) as our target stock to perform analysis and demonstrate the TradeForecast framework. The Ticker for this particular stock is `GOOG`. For training the models historical stock price data from dates *2015-01-01—2024-12-06* is used. For all model versions the look-back period is set to 60 days and the forecast period is set to 5 days meaning that the models forecast next 5 days closing price based on previous 60 days feature values.

4.1 Hyper-Parameter Tuning

All three models have various hyper-parameters and should be fine-tuned to extract the best results from the models. These hyper-parameters vary in each model and to tune them a grid search analysis is performed which is a brute force technique that uses different combinations of parameters to train the models and see how they perform.

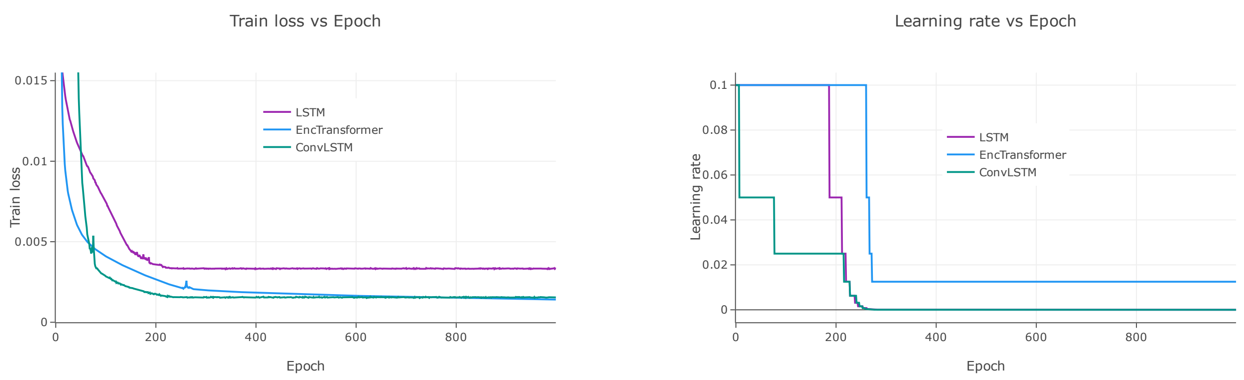
[2] https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.ReduceLROnPlateau.html

To verify that the feature engineering is helping the models to learn more about the stock price and have better forecasting results this feature engineering step is incorporated as a tuning parameter meaning that all models are trained using the raw data which only consists OHLCV as features (indicated by `data_version` = "`base`" in grid search results) and the feature engineered data which consists a combination of technical indicators and temporal features (indicated by `data_version` = "`feat_engg`" in grid search results). Which means that all combinations of models are trained using both raw data and feature engineered data.

Since this is a brute force technique and some variants of models' have more than one million parameters the run times can increase rapidly. Considering this, the number of epochs for all models in grid search analysis is limited to 500 and only important parameters are altered to reduce the number of possible combinations. The source code for the grid search analysis can be found in ./grid_search.py file in the GitHub repository. The results for the grid search analysis are mentioned in Appendix B.

4.2 Training

The best parameters for each model architecture are selected based on the grid search results. Using these parameters and the feature engineered data, each model is now trained for 1000 epochs and the training phase is logged to visualize how the models are learning. The source code for this training can be found in the ./notebooks/training.ipynb notebook in the GitHub repository.



(a) (b)

Figure 3: (a) Training loss and (b) Learning rate w.r.t Epoch.

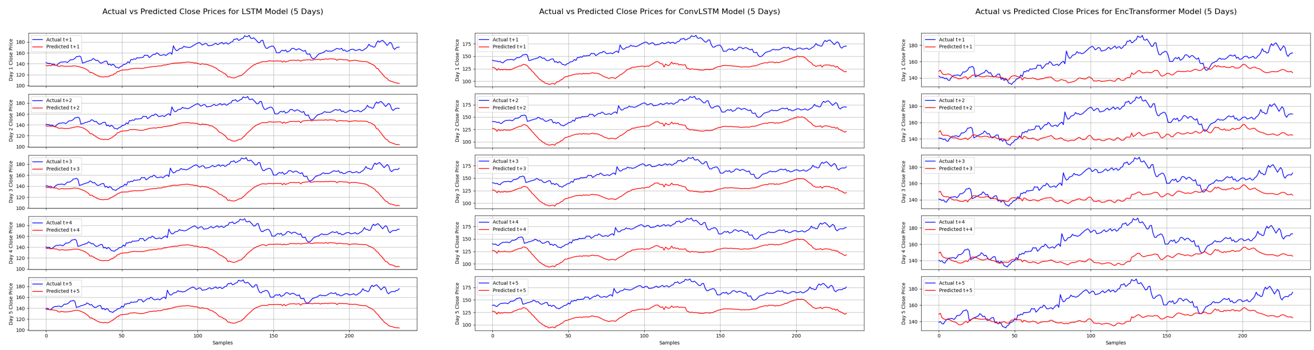
Figure 3a shows the decay of training loss w.r.t epochs for all three models and figure 3b shows the learning rate decay (altered by the learning rate scheduler) w.r.t epochs. We can see that `ConvLSTM` achieves faster convergence compared to other models which adds to the fact that we have implemented Batch Normalization in that model architecture. `LSTM` model performs poorly compared to the other two models as it stops to learn after 200 epochs even though the learning rate rate is decayed to 0.00001. `EncTransformer` seems to learn more as the training loss is still decreasing over each epoch and its' learning rate is around 0.0175 meaning that it can be trained even longer.

Model	Train loss	Test loss	MAE	MSE	RMSE	R-squared
LSTM	0.00332	0.20634	0.37785	0.20377	0.45140	-4.41385
ConvLSTM	0.00154	**0.07192**	0.50164	0.28388	0.53280	-6.54086
EncTransformer	**0.00141**	0.10145	**0.26268**	**0.10176**	**0.31900**	**-1.70395**

Table 1: Performance metrics calculated using test dataset.

Various performance metrics like MAE, MSE, RMSE and R-squared are calculated using the test dataset for these models and the results are tabulated. Table 1 displays the performance metrics for all models, we can see that the best test loss is achieved by `ConvLSTM` model and `EncTransformer` achieves the best scores for all other metrics i.e. train loss, MAE, MSE, RMSE and R-squared.

Although it looks like both `LSTM` and `EncTransformer` models outperformed `ConvLSTM` model in MAE, MSE, RMSE and R-squared metrics when we stack all the predicted and actual values for the test dataset and visualize them based on different time horizons (example t+1 time step or the next day predictions are compared with actual next day closing price and similarly t+n time horizon predicted values are compared with t+n actutal values), `ConvLSTM` seems to capture the trends much better that `LSTM` and marginally better that `EncTransformer`. This can be seen in figure 4 where the red line indicates predicted values and blue line indicates actual values.



(a) (b) (c)

Figure 4: (a) `LSTM`, (b) `ConvLSTM`, and (c) `EncTransformer` actual vs predicted across different time-horizon's.

5 Conclusion

This report introduces TradeForecast, a robust and scalable framework for multi-horizon stock price forecasting. Through the use of deep learning architectures—`LSTM`, `ConvLSTM`, and `EncTransformer`—the framework is well-equipped to model complex temporal dependencies and dynamic patterns inherent in stock price data. The modular feature engineering pipeline further enhances the flexibility of the framework, enabling users to integrate domain-specific indicators and temporal attributes to tailor the forecasting models to their specific needs.

The frameworks design emphasizes extensibility, allowing for the seamless incorporation of new features, technical indicators, or additional datasets. This ensures that the system remains adaptable to evolving requirements in financial forecasting. Furthermore, the integration of a robust training strategy, including dynamic learning rate adjustments, underscores the focus on achieving high-performance and stable model convergence.

TradeForecast not only serves as a valuable tool for professional traders seeking data-driven insights but also provides a versatile platform for researchers exploring advancements in time series forecasting. Future enhancements may include automated data pipelines, deployment-ready CI/CD workflows, and the exploration of hybrid modeling approaches to further expand the frameworks capabilities.

6 Future Works

- `EncTransformer` model can be trained for more epochs to enhance its' performance.
- `ConvLSTM` performs well on training data but not so well on test data (although it can capture trends better that other two models) indicating that the model is over-fitting. This can be optimized by reducing model complexity or by introducing regularization.
- The trained models are stored in ./models directory in the GitHub repository, an inference pipeline should be built which uses the trained models to forecast real-time stock prices.

- Build and streamline the ETL pipeline and containerize the framework using Docker so that this framework can be incorporated as a backbend for a web application.
- Use of 1D CNN layers in `ConvLSTM` model has helped the model in capturing trends better, we can incorporate this 1D CNN layer in `EncTransformer` before positional encoding to extract local trends. This might boost the performance significantly.

References

[1] R. Wen, K. Torkkola, B. Narayanaswamy, and D. Madeka, "A multi-horizon quantile recurrent forecaster. arxiv 2017," *arXiv preprint arXiv:1711.11053*, 2017.

[2] B. Lim, S. Ö. Arık, N. Loeff, and T. Pfister, "Temporal fusion transformers for interpretable multi-horizon time series forecasting," *International Journal of Forecasting*, vol. 37, no. 4, pp. 1748–1764, 2021.

[3] C. Eisenach, Y. Patel, and D. Madeka, "Mqtransformer: Multi-horizon forecasts with context dependent and feedback-aware attention," *arXiv preprint arXiv:2009.14799*, 2020.

[4] L. B. Ferreira and F. F. da Cunha, "Multi-step ahead forecasting of daily reference evapotranspiration using deep learning," *Computers and electronics in agriculture*, vol. 178, p. 105728, 2020.

[5] S. Hochreiter, "Long short-term memory," *Neural Computation MIT-Press*, 1997.

[6] S. Kiranyaz, O. Avci, O. Abdeljaber, T. Ince, M. Gabbouj, and D. J. Inman, "1d convolutional neural networks and applications: A survey," *Mechanical systems and signal processing*, vol. 151, p. 107398, 2021.

[7] A. Vaswani, "Attention is all you need," *Advances in Neural Information Processing Systems*, 2017.

Appendices

A Python packages

Package	Usage
yfinance	Scraping stock prices
pandas	Data manipulation and results formatting
polars	Feature engineering
sklearn	Preprocessing and performance metrics calculation
pytorch	Building deep learning models
lightning	PyTorch Lightning for training deep learning models
tensorboard	Visualization toolkit for deep learning models

Table 2: Python packages used in TradeForecast

B Grid Search Results

The grid search analysis results for `LSTM`, `ConvLSTM`, and `EncTransformer` models are listed below in tables 3, 4 and 5 respectively. The models are sorted based on `train_loss` in an ascending order meaning that the best performing models are listed on the top and vice versa. Since the space is limited the font size is bumped down to fit all the information. However the `.csv` files for these grid search results can be found in ./results/grid_search diectory in the GitHub repository. We can see that all three models perform better when they are trained using the feature engineered data compared to models trained on raw data.

model	data_version	n_feat	hidden_size	n_LSTM	dropout	criterion	init_lr	final_lr	train_loss	test_loss	n_params	MAE	MSE	RMSE	R-squared
0	feat_engg	15	64	2	0.0	mse_loss	0.1	0.1	0.00248	0.35312	54341	0.52575	0.34173	0.58458	-8.07962
1	feat_engg	15	32	2	0.0	mse_loss	0.1	0.05	0.0029	0.20603	14885	0.37951	0.20281	0.45034	-4.38826
2	feat_engg	15	32	2	0.05	mse_loss	0.1	1e-05	0.00333	0.2076	14885	0.38075	0.20561	0.45344	-4.46257
3	feat_engg	15	64	2	0.05	mse_loss	0.1	1e-05	0.00364	0.23829	54341	0.42369	0.23525	0.48502	-5.25031
4	base	5	64	2	0.05	mse_loss	0.1	1e-05	0.01543	0.1761	51781	0.3791	0.17515	0.41851	-3.54988
5	base	5	32	2	0.05	mse_loss	0.1	1e-05	0.01557	0.18942	13605	0.39337	0.18617	0.43148	-3.83658
6	base	5	32	2	0.0	mse_loss	0.1	0.1	0.01576	0.168	13605	0.3648	0.16625	0.40773	-3.31887
7	base	5	64	2	0.0	mse_loss	0.1	0.1	0.01583	0.17554	51781	0.37595	0.17357	0.41661	-3.50888
8	feat_engg	15	32	2	0.0	l1_loss	0.1	0.00625	0.02409	0.39636	14885	0.39758	0.19918	0.4463	-4.29171
9	feat_engg	15	64	2	0.0	l1_loss	0.1	0.00625	0.03066	0.53218	54341	0.52138	0.3334	0.57741	-7.85845
10	feat_engg	15	64	2	0.05	l1_loss	0.1	1e-05	0.0373	0.45742	54341	0.45444	0.26281	0.51265	-5.98271
11	feat_engg	15	32	2	0.05	l1_loss	0.1	1e-05	0.03888	0.35054	14885	0.34982	0.16116	0.40144	-3.28151
12	base	5	64	2	0.0	l1_loss	0.1	1e-05	0.08551	0.33248	51781	0.32819	0.13963	0.37367	-2.62647
13	base	5	32	2	0.0	l1_loss	0.1	1e-05	0.08566	0.35636	13605	0.35093	0.15719	0.39647	-3.08311
14	base	5	32	2	0.05	l1_loss	0.1	1e-05	0.08793	0.33235	13605	0.32783	0.13851	0.37217	-2.59766
15	base	5	64	2	0.05	l1_loss	0.1	1e-05	0.08841	0.3244	51781	0.32147	0.13465	0.36695	-2.49739

Table 3: `LSTM` grid search results.

model	data_version	n_feat	conv_out_size	kernel_size	hidden_size	n_LSTM	dropout	criterion	init_lr	final_lr	train_loss	test_loss	n_params	MAE	MSE	RMSE	R-squared
0	feat_engg	15	30	10	64	2	0.0	mse_loss	0.1	0.025	0.00073	0.72569	62771	0.49773	0.29376	0.542	-6.80461
1	feat_engg	15	30	5	64	2	0.0	mse_loss	0.1	0.025	0.00087	0.0808	60521	0.50449	0.2941	0.54231	-6.81261
2	feat_engg	15	30	10	32	2	0.0	mse_loss	0.1	0.00625	0.0012	0.51727	21395	0.44795	0.27439	0.52382	-6.28846
3	feat_engg	15	30	5	32	2	0.0	mse_loss	0.1	0.00625	0.00161	0.27197	19145	0.43094	0.22028	0.46933	-4.85148
4	feat_engg	15	30	10	64	2	0.05	mse_loss	0.1	1e-05	0.0019	0.77826	62771	0.4934	0.28625	0.53503	-6.60492
5	feat_engg	15	30	10	32	2	0.05	mse_loss	0.1	1e-05	0.00247	0.2487	21395	0.45799	0.27709	0.5264	-6.36032
6	feat_engg	15	30	5	64	2	0.05	mse_loss	0.1	1e-05	0.00265	0.06383	60521	0.51573	0.29772	0.54564	-6.90825
7	feat_engg	15	30	5	32	2	0.05	mse_loss	0.1	1e-05	0.00423	0.18761	19145	0.49069	0.27379	0.52325	-6.27268
8	base	5	10	5	32	2	0.0	mse_loss	0.1	0.0125	0.00635	0.33268	14525	0.35699	0.18065	0.42503	-3.69188
9	base	5	10	5	32	2	0.05	mse_loss	0.1	1e-05	0.00918	0.43605	14525	0.41096	0.21917	0.46816	-4.69258
10	feat_engg	15	30	10	32	2	0.0	l1_loss	0.1	0.00313	0.01894	0.5728	21395	0.49028	0.28038	0.52951	-6.44822
11	feat_engg	15	30	5	32	2	0.0	l1_loss	0.1	0.00156	0.02287	0.29954	19145	0.47771	0.25657	0.50653	-5.81534
12	feat_engg	15	30	10	32	2	0.05	l1_loss	0.1	1e-05	0.02938	0.52801	21395	0.49334	0.30013	0.54784	-6.97167
13	feat_engg	15	30	5	32	2	0.05	l1_loss	0.1	1e-05	0.04485	0.45661	19145	0.52169	0.30554	0.55276	-7.11593
14	base	5	10	10	32	2	0.0	mse_loss	0.1	0.00625	0.06315	0.1692	14775	0.67176	0.50519	0.71077	-12.12028
15	base	5	10	5	64	2	0.0	mse_loss	0.1	1e-05	0.23766	1.06298	53341	1.13966	1.33652	1.15608	-33.71431
16	base	5	10	5	64	2	0.05	mse_loss	0.1	1e-05	0.23767	1.0638	53341	1.13994	1.33716	1.15636	-33.73103
17	base	5	10	10	64	2	0.0	mse_loss	0.1	1e-05	0.23957	1.1621	53591	1.1447	1.34813	1.16109	-34.01694
18	base	5	10	10	64	2	0.05	mse_loss	0.1	1e-05	0.23964	1.16199	53591	1.14479	1.34834	1.16118	-34.02258
19	base	5	10	10	32	2	0.05	mse_loss	0.1	1e-05	0.24172	1.38807	14775	1.15699	1.37714	1.17352	-34.76929
20	feat_engg	15	30	10	64	2	0.0	l1_loss	0.1	0.00039	0.37894	1.1977	62771	1.28394	1.69136	1.30052	-43.93186
21	feat_engg	15	30	10	64	2	0.05	l1_loss	0.1	1e-05	0.39482	1.22619	62771	1.27996	1.67946	1.29594	-43.61323
22	feat_engg	15	30	5	64	2	0.0	l1_loss	0.1	1e-05	0.40187	1.17097	60521	1.26561	1.63959	1.28046	-42.55704
23	feat_engg	15	30	5	64	2	0.05	l1_loss	0.1	1e-05	0.40195	1.17063	60521	1.26517	1.63862	1.28009	-42.53143
24	base	5	10	5	32	2	0.0	l1_loss	0.1	1e-05	0.41304	1.29417	14525	1.31626	1.77041	1.33057	-44.98507
25	base	5	10	5	32	2	0.05	l1_loss	0.1	1e-05	0.41307	1.29417	14525	1.31592	1.76951	1.33023	-44.96159
26	base	5	10	10	64	2	0.0	l1_loss	0.1	1e-05	0.41406	1.29832	53591	1.32181	1.7857	1.3363	-45.38212
27	base	5	10	10	64	2	0.05	l1_loss	0.1	1e-05	0.41411	1.29821	53591	1.32185	1.7858	1.33634	-45.38477
28	base	5	10	5	64	2	0.05	l1_loss	0.1	1e-05	0.4142	1.29236	53341	1.32155	1.78486	1.33599	-45.36045
29	base	5	10	5	64	2	0.0	l1_loss	0.1	1e-05	0.41421	1.29217	53341	1.32137	1.78439	1.33581	-45.34846
30	base	5	10	10	32	2	0.05	l1_loss	0.1	1e-05	0.41446	1.33155	14775	1.32229	1.78708	1.33682	-45.41584
31	base	5	10	10	32	2	0.0	l1_loss	0.1	1e-05	0.41447	1.33148	14775	1.32215	1.78669	1.33667	-45.40571

Table 4: `ConvLSTM` grid search results.

model	data_version	n_feat	nhead	d_model	num_layers	dropout	criterion	init_lr	final_lr	train_loss	test_loss	n_params	MAE	MSE	RMSE	R-squared
0	feat_engg	15	2	64	2	0.0	mse_loss	0.1	0.0125	0.00164	0.11644	563653	0.28859	0.11591	0.34046	-2.07985
1	feat_engg	15	4	64	2	0.0	mse_loss	0.1	0.0125	0.00173	0.1028	563653	0.26622	0.10309	0.32108	-1.73919
2	feat_engg	15	2	64	2	0.05	mse_loss	0.1	1e-05	0.0057	0.11783	563653	0.32473	0.13454	0.3668	-2.57472
3	feat_engg	15	4	64	2	0.05	mse_loss	0.1	1e-05	0.00599	0.11457	563653	0.32409	0.13241	0.36389	-2.5182
4	base	5	4	64	2	0.0	mse_loss	0.1	0.00625	0.01423	0.16538	563013	0.36514	0.16366	0.40455	-3.2518
5	base	5	2	64	2	0.05	mse_loss	0.1	1e-05	0.01518	0.17498	563013	0.40424	0.19773	0.44467	-4.13583
6	base	5	4	64	2	0.05	mse_loss	0.1	1e-05	0.01711	0.17943	563013	0.40936	0.20224	0.44971	-4.25293
7	feat_engg	15	2	64	2	0.0	l1_loss	0.1	0.00313	0.01895	0.30849	563653	0.30331	0.12305	0.35078	-2.26839
8	base	5	2	64	2	0.0	l1_loss	0.1	0.00156	0.02644	0.25501	563013	0.2519	0.09405	0.30667	-1.44284
9	feat_engg	15	4	64	2	0.0	l1_loss	0.1	0.00039	0.0281	0.33601	563653	0.33103	0.14624	0.38241	-2.88503
10	base	5	4	64	2	0.0	l1_loss	0.1	0.00313	0.02969	0.25846	563013	0.25598	0.0945	0.3074	-1.45407
11	base	5	2	64	4	0.0	l1_loss	0.1	0.00313	0.03278	0.30273	1125317	0.29473	0.12335	0.35121	-2.20306
12	base	5	4	64	4	0.0	l1_loss	0.1	0.00313	0.03371	0.31149	1125317	0.30277	0.12943	0.35976	-2.36131
13	feat_engg	15	4	64	4	0.0	l1_loss	0.1	0.00078	0.03541	0.3241	1125957	0.31723	0.13426	0.36641	-2.56635
14	feat_engg	15	2	64	4	0.0	l1_loss	0.1	0.00078	0.03717	0.32254	1125957	0.31783	0.13188	0.36315	-2.50355
15	feat_engg	15	2	64	2	0.05	l1_loss	0.1	1e-05	0.04338	0.35612	563653	0.36891	0.17565	0.41911	-3.66603
16	feat_engg	15	4	64	2	0.05	l1_loss	0.1	1e-05	0.04341	0.31873	563653	0.33507	0.14503	0.38083	-2.85267
17	feat_engg	15	2	64	4	0.05	l1_loss	0.1	1e-05	0.05445	0.31317	1125957	0.32972	0.14271	0.37777	-2.79223
18	feat_engg	15	4	64	4	0.05	l1_loss	0.1	1e-05	0.05916	0.3628	1125957	0.37635	0.18006	0.42434	-3.78311
19	base	5	2	64	4	0.05	l1_loss	0.1	1e-05	0.06812	0.3336	1125317	0.3479	0.15791	0.39738	-3.10077
20	base	5	4	64	4	0.05	l1_loss	0.1	1e-05	0.06916	0.33223	1125317	0.34718	0.15597	0.39493	-3.05046
21	base	5	4	64	2	0.05	l1_loss	0.1	1e-05	0.06971	0.31006	563013	0.34257	0.14909	0.38612	-2.87238
22	base	5	2	64	2	0.05	l1_loss	0.1	1e-05	0.07745	0.31566	563013	0.34701	0.15005	0.38737	-2.89726
23	base	5	2	64	2	0.0	mse_loss	0.1	0.0002	0.15042	0.9365	563013	0.94304	0.92629	0.96244	-23.05601
24	feat_engg	15	2	64	4	0.05	mse_loss	0.1	1e-05	0.20844	1.12594	1125957	1.0368	1.1128	1.05489	-28.5708
25	feat_engg	15	4	64	4	0.05	mse_loss	0.1	1e-05	0.22141	1.19244	1125957	1.06736	1.17666	1.08474	-30.26392
26	feat_engg	15	4	64	4	0.0	mse_loss	0.1	1e-05	0.2422	1.31503	1125957	1.12317	1.29901	1.13974	-33.50938
27	feat_engg	15	2	64	4	0.0	mse_loss	0.1	1e-05	0.2425	1.31645	1125957	1.12377	1.30037	1.14034	-33.54539
28	base	5	2	64	4	0.0	mse_loss	0.1	1e-05	0.24305	1.38195	1125317	1.15199	1.36559	1.16858	-34.46993
29	base	5	4	64	4	0.0	mse_loss	0.1	1e-05	0.24306	1.38154	1125317	1.15182	1.36518	1.16841	-34.45932
30	base	5	2	64	4	0.05	mse_loss	0.1	1e-05	0.24321	1.36638	1125317	1.15088	1.365	1.16833	-34.45527
31	base	5	4	64	4	0.05	mse_loss	0.1	1e-05	0.24427	1.36624	1125317	1.15179	1.36723	1.16928	-34.51238

Table 5: `EncTransformer` grid search results.