# Electrical Power Consumption Forecasting with Transformers

Jun Wei Chan
*School of Computer Science and Engineering*
*Nanyang Technological University*
Singapore, Singapore
s200132@e.ntu.edu.sg
Chai Kiat Yeo
*School of Computer Science and Engineering*
*Nanyang Technological University*
Singapore, Singapore
asckyeo@ntu.edu.sg

2022 IEEE Electrical Power and Energy Conference (EPEC) | 978-1-6654-6318-8/22/$31.00 ©2022 IEEE | DOI: 10.1109/EPEC56903.2022.10000228

***Abstract*—Until recently, state of the art (SOTA) deep learning methods for time series prediction problems, such as electricity load forecasting, have been based on recurrent neural networks (RNN), convolutional neural networks (CNN), or combinations thereof. However, RNNs involve sequential computations that cannot be parallelized on machine learning accelerators, while CNNs require very deep networks to capture long distance relationships. This paper proposes a sparse transformer based method for time series prediction. The proposed model achieves comparable accuracy to the SOTA method, TSRNN, on the London Smart Meter dataset while achieving up to 10 times faster inference speed.**

## I. Introduction

Electrical power consumption prediction over short-term periods can help power generation companies better respond to load variations [1]. Currently, common deep learning methods for such time series forecasting tasks are built upon recurrent neural networks (RNN) and variants of it such as long short term memory (LSTM), convolutional neural networks (CNN), and combinations of them. However, the use of recurrence in RNNs results in sequential computation that cannot be parallelized during model training. Likewise, the limited size of convolution kernels in CNNs means that very deep networks are required to ensure that the receptive field covers all inputs.

The transformer architecture has been introduced to address the sequential computation shortcoming of RNNs [2]. Instead of processing each element of a sequence sequentially as in RNNs, transformers interleave feed-forward layers, shared between all positions of a sequence, with mixing layers that allow elements in different positions to interact in forming the output. Since its introduction in the natural language processing (NLP) field, transformer based methods have become the gold standard there, surpassing previous methods based on RNNs. Beyond NLP, transformers have also been successfully applied to computer vision tasks [3].

This paper proposes a transformer based model for electricity consumption forecasting incorporating a sparse attention mechanism to scale to long input sequences, and benchmarks the model against the convolutional RNN SOTA method called TSRNN proposed in [4]. The proposed method achieves comparable accuracy to TSRNN, but is faster than TSRNN in inference.

The rest of this paper is organized as follows: section II discusses the relevant background and related works, section III elaborates the proposed method, and section IV details the experiments conducted and presents the results. Finally, section V concludes this paper.

## II. Related Work

### A. Recurrent Neural Networks

Recurrent neural networks model a sequence as a recurrence relation involving internal states of the network. This is implemented by feeding the output of the neural network, or some intermediate result, back into the front network as an input. However, RNNs are difficult to train on long sequences due to vanishing or exploding gradients, as gradients have to be propagated across many time steps. LSTMs have been introduced to address this problem by providing a separate path for information to propagate, bypassing the major computational layers.

Reference [4] proposed reshaping a time series into a sequence of 2D images, then using Eidectic LSTMs [5] to reconstruct the masked unknown pixels (datapoints to be predicted), providing a prediction of future values. The dimensions of the image are equal to the number of samples per day, and the number of samples in 7 days (1 week). This arrangement allows the transformed images to better demonstrate daily and weekly periodic trends. This transformation also drastically reduces the number of 'time steps' the Eidectic LSTM needs to process the entire input, making it less susceptible to exploding/vanishing gradients. To further alleviate vanishing gradients, the Eidectic LSTM also adapts a technique from fully connected LSTMs, whereby each state depends on $\tau > 1$ previous hidden states instead of only the immediately preceding hidden state as in RNNs. However, the recurrent nature of TSRNN means it is restricted to processing transformed images sequentially.

### B. Transformers

Transformers have demonstrated competitive performance in other tasks beyond NLP where it was introduced, including in time series prediction [6]. Unlike RNNs, transformers process all inputs in a sequence simultaneously and in parallel through alternating blocks of feed-forward and attention modules. Feed-forward blocks are shared between all positions in the sequence and can be applied in parallel. The attention blocks compute, for each position in the output sequence, a weighted sum of transformed inputs. Thus, each position in the output can be said to pay varying attention to the input positions, allowing different positions to interact. At its introduction, dot product attention has been used in these blocks, which computes the attention weights from dot products between all pairs of vectors transformed from input elements at their respective positions. This results in an $O(N^2)$ memory cost which dominates the cost of the transformer for long sequences. To address this limitation, alternative means of mixing have been proposed, such as sparse attention mechanisms [7], [8], and Fourier transforms [9].

### C. Sparse Attention

Sparse attention is a class of attention mechanisms with lower computational cost compared to dot product attention's $O(N^2)$ cost in the length of the input sequence, allowing transformers to better scale to longer inputs. This is achieved by imposing sparsity patterns in the attention map, reducing the number of attention weights that need to be computed and applied. For example, LogSparse attention allows a position to attend only to previous positions in the preceding layer separated by an exponentially increasing step size [7]. Thus, the cost of a single layer is reduced to $O(N\log(N))$, although additional layers are needed to recover full connectivity between all positions in the output and input, bringing the overall complexity to $O(N\log^2(N))$.

Big Bird [8] is another sparse attention pattern that uses reshaping operations to efficiently implement sparsity. Big Bird exploits the principle of locality and allows output positions to attend to a window of surrounding positions. Hence, its attention map is approximately a band matrix. The input is chunked into blocks and reshaped before multiplication so that the number of computations is reduced. A small subset of positions, the length of one block, are still allowed global attention (to attend to all positions from the previous layer).

## III. Proposed Method

This paper proposes a transformer equipped with sparse attention to generate forecasts. Sparse attention transformers are better suited for electrical load forecasting compared to full dot-product attention as their reduced memory requirements allow them to handle longer inputs and make forecasts based on a longer history. Figure 1 shows the overall model architecture.

The transformer consists of two major sections, the first is the encoder, which transforms the input sequence through repeated applications of dot-product attention within the input sequence. This step preserves the length of the sequence. Then, the decoder generates the output sequence by applying dot-product attention between the encoded input and the output sequences as it is being formed, or a placeholder sequence in the first decoder layer. Some works use only the encoder of the transformer [6]. For the proposed model however, the full encoder-decoder structure is used as the decoder can directly generate a forecast sequence of the desired length.
The main components of the proposed model are detailed in the following subsections.

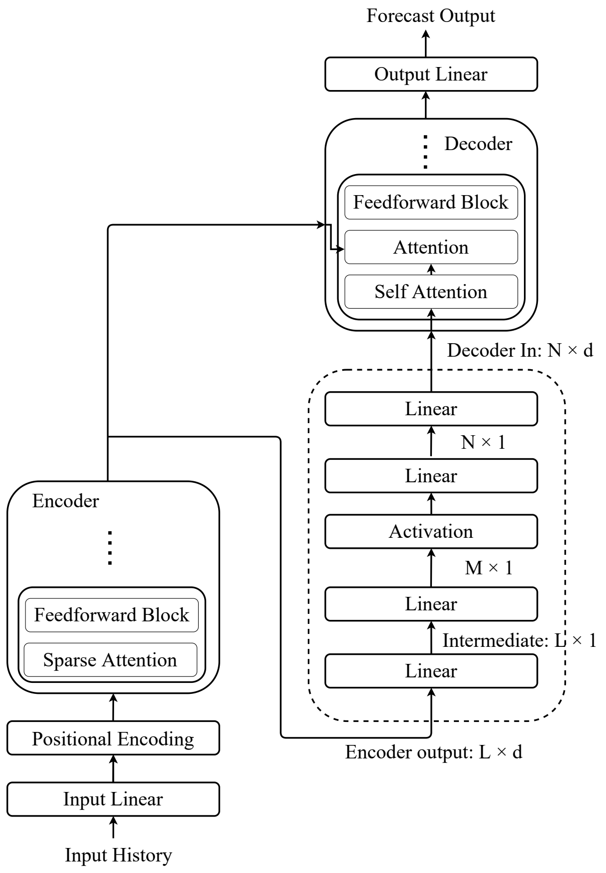

Fig. 1. Overall structure of the proposed transformer based model. L is the length of the input sequence. d is the embedding dimension of the transformer. N is the length of the output sequence. M is the internal intermediate length for the decoder initial sequence generator.

### A. Input

The attention block uses the dot product between vectors to compute the attention weights. Hence, the expressive power of the attention mechanism is tied to the size of the vectors comprising the input. In this case, the electricity power consumption data is 1 dimensional. Therefore, the input is expanded to a larger embedding dimension by a linear layer. A linear layer at the output of the decoder transforms the output of the decoder into the 1 dimensional forecast sequence.

The feed-forward layers of the transformer is shared for all sequence positions, and the behaviour of the attention operation does not vary with position (i.e. permutation invariant). Therefore, information about a value's position in the sequence needs to be added to augment the input. A fully learnable positional encoding is added to the input after the input linear layer.

## B. Encoder

The encoder consists of a stack of encoder layers. Each layer has a multihead self-attention module followed by a two layer feedforward section. Figure 2 shows the construction of the encoder half of the transformer. The attention mechanism is further explained in the next subsection.

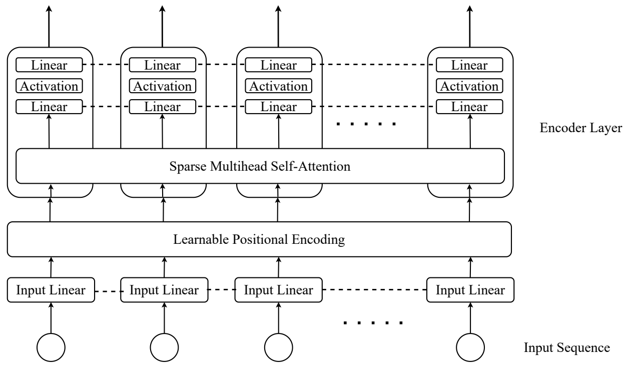

Fig. 2. Transformer structure, encoder side. Dashed lines indicate layers which are shared between inputs. Only the first encoder layer is shown, although the same structure is repeated in later layers.

## C. Attention

Dot product attention is the most widely used mixing mechanism in transformers. Dot product attention is called as such because the attention weights are computed from dot products between sets of query and key vectors. Self-attention, used in the encoder, is described by the following equation [2]:

$$\text{Attention}_{\text{self}}(X) = \sigma\left(\frac{Q(X)(K(X)))^T}{\sqrt{d}}\right)V(X) \quad (1)$$

Where $X \in \mathbb{R}^{n\times d}$ is the $n$ length sequence of $d$ dimensional input vectors. $Q$, $K$, and $V$ are the query, key, and value[1] linear transformations, possibly including bias vectors. $\sigma$ is the softmax function.

This is called self-attention as the query and key vectors are both derived from the input sequence. In the decoder attention block, the query vectors are derived from the output sequence, while the key vectors are derived from the encoder output, as follows:

$$\text{Attention}(X,Y) = \sigma\left(\frac{Q(Y)(K(X)))^T}{\sqrt{d}}\right)V(X) \quad (2)$$

Where $Y \in \mathbb{R}^{n'\times d}$ is the sequence of output vectors of some other length $n'$. In this case, $n'$ would be the desired forecast horizon. Figure 3 contains a graphical summary of the attention functions as used in the encoder and decoder.

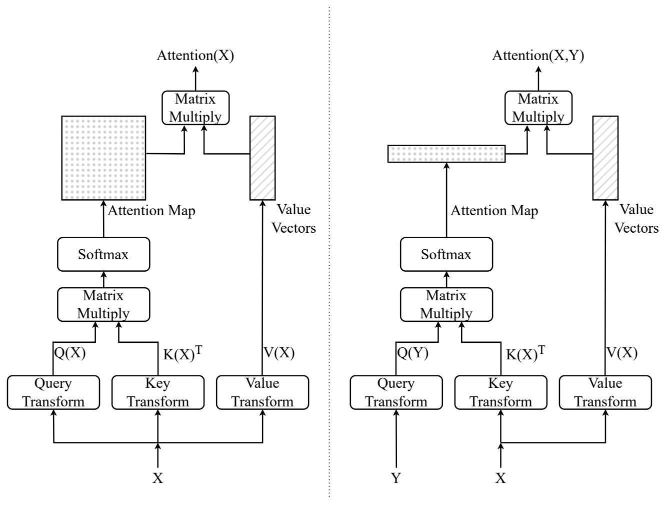

Fig. 3. Visual representation of the attention function as used in the transformer. Left side shows the self-attention as used in the encoder. Right side shows attention as used in the decoder. Variable labels correspond to the variables in equations 1 and 2.

Typically, a variant of attention known as multihead attention is used, where the $d$ dimensions of the vectors are split into groups called heads, and attention is applied to each group separately, before being combined by a concatenation. This allows different attention patterns to be learned in a single layer.

The matrix multiplication of the attention map with the value vectors can be interpreted as, for each position in the output, selecting value vectors to contribute based on their attention weight. This allows a value vector at one end of a sequence to influence the opposite end of the sequence if the corresponding attention weight is large enough, and is how long distance dependencies can be modeled in a single layer. However, in the encoder, this results in a quadratic number attention weights that need to be computed. This cost of the encoder multihead self-attention dominates the computational cost of the entire transformer. Hence, the encoder layers would benefit most from replacing the self-attention with sparse versions.

Big Bird's [8] sparse attention technique is used as it efficiently implements sparsity without using gather operations. Its sparsity pattern is depicted in Figure 4.

## D. Decoder

Similar to the encoder, the decoder consists of a stack of decoder layers. Starting from some initial value, the output is constructed layer by layer through the decoder. Each decoder layer applies self-attention to the output, followed by attention between the output and the processed sequence output from the encoder. As with encoder layers, this is followed by a two layer feedforward section.

[1]These names are derived from their origins in NLP

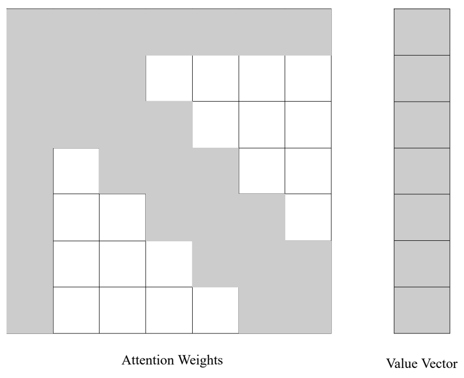

Fig. 4. Sparsity pattern for Big Bird attention. Shaded blocks indicate active attention weights while the attention weights in the unshaded blocks are 0. [8]

The decoder requires an initialization sequence of the same length as its output. This is generated from the encoder output by a 4 layer feedforward network. The first linear layer reduces the encoder output from the embedding dimension back to 1 dimension. The next fully connected linear layer stretches the sequence to a larger intermediate size (M in Figure 1) in the time dimension. An activation function introduces nonlinearity before the third layer compresses the sequence down to the length of the forecast horizon. The fourth layer expands the dimension of the sequence up to the embedding dimension as in the input linear layer. At the end of the decoder, the output of the decoder is reduced to a 1 dimensional forecast sequence by an output linear layer.

GELU activation is used for all activation functions. Following the recommendations from [6], batch normalization is used in place of layer normalization, and learnable positional encoding is used. The transformer used has an embedding dimension size of 24, with 4 heads, and consists of 4 encoder layers and 4 decoder layers. The decoder input generator first transforms the encoder output sequence into an intermediate sequence of length $2560$, before converting it into the required length.

## IV. Experiments and Results

Experiments were conducted to evaluate the performance of the proposed model, comparing against the SOTA TSRNN alongside other baseline methods. The three baseline methods include traditional time series prediction methods, Seasonal Auto Regressive Integrated Moving Average (SARIMA, [10]) and Exponential Smoothing [11] as well as deep learning model, long short-term memory (LSTM).

### A. Dataset

All the experiments are done on the London Smart Meter dataset [12]. This benchmark contains the energy consumption readings of 5,567 households in 112 blocks between November 2011 and February 2014 in kilowatt-hours (kWh). It consists of 5,567 half-hourly series, and the starting and ending dates are not the same for all series. The min and max consumption values of the different series are also different, and all the consumption values are in the range of [0 to 10] kWh.

### B. Hardware

Training, and accuracy tests were performed on 2 Nvidia RTX 3090 GPUs.
For the timing tests, only 1 GPU was used.

### C. Implementation

The PyTorch framework was used for these experiments. As TSRNN was originally implemented in Tensorflow, in order to compare against TSRNN in beyond the conditions in [4], TSRNN has been reimplemented in PyTorch in accordance with the description, codes and model parameters given in [4].

With regards to the dataset, households are randomly distributed into 60% train, 20% validation, and 20% test splits. Since the energy consumption of each household in a block is stored as one time series, the models in the experiments thus learn from multiple series of the households in the training set and then make forecast on the series from the testing set. Each household series is further chunked into disjoint segments containing both the look-back period and the prediction horizon ground truth. Look-back periods of 5 weeks and 9 weeks (corresponding to 1632 and 2928 half-hourly samples) are adopted in the proposed model. The prediction horizon is set to 24 hours corresponding to 48 samples. The look-back periods and prediction horizon are chosen to match those used by TSRNN. Note that TSRNN's design requires the total duration of the look-back and the forecast periods to be an integer number of weeks.

Segments with more than $80\%$ invalid (missing or 0) values are discarded. Following the procedure in [4], the dataset is normalized using min-max normalization on a per household basis. Batch size is set to the largest multiple of 8 that can still fit in the available VRAM.

The transformer model is subclassed from the transformer implementation packaged with PyTorch, with the self-attention functions redefined accordingly. The code for the Big Bird sparse attention module was adapted from the Huggingface Transformers library [13].

### D. Training

For TSRNN, training follows the same setup as described in [4], except for the definition of the number of training epochs. Due to the large size of the London dataset, [4] used a random subset of 100 households for 1 epoch, and trained the model for 1000 such epochs. However, in this experiment the entire training set is used for 1 epoch. Correspondingly, the number of training epochs is reduced to 30. Adam was used as the optimizer with learning rate $1\text{e}{-4}$ and weight decay $1\text{e}{-2}$.

For the proposed model, Adam was also used as the optimizer, with initial learning rate $1\text{e}{-4}$ and weight decay

2e−3. The loss function used was the mean squared error loss. Learning rate scheduling by plateau reduction (ReduceLROn-Plateau) was used with the following parameters:

- factor = 0.5
- patience = 7
- minimum learning rate = 1e−6
- threshold = 5e−5

Early stopping is used in the training of the proposed model when the learning rate reaches the minimum setting, which ends training after 10 epochs without improvement in the validation loss, and the model weights corresponding to the best validation loss are restored.
Missing values in the inputs were set to 0s, while missing values in the ground truth prediction were ignored in the computation of the loss.

The transformer trains much faster than TSRNN, at around 82 seconds per epoch compared to TSRNN at around 265 seconds per epoch, but was observed to converge slower than TSRNN. Therefore it is trained for a maximum number of 300 epochs, although early stopping was usually triggered before this limit was reached. Accelerated multi-precision with BFloat16 was used for training both the transformer model and TSRNN.

### E. Results

From empirical results, it was observed that all weights and biases of the query and key transformations would sometimes converge to 0 with no visibly perceptible degradation in the forecast outputs when comparing the graphs of the predictions and the ground truths. Hence, a variant of the proposed prediction model is proposed and it is annotated by '(fixed)' hereinafter. By '(fixed)', it means that the transformer model in the proposed model has been modified to use fixed attention maps which are gleaned from prior empirical studies.

*1) Accuracy:* Root mean squared error (RMSE), mean absolute error (MAE), and symmetrical mean abolute percentage error (sMAPE) results are computed as the average of 3 training runs. The accuracy results are reported in Tables I and II.
Note that the results for the baseline methods, SARIMA, Exponential Smoothing and LSTM in Table I are obtained directly from [4].

The proposed model performs very slightly worse than TSRNN for input length (look-back period) of 5 weeks, but catches up with TSRNN when the look-back period is increased to 9 weeks.

*2) Inference Time:* To compare the inference time of the models, the models are run on 10 random inputs and the inference time is averaged. All tests are scripted to run consecutively in a single session to minimize variation due to changes in the environment. The inference times are reported in Tables III and IV.

The proposed model and its fixed attention variant are significantly faster than TSRNN. Furthermore, the variant with fixed attention weights is almost 2 times faster than the original proposed model and there is no noticeable decrease in accuracy.

TABLE I
ACCURACY SCORES FOR LOOK-BACK PERIOD OF 5 WEEKS

| Model | RMSE | MAE | sMAPE |
|---|---|---|---|
| SARIMA [4] | 0.126 | 0.089 | 0.806 |
| Exponential Smoothing [4] | 0.124 | 0.087 | 0.844 |
| LSTM [4] | 0.124 | 0.087 | 0.844 |
| TSRNN [4] | **0.094** | **0.063** | 0.711 |
| TSRNN (reimplemented) | 0.0956 | 0.0634 | **0.662** |
| Proposed transformer based model | 0.0950 | 0.0634 | 0.675 |
| Proposed variant model (fixed) | 0.0956 | 0.0637 | 0.675 |

TABLE II
ACCURACY SCORES FOR LOOK-BACK PERIOD OF 9 WEEKS.

| Model | RMSE | MAE | sMAPE |
|---|---|---|---|
| TSRNN (reimplemented) | 0.0888 | **0.0587** | **0.645** |
| Proposed transformer based model | 0.0886 | 0.0596 | 0.668 |
| Proposed variant model (fixed) | **0.0883** | 0.0592 | 0.663 |

TABLE III
INFERENCE TIMES FOR LOOK-BACK PERIOD OF 5 WEEKS.

| Model | Inference Time (ms) |
|---|---|
| TSRNN (reimplemented) | 25.8 |
| Proposed transformer based model | 8.80 |
| Proposed variant model (fixed) | 4.70 |

TABLE IV
INFERENCE TIMES FOR LOOK-BACK PERIOD OF 9 WEEKS.

| Model | Inference Time (ms) |
|---|---|
| TSRNN (reimplemented) | 56.5 |
| Proposed transformer based model | 9.70 |
| Proposed variant model (fixed) | 5.40 |

## V. CONCLUSION

This paper proposes a sparse transformer based method for time series prediction of electrical power consumption. The proposed method achieves comparable accuracy to current SOTA RNN based method, TSRNN, but with much faster inference speeds. The paper has also demonstrated that using fixed attention patterns which have been empirically observed in the transformer further speeds up training without measurably degradation in prediction performance.

## REFERENCES

[1] A. B. Nassif, B. Soudan, M. Azzeh, I. Attilli, and O. Almulla, "Artificial intelligence and statistical techniques in short-term load forecasting: a review," *International Review on Modelling and Simulations (IREMOS)*, vol. 14, p. 408, 12 2021.

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[3] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," pp. 213–229, 2020.

[4] Y. Liu, S. Dutta, A. W.-K. Kong, and C. K. Yeo, "An image inpainting approach to short-term load forecasting," *IEEE Transactions on Power Systems*, pp. 1–1, 2022.

[5] Y. Wang, L. Jiang, M.-H. Yang, L.-J. Li, M. Long, and L. Fei-Fei, "Eidetic 3d lstm: A model for video prediction and beyond," 2019. [Online]. Available: https://openreview.net/forum?id=B1lKS2AqtX

[6] G. Zerveas, S. Jayaraman, D. Patel, A. Bhamidipaty, and C. Eickhoff, "A transformer-based framework for multivariate time series representation learning." ACM, 8 2021, pp. 2114–2124.

[7] S. Li, X. Jin, Y. Xuan, X. Zhou, W. Chen, Y.-X. Wang, and X. Yan, "Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting," H. Wallach, H. Larochelle, A. Beygelzimer, F. d Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper/2019/file/6775a0635c302542da2c32aa19d86be0-Paper.pdf

[8] M. Zaheer, G. Guruganesh, K. A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang, and A. Ahmed, "Big bird: Transformers for longer sequences," H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 17 283–17 297. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/c8512d142a2d849725f31a9a7a361ab9-Paper.pdf

[9] J. Lee-Thorp, J. Ainslie, I. Eckstein, and S. Ontanon, "Fnet: Mixing tokens with fourier transforms." Association for Computational Linguistics, 2022, pp. 4296–4313.

[10] J. Durbin and S. J. Koopman, *Time Series Analysis by State Space Methods*. Oxford University Press, 5 2012.

[11] R. Hyndman, A. Koehler, K. Ord, and R. Snyder, *Forecasting with Exponential Smoothing*. Springer Berlin Heidelberg, 2008.

[12] J.-M. D, "Smart meters in london — kaggle," 2017. [Online]. Available: https://www.kaggle.com/datasets/jeanmidev/smart-meters-in-london?datasetId=4021

[13] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush, "Transformers: State-of-the-art natural language processing." Association for Computational Linguistics, 10 2020, pp. 38–45. [Online]. Available: https://www.aclweb.org/anthology/2020.emnlp-demos.6