基于ESP32微控制器的端侧语音处理系统设计与产品实现

摘 要： 本项目成功设计并实现了一款基于ESP32微控制器的端侧语音处理系统，旨在结合人工智能技术与物联网嵌入式系统，提供低成本、高效率的AI模型部署。项目成果是一款集硬件设备和软件系统于一体的智能手表，具备手势控制和语音识别功能。硬件设计采用ESP32-PICO-V3-02为核心，集成Wi-Fi和蓝牙功能，并通过SiP封装技术实现小型化。软件方面，基于MFCC和CNN的音频识别模型在MCU上成功部署，准确率达到95%，响应时间在300ms以内。系统集成了环境光传感器、全向麦克风、扬声器功放、陀螺仪和TFT显示屏等外设，支持时钟、日历、设置、手电筒和AI问答等功能。项目通过手势传感器和实体按钮实现用户交互，并具备云端语音识别助手功能，通过流式传输接入云端大语言模型，实现智能化控制。最终，项目满足了设计目标，展现了系统的智能化和稳定性，为未来物联网端侧智能设备的发展提供了新的可能性。

关键词： 可穿戴式设备、微型电子设备、嵌入式人工智能部署、云端模型部署

**Design and Product Implementation of an Edge Voice Processing System Based on ESP32 Microcontroller**

**Abstract:** This project successfully designed and implemented an edge voice processing system based on the ESP32 microcontroller, aiming to combine artificial intelligence technology with IoT embedded systems to provide low-cost, high-efficiency AI model deployment. The project outcome is an intelligent watch that integrates hardware and software systems, featuring gesture control and voice recognition functions. The hardware design uses the ESP32-PICO-V3-02 as the core, integrating Wi-Fi and Bluetooth functions, and achieving miniaturization through SiP packaging technology. On the software side, an audio recognition model based on MFCC and CNN was successfully deployed on the MCU, with an accuracy rate of 95% and a response time within 300ms. The system integrates peripherals such as ambient light sensors, omnidirectional microphones, speaker amplifiers, gyroscopes, and TFT displays, supporting functions like clock, calendar, settings, flashlight, and AI Q&A. The project realizes user interaction through gesture sensors and physical buttons and has a cloud voice recognition assistant function, accessing cloud large language models through streaming transmission to achieve intelligent control. Ultimately, the project meets the design goals, demonstrates the system's intelligence and stability, and provides new possibilities for the development of future IoT edge intelligent devices.

**Keywords:** Wearable devices, Miniature electronic devices, Embedded AI deployment, Cloud model deployment

0 引 言

伴随着人工智能技术的兴起以及物联网嵌入式系统的需要，部署在端侧的人工智能方案逐步进入主流视野。常规的端侧计算单元采用微型电脑（如树莓派）或FPGA等器件，虽然性能强悍但成本居高，不适合大面积部署。如今MCU的算力已经大大增强，甚至有厂商专门为矩阵计算做到了硬件级的优化；同时有赖于内部集成的无线模块，在本地算力不足时也可以方便地实现对接云计算服务。基于此，可以在有限的成本内实现一定程度的端侧智能应用，具有广阔的使用前景。

本项目将实践在低成本MCU中进行AI模型部署和应用，充分利用芯片的运算能力和内存空间，设计一款面向日常应用的AI硬件设备。该硬件设备将专注于处理特定的AI任务，通过优化算法和模型压缩技术，确保在资源受限的MCU上也能高效运行。此外，考虑到用户交互的便捷性，设备将配备基本的输入输出接口，如按钮、LED指示灯和显示屏，以提供直观的操作体验。最终，本项目旨在开发出一款既经济实惠又功能强大的AI硬件产品，提供智能化应用解决方案。

根据前面的分析可见，设计一款便于模型端侧部署和云端联调的软硬件协同系统具有实际价值。本项目的研究目标为：1.设计一款基于ESP32-PICO-V3-02核心的音频开发板。2.硬件系统小型化：体积不超过43mm\*35mm\*12mm，以制作成穿戴式智能手表为目标3.软件上，在MCU上使用FreeRTOS实现基本的操作系统，并设计合理的清晰的基础接口，将显示、控制、数据处理等部分解耦，便于更多人再次该系统基础上的二次开发4.在MCU上部署基于TensorFlow-lite的语音识别模型5.使用无线功能接入云端模型（流式传输）6.模型部署完成后，利用语音识别的功能，实现一定程度的智能化操作。7.软硬件系统具有长期稳定性，保证长期使用的稳定性而不是实验性产品。

论文第一节描述该项目的硬件系统设计，第二节描述了该项目的软件系统设计，第三节描述了该项目的实现效果，第四节描述了该项目的最终成果和结论。

1 硬件系统设计与实现

1.1 器件选型

为保证该硬件系统足以承载端侧语音识别模型的运算以及云端模型的快速响应，必须选择一款具有高频、充足运行空间、集成无线传输功能的微控制器。ESP32-PICO-V3-02 的核心是 ESP32 (ECO V3) 芯片 \*。ESP32 是集成2.4GHzWi-Fi 和蓝牙双模的单芯片方案，采用台积电(TSMC)超低功耗的40纳米工艺。ESP32-PICO-V3-02SiP已将晶振、flash、PSRAM、滤波电容、RF 匹配链路等所有外围器件无缝集成进封装内，不再需要外围元器件即可工作。此时，SiP的组装和测试都在SiP 层面完成，因此ESP32-PICO-V3-02 可以大大降低供应链的复杂程度并提升管控效率。

其具有的8MB外置flash及2MB的PSRAM，可以保证端侧模型有足够的数据空间；且实现了系统级的芯片封装，只需要外部提供电源和匹配的天线即可正常工作。

为缩小硬件尺寸，本项目自行绘制开发板，在ESP32-PICO-V3-02核心的最小系统基础上（而不使用现有的ESP32成品模块，体积太大），整个PCB以1.69寸TFT屏幕以及304030锂电池作为尺寸标定。同时，创新性地使用沉金焊盘搭配磁吸数据线，替代占体积较大的USB接口实现供电与数据传输，也增强了设备的防水密闭性。

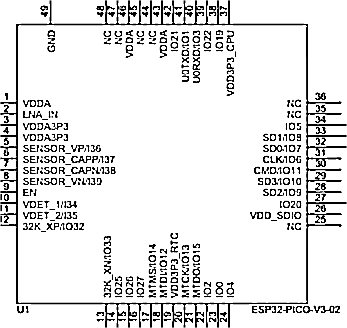
为了保证该系统的实用性和长期稳定性，需要对系统功耗作一定要求。使用AIR700E低功耗4G芯片替代ESP32的无线功能以降低整体功耗，并且实现脱离WIFI的广范围工作。

.基本硬件电路：

1. 串口：CH340K （UART）
2. 环境光传感器：APDS9960（I2C）以实现手势控制功能，为设备提供控制媒介
3. 全向麦克风：MSM261；扬声器功放；MAX98357A。（I2S）
4. 其他外设：陀螺仪：MPU6050。（I2C)
5. 显示：1.69寸 TFT屏，ST7789V驱动（SPI）

1.2 硬件布局、引脚分配

主控核心ESP32-PICO-V3-02共有48个引脚，其中包括7个NC分配引脚、8个电源引脚、6个仅输入引脚、1个使能、4个已连接内部flash、psram的不可用引脚和5个strapping引脚。排除掉其中不可用部分后，整个核心的引脚资源非常有限，因此必须作合理分配优化。



ESP32-PICO-V3-02原理图及封装

最终设计分配如下：

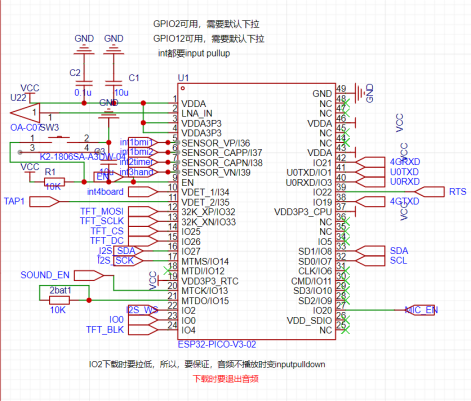


其中麦克风和扬声器共用一路I2S，分时复用

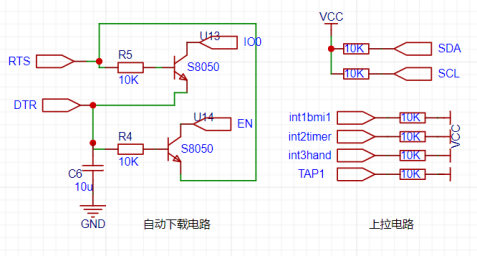
I2C地址分配：**CW2015电量计**0x62 ;**APDS9960手势传感器**0x68 ;**PCF8563 RTC时钟**0xa3 **;MPU6050 六轴陀螺仪**0x69

1.3 PCB设计

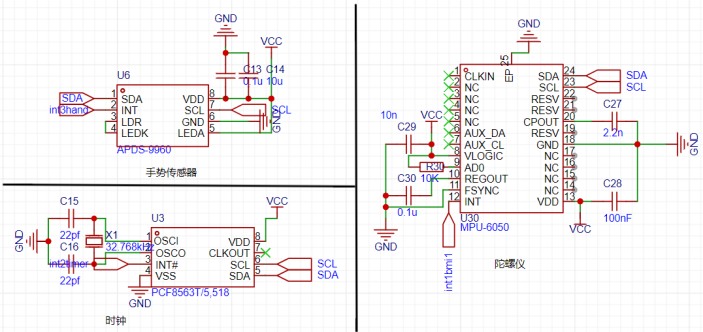
**（1）原理图**



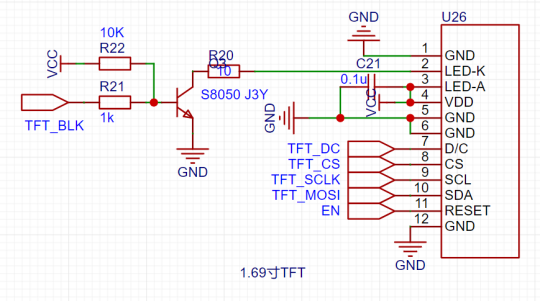
微控制器最小系统及外设引脚分配



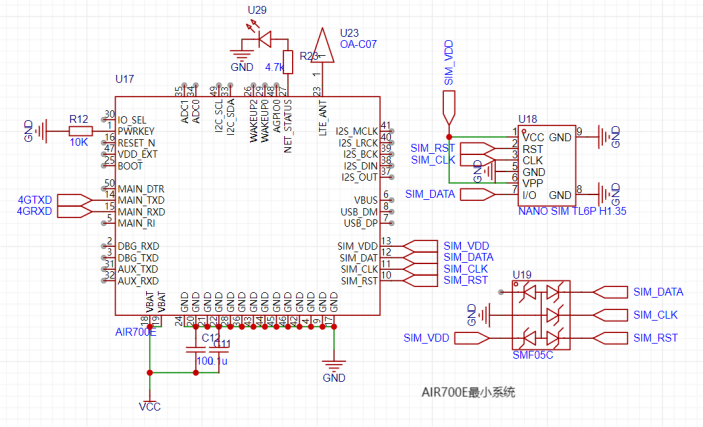
通过CH340K控制RTS和DTR实现上电瞬间进入下载模式而非运行模式实现自动下载



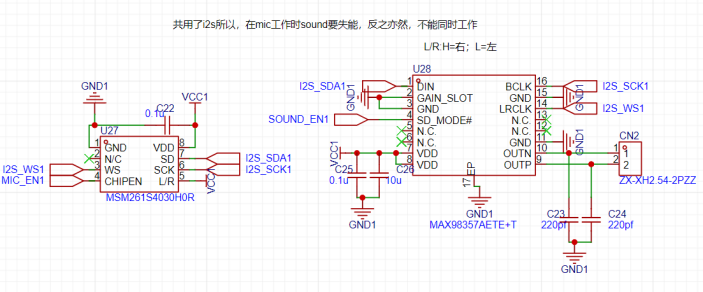
主要I2C外设



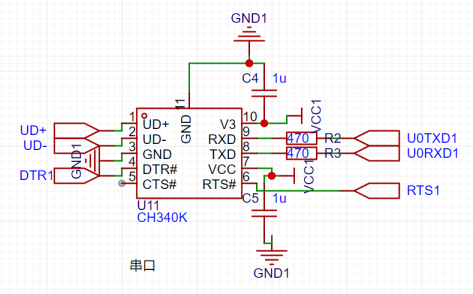
SPI TFT显示屏



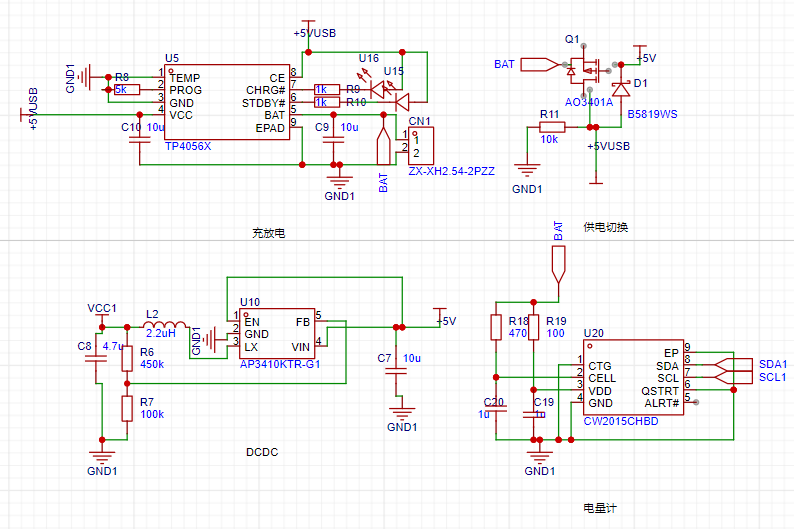
4G模块最小系统



麦克风 扬声器I2S外设

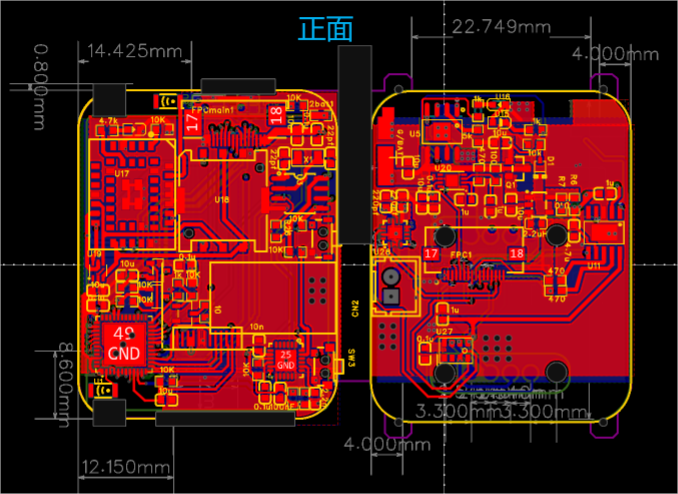


串口通信电路

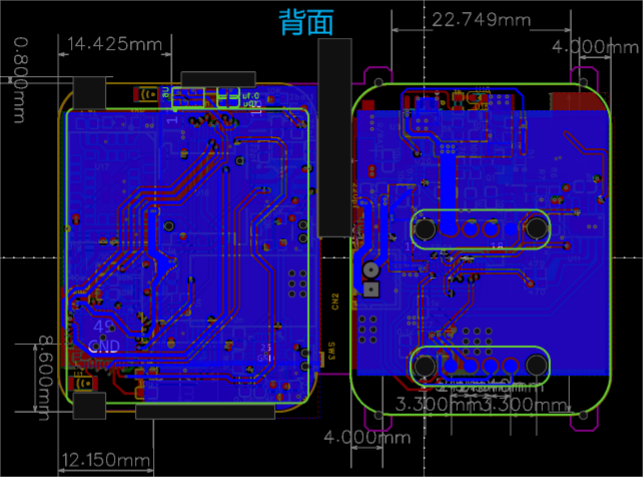


电源管理相关

**（2）PCB版图设计**



主板 背板



主板 背板

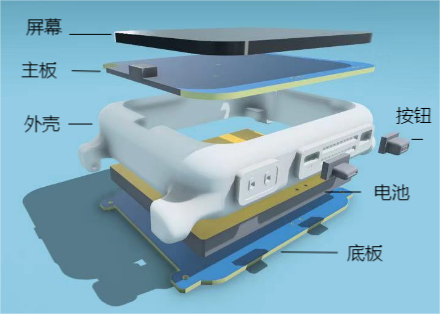
本设计采用了两块4层板的组合，二者使用FPC软排线连接，主板上分配了核心模块和主要功能外设，背板则分配了电源管理电路、串口电路和音频（功耗大）电路，以便将大功率和低功率元件分离；同时背板也作为整个硬件的后盖，通过磁吸接口实现外部供电和数据传输。

1.4 硬件外壳设计

使用SHAPR3D进行外壳设计，充分利用硬件空间，合理优化布局，尽可能把尺寸缩小

模型仿真图



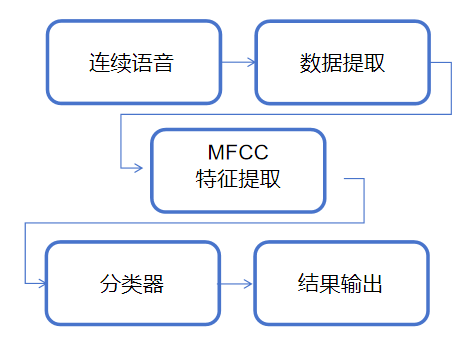
元件分解图

产品主体 佩戴演示

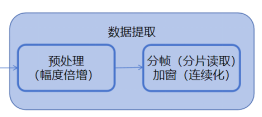
2 软件设计

2.1 基于梅尔倒谱系数的音频识别模型



识别流程图

2.1.1数据提取



**（1）预处理**

i2s\_read(I2S\_IN\_PORT, (void \*)sampleBuffer, i2s\_bytes\_to\_read, &bytes\_read, 100);

      for (int x = 0; x < i2s\_bytes\_to\_read / 2; x += 1) {

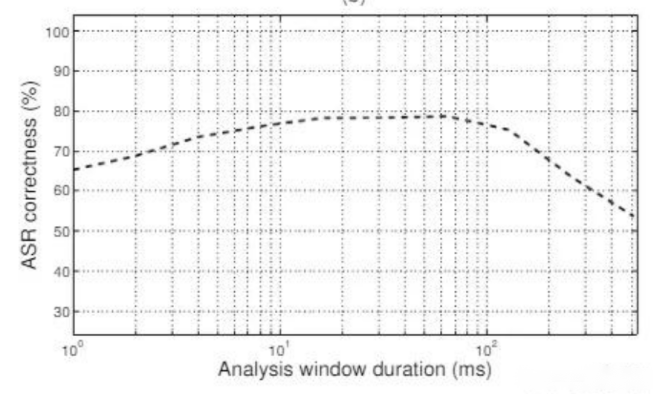
        sampleBuffer[x] = (int16\_t)(sampleBuffer[x]) \* 16;

      }对数

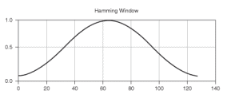
数据从全向麦克风MSM261中以I2S协议传出，每次获取512byte（一个缓冲区大小）数据，并将每个数据16倍增，以供后续处理。

**（2）分帧、加窗原理**

我们将获取到的音频数据连续地进行拆分，以便对数据进行快速分析和特征提取，我们将该过程成为分帧。但分帧的过程回产生一个担忧：由于我们对（相对而言）周期信号进行了截断，这将造成频谱泄漏的问题，突然截断的时域波形将产生极高频的分量（以及低频分量），因此我们需要添加适当的滤波器来抵消这一问题，这一步骤成为加窗（汉明窗）



分帧示意图[[1]](#footnote-0)



汉明窗示意图[[2]](#footnote-1)

**（3）数据提取的核心代码**

    // 根据频率将时间（0.25秒）转换为帧数

    const size\_t frame\_length\_values = frequency \* config.frame\_length;

    const size\_t frame\_stride\_values = frequency \* config.frame\_stride;

    const int frame\_overlap\_values = static\_cast<int>(frame\_length\_values) - static\_cast<int>(frame\_stride\_values);

上述代码完成了对于一组数据的帧分割准备：frame\_length\_values 和 frame\_stride\_values 分别计算帧长和帧移的采样点数。frame\_overlap\_values 计算帧重叠的采样点数。

    static int stack\_frames(stack\_frames\_info\_t \*info,

                            float sampling\_frequency,

                            float frame\_length,

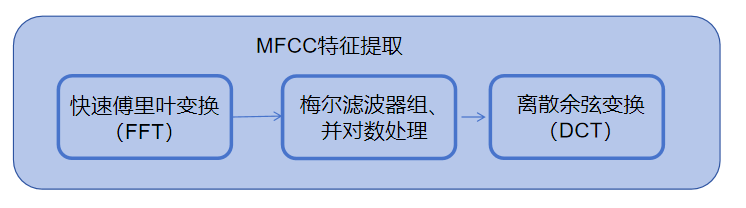
                            float frame\_stride,

                            bool zero\_padding,

                            uint16\_t version)

帧分割函数声明

2.1.2 MFCC特征提取的基本原理和流程

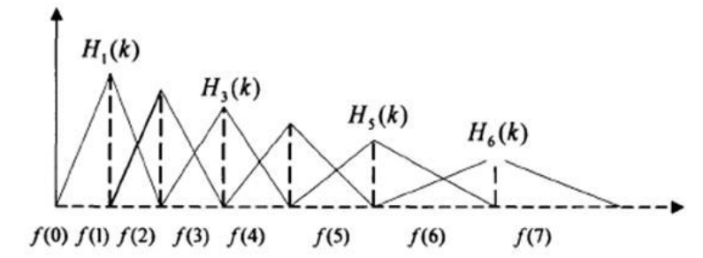


语音识别主要的分析对象是人的音色（而非音高）；它们主要体现在功率域（频域）上，音色和音高的叠加在频域上表现为不同频率点上的功率值不同；不同的人在说同一句话时的音高困难不同（高音和低音）但在语音识别的结果是一致的。这表明，我们需要区分音色和音高，并主要对音色进行信息识别和分类。

1. **快速（离散）傅里叶变换FFT**

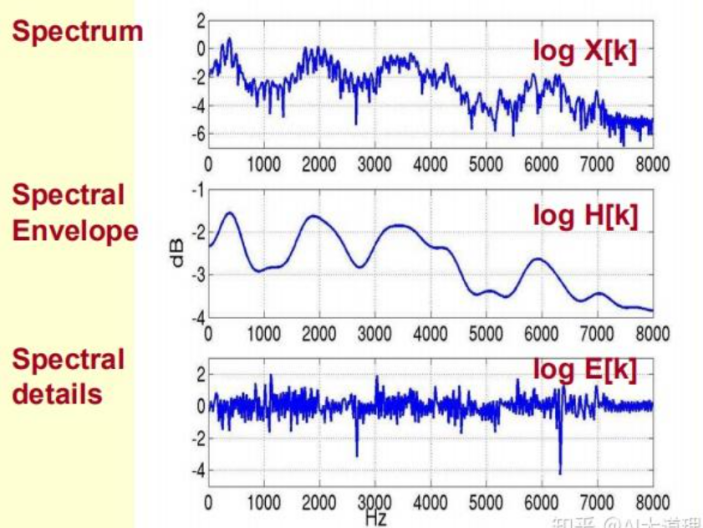
将语音信号从时域转化到频域（功率域）进行分析，获取音色和音高的混叠信息。（时域卷积-频域相乘，卷积-乘性转变）

1. **通过梅尔滤波器获得梅尔频谱**

将能量谱通过一组Mel尺度的三角形滤波器组，定义一个有M个滤波器的滤波器组（滤波器的个数和临界带的个数相近），采用的滤波器为三角滤波器，中心频率为 。M通常取22-26。各f(m)之间的间隔随着m值的减小而缩小，随着m值的增大而增宽，如图所示

梅尔滤波器的频域特性

人在接受声音信号时，耳蜗就相当于滤波器，耳蜗会对声音信号进行滤波，使人的听觉会对低频信号更加的敏感，这一特性保证了人可以在环境嘈杂的情况下仍然能够识别出各种声音。Mel 滤波器组通过模拟这种听觉特性，从而更有效地处理语音信号，使得提取的特征更符合人类的听觉习惯，Mel 滤波器还可以将信号转换为更低维度的特征表示，减少计算量，同时保留对声纹识别更重要的信息。[[3]](#footnote-2)因此设计梅尔滤波器组由低频到高频逐渐变疏、且幅值逐渐减小。

1. **对数据进行取对数处理，实现乘性-加性分离**

通过梅尔滤波器后的数据示意图

其中频谱的“缓变”部分为语音的音高，而快速变化的高频部分则为人的音色，（音高体现了一个人平均而言的语音功率主要集中的地方，反映为人的音调高低，而不携带有效信息）。我们将该数据通过对数之后，便将原本乘性的音色和音高信息转化为加性，便于我们区分。

**（4）离散余弦变换DCT**

根据上述分析，我们对梅尔频谱图再次进行变换分析，我们在取对数的过程中已经忽略复频率中的相位信息，因此对该频谱只需要作余弦变换结果，得到的结果中低频成分为音高信息，高频成分为音色信息。

至此，我们完成了音频数据的特征提取

**（5）经过调试后，我们设置MFCC的参数如下**：



2.1.3 MFCC核心代码

    static int mfcc(matrix\_t \*out\_features, signal\_t \*signal,

        uint32\_t sampling\_frequency, float frame\_length, float frame\_stride,

        uint8\_t num\_cepstral, uint16\_t num\_filters, uint16\_t fft\_length,

        uint32\_t low\_frequency, uint32\_t high\_frequency, bool dc\_elimination,

        uint16\_t version)

MFCC核心代码声明

**（1）快速傅里叶变换得到功率谱（实际隐含在MFE计算函数中了）**

   ret = numpy::power\_spectrum(

                signal\_frame.buffer,

                stack\_frame\_info.frame\_length,

                power\_spectrum\_frame.buffer,

                power\_spectrum\_frame\_size,

                fft\_length

            );

**（2）计算梅尔频谱图（MFE）特征**

// 计算 MFE 特征

ret = mfe(&features\_matrix, &energy\_matrix, signal, sampling\_frequency, frame\_length, frame\_stride, num\_filters, fft\_length, low\_frequency, high\_frequency, version); // 如果 MFE 计算失败，返回错误

if (ret!= EIDSP\_OK){ EIDSP\_ERR(ret); }

// 对 MFE 特征取对数

ret = numpy::log(&features\_matrix);

if (ret!= EIDSP\_OK) {EIDSP\_ERR(ret); }

获取梅尔频谱数据并将其取对数（卷积->乘->加）

// 对对数特征进行 DCT 变换

ret = numpy::dct2(&features\_matrix, DCT\_NORMALIZATION\_ORTHO);

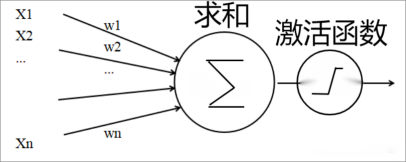
if (ret!= EIDSP\_OK) { EIDSP\_ERR(ret); }

对取对数后的频谱再进行离散余弦变化，得到梅尔频谱的倒系数谱MFCC，完成特征提取。

2.1.4卷积神经网络分类器

我们将得到的MFCC数据以图像的角度进行理解，则可以充分利用卷积神经网络进行分类学习。

**(1)神经网络分类算法原理**



神经元模型（单层感知机）

神经网络的最基本组成单元，其本质是一个加权求和器后接一个阈值激活函数（门），加权求和器模拟人的神经突触，从多点获取信息，激活函数模拟神经信号的选择性传递，直到达到特定阈值后才能进行有效输出（产生决策平面）。



多层感知机

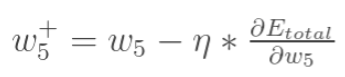
单层感知机具有一个显著缺陷，由于信息的获取来源于线性加权叠加，因此它通过激活函数产生的决策平面一定是线性的，也即它只是一个线性分类器。为了解决问题，我们引入了多层感知机的概念，将神经元级联，每一次的加权求和都是一次坐标变换，从而通过多次线性的坐标变换产生非线性的决策平面。

**（2）神经网络的学习原理（有监督学习）**

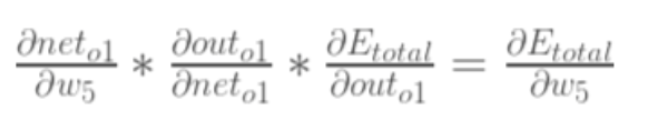
我们将从输入-结果输出的过程成为前向传播，而我们对于每个神经元的加权参数是未知的，因此需要通过一定的学习算法自主对加权矩阵w[]进行最优求解——反向传播。



误差函数示例



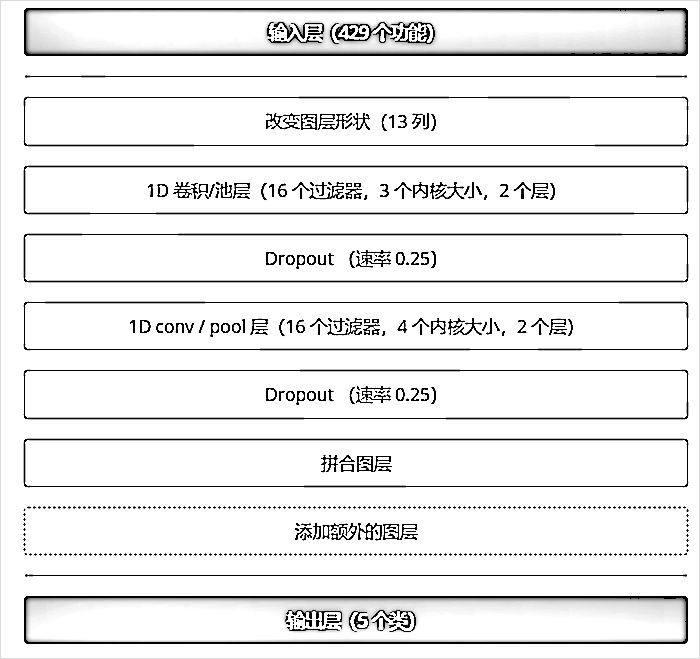
更新函数示例



链式法则示例

反向传播机制主要规定了误差函数Loss为输出结果对标签结果的方差，并将更新函数设置为w=w-adL/dw，其中a为学习率，dL/dw可以由链式法则得到，从结果往输入侧逐层传递，因此称为反向传播机制。现有的有监督学习算法大多参考多层感知机采用误差反向传播实现[[4]](#footnote-3)

**（3）本设计中的神经网络配置**



神经网络架构图

首先输入层接收来自MFCC的输出，共13（梅尔滤波器数目）\*33（均分频谱宽度）=429个输入参数。

然后经过一个二层1D卷积层（2D卷积层资源开销太大），进行第一次特征变换。

再经过一个Dropout层，使得每次训练时随机由0.25的神经元失效，防止网络过拟合，提高鲁棒性。

再经过第二组1D卷积层，进行更抽象维度的特征变换（并同样附加Dropout层）。

最终通过输出池化层进行归一化输出。

为了保证模型可以在较低算力的微控制器上高效运行，本设计采用基于TensorFlow-lite框架的人工神经网络，在PC端将模型训练好后，再导入到终端设备上进行测试验证

2.1.5卷积神经网络分类器的核心代码

/\*\* \* @brief 在给定的脉冲（ei\_impulse\_handle\_t）上运行神经网络推理，并将结果存储在 ei\_impulse\_result\_t 结构体中

\* @param impulse 脉冲对象

\* @param fmatrix 特征矩阵

\* @param learn\_block\_index 学习块索引

\* @param input\_block\_ids 输入块 ID 数组

\* @param input\_block\_ids\_size 输入块 ID 数组大小

\* @param result 推理结果

\* @param config\_ptr 配置指针

\* @param debug 是否启用调试输出

\* @return EI\_IMPULSE\_ERROR 错误代码 \*/

EI\_IMPULSE\_ERROR run\_nn\_inference(const ei\_impulse\_t \*impulse,ei\_feature\_t \*fmatrix,uint32\_t learn\_block\_index, uint32\_t\* input\_block\_ids,uint32\_t input\_block\_ids\_size, ei\_impulse\_result\_t \*result,void \*config\_ptr,bool debug)

分类器核心入口

  for (size\_t i = 0; i < operators\_size; ++i) {

    TfLiteNode\* node =

    &(subgraph\_allocations\_[subgraph\_idx].node\_a nd\_registrations[i].node);

    const TfLiteRegistration\* registration = \_allocations\_[subgraph\_idx].node\_and\_registrati ons[i].registration;

   TFLITE\_DCHECK(registration->invoke);

   TfLiteStatus invoke\_status = registration->invoke(context\_, node);

   allocator\_->ResetTempAllocations();

    if (invoke\_status == kTfLiteError) {

      MicroPrintf("Node %s (number %d) failed to invoke with status %d",OpNameFromRegistration(registration), i, invoke\_status);

      return kTfLiteError;

    } else if (invoke\_status != kTfLiteOk) {

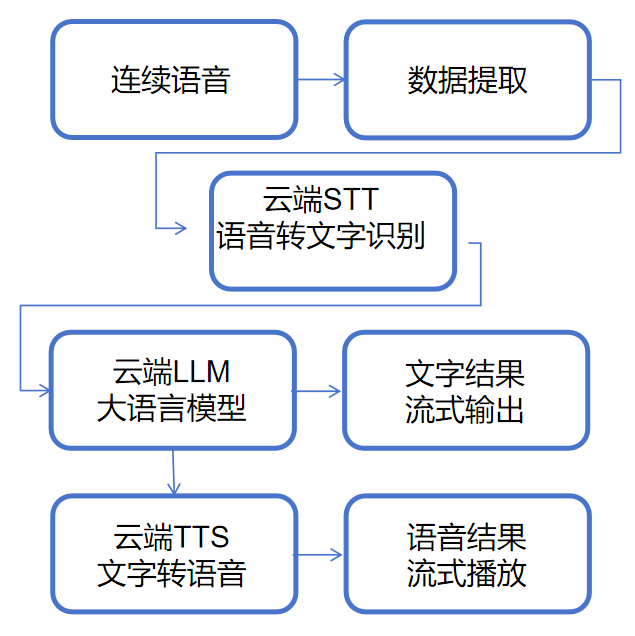
      return invoke\_status;

    }

  }

使用tensorflow lite为框架，基于前述结构生成前向传播网络，在实现上遍历子图的每一个操作，（对每层神经元进行运算），并最终得到结果输出。

2.2 云端模型的流式调用



系统流程图

2.2.1数据采集

uint8\_t \*pcm\_data = (uint8\_t\*)ps\_malloc(BUFFER\_SIZE);

//在psram中为音频数据开辟空间，避免占用内部ram

esp\_err\_t result = i2s\_read(I2S\_IN\_PORT, data, sizeof(data), &bytes\_read, portMAX\_DELAY);

memcpy(pcm\_data + recordingSize, data, bytes\_read);

//...中间省略

// 计算平均值

uint32\_t sum\_data = 0;

for (int i = 0; i < bytes\_read / 2; i++) {

   sum\_data += abs(data[i]);

  }

sum\_data = sum\_data / bytes\_read;

原始数据同样由I2S协议从麦克风中读出，并分次循环存入预设的数组内，期间每次对采集到的数据作均值处理，若发现数据均值远小于先前的数据均值，则判定为语音已结束，终止音频录制。

  int audio\_data\_len = audioDataSize \* sizeof(char) \* 1.4;

  unsigned char \*audioDataBase64 = (unsigned char \*)ps\_malloc(audio\_data\_len);

此后使用Base64格式对音频数据进行编码，上传到百度智能云提供的STT服务器中进行处理。

2.2.2云端STT（语音转文字）处理

String baiduSTT\_Get(String access\_token, uint8\_t \*audioData, int audioDataSize)

核心函数声明

将打包好的音频数据与预设请求头拼接形成json数据，使用POST方式向服务器请求响应

  if (httpCode > 0) {

    if (httpCode == HTTP\_CODE\_OK) {

      // 获取返回结果

      String response = http\_client.getString();

      Serial.println(response);

      // 从json中解析对应的result

      JsonDocument doc;

      DeserializationError error = deserializeJson(doc, response);

      if (error) {

        Serial.print("deserializeJson() failed: ");

        Serial.println(error.c\_str());

      }

      recognizedText = String(doc["result"][0]);

#ifdef LVUSED

      if (dispnow == 0) lv\_textarea\_set\_text(ui\_aichattext2, recognizedText.c\_str());

#endif

    }

此后判断返回值是否有效，有效则进行返回值解析

2.2.3接入云端大语言模型、流式响应

void SparkLLM\_Get(const char \*requestBody1, const char \*&label\_change)

核心函数声明

将STT结果以const char\*形式传入该函数，同样将其与预设请求头（要求使用流式返回）拼接形成json数据，使用POST方式向服务器请求响应



返回结果如上，分时连续多次提供，需要与服务器保持长连接直到最终接收到终止符[DONE]

      while (http.connected()) {

        response = stream->readStringUntil('\n');

        if (response.startsWith("data: ")) {

          response = response.substring(6);

          if (response.startsWith("[DONE]")) break;

          JsonDocument doc;

          DeserializationError error = deserializeJson(doc, response);

          label\_change = doc["choices"][0]["delta"]["content"];  // "今天是"

          char \*resp = (char \*)ps\_malloc(Chatlabelmaxlen);

          Serial.println(label\_change);  //

          LLMResponse = LLMResponse + string(label\_change);

          Serial.println(LLMResponse.c\_str());

        }

        vTaskDelay(10);

      }

函数的第二个形参const char \*&label\_change为返回值的取址，在数据接收完成前，每次读到新一组数据响应都预先传入该参数中，从而实现在屏幕上实时显示流式响应的结果而不需要等待数据全部接收完成后再统一显示

2.2.4LLM响应结果接入云端TTS（文字转语音）处理并流式返回

void baiduTTS\_Get(String access\_token, String text)

核心函数声明

bytesRead = stream->readBytes(buffer, ((size > sizeof(buffer)) ? sizeof(buffer) : size));

同样对LLM获得的返回值进行json格式化处理后上传至云端服务器，并使用stream的流式数据接收，每次只从服务器中获取512bit的音频数据，并实时播放，在保证音频连续播放的同时极大减小了微控制器的ram负担。

2.3 系统整合与设计

2.3.1系统概述

该硬件系统拥有，环境光传感器：APDS9960（I2C）以实现手势控制功能，为设备提供控制媒介；全向麦克风：MSM261；扬声器功放；MAX98357A。（I2S）；其他外设：陀螺仪：MPU6050。（I2C)；显示：1.69寸 TFT屏，ST7789V驱动（SPI）等丰富的外设资源，并已经集成到手表核心板上，因此，本系统将以智能语音手表为出发点进行设计和整合，其基础功能有：

（1）时钟（clock）（2）日历（calendar)

（3）系统设置（setting）（4）手电筒（light）

（5）在线AI语音问答

其智能控制功能有

（1）离线AI语音助手控制（关键词识别）

（2）在线AI语音助手控制（自主任务推断）

可以通过全局实时唤出的语音助手对系统进行控制，按指令要求跳转至对应的任务函数和UI界面中。

2.3.2系统实现

1. 软件系统采用FreeRTOS作为底层操作系统，充分利用ESP32的双核处理器，将UI任务开辟在CORE 1，其余任务开辟在CORE 0，互不干扰，减少由于程序阻塞造成的系统卡顿

xTaskCreatePinnedToCore(lvgl\_task, "lvgl\_task", 8192, NULL, 10, NULL, 1);

1. 使用LVGL+软件SPI+DMA双缓冲机制作为屏幕刷新机制，在有限资源下极大提高了屏幕刷新率和显示效果

void my\_disp\_flush(lv\_disp\_drv\_t \*disp, const lv\_area\_t \*area, lv\_color\_t \*color\_p) {

  if (dmaBufferSel) {

    dmaBufferPtr = buf\_3\_2;

  } else {

    dmaBufferPtr = buf\_3\_1;

  }

  dmaBufferSel = !dmaBufferSel;

  uint32\_t w = (area->x2 - area->x1 + 1);

  uint32\_t h = (area->y2 - area->y1 + 1);

  tft.startWrite();

  tft.setAddrWindow(area->x1, area->y1 + 20, w, h);

  tft.pushPixelsDMA((uint16\_t \*)&color\_p->full, w \* h);

  tft.endWrite();

  lv\_disp\_flush\_ready(disp);

}

1. 前后台任务与UI界面的调度与配合

本系统为规范化子程序设计，提供了一套清晰的子程序开发流程；

每个APP需要维护三个主题部分（以时钟程序为例）

1. 后台程序clock.c/.h



程序采用C++，编写，如果使用C语言，将会使很多函数（包含类对象的）无法被调用。

头文件使用c语言进行包装，以适配LVGL（基于C的图型库），

#ifndef \_CLOCK\_H

#define \_CLOCK\_H

#endif

编译说明，避免同一个文件重复编译造成错误

1. 前台显示 UI\_CLOCK.c



1. ui逻辑（补充在ui.c/.h中）

// SCREEN: ui\_CLOCK

void ui\_CLOCK\_screen\_init(void);

lv\_obj\_t \*ui\_CLOCK;

lv\_obj\_t \*ui\_Container3;

//void ui\_event\_ButtonDown1(lv\_event\_t \*e);

lv\_obj\_t \*ui\_ButtonDown1;

lv\_obj\_t \*ui\_ButtonUp1;

lv\_obj\_t \*ui\_ButtonCen1;

lv\_obj\_t \*ui\_Container5;

lv\_obj\_t \*ui\_HOUR;

lv\_obj\_t \*ui\_MIN;

lv\_obj\_t \*ui\_Panel2;

lv\_obj\_t \*ui\_Label3;

由系统桌面UI逻辑选择后进入子程序UI界面，进入时创建对应的task，与操作逻辑相关的一切程序在ui.c/h和UI\_CLOCK.c中声明、定义，后台程序调度（如时间数据的获取，WIFI进程等）则在后台task中实现

void ui\_event\_ButtonBACK(lv\_event\_t \*e);

lv\_obj\_t \*ui\_ButtonBACK;

dispnow = 1;

ui\_screen\_change(&ui\_CLOCK, LV\_SCR\_LOAD\_ANIM\_FADE\_ON, 200, 0, &ui\_CLOCK\_screen\_init);

Rtctaskcreate();

返回桌面时向task提供标志位信号，使其在内部自行清除task。（除非保留后台运行）

void ui\_event\_ButtonBACK(lv\_event\_t \*e) {

  lv\_event\_code\_t event\_code = lv\_event\_get\_code(e);

  if (event\_code == LV\_EVENT\_CLICKED) {

    dispnow = -1;

    \_ui\_screen\_delete(&ui\_CLOCK);

    \_ui\_screen\_change(&ui\_DESK, LV\_SCR\_LOAD\_ANIM\_FADE\_ON, 300, 0, &ui\_DESK\_screen\_init);

    Rtctaskdelete();

  }

}

基于上述的程序设计规范，可以有效避免单线程下的程序阻塞问题，同时可以避免由于内存分配和释放不合理导致的系统崩溃问题。

3 设计结果

3.1各功能实现展示

3.1.0操作逻辑介绍



本系统采用手势传感器检测（上下左右、按下）五个维度的操作，为主要的UI控制手段；同时右上角的实体按钮作为功能键使用。

**（1）手势控制：**

在所有UI界面上，可以通过上下滑动来切换选中的对象，朝手势传感器按下则表示确认操作；

在没有明确提供返回按钮的界面上，支持统一的右滑返回桌面菜单；

在具有多个按钮、滑块的界面，左右滑动被定义为关闭、开启该选项，并提供了明确的返回按钮。

（具体操作界面见后）

**（2）功能键**

功能键有短按和长按两个功能。

短按功能键将进入息屏休眠，长按功能键进入AI语音助手界面，据使用者的语音指令执行对应操作

3.1.1时钟(clock)

桌面上时钟图标 时钟程序界面

从桌面按下确认后进入时钟程序，由板载RTC芯片提供计时功能并由后台任务RTC\_task实时刷新时间数据，结合设置(setting)界面的手动、自动校时功能，可以保证时间的准确性。

3.1.2设置(setting)



桌面上设置图标 设置主界面 校时界面

从桌面按下确认后进入设置主界面，目前功能有：屏幕连续亮度调节、WIFI开启/关闭、时钟校准、网页端设置功能。

其中按下时间校准按钮后会跳转至子界面可选手动设置时间和联网校准时间。

网页端设置主要用于手表端不便于操作的部分，如WIFI信息录入等。

3.1.3日历(calendar)

桌面上日历图标 日历主界面

在日历主界面中，上下滑动可以切换显示的月份，右滑退出程序返回主界面

3.1.4 手电筒(light)

桌面上电筒图标 电筒主界面

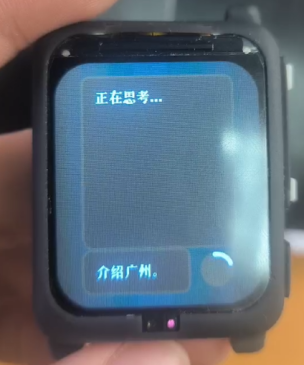
在电筒主界面中，可以通过上下滑动增加/降低亮度，右滑退出程序返回主界面

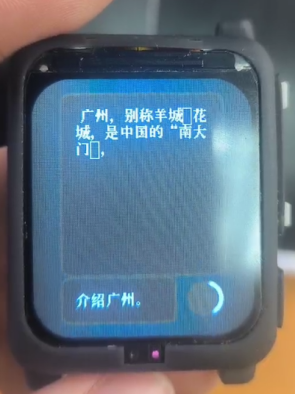
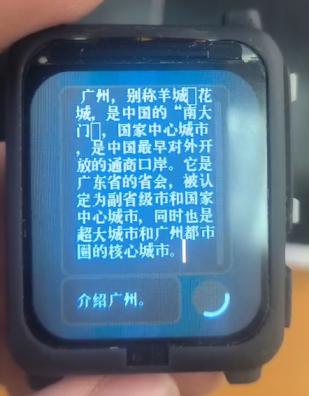
3.1.5 AI问答

桌面上语音问答图标 主界面

进入程序后，首先进行鉴权请求，如果网络未连接，或请求失败，将会提示“未连接网络”。

在主界面按下手势传感器后，将会进入问答过程： 

（1）语音获取 （2）语音分析 

（3）回答获取和语音播报（流式）

在问答期间可以随时再次按下手势传感器取消对话或右滑退出该程序。

3.2 AI语音助手

AI语音助手没有独立的桌面图标入口，而是使用长按功能键在任意界面唤出的方式。

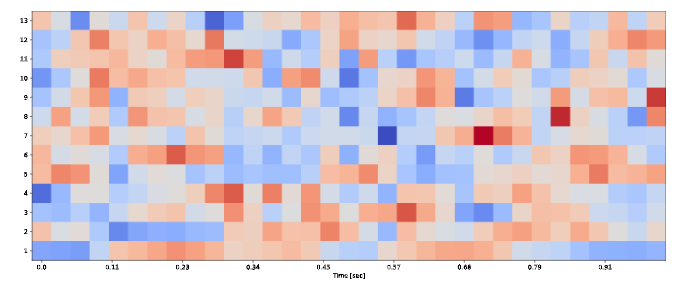
3.2.1 本地识别模型



**（1）功能的实现效果**

采用MFCC进行特征提取后，使用（主体）4层卷积神经网络进行分类，得到结果后，返回指定的标签值（如setting），在检测到label发生变化后，将自动跳转到对应的界面，实现简单的语音助手功能。

**（2）MFCC特征提取效果**



音频数据经过MFCC特征提取后的谱数据（13维）



特征向量示意图

可以看到，除了标签为noise的数据分布有一定的随机性，其余标签间较为分散，这为之后的分类提供了极大便利。

**（3）模型训练数据收集**

由于音频数据采集格式不同等原因，最好的数据收集方式是基于已有硬件直接获取数据。常见的数据集获取方式一般是将数据录制到SD卡上，再通过SD卡将数据导入到PC端。但由于此系统的引脚资源非常紧张，已经无法为SD卡（SPI）提供空间，因此本系统创新性地使用了串口实时数据传输的方式对音频数据进行采集，主要实现如下：

void loop() {

  uint32\_t sample\_size = (SAMPLE\_RATE \* SAMPLE\_BITS / 8) \* RECORD\_TIME\_SEC;

  uint8\_t \*rec\_buffer = (uint8\_t \*)ps\_malloc(sample\_size);

  if (rec\_buffer == NULL) {

    Serial.println("Failed to allocate memory!");

    return;

  }

Serial.write(signatureStart,strlen(signatureStart));

  // Start recording

  size\_t bytes\_read = 0;

  i2s\_read(I2S\_IN\_PORT, rec\_buffer, sample\_size, &bytes\_read, portMAX\_DELAY);

  if (bytes\_read == 0) {

    Serial.println("Record Failed!");

  } else {

    // Increase volume

    for (uint32\_t i = 0; i < bytes\_read; i += SAMPLE\_BITS / 8) {

      (\*(uint16\_t \*)(rec\_buffer + i)) <<= VOLUME\_GAIN;

    }

    Serial.write(signature, strlen(signature));

    // Send audio data through serial

    Serial.write(rec\_buffer, bytes\_read);

    //Serial.println("1 second of audio data sent");

  }

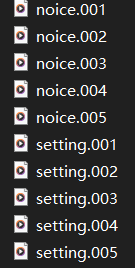
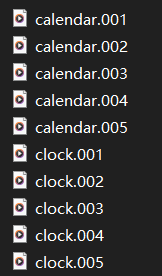
  // Free the memory

  free(rec\_buffer);

  delay(2000);  // Wait for 1 second before recording again

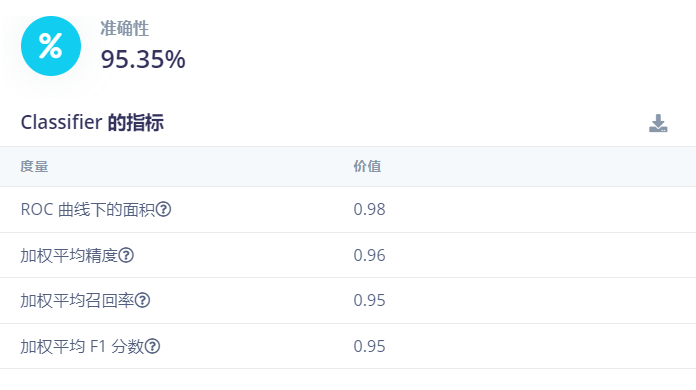
}

通过标识符向PC端提供开始和结束信号，PC端使用python脚本对数据进行处理并存储为wav格式，数据采集结果如下：



**（4）模型的准确度**

训练完成后，平均正确率达到95%以上



经过量化（int8）后，该神经网络仍有较好的性能。

3.2.2云端语音识别助手



通过类似于AI问答的流程，将采集到的语音信息转文字后让云端LLM模型自主判断，返回什么参数，本地解析返回之后执行对应指令，具有一定的任务自主规划能力。

prompt如下：

你需要根据我提供的信息进行推理并返回指定的数据：你现在可以控制一块智能手表，返回值只允许必须从以下范围取出：（setting，light，calendar，clock），请务必按要求根据下面的任务要求返回指定的字符串，不允许返回上述字符串以外的任何值，如calendar，日期相关请返回calendar，时刻相关返回clock以下是任务：

通过以上设定可以使系统按预期工作。

使用prompt直接接入LLM的好处是，不需要针对具体长期收集大量训练数据集进行微调，只需要逐步优化prompt提示词即可，降低使用者的调节门槛，同时减少对特定模型的依赖性，实现去耦的效果。

4 结论

本项目立项之初的目标是：

1.设计一款基于ESP32-PICO-V3-02核心的音频开发板。

2.硬件系统小型化：体积不超过43mm\*35mm\*12mm，以制作成穿戴式智能手表为目标

3.软件上，在MCU上使用FreeRTOS实现基本的操作系统，并设计合理的清晰的基础接口，将显示、控制、数据处理等部分解耦，便于更多人再次该系统基础上的二次开发

4.在MCU上部署基于TensorFlow-lite的语音识别模型

5.使用无线功能接入云端模型（流式传输）

6.模型部署完成后，利用语音识别的功能，实现一定程度的智能化操作。

7.软硬件系统具有长期稳定性，保证长期使用的稳定性而不是实验性产品。

最终的项目结果为：

1. ESP32-PICO-V3-02核心的音频开发板设计完成。
2. 最终硬件尺寸（含外壳）：43.9mm\*35.87mm\*10.99mm，除长度多了1mm之外，其他指标均符合要求，成品符合手表设备的一般尺寸
3. 基于FreeRTOS系统实现了对ESP32多核的编程，并提供了规范化的子程序设计标准，便于二次开发
4. 在MCU上成功部署了基于TensorFlow-lite的语音识别模型，理论准确定达到95%，响应时间在300ms以内
5. 成功使用流式传输的方式接入云端大语言模型
6. 通过prompt配置使得大语言模型有一定的智能化控制能力
7. 该系统具有长期稳定性，硬件成功制作3套硬件设备，且日常使用续航时间达到1天以上

综上，本项目采用手势传感器和实体按钮作为主要的UI控制手段，支持多种操作逻辑，如上下滑动切换选中对象、按下确认操作等。同时，系统集成了环境光传感器、全向麦克风、扬声器功放、陀螺仪和TFT显示屏等丰富的外设资源。

在软件设计方面，项目使用了基于梅尔倒谱系数(MFCC)的音频识别模型，并通过卷积神经网络(CNN)进行分类学习。此外，还实现了云端语音识别助手功能，通过流式传输接入云端大语言模型，并通过配置prompt使模型具备一定的智能化控制能力。

硬件设计包括了PCB布局、外壳设计以及硬件系统的集成，确保了系统的实用性和长期稳定性。硬件尺寸和功能均符合设计要求，能够实现穿戴式智能手表的制作目标。

最终，项目实现了设计目标，包括硬件和软件的整合，以及云端模型的流式调用，确保了系统的智能化和稳定性，为低算力、低功耗的端侧智能应用产品提供了系统思路和设计借鉴。

参 考 文 献：

1. 郭佳淇,张继通.基于MFCC和HMM的语音识别优化方法研究[J].电声技术,2024,48(10):83-85.DOI:10.16311/j.audioe.2024.10.024.
2. Preference for 20-40 ms window duration in speech analysis
3. [杨阳;陈永明;.声纹识别技术及其应用[J].电声技术,2007(02):46-47+51.](https://kns.cnki.net/kcms2/article/abstract?v=-xbefZa1Cdv5UvCjfog01pfrnGmrvW-1ojeagkwCyyfbtr2bnd7OmMw2fxyBM59IFs7HmKKmI-mmrthacTblfrB4r5jDvAGuOs4rfFxdZGHzsYXGT5ZgbiuJBtRTNDm221rDaG3FpJ8l8LWZ3Fl_YfCUNT2dh4AZsdB7k_tsLezvJMOaayuypEMKPMyTRLvp&uniplatform=NZKPT&language=CHS" \t "https://kns.cnki.net/reader/DSJS200702015)
4. 王刚.基于脉冲神经网络的音频信号分类算法研究[D].东南大学,2021.DOI:10.27014/d.cnki.gdnau.2021.004522.
5. 卢潇. 基于FPGA的声纹识别系统研究与实现[D].电子科技大学,2024.DOI:10.27005/d.cnki.gdzku.2024.001946.

1. Preference for 20-40 ms window duration in speech analysis [↑](#footnote-ref-0)
2. Preference for 20-40 ms window duration in speech analysis [↑](#footnote-ref-1)
3. [杨阳;陈永明;.声纹识别技术及其应用[J].电声技术,2007(02):46-47+51.](https://kns.cnki.net/kcms2/article/abstract?v=-xbefZa1Cdv5UvCjfog01pfrnGmrvW-1ojeagkwCyyfbtr2bnd7OmMw2fxyBM59IFs7HmKKmI-mmrthacTblfrB4r5jDvAGuOs4rfFxdZGHzsYXGT5ZgbiuJBtRTNDm221rDaG3FpJ8l8LWZ3Fl_YfCUNT2dh4AZsdB7k_tsLezvJMOaayuypEMKPMyTRLvp&uniplatform=NZKPT&language=CHS" \t "https://kns.cnki.net/reader/DSJS200702015) [↑](#footnote-ref-2)
4. 王刚.基于脉冲神经网络的音频信号分类算法研究[D].东南大学,2021.DOI:10.27014/d.cnki.gdnau.2021.004522. [↑](#footnote-ref-3)