Prediction of Energy Demand by Neighborhood in Barcelona for Sustainable Resource Management

Aleix Francía Albert

Abstract– Electricity is a fundamental element of contemporary society and an indicator of technological development.
The challenge of ensuring the sustainable use of resources is presented as a global priority, where optimization becomes an essential tool. According to the International Energy Agency (IEA), energy consumption in the European Union decreased by 3.2% in 2022, the second-largest drop since 2009. Although recovery is expected, negative electricity prices have become more common, reflecting market instability and the need for better management.
This study focuses on predicting electricity demand in the city of Barcelona, using artificial intelligence algorithms. These predictions and the associated uncertainty provide tools for the optimization and management of resources.

Keywords– Energy efficiency, electricity consumption, demand forecasting, artificial intelligence, market volatility, Barcelona.

• *Contact email:* afranciaa2501@gmail.com
• *Supervised by:* Albert Romero Sánchez (Department of Computer Science)

CONTENTS

1 Introduction

The growing energy demand, combined with the urgency of minimizing the carbon footprint, calls for the design of public policies that ensure the transition to a sustainable model. In light of this reality, it is essential to understand the factors that determine electricity demand, which is especially relevant as energy continues to be a scarce resource due to the impossibility of storing it efficiently.

Domestic electricity consumption has undergone a significant increase in Spain in recent decades, driven by a combination of economic, climatic, and social factors. One of the main determinants of this increase is the positive correlation between the rise in household purchasing power and the demand for electricity.

This study aims to predict energy demand in Barcelona, broken down by postal codes, using data corresponding to the period between 2019 and 2024 and the application of artificial intelligence systems. Furthermore, the uncertainty associated with these predictions is analyzed to ensure the reliability of the results obtained and to provide an additional tool for decision-making.

The analysis provides empirical evidence that facilitates the understanding of local energy dynamics, thus generating a tool for the design of adapted strategies, representing progress toward more efficient resource management without compromising the development and inherent needs of the contemporary city.

2 Dataset

The predictive model was developed based on a multidimensional dataset containing energy, meteorological, and economic information related to the city of Barcelona, covering the years 2019 to 2024. This dataset was compiled from multiple public sources, ensuring its coherence and compatibility for subsequent analysis.

- **Electricity consumption:** Obtained from the Barcelona Energy Observatory and the Barcelona Open Data portal [1, 2]. These sources provide disaggregated records by postal code and time slot.
- **Meteorological data:** Extracted via the historical weather API from Open Meteo [3]. Includes multiple variables such as temperature, wind speed, solar radiation, and sunshine duration.
- **Economic indicators:** The interannual rate of the Industrial Production Index (IPI) was incorporated, obtained from EPData in collaboration with the National Statistics Institute (INE) [4].

Table 1 provides an overview of the variables used in this study, including their descriptions and data types. The original dataset, which featured disaggregated energy consumption data across various sectors, was aggregated to represent total consumption. This aggregation was necessary to enable a comprehensive analysis of local energy usage patterns, consistent with the study's objective of forecasting energy demand.

A preprocessing stage was also carried out to improve model performance; this included the elimination of missing values, normalization of data scales, and the encoding of categorical variables.

Variable	Description	Type
Year	Year of the record.	Discrete numeric
Month	Month of the year.	Ordinal categorical
Day	Day of the month.	Ordinal categorical
Time_Slot	Time slot (1–4).	Ordinal categorical
Postal_Code	Geographical area.	Nominal categorical
Value	Target value to predict.	Continuous numeric
temperature_2m	Temperature at 2m.	Continuous numeric
apparent_temperature	Perceived temperature.	Continuous numeric
rain	Rain (yes/no).	Binary
wind_speed_10m	Wind speed.	Continuous numeric
is_day	Daytime indicator.	Binary
sunshine_duration	Minutes of sunshine.	Continuous numeric
direct_radiation	Direct solar radiation.	Continuous numeric
Day_Of_Week	Day of the week (0–6).	Ordinal categorical
Holiday	Holiday indicator.	Binary
IPI Interannual Rate	Industrial production index.	Continuous numeric
dew_point_2m	Dew point.	Continuous numeric
Dataset dimensions: 348,030 observations × 17 variables		

TABLE 1: Dataset variables with description and type.

3 EDA (*Exploratory Data Analysis*)

After completing the data preprocessing, an exploratory analysis was carried out using statistical and visual techniques to understand the nature of the phenomenon, facilitating the subsequent development of the predictive model.

The main objective is to identify patterns, trends, outliers, and correlations between the various independent variables and the dependent variable.

3.1 Energy Consumption Distribution by Postal Code

As shown in the heat map in Figure 1, energy consumption is unevenly distributed between the different postal codes within the Barcelona area. This spatial variation reflects underlying differences in population density, land use, or economic activity.

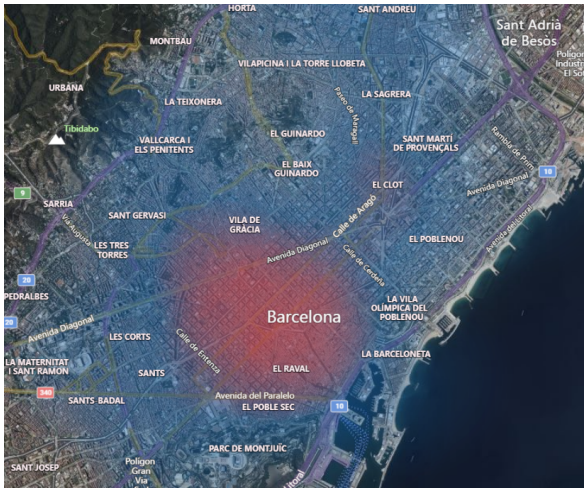

Fig. 1: Spatial Distribution of Energy Consumption in Barcelona.

3.2 Histogram and Temporal Trend Analysis

The histogram presented in Figure 2 and boxplot presented in Figure 3 reveal a general downward trend in energy consumption over the years. This pattern may be attributed to growing environmental awareness, policy interventions, and improvements in energy efficiency technologies.

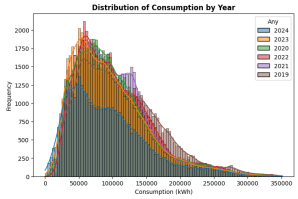

Fig. 2: Histogram showing the distribution of energy consumption values across the dataset.

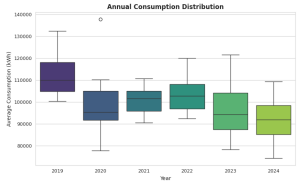

Fig. 3: Boxplot illustrating the annual trend in energy consumption.

3.3 Monthly and Daily Consumption

As shown in Figure 4, monthly energy consumption showshows a clear and recurring pattern that significantly influences the overall demand. In contrast, daily consumption in Figure 5 appears to be more variable and does not show a consistent distributional trend.

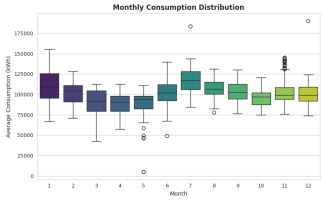

Fig. 4: Monthly Distribution.

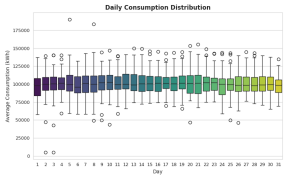

Fig. 5: Daily distribution.

3.4 Distribution by Time Slot

The time slot is an independent variable that directly influences energy demand, with this relationship primarily driven by patterns of socio-economic activity. Consumption tends to peak during the **12:00–18:00** time slot, when socio-economic activities are typically at their highest, such as work and commercial operations. In contrast, the **00:00–06:00** slot exhibits the lowest energy consumption, characterized by minimal activity and almost negligible variability. Furthermore, time slots associated with similar socio-economic conditions tend to display comparable levels of variability, as reflected in their standard deviations, suggesting consistent energy usage patterns across these periods.

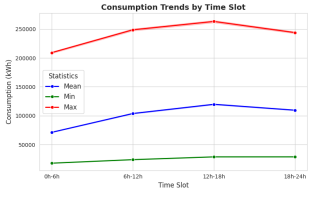

Fig. 6: Time Slot Evolution.

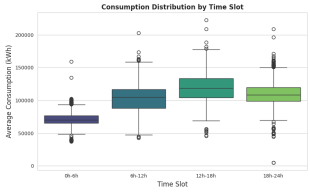

Fig. 7: Boxplot by Time Slot.

3.5 Influence of Independent Variables

To gain a comprehensive understanding of the influence of the selected independent variables on the dependent variable, it is essential to analyze their interrelationships. Correlation coefficients quantify the strength and direction of these associations, serving as critical indicators of how one variable may affect another. This analysis is instrumental in identifying significant dependencies, uncovering patterns, and detecting potential multicollinearity, thereby informing the selection of suitable models and predictive strategies.

- **Pearson**: Measures the linear relationship between two variables. The formula for Pearson's correlation coefficient (r) is:

$$r = \frac{\sum(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum(X_i - \bar{X})^2 \sum(Y_i - \bar{Y})^2}}$$

  where X_i and Y_i are the observations of the two variables, and $\bar{X}$ and $\bar{Y}$ are the means.

- **Spearman**: Measures the monotonic relationship (when two variables increase together, but not necessarily linearly). The formula for Spearman's rank correlation (ρ) is:

$$\rho = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)}$$

  where d_i is the difference in ranks of the two variables, and n is the number of observations.

- **Distance Correlation (DCOR)**: Measures all types of dependencies, both linear and non-linear. The formula is:

$$\text{DCOR}(X, Y) = \frac{\sqrt{\sum_{i,j}(d(X_i, X_j) - d(Y_i, Y_j))^2}}{n}$$

  where $d(X_i, X_j)$ and $d(Y_i, Y_j)$ represent distances between observations, and n is the number of observations.

The following heatmap, in Table 2, visualizes the correlation coefficients between the independent variables and the dependent variable:

Variable	Pearson	Spearman	Distance Correlation
Is_Day	0.205	0.213	0.907
Time_Slot	0.258	0.287	0.283
Sunshine_Duration	0.217	0.229	0.220
Direct_Radiation	0.197	0.240	0.201
Postal_Code	0.037	-0.029	0.126
Temperature_2m	0.133	0.121	0.124
Holiday	-0.054	-0.049	0.115
Wind_Speed_10m	0.102	0.117	0.109
Apparent_Temperature	0.111	0.100	0.106
Weekday	-0.111	-0.099	0.102
Rain	0.013	0.016	0.095
Year	-0.084	-0.091	0.078
Yearly_IPI_Rate	0.008	-0.007	0.053
Dew_Point_2m	0.044	0.039	0.050
Month	0.019	0.018	0.042
Day	0.001	0.001	0.006

TABLE 2: COMPARISON OF PEARSON, SPEARMAN, AND DISTANCE CORRELATION WITH THE TARGET VARIABLE

4 Random Forest Regressor

The **RandomForestRegressor** is a regression model that uses multiple decision trees generated through the **bagging** technique, where each tree is built from a random subset of the data and features. The final prediction is the average of the predictions made by the generated trees.

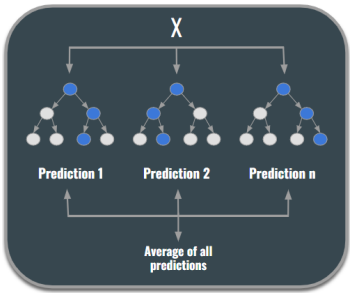

Fig. 8: RandomForestRegressor.

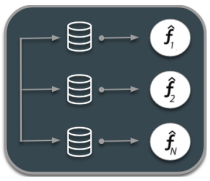

Fig. 9: Bagging.

It is a suitable model for the study, as training each tree with different subsets of the data minimizes the dependency between them, increasing the generalization capability. Furthermore, it captures non-linear relationships and effectively handles noisy data or *outliers*.

4.1 Uncertainty Quantification

The model quantifies the uncertainty in each prediction, determined by calculating the standard deviation of the set of predictions from the individual decision trees, reflecting the variability in the model's outputs and thus facilitating the development of adaptive strategies.

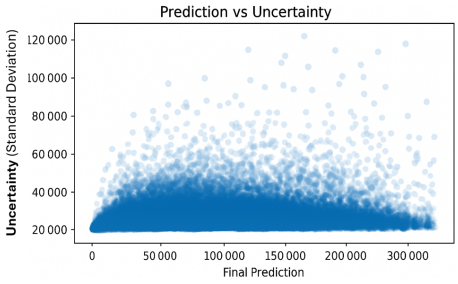

Fig. 10: Relationship between the final predictions and the associated uncertainty (standard deviation) for each observation.

5 Extreme Gradient Boosting

XGBoost[5] is a scalable and distributed machine learning technique based on gradient-boosted decision trees (**GBDT**), and it belongs to the family of *ensemble learning* models. **GBDT** models train a sequence of shallow decision trees, where each iteration uses the residual errors of the previous model. The new trees are optimized to correct the errors of the previous ones. The final prediction is a weighted sum of all the tree predictions.

Both **Random Forest** and **GBDT** build models composed of multiple decision trees, but the difference lies in how the trees are built and combined.

The term *gradient boosting* refers to the process of improving a weak model by combining it with other weak models to generate a strong one.

This method optimizes objective functions for the subsequent model, improving it through the second derivative (*Taylor expansion*) of the loss function, which makes it faster compared to traditional gradient boosting techniques.

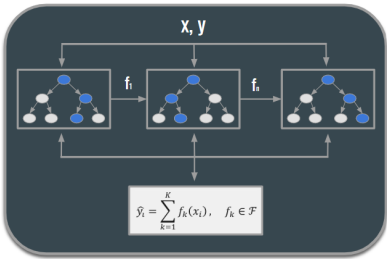

Fig. 11: XGBoost.

Fig. 12: Boosting.

5.1 Selected Hyperparameters

Hyperparameters control the learning process of a model. In the case of XGBoost, these hyperparameters determine the growth of trees and the optimization of the model.

A hyperparameter grid was defined, exploring different value combinations to find the configuration that yields the best results for our dataset. This task is performed using the **GridSearchCV**[6] optimization technique, already used previously. Additionally, regularization is applied using **Elastic Net**.

- **reg lambda**: Applies quadratic penalty to tree weights to reduce variability and prevent excessively large weights:

$$\text{L2 Regularization} = \lambda \sum_{i=1}^{n} w_i^2$$

- **reg alpha**: Applies absolute penalty to weights, promoting sparsity in the model:

$$\text{L1 Regularization} = \alpha \sum_{i=1}^{n} |w_i|$$

- **Elastic Net**: Combines L1 (Lasso) and L2 (Ridge) penalties:

$$\text{Minimize} \quad \text{Loss Function} + \lambda_1 \sum |w_i| + \lambda_2 \sum w_i^2$$

5.2 Feature Selection

With the aim of reducing the number of variables and optimizing the model's performance, the recursive feature elimination with cross-validation (RFECV) technique has been applied. This method iteratively removes the least relevant features, using cross-validation to determine the optimal number of variables that maximizes the performance metric, in this case, the coefficient of determination R^2 of the XGBoost model.

This technique is more computationally efficient than exhaustive feature selection, as it does not evaluate all possible combinations.

The selection process is detailed in the pseudocode 1 presented below. This scheme illustrates the main steps for optimizing the feature set.

Algorithm 1 Simplified RFECV pseudocode

```
Initialize the feature set F.
Initialize the best score best_score ← 0
while the stopping criterion is not met do
    for each fold in cross-validation do
        Train the model with features F on the training data
        Evaluate the model on the validation data
    end for
    Compute the average score score
    if score > best_score then
        best_score ← score
        Save the current set F as the best set
    end if
    Remove the least important features from F
end while
Return the best feature set. =0
```

Three XGBoost configurations have been evaluated to measure the impact of optimization: the base model, the model tuned with GridSearchCV, and the model combined with GridSearchCV and RFECV feature selection. Table 3 shows the results obtained.

Configuration	**MAPE** (0-100) ↓	R^2 (0-1) ↑
XGBoost (no optimization)	2.50	0.94
XGBoost with GridSearchCV	**1.71**	**0.97**
XGBoost with GridSearchCV + RFECV	1.82	0.97

TABLE 3: COMPARISON OF METRICS FOR THE XGBOOST MODEL UNDER DIFFERENT CONFIGURATIONS.

This feature selection process has been implemented using the `scikit-learn` library, which provides efficient and widely recognized tools, among them recursive feature elimination (RFECV). By using RFECV, the most relevant features have been systematically identified, progressively removing the less informative variables during the cross-validation process. This methodology has not improved the model's accuracy but has reduced its complexity. Therefore, it will not be used in subsequent sections.

5.3 Monte Carlo Ensemble

The *Monte Carlo* method refers to a class of computational algorithms that use repeated random sampling to solve complex deterministic tasks with probabilistic techniques. The name *Monte Carlo* comes from the famous casino in Monaco and was popularized by Nicholas Metropolis and Stanislaw Ulam in their 1949 work [7, 8].

A fundamental principle supporting this methodology is the *Law of Large Numbers*, which states that as the number of samples increases, the sample mean approaches the expected value. This principle is essential for the Monte Carlo method, as it justifies the use of a large number of simulations.

Figures 13 and 14 clearly show how, as the sample size increases, the resulting distribution increasingly fits the true normal distribution.

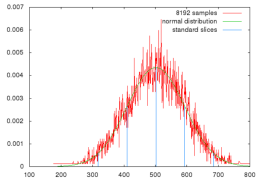

Fig. 13: Normal distribution with ∼8,000 samples [9].

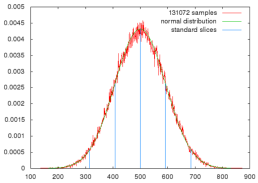

Fig. 14: Normal distribution with ∼131,000 samples [9].

In this study, the **Monte Carlo** method is applied as a statistical technique to estimate probability distributions through an *ensemble* of XGBoost models trained with different random seeds (`random_state`). This procedure, described in the pseudocode in Section **??**, generates a distribution of predictions per input. The mean of these predictions provides the point estimate, while their standard deviation allows quantifying the associated uncertainty, reflecting the model's variability and the reliability of the predictions.

- **Point Prediction**:$\hat{y} = \frac{1}{N}\sum_{i=1}^{N} \hat{y}_i$
- **Uncertainty (Standard Deviation)**:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - \hat{y})^2}$$

- **Confidence Interval**: $\text{CI}_{95\%} = [\hat{y}_{2.5\%}, \hat{y}_{97.5\%}]$

In this study, the **Monte Carlo** method is applied as a statistical technique to estimate probability distributions through an *ensemble* of XGBoost models trained with different random seeds (`random_state`).

This procedure, described in the pseudocode in Section **??**, generates a distribution of predictions per input. The mean of these predictions provides the point estimate, while their standard deviation allows quantifying the associated uncertainty. This approach facilitates a more robust analysis in contexts where uncertainty is a critical factor. Results of **1.74** MAPE and **0.97** R^2 have been achieved on the test set.

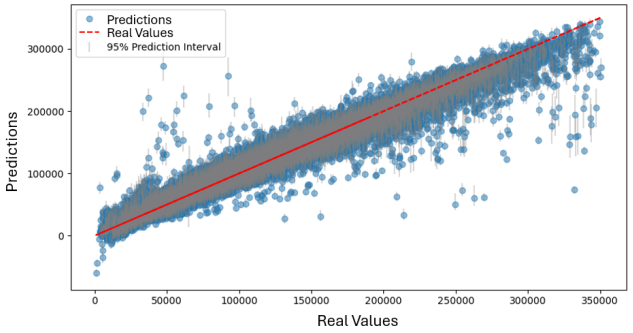

Fig. 15: Monte Carlo ensemble predictions with 95% confidence intervals.

5.4 Bootstrap Ensemble

Bagging (*Bootstrap Aggregating* [10] is a statistical technique that builds multiple models from random subsets of the original data. The objective of this technique is to quantify the uncertainty associated with the predictions.

From a training set D of size n, m new datasets D_i of size n' are generated through random sampling, with or without replacement. For $n' = n$ and a large n, it is expected that approximately $(1 - 1/e) \approx 63.2\%$ of the samples will be unique, while the rest will be duplicates—this is the *bootstrap sample*[11].

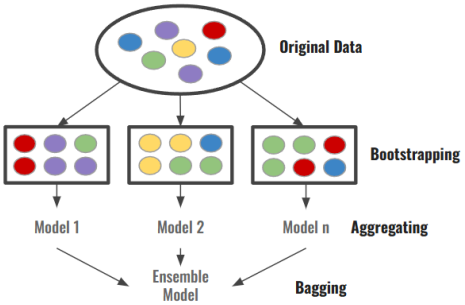

Fig. 16: Bootstrap aggregating illustration.

Each bootstrap set is used to train an independent model. For prediction, the results of all models are combined, obtaining an aggregated estimate. This allows obtaining both a point prediction and an uncertainty estimate through the distribution of the models' results, similarly to the **Monte Carlo Ensemble** described above.

This technique enables the construction of confidence intervals to reflect the modeled uncertainty, providing a framework for decision-making based on uncertainty. Results of **1.8** MAPE and **0.94** R^2 have been achieved on the test set.

6 Neural Network

An artificial neural network[12] is a learning and automatic processing paradigm inspired by the way the nervous system of animals works. It is a system of interconnected neurons in a network that collaborate to produce an output stimulus. This network of interconnected artificial neurons uses a computational model inspired by biological neural systems, where neurons are connected by weighted links and use nonlinear activation functions to process information. Following the connectionist approach, the network learns complex patterns by adjusting these weights through training algorithms like **backpropagation**.

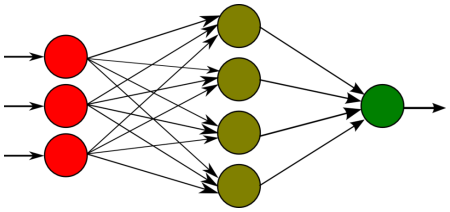

Fig. 17: **Artificial Neural Network**. A simple perceptron with 3 input neurons, 4 hidden-layer neurons, and 1 output neuron [12].

The models have been generated using a base model shown in Table 4, adapted from the work presented in [13]. The model has been fine-tuned to fit the specific characteristics of the dataset used in this study. To better capture the temporal patterns present in the data, *Long Short-Term Memory (LSTM)* units [14] were incorporated into the architecture.

Layer	Input Shape	Output Shape	Param #
InputLayer	(None, 15, 1)	(None, 15, 1)	0
LSTM	(None, 15, 1)	(None, 15, 64)	16,896
GlobalAvgPool1D	(None, 15, 64)	(None, 64)	0
Dense(ReLU)	(None, 64)	(None, 256)	16,640
BatchNorm	(None, 256)	(None, 256)	1,024
Dropout(0.19)	(None, 256)	(None, 256)	0
Dense(ReLU)	(None, 256)	(None, 32)	8,224
BatchNorm	(None, 32)	(None, 32)	128
Dropout(0.5)	(None, 32)	(None, 32)	0
Dense(Linear)	(None, 32)	(None, 1)	33

TABLE 4: Reference Neural Network.

6.1 Activation

The activation transforms a neuron's input into an output through an activation function, introducing nonlinearity into the neural network. For the case study, several variants of the ReLU function will be used.

- **ReLU (Rectified Linear Unit):**

$$f(x) = \max(0, x)$$

- **LeakyReLU (Leaky Rectified Linear Unit):**

$$f(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha x, & \text{if } x < 0 \end{cases}$$

- **ELU (Exponential Linear Unit):**

$$f(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha(e^x - 1), & \text{if } x < 0 \end{cases}$$

As shown in the Table 5, based on the study [15], each activation function exhibits different behavior depending on the dataset used.

Accuracy results comparison over CIFAR100 dataset					
Activation	**MobileNet**	**GoogleNet**	**ResNet50**	**SENet18**	**DenseNet121**
ELU	61.97 ± 0.24	72.57 ± 1.76	71.41 ± 1.63	71.27 ± 0.58	72.06 ± 1.93
ReLU	61.33 ± 0.34	74.05 ± 1.69	71.96 ± 0.94	70.45 ± 0.73	72.99 ± 1.35
LReLU	61.13 ± 0.41	63.79 ± 2.38	72.77 ± 0.49	70.58 ± 0.45	73.33 ± 1.25
Training time comparison over CIFAR100 dataset					
Activation	**MobileNet**	**GoogleNet**	**ResNet50**	**SENet18**	**DenseNet121**
ELU	00:31:05	04:57:37	03:36:47	01:13:25	04:08:39
ReLU	00:31:22	04:55:10	03:32:30	01:15:33	04:15:06
LReLU	00:31:48	05:01:30	03:33:00	01:18:38	04:14:09

TABLE 5: Comparison over CIFAR100 adapted from "Activation Functions in Deep Learning: A Comprehensive Survey and Benchmark" [15].

6.2 Long Short-Term Memory (LSTM)

Initially, the model was trained and evaluated using the *ReLU* (Rectified Linear Unit) activation function, known for its computational simplicity and efficiency in gradient propagation. Moreover, *ReLU* accelerates training time compared to other alternatives such as the sigmoid or tanh functions, due to its simpler behavior during backpropagation. At the same time, long-term memory was incorporated through *LSTM* (Long Short-Term Memory), a specific architecture within recurrent neural networks (*RNN*), designed to overcome the inherent limitations of traditional RNNs that often struggle to model long-term dependencies.

LSTMs were developed to address the difficulties of classical *RNNs*, which have trouble retaining relevant information over long time periods due to the phenomenon of vanishing or exploding gradients. Unlike conventional RNNs, *LSTMs* introduce a gated structure (*input*, *forget*, and *output gates*) that regulates the flow of information through the network, allowing important information to be retained or discarded according to its temporal relevance.

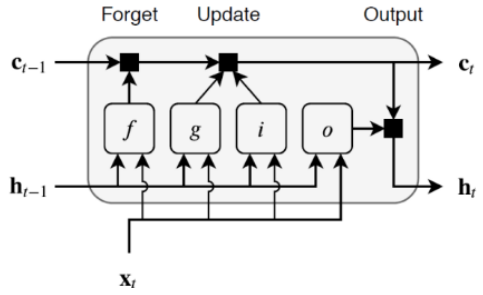

Fig. 18: LSTM.

This architecture facilitates learning complex and temporally long sequences, improving model performance in tasks such as pattern recognition or time series prediction.

LSTMs operate through the following gates that control the flow of information:

- **Forget gate**: Decides which information in the memory is kept or discarded, with the gate value being a score between 0 (information is discarded) and 1 (information is kept).

- **Input gate**: Determines which new information is added to the memory by combining the current input with the previous state. The input gate uses an activation function (usually a **sigmoid**) to control the update.

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

- **Output gate**: Determines which information is passed to the next layer or as output, also using the **sigmoid** activation.

The traditional **LSTM** processes the input sequence in a single direction (usually left to right). The network state at each time step depends on the current input and the previous state.

6.3 Bidirectional LSTM

The **Bidirectional LSTM** consists of two parallel input pathways, each processing the sequence in opposite directions. This structure allows the model to obtain a more comprehensive view of the input sequence.

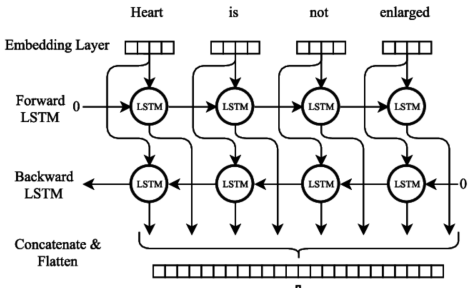

Fig. 19: Bidirectional LSTM.

To evaluate the impact of bidirectional LSTM architectures compared to traditional ones, both a **traditional LSTM** model and a bidirectional structure model will be generated for each type of activation mentioned previously. All models will maintain the same initial architecture, allowing the effect of directionality on model performance to be directly compared.

6.4 Evaluation Metrics

The metrics used in this study to assess how suitable the model and activation function are for our task are: **MSE** as the loss function, and **MAPE** and **R²** as evaluation metrics.

Here is the English translation with LaTeX formatting preserved:

- **MSE** (Mean Squared Error): the mean of the squared differences between the predicted and actual values. It penalizes larger errors more heavily. This is the metric used as the loss function during the training of the different models.

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

- **MAPE** (Mean Absolute Percentage Error): the mean of the absolute relative errors expressed as a percentage. It is a widely recognized metric to quantify the deviation between predictions and actual values, especially valued for its easy interpretability.

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \times 100$$

- **R²** (Coefficient of Determination): represents the proportion of variance in the dependent variable explained by the model relative to the total observed variance. It is interpreted as a measure of how well the model fits the actual values. A value close to 1 indicates high explanatory power, while values near 0 indicate a very limited explanation of variability.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

6.5 Results

As can be observed in Table 6, models utilizing *LSTM* generally achieve better results with the same activation function. The best-performing model on the test set is the traditional *LSTM* with *ELU* activation. Although *ELU* is the most computationally expensive activation function, it stood out for its favorable performance-computation trade-off compared to other *ReLU*-based activations, followed by *LeakyReLU*, and lastly *ReLU*, which is the least costly.

Model	**MAPE** (0-100) ↓	R^2 (0-1) ↑
LSTM ELU	**1.7424**	**0.9683**
BI LSTM ELU	1.7919	0.9671
BI LSTM LEAKY	1.8283	0.9659
LSTM LEAKY	1.8567	0.9647
LSTM RELU	2.0296	0.9591
BI LSTM RELU	2.2039	0.9550

TABLE 6: METRICS OF DIFFERENT MODELS, SORTED BY INCREASING MAPE.

Regarding convergence during training, the loss curve for all models exhibits a smooth and consistent downward trend. This gradual decrease in loss demonstrates that the models are progressively improving their fit to the training data without abrupt fluctuations that could suggest instability or divergence. The detailed progression of the validation loss throughout the training epochs for each model is illustrated in Figure 20, providing a clear visual comparison of their convergence behavior and generalization capability over time.

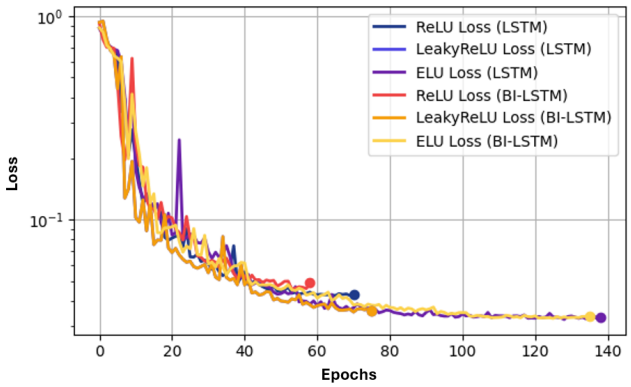

Fig. 20: Loss of different models during training, for the validation set.

Despite the complexity of the baseline model, it achieves early convergence within the first epochs. Strong regularization has been applied, thus preventing overfitting.

This encourages the model to learn the underlying temporal patterns rather than memorizing the training data. This improves overall performance but also causes a slower convergence rate during the later stages of training.

Regarding the test data, all models performed better than during training, showing metrics very similar to those of the validation set. This is attributed to strong regularization techniques, such as the use of *Dropout* layers and *Batch Normalization*.

6.6 Effect of Batch Normalization

Batch Normalization [16] is a technique introduced by Sergey Ioffe and Christian Szegedy in 2015 to improve the training process of neural networks and prevent *gradient explosion* [17], a phenomenon that occurs when error gradients become excessively large, causing disproportionate updates to model weights during training.

This method normalizes the activations of layers so that they have a mean close to zero and a standard deviation close to one. This normalization reduces the internal covariate shift and accelerates the model's convergence [18].

- **During training:** the layer normalizes its output using the mean and standard deviation of the current input *batch*. For each channel being normalized, the layer returns:

$$\gamma \cdot \frac{\text{batch} - \text{mean}(\text{batch})}{\sqrt{\text{var}(\text{batch}) + \epsilon}} + \beta$$

  where:

  - ϵ is a small constant to prevent division by zero,
  - γ is a learned scaling factor (initialized to 1),
  - β is a learned shifting factor (initialized to 0).

- **During validation and testing:** the layer normalizes its output using a moving average of the mean and variance collected during training, providing stability and better generalization. It computes:

$$\gamma \cdot \frac{\text{batch} - \text{self.moving\_mean}}{\sqrt{\text{self.moving\_var} + \epsilon}} + \beta$$

  where **self.moving_mean** and **self.moving_var** are non-trainable variables updated iteratively during training.

It is important to note that normalization is only applied during inference after the layer has been trained with data having statistics similar to those observed during training.

7 Effect of Dropout

Dropout [19] is a regularization technique that randomly deactivates a percentage of neurons during each *forward pass* in training. This process forces the model to avoid excessive reliance on specific neurons, promoting redundancy and robustness in the learned representations. By preventing co-adaptation among neurons, Dropout reduces the risk of overfitting and encourages the network to generalize better to unseen data. Consequently, models trained with Dropout often achieve improved performance and stability across different datasets.

- **During training:** The model is trained with a "thinned" architecture, reducing its capacity and increasing the loss during this phase.

- **During validation/testing:** Dropout is not applied, and all neurons remain active, allowing the model to operate at its full capacity.

7.1 Uncertainty Estimation Using Monte Carlo Dropout

To quantify the uncertainty of the model's predictions, the **Monte Carlo Dropout** technique is applied. This approach approximates Bayesian inference by keeping Dropout layers active during both training and inference, as introduced in [20].

Given input samples $\mathbf{X} = \{x_i\}_{i=1}^{N}$ with labels $\mathbf{Y} = \{y_i\}_{i=1}^{N}$, a Dropout-regularized neural network approximates a function $\hat{f}(x;\theta)$, where θ are the learned parameters. Under standard inference, predictions are computed as $\hat{y} = \hat{f}(x;\theta)$

With Dropout enabled at inference, multiple stochastic forward passes are performed, generating a distribution of outputs, allowing the model to estimate uncertainty by capturing prediction variability.

The following outlines the full algorithmic procedure for Monte Carlo Dropout:

Algorithm 2 Monte Carlo Dropout with Neural Networks

Require: Neural network model with Dropout (`model`), input data X, number of simulations T

Ensure: Predictive mean and standard deviation for each input

1: Initialize an empty list `f_preds`
2: **for** $t = 1$ to T **do**
3: **// Enable Dropout during inference**
4: Generate predictions: `preds = model(X, training=True)`
5: Append `preds` to `f_preds`
6: **end for**
7: Convert `f_preds` to a matrix
8: Compute mean and standard deviation across T predictions
9: **return** `mean_preds`, `std_preds` =0

Figures 21 and 22 visualize the standard Dropout and Monte Carlo Dropout inference modes.

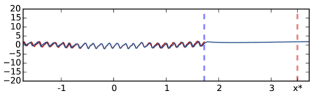

Fig. 21: Standard Dropout with mean weight [20].

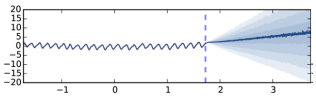

Fig. 22: MC Dropout with ReLU nonlinearities [20].

8 Conclusions

The systematic construction and evaluation of various predictive models using a consistent set of quantitative metrics has enabled an objective comparison of their performance. Based solely on these numerical results, the models that demonstrated the highest levels of predictive accuracy and reliability were: the Long Short-Term Memory (LSTM) network with Exponential Linear Unit (ELU) activation combined with Monte Carlo Dropout, the XGBoost model with Monte Carlo Ensemble, and the RandomForestRegressor. These models consistently outperformed the others in terms of error reduction and generalization capabilities, as shown in Table 7.

Model	MAPE	R^2
LSTM Models		
LSTM ELU	1.7424	**0.9683**
LSTM ELU + Monte Carlo Dropout	1.7648	**0.9678**
LSTM LEAKY	1.8567	0.9647
LSTM RELU	2.0296	0.9591
Bidirectional LSTM Models		
BI LSTM ELU	1.7919	0.9671
BI LSTM LEAKY	1.8283	0.9659
BI LSTM RELU	2.2039	0.9550
XGBoost Models		
XGBoost using a Monte Carlo Ensemble	1.7417	0.9665
XGBoost using a Bootstrap Ensemble	1.8020	0.9647
Random Forest Model		
RandomForestRegressor	**1.5753**	0.9646

TABLE 7: Results of different models.

Although all the implemented models incorporate effective regularization mechanisms, the **LSTM ELU** with **Monte Carlo Dropout** has demonstrated consistently robust performance. This is reflected in the error distribution obtained through Monte Carlo Dropout and shown in Figure 23, where the LSTM maintains high resilience to variations in the input data.

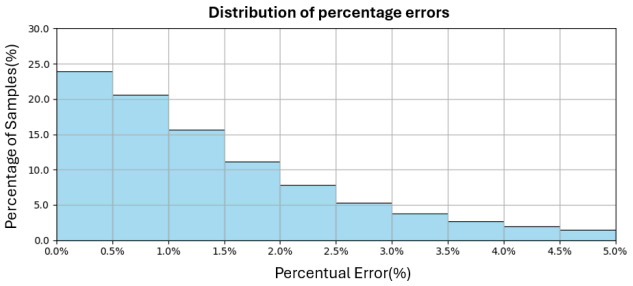

Fig. 23: Normalized percentage error distribution for the LSTM ELU model with Monte Carlo Dropout.

While the **LSTM ELU** with **Monte Carlo Dropout** stands out for its capacity to model the phenomenon, it is worth noting that the **Random Forest** outperformed it in terms of overall performance, as shown in Table 7 and in the comparison of percentage error distributions in Figure 24. It offers a solution with significantly lower computational complexity.

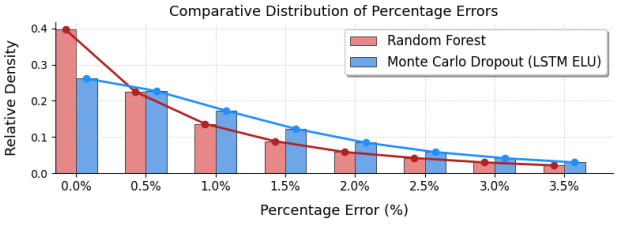

Fig. 24: Comparison of the normalized percentage error distributions between the Monte Carlo Dropout (LSTM ELU) and Random Forest model.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to all the people who have accompanied and supported me unconditionally throughout this process, allowing me to dedicate quality time to research. In particular, I thank Albert Romero Sánchez and Xavier Miquel Armengol for their dedication in introducing me to this field, and I especially acknowledge the constant effort and guidance of Albert Romero during the supervision of this work.

REFERENCES

[1] A. de Barcelona, "L'observatori de l'energia de barcelona."

[2] ——, "Consum d'electricitat a barcelona open data."

[3] O. Meteo, "Weather data api archive."

[4] EPData, "Índex de producció industrial a catalunya (ine)."

[5] nvidia, "What is xgboost?" 2025.

[6] P. Banerjee, "Xgboost hyperparameters tuning," 2020.

[7] W. contributors, "Nicholas metropolis — wikipedia, the free encyclopedia," 2025.

[8] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," Los Alamos Scientific Lab., Los Alamos, NM (United States); Univ. of Chicago, IL (United States), Tech. Rep., 03 1953.

[9] Wikipedia contributors, "Monte carlo method," 2024.

[10] ——, "Bootstrap aggregating — Wikipedia, The Free Encyclopedia," 2024.

[11] ——, "Bootstrapping (statistics) — Wikipedia, The Free Encyclopedia," 2024.

[12] Wikipedia, "Xarxa neuronal artificial," 2025.

[13] D. Zholtayev, D. Dauletiya, A. Tileukulova, D. Akimbay, M. Nursultan, Y. Bushanov, A. Kuzdeuov, and A. Yeshmukhametov, "Smart pipe inspection robot with in-chassis motor actuation design and integrated ai-powered defect detection system," *IEEE Access*, vol. 12, pp. 119 520–119 534, 2024.

[14] GeeksforGeeks, "What is lstm – long short term memory?" 2025.

[15] S. R. Dubey, S. K. Singh, and B. B. Chaudhuri, "Activation functions in deep learning: A comprehensive survey and benchmark," *Neurocomputing*, vol. 503, pp. 92–108, 2022.

[16] K. Team, "Batch normalization layer - keras api documentation," 2024.

[17] GeeksforGeeks, "Vanishing and exploding gradients problems in deep learning," 2020, accessed: 2025-04-30.

[18] DeepAI, "Batch normalization," 2024.

[19] K. Team, "Dropout layer - keras api documentation," 2024.

[20] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *International Conference on Machine Learning (ICML)*, 2016, pp. 1050–1059.

[21] Àrea Metropolitana de Barcelona, "Mapa energètic."

[22] G. James, D. Witten, T. Hastie, R. Tibshirani, and J. Taylor, *An Introduction to Statistical Learning: with Applications in Python*. Springer, 2023.

[23] Laura Melgar García, José Francisco Torres Maldonado, Alicia Troncoso, and José Cristóbal Riquelme Santos, "Técnicas big data para la predicción de la demanda y precio eléctrico," *Economia Industrial*, 2024.

[24] N. I. M. S. N. IMS), "The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset," *ScienceDirect*, 2020.

[25] S. M. Z. Z. Arjun Ghosh, Nanda Dulal Jana, "Designing optimal convolutional neural network architecture using differential evolution algorithm," *CellPress*, 2022.

[26] Keras Documentation, "Reduce learning rate on plateau callback," 2025.

[27] R. Tavenard, *Deep Learning Basics (lecture notes)*, 2023.

[28] P. Li, "Optimization algorithms for deep learning," 2017.

[29] S. M. Z. Z. Arjun Ghosh, Nanda Dulal Jana, "Energy consumption prediction using modified deep cnn-bi lstm with attention mechanism," *Heliyon*, 2025.

[30] xgboosting, "Estimating the uncertainty or confidence of an xgboost model," 2024.

[31] J. Brownlee, "How to tune the number and size of decision trees with xgboost in python," 2020.

[32] GeeksforGeeks, "Confidence intervals for xgboost," 2024.

[33] M. G.-T. F. M.- A. T. Federico Divina, José Francisco Torres Maldonado, "Spanish short-term electric energy consumption forecasting," 2020.

[34] L. Yang, H. Ma, Y. Zhang, and S. Li, "Research on energy consumption prediction of public buildings based on improved support vector machine," in *2023 35th Chinese Control and Decision Conference (CCDC)*, 2023, pp. 2699–2704.

[35] J. Li, X. Jiang, L. Shao, H. Liu, C. Chen, G. Wang, and D. Du, "Energy consumption data prediction analysis based on eemd-arma model," in *2020 IEEE International Conference on Mechatronics and Automation (ICMA)*, 2020, pp. 1338–1342.

[36] M. A. N. M. Asri, N. Zaini, and M. F. A. Latip, "Development of an lstm-based model for energy consumption prediction with data pre-analysis," in *2021 11th IEEE International Conference on Control System, Computing and Engineering (ICCSCE)*, 2021, pp. 228–233.

[37] K. Yang, X. Yang, L. Zhou, L. Li, and X. Wang, "Regional energy prediction model based on cdc model," in *2022 IEEE 5th Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, vol. 5, 2022, pp. 10–14.

[38] N. I. N. Komori, N. Zaini, and M. F. A. Latip, "Classification and profiling of electricity consumption patterns using bayesian networks," in *2022 IEEE Symposium on Industrial Electronics Applications (ISIEA)*, 2022, pp. 1–6.

[39] J. Brownlee, "Time series forecasting with lstms," 2020.

[40] P. Kontschieder, M. Fiterau, A. Criminisi, and S. R. Bulò, "Deep neural decision forests," in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1467–1475.

[41] K. Amasyali and N. M. El-Gohary, "A review of data-driven building energy consumption prediction studies," *Renewable and Sustainable Energy Reviews*, vol. 81, pp. 1192–1205, 2018.

[42] F. S. Lars Siemer and D. Kleinhans, "Cost-optimal operation of energy storage units: Benefits of a problem-specific approach," *Journal of Energy Storage 6*, 2016.

[43] K. Biel and C. H. Glock, "Systematic literature review of decision support models for energy-efficient production planning," *Computers Industrial Engineering*, vol. 101, pp. 243–259, 2016.

[44] Y. Chen, B. shen, A. Ali, and S. reyes, "Econometric analysis of factors influencing electricity consumption in spain: Implications for policy and pricing strategies," *Heliyon*, vol. 10, no. 17, p. e36217, 2024.

[45] AFAcoding, "Energy demand prediction for neighborhoods in barcelona for sustainable resource management," 2025. [Online]. Available: https://github.com/AFAcoding/Energy-Demand-Prediction

[46] S. Raschka, "mlxtend: Machine Learning Extensions," https://rasbt.github.io/mlxtend/, 2025.

APÈNDIX

A.1 Code

For more details on the analysis process, the GitHub repository [45] provides the complete code.

A.2 Pseudocode of the Different Algorithms

Algorithm 3 Monte Carlo Ensemble with XGBoost

Require: Training data (X, y), number of models n
Ensure: List of trained XGBoost models
Initialize an empty list `models`
for $i = 1$ to n **do**
  // Generate one XGBoost model in each iteration
  Set the random seed to i **// Ensures diversity**
  Initialize an XGBoost model.
  Train the model with (X, y)
  Add the trained model to the list `models`
end for
return `models` =0

Algorithm 4 Bootstrap Ensemble with XGBoost

Require: Training data (X, y), number of models n
Ensure: List of XGBoost models trained with bootstrap samples
Initialize an empty list `models`
for $i = 1$ to n **do**
  // Generate one XGBoost model per sample
  Generate a random sample with replacement of the indices of X **(bootstrap)**
  Build $X_{\text{boot}}, y_{\text{boot}}$ with the selected samples
  Initialize an XGBoost model.
  Train the model with $(X_{\text{boot}}, y_{\text{boot}})$
  Add the trained model to the list `models`
end for
return `models` =0

Algorithm 5 Monte Carlo Dropout with Neural Networks

Require: Neural network model with Dropout (`model`), input data X, number of simulations n
Ensure: Mean and standard deviation of predictions
Initialize an empty list `f_preds`
for $i = 1$ to n **do**
  // Repeat inference with Dropout enabled
  Generate predictions from the model: `preds = model(X, training=True)`
  Convert `preds` to `numpy` and append to `f_preds`
end for
Convert `f_preds` to a `numpy` matrix
Compute the mean of the predictions per sample
Compute the standard deviation per sample
return `mean_preds, std_preds` =0