

# Internet of Things Fundamentals

## *Subject Project*

---

BS AI 6<sup>th</sup> Semester SP-25 (AIE-3079)

Date:

**Project Title:**

Automated Solar Panel Cleaning System

**Group Name/no.:**

TresMentis

**Team Members:**

*Members*

Registration no

Name

Signature

<b><i>Member-1 (Leader)</i></b>	<b>22-NTU-CS-1374</b>	<b>Subaina Norab</b>	
<b><i>Member-2</i></b>	<b>22-NTU-CS-1373</b>	<b>Shaham Hijab</b>	
<b><i>Member-3</i></b>	<b>22-NTU-CS-1343</b>	<b>Hadia Alvi</b>	
<b><i>Member-4</i></b>			

Contributions in % of each Team Members for each component					
		Member-1	Member-2	Member-3	Member-4
Distribution Components		Subaina	Shaham	Hadia	Name
Coding	ESP32-coding	50	25	25	
	Python Coding	20	40	40	
UI Design		30	30	40	
Database		40	30	30	
Cloud Integration		25	25	50	
IoT Gateway		30	40	30	
Edge Processing		30	40	30	
Documentation		30	30	40	
Presentation Design		35	30	35	
Hardware Integration		25	50	25	
Hardware Debugging		40	30	30	
Dashboard(Graphs)		40	30	30	
Firebase connection		30	30	35	

*Tobefilledbytheevaluator*

### Team-Based Evaluation (60 Marks)

Criteria	Obtained Marks	Out of
System Design & Architecture		10
Hardware Integration & Circuit Setup		10
IoT Gateway and Cloud Communication		10
Working Prototype Demonstration		10
Performance & Reliability Testing		10
Presentation		10
Total (Team-Based)		60

### Individual-Based Evaluation (40 Marks per Member)

	Member 1	Member 2	Member 3	Member 4
Criteria				
Understanding of the Project & Role	/10	/10	/10	/10
Code Contribution and Explanation	/10	/10	/10	/10
Q/A VIVA	/10	/10	/10	/10
Documentation/Reporting & Communication	/10	/10	/10	/10
Total (Individual-Based)	/40	/40	/40	/40
Total Overall (60+40)	/100	/100	/100	/100
Weightage Lab Grade (50)				

# 1. Abstract / Executive Summary

Automated Solar Panel Cleaning System is designed to maintain efficiency by detecting dust and triggering a cleaning mechanism. Using sensors and an ESP32 controller, the system monitors dust levels and power output in real-time. When thresholds are crossed, it activates a water spray to clean the panel. Data is sent to the cloud for monitoring, and a machine learning model predicts future cleaning needs. This solution offers smart, safe, and remote maintenance for solar installations.

# 2. Table of Contents

1. Abstract / Executive Summary ..... 3

2. Table of Contents ..... 3

3. Introduction ..... 5

3.1. Background & Motivation ..... 5

Solar panels suffer significant efficiency loss due to dust accumulation, especially in remote or industrial areas. Manual cleaning is labor-intensive and unsafe. Automating this process with IoT and ML can improve energy output, reduce maintenance costs, and enhance safety. .... 5

3.2. Problem Statement ..... 5

Dust on solar panels leads to reduced efficiency and power generation. Manual cleaning is not scalable and poses risks in large or hard-to-access installations. .... 5

3.3. Project Goals ..... 5

4. Literature Review (Optional) ..... 6

4.1. Relevant IoT/ESP32 Concepts ..... 6

4.2. Similar Projects / Research ..... 6

5. Methodology / System Design ..... 6

5.1 Hardware Components ..... 6

5.2 Circuit diagram (Fritzing/proteus/other) with labels ..... 7

5.3 Software Design ..... 8

Flowchart/system architecture ..... 8

6. Implementation ..... 9

6.1. Step-by-step setup (wiring, configurations) ..... 9

6.1.1. Wiring ..... 9

6.2. Code snippets (with comments) ..... 10

.....	18
6.3.Challenges & Solutions.....	18
7. Results & Discussion .....	19
7.1.Screenshots/output (e.g., sensor data on Serial Monitor, MQTT logs).....	19
7.3.Comparison with Expectations .....	21
8. Testing & Validation / Limitations.....	22
8.1.Test Cases.....	22
8.2.Limitations .....	22
9. Conclusion & Future Work.....	22
9.1.Key Takeaways.....	22
9.2.Potential Improvements .....	22
10. References .....	22
11. Links .....	23

1. Abstract / Executive Summary
2. Table of Contents
3. Introduction
4. Literature Review
5. Methodology / System Design
6. Implementation
7. Results & Discussion
8. Testing & Validation
9. Conclusion & Future Work
10. References
11. Links

## **3. Introduction**

### **3.1. Background & Motivation**

Solar panels suffer significant efficiency loss due to dust accumulation, especially in remote or industrial areas. Manual cleaning is labor-intensive and unsafe. Automating this process with IoT and ML can improve energy output, reduce maintenance costs, and enhance safety.

### **3.2. Problem Statement**

Dust on solar panels leads to reduced efficiency and power generation. Manual cleaning is not scalable and poses risks in large or hard-to-access installations.

### **3.3. Project Goals**

- Detect dust on solar panels using sensors
- Measure voltage and current to monitor efficiency
- Trigger a cleaning mechanism automatically
- Enable remote monitoring via cloud dashboards
- Use ML to predict optimal cleaning

## 4. Literature Review (Optional)

### 4.1.Relevant IoT/ESP32 Concepts

The ESP32-S3 is a powerful microcontroller with integrated Wi-Fi, ideal for IoT applications. It supports various sensor interfaces and cloud integration. Dust sensors (GP2Y1010AU0F), power sensors (INA219), and solenoid-based actuators are commonly used in solar maintenance solutions.

### 4.2.Similar Projects / Research

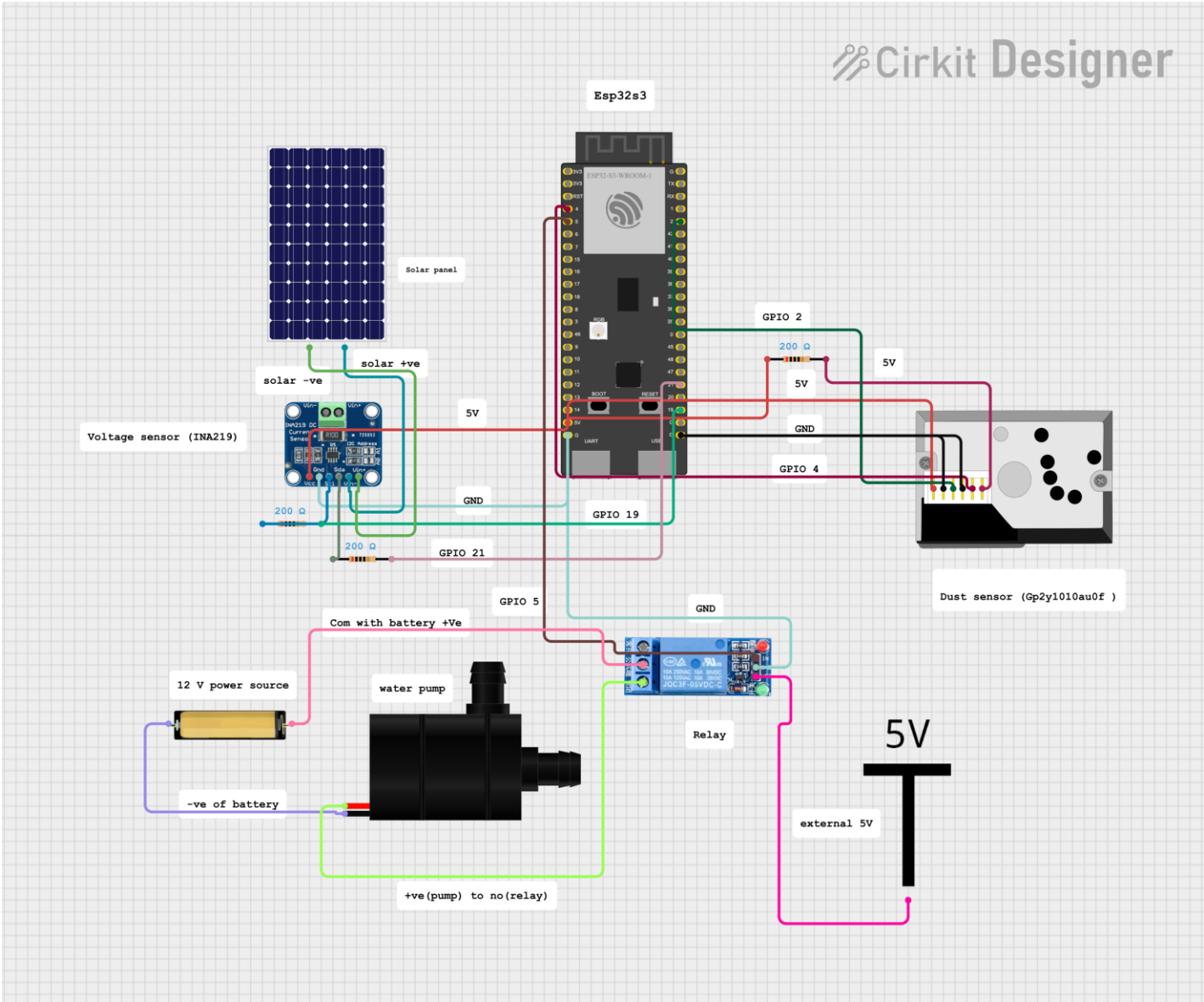
- IoT-based smart solar panel cleaning using Arduino and ThingSpeak
- AI-based predictive maintenance for solar farms
- Cloud dashboards (Blynk, Firebase) in remote IoT monitoring

## 5. Methodology / System Design

### 5.1 Hardware Components

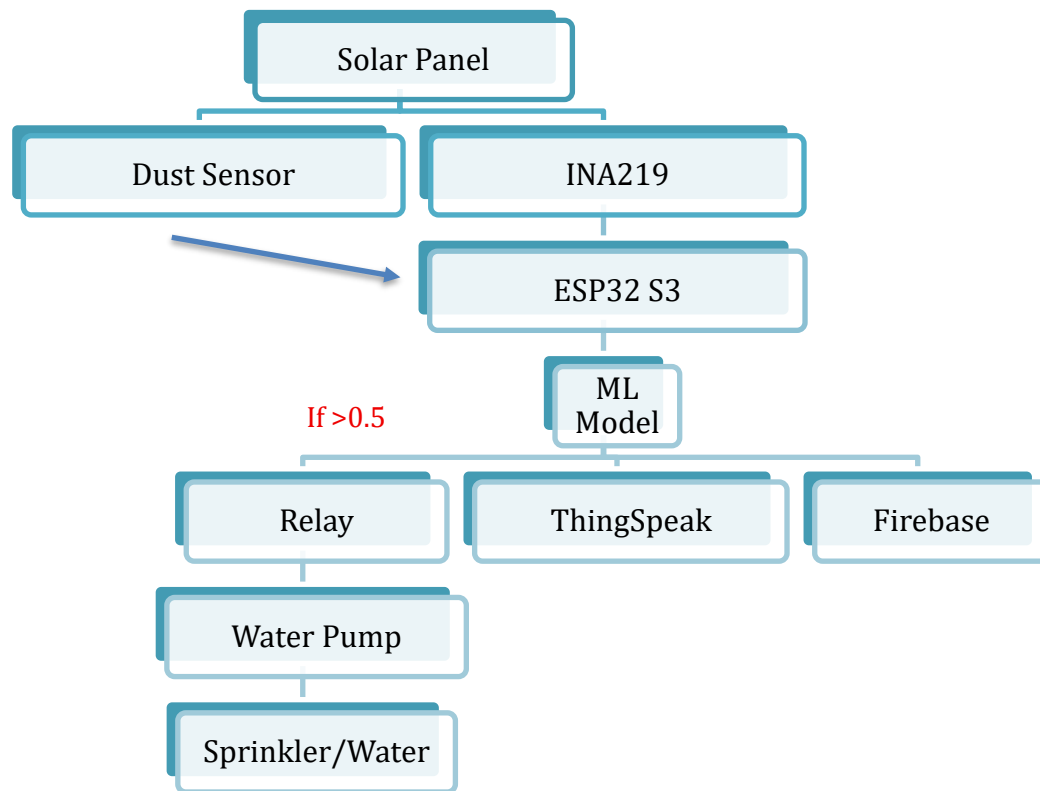
Component	Purpose
ESP32-S3	Central controller with Wi-Fi for IoT connectivity
INA219 DC Power Sensor (GY-219)	Measures voltage, current, and power (efficiency)
GP2Y1010AU0F Dust Sensor	Detects dust level on the solar panel
24V DC Solenoid Valve (1/4 inch)	Controls water flow for cleaning
5V Relay Module (1-channel)	Triggers solenoid valve using ESP32
Jumper Wires, Breadboard	Wiring and prototyping
24V DC Power Supply	Powers the solenoid valve
Sprinkler	To spray water

5.2Circuit diagram (Fritzing/proteus/other) with labels



## 5.3 Software Design

### Flowchart/system architecture



### Libraries/tools used (Arduino IDE, PlatformIO, MQTT, etc.)

Tool/Platform	Purpose
Arduino IDE / Thonny	Microcontroller programming and uploading
Firebase	Stores data
Google Colab / Python	ML model training and data visualization
ThingSpeak	Cloud-based IoT platform to display live data in charts and graphs for quick monitoring
HTTP	



## 6. Implementation

### 6.1.Step-by-step setup (wiring, configurations)

#### 6.1.1. Wiring

##### Dust sensor

<b>VCC</b>	ESP 5V through resistor
<b>V0</b>	GPIO 4
<b>S-GND</b>	ESP GND
<b>LED</b>	GPIO 2
<b>LED-GND</b>	ESP GND
<b>V-LED</b>	ESP 5V

##### INA219 Voltage sensor

<b>VCC</b>	ESP 5V
<b>GND</b>	ESP GND
<b>SCL</b>	GPIO 19 through resistor
<b>SDA</b>	GPIO 21 through resistor
<b>VIN -</b>	Solar panel -ve terminal
<b>VIN +</b>	Solar panel +ve terminal

##### Relay

<b>VCC</b>	External 5V
<b>GND</b>	ESP GND
<b>IN</b>	GPIO 5
<b>COM</b>	+ve of 12 V power source
<b>NO</b>	+ve terminal of water pump

##### Water pump

<b>+ve terminal</b>	NO of relay
<b>-ve terminal</b>	-ve of 12 V power source

## 6.2.Code snippets (with comments)

```
# Split data
Xtrain, Xtest, ytrain, ytest = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# logistic regression as 1-layer NN with sigmoid
model = Sequential([
    Dense(1, activation='sigmoid', input_shape=(2,))
])

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

model.fit(Xtrain, ytrain, epochs=100, batch_size=16, verbose=0)

loss, acc = model.evaluate(Xtest, ytest, verbose=0)
print("TF Logistic Regression Accuracy:", acc)

# Save the model
model.export('logistic_model_savedmodel')
```

Training logistic regression with dust and voltage value to predict whether solar panel is clean or not. It is then saved.

```
/ Users / Subana / OneDrive - Higher Education Commission / Documents / Academic / 6th semester / Automated-solar-
1  import tensorflow as tf
2
3  converter = tf.lite.TFLiteConverter.from_saved_model('logistic_model_savedmodel')
4  tflite_model = converter.convert()
5
6  with open('logistic_model.tflite', 'wb') as f:
7      f.write(tflite_model)
8
9  print("TFLite model saved as 'logistic_model.tflite'")
10
```

Saving model as tflite

```
def bin_to_c_array(input_file, output_file, array_name="model_tflite"):
    with open(input_file, "rb") as f:
        data = f.read()

    with open(output_file, "w") as f:
        f.write(f"// Converted from {input_file} to C array\n")
        f.write(f"const unsigned char {array_name}[] = {{\n")

        for i in range(0, len(data), 12):
            chunk = data[i:i+12]
            line = ", ".join(f"0x{b:02x}" for b in chunk)
            if i + 12 >= len(data): # last line, remove trailing comma
                line = line.rstrip(",")
            f.write(f"    {line},\n")

        f.write("};\n")
        f.write(f"const unsigned int {array_name}_len = {len(data)};\n")
        print(f"C array written to '{output_file}' with {len(data)} bytes.")

# Run it
bin_to_c_array("logistic_model.tflite", "logistic_model.h")
```

Now tflite is converted into C array to deploy on esp32 s3

```
// Variables to store the latest sensor values
float lastDust = 0.0;
float lastVoltage = 0.0;
float lastProbability = 0.0;
String lastPrediction = "Unknown";

// converted model
#include "logistic_model.h" // Should define 'model_tflite'
```

Variables to store values of dust,voltage,probability and decision to send it to firebase and thingspeak.

Also , added trained logistic model.

```
// ===== Pins & Constants =====
const int dustLEDPin = 2;
const int dustAnalogPin = 4;
const int RELAY_PIN = 5;      // Relay control pin
const int SDA_PIN = 21;
const int SCL_PIN = 19;

#define DUST_MEAN 2.6
#define DUST_STD 1.6
#define VOLT_MEAN 17.7
#define VOLT_STD 1.2
```

Defined pins for sensors and mean values of dust and voltage

```
// ===== TensorFlow Lite Micro Setup =====
constexpr int kTensorArenaSize = 10 * 1024; // Increase if needed
uint8_t* tensor_arena = (uint8_t*) heap_caps_malloc(kTensorArenaSize, MALLOC_CAP_SPIRAM);

tf::MicroInterpreter* interpreter;
TfLiteTensor* input;
TfLiteTensor* output;
```

This code sets up TensorFlow Lite Micro to use external PSRAM on the ESP32-S3 by allocating a 10KB memory block (tensor\_arena) from PSRAM. This memory is used to store tensors and run the ML model. It also prepares pointers for the interpreter and input/output tensors needed for inference.

```

void setup() {
  Serial.begin(115200);
  delay(1000);

  pinMode(RELAY_PIN, OUTPUT);
  digitalWrite(RELAY_PIN, RELAY_OFF); // Motor OFF at startup

  connectWiFi();

  // Initialize I2C and INA219
  Wire.begin(SDA_PIN, SCL_PIN);
  delay(500);
  if (!ina219.begin(&Wire)) {
    Serial.println("INA219 not found!");
    while (1);
  }
  if (!tensor_arena) {
    Serial.println("Failed to allocate tensor arena in PSRAM!");
    while (1);
  }

  // Load ML model

```

Configuring relay and INA219, then checking is it found and also checking psram allocation

```

// Load ML model
const tflite::Model* model = tflite::GetModel(model_tflite);
if (model->version() != TFLITE_SCHEMA_VERSION) {
    Serial.println("Model version mismatch!");
    while (1);
}
static tflite::MicroMutableOpResolver<6> resolver;
resolver.AddFullyConnected();
resolver.AddLogistic();
resolver.AddReshape();
resolver.AddQuantize();
resolver.AddDequantize();
static tflite::MicroInterpreter static_interpreter(
    model, resolver, tensor_arena, kTensorArenaSize);
interpreter = &static_interpreter;
if (interpreter->AllocateTensors() != kTfLiteOk) {
    Serial.println("Tensor allocation failed!");
    while (1);
}
input = interpreter->input(0);
output = interpreter->output(0);
Serial.println("System ready!");
}

```

It loads a TensorFlow Lite model, checks its version, registers needed operations, creates an interpreter using PSRAM memory, allocates tensors, and sets up input/output tensors. If successful, it prints "System ready!".

```
/ float readDustSensor() {  
    digitalWrite(dustLEDPin, LOW);  
    delayMicroseconds(280);  
    int rawADC = analogRead(dustAnalogPin);  
    delayMicroseconds(40);  
    digitalWrite(dustLEDPin, HIGH);  
    delayMicroseconds(9680);  
  
    float voltage = rawADC * (3.3 / 4095.0);  
    float dustDensity = 0.17 * voltage - 0.1;  
    if (dustDensity < 0) dustDensity = 0;  
    return dustDensity;  
}
```

This function reads dust concentration by briefly turning on the sensor's LED, measuring the analog voltage, converting it to dust density using a formula, and returning the result in  $\text{mg/m}^3$

```

void loop() {
    float dust = readDustSensor();

    // INA219 readings
    float busVoltage = ina219.getBusVoltage_V();
    float shuntVoltage = ina219.getShuntVoltage_mV() / 1000.0;
    float loadVoltage = busVoltage + shuntVoltage;

    // Normalize inputs
    float normDust = (dust - DUST_MEAN) / DUST_STD;
    float normVolt = (loadVoltage - VOLT_MEAN) / VOLT_STD;

    input->data.f[0] = normDust;
    input->data.f[1] = normVolt;

    if (interpreter->Invoke() != kTfLiteOk) {
        Serial.println("Inference failed!");
        return;
    }

    float prob = output->data.f[0];
    const char* result = (prob < 0.5) ? "Needs_cleaning" : "Clean";
}

```

It reads dust and voltage values, normalizes them using predefined mean and standard deviation, and feeds them into the TensorFlow Lite model for inference. It then runs the model and interprets the output probability to determine if the solar panel "Needs\_cleaning" or is "Clean".

```

    if (prob < 0.5) {
        Serial.println("Triggering solenoid for cleaning...");
        digitalWrite(RELAY_PIN, RELAY_ON);
        delay(10000);
        digitalWrite(RELAY_PIN, RELAY_OFF);
        Serial.println("Cleaning complete.");
    }
}

```

If probability is less than 0.5 , it will trigger relay to clean solar panel



```

| // Send to Firebase
if (WiFi.status() == WL_CONNECTED) {
    HTTPClient http;

    // Firebase URL (change YOUR_PROJECT_ID)
    String firebaseUrl = String("https://solarsystem-2babe-default-rtdb.firebaseio.com//solarData.json?auth=ACOZE3BvabpPdNGuu83DAyVm2NkR1EzUg3bPgZWn");

    // Format payload as JSON
    String payload = "{";
    payload += "\"dustDensity\":" + String(dust) + ",";
    payload += "\"voltage\":" + String(loadVoltage) + ",";
    payload += "\"probability\":" + String(prob) + ",";
    payload += "\"prediction\":" + String(result) + "\"";
    payload += "}";

    // Start connection
    http.begin(firebaseUrl);
    http.addHeader("Content-Type", "application/json");

    // Send POST
    int responseCode = http.POST(payload);
    Serial.print("Firebase Response: ");
    Serial.println(responseCode);

    if (responseCode > 0) {
        Serial.println("Data sent successfully!");
    } else {
        Serial.print("Error sending to Firebase: ");
        Serial.println(http.errorToString(responseCode));
    }

    http.end();
} else {
    Serial.println("WiFi not connected.");
}

```



If Wi-Fi is connected, then sends the dust, voltage, model probability, and prediction result to a Firebase Realtime Database using an HTTP POST request with a JSON payload. It prints the Firebase response code and confirms whether the data was sent successfully or not.

```

// ===== Send to ThingSpeak =====
if (WiFi.status() == WL_CONNECTED) {
  HTTPClient http;

  String thingSpeakAPIKey = "UFJX6PHYTC441XUE"; // Your Write API key

  // Convert prediction to numeric: 1 = Clean, 0 = Needs_cleaning
  int predictionCode = (String(result) == "Clean") ? 1 : 0;

  String url = "http://api.thingspeak.com/update?api_key=" + thingSpeakAPIKey;
  url += "&field1=" + String(dust, 2);
  url += "&field2=" + String(loadVoltage, 2);
  url += "&field3=" + String(prob, 4);
  url += "&field4=" + String(predictionCode); // Send numeric value now

  http.begin(url);
  int httpCode = http.GET();

  Serial.print("ThingSpeak Response: ");
  Serial.println(httpCode);

  if (httpCode > 0) {
    Serial.println("ThingSpeak update success!");
  } else {
    Serial.println("ThingSpeak update failed.");
  }

  http.end();
}

```

It sends data to ThingSpeak if Wi-Fi is connected. It formats the dust density, voltage, prediction probability, and a numeric prediction (1 = Clean, 0 = Needs\_cleaning) into a URL using your API key, then sends it via an HTTP GET request. It prints the HTTP response and whether the update was successful.

## 6.3.Challenges & Solutions

- **INA219 Not Reading Values**

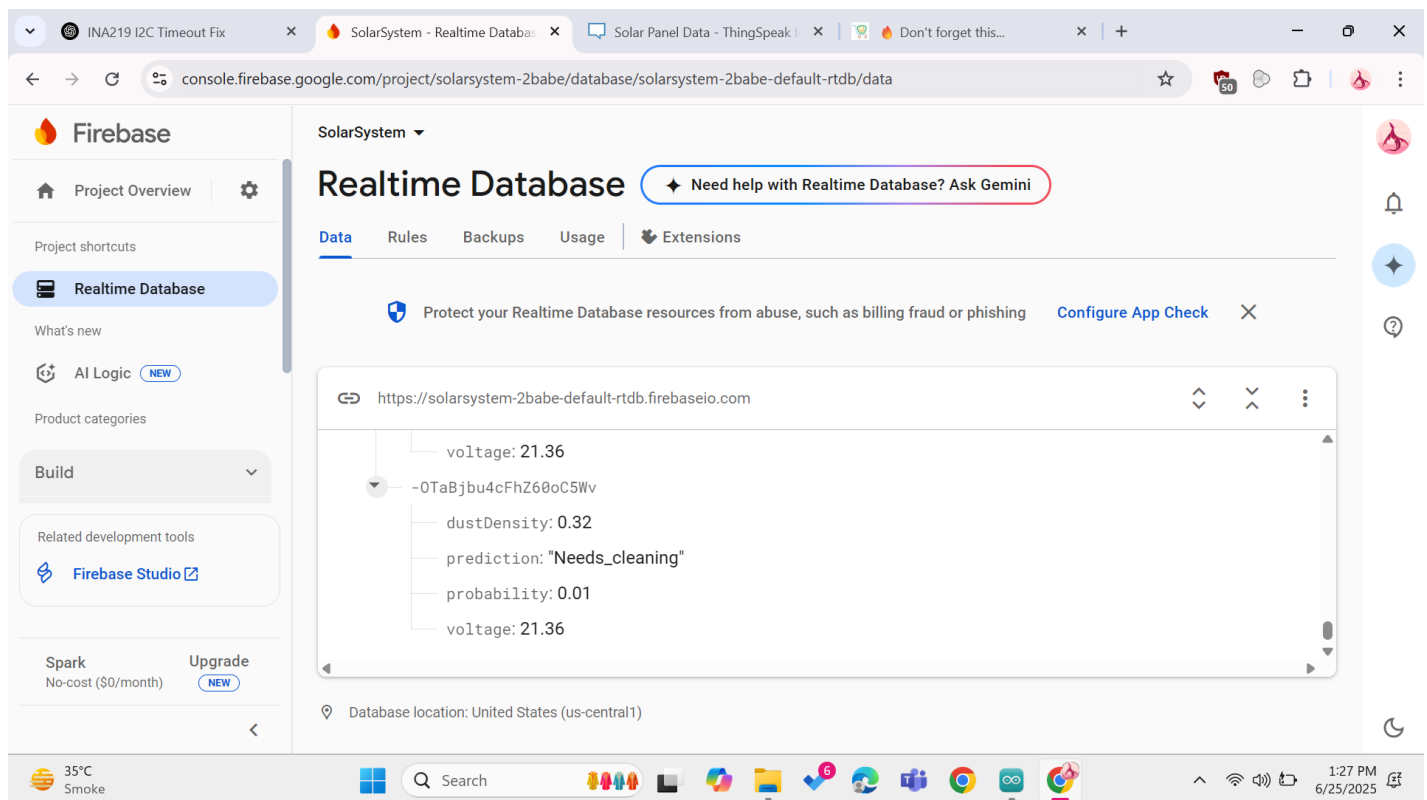
Challenge: The INA219 sensor initially failed to return voltage/current readings.

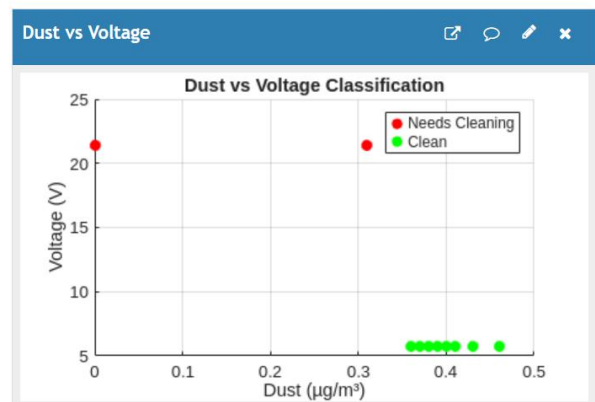
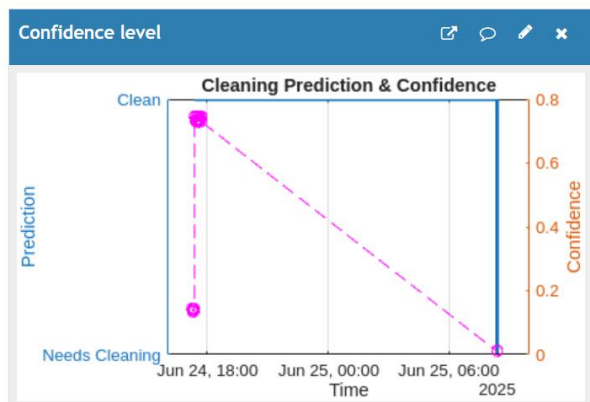
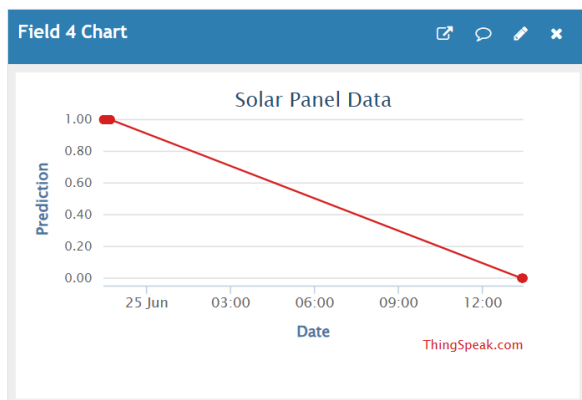
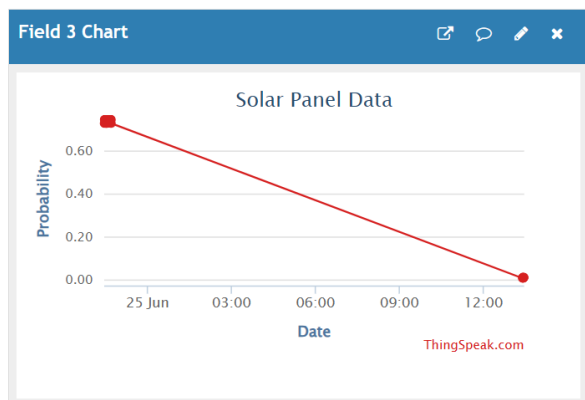
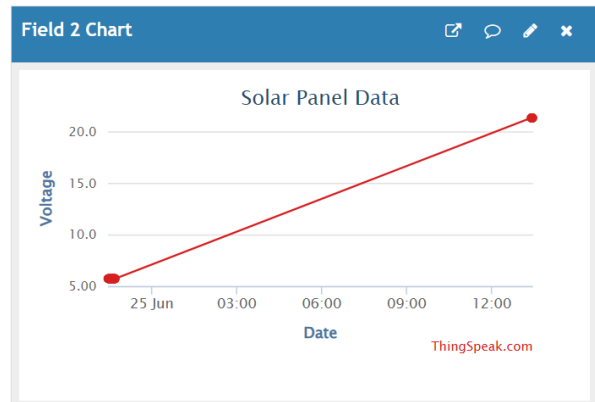
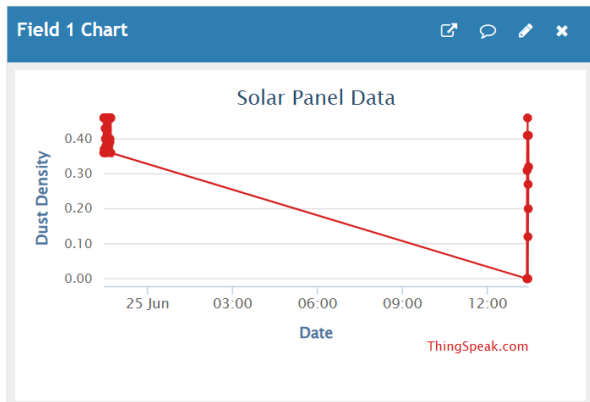
Solution: External pull-up resistors were added to the I2C lines (SDA and SCL), which stabilized communication between the sensor and ESP32.

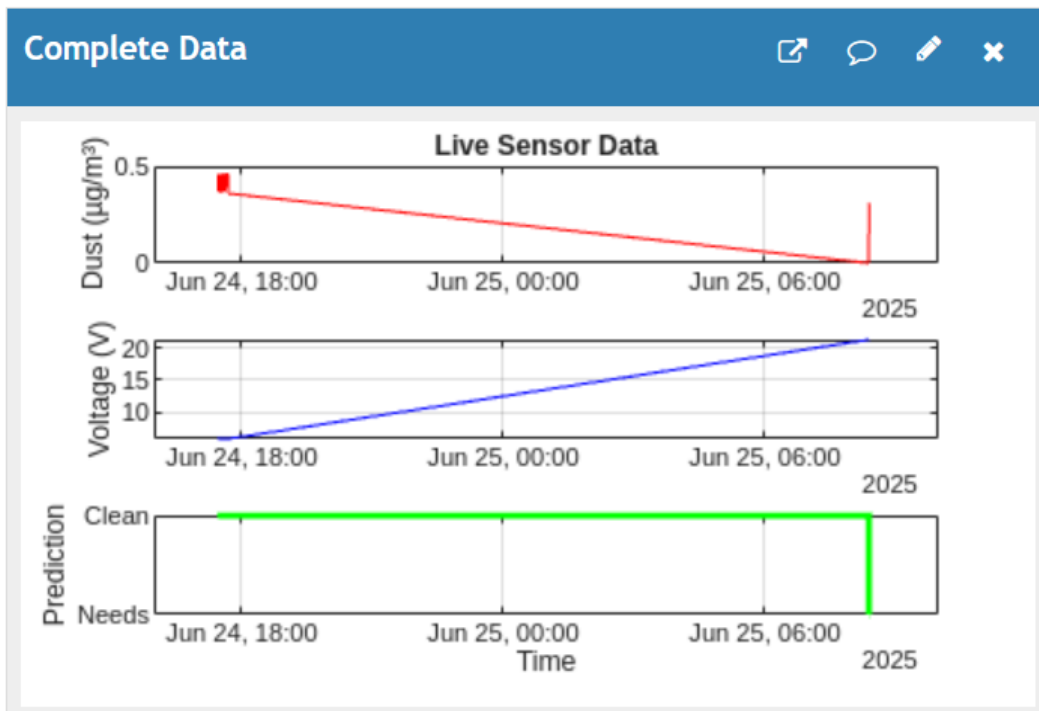
## 7. Results & Discussion

### 7.1. Screenshots/output (e.g., sensor data on Serial Monitor, MQTT logs)

```
Dust: 0.00 µg/m³, Voltage: 21.36 V
Predicted probability: 0.01
Prediction: Needs_cleaning
Triggering solenoid for cleaning...
Cleaning complete.
Firebase Response: 200
Data sent successfully!
ThingSpeak Response: 200
ThingSpeak update success!
-----
```







## 7.2. Performance analysis (accuracy, latency, reliability)

the appropriate compiler flags.

TF Logistic Regression Accuracy: 0.9666666388511658

Latency without cleaning: ~1.6 seconds

Latency with cleaning: ~18.6 seconds

**Reliability:** Good — the system reads sensors, performs ML inference, and sends data reliably under normal conditions.

## 7.3. Comparison with Expectations

- System responded well to both efficiency drops and high dust levels
- Cloud and local data were synced accurately
- ML model successfully predicted cleaning need in test runs

## **8. Testing & Validation / Limitations**

### **8.1. Test Cases**

- Dust level manually increased → System triggered cleaning
- Artificial light intensity variations → System ignored unless efficiency dropped

### **8.2. Limitations**

- Cleaning relies on water, not suitable in water-scarce areas
- ML model needs more real training data for accuracy

## **9. Conclusion & Future Work**

### **9.1. Key Takeaways**

An IoT-enabled solution developed to automate solar panel cleaning using real-time sensor data. The system improves performance and reduces the need for manual intervention.

### **9.2. Potential Improvements**

- Integrate solar-powered water pump
- Expand ML model with current, accuracy and weather
- Image based cleaning system
- A Solar Tracking (Rotating) System

## **10. References**

<https://www.mdpi.com/1996-1073/16/24/7960>

[https://www.erpublications.com/uploaded\\_files/download/prof-a-sankpal-miss-g-priyanka-miss-b-asha-miss-s-rutuja\\_fcWvv.pdf](https://www.erpublications.com/uploaded_files/download/prof-a-sankpal-miss-g-priyanka-miss-b-asha-miss-s-rutuja_fcWvv.pdf)

## 11. Links

GitHub Repository Link (links from each member)

Video Demo (embedded link or QR code / optional with Bonus)