Università di Perugia
Dipartimento di Matematica e Informatica

Corso di Laurea in Informatica

A Deep Learning approach for Time Series Imputation on Photovoltaic data

Relatori
Valentina Poggioni
Enrico Bellocchio
Alessandro Devo

Laureando
Nicolò Vescera

Anno Accademico 2022-2023

Contents

Abstract

The growing need for the adoption of tools capable of generating clean energy from renewable and sustainable sources has led to extensive generation and collection of energy production data, especially from photovoltaic panels installed worldwide. However, these data often have gaps and deficiencies due to various factors such as temporary failures, adverse weather conditions, or malfunctions of sensors and data collection instruments. Accurate imputation of these gaps is crucial to ensure the reliability of analyses and predictions based on this data. This thesis aims to address the problem of imputing time series data from photovoltaic panels using advanced deep learning techniques. In particular, three deep learning models based on Multi-Layer Perceptron (MLP), Recurrent Neural Networks (RNN) and Transformers are proposed to capture the complex temporal relationships between the total energy produced (target feature) and various components of the system. The models were trained on a dataset consisting of real data from a photovoltaic system with a power capacity of approximately $1MW$.

Chapter 1

Background

1.1 Problem Definition

This thesis aims to address the problem of imputing time series data from photovoltaic systems. Specifically, it often happens that data acquisition instruments in a system temporarily fail, causing a period of time, more or less extended, where the curve of the total energy produced is missing. Closing these gaps is a highly complex task because we must consider not only various factors such as solar irradiance, ambient temperature, cloud cover, rainfall, etc., but also the changing seasons, the passage of hours, and the day/night cycle.

Formally, we can define the problem as follows.

Definition 1.1.1 (**Imputation problem**)**.** Given a set of N time series data representing a photovoltaic system, $S = \{S_1, S_2, \ldots, S_N\}$, where each time series $S_i = (t_i, v_i)$, represents the moment t_i when the value v_i was recorded and $S^* \in S$ is a specific target series that represent the *Total Generated Energy*, the objective of the **imputation problem** is to estimate eventual missing values $v'_j, \ldots, v'_{j+h}$ belonging to S^*, using information from other time points temporally adjacent to t'_j and t'_{j+h}.

1.2 Photovoltaic Implant

Solar photovoltaic (PV) energy systems consist of various components, each with a distinct function. The selection of components within the system is contingent on the specific type of system and its intended use. For instance, in a basic PV-direct system, the system comprises a solar module or an array (comprising two or more interconnected modules) and the load, which is the device consuming the generated energy [21]. In more complex installations, which may also include energy storage systems, we can find a significantly higher number of components.

1.2.1 Solar Module

The majority of solar modules available on the market and used for residential and commercial solar systems are silicon-crystalline. These modules consist of multiple strings of solar cells, wired in series (positive to negative), and are mounted in an aluminum frame. The size or area of the cell determines the amount of amperage: the larger the cell, the higher the amperage [21].

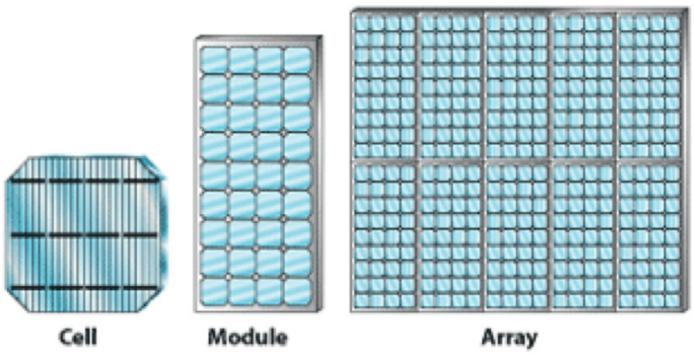


Figure 1.1: The solar cell is the basic component. Cells wired together and mounted in a frame compose a solar module. Several modules wired together form an array [21].

1.2.2 Solar Array

The solar array is made up of multiple PV modules wired together. Connecting the negative wire of one module to the positive wire of a second module is the beginning of a series string. Wiring modules in series results in the voltage of each of the two modules is added together. A series string represents the summed voltages of each individual module. The negative cable of one module is connected to the positive cable of the next module. In a large system, multiple strings are assembled and the non-connected ends are connected to homerun leads which are landed at the terminals of an enclosure located near the array. The goal is to wire modules in series to build voltage.

1.2.3 Junction Box

A PV system array with multiple strings of modules will have a positive lead and a negative lead on the end of each string. The positive leads will be connected to individual fuses and the negative leads will be connected to a negative busbar in an enclosure. This is called the source circuit [21]. The junction box serves

to “combine” multiple series strings into one parallel circuit. For example, an array with three strings of 10 modules wired in series would produce 300 volts (10 modules x 30 volts) per string and 4 amps per string. When the leads are landed in the combiner box, the circuit would produce 300 volts at 12 amps (3 strings x 4 amps/string). Once the circuits are combined, leaving the box it is referred to as the output circuit.

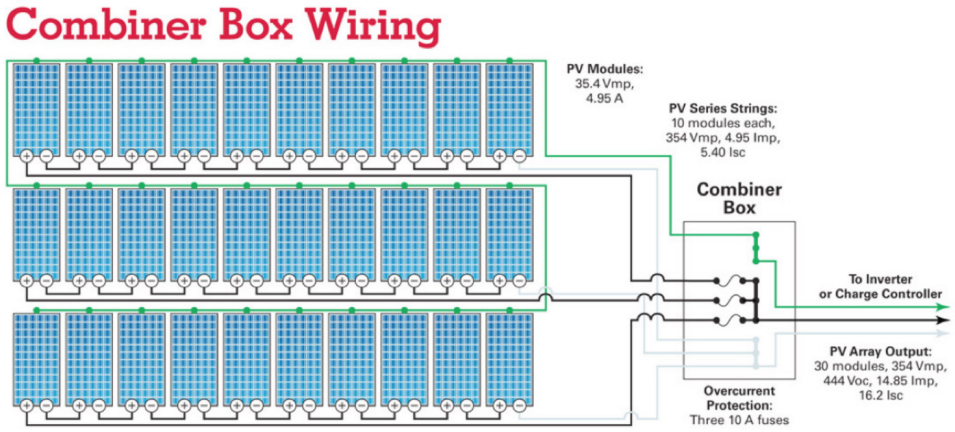


Figure 1.2: This figure represent an output circuit made of 3 string, each one hosts 10 solar modules [21].

1.2.4 Inverter

Energy from an array or a battery bank is direct current. This will provide for DC loads such a lights, fans, pumps, motors, and some specialty equipment. However, if the energy is to be used to power loads that operate on alternating current (AC), as what is found in a residence, the current needs to be converted. The inverter changes DC energy to AC energy. Inverters are available in many different sizes for various-sized loads. A string inverter is used to convert DC power from a solar array to AC power and can be connected to an AC distribution power panel (service panel) in a residence or facility [21].

1.2.5 System Metering

Several tools are available to help the solar user to monitor their system. On stand-alone or off-grid PV systems, the battery meter is used to measure the energy coming in and going out of the battery bank. Charging and discharging of batteries, and proper functioning of the charging system is important to alert the user to incomplete charging, battery decline, or possible system shutdown[21].

System monitoring with web-based tools and apps allow the solar user to see system activity using a cell phone or tablet from a location away from their system.

1.3 Open-Meteo

Open-Meteo is an open-source weather data platform and community-driven weather forecasting project. It aims to provide access to weather data and forecasts that are openly accessible to the public and can be used for a variety of applications, including research, development, and personal use. It offers a diverse range of APIs that go beyond traditional weather forecasting such as past weather data, ocean data, air quality, ensemble forecasts, climate forecasts based on IPCC predictions, and even floods [24]. Open-Meteo offers over 80 years of hourly weather data, covering any location on earth, all at a 10 kilometer resolution. This extensive dataset is very useful to delve into the past and analyze historical weather patterns.

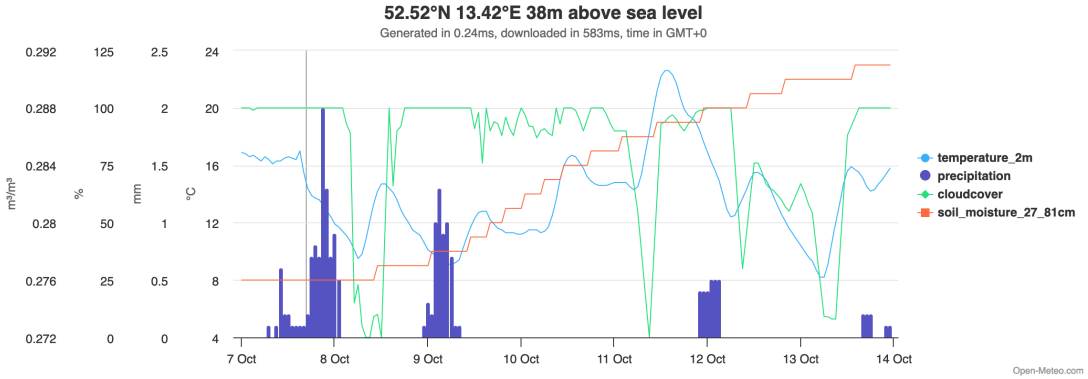


Figure 1.3: A plot about some Open-Meteo data.

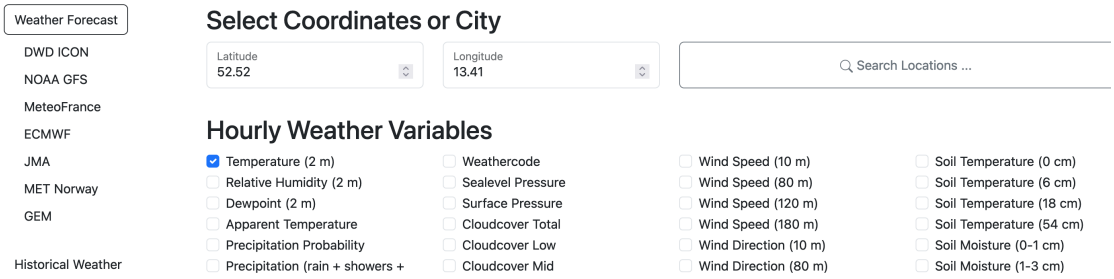


Figure 1.4: Open-Meteo forecast features.

1.4 Multi Layer Perceptron

The Multi Layer Perceptron model is the most known and most frequently used type of neural network. The signals are transmitted within the network in one direction, from input to output: there is no loop, the output of each neuron does not affect the neuron itself. Layers which are not directly connected to the environment are called hidden.

There are also feed-back networks, which can transmit impulses in both directions, due to reaction connections in the network. These types of networks are very powerful and can be extremely complicated. Introduction of several layers was determined by the need to increase the complexity of decision regions.

A perceptron with a single layer and one input generates decision regions under the form of semi planes. By adding another layer, each neuron acts as a standard perceptron for the outputs of the neurons in the anterior layer, thus the output of the network can estimate convex decision regions, resulting from the intersection of the semi planes generated by the neurons.

The power of the multilayer perceptron comes precisely from non-linear activation functions. Almost any non-linear function can be used for this purpose, except for polynomial functions. Currently, the functions most commonly used today are the single-pole (or logistic) sigmoid [17].

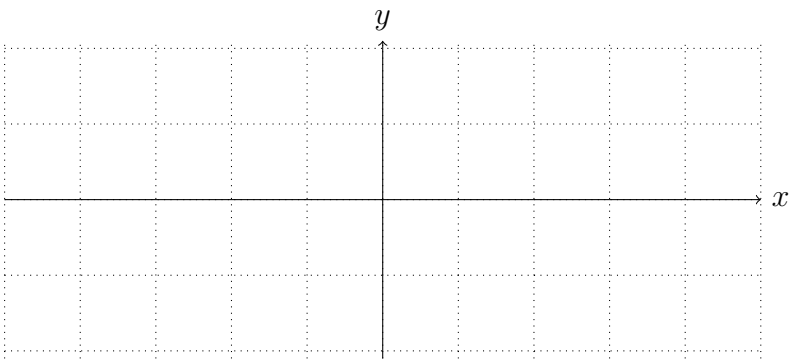


Figure 1.5: Sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$

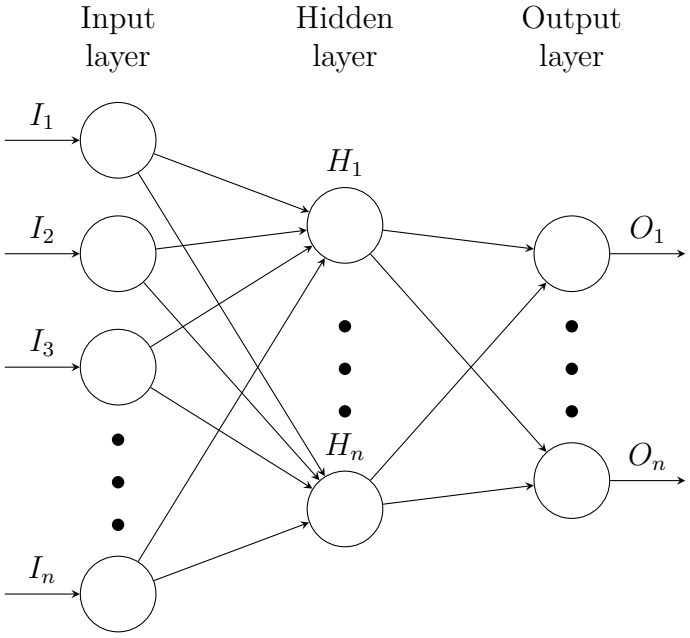
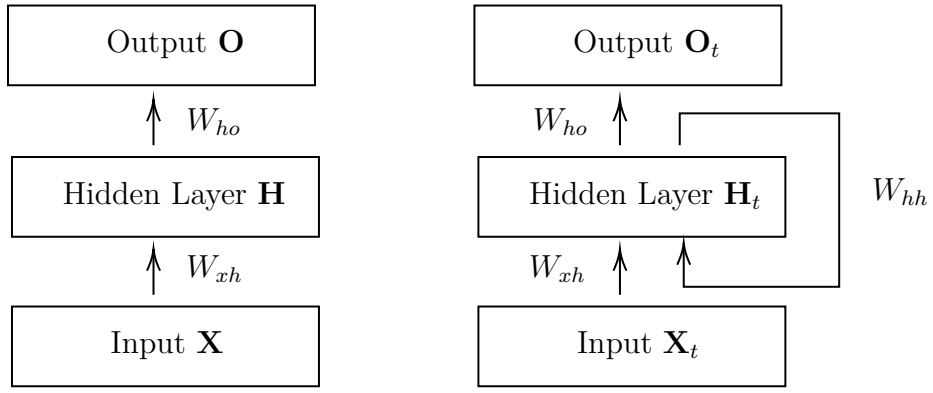


Figure 1.6: Multi Layer Perceprton architecture.

1.5 Recurrent Neural Network

Recurrent Neural Networks (RNNs) are a type of neural network architecture which is mainly used to detect patterns in sequences of data. Such data can be handwriting, genomes, text or numerical time series which are often produced in industry settings (e.g. stock markets or sensors). However, they are also applicable to images if these get respectively decomposed into a series of patches and treated as a sequence. On a higher level, RNNs find applications in Language Modelling and Generating Text, Speech Recognition, Generating Image Descriptions or Video Tagging.

What differentiates Recurrent Neural Networks from Multi-Layer Perceptrons (MLPs) and in general Feedforward Neural Networks is how information gets passed through the network. While Feedforward Networks pass information through the network without cycles, the RNN has cycles and transmits information back into itself [18]. This enables them to extend the functionality of Feedforward Networks to also take into account previous inputs $X_{0:t-1}$ and not only the current input X_t. This difference is visualised on a high level in Figure 1.7. Note, that here the option of having multiple hidden layers is aggregated to one Hidden Layer block H. This block can obviously be extended to multiple hidden layers.



Feed Forward Neural Network *Recurrent Neural Network*

Figure 1.7: Visualization of difference between MLPs and RNN.

1.5.1 Gated Recurrent Unit

Gated Recurrent Units (GRU) are an advanced variation of RRNs (Recurrent Neural Network) [1]. GRUs use update gate and reset get for solving a standard RNNs vanishing gradient issue. These are essentially 2 vectors that decide the type of information to be passed towards the output. What makes these vectors special is that programmers can train them to store information, especially from long ago [3].

They do all of this by utilizing its gated units which help solve vanishing/exploding gradient problems often found in traditional recurrent neural networks. These gates are helpful for controlling any information to be maintained or discarded for each step. It is also worth keeping in mind that gated recurrent units make use of reset and update gates.

- **Update Gates Function**: The main function of the update gate is to determine the ideal amount of earlier info that is important for the future. One of the main reasons why this function is so important is that the model can copy every single past detail to eliminate fading gradient issue.
- **Reset Gate's Function**: A major reason why reset gate is vital because it determines how much information should be ignored. It would be fair to compare reset gate to LSTMs forget gate because it tends to classify unrelated data, followed by getting the model to ignore and proceed without it.

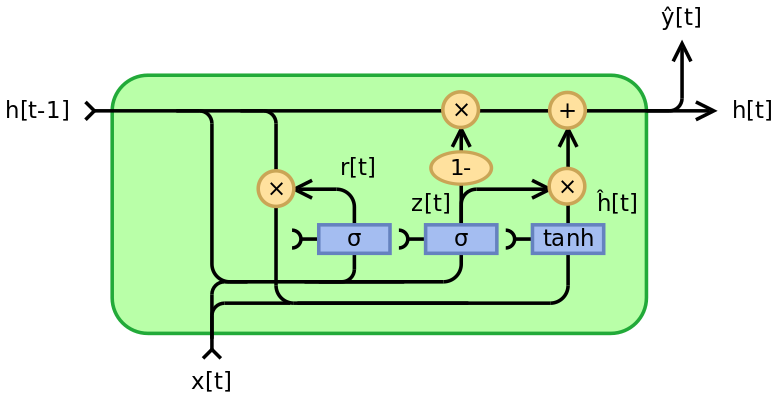


Figure 1.8: Gated Recurrent Unit (GRU) architecture.

1.6 Transformer

Recurrent neural networks, long short-term memory and gated recurrent neural networks in particular, have been firmly established as state of the art approaches in sequence modeling and transduction problems such as language modeling and machine translation [22]. Recurrent models typically factor computation along the symbol positions of the input and output sequences. Aligning the positions to steps in computation time, they generate a sequence of hidden states h_t, as a function of the previous hidden state h_{t-1} and the input for position t. This inherently sequential nature precludes parallelization within training examples, which becomes critical at longer sequence lengths, as memory constraints limit batching across examples. Attention mechanisms [23] have become an integral part of compelling sequence modeling and transduction models in various tasks, allowing modeling of dependencies without regard to their distance in the input or output sequences. The Transformer allows for significantly more parallelization [4].

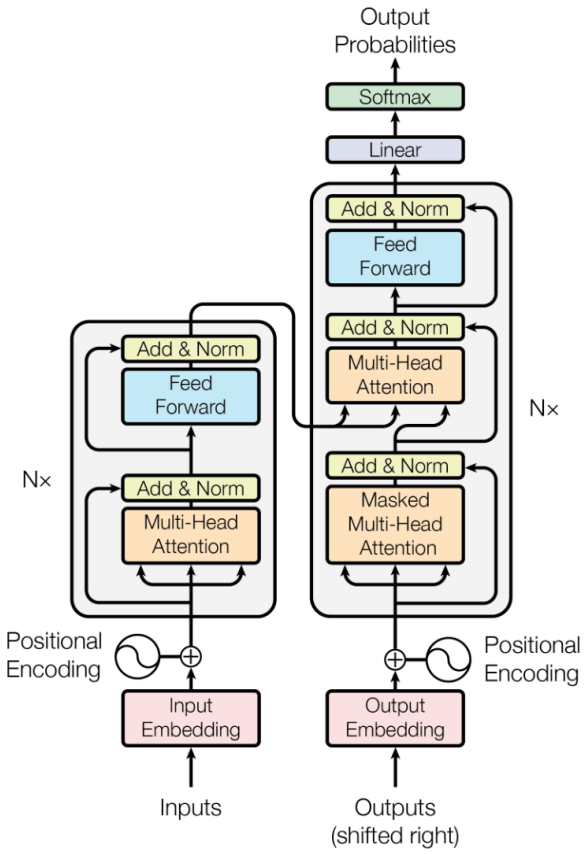


Figure 1.9: This figure shows the Transformers model architecture [23].

Most competitive neural sequence transduction models have an encoder-decoder structure [22]. Here, the encoder maps an input sequence of symbol representations $(x_1, \ldots, x_n)$ to a sequence of continuous representations $z = (z_1, \ldots, z_n)$. Given z, the decoder then generates an output sequence $(y_1, \ldots, y_m)$ of symbols, one element at a time. At each step the model is auto-regressive, consuming the previously generated symbols as additional input when generating the next.

The Transformer follows this overall architecture using stacked self-attention and point-wise, fully connected layers for both the encoder and decoder, shown in the left and right halves of Figure 1.9, respectively.

- *Encoder*: The encoder is composed of a stack of $N = 6$ identical layers. Each layer has two sub-layers. The first is a multi-head self-attention mechanism [23], and the second is a simple, position-wise fully connected feed-forward

network. Residual connection are employed around each of the two sub-layers, followed by layer normalization. That is, the output of each sub-layer is LayerNorm(x + Sublayer(x)), where Sublayer(x) is the function implemented by the sub-layer itself. To facilitate these residual connections, all sub-layers in the model, as well as the embedding layers, produce outputs of dimension $dmodel = 512$.

- *Decoder*: The decoder is also composed of a stack of $N = 6$ identical layers. In addition to the two sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs multi-head attention over the output of the encoder stack. Similar to the encoder, there are residual connections around each of the sub-layers, followed by layer normalization. The self-attention sub-layer is modified in the decoder stack to prevent positions from attending to subsequent positions. This masking [23], combined with fact that the output embeddings are offset by one position, ensures that the predictions for position i can depend only on the known outputs at positions less than i.

- *Attention*: An attention [23] function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors [4]. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key [23].

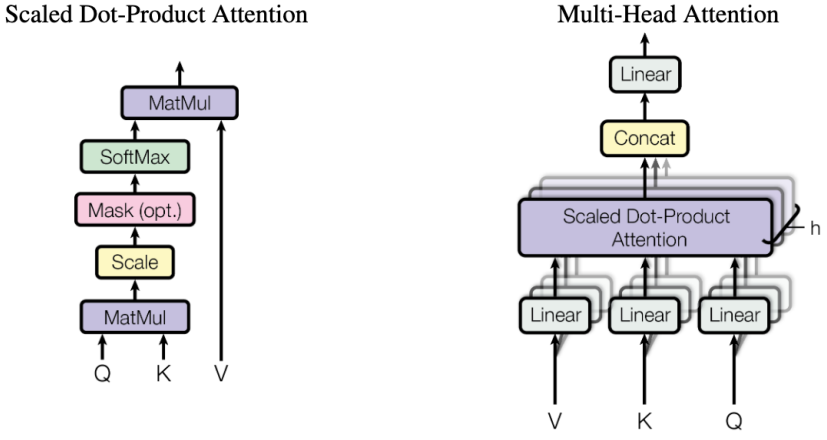


Figure 1.10: Scaled Dot-Product Attention [23] (left). Multi-Head Attention [23] consists of several attention layers running in parallel (right).

Chapter 2

Dataset and Data Preprocessing

In this chapter, we will analyze the dataset in detail, focusing on its structure, the various features it contains, and the procedures we have adopted to transform it into a new database with a better and more efficient format to be used in model training and evaluation. We will also see how the Open-Meteo meteorological service was used and how the most important features were selected to enable the training of various models.

2.1 Original Dataset

The dataset at our disposal describes a period of almost two years (from 02-02-2022 to 16-06-2023) and comes from a 978 kW photovoltaic plant located in the province of Bari. It consists of 3 inverters, 27 field panels, 1 meter, 1 solarimeter, 2 interface protections and 1 "system" device in which data from Solargis are stored. The dataset is organized into files, one for each type of device, representing each individual day, and the data is aggregated every 5 minutes. Each row contains a reference to the device it belongs to (`deviceName` and `deviceId`).

File Name
2022_02_02_Rofilo_NP00003174_inverter.csv
2022_02_02_Rofilo_NP00003174_meteorology.csv
2022_02_02_Rofilo_NP00003174_meter.csv
2022_02_02_Rofilo_NP00003174_other.csv
2022_02_02_Rofilo_NP00003174_plantDevice.csv
2022_02_02_Rofilo_NP00003174_stringbox.csv
2022_02_03_Rofilo_NP00003174_inverter.csv
2022_02_03_Rofilo_NP00003174_meteorology.csv
2022_02_03_Rofilo_NP00003174_meter.csv
2022_02_03_Rofilo_NP00003174_other.csv
2022_02_03_Rofilo_NP00003174_plantDevice.csv
2022_02_03_Rofilo_NP00003174_stringbox.csv
...

Table 2.1: Some files from our dataset.

timestamp	serial	...	TotalEnergy (kWh)	Frequency (Hz)	deviceid
2022-10-23 04:30:00	INV01	...	357196.88	50.06	83204
2022-10-23 04:35:00	INV01	...	357196.88	50.06	83204
...	...	...	...	...	...
2022-10-23 13:00:00	INV01	...	357921.16	49.95	83204
2022-10-23 13:05:00	INV01	...	357935.24	49.95	83204
...	...	...	...	...	...
01/02/2023 23:45	INV01	...	431324.36	49.88	83204
01/02/2023 23:50	INV01	...	431324.36	49.88	83204

Table 2.2: Some lines from an Inverter file.

2.1.1 Inverter

Inside the files related to the inverters, we can find data such as the processor's operating temperature (`InternalTemperature`), various Alternating and Direct currents produced (`CurrentAC` and `CurrentDC`), Delivered Power, total energy produced by the individual inverter and other status and control bits that indicate various stages of the inverter's life (boot, reset, ready, etc.).

Down below a table with all the available features:

Name	Unit Symbol	Description
CommunicationCode	-	Communication Code
Failure 3	-	Active Alarm
Failure 4	-	Isolation Alarm
CurrentDC	A	Photovoltaic field Current
CurrentAC	A	Network Current
CurrentAC Phase1	A	Line RMS Current Phase R
CurrentAC Phase2	A	Line RMS Current Phase S
CurrentAC Phase3	A	Line RMS Current Phase T
TotalEnergy	kWh	Active Energy Delivered
Frequency	Hz	Network Frequency
PowerAC Phase1	kW	Line PA Phase R
PowerAC Phase2	kW	Line PA Phase S
PowerAC Phase3	kW	Line PA Phase T
PowerAC	kW	Delivered PA
PowerDC	kW	Photovoltaic field Power
Status	-	Inverter State
Failure	-	Network docking PLL State
Failure 2	-	Network State 1
Failure 1	-	Network State 2
InternalTemperature	C	CPU Temp.
HeatSinkTemperature	C	IGBT Temp.
VoltageDC	V	Photovoltaic field Voltage
VoltageAC	V	Network Voltage
VoltageAC Phase1	V	Line RMS Voltage Phase R
VoltageAC Phase2	V	Line RMS Voltage Phase S
VoltageAC Phase3	V	Line RMS Voltage Phase T

Table 2.3: All available features from an `inverter` file.

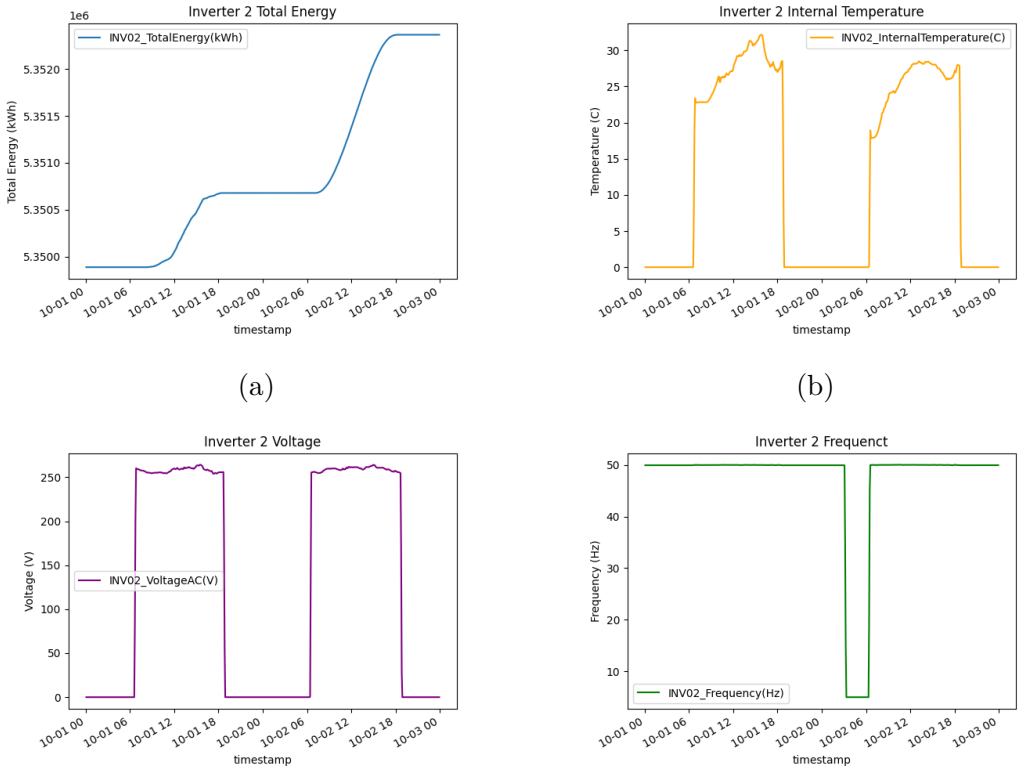


Figure 2.1: The image shows plots of some features of the available plant related to Inverter 2. In (a), the Total Energy is displayed, in (b), the CPU Temperature, in (c), the Voltage AC, and in (d), the Frequency.

2.1.2 Junction Box

In the files related to the Junction Box or Combiner Box, we find data that describes the current production of the various strings they are connected to (`CurrentString1-7`, in general, each Junction Box manages 7 strings), some temperatures (such as `ModuleTemperature`), and some control bits to check proper operation.

Name	Unit Symbol	Description
CommunicationCode	-	Communication Code
Failure	-	Strings Alarm
CurrentString1	A	Current I1
CurrentString2	A	Current I2
CurrentString3	A	Current I3
CurrentString4	A	Current I4
CurrentString5	A	Current I5
CurrentString6	A	Current I6
CurrentString7	A	Current I7
AverageStringCurrent	A	Average Current
Irradiance	W/m^2	Modules Irradiation
Failure 1	-	Open Strings
Failure 2	-	Not Perform. Strings
EnvironmentTemperature	C	Environment Temperature
ModuleTemperature	C	Modules Temperature
InternalTemperature	C	Board Temperature

Table 2.4: All available features form a `stringbox` file.

2.1.3 Solargis

Solargis is a company specialized in providing solar data and forecasting services for photovoltaic installations and solar energy-related projects [19]. Their main goal is to provide precise and reliable information on solar irradiation and solar weather conditions anywhere in the world. This data is essential for the design, optimization, and management of photovoltaic systems. Solargis collects and provides detailed data on global, direct, and diffuse solar irradiation at every geographical location [19]. This data allows photovoltaic system developers to assess the amount of available solar energy in a given area, which is crucial for properly sizing the system and calculating production forecasts.

timestamp	...	SolargisGHI(W/m2)	SolargisGTI(W/m2)
2022-08-01 11:40:00	...	896	978
2022-08-01 11:45:00	...	896	978
2022-08-01 11:50:00	...	896	978
2022-08-01 11:55:00	...	914	1001
2022-08-01 12:00:00	...	914	1001
2022-08-01 12:05:00	...	914	1001
2022-08-01 12:10:00	...	928	1019
2022-08-01 12:15:00	...	928	1019
2022-08-01 12:20:00	...	928	1019

Table 2.5: This table displays some Solargis data from "2022_08_01_Rofilo_NP00003174_plantDevice.csv" file.

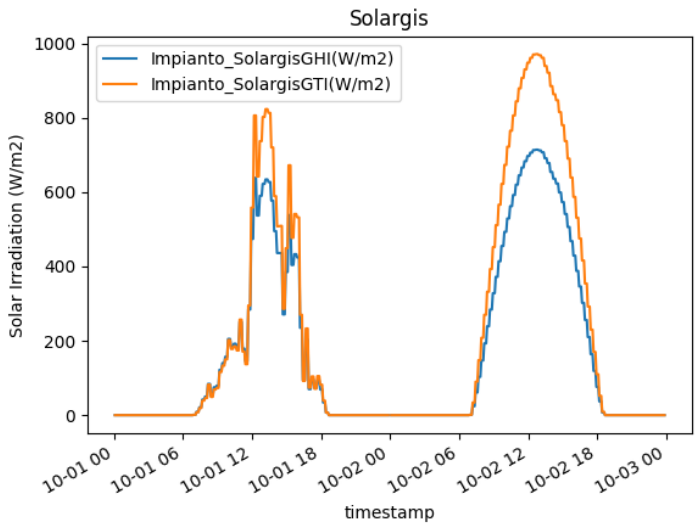


Figure 2.2: Solargis GHI & GTI plot.

Solar radiation takes a long journey until it reaches Earths surface. So when modelling solar radiation, various interactions of extra-terrestrial solar radiation with the Earths atmosphere, surface and objects are to be taken into account[19]. The component that is neither reflected nor scattered, and which directly reaches the surface, is called direct radiation; this is the component that produces shadows. The component that is scattered by the atmosphere, and which reaches the ground is called diffuse radiation. The small part of the radiation reflected by the

surface and reaching an inclined plane is called the reflected radiation. These three components together create global radiation.

In solar energy applications, the following parameters are commonly used in practice:

- Direct Normal Irradiation/Irradiance (DNI) is the component that is involved in thermal (concentrating solar power, CSP) and photovoltaic concentration technology (concentrated photovoltaic, CPV) [19].
- Global Horizontal Irradiation/Irradiance (GHI) is the sum of direct and diffuse radiation received on a horizontal plane. GHI is a reference radiation for the comparison of climatic zones; it is also essential parameter for calculation of radiation on a tilted plane [19].
- Global Tilted Irradiation/Irradiance (GTI), or total radiation received on a surface with defined tilt and azimuth, fixed or sun-tracking. This is the sum of the scattered radiation, direct and reflected. It is a reference for photovoltaic (PV) applications, and can be occasionally affected by shadow[19].

Name	Unit Symbol	Description
SolargisGHI	W/m^2	Solargis Global Horizontal Irradiation
SolargisGTI	W/m^2	Solargis Global Tilted Irradiation

Table 2.6: All available features from a `plantDevice` file.

2.1.4 Meteorology

In the Meteo files, we can find some environment temperature and solar irradiance data. Is important to mention that these features are not as powerful as weather forecast or Solargis data for the Imputation task.

Name	Unit Symbol	Description
COMMUNICATION_CODE SOL	-	Communication Code
Irradiance SOL	W/m^2	Irradiance
Module Temperature HEX SOL	-	All Registers
Module Temperature SOL	C	Module Temperature

Table 2.7: All available features from a `meteo` file.

2.1.5 Meter

The Meter files contain information about the current injected into and drawn from the network. It is from here that we get the values of our target feature `Cont_TotalEnergy(kWh)`.

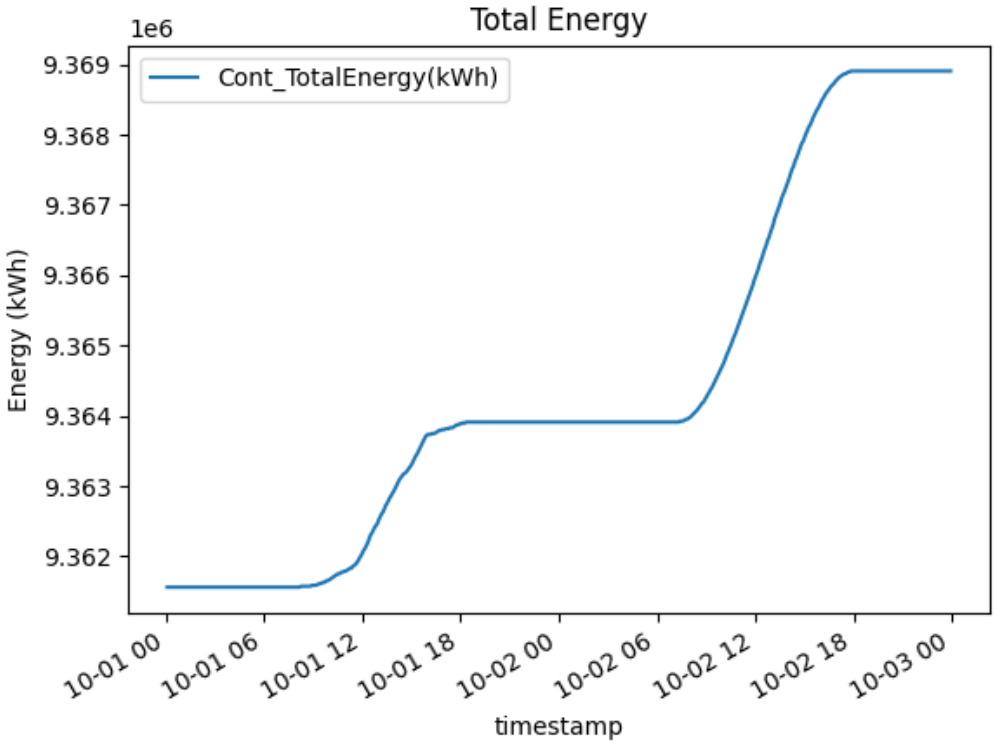


Figure 2.3: Total Energy plot, our target feature.

Name	Unit Symbol	Description
COMMUNICATION_CODE Cont	-	Communication Code
Status Cont	-	Status
Totale Energia Immessa Cont	kWh	Total Energy
Totale Energia Prelevata Cont	kWh	Total Energy Imported

Table 2.8: All available features from a `meter` file.

2.1.6 Other

In the Other type files, we can find the remaining, less relevant, features that describe the behavior and functioning of the leftover elements that make up the photovoltaic plant system.

Name	Unit Symbol	Description
COMMUNICATION_CODE NV10P	-	Communication Code
CB-State NV10P	-	CB State
Frequency NV10P	Hz	Frequency
IN1 NV10P	-	Digital IN 1
IN2 NV10P	-	Digital IN 2
Last Trip Cause NV10P	-	Last Trip Cause
NV10P - Trip BF	-	Digital IN 25
NV10P - Trip f<	-	Digital IN 24
NV10P - Trip f>	-	Digital IN 23
NV10P - Trip U<	-	Digital IN 21
NV10P - Trip U>	-	Digital IN 22
UE NV10P	V	UE
UL1 NV10P	V	Voltage AC Phase 1
UL2 NV10P	V	Voltage AC Phase 2
UL3 NV10P	V	Voltage AC Phase 3
Un NV10P	V	Un
Unp NV10P	V	Unp
COMMUNICATION_CODE NA16	-	Communication Code
CB-State NA16	-	CB State
IE NA16	A	IE
IL1 NA16	A	Current AC Phase 1
IL2 NA16	A	Current AC Phase 2
IL3 NA16	A	Current AC Phase 3
IN1 NA16	-	Digital IN 1
IN2 NA16	-	Digital IN 2
NA16 - Protection Trip	-	Digital IN 25

Table 2.9: All available features from an `other` file.

2.2 Dataset Realization

For the creation of a dataset taht can be passed directly to our imputation models, we devised a procedure that allowed us to merge all the original files into a single

file, where for each timestamp, we have all the data for the entire system at that exact moment. Below is the final structure of the dataset and the algorithm for generating it.

timestamp	**DEV.NAME$_1$_FEAT$_1$**	...	**DEV.NAME$_n$_FEAT$_n$**
01/10/2022 10:00	...	...	...
01/10/2022 10:05	...	...	...
01/10/2022 10:10	...	...	...

Table 2.10: Final dataset feature structure.

Algorithm 1 Dataset aggregation algorithm

Require: data_folder
Ensure: `data_folder` exists
  dev_types ← find all file types inside `data_folder` (e.g. meter, inverter, ...)
  dev_content ← a dictionary with `dev_types` types as *keys* and empty *values*
  for each key **in** dev_conent.keys **do**
    files ← find all file matching type `key` inside `data_folder`
    sort `files` by date (asc.)
    temp_type_aggregate ← and empty csv table
    for each file **in** files **do**
      append all `file` lines to `temp_type_aggregate` table
    end for
    dev_content[key] ← temp_type_aggregate
  end for
  ▷ At this time we have a dictionary mapping a file type with all its available data

  dataset ← an empty csv table
  for each type, data **in** dev_content **do** ▷ `type` is *key*, `data` is *value*
    rename all `data` *columns* to `data.deviceID_`*column*`.name`
    except for 'timestamp' *column*
    dataset ← `dataset` and `data` tables using 'timestamp' column
  end for
  save `dataset` table to file

timestamp	INV01_PowerAC	...	Cont_TotalEnergy
2022-02-02 00:05:00	NaN	...	NaN
2022-02-02 00:10:00	NaN	...	NaN
...	...	...	...
2022-06-22 10:20:00	175.66	...	8900941.5
2022-06-22 10:25:00	178.29	...	8900995.5
2022-06-22 10:30:00	180.82	...	8901036.0
...	...	...	...
2023-06-16 18:00:00	NaN	...	NaN
2023-06-16 18:05:00	NaN	...	NaN

Table 2.11: Some data from dataset after running Algorithm 1

2.2.1 Timestamp cyclical encoding

To enable the models to learn the alternation of minutes, days, and months as effectively as possible during the training phase, we applied a procedure to transform each individual timestamp into a pair of sine and cosine values, thus performing a cyclic encoding of various seasonalities.

Algorithm 2 Cyclical Encoding Algorithm

Require: dataset table

Ensure: `dataset` **is not** empty

dataset['minute_sin'] $\leftarrow \sin(2\pi(\frac{\texttt{dataset.timestamp.minute}}{60}))$

dataset['minute_cos'] $\leftarrow \cos(2\pi(\frac{\texttt{dataset.timestamp.minute}}{60}))$

dataset['hour_sin'] $\leftarrow \sin(2\pi(\frac{\texttt{dataset.timestamp.hour}}{24}))$

dataset['hour_cos'] $\leftarrow \cos(2\pi(\frac{\texttt{dataset.timestamp.hour}}{24}))$

dataset['day_sin'] $\leftarrow \sin(2\pi(\frac{\texttt{dataset.timestamp.day}}{31}))$

dataset['day_cos'] $\leftarrow \cos(2\pi(\frac{\texttt{dataset.timestamp.day}}{31}))$

dataset['month_sin'] $\leftarrow \sin(2\pi(\frac{\texttt{dataset.timestamp.month}}{12}))$

dataset['month_cos'] $\leftarrow \cos(2\pi(\frac{\texttt{dataset.timestamp.month}}{12}))$

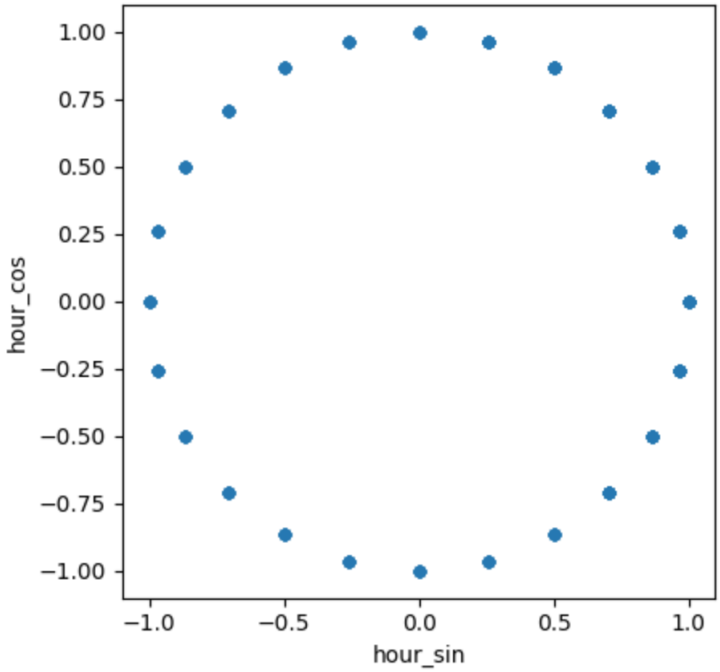


Figure 2.4: Hour cyclical encoding plot.

2.2.2 Historical weather

In addition to the information from Solargis (see Section 2.1.3), the models may require other weather data, such as cloud cover, rainfall, etc. Therefore, we made an API call to Open-Meteo servers (see Section 1.3), using the "Historical Weather" functionality, to obtain the accurate past forecasts for the time period covered by our dataset (from 02-02-2022 to 18-06-2023).

Algorithm 3 Open-Meteo data request Algorithm

Require: dataset table
Ensure: `dataset` **is not** empty
meteo_data ← require meteo feature from Open-Mete public API
dataset ← merge `dataset` and `meteo_data` tables using `timestamp` column

/* At this time we have `dataset` timestamps as 5 minute frequency and `meteo_data` as 1 hour frequency. */

use a *forward fill* method to fill gaps inside `dataset` table only for `meteo_data` columns.

We have obtained most of the information made available by Open-Meteo since the API request is free [24] and computationally and storage-wise inexpensive. These will be filtered at a later time (see Section 2.3). Among these, we find `sunrise` and `sunset`, which indicate the exact time (down to the minute) of when dawn and sunset occurred, respectively. These two features will be used to create a new column in our dataset called `is_day`, which will allow the model to understand whether it is analyzing a period of time during the day or night. This is to ensure that it does not make energy production predictions during the night.

Feature	Unit	Note
`temperature_2m`	řC	Air temperature at 2 meters above ground.
`relativehumidity_2m`	%	Relative humidity at 2 meters above ground.
`dewpoint_2m`	řC	Dew point temperature at 2 meters above ground. The dew point of a given body of air is the temperature to which it must be cooled to become saturated with water vapor. This temperature depends on the pressure and water content of the air.
`apparent_temperature`	řC	Apparent temperature is the perceived feels-like temperature combining wind chill factor, relative humidity and solar radiation.
`precipitation`	mm	Total precipitation (rain, showers, snow) sum of the preceding hour. Data is stored with a 0.1 mm precision. If precipitation data is summed up to monthly sums, there might be small inconsistencies with the total precipitation amount.
`weathercode`	wmo	Weather condition as a numeric code. Follow WMO weather interpretation codes. Weather code is calculated from cloud cover analysis, precipitation and snowfall. As barely no information about atmospheric stability is available, estimation about thunderstorms is not possible.
`pressure_msl` `surface_pressure`	hPa	Atmospheric air pressure reduced to mean sea level (msl) or pressure at surface. Typically pressure on mean sea level is used in meteorology. Surface pressure gets lower with increasing elevation.
`cloudcover` Instant	%	Total cloud cover as an area fraction.

et0_fao_ evapotranspiration	mm	ET_0 Reference Evapotranspiration of a well watered grass field. Based on FAO-56 Penman-Monteith equations ET_0 is calculated from temperature, wind speed, humidity and solar radiation. Unlimited soil water is assumed. ET_0 is commonly used to estimate the required irrigation for plants.
vapor_pressure_ deficit	kPa	Vapor Pressure Deificit (VPD) in kilopascal (kPa). For high VPD (>1.6), water transpiration of plants increases. For low VPD (<0.4), transpiration decreases.
windspeed_10m windspeed_100m	km/h	Wind speed at 10 or 100 meters above ground. Wind speed on 10 meters is the standard level.
winddirection_10m winddirection_100m	ř	Wind direction at 10 or 100 meters above ground.
windgusts_10m	km/h	Gusts at 10 meters above ground of the indicated hour. Wind gusts in CERRA are defined as the maximum wind gusts of the preceding hour.
soil_temperature_0_ to_7cm soil_temperature_7_ to_28cm soil_temperature_28_ to_100cm soil_temperature_100_ to_255cm	řC	Average temperature of different soil levels below ground.
soil_moisture_0_ to_7cm soil_moisture_7_ to_28cm soil_moisture_28_ to_100cm soil_moisture_100_ to_255cm	m^3/m^3	Average soil water content as volumetric mixing ratio at 0-7, 7-28, 28-100 and 100-255 cm depths.

Table 2.12: All feature selected from Open-Meteo with description[24].

The new feature `is_day` contains two values: 0 and 1, representing *Night* and *Day*, respectively. To create this feature, we add 0 with a frequency of 5 minutes until sunrise, then add 1 until sunset, and finally add more 0s until the end of the day. We repeat this process for each day in the dataset to create a new column that, for each timestamp, will indicate whether it is day or night.

Algorithm 4 is_day feature generation Algorithm

Require: dataset table, sunrise and sunset table
Ensure: `dataset` **is not** empty, `sunrise` and `sunset` tables **match** `dataset` days

```
is_day ← empty table column
for each day in dataset.days do
    temp_is_day ← empty table column
    day_start ← sunrise[day]                    ▷ get sunrise timestamp for that day
    day_start ← find the nearest dataset.timestamp to day_start
    day_end ← sunset[day]
    day_end ← find the nearest dataset.timestamp to day_end
    /* if sunrise[day] = 7:04 then day_start = 7:05 */

    /* 0 means Night, 1 mean Day */
    temp_is_day ← add 0s starting from 00:00 to day_start
     as 5 minute frequency
    temp_is_day ← add 1s starting from day_start to day_end
     as 5 minute frequency
    temp_is_day ← add 0s starting from day_end day end
     as 5 minute frequency
    append to is_day column temp_is_day values
end for
merge is_day column to dataset table
```

2.2.3 Dealing with gaps

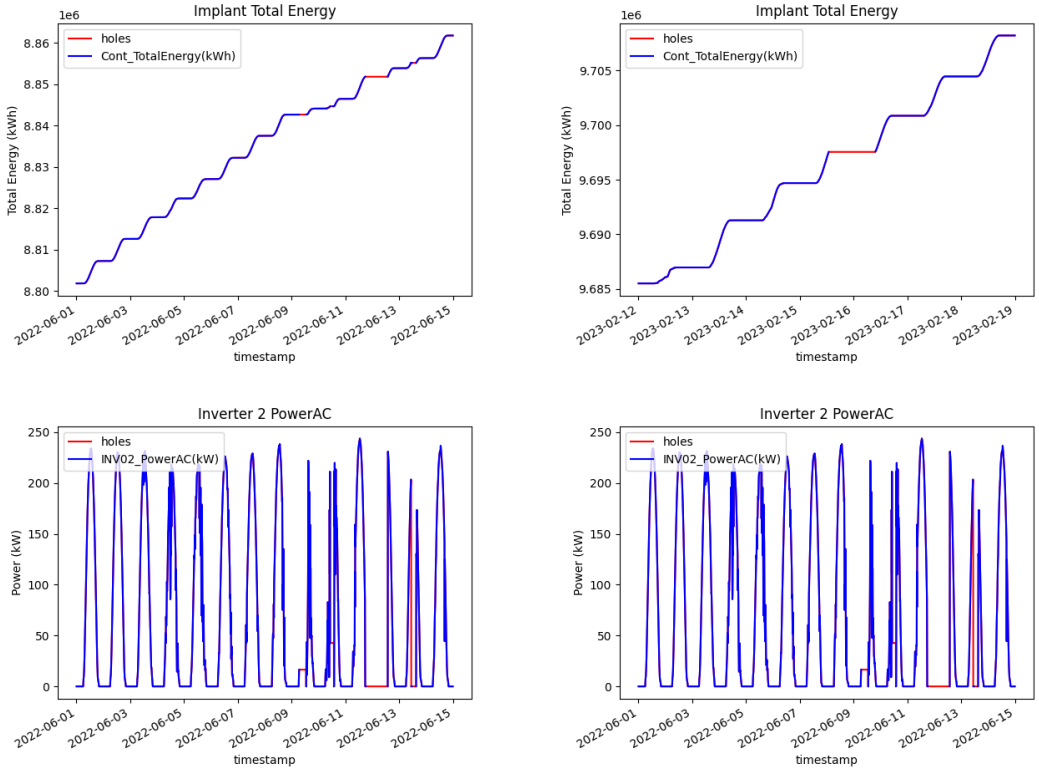


Figure 2.5: Some dataset "holes". The charts at the top refer to the Implant's Total Energy, while those at the bottom refer to the Inverter 2's Power. The charts on the left range from 01-06-2022 to 15-06-2022, while those on the right range from 12-02-2023 to 19-02-2023.

As we can see from Figure 2.5, there are certain periods within the dataset (highlighted in red) where data is missing, which we refer to as "holes". These data gaps can cause problems during model training (hindering the correct calculation of the loss function), and therefore, they need to be removed.

One possible approach for holes removal is to perform a *fill* operation: filling the gap with the first available non-null value. This tactic may be considered acceptable only if the time span involves just a few timestamps. However, if we are talking about several hours or even days, it significantly distorts the overall production and instantaneous power trends, resulting, especially in very unfortunate cases, with extremely abnormal production curves.

The solution we have adopted to address this problem is the removal of the entire day when a hole occurs. For example, if we have a data gap from 12-02-2023

23:00 to 13-02-2023 10:40, the days 12-02-2023 and 13-02-2023 will be completely removed. With this method, we lose some data, but as we will see later, the number of gaps is not extremely high, and this data loss is not debilitating. The following algorithm summarizes what has just been described.

Algorithm 5 Holes Removal Algorithm.

Require: dataset table
Ensure: `dataset` **is not** empty
  holes ← find all timestamp from `dataset` table, where there are some `Nans` inside the columns
  for each row **in** `dataset.rows` **do**
    if row.timestamp **is in** `holes` **then**
      drop `row` from `dataset` table
    end if
  end for

Timestamp
2022-06-09
2022-06-10
2022-06-11
2022-06-12
2022-06-13
2022-06-28
2022-06-29
2022-06-30
2022-08-26
2022-09-23
2022-10-06
2023-02-03
2023-02-15
2023-02-16
2023-03-26

Table 2.13: Timestamps deleted after running the Algorithm 5

2.2.4 Target Feature

For the deep learning models that we will introduce later, it could be very difficult learning how to predict the Total Missing Energy, as it is a curve that grows constantly, potentially to infinity. These models need to have features that vary within a limited range of values. To achieve this, we transformed the variable `Cont_TotalEnergy(kWh)` from cumulative to instantaneous values of produced energy, thereby limiting its values within a finite range. This new feature was added to the dataset with the name `target`.

Algorithm 6 Cumulative Energy to Instant Energy Algorithm.

Require: dataset table
Ensure: `dataset` **is not** empty
  target ← empty column
  for each (prev_val, val) **in** dataset["Cont_TotalEnergy(kWh)"] **do**
    diff ← val − prev_val
    append `diff` value to `target` column
  end for
  /* Now we have target column with first value missing */
  /* We can add a 0 as fist value because its night time */
  append (in head) 0 to `target` column
  merge `target` and `dataset`

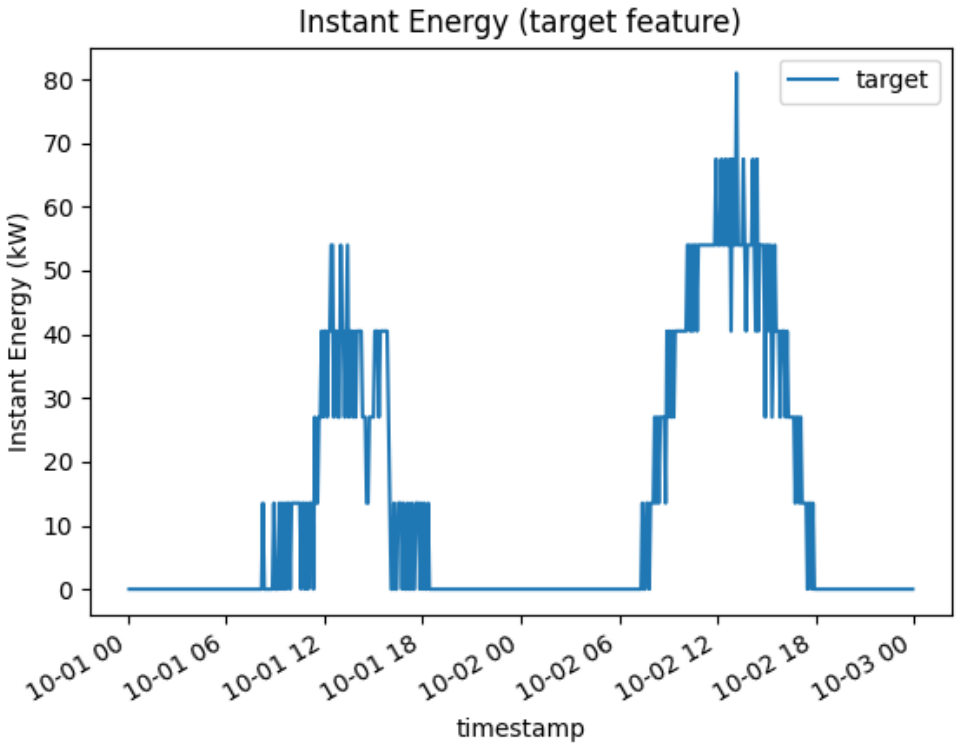


Figure 2.6: Instant Energy (`target` feature) 2 days plot.

2.2.5 Re-Sampling

Finally, after all the operations described above, we obtain a dataset with samples taken at 5-minute intervals. As we can see from Figure 2.6, this data appears to be quite noisy, which could pose challenges during the training phase. Moreover, most of the reports provided by photovoltaic plant operators are at a 15-minute frequency. To prevent potential issues and align with this standard, we have written and applied a re-sampling procedure.

Algorithm 7 15 minute Re-Sampling Algorithm.

Require: dataset table
Ensure: `dataset` **is not** empty
  for each feature **in** dataset.features **do**
    if feature **is** Cumulative **then**
      feature ← sample `feature` column using 15 min. sampling rate
    else
      feature ← aggregate values every 15 minutes and sum them to form a single value
    end if
  end for

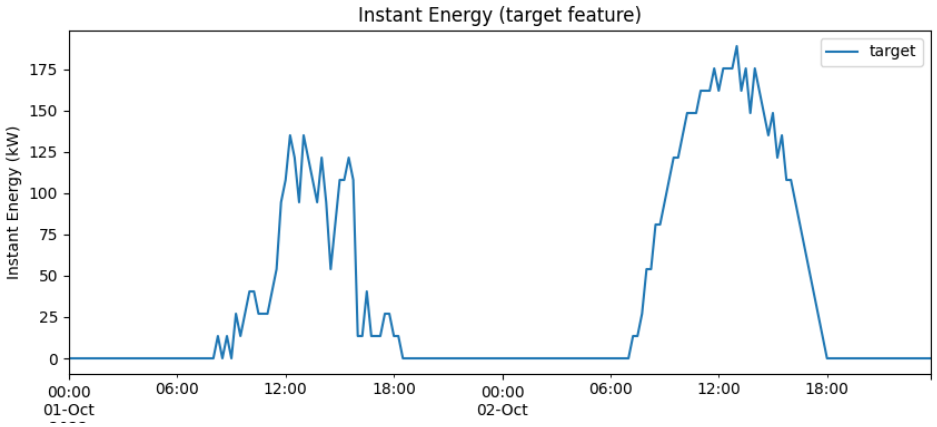


Figure 2.7: Instant Energy feature on 15 minute re-sampled dataset.

2.3 Feature Selection

After applying all the procedures described in the previous section, we find ourselves with a dataset composed of 764 columns. This high number of features would make it nearly impossible to train various models, both from a purely hardware perspective and due to the excessive information redundancy. In an attempt to minimize the number of features as much as possible, we applied and combined the results of two essential tools for feature selection: the *Correlation Matrix* [14] and the *Power Predictive Score* [6].

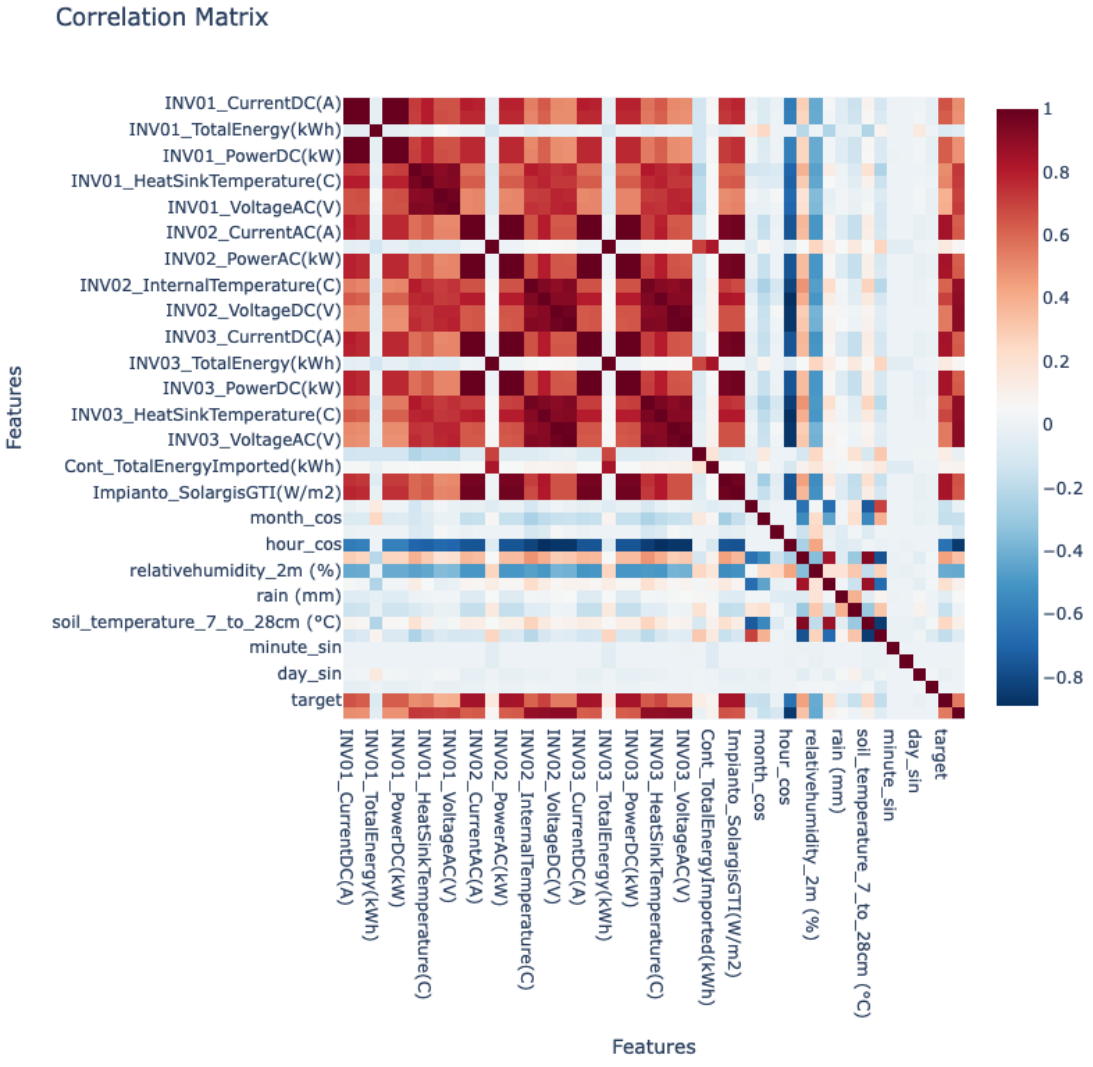


Figure 2.8: Correlation Matrix

As we can infer, even with a naive approach, all the information from the various String Boxes and Junction Boxes is perfectly summarized by the data provided by the various Inverters (see Section 1.2); a similar reasoning can be applied to the features coming from *Other* (those starting with `other_`), as they do not provide any useful information for energy estimation. Consulting the Correlation Matrix (Figure 2.8), we can see that this naive reasoning is confirmed because the features of the junction boxes have a high level of correlation with those of the Inverters. In general, we want to retain features that are minimally correlated with each other and aim to discard highly correlated ones, keeping only those that are more representative.

Applying this approach straightforwardly, however, is not the best choice because it could lead to the loss of important information. Let's consider an example: let's take into account the features `INV01_Power(kW)`, `INV02_Power(kW)`, and `INV03_Power(kW)`; by examining the correlation matrix, these three features are highly correlated, and, therefore, we should retain only one of them. However, all three are important for predicting our target. For instance, if Inverter 1 experiences any issues and is not operating at 100%, this will impact the total energy production, making all three of these features indispensable. Therefore, in conjunction with the correlation matrix, we have utilized the *Power Predictive Score* (PPS): an asymmetric, data-type-agnostic score that can detect linear or non-linear relationships between two or more columns [6]. The score ranges from 0 (no predictive power) to 1 (perfect predictive power) [6].

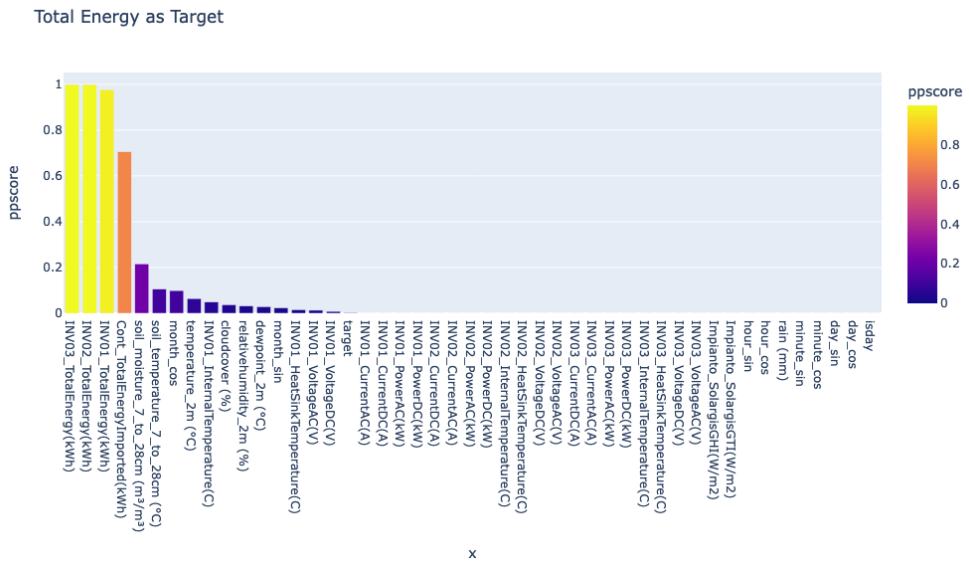


Figure 2.9: All features PPS using `Cont_TotalEnergy(kWh)` as target.

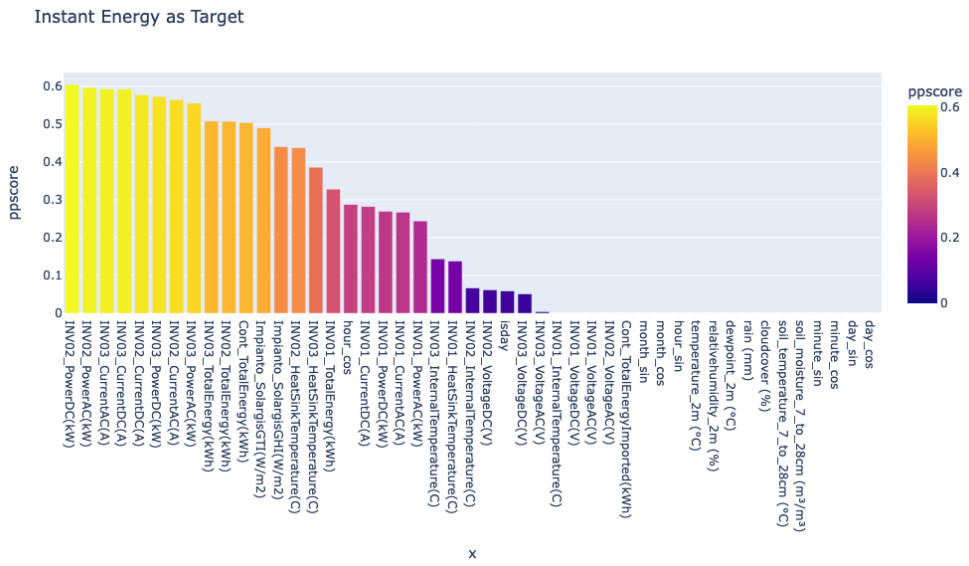


Figure 2.10: All feature PPS using Instant Energy (`target`) as Target.

As we can see in Figures 2.9 and 2.10, the features that perform better in predicting the instantaneous energy production trend are indeed those of the Inverters, confirming the reasoning described earlier. Finally, by combining the results obtained from the Correlation Matrix and the Power Predictive Score, we obtain a final set of 33 features.

INV01_PowerAC(kW)	INV03_TotalEnergy(kWh)	month_cos
INV01_PowerDC(kW)	INV01_HeatSinkTemperature(C)	min_sin
INV02_PowerAC(kW)	INV02_HeatSinkTemperature(C)	min_cos
INV02_PowerDC(kW)	INV03_HeatSinkTemperature(C)	target
INV03_PowerAC(kW)	Impianto_SolargisGHI(W/m2)	is_day
INV03_PowerDC(kW)	Impianto_SolargisGTI(W/m2)	
INV01_CurrentAC(A)	Cont_TotalEnergy(kWh)	
INV01_CurrentDC(A)	rain (mm)	
INV02_CurrentAC(A)	cloudcover (%)	
INV02_CurrentDC(A)	hour_sin	
INV03_CurrentAC(A)	hour_cos	
INV03_CurrentDC(A)	day_sin	
INV01_TotalEnergy(kWh)	day_cos	
INV02_TotalEnergy(kWh)	month_sin	

Table 2.14: All feature selected after this phase.

2.4 Dataset Splitting

After completing the Feature Selection phase (Section 2.3), we decided to divide the obtained dataset into three different sets:

- *Training*: It covers the period from 01-06-2022 to 28-02-2023. This set will be used during the training phase to allow the model to learn the plant's performance and how to predict energy trends during various gaps. It starts in June because there are some issues with the Inverter 1's features before that.
- *Validation*: It spans from 01-03-2023 to 31-03-2023. It will be used in the training phase to assess the presence of overfitting or if this phase might have failed.
- *Test*: It starts on 01-04-2023 and continues until 30-04-2023. This set will be used in the Evaluation phase to estimate the model's performance on a data period it has never seen before.

	Start	**End**	**Rows**
Training	01-06-2022	28-02-2023	24864
Validation	01-03-2023	31-03-2023	2880
Testing	01-04-2023	30-04-2023	2880

Table 2.15: Summary table of how the dataset was divided.

Chapter 3

Deep Learning Models

In this chapter, we will present three models based on different architectures: the first, our baseline, is based on a Multi-Layer Perceptron, the second model is based on a Recurrent Neural Network, and finally, the last one utilizes the newer and more powerful Transformer layers. We will describe in detail these architectures, the corresponding training phases, and any associated issues along with their strengths and weaknesses.

It's important to note that these models were implemented using only the PyTorch [16] library. Each run was preceded by setting a seed (the same for all runs of each model) to allow for comparisons and ensure the repeatability of the various operations. For training phase, the dataset created in the previous Chapter 2 was used and split into training, validation, and testing sets as described in Section 2.4. All of these phases were executed on a machine with 64 GB (2x32GB) DDR5 5600MT/s of RAM, an AMD Ryzen 9 7950X3D processor, and an Nvidia 4090 GPU.

3.1 MLP

In this section, we will introduce the first model, which we will refer to as the baseline model, based on a Multi-Layer Perceptron. We will analyze its architecture and the training phase.

3.1.1 Architecture

This neural network is designed to predict the instantaneous energy output over a period of exactly 2 days. To do this, it requires input data that includes the performance of the system (features selected during the Preprocessing phase, see Chapter 2) for exactly one day before and one day after the period in question.

This enables it to understand how the system is performing and, consequently, provide the energy output trend. The model consists of 5 main layers:

- **Input layer**: This is the first layer of the network. It takes two input tensors: *before* and *after.*, representing the day before and the day after the specific period we want to predict. They have the shape `[BATCH_SIZE, 96, 33]`, where 96 is the number of timestamps in our dataset that make up one day, and 33 represents the features obtained from the Data Preprocessing phase. These tensors are then flattened and concatenated, obtaining a layer with 6336 nodes, to be passed to the subsequent layer. The output of this layer goes through a Batch Normalization [10] layer, and the Rectified Linear Unit (ReLU) [5] activation function is used.
- **Hidden layers**: In total, there are 3 layers, each of which takes the output of the previous layer as input and reduces the number of neurons by approximately half. The 3 hidden layers are composed of 3069, 1527, and 986 nodes, respectively. Batch Normalization [10] is applied to the result, and the Rectified Linear Unit (ReLU) [5] is used as the activation function.
- **Output layer**: The final layer of our network, it takes the result from the previous layer and outputs the value of the Instantaneous Energy Produced during the specific period. It produces a tensor with a shape of `[BATCH_SIZE, 192, 1]`. The SoftPlus [5] function is used as the final activation function.

In Table 3.1 main characteristics of this model are summarized. We can note that the occupation memory is around 100 MB, so the model is relatively compact and it can run in any device.

Total Parameters (#)	26,543,400
Trainable Parameters (#)	26,543,400
Training Duration (s)	24.0
Model Size (MB)	101.3

Table 3.1: Baseline Model specification.

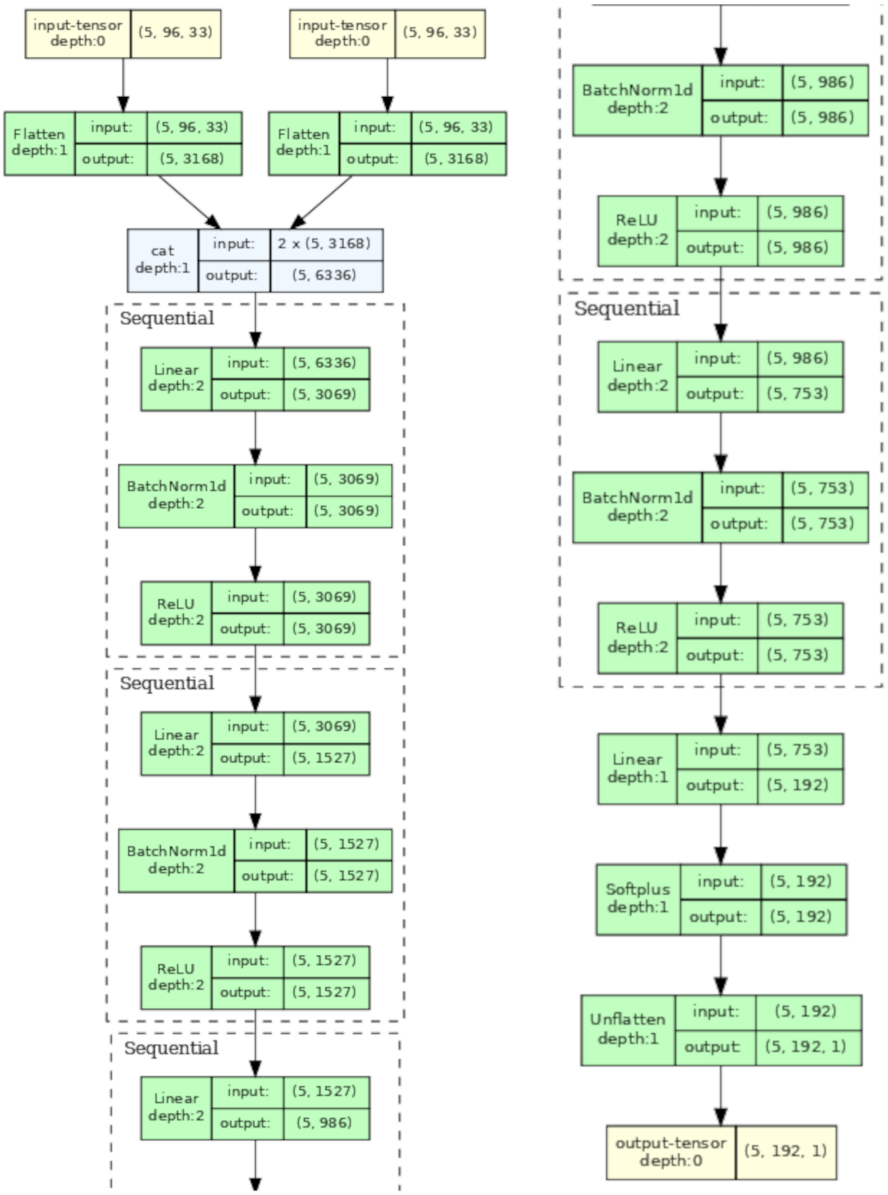


Figure 3.1: Beseline Model architecture visualization.

3.1.2 Training

The model was trained by artificially creating gaps of two days in length within the training dataset. These gaps were passed to the network in the format described earlier, and the output result was compared to the actual instantaneous energy produced during the gap. Unlike other prediction problems, imputation faces an extra challenge: it must precisely match the last values before the gap and the initial value after the gap. We obtain these performing a rescaling procedure after the output computation.

An *Early Stopping* [8] procedure was used to prevent training from continuing if the model was not improving its performance. A procedure, called *Save Best*, has also been integrated, which saves the model to a file whenever the Validation Loss improves. The validation dataset was applied in this phase to assess the learning progress at the end of each epoch.

Adam [11] was used as the optimizer, and L1Loss [2] (Equation 3.2) served as the loss function. We chose to set the batch size parameter to 10, the learning rate λ to 0.01, a maximum of 100 epochs, and a patience value of 20 for Early Stopping.

$$L = \{l_1, \dots, l_N\}^\top, \quad l_n = |x_n - y_n|, \tag{3.1}$$

$$\ell(x, y) = \text{mean}(L) \tag{3.2}$$

From the training phase plot shown in Figure 3.2, it's clear that the training phase concluded, but not without some issues. The values of the two loss functions don't seem to decrease significantly as the epochs progress, indicating that the model may not have generalized well. However, it's important to note that the two curves representing the training loss and validation loss don't diverge significantly but remain at a relatively constant distance from each other, suggesting that there is no overfitting.

From the graphs in Figure 3.3, it is apparent that the machine on which this model was trained was relatively underutilized. In particular, it's worth noting that GPU utilization did not exceed 30%, and GPU memory was barely utilized, staying at around 20%. This suggests that the model can be trained on less powerful machines as well. The model is also relatively lightweight in terms of memory, with just 100 MB (see Table 3.1), and the training time did not exceed 30 seconds. Inference time is extremely fast, taking less than a second.

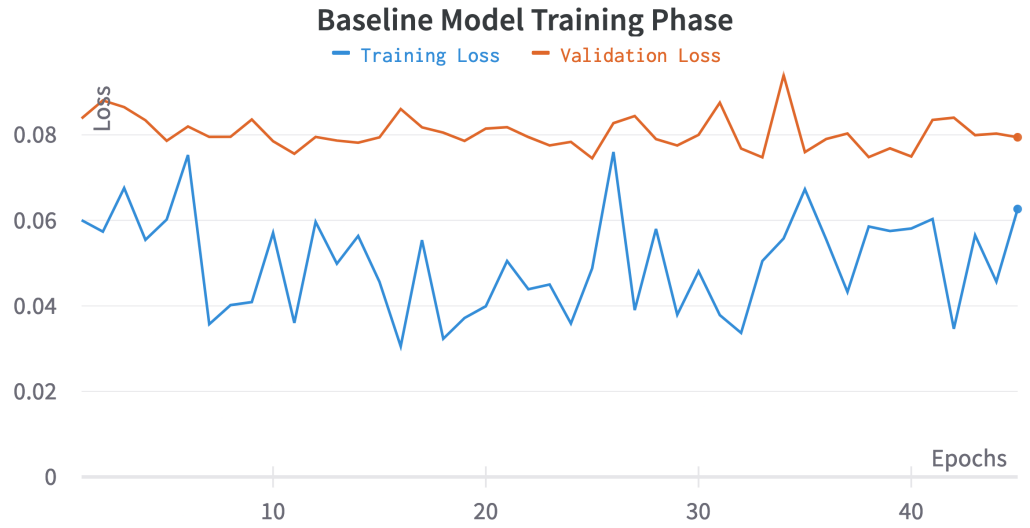


Figure 3.2: The chart displays the loss progression during the training phase. The blue line represents the Training Loss, while the orange line represents the Validation Loss.

Algorithm 8 MLP model Training Algorithm

Require: train/validation datasets; Baseline Neural Network Model
  Batch Size ← 10
  Learning Rate λ ← 0.01
  Epochs ← 100
  Patience ← 20
  loss ← L1Loss()
  Optimizer ← Adam Optimizer

  for each epoch **in** epochs **do**
    for each (batch_id, before, after, target) **in** train.next_batch() **do**
      train_prediction ← model(before, after) ▷ Model inference
      train_prediction ← $\frac{\text{train\_prediction} \cdot \sum \text{train\_prediction}}{\sum target}$ ▷ Area normalization
      train_loss ← loss(train_prediction, target)
      Optimizer step
      Back Propagation
    end for
    stop computing gradient
    for each (batch_id, vbefore, vafter, vtarget) **in** validation.next_batch() **do**
      val_prediction ← model(vbefore, vafter) ▷ Model inference
      val_prediction ← $\frac{\text{val\_prediction} \cdot \sum \text{val\_prediction}}{\sum vtarget}$ ▷ Area normalization
      val_loss ← loss(val_prediction, vtarget)
    end for
    check for Early Stopping
    check for Save Best Result
    start computing gradient
  end for

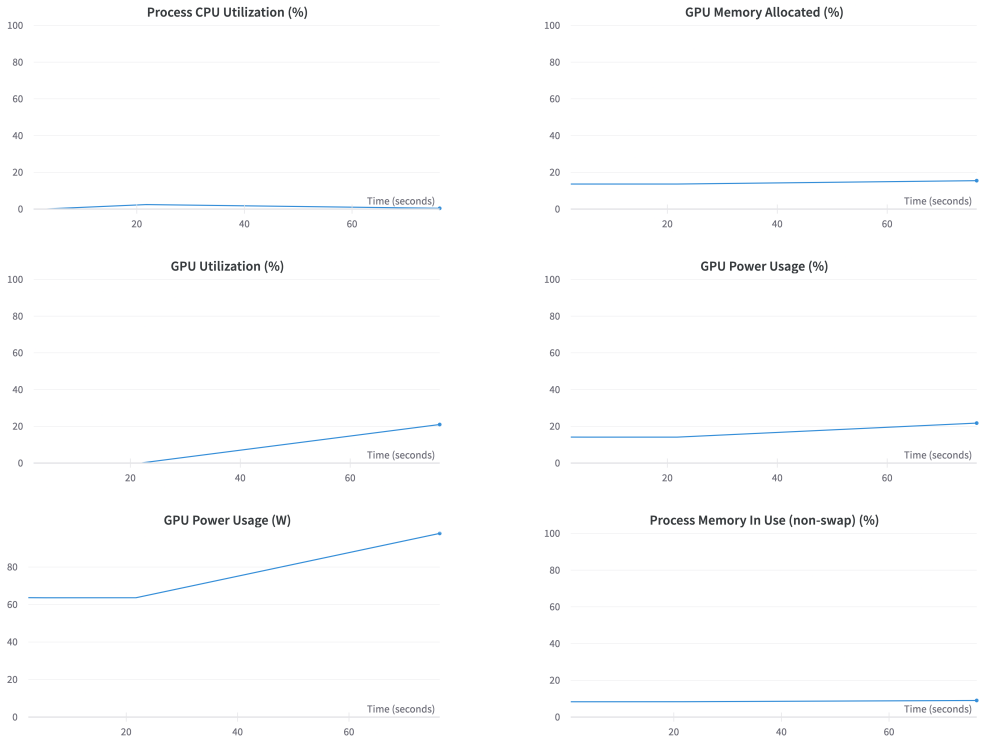


Figure 3.3: System resources utilized during the Training phase.

3.2 RNN

We will now introduce a second model based on Recurrent Neural Networks. We will examine its structure and analyze the training phase.

3.2.1 Architecture

This model is designed to make the most of the capabilities of Recurrent Neural Networks to predict the behavior of instant energy production during a variable-length gap period.

The required input format is quite similar to the previous one: a tensor named *before* containing the plant's status for a period before the gap, a tensor named *after* with information about the plant for a period after the gap, and an additional tensor named *future* containing features that are *always* available even during blackout periods. These features specifically include `Solargis`, `isday`, and information obtained from Open-Meteo. This last tensor should assist the model in prediction by helping it better understand the state of meteorological conditions and adapt instant production accordingly.

Now, let's display the main elements that compose it.

- **Input**: As described earlier, the model requires 3 input tensors: *before*, *after*, and *future*. The first two should contain plant's information from some period of time before and after the gap, while the last one will have weather features to support predictions, which can vary in length. An example of input could be `[BATCH_SIZE, 23, 33]` for *before*, `[BATCH_SIZE, 120, 33]` for *after*, and `[BATCH_SIZE, 79, 13]` for *future*.

- **Encoder**: This component's role is to identify the most important features described by the *before* and *after* tensors to understand how the plant is operating. These two tensors are passed to two different GRU [1] layers, which will analyze these time series and extract key information. The final output of each GRU will then be extracted and concatenated together to form the Hidden State for the *Decoder*.

- **Middle layer**: This layer consists of 2 Fully Connected Layers. It takes as input the result of the Encoding phase and reshapes it to the necessary format for input to the *Decoder*. For both layers, the ReLU [5] function is used as the activation function.

- **Decoder**: Its role is to reprocess the information obtained from the Encoder and predict the instant energy produced during the blackout. It consists of a GRU layer that takes the *future* tensor as input and uses the result of the Middle Layer as the hidden state.

- **Output Layer**: This is the final layer of this architecture, and its task is to transform the output from the Decoder into a tensor of shape `[Batch_Size, Gap_Len, 1]`, representing the instant energy produced by the plant during the gap. It consists of a Fully Connected Layer. Its output is then multiplied by the feature `isday` to ensure that the model's prediction is always zero during the night.

Total Parameters (#)	92,737
Trainable Parameters (#)	92,737
Training Duration (s)	49.0
Model Size (KB)	366.6

Table 3.2: RNN based model specification.

As we can immediately see from Table 3.2, the model turns out to be extremely compact, with only 92,737 parameters and a total file weight of just around 400 KB. This makes it extremely lightweight, and it could potentially be used in different architectures with varying computational capabilities.

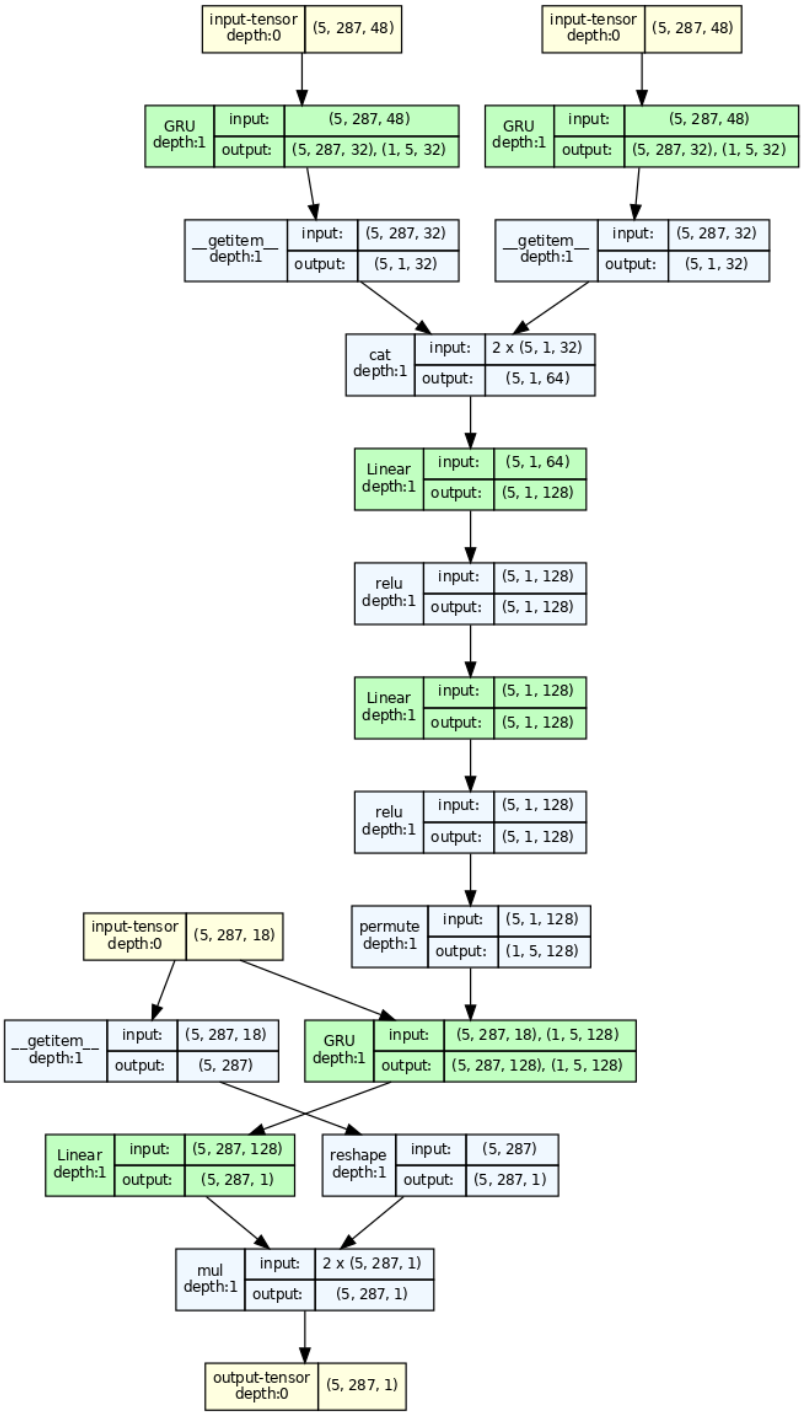


Figure 3.4: Recurrent Neural Network Based Model architecture visualization.

3.2.2 Training

The model was trained to learn the trend of the instant energy production curve of the plant during gaps of variable lengths, ranging from a minimum of 60 timestamps (corresponding to 15 hours) to a maximum of 4 days, using variable dimensions for both *before* and *after*, ranging from 2 timestamps (corresponding to 30 minutes) up to a maximum of 1 day. These gaps were artificially generated in the training dataset and provided to the model as described earlier, with attention to grouping gaps of the same length in batches to avoid complications during training.

In this case, an *Early Stopping* and *Save Best* procedure were used as well to ensure that the model always saves the best-performing model and to prevent resource waste.

The validation dataset was applied in this phase to conduct an initial and preliminary evaluation of the training process and highlight potential issues. A normalization procedure was also applied to scale the prediction area with respect to the ground truth area. This allows the model to learn the shape of the curve and not exceed the limits of energy production during the gap.

The Adam optimizer was used, and the L1Loss (Equation 3.2) was applied as the loss function. The batch size was set to 10, the learning rate (λ) was set to 0.0001, a maximum of 100 epochs was set, and the patience was set to 20.

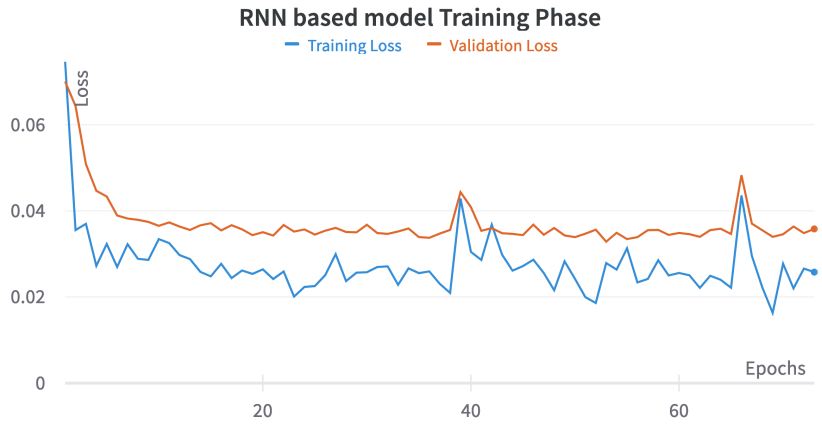


Figure 3.5: The chart displays the loss progression during the training phase.The blue line represents the Training Loss, while the orange line represents the Validation Loss.

From the plot shown in Figure 3.5, we can see that the training phase was successful, and after an initial descent, the loss values remained relatively constant

without displaying any abnormal trends. The statistics presented in Figure 3.6 reveal that the machine at our disposal was not fully utilized, suggesting that this architecture can be trained and used on less powerful computers than the one we have. Additionally, the model appears to be very lightweight, both in terms of the relatively small number of parameters and its weight, which doesn't exceed 400 KB. The inference time does not even approach one second, making it extremely fast in execution.

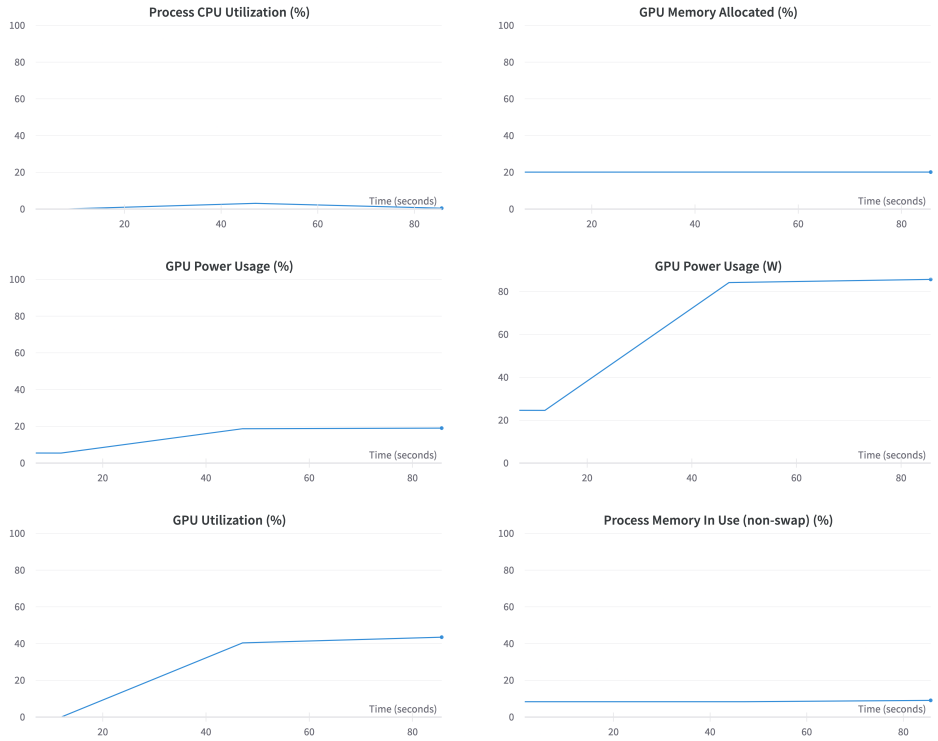


Figure 3.6: System resources utilized during the Training phase.

Algorithm 9 RNN model Training Algorithm

Require: train/validation datasets; Baseline Neural Network Model
  Batch Size ← 10
  Learning Rate λ ← 0.01
  Epochs ← 100
  Patience ← 20
  loss ← L1Loss()
  Optimizer ← Adam Optimizer
  Min Gap size ← 1 day
  Max Gap size ← 4 days

  for each epoch **in** epochs **do**
    for each (batch_id, before, after, future, target) **in** train.next_batch() **do**
      train_prediction ← model(before, after, future) ▷ Model inference
      train_prediction ← $\frac{\text{train\_prediction} \cdot \sum \text{train\_prediction}}{\sum target}$ ▷ Area normalization
      train_loss ← loss(train_prediction, target)
      Optimizer step
      Back Propagation
    end for
    stop computing gradient
    for each (batch_id, vb, va, vf, vt) **in** validation.next_batch() **do**
      val_prediction ← model(vb, va, vf) ▷ Model inference
      val_prediction ← $\frac{\text{val\_prediction} \cdot \sum \text{val\_prediction}}{\sum vt}$ ▷ Area normalization
      val_loss ← loss(val_prediction, vt)
    end for
    check for Early Stopping
    check for Save Best Result
    start computing gradient
  end for

3.3 Transformers

Now, we will introduce a third model with an architecture based on Transformers[23]. We will analyze its layers, the training phase, highlighting the differences compared to the previous ones, especially in the construction of the input.

3.3.1 Architecture

The input and operation of this model are slightly different from what we have seen previously. This model requires only one tensor as input, representing the data for an entire week, with the `target` feature containing a variable period of values set to "-1". This special character, called *placeholder*, indicates to the model the presence of a gap that needs to be filled. This gap can have an extremely variable length, ranging from a minimum of 2 timestamps to the equivalent of one-third of a week, and it can be positioned anywhere within the week. This is a very interesting aspect of the model, as it allows it not only to detect the presence of gaps and fill them easily but also enables the immediate construction and preparation of the input.

The model's objective is to learn to predict the entire week, including the gap. Through a masking system, we can then extract only the values of interest. Let's now delve into the specific elements that characterize and make up this model:

- **Input**: As described earlier, the input is a tensor containing one week of data with the target feature that includes a period of -1 (representing the gap).
- **Convolution**: The input goes through two convolutional layers to extract and understand the most important and representative features of the time series. Each layer undergoes batch normalization, and the Gaussian Error Linear Units (GELU) [9] function is used as the activation function.
- **Positional Encoder**: To help the model understand the temporal sequence and the order of the different time series, a positional encoder called Time Absolute Position Encoding (tAPE)[7] is implemented. It incorporates the series length and input embedding dimension for absolute position encoding [7].
- **Transformer Encoder**: After performing positional encoding on the output of the convolutional layers, it is passed to the transformer. This transformer [23] consists of 2 transformer layers, each of which contains 8 heads for multi-head attention [23], 64 layers for the feed-forward part, a dropout rate of 0.1, and GELU as the activation function.
- **Output**: This is the final layer of the network, responsible for reshaping the transformer's output to the required dimension for use. It is a feed-forward layer with an output dimension of 1, and it applies the Softplus [5] function as the activation function.

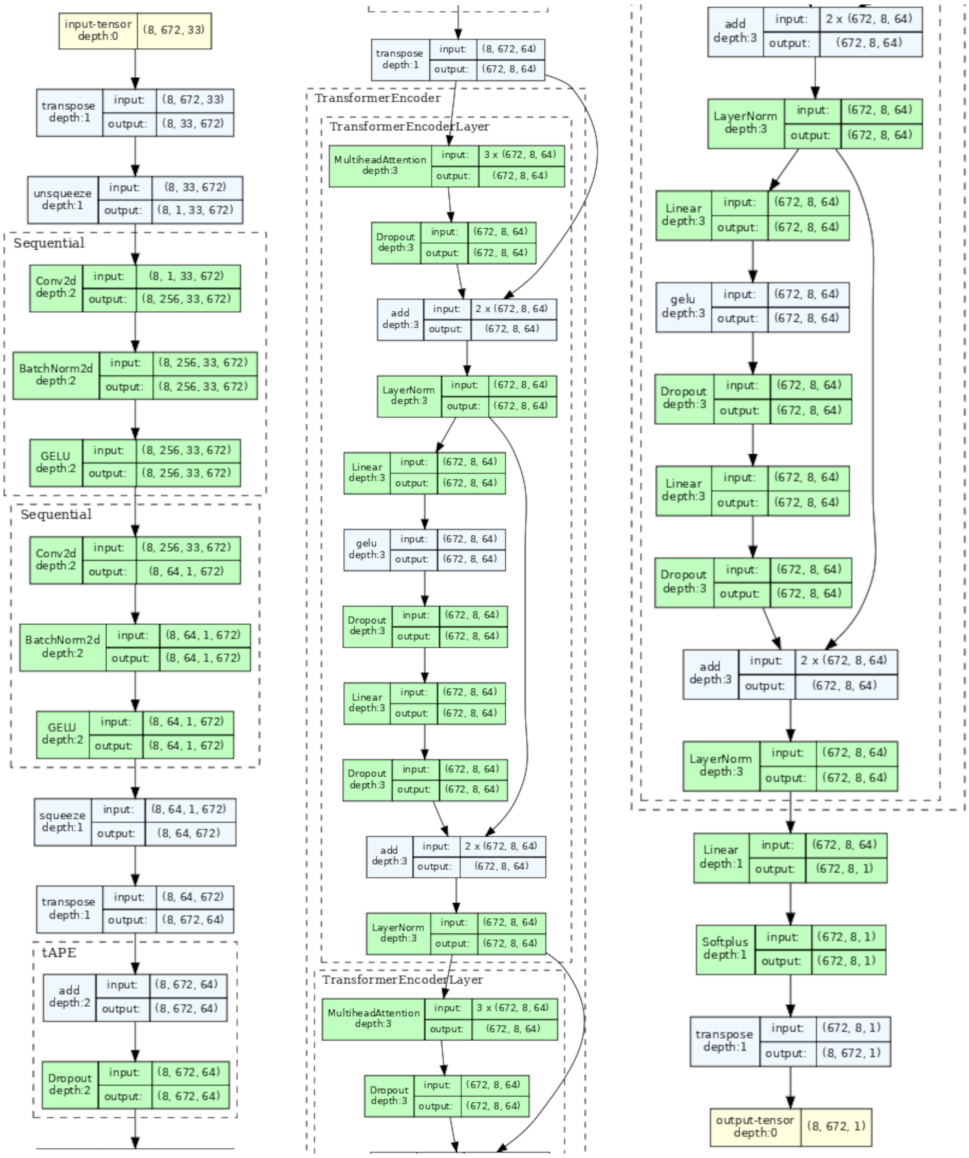


Figure 3.7: Transformer Based Model architecture visualization.

3.3.2 Training

The model was trained using the input format discussed earlier, generating gaps artificially in the training dataset, while also ensuring the creation of a gap mask. This mask is an array of the same length as the input time series, composed of two elements: 1 to indicate the gap timestamps and 0 for regular timestamps. This allows obtaining only the predicted values within the gap, which are used for evaluating the loss function.

In this case, a procedure for rescaling the area was applied to ensure that the model learns the gap curve's behavior. The L1Loss (Equation 3.2) function was employed as the loss function, and AdamW [12] was used as the optimizer with a learning rate (λ) set to 0.00001. The learning rate is then adjusted during training using a cosine annealing scheduler [15] [13] to provide optimal conditions for the model during this phase.

Additionally, a validation dataset was used to assess the model's performance during training and to enable the *Early Stopping* and *Save Best* procedures, which were also used in this case.

Total Parameters (#)	594,177
Trainable Parameters (#)	594,177
Training Duration (s)	900.0
Model Size (MB)	2.4

Table 3.3: Transformer based model specification.

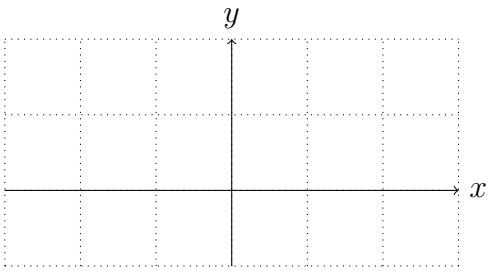


Figure 3.8: Gaussian Error Linear Units (GELU). $GELU(x) = x * \Phi(x)$

From the plot shown in Figure 3.9, we can observe that the training and validation loss curves, after an initial descent, remain relatively constant, and most importantly, they do not seem to diverge from each other. This indicates that the training phase has concluded successfully, with the validation loss value being lower by 0.01 compared to the RNN-based model.

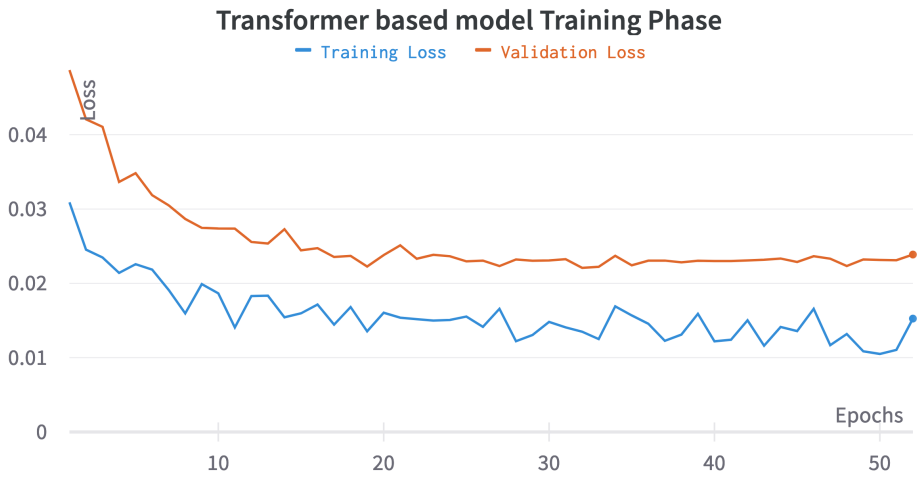


Figure 3.9: The chart displays the loss progression during the training phase.The blue line represents the Training Loss, while the orange line represents the Validation Loss.

From the plots in Figure 3.10, which show the machine's usage during the training phase, and from Table 3.3, it can be inferred that the model utilizes almost all of the available hardware resources, suggesting that it is computationally intensive and might be challenging to train on less powerful machines. However, it's essential to note that the inference time is extremely fast, not exceeding one second, and the model file size is very compact, with a dimension of approximately 2 MB.

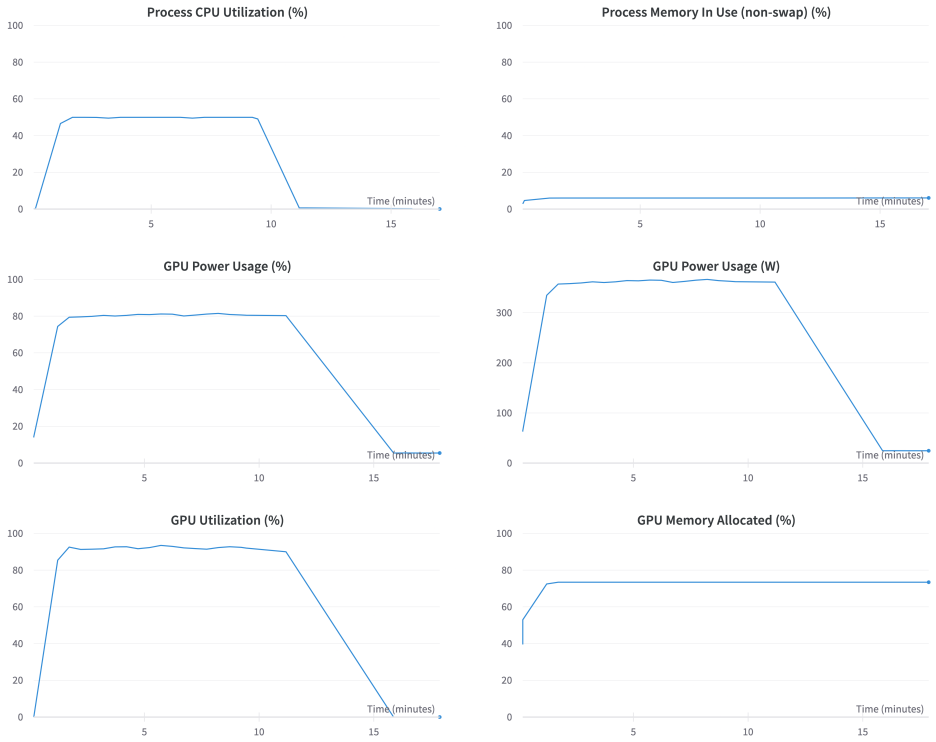


Figure 3.10: System resources utilized during the Training phase.

Algorithm 10 Transformer based model Training Algorithm

Require: train/validation datasets; Transformer based Neural Network Model

Batch Size $\leftarrow 8$
Learning Rate $\lambda \leftarrow 0.00001$
Epochs $\leftarrow 100$
Patience $\leftarrow 20$
loss $\leftarrow$ L1Loss()
Optimizer $\leftarrow$ AdamW Optimizer
Scheduler $\leftarrow$ CosineAnnealingScheduler
Min Gap size $\leftarrow 2$ timestamps
Max Gap size $\leftarrow \frac{\text{week length}}{3}$ timestamps

for each epoch **in** epochs **do**
  for each (batch_id, src_data, tgt_data, mask) **in** train.next_batch() **do**
    train_prediction $\leftarrow$ model(src_data) ▷ Model inference
    train_prediction $\leftarrow$ train_prediction $\cdot$ mask
    train_prediction $\leftarrow \frac{\text{train\_prediction} \cdot \sum \text{train\_prediction}}{\sum tgt\_data}$ ▷ Area normalization
    train_loss $\leftarrow$ loss(train_prediction[mask = 1], tgt_data[mask = 1])
    Optimizer step
    Back Propagation
  end for
  Scheduler step
  stop computing gradient
  for each (batch_id, vs, vt, vm) **in** validation.next_batch() **do**
    val_prediction $\leftarrow$ model(vs) ▷ Model inference
    val_prediction $\leftarrow$ val_prediction $\cdot$ vm
    val_prediction $\leftarrow \frac{\text{val\_prediction} \cdot \sum \text{val\_prediction}}{\sum vt}$ ▷ Area normalization
    val_loss $\leftarrow$ loss(val_prediction[vm = 1], vt[vm = 1])
  end for
  check for Early Stopping
  check for Save Best Result
  start computing gradient
end for

Chapter 4

Experimental Results

In this final chapter, we will analyze the results of the models previously described by comparing their respective test phases, highlighting any issues and strengths of the model. We will test their performance using the Test Set, as described in Section 2.4, and evaluate them using different metrics. Among these metrics, we will employ MAE (Mean Absolute Error) and MAPE (Mean Absolute Percentage Error) [20] to highlight the difference between the model's predictions and the ground truth and the R^2 (R-squared) [20] index to understand how well the model-predicted curve approximates the real one. Here's a brief introduction to the indices we will use:

- **Mean Absolute Error (MAE)**: This metric represents the average point-wise error that the model makes compared to the ground truth. It is expressed in kW.
- **Mean Absolute Percentage Error (MAPE)** [20]: It indicates the error of the model relative to the ground truth, measured in percentage. The calculation of this metric in our case can be quite problematic due to the presence of many values close to zero or exactly zero. Therefore, the calculation is performed by only considering non zero values. Specifically, in addition to the standard MAPE, we will employ a refined version named as MAPE@k. In this variant, the metric is calculated exclusively for values in the ground truth greater than k. This choice is driven by the distortion introduced in MAPE values when incorrect predictions are linked to the lowest values. Such predictions might exhibit low absolute errors, yet correspond to significantly high percentage errors.
- R^2 **(R-squared)**: This index quantifies how well the prediction curve approximates the ground truth. It helps us understand how well the model

approximates the trend of the reference instant prediction. This varies between 0.0, the worst case, to an ideal value of 1.0 [20].

4.1 Testing Procedure

All models will be tested using the same test set introduced in the previous Chapter 2. It's important to note that this dataset consists of 33 features, which have been carefully selected as described in Section 2.3, and it contains all the data sampled at 15-minute intervals during the month of April (2880 records). Each model will be tested at least once on this set, according to its suitable input format and capabilities.

The testing procedure applied to the MLP and RNN models for the purpose of comparison (with a fixed 2-day gap size) will be conducted as follows: a 4-day time window will be applied to the testing dataset, shifting forward by 1 day at each iteration. This approach will generate a 2-day gap to predict, along with the corresponding day before and after. For example, consider the time interval from April 2nd to April 5th (time window). In this case, the gap to predict includes all the data of April 3rd and April 4th, while April 2nd and April 5th are used, respectively as the data before and after the gap.

For the Transformer model, the testing procedure will be similar, with the only difference being the size of the time window. In the case of the Transformer, a 1-week time window will be used, and it will shift forward by 1 day at each iteration due to the model's input format requirements. This approach will ensure that a 2-day gap with relevant data before and after is available for the Transformer model as well.

4.2 MLP based model Evaluation

Analyzing the data presented in Tables 4.2 and 4.1, it can be confirmed that the performance of this model is quite limited. The R^2 index hovers around 0.60, with an average of 0.69 and a standard deviation of 0.17. This indicates that the target curve is not well approximated by the model's output. While the *MAE* appears to be relatively low (it's important to consider that the nighttime period significantly lowers the average, as the difference between target and prediction is zero during that time) the values of *MAPE* and *MAPE@20* are very high, at times even reaching 100% error, with an average value of 71% and a standard deviation of 28% for MAPE and 53.01% with a standard deviation of 20.11% for MAPE@20.

	MAE (kW)	**MAPE (%)**	**MAPE@20 (%)**	**R^2**
AVG	14.11	70.98	53.01	0.69
STD. DEV	3.81	27.99	20.11	0.17

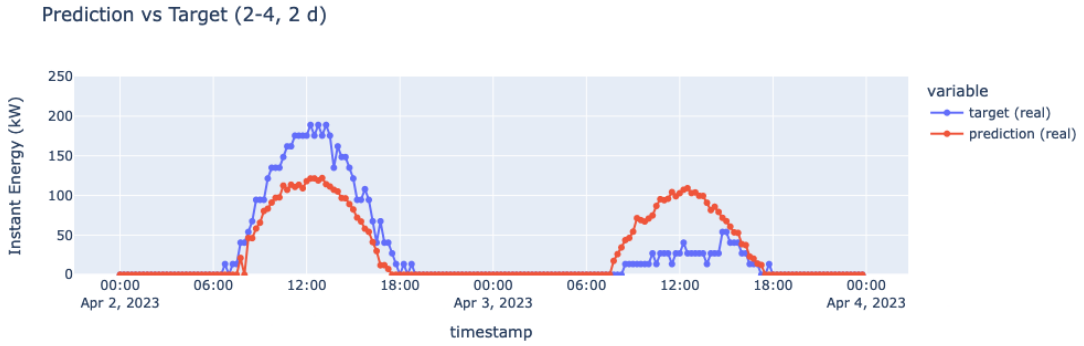
Table 4.1: MLP based model Global metrics.

Analyzing the plots presented in Figure 4.1, we can visually confirm that the model did not learn well how to predict the trend of instant energy production from the plant. Plot 4.1(a) clearly shows how the model's prediction is very timid and fails to grasp the possibility that some days may be more productive than others. In plot 4.1(b), we can see how the model almost approximates the second day but makes a significant mistake on the first, failing to understand the potential for production spikes during the day. In 4.1(c), on the other hand, we can see how the model's output manages to understand the ground truth's behavior quite well. In fact, we have an R^2 index value of 0.96, with values for MAE, MAPE, and MAPE@20 that are not excessively high compared to those seen previously. These claims are supported by the data shown in Table 4.2.

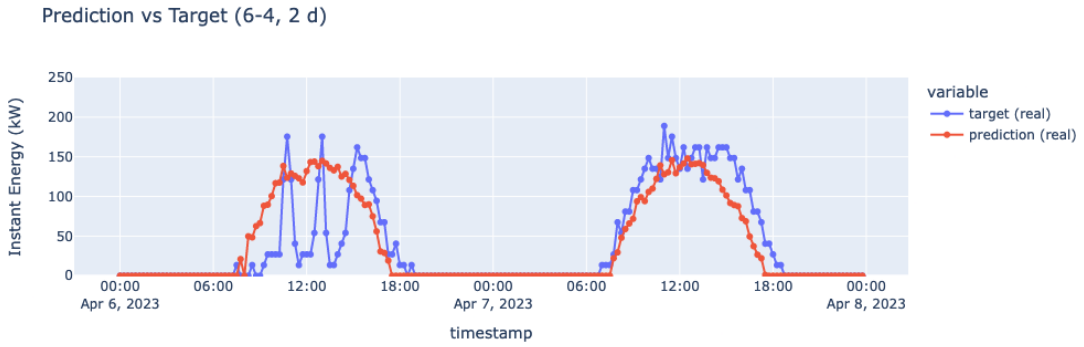
It's also evident that the model struggles to handle the day/night cycle, often ending production much earlier than it should. However, it consistently predicts zero energy production during the night, which is a positive aspect.

	Gap size	**MAE (kW)**	**MAPE (%)**	**MAPE@20 (%)**	**R^2**
Fig 4.1 (a)	96	18.63	128.47	82.27	0.64
Fig 4.2 (a)	96	6.90	64.97	45.58	0.52
Fig 4.2 (b)	96	17.99	109.78	85.16	0.42
Fig 4.1 (b)	96	20.07	113.99	88.56	0.62
Fig 4.1 (c)	96	10.66	33.84	39.23	0.92

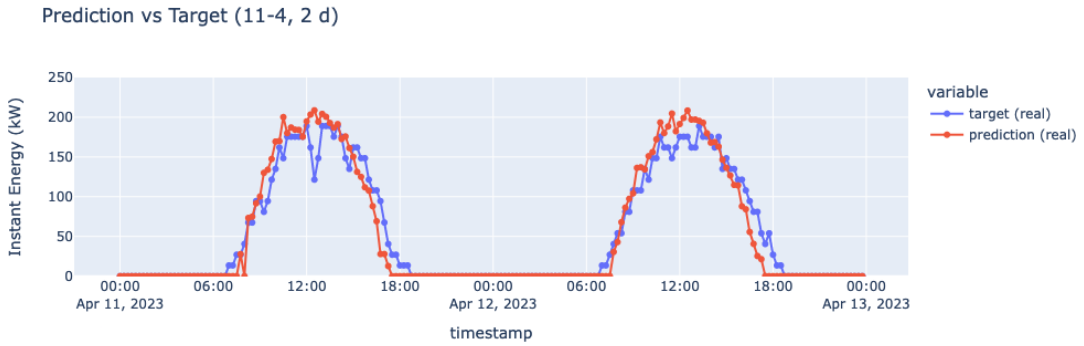
Table 4.2: The table displays the values of MAE (Mean Absolute Error) [20], MAPE (Mean Absolute Percentage Error) [20], MAPE@20 and the R^2 (R-squared) [20] index applied to the model predictions shown in Figures 4.1 and 4.2.



(a)



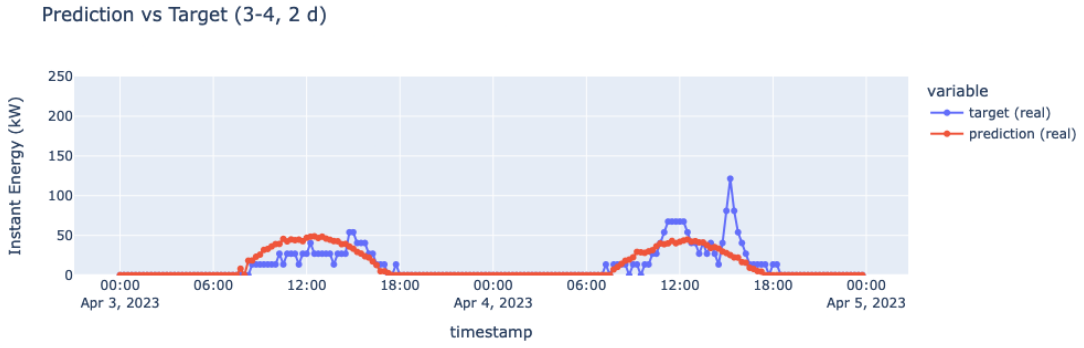
(b)



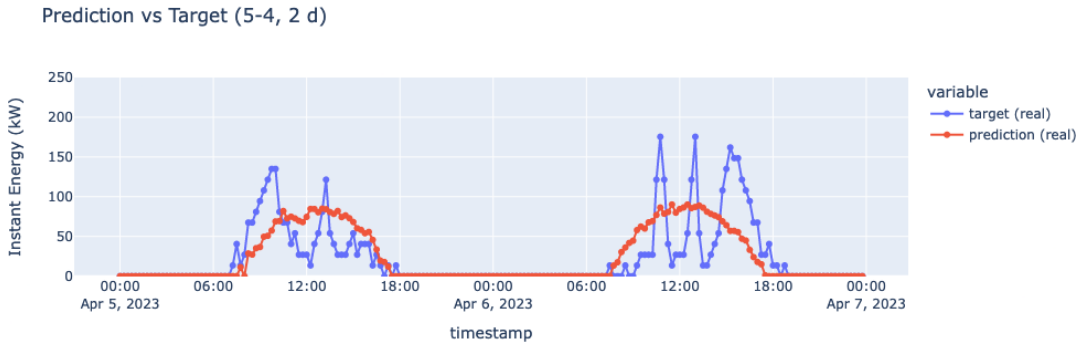
(c)

Figure 4.1: The figure displays three predictions made by the model, where you can observe the network's output (in red) and the ground truth (in blue). The model was tested on two gaps, each covering a period of 2 days. The first gap (a) spans from 02-04-2023 to 04-04-2023, the second gap (b) spans from 06-04-2023 to 06-08-2023 and the third starts from 11-04-2023 to 12-04-2023.

Figure 4.2 highlights the significant limitations of the model, demonstrating how it fails to identify potential peak values in production and how the predicted day curves are consistently similar to each other. It's also important to note that the model can only work with gaps of a fixed size, and if a gap of a larger size is encountered, it would be necessary to retrain the model.



(a)



(b)

Figure 4.2: The figure displays two model predictions (in red) for two-day gaps and their respective ground truth (in blue).

In conclusion, as expected considering the straightforward implementation and training process, this model is not suitable for effectively addressing the problem of data imputation due to the extremely high values of MAE, MAPE and MAPE@20, as well as the low value of the R^2 index.

We also made variations to the model's structure, reducing the number of layers (from 5 to 3) or increasing them (from 5 to 9), increasing the number of nodes in the various layers, changing the loss function to Mean Squared Error (MSE), and adding dropout operations between the layers. All of these operations were also

combined to try to obtain a more complex and comprehensive architecture, but with no success. From the different tests conducted, we found very similar results, with an average R^2 value ranging from 0.69 to 0.71.

Hence, considering also these additional tests, we can conclude that an architecture based on MLP will not be able to solve the problem of imputing the instant energy production of photovoltaic plants.

4.3 RNN based model Evaluation

Taking into account the data in Tables 4.3-4.6, it can be confirmed that the model based on Recurrent Neural Network introduced in Section 3.2 is highly performant. The average MAE value is 6.92 kW±1.74, which is relatively low, while the average R^2 index is 0.92±0.05, and it almost always approaches the ideal value of 1. Furthermore, the average MAPE value is 29.90%±9.45, and the average MAPE@20 is 23.58%±7.67, which is a significant improvement compared to the previous model.

	MAE (kW)	**MAPE (%)**	**MAPE@20 (%)**	**R^2**
AVG	6.92	29.90	23.58	0.92
STD. DEV	1.74	9.45	7.67	0.05

Table 4.3: Global metrics for the RNN-based model calculated on gaps of varying sizes.

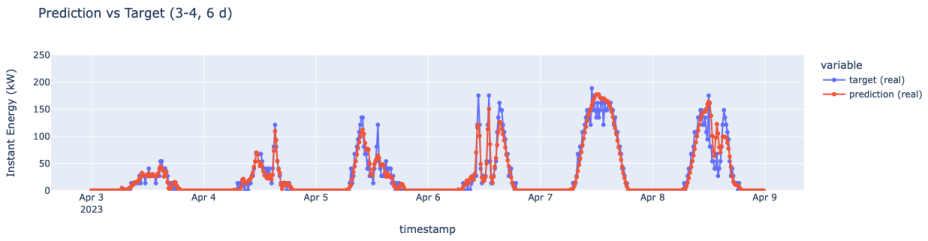
Analyzing some model predictions as shown in Figure 4.3, we can observe how the real instant energy production curves (displayed in blue) are closely approximated by the model (red curves). The model effectively understands the plant's behavior, even managing to predict production spikes. We can also see that energy production is consistently zero during the night, and it adeptly captures the day/night cycle by gradually reducing production as sunset approaches.

In plot 4.3(a), we can see a 4-days gap from 02-04-2023 to 06-04-2023. The first two days are predicted very well, while in the last part of the plot, we can see that a production spike was not detected. For this period we have a quite high R^2 index of 0.96, 4.72 kW as MAE and 28.12 % for the MAPE, which are quite good values. In plot 4.3(b), we can observe a 1-day gap on 04-04-2023. We notice that the overall trend is almost entirely approximated correctly, except for some time intervals around 12:00, indeed, we have slightly better results with an MAE of 3.60 kW and a MAPE of 27.15%. The last plot 4.3(c) is related to a 3-day gap, and we can see that the first two days are approximated well, while in the last day, the production spikes are identified but with values not entirely similar to those of

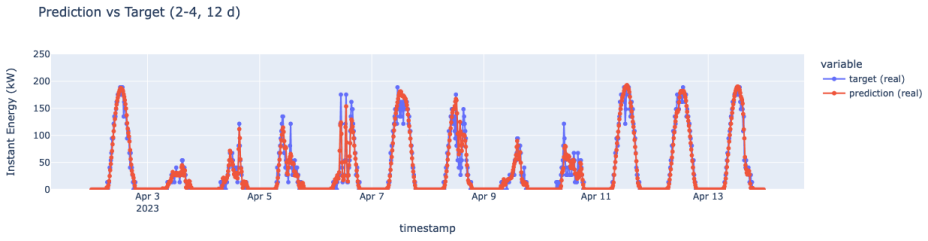
the ground truth. All the results presented so far are consistent with the average values shown in Table 4.3 demonstrating that the model returns fairly consistent outputs among them.

	Gap size	**MAE (kW)**	**MAPE (%)**	**MAPE@20 (%)**	**R^2**
Fig. 4.3(a)	384	4.72	28.12	19.59	0.96
Fig. 4.3(b)	96	3.60	27.15	14.79	0.93
Fig. 4.3(c)	288	6.61	20.68	18.16	0.96

Table 4.4: The table displays the MAE, MAPE, MAPE@20 and R^2 values applied to some model predictions during the testing phase, which are shown in Figure 4.3.



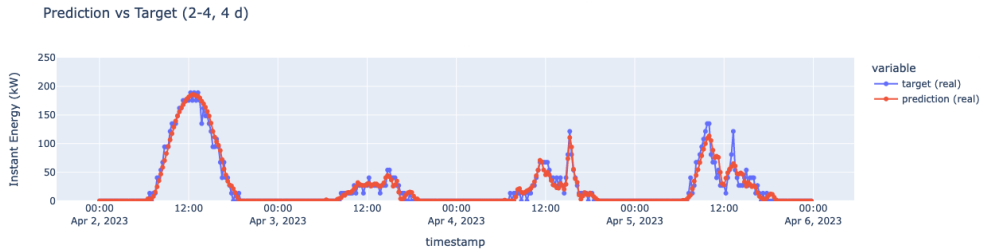
(a)



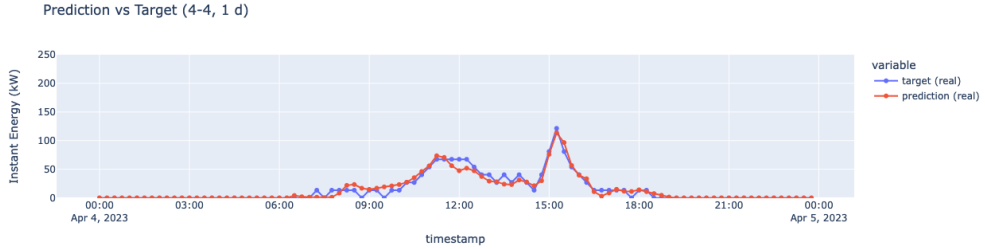
(b)

Figure 4.4: The plots depict two model predictions for gaps that exceed the maximum limit of days set during the training phase. The first one (a) shows a 6-day gap, while the second one (b) presents a 12-day gap.

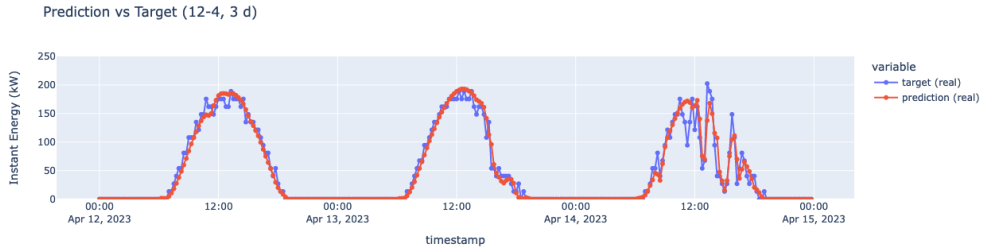
It is interesting to note how the model still performs well even when presented with gaps that exceed the maximum length set during training. In Figure 4.4, two plots are shown: 4.4(a) represents a 6-day gap (two days longer than the training length), and 4.4(b) a 12-day gap (eight days longer than the training length).



(a)



(b)



(c)

Figure 4.3: In the figure, three model predictions (in red) are shown alongside the ground truth (in blue) for gaps of varying sizes. The tensors *before*, *after*, and *future* have variable dimensions, as described earlier.

	Gap size	MAE (kW)	MAPE (%)	MAPE@20 (%)	R^2
4.4(a)	576	6.56	30.94	25.67	0.92
4.4(b)	1152	5.99	27.50	21.96	0.95

Table 4.5: The table displays the MAE, MAPE, MAPE@20 and R^2 values applied to the model predictions shown in Figure 4.4.

Gap Size	AVG MAE (kW)	AVG MAPE (%)	AVG MAPE@20 (%)	AVG R^2
12 days	6.64 ± 0.79	25.66 ± 5.90	19.59 ± 4.25	0.95 ± 0.02
6 days	6.85 ± 0.93	27.44 ± 6.78	21.30 ± 4.83	0.94 ± 0.02
2 days	6.86 ± 1.87	28.83 ± 10.02	21.02 ± 8.67	0.92 ± 0.06
< 60 ts	8.93 ± 2.22	36.14 ± 4.40	34.13 ± 5.20	0.86 ± 0.07

Table 4.6: Global results for the RNN-based model with fixed gap size. The table provides the average values along with their respective standard deviations.

Equally positive results are obtained for gaps that exceed the maximum training phase length of 4 days. From Table 4.6, we can see that for the 12, 6 and 2-day gaps, the average MAE, MAPE, MAPE@20 and R^2 index are quite similar compared to the values shown in Table 4.3, some are slightly better, while others are slightly worse, but they remain stable and confirm the values shown earlier. Clearly, we see a drop in performance with gap sizes shorter than 60 timestamps, as evident from the increase in MAE, MAPE and MAPE@20 and the decrease in the R^2 index.

In conclusion, it can be affirmed that this model effectively harnesses the potential offered by Recurrent Neural Networks and excels in predicting the trend of instant energy production during data gaps, even for gaps spanning multiple days.

4.4 Transformer based model Evaluation

Analyzing the results of the evaluation phase presented partly in Table 4.7 and in the graphs in Figure 4.6, we can see that this model is very effective in predicting instantaneous energy during gaps of varying sizes. It excels in this task, whether for gaps of nearly a day or up to three days. We can observe that the R^2 index in these cases is consistently high, almost always hovering around the value of 0.97, with relatively low MAE, MAPE and MAPE@20 values.

Examining the global metrics reported in Table 4.8(a), we can confirm these findings. It shows an extremely low average MAE of 3.83 kW with a standard

deviation of 1.40 kW, an average MAPE of 18.49% with a standard deviation of 7.99%, an average MAPE@20 of 11.29±4.73 and an average R^2 value of 0.97±0.05. This demonstrates how the model effectively approximates the ground truth with very low errors.

	Gap size	**MAE (kW)**	**MAPE (%)**	**MAPE@20 (%)**	**R^2**
Fig. 4.5(a)	265	5.02	22.38	15.98	0.98
Fig. 4.5(b)	40	7.24	11.72	10.80	0.96
Fig. 4.5(c)	189	3.14	25.11	12.74	0.99
Fig. 4.5(d)	87	4.02	24.40	14.70	0.96

Table 4.7: The table displays the values of MAE (Mean Absolute Error), MAPE (Mean Absolute Percentage Error), MAPE@20 and the R^2 (R-squared) index applied to some model predictions made during testing phase shown in Figures 4.6.

variable gap size

	AVG
AVG MAE (kW)	3.83 ± 1.40
AVG MAPE (%)	18.49 ± 7.99
AVG MAPE@20 (%)	11.29 ± 4.37
AVG R^2	0.97 ± 0.05

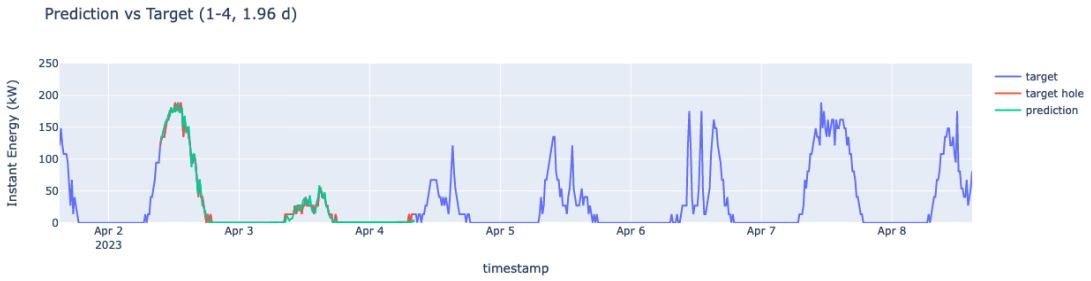
(a)

2 days gap size

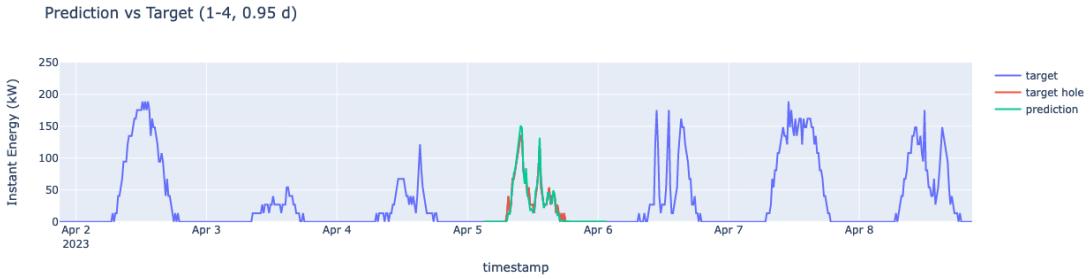
	AVG
AVG MAE (kW)	3.76 ± 0.39
AVG MAPE (%)	18.14 ± 6.76
AVG MAPE@20 (%)	11.18 ± 3.33
AVG R^2	0.98 ± 0.02

(b)

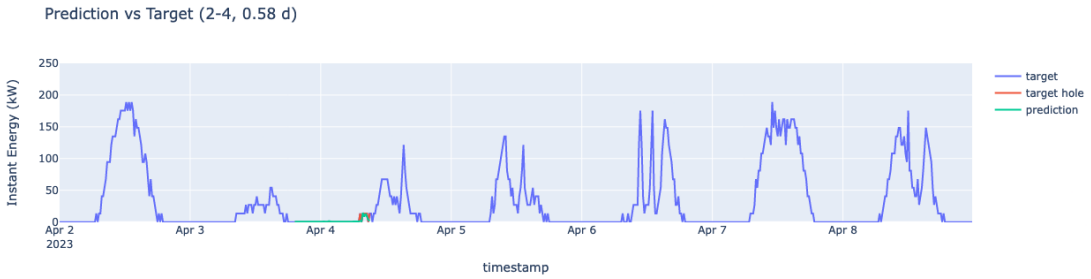
Table 4.8: Table (a) displays the global metrics calculated on the model's output with variable gap sizes, while (b) refers to the global metrics with a fixed two-day gap size. Both tables include the mean value and the standard deviation.



(a)

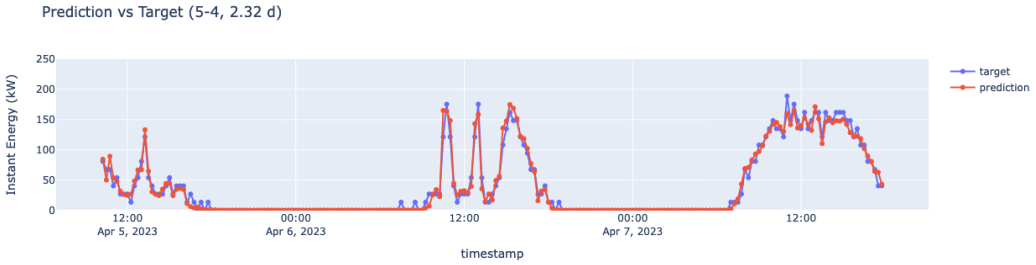


(b)

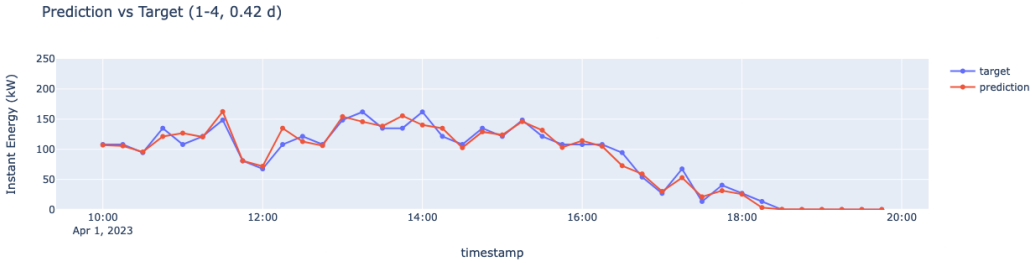


(c)

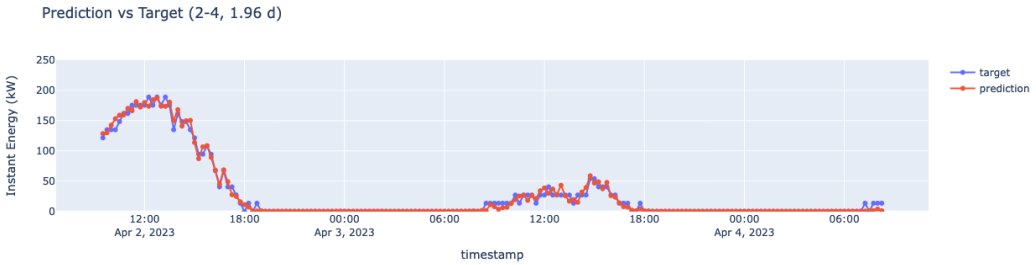
Figure 4.5: This figure shows some of the input time series provided to the model during the testing phase. We can see the entire series (in blue), the ground truth only for the gap period (in red), and the model's output only in the relevant gap period (in green). In plot (a), we have a gap size of almost two days, in (b) almost one day, and in (c) just over half a day.



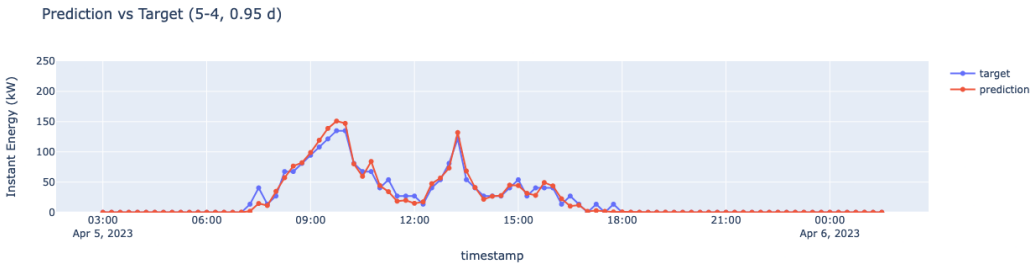
(a)



(b)



(c)



(d)

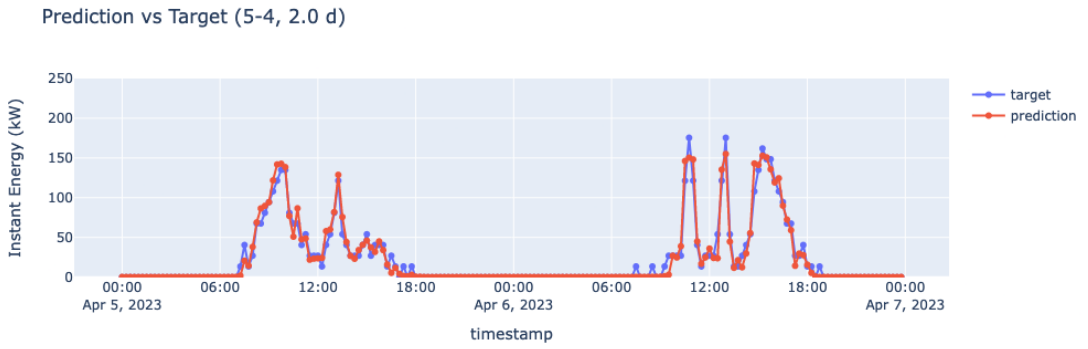
Figure 4.6: This figure displays some results of the model obtained during the testing phase, using gaps of varying lengths. You can see the model's outputs (in red) compared to their respective ground truths (in blue).

By repeating the evaluation phase with fixed gap sizes of two days to allow for a comparison with the other models, we can notice a slight improvement in the model's performance. Analyzing the data presented in Table 4.8(b), we have an average MAE of 3.76±0.39 kW (2% improvement with a 72% better standard deviation), an average MAPE of 18.14±6.76% (2% improvement with a 15% better standard deviation), an average MAPE@20 of 11.18±3.33 (1% improvement with a 23% better standard deviation) and an average R^2 value of 0.98±0.02 (1% improvement). In general, there is a 2% overall increase in performance with a significant reduction in the standard deviation values.

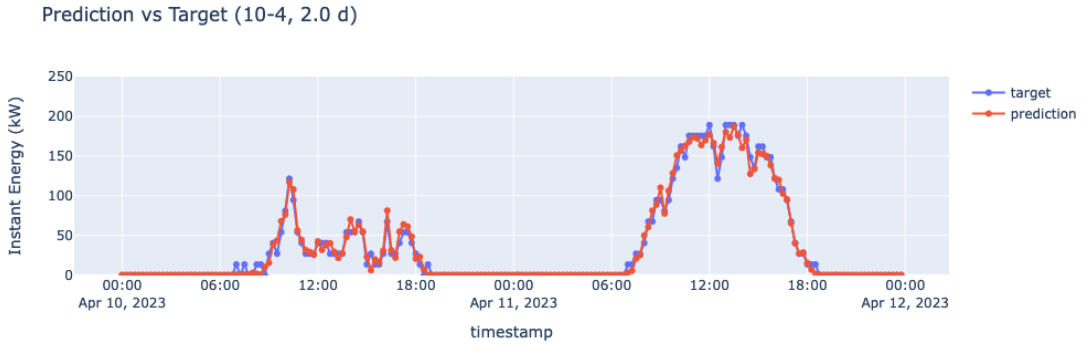
	Gap size	**MAE (kW)**	**MAPE (%)**	**MAPE@20 (%)**	**R^2**
Fig. 4.7(a)	96	4.25	26.99	19.673	0.96
Fig. 4.7(b)	96	3.88	20.91	11.94	0.99
Fig. 4.7(c)	96	3.52	9.25	7.09	0.99

Table 4.9: The table displays the values of MAE (Mean Absolute Error), MAPE (Mean Absolute Percentage Error), MAPE@20, and the R^2 (R-squared) index applied to some model predictions made during the testing phase with a fixed gap size of 2 days shown in Figures 4.7.

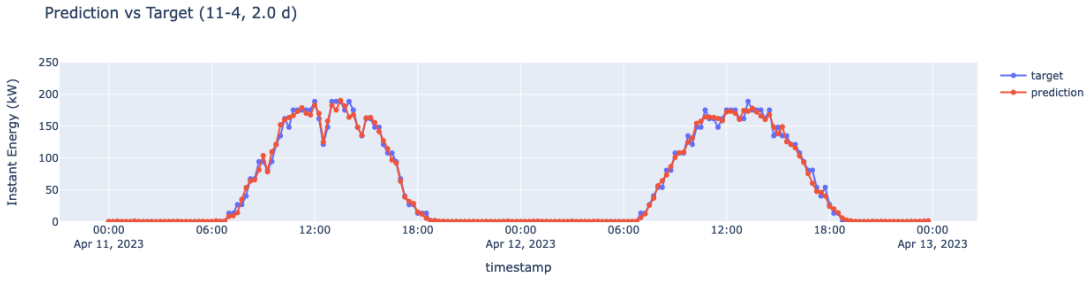
In conclusion, we can affirm that, despite the more demanding training phase, this model excels in the task of estimating instant energy production during periods of significantly variable gap sizes, achieving significantly better MAE, MAPE, MAPE@20, and R^2 index values compared to those of the previous architectures.



(a)



(b)



(c)

Figure 4.7: In this figure, some outputs from the testing phase of the model with a fixed 2-day gap size are presented. The model's predictions are shown in red, while the ground truth is in blue.

4.5 Model Comparisons

In this final section, we will compare the three models discussed in this thesis, analyzing their training phases, performance during testing phase, and identifying their strengths and weaknesses through the previous evaluation phase. We will determine which of these models is more suitable for solving the problem of time series imputation of photovoltaic data. It's essential to specify that to perform these comparisons, we reran the evaluation phase, fixing the gap size at 2 days and recalculating the various metrics used.

	Model Size (KB)	**Train Time (s)**	**GPU Usage (%)**	**CPU Usage (%)**
MLP	101.30	24.00	20	10
RNN	366.60	49.00	20	27
Transformer	2400.0	900.00	96	56

Table 4.10: The table presents some data obtained during the training phase of each model.

Analyzing the data shown in Table 4.10, we can see that the fastest and lightest model to train is the MLP-based one, followed by the RNN-based architecture, and finally, the Transformer. As seen in the previous sections, the first two models can certainly be trained on less powerful hardware compared to the one used for this thesis, while it might be challenging for the last model. However, it's important to note that the inference time for each model is less than a second.

For the models based on RNN and Transformer, the training phase was successfully completed without encountering significant issues. For the one based on MLP, we encountered some problems, especially with the values of the validation loss, as discussed in Section 3.1. Comparing the plots showing the training and validation loss curves, as shown in Figure 3.2, 3.5, and 3.9, we can see that the best one is the Transformer-based model with a final validation loss value of 0.02, which is 0.01 lower than that of the RNN-based model.

	MLP	RNN	Transformer	Incr. Gain (%)	
				RNN	Tran.
MAE (kW)	14.11 ± 3.81	6.86 ± 1.87	3.76 ± 0.39	51.38	45.18
MAPE (%)	70.98 ± 27.99	28.83 ± 10.02	18.14 ± 6.76	59.38	30.07
MAPE@20 (%)	53.01 ± 20.1	21.02 ± 8.67	11.18 ± 3.33	60.34	46.81
R^2	0.69 ± 0.17	0.92 ± 0.06	0.98 ± 0.02	25.00	6.12

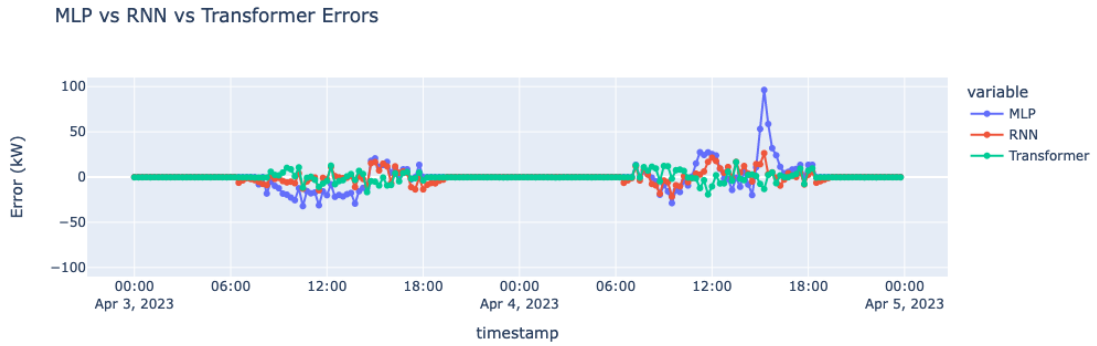
Table 4.11: In the table, you can find the average values of MAE, MAPE, MAPE@20, and the R^2 index for the three models, along with their respective standard deviations. Additionally, the Incremental Gain is provided, showing the percentage increase of the RNN model compared to MLP and the Transformer compared to RNN.

Examining Table 4.11, which shows the average values of MAE, MAPE, MAPE@20, and the R^2 index for the three models, we can observe that the one based on the Transformer is significantly better. The RNN-based model is undoubtedly better than the MLP-based one, with the MAE value being 51% lower, the MAPE showing an improvement of almost 60% and the MAPE@20 is 60% better. The R^2 index has also improved by 25%, indicating that the output of this model much better approximates the instantaneous energy values produced by the plant during 2-day gap periods.

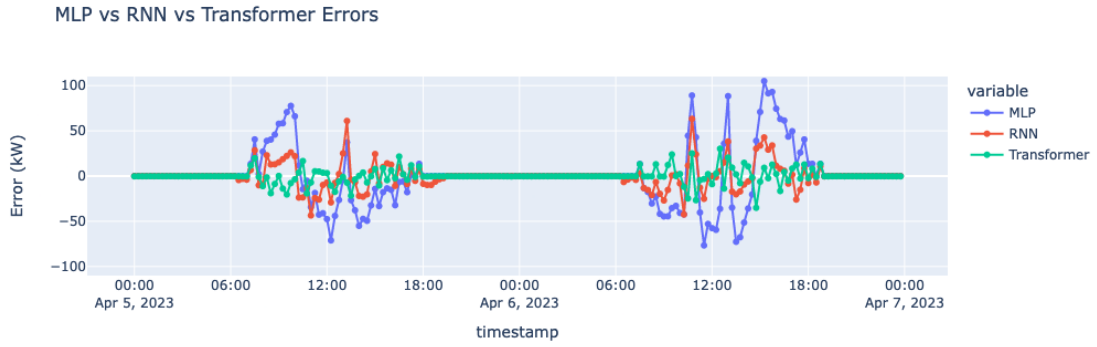
However, the Transformer-based architecture excels in this task, with a 45% improvement in MAE compared to the RNN-based model, a 30% improvement in MAPE, a 47% better MAPE@20 and a 6% improvement in the R^2 index. This model performs significantly better in predicting the plant's instantaneous energy compared to the RNN-based model, with the added capability of better detecting and understanding the presence of production peaks even over significantly variable time intervals.

	AVG MAE (kW)	AVG Min Err (kW)	AVG Max Err (kW)
MLP	13.91	0.35	96.81
RNN	7.14	0.20	65.56
Transformer	3.76	0.12	26.39

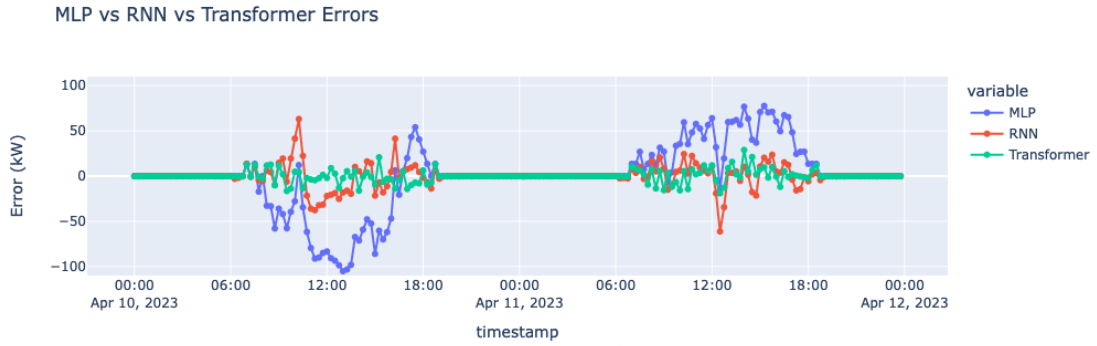
Table 4.12: The table displays the average values of the metrics presented in Table 4.13.



(a)



(b)



(c)

Figure 4.8: The figure compares the errors made by the three models in predicting gaps with a two-day gap size. The blue curve represents the errors of the MLP-based model, the red curve represents the errors of the RNN-based model, and the green curve represents the errors of the Transformer-based model.

These statements are supported by the plots presented in Figure 4.8 and the corresponding data in Tables 4.13 and 4.12. The plots show the differences in point errors made by the models. From these, we can see that the Transformer's errors (green line) are extremely compact and often close to the ideal value of 0, with an average value of 3.76 kW. The model based on RNN also has a similar curve (red line) but with values slightly farther from the ideal value and occasional error peaks, with an average value of 7.14 kW. The model based on MLP shows the worst error curve, often significantly deviating from the value of 0, featuring high error peaks, and having an average value of 13.91 kW.

We can also observe that the model based on the Transformer has a much lower average maximum point error, at 26.40 kW, compared to the RNN at 65.55 kW and the MLP at 98.81 kW. Similarly, for the average minimum point error, the Transformer-based architecture consistently performs the best, with an average value of 0.12 compared to 0.20 for RNN and 0.35 for MLP.

	MAE (kW)			**Incr. Gain (%)**	
	MLP	**RNN**	**Tran.**	**RNN**	**Tran.**
Fig. 4.8(a)	6.90	3.65	2.73	47.10	25.21
Fig. 4.8(b)	17.99	7.83	4.25	56.48	45.72
Fig. 4.8(c)	22.14	6.90	3.88	68.83	43.76

(a)

	Min. Err. (kW)			**Incr. Gain (%)**	
	MLP	**RNN**	**Tran.**	**RNN**	**Tran.**
Fig. 4.8(a)	0.10	0.05	0.04	50.00	20.00
Fig. 4.8(b)	0.07	0.03	0.02	57.14	33.33
Fig. 4.8(c)	0.19	0.13	0.10	31.57	23.07

(b)

	Max. Err. (kW)			**Incr. Gain (%)**	
	MLP	**RNN**	**Tran.**	**RNN**	**Tran.**
Fig. 4.8(a)	96.28	26.38	19.22	72.60	27.14
Fig. 4.8(b)	105.01	63.29	35.10	39.73	44.54
Fig. 4.8(c)	105.20	63.05	28.85	40.06	54.24

(c)

Table 4.13: In the table, the values of MAE (a), Minimum Error (b), and Maximum Error (c) are reported for each model in their respective testing periods. Additionally, the Incremental Gain in percentage is also shown for the RNN compared to the MLP and the Transformer compared to the RNN. These data correspond to the graphs in Figure 4.8.

Finally, after the analysis presented in this chapter, we can conclude that the best model for solving the imputation problem with data from time series of photovoltaic plants, despite the somewhat resource-intensive training phase, is undoubtedly the one based on the Transformer. This is due to its low MAE, MAPE and MAPE@20 values, and its near-perfect R^2 index. Moreover, the RNN-based model performs reasonably well in this task and could be a good alternative if one is willing to accept a slightly higher overall error in exchange for a significantly faster and lighter training phase compared to the Transformer.

Chapter 5

Final Conclusions

In conclusion, the models described and evaluated in this work have shown varying degrees of success in predicting the instant energy production curve of the plant during gaps. The Multi-Layer Perceptron Neural Network-based model, as expected, demonstrated limited ability to generalize and predict production variations effectively. It struggled to capture the possibility of production spikes and had difficulty managing the day/night cycle.

The Recurrent Neural Network-based model exhibited promise, particularly in capturing diverse energy production variations and handling more complex and variable patterns, it outshone the MLP-based model in various ways. This model can effectively handle gaps of various sizes, ranging from just a few timestamps to a large number of days. Furthermore, it demonstrates the ability to distinguish the day/night cycle and adjust production output accordingly. Its training phase is extremely fast and lightweight, making it very suitable for practical applications in addressing this problem, despite slightly higher errors.

Lastly, the absolute best model is the one based on the Transformer architecture. Its low error compared to the others and excellent ability to handle time series of highly variable lengths make it ideal for this task. It excels in approximating the energy produced by the plant during data gaps, comprehending the day/night alternation well and predicting potential energy spikes. Unfortunately, the training phase is quite resource-intensive, but once this hurdle is overcome, the inference time is extremely low.

Overall, the models provide valuable insights into energy production trends, and their predictability can be useful in real photovoltaic implants contexts. However, further model refinement and optimization, along with the introduction of techniques that can handle variable gap sizes and more complex patterns, may be necessary to achieve more accurate and reliable predictions. Additionally, expanding the dataset and applying advanced techniques for time series prediction could contribute to more successful models in the future.

Bibliography

[1] Kyunghyun Cho et al. *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*. 2014. arXiv: 1406.1078 [cs.CL].

[2] Lorenzo Ciampiconi et al. *A survey and taxonomy of loss functions in machine learning*. 2023. arXiv: 2301.05579 [cs.LG].

[3] datascience.eu. *Gated Recurrent Units Understanding the Fundamentals*. 2022. URL: https://datascience.eu/machine-learning/gated-recurrent-units-understanding-the-fundamentals/.

[4] Ketan Doshi. *Transformers Explained Visually (Part 1): Overview of Functionality*. 2020. URL: https://towardsdatascience.com/transformers-explained-visually-part-1-overview-of-functionality-95a6dd460452.

[5] Shiv Ram Dubey, Satish Kumar Singh, and Bidyut Baran Chaudhuri. *Activation Functions in Deep Learning: A Comprehensive Survey and Benchmark*. 2022. arXiv: 2109.14545 [cs.LG].

[6] Surya Krishnamurthy. Florian Wetschoreck Tobias Krabel. *RIP correlation. Introducing the Predictive Power Score*. 2020. URL: https://towardsdatascience.com/rip-correlation-introducing-the-predictive-power-score-3d90808b9598.

[7] Navid Mohammadi Foumani et al. *Improving Position Encoding of Transformers for Multivariate Time Series Classification*. 2023. arXiv: 2305.16642 [cs.LG].

[8] Dr. Saptarsi Goswami. *Introduction to Early Stopping: an effective tool to regularize neural nets*. 2020. URL: https://towardsdatascience.com/early-stopping-a-cool-strategy-to-regularize-neural-networks-bfdeca6d722e.

[9] Dan Hendrycks and Kevin Gimpel. *Gaussian Error Linear Units (GELUs)*. 2023. arXiv: 1606.08415 [cs.LG].

[10] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: 1502.03167 [cs.LG].

[11] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[12] Ilya Loshchilov and Frank Hutter. *Decoupled Weight Decay Regularization*. 2019. arXiv: 1711.05101 [cs.LG].

[13] Ilya Loshchilov and Frank Hutter. *SGDR: Stochastic Gradient Descent with Warm Restarts*. 2017. arXiv: 1608.03983 [cs.LG].

[14] Giuseppe Mastrandrea. *Correlation Matrix, Demystified*. 2022. URL: https://towardsdatascience.com/correlation-matrix-demystified-3ae3405c86c1.

[15] Leonie Monigatti. *A Visual Guide to Learning Rate Schedulers in PyTorch*. 2022. URL: https://towardsdatascience.com/a-visual-guide-to-learning-rate-schedulers-in-pytorch-24bbb262c863.

[16] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[17] Marius-Constantin Popescu et al. "Multilayer perceptron and neural networks". In: *WSEAS Transactions on Circuits and Systems* 8 (July 2009).

[18] Robin M. Schmidt. "Recurrent Neural Networks (RNNs): A gentle Introduction and Overview". In: *CoRR* abs/1912.05911 (2019). arXiv: 1912.05911. URL: http://arxiv.org/abs/1912.05911.

[19] solargis.com. *Solar irradiance data | Solargis*. URL: https://solargis.com/.

[20] Juan Terven et al. *Loss Functions and Metrics in Deep Learning*. 2023. arXiv: 2307.02694 [cs.LG].

[21] College of AgriCulture The University of Arizona and life sciences. *Solar Photovoltaic (PV) System Components*. 2018. URL: https://extension.arizona.edu/sites/extension.arizona.edu/files/pubs/az1742-2018.pdf.

[22] turing.com. *Understanding Transformer Neural Network Model in Deep Learning and NLP*. URL: https://www.turing.com/kb/brief-introduction-to-transformers-and-their-power.

[23] Ashish Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL].

[24] Patrick Zippenfenig. *Open-Meteo.com Weather API*. DOI: 10.5281/zenodo.7970649. URL: https://github.com/open-meteo/open-meteo.