Integration of IoT with AI for Solar Monitoring & Optimization

By:

MOHAMED ABDELHAMID TAMEEM: 206810

SHERIF ASHRAF HELMY: 212982

YOUSSEF ASHRAF HASSAN: 208038

MOAMEN GAMAL MAHMOUD: 218344

YOUSSEF KHALED KAMEL: 218180

A Dissertation submitted to the Faculty of Engineering, The British University in Egypt, in Partial Fulfillment of the requirements for the bachelor's degree in electrical engineering

Supervisor(s):

DR. ZAHRAA ISMAIL

ENG. HESHAM HANY

June, 2025

Electrical & Communication Engineering Department

British University in Egypt Faculty of Engineering

“INTEGRATION OF IOT WITH AI FOR SOLAR MONITORING & OPTIMIZATION”

Dissertation Approved:

Committee member

Committee member

External committee member

Affiliation:

Dean Faculty of Engineering

Acknowledgments

First and foremost, we would like to sincerely thank our supervisors, both Dr. Zahraa and Engineer Hesham, for their support and guidance throughout the project.

Abstract

The integration of Artificial Intelligence (AI) with Internet of Things (IoT) presents a transformative approach to enhancing the efficiency and performance of solar energy systems. This project includes the designing of the real-time solar monitoring embedded system based on a microcontroller, with the purpose of acquiring and processing the environmental and electrical data measured from a polycrystalline silicon solar cell. The system consists of a combination of several sensors that can measure temperature and humidity, light levels, voltage and current, and the data is logged locally on an SD card and in the cloud on the ThingSpeak IoT platform. This system will be combined with AI machine learning algorithms such as Random Forest and SVM to help monitor and predict the current weather condition surrounding the solar panel. This system prototype will utilize the following components: ESP32 microcontroller, temperature and humidity (DHT22), light intensity (LDR), and current (INA219), to collect real-time environmental and electrical data such as open-circuit voltage and short-circuit current. The system collected a sizeable amount of data over a course of eight days, with an average of more than 57,000 readings per day, including changes in the behavior of the solar panels due to daily light cycles. The trained machine learning models the weather condition as the labels and the collected data as the input. The Random Forest model provided ideal classification measurements, whereas the SVM model had slightly reduced recall on cloudy conditions. Overall, this research highlights the benefits of real-time monitoring and predictive analytics in improving solar energy efficiency, offering a scalable and sustainable solution for future energy needs.

Table of Contents

List of figures

List of Tables

1. Chapter (1): Introduction

1.1. General Background: Global Energy Landscape

It is no secret that energy, in all its forms, plays such a major role in every aspect of today's modern world. Energy is key driving force behind humans' daily activities, from lighting homes, heating food to powering industries and global transportation, there is no doubt of the significance of energy in day-to-day life. Transportation systems, for instance, are one of the most heavily reliant applications utilizing energy worldwide. Due to the increased urbanization in the past decade, a higher demand for transportation systems has become needed, and the demand for it is expected to keep increasing till 2050 [1].

In fact, according to UN (United Nations), urbanized areas account for 70% of energy usage, even though they just take only 3% of Earth's surface [2]. Moreover, there have been reports utilizing energy use in the agricultural sector. Energy usage and cost has drastically increased as it has been employed in agricultural cropping systems [3]. Not only that, but energy is fundamental for providing critical services such as education and healthcare, making it indispensable to human progress. Overall, energy has played such a crucial role in socio-economic development, fueling both progress and growth across centuries [7].

There are two types of energy sources: renewable and non-renewable. One of the major challenges in today's era is the full reliance on the non-renewable energy sources. Non-renewables such as coal, oil, and natural gases are at least 80% of global annual energy consumption while renewables are in the minority [4] as can be seen in figure 1-1.

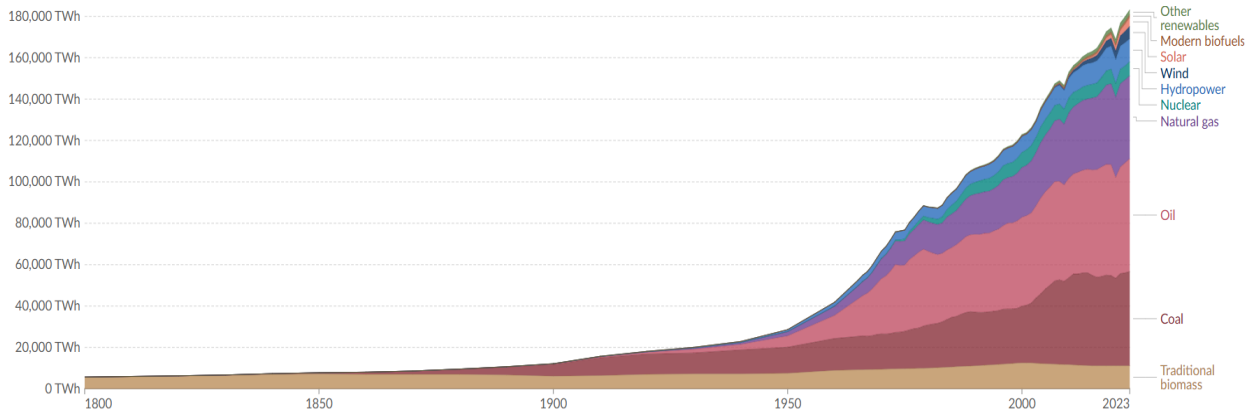

Figure 1-1. Energy Consumption Globally throughout 1800 to 2023 [4]

However, as populations arise, the demand of energy also arises. Population is expected to reach 8.5, 9.7, and 11.2 billion residents by 2030, 2050 and 2100 respectively [9]. Non-renewables such as fossil fuels and oil are limited in supply, meaning that one day they will eventually run out. Sooner or later, this will pose a threat to the world's energy demand. In addition, scientists and researchers have become aware of the potential environmental concerns surrounding the use of non-renewables [6]. Non-renewable sources cause the release of harmful emissions, which in turn will cause air pollution, habitat destruction, and climate change.

Accordingly, if industries continue to fully depend on these energy sources, this will create risks for both the planet's environment and humanity as a whole. Carbon dioxide (CO_2) which is known to be one of the major Green House Gases (GHG), gets released from oil thermal powerplants which is a non-renewable source. Not only is it released alone but it also releases sulfur oxides, nitrogen oxides and particulate matter, which all contribute to harming the atmosphere [7].

It is estimated that these oil thermal powerplants emit about 700 to 800 grams of CO_2/kWh which is significant considering it is only coming out from oil sources only, not to mention the other non-renewables [7]. Nevertheless, GHG emissions have rapidly increased across the globe with developing countries surpassing developed countries in 2008 [9]. Figure 2 shows each region's CO_2 emission. It is clearer than ever that alternatives for these non-renewable sources have become needed. China and United states are the countries responsible for most CO_2 emissions globally [10].

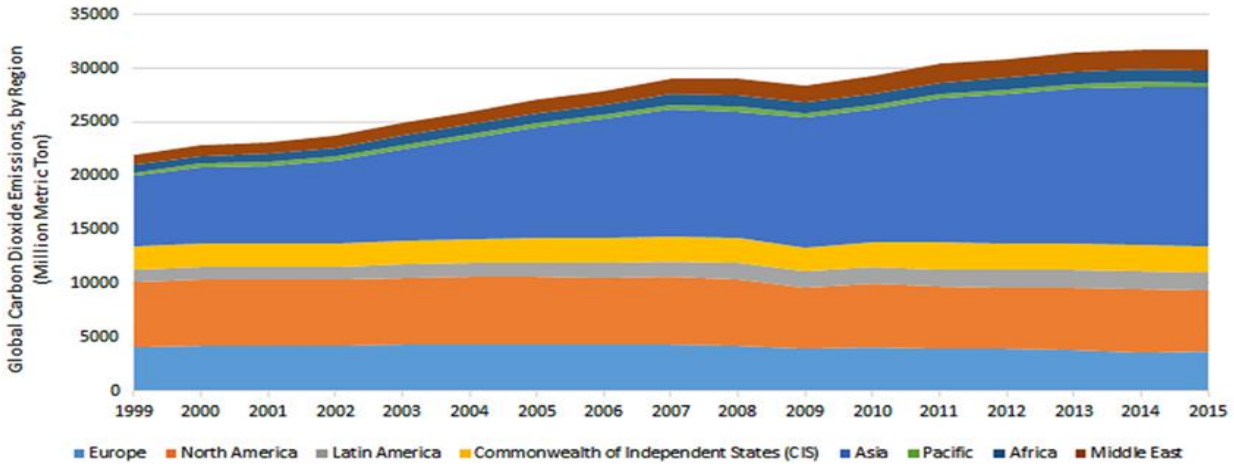

Figure 1-2. Regional Analysis of CO_2 emission throughout 1999 to 2015 [9]

Thus, naturally occurring renewable sources such as solar, hydro, wind, biomass, hydrogen and geothermal have been introduced to address such concerns [6]. In contrast to non-renewables, these sources are inexhaustible [13]. Renewable energy sources do specialize in being abundant, sustainable, and eco-friendly, which in turn offers a cleaner and more reliable energy solution, adding more to the reason as to why they are getting more attention lately. By investing in renewables like solar energy, environmental harm such as CO_2 emissions can be reduced. This overall ensures clean energy's availability to future generations.

Now speaking of renewables, solar energy is considered one of the most valuable energy sources on earth. In fact, solar radiation alone gives earth about eleven thousand times the global energy use [7].

1.2. Problem Statement: Limited Monitoring in Solar Energy

There's no doubt that solar energy's rise in popularity means that the current existing technology needs even greater progress. Environmental factors such as sunlight's intensity, temperature variations, and obstacles from clouds have a big impact on how well solar photovoltaic (PV) systems perform. Such variables directly affect the power output of solar panels, which is why it is necessary to frequently check on the system and its surrounding environment. Typically, solar modules or panels need to be monitored with instruments like multimeters, but this proved to be difficult and often considered impossible for large-scale systems or systems that are in remote areas [49]. This means solar panels have to wait for their next maintenance to find out if there are any issues due to dust accumulation, overheating, or shading from clouds or any problems that can influence the system's efficiency and decrease the expected energy generated. In other words, the manual methods simply do not provide real-time feedback on the system's performance and that is where solar monitoring systems have come into play.

(a) (b)

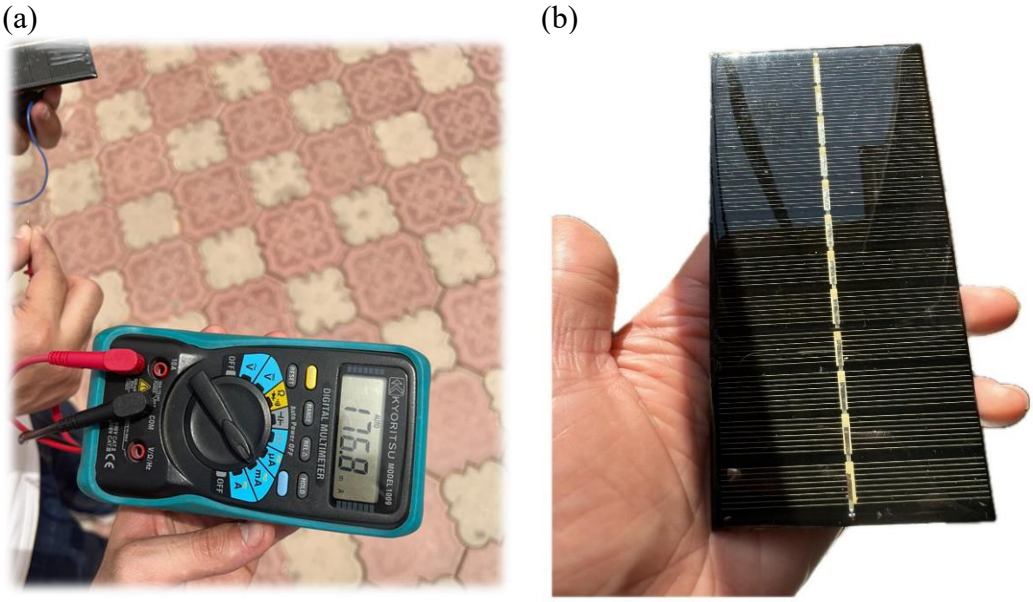

Figure 1-3. Manual Monitoring (a) Multimeter Measurements (b) holding Solar Cell

In order to handle these issues, researchers have recommended using Internet of Things (IoT) systems for monitoring, so data can be accessed fast, remotely, and all automated faults can be identified. Combining various devices like LDRs, temperature/humidity detectors, and current/voltage sensors with microcontrollers such as Arduino, NodeMCU, or ESP32 enables these systems to track both the environment and the electrical performance parameters [50,51] As a result, the impact of environmental conditions on the system is considered during analysis and optimization. An example for that is that intense heat can cut down the efficiency of solar panels even when there is a lot of sunlight, and unexpected cloud cover can throw off measures of how well they perform without checking the actual weather conditions through real time monitoring [52].

It is now clear that without automated weather tracking systems, it becomes utterly difficult to detect issues within the system. If power output suddenly falls, it might be due to shadows of a structure, or it might just be due to a serious defect in the panel. It's difficult to know the difference without comparing it to other nearby readings. To solve this, IoT-based systems look at how weather impacts power output, making it easier to manage and improve the solar system [53, 59]. For instance, systems like the one suggested by Sheikh in [51] adds temperature and light intensity sensors to help operators tell if a drop in performance results from a temperature issue, a lack of sunlight, or some kind of hardware error.

Figure 1-4. Solar Modules under Different Weather Conditions (Sunny VS Cloudy)

Additionally, in areas with variable weather conditions, such as tropical or monsoon-prone regions, real-time environmental monitoring becomes even more critical. In these climates, solar power generation systems become highly vulnerable to a frequent shift in irradiance and temperature, which can then cause the system to operate unpredictably and might place some stress on the equipment. Employing platforms such as ThingSpeak and AWS makes it possible to watch over, keep, and study data about the environment from a distance, supporting long-term analysis of changes that may influence the solar energy supply [49, 59]. These functions help avoid situations of poor energy usage, late fault detection, and more money being spent on operations.

Overall, monitoring weather conditions surrounding solar systems using the traditional methods is considered outdated and can become an obstacle in increasing how efficient and reliable power generation can be. By using IoT, system designers can monitor the environment as well as energy output to allow for a smart approach in handling these issues and ensure both reliable forecasts and enhanced performance.

1.2.1. Solar Cells Efficiency

Before addressing problems themselves, one needs to understand efficiency calculations foremost. A key parameter in evaluating performance of a solar cell is known as the Fill Factor (FF), which provides insights about how well the cell gathers the carriers generated by solar radiation as well as the overall quality of the semiconductor junction [21]. FF can is calculated using the formula shown, which utilizes data obtained from the I-V characteristics curve.

$$FF = \frac{V_{mpp} I_{mpp}}{V_{O.C} I_{S.C}}$$ *Equation 1-1 Fill Factor*

However, the primary parameter to be calculated here is the power conversion efficiency (η). It is calculated using the formula [21]:

$$\eta = \frac{P_{max}}{P_{input}} = \frac{V_{mpp} I_{mpp}}{P_{input}} = \frac{V_{O.C} I_{S.C}\ (FF)}{P_{input}}$$ *Equation 1-2. Solar Cell Efficiency*

Whereas efficiency is a relation between maximum output power (P_{max}) and total input solar power received by solar radiation (P_{input}), where maximum power is calculated by multiplying max voltage with max current. Additionally, FF can be substituted inside and giving the final formula shown above [21].

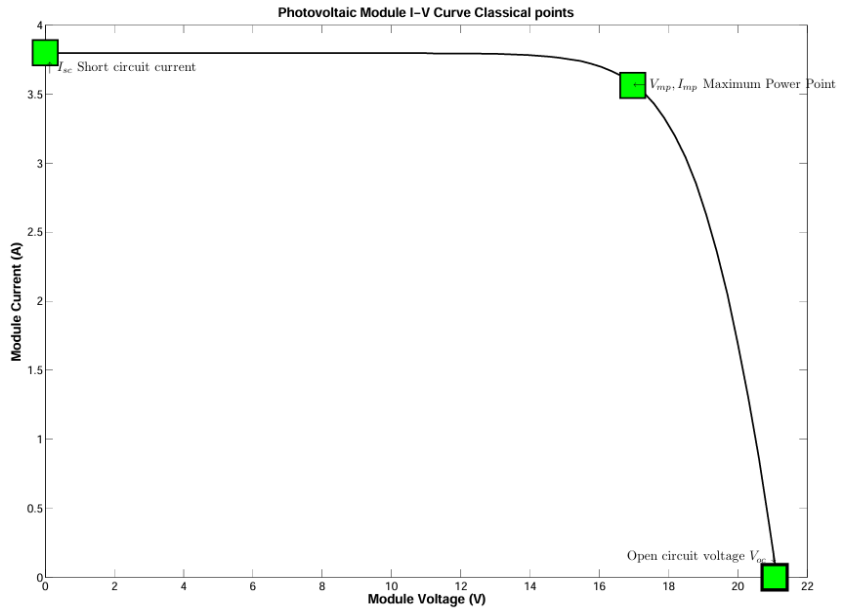

Figure 1-5. Typical I-V characteristic curve of PV cells [21]

As mentioned, several factors such as temperature and light intensity can affect the performance of PV cells, which presents challenges in the solar energy industry. For instance, one of the most popular factors is the dust particles blocking solar radiation from solar modules, which causes drastic power losses, thus, affecting the overall efficiency of the solar cell [22].

For decades, researchers have relied on tools such as solar simulators to test and evaluate the efficiency of solar cells. Solar simulators provide an artificial lighting that closely mimics sunlight [15]. Without going into much detail about these tools, solar simulators can explicitly test PV technologies by measuring the I-V characteristics of solar cells, and based on that, both performance and efficiency can be evaluated [16].

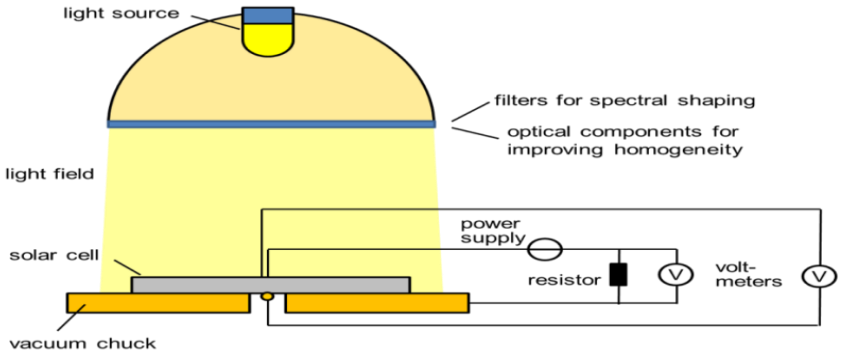

Figure 1-6. Test Setup of PV solar cells using a solar simulator [20]

Anyhow, solar simulators contribute greatly to PV technology advancement by testing them. Testing of the solar cells is one of the main methods used for optimizing them. Furthermore, one of the crucial considerations that researchers must keep in mind when testing solar cells is STCs. Standardized Testing Conditions or (STC) are conditions all researchers must comply with when evaluating PV cells [21]. These conditions are essential as they provide a reliable testing framework for solar cells from different manufacturers, ensuring efficiency ratings are fairly comparable since they are all now tested under same conditions.

STC includes having:

- Irradiance of 1 sun ($1000 W/m^2$)
- Distribution of Light Spectra at AM1.5G spectrum
- Temperature of 25 degrees Celsius with ±2% tolerance

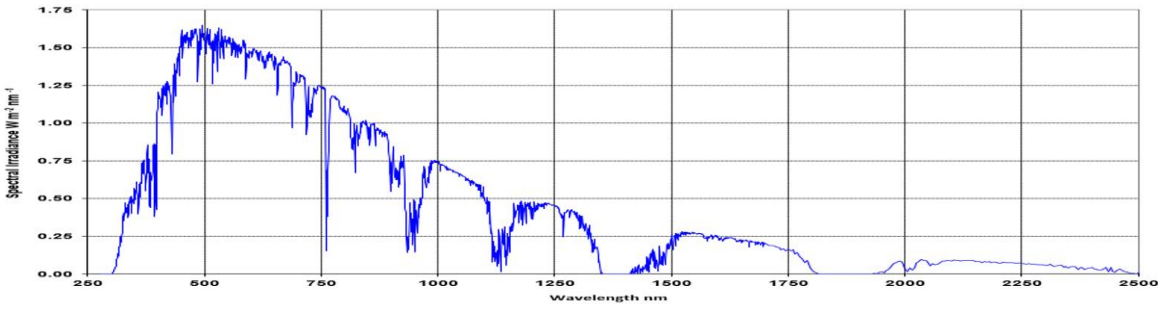

Figure 1-7. AM1.5G Solar Reference Spectra for PV evaluation [23]

With all that said, the problem that this research aims to address is the much-needed advancements for PV cells technology. This paper aspires to find a solution to help deal with several factors that might affect the efficiency of the solar cells, specifically by real-time monitoring weather conditions due to their significant impact on efficiency.

1.3. Research Objective

The aim of this research is to design and integrate an AI algorithm with an IoT framework utilizing a microcontroller, alongside a temperature & humidity, light intensity and current sensors for the sake of this phase of the project. These sensors have the purpose of detecting factors that might affect solar cells' efficiency. Potentially, other sensors might be added in the future. For this phase, however, the project will settle on utilizing the three mentioned sensors only for now. The central objective of this system is to improve the efficiency of solar cells by continuously monitoring the environmental and electrical factors that impact their performance. Throughout AI employment, the system will analyze real-time data gathered from these sensors to detect patterns, correlations, and anomalies that could influence solar energy production. Variables such as ambient temperature, humidity, solar irradiance, and fluctuations in electrical output will be processed by the AI model to forecast and address potential inefficiencies. AI model is yet to be chosen, with SVM (Support Vector Machine) and Random Forest model being potential candidates. The AI-driven optimization strategy will allow the system to dynamically modify operational parameters, including load distribution or panel orientation (if applicable), to enhance energy output and extend the lifespan of the solar panels. The microcontroller will act as the primary processing unit, enabling data collection, wireless communication, and the implementation of the AI model. This research aspires to illustrate how AI-enhanced IoT systems can advance smart energy management.

1.4. Research Questions

This research aims to answer these questions:

❖ What challenges and potential limitations do solar power monitoring and optimization face when using both IoT and AI technology while utilizing a Microcontroller?

❖ To what extent will integrating IoT systems and AI help monitor solar cells, and further optimize the performance?

❖ What are the factors affecting solar cells' efficiency that the IoT systems can detect while utilizing sensors? And what are the most appropriate sensors for each factor to be integrated with the system? All these questions will need to be answered for reliable monitoring.

❖ What benefits and drawbacks exist among different AI predictive models used to predict solar energy production while finding system abnormalities? Which raises another question: What is the most optimal AI algorithm model for this prototype of a system?

❖ What benefits does real-time artificial intelligence help us achieve with solar energy and operations?

❖ What are the predictive insights that the AI model can gather from sensor data in order to optimize solar cell's efficiency?

2. Chapter (2): Literature Survey

2.1. Overview of Solar Energy Systems

Solar energy technology is the second most installed renewable energy source just after hydro power [5], and yet again, due to the declination of fossil fuels and the rise of the global warming phenomena, it has never been a better time to start shifting toward renewable energy sources [8]. Recently, solar energy has started to gain more recognition as one of the most prominent renewable energy sources, as there has been a major increase of 22% in solar energy capacity just in one year (from 2021 to 2022) [5], and that is rightfully so due to its cleanness as an energy source. Practically, sunlight is to be considered as limitless, which makes it a perfect candidate for a clean reliable energy source. Unlike its adversaries, one of solar energy's main advantages over all non-renewable sources is that it is in fact a clean source to produce electricity without affecting the atmosphere [5,6].

Two of the main solar technologies used nowadays are: Photovoltaic (PV) cells and Concentrated Solar Power (CSP) systems, each with their own unique properties [5,11]. PV solar cells work by converting the sunlight into electrical energy or simply electricity. PV cells absorb light and loosen the electrons, which in turn generates a DC (Direct Current) that can later be inverted into AC (Alternating Current) [11]. On the other hand, CSP's operation involves concentrating sunlight using reflectors to generate heat, and then it converts that heat into electrical energy [11], with CSPs being mostly employed in intense solar radiation areas [12]. Generally, PV cells are more widely installed globally [5]. They are preferred due to their accessibility, being more versatile and having lower costs [11]. China is known to have the largest PV market, holding approximately one third of the world's capacity. As a matter of fact, Asia overall, as continent, will continue to lead the PV technology market by 65% of installations worldwide by 2030 [5].

(a) (b)

Figure 2-1. (a) PV Panels (b) CSP Concentrator [11,14]

To sum up, solar panels can be installed in a variety of locations, making them adaptable to various environments that require energy needs. This adaptability ensures that solar energy can serve all types of communities such as urban, suburban and rural. Nonetheless, solar energy continues to advance and become more and more prominent in the industries as years pass by. It also offers energy-independence and security for users. By producing power locally, it massively reduces the reliance on non-renewable fuels. As for business owners, installing solar panels leads to significant savings on electricity costs over time.

However, as every rose has its own thorns, while it might seem that solar energy is such a booming technology, further adjustments in efficiency are still rather needed. Several factors can affect the efficiency of these PV solar cells, which can be problematic. These factors include temperature, light intensity of solar radiation, weather conditions, parasitic resistances, and the actual design of the solar cell, and the list goes on. All these factors can contribute greatly to reducing the efficiency [17].

The main structure of a solar cell consists of a P-N junction, containing a P-type region and an N-type region, which is essentially a diode that has been carefully designed to generate power by absorbing sunlight's energy photons [17]. PV energy utilizes electronic semiconductor materials such as crystalline silicon, thin films, and much more. Classification of PV cells materials can be observed in Figure 2-2.

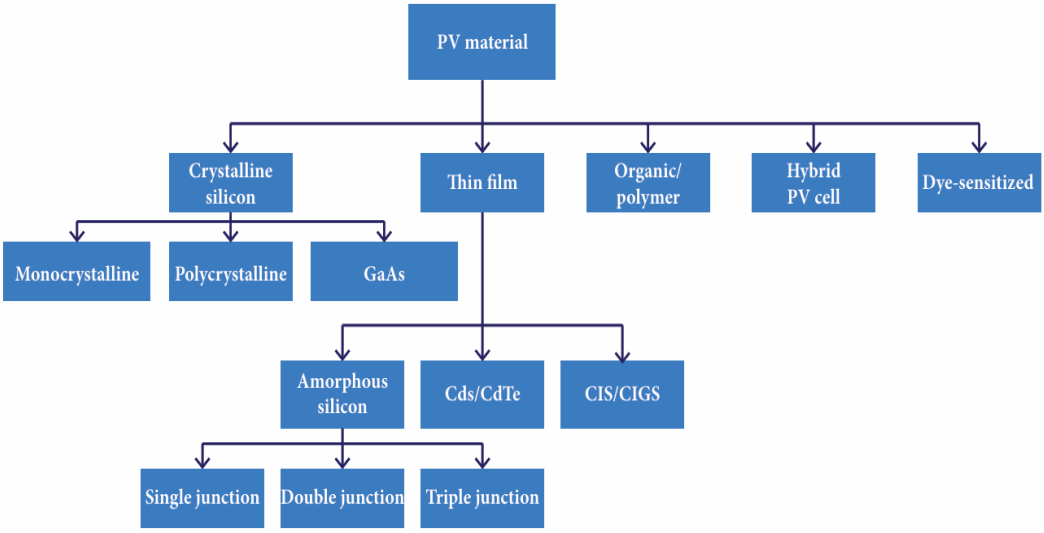

Figure 2-2. Classification of PV Material [19]

PV systems consist of a solar panel, a super capacitor that provides a backup additional energy source, and an inverter, whereas a solar panel consists of multiple solar cells that are connected in series or parallel formation, all with the purpose of generating a specified power output value [17]. As mentioned before, inverter simply converts the DC generated current into AC [11]. This inversion is necessary for conventional application purposes as can be seen in Figure 2-2.

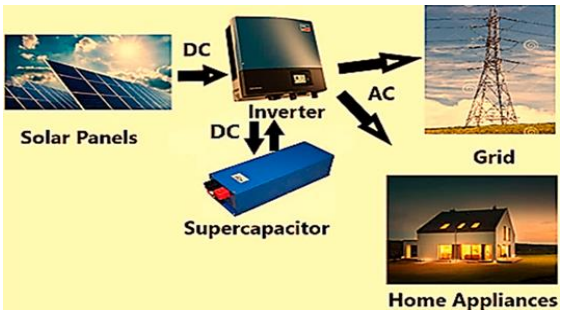

Figure 2-3. PV System Structure [17]

2.2. Literature Review: Reported Solar Monitoring Systems

An important question to be asked: what are solar monitoring systems? A solar monitoring system can be defined as a setup of a microcontroller and components such as sensors that have the main purpose of watching and analyzing PV cell performance, monitoring so-to-speak.

Reviewed studies indicate that typically, such systems measure voltage, current, power, and environmental things such as temperature and light by using sensors. The results of the sensors are stored in data loggers such as ATmega328, ESP32, or NodeMCU, and then passed on to the cloud using Wi-Fi or mobile techniques [53, 59, 60]. The system's main goal is to help the installation produce energy efficiently, be remotely managed, and keep problems at bay for locations that are far from service centers. Various papers on solar observing systems with their diagrams have been found and will be explained in this review.

In [49], a real-time solar power monitoring system using IoT was developed. This system is powered by a NodeMCU microcontroller with an ESP8266 Wi-Fi module to both collect data and send it from photovoltaic panels. A sensor known as INA219 and others like a digital temperature sensor and LDR are used by the system to measure voltage, current, temperature, and the level of light. When data from the sensors is sent to the ThingSpeak cloud, it can be seen remotely from a web app. It is possible to find faults, assess performance, and take care of the system using automated tools, which makes solar systems run more smoothly and effectively.

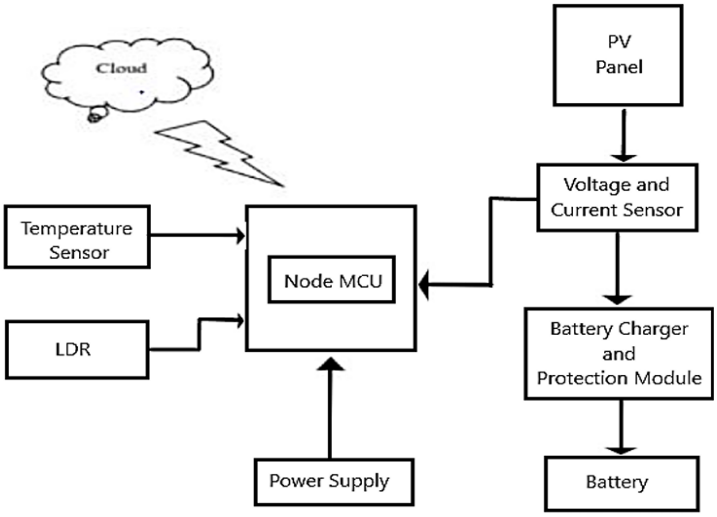

Figure 2-4. Solar Monitoring System in [49]

Similarly, in [51], a real-time solar energy monitoring system was suggested using a voltage, current, temperature, and light intensity sensor integrated with an ESP32 microcontroller. The authors used the Blynk mobile app and LCD displays to show the data as it occurred. Watching these five important variables, the system was able to find out quickly if there was overheating or less sunlight due to clouds and then accordingly, take action. Being able to control the ESP32 from the cloud and with mobile devices was crucial, since it makes it easy for users to operate and check the system. What makes this system unique is its simple design and pricing that helps with small-to-medium solar installation [51].

The work reported in [59] introduced a cloud-connected system also done with the ESP32 Arduino master board and introduced MQTT messaging system, while AWS provided cloud services. They worked on making the system easy to change when needed and scalable in the future. Information about voltage, current, and power production was immediately recorded on AWS so that it could be viewed and analyzed using specially designed dashboards. Because of this arrangement, analysis of historical trends allows the identification of the most likely future issues. This research realized that cloud analytics and visualizing data strengthen the choice-making process and that this could easily be used in large solar farms and decentralized energy schemes.

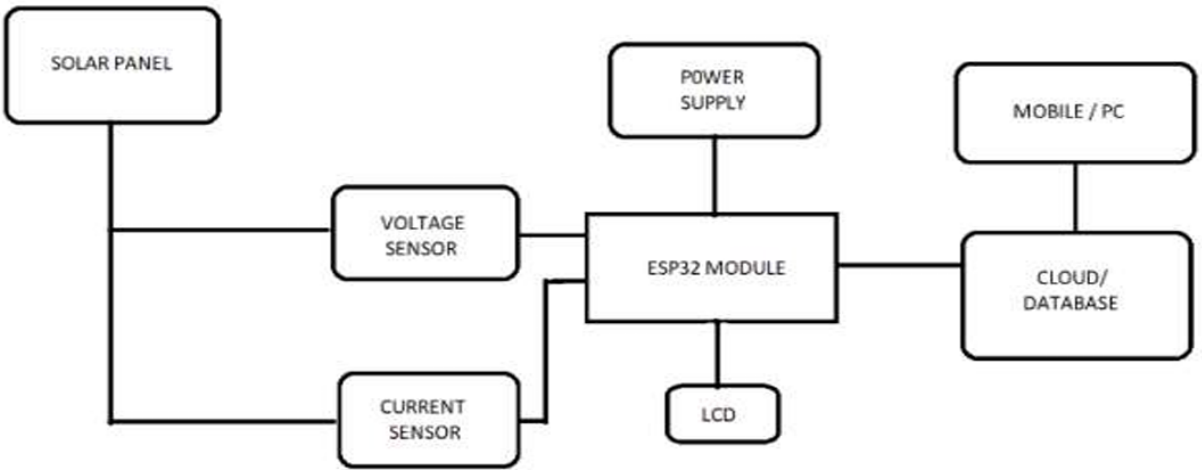

Figure 2-5. Solar Monitoring System in [59]

Meanwhile, a system in [53] opted for a simple but functional architecture by combining the ATmega328 microcontroller with the INA219 voltage and current sensors. In order to visualize live data and find trends, this system was connected to ThingSpeak, a popular IoT platform that is free to use. It was perfect for businesses located in rural or remote regions because it was very economical. Besides, this system used alerts so that notifications were produced if the power output fell below particular values, which might have happened due to soiling, shade, or problems with the hardware. Thanks to this feature, issues can be identified ahead of time, which results in less time when the system is down.

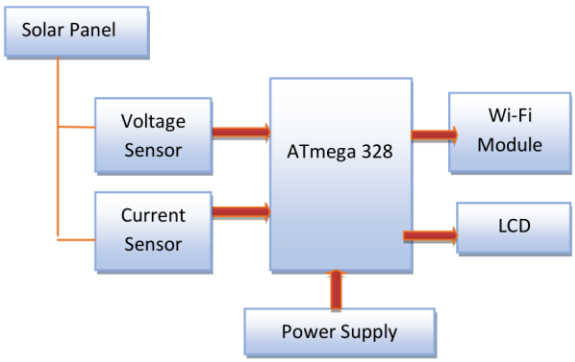

Figure 2-6. Solar Monitoring System in [53]

A similar approach was implemented in [52], where the Raspberry Pi and Flask web framework were blended to make a web solar monitoring system. Thanks to this web server, the system had more control over handling and displaying different sets of data compared to basic mobile applications. It kept tabs on voltage, current, and power and also allowed environmental sensors such as temperature and irradiance monitors to be included. Authors suggested that the same system could also be used in solar cities or microgrids, where a centralized dash for multiple panels or systems would be required.

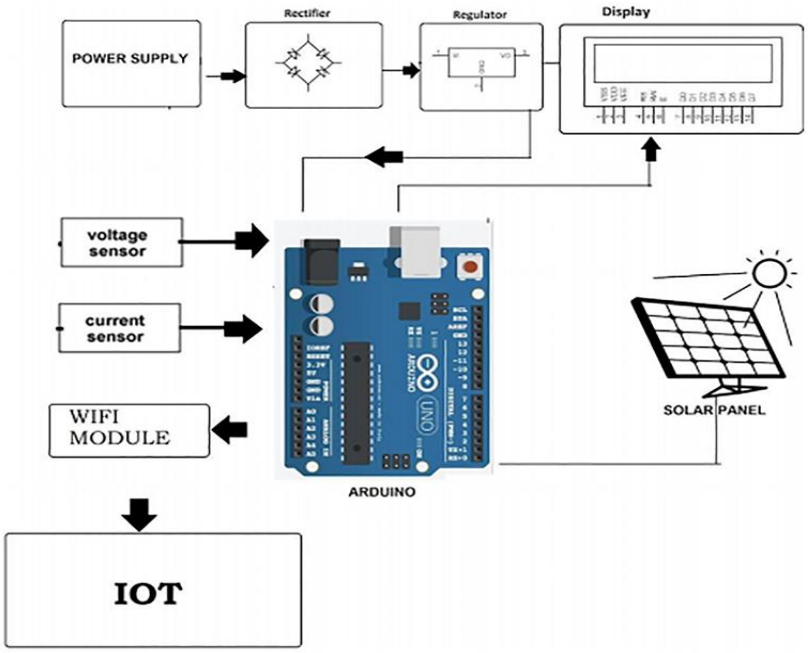

Figure 2-7. Solar Monitoring System in [52]

In [60], a study discussed how to improve solar energy efficiency with a hybrid weather and solar monitoring system. Besides being monitored, solar panels were included in the design to power the entire system. The system used NodeMCU (based on the ESP8266) to measure important atmospheric elements such as temperature, humidity, air pressure, and the intensity of light. The sensor data was put into the cloud so that people could look at it and keep a copy. This work was notable for paying special attention to observing the site's performance and running it without a grid connection using solar energy.

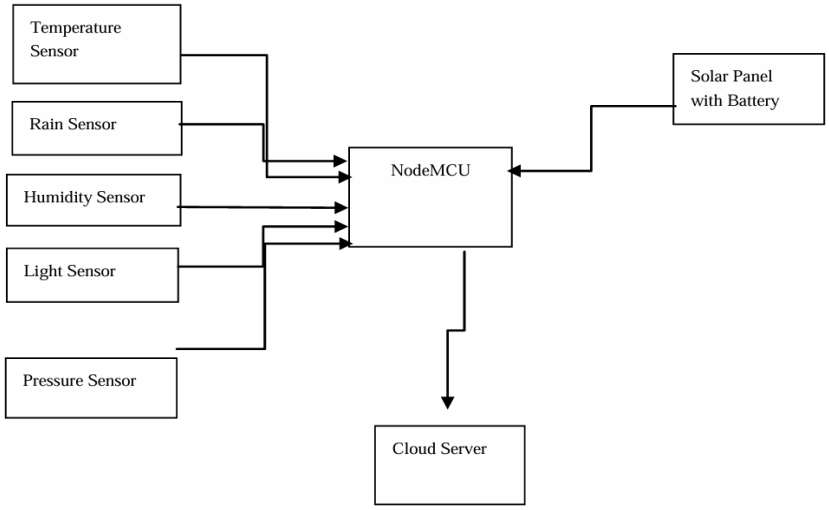

Figure 2-8. Solar Monitoring System in [60]

Regardless of whether it is ATmega328, ESP32, NodeMCU, or Raspberry Pi, each system uses the same architecture by collecting data, processing it, and routing it to the cloud. By using this modular plan, these systems can be changed to fit areas from small residential areas to huge industrial projects.

Apart from the environmental benefits, another important area in the literature is how sustainable and accessible these systems can be. For instance, some researchers have decided to use open-source materials and free cloud systems (Such as ThingSpeak, Blynk and Firebase) to avoid spending too much money and to ease implementation. By replacing manual inspections with tools from IoT and using predictive maintenance, systems designed for solar monitoring are more reliable, conserve more energy, and help the environment for a longer time. Artificial intelligence and advanced analytics are likely to be used in the future to boost these systems, link them with smart network grids, and add environmental data for even better and automatic control.

2.3. IoT in Solar Monitoring

Internet of Things technology lets solar energy systems get real-time monitoring using key components like sensors and ESP32 microcontrollers integrated together in a communication network. Solar energy management improves substantially through IoT by collecting constant data about environmental and electrical parameters which enables enhanced decision-making and efficiency enhancements [29] .The systems can incorporate crucial sensors featuring temperature sensors for monitoring panel temperature and light detection sensors (LDR) for sunlight evaluation as well as current and voltage measurement sensors [26]. The sensors provide efficient performance evaluation together with fault detection capabilities [27]. IoT also provides a user-friendly interface [18].

The function of Microcontrollers consists of receiving information from sensors followed by data transmission to cloud-based platforms [31]. The processing of local data reduces the time required through edge computing [28].

Communication Protocols: Solar energy systems use MQTT and HTTP and Constrained Application Protocol to establish efficient communication between solar devices, cloud databases and user interface dashboards[30].

2.3.1. Communication Modules

The essential need for reliable data transmission exists between System Components that include Sensors, Microcontrollers, and Cloud Platforms to enable Real-time monitoring together with analysis. Communication protocols determine the data flow procedures among system elements.

- The Message Queuing Telemetry Transport Protocol (MQTT)

It functions as a Lightweight Publish-Subscribe System that specializes in IoT applications. The minimal bandwidth usage capabilities make this protocol perfect for solar monitoring systems situated in distant areas because it allows efficient data exchange. An MQTT broker receives sensor data from ESP32 before AI models based in the cloud can use this data through simple integration [29].

- The Hypertext Transfer Protocol (HTTP)

It helps the ESP32 microcontroller establish an immediate connection with cloud servers to exchange data in Real-Time for dashboard monitoring together with System Control. HTTP requests as GET and POST methods enable the ESP32 to transmit sensor readings to Databases while simultaneously letting it obtain AI-based optimization directives and activate system modifications. Web applications rely on HTTP for reliability, but MQTT offers more bandwidth efficiency which makes it better for real-time streaming data. Representational State Transfer Application Programming Interface (RESTful API) integrations enable easy connection to AWS, Google Cloud and Firebase platforms which allows centralized solar energy system monitoring and control from any location [30] .

2.3.2. Data Collection

Data collection involves gathering key parameters essential for monitoring and optimizing solar energy generation. This includes sunlight intensity, which directly affects power output, as well as temperature and humidity levels, which influence panel efficiency [35]. Additionally, current and voltage measurements are recorded to assess the system's overall performance. The collected data provides valuable insights into the solar panel's health and efficiency. Variations in sunlight intensity directly impact energy production, while temperature and humidity levels influence the panel's performance over time. Abnormal current or voltage readings may indicate potential faults, such as wiring issues or panel degradation. By analyzing these parameters, the system can detect inefficiencies, predict maintenance needs, and optimize energy generation for maximum output [34].

2.3.3. Data Analysis & Preprocessing

The preprocessing of data requires immediate attention because it determines both the quality and integrity of data before machine learning model usage. The data processing demands following vital procedures: Raw data includes many irrelevant or incorrect elements that important patterns which can be removed through filtration techniques. To improve signal quality various smoothing algorithms eliminate measurement noise [39]. The absence of data values leads to faulty model conclusions due to improper handling. Two methods exist to deal with missing data: researchers either substitute predicted values through data imputation or eliminate records lacking information [40]. Model performance gets negatively affected because data features exhibit different value scales. Normalizing data with methods like min-max scaling or z-score standardization gives data a standardized scale that maintains original distribution ranges in values [39].

Pattern recognition becomes possible with aggregated data that results from combining multiple datasets together or condensing data into specific time intervals [41].

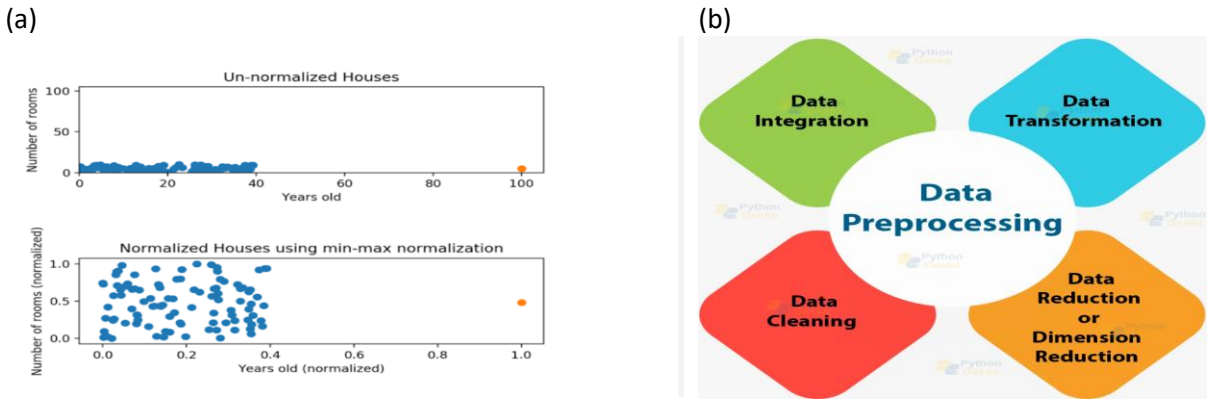

Figure 2-9. (a) Data Normalization (b) Data Preprocessing [54,55]

2.3.4. Predictive Analysis

The forecasting of energy production together with anomaly detection and maintenance requirements prediction functions through predictive analytics in solar energy systems. Time series analysis serves as the standard approach to produce energy output predictions which use collected historical data. The ability of LSTM networks to detect temporal relations in sequential information makes them an ideal choice for energy production forecasting tasks. Autoencoders function as neural networks which learn to reconstruct input information and produce anomalous detection when original and reconstructed data differ substantially [43].

2.3.5. Optimization Algorithms

Improving solar energy systems depends on the combination of optimal parameters which optimize their energy production capabilities. Reinforcement Learning (RL) operates as an agent learning method which teaches decision making through action-performance followed by reward-feedback. Solar energy optimization includes the use of RL to automatically modify solar panel tilt angles which optimizes the overall energy collection capabilities. The concept of natural selection drives Genetic Algorithms (GA), which serves as another approach for optimization. The optimization process starts by creating multiple candidate solutions that live through successive generations while focusing reproduction on the most successful options. GA demonstrates effectiveness for optimizing energy-efficient production schedules through their ability to handle multiple objectives combined with multiple constraints [44].

(a) (b)

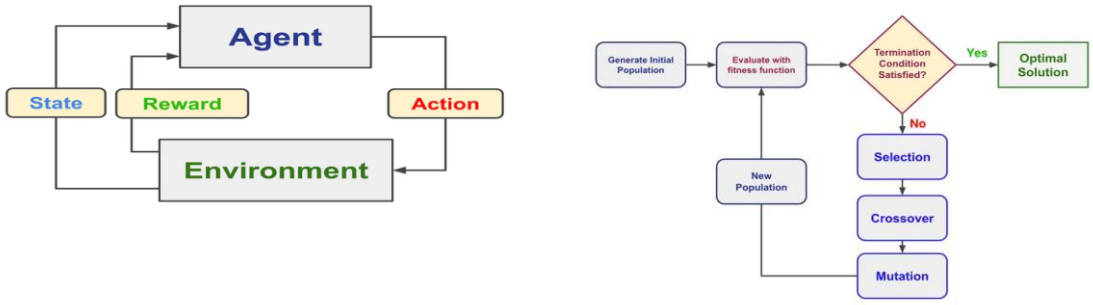

Figure 2-10. (a) Reinforcement Learning (b) Genetic Algorithm [44]

For example : There have been an implemented RL-based system which managed to optimize solar panel tilt angles throughout daily operations for improved energy capture results. A Q-learning and Deep Reinforcement Learning (DRL) method worked together to determine best suitable tilt angles by processing solar irradiance together with weather information. The RL agent adapted through time to real-time environmental changes thus increasing solar power efficiency by 18% above traditional stationary panel configurations [45].

So, The Artificial Intelligence learns such training processes through Supervised Learning that relies on historical data. The training process for predictive analytics uses historical energy production data to create models which discover patterns and relationships that allow them to make predictions [46]. The trained Artificial Intelligence models undergo validation using real-time inputs to determine their operational effectiveness together with their ability to generalize. The validation procedure verifies that the created models successfully predict energy output values for upcoming periods and find unexpected patterns across diverse operational environments [47].

2.4. AI Techniques

Random Forest and Support Vector Machine (SVM) models are some of the most compatible algorithms for this specific project. Random Forest is used for fault detection, using multiple decision trees to classify system states and improve accuracy [32]. Its ability to handle noisy and complex data makes it effective for identifying faults in sensor readings. SVM, on the other hand, models non-linear relationships by mapping data to higher dimensions and fitting a hyperplane to predict continuous values like energy output, making it ideal for predicting solar generation based on varying environmental factors.

The training process for both models involves using a dataset that includes sensor data, along with other historical datasets.

	A	B	C	D	E	F	G	H	I	J	K	L
1	Date	Time	Pm	Voc	Isc	Tbom	Gpoa	Gglob_hor	Gdiff_hor	RH_AIR	AIR_TEMP	WIND_SPEED
2	01.07.2018	7:05:00	99.556	36.8285	3.44285	36.926376	391.7520581	488.3233333	70.64666667	52.64	24.74	0.32466667
3	01.07.2018	7:10:00	104.164	36.7566	3.62592	37.7741009	408.7499749	501.8	71	51.84	25.1	0.31666667
4	01.07.2018	7:15:00	109.001	36.7979	3.78785	37.8629406	426.5214765	515.5133333	73.20333333	50.94	25.18	0.878
5	01.07.2018	7:20:00	114.072	36.8655	3.96429	38.4461849	443.880778	529.6066667	75.30666667	51.52	24.8	0.44066667
6	01.07.2018	7:25:00	118.973	36.8881	4.13587	38.961256	461.9541555	543.6266667	78.17	50.8	25.1	0.43
7	01.07.2018	7:30:00	123.166	36.822	4.30072	39.8595783	477.6197478	555.64	81.55	50.56	25.16	0.77933333
8	01.07.2018	7:35:00	126.988	36.7834	4.45861	40.656081	492.4850731	567.6933333	83.44333333	52.12	24.98	0.64733333
9	01.07.2018	7:40:00	131.017	36.7423	4.60613	42.2361645	509.2016684	581.01	84.69	51.08	25.32	0.24933333
10	01.07.2018	7:45:00	134.627	36.6649	4.75372	43.2782982	523.2646309	591.15	86.20333333	50.04	25.86	0.78666667
11	01.07.2018	7:50:00	138.897	36.5921	4.9343	44.5568104	540.2458886	605.5133333	86.09	50.72	25.42	0.478
12	01.07.2018	7:55:00	143.038	36.4461	5.10213	45.5048447	557.9847325	620.2266667	86	50.36	25.62	0.35733333
13	01.07.2018	8:00:00	147.125	36.4392	5.27733	45.9008385	575.076084	633.3333333	87.99666667	48.54	26.22	0.59666667
14	01.07.2018	8:05:00	151.875	36.5458	5.43456	45.7809757	590.7182645	645.5033333	92.33333333	48.76	26.1	0.89333333
15	01.07.2018	8:10:00	155.848	36.5216	5.58922	46.7120766	606.5225778	658.4733333	93.53333333	49.32	25.96	0.84933333
16	01.07.2018	8:15:00	159.345	36.5578	5.71942	46.0190483	620.6459979	668.56	95.60666667	48.56	25.94	1.48266667

Figure 2-11. Historical Dataset Example

The dataset undergoes preprocessing to enhance its quality for training. This process involves cleaning to eliminate noise and handle missing values, normalization to maintain a uniform scale, and feature selection to determine the most significant variables for accurate predictions. The dataset is divided into training and validation sets to assess performance of models and prevent overfitting. The models are then trained on these subsets and later tested with unseen data to verify their ability to predict energy output or identify faults accurately.

These models are integrated into the system to continuously analyze incoming sensor data, enabling instant insights into the solar energy generation process. In real-time applications, Random Forest is employed for fault detection, classifying sensor readings as normal or anomalous by processing data streams as they are received [33]. This allows for immediate identification of any faults or changes in the system. Likewise, SVM analyzes real-time data to forecast energy output by considering current environmental factors and system conditions. These AI models enable the system to adjust operational settings and enhance performance with minimal delay.

AI enhances decision-making by analyzing sensor data to optimize system performance. It predicts the ideal tilt angle for maximizing solar energy generation and detects anomalies that may indicate potential faults or maintenance needs. By continuously learning from real-time data, AI ensures efficient operation and reduces downtime.

2.5. Project Approach

2.5.1. Hardware Components and Sensors

2.5.1.1. ESP32 microcontroller

The Primary duty of the ESP32 microcontroller involves managing sensor measurements as well as Data evaluation and functionality execution between System Elements. The dual-core processor and Bluetooth abilities and Wi-Fi integration and numerous General-Purpose Input/Output (GPIO) pins feature in microcontroller that makes it suitable for IoT-based applications [28]. The ESP32 microcontroller performs data processing efficiently because it reaches speeds of up to 240 MHz while integrating Analog-to-Digital Converter (ADCs). The device enables operation with minimal power requirements thus making it ideal for solar monitoring systems that need to be energy-efficient. The microcontroller properly distributes sensor information to AI-based algorithms and manages performance output while minimizing power consumption [28].

Figure 2-12. ESP32 Boards [56]

Better Solar monitoring performance requires Real-time data acquisition from different Sensors which must deliver precise Information. Different sensor devices detect exact Environmental data alongside Electrical Signatures which becomes the Fundamental input for Artificial Intelligence optimization systems [24].

2.5.1.2. Polycrystalline Silicon Solar Cell

Figure 2-13. Provided Polycrystalline Silicon Solar Cell

In the prototype, the solar cell is the main part that is closely examined and analyzed. The purpose is to produce electricity using sunlight, allowing the system to check important performance indicators such as open-circuit voltage (V_{OC}) and short-circuit current (I_{SC}). So, these values indicate how effectively the solar cell operates under various external conditions, making it easy to check its current energy output.

2.5.1.3. DHT22/AM2302

DHT sensor functions as a Digital Sensor to measure temperature as well as humidity with high accuracy that suits monitoring of atmospheric conditions that impact Solar Panel Performance. The sensor performs with two components which include a Capacitive Humidity sensor alongside a thermistor that transmits data through the single-wire digital protocol. Solar panel performance requires Permanent observation to assess how Temperature shocks and high moisture content affect its operation [25].

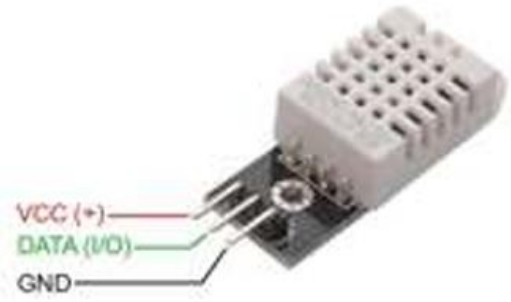

Figure 2-14. DHT22 sensor [50]

2.5.1.4. The LDR Sensors with LM393

It exhibits Solar Irradiance monitoring capability through Light-dependent Resistors that transform their Resistance values based on Light Intensity detection. The resistance variations from the LM393 comparator circuit lead to digital signal production. The devices detect shading effects and determine the best placement position of panels because these factors degrade energy efficiency [26].

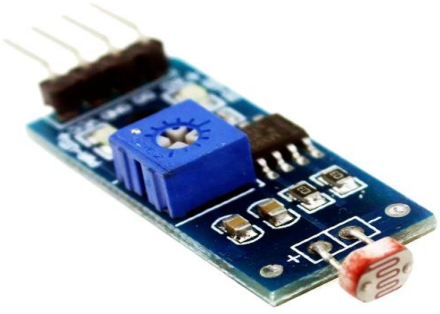

Figure 2-15. LDR sensor [26]

2.5.1.5. INA219 Current Sensor

The INA219 sensor in a solar monitoring system measures current with high precision, allowing real-time calculation of power output from the solar panel. It enables accurate monitoring of specifically the short-circuit current (I_{SC}) and overall energy production, which is vital for assessing panel efficiency and system health. Its low-power operation and I2C communication make it ideal for embedded solar data loggers.

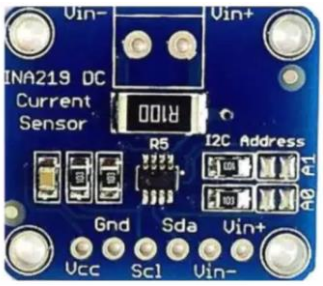

Figure 2-16. INA219 current sensor [27]

2.5.1.6. RTC Module PCF85

The Real-Time Clock (RTC) module in a solar monitoring system provides accurate timekeeping for timestamping collected data, such as temperature, humidity, voltage, and current. This ensures that every reading is associated with a precise date and time, which is essential for analyzing performance trends, detecting anomalies, and synchronizing with cloud platforms. The RTC maintains time even during power loss, making it reliable for long-term monitoring.

Figure 2-17. RTC PCF85 Module[61]

2.5.2. Data Management (Storage Methods)

Regarding storage of data, the preferred method in our case is the Hybrid Approach, instead of only local/cloud database. Hybrid Approach is a mixture of both cloud and local databases. In order to do that, an SD Card Module will be merged with the ESP32. An SD card module is a hardware component designed to enable microcontrollers or embedded systems to interact with SD (Secure Digital) cards [36]. SD cards are inexpensive and provide a large amount of storage (16GB, 32GB, and more). They also act as a backup; in case the cloud database fails/loses the data. After implementing the SD Card, the local database must be synchronized with the cloud database.

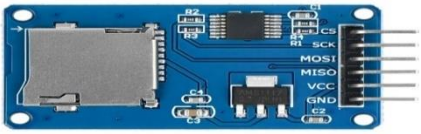

Figure 2-18. SD Card Module [36]

The local database to be potentially used is SQLite, which is a library that integrates a self-sufficient, serverless SQL database engine, offering transactional capabilities with no need for configuration [37]. It can be used in small-scale applications, mobile devices and embedded systems.

One of the biggest differences between SQLite and traditional databases is that it doesn't require a separate database server, but rather it stores data in a single file. It also makes managing structured data easy due to SQL queries. Using the SD Card mounted on ESP32, it will be integrated with SQLite, safely securing data. The data will be uploaded to the cloud database and subsequently processed for AI-driven optimization.

Figure 2-19. SQLite [57]

A proposed cloud database is Firebase Cloud Storage, which is provided by Google. Firebase in general is a platform that enables development of various applications for Android, IOS, and the internet. It is backed by Google [38]. It is easily integrated with Firestore, Firebase Realtime Database, and Firebase Authentication which are all part of Google's Firebase Ecosystem, providing controlled access and data management. In our case, it will store sensor data logs (in CSV or JSON format) from local database SQLite which is synced with ESP32. It can also store historical performance datasets and can be used for real-time monitoring by syncing with a mobile app or web dashboard.

Figure 2-20. Firebase Cloud Storage [58]

In order for all this to work, these 3 things must be seamlessly integrated together: SD Card – SQLite – Firebase Cloud Storage. The first step is integrating the ESP32 with an SD card module. Then, the SD Card and SQLite must be integrated, which is done by creating an SQLite database file where the sensor readings will be stored. The ESP32 will then simply read sensor data and insert it into the SQLite database. The final step is uploading the stored data to the Firebase Cloud Storage in a structured format (For Example: JSON or CSV). This will be done by a method called HTTP Requests. Overall, this approach will allow the system to function even without an internet connection while enabling cloud-based monitoring and analysis when connectivity is available.

Figure 2-21. ThingSpeak IoT Platform [62]

Finally, ThingSpeak is an open-source IoT platform that allows users to collect, store, analyze, and visualize sensor data in real time. It supports integration with MATLAB analytics, making it particularly useful for educational and research projects such as this one. Many solar monitoring systems utilize ThingSpeak due to its ease of use, built-in charting tools, and the ability to set up triggers or alerts when specific thresholds are crossed. For example, when the solar panel output drops below a defined limit, ThingSpeak can notify users via email or API integration [62].

- ✓ **For this project's sake, it was finally decided that this IoT platform, ThingSpeak, would be a fitting addition to incorporate in this type of solar monitoring system instead of SQLite and Firebase.**

2.6. Integration of both AI & IoT

IoTs are implemented to ensure the PV cells operate at the highest possible efficiency. IoT systems specialize in collecting real-time data which can detect these factors through a system of interconnected devices. IoT also allows for user-friendly interface that can be linked using protocols and a cloud system [18].

In addition, an AI machine learning model can be used to predict outcomes or efficiency based on the collected data, which in turn works in improving the overall efficiency of the solar cell. In this paper, a proposed method to address this issue is going to be discussed. The method revolves around integrating both AI and IoT, making the best out of both worlds, so-to-speak.

Integrating IoT with AI can ensure a powerful approach in efficiency improvement, contributing to reducing all negative effects caused by these so-called factors affecting PV cells.

3. Chapter (3): System Implementation

3.1. System Prototype

The proposed system of this project is simply a real-time monitoring system that combines both environmental and electrical parameters tracking by utilizing an IoT based embedded system. This system is designed to assess performance a solar cell that has been provided by the FabLab facility within the British University in Egypt. This solar cell is a polycrystalline silicon cell that is approximately 165x65mm in area.

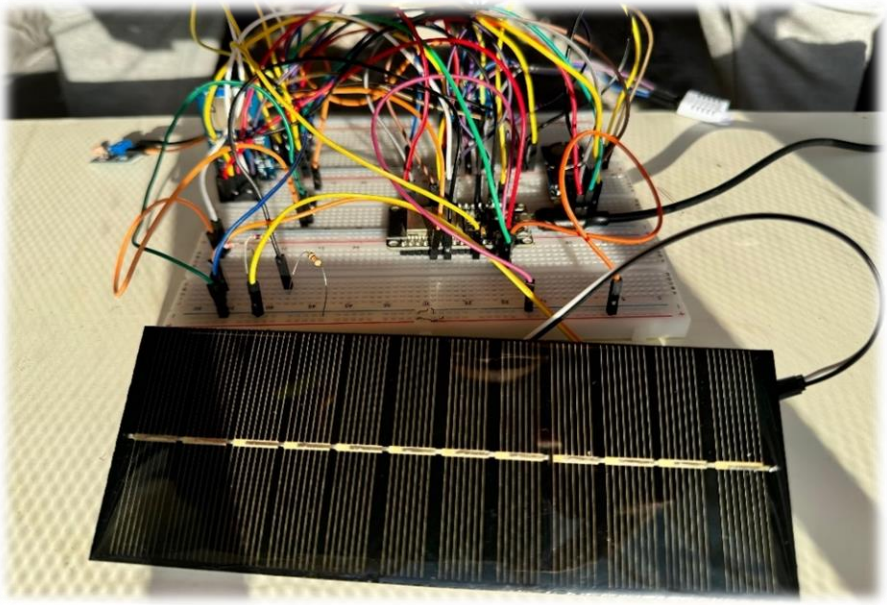

Figure 3-1. Implemented System Prototype

Now, the system simply monitors the performance of this cell by monitoring weather conditions. This is done by capturing environmental data like temperature, relative humidity and light intensity using DHT22 and LDR sensors respectively. Electrical parameters, on the other hand, include the open-circuit voltage (V_{OC}), measured by the voltage divider, and short-circuit current (I_{SC}) which will be measured using INA219 current sensor.

Both of these environmental and electrical parameters highly contribute to the efficiency of the solar cell and will be used mainly to allow monitoring of the current weather condition. This will let the user make smart decisions according to the current weather condition. Those include sunny, cloudy (shade) or nighttime where there's barely any sunlight. The measurements of all parameters will be collected at regular time intervals, saved on local SD card while at the same time will simultaneously be transmitted to an online cloud IoT system (ThingSpeak in this case) for remote access. As for the prediction AKA monitoring weather phase, this is exactly where the AI machine learning models get introduced. They will allow for the prediction of the current weather condition based on the mentioned parameters. The methodology of machine learning will employ a python-based code that will allow for prediction of the weather.

This is just a brief introduction to the purpose of the proposed system prototype in figure 3-1.

As for the design, at the core of the system lies the programmed ESP32 microcontroller which is considered the brain of this whole system. The ESP32 manages all the sensor reading and data storage. As mentioned before, the sensors DHT22 and LDR basically measure temperature, humidity and light intensity. Electrical parameters measurements, however, involve a little bit of understanding. The first electrical parameter, V_{OC} is measured across a voltage divider connected to the polycrystalline solar cell when the circuit is open then reading gets scaled to estimate actual value. As for the I_{SC}, it is measured using an employed sensor Adafruit INA219 under short circuit conditions. This system also utilizes an IRFD014 MOSFET to isolate both readings by switching in a synchronized manner.

(a) (b)

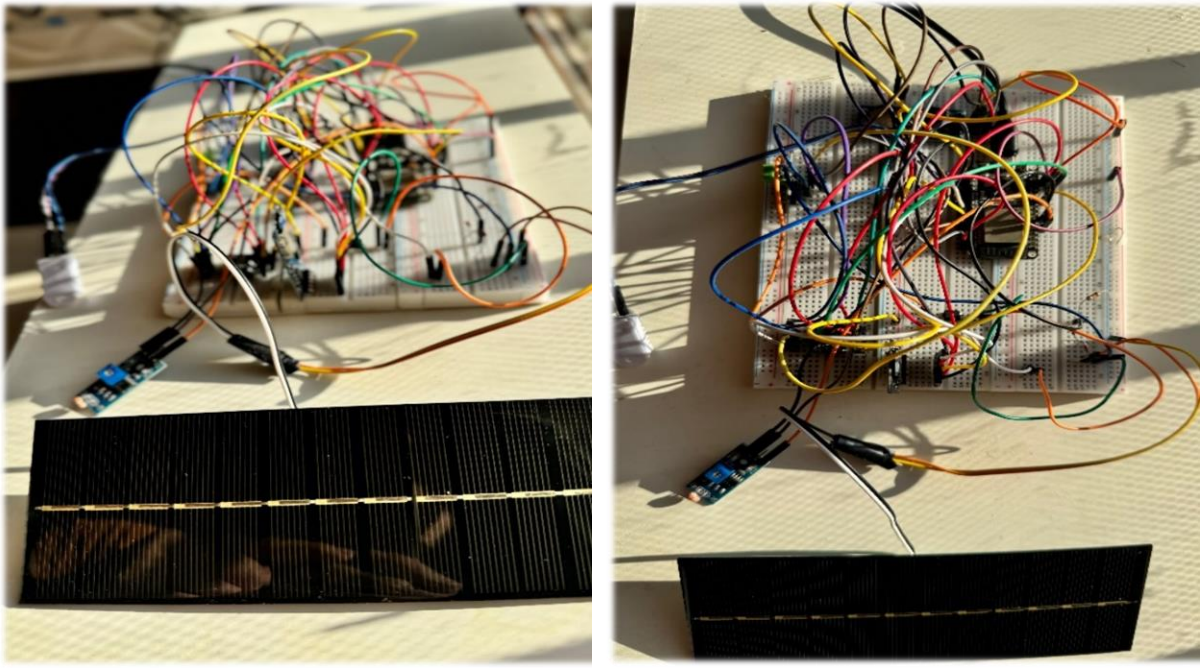

Figure 3-2. System Prototype (a) Front View (b) Top View

Data storage is done using two main approaches: a local approach using the SD card module and an IoT cloud-based system. The SD card module is employed here mainly to add redundancy to ensure that data is still collected in case any issue may occur during the cloud data transfer. Data is stored in CSV format that can later be processed on Microsoft excel. Time synchronization is provided by the Real-time clock (RTC) module PCF8563.

It is important to mention that the Wi-Fi connectivity is made possible by the ESP32's module that supports interaction with the cloud. After gathering the data, the system puts it on ThingSpeak using HTTP GET requests. By using cloud integration in real time, users can check how the system is running from any location using the preferred way to display data, for example as dashboards or charts.

The main goal of the system is to give a complete and a cost-effective approach for analyzing solar energy's performance in different environments. With the ability to monitor the conditions of the weather, the system helps detect errors, assess the top performance, and look ahead for forecasts. Also, being modular and scalable, the technology can easily work in schools, homes, or research settings.

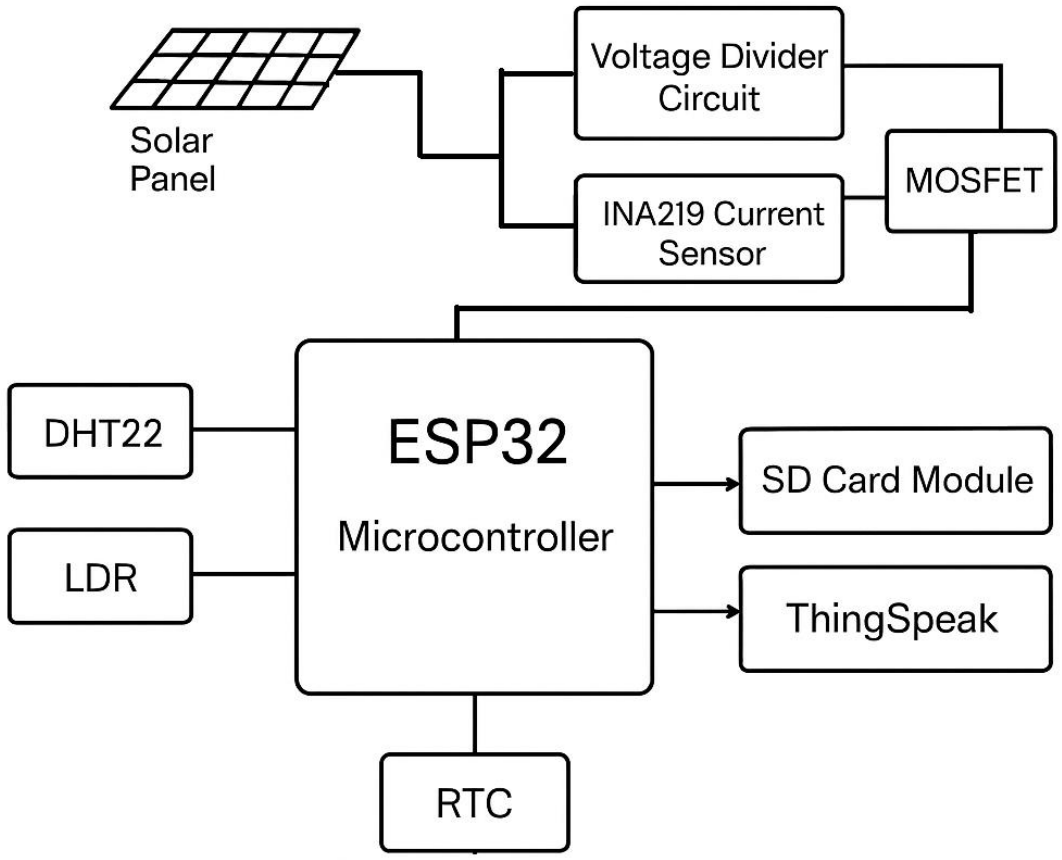

Figure 3-3. Block Schematic Diagram of System Prototype

Figure 3-3 shows the block schematic diagram of the proposed system operating in the processes described.

3.2. ESP32 Programming (ARDUINO IDE Code Explanation)

As mentioned earlier, the microcontroller basically serves as the central processing unit for this entire system. A code was written and uploaded on the ESP using Arduino IDE software which supports ESP32 through open-source libraries. The program code incorporates several functions, which include checking temperature and humidity using a DHT22, measuring light intensity through an LDR, reading electrical parameters with an INA219, and synchronizing time using the PCF8563. It is able to log data onto the SD card and also periodically transfer values to the ThingSpeak platform via Wi-Fi. This portion of the document explains the structure of the code and features important functions, all of which work together to help the solar monitoring system function dependably and smoothly.

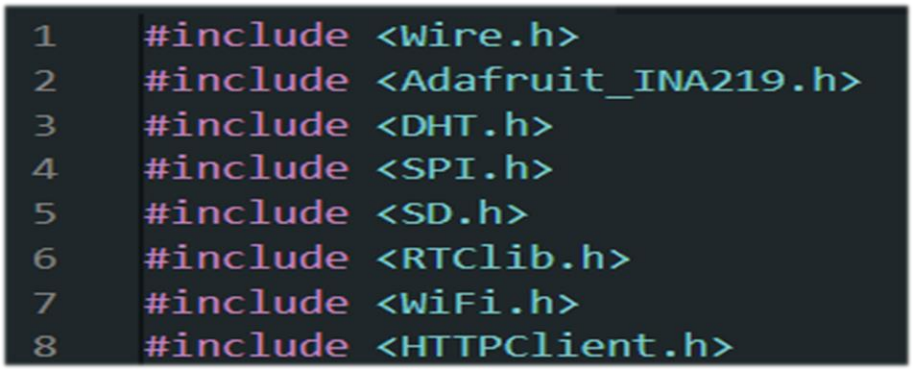

Figure 3-4. ESP32 Programming Code Screenshot1

The first section of the code contains the required libraries needed to correctly communicate the ESP32 with each hardware part in the solar system. With The Wire.h library, I2C protocol for communication is enabled which is required for sensors such as the INA219 and with the RTC module.

Moreover, Adafruit_INA219.h has functions that enable the user to connect with the INA219 sensor for reading current and voltage from the solar cell.

The DHT.h library permits us to check the temperature and humidity from the DHT22 sensor. SPI.h and SD.h are included so the SD card module can be used to save data locally in the format CSV.

RTClib.h enables the controller to send data to the PCF8563 real-time clock and correctly record each reading's time.

Then, the libraries WiFi.h and HTTPClient.h are used for creating a WiFi link and controlling which HTTP commands are sent to ThingSpeak cloud platform. All these libraries work together to support the main software of the system, making it strong and efficient for data collection and sharing.

```
// DHT and pins
#define DHTPIN 2
#define DHTTYPE DHT22
DHT dht(DHTPIN, DHTTYPE);

// LDR and Voltage Divider
#define LDR_PIN 35
#define VOLTAGE_PIN 33
```

Figure 3-5. ESP32 Programming Code Screenshot2

In this part of the code, the sensors used for environmental and electrical measurements are configured. The DHT22 sensor in particular is connected to digital pin 2 on the ESP32. This is specified using "define" statements to clearly indicate which pin is being used and what type of DHT sensor is connected. The sensor is then initialized so it can be used later in the program to collect readings. In a similar manner, the LDR is connected to analog pin 35. Additionally, analog pin 33 is assigned for reading the output from a voltage divider, which is used to estimate the open-circuit voltage of the solar cell. These pin setups are essential because they tell the ESP32 exactly where to look for each piece of data during operation.

```
// MOSFET control pin
#define MOSFET_PIN 14

// SD card
#define SD_CS 5
File dataFile;

// INA219
Adafruit_INA219 ina219;

// RTC
RTC_PCF8563 rtc;
```

Figure 3-6. ESP32 Programming Code Screenshot3

Next, the code sets up the remaining hardware components used in the system. First, the MOSFET control pin is defined and assigned to digital pin 14. This pin 's main function is to switch between measuring open-circuit voltage and short-circuit current from the solar cell. This is done by controlling the state of the MOSFET's. As for the SD card, the module is configured by defining its chip select (CS) pin as pin 5. A file object named dataFile is also declared, which will later be used as a data storage for the collected data and be kept in CSV format in SD card. For measuring current and voltage, the INA219 sensor is initialized. This sensor is crucial for tracking the short-circuit current (I_{SC}) accurately. Lastly, a real-time clock (RTC) module is set up using the PCF8563 chip. The RTC ensures that every data reading is timestamped correctly, which is important for organizing and analyzing the system's performance over time.

```
// Sampling
#define SAMPLING_FREQ_HZ 1
#define SAMPLING_INTERVAL_MS (1000 / SAMPLING_FREQ_HZ)
unsigned long lastSampleTime = 0;

// WiFi & ThingSpeak
const char* ssid = "MoTamim";
const char* password = "limeiscorpion";
const String apiKey = "FK62ZH9C2AYARZI1";
const String serverName = "http://api.thingspeak.com/update";

// Forward declaration
void sendToThingSpeak(float temp, float hum, int light, float voc, float curr);
```

Figure 3-7. ESP32 Programming Code Screenshot4

Here, the system's data acquisition rate as well as the settings for communication with the cloud are set. The sampling frequency is set at 1 Hz, so the system will take a new reading after each second. The first "define" statement gives the frequency and the second calculates the time interval in milliseconds. A variable called lastSampleTime is also made to help retrieve the timing of all recordings during the loop later on in the code.

The following step is to configure Wi-Fi and ThingSpeak for the project. The SSID and password are assigned to character arrays so that the ESP32 is able to connect online. The API key and the server's update address are provided to secure and direct the communication between your ESP32 and ThingSpeak platform. The last step is to put a forward declaration for the sendToThingSpeak function, so it will be able to upload the collected values (temperature, humidity, light, voltage, and current) to the cloud. As a result, the system can work as data is acquired, and users can access it by logging into a web dashboard from any location.

```
46  void setup() {
47    Serial.begin(115200);
48
49    pinMode(MOSFET_PIN, OUTPUT);
50    digitalWrite(MOSFET_PIN, LOW);
51
52    dht.begin();
53    ina219.begin();
54    rtc.begin();
55
56    DateTime now = rtc.now();
57    if (now.year() < 2000) {
58      rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));
59    }
60
61    SD.begin(SD_CS);
62
63    // Connect to WiFi
64    WiFi.begin(ssid, password);
65    Serial.print("Connecting to WiFi");
66    while (WiFi.status() != WL_CONNECTED) {
67      delay(500);
68      Serial.print(".");
69    }
70    Serial.println("\nWiFi connected");
71  }
```

Figure 3-8. ESP32 Programming Code Screenshot5

In the setup() function, all the elements are set up and checked before the system begins normal use. To start debugging and to view information in real time, serial communication must be started at 115200 baud. After that, the MOSFET pin is made into an output and assigned a value of LOW. This allows the system to safely start, so it is able to measure the open-circuit voltage before the first switch.

Afterward, all the sensors and modules get initialized point by point. The date and time are double-checked; if the year is less than 2000, the code updates them to current date and time at the time of compilation.

In this step, the SD card module is initialized too, so the system can begin saving files locally. After that, ESP32 establishes a connection to the Wi-Fi network by using the credentials included in the earlier lines of code. During connecting, the printer shows a series of dots to show the work is underway. When the process is finished, a message appears to confirm the connection. The steps here make certain that the system has prepared all its parts and is set before it initiates the main cycle.

```
73  void loop() {
74    unsigned long currentTime = millis();
75    if (currentTime - lastSampleTime >= SAMPLING_INTERVAL_MS) {
76      lastSampleTime = currentTime;
77
78      DateTime now = rtc.now();
79      char timeStr[20];
80      sprintf(timeStr, "%04d-%02d-%02d %02d:%02d:%02d", now.year(), now.month(), now.day(), now.hour(), now.minute(), now.second());
81
82      float temp = dht.readTemperature();
83      float hum = dht.readHumidity();
84      int ldrVal = analogRead(LDR_PIN);
85
86      // Voltage (Voc)
87      digitalWrite(MOSFET_PIN, LOW);
88      delay(5);
89      float raw = analogRead(VOLTAGE_PIN);
90      float v_adc = raw * (3.3 / 4095.0);
91      float volt = v_adc * (6.05/2.22); // Adjust based on voltage divider
92
93      // Current (Isc)
94      digitalWrite(MOSFET_PIN, HIGH);
95      delay(5);
96      float curr_mA = ina219.getCurrent_mA();
97      digitalWrite(MOSFET_PIN, LOW);
```

Figure 3-9. ESP32 Programming Code Screenshot6

The main purpose of the program is achieved by having the loop() function start and continue running throughout. The first step is to see if it has been long enough, compared to the sampling interval decided earlier, since the last time we took a measurement. If the interval is finished, the system starts to gather fresh data.

The loop starts by receiving both the current date and time from the RTC module using the rtc.now() function and putting them into a DateTime object. After completing the data filtering, the report includes a timestamp in the format "YYYY-MM-DD HH:MM:SS" thanks to the sprintf() function. A memory space containing 20 letters or less is made by declaring "timeStr[20]" to receive this time sting. Because 20 has enough spaces, all the characters of the formatted date and time can be saved correctly. At a later stage, the timestamp will be associated with every set of data for reference and monitoring.

After that (in lines 82 to 84), the DHT22 sensor is used to get the current readings for temperature and humidity. To find out how much sunlight the solar panel is receiving, the light intensity is measured from the LDR connected to analog pin.

In order to measure the open-circuit voltage (V_{OC}) of the solar cell, the MOSFET is set to LOW so that the load is not connected. A brief pause of 5ms makes the reading settle in order to ensure an accurate measurement using the analog input connected to the voltage divider. The raw analog reading is converted into an actual voltage value using a scaling factor derived from the resistors' values in the voltage divider.

Then, the MOSFET is changed to HIGH, which directly shortens the solar cell and lets the system find out the short-circuit current (I_{SC}). The current is sensed with the help of the INA219 sensor. Once the process is done, the MOSFET switches to its initial LOW setting in readiness for the upcoming cycle. With this system, the controlled switching between measurement of voltage and current is safe and convenient because it is entirely automatic without needing any manual intervention. This is simply the summary of lines 86 to 97.

```
98
99      Serial.printf("Time: %s | Temp: %.2f C | Hum: %.2f %% | Light: %d | Voc: %.2f V | Isc: %.2f mA\n",
100                   timeStr, temp, hum, ldrVal, volt, curr_mA);
101
102     dataFile = SD.open("/datalog.csv", FILE_APPEND);
103     if (dataFile) {
104       dataFile.printf("%s,%.2f,%.2f,%d,%.2f,%.2f\n", timeStr, temp, hum, ldrVal, volt, curr_mA);
105       dataFile.close();
106     }
107
108     sendToThingSpeak(temp, hum, ldrVal, volt, curr_mA);
109   }
110 }
```

Figure 3-10. ESP32 Programming Code Screenshot7

Continuing the main loop (starting from line 98), when all the readings are gathered, the system uses a format that shows the timestamp, temperature, humidity, light level, open-circuit voltage, and short-circuit current. It is useful for checking how the program works as tests are being carried out and fixing any errors.

After that, the system saves this data to the SD card. It makes a file named datalog.csv and updates it with all the picked values in the CSV format. As a result, data can be analyzed with Excel or MATLAB in the future. Once the entry is finished, the file is immediately shut to make sure no data is lost.

In the end, the same readings are sent to the ThingSpeak cloud using the sendToThingSpeak() function. Because of this, the solar panel's performance can be checked online from anywhere internet connection is present. As a result, this section guarantees that every platform shares the data in real time and also stores a backup locally.

```
112  void sendToThingSpeak(float temp, float hum, int light, float voc, float curr) {
113    if (WiFi.status() == WL_CONNECTED) {
114      HTTPClient http;
115      String url = serverName + "?api_key=" + apiKey +
116                   "&field1=" + String(temp) +
117                   "&field2=" + String(hum) +
118                   "&field3=" + String(light) +
119                   "&field4=" + String(voc) +
120                   "&field5=" + String(curr);
121
122      http.begin(url);
123      int httpCode = http.GET();
124      if (httpCode > 0) {
125        Serial.println("ThingSpeak Response Code: " + String(httpCode));
126      } else {
127        Serial.println("Error sending to ThingSpeak: " + http.errorToString(httpCode));
128      }
129      http.end();
130    } else {
131      Serial.println("WiFi not connected!");
132    }
133  }
```

Figure 3-11. ESP32 Programming Code Screenshot8

The main aim of the sendToThingSpeak() function is collecting data, which then transfers it to the ThingSpeak platform. The function begins by checking if the ESP32 is attached to Wi-Fi using the code 'if (WiFi.status() == WL_CONNECTED)'. If it exists, the next line is HTTPClient http; which makes an HTTP client to control the network request.

In line 115, the URL is built by adding ServerName with other parts "String url = serverName+ etc.". The value of the API key and sensor readings of temperature, humidity, light, open-circuit, and short-circuit current are passed to the ThingSpeak update URL as parameters in this line.

When the whole URL is set up, the line http.begin(url); launches the HTTP request, while "int httpCode = http.GET()" sends it to the server. If everything goes as intended and the response code is higher than 0, the system uses Serial.println to show the response code in the serial monitor. Should there be an error, it gives the matching error message using "Serial.println("Error sending to ThingSpeak: " + http.errorToString(httpCode));".

The last step, calling http.end();, ends the connection to free memory and other resources. If at first the Wi-Fi check fails, the ESP32 is not connected and the else block runs, displaying the message "WiFi not connected!" the result is printed out in the serial monitor. This procedure ensures the sensor's collected information is stored in the cloud every so often, so people can supervise everything online whenever they want.

The figure below shows the workflow diagram of this system according to the programmed code on the ESP32 microcontroller that has been discussed.

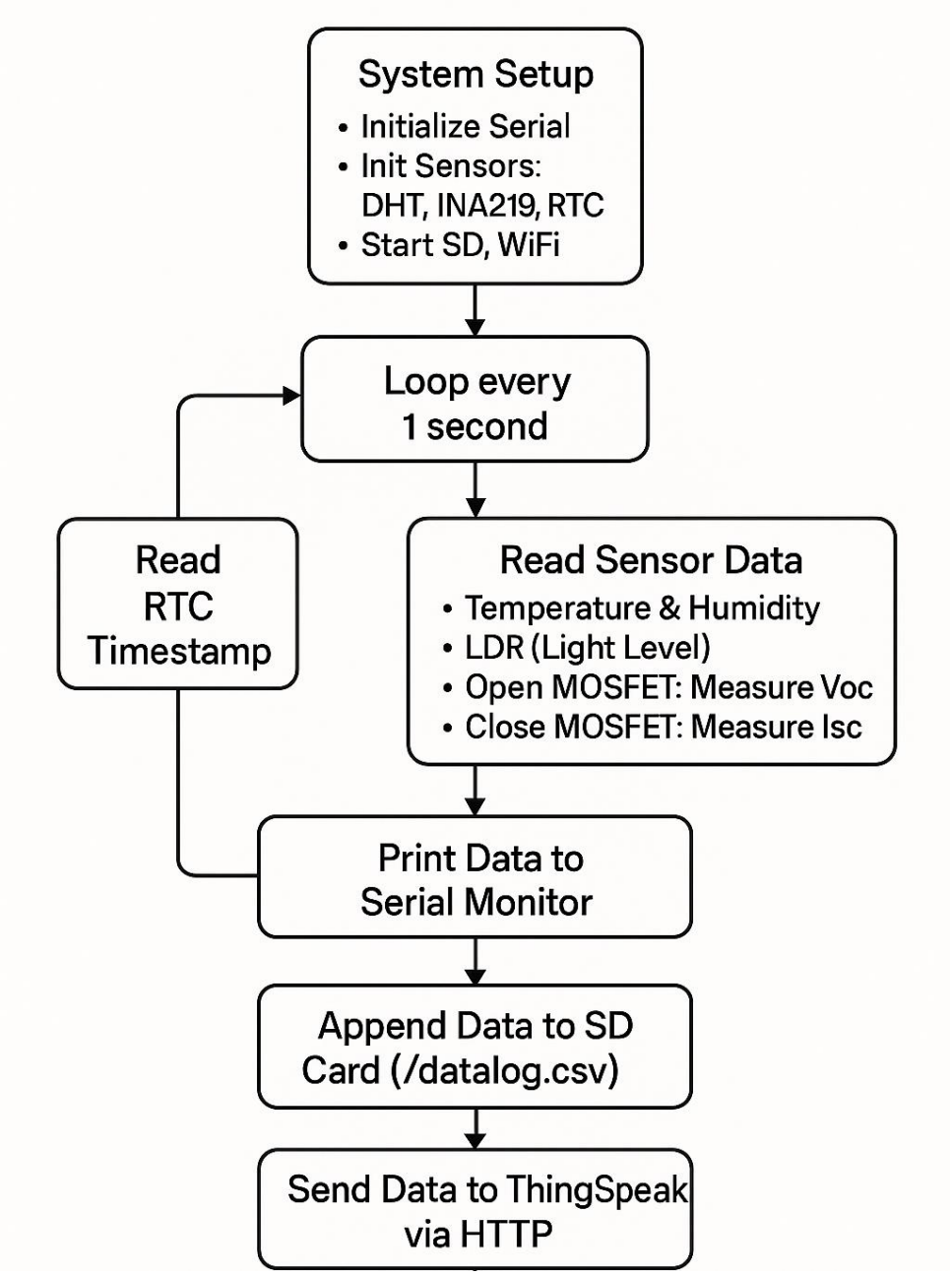

Figure 3-12. Workflow Diagram of Implemented System

3.3. Machine Learning Models

3.3.1. Random Forest

The Random Forest Classifier combines various decision trees to both increase accuracy and minimize the chance of overfitting. Every tree in the forest is made on a random sample of the data and features, which strengthens the model and helps it ignore random data. To make predictions, the classifier combines the results of the trees by voting them and gets better and more stable results. It works well with complicated datasets that contain a lot of information and also have missing values. Also, tree-based models can measure which variables have the strongest influence on the dataset's outcome. Many researchers use Random Forests in bioinformatics, finance, and remote sensing thanks to their adaptability and good results [63].

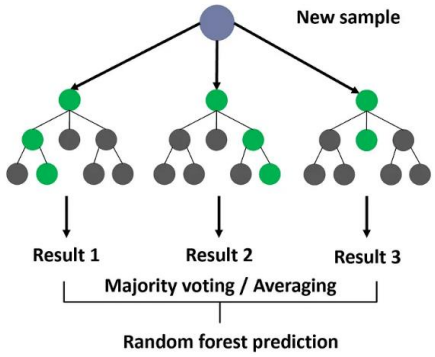

Figure 3-13. Random Forest Classifier [64]

Each random forest combines many decision trees, which look like trees. chains of decision points set in order to teach a computer to guess the desired age and color are two features that are used as inputs and then form the output value. price). At every node, decision trees take the input features to generate criteria. To create a binary logic model, the computer decides which of the trees is likely to be generated when solving the problem by sorting training data so it matches the aim of the machine learning process variable. A random forest is made by training numerous decision trees separately. find out the accuracy of the results on a random mini-group of data from the set.

3.3.1.1. Random Forest Training Code

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
import joblib
```

Figure 3-14. Random Forest Screenshot1

Here, all the necessary libraries are imported to help produce, review, and keep a model for air pollution classification. Working with the `pandas` library, data can be processed, Excel files can be read, and generally any information in tables can be handled.

Both `matplotlib.pyplot` and `seaborn` are used to make the confusion matrix and other plots to understand what the model has learned. Some key modules are loaded from `sklearn` for this job: `train_test_split` for organizing the dataset, the `RandomForestClassifier` algorithm, and the `classification_report` tool to overview the predictions and the `confusion_matrix` to further analyze them. Finally, the `joblib` module is imported to save the trained model in a `.pkl` file format, to make it easy to reload and use later without retraining. Together, these libraries form the foundation of the machine learning pipeline used in this script.All of these libraries serve as the base for the machine learning process used in this script.

```
# 1. Read the Excel file (modify according to your file name)
df = pd.read_excel("day2_labeled.xlsx")  # Change "data.xlsx" to your file path

# 2. Drop the 'Date_Time' column if it exists
if "Date_Time" in df.columns:
    df.drop(columns=["Date_Time"], inplace=True)
    print("🗑️ Dropped column: 'Date_Time'")
else:
    print("ℹ️ Column 'Date_Time' not found — nothing dropped.")



# 3. Convert String labels to Numerical Values
df["Label"] = df["Label"].map({"Night": 0, "Cloudy": 1, "Sunny": 2})

# 4. Split features and labels
X = df.drop(columns=["Label"])
y = df["Label"]
```

Figure 3-15. Random Forest Screenshot2

At this stage, the code takes care of preparing the dataset so it can be used in the machine learning training process. The process starts by reading the Excel file marked“day2_labeled.xlsx” with `pandas`. Then, the data is loaded as a DataFrame for additional steps. Here, the code checks whether the column named `"Date_Time"` is present or not after loading the data. If so, it is possible to remove the column with `df.drop(...)`, because its not necessary for the classification problem at all. Some kind of alert is given to show whether the column was taken or was not present.

After that, the program changes the class labels from “Night”, “Cloudy”, and “Sunny” into numerical values to be understood by the model. This happens by using the `.map()` function to give 0 to “Night”, 1 to “Cloudy”, and 2 to “Sunny” data.

After encoding the labels is complete, all columns in the data are broken up: those except the `"Label"` column go into `X` (as input features), while the `"Label"` column is kept as `y` to represent the actual values. Setting this up is necessary for proceeding with the classification training.

```
# 5. Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split( *arrays: X, y, test_size=0.2, random_state=42)
print("🔁 Data split into training and testing sets")

# 6. Train the SVM model
print("⚙️ Training Random Forest model...")
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
print("✅ Random Forest model trained successfully")

# 7. Evaluate the model
y_pred = model.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
report = classification_report(y_test, y_pred)
print("📊 Classification Report:\n", report)
```

Figure 3-16. Random Forest Screenshot3

This section is about preparing and using a machine learning model for evaluation. First, data gets divided into training and testing sets with `train_test_split()`, so 80% is used to train and the rest (20%) for testing the model. Setting `random_state` to 42 allows the code to create the same split every time it is run.

Next, 100 decision trees are used and the same parameter for reproducibility, `random_state`, is applied in the Random Forest classifier initialization. After that, the model is taught by passing `X_train` and `y_train` into the `.fit()` method. Once the training ends, a notification is printed to indicate that everything worked as expected.

Testing is done with the model by using the test data. Predictions are obtained by running `X_test` through the `.predict()` method, and results are then compared with the actual labels found in `y_test`. A confusion matrix is also created by running `confusion_matrix()` to see how correctly the model can tell the classes apart. To gather more details, a specific classification report is generated through `classification_report()`, which has precision, recall, and F1-score for every class. A summary of the model's performance is visualized.

```
# 8. Plot confusion matrix
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["Night", "Cloudy", "Sunny"], yticklabels=["Night", "Cloudy", "Sunny"])
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Confusion Matrix - Random Forest")
plt.tight_layout()
plt.savefig("confusion_matrix_random.png")
plt.show()
print("📌 Confusion matrix saved to confusion_matrix_random.png")
```

Figure 3-17. Random Forest Screenshot4

In the final of the code, the confusion matrix is displayed by using `matplotlib` and `seaborn`. A figure shaped as 6 by 4 inches is made and uses heatmap() from seaborn to make the matrix. Each label in the heatmap represents how many times classification was correct or how many it was wrong for the labels “Night”, “Cloudy”, and “Sunny”. A title is added and labels are put on the x-axis and y-axis to explain the difference between predicted and actual values. To make the labels easily visible, `tight_layout()` is applied, and the matrix is saved as “confusion_matrix_random.png”, and `plt.show()` shows it.

Following that, the saved Random Forest model is put into a file called `"modelrandom.pkl"` by using `joblib.dump()`. It enables the model to predict things again later without having to be retrained. At the end, the report obtained through classifier is saved to a text file known as `"classification_report_random.txt"`.

```
# 9. Save the model for later use
joblib.dump(model, filename: 'modelrandom.pkl')
print("💾 Random Forest model saved to .pkl file")

# 10. Save classification report
with open("classification_report_random.txt", "w") as f:
    f.write(report)
print("📄 Classification report saved to classification_report_random.txt")
```

Figure 3-18. Random Forest Screenshot5

3.3.1.2. Random Forest Testing Code

```python
import pandas as pd
import joblib

# 1. Load the saved model
model = joblib.load('modelrandom.pkl')

# 2. Read a new Excel file (change the file name here)
df_new = pd.read_excel('day3_labeled.xlsx')

if "Date_Time" in df_new.columns:
    df_new.drop(columns=["Date_Time"], inplace=True)
    print("🗑️ Dropped column: 'Date_Time'")
else:
    print("ℹ️ Column 'Date_Time' not found — nothing dropped.")

# 3. Keep the original label column for later comparison
if "Label" in df_new.columns:
    actual_labels = df_new["Label"]
    df_features = df_new.drop(columns=["Label"])
else:
    actual_labels = None
    df_features = df_new.copy()
```

Figure 3-19. Random Forest Test Screenshot1

In the testing code phase, it needs to import the `pandas` library for working with data and the `joblib` library to load the saved Random Forest model (named `modelrandom.pkl`). `joblib.load()` is then used to get the chosen model from disk and put it into memory. Following this, the script reads the latest Excel file named `"day3_labeled.xlsx"` which has the data to be used. If the data features a `Date_Time` column, it is deleted as it is not necessary for prediction.

Accuracy evaluation will be made possible as the code makes sure the already existing `"Label"` column which includes the actual classes exists. If the condition holds, the actual label is put in `actual_labels`, and the rest of the data is given as an input for prediction. When the Label column is not found, the script assumes the file has no label and goes on as planned.

```
# 4. Predict the classes using the model
predicted_labels = model.predict(df_features)
df_new["predicted_label"] = predicted_labels

# 5. Map numeric predictions to readable labels
label_map = {0: "Night", 1: "Cloudy", 2: "Sunny"}
df_new["predicted_label_name"] = df_new["predicted_label"].map(label_map)

# 6. Save the results to an Excel file
output_file = "predicted_results_randomforest.xlsx"
df_new.to_excel(output_file, index=False)
print(f"📄 Predictions saved to: {output_file}")
```

Figure 3-20. Random Forest Test Screenshot2

After that, prediction of labels for the input data is made using `model.predict()`. In a new column called `"predicted_label"` the predictions are added to the DataFrame. To make the outcomes easier to understand, the dictionary is used to convert the numeric values into original class names such as “Night”, “Cloudy”, and “Sunny”.

At last, the DataFrame with the prediction outcomes and the original data is saved to an Excel file named `"predicted_results_randomforest.xlsx"`. A print statement proves that predictions are saved, marking the end of the inference phase in machine learning pipeline.

```
# 7. Compare predictions with actual labels
if actual_labels is not None:
    # Convert actual labels to numeric for comparison
    actual_labels_mapped = actual_labels.map({"Night": 0, "Cloudy": 1, "Sunny": 2})

    # Compute accuracy
    accuracy = accuracy_score(actual_labels_mapped, df_new["predicted_label"])
    print(f"📈 Prediction Accuracy on Test File: {accuracy * 100:.2f}%")
else:
    print("⚠️ No actual labels found for comparison.")
```

Figure 3-21. Random Forest Test Screenshot3

The segment of the code examines if the predictions made by the model match the true labels, if any are provided. The `accuracy_score` function from `sklearn.metrics` is required to find out how many predictions had correct answers. Afterwards, the script checks if `actual_labels` is not missing, to show that the input dataset has true labels. If needed, the string labels change into numbers using the map() method so that they can align with the predictions from the model. After the data is transformed into numbers (0, 1 or 2), the `accuracy_score` function helps you find out the percentage of the correct predictions. The information is then sent to the console in a well-formatted way. In the absence of labels, the script issues a notification to the user that the chance to check the accuracy is not available. As a result, the script knows when to switch between testing and just predicting, based on what is set in the input file.

3.3.2. Support Vector Machine

The Support Vector Machine (SVM) is a widely recognized supervised machine learning algorithm utilized for tasks involving both regression and classification [65]. This technique entails the formulation of a line in a two-dimensional context or a plane in a three-dimensional context to delineate the optimal boundary, commonly referred to as a hyperplane. Data points associated with one category are positioned on one side of this hyperplane, while those belonging to the alternative category are found on the opposite side. The data points that are closest to the hyperplane are identified as Support Vectors. The fundamental aim of SVM is to maximize the margin, which is defined as the separation between the hyperplane and the support vectors, thereby facilitating the identification of the ideal hyperplane.

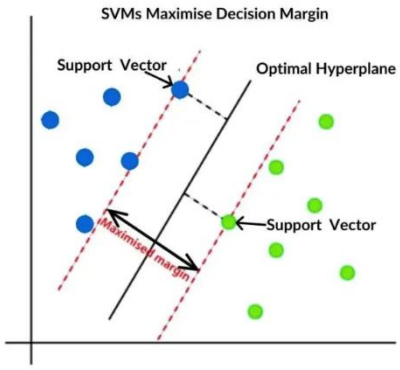

Figure 3-22. Example of SVM Hyperplane [66]

An SVM AI model will be used to predict if the weather is Sunny, Cloudy, or Night, depending on the values of Temperature, Light Intensity, and Humidity recorded by the sensors. Two python codes have been written. The initial code is designed to train the Support Vector Machine (SVM) model using labeled data, whereas the second code evaluates the trained model by generating predictions based on a specified dataset.

3.3.2.1. SVM Training Code

The first step is to import all of the important libraries that will be used. This is done using the following block of code:

```
import pandas as pd
import joblib
import matplotlib.pyplot as plt
import seaborn as sns
import os
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.exceptions import ConvergenceWarning
import warnings

```

Figure 3-23. SVM Screenshot1

The first library being imported, *pandas,* is essential for data manipulation and analysis. In this code we use it to load the excel file (.xlsx) which contains the dataset.

The second library, *joblib*, is used for saving and loading the trained model and scaler objects. This allows us to extract the model and scaler .pkl files which will be used in the testing code.

The third library, *matplotlib.pyplot*, is a plotting library that is used to visualize results. In this code, it will be used to visualize the confusion matrix. The fourth library *seaborn* is actually used in conjunction with matplotlib to generate a more visually appealing confusion matrix.

The fifth library *os* simply provides utilities to interact with the operating system. In this case, it simply checks if the file we're attempting to load exists, to prevent errors.

All of the functions beginning with sklearn are under the library called *scikit-learn*, which is a powerful machine learning toolkit in Python.

The first function (line 6) is employed to divide the dataset into train-test partitions.

The second function (line 7) helps many machine learning algorithms, including SVM, to perform better, by making all features on the same scale. This is done by subtracting the mean and adjusting to have a variance of one.

The third function (line 8)*, is the Support Vector Classifier*. This is basically a tool that helps train the SVM model for classifying the weather conditions based on sensor data.

The fourth function (line 9) imports the tools needed to assess the effectiveness of the trained model. These evaluation tools include the confusion matrix and the classification report, which provides key performance metrics such as precision, recall, and F1-score.

The fifth and final function (line 10) , helps in managing and suppressing convergence warnings that may arise during model training, therefore keeping the output clean and focusing only on important messages.

Finally, *import warnings* is used to manage warning messages in Python. It helps hide or control non-critical alerts, keeping the output clean. In this case, as seen in the code shown below, it ignores SVM convergence warning that might occur during model training.

```
13  # Ignore SVM convergence warnings
14  warnings.filterwarnings( action: "ignore", category=ConvergenceWarning)
15
```

Figure 3-24. SVM Screenshot2

After setting up the necessary libraries, the initial step is to insert the dataset. The code for this is shown below:

```
# 1. Get Excel filename from user
filename = input("📥 Enter the name of the Excel file (e.g., data.xlsx): ").strip()

# 2. Check if file exists
if not os.path.exists(filename):
    print(f"❌ The file '{filename}' was not found!")
    exit()

# 3. Load the dataset
df = pd.read_excel(filename)
print(f"✅ File loaded: {filename}")

```

Figure 3-25. SVM Screenshot3

The first line prompts the user to enter the name of the excel file that contains the dataset. For example, *data.xlsx*. strip() is used to remove any leading or trailing spaces in the filename to avoid errors. The following block of code examines if the file actually exists, and if it doesn't, the code will print an error and stop the program using exit() to prevent crashes from happening. Finally, the third block will load the excel file using *pandas* library (pd.read) for further processing, and if successful will also print a message.

The next step is to prepare the dataset for the training/testing process. The first step in this process will be to remove a unnecessary column in the dataset.

```
# 4. Drop the 'Date_Time' column if it exists
if "Date_Time" in df.columns:
    df.drop(columns=["Date_Time"], inplace=True)
    print("🧹 Dropped column: 'Date_Time'")
else:
    print("ℹ️ Column 'Date_Time' not found — nothing dropped.")

```

Figure 3-26. SVM Screenshot4

In this case, the "Date_Time" column is not needed for the training/testing process. Since the goal is to classify weather conditions based on sensor readings, this column is not directly relevant. First, the column will be checked for using an if condition. After that, the DataFrame is directly modified by removing the column using *df.drop*. A message is also printed to confirm whether the column was dropped or wasn't found.

The second step in preparing the dataset for training/testing is to change any input strings to numerical inputs.

```
35  # 5. Convert String labels to Numerical Values
36  df["Label"] = df["Label"].map({"Night": 0, "Cloudy": 1, "Sunny": 2})
37  df['Temperature'] = df['Temperature'].fillna(df['Temperature'].mean())
38  df['Humidity'] = df['Humidity'].fillna(df['Humidity'].mean())
39  print(df.isnull().sum())
40
```

Figure 3-27. SVM Screenshot5

In this dataset, it contains a column labeled "Label" that specifies whether the entire row of inputs results in the condition being Night, Cloudy, or Sunny. Since machine learning models require numerical input and not strings, we must convert them to numbers. Using df["Label"] the column is accessed, and .map({}) transforms each string in the column to the value provided. In this case, Night is given 0, Cloudy is given 1, and Sunny is given 2. Utilizing the fillna() function, one can impute NaN (null) entries in the Temperature and Humidity by replacing them with the corresponding means of their respective columns. n. This is done as SVM models cannot work with NaN entries. df.isnull().sum() is employed to verify that all absent values have been filled.

The third step would be to separate the dataset into features and label.

```
41  # 6. Split features and labels
42  X = df.drop(columns=["Label"])
43  y = df["Label"]
44
```

Figure 3-28. SVM Screenshot6

X are the features, which are the input data used to predict. y on the other hand is the label, which is the target/output we're trying to predict. X will be a new dataframe containing all columns except "Label", which are all the sensor readings. As for y, the "Label" column will be stored there, which contains all the output classes.

The fourth and final step before starting the training/testing process is to normalize (or standardize) the input data.

```
45    # 7. Normalize the data
46    scaler = StandardScaler()
47    X_scaled = scaler.fit_transform(X)
48
```

Figure 3-29. SVM Screenshot7

The first line, scaler = StandardScaler(), standardizes to have a mean of zero and a standard deviation of one. The second line evaluates both for each feature using scaler.fit, and transforming the data using "_transform(X)" to give normalize each feature.

Given that many machine learning models, such as SVM, are sensitive to the size of input features, this is a crucial step. For example, If light values are in the thousands, and temperature is in the 20s, the model might give more importance to light just because it's on a bigger scale, even if it's not actually more important. Standardizing puts all features on the same scale, so each one is treated fairly.

Now, the dataset must be split into training and test portions, which is achieved through the train_test_split utility.

```
49    # 8. Split into training and testing sets
50    X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
51    print("🔗 Data split into training and testing sets")
52
```

Figure 3-30. SVM Screenshot8

Accordingly, the data has been segmented so that 20% is used for testing, and 80% is used for training the model. The parameters names are written above. This is done by setting the test size to 0.2. For consistency during development and debugging, random state is given a value of 42, which guarantees that the split is reproducible each time the code executes.

The next section will train the SVM using preprocessed data.

```
53    # 9. Train the SVM model
54    print("⚙️ Training SVM model...")
55    model = SVC(kernel='rbf', probability=True, random_state=42)
56    model.fit(X_train, y_train)
57    print("✅ SVM model trained successfully")
58
```

Figure 3-31. SVM Screenshot9

The model is created using the SVC class, with the kernel parameter set to rbf (radial basis function), which helps in handling non-linear patterns in the dataset. The probability=True option enables the model to compute class probabilities. The random_state=42 ensures consistent results each time the code is executed by fixing the randomness in the model's internal operations. After defining the model, the model. fit(X_train_small, y_train_small) function is called to train it using the scaled training data and their corresponding labels, allowing the SVM to learn how to distinguish between different weather conditions based on input features.

The next step will be to evaluate performance using unseen data.

```
59  # 10. Evaluate the model
60  y_pred = model.predict(X_test)
61  cm = confusion_matrix(y_test, y_pred)
62  report = classification_report(y_test, y_pred)
63  print("📊 Classification Report:\n", report)
64
```

Figure 3-32. SVM Screenshot10

The assessment of performance is conducted by initially generating predictions on the test set utilizing the trained model using model.predict function. To evaluate the model's performance, the predicted outputs (y_pred) are compared against the true labels (y_test). A Confusion Matrix is generated to show how many predictions the model got correct and where it made mistakes across different classes. A Classification Report is also printed, which includes important metrics. These evaluation findings aid in assessing the model's efficiency and pinpointing any potential areas for development.

The next block of code is used to plot the Confusion Matrix.

```
65  # 11. Plot confusion matrix
66  plt.figure(figsize=(6, 4))
67  sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["Night", "Cloudy", "Sunny"], yticklabels=["Night", "Cloudy", "Sunny"])
68  plt.xlabel("Predicted")
69  plt.ylabel("True")
70  plt.title("Confusion Matrix - SVM")
71  plt.tight_layout()
72  plt.savefig("confusion_matrix_svm.png")
73  plt.show()
74  print("📌 Confusion matrix saved to confusion_matrix_svm.png")
75
```

Figure 3-33. SVM Screenshot11

The code generates a heatmap representation of the confusion matrix, which depicts the amount of accurate and inaccurate predictions for each class, utilizing the Matplotlib and Seaborn libraries. The procedure starts with the command plt.figure(figsize=(6, 4)), which establishes a figure with designated dimensions of 6 inches in width and 4 inches in height. The actual heatmap is generated using sns.heatmap line. This line draws the heatmap using the confusion matrix (cm) calculated earlier, adds numerical annotations inside each cell (annot=True), formats them as integers (fmt="d"), and applies a blue color scale (cmap="Blues").

The xticklabels and yticklabels explicitly set the labels for the predicted and actual classes, respectively, making the matrix easier to interpret.

Next, plt is used to label the X and Y axis, with ("Predicted") and ("True") respectively. plt.title("Confusion Matrix - SVM") adds the title. The plt.tight_layout() code is written to simply modify the plot's spacing to prevent overlapping. Then, plt.savefig("confusion_matrix_svm.png") makes the plot be saved as an image file. Finally, plt.show() shows the plot on the screen. Print is once again used as a confirmation that the Confusion Matrix was successfully made and saved.

The final step for this code is to save the model and scaler, which will be used in the testing code, and also save the classification report.

```
76  # 12. Save model and scaler
77  joblib.dump(model, filename: "svm_model.pkl")
78  joblib.dump(scaler, filename: "scaler.pkl")
79  print("💾 SVM model and scaler saved to .pkl files")
80
81  # 13. Save classification report
82  with open("classification_report_svm.txt", "w") as f:
83      f.write(report)
84  print("📄 Classification report saved to classification_report_svm.txt")
85
86
```

Figure 3-34. SVM Screenshot12

The saving process is done using the joblib library. The line joblib.dump(model, "svm_model.pkl") serializes and subsequently stores the trained Support Vector Machine model in a file designated as svm_model.pkl. while joblib.dump(scaler, "scaler.pkl") does the same for the scaler used to normalize the data. Additionally, the classification performance summary generated earlier is saved to a text file. This is done using a with open(...) block, which opens a file named classification_report_svm.txt in write mode and stores the classification report inside. This approach ensures that the evaluation metrics (precision, recall, f1-score, and support) are archived in a text file.

3.3.2.2. SVM Testing Code

Using the previous trained SVM model, this code will evaluate a new, unseen dataset. The first step once again is to import the libraries that will be used. In this case, there are only three. "pandas" will be used to read the excel dataset file, "joblib" will load the trained scaler and model .pkl files, and "os" simply prevents errors by checking if the new dataset actually exists.

```
1  import pandas as pd
2  import joblib
3  import os
4
```

Figure 3-35. SVM Test Screenshot1

The next step, exactly similar to the previous code, is to input the new dataset.

```
5  # 1. Enter the name of the new test Excel file
6  filename = input("📥 Enter the name of the new Excel file to test (e.g., new_data.xlsx): ").strip()
7
8  # 2. Check if the file exists
9  if not os.path.exists(filename):
10     print(f"❌ The file '{filename}' was not found!")
11     exit()
12
13 # 3. Load the new data
14 df_new = pd.read_excel(filename)
15 print(f"✅ File loaded: {filename}")
16
```

Figure 3-36. SVM Test Screenshot2

The "Date_Time" column will be dropped again as it is not at all needed.

```
17 #   Drop the 'Date_Time' column if it exists
18 if "Date_Time" in df_new.columns:
19     df_new.drop(columns=["Date_Time"], inplace=True)
20     print("🗑️ Dropped column: 'Date_Time'")
21 else:
22     print("ℹ️ Column 'Date_Time' not found — nothing dropped.")
23
```

Figure 3-37. SVM Test Screenshot3

Once again, the code all handles all missing values in "Temperature" and "Humidity".

```
#  Convert String labels to Numerical Values
#df_new["Label"] = df_new["Label"].map({"Night": 0, "Cloudy": 1, "Sunny": 2})
df_new['Temperature'] = df_new['Temperature'].fillna(df_new['Temperature'].mean())
df_new['Humidity'] = df_new['Humidity'].fillna(df_new['Humidity'].mean())
print(df_new.isnull().sum())

```

Figure 3-38. SVM Test Screenshot4

The next block of code is responsible for preserving the actual weather conditions for later comparison, while ensuring that the model does not use them during prediction.

```
# 4. Keep the original label column for later comparison
if "Label" in df_new.columns:
    actual_labels = df_new["Label"]
    df_features = df_new.drop(columns=["Label"])
else:
    actual_labels = None
    df_features = df_new.copy()

```

Figure 3-39. SVM Test Screenshot5

In this step, the code checks if the Excel file contains a column named "Label" which holds the actual weather conditions. If it does, the values from this column are stored in a variable called actual_labels for later reference. At the same time, this "Label" column is removed from the feature set using df_new.drop(columns=["Label"]) to ensure that the model does not use the known outputs when making predictions. If the label column is not found, it simply copies the entire dataset as the feature set.

After that, the previously trained SVM model and scaler are to be imported.

```
# 5. Load the trained SVM model and scaler
model = joblib.load("svm_model.pkl")
scaler = joblib.load("scaler.pkl")

```

Figure 3-40. SVM Test Screenshot6

The saved SVM model is first retrieved using the first line, while the scaler file that was used to normalise the training data in the earlier code is also loaded by the second line. This guarantees that the transformation of the new test data is identical to that of the training data.

In the next line, scaler.transform(df_new) is applied to scale the new dataset.

```
42  # 6. Apply the same scaler used in training
43  X_new_scaled = scaler.transform(df_features)
44
```

Figure 3-41. SVM Test Screenshot7

The new dataset is ready for the SVM model to start generating predictions.

```
45  # 7. Make predictions
46  predictions = model.predict(X_new_scaled)
47  df_new["predicted_label"] = predictions
48
```

Figure 3-42. SVM Test Screenshot8

The line predictions = model.predict(X_new_scaled) takes the new dataset (after scaling) and allows the model to start generating predictions. These predictions are numerical values (e.g., 0 for Night, 1 for Cloudy, 2 for Sunny). The next line, df_new["predicted_label"] = predictions, adds a new column to the DataFrame called "predicted_label", which stores the predicted results alongside the original input data. This makes it easier to review and analyze which weather condition each test sample has been classified as.

In order to make the new excel file containing predictions much easier to read, the numerical values for predictions will be transformed into strings.

```
49  # 8. Map numeric predictions to readable labels
50  label_map = {0: "Night", 1: "Cloudy", 2: "Sunny"}
51  df_new["predicted_label_name"] = df_new["predicted_label"].map(label_map)
52
```

Figure 3-43. SVM Test Screenshot9

The dictionary label_map = {0: "Night", 1: "Cloudy", 2: "Sunny"} defines a mapping between the numerical class values and their corresponding weather condition names. Then, the second line creates a new column in the DataFrame called "predicted_label_name", where each numeric prediction is replaced with its corresponding label.

After that, the next step is to save the new excel file containing the predicted results.

```
53  # 9. Save the results to an Excel file
54  output_file = "predicted_results_svm.xlsx"
55  df_new.to_excel(output_file, index=False)
56  print(f"📄 Predictions saved to: {output_file}")
57
```

Figure 3-44. SVM Test Screenshot10

First, the variable output_file is set to the name "predicted_results_svm.xlsx", which will be the filename for the output. Then, df_new.to_excel(output_file, index=False) writes the entire updated DataFrame, including the original data along with the predicted labels, to this Excel file. The parameter index=False ensures that the row indices are not saved as an extra column in the Excel sheet. Lastly, a confirmation message is printed to inform the user that the predictions have been successfully saved to the specified file.

The final part of the code is a comparison between the original labels and the new predicted labels.

```
60  # 10. Compare predictions with actual labels
61  if actual_labels is not None:
62      # Convert actual labels to numeric for comparison
63      actual_labels_mapped = actual_labels.map({"Night": 0, "Cloudy": 1, "Sunny": 2})
64      df_new["actual_label_mapped"] = actual_labels_mapped
65
66      # Compute accuracy
67      accuracy = accuracy_score(actual_labels_mapped, df_new["predicted_label"])
68      print(f"📈 Prediction Accuracy on Test File: {accuracy * 100:.2f}%")
69  else:
70      print("⚠️ No actual labels found for comparison.")
71
```

Figure 3-45. SVM Test Screenshot11

This block of code compares the predicted results with the existent labels from the Excel file (if accessible, which is verified with the “if actual_labels is not None” line) to assess how well the SVM model operates on a fresh dataset. The format of the excel file is then matched to the model’s predictions by first mapping the "Label" column to numerical values. The script then determined the number of predictions that matched the real labels using scikit-learn's accuracy_score function. The outcome is viewed as a percentage.

4. Chapter (4): Results and Discussion

4.1. Collected Datasets

Data in this project was collected via the monitoring system composed of an ESP32 circuit prototype. During this time the system was operated approximately 16 hours daily, starting at or near sunrise and running till nightfall. The data gathered was in the form of temperature, humidity, light intensity, open-circuit voltage, and short-circuit current measured with the aid of sensors. To keep time properly a real time clock module was employed to time stamp each reading.

Day	No. of Collected Data
1	53,695
2	60,286
3	58,673
4	57,516
5	58,108
6	57,678
7	61,295
8	54,548

Table 4-1. No. of Collected Data Per Day

All the data was saved in two parallel forms: locally, on a microSD card, and in the cloud on the IoT platform ThingSpeak. This was a safe backup stored locally and enabled in-depth analysis of system performance offline, and ThingSpeak offered real-time access to the system performance anywhere and visualized it remotely. The system was averagely recording about 57,700 data entries daily, resulting in a big and rich dataset that'll be used to train machine learning models.

	Date_Time	Temperature	Humidity	Light	Voltage	Current	Label
2	2025-05-27 08:56:20	31.7	40.3	25	6.43	52.7	Sunny
3	2025-05-27 08:56:21	31.7	40.3	18	6.43	52.7	Sunny
4	2025-05-27 08:56:24	31.8	40	22	6.43	52.1	Sunny
5	2025-05-27 08:56:25	31.8	40	25	6.42	51.2	Sunny
6	2025-05-27 08:56:26	31.8	40	26	6.42	52.1	Sunny
7	2025-05-27 08:56:27	31.8	40	25	6.42	51	Sunny
8	2025-05-27 08:56:28	31.9	39.9	27	6.4	50.7	Sunny
9	2025-05-27 08:56:29	31.9	39.9	23	6.4	50.8	Sunny
10	2025-05-27 08:56:30	31.9	39.8	25	6.41	50.7	Sunny
11	2025-05-27 08:56:31	31.9	39.8	25	6.41	50.5	Sunny
12	2025-05-27 08:56:32	31.9	39.7	27	6.4	50.7	Sunny
13	2025-05-27 08:56:33	31.9	39.7	23	6.41	50.5	Sunny
14	2025-05-27 08:56:34	32	39.5	26	6.41	50.4	Sunny
15	2025-05-27 08:56:35	32	39.5	26	6.41	50.7	Sunny
16	2025-05-27 08:56:36	32	39.5	28	6.41	50.5	Sunny
17	2025-05-27 08:56:37	32	39.5	30	6.4	50.7	Sunny
18	2025-05-27 08:56:38	32	39.3	27	6.4	50.6	Sunny
19	2025-05-27 08:56:39	32	39.3	26	6.4	50.6	Sunny
20	2025-05-27 08:56:40	32.1	39.3	29	6.4	50.6	Sunny
21	2025-05-27 08:56:41	32.1	39.3	28	6.4	50.7	Sunny
22	2025-05-27 08:56:42	32.1	39.3	26	6.4	50.4	Sunny
23	2025-05-27 08:56:43	32.1	39.3	27	6.4	50.5	Sunny
24	2025-05-27 08:56:44	32.1	39.3	25	6.4	50.8	Sunny
25	2025-05-27 08:56:45	32.1	39.3	29	6.39	50.8	Sunny

Figure 4-1. Day1 Dataset (Sunny)

Such a two-storage solution provided not only the security of the data but also the comfort of the user who could track the behavior of solar panels and the environmental parameters in real-time wherever he or she was. An important note is that the LDR reading (columns 4 from left to right) readings are low during sunny days (even reaching values of 0) and it's very high (3000 to 4095) during night or when very low light intensity is present. During sunny times, much light reaches the LDR and therefore its resistance is very low. In the classic voltage divider arrangement this would have the effect of dropping the majority of the voltage across the fixed resistor in the circuit and the analog pin connected to the LDR will read a low voltage, thus giving low analog values. On the contrary, during nighttime, the amount of light is low or absent altogether hence the resistance of LDR is very high. This reverses the voltage drop in the divider and the analog pin is supplied with higher voltage, which results in high analog values.

4	2025-05-27 19:31:18	27.9	52.8	785	2.82	0.1	Cloudy
5	2025-05-27 19:31:19	28	52.8	787	2.81	0.3	Cloudy
6	2025-05-27 19:31:20	28	52.8	790	2.82	0	Cloudy
7	2025-05-27 19:31:21	27.9	52.8	787	2.82	-0.1	Cloudy
8	2025-05-27 19:31:22	27.9	52.8	787	2.81	-0.1	Cloudy
9	2025-05-27 19:31:23	27.9	52.8	788	2.81	0	Cloudy
0	2025-05-27 19:31:24	27.9	52.8	790	2.81	0.1	Cloudy
1	2025-05-27 19:31:25	27.9	52.8	787	2.81	0.1	Cloudy
2	2025-05-27 19:31:26	27.9	52.8	790	2.81	0.1	Cloudy
3	2025-05-27 19:31:27	27.9	52.8	787	2.81	0.1	Cloudy
4	2025-05-27 19:31:28	27.9	52.8	793	2.81	0	Cloudy
5	2025-05-27 19:31:29	27.9	52.8	791	2.81	0.1	Cloudy
6	2025-05-27 19:31:30	27.9	52.8	790	2.81	0.2	Cloudy
7	2025-05-27 19:31:31	27.9	52.8	791	2.81	0.1	Cloudy
8	2025-05-27 19:31:32	27.9	52.8	791	2.8	0	Cloudy
9	2025-05-27 19:31:33	27.9	52.8	790	2.81	0	Cloudy
0	2025-05-27 19:31:34	27.9	52.8	794	2.81	0	Cloudy
1	2025-05-27 19:31:35	27.9	52.8	794	2.81	0	Cloudy
2	2025-05-27 19:31:36	27.9	52.8	791	2.8	0.3	Cloudy
3	2025-05-27 19:31:37	27.9	52.9	794	2.8	0	Cloudy
4	2025-05-27 19:31:38	27.9	52.9	794	2.8	0.2	Cloudy
5	2025-05-27 19:31:39	27.9	52.8	793	2.8	0.2	Cloudy
6	2025-05-27 19:31:40	27.9	52.8	797	2.8	0	Cloudy

Figure 4-2. Day1 Dataset (Cloudy)

2025-05-27 21:45:10	27.3	52.9	3947	0	-0.2	Night
2025-05-27 21:45:11	27.3	52.9	3952	0	-0.3	Night
2025-05-27 21:45:12	27.3	52.9	3951	0	-0.5	Night
2025-05-27 21:45:13	27.3	52.9	3946	0	-0.4	Night
2025-05-27 21:45:14	27.3	53.1	3940	0	-0.5	Night
2025-05-27 21:45:15	27.3	53.1	3951	0	-0.3	Night
2025-05-27 21:45:16	27.3	53.3	3952	0	-0.4	Night
2025-05-27 21:45:17	27.3	53.3	3947	0	-0.4	Night
2025-05-27 21:45:18	27.3	53.5	3948	0	-0.3	Night
2025-05-27 21:45:19	27.3	53.5	3952	0	-0.5	Night
2025-05-27 21:45:20	27.3	53.8	3949	0	-0.4	Night
2025-05-27 21:45:21	27.3	53.8	3952	0	-0.3	Night
2025-05-27 21:45:22	27.3	53.7	3941	0	-0.2	Night
2025-05-27 21:45:23	27.3	53.7	3945	0	-0.2	Night
2025-05-27 21:45:24	27.3	53.6	3952	0	-0.4	Night
2025-05-27 21:45:25	27.3	53.6	3950	0	-0.1	Night
2025-05-27 21:45:26	27.3	53.4	3950	0	-0.2	Night
2025-05-27 21:45:27	27.3	53.4	3952	0	-0.3	Night
2025-05-27 21:45:28	27.3	53.1	3952	0	-0.4	Night
2025-05-27 21:45:29	27.3	53.1	3951	0	-0.3	Night
2025-05-27 21:45:30	27.3	53	3951	0	-0.3	Night
2025-05-27 21:45:31	27.3	53	3942	0	-0.3	Night
2025-05-27 21:45:32	27.3	53	3949	0	-0.2	Night
2025-05-27 21:45:33	27.3	53	3953	0	-0.1	Night
2025-05-27 21:45:34	27.3	53.2	3959	0	-0.3	Night

Figure 4-3. Day1 Dataset (Night)

4.2. Machine Learning Results

4.2.1. Random Forest Training Code Results

There were no mistakes by the model as the precision, recall, and F1-score all stand at 1.00 for Night (0), Cloudy (1), and Sunny (2). The confusion matrix proves this by showing no errors between what was predicted and what the results were in reality. The matrix was made into an image, the trained model was saved as a .pkl file, and the classification report was left in a text file. The machine learning workforce finished its tasks without a problem (Completed with a success code of 0), which shows that all went smoothly.

```
Dropped column: 'Date_Time'
Data split into training and testing sets
Training Random Forest model...
Random Forest model trained successfully
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1714
           1       1.00      1.00      1.00      1623
           2       1.00      1.00      1.00      8721

    accuracy                           1.00     12058
   macro avg       1.00      1.00      1.00     12058
weighted avg       1.00      1.00      1.00     12058

Confusion matrix saved to confusion_matrix_random.png
Random Forest model saved to .pkl file
Classification report saved to classification_report_random.txt
```

Figure 4-4. Random Forest Results

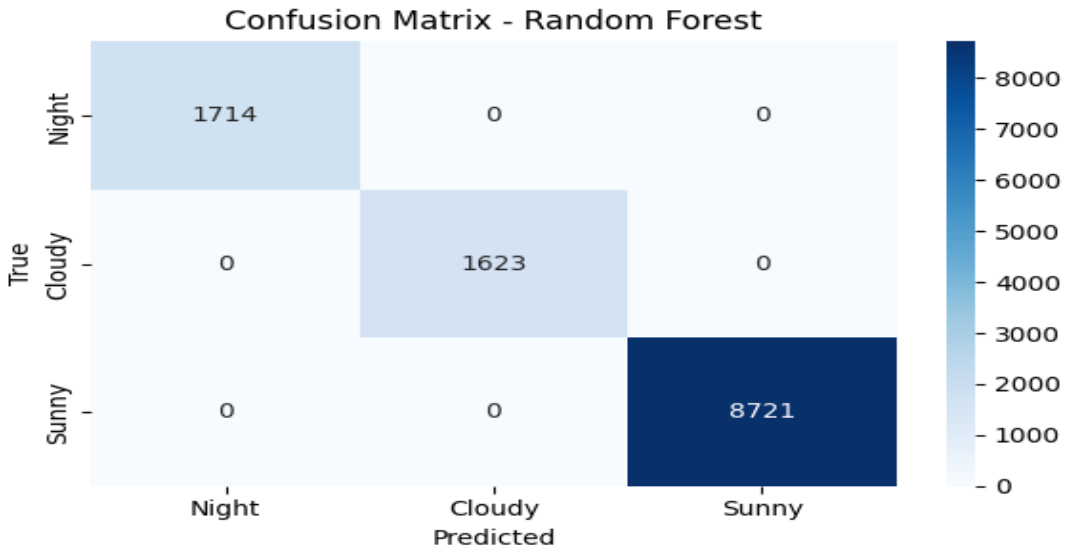

Figure 4-5. Random Forest Confusion Matrix

4.2.2. Random Forest Testing Code Results

```
Dropped column: 'Date_Time'
Predictions saved to: predicted_results_randomforest.xlsx
Prediction Accuracy on Test File: 100.00%

Process finished with exit code 0
```

Figure 4-6. Random Forest Test Results

According to the console, the code worked properly and there were no errors during the prediction (Process finished with exit code 0). First, the script confirmed that the extra "Date_Time" column was spotted in the input dataset and it was removed. It goes on to store the results of the predictions in an Excel file called predicted_results_randomforest.xlsx.

Most importantly, the Random Forest model reached an accuracy of 100.00% on the test file since all its predictions were in agreement with the actual classes in the dataset. The outcome shows that the model is very effective with this set of data. Still, a result of 100% accuracy might require more review, just to ensure that the test data covers a wide range and does not resemble the data used to train the model.

This was verified by using a different dataset and still, 100% accuracy was achieved.

Temperature	Humidity	Light	Voltage	Current	Real Weather	predicted_label	predicted_label_name
26.3	56.4	3855	0	-0.3	Night	0	Night
24.1	58.7	2347	0.52	-0.3	Cloudy	1	Cloudy
24.1	58.7	2352	0.51	-0.3	Cloudy	1	Cloudy
24.2	58.7	2352	0.52	-0.5	Cloudy	1	Cloudy
24.2	58.7	2351	0.51	-0.4	Cloudy	1	Cloudy
24.3	58.6	2346	0.52	-0.3	Cloudy	1	Cloudy
24.3	58.6	2340	0.52	-0.3	Cloudy	1	Cloudy
24.3	58.8	2339	0.53	-0.5	Cloudy	1	Cloudy
24.3	58.8	2335	0.53	-0.2	Cloudy	1	Cloudy
24.3	59	2337	0.52	-0.4	Cloudy	1	Cloudy
24.3	59	2339	0.52	-0.3	Cloudy	1	Cloudy
24.4	59.1	2338	0.52	-0.2	Cloudy	1	Cloudy
24.4	59.1	2339	0.53	-0.4	Cloudy	1	Cloudy
24.4	59	2335	0.53	-0.4	Cloudy	1	Cloudy
24.4	59	2336	0.53	-0.1	Cloudy	1	Cloudy
24.4	58.9	2327	0.53	0.1	Cloudy	1	Cloudy
24.4	58.9	2327	0.53	-0.4	Cloudy	1	Cloudy
24.5	58.8	2322	0.53	-0.4	Cloudy	1	Cloudy
24.5	58.8	2325	0.53	-0.4	Cloudy	1	Cloudy
24.5	58.8	2321	0.53	-0.2	Cloudy	1	Cloudy
24.5	58.8	2321	0.53	-0.2	Cloudy	1	Cloudy
24.5	58.8	2319	0.53	-0.3	Cloudy	1	Cloudy
24.5	58.8	2321	0.53	-0.3	Cloudy	1	Cloudy
24.6	58.8	2315	0.53	-0.5	Cloudy	1	Cloudy

Figure 4-7. Random Forest Accurate Predictions

4.2.3. SVM Training Code Results

The first step upon running the code was to provide the name of the Excel file that will be used as a dataset. In this case, the file is called day2_labeled.xlsx. This input allowed the program to locate and load the dataset. Once the file was successfully read, the code proceeds through the steps described in the code explanation. The column “Date_Time” is dropped. The outcomes include a printed classification report, a Confusion Matrix, and the export of both the trained model and scaler for the testing code.

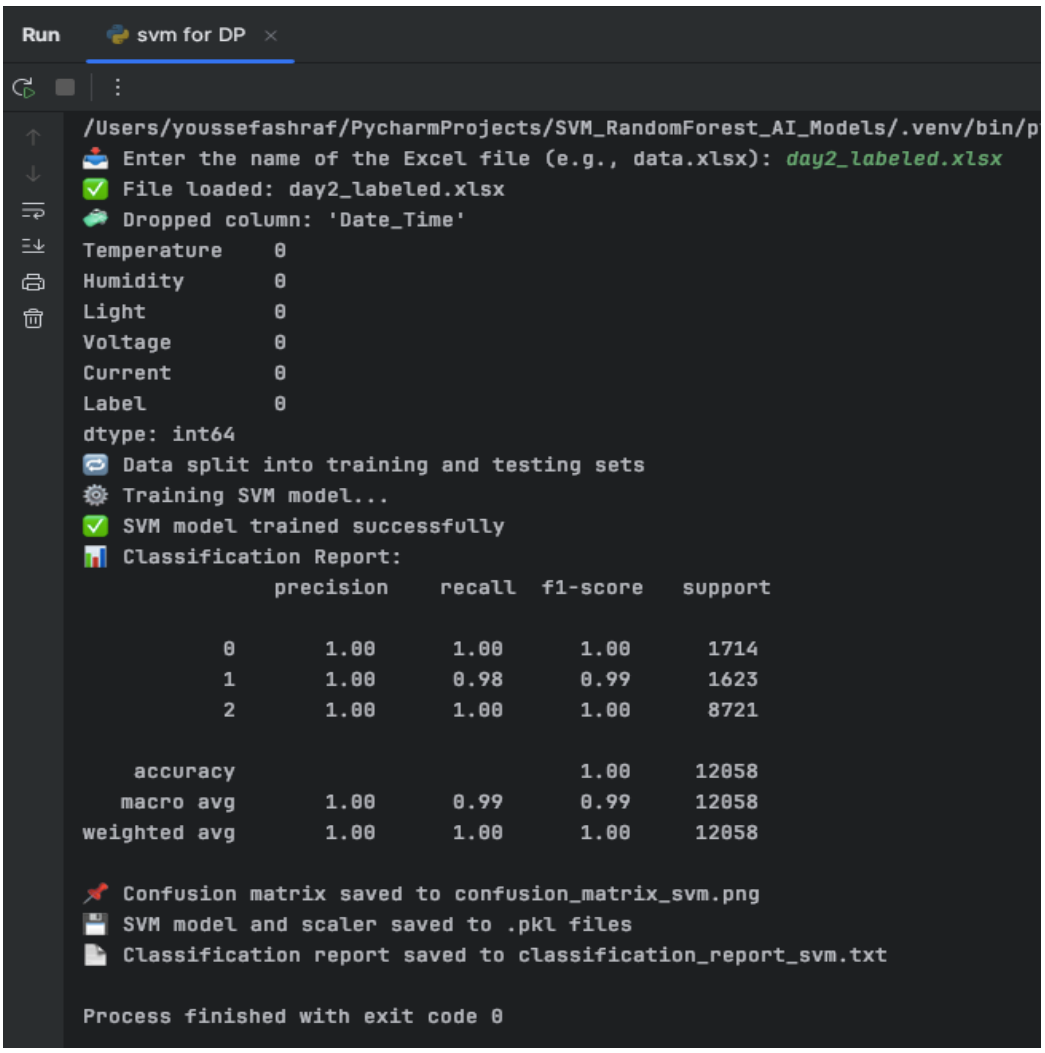

Figure 4-8. SVM Results

The classification report offers an in-depth assessment of how well the SVM model performs regarding the three non-identical weather condition classifications, which are Night (0), Cloudy (1), and Sunny (2). For each class, three key metrics are presented: precision, recall, and f1-score. Precision refers to the accuracy of the positive predictions. For example, the model predicted "Sunny" correctly 100% of the time when it claimed a sample was Sunny. Recall indicates how well the model was able to identify all actual instances of a class. In this case, 100% of actual Sunny cases were correctly identified.

The overall efficacy for each class is represented by the F1-score which is a metric that represents a balanced mean of both precision and recall. The amount of samples in each class in the test set is displayed in the "support" column. The general accuracy of the model on the 12,058 test samples was 100%. The macro average gives an unweighted mean across all classes, while the weighted average considers the class distribution. These results suggest that the model is performing the best at “Sunny” predictions, while struggling a bit with “Night” and “Cloudy”.

Figure 3-40 shows the Confusion Matrix:

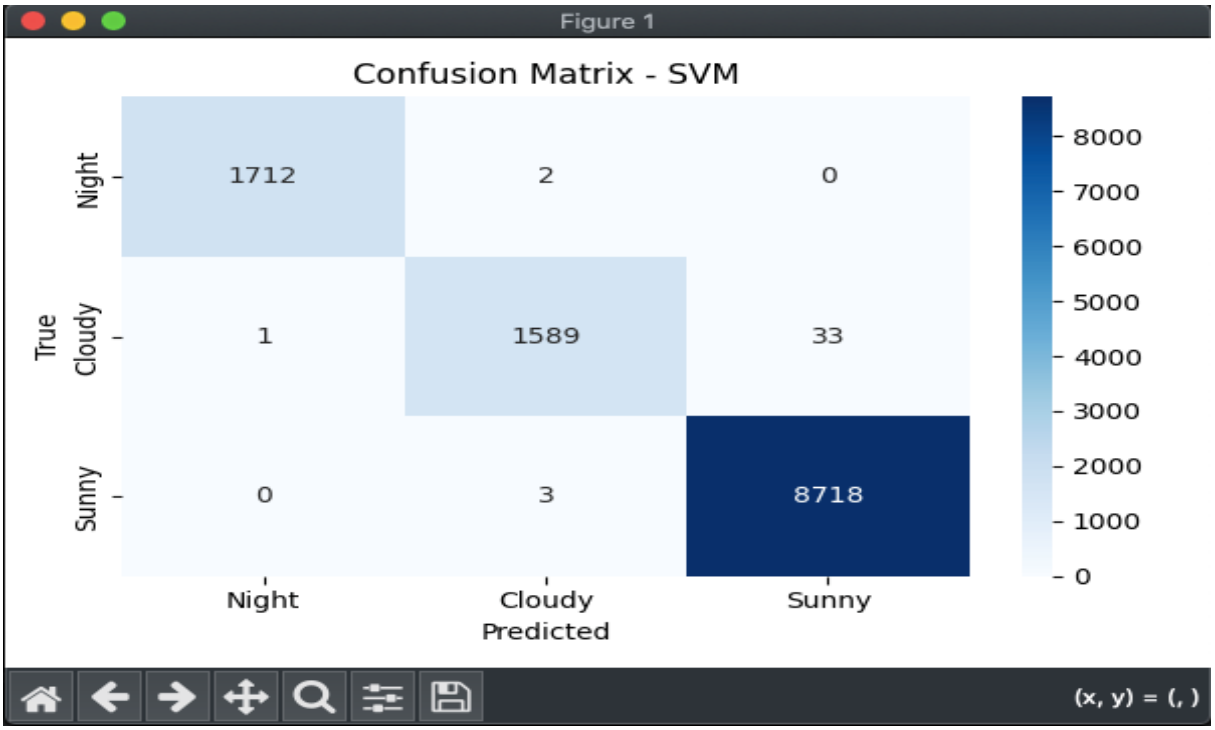

Figure 4-9. SVM Confusion Matrix

In the confusion matrix, each row signifies the true class, whereas each column denotes the predicted class. The accurate classifications are found along the diagonal of the matrix, where the true class aligns with the predicted class.

From the matrix, we observe that the SVM model correctly identified 1712 instances of "Night", 1589 instances of "Cloudy", and 8718 instances of "Sunny". However, there are some misclassifications. There are 2 "Night" instances that were misclassified as "Cloudy". Similarly, the model misclassified 1 "Cloudy" instance as "Night" and 33 as "Sunny". In the case of "Sunny", 3 instances were predicted as "Cloudy".

These results indicate that while the model performs amazingly well, it is still not perfect. Overall, the Classification Matrix provided a visual summary that can be used to highlight areas for potential improvement.

4.2.4. SVM Testing Code Results

Similar to the training code, once the testing code is ran, it requires an input from the user of the new excel dataset file in .xlsx format. Once that is input, the code performs all the steps described above, and outputs the percentage of accuracy, as well as a new excel file containing the predicted values as well as everything from dataset file that was input from the user.

Figure 4-10. SVM Test Results

The SVM model achieved an accuracy of 95.39% in predicting weather conditions on a new dataset. The results, saved in a new Excel file, show that while the SVM provides amazing predictive power, it can still make some mistakes. Notably, the Random Forest model used demonstrated significantly better accuracy, highlighting its stronger suitability for this classification task.

Figure 3-53 below is a snippet of the output excel file called “predicted_results_svm.xlsx”.

	A	B	C	D	E	F	G	H
1	Temperature	Humidity	Light	Voltage	Current	Label	predicted_label	predicted_label_name
2	26.3	56.4	3855	0	-0.3	Night	0	Night
3	24.1	58.7	2347	0.52	-0.3	Cloudy	0	Night
4	24.1	58.7	2352	0.51	-0.3	Cloudy	0	Night
5	24.2	58.7	2352	0.52	-0.5	Cloudy	0	Night
6	24.2	58.7	2351	0.51	-0.4	Cloudy	0	Night
7	24.3	58.6	2346	0.52	-0.3	Cloudy	0	Night
8	24.3	58.6	2340	0.52	-0.3	Cloudy	0	Night
9	24.3	58.8	2339	0.53	-0.5	Cloudy	0	Night
10	24.3	58.8	2335	0.53	-0.2	Cloudy	0	Night
11	24.3	59	2337	0.52	-0.4	Cloudy	0	Night
12	24.3	59	2339	0.52	-0.3	Cloudy	0	Night
13	24.4	59.1	2338	0.52	-0.2	Cloudy	0	Night
14	24.4	59.1	2339	0.53	-0.4	Cloudy	0	Night
15	24.4	59	2335	0.53	-0.4	Cloudy	0	Night
16	24.4	59	2336	0.53	-0.1	Cloudy	0	Night
17	24.4	58.9	2327	0.53	0.1	Cloudy	0	Night

Figure 4-11. SVM Predicted1

As seen, the model incorrectly predicts “Cloudy” weather as “Night” in the first rows. After that though, it starts predicting correctly, as seen in the next snippet.

35	24.6	58.4	2298	0.55	-0.3	Cloudy	1	Cloudy
36	24.6	58.4	2301	0.54	-0.5	Cloudy	0	Night
37	24.6	58.3	2294	0.55	-0.5	Cloudy	1	Cloudy
38	24.6	58.3	2295	0.55	-0.1	Cloudy	1	Cloudy
39	24.7	58.2	2298	0.55	-0.3	Cloudy	1	Cloudy
40	24.7	58.2	2303	0.55	-0.3	Cloudy	1	Cloudy
41	24.7	58.2	2289	0.55	-0.3	Cloudy	1	Cloudy
42	24.7	58.2	2295	0.56	-0.4	Cloudy	1	Cloudy
43	24.7	58.1	2289	0.56	-0.4	Cloudy	1	Cloudy
44	24.7	58.1	2288	0.56	-0.3	Cloudy	1	Cloudy
45	24.7	58	2285	0.56	-0.3	Cloudy	1	Cloudy
46	24.7	58	2284	0.57	-0.4	Cloudy	1	Cloudy
47	24.7	58	2286	0.56	-0.3	Cloudy	1	Cloudy
48	24.7	58	2282	0.57	-0.3	Cloudy	1	Cloudy
49	24.7	57.9	2283	0.56	-0.4	Cloudy	1	Cloudy
50	24.7	57.9	2281	0.57	-0.1	Cloudy	1	Cloudy
51	24.7	57.8	2277	0.57	-0.2	Cloudy	1	Cloudy
52	24.7	57.8	2276	0.57	-0.3	Cloudy	1	Cloudy
53	24.8	57.7	2269	0.57	0	Cloudy	1	Cloudy
54	24.8	57.7	2273	0.57	-0.2	Cloudy	1	Cloudy
55	24.8	57.7	2271	0.58	-0.4	Cloudy	1	Cloudy
56	24.8	57.7	2269	0.57	-0.4	Cloudy	1	Cloudy
57	24.8	57.6	2271	0.58	-0.4	Cloudy	1	Cloudy
58	24.8	57.6	2271	0.58	-0.2	Cloudy	1	Cloudy
59	24.8	57.5	2261	0.58	-0.3	Cloudy	1	Cloudy
60	24.8	57.5	2263	0.57	-0.2	Cloudy	1	Cloudy
61	24.8	57.4	2257	0.59	-0.2	Cloudy	1	Cloudy
62	24.8	57.4	2258	0.58	-0.4	Cloudy	1	Cloudy
63	24.8	57.3	2254	0.59	0.1	Cloudy	1	Cloudy

Figure 4-12. SVM Predicted2

	Random Forest			SVM		
Class	**Precision**	**Recall**	**F1-Score**	**Precision**	**Recall**	**F1-Score**
Night(0)	1	1	1	1	1	1
Cloudy(1)	1	1	1	1	0.98	0.99
Sunny(2)	1	1	1	1	1	1
Testing on another Dataset Accuracy	100%			95.39%		

Table 4-2. Comparison Between Random Forest and SVM Performance

4.3. ThingSpeak Functionality

ThingSpeak platform is currently fully operational and obtains real-time data fed by the ESP32-based solar monitoring system. The system is capable of sending sensor values comprising of temperature, humidity, light intensity, voltage and current to specified fields in a personal ThingSpeak channel. There is also the Wi-Fi connectivity option, and at set intervals, the data is uploaded, and the user is then able to remotely monitor the system performance via graph and dashboards in real-time. This live visualization is a comfortable mode to monitor the variations of environmental and electrical values during the day, particularly helpful to notice the tendency or the most productive sun hours or misconduct in the solar panel. ThingSpeak has become a stable cloud-based interface that supplements the local data storage of the SD card.

Figure 4-13. ThingSpeak Visualization

4.4. Real-time Monitoring

Now, readings can be seen real-time through the IoT platform ThingSpeak, but what about the AI integration? And where exactly is solar monitoring of the weather condition? That is where real monitoring python codes have come into play. From the machine learning models files where AI have learnt from the collected dataset, the python codes (Check Appendix D) will allow the allow the user to see current weather condition real-time by integrating the AI models.

Figure 4-14. Real-time Monitoring Through Telegram on Sunny Day

Both Python script are an automated monitoring and alerting system that consists of ThingSpeak, machine learning, and Telegram messaging. They primarily read real-time solar monitoring data off the ThingSpeak channel, run it through a pre-trained machine learning models (Random Forest/SVM) to predict the current environmental state whether it be sunny, cloudy or night, and finally post a prediction and the most recent sensor values to a Telegram chatbot.

Both codes start with filling in the values needed of ThingSpeak (channel ID) and Telegram (bot token and chat ID). The script loads the trained model (modelrandom.pkl) and specified expected features such as the temperature, humidity, light, voltage, and current. It is a continually running loop that collects the updated sensor data at a 15-second interval on ThingSpeak. They predict with both models after extracting and formatting the sensor values. The label predicted is then decoded into a string text telling the current weather.

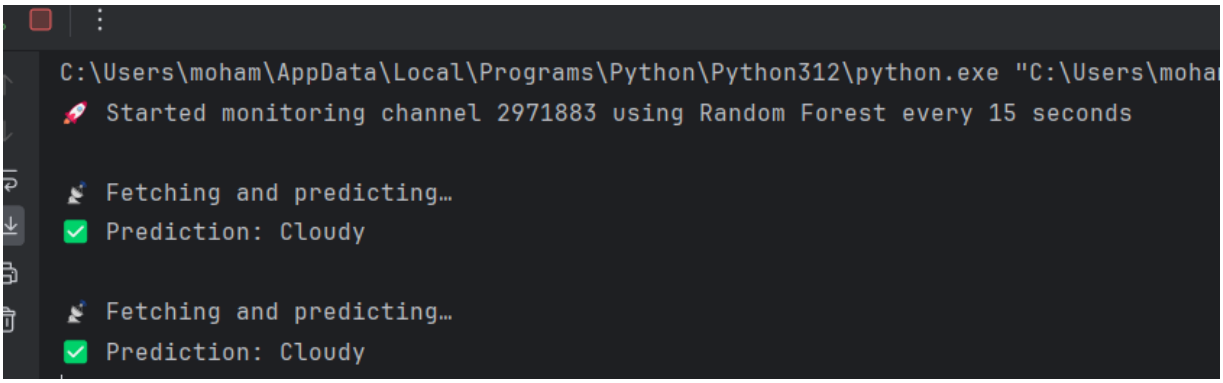

Figure 4-15. Real-time Code Results

Lastly, the message is forwarded through a Telegram bot to a user or group. With this arrangement, it is possible to monitor weather conditions in real-time and with intelligent notifications, users can be made immediately aware of the weather and solar panel conditions without having to manually refresh the ThingSpeak dashboard.

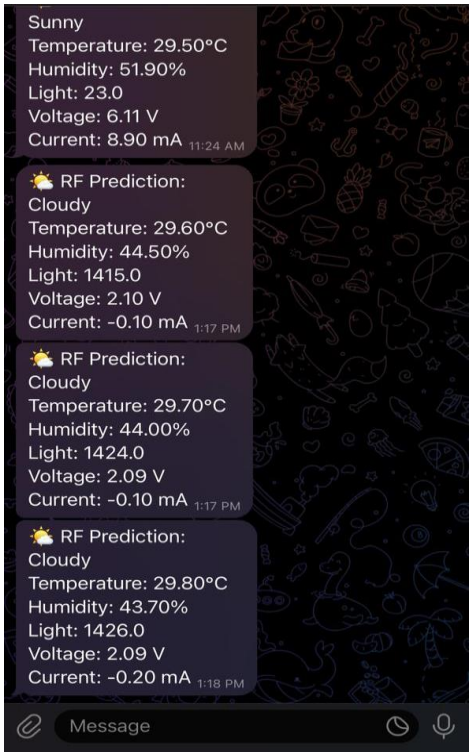

Figure 4-16. Real-Time Monitoring Through Telegram Chat

5. Chapter (5): Conclusion

In essence, this project was able to design and develop a real-time solar monitoring system based on the ESP32 microcontroller by combining environmental and electrical sensors to collect the necessary data to analyze the performance panel through monitoring weather condition. The data was stored reliably both locally on an SD card and on the ThingSpeak IoT platform remotely which provided options of continuous monitoring and remote access of the data. The system was able to gather massive amounts of data within 8 days of operation, with an average of more than 57000 records per day that offers a significantly-sized dataset to perform machine learning classification. Random Forest and Support Vector Machine (SVM) were the two models that were trained to categorize the weather conditions with respect to sensor input. Both models indicated excellent accuracy, but the Random Forest model gained the highest marks in all assessment metrics, which is somewhat better than the SVM that encountered a slight decrease in recall in cloudy weather. All in all, the project demonstrates how low-cost IoT hardware and machine learning methods could be used to develop intelligent, real-time monitoring systems that are capable of supporting energy efficiency.

5.1. Future Recommendations

In the future, there are many possibilities to augment and enhance the system for monitoring solar energy. While the prototype offers good results, improving it could lead to better accuracy, more sturdy machines and more uses.

An important update would be to put more sensors. Currently, the system reads temperature, humidity, how much light there is, current and voltage. Sensors that measure sunlight, how dirty or clean the panels are, the direction the panels are facing, and wind speed would give a better idea of aspects affecting solar power. Using more information would allow the AI model to make better weather predictions and find the best way to use energy.

To this day, developers still focus on making the AI models better. High-tech AI techniques may assist in predicting changes in the weather by improving the process. In the future, machines might suggest how often panels should be cleaned or checked based on various environmental facts. Additionally, If a site's Wi-Fi is not reliable, sending data using LoRa or GSM might improve the data transfer process as they can communicate further and use less power [67].

Improving cloud services is also possible. Although ThingSpeak is decent for the most basic data logging and representation, more developed features could be included in future versions through AWS IoT or Google Cloud .

The project can achieve significant growth in the future. As there are more sensors, smart AI, improved ways to communicate and modular designs, future developments that will strengthen the system's reliability.

References

[1] C. Breyer, S. Khalili, and D. Bogdanov, "Solar photovoltaic capacity demand for a sustainable transport sector to fulfil the Paris Agreement by 2050," *Prog. Photovolt. Res. Appl.*, vol. 27, pp. 978–989, 2019, doi: 10.1002/pip.3114.

[2] J. S. G. Collazos, L. M. C. Ardila, and C. J. F. Cardona, "Energy transition in sustainable transport: concepts, policies, and methodologies," *Environ. Sci. Pollut. Res.*, vol. 31, pp. 58669–58686, 2024, doi: 10.1007/s11356-024-34862-x.

[3] P. Soni, R. Sinha, and S. R. Perret, "Energy use and efficiency in selected rice-based cropping systems of the Middle-Indo Gangetic Plains in India," *Energy Rep.*, vol. 4, pp. 554–564, 2018, doi: 10.1016/j.egyr.2018.09.001.

[4] H. Ritchie and P. Rosado, "Energy Mix," *OurWorldinData.org*, 2020. [Online]. Available: https://ourworldindata.org/energy-mix. [Accessed: Feb. 1, 2025].

[5] H. H. Pourasl, R. Vatankhah Barenji, and V. M. Khojastehnezhad, "Solar energy status in the world: A comprehensive review," *Energy Rep.*, vol. 10, pp. 3474–3493, 2023, doi: 10.1016/j.egyr.2023.10.022.

[6] T. Z. Ang, M. Salem, M. Kamarol, H. S. Das, M. A. Nazari, and N. Prabaharan, "A comprehensive study of renewable energy sources: Classifications, challenges and suggestions," Energy Strat. Rev., vol. 43, p. 100939, 2022, doi: 10.1016/j.esr.2022.100939.

[7] A. Shahsavari, A. Karimi, M. Akbari, and M. Alizadeh-Noughani, "Environmental impacts and social cost of non-renewable and renewable energy sources: A comprehensive review," *J. Renew. Sustain. Energy*, vol. 11, pp. 12–27, 2024.

[8] S. Pilakkat, “Study of the Importance of MPPT Algorithm for Photovoltaic Systems under Abrupt Change in Irradiance and Temperature Conditions,” WSEAS Trans. Power Syst., vol. 15, 2020.

[9] A. Shahsavari and M. Akbari, "Potential of solar energy in developing countries for reducing energy-related emissions," *Renew. Sustain. Energy Rev.*, vol. 90, pp. 275–291, 2018, doi: 10.1016/j.rser.2018.03.065.

[10] Enerdata. Yearbook. Global energy statistical yearbook 2016. CO2 emissions from fuel combustion; 2016.

[11] "CSP and PV: Differences & Comparison," *SolarFeeds Magazine*, [Online]. Available: https://www.solarfeeds.com/mag/csp-and-pv-differences-comparison/. [Accessed: Feb. 1, 2025].

[12] U. Desideri, F. Zepparelli, V. Morettini, and E. Garroni, "Comparative analysis of concentrating solar power and photovoltaic technologies: Technical and environmental evaluations," *Appl. Energy*, vol. 102, pp. 765–784, 2013, doi: 10.1016/j.apenergy.2012.08.033.

[13] M. R. A. A. El-Samahy, M. Daowd, and A. M. A. Amin, "A comparative study on photovoltaic and concentrated solar thermal power plants," *Recent Advances in Environmental and Earth Sciences and Economics*, vol. 3, no. 5, pp. 48–56, 2014.

[14] "How Concentrated Solar Power (CSP) Works," *JLanka Technologies*, [Online]. Available: https://www.jlanka.com/3-3-concentrated-solar-power-csp-work/. [Accessed: Feb. 8, 2025].

[15] W. Wang and B. Laumert, “Simulate a Sun for Solar Research: A Literature Review of Solar Simulator Technology”, Stockholm, Sweden, 2014.

[16] D. Shah, “Development and characterization of solar simulator for solar cells,” J. Nanoelectron. Optoelectron., vol. 15, pp. 673–776, 2020.

[17] A. S. Al-Ezzi and M. N. M. Ansari, "Photovoltaic Solar Cells: A Review," *Applied System Innovation*, vol. 5, no. 4, p. 67, 2022, doi: 10.3390/asi5040067.

[18] K. Z. Mostofa and M. A. Islam, "Creation of an Internet of Things (IoT) system for the live and remote monitoring of solar photovoltaic facilities," *Energy Reports*, vol. 9, Suppl. 11, pp. 422-427, 2023, doi: 10.1016/j.egyr.2023.09.060.

[19] B. Safyanu, M. N. Abdullah, and Z. Omar, "Review of Power Device for Solar-Powered Aircraft Applications," *Journal of Aerospace Technology and Management*, vol. 11, 2019, doi: 10.5028/jatm.v11.1077.

[20] Schematic of a solar simulator with a Xenon arc lamp for measuring the IV characteristics of solar cells under illumination,”PVmet Wiki” [Online]. Available: https://wiki.pvmet.org/index.php?title=File:Figure_2 _Schematic_of_a_solar_simulator_with_a_Xenon_arc_lamp_for_measuring_the_IV_characteristics _of_solar_cells_under_illumination.png. [Accessed: Feb. 08, 2025].

[21] Dzimano, G. Modeling of Photovoltaic Systems; Ohio State University: Columbus, OH, USA, 2008.

[22] Kawamoto, H.; Guo, B. Improvement of an electrostatic cleaning system for removal of dust from solar panels. J. Electrostat. 2018, 91, 28–33.

[23] National Renewable Energy Laboratory (NREL), “Solar Spectra - Air Mass 1.5,” NREL, 2024. [Online]. Available: https://www.nrel.gov/grid/solar-resource/spectra-am1.5.html/. [Accessed: Feb. 8, 2025].

[24] S. Ansari, A. Ayob, M. S. Hossain Lipu, M. H. Md Saad, and A. Hussain, "A Review of Monitoring Technologies for Solar PV Systems Using Data Processing Modules and Transmission Protocols: Progress, Challenges and Prospects," Sustainability, vol. 13, p. 8120, 07/21 2021, doi: 10.3390/su13158120.

[25] Instructables, “Arduino and DHT22 (AM2302) Temperature Measurement,” *Instructables*, Aug. 18, 2016. https://www.instructables.com/Arduino-and-DHT22-AM2302-Temperature-Measurement .

[26] ”Buy LDR Module 4 pin LM393 Photosensitive Light-Dependent Control Sensor in Egypt @,” *Micro Ohm Electronics Egypt - Arduino Egypt*, Apr. 19, 2024. https://microohm-eg.com/product/lm393-photosensitive-light-dependent-control-sensor-ldr-module-4-pin/?srsltid=AfmBOoqtX0KjkZbVDfXwO7uhI7S0J4w7t_9j9IJUBkVKV-DL1Hc2laGC .

[27] J. Wilson, “INA219 Current Sensor Module Datasheet, Pinout, Features & Applications,” *The Engineering Projects*, Feb. 8, 2021. [Online]. Available: https://www.theengineeringprojects.com/2021/02/ina219-current-sensor-module-datasheet-pinout-features-applications.html

[28] M. Babiuch, P. Foltynek, and P. Smutný, Using the ESP32 Microcontroller for Data Processing. 2019, pp. 1-6.

[29] M. Abu Radia et al., "IoT-Based Wireless Data Acquisition and Control System for Photovoltaic Module Performance Analysis," *e-Prime - Advances in Electrical Engineering, Electronics and Energy*, vol. 6, p. 100348, 2023.

[30] “ESP32 HTTP GET and HTTP POST with Arduino IDE | Random Nerd Tutorials,” Apr. 08, 2020. https://randomnerdtutorials.com/esp32-http-get-post-arduino/

[31] Admin, “IoT Based Solar Power Monitoring System with ESP32,” *How To Electronics*, Nov. 19, 2022. https://how2electronics.com/iot-based-solar-power-monitoring-system-with-esp32/

[32] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, pp. 5–32, 2001, doi: 10.1023/A:1010933404324.

[33] R. Fezai, K. Dhibi, M. Mansouri, M. Trabelsi, M. Hajji, K. Bouzrara, H. Nounou, and M. Nounou, "Effective Random Forest-Based Fault Detection and Diagnosis for Wind Energy Conversion Systems," *IEEE Sensors Journal*, vol. 1, pp. 1530-437, 2021, doi: 10.1109/JSEN.2020.3037237.

[34] M. Helu, T. Sprock, D. Hartenstine, R. Venketesh, and W. Sobel, "Scalable data pipeline architecture to support the industrial internet of things," *CIRP Annals*, vol. 69, no. 1, pp. 385-388, 2020. doi: 10.1016/j.cirp.2020.04.006

[35] J. Machacek, Z. Prochazka, and J. Drapela, "System for measuring and collecting data from solar-cell systems," in Proceedings of the 2007 International Conference on Electrical Power Quality and Utilisation (EPQU), 2007, pp. 1-4, doi: 10.1109/EPQU.2007.4424164

[36] **"SD Card Module,"** Scribd, [Online]. Available: https://www.scribd.com/document/737933911/SD-Card-Module. [Accessed: 2-Feb-2025]

[37] **S. Bhosale, M. Patil, and P. Patil,** "SQLite: Light Database System," *International Journal of Computer Science and Mobile Computing*, vol. 44, pp. 882-885, 2015.

[38] **P. Chougale, V. Yadav, A. Gaikwad, and B. Student,** "FIREBASE – OVERVIEW AND USAGE," *Journal of Engineering and Technology Management*, vol. 2022, pp. 2582-5208, 2022.

[39] Afreen Khalfe, “Data Preprocessing: Cleaning, Transforming, and Normalizing Data for Analysis – Talent500-blog,” Talent500.com, Sep. 07, 2023. https://talent500.com/blog/data-preprocessing/

[40] D. Jain, “Data Preprocessing in Data Mining,” GeeksforGeeks, Mar. 12, 2019. https://www.geeksforgeeks.org/data-preprocessing-in-data-mining/

[41] S. K. Pal and P. Mitra, Pattern recognition algorithms for data mining. Chapman and Hall/CRC, 2004.

[42] S. Raschka, J. Patterson, and C. Nolet, "Machine Learning in Python: Main Developments and Technology Trends in Data Science, Machine Learning, and Artificial Intelligence," Information, vol. 11, no. 4, p. 193, 2020. [Online]. Available: https://www.mdpi.com/2078-2489/11/4/193.

[43] Z. Zamanzadeh Darban, G. I. Webb, S. Pan, C. Aggarwal, and M. Salehi, "Deep learning for time series anomaly detection: A survey," ACM Computing Surveys, vol. 57, no. 1, pp. 1-42, 2024.

[44] Neelarghya, "Reinforcement Learning vs Genetic Algorithm — AI for Simulations," XRPractices, Jul. 26, 2021. https://medium.com/xrpractices/reinforcement-learning-vs-genetic-algorithm-ai-for-simulations-f1f484969c56

[45] P. W. Khan, Y.-C. Byun, and S.-J. Lee, "Optimal photovoltaic panel direction and tilt angle prediction using stacking ensemble learning," Frontiers in Energy Research, vol. 10, p. 865413, 2022.

[46] T. Le, C. P. Jayabal, H. Le, N. V. L. Le, M. Duong, and D. N. Cao, "Harnessing artificial intelligence for data-driven energy predictive analytics: A systematic survey towards enhancing sustainability," International Journal of Renewable Energy Development, vol. 13, 02/10 2024, doi: 10.61435/ijred.2024.60119.

[47] F. Maleki, N. Muthukrishnan, K. Ovens, C. Reinhold, and R. Forghani, "Machine learning algorithm validation: from essentials to advanced applications and implications for regulatory certification and deployment," Neuroimaging Clinics, vol. 30, no. 4, pp. 433-445, 2020.

[48] D. Hercog, T. Lerher, M. Truntic, and O. Težak, "Design and Implementation of ESP32-Based IoT Devices," *Sensors*, vol. 23, no. 15, p. 6739, 2023, doi: 10.3390/s23156739. idkkkkkkkk

[49] S. H. Cheragee, N. Hassan, S. Ahammed, and A. Z. M. T. Islam, "A Study of IoT Based Real-Time Solar Power Remote Monitoring System," *Int. J. Ambient Syst. Appl. (IJASA)*, vol. 9, no. 1/2, pp. 27–36, Jun. 2021, doi: 10.5121/ijasa.2021.9204.

[50] S. Madadi, "A Study of Solar Power Monitoring System Using Internet of Things (IOT)," *Int. J. Innov. Sci. Res. Technol.*, vol. 6, no. 5, pp. 347–350, May 2021

[51] S. A. Sheikh, D. Dhuware, R. Gharsele, A. Mishra, S. Wakade, and S. Sadmake, "Solar Power Monitoring System Using IoT," *Int. Res. J. Mod. Eng. Technol. Sci.*, vol. 5, no. 4, pp. 2847–2852, Apr. 2023, doi: 10.56726/IRJMETS36223.

[52] G. V. Bhau, R. G. Deshmukh, T. R. Kumar, S. Chowdhury, Y. Sesharao, and Y. Abilmazhinov, "IoT based solar energy monitoring system," *Mater. Today Proc.*, vol. 80, pp. 3697–3701, 2023, doi: 10.1016/j.matpr.2021.07.364.

[53] M. Katyarmal, S. Walkunde, A. Sakhare, and U. S. Rawandale, "Solar power monitoring system using IoT," *Int. Res. J. Eng. Technol. (IRJET)*, vol. 5, no. 3, pp. 3431–3432, Mar. 2018.

[54] Codecademy Team, "Normalization," *Codecademy*, [Online]. Available: https://www.codecademy.com/article/normalization. [Accessed: Feb. 13, 2025].

[55] PythonGeeks Team, "Data Preprocessing in Machine Learning," *Python Geeks*, [Online]. Available: https://pythongeeks.org/data-preprocessing-in-machine-learning/. [Accessed: Feb. 13, 2025].

[56] M. Babiuch, P. Foltynek, and P. Smutný, "Using the ESP32 Microcontroller for Data Processing," in *Proc. Carpathian Control Conf.*, Kraków, Poland, 2019, pp. 1–6, doi: 10.1109/CarpathianCC.2019.8765944.

[57] Sviluppo Mania Team, "Using SQLite with Python," *Sviluppo Mania*, Dec. 21, 2021. [Online]. Available: https://www.sviluppomania.com/en/usare-sqlite-con-python/. [Accessed: Feb. 13, 2025].

[58] Firebase Developers, "How to Access and Download Files in Cloud Storage," *Medium*, [Online]. Available: https://medium.com/firebase-developers/firebase-how-to-access-and-download-files-in-cloud-storage-b8f2cf49aa13. [Accessed: Feb. 13, 2025].

[59] N. Datar, S. Bhoyar, A. Khan, S. Dekapurwar, H. Wankhede, S. Sonone, V. Bapat, and N. K. Dhote, "Solar Power Monitoring System Using IOT," *J. Emerg. Trends Electr. Eng.*, vol. 3, no. 1, pp. 1–5, Dec. 2023.

[60] F. I. Mulani, S. H. Kirkire, P. N. Pawar, and A. O. Mulani, "Weather Monitoring System using Internet of Things (IoT) and Solar Panel," *Int. J. Res. Publ. Rev.*, vol. 5, no. 5, pp. 3417–3420, May 2024.

[61] ESPBoards, "ESP32 PCF8563 Real-Time Clock (RTC)," *ESPBoards.dev*, Dec. 4, 2023. [Online]. Available: https://www.espboards.dev/sensors/pcf8563/

[62] S. Barker, "ThingSpeak: The Ultimate IoT Platform for Data Collection and Analysis," *ExpertBeacon*, Oct. 11, 2024. [Online]. Available: https://expertbeacon.com/thingspeak-the-ultimate-iot-platform-for-data-collection-and-analysis/

[63] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*, 2nd ed. Sebastopol, CA, USA: O'Reilly Media, 2019

[64] R. Yehoshua, "Random Forests," *Medium*, Mar. 25, 2023. [Online]. Available: https://medium.com/@roiyeho/random-forests-98892261dc49

[65] "Support Vector Machine (SVM) Algorithm," *GeeksforGeeks*, [Online]. Available: https://www.geeksforgeeks.org/support-vector-machine-algorithm/ [Accessed: June 1, 2025].

[66] "Support Vector Machines (SVM) Made Simple & How To Tutorial," Spot Intelligence, May 6, 2024. [Online]. Available: https://spotintelligence.com/2024/05/06/support-vector-machines-svm/ [Accessed: June 1, 2025].

[67] A. Lavric and P. Valentin, *Internet of Things and LoRa™ Low-Power Wide-Area Networks: A survey*. 2017, pp. 1-5.

Appendix A: ESP32 Programming Code on Arduino IDE

```
#include <Wire.h>
#include <Adafruit_INA219.h>
#include <DHT.h>
#include <SPI.h>
#include <SD.h>
#include <RTClib.h>
#include <WiFi.h>
#include <HTTPClient.h>
// DHT and pins
#define DHTPIN 2
#define DHTTYPE DHT22
DHT dht(DHTPIN, DHTTYPE);
// LDR and Voltage Divider
#define LDR_PIN 35
#define VOLTAGE_PIN 33
// MOSFET control pin
#define MOSFET_PIN 14
// SD card
#define SD_CS 5
File dataFile;
// INA219
Adafruit_INA219 ina219;
// RTC
RTC_PCF8563 rtc;
// Sampling
#define SAMPLING_FREQ_HZ 1
#define SAMPLING_INTERVAL_MS (1000 / SAMPLING_FREQ_HZ)
unsigned long lastSampleTime = 0;
// WiFi & ThingSpeak
const char* ssid = "MoTamim";
const char* password = "limeiscorpion";
const String apiKey = "FK62ZH9C2AYARZI1";
const String serverName = "http://api.thingspeak.com/update";
```

```
// Forward declaration
void sendToThingSpeak(float temp, float hum, int light, float voc, float curr);
void setup() {
  Serial.begin(115200);
  pinMode(MOSFET_PIN, OUTPUT);
  digitalWrite(MOSFET_PIN, LOW);
  dht.begin();
  ina219.begin();
  rtc.begin();
  DateTime now = rtc.now();
  if (now.year() < 2000) {
    rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));
  }
  SD.begin(SD_CS);
  // Connect to WiFi
  WiFi.begin(ssid, password);
  Serial.print("Connecting to WiFi");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nWiFi connected");
}
void loop() {
  unsigned long currentTime = millis();
  if (currentTime - lastSampleTime >= SAMPLING_INTERVAL_MS) {
    lastSampleTime = currentTime;
    DateTime now = rtc.now();
    char timeStr[20];
    sprintf(timeStr, "%04d-%02d-%02d %02d:%02d:%02d", now.year(), now.month(), now.day(), now.hour(), now.minute(),
now.second());
    float temp = dht.readTemperature();
    float hum = dht.readHumidity();
    int ldrVal = analogRead(LDR_PIN);
```

```
    // Voltage (Voc)
    digitalWrite(MOSFET_PIN, LOW);
    delay(5);
    float raw = analogRead(VOLTAGE_PIN);
    float v_adc = raw * (3.3 / 4095.0);
    float volt = v_adc * (6.05/2.22); // Adjust based on voltage divider
    // Current (Isc)
    digitalWrite(MOSFET_PIN, HIGH);
    delay(5);
    float curr_mA = ina219.getCurrent_mA();
    digitalWrite(MOSFET_PIN, LOW);
    Serial.printf("Time: %s | Temp: %.2f C | Hum: %.2f %% | Light: %d | Voc: %.2f V | Isc: %.2f mA\n",
                  timeStr, temp, hum, ldrVal, volt, curr_mA);
    dataFile = SD.open("/datalog.csv", FILE_APPEND);
    if (dataFile) {
      dataFile.printf("%s,%.2f,%.2f,%d,%.2f,%.2f\n", timeStr, temp, hum, ldrVal, volt, curr_mA);
      dataFile.close();
    }
    sendToThingSpeak(temp, hum, ldrVal, volt, curr_mA);
  }
}
void sendToThingSpeak(float temp, float hum, int light, float voc, float curr) {
  if (WiFi.status() == WL_CONNECTED) {
    HTTPClient http;
    String url = serverName + "?api_key=" + apiKey +
                 "&field1=" + String(temp) +
                 "&field2=" + String(hum) +
                 "&field3=" + String(light) +
                 "&field4=" + String(voc) +
                 "&field5=" + String(curr);
    http.begin(url);
    int httpCode = http.GET();
    if (httpCode > 0) {
      Serial.println("ThingSpeak Response Code: " + String(httpCode));
```

```
    } else {
      Serial.println("Error sending to ThingSpeak: " + http.errorToString(httpCode));
    }
    http.end();
  } else {
    Serial.println("WiFi not connected!");
  }
}
```

Appendix B: Random Forest Machine Learning Codes

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
import joblib

# 1. Read the Excel file (modify according to your file name)
df = pd.read_excel("day2_labeled.xlsx")  # Change "data.xlsx" to your file path

# 2. Drop the 'Date_Time' column if it exists
if "Date_Time" in df.columns:
    df.drop(columns=["Date_Time"], inplace=True)
    print("🗑️ Dropped column: 'Date_Time'")
else:
    print("ℹ️ Column 'Date_Time' not found — nothing dropped.")

# 3. Convert String labels to Numerical Values
df["Label"] = df["Label"].map({"Night": 0, "Cloudy": 1, "Sunny": 2})

# 4. Split features and labels
X = df.drop(columns=["Label"])
y = df["Label"]

# 5. Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
print("🔁 Data split into training and testing sets")

# 6. Train the SVM model
print("⚙️ Training Random Forest model...")
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
print("✅ Random Forest model trained successfully")

# 7. Evaluate the model
y_pred = model.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
report = classification_report(y_test, y_pred)
print("📊 Classification Report:\n", report)
```

```
# 8. Plot confusion matrix
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["Night", "Cloudy", "Sunny"], yticklabels=["Night",
"Cloudy", "Sunny"])
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Confusion Matrix - Random Forest")
plt.tight_layout()
plt.savefig("confusion_matrix_random.png")
plt.show()
print("📌 Confusion matrix saved to confusion_matrix_random.png")

# 9. Save the model for later use
joblib.dump(model, 'modelrandom.pkl')
print("💾 Random Forest model saved to .pkl file")

# 10. Save classification report
with open("classification_report_random.txt", "w") as f:
    f.write(report)
print("📄 Classification report saved to classification_report_random.txt")
```

……

```
import pandas as pd
import joblib

# 1. Load the saved model
model = joblib.load('modelrandom.pkl')

# 2. Read a new Excel file (change the file name here)
df_new = pd.read_excel('day3_labeled.xlsx')

if "Date_Time" in df_new.columns:
    df_new.drop(columns=["Date_Time"], inplace=True)
    print("🧠 Dropped column: 'Date_Time'")
else:
    print("ℹ️ Column 'Date_Time' not found — nothing dropped.")

# 3. Keep the original label column for later comparison
if "Label" in df_new.columns:
    actual_labels = df_new["Label"]
    df_features = df_new.drop(columns=["Label"])
else:
    actual_labels = None
    df_features = df_new.copy()

# 4. Predict the classes using the model
predicted_labels = model.predict(df_features)
df_new["predicted_label"] = predicted_labels

# 5. Map numeric predictions to readable labels
label_map = {0: "Night", 1: "Cloudy", 2: "Sunny"}
df_new["predicted_label_name"] = df_new["predicted_label"].map(label_map)

# 6. Save the results to an Excel file
output_file = "predicted_results_randomforest.xlsx"
df_new.to_excel(output_file, index=False)
print(f"📄 Predictions saved to: {output_file}")
```

```
from sklearn.metrics import accuracy_score

# 7. Compare predictions with actual labels
if actual_labels is not None:
    # Convert actual labels to numeric for comparison
    actual_labels_mapped = actual_labels.map({"Night": 0, "Cloudy": 1, "Sunny": 2})

    # Compute accuracy
    accuracy = accuracy_score(actual_labels_mapped, df_new["predicted_label"])
    print(f"📈 Prediction Accuracy on Test File: {accuracy * 100:.2f}%")
else:
    print("⚠️ No actual labels found for comparison.")
```

Appendix C: Support Vector Machine Learning Codes

```
import pandas as pd
import joblib
import matplotlib.pyplot as plt
import seaborn as sns
import os
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.exceptions import ConvergenceWarning
import warnings

# Ignore SVM convergence warnings
warnings.filterwarnings("ignore", category=ConvergenceWarning)

# 1. Get Excel filename from user
filename = input("📥 Enter the name of the Excel file (e.g., data.xlsx): ").strip()

# 2. Check if file exists
if not os.path.exists(filename):
    print(f"❌ The file '{filename}' was not found!")
    exit()

# 3. Load the dataset
df = pd.read_excel(filename)
print(f"✅ File loaded: {filename}")

# 4. Drop the 'Date_Time' column if it exists
if "Date_Time" in df.columns:
    df.drop(columns=["Date_Time"], inplace=True)
    print("🧠 Dropped column: 'Date_Time'")
else:
    print("ℹ Column 'Date_Time' not found — nothing dropped.")

# 5. Convert String labels to Numerical Values
df["Label"] = df["Label"].map({"Night": 0, "Cloudy": 1, "Sunny": 2})
df['Temperature'] = df['Temperature'].fillna(df['Temperature'].mean())
df['Humidity'] = df['Humidity'].fillna(df['Humidity'].mean())
print(df.isnull().sum())

# 6. Split features and labels
```

```
X = df.drop(columns=["Label"])
y = df["Label"]

# 7. Normalize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 8. Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
print("🔁 Data split into training and testing sets")

# 9. Train the SVM model
print("⚙️ Training SVM model...")
model = SVC(kernel='rbf', probability=True, random_state=42)
model.fit(X_train, y_train)
print("✅ SVM model trained successfully")

# 10. Evaluate the model
y_pred = model.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
report = classification_report(y_test, y_pred)
print("📊 Classification Report:\n", report)

# 11. Plot confusion matrix
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["Night", "Cloudy", "Sunny"], yticklabels=["Night",
"Cloudy", "Sunny"])
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Confusion Matrix - SVM")
plt.tight_layout()
plt.savefig("confusion_matrix_svm.png")
plt.show()
print("📌 Confusion matrix saved to confusion_matrix_svm.png")

# 12. Save model and scaler
joblib.dump(model, "svm_model.pkl")
joblib.dump(scaler, "scaler.pkl")
print("💾 SVM model and scaler saved to .pkl files")

# 13. Save classification report
with open("classification_report_svm.txt", "w") as f:
    f.write(report)
print("📄 Classification report saved to classification_report_svm.txt")
```

……..

```
import pandas as pd
import joblib
import os

# 1. Enter the name of the new test Excel file
filename = input("📥 Enter the name of the new Excel file to test (e.g., new_data.xlsx): ").strip()

# 2. Check if the file exists
if not os.path.exists(filename):
    print(f"❌ The file '{filename}' was not found!")
```

```
    exit()
# 3. Load the new data
df_new = pd.read_excel(filename)
print(f"✅ File loaded: {filename}")

#   Drop the 'Date_Time' column if it exists
if "Date_Time" in df_new.columns:
    df_new.drop(columns=["Date_Time"], inplace=True)
    print("🧽 Dropped column: 'Date_Time'")
else:
    print("ℹ️ Column 'Date_Time' not found — nothing dropped.")

#  Convert String labels to Numerical Values
#df_new["Label"] = df_new["Label"].map({"Night": 0, "Cloudy": 1, "Sunny": 2})
df_new['Temperature'] = df_new['Temperature'].fillna(df_new['Temperature'].mean())
df_new['Humidity'] = df_new['Humidity'].fillna(df_new['Humidity'].mean())
print(df_new.isnull().sum())

# 4. Keep the original label column for later comparison
if "Label" in df_new.columns:
    actual_labels = df_new["Label"]
    df_features = df_new.drop(columns=["Label"])
else:
    actual_labels = None
    df_features = df_new.copy()

# 5. Load the trained SVM model and scaler
model = joblib.load("svm_model.pkl")
scaler = joblib.load("scaler.pkl")

# 6. Apply the same scaler used in training
X_new_scaled = scaler.transform(df_features)

# 7. Make predictions
predictions = model.predict(X_new_scaled)
df_new["predicted_label"] = predictions

# 8. Map numeric predictions to readable labels
label_map = {0: "Night", 1: "Cloudy", 2: "Sunny"}
df_new["predicted_label_name"] = df_new["predicted_label"].map(label_map)

# 9. Save the results to an Excel file
output_file = "predicted_results_svm.xlsx"
df_new.to_excel(output_file, index=False)
print(f"📄 Predictions saved to: {output_file}")

from sklearn.metrics import accuracy_score

# 10. Compare predictions with actual labels
if actual_labels is not None:
    # Convert actual labels to numeric for comparison
    actual_labels_mapped = actual_labels.map({"Night": 0, "Cloudy": 1, "Sunny": 2})
    df_new["actual_label_mapped"] = actual_labels_mapped

    # Compute accuracy
    accuracy = accuracy_score(actual_labels_mapped, df_new["predicted_label"])
    print(f"📈 Prediction Accuracy on Test File: {accuracy * 100:.2f}%")
```

```
else:
    print("⚠️ No actual labels found for comparison.")
```

Appendix D: Real-time Monitoring Codes

```
import requests
import joblib
import time
import pandas as pd

# --- ThingSpeak settings ---
CHANNEL_ID = "2971883"
API_KEY = ""  # Leave empty if the channel is public
URL = f"https://api.thingspeak.com/channels/{CHANNEL_ID}/feeds/last.json"

# --- Telegram settings ---
BOT_TOKEN = "7542991347:AAEm33TeYhQUHBCOLstVglb2HuqlSwJhhAU"
CHAT_ID = "5089382659"
TELEGRAM_URL = f"https://api.telegram.org/bot{BOT_TOKEN}/sendMessage"

# --- Load the model ---
model = joblib.load("modelrandom.pkl")

FEATURE_NAMES = ["Temperature", "Humidity", "Light", "Voltage", "Current"]

def fetch_data():
    try:
        r = requests.get(URL, timeout=10)
        if r.status_code != 200:
            print("❌ Failed to fetch data – code", r.status_code)
            return None
        data = r.json()

        # Extract values from the fields
        values = []
        for i in range(1, 6):
            val = data.get(f"field{i}")
            if val is None:
                val = 0.0
            values.append(float(val))
        return values

    except Exception as e:
        print("⚠️ Error fetching data:", e)
        return None

def predict_rf(values):
    df = pd.DataFrame([values], columns=FEATURE_NAMES)
    prediction = model.predict(df)[0]
    return prediction

def label_to_text(label):
    return {0: "Night", 1: "Cloudy", 2: "Sunny"}.get(label, "Unknown")

def send_telegram_message(message):
    try:
        payload = {
```

```
            "chat_id": CHAT_ID,
            "text": message
        }
        headers = {
            "Content-Type": "application/json"
        }
        response = requests.post(TELEGRAM_URL, json=payload, headers=headers)
        if response.status_code != 200:
            print("❌ Failed to send Telegram message, status code:", response.status_code)
            print("Response:", response.text)
    except Exception as e:
        print("⚠️ Telegram exception:", e)

def main_loop():
    print(f"🚀 Started monitoring channel {CHANNEL_ID} using Random Forest every 15 seconds")

    while True:
        print("\n📡 Fetching and predicting…")
        data = fetch_data()
        if data is None:
            print("⚠️ Skipped prediction—no valid data")
        else:
            pred = predict_rf(data)
            label_text = label_to_text(pred)
            print(f"✅ Prediction: {label_text}")

            msg = (f"🌦️ RF Prediction:\n"
                   f"{label_text}\n"
                   f"Temperature: {data[0]:.2f}°C\n"
                   f"Humidity: {data[1]:.2f}%\n"
                   f"Light: {data[2]}\n"
                   f"Voltage: {data[3]:.2f} V\n"
                   f"Current: {data[4]:.2f} mA")
            send_telegram_message(msg)

        time.sleep(15)

if __name__ == "__main__":
    main_loop()
```

……..

```
import requests
import joblib
import time
import pandas as pd
# --- ThingSpeak Settings ---
CHANNEL_ID = "2971883"
API_KEY = ""  # Leave empty if the channel is public
URL = f"https://api.thingspeak.com/channels/{CHANNEL_ID}/feeds/last.json"
# --- Telegram Settings ---
BOT_TOKEN = "7745200751:AAGo9tG0pTyRHEvnBOUni_xc1dmSqGEdsg4"
CHAT_ID = "5870120331"
TELEGRAM_URL = f"https://api.telegram.org/bot{BOT_TOKEN}/sendMessage"
# --- Load the model and scaler ---
model = joblib.load("svm_model.pkl")
scaler = joblib.load("scaler.pkl")
```

```
FEATURE_NAMES = ["Temperature", "Humidity", "Light", "Voltage", "Current"]

def fetch_data():
    try:
        r = requests.get(URL, timeout=10)
        if r.status_code != 200:
            print("❌ Failed to fetch data – code", r.status_code)
            return None
        data = r.json()

        required_fields = [f"field{i}" for i in range(1, 6)]
        for f in required_fields:
            if f not in data:
                print(f"⚠️ Missing field: {f}")
                return None

        values = []
        for i in range(1, 6):
            val = data.get(f"field{i}")
            if val is None:
                val = 0.0
            values.append(float(val))
        return values

    except Exception as e:
        print("⚠️ Error fetching data:", e)
        return None

def predict_svm(values):
    df = pd.DataFrame([values], columns=FEATURE_NAMES)
    X_scaled = scaler.transform(df)
    prediction = model.predict(X_scaled)[0]
    return prediction

def label_to_text(label):
    return {0: "Night", 1: "Cloudy", 2: "Sunny"}.get(label, "Unknown")

def send_telegram_message(message):
    try:
        payload = {
            "chat_id": CHAT_ID,
            "text": message
        }
        headers = {
            "Content-Type": "application/json"
        }
        response = requests.post(TELEGRAM_URL, json=payload, headers=headers)
        if response.status_code != 200:
            print("❌ Failed to send Telegram message, status code:", response.status_code)
            print("Response:", response.text)
    except Exception as e:
        print("⚠️ Telegram exception:", e)

def main_loop():
    print(f"🚀 Started monitoring channel {CHANNEL_ID} using SVM every 15 seconds")
```

```
    while True:
        print("\n 📡 Fetching and predicting…")
        data = fetch_data()
        if data is None:
            print(" ⚠️ Skipped prediction—no valid data")
        else:
            pred = predict_svm(data)
            label_text = label_to_text(pred)
            print(f" ✅ Prediction: {pred} ({label_text})")

            msg = (f" 🌤️ SVM Prediction:\n"
                   f"<label>{label_text}</label>\n"
                   f"Data:\n"
                   f"Temperature: {data[0]:.2f}°C\n"
                   f"Humidity: {data[1]:.2f}%\n"
                   f"Light: {data[2]}\n"
                   f"Voltage: {data[3]:.2f} V\n"
                   f"Current: {data[4]:.2f} mA")
            send_telegram_message(msg)

        time.sleep(15)

if __name__ == "__main__":
    main_loop()
```