IoT based Gas Detection Using Arduino and ESP8266

by

Khushboo Gandhi

B.Tech., Amity University, India 2010

A Project Report Submitted in Partial Fulfillment of the

Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

University of Victoria

We acknowledge and respect the Lək^{w}əŋən (Songhees and X^{w}sepsəm/ Esquimalt) Peoples on whose territory the university stands, and the Lək^{w}əŋən and W̱SÁNEĆ Peoples whose historical relationships with the land continue to this day.

IoT based Gas Detection Using Arduino and ESP8266

by

Khushboo Gandhi

B.Tech., Amity University, India 2010

Supervisory Committee

Dr. Yvonne Coady, Supervisor

Department of Computer Science

Dr. Anthony Estey, Departmental Member

Department of Computer Science

Abstract

This report presents the development of a scalable Internet of Things (IoT)-based gas detection system leveraging the Arduino Uno, ESP8266 Wi-Fi module, and MQ-5 gas sensor. The primary objective of the system is to detect combustible gases such as methane and propane and to transmit real-time sensor data to a cloud-connected platform.

The prototype serves as a foundational proof of concept for broader safety applications, particularly in industrial environments where gas leaks pose significant hazards. A practical and educational system design has been implemented with attention to hardware interfacing, embedded programming, and network communication.

To complement the core system, a STEM-focused tutorial is provided, introducing learners to sensor integration and cloud data transmission using microcontrollers. The report further outlines a robust future scope wherein the existing system can evolve into a distributed sensor network integrated with Amazon Web Services (AWS) for intelligent alerting, secure data storage, and real-time analytics across geographically distributed facilities.

Table of Contents

List of Figures

List of Tables

Acknowledgements

I would like to express enormous gratitude to my supervisor, **Dr. Yvonne Coady** for granting me the opportunity and trusting me to complete this degree. I'm excessively thankful for her continuous and persistent encouragement, guidance, patience and support.

My **family and friends** for continuously motivating me and believing in me even when it was difficult to do so. Dedicated to the perseverance of my mother.

I'm grateful to my friends for helping me brainstorm possible solutions when I was stuck in limbo, and for their constant support especially in troubleshooting issues.

Lastly, I would like to take the opportunity to thank all the staff at UVic from various departments like **CSC, UVic Wellness Centre, Faculty of Graduate Studies, GARO** and the **International Office** for their endless help and excessive patience, and for playing a vital role in helping me bring this to a fruitful end.

Chapter One

1 Introduction

The need for real-time environmental monitoring is growing rapidly across industries where gas leaks and hazardous emissions pose serious health, operational, and safety risks. [1] Legacy systems often rely on manual inspection or isolated sensors, making them inefficient for detecting and responding to emergencies. The integration of IoT into gas detection not only reduces response times but enables proactive interventions before hazards escalate.

Gas leakage detection plays a critical role in maintaining occupational safety, environmental compliance, and the integrity of equipment in facilities such as oil refineries, chemical plants, and waste processing units. With the increasing digitization of safety infrastructure, smart detection systems can now offer live updates, remote access, and predictive responses powered by cloud computing and sensor fusion.

This report presents the development and evolution of an IoT-based gas detection system using an Arduino Uno, ESP8266 Wi-Fi module, and an MQ-5 gas sensor. The system functions as a compact, autonomous sensing unit that collects gas concentration data and uploads it to a cloud platform for remote monitoring.

The document is structured to offer a holistic view of the system:

- A technical overview of the hardware, software, and communication layers.
- A tutorial section designed for beginners interested in electronics and coding.
- A forward-looking section mapping the pathway to distributed deployments via AWS, MQTT, and cloud analytics.

By combining educational relevance with practical deployment potential, the project becomes both a learning tool and a prototype for scalable, cloud-connected safety infrastructure.

1.1 Motivation

The motivation for this project lies in addressing the dual objective of delivering a practical IoT-based gas detection solution and creating a valuable educational resource. The use of accessible, low-cost components like the Arduino Uno and MQ-5 sensor, combined with cloud connectivity via ESP8266, makes it feasible to design scalable systems without significant overhead. Rather than being a response to a specific incident, this project aims to contribute to the broader movement toward proactive environmental safety and learning accessibility.

Arduino Uno R3 is the ideal board for beginner-friendly learning as it is a robust, scalable, and extremely capable tool to expand projects further. Since it is an open-source platform, abundant

documentation, tutorials and examples are readily available for increasing proficiency in a short amount of time. Arduino platforms are widely adopted in both academic and prototyping environments, making them ideal for demonstrating concepts such as real-time monitoring, data logging, and remote alerting. By incorporating cloud integration through AWS, this system bridges educational and industrial use cases, showcasing how a simple prototype can serve as a stepping stone to fault-tolerant and intelligent safety networks.

1.2 Approach

The approach taken involves a bottom-up design methodology, beginning with the construction of a basic sensor prototype on the Arduino platform. The system is first validated through local operation and cloud data transmission, followed by educational enhancements for learning, and ending with a future roadmap for industrial-scale deployment using modern cloud tools such as AWS IoT Core, Lambda, and DynamoDB. Code snippets are presented in the main body for logic illustration, with full-length versions stored in the appendices for reference.

Chapter Two

2 Technical Overview

This section provides a technical overview of the IoT-based gas detection system. It breaks down the primary components of the system including the hardware used, software implementation, and how data flows from sensor detection to cloud storage. This modular breakdown highlights how the prototype was constructed, validated, and how it supports both educational and practical use cases.

2.1 Hardware Summary

The hardware used in this gas detection prototype includes low-cost, widely available components that make the system both accessible for learners and applicable for scalable sensor networks. The core components include:

- **Arduino Uno**: A microcontroller development board based on 8-bit ATmega328P MCU IC (Microcontroller Unit- Integrated Circuit) [2] [3] and comprises of other components like crystal oscillator, serial communication, voltage regulator, etc. for support. It is used to read gas sensor values and control the output devices. It forms the central control unit of the system, interfacing with input and output components to ensure logical response.

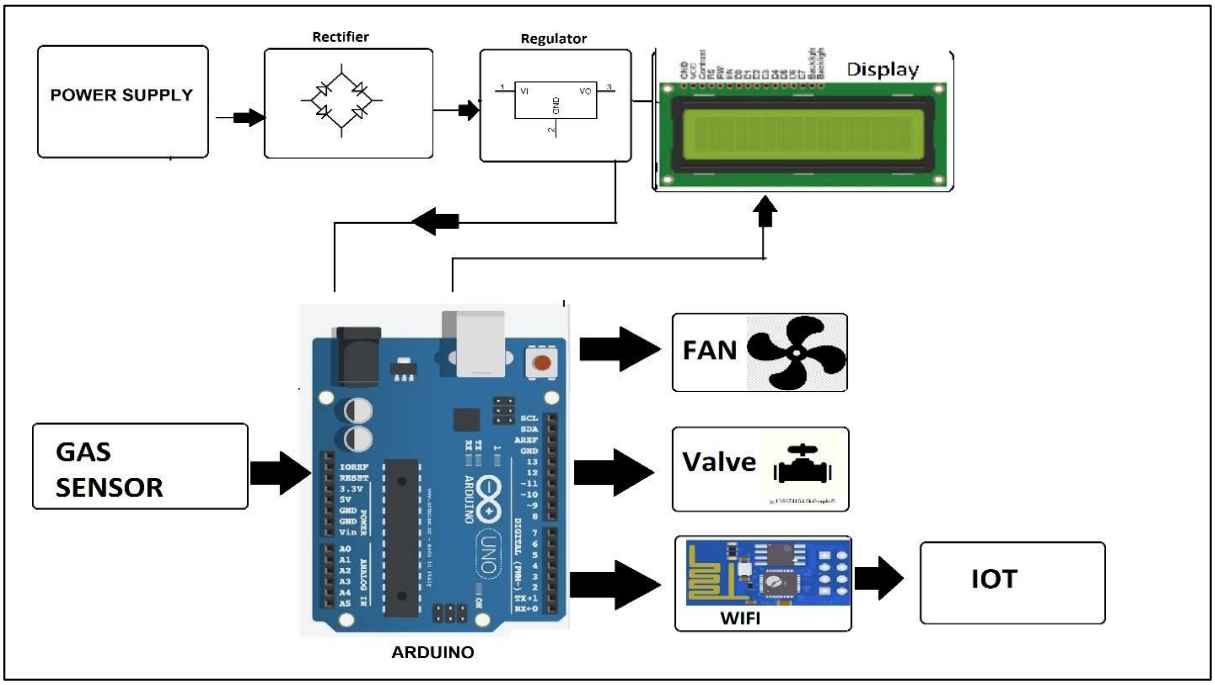
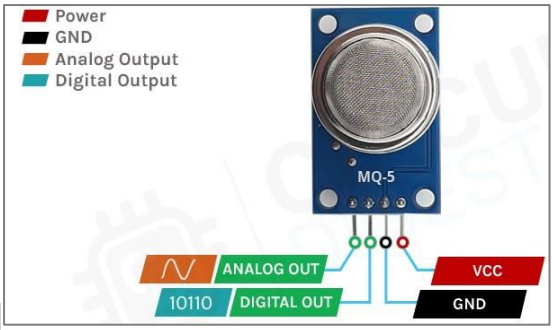


Figure 2.1: Arduino Uno serving as the central microcontroller in the gas detection system by KGandhi

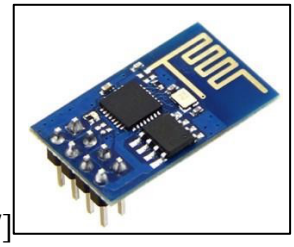
- **MQ-5 Gas Sensor**: This sensor detects a range of combustible gases by sensing changes in resistance when exposed to gases like methane or propane. It outputs an analog signal that the Arduino reads and compares against preset safety thresholds. [4]



[5]

Figure 2.2: MQ-5 gas sensor with 4 pins- DOUT, AOUT, GND, VCC

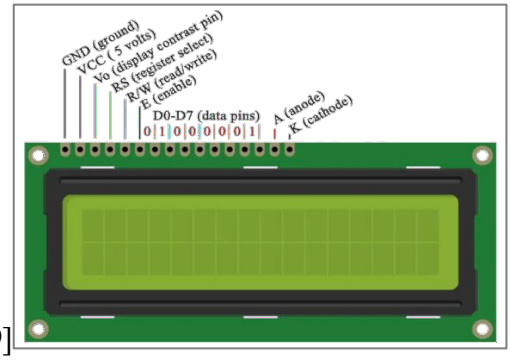
- **ESP8266 Wi-Fi Module***: Enables real-time wireless communication between the Arduino and the cloud-based IoT platform. It is programmed to send HTTP requests or MQTT packets depending on the chosen service. [6]



[7]

Figure 2.3: ESP8266 Wi-Fi module

- **LCD 16x2 Display**: Displays live gas readings directly from the sensor, allowing for local monitoring without needing to access the cloud dashboard. [8]



[9]

Figure 2.4: LCD display to show sensor output in real time

* ESP8266 is obsolete now and there are many ESP32 variants designed to replace and transition it.

- **Buzzer, LEDs, Fan, Valve**: These components act as actuators for alerting or responding to high gas levels. The buzzer and red LED alert users audibly and visually, while the green LED indicates normal conditions. A small DC fan or solenoid valve can simulate ventilation or emergency shutdown mechanisms.
  Refer to **Figure 2.1** for full hardware prototype showing LED, buzzer, fan, valve, and LCD connections.
- **PCB with Top View Wiring**: A custom PCB design was fabricated to organize the components on a single board for better durability and test deployment.

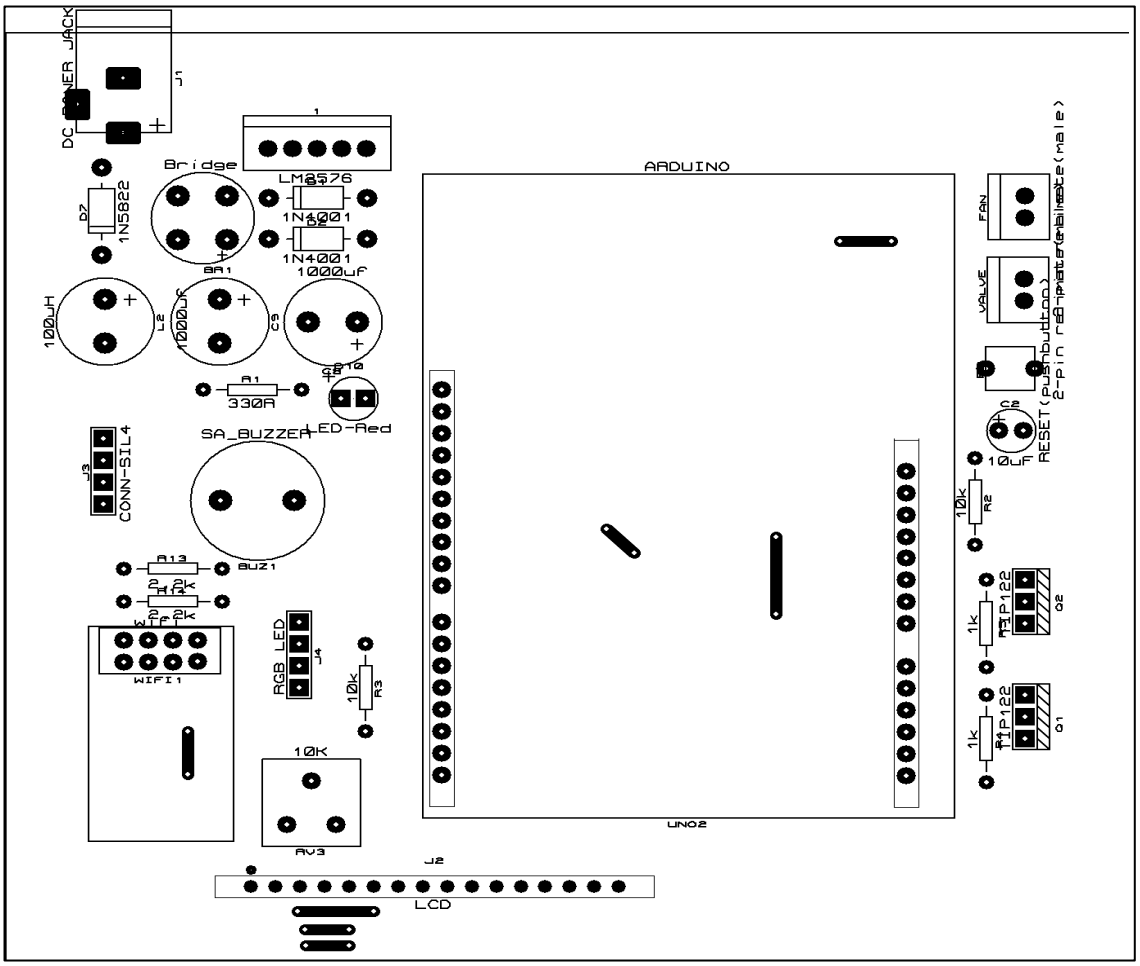


Figure 2.5: Top view of final PCB layout for with soldered components, matching the prototype logic used in breadboard testing by KGandhi

- The full list of all the **Hardware Components** along with their usage in this setup are listed in Table 2.1 below.

Component	Quantity	Description
Arduino Uno	1	Microcontroller
ESP8266 Wi-Fi Module	1	IoT Communication Module
MQ-5 Gas Sensor	1	Gas leakage detection

LCD 16x2	1	Real-time value display
LEDs (Red/Green)	2	Status indicators
Buzzer	1	Audio alert
Fan and Valve	1 each	Simulated actuator control
Resistors, wires	Various	For voltage dividers and signal control
Power Supply	1	5V DC Adapter or USB

Table 2.1: Bill of materials

2.2 Software Overview

The software implementation of this gas detection system is designed to be lightweight yet functional, using the Arduino IDE to write, compile, and upload embedded code to the microcontroller. The primary goal is to read real-time gas concentration values, process them logically, and initiate both local and remote responses when hazardous levels are detected.

Key Functional Elements:

- **Sensor Reading & Threshold Logic**
  The system continuously reads analog values from the MQ-5 sensor. These readings are compared against predefined thresholds to determine whether a gas leak is suspected. The system is calibrated such that any value exceeding the upper threshold activates alerts.
- **Local Feedback & Output Control**
  Based on sensor readings, the Arduino activates output components:
  - Red LED and buzzer for gas leak alerts
  - Green LED when conditions are normal
  - Optionally, a fan or valve is triggered for simulated ventilation or mitigation
- **Real-Time Data Display**
  An LCD module displays the gas value, updating continuously for immediate visibility. This helps verify functionality and provides visual confirmation during testing and deployment.
- **Wi-Fi Communication with IoT Platform**
  Using the ESP8266 Wi-Fi module, the system connects to a specified wireless network and transmits sensor data to an IoT platform.

  This is done through:
  - Basic AT commands
  - HTTP GET requests to cloud servers
  - Optional MQTT-based payloads (for future scope)

- **Minimal Delay Management**
  A non-blocking delay (usually 10–15 seconds) is included after each data transmission to ensure:
    - The network module doesn't get overwhelmed
    - Users can view LCD data before it refreshes
    - Power consumption is minimized

Snippets of this logic are provided in Section 2.4. Full source code is documented in Appendices A & B.

2.3 Flowcharts and Circuit Diagrams

This section provides visual representations of the system logic, and the electrical layout used in the gas detection prototype. These diagrams are essential for understanding both the software decision-making process and the physical component connections.

2.3.1 Program Logic Flowchart

The following flowchart describes the high-level logic used in the Arduino program:

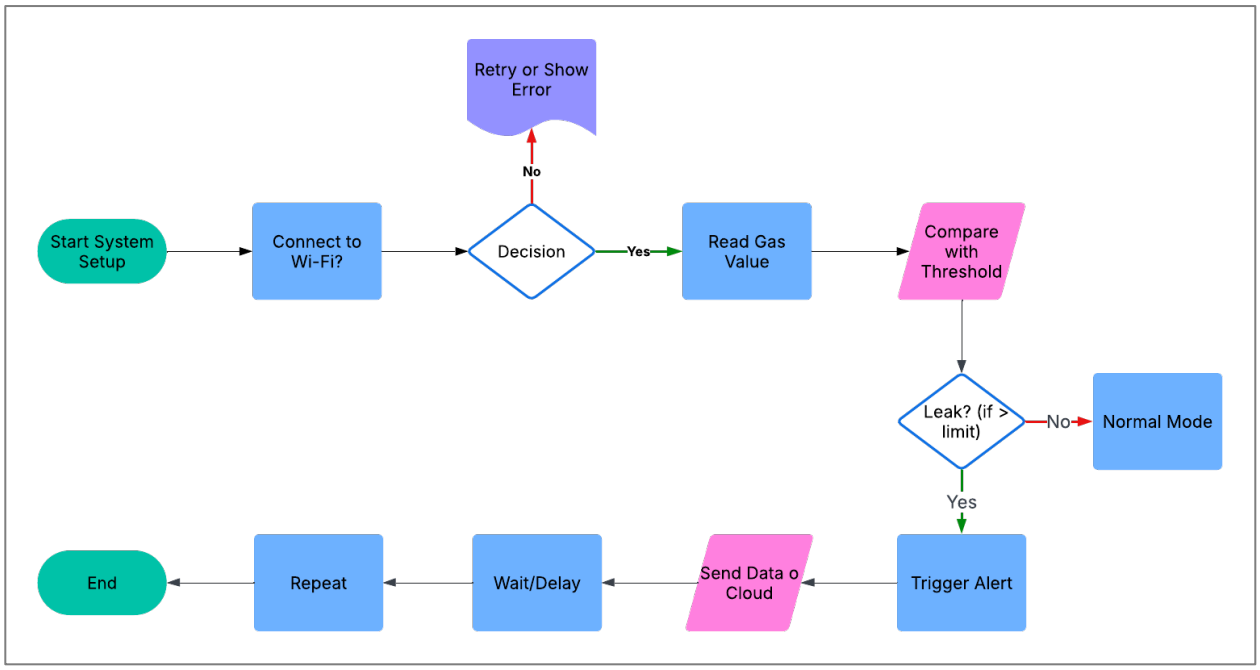


Figure 2.6: System Logic Flowchart for Gas Detection by KGandhi

2.3.2 Circuit Diagram Overview

The circuit schematic was designed using standard prototyping techniques and is translated into a fabricated PCB. The diagram below shows the main connections among the components:

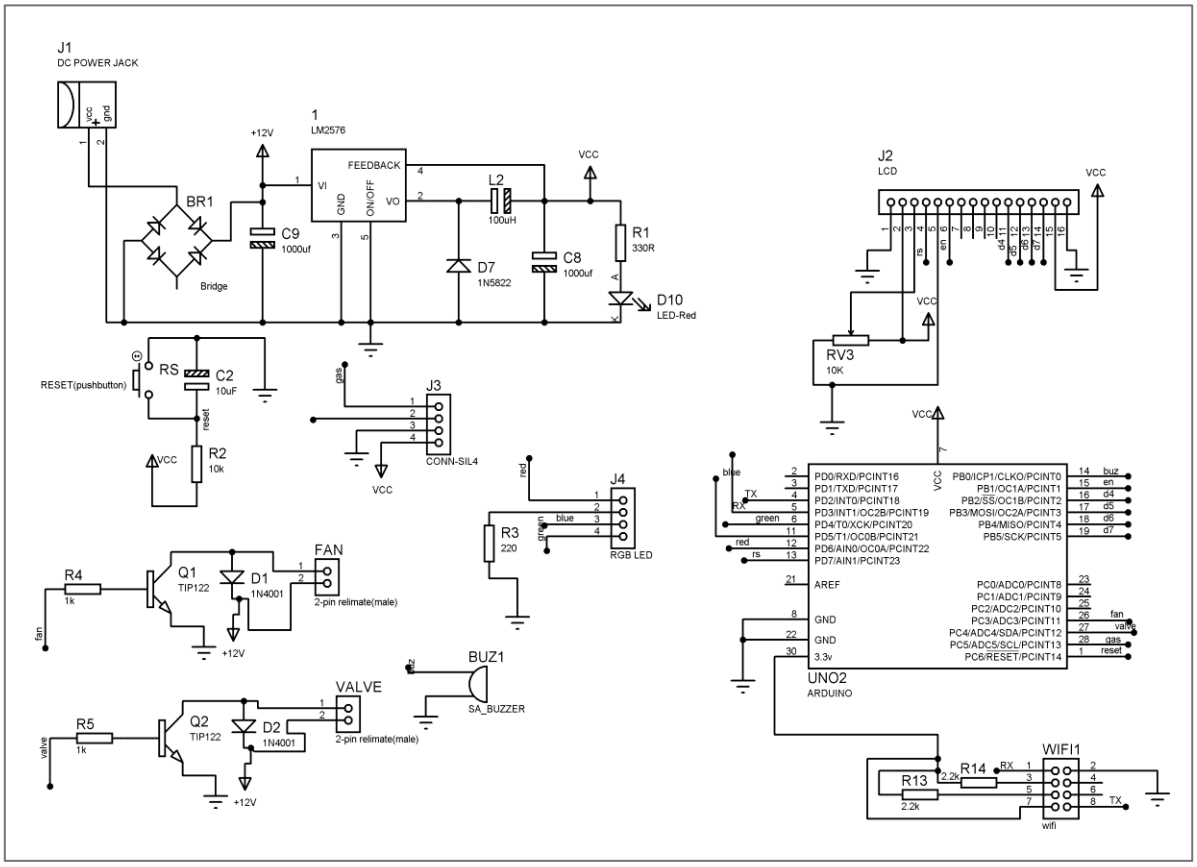


Figure 2.7: Circuit Diagram of the Gas Detection System by KGandhi

- The MQ-5 gas sensor is connected to the **analog input (A0)** of the Arduino Uno.
- The ESP8266 module is interfaced using **TX/RX pins** via a voltage divider for safe operation.
- Output devices (LEDs, buzzer, fan, valve) are connected to **digital pins D2 to D6** with current-limiting resistors.
- The LCD module uses either **I2C*** [10] **or 4-bit mode** for display communication.
- The system is powered via **USB or a regulated 5V adapter**.

Refer to Figure 2.5 earlier in the report for the assembled PCB layout.

* Inter-integrated circuit (I2C) protocol

2.3.3 Arduino Pin Mapping for Gas Detection System Components

This table outlines the connection details between the Arduino and various system components, specifying the corresponding pins and their operating modes (analog or digital).

Component	Function	Arduino Uno Pin (Name)	Pin Number (Physical)	Notes
MQ-2 Gas Sensor	AO (Analog Out)	A0	A0	Analog gas reading
	DO (Digital Out)	D2	4	Optional digital threshold
	VCC	5V	-	Power (Do **not** use 3.3V)
	GND	GND	-	Ground
ESP8266 Module	TX	D8 (Arduino RX)	14	Via voltage divider (ESP TX → Arduino RX)
	RX	D7 (Arduino TX)	13	Via voltage divider (Arduino TX → ESP RX)
	VCC	3.3V	-	Power from Arduino 3.3V only (not 5V!)
	GND	GND	-	Shared Ground
Green LED	Positive Lead	D3	5	Via 220Ω resistor
	Negative Lead	GND	-	Ground
Red LED	Positive Lead	D4	6	Via 220Ω resistor
	Negative Lead	GND	-	Ground
Buzzer	Positive Lead	D5	11	Active High
	Negative Lead	GND	-	Ground
I2C LCD Display	SDA	A4	27	I2C Communication Line
	SCL	A5	28	I2C Communication Line
	VCC	5V	-	Power
	GND	GND	-	Ground
Voltage Divider	D7/D8 to ESP8266	1kΩ + 2kΩ Resistors	-	Protect ESP8266 from 5V TX line

Table 2.2: Pin Mapping Table by KGandhi

2.4 Code Logic and Delay Management

This section outlines the core programming logic of the gas leakage detection system written in Arduino C/C++. The program is structured to support autonomous gas monitoring, actuator control, and cloud communication.

2.4.1 Initialization and Setup

Upon startup, the `setup()` function initializes the system's hardware and software interfaces:

- Begins communication with the ESP8266 and serial monitor.
- Sets all relevant pins for sensors, actuators, and indicators.
- Produces two audible beeps to confirm boot.
- Initializes a 16×2 LCD and shows the system title.
- Connects the ESP8266 to a Wi-Fi network using AT commands.

```
void setup() {
  esp8266.begin(9600);          // Initialize ESP8266 serial at 9600 baud
  Serial.begin(9600);           // Debug serial for monitoring

  // Setup all I/O pins
  pinMode(GAS_SENSOR, INPUT);
  pinMode(RED_LED, OUTPUT);
  pinMode(GREEN_LED, OUTPUT);
  pinMode(BUZZER_PIN, OUTPUT);
  pinMode(VALVE_PIN, OUTPUT);
  pinMode(FAN_PIN, OUTPUT);

  beep();                       // Beep twice as power-on signal

  // Initialize the LCD with a welcome message
  lcd.begin(16, 2);
  lcd.clear();
  lcd.print(F("IOT Gas Leakage"));
  lcd.setCursor(0, 1);
  lcd.print(F("   Detector!"));
  delay(2000);                  // Allow time for user to read LCD

  esp8266.listen();             // Set ESP8266 in listening mode
  init_wifi();                  // Initialize Wi-Fi connection
  delay(2000);                  // Wait to ensure stable connection
}
```

2.4.2 Continuous Monitoring in Loop

The `loop()` function runs continuously and manages system behavior using a simple state machine:

- **State 0** – Connects to the IoT server and confirms with an LCD message.
- **State 1** – Requests threshold values (`gas_min`, `gas_max`), then shows them.

- **State 2** – Continuously reads gas sensor, maps to %, shows reading, checks for leaks, and updates the server.

```
void loop() {
  switch (state) {
    case 0:
      hit_link("1", id, pass);                // Request connection to IoTGecko
      show("Connected to", "IotGecko!");      // Display success
      delay(2000);
      state = 1;                              // Move to threshold fetching
      break;

    case 1:
      getThresholds();                        // Fetch min/max gas levels
      delay(3000);                            // Allow time to view message
      state = 2;                              // Begin monitoring
      break;

    case 2: {
      int raw = analogRead(GAS_SENSOR);       // Read analog gas sensor
      int val = map(raw, 0, 1023, 0, 100);    // Convert to 0-100% scale

      // Display gas concentration percentage
      lcd.clear();
      lcd.print(F("Gas Value: "));
      lcd.setCursor(11, 0);
      lcd.print(val);
      lcd.print(F(" %"));

      resetActuators();                       // Ensure safe state by default
      digitalWrite(GREEN_LED, HIGH);          // Green LED = safe

      checkLeak(val);                         // If above threshold, trigger alert

      hit_link2("1", id, pass, val);          // Upload value to cloud
      delay(1000);                            // Wait before next read
    } break;
  }
}
```

2.4.3 Threshold-Based Decision Logic

When the gas value exceeds `gas_max`, the system activates alarms and safety measures including:

- Sounding a buzzer
- Turning on ventilation

- Closing the gas valve
- Red LED as a visual warning

```
void checkLeak(int value) {
  if (value > gas_max) {
    lcd.clear();
    lcd.print(F("Leakage detected"));
    lcd.setCursor(0, 1);
    lcd.print(F("Valve Closed! "));

    digitalWrite(BUZZER_PIN, HIGH);  // Sound alarm
    digitalWrite(VALVE_PIN, HIGH);   // Shut off gas valve
    digitalWrite(FAN_PIN, HIGH);     // Start fan
    digitalWrite(RED_LED, HIGH);     // Show danger signal
    delay(3000);                     // Hold alert for 3 seconds
  }
}
```

2.4.4 Delay Management

- **Startup delays** like `delay(2000)` and `delay(3000)` give users time to read LCD messages.
- **Loop delay** (`delay(1000)`) throttles how often sensor data is sent to the cloud.
- While `millis()`-based timing could reduce blocking, `delay()` simplifies behavior and aligns with system use.

***Note**: Thresholds (`gas_min`, `gas_max`) are assigned only after establishing a Wi-Fi connection via the ESP8266.*

Full code is available in **Appendix A**: Arduino Code for Prototype System**.**

Chapter Three

3 STEM Tutorial: Learning Arduino Through a Gas Detection System

This section provides a complete hands-on learning experience using Arduino as the foundation for building an IoT-based gas detection system. Designed for students, educators, and self-learners, the tutorial assumes no prior experience and introduces essential concepts through a real-world use case: detecting gas leaks and responding via alerts and cloud monitoring.

3.1 Basic Definitions

3.1.1 Defining Arduino

Arduino is an open-source hardware and software platform used for building digital devices that interact with the physical world. The most commonly used board, the Arduino Uno features analog and digital I/O pins [3], a USB interface for programming, and onboard power regulation. It is a beginner-friendly, but powerful enough for advanced automation and IoT systems.

It's widely adopted in:

1. Education (STEM labs, robotics)
2. Prototyping (IoT sensors, automation)
3. Real-world applications (smart homes, safety systems)

3.1.2 IoT

Internet of Things is the concept of connecting physical devices (sensors, actuators) to the internet. This allows devices to collect, send, and act upon data, often remotely. In this tutorial, Arduino is used as the local device and the ESP8266 module provides Wi-Fi connectivity to transmit sensor data to the cloud.

3.1.3 Key Concepts

- A **microcontroller** [11] is a tiny computer embedded inside the Arduino board (e.g., ATmega328P on the Uno).
- The board has **digital and analog input/output pins**, used to interact with sensors and actuators.
- The **Arduino IDE (Integrated Development Environment)** [12] is the software where you write code (**called sketches**) and upload it to Arduino Uno board via USB.

3.2 Project Objective and Learning Outcomes

Learners will:

- Understand how gas sensors work and interact with microcontrollers
- Wire and configure a complete system on a breadboard
- Write Arduino code to read sensor values and trigger alerts
- Display data on an LCD and send it to an IoT platform
- Simulate the project in Wokwi [13] or Tinkercad [14] if hardware is unavailable

This project is framed around a **real-world problem**: detecting gas leaks in time to trigger alerts and enable remote monitoring which is a life-saving application in industrial or home safety systems.

3.3 Understanding the MQ-5 Gas Sensor

The MQ-5 [15] sensor is commonly used to detect LPG[1], methane, propane, hydrogen, and smoke**.** It provides an analog voltage output that corresponds to the concentration of gas in the environment.

How the MQ-5 Works:

- **Heating Element (Heater Coil):** Internally, the sensor contains a small heating element (typically tungsten) that maintains a constant temperature to keep the sensing layer reactive.
- **Gas-Sensitive Layer (SnO_2 - Tin Dioxide):** The sensing element is coated with tin dioxide (SnO_2), which changes its electrical resistance in the presence of combustible gases.
    - Clean air = high resistance
    - Gas presence = lower resistance
- **Analog Output Signal:** As the resistance changes, it alters the voltage across a load resistor. This voltage is read via an analog pin on the Arduino/microcontroller in the 0-1023 range (10-bit ADC[2] = 0 - 5V) which is mapped to a threshold to detect leakage.
- **Warm-Up Time:** The MQ-5 sensor requires a preheating time of 30–60 seconds to stabilize and provide reliable readings.

3.4 Components Required

Component	Quantity	Description
Arduino Uno	1	Main microcontroller board

[1] Liquefied Petroleum Gas
[2] Analog-to-Digital Converter**:** converts analog signals (like voltages) into digital numbers.

MQ-5 Gas Sensor	1	Detects combustible gases
ESP8266 Wi-Fi Module	1	Sends data to cloud
LCD 16x2 Display (I2C)	1	Displays gas concentration
Red/Green LEDs	2	Indicate danger/safe levels
Buzzer	1	Sound alert when threshold is crossed
Breadboard + Jumper Wires	—	For building the circuit
Resistors (220Ω, 1kΩ, 2kΩ)	—	For LEDs and voltage dividers
USB Cable	1	Connect Arduino to PC

Table.3.1: List of hardware components

3.5 Wiring the Hardware

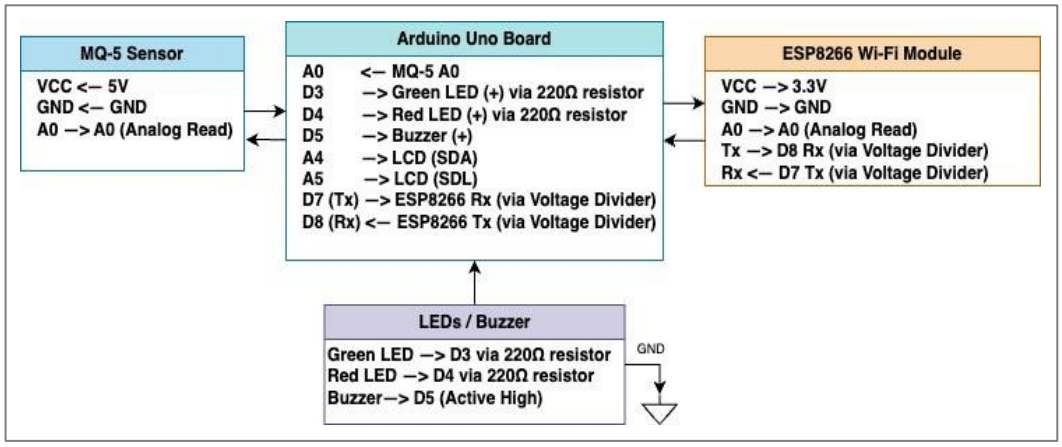


Figure 3.1: Wiring Layout with Arduino, MQ-5, LCD, and ESP8266 connections by KGandhi

3.6 Setting up the Arduino IDE

- Download and install Arduino IDE.
- Connect your Arduino Uno via USB to your computer.
- Go to **Tools → Board** [16] and select *Arduino Uno.* This guides the IDE on protocols to be used to upload the sketch and how to compile it.

Figure 3.2: Selecting Arduino Uno board in Tools

- Go to **Tools → Port** and select the correct port which is the actual board connected to your computer (mostly, these are automatically recognized by the Arduino IDE).

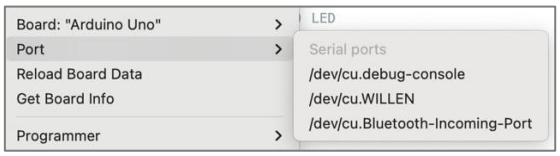


Figure 3.3: Selecting the correct port in Tools

Install required libraries:

  - LiquidCrystal_I2C (LCD)
  - SoftwareSerial (ESP communication)
  - StringSplitter (Adds splitting String functionality, i.e., making substrings)
  - (Optional) ESP8266WiFi for advanced network control

3.7 Programming Arduino

- To learn how to start coding and run your first program on the Arduino IDE, access the multiple pre-made programs meant for learning already accessible on the Arduino website: https://docs.arduino.cc/built-in-examples/.

- Load the example code for blinking an LED on the Arduino compiler by following this path Files>Examples>0.1Basics>Blink. Access the original code and instructions here : https://docs.arduino.cc/built-in-examples/basics/Blink/

- To see the results of the how the code works without the necessity of access to hardware, this easy-to-follow tutorial and simulation on Tinkercad for "Blinking an LED" is a great way to see the collaboration between code on the Arduino compiler and a wired setup.

```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);  // turn the LED on (HIGH is the
voltage level)
  delay(1000);                      // wait for a second
  digitalWrite(LED_BUILTIN, LOW);   // turn the LED off by making the
voltage LOW
  delay(1000);                      // wait for a second
}
```

Figure 3.4: Code snippet for Blinking an LED displays the basic functioning of a sketch

- Arduino uses C++ as the basis to write and interpret syntax and code. The basics of Arduino programming: Program Structure, Functions, Variables, Operators, Loops, Conditions, Objects, Inputs & Outputs are vital in creating an understanding of how logic flows in a sketch. Access this clear and comprehensible article to get a quick grasp to get started on your first Arduino program: https://techexplorations.com/guides/arduino/begin/ls6/

3.8 Writing and understanding a sketch

There are 2 main elements of an Arduino sketch:

- **setup() Function:** Runs once when the Arduino starts. Used for initializing pins, displays, and serial communication.
- **loop() Function:** Runs continuously. Contains logic for reading sensors, displaying data, triggering alerts, and sending data to the cloud.

sketch_test.ino

```
1   void setup() {
2     // put your setup code here, to run once:
3
4   }
5
6   void loop() {
7     // put your main code here, to run repeatedly:
8
9   }
10
```

Figure 3.5: Arduino sketch with setup() and loop() functions

3.9 Code Overview

In this section, we'll walk through the code that runs on the Arduino. The code will do the following:

- **Initialize the components**: Set up sensors and actuators.
- **Connect to Wi-Fi**: Using the ESP8266 module to send data.
- **Read Gas Sensor Data**: Continuously monitor the gas sensor for any changes.
- **Take Action**: If a high gas level is detected, trigger the alarm, close the valve, and turn on the fan.
- **Send Data to the Cloud**: Upload the gas level to an IoT platform for monitoring.

3.9.1 Initial Setup and Wi-Fi Configuration: `setup()`

1. Initializing Components
   - **Purpose:** Prepare all the hardware pins and serial interfaces the system will need:
     i. ESP8266 for Wi-Fi.
     ii. Serial monitor for debugging.
     iii. Sensors/actuators (gas sensor input, LEDs, buzzer, valve, fan).
   - **Details:**
     i. esp8266.begin(9600) and Serial.begin(9600) *[17] s*tart the two serial ports at 9600 baud.∗
     ii. Each pinMode(...) call configures a pin as INPUT (sensor) or OUTPUT (LEDs, buzzer, valve, fan).

```
esp8266.begin(9600);  // Initialize the ESP8266 serial connection
Serial.begin(9600);     // Initialize the serial monitor
pinMode(GAS_SENSOR, INPUT);  // Set gas sensor as input
pinMode(RED_LED, OUTPUT);     // Set red LED as output
pinMode(GREEN_LED, OUTPUT);  // Set green LED as output
pinMode(BUZZER_PIN, OUTPUT); // Set buzzer as output
pinMode(VALVE_PIN, OUTPUT);  // Set valve as output
pinMode(FAN_PIN, OUTPUT);     // Set fan as output
```

2. Connecting to Wi-Fi
   - **Purpose:** Establish a Wi-Fi connection so the Arduino can communicate with the cloud platform.
   - **Details:**
     i. `AT+RST` resets the module to a known state.
     ii. `AT+CWMODE=1` configures it as a Wi-Fi station (client).
     iii. `AT+CWJAP="SSID","PASS"` provides credentials to join your network.

```
esp8266.println("AT+RST");  // Reset the ESP8266
esp8266.println("AT+CWMODE=1"); // Set the Wi-Fi mode to Station
esp8266.println("AT+CWJAP=\"YourSSID\",\"YourPassword\"");  // Connect to Wi-Fi
```

3.9.2 Main Program Loop (State Machine)

- **Purpose:**
  Drive the three main phases of operation—connect**,** fetch thresholds, and monitor—using a simple state machine.

∗ Baud rate = 9600 indicates that the serial port is capable of transferring a maximum of 9600 bits/s.

- **Details:**
  - **State 0**: Calls `hit_link` to connect to the IoT server and displays success.
  - **State 1**: Invokes `getThresholds` to fetch `gas_min` and `gas_max` values from IoT platform.
  - **State 2**:
    i. Reads the gas sensor and converts the raw reading into a percentage.
    ii. Prints the value to Serial and updates the LCD.
    iii. Resets actuators, lights the green LED, then calls `checkLeak` to handle any over-threshold condition.
    iv. Sends the live reading to the server via `hit_link2`.

```
void loop() {
  switch (state) {
    case 0:
      hit_link("1", id, pass);
      show("Connected to", "IotGecko!");
      delay(2000);
      state = 1;
      break;

    case 1:
      getThresholds();
      delay(3000);
      state = 2;
      break;

    case 2: {
      int raw = analogRead(GAS_SENSOR);
      int val = map(raw, 0, 1023, 0, 100);
      Serial.println("Gas Value: " + String(val) + "%");
      lcd.clear();
      lcd.print(F("Gas Value: "));
      lcd.setCursor(11, 0);
      lcd.print(val);
      lcd.print(F(" %"));
      resetActuators();
      digitalWrite(GREEN_LED, HIGH);
      checkLeak(val);
      hit_link2("1", id, pass, val);
      delay(1000);
    } break; }}
```

3.9.3 Connecting to Wi-Fi and Sending Data

- **Purpose:** Show how to upload the current gas reading to the cloud dashboard using an HTTP GET request.

- **Details:**
  - `client.connect(...)` opens a TCP connection on port 80.
  - The `url` string embeds the `gasLevel` value.
  - `client.print(...)` sends a minimal HTTP GET command with the correct Host header.

```
if (client.connect("api.iotgecko.com", 80)) {
  String url = "/update?value=" + String(gasLevel);
  client.print("GET " + url + " HTTP/1.1\r\nHost: api.iotgecko.com\r\n\r\n");
}
```

3.9.4 Testing and Calibration

- **Purpose:** Verify that your gas sensor is responding correctly and set accurate threshold values for reliable detection.
- **Details:**
  i. **Testing:**
    - Open the Arduino Serial Monitor to observe live sensor readings.
    - Introduce a non-flaming smoke source (e.g., unlit lighter, incense) near the sensor.
    - Typical readings:
      - Idle (clean air): ~200
      - Smoke present: ~600

  ii. **Calibration:**
    - Allow the MQ sensor to warm up for 24–48 hours for stable baseline readings.
    - Adjust `gas_min` and `gas_max` in your IoT dashboard or code based on observed idle and smoke values.
    - (Optional) Test with different gas types to account for cross-sensitivity.

3.9.5 Flowchart: Sensor-to-Action Logic

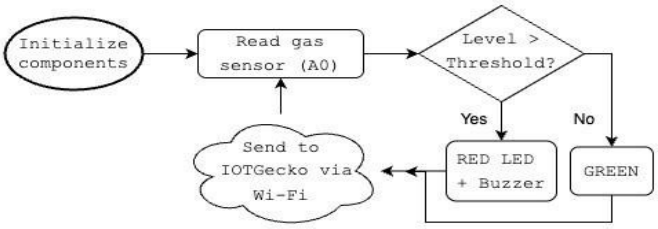


Figure 3.6: Sensor to Action Logic Flowchart by KGandhi

- **Purpose:** Provide a visual roadmap of the system's decision process, helping beginners follow the code logic.
- **Details:**
  i. Initialization: Set up all pins, displays, and Wi-Fi.
  ii. Read Sensor: Continuously sample the gas sensor on analog pin A0.

iii. Threshold Check: Compare reading to `gas_max`.
   - No: Turn on the green LED (safe).
   - Yes: Activate red LED, buzzer, valve, and fan (danger).

iv. Data Upload: Regardless of state, upload the latest reading to the cloud.

v. Loop: Repeat indefinitely.

3.9.6 Web Dashboard Integration

- **Purpose:** Learn how to visualize and monitor sensor data over time using an online IoT platform.
- **Details:**
  i. Platform Setup:
     - Create a new channel on ThingSpeak**,** IotGecko**,** or similar.
     - Copy the API key provided by the platform.
  ii. Data Visualization:
     - Configure charts to display gas readings versus time.
     - Set up dashboard widgets (e.g., gauges, line graphs).
  iii. Alerts & Triggers:
     - Define alert conditions (e.g., reading > `gas_max`).
     - Configure email or mobile notifications when a leak is detected.

3.10 Simulating Without Hardware

Wokwi (Recommended)

- https://wokwi.com
- Simulate MQ-5, LCD, and Arduino
- Real-time data and code testing

Tinkercad Circuits

- https://tinkercad.com/learn/circuits
- Simulates LEDs, LCD, MQ-5 (but not ESP8266)

3.11 Understanding the Data Flow

This architecture allows remote safety alerts and real-time environmental tracking as displayed in Figure 3.6 and detailed in Table 3.2 below.

Component	Function	Output/Next Step
MQ-5 Sensor	Detects gases (LPG, CO, smoke)	Sends gas data to Arduino

Arduino Uno	Processes sensor data	Sends to LCD and ESP8266
LCD Display	Displays real-time gas levels locally	Triggers local alerts if thresholds are met
Wi-Fi Module (ESP8266)	Connects Arduino to the internet, transmits data	Sends to Cloud Dashboard
Cloud Dashboard	Logs and visualizes sensor data in real time	Enables remote monitoring & alerting
Local Alerts	Provides on-site safety notifications (buzzer/LED)	Immediate awareness
Mobile App / Remote Monitoring	Accesses cloud data for remote tracking and alerts	User receives safety notifications anywhere

Table 3.2: System Data Flow Overview

Key Benefits of This Architecture:

- **Real-time Monitoring**: Both locally via LCD and remotely via the cloud.
- **Instant Alerts**: Local buzzers + remote notifications ensure quick action.
- **Scalability**: Can be integrated with more sensors or smart home systems.
- **Mobility**: Cloud-connected data allows tracking from mobile devices.

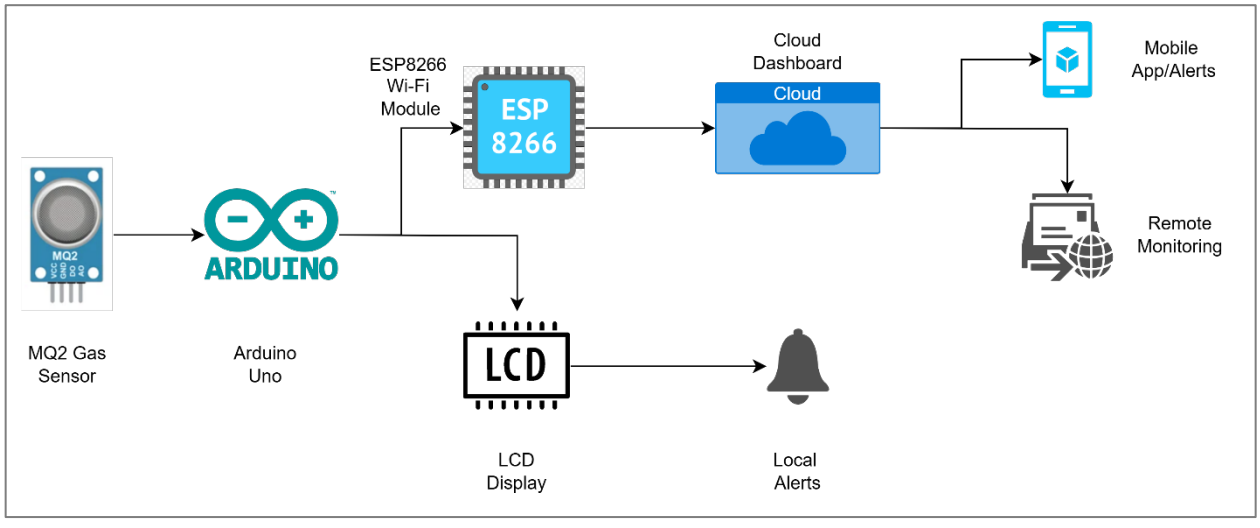


Figure 3.7: Gas Sensor IoT Flow by KGandhi

3.12 Additional Learning Resources

Platform	**What You Can Learn**
Arduino Project Hub [18]	Beginner to advanced Arduino projects
Arduino Programming [19]	Arduino programming language and other compatible languages
Hackster [20]	Learning hardware, from beginner to pro
Tinkercad Circuits [14]	Breadboarding, coding, simulating
Random Nerd Tutorials	IoT projects using ESP8266 and Arduino
ThingSpeak Guide	IoT data logging & dashboarding & cloud APIs
Wokwi Simulator	Arduino simulation and debugging
Circuit.io	Arduino projects for beginners
Tech Explorations	Arduino tutorials for beginner and intermediate Makers

For learners who wish to dive deeper, this setup can be extended into home automation or industrial safety systems. Full source code for connecting to another IOT platform, ThingSpeak is available in Appendix B: Basic Arduino + ThingSpeak Tutorial Code for further learning.

Chapter Four

4 Future Scope -Distributed AWS Integration for Real-World Deployment

As the gas detection prototype demonstrates the feasibility of low-cost sensor-based IoT systems, the next logical step is to scale this design to industrial environments through distributed sensing and cloud connectivity. Leveraging platforms like **Amazon Web Services (AWS)** can transform this Arduino-based prototype into a scalable, intelligent, and fault-tolerant safety network.

4.1 Introduction

To transition from a single-sensor educational build to a large-scale deployment, this section explores the design of a **distributed gas sensing mesh** architecture. The goal is to allow **real-time monitoring**, **automated alerts**, and **historical data logging** across multiple facilities using **ESP8266 + AWS integration**.

4.2 Limitations of the Current System

Limitation	**Description**
Single-point monitoring	Covers only one location at a time
Local-only alerts	No remote communication for alerts
No recovery mechanisms	System is vulnerable to node failure
No data persistence	Sensor data is not stored long-term
Static thresholding	No intelligent adaptation to patterns

Table 4.1: Limitations of current setup

4.3 Distributed Methane Sensing Mesh Architecture

Recent research has emphasized the role of edge computing in optimizing gas detection networks by enabling preliminary data filtering and fault tolerance at the sensor level [21]. By embedding lightweight analytics into each node, the architecture can offload minor decisions locally while still publishing critical data to AWS IoT Core using MQTT.

In a distributed setup:

- Multiple **ESP8266-based nodes** are deployed across zones.

- Each node reads gas concentration using MQ-series sensors.
- Data is published via **MQTT** protocol to **AWS IoT Core**.
- AWS **Lambda functions** process data and trigger alerts or log to databases.

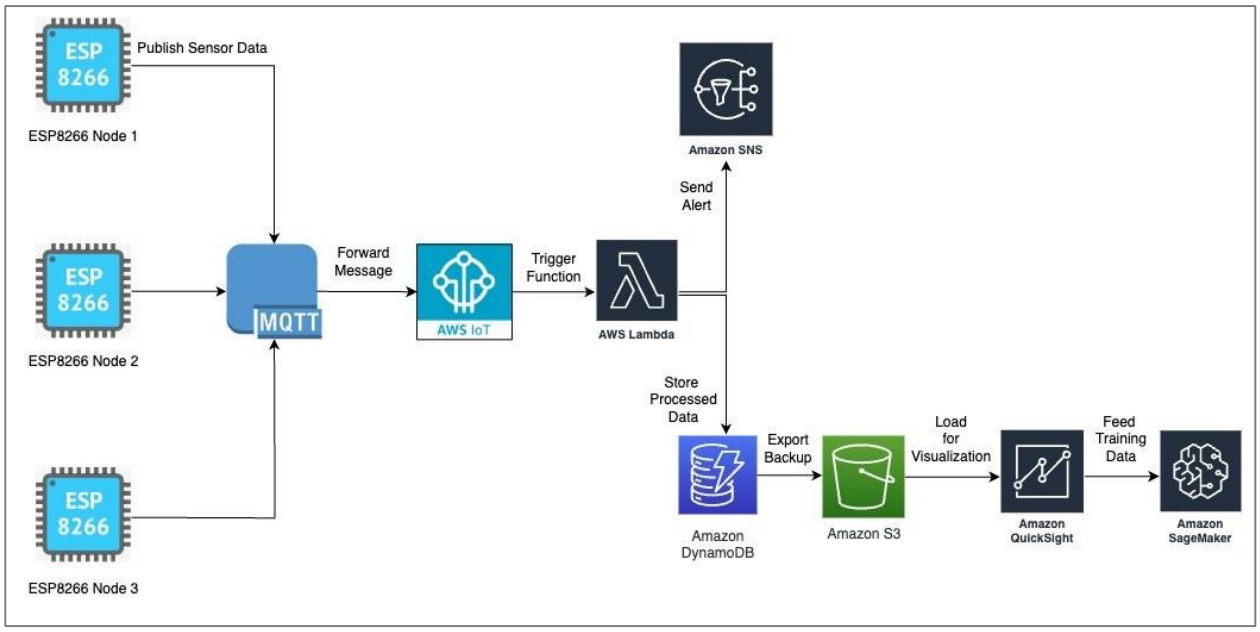


Figure 4.1: IoT Data Flow with AWS Services by KGandhi

4.4 AWS-Based IoT Architecture

Building scalable, secure, and fault-tolerant systems over MQTT requires robust cloud-side infrastructure and protocol handling. IBM's research in cloud resilience outlined the best practices for handling message retention, TLS encryption, and retry mechanisms over MQTT-based cloud deployments [22]. AWS documentation further emphasizes secure device onboarding, role-based access control (IAM), and certificate-based TLS handshakes for production-grade IoT deployments. [23] [24] [25]

Component	**Purpose**	**Description**
ESP8266 / NodeMCU	Sensor node hardware	Sends gas readings via MQTT
MQTT	IoT communication protocol	Lightweight, reliable pub/sub
AWS IoT Core [26]	Device communication gateway	Secure message broker

AWS Lambda	Serverless processing logic	Triggers alerts, stores data
Amazon DynamoDB	Real-time database	Stores timestamped sensor values
Amazon S3	Backup/archive storage	Long-term data storage
AWS SNS	Real-time alerting	Sends SMS, email, or app notifications
AWS QuickSight	Visualization/dashboarding	Generates interactive graphs
Amazon SageMaker	Predictive analytics (optional)	Trains ML models for trend forecasting

Table 4.2: Breakdown of AWS IoT Components in the Proposed System

4.5 Real-World References & Applications

Similar deployments have been implemented at scale across various industries. Alam et al. [27] proposed an AI-enabled IoT gas detection network for industrial sites using ESP and cloud alerts, demonstrating low-latency response times in real-world testing. In addition, AWS's own reference architectures showcase successful implementations of methane detection in oil and gas facilities with real-time dashboards and push alerts.

Application	**Details / Source**
Automated Pipeline Corrosion Monitoring with AWS IoT Core [28]	End-to-end pipeline monitoring solution using AWS IoT core and analytics services
Microgrid for integrating renewable energy systems [29]	AWS IoT analytics platform for microgrid operation management
Low-cost wireless sensor network for particulate matter monitoring [30]	Cloud-linked particulate matter monitoring: Implementation, calibration, and field-test
Everynet [31]	Data Platform Integrating AWS IoT Core and LoRaWan Networks
Microcontrollers Lab Tutorial [32]	MQTT alert system using ESP8266 + AWS IoT Core
Connect device and cloud: Arduino with ESP8266 (EPS32) and AWS IoT [33]	How to Send Data from ESP8266 to AWS IoT Core using MQTT

Table 4.3: Real-World examples of utilizing AWS-IOT

4.6 Glossary of Cloud & IoT Terms

- **MQTT** Message Queuing Telemetry Transport: Lightweight communication protocol for sensors
- **TLS v1.2**: Encryption standard for secure messaging
- **IAM**: Identity and Access Management for AWS roles
- **Lambda**: Serverless function that executes code on-demand
- **QuickSight**: AWS service to build data dashboards
- **SageMaker**: Machine learning platform for model training and deployment
- **SNS:** Simple Notification Service is fully managed service that provides message delivery from publishers (producers) to subscribers (consumers). [34]

4.7 Summary

This section presents a clear and scalable roadmap for transitioning from a basic gas detection prototype to a cloud-integrated industrial monitoring network. The proposed architecture leverages modern AWS tools to enable:

1. Scalable multi-node deployment
2. Real-time remote monitoring
3. Secure, fault-tolerant data management
4. Predictive maintenance and analytics

By extending the Arduino + ESP8266 platform with MQTT and AWS services, the system evolves into a future-ready solution for **safety-critical industrial use**. With this architecture, the project can serve as the **first building block** of a larger industrial safety system, demonstrating the power of Arduino, cloud platforms, and real-time data fusion in critical infrastructure monitoring.

Conclusion

This project demonstrates how microcontroller platforms like Arduino, when paired with internet-enabled modules such as the ESP8266, can be effectively applied to real-world sensing and alert systems. The gas detection prototype successfully integrates hardware and software to provide local alerts and cloud connectivity for continuous environmental monitoring.

In addition to the practical implementation, the project has served an educational purpose by introducing learners to sensor interfacing, basic programming, and data transmission using IoT concepts. The accompanying tutorial aims to simplify these concepts while maintaining technical accuracy, allowing the system to be recreated or expanded by those new to embedded systems.

The proposed AWS-based distributed architecture extends the system's capabilities for industrial use, offering a reliable framework for remote monitoring, data storage, and predictive analytics. This positions the project at the intersection of embedded systems, cloud computing, and smart infrastructure. Beyond the prototype, it presents a foundation for building a fully distributed, fault-tolerant safety network powered by services like AWS IoT Core, Lambda, and DynamoDB.

By combining accessible components with scalable cloud tools, the system balances feasibility with functionality, making it adaptable for both educational and industrial applications.

References

Alam, Mahmudul, et al. “An IoT Based Real-Time Environmental Monitoring System for Developing Areas.” Journal of Advanced Research in Applied Sciences and Engineering Technology, vol. 52, no. 1, pp. 106–21. semarakilmu.com.my, https://doi.org/10.37934/araset.52.1.106121.

Alam, Mamtaz. “Connecting ESP8266 to Amazon AWS IoT Core Using MQTT.” How To Electronics, 15 Jan. 2022, https://how2electronics.com/connecting-esp8266-to-amazon-aws-iot-core-using-mqtt/.

Arduino MQ5 Gas Sensor Tutorial - How MQ5 Gas Sensor Works and Interfacing MQ5 Gas Sensor with Arduino. https://circuitdigest.com/microcontroller-projects/interfacing-mq5-gas-sensor-with-arduino.

AWS IoT Core Documentation. https://docs.aws.amazon.com/iot/.

Chakraborty, Sudip, and Sreeramana Aithal. “Smart LPG Leakage Monitoring and Control System Using Gas Sensor (MQ-X), AWS IoT, and ESP Module.” International Journal of Applied Engineering and Management Letters, Feb. 2024, pp. 101–09, https://doi.org/10.47992/IJAEML.2581.7000.0214.

Field Notes: Building Automated Pipeline Corrosion Monitoring with AWS IoT Core | AWS Architecture Blog. 16 Apr. 2021, https://aws.amazon.com/blogs/architecture/field-notes-building-automated-pipeline-corrosion-monitoring-with-aws-iot-core/.

Getting Started with AWS IoT Core Tutorials - AWS IoT Core. https://docs.aws.amazon.com/iot/latest/developerguide/iot-gs.html.

“GitHub - Mmaro/Aws-Iot-Projects: Projects Using AWS Services Including AWS IoT (Internet of Things), Amazon Kinesis, and so On.” GitHub, https://github.com/mmaro/aws-iot-projects.

González, Ruth, et al. “This Is How We Deployed a Data Platform Integrating AWS IoT Core and LoRaWan Networks.” Keepler | The AI Enabler Partner, https://keepler.io/2024/06/04/this-is-how-we-deployed-a-data-platform-integrating-aws-iot-core-and-lorawan-networks/.

Guide to Getting Set up with Amazon SageMaker AI - Amazon SageMaker AI. https://docs.aws.amazon.com/sagemaker/latest/dg/gs.html.

Hamid, Abdul. “How to Communicate with Arduino to Esp8266 Wifi Module via Serial Communication.” Medium, 7 Oct. 2023, https://medium.com/@abdulhamidrpn/how-to-communicate-with-arduino-to-esp8266-wifi-module-via-serial-communication-2110bc626b91.

I2C. https://docs.arduino.cc/learn/communication/wire/.

Idrees, Zeba, et al. “Edge Computing Based IoT Architecture for Low Cost Air Pollution Monitoring Systems: A Comprehensive System Analysis, Design Considerations & Development.” Sensors, vol. 18, no. 9, Sept. 2018, p. 3021. DOI.org (Crossref), https://doi.org/10.3390/s18093021.

Kumar, Kanak, et al. “IoT-HGDS: Internet of Things Integrated Machine Learning Based Hazardous Gases Detection System for Smart Kitchen.” Internet of Things, vol. 28, Dec. 2024, p. 101396. DOI.org (Crossref), https://linkinghub.elsevier.com/retrieve/pii/S2542660524003378.

Lab, Microcontrollers. “Connect ESP32 to AWS IoT MQTT and Publish Sensor Readings.” Microcontrollers Lab, 7 Aug. 2021, https://microcontrollerslab.com/aws-iot-mqtt-esp32-publish-sensor-readings/.

“Lesson 1: What Is the Arduino?” Tech Explorations, https://techexplorations.com/guides/arduino/begin/lss1.

Mukherjee, Mithun, et al. Cloud-Based Data-Intensive Framework towards Fault Diagnosis in Large-Scale Petrochemical Plants. Sept. 2016. repository.lincoln.ac.uk, https://doi.org/10.1109/iwcmc.2016.7577209’].

Open Source IOT Development Platform | IOTGecko. http://iotgecko.com/.

research!, mellisCheck out my. “Make Your Own Cellphone From Scratch.” Instructables, https://www.instructables.com/Make-your-own-cellphone-from-scratch/.

Santos, Sara. ESP8266 NodeMCU Publish Sensor Readings to ThingSpeak (Easiest Way) | Random Nerd Tutorials. 17 June 2021, https://randomnerdtutorials.com/esp8266-nodemcu-thingspeak-publish-arduino/.

Security Best Practices in AWS IoT Core - AWS IoT Core. https://docs.aws.amazon.com/iot/latest/developerguide/security-best-practices.html.

Software. https://www.arduino.cc/en/software/.

ThingSpeak for Students and Educators - ThingSpeak IoT. https://thingspeak.mathworks.com/pages/education.

"ThingSpeak Library." Https://Docs.Arduino.Cc/, https://docs.arduino.cc/libraries/thingspeak/.

"Tinkercad." Tinkercad, https://www.tinkercad.com/.

What Is Amazon SageMaker AI? - Amazon SageMaker AI. https://docs.aws.amazon.com/sagemaker/latest/dg/whatis.html.

Wokwi - World's Most Advanced ESP32 Simulator. https://wokwi.com/.

Zafra-Pérez, A., et al. "Designing a Low-Cost Wireless Sensor Network for Particulate Matter Monitoring: Implementation, Calibration, and Field-Test." Atmospheric Pollution Research, vol. 15, no. 9, Sept. 2024, p. 102208. DOI.org (Crossref), https://doi.org/10.1016/j.apr.2024.102208.

Zhifeng, Yang, et al. "Cloud Computing and Big Data for Oil and Gas Industry Application in China." Journal of Computers, vol. 1, 2019.

Bibliography

[1] "https://www.sciencedirect.com/journal/internet-of-things," [Online]. Available: https://doi.org/10.1016/j.iot.2024.101396.

[2] "https://semiconductor.samsung.com/," [Online]. Available: https://semiconductor.samsung.com/support/tools-resources/dictionary/semiconductor-glossary-mcu/#:~:text=MCU%20(Micro%20Controller%20Unit)%20An,device%20operations%20and%20specific%20systems...

[3] "https://components101.com/," [Online]. Available: https://components101.com/microcontrollers/arduino-uno .

[4] H. Patel, "Circuit Digest MQ-5," [Online]. Available: https://circuitdigest.com/microcontroller-projects/interfacing-mq5-gas-sensor-with-arduino.

[5] H. Patel, "https://circuitdigest.com/microcontroller-projects/interfacing-mq5-gas-sensor-with-arduino," [Online]. Available: https://circuitdigest.com/sites/default/files/inlineimages/u4/MQ5-Sensor-Pinout.jpg.

[6] TonesB, "ESP8266 WiFi Module for Dummies," [Online]. Available: https://www.instructables.com/ESP8266-WiFi-Module-for-Dummies/.

[7] TonesB, "https://www.instructables.com/ESP8266-WiFi-Module-for-Dummies/," Autodesk Instructables, [Online]. Available: https://content.instructables.com/FJV/FBRR/IRLQIJ1O/FJVFBRRIRLQIJ1O.jpg?auto=webp&frame=1&fit=bounds&md=MjAxNi0wOC0wOSAwMDoxMTo0NS4w.

[8] Dejan, "Arduino 16×2 LCD Tutorial," [Online]. Available: https://howtomechatronics.com/tutorials/arduino/lcd-tutorial/.

[9] Dejan, "https://howtomechatronics.com/tutorials/arduino/lcd-tutorial/," How To Mechatronics, [Online]. Available: https://howtomechatronics.com/wp-content/uploads/2015/07/LCD-Display-Tutorial.png.

[10] "Inter-Integrated Circuit (I2C) Protocol," [Online]. Available: https://docs.arduino.cc/learn/communication/wire/.

[11] "https://www.sciencedirect.com/," [Online]. Available: https://www.sciencedirect.com/topics/engineering/microcontroller#:~:text=A%20microcontroller%20is%20a%20small,converters%2C%20and%20serial%20communication%20interfaces .

[12] "Arduino IDE 2.3.6," [Online]. Available: https://www.arduino.cc/en/software/.

[13] "Wokwi Simulator," [Online]. Available: https://wokwi.com/.

[14] "Tinkercad," [Online]. Available: https://www.tinkercad.com/login .

[15] H. Electronics, "MQ-5 Gas Sensor Datasheet," Hanwei Electronics, [Online]. Available: https://www.winsen-sensor.com/d/files/MQ-5.pdf.

[16] "Select board and port in Arduino IDE," [Online]. Available: https://support.arduino.cc/hc/en-us/articles/4406856349970-Select-board-and-port-in-Arduino-IDE.

[17] "Serial.begin()," [Online]. Available: https://docs.arduino.cc/language-reference/en/functions/communication/serial/begin/.

[18] "Arduino ProjectHub," [Online]. Available: https://create.arduino.cc/projecthub.

[19] "Arduino Programming," [Online]. Available: https://docs.arduino.cc/programming/.

[20] "Hackster.io, an Avnet Community," [Online]. Available: https://www.hackster.io/.

[21] "Edge Computing Based IoT Architecture for Low Cost Air Pollution Monitoring Systems: A Comprehensive System Analysis, Design Considerations & Development," [Online]. Available: https://doi.org/10.3390/s18093021.

[22] " IBM Watson IoT Platform," [Online]. Available: https://ibm-watson-iot.github.io/iot-python/mqtt/.

[23] "Provisioning identity in AWS IoT Core for device connections," [Online]. Available: https://docs.aws.amazon.com/whitepapers/latest/device-manufacturing-provisioning/provisioning-identity-in-aws-iot-core-for-device-connections.html.

[24] "Security best practices in AWS IoT Core," [Online]. Available: https://docs.aws.amazon.com/iot/latest/developerguide/security-best-practices.html.

[25] "Server authentication," [Online]. Available: https://docs.aws.amazon.com/iot/latest/developerguide/server-authentication.html.

[26] "AWS IoT security," [Online]. Available: https://docs.aws.amazon.com/iot/latest/developerguide/iot-security.html.

[27] M. Alam, M. M. Islam, N. M. Nayan and J. Uddin, "An IoT Based Real-Time Environmental Monitoring System for Developing Areas," [Online]. Available: https://semarakilmu.com.my/journals/index.php/applied_sciences_eng_tech/article/view/4505.

[28] "https://aws.amazon.com/blogs/architecture/," [Online]. Available: https://aws.amazon.com/blogs/architecture/field-notes-building-automated-pipeline-corrosion-monitoring-with-aws-iot-core/.

[29] "https://www.sciencedirect.com/," [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0360835222003837.

[30] "Designing a low-cost wireless sensor network for particulate matter monitoring: Implementation, calibration, and field-test," [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1309104224001739?via%3Dihub.

[31] "Data Platform Integrating AWS IoT Core and LoRaWan Networks," [Online]. Available: https://keepler.io/2024/06/04/this-is-how-we-deployed-a-data-platform-integrating-aws-iot-core-and-lorawan-networks/.

[32] "Connect ESP32 to AWS IoT MQTT and Publish Sensor Readings," [Online]. Available: https://microcontrollerslab.com/aws-iot-mqtt-esp32-publish-sensor-readings/.

[33] T. Eguchi, "How to Send Data from ESP8266 to AWS IoT Core using MQTT," [Online]. Available: https://egctoru.medium.com/how-to-send-data-from-esp8266-to-aws-iot-core-using-mqtt-6dea63dc8040.

[34] "What is Amazon SNS?," [Online]. Available: https://docs.aws.amazon.com/sns/latest/dg/welcome.html.

Learning Resources

- Arduino Project Hub : https://create.arduino.cc/projecthub
- Arduno Built-in examples: https://docs.arduino.cc/built-in-examples/.
- Tinkercad Circuits : https://tinkercad.com/learn/circuits
- Random Nerd Tutorials : https://randomnerdtutorials.com
- ThingSpeak Documentation : https://thingspeak.com/docs
- ESP8266 Arduino Core : https://github.com/esp8266/Arduino
- AWS IoT Documentation : https://docs.aws.amazon.com/iot
- PubSubClient GitHub : https://github.com/knolleary/pubsubclient
- Wokwi Embedded Simulator : https://wokwi.com/
- Arduino UNO pin diagram https://components101.com/microcontrollers/arduino-uno
- Online PCB design and circuit simulator https://easyeda.com/
- https://chatgpt.com/
- https://www.grammarly.com/
- https://quillbot.com/paraphrasing-tool
- draw.io
- Lucidchart | Diagramming Powered By Intelligence
- https://www.hackster.io/
- https://www.circuito.io/blog/arduino-projects-for-beginners/

Appendices

Appendix A: Arduino Code for Prototype System

This code is the full working version for the current gas detection system, using an MQ-5 sensor connected to an Arduino Uno and ESP8266 Wi-Fi module. The system sends gas sensor values to the IOTGecko platform and triggers alerts based on user-defined thresholds for gas concentration.

```
#include "StringSplitter.h" // Library for parsing responses from IoT server
#include <SoftwareSerial.h>   // Software serial library for ESP8266 communication
#include <LiquidCrystal.h>    // Library for controlling the LCD display

// LCD pins: RS, E, D4, D5, D6, D7
LiquidCrystal lcd(7, 9, 10, 11, 12, 13);  // Initialize LCD display with pin numbers
// ESP8266 serial (RX, TX)
SoftwareSerial esp8266(3, 2);  // Initialize SoftwareSerial for ESP8266 communication

// IoT credentials - unique ID and password for authentication
const String id = "Baderfm97project@gmail.com"; // Unique ID for the IoT device
const String pass = "4441"; // Password for IoT device

// Pin definitions for different hardware components
#define GAS_SENSOR A5    // Gas sensor connected to analog pin A5
#define RED_LED 6        // Red LED for gas leakage alert, connected to pin 6
#define GREEN_LED 5      // Green LED to show normal status, connected to pin 5
#define BUZZER_PIN 8     // Buzzer pin for alarm sound when leakage is detected,
connected to pin 8
#define VALVE_PIN A4     // Valve pin to control the gas shut-off valve, connected to
pin A4
#define FAN_PIN A3       // Fan pin to activate ventilation when leakage is detected,
connected to pin A3

// Thresholds for gas levels (min and max) received from IoT server
int gas_min = 0;  // Minimum gas threshold percentage
int gas_max = 0;  // Maximum gas threshold percentage

// States for the device: 0 = connecting, 1 = fetching thresholds, 2 = monitoring gas
levels
int state = 0;

// Function prototypes (declare functions before defining them)
void beep();
void show(const char* l1, const char* l2);
void getThresholds();
void resetActuators();
void init_wifi();
```

```
void hit_link(const char* type, const String& id, const String& pass);
void hit_link2(const char* type, const String& id, const String& pass, int value);
void checkLeak(int value);

// Setup function runs once when the device starts
void setup() {
  esp8266.begin(9600);  // Start ESP8266 communication at 9600 baud rate
  Serial.begin(9600);    // Start Serial Monitor for debugging
  pinMode(GAS_SENSOR, INPUT);  // Set gas sensor pin as input
  pinMode(RED_LED, OUTPUT);     // Set red LED pin as output
  pinMode(GREEN_LED, OUTPUT);  // Set green LED pin as output
  pinMode(BUZZER_PIN, OUTPUT); // Set buzzer pin as output
  pinMode(VALVE_PIN, OUTPUT);  // Set valve pin as output
  pinMode(FAN_PIN, OUTPUT);     // Set fan pin as output

  beep();  // Play a short beep sound to indicate the system is starting
  lcd.begin(16, 2);  // Initialize the LCD display (16 columns, 2 rows)
  lcd.clear();        // Clear any previous data on the LCD
  lcd.print(F("IOT Gas Leakage"));  // Display a message on the LCD
  lcd.setCursor(0, 1);
  lcd.print(F("    Detector!"));
  delay(2000);  // Wait for 2 seconds

  esp8266.listen();  // Start listening for incoming data from ESP8266
  init_wifi();  // Initialize Wi-Fi connection
  delay(2000);  // Wait for 2 seconds before continuing
}

// Main loop function runs repeatedly after setup
void loop() {
  switch (state) {
    case 0:  // Connecting to IoT server
      hit_link("1", id, pass);  // Send connection request to IoT server
      show("Connected to", "IotGecko!");  // Display connection success on LCD
      delay(2000);  // Wait for 2 seconds
      state = 1;  // Change state to "fetching thresholds"
      break;

    case 1:  // Fetching gas threshold values from server
      getThresholds();  // Get the gas sensor thresholds (min and max values) from the
IoT server
      delay(3000);  // Wait for 3 seconds to allow the thresholds to be fetched
      state = 2;  // Change state to "monitoring gas levels"
      break;

    case 2: {  // Monitoring the gas sensor for leakage
      int raw = analogRead(GAS_SENSOR);  // Read raw value from the gas sensor
```

```
      int val = map(raw, 0, 1023, 0, 100);  // Map the raw value to a percentage (0-
100%)
      Serial.println("Gas Value: " + String(val) + "%");  // Print gas value to Serial
Monitor
      lcd.clear();  // Clear the LCD screen
      lcd.print(F("Gas Value: "));  // Display "Gas Value: " on the LCD
      lcd.setCursor(11, 0);
      lcd.print(val);  // Display the gas value percentage on the LCD
      lcd.print(F(" %"));
      resetActuators();  // Reset all actuators (valve, fan, LED, buzzer)
      digitalWrite(GREEN_LED, HIGH);  // Turn on green LED to show normal status
      checkLeak(val);  // Check if the gas value exceeds the threshold for leakage
      hit_link2("1", id, pass, val);  // Send gas value to the IoT server
      delay(1000);  // Wait for 1 second before checking again
    } break;
  }
}

// Beep function to play a short beep sound using the buzzer
void beep() {
  for (int i = 0; i < 2; i++) {  // Play 2 beeps
    digitalWrite(BUZZER_PIN, HIGH);  // Turn on the buzzer
    delay(100);  // Wait for 100ms
    digitalWrite(BUZZER_PIN, LOW);  // Turn off the buzzer
    delay(100);  // Wait for 100ms
  }
}

// Display function to show messages on the LCD screen
void show(const char* l1, const char* l2) {
  lcd.clear();  // Clear the LCD screen
  lcd.print(l1);  // Display the first line of text
  lcd.setCursor(0, 1);  // Move cursor to the second line
  lcd.print(l2);  // Display the second line of text
}

// Function to fetch gas threshold values (min and max) from the IoT server
void getThresholds() {
  hit_link("2", id, pass);  // Send request to get the gas thresholds from the server
  beep();  // Play a beep sound to indicate that the thresholds have been received
  lcd.clear();  // Clear the LCD screen
  lcd.print(F("   GOT VALUE!   "));  // Display "GOT VALUE!" message
  lcd.setCursor(0, 1);  // Move to the second line
  lcd.print("Min:" + String(gas_min) + "%");  // Display the minimum gas threshold on
LCD
  lcd.print(" Max:" + String(gas_max) + "%");  // Display the maximum gas threshold on
LCD
}
```

```
// Reset all actuators to default state (turn off buzzer, valve, fan, LEDs)
void resetActuators() {
  digitalWrite(BUZZER_PIN, LOW);  // Turn off the buzzer
  digitalWrite(VALVE_PIN, LOW);    // Close the valve
  digitalWrite(FAN_PIN, LOW);      // Turn off the fan
  digitalWrite(RED_LED, LOW);      // Turn off the red LED
  digitalWrite(GREEN_LED, LOW);    // Turn off the green LED
}

// Initialize Wi-Fi connection using AT commands for ESP8266
void init_wifi() {
  esp8266.println("AT"); delay(500);  // Check if ESP8266 is responding
  esp8266.println("AT+RST"); delay(2000);  // Reset ESP8266 module
  esp8266.println("AT+CWMODE=1"); delay(500);  // Set Wi-Fi mode to station
  esp8266.println("AT+CWJAP=\"YourSSID\",\"YourPassword\""); delay(5000);  // Connect
to Wi-Fi
  esp8266.println("AT+CIFSR"); delay(500);  // Get IP address of ESP8266
}

// Function to send requests to the IoT server (first step in communication)
void hit_link(const char* type, const String& id, const String& pass) {
  // TODO: send AT commands via ESP8266 to fetch connection status or thresholds
}

// Function to send gas sensor data to the IoT server
void hit_link2(const char* type, const String& id, const String& pass, int value) {
  // TODO: send AT commands via ESP8266 to upload the gas value
}

// Check if gas value exceeds max threshold, indicating a gas leakage
void checkLeak(int value) {
  if (value > gas_max) {  // If gas value exceeds max threshold
    lcd.clear();  // Clear the LCD screen
    lcd.print(F("Leakage detected"));  // Display "Leakage detected" on LCD
    lcd.setCursor(0, 1);  // Move to the second line
    lcd.print(F("Valve Closed!  "));  // Display "Valve Closed!" on LCD
    digitalWrite(BUZZER_PIN, HIGH);  // Turn on buzzer for alarm
    digitalWrite(VALVE_PIN, HIGH);    // Close the gas valve
    digitalWrite(FAN_PIN, HIGH);      // Turn on the fan for ventilation
    digitalWrite(RED_LED, HIGH);      // Turn on the red LED to indicate danger
    delay(3000);  // Wait for 3 seconds before resetting
  }
}
```

Appendix B: Basic Arduino + ThingSpeak Tutorial Code

This is a simple Arduino sketch for transmitting gas sensor data to ThingSpeak. It is an extension of the tutorial section and is intended for beginner learners to understand basic IoT data uploading using HTTP GET requests.

```
#include <ESP8266WiFi.h>
#include <ESP8266HTTPClient.h>

// WiFi credentials
const char* ssid     = "YourSSID";         // Replace with your network SSID
const char* password = "YourPassword";     // Replace with your network password

// ThingSpeak API configuration
const char* host     = "api.thingspeak.com"; // ThingSpeak server
const String apiKey = "YOUR_API_KEY";       // Your ThingSpeak Write API Key
const uint16_t httpPort = 80;                // HTTP port for ThingSpeak

// Timing constants
const unsigned long uploadInterval = 15000;    // Interval between sensor reads (ms)
unsigned long lastUploadTime = 0;              // Timestamp of last upload

void setup() {
  // Initialize serial for debugging
  Serial.begin(115200);
  delay(10);
  Serial.println();
  Serial.println("Booting up...");

  // Connect to WiFi network
  Serial.printf("Connecting to %s", ssid);
  WiFi.begin(ssid, password);

  // Wait for connection and provide visual feedback
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println();
  Serial.print("WiFi connected, IP address: ");
  Serial.println(WiFi.localIP());
}

void loop() {
  // Check if it's time to upload data
  unsigned long currentMillis = millis();
```

```
  if (currentMillis - lastUploadTime < uploadInterval) {
    return;  // Not time yet, exit early
  }
  lastUploadTime = currentMillis;

  // Read gas sensor value from analog pin A0
  int gasValue = analogRead(A0);
  Serial.printf("Gas sensor reading: %d\n", gasValue);

  // Ensure WiFi is still connected
  if (WiFi.status() != WL_CONNECTED) {
    Serial.println("WiFi disconnected, attempting reconnection...");
    WiFi.begin(ssid, password);
    unsigned long reconStart = millis();
    while (WiFi.status() != WL_CONNECTED && millis() - reconStart < 10000) {
      delay(500);
      Serial.print(".");
    }
    Serial.println();
    if (WiFi.status() != WL_CONNECTED) {
      Serial.println("Failed to reconnect to WiFi");
      return;  // Skip upload if connection fails
    }
  }

  // Build HTTP GET request URL
  String url = String("http://") + host + "/update?api_key=" + apiKey + "&field1=" +
gasValue;

  // Send HTTP request using HTTPClient for simplicity
  HTTPClient http;
  http.begin(url);                 // Initialize HTTP client
  int httpCode = http.GET();       // Send GET request

  // Check response code
  if (httpCode > 0) {
    Serial.printf("HTTP Response code: %d\n", httpCode);
    String payload = http.getString();  // Get response payload
    Serial.printf("Response payload: %s\n", payload.c_str());
  } else {
    Serial.printf("HTTP request failed, error: %s\n",
http.errorToString(httpCode).c_str());
  }
  http.end();  // Close connection

  // Optional: enter deep sleep to save power until next cycle
  // ESP.deepSleep(uploadInterval * 1000);  // Uncomment to enable deep sleep
}
```