基于声纹与人脸双重识别的考勤系统

——多媒体信息处理实验报告

课堂考勤是当代大学课堂不可缺失的一环，但现有的传统考勤方式却常常因为点名时间长、识别精度低遭到诟病。半自动化与自动化的考勤方式具有巨大的应用前景，但是单纯基于面部识别的考勤方法常常会面临因为相机位置不佳导致人脸大量遮挡、镜头分辨率低难以提取人脸特征以及摄像头视野不够难以涵盖全部场景等问题，导致识别的精确率降低、识别周期增长。而单纯基于声纹的检测方法则难以高精度应对高噪声环境下的识别问题。为了提高考勤效率并克服基于单纯人脸识别技术或声纹识别技术的考勤系统的弊端，我们团队开发了一种基于声纹与图像双重识别的考勤程序，制作了便于用户交互的UI界面，并实现了本地检测与实时检测两种方法。

由于小组作业常常会因为数据缺失难以训练模型，我们团队选择使用基于深度学习的预训练模型，积极探索了few-shot技术，并借助孪生网络增大了不同样本特征间的区分度，在较好地克服模型泛化能力不足问题的同时提高了检测精度。此外，为了实现声纹与图像双重识别的目标，我们团队选择先从声纹识别与图像识别两个子问题展开，并尝试了从结果层面融合与特征层面融合两种方式探索性能更佳的识别方法。

下面将一一介绍我们项目的各个部分所面临的问题以及我们经过方法调研最终选取的解决方案。

Web UI方面，所面临的主要问题是设计不同任务对应的用户交互界面以及与主函数的交互逻辑。具体而言，Web UI要实现三个目标：样本输入——用户输入待检测人员（如学生）的照片、声音与姓名至UI界面，UI界面将样本信息传给主函数并给出样本输入的信号，当主函数处理完成后将信号反馈给UI呈现给用户；本地视频检测——用户选择从本地上传视频或通过电脑内置的摄像头与麦克风进行录制，UI将视频传递给主函数进行处理，主函数将处理结果，即到场人员名单和缺勤人员名单以字典的形式返回给UI并输出给用户；实时检测——基本逻辑与本地视频检测相近，区别在于其需要处理流式的输入并以流式的形式将点到的音频输出到用户界面，这种异步逻辑的处理有一定的难度，体现了研究的创新性与复杂性。

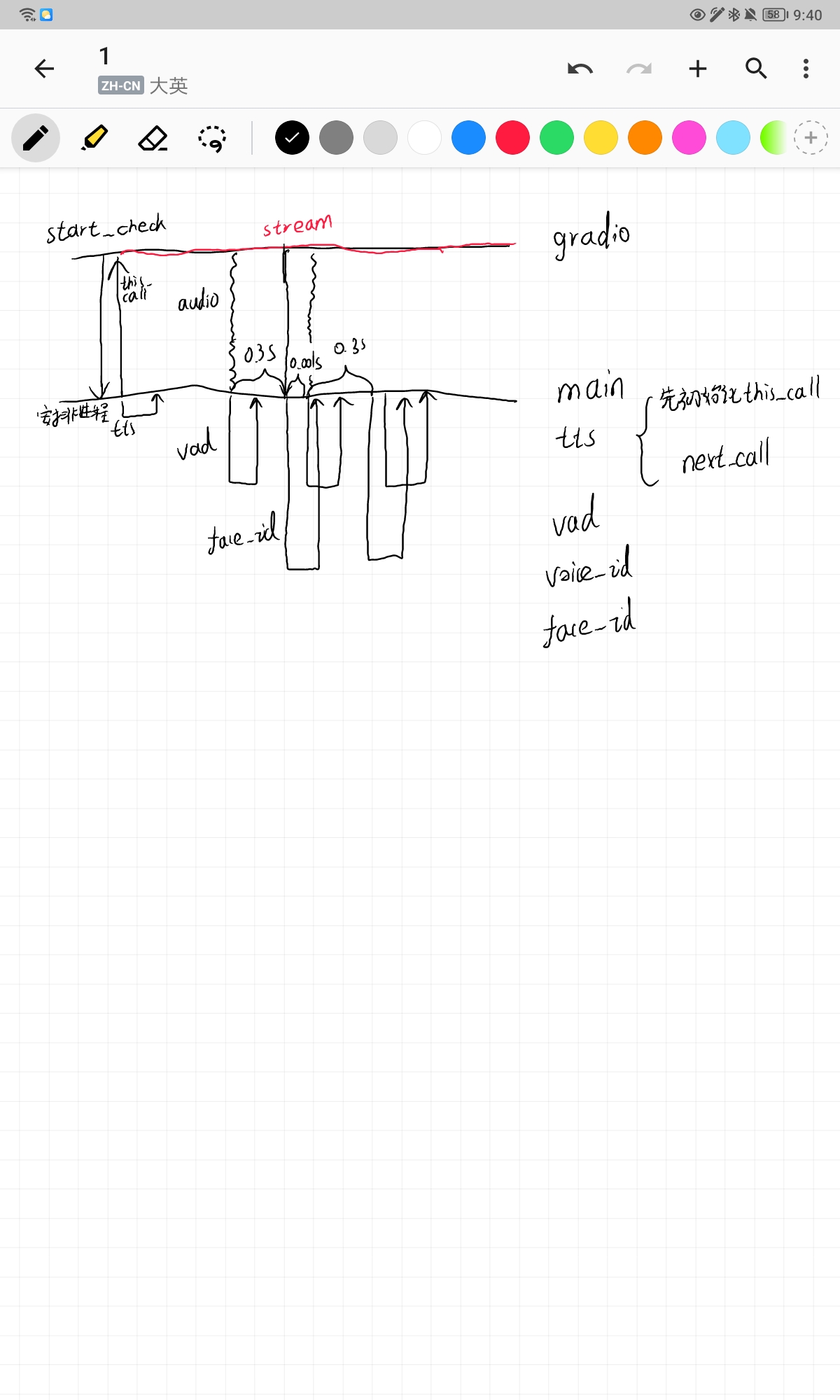
为了实现这个前端开发任务，我尝试了两种Web UI开发软件。其一是使用由饿了吗团队开发的基于Vue的Element UI开发软件，其优势是产品可以直接作为app嵌入移动设备使用，但难点是前置知识较多且与主程序交互较为复杂。其二是使用可以在python中导入的gradio 库进行开发。其优势在于与python程序交互方便并且易于上手，缺陷则在于难以作为app集成到移动设备。最终，为了接口易于交互以及调试方便，我们团队选择使用gradio作为UI开发的媒介。

而在gradio中，有两种Web UI开发方式：其一是使用gradio.Interface()函数作为基础，通过对其设置参数的方法开发设计页面；其二是使用gr.Blocks()，以块状、类似网页开发的方式进行UI开发。前者的好处在于功能集成度高，尤其对于流式输入输出的实现较为简单；但劣势也非常明显，难以处理多事件触发的逻辑。而后者的实现则较为复杂，并且函数的输入输出逻辑需要精心设计，但在实现多事件的触发与监听方面灵活性更大。为了实现样本输入、本地视频检测以及实时检测的复杂逻辑，我们选择使用gr.Blocks()作为UI设计的框架。

就技术方案而言，首先，我们确定了gradio的块层次代码框架，通过对各个组件的协调排版构建出整体的网页格局。其次，通过分页将各个任务的监听逻辑进行了较好的区分，降低了逻辑处理的复杂性。最后，通过监测按钮的状态、内置状态的变化以及流式输入的信息调度相关函数运转，最终将信息反馈给主函数进行处理，并接受返回值输出到用户界面。这样便实现了用户与程序的交互。

接下来通过介绍流式处理部分，来一窥整个项目所需的模型及主程序的逻辑：

首先，因为我们选择的是gradio框架，而gradio无法直接实现实时的流式效果，只能把音频流和图像流切分为无数多个小部分，分别传入函数运行，以达到近似实时的效果，这要求程序的效率足够高。又由于对输入数据的处理分为对图像的人脸检测与对音频的语音检测两部分，为避免其中一个线程的堵塞造成整个函数的堵塞，从而不符合我们预期的实时检测效果的情况，我们决定使用异步处理操作来提高整个程序的效率，以达到近似实时的效果。接下来将围绕下面的流程图详细介绍一下流式处理的逻辑过程：



根据我们设想的自动点名过程，程序首先根据点名表生成音频返回给gradio界面，gradio界面检测到有视频输入时自动播放点名。当被点名者答“到”时，vad模型识别到一轮问答结束，返回下一个人的点名音频，以此循环，直至点名表上的所有人都被点名。

在此基础上，我们整体的流式处理过程如下：程序启动时，将初始化生成第一个人的点到音频“this call”。当用户点击gradio界面的“开始检测”按钮，程序会监听此事件并返回“start\_check”的mode信号给处理流式的函数handle\_stream()，其会返回this call给gradio界面自动播放，并启动流式（stream）输入与处理。在流式输入模式下，将以0.3秒为一个chunk输入音频流，持续3秒；在此之后，将以0.01秒为一个chunk输入图像流，持续0.01秒，以此循环。在实际使用中，可以根据音频流与图像流的处理情况对这些参数进行调整，从而获得最佳的识别性能。

在流式处理中，每次主函数接收音频流，便会将音频传给vad和voice\_id模型，并为vad模型创建一个进程用于判断一轮问答是否结束。与此同时，检索该进程下有没有已完成的任务，如果有，并且任务返回的结果是“True”，便为voice\_id模型创造一个新进程用作提取音频特征向量，并生成next\_call用作点下一个人的名字；而如果结果是“False”,则维持其他功能的运行。而每次接受图像流时，主函数将图像传给face\_id模型，并为face\_id模型创造一个进程用作提取人脸特征向量。无论接受音频流还是图像流，最后都会检索提取特征向量的进程是否有已完成的任务，如果有，为database模型创建一个新进程用作识别特征向量，返回已经到场的名单，主程序将到场人员的姓名以字典的形式返回给gradio显示；如果没有，则结束，等待gradio的下次调用。

设想是美好的，现实是骨感的。在程序运行时却发现一些模型是不可序列化的，因此只能换种思路。考虑到python在GPU没有gil（全局解释器锁），所以利用多线程仍然可以在GPU上实现并行任务。同时为了避免任务被识别成协程而报错，我们通过asyncio库把同步函数包装成异步函数，最终程序成功运行。经实战检验，我们使用的模型都算轻量化的模型，实际运行起来并不耗多少时间，即使使用同步程序，也一样能够实现近似实时检测的效果。为了代码的简洁性，我们最终把异步过程做成了同步函数并实现了预期效果。

由于时间与经费的限制，我们的研究仍有进一步的拓展空间。首先，我们可以进一步收集点到数据，并训练语音“到”的关键词唤醒模型，提高程序的自动化程度。同时，可以尝试增量学习，同一样本只用采样一次，大大降低数据库的收集成本。此外，我们可以尝试将模型轻量化处理，使之可以集成到可移动设备上。