Forecasting axial offset using tree-based models: a step towards improved nuclear power plants manoeuvrability

Madina Traoré, Giorgio Simonini, Yannick Goude, Mathieu Lagrange, Alexandre Girard, Jérôme Idier

Forecasting axial offset using tree-based models: a step towards improved nuclear power plants manoeuvrability

M. Traoré[1,2], G. Simonini[1], Y. Goude[1], M. Lagrange[2], A. Girard[1] and J. Idier[2]
[1]EDF R&D
7 boulevard Gaspard Monge, 91120 Palaiseau, France

[2]Nantes Université, École Centrale Nantes, CNRS, LS2N, UMR 6004, F-44000 Nantes, France
1, rue de la Noë, 44321 , Nantes, France

madina.traore@edf.fr, giorgio.simonini@edf.fr, yannig.goude@edf.fr, mathieu.lagrange@ls2n.fr
alexandre.girard@edf.fr, jerome.idier@ls2n.fr

ABSTRACT

Due to the significant increase in the share of variable renewable electricity generation, the power grid flexibility must be enhanced. To achieve this while reducing CO2 emissions, improving the manoeuvrability of low-carbon sources such as nuclear power plants (NPP) is essential. Time series forecasting is a crucial task to enhance NPPs manoeuvrability. In particular, because the ability to forecast critical core operating parameters in a fast and trustworthy manner would substantially help operators perform load-following operations. Traditionally, core nuclear reactor operating parameters are estimated through neutron transport simulations, which require a significant amount of computational resources. Alternatively, machine learning (ML) techniques such as neural networks have been increasingly used to address this type of problem. In this study, we propose a novel data-based ML method combining tree-based regressors (Random Forest or Gradient Boosted Trees) and time series models to forecast the normalized axial offset, a core parameter that must be monitored and controlled by the operators working in the control room. We consider a rigorous optimization procedure to determine hyperparameter values leading to high performance. Finally, we validate our approach on French 1300 MW pressurized water reactors (PWR) historical data. We show that, knowing the operators' projected commands, we can compute minute-by-minute forecasts in a few fractions of a second for the next eight hours, with suitable accuracy for the task at hand.

KEYWORDS: Machine Learning, Time series forecasting, Gradient Boosting, Nuclear power plants, Axial offset

1. INTRODUCTION

The projected rise in electricity demand due to electrification, together with the significantly increased share of variable renewable electricity generation, calls for extensive efforts to ensure the stability and flexibility of electricity supply [1]. Low-carbon, controllable energy sources, such as nuclear power plants (NPPs), are needed to address these new challenges. Furthermore, NPPs flexibility is currently underexploited, especially in France [2]. NPPs should, therefore, technically be able to compensate for production variability from renewable energy sources. This can be achieved by increasing the number of reactors participating in load-following as well as the number and amplitude of power transients (up and down ramps) performed by each reactor [3, 4]. To effectively handle these numerous and challenging power transients, operators need more efficient decision support tools for the monitoring and control of critical operating parameters.

Estimating core nuclear reactor operating parameters. Traditionally, core nuclear reactor operating parameters are estimated through neutron transport simulations. Different approaches based on neutron transport have emerged over time - leading to the development of numerous nuclear codes - such as point reactor core modeling, one-dimensional reactor core modeling, multi-node core modeling and three-dimensional

reactor core modeling [5]. Due to the development of powerful Machine Learning (ML) algorithms capable of capturing complex dynamics, ML techniques have been increasingly used to identify nuclear power plants (NPPs) dynamics and address operating parameters forecasting problems. For instance, Kozma et al. [6] and Kim et al. [7] used neural networks to predict multiple core parameters to help with core monitoring and fuel reloading problems, respectively. Several researchers used recurrent neural networks to identify the core dynamics of Water-Water Energy Reactors (WWERs) and Pressurized Water Reactors (PWRs) by building models able to forecast core parameters such as the axial offset [8, 9]. Zio et al. proposed to use Locally Recurrent Neural Networks to estimate the neutron flux, only knowing the reactivity evolution [10]. More recently, Naimi et al. [11], used a dynamic neural network to forecast the neutronic power of a PWR. We noted that most of these studies have been carried out with simulators or research reactors and proposed solutions have not been evaluated on data from commercial NPPs.

Contributions. Our contributions are as follows:

- We propose a novel approach combining tree-based and time series models to forecast the normalized axial offset in short and long horizons (one minute to eight hours).
- We demonstrate how to efficiently optimize the hyperparameters of our models by performing a Bayesian Search.
- We compare the performance and computational time of two tree-based methods, a Random Forest (RF) and Gradient Boosted Trees (GB) trained on a large amount of operational data from the French 1300 MW PWRs.
- We validate our hybrid method on a different subset of operational data from these reactors.

2. THEORETICAL BACKGROUND

2.1. Tree-based methods

Tree-based methods have become very popular in the field of statistical learning due to their simplicity, interpretability and high performance on supervised classification and regression problems. These non-parametric methods require minimal assumptions about the underlying data generating process, making them highly versatile and applicable to a wide range of problems including nonlinear ones. Moreover, it has been demonstrated that they can outperform Deep Learning methods when applied to tabular data like ours [12]. In this section we describe three tree-based methods that were used in this work: Regression Trees, Random Forests and Gradient Boosted Trees.

Regression trees. A regression tree is a predictive model that takes the form of a binary tree where each internal node tests the value of a given input feature and each leaf node contains the predicted value of

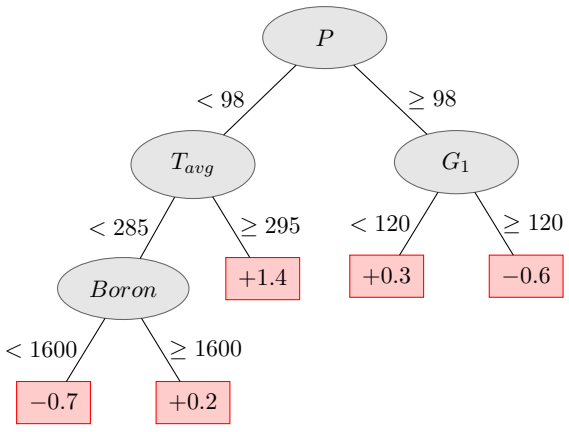

Figure 1: Example of a regression tree built to predict the normalized axial offset ΔI in a nuclear reactor core, based on the input features P, T_{avg}, $G1$ and $Boron$ (see section 3).

a response variable. The most popular algorithm for building classification and regression trees, namely Classification and Regression Trees (CART), was introduced by Leo Breiman et al. [13] in 1984. To apply it, we first need to construct a dataset $\mathcal{D}$ composed of $n \in \mathbb{N}$ points such that $\mathcal{D} = \{x_i^1, ..., x_i^p, y_i\}_{i\in[1,n]}$ where $\{x^1, ..., x^p\}$ are the input features and y is the response variable. Then, we partition the input space into smaller regions, with each region corresponding to a different prediction. At each step of the tree construction, the algorithm chooses the feature and the threshold that best splits the data into two subsets that are most different with respect to the corresponding value of the target variable. This process is repeated recursively until a stopping criterion is met, such as reaching a minimum number of samples in a leaf node or a maximum depth. The output of a regression tree is the predicted numerical value associated with the leaf node that corresponds to the input feature values (see Figure 1).

Random Forest. Algorithms using multiple regression trees were introduced to address some of the limitations of single regression trees such as overfitting and high variance in their predictions. One of the most popular one is an ensemble learning algorithm called Random Forest. It was first introduced by Tin Kam Ho in 1995 [14] and extended by Leo Breiman et al. [15] six years later. This algorithm, as introduced by Leo Breiman et al., works by creating a large number of regression trees, each trained on a random subset of the training data and a random subset of the features.

Input: Training dataset $\mathcal{D} = \{x_i^1, ..., x_i^p, y_i\}_{i\in[1,n]}$, number of trees $k \in \mathbb{N}$, data and features subsets sizes $l, m \in \mathbb{N}$
Output: Random forest $\mathcal{F}$
$\mathcal{F} \leftarrow \emptyset$
for $i \leftarrow 1$ *to* k **do**
  $D_i \leftarrow$ set of l points randomly drawn from $\mathcal{D}$
  $T_i \leftarrow$ regression tree trained on D_i while randomly selecting m possible splitting features for each split
  Add T_i to $\mathcal{F}$
return $\mathcal{F}$

Algorithm 1: Random Forest

The final prediction is made by aggregating the predictions of all the individual trees in the forest. The randomness in feature and data selection helps to reduce overfitting and increase the diversity of the trees, which in turn leads to better generalization and improved accuracy on test data.

Gradient Boosted Trees. In 2001, Friedman et al. introduced a new algorithm involving multiple regression trees: the Gradient Boosted Trees algorithm [16]. As classical boosting algorithms, it is based on the idea of iteratively adding weak learners - regression trees in this case - to the model, each correcting the

Input: Training dataset $\mathcal{D} = \{x_i^1, ..., x_i^p, y_i\}_{i\in[1,n]}$, differentiable loss function $\mathcal{L}$, number of iterations $M \in \mathbb{N}$
Output: Final model $F_M(x)$
Initialize the model with a constant value: $F_0(x) \leftarrow \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} \mathcal{L}(y_i, \gamma)$
for $m = 1$ *to* M **do**
  Compute the pseudo-residuals: $r_{im} = -\left[\frac{\partial \mathcal{L}(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$ for $i \in [1, n]$
  Fit a regression tree h_m to the pseudo-residuals: $h_m \leftarrow \underset{h}{\operatorname{argmin}} \sum_{i=1}^{n} \mathcal{L}(r_{im}, h(x_i))$;
  Compute the step size: $\gamma_m \leftarrow \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} \mathcal{L}(y_i, F_{m-1}(x_i) + \gamma h_m(x_i))$
  Update the model: $F_m(x) \leftarrow F_{m-1}(x) + \gamma_m h_m(x)$;
end
return F_m;

Algorithm 2: Gradient Boosted Trees

errors made by the previous learners. Here, the term **gradient** refers to the fact that the algorithm minimizes the loss function by computing the gradient of the loss with respect to the predictions returned by the model and updating it in this direction. Since each individual tree has been built to predict the residuals of the previous tree, the final prediction is simply obtained by summing the predictions of all the individual trees.

2.2. Time series forecasting

The time series forecasting problem can be considered as a particular type of regression problem. The aim is to predict the future values of a sequence based on its historical data. Various time series forecasting techniques have emerged over time such as ARIMA, Exponential Smoothing, Recurrent Neural Networks or Transformers [17]. We will specifically focus on the Persistence model and the ARIMA model in this study.

Persistence. The Persistence model is a naive model often used as a baseline for comparison with more complex models. It consists in using the last observed value of a time series as the prediction for the next time step. In other words, the model assumes that the future values of the series will be the same as the most recent past value. It can be represented by the following formula:

$$\text{Persistence}: \quad y_t = y_{t-1} \tag{1}$$

While this model is simple and may not always provide accurate predictions, it can be a useful benchmark for appreciating the performance of more sophisticated models.

Autoregressive Integrated Moving Average. The ARIMA model is a time-series forecasting model that was introduced in 1970 by Box et al. [18]. Since then, it has been widely used in various fields such as economics, finance, and engineering. The model has three components: the autoregressive (AR) component, the integrated (I) component, and the moving average (MA) component. The AR component models the dependence of the current value on the previous values, the MA component models the dependence on the previous errors, and the I component models the differencing needed to remove any trend or seasonal effects that may be present in the data. The general equation for the *ARIMA* model is the following:

$$\text{ARIMA}(p,d,q): \quad \phi_p(B)(1-B)^d y_t = \theta_q(B)\epsilon_t \tag{2}$$

where $\phi_p(B)$ is the autoregressive polynomial of order $p \in \mathbb{N}$, $\theta_q(B)$ is the moving average polynomial of order $q \in \mathbb{N}$, B is the backshift operator, $d \in \mathbb{N}$ is the degree of differencing, y_t is the time series and ϵ_t is white noise.

3. DATA

The data we collected are measurements from sensors of 20 PWRs belonging to the French 1300 MW nuclear fleet. From these data, we built 168 datasets, each representing an operating cycle of one of the reactors considered. The variables are the following: thermal power (as a percentage of the nominal power) P, boron concentration $Boron$, reference temperature (setpoint temperature defined according to the thermal power) T_{ref}, average core temperature T_{avg}, positions of the six banks of control rods $G1$, $G2$, $N1$, $N2$, $R1$, $R2$ and normalized axial offset ΔI. The thermal power P refers to the amount of heat energy produced by the fission of nuclear fuel in the reactor core. The power output is controlled by adjusting the number of free neutrons available to sustain the chain reaction. This is typically achieved by inserting or withdrawing the six banks of control rods mentioned - which are made of neutron-absorbing materials - from the reactor core. Boron is also used in the coolant to control the overall reactivity of the reactor and maintain a stable power output. Given that the acquisition frequency of the sensors that capture the operating signals was not the same for each variable, we standardized the dataset temporally. Specifically, we chose a uniform sampling frequency of one minute to simplify data manipulation and analysis.

4. PROBLEM FORMULATION

Notations. The following notations are used in this section: t_0 refers to the current instant; H is the maximum prediction horizon; y_t refers to the normalized axial offset value at time t; $\mathbf{x}_t$ is the vector of

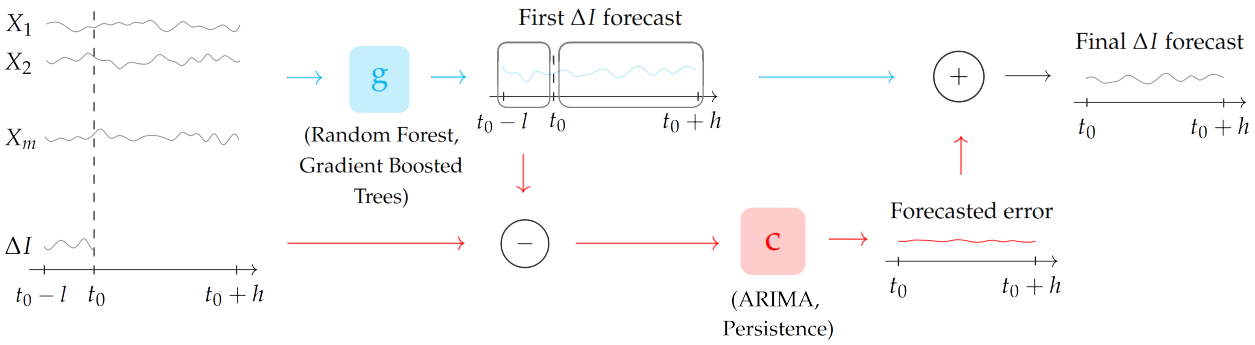

Figure 2: Illustration of the proposed solution for forecasting the normalized axial offset. The final ΔI forecast is obtained by combining g and c predictions.

input features values mentioned in section 3 at time t; l is the number of previous time steps that contain relevant information and ϵ_t is white noise.

While performing load-following operations, operators in the control room must monitor and control the normalized axial offset ΔI, defined by the difference between the thermal power generated in the top half and that in the bottom half of a core reactor in the axial direction as a percentage of the nominal power:

$$\Delta I = \frac{P_t - P_b}{P_{nom}} = \frac{P_t - P_b}{P_t + P_b} \times P_r = AO \times P_r \tag{3}$$

where P_t and P_b denote the thermal power generated in the top and bottom halves of the core, respectively, P_{nom} is the nominal power, P_r is the relative core thermal power and AO is the axial offset. The actions performed by the operators can have a direct impact (e.g.: power drop in the top half of the core caused by the insertion of a neutron-absorbing control rod) or an indirect one (e.g.: Xenon oscillation indirectly caused by the insertion of a control rod) on the ΔI in the long term. It is therefore very useful for them to know how ΔI will evolve over the next hours as a result of their actions. Thus, we seek to estimate the function f such that:

$$\forall h \in [1, ..., H], \quad \widehat{y}_{t_0+h} = f(y_{t_0-l}, ..., y_{t_0-1}, \mathbf{x}_{t_0-l}, ..., \mathbf{x}_{t_0+h}) + \epsilon_{t_0+h} \tag{4}$$

5. PROPOSED METHODOLOGY

Hybrid method. We propose a hybrid method to address this problem. First, we use a tree-based model (Random Forest or Gradient Boosted Trees) to predict the future values of ΔI given the future values of the input features, then we use a time series model (Persistence or ARIMA) to forecast the tree-based model residuals (Figure 2). Thus, we learn two functions g and c such that:

$$\forall h \in [1, ..., H], \quad \widehat{y}_{t_0+h} = g(\mathbf{x}_{t_0+h}) + c((y_t - g(\mathbf{x}_t))_{t \in \{t_0-l, ..., t_0\}}) \tag{5}$$

where g is the tree-based model and c is the time series model mapping past g residuals to g residual at time $t_0 + h$.

Selection of the tree-based model g. The first step of our approach is to select the tree-based model that best capture the instantaneous information in the training dataset. To this end, we start by optimizing the hyperparameters of both the Random Forest and the Gradient Boosted Trees models implemented in the Python **xgboost** package. The best hyperparameter values are estimated by defining a prior distribution for each hyperparameter first, then performing a Bayesian Search based on the Tree-structured Parzen Estimator Approach [19]. At each iteration, the candidate (set of hyperparameter values) is evaluated by averaging the Root Mean Squared Errors (RMSE) over every validation sets. Figure 3 reveals that the performance of the best Random Forest and the best Gradient Boosted Trees models found with

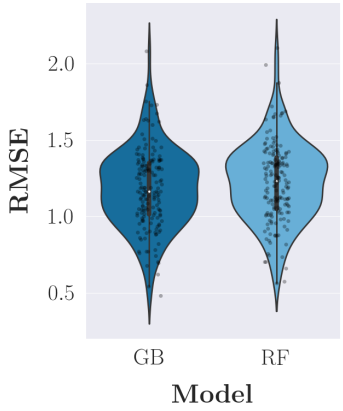

Figure 3: Violin plot showing the distribution of RMSEs for the optimized Gradient Boosted Trees (dark blue) and Random Forest (light blue) models on every validation datasets. Each black dot represents the RMSE for a specific validation dataset. The width of the violin indicates the density of scores at that level.

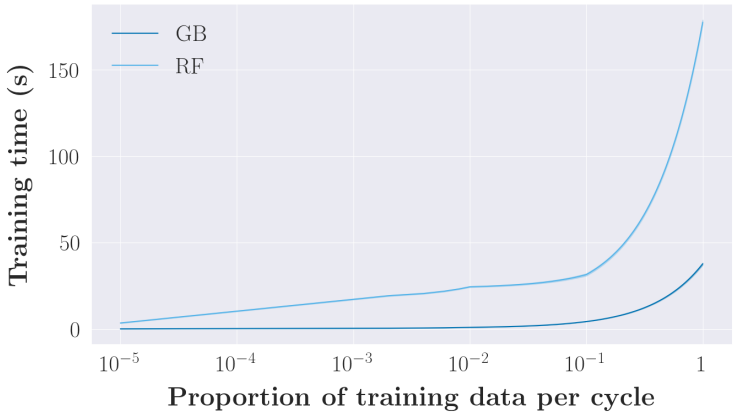

Figure 4: Training times of the optimized Gradient Boosted Trees (dark blue) and Random Forest (light blue) models for different training dataset sizes. The x-axis represents the proportion of training data selected from each dataset to form the final training dataset, while the y-axis represents the training time in seconds. Each curve depicts the average training time for a particular model, with a shadow indicating the standard deviation across 168 runs.

this method is comparable. Besides comparing the performance of the two tree-based models, we compare their training times. To achieve this, we form multiple training datasets by randomly selecting a proportion $p \in \{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1\}$ of training data per cycle (see section 3 for further details on the data), we train the two tree-based models on those training datasets with a Tesla V100 SXM2 32GB GPU and record their training times. Since Gradient Boosted Trees exhibit significantly faster training times, as illustrated in Figure 4, we only use this tree-based model to predict ΔI in the subsequent study.

Selection of the *ARIMA* model. To select the optimal *ARIMA* model for predicting *GB* residuals, we use the Python *AutoARIMA* function implemented in the **pmdarima** package [20]. The function performs several steps to automatically select the best *ARIMA* model. First, it checks for stationarity in the input time series data using the Augmented Dickey–Fuller (ADF) test. If the data is not stationary, the function applies differencing to make it stationary. Next, it searches for the best *ARIMA* model by evaluating a range of models based on their Akaike Information Criterion (AIC). Once the optimal model is selected, the function performs model diagnostics to check the normality of its residuals using the Ljung-Box test. If the model fails these tests, the function searches for a new best model.

Training and prediction phases of the hybrid model. To predict the target variable ΔI using the proposed hybrid model on one of the 168 validation datasets constructed (see Section 3), we first train the Gradient Boosted Trees model on all the datasets except the validation one. We then split the validation dataset into as many H minutes windows as possible. As shown on Figure 2, for each window, we compute the last l *GB* residuals, train c on those residuals, and predict future residuals for the next H minutes. The **pmdarima** package is used to identify the best *ARIMA* model that fits the last l residuals. The Persistence model does not require fitting, it simply computes the last known *GB* residual and propagates it on the full prediction horizon. The final ΔI forecast is eventually obtained by adding the H residuals predicted by c to the initial *GB* prediction.

6. RESULTS

Notation. The following notation is used in this section: *GB +c* refers to *GB* combined with the time series model c.

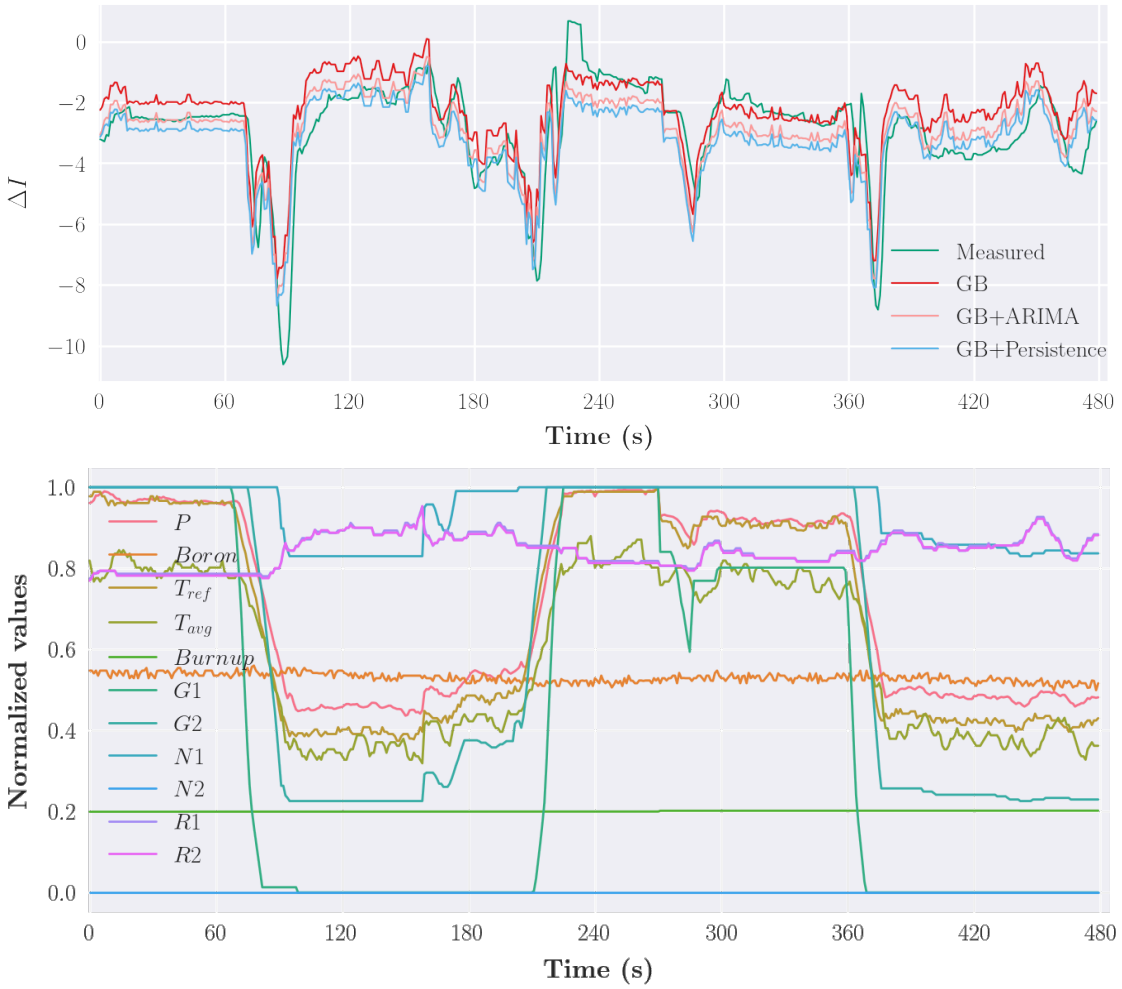

Figure 5: The top figure displays ΔI forecasts (represented by red, pink and blue curves) generated by *GB* , *GB+ARIMA* and *GB+Persistence* , respectively, for a validation power transient. The green line corresponds to the measured ΔI values. The bottom figure represents the normalized input feature values over time for this validation power transient.

Our objective was to be able to compute minute-by-minute forecasts of ΔI for the next eight hours, providing operators with enough time to take corrective measures if ΔI deviated from its set point.

Case study. Here, we show how our models behave when trying to forecast ΔI during a load following operation (Figure 5). This power transient includes a decrease of the core power from 100% to 50%, constant power at 50% for 2 hours, a return to full power in 30 minutes and another decrease of the core power from 100% to 50% in two steps. The eight-hour-long forecasts produced by *GB* , *GB+Persistence* and *GB+ARIMA* on this power transient are shown at the top of Figure 5. As we can see, the predicted values (in red, pink and blue) are very close to the actual ones (in green), indicating that our model is performing well on this transient. The root mean squared error (RMSE) between the predicted and actual values is only 1.15 for *GB* , 1.07 for *GB+Persistence* and 1.01 for *GB+ARIMA* , which is much lower than the uncertainty measurement of ΔI, that is around 2. Note that it only took 0.129 seconds for *GB+ARIMA* , which is our slowest model, to compute this forecast.

Reliability of the proposed hybrid model. The results we obtained when comparing the performance of *GB*, *GB+Persistence* and *GB+ARIMA* on the entire dataset are displayed on Figure 7. We produced them by performing a custom cross validation procedure depicted in Figure 6. Knowing that the uncertainty measurement of ΔI is about 2, our goal was to obtain errors lower than 2 for every prediction horizon. As

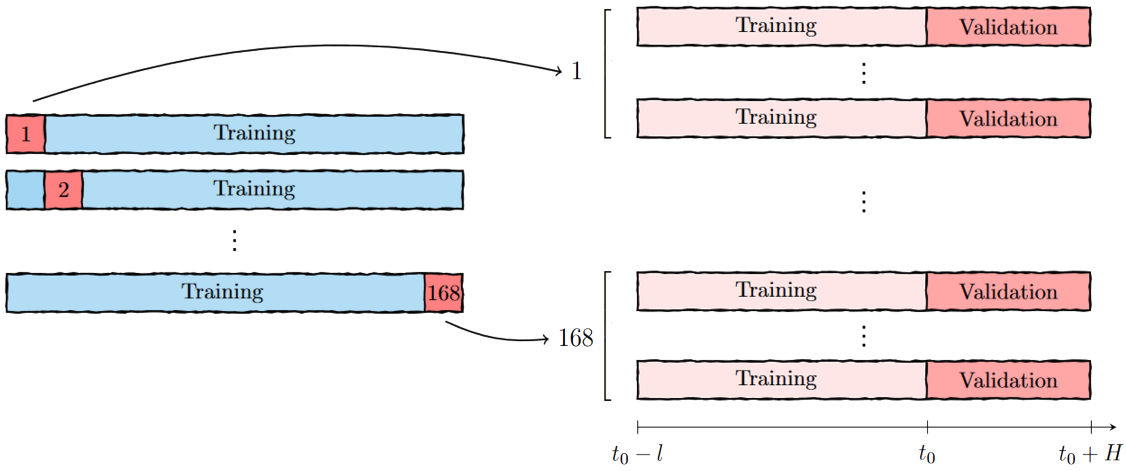

Figure 6: Illustration of the cross-validation procedure performed in this study. For each validation dataset $\mathcal{D}_i$, with $i \in [1, ..., 168]$, the *GB* model is trained on the blue part which corresponds to the 167 remaining datasets. For every H-minutes window in $\mathcal{D}_i$, the *ARIMA* model is trained on the l previous timesteps represented in light pink. The hybrid model, which combines the predictions of the *GB* model and the *ARIMA* or Persistence model is evaluated on the light red part in the far right.

we can see on Figure 7, we achieved this on the majority of the validation data - data points that reside below the upper whisker of the boxplot - with our two hybrid models *GB+ARIMA* and *GB+Persistence* until the 2 hour horizon. To the best of our knowledge, this is the first study conducted using real-world data on flexible nuclear power plants operations, with EDF being the sole owner of such data. As a consequence, we were unable to compare our results with other studies, which were carried on simulated or research reactor data.

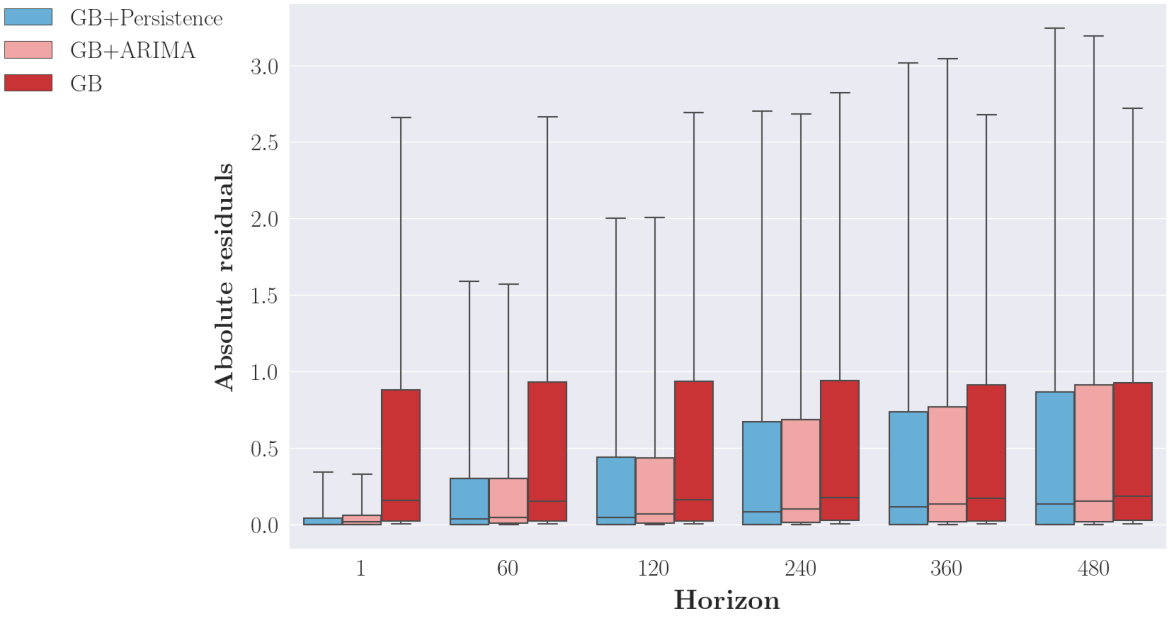

Figure 7: Absolute residuals of *GB* , *GB+Persistence* and *GB+ARIMA* on the entire dataset. These residuals were obtained for every horizon ranging from 1 to 480 with our cross validation procedure but only the results for horizons 1, 60, 120, 240, 360 and 480 are displayed here.

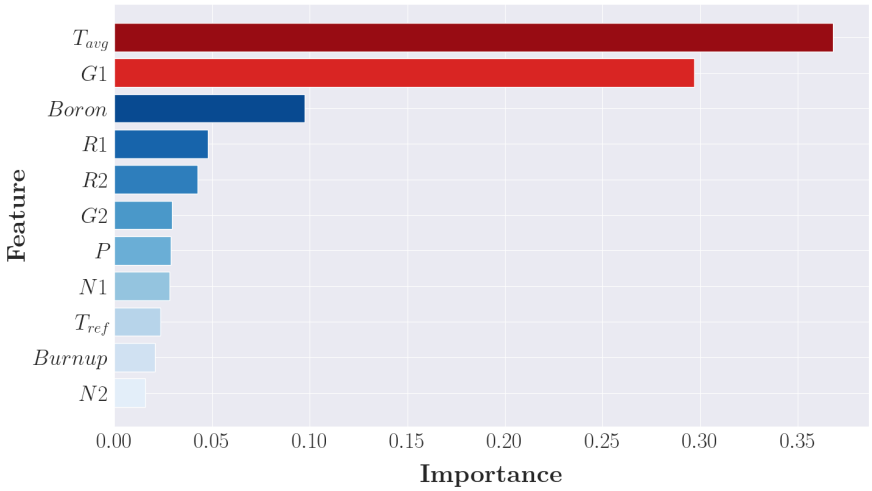

Figure 8: Feature importance plot showing that T_{avg} and $G1$ are the most important features in predicting the normalized axial offset for the Gradient Boosted Trees model.

The feature importance plot generated by the Gradient Boosted Trees model (Figure 8) indicates that the average temperature and the position of the $G1$ control rods bank are the two most important features in predicting the normalized axial offset. This suggests that these two variables have a strong relationship with the target variable. In fact, ΔI is highly dependent of P, while T_{avg} and $G1$ are both functions of P in the real-world system. As such, the results are in line with the real behavior of the system, which highlights the effectiveness of the Gradient Boosted Trees model in capturing the underlying relationships between the input features and ΔI. However, as we can see on figures 5 and 7, the models' predictions do not always align with the actual data. This can be attributed to the fact that the models are not capturing the complex temporal relationships between the input features and the output variable. Alternatively, there may be important variables that are not included in the model, which could prevent them from performing well in specific situations. Another limitation of our models is the closed-loop effect induced by the fact that the operators perform actions to limit the axial-offset excursion, such as moving the control rods or injecting boron. A part of the natural evolution of the axial offset is then contained in the behavior of these actuators. Future work should aim to address these issues in order to improve the overall performance and reliability of the model.

7. CONCLUSIONS

In this work, we describe a novel approach combining tree-based and time series models to forecast the normalized axial offset. Using a large dataset of commercial nuclear power plants operating data, we demonstrate the efficiency and speed benefits of our approach. To the best of our knowledge, evaluating the performance of such methods on a real dataset of this size has not been reported in the literature before. In future works, we plan to enhance our models by incorporating physics knowledge, such as the Bateman equations, to estimate Xenon concentration. Additionally, we aim to compare our approach with deep learning models, including LSTM, GRU, Transformer or recently proposed latent variable models such as variational diffusion models, both in terms of efficiency and explainability. Finally, to improve the reliability of our solution, we are considering to investigate recent and promising probabilistic forecasting techniques such as conformal prediction.

ACKNOWLEDGEMENTS

We would like to thank EDF R&D and the French ANRT doctoral program for supporting this work. We also thank Mathieu DAGRÉOU and Margaux ZAFFRAN for their initial work, which led to this paper.

REFERENCES

[1] P. Morilhat, S. Feutry, C. Le Maitre, and J. M. Favennec. "Nuclear power plant flexibility at EDF." (2019). URL https://hal-edf.archives-ouvertes.fr/hal-01977209.

[2] C. Cany, C. Mansilla, G. Mathonnière, and P. Da Costa. "Nuclear power supply: Going against the misconceptions. Evidence of nuclear flexibility from the French experience." *Energy*, **volume 151**, pp. 289–296 (2018).

[3] S. Bouckaert, A. F. Pales, C. McGlade, U. Remme, and B. Wanner. "Net Zero by 2050. International Energy Agency." https://www.iea.org/reports/net-zero-by-2050 (2021). Accessed: 2022-09-30.

[4] A. J. Davidson. *The Role of Nuclear Energy in the Global Energy Transition* (2022).

[5] G. Li, X. Wang, B. Liang, and al. "Modeling and control of nuclear reactor cores for electricity generation: A review of advanced technologies." *Renewable and Sustainable Energy Reviews*, **volume 60**, pp. 116–128 (2016).

[6] R. T. Kozma, S. Sato, and al. "Generalization of knowledge acquired by a reactor core monitoring system based on a neuro-fuzzy algorithm." *Progress in Nuclear Energy*, **volume 29**, pp. 203–214 (1995).

[7] H. G. Kim, S. H. Chang, and B. H. Lee. "Pressurized water reactor core parameter prediction using an artificial neural network." *Nuclear Science and Engineering*, **volume 113**(1), pp. 70–76 (1993).

[8] T. Adali, B. Bakal, M. Sönmez, and al. "Modeling nuclear reactor core dynamics with recurrent neural networks." *Neurocomputing*, **volume 15**(3), pp. 363–381 (1997).

[9] M. Boroushaki and al. "Identification of a nuclear reactor core (VVER) using recurrent neural networks." *Annals of Nuclear Energy*, **volume 29**(10), pp. 1225–1240 (2002).

[10] F. Cadini, E. Zio, and N. Pedroni. "Simulating the dynamics of the neutron flux in a nuclear reactor by locally recurrent neural networks." *Annals of Nuclear Energy*, **volume 34**, pp. 483–495 (2007).

[11] A. Naimi, J. Deng, and al. "Dynamic Neural Network-based System Identification of a Pressurized Water Reactor." In *2020 8th International Conference on Control, Mechatronics and Automation (ICCMA)*, pp. 100–104 (2020).

[12] R. Shwartz-Ziv and A. Armon. "Tabular data: Deep learning is not all you need." *Information Fusion*, **volume 81**, pp. 84–90 (2022). URL https://www.sciencedirect.com/science/article/pii/S1566253521002360.

[13] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. "Classification and Regression Trees." (1984).

[14] T. K. Ho. "Random decision forests." *Proceedings of the 3rd International Conference on Document Analysis and Recognition*, pp. 278–282 (1995).

[15] L. Breiman. "Random forests." *Machine learning*, **volume 45**(1), pp. 5–32 (2001).

[16] J. H. Friedman. "Greedy Function Approximation: A Gradient Boosting Machine." *The Annals of Statistics*, **volume 29**(5), pp. 1189–1232 (2001).

[17] B. Lim and S. Zohren. "Time-series forecasting with deep learning: a survey." *Philosophical Transactions of the Royal Society A*, **volume 379**(2197), p. 20200209 (2021).

[18] G. E. Box and G. M. Jenkins. "The Box-Jenkins approach to time series analysis." *Journal of the Royal Statistical Society, Series B (Methodological)*, **volume 32**(2), pp. 171–212 (1970).

[19] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. "Algorithms for hyper-parameter optimization." *Advances in neural information processing systems*, **volume 24** (2011).

[20] S. J. Taylor, J. K. Lefort, and C. H. Langford. "pmdarima: ARIMA and Seasonal ARIMA Model Selection in Python." (2021). URL https://github.com/statsmodels/statsmodels. Software package.