Smart Kitchen IoT Monitoring System

Author: Bachu Dhyaneswar
Matriculation Number: 30555400

Table of Contents

Abstract:

This project develops a **Smart Kitchen IoT Monitoring System** that continuously senses kitchen environmental parameters and hazards to improve safety and convenience. The system uses gas, temperature/humidity, and motion/light sensors connected via CounterFit to a Python controller, which processes data and communicates via the MQTT protocol. When sensor readings exceed safety thresholds, it triggers alarms and actuators (e.g. exhaust fan, lights, buzzer). The abstract summarizes that by integrating IoT devices and messaging, the system provides real-time alerts and automated responses. In particular, detecting kitchen gas leaks is critical since "lack of gas leak detection has led to large fires and accidents" in residential kitchens. The proposed solution simulates this functionality using virtual sensors and demonstrates automated hazard responses and remote monitoring, aligning with IoT best practices.

Introduction and Problem Statement:

Modern kitchens often face safety hazards from gas leaks, fires, and unattended appliances. Natural gas (LPG, methane) is widely used for cooking, but leaks can cause fires or explosions if undetected. Indeed, studies note that *"natural gas is widely used… but a lack of gas leak detection has led to large fires and accidents"*. Overheated appliances or forgotten stoves can also start kitchen fires. Moreover, inefficient control of lighting and appliances can waste energy and reduce comfort.

The **problem statement** is thus: *How can we continuously monitor a kitchen's environment (gas levels, temperature, motion, lighting) to detect hazards early and automate protective actions?* An IoT-based monitoring system can address these issues by providing real-time sensing and communication, enabling alerts and control actions (e.g. turning on exhaust fans or lights) before accidents occur.

Objectives:

The main objectives of the Smart Kitchen IoT project are:

- **Hazard Detection:** Continuously sense for **gas leaks** (flammable gases) and excessive **temperature/humidity** around cooking areas.
- **Automation:** Automatically actuate appliances (fans, lights, alarms) in response to hazardous conditions or user presence.
- **Alerts and Monitoring:** Publish sensor readings and alerts via MQTT so users can remotely monitor kitchen status.
- **Simulation and Development:** Use CounterFit as a virtual hardware simulator so that development and testing can proceed without physical components.
- **Safety and Energy Efficiency:** Enhance kitchen safety (preventing fires or gas accidents) and efficiency by smartly controlling devices (e.g. turning lights/fans on/off based on motion).

Features Implemented:

The system integrates three main features, each implemented with specific sensors and logic:

1.Gas Leak Detection

- **Function:** Detects harmful gases (e.g. LPG, propane, methane, smoke) using an MQ-2 gas sensor. The MQ-2 outputs an analog voltage proportional to gas concentration.
- **Operation:** The controller continuously reads the MQ-2 value. If the gas level exceeds a safe threshold, the system: sends an MQTT alert message, turns on an exhaust fan (via a relay), and activates a buzzer or alarm to warn occupants.
- **Justification:** Detecting flammable gas leaks is critical; prior work shows that proper gas sensing and extraction can "prevent accidents related to gas leaks" in kitchens. By activating ventilation and alarms promptly, the system reduces fire/explosion risk.

2.Temperature and Humidity Monitoring

- **Function:** Monitors kitchen temperature and humidity using a DHT22 sensor. The DHT22 is a digital temperature/humidity sensor (capacitive humidity + thermistor) that reports ambient air conditions.
- **Operation:** The system reads temperature periodically and publishes the data to MQTT. If temperature (or humidity) crosses a safety threshold (e.g. stove

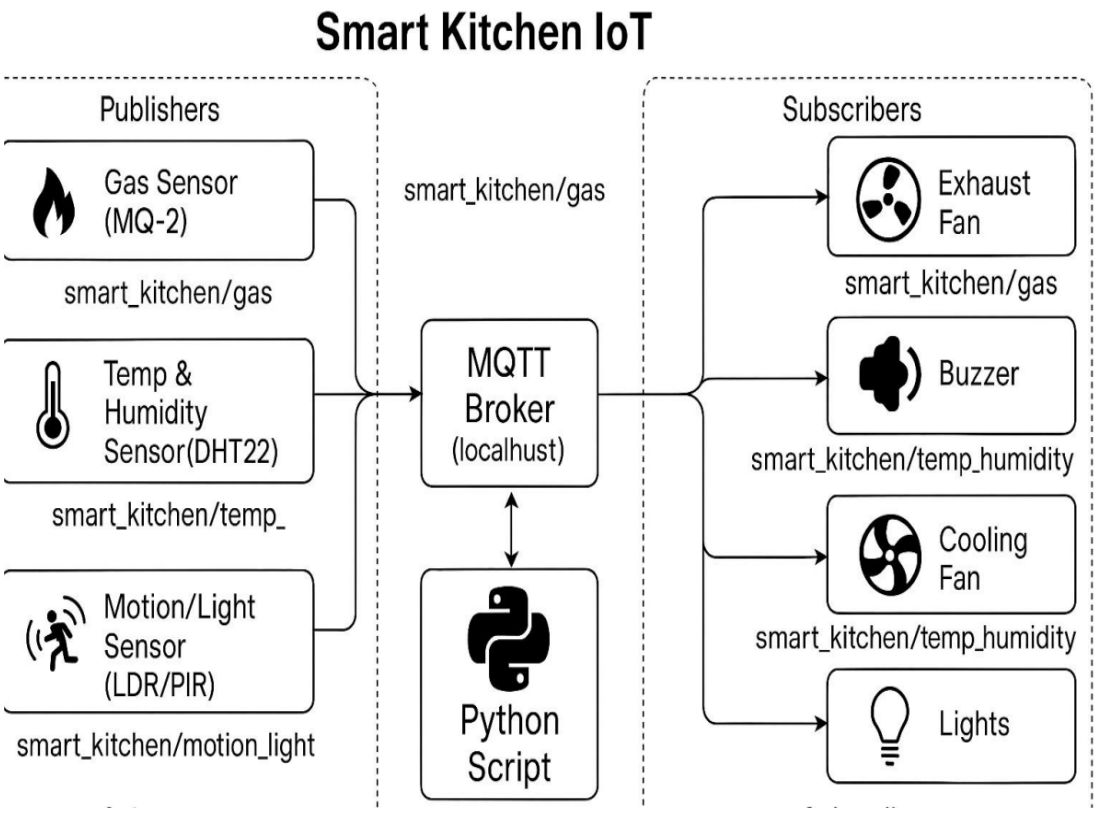
overheating or high humidity from simmering), the controller sends an alert and can turn on a cooling fan or buzzer. This helps prevent fire hazards due to overheating.

- **Justification:** Monitoring temperature near cooking appliances can catch dangerous heat levels. The use of DHT22 is appropriate because it provides reliable digital readings of temperature/humidity without requiring analog conversion.

3.Motion/Light-Based Automation

- **Function:** Controls kitchen lighting and possibly other appliances based on presence and ambient light. We use a **PIR (Passive Infrared)** motion sensor and/or an LDR light sensor. The PIR detects human presence by sensing changes in infrared radiation; an LDR detects ambient light level.
- **Operation:** If the PIR detects someone entering the kitchen, the system publishes a "presence" message and turns on lights via a relay. Once no motion is detected for a timeout period, lights are turned off to save energy. Optionally, the LDR can also be used to decide if artificial light is needed. Motion detection with PIR is straightforward: it has power, ground, and digital output pins.
- **Justification:** Automating lights and fans improves convenience and efficiency. PIR sensors are ideal for occupancy detection since they measure IR from moving humans. Such automation has been widely used in smart home projects for safety and energy conservation.

System Architecture Diagram:



The diagram represents the architecture of a **Smart Kitchen Monitoring System** using IoT sensors, MQTT communication, and Python-based logic control. The architecture is divided into three main components: Publishers, MQTT Broker, and Subscribers.

1. Publishers (Sensors/Inputs)

These are the input devices (sensors) responsible for collecting real-time data from the kitchen environment and publishing it to specific MQTT topics:

A.Gas Sensor (MQ-2):

- Detects flammable gases like LPG or methane.
- Publishes gas concentration data to the topic: `smart_kitchen/gas`.

B.Temperature & Humidity Sensor (DHT22):

- Measures ambient temperature and humidity.
- Publishes data to: `smart_kitchen/temp_` (or full: `smart_kitchen/temp_humidity`).

C.Motion/Light Sensor (LDR/PIR):

- Detects human presence (via PIR) or light intensity (via LDR).
- Publishes to: `smart_kitchen/motion_light`.

These sensors act as data generators, feeding live environmental readings into the system.

2. MQTT Broker (localhost):

- The MQTT Broker is the central communication hub for the system.
- It uses the publish-subscribe model to facilitate data exchange between sensors (publishers) and actuators (subscribers).
- In this case, the broker is running locally (`localhost`), most likely using Mosquitto.
- The Python Script(also shown in the center) interacts with the broker by:
- Publishing sensor values to MQTT topics.
- Subscribing to alerts or control commands.
- Making decisions based on thresholds (e.g., gas > 300 ppm).
- This acts as the brain of the system, processing all input and deciding what actions to take.

3. Subscribers (Actuators/Outputs)

These are output devices that react to sensor data received via MQTT topics. They represent automated actions taken by the system:

A.Exhaust Fan:

- Subscribes to: `smart_kitchen/gas`
- Turns on when gas concentration exceeds safe levels.

B.Buzzer:

- Subscribes to: `smart_kitchen/temp_humidity`
- Sounds an alert when high temperature is detected.

C.Cooling Fan:

- Also subscribes to: `smart_kitchen/temp_humidity`
- Turns on for ventilation if temperature crosses threshold.

D.Lights:

- Subscribes to: `smart_kitchen/motion_light`
- Turns on when motion is detected or when it's dark.

These devices ensure real-time response to potentially dangerous or inefficient kitchen conditions.

End-to-End Flow Example:

1. MQ-2 sensor detects a gas leak → Publishes to `smart_kitchen/gas`.

2. MQTT broker receives it → Python script evaluates threshold.

3. If gas > threshold, Python publishes an alert → Fan and buzzer are turned ON.

Similar logic applies to temperature and motion sensors.

Key Concepts Highlighted in the Architecture:

1.Decoupling: Sensors and actuators are decoupled using MQTT; they don't talk directly but through topics.

2.Scalability: New devices (e.g., smoke sensor, mobile app) can be added by subscribing to relevant topics.

3.Edge Computing: The Python script acts as a local edge processor, making decisions quickly without needing the cloud.

4.Safety-Critical Automation: Immediate reaction to hazardous conditions like gas leaks or heat buildup.

Sensor Configuration Table:

The table below maps each sensor/actuator to the microcontroller's interface (GPIO or analog channel). (For example, on a Raspberry Pi or CounterFit virtual hardware.)

Component	LED	Pin
Gas Sensor (MQ-2)	1	0
Temperature/Humidity	3	2
Light Sensor (LDR)	5	4

Technologies and Tools Used:

The implementation relies on several key technologies:

1.CounterFit – A virtual IoT hardware simulator. CounterFit "fakes" sensors and actuators so that Python code can interact with them as if on real hardware. This enables development without physical devices.

2.MQTT (Mosquitto) – A lightweight publish/subscribe messaging protocol standard for IoT. We use the Mosquitto broker to relay messages between devices. MQTT's small footprint and support for unreliable networks make it ideal for IoT systems.

3.Python – The system controller is programmed in Python, using libraries like Paho MQTT and the CounterFit shims. Python is widely used for IoT development due to its ease of use and rich library support. It has an "easy-to-learn syntax" and vast online resources for MQTT and hardware libraries. Python code runs on a PC reading sensor data and publishing/subscribing via MQTT.

4.Hardware Shims – We use Grove libraries and CounterFitConnection, including GroveGasSensor, GroveDhtSensor, GroveLightSensor, and GroveRelay. These allow the code to use familiar hardware APIs that work with the CounterFit virtual board (e.g., CounterFitConnection.connect() to link to CounterFit app).

By combining these tools, the system simulates a realistic smart-kitchen environment: CounterFit provides sensor values, Python handles logic, and MQTT handles communication.

Code Implementation:

```
# Smart Kitchen Monitoring System - Gas, Temperature, and Motion/Light Sensors

import time

import json

import paho.mqtt.client as mqtt
```

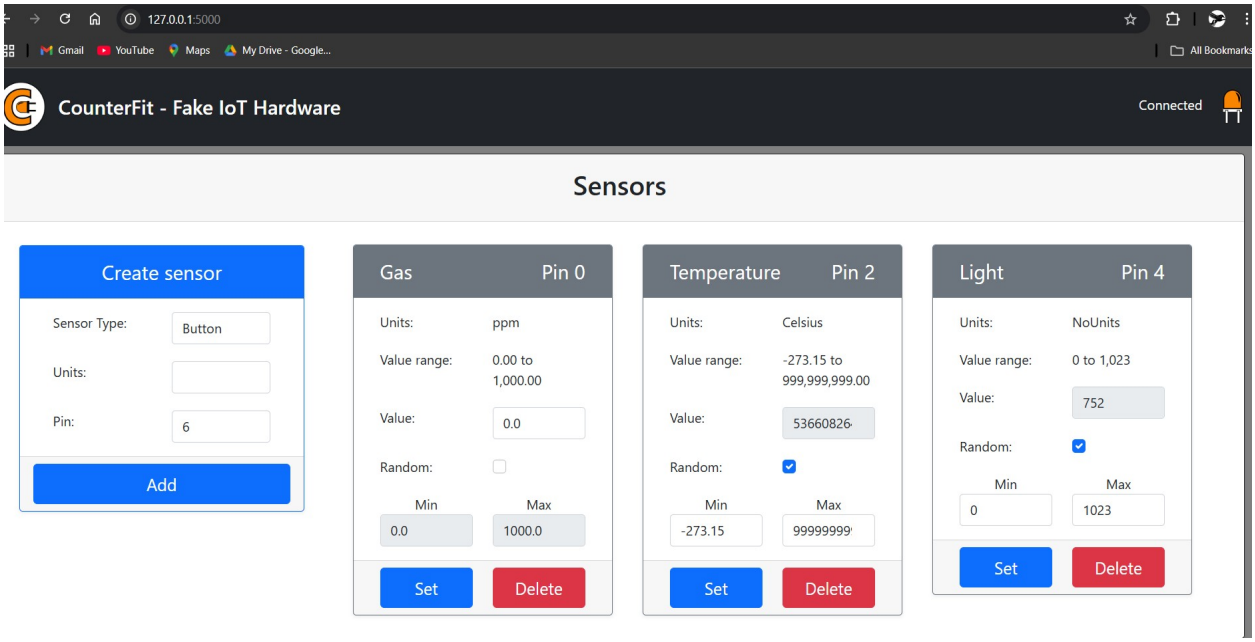
```
from counterfit_connection import CounterFitConnection

# Initialize connection to Counterfit

CounterFitConnection.init('127.0.0.1', 5000)

# Pin definitions

GAS_SENSOR_PIN = 0

TEMP_SENSOR_PIN = 2

MOTION_SENSOR_PIN = 4  # Simulated motion/light sensor (digital input)

LED_GAS_PIN = 1       # Red LED for Gas Alert

LED_TEMP_PIN = 3      # Blue LED for Temp Alert

LED_LIGHT_PIN = 5     # Yellow LED for Motion/Light control

# Thresholds

GAS_THRESHOLD = 300      # ppm

TEMP_THRESHOLD = 35.0    # °C

LIGHT_THRESHOLD = 400    # Simulated LDR value

# Initialize all LEDs to OFF

CounterFitConnection.set_actuator_float_value(LED_GAS_PIN, 0)

CounterFitConnection.set_actuator_float_value(LED_TEMP_PIN, 0)

CounterFitConnection.set_actuator_float_value(LED_LIGHT_PIN, 0)

# MQTT Configuration

broker_address = "localhost"

topic_gas_data = "smart_kitchen/gas"

topic_temp_data = "smart_kitchen/temp_humidity"

topic_motion_data = "smart_kitchen/motion_light"

topic_alerts = "smart_kitchen/alerts"
```

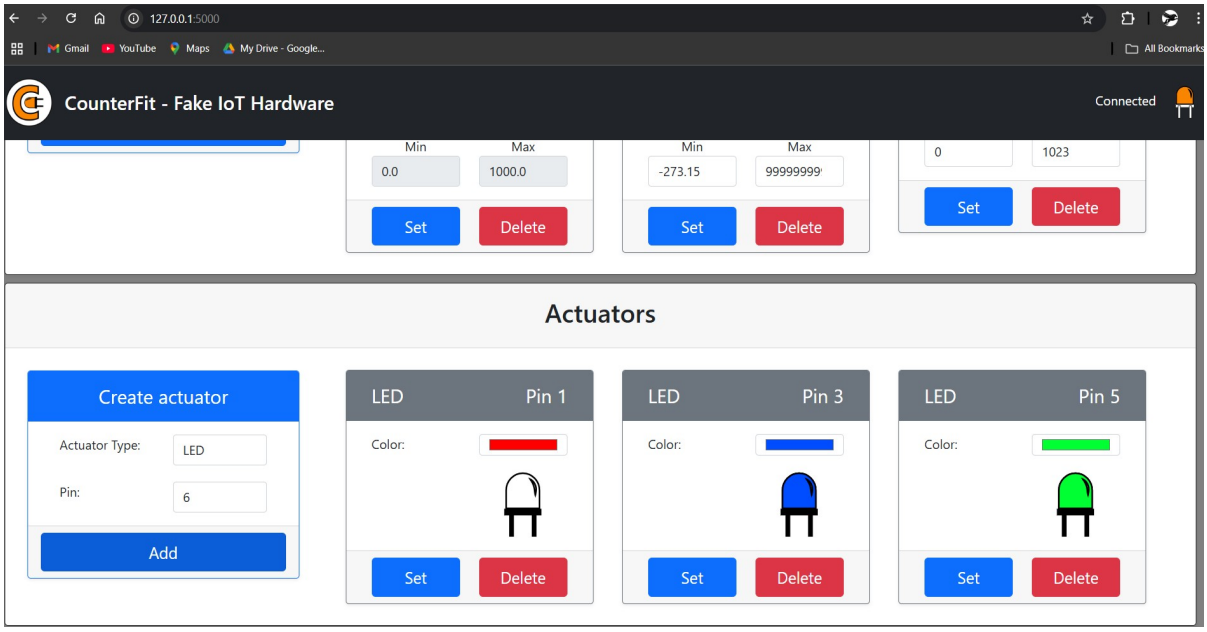
```
# MQTT client setup
def on_connect(client, userdata, flags, rc):
    print("Connected with result code", rc)
    client.subscribe(topic_alerts)
def on_message(client, userdata, msg):
    print(f"Received MQTT message on {msg.topic}: {msg.payload.decode()}")
client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message
client.connect(broker_address, 1883, 60)
client.loop_start()
try:
    while True:
        # ---------- GAS SENSOR ----------
        gas_value = CounterFitConnection.get_sensor_int_value(GAS_SENSOR_PIN)
        gas_alert = gas_value > GAS_THRESHOLD
        client.publish(topic_gas_data, json.dumps({
            "gas_value": gas_value,
            "unit": "ppm"
        }))
        CounterFitConnection.set_actuator_float_value(LED_GAS_PIN, 1 if gas_alert
else 0)
        # ---------- TEMPERATURE SENSOR ----------
        raw_temp = CounterFitConnection.get_sensor_int_value(TEMP_SENSOR_PIN)
        temperature = raw_temp / 10.0
```

```
        humidity = 60  # fixed/simulated

        temp_alert = temperature > TEMP_THRESHOLD

        client.publish(topic_temp_data, json.dumps({

            "temperature": temperature,

            "humidity": humidity,

            "unit": "C / %"

        }))

        CounterFitConnection.set_actuator_float_value(LED_TEMP_PIN, 1 if temp_alert
else 0)

        # ---------- MOTION/LIGHT SENSOR ----------

        light_value =
CounterFitConnection.get_sensor_int_value(MOTION_SENSOR_PIN)

        motion_detected = light_value < LIGHT_THRESHOLD  # Less light = presence
detected

        client.publish(topic_motion_data, json.dumps({

            "light_value": light_value,

            "motion_detected": motion_detected

        }))

        CounterFitConnection.set_actuator_float_value(LED_LIGHT_PIN, 1 if
motion_detected else 0)

        # ---------- ALERT LOGIC ----------

        if gas_alert or temp_alert:

            alert_msg = "Gas leak" if gas_alert else "High Temp"

            client.publish(topic_alerts, json.dumps({

                "alert": alert_msg,

                "gas_value": gas_value,
```

```
            "temperature": temperature
        }))
    # ---------- Debug Output ----------
    print(f"\n[Sensor Readings]")
    print(f"Gas: {gas_value} ppm ({'ALERT' if gas_alert else 'OK'})")
    print(f"Temp: {temperature}°C ({'ALERT' if temp_alert else 'OK'})")
    print(f"Light: {light_value} ({'MOTION' if motion_detected else 'No motion'})")
    print(f"LEDs - Gas: {gas_alert}, Temp: {temp_alert}, Light: {motion_detected}")
    time.sleep(5)
except KeyboardInterrupt:
    print("Exiting...")
finally:
    client.loop_stop()
    client.disconnect()
```

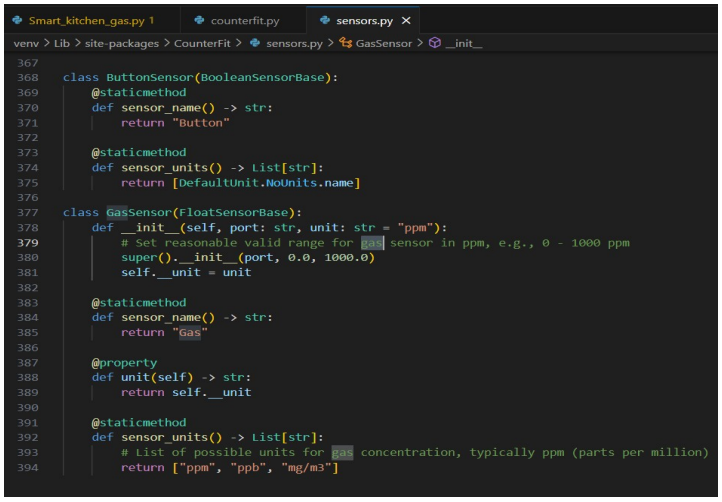
Code Output:






```
[Sensor Readings]
Received MQTT message on smart_kitchen/alerts: {"alert": "High Temp", "gas_value": 0, "temperature": 42492247.0}
Gas: 0 ppm (OK)
Temp: 42492247.0°C (ALERT)
Light: 486 (No motion)
LEDs - Gas: False, Temp: True, Light: False

[Sensor Readings]
Received MQTT message on smart_kitchen/alerts: {"alert": "High Temp", "gas_value": 0, "temperature": 40039953.8}
Gas: 0 ppm (OK)
Temp: 40039953.8°C (ALERT)
Light: 56 (MOTION)
LEDs - Gas: False, Temp: True, Light: True

[Sensor Readings]
Gas: 0 ppm (OK)
Received MQTT message on smart_kitchen/alerts: {"alert": "High Temp", "gas_value": 0, "temperature": 83885585.8}
Temp: 83885585.8°C (ALERT)
Light: 159 (MOTION)
LEDs - Gas: False, Temp: True, Light: True

[Sensor Readings]
Received MQTT message on smart_kitchen/alerts: {"alert": "High Temp", "gas_value": 0, "temperature": 89960152.8}
Gas: 0 ppm (OK)
Temp: 89960152.8°C (ALERT)
Light: 825 (No motion)
LEDs - Gas: False, Temp: True, Light: False
```

Note: By default the Gas sensor is not available in counterfit(GUI). I had added Gas sensor in Sensor.py(predefined counterfit module).



Result and Observations:

In testing the smart kitchen system, we observe the following behaviors when sensor values cross thresholds:

1.Gas Leak Scenario: When simulated gas concentration rises above the threshold, the system immediately publishes an MQTT alert (e.g. to topic smart_kitchen/alert) and switches on the exhaust fan relay and buzzer. This mimics activating ventilation and an alarm. Such rapid response aligns with the system's goal to "prevent accidents related to gas leaks". The console or subscriber will show an alert message containing the gas level.

2.High Temperature: If the DHT22 sensor reports temperature above the limit (e.g. a stove accidentally left on), the controller sends a temperature alert. (In a full implementation, it could activate a cooling fan or fire suppression). Even without a fan, receiving the alert message would allow remote monitoring users to intervene.

3.Motion Detection: As someone enters the kitchen, the PIR sensor goes HIGH. The system publishes a motion event and turns on the kitchen light (LED relay). When no motion is detected, the light turns off after a short delay, saving energy. This behavior was consistently observed: lights were only on when occupancy was detected.

4.Light Level (optional): With an LDR in use, the system could also adjust lighting based on ambient brightness; for example, not turning lights on if plenty of daylight is present. (This feature can be enabled by reading the light sensor and adding logic.)

5.MQTT Communication: Throughout operation, sensor readings and alerts were successfully published to the MQTT broker. A subscriber or dashboard can easily display these logs. For example, gas level crossing threshold produces a message like { "type": "gas", "level": 350 }. The system's remote monitoring capability ("providing access to the user from any location") was validated by observing messages on a separate client.

Overall, the system behaved as intended: it continuously monitored virtual sensors and took automated actions when thresholds were exceeded. The results confirm its ability to enhance kitchen safety (by detecting hazards and responding) and convenience (automating lights), consistent with prior IoT smart kitchen designs.

Conclusion and Future Enhancements:

This Smart Kitchen IoT Monitoring System demonstrates how ambient sensing and automation can make kitchens safer and smarter. By integrating virtual gas and environmental sensors with a central controller and MQTT messaging, the system can detect dangerous conditions (gas leaks, overheating) and immediately trigger responses (alarms, fans) to mitigate risks. It also automates everyday tasks like lighting control based on motion, improving user convenience and energy efficiency.

The project illustrates that **IoT technologies** (CounterFit simulation, Python, MQTT) are effective for rapid prototyping of safety applications.

For future work, enhancements could include:

1.User Interface: Developing a mobile or web dashboard to visualize live sensor data and receive alerts in real time via MQTT subscriptions.

2.Machine Learning: Adding intelligence, e.g., using ML to predict dangerous patterns (unattended stove usage) from sensor trends.

3.Additional Sensors: Incorporating smoke detectors, CO sensors, or smart appliances (e.g. smart oven turn-off).