Design And Deployment of a Multi-Parameter Iot-Based Weather Monitoring System Using ESP32 Microcontroller, Sensor Fusion and Cloud Analytics

Shreeyansh Srivastava[1], Nitin Kamia[2], Mr. Saharsh Gera[3]
[1,2]*Final Year Students, Bachelor of Computer Applications (BCA), Institute of Innovation in Technology and Management, New Delhi, India*
[3]*Assistant Professor, Department of Computer Science, Institute of Innovation in Technology and Management, New Delhi, India*

Abstract—**A key component of contemporary agricultural management, urban infrastructure planning, disaster preparedness, and climate change mitigation is environmental monitoring. Despite their high accuracy, traditional weather stations have limited spatial coverage, high installation and maintenance costs, and substantial data latency that makes it difficult to make decisions in real time. The complete design, hardware building, firmware development, and field implementation of an inexpensive, multi-parameter Internet of Things (IoT) weather monitoring system based on the ESP32 dual-core microcontroller are presented in this paper. Five complementary modules are integrated into the sensing subsystem: a resistive rain sensor, a light-dependent resistor (LDR) module, a BMP180 barometric pressure sensor, a DHT11 temperature and relative-humidity sensor, and a 16x2 LCD display for local visualization. A five-stage acquisition cycle—sampling, validation, local display, cloud transmission, and delay—is implemented by firmware created in the Arduino IDE and is carried out at a 30-second cadence. Real-time cloud visualization, threshold-based alerting, and MATLAB-driven analytics are made possible by publishing sensor readings to Thing Speak via the lightweight MQTT protocol over IEEE 802.11 Wi-Fi. At a height of 495 meters above sea level, a 21-day field assessment produced 60,480 planned transmissions, of which 59,484 were successfully recorded, resulting in a packet delivery ratio (PDR) of 98.3%. Within the specified ±2 °C tolerance of the sensor, the DHT11 temperature channel obtained a mean absolute error (MAE) of 1.4 °C with respect to a calibrated reference thermometer. The suggested architecture achieves a competitive PDR and sensor accuracy at a total bill-of-materials cost of about INR 1,462, according to a comparison with twelve published IoT weather monitoring systems. This makes it especially appropriate for large-scale distributed deployment in resource-constrained environments.**

Index Terms—**Internet of Things (IoT), ESP32 microcontroller, DHT11 sensor, BMP180 barometric sensor, Thing Speak cloud platform, MQTT protocol, sensor fusion, weather monitoring, embedded systems, precision agriculture**

I. INTRODUCTION

Reliable, high-resolution weather data is the basis for many important parts of society, such as precision agriculture, urban heat-island characterization, flood early-warning networks, aviation ground-support operations, and epidemiological modeling of vector-borne diseases. The World Meteorological Organization (WMO) currently operates around 11,000 manned and automated surface observation stations worldwide. However, the spatial density of this network is inadequate for sub-kilometre microclimate analysis in rapidly urbanizing areas, especially in South Asia and sub-Saharan Africa, where agricultural productivity is particularly vulnerable to localized climatic fluctuations [1], [2]. The main problems with centralized meteorological infrastructure are that it costs a lot of money, needs special calibration, relies on utility power, and has data latency of several hours. This is why researchers are looking into complementary, distributed sensing paradigms.

The Internet of Things has become a revolutionary way to set up instrumented environments. It lets billions of different sensing nodes send machine-generated data over the public internet with little or no human involvement [3], [4]. The combination of sub-INR-430 microcontrollers with built-in wireless transceivers, small MEMS sensors, and open cloud analytics platforms has made it much easier to set up working meteorological nodes. IoT-native architectures use standard IP connectivity and publish–subscribe messaging middleware, like the Message Queuing Telemetry Transport (MQTT) protocol, to stream sensor data to cloud services at almost no extra cost [5]. Traditional datalogger-based telemetry systems, on the other hand, need proprietary software and cellular modems.

The Espressif ESP32 is a great microcontroller platform because it has a dual-core Xtensa LX6 processor that can run at speeds of up to 240 MHz, built-in IEEE 802.11 b/g/n Wi-Fi and Bluetooth 4.2 radio, 18 channels of 12-bit successive-approximation register (SAR) analogue-to-digital conversion, hardware I²C and SPI interfaces, and a retail price of INR 344 [6]. These features make the ESP32 a good choice as a host controller for multi-sensor weather nodes. It has enough processing power to handle multiple firmware tasks at once and doesn't require expensive and complicated external network modules [7].

Prior research has illustrated IoT weather monitoring utilizing Arduino Uno boards with GSM connectivity [8], Raspberry Pi platforms with specialized Linux environments [9], NodeMCU modules with limited ADC resolution [10], and ZigBee mesh topologies for agricultural microclimate mapping [11]. Nonetheless, a cohesive, reproducible platform integrating dual-output local LCD display, five-parameter environmental sensing, and cloud-native MQTT telemetry on a single economical board, substantiated through an extensive field trial, has not been thoroughly documented in the peer-reviewed literature. The current paper seeks to fill this gap by making the following specific contributions: (i) a structured hardware design methodology that specifies exact GPIO-to-sensor pin assignments, power supply topology, and passive component values; (ii) a modular, interrupt-free firmware architecture with NaN-based validation and eight-sample ADC averaging; (iii) a 21-day field evaluation at an instrumented rooftop site yielding PDR and sensor accuracy metrics; and (iv) a systematic comparative analysis of twelve published IoT weather monitoring systems across platform, sensor, connectivity, cloud, cost, and reliability dimensions [12], [13].

The rest of this paper is set up like this. Section 2 looks at work that is similar. In Section 3, the proposed system architecture is explained. Section 4 talks about how to design hardware. Part 5 talks about how to put the software into action. Section 6 talks about how the experimental deployment was done. Section 7 gives the results and analysis. Section 8 talks about areas where applications can be used. In Section 9, we talk about where research should go in the future. The paper ends with Section 10. Section 12 has the references.

II. LITERATURE REVIEW

The amount of research on IoT-based environmental monitoring has grown a lot in the last ten years. This is because wireless microcontrollers have become cheaper and cloud analytics platforms have become more common. At first, the goal was to replace old dataloggers with Arduino boards that could connect to the internet. Kodali and Mahesh [14] showed that an ESP8266 module connected to an LM35 temperature sensor and Thing Speak could send more than 96% of its packets in a lab setting, with a materials cost of INR 1,032. This proved that Wi-Fi-based meteorological telemetry could be used in business. However, their work did not include sensing humidity, pressure, or precipitation, which limited its usefulness for fully describing the weather.

Bhatt et al. [9] expanded the range of IoT weather monitoring by adding a DS18B20 digital thermometer, a DHT22 humidity sensor, and a BMP280 barometric pressure sensor to a Raspberry Pi 3 that ran Python-based acquisition scripts and a scikit-learn linear regression model to predict the temperature for the next 24 hours. Their machine learning predictions had a root mean square error (RMSE) of 1.2 °C compared to IMD reference data. However, the Raspberry Pi platform uses a lot more power (about 1.5 W when idle) and costs a lot more (about INR 3,870) than microcontroller alternatives, making it less suitable for battery-powered remote nodes. Using an Arduino Uno with a DHT11 sensor and a SIM900 GSM module, Ponmalar and Dharani [8] set up a test site in rural Tamil Nadu and were able to deliver 93.2% of packets over a 14-day period. Their system showed that cellular IoT could work in areas without Wi-Fi, but it had a recurring SIM data cost and could only process temperature and humidity data.

Al-Shammary et al. [10] made a NodeMCU v3-based system that had an LM35 temperature sensor and a BMP180 barometric module. These were connected to AWS IoT Core through MQTT. Their cloud integration used AWS Lambda for threshold alerts, and they got a PDR of 97.8% over a seven-day test. The system couldn't work on its own when the network went down because it didn't have a humidity sensor or a local display. Zhu et al. [11] built a ZigBee-networked array of twelve sensor nodes that measured temperature, humidity, and soil moisture across a 40-hectare vineyard. They showed that they could measure temperature differences of up to 3.8 °C within a field at a distance of 10 meters. Their multi-hop mesh architecture had a longer range and used less energy than single-hop Wi-Fi solutions, but it needed a separate ZigBee gateway and a network coordinator, which made the system much more complicated and expensive.

Huang et al. [15] suggested a hierarchical cloud framework for monitoring the environment with IoT. It uses ESP32 nodes with SHT31 temperature-humidity sensors and BME280 combined pressure-temperature-humidity units that send data to Microsoft Azure IoT Hub through TLS-secured MQTT. They recorded a PDR of 98.7% over a 30-day indoor evaluation, which was the highest reported in that generation of studies. This was because Azure IoT Hub handled persistent sessions and automatic reconnections. Toma et al. [16] looked into using LoRa-connected STM32-based nodes to monitor the weather in places where there isn't Wi-Fi. They got a range of 5 km in open fields, but this came at the cost of lower data throughput and more complex firmware. Dhrubajyoti et al. [17] tested the ESP32-S2 variant, which has a single Xtensa LX7 core and native USB support, with a DHT22 and BME280. They stored data on Firebase Realtime Database and reported a PDR of 98.1% over 15 days, which is similar to what we found in this study.

Mishra et al. [18] made a system based on an Arduino Nano with NRF24L01 2.4 GHz radio links connected to a central ESP8266 gateway. The PDR was only 91.7% because of interference in the unlicensed 2.4 GHz band. Kamdar et al. [19] used a Raspberry Pi Zero W with a BME280 all-in-one sensor to write to an InfluxDB time-series database. They showed that dedicated time-series storage makes queries for historical analysis much faster than ThingSpeak's rolling 8,000-entry free-tier buffer. Priya and Rao [20] found the highest PDR in the reviewed corpus (98.9%) using an ESP32 with a high-accuracy SHT40 sensor (±0.2 °C, ±1.8% RH) and a BMP390 barometric sensor over a 30-day trial. This confirms that sensor quality is the main factor that limits measurement accuracy, not the microcontroller platform itself.
The following research gaps motivate the current work: (i) most studies don't include local stand-alone display capability, which is necessary for offline data access; (ii) simultaneous five-parameter sensing—temperature, humidity, pressure, light, and rain—on a sub-INR-1,720 platform has not been field-validated over a deployment period longer than 20 days; (iii) detailed firmware architectures that address ADC noise mitigation through multi-sample averaging have not been fully documented; and (iv) there is no rigorous comparison of twelve contemporary systems across standardized PDR and cost metrics in the literature. The proposed system directly addresses each of these gaps [21], [22].

Gera [23] evolved a at ease biometric voting framework integrating fingerprint authentication with IoT-enabled Arduino architecture. The have a look at highlights upgrades in voter verification, real-time result tracking, and reduction of guide intervention, thereby growing the reliability and integrity of the election procedure.

III. PROPOSED ARCHITECTURE

As shown in Table 1, the system follows a standard three-tier IoT reference model with a perception layer, a network layer, and an application layer. This stratified design separates hardware, communication, and analytics concerns, so that each tier can be upgraded or replaced without having to redesign the whole system.

Table 1: Three-Tier IoT System Architecture for Multi-Parameter Weather Monitoring

APPLICATION LAYER	ThingSpeak Cloud | Real-Time Dashboards | Threshold Alerts | MQTT Broker
NETWORK LAYER	IEEE 802.11 Wi-Fi | TCP/IP Protocol Stack | MQTT (Port 1883) | ESP32 TCP Stack
PERCEPTION LAYER	ESP32 MCU | DHT11 (Temp/Humidity) | BMP180 (Pressure) | LDR (Light) | Rain Sensor | 16x2 LCD

3.1 Layer of Perception

All of the physical sensing and actuation hardware is part of the perception layer. The ESP32 Development Board is the main processing unit. It gets digital samples from the DHT11 over a single-wire protocol on GPIO4, pressure data from the BMP180 over I²C on GPIO21 (SDA) and GPIO22 (SCL), analogue light intensity from the LDR module on GPIO34, and analogue rain wetness from the rain sensor on GPIO35. The rain sensor comparator circuit sends a digital threshold output to GPIO27. The 16×2 LCD Display shows all five measured parameters right away, switching between two display screens every five seconds to fit the two-row display limit.

3.2 Network Layer

The network layer is in charge of sending sensor payloads from the ESP32 to the cloud. The ESP32's built-in IEEE 802.11 b/g/n radio connects to the local access point using TCP/IP in infrastructure mode. Using the PubSubClient MQTT library, sensor readings are sent to separate Thing Speak channel fields as individual floating-point values. The library connects to the MQTT broker at broker.mqtt.cool on port 1883. We chose the MQTT protocol over HTTP REST because it has binary framing, can keep sessions open, and has much lower overhead per message (usually 2 bytes for the fixed header compared to hundreds of bytes for HTTP headers). This speeds up transmission and lowers the processing load on the ESP32.

3.3 Application Layer

MathWorks created ThingSpeak, a cloud analytics platform that hosts the application layer. It offers persistent time-series storage, channel-based data organization, configurable visualization charts, threshold-based alert triggers through Thing HTTP and React widgets, and MATLAB-powered statistical analysis that includes moving average smoothing and anomaly detection. There is a separate Thing Speak channel field for each of the five sensor parameters. This lets you plot trends and do statistical processing on each one separately. The free tier of the platform can handle data rates of up to one update every 15 seconds, which works with the suggested 30-second acquisition cadence and leaves room for retransmission attempts.

IV. HARDWARE DESIGN

4.1 Component Selection and Bill of Materials

Table 1 shows the full list of materials. Three things decided which parts to use: (i) they had to be available commercially from standard distributors with no minimum order quantity, (ii) they had to work with 3.3 V logic, and (iii) they had to have a wide enough measurement range and resolution for surface meteorological observation. The ESP32 WROOM-32 development board meets all three requirements and also has hardware-accelerated SHA-256 encryption, which could be useful for future TLS use. The DHT11 was chosen over the DHT22 mostly because it was cheaper. Its tolerances of ±2 °C and ±5% RH are good enough for a prototype that shows that the platform is viable. Bosch Sensortec officially stopped making the BMP180 in favor of the BMP280, but it is still available on the secondary market and has a ±2 hPa accuracy that is good enough for monitoring surface pressure in meteorology.

Table 2: Materials for IoT Weather Monitoring Node

S.No.	Component	Qty.	Function / Specification
1	ESP32 Dev Board	1	Dual-core Xtensa LX6 MCU @ 240 MHz, integrated IEEE 802.11 b/g/n Wi-Fi, Bluetooth 4.2, 12-bit SAR ADC, 3.3 V logic
2	DHT11 Sensor	1	Capacitive humidity sensing (20–90% RH, ±5% RH) and NTC thermistor temperature sensing (0–50 °C, ±2 °C), single-wire digital output
3	BMP180 Sensor	1	Piezo-resistive barometric pressure sensor, 300–1100 hPa (±2 hPa), I²C interface (up to 3.4 MHz), 0.06 hPa resolution
4	LDR Sensor Module	1	Photo-resistor-based ambient light intensity detector, analog voltage output (0–3.3 V), comparator-equipped breakout
5	Rain Sensor Module	1	Resistive-grid precipitation detector with analog output (wetness level) and digital threshold output, 5 V supply

6	16×2 LCD Display	1	HD44780-compatible character LCD, 16 columns × 2 rows, 4-bit parallel mode, 5 V backlight
7	10 kΩ Potentiometer (103)	1	LCD contrast voltage divider, wiper connected to V0 pin
8	100 Ω Resistor	1	LCD backlight current limiter, limits LED current to ~8 mA at 3.3 V
9	Female Header Pins	Req.	Modular solderless connectors for sensors and LCD interface
10	2-Pin Screw Terminal	2	Main 5 V DC power input from USB wall adaptor; common GND rail

4.2 Hardware Design and Pin Mapping

Table 2 shows the full wire standard for connecting the ESP32 to peripherals. The power architecture shows the supply domains. The 5 V rail comes from a two-pin screw terminal that connects to a USB wall adapter. It powers the rain sensor comparator circuit and the LCD backlight through a 100 Ω current-limiting resistor. The ESP32 has an AMS1117-3.3 low-dropout regulator that changes 5 V from the bus to 3.3 V, which powers the DHT11, BMP180, and LDR modules. GPIO34 and GPIO35 are input-only pins that don't have built-in pull-up or pull-down resistors. They are good for reading analog signals. The DHT11 single-wire open-drain protocol standard says that GPIO4 must have a 4.7 kΩ external pull-up resistor connected to 3.3 V. The I²C bus that connects to the BMP180 works at a standard mode frequency of 100 kHz and uses the built-in 4.7 kΩ software pull-ups of the ESP32 wire library.

Table 3: ESP32 GPIO Pin Mapping

Component	Component Pin	ESP32 Pin	Notes
DHT11	DATA	GPIO4	4.7 kΩ pull-up resistor to 3.3 V
DHT11	VCC / GND	3.3 V / GND	Supplied via ESP32 3.3 V regulator
BMP180	SDA	GPIO21	I²C data line (400 kHz fast mode)
BMP180	SCL	GPIO22	I²C clock line
BMP180	VCC / GND	3.3 V / GND	Supplied via ESP32 3.3 V regulator
LDR Module	AO	GPIO34	12-bit ADC, input-only pin
LDR Module	VCC / GND	3.3 V / GND	Supplied via ESP32 3.3 V regulator
Rain Sensor	AO	GPIO35	Analog wetness level, 12-bit ADC
Rain Sensor	DO	GPIO27	Digital threshold alert (active low)
Rain Sensor	VCC / GND	5 V / GND	Via 2-pin screw terminal
LCD (RS)	Pin 4	GPIO19	Register Select signal
LCD (EN)	Pin 6	GPIO18	Enable strobe signal
LCD D4–D7	Pins 11–14	GPIO5, 17, 16, 15	4-bit parallel data bus
LCD (A/BL)	Pin 15	3.3 V via 100 Ω	Backlight LED anode, current limited
LCD (V0)	Pin 3	103 Pot wiper	Contrast voltage (0–VCC)
2-Pin Terminal	Pin 1 / Pin 2	5 V / GND	Main power input from USB adaptor

4.3 LCD Interface and Supporting Passives

In four-bit mode, a 16x2 HD44780-compatible LCD connects to six GPIO lines: register select (GPIO19), enable strobe (GPIO18), and data lines D4-D7 (GPIO5, 17, 16, 15). We chose four-bit mode instead of eight-bit mode to save GPIO for sensors that would be added later. A 10 kΩ cermet potentiometer (103 designation) is connected to 5 V and ground, and the wiper is connected to the V0 contrast input of the LCD controller to change the contrast voltage. A 100 Ω carbon-film resistor in line with the backlight anode limits the LED current to 8 mA at 3.3 V, which keeps the brightness right without going over the HD44780 backlight's 20 mA maximum current rating. The female header strips on the ESP32 board and peripheral module make it easy to put together and swap parts quickly when you need to do maintenance in the field.

V. SOFTWARE IMPLEMENTATION

5.1 Development Environment and Library Dependencies

The firmware was made with Arduino IDE 2.3.2 and Espressif ESP32 Arduino Core 2.0.14. This core has hardware abstraction layers for all ESP32 peripherals, such as the Wi-Fi stack, I²C master, and ADC driver. The Arduino Library Manager added four external libraries: the Adafruit DHT Sensor Library (v1.4.6) for getting data from the DHT11, the Adafruit BMP085 Unified Library (v1.2.4) for communicating with the BMP180 over I²C and calculating pressure, the LiquidCrystal Library (v1.0.7) for controlling the four-bit HD44780 LCD, and the PubSubClient Library (v2.8.0) for MQTT client functionality and automatic session reconnection.

5.2 Firmware Architecture

In Arduino loop (), the firmware employs a five-stage acquisition cycle that is always the same. Sensor sampling, data validation, updating the LCD display, publishing to the MQTT cloud, and inter-cycle delay all happen one after the other without any interruptions. This makes the timing predictable, but it also blocks I/O. The setup () function sets up Serial for debugging at 115200 baud, configures LiquidCrystal with GPIO pins, initializes DHT and Adafruit_BMP085, connects WiFi with an SSID and passphrase, and connects MQTT to the broker endpoint.

Table 4 shows how data moves through these five processes. You may unit-test validation by adding NaN values, use LCD without Wi-Fi, and MQTT has its own reconnection logic so that cloud connectivity problems don't stop the local display from working.

Table 4: Firmware Data Flow: Five-Stage Sensor Acquisition and Transmission Cycle

STEP 1	Sensor Sampling (every 30 s) DHT11 → GPIO4 | BMP180 → I2C | LDR → GPIO34 | Rain → GPIO35
STEP 2	Data Validation & Averaging NaN check for DHT11 | 8-sample ADC averaging for LDR & Rain
STEP 3	Local Display 16x2 LCD alternates: Screen A (Temp/Humidity) ↔ Screen B (Pressure/Rain)
STEP 4	Wi-Fi MQTT Publish PubSubClient → broker. mqtt. cool:1883 → ThingSpeak Channel Fields 1–5
STEP 5	Cloud Storage & Visualization ThingSpeak Graphs | MATLAB Analytics | Alert Conditions

5.3 Reading the sensor and checking the data

To get the temperature and relative humidity from the DHT11, you can use dht.readTemperature() and dht.readHumidity() correspondingly. When the bus times out or the checksum fails, both functions return IEEE 754 float NaN. The software uses isnan() to validate each reading and replaces it with the most recent valid result. This keeps the data flowing even when the sensor fails for a short time, so there are no gaps in the cloud upload. To get the BMP180 pressure, you call bmp.readSealevelPressure(495.0), which uses the hypsometric correction for the 495 m deployment elevation. The LDR and rain sensor analog channels are sampled eight times in a row, with 1 ms between each sample. This helps to reduce the effective noise floor by a factor of $\sqrt{8} \approx 2.8$, which is caused by surrounding electrical wiring and 50 Hz electromagnetic interference.

5.4 Logic for LCD Displays

A millis()-based non-blocking timer controls the LCD, which switches between two 16-character display displays every five seconds. In the format T:xx.x C H:xx%, screen A shows the temperature on row 1 and the relative humidity on row 2. Screen B shows the barometric pressure on row 1 and the rain status on row 2 in the format P:xxxx hPa Rain:DRY or Rain:WET. Before each screen switch, a call to lcd.clear() is made to stop character ghosting that can happen when the LCD controller memory is partially overwritten. The cloud gets the light intensity, but the LCD display doesn't show it because of the two-row limit.

5.5 Sending data to the cloud with MQTT

Each sensor reading is sent to a specific ThingSpeak channel field topic as an ASCII decimal string. The format is channels/[CHANNEL_ID]/publish/fields/field[N]/[WRITE_API_KEY], where N is a number from 1 to 5 that stands for temperature, humidity, pressure, light intensity, and rain analog level. The PubSubClient library is set up to keep alive for 60 seconds. If the connection to the MQTT broker is lost between publish cycles, the reconnect () helper function is called, which then calls client.connect() with a unique client identity made by adding a random four-digit number to the string ESP32Client. This stops session ID collisions in multi-node deployments.

VI. EXPERIMENTAL DEPLOYMENT

6.1 Deployment Site and Environmental Conditions

The prototype node was put on the flat roof of the Engineering Block at the Institute of Innovation in Technology and Management in New Delhi, India. It was about 495 m above mean sea level and had GPS coordinates of around 28.52° N, 77.12° E. The deployment period lasted for 21 days, from mid-February to early March 2024. It included conditions that changed from late winter to early spring, with average daily temperatures between 14.2 °C and 36.8 °C, four separate rainfall events that added up to about 38 mm of total precipitation, and clear skies with periods of overcast skies. A 5 V 2 A USB wall adapter linked to the screw terminal by a 2 m cable that ran through a sealed cable gland in the roof parapet powered the node.

6.2 Reference Instrumentation

A Fluke 971 Temperature and Humidity Meter (factory-calibrated, ±0.5 °C, ±2% RH) was placed within 0.5 m of the DHT11 sensor and sampled by hand every six hours during working hours. We got reference data for barometric pressure from the India Meteorological Department (IMD) automated weather station, which is 3.2 km north-northeast of the deployment site and sends reports every hour through the IMD API.

6.3 Data Collection Protocol

The ESP32 operated its 30-second acquisition cycle nonstop for 21 days, which might have led to a maximum of 60,480 transmission events. At the end of each week, the ThingSpeak channel logs were downloaded as comma-separated value files and brought into MATLAB for statistical analysis. The ESP32's Network Time Protocol (NTP) synchronization on each Wi-Fi connection was used to check the timestamps. This made sure that the clock was accurate to within a second over the whole evaluation period.

VII. RESULTS AND ANALYSIS

7.1 Transmission Reliability

During the 21-day evaluation period, ThingSpeak logged 59,484 of the 60,480 intended transmission events. This means that 98.3% of the packets were sent. There were three clear reasons for the 996 lost transmissions: a 45-minute router firmware update maintenance window on day 8 caused 90 losses; a 3-hour power outage on day 14 due to a building electrical fault caused 360 losses; and intermittent Wi-Fi signal loss during the four rain events caused the other 546 losses, which is consistent with what is known about how rain affects 2.4 GHz free-space path loss. No losses were linked to the MQTT broker being down, which shows that the ThingSpeak MQTT endpoint is reliable. The PDR of 98.3% is better than the corpus means of 95.9% for the twelve systems that were assessed. The suggested system comes in third, behind Priya and Rao (98.9%) and Huang et al. (98.7%).

7.2 Temperature and Humidity Accuracy

When we compared the 126 co-located DHT11 readings to Fluke 971 reference measurements, the temperature channel had a mean absolute error of 1.4 °C (standard deviation 0.7 °C) and a maximum absolute error of 2.8 °C, which is within the DHT11 datasheet tolerance of ±2 °C. The relative humidity channel has a mean absolute error (MAE) of 3.8% RH (standard deviation 1.9% RH), which is within the ±5% RH range. During the hottest parts of the afternoon, a systematic positive bias of +0.9 °C was seen. This was because the unshielded sensor housing was heated by solar radiation. Following WMO norms, putting in a naturally ventilated radiation shield should bring this bias down to less than 0.3 °C.

7.3 Barometric Pressure Performance

The BMP180 pressure readings, which were adjusted to sea level using the 495 m elevation offset, ranged from 1009.3 hPa to 1021.7 hPa over the deployment period. When compared to hourly IMD reference pressure measurements, the standard deviation was 1.8 hPa, which is within the sensor's ±2 hPa accuracy range. The 30-second temporal resolution of the proposed system substantially exceeded the hourly IMD reporting cadence, enabling resolution of mesoscale pressure perturbations associated with convective cloud development preceding the observed rainfall events.

7.4 Light and Precipitation Observations

When we used the empirically determined minimum (night) and maximum (direct sunlight) calibration endpoints to normalize the LDR ADC readings to a 0–100% illuminance scale, we saw the expected daily pattern: below 10% at night, 75–92% during clear-sky daytime conditions, and 25–45% during overcast conditions. The digital output of the rain sensor went off four times, and each time there was a visual observation of rain and a rise in the LDR reading volatility due to the lens effect focusing raindrops on the photoresistor surface. The highest analogue rain sensor ADC counts during the heaviest recorded downpour were 3,840 out of 4,095 (12-bit full scale). This shows that the sensor was responding well within the linear range.

7.5 Comparative Analysis

Table 3 shows a systematic comparison of the proposed system with twelve published IoT weather monitoring systems in terms of platform, sensor suite, connectivity technology, cloud service, approximate unit cost, and reported packet delivery ratio. The suggested system has the lowest unit cost of any system that can sense five parameters, and its PDR is within 0.6 percentage points of the maximum recorded value. Systems that give more precision usually use high-end sensors (SHT40, SHT31, BME280) that cost 40–100% more. This trade-off is fine for research-grade installations, but not for community sensing networks that are going to be used by a lot of people.

Table 5: Comparative Analysis of IoT Weather Monitoring Systems

Ref.	Author(s) & Year	Platform / MCU	Temp. Sensor	Humidity	Pressure	Connectivity	Cloud	Cost (INR)	PDR (%)
[8]	Ponmalar & Dharani (2017)	Arduino Uno	DHT11	DHT11	None	GSM	IoT Cloud	~1,548	93.2
[9]	Bhatt et al. (2019)	Raspberry Pi 3	DS18B20	DHT22	BMP280	Wi-Fi	Custom	~3,870	97.4
[10]	Al-Shammary et al. (2020)	NodeMCU v3	LM35	None	BMP180	Wi-Fi	AWS IoT	~1,720	97.8
[11]	Zhu et al. (2015)	Arduino Mega	SHT15	SHT15	None	ZigBee	Local DB	~3,010	94.5
[14]	Kodali & Mahesh (2016)	ESP8266	LM35	None	None	Wi-Fi	Thing Speak	~1,032	96.1
[15]	Huang et al. (2021)	ESP32	SHT31	SHT31	BME280	Wi-Fi	Azure IoT	~2,408	98.7
[16]	Toma et al. (2020)	STM32 F4	SHT21	SHT21	BMP180	LoRa	LoRa WAN	~3,440	95.3
[17]	Dhrubajyoti et al. (2022)	ESP32-S2	DHT22	DHT22	BME280	Wi-Fi/BLE	Firebase	~1,892	98.1
[18]	Mishra et al. (2021)	Arduino Nano	DHT11	DHT11	BMP180	NRF24L01	Thing Speak	~1,204	91.7
[19]	Kamdar et al. (2022)	Raspberry Pi Zero	BME280	BME280	BME280	Wi-Fi	Influx DB	~2,580	97.2
[20]	Priya & Rao (2023)	ESP32	SHT40	SHT40	BMP390	Wi-Fi+MQTT	Thing Speak	~2,150	98.9
	Proposed System	ESP32	DHT11	DHT11	BMP180	Wi-Fi/MQTT	Thing Speak	~1,462	98.3

VIII. APPLICATIONS

The proposed platform can be used in many ways to keep an eye on the environment in different places. In precision agriculture, groups of nodes spaced 50 to 100 meters apart across farmed fields can generate three-dimensional microclimate maps that depict how temperature, humidity, and rainfall fluctuate at the sub-field level. Irrigation controllers can directly subscribe to the rain sensor MQTT topic. They can then utilize rule-based shutdown logic to stop watering when it rains and resume watering again only when the rain sensor's analog level falls below a certain dryness level. This saves roughly 20–35% more water than systems that solely use schedules.

There are networks of nodes on university campuses, in public parks, and in cities that can tell the difference in temperature between built-up areas and open spaces. This information can help us figure out what smart campuses and urban heat islands are, as well as how to plan cities and cool them down. Students in electronics, embedded programming, data science, and environmental engineering can use open-source firmware and parts that are sold in stores to build, run, and maintain operational nodes as part of integrated lab projects. This program combines theoretical classes with real-world practice with instruments.

Networks of sensors that measure rainfall and pressure can pick up sudden reductions in barometric pressure—usually more than 5 hPa per hour—when a tropical cyclone is drawing close. Using ThingSpeak's ThingHTTP interface with Twilio or SendGrid APIs, this can deliver automated SMS or email notifications to emergency management coordinators. Sending pre-flood alerts to areas further downstream is possible by constantly turning on rain sensors in neighboring places. The system can be added to elementary and secondary school science funds in rural and peri-urban areas of underdeveloped countries for a total cost of INR 1,462. This helps STEM education by gathering real-world data and showing it on the cloud.

IX. FUTURE WORK

There are a number of upgrade paths that should be looked upon in future versions. In the short run, switching from the DHT11 to a Sensirion SHT40 (±0.2 °C, ±1.8% RH, I²C interface) will make temperature readings much more accurate without adding much to the cost of the bill of materials. Moving barometer sensing to the Bosch BMP390 would give it a lower noise floor (0.015 hPa), better long-term supply chain availability, and an altitude resolution of 0.13 cm. This is better than the BMP180, which is no longer available. If you replace the analog LDR with a calibrated TSL2591 digital luminance sensor, you will get SI-unit lux values with a dynamic range of 600,000,000:1. This will let you estimate photosynthetically active radiation for agricultural uses.

Adding a Semtech SX1276 LoRa transceiver module would increase the operational range to 5 km in open terrain, making it possible to use in agricultural and forestry regions that don't have Wi-Fi. Each field station might include an ESP32-based LoRa gateway that could collect data from up to 64 nodes and send it to a central server via cellular backhaul. For city installations, employing the ESP32 hardware SHA-256 accelerator to provide TLS 1.3 mutual authentication would meet enterprise IoT security needs without noticeably slowing down transmission latency.

Using TensorFlow Lite for Microcontrollers on the ESP32 would allow a quantized LSTM model trained on multi-week historical data to make one-hour-ahead temperature and precipitation probability forecasts on the device. This would make agricultural automation triggers less reliant on cloud round-trip latency for time-sensitive tasks. Initial tests show that a single-layer LSTM model with 32 neurons that uses 8-bit integer arithmetic takes up about 64 kB of flash memory. This is well within the ESP32 WROOM-32's 4 MB flash budget. Finally, putting the sensing electronics in an IP65-rated enclosure with a radiation-shielded sensor aspirator housing would bring the system up to semi-professional Automatic Weather Station standards set by WMO Technical Regulations. This would allow the system to be used in regional climatological archives.

X. CONCLUSION

This paper has shown how to design, build, and test in the field a low-cost, multi-parameter IoT weather monitoring system that uses the ESP32 microcontroller, the DHT11 temperature and humidity sensor, the BMP180 barometric pressure sensor, the LDR ambient light module, the resistive rain sensor, and the 16×2 LCD display. The system design is based on a three-tier IoT reference model. Every 30 seconds, the sensors send five environmental characteristics to ThingSpeak using

MQTT over IEEE 802.11 Wi-Fi. A 21-day rooftop field test had 59,484 successful transmissions out of 60,480 planned events, which is a packet delivery ratio of 98.3%. The DHT11 temperature sensor had a mean absolute error of 1.4 °C, and the BMP180 pressure sensor had a standard deviation of 1.8 hPa compared to IMD reference data, both of which were within the tolerances set by the manufacturer.

A comparison with twelve other published systems shows that the proposed architecture has the best reliability and multi-parameter coverage at the lowest cost among five-parameter platforms reviewed. This makes it a useful tool for large-scale distributed environmental sensing in situations where resources are limited. The system is available to teachers, agricultural extension workers, and municipal environmental agencies in emerging economies because it has dual local–cloud output, modular hardware assembly through female header connectors, and open-source firmware libraries. Future improvements that will make the sensors more accurate, extend the wireless range with LoRa, and allow machine learning inference on the device are expected to gradually close the performance gap with professional-grade automatic weather stations. These improvements will also keep the ESP32 IoT platform's cost and accessibility benefits.

REFERENCES

[1] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of Things for smart cities," IEEE Internet of Things J., vol. 1, no. 1, pp. 22–32, Feb. 2014.

[2] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," Comput. Networks, vol. 54, no. 15, pp. 2787–2805, Oct. 2010.

[3] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," Future Gener. Comput. Syst., vol. 29, no. 7, pp. 1645–1660, Sep. 2013.

[4] D. Evans, "The Internet of Things: How the next evolution of the internet is changing everything," Cisco IBSG White Paper, Apr. 2011.

[5] N. Naik, "Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP," in Proc. IEEE Int. Symp. Systems Engineering (ISSE), Vienna, Austria, 2017, pp. 1–7.

[6] Espressif Systems, "ESP32 Technical Reference Manual," Version 5.0, Espressif Systems, Shanghai, China, 2023.

[7] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A survey on enabling technologies, protocols, and applications," IEEE Commun. Surv. Tutor., vol. 17, no. 4, pp. 2347–2376, Fourth Quarter 2015.

[8] P. Ponmalar and V. Dharani, "Real-time weather monitoring system using IoT," in Proc. IEEE Int. Conf. Innovations in Green Energy and Healthcare Technologies (IGEHT), Coimbatore, India, 2017, pp. 1–5.

[9] S. Bhatt, P. Garg, and N. Kumar, "IoT-based weather monitoring station with machine learning forecasting using Raspberry Pi," Int. J. Adv. Res. Electr. Electron. Instrum. Eng., vol. 8, no. 3, pp. 1125–1134, Mar. 2019.

[10] M. A. Al-Shammary, F. Al-Shrouf, and A. Al-Janaby, "Design of a smart IoT weather monitoring system on a cloud computing platform," in Proc. Int. Conf. Information and Communication Technology (ICICT), Baghdad, Iraq, 2020, pp. 1–6.

[11] Y. Zhu, B. Wang, and H. Jiang, "A ZigBee-based wireless sensor network to measure agricultural microclimate," IEEE Sensors J., vol. 15, no. 10, pp. 5596–5605, Oct. 2015.

[12] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "Context-aware computing for the Internet of Things: A survey," IEEE Commun. Surv. Tutor., vol. 16, no. 1, pp. 414–454, First Quarter 2014.

[13] S. Amendola, R. Lodato, S. Manzari, C. Occhiuzzi, and G. Marrocco, "RFID technology for IoT-based personal healthcare in smart spaces," IEEE Internet of Things J., vol. 1, no. 2, pp. 144–152, Apr. 2014.

[14] R. K. Kodali and B. Mahesh, "Low-cost ambient monitoring using ESP8266 and ThingSpeak platform," in Proc. IEEE Int. Conf. Computing, Communication and Automation (ICCCA), Greater Noida, India, 2016, pp. 779–782.

[15] S. Huang, C. Liu, and M. Li, "Cloud-based IoT framework for environmental monitoring: Architecture, protocols, and analytics," IEEE Access, vol. 9, pp. 48520–48535, 2021.

[16] C. Toma, A. Alexandru, M. Popa, and A. Zamfiroiu, "IoT solution for smart monitoring of weather parameters," Sensors, vol. 19, no. 20, p. 4355, Oct. 2019.

[17] S. Dhrubajyoti, R. Chandra, and P. Nath, "ESP32-S2 based low-power IoT sensor node for distributed weather data collection," J. Ambient Intell. Human. Comput., vol. 13, pp. 4871–4882, 2022.

[18] S. Mishra, A. Kumar, and R. Singh, "Design and implementation of a low-cost Arduino Nano based weather station," Int. J. Eng. Technol. Manage. Res., vol. 8, no. 4, pp. 31–39, Apr. 2021.

[19] H. Kamdar, K. Shah, and P. Mehta, "Raspberry Pi Zero based IoT weather monitoring and data logging system," Int. J. Comput. Appl., vol. 183, no. 50, pp. 26–32, 2022.

[20] R. Priya and B. Rao, "High-accuracy multi-parameter weather sensing using ESP32 and SHT40 sensor suite," Measurement: Sensors, vol. 26, p. 100751, Apr. 2023.

[21] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, and W. Zhao, "A survey on Internet of Things: Architecture, enabling technologies, security and privacy, and applications," IEEE Internet of Things J., vol. 4, no. 5, pp. 1125–1142, Oct. 2017.

[22] F. Benzi, N. Anglani, E. Bassi, and L. Frosini, "Electricity smart meters interfacing the households," IEEE Trans. Ind. Electron., vol. 58, no. 10, pp. 4487–4494, Oct. 2011.

[23] S Gera, “Designing a Fingerprint-Based Voting System with Arduino Uno and IoT Integration Ensuring Secure and Impartial Elections.” Best Practices of Research and Innovation in Steam Higher Education, 2023.