# Apply Recurrent Neural Networks and Transformer to Predict Ventilator Pressure

**Na-Yun Lai** [1]

## Abstract

Many real-world applications require the prediction of sequence tim-e series such as electricity consumption, stock markets, and language processing. The recurrent neural network(RNN) and Transformer are good tools for solving the time-series data. However, there are still some obstacles to these two methods, RNN may face vanishing gradients problems which cause more error and too much computation time problem. Forward-propagation algorithm(FPTT) tries to divide the time into smaller fractions to get better gradients. Transformer has better performance than RNN because Transformer takes the key vectors from the sequence input and gives weight to the decoder layer to predict the output. The improved Transformer: Autoformer decomposes the vector in the inner layer, Encoder, and Decoder to solve the long sequence data which may cause the model out of the memory. Therefore, I am interested in dealing with time-series data. To compare RNN and Transformer, I selected the time-series data set from `https://reurl.cc/KXXkL9`. The data is about ventilator data with the air pressure which will change over time. In the end, the RNN has better performance than Transformer. The RNN accuracy is up to 85% and the Transformer accuracy is 10% . The reason I consider is that the input features from the dataset can not reflect the key factors of the air pressure. As the result, RNN which computes the former data relationship with the next data may have a better performance.

## 1. Substantive review and critique

### 1.1. Introduction

Time-series data, also referred to as time-stamped data, is a sequence of data points indexed in time order. Stock markets, weather prediction, and language translation are all time series data. Therefore, how can we deal with the time-series data and predict the future?

The recurrent neural network(RNN) and the Transformer are good tools to solve sequence problems. Long-Short-Term Memory(LSTM), Back-propagation through time(BPTT), and Forward-propagation through time(FPTT) are the major methods of recurrent neural networks(RNN). In the paper I selected: ”Training Recurrent Neural Networks via Forward Propagation Through Time”(Kag & Saligrama, 2021). They introduce RNN and LSTM and they also use a novel forward-propagation algorithm (FPTT) to improve the LSTM. They update RNN parameters by optimizing an instantaneous risk function. They take a gradient step of an instantaneously constructed regularized risk at time $t$. The regularizer can evolve dynamically based on previous loss. Comparing to ordinary LSTM model, the LSTM model with FPTT has lower loss and higher accuracy.

The other paper I selected are ”Autoformer: Decomposition Transformers with Auto-Correlation for Long-Term Series Forecasting”(Wu et al., 2021), ”Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting”(Zhou et al., 2020) and ”Attention is all you need”(Vaswani et al., 2017). They all mention Transformer is an extinguished tool for sequence modeling and transduction tasks and is notable for its use of attention to model long-range dependencies in the data. Transformer models have shown superior performance in capturing long-range dependency than RNN models.(Zhou et al., 2020). However, there are still some problems for Transformer in long-term future prediction such as quadratic time complexity, high memory usage, and the inherent limitation of the encoder-decoder architecture. In these papers, there are some solutions such as Autoformer: breaking the pre-processing convention of series decomposition and renovating it as a basic inner block of deep models.

### 1.2. Methods

#### 1.2.1. Recurrent Neural Network

Recurrent Neural Networks (RNNs) have been successfully employed in many sequential learning tasks including language modeling, speech recognition, and terminal prediction.(Kag & Saligrama, 2021)

At every time step, the network can be unfolded for k time steps to get the output at k+1 time step . The unfolded net-

| Features | Descriptions |
|---|---|
| x$_t \in R$ : | The input at time step t |
| y$_t \in R$ : | The output of the network at time step t |
| h$_t \in R^m$ : | Vector stores the values of the hidden units/states at time |
| w$_x \in R^m$ : | Weights associated with inputs in the recurrent layer |
| w$_h \in R^m$ : | weights associated with hidden units in the recurrent layer |
| w$_y \in R^m$ : | Weights associated with hidden units to output units |
| b$_h \in R^m$ : | Bias associated with the recurrent layer |
| b$_y \in R$ : | Bias associated with the feedforward layer |

Table 1: Description of data

work is very similar to the feedforward neural network. The hidden layer factor at t+1 can be expressed as an activation function f:

$$h_{t+1} = f(x_t, h_t, w_x, w_h, b_h) = f(w_x x_t + w_h h_t + b_h)$$

The output $y_t$ can be computed as:

$$y_t = f(h_t, w_y) = f(w_y h_t + b_y)$$

Hence, the network computes the values of the hidden units and the output after time steps. The weights associated with the network are shared temporally. Each recurrent layer has two sets of weights: one for the input and the second for the hidden unit. The last layer computes the final output for the kth time step, is just like an ordinary layer of a traditional network.

The paper also introduces LSTM which is also an application in RNN. LSTM can learn how to bridge minimal time lags of more than 1,000 discrete time steps.(Staudemeyer & Morris, 2019) In Fig.1, to determine what to discard from the block in the forget gate. In the input gate, the sigmoid is the f function in a previous equation which decides values to let through 0 and 1. Tanh function gives weight to the values which are passed deciding their level of importance ranging from -1 to 1. In the output gate, the input and the memory of the block are used to decide the output. In the ordinary LSTM, the parameter will be upgraded after the T length sequence. The T length sequence can be composed of small t time steps. In contrast, FPTT updates parameters

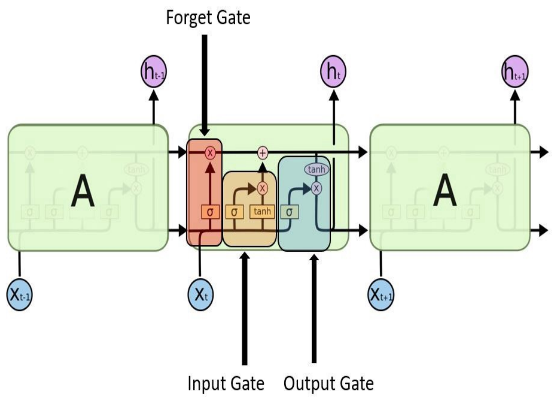


Figure 1: Transformer architecture from Understanding RNN and LSTM

at each t time steps. The new parameters(t+1) will be calculated by minimizing previous loss of parameter(t) and regularizer. It requires more stability and more computation resources.

### 1.2.2. TRANSFORMER

Transformer is a sequence-to-sequence(Seq2Seq) architecture. Seq2Seq transforms a given sequence of elements, such as the sequence of words in a sentence, into another sequence as output. Seq2Seq models consist of an Encoder and a Decoder. The Encoder takes the input sequence and maps it into a higher dimensional space (n-dimensional vector). That abstract vector is fed into the Decoder which turns it into an output sequence.Fig 2 is the transformer architecture from "Attention is all you need"

An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors.(Vaswani et al., 2017) On the other hand, comparing to the LSTM in RNN. An attention takes the key vectors from the sequence input and gives weight to the decoder layer to predict the output. However, RNN only takes the previous vectors as input, which may miss the key points of the architecture. The equation of the Attention can reveal the different concepts with RNN.

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

$Q$ is a matrix that contains the query $K$ are all the keys(vectors in linear output) that describe the relationship from input to output. $V$ are the Learned vector( Linear layer output) as a result of calculations, related to input. For example, in the text sequence, one word is the **query** of the

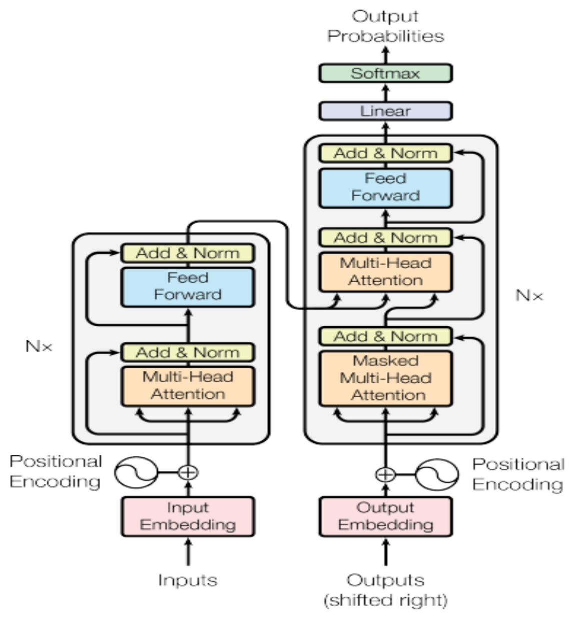


Figure 2: Transformer architecture from (Vaswani et al., 2017)

matrix $Q$. $K$ are vector representations of all the words in the sequence and the $V$ is the best matched **values**.
Even the Transformer shows superior performance in capturing long-range dependency than RNN models. L-quadratic computation and memory consumption on L-length inputs/outputs cause the models can not afford. There are lots of improvements of Transformers to computation and architecture efficiency, as well as maintaining higher prediction capacity. (Zhou et al., 2020) mentions three significant limitations in Transformer to solve the long-term series problems. The first one is the quadratic computation of self-attention. The operation of the self-attention mechanism will cause the time complexity and memory usage to be $O(L^2)$. Second is the memory bottleneck in long inputs. There is a memory problem if receiving long sequence inputs. The third is when the Transformer dynamic decoding the computation may slow down the whole model. To solve the problems: Autoformer, one of the improved methods for the Transformer. The method decomposes the vector in the inner layer, Encoder, and Decoder. First, the methods create a block that extracts the long-term stationary trend from predicted intermediate hidden variables progressively. It applies average pooling to smooth the periodic fluctuations and keep the same series unchanged. Second, Autoformer embeds the series decomposition block as an inner operator, which can progressively aggregate the long-term trend part from intermediate prediction. On the other hand, the auto-correlation conduct dependencies discovery and information aggregation at the series level. It can record the computation efficiency and information utilization.

### 1.3. Critque Review

To solve the time-series problems, the RNN and Transformer are introduced. Three paper all show their unique methods to decrease the error, decrease computation units, improve accuracy, and improve efficiency. In the RNN model, (Kag & Saligrama, 2021) introduced RNN, LSTM, and FPTT to solve the time-series problems. When calculating the parameters, they divide the time into smaller fractions to decrease the error. I think it is a good idea to get more accuracy. But the big problem I am concerned is the time-consuming problem. If I have to calculate the gradient at a small fraction of time and compare the loss with the previous loss. I consider it will cost lots of computation units and time. In the result they presented, they only focus on the numbers of the parameters and the accuracy they increase.
In the Transformer, the (Zhou et al., 2020) mentions some limitations in the Transformer. They also do several experiments to compare different Transformers. It is important to learn the architecture of the Transformer and what is related to the efficiency of Transformers. The way to decrease memory usage and computational units help me to design my Transformer model in the future. Most of all, knowing the complexity of different layers can help me to modify and improve Transformers from others. In the Autoformer literature, they introduce the concepts of decomposition which deconstruct a time series into several components. As the result, each representing one of the underlying categories will be more predictable. They embedded their decomposition block in the Encoder layer and Decoder layer to decrease the loss. I think that is a brilliant idea, giving the average pooling to smooth the fluctuation and remain the same shape. Another design of the Autoformer is the auto-correlation mechanism which conducts dependencies discovery can reflect the efficiency. However, the paper also does not discuss and compare the time-consuming problems.
Next, after understanding the contents of the literature, I would like to find the time-series dataset and apply RNN and Transformer to it. I would like to compare the difference if the Transformer is better than the RNN models. I am going to discuss the dataset and the application in the next chapter.

## 2. Description of implementation, evaluation and discussion

### 2.1. Dataset

I selected the dataset from `https://reurl.cc/KXXkL9`, The data is about ventilator data. A ventilator is a rescue machine when a patient has respiratory problems. The doctors insert a tube into the windpipe of a sedated patient and use a ventilator to push oxygen into the lungs. Mechanical breathing requires a clinician's expertise, which can be a drawback in certain situations, such as dur-

ing Covid-19 pandemic. Therefore, creating an automatic model to detect patients' situations and adjust the parameters of the ventilator to control the airway pressure. In the data, there are 8 columns of features. Id, breath_id, R, C, time_step, u_in, u_out are the input features. The pressure is my target which I will predict by RNN and Transformer.
As we know, breathing is a cycle progress. The pressure

| Features | Descriptions |
|---|---|
| breath_id | Globally-unique time step for breaths |
| R | Lung attribute.The restriction of airway(cm $H_2O/L/S$). The higher R means that it is more difficult to blow the air from the lungs |
| C | Lung attribute. The compliant level of the lung (ml/cm$H_20$). It is the change in volume per change in pressure. The higher C means it is easier to blow from the lung. |
| time_step | The actual time stamp |
| u_in | The control input for the inspiratory solenoid valve from 0 to 100 |
| u_out | A binary data (0 or 1). The control input for the exploratory solenoid valve |
| pressure | The output, the airway pressure measured in the respiratory circuit (cmH20) |

Table 2: Description of data

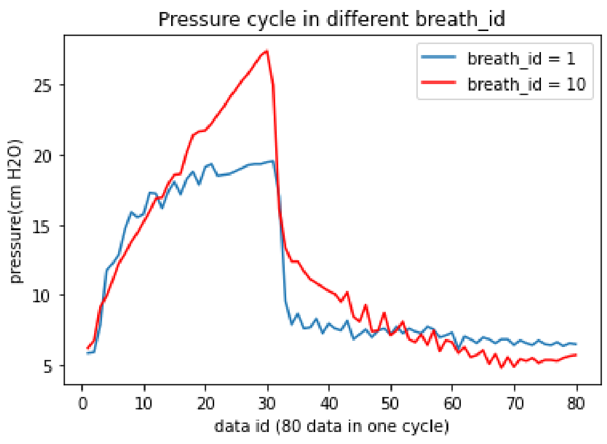


Figure 3: pressure cycle in different breath_id

will change through inhaling and exhaling air. For this reason, the pressure data can be regarded as time series data. The pressure may change according to previous data. In the table, each breath_id represents one patient or one breathing cycle. There are 80 rows of data in one breath_id. By different R, C, u_in and u_out, the different pressure can be analyzed.

## 2.2. Data Preprocess

Table 2 reveals that there are only 6 columns to predict the pressure, which means there are not plenty of features. Moreover, the breath_id is not an input feature to pressure and the time_step are similar to each other to different breath_id.h As a result, I group the data with the same breath_id. Every 80 rows are one breathing cycle. To apply RNN and Transformer, I shift the data backward and forward for 1 row, 2 rows,3 rows, and 4 rows as lag data and the next data. Besides, I also subtract the shifting data as a difference and store it in new columns.

**Algorithm 1** Create New columns

```
u_out_lag1 ← u_out.shift(-1)
u_out_lag2 ← u_out.shift(-2)
u_out_next1 ← u_out.shift(1)
u_out_next2 ← u_out.shift(2)
u_out_diffu_out - u_out_lag1)
u_out_diff2 ← u_out - u_out_lag2)
```

Moreover, the R and C don't change in the same breath_id sample. Therefore, I can create a conditional column for a specific circumstance such as creating a column $C = 20$. If the data $C = 20$, the cell will be 1. A dummy matrix is composed of conditional columns with the value of C, R, the relationship of C and R. Afterward, I have a sparse matrix which is more efficient for calculations. As the Fig. 4, I have 3 conditions for C and 3 conditions for R, and 9 combinations of C and R.I will create 18 new columns dummy matrix. At the end of the process, I drop the "id", "breath_id", which is not related to pressure as input train data. I also drop the "time_step data" because the time_step data is similar to each other so in each breath_id I regard it as an index data. After the preprocessing data, I got a new data set with 62 columns.

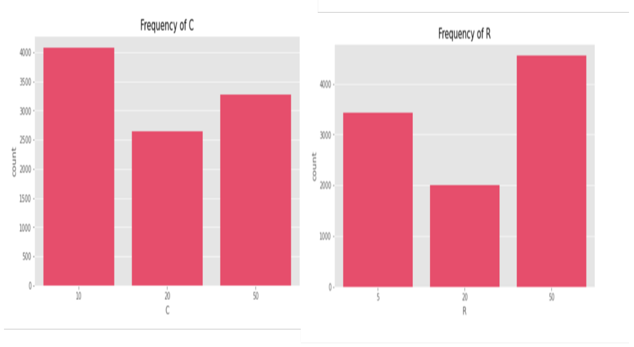
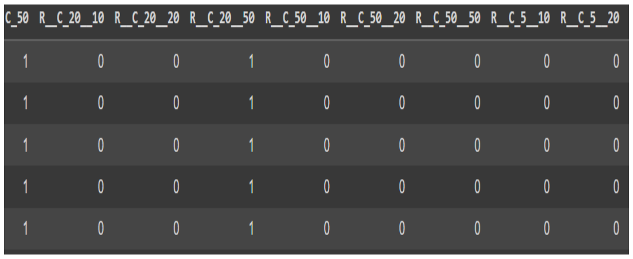


Figure 4: Categories of R and C



| C_50 | R__C_20__10 | R__C_20__20 | R__C_20__50 | R__C_50__10 | R__C_50__20 | R__C_50__50 | R__C_5__10 | R__C_5__20 |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

Figure 5: Dummy Matrix of C and R

## 2.3. Code and Methods Implementation

In the code part, I also write the LSTM model code and take reference from `https://github.com/anilkagak2/FPTT`. I take reference of the Transformer code from the Kaggle.

### 2.3.1. RNN

In the RNN model, I apply tensor.keras.layers create LSTM layers. There are 5 LSTM layers, 1 activation layer and 1 output layer. Besides, I use LearningRateScheduler to find the best learning rate during the training section.

### 2.3.2. Transformer

In the Transformer, the layers are multi-head attention, 128 hidden layers in feed forward network, 2 layers of normalization, and drop-out layers. First, to create a model, 128 hidden layers are regarded as embedding layers. Next, place the embedding layers into multi-head attention, which is in regarded as the encoding layer. Afterward, put the output from the encoding layer to multi-head attention and decode it. At the same time, the loss and cross-validation errors are recorded.

## 2.4. Experiment Results

I apply 3 epochs for each model. I also record the time of training model, the loss through the epochs and the prediction pressures.

The R square of RNN model in the validation data is 0.85, which means there is 85% accuracy. On the other hand, the R square of the Transformer is only 0.1, which shows there is only 10 % accuracy. In the time-consuming part, the RNN consumes longer than Transformer model.

| Model | Time consumption | R square |
|---|---|---|
| RNN | 595.99 s | 0.85 |
| Transformer | 355.35 s | 0.1 |

Table 3: Comparison of two models

As the loss figures, the loss decreases dramatically through the epochs, which shows that the RNN and the Transformer

In the loss of RNN, the error in training data and validation data decrease dramatically. They are really close at epoch 3.

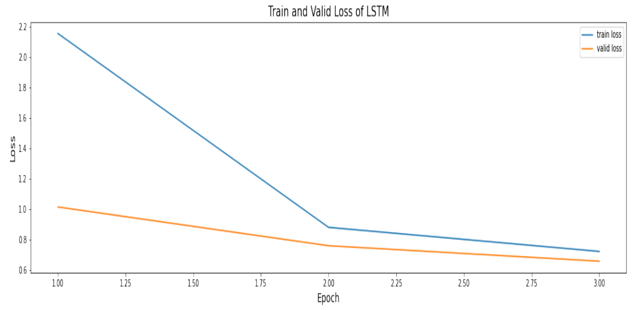


Figure 6: Loss of the RNN

In the prediction of RNN, the predicted data almost fit the true data. Even in the peak region, the fluctuating way also looks similar.

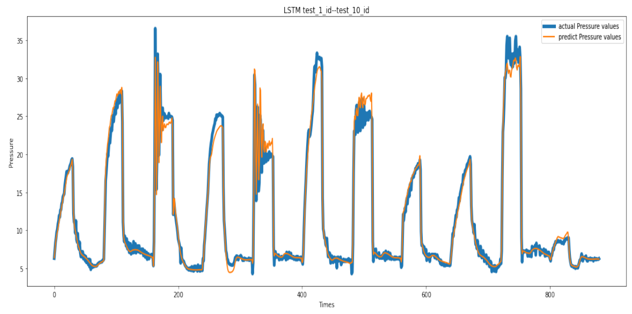


Figure 7: True pressure and validation pressure

In the loss of Transformer, the error in training data and validation data decrease. However, The loss crosses at epoch 3 which means the model may be overfitting.

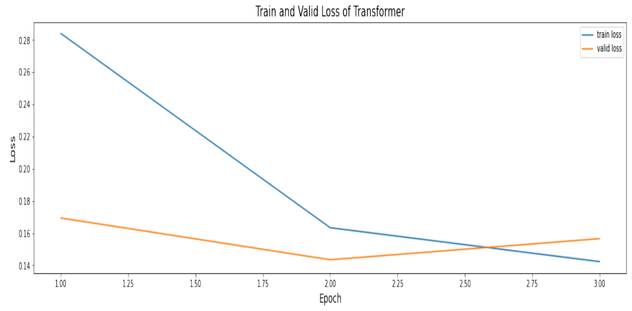


Figure 8: Loss of the Transformer

In the prediction of Transformer, the performance is not as good as RNN. There are still a bias between predicted data and true data. In the flat region, the model predicts several peaks, which cause a large error. However, if the peak occurs, the Transformer model does predict every peak in these 10 samples.

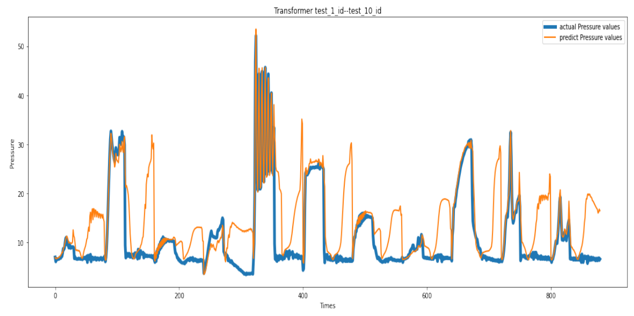


Figure 9: True pressure and validation pressure

are both the suitable model for this time-series dataset. However, the prediction is out of what I expected. At first, I thought the Transformer would get a better performance. However, through the figures, in the ventilator dataset. the RNN model has a better performance. The drawback from th RNN is the running time. The RNN train model running time is 1.5 times greater than the Transformer model.

### 2.5. Discussion

As a result, the performance of the RNN is better than the Transformer model. In my opinion, the type of data may be the big reason. The air pressure is the output of the human body. Each sample from a different breath_id is independent. The main feature which controls the pressure may not input data. That means it is more difficult for Transformer to find the key value of the sequence data. It is possible there is no key vector in the input data. On the other hand, the RNN is good at taking the gradients from previous data, which is more similar to the breathing cycle. In my opinion, the air pressure can be the description from human mechanism. The influence without data may be larger than the influence of the input data.

## 3. Conclusion

In the paper, I study the long-sequence time-series forecasting problem and understand the Recurrent Neural Network and the Transformer. They are both good tools for solving time series problems. The LSTM stores parameter and pass to the next layer. The Attention can determine the key vectors and combine them and compute parallel in the training model. The self-attention mechanism can reduce the maximum length of network signals traveling paths and avoid the recurrent structure. The literature all discusses decreasing error and improving efficiency. They add Forward Propagation Through Time in LSTM and the Autoformer adds a decomposition block into the encoder and decoder layer. I apply their concepts and how they compare the difference, how they select data, how they preprocess data, and how they explain data. In the end, the RNN has better performance than Transformer in the Ventilator Pressure dataset

## References

Kag, A. and Saligrama, V. Training recurrent neural networks via forward propagation through time. In Meila, M. and Zhang, T. (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 5189–5200. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/kag21a.html.

Staudemeyer, R. C. and Morris, E. R. Understanding LSTM - a tutorial into long short-term memory recurrent neural networks. *CoRR*, abs/1909.09586, 2019. URL http://arxiv.org/abs/1909.09586.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Wu, H., Xu, J., Wang, J., and Long, M. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. *CoRR*, abs/2106.13008, 2021. URL https://arxiv.org/abs/2106.13008.

Zhou, H., Zhang, S., Peng, J., Zhang, S., Li, J., Xiong, H., and Zhang, W. Informer: Beyond efficient transformer for long sequence time-series forecasting. *CoRR*, abs/2012.07436, 2020. URL https://arxiv.org/abs/2012.07436.