Financial time series forecasting with Transformers: Impact of additional features and cross-validation on model performance

Anton Zaitsev — anton.zaitsev.001@student.uni.lu — University of Luxembourg

June 24, 2024

Abstract

In this study, we explore various strategies to assess performance of Transformer-based models in forecasting on market data. Specifically, we compare the effectiveness of using only the historical prices of a single ticker against adding additional stock features, as well as adding prices from the correlated tickers. Furthermore, we propose a data splitting technique in which small validation batches are allocated in-between training batches throughout the timeline. The performance is evaluated with the mean absolute error (MAE) and mean absolute percentage error (MAPE). The results show that the default data splitting leads to consistently lower errors. The incorporation of additional features in train data does not improve model performance.

Keywords: forecasting; stock markets; time series; deep learning; transformer models; feature selection; time series splitting.

1 Introduction

1.1 Background

Financial time series forecasting has always been a challenge due to complexity, volatility, and randomness inherent in data. Traditional methods for stock price prediction often rely on statistical models or machine learning techniques, which may not fully capture complex spatial and temporal dependencies present in data. The rise of deep learning models has offered a promising way to better capture such relationships and enhance forecasting accuracy. Recently developed Transformer architectures, initially introduced by Vaswani et al. (2017)[9] for natural language processing tasks, have proved to work well with time series data (Wen et al., 2022)[10]

1.2 Problem statement

Despite the success of Transformer-based models in time series prediction tasks, optimizing their performance for stock market forecasting still remains a challenge. The question remains as to which data should be used to help the model grasp complex relationships so that it can yield more accurate results. Key questions include: Does adding stock-related features introduce more noise or improve prediction accuracy? Does including correlated stock price data enhance the model's predictive power?

The conventional methods for splitting data into training and validation sets may be suboptimal in time series forecasting. Given that market time series data is nonstationary, i.e., its statistical properties alter over time, training a model on historical data and validating on subsequent data creates a mismatch. In other words, the model is essentially being evaluated on a distribution it hasn't seen during training, which greatly limits its generalization capabilities in forecasting. A natural question arises: How can we design a splitting technique that better captures the temporal nature of nonstationary time series?

1.3 Research hypotheses

We propose two hypotheses to assess the prediction accuracy of Transformer-based models for time series forecasting:

1. Introducing more input features: By feeding more training features into the Transformer-based model we assume that the quality of its predictions will improve. By incorporating a richer set of input features, we believe that the model can capture not only temporal, but also spatial dependencies within the data, which in turn may lead to better forecasting performance.

2. Changing the cross-validation technique: By implementing a custom data splitting technique that includes multiple validation splits throughout the time series, we anticipate an increase in the model's generalization capabilities and thus in prediction quality. Since stock data has temporal patterns which are not well-represented in the sequential split, we believe this approach may better capture such variations.

1.4 Research contributions

We study whether adding more data features, in the form of either correlated stocks or additional stock features, leads to better performance of Transformer-based models for financial time series forecasting task. Additionally, we evaluate a custom train/validation splitting approach against a traditional method and deduce whether it has a positive impact on the model forecasting accuracy.

2 Related work

This section reviews some of the existing literature on methods and models employed in financial time series forecasting.

Some recent studies have explored the integration of Transformer architectures in stock market forecasting. For example, Li et al. (2023) [5] proposed the MASTER (Market-Guided Stock Transformer) network for stock price forecasting. MASTER models the realistic stock correlation and guide feature selection with market information, which might prove useful in combination with regression models for price prediction task. Their proposed Transformer design demonstrated superior performance compared to various established baselines, underscoring the potential of customized approaches in enhancing the forecasting performance of deep learning models in financial markets.

Muhammad et al. (2022) [6] employed a Transformer-based model to predict price movements across multiple stocks. While they demonstrated the forecasting capabilities of their model, the study lacked a comparison with a naive baseline model where future values are predicted based on current values. This comparison could provide insights into the relative effectiveness of Transformer-based approaches in financial forecasting tasks.

In a systematic review, Zou et al. (2020) [12] examined the use of various deep learning models for stock price prediction. They show that the Long Short-Term Memory (LSTM) models are widely applied in the forecasting task, even though Transformers, a more recent advancement, have shown superior performance.

3 Materials

3.1 Data

The research is based on the historical stock market data[1]. The tickers considered in the analysis are highly correlated (by price) technology stocks in the US: *ORCL*, *CSCO* and *QCOM* (see Figure 1). Each ticker is described by multivariate time series of daily historical price and volumes values, daily technical indicator figures (*MACDEXT*, *RSI* and *BBANDS*), and quarterly balance sheet statements (cash flow, earnings and income).

Prices are available in *OHLCV* format but only **c**losing and **v**olume values are used.

[1] All data are available online via Alphavantage API.

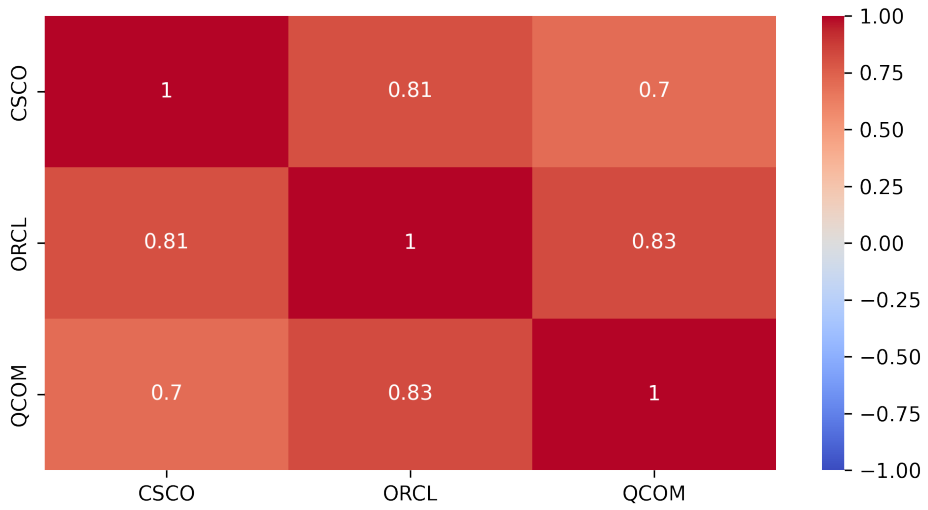

Figure 1: Rank correlations between tickers by price

Balance sheets are manually cleared to ensure no overlapping occurs between sheets and within statements, even though features are later selected algorithmically during processing. Given the inconsistencies in balance sheets between tickers, only the intersection of statements among all tickers is considered. Since statements are fiscal and available on a quarterly basis, the reported figures are propagated backwards to unify the time interval between features, so that the merging of columns by timestamp is possible.

The observable time period is set between Feb-2000 and Apr-2024 and the total number of observations is $N = 6092$.

The aggregated dataset can then be described as a model input $\left[X_{t,n} \colon \left(X^1_{t,n_1} \bigcup X^2_{t,n_2}\right), Y_t\right]$ where $Y_t = \{c_t, l_t\}$ is a set of daily closing historical stock prices c_t and a directional label l_t of the target ticker, X^1 is the matrix of balance sheet statements, X^2 is the matrix describing figures corresponding to technical indicators, volumes and closing prices of correlated tickers. The subscript t refers to number of rows (timestamps) and n to number of columns, respectively.

The final set of features for each ticker is $X_{t,n} \subset \left(X^1_{t,n_1} \bigcup X^2_{t,n_2}\right)$. Inclusion of features in X depends on the importance score towards the target ticker's c_t, bar that of the correlated tickers. The importance scores for each variable of each ticker are calculated using Random Forest [2] and XGBoost [3] models. The scores are then summed and divided by the number of tickers. Features with an importance score (towards c_t) exceeding a threshold of 5% are part of the final feature set. Such features have turned out to be only $\{BBANDS\} \supset X^2$.

The directional label is used by the model to output probabilities of the direction it predicts. l_t takes values in the set $\{0, 1, 2\}$ and is calculated as follows:

$$l_t = \begin{cases} 0, \frac{c_t}{c_{t-1}} - 1 < -0.05 \\ 1, -0.05 \leq \frac{c_t}{c_{t-1}} - 1 \leq 0.05 \\ 2, \frac{c_t}{c_{t-1}} - 1 > 0.05 \end{cases}$$

Based on this, we test the hypotheses by training the model on three different sets: **(a)** only c_t is used to predict the future prices; **(b)** additionally to **(a)**, matrix X^2 is used to predict the future prices; **(c)** additionally to **(a)**, prices of correlated tickers are used to predict the future prices. The target set y remains equal across all training configurations.

3.2 Sliding window

To conform to the Transformer-based models' requirements, we segment the time series data into patches by using sliding window approach (see Figure 2). Each patch captures a specific window of historical data. In our case, the history window, L, is set to 64 timestamps (days) and forecast window is set to 1, meaning that the model takes past 64 days of data to predict the target values for the next day. The stride is set to 1.

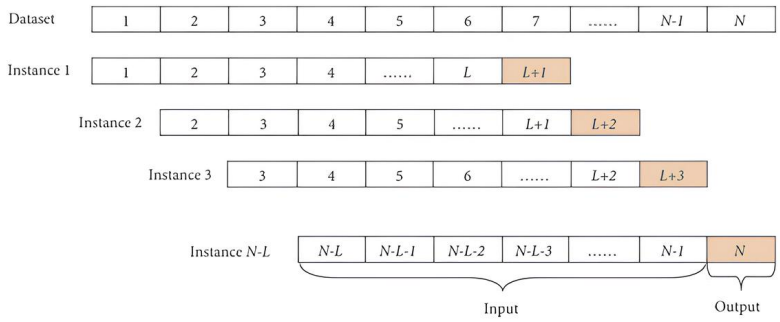

Figure 2: Sliding window technique for time series data. Source: [11].
In our case, $L = 64$ and $N = 6092$ (total number of timestamps).

3.3 Train & Validation Splits

Figure 3 illustrates the default train/validation splitting approach. In this approach, the last 20% of the time series data are reserved for validation, while the initial 80% are used for training.

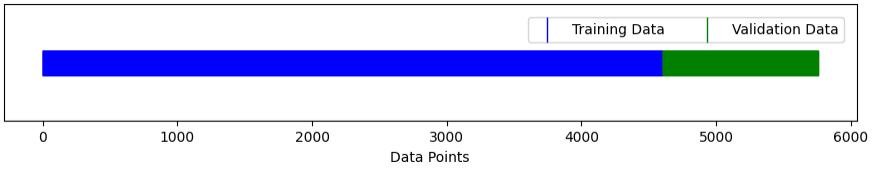

Figure 3: Default train/validation data splitting approach.
Ratio of the validation data is 20%.

Our proposed custom train/validation data splitting approach is shown in Figure 4. In this approach, the validation batches are allocated throughout the entire data timeline. The size of the training/validation batches is optimized by iteratively adjusting the training and validation window sizes to maximize the total data used while ensuring that the validation set comprises 15% to 20% of the total data. The number of validation batches is set to 24 (relating to the fact that we have 24 years of training data).

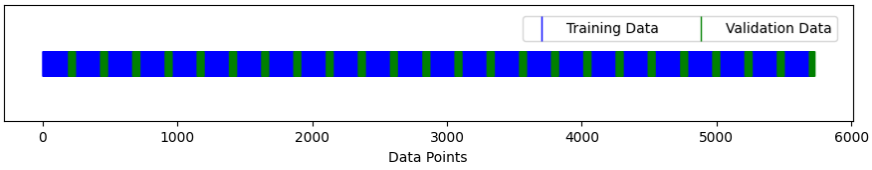

Figure 4: Custom train/validation data splitting approach.
Calculated training batch size is 190 and validation batch size is 46; ratio of the validation data is $\approx 19.7\%$.

Note that before splitting the data into train/validation sets, we reserve 5% of the latest data for inference.

3.4 Preprocessing

To ensure consistency in our model's inputs, we standardize the entire dataset (including the test data) to have a mean of 0 and a variance of 1, based on the statistics calculated from the training indices. Thus, the scaling differs between the default and custom splitting approaches, since the training indices are different.

4 Methods

4.1 Transformer

As a Transfomer backbone model we use PatchTST introduced by Nie et al. (2023)[7], which currently holds the state-of-the-art position for multivariate time series forecasting tasks. PatchTST does not support an inconsistent number of input and output channels (see Figure 5), meaning that that the number of features used for training should equal to the number of predicted features. To address this limitation, we modify PatchTST to output a single channel by reconstructing the last fully connected layer (see Figure 6). Additionally, to enable the model to output both price values and probabilities for three defined classes, we design a custom head with two outputs. The first output, for price prediction, is produced by a fully connected layer with an output dimension of 1. The second output, for label classification, is produced by a fully connected layer with an output dimension of 3. For the classification task, we use the cross-entropy loss, which requires raw outputs rather than applying a softmax activation function.

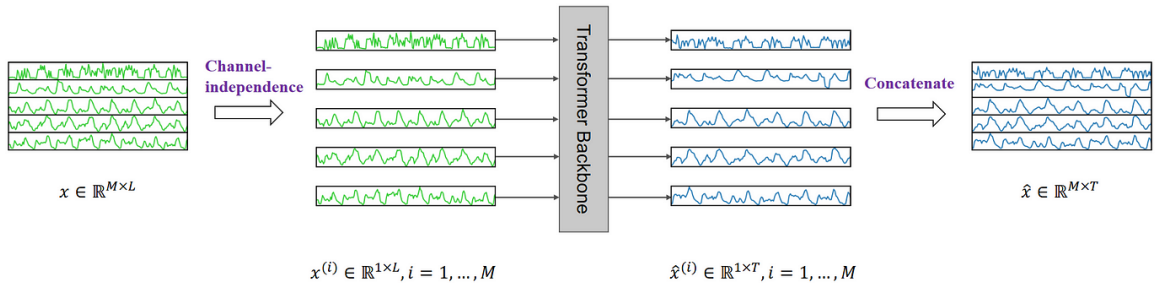

Figure 5: Original PatchTST model overview.
Here, M should be consistent before and after passing the data through the Transformer Backbone.
Source [7].

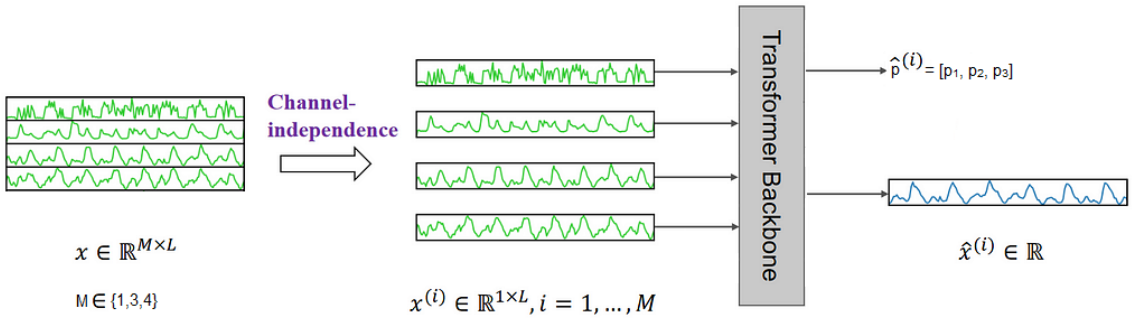

Figure 6: Model used: PatchTST backbone and custom head.

The specific parameters and configurations used in the model are as follows (see Figure 7 for the visual representation of the parameters):

1. **Number of encoder layers (n_layers)**: the depth of the *Transformer Encoder* part is set to **3** layers.
2. **Number of attention heads (n_heads)**: each *Multi-Head Attention* block in *Transformer Encoder* uses **16** heads.
3. **Model dimension (d_model)**: the dimensionality of the input and output *Position Embeddings* is **64**.
4. **Feedforward dimension (d_ff)**: the dimension of the feedforward layer within *Transformer Encoder* is **256**.
5. **Attention dropout (attn_dropout)**: A dropout rate of **0.2** is used for the attention probabilities in *Multi-Head Attention* blocks.
6. **Dropout (dropout)**: A dropout rate of **0.2** is used for all fully connected layers in the *Transformer Encoder*.
7. **Stride (stride)**: The patch stride length is set to **16**.
8. **Patch length (patch_len)**: Each patch of the input sequence is of length **16**.

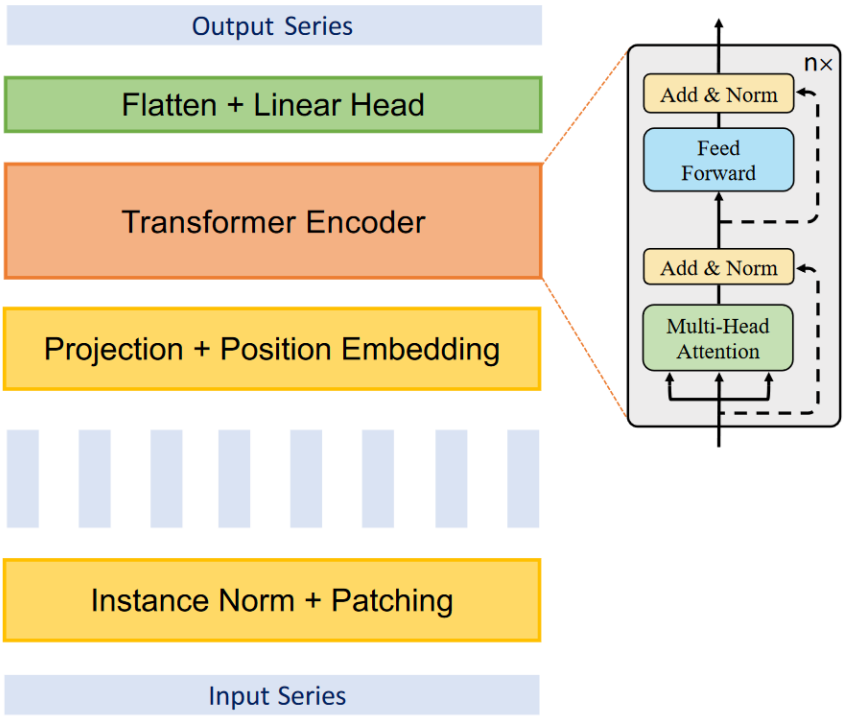

Figure 7: PatchTST "Transformer backbone" architecture overview.
Source [7].

Note that these parameters were found using *Optuna* [1] library with 400 trials and *TPESampler* sampler. The optimization process is run on the task **(a)** with the default train/validation split.

4.2 Training

During the training stage, we set the torch seed to 137. We use **Adam** [4] optimizer with 0.001 as the learning rate and utilize **1cycle** [8] policy for faster training convergence with the maximum learning rate of ≈ 0.00038 (which is also found using *Optuna*). We use the mean squared error (MSE) to calculate the loss for the price (*Close*) predictions and the weighted cross-entropy (CE) to calculate the loss for the label (l) predictions with the following weights for the labels $[0, 1, 2] : [2, 1, 2]$. For the total loss, we take the weighted sum of these two losses: we multiply the price loss by 10 and add the label loss.

After the epoch of training, we evaluate the current model state on the validation set. We save the best state of the model based on the total loss (computed the same way as during training stage) for the validation set. Since the training process is rather fast, we do not use early stopping technique. However, during training, we only save the model that achieved the lowest total loss value.

5 Results and discussion

5.1 Results

We compare the models' performance against each other and the naive approach, in which the next stock price value equals the today's value, i.e., $y_{t+1} = c_t$. The results for the inference on the allocated 5% of the data (test set) can be seen in Table 1. The best model and metric values are shown in **bold**; the second best are underlined.

Task / Model	Split	Price MAE	Price MAPE	Label CE
baseline	-	**0.4103**	**0.8052**	-
a	default	0.4526	0.8898	1.1026
a	custom	0.4820	0.9454	1.1017
b	default	0.5753	1.1200	1.1302
b	custom	0.5613	1.0995	1.1151
c	default	0.5418	1.0694	1.0856
c	custom	0.6102	1.2044	1.0640
d_baseline	-	**1.2497**	**1.1808**	-
d	default	1.3514	1.2776	-
d	custom	1.3765	1.3075	-

Table 1: Performance metrics for different model configurations.

Based on the conducted analysis, none of the hypotheses are successful. The models trained on the custom train/validation splits consistently show subpar results compared to models trained on default train/validation splits. Moreover, adding additional features into the training set (tasks **(b)** and **(c)**) does not enhance model performance. Ultimately, each model performs poorer in comparison with the naive baseline.

Regarding l_t, each model demonstrates inability to successfully predict the direction of market prices throughout time. Probabilities remain consistently below 50% no matter the label. For any t, the models' confidence remain between 30%—40%.

Since our initial approach does not yield satisfactory results, we speculate that Transformers are not well-suited for univariate financial time series prediction. However, there could be potency in a multivariate case. Thus, we consider a specification **(d)** in which prices of three correlated stocks are used in training as well as in predicting; directional labels are ignored. The performance of **(d)** is compared against the similar baseline model, where MAE and MAPE are calculated as the average across the errors for each price. The results do not support the aforementioned hypothesis: the model trained on the default train/validation splits still performs better. Neither of the two models surpass the performance of the naive model.

5.2 Discussion

Upon reviewing the results, we confirm that financial time series forecasting presents significant challenges. Simple next-value predictions, even with Transformers, are not sufficient — more sophisticated and out-of-the-box approaches are necessary.
The insights gained after this study are the following:

1. **Modify custom cross-validation approach**: after the study, we noticed that our train / validation splitting approach might conflict with the sliding window technique. Namely, since all of the training batches are used during one training iteration, at some point during training model has to predict indices that are in the next training batch. The same applies to the validation set. Naturally, this creates confusion for the model. One way to address this is to train the model iteratively, batch by batch. First, train the model on the first batch, then evaluate it on the first batch. Next, train the model on the second batch and evaluate it on the second batch. Continue this process for each batch until the last batch is reached. This entire cycle would constitute one epoch. This approach would eliminate all inconsistencies with the training and validation indices.

2. **Utilize time series K-Fold cross validation**: another way to improve the forecasting performance of the model is to use time series K-Fold cross validation. This will likely enhance the model's accuracy, although it will substantially increase the training time.

3. **Try other models**: PatchTST may not be the best choice for this task. Exploring other models could help identify one that better fits the requirements.

4. **Ensemble prediction**: after identifying models with the lowest error values, combining predictions from multiple models into a single weighted prediction could help mitigate individual model weaknesses and improve overall forecasting performance.

6 Conclusion

In this study, we explore various strategies to assess the performance of Transformer-based models for financial time series forecasting tasks. Namely, we utilize PatchTST as the backbone model with a custom-designed head for univariate and multivariate prediction. We propose a cross-validation technique for time series data and test whether incorporating additional features in the training data can improve the Transformer model performance. Ultimately, none of these strategies are proven to be effective. The models trained on the default train/validation splits consistently outperform those trained on custom splits. Furthermore, the models trained solely on price data perform better than those trained on prices as well as additional stock features.

References

[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework, 2019.

[2] L. Breiman. Random forests. *Machine Learning*, 45:5–32, 2001.

[3] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16. ACM, August 2016.

[4] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[5] Tong Li, Zhaoyang Liu, Yanyan Shen, Xue Wang, Haokun Chen, and Sen Huang. Master: Market-guided stock transformer for stock price forecasting, 2023.

[6] Tashreef Muhammad, Anika Bintee Aftab, Muhammad Ibrahim, Md. Mainul Ahsan, Maishameem Meherin Muhu, Shahidul Islam Khan, and Mohammad Shafiul Alam. Transformer-based deep learning model for stock price prediction: A case study on bangladesh stock market. *International Journal of Computational Intelligence and Applications*, 22(03), April 2023.

[7] Yuqi Nie, Nam H. Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. A time series is worth 64 words: Long-term forecasting with transformers, 2023.

[8] Leslie N. Smith. Cyclical learning rates for training neural networks, 2017.

[9] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[10] Qingsong Wen, Tian Zhou, Chaoli Zhang, Weiqi Chen, Ziqing Ma, Junchi Yan, and Liang Sun. Transformers in time series: A survey, 2023.

[11] Huihui Zhang, Shicheng Li, Yu Chen, Jiangyan Dai, and Yugen Yi. A novel encoder-decoder model for multivariate time series forecasting. *Computational Intelligence and Neuroscience*, 2022:1–17, 04 2022.

[12] Jinan Zou, Qingying Zhao, Yang Jiao, Haiyao Cao, Yanxi Liu, Qingsen Yan, Ehsan Abbasnejad, Lingqiao Liu, and Javen Qinfeng Shi. Stock market prediction via deep learning techniques: A survey, 2023.