太棒了！下面给你一份\*\*针对宿舍场景\*\*、只用\*\*ESP32 + 火焰传感器 + MQ-2 + 0.96 OLED + 小风扇 + Flask Web\*\*的\*\*极简火灾报警系统\*\*的完整步骤与指南（含可直接使用的示例代码）。

---

# 1. 物料与连接

## 1.1 物料清单（BOM）

\* ESP32 开发板（带 USB）

\* 扩展板/面包板、杜邦线、排针排母

\* 火焰传感器模块（模拟量输出 AO）

\* MQ-2 烟雾传感器模块（模拟量输出 AO）

\* 0.96" OLED（I2C，SSD1306 128×64）

\* 130 小风扇（5V/3.3V，推荐配\*\*NPN 三极管 + 二极管\*\*或小继电器）

\* 3.3 V 稳压模块（若传感器需 3.3V 供电）

\* USB 线、电源

> MQ-2 预热 24–48 小时更稳定；如采用 5V 供电的 MQ-2，请\*\*分压\*\*或\*\*运放\*\*把 AO 调到 ESP32 可承受范围（0–3.3V）。

## 1.2 引脚与接线（推荐）

| 模块 | 引脚 | ESP32 引脚 | 说明 |

| ---------- | ------- | ------------------ | --------------- |

| 火焰传感器 | AO | GPIO34 (ADC1\\_CH6) | 仅输入 |

| MQ-2 | AO | GPIO35 (ADC1\\_CH7) | 仅输入；注意电平 |

| OLED (I2C) | SDA/SCL | GPIO21 / GPIO22 | 3.3V 供电 |

| 风扇驱动 | IN | GPIO25 | 用 NPN/继电器，别直连风扇 |

| 供电 | VCC/GND | 3.3V/ GND | 传感器/ESP32 共地 |

\*\*风扇驱动参考\*\*：

ESP32 GPIO25 → 电阻 1k → NPN(B)；NPN(C) 接风扇负极，风扇正极接 5V；NPN(E) 接 GND；风扇两端并联\*\*续流二极管\*\*（1N4148/1N5819）。

---

# 2. 系统功能与阈值策略（宿舍极简）

\* \*\*触发条件（任一满足即报警）\*\*

\* 火焰模拟值低于阈值（许多火焰模块输出“有火→电压低”）：`flame\_raw < FLAME\_THR`

\* MQ-2 模拟值高于阈值：`smoke\_raw > MQ2\_THR`

\* \*\*动作\*\*

\* OLED 红色页/反白：“🔥 火灾报警”

\* 接蜂鸣器报警

\* 以 JSON 通过 HTTP POST 上传 Flask：`/api/data`

\* \*\*去抖\*\*：同一状态持续 N 秒只报一次；恢复正常后再允许下一次触发

\* \*\*校准\*\*：首次部署记录 1–2 天“正常环境”平均值，动态微调阈值

\*\*起始阈值建议（需现场实测再微调）\*\*

\* `FLAME\_THR = 1200`（0–4095 之间，数值越小越可能有火）

\* `MQ2\_THR = 1200`（0–4095 之间，数值越大烟雾越浓）

---

# 3. ESP32 端示例（Thonny）

#3. 1. 准备与刷写

1. \*\*烧录 MicroPython 固件\*\*

\* 到 micropython.org 下载 ESP32 固件 (`.bin`)。

\* 用 `esptool.py` 或 Thonny/MPy-Tool 烧录：

```bash

esptool.py --chip esp32 --port COM5 erase\_flash

esptool.py --chip esp32 --port COM5 --baud 460800 write\_flash -z 0x1000 esp32-<version>.bin

```

3.2. \*\*依赖文件\*\*（复制到开发板 / 根目录）

\* `main.py`（主程序，下面给出）

\* `secrets.py`（Wi-Fi 与服务器配置）

\* `urequests.py`（若固件不带就上传一份，常见版本可在 micropython-lib 找到）

\* `ssd1306.py`（有些固件自带；若没有，上传官方驱动）

---

# 1. 硬件连线（与前文一致）

| 模块 | 引脚 | ESP32 引脚 | 说明 |

| ---------- | ------- | ------------------ | -------------------- |

| 火焰传感器 | AO | GPIO34 (ADC1\\_CH6) | 仅输入 |

| MQ-2 | AO | GPIO35 (ADC1\\_CH7) | 仅输入（⚠️确保≤3.3V，必要时分压） |

| OLED (I2C) | SDA/SCL | GPIO21 / GPIO22 | 3.3V 供电，地址 0x3C |

| 风扇驱动 | IN | GPIO25 | 用 NPN/继电器，风扇两端并联二极管 |

| 供电 | VCC/GND | 3.3V / GND | 共地 |

> \*\*安全提醒\*\*：多数 MQ-2 模块输出随供电电压变化。若用 5V 供电，AO 也会接近 5V，\*\*必须分压到 ≤3.3V\*\* 再接 ESP32 ADC。

---

# 2. `secrets.py`（配置）

```python

# secrets.py

WIFI\_SSID = "your\_wifi"

WIFI\_PSK = "your\_pass"

# 你的 Flask 服务地址

SERVER\_URL = "http://<你的服务器或域名>:5000/api/data"

# 设备标识

DEVICE\_ID = "ESP32-001"

```

---

# 3. `main.py`（MicroPython 主程序）

```python

# main.py -- MicroPython on ESP32

import time, sys, json

from machine import ADC, Pin, I2C, Timer, reset

import network

# ==== 配置 ====

from secrets import WIFI\_SSID, WIFI\_PSK, SERVER\_URL, DEVICE\_ID

# 引脚映射

PIN\_FLAME = 34 # 火焰 AO -> ADC1\_CH6

PIN\_MQ2 = 35 # MQ-2 AO -> ADC1\_CH7

PIN\_FAN = 25 # 风扇驱动 -> GPIO25 (NPN/继电器)

# 阈值（先用默认，部署后再调）

FLAME\_THR = 1200 # 火焰：值越小越可能有火（部分模块相反，实测后调整判定符号）

MQ2\_THR = 1200 # 烟雾：值越大越浓

SAMPLE\_MS = 500 # 采样周期

POST\_COOLDOWN\_MS = 5000 # 上报去抖

WIFI\_RETRY\_S = 5

# ==== OLED 驱动 ====

# 若固件无 ssd1306 模块，请上传 ssd1306.py

try:

from ssd1306 import SSD1306\_I2C

HAVE\_OLED = True

except Exception as e:

HAVE\_OLED = False

i2c = I2C(0, scl=Pin(22), sda=Pin(21), freq=400000)

oled = None

if HAVE\_OLED:

try:

oled = SSD1306\_I2C(128, 64, i2c, addr=0x3C)

except:

oled = None

def oled\_show(lines, alarm=False):

if not oled:

return

oled.fill(0)

y = 0

for s in lines:

oled.text(s, 0, y)

y += 10

if alarm:

oled.fill\_rect(0, 48, 128, 16, 1)

oled.fill\_rect(2, 50, 124, 12, 0) # 边框反白

oled.text("FIRE ALARM!", 8, 52, 1)

oled.show()

# ==== Wi-Fi ====

def wifi\_connect():

sta = network.WLAN(network.STA\_IF)

if not sta.active():

sta.active(True)

if not sta.isconnected():

sta.connect(WIFI\_SSID, WIFI\_PSK)

t0 = time.ticks\_ms()

while not sta.isconnected() and time.ticks\_diff(time.ticks\_ms(), t0) < 15000:

oled\_show(["WiFi connecting...", WIFI\_SSID])

time.sleep\_ms(300)

return sta

# ==== HTTP 上报 ====

def post\_json(url, data):

# 某些固件自带 urequests；没有就上传 urequests.py

import urequests as requests

try:

r = requests.post(url, data=json.dumps(data), headers={"Content-Type":"application/json"})

r.close()

return True

except Exception as e:

return False

# ==== 传感器 & 风扇 ====

adc\_flame = ADC(Pin(PIN\_FLAME, Pin.IN))

adc\_flame.atten(ADC.ATTN\_11DB) # 0~3.3V 区间

adc\_flame.width(ADC.WIDTH\_12BIT) # 0~4095

adc\_mq2 = ADC(Pin(PIN\_MQ2, Pin.IN))

adc\_mq2.atten(ADC.ATTN\_11DB)

adc\_mq2.width(ADC.WIDTH\_12BIT)

fan = Pin(PIN\_FAN, Pin.OUT)

fan.value(0)

# 状态

alarm\_on = False

last\_post\_ms = 0

def read\_avg(adc, n=5, delay\_ms=5):

s = 0

for \_ in range(n):

s += adc.read()

time.sleep\_ms(delay\_ms)

return s // n

def loop():

global alarm\_on, last\_post\_ms

sta = wifi\_connect()

if not sta.isconnected():

oled\_show(["WiFi FAIL", "reboot in 5s"])

time.sleep(5)

reset()

while True:

flame = read\_avg(adc\_flame)

smoke = read\_avg(adc\_mq2)

# 如果你的火焰模块输出“有火=高电平”，把下面判定改为 flame > FLAME\_THR

alert = (flame < FLAME\_THR) or (smoke > MQ2\_THR)

# 控制风扇

if alert and not alarm\_on:

alarm\_on = True

fan.value(1)

elif not alert and alarm\_on:

alarm\_on = False

fan.value(0)

# OLED

lines = [

"Dorm Fire Monitor",

"Flm: %4d Thr:%d" % (flame, FLAME\_THR),

"MQ2: %4d Thr:%d" % (smoke, MQ2\_THR),

"WiFi: %s" % ("OK" if network.WLAN(network.STA\_IF).isconnected() else "X")

]

oled\_show(lines, alarm=alert)

# 上报（去抖）

now = time.ticks\_ms()

if time.ticks\_diff(now, last\_post\_ms) >= POST\_COOLDOWN\_MS:

payload = {

"device\_id": DEVICE\_ID,

"flame": flame,

"smoke": smoke,

"alert": alert

}

ok = post\_json(SERVER\_URL, payload)

last\_post\_ms = now

time.sleep\_ms(SAMPLE\_MS)

# ==== 入口 ====

try:

loop()

except Exception as e:

# 简单兜底：异常后重启

try:

oled\_show(["EXCEPTION", str(e)[:16], "Reboot 3s..."])

time.sleep(3)

except:

pass

reset()

```

---

# 4. 现场调校与建议

1. \*\*基线采样\*\*：上电后 10–30 分钟在“正常宿舍环境”记录 flame/mq2 均值，按经验设：

\* `FLAME\_THR` ≈ 正常 flame 基线 × 0.7（火焰越接近越小）

\* `MQ2\_THR` ≈ 正常 mq2 基线 × 1.5–2.0

2. \*\*预热\*\*：MQ-2 需要预热，初期波动大；阈值请在预热稳定后调整。

3. \*\*误报抑制\*\*：若误报多，可把 `POST\_COOLDOWN\_MS` 提高，并在 `alert` 判定前加\*\*连续 N 次满足\*\*再报警（例如在循环里维护计数）。

4. \*\*供电/保护\*\*：风扇等感性负载务必三极管 + 续流二极管；MQ-2 若 5V 供电，一定分压 AO。

5. \*\*掉线重连\*\*：代码已在失败时重启；也可改成更温和的 `sta.disconnect(); sta.connect()`。

---

# 5. 与 Flask 后端对接

\* Flask 端保持之前给你的 `POST /api/data` 接口即可，接收 JSON：

```json

{"device\_id":"ESP32-001","flame":1032,"smoke":1450,"alert":true}

```

\* 前端页面每 1.5s 拉取 `/api/data/recent?limit=20` 展示最新状态（已给出模板）。

---

# 6. 可选增强（MicroPython 侧）

\* \*\*本地蜂鸣器\*\*（GPIO26）报警声；

\* \*\*动态阈值\*\*：滑动均值 + 偏移；

\* \*\*MQTT 模式\*\*：将 `post\_json` 改为 MQTT 发布，更实时；

\* \*\*低功耗\*\*：空闲时可降低采样频率，或按键进入 light sleep。

---

如果你需要，我可以把这套文件按\*\*可直接拷贝的项目结构\*\*列出来（含 `urequests.py / ssd1306.py` 常用版本）并附上\*\*阈值连续触发去抖\*\*的变体代码。

> 若你的火焰模块“有火输出高电平”，将条件改为 `flame > FLAME\_THR` 即可。

---

# 4. Flask 端最小可跑（含 API + 简单网页）

> 依赖：`Flask`, `Flask-Cors`, `sqlite3`（标准库）, `jinja2`（随 Flask）。

> 建议：Nginx 反代 + HTTPS；生产用 `gunicorn`.

\*\*requirements.txt\*\*

```

Flask==3.0.0

Flask-Cors==4.0.0

```

\*\*app.py\*\*

```python

from flask import Flask, request, jsonify, render\_template

from flask\_cors import CORS

import sqlite3, time, os

DB = "fire.db"

app = Flask(\_\_name\_\_)

CORS(app)

def init\_db():

with sqlite3.connect(DB) as conn:

conn.execute("""CREATE TABLE IF NOT EXISTS fire\_data(

id INTEGER PRIMARY KEY AUTOINCREMENT,

device\_id TEXT,

flame INTEGER,

smoke INTEGER,

alert INTEGER,

ts INTEGER

)""")

init\_db()

@app.post("/api/data")

def ingest():

data = request.get\_json(force=True, silent=True) or {}

device\_id = data.get("device\_id", "ESP32-001")

flame = int(data.get("flame", 0))

smoke = int(data.get("smoke", 0))

alert = 1 if data.get("alert") else 0

ts = int(time.time()\*1000)

with sqlite3.connect(DB) as conn:

conn.execute("INSERT INTO fire\_data(device\_id,flame,smoke,alert,ts) VALUES(?,?,?,?,?)",

(device\_id, flame, smoke, alert, ts))

conn.commit()

return {"ok": True}

@app.get("/api/data/recent")

def recent():

limit = int(request.args.get("limit", 50))

with sqlite3.connect(DB) as conn:

cur = conn.execute("SELECT device\_id, flame, smoke, alert, ts FROM fire\_data ORDER BY id DESC LIMIT ?", (limit,))

rows = [dict(device\_id=r[0], flame=r[1], smoke=r[2], alert=bool(r[3]), ts=r[4]) for r in cur.fetchall()]

return jsonify(rows)

@app.get("/")

def dashboard():

return render\_template("index.html")

if \_\_name\_\_ == "\_\_main\_\_":

app.run("0.0.0.0", 5000, debug=True)

```

\*\*templates/index.html（极简仪表盘）\*\*

```html

<!doctype html>

<html lang="zh">

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width,initial-scale=1" />

<title>宿舍火灾报警系统</title>

<style>

body{font-family:system-ui,Segoe UI,Roboto,Helvetica,Arial,sans-serif;margin:0;padding:24px;background:#f7f7f9;color:#111}

.grid{display:grid;grid-template-columns:repeat(3,minmax(0,1fr));gap:16px}

.card{background:#fff;border-radius:16px;padding:16px;box-shadow:0 1px 10px rgba(0,0,0,.06)}

.danger{background:#ffe6e6;border:1px solid #ff8080}

h1{margin:0 0 16px;font-size:24px}

.muted{color:#666;font-size:12px}

table{width:100%;border-collapse:collapse}

th,td{padding:8px;border-bottom:1px solid #eee;text-align:left;font-size:14px}

.tag{display:inline-block;padding:2px 8px;border-radius:999px;font-size:12px}

.ok{background:#e6f7ee;color:#0a7d38}

.alert{background:#ffe8e8;color:#b00020}

</style>

</head>

<body>

<h1>宿舍火灾报警系统</h1>

<div id="alertBox" class="card">状态：<span id="status" class="tag ok">正常</span></div>

<div class="grid">

<div class="card"><b>火焰模拟值</b><div id="flame" style="font-size:28px;margin-top:8px">--</div><div class="muted">值越低越可能有明火</div></div>

<div class="card"><b>MQ-2 烟雾值</b><div id="smoke" style="font-size:28px;margin-top:8px">--</div><div class="muted">值越高烟雾越浓</div></div>

<div class="card"><b>最近时间</b><div id="time" style="font-size:28px;margin-top:8px">--</div><div class="muted">最新一条数据时间</div></div>

</div>

<div class="card" style="margin-top:16px">

<b>最近记录</b>

<table id="tbl"><thead><tr><th>时间</th><th>火焰</th><th>烟雾</th><th>告警</th></tr></thead><tbody></tbody></table>

</div>

<script>

async function fetchData(){

const res = await fetch('/api/data/recent?limit=20');

const rows = await res.json();

if(rows.length){

const r = rows[0];

document.getElementById('flame').textContent = r.flame;

document.getElementById('smoke').textContent = r.smoke;

document.getElementById('time').textContent = new Date(r.ts).toLocaleString();

const status = document.getElementById('status');

const alertBox = document.getElementById('alertBox');

if (r.alert){

status.textContent = '火灾报警';

status.className = 'tag alert';

alertBox.className = 'card danger';

}else{

status.textContent = '正常';

status.className = 'tag ok';

alertBox.className = 'card';

}

const tbody = document.querySelector('#tbl tbody');

tbody.innerHTML = rows.map(x=>`

<tr>

<td>${new Date(x.ts).toLocaleString()}</td>

<td>${x.flame}</td>

<td>${x.smoke}</td>

<td>${x.alert? '是' : '否'}</td>

</tr>

`).join('');

}

}

setInterval(fetchData, 1500);

fetchData();

</script>

</body>

</html>

```

\*\*运行\*\*

```bash

pip install -r requirements.txt

python app.py

# 浏览器打开 http://服务器IP:5000

```

---

# 5. 联调与验证

1. \*\*连接 Wi-Fi\*\*：ESP32 串口监视器确认联网成功。

2. \*\*静态环境\*\*：记录 5–10 分钟火焰/烟雾基线值，微调 `FLAME\_THR`、`MQ2\_THR`。

3. \*\*触发测试\*\*：

\* 打火机火焰靠近传感器（注意安全）

\* 烟雾（点香/雾化烟雾）靠近 MQ-2

\* 观察：OLED 告警 + 风扇启动 + Web 端变红 + 表格新增记录

4. \*\*误报控制\*\*：若夜间误报多，适当提高 MQ-2 阈值、延长去抖时间（比如 3–5s 连续满足再报警）。

---

# 6. 部署与安全

\* \*\*Web 端\*\*：Nginx 反向代理 Flask、签 TLS（Let’s Encrypt），限制外网只开放 80/443/5000（视需要）。

\* \*\*内网优先\*\*：宿舍路由器内网部署较安全；外网访问需鉴权。

\* \*\*传感器供电\*\*：确保 MQ-2 模块 AO 不超过 3.3V；必要时分压。

\* \*\*风扇/负载\*\*：务必使用三极管/继电器隔离，串二极管防反灌。

\* \*\*日志与维护\*\*：SQLite 文件每日/每周归档；超过 30 天自动清理。

---

# 7. 可选增强（保持极简前提下）

\* 加入 \*\*蜂鸣器\*\*（GPIO26）本地声光报警

\* \*\*短信/微信机器人\*\*：Flask 中在 `alert==true` 时调用通知接口

\* \*\*自学习阈值\*\*：用最近一段时间的滑动均值±偏移做动态阈值

\* \*\*MQTT 备选\*\*：后端改为订阅 MQTT，再写库（便于多设备扩展）

---

需要我把上述代码打包成\*\*可运行项目结构\*\*（含 `static/`、`templates/`、`systemd`/`gunicorn` 启动脚本、Nginx 样例）吗？我可以直接给你完整目录和文件内容，拷贝即用。