基于 STM32G474R 的边缘 AI 风机振动异常监测系统

摘要

本项目基于 STM32G474 构建了一套本地边缘 AI 风机振动监测系统，实现了离线状态下的异常识别与风机远程控制，配合 ADXL335 三轴加速度传感器实现高频振动数据采集，通过 DMA 与定时器构建高效数据通道，采样频率为 1kHz。采集的数据通过 NanoEdge AI Studio 训练生成的模型在本地推理，输出当前状态相似度。系统无需联网即可在边缘完成异常识别，具备良好的实时性与数据隐私保护能力。

用户可通过 OLED 显示屏查看设备运行状态与相似度指标，也可使用手机 APP 实现风机启停与状态反馈。系统具备自动异常恢复机制，采用 IWDG（硬件独立看门狗）监控运行状态，确保长期部署的稳定性。机械结构采用激光切割亚克力板构建，内部模块布局紧凑，便于后期维护。

本系统已在实验环境中完成搭建与测试，可广泛应用于工业设备振动监测、教学演示、旋转机械健康评估等场景，具备部署成本低、开发路径清晰、技术复用性强等优势。

第一部分　作品概述

1.1 功能与特性

本系统主要功能包括：高频三轴振动数据采集、本地异常检测与分级状态报警，以及远程控制与反馈。系统通过 STM32 定时器触发 ADC + DMA 实现高频同步采样，将数据无损存储到缓存区等待 NanoEdge AI 模型进行异常推理。检测结果以相似度形式表示，并结合设定阈值进行正常、警告、异常三种状态的判定。

在交互方面，用户可通过 0.96 寸 OLED 实时查看状态信息，或通过手机 APP 实现启停操作，系统整体运行模式可灵活切换。系统采用独立看门狗策略确保在意外中断或卡死时自动复位恢复，满足对鲁棒性与长期运行的高要求。

1.2 应用领域

本系统适用于工业设备（如鼓风机、水泵、电机等）的预测性维护和异常监测，尤其适合边缘部署场景，如工厂车间或电机控制柜中。无需依赖网络，便可实时识别设备运行状态，对于没有 IT 网络基础的老旧工厂尤为适用。

此外，该系统可作为高校电子工程类课程中的教学实验平台，覆盖传感器接口、电机控制、边缘 AI 推理、嵌入式通信等多项核心知识点，具备良好的教学拓展性。系统结构简单，维护成本低，适合批量部署在高校实验室或初创工程试验平台中。

1.3 主要技术特点

系统核心采用 STM32G474RE 微控制器，结合 ADC+DMA+TIMER 完成三轴振动数据采集，并通过 NanoEdge AI 模型在本地完成推理判断，完全脱离网络依赖，具备边缘智能能力。通信部分使用 ESP32，运行 MicroPython 搭建的简易 Web 服务器，手机可直接通过 APP 访问控制风机启停与系统状态。

系统控制协议采用字符型指令格式，结构简洁，扩展性好。整机启用独立看门狗，用于监控 MCU 是否死机，增强系统鲁棒性。显示端采用 OLED 模块输出状态、相似度等信息，提升交互体验。

机械部分采用亚克力激光切割板件配合铜柱固定，布局整洁，易于展示与维护。

1.4 主要性能指标

表 1 主要性能指标

名称	参数值	说明
采样频率	1Hz	三轴同时采样
AI 检测延迟	500ms	从采样到输出
OLED 刷新率	1FPS	视觉刷新频率
控制响应延迟	<100ms	本地串口响应速度
无线通信范围	25 米	开阔无障碍环境下

1.5 主要创新点

（1）NanoEdge AI 模型部署在 STM32 本地，完成无联网状态下的异常检测推理，减少系统延迟与资源开销。

（2）自研安卓 APP 替代网页平台，通过 WiFi 与 ESP32 建立通信，实现移动端控制与状态反馈。

（3）采用字符型指令通信协议，协议结构简单、扩展方便，降低调试成本。

1.6 设计流程

ADXL335 输出模拟振动信号，经 STM32 的 ADC 在定时器触发下采样，DMA 自动搬运数据，主循环判断采样窗口是否填满。填满后调用 NanoEdge AI 模型进行本地推理，输出相似度与状态等级。STM32 通过串口将状态发送至 ESP32，ESP32 再通过局域网将数据发送至手机 APP，同时接收控制指令回传给 STM32 控制电机启停。OLED 同步显示当前状态，形成采集—处理—反馈—控制的闭环链路。

第二部分　系统组成及功能说明

2.1 整体介绍

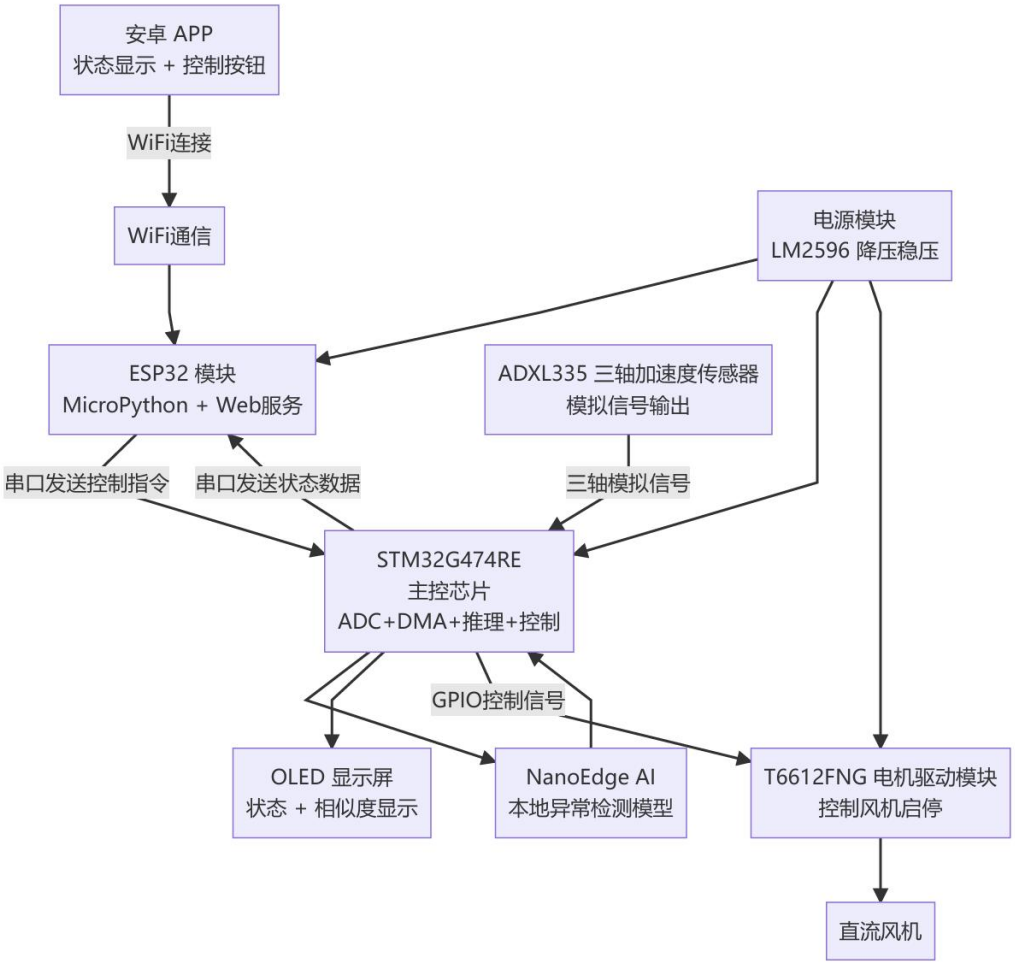

图 1 系统框图

本系统采用模块化设计，如**图 1** 所示，围绕 STM32G474RE 构建，完成了从振动采集、边缘 AI 推理到状态反馈与远程控制的完整闭环。各功能模块通过标准通信接口协同工作，整体结构紧凑，具备嵌入式智能能力。

系统通过 ADXL335 传感器采集三轴振动信号，STM32 定时器触发 ADC 同步

采样，DMA 实现高效传输。数据填满后调用 NanoEdge AI 模型进行本地推理，输出状态相似度。

推理结果经 OLED 实时显示，并通过串口发送至 ESP32，由其转发至手机 APP，实现状态监控与控制指令下发。风机由 TB6612FNG 驱动模块控制，STM32 输出 GPIO 信号实现启停。

供电采用 12V 输入，DC-DC 转换为 5V 与 3.3V，分别供各模块使用，确保系统稳定运行。整体具备低延迟、高响应、脱网运行等特点，适用于风机等旋转设备的振动异常检测与远程维护。

2.2 硬件系统介绍

2.2.1 硬件整体介绍；

（1）本系统的硬件部分围绕 STM32G474RE 微控制器构建，采用模块化设计思想，将传感器采集、边缘 AI 推理、状态反馈与无线通信等多个功能模块进行有机整合。

（2）系统核心为 STM32G474RE 微控制器，负责三轴振动信号的高速采样、边缘 AI 模型的本地推理执行，以及各模块之间的协调控制。采样部分采用 ADXL335 三轴加速度传感器，通过三个模拟输出通道将 X、Y、Z 方向的振动电压信号送入 STM32 的 ADC 通道。系统使用定时器触发 ADC 采样，配合 DMA 实现数据的高速搬运，确保采样过程稳定、无阻塞。

（3）为了实现状态信息的本地可视化显示，系统集成了一块 0.96 英寸 OLED 显示屏，采用 I2C 接口连接至 STM32 微控制器，实时输出当前相似度数值与状态等级，便于现场观察系统运行情况。显示模块占用资源少、更新速度适中，适配本项目对实时性与功耗的平衡需求。

（4）在无线通信方面，系统搭载了一块 ESP32 模块，作为通信从机，通过 UART 串口与 STM32 进行数据交互。ESP32 内部运行 MicroPython 程序，构建轻量级 Web 控制平台，可实现与手机端 APP 的双向通信，用户通过 WiFi 网络可实现风机启停控制，并接收异常监测结果的反馈信息，提升系统的远程操作能力与用户体验。

（5）风机电机控制采用 TB6612FNG 电机驱动模块，驱动信号由 STM32 生成并通过 GPIO 引脚送出，控制风机的启动与停止。该模块具备双通道驱动能力和过热、过流保护功能，保障在持续运行条件下的电气安全性。

（6）电源系统方面，采用 12V 电源搭配 3.3V/5V 降压模块，保障各功能模块长期稳定运行。

2.2.2 机械设计介绍

图 2 CAD 平面图

本系统的结构框架采用亚克力板+铜柱的分层组装形式，三块亚克力板都采用相同的设计，如图 2 所示，整体造型规整、模块布局紧凑，兼顾展示性与可维护性。主体由三块尺寸相同（约长 140mm 宽为 78mm 高为 2mm）的激光切割亚克力板构成，三层板通过八根 M3 铜柱及螺丝固定，形成稳定的垂直支撑结构。

结构设计充分考虑了通风散热、电源走线与信号隔离等工程因素。

2.2.3 电路各模块介绍

本系统采用 ST 官方的 STM32G474RE Nucleo 开发板作为主控核心，辅以面包板完成外围模块连接与功能验证，整个电路以原型验证为导向，不进行 PCB 定制设计，便于调试与快速搭建。

整机电路分为五个功能子模块：

（1）主控核心模块（STM32G474 Nucleo 开发板）

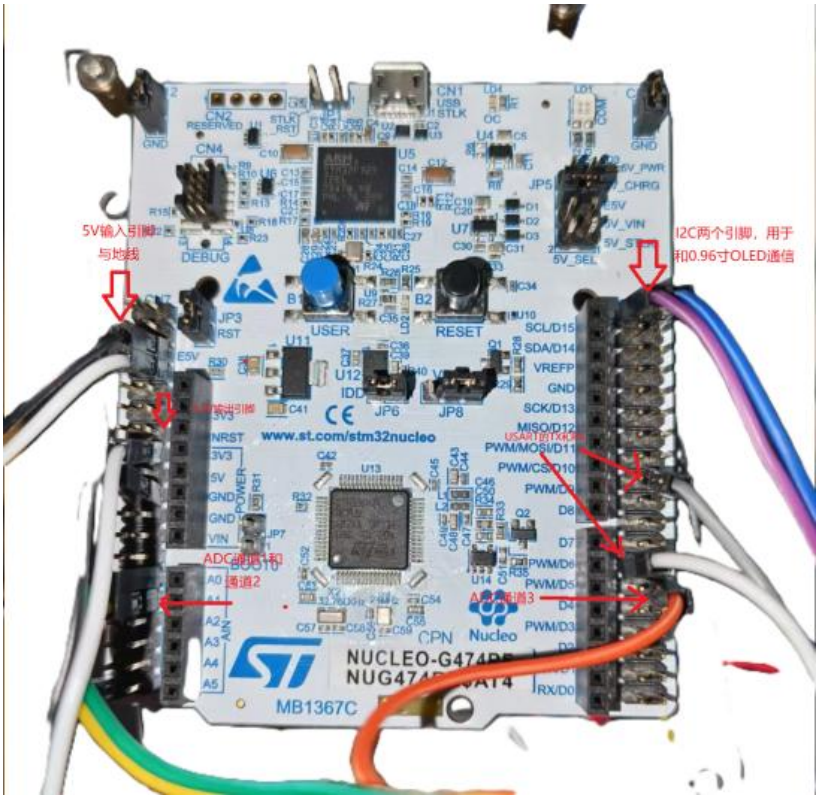

图 3 实物系统板电路连接指示图

主控采用 STM32G474RE 官方 Nucleo 板，具体的电路连接如**图 3** 所示，该系统板具备完整的调试接口（ST-Link）、板载电源管理和 IO 引出排针。使用其内部 ADC 模块进行三轴振动信号采样，利用 TIM + DMA 提高采样效率。

ADC 通道：PA0、PA1、PB14 接入 ADXL335 三路模拟信号；

UART 串口：PA10（TX）、PA11（RX）连接 ESP32；

I2C 接口：PB9（SCL）、PB10（SDA）连接 OLED；

GPIO 控制：PC2 高低电平控制风机停与转。

（2）三轴加速度模块（ADXL335）

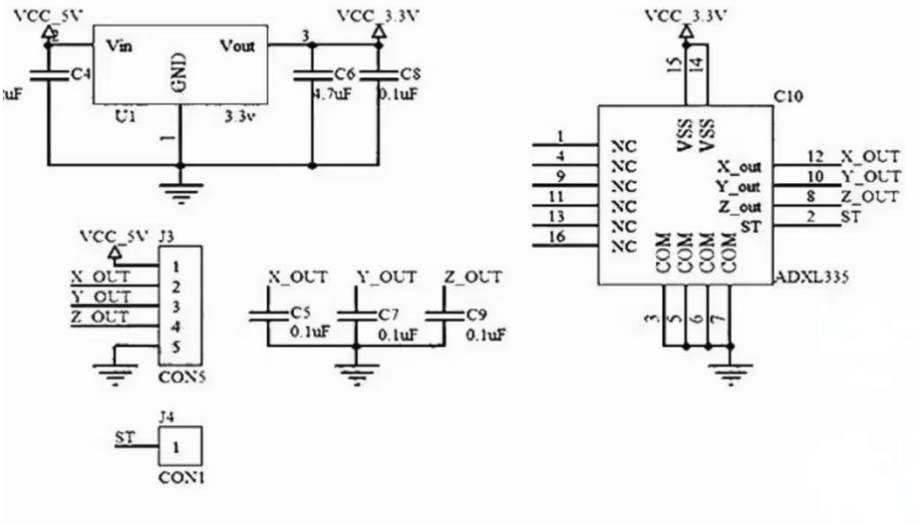

图 4 ADXL335 模块原理图

ADXL335 三轴加速度传感器通过 3 个模拟输出分别连接至 STM32 ADC 通道，模块供电由开发板的 3.3V 引脚提供，其原理图由**图 4** 所示。输出信号通过面包板跳线连接，无滤波处理，靠软件后处理完成数据清洗。

X 轴 → PA0（通道 0）

Y 轴 → PA1（通道 1）

Z 轴 → PB14（通道 2）

（3）显示模块（OLED）

使用常见 0.96 寸 OLED 显示屏，通信方式为 I2C，SCL/SDA 分别接至 STM32 的 PB9、PB10。由 STM32 控制刷新，显示当前相似度值和状态等级。供电由 3.3V 引脚提供，显示内容由主循环内定时更新。

（4）无线通信模块（ESP32）

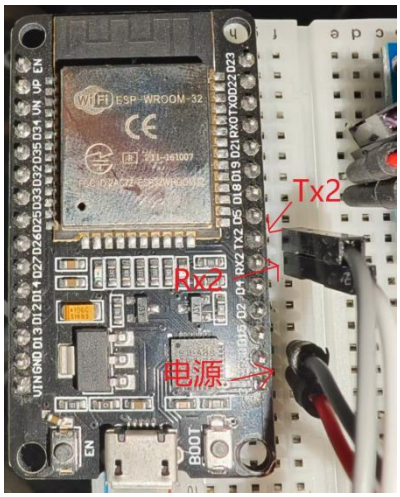

图 5 STM32 实物电路示意图

使用ESP32开发板,通过 UART 与 STM32 通信。由 ESP32 运行 MicroPython 程序，搭建 WiFi 控制端，并将手机指令转发给 STM32，同时上传当前系统状态。电路连接可由**图 5** 知晓，即 ESP32 RX2 连接 STM32 TX（PB10），ESP32 TX2 连接 STM32 RX（PB11），GND 与整个系统共地，+5V 供电。

（5）风机驱动模块（TB6612 + 风机）

如**图 6** 所示，该电路巧妙地运用了 TB6612 模块本身的特性，因为无需对风机进行控速，而只需要控制停与转。

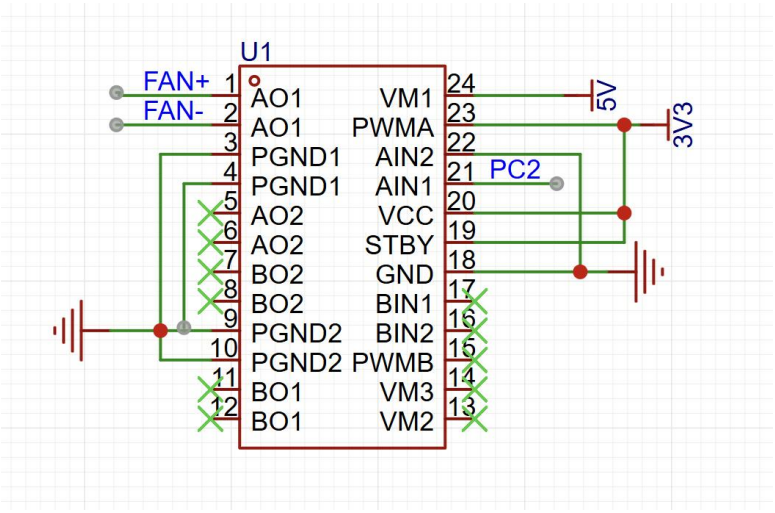

图 6 TB6612 简化连接原理图

STM32 的 PC2 输出高低电平信号，分别控制风机的转与停（风机的电源先分别连接 AO1、AO2），模块接线简洁，支持后期替换为其他有刷电机模块。

（6）电源模块

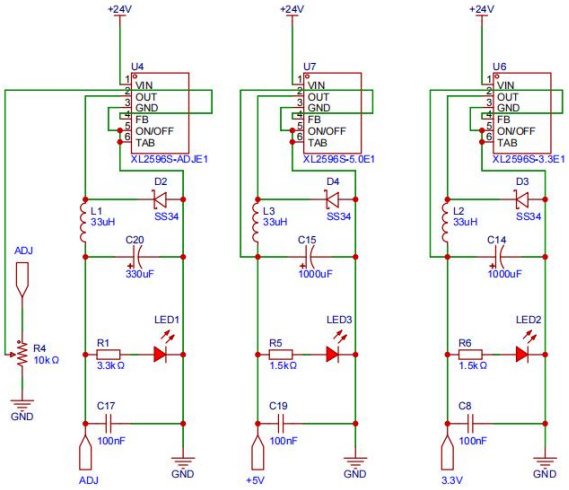

图 7 降压模块原理图

电源模块采用基于 LM2596 的降压稳压电路，通过淘宝购入的成品模块实现，原理图如**图 7** 所示。该模块支持 12V 输入，稳压输出为 5V 和 3.3V，用于为主控板与各功能模块提供稳定供电。为了增强文档的可读性与电路理解，本报告附上该模块的手绘电路图，标明了关键引脚及外围元件（包括整流二极管、电感、滤波电容等），便于后期调试与自主搭建。

2.3 软件系统介绍

2.3.1 软件整体介绍；

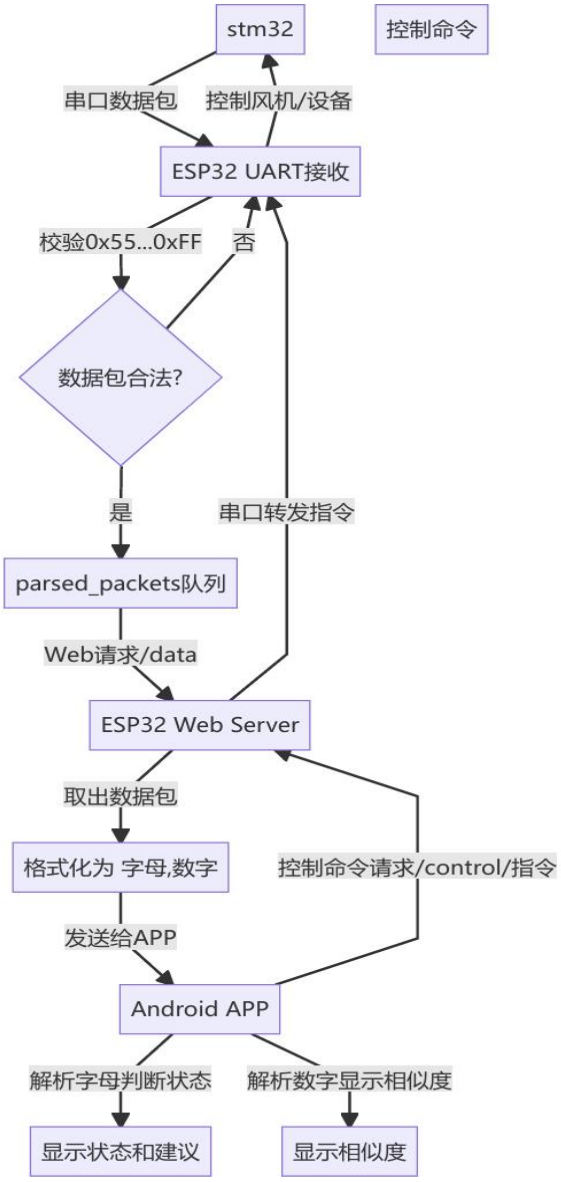

图 8 系统整体软件流程图

本系统的软件部分由 STM32 主控程序、ESP32 通信服务程序和安卓 APP 客户端三部分组成，构建了完整的边缘智能监测与远程控制闭环，如图 8 所示。STM32 程序基于裸机结构开发，负责完成传感器数据采集（ADC+DMA）、边缘 AI 推理（NanoEdge 模型）、状态判断与 OLED 显示，并通过 UART 与 ESP32 通信，同时配合硬件独立看门狗实现系统自恢复。ESP32 采用 MicroPython 编写，作为通信桥梁负责创建本地 WiFi 热点，接收 STM32 发送的状态信息，提供网页或 APP 控制接口，将用户指令回传给主控，实现

脱网环境下的远程操作。安卓 APP 由 Kotlin 编写，通过 WiFi 与 ESP32 热点直连，提供启停控制按钮、实时状态与相似度显示，以及异常报警提示等功能，提升用户交互体验。三者协同运行，系统具备良好的功能集成度、响应速度与可移植性。

2.3.2 软件各模块介绍

（1）STM32 部分：

STM32 作为系统的核心控制器，软件采用模块化结构开发，主要负责数据采集、异常检测、状态判断与信息交互等功能。主程序通过化 ADC、DMA、USART、I2C、TIM、独立看门狗等外设资源，系统上电后完 成初始化并进入主循环。采集部分通过定时器触发三轴加速度数据的高速采样，采样频率为 500Hz，存入 adc_buffer 缓冲区；DMA 设置数据就绪标志。

在数据处理环节，主循环检测数据是否就绪，若满足条件则停止采样，连续进行三轮 NanoEdge AI 推理以获取多个相似度值，最终通过去极值平均函数 Similarity_Standard() 得出稳定的相似度结果，用于后续状态判断。推理流程中调用 Run_NanoEdge_Inference() 加载模型、填充输入并获取当前运行状态的相似度分值（0~100）。

通信交互部分使用串口 UART3 与 ESP32 建立指令通道，支持“启动检测”、“停止检测”、“风机开启”、“风机关闭”四类命令，通过空闲中断与 DMA 协同实现非阻塞收发。OLED 显示模块通过 I2C 与 STM32 通信，调用 OLED_ShowStatusAndSimilarity() 显示当前运行状态与相似度百分比，便于用户实时查看。

此外，系统集成独立看门狗 IWDG，周期性在主循环中刷新，若系统卡死可自动复位恢复，提升长期部署稳定性。整个 STM32 软件部分运行稳定，结构清晰，逻辑严谨，能有效支撑边缘 AI 振动监测任务。

（2）ESP32 部分：

ESP32 作为本系统中的无线通信模块，主要承担与 STM32 的数据交互以及与 APP 之间的 WiFi 通信桥梁功能。其软件系统基于 MicroPython 实现，结构简洁明晰。程序运行后，首先通过 network 库配置 ESP32 为 WiFi AP 模

式，创建名为“ESP32-FAN”的局域网热点，IP 地址固定为 192.168.4.1，供 APP 端接入。随后初始化 UART2 串口（TX=17，RX=16），波特率设为 9600，用于与 STM32 进行指令和数据的交互。

系统通过一个异步任务 read_uart()持续读取来自 STM32 的 UART 串口数据，并利用自定义协议格式（以 0x55 为包头、0xFF 为包尾，总长 4 字节）对数据流进行解析。解析出的数据包被暂存于一个队列中，等待被 APP 端访问调用。

为便于 APP 访问数据，ESP32 搭建了一个基于 Microdot 框架的轻量级 Web 服务器，并定义了两个主要接口：/data 与/control/<cmd>。当 APP 向/data 请求时，服务器返回解析出的数据包（格式为 A,85 等）。当 APP 向/control/R 这类路径发送控制命令时，ESP32 将该命令转发至 STM32 的 UART，实现风机的启停控制、运行状态切换等操作。整个系统以异步方式运行，主事件循环通过 asyncio.run() 启动，确保串口监听与 HTTP 服务并发处理，系统响应灵敏、结构稳定，能有效支持边缘 AI 设备的远程控制与数据监测功能。

```
While True:
if uart 有数据:
    追加到缓冲区
while 缓冲区长度 >= 4:
    if 第 1 字节是 0x55 且第 4 字节是 0xFF:
        提取合法数据包
        存入 parsed_packets 队列
    else:
        丢弃首字节
```

（3）安卓端自研 APP

APP 模块介绍（Android 端）

本系统配套设计了一款基于 Android Studio 开发的简易 APP，用于与 ESP32 无线热点通信，实现风机控制与异常监测信息的展示。

APP 开发采用可视化设计方式完成，界面结构与功能模块通过 Android Studio 构建，以 Kotlin 语言编写，部分通信功能借助网络请求库实现。

APP 主要由以下几个界面组件构成：

状态显示区域：用于实时显示从 ESP32 获取的风机运行状态（正常/警告/异常）；

相似度显示区域：以文本，展示相似度数值（0~100）；

控制按钮区域：提供“运行（R）”、“暂停（S）”、“风机开（F）”、“风机关（X）”等四个功能按钮；

网络状态显示：显示是否成功连接 ESP32 热点。

如下是经过简化的伪代码：

```
开机时:
    自动尝试连接 ESP32 创建的 WiFi（ESP32-FAN）

每隔 2 秒:
    通过 GET 请求访问 http://192.168.4.1/data
    如果返回为 "A,91":
        状态文字显示“正常”，相似度为 91
    如果返回为 "D,65":
        状态显示“异常”，相似度为 65

按钮被点击时:
    向 http://192.168.4.1/control/<指令> 发送 GET 请求
    如按钮“暂停”对应 control/S，风机立即暂停
```

同时 APP 通过 Kotlin 的事件绑定机制，将按钮点击事件与对应网络请求绑定；所有通信均基于 HTTP 协议，通过固定 IP 192.168.4.1 实现局域网通信；未使用复杂的数据存储或本地数据库，仅实现最小可用功能。

2.3.3 模型的训练与评估

本作品采用 ST 推荐的 NanoEdge AI Studio 训练机器学习模型，使用内置功能 AD（Anomaly Detection，即异常检测，方法是对比当前采集的值与预训练数据之间的相似度，进而检测当前状态是否异常），最终模型具备较小的内存占用和较高的准确率，满足嵌入式部署要求。

（1）数据的采集

首先，数据的采集格式为三轴加速度数据，每个轴采集 512 个 12 位无符号整数，由 STM32 采集，每组包含正常与异常样本各一份（其中异常数据是在风机正常运行时手动加入一些人为的干预，如在叶片上安装异物，或者拧松风机平台固定螺丝等），然后通过串口发送至 PC 端，PC 端使用 Python 程序通过循环监测并存储数据为 CSV 格式数据，精选代码如下：

```
for group in range(GROUP_COUNT):
    group_data = []
    while len(group_data) < SAMPLE_PER_GROUP:
        raw = ser.read(2)
        if len(raw) == 2:
            value = struct.unpack('<H', raw)[0]  # 小端解码 uint16
            group_data.append(value)
    all_data.append(group_data)
    print(f"已接收第 {group+1}/{GROUP_COUNT} 组数据")

ser.close()
```

（2）数据的清洗

数据采集过后，同样通过 python 将六组（三轴的正常与异常数据）数据合并并清洗，接着另外添加一些异常特征（如 0 数据），确保 CSV 数据的健壮性，接着做归一化。最终将正常和异常的数据分别合并为两个 CSV 文件，已适配 NanoEdgeAI Studio 训练模型的要求，如图 9 所示。

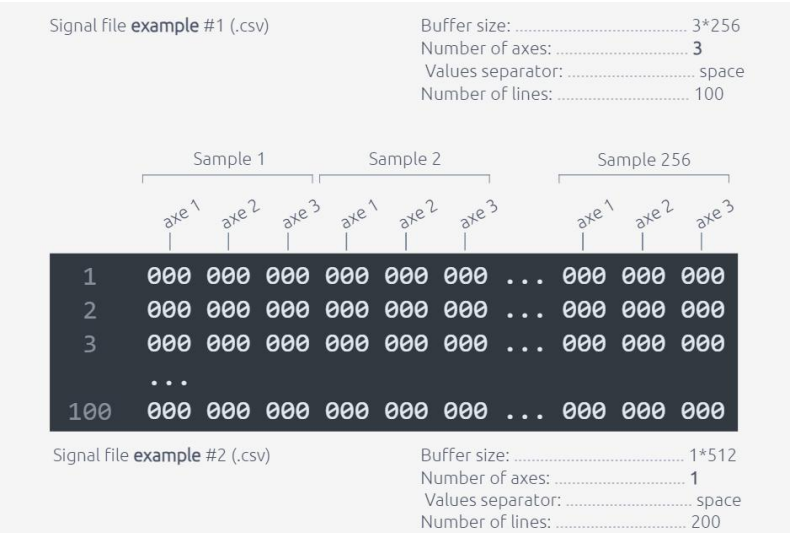

图 9 NanoEdge 数据标准图

（3）数据集特征的提取

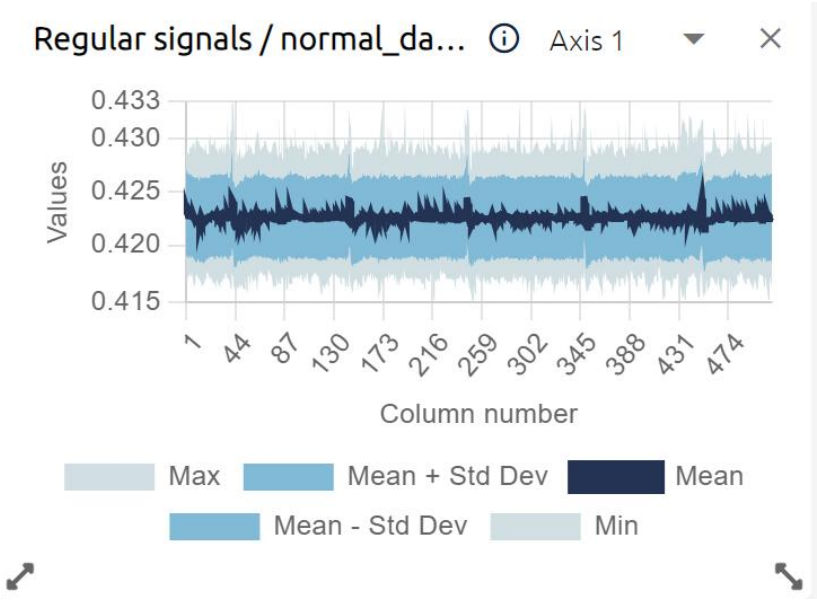

图 10 Axis1 中值的特征提取

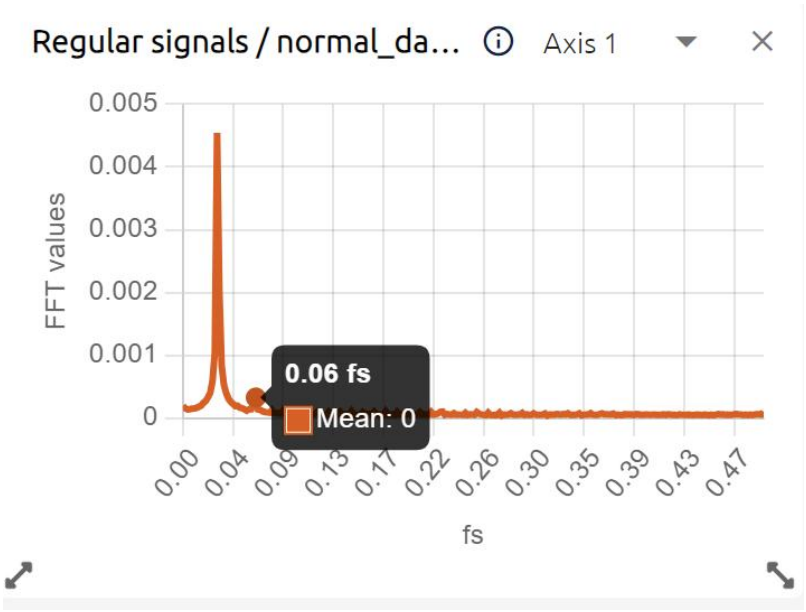

图 11 频域内的特征图

由图 10、图 11，这里以 Axis1（X 轴）为例，列出特征值的时域和频域图，尤其是频域图，其中的 0.00 到 0.04 中的某一段值使得 FFT 值极高，这一点可以体现出异常数据。表明采集数据与清洗流程具备较高的质量。

（4）模型的训练

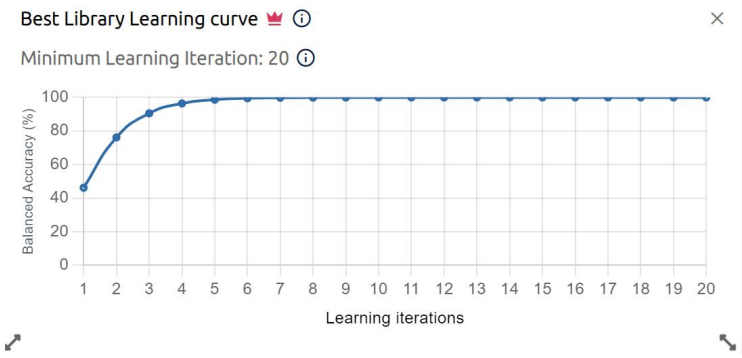

图 12 训练轮数与准确度的关系

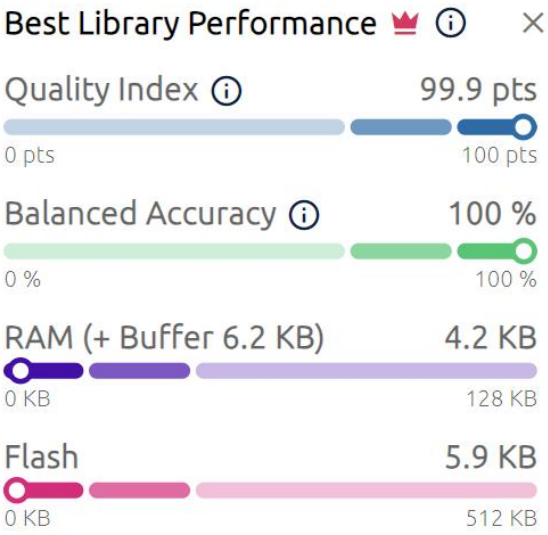

图 13 表现最优的模型参数

Library ID	Quality Index	Balanced Accuracy	RAM	Flash	Minimum Learning Iteration	Execution Time	Model
38	99.9 pts	100.00 %	4.2 KB	5.9 KB	20		MML
40	99.9 pts	100.00 %	3 KB	3.1 KB	20		ZSM
32	99.9 pts	100.00 %	4.7 KB	8.5 KB	20		ICM
36	99.9 pts	100.00 %	6.8 KB	8.3 KB	20		MML
39	99.8 pts	100.00 %	7.5 KB	6 KB	20		ZSM
37	99.8 pts	100.00 %	9.3 KB	7.3 KB	20		ZSM

图 14 训练出的若干模型的比较

本次模型的训练采用的是“黑盒”的形式，在 NanoEdgeAI Studio 中导入正常与异常数据的 CSV 文件后，就可以直接开始训练了，最终训练模型的好坏取决于数据集的质量和软件内置的预训练模型与数据集的匹配程度，即通过**图 14 中**的 Quality Index（质量指标，使用加权指标计算的，以在可靠性、性能和内存占用之间找到最佳平衡）与 Balanced Accuracy（平衡精度，灵敏度和特异性的平均值，就是指该模型优不优秀）。

经过对不同模型的 20 轮训练，最终得到了表现最好的模型，其精确度和内存占用都非常优秀，**图 14** 中最优秀的模型即为**图 13** 所示，可见**图 14** 中序号为 38 的模型在准确率与 RAM 占用之间达到了较优平衡，需要说明的是这里的

Balanced Accuracy 的值为 100 并不是传统机器学的“过拟合现象”，而是平衡精度，即模型与数据集匹配程度高不高，一般情况下如果数据集质量够高 Balanced Accuracy 为 100 很正常。

（5）模型精确度的评估

NANOEDGE AI STUDIO

Library Report

project-2025-05-31-13-47
2025-05-31-13-49-Benchmark
Created Date: 31 May 2025 13:49
Library ID 38
5.0.2

CONFUSION MATRIX

		Predicted	
		nominal	anomaly
Actual	no...	40	0
	ano...	0	72

DATA

Data Name	Sensor Type	Signal Data Length	Number of signals	Data Repartition Score
normal_data.csv	multiGeneric	1536 (512 * 3 axis)	40	Average ★★★☆☆
abnorma_data.csv			72	

图 15 模型数据集数据

PERFORMANCE

Performance	Value
Quality Index	99.87 pts
Balanced Accuracy	100.00 %
RAM	4.2 KB
Flash	5.9 KB
Accuracy	100.00 %
F1-Score	100.00 %
ROC AUC Score	100.00 %
Functional Margin	100.00 %

图 16 模型参数详细评估

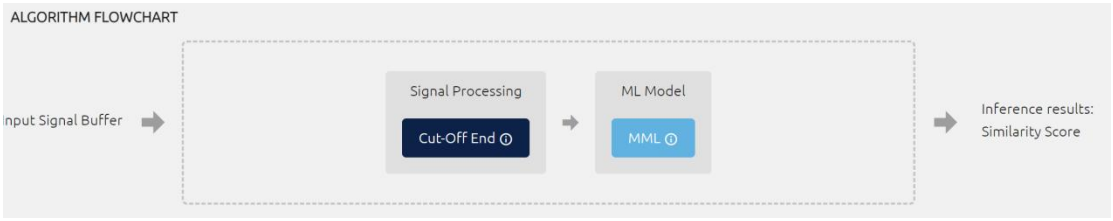

图 17 迁移学习示意图

说明评估结果之前，先解释一下相关专业术语：

Accuracy：最基本的度量，是预测值与实际值之间的比率

F1-Score：一个指标，通过考虑所有的好的和错误分类的样本来反映分类器的整体性能。

ROC AUC Score:描述决策阈值与分类器性能之间的关系。

Functional Margin:函数边际揭示了类之间的分离距离。它代表了一个正确而自信的预测。

图 15、**图 16**、**图 17** 三张图为 NanoEdgeAI Studio 输出的报告，通过评估，我们知道其参数设置合适，模型在提供的数据集上准确率符合标准。同时，在多轮模型对比中，该模型在准确率与资源占用之间达成最优权衡，适合部署在资源有限的 STM32 平台。综上，所训练模型满足本项目对识别准确性与资源占用的综合要求。

第三部分　完成情况及性能参数

3.1　整体介绍

本系统最终完成了从传感器数据采集、异常检测分析、到无线控制与数据显示的完整闭环。实物整体结构如**图 18**、**图 19** 所示，主要由三层亚克力板构建，内部通过铜柱和螺丝紧固，结构稳定，便于拆装调试。

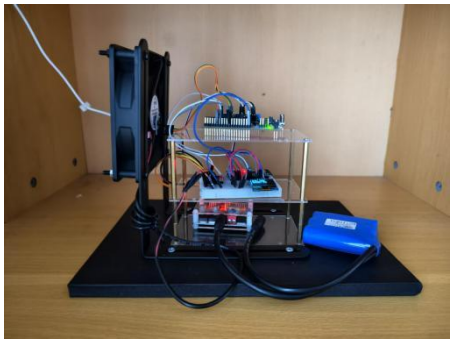

图 18 实物正面图

板载模块包括：STM32 主控板、ADXL335 三轴加速度模块、OLED 屏幕、ESP32 通信模块、电源模块、风机本体以及 TB6612 驱动模块等。外接面包板用于电路连接，布局合理，利于开发与修改。

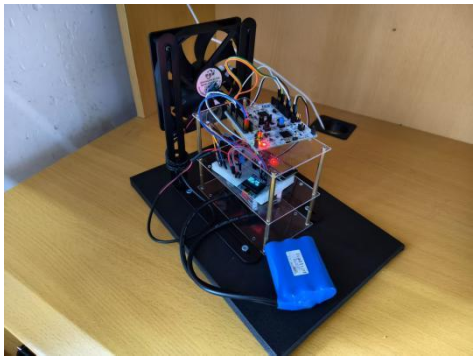

图 19 实物斜 45° 图

3.2 工程成果

3.2.1 机械成果；

图 20 机械成果图

如图 20，本作品的机械成果是一个由 PVC 贴纸与木板组成的底座以及由三块亚克力板与若干铜柱螺丝组成的平台。

3.2.2 电路成果；

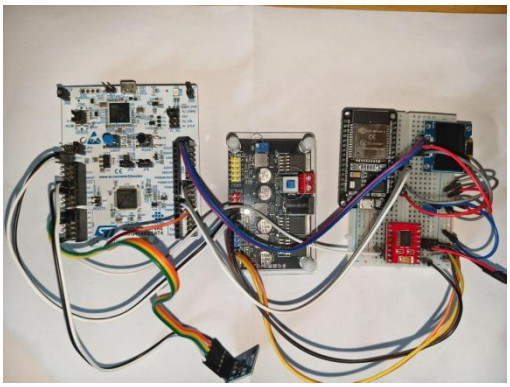

图 21 电路成果图

如图 21，本作品未采用传统的 PCB 板而使用模块化的电路结构和面包板通过杜邦线连接，具有极高的灵活性和可扩展性。

3.2.3 软件成果；

图 22 软件成果图

如图 22，本作品配套了一个自研的安卓端 APP，可检测热点“ESP32 Fan”是否已连接，可显示接收到的数据和已发送的命令，另外还有四个按键功能。

3.3 特性成果

3.3.1 APP 四个按钮功能描述

图 23 软件按钮功能图

如图 23，这里解释一下按键作用：

启动系统（R）：打开系统，为上电后的默认状态

停止系统（S）：发送停止命令，让 STM32 内部的推理模型停止运行，同时停止数据的发送和 OLED 的刷新

打开风机（F）：启动风机

关闭风机（X）：关闭风机，为上电后的默认状态

3.3.2 风机的正常运行状态

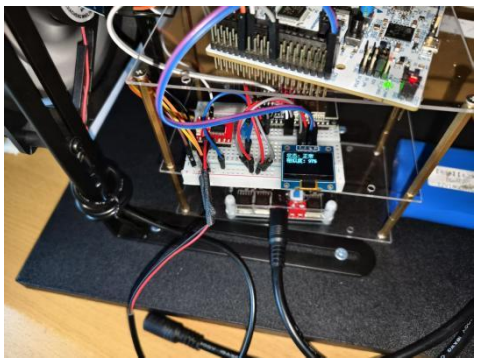

图 24 正常运行时实物图

图 25 正常运行时软件图

如图 24、图 25 所示，在风机正常的运行状态下，相似度在 90%以上。

3.3.3 风机的警告运行状态

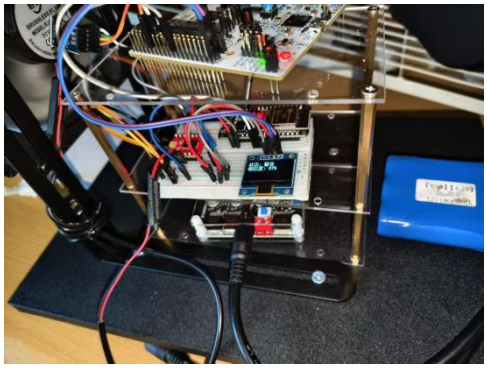

图 26 警告状态实物图

图 27 警告状态软件图

如图 26、图 27 所示，由于“警告”状态属于一种过渡期，故通过倾斜平台来模拟，相似度为 80%~90%。

3.3.4 风机的异常运行状态

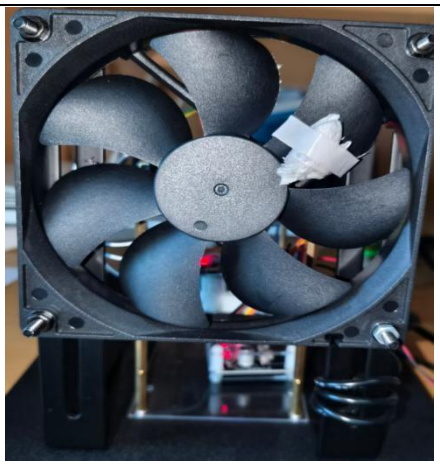

图 28 风机有异物图

如图 28 所示，在风机叶片上绑上异物，以模拟风机叶片损坏或者轴心偏移的情况。

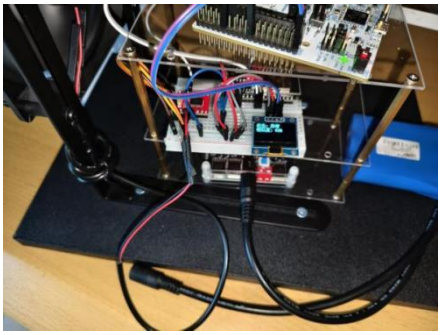

图 29 异常实物图

图 30 软件异常图

如图 29、图 30 所示，在异常的状态下，相似度会在 80%一下，通常在 60 左右。

第四部分　总结

4.1　可扩展之处

本系统具备良好的模块化结构与可扩展性。在通信层面，ESP32 模块可替换为支持 LoRa 或 NB-IoT 的通信芯片，实现远距离无线组网与云平台对接。在分析方法上，系统可引入 FFT 或小波分析方法进行频域特征提取，增强模型对不同类型异常的识别能力。在控制策略上，风机启停可进一步细化为多档 PWM 调速控制，适配更多工业设备场景。在平台兼容性方面，系统可移植至 STM32H7 等资源更丰富的 MCU，或配合 RT-Thread、FreeRTOS 实现多线程任务调度，进一步提升实时性与可维护性。同时，自研 APP 可添加报警推送、日志记录、图形化报表等功能，增强实用性与数据可追溯性。

4.2　心得体会

本次项目的整体设计和实现过程，是一段既痛苦又充实的旅程。从最开始确定系统架构开始，到后期各个模块联调，每个阶段都有自己的挑战。STM32 作为主控，负责数据采集和控制逻辑，而 ADL335 三轴加速度传感器则通过 ADC+DMA 机制完成信号采样，整个流程的实时性和稳定性是最关键的一环。推理部分使用了 ST 官方的 NanoEdge AI 库，在数据处理策略上加入了多次采样、去极值平均等方式，使得异常检测结果更加可靠。最初我以为 AI 部分是“黑盒子”，但越做越发现，数据预处理远比想象中重要。

通信方面，我选用了 ESP32 模块，通过 UART 与 STM32 通信，同时充当 WiFi 热点并运行一个简易的 REST 服务，供手机 APP 访问。ESP32 端基于 Microdot 构建了一个异步 Web 服务器，采用环形缓冲+固定格式数据包协议，解决了串口数据不稳定、APP 控制指令不及时等问题。APP 由 Android Studio 编写，用 Kotlin 实现了简洁的界面和基本的控制功能，虽然界面不复杂，但基本功能全部打通。最终形成了“传感器-STM32 边缘 AI 分析-ESP32 转发-APP 显示与控制”的闭环系统，实现了远程实时监测与控制。

整个开发过程虽然资源有限，使用了面包板、模块化拼接等“非正式手

段”，但每个功能都是在调试和验证中跑通的。从电路设计、软件开发，到系统调试，我对嵌入式开发流程的完整链条有了更深入的认识。这次经历让我理解了“功能实现”与“系统鲁棒性”之间的差距，也让我意识到代码逻辑清晰、模块解耦的重要性。虽然过程中也遇到了不少挫折，比如串口通信中断、数据帧错乱，但这些都成为我成长过程中的宝贵经验。未来如果有机会继续优化这个项目，我希望能进一步规范 PCB 设计、优化通信协议、提升 APP 交互体验，让这个系统从“能跑”走向“能用”。

第五部分　参考文献

按照标准格式，限 20 篇以内。

[1] STMicroelectronics. STM32G474RE Datasheet [EB/OL]. https://www.st.com

[2] STMicroelectronics. STM32CubeMX User Manual [EB/OL].

[4] NanoEdge AI Studio: User Manual. ST, 2024.

[5] Espressif. ESP32 Technical Reference Manual [EB/OL].