INTERNET OF THINGS

An IoT-Enabled Multisensor Air Quality Monitoring System with Edge-Based AQI Computation and Intelligent Mobile Analytics

T N. Nihar, Amith C. Patil, B Nithya Shree, P S. Varun, Bhagyashri Wakde

Department of Computer Science and Engineering, Rajiv Gandhi Institute of Technology, Bengaluru, India

Published November 2026

Abstract

Inexpensive air quality monitors based on microcontroller units (MCUs) generally end at sensor data collection and numerical displays, with no further interpretation of data meaning and possible actions left to the user. The paper describes a multisensor air quality monitoring system combining an ESP32-WROOM MCU, DHT22 temperature humidity sensor, analog MQ-series gas sensor, infrared optical dust sensor and a NEO-6M GPS module together with a custom cross-platform Flutter application. Unlike cloud-based IoT systems, which depend on an active Wi-Fi/cloud connection, the sensor node communicates with the application over a dedicated Bluetooth Classic serial connection sending newline-delimited JSON payloads, which are processed by the application to display live data in a dashboard including an overall computed air quality index in addition to individual sensor values and GPS status. The application includes a feature to send an on-demand analysis of the current reading set to a cloud-based generative-AI model (Google Gemini) using a constrained prompt format and receive back a structured non-diagnostic report of environmental risks, potential causes, precaution and short-term forecast. The functional tests have confirmed the consistency of JSON framing for both USB-serial and Bluetooth-serial connections as well as correct response of the system's air quality classification algorithm from a low pollutant baseline reading to high particulates test scenario. Overall, a combination of Bluetooth serial link from a device to a phone and cloud-based generative-AI model analysis proves to be an inexpensive and feasible alternative to IoT cloud infrastructure for contextual air quality information delivery.

Keywords: Air Quality Monitoring; Bluetooth; ESP32; Flutter; Generative AI

1 Introduction

Human health, comfort, and productivity are affected by the quality of air and environment [11, 12]. Environmental conditions such as air quality have a quantifiable impact on human being's physical well-being, comfort, and productivity, thus, the issue of continuous local environmental monitoring has become a matter of engineering concern as seen by a Raspberry-Pi-powered air-quality monitoring web node [1], a weather reporting node using ESP32 [2], an ESP32 smart environmental monitor [3], along with a weather station using ESP32 [4]. Variables like temperature, humidity, particulate matter content, and concentrations of gases are not constant in time scale and geographical space and hence values obtained from some selected central weather stations may not be sufficiently detailed to provide information on a particular room, car, or street [4]. This gap creates the need for cost-effective sensors that are able to monitor the environment in its proximity.

Existing technologies used to gauge local environmental awareness include manual gauging and the use of single-purpose analog gauges. However, such technologies cannot support continuous multi-

factor measurements, nor do they provide any means for automatic analysis of the data when changes occur. The use of electronic devices that simply show the results on the small screen of the gadget does not solve the problem, since no information is provided on how to interpret the results.

The reduction in price of system-on-chip microcontrollers that come with built-in wireless communications has increased the capabilities of hobbyist-level monitoring nodes. For instance, an ESP32/DHT22 node is able to communicate with the Blynk system using temperature and humidity data that it is able to collect due to having all the required components on board, including dual-core CPU, Wi-Fi and Bluetooth antennas, and analog/digital I/O pins [2], ESP32 device using DHT11, MQ-7, and MQ-135 sensors along with OLED and LED display [3], and a station for ESP32 WROOM DEVKIT made up of a climate sensor BME280, a gas sensor MQ-135, and a UV sensor ML8511 [4]. Used in conjunction with cloud-based IoT services like ThingSpeak, Blynk, or Google Sheets/Email-based services, this has enabled the creation of low-cost real-time monitoring nodes at a far lower cost than commercial instruments as demonstrated by a Raspberry Pi-based node based on the ThingSpeak cloud-based service [1].

However, one issue which has plagued these systems is that their utility lies only up to the level of data collection and presentation. Most of these systems display either the data from the sensors or the color depending on the threshold. But no system described in Section 2 offers an interpretation of the sensed environment, which could help in understanding what should be done based on this information.

Narrowing this gap does not necessarily need to involve a machine learning model running on the device itself. A separate application able to encode the state of the sensors at hand into a narrowly defined natural language query, and direct that query to an appropriate cloud-based generative model as needed, is able to provide contextually-relevant guidance in natural language without the overhead associated with a custom-trained classifier, so long as the query and response are appropriately constrained such as not to invent readings or diagnose conditions.

In this paper, we propose an air quality monitoring solution which is based on the following three components: (i) the sensing node built around the ESP32 device with a DHT22, analog MQ series gas sensor, infrared optical dust sensor, and GPS, computing the air quality index locally and outputting a timestamped JSON over a Bluetooth Classic serial connection; (ii) a cross-platform Flutter application for parsing the data stream and displaying it in real-time in a dashboard format; and (iii) an on-the-fly analysis component where the application makes use of a cloud-based generative AI model (Gemini) to submit the current readout set via a prompt template to generate an environment-related non-diagnostic report. Our contributions in this paper include: (1) Bluetooth-enabled ESP32 sensing pipeline delivering the payload JSON data validated via a USB-serial connection as well as the mobile application parser; (2) a Flutter application design which combines live multi-sensor visualization, GPS connectivity, and AI-driven interpretation into one compact cross-platform framework; (3) a performance evaluation of the entire pipeline on a low-pollutant baseline as well as the particulate-increased test scenario, with an explicit discussion of the sensing and connectivity limitations observed during testing.

2 Literature survey

All environmental monitoring systems described in the scientific literature follow a similar concept: the microcontroller or single board computer that is connected to a low-cost sensor suite, wireless communication to send data, and either cloud computing or widgets for storing and displaying data.

The variations among these systems arise primarily from the selection of the processing platform, sensor suite, communication method to leave the node, and finally, whether any sort of data interpretation has been carried out prior to the data reaching the end user.

The use of Raspberry Pi computers is preferred due to the processing power of a complete single-board computer. Balasubramaniyan and Manivannan [1], created an IoT node based on the Raspberry Pi with a GrovePi+ Shield that integrates the DHT, dust, and MQ family of gas sensors and streams sensor data to the cloud through Wi-Fi/HTTP communication; MATLAB software executed on the cloud side sent notifications via Twitter. This system showcases the advantage of conducting computations in the cloud instead of the IoT node; however, it is constrained by a single social media notification service, a maximum of four nodes, and lacks a client application.

Many of the ESP32-based systems focus on cost savings and energy savings, but all of them depend on the consistent connection to the cloud through Wi-Fi. Adeagbo [2] paired the ESP32 with the DHT22 and the Blynk Internet of Things (IoT) framework to send the readings on temperature and humidity using a mobile widget. The results were validated using the official city weather data, which indicated that humidity readings were reasonably accurate while there was an error of around 20% for temperature readings. In Omkar et al. [3], the list of sensors on the ESP32 node has been extended to incorporate DHT11, MQ-7, and MQ-135 gas sensors with OLED screen, LED status indicators, and Blynk as an output device; this is a more effective solution on the node side but still uses a generic cloud-based widget without any additional interpretation layer. Uddin and Praharaj [4] combined the BME280 Climate Sensor, MQ-135 Gas Sensor, and ML8511 UV Sensor on ESP32 WROOM DEVKIT that uploads data on Google Sheets and HTML web dashboard as well as mirrors data on Blynk; however, the email alert feature claimed by the project is yet to be tested, and the user interface is still limited to web dashboard and not native app on any platform. Three limitations recur in these systems. First, every design reviewed still needs a steady Wi-Fi link to push data to its cloud, or to whatever widget platform it uses, and there is no backup route if a node ends up outside network coverage. Second, the part people see is usually either a third-party widget service like Blynk, a static web page, or an LED/OLED display, rather than a purpose-built cross-platform application. Third, and most importantly for this work, none of the surveyed setups includes any automated interpretation of the combined sensor state beyond simple fixed threshold checks, which are evaluated either right on the node or in a simple cloud script. None of them outputs a human-readable explanation of what a specific combination of readings means or what a user should do next. These systems are compared in Table 1.

Table 1. Comparison of related environmental monitoring systems.

Ref.	Controller	Sensor/ Parameters	Communication	User Interface	AI Intelligent Analysis	Major Limitation
[1]	Raspberry Pi + GrovePi+	DHT (T/RH), dust, MQ-5, MQ-7, MQ-135 (gases)	Wi-Fi HTTP → ThingSpeak	ThingSpeak web channel	Cloud MATLAB analytics → Twitter alert (rule-based)	No dedicated app; node count limited to 4
[2]	ESP32 / NodeMCU	DHT22 (temp., humidity)	Wi-Fi Blynk	Blynk mobile widget	None (qualitative label only)	Only 2 parameters; ~20% temp. error

Ref.	Controller	Sensor/ Parameters	Communication	User Interface	AI Intelligent Analysis	Major Limitation
[3]	ESP32	DHT11, MQ-7, MQ-135	Wi-Fi → Blynk ThingSpeak	OLED, LED, Blynk	None (LED threshold only)	No trend view; no cross-platform app
[4]	ESP32 WROOM DEVKIT	BME280, MQ-135, ML8511 (UV), rain sensor	Wi-Fi HTTP (Google Sheets, Blynk)	HTML web dashboard, Blynk	None; threshold-driven email (unvalidated)	Web-only UI; alerts not fully validated
Proposed	ESP32-WROOM-32	DHT22, MQ-series gas sensor, infrared optical dust sensor, NEO-6M GPS	Bluetooth Classic SPP (newline-delimited JSON)	Dart/Flutter cross-platform app	On-demand generative-AI analysis (Gemini) + on-device AQI category	—

These properties have been summarized in Table 1 along with the system proposed in this paper. This comparison clearly demonstrates the research gap that the current paper tries to bridge because the previously discussed ESP32- and Raspberry-Pi-powered air-quality monitors can only integrate, at best, multi-sensor capabilities, cloud and widget-based visualization, and alerts that are based on exceeding thresholds, but do not feature a dedicated cross-platform application with any form of context-aware, natural-language processing of the sensor status. In contrast, the system in question uses a direct connection between the node and the phone via Bluetooth that replaces remote access capabilities with the absence of Wi-Fi dependence and an additional layer of on-demand AI-based analysis.

3 Proposed methodology

The proposed system is organized as an end-to-end pipeline, where environmental sensors send their data into an ESP32 acquisition node; then the node computes a composite air quality reading right there on-device, and serializes it as JSON. The payload is then streamed over a direct Bluetooth Classic serial link, to a paired smartphone. A Flutter application then parses the stream into a live dashboard, and when the user requests it, the app submits the current reading set to a cloud hosted generative AI model to get back a more structured environmental analysis. Each stage is described in the following subsections.

3.1 Overall system architecture

The system is made up of four cooperating layers, as shown in Figure 1. The perception layer uses an ESP32-WROOM-32 development board [5] that is wired on a breadboard to the sensor suite discussed in Section 3.2. The acquisition layer, running on the ESP32, reads each sensor, computes a composite air-quality index and category on the device, and then serializes the outcome into JSON. The link layer then transports the payload over a Bluetooth Classic Serial Port Profile (SPP) connection, directly to a paired Android device, without relying on Wi-Fi or an intermediate cloud broker. The application layer is the Flutter dashboard. It receives the stream, parses the incoming messages, and renders the live sensor signals along with the GPS state, and, only when the user explicitly requests it, sends one HTTPS request to a cloud generative-AI endpoint. This call fetches the analysis described in Section 3.6.

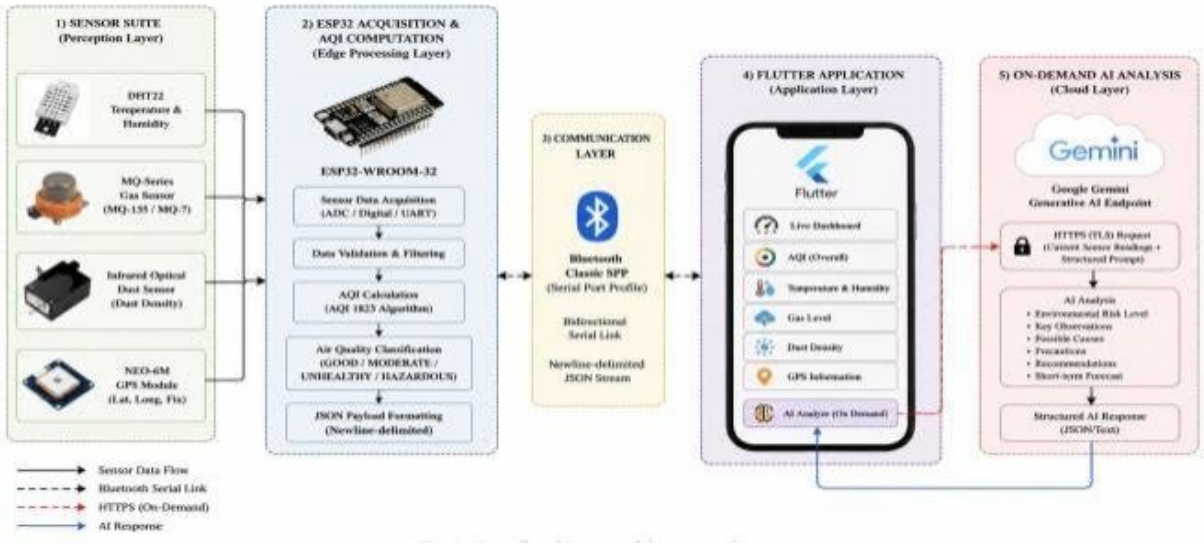


Figure 1. Overall system architecture.

3.2 Environmental sensing unit

The sensing unit combines four modules wired to the ESP32, as shown in Figure 2. A DHT22 digital sensor [6] provides temperature and relative humidity over a single-wire link. An analog MQ-series gas sensor module provides a raw ADC readout about the presence of usual combustible and pollutant gases; since it is taken directly as a 12-bit ADC count rather than a calibrated concentration, the "gas" reading is treated throughout the system as a relative indicator, not as a parts-per-million value. An infrared optical dust sensor outputs a raw ADC count plus a derived analog voltage and also a dust-density estimate in $\mu g/m^3$, following the raw/voltage/density signal chain described in published studies of low-cost optical particulate sensors. A NEO-6M GPS module [7] provides location, altitude, ground speed, and satellite count over UART, so when a satellite fix is around, a reading can be geotagged. Because these parts are low cost, the gas and dust channels do not claim laboratory-grade accuracy. The AQI value described in Section 3.3 is an on-device, sensor-derived score not a certified regulatory air quality index, and this distinction is deliberately maintained in both the application UI and the AI analysis prompt (Section 3.6); the value is not presented to the user as an official measurement.

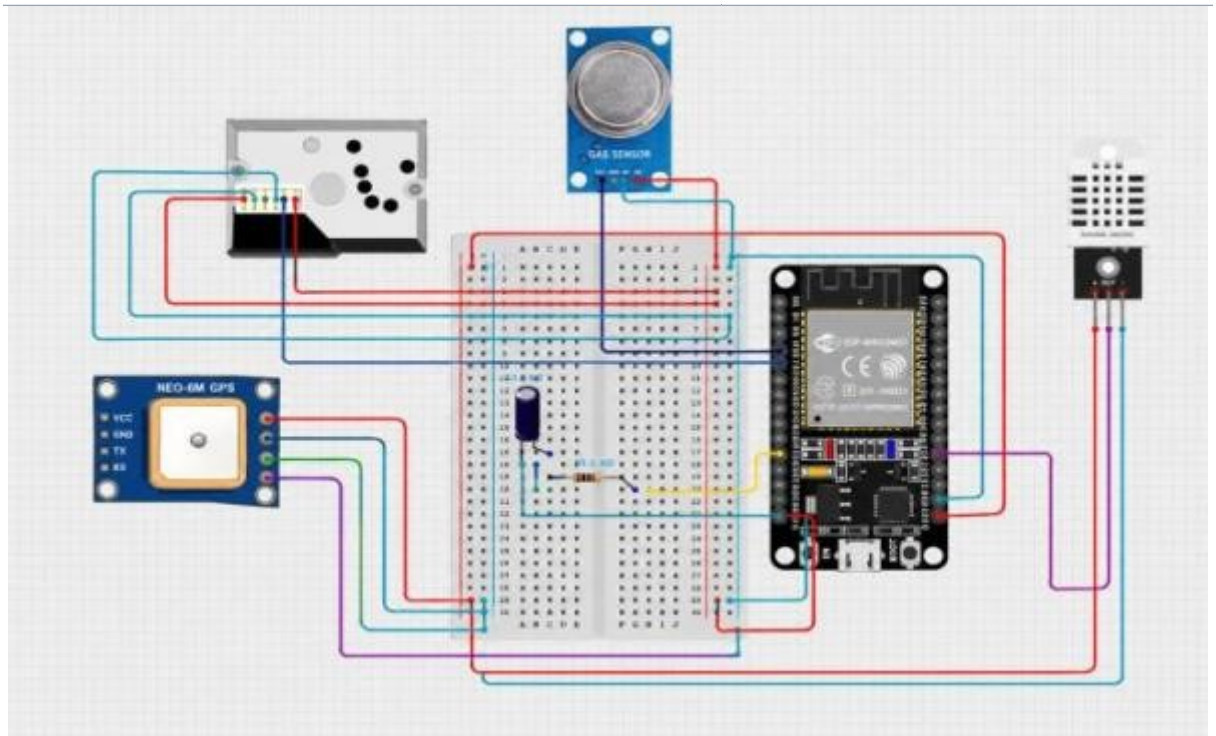


Figure 2. ESP32 connection diagram.

3.3 ESP32-based data acquisition

The ESP32 firmware samples the DHT22, the gas sensor's analog channel, and the dust sensor's analog channel, plus it reads the NEO-6M GPS module via UART. From these inputs the firmware computes a composite AQI integer and a matching category string (like "GOOD" during a low pollutant baseline test, as described in Section 4) using rule-based logic implemented on the node; the exact way the gas channel and dust channel are weighted inside that composite score is internal firmware logic and is not re-derived here. Every processed set of readings is packed into a single-line JSON object containing temperature, humidity, gas, dust, aqi, airQuality, latitude, longitude, altitude, speed, and satellites, which is written to both the USB serial port and the Bluetooth Classic output at 115200 baud; during testing the two channels were observed to carry the same payload, so the Bluetooth stream stays faithful to what is available for on-device debugging (Section 4). The GPS-related fields and the satellite count remain at zero until a fix is obtained, which did not happen in the indoor test set-up (Section 4).

3.4 Device-to-application communication (Bluetooth)

Instead of a Wi-Fi cloud uplink, the ESP32 presents itself as a Bluetooth Serial Port Profile device; it is advertised with the name "ESP32_AQI." On the Flutter side, via the flutter_bluetooth_serial_plus package [10], the app starts by making sure the phone's Bluetooth is turned on, searches the phone's list of bonded devices for one called "ESP32_AQI". It then opens a Bluetooth connection using the device's address. The user therefore needs to pair the module from the phone's Bluetooth settings first, otherwise the application can't connect. Once a connection is up, the app listens to the connection input byte stream, decodes each incoming chunk as UTF-8, and appends it to an internal buffer. Whenever the buffer contains a newline, it extracts the substring up to the newline, trims it, and runs it through a JSON decoder. The remainder is kept in the buffer for the next round. This buffering is required because Bluetooth SPP gives you an arbitrary byte stream, not neat message-framed

packets, so one JSON object can show up in multiple callbacks or be split across chunks. The only internet link is created by the phone application itself, and it happens on demand when the user requests the AI analysis (Section 3.6). The ESP32 node has no direct internet access, nor any cloud connectivity, in this current implementation.

3.5 Flutter application

The dashboard is implemented as a single-page Flutter application [9] rather than a native-per-platform client, so that the same codebase can, in principle, target Android, Windows and Linux from one source tree, although the results reported in Section 4 were obtained on Android. The app uses a deliberately lightweight architecture built around a single stateful widget: when Bluetooth data comes in, it is parsed straight into typed fields (temperature, humidity, gas, dust, aqi, airQuality, latitude, longitude, altitude, speed, satellites) that live inside the widget state, and every successful parse causes a UI refresh via Flutter's usual setState mechanism. The interface comprises a Bluetooth connection card that shows the device name plus the current live connection status, a large AQI card in which the number and the category text are color-coded across six bands (green, orange, deep orange, red, purple, and brown) with increasing severity. separate sensor cards for temperature, humidity, gas, and dust density, a GPS information card, and an AI-analysis button with a result panel. In this implementation, the app does not save past readings across sessions, and it does not draw time-series charts, so each screen mirrors only the most recently parsed reading. Trend visualization and offline caching are treated as future work (Section 5) rather than as implemented features.

Finally, Table 2 lists the two Dart packages used for Bluetooth communication and the AI HTTP request.

Table 2. Dart packages used for Bluetooth communication and AI requests.

Component	Package	Purpose
Bluetooth link	flutter_bluetooth_serial_plus 0.5.x	Discover the paired ESP32_AQI device, open a Bluetooth Classic SPP connection and stream JSON data.
AI HTTP client	http 1.6.x	POST the constrained prompt to the Gemini generateContent REST endpoint and parse the response.

3.6 Environmental analysis using AI (Gemini)

The application makes a distinction between two ways of providing information to the user through either deterministic or generative methods. The former is the calculation of the on-device AQI category by the ESP32 firmware itself (Section 3.3) that the application merely presents in a colored format; this piece of information is readily available every time the Bluetooth connection is established and does not require any AI service or internet connection. The latter is the on-demand environmental analysis done using the cloud: clicking the “AI Analyze” button, available only when the Bluetooth connection is established, builds a text prompt with the current temperature, humidity, gas ADC value, dust density, AQI value and category, and GPS status and sends it via an HTTPS request to Google's Gemini generative-AI API [8] (generateContent endpoint). The request clearly tells the model to write its response using specific headings (Overall Assessment; Risk Level, which can be either LOW, MODERATE, HIGH, or CRITICAL; Key Observations; Possible Causes; Precautions; Health Considerations; Environmental Recommendations; Short-Term Prediction; and Important Warning) and imposes certain restrictions on the response (the model has to rely only on the sensor values provided, the AQI has to be considered a custom score calculated from the sensor data and not an

actual regulatory index, the gas value should be understood as an ADC reading, and not as a concentration measurement, the model cannot provide a medical diagnosis, and the response has to be practical and non-repetitive). The resulting text is shown without changes in a special result card. In this way, the interpretation stage is intentionally kept outside of both the firmware and the locally trained model, sacrificing the speed of the classification done on-device for the more context-dependent reasoning done by the general-purpose language model on demand.

3.7 Data processing and decision flow

The end-to-end decision flow consists of the following steps: (1) ESP32 gets readings from the DHT22, gas, dust, and GPS sensors; (2) firmware calculates the combined AQI value and category string; (3) readings list is converted to the single-line JSON object; (4) this object is sent through the USB serial interface and Bluetooth Classic SPP in parallel; (5) the listener of the input stream in the Flutter application collects incoming bytes until the newline is detected; (6) message is decoded in JSON and all the sensor, AQI, and GPS state fields are populated in the application; (7) the UI is rebuilt, and AQI, sensor, and GPS cards are repainted; (8) when the user clicks on the "AI Analyze" button while connected to the device, the current state is transformed to the constrained prompt defined in Section 3.6; (9) prompt is passed to the Gemini API using HTTPS, and the text answer is extracted from the response; and (10) generated analysis is shown in the result card. Steps 1–7 are performed repeatedly when new Bluetooth messages appear, whereas steps 8–10 happen only on demand.

4 Experimental results and discussion

The developed system was assessed on a functional level, as there was no available comparison tool with certification, which could be used for statistical calibration; therefore, the results presented below are the observations that were made directly by analyzing the output from ESP32, the Flutter application through Bluetooth connection, and Gemini analysis tool.

Table 3 gives the sensor measurements collected in two different test environments with the same firmware version. The first set is a low pollutant indoor baseline, collected from the Arduino IDE serial monitor and compared with the Bluetooth JSON payload; the gas ADC value is close to zero, the dust sensor gives a density close to zero, and the firmware computes the AQI as 5 with the category "GOOD." The second set is another test environment recorded directly from the Flutter application after exposure to an environment with higher particulates; the gas ADC value is 186, the dust sensor measures a density higher than 2900 $\mu g/m^3$, and the firmware computes the AQI as 1823 with the category "HAZARDOUS." While both tests have been conducted without a controlled environment and cannot be used for calibration purposes, the switch shows that the AQI category computation and assignment are as expected in response to changes in the environment and are accurately reflected in the Bluetooth JSON payload to the color-coded AQI card in the application (Figure 3).

Table 3. Representative sensor readings under two test conditions.

Parameter	Baseline (indoor, serial log)	Elevated condition (app test)
Temperature (°C)	28.90	29.3
Humidity (%)	70.00	70.0
Gas (raw ADC)	0	186
Dust density ($\mu g/m^3$)	0.00	≈2947–2967

Parameter	Baseline (indoor, serial log)	Elevated condition (app test)
AQI (on-device score)	5	1823
Air-quality category	GOOD	HAZARDOUS
GPS satellites / fix	0 / no fix	0 / no fix



Figure 3. Flutter application dashboard under the elevated-particulate test condition.

The Bluetooth communication connection itself proved to be stable at the message level – the newline separated JSON formatting used in the application's stream buffer successfully recovered full objects for the sensors during the entire test session, and the payload that was read via Bluetooth was identical to the payload displayed on the USB serial monitor at the same time, proving that the JSON formatting process works consistently on both output streams. The GPS module was unable to obtain a satellite fix in both test sessions, indicating that no satellites have been found and staying in the "Waiting for satellites…" state all the time; however, this is consistent with the characteristics of the NEO-6M module and is considered to be an environmental issue rather than a flaw in the satellite acquisition process, since the latitude, longitude, altitude, and speed fields were correctly zeroed out and parsed by the application.

The "on-demand generative-AI analysis" feature was tested using the Gemini API as the application was being used to connect to the sensor node. As opposed to a classifier, the endpoint here is a generic generative AI model and hence does not return a numeric score but rather free text. As such, there are no accuracy percentages for the results obtained, although what should be evaluated is whether the AI-generated text meets the required constraints in structure and behavior. Throughout

all the test sessions that were done, the analysis that was received was able to follow the required structure in the prompt (Overall Assessment, Risk Level, Key Observations, Possible Causes, Precautions, Health Considerations, Environmental Recommendations, Short-Term Prediction, and Important Warning), use only one of the four risk-level categories and avoid using the AQI as a certified regulatory measure or the gas reading as a calibrated concentration as stated in the prompt's constraints.

5 Conclusion

In this paper, a Bluetooth-based ESP32 air quality monitor that combines a DHT22, an analog MQ-series gas sensor, an infrared optical dust sensor, a NEO-6M GPS module, and a lightweight cross-platform Flutter application was proposed, with the replacement of a local recommendation model by a demand-driven, cloud-based analysis based on generative-AI. Functional testing has proven that the firmware JSON payload is transferred successfully through USB serial and Bluetooth Classic SPP, the stream buffering mechanism implemented within the application is able to parse messages using newline delimiter, and the calculated value of AQI and its category change in the predicted direction from the low-pollutant case (AQI 5/GOOD) to the high particulate concentration condition (AQI 1823/HAZARDOUS). The analysis feature using the Gemini model has proven reliable in generating structured environmental reports that comply with the imposed constraints without making any diagnosis claim or presenting the sensor AQI value as a certified measurement. These findings suggest the feasibility of the direct device-to-phone Bluetooth-based architecture combined with a demand-driven large-language model interpretation layer as an affordable solution to the problem of IoT cloud pipeline. In the current iteration, it is constrained by the fact that the range is dependent on Bluetooth connectivity and phone pairing rather than remote connectivity, that the system is not calibrated using a lab instrument for gas and dust sensors, that there is no recording of history or trend graph, and finally that the NEO-6M module does not acquire satellites indoors; some of these constraints have been highlighted as areas of improvement for future iterations.

Acknowledgment

The authors thank the Department of Computer Science and Engineering, Rajiv Gandhi Institute of Technology, Bengaluru, for the facilities and support provided for this work. They are also grateful to the faculty for their guidance, and to their families and friends for encouragement.

References

[1] Balasubramaniyan C, Manivannan D. IoT enabled air quality monitoring system (AQMS) using Raspberry Pi. *Indian J Sci Technol*. 2016;9(39). https://doi.org/10.17485/ijst/2016/v9i39/90414

[2] Adeagbo AA. *IoT based environment monitoring system using ESP32* [technical report]. Edwardsville (IL): Southern Illinois University Edwardsville; 2022.

[3] Omkar N, Rahul S, Vijaykrishna R, Swapna N, Sura MR. Smart environmental monitoring using ESP32 microcontroller. *IOSR J Electron Commun Eng*. 2024;19(3):7–12. https://doi.org/10.9790/2834-1903010712

[4] Uddin MS, Praharaj MK. IoT based weather monitoring system using ESP32 microcontroller. *Int J Adv Res Sci Commun Technol*. 2025;5(2):693–701. https://doi.org/10.48175/IJARSCT-29283

[5] Espressif Systems. *ESP32 series datasheet, version 3.8*. Shanghai: Espressif Systems; 2021.

[6] Aosong Electronics. *Digital-output relative humidity & temperature sensor/module DHT22 (AM2302)* [datasheet]. Guangzhou: Aosong Electronics.

[7] u-blox AG. *NEO-6 u-blox 6 GPS modules data sheet (GPS.G6-HW-09005)*. Thalwil: u-blox AG; 2011.

[8] Google LLC. *Gemini API reference: generating content* [Internet]. Mountain View (CA): Google LLC. Available from: https://ai.google.dev/api/generate-content

[9] Google LLC. *Dart programming language documentation* [Internet]. Mountain View (CA): Google LLC. Available from: https://dart.dev/

[10] flutter_bluetooth_serial_plus [software package]. pub.dev. Available from: https://pub.dev/packages/flutter_bluetooth_serial_plus

[11] World Health Organization. *WHO global air quality guidelines: particulate matter (PM2.5 and PM10), ozone, nitrogen dioxide, sulfur dioxide and carbon monoxide*. Geneva: World Health Organization; 2021.

[12] ASHRAE. *Thermal environmental conditions for human occupancy*. ANSI/ASHRAE Standard 55-2017. Atlanta (GA): ASHRAE; 2017.

[13] Wakde B, Patil AC, Nithya Shree B, Nihar TN, Varun PS. *AI-integrated smart environment monitoring system with real-time Dart dashboard: A unified IoT framework using ESP32, multisensory array, and intelligent alert mechanism*. Int J Sci Res Eng Manag. 2026;10(4).

[14] Bakare MS, Abubaker K. IoT-based indoor environmental monitoring system using multi-parameter sensing and ESP32-WROOM integration. *Discov Electron*. 2026;3. https://doi.org/10.1007/s44291-026-00157-3

[15] Google LLC. *TensorFlow Lite guide* [Internet]. 2024. Available from: https://www.tensorflow.org/lite/

[16] Jo JH, Jo B, Kim JH, Kim S, Han W. Development of an IoT-based indoor air quality monitoring platform. *J Sens*. 2020;2020:8749764. https://doi.org/10.1155/2020/8749764

[17] de Medeiros HPL, Girão G. An IoT-based air quality monitoring platform. In: *2020 IEEE International Smart Cities Conference (ISC2)*; 2020. https://doi.org/10.1109/ISC251055.2020.9239070

[18] Jo JH, Jo B, Kim JH, Choi I. Implementation of IoT-based air quality monitoring system for investigating particulate matter (PM10) in subway tunnels. *Int J Environ Res Public Health*. 2020;17(15):5429. https://doi.org/10.3390/ijerph17155429

[19] Muhammad K, Ramadhan Putra RR, Purnomo E. Design and implementation of microcontroller-based air filter system with Android application interface. *Emitor J Tek Elektro*. 2026;26(1):71–81. https://doi.org/10.23917/emitor.v26i1.16143