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.

1 Introduction

The growing energy demand, combined with the need of minimizing the carbon footprint, encourages the design of public policies. In this context, it is essential to understand the factors that determine electricity demand, especially as energy remains a scarce resource due to the inefficiency of storage methods.

Residential electricity consumption in Spain has significantly increased 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 electricity demand.

This study aims to predict energy demand in Barcelona, disaggregated by postal code, using data from the period 2019 to 2024 and artificial intelligence systems. It also analyzes the uncertainty associated with these predictions to ensure the reliability and provide an additional tool for decision-making.

The analysis offers empirical evidence that helps understand local energy dynamics, providing a tool for designing tailored strategies. This represents a step toward more efficient resource management without compromising development and the inherent needs of a contemporary city.

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

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.

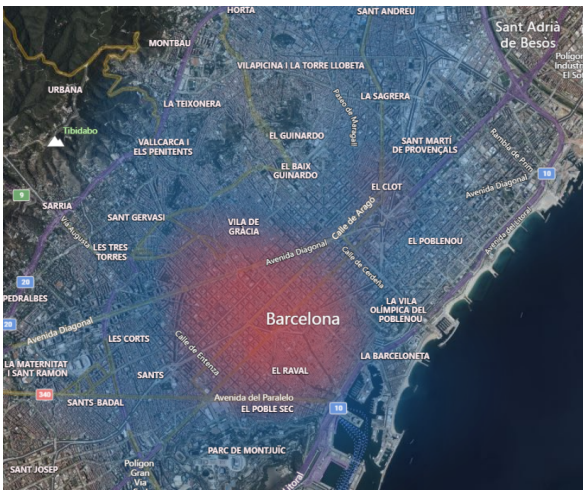

Fig. 1: Spatial Distribution of Energy Consumption in Barcelona.

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.

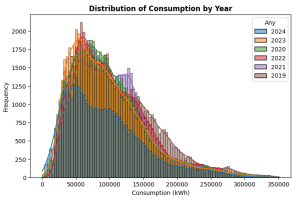

Fig. 2: Histogram showing the distribution of energy consumption values across the dataset.

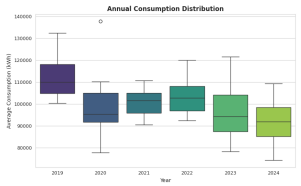

Fig. 3: Boxplot illustrating the annual trend in energy consumption.

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.

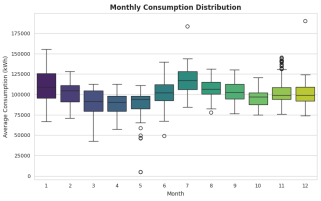

Fig. 4: Monthly Distribution.

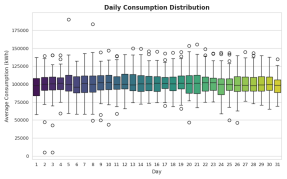

Fig. 5: Daily distribution.

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.

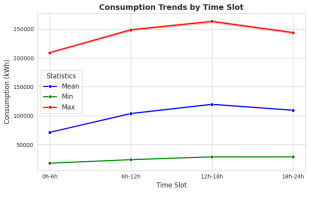

Fig. 6: Time Slot Evolution.

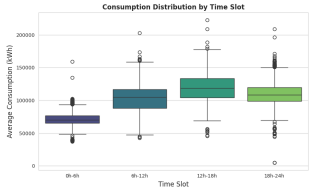

Fig. 7: Boxplot by Time Slot.

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 leverages an ensemble learning technique, utilizing a collection of decision trees. This model applies the **bagging** (Bootstrap Aggregating) technique, which involves constructing multiple decision trees in parallel, each trained on a random subset of the data and features. This approach helps to reduce variance and prevent overfitting, as it introduces diversity within the trees, making the model more generalizable.

During the **bagging** process, each decision tree is built independently, using a random subset of both the input data and the feature space. Once all the trees are trained, the final prediction is obtained by averaging the individual predictions from each tree, thereby leveraging the strength of the ensemble to produce a more accurate and stable outcome.

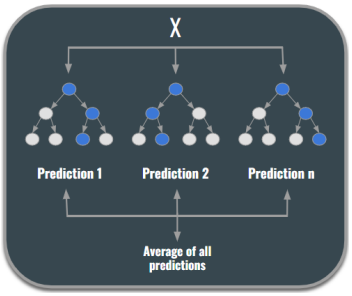

Fig. 8: RandomForestRegressor.

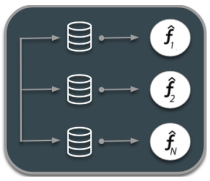

Fig. 9: Bagging.

4.1 Uncertainty Quantification

The model quantifies uncertainty in each prediction, aiding the development of adaptive strategies. This uncertainty is determined by calculating the standard deviation of predictions from individual decision trees, reflecting variability in model outputs.

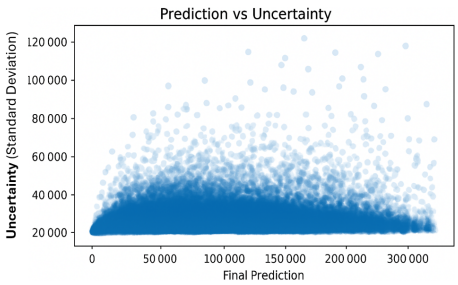

Fig. 10: Uncertainty Across Prediction Magnitudes.

5 EXTREME GRADIENT BOOSTING

XGBoost[5] is a scalable and distributed machine learning technique based on gradient-boosted decision trees (**GBDT**), which falls under the ensemble learning model family. **GBDT** trains a sequence of shallow decision trees, where each iteration uses the residuals 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 tree predictions.

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

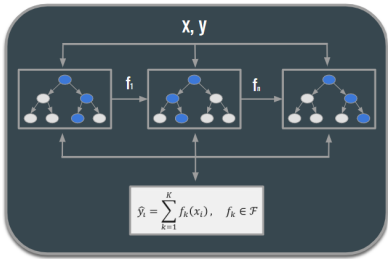

Fig. 11: XGBoost.

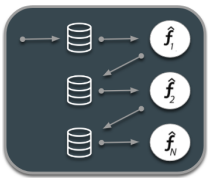

Fig. 12: Boosting.

The term *gradient boosting* refers to the concept of boosting or improving a single weak model by combining it with a series of other weak models to generate a strong collective model. Gradient boosting optimizes objective functions for the next model, it is improved using the second derivative (Taylor Expansion) on the loss function, making it significantly faster compared to traditional gradient boosting techniques.

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 model performance, recursive feature elimination with cross-validation (RFECV) has been applied. This method iteratively removes the least relevant features, using cross-validation to determine the optimal number of variables that maximize 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 but progressively eliminates features until the most suitable subset is found. Moreover, RFECV integrates feature selection directly within the model validation process, ensuring a robust choice of variables.

To evaluate the impact of optimization techniques on model performance, three configurations of the XGBoost model were considered. These include the baseline model without optimization, a version with parameter tuning via GridSearchCV, and a third configuration combining GridSearchCV with recursive feature selection using RFECV.

Table 3 summarizes the results obtained for each case.

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 was implemented using the `scikit-learn` library, which provides tools, including recursive feature elimination with cross-validation (RFECV).

By using RFECV, the most relevant features were systematically identified, progressively eliminating the least informative variables. This methodology did not improve the model's accuracy, but it did reduce 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 rely on repeated random sampling to obtain numerical results. These methods are particularly effective for solving deterministic problems with complex formulations or uncertain data inputs by leveraging probabilistic techniques. The name *Monte Carlo*—inspired by the famous casino in Monaco—was popularized by Nicholas Metropolis and Stanislaw Ulam in their seminal 1949 work [7, 8], reflecting the stochastic nature of the approach.

A fundamental principle that supports this methodology is the *Law of Large Numbers*, which states that as the number of samples increases, the sample mean converges to the expected value. This is visually illustrated in Figures 13 and 14, where increasing the sample size yields a distribution closer to the true normal distribution.

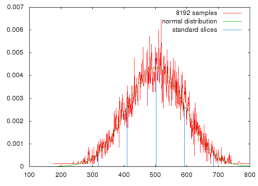

Fig. 13: Normal distribution with ∼8,000 samples [9].

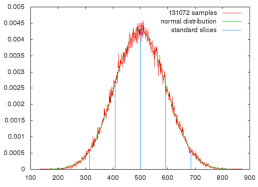

Fig. 14: Normal distribution with ∼131,000 samples [9].

In this study, the **Monte Carlo method** is applied as a statistical tool to estimate probability distributions through ensemble modeling. Specifically, multiple XGBoost regression models are trained with different random seeds, producing a distribution of predictions for each input rather than a single deterministic output. This set of models forms a *Monte Carlo ensemble*, where randomness is introduced via the `random_state` parameter to ensure distinct initialization for each model.

This ensemble approach allows the computation of both a central point estimate and the associated uncertainty. The mean of the ensemble predictions provides the point estimate, while the standard deviation quantifies uncertainty. Additionally, percentile-based confidence intervals can be derived to capture prediction reliability:

- **Point Prediction**:

$$\hat{y} = \frac{1}{N} \sum_{i=1}^{N} \hat{y}_i$$

  where $\hat{y}$ is the ensemble mean prediction, $\hat{y}_i$ is the prediction from the i-th model in the ensemble, and N is the total number of models.

- **Uncertainty (Standard Deviation)**:

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

- **95% Confidence Interval**:

$$\text{CI}_{95\%} = [\hat{y}_{2.5\%}, \hat{y}_{97.5\%}]$$

As illustrated in Figure 15, the implementation of Monte Carlo ensembles enables the generation of predictive outputs accompanied by 95% confidence intervals. This approach not only provides a central estimate but also quantifies the associated uncertainty, thereby offering a comprehensive and statistically grounded framework for robust, uncertainty-aware forecasting. By incorporating variability across multiple model realizations, this method enhances the interpretability and reliability of the predictions, which is particularly valuable in data-driven decision-making contexts.

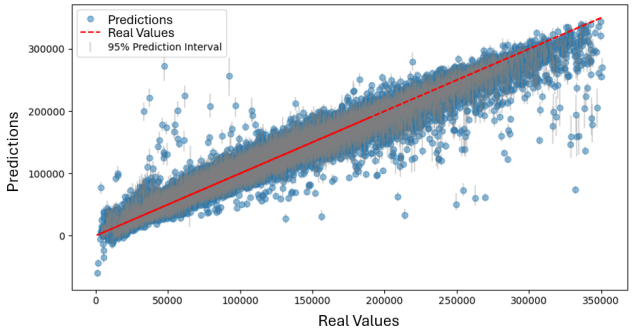

Fig. 15: Monte Carlo ensemble predictions with 95% confidence intervals.

5.4 Bootstrap Ensemble

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

Given a training set D of size n, m new datasets D_i of size n' are generated via random sampling, possibly with replacement. For $n' = n$ and large n, about $(1 - 1/e) \approx 63.2\%$ of the samples are expected to be unique; the rest are duplicates—this is called the *bootstrap sample*[11].

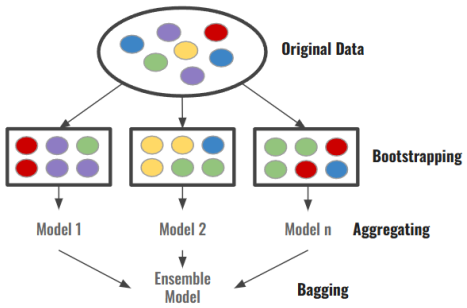

Fig. 16: Bootstrap aggregating illustration.

Each bootstrap set is used to train an independent model. For prediction, the outputs of all models are combined to obtain an aggregated estimate, enabling both a point prediction and an uncertainty estimation through the distribution of model outputs, in a manner similar to the previously described **Monte Carlo Ensemble**.

This technique allows for the construction of confidence intervals to reflect modelled uncertainty, providing a framework for decision-making under uncertainty. Results of **1.8** MAPE and **0.94** R^2 were achieved on the test set.

6 Neural Network

An artificial neural network[12] is a paradigm of automatic learning and processing inspired by the way animal nervous systems work. It is a system of interconnected neurons collaborating to produce an output stimulus. This system uses a mathematical or computational model based on a connectionist approach to computation.

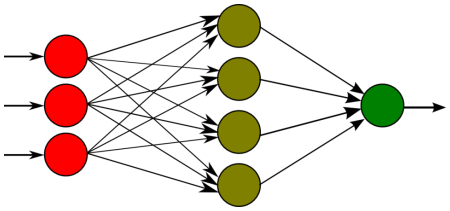

Fig. 17: **Artificial Neural Network**. A simple perceptron with 3 input neurons, 4 hidden-layer neurons, and 1 output neuron [12].

Initial experiments were carried out using a baseline model, shown in Table 4, adapted from the architecture proposed in [13].

To better align the model with the temporal characteristics of the dataset, *Long Short-Term Memory (LSTM)* units [14] were incorporated. LSTMs are particularly effective for time series forecasting, as their gating mecha-

nisms retain long-term dependencies and address the vanishing gradient problem. These adjustments aimed to enhance predictive accuracy and more effectively capture the underlying temporal patterns.

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

Activation is the process by which a neuron transforms its input into an output using an activation function. It is essential for introducing **non-linearity** into the neural network, which is the essence of neural networks. For the study, various variations of the **ReLU** function are 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)

First, the model was trained and evaluated using the *ReLU* activation function (Rectified Linear Unit), known for its computational simplicity and gradient propagation efficiency. In parallel, long-term memory was incorporated via *LSTM* (Long Short-Term Memory), a specific architecture within recurrent neural networks (*RNN*).

LSTMs are designed to overcome the limitations of traditional *RNNs*, which often struggle to retain relevant information over time due to the vanishing or exploding gradient problem. Unlike these, *LSTMs* introduce a gate-based structure (*input*, *forget*, and *output gates*) that regulates the information flow through the network.

This control mechanism, shown in Figure 18 allows the model to retain or discard information based on its relevance.

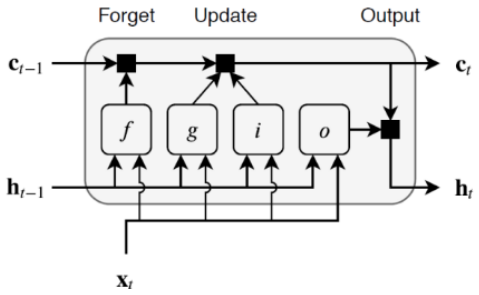

Fig. 18: LSTM.

LSTMs operate through the following gates that manage the information flow:

- **Forget gate**: Determines which information from memory is retained or discarded. The gate's value ranges from 0 (information discarded) to 1 (information retained).
- **Input gate**: Decides what new information is added to the memory, combining the current input with the previous state. It uses an activation function (usually a **sigmoid**) to control updates:

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

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

6.3 Bidirectional LSTM

The **Bidirectional LSTM** consists of two parallel input paths: one processes the sequence left-to-right and the other right-to-left. This structure gives the model a full view of the sequence, especially useful in tasks where an element's meaning depends on both previous and future elements, improving context capture and prediction quality.

To evaluate the different model types, we will generate a **traditional LSTM** and a **bidirectional** one for each of the mentioned activation functions, keeping the same initial architecture in all cases. This will allow a direct comparison of the impact of directionality on model performance.

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.

- **MSE** (Mean Squared Error): the mean of the squared differences between predicted and true values. It penalizes large errors more strongly. This is the loss function used during training.

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

- **MAPE** (Mean Absolute Percentage Error): the mean of the absolute relative errors expressed as a percentage. It is a widely used metric due to 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 that is explained by the model. A value close to 1 indicates strong explanatory power; values near 0 imply poor explanatory ability.

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

6.5 Results

As shown in Table 6, models incorporating LSTM units consistently outperformed their counterparts in terms of predictive accuracy. Among these, the traditional LSTM model using the *Exponential Linear Unit (ELU)* activation function achieved the best performance on the test set.

Although ELU is the most computationally demanding among the evaluated activation functions, it clearly demonstrated superior effectiveness, outperforming both *LeakyReLU* and the standard *ReLU*. In terms of performance hierarchy, *ELU* activation led to the most accurate results, followed by *LeakyReLU*, with *ReLU* being the least effective—though it remains the most computationally efficient. This trade-off highlights the importance of carefully selecting activation functions based on the performance–efficiency balance required by the task.

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 THE DIFFERENT MODELS, ORDERED BY INCREASING MAPE.

The training loss curve shows a smooth, steady decline, with early convergence achieved despite the model's complexity. Strong regularization prevents overfitting by promoting the learning of temporal patterns, improving generalization but slightly slowing convergence in later epochs. Figure 19 illustrates the evolution of validation loss across models.

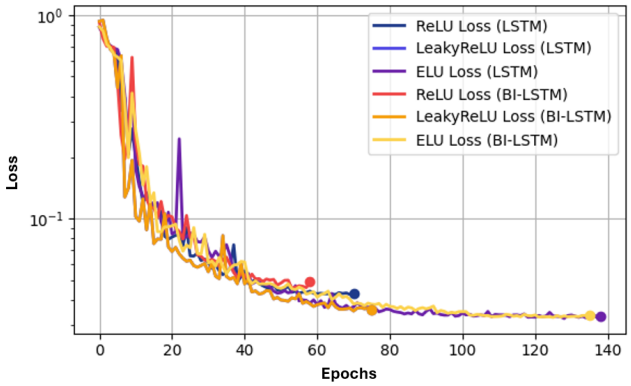

Fig. 19: Loss of the different models during training,for the validation dataset.

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

6.6 Effect of Batch Normalization

Batch Normalization [16] is a technique introduced by Sergey Ioffe and Christian Szegedy in 2015 to improve deep neural network training and prevent the *exploding gradient* problem [17], where gradients become excessively large, causing erratic weight updates.

This method normalizes layer activations so they have a mean close to zero and a standard deviation near 1. It reduces internal covariate shift and accelerates convergence [18].

- **During training:** the layer normalizes its output. For each channel:

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

  Where:

  - ϵ is a configurable constant,
  - γ is a learned scaling factor (initially 1),
  - β is a learned shift factor (initially 0),

- **During validation/testing:** the layer normalizes its output using a moving average of the mean and variance from training.

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

  Where:

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

  The layer will only normalize its inputs during inference.

7 Effect of Dropout

Dropout [19] is a regularization technique that randomly deactivates a percentage of neurons during each forward pass in training, forcing the model to avoid over-reliance on specific neurons.

- **During training:** Dropout disables random neurons, training a reduced model and promoting redundancy.
- **During validation/testing:** Dropout is disabled, and all neurons remain active, allowing the model to utilize its full representational 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 1 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 20 and 21 visualize the standard Dropout and Monte Carlo Dropout inference modes.

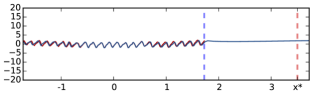

Fig. 20: Standard Dropout with mean weight [20].

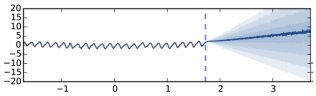

Fig. 21: MC Dropout with ReLU nonlinearities [20].

8 Conclusions

The models that demonstrated the highest levels of predictive accuracy 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 outperformed generalization capabilities, as presented 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	1.4533	**0.9684**

TABLE 7: Results of different models.

While all implemented models incorporate effective regularization mechanisms, the **LSTM ELU** with **Monte Carlo Dropout** consistently demonstrated the most robust performance. Beyond achieving strong quantitative results (Table 7), it excelled qualitatively by effectively capturing predictive uncertainty and highlighting instances with a higher likelihood of error. This capability is visually substantiated by the error distribution shown in Figure 22, where the model exhibits a well-calibrated response across varying conditions and maintains resilience to shifts in input data.

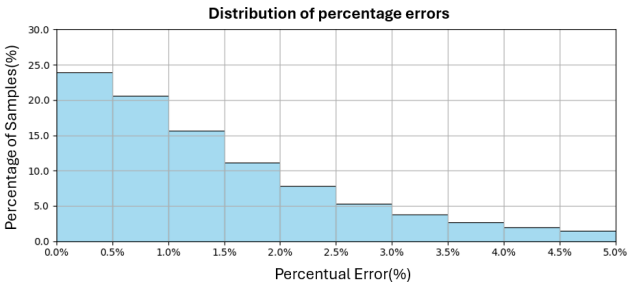

Fig. 22: Distribution of normalized percentage error in the LSTM ELU model prediction with Monte Carlo Dropout.

Acknowledgments

I would like to express my deep gratitude to all those who have been part of this process, who have accompanied me, and who have supported me unconditionally throughout this period. Their support and understanding have been crucial

in allowing me to dedicate quality time to this research. A special thanks goes to Albert Romero Sánchez and Xavier Miquel Armengol for introducing me to this field of study and for generously sharing their knowledge with passion and dedication. I also want to acknowledge the constant effort and invaluable guidance of Albert Romero throughout the entire tutoring process of this research project.

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. [Online]. Available: https://www.geeksforgeeks.org/vanishing-and-exploding-gradients-problems-in-deep-learning/

[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 2 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 3 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 4 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