Established – 1961 Subject: Internet of Things

SEVA SADAN'S

R. K. TALREJA COLLEGE

OF

ARTS, SCIENCE & COMMERCE

ULHASNAGAR – 421 003

CERTIFICATE

This is to certify that Mr.Ms. Ruchita Uttam Gawade of S.Y. Computer Science (SYCS) Roll No. 2524012 has satisfactorily completed The Internet Of Thing Mini Project entitled Gas Leakage detection with mobile alert during the academic year 2025 – 2026, as a part of the practical requirement. The project work is found to be satisfactory and is approved for submission.

PROF. INCHARGE **HEAD OF DEPT**

_____ _____

INDEX

1. Introduction

1.1 Overview of the Problem

Gas leakage is one of the major safety hazards in residential homes, laboratories, hotels, and industrial environments. Leakage of gases such as LPG (Liquefied Petroleum Gas), Carbon Dioxide (CO_2), Carbon Monoxide (CO), and Ammonia can lead to serious accidents including fire outbreaks, explosions, suffocation, and even death. Many accidents occur due to delayed detection of gas leakage. Traditional methods depend on human smell detection, which is unreliable and dangerous.

In modern smart homes and industries, automated systems are required to continuously monitor the environment and provide real-time alerts. An IoT-based gas detection system helps in identifying leakage at an early stage and prevents major disasters.

1.2 Purpose of the Project

The purpose of this project is to design and implement an intelligent gas leakage detection system using NodeMCU ESP8266 and MQ-135 gas sensor. The system continuously monitors air quality and gas concentration levels. If gas concentration exceeds a predefined threshold value, the system:

- Activates a buzzer (sound alert)
- Turns ON LED indicator
- Switches ON exhaust fan (DC motor)
- Sends notification to user's mobile phone via WiFi

This ensures immediate action can be taken to prevent accidents.

1.3 Importance and Real-Life Applications

This system has wide real-life applications:

- Homes and kitchens
- Chemical laboratories
- Hotels and restaurants
- LPG storage areas
- Industrial plants
- Hospitals
- Parking areas

Advantages:

- Prevents fire hazards
- Reduces health risks
- Low-cost solution
- Real-time remote monitoring
- Automatic ventilation system

1.4 Objectives of the System

The main objectives of this project are:

1. To detect harmful gases using MQ-135 sensor.
2. To process sensor data using NodeMCU ESP8266.
3. To send real-time mobile alerts using IoT platform.
4. To automatically activate buzzer, LED, and exhaust fan.
5. To design a low-cost, reliable, and efficient safety system.

2.Requirement Specification

2.1 Hardware Requirements

1. NodeMCU ESP8266 (WiFi Microcontroller)
2. MQ-135 Gas Sensor
3. Buzzer
4. LED with resistor
5. Motor (Exhaust Fan)
6. 6vBattery
7. Breadboard
8. Connecting Wires
9. Power Supply Adapter

Description of Components

NodeMCU ESP8266:
It is a WiFi-enabled microcontroller used to process sensor data and send mobile notifications through internet.

MQ-135 Sensor:
It detects gases like CO_2, NH_3, smoke, and alcohol. It provides analog output depending on gas concentration.

Motor Driver:
Since NodeMCU cannot directly drive high-current motor, motor driver is used to control exhaust fan safely.

Battery:
Provides independent power supply for motor and system.

2.2 Software Requirements

- Arduino IDE
- ESP8266 Board Package
- Embedded C Programming
- HTTP Protocol

- USB Driver for NodeMCU

2.3 Functional Requirements

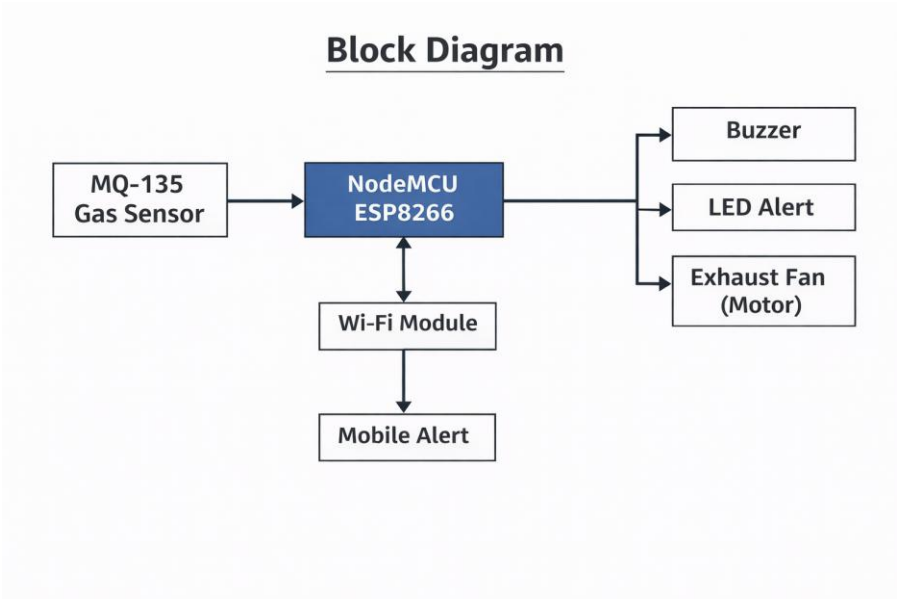
- Continuous monitoring of gas level
- Comparison with predefined threshold
- Automatic alarm activation
- Motor control through motor driver
- WiFi connection setup
- Sending notification to mobile app

2.4 Non-Functional Requirements

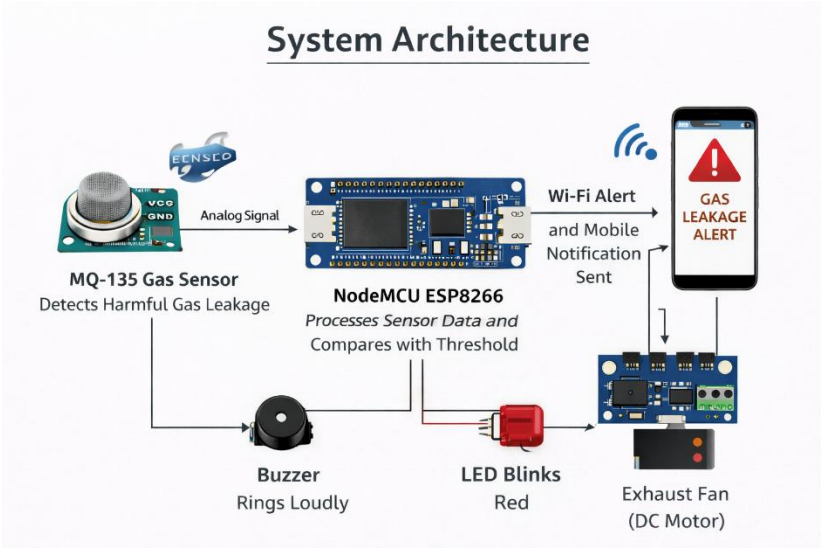
- Low power consumption
- Quick response time (2–3 seconds)
- Reliable internet communication
- User-friendly interface
- High accuracy detection

3.System Design

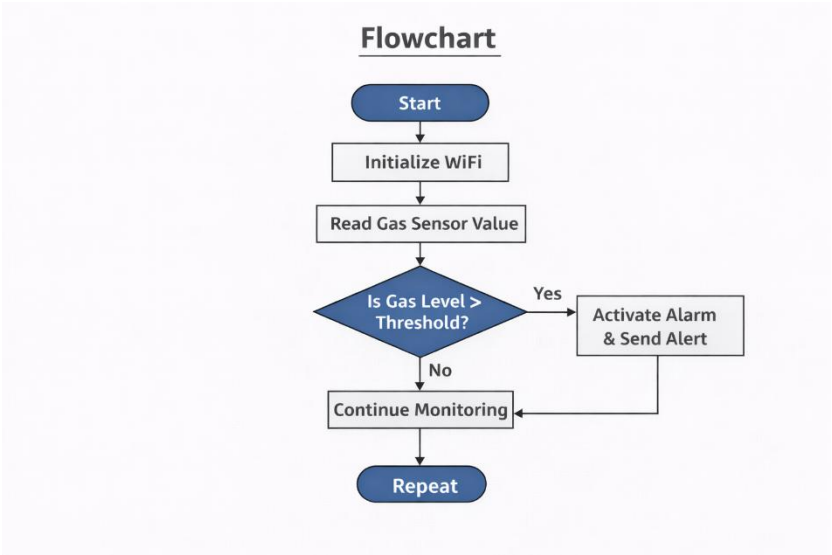
3.1 Block Diagram Explanation



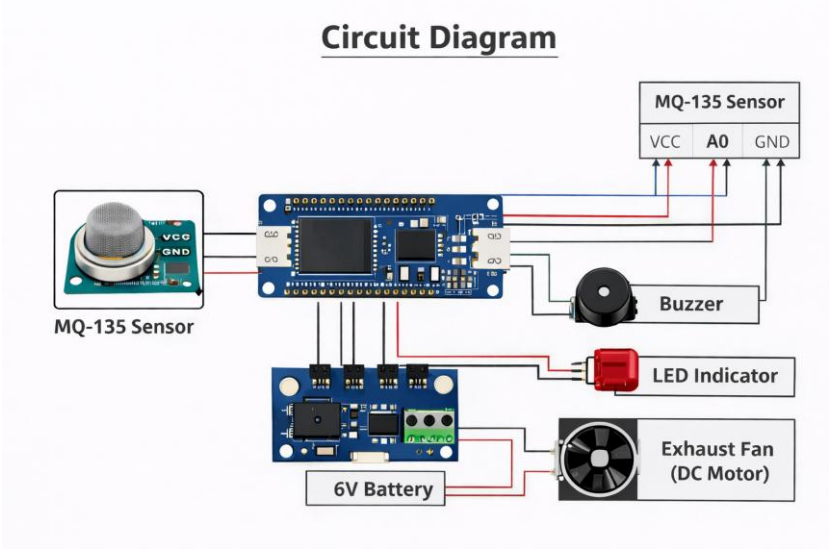
3.2 Architecture Diagram Explanation

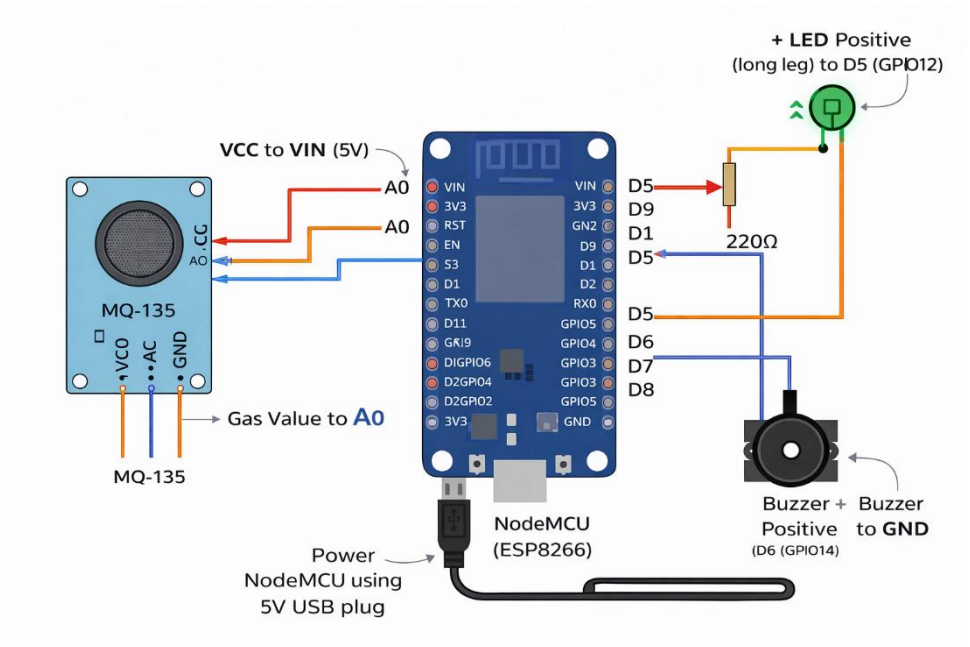


3.3 Flowchart Explanation



3.4 Circuit Design Description





Connections :

MQ-135 sensor:

MQ-135 Pin	Connection
VCC	5V
GND	GND
A0	A0 (NodeMCU)

Buzzer:

Buzzer Pin	Connected To	Purpose
Positive (+)	D6 (GPIO14)	Alert output
Negative (-)	GND	Ground

LED:

LED Pin	Connected To	Purpose
Anode (+)	D5 (GPIO12) via 220Ω resistor	Indicator output
Cathode (-)	GND	Ground

Power Connection:

Power Source	Connected To
USB 5V	NodeMCU USB Port

Power Source	Connected To
VIN	MQ-135 VCC
Common GND	All components

4.Implementation

The gas leakage detection system is implemented using a **NodeMCU ESP8266 microcontroller** which continuously monitors air quality using the **MQ-135 gas sensor**.
When the gas concentration exceeds a predefined threshold:

- LED turns ON
- Buzzer activates
- Exhaust fan (DC motor) starts
- Mobile alert is sent through WiFi

The system operates on a **6V battery supply** connected through a motor driver module for proper power distribution.
The complete implementation consists of:

1. Hardware setup
2. Software programming
3. WiFi configuration
4. Threshold calibration
5. Testing and validation

2. Technology Used

- **Hardware Components**

- NodeMCU ESP8266
- MQ-135 Gas Sensor
- Buzzer
- LED Indicator
- L298N / Motor Driver Module
- DC Motor (Exhaust Fan)
- 6V Battery
- Connecting Wires

- **Software & Platform**

- Arduino IDE

Embedded C / Arduino Programming

- ESP8266 WiFi Library
- HTTP Web Server (for mobile alerts)

- **Communication Technology**

- WiFi (IEEE 802.11 b/g/n)
- TCP/IP Protocol

Programming Logic (Short Explanation)

1.**Initialize System**

- Start Serial communication.
- Set LED, Buzzer, Sensor, and Stepper motor pins.
- Warm up MQ-135 sensor (1 minute).
- Connect NodeMCU to WiFi.
- Start Web Server.

2. **Read Gas Sensor**

- Read analog value from MQ-135 (A0).
- Take 20 readings.
- Calculate average value.
- Store in gasValue.

3.**Compare with Threshold**

- If gasValue > thresholdHigh (100)
  → Gas Detected
- If gasValue < thresholdOff (70)
  → System Safe

(Two thresholds are used to avoid frequent ON/OFF switching — called hysteresis control.)

4. **If Gas Detected**

- Turn OFF Green LED
- Turn ON Red LED
- Turn ON Buzzer
- Rotate Stepper Motor (open ventilation)
- Show "WARNING: GAS LEAKED!" on web page

5. **If Gas Level Normal**

- Turn ON Green LED
- Turn OFF Red LED
- Turn OFF Buzzer
- Rotate Stepper Motor back (close ventilation)
- Show "SYSTEM SAFE" on web page

6. **Web Monitoring**

- Web server refreshes every 2 seconds.
- Displays:
  - Gas Value
  - System Status (Safe / Warning)

Loop Process

The system continuously:

- Reads sensor
- Compares values

- Updates outputs
- Updates web page
- Repeats every 1 second

Sample aurdino code

```
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
#include <Stepper.h>

const char* ssid = "";
const char* password = "";

ESP8266WebServer server(80);

#define Buzzer D5
#define Green  D6
#define Red    D7
#define Sensor A0

#define IN1 D1
#define IN2 D2
#define IN3 D3
#define IN4 D4

int gasValue = 0;
int thresholdHigh = 100;
int thresholdOff  = 70;

bool gasDetected = false;
bool motorMoved = false;

const int stepsPerRevolution = 2048;
Stepper myStepper(stepsPerRevolution, IN1, IN3, IN2, IN4);

void handleRoot() {

  String statusText;
  String statusColor;

  if (gasDetected) {
    statusText = "&#9888; WARNING: GAS LEAKED!";
```

```
    statusColor = "#dc3545";
  } else {
    statusText = "&#10004; SYSTEM SAFE";
    statusColor = "#28a745";
  }

  String html = "<!DOCTYPE html><html><head>";
  html += "<meta charset='UTF-8'>";
  html += "<meta name='viewport' content='width=device-width, initial-
scale=1.0'>";
  html += "<meta http-equiv='refresh' content='2'>";
  html += "<title>Gas Detector</title>";

  html += "<style>";
  html += "body{font-family:Arial;text-
align:center;background:#eef2f7;margin:0;}";
  html += "h1{background:#007bff;color:white;padding:15px;margin:0;}";
  html += ".card{background:white;padding:20px;margin:20px;border-
radius:12px;box-shadow:0 4px 8px rgba(0,0,0,0.1);}";
  html += ".status{font-size:22px;color:white;padding:15px;border-
radius:8px;font-weight:bold;}";
  html += "</style>";

  html += "</head><body>";
  html += "<h1>Gas Detector System</h1>";
  html += "<div class='card'>";
  html += "<h2>Gas Value: " + String(gasValue) + "</h2>";
  html += "<div class='status' style='background:" + statusColor + ";'>";
  html += statusText;
  html += "</div>";
  html += "</div></body></html>";

  server.send(200, "text/html", html);
}

void setup() {

  Serial.begin(115200);

  pinMode(Green, OUTPUT);
  pinMode(Red, OUTPUT);
  pinMode(Buzzer, OUTPUT);
  pinMode(Sensor, INPUT);

  digitalWrite(Buzzer, LOW);
  digitalWrite(Green, HIGH);
  digitalWrite(Red, LOW);
```

```
  myStepper.setSpeed(10);

  delay(60000);

  WiFi.begin(ssid, password);
  Serial.print("Connecting to WiFi");

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("\nWiFi Connected");
  Serial.print("IP Address: ");
  Serial.println(WiFi.localIP());

  server.on("/", handleRoot);
  server.begin();
}

void loop() {

  server.handleClient();

  long sum = 0;
  for (int i = 0; i < 20; i++) {
    sum += analogRead(Sensor);
    delay(5);
  }
  gasValue = sum / 20;

  Serial.print("Gas Value: ");
  Serial.println(gasValue);


  if (gasValue > thresholdHigh) {

    gasDetected = true;

    digitalWrite(Green, LOW);
    digitalWrite(Red, HIGH);
    digitalWrite(Buzzer, HIGH);

    if (!motorMoved) {
      myStepper.step(stepsPerRevolution / 4);
      motorMoved = true;
```

```
    }

    Serial.println("WARNING: GAS LEAKED!");
  }

  else if (gasValue < thresholdOff) {

    gasDetected = false;

    digitalWrite(Green, HIGH);
    digitalWrite(Red, LOW);
    digitalWrite(Buzzer, LOW);

    if (motorMoved) {
      myStepper.step(-(stepsPerRevolution / 4));
      motorMoved = false;
    }

    Serial.println("System SAFE");
  }

  delay(1000);
}
```

Sensor Integration

1. Sensor Used: MQ-135 Gas Sensor
The MQ-135 sensor is used to detect harmful gases in the environment such as:

- Carbon Dioxide (CO_2)
- Ammonia (NH_3)
- Benzene
- Smoke
- Alcohol Vapors

It works based on change in resistance when exposed to different gases.

Most Important: Callibaration (preheating)

Step 1: Preheat Sensor

- First time: Keep ON for **24 hours**
- Normal use: Warm up for **5–10 minutes**

Step 2: Keep in Fresh Air

- Place sensor in **clean outdoor air**
- No smoke or gas nearby

2. Working Principle

The MQ-135 contains:

- A heating element
- A sensing layer

When harmful gas is present:

- Sensor resistance changes
- Output voltage changes
- Analog signal is sent to NodeMCU

Higher gas concentration → Higher analog value.

- **MQ-135 Preheating & Calibration Steps (Proper Method)**

Step 1: Initial Preheating

1. Connect MQ-135 to **5V supply** (mobile charger + USB is OK).
2. Turn ON the sensor.
3. Keep it ON continuously for **24–48 hours** (first time use).
4. Do not switch OFF frequently during first heating.

You kept it ON for 2 days — that is correct.

Step 2: Why Value Dropped from 900 to 50–60?

This is **normal behavior**.

- At the beginning, sensor is unstable → shows very high values (like 900).
- After 1–2 days, internal heater becomes stable.
- Sensor resistance becomes stable.
- Readings drop to normal environmental range (50–100 in clean air).

So **900 → 50–60 drop is normal and correct.**

Step 3: Proper Calibration Environment

For final calibration:

1. Place sensor in **fresh outdoor air**.
2. Avoid smoke, perfume, kitchen gas.
3. Keep it stable for 10–15 minutes.
4. Then calculate R0 value.

Always take calibration value in clean air, not inside house.

Step 4: Take Average Reading

1. Take 20–30 readings.
2. Calculate average value.
3. Use that average to calculate **R0**.
4. Save R0 in your main program.

Final Conclusion

- Your 2-day preheating was correct

- Value drop is normal
- Sensor is now stable
- Now you can do final calibration in fresh air

3. Hardware Connection

MQ-135 Pin	Connection
VCC	5V
GND	GND
A0	A0 (NodeMCU)

Important:

- Sensor requires warm-up time (1–5 minutes minimum, best accuracy after long preheating).
- All grounds must be common.

4. Software Integration

In your code:

#define Sensor A0

Reading sensor value:

gasValue = analogRead(Sensor);

To improve accuracy:

- 20 readings are taken
- Average value is calculated

```
long sum = 0;
for (int i = 0; i < 20; i++) {
  sum += analogRead(Sensor);
  delay(5);
}
gasValue = sum / 20;
```

This reduces noise and gives stable reading.

5. Threshold Calibration

Two thresholds are used:

- thresholdHigh = 100 → Gas detected
- thresholdOff = 70 → System safe

Using two thresholds prevents frequent ON/OFF switching (Hysteresis control).

6.Sensor Data Flow

MQ-135

⬇

Analog Signal (0–1023)

⬇

NodeMCU A0

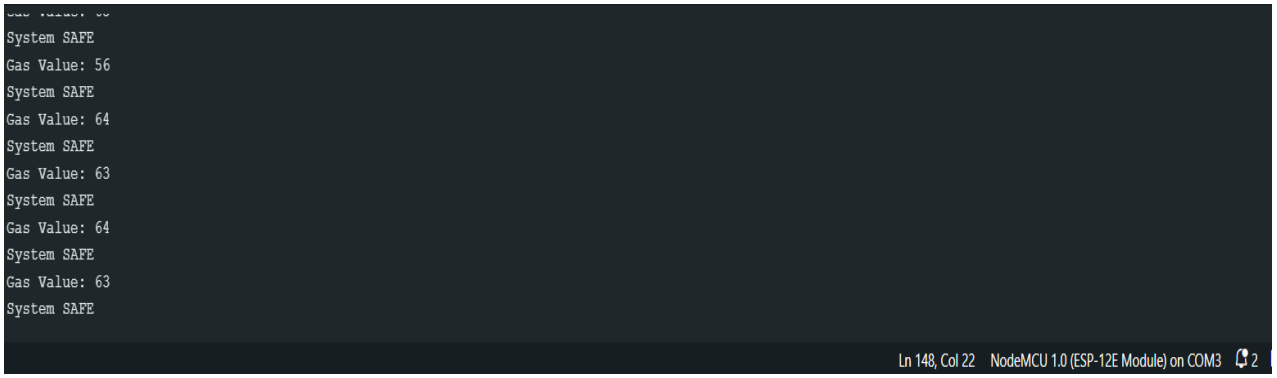
⬇
Comparison with threshold
⬇
Activate alarm / ventilation

6. Communication Protocols

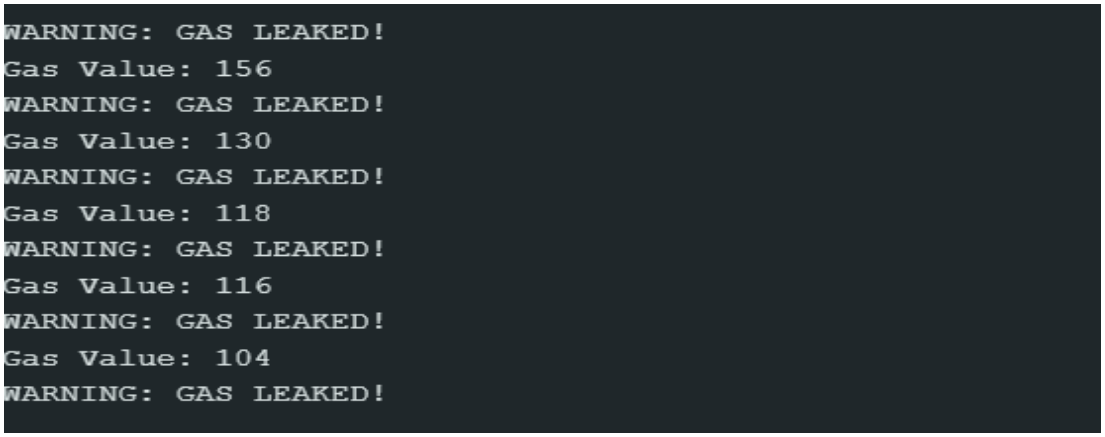
HTTP Protocol

Used to send alert to mobile app or web server.

Screenshots / Code Explanation



Warning Mode



When gas value crosses threshold:

- LED glows red
- Buzzer rings
- Fan starts
- Mobile receives notification:
  "Gas Leakage Detected!"

Advantages of Implementation

- Low cost
- Easy to install
- Wireless monitoring
- Automatic ventilation
- Real-time alerts

Code Explanation:

Code Part	Function
analogRead(A0)	Reads gas sensor value
if (gasValue > threshold)	Checks gas level
digitalWrite(buzzer, HIGH)	Activates alarm
WiFi.begin()	Connects to WiFi
delay(1000)	Wait 1 second

Code Explanation

Gas Leakage Detection System using NodeMCU ESP8266

This program is designed to:

- Detect gas using MQ-135 sensor
- Control LEDs and buzzer
- Rotate stepper motor for ventilation
- Display real-time status on web page

1. Library Inclusion

```
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
#include <Stepper.h>
```

Explanation:

- ESP8266WiFi.h → Connects NodeMCU to WiFi
- ESP8266WebServer.h → Creates local web server
- Stepper.h → Controls stepper motor

2. WiFi Credentials

```
const char* ssid = "";
const char* password = "";
```

Stores WiFi network name and password.

3. Web Server Initialization

```
ESP8266WebServer server(80);
```

Creates web server on **Port 80** (default HTTP port).

4. Pin Definitions

```
#define Buzzer D5
#define Green  D6
#define Red    D7
#define Sensor A0
```

- D5 → Buzzer
- D6 → Green LED
- D7 → Red LED
- A0 → MQ-135 Sensor

Stepper motor pins:

```
#define IN1 D1
#define IN2 D2
#define IN3 D3
#define IN4 D4
```

These control motor rotation sequence.

5. Variables

```
int gasValue = 0;
int thresholdHigh = 100;
int thresholdOff  = 70;
```

- gasValue → Stores sensor reading
- thresholdHigh → Gas detection level
- thresholdOff → Safe return level

Using two thresholds avoids frequent ON/OFF switching (Hysteresis control)

```
bool gasDetected = false;
bool motorMoved = false;
```

- gasDetected → Stores system state
- motorMoved → Prevents repeated motor rotation

6. Stepper Motor Setup

```
const int stepsPerRevolution = 2048;
Stepper myStepper(stepsPerRevolution, IN1, IN3, IN2, IN4);
```

- 2048 steps = full rotation
- Motor rotates 1/4 turn during gas detection

7. Web Page Function

void handleRoot()

This function:

- Checks system status
- Creates dynamic HTML page
- Displays:
  - Gas value
  - System status (Safe / Warning)

Page refreshes every 2 seconds:

```
<meta http-equiv='refresh' content='2'>
```

8. Setup Function

void setup()

Steps:

1. Start Serial Monitor
2. Set pin modes
3. Set initial LED state:
   - Green ON (Safe)
4. Set motor speed

5. Warm up sensor (60 seconds)
6. Connect to WiFi
7. Print IP address
8. Start web server

9. Loop Function

void loop()

Runs continuously.

Step 1: Handle Web Client

server.handleClient();

Allows browser to access web page.

Step 2: Read Gas Sensor

```
long sum = 0;
for (int i = 0; i < 20; i++) {
  sum += analogRead(Sensor);
}
gasValue = sum / 20;
```

- Takes 20 readings
- Calculates average
- Reduces noise

Step 3: Gas Detection Logic

If Gas Detected:

if (gasValue > thresholdHigh)

System actions:

- Green LED OFF
- Red LED ON
- Buzzer ON
- Stepper rotates 1/4 turn

- Displays warning

If Gas Normal:

else if (gasValue < thresholdOff)

System actions:

- Green LED ON
- Red LED OFF
- Buzzer OFF
- Stepper returns to original position
- Displays SAFE

Delay

delay(1000);

Waits 1 second before next reading.

10. Overall Working Summary

1. System connects to WiFi
2. MQ-135 reads gas level
3. Value compared with threshold
4. If gas detected:
    - Alarm ON
    - Ventilation ON
    - Warning on web page
5. If safe:
    - Alarm OFF
    - Ventilation OFF
    - Safe status displayed

- **Key Features in Code**

1.WiFi-based monitoring
2. Real-time web interface
3. Hysteresis control
4.Noise reduction using averaging
5.Automatic ventilation

5.System Testing and Result

The Gas Leakage Detection System was tested under different environmental conditions to verify proper working of the MQ-135 sensor, NodeMCU ESP8266, buzzer, LEDs, stepper motor, and web monitoring system.

5.1 Test Cases

Test Case No.	Test Condition	Expected Result	Actual Result	Status
1	Power ON system	Green LED ON, System SAFE message	Working as expected	Pass
2	Normal air condition	Gas value below threshold	SAFE status shown	Pass
3	Small gas exposure	Gas value increases	Red LED ON, Buzzer ON	Pass
4	High gas exposure	Gas value > thresholdHigh	Motor rotates, Warning displayed	Pass
5	Gas removed	Gas value < thresholdOff	System returns to SAFE	Pass
6	Web page access	Enter IP in browser	Live gas value displayed	Pass

5.2 Input / Output Verification

Input

- Analog gas sensor value (0–1023 range)
- WiFi credentials
- Power supply (6V battery)

Output

Input Condition	Output Response
Gas < 70	Green LED ON, Buzzer OFF
Gas > 100	Red LED ON, Buzzer ON
Gas > 100	Stepper Motor rotates
Gas normal again	Motor returns to original position
Web access	Gas value + Status shown

5.3 Performance Analysis

1. Response Time

- Detection time: ~1–2 seconds
- Web page refresh: Every 2 seconds
- Alert activation: Immediate after threshold crossing

3. Accuracy

- Average reading method improves stability
- Dual threshold reduces false triggering
- Sensor warm-up improves reliability

4. System Stability

- Continuous monitoring without crash
- WiFi reconnection stable
- No frequent ON/OFF flickering due to hysteresis logic

5. Power Consumption

- Low power consumption
- Suitable for battery operation
- Stepper motor consumes higher current during rotation only

5.4 Result

The system successfully:

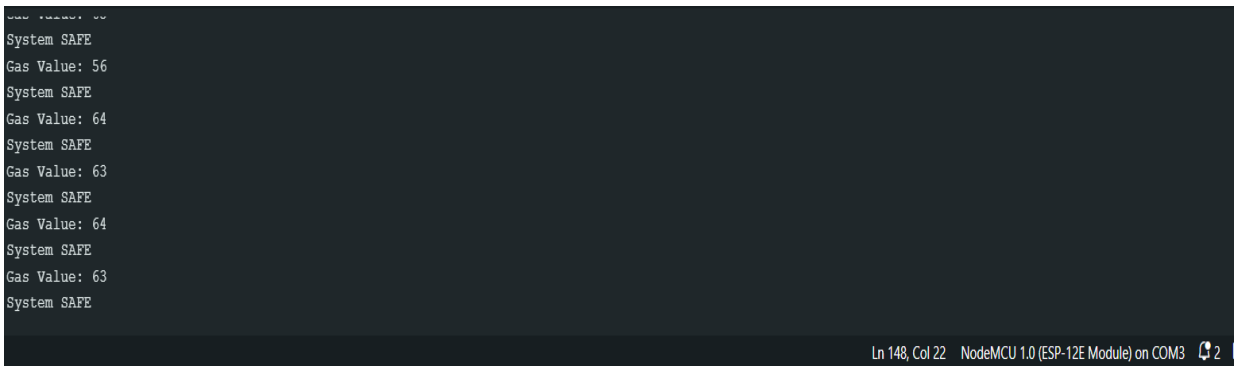
1. Detected gas leakage
2. Activated alarm system
3. Rotated stepper motor for ventilation
4. Displayed real-time data on web server
5. Automatically returned to safe mode

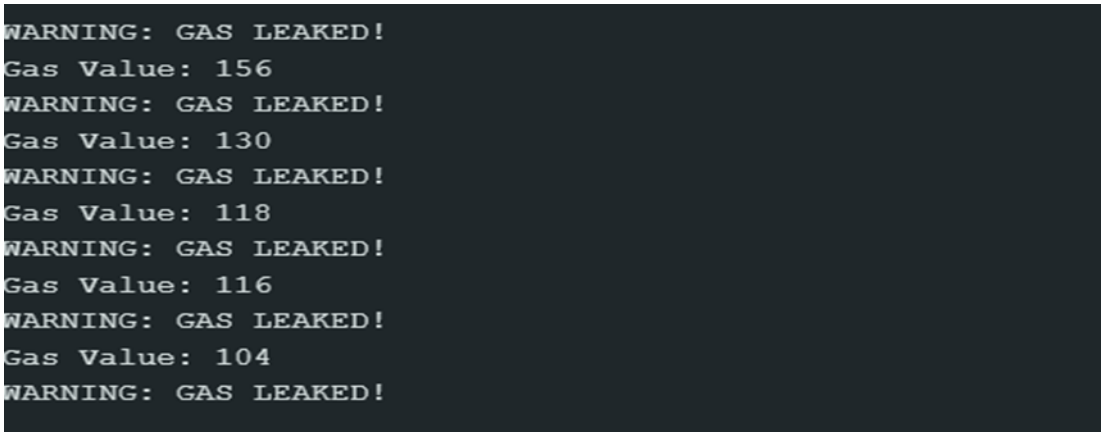
The overall performance of the system is reliable and efficient for indoor gas monitoring applications.

5.5 Result Screenshots

You can include the following screenshots:

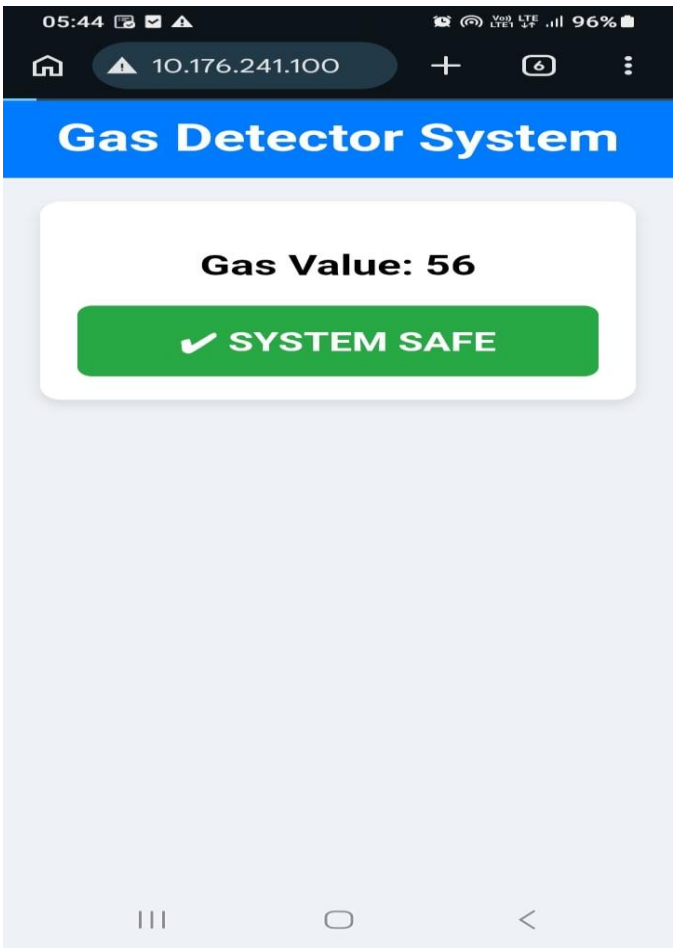
1. Serial Monitor Output





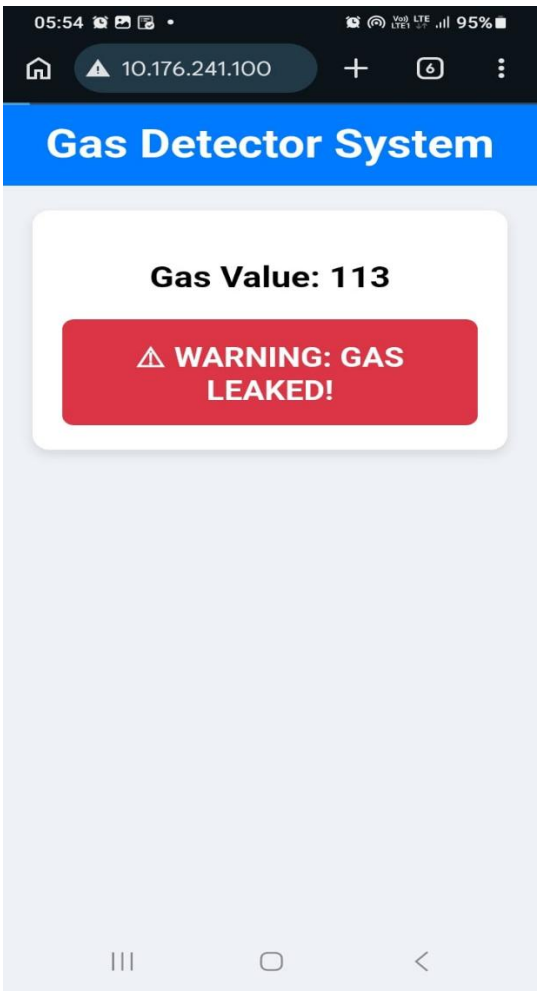
2. Web Page Display (SAFE Mode)

- Gas Value: 60
- Status: SYSTEM SAFE (Green)



4. Web Page Display (Warning Mode)

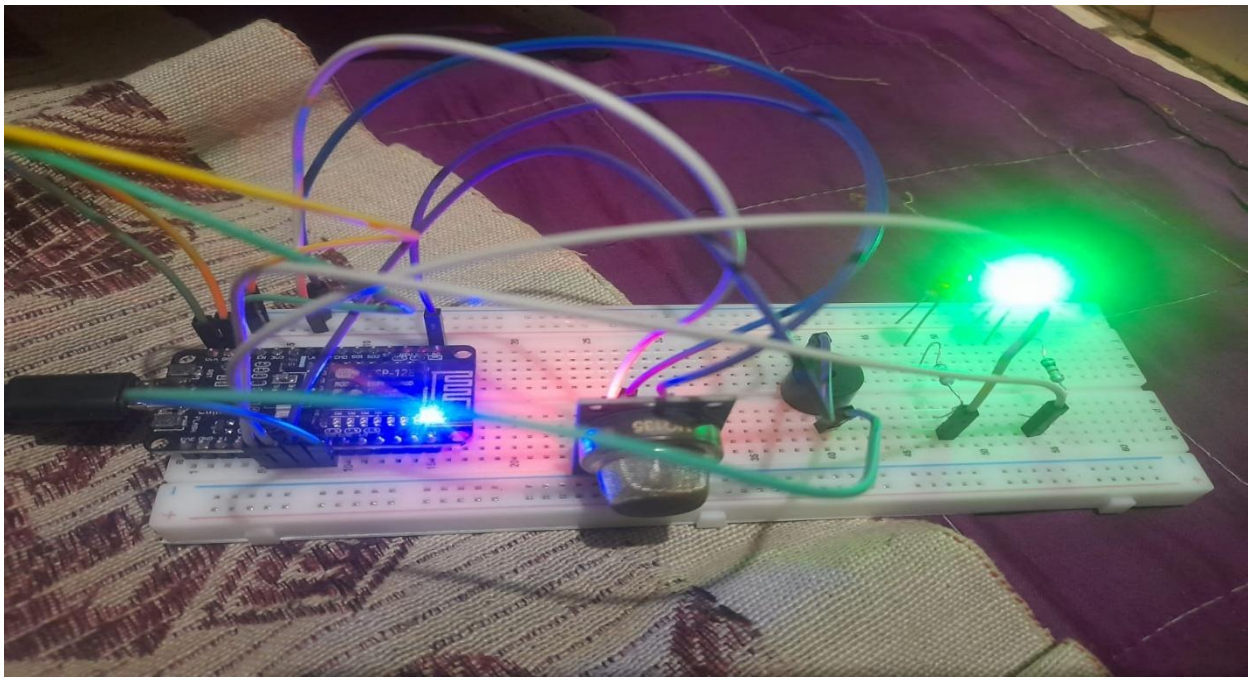
- Gas Value: 150
- Status: WARNING: GAS LEAKED! (Red)



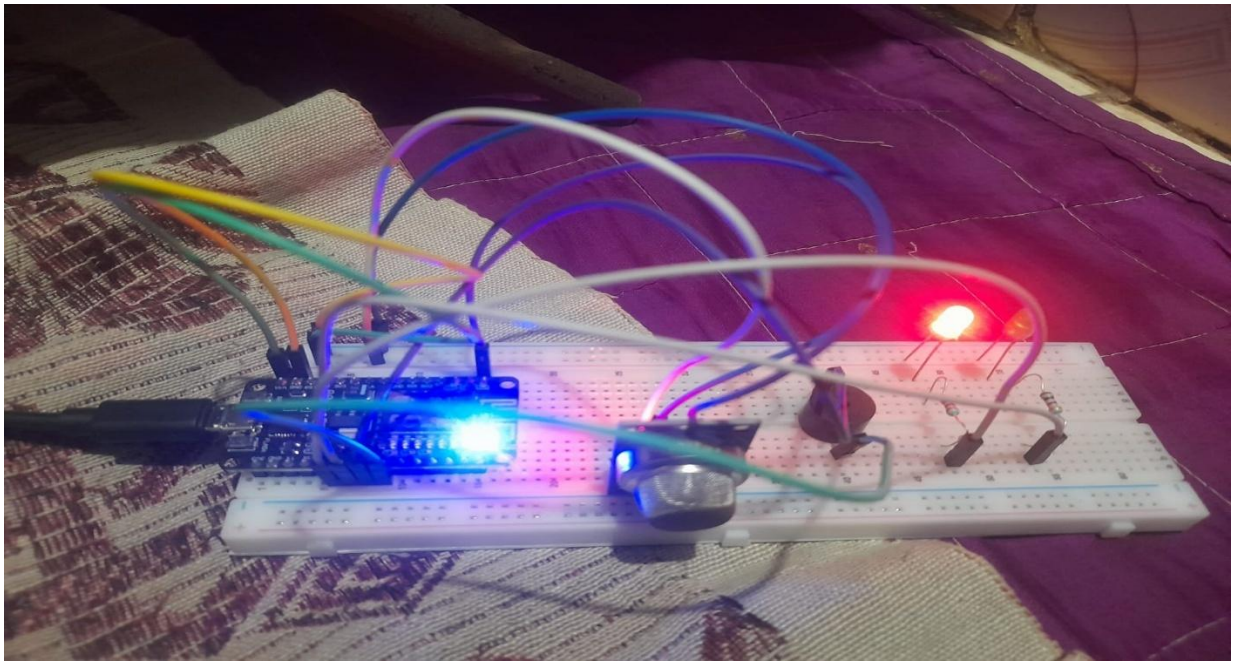
5. Hardware Image During Alarm

- Red LED glowing
- Buzzer ON
- Stepper motor rotated

Safe mode:



Warning mode :



- **Final Result Statement**

The Gas Leakage Detection System using NodeMCU ESP8266 and MQ-135 sensor works successfully with real-time monitoring, alarm activation, ventilation control, and web-based status display.

Future Scope and Conclusion

1. **Multiple Gas Detection**:
   - Integrate additional sensors like **MQ-2, MQ-7, or MQ-9** ,MQ-135to detect specific gases (CO, LPG, Methane, Smoke).
2. **Industrial Application**:
   - Scale the system for **factories, laboratories, and large buildings** with multiple sensor nodes and centralized monitoring.
3. **SMS/Email Alerts**:
   - Integrate **GSM module** or cloud-based service to send **SMS or email alerts** in addition to mobile app notifications.
4. **Real-Time Monitoring Dashboard**:
   - Develop a **web or mobile dashboard** to monitor gas levels continuously with historical data.
5. **Automatic Safety Actions**:
   - Connect the stepper motor to **automatic gas valves or exhaust fans** to control leakage immediately.
6. **Energy Efficiency & Solar Power**:
   - Use **solar panels or rechargeable batteries** to make the system self-sustainable.
7. **AI-Based Analytics**:
   - Use **machine learning** to predict leakage patterns and prevent potential hazards.

REFERENCES

ESP8266 Technical Documentation

MQ-135 Datasheet

Arduino Official Website

Chat Gpt

GLOSSARY

IoT – Internet of Things
ESP8266 – WiFi-enabled Microcontroller
MQ-135 – Gas and Air Quality Sensor
Relay – Electrically Operated Switch
LPG – Liquefied Petroleum Gas