UNIVERSITY OF PISA

Department of Information Engineering

Master's degree in Artificial Intelligence and Data Engineering

Internet of Things

EV SMART CHARGING

Work Group:

Filippo Gambelli

Andrea Bochicchio

ACADEMIC YEAR 2024/2025

Contents

1 Introduction

The integration of the Internet of Things (IoT), renewable energy, and electric mobility is a key aspect of the ongoing global energy transition. The rapid increase in electric vehicles, combined with the growing demand for sustainable energy sources, underscores the need for innovative systems capable of managing energy resources efficiently. Conventional charging infrastructures often fail to account for the variability of renewable energy sources and the fluctuating demands of electric vehicle users. Therefore, intelligent solutions that can balance energy supply and demand in real time, minimize dependency on the conventional power grid, and promote environmental sustainability are increasingly necessary.

This project specifically addresses the domain of **Electric Vehicles and Charging Infrastructure**, one of the six use-case domains for IoT applications focused on energy efficiency. The main objective is to design and implement a smart charging system that optimizes the allocation of available energy to connected electric vehicles based on both current solar power generation and short-term forecasts.

The system combines IoT sensors, predictive algorithms, and dynamic power management. Solar panels monitor environmental parameters such as sunlight intensity, ambient temperature, and sun position. These measurements are processed using Machine Learning models to forecast solar energy availability over the next few hours. The predicted energy is then used by the power manager to dynamically allocate charging power to vehicles, ensuring maximum use of renewable energy while limiting reliance on the traditional electricity grid.

Beyond technical performance, the proposed system demonstrates practical applicability in a variety of environments, including corporate parking lots, public charging stations, residential neighborhoods, and urban mobility hubs. It not only improves energy efficiency and reduces operational costs but also contributes to sustainability goals by lowering carbon emissions and promoting green technologies. In doing so, the project exemplifies how IoT can transform conventional charging points into intelligent, adaptive nodes within a smart energy network.

The complete source code is available in the GitHub repository: EV Smart Charging

2 Dataset and Machine Learning Model

To develop a prediction system for the energy production of photovoltaic panels, a dataset from **Photovoltaic Geographical Information System (PVGIS)** was used (link to dataset). PVGIS is an official service of the European Commission that provides accurate data on the energy production of photovoltaic systems based on geographic location and technical system characteristics.

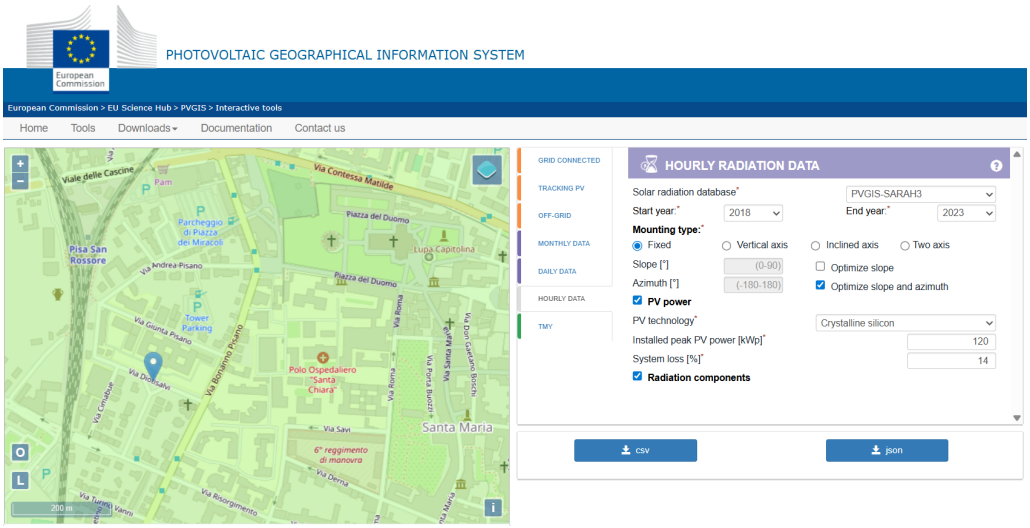


Figure 2.1: Photovoltaic Geographical Information System (PVGIS)

2.1 Photovoltaic panels

For this project, the chosen location for the photovoltaic system is **Via Diotisalvi, Pisa**, where the School of Engineering of the University of Pisa is located. This area was selected as a realistic scenario for testing a smart charging system in an urban environment.
The photovoltaic system considered in this project is based on a commonly used panel type, with the following specifications:

- **Panel type**: monocrystalline
- **Average efficiency**: 20%
- **Nominal power per panel**: 500 W
- **Space required per panel (including spacing)**: 3 m^2
- **Total available area**: 900 m^2
- **Total number of panels**: 900 m^2/3 m^2 = 300 panels
- **Total nominal system power**: approximately 120 kWp
- **Estimated system losses**: 14%

These parameters represent a medium-to-large sized photovoltaic system, suitable for applications in corporate or university settings, and serve as the foundation for the simulation of the smart charging system.

By inputting the above information into the PVGIS platform, several variables describing the behavior of the photovoltaic system and local environmental conditions were obtained, covering the period from 2018 to 2023.
The dataset provides hourly values of:

- **P**: instantaneous power output of the PV system (W)
- **Gb**: direct irradiance on the inclined plane of the modules (W/m^2)
- **Gd**: diffuse irradiance on the inclined plane of the modules (W/m^2)
- **Gr**: reflected irradiance on the inclined plane of the modules (W/m^2)
- **H_sun**: sun height above the horizon (°)
- **T2m**: air temperature at 2 meters above ground (°C)
- **WS10m**: wind speed at 10 meters above ground (m/s)

Since PVGIS provides hourly data, a temporal interpolation was applied to obtain values every 15 minutes. This step was necessary to simulate the real-time operation of the system, allowing the power manager to make decisions with finer granularity based on rapidly changing weather conditions.

Thanks to this dataset, we obtained a complete and reliable source of information, accurately representing both the energy production of the photovoltaic system and the environmental factors that influence it.

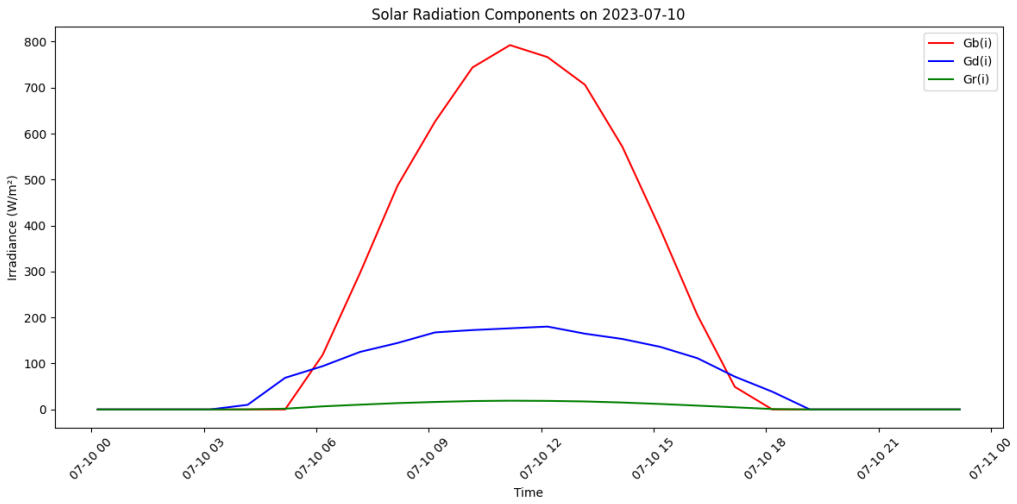


Figure 2.2: Example Solar Radiation Components

2.2 Machine Learning Model

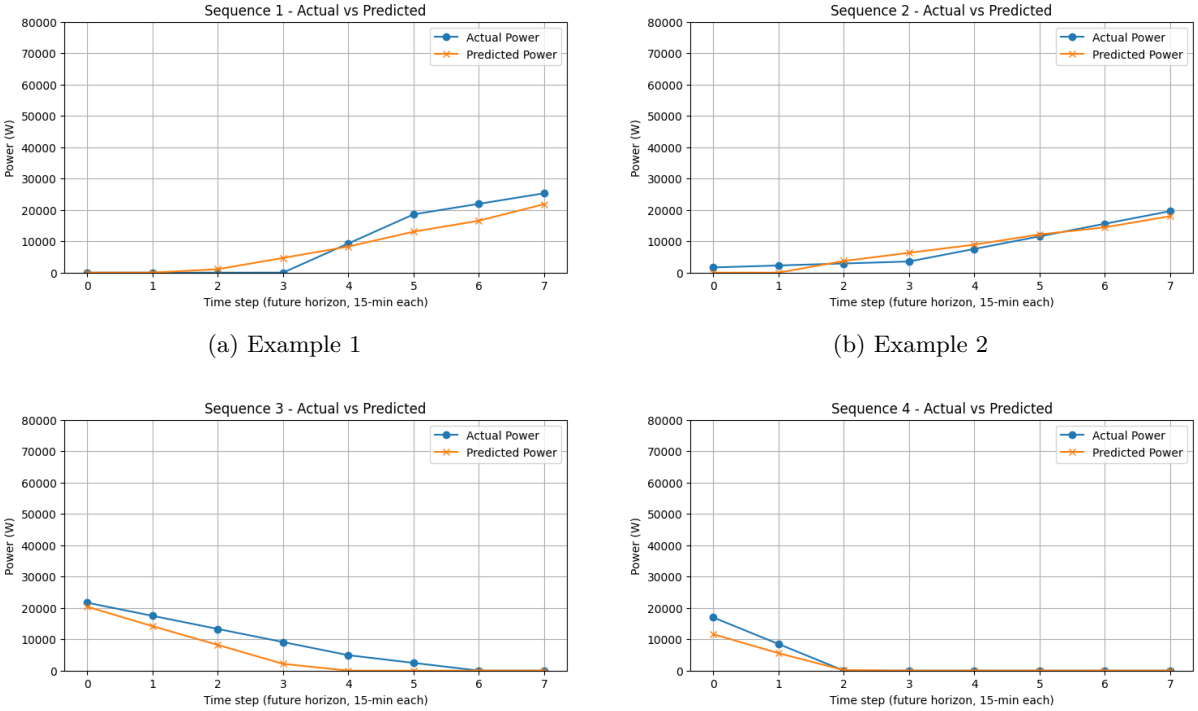
To predict the energy production of the photovoltaic panels, a **feed-forward neural network** was developed. The model estimates the power generated in the next hours based on historical data and environmental conditions.

The dataset was first preprocessed and split into a training set (2018–2022) and a test set (2023). To capture temporal relationships, sequences of **8 past steps (look-back)** were used, representing 2 hours of historical data at **15-minute intervals**, and the model predicts the **next 8 steps (2-hour forecast horizon)**. These sequences were flattened to fit the input shape required by a feed-forward neural network.

The output layer predicts the power for each step in the forecast horizon. The model was trained using the Adam optimizer and Mean Squared Error (MSE) as the loss function. Early stopping was applied to stop training if the validation loss did not improve for 5 consecutive epochs, preventing overfitting.

After training, the model was evaluated on the test set. Step-by-step errors for each forecast horizon and overall metrics were calculated, showing that the model can accurately follow the dynamics of solar energy production even with varying weather and time conditions. Furthermore, some examples of comparisons between predicted and actual values are reported, as illustrated in Figure 2.3.

Finally, to allow the model to run directly on IoT devices, it was converted to C code using **emlearn**. This enables integration with the energy management system of the solar panels in real time, allowing the power manager to allocate energy to the charging stations intelligently, based on the model's predictions.



(a) Example 1

(b) Example 2

(c) Example 3

(d) Example 4

Figure 2.3: Comparisons between predicted and actual values across different test cases.

3 System Architecture

This chapter describes the architecture of the proposed system for managing photovoltaic energy production and its integration with electric vehicle charging stations and the power grid. The architecture is composed of multiple interconnected nodes, each with a well-defined role: the **Sensor PV** node collects environmental data and forecasts energy production, the **Charging Station** nodes manage vehicle charging sessions and monitor renewable energy usage, the **Smart Grid** node regulates power exchanges with the grid, while the **Central Node** coordinates the overall system through the power manager algorithm. By combining real-time measurements, predictive models, and priority-based power distribution, the system ensures an efficient and adaptive use of solar energy, minimizing grid dependency while guaranteeing reliable charging services.

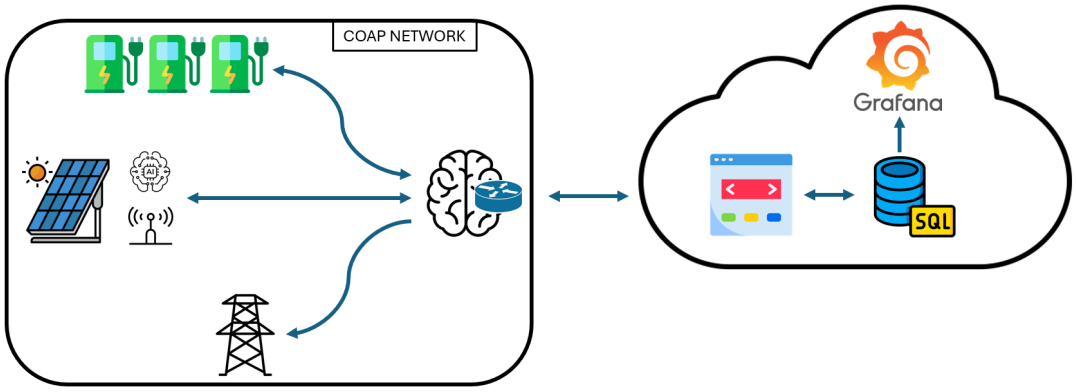


Figure 3.1: Enter Caption

3.1 Sensor PV

A node within our architecture is the PV Sensor, which is responsible for managing all sensors associated with solar panels and for predicting the energy production through the forecasting model.
The resources available in this node are:

- **Solar Data** (`res-solar-obs.c`): collects and manages all data from the solar panel sensors, which are measured every 20 seconds, since these parameters do not change significantly in a short time. The collected data are then processed and used as input for the forecasting model to predict power generation. The sensors include:
  - **Pyranometers**: for measuring solar irradiance (direct, diffuse, and reflected components)
  - **Thermometer**: measuring air temperature at 2 meters above ground level
  - **Anemometer**: measuring wind speed at 10 meters above ground level
  - Sun elevation angle measurement

  These data are essential for monitoring and predicting solar energy production.
- **Real Power Data** (`res-real-power-obs.c`): provides real-time data on the energy currently generated by the solar panels and is monitored every 5 seconds, which is crucial for effective real-time management.
- **Predicted Power Data (`res-pred-power.c`):** this resource uses a machine learning model to forecast energy production for the next 2 hours, providing predictions at 15-minute intervals. The

model outputs a sequence of predicted power values P_i for each interval $i = 0, \ldots, HORIZON - 1$ where, in this case, $HORIZON$ is equal to 8. The production trend is then calculated as a weighted sum of differences between consecutive predicted values:

$$\text{trend\_value} = \sum_{i=0}^{HORIZON-2} w_i \left(P_{i+1} - P_i\right)$$

where the weights w_i give more importance to near-term intervals and decrease for later intervals:

$$\mathbf{w} = [0.35, 0.25, 0.15, 0.10, 0.08, 0.05, 0.02]$$

The trend is finally determined as:

$$\text{trend} = \begin{cases} 1 & \text{if trend\_value} \geq 0 \\ -1 & \text{if trend\_value} < 0 \end{cases}$$

When a client performs a GET request on this resource, the server returns both the **trend** (1 for increasing, -1 for decreasing) and the **first predicted power value** for the upcoming 15-minute interval.

Both the **Solar Data** and the **Real Power Data** resources are implemented as *observable* resources, since they are updated at frequent and regular intervals directly from the measurements. This allows clients to automatically receive notifications whenever new data is available, without the need for repeated requests. On the other hand, the **Predicted Power Data** resource is not observable, as the prediction process is computationally more demanding and it is up to the central node to decide when to request a new forecast, depending on the specific needs of the system.

3.2 Charging Station

In our system, the number of charging stations nodes matches the number of stations in the deployment site, with each station represented and managed by its own node.
The operations handled by the charging stations are as follows:

- **Station Registration**: when a charging station is set up, it communicates its availability and maximum power output to the system. This information is essential for optimizing energy distribution and planning charging sessions efficiently.
- **Vehicle Connection** for charging: when a vehicle connects to a charging station, it sends the following key information: *Maximum battery capacity*, *Maximum supported charging power*, *Current battery percentage*, *Desired charge percentage*, *License plate number*, which allows the system to identify the vehicle and apply priority rules if necessary. Priority management enables the system to give certain vehicles higher charging priority, for example, in cases of emergency, premium subscriptions etc...
- **Vehicle Disconnection**: when charging is complete or the user decides to stop, the vehicle disconnects and the system updates the station status, making it available again.

The resource available in this node is:

- **Charging Status** (`res-charging-status.c`): The purpose of this resource is to track how much energy the charging station needs to deliver to the vehicle to complete the charging within the expected time, while also optimizing the use of energy produced by the solar panels based on

predicted future production. In addition, the resource monitors how much of the energy being delivered is renewable. Finally, it also manages the end of the charging process, controlling the station's LEDs to indicate whether charging is still in progress or if the vehicle has reached the target charge percentage.

3.3 Smart Grid

This node is responsible for the management of the electrical grid. Its main task is to either draw power from or supply energy to the grid based on demand. For modeling purposes, it is assumed that there is no limit to the requested power, so the required kW is always available.
The node resource is:

- **Status Power Grid** (`res-status-smart-grid.c`): which receives updates on power needs and indicates how much power should be supplied or drawn, taking into account the energy produced by the solar panels.

3.4 Central Node (border router)

The central node manages the charging stations, solar panels, and the power grid. It runs all the logic for power management. The main tasks are monitoring the resources that provide the actual real-time power produced by the solar panels, requesting forecasts of the power that will be generated in the future, and analyzing the related trend. It also has dedicated resources for managing charging stations and for setting the execution interval of the power production forecasts from the SensorPV node. In particular:

- **Register Charger:** (`res-charger-reg.c`): handles the registration of the charging station and the management of all related data about the station and the connected vehicle, providing the application layer with the necessary information for monitoring
- **Register Car:** (`res-car-reg.c`): handles the registration of vehicle data when it connects to the charging station. It saves all the information sent by the vehicle (through the station) and estimates the time needed to complete charging, considering different parameters such as priority, current charge level, target charge level, and battery capacity
- **ML Prediction Interval:** (`res-ml-pred-interval.c`): allows the application layer to configure how often the production forecasts are executed, or to disable them completely

3.4.1 Power Manager Algorithm

The algorithm implemented in the power manager is designed to dynamically distribute power among electric vehicles connected to charging stations, making the best use of the available solar energy and relying on the power grid only when necessary.
Its operation can be described in the following main steps:

1. **Calculation of the energy delivered since the last execution** - In each cycle, the system calculates the time elapsed since the previous run and updates the state of charge (SoC) of each vehicle based on the power assigned to it. If a vehicle has already reached its target charge level, charging is stopped.
2. **Sorting of charging stations** - The charging stations are ordered according to two criteria:
    - Vehicle priority: some vehicles may have higher priority (for example, in a hospital, emergency vehicles may have higher priority than staff vehicles)

- The remaining time to reach the target charge level (vehicles with less time available are managed first)

3. **Assignment of charging power** - For each vehicle still charging, the system calculates the power required to reach the target charge level in the remaining time.
   - *Case 1 – Sufficient solar power:* the required power is fully covered by the solar panels.
   - *Case 2 – Insufficient solar power:* if the forecast indicates an *increasing trend* in solar production, the system first uses as much solar energy as possible and, only if necessary, adds power from the grid. If the forecast indicates *decreasing trend*, there is no reason to wait, so both solar and grid power are used.
4. **Updating information** - For each vehicle, updated information is periodically sent to the charging station. At the same time, the system communicates the overall status to the power grid: excess power exported to the grid if solar panels produce more than needed; power requested from the grid if solar panels cannot cover the demand.

3.5 CoAP Protocol

The Constrained Application Protocol (CoAP) has been chosen as the communication protocol for the proposed IoT-based smart charging system due to its lightweight nature, low overhead, and suitability for resource-constrained devices. CoAP is designed specifically for applications where devices have limited computational resources, memory, and energy, making it ideal for the embedded nodes. Several factors influenced the decision to use CoAP over alternative protocols such as MQTT:

- **Direct Client-Server Interaction**: CoAP follows a RESTful architecture similar to HTTP, allowing each node to expose its data as resources that can be queried or updated directly by other nodes. This is particularly useful in our system, where sensor PV nodes, charging stations, and the central node need to exchange real-time data without relying on an intermediary.
- **Elimination of a Central Broker**: MQTT is a publish-subscribe protocol that requires a centralized broker to relay messages between clients. In our architecture, the presence of a broker would introduce an unnecessary dependency and a potential single point of failure. CoAP, on the other hand, allows direct peer-to-peer communication, which perfectly matches the needs of our system.
- **Lightweight and Efficient Communication:** CoAP messages are significantly smaller than MQTT messages and are transmitted over UDP, reducing latency and network overhead. This is particularly important in scenarios where frequent updates are sent, such as real-time power measurements from the photovoltaic system, environmental sensor data, and charging status notifications.
- **Observability of Resources:** CoAP supports an observe mechanism, allowing clients to subscribe to resources and receive automatic updates when their state changes. This is essential in our architecture, where the central node and charging stations need continuous data on solar irradiance, real power, and predicted trends. MQTT, based on a brokered message system, achieves similar behavior through topic subscriptions. CoAP can replicate this using observable resources, enabling automatic notifications while avoiding the extra logic and potential network load that MQTT might require

Given these considerations, CoAP is a natural and efficient choice for the EV Smart Charging architecture. It enables decentralized, real-time, event-driven communication, reduces network overhead.

3.6 Data Encoding

The system uses different data encoding formats depending on the type of communication and the resources available on the devices.

- **IoT Node-to-Node Communication**: for messages exchanged between IoT nodes within the network, we use a structured `text/plain` format. An example message is:

  trend=-1&firstPrediction=78.453

  This format is chosen because it is lightweight and requires minimal processing, which reduces energy consumption on resource-constrained IoT devices. Unlike JSON, extracting values from this simple key-value format is faster and less demanding for small embedded nodes.

- **Node-to-Cloud Communication**: when sending data from the IoT nodes to the cloud application, we use `JSON`. The cloud has sufficient computational resources to parse JSON quickly, so the overhead is not an issue. JSON also provides a standardized, human-readable format that is convenient for integrating with cloud services, logging, and further processing.

- **Cloud-to-Node Communication**: messages sent from the cloud application back to the IoT nodes again use the lightweight `text/plain` format with the `&` separator. This ensures minimal processing and energy use on the embedded devices while maintaining consistency with the node-to-node message format.

In summary, the encoding strategy balances efficiency and readability: lightweight text-based messages for energy-constrained devices, and JSON for cloud communications where resources are abundant.

4 Cloud Application

The cloud application plays a central role in the EV Smart Charging system, serving both as a data repository and an active management interface for the IoT network. Implemented entirely in Java, the application provides a seamless connection between the IoT devices deployed on the charging infrastructure and the system's decision-making logic.
Its main functions include:

- **Data Storage**: the cloud application collects and stores all sensor readings from the photovoltaic system, ensuring a reliable and centralized source of historical data for monitoring, analysis, and future forecasting.
- **Vehicle Priority Management**: a dedicated resource allows the central node to query the priority associated with any connected vehicle using its license plate. This enables the system to make real-time charging decisions based on predefined vehicle priorities.
- **Remote Device Control**: the application provides the ability for users to modify certain operational parameters of the IoT network, such as adjusting the execution interval of the machine learning model. At the same time, the cloud application automatically manages the model's operation, for example stopping it during the night and restarting it in the morning, ensuring efficient and autonomous system operation.
- **Automatic Node Registration**: the cloud application also acts as a CoAP server that manages the registration of IoT devices. Each node can automatically register, discover the required peers, and obtain their IPv6 addresses, enabling dynamic device discovery and seamless communication without manual configuration.

Overall, the cloud application not only serves as a backend database but also actively supports the energy management system, enhancing both efficiency and flexibility of the smart charging network.

4.1 Database

The cloud application is supported by a MySQL database, which stores all relevant system data in a structured and organized manner. The database includes tables designed to manage sensor data, real-time power measurements, vehicle priorities and registered nodes.
The tables are:

- **platePriority**: stores vehicle license plates and their associated charging priority. The priority is used by the system to allocate power according to predefined rules, ensuring that critical or high-priority vehicles are charged first.
- **realPower**: records the real-time energy output of the photovoltaic system. Each entry is timestamped to allow historical tracking and analysis of the energy production trends.
- **solarData**: contains environmental parameters collected from the solar panel sensors, including direct, diffuse, and reflected irradiance, sun elevation, air temperature, and wind speed.
- **device**: stores information about devices that have connected to the system. Each record includes a identifier (`id`), the node type (`nodeType`), and the endpoint (`EP`). The primary key is a composite of `id` and `nodeType`, ensuring uniqueness of each device entry.

4.2 CoAP Server for Automatic Node Registration

In addition to its data storage and management functionalities, the cloud application also acts as a **CoAP server** for automatic registration of IoT devices. This server allows each node in the smart charging network to register itself and obtain the IPv6 addresses of the other required nodes, enabling full automatic discovery within the system. The registration process works as follows:

- Each device sends a registration request to the server, specifying its type (`nodeType`) and a list of required nodes (`requiredNodes`).
- The server stores the device information in the database, including the device identifier, node type, and endpoint.
- If all required nodes are already registered, the server responds immediately with the corresponding IPv6 addresses. Otherwise, the registration request is temporarily held until the missing nodes register.
- Once all required nodes are available, the server automatically completes the registration by providing the IPv6 addresses of the requested nodes.

This mechanism ensures that devices can dynamically discover and communicate with each other without manual configuration.

4.3 Sensor Data Acquisition and Storage

The cloud application continuously collects data from the IoT network by observing specific resources exposed by the SensorPV node. Two key resources are monitored: **Solar Data** (res-solar-obs.c), **Real Power Data** (res-real-power-obs.c). To achieve this, the cloud application implements a CoAP observer that subscribes to these resources. Whenever a sensor reading is updated, the cloud application receives a notification containing the latest measurements. Each received update is parsed and stored in the MySQL database.

4.3.1 Automatic Machine Learning Model Control

In addition to passive data collection, the cloud application actively participates in the management of the machine learning (ML) model. The application monitors the real-time power output and uses this information to make decisions regarding the activation or deactivation of the ML model:

- If the ML model is inactive and the system detects positive power generation from the solar panels, the application automatically trigger the model to start.
- If the ML model is running and the system detects a prolonged period of zero power generation (e.g., during the night), the application can stop the model automatically to save computational resources. Specifically, if the real power output remains zero for a continuous period of 30 minutes, the cloud application stops the ML model to avoid unnecessary computation.

This dynamic control mechanism ensures that the ML model is only active when it can provide meaningful forecasts, optimizing both system performance and computational efficiency.

4.4 Vehicle Priority Resource

The cloud application exposes a dedicated CoAP resource named `plate`, which allows the central node to query the charging priority associated with a specific vehicle. This resource is an essential part of the system's energy management strategy, as it enables the smart charging system to make real-time decisions regarding the estimated charging time for each vehicle based on their priorities.
When a client sends a request to the `plate` resource, the request must include the vehicle's license plate as a query parameter. The cloud application first validates the provided license plate to ensure it is not missing or empty. It then queries the MySQL database to retrieve the vehicle's predefined priority. If the vehicle is registered, the corresponding priority is returned; otherwise, a default standard priority is sent back.
This CoAP resource allows the central node to implement intelligent power allocation by dynamically retrieving up-to-date vehicle priority information from the central database. It also provides a standardized interface for other applications to access this data. By exposing the plate resource, the cloud application integrates vehicle-specific priority management into the real-time energy distribution logic of the EV Smart Charging system, enhancing both efficiency and fairness in charging operations.

4.5 Command-Line User Interface

The cloud application includes a command-line user interface (CLI) that provides a simple yet powerful means for operators to interact with the EV Smart Charging system.
The main functionalities available through the CLI include:

- **Database Management**: users can add new vehicle license plates along with their associated charging priority to the database.
- **ML Prediction Interval Configuration**: operators can modify the time interval at which the ML model produces predictions.
- **Monitoring Smart Grid Status**: the interface provides real-time and cumulative metrics for the smart grid, including current power consumption from the grid, energy exported back to the grid, total costs and earnings, and the overall energy balance.
- **Monitoring Charging Stations**: users can view the status of all charging stations, including whether a vehicle is connected, whether charging is in progress, the amount of power allocated from solar panels and the grid, vehicle battery capacity, current state of charge, target charge level, estimated remaining time and energy, license plate, and charging priority.
- **Viewing Stored Data**: the CLI provides access to historical data stored in the database, including real power measurements, solar irradiation data, and the list of registered license plates with their priority levels. Users can specify how many recent records to display, allowing for flexible data inspection.
- **System Status Checks**: the interface allows users to quickly check if the ML model is currently running, giving immediate feedback about the operational state of the predictive component.

The CLI is organized as a main menu, where each option corresponds to a specific system operation. Users interact by entering numeric choices, making the interface straightforward and intuitive while still providing full access to the system's monitoring and control capabilities.

4.6 Grafana Dashboard

Grafana is an open-source platform for monitoring and data visualization. In our project it is directly connected to the MySQL database, enabling real-time visualization of both environmental and charging data through graphs and tables. The dashboard we designed (Fig 4.1 and Fig 4.2) includes:

- **Irradiance Graph**: comparison of direct (Gb), diffuse (Gd), and reflected (Gr) irradiance.
- **Sun Height Graph**: evolution of the solar elevation angle (H_sun) over time.
- **Temperature Graph**: monitoring of ambient temperature ($T2m$) over time.
- **Real Power Graph**: actual power output of the photovoltaic system.
- **Connected Devices Table**: list of devices that have connected to the system along with their IPv6 addresses.
- **Vehicle Priority Table**: list of license plates with the corresponding charging priority.

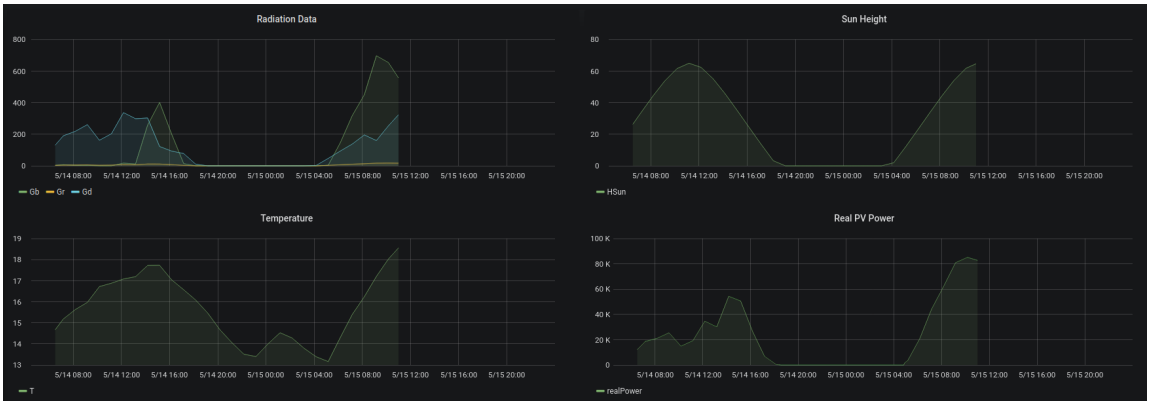


Figure 4.1: Sensor PV data in Grafana

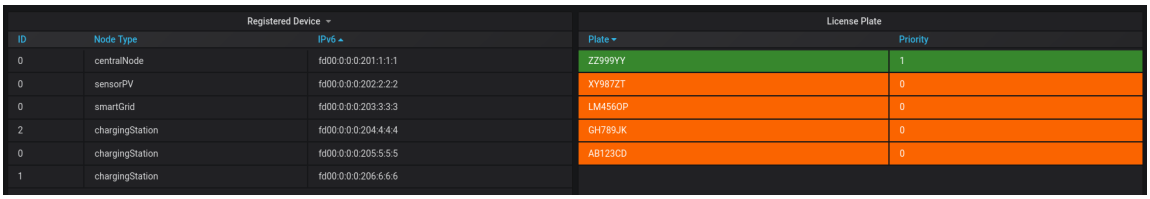


Figure 4.2: Connected Devices and Vehicle Priority data in Grafana