MÁSTER UNIVERSITARIO EN INDUSTRIA CONECTADA

TRABAJO FIN DE MÁSTER

Modelos de series temporales basados en aprendizaje profundo para la predicción de la generación eléctrica en parques eólicos

Madrid
2025

Autor: Claudia Coduras Gracia
Director: Fernando San Segundo Barahona
Subdirector: Antonio Muñoz San Roque

Table of contents

Abstract

Offshore wind energy offers high generation potential but poses forecasting challenges due to wind variability. Accurate short-term predictions are essential for intraday electricity market operations. This study benchmarks traditional and deep learning models—ARIMA, XGBoost, LSTM, GRU—against recent architectures like PatchTST (Transformer-based) and RMoK (KAN) for 3-hour-ahead offshore wind power forecasting. Using real-world data from **Alpha Ventus** and a public **Kaggle** dataset, results show that modern models consistently outperform classical ones in accuracy and generalization.

Keywords: Offshore wind, Deep Learning, Time series forecasting, Transformer, KAN, Renewable energy, Intraday market.

1 Introduction

The problem of accurate forecasting is crucial for integrating renewable energy sources, particularly offshore wind, which is highly variable yet promising. This thesis builds on previous work by Carlos de los Santos [1] and extends it by incorporating recent deep learning models such as Transformers and Kolmogorov–Arnold Networks (KAN) [2]. The main objective is to benchmark the performance of various time series models for short-term wind power prediction over a 3-hour forecast horizon, which is significant for intraday market operations. The models evaluated include traditional ones (ARIMA, XGBoost, LSTM, GRU) as well as newer architectures (Transformer, MLP, KAN). The evaluation focuses on key aspects like forecasting accuracy (MAE), scalability, generalization, and reproducibility, using open-source implementations.

2 Methodology

This Master's Thesis compares classical statistical, machine learning, and deep learning methods for forecasting electricity generation in offshore wind farms using Python, the dominant language for time series analysis. While traditional models like ARIMA were implemented in R, deep learning libraries like **TensorFlow/Keras** and **PyTorch** have fragmented the Python ecosystem. The thesis evaluates Python libraries (**statsmodels**, **scikit-learn**, **sktime**, **darts**) but highlights their lack of support for advanced deep learning models, addressing this gap with the **Nixtlaverse**, a modern set of open-source libraries for time series forecasting which includes:

- **StatsForecast** for classical models like ARIMA and exponential smoothing,
- **MLForecast** for integrating machine learning regressors like scikit-learn and XGBoost,
- **NeuralForecast** for deep learning models such as MLPs, LSTMs, and Transformer-based architectures.

This methodology enables comprehensive and up-to-date benchmarking of both traditional and deep learning models in a coherent Python framework.

3 Data Description

The workflow followed involves the use of two datasets:

- **Alpha Ventus** Dataset: Inherited from Carlos de los Santos [1], including data preprocessing and the selection of relevant variables.
- Public Dataset Available on **Kaggle** [3]: This work uses a dataset obtained from Kaggle, a widely recognized platform for open data. Its inclusion allows evaluating the generalization and scalability capabilities of the developed models in a realistic and replicable environment.

For both datasets, rigorous preprocessing was applied, including missing value imputation, variable normalization, and the generation of temporal features (lags, moving averages, etc.) to enrich the model inputs. The data was then split into training, validation, and test sets using a temporal hold-out strategy, maintaining chronological order to avoid **data leakage**.

4 Models Compared

Model	Type	Key Feature
ARIMA	Statistical	Linear baseline
XGBoost	ML Ensemble	Gradient Boosting with feature selection
LSTM, GRU	Recurrent NN	Captures temporal dependencies
PatchTST, TimeXer	Transformer	Attention-based sequence modeling
NBeats, NHits, TiDE	MLP + Seq2Seq	Dense temporal encoders
RMoK	Kolmogorov Network	Mixture of Experts using learnable activation

This paper will focus exclusively on the implementation and results of the new architectures based on Transformers, MLP, and KAN.

5 Transformer-based Models

Introduction

The Transformer architecture, introduced in the 2017 paper *Attention is All You Need* [4], revolutionized machine translation by replacing recurrent models with parallel processing via GPUs. Key innovations include **embeddings** with positional encodings and the **attention mechanism** for better long-range dependency modeling. Additional features like **residual connections** and **layer normalization** improved training stability.

Since 2019, researchers have adapted Transformers for **time series forecasting**, addressing challenges such as:

- **Improving locality**: Time series often rely on local patterns, leading to the introduction of convolutional self-attention for focusing on nearby time points.
- **Reducing memory usage**: To tackle quadratic scaling, sparse attention mechanisms, such as logarithmic selection of prior time steps, were proposed for efficiency.

6 Patch-TST model

PatchTST, a Transformer-based model designed for long-term time series forecasting, processes input sequences as non-overlapping patches to capture local patterns efficiently. By leveraging a patching mechanism inspired by vision transformers, PatchTST enhances forecasting accuracy while reducing computational complexity.

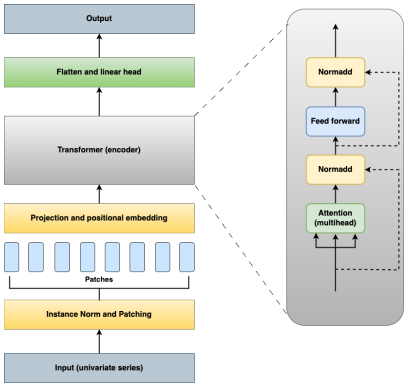


Figure 1 : Model Architecture of PatchTST

Implementation and results

The implementation of PatchTST used in this work is the one available in Nixtla's neuralforecast library [5], specifically through the `PatchTST` and `AutoPatchTST` classes, following the procedure previously described in Section 2 of the Master's Thesis. The result is available in the *models/PatchTST* folder of the Github repository created for this Master's Thesis .

Along with the fixed values described in Section 2 of the Master's Thesis, the list of specific hyperparameters considered in this work is:

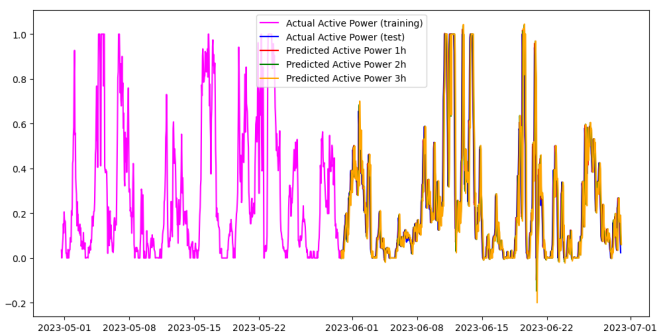
Parameter	Description
`encoder_layers`	Number of layers in the encoder part of the model
`n_heads`	Number of attention heads in the attention block
`hidden_size`	Number of neurons in the model's feedforward networks
`dropout`	Dropout rate for the residual connections
`head_dropout`	Dropout rate for the linear layer
`attn_dropout`	Dropout rate for the attention layer
`patch_len`	Length of the input segment (patch)
`learning_rate`	Learning rate

Results for the AV and KaggleWPGD datasets using PatchTST.

The models corresponding to the AV and KaggleWPGD data are found in the Jupyter notebooks named `PatchTST_AV.ipynb` and `PatchTST_KaggleWPGD.ipynb`, respectively. The results of the execution of these models can be found in the documents PatchTST_AV.html and PatchTST_KaggleWPGD.html, available in the code repository.

With an input size of 6 hours and a prediction horizon of 3 hours, the predictions shown in Figure 2 are obtained. The training time for both models (including hyperparameter selection) is approximately 12 minutes. The validation loop time for the AV dataset is approximately 80 minutes.

Below are the predictions from the PatchTST model, illustrating the prediction evolution over the forecast horizon.



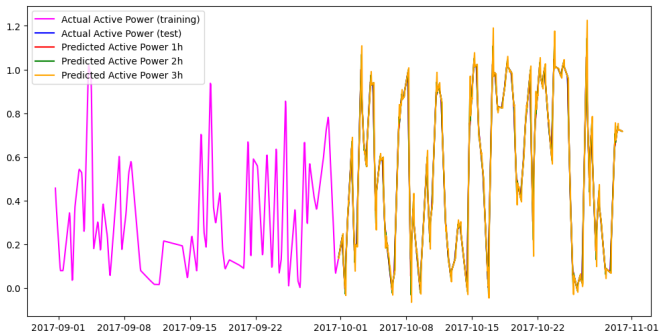


Figure 2 : Predictions of PatchTST model for the AV and Kaggle dataset

The Mean Absolute Error (MAE) values obtained for the test set predictions are presented in Figures 11 and 12.

- Qualitatively, it can be observed that the model produces reasonably good predictions. However, due to its architecture, in the case of the AV dataset, it yields results outside the [0, 1] interval to which the original (normalized) time series values belong. This behavior could be easily corrected through post-processing, but the best approach would be to modify the model architecture so that the final prediction layer is a sigmoid layer, naturally producing predictions within the desired interval.
- Most importantly: the PatchTST model **does not incorporate the use of exogenous variables** as predictors in its design. For that reason, we have explored a transformer-based model that natively supports them.

7 TimeXer model

TimeXer, a Transformer-based architecture tailored for long-term time series forecasting, uses **cross-channel attention** to model dependencies across multiple variables. Its design focuses on capturing inter-series interactions efficiently, improving performance in multivariate forecasting tasks.

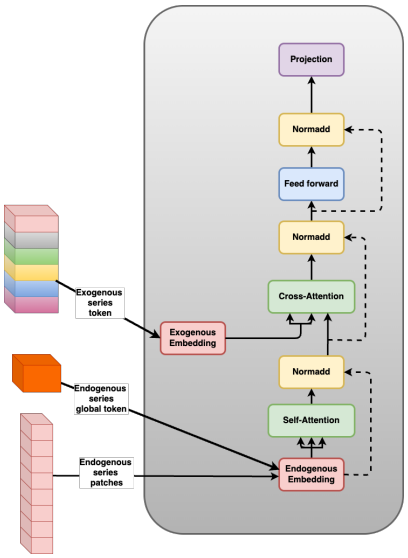


Figure 3 : Model Architecture of TimeXer

Implementation and results TimeXer

For the TimeXer model, we have again used an implementation available in the Nixtla neuralforecast library [5], through the `TimeXer` and `AutoTimeXer` classes, following the procedure described earlier in Section 2 of the Master's Thesis. The result is available in the *models/TimeXer* folder of the GitHub repository.

Along with the fixed values described in Section 2 of the Master's Thesis, the list of specific hyperparameters considered in this work is:

Parameter	Description
`encoder_layers`	Number of layers in the encoder part of the model
`n_heads`	Number of attention heads in the attention block
`hidden_size`	Number of neurons in the model's feedforward networks
`dropout`	Dropout rate for the residual connections
`head_dropout`	Dropout rate for the linear layer
`attn_dropout`	Dropout rate for the attention layer
`patch_len`	Length of the input segment (patch)
`learning_rate`	Learning rate

Results for the AV and KaggleWPGD datasets using TimeXer.

The models corresponding to the AV and KaggleWPGD data are found in the Jupyter notebooks named `TimeXer_AV.ipynb` and `TimeXer_KaggleWPGD.ipynb`, respectively. The results of the execution of these models can be found in the documents TimeXer_AV.html and TimeXer_KaggleWPGD.html, available in the code repository.

With an input size of 6 hours and a prediction horizon of 3 hours, the predictions shown in Figure 4 are obtained. The training time for both models (including hyperparameter selection) is approximately 12 minutes. The validation loop time for the AV dataset is approximately 80 minutes.

Below are the predictions from the TimeXer model, illustrating the prediction evolution over the forecast horizon.

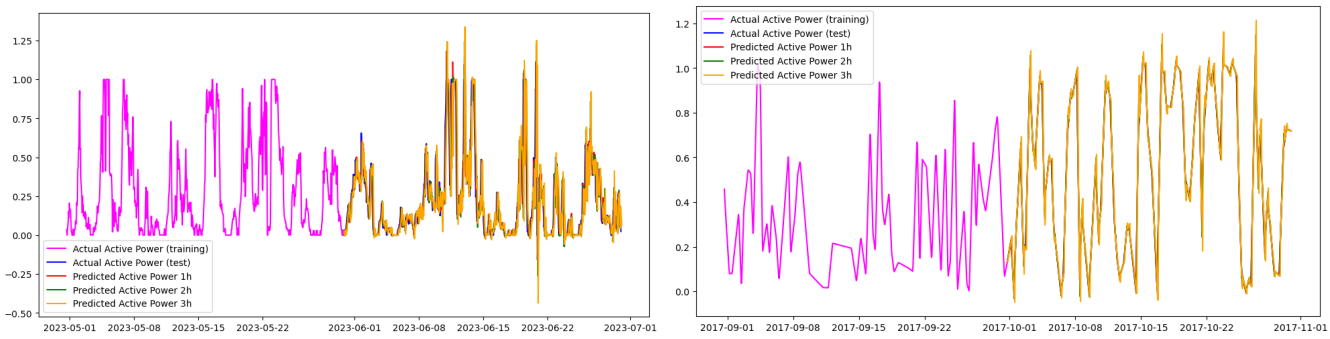


Figure 4 : Predictions of the TimeXer model for the AV and Kaggle dataset

The Mean Absolute Error (MAE) values obtained for the test set predictions are presented in Figures 11 and 12.

- Qualitatively, it can be seen that the model produces acceptably good predictions. However, due to its architecture, in the case of the AV dataset, it yields results outside the [0, 1] range to which the values of the original (normalized) time series belong. This behavior has already been discovered in PatchTST and the best approach is the same.
- From a computational perspective, a notable increase in the validation cost is observed compared to models like LSTM or XGBoost. In particular, the KaggleWPGD dataset has been a bottleneck in terms of efficiency, with validation times exceeding 7 hours. This high load could be due to the intensive use of attention mechanisms over long windows.

8 MLP-based Models

In recent years, MLP-based architectures have regained popularity in time series forecasting due to their simplicity and efficiency. Models like TiDE (Time-series Dense Encoder) [6] and N-HiTS (Neural Hierarchical Interpolation for Time Series) [7] leverage dense layers and hierarchical structures to capture nonlinear relationships and improve generalization. These models offer competitive performance against more complex architectures like Transformers, providing a scalable and efficient solution for time series prediction.

9 NHiTS model

N-HiTS (Neural Hierarchical Interpolation for Time Series Forecasting), a deep learning model, uses **hierarchical interpolation** and **residual learning** to capture patterns at multiple temporal resolutions. Its modular architecture enables efficient and accurate long-term forecasting by focusing on different frequency components of the time series.

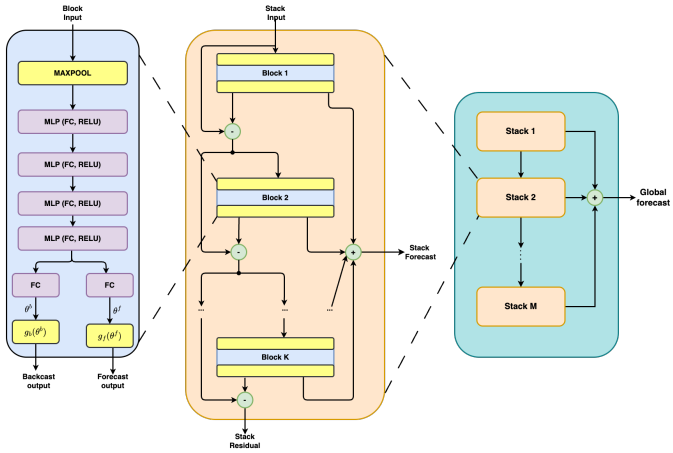


Figure 5 : Model Architecture of N-HiTS

Implementation and results NHiTS

The implementation of N-HITS that we have used is the one available in the Nixtla neuralforecast library [5], specifically through the `NHITS` and `AutoNHITS` classes, following the procedure described earlier in Section 2 of the Master's Thesis. The result is available in the *models/NHITS* folder of the GitHub repository.

Along with the fixed values described in Section 2 of the Master's Thesis, the list of specific hyperparameters considered in this work is:

Parameter	Description
`encoder_layers`	Number of layers in the encoder part of the model
`n_heads`	Number of attention heads in the attention block
`hidden_size`	Number of neurons in the model's feedforward networks
`dropout`	Dropout rate for the residual connections
`head_dropout`	Dropout rate for the linear layer
`attn_dropout`	Dropout rate for the attention layer
`patch_len`	Length of the input segment (patch)
`learning_rate`	Learning rate

Results for the AV and KaggleWPGD datasets using NHiTS.

The models corresponding to the AV and KaggleWPGD data are found in the Jupyter notebooks named `NHiTS_AV.ipynb` and `NHiTS_KaggleWPGD.ipynb`, respectively. The results of the execution of these models can be found in the documents NHiTS_AV.html and NHits_KaggleWPGD.html, available in the code repository.

With an input size of 6 hours and a prediction horizon of 3 hours, the predictions shown in Figure 6 are obtained. The training time for both models (including hyperparameter selection) is approximately 12 minutes. The validation loop time for the AV dataset is approximately 80 minutes.

Below are the predictions from the NHiTS model, illustrating the prediction evolution over the forecast horizon.

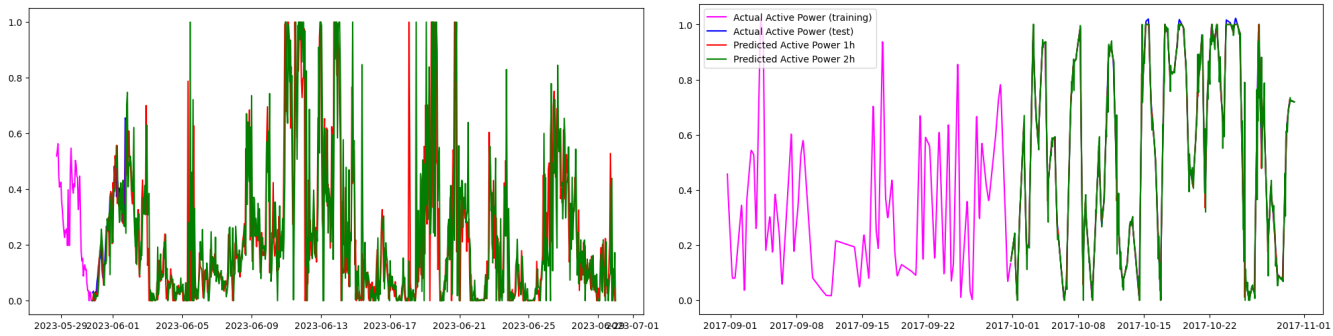


Figure 6 : Predictions of the N-HiTS modelfor the AV and Kaggle dataset

The Mean Absolute Error (MAE) values obtained for the test set predictions are presented in Figures 11 and 12.

The N-HiTS model has proven to be effective at modeling complex temporal dynamics due to its hierarchical architecture based on backcasting/forecasting blocks.

However, this architectural improvement comes with increased computational complexity and greater sensitivity to hyperparameter selection, which requires careful attention during the tuning process through validation. In this work, this selection was performed using the AutoNHITS class from the neuralforecast library, which allowed automating this task and reducing the risk of overfitting.

10 TiDE model

TiDE (Time series Decomposition and Encoding), a model designed for long-term time series forecasting, decomposes the input series into trend, seasonality, and residual components. By encoding these components separately, TiDE improves forecasting accuracy while handling complex patterns and non-stationarities in the data.

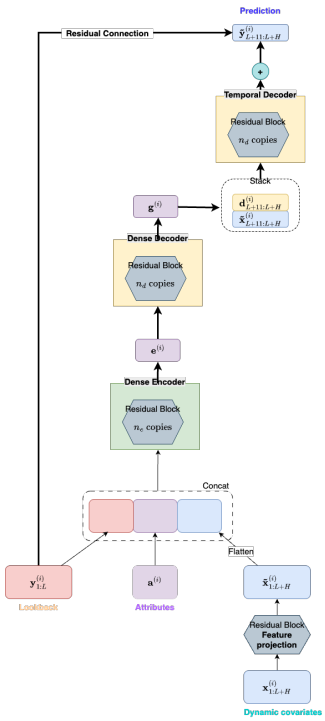


Figure 7 : Model Architecture of TiDE

Implementation and results TiDE

The TiDE implementation we used is the one available in Nixtla's neuralforecast library [5], specifically through the `TiDE` and `AutoTiDE` classes, following the procedure described earlier in Section 2 of the Master's Thesis. The result is available in the *models/TiDE* folder of the GitHub repository.

Along with the fixed values described in Section 2 of the Master's Thesis, the list of specific hyperparameters considered in this work is:

Parameter	Description
`decoder_output_dim`	Number of units in the dense decoder output layer
`hidden_size`	Number of units in the dense layers
`num_encoder_layers`	Number of encoder layers
`num_decoder_layers`	Number of decoder layers
`temporal_decoder_dim`	Number of units in the temporal decoder dense layers
`dropout`	Dropout rate

Parameter	Description
`learning_rate`	Learning rate

Results for the AV and KaggleWPGD datasets using TiDE.

The models corresponding to the AV and KaggleWPGD data are found in the Jupyter notebooks named `TiDE_AV.ipynb` and `TiDE_KaggleWPGD.ipynb`, respectively. The results of the execution of these models can be found in the documents TiDE_AV.html and TiDE_KaggleWPGD.html, available in the code repository.

With an input size of 6 hours and a prediction horizon of 3 hours, the predictions shown in Figure 8 are obtained. The training time for both models (including hyperparameter selection) is approximately 25 minutes. The validation loop time for the AV dataset is *over six hours*.

Below are the predictions from the TiDE model, illustrating the prediction evolution over the forecast horizon.

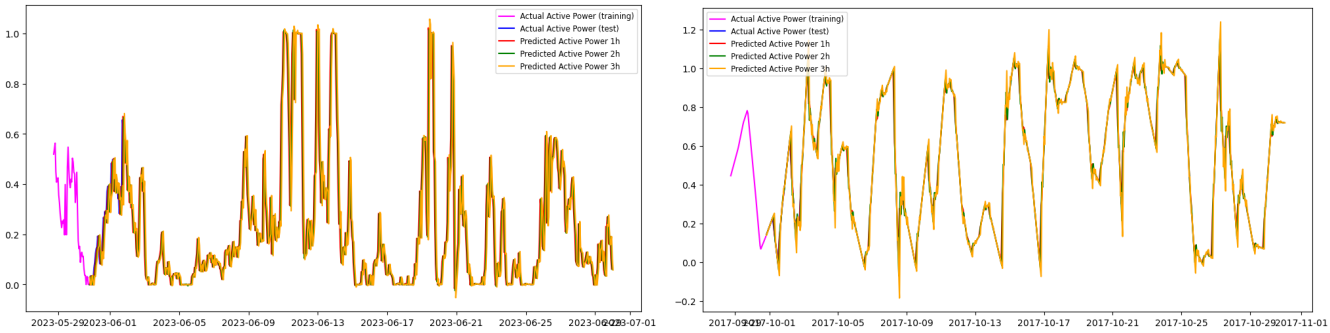


Figure 8 : Predictions of the TiDE model for the AV and Kaggle dataset

The Mean Absolute Error (MAE) values obtained for the test set predictions are presented in Figures 11 and 12.

- Qualitatively, the predictions reflect a good ability to capture the dynamics of the series, both in the short and medium term.
- Unlike models like PatchTST, TiDE is designed to natively incorporate exogenous variables. This feature is especially relevant in energy prediction tasks, where additional information (such as weather or operational data) can be crucial in improving the model's accuracy.

11 KAN-based Models

RMoK is a time series forecasting model proposed in 2024 as an adaptation of Kolmogorov–Arnold Networks (KAN) [8]. It leverages a minimal architecture with specialized KAN experts and a gating mechanism, aiming for high interpretability and efficiency.

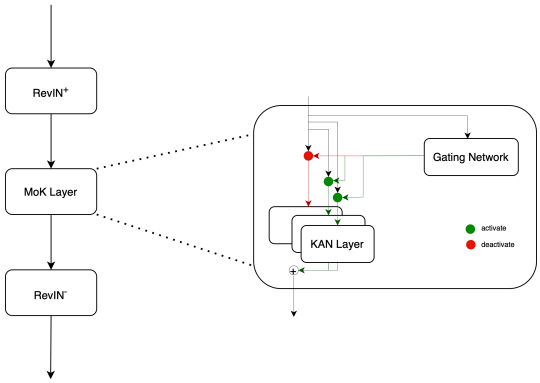


Figure 9 : Model Architecture of RMoK

Implementation and results RMoK

Once again, in this case, we were able to use the RMoK implementation available in Nixtla's *neuralforecast* library [5], which is especially noteworthy for a model *published in August 2024*. The implementation uses the `RMoK` and `AutoRMoK` classes, similarly to what we have seen in other models, following the procedure described in Section 2 of the Master's Thesis. The result is available in the *models/RMoK* folder of the GitHub repository.

Along with the fixed values described in Section 2 of the Master's Thesis, the list of specific hyperparameters considered in this work is:

Parameter	Description
`taylor_order`	Order of the Taylor polynomial used
`wavelet_function`	Shape of the wavelet function
`dropout`	Dropout rate
`learning_rate`	Learning rate

Results for the AV and KaggleWPGD datasets using RMoK.

The models corresponding to the AV and KaggleWPGD data are found in the Jupyter notebooks named `RMoK_AV.ipynb` and `RMoK_KaggleWPGD.ipynb`, respectively. The results of the execution of these models can be found in the documents RMoK_AV.html and RMoK_KaggleWPGD.html, available in the code repository.

With an input size of 6 hours and a prediction horizon of 3 hours, the predictions shown in Figure 10 are obtained. The training time for both models (including hyperparameter selection) is approximately 14 minutes. The validation loop time for the AV dataset is *two hours*.

Below are the predictions from the RMoK model, illustrating the prediction evolution over the forecast horizon.

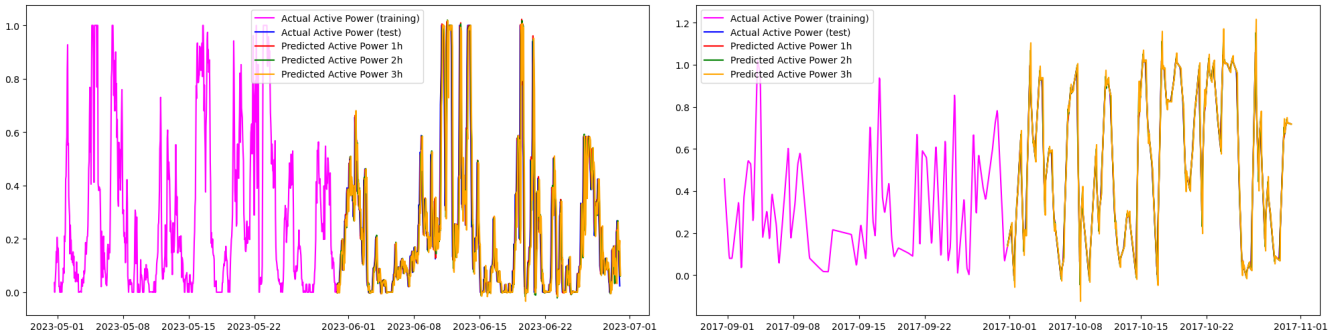


Figure 10 : Predictions of the RMoK model for the AV and Kaggle dataset.

The Mean Absolute Error (MAE) values obtained for the test set predictions are presented in Figures 11 and 12.

For the AV dataset, the results are comparable to those obtained with TiDE, slightly better in the long term (3h). Like TiDE, RMoK is designed to natively incorporate exogenous variables, making it a suitable model for complex contexts where the target variable depends on multiple external factors. This feature gives it a significant advantage over architectures such as PatchTST, which do not support this integration directly.

12 Results

Results Key Findings from the AV Dataset Analysis

The following figure presents the comparative analysis and applicability assessment based on the AV dataset. This analysis allows us to evaluate the performance of the different models considered for predicting electricity generation in offshore wind farms. The results provide a detailed insight into the accuracy and reliability of the models in the context of this specific dataset.

AV	XGBoost	LSTM	Patch_TST	TimeXer	Nbeats	TiDE	RMoK
1h	320.90	530.91	236.98	263.77	394.64	236.97	239.57
2h	384.08	634.94	375.87	411,96	716.48	377.24	378.78
3h	442.75	736.11	487.08	518.84	901.91	486.45	484.76

Figure 11 : Comparative analysis and applicability assessment MAE for AV Dataset

- Transformer (PatchTST) and KAN-based models (RMoK) clearly outperform classical and recurrent architectures.
- PatchTST proves to be the most effective model for capturing short-term temporal dependencies (e.g., 1-hour forecasts), while TiDE and RMoK demonstrate superior performance as the forecasting horizon increases, making them more suitable for modeling long-term temporal dynamics.
- RMoK stands out for its balance between accuracy and stability, making it a strong candidate for production environments where robust multi-horizon forecasting is required.
- LSTM and N-BEATS exhibit clear limitations, with performance deteriorating significantly over time.

.

Results and key Findings from the Kaggle Dataset

Kaggle	XGBoost	LSTM	Patch_TST	TimeXer	Nbeats	TiDE	RMoK
1h	0.0280	0.0170	0.0035	0.0034	0.0067	0.0053	0.0041
2h	0.0478	0.0236	0.0117	0.0113	0.0229	0.0157	0.0129
3h	0.0664	0.0266	0.0242	0.0241	0.0467	0.0290	0.0262

Figure 12 : Comparative analysis and applicability assessment MAE for Kaggle Dataset

- Transformers (TimeXer and PatchTST) dominate the benchmark on this dataset, achieving the best results across all forecast horizons. Their extremely low MAE confirms their suitability for clean and complex data.
- RMoK and TiDE prove to be very strong alternatives, especially for tasks where a trade-off between accuracy and interpretability is important.
- XGBoost and LSTM are clearly outperformed, reinforcing the notion that modeling deep temporal dependencies requires modern architectures.
- The consistency between the AV and Kaggle results strengthens the conclusions of this work and suggests that advanced models (Transformers and KAN) offer superior generalization capabilities.

Referencias

[1] C. de los Santos Jiménez, "Análisis comparativo de modelos de aprendizaje automático para la predicción de la generación eléctrica de un parque eólico marino." Universidad Pontificia Comillas, 2023. Available: https://repositorio.comillas.edu/xmlui/handle/11531/83380

[2] L. H. Hao Wang Jingzhen Ye, "A multivariable hybrid prediction model of offshore wind power based on multi-stage optimization and reconstruction prediction." 2023. Available: https://www.sciencedirect.com/science/article/abs/pii/S0360544222023106

[3] M. Rahim, "Wind power generation data – forecasting." Kaggle dataset, 2024. Available: https://www.kaggle.com/datasets/mubashirrahim/wind-power-generation-data-forecasting

[4] A. Vaswani *et al.*, "Attention is all you need," in *Advances in neural information processing systems*, Curran Associates, Inc., 2017, pp. 5998–6008. Available: https://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

[5] N. Developers, "Nixtla: Machine learning and statistical methods for time series forecasting." https://nixtla.github.io/, 2023.

[6] W. Kong, A. Das, A. Leach, R. Sen, and R. Yu, "Long-term forecasting with TiDE: Time-series dense encoder," *arXiv preprint arXiv:2304.08424*, 2023, Available: https://arxiv.org/abs/2304.08424

[7] C. Challu, K. G. Olivares, B. N. Oreshkin, F. Garza, M. Mergenthaler-Canseco, and A. Dubrawski, "N-HiTS: Neural hierarchical interpolation for time series forecasting," *arXiv preprint arXiv:2201.12886*, 2022, Available: https://arxiv.org/abs/2201.12886

[8] X. Han, X. Zhang, Y. Wu, Z. Zhang, and Z. Wu, "Are KANs effective for multivariate time series forecasting?" 2025. Available: https://arxiv.org/abs/2408.11306