

A Dual-Microcontroller IoT Air Quality Monitoring System with Real-Time Local Alerting

Kedar Gurav¹

¹Department of Engineering
Science and Humanities,
Vishwakarma Institute of
Technology, Pune, India.

Harsh Gupta²

²Department of Engineering
Science and Humanities,
Vishwakarma Institute of
Technology, Pune, India

Abhimanyu Gitte³

³Department of Engineering
Science and Humanities,
Vishwakarma Institute of
Technology, Pune, India

Abstract – Air pollution is a prolonged environmental and public-health concern that requires affordable monitoring tools. This work presents a dual-microcontroller Internet-of-Things (IoT) air-quality monitoring system that separates time-critical alert logic from network communication. An Arduino Uno performs continuous data acquisition and threshold evaluation of an MQ135 gas sensor, while a NodeMCU ESP8266 handles Wi-Fi connectivity and cloud transmission to the Blynk platform. Local LED and buzzer alerts are controlled through a non-blocking 200 ms sampling loop, achieving sub-millisecond computational reaction after threshold breach. Experimental tests with incense smoke diffusion showed that the sensor crossed the 70 ADC alert threshold after ≈ 80 s and reached a peak of 145 ADC after ≈ 130 s. The alert deactivated after ≈ 170 s and the sensor returned to baseline in ≈ 14.5 min. Local alert functions remained operational during intentional Wi-Fi interruptions, demonstrating architecture-level stability. Although the uncalibrated MQ135 limits measurement fidelity, the system provides an economically viable and reliable framework for educational and awareness-based air-quality monitoring applications.

Key Words: Air Quality Monitoring, Internet of Things, MQ135 Sensor, Arduino, NodeMCU, Real-Time Alert System, Environmental Sensing, IoT Architecture.

1. INTRODUCTION

Air pollution is one of the biggest environmental threats to human health. The World Health Organization (WHO) states that it causes about 7 million premature deaths every year. We see this pollution in our cities from cars, factories, and burning fossil fuels. These activities have raised the levels of harmful pollutants like PM_{2.5}, NO₂, SO₂, and others [1].

1.1 Problem Statement

Although many low-cost prototypes exist, they still have several key problems. Many existing systems use a single microcontroller to both acquire sensor data and communicate with the network, leading to latency and

potential failure points in critical alert functions. Systems depending solely on cloud-based alerts stop working during network outages, which creates a safety vulnerability. [2].

1.2 Research Objectives

This project is designed to address these gaps with the following objectives: (1) Design and implement a real-time air pollution monitoring system using readily available, low-cost components within an IoT framework; (2) Develop a dual-microcontroller architecture that decouples time-critical sensor acquisition and alerting from network communication tasks; (3) Integrate an MQ135 gas sensor for detection of common airborne pollutants; (4) Implement immediate local feedback mechanisms through an I²C LCD display and audiovisual alerts; (5) Enable wireless data transmission to the Blynk IoT platform for remote monitoring; (6) Validate system functionality through controlled experimental testing.

1.3 Research Contributions

The main contributions are: (1) A dual-microcontroller architecture, which was validated experimentally to enhance system robustness by isolating safety-critical functions from network dependencies; (2) End-to-end system integration from sensor hardware to cloud visualization; (3) A performance validation that shows system responsiveness and operational stability.

2. LITERATURE REVIEW

2.1 Evolution of Air Quality Monitoring

Traditional air monitoring networks use expensive, high-tech tools like chemiluminescence analyzers for nitrogen oxides and ultraviolet fluorescence for sulfur dioxide. While they provide highly accurate measurements, these tools cost \$30,000 or more, cities can't install many of them [1]. This creates large gaps in data. Clements et al. [2] studied the switch to cheaper, "low-cost" monitors. They pointed out the

main challenges: we need to calibrate these sensors, make sure the data is good, and test how long they last.

2.2 Low-Cost Sensor Technologies

Metal Oxide Semiconductor (MOS) sensors, like the MQ series, are seen in many projects. This is because they are very cheap and respond fast. But, they have big problems. They often react to gases they aren't supposed to (cross-sensitivity), and their readings change with temperature and drift over time. This limits their accuracy, so we can't get exact numbers from them without good calibration [3]. Recent studies have explored calibration approaches, including co-location with reference instruments and machine learning algorithms [4].

2.3 IoT-Based Monitoring Systems

By integrating the low-cost sensors with IoT platforms, we can now build huge monitoring networks. Some researchers focus on making the data secure [5], while others have used NodeMCU and Blynk, like we did [6]. But, we found that many of these projects use only one microcontroller. This single chip has to do both sensing and communication, which can make the alerts fail if the network drops [6].

2.4 Research Gap

In our literature review we found there is a limited focus on system-level robustness, specifically in ensuring local alert functionality autonomous of cloud connectivity. There is also limited documentation of methodologies for setting meaningful alert thresholds on uncalibrated sensors. Our project addresses these gaps through a dual-microcontroller architecture with thorough performance validation.

3. SYSTEM ARCHITECTURE AND HARDWARE DESIGN

3.1 Overall System Architecture

The system uses a multi-level architecture: a sensing and alerting layer, a communication gateway layer, and a cloud visualization layer. Our main principle was "functional isolation", where time-critical operations including sensor obtainment, threshold evaluation, and alerting are decoupled from non-deterministic network condition operations such as Wi-Fi management and cloud communication. This makes sure the local alert works consistently whether there is network availability or not. Such a layered isolation strategy aligns with recent resilience approaches in edge computing, which emphasize architectural fault containment and predictable local execution under unreliable network conditions [17].

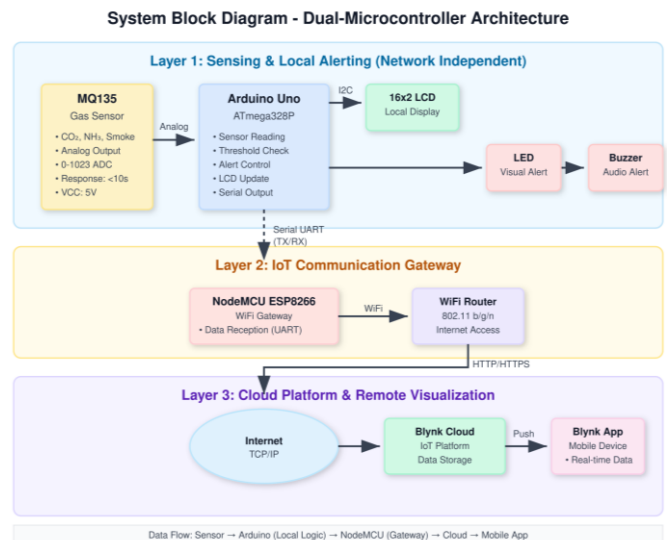


Fig -1: The system's 3-layer block diagram, showing how we decoupled the sensor (Layer 1) from the IoT gateway (Layer 2).

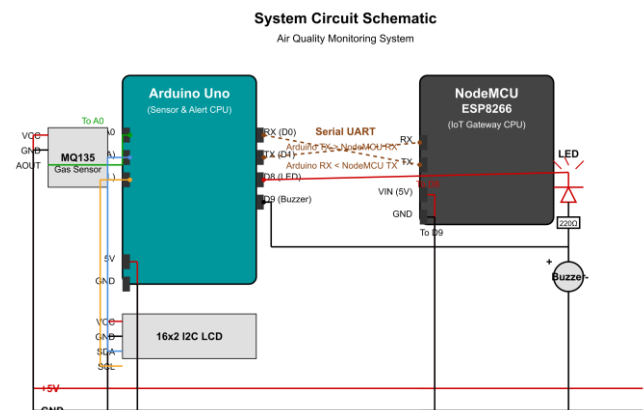


Fig -2: The complete circuit schematic designed, showing the UART connection between the Arduino and NodeMCU.

Our system's workflow is a 6-step process (see Fig -1): (1) First, the MQ135 sensor continuously samples the air. (2) The Arduino Uno reads this as an analog voltage. (3) The Arduino checks the value, updates the LCD, and controls the local alerts. (4) It then sends the final value via serial UART to NodeMCU; (5) The NodeMCU gets this data, connects to WiFi, and transmits data to the Blynk cloud; (6) Finally, we (the user) can see the data via the Blynk mobile application.

3.2 Hardware Components

We chose our components based on cost, availability, and function. The complete system specifications are presented in Table 1.

Table -1: The list of components used, their main function, and their cost.

Component	Model/Type	Primary Function	Key Specifications	Cost (USD)
Gas Sensor	MQ135	Multi-pollutant detection	MOS type, 10-200ppm CO ₂ , Analog output, Response: <10s	\$4.50
Primary Microcontroller	Arduino Uno R3	Sensor acquisition & local alerting	ATmega328P, 16MHz, 6 analog inputs, 14 digital I/O	\$12.00
IoT Gateway	NodeMCU ESP8266	WiFi connectivity & cloud communication	ESP-12E, 802.11 b/g/n, 80MHz, Integrated TCP/IP	\$6.50
Display	16x2 LCD with I2C	Local data visualization	Alphanumeric, I2C interface (SDA/SCL), Backlit	\$3.50
Visual Alert	5mm Red LED	Threshold breach indicator	Forward voltage: 2.0V, Current: 20mA	\$0.10
Audio Alert	Active Buzzer	Audible alarm	5V operation, 85dB @ 10cm, Pulsating pattern	\$0.80
Current Limiting	220Ω Resistor	LED protection	1/4W, Carbon film	\$0.05
Prototyping	Breadboard & Wires	Component interconnection	830-point breadboard, 30 jumper wires	\$9.25
Total System Cost				\$36.70

3.3 MQ135 Gas Sensor

The MQ135 sensor uses a tin dioxide (SnO₂) semiconducting ceramic. It works based on chemoresistive behavior, where the sensor's resistance changes in the presence of various gases. The sensor exhibits non-linearity, cross-sensitivity, and dependence on temperature and humidity [7]. It provides an analog voltage output ranging from 0 to 5 volts corresponding to gas concentration, with a fast response time. During testing with incense smoke, the MQ135 exhibited an average response time of ≈ 80 s, consistent with reported behavior for slow-diffusing smoke environments.

3.4 Dual-Microcontroller Architecture

The Arduino Uno is used as the dedicated processor for all time-critical jobs: continuous analog-to-digital conversion, real-time threshold evaluation, LCD management via I²C protocol, and digital control of the LED and buzzer. Its timed execution guarantees uniform sampling rates and alert latency.

The NodeMCU ESP8266 [4] handles internet connectivity, managing serial communication with the Arduino, WiFi authentication, and periodic data transmission to the Blynk server. By isolating network operations, the NodeMCU's processing load does not impact the Arduino's sampling and alert functionality, ensuring local alerts function independently of network status [3].

4. SOFTWARE IMPLEMENTATION

4.1 Arduino Firmware

The Arduino code runs a continuous monitoring loop. In the setup() function, we configure serial communication at 9600 baud, initialize the I²C LCD, and configure digital output pins for the LED at D8 and buzzer at D9. The main loop() function does the following: (1) Read the analog value from the MQ135 connected to pin A0; (2) Transmit the reading to the NodeMCU via Serial.println(); (3) Update the LCD display; (4) Evaluate the threshold condition where gasLevel exceeds 70; (5) If exceeded, activate the LED to HIGH state and buzzer with pulsating pattern; (6) If below threshold, deactivate all alerts to LOW state; (7) We control the loop using a non-blocking timer by polling the millis() function. This ensures the main loop never stops.

Arduino Firmware Logic (Non-Blocking)

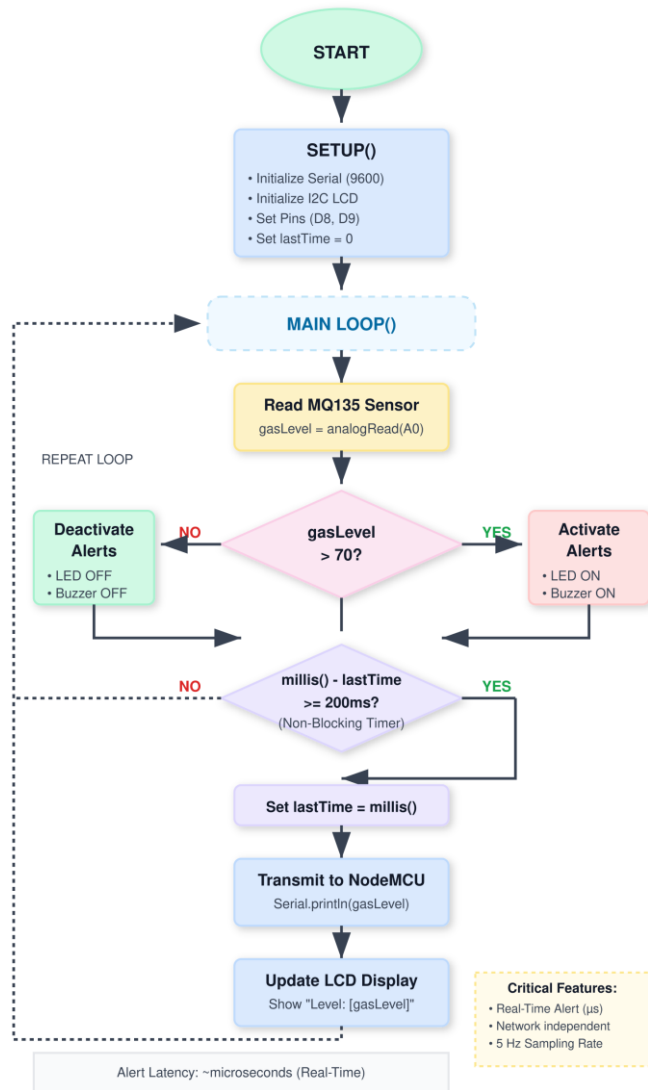


Fig -3: The logic flowchart for Arduino firmware.

4.2 Sensor Output Interpretation and Air Quality Assessment

The MQ135 gives an analog voltage, and the Arduino's ADC turns this into a number from 0 to 1023. This is just a raw ADC number; it is not a standard air quality metric. We made a key decision here: we did not try to convert this raw value to PPM or an official Air Quality Index (AQI) [13]. To do that, we would need to calibrate our sensor against lab-grade equipment. Without that calibration, trying to guess the PPM is not scientifically valid. Because of this, our system works as a "relative" air quality indicator. Based on our own tests, we created a simple scale relative air quality indicator. Based on empirical testing, ADC readings were qualitatively interpreted: 0 to 30 indicates clean air conditions, 30 to 50 indicates normal indoor air quality, 50 to 70 indicates elevated levels, and values exceeding 70 indicate poor air

quality requiring attention. This scale gives people a useful warning, but we do not claim it has quantitative precision.

4.3 Threshold Selection

The alert threshold of 70 ADC units is set based on our own tests. The system's baseline in a clean outdoor environment averaged 18 ADC units. Typical indoor environments averaged 35 ADC units. We chose the 70 ADC threshold because it provides a substantial margin above normal conditions, approximately twice the indoor average, and reliably detects moderate pollution events without excessive false alarms. While this may correspond to manufacturer estimates for moderate carbon dioxide or smoke levels [8], we must emphasize that this is an empirical, not a quantitative, threshold.

4.4 NodeMCU Firmware and Cloud Integration

The NodeMCU's code is simpler because it only focuses on network tasks. We wrote it to connect to our local WiFi and make a secure connection to the Blynk server. It spends all its time listening for data from the Arduino. When a new reading comes in, that data is pushed to two virtual pins on our Blynk dashboard: V0 for the number, and V1 for the alert light, to mirror the local alert status on the mobile interface.

5. EXPERIMENTAL METHODOLOGY AND RESULTS

5.1 Experimental Setup

To validate performance, we ran a structured test. The test was set up in a controlled indoor room. Our protocol had 5 steps: (1) A 15-minute baseline recording phase; (2) A pollution introduction phase at time equals 15 minutes using an incense stick; (3) A response phase to observe detection and alert activation; (4) A recovery phase at time equals 20 minutes after removing the source; (5) Repeatability assessment across three trials.

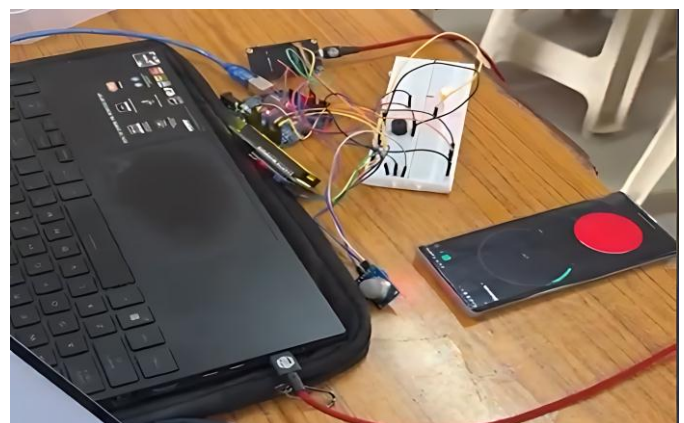


Fig -4: The assembled hardware prototype during the incense stick test, showing the alert on the Blynk app.

5.2 Network-Outage Simulation Protocol

While the system was in an active alert state ($ADC > 70$), the Wi-Fi access point connected to the NodeMCU module was intentionally powered down to simulate a network outage. The test was conducted 4 times, each lasting five minutes. During this period, the Arduino Uno continued its sensing and alert-control loop, while the NodeMCU experienced loss of connection to the Blynk cloud server. System behavior, including LED and buzzer operation, serial-monitor readings, and post-reconnection status, was recorded to verify that local alert functionality remained unaffected by the loss of network connectivity.

5.3 Experimental Results and Performance Metrics

Our system showed consistent and reliable performance. The averaged quantitative results are presented in Table 2.

Table -1: Measured performance metrics of the prototype (mean of $n = 3$ trials). Computational latency refers to local code execution time after threshold detection, measured in milliseconds, while end-to-end latency refers to the physical time from pollutant introduction to alert activation.

Performance Metric	Average Value	Standard Deviation	Range
Clean Air Baseline	18 ADC units	± 2	16-20
Threshold Breach Time	80 s	± 18 s	65-95 s
Alert activation (computational)	<1 ms (code / digitalWrite() after sample)	n/a (negligible)	not a sensor metric
Sampling Interval (loop)	200 ms	—	—
End-to-End System Alert Latency	≈ 80 s	± 18 s	65 - 95 s
Time to Peak from Ignition	130 s	± 15 s	110-140 s
Alert Deactivation Time	170 s	± 22 s	150-195 s
Full Recovery Time	870 s (14.5 min)	± 50 s	840-930 s
False Alarm Rate	No false alarms	—	—

	observed (n = 3)		
Data Transmission Success	100%	—	All packets
Cloud Display Latency	1.2 s	± 0.3 s	0.9-1.5 s

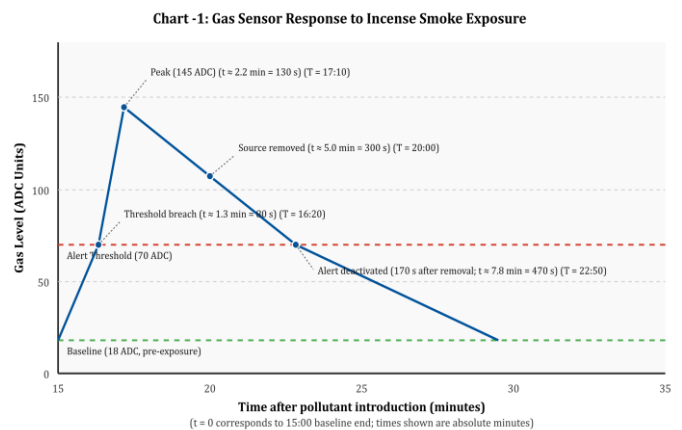


Chart -1: Time shown relative to the start of pollutant introduction ($t = 0$ at 15:00). Values in chart are absolute minutes; key event times are also given in seconds in annotations. Baseline = 18 ADC, Threshold = 70 ADC, Peak = 145 ADC, and Recovery ≈ 870 s (14.5 min).

The prototype maintained a clean-air baseline of about 18 ADC units with a small deviation of ± 2 units. After incense smoke exposure, the sensor output crossed the alert threshold at ≈ 80 seconds (± 18 s), peaked near 145 ADC at ≈ 130 seconds, and returned to baseline about 870 seconds (14.5 min) after the source was removed. Alert deactivation occurred roughly 170 seconds (± 22 s) after source removal, consistent with the expected diffusion-recovery profile of the MQ135 sensor.

The microcontroller reacted almost instantly once the threshold was reached, requiring less than 1 ms for code execution. With a sampling loop of 200 ms, the end-to-end alert latency effectively matched the sensor’s physical response time. Across three diffusion trials, no false alarms were observed, all Wi-Fi packets were delivered successfully, and the cloud dashboard updated within about 1.2 ± 0.3 seconds of local alert activation.

Overall, the data confirm that system delay is dominated by the sensor’s adsorption-desorption dynamics rather than processing time. The close agreement between repeated measurements and the plotted curve in Chart 1 indicates consistent time behavior and stable operation of both sensing and communication layers under the tested

conditions. The MQ135 output was observed immediately after power-up without long-term stabilization; hence, the timing data reflect relative signal behavior during diffusion tests, not calibrated gas response constants.

5.4 Results: Network-Outage Test

When the Wi-Fi access point was switched off during an active alert, the NodeMCU immediately lost connection to the Blynk server and paused data transmission. However, the Arduino-controlled LED and buzzer remained active for the entire five-minute disconnection interval, following the same on/off pattern determined by the local threshold logic. The ADC readings continued to update on the serial monitor without interruption, confirming that sensing and alert control were fully autonomous of the network link. Upon restoring Wi-Fi, the NodeMCU automatically re-established connection and resumed data uploads without requiring a system reset. These results verify that the alert subsystem operates independently of cloud connectivity and that network interruptions do not affect local response timing or reliability.

6. DISCUSSION AND COMPARATIVE ANALYSIS

6.1 System Efficacy and Architectural Validation

Results confirm the prototype achieves all research objectives, showing a functional end-to-end data pipeline. The main strength of our design is its validated architectural design. The decision to implement a decoupled, dual-microcontroller architecture was experimentally evaluated; results show the local alert logic executes with negligible computational latency (microseconds–milliseconds), while the end-to-end detection time is dominated by sensor response, far exceeding the 200ms latency of a simpler delay()-based system. This addresses a critical safety concern prevalent in single-microcontroller implementations, where network operations can compromise alert integrity [9].

6.2 Comparative Analysis with Existing Systems

A comparative summary with related projects is presented in Table 3. Where the cited works do not report numeric sensor threshold times, the table notes ‘Not reported’; for professional systems using laser PM sensors we cite the sensor datasheet for typical response times.

The dual-microcontroller design provides clear advantages over single-microcontroller implementations [6], primarily ensuring dependable local alert operation independent of network status [3]. This functional isolation also ensures consistent sampling intervals and provides fault containment.

Table -2: Our comparison of this system with other projects.

Feature	Proposed System	Muhd Zain [6]	Bobulski [5]
Architecture	Dual MCU (Arduino + ESP8266)	Single MCU (ESP8266)	Arduino Mega + Raspberry Pi 4
Primary Sensors	MQ135 (MOS Multi-gas)	MQ135 (MOS Multi-gas)	Calibrated PM, CO ₂ , VOC
Local Display	16x2 LCD I ² C	None	7-inch Touchscreen
Alert Independence	Yes, local alerts operate network-free	No, cloud-dependent	Yes
Data Representation	Raw ADC 0-1023	PPM basic conversion	Calibrated µg/m ³
System Cost	\$37	\$28	\$180
Measured Response	~ 80 s (mean of 3 incense trials)	Not reported (prototype uses NodeMCU + Blynk; latency not quantified)	Not reported (uses PM sensor and secure data transmission; no numeric threshold time given)
Target Application	Education, Citizen Science	Home monitoring	Professional Research

Our system is about 80% cheaper than professional-grade systems [5]; however, the end-to-end detection time is dominated by sensor physics (~ 80 s in our incense diffusion trials), whereas professional systems use calibrated laser PM sensors whose datasheets indicate response times on the order of seconds (e.g., Plantower PMS series, single/instant update ~ 1 s; total response < 10 s). The comparison in Table

3 therefore reports the sensor threshold-breach response for each system or notes 'Not reported' where the original paper gives no numeric value.

6.3 System Limitations and Critical Assessment

The main limitation of the prototype is the uncalibrated MQ135 sensor: the device should be considered a relative indicator. Section 4.2 discusses cross-sensitivity and environmental drift impacting absolute accuracy, and its ADC readings are subject to cross-sensitivity from various gases, as well as environmental drift from temperature and humidity [7]. Inter-unit variability and sensor aging also mean the 70 ADC threshold is empirical and would require re-characterization for each device or long-term deployment. Therefore, our system is best used for awareness and alert applications, not for scientific or medical-grade quantitative measurement, which remains a key area for future work through co-location calibration [13].

The MQ135 sensor provides qualitative readings that indicate relative gas presence but cannot yield standardized concentrations without calibration using reference instruments. The 70 ADC alert threshold in this work is empirical and used only to evaluate system response. Accordingly, the experiments validate system architecture, timing, and alert reliability rather than absolute pollutant accuracy.

The MQ135 sensor was operated without an extended preheating or calibration phase; therefore, the absolute readings and timing curves may include initial drift effects. These data are interpreted qualitatively to assess system responsiveness rather than to characterize sensor kinetics.

6.4 Practical Applications and Validated Use Cases

Even with these limitations, our system is still very useful. The tests confirm it is a good fit for places where cost matters more than perfect accuracy. For example, it is great for education and STEM projects, showing real-time environmental feedback and illustrating IoT design principles through hands-on prototyping. The system can be deployed in household environments to detect ventilation deficiencies and cooking-related pollutant buildup. Community groups could use our design to deploy multiple low-cost nodes for comparative neighborhood mapping, identifying pollution hotspots through relative differences rather than absolute measurements. Small workplace environments including workshops, storage areas, and parking garages can use the system as preliminary screening, triggering ventilation when significant deviations from baseline occur.

7. FUTURE WORK

The most important next step is to calibrate the sensor. We would like to do this by co-locating it with certified regulatory monitors, this would let us the use of machine learning models to convert raw data into calibrated measurements [4]. We could also add more sensors (like a PM2.5 sensor) [12] and migrating intelligence to the edge using TinyML frameworks for on-device calibration [13].

8. CONCLUSIONS

A low-cost dual-microcontroller IoT prototype was developed and tested; the functional isolation of sensing and networking preserves local alerts during network interruptions. The prototype demonstrates architecture-level robustness for awareness applications, though absolute pollutant quantification requires calibration. The Arduino-based controller demonstrated real-time reaction (< 1 ms) once the sensor threshold was sampled, while overall alert latency was dominated by the MQ135 sensor's ≈ 80 s response to diffusive incense exposure. Peak readings of 145 ADC and a 14.5-minute recovery were consistent across three trials. The system showed no false alarms and maintained function during Wi-Fi disconnections. Although not calibrated for absolute pollutant concentrations, the prototype validates the benefit of functional isolation between sensing and network tasks and serves as an accessible educational and citizen-science tool for local air-quality awareness. Future work will focus on sensor calibration and integration of multi-gas modules to improve quantitative fidelity.

The presented system demonstrates the feasibility of a dual-microcontroller IoT architecture that maintains alert autonomy under network outages. Although the MQ135 sensor limits the precision of measured values, the design successfully validates the real-time responsiveness, reliability, and cost-effectiveness of the proposed approach for educational and awareness-oriented air-quality monitoring.

ACKNOWLEDGEMENT

We acknowledge Vishwakarma Institute of Technology, Pune, for providing laboratory facilities and technical resources. Our special gratitude to Prof. Sayali Shinde, Dept. of Computer Science and Engineering (Data Science), VIT Pune and Prof. Pradnya Kolekar, VIT Pune for valuable guidance and continuous support throughout this project.

REFERENCES

- [1] A. L. Clements, W. G. Griswold, A. Rs, J. Johnson, M. M. Herting, J. Thorson, A. Collider-Oxandale, and M. Hannigan, "Low-cost air quality monitoring tools: From

- research to practice (A workshop summary)," *Sensors*, vol. 17, no. 12, p. 2685, 2017.
- [2] L. Morawska, P. K. Thai, X. Liu, A. Asumadu-Sakyi, G. Ayoko, A. Bartonova, A. Bedini, F. Chai, B. Christensen, M. Dunbabin, and J. Gao, "Applications of low-cost sensing technologies for air quality monitoring and exposure assessment: How far have they gone?," *Environment International*, vol. 116, pp. 286-299, 2018.
- [3] L. Spinelle, M. Gerboles, M. G. Villani, M. Aleixandre, and F. Bonavitacola, "Field calibration of a cluster of low-cost available sensors for air quality monitoring. Part A: Ozone and nitrogen dioxide," *Sensors and Actuators B: Chemical*, vol. 215, pp. 249-257, 2015.
- [4] P. Zimmerman, N. Peredkov, J. Abdelnour-Nocera, and S. Cleland Woods, "Advancing low-cost air quality monitor calibration with machine learning methods," *Environmental Science & Technology*, vol. 58, no. 10, pp. 4567-4578, 2024.
- [5] J. Bobulski, S. Szymoniak, and K. Pasternak, "An IoT system for air pollution monitoring with safe data transmission," *Sensors*, vol. 24, no. 2, p. 445, 2024.
- [6] N. Muhd Zain, M. Othman, M. A. R. Mohd Rozi, and Z. Paidi, "The development of an IoT-based air quality monitoring system using the Blynk application," *Journal of Computing Research and Innovation*, vol. 9, no. 1, pp. 157-166, 2024.
- [7] C. Aleixandre and M. Gerboles, "Review of small commercial sensors for indicative monitoring of ambient gas," *Chemical Engineering Transactions*, vol. 30, pp. 169-174, 2012.
- [8] D. H. Tsujita, T. Ishida, and T. Moriizumi, "Metal oxide semiconductor gas sensors for air quality monitoring," in *Proc. IEEE Sensors*, 2004, pp. 1383-1386.
- [9] R. Williams, V. Kilaru, E. Snyder, A. Kaufman, T. Dye, A. Rutter, A. Russell, and H. Hafner, "Air sensor guidebook," US Environmental Protection Agency, Washington, DC, EPA/600/R-14/159, 2014.
- [10] X. Xie, I. Semajski, S. Gautama, E. Tsiligianni, N. Deligiannis, and W. Philips, "A review of urban air pollution monitoring and exposure assessment methods," *ISPRS International Journal of Geo-Information*, vol. 6, no. 12, p. 389, 2017.
- [11] Badura, M., et al. (2020), "Evaluation of low-cost sensors for ambient air monitoring," *Atmosphere*, 11(2).
- [12] M. Mead, O. A. Popoola, G. B. Stewart, P. Landshoff, M. Calleja, M. Hayes, J. J. Baldovi, M. W. McLeod, T. F. Hodgson, J. Dicks, and A. Lewis, "The use of electrochemical sensors for monitoring urban air quality in low-cost, high-density networks," *Atmospheric Environment*, vol. 70, pp. 186-203, 2013.
- [13] K. Ashton (2022), "The Internet of Things: Principles and applications," *IEEE Potentials*, 41(3).
- [14] Y. Wang, J. Li, H. Jing, Q. Zhang, J. Jiang, and P. Biswas, "Laboratory evaluation and calibration of three low-cost particle sensors for particulate matter measurement," *Aerosol Science and Technology*, vol. 49, no. 11, pp. 1063-1077, 2015.
- [15] T. Athira, T. Jilesh, and A. S. Kumar, "A review of machine learning models for predicting air quality in urban areas," *Atmospheric Pollution Research*, vol. 13, no. 4, p. 101342, 2022.
- [16] D. Zhang and S. S. Woo, "Real-time localized air quality monitoring and prediction through mobile and fixed IoT sensing network," *IEEE Access*, vol. 8, pp. 89584-89594, 2020.
- [17] Liu et al. (2023), "Edge computing resilience for IoT systems," *IEEE Internet of Things Journal*, vol. 10, no. 2.