Predicting the Energy Behavior of Prosumers Using Time-Series Transformers

Joosep Hook
University of Tartu
Institute of Computer Science
joosep.hook@ut.ee

Kea Kohv
University of Tartu
Institute of Computer Science
kea.kohv@ut.ee

Abstract

This project investigates the application of time-series transformers to forecasting the energy behavior of prosumers. Prosumers are individuals who both consume and produce energy, in this case solar energy. The project uses the dataset of a Kaggle competition organized by Enefit, an energy company in the Baltics. The hourly consumption and production predictions are made for specific segments in 48h increments. The evaluation is based on the Mean Absolute Error. The features include the forecasted weather data, gas and electricity prices, and prosumer-specific information. The project compares the Vanilla Transformer as well as modified time-series transformers - the Informer, the Autoformer, the Patch Time-Series Transformer, and the Temporal Fusion Transformer, against non-transformers. In experiments with NeuralForecast models, the Temporal Fusion Transformer achieves the best performance. We find that models that support past and future covariates have an advantage over the models that do not and that optimizing transformers' hyperparameters, including the dropout rate, learning rate, and the scaler type, can help improve the accuracy of forecasts but may also risk overfitting the models.

1 Introduction

We use time-series transformers to forecast prosumers' energy consumption and production. More specifically, we predict the amount of electricity produced and consumed by Enefit customers with solar panels installed. This prediction task is part of a Kaggle competition[1] organized by Enefit, a prominent energy company operating in the Baltic region.

Prosumers produce energy by using renewable sources such as sunlight, wind, or water. The produced energy can be self-consumed by prosumers or sold into the national power grid. The number of prosumers is on the rise, and assuming this trend continues, by 2050 almost half of the total renewable electricity in the European Union will be created by prosumers. Inaccurate estimation of prosumers' energy consumption can result in higher operating expenses (overproducing energy), unstable grid conditions (underproducing energy), and wasteful use of energy resources.[2]

In the Kaggle competition, the forecast target is the amount of energy produced and consumed hourly in different market segments. Segments are unique combinations of counties (15 Estonian counties), business / private customer indicators (Yes / No), and different product types (Combined / Fixed / General service / Spot). New segments may appear in the future and current segments may be removed or deprecated. Forecasts are made daily for the following 48 hours of which the last 24 hours are scored on the competition's hidden test set.

In the competition dataset, the features include the electricity produced and consumed hourly by the segment, the forecasted and measured weather data for that hour, gas and electricity prices, and information about the prosumers, such as their energy production capability (photovoltaic capacity). The competition uses Mean Absolute Error to evaluate submissions. The competition started on November 1, 2023, and ends on January 31, 2024, after which there is a 3-month evaluation period where submissions are evaluated against future data.

There are many approaches to energy consumption and production forecasting, such as artificial neural networks, support vector machines, and tree-based models (Bouquet et al., 2023). Our focus is on using transformers (Vaswani et al., 2017) to forecast solar energy consumption and production.

Energy consumption and production forecasting

[1] Predict energy behavior of prosumers. Kaggle

[2] Prosumers: when energy consumers become energy producers.

involves long sequences which require the forecasting model to have long-range alignment ability and efficient operations on long sequence inputs and outputs (Zhou et al., 2023). Transformers have shown promise in capturing long-range dependencies (Zhou et al., 2023). But, when applied to long-sequence time-series, the Vanilla Transformer model is limited by the attention mechanism's quadratic time complexity and high memory usage (Zhou et al., 2021). To overcome these issues, several modified transformer models have been proposed, such as the Temporal Fusion Transformer, the Patch Time-Series Transformer, the Autoformer, and the Informer, all of which we use in our experiments.

The original transformer (Vaswani et al., 2017) model was designed for natural language processing tasks. Transformers adapted for time-series tasks also need to take into account additional features as well as distinguish the trend cycle and seasonal variations from noise in the time series[3]. For example, recurrent layers can be used to capture temporal relationships and gating mechanisms can help distinguish important covariates that explain the target (Lim et al., 2021).

In time-series forecasting, only relying on the target's past values may not be enough. Additional information from static, past and future covariates can help predict the target's future values. Past covariates are features known only into the past (e.g. historic weather), future covariates are features known into the future (e.g., national holidays), and static covariates are features that remain constant over time (e.g., county id)[4]. The production of solar energy is reliant on weather conditions, consumption of energy is related to national holidays, therefore it can be assumed that using future covariates in the model leads to more accurate predictions.

Transformer models that predict one target (univariate) have shown better performance than models predicting many targets (multivariate) (Murphy and Chen, 2023). In addition, univariate models have been observed to converge faster, whereas the multivariate model might need more of data (Murphy and Chen, 2023).

We conduct experiments to assess the following:

1. which model achieves the lowest MAE on the test set,
2. whether models that support past and future covariates outperform models that only rely on the target's past values and time features,
3. whether we are able to improve the models' performance with hyperparameter tuning,
4. whether having two separate models for production and consumption prediction is better that having one model predict both.

We find that the Temporal Fusion Transformer (Lim et al., 2021) achieves the lowest production, consumption as well as overall MAE on the test set outperforming other transformers and selected non-transformers from the Neuralforecast library. In our experiments, models that support past and future covariates have an advantage over models that do not. Hyperparameter optimization, which was conducted for a selection of the models, improved the results for some models but caused overfitting on validation set for some. In experiments with univariate versus multivariate model which we trained on only one segment, we saw better results from the multivariate model. [5]

2 Related work

The effectiveness of using transformers for time-series forecasting has been questioned. It has been argued that in time-series forecasting the sequence order plays an important role but the transformer self-attention mechanism is inherently permutation-invariant resulting in temporal information loss (Zeng et al., 2023). In a study where the Informer model's self-attention layers were gradually replaced with linear layers, the performance of the model grew with simplification - showing that adding self-attention layers worsened the performance (Zeng et al., 2023). However, the conclusions of this study were challenged in an experiment where the Autoformer was compared against an equally sized linear model and it outperformed the linear model[6].

Utilizing correlation information among subsequences can be challenging for linear models. The self-attention mechanism can capture correlation between tokens and,in a time-series forecasting it has also been extended to channels that carry information about the exogenous variables, i.e. features that help explain the target (Xue et al., 2023).

[3]Autoformer. Huggingface
[4]Covariates. Darts

[5]Link to GitHub
[6]Yes, Transformers are Effective for Time Series Forecasting (+ Autoformer)

The Vanilla Transformer, the Informer as well as various non-transformer models have been compared on a wide variety of time-series forecasting tasks (Godahewa et al., 2021). In an hourly electricity consumption prediction task, several models outperformed the transformer-based models (Godahewa et al., 2021). The best performing model in hourly electricity consumption prediction was WaveNet which is a deep convolutional network adapted for time-series forecasting (Borovykh et al., 2017).

One of the challenges of training transformers is the model size (Ahmed et al., 2023). Training a large model generally requires a high-memory GPU. The training set needs to be large to avoid overfitting (Ahmed et al., 2023). Deep Learning models like transformers have many tunable hyperparameters, such as learning rate, which affect how well the model learns and performs (Ahmed et al., 2023). For this reason, hyperparameter tuning is recommended to get the best performing model. High noise levels in time-series data can also lead to transformers overfitting (Xue et al., 2023).

In Kaggle competitions where the datasets are on the smaller side, tree-based boosting algorithms such as XGBoost are a popular choice. The majority of the public notebooks of this Kaggle competition use tree-based ensembles and no transformer-based models have been published in this competition.

3 Methods

The joining of data and feature generation are based on public Kaggle notebooks [7]. Most of the data processing in Kaggle notebooks is focused on creating features suitable for non-time-series models like (XGBoost) regression. We adapt the data processing for time-series models, including the removal of missing data. However, in feature generation we do not create lagged variables because the time-series models take care of learning temporal relationships in the data. We generate a feature for Estonian holidays - whether a particular timestamp falls on a national holiday.

On a high level, the file containing the measurements (train.csv) is used as a reference table onto which information from auxiliary tables (gas prices, electricity, forecast weather, historical weather) is added. For each row in the measurements file, we know the weather forecast and actual weather information for that date and time with 1 hour granularity. Electricity price forecasts are also known with 1 hour granularity, whereas gas price forecasts are available for whole days.

There are no explicit identifiers that uniquely determine a time series, but we can uniquely determine a data generating process using a composite key of (county, is_business, product type, is_consumption). In the Kaggle competition, the hourly production and consumption predictions need to be made for each segment.

To remove observations with missing data, we only keep measurements with timestamps from 2022-11-03 00:00:00 up to 2023-05-24 23:00:00 inclusive. Within this time frame there are 62 different segments. This leaves us with 4872 observations per segment. Within this time period, if any data is missing, then we interpolate the missing data using Pandas' time-based interpolation method. After interpolation we're left with a little over 300k measurements for training. Out of this 4872, we keep the last 10 % for validation and evaluation (5 % + 5 %). The prediction horizon is fixed at 48 (hours) and the lookback window is 168 (1 week in hours) for default settings. However, during hyperparameter tuning we also tune the size of the lookback window. The features are not scaled by default, unless some model's implementation has a default feature scaling method or we are doing hyperparameter optimization.

We compare 5 transformer models and 4 non-transformer models. We use the NeuralForecast library's (Olivares et al., 2022) implementations of the Vanilla Transformer (Vaswani et al., 2017), the Informer (Zhou et al., 2021), the Autoformer (Wu et al., 2021), the Patch Time-Series Transformer (PatchTST) (Nie et al., 2022) and the Temporal Fusion Transformer (TFT) (Lim et al., 2021). Of these models only the implementation of the Temporal Fusion Transformer supports static, past and future covariates. In addition, we use the Huggingface transformer library's (Wolf et al., 2019) implementation of the Vanilla Transformer for time-series called Time Series Transformer.

The Informer uses a generative-style decoder, a KL-divergence based self-attention mechanism called ProbSparse, and halves the cascading layer input in self-attention (Zhou et al., 2021). The Autoformer uses an auto-correlation mechanism to discover dependencies and aggregate information

[7] Enefit: Object Oriented GBDT, Enefit | Darts Study Notes, Enefit Generic Notebook

at the series level (Wu et al., 2021). The Patch Time-Series Transformer segments time series windows into subseries-level patches and uses channel-independence where each input token only contains information from a single channel/time-series (Nie et al., 2022). Prior to the Patch Time-Series Transformer, the transformers' point-wise self-attention treated each timestamp like a token. However, a single timestamp does not carry a lot of information and made transformers prone to overfitting. In Patch Time-Series Transformer, each time series has its own attention weights. In addition, instead of treating each timestamp as an input token, patches of timestampes together form input tokens.[8]. The Temporal Fusion Transformer uses gated residual networks to select relevant variables and to improve generalization ability (Lim et al., 2021). The model captures local dependencies with a sequence-to-sequence layer and long-term dependencies with a multi-head attention block.

The Vanilla Transformer, also from the NeuralForecast library[9] follows the implementation of the model as a baseline from the Informer paper. It uses absolute positional embeddings obtained from calendar features. The window-relative positional embeddings are derived from harmonic functions. It encodes autoregressive features using a convolutional neural network. The embedding-creation strategy is the same for the Informer and the Autoformer. The Temporal Fusion Transformer uses sequence-to-sequence layers to create embeddings as opposed to standard positional encoding.

As non-transformers, we include the Darts library's (Herzen et al., 2022) implementation of Time-series Dense Encoder (TiDE) (Das et al., 2023), and the NeuralForecast's implementations of the Multi Layer Perceptron (MLP) (Rosenblatt, 1958), Neural Hierarchical Interpolation for Time Series (NHITS) (Zhou et al., 2020), and TimesNet (Wu et al., 2023). TimesNET is a CNN-based model which transforms time series from 1 dimensional into 2 dimensional space. It has a convolutional Inception block for capturing temporal variations[10]. NHITS is composed of several MLPs with ReLU non-linearities[11]. It also uses multi-rate input pooling, hierarchical interpolation and backcast residual connections. TiDE is a transformer-like model without the attention mechanism. It encodes the time-series with covariates using dense MLPs and then decodes the time-series together with future covariates also using dense MLPs. TiDE is 5-10x faster than transformer models that use attention (Herzen et al., 2022).

[8]Medium. The Return of the Fallen: Transformers for Forecasting

[9]Vanilla Transformer

[10]TimesNET. NeuralForecast

[11]NHITS. NeuralForecast

Model	**Static**	**Historic**	**Future**
TFT	YES	YES	YES
PatchTST	NO	NO	NO
NHITS	YES	YES	YES
MLP	YES	YES	YES
TimesNet	NO	NO	YES
Vanilla	NO	NO	NO
Informer	NO	NO	NO
Autoformer	NO	NO	NO

Table 1: Support for exogenous variables in NeuralForecast. Static, Historic and Future represent the covariates referenced in NeuralForecast code.

The models from the NeuralForecast library can only be used in a univariate setting, meaning we have to create one model for predicting production and the other for predicting consumption. In addition, not all models support static, past and future covariates as can be seen from Table1. In experiments, we provide static, past and future covariates to the models that support them.

We train each model once on the training set and use the cross-validation function[12] from the Neuralforecast library to perform rolled predictions on the test set meaning that the test set is divided into windows that are moved using step size 1. The models were fitted using default hyperparameters, including the maximum number of steps. We used early stopping with patience 3.

For hyperparameter optimization, we use the Optuna library (Wolf et al., 2019). The hyperparameters were optimized for the vanilla transformer, Temporal Fusion Transformer, PatchTST and MLP. We used Optuna's default sampler, the TPE (Tree-structured Parzen Estimator) algorithm which fits one Gaussian Mixture Model (GMM) $l(x)$ to the set of parameter values associated with the best objective values, and another GMM $g(x)$ to the remaining parameter values. It chooses the parameter value x that maximizes the ratio $\frac{l(x)}{g(x)}$[13]. We use 20 trials per study which is the suggested minimum[14].

[12]Cross-validation. NeuralForecast

[13]TPESampler. Optuna

[14]Hyperparameter Optimization. NeuralForecast

To compare the univariate vs multivariate setting, we use the transformer-like model TiDE from the Darts library. In both settings we first fit the model using the default hyperparameters, then optimize the hyperparameters with Optuna, refit the model on the test set and compare test MAE. Only one segment was used in this experiment.

For a Kaggle submission we constructed an ensemble of TiDE models with Darts. We divided the dataset into time series per each county and fitted the multivariate TiDE model on a list of time series with segments from that county. The submission model is trained on a longer dataset, from 2022-01-01 00:00:00 to 2023-05-24 23:00:00 inclusive as the model is easily able to handle missing datetimes. We submitting the ensemble trained with default hyperparameters as well as trained with optimal hyperparameters found with Optuna on one segment. As the model is not able to make predictions for segments not seen in training but which have appeared in the hidden test data, we fill these values with the predicted target mean for that day.

We also use Vanilla Transformers from HuggingFace as they support both future covariates and training multivariate forecasting models. This is useful for investigating the effect of having 2 separate vs 1 single model.

Model	Prod	Cons	Ovr	#
TFT	127.9	70.4	99.1	3.98E6
TFT*	94.8	70.1	82.5	3.98E6
PatchTST	132.8	73.6	103.2	5.31E5
NHITS	139.7	75.6	107.6	6.75E6
MLP	119.6	74.9	97.2	1.27E6
TimesNet	158.5	83.2	120.9	4.72E6
VT	323.6	372.6	348.1	2.92E5
VT*	159.4	77.9	118.6	2.92E5
Informer	338.3	334.6	336.4	3.42E5
Informer*	273.1			3.42E5
Autoformer	283.3	113.9	198.6	2.91E5
Ensemble	113.7	63.4	88.6	-

Table 2: Test set Mean Absolute Error values for models from NeuralForecast. The result with the optimized model is in brackets. Prod=Production, Cons=Consumption, Ovr=Overall, VT = Vanilla Transformer, Ensemble = Ensemble of TFT+PatchTST+NHITS, * = model with optimized hyperparameters, # = number of parameters.

4 Experimental Results

In the NeuralForecast model experiments detailed in Table 2, the optimized Temporal Fusion Transformer emerged as the top performer, showcasing the lowest overall MAE on the test set. It also had the lowest MAE when compared to the production and consumption models' test results separately. 3 models out of the 4 top-performing models support past and future covariates. Examples of the predictions vs target production and consumption for the best performing model, the optimized Temporal Fusion can be found in Appendix A. The ensemble of the top 3 default models, where each of the models predictions was summed and the total was divided by 3, outperformed any single default model.

Hyperparameter optimization was not carried out for all the models. The 20 trial hyperparameter tuning was done for the Temporal Fusion Transformer, the MLP, the Vanilla Transformer, and the PatchTST. However, the MLP and PatchTST seemed to overfit on the validation set, demonstrated by a large decrease in the validation set MAE but a large increase in MAE on the test set. In contrast, the Vanilla Transformer and the Temporal Fusion Transformer benefitted from hyperparameter optimization. With the Informer we experienced GPU out of memory error when optimizing hyperparameters, even with setting a small batch size, smaller hidden size and head number values, using a 40GB-RAM GPU and cleaning the cache after every trial. However, even with a few trials we were able to find more optimal hyperparameters for the production model.

Setting a larger dropout rate, adjusting the input size, learning rate, feature scaler type, and the windows' batch size may have contributed to the Vanilla Transformer's better performance after optimization. Interestingly, the optimal learning rate for the consumption model is much higher than for the production model. The default and optimized hyperparameters, as well as the hyperparameter search ranges are in Appendix B.

The Optuna study' showed the random seed as the most important hyperparameter value in Vanilla Transformer production model optimization. To study this more closely, we conducted experiments where we fixed the other found optimal hyperparameters and ran the model with different seeds and trainer settings, including the number of validation check steps and early stopping patience. We found that there was variation in the model's performance when changing the random seed, the maximum change in MAE when changing only the random seed was 24.38 as can be seen in Appendix C. However, we found that changing trainer settings - increasing the maximum steps and early stopping patience to -1 - had a positive effect on the performance. This is also intuitive - the model had more time to train.

To examine the effect of future covariates on the predictions, we took one example segment (001), normalized the target production, Temporal Fusion Transformer's predictions, and PatchTST's predictions by dividing them with the installed capacity of that segment. Then we scaled the surface solar radiation downwards into a [0,1] range and plotted them on one graph as can seen in Figure 1. The surface solar radiation downwards is positively correlated with the target. On 2023-05-16 there was a drop in the amount of solar radiation. The Temporal Fusion Transformer, which supports future covariates, predicted lower production for that day. The PatchTST, which does not support future covariates in NeuralForecast, predicted a similar value as in the previous day.

Experiments with the TiDE model implemented in Darts showed that after optimizing both in the univariate and multivariate model, the multivariate model gave a better performance in prediction consumption and production. The summarized results are in Table 3. However, the experiment was conducted only by training the model on one segment. Therefore the MAE results are not directly comparable to the results of the NeuralForecast models.

In contrast to the TiDE models, Vanilla Transformers implemented via HuggingFace were trained on all available segments in both univariate and multivariate models. Having 2 univariate models leads to a lower overall MAE (82.76 vs 87.93), in addition to having a lower consumption MAE compared to the multivariate model (75.02 vs 94.79). Interestingly, training a multivariate model leads to a lower production MAE compared to the univariate model (81.08 vs 90.50). This can also be seen in the TiDE models. However, for TiDE models, the multivariate model improves for both consumption and production prediction. It is possible that predicting consumption is hard for the Vanilla Transformer and since the multivariate model must perform well when predicting both consumption and production, the performance drops. Conversely, it seems that training the Vanilla Transformer on both the consumption and production data makes it easier for the Vanilla Transformer to predict production (90.5 vs 81.1). The HuggingFace Vanilla Transformer results are not comparable to results in Table 2 because the MAE is calculated on a smaller sample.

	Cons	**Prod**	**Overall**	**#**
Uni(TiDE)	43.9	156.7	100.3	7.23E5
Multi(TiDE)	42.2	133.9	88.0	7.23E5
Uni(VT)	75.0	90.5	82.8	1.06E5
Multi(VT)	94.8	81.1	87.9	1.06E5

Table 3: Test set Mean Absolute Errors for Univariate and Multivariate TiDE models and Vanilla Transformer (HuggingFace) models. The TiDE models were trained on a single segment, but the Vanilla Transformers were trained on all segments available.

With TiDE we experimented with the model per county architecture, where one TiDE model was trained per county making 15 models in total and each training set had a list of time-series for segments in that county. This set-up achieved a 80.97 score on the Kaggle competition's hidden test set. The same ensemble trained with optimal hyperparameters found on one segment, decreased performance on the hidden test set to MAE 98.67 indicating that optimal hyperparameters for one segment might not suit a model with a larger dataset.

The Kaggle competition's had a hidden set and a blind submission procedure which made submission debugging extremely time-consuming. In addition, the data in the hidden set had duplicates and possibly also missing dates. For this reason we focused on comparing the models using the publicly available data.

5 Discussion

In solar energy production and consumption prediction, transformers adapted to time-series can perform well if optimized and implemented with support for past and future covariates. The lack of support for explanatory features is a serious drawback for the time-series transformers implemented in NeuralForecast.

In the Enefit's Kaggle competition, a new time series (county/is-business/product-type combination) can appear in the test set. For the models implemented in Darts and NeuralForecast, predictions can only be made for time series unique id-s (segments) that have been seen in training. This limitation leaves new appearing segments without

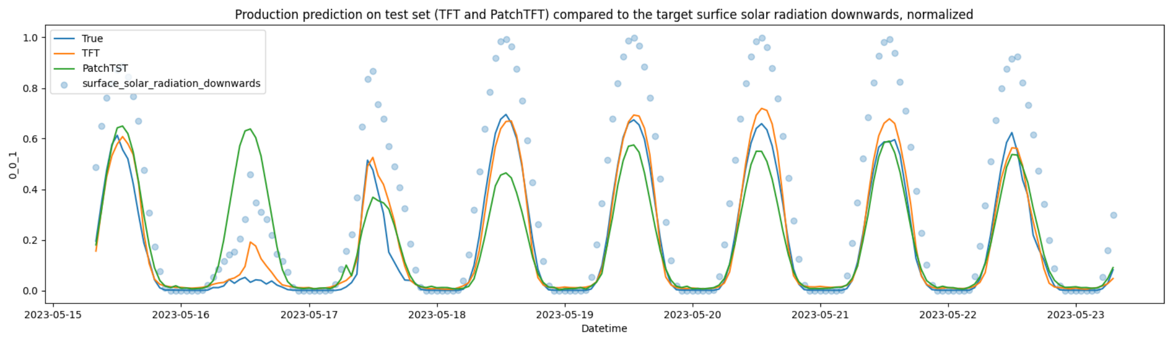

Figure 1: The relationship between surface solar radiation downwards, the target (True) and the predictions of the Temporal Fusion Transformer (TFT), and the Patch Time-Series Transformer (PatchTST). We used normalization to bring the values into a [0,1] range.

predictions which have to be imputed. This is one of the reasons why the competition is dominated by non-time-series models, e.g. tree-based boosting ensembles.

Deep neural networks can be sensitive to hyperparameters which is why they should be optimized. Optimization of Vanilla Transformer increased the performance of the model significantly. The learning rate, dropout rate, and scaler type are among the important hyperparameters. In the case of transformers, the model size and slow training due to the attention mechanism can be an obstacle as a fast high-memory GPU is needed and optimization takes a lot of resources. However, hyperparameter search did not yield better results for all models, in contract it may cause overfitting on the validation set.

As future work, wider hyperparameter search ranges can be used and optimization done for all the models. Using models that support past and future covariates is advisable. Newer time-series transformers that allegedly outperform the transformers used in this project, such as (Xue et al., 2023), can be tried.

As the ensemble of top 3 default Neuralforecast models gave a better test result than any model alone, more experimentation can be done in ensambling models, including transformers with non-neural models like tree-based boosting algorithms.

6 Conclusion

Transformers, which are the preferred choice in many natural language processing task, can also perform well on time-series forecasting tasks if optimized and trained on a large dataset. In experiments with the NeuralForecast library with several transformers and non-transformers, the best performing model was the Temporal Fusion Transformer. However, the MLP-based encoder-decoder TiDE implemented in Darts showed promising results in a model per county multivariate setting where it achieved 80.97 score on the competition's hidden test set.

We observe that transformer models without support for future covariates have difficulty with predicting solar consumption on days that are out of the ordinary, e.g. a day in May with less sunshine. Whereas a model which supports forecast weather as a future covariate is able to capture the decrease in production. The 3 out of 4 top models supported static, past and future covariates.

References

Sabeen Ahmed, Ian E. Nielsen, Aakash Tripathi, Shamoon Siddiqui, Ravi P. Ramachandran, and Ghulam Rasool. 2023. Transformers in time-series analysis: A tutorial. *Circuits, Systems, and Signal Processing*, 42(12):7433–7466.

Anastasia Borovykh, Sander Bohte, and Cornelis W Oosterlee. 2017. Conditional time series forecasting with convolutional neural networks. *arXiv preprint arXiv:1703.04691*.

Pierre Bouquet, Ilya Jackson, Mostafa Nick, and Amin Kaboli. 2023. Ai-based forecasting for optimised solar energy management and smart grid efficiency. *International Journal of Production Research*, pages 1–22.

Abhimanyu Das, Weihao Kong, Andrew Leach, Shaan K Mathur, Rajat Sen, and Rose Yu. 2023. Long-term forecasting with tiDE: Time-series dense encoder. *Transactions on Machine Learning Research*.

Rakshitha Godahewa, Christoph Bergmeir, Geoffrey I Webb, Rob J Hyndman, and Pablo Montero-Manso.

2021. Monash time series forecasting archive. *arXiv preprint arXiv:2105.06643*.

Julien Herzen, Francesco LÃ¤ssig, Samuele Giuliano Piazzetta, Thomas Neuer, LÃ©o Tafti, Guillaume Raille, Tomas Van Pottelbergh, Marek Pasieka, Andrzej Skrodzki, Nicolas Huguenin, Maxime Dumonal, Jan KoÅ›cisz, Dennis Bader, FrÃ©dÃ©rick Gusset, Mounir Benheddi, Camila Williamson, Michal Kosinski, Matej Petrik, and GaÃ«l Grosch. 2022. Darts: User-friendly modern machine learning for time series. *Journal of Machine Learning Research*, 23(124):1–6.

Bryan Lim, Sercan Ö Arık, Nicolas Loeff, and Tomas Pfister. 2021. Temporal fusion transformers for interpretable multi-horizon time series forecasting. *International Journal of Forecasting*, 37(4):1748–1764.

William Michael John Murphy and Ke Chen. 2023. Univariate vs multivariate time series forecasting with transformers.

Yuqi Nie, Nam H Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. 2022. A time series is worth 64 words: Long-term forecasting with transformers. *arXiv preprint arXiv:2211.14730*.

Kin G. Olivares, Cristian Challú, Federico Garza, Max Mergenthaler Canseco, and Artur Dubrawski. 2022. NeuralForecast: User friendly state-of-the-art neural forecasting models. PyCon Salt Lake City, Utah, US 2022.

F. Rosenblatt. 1958. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and Jamie Brew. 2019. Huggingface's transformers: State-of-the-art natural language processing. *CoRR*, abs/1910.03771.

Haixu Wu, Tengge Hu, Yong Liu, Hang Zhou, Jianmin Wang, and Mingsheng Long. 2023. Timesnet: Temporal 2d-variation modeling for general time series analysis.

Haixu Wu, Jiehui Xu, Jianmin Wang, and Mingsheng Long. 2021. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. *Advances in Neural Information Processing Systems*, 34:22419–22430.

Wang Xue, Tian Zhou, Qingsong Wen, Jinyang Gao, Bolin Ding, and Rong Jin. 2023. Make transformer great again for time series forecasting: Channel aligned robust dual transformer.

Ailing Zeng, Muxi Chen, Lei Zhang, and Qiang Xu. 2023. Are transformers effective for time series forecasting? In *Proceedings of the AAAI conference on artificial intelligence*, volume 37, pages 11121–11128.

Haoyi Zhou, Jianxin Li, Shanghang Zhang, Shuai Zhang, Mengyi Yan, and Hui Xiong. 2023. Expanding the prediction capacity in long sequence time-series forecasting. *Artificial Intelligence*, 318:103886.

Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. 2020. Informer: Beyond efficient transformer for long sequence time-series forecasting. *CoRR*, abs/2012.07436.

Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. 2021. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 11106–11115.

A An example of Temporal Fusion Transformer's predictions on one segment

B Vanilla Transformer's hyperparameters

Parameter	Default	Prod	Cons
input_size	168	111	281
hidden_size	128	64	128
n_head	4	4	16
learning_rate	1E-4	9.8E-4	1.2E-3
scaler_type	identity	standard	robust
batch_size	32	32	16
w_batch_size	1024	128	512
drop_out_rate	0.05	0.1	0.3
random_seed	1	18	3

Table 4: Best hyperparameters for the Vanilla Transformer. Prod = Production, Cons = Consumption, w_batch_size = windows_batch_size.

Hyperparameter importances for the Vanilla Transformer production model tuning study:

1. random_seed = 0.300
2. drop_out_rate = 0.218
3. input_size = 0.200
4. scaler_type = 0.101
5. batch_size = 0.047
6. learning_rate = 0.040
7. n_head = 0.035
8. windows_batch_size = 0.032
9. hidden_size = 0.028

C Neuralforecast's vanilla transformer's experiments with random seed

Seed	Max steps	Val check steps, p	MAE
1	1000	100, 3	183.79
18	1000	100, 3	159.41
18	5000	100, 3	154.12
3	5000	100, 3	159.94
18	5000	100, -1	139.34
1	5000	100, -1	136.75
7	5000	100, -1	125.20
1	10000	100, 5	165.3
1	6000	100, -1	141.93

Table 5: Results and settings for the Neuralforecast's vanilla transformer with the same hyperparameters, changing the random seed, maximum steps, and early stopping for the production model.

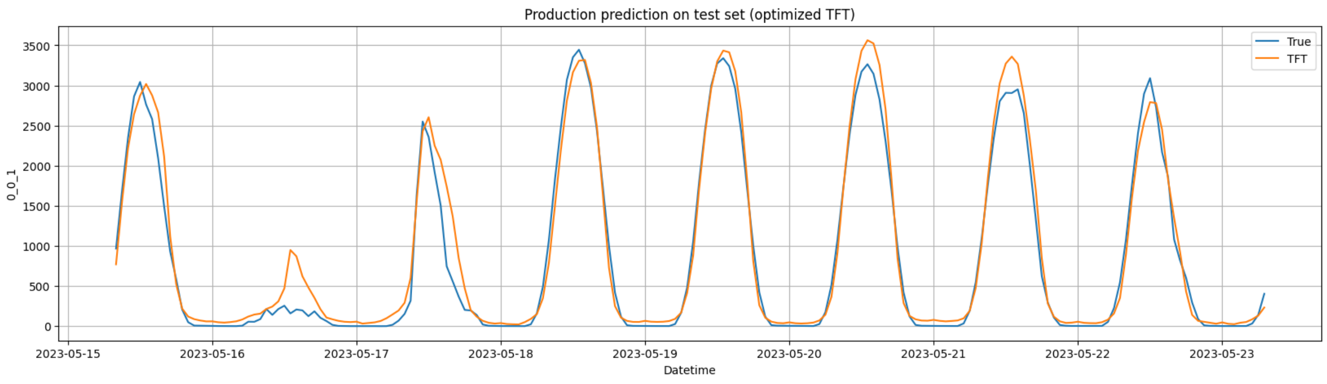

Figure 2: Energy production prediction for segment 001 using optimized TFT.

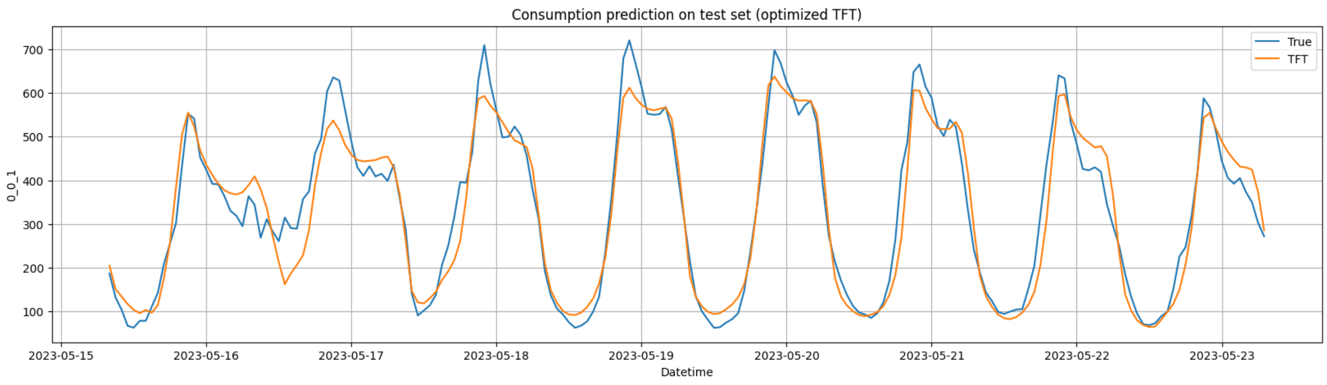

Figure 3: Energy consumption prediction for segment 001 using optimized TFT.