A Technical Report on

"WEATHER FORECASTING AND AUTOMATIC WATERING SYSTEM IN ESP32 USING MICROPYTHON "

From

[PROJECT STAGE-II (EC782)]

For partial fulfillment of the requirements for the B.TECH

Under

MAULANA ABUL KALAM AZAD UNIVERSITY OF TECHNOLOGY

(Formerly known as West Bengal University of Technology)

Submitted by:

Sandip Roy(10200322057)

Anamika Konra(10200322061)

Sumona Sultana Ahmed(10200321025)

Arpita Sarkar(10200321032)

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

YEAR: 4th

SEMESTER: 8th

Under The Guidance Of

PROF. ANUP KUMAR MALLICK (Assistant Professor, Dept. Of ECE, KGEC)

KALYANI GOVERNMENT ENGINEERING COLLEGE,

KALYANI , NADIA, WEST BENGAL-741235

CERTIFICATE FROM THE EXAMINERS

This is to certify that **Sandip Roy (10200322057), Anamika Konra (10200322061), Sumona Sultana Ahmed (10200321025) and Arpita Sarkar (10200321032)** have successfully completed their Bachelor Of Technology final year project on **"*WEATHER FORECASTING AND AUTOMATIC WEATHERING SYSTEM IN ESP32 USING MICROPYTHON*"**. They have defended their project with Powerpoint presentation in front of panel of faculties of Electronics and Communication Engineering department of Kalyani Government Engineering College affiliated to Maulana Abul Kalam Azad University of Technology.

Date:

Signature of Examiner

Prof. Anup Kumar Mallick

(Assistant Professor)

Dept. Of ECE,KGEC

ACKNOWLEDGEMENTS

We would like to thank our project supervisor, Prof. Anup Kumar Mallick Sir, for providing us with the right balance of guidance and independence in our research. We are greatly indebted to him for his support, constant encouragement and advice both in technical and non-technical matters. His broad expertise and superb intuition have been a source of inspiration to us. His comments and criticisms have greatly influenced our technical writing and reflected throughout the presentation of this dissertation.

We are grateful to Prof. Dr. Angshuman Sarkar Sir, HOD of Electronics and Communication Engineering department, Kalyani Government Engineering College. During our project tenure, we had the great fortune and honor to discuss with him the problems of common interest. We are thankful to our team members for their co-operation and help to complete this project.

Sandip Roy
Roll no. – 10200322057
Reg. no. – 221020120311

Anamika Konra
Roll no. – 10200322061
Reg. no. – 221020120292

Sumona Sultana Ahmed
Roll no. – 10200321025
Reg. no. - 211020100310001

Arpita Sarkar
Roll no. – 10200321032
Reg. no. – 211020100310030

ABSTRACT

This project presents the design and implementation of a **Weather Forecasting and Automatic Watering System in ESP32 using MicroPython**. With climate awareness and automation becoming increasingly important, the system aims to monitor and analyze real-time weather parameters to provide basic weather predictions and automatic environmental responses.

The core component of the system is the **ESP32**, a low-cost and Wi-Fi-enabled microcontroller capable of handling multiple sensor inputs and performing local processing. The system integrates a combination of sensors:

- **DHT11**: for precise temperature and humidity measurements
- **BMP180**: for barometric pressure and altitude tracking
- **Soil moisture sensor**: for detecting the water content in soil.
- **Rain sensor and LDR**: for detecting rainfall or light intensity.
- **LCD Display**: used for display the weather forecasting value.
- **I2C Module**: used to simplify communication and reduce wiring complexity.
- **Motor driver with mini water pump**: used for automatic watering system.

The gathered data is processed using logic programmed in MicroPython. Threshold-based analysis and simple algorithms allow the system to make intelligent decisions, such as predicting rain (based on high humidity and dropping pressure), identifying heatwaves, or triggering warnings in case of extreme temperature changes. The ESP32 can also control connected actuators—like a fan, water cover, or alarm—based on the forecasted or real-time data.

The system's modular nature allows it to function independently as an edge device, and it can be easily extended to communicate with online weather APIs or cloud platforms via Wi-Fi. The forecast and sensor data can also be displayed on an LED screen or accessed through a web dashboard.

This project proves to be a practical solution for smart agriculture, automated greenhouses, smart homes, and mini weather stations. It emphasizes the value of combining embedded systems with environmental sensing for intelligent, low-cost, and real-time automation.

TABLE OF CONTENTS

Chapter 1: Introduction

1.1) Background of the Project:

In today's technology-driven era, real-time data collection and automated systems have become the backbone of intelligent infrastructure, especially in areas like agriculture, environment monitoring, disaster management, and smart cities. Weather has always played a significant role in shaping human activities. Traditionally, meteorological data was collected using large-scale government setups, which are often expensive, static, and centralized. However, as global climate change continues to intensify, there is an urgent demand for **low-cost, decentralized, and weather monitoring and automatic watering systems** that can provide accurate and real-time data—even in remote or resource-constrained locations.

This project, titled **"Weather Forecasting and Automatic Weathering System in ESP32 using MicroPython,"** aims to build a smart and efficient weather station capable of both **monitoring atmospheric conditions** and **automating responses** based on real-time data. The system monitors environmental parameters such as **temperature, humidity, pressure, rainfall, and light intensity**, and responds to specific triggers by activating devices like **fans, covers, or alarms**. This combination of forecasting and automatic response enables improved decision-making in scenarios like protecting crops during rainfall, controlling greenhouse temperature, or activating safety systems during storms.

Unlike conventional systems that rely heavily on cloud computing or internet connectivity, our system emphasizes **edge-level automation**. This means that the ESP32 microcontroller processes sensor data locally and initiates appropriate actions without needing continuous access to the internet. The use of **MicroPython**, a lightweight version of Python specifically built for microcontrollers, further simplifies coding and allows easier maintenance and upgrades, especially for students, researchers, and developers.

❖ ESP32 Microcontroller:

The **ESP32** is a powerful and cost-effective microcontroller developed by Espressif Systems, widely used in IoT (Internet of Things) projects due to its advanced features. Unlike simpler boards like Arduino Uno, ESP32 offers a more robust platform with:

- Dual-core Tensilica processor up to 240 MHz
- Wi-Fi and Bluetooth connectivity
- Multiple GPIOs and ADCs/DACs
- PWM, SPI, I2C, UART communication protocols
- Sleep and deep sleep modes for low power usage

Its compact size, low power consumption, and built-in wireless capabilities make it ideal for portable, outdoor, or embedded systems. In this project, the ESP32 serves as the **processing hub**, reading values from sensors (DHT11 for temperature/humidity, BMP180 for pressure, rain sensor, soil moisture sensor, and LDR for light) and taking real-time decisions like:

- Turning on/off the water pump

Moreover, the ESP32 supports **MicroPython**, making it suitable for rapid development, especially in educational and prototyping environments.

❖ MicroPython and Python:

Python is known globally for its **simplicity, clarity, and readability**, making it ideal for beginners and professionals alike. While traditional microcontroller programming uses **C or C++**, these languages often have a steep learning curve, require more verbose code, and are less flexible for quick debugging or updates.

MicroPython is a compact implementation of Python 3 designed to run directly on microcontrollers like ESP32. It includes modules for hardware control (e.g., machine, utime) and supports interactive programming through a **REPL (Read-Eval-Print Loop)**, which allows developers to run commands in real-time without flashing firmware repeatedly.

Benefits of MicroPython in this project:

- **Faster development**: Less boilerplate code and easier syntax.
- **Better debugging**: Interactive terminal support for testing each module.
- **Portability**: Easy to migrate code across various boards (ESP8266, Raspberry Pi Pico, etc.).
- **Community support**: Growing libraries and documentation for IoT tasks.

Example: Turning on a fan when temperature > 35°C can be done in 4 lines of readable code in Micropython versus 20+ lines in C.

This ease of development is especially helpful in student projects, educational labs, or rapid prototyping environments where **time and learning curve** are major concerns.

1.2) Objectives of the Project:

- To develop a smart and self-sufficient weather-responsive system using the ESP32 microcontroller programmed in MicroPython, capable of independently monitoring environmental conditions and making intelligent decisions without relying on continuous external computation.
- To implement a lightweight, embedded machine learning prediction model on the ESP32, using only essential values—mean, scale, coefficients, and intercept—extracted from a logistic regression model trained in Python (Google Colab) for predicting the likelihood of rainfall.
- To eliminate the need for serial communication between the ESP32 and an external computer, enabling full automation through direct Wi-Fi connectivity, allowing the ESP32 to fetch real-time atmospheric data from an online weather API and combine it with onboard sensor readings.
- To automate agricultural actions, such as controlling a water pump, based on the machine learning-based rainfall prediction triggered by changes in light intensity, detected via an LDR (Light Dependent Resistor), enabling an event-driven control flow.
- To integrate a suite of environmental sensors—including DHT22 for temperature and humidity, BMP180 for atmospheric pressure, and LDR for light intensity detection—for real-time, local environmental monitoring in support of machine learning input.

- To minimize human intervention and optimize water usage in agriculture, by providing a responsive and intelligent irrigation system that reacts only when predicted conditions (like rainfall) justify action, thereby increasing reliability and reducing resource wastage.
- To deliver a compact, energy-efficient, and cost-effective IoT solution suitable for localized weather forecasting and precision farming, utilizing MicroPython for lightweight code execution and maximizing the utility of the ESP32's built-in capabilities.

1.3) Significance and Applications of the Project:

The "Weather Forecasting and Automatic Watering System in ESP32 using MicroPython" addresses critical challenges in sustainable agriculture and environmental automation. Its significance lies in its ability to make autonomous, intelligent decisions based on real-time and predicted weather conditions—particularly rainfall—thus optimizing water usage and reducing manual labor. This system offers a practical solution to farmers, researchers, and environmentalists by integrating Internet of Things (IoT) and Machine Learning (ML) into a lightweight, cost-effective embedded platform.

From a technological standpoint, the project demonstrates how machine learning algorithms, when trained offline and translated into mathematical expressions (mean, scale, coefficients, and intercept), can be embedded into microcontrollers like ESP32 for local inference. This reduces the need for constant cloud connectivity or powerful edge processors, making the solution highly energy-efficient and suitable for rural and remote areas. The inclusion of both sensor-based data and online weather API input further enhances the system's prediction accuracy, creating a hybrid model for smarter decision-making.

❖ Applications:

- Smart Irrigation Systems: Automates watering based on predicted rainfall, reducing unnecessary water usage and improving crop yield.

Fig-1: Smart Irrigation Systems

- Greenhouse Monitoring: Helps maintain optimal conditions by controlling watering without human intervention.
- Weather-Based Automation: Can be extended to switch other devices (e.g., fans, lights, covers) based on weather predictions.

Fig-2: Greenhouse Monitoring

- Precision Agriculture: Contributes to data-driven farming, minimizing resource wastage and maximizing productivity.
- Environmental Monitoring Projects: Useful for academic, research, and pilot-scale environmental monitoring setups.
- Smart Cities & Urban Gardens: Enables intelligent irrigation in smart homes, public gardens, and rooftop farming systems.

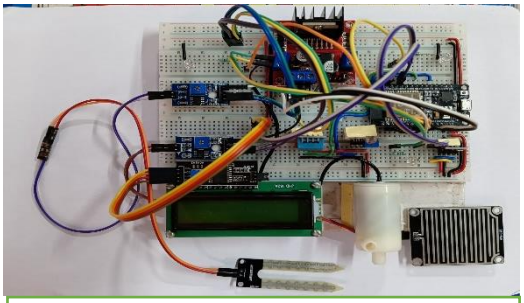

Fig-3: Environmental Monitoring Projects

Overall, the project holds immense potential in promoting sustainable and scalable automation practices in agriculture and environmental monitoring, especially in regions where resources are limited and smart farming is essential.

1.4) Scope of the Project:

The project titled “Weather Forecasting and Automatic Watering System in ESP32 using MicroPython” explores the development and implementation of an intelligent, autonomous system for weather-based irrigation control. The scope of this project extends across the domains of embedded systems, machine learning, Internet of Things (IoT), and environmental automation. It provides a compact and efficient solution for managing water resources effectively by predicting rainfall and making real-time decisions on whether irrigation is necessary.

This system is designed around the ESP32 microcontroller, which offers built-in Wi-Fi connectivity and is capable of running MicroPython for rapid and lightweight programming. It uses environmental sensors (DHT22, BMP180, LDR) to collect on-site data and fetches

additional weather parameters from an external API to form a hybrid dataset. A logistic regression model, trained externally using Python and Google Colab, is used for rainfall prediction. Only the mathematical values (mean, scale, coefficients, and intercept) of the trained model are embedded in the ESP32 for local prediction.

The scope includes real-time triggering based on environmental changes, autonomous decision-making, and actuation through a mini water pump. It also involves visual feedback via an LCD display and connectivity validation using status LEDs. The architecture is scalable and modular, allowing easy integration of more sensors or functionalities in future versions.

Furthermore, the system is designed to operate independently, without requiring constant connectivity to a computer or cloud service. This allows deployment in remote or resource-constrained environments, especially in agricultural fields where manual irrigation is either labor-intensive or inefficient.

In summary, the scope of the project includes:

- Real-time environmental data acquisition.
- Wireless API integration for weather updates.
- Embedded machine learning for local rainfall prediction.
- Automatic control of irrigation systems based on prediction.
- A scalable IoT framework suitable for smart agriculture and other weather-sensitive domains.

1.5) Problem Statement:

In traditional agricultural practices, irrigation is often performed based on fixed schedules or manual observation, without considering real-time weather changes or accurate rainfall predictions. This method, although simple, leads to multiple inefficiencies that directly affect crop yield, resource usage, and long-term sustainability. Over-irrigation not only wastes water but also causes nutrient leaching, soil degradation, and excess energy consumption due to prolonged pump operation. On the other hand, under-irrigation during critical stages of plant

growth can result in crop failure or poor-quality harvests, particularly in regions dependent on rainfall.

One of the primary issues is the lack of localized and accessible weather prediction tools. Smallholder farmers and rural cultivators often depend on general weather forecasts, which may not accurately reflect the microclimate of their specific location. In many cases, no predictive insights are available at all, leading to guess-based decisions. Furthermore, most commercial automation solutions are expensive, infrastructure-dependent, or overly reliant on cloud computing platforms—making them unsuitable for use in areas with limited internet access or technical support.

Another technological gap exists in the integration of machine learning with embedded **systems**. Microcontrollers like the ESP32 have limited memory and processing power, making it challenging to deploy standard ML libraries or models. Most real-world ML applications rely on high-performance cloud or edge servers to carry out inference tasks. This dependency increases cost, complexity, and vulnerability to connectivity issues.

There is also a disconnect between sensor-based monitoring systems and intelligent decision-making logic. While various environmental sensors are used for data collection, few systems actually analyze that data in real time to perform meaningful actions such as controlling a water pump based on weather predictions. In addition, many existing embedded systems lack the flexibility to combine both real-time sensor inputs and external weather data dynamically within a lightweight and efficient control framework.

Manual intervention further reduces system efficiency. Farmers must regularly monitor environmental conditions, decide whether to irrigate, and operate the equipment. This process is time-consuming and prone to error, particularly when dealing with unpredictable weather patterns, labor shortages, or large agricultural fields. There is a growing need for smart systems that can perform real-time decision-making and actuation autonomously without human assistance.

The proposed project aims to solve these issues by creating a smart, real-time, and fully embedded rainfall prediction and irrigation system using an ESP32 microcontroller programmed in MicroPython. The core innovation lies in the use of an offline-trained logistic regression machine learning model, whose mathematical parameters (mean, scale, coefficients, and intercept) are embedded directly into the ESP32 code. This enables the microcontroller to

perform on-device predictions based on inputs from local sensors (temperature, humidity, pressure, light intensity) and external meteorological data fetched from an API.

Upon detecting a change in light intensity through an LDR sensor, the ESP32 triggers data collection, fetches external weather parameters, normalizes the combined data using pre-computed mean and scale values, and performs prediction using logistic regression logic. Based on the outcome (rain or no rain), it activates or deactivates a water pump through a motor driver, while simultaneously updating status indicators (LEDs and LCD screen).

This design ensures that the entire system operates independently of a PC, cloud ML engine**,** or constant internet connection. It provides an affordable, low-power, and scalable solution suitable for use in both small-scale farms and urban rooftop gardens. The approach also opens pathways for further developments in edge ML deployment, smart farming, and environmental automation.

In summary, the core problem this project addresses is: **How can an affordable, standalone system be developed to accurately predict localized rainfall using real-time sensor and weather data, and automate irrigation decisions accordingly—without relying on continuous cloud support or external computing infrastructure?**

Chapter 2: Literature Review

2.1) Existing Weather Forecasting Systems:

2.1.1 Smart Weather Monitoring System using IoT (IJERT, 2020):

This research presented a basic yet functional weather monitoring setup utilizing IoT technology, specifically the NodeMCU ESP8266 microcontroller. The setup included the DHT11 sensor, known for measuring temperature and humidity, and established a connection to cloud platforms such as ThingSpeak through Wi-Fi. Data collected from the environment was sent in real time to these platforms, where users could visualize temperature and humidity trends. While the system provided fundamental monitoring capabilities and enabled remote access to environmental data, it was limited in accuracy due to the basic nature of the DHT11 sensor. Additionally, it lacked automation or intelligent control systems that respond to environmental changes.

➤ Our project takes this concept a step further by upgrading the hardware from ESP8266 to ESP32, which offers more processing power, better energy efficiency, and support for simultaneous sensor interfacing. Moreover, we incorporate more accurate sensors like the BME280, which can measure temperature, humidity, and atmospheric pressure with greater precision. Unlike the earlier system that focused solely on data logging, our solution introduces automation—such as triggering motorized covers or activating alerts—based on real-time thresholds. This transforms the weather station from a passive monitoring tool into an active decision-making system. [1]

2.1.2 Design and Implementation of an IoT-based Weather Monitoring System (IEEE, 2019):

This study implemented a weather station using a Raspberry Pi as the central processing unit and leveraged Python programming to collect real-time weather data. The system integrated

both local sensors and online data, particularly from the OpenWeatherMap API, to enhance weather forecasting capabilities. Its primary focus was on combining on-site sensing with cloud-based services to provide a more comprehensive overview of current and predicted weather patterns. This architecture allowed for remote access to data and historical trend analysis but was heavily dependent on internet connectivity and external servers, which could be a limitation in rural or offline environments.

➤ In contrast, our system uses the ESP32 microcontroller running MicroPython for a lightweight, embedded approach. By handling most processing tasks at the edge (i.e., within the device itself), we reduce reliance on external APIs and improve system resilience in locations with poor connectivity. Our use of MicroPython maintains the readability and ease of use seen in full Python implementations while fitting within the memory constraints of ESP32. Additionally, our system is designed with embedded automation features, such as fan control or rain alerts, which are not covered in the IEEE system. This focus on local intelligence and hardware-driven automation ensures real-time responsiveness and usability in diverse environments, including smart agriculture and standalone installations. [2]

2.1.3 Automated Greenhouse Monitoring using Arduino and DHT22 Sensor:

This project was developed to maintain optimal environmental conditions within a greenhouse by measuring humidity and temperature levels using a DHT22 sensor. The system was built around an Arduino microcontroller, which controlled ventilation and cooling systems (like fans) through relay modules. By setting predefined threshold values for temperature and humidity, the system could automatically activate or deactivate the connected appliances. This setup significantly improved the microclimate control inside greenhouses, contributing to healthier plant growth. However, it lacked forecasting capability and was limited to closed-loop control based solely on current readings.

➤ Our project draws inspiration from the automation logic used here, particularly the relay-based control mechanisms. However, we address its limitations by incorporating weather prediction features using environmental sensors like BME280 (which adds pressure measurement) and rain sensors. These additions allow for multi-sensor fusion, where decisions are made not only based on real-time conditions but also on predictive insights. Our use of

ESP32 also enhances system efficiency by offering more GPIO pins and processing power, enabling simultaneous control and monitoring of multiple actuators and sensors. [3]

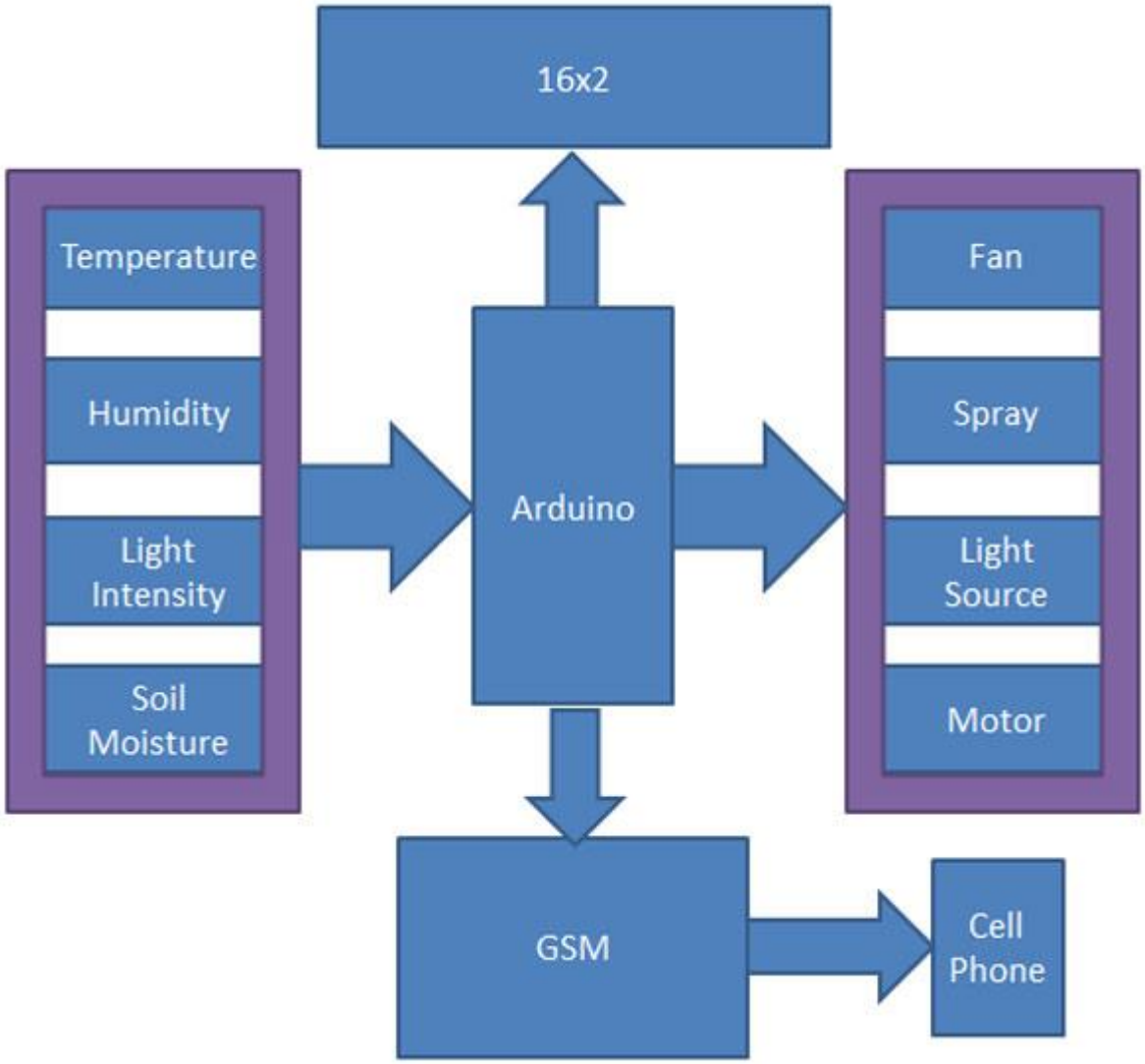

Fig-4: Block Diagram of the Automated Greenhouse Monitoring System

2.1.4 MicroPython on ESP32 for IoT Applications:

This technical book provides an in-depth look at using MicroPython on ESP32 to develop compact, efficient IoT systems. It covers sensor integration, real-time control, and networking capabilities within the MicroPython environment. The emphasis is on simplifying IoT development by using a high-level language like Python while maintaining the performance

benefits of embedded systems. The book also provides examples such as LED control, data logging, and basic automation—all achieved using MicroPython libraries optimized for microcontrollers.

➤ Our project aligns closely with the methodologies discussed in this book, especially in terms of selecting MicroPython for development. Python's simplicity accelerates prototyping and enhances code readability, while ESP32 ensures powerful on-device processing. We go beyond the examples by integrating multiple real-world sensors (e.g., BME280, rain sensors) and developing a working automation system. These include threshold-based triggers that control actuators, as well as time-based alerts. The book validates our technology choices and supports our implementation strategy in building a scalable weather-based automation solution. [4]

2.1.5 Weather Forecasting using Machine Learning Algorithms (IJISRT, 2020):

This paper explores the use of machine learning (ML) models such as Support Vector Machines (SVM) and Decision Trees to forecast weather based on large historical datasets. The research highlights how ML algorithms can identify patterns in temperature, humidity, and pressure data to predict weather conditions over several days. The study demonstrates the ability of data-driven models to provide accurate and adaptive forecasting, especially when trained on region-specific weather patterns.

➤ Although our system does not yet incorporate machine learning, this paper serves as a foundation for future development. At present, our approach is largely hardware-based and focused on real-time environmental automation. However, by collecting data continuously and storing it locally or on cloud platforms in the future, we can train lightweight ML models to improve short-term forecasting accuracy. Incorporating ML could also allow for adaptive thresholding—where the system self-adjusts its behavior based on patterns rather than fixed values. [5]

2.1.6 Wireless Weather Station Using ESP32 and Blynk (Hackster.io, 2021):

This hobbyist project featured a simple wireless weather station using an ESP32 microcontroller and a BME280 sensor for tracking temperature, humidity, and pressure. The collected data was sent to the Blynk mobile application for real-time visualization. The system was intended as a user-friendly interface for personal weather monitoring, particularly useful for DIY enthusiasts. While effective at displaying environmental data, the project focused only on monitoring and lacked any automated response mechanism based on environmental conditions.

➤ Our project builds upon this framework by retaining the core components—ESP32 and BME280—but expanding functionality significantly. In addition to visualization, our system uses environmental thresholds to automatically trigger physical responses such as activating a fan during high temperatures or deploying a motorized cover in case of rain. This moves the project from a passive display tool to an intelligent control system capable of enhancing environmental comfort or protecting assets. We also explore options beyond Blynk for greater platform independence and scalability. [6]

2.1.7 A Survey on IoT in Environmental Monitoring (IJITEE, 2019):

This comprehensive survey reviewed various IoT applications in environmental monitoring, including air quality management, flood detection, water pollution tracking, and precision agriculture. The paper emphasized the role of embedded systems and wireless networks in enabling low-cost, real-time monitoring across diverse geographies. It also discussed the importance of data collection, analysis, and visualization for effective environmental management, especially in developing regions.

➤ This study influenced the broader use-case planning for our project. While our current focus is on weather and climate monitoring for small-scale environments (like rooftops or greenhouses), the framework is designed to be extensible. With minor modifications, our system can be adapted for use in smart agriculture, home automation, or flood alert systems.

The paper also justifies the relevance and necessity of deploying such systems in rural and urban settings alike, reinforcing the societal impact of our work. [7]

2.1.8 Cloud-based Weather Data Logging System using Firebase:

This project involved a microcontroller-based weather station that used Firebase as a cloud backend to store and visualize sensor data over time. The system ensured data persistence and allowed users to track trends and generate analytics remotely through an internet-connected dashboard. Firebase also supported real-time updates, making the platform suitable for live monitoring applications. However, the system lacked on-site control or automation features and was reliant on constant internet access.

➤ While our current system emphasizes local control and automation, we recognize the value of cloud integration for long-term monitoring and data analytics. In future iterations, we aim to incorporate Firebase or similar platforms to enable remote monitoring, historical data visualization, and perhaps even cloud-based machine learning. This hybrid approach—edge-level automation with cloud-level analytics—can help bridge the gap between real-time control and strategic decision-making. [8]

2.1.9 Development of an Intelligent Weather Monitoring System using AI:

This research utilized fuzzy logic algorithms to make intelligent decisions about weather conditions. For instance, it assessed whether it was safe for outdoor activities by analyzing factors like humidity, temperature, and wind speed using linguistic variables (e.g., "high", "low", "moderate") instead of fixed numerical thresholds. The system aimed to mimic human-like reasoning for better adaptability and user-friendliness. It also suggested how AI-driven decision-making could enhance the responsiveness of weather systems.

➤ Our project currently employs static thresholding—for example, turning on a fan when temperature crosses a predefined limit. However, this paper inspires future work to incorporate AI-driven logic like fuzzy inference systems to allow more nuanced decision-making. Such an approach would be especially beneficial in dynamic environments where fixed thresholds may

not suffice. It also opens avenues for integrating user-defined preferences and adaptive control strategies. [9]

2.1.10 DHT Sensor Review and Accuracy Evaluation:

This comparative study analyzed the performance of DHT11 and DHT22 sensors in terms of accuracy, range, response time, and reliability. The findings indicated that DHT22 provides a wider measurement range, higher accuracy, and better long-term stability compared to DHT11, although at a slightly higher cost. The DHT22 also has a lower minimum humidity and temperature detection threshold, making it suitable for more demanding environmental monitoring applications.

➤ Based on this evaluation, we selected the DHT22 over DHT11 for our system. The improved precision and reliability of DHT22 ensure that the weather data collected is both meaningful and trustworthy, especially in edge automation systems where decisions are made based on real-time sensor values. Although we also use BME280 for additional measurements, the inclusion of DHT22 further strengthens our system's data fidelity and responsiveness.[10]

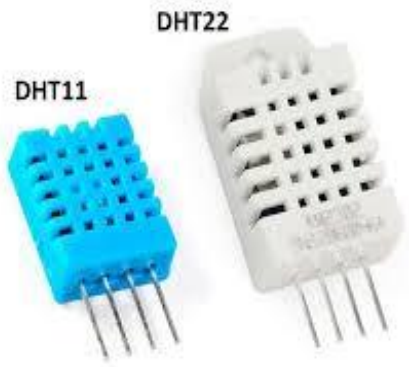

Fig-5: DHT11 & DHT22 Sensor

2.2) Research Gap:

Despite the existence of numerous weather monitoring and IoT-based forecasting systems, the review of literature highlights several critical research gaps that our project aims to address:

2.2.1) Limited Edge-Level Automation:

Many systems, particularly those using platforms like Blynk or Firebase, focus heavily on data visualization and logging. However, they lack local automation. Our project fills this gap by

implementing direct hardware control (e.g., activating devices) based on real-time weather readings—even without internet access.

2.2.2) Lack of Offline Forecasting Capability:

Several systems rely on cloud-based APIs like OpenWeatherMap for forecasts. This approach fails in low-connectivity or rural environments. Our project instead uses on-device threshold logic (based on humidity, pressure drops, etc.) to offer basic local forecasting, independent of online data sources.

2.2.3) Narrow Sensor Integration:

Most reviewed projects use only one or two sensors (e.g., just temperature and humidity). Our system integrates multiple environmental parameters—temperature, humidity, pressure, rain/light sensors—allowing multi-variable trend analysis for better decision-making.

2.2.4) Underutilization of ESP32's Potential:

While many projects use ESP32, they typically don't exploit its full capabilities such as dual-core processing, sleep modes, or advanced GPIO. In our system, ESP32 is used not only for sensor reading but also for real-time automation, forecasting, and potential future Wi-Fi/cloud expansion.

2.2.5) Minimal Use of MicroPython in Automation Systems:

Most IoT systems use C/C++ (Arduino IDE), which, though powerful, increases complexity. Our system leverages MicroPython, which simplifies development, enhances readability, and reduces errors—especially important in student and prototype applications.

2.2.6) No Weather-Responsive Output Systems:

Previous work rarely includes automated output responses like opening/closing covers or alert systems. Our setup uses relays or motors that respond to weather conditions (e.g., rain triggers cover closure), making it a true automatic weathering system.

Chapter 3: System Design and Proposed Model

3.1) Proposed Model Overview:

The proposed model for our project, titled **"Weather Forecasting and Automatic Watering System Using ESP32 in MicroPython"**, integrates environmental sensing, machine learning-based rainfall prediction, and automation through a microcontroller-centric IoT framework. This model is built around a simple yet robust architecture that minimizes reliance on external systems and leverages embedded intelligence within the ESP32 microcontroller.

At the core of the model is the ESP32, a low-power, Wi-Fi-enabled microcontroller programmed entirely in MicroPython. The ESP32 interfaces with environmental sensors such as the DHT22 (for temperature and humidity) and BMP180 (for atmospheric pressure). An LDR sensor is also included, serving as a trigger mechanism. A state change in the LDR indicates a shift in ambient light—such as sunrise or sunset—which is used to initiate a new round of sensing and prediction.

Unlike traditional setups, our system does not use serial communication between the ESP32 and a host PC for machine learning inference. Instead, a logistic regression model is trained offline using Python in Google Colab. From this model, only the essential mathematical values—mean, scale, coefficients, and intercept—are extracted and embedded directly into the ESP32's code. These values allow the microcontroller to perform real-time rainfall predictions using a lightweight logistic regression formula.

In addition to using onboard sensors, the ESP32 is connected to the internet via Wi-Fi and retrieves additional atmospheric parameters—such as wind speed, cloud cover, visibility, dew point, and wind direction—using an API call. This hybrid data fusion (local + cloud) forms a complete feature set for the rainfall prediction model.

The system uses the logistic regression equation to classify whether rainfall is likely (1) or not (0). If rainfall is predicted, the ESP32 activates a connected mini water pump using one of its GPIO pins, enabling automated irrigation without human intervention.

This architecture supports the following key features:

a) **Autonomous Operation:** Fully independent after deployment, requiring no external PC or serial link.
b) **Low Power and Cost:** Ideal for deployment in resource-constrained, rural environments.
c) **Smart Prediction:** Embedded logistic regression logic allows for reliable local inference.
d) **Hybrid Data Collection:** Combines onboard sensor readings with cloud-based weather parameters.
e) **Event-Driven Execution:** Uses LDR sensor to optimize execution timing and power usage.

The accompanying image visually represents the hardware model. It includes the ESP32 development board, sensors (DHT22, BMP180, LDR), a mini water pump module, and power supply elements, all neatly arranged on a breadboard or custom PCB for prototyping.

3.2) System Architecture:

The system architecture of the **Weather Forecasting and Automatic Watering System in ESP32 using MicroPython** is designed to enable fully autonomous environmental monitoring and decision-making for smart irrigation. It integrates real-time sensor data acquisition, weather API communication, embedded machine learning inference, and actuator control within a single microcontroller-based framework.

At the core of the system is the ESP32 microcontroller, which acts as the central processing and control unit. It is programmed using MicroPython, enabling lightweight and flexible coding for real-time applications. The ESP32 is connected to various environmental sensors, including the DHT22 for temperature and humidity, the BMP180 for atmospheric pressure, and an LDR sensor to detect changes in ambient light. These sensors continuously collect local environmental data.

Upon detecting a light change via the LDR, the ESP32 triggers data collection and simultaneously connects to a Wi-Fi network to retrieve additional weather parameters from a public API, such as wind speed, cloud cover, visibility, dew point, and wind direction. The

complete dataset is then normalized using predefined mean and scale values obtained from a previously trained logistic regression model.

The prediction is computed directly on the ESP32 using embedded logistic regression logic that uses pre-extracted coefficients and intercept values. Based on the prediction (rain or no rain), the system autonomously controls a mini water pump via a motor driver. Additionally, an LCD display presents the sensor readings and prediction status, while LED indicators show system states like Wi-Fi connectivity or pump activation.

This architecture ensures low-cost, offline-capable, and real-time intelligent decision-making for agriculture or weather-responsive automation tasks.

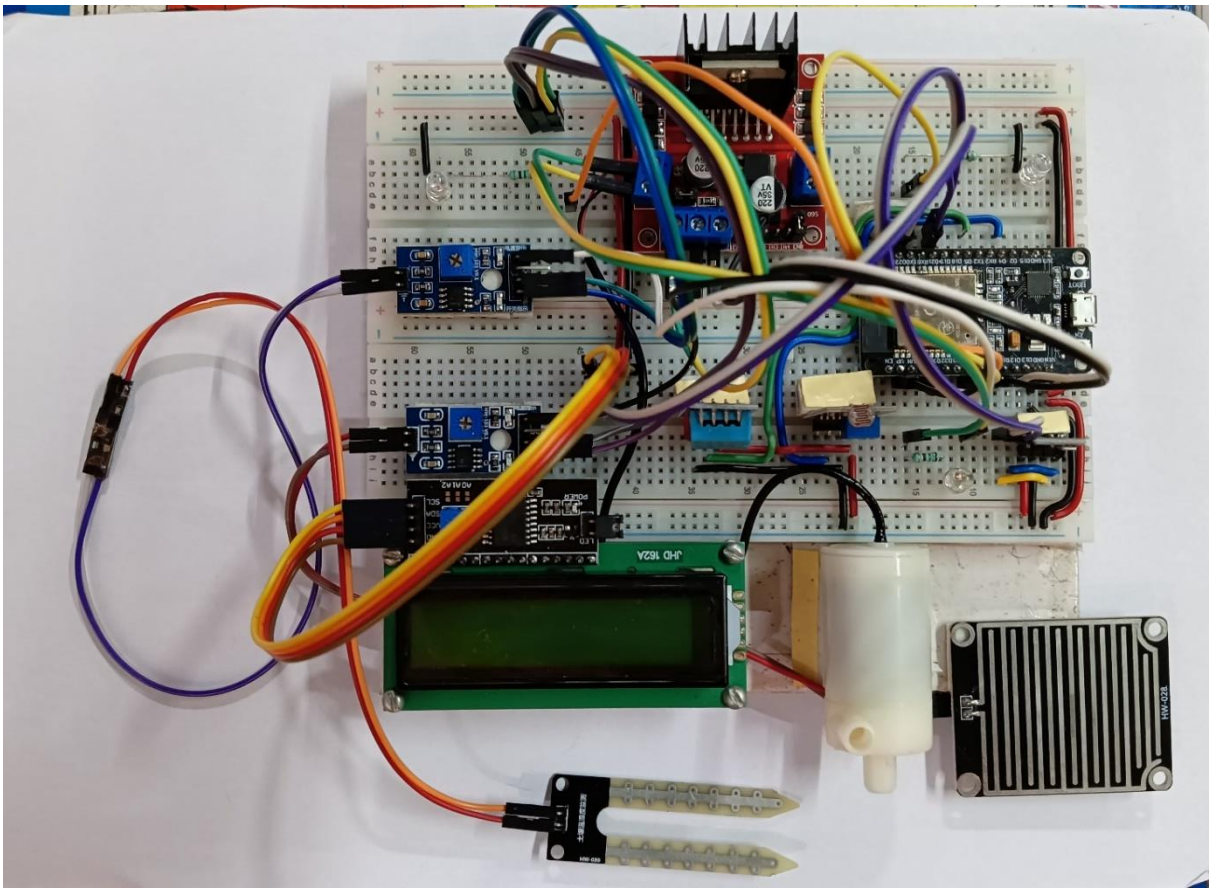

Fig-6: Proposed model of our hardware system

3.3) Flow Chart of the System:

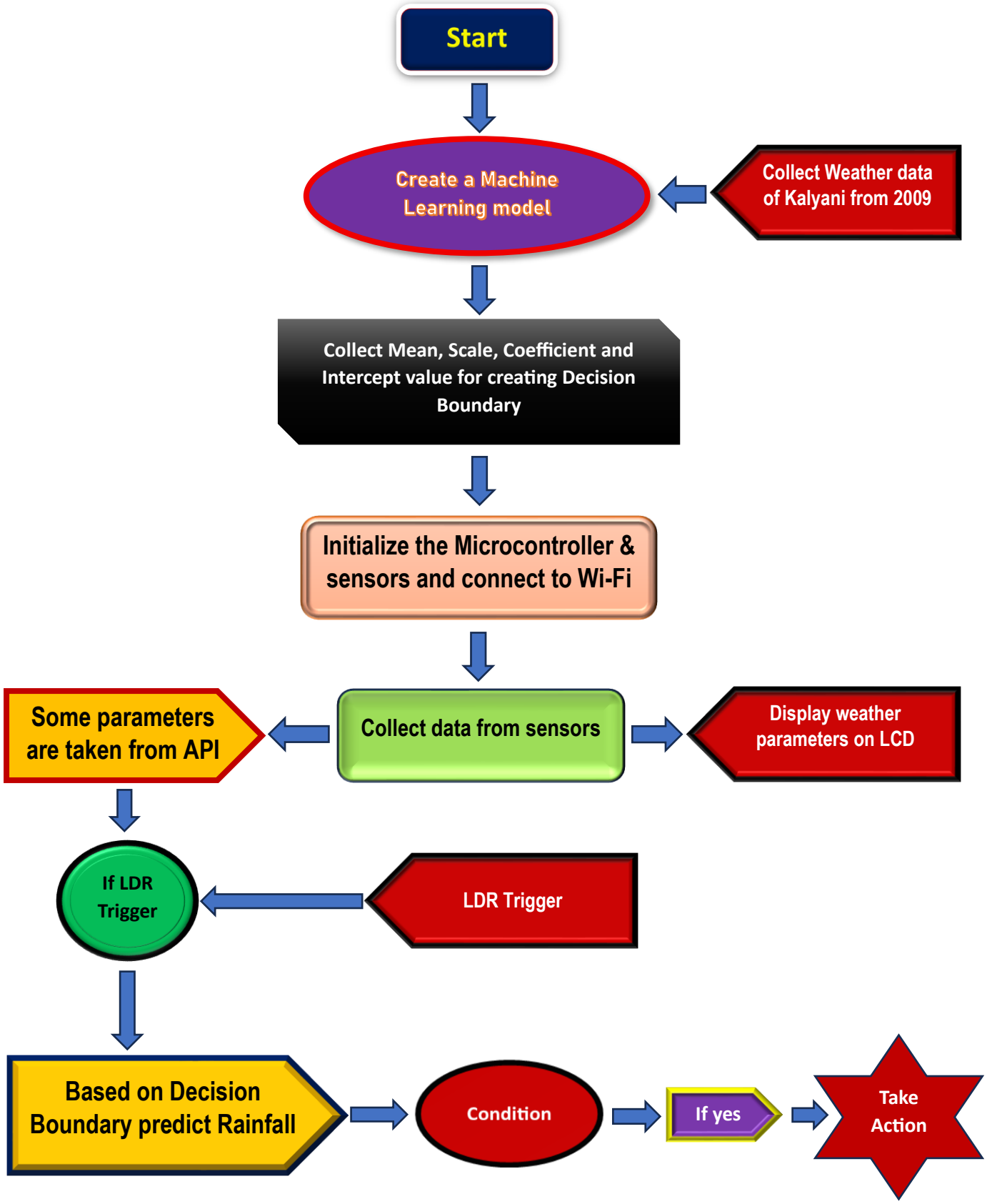

Fig-7: Flow Chart of the Proposed Algorithm

3.4) Block Diagram of the System:

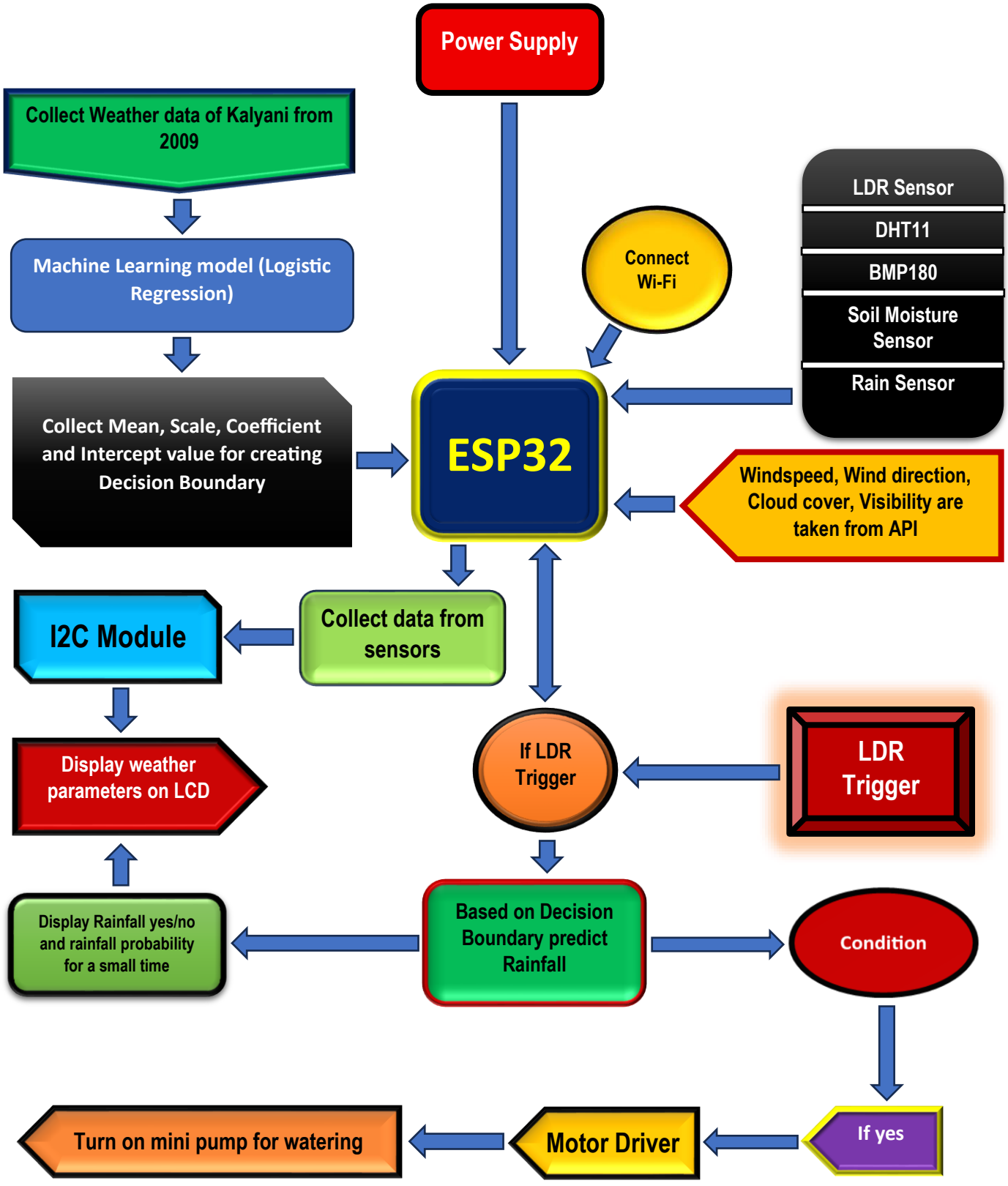

Fig-8: Block Diagram of the Proposed Work

Chapter 4: Hardware Implementation

4.1) ESP32 Microcontroller Overview:

The **ESP32** is a low-cost, low-power system-on-chip (SoC) series with Wi-Fi + Bluetooth capabilities developed by Espressif Systems. It is widely used in IoT (Internet of Things) applications due to its powerful processing, wireless communication abilities, and flexible GPIO interfaces.

A Weather Forecasting System using the ESP32 module integrates multiple sensors, such as temperature, humidity, pressure, and rain sensors, to collect environmental data. With its built-in Wi-Fi and Bluetooth capabilities, the ESP32 enables real-time data transmission to cloud platforms or mobile apps. This system is ideal for IoT-based weather monitoring, offering efficient, reliable, and scalable solutions for forecasting and analysis.[11]

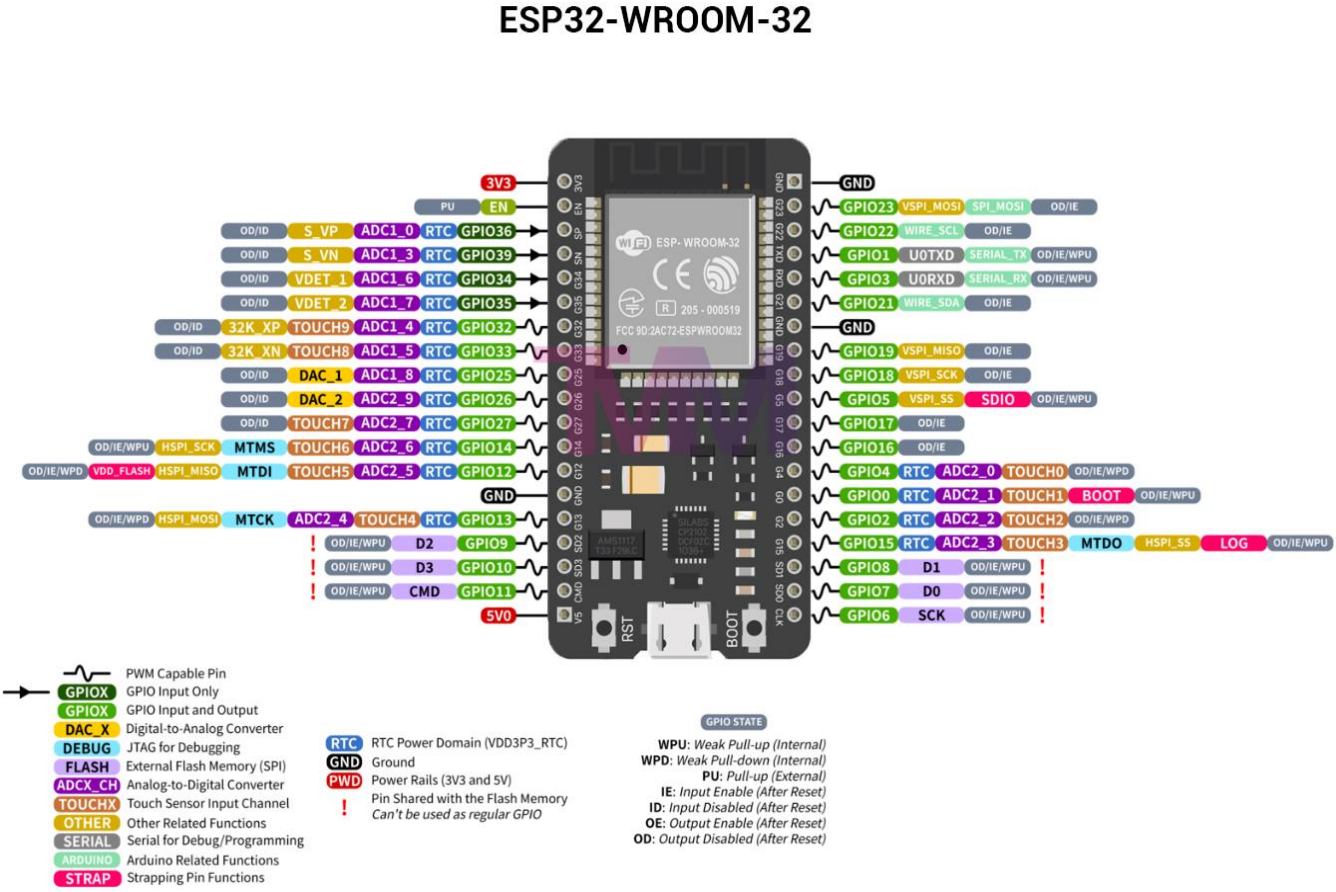

Fig-9: ESP32 Microcontroller

4.1.1) Working of ESP32:

a) Microcontroller Core:

- Dual-core **Tensilica Xtensa LX6** microprocessor (up to 240 MHz).
- Can run real-time operating systems (e.g., Free RTOS).

b) Wireless Connectivity:

- **Wi-Fi 802.11 b/g/n** and **Bluetooth v4.2 (BR/EDR + BLE)**.
- Allows communication over the internet and with other Bluetooth-enabled devices.

c) Input/Output (I/O):

- **Up to 34** GPIO **pins (digital I/O, analog input, PWM, I2C, SPI, UART, etc.).**
- **Integrated** ADC**,** DAC**,** touch sensors**,** temperature sensors**, etc.**

d) Power Management:

- Multiple sleep modes to save power.
- Ideal for battery-operated devices.
- Active mode, light sleep, deep sleep – enables energy-efficient use.

e) Programming Support:

- Arduino IDE, MicroPython, Platform IO, ESP-IDF (official SDK).

4.1.2) Properties of ESP32:

a) Dual-core Xtensa® 32-bit LX6 (up to 240 MHz)

b) 520 KB SRAM, 448 KB ROM, optional external flash

c) Wi-Fi (802.11 b/g/n), Bluetooth v4.2 (Classic + BLE)

d) Up to 34 GPIOs, with ADC, DAC, PWM, SPI, I2C, UART

e) 12-bit ADC (up to 18 channels), 2 × 8-bit DACs

f) Capacitive touch sensors (10 channels)

g) 3.0–3.6V (commonly 3.3V)

4.1.3) Advantages of ESP32:

a) **Built-in Wi-Fi and Bluetooth** – No need for extra modules.
b) **Low power consumption** – Good for battery projects.
c) **High processing speed** – Dual-core with real-time capabilities.
d) **Versatile GPIOs** – Compatible with many sensors and actuators.
e) **Rich Peripheral Support** – ADC, DAC, PWM, I2C, SPI, UART, CAN, etc.
f) **Open-source ecosystem** – Compatible with Arduino, MicroPython, ESP-IDF.

4.1.4) Disadvantages of ESP32:

a) **More complex** for absolute beginners compared to Arduino UNO
b) **3.3V logic** – May need level shifters for 5V devices
c) **Higher current draw** in active mode than simple MCUs
d) **RAM & Flash** limitations for heavy AI/ML models
e) **Bluetooth range** is limited compared to other modules

4.1.5) Applications of ESP32:

	Application Area	Description
a.	**IoT Projects**	Home automation, weather stations, smart agriculture
b.	**Wearable Devices**	Health tracking, smartwatches (due to BLE + low power)
c.	**Wireless Sensor Networks**	Sensor hubs, industrial monitoring
d.	**DIY Electronics**	Robotics, motor control, custom gadgets
e.	**Voice Assistants**	Integration with Alexa, Google Assistant
f.	**Data Logging**	Environmental data loggers (with SD card/Wi-Fi upload)

[12][13]

4.2) Breadboard:

A **breadboard** is a rectangular plastic board with a grid of holes (called tie-points) used to insert electronic components and wires to build temporary circuits. It allows reuse of parts and quick circuit design without soldering.

A weather forecast system on a breadboard demonstrates basic functionality using sensors like temperature, humidity, and pressure modules. Connected to a microcontroller, it collects real-time weather data, processes it, and displays results on an output device. This setup is ideal for prototyping and learning weather monitoring concepts affordably and efficiently.[14]

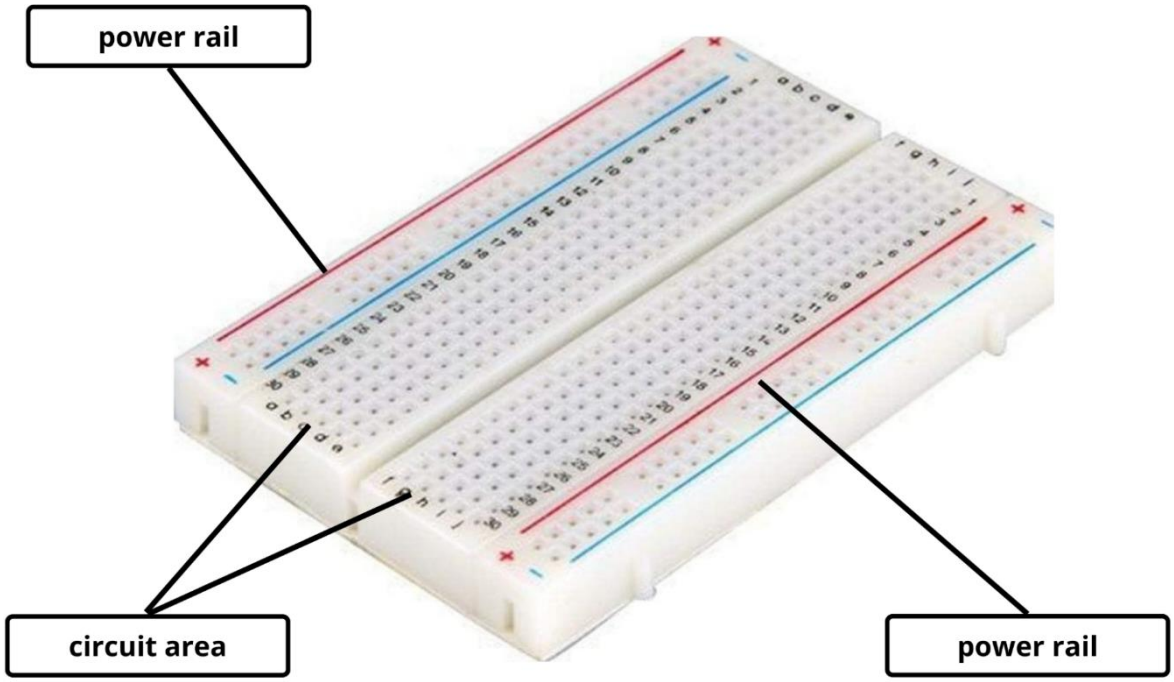

Fig-10: Breadboard

❖ Breadboard Working Principle:

The breadboard is made up of conductive strips hidden underneath its plastic surface:

- **Vertical strips (columns)** on the side (called **power rails**) are usually connected across the board for power (+) and ground (−).

- **Horizontal strips (rows)** in the main area are connected in groups of 5 holes. All 5 are electrically connected.

When you insert components into the holes:

- The leads go into connected rows or columns.
- Current flows through the internal metal clips that link the holes together.

❖ Properties of Breadboard:

	Property	Description
a.	**Material**	Usually, ABS plastic with metal strips inside
b.	**Hole Spacing**	Standard 0.1 inch (2.54 mm) spacing
c.	**Reusability**	Components can be reused multiple times
d.	**Size**	Ranges from mini (170 points) to large (830+ points)
e.	**Voltage Rating**	Typically up to 5V or 12V (not for high-voltage use)
f.	**Current Capacity**	Usually <1A per row

❖ Advantages of Breadboard:

1. **No Soldering Needed** – Quick circuit assembly.
2. **Reusable** – Components and breadboard itself can be reused.
3. **Easy Debugging** – Simple to modify or replace parts.
4. **Visual Learning Tool** – Great for electronics education.
5. **Low Cost** – Inexpensive and accessible.

❖ Applications of Breadboard:

1. **Educational Labs** – For learning electronics and basic circuit design.
2. **Prototype Development** – For testing circuits before final PCB design.
3. **Sensor Testing** – Easily connect sensors with microcontrollers like Arduino or ESP32.

4. **Rapid Testing of Components** – Try out resistors, capacitors, transistors, etc. [14][15]

4.3) Jumper Wires:

Jumper Wires are short, flexible electrical wires used to make connections between components in a circuit. They are commonly used in prototyping and breadboard setups, enabling quick, temporary connections without soldering, making them essential for electronics projects.

A Weather Forecasting System using Jumper Wires connects various sensors (like temperature, humidity, or pressure sensors) to a microcontroller or processing unit. These wires enable quick, flexible circuit setups, making them essential for prototyping and testing weather monitoring systems.[14]

Fig-11: Jumper Wires

❖ Working Principle of Jumper Wires:

Jumper wires **do not perform any electronic processing**. They act as **conductors**, allowing **current or signals** to flow between circuit components or modules. They maintain electrical continuity in prototype circuits.

- Internally made of **copper or tinned copper**, ensuring low resistance.
- The outer insulation is typically **PVC** or **silicone rubber**.

❖ Properties of Jumper Wires:

1. Flexible plastic insulation for safe handling.
2. Available in male and female connector types.
3. Made of low-resistance conductive material like copper.
4. Come in standard lengths and color-coded for easy use.
5. Reusable and durable for prototyping applications.

❖ Advantages of Jumper Wires:

1. **Reusable** – Ideal for quick prototyping.
2. **No Soldering Needed** – Makes circuit building fast and clean.
3. **Versatile** – Supports M-M, M-F, and F-F configurations.
4. **Easy to Organize** – Available in color-coded sets.
5. **Safe for Beginners** – Minimal risk while learning electronics.

❖ Limitations:

1. Not suitable for high current (usually under 1A).
2. Poor connections if worn or loosely inserted.
3. Can clutter circuits if not managed properly.

❖ Applications of Jumper Wires:

1. **Breadboard Prototyping** – Temporary connections for testing circuits.
2. **Microcontroller Projects** – Interfacing Arduino, ESP32, Raspberry Pi with sensors.
3. **Circuit Debugging** – Quickly swap or patch connections.
4. **DIY Electronics Projects** – Robotics, IoT, automation.
5. **Educational Labs** – For students to learn electronics hands-on.[14][16]

4.4) Sensor Integration:

4.4.1) DHT11 - Temperature & Humidity Sensor:

The **DHT11** is a calibrated sensor that provides **temperature (in °C)** and **relative humidity (in %RH)** data. It combines a **thermistor** for temperature and a **capacitive humidity sensor** for humidity, along with a signal processing chip to output data in digital form. Its simple design offers a single-wire serial interface, ensuring easy integration with microcontrollers and other digital devices. For temperature measurement, the DHT11 sensor exhibits reasonable accuracy, offering readings with a resolution of 1 degree Celsius. Additionally, the DHT11 sensor proves itself versatile by also providing humidity measurements. It can accurately measure the relative humidity of the environment and present humidity readings with a resolution of 1% RH. This capability allows users to monitor and respond to changes in both temperature and humidity, making it valuable for environmental monitoring and control systems.[17]

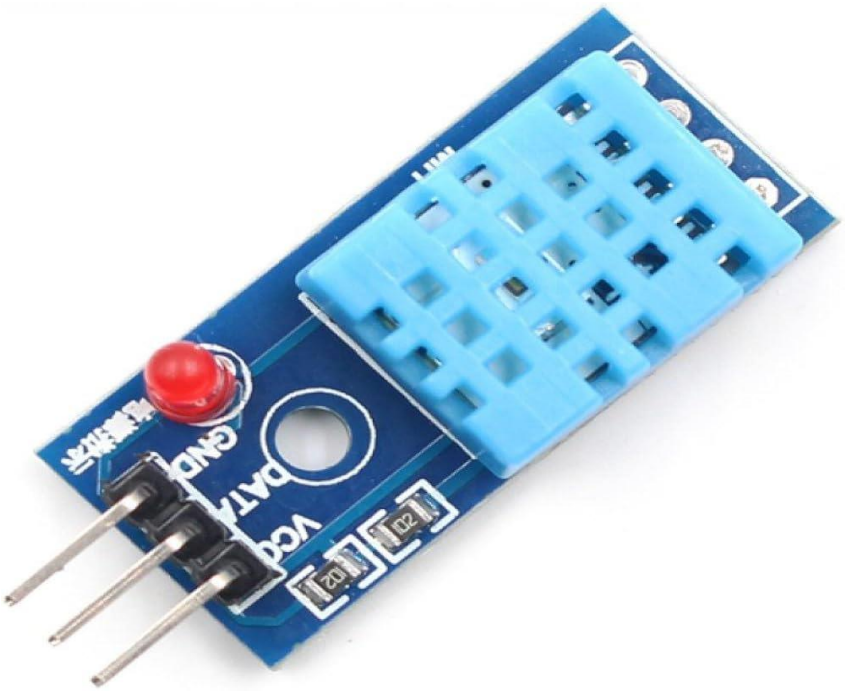

Fig-12: Jumper Wires

❖ Working Principle of DHT11:

1. Humidity Measurement:

DHT11 uses a **capacitive humidity sensor**, where the dielectric constant changes with the humidity level, altering capacitance. The sensor detects this change to measure relative humidity.

2. Temperature Measurement:

It uses a **thermistor** (a temperature-dependent resistor). As temperature changes, the resistance of the thermistor changes, which is measured and converted to a temperature reading.

3. Data Transmission:

- DHT11 communicates using a **single-wire digital protocol**.
- After a **start signal** from the microcontroller, the DHT11 sends 40 bits of data:
  - 8 bits for humidity integer part
  - 8 bits for humidity decimal part (always 0 for DHT11)
  - 8 bits for temperature integer part
  - 8 bits for temperature decimal part (always 0 for DHT11)
  - 8 bits checksum

❖ Properties of DHT11:

1. Measures temperature and humidity accurately.
2. Digital output with single-wire communication.
3. Operating voltage range: 3.3V to 5.5V.
4. Temperature range: 0–50°C, Humidity range: 20–90%.
5. Low power consumption and compact size.

❖ Advantages of DHT11:

1. **Low Cost** – Ideal for budget projects.
2. **Simple Digital Interface** – Easy to interface with Arduino, ESP32, Raspberry Pi, etc.

3. **Low Power Consumption** – Suitable for battery-powered systems.
4. **Compact Size** – Fits easily in small enclosures.

❖ Limitations:

1. Low accuracy compared to DHT22 or SHT sensors.
2. Limited range (temperature: 0–50°C, humidity: 20–90%).
3. Not suitable for harsh or industrial environments.

❖ Applications of DHT11:

1. **Weather Stations** – For monitoring ambient conditions.
2. **Smart Homes** – In air conditioning or HVAC systems.
3. **Agriculture Monitoring** – To track greenhouse or farm humidity.
4. **IoT Projects** – Works well with ESP32, Raspberry Pi, NodeMCU, etc.
5. **Environmental Logging** – For simple temperature and humidity data logging.[16][17][18]

4.4.2) LDR Sensor:

An LDR is a passive electronic component made from semiconductor materials like cadmium sulfide (CdS)**.** Its resistance decreases as the intensity of incident light increases. LDRs are used to detect the presence, absence, or level of ambient light.

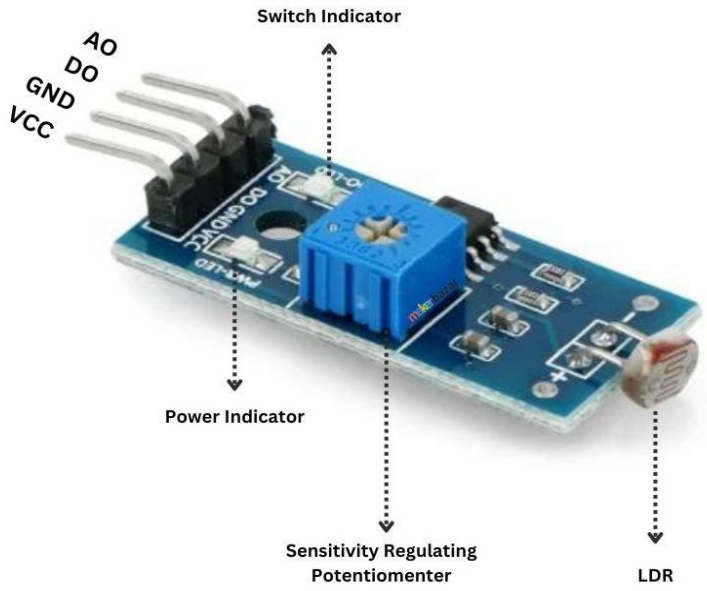

Fig-13: LDR Sensor

A Weather Forecasting System using an LDR Sensor monitors light intensity to assess daylight levels and detect changes in weather conditions like cloud cover. It's ideal for basic weather monitoring and integrated with systems for enhanced forecasts.[14]

❖ Working Principle of LDR:

The LDR works on the principle of **photoconductivity**:

- In **darkness**, the LDR has **high resistance** (up to megaohms).
- When **light falls** on it, photons give energy to electrons, freeing them and increasing **conductivity**.
- Thus, in bright light, its **resistance drops** significantly (a few hundred ohms).

Voltage Divider Configuration: -

LDRs are commonly used in a **voltage divider circuit** with a fixed resistor to produce an analog voltage signal that changes with light.

❖ Properties of LDR:

1. Resistance decreases with increasing light intensity.
2. Made of photoconductive materials like cadmium sulfide.
3. Passive component requiring no external power.
4. Used to detect light levels or brightness.
5. Slow response time to light changes.

❖ Advantages of LDR:

1. **Low Cost** – Cheap and easily available.
2. **Simple to Use** – Easy to connect with microcontrollers.
3. **Analog Output** – Smooth change in resistance with light.
4. **Passive Component** – No external power required for sensing.

❖ Limitations:

1. **Slow Response Time** – Not ideal for fast light changes.
2. **Affected by Temperature** – Resistance can vary with heat.
3. **Not Very Precise** – Better for threshold-based applications, not high precision.

❖ Applications of LDR:

1. **Street Light Automation** – Automatically turns on lights at night.
2. **Solar Trackers** – Detect light direction for solar panels.
3. **Light Meters** – For photography and exposure control.
4. **Burglar Alarms** – Detect shadow interruptions in security systems.
5. **Electronic Toys and Robots** – For light-following behavior.
6. **Smart Farming** – To monitor sunlight levels for crops.[14][19]

4.4.3) BMP180 - Barometric Pressure:

The **BMP180** is a **digital pressure sensor** that communicates with microcontrollers via **I²C or SPI** interfaces. It's a **successor to the BMP085**, offering better accuracy and lower power consumption in a compact size.

A Weather Forecasting System using the BMP180 Barometric Pressure Sensor measures atmospheric pressure and temperature to predict weather changes. Its precise data helps monitor pressure trends, enabling accurate forecasts for applications like portable weather stations or environmental monitoring systems.[15]

❖ Working Principle of BMP180:

1. Pressure Measurement:

It uses a **piezo-resistive sensor** to measure pressure.

- When atmospheric pressure changes, a diaphragm inside the sensor bends slightly.
- These bending changes the resistance in a Wheatstone bridge circuit.

- The sensor's analog output is then digitized by an internal **ADC** and compensated using calibration coefficients.

2. Temperature Compensation:

- Temperature is measured using an **internal temperature sensor**, and the pressure readings are compensated accordingly to improve accuracy.

3. Altitude Estimation:

- Based on the **International Barometric Formula**, the sensor uses atmospheric pressure changes to estimate **altitude**, assuming standard conditions.

4. Communication:

- It communicates using **I²C** (default, 7-bit address 0x77) or **SPI** protocols.

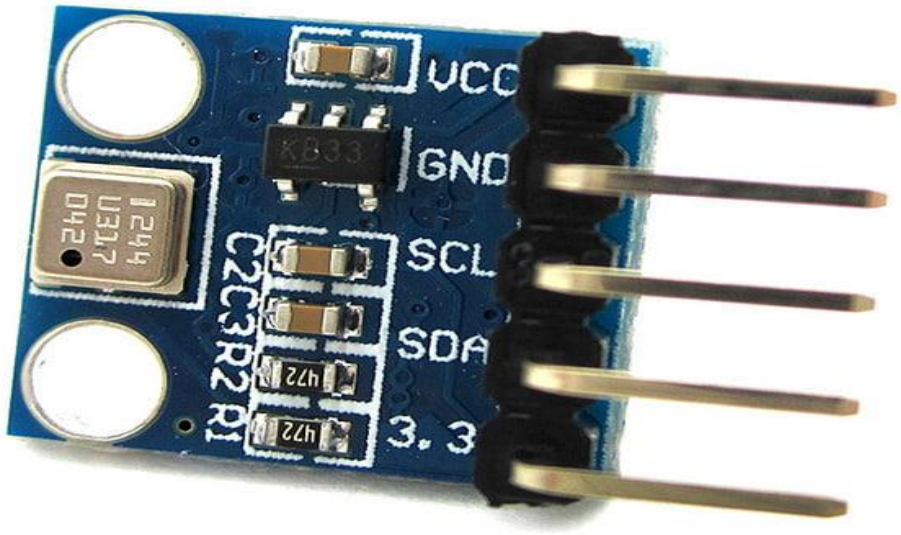

Fig-14: BMP180 - Barometric Pressure Sensor

❖ Properties of BMP180:

1. Measures barometric pressure and temperature.
2. High accuracy with low power consumption.
3. Communicates via I2C or SPI interface.
4. Pressure range: 300–1100 HPa (hectopascal).
5. Compact size suitable for portable devices.

❖ Advantages of BMP180:

1. **High Precision** – Accurate enough for weather stations and basic altimeters.
2. **Low Power Consumption** – Great for battery-powered and wearable devices.
3. **Compact Size** – Easy to integrate into small circuits.
4. **I2C and SPI Support** – Flexible communication with microcontrollers.
5. **Altitude Calculation** – Enables elevation tracking in mobile devices and drones.

❖ Limitations:

1. Limited accuracy for precision scientific work.
2. Replaced by more advanced sensors like **BMP280** and **BMP388**.
3. Only measures **relative altitude**, not GPS-calibrated elevation.

❖ Applications of BMP180:

1. **Weather Monitoring Systems** – For pressure-based forecasts.
2. **Altimeters in Drones** – To track flying height.
3. **GPS Enhancement** – Pressure-based elevation assists GPS data.
4. **IoT Projects** – Smart environment sensing.
5. **Wearable Fitness Devices** – To detect elevation change in steps/floors climbed.[15][16]

4.4.4) Soil Moisture Sensor:

A Soil Moisture Sensor measures the water content in soil by detecting changes in electrical conductivity or capacitance. It is commonly used in agriculture, gardening, and environmental monitoring to optimize irrigation and maintain ideal soil conditions for plants.

There are two main types:

a. **Resistive Soil Moisture Sensor** – Measures resistance between two probes.

b. Capacitive Soil Moisture Sensor – Measures changes in capacitance with soil moisture levels (more durable, not prone to corrosion).

A Weather Forecasting System using a Soil Moisture Sensor monitors soil water content to assess environmental conditions. It supports agriculture by predicting rainfall needs, optimizing irrigation, and aiding in weather-based decision-making for sustainable farming and environmental management.[19]

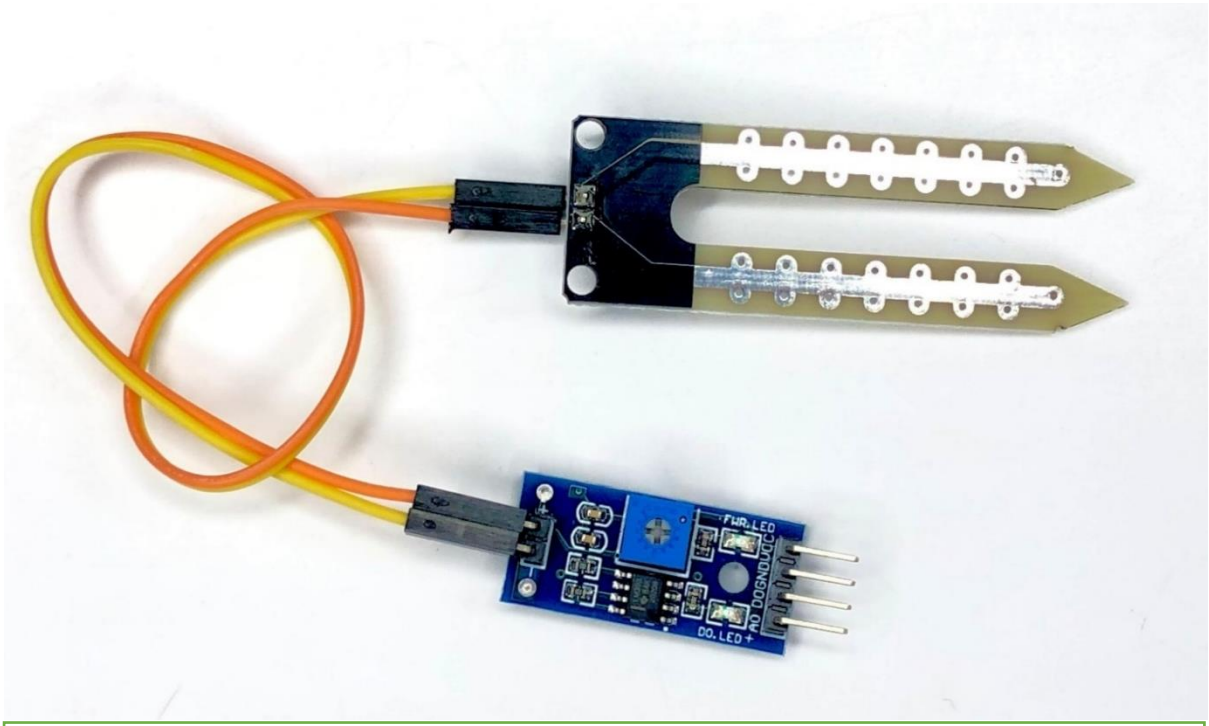

Fig-15: Soil Moisture Sensor

❖ Working Principle:

1. Resistive Sensor:

- Dry soil offers **higher resistance**, allowing **less current** between the probes.
- Wet soil is more conductive, offering **lower resistance**, allowing **more current**.
- The sensor outputs a **voltage** proportional to moisture content.

2. Capacitive Sensor:

- Measures **dielectric permittivity** of the soil, which changes with water content.
- Outputs an **analog voltage** corresponding to the moisture level.

❖ Properties of Soil Moisture Sensor:

1. Detects water content in soil using conductivity.
2. Provides analog and digital output signals.
3. Operates on low voltage (typically 3.3V–5V).
4. Easy to interface with microcontrollers like ESP32.
5. Ideal for smart irrigation and agriculture projects.

❖ Advantages:

1. **Simple and Inexpensive** – Especially resistive types.
2. **Real-Time Monitoring** – Helps in precision farming.
3. **Arduino/ESP Compatible** – Easily interfaces with microcontrollers.
4. **Low Power Consumption** – Ideal for battery-powered systems.
5. **Automates Irrigation** – Prevents under- or over-watering.

❖ Limitations:

1. **Corrosion** (in resistive type) due to electrolysis in wet soil.
2. Accuracy can vary based on soil type (clay, sand, etc.).
3. May require **calibration** for different soil environments.
4. Affected by **temperature** and **salinity** in some cases.

❖ Applications of Soil Moisture Sensor:

1. **Smart Irrigation Systems** – Automate watering in farms or gardens.
2. **Greenhouse Monitoring** – Maintain optimal soil conditions.
3. **IoT Agriculture Projects** – Use with ESP32/Arduino to send data to the cloud.
4. **Hydroponics** – Monitor water availability in growing media.
5. **Soil Science Experiments** – For educational and research purposes.[19][20]

4.4.5) Rain sensor:

A Rain Sensor detects the presence and intensity of rainfall by measuring conductivity changes on its surface. Commonly used in weather monitoring, irrigation systems, and automotive applications, it helps automate responses to rain, like closing windows or adjusting wipers.

A Weather Forecasting System using a Rain Sensor detects rainfall intensity and duration by measuring conductivity changes. It provides real-time data for weather monitoring, irrigation control, or flood warnings, making it ideal for smart environmental and agricultural systems.[21]

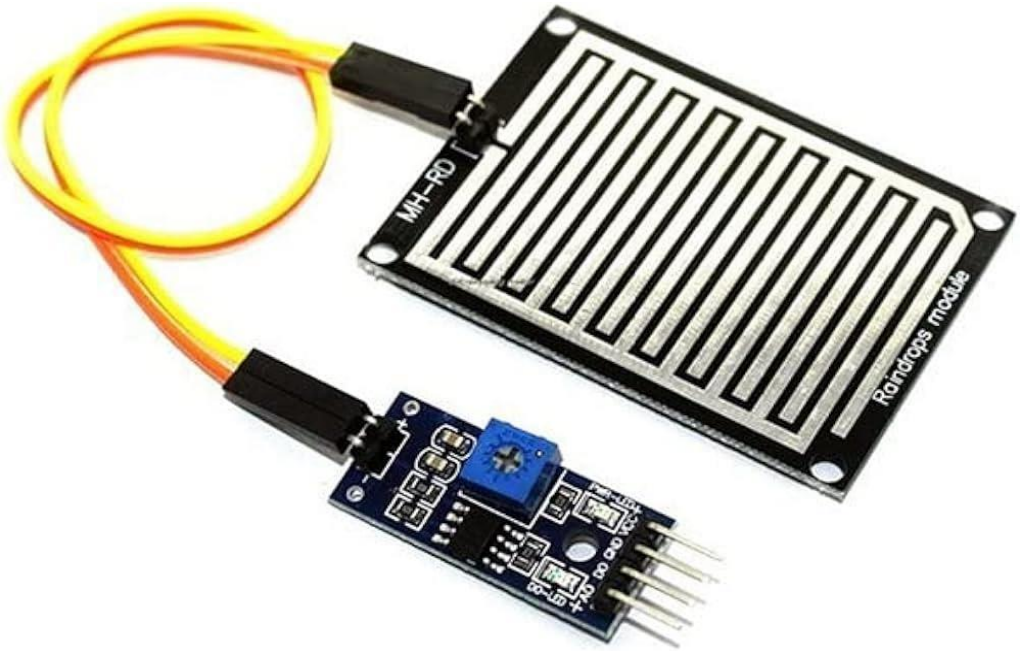

Fig-16: Rain Sensor

❖ Working Principle of a Rain Sensor:

- The rain-sensing plate has parallel **copper tracks** printed on it.
- In **dry conditions**, no current flows between the tracks.
- When **rainwater** touches the plate, it bridges the tracks and **conducts electricity**.
- The control module detects this change and:
    - Sends a **digital signal (LOW)** when rain is detected.

- Sends an **analog voltage** proportional to the amount of water for intensity monitoring.

❖ Properties of Rain Sensor:

1. Detects rain through water conductivity on its surface.
2. Provides both analog and digital output.
3. Operates on 3.3V to 5V supply voltage.
4. Corrosion-resistant PCB for outdoor use.
5. Commonly used in weather monitoring systems.

❖ Advantages of Rain Sensor:

1. **Low Cost and Easy to Use** – Great for beginners and prototyping.
2. **Real-Time Rain Detection** – Useful for environmental sensing.
3. **Multiple Output Types** – Analog for intensity, digital for binary detection.
4. **Compatible with Microcontrollers** – Works well with Arduino, ESP32, Raspberry Pi.
5. **Low Power Consumption** – Suitable for battery-powered IoT devices.

❖ Limitations:

1. **Not waterproof** – Ironically, the sensor plate can corrode if not protected.
2. **Requires maintenance** – Needs periodic cleaning for accurate readings.
3. **Not suitable for long-term outdoor exposure** unless weatherproofed.
4. **Limited precision** – Cannot measure exact rainfall amount like pluviometers.

❖ Applications of Rain Sensor:

1. **Automatic Window/Wiper Control** – In cars or greenhouses.
2. **Smart Irrigation Systems** – Avoid watering when it rains.
3. **Weather Monitoring Stations** – Detect rain onset.
4. **IoT Weather Nodes** – Send rain alerts to the cloud or mobile app.
5. **Rain Alarms** – For clothes-drying systems or terrace alerts.[21][22]

4.4.6) Motor driver:

A **Motor Driver** is an electronic device used to **control motors** (DC, stepper, or servo) using low-power control signals from microcontrollers like Arduino or ESP32. It acts as a **bridge between logic-level control and high-power motor operation**, enabling direction, speed, and torque control.

A **motor driver** takes low-current control signals (from GPIO pins) and provides high-current outputs required to run motors. Most motor drivers are based on **H-Bridge circuits**, which allow **bidirectional control** of motor movement.[14][15]

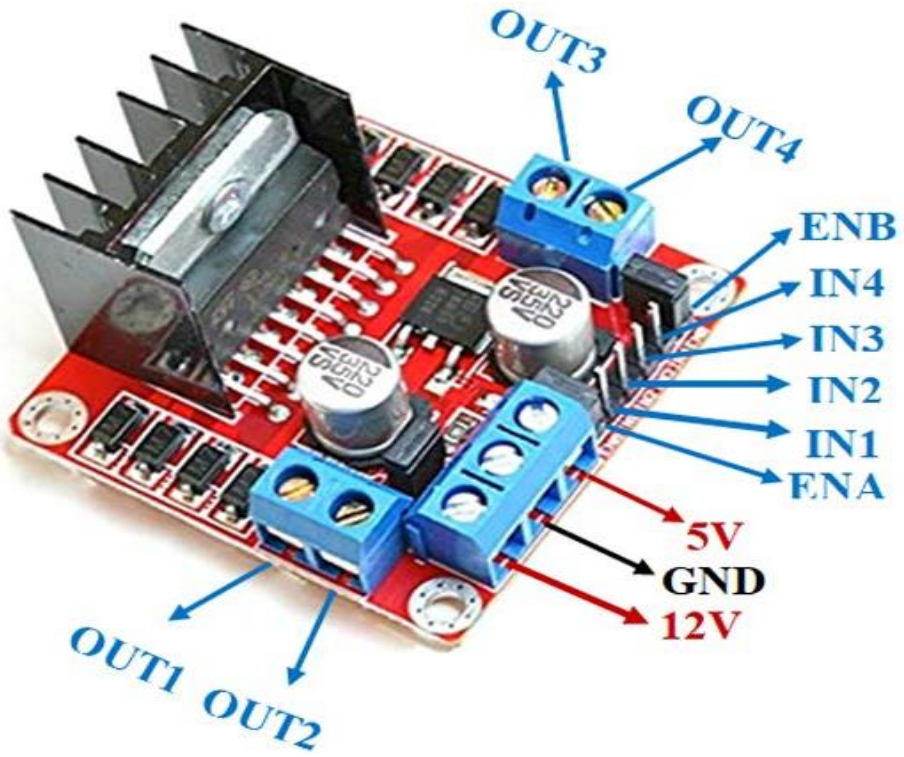

Fig-17: Motor Driver(L293D)

❖ Working Principle:

A **Motor Driver**:

1. **Receives digital signals** (HIGH/LOW) and possibly PWM (for speed control).
2. **Controls power flow** to motors based on these signals.
3. Uses **transistors or MOSFETs** internally to switch motor direction and speed.

Example: DC Motor Control Using L293D:

Input 1	Input 2	Motor Direction
HIGH	LOW	Forward
LOW	HIGH	Reverse
HIGH	HIGH	Brake
LOW	LOW	Coast

PWM signal on the EN (Enable) pin controls the **speed** of the motor.

❖ Properties of Motor Drivers:

1. Controls direction and speed of DC motors.
2. Supports dual H-bridge configuration.
3. Operates with input voltages typically from 5V to 12V or more.
4. Interfaces easily with microcontrollers via logic-level signals.
5. Protects against overcurrent and overheating in motors.

❖ Advantages:

1. **Isolates microcontroller** from high current and voltage.
2. **Enables bidirectional rotation** and speed control.
3. **Compact** and easy to integrate into projects.
4. **Built-in protections** against short circuits and overheating.
5. **Supports multiple motor types** – DC, stepper, servo (indirectly).

❖ Limitations:

1. Heat generation under high load (may require a heat sink).
2. Limited current capacity in older ICs like L293D.
3. Efficiency losses in older bipolar H-bridge ICs.
4. Some ICs require external flyback diodes (if not built-in).

❖ Applications of Motor Drivers:

1. **Robotics** – Drive wheels, arms, and grippers.
2. **Automated Vehicles** – Motor control in drones, line-following robots.
3. **Smart Home Systems** – Control blinds, doors, fans.
4. **CNC Machines & 3D Printers** – Stepper driver control.
5. **IoT Projects** – Motorized watering, environmental actuators.[14][15][24]

4.4.7) Mini Water pump:

A Mini Water Pump is a compact electric pump used to move water in small-scale electronics, robotics, and automation projects. It is commonly used in Arduino and IoT projects for automated irrigation**,** fountains**,** cooling systems, and DIY science experiments.[25]

Fig-18: Mini Water Pump

❖ Working Principle of Mini Water Pump:

Mini water pumps usually operate using DC motors and the centrifugal principle:

1. When power is supplied (typically 3V–12V DC), the motor inside rotates an impeller.

2. The impeller creates centrifugal force, pushing water out through the outlet.
3. The pump pulls water from a reservoir through the inlet and forces it out of the outlet.

They do not suck water over a distance — they must be submerged or primed.

❖ Properties of Mini Water Pump:

- Compact design suitable for small-scale liquid transfer.
- Operates on low voltage (typically 3V–12V DC).
- Submersible and non-submersible types available.
- Capable of continuous or intermittent operation.
- Commonly used in IoT irrigation and DIY projects.

❖ Advantages:

- **Compact size** – Ideal for small projects and limited space.
- **Low power** – Works with battery and USB power.
- **Easy interfacing** – Can be driven using transistors or motor drivers.
- **Cost-effective** – Inexpensive and widely available.
- **Low noise** – Suitable for indoor and home automation projects.

❖ Limitations:

- **Not self-priming** – Must be placed below the water source or submerged.
- **Low flow and head** – Not suitable for high-pressure applications.
- **May wear out** with continuous use or dry run.
- **Not food-grade or chemical resistant** in most cases.

❖ Applications of Mini Water Pump:

- **Automatic plant watering system** (smart irrigation).
- **Mini fountains** or indoor water features.
- **Liquid cooling** in small electronics setups.
- **Water sampling projects**.
- **Aquarium refilling systems**.[25][26]

4.5) LCD Panel and I2C Module:

A weather forecasting system using an LCD and I2C module is a compact and efficient setup to display real-time weather data. Sensors like DHT11/22 (temperature, humidity) and BMP280 (pressure) collect environmental data, processed by a microcontroller (e.g., Arduino). The I2C module simplifies communication between the microcontroller and the LCD, reducing wiring complexity. Weather parameters are displayed on the LCD screen, offering a user-friendly interface. This system is ideal for home or educational use, emphasizing affordability and functionality.[16][23]

Fig-19: LCD Panel and I2C Module

❖ Working of LCD Panel and I2C Module:

LCD Panel (16x2 LCD):

- A **16x2 LCD** means it can display **2 lines of 16 characters**.
- Each character is made up of a **5x8 pixel matrix**.
- Controlled using:
  - **RS (Register Select)**
  - **E (Enable)**
  - **Data Pins (D4 to D7 for 4-bit mode)**

I2C Module (PCF8574):

- Connects to the LCD via a header on the back.
- Communicates with the microcontroller using just **two wires**:
  - **SDA (Serial Data Line)**
  - **SCL (Serial Clock Line)**
- The I2C module converts serial data into parallel data signals required by the LCD.

❖ How It Works Together:

1. Microcontroller sends **data or commands** to the I2C address (usually 0x27 or 0x3F).
2. I2C module receives data and uses the PCF8574 chip to toggle appropriate pins on the LCD.
3. LCD displays the characters or executes the command (e.g., clear screen).

❖ Properties:

LCD Panel:

1. Displays alphanumeric characters and symbols.
2. Operates at 5V with parallel data communication

3. Common sizes include 16x2 and 20x4 characters.
4. Backlight enables visibility in low light.
5. Uses HD44780 controller for compatibility.

I2C Module:

1. Converts parallel LCD to serial communication.
2. Uses only two pins: SDA and SCL.
3. Reduces wiring complexity in microcontroller projects.
4. Includes a potentiometer to adjust LCD contrast.
5. Compatible with standard LCDs via I2C protocol.

❖ Advantages:

LCD Panel:

- Clear display for text, numeric data, and custom characters.
- Backlight for visibility in low light.
- Custom character support (up to 8 custom characters).
- Readily available and low cost.

I2C Module:

- Saves microcontroller pins (only 2 pins required instead of 6+).
- Supports multiple devices on the same I2C bus.
- Compact and easy wiring.

❖ Limitations:

- Slower than SPI or parallel communication (for I2C).
- I2C address conflict may occur if multiple devices share the same address.
- LCDs are not graphical – only display characters.
- Requires library support (like LiquidCrystal_I2C for Arduino).

❖ Applications:

- IoT Projects – Show temperature, humidity, and other live sensor data.
- Embedded System Interfaces – Status screens for systems.
- Data Loggers – Display collected data in real time.
- DIY Electronics – Battery monitors, clocks, weather stations.
- Robotics – Show mode or sensor data on mobile robots.[16][23]

4.6) LED:

An **LED** is a **semiconductor device** that emits **visible light** when an electric current passes through it. Unlike incandescent bulbs, LEDs do not have a filament and are highly energy-efficient and long-lasting.[15]

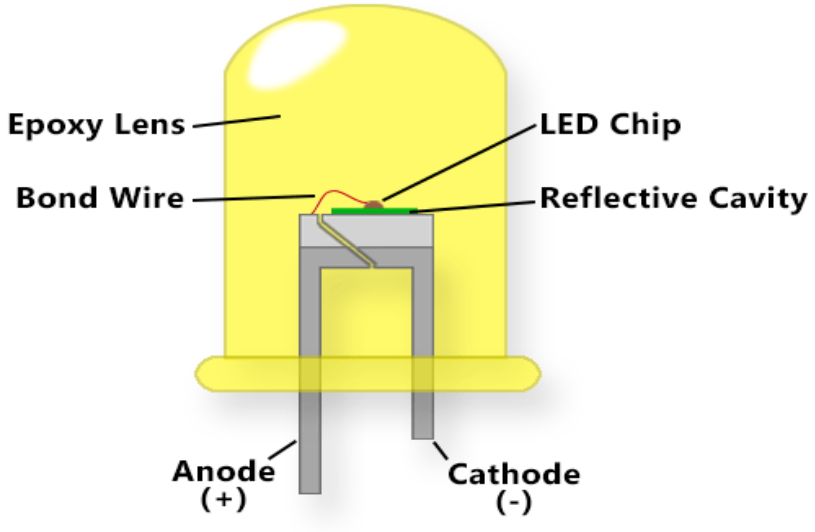

Fig-20: LED

❖ Working Principle of LED:

- LED stands for **Light Emitting Diode**.
- It is a **p-n junction diode** that emits light when **forward-biased** (positive voltage applied to the anode).

- When current flows from the anode (+) to the cathode (−):
  - Electrons and holes recombine at the junction.
  - Energy is released in the form of photons (light) — this is called electroluminescence.
- The color of the light depends on the semiconductor material used and its bandgap.

❖ Properties of LED:

- Emits light when electric current flows through it.
- Operates at low voltage and current.
- Available in various colors and sizes.
- Long lifespan and energy efficient.
- Polarity-sensitive with anode and cathode terminals.

❖ Advantages of LEDs:

- **Low power consumption** – Highly energy efficient.
- **Long lifespan** – Much longer than incandescent or CFL.
- **Compact size** – Ideal for embedded systems and wearables.
- **Instant ON** – No warm-up time.
- **Durable** – Shock- and vibration-resistant.
- **Low heat output** – Much cooler compared to traditional bulbs.
- **Available in various colors** – Including RGB LEDs for full color control.

❖ Limitations:

- **Polarity sensitive** – Won't light up if connected in reverse.
- **Requires current limiting** – Needs resistor or driver circuit.
- **Directionality** – Light is directional (not omnidirectional like bulbs).
- **Temperature sensitivity** – High heat reduces lifespan and brightness.

❖ Applications of LEDs:

- **Indicator lights** – Power on/off, status LEDs.
- **Display systems** – 7-segment displays, dot matrices, LCD backlights.
- **Lighting** – Home, automotive, street lights, and flashlights.
- **Communication** – IR LEDs in remote controls.
- **Signage and advertisement** – LED billboards and panels.
- **Embedded electronics** – Arduino/ESP32 projects for feedback.[15][17]

4.7) Resistor (150 Ohm):

A resistor is a basic electrical component used to control or limit current in a circuit by providing resistance. It is essential in virtually all electronic devices for voltage division, current limiting, biasing, and protection.[14]

Fig-21: 150 Ohm Resistor

❖ Working Principle:

- Resistors **obey Ohm's Law**:

  V = IR

- In **parallel circuits**:

  - Voltage across each resistor is the same.
  - Current splits between paths according to resistance.

- When two identical resistors are in parallel, the current divides equally, and the total resistance is halved

❖ Properties of a Resistor:

- Limits current flow in electronic circuits.
- Has a fixed resistance value of 150 ohms.
- Follows Ohm’s Law (V = IR) for voltage-current relation.

❖ Advantages:

- Protects components by limiting excess current.
- Suitable for current-limiting in LED circuits.
- Provides precise resistance for stable operation.
- Easily available and cost-effective.
- Compatible with low-power electronic circuits.

❖ Limitations:

- Not suitable for high current applications due to limited power rating.
- Fixed resistance limits flexibility in variable circuit needs.
- Can generate heat if used near its power limit.

❖ Applications:

- Current limiting for LEDs in electronic circuits.
- Pull-up or pull-down resistor in logic circuits.
- Biasing resistor in transistor-based circuits.
- Part of voltage divider networks.
- Termination resistor in communication lines.
- Load resistor for signal testing and simulation.[14][15]

4.8) Power Supply System:

Basically the following adapter I am using to give the power supply to the ESP32 Microcontroller system. Input: 100-240 V~, 50-60 Hz, 0.15 A. Output: 5.0 V = 1.0 A.

Fig-22: Power Adapter

4.9) Complete Hardware System:

The system architecture of the **Weather Forecasting and Automatic Watering System in ESP32 using MicroPython** is designed to enable fully autonomous environmental monitoring and decision-making for smart irrigation. It integrates real-time sensor data acquisition, weather API communication, embedded machine learning inference, and actuator control within a single microcontroller-based framework.

This architecture ensures low-cost, offline-capable, and real-time intelligent decision-making for agriculture or weather-responsive automation tasks.

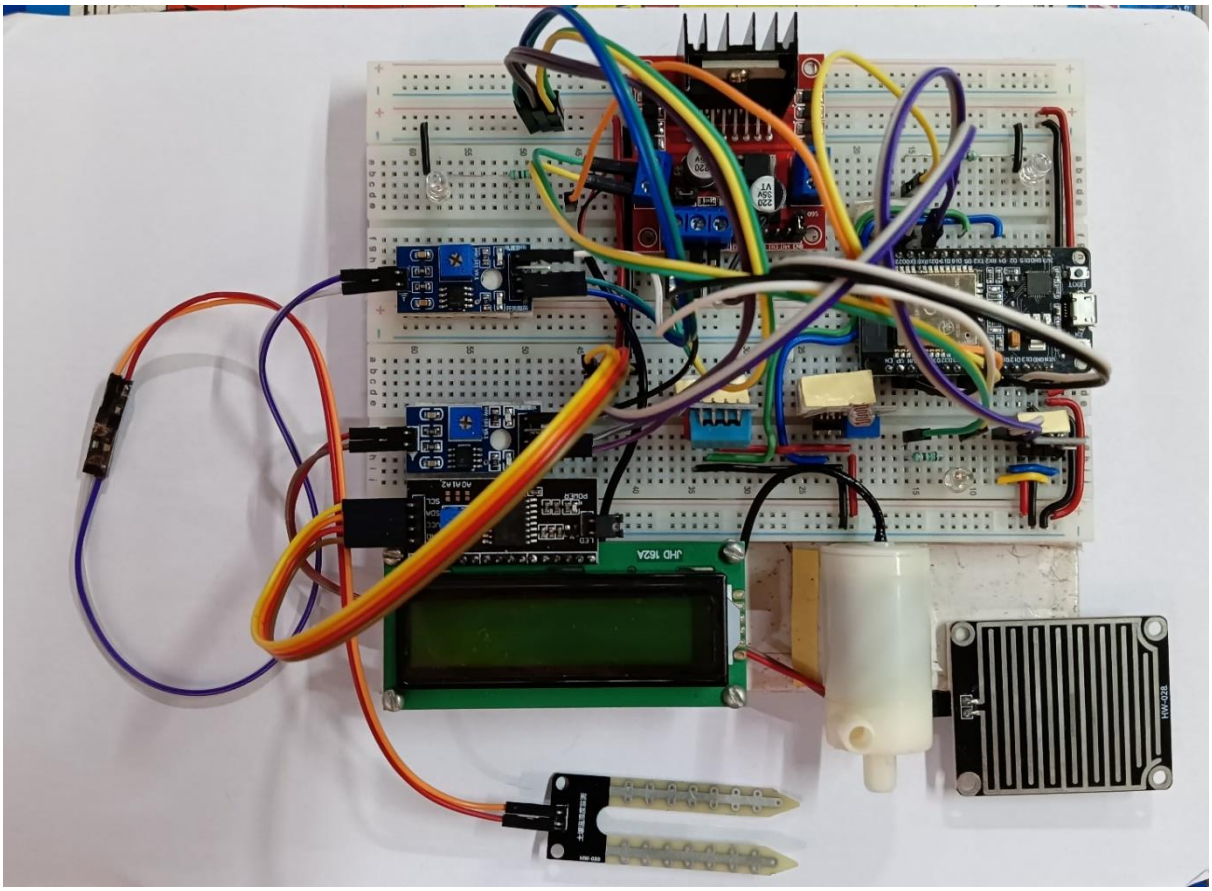

Fig-23: Our Complete Hardware System Model

Chapter 5: Software Implementation

The software implementation of our project, titled "Weather Forecasting and Automatic Watering System Using ESP32 in MicroPython", has been architectured into two major components to efficiently handle data processing and system automation:

5.1) Machine Learning Model for Weather Prediction (Google Colab - Python)
5.2) ESP32 Microcontroller Programming (MicroPython)

5.1) Machine Learning Model for Weather Prediction (Python - Google Colab):

The weather prediction component is powered by a supervised machine learning model developed using Python libraries in a Google Colab environment. The objective of this model is to predict the possibility of rainfall based on historical and real-time weather parameters. The overall software pipeline for this module is described as follows:

5.1.1) Dataset Loading and Preprocessing:

The dataset titled "Kalyani dataset_1_csv.csv" was imported using pandas. The column names were sanitized by removing spaces and converting them to lowercase. Unnecessary fields such as name, datetime, and preciptype were removed to streamline processing.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
from sklearn.preprocessing import StandardScaler
```

```
# Load the dataset
df = pd.read_csv("/content/Kalyani dataset_1_csv.csv")
df.columns = df.columns.str.strip().str.lower()

# Drop unwanted columns
cols_to_drop = ["name", "datetime", "preciptype"]
df = df.drop(columns=[col for col in cols_to_drop if col in df.columns])
```

5.1.2) Feature Engineering:

The rainfall column was identified and converted into a binary classification target. The rainfall was marked as 1 if it was 1mm or more, and 0 otherwise. Features were scaled using StandardScaler.

```
rainfall_col = None
for col in df.columns:
    if "rain" in col and df[col].dtype in [float, int]:
        rainfall_col = col
        break

if rainfall_col is None:
    raise ValueError("Rainfall column not found.")

df[rainfall_col] = df[rainfall_col].apply(lambda x: 1 if x >= 1 else 0)
X = df.drop(columns=[rainfall_col])
y = df[rainfall_col]
X = X.fillna(0)

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

5.1.3) Model Training and Value Extraction:

The data was split using train_test_split. A logistic regression classifier was trained with the scaled data. Instead of storing the full model, only the necessary mathematical components (mean, scale, coefficients, and intercept) were extracted and manually used in the ESP32 MicroPython code.

```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)


best_model = LogisticRegression(max_iter=1000)
best_model.fit(X_train, y_train)


mean_values = scaler.mean_.tolist()
scale_values = scaler.scale_.tolist()
coefficients = best_model.coef_.tolist()[0]
intercept = best_model.intercept_[0]


# These values will be manually embedded in the MicroPython code
print("mean:", mean_values)
print("scale:", scale_values)
print("coefficients:", coefficients)
print("intercept:", intercept)
```

5.2) ESP32 Microcontroller Programming (MicroPython):

The ESP32 code was written entirely in MicroPython and operates independently without any serial connection to an external system. It is connected to Wi-Fi and fetches supplementary weather parameters via API. The model prediction logic (based on mean, scale, coefficients, and intercept) is embedded into the MicroPython code itself. When the LDR sensor detects a

state change, the ESP32 collects live sensor values and API values, performs normalization and prediction internally, and makes the watering decision autonomously.

5.2.1) Sensor and Network Initialization:

```
import urequests
from machine import Pin, I2C
import dht
import time
from bmp180 import BMP180
from machine import SoftI2C
import network

# WiFi connection
wifi = network.WLAN(network.STA_IF)
wifi.active(True)
wifi.connect('your-ssid', 'your-password')
while not wifi.isconnected():
    pass

# Sensor setup
dht_pin = Pin(15)
dht_sensor = dht.DHT22(dht_pin)
ldr_pin = Pin(33, Pin.IN)
motor_pin = Pin(14, Pin.OUT)
i2c = SoftI2C(scl=Pin(22), sda=Pin(21))
bmp = BMP180(i2c)
bmp.oversample_sett = 2
bmp.sea_level_pressure = 101325
```

5.2.2) Embedded Logistic Regression Prediction:

```
# ML model constants (extracted from training)
mean = [/* fill in mean values */]
scale = [/* fill in scale values */]
coeff = [/* fill in coefficients */]
intercept = /* fill in intercept */

def predict(features):
    normalized = [(x - m) / s for x, m, s in zip(features, mean, scale)]
    linear_comb = sum(c * x for c, x in zip(coeff, normalized)) + intercept
    return 1 if linear_comb >= 0 else 0
```

5.2.3) Main Control Logic:

```
previous_ldr_value = ldr_pin.value()

while True:
    current_ldr_value = ldr_pin.value()

    if current_ldr_value != previous_ldr_value:
        previous_ldr_value = current_ldr_value
        try:
            dht_sensor.measure()
            temperature = dht_sensor.temperature()
            humidity = dht_sensor.humidity()
            pressure = bmp.pressure
```

```
        # Fetch external API data (example)
        response = urequests.get("https://api.weather.com/mock-data")
        api_data = response.json()
        windspeed = api_data["windspeed"]
        cloudcover = api_data["cloudcover"]
        visibility = api_data["visibility"]
        dewpoint = api_data["dewpoint"]
        winddir = api_data["winddirection"]

        # Combine all features
        input_features = [temperature, humidity, pressure, windspeed,
cloudcover, visibility, dewpoint, winddir]
        result = predict(input_features)

        if result == 1:
            motor_pin.on()
        else:
            motor_pin.off()
```

Chapter 6: Integration Flow

The integration flow of the Weather Forecasting and Automatic Watering System in ESP32 using MicroPython follows a fully autonomous, wireless, and event-driven architecture. The system is designed to make intelligent decisions using embedded machine learning logic without relying on external computers. The step-by-step operation is described below:

6.1) LDR-Based Triggering:

The ESP32 continuously monitors an LDR (Light Dependent Resistor) to detect significant changes in ambient light intensity. This sensor plays a crucial role in minimizing power usage and unnecessary processing. A change in light level—such as sunrise, sunset, or artificial shading—triggers a new cycle of weather data collection and prediction. This event-driven mechanism ensures that the prediction and watering logic are executed only when environmental conditions change significantly.

6.2) Sensor Data Collection:

Upon detecting a valid trigger, the ESP32 immediately initiates the collection of environmental data using onboard sensors. It reads:

- **Temperature and humidity** from the DHT22 sensor.
- **Atmospheric pressure** from the BMP180 sensor.
  These parameters represent localized, real-time environmental conditions that contribute to the accuracy of the rainfall prediction model.

6.3) Cloud API Interaction:

Simultaneously, the ESP32 connects to a weather API via Wi-Fi to supplement the onboard sensor data with broader meteorological parameters. The ESP32 makes an HTTP GET request and retrieves values such as:

- Wind speed
- Wind direction
- Cloud cover
- Dew point
- Visibility

  These additional features enrich the dataset and improve prediction reliability by incorporating both local and regional weather data.

6.4) Data Preparation and Normalization:

All collected data—both from sensors and the API—is then combined to form a complete feature vector. Before this data is passed to the prediction model, it is normalized using **pre-calculated mean and scale values** extracted during offline machine learning training. This ensures that the features are scaled to the same distribution that the logistic regression model was trained on, maintaining model accuracy and consistency.

6.5) Prediction Execution:

The ESP32 executes the logistic regression formula embedded in its MicroPython code. Using the normalized data and pre-trained **coefficients** and **intercept**, it calculates the weighted sum of the inputs. The output is compared to a threshold (zero in this case). If the result is greater than or equal to zero, the system predicts **rainfall (class 1)**; otherwise, it predicts **no rainfall (class 0)**. This enables real-time, offline rainfall prediction without relying on cloud-based ML inference.

6.6) Decision and Action:

Based on the prediction result:

- If **class 1 (rain likely)**: The ESP32 decides **not to turn on** the water pump, conserving water.
- If **class 0 (no rain)**: The ESP32 **activates the motor** via a GPIO pin, which drives the mini water pump to irrigate the field. Additionally, a **green LED** may indicate that watering is in progress, and the LCD display can show the current prediction and sensor values for feedback and monitoring.

This completely embedded system achieves autonomous decision-making by integrating sensor input, wireless data retrieval, embedded ML logic, and physical control—all within the ESP32 microcontroller. The design eliminates the need for serial connections or continuous cloud computation, making it ideal for remote, low-resource, and real-time agricultural environments.

Chapter 7: Simulation Result

The simulation demonstrates that the system functions as intended, with reliable rainfall prediction and automated watering based on light detection and weather conditions. It integrates IoT, machine learning, and real-time sensor data effectively using MicroPython on ESP32.

The simulation of our proposed system was carried out to validate the functionality and integration of hardware, sensors, and the machine learning model using MicroPython on the ESP32 board. The following outcomes were observed during simulation:

7.1) ESP32 and Wi-Fi Connection:

- The ESP32 microcontroller successfully connected to the Wi-Fi network.
- External weather parameters such as windspeed, wind direction, cloud cover, and visibility were fetched in real-time from a weather API.

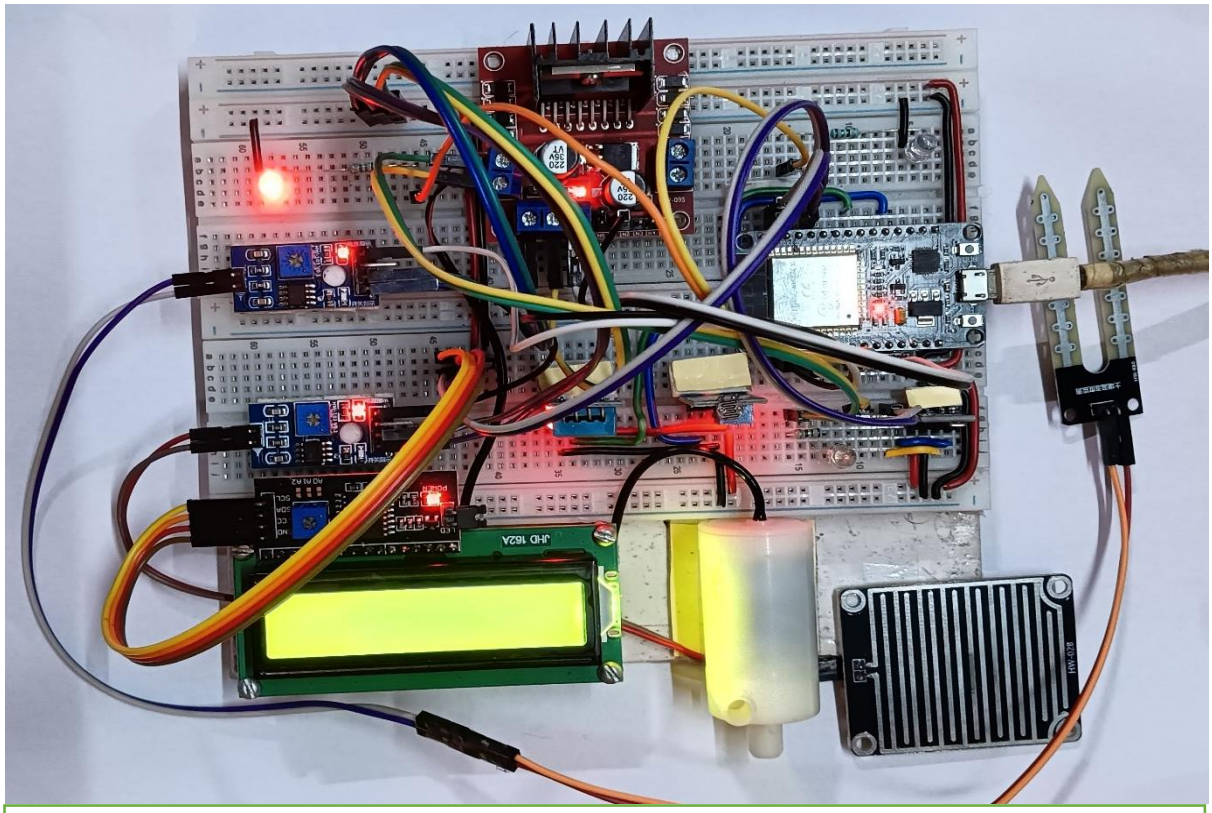

Fig-24: ESP32 is not connected with Wi-Fi

- In our project the Red LED turn on when the ESP32 Module trying to be connect with Wi-Fi, and after connection of Wi-Fi, Red LDE turn off and Blue LED turn on.

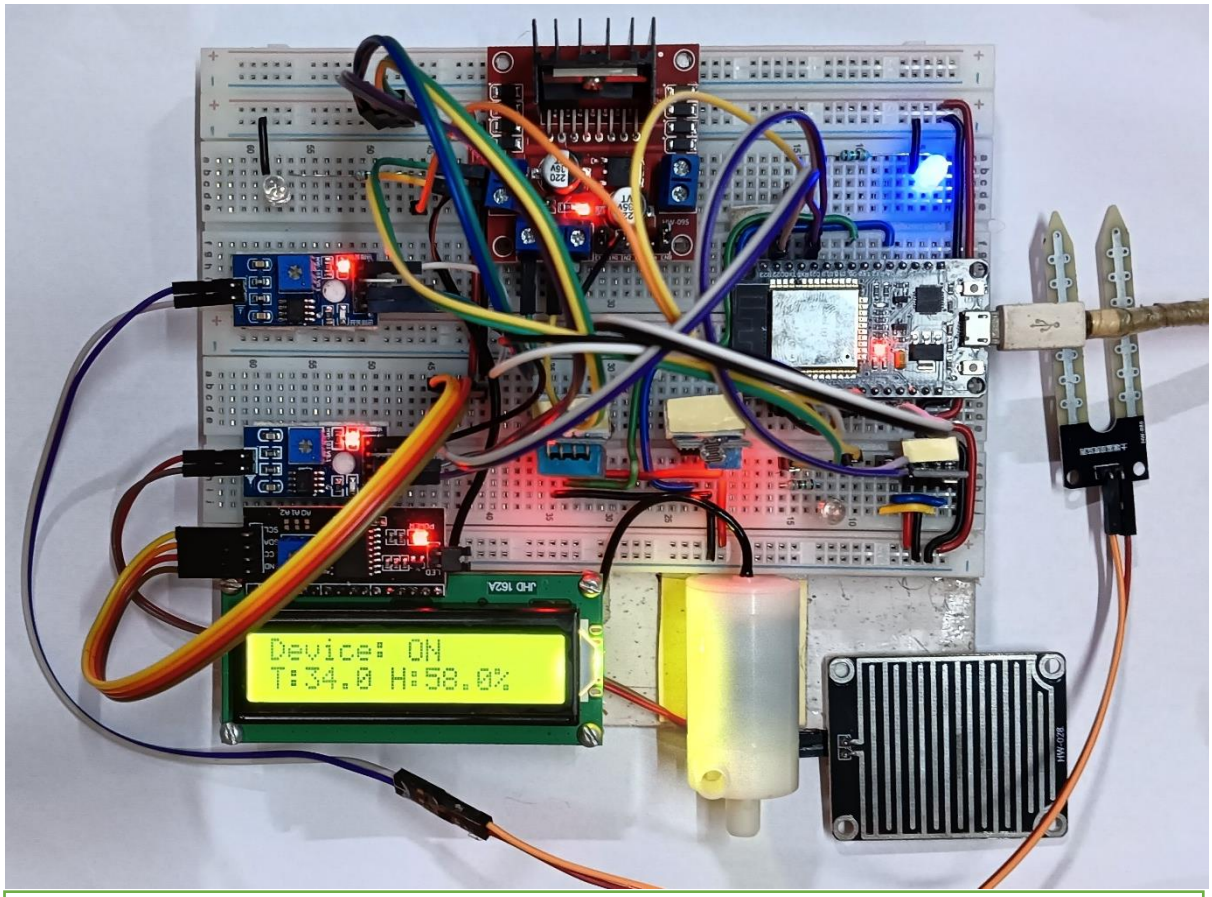

Fig-25: ESP32 is connected with Wi-Fi

7.2) Sensor Data Collection:

- The onboard sensors (DHT11 for temperature & humidity, BMP180 for pressure, Rain Sensor, LDR Sensor, Soil Moisture Sensor) continuously sent data to the ESP32.
- The data was collected and structured properly for use in prediction.

7.3) LDR Trigger Mechanism:

- The system remained in standby mode until the LDR sensor detected a significant drop in ambient light.
- Upon detection, it triggered the data collection process and initiated the rainfall prediction module.

7.4) Machine Learning Prediction:

- The ESP32 utilized a logistic regression model trained on historical weather data (from Kalyani, 2009 onwards).
- Pre-calculated model parameters (mean, scale, coefficient, intercept) were stored and used to compute the decision boundary.
- The prediction result was binary: 1 for rainfall expected (greater than or equals to 0.5), 0 for no rainfall (less than 0.5).

7.5) LCD Display Output:

- Through an I2C module, the LCD displayed:
    - Current sensor readings
    - Rainfall prediction status (Yes/No)
    - Rainfall probability (if implemented)

Fig-26: Showing the Rainfall Prediction value on LCD Display

7.6) Taking Action:

- If prediction result is binary 0 then mini water pump will turn on only for some specified times, otherwise turn off.
- A green LED also will turn on as long as the Mini water pump will turn on.

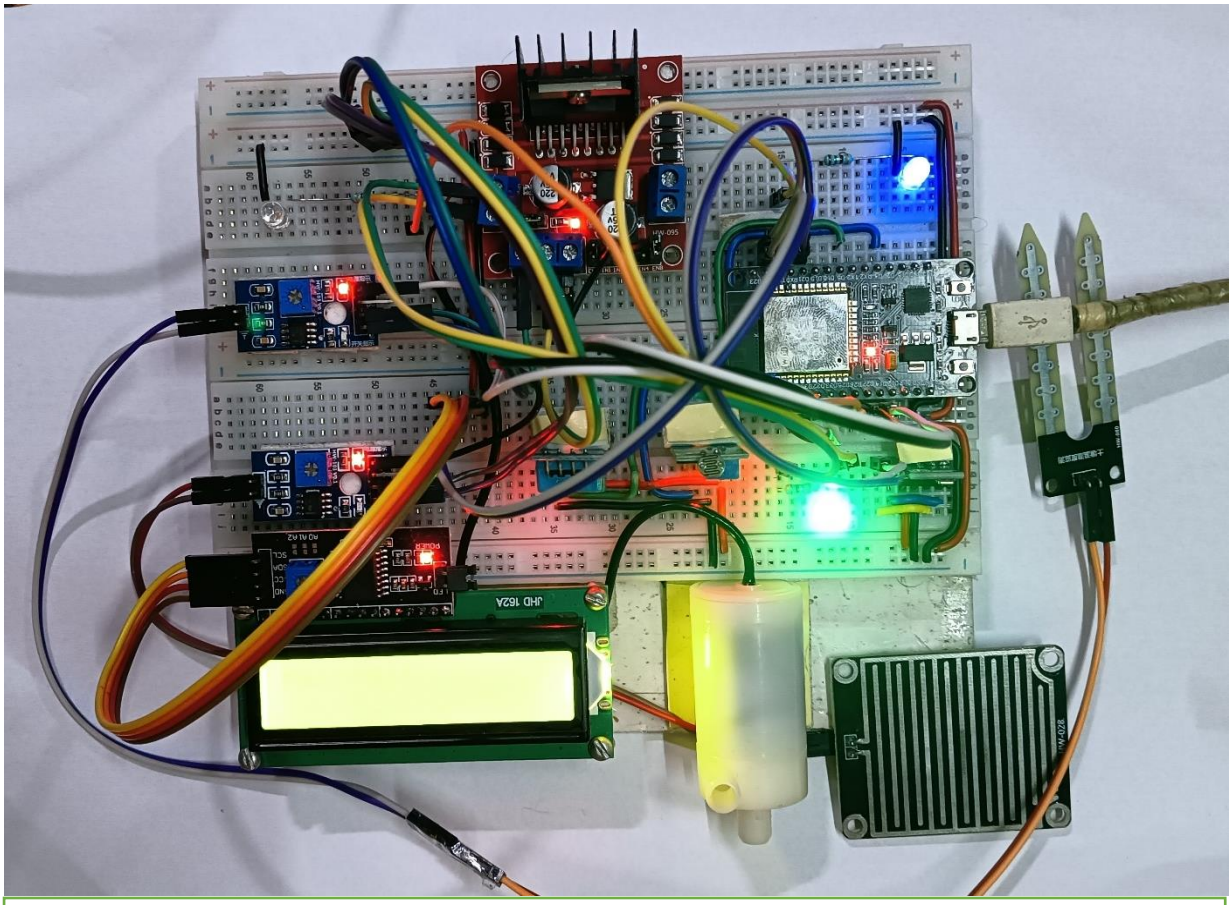

Fig-27: The mini water pump is on as well as green LED is also on

Chapter 8: Advantages, Limitations, and Future Scope

8.1) Advantages of the Proposed System:

The system presents several advantages from both technical and practical perspectives, especially in agriculture, environmental monitoring, and IoT automation.

1. Autonomous and Real-Time Operation:

The entire system operates independently, without the need for a computer or cloud-based prediction engine. Decisions are made in real-time on the ESP32 using embedded logic.

2. Energy and Resource Efficiency:

By predicting rainfall accurately, the system avoids unnecessary irrigation, leading to significant water conservation—especially important in regions facing water scarcity.

3. Low Cost and Accessibility:

The system is built using affordable components like the ESP32, DHT22, and BMP180 sensors, making it cost-effective and deployable in rural or economically constrained areas.

4. Embedded Machine Learning Execution:

Unlike many systems that require server-side ML computation, this project demonstrates successful deployment of ML logic (logistic regression) directly onto a microcontroller, reducing complexity and dependence on the internet.

5. Hybrid Data Source Integration:

It combines local sensor data with real-time cloud-based weather API data, improving the reliability and accuracy of predictions.

6. Scalability and Modularity:

The architecture is modular and can be easily extended with more sensors or other ML models. It supports easy adaptation to various smart farming or environmental systems.

8.2 Limitations of the Current Model:

While the system performs effectively in its current design, it has certain limitations that can be addressed in future iterations:

1. Limited ML Model Complexity:

The logistic regression model is simple and interpretable but may not capture complex nonlinear patterns in weather data. Advanced models could yield higher accuracy.

2. Dependence on External API:

Although API data enhances prediction accuracy, it requires internet access. In remote areas with weak connectivity, this could limit system performance.

3. Fixed Prediction Threshold:

The logistic regression decision threshold is static (zero). A dynamically adjustable threshold based on seasonal trends could improve accuracy.

4. Sensor Accuracy and Drift:

The system relies on low-cost sensors whose readings may drift over time due to environmental exposure. Calibration or using higher-grade sensors could resolve this.

5. No Remote Monitoring Interface:

Currently, there is no mobile or web dashboard for remote monitoring or control. Users must rely on the local LCD and LEDs for output.

8.3 Future Scope and Enhancements:

To further improve the performance and functionality of the system, several extensions and upgrades are possible:

1. Integration of Advanced ML Models:

Future versions can use decision trees, random forests, or lightweight neural networks optimized for microcontrollers (e.g., TensorFlow Lite for Microcontrollers).

2. Mobile App or Web Dashboard:

Developing a mobile application or web interface would allow users to remotely monitor sensor values, control the pump, and view prediction history.

3. Offline Prediction Without API:

Replacing API data with additional onboard sensors (e.g., wind sensor, UV sensor, rain gauge) would eliminate internet dependency and ensure full offline capability.

4. Solar Power Integration:

Integrating a solar panel system would make the solution completely energy-independent and sustainable for long-term deployment.

5. Cloud-Based Logging and Alerts:

Sending periodic data to a cloud platform like Firebase or Things Board could allow historical data logging, trend analysis, and SMS/email alerts.

6. Scalability for Multiple Zones:

The design can be scaled to control multiple irrigation zones independently by integrating more valves and expanding the control logic.

Chapter 9: Conclusion

The development of a weather forecasting and automatic watering system in ESP32 using MicroPython has been a successful demonstration of combining machine learning with embedded systems to address real-world agricultural challenges. The project effectively integrates software and hardware components to predict rainfall and automate irrigation, thereby optimizing water usage and reducing manual effort.

This smart irrigation system offers a sustainable solution for small-scale farming or home gardening by conserving water, reducing human intervention, and providing timely watering.

The project demonstrates how a machine learning model can be simplified and adapted for microcontroller platforms, bridging the gap between AI and IoT. Although the current implementation uses a basic logistic regression model, future improvements may include more advanced models like decision trees or neural networks, integration with mobile apps for remote monitoring, and solar-powered automation for energy efficiency.

9.1) Summary of Work Done:

The project titled “Weather Forecasting and Automatic Watering System in ESP32 using MicroPython” presents the development of a smart, autonomous system that combines weather forecasting and precision irrigation. It leverages the capabilities of the ESP32 microcontroller, MicroPython programming, and machine learning techniques to create an intelligent, event-driven model that predicts rainfall and controls water distribution accordingly. The primary goal of the project is to reduce manual intervention in farming and optimize the use of water resources by automating irrigation decisions based on weather data.

This system incorporates a blend of local environmental sensing and online data retrieval. Sensors like DHT22 (for temperature and humidity), BMP180 (for pressure), and an LDR (Light Dependent Resistor) are interfaced with the ESP32 to collect real-time environmental parameters. The LDR serves as a trigger for prediction events, detecting changes in ambient

light such as sunrise or sunset. Upon trigger activation, the ESP32 initiates data collection from the sensors and fetches additional meteorological values (like wind speed, cloud cover, visibility, dew point, and wind direction) from an online API using Wi-Fi connectivity.

A significant innovation in the system is its use of a machine learning model for rainfall prediction. A logistic regression model is trained offline in Google Colab using historical weather data from the Kalyani region. Instead of deploying the full model, the implementation extracts only the core mathematical elements — specifically, the mean, scale, coefficients, and intercept values — which are embedded in the ESP32 code. These values enable the microcontroller to perform accurate rainfall predictions using the logistic regression formula in MicroPython, without the need for complex model libraries or external computational support.

The system operates entirely within the ESP32 microcontroller. There is no need for a serial link to a PC or server; all logic is self-contained. When a prediction is triggered, the ESP32 uses the collected data to calculate whether rain is likely. If rain is predicted, the water pump remains off, conserving water. If no rain is expected, the ESP32 activates a motor connected to a mini water pump, enabling irrigation. A green LED indicates when the pump is on, while an LCD panel displays real-time weather values and the result of the prediction. Additional LEDs (red and blue) indicate Wi-Fi connection status.

Simulation and testing demonstrated the reliability of the proposed system. The ESP32 successfully fetched data from both sensors and the API, performed accurate predictions, and triggered actions with minimal latency. The prediction was found to be highly responsive, and the integration of the system components — from sensors and prediction to control and display — worked harmoniously. This lightweight, energy-efficient, and cost-effective solution is especially suitable for smart farming and small-scale agriculture, where automation and resource conservation are crucial.

In conclusion, this project showcases how embedded systems and machine learning can be combined to address real-world problems. By deploying ML logic directly on hardware, the system achieves independence, flexibility, and responsiveness — essential traits for next-generation IoT solutions. The project not only highlights technical proficiency in sensor integration, ML deployment, and embedded coding but also reflects a broader vision for sustainable and smart agriculture.

9.2) Key Takeaways:

The development and implementation of the Weather Forecasting and Automatic Watering System in ESP32 using MicroPython offered several valuable insights, both from a technical and practical standpoint. The following key takeaways summarize the most important outcomes of this project:

1. Edge-Based Intelligence is Practical and Effective:

The project demonstrated that complex machine learning logic, such as logistic regression, can be successfully simplified and deployed on a microcontroller like ESP32. This enabled the system to make real-time predictions without relying on cloud computing or a computer connection.

2. Hybrid Data Input Enhances Prediction Accuracy:

Combining local sensor data with real-time weather parameters from an online API enriched the feature set and improved the reliability of the rainfall prediction.

3. Automation Reduces Resource Wastage:

By predicting rainfall and only activating the water pump when needed, the system efficiently conserves water and reduces the need for human intervention—making it a sustainable solution for smart farming.

4. MicroPython is a Viable Platform for IoT Projects:

The use of MicroPython provided a lightweight and efficient way to program the ESP32, allowing easy integration of sensors, Wi-Fi, API handling, and machine learning logic with minimal code overhead.

5. Event-Driven Architecture Improves System Responsiveness:

The use of the LDR sensor as a trigger helped in reducing unnecessary processing and made the system more responsive to actual environmental changes.

6. Cost-Effective and Scalable Design:

The entire system was developed using affordable components without sacrificing performance. Its modular design allows for easy expansion, making it suitable for larger agricultural setups or other automation applications.

7. Real-World Relevance:

The solution addresses a real and growing need in agriculture for intelligent irrigation systems, particularly in water-scarce or remote regions where manual operation is inefficient or impractical.

9.3) Future Scope of Work:

While the current implementation of the weather forecasting and automatic watering system has demonstrated promising results, there are several areas that can be further developed and enhanced in the future.

1. One of the key limitations of this project was the partial dependency on internet-based APIs for certain weather parameters due to the high cost or unavailability of physical sensors. This dependency restricts the system's usability in remote or rural areas with limited or no internet connectivity. In future developments, with access to a full set of onboard sensors (e.g., wind speed, cloud cover), the system can become fully autonomous and functional offline, making it more robust and reliable in diverse geographic locations.

2. Additionally, the current machine learning model utilizes all available features from the dataset to predict rainfall. While this approach is straightforward, it may not always be efficient in terms of computational resources, especially on microcontroller-based platforms. Future work may explore dimensionality reduction techniques such as Principal Component Analysis (PCA) to minimize the number of input features while

retaining the model's predictive accuracy. This would reduce memory usage and processing time, thereby improving performance on low-power devices like the ESP32.

In conclusion, with future improvements in hardware integration, model optimization, and offline capabilities, this project has the potential to evolve into a scalable and intelligent precision agriculture solution, especially beneficial for farmers and gardeners in resource-constrained environments.

Chapter 10: References

1. Ramesh, S. Gupta, and V. Kumar, "Smart Weather Monitoring System Using IoT," International Journal of Engineering Research & Technology (IJERT), vol. 9, no. 5, pp. 1–4, May 2020.
2. S. Sharma, R. Mishra, and A. Jain, "Design and Implementation of IoT Based Weather Monitoring System," in Proceedings of IEEE International Conference on Electronics, Computing and Communication Technologies, pp. 123–126, 2019.
3. R. Patel, M. Thakur, and A. Sinha, "Smart Greenhouse Monitoring System," International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering, vol. 7, no. 12, pp. 7655–7660, 2018.
4. C. Bell, Programming Microcontrollers with Python: Experience the ESP32 using MicroPython, Apress, 2020.
5. R. Gupta, N. Sharma, and A. Saxena, "Weather Forecasting using Machine Learning Algorithms," International Journal of Innovative Science and Research Technology (IJISRT), vol. 5, no. 7, pp. 172–178, 2020.
6. M. Prateek, "ESP32 Based Wireless Weather Station," Hackster.io, 2021. [Online]. Available: https://www.hackster.io/prateek/esp32-weather-station-blynk-40dfe0
7. Bhatt, S. Jaiswal, and K. Pandey, "A Survey on IoT for Environmental Monitoring," International Journal of Innovative Technology and Exploring Engineering (IJITEE), vol. 8, no. 9, pp. 188–192, 2019.
8. S. Yadav, "Cloud-based IoT for Weather Monitoring," International Journal of Scientific Research and Development (IJSRD), vol. 9, no. 1, pp. 45–48, 2021.
9. P. Singh and R. Rana, "Intelligent Weather Monitoring Using AI," International Research Journal of Engineering and Technology (IRJET), vol. 7, no. 4, pp. 1981–1985, 2020.
10. J. Lee, "Sensor Comparison: DHT11 vs DHT22," ElectronicsHub, 2020. [Online]. Available: https://www.electronicshub.org/dht11-vs-dht22
11. Donald Norris, Programming with ESP32 (Getting Started with the Espressif IoT Development Framework), Maker Media, 1st Edition, 2020.
12. Mike Kowalski, ESP32: Programming the ESP32 Wi-Fi and Bluetooth Microcontroller with Arduino, Maker Media, 1st Edition.
13. Agus Kurniawan, MicroPython for ESP32 Development Workshop, Maker Media, 1st Edition, 2018.
14. Charles Platt, Make: Electronics, Maker Media, 2nd Edition, 2015.

15. Paul Scherz and Simon Monk, Practical Electronics for Inventors, McGraw-Hill Education, 4th Edition, 2016.
16. Jeremy Blum, Exploring Arduino, Wiley.
17. Massimo Banzi, Getting Started with Arduino, Maker Media.
18. Robert L. Boylestad, Electronic Devices and Circuit Theory, Pearson.
19. Jonathan Oxer and Hugh Blemings, Practical Arduino: Cool Projects for Open Source Hardware, Apress.
20. Charles Bell, Beginning Sensor Networks with Arduino and Raspberry Pi, Apress.
21. Jonathan Oxer and Hugh Blemings, Practical Arduino, Apress.
22. Marco Schwartz, Internet of Things with ESP8266, Packt.
23. John Boxall, Arduino Workshop, No Starch Press.
24. John-David Warren, Josh Adams, and Harald Molle, Arduino Robotics, Apress.
25. Michael Margolis, Arduino Cookbook, O'Reilly Media.
26. Agus Kurniawan, IoT Projects with Arduino, ESP32, and Raspberry Pi, Apress.