Forecasting Energy Demand with Transformers
ETF: CS-7643

Gregory Loshkajian
Georgia Institute of Technology
gloshkajian3@gatech.edu

Nurettin Mert Ersöz
Georgia Institute of Technology
mert.ersoz@gatech.edu

Nilesh Domadia
Georgia Institute of Technology
ndomadia3@gatech.edu

Abstract

The ability to predict demand for different power sources is useful for a variety of applications, such as power trading and electricity grid management. We typically consider this a sequence-to-sequence task, where given the previous period(s) of some number of time series, a network learns to predict the next member(s) of a time series. While there have been strong results for short/medium term demand forecasts, longer horizons are often untenable computationally, or produce strong short-term forecasts which degrade in performance over time. To address these problems, we explored the performance of multiple transformer architectures on a dataset provided in ***Long-Term Planning of Integrated Local Energy Systems Using Deep Learning Algorithms*** *[4]. Below, we document our experimental setup, report our results, and interpret findings. Of particular interest, we find that Informer improves significantly on [4] for both electricity and heat demand over short and long-term horizons.*

1. Introduction/Background/Motivation

Accurate forecasts of energy demand are useful and necessary across many areas, such as operating power transmission systems, and wholesale energy pricing. This importance has only grown in recent years, with the introduction of renewable (often intermittent) energy resources, which groups like power grid managers need to account for when optimizing resources. Given the importance of having reliable input for such optimization tasks, having high quality forecasts is essential.

Forecasting on power time series data can be considered a sequence-to-sequence task, where given previous members of a sequence, we learn to predict the next member(s) of said sequence. While this might appear a good fit for classical methods like ARIMA, or recurrent approaches like LSTM; ARIMA has seen limited success outside of short-term cases with single time series, and LSTM approaches have been shown to degrade over truly long-term horizons (e.g. 1 week to 1 month) without assistance from convolutional networks or predesigned temporal features [5,7, 9].

To address this issue of capturing long-term temporal dependency, we explore the performance of multiple Transformer architectures for predicting energy demand; the canonical Transformer implementation as well as two extensions, Temporal Fusion Transformer, and Informer. To our knowledge, transformer-based architectures are new to this space. We also compare against two baselines, ARIMA, as well as the architecture specified by Taheri *et al.* [4] and explore the impact on predictive strength of different temporal feature representations and attention mechanisms.

Time series forecasting has a long and venerable history within statistical literature. Methods such as ARIMA have traditionally accounted for autoregressive temporal dependencies, and further extensions such as VAR have allowed for multivariate input/outputs to forecasting models with proper treatment of cointegration between different time series. Deep learning-based approaches to time series forecasting began to emerge more recently; with initial success from architectures based on specialized causal convolutional layers [2] and recurrent layers [3], both designed to be suitable for capturing sequential dependencies.

Later, LSTM based architectures [4] were introduced to address the exploding and vanishing gradients problem, which inhibited recurrent architectures from learning long-range dependencies in the data. The most successful neural network methods for power demand forecasting have generally leveraged such methods, as well as hybrid combinations of LSTM and CNN models [5,6,7]. Such approaches have traditionally relied on multivariate input, and pre-generated temporal features (e.g. weekdays) as input to help better capture seasonal/trend effects [7], but to our knowledge, represent state of the art in this space.

More recently, the development of attention mechanisms has further improved long term

dependency learning. Transformer architectures [8] have done particularly well on this score (especially in natural language processing applications), but have several limitations for time series forecasting, w.r.t. interpretability and performance. To address the former, Lim et *al.* introduced the Temporal Fusion Transformer [1], reporting ~9% better P90 loss compared to competing architectures, while also offering interpretable attention representation and feature importance metrics.

By contrast, architectures such as Informer [9] aimed to address the second problem via the introduction of sparse attention maps and self-attention distillation to eliminate the quadratic time/memory complexity of canonical attention calculation. These architectural choices allow for longer input sequences, holding computational resources fixed. Due to this, Zhou *et al.* show that Informer outperforms baselines like ARIMA, LSTM, and competing transformer forecasting architectures (e.g. LogTrans) by 10-20% on univariate and multivariate forecasts for short (< 1 week) and long term (>1 week) forecast horizons.

For this project, we utilize a building energy demand dataset (provided by Taheri *et al.* [4] as a test set for their LSTM implementation, obtained from IEEE [11]) to perform time-series forecasting of electricity and heat power demand. This dataset represents hourly energy audit results of a building in London between December 2010 and November 2018. More information such as a data dictionary and PCA analysis, are shown in appendix B.

Some of the data in this dataset happened to be missing, likely due to faulty collection procedures. In these cases, we fill the missing data with the previous value for a particular feature. We also removed the total cloud cover feature from the dataset process; not all samples were properly labeled. This leaves a dataset with ~70000 observations for 7 different features, two of which (electricity and heat demand) were response variables. Further processing was performed to generate temporal features (months, week, day of year) as one-hot-encoded variables. This enriched dataset, along with the original, were used in training, particularly to replicate the approach taken in [4].

We also note this dataset is highly seasonal, which any forecasting model must account for. Power and heat demand have multiple cycles, e.g. hourly and yearly, as Figure 1 shows, but heat demand has greater variance.

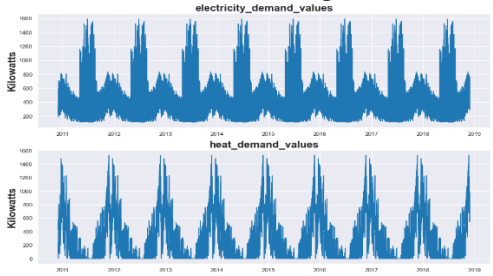

Figure 1. Electricity and Heat Demand Plots

2. Approach

For this project, we explored the performance of four deep architectures, compared with ARIMA as a baseline. In this section, we will describe these approaches. Where well documented, reproducible implementations of these approaches exist, we leveraged them. Otherwise, we developed Pytorch implementations. For all models, we assume input of $X \in \mathbb{R}^{B \times S_i \times F}$, where B is batch size, S_i and F is the sequence dimension of the input and features, respectively, and output a univariate sequence (the forecast) as $\mathbb{R}^{B \times S_o \times 1}$. S_o represents the length of the output sequence, we assume that this may be different for input and output sequences.

2.1. DTO-DRNN

In [4], Taheri *et al*. report that their best results, based on parsimony (less parameters) and high performance, come from a S,DTO-DRNN design. This architecture, shown in figure 2, consists of a stack of multiple LSTM layers (each with 300 units), followed by a stack of 3 linear layers with a fixed layer size of 15, which outputs the prediction sequence. Sigmoid activations are used after each layer, except for the LSTM cell gate, where tanh activation is used, as is standard, and the final layer.

While the authors have made available the public implementation for their LSTM model [10], their code has limited documentation, and leveraged a different framework than used by our other implementations. Therefore, we chose to use this repository as inspiration, and rewrote S, DTO-DRNN in PyTorch. Where the authors specify details re: number of nodes, weight decay, gradient clipping, etc., we reimplement these.

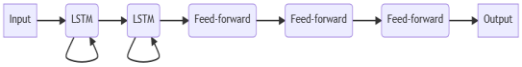

Figure 2: High Level Diagram of S, DTO-DRNN, based on [4]

2.2. Transformer

The overall architecture of Transformer including the embedding of sequential information directly follows the implementation detailed in the original paper [8]. Given an input X, the attention matrix is calculated from the query$\left(Q = XW_Q\right)$, key$(K = XW_K)$ and value $(V = XW_V)$ matrices for each attention head.

$$Attention(Q,K,V) = softmax\left(QK^T/\sqrt{d}\right)V \quad (1)$$

where d denotes the hidden dimension. Here, we also referenced a repository which implemented Transformer to compare with their Query Selector mechanism [13]. To reduce overfitting, we added dropout in the encoder after the positional embedding.

2.3. Temporal Fusion Transformer

The Temporal Fusion Transformer (TFT) [1] is an attention-based architecture for multi-horizon time series forecasting, designed to be interpretable w.r.t. temporal dynamics. TFT combines recurrent layers for processing temporal relationships with an interpretable self-attention mechanism for long term dependency.

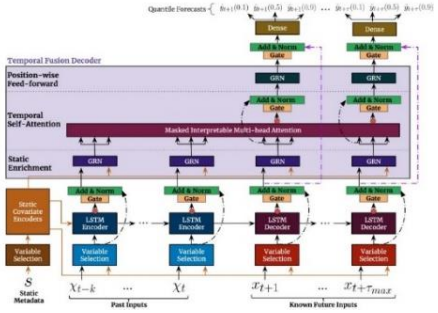

Figure 3. Temporal Fusion Transformer Architecture [1]

TFT also encompasses components for selecting relevant features and gating layers to suppress irrelevant components. All inputs to encoder and decoder layers first pass through this "Variable Selection Network".

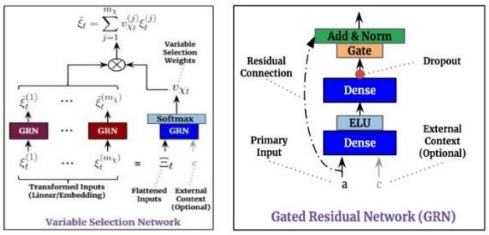

Figure 4. TFT Variable Selection and Gated Residual Networks [1]

TFT enables the use of static covariates (time invariant) by using a "static covariate encoder". Here, known feature inputs such as month of year are provided to the encoder and decoder. Other features which are known only in the past (e.g. weather) are only passed to the encoder layer.

Both encoder and decoder inputs are enriched with static covariates at the "Gated Residual Network (GRN)" and are passed to the "Masked Interpretable Multi Head Attention" mechanism. The output of the attention head is passed to another layer of GRN, then to a dense network before being passed to the loss function.

To implement TFT, we leverage an implementation from Pytorch Forecasting [10], which comes with multiple extensions, e.g. auto tuning, learning rate identification and TensorBoard monitoring.

2.4. Informer

Below, we step through the architecture for Informer, shown visually in Figure 5. The Pytorch implementation in was made public in [14]. We therefore leverage this repository as our testbed for Informer, with various modifications to support visualization and different training strategies (for example, we needed to modify our learning rate decay and regularization strategies from the approach Informer's authors leveraged). We also modified the Informer training codebase to support S, DTO-DRNN.

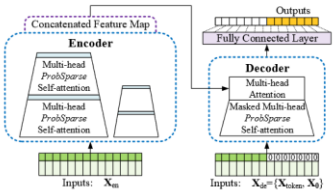

Figure 5. High level overview of informer from [9]

2.4.1 Temporal Embedding layer

In Informer, the input X is passed through a one dimensional convolution layer to obtain a scalar projection vector with a specified dimension d. Next, a positional encoding (cosine and sine) is added to this embedding to preserve local context, and then a global time stamp embedding is added to represent further temporal information, such as week, month, and holiday. This allows Informer to incorporate a learnable representation of context, which might need to be provided as separate features in other approaches, such as S,DTO-DRNN. This allows attention layers deeper in the network to attend on position in sequence and temporally relevant cues (e.g. electricity needs are greater during working hours), which we believe would provide a good replacement for the need to incorporate time specific features in preprocessing.

2.4.2 Encoder: ProbSparse Attention

The generated embeddings from 2.4.1 are then passed through an encoder with a configurable number of attention blocks, each with (uniquely to Informer) sparse attention mechanism and self attention distilling.

While the canonical attention mechanism highlighted in 2.2 allows transformer architectures to more easily learn long term dependencies, it can be subject to large time and memory demands, which limits its ability to predict over larger horizons and large inputs. Therefore, Informer leverages the observation that attention is often sparse; only a few dot-product pairs contribute to most of the attention in any given layer. To clarify this notion, we define sparsity for any given query i as:

$$M(q_i, K) = ln \sum_{\{j=1\}}^{L_k} e^{\frac{q_i k_j^T}{\sqrt{d}}} - \frac{1}{L_K} \sum_{\{j=1\}}^{L_k} \frac{q_i k_j^T}{\sqrt{d}} \quad (2)$$

i.e., the Kullback-Leibler divergence between the probability distribution implied by canonical attention for some query i, and unform attention across all dot-product pairs. In effect, queries with a higher $M(q_i, K)$ value are more likely to be dominant and would be

included when calculating attention. Based on this, we modify Eq. 1 to obtain ProbSparse attention:

$$Attention(Q, K, V) = softmax(\bar{Q}K^T/\sqrt{d})V \quad (3)$$

where $\bar{Q}$ is a sparse matrix of the same size as Q containing only the u-top queries, based on $M(q, K)$. u is set as $u = c * lnL_q$, where c is constant. ProbSparse has logarithmic time and memory complexity with minimal information loss, since for multi-head attention, we generate different sparse pairs per head. There is considerable evidence from [9] that Informer's attention architecture allows for larger encoder/decoder inputs, which leads to better performance.

Following ProbSparse, similarly to the standard Transformer architecture introduced by Vaswani *et al*, we have a residual connection with layer normalization and dropout, followed by a position-wise feed-forward layer with GELU activation and residual connection.

2.4.3 Encoder: Attention Distillation

While the ProbSparse attention mechanism helps enhance Informer's efficiency, it introduces a separate problem, namely, redundant combinations of value V. To account for this, we perform attention distillation by applying a 1-dimensional convolution with ELU activation. Max-pooling with stride 2 is added after convolution to save memory, and the result is output as a feature map.

2.4.4 Generative Style Decoder

Informer leverages a more standard decoder mechanism, where an input embedding, alongside the feature map generated by the encoder through a stack of multi-head attention layers. Each attention layer has masked multi-head attention (Where masked dot products are set to $-\infty$. This ensures positions do not attend to future positions and preserves auto-regressive properties). Unlike Transformer, ProbSparse attention is used here. A fully connected layer acquires the final output and projects to the appropriate dimensions for the desired forecast.

Uniquely to Informer, the decoder input 'start token' is represented as the embedding of some configurable slice of the input sequence (such as the last day before prediction), and then concatenated with a placeholder vector of the same length as the target sequence (with all members equal to 0). This way, the decoder predicts an output sequence in one pass, as opposed to a less efficient dynamic decoding style, as is often done in encoder-decoder architectures.

3. Experimental Setup and Results

3.1. Training Scheme

For our experiments, we leverage the first 80% of the data set for training, as mentioned in [4]. The last S_0 hours of the data are used as a testing set, with the remaining data used as a validation set. This choice is different from Taheri *et. al*, but we wanted a measure of performance on unseen data, as the validation set can leak into a model via tuning. All data is standard scaled, including targets, and mean squared error (MSE) is used as the loss function across all models. As mentioned in section 1, we experiment with two variants of the data, one with air pressure, temperature, humidity, wind speed and solar irradiation, and another that also includes temporal features. To report performance on the test sequence, we use Root Mean Square Error (RSME) at scaled values. Although some of the models we explored support multivariate forecasts, we set univariate targets for heat and electricity to compare with [4].

$$RMSE = \frac{\sum_{t=0}^{T}\sqrt{(y_t - \hat{y_t})^2}}{T} \quad (4)$$

Our experiments include model modification, hyper parameter tuning, as well as performing predictions at 1, 3, and 7 day intervals. We also experimented with 48 (two days) and 336 hour (two weeks) horizons for Informer. Experiments were performed on a GeForce 3070 GPU. More information w.r.t. hyperparameter exploration, and our choices for our implementation of [4] can be found in appendix D.

3.2. Transformers vs. Baselines

We compare Transformer, TFT, Informer, and our implementation of [4] to our baseline of ARIMA, as a means of measuring improvement over classical and recurrent frameworks with transformer architectures. Table 1 shows the validation loss MSE of our best performing models for each target over 1, 3, and 7 days. Immediately, we notice that each of our neural architectures outperforms ARIMA, and that they smoothly lose accuracy as the forecast horizon increases. Furthermore, S, DTO-DRNN performs well on both targets, in line with results from [4]. TFT also performs particularly well on heat data. However, Informer outperforms the others, across all horizons. Further selected experiments can be found in Table 2.

Model + Prediction		Electricity	Heat
ARIMA	1 day	3.23	2.04
	3 days	10.23	13.91
	7 days	1.72	2.53
S, DTO-DRNN	1 days	0.06	0.07
	3 days	0.13	0.15
	7 days	0.18	0.2
Transformer	1 days	0.3	0.24
	3 days	0.52	0.42
	7 days	0.54	0.63
TFT	1 days	0.28	0.2
	3 days	0.26	0.23
	7 days	0.5	0.27
Informer	1 days	**0.05**	**0.05**
	3 days	**0.07**	**0.06**
	7 days	**0.09**	**0.1**

Table 1. Best Validation MSE for each model.

Below, we explore further observations from our results.

3.3. Parameter Sensitivity

In general, our models were very sensitive to parameterization. As an initial example, many of our models did better with higher learning rates for

Model	Dataset	Prediction Length	Batch Size	Epochs	Initial Learning Rate	Include Time Features	Dropout	Optimizer	Number of Attention Heads	Sequence Input Length	Training MSE	Validation MSE	Test RMSE
S, DTO-DRNN	Heat	24	64	10	0.001	TRUE	0.1	Adam	NA	168	0.28	0.23	0.43
S, DTO-DRNN	Electricity	24	64	40	0.001	FALSE	0.1	Adam	NA	168	0.23	0.18	0.24
S, DTO-DRNN	Electricity	72	64	40	0.001	TRUE	0.1	Adam	NA	240	0.19	0.07	0.32
Transformer	Electricity	72	128	10	0.001	FALSE	0.1	Adam	4	128	0.16	0.5	0.15
Transformer	Electricity	72	128	10	0.001	FALSE	0.1	Adam	3	128	0.27	0.6	0.17
Transformer	Heat	72	128	10	0.001	FALSE	0.1	Adam	2	128	0.18	0.62	0.34
TFT	Heat	48	32	10	0.02	TRUE	0.12	Adam	3	144	0.22	1.15	0.81
TFT	Heat	48	128	10	0.02	TRUE	0.12	Adam	3	144	0.13	0.36	0.16
TFT	Electricity	72	128	10	0.02	TRUE	0.5	Adam	1	144	0.13	0.26	0.38
TFT	Electricity	72	128	10	0.02	FALSE	0.5	Adam	1	144	0.54	0.43	0.46
Informer	Heat	72	32	10	0.0001	FALSE	0.05	Adam	8	144	0.01	0.06	0.19
Informer	Heat	168	32	10	0.0001	FALSE	0.1	Adam	8	168	0.03	0.07	0.26
Informer	Electricity	168	32	10	0.0003	FALSE	0.2	Adam	8	336	0.01	0.09	0.15

Table 2: Selected Experiment Result

electricity than heat. For example, Informer required a learning rate of 0.0003 for electricity vs 0.0001 for heat. We suspect this, as well as other differences between the two targets was due to the differing seasonal patterns between electricity and heat, where electricity needed larger gradient steps to avoid focusing too strongly on seasonality, which could be viewed as a local minimum. As an example, we note that S, DTO-DRNN, while doing well on the validation set, struggles at test time precisely because it focuses too heavily on seasonal patterns, and therefore experiences degrading test performance beyond 24 hours, see appendix C for more detail.

Similarly, depending on the prediction length, our transformers would perform worse holding capacity fixed, and required higher capacity to appropriately predict. For example, for TFT, both targets performed better at 24-hour forecasting using 24 hidden sizes in the continuous layer, while for longer periods, increasing the size to 48 contributed to better performance.

For the canonical transformer, increasing the number of attention heads in conjunction with the number of encoder and decoder layers had a significant positive impact on performance, as can be seen in Figure 6.

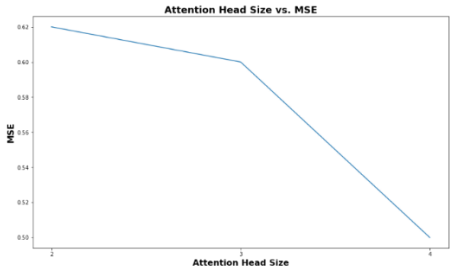

Figure 6. Transformer validation MSE reduces as we increased number of attention heads

Similarly, figure 7 shows how, given larger inputs to the encoder, Informer performs significantly better, even as the prediction length increases.

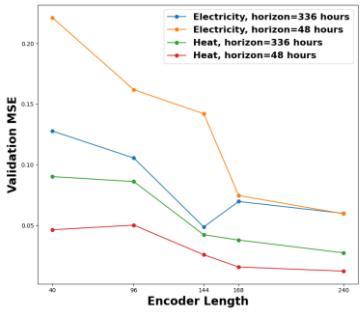

Figure 7. Informer sensitivity to encoder length

3.4. The Importance of Attention

Given the inherent seasonality in our dataset, the construction of attention maps is very important to ideal forecasting. If our transformers attend the wrong points in time, this can lead to significantly worse predictions. To test this assertion, and potentially explain Informer's strong performance. we performed an ablation study on Informer by replacing its ProbSparse + attention distillation blocks with canonical attention layers, then predicting on heat data with increasing encoder input lengths. The results are in Table 3, which shows that MSE near doubles with canonical attention, tracking closely to the results we obtained from our experiments on Transformer. Sparse attention also allows for larger inputs, and better performance. This observation tracks closely with results from [9].

On another note, we observe that in all cases, the difference between training and validation loss decreases by 50% when using ProbSparse attention, which suggests that sparse attention may have a regularizing effect, similarly to droput. From Table 2, we notice that Informer tends to overfit more strongly than any of the other architectures even as dropout increases, but ProbSparse helps mitigate this.

Prediction Length	2 days				2 weeks			
Encoder Input (hours)	168	240	336	400	168	240	336	400
Informer	0.0746	0.0596	0.0442	0.0323	0.0487	0.0599	0.0456	0.034
Informer-α	0.2448	0.17535	0.1435	-	0.1822	0.1534	-	-

Table 3. Informer ablation study on validation MSE, Informer-α represents Informer without ProbSparse or attention distillation, i.e. canonical transformer. '–'=OOM

The success of sparse attention in this context, may also explain the struggles TFT faces in prediction. To examine this, we note that, from Figures 14 and 15, TFT appears to make lagged predictions, even though their period and amplitude is very close to the actual result.

As a potential explanation, we turn to Figure 8, which shows TFT's attention paid across time. We notice that while recent data is attended to strongly, TFT also pays close attention to data 100-120 hours before the prediction sequence. Given that sparse attention was successful for Informer, we suspect that TFT's more spread attention may be causing these less ideal predictions.

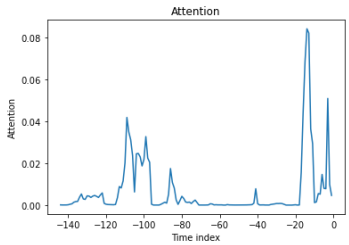

Figure 8. TFT Temporal Attention Visualization (Heat forecast).

3.5. Temporal Sensitivity

Finally, we note the impact of temporal features on our model performance. This point is particularly important, given the attention Taheri *et al.* [4] pay to their selection of temporal features as a component of their architecture's success. Our results align with this understanding, without temporal features, validation MSE and test RMSE were both 0.1 higher, for both heat and electricity for S, DTO-DRNN, than when temporal features were included.

We notice similar, yet more complicated behavior with TFT. In this case, two components (month and year) were found to play a critical role in model performance. When neglecting to include these features, TFT performs poorly for electricity prediction (0.38 to 0.46 RSME). However, for heat demand, adding these features decreases model efficiency. Indeed, Figure 9 shows that TFT directly places high importance on temporal variables, like the month and hour.

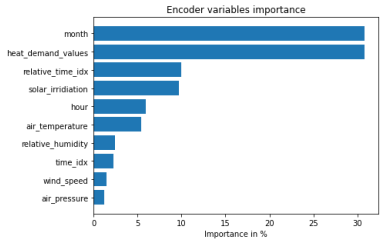

Figure 9. TFT Encoder Variable Importance (Electricity forecast). Adding time features improves model accuracy.

We believe this behavior is likely caused for similar reasons as Informer doing better with higher learning rates on electricity. Since electricity demand follows a more predictable cyclic shape, time is a relevant feature, and our largest concern is avoiding overfitting to seasonality. However, this is not as obvious with heat demand, given that series' stronger variance, and therefore focusing on time more often leads to overfitting.

3.6. Comparison with Prior Art

Given our results above, particularly w.r.t. to Informer, one may wonder how our work compares to current state of the art on this problem. To address this, we took our best performing model (Informer), our S, DTO-DRNN implementation and compared these results with [4], which can be found in Table 1. While we cannot fully confirm this, due to the lack of an explicit implementation of [4], we suspect that, given this architecture performed closest to Taheri *et al*'s results at a 24-hour prediction length, we assume that this was the prediction length used in [4].

Therefore, we compare our implementation and Informer trained on the same metric used in that paper, root mean squared error (RMSE), divided by the root mean square of training data for the target series (RMSTD). We notice that our implementation of [4] is competitive with the original results, and even does slightly better for the heat dataset. But Informer shines brightest, performing slightly better for electricity, and ~50% better for heat forecasting, which is encouraging for Informer's abilities in this space.

Model	Electricity	Heat
S, DTO-DRNN [4]	21.50%	23.60%
S, DTO-DRNN-Ours	24.45%	23.34%
Informer	**19.80%**	**12.50%**

Table 4. RMSE/RMSTD Comparison

4. Conclusion

In conclusion, we present evidence showing that transformers, particularly with sparse attention mechanisms, are capable of outperforming baselines, and even reaching state-of-the-art performance for energy forecasting problems.

With future work, we aim to experiment further on this problem, leveraging the insights we gained from Informer in particular. In particular, we would like to explore and potentially develop new methods to generate sparse attention maps, in the vein of work by Klimek *et al.* [13] and others.

5. Work Division

Our team's contribution summary can be found in Table 7, in Appendix G.

6. References

[1] Lim, B., Arik, S. O., Loeff, N., and Pfister, T., "Temporal Fusion Transformers for Interpretable Multi-horizon Time Series Forecasting", *arXiv e-prints*, 2019.

[2] Borovykh A, Bohte S, Oosterlee CW. Conditional Time Series Forecasting with Convolutional Neural Networks. arXiv e-prints. 2017;p. arXiv:1703.04691.

[3] Wang Y, Smola A, Maddix D, Gasthaus J, Foster D, Januschowski T. Deep Factors for Forecasting. In: Proceedings of the International Conference on Machine Learning (ICML); 2019.

[4] Taheri, S., Jooshaki, M. and Moeini-Aghtaie, M., 2021. Long-term planning of integrated local energy systems using deep learning algorithms. *International Journal of Electrical Power & Energy Systems*, *129*, p.106855.

[5] Rosato A, Araneo R, Andreotti A, Panella M (2019) 2-D Convolutional Deep Neural Network for Multivariate Energy Time Series Prediction. In: 2019 IEEE International Conference on Environment and Electrical Engineering and 2019 IEEE Industrial and Commercial Power Systems Europe (EEEIC / I&CPS Europe); 2019 11–14 June

[6] Kim TY, Cho SB (2019a) Predicting residential energy consumption using CNN-LSTM neural networks. Energy.182:72–81. https://doi.org/10.1016/j.energy.2019.05.230

[7] Vanting, N.B., Ma, Z. & Jørgensen, B.N. A scoping review of deep neural networks for electric load forecasting. *Energy Inform* **4,** 49 (2021). https://doi.org/10.1186/s42162-021An-00148-6

[8] Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, et al. Attention is All you Need. In: Advances in Neural Information Processing Systems (NIPS); 2017.

[9] Zhou, H., Zhang, S., Peng, J., Zhang, S., Li, J., Xiong, H. and Zhang, W., 2021, May. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of AAAI*.

[10] Pytorch Forecasting https://pytorch-forecasting.readthedocs.io/en/latest/index.html

[11] Optuna - A hyperparameter optimization framework

[12] 8 years of hourly heat and electricity demand for a residential building | IEEE DataPort (ieee-dataport.org)

[13] Klimek, J., Klimek, J., Kraskiewicz, W. and Topolewski, M., 2021. Long-term series forecasting with Query Selector--efficient model of sparse attention. arXiv preprint arXiv:2107.08687.

[14] https://github.com/zhouhaoyi/Informer2020

[15] https://github.com/moraieu/query-selector-classification/tree/master/query_selector

7. Appendix

A. Project Code Repository

All code for this project may be found at https://github.gatech.edu/etffall2021dl/project_work. Implementations for Transformer, TFT, and Informer mostly come from [10, 14, 15] as an initial starting point. We've made the following change, using those code-bases as a starting point.

- Added notebooks to facilitate training on ARIMA/TFT baselines, exploratory data analysis, and preprocessing
- Implemented S, DTO-DRNN [4], from scratch, and modified Informer's training logic to also support this architecture as implemented in [4] (namely adding support for L2 regularization and gradient clipping).
- Modified Informer/TFT training to record RMSE/mean squared average of loss
- Added extra dropout layers to the implementation from [15].

B. Dataset Information

Table 5 summarizes the data set, where electricity/heat demand values are the variables in interest where forecast was conducted.

Column	Unit	Non-Null Row Count	Description
Time	Datetime	70080	One Hour Aggregation
air_pressure	mmHg	69934	Outside Air Pressure
air_temperature	^{O}C	69903	Outside Air Temperature
relative_humidity	%	69903	Outside Air Relative Humidity
wind_speed	m/s	69125	Wind Speed
solar_irridiation	W/m^2	70080	Power Received from the Sun
total_cloud_cover	categorical	69837	0-10 categorical
electricity_demand_values	kW	70073	HVAC System, convenience power, elevator, etc
heat_demand_values	kW	70073	Household Heating Load

Table 5. Dataset Description

A correlation matrix of the dataset is shown in Figure 9 below. We note that none of the features alone shows significant correlation to the target values, except for solar irradiation to electricity demand.

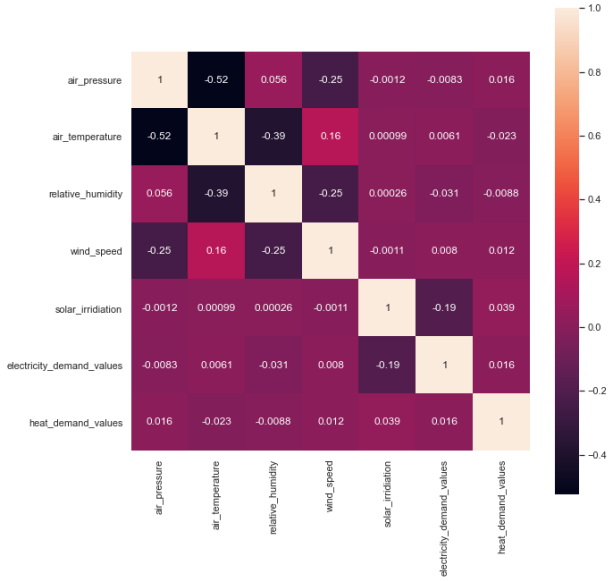

Figure 10. Correlation Matrix

Principal component analysis can be seen in Figure 11, where air pressure, air temperature and relative humidity explains most of the variance in the data.

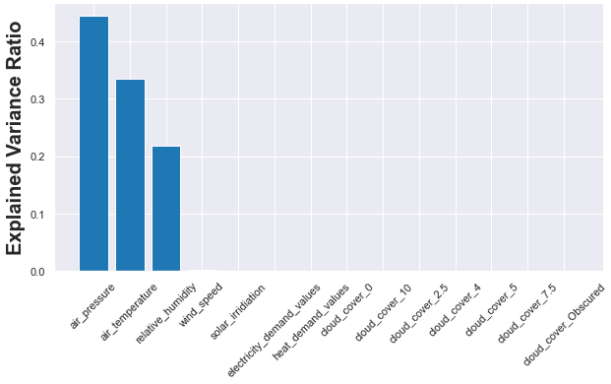

Figure 11. Principal Component Analysis

C. Example Predictions

All predictions shown below were taken at for a 72 hour (3 day) sequence, for both electricity and heat.

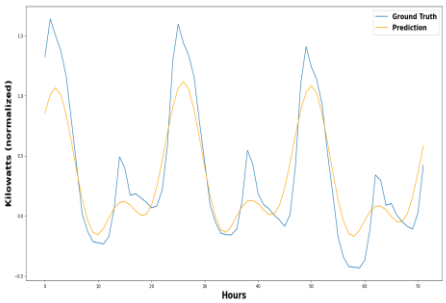

Figure 12. S, DTO-DRNN, Electricity Demand Values, 72-hour forecast. RMSE 0.23

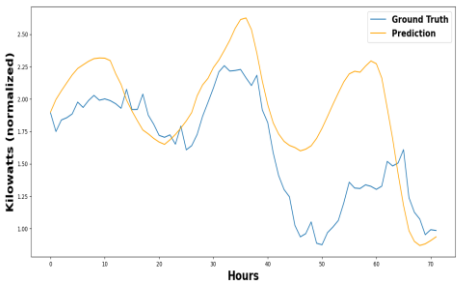

Figure 13. S, DTO-DRNN, Heat Demand Values, 72-hour forecast. RMSE 0.43

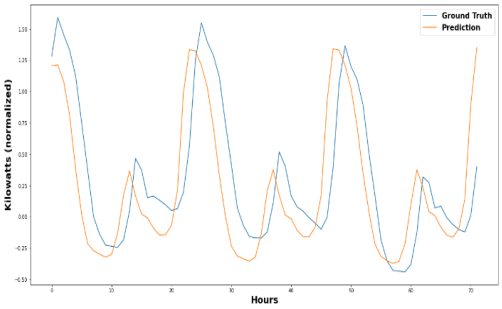

Figure 14. TFT, Electricity Demand Values, 72-hour forecast. RMSE 0.38.

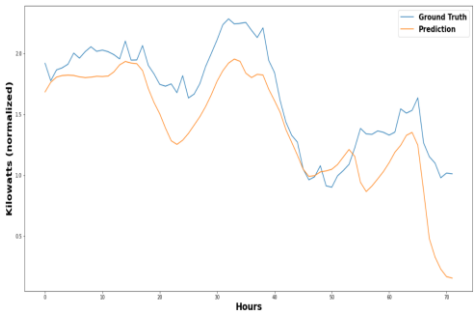

Figure 15. TFT, Heat Demand Values, 72-hour forecast. RSME 0.33.

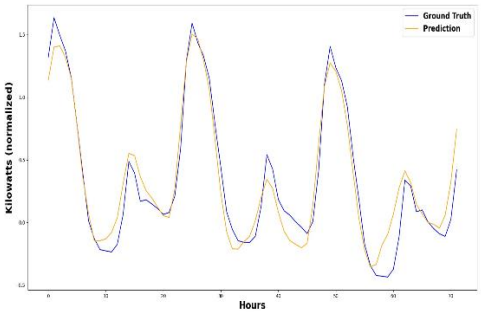

Figure 16. Transformer, Electricity Demand Values, 72-hour forecast. RMSE 0.15

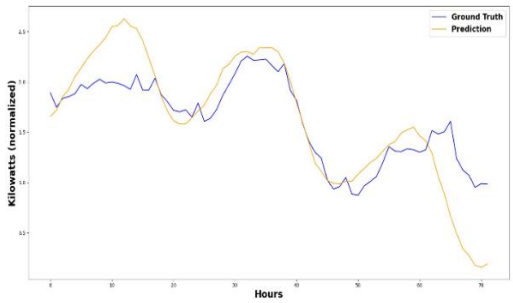

Figure 17. Transformer, Heat Demand Values, 72-hour forecast. RMSE 0.34

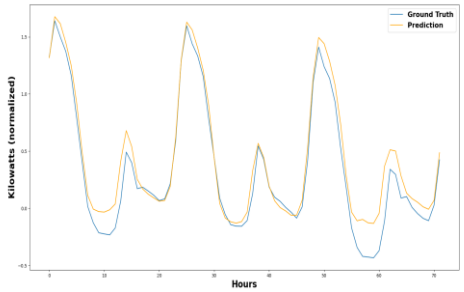

Figure 18. Informer, Electricity Demand Values, 72-hour forecast. RMSE 0.14

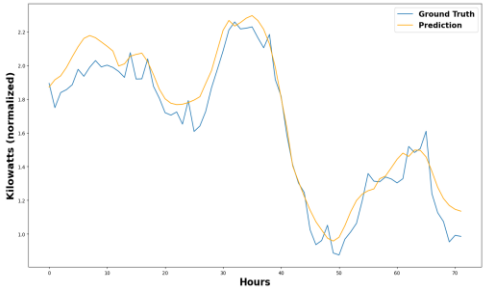

Figure 19. Informer, Heat Demand Values, 72-hour forecast. RMSE 0.1

D. Hyper-Parameter Tuning Range.

Here, we present a list of tuning decisions shared by all models, then highlight explicit tuning choices for specific models.

Optimizer	Adam
Learning rate	0.0001 - 2
Epochs	[10 , 20 … , 100]
Batch Size	[32, 64, 128]
Dropout	[0.1, 0.2, … 0.5]
Encoder Input Length	[48, 96, 144, 168, 240, 330, 440]
Decoder Input Length	[24, 48, 96, 312]
Encoder Layers	[1, 2, 3, 4, 8]
Decoder Layers	[1, 2, 3, 4]
Prediction Lengths	[24, 72, 168]
Number of Attention Heads	[1, 2, 3, 4, 8]
Add Temporal One Hot Features	[Yes, No]

Table 6. Hyper Parameter Ranges Shared Across Architectures. X – Y specifies that a continuous range of values were chosen between X and Y

For S, DTO-DRNN, we also tested numbers of LSTM layers between 1 and 4, while holding this value at 2 for TFT. For LSTM, to replicate [4] as closely as possible, we chose gradient clipping of 10, and an L2 weight decay of 0.0001. For TFT, given that its architecture supports various options like including temporally static variables, target scales (i.e. mean and standard deviation of the target variable), and variables known in the future (e.g. time index), we also experimented with the inclusion of these different variables.

Further, for TFT, hyperparameter tuning is performed using 100 trials via Optuna [11] for each target individually, which is built into pytorch-forecasting.

The Transformer performed well when hidden size was set to 312 for electricity as the target. Whereas, with heat as the target the lower values of hidden size (83-96) resulted in better performance. The learning rate was held constant at 0.001.

Finally, for Informer, we deviate from the learning rate decay strategy specified in [9], instead decaying by 0.1* lr after each 7^{th} and 9^{th} epoch. We found the originally proposed learning rate decay strategy kept Informer stuck in local minima and keeping the learning rate higher for longer allowed the network to fully explore the contours of the optimization problem. We suspect this may have to do with the nature of the seasonality behind our datasets, since we needed a higher learning rate for electricity, which has a stronger seasonal pattern.

Finally, each experiment was run 5 times, with the best score across all experiments taken for a particular setting.

E. TFT Variable Importance

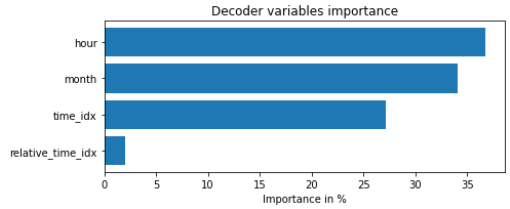

Figure 20. TFT Decoder Variable Importance (Electricity forecast)

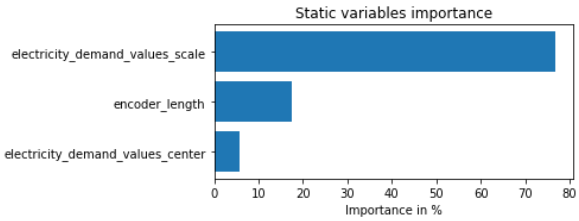

Figure 21. TFT Static Variable Importance (Electricity forecast)

F. Prob Sparse vs Canonical Attention Maps.

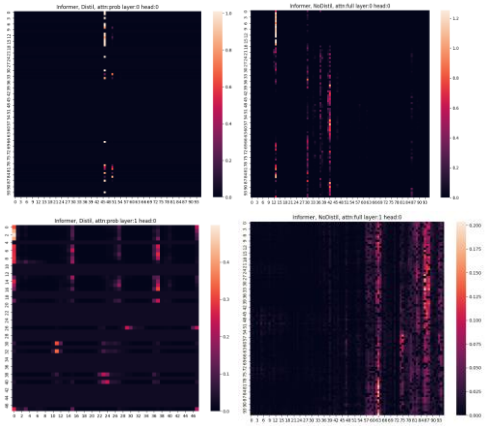

Figure 22. Encoder Attention Maps (Heat) The left maps represent Informer, the right maps are canoncial. From top to bottom, we show maps from deeper in a 2 layer encoder. We note that ProbSparse provides sparser attention maps at each layer, compared with the canonical representation.

G. Team Member Contributions

Student	Contributed Aspects	Details
Gregory Loshkajian	Informer, Taheri reimplementation	1. modified informer training code to support our training scheme with split, weight decay, clipping, Taheri performance metric calculation (see exp_informer.py, data_loader.py, main_informer.py) in repo section 3.6 in report) 2. added S, DTO-DRNN model from scratch in Pytorch (see lstm_paper.py). 3. Performed parameter analysis on Informer + S, DTO-DRNN along with ablation studies on Informer, see section 3. 4. Reviewed papers for informer and DTO-DRNN, particularly to understand how to reimplement the latter. (sections 2.1, 2.4). 5. Extended informer codebase with visualization code for prediction plotting (appendix C). 6. Wrote logic for attention visualization plots to show difference between canoncial and probsparse attention (see section 3.4)
Nurettin Mert Ersöz	Dataset, EDA, TFT	1. Identified dataset and corresponding paper, applicable for time series forecasting (see section 1) 2. Performed EDA to discover insights from the data such as features of importance and seasonality (see section 1, section 3 appendix B, E) 3. Modelled using TFT and performed experiments and reported interpretability features (see sections 2.2 and 3, tft notebooks in notebooks folder).
Nilesh Domaida	ARIMA, Transformer	1. Build the base line ARIMA model from scratch for forecasting heat and electric demand. Experimented various models like pmd Autoarima, ARIMA from stats model. Performed pre-processing and hyper parameter tuning. section 3.2 2 Build the base transfomer by incorporating components including Attention block, position embedding scheme (Vaswani et. al)from the query_selector code and modified Encoder-decoder by adding layers like dropouts. Also enhanced training to support train and split of the data into training, validation and testing data, (see train.py, data_loader.py) in the repo) 3. Performed parameter analysis and tuning on the Transformer. see section 3. 4. Reviewed papers of Vaswani et al (attention is all you need) and query_selector to understand how the Transformer blocks work. (sections 2.2). 5. Added visualization for transfomer including prediction plots (appendix C) and attention vs MSE (appendix D)
All	Writing the Report, Code Reviews	Multiple meetings each week, paper reviews, code reviews, etc.

Table 7. Contributions of team members.