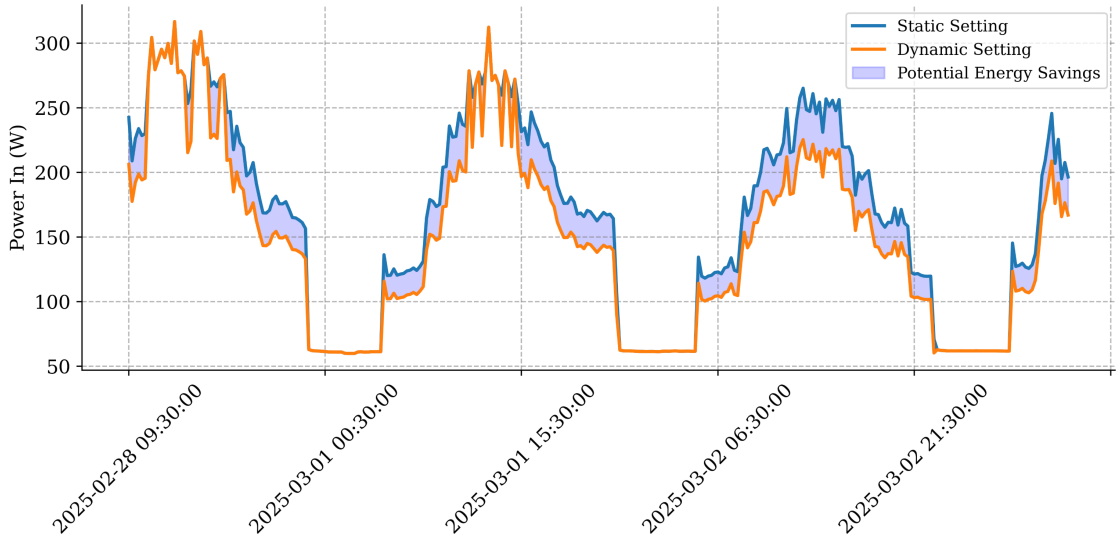


Dynamic Voltage Optimization for Energy Efficient Radios

Comparing Problem Formulations for Adjusting Voltage in Radios Using Temporal Patterns

Master's thesis in Data Science and AI

CECILIA NYBERG
ELIN STIEBE

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025

Master's thesis 2025

Dynamic Voltage Optimization for Energy Efficient Radios

Comparing Problem Formulations for Adjusting Voltage in Radios Using Temporal Patterns

CECILIA NYBERG
ELIN STIEBE

Department of Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden 2025

Dynamic Voltage Optimization for Energy Efficient Radios
Comparing Problem Formulations for Adjusting Voltage in Radios Using Temporal Patterns
CECILIA NYBERG, ELIN STIEBE

Supervisor: Linus Aronsson, Computer Science and Engineering
Advisor: Thomas Lejon, Thord Hyllander, Ericsson
Examiner: Morteza Haghir Chehreghani, Computer Science and Engineering

Master's Thesis 2025
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: The image illustrates energy savings resulting from dynamic voltage level switching in a radio unit.

Typeset in LaTeX
Gothenburg, Sweden 2025

Dynamic Voltage Optimization for Energy Efficient Radios
Comparing Problem Formulations for Adjusting Voltage in Radios Using Temporal Patterns
Cecilia Nyberg
Elin Stiebe
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Reducing energy waste in radios is essential for lowering the CO_2 footprint of cellular connectivity. In current systems, radios operate at a static high voltage regardless of their physical resource block utilization (PRB-U), resulting in inefficiencies. This thesis proposes a machine learning (ML)-based approach to enable dynamic voltage control and evaluates which models and task formulations are most suitable for this purpose. Eight ML models were tested across two formulations: (1) classification of voltage levels and (2) regression to predict PRB-U values and map them to voltage levels.

The results show that ML can significantly reduce energy waste in certain radios, although effectiveness varies by device - some perform comparably using simpler methods. Classification outperformed regression in reducing voltage underestimations.

The Random Forest (RF) classifier and a customized Feedforward Neural Network (FFNN) classifier emerged as top performers. The FFNN achieved an F1 score of 0.35, saved 12.41% energy, and resulted in 34 underestimations. The RF reached an F1 score of 0.29, saved 12.43%, and had 42 underestimations. In contrast, the best-performing baseline model had an F1 score of only 0.19 with 55 underestimations out of 68, underscoring the benefit of ML-based approaches.

Shifting the classification threshold helped manage the trade-off between energy savings and underestimations. Notably, the FFNN achieved just 5 underestimations while maintaining 10% energy savings.

SHAP-based explainability analysis showed that PRB-U at the current timestep was the most influential feature. The RF leveraged a broader feature set, while the FFNN focused more narrowly on a dominant input.

In conclusion, this thesis demonstrates that intelligent, ML-driven voltage control can enhance energy efficiency in radios. It also emphasizes the importance of balancing energy savings and prediction errors during model deployment, potentially by adjusting class probability thresholds.

Keywords: AI, dynamic voltage, energy efficiency, machine learning, optimization, physical resource block utilization, radio, sustainability, time series, Optuna, SHAP, traffic load prediction.

Acknowledgements

We would like to thank our supervisor *Linus Aronsson* for his support in forming the outline of this thesis as well as providing concrete input on our work and implementation. Furthermore, we are grateful to our examiner *Morteza Haghir Chehreghani* who had extensive meetings with us and was an immense help in structuring the scope of the project and contributing with discussions.

We are grateful to our supervisors at Ericsson *Thomas Lejon* and *Thord Hyllander* who gave continuous input and gave us a deeper understanding of radio functionality and development. Furthermore, we are grateful for the topic of this thesis first laid out by *Thomas Lejon*. We also want to thank *Johan Seger* for welcoming us to his team and inviting us to several educational occasions such as bringing us into a radio algorithm testing lab. Lastly, thank you *Jing Liu B* and other colleagues within Radio Systems and Technology for guiding us within your respective fields.

Cecilia Nyberg, Elin Stiebe, Gothenburg, 2025-06-11

Contents

List of Figures

List of Tables

Glossary

4G 4th generation of network technology.

5G 5th generation of network technology.

AI Artificial intelligence.

Cell Specific section of a geographical area covered by a radio.

CNN Convolutional neural network - ML architecture.

CO_2 The green house gas carbon dioxide.

DL Deep Learning.

Energy Measured in kilo watt hours (kWh).

FFNN feedforward neural network - ML architecture.

TFT Temporal fusion transformer neural network - ML architecture.

KNR/KNC K-neighbors regression/classification - ML architecture.

LSTM Long short term memory neural network - ML architecture.

ML Machine learning.

Radio Receiver and transmitter of radio signals.

RF Random forest - ML architecture.

Power Measured in Watts (W).

$\mathbf{P}_{in}$ Power in to a radio (W).

$\mathbf{P}_{out}$ Power used for radio transmission (W).

Voltage Measured in volts (V).

PRB Physical resource block.

PRB-U PRB utilization

SVR/SVC Support vector regression/classification - ML architecture.

XGB Extreme gradient boosting - ML architecture.

1

Introduction

The Paris Agreement is one of the most significant milestones in climate policy making, adopted on December 12, 2015 [1]. This legally binding treaty, established under the United Nations, outlines in Article *2a* the goal of "holding the increase in the global average temperature to well below 2°C above pre-industrial levels and pursuing efforts to limit the increase to 1.5°C". January 2025 was the warmest ever recorded with temperature levels 1.75°C above pre-industrial levels [2]. The month was one of the 18 out of the past 19 months where global temperatures exceeded 1.5°C above pre-industrial levels.

The Information and Communication Technology (ICT) sector accounts for 4% of global energy consumption during its use phase and 1.4% of greenhouse gas emissions in 2020 [3]. These figures do not account for emissions from the development or end-of-life phases. The environmental impact of ICT is closely tied to the source of electricity. For instance, in 2023, Sweden had an estimated carbon intensity of 41 gCO_2 per kWh, in contrast to Germanys 381 and Chinas 582 gCO_2 per kWh [4]. In addition to environmental concerns, energy use also represents a major cost for telecom operators estimated at up to 5% of revenue [5]. This dual incentive, with environmental and financial factors emphasize the importance of energy-efficient solutions in ICT.

Machine learning (ML) has emerged as a powerful tool to improve sustainability by predicting patterns and optimizing energy usage. In integrated energy systems, which connect multiple energy sources, ML has shown potential in forecasting and optimizing operations to achieve net-zero CO_2 emissions [6]. Similarly, in the context of smart homes, ML models have been used to segment households based on their energy usage, allowing for more tailored and effective energy management strategies [7]. Another promising application is dynamic power management by automatically shutting down idle devices. Studies show that ML can successfully determine optimal shut-down strategies based on the state of a device, leading to substantial energy savings [8].

Time series forecasting is a field within ML that has advanced rapidly in recent years, supported by the increasing availability of data [9]. It is widely applied in fields such as healthcare, finance, and industrial production [9], and has also shown promise in energy efficiency applications [10]. A wide range of methods exist, from statistical models to machine learning techniques [11].

This thesis contributes to the area of sustainable ML-driven energy optimization by proposing a dynamic voltage strategy for Ericssons radio units using time series data. While Ericsson already implements features like sleep mode, allowing radios to be turned off during low traffic hours, this thesis focuses on periods when radios must remain active but are not required to operate at full capacity. By providing a strategy for dynamically adjusting voltage levels in real-time based on demand, this research aims to reduce power consumption and, by extension, carbon emissions.

1.1 Problem Statement

Each radio operates at a preconfigured power level, determined by the voltage setting of its power amplifier, which directly affects its power consumption. Currently, these units are always set to their maximum voltage to accommodate peak demand at any time. However, this approach leads to excessive energy consumption, resulting in both environmental impact and higher operational costs for customers.

This thesis addresses the energy inefficiency by developing a machine learning-based methodology capable of predicting power demand for the next time step. The goal is to enable dynamic voltage adjustment between predetermined levels in radio units, ensuring that power levels are aligned with actual utilization. By intelligently scaling voltage in response to predicted demand, the solution aims to reduce unnecessary energy consumption without compromising performance.

1.2 Aim

The aim of this thesis is to develop a strategy for reducing excess energy consumption in radio products by predicting usage patterns through *time series forecasting*. Based on these predictions, the system can dynamically adjust the voltage settings of radio units in real time to align with actual demand. This approach seeks to improve energy efficiency, lower operational costs, and contribute to broader sustainability goals.

The project proceeds in four main stages. First, it investigates whether machine learning-based methods offer performance advantages over simpler statistical approaches for the specific task. Since machine learning models are generally more complex and resource-intensive, simpler methods would be preferred if they yield comparable results.

Second, the project compares two possible formulations of the prediction task: regression and classification. Since the voltage can only be set to a fixed number of discrete levels, it is not immediately clear which approach is more suitable. In the classification setting, the model predicts the voltage level directly as a class label, whereas in the regression setting, a continuous value is predicted and mapped to the nearest voltage level. Effectively all versions work to classify the voltage levels, though they are optimized under different conditions. Including both formulations allows this thesis to evaluate which approach yields better performance for the given

application.

For regression, hyperparameters are tuned using two separate objectives — Mean Squared Error (MSE) and F1 score — to better address class imbalance. All models are optimized using Optuna [12], an advanced hyperparameter tuning framework [11]. The length of historical input data is treated as a tunable hyperparameter, enabling investigation of how its role varies across task formulations.

Third, the thesis compares the models predictive performance to identify the best-performing architecture for the task. This includes an analysis of different architectural configurations to determine which components contribute most to performance. Additionally, the optimal hyperparameters obtained through Optuna tuning are examined for each model to gain insight into configurations associated with strong performance. As the models are intended for deployment in a resource-constrained environment, the complexity of the tuned architectures is also evaluated.

Lastly, the thesis investigates which variables are most important for determining the voltage level in radios. This is done using SHAP [13] to analyze the best-performing models.

The contribution of this thesis will be made through answering the following research questions:

- **RQ1**: Are machine learning techniques useful for dynamic adaptation of voltage in radios?
- **RQ2**: What are the advantages and disadvantages of using classification versus regression in the case of energy-efficient radios?
  - **RQ2.1:** Given the class imbalance, should the hyperparameters for regression models be tuned using MSE or F1 score?
  - **RQ2.2:** How is historical information handled differently across the approaches?
- **RQ3**: Which machine learning models are most beneficial in terms of energy saving and optimal connectivity?
- **RQ4**: Which input variables are most influential in determining voltage levels in energy-efficient radios? This is investigated through explainability analyses of the top-performing models.

1.3 Limitations

The goal of reducing energy consumption in radios is inherently constrained to the dynamic portion of total energy use. As discussed later in Section 2.1, a percentage of the power consumption is static and does not depend on traffic load. Furthermore, this thesis only evaluates two levels of voltage as this is easier to implement in radio hardware. An assumption for the lower level is that a reduction in voltage level by 50% corresponds to an approximate 15% decrease in power. This is further

discussed in Section 2.1.2. Further research is encouraged to more precisely quantify the relationship between voltage decrease and a shift in power across different operational settings and radios. This thesis relies on several other theoretical and mathematical assumptions. While these are clearly stated in the relevant sections, it should be noted that such assumptions may introduce minor discrepancies, as real-world behavior rarely aligns perfectly with idealized models.

Moreover, the primary focus of this study is on single-cell radio systems. While the multi-cell scenarios are described, it is not part of the investigation of this research. Hence, the generalizability of the proposed method will be discussed and evaluated as part of the concluding analysis.

Additionally, the training and validation data correspond to five selected radios' recorded during the time period from January 17, 2025, to March 6, 2025. The test data is retrieved from March 8 2025 until April 1 2025. The 7th of March was excluded to avoid creating any overlap between the training and testing phase as the time series data has a lag and hence has knowledge about data back in time. This fact is explained in more detail in Section 2.3. The selection of time intervals to train and test the models within introduce a limitation as there is no way of concluding the models' effectiveness in another month or year. Similarly, the models were selected as a subset of options. This thesis only examines the performance on eight potentials on the task. It is possible that other ML based approaches that were not included in this thesis could perform better.

1.4 Thesis outline

- **Chapter 1** Sets the background of this thesis, introduces relevant literature, describes its limitations and states the research questions.
- **Chapter 2** Contains theory on the radio context, data and different ML architectures. It further introduces the Optuna framework used for tuning all of the models.
- **Chapter 3** Describes the methodology used to answer the research questions through visualizing the data, describing the hyperparameter tuning and evaluation tactic of this thesis.
- **Chapter 4** Shows the result of this thesis through the evaluation results of the models and relevant statistics tied to those results.
- **Chapter 5** Discusses the thesis' results, reliability and limitations, and addresses the research questions.
- **Chapter 6** Answers the research questions and states the conclusions of this thesis.

2

Theory

This chapter dives into the theory and background knowledge needed to follow the problems solved within this thesis. Section 2.1 describes the radio functionality. All facts in Section 2.1 are based on Expert knowledge within Ericsson. Its subsection 2.1.1 discusses the knowledge needed within electricity and 2.1.2 visualizes and makes the potential saving more concrete. Later Sections dive into technical background for the ML and methodology of this thesis, Section 2.2 discusses general ML theory, Section 2.3 discusses time series data which is used in this thesis and Section 2.4 brings up theory about the investigated models. Section 2.5 describes the hyperparameter tuning conducted with *Optuna*, and lastly 2.6 describes the SHAP method, used to analyze and add explainability to some models' outputs.

2.1 Radio Functionality

Radios receive digital signals from a *baseband unit*, process them, and transmit analog signals to enable telecommunication services such as 4G or 5G. Each radio is configured for specific capabilities, including the frequency range referred to as the *band* in which it operates. These frequency allocations vary by country, depending on regulatory constraints. Fig. 2.1 illustrates possible intervals of frequency bands. Lower generations of mobile technology typically operate in lower frequency ranges, while newer generations utilize a broader spectrum, expanding both upward and downward. Lower frequencies get less disturbances from physical obstructions such as walls, making them more suitable for indoor coverage. For this reason, many operators prioritize frequencies between 700 MHz and 2 GHz.

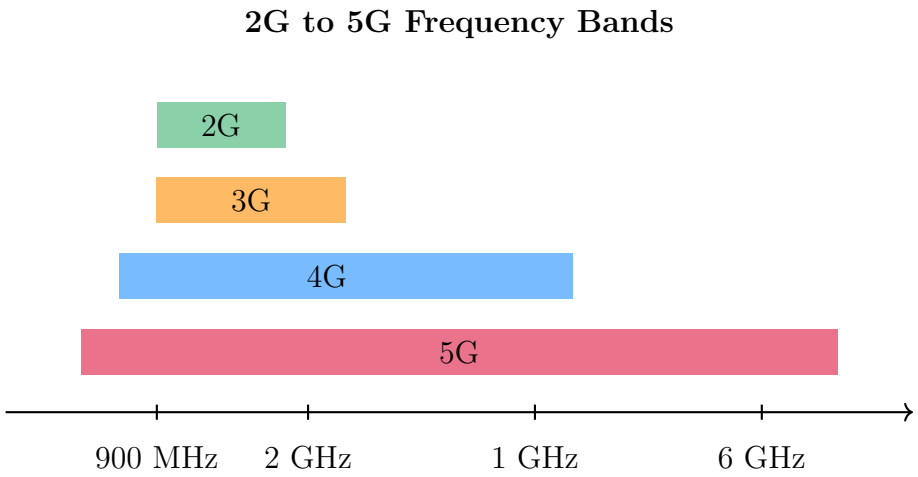


Figure 2.1: 2G to 5G Frequency Bands.

The target variable used in this thesis is *Physical Resource Block utilization* (PRB-U), which is the extent to which a radios PRB is utilized. Fig. 2.2 shows the physical resource block for LTE (4G) cells depicted by [14]. A PRB is a number of symbols in the time domain and a number of subcarriers in the frequency domain [15], the symbol and sub-carrier can be seen to the upper right in Fig. 2.2. For the purposes of this study, the key insight is that PRB-U reflects the proportion of a radio's total data-handling capacity that is currently in use.

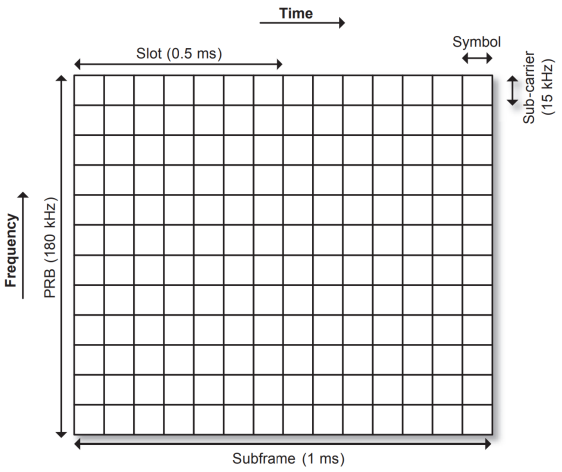


Figure 2.2: Physical resource block for LTE cells (4G) [14].

The total power consumption of a radio depends on several factors, of which only a subset is related to its utilization. This thesis specifically targets the power which can be dynamic due to user patterns. The portion of power consumption attributed to utilization is referred to as P_{out} throughout this thesis and is assumed to be fully proportional to the radio's utilization, as described in Eq. 2.1. The term $P_{\text{out max}}$ denotes the maximum power that can be allocated for utilization, and this

value varies between radio units. For the specific radio examined in this thesis, $P_{\text{out max}} = 160\,\text{W}$.

$$P_{\text{out}} = P_{\text{out max}} * \text{PRB-U} \tag{2.1}$$

Each radio unit can consist of multiple cells, where each cell is individually configured and acts as a connection point for users. The total number of users connected to a radio is the sum of users across all its cells. Each cell has its own allocation of Physical Resource Blocks (PRBs) and consumes power based on two components: the traffic-dependent power and a constant base power required for other cell operations. This relationship is expressed in Eq. 2.2.

$$\text{Power}_{\text{cell}} = \sum \text{Power}_{\text{traffic}} + \text{Power}_{\text{other cell operations}} \tag{2.2}$$

The energy consumption of radios are driven by activities in its cells, as well as by internal operations and communication with other network elements. This relationship is described in Eq. 2.3.

$$Power_{radio} = \sum_{cells \in Radio} Power_{cells} + Power_{\text{other radio operations}} \tag{2.3}$$

Radios may have several cells from different technology generations (such as having both 4G and 5G), which behave differently in terms of energy efficiency. The 5G cells are typically more energy-efficient during stand-by hours of lower overall usage. However, since the radio's total power is the sum of all its cells, such generational nuances get lost at the aggregated radio level.

Furthermore, customers have the option to activate sleep mode on their radio units, activation makes specific cells within the radio disabled during predetermined hours. This capability is demonstrated later in Fig. 3.3, where utilization rates drop to zero during nighttime and early morning periods. As all radios analyzed in this study have sleep mode activated, the potential for additional energy savings on these units is likely diminished compared to radios without sleep mode, given that nighttime utilization is inherently low. Additionally, the application of dynamic voltage scaling does not yield further energy-saving benefits during intervals when sleep mode is engaged.

2.1.1 Electronics

The aim of this thesis is to develop a dynamic voltage strategy that adapts to user utilization levels. As previously described in Eq. 2.3, the total energy consumed by a radio includes not only the power used by the cells that connect users but also the energy required for other radio operations. Therefore, the input power to the radio, denoted as P_{in}, is not solely determined by PRB utilization (PRB-U) and cannot be entirely regulated based on utilization alone.

As introduced in Section 2.1, the maximum transmission power output from the radio, P_{out}, is 160 W, while the total input power, P_{in}, is 600 W. This yields a maximum power conversion efficiency of approximately 26.7%. The remaining 73.3% of the energy is consumed by other operational needs within the radio or is lost as heat. Fig. 2.3 provides a simplified illustration of the relationship between the input power, P_{in}, and the output power, P_{out}, where the latter scales with user utilization.

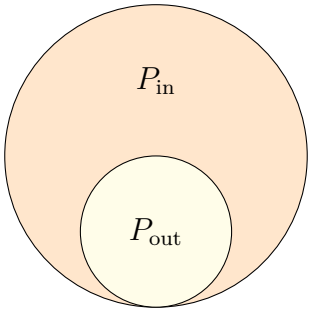


Figure 2.3: Venn diagram of the full power (P_{in}) and the subset of power used for user connectivity (P_{out}).

As previously shown in Eq. 2.1, P_{out} is directly dependent on the radio's utilization, consequently P_{out} is not a tunable parameter. Similarly, the power capacity P_{in} is fixed. However, the voltage level is adjustable, and targeted throughout this thesis. Voltage, P_{out}, and resistance R are closely linked through the relationship in Eq. 2.4. Although resistance may vary slightly due to factors like temperature increases when the radio is active, it is assumed constant in this context. The relationship can thus be simplified as shown.

$$P_{\text{out}} \propto U^2/R \propto U^2 \tag{2.4}$$

Eq. 2.4 gives the relationship between voltage adjustment and P_{out} capabilities. The change in P_{out} with a 50% voltage is as shown in Eq. 2.5. Such an adjustment decreases P_{out} by a factor of $\frac{1}{4}$. An important thing to note is that, setting P_{out} capabilities too low result in connective issues for users. Hence any underestimation of the voltage would have a negative impact on the product performance.

$$P_{\text{out}}^{new} = P_{\text{out}}^{old} * (\frac{1}{2})^2 \tag{2.5}$$

As previously described in this section, P_{out} represents only a subset of the total input power, P_{in}. Therefore, a reduction in P_{out} does not directly correspond to the same proportion of energy savings. The exact impact on P_{in} is not precisely known; however, estimates were made using internal tools for this thesis. According to internal resources at Ericsson, it is reasonable to assume that a 15% reduction in P_{in} can be achieved with a 50% decrease in the voltage for the specific radio studied in this thesis. This voltage shift enables energy savings, which are discussed in further detail in Section 2.1.2. The 15% energy saving is an important assumption throughout this thesis.

2.1.2 Energy Saving

The energy savings enabled by this thesis are achieved through dynamically adjusting the radio's power used for user connectivity (P_{out}) based on demand. Specifically, the voltage is reduced when P_{out} falls below a predefined threshold, which subsequently reduces the input power P_{in}, of which P_{out} is a subset. As previously noted, a 50% decrease in voltage leads to an estimated 15% reduction in P_{in}, representing the energy savings. Energy consumption, measured in kilowatt-hours (kWh), is calculated as the integral under the power curve (measured in watts). The resulting energy savings from shifting the P_{in} curve are thus given by Eq. 2.6. The power input to the radio, P_{in}, is denoted using subscripts to reflect the applied voltage strategy:

- $P_{\text{in,static}}$ refers to the power input when the voltage is fixed at a high level regardless of demand.
- $P_{\text{in,dynamic}}$ is the power input when the voltage is adjusted dynamically according to user utilization, as previously described.

The total energy saved over a time interval $[t_1, t_2]$ is given by:

$$E_{\text{saved}} = \int_{t_1}^{t_2} (P_{\text{in,static}} - P_{\text{in,dynamic}})\, dt \tag{2.6}$$

Concretely, applying a voltage reduction results in a shift in the P_{in} curve, as visualized in Fig. 2.4. Instances where P_{out} is above the threshold remain unaffected, the reduction is only applied during the periods when P_{out} is below the threshold. The energy savings in Fig 2.4 correspond to the light blue shaded area between the original P_{in} curve and the adjusted curve. The light green areas are those where P_{in} could be decreased due to P_{out} being under the set threshold.

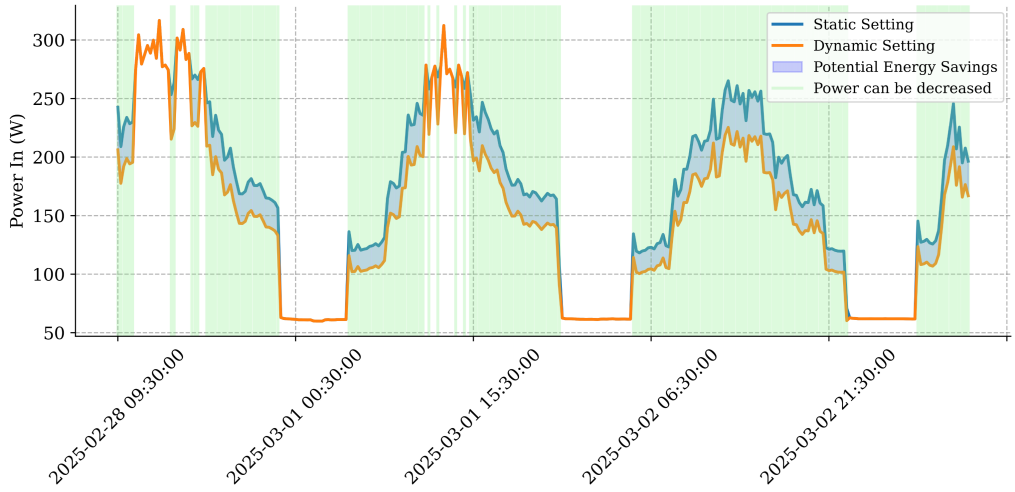


Figure 2.4: Potential power save by halving the voltage when P_{out} is below 40W.

2.2 Machine Learning

ML models can produce either discrete or continuous outputs [16]. When the output is discrete, the task is known as classification; when it is continuous, it is referred to as regression. Binary classification, as the name suggests, involves only two possible output classes. A key concept in ML is supervised learning, where models are trained on labeled data - that is, data for which the desired output is known [16]. Both regression and classification are common types of supervised learning tasks. On the other hand, *unsupervised learning* is ML trained without any labels, a common example being clustering where data is grouped without label knowledge [16].

The process of applying algorithms to data is called *learning* or *training*. The *generalizability* of a model is its ability to predict on new data instances [16]. A well trained ML model should work well over the entire sample size. A challenge generated by many datasets are that of building models on *imbalanced data*. Having a disproportion between classes creates a bias for the more common group [17]. Most studies suggest that an imbalanced dataset has a proportion of 1:4 up to 1:100 though there might be more skewed distributions in real life applications.

Deep learning (DL) is a subfield within ML utilizing several layers of representation [18]. Examples of DL models are *feedforward neural network*, *convolutional neural network*, *long short term memory* and *temporal fusion transformer* which are discussed further in Section 2.4. Fig. 2.5 shows the connection between AI, ML and DL.

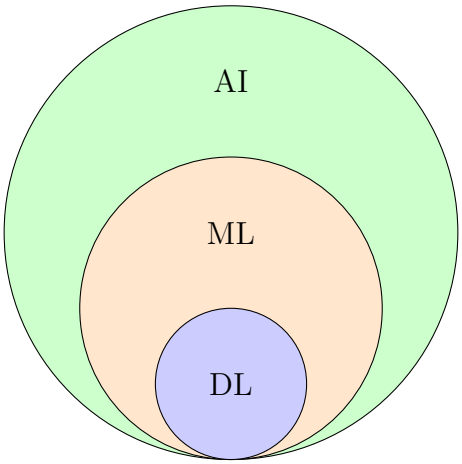


Figure 2.5: Venn diagram of relevant AI concepts.

Hyperparameters are an important concept in machine learning [19]. In ML models, there are two types of parameters: *model parameters*, which are learned and updated during training, and *hyperparameters*, which relate to the model's structure and training configurations. Hyperparameters cannot be learned from the data and must instead be specified prior to training [19]. These parameters can significantly impact model performance and must therefore be chosen carefully. They may relate to model complexity, such as the number of layers in a neural network, or the learning process itself, such as the learning rate.

Tuning hyperparameters is still considered something of a black art [19]. Several tuning strategies exist, each with advantages and limitations. Common methods include Grid Search (GS), which exhaustively searches the hyperparameter space; Random Search (RS), which samples combinations randomly; and Bayesian Optimization (BO), a probabilistic approach that models the performance landscape to guide the search more efficiently.

2.3 Time Series Data

Time series data is a collection of observations recorded in chronological order. The goal of modeling time series data is often to predict future target values that follow certain patterns over time. The following key characteristics are important for time series modeling [20]:

(i) *Univariate/multivariate.* Whether the data consists of a single time series or multiple channels.

(ii) *Stationarity.* Whether the mean and variance remain constant over time.

(iii) *Seasonality.* The presence of short-term patterns that repeat at regular intervals.

(iv) *Outliers and noise.* Irregular data points or fluctuations that deviate from expected patterns and can complicate prediction and analysis.

(v) *Interdependencies between channels.* The degree to which different time series channels are correlated.

Each data instance represents a specific time step t, and can be divided into x_t, a vector of features, and y_t, the corresponding label. x_t may include several lagged instances, making the data multivariate, as illustrated in Fig. 2.6. The figure shows an example of two features and how they vary over time. In the multivariate case two or more such instances are included in the dataset. Multivariate time series data typically offer greater predictive power, though they are more complex and may require larger datasets [20]. Forecasting the next value in a time series typically involves using a window of past observations. For example, to predict the value at time step y_{t+1}, the model uses the input sequence $[x_t, x_{t-1}, \ldots, x_{t-w}]$, where w is the window size, also known as the *lag*.

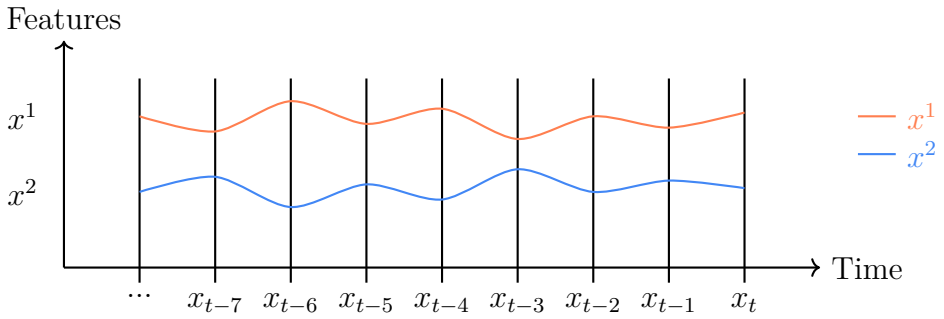


Figure 2.6: Two-dimensional multivariate time series showing variation over time.

2.3.1 Timeseries data for ML

When preparing time-series data for machine learning tasks, an important design choice is how to incorporate past values into the input, a process commonly referred to as defining the *estimation window* [21]. Two standard approaches are the *sliding* (or *rolling*) window and the *expanding* window. In the sliding window approach, the input consists of a fixed number of past observations, referred to as the *look-back length*, which determines how far back the model considers for each prediction. Each input vector in the time-series data is constructed by extending the current observation with past values up to this look-back length - from the single time series in the univariate case, or from all variables in the multivariate case. In contrast, the expanding window grows over time: for a prediction at time t, the input includes all observations up to t; for time t+1, it expands to include the observation at t as well.

The window size in the sliding window approach can be considered a hyperparameter [22]. It is particularly important because it determines how much past information the model receives for each prediction. Ideally, this information should capture both short- and long-term dependencies, as well as recurring patterns such as seasonality [22]. If the window is too small, the model may not receive sufficient context; if too large, it may struggle to identify relevant signals.

After augmenting the input data with past values, the target labels must be created. This requires specifying the *prediction horizon*, which defines how many steps ahead the model should forecast. For time-series data, labels are generated by shifting the values of the target variable forward by the length of the prediction horizon.

2.4 Models

This thesis implements eight different machine learning models for three task formulations and evaluates them according to metrics later described. This section describes the architecture and functionality of each model.

2.4.1 Random Forest

The Random Forest model is an ensemble of *tree predictors* [23]. Each tree has a structure similar to the one shown in Fig. 2.7 [24], and is trained on a sub-sample of the data selected through bagging (bootstrap aggregating) combined with random feature selection [23]. As described in the same article, each tree $tree_t$ has a vector of features Θ_t of length k where each vector is sampled from the same distribution independently from the previous samples.

Each branch is made on a *split selection* that aims to increase the purity within each branch, meaning the representation of one single class [16]. One such measurement is entropy, where lower entropy defines higher purity. Two other measurements used are log loss and gini impurity index [25].

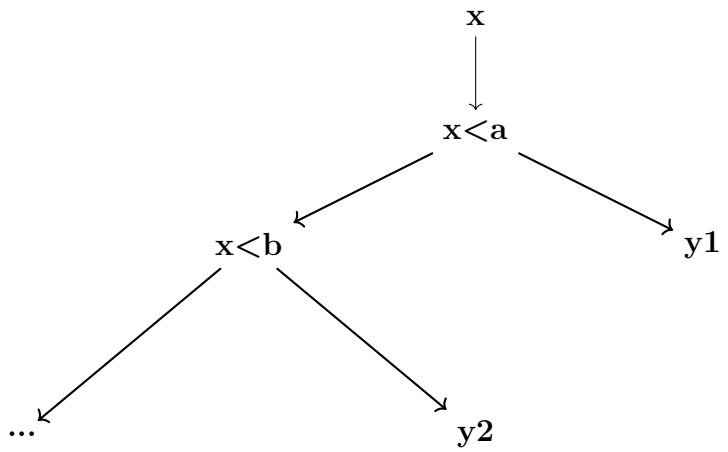


Figure 2.7: Decision tree architecture.

The complete Random Forest consists of a collection of such decision trees, as further illustrated in Fig. 2.8.

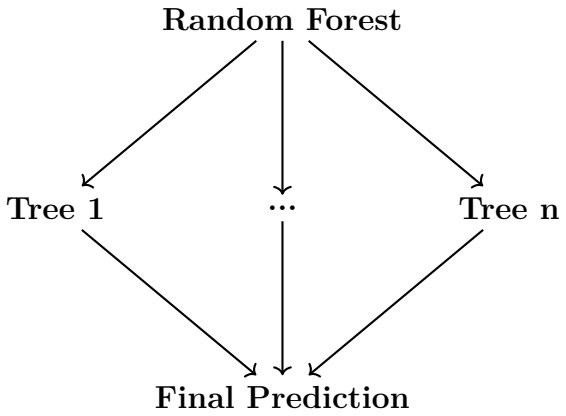


Figure 2.8: Random forest architecture.

A random forest can be described as an ensemble of tree-structured classifiers denoted by $h(x, \Theta_k),\ k = 1, 2, \ldots$ [23]. Each Θ_k represents an independently and iden-

tically distributed (i.i.d.) random vector that governs the construction of tree k. For a given input x, each tree outputs a classification result, and the final prediction is made based on the majority vote among all trees in the forest. This describes how RF predicts classification. On the other hand, the result for regression output is the mean of the trees included in the forest as Eq. 2.7 shows [26].

$$h(x) = \frac{1}{K} \sum_{k=1}^{K} h(x, \theta_k) \tag{2.7}$$

2.4.2 Extreme Gradient Boosting

Extreme Gradient Boosting (XGB) is a widely used tree boosting method [27]. Gradient boosting applied to regression trees yields models that are not only robust and interpretable but also highly effective for both regression and classification problems. This approach is particularly well-suited for handling data that may be noisy or imperfect [28]. The algorithm builds trees sequentially, with each new tree trained to correct the errors of the previous ones, as illustrated in Fig. 2.7. More precisely, each new regression tree is trained on the negative gradient of the loss function [29].

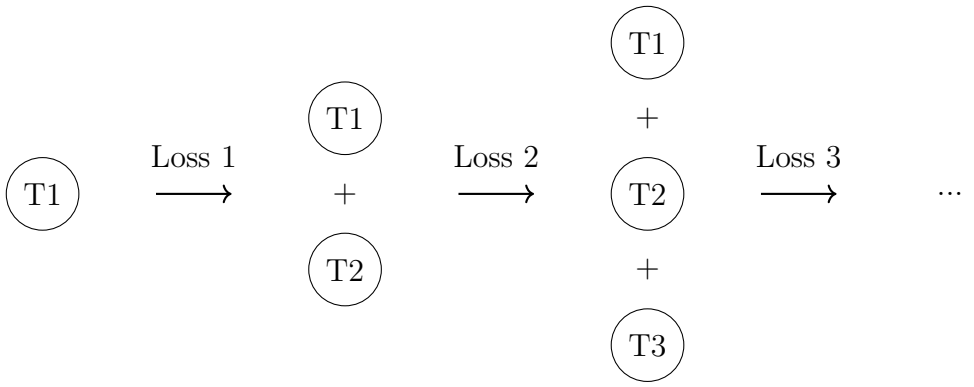


Figure 2.9: XGboost's iterative training process.

2.4.3 K-Nearest Neighbors

The K-Nearest Neighbors classifier (KNC) works by clustering input according to its features. The algorithm is as follows; All training instances get a label, the instance to be predicted is then put into the space and the distances are measured between the new instance and all training instances [30]. The number of k nearest instances are selected, the predicted class is decided as a majority vote. It is common to give closer neighbors increased weight in their vote [30]. The regression version (KNR) is similar, though the prediction label is calculated as a mean of the neighbors' labels [31].

KNC is easily explained as the prediction depends on its neighbors [30]. On the other hand, it can have a poor run-time. Fig. 2.10

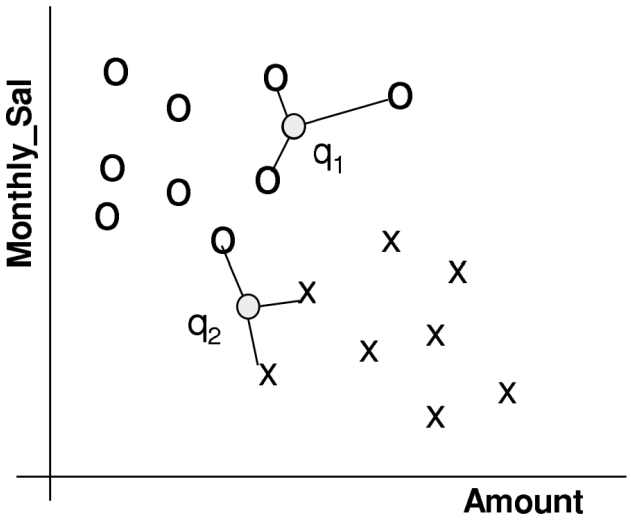


Figure 2.10: Example of KNC algorithm, k=3 [30].

2.4.4 Support Vector Machine

Support Vector networks were initially introduced as a method for classification. The classification version, support vector classification, is refered to as SVC throughout this report. The core idea is to map input vectors non-linearly into a high-dimensional feature space, where a linear decision boundary can be constructed [32]. This linear separator defines an optimal hyperplane that maximizes the margin between different classes. Data points that lie on the margin boundaries are known as *support vectors* [32]. When the data is not linearly separable, slack variables ε are introduced to allow for some misclassifications. These slack variables are penalized in the optimization problem to ensure the classification error is minimized as much as possible.

The support vector machine (SVM) framework was extended to handle regression problems in 1996, resulting in the Support Vector Regression (SVR) algorithm [33]. As described by Basak et al., "The idea of SVR is based on the computation of a linear regression function in a high-dimensional feature space where the input data are mapped via a nonlinear function" [34]. In SVR, a similar principle is applied: instead of finding a hyperplane that separates classes, the goal is to find a function that approximates the target values as accurately as possible within a specified margin of tolerance. The regression model attempts to fit the data points while ignoring deviations smaller than a predefined threshold, thereby controlling model complexity and generalization. Fig. 2.11 illustrates the prediction methodology employed by the SVR algorithm [33].

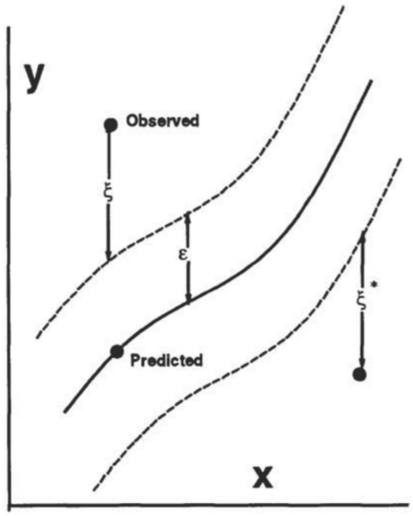


Figure 2.11: Prediction methodology of the SVR algorithm [33].

2.4.5 Feedforward Neural Network

Feedforward Neural Networks (FFNNs) are characterized by a layered architecture consisting of an input layer, one or more hidden layers, and an output layer [35]. Fig. 2.12 illustrates the structure and interaction between these layers. The input layer receives the raw input data, denoted as x in the figure, with three features shown as an example. The hidden layers apply learned transformations to the input, and the output layer generates the final predictions of the model.

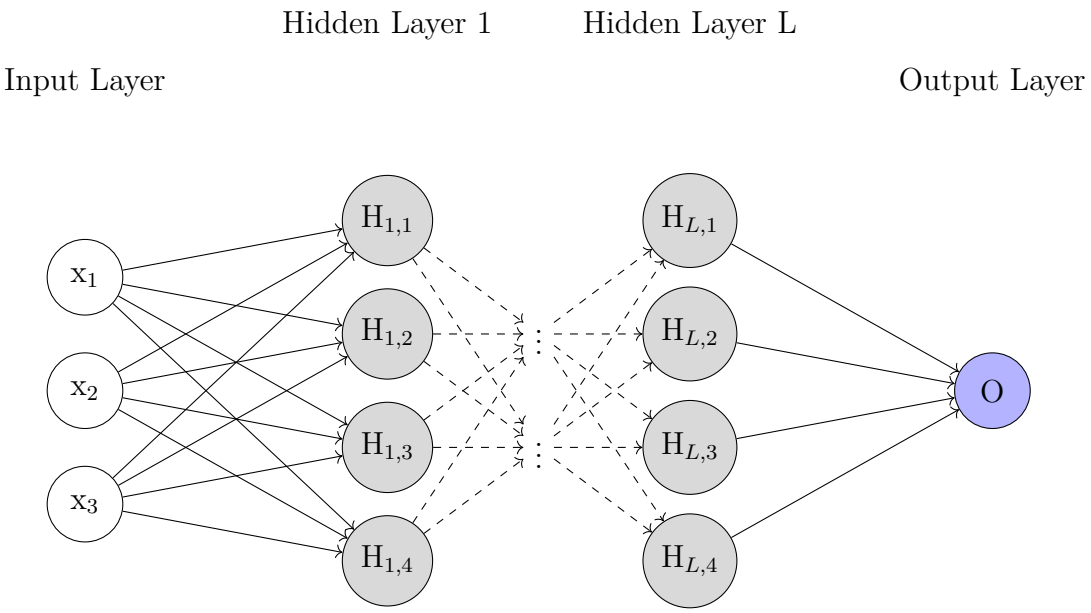


Figure 2.12: Example of a feedforward neural network with four neurons and multiple hidden layers.

FFNNs commonly apply affine transformations followed by specified non-linear ac-

tivation functions [35], as shown in Eq. 2.8 and Eq. 2.9. In Eq. 2.8, $\mathbf{z}^{(1)}$ denotes the pre-activation values of the first layer, computed as a linear combination of the input $\mathbf{x}$ with the learned weight matrix $\mathbf{W}^{(1)}$ and bias vector $\mathbf{b}^{(1)}$. In Eq. 2.9, $\mathbf{h}^{(1)}$ represents the output of the first hidden layer, obtained by applying the non-linear activation function g to $\mathbf{z}^{(1)}$. The activation function introduces non-linearity into the network, enabling it to learn complex patterns [35].

$$\mathbf{z}^{(1)} = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)} \tag{2.8}$$

$$\mathbf{h}^{(1)} = g(\mathbf{z}^{(1)}) \tag{2.9}$$

This process is repeated for each hidden layer, where the output from the previous layer becomes the input to the next. More generally, the equivalent of $\mathbf{x}$ in Eq. 2.8 for layer l is $\mathbf{h}^{(l-1)}$, resulting in:

$$\mathbf{z}^{(l)} = \mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}, \quad \mathbf{h}^{(l)} = g(\mathbf{z}^{(l)}) \tag{2.10}$$

In the final layer, the output is reduced to the desired output size, which is a single scalar value in Fig. 2.12. The final output, denoted by $\hat{y}$, is obtained through a linear transformation, as shown in Eq. 2.11. This output can optionally be passed through an activation function, depending on the task.

$$\hat{y} = \mathbf{w}^{(L+1)}\mathbf{h}^{(L)} + b^{(L+1)} \tag{2.11}$$

Here, L is the index of the last hidden layer.

Various activation functions, $g(\cdot)$, are available, each suitable for different types of problems. The rectified linear unit (ReLU) is a commonly used activation function in deep learning [36]. Table 2.1 presents the definition of ReLU along with several other popular activation functions commonly used to enhance the computational capabilities of DL models [37]. $\Phi(z)$ in the second row of Table 2.1 refers to the cumulative distribution function for a Gaussian distribution. The input, z, to the activation functions is the raw output from a layer before the activation function is applied,

Activation function	**Equation**
ELU	$\mathrm{ELU}(z) = \begin{cases} z & \text{if } z > 0 \\ \alpha\,(e^z - 1) & \text{if } z \leq 0 \end{cases}$
GeLU	$\mathrm{GeLU}(z) = z \cdot \Phi(z)$
Leaky ReLU	$\mathrm{LReLU}(z) = \begin{cases} z & \text{if } z > 0 \\ \alpha z & \text{if } z \leq 0 \end{cases}$
ReLU	$\mathrm{ReLU}(z) = \max(0, z)$
Sigmoid	$\sigma(z) = \frac{1}{1+e^{-z}}$
Softmax	$\mathrm{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1} e^{z_j}}$
Tanh	$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$

Table 2.1: Some activation functions used to improve learning [37], [38].

While activation functions introduce non-linearity and enable neural networks to learn complex patterns, other techniques address additional challenges during training. One such challenge is overfitting to the training data. *Dropout* is a regularization technique that mitigates overfitting in deep neural networks by randomly deactivating units during training [39]. This prevents the model from becoming overly reliant on specific neurons, encouraging them to learn features independently and reducing co-adaptation. At test time, the full network is used, with outputs scaled to approximate the behavior of the many smaller sub-networks seen during training. This improves generalization to unseen data.

Another challenge for neural networks is the shifting distribution of layer inputs as the network trains, known as internal covariate shift [40]. *Batch Normalization* was introduced to mitigate this problem by normalizing the inputs to each layer within each mini-batch. The method improves training stability and speed, allows for higher learning rates, and reduces sensitivity to weight initialization. It can also reduce the need for regularization techniques such as dropout [40].

With deep learning, one challenge is that information can degrade as it propagates through many layers [41]. *Residual connections* address the challenge of training very deep neural networks by reformulating the learning objective. Instead of learning a direct mapping, each layer learns a residual function relative to its input, representing the difference between the desired output and the input. This structure improves gradient flow, making optimization easier and enabling the training of significantly deeper models. In practice, a portion of the input bypasses intermediate layers and is directly added to the output. This skip connection helps preserve important features from earlier layers and mitigates information loss.

2.4.6 Recurrent Neural Network

Recurrent neural networks extend feedforward architectures by introducing recurrent connections, allowing them to capture dependencies across time steps in sequential

data [42]. These recurrent connections link a layer back to itself, enabling information to flow from previous hidden states to the current state. This information flow is illustrated in Fig. 2.13, where the dashed edges represent connections between a node and its states at adjacent time steps.

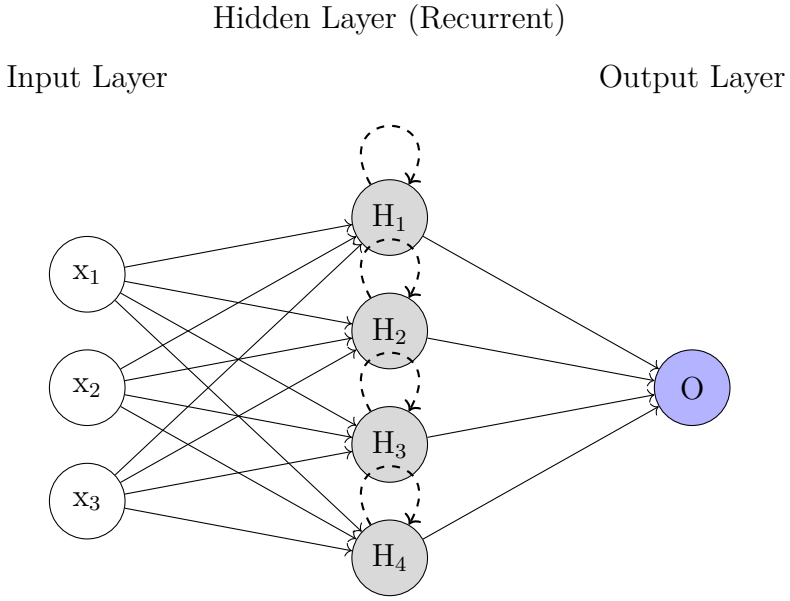


Figure 2.13: Example of recurrent neural network with a single hidden layer and dashed recurrent connections

The forward pass can be computed as shown in Eq. 2.12, where $\mathbf{h}_t$ is the hidden state at time t, $\mathbf{W}_x$ is the input weight matrix, $\mathbf{x}_t$ is the input vector, and $\mathbf{W}_h$ and $\mathbf{b}_h$ are the recurrent weight matrix and bias vector, respectively. The function σ denotes the sigmoid activation function, as defined in Table 2.1.

In Fig. 2.13, the input vector $\mathbf{x}_t$ consists of three features. The input weight matrix $\mathbf{W}_x$ contains the weights associated with the connections between the input features x_1, x_2, x_3 and the four neurons H_1, H_2, H_3, and H_4. The recurrent weight matrix $\mathbf{W}_h$ and the recurrent bias vector $\mathbf{b}_h$ correspond to the recurrent connections between each neuron and its previous state, as illustrated in Fig. 2.13.

$$\mathbf{h}_t = \sigma(\mathbf{W}_x\mathbf{x}_t + \mathbf{W}_h\mathbf{h}_{t-1} + \mathbf{b}_h) \tag{2.12}$$

This forward pass differs slightly from the one described in Eq. 2.9 in the previous section.

2.4.7 Long Short Term Memory

Long short term memory (LSTM) is a special case of its predecessor RNN [43]. The structure was created to address the exploding/vanishing gradients problem [44]. Vanishing gradients occur when gradients become too small during backpropagation, making it difficult for the network to learn. This issue is common in time

series models, which often rely on deep architectures. LSTM solved this problem by introducing a memory cell that stores information over time. The memory cell consists of an input gate and output gate. The LSTM also contains a *forget gate* which provides the ability to reset its state. The forget gates decides what information should be kept or removed from the memory cell [45]. The *input gate* controls what information should be used to update recent block states. The *output gates* decides the prediction based on the current block state. The architecture of the LSTM can be seen in Fig. 2.14.

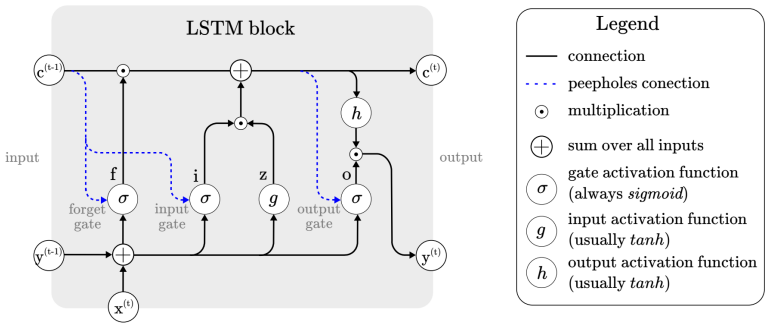


Figure 2.14: LSTM Architecture [44]

2.4.8 Convolutional Neural Network

One of the first highly successful Convolutional Neural Networks (CNNs) was introduced in 1995 [46]. CNNs are commonly used in image processing and recognition tasks due to their ability to extract spatial hierarchies of features. Their architecture is particularly adept at identifying contextual information, which helps overcome the limitations of pixel-wise comparisons in image analysis [47]. Another notable advantage of CNNs, especially in the context of radio efficiency, is that their architecture can be efficiently implemented in hardware [46].

CNN architectures are typically constructed by stacking three main types of layers: convolutional, pooling, and fully connected layers [48]. The convolutional layer performs element-wise multiplications between a learnable weight matrix, known as a *kernel*, and local regions of the input. Pooling layers reduce the spatial dimensions of the data by downsampling, helping to control overfitting and reduce computation. Finally, fully connected layers map the learned features to the output space [48].

The convolution operation in the 2D case is illustrated in Fig. 2.15, where a kernel slides across the input matrix and performs weighted summations.

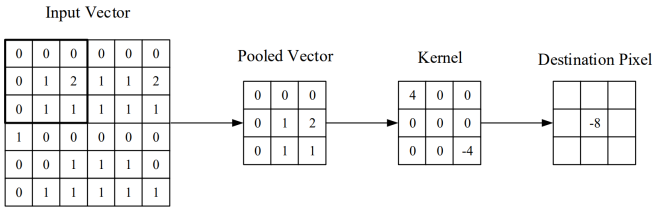


Figure 2.15: Kernel functionality in a 2D convolutional layer [48].

Convolutional layers are controlled by several hyperparameters that affect the dimensions of the input and output. As with any neural network, it is important to understand how each layer transforms the data. This becomes particularly crucial for CNNs, where valid values for certain hyperparameters often depend on others. Further details are provided in Section 3.6.2.

2.4.9 Temporal Fusion Transformer

Temporal fusion transformer (TFT) is a specialized version of transformers created for time series data. A key characteristic of the TFT model is that it outputs quantiles instead of just one prediction, this can be helpful depending on the application, as it indicates how confident the model is of its output [49]. The output for each quantile is calculated in Eq. 2.13 where τ represents the forecasting horizon, k the look-back period, $y_{i,t-k:t}$ previous target values, $z_{i,t-k:t}$ unknown inputs and $x_{i,t-k:t+\tau}$ known inputs. The standard loss function that is minimized during training is the joint quantile loss over all quantiles [49].

$$\hat{y}(q,t,\tau) = f_q(\tau, y_{i,t-k:t}, z_{i,t-k:t}, x_{i,t-k:t+\tau}, s_i) \tag{2.13}$$

The TFT can differentiate between different types of data [49]. Fig. 2.16 demonstrates how the input data is divided into three different categories; static metadata, time-varying past inputs and time-varying known future inputs. Data from these categories are handled in different ways, which gives the TFT advantages in leveraging information in the input data. For each type of data, a feature selection is made to filter out the most prominent variables.

The main components of the TFT are LSTM, previous explained in Section 2.4.7, and Gated Residual Networks (GRN) which is a neural network specifically created for the TFT [49]. The GRN controls the information flow in the TFT. They contain gates and skip connections, which makes it possible for the TFT to only apply nonlinear processing when needed. This can reduce complexity for smaller datasets.

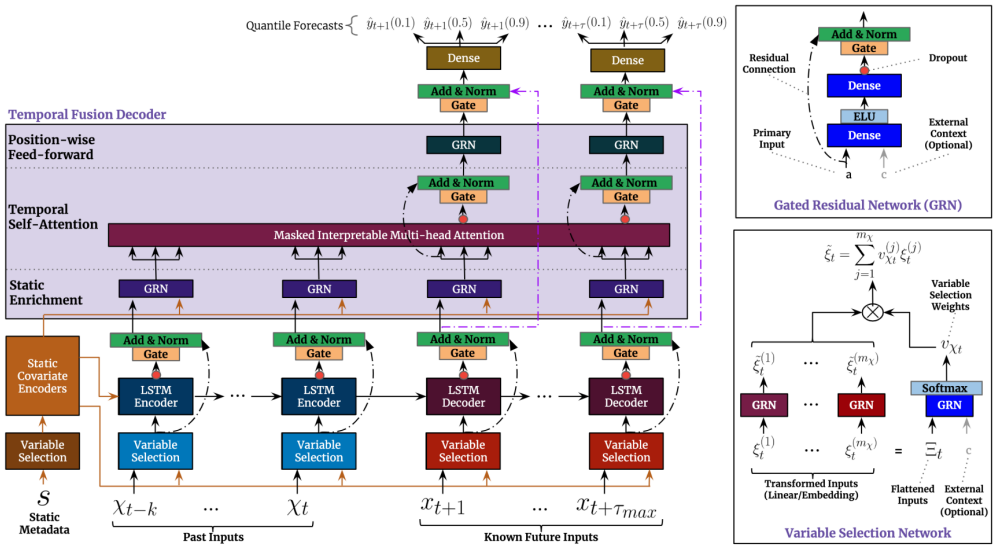


Figure 2.16: Temporal Fusion Transformer architecture [49].

2.5 Optuna

Optuna is an open-source framework for automatic hyperparameter optimization in Python. The framework enables utilization of Bayesian methods for guiding the search for optimal hyperparameters [50]. This means that optimal solutions can be found quicker than when using random search or grid search.

Optuna offers several different samplers and pruners for the user to chose between [51]. The default sampler is the Tree-structured Parzen Estimator algorithm (TPESampler). The TPESampler fits Gausian Mixture models to the parameters.

2.6 SHAP

SHAP (SHapley Additive exPlanations) is a unified framework for interpreting model predictions [13]. It was developed to improve interpretability in complex models by addressing the trade-off between predictive accuracy and transparency. SHAP uses a model-agnostic approach that applies to a wide range of algorithms, helping to standardize explainability methods.

SHAP assigns an importance value, called a *SHAP value*, to each feature for a specific prediction [13]. These values, grounded in game theory [52], represent each features contribution to the model output. The explanation of a prediction is treated as a separate, interpretable approximation of the original modelreferred to as the *explanation model* [13].

SHAP values quantify how much a feature changes the prediction when its value is altered, and they support both local and global interpretability [13].

3

Methodology

This chapter begins by providing a high-level overview of the solution in a step-by-step list in Section 3.1. Section 3.2 provides information about the data creation in this thesis. Section 3.3 presents background information on the dataset and highlights key patterns within it. Section 3.4 continues by describing the data preprocessing made to prepare it for the ML algorithms. Section 3.5 outlines the setup of the prediction task, Section 3.6 explains the rationale behind the choice of models, and Section 3.7 details the hyperparameter optimization process. Section 3.8 outlines the modifications made to the test data to simulate high utilization for evaluation purposes. Section 3.9 outlines the model evaluation strategy, Section 3.10 explains how the predictions of certain models are made interpretable, and finally, Section 3.11 introduces the Python libraries used as tools throughout this thesis.

3.1 Solution Architecture

The objective of this thesis is to design a system that dynamically adjusts the voltage setting in radios based on predicted traffic demands expressed through the target variable PRB-U. As described in Section 2.1, PRB-U refers to the proportion of physical resource blocks utilized by the radio's cell for signal transmission. A high PRB-U indicates a heavily loaded radio.

As previously described in Section 2.1.2, it is not possible to directly tune the power of the radio. Instead, the voltage is adjusted to achieve the desired power level. The dynamic voltage adjustment process is structured into the following steps for each predictive model:

1. **Predict PRB-U** based on time series data:
   (a) **Classification**: Directly predict whether the target variable corresponds to the higher (1) or lower (0) class.
   (b) **Regression**:
      i. Predict the exact utilization.
      ii. Map the prediction to the higher or lower class as Eq. 3.1.

$$\hat{y} = \begin{cases} 0, & \text{if } \hat{y}_{continious} < 0.25 \\ 1, & \text{otherwise} \end{cases} \tag{3.1}$$

2. **Adjust the voltage level** according to which level was predicted as:
   (a) **High level**: 0% change
   (b) **Low level**: 50% decrease
3. **Deduce energy save from** P_{in} depending on Voltage level as:
   (a) **High level**: 0% change
   (b) **Low level**: 15% decrease

3.2 Data Creation from an ML Perspective

This section describes how the dataset was constructed from raw sensor data collected from five randomly selected radios, all located within the same country.

3.2.1 Data Collection and Sources

The primary dataset was collected from two internal sources and spans from 2025-01-17 to 2025-03-06, resulting in a total of $23,520$ raw data points. An independent test set was created using data from 2025-03-08 to 2025-03-22, resulting in $7,200$ raw instances. The dataset includes both time-series and static features, where the time-series data consists of 15-minute interval measurements. One of the key time-series variables is PRB-U, the target variable described in Section 2.1.

3.2.2 Creating Lagged Features

Each input vector was enriched with lagged values from the time-series features using a sliding window approach, as outlined in Section 2.3.1. The window size, referred to as the *look-back length*, was treated as a tunable hyperparameter. To maintain computational feasibility and ensure coverage of one full day, the maximum window length was capped at 96 (equivalent to 24 hours).

When generating lagged features, initial data points without sufficient historical context were discarded. Since the optimal window size varied between models, the first 96 data points were consistently removed from the training, validation, and test sets to ensure all datasets had equal lengths across model configurations.

3.2.3 Forecasting Horizon and Target Construction

The forecasting horizon was set to one time step, corresponding to 15 minutes ahead. This choice was based on the practical consideration that voltage levels can be changed within roughly one second, and a short prediction horizon simplifies the learning task. Consequently, the target variable, PRB-U, was shifted forward by one time step to align with this forecast horizon.

3.2.4 Data Splitting

Following preprocessing and feature construction, which will be explained in Section 3.4, the data was partitioned into training, validation, and test sets. The training and validation sets were extracted from a single continuous period, with 80% of the data used for training (18 428 instances after removing the first 96 due to lag construction) and the remaining 20% for validation (4 125 instances, also after removing the first 96). The processed test set contains 6 710 instances after the same adjustment.

The training, validation, and test datasets contain combined data from all five radios, meaning the models were trained and evaluated on aggregated data without explicit information about which radio each instance originated from. On a per-radio basis, this corresponds to approximately $\frac{18{,}428}{96\times5} \approx 38.4$ days of data per radio in the training set, $\frac{4{,}125}{96\times5} \approx 8.6$ days in the validation set, and $\frac{6{,}710}{96\times5} \approx 14$ days in the test set.

3.2.5 Handling Missing Data

The test dataset contained a small number of missing time steps. These were imputed using the most recent available value from the same radio. Table 3.1 lists the missing timestamps by radio number. Additional feature engineering steps are described in Section 3.4.

Radio	Missing timesteps
1	2025-03-12 08:30, 2025-03-21 12:30, 2025-03-22 00:30, 2025-03-22 01:45
3	2025-03-12 08:45
4	2025-03-12 08:00, 2025-03-12 08:30
5	2025-03-12 08:00, 2025-03-12 08:30, 2025-03-12 09:00

Table 3.1: Missing timesteps in test data per radio.

The training, validation and test data conatains combined data from all five radios. Meaning, the models were trained on data from all radios without an identification of which radio was whitch.

3.3 Data Inspection

In this section, the data is examined and visualized to determine its characteristics and patterns. All data visualized in the following sections is data used for training and validation while the test data remains untouched.

3.3.1 Imbalance in Target

First, an inspection of the target variable *PRB-U*'s distribution was performed. As shown in Table 3.2, the dataset exhibits class imbalance, with the negative class consistently underrepresented across all splits. A negative class instance is one in

the lower voltage category, and a positive instance is one in the higher. This overall low utilization is further confirmed by the histograms in Fig. 3.1, which display the target distribution for radios 1 and 2. The left plots show the distribution including sleep mode, where utilization is actively set to zero during nighttime hours, while the right plots exclude data from sleep mode hours. Both indicate generally low utilization and consequently a significant potential for energy savings beyond the current night mode setting. The plots assume sleep mode is active from 00:00 to 05:00, during which utilization is zero. Histograms for the other radios are provided in Appendix A.

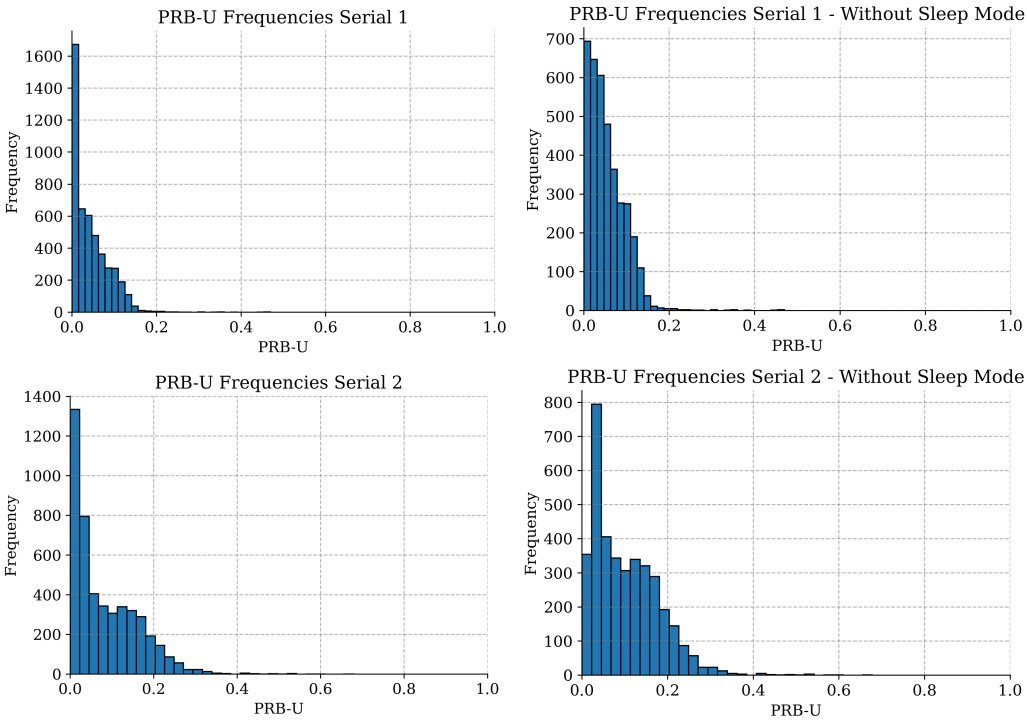


Figure 3.1: Histograms of the target variable *PRB-U* with and without sleep mode for radios 1 and 2.

Data Set	Positive	Negative	Negative to Positive
Training	158	18 272	116:1
Validation	66	4 059	62:1
Test	68	6 642	98:1

Table 3.2: Class distribution in each data set.

3.3.2 Seasonality

All correlation measurements in this section use the Pearson correlation and is calculated as shown in Eq. 3.2, where $PRB_{t,i}$ denotes the *PRB-U* measured t time steps prior to the current value. A correlation above 0.7 is often considered strong,

between 0.6 and 0.4 is considered to be moderate by many fields while a correlation below 0.3 is weaker [53].

$$Corr(PRB, PRB_t) = \frac{\sum_{i=1}^{n}(PRB_i - P\bar{R}B)(PRB_{t,i} - P\bar{R}B_{t,i})}{\sqrt{\sum_{i=1}^{n}(PRB_i - P\bar{R}B_i)^2}\sqrt{\sum_{i=1}^{n}(PRB_{t,i} - P\bar{R}B_{t,i})^2}} \quad (3.2)$$

As outlined in Section 1, this thesis utilizes time series data as the foundation for building its machine learning models. Fig. 3.2 show the correlation between the target variable, PRB-U in the next time step, and the previous values. This heatmap illustrates the relevance of past utilization values in predicting the target variable. It highlights a reduced importance for intermediate time lags and an increased relevance as the lag approaches 24 hours, suggesting a seasonal pattern with a 24-hour cycle. An almost identical correlation pattern can be seen for the other lag variable mean power consumption. The exact plot can be seen in Appendix A.

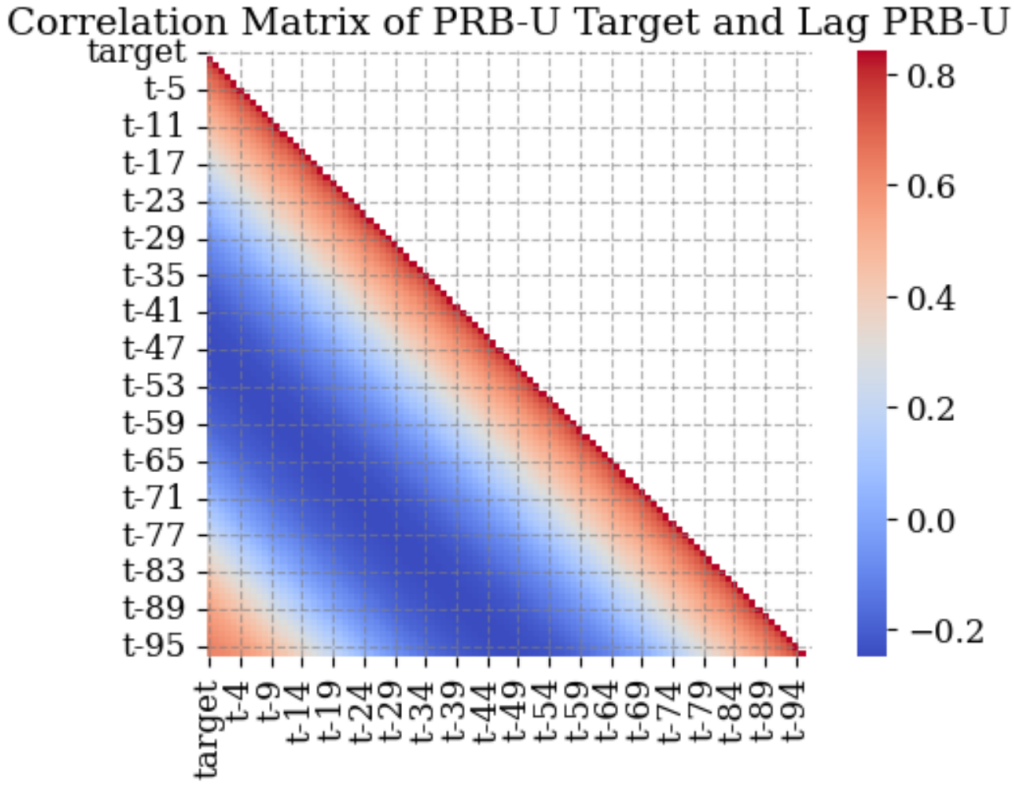


Figure 3.2: Correlation between the target variable and the investigated lags.

Fig. 3.3 provides an overview of the weekly and daily patterns of *PRB-U* across the radios with a mean of the target variable. All units show zero utilization from 11 p.m. until approximately 5 a.m., indicating that the radios are configured with *sleep mode* enabled. Another notable observation is the overall utilization level. The maximum average utilization across the five radios is approximately 12%, suggesting generally low utilization throughout the analyzed period.

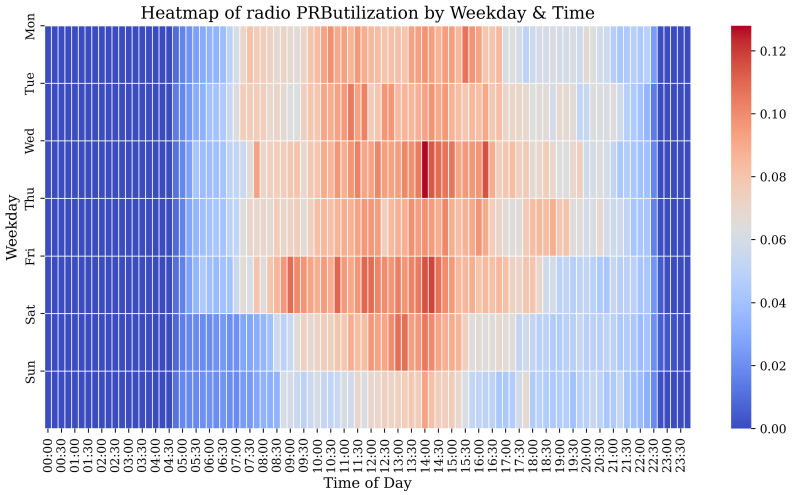


Figure 3.3: Heatmap of average PRB-U over weekday and time of day.

Furthermore, the target variables of the radios are correlated, as shown in Fig. 3.4. Radios 2 and 3, 2 and 4, as well as 3 and 4, exhibit stronger correlations, suggesting that the target patterns of one can provide informative signals for the others. In contrast, radio 5 shows lower similarity to the others, as indicated by the bottom row of Fig. 3.4, which reflects its weaker correlations with the rest.

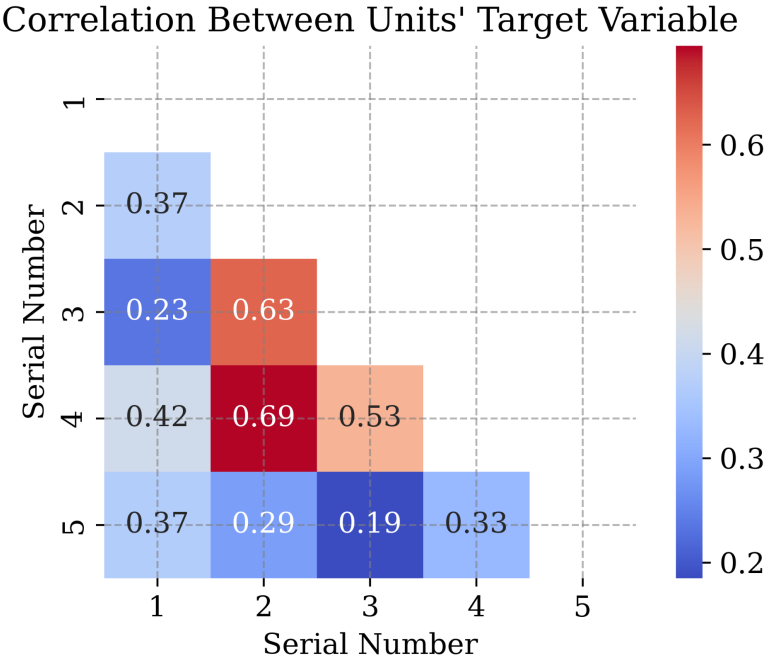


Figure 3.4: Heatmap of the correlation between radios' PRB-U.

Inspecting the heatmaps of mean PRB-U per weekday and time on a one radio level gives a more detailed view of the utilization patterns, as Fig. 3.3 reveals that each radio exhibits distinct utilization patterns. Radios 2 and 4, which show relatively high correlation in Fig. 3.4, also display similar patterns in their respective heatmaps in

Fig. 3.5. Notably, both radios maintain a consistent 0% utilization during nighttime hours, indicating that their sleep mode settings are activated at the same times.

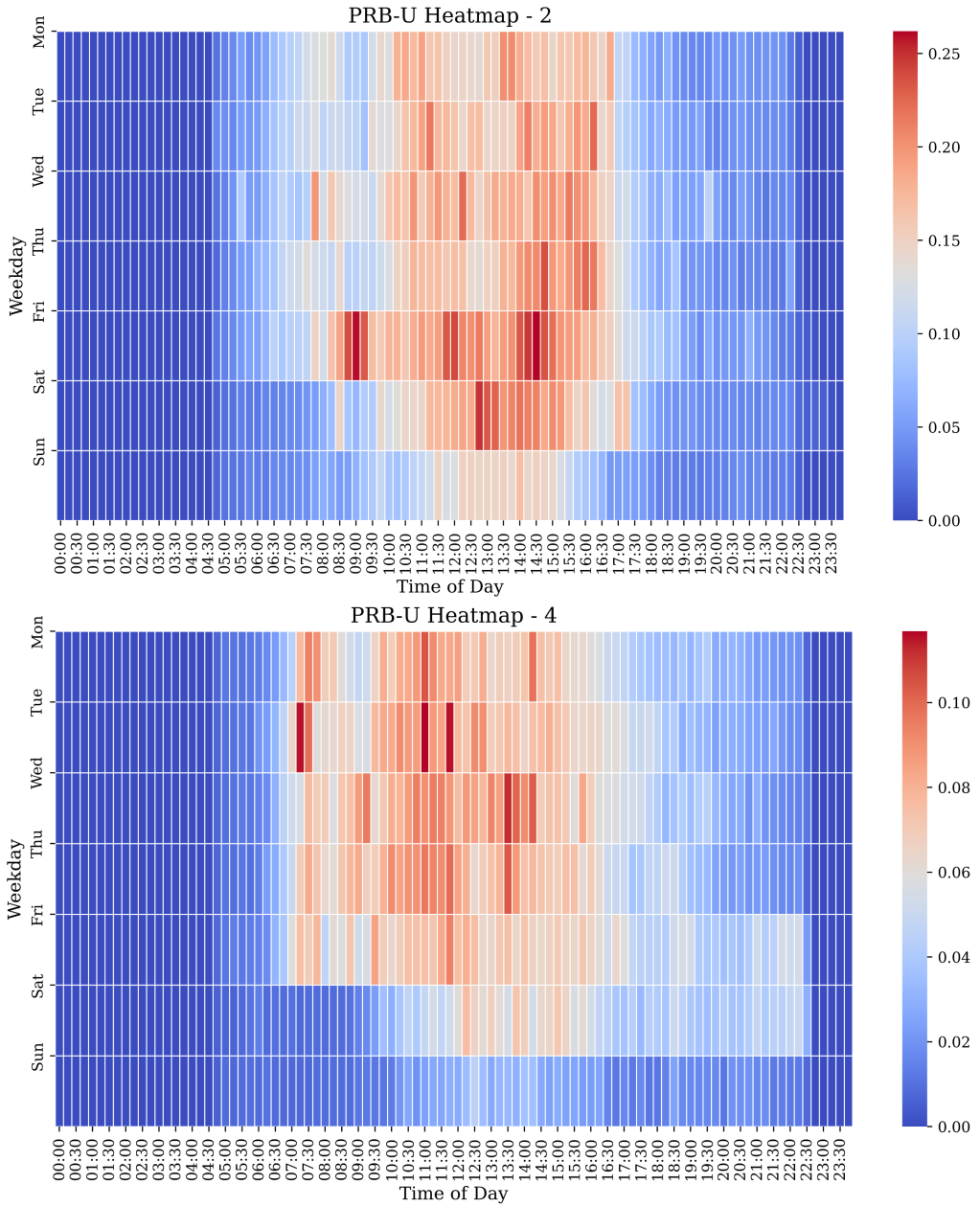


Figure 3.5: The target variable over weekday and time of day for radio 2 and 4.

Furthermore, comparing the heatmap to the PRB-U curve of each radio strengthens the same understanding of the pattern. The heatmap previously shown in Fig. 3.5 is reflected in the curve of PRB-U for the two units in Fig. 3.6. Radio 2 is mostly active during mid-day as is radio 4 though it has lower utilization on weekends. This lower utilization during weekends is clear in day 3 (2025-01-19) and the last day (2025-01-26) of Fig. 3.6 which are both Sundays and had the lowest overall utilization in Fig. 3.5.

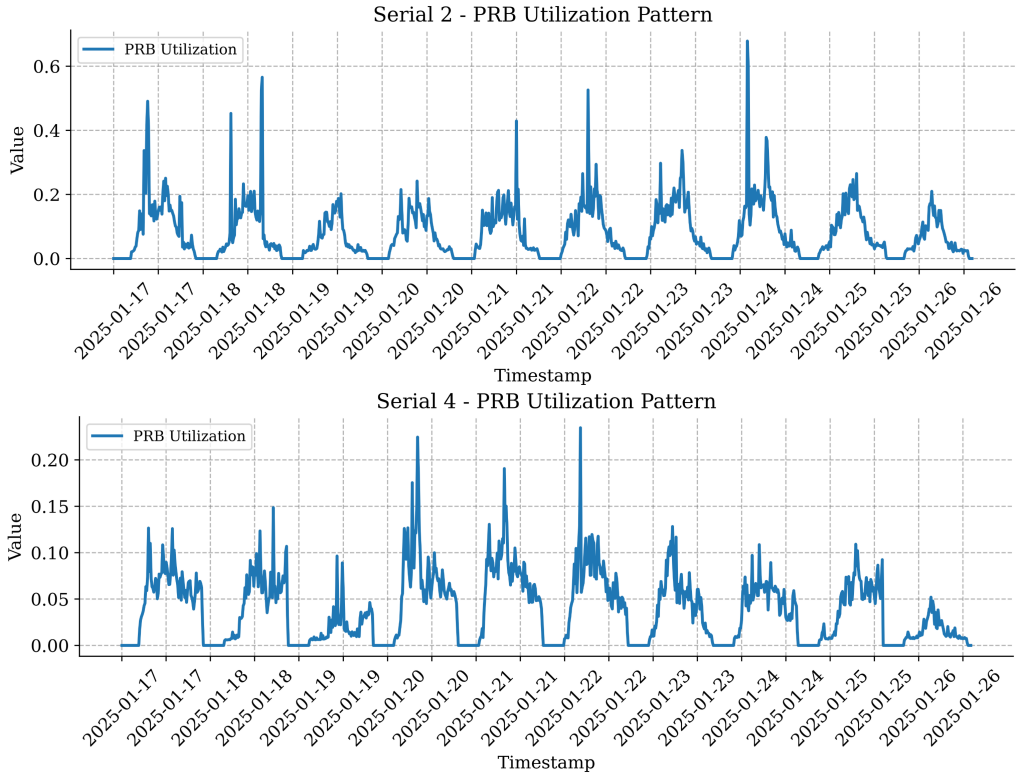


Figure 3.6: PRB-U patterns for radio 2 and 4.

The heatmap in Fig. 3.7 gives an overview of the pattern of radio 5, showing distinct differences from the previous heatmaps of radio 2 and 4 in Fig. 3.5. There is less utilization mid-day and more towards the evenings.

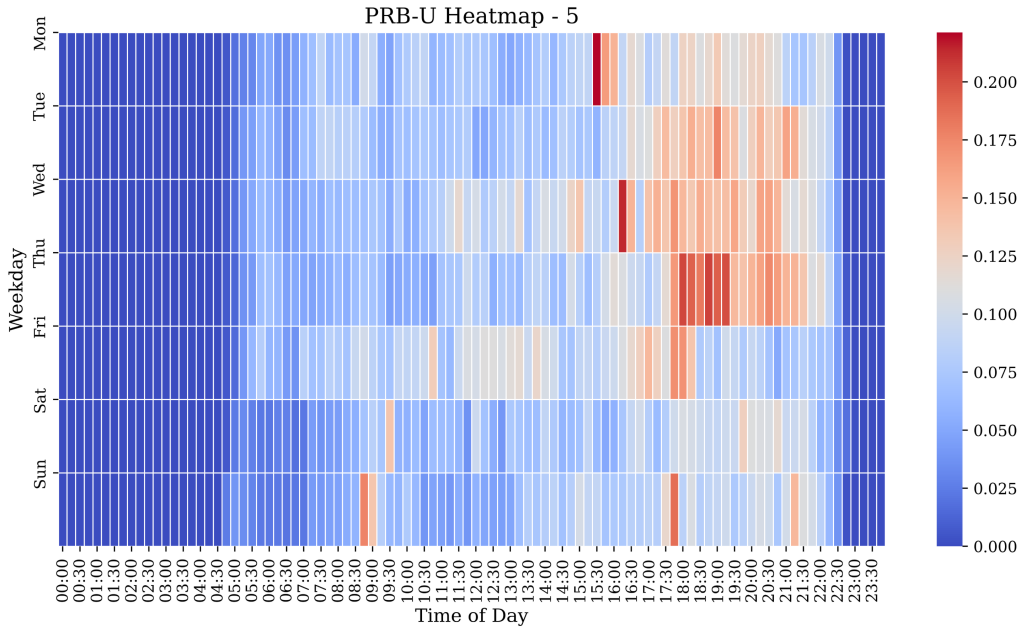


Figure 3.7: The target variable over weekday and time of day for radio 5.

The previous heatmap, Fig. 3.7, indicated that radio 5 has higher utilization during Wednesdays and Thursdays, which is reflected in the curve of PRB-U for radio 5 in Fig. 3.8. The two most significant peaks occur on a Wednesday and Thursday, one on 2025-01-22 which is a Wednesday and another on 2025-01-23 which is a Thursday.

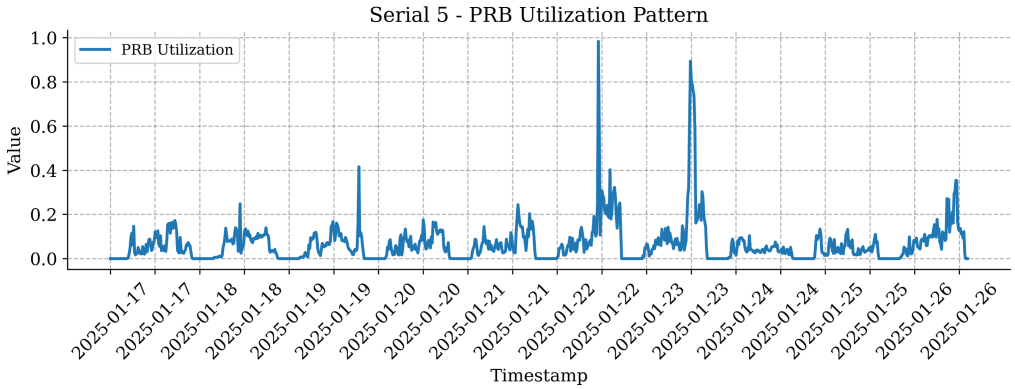


Figure 3.8: PRB-U pattern for radio 5.

3.4 Feature Engineering

The data is mainly collected through sensors and is kept on servers at Ericsson. Data was pulled from two internal sources which resulted in the following list of available features; *serialNumber*: which radio the data belongs to, *timestamp*: the date and time of the measurement, *PowerConsumption* an array power consumed every 6 seconds where every array contained 150 values, *MinPowerConsumption* was the lowest value in the PowerConsumption array while *MaxPowerConsumption* is its maximum value. The feature *PRB-U* is the target variable and is measured as the average utilization during the 15 minute interval, *voltage* is an array of voltages measured every 6 seconds and *current* is an array of current measurements every 6 seconds. The *PowerConsumption* array contained some missing entries which were addressed by forward-filling, assuming the last observed measurement remained valid until a new value was recorded.

The feature *Current* was excluded from the analysis to avoid multicollinearity, as it was found to be highly correlated with *Power*. The relationship between current and power is described is $P = U * I$, where U denotes the voltage and I the current. Since all radios operate with static voltage, any variation in power (P) directly corresponds to changes in current (I). Additionally, *Voltage* was removed from the dataset, as it is theoretically static and, in practice, exhibits very low variance both within and across the analyzed radios.

There were mainly two preprocessing actions taken. First, the mean and median values are calculated from the six-second interval sensor data for power measurements. Second, three features were extracted from the timestamp and were processed to contain their cyclic nature. A challenge in ML is that the raw data does not always accurately reflect its state. Time data serves as a typical example, where, for instance, hour 23:00 and hour 01:00 are numerically distant but conceptually

close. Without appropriate preprocessing, such cyclic relationships remain unclear to many ML models. One strategy to address this issue is to transform the data using sine and cosine functions, as shown in Eq. 3.3. This transformation captures the underlying cyclic behavior of the variable. The divisor in the sine and cosine expressions represents the maximum value of the variable. For example, 24 for hours and 7 for weekdays. x in Eq. 3.3 denotes the variable.

$$x_{cos} = \cos\left(2\pi \frac{x}{max(x)}\right), \; x_{sin} = \sin\left(2\pi \frac{x}{max(x)}\right) \tag{3.3}$$

Fig. 3.9 illustrates the correlations between the features and the target variable, *PRB-U* at the subsequent time step. The bottom row of the figure shows the absolute correlation values between each feature and all other variables in the dataset.

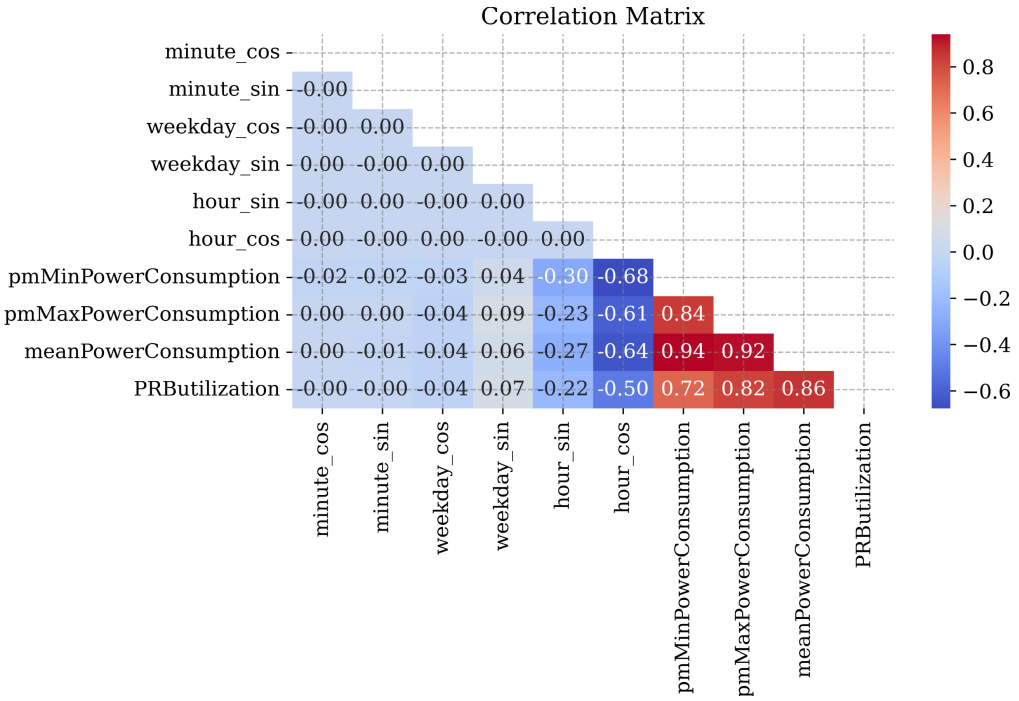


Figure 3.9: Correlation matrix of the features and the target variable.

Table 3.3 shows the full list of included features. The features *maxPowerConsumption*, *minPowerConsumption*, and *medianPowerConsumption*, along with the sinusoidal and cosine-transformed versions of *weekday*, *hour*, and *minute*, are treated as *non-temporal data*, meaning no lagged versions of these variables are included. In contrast, *meanPowerConsumption* is incorporated with lagged values, where the specific lag is determined through hyperparameter optimization.

All features in Table 3.3 were kept in the dataset for all models, despite the correlation matrix in Fig. 3.9 suggesting that some had weak individual correlation with the target variable. Specifically, the *minute* and *weekday* features were considered for removal. However, such features may still contribute through interactions with

other variables or capture temporal patterns not reflected in simple pairwise correlations. For example, the *minute* feature could represent behavioral patterns such as periodic bus traffic near a radio tower. Moreover, these features may become more important in future applications involving more fine-grained prediction tasks. Given the limited total number of features and their potential value, no feature was excluded.

Feature	**Description**
serial	annonimous categorization from 1-5 of included radios
PRB Utilization	Utilization of broadcast abilities every 15 minutes
MaxPowerConsumption	max power consumed during 15 minute interval
MinPowerConsumption	min power consumed during 15 minute interval
meanPowerConsumption	mean power consumed during 15 minute interval
medianPowerConsumption	median power consumed during 15 minute interval
minute_sin	sinusodial transformation of minute
minute_cos	cosinusodial transformation of minute
hour_sin	sinusodial transformation of hour
hour_cos	cosinusodial transformation of hour
weekday_sin	sinusodial transformation of weekday
weekday_cos	cosinusodial transformation of weekday

Table 3.3: Features after data preprocessing.

3.5 Prediction Task Formulation

Following the general objective of the thesis and the examination of data characteristics, this section presents the formulation of the prediction task. In this thesis, the specific task is to predict the voltage level at the next time step, which can be framed as either a classification or a regression problem, as discussed in Section 3.1.

In a regression setting, the model aims to predict a continuous PRB-U value which is then mapped to a voltage level. Performance can be evaluated using metrics such as the mean squared error (MSE), which measures how close the predicted values are to the actual ones. This formulation allows for evaluating how well the model captures the underlying dynamics of the data. Alternatively, the task can be formulated as a binary classification problem, where the objective is to predict whether the voltage level will be high or low. In the context of this study, since the voltage only takes on two discrete levels, the exact numerical value is less important. Thus, it is reasonable to argue that, in this case, the primary concern is the correct classification of the voltage level, rather than the precise numeric deviation.

On the other hand, using a regression formulation provides continuous-valued feedback, which may allow the model to capture subtle variations and trends in the data that are not reflected in a binary classification. In addition, regression outputs can be easier to interpret visually, as predicted values can be directly compared to true values in a plot to observe deviations over time.

Given the respective advantages of both approaches, the prediction task was formulated in two ways: as a regression task and as a classification task. This dual formulation enables a direct comparison of model performance under different learning objectives and evaluation metrics.

To compare model behavior under these learning objectives, the regression task was evaluated using both MSE and F1 score, while the classification task was evaluated using F1 score alone. These two scores are explained in detail in Section 3.9, and the rationale and implementation of this dual-objective tuning strategy are discussed further in Section 3.7.

3.6 Included Models and Their Optimization

This section presents the models selected to address the prediction task introduced in the previous section. A diverse set of models was considered to enable a comparative evaluation of different methodological approaches with distinct strengths and assumptions. The selected models are described in the following subsection, followed by a detailed explanation of the custom-designed hybrid models.

3.6.1 Model Selection

Models were selected to reflect a range of fundamentally different algorithmic strategies. Since the target deployment environment, being radio stations, has limited computational resources, particular emphasis was placed on including a set of less complex, statistical models. If these models performed comparably to more complex alternatives, they would be preferred due to their lower inference cost. For this purpose, Random Forest (RF), Support Vector Machine (SVM), Extreme Gradient Boosting (XGB), and K-Nearest Neighbors (KNN) were selected. These models were included to evaluate whether simpler, well-established techniques could offer competitive predictive performance while being more suitable for deployment. They were also selected because they have, in many cases, demonstrated performance comparable to, or even better than, more complex neural network architectures on structured prediction tasks. For example, tree-based models such as XGB and RF have been shown to outperform deep learning models on a variety of medium-sized tabular datasets [54].

Several neural network architectures were included in addition to the statistical models. They were included since neural networks have shown competitiveness in capturing complex and long-term patterns in data [9]. These consisted of a Feedforward Neural Network (FFNN), a CNN-based model later introduced as HybridCNN, a LSTM-based model later referred to as HybridLSTM, and the Temporal Fusion

Transformer (TFT). The TFT is specifically designed for time series forecasting and has shown competitive performance for such tasks [49]. The hybrid models were specifically designed to process sequential and static features through separate streams, allowing each input type to be modeled more effectively. These architectures are described in greater detail in the following section.

3.6.2 Model Specifications

The statistical models were implemented using their conventional formulations without any custom architectural modifications. In contrast, the neural network architectures were adapted to better suit the specific task.

This subsection presents each neural network architecture in detail, along with an explanation of which structural elements were included as hyperparameters. Certain layers are conditionally activated based on the hyperparameter configuration. One such example is the hyperparameter indicating the number of layers introduced in the HybridLSTM and FFNN. More specific components are illustrated in the figures using dashed outlines, rather than solid lines. The architectural descriptions begin with the FFNN model, followed by the HybridLSTM and HybridCNN.

The FFNN architecture is illustrated in Fig. 3.10. The model processes the input sequentially through three modules: the input module, one or more hidden modules, and the output module. The input module consists of a linear layer, followed by batch normalization, an activation function, and dropout. Batch normalization was included to improve stability during training, and dropout to prevent overfitting, as described in Section 2.4.5. Dropout was treated as a tunable hyperparameter, with values ranging from 0 to 0.5, allowing the optimization process to determine whether regularization was beneficial and, if so, to what extent. In contrast, batch normalization was not treated as a hyperparameter; it was consistently included in all configurations. This decision was based on initial manual experiments, which indicated improved performance when batch normalization was applied. To reduce the complexity of the hyperparameter search space, it was therefore fixed.

Each hidden module in the FFNN is structurally identical to the input module. Depending on a hyperparameter setting, residual connections may be included; this is illustrated by the dotted lines in the figure. Residual connections were considered due to their potential to improve the training of deep models, as explained in Section 2.4.5. The FFNN architecture allows for up to four stacked hidden layers-which does not constitute an exceptionally deep network-the inclusion of residual connections was made configurable to assess whether they could still provide benefits in this context.

The final component of the FFNN, the output module, comprises a linear layer, optionally followed by a sigmoid activation function. For the binary classification task, the sigmoid activation was used to ensure the outputs represented class probabilities for the positive class. In contrast, for the regression task, the inclusion of the sigmoid activation was treated as a tunable hyperparameter.

While sigmoid functions are standard in classification tasks due to their probabilistic

interpretation, they are not typically used in regression settings, where unbounded outputs are often required. However, in this work, the regression task involved target values constrained to the [0,1] interval, motivating the consideration of the sigmoid activation. Preliminary experiments indicated that, without the sigmoid function as a bounding mechanism, the model occasionally produced outputs outside the valid range. Introducing a sigmoid function mitigated this issue but introduced its own limitations: values already within the valid range could be non-linearly distorted. For example, a raw model output of 0 would be mapped to 0.5 after applying the sigmoid, potentially degrading prediction quality. Due to these trade-offs, the use of sigmoid activation in the regression task was included as a tunable design choice.

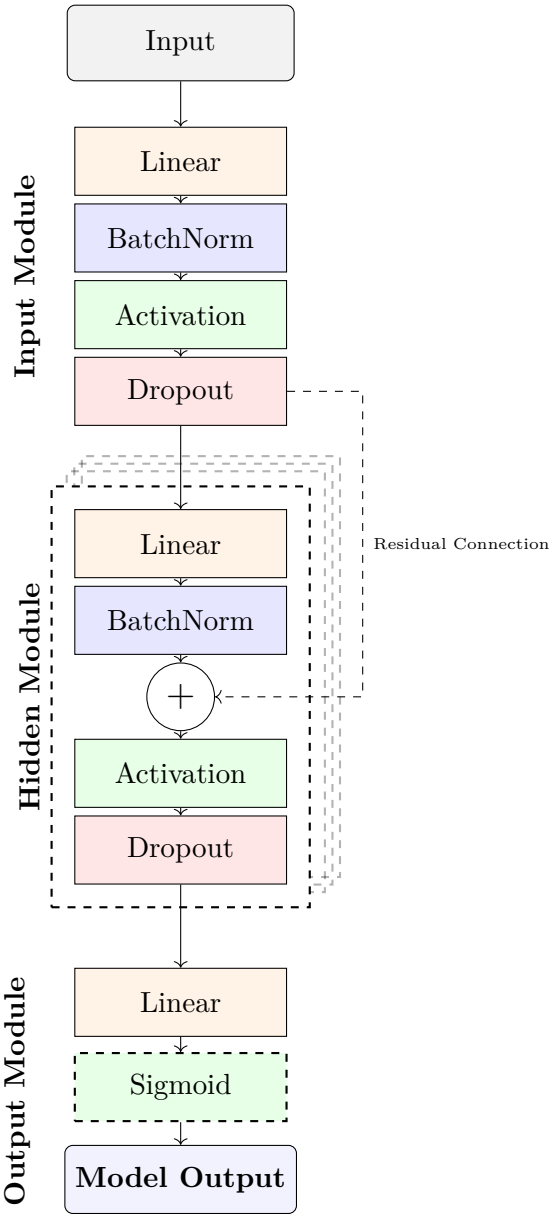


Figure 3.10: FFNN architecture with Stacked Hidden Layers.

The subsequent custom model is the HybridLSTM architecture. Given that LSTM networks are a type of RNN, thereby capable of capturing temporal dependencies in sequential data, as outlined in Section 2.4.6, the input was partitioned into two distinct streams: one comprising sequential features and the other static features. This design choice is motivated by the assumption that the temporal evolution of sequential features may contain valuable information relevant to the prediction task, whereas static features lack such temporal characteristics.

The architecture, illustrated in Fig. 3.11, is composed of three principal components: an LSTM module processing the sequential input, a static module that applies linear transformations to the static input, and a concatenation module that combines the outputs of both streams and generates the final output. The sequential input includes two time-dependent features, PRB-U and mean power consumption, while the static input encompasses the non-temporal features described in Section 3.4.

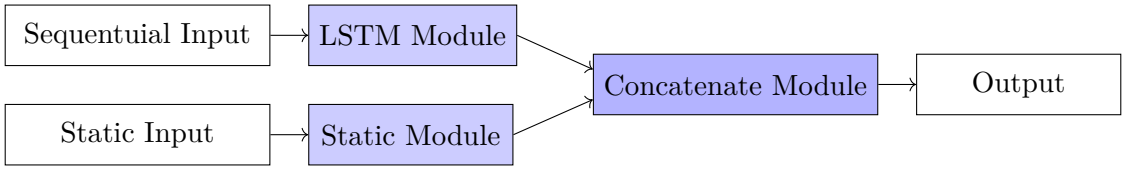


Figure 3.11: HybridLSTM with its modules and input streams.

Fig. 3.12 presents each module in the HybridLSTM in detail. The LSTM module, shown in part a), consists of two LSTM blocks. The first block contains between one and four stacked LSTM layers. As part of the hyperparameter optimization, the dropout rate was treated as a tunable parameter to determine whether dropout should be applied between the stacked layers, following the same approach as in the FFNN architecture.

A residual connection was incorporated to allow the input to bypass the first LSTM block. This design was motivated by the need to preserve critical information that might otherwise be diminished through sequential transformations (see Section 2.4.5). Initial experiments revealed that the LSTM architecture struggled to extract meaningful patterns from the data. As a result, several architectural variations were explored, and the inclusion of residual connections consistently improved both training stability and predictive performance.

The output from the first LSTM block is combined with the residual connection and subsequently passed through a batch normalization layer and a ReLU activation function. As with the residual connection, the inclusion of batch normalization was motivated by preliminary experiments, which showed improved training stability and performance. Depending on the hyperparameter setting, dropout may then be applied, with values ranging from 0 to 0.5. Finally, a second LSTM block, consisting of a single LSTM layer, is added to further abstract the fused representations produced by the residual-enhanced output of the first block. The decision to include this final LSTM layer was also based on early manual experiments, which indicated performance benefits.

The Static Module, shown in part b), consists of two hidden linear layers with ReLU activation functions. The two hidden layers do not necessarily have the same dimensionality; their sizes are treated as independent hyperparameters. Dropout may be applied after the first linear layer, controlled by the same hyperparameter used for the LSTM module. This choice was made to reduce the number of configurations explored during hyperparameter optimization.

The final component of the HybridLSTM architecture, the concatenation module, first concatenates the outputs of the LSTM and Static modules. Batch normalization is then applied to the combined representation, followed by optional dropout, controlled by the same hyperparameter as in the LSTM and Static modules. Finally, a linear layer transforms the concatenated vector into a single output value.

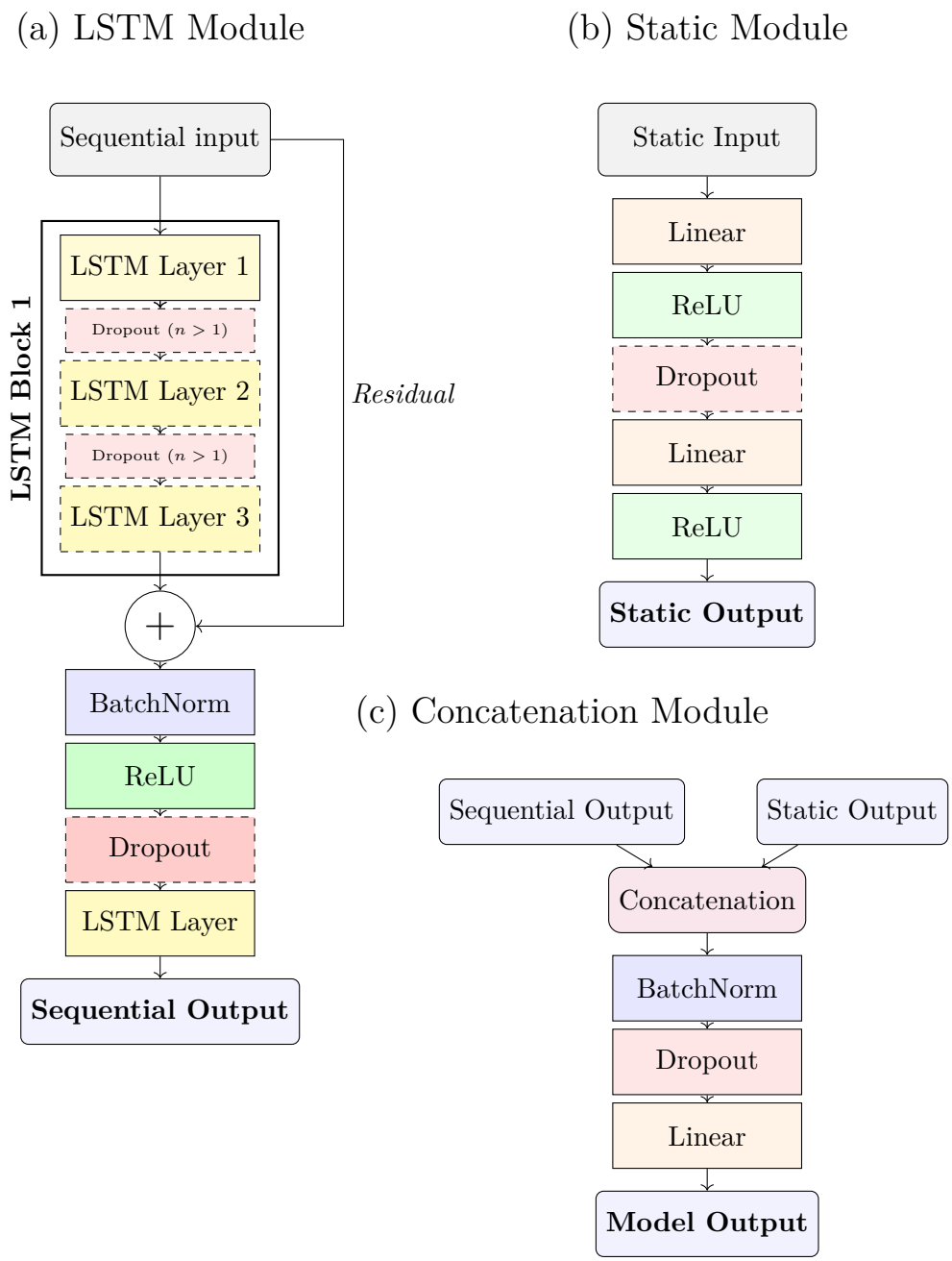


Figure 3.12: Overview of the components of the HybridLSTM architecture.

Next, the HybridCNN neural network is presented. The modules of the network are connected as shown in Fig. 3.13. CNN layers process time series by convolving over the interval and performing operations on the input to uncover patterns in the data. However, unlike the FFNN architecture in Fig. 3.10, CNN layers cannot simultaneously handle both temporal and non-temporal patterns within the same input stream. Therefore, the inputs were divided into three separate streams.

In contrast to FFNN and HybridLSTM, the two sequential features were split into two dedicated streams: one for PRB-U and one for mean power consumption. These are shown as Input 1 and Input 2 in Fig. 3.13. Separate branches were created to

allow for different hyperparameters in each of the two blocks. Input 3 contains the non-sequential features such as the hour and day, these non-sequential features were previously defined in Section 3.4.

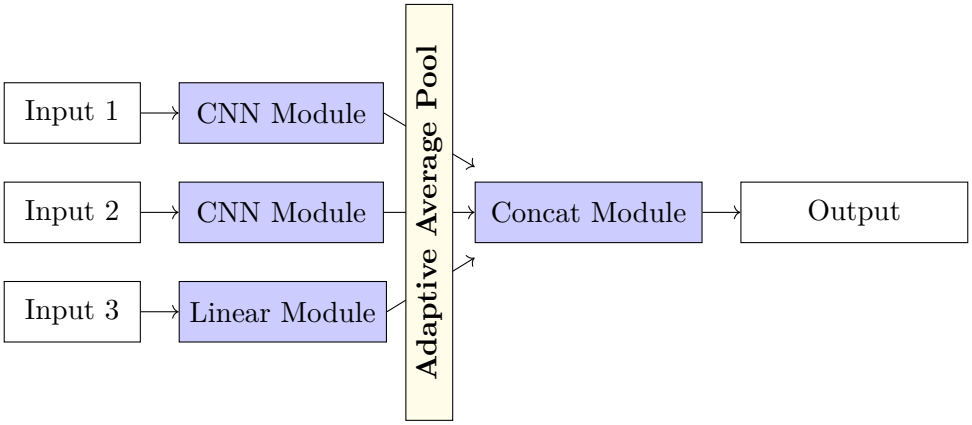


Figure 3.13: HybridCNN with its modules and input streams.

The HybridCNNs hyperparameter space was carefully constructed, as certain combinations of hyperparameters are invalid for CNN layers. Each sampled hyperparameter combination was checked and, if necessary, modified before the hyperparameter optimization was carried out using Optuna. These checks occurred at multiple levels. First, kernel sizes were always sampled from a space where the lag defined the maximum value – ensuring that the kernel size never exceeded the input length. Second, after the optimizer selected a set of hyperparameters, the resulting output size of the CNN modules was evaluated. If the output size was smaller than 1, the stride and kernel size were iteratively reduced by one until the output size became greater than or equal to 1.

Fig. 3.14 shows the modules of the CNN. (a) shows the architecture of the two sequential input streams. Each stream goes through a convolutional layer convolving over the input through a specified kernel size that takes steps to the next convolution as the hyperparameter stride specifies. The convolutional layer also has the option to pad the input depending on a specified hyperparameter. The next layer is an activation function applied to its input. The next layer is max pooling which is a down sampling operation utilized by sliding a specified window size called kernel over the input and only keeping the maximum value that it covers in each step. The next layer is Adaptive Average Pool averages the input to a new specified size according to an input hyperparameter. The flatten layer changes the size of the data from (batch_size, channels, length) to (batch_size, channels * length) which is of importance for CNN layers as they output several feature maps.

The linear module in (b) is a simple stacking of linear layers with an activation function between. The concatenation module in (c) is similar, all data goes through a layer of adaptive average pool before being concatenated and processed by a linear layer, an activation function, another linear layer and lastly the sigmoid function.

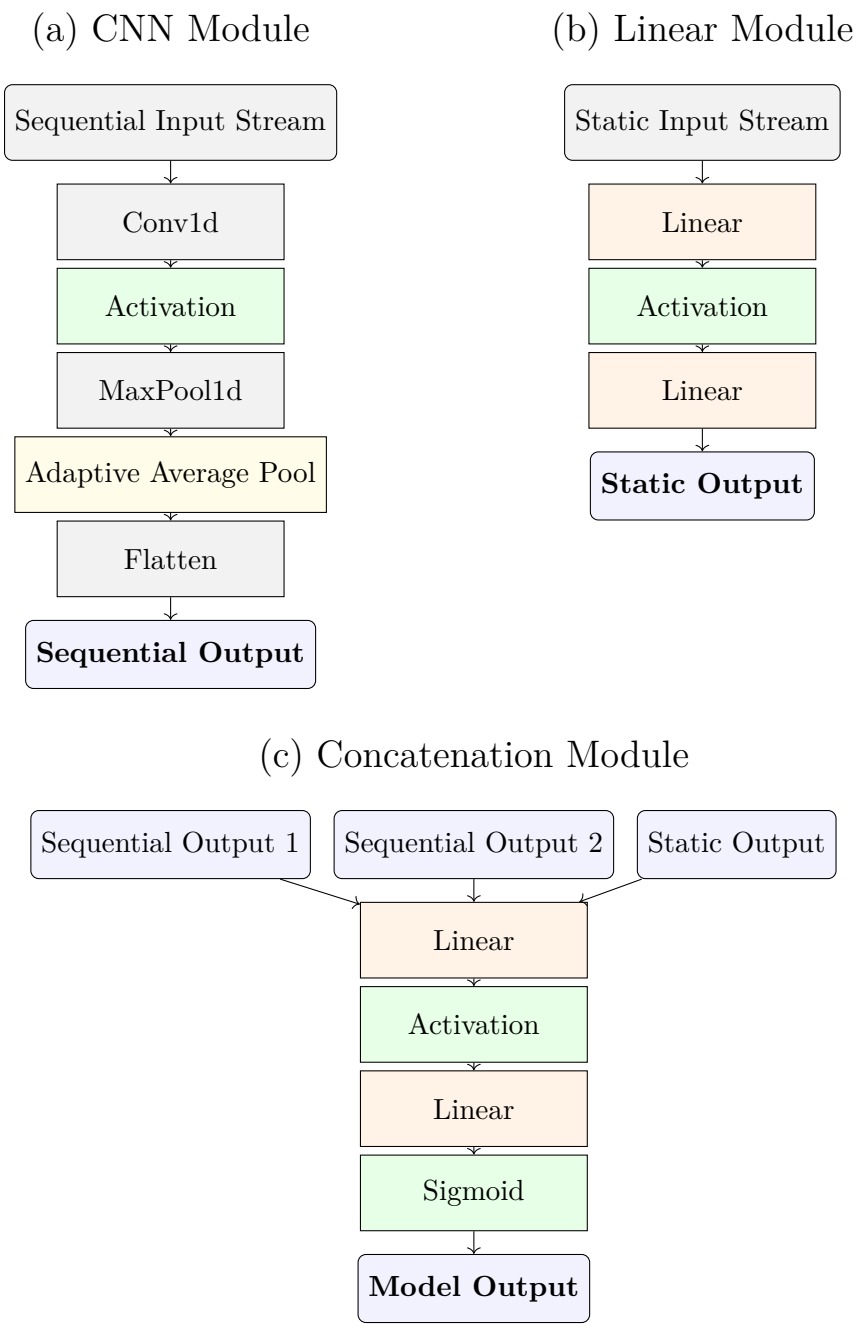


Figure 3.14: Overview of the components of the HybridCNN architecture.

The TFT neural network was implemented using the `PyTorch Forecasting` library later mentioned in Section 3.11. The library simplifies the configuration and tuning of various components of the TFT architecture, including the number of LSTM layers, the use of dropout, and whether to share a variable selection network between the encoder and decoder. These settings were included in the hyperparameter optimization process. The `PyTorch Forecasting` library also provides a *TimeSeriesDataSet* class, which was used to structure the data for training the TFT. This interface simplified the tuning of additional parameters such as the lengths of the encoder and decoder windows.

Sequential and static features are specified separately at model definition, and the library internally handles the separation of input streams. As a result, only minimal adjustments to the data and task setup were required, while still allowing the model to process sequential and static inputs through distinct streams.

3.7 Hyperparameter Tuning

A hyperparameter tuning pipeline was created for the selected models. All models were tuned using the Optuna framework, as detailed in Section 2.5. Optuna was chosen due to its ability to efficiently explore the hyperparameter space using advanced optimization techniques. Compared to traditional grid or random search, Optuna uses smarter search strategy such as Bayesian search methods [50].

Hyperparameter tuning was conducted separately for the regression and classification formulations. For the regression task, a standard approach is to evaluate model performance using MSE to compare how close to the true values the predictions are. However, selecting the best configuration based only on MSE may not yield the most effective model for the specific objective of switching voltage levels between two settings. While the model with the lowest MSE may capture the underlying patterns in the data most accurately, it does not necessarily optimize for the desired switching behavior between two levels.

To account for both general predictive performance and task-specific accuracy, hyperparameter tuning for the regression task was conducted in two distinct ways. In the first approach, performance was evaluated directly as a regression task using MSE. In the second approach, the continuous regression outputs were mapped to their corresponding voltage levels as explained in Section 3.1, and performance was evaluated as a binary classification task.

To evaluate the binary switching behavior, accuracy might initially appear to be an appropriate metric. However, as discussed in Section 3.3.1, the target distribution is highly imbalanced. To avoid favoring model configurations that predominantly predict the majority class, the F1 score was used as the evaluation metric for the binary setting.

This led to three separate settings for hyperparameter tuning:

1. Regression tuned using MSE
2. Regression tuned using F1 score (after thresholding)
3. Classification tuned using F1 score

3.7.1 Training specifications

For the binary classification task, class imbalance is addressed by computing a positive class weight, which is used with the internal loss function binary cross-entropy loss. For the regression task, models are trained using their respective default loss

functions: MSE for RF and XGB, epsilon-insensitive loss for SVR, neighbor averaging for KNR, and quantile loss with nine quantiles for the TFT. The custom neural networks use MSE as their internal loss function for the regression setting.

For the neural models, the number of training epochs was fixed based on preliminary analysis of training and validation loss curves from an FFNN model with standard hyperparameter values trained on the regression task for 200 epochs. As shown in Fig. 3.15, both the training and validation losses plateau after approximately 125 epochs. Consequently, the number of epochs was set to 125 for all neural network models in order to reduce the size of the hyperparameter search space. The y-axis in Fig. 3.15 is displayed on a logarithmic scale to improve readability. Without log-scaling, the large loss reduction in the initial epochs would dominate the plot, making it difficult to observe the smaller improvements in later epochs.

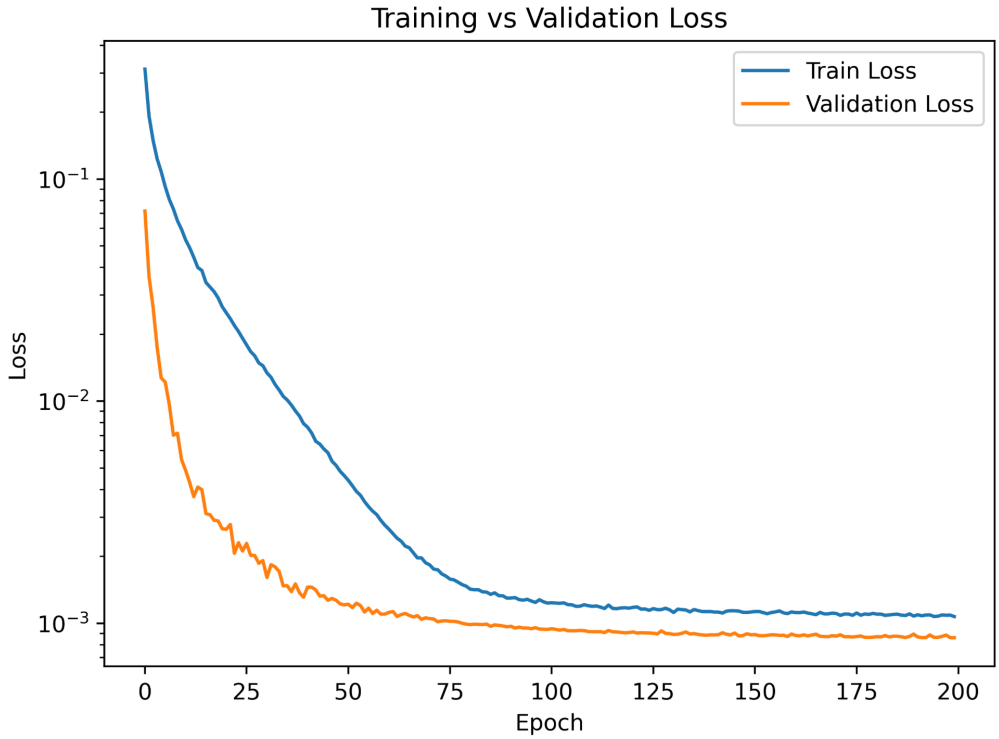


Figure 3.15: Training and validation loss for FFNN with log scaled y-axis.

3.7.2 Tuning specifications

A modular pipeline was developed to facilitate hyperparameter tuning using the Optuna library. This pipeline consists of three main components: (1) definition of search spaces, (2) an objective class that encapsulates training and evaluation logic, and (3) a tuning loop that manages execution and storage.

Hyperparameters are defined through predefined functions that take the Optuna trial object as input and return model-specific and shared parameters. For example, neural models use a shared search space function for common hyperparameters such as the learning rate, gradient clipping, scaler, and lag, but also has their own

for model specific hyperparameters. The full list of hyperparameters per model is demonstrated in Appendix B.

The core logic is implemented within an Objective class that overloads the `__call__` method. The objective class is called for each trial in the tuning loop. Within the Objective call function, the input data is lagged according to the suggested *lag_value* and transformed into binary targets if needed. The model is then constructed using the suggested hyperparameters and trained on the provided data. After training, predictions are evaluated on the test set using the F1 score or MSE depending on the current task formulation. This score is returned by the call function to the Optuna study.

The tuning loop defines the study using the `TPESampler` and `MedianPruner`, and manages storage through a local SQLite database. Completed trials are tracked, and tuning is resumed if the desired number of successful trials has not been reached. Results are exported to CSV for later analysis. Each model–task combination was tuned using 100 trials, which aligns with the minimum number of trials recommended when using the `TPESampler` [55]. All search spaces are specified in Appendix B.

3.7.3 Hyperparameter Analysis

Following the tuning phase, the results from all 100 trials for each model were saved as CSV files. Each row in these CSV file make up one trial and consists of the score of the trial (either F1 score or MSE) and the hyperparameter configuration used. These files were used to analyze the influence of various hyperparameters on model performance. A central focus of the analysis was the lag length.

The analysis was carried out in multiple ways. First, a RF was fitted on each tuning-results file to gain deeper insights into the influence of specific hyperparameters. For each tuned model, an RF was trained using the trial data: the hyperparameter configurations served as input features ($\mathbf{x}$), and the corresponding tuning scores (e.g., F1 or MSE) were used as the target variable (y). The fitted RF approximates the relationship between the hyperparameter space and model performance.

SHAP values were then computed for the RF to assess the contribution of a selected hyperparameter, specifically, lag length. This procedure was repeated for all models, and the resulting SHAP values for lag were visualized both per model and aggregated across models within each task and tuning specification (classification or regression, F1- or MSE-optimized) to identify general patterns and task-specific differences.

In addition to the aggregated analysis, a focused comparison was conducted for the temporal models, HybridLSTM and TFT. The aim was to investigate whether incorporating explicit lagged inputs provides meaningful benefits beyond the models internal temporal mechanisms, as discussed in Section 3.6.2. By analyzing the SHAP-based importance of lag within the HybridLSTMs tuning results, the extent to which the model utilizes explicit temporal features, relative to its recurrent architecture, was assessed.

3.8 Synthetic Data Generation

The trained models were further evaluated on a shifted dataset that corresponds to a period of higher radio utilization. This modified dataset was constructed to increase the number of positive class instances, thereby improving class balance and offering a more generalizable basis for model evaluation. Importantly, this data was held out during training and used exclusively for testing purposes.

The new dataset was created by modifying the target variable, PRB-U, of the previous test data, using a change factor (cf). The change factor was set to *2* in the performed stress test. Input variables were subsequently adjusted according to their correlations with the target, as previously shown in Fig. 3.9. The correlations were as follows, corr(PRB-U, MeanPowerConsumption) = 0.86, corr(PRB-U, MaxPowerConsumption) = 0.82, corr(PRB-U, MinPowerConsumption) = 0.72. This transformation is intended to simulate a busier period for the radios. The specific changes applied were as follows:

1. $PRB - U_{new} = PRB - U * cf$
2. $MeanPowerConsumption_{new} = MeanPowerConsumption * cf * 0.86$
3. $MaxPowerConsumption_{new} = MaxPowerConsumption * cf * 0.82$
4. $MinPowerConsumption_{new} = MinPowerConsumption * cf * 0.72$

Furthermore, the power consumption variables were restricted to a maximum value of 600 and PRB-U to 1 as these are their boundaries. The periods of sleep mode remained unaffected by the shifts as PRB-U is set to 0 during those times.

3.9 Evaluation

While the previous Section focused on optimizing models using standard metrics MSE and F1 score, this Section presents additional strategies used to evaluate the resulting best-performing configurations in the context of dynamically switching voltage levels in radios. In addition to reporting MSE and F1 score, the evaluation includes task-specific analyses designed to better capture practical performance in a deployment setting.

An important emphasis is put on the asymmetry in under- and overestimating the target variable. An underestimation of *PRB-U*, i.e. a false negative, results in the voltage being set too low, which may cause temporary issues for connected users. Conversely, overestimating utilization leads to unnecessary energy consumption. Therefore, under prediction and over prediction of power have two distinct consequences.

3.9.1 Metrics

The two main metrics for model evaluation are *percent energy saving* and *under estimations*. The percentage energy save is calculated as the fraction of the cur-

rent integral of power and the change in P_{in} occurring from the change in voltage. The ideal case would maximize the energy saving in Eq. 3.4, while minimizing the amount of underpredictions in Eq. 3.5. $P_{\text{in static}}$ refers to the power (W) with no change depending on utilization while $P_{\text{in dynamic}}$ is the adjusted power depending on utilization. $\hat{y}$ in Eq. 3.5 is the predicted target variable, the metric counts the times the predicted is below the true value y. The variables $t1$ and $t2$ refer to a specified time interval.

$$E_{saved} = \frac{\int_{t1}^{t2} P_{\text{in static}} - P_{\text{in dynamic}}}{\int_{t1}^{t2} P_{\text{in static}}} * 100\% \tag{3.4}$$

$$Underestimations = \sum_{i=t1}^{t2} \mathbf{1}_{\{\hat{y}_i < y_i\}} \tag{3.5}$$

The metrics above were implemented as it would be problematic to only measure the models by *mean squared error* (MSE) because an over prediction and under prediction have different consequences. An over prediction resulting in energy waste while an under prediction leads to problems in radio performance. MSE, defined in Eq. 3.6, does not account for any difference between over- and under-predictions. The same issue applies to *mean absolute error* (MAE), defined in Eq. 3.7.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{3.6}$$

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{3.7}$$

F1 score is introduced as it evaluates the models ability to make correct positive predictions while penalizing both over- and under-prediction, making it suitable for assessing performance in more imbalanced classification tasks. *TP* (true positives) refers to instances correctly classified as belonging to the positive class. *FP* (false positives) are instances incorrectly classified as positive. *FN* (false negatives) are instances that were wrongly classified as negative despite belonging to the positive class. The metric is defined in Eq. 3.8 [56]. The metric was calculated using the *f1_score* metric from `Sklearn.metrics` [56].

$$F1 = \frac{2 \cdot TP}{2 \cdot TP + FP + FN} \tag{3.8}$$

Lastly, the energy consumed during prediction for the two best performing models where calculated. The `codecarbon` library [57] was used to estimate the energy consumption in kilowatt-hours (kWh) for these model. This evaluation ensured that the energy savings achieved by deploying the model exceeded the energy required to run it. We define the *Energy Ratio* (R_E) for each model as Eq. 3.9. Where E_{model} is the energy consumed by the model and E_{savings} is the savings of the model from adapting a dynamic voltage.

$$R_E = \frac{E_{\text{model}}}{E_{\text{savings}}} \tag{3.9}$$

E_{model} is the energy consumed by the model during prediction, calculated as the mean value of 10 runs. E_{savings} is the total potential energy saved by applying the model. A lower value of R_E indicates greater energy efficiency, with values less than 1 implying net positive energy savings.

Table 3.4 displays optimal values for the classic ML metrics used. The optimal value for energy saving and underestimations are stated in the result Section as it is based on the test data set.

Metric	Range	Optimal
MSE	$[0, \infty)$	0
MAE	$[0, \infty)$	0
F1	[0, 1]	1
Prediction Energy	$[0, \infty)$	0

Table 3.4: Optimal values and ranges for the metrics.

3.9.2 Threshold Tuning

Since underestimations are considered more serious than overestimations in this application, an additional analysis was performed to investigate how the number of underestimations could be minimized by adjusting the decision threshold for predicting class 1. This threshold determines the minimum predicted probability required for the model to classify a sample as class 1. The analysis also examined how this adjustment influenced the amount of energy saved. The aim was to determine whether certain models were more effective at reducing the number of restarts while still preserving a substantial portion of the energy savings, whereas others might exhibit a sharper trade-off.

This analysis was conducted for the classification task, where class probabilities are readily available. The threshold was varied from 0.01 to 0.99, and for each model, the trade-off between the number of restarts and the corresponding energy savings was visualized.

3.9.3 Benchmarking Models

A selection of benchmarking models were implemented to evaluate the performance of more advanced approaches. These baselines served as reference points for comparison and specifically for comparing in ML based methods are beneficial in this specific case, or if cheaper statistical methods could perform as good or better. First, the naive benchmark [58] assumed that the value at the next time step will be equal to the value at the current time step, as shown in Eq. 3.10:

$$x_t = x_{t-1} \tag{3.10}$$

Second, the mean benchmark [58] predicted the next value as the average of the previous n values, as shown in Eq. 3.11. The naive mean was implemented for n=1, ..., 96 with 96 as the upper limit corresponding to a full 24-hour period.

$$x_t = \frac{\sum_{b=1}^{n} x_{b-1}}{n} \tag{3.11}$$

Third, the Autoregressive Integrated Moving Average (ARIMA) model was used. ARIMA makes predictions by combining past values (AR), past errors (MA), and differencing to remove non-stationarity (I) [20]. Eq. 3.12 illustrates the ARIMA model for a stationary time series:

$$y_t = \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \cdots + \theta_q \epsilon_{t-q} \tag{3.12}$$

Where:

- y_t is the value of the target variable in the current point in time
- $\phi_1, \phi_2, \ldots, \phi_p$ are the autoregressive (AR) parameters
- ϵ_t is the error term at time t
- $\theta_1, \theta_2, \ldots, \theta_q$ are the moving average (MA) parameters, indicating the dependence on previous errors in forecasting

3.10 Explainability

Explainability techniques were applied to selected models after evaluation, to gain a deeper understanding of model behavior and decision-making. These methods provide insight into which input features the models rely on. In addition to guiding future improvements, this level of transparency can be especially important for users or stakeholders without a technical background in AI, as it helps them understand and trust the models decisions.

Two models were selected for the explainability analysis. Given the importance of computational simplicity and energy efficiency in this application, one model was chosen from the statistical methods. To enable comparison between fundamentally different model methods, a neural network-based model was also selected. This allowed for examination of whether the models emphasized similar or different features when making predictions.

The selected models were models which had achieved the lowest number of restarts while maintaining acceptable energy savings, since this trade-off constituted the

primary evaluation criterion. The objective was to understand not only which features were most influential, but also how different types of models approached the challenge of minimizing restarts without sacrificing performance.

Shapley Additive exPlanations (SHAP) were used to analyze model explainability [13]. For the statistical model, SHAPs `TreeExplainer` was applied, while the neural network model was examined using the `GradientExplainer`. In both cases, SHAP summary plots were generated to identify the most influential features and to illustrate how feature values contributes to the models predictions. Additionally, waterfall plots were created to explain individual predictions. These plots were generated for one instance of each output type, i.e., True Positive, False Negative, True Negative, and False Positive, for both models.

3.11 Libraries

The libraries used throughout this thesis are summarized in Table 3.5, along with their respective application categories and a brief description of how they were used. These include libraries for implementing machine learning models, hyperparameter tuning, and visualization.

Library	**Category**	**Description**
Codecarbon	Evaluation	Used in the evaluation to measure model prediction kWh intensity [57].
Matplotlib.pyplot	Visualization	Used to generate all line plots with a consistent template [59].
NumPy	Numerical computing	Used for efficient array operations and numerical computations [60].
Optuna	Hyperparameter tuning	Used for automated hyperparameter optimization [61] with `TPESampler` [62] and `Median Pruner` [63]
Pandas	Data manipulation	Used for handling tabular data, preprocessing, and transforming features [64].
PyTorch	Neural networks	Used to construct the FFNN, HybridLSTM, and HybridCNN models using core layers such as `Linear` [65], `Dropout` [66], `BatchNorm` [67], `LSTM` [68], `Conv1d` [69], and pooling layers `MaxPool1d` [70], `AdaptiveAvgPool1d` [71]
PyTorch Forecasting	Time series library	Used for implementing the TFT model [72] and preparing data with the `TimeSeriesDataSet` class [73]. The library is built on `PyTorch Lightning` [74] to simplify time series forecasting.
Scikit-learn	Statistical models	Used to implement statistical ML models: Random Forest [75], XGBoost [29], Support Vector Machine [76], and K-Nearest Neighbors [77].
Seaborn	Visualization	Used specifically to visualize heatmaps [78].
SHAP	Explainability	Used for interpreting model predictions [79]
SKlearn.metrics	Evaluation	Used to calculate f1 score [56].

Table 3.5: Libraries used and their purposes.

4

Results

This chapter presents the results obtained from applying the proposed methodology. An analysis of lag values across task formulations, and the results from the hyperparameter optimization, is presented in Section 4.1. Section 4.2 outlines the performance of each model across the three task formulations, over all radios. Section 4.3 presents the performance on a per-radio unit granularity. Section 4.7 evaluates the energy used per model to perform the predictions. Section 4.5 explores how adjusting the classification threshold can tune the trade-off between underestimations and energy savings. Section 4.4 assesses model performance in a senario with a higher number of class 1 instances. Finally, Section 4.6 provides SHAP-based explanations of the top-performing models identified in the earlier evaluation. Throughout this chapter, any optimal value in a table column is highlighted in bold.

4.1 Hyperparameter Optimization

Optuna was used for hyperparameter tuning across all ML models. This section presents the results and insights obtained from the hyperparameter optimization process. During tuning, models were trained and validated on data aggregated from all five radios, as described in Section 3.2. The training set was used for model fitting, while the validation set guided the optimization process.

Hyperparameter tuning was performed for both task formulations: classification and regression. For the regression task, two optimization strategies were employed, using either the F1 score or MSE as the objective function. Models optimized with respect to the F1 score, including both classification and regression models, were tuned with F1 score as the objective function, whereas regression models optimized for MSE employed MSE as the tuning objective.

The specific hyperparameters tuned and their respective search ranges are presented in appendix B. The optimal hyperparameters identified for each model are listed in Appendix C.

4.1.1 Optimized Lag values

After completing the optimization loop, the optimal lag lengths for each model and task–tuning formulation were analyzed in greater detail. Table 4.1 shows the lag values identified by Optuna for each model. On average, the classification models

exhibited the shortest lag lengths, followed by the regression models optimized with MSE, and finally the regression models optimized with the F1 score.

Model	Lag classification	Lag Regression F1	Lag Regression MSE
rf	6	96	96
xgb	79	94	6
knn/r	1	1	1
svm/r	71	96	67
ffnn	5	74	25
lstm	5	22	17
cnn	85	96	95
tft	49	26	43
Average	37.6	63.1	43.8

Table 4.1: Optimal lag per model and Optuna setup.

To further analyze the effect of lag length on model performance, a RF model was trained on the results from the Optuna tuning process, using the objective scores as targets. This enabled an approximation of the relationship between hyperparameters and model performance. Unlike correlation analysis, this method can also model potential interaction effects, where the influence of lag length on the objective depends on the values of other hyperparameters. In the classification and F1-tuned regression tasks, the objective was the F1 score, while in the MSE-tuned regression task, it was the MSE. Accordingly, SHAP values were computed to assess the relative importance of the lag feature with respect to these objectives. In the classification and F1-tuned regression tasks, positive SHAP values indicate a positive contribution to performance, whereas in the MSE-tuned regression task, lower SHAP values are favorable. Fig. 4.1 presents the SHAP values for the lag feature across models and tasks. The relationship between lag length and SHAP values seem to vary across these models and tasks. For the RF model (top row), a clear pattern is visible: in the classification task, shorter lag lengths contribute positively to the F1 score; in the F1-tuned regression task, longer lag lengths are associated with higher SHAP values, indicating a preference for longer lags; and in the MSE-tuned regression task, longer lags correspond to negative SHAP values. This suggest that, for the RF, longer lags were beneficial for both regression tasks and shorter lags were preferred for the classification task.

This pattern does not generalize across all models. For instance, the KNC/KNR models (row 4) consistently favor shorter lag values across all tasks. For models that inherently capture temporal dependencies, such as the HybridLSTM (row 6) and the TFT (row 8), the SHAP values help to assess whether performance can be further improved by providing explicit lagged inputs or relying solely on internal memory mechanisms. In these models, SHAP values display similar trends across tasks: lag values contribute positively to performance up to a certain threshold, beyond which their impact becomes detrimental. Specifically, for both the classification and F1-optimized regression tasks, the HybridLSTM exhibits positive SHAP values up to a

lag length of approximately 40, while the TFT shows positive contributions up to $\tilde{\text{l}}$ag 20. Beyond these points, SHAP values become negative, indicating that larger lag inputs reduce performance in terms of F1 score.

In the MSE-optimized regression task, the trend for the HybridLSTM is reversed: lag values below approximately 20 exhibit negative SHAP values, while larger lag values yield increasingly positive SHAP values. Since lower MSE is the objective, positive SHAP values in this context indicate that these lag values negatively impact model performance. For the TFT, the pattern is less distinct; however, SHAP values tend to increase with longer lag values.

Fig. 4.2 presents the aggregated SHAP values for lag length across all models, grouped by tuning specification. The results reveal contrasting trends between the classification task and the regression task optimized for the F1 score. Specifically, the classification models tend to assign higher SHAP values to shorter lag lengths, whereas the F1-optimized regression models exhibit higher SHAP values for longer lag lengths. In contrast, the MSE-optimized regression models display less distinct patterns: many lag values have SHAP values close to zero, making it difficult to discern clear trends within this tuning specification.

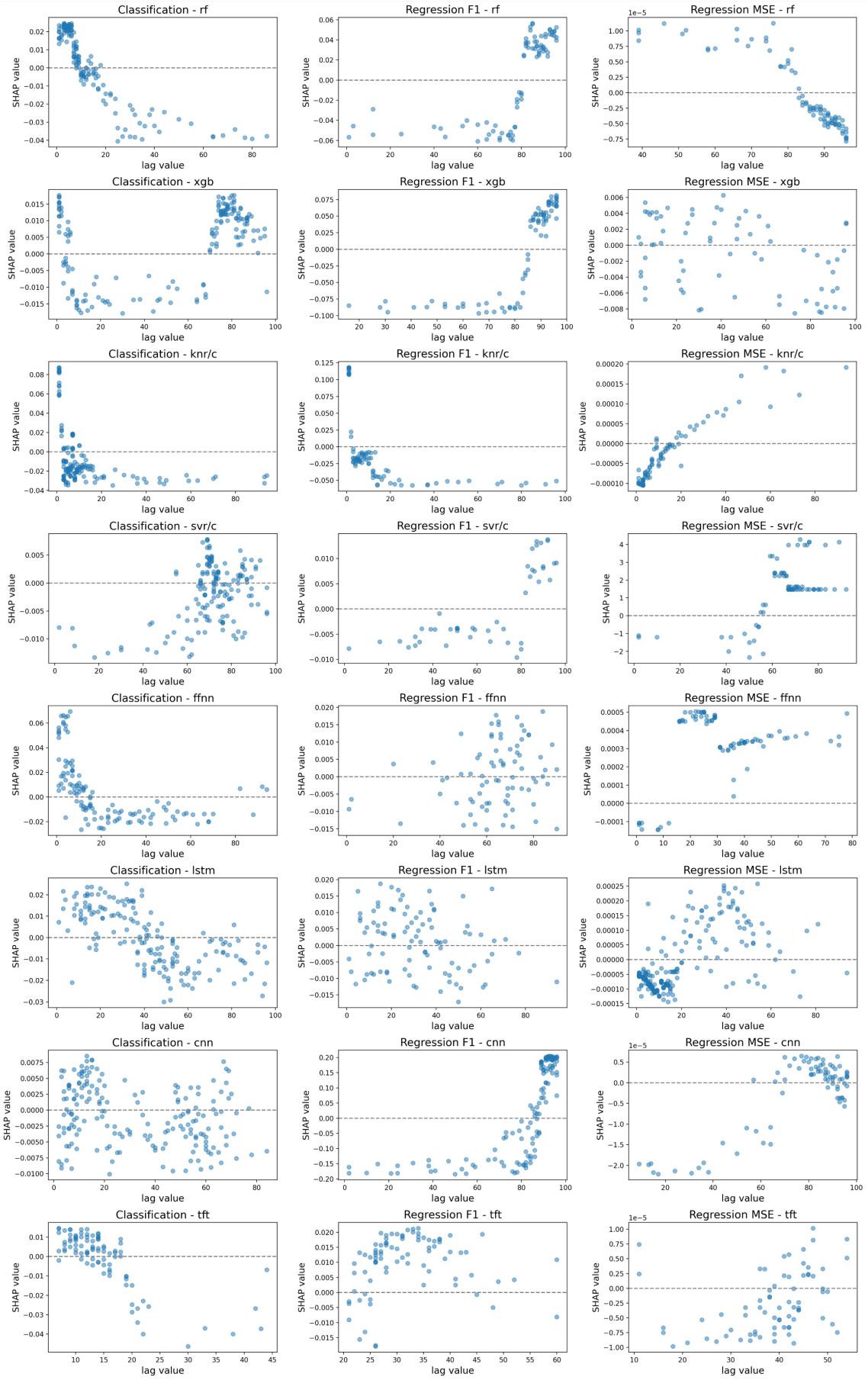


Figure 4.1: SHAP values for lag per model over the three task formulations.

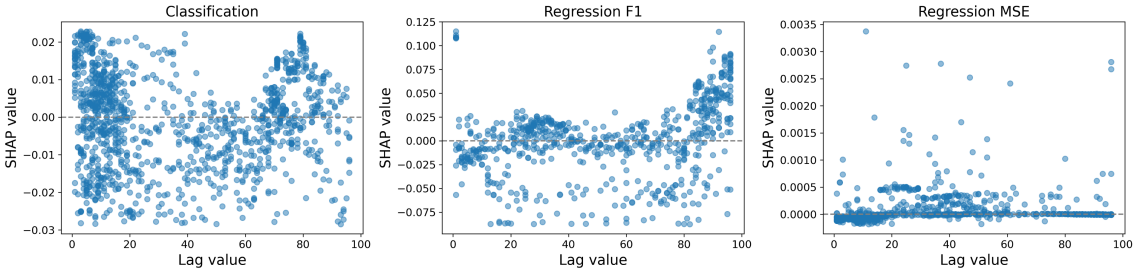


Figure 4.2: Aggregated lag SHAP values for all models over the three task formulations.

4.1.2 Model-Specific Hyperparameters

Certain components of the customized neural network architectures – FFNN, HybridLSTM, and HybridCNN – were included as hyperparameters in the optimization loop. For the FFNN model, two key architectural choices were subjected to tuning: the use of residual connections and the application of a sigmoid activation function in the regression task.

Fig. 4.3 illustrates the effect of including residual connections, with performance measured using the F1 score for the classification task and the F1 tuned regression task, and MSE for the second regression task. The results in Fig. 4.3 suggest that incorporating residual connections leads to a marginal decrease in F1 score for the classification task, comparable performance in the first regression task, and a slight improvement in MSE in the second regression task. Table 4.2 shows the number of trials were Optuna chose to include and exclude residuals. The statistics indicate that Optuna systematically favored the exclusion of residual connections in the classification setting, whereas their inclusion was more frequently selected in the regression scenarios.

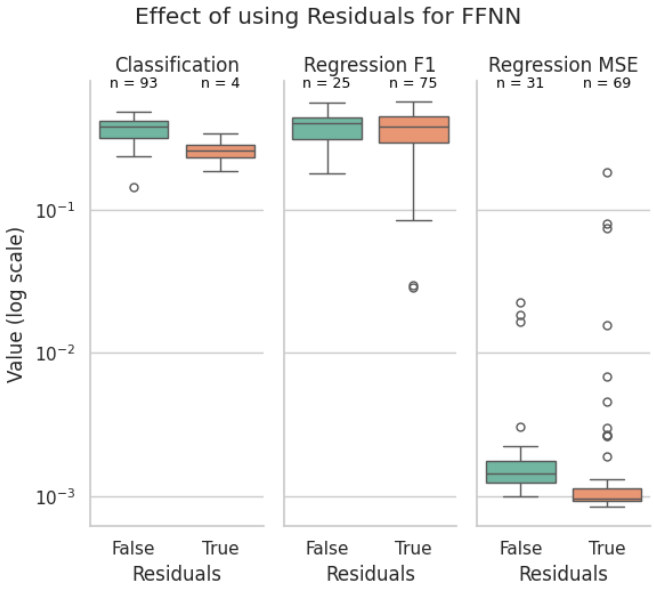


Figure 4.3: Performance of the FFNN model with and without residual connections. The y-axis shows the evaluation metric for each task: F1 score for classification and regression tuned on F1, and MSE for Regression tuned on MSE.

Setup	**Trials with Residual**	**Total Trials**
Classification	4	97
Regression F1	75	100
Regression F1	69	100

Table 4.2: Statistics over residual inclusion for FFNN trials.

Fig. 4.4 illustrates the effect of including a sigmoid activation function. In the classification task, the sigmoid function was always included, as the model was required to output class probabilities. In contrast, its inclusion was a tunable hyperparameter in the two regression tasks. The results show that applying a sigmoid activation function in regression led to reduced performance, shown as a lower F1 score in the first regression task and a higher MSE in the second. Table 4.3 shows the exact statistics over number of trials where the sigmoid function was included as a last layer. The results show that Optuna consistently favored the exclusion of sigmoid activation for regression tasks, aligning with the observed decrease in performance when it was included.

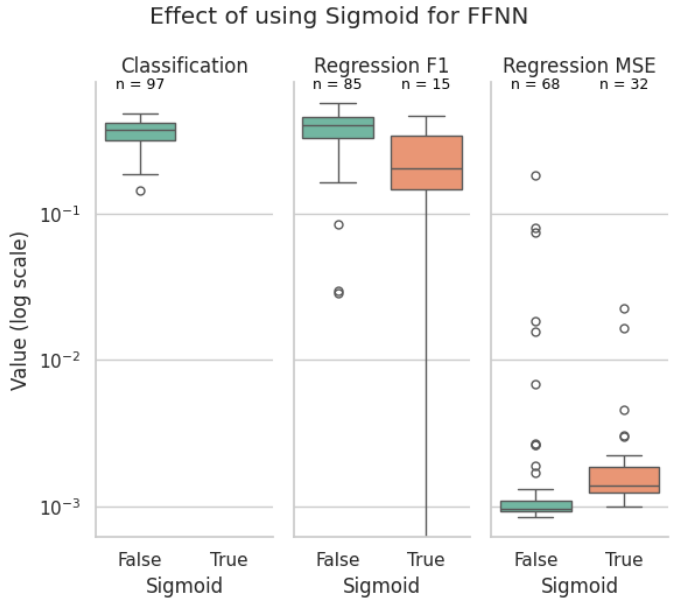


Figure 4.4: Performance of the FFNN model with and without sigmoid. The y-axis shows the evaluation metric for each task: F1 score for classification and Regression tuned on F1, and MSE for Regression tuned on MSE.

Setup	**Trials with Sigmoid**	**Total Trials**
Classification	97	97
Regression F1	15	100
Regression F1	32	100

Table 4.3: Statistics over sigmoid inclusion for FFNN trials.

Selected hyperparameters from the best-performing FFNN configurations, as determined by the Optuna tuning loop, are presented in Table 4.4. Residual connections were included in both regression models but omitted in the classification model. The classification model also employed the deepest architecture, with three layers, while both regression models used a single layer. The sigmoid activation function was not selected in either regression configuration. Dropout was included in all configurations.

Hyperparameter	**Classification**	**Regression F1**	**Regression MSE**
use_residual	False	True	True
use_sigmoid	Not tuned	False	False
dropout	0.3	0.2	0.4
num_layers	3	1	1

Table 4.4: FFNN architectural hyperparameters.

Table 4.5 presents selected hyperparameter configurations for the optimal HybridLSTM models. Dropout was included in all configurations except for the classification model. The number of LSTM layers was two for both the classification model and the regression model optimized for the F1 score, and three for the regression model optimized for MSE.

The selected lag values are presented in Table 4.1. The results are consistent with those shown in Fig. 4.1, indicating that small to medium lag lengths were preferred.

Hyperparameter	**Classification**	**Regression F1**	**Regression MSE**
params_lstm_dropout	0.0	0.5	0.1
params_lstm_layers	2	2	3

Table 4.5: LSTM architectural hyperparameters and optimal lag values.

4.2 Results on All Radios

Implemented models and benchmark methods were evaluated on the test dataset, which comprises aggregated data from all five radios. The results presented below reflect overall model performance on this combined dataset. Evaluation metrics are defined in Section 3.9. Metrics applied across all models include F1 score, energy savings, and the number of underestimations (# Underestimations), while regression models were additionally evaluated using MSE and MAE. Table 4.6 reports the potential energy savings and number of underestimations under perfect predictions on the test data, providing an upper bound for model performance. For reference, the theoretical ideal values for MSE, MAE, and F1 score (0, 0, and 1, respectively) are listed in Table 3.4.

Metric	Range	Optimal
Energy Saving	[0, 12.83]	12.83
# Underestimations	[0, 68]	0

Table 4.6: Optimal values and ranges for underestimations and energy saving.

Table 4.7 presents the evaluation results of all ML models and selected baseline models on the test data. The model names in Table 4.7 include suffixes indicating the task formulation and tuning objective. Models labeled with _c were trained as classifiers. Models labeled with _F1_r were trained as regressors and tuned using the F1 score, while models labeled with _r were also regressors but tuned using MSE. This notation is used consistently across several result tables.

The full list of 99 implemented baseline models was described in Section 3.9.3; therefore, only the top-performing models from the mean baseline group, as defined in Eq. 3.11, are included in Table 4.7. *Naive_mean_48* represents the average over

the past 48 timesteps and was included because it achieved the highest energy savings. *Naive_mean_3* was selected for having the lowest MSE, and *Naive_mean_2* for achieving the highest F1 score. *Naive_mean_1*, shown in Eq. 3.10, resulted in the fewest underestimations. Three additional benchmarks (*ARIMA*, *last_week*, *last_24h*) were included, as motivated in Section 3.9.3.

As shown in Table 4.7, the FFNN classifier achieved the highest F1 score at 0.35 and recorded the second-lowest number of underestimations. The lowest energy saving was observed for the TFT classifier, at 11.78%, while all other models saved at least 12.34% of the previously utilized kWh. In contrast, the TFT classifier achieved the best result in terms of minimizing underestimations. The highest energy savings, at 12.83%, were achieved by the *naive_mean_48* benchmark model, closely followed by the KNR and TFT regression models. However, these models also produced the maximum possible number of underestimations and an F1 score of 0, effectively never predicting the positive class correctly.

Model	F1 Score	Energy Saving (%)	# Underestimations
rf_c	0.29	12.43	42
rf_F1_r	0.15	12.79	62
rf_r	0.18	12.81	61
xgb_c	0.18	12.34	50
xgb_F1_r	0.20	12.76	59
xgb_r	0.19	12.79	60
knc_c	0.10	12.78	64
knr_F1_r	0.13	12.70	61
knr_r	0.00	12.82	68
svc_c	0.16	12.39	53
svr_F1_r	0.28	12.76	55
svr_r	0.03	12.81	67
ffnn_c	**0.35**	12.41	34
ffnn_F1_r	0.07	12.78	65
ffnn_r	0.03	12.80	67
lstm_c	0.23	12.65	54
lstm_F1_r	0.20	12.75	59
lstm_r	0.03	12.81	67
cnn_c	0.19	12.67	57
cnn_F1_r	0.10	12.78	64
cnn_r	0.06	12.72	65
tft_c	0.20	11.78	**31**
tft_F1_r	0.16	12.74	60
tft_r	0.00	12.82	68
ARIMA	0.09	12.74	64
naive_last_24	0.13	12.61	59
naive_last_week	0.03	12.66	66
naive_mean_1	0.19	12.60	55
naive_mean_2	0.20	12.67	56
naive_mean_3	0.20	12.70	57
naive_mean_48	0.00	**12.83**	68

Table 4.7: Evaluation results for all models, including benchmarks and machine learning methods, on the test dataset.

Fig. 4.5 summarizes two of the three evaluation metrics for each model: percentage energy saved and number of underestimations. The figure is based on the statistics presented in Table 4.7. The scatter plot shows a distinct separation between the classification models and the regression-based approaches.

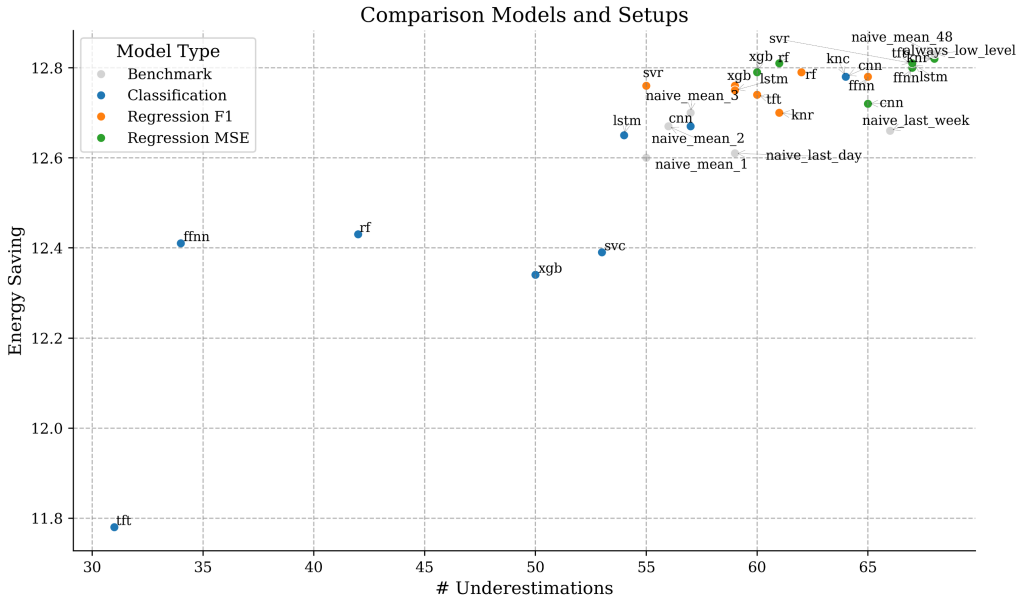


Figure 4.5: Scatter summary of all models.

The bar charts in Fig. 4.6 illustrate the performance of the three machine learning model setups across the three shared evaluation metrics: energy savings, number of underestimations, and F1 score. The first bar chart in Fig. 4.6, shows a consistent trend in energy savings across all models. Notably, the only model that fails to achieve at least 12% energy savings is the TFT classifier.

The second bar chart in Fig. 4.6 highlights a more varied performance in terms of underestimations. The classifier versions of the models generally perform best, followed by the regression model selected based on F1 score. In contrast, the regression model chosen using the more conventional MSE criterion exhibits the weakest performance.

The third bar chart in Fig. 4.6 illustrates model performance in terms of F1 score across the different setups, which shows the greatest variation among the three evaluation metrics. The figure shows that the classification variant achieves the highest F1 score in five of the eight model types. The regression variant optimized for MSE performs the worst in almost all models, except for RF, where it ranks second. The regression variant optimized for F1 score generally achieves intermediate performance.

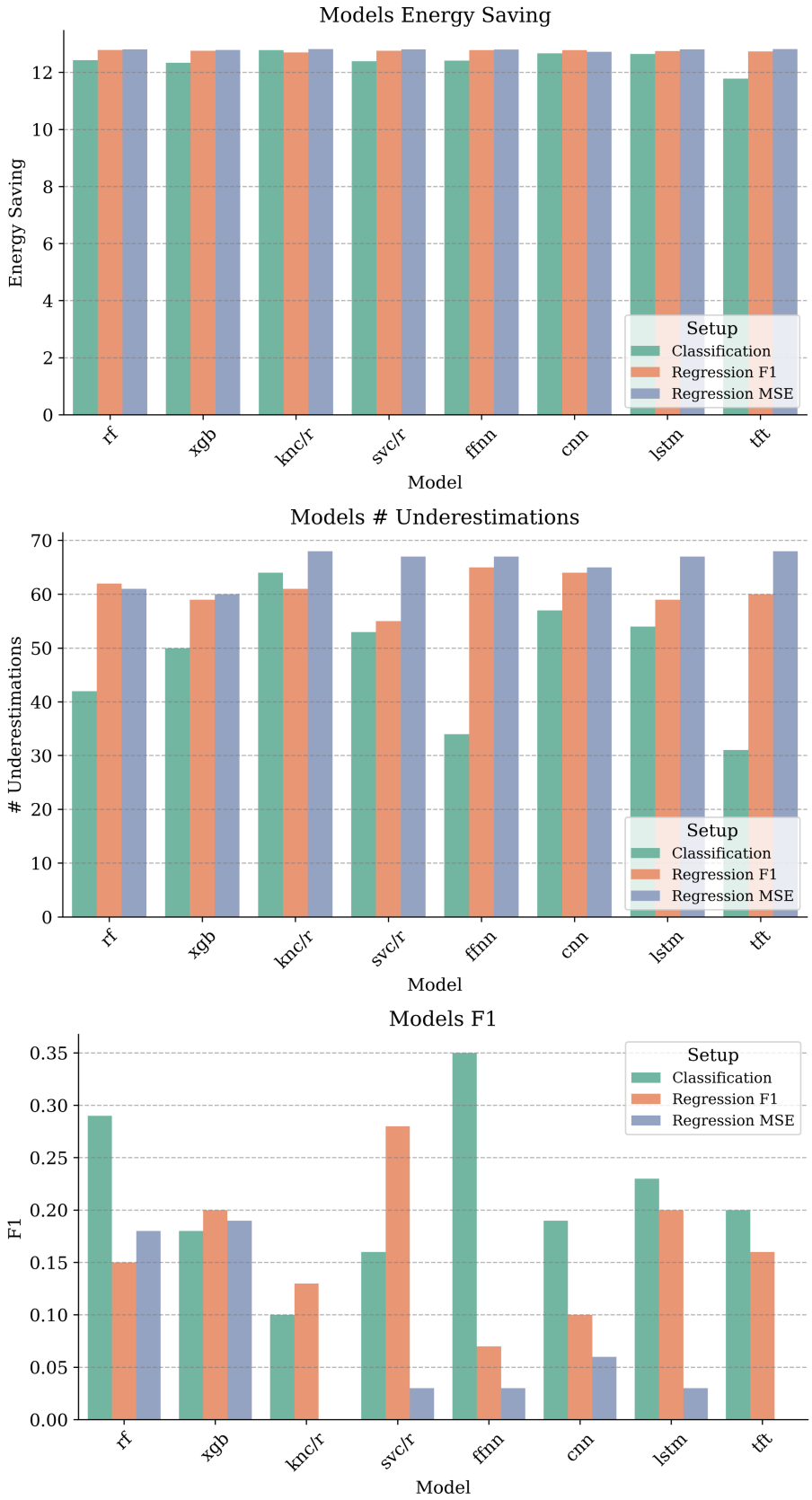


Figure 4.6: Grouped bar charts of model performance in energy savings, underestimations and F1 score.

An additional analysis was conducted on the naive mean benchmarks to examine the effect of varying the mean window size. Increasing the window size resulted in both a higher number of underestimations and greater energy savings. Accordingly, the trade-off between these two metrics was also present in the benchmark results. This trade-off is depicted in Fig. 4.7.

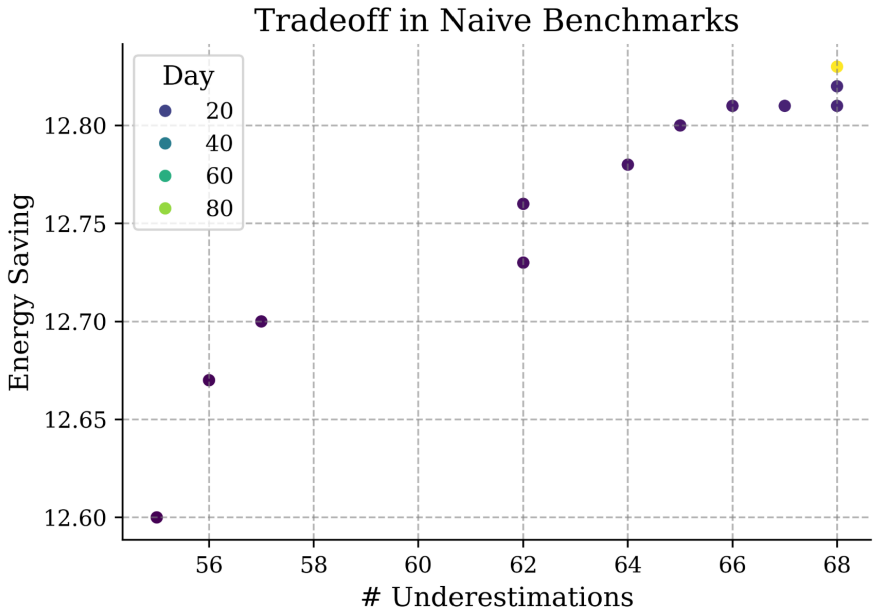


Figure 4.7: Trade-off between energy savings and underestimations across different window sizes in baseline models.

The regression models were evaluated on two additional metrics: MSE and MAE. The results are presented in Table 4.8. The Random Forest regressor selected based on MSE (row 1) achieved the best performance on both metrics, closely followed by the same architecture selected based on F1 score (row 2). Additionally, the XGB model, also based on decision trees, produced comparable results. In contrast, the TFT model selected using MSE showed the weakest performance on both MSE and MAE.

model	MSE	mae
rf_r	**0.00066**	**0.01407**
rf_F1_r	0.00068	0.01428
svr_r	0.00072	0.01451
svr_F1_r	0.00070	0.01506
knr_r	0.00073	0.01494
knr_F1_r	0.00108	0.01762
xgb_r	0.00069	0.01458
xgb_F1_r	0.00070	0.01460
ffnn_r	0.00073	0.01677
ffnn_F1_r	0.00080	0.01730
lstm_r	0.00081	0.01680
lstm_F1_r	0.00085	0.01779
cnn_r	0.00078	0.01628
cnn_F1_r	0.00095	0.02105
tft_r	0.00495	0.05320
tft_F1_r	0.00107	0.02001
ARIMA	0.00107	0.01971
naive_last_24	0.00223	0.02672
naive_last_week	0.00290	0.03479
naive_mean_1	0.00129	0.01945
naive_mean_2	0.00118	0.01936
naive_mean_3	0.00118	0.01988
naive_mean_48	0.00390	0.04822

Table 4.8: Results from regression specific evaluations.

4.3 Results on Individual Radios

Since the five radios differ in their operational patterns, such as the number of positive class instances and the range of feature values, the performance of each model was further evaluated on a per-radio basis. This analysis was conducted by filtering the test data for each individual radio and generating predictions using the pre-trained models. It is important to note that the models were not retrained for each radio; they remained trained on the aggregated data from all five radios. This procedure was solely intended to examine how well the models generalized to individual radios and to identify any variability in performance across different devices.

Table 4.9 presents the upper limit of energy savings and the number of positive class instances (i.e., number of underestimations) for each radio. Tables 4.10 through 4.14 show the evaluation results for all models on a per-radio basis.

Table 4.10 shows the evaluation for radio 1. It is highly imbalanced and only has three possible underestimations and thus three instances in the positive class. None of the machine learning or benchmark methods were able to accurately predict any

Radio	Energy Saving	# Underestimations
1	12.87	3
2	13.04	30
3	12.75	2
4	13.01	0
5	12.42	33

Table 4.9: Upper range of metrics for each unit.

of these instances. The benchmark models performed comparably to the machine learning models on this test set. No model demonstrated superior performance; accordingly, no values are highlighted in bold.

model	F1	Energy Saving	# Underestimations
rf_c	0.0	12.84	3
rf_F1	0.0	12.87	3
rf_r	0.0	12.87	3
xgb_c	0.0	12.83	3
xgb_F1	0.0	12.87	3
xgb_r	0.0	12.85	3
knc_c	0.0	12.84	3
knr_F1	0.0	12.86	3
knr_r	0.0	12.87	3
svc_c	0.0	12.81	3
svr_F1	0.0	12.84	3
svr_r	0.0	12.84	3
ffnn_c	0.0	12.73	3
ffnn_F1	0.0	12.87	3
ffnn_r	0.0	12.87	3
lstm_c	0.0	12.87	3
lstm_F1	0.0	12.84	3
lstm_r	0.0	12.87	3
cnn_c	0.00	12.84	3
cnn_F1	0.0	12.84	3
cnn_r	0.0	12.87	3
tft_c	0.0	12.81	3
tft_F1	0.0	12.87	3
tft_r	0.0	12.87	3
arima_r	0.0	12.85	3
naive_mean_1	0.0	12.82	3
naive_mean_2	0.0	12.86	3
naive_mean_3	0.0	12.87	3
naive_mean_48	0.0	12.87	3
naive_last_day	0.0	12.82	3
naive_last_week	0.0	12.86	3

Table 4.10: Full evaluation radio 1.

Table 4.11 presents the evaluation results for radio 2. This dataset contains more instances of the positive class compared to radio 1. In this case, the machine learning models outperform the benchmarks in terms of underestimations. The *Naive_mean_48* model achieves the highest energy savings, although this is accompanied by the maximum number of underestimations. The second-best performers in terms of energy savings are two machine learning models: the TFT regressor optimized for MSE and the KNR model optimized for MSE, both achieving savings of 13.02%. However, similar to `naive_mean_48`, neither of these models successfully predicted any positive class instances. The model achieving the highest energy savings while also identifying some positive instances is the HybridLSTM optimized for

MSE, which attained 12.98% savings and correctly predicted 1 out of 30 positive instances. The second-best in this regard is the Random Forest optimized for MSE, which achieved 12.96% savings with 28 out of 30 underestimations.

In terms of underestimations, the TFT classifier achieved the best performance with only 11 underestimations, correctly predicting approximately 60% of the positive class instances. The FFNN classifier had 15 underestimations and achieved 12.03% energy savings, compared to 9.81% for the TFT classifier.

model	F1	Energy Saving	# Underestimations
rf_c	0.21	11.73	18
rf_F1	0.11	12.94	28
rf_r	0.11	12.96	28
xgb_c	0.21	11.47	16
xgb_F1	0.18	12.83	26
xgb_r	0.05	12.89	29
knc_c	0.05	12.91	29
knr_F1	0.08	12.72	28
knr_r	0.0	13.02	30
svc_c	0.18	11.63	19
svr_F1	0.15	12.87	27
svr_r	0.0	12.99	30
ffnn_c	**0.31**	12.03	15
ffnn_F1	0.05	12.86	29
ffnn_r	0.06	12.95	29
lstm_c	0.19	12.41	23
lstm_F1	0.18	12.84	26
lstm_r	0.06	12.98	29
cnn_c	0.14	12.61	26
cnn_F1	0.05	12.90	29
cnn_r	0.11	12.63	27
tft_c	0.16	9.81	**11**
tft_F1	0.0	12.85	30
tft_r	0.0	13.02	30
arima_r	0.0	12.80	30
naive_mean_1	0.1	12.57	27
naive_mean_2	0.07	12.61	28
naive_mean_3	0.11	12.67	27
naive_mean_48	0.0	**13.04**	30
naive_last_day	0.07	12.58	28
naive_last_week	0.04	12.75	29

Table 4.11: Full evaluation radio 2.

Radio 3, similar to radio 1, contains few instances of the positive class-two in total. None of the ML models or benchmarks successfully predict any of these positive

instances. Notably, 17 out of 24 ML models achieve the maximum possible energy saving of 12.75%, as do several of the benchmarks. The results is provided in Table 4.12.

model	F1	Energy Saving	# Underestimations
rf_c	0.0	12.70	2
rf_F1	0.0	12.75	2
rf_r	0.0	12.75	2
xgb_c	0.0	12.70	2
xgb_F1	0.0	12.75	2
xgb_r	0.0	12.75	2
knc_c	0.0	12.75	2
knr_F1	0.0	12.73	2
knr_r	0.0	12.75	2
svc_c	0.0	12.67	2
svr_F1	0.0	12.75	2
svr_r	0.0	12.75	2
ffnn_c	0.0	12.70	2
ffnn_F1	0.0	12.75	2
ffnn_r	0.0	12.75	2
lstm_c	0.0	12.75	2
lstm_F1	0.0	12.75	2
lstm_r	0.0	12.75	2
cnn_c	0.0	12.75	2
cnn_F1	0.0	12.75	2
cnn_r	0.0	12.75	2
tft_c	0.0	12.67	2
tft_F1	0.0	12.75	2
tft_r	0.0	12.75	2
arima_r	0.0	12.75	2
naive_mean_1	0.0	12.72	2
naive_mean_2	0.0	12.75	2
naive_mean_3	0.0	12.75	2
naive_mean_48	0.0	12.75	2
naive_last_day	0.0	12.73	2
naive_last_week	0.0	12.75	2

Table 4.12: Full evaluation radio 3.

Table 4.13 presents the results for radio 4, which contains zero instances of the positive class. Consequently, the voltage can always be set to the lower level. All benchmark models and ML models, except two configurations, correctly maintain the voltage at the lower level throughout. This is evidenced by their achieving the maximum possible energy savings for radio 4, as shown in Table 4.13.

model	F1	Energy Saving	# Underestimations
rf_c	0.0	13.01	0
rf_F1	0.0	13.01	0
rf_r	0.0	13.01	0
xgb_c	0.0	13.01	0
xgb_F1	0.0	13.01	0
xgb_r	0.0	13.01	0
knc_c	0.0	13.01	0
knr_F1	0.0	12.99	0
knr_r	0.0	13.01	0
svc_c	0.0	13.01	0
svr_F1	0.0	13.01	0
svr_r	0.0	13.01	0
ffnn_c	0.0	13.01	0
ffnn_F1	0.0	13.01	0
ffnn_r	0.0	13.01	0
lstm_c	0.0	13.01	0
lstm_F1	0.0	13.01	0
lstm_r	0.0	13.01	0
cnn_c	0.0	13.01	0
cnn_F1	0.0	13.01	0
cnn_r	0.0	13.01	0
tft_c	0.0	12.92	0
tft_F1	0.0	13.01	0
tft_r	0.0	13.01	0
arima_r	0.0	13.01	0
naive_mean_1	0.0	13.01	0
naive_mean_2	0.0	13.01	0
naive_mean_3	0.0	13.01	0
naive_mean_48	0.0	13.01	0
naive_last_day	0.0	13.01	0
naive_last_week	0.0	13.01	0

Table 4.13: Full evaluation radio 4.

Lastly, Table 4.14 presents the results for radio 5, which has a similar proportion of positive instances as radio 2 (see Table 4.11). Several machine learning models, including the RF and XGB optimized for MSE, achieve the maximum possible energy savings of 12.42% while also correctly predicting some of the higher-utilization instances. The FFNN classifier demonstrates the best performance in terms of underestimations, closely followed by the TFT classifier. However, the FFNN classifier outperforms the TFT classifier across all evaluation metrics.

model	F1	Energy Saving	# Underestimations
rf_c	**0.47**	12.12	19
rf_F1	0.21	12.37	29
rf_r	0.26	12.42	28
xgb_c	0.14	12.02	29
xgb_F1	0.24	12.33	28
xgb_r	0.35	12.42	26
knc_c	0.16	12.38	30
knr_F1	0.21	12.25	28
knr_r	0.0	12.42	33
svc_c	0.15	12.12	29
svr_F1	0.43	12.35	23
svr_r	0.06	12.40	32
ffnn_c	0.46	11.73	**14**
ffnn_F1	0.11	12.40	31
ffnn_r	0.0	12.42	33
lstm_c	0.31	12.32	26
lstm_F1	0.24	12.33	28
lstm_r	0.0	12.42	33
cnn_c	0.27	12.19	26
cnn_F1	0.16	12.38	30
cnn_r	0.0	12.42	33
tft_c	0.33	11.40	15
tft_F1	0.31	12.21	25
tft_r	0.0	12.40	33
arima_r	0.19	12.33	29
naive_mean_1	0.3	11.97	23
naive_mean_2	0.36	12.18	23
naive_mean_3	0.31	12.23	25
naive_mean_48	0.0	12.42	33
naive_last_day	0.19	11.97	26
naive_last_week	0.03	11.94	32

Table 4.14: Full evaluation radio 5.

4.4 Generalizability Evaluation

To evaluate the generalizability of the classification models, a synthetic dataset was constructed as described in Section 4.4. The trained classification models were then applied to this dataset to assess whether their performance holds in a scenario with a higher proportion of class 1 instances. This in turn creates a more balanced data set amongst the two classes. This test serves to examine the resilience of the models when exposed to data distributions that differ from the original training and testing conditions. The results are shown in Table 4.15, which summarizes model performance on the classification task. The dataset contained a total of 747 instances

belonging to the positive class, meaning there were 747 potential underestimations.

Among all models, the TFT had the lowest number of underestimations, with only 202 out of 747. The FFNN followed with 250 underestimations. Although the TFT had the fewest restarts, it also resulted in the lowest energy saving. The XGB model achieved the highest energy saving, but this was due to it consistently predicting the lower voltage level. Hence, it reached the optimal energy saving at 10.97% but without ever adjusting for any peaks.

In terms of F1 score, the two naive mean models with the lowest mean values (*naive_mean_2* and *naive_mean_3*) performed best, each achieving a score of 0.66. These were closely followed by the HybridLSTM model with a score of 0.643, and the FFNN with 0.633. Overall, the neural network-based models outperformed the classical statistical models, with the exception of the HybridCNN. Based on both F1 score and number of underestimations, the neural networks, together with *naive_mean_2* and *naive_mean_3*, occupy the top positions, indicating stronger overall performance on the task.

Model	**F1 Score**	**Energy Saving (%)**	**# Underestimations**
RF	0.46	10.81	509
XGB	0.00	**10.97**	747
KNC	0.30	10.79	608
SVC	0.02	10.66	736
FFNN	0.63	9.90	250
LSTM	0.64	10.53	339
CNN	0.26	10.94	634
TFT	0.56	8.88	**202**
ARIMA	0.04	10.94	732
naive_mean_1	0.65	10.1	261
naive_mean_2	**0.66**	10.22	269
naive_mean_3	**0.66**	10.21	267
naive_mean_48	0.17	10.09	643
naive_last_day	0.52	9.94	353
naive_last_week	0.38	10.32	515

Table 4.15: F1 score, energy saved, and number of underestimations for each model at threshold 0.5.

The synthetic data generates a higher number of instances in the positive class, which also leads to an increased risk of underestimation. Fig. 4.8 illustrates the performance of all naive mean benchmarks on this data, providing insight into how the choice of window size affects outcomes. Specifically, a larger window size tends to result in more underestimations but also yields greater energy savings. Conversely, a smaller lag reduces the number of underestimations but at the cost of lower energy savings. The synthetic data gave more nuance to the effect of window for benchmark compared to that in Fig 4.7 where a smaller window increase yielded a greater change.

This is seen by Fig. 4.8 having more color nuances than Fig. 4.7.

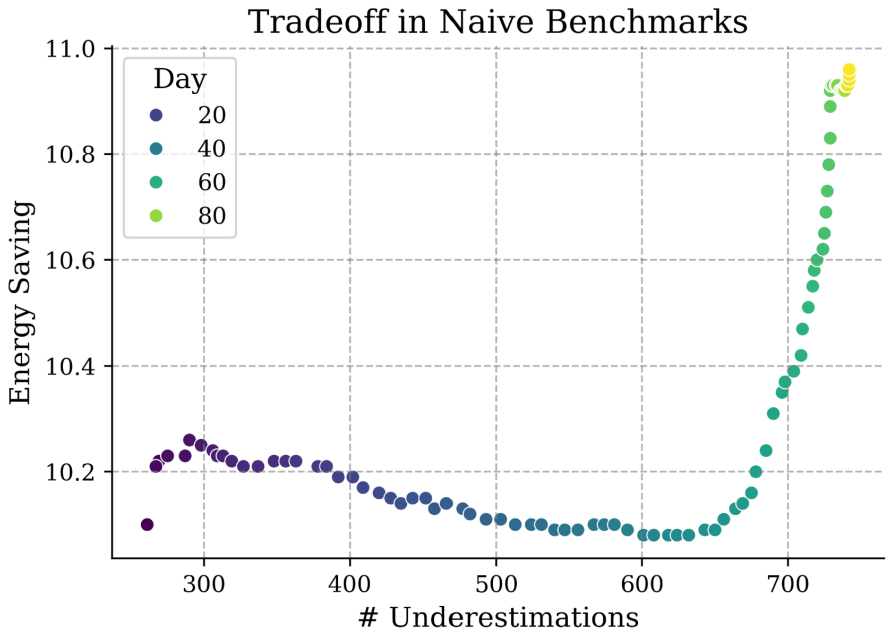


Figure 4.8: Trade-off between energy savings and underestimations across different window sizes in baseline models on synthetic data.

4.5 Exploring the Trade-Off Between Energy Savings and Underestimations

As described in Section 3.9.2, probability predictions from the classification models were used to illustrate the trade-off between energy savings and underestimations. This analysis can serve as a basis for adjusting the decision threshold in deployment, enabling the system to prioritize fewer restarts at the potential expense of reduced energy savings. Fig. 4.9 shows the resulting curve of each model when shifting the classification threshold between 0.01 and 0.99. The threshold is indicated by the color of the dot. The scale of the threshold is shown in the bottom of Fig. 4.9. This thresholding exercise shows the exact tradeoff between the two metrics for each model. It indicates that some models are more sensitive to threshold tuning; for example, the SVC model exhibits a substantial increase in both restarts and energy savings even with small shifts in the threshold value. The distinct behavior of the KNC model is explored in more detail in Section 5. The FFNN model demonstrates the steepest improvement curve with low numbers of restarts, achieving only 5 underestimations at 10% energy saving. FFNN was followed by the HybridLSTM and RF models. Both models are able to achieve approximately 10% energy savings with no more than 10 restarts in total.

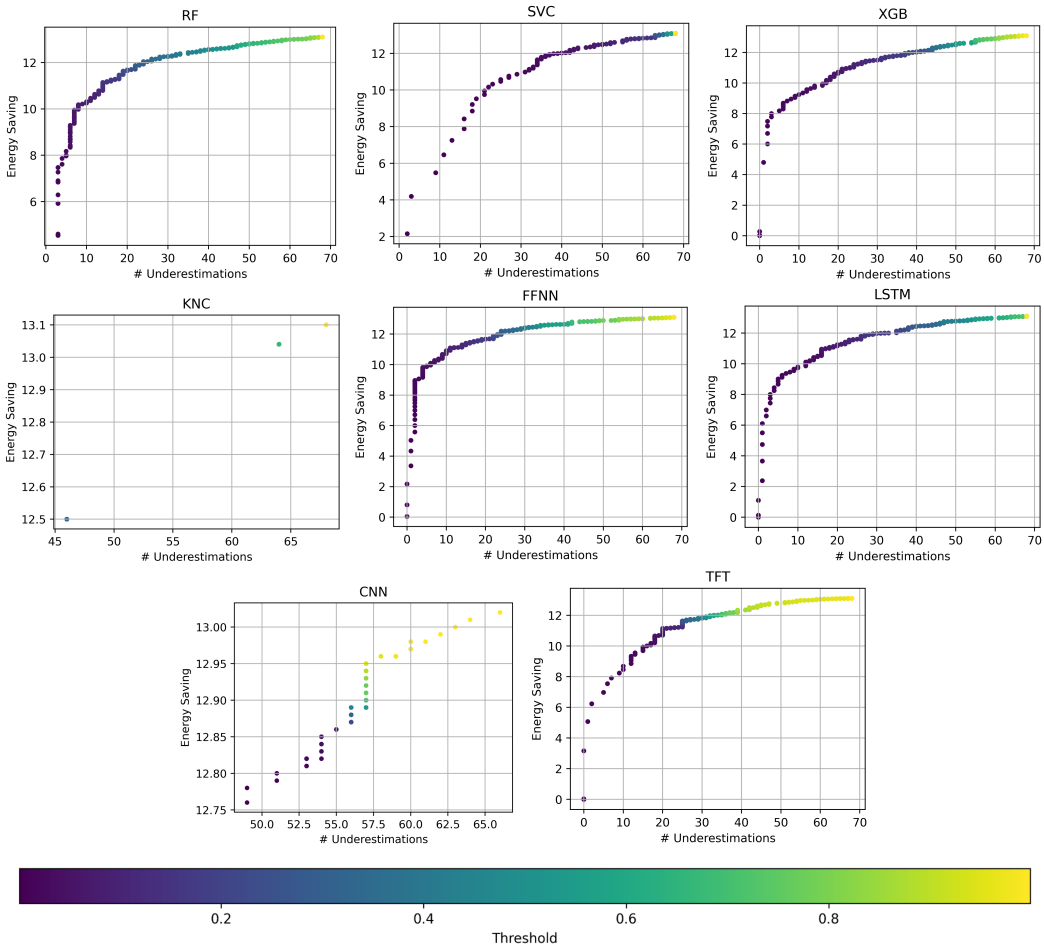


Figure 4.9: Model performance trade-offs under varying thresholds. Color encodes threshold level.

4.6 SHAP-Based Interpretability of Top Models

As explained in Section 2.2, machine learning models often function as black boxes, making it challenging to interpret the reasoning behind their predictions. To address this, an additional analysis was conducted to identify which features contributed the most to the predictions of two selected models. The best performing neural network and statistical model in terms of number of restarts were chosen for this analysis, and SHAP was used to interpret their behavior.

The statistical model with the lowest number of restarts was the RF classifier, as shown in Table 4.7. Therefore, it was selected for further explainability analysis. For the neural network models, the FFNN classifier was chosen. Although the TFT slightly outperformed the FFNN in terms of restarts, its energy-saving performance was significantly lower as can be seen in Table 4.7. As a result, the FFNN was considered a more balanced choice.

Fig. 4.10 presents the SHAP summary plot for the RF classifier, illustrating the contribution of each feature to the prediction of class 1. The feature with the greatest

impact is *PRB-U* at the current time step (*t*–0). Lower values of this feature are associated with negative SHAP values, indicating a reduced likelihood of predicting class 1, whereas higher values contribute positively, increasing the probability of a high-level classification.

The subsequent important features are also associated with the current time step, suggesting that the RF model primarily rely on recent input values. Among the two sequential features, *PRB-U* appear more frequently near the top of the importance ranking, indicating a stronger influence on the models predictions compared to *meanPowerConsumption*. In contrast, static features such as hour, minute, and weekday show low SHAP values, suggesting that the RF model did not derive significant benefit from these inputs and instead focuses on short-term temporal patterns.

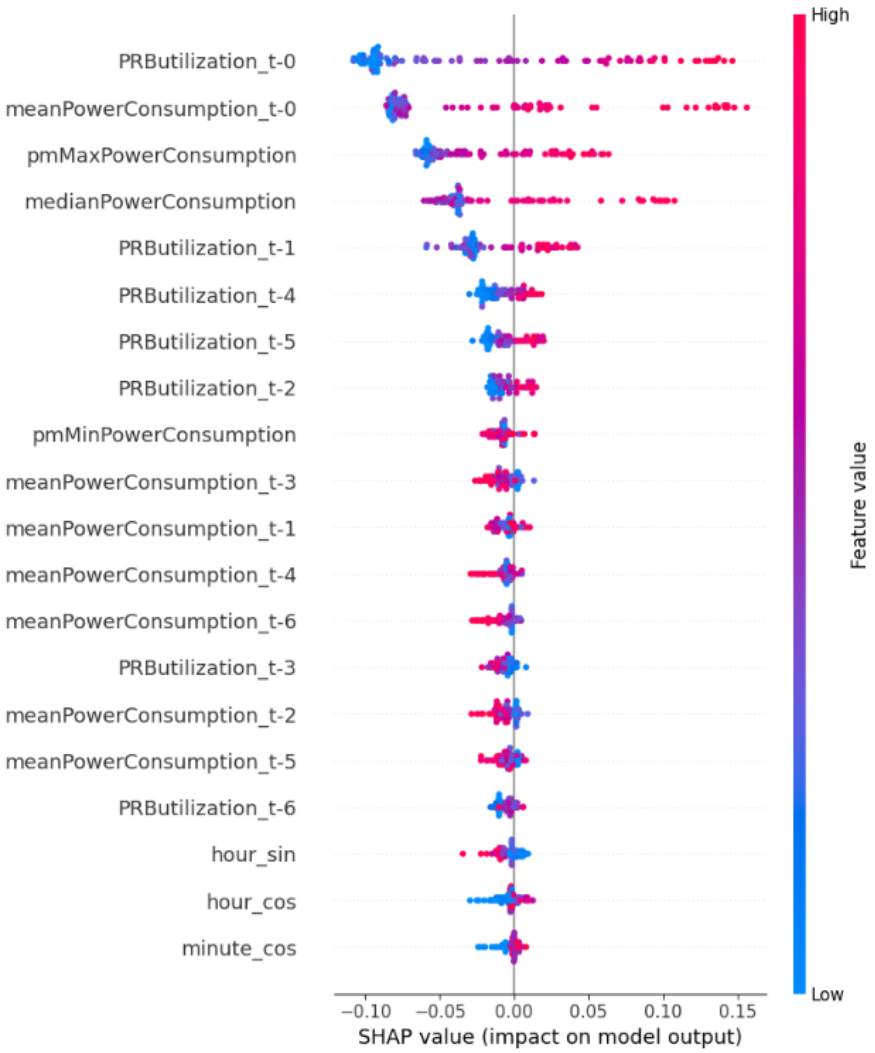


Figure 4.10: RF SHAP values for class 1.

For the FFNN model, a SHAP summary plot of class 1 predictions is shown in Fig. 4.11. The plot reveals a strong reliance on *PRB-U* values at the current time step (*t*–0). Low values of this feature negatively impact the model's output (i.e.,

the predicted probability of class 1), whereas high values contribute positively. The remaining features have much smaller SHAP values. Some features exhibit an opposite pattern to *PRB-U* at t–0. For example, high values of mean power consumption at t–3 and *PRB-U* at t–1 are associated with negative SHAP values, while low values have a positive effect.

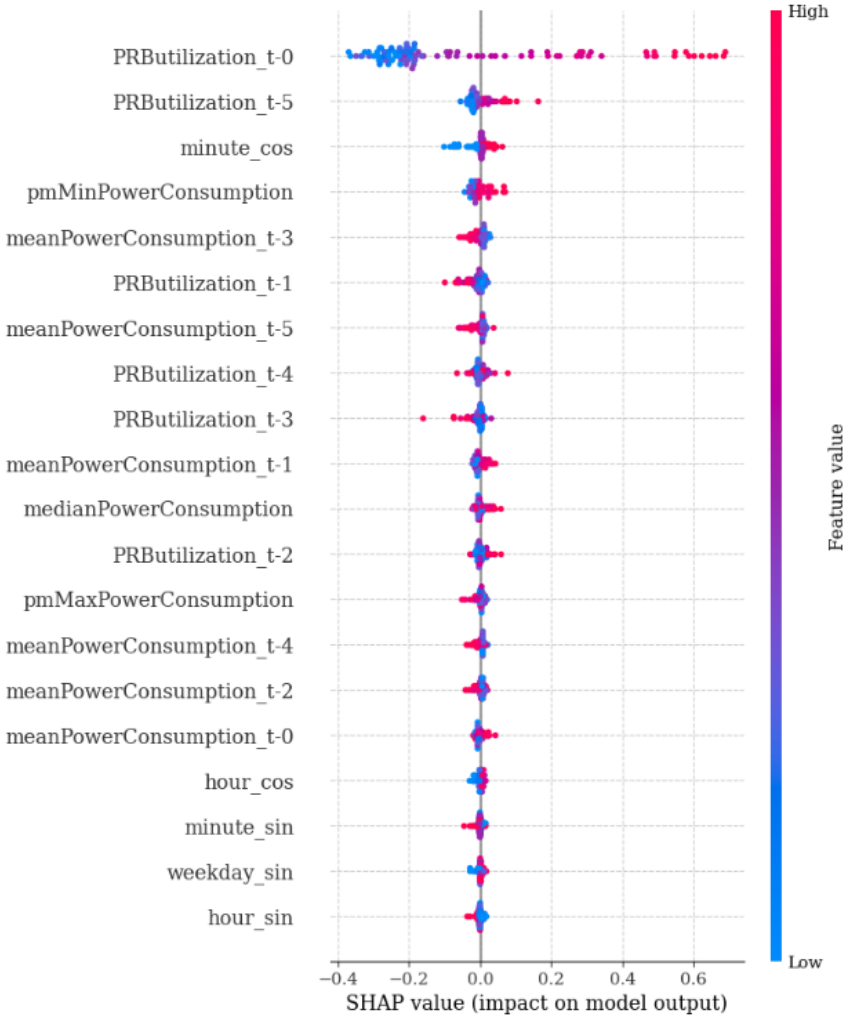
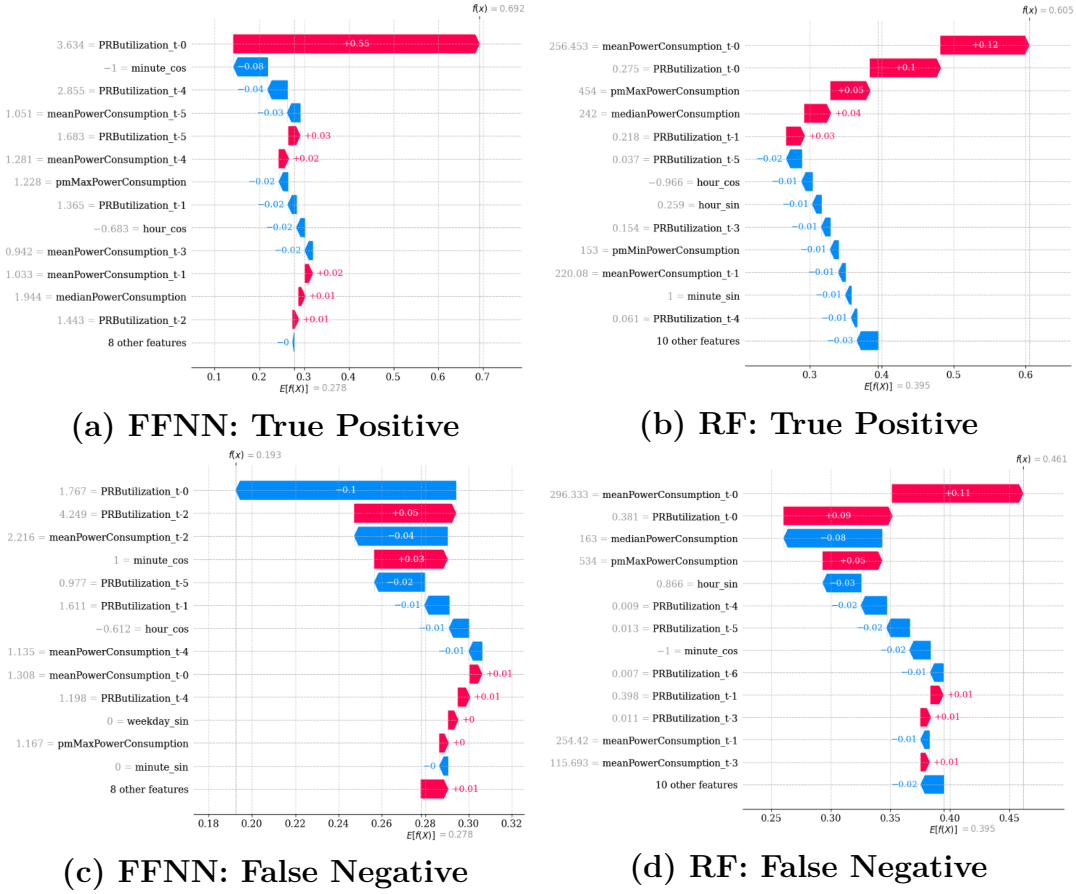


Figure 4.11: FFNN SHAP values for class 1.

SHAP explanations were also extracted for individual predictions. Fig. 4.12 shows SHAP values for positive predictions from both the RF and FFNN models. The subfigures (a) and (b) illustrate an instance that was correctly classified as class 1, while (c) and (d) show a positive instance that was incorrectly classified as class 0 (i.e., a false negative). In both cases, the FFNN model places most emphasis on PRB-U at the current timestep, whereas the RF model focuses primarily on mean power consumption at the current timestep. For the false negative, the FFNN assigns only a 19% probability to the positive class, while the RF model is more confident, assigning a probability of 46%.

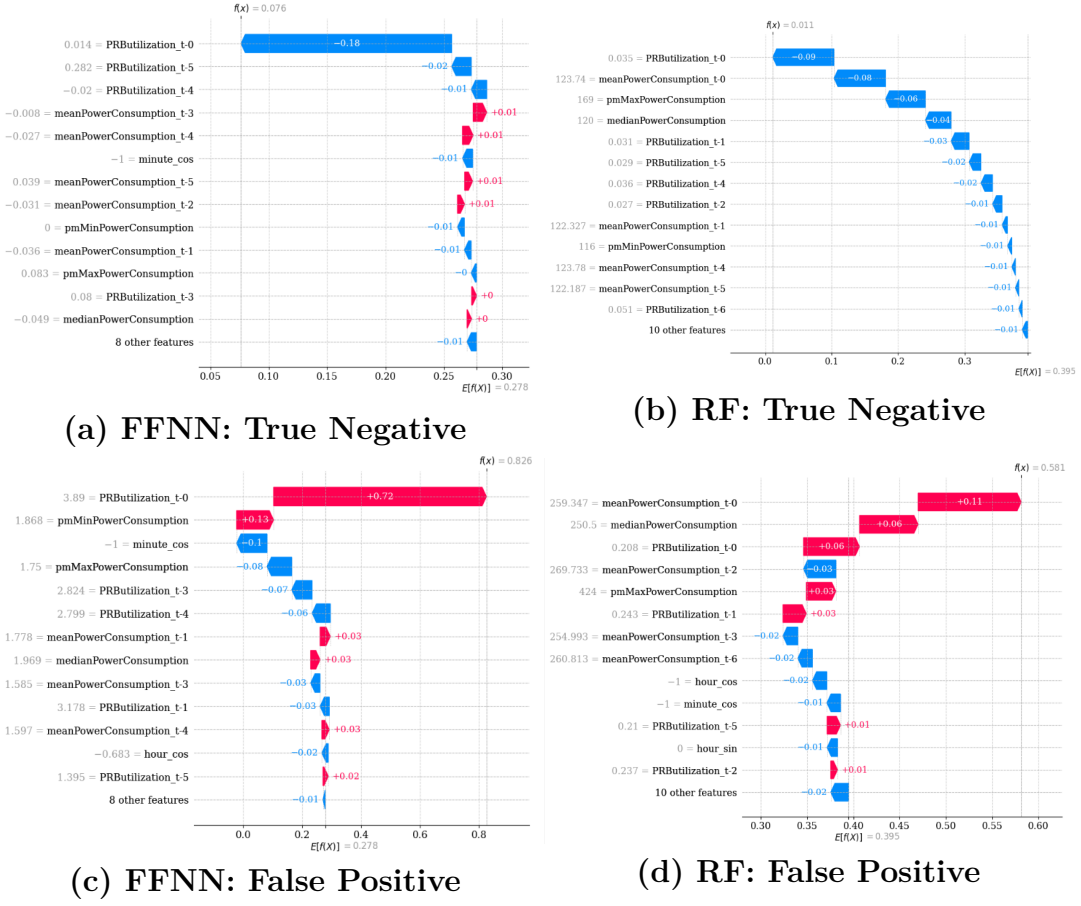


(a) FFNN: True Positive **(b) RF: True Positive**

(c) FFNN: False Negative **(d) RF: False Negative**

Figure 4.12: SHAP explanations for positive samples. True positives and false negatives are shown for both models.

Fig. 4.13 shows SHAP explanations for the negative class. The subfigures (a) and (b) illustrate correctly classified negative instances, while (c) and (d) show instances that were mistakenly classified as positive (i.e., false positives). PRB-U at the current timestep remains the most influential feature for the FFNN in both cases. For the RF model, PRB-U at the current timestep is the most important feature in the true negative case, whereas mean power consumption at the current timestep dominates in the false positive case. For the false positive, the FFNN assigns a high probability of 83% to the positive class, while the RF model assigns a lower probability of 58%.



(a) FFNN: True Negative **(b) RF: True Negative**

(c) FFNN: False Positive **(d) RF: False Positive**

Figure 4.13: SHAP explanations for negative samples. True negatives and false positives are shown for both models.

4.7 Model Energy Consumption

Energy consumption was evaluated for the two top-performing models, as motivated in Section 4.6. As previously laid out in Section 3.9.1, the `codecarbon` library was used to measure the energy consumed during inference, and the results are presented in Table 4.16. As can be seen in the table, the ratio of the consumed energy when predicting is very small in relation to the saved energy, less than 10^{-9} for both models.

Model	Classification R_E
RF	$1.15 * 10^{-11}$
FFNN	$7.33 * 10^{-10}$

Table 4.16: Ratio of energy used for two ML models in regards to saved energy.

5

Discussion

Sections 5.1, 5.2 and 5.3 present the results with each research question in focus. Section 5.5, discusses the limitations of the findings of this thesis.

5.1 RQ1: Viability of ML-based solutions

The first question examined in this thesis is whether there is use for ML methods in creating a dynamic voltage setting in radios to save energy. The results presented in Section 4 demonstrated that there is potential to reduce energy consumption by a total of 12.83% across the five radios in the test set ranging from 2025-03-08 to 2025-03-22. On the other hand, this comes with the challenge of not underestimating the voltage need which would result in connectivity issues for users.

The baseline models, as shown in Table 4.7, achieved energy savings of 12.6% or more. However, they also produced at least 55 underestimations out of 68 possible high-demand instances, correctly identifying only about 20% of the positive cases. This highlights a key limitation: while effective in reducing energy usage, these models struggle to detect periods of high demand. A notable observation is that averaging over a larger time window results in a less dynamic model that misses more positive instances. This outcome is expected, given the unbalanced nature of the dataset, which is skewed toward the negative class. The pattern is evident in Fig. 4.7, where lower lag values resulted in fewer underestimations, but also reduced energy savings.

Section 4.3 highlighted differences in both potential performance and actual model performance across individual radio units. This raises the question of whether certain radio units benefit more from machine learning models than others. Table 4.9 presented the optimal energy savings and worst-case number of restarts per unit, with energy savings ranging from 12.42% to 13.04% and restarts ranging from 0 to 33. Detailed model performance on each individual radio was provided in Tables 4.10 through 4.14. Radios 1, 3, and 4 contain very few instances in the higher PRB-U range, whereas radios 2 and 5 have a greater concentration of such instances. It is likely that other units in the broader population display even more extreme behaviors, if not consistently, then at least during certain time intervals.

The evaluations on individual radios showed that the benchmark models were highly competitive with the ML models in terms of energy savings for radios such as 1, 3,

and 4, which contained few instances of the positive class. Conversely, Tables 4.11 and 4.14 showed that radios 2 and 5 benefited more from ML techniques, achieving higher energy savings while also minimizing the number of underestimations.

Hence, machine learning methods may be beneficial for some, but not all, radio units when considering a two-voltage level scenario with the lower level set at 50%. One option would be to employ simpler statistical models, such as the implemented benchmarks, for radios like 1, 3, and 4, or to operate these radios at a fixed low voltage level. However, this approach raises concerns regarding potential changes in the operational environment. Is it reasonable to assume that past behavior will remain consistent throughout the entire lifetime of the unit? This is a strong assumption, and if violated, could lead to degraded performance or suboptimal outcomes for customers if the voltage requirements of a unit change over time. An alternative strategy would be to apply basic benchmarking models only to radios that exhibit a consistently low voltage level.

Although machine learning was found to be relevant for the dynamic voltage scenario in radios, an important question prior to implementation is how many underestimations are acceptable for a given customer and radio unit. Section 4.5 showed how decision threshold for the classification models can be shifted to achieve different positions within the energy save-under estimation trade off. This demonstrated an easy way to utilize one algorithm, but allow for unit specific preferences and is thus one way to customize an algorithm per user or environment for each specific radio.

The energy consumed for making predictions, as shown in Table 4.16, was less than 10^{-9} of the saved energy for the two selected top-performing models. This very low ratio indicates that the energy savings achieved far outweigh the computational energy cost of running the models, when considering the net effect in kWh.

It should also be noted that machine learning models, particularly deep learning models, tend to perform significantly better when trained on larger datasets. The dataset used for training in this thesis consists of approximately 18 000 instances, which can be considered medium-sized in the context of structured time-series data. However, the dataset is highly imbalanced, resulting in few instances of the positive class. This scarcity may have constrained the models ability to accurately learn patterns associated with the minority class. Consequently, the ML-based models used in this work might achieve better performance if trained on a larger and more balanced dataset.

In addition, the models were trained on aggregated data from all five radios. It is possible that patterns learned across radios may not generalize well to individual units. With only approximately 3,600 training instances per radio, the available data was insufficient to successfully train radio-specific models and enable a meaningful comparison of their performance. However, given the differing characteristics of the individual radios, there is reason to believe that radio-specific models could potentially achieve improved performance. The baseline models calculated mean and other statistical values per individual radio during both training and prediction; therefore, their performance would not change in a radio-specific setup.

In conclusion, implementing machine learning algorithms for dynamic voltage control in radios introduces both the cost and risk of underestimating voltage requirements. The trade-off between energy savings and the potential impact of underestimations must be carefully considered. Machine learning strategies were shown to outperform simpler approaches in cases where the positive class was more prevalent. With more data, the machine learning models might demonstrate even stronger performance, as it remains unclear how well patterns generalize across radios, and the number of radio-specific instances in the training data is relatively low.

5.2 RQ2: Classification vs. Regression

Section 4.2 presented the evaluation of all machine learning models on the test data across three configurations: (1) classification, (2) regression optimized using the F1 score, and (3) regression optimized using MSE. Each configuration demonstrated a trade-off between the number of underestimations and the amount of energy saved. Although the regression models achieved higher energy savings, this came at the cost of an increased number of underestimations. Conversely, the classification configuration resulted in fewer underestimations but lower energy savings. Furthermore, when comparing the two regression approaches, tuning models based on the F1 score resulted in fewer underestimations than tuning on MSE, though with slightly lower energy savings. It is important to note that the difference in energy savings between these configurations was minor, as illustrated in Fig. 4.6. An additional advantage of the classification approach is that it provides access to class probabilities, which enables threshold adjustment between class 0 and class 1, as demonstrated in Section 4.5.

An important factor contributing to the strong performance of the classification task in minimizing underestimations was likely the use of *BCEWithLogitsLoss* as internal loss function, which helped address the class imbalance in the data. A potential direction for future research is to explore analogous techniques for the regression tasks. Specifically, methods for minimizing underestimations or incorporating class-weighting strategies similar to those used in classification. This way, specific penalizing could be given to models missing sudden, uncommon spikes in the data.

Furthermore, lag values across the different task formulations were analyzed using SHAP values to compare how models for the various tasks utilized temporal information, as shown in Fig. 4.1. The visualization reveals distinct patterns across models. For some models, such as the RF, the pattern is particularly clear: in the classification setting, the model relies on shorter lags compared to the regression setting. In general, models were shown to rarely prefer longer lag values in regression compared to classification. The only model exhibiting this behavior is the SVC/R, where high SHAP values are associated with longer lag values in the classification task, while longer lags are less beneficial in the regression task optimized for MSE. For all other models, either shorter or similar lag lengths are preferred in the classification task compared to the regression tasks. Given that models deployed in radios must be both energy- and memory-efficient, the classification approach may be preferred

from an inference perspective, as it enables models with fewer parameters due to the reduced number of input features.

Having discussed model performance and preferred lag configurations within each setting, it is now possible to summarize the full range of characteristics for each task formulation. An overview is provided in Table 5.1, where B denotes the benchmark models, C the classification setup, R(F1) the regression models selected based on F1 score, and R(MSE) the regression models selected based on MSE. The table consolidates the key findings of this section, emphasizing that the optimal task formulation depends on the trade-off between energy savings and the number of underestimations, and that the preference of lag differs between tasks. Among the models evaluated, classification was the most favorable for minimizing underestimations. Regression models tuned on MSE resulted in the highest number of underestimations, while regression models selected based on the F1 score yielded slightly better performance, although still close to the performance of the ones selected on MSE. In contrast, when considering energy savings, the order was reversed and it was still a small difference between the two regression cases.

When considering inference cost, the classification task demonstrated the ability to perform well with shorter lag values, thereby requiring fewer input features and resulting in a lower number of operations during prediction. Classification was followed by regression selected on MSE and lastly regression selected on F1 score, as seen in Table 4.1.

As for the benchmarking models, they had similar energy saving and underestimations as the regression versions though a smaller lag as the best performing naive benchmarks were selected and those had a lag of 1, 2, 3 and 48.

Relative Performance	**B**	**C**	**R(F1)**	**R(MSE)**
Energy Saving	Favorable	Intermediate	Favorable	Favorable
Underestimation Rate	Inferior	Favorable	Inferior	Inferior
Lag value	Favorable	Favorable	Inferior	Intermediate

Table 5.1: Comparison of model performance across different evaluation aspects.

Lastly, in the classification task the threshold for predicting class 0 or 1 can easily be adjusted to account for the trade-off between under estimations and saved energy. Consequently, classification can be recommended for scenarios where reduced inference cost and minimization of underestimations are prioritized. On the other hand, if small changes in energy savings or closer alignment with the PRB-U curve is of greater importance, the MSE-based regression approach may be preferred.

5.3 RQ3: Model Selection

This section aims to address the question of which models are most optimal for dynamic voltage control in radios.

5.3.1 Architectural Preferences

A key aspect of this thesis was allowing important hyperparameters to vary within the Optuna tuning loop. One such variable that remained dynamic throughout the tuning process was the window size. Table 4.1 presents the lag values selected for each model by Optuna, while Fig. 4.2 displays the corresponding SHAP values of lag in relation to model performance. Including lag as a tunable hyperparameter enabled model-specific feature reduction, which is important in time-series forecasting, as retaining only the most informative features can improve performance [80]. In addition to temporal features, some architectural choices were also included in the hyperparameter tuning of the neural network models. For the FFNN, two such components were whether to use residual connections and whether to apply a sigmoid activation function in the regression setting. The box plot in Fig. 4.2 shows that including residual connections led to slightly worse or comparable performance across all three task formulations. However, Fig. 4.4 reveals that Optuna tended to favor trials with residual connections in both regression settings. This suggests that residual connections may contribute to increased stability in intermediate representations, even if they do not improve final performance.

Among the tuned FFNN models (Table 4.4), both regression models included residual connections, whereas the classifier did not. Interestingly, the deepest FFNN model—the classifier with three layers—did not use residual connections, while the shallower regression models, each with only one layer, did. One possible explanation is that, in the regression case, the target values are very small, often zero or below 0.2. This may make the vanishing gradient problem more prominent than in the classification setting, where the labels are binary and cross-entropy loss was used with weighting for the positive class. The regression models were trained using the MSE loss function, and the small target values can lead to very small loss values. This is evident in Fig. 3.15, where loss values below 10^{-3} were reached during training on the regression task. This may explain why the regression task benefits more from including residual connections, even though the networks are shallow. Using an alternative loss function in the regression setting could potentially mitigate this issue.

The results for the sigmoid activation parameter were more definitive. Fig. 4.4 shows that performance was worse when sigmoid activation was used in both regression settings, and Table 4.3 indicates that Optuna increasingly avoided trials with sigmoid activation. Consistently, Table 4.4 confirms that none of the tuned regression FFNN models included a sigmoid activation.

Dropout, as shown in Table 4.4, was included in each of the optimized FFNN models. This suggests that dropout can still provide regularization benefits even when batch normalization is applied, which contrasts with claims in Section 2.4.5 and the original BatchNorm paper [40], where it was suggested that batch normalization might reduce or eliminate the need for dropout. One possible explanation is that these techniques address different aspects of the training process: batch normalization stabilizes the distribution of activations and allows for higher learning rates, while dropout reduces co-adaptation among neurons by forcing the network

to learn redundant representations, as discussed in Section 2.4.5. In combination, they may offer complementary regularization effects, particularly in the context of the structured time-series data used in this thesis.

However, it is also possible that the inclusion of dropout does not significantly impact performance. To verify this, a similar analysis should be conducted for dropout, as was performed for the residual connections and sigmoid activation parameters.

These results suggest that the FFNN model used for regression should not employ a sigmoid activation for this task. The impact of including residual connections is less conclusive, although they were included for both regression tasks. Further analysis is needed to better understand why Optuna favored trials with residual connections in the regression settings. One hypothesis proposed in this thesis is that this may be related to the low target values. Dropout was included for all settings, suggesting that it might provide additional regularization effects despite the presence of batch normalization after each layer in the FFNN architecture.

Some hyperparameters for the HybridLSTM were presented in Section 4.1.2. Fig. 4.1 showed the effect of lag on performance for all models. For the HybridLSTM, it seems lag values up to a certain limit, around 40 for the classification and F1 score based regression task, and around 20 for the MSE based regression task, improves the model performance. Longer lag values, however, seem to worsen the performance. This suggests that even though the HybridLSTM inherently can account for time aspect in its input data, the performance can be further improved by providing additional temporal information in the input features, to a certain point.

The dropout values for the optimal HybridLSTM model configurations, presented in Table 4.5, exhibit varying preferences across task formulations. For the regression tasks, dropout appears to be beneficial-it is included in both the F1-optimized and MSE-optimized models. Notably, the F1-based model adopts the maximum permitted value of 0.5, while the MSE-based model uses a more moderate value of 0.1. In contrast, dropout is excluded entirely from the optimal configuration for the classification task. These results suggest that the effectiveness and optimal magnitude of dropout could be task-dependent. Stronger regularization may be advantageous in regression scenarios, particularly when addressing class imbalance, whereas in classification, it may not contribute meaningfully or could even be detrimental. This pattern differs somewhat from the results observed for the FFNN models, where dropout was included across all configurations. One possible explanation is that the different model architectures have different regularization requirements or sensitivities. Nevertheless, due to the conflicting patterns observed, a more comprehensive analysis of dropout values across all trials would be necessary to draw firmer conclusions about its overall impact.

In summary, the number of temporal features along with several architectural components were evaluated for their impact on performance. For the FFNN model, the inclusion of residual connections was favored by the hyperparameter tuner in the regression settings; however, the resulting performance over the tuning trials was slightly worse or comparable to models without residual connections. The use of a sigmoid activation function in the regression setting was consistently detrimental, de-

spite its theoretical ability to constrain outputs within an acceptable range. Dropout was included across all optimized FFNN configurations, suggesting that it provided additional regularization benefits even in the presence of batch normalization.

For the HybridLSTM, incorporating explicit time features (through lag selection) proved beneficial, even though the model is inherently designed to capture temporal dependencies. The use of dropout showed more task-dependent behavior: it was included in the optimized regression models, but not in the classification model, indicating that its effectiveness may vary with model architecture and task formulation.

5.3.2 Model Performance

As for evaluation, the relatively small differences in energy savings become evident when comparing the top and bottom performers in the full evaluation, presented in Table 4.7. The KNR and TFT models, when selected based on MSE, achieved the highest energy savings at 12.82%, closely followed by the RF model, also selected on MSE, with 12.81%. The lowest energy savings were recorded by the TFT classifier, which achieved 11.78%. This results in a total range of only 1.03 percentage points between the best and worst models, corresponding to a difference of approximately 8% relative to the top-performing model.

Despite being the lowest performer in terms of energy savings, the TFT classifier achieved the best performance in minimizing underestimations. This clearly illustrates the trade-off between energy savings and prediction accuracy during high-load periods. The range of underestimations shown in Table 4.7 spans from 31 to 68, representing more than a 100% increase from the best to the worst performer in this metric. A key question that emerges is: What constitutes an acceptable balance between potential performance loss due to underestimations and the marginal gain of saving an additional percentage point of energy?

As for CNN it did not outperform LSTM on the test data, contrasting to performance noted by other studies [81]. Neither did the TFT even though it should be an improved variant of the LSTM. Since deep models usually outperform more shallow ones with more data, it is possible that the results would be different if the models had been trained on more data.

Prior research has demonstrated that tree-based models can outperform deep learning approaches in certain structured data tasks [54]. In the context of this thesis, as shown in Table 4.8, the RF model outperformed the other models in terms of MSE and MAE in the version optimized for MSE. The version optimized for F1 score also performed strongly, ranking second, despite competing against models selected using an MSE objective.

However, with respect to the two key metrics of this thesis-energy savings and number underestimations-the RF classifier exhibited more underestimations than most of the neural network models. Only the HybridLSTM produced a higher number of underestimations than the RF classifier in the classification setting. In the F1-optimized regression setting, however, the RF outperformed both the HybridCNN

and the FFNN in terms of energy savings. Fig. 5.1 highlights the RF model's comparative performance.

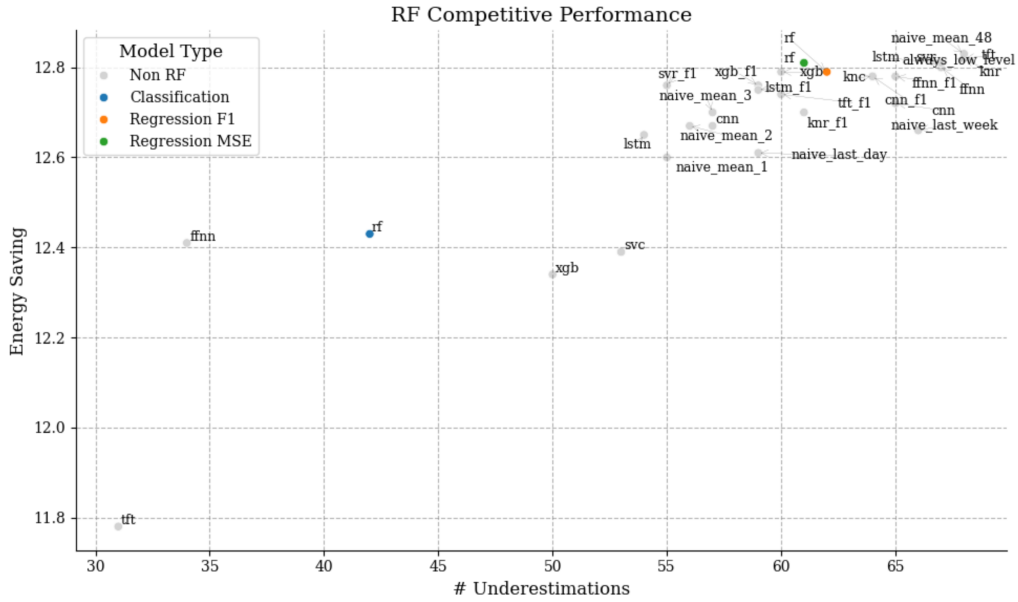


Figure 5.1: RF remained competitive in the overall evaluation.

In conclusion, selecting an optimal model for implementing dynamic voltage control in radios depends on the trade-off between maximizing energy savings and minimizing underestimations of voltage needs. Achieving higher energy savings comes at the cost of missing spikes in power demand. Further research is needed to better understand the negative effects of underestimations relative to energy savings and how these factors should be weighted against each other.

Two models with particularly favorable characteristics are the RF and FFNN classifiers. Both achieve energy savings of approximately 12.4%, while producing fewer underestimations compared to the majority of the evaluated models. In addition, selecting these classifiers allows for flexible adaptation of the decision threshold, as demonstrated in Section 4.5. Between the two, the FFNN offers an advantage in terms of producing fewer underestimations, while the RF is preferable in terms of lower model complexity.

5.3.3 Performance on Synthetic Data

The original data used for training and prediction in this thesis is heavily weighted toward the negative class, as previously discussed in Section 3.3.1. This reflects the fact that most utilization levels are low, meaning the radios typically operate below their capacity. An important question is how well the same models would perform under more extreme, high-utilization conditions. This question was addressed in Section 4.4 of the results. Since higher utilization data presents different potentials for energy savings and underestimations, the following discussion always considers performance as a percentage of the maximum achievable in each scenario.

Fig. 5.2 illustrates the performance of each model on the two datasets, expressed as a percentage of the optimal performance for both metrics. RF performs similarly in both contexts, though with a noticeable performance decrease in underestimations on the synthetic data. The XGB model shows good performance in reducing underestimations under the test condition but performs inefficiently in the synthetic high-utilization scenario. SVC's performance in underestimations also drops noticeably in the synthetic data.

On the other hand, KNC performs better on the synthetic data, maintaining energy savings close to the optimum while reducing underestimations. The FFNN reduces underestimations in the synthetic scenario, though with a slight drop in energy savings. The HybridLSTM also improves its underestimation performance under these conditions, as does the TFT. The HybridCNN maintains stable performance across both cases.

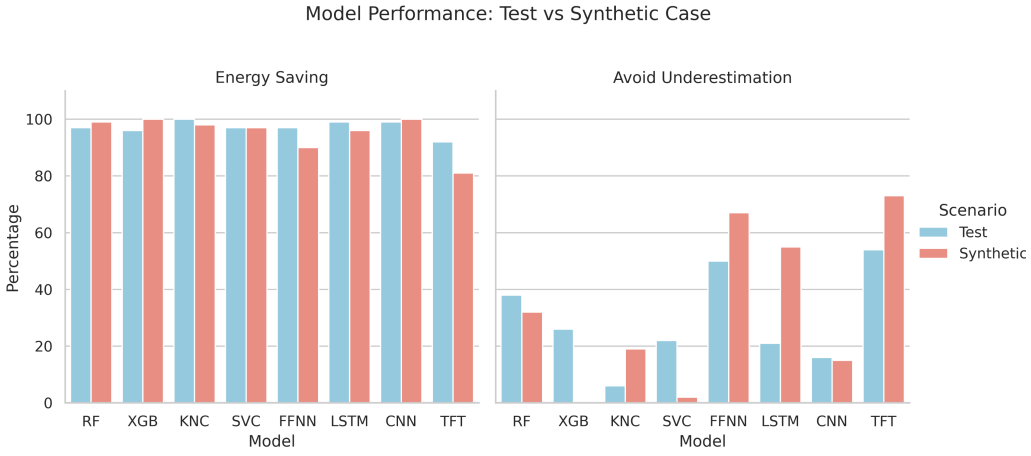


Figure 5.2: Performance difference between test set and the synthetic data set.

Another observation is that the benchmarking models became more competitive on the synthetic dataset. As shown in Table 4.15, the naive_mean_1 model had 261 underestimations and achieved a 10.1% energy savingcomparable to the FFNN with 250 underestimations and 9.9% savings, and the TFT with 241 underestimations and 8.88% savings. This increased competitiveness of the baseline models reinforces the earlier argument that the optimal solution may vary across radios or scenarios, and that even simple models can perform well in certain settings.

Concluding, both the RF and FFNN previously identified as top performers demonstrated relatively stable performance on the synthetic data. However, the RF showed a slight decline in underestimation accuracy, while the FFNN experienced a slight drop in energy savings.

5.4 RQ4: Explainability

An investigation of the top-performing models was conducted to better understand how they interpret the input features. Section 4.6 presented feature importances

with respect to predicting the higher voltage level, i.e., the positive class. Fig. 4.10 displayed the SHAP values for the RF classifier, while Fig. 4.11 presented those for the FFNN.

The SHAP analysis provides insight into how the two models make their predictions by identifying which input features contribute most strongly to the predicted outcomes. Across both models, PRB-U at the current timestep (t–0) emerged as the most influential feature. In both cases, this feature consistently exhibited high SHAP values: low PRB-U values reduced the predicted probability of class 1, while high values increased it. This finding aligns with domain expectations, as PRB-U in the next step is the target variable.

However, this feature is significantly more dominant in the FFNN than in the RF model. This suggests that PRB-U at t–0 has a disproportionately large influence on the FFNNs predictions, whereas the RF model distributes importance more evenly across multiple features. As a result, the FFNN may be more sensitive to outliers or noise in the PRB-U feature, since it relies heavily on this input. In contrast, the RF model may exhibit greater robustness, as it draws on a broader set of features.

The pattern remained consistent when inspecting individual predictions. In the selected examples, the FFNN assigned a higher probability than the RF model when correctly classifying the true positive instance, but it also assigned a higher probability to the true negative instance being classified as positive. Conversely, for false negative and false positive cases, the RF model was less confident in its incorrect predictions compared to the FFNN, suggesting potentially more cautious decision-making.

Moreover, some features exhibited the opposite SHAP value trend compared to PRB-U at t–0. For example, high values of mean power consumption at t–3 and PRB-U at t–1 were associated with negative SHAP contributions in the FFNN model, as shown in Fig. 4.11. Similar patterns were also observed in the RF model. This contrast suggests that both models may be capturing temporal dynamics, where high values in earlier timesteps indicate a reduction or reversal in the likelihood of the positive class.

In summary, the SHAP analysis provided insights into how the top-performing models utilize input features to make predictions. Both the RF and FFNN models relied heavily on PRB-U at the current timestep, which aligns with domain expectations. However, the FFNN exhibited a stronger dependence on this feature, potentially making it more sensitive to variations in PRB-U. In contrast, the RF model distributed importance more evenly, which may contribute to greater robustness. Additionally, both models appeared to capture temporal patterns, with certain earlier features contributing negatively to the likelihood of a positive classification. These findings illustrate that while both models are effective, their predictive strategies differ in ways that may have practical implications for model selection and deployment in dynamic voltage control scenarios.

5.5 Reliability of Results

An important limitation of this thesis is the selection of radios. The research was conducted on a single radio type, and it is therefore unclear whether the results are applicable to other radio types or to radios with more than one cell. Furthermore, this study examined only five randomly selected radios; an examination of the full population could yield different results.

Another important consideration is that all of the studied radios had sleep mode activated. A hypothesis emerging from this thesis is that radios without sleep mode could benefit even more from a dynamic voltage setting, as they are likely to experience low utilization during nighttime hours when voltage could be reduced.

The extent of the available data also presents a limitation. The dataset was drawn from the same year and covered only a limited number of months. As a result, potential monthly patterns and seasonal effects may not have been captured. In addition, the data was limited in terms of the number of instances in the positive class, reducing the reliability of certain results. This issue was partially addressed through the creation of synthetic data, as described in Section 4.4.

Finally, as noted in the delimitation of this thesis, only eight machine learning algorithms were examined. While these were considered the most relevant given the current state of the field, other algorithms could potentially yield favorable results as well.

6

Conclusion

This thesis implemented and compared three machine learning approaches for dynamically adapting voltage levels in radios, aiming to improve energy efficiency and thereby optimize for sustainability.

The results indicate that machine learning is a valuable approach for voltage control over time (RQ1). However, introducing a dynamic voltage creates a trade-off: while substantial energy can be saved, potential underestimations can lead to connectivity disturbances. The study also highlights radio variability, some radios benefit less from ML-based approaches and may perform as good or better with simple benchmark models or a static voltage reduction.

Among the implemented approaches, the classification formulation outperformed the regression setups in terms of reducing underestimations (RQ2). When using regression, tuning based on the F1 score rather than MSE is recommended, as it lowers the number of restarts while preserving energy savings (RQ2.1). Each setup showed different preferences regarding length of the lag parameter from the Optuna trials (RQ2.2). Classification models tended to rely less on long temporal histories (mean preferred lag: 37.6), while regression models selected using F1 and MSE preferred longer lags (mean preferred lags: 63.1 and 43.8, respectively).

The thesis proposes a Random Forest (RF) classifier and a customized Feedforward Neural Network (FFNN) classifier as promising candidates (RQ3). Both achieved substantial energy savings and relatively few underestimations: the RF had 42 underestimations with a 12.43% energy saving, while the FFNN had 34 underestimations with a 12.41% saving. The robustness of the models was validated using a synthetic test set where PRB utilization (PRB-U) was doubled. Both models maintained relatively stable performance. The FFNN showed reduced energy savings but fewer underestimations, while the RF experienced increased energy efficiency and a slight drop in underestimations

Furthermore, a specified advantage of classification is the availability of prediction probabilities, which enables easy control over the trade-off between energy savings and underestimations. As shown in this thesis, the FFNN classifier could be tuned to achieve only 5 underestimations while saving 10% energy, by adjusting the decision threshold.

Explainability analysis of the RF and FFNN classifier confirmed that the PRB-U in the current timestep is the most important feature for both models (RQ4), which

aligns with domain knowledge. Notably, the RF model distributed importance more evenly across features, suggesting higher robustness, while the FFNN showed a heavier reliance on one dominant feature.

In summary, this work contributes to the area of building energy-aware radios and emphasizes the importance of balancing energy efficiency and underestimations in deployment. Future work should investigate the impact of voltage underestimations to better inform model selection, objective function design, and deployment strategies.

Bibliography

[1] U. N. F. C. on Climate Change (UNFCCC), *The paris agreement*, Available at, 2015. [Online]. Available: https://unfccc.int/process-and-meetings/the-paris-agreement/the-paris-agreement.

[2] C. C. C. Service, *Copernicus: January 2025 was the warmest on record globally, despite an emerging la niña*, Accessed: 2025-03-18, Feb. 2025. [Online]. Available: https://climate.copernicus.eu/press-release/copernicus-january-2025-was-warmest-record-globally-despite-emerging-la-ni%C3%B1a.

[3] J. Malmodin, N. Lövehagen, P. Bergmark, and D. Lundén, *Ict sector electricity consumption and greenhouse gas emissions 2020 outcome*, Available at SSRN: https://ssrn.com/abstract=4424264 or http://dx.doi.org/10.2139/ssrn.4424264, Apr. 2023.

[4] Ember and E. Institute, *Carbon intensity of electricity generation ember and energy institute*, Dataset, Major processing by Our World in Data. Original data from Ember, "Yearly Electricity Data"; Energy Institute, "Statistical Review of World Energy". Retrieved March 21, 2025., 2024. [Online]. Available: https://ourworldindata.org/grapher/carbon-intensity-electricity.

[5] J. G. Gómez, A. Matlok, T. Silveira, and L. Verboven, "The growing imperative of energy optimization for telco networks," Feb. 2024, Accessed: 2025-04-02. [Online]. Available: https://www.mckinsey.com/industries/technology-media-and-telecommunications/our-insights/the-growing-imperative-of-energy-optimization-for-telco-networks.

[6] T. M. Alabi, E. I. Aghimien, F. D. Agbajor, *et al.*, "A review on the integrated optimization techniques and machine learning approaches for modeling, prediction, and decision making on integrated energy systems," *Renewable Energy*, vol. 194, pp. 822–849, 2022, ISSN: 0960-1481. DOI: https://doi.org/10.1016/j.renene.2022.05.123. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0960148122007741.

[7] D. Koolen, N. Sadat-Razavi, and W. Ketter, "Machine learning for identifying demand patterns of home energy management systems with dynamic electricity pricing," *Applied Sciences*, vol. 7, no. 11, 2017, ISSN: 2076-3417. DOI: 10.3390/app7111160. [Online]. Available: https://www.mdpi.com/2076-3417/7/11/1160.

[8] G. Dhiman and T. S. Rosing, "Dynamic power management using machine learning," in *Proceedings of the 2006 IEEE/ACM International Conference on Computer-Aided Design*, ser. ICCAD '06, San Jose, California: Association for

Computing Machinery, 2006, pp. 747–754, ISBN: 1595933891. DOI: 10.1145/1233501.1233656. [Online]. Available: https://doi.org/10.1145/1233501.1233656.

[9] C. Zhang, L. Zhao, Z. Yin, and Z. Zhang, "Causalformer: Causal discovery-based transformer for multivariate time series forecasting," in *2023 16th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, IEEE, 2023, pp. 1–10. DOI: 10.1109/CISP-BMEI60920.2023.10373365. [Online]. Available: https://doi.org/10.1109/CISP-BMEI60920.2023.10373365.

[10] M. Tzelepi, C. Symeonidis, P. Nousi, *et al.*, *Deep learning for energy time-series analysis and forecasting*, 2023. arXiv: 2306.09129 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2306.09129.

[11] B. Yildirim and M. Taskiran, "Optuna based optimized transformer model approach in bitcoin time series analysis," in *2024 26th International Conference on Digital Signal Processing and its Applications (DSPA)*, 2024, pp. 1–6. DOI: 10.1109/DSPA60853.2024.10510091.

[12] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '19, Anchorage, AK, USA: Association for Computing Machinery, 2019, pp. 2623–2631, ISBN: 9781450362016. DOI: 10.1145/3292500.3330701. [Online]. Available: https://doi.org/10.1145/3292500.3330701.

[13] S. Lundberg and S.-I. Lee, *A unified approach to interpreting model predictions*, 2017. arXiv: 1705.07874 [cs.AI]. [Online]. Available: https://arxiv.org/abs/1705.07874.

[14] O. Liberg, M. Sundberg, Y.-P. E. Wang, J. Bergman, and J. Sachs, "Chapter 5 - lte-m," in *Cellular Internet of Things*, O. Liberg, M. Sundberg, Y.-P. E. Wang, J. Bergman, and J. Sachs, Eds., Academic Press, 2018, pp. 135–197, ISBN: 978-0-12-812458-1. DOI: https://doi.org/10.1016/B978-0-12-812458-1.00005-8. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128124581000058.

[15] A. Ghosh, J. Zhang, J. G. Andrews, and R. Muhamed, *Fundamentals of LTE*. Pearson Education, 2010.

[16] Z.-H. Zhou, *Machine Learning*. Springer Nature, 2021.

[17] B. Krawczyk, "Learning from imbalanced data: Open challenges and future directions," *Progress in artificial intelligence*, vol. 5, no. 4, pp. 221–232, 2016.

[18] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[19] Z. S. Kadhim, H. S. Abdullah, and K. I. Ghathwan, "Artificial neural network hyperparameters optimization: A survey," *Int. J. Online Biomed. Eng.*, vol. 18, pp. 59–87, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:254391194.

[20] J. Kim, H. Kim, H. Kim, *et al.*, "A comprehensive survey of time series forecasting: Architectural diversity and open challenges," 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:273963385.

[21] Y. Feng, Y. Zhang, and Y. Wang, "Out-of-sample volatility prediction: Rolling window, expanding window, or both?" *Journal of Forecasting*, vol. 43, no. 3, pp. 567–582, 2024.

[22] J. D. Freitas, C. Ponte, R. Bomfim, and C. Caminha, "The impact of window size on univariate time series forecasting using machine learning," *Anais do XI Symposium on Knowledge Discovery, Mining and Learning (KDMiLe 2023)*, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:263723373.

[23] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, Oct. 2001, ISSN: 1573-0565. DOI: 10.1023/A:1010933404324. [Online]. Available: https://doi.org/10.1023/A:1010933404324.

[24] J. R. Quinlan, "Induction of decision trees," *Machine learning*, vol. 1, pp. 81–106, 1986.

[25] Scikit-learn developers. "Decision trees." Accessed: 2025-05-15. (2025), [Online]. Available: https://scikit-learn.org/stable/modules/tree.html#tree-mathematical-formulation.

[26] M. R. Segal, "Machine learning benchmarks and random forest regression," 2004.

[27] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794.

[28] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.

[29] Scikit-learn developers. "Gradientboostingregressor." Accessed: 2025-05-13. (2025), [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html.

[30] P. Cunningham and S. Delany, "K-nearest neighbour classifiers," *Mult Classif Syst*, vol. 54, Apr. 2007. DOI: 10.1145/3459665.

[31] O. Kramer and O. Kramer, "K-nearest neighbors," *Dimensionality reduction with unsupervised nearest neighbors*, pp. 13–23, 2013.

[32] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, pp. 273–297, 1995.

[33] H. Drucker, C. J. C. Burges, L. Kaufman, A. Smola, and V. Vapnik, "Support vector regression machines," in *Advances in Neural Information Processing Systems*, M. Mozer, M. Jordan, and T. Petsche, Eds., vol. 9, MIT Press, 1996. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/1996/file/d38901788c533e8286cb6400b40b386d-Paper.pdf.

[34] D. Basak, S. Pal, D. C. Patranabis, *et al.*, "Support vector regression," *Neural Information Processing-Letters and Reviews*, vol. 11, no. 10, pp. 203–224, 2007.

[35] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[36] E. Alpaydin, *Introduction to machine learning*. MIT press, 2020.

[37] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation functions: Comparison of trends in practice and research for deep learning," *arXiv preprint arXiv:1811.03378*, 2018.

[38] P. Contributors, *Gelu*, Accessed: 2025-05-15, 2025. [Online]. Available: https://docs.pytorch.org/docs/stable/generated/torch.nn.GELU.html.

[39] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, pp. 1929–1958, 2014. [Online]. Available: https://api.semanticscholar.org/CorpusID:6844431.

[40] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *ArXiv*, vol. abs/1502.03167, 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:5808102.

[41] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:206594692.

[42] Z. C. Lipton, J. Berkowitz, and C. Elkan, *A critical review of recurrent neural networks for sequence learning*, 2015. arXiv: 1506.00019 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1506.00019.

[43] A. Sherstinsky, "Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network," *Physica D: Nonlinear Phenomena*, vol. 404, p. 132 306, 2020.

[44] G. Van Houdt, C. Mosquera, and G. Nápoles, "A review on the long short-term memory model," *Artificial Intelligence Review*, vol. 53, Dec. 2020. DOI: 10.1007/s10462-020-09838-1.

[45] B. Xu, "Multi-channel convolutional neural network with long short-term memory for power grid safety monitoring," *2024 Second International Conference on Data Science and Information System (ICDSIS)*, pp. 1–4, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:271295895.

[46] Y. Lecun and Y. Bengio, "Convolutional networks for images, speech, and time-series," Jan. 1995.

[47] S. Indolia, A. K. Goswami, S. Mishra, and P. Asopa, "Conceptual understanding of convolutional neural network- a deep learning approach," *Procedia Computer Science*, vol. 132, pp. 679–688, 2018, International Conference on Computational Intelligence and Data Science, ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2018.05.069. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050918308019.

[48] K. O'shea and R. Nash, "An introduction to convolutional neural networks," *arXiv preprint arXiv:1511.08458*, 2015.

[49] B. Lim, S. O. Arik, N. Loeff, and T. Pfister, *Temporal fusion transformers for interpretable multi-horizon time series forecasting*, 2020. arXiv: 1912.09363 [stat.ML]. [Online]. Available: https://arxiv.org/abs/1912.09363.

[50] J. L. Joy and M. P. Selvan, "A comprehensive study on the performance of different multi-class classification algorithms and hyperparameter tuning techniques using optuna," *2022 International Conference on Computing, Communication, Security and Intelligent Systems (IC3SIS)*, pp. 1–5, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:252312119.

[51] O. Contributors, *Efficient optimization algorithms*, Accessed: 2025-04-07, Optuna Documentation, 2018. [Online]. Available: https://optuna.readthedocs.io/en/stable/tutorial/10_key_features/003_efficient_optimization_algorithms.html#pruning.

[52] V. Vimbi, N. Shaffi, and M. Mahmud, "Interpreting artificial intelligence models: A systematic review on the application of lime and shap in alzheimers disease detection," *Brain Informatics*, vol. 11, no. 1, p. 10, 2024. DOI: 10.1186/s40708-024-00222-1. [Online]. Available: https://doi.org/10.1186/s40708-024-00222-1.

[53] H. Akoglu, "User's guide to correlation coefficients," *Turkish journal of emergency medicine*, vol. 18, no. 3, pp. 91–93, 2018.

[54] L. Grinsztajn, E. Oyallon, and G. Varoquaux, *Why do tree-based models still outperform deep learning on tabular data?* 2022. arXiv: 2207.08815 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2207.08815.

[55] O. Contributors, *Optuna.samplers*, Accessed: 2025-04-07, Optuna, 2018. [Online]. Available: https://optuna.readthedocs.io/en/stable/reference/samplers/index.html#optuna-samplers.

[56] S. learn developers, *F1_score*, https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html, Accessed: 2025-05-13, 2025.

[57] B. Courty, V. Schmidt, S. Luccioni, *et al.*, *Mlco2/codecarbon: V2.4.1*, version v2.4.1, May 2024. DOI: 10.5281/zenodo.11171501. [Online]. Available: https://doi.org/10.5281/zenodo.11171501.

[58] R. Wan, S. Mei, J. Wang, M. Liu, and F. Yang, "Multivariate temporal convolutional network: A deep neural networks approach for multivariate time series forecasting," *Electronics*, vol. 8, no. 8, 2019, ISSN: 2079-9292. DOI: 10.3390/electronics8080876. [Online]. Available: https://www.mdpi.com/2079-9292/8/8/876.

[59] Matplotlib Development Team, *Matplotlib.pyplot*, Accessed: 2025-05-13, Matplotlib, 2022. [Online]. Available: https://matplotlib.org/3.5.3/api/_as_gen/matplotlib.pyplot.html#module-matplotlib.pyplot.

[60] T. N. D. Team, *Numpy: Fundamental package for scientific computing with python*, https://numpy.org/, Version 1.26.x, 2024.

[61] Optuna Contributors, *Optuna: A Hyperparameter Optimization Framework*, https://optuna.org/, Accessed: 2025-05-28, 2024.

[62] Optuna Contributors, *TPESampler Optuna Documentation*, https://optuna.readthedocs.io/en/stable/reference/samplers/generated/optuna.samplers.TPESampler.html, Accessed: 2025-05-28, 2024.

[63] Optuna Contributors, *MedianPruner Optuna Documentation*, https://optuna.readthedocs.io/en/stable/reference/generated/optuna.pruners.MedianPruner.html, Accessed: 2025-05-28, 2024.

[64] T. pandas development team, *Pandas: Powerful python data analysis toolkit*, https://pandas.pydata.org/, Version 2.x, 2024.

[65] PyTorch Contributors, *torch.nn.Linear PyTorch Documentation*, https://pytorch.org/docs/stable/generated/torch.nn.Linear.html, Accessed: 2025-05-28, 2024.

[66] PyTorch Contributors, *torch.nn.Dropout PyTorch Documentation*, https://pytorch.org/docs/stable/generated/torch.nn.Dropout.html, Accessed: 2025-05-28, 2024.

[67] PyTorch Contributors, *torch.nn.BatchNorm1d PyTorch Documentation*, https://pytorch.org/docs/stable/generated/torch.nn.BatchNorm1d.html, Accessed: 2025-05-28, 2024.

[68] PyTorch Contributors, *torch.nn.LSTM PyTorch Documentation*, https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html, Accessed: 2025-05-28, 2024.

[69] PyTorch Contributors, *torch.nn.Conv1d PyTorch Documentation*, https://pytorch.org/docs/stable/generated/torch.nn.Conv1d.html, Accessed: 2025-05-28, 2024.

[70] P. Contributors, *Maxpool1d*, Accessed: 2025-05-13, 2025. [Online]. Available: https://docs.pytorch.org/docs/stable/generated/torch.nn.MaxPool1d.html#torch.nn.MaxPool1d.

[71] P. Contributors, *Adaptiveavgpool1d*, Accessed: 2025-05-13, 2025. [Online]. Available: https://docs.pytorch.org/docs/stable/generated/torch.nn.AdaptiveAvgPool1d.html.

[72] PyTorch Forecasting Contributors, *TemporalFusionTransformer PyTorch Forecasting Documentation*, https://pytorch-forecasting.readthedocs.io/en/v1.0.0/api/pytorch_forecasting.models.temporal_fusion_transformer.TemporalFusionTransformer.html, Accessed: 2025-05-28, 2024.

[73] PyTorch Forecasting Contributors, *TimeSeriesDataSet PyTorch Forecasting Documentation*, https://pytorch-forecasting.readthedocs.io/en/v1.0.0/api/pytorch_forecasting.data.timeseries.TimeSeriesDataSet.html, Accessed: 2025-05-28, 2024.

[74] PyTorch Lightning Contributors, *PyTorch Lightning Documentation*, https://lightning.ai/docs/pytorch/stable/, Accessed: 2025-05-28, 2024.

[75] S. learn developers, *Randomforestregressor*, https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html, Accessed: 2025-05-13, 2025.

[76] S. learn developers, *Svr*, https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html, Accessed: 2025-05-13, 2025.

[77] S. learn developers, *Kneighborsregressor*, https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html, Accessed: 2025-05-13, 2025.

[78] M. Waskom, *Seaborn.heatmap*, Accessed: 2025-05-13, 2024. [Online]. Available: https://seaborn.pydata.org/generated/seaborn.heatmap.html#seaborn.heatmap.

[79] SHAP Contributors, *SHAP Documentation*, https://shap.readthedocs.io/en/latest/, Accessed: 2025-05-28, 2024.

[80] M. Ashraf, F. Anowar, J. H. Setu, *et al.*, "A survey on dimensionality reduction techniques for time-series data," *IEEE Access*, vol. 11, pp. 42 909–42 923, 2023. DOI: 10.1109/ACCESS.2023.3269693.

[81] M. Kirisci and O. Cagcag Yolcu, "A new cnn-based model for financial time series: Taiex and ftse stocks forecasting," *Neural Processing Letters*, vol. 54,

no. 4, pp. 3357–3374, Aug. 2022, ISSN: 1573-773X. DOI: 10.1007/s11063-022-10767-z. [Online]. Available: https://doi.org/10.1007/s11063-022-10767-z.

A

Completing Data Visualization

This appendix contains additional figures and data analysis over the system.

A.1 Histograms of the Target Variable

Fig. A.1, A.2, and A.3 display the distribution of the target variable PRB utilization for serial numbers 3, 4 and 5. The plots for serial 1 and 2 can be found in Fig. 3.1.

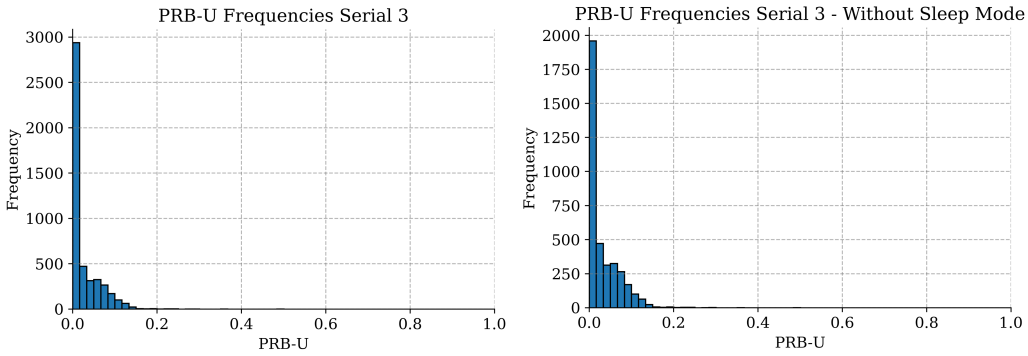


Figure A.1: Histogram of the target variable *PRB utilization* with and without sleep mode for serial 3.

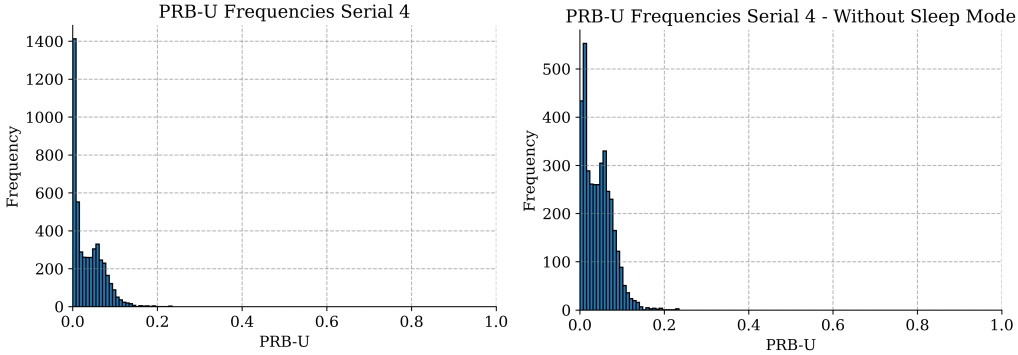


Figure A.2: Histogram of the target variable *PRB utilization* with and without sleep mode for serial 4.

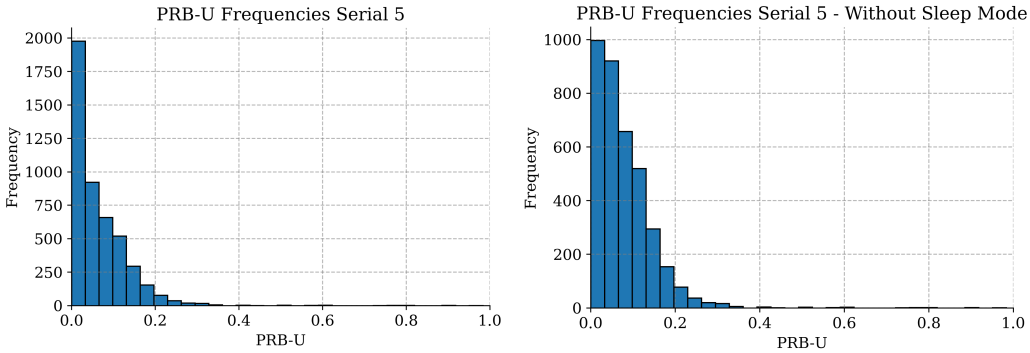


Figure A.3: Histogram of the target variable *PRB utilization* with and without sleep mode for serial 5.

A.2 Heatmap Target and Mean Power Consumption

Fig. A.4 shows the correlation of the tarvet variable with the mean P_{in} which shows very similar patterns as previously shown an discussed in section 3.3.2. The lag t-96 reflexts a full 24h.

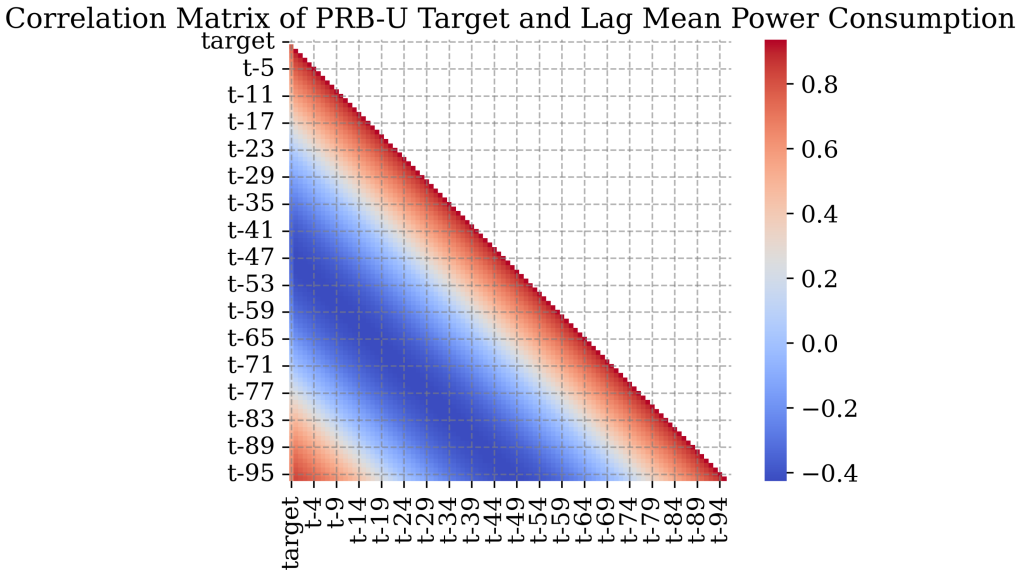


Figure A.4: Correlation between the target variable and the previous mean power consumption lag values.

A.3 PRB Utilization: Weekly Patterns

The report showed heatmaps of serial 2, 4, and 5 in section 3.3.2. Fig. A.5 and A.6 show the other serial numbers' heat maps.

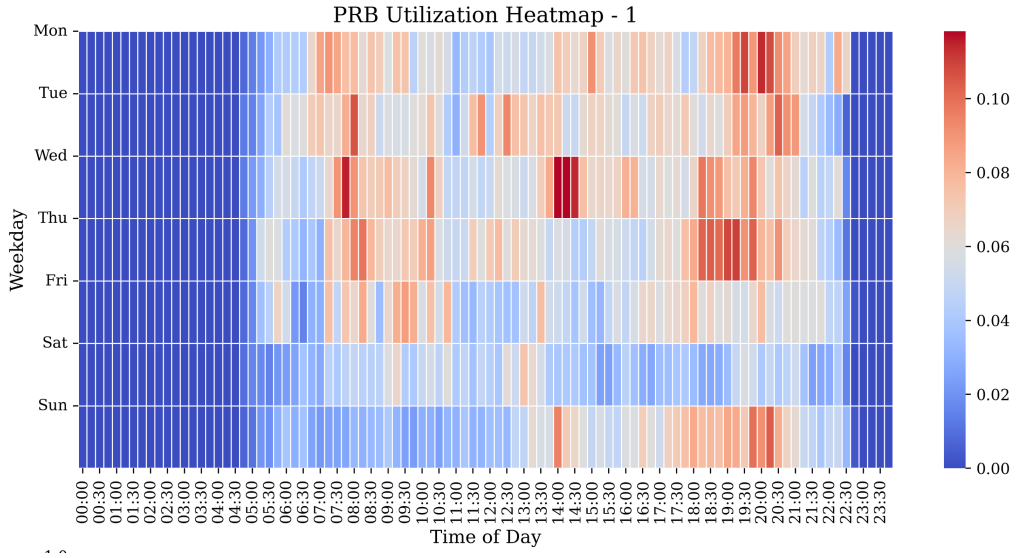


Figure A.5: *PRB utilization* heatmap serial 1.

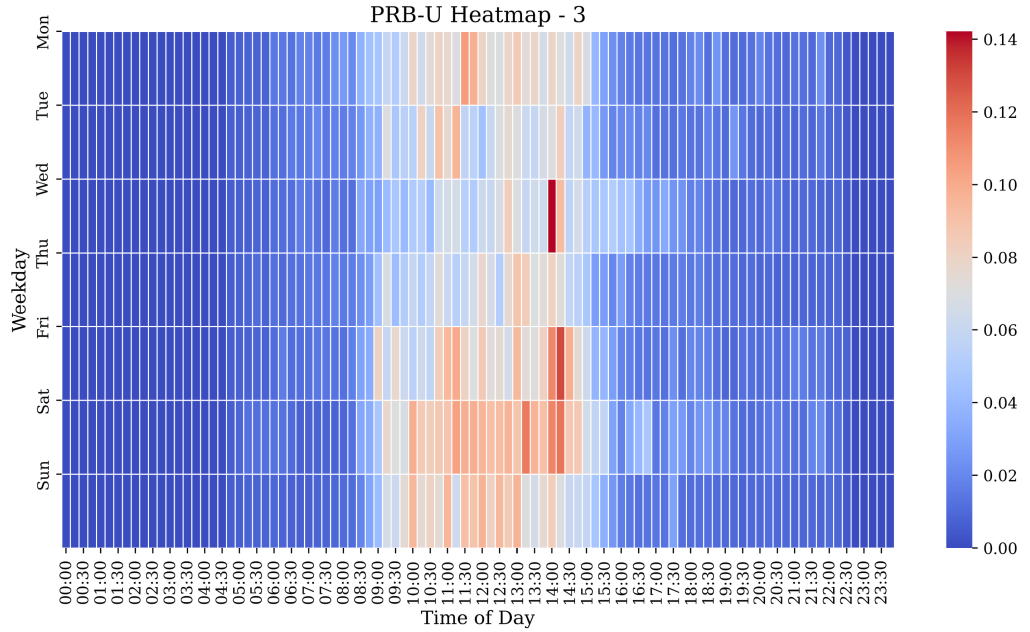


Figure A.6: *PRB utilization* heatmap serial 3.

A.4 PRB Utilization: Ten Day Curves

Section 3.3.2 showed patterns in PRB utilization for serial 2 and 4 in Fig. 3.6 and for serial 5 in Fig. 3.8. Fig. A.7 and A.8 completes these plots by showing the pattern of serial 1 and 3.

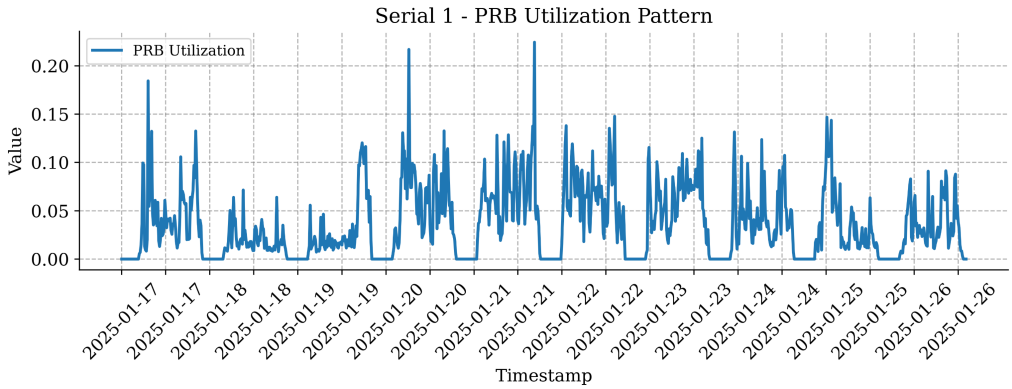


Figure A.7: PRB utilization pattern for serial 1.

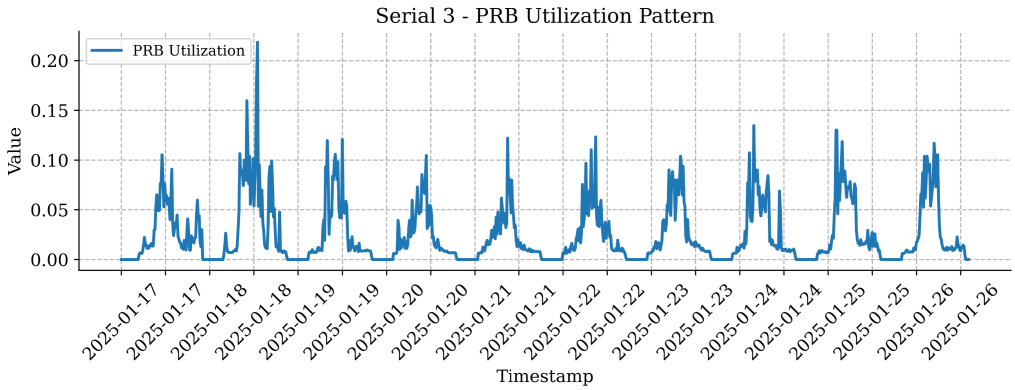


Figure A.8: PRB utilization pattern for serial 3.

B

Hyperparameter Search Space

This appendix specifies the hyperparameter space which was investigated for each model. Table B.1 shows abbreviations used for those including scaler options. Any cell marked with - indicate that the other task formulations have that parameter but it is not available in that specific one.

Scaler	Abbreviation
RobustScaler	standard
MinMaxScaler	minmax
StandardScaler	robust
EncoderNormalizer	encnorm

Table B.1: Scalers and their abbreviations.

B.1 Common Parameters

Several hyperparameters were shared across multiple models. Among these were the choice of scaler and the lag in the data. Scalers were adopted from the `Scikit-learn` library and the lag was set to be up to one day.

For the neural network models, additional common hyperparameters included the optimizer, learning rate, and number of training epochs. All except the number of epochs were included as tunable parameters within the Optuna optimization process. Epochs was set to 125 across models as described in more details in section 3.7.

B.2 Statistical Models

The search spaces for the hyperparameters related to data preprocessing, which are shared across most non-neural models, are presented in Table B.2. Where scaler is the scaler used for the data and lag the amount of look back the model gets. The hyperparameter search spaces specific to the different models are displayed in the following tables.

Parameter	Type	Range
`scaler`	Categorical	`standard`, `minmax`, `robust`
`lag_value`	Integer	(1, 96)

Table B.2: Hyperparameter search space for data preprocessing

B.2.1 Random Forest Regressor

Table B.3 gives the optimal parameters found through optuna for RF. *n_estimators* refers to the number of trees in the forest, *max_depth* to the depth that the trees are allowed, *min_samples_split* is the minimum number of samples needed for a split, and *min_samples_leaf* the minimum number of samples required to create a node [75].

Parameter	Type	Range
`n_estimators`	Integer (log-uniform)	(50, 300)
`max_depth`	Integer (log-uniform)	(4, 20)
`min_samples_split`	Integer	(2, 10)
`min_samples_leaf`	Integer	(1, 10)

Table B.3: Hyperparameter search space for RF.

B.2.2 Extreme Gradient Boosting

Similarly to RF, *n_estimators* indicated the number of trees [29]. *learning_rate* decreases the contribution of each tree by its rate, *max_depth* is as in RF the max allowed depth of the trees, *subsample* is the percentage of samples used to train each individual tree. Finally, *colsample_bytree*

Parameter	Type	Range
`n_estimators`	Integer (log-uniform)	(50, 300)
`learning_rate`	Float (log-uniform)	(0.01, 0.2)
`max_depth`	Integer	(3, 10)
`subsample`	Float	(0.7, 1.0)
`colsample_bytree`	Float	(0.7, 1.0)

Table B.4: Hyperparameter search space for XGB.

B.2.3 K-Nearest Neighbors

The unsupervised ML algorithm was given the search space as shown in Table B.5. First, *n_neighbors* is the number of neighbors used to make a prediction, *weights* decides how much each neighbors' value contributes with [77]. *leaf_size* aids in memory and speed of the algorithm, and p decides the function for distances.

Parameter	Type	Range
`n_neighbors`	Integer	(2, 30)
`weights`	Categorical	`uniform`, `distance`
`algorithm`	Categorical	`auto`, `kd_tree`
`leaf_size`	Integer	(20, 50)
`p`	Categorical	1, 2

Table B.5: Hyperparameter search space for KNC/R.

B.2.4 Support Vector Machine

The C parameter controls the strength of the regularization [76]. The kernel coefficient is calculated according to the parameter *gamma*, *scale*, *epsilon* controls the radius out from which there is no penalty added during training phase. Lastly, *kernel* is used to calculate the similarity or difference between data points. Selecting its function is crucial for creating a functional model.

Parameter	Type	Range
`C`	Float (log-uniform)	(0.01, 10)
`gamma`	Categorical	`scale`, `auto`, 0.001, 0.01, 0.1
`epsilon`	Float (log-uniform)	(0.001, 0.1)
`kernel`	Categorical	`linear`, `rbf`

Table B.6: Hyperparameter search space for SVC/R.

B.3 Neural Networks

The common search spaces are presented in Table. B.7. *Scaler* describes the way data is transformed for the ML models, *lag* value defines the lag in the data and hence the information back in time that the models get. *Batch_size* is the number of samples used at a time during training, *optimizer* controls the change of the model during training phase. The learning rate *lr* is closely connected to the optimizer in that it chooses the amount of change that is done in every training loop. The parameter *grad_clip* can be activated to decrease the change of exploding gradients. *hidden_size* is the number of neurons in the hidden layers and *use_sigmoid* defines if a last sigmoid activation function is used in the model.

Parameter	Type	Range
`scaler`	Categorical	`standard`, `minmax`, `robust`
`lag_value`	Integer	(1, 96)
`batch_size`	Categorical	16, 32, 64, 128
`optimizer`	Categorical	`adam`, `rmsprop`, `sgd`
`lr`	Float (log-uniform)	$(1 \times 10^{-5}, 1 \times 10^{-2})$
`grad_clip`	Boolean	`True`, `False`
`hidden_size`	Integer (step = 32)	(32, 256)
`use_sigmoid`	Boolean	`True`, `False`

Table B.7: Hyperparameter search space for neural networks' common hyperparameters.

B.3.1 Feed Forward Neural Network

Num_layers is the number of hidden layers in the neural net, *dropout_rate* prevents overfitting by setting a neuron's output to 0 with the specified rate. The *activation_func* controls the activation function used and *use_residuals* controls weather residuals are used.

Parameter	Type	Range
`num_layers`	Integer	(1, 4)
`dropout_rate`	Float (step = 0.1)	(0.0, 0.5)
`activation_func`	Categorical	`relu`, `leaky_relu`, `elu`, `gelu`
`use_residuals`	Boolean	`True`, `False`

Table B.8: Hyperparameter search space for FFNN

B.3.2 Convolutional Neural Network

Table B.9 shows the hyperparameter space for the hybrid CNN model. 1 and 2 in the end of the first 12 parameters indicates weather it is applied to the first or second convolutional block. The channels created by the convolution is controlled by *branch_out_channels*, *kernel_size* is the size of the kernel convolving over the sequence. Kernel size is defined to never be larger than the input size, lag, as it is impossible to convolve a larger kernel than the input.

The *stride* defines the jump after each convolution, *padding* parameter is the potential extra input added to the beginning and end of the sequence [69]. The parameter *ad_avg_pool* controlls the size of the output of the AdaptiveAvgPool1d layer, which takes a mean and creates an output of the specified size [71]. The *max_pooling* parameter controls the kernel_size parameter of the MaxPool1d layer [70].

Static_hidden is the neuron size of the non sequential hidden layer and *out_static* of its output layer. *Conc_hidden* is the neuron size of the concatenation later. *Out_to_concat* controls the resize using a AdaptiveAvgPool1d layer of each indi-

vidual output before entering the concatenation layer. The *activation* parameters control the activation function for each block as indicated by its suffix.

Parameter	Type	Range
branch_out_channels1	Integer (step=8)	(8, 128)
branch_out_channels2	Integer (step=8)	(8, 128)
kernel_size1	Integer (step=1)	(1, min(10, lag))
kernel_size2	Integer (step=1)	(1, min(10, lag))
stride1	Integer (step=1)	(1, 7)
stride2	Integer (step=1)	(1, 7)
padding_type1	Categorical	same, valid
padding_type2	Categorical	same, valid
ad_avg_pool1	Integer (step=8)	(8, 128)
ad_avg_pool2	Integer (step=8)	(8, 128)
max_pooling_kernel1	Integer (step=1)	(1, 10)
max_pooling_kernel2	Integer (step=1)	(1, 10)
static_hidden	Integer (step=8)	(8, 128)
out_static	Integer (step=8)	(8, 128)
conc_hidden	Integer (step=8)	(8, 128)
out_to_concat	Integer (step=8)	(8, 128)
activation_seq1	Categorical	relu, leaky_relu, sigmoid, tanh
activation_seq2	Categorical	relu, leaky_relu, sigmoid, tanh
activation_concat	Categorical	relu, leaky_relu, sigmoid, tanh
activation_static	Categorical	relu, leaky_relu, sigmoid, tanh

Table B.9: Hyperparameter search space for CNN

B.3.3 Long Short Memory

Table B.10 shows the individual tuning parameters for the hybrid LSTM model. *lstm_layers* represents the number of hidden layers included in each LSTM module in the model. The *lstm_dropout* parameter determines the dropout rate applied between hidden layers inside each LSTM module; it is only used if the number of hidden layers is greater than one. The *static_hidden_size_1* and *static_hidden_size_2* parameters control the sizes of the first and second linear layers in the static branch of the LSTM.

Parameter	Type	Range
`lstm_layers`	Integer	(1, 3)
`lstm_dropout`	Float (step = 0.1)	(0.0, 0.5
`static_hidden_size_1`	Integer (step = 16)	(16, 128)
`static_hidden_size_2`	Integer (step = 8)	(8, 64)

Table B.10: Hyperparameter search space for LSTM

B.4 Temporal Fusion Transformer

The search space for the TFT is presented in Table B.12. Since the TFT was implemented using the `pytorch-forecasting` library, the data was converted to a PyTorch Forecasting time series dataset. While tuning the TFT model, the parameters for the time series dataset were tuned as well. Since the Pytorch Forecasting library has different namings and configurations than regular neural networks implemented in torch, the TFT did not share the same common parameters as the other neural networks.

The search space for the dataset configuration is presented in Table B.11. *max_encoder_length* and *max_prediction_length* define the maximum number of time steps included in the encoder and decoder, respectively. The length of the encoder is the TFTs equivalent of the *lag_value*. *encoder_random_ratio* controls the fraction of *max_encoder_length* to select as the minimum encoder length. The Boolean flag *add_relative_time_idx* controls whether to include relative time indices for sequences. These indices range from encoder length to prediction length. *add_target_scales* decides if target scales for static real features should be added, and *add_encoder_length* if encoder length should be added to static real variables. *randomize_length* enables random variation in encoder lengths per sample. The *target_normalizer* and *scaler* parameters define the normalization strategies applied to the target and input features, respectively.

Parameter	**Type**	**Range**
`max_encoder_length`	Integer	(7, 60)
`encoder_random_ratio`	Float	(0.5, 1.0)
`max_prediction_length`	Integer	(3, 30)
`add_relative_time_idx`	Boolean	`True`, `False`
`add_target_scales`	Boolean	`True`, `False`
`add_encoder_length`	Categorical	`"auto"`, `True`, `False`
`randomize_length`	Boolean	`True`, `False`
`target_normalizer`	Categorical	`auto`, `TorchNormalizer`, `GroupNormalizer` `EncoderNormalizer`
`scaler`	Categorical	`StandardScaler`, `RobustScaler` `EncoderNormalizer`

Table B.11: Hyperparameter search space for the dataset configuration

The parameter setup for the TFT model is shown in Table B.12. The *hidden_size* represents the number of neurons in each hidden layer. The *lstm_layers* parameter determines the number of LSTM layers to include in the model. The dropout rate is controlled by the *dropout* parameter. The *attention_head_size* represents the number of attention heads to include, and *hidden_continuous_size* specifies the hidden size for the layers processing continuous variables. *reduce_on_plateau_patience* controls the patience before reducing the learning rate by a factor of 10. The hyperparameter *share_single_variable_networks* determines whether the encoder and

decoder share the single-variable networks. *causal_attention* controls whether the decoder is allowed to attend to future time steps. The *learning_rate* and *batch_size* parameters were described in Section B.3.

Parameter	**Type**	**Range**
`hidden_size`	Categorical	`8`, `16`, `32`, `64`, `128`, `256`
`lstm_layers`	Integer	(1, 3)
`dropout`	Float	(0.05, 0.4)
`attention_head_size`	Integer	(1, 8)
`hidden_continuous_size`	Categorical	`4`, `8`, `16`, `32`
`reduce_on_plateau_patience`	Integer (log-uniform)	(5, 15)
`share_single_variable_networks`	Boolean	`True`, `False`
`causal_attention`	Boolean	`True`, `False`
`learning_rate`	Float (log-uniform)	(10^{-4}, 10^{-2})
`batch_size`	Categorical	`32`, `64`, `128`

Table B.12: Hyperparameter search space for TFT

C

Optimal Hyperparameters

This appendix shows the optimal hyperparameter space for each model.

Hyperparameter	C	R(F1)	R(MSE)
params_lag_value	6	96	96
params_max_depth	19	17	15
params_n_estimators	165	92	155
params_min_samples_leaf	10	3	7
params_min_samples_split	3	9	4

Table C.1: Optimal hyperparameters RF.

Hyperparameter	C	R(F1)	R(MSE)
params_lag_value	79	94	6
params_max_depth	10	6	4
params_n_estimators	199	254	88
params_subsample	0.77	0.97	0.98
params_colsample_bytree	0.72	0.78	0.83
params_learning_rate	0.019	0.057	0.12

Table C.2: Optimal hyperparameters XGB.

Hyperparameter	C	R(F1)	R(MSE)
params_lag_value	1	1	1
params_algorithm	auto	auto	kd_tree
params_leaf_size	34	23	35
params_n_neighbors	3	2	29
params_p	2	1	2
params_weights	uniform	distance	distance
params_scaler	standard	robust	robust

Table C.3: Optimal hyperparameters KNC/R.

Hyperparameter	C	R(F1)	R(MSE)
params_lag_value	71	96	67
params_C	6.99	1.09	0.11
params_gamma	auto	scale	scale
params_kernel	rbf	linear	rbf
params_epsilon	-	0.0097	0.0010
params_scaler	robust	minmax	minmax

Table C.4: Optimal hyperparameters SVC/R.

Hyperparameter	C	R(F1)	R(MSE)
params_lag_value	5	74	25
params_activation_func	relu	leaky_relu	elu
params_dropout_rate	0.3	0.2	0.4
params_gradient_clip	True	True	True
params_hidden_size	192	256	96
params_lr	0.000010	0.00027	0.00067
params_num_layers	3	1	1
params_use_residuals	False	True	True
params_use_sigmoid	False	False	False
params_batch_size	64	64	128
params_scaler	robust	robust	robust

Table C.5: Optimal hyperparameters FFNN.

Hyperparameter	C	R(F1)	R(MSE)
params_gradient_clip	True	False	True
params_hidden_size	96	32	224
params_lag_value	45	8	79
params_lr	0.00013	0.009713	0.000033
params_lstm_dropout	0.0	0.5	0.1
params_lstm_layers	2	2	3
params_scaler	standard	robust	standard
params_static_hidden_size_1	64	64	112
params_static_hidden_size_2	8	40	8
params_batch_size	64	64	64
params_optimzier	adam	adam	adam

Table C.6: Optimal hyperparameters LSTM.

Hyperparameter	C	R(F1)	R(MSE)
params_activation_concat	tanh	relu	leaky_relu
params_activation_seq1	relu	relu	tanh
params_activation_seq2	leaky_relu	tanh	leaky_relu
params_activation_static	relu	sigmoid	relu
params_ad_avg_pool1	40	64	40
params_ad_avg_pool2	64	8	96
params_branch_out_channels1	64	16	96
params_branch_out_channels2	64	96	32
params_conc_hidden	128	8	48
params_gradient_clip	True	True	True
params_kernel_size1	10	7	3
params_kernel_size2	7	3	2
params_lag_value	85	96	95
params_max_pooling_kernel1	8	3	4
params_max_pooling_kernel2	2	10	6
params_out_static	88	16	112
params_out_to_concat	48	40	56
params_padding_type1	valid	valid	same
params_padding_type2	same	same	same
params_static_hidden	40	56	8
params_stride1	1	7	3
params_stride2	3	2	1
params_optimzier	nan	nan	adam
params_lr	0.00045	0.00010	0.00050
params_scaler	minmax	standard	robust

Table C.7: Optimal hyperparameters CNN.

Hyperparameter	C	R(F1)	R(MSE)
params_lag_value	49	26	43
params_add_encoder_length	-	-	False
params_add_relative_time_idx	False	True	False
params_add_target_scales	-	-	True
params_attention_head_size	2.00	8	5
params_causal_attention	False	False	True
params_dropout	0.25	0.15	0.08
params_encoder_random_ratio	0.75	0.53	0.90
params_epochs	-	-	170
params_hidden_continuous_size	32.00	4	32
params_hidden_size	32.00	128	32
params_lstm_layers	1.00	3	2
params_max_prediction_length	19.00	8	3
params_randomize_length	True	False	True
params_reduce_on_plateau_patience	9.00	7	5
params_share_single_variable_networks	False	True	True
params_target_normalizer	-	-	encnorm
params_batch_size	-	-	128
params_learning_rate	0.00	0.01	0.00
params_scaler	encnorm	standard	standard

Table C.8: Optimal hyperparameters TFT.