合肥工業大學

信息系统软件设计

报告

学　　院　　计算机与信息学院

专　业　班　级　　电子信息工程 23 级-3 班

学生姓名及学号　　丁志杰　2023212089

指　导　教　师　　张国富　苏兆品　李小红

课　题　名　称　　音视频传输认证系统

2025 年 1 月 7 日

一、课题概述

现在我们的生活越来越数字化，身份认证的使用也越来越频繁，只靠密码这一种方式验证身份，不仅容易被伪造破解，还受设备限制，问题越来越突出。好在计算机视觉和语音处理技术飞速发展，使得能用来做不用接触、安全性又高的身份验证。

本课题的目标，就是设计并做出一套远程身份认证系统，核心是靠人脸和声音这两种特征来检验身份。我会把音视频一起采集、TCP 可靠传输以及身份算法识别和后台 opengauss 数据库管理这些功能整合到一起，完成一个从采集端到后台服务端处理以及数据库一体化的完整系统平台。此外，本课题要求基于文献复现至少两种人脸识别技术和两种语音识别技术，并进行对比分析，说明各自的技术细节和优劣。而且将整个项目部署到华为鲲鹏开发板上，数据库也采用华为自研 opengauss 数据库，实现整个项目的国产化。因此本课题具有重要意义，既是对现代身份认证系统的一次创新性尝试又是个人为实现国产化贡献的一份力量。

二、课题任务

本课题目的是设计并实现一款人脸与音频多模态融合的身份认证系统，通过集成两种人脸识别技术+两种音频识别技术，并部署到华为鲲鹏开发板上，具体内容如下：

① 考虑到边缘嵌入式设备的有限算力，选择并实现两种兼具准确率以及快速性的人脸识别算法和音频识别算法。

② 规定采集端和处理端的通信协议。

③ 完成 TCP 发送端也就是采集端功能，采集用户的人脸，音频数据以及用户姓名等身份信息。

④ 完成 TCP 接收端也就是服务处理端，集成识别算法并对采集端传送回来的数据进行处理。

⑤ 美观简洁的 GUI 交互界面。

⑥ 将注册信息与识别记录存入 opengauss 数据库，实现信息的持久化存储和大量的访问需求。

⑦ 在鲲鹏开发板上部署并配置 opengauss 数据库。

⑧ 在鲲鹏开发板上调试完成采集设备（摄像头，麦克风）的驱动协议，采用模块化的方式方便上层功能代码调用。

⑨ 将整个项目部署在开发板上。

三、技术方案及关键问题

3.1 技术方案

3.1.1 数据采集与传输

在多模态识别系统中，采集端的采集质量及方案选择直接决定了整个系统的识别效果与相应速度，本课题针对 openEuler 环境下的鲲鹏开发板，设计了一套采集处理流程。

摄像头：

本课题选用了支持 1080P 的高清 USB 摄像头。虽然高清的原图能展示更丰富的面部细节特征，但直接传输原始图像会对板卡的 CPU 造成巨大的压力，同时增加 TCP 的传输延迟，因此采用如下策略：

① **硬件采集：**利用 V4L2 驱动把摄像头采集分辨率设为 **1920×1080。**

② **实时下采样：**为平衡传输延迟和识别精度，在采集完数据之后，将图像调制成 640×480 大小，以保证较低的传输延迟和符合后续人脸检测模型的输入。

③ **图像归一化：**摄像头采集图像的图像是 BGR 格式，为符合后续算法处理的输入要求，一律转化为 RGB 格式。

麦克风：

音频模块采用 Pyaudio 库与 ALSA 驱动进行交互，考虑实际麦克风的性能，同时也要满足声音识别的准确性，本课题对麦克风参数进行了精细化标定。

① **采样率：**设为 **48000HZ**，相比传统 16000HZ，高采样率能更好捕捉声音中的高频分量，增强识别伪造的能力。

② **声道和位深：**单声道采集，量化位深 16 位，为保证动态范围和方便后续直接转化成 Numpy 数组用于 MFCC 特征提取。

③ **缓冲区控制：**将 CHUNK 调制到 1024，减缓采集延迟，并结合 threading.Thread 异步录制，防止音视频同步过程中出现断续的问题。

TCP 通信：

利用 TCP 协议进行发送端与接收端之间的可靠数据传输，自定义发送端与接收端之间的通信协议，保证数据传输的稳定和可靠。将发送端采集到的人脸数据、操作类型，音频数据，姓名和学号封装成包发送给接收端，接收端接收数据解包后进行后续处理。

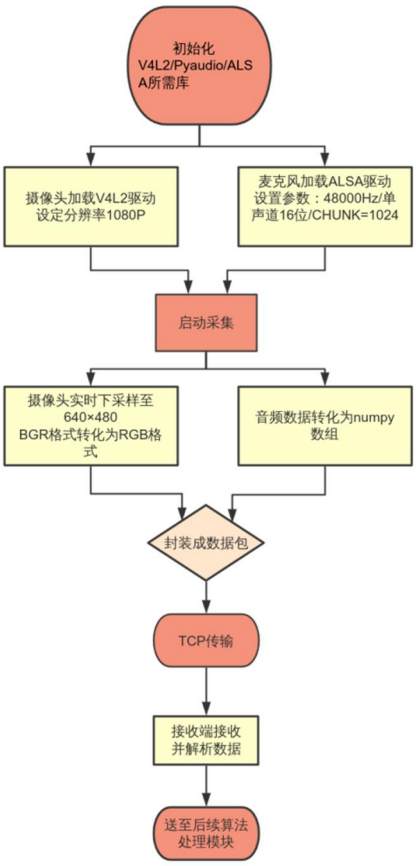


图 3.1 数据采集与发送模块流程

3.1.2 人脸识别技术

针对嵌入式设备有限性能的条件下，为保证快速性和准确性，本课题设计了一个“**传统纹理特征+深度学习特征**”的双重识别方案，兼顾**计算效率**与**识别精度**。

① **基于深度学习的 FaceNet 特征提取：**通过 Pytorch 框架搭建 inceptionResnetV1 架构，去掉原本的全连接层，使最后输出的是 512 维人脸特征向量，然后用人脸图片与标签训练这个模型，保存最优权重件。这个技术通过三元组损失学习，对各种环境都有极强的适应性。

② **基于 LBPH 的传统纹理特征识别：**采用 LBPH 算法。通过提取图像的局部纹理描述子，并利用其统计特性与数据库存储的进行快速匹配。该方案在小规模底库下具有极高的实时性。

③ **人脸区域检测与裁剪：**为确保识别算法输入数据的纯净度，系统首先对 640×480 图像进行空间上的特征定位，通过集成人脸识别器，实时计算人脸的边界框坐标，然后根据检测到的坐标，将人脸区域从原图中裁剪出来归一化到 160×160，然后再进行相应算法的处理。

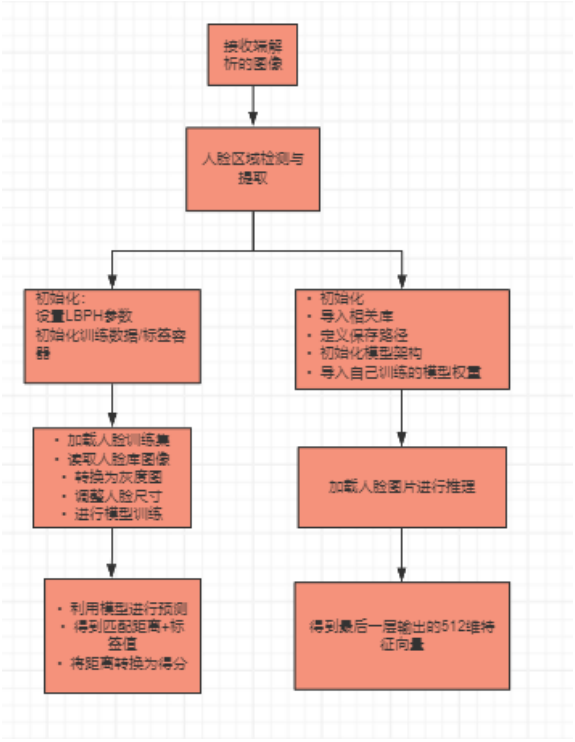


图 3.2 人脸处理流程图

3.1.3 音频识别技术

针对音频特征，本课题结合了前沿的深度学习嵌入与经典的时域统计特征，同人脸识别一样采用一种深度学习算法和一种机器学习算法相融合，确保在板端的识别速度与精度：

基于 Resemblyzer 的深度声纹嵌入：利用基于广义端到端损失训练的 VoiceEncoder 模型，提取能够表征说话人音色本质的 256 维特征向量。该技术通过 preprocess_wav 自动重采样并过滤静音，提取出特征具有高度的唯一性。

基于 MFCC 静态统计特征比对：作为第二种快速算法，系统提取语音的梅尔倒谱系数，并计算其均值与标准差作为身份描述子。该方法对语音长度要求较低，增强了系统在极端短语音场景下的可用性。

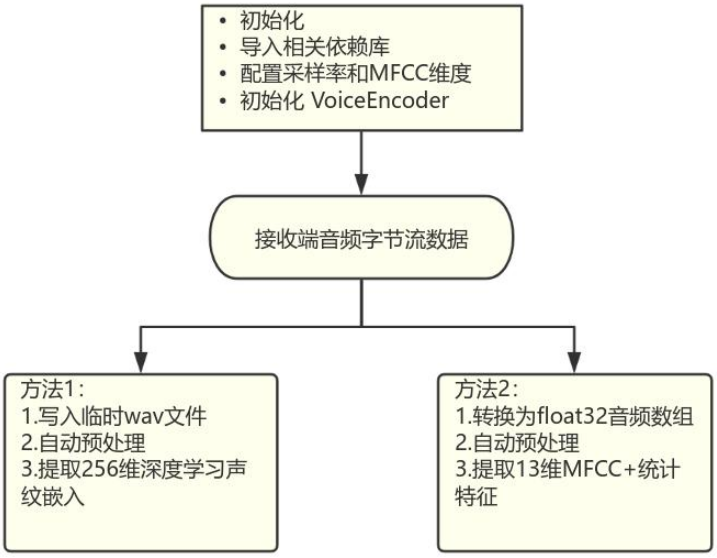


图 3.3 音频处理流程图

3.1.4 数据库管理

采用 openGauss 数据库，创建两个表：

① 注册记录：这个表中存储用户姓名，学号，以及上述四种算法处理过后提取出来的特征，以及存储的人脸图片路径和音频 wav 文件路径。

② 识别记录：这个表中存储识别出的用户姓名，学号，识别时间，人脸置信度，声音置信度，识别过程中抓拍和截取的人脸图片与音频文件路径。

3.1.4 多模态融合

模态内算法融合策略：

① **人脸识别融合**：人脸识别融合：通过深度学习算法优先进行人脸识别。FaceNET 提取的 512 维特征匹配分数大于 0.80，则认可 FaceNet 结果，分值小，则加入 LBPH 的纹理特征分数按 0.7:0.3 的比例进行加权，当两个算法判断为一个人时，得分乘 1.15 提高弱光识别通过率。通过融合策略，既能在性能过剩时保证较高的准确率，又能在系统繁忙时采用传统算法保证可靠性。

② **声音识别融合**：在声纹特征复杂的情况下，同样对深度学习 Resembzer 模型赋予了很高的信任权重（75%），MFCC 统计特征占 25%的比重，这样的比例分配能过滤掉环境的白噪音，并且只有当生物特征极度相似时才会发出识别信

号。

多模态认证逻辑：

在得到音频和人脸各自的得分和结果后，进行两个模态的融合，按照如下策略去融合：

① 若人脸和音频的得分都低于 0.5，则判定无此人。

② 若人脸和音频结果判定结果为同一人，则对得分进行加权平均。

③ 若人脸和音频结果判定结果不是用一个人，以得分高的识别结果为准。

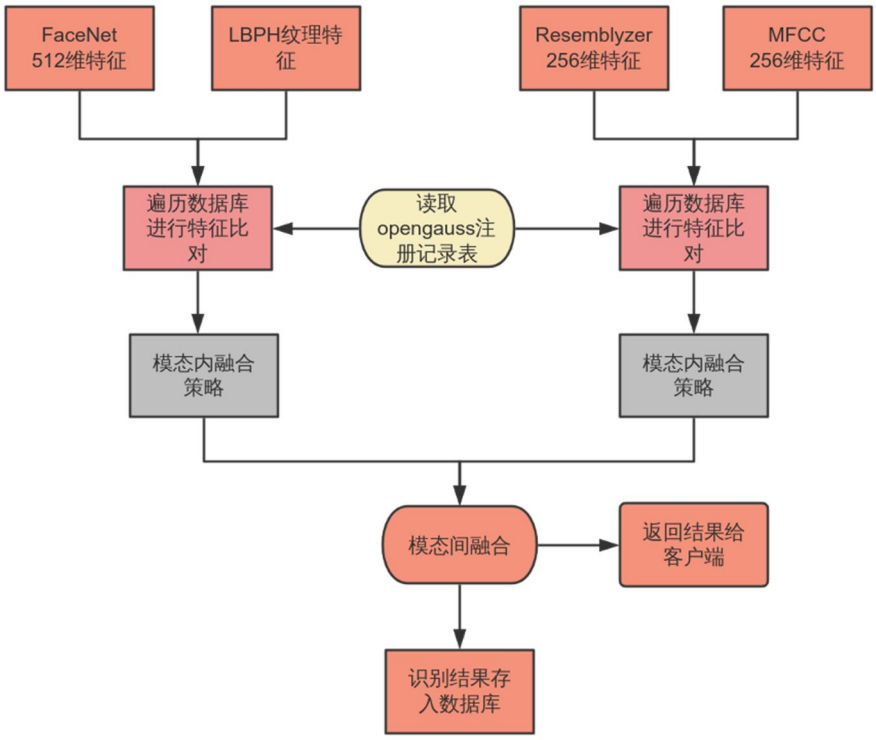


图 3.4 总体认证逻辑

3.1.5 GUI 界面设计

往往采集端需要和用户直接进行交互，所以设置一个简洁美观的 UI 界面非常重要，本课题采用 Pyqt5 实现 GUI 界面的设计，使用户能方便的完成注册信息和身份识别的操作。

3.2 拟解决的核心问题

3.2.1 识别准确率

本课题通过在采集端采集的数据进行适合准确的预处理策略并选择两种人脸识别算法+两种声音识别算法，并规定合适的融合策略， 确保识别结果准确和可靠。

3.2.2 识别速度

需要确保整套系统能在嵌入式设别上部署并正常运行，需要保证算法的复杂度不过高，处理流程高效。

3.2.3 嵌入式设备系统的兼容性

因为电脑的 windows 环境和板端的 openEuler 系统环境差异较大，架构也不一样，一个是 x64 一个是 arm 。很多代码需要修改移植，还要考虑 python 相关依赖在不同系统和架构下的兼容性。

3.2.4 用户操作便捷性

根据这套多模态身份认证系统的主要需求，设计简洁的交互界面，应用人脸图像和音频数据采集、身份核验和打卡等实际操作;通过界面直观展现所有功能，使用户看到的东西都知道就是 GUI，提高系统的实用性。

3.2.5 数据存储与管理问题

本课题采用 openGauss 数据库进行数据存储，确保数据有效的存储和查询，要设置合适的表格以及属性，还有高效的查询逻辑，确保数据能更快更准确的进行交互，在满足速率的前提下，为服务端的算法处理提供全力支持。

四．系统设计实现及测试

4.1 发送端设计实现及测试

4.1.1 发送端 UI 界面设计

采集端用户界面与交互部分使用 Pyqt5 开发，提供简洁舒适的用户体验。这个主要包括用户相关信息输入框，音频和视频录制按钮，摄像头实时预览等功能。

如下是创建 UI 界面的流程图：

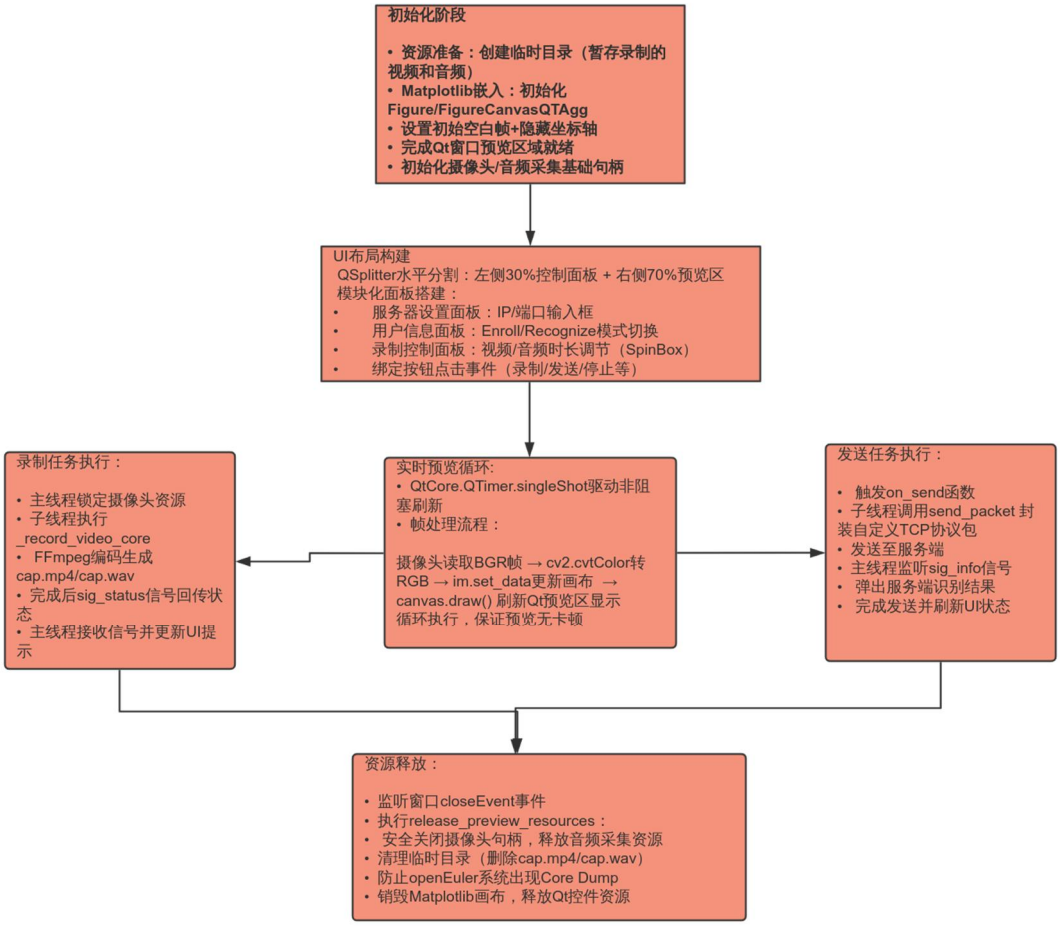


图 4.1.1 发送端 UI 界面设计流程图

本课题的采集端主要按照 Matplotlib 实时预览循环、多线程音视频录制以及自定义协议可靠传输展开。以下是对 SenderWindow 类及其相关核心功能函数的深度解析：

1. SenderWindow 类核心成员函数

- `__init__(self)`

功能：初始化发送端主窗口，设置窗口标题及 UI 规格（1000x600）。

核心逻辑：创建临时存储目录用于暂存音视频文件，初始化服务器配置（IP、端口）与 Matplotlib 预览画布。

- `_init_plot_canvas(self)`

功能：创建并初始化 Matplotlib 画布对象，代替不稳定的原生 OpenCV 预览窗口（解决开发时的 QT 冲突问题）。

关键点：初始化 FigureCanvasQTAgg 对象，嵌入 pyQt 布局，初始化 640x480 空白 RGB 帧。

- `_update_frame(self, fps)`

功能：驱动摄像头画面在 Matplotlib 画布上的非阻塞式实时刷新。

工作流程：

① 从 plt_cap 对象中读取当前 BGR 帧。

② 使用 CV2.CVtcolor 将色彩空间转换成 RGB，使用 IM.set_data 更新画布底层数据

③ 使用 QTimer.singleShot 定时发出下一帧更新，避免预览过程占用主线程资源。

- on_record_video(self)

功能：调度视频录制任务，处理硬件初始化与多线程并发冲突。

工作流程：

① 调用 release_preview_resources 释放之前预览占用的摄像头句柄，防止设备独占冲突。

② 强制指定 cv2.CAP_V4L2 驱动初始化 /dev/video0 设备，并配置 1080P/720P 兼容的 640x480 分辨率。

③ 启动子线程执行 _record_video_core 进行视频编码，同时主线程保持 Matplotlib 画面的持续渲染。

- on_send(self)

功能：实现多模态认证数据的协议打包与 TCP 可靠传输。

工作流程：

① **完整性校验：**通过 os.path.getsize 验证本地 MP4 视频与 WAV 音频文件的有效性。

② **协议封装：**根据当前选中的“录入”或“识别”模式构造包含 JSON 元数据（姓名、学号、模式）的头部信息。

③ **网络交互：**启动子线程运行 send_packet。该函数采用阻塞式套接字确保大容量音视频流完整送达，并利用信号机制将服务端返回的认证结果回传至 UI 界面。

图 4.1.2 采集端 UI 界面示意图

4.1.2 录制核心函数

数据采集部分主要通过连接在鲲鹏开发板的摄像头与麦克风实时获取人脸图像及声音数据。系统利用 OpenCV 库的 VideoCapture 类捕获视频流，并结合 PyAudio 库访问 ALSA 音频驱动。为确保采集过程的实时性与界面响应的稳定性，系统采用了多线程编程技术将耗时的录制任务与主 UI 线程分离。
具体流程图如下：[6]

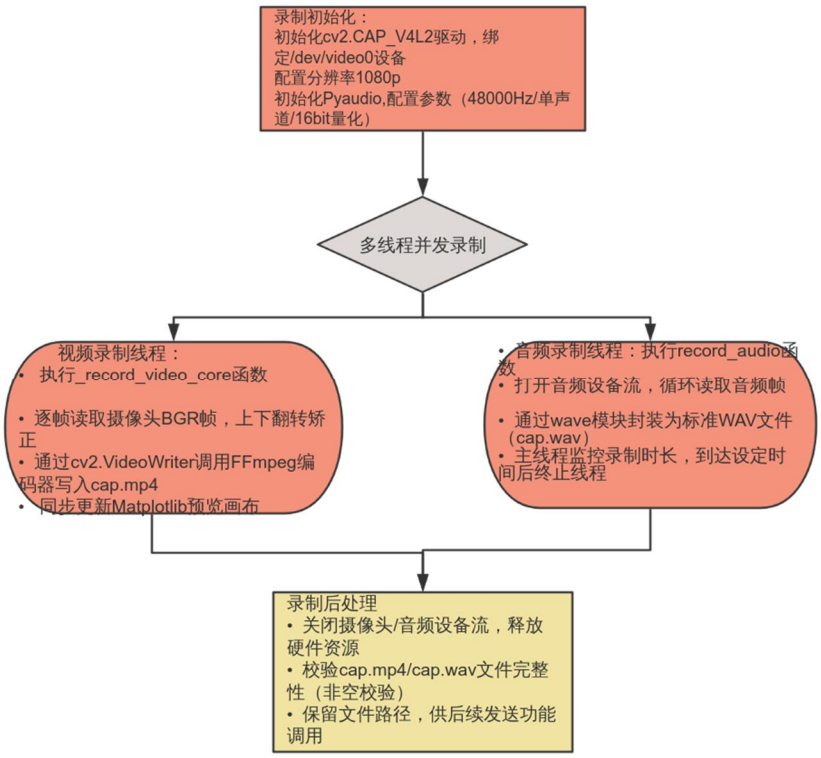


图 4.1.3　录制过程流程图

函数定义与代码逻辑描述：

- _update_frame 图像转换与预览函数

功能：从摄像头实时读取帧数据，转换为适合 Matplotlib 显示的格式并刷新界面。

步骤：

①　使用 self.plt_cap.read()函数从打开的摄像头进程中获得当前 BGR 帧。

②　利用 cv2.cvtColor 将 OpenCV 默认的 BGR 格式转换为 RGB 格式。

③　通过 self.im.set_data 更新画布底层像素数据，并调用 self.canvas.draw() 完成 PyQt 界面内的图像渲染。

- on_record_video 视频录制调度函数

功能：启动异步录制线程，执行视频流的本地暂存。

逻辑：在主线程锁定 cv2.CAP_V4L2 驱动的摄像头资源后，启动子线程调用 _record_video_core。该核心函数通过 cv2.VideoWriter 将每一帧写入 MP4 容器，同时通过回调机制保持预览画面的持续更新。

- record_audio 音频录制函数

① **功能：**适配 openEuler 硬件环境，录制高保真声纹数据。

② **参数设置：**针对特定硬件，内部强制设定采样率为 48000Hz，声道数为单声道，采样位数位 16bit。

③ **步骤：**通过 pyaudio.PyAudio 打开指定索引的音频设备，利用 stream.read 循环读取音频帧，最后通过 wave 模块封装为标准的 WAV 文件。

- _record_video_core(cap,out_path,seconds)

功能：执行视频流的底层编码与暂存。

参数：cap（已打开的摄像头对象）、out_path（文件输出路径）、seconds（录制时长）。

核心逻辑：共享主线程的摄像头对象，逐帧读取已矫正（上下翻转）的图像并推流至 FFmpeg 编码器，避免了硬件重复初始化带来的延迟。

运行结果展示：

图 4.1.4 录制过程实时显示

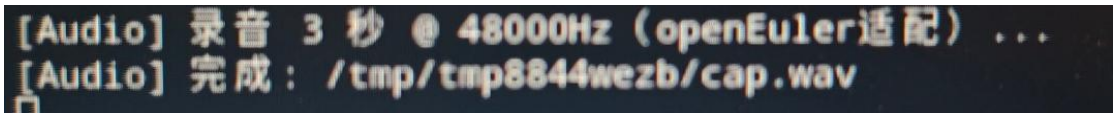


图 4.1.5 麦克风录制日志

图 4.1.6 摄像头录制日志

4.1.3 发送功能核心函数

在采集到音频和视频数据后，采集端将本地的 MP4、WAV、用户身份信息数据合并编译为数据包发送给接收端。课题是采用基于 TCP 协议的自定义应用层协议，采用多线程传输方式避免了 UI 的交互，作为发送任务调度器的功能是文件校验、header 建立、线程分发。

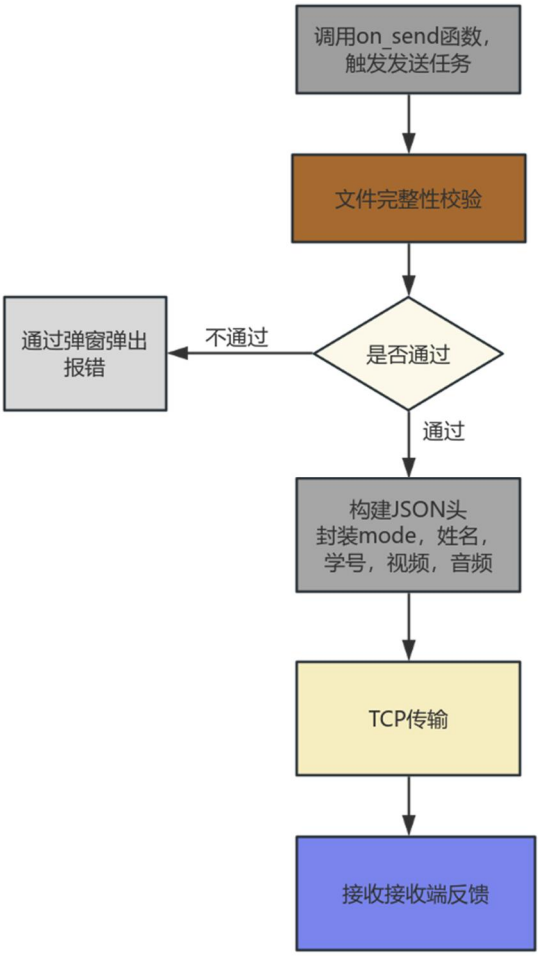


图 4.1.7 发送端发送逻辑

- on_send 数据打包逻辑函数

功能：作为发送任务的调度器，负责文件校验、Header 构建及线程分发。
步骤：

① **文件完整性校验：**文件完整性校验：通过 OS.Path.exists 和 OS.Path.getsize 查看 VPath、APath 是否有效，缺失或者大小为 0 时发出错误弹窗。

② **元数据封装：**根据 UI 选择的模式（enroll 或 recognize），构建包括 Mode、name 和 Student_id 的控制字典 header。

③ **二进制流读取：**：以'Rb'模式读取 MP4 和 WAV 文件内容作为 TCP 载荷（Payload）传输。

- send_packet 协议传输核心函数

功能：按照自定义的应用层协议将采集端数据发送给接收端。
步骤：

① **长度前缀封装：**输入 jsON 头长度，用 struct.pack 封装 4 字节的长度

前缀。

② **全量数据发送：**调用 sendall 将数据依次发送给“长度前缀+jsON 头+视频数据+音频数据”

③ **结果同步反馈：**函数停止后，接收端收到识别结果并将 JSON 格式的响应数据返回 GUI 线程展示。

- run 异步通信线程

功能：在后台执行 send_packet 任务，通过 pyqtSignal 跨线程更新界面状态。

效果：在数据发送期间，通过 setEnabled(False) 锁定发送按钮，防止重复提交；在收到结果后，通过 sig_info 弹出包含姓名、学号及识别状态的成功提示框。

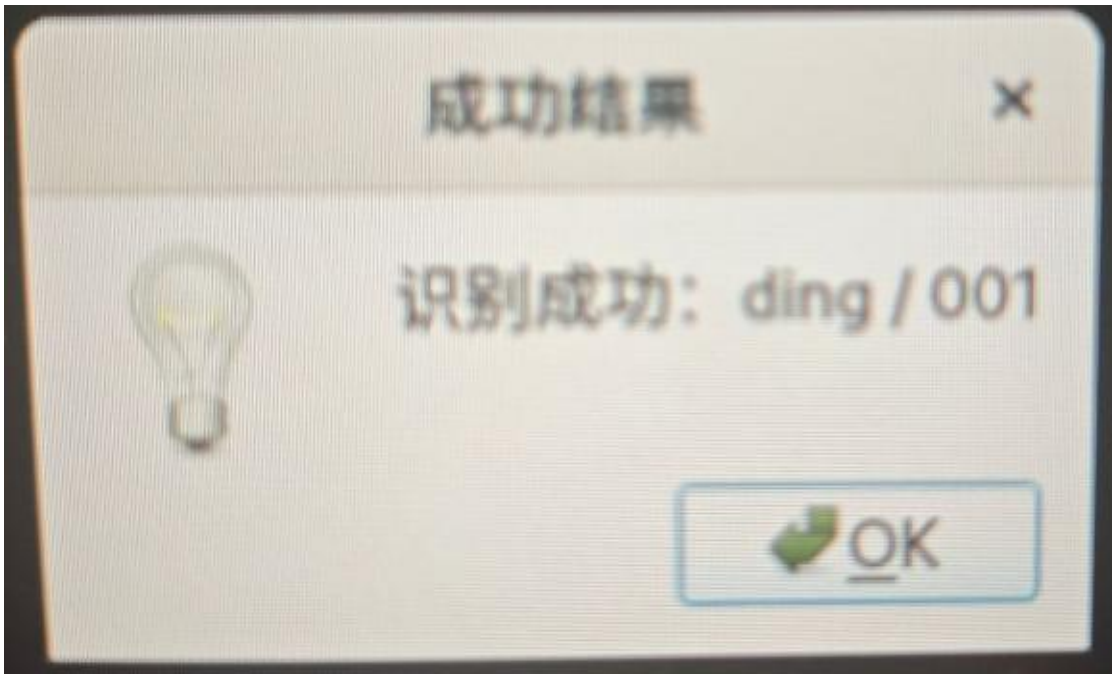


图 4.1.8 采集端接收到服务端的处理结果

4.2 接收端设计实现及调试

4.2.1 接收端整体实现

在完成发送端的实现后，系统已经具备采集音视频能力以及将相关信息发送的功能，那么下一步就要进行接收端的编写，接收端主要集成调用封装好的算法模块对发送端发过来的音视频进行处理识别，并完成与数据库间的交互，主要函数如下：

- handle_enroll(header, video_bytes, audio_bytes)

功能：执行用户身份的首次录入。

工作流程：

① **图像预处理：**保存临时视频文件，调用 extract_face_frames 提取人脸关键帧，并使用 FaceNet 模型生成 512 维特征向量。

② **音频预处理：**保存 .wav 音频，并使用 Resemblyzer 提取深度声纹嵌入与 MFCC 静态特征。

③ **持久化存储：**调用 upsert_person_full_path 将姓名、学号、人脸音频特征及文件路径同步写入 openGauss 数据库，并重置 LBPH 更新标记。

- handle_recognize(header, video_bytes, audio_bytes)

功能：实现用户身份的识别。

工作流程：

① **特征对比：**从采集端采集回来的音视频数据加载算法模块进行识别得到相应的特征，分别与数据库中计算人脸与声音的得分。

② **执行融合：**调用融合函数（按照上面提到的融合策略）计算融合，并将识别结果返回给发送端，并调用 insert_record_paths 将本次识别的截取文件路径及得分存入 recognition_records 表中。

- update_lbph_model(persons)

功能：实现 LBPH 传统识别算法的动态热更新机制。

逻辑：遍历现在数据库中所有注册过的人员的照片路径，使用 cv2.imread 加载图像并调用 train_lbph 重新训练分类器，确保整个系统具备实时识别的的能力。

- extract_face_frames(video_path, max_frames)

功能：从接收到的图像种提取裁剪出人脸区域。

逻辑：利用 np.linspace 在视频时间维度上均匀采样，并定位人脸区域并执行裁剪。

图 4.2.1 提取人脸所在帧及所在区域的裁剪结果

- predict_voice_cosine(curr_vec, enroll_vecs)

功能：实现声音的特征比对。

核心算法：使用用 np.dot 计算音频特征向量的余弦相似度。通过遍历数据库底库字典，寻找到比对的得分最高的人。

- fuse_face_predictions(lbph_res, dl_res)

功能：实现人脸的特征比对。

策略：将算法识别输出与数据库中存入的注册信息进行比对，通过遍历数据库底库字典，寻找得分最高的人。

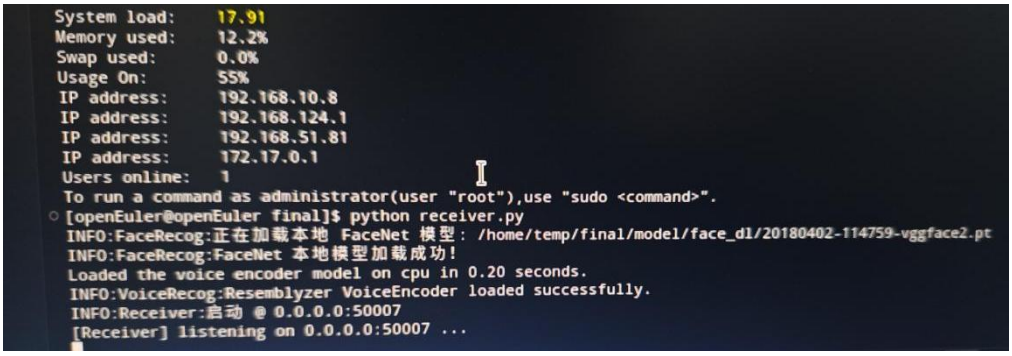


图 4.2.2 接收端初始化过程展示

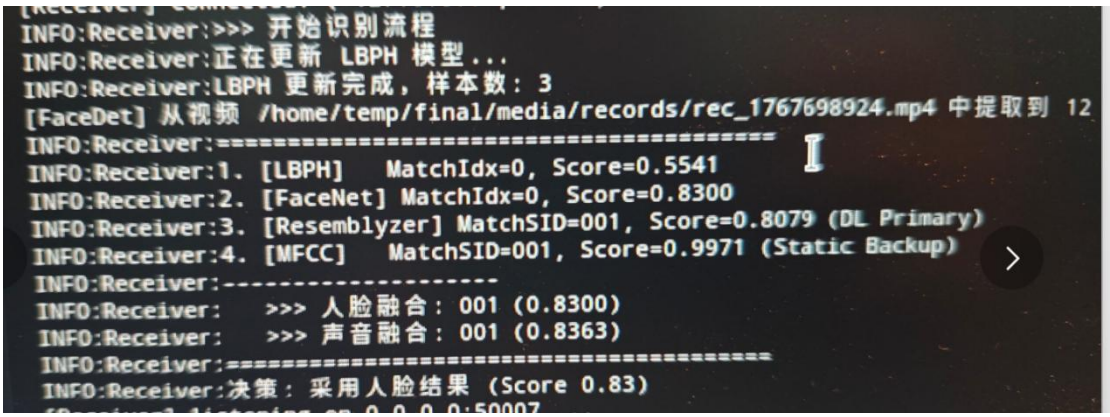


图 4.2.3 识别流程展示

4.2.2 接收端与数据库之间的操作

本系统采用华为openGauss数据库对数据进行存储，完成对用户信息的存储，系统通过 psycopg2 作为驱动与数据库之间进行交互。[7]

核心表结构设计 系统在 init_db 阶段通过我提前写好的 SCHEMA_SQL 脚本自动建立两张表，分别是如下两张：

① **人员特征表：**存储用户的主要信息如：姓名、学号以及声音和人脸特征向量（voice_embed, face_embed）。其中 student_id 设为唯一约束。

② **识别记录表：**记录每次任务的详细信息。包含识别模式（mode）、匹配得分（face_score, voice_score）以及任务书里要求的音视频存放路径（video_path, audio_path）。

关键重要函数说明：

- upsert_person_full_path 人员信息同步函数

功能：在录入模式下实现特征数据的更新和插入。

步骤：

① **尝试更新：**首先根据 student_id 尝试修改已存在的人员及相关文件存放路径。

② **兼容判定：**通过检测 cur.rowcount，如果受影响行数为 0，则说明这个人员为首次录入。

③ **执行插入：**在更新失败的情况下执行新记录插入，确保了在不同版本环境下的语法兼容性。

- list_persons 底库加载函数

功能：在识别任务之前，提取数据库中的存储记录。

逻辑：通过 RealDictCursor 将查询结果转换成字典，便于识别算法模块快速反序列化 JSON 格式的特征向量，以此来实现高效的对比。

- insert_record_paths 认证结果记录函数

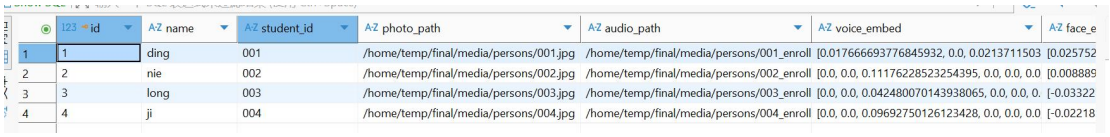
功能：完成题目要求的“识别结果入库”与“多模态数据索引”功能。

步骤：

① 接收模态识别算法模块计算的加权融合得分。

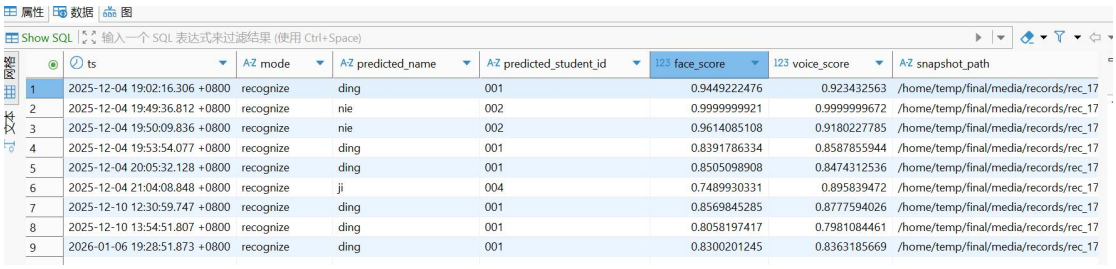
② 执行 SQL 插入命令，将这一次认证抓拍的图片路径及音频采样路径同步保存至 recognition_records 表中。

③ 通过 conn.commit() 提交记录。



	id	name	student_id	photo_path	audio_path	voice_embed	face_e
1	1	ding	001	/home/temp/final/media/persons/001.jpg	/home/temp/final/media/persons/001_enroll	[0.017666693776845932, 0.0, 0.0213711503	[0.025752
2	2	nie	002	/home/temp/final/media/persons/002.jpg	/home/temp/final/media/persons/002_enroll	[0.0, 0.0, 0.11176228523254395, 0.0, 0.0, 0.0	[0.008889
3	3	long	003	/home/temp/final/media/persons/003.jpg	/home/temp/final/media/persons/003_enroll	[0.0, 0.0, 0.042480070143938065, 0.0, 0.0, 0.	[-0.03322
4	4	ji	004	/home/temp/final/media/persons/004.jpg	/home/temp/final/media/persons/004_enroll	[0.0, 0.0, 0.0969275012623428, 0.0, 0.0, 0.0	[-0.02218

图 4.2.4 persons 表展示



	ts	mode	predicted_name	predicted_student_id	face_score	voice_score	snapshot_path
1	2025-12-04 19:02:16.306 +0800	recognize	ding	001	0.9449222476	0.923432563	/home/temp/final/media/records/rec_17
2	2025-12-04 19:49:36.812 +0800	recognize	nie	002	0.9999999921	0.9999999672	/home/temp/final/media/records/rec_17
3	2025-12-04 19:50:09.836 +0800	recognize	nie	002	0.9614085108	0.9180227785	/home/temp/final/media/records/rec_17
4	2025-12-04 19:53:54.077 +0800	recognize	ding	001	0.8391786334	0.8587855944	/home/temp/final/media/records/rec_17
5	2025-12-04 20:05:32.128 +0800	recognize	ding	001	0.8505098908	0.8474312536	/home/temp/final/media/records/rec_17
6	2025-12-04 21:04:08.848 +0800	recognize	ji	004	0.7489930331	0.895839472	/home/temp/final/media/records/rec_17
7	2025-12-10 12:30:59.747 +0800	recognize	ding	001	0.8569845285	0.8777594026	/home/temp/final/media/records/rec_17
8	2025-12-10 13:54:51.807 +0800	recognize	ding	001	0.8058197417	0.7981084461	/home/temp/final/media/records/rec_17
9	2026-01-06 19:28:51.873 +0800	recognize	ding	001	0.8300201245	0.8363185669	/home/temp/final/media/records/rec_17

图 4.2.5 recognition_records 表展示

4.3 人脸识别算法与音频识别算法实现

4.3.1 人脸识别算法

方法 1：深度学习方法：

Inception-ResNet 是利用卷积神经网络训练复杂人脸识别模型,通过具有多维度的卷积，和 RESNet 的残差连接结合能提高识别的准确率。利用多层卷积和池化的方式可以提取人脸图像的高级语义特征，具有非常好的特征表达能力[1]。

训练数据是开源数据集 CASIA-WebFace 数据集总共 47,958 张图片，自己搭建模型架构 15 层，其中包含卷积层、批量归一化层、激活函数、最大池化层及输出 512 维的特征向量全连接层。

1.InceptionResne 模型架构子模块

- BasicConv2d (卷积块)

  功能：构建包含卷积（Conv）、批量归一化（BN）与线性激活（ReLU）的复合层。

  作用：当作网络的基础模块。

- Block35 (Inception-A 模块)[2]

  功能：35x35 特征图像的高级特征提取。

  逻辑：3 个并行分支（1x1 卷积、3x3 卷积组合、双 3x3 卷积组合）获取多尺度细节，利用残差缩放因子与原始输入加总得到细节。

- Block17 (Inception-B 模块)[2]

  功能：处理 17x17 尺寸特征图的中层特征提取。

  逻辑：采用非对称卷积（1x7 和 7x1 卷积核），既能减少参数量也可以增加网络的感受野。

- Block8 (Inception-C 模块)[2]

  功能：用来对 8x8 尺寸的特征图提取深层语义特征。

  逻辑：在网络末端进行高维特征汇聚，通过 no_relu 参数灵活控制残差叠加后的非线性处理，确保 512 维特征嵌入的区分度。

- __init__（）

  功能：整体架构初始化。

  逻辑：组合包含卷积、池化在内的 Stem 初始层，主要用途是快速降低输入维度，然后通过串联模块达到深度特征演化的目的，然后映射为归一化的嵌入向量。

- forward(self, x)

  功能：定义张量的前向传播。

2.模型训练代码

- train_face_model（） 模型训练模块
- transforms.Compose（） (预处理流水线)

  功能：对 47,958 张原始图像进行预处理。

  作用：将图像统一下采样变成 160x160 像素，并通过均值与标准差归一化，消除一些干扰。

  datasets.ImageFolder（）

  功能：自动扫描并标签映射数据

  作用：递归地扫描./data/face_dataset 下的所有子文件夹，将子文件夹名转化为训练标签，方便管理大样本。

- nn.TripletMarginLoss(margin=1.0)

  功能：定义三元组边际损失函数。

逻辑：faceNet 模型算法是拉近锚点与正样本距离、推开锚点与负样本距离的，在 512 维特征空间具有强的判别力

$$L = \sum_i^N [\,||f(x_i^a) - f(x_i^p)||_2^2 - ||f(x_i^a) - f(x_i^n)||_2^2 + \alpha]_+$$

（4.1）

其中 x_i^a 为锚点图像，x_i^p 为正样本，x_i^n 为负样本，α 为间隔。

- optimizer.zero_grad() 与 loss.backward()
  功能：梯度清零和反向传播。
  逻辑：在每个 Batch 迭代中清除旧梯度，计算当前的损失对 15 层网络参数的偏导数，为后续参数更新提供方向。
- optimizer.step()
  功能：执行权重更新。
  逻辑：利用 Adam 优化算法动态调整 15 层网络内部的所有权重参数，驱动 Loss 值稳步下降。
- torch.save(model.state_dict(), ...)
  功能：保存模型权重。

作用：将训练效果最好的参数保存为 final_model.pt 文件，供接收端在 handle_enroll 或 handle_recognize 任务中随时调用。

方法 2：传统方法

LBPH 技术是人脸识别领域应用较广泛的算法，局部二值模式是提取人脸图像局部纹理特征的核心算子，它能够精准地捕捉到人脸区域的细节纹理信息，该算法以其算力低、准确率高的特点而广泛运用于人脸识别领域

LBP 原理：LBP 的基础运算在 3×3 像素窗口中展开：以窗口中心像素点的灰度值为基准，依次将周围的 8 个像素灰度值与这个中心阈值相比较，如果周围像素的灰度值大于等于中心灰度值，则这个位置记为 1，否则记为 0。[5]

$$LBP(x_c, y_c) = \sum_{p=1}^{8} s\,(I(p) - I(c)) * 2^p \qquad (4.2)$$

度值，公式中的 s(x)为符号函数，定义如下：

$$s(x) = \begin{cases} 1, & x \geq 0 \\ 0, & otherwise \end{cases} \qquad (4.3)$$

LBP 特性：

① 记录中心像素与邻域像素的灰度差，而不是像素本身的绝对灰度值；

② 当光照变化导致图像整体灰度增减时，LBP 特征的变化程度相对较小，所以在光照波动的场景下适应性更强。

4.3.2 音频识别算法

方法 1：

Resemblyzer 模型架构复现：

音频识别模块基于 resembzer 模型架构，采用深度循环神经网络提取声纹特征，通过 Ge2E 来优化模型。下面介绍搭建与训练部分关键函数：[3]

VoiceEncoder 模型架构模块

- __init__(self, device)

  功能：初始化声纹编码器神经网络架构。

  关键组件：定义 3 层叠加 LSTM 单元用于获取变长语音信号的长距离时序相关性。In_size=40 匹配预处理得到的 Mel 频谱，采用 hidden_size=256 隐藏层高维特征建模。

- forward(self, utterances)

  功能：定义音频数据的前向特征提取流向。

  逻辑步步骤：

  ① me 频谱序列进入 LSTM 循环网络中提取包含整段语音信息最后一层隐藏状态

  ② 在 Projection 线性层中将特征映射到 256 维 Embedding 空间；

  ③ 对输出的声纹向量进行 L2 归一化，使得最终的声纹向量出现在一个超球面上，从而为后面基于余弦相似度的身份比对提供数据基础。

GE2ELoss 与训练调度模块：

- GE2ELoss.forward(self, embeddings)

  功能：广义端到端损失计算，是声纹区分度最基础的算法。

  核心逻辑：

  中心计算：使用 Torch.Mean 计算每一个说话人在当前批次中的特征中心。

  相似度缩放：通过可学习参数 w 和 b 对余弦相似度矩阵进行线性变换（$S=W\cdot\cos(\theta)+b$），强化同一个人的声纹的凝聚和加大异人声纹的距离。

  损失优化：使得某个话语和某一个说话人中心之间的相似度最大化，学到非常具有辨识度的声纹表征。

- train_voice_encoder(data_loader, model, device)

  功能：驱动声纹编码器参数迭代和权重优化。

  关键操作：

  利用 Adam 优化器更新模型参数和损失函数的缩放因子，在 500 个 EPoch 的训练中，通过 embeddings.View 动态变化张量形状以达到 GE2E 的矩阵运算要求，调用 torch.NN.utils.clip_Grad_norm_将梯度范围限制在 3 以内，有效解决深层 LSTM 在长序列音频训练时较为容易发生梯度爆炸问题，保证训练数值稳定。

方法 2：

传统的音频识别算法有 MFCC-GMM 联合建模法，采用梅尔倒谱系数计算语音的频谱包络，高斯混合模型对特征分布进行拟合。这种联合架构的特点是能够从

统计的角度对说话人的音色特征进行刻画，准确率和计算速度较高[4]。

算法逻辑：该算法的实现分为特征解析和统计建模两个核心阶段：

MFCC 特征提取：模拟人耳非线性频率，将原始语音信号转换成 13 阶倒谱系数序列，将时域波形缩放为能够对应声道形状的特征参数集。[8]

GMM 概率建模：通过多个高斯分量加权组合得到 MFCC 特征在维度空间的分布密度。其中主要概率预测公式如下：

$$P(X|\lambda) = \sum_{i=1}^{k} \omega_i \cdot \mathcal{N}(X|\mu_i, \Sigma_i) \quad (4.4)$$

其中，X 是提取的 MFCC 特征向量，k 为高斯分量数（本项目设为 8），$\omega_i, \mu_i, \Sigma_i$ 分别代表每个高斯成分的权重、均值和方差，共同组成了说话人的声纹指纹。

关键函数介绍：

- train_gmm(wav, sr)

功能：搭建说话人的声纹模型底库。

逻辑：首先调用 Librosa.feature.mfcc 获取特征矩阵 X，然后送入 GaussianMixture 进行 EM 算法训练，训练后保存均值和方差参数，并以文件的形式保存为模型，数字化存储音色。

- calculate_gmm_score(gmm, wav, sr)

功能：计算待测音频与底库模型的匹配似然度。

逻辑：对要测试的音频提取 MFCC 特征，然后计算在已训练 GMM 模型下的对数似然得分。该得分直接反映了当前语音属于该用户的概率，得分越高，身份匹配度越高。

- fuse_voice_predictions( )

功能：完成统计特征算法和深度学习特征算法的加权融合。

方法：将 GMM 似然得分归一化，并与 Resemblezer 深度向量比对结果进行 0.4:0.6 的加权融合，系统能够满足不同语速、音量情况下的鲁棒性。

4.4 技术对比情况

表 4.1 四种算法对比表

技术类别	LBPH 技术	Inception-ResNet	MFCC-GMM	Resemblyzer
识别准确率	86.20%	**98.60%**	82.50%	**96.80%**
特征维度	局部直方图	512 维 嵌入向量	26 维 统计向量	256 维 嵌入向量
训练需求	无需大规模预训练，即录即用	需要大量数据集进行训练	少量样本即可完成 GMM 拟合	需要大量声纹语料构建编码器
适用范围	小规模数据，适应性强	适合大规模以及复杂情况人脸识别	适合超短语音片段的快速验证	适合环境噪音干扰较大的声纹识别

技术类别	LBPH 技术	Inception-ResNet	MFCC-GMM	Resemblyzer
计算性能	算力需求极低	对 CPU/GPU 算力有一定要求	运算很快，适合嵌入式端部署	推理时间略长，适合服务端处理

五．课程设计总结与心得体会

5.1 设计总结

任务完成情况：本次课程设计，我完成了一套人脸与声音多模态融合的身份识别系统，复现了两种人脸识别算法与两种音频识别算法，并部署到嵌入式设备上，完成了基于 openEuler 22.03 系统的多模态身份认证系统整个开发流程。硬件方面，系统能正常调用 CAP_V4L2 视频驱动和 PyAudio 音频驱动，把音视频数据同步采进来，画质和音质都能能满足处理要求；软件上，用 PyQt5，做了一个简洁的 qt 界面，让系统与用户交互更便捷。

创新点：

① 自己搭了深度学习模型的底层架构，不仅仅是调用官方的模型以及与预训练权重，而且自己学习了一下相关模型的底层架构，并选取合适的数据集进行自定义的训练。

② 此外创新型的设计了模态间和模态内的融合策略，使得识别结果能更稳定准确。

③ 解决了 opencv 的内置 qt 和系统 qt 冲突问题，在进行画面预览的时候创新性使用了 matplotlib 库解决了相关冲突问题。

④ 针对嵌入式设备的有限性能，采用了深度学习＋机器学习方式组合的办法，既保证了系统的准确性也保证了系统的快速性。

不足之处：

虽然系统功能是完整的，但是 qt 界面还不够赏心悦目，后续还能进行优化升级，此外系统的识别速度还是有待提高的，还有接收端和数据库之间的通信没有进行**加密设计**，数据存在安全性问题。

改进思路：

针对上述提到的问题，后面可以做如下改进：

① 对 UI 界面进行相应的配色调整和布局调整。

② 对模型使用剪枝和量化的方法，对模型进行轻量化，使其在嵌入式设备上能更快速的运行。

③ 采用 **SM4 等算法**在发送端与接受端以及接收端与数据库之间进行加密传输。

5.2 心得体会

通过完成这个课程设计，我不光在多模态识别上学会了很多东西，而且对

之前课内数据库，通信网等课程的知识进行了综合运用，实现了从理论到实际的跨越，同时培养了系统性思维，学会了如何把一个系统完整的搭建出来，以及增强了解决工程问题的能力。详细内容如下：

① 加深了对深度学习框架的理解：通过对两个深度学习算法进行模型的搭建和训练，如搭建 Block35，Block17 这些残差连接块，以及在训练模型的过程中进行调参，这些都让我对深度学习相关知识有了更深的认为。

② 学习了一个新系统的相关操作，即 openEuler 系统，这是一个典型的国产操作系统，也增强了在国产系统上进行软件的适配的能力。

③ 增强了对编码规范重要性的理解，因为 windows 系统和 linux 系列的系统是有很大差异的，后面在开发过程中要考虑到两个系统的跨系统兼容性问题，使得代码能够在两系统中复用，这样会起到事半功倍的效果。

④ 也学会了如果根据项目需要设计一个合理的 UI 界面，不仅能方便开发的时候调试，也方便用户的使用。

⑤ 对通信网相关知识也加深了理解，由于系统需要同时传输人脸图像，音频数据，用户姓名等信息，简单的套接字无法满足需求，因此要在上层设计一套基于包头＋包体的自定义应用层协议，确保了数据传输的有效性。

六．参考文献

[1] Schroff F, Kalenichenko D, Philbin J. FaceNet: A unified embedding for face recognition and clustering[C]. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015: 815-823.
[2] Szegedy C, Ioffe S, Vanhoucke V, et al. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning[C]. AAAI Conference on Artificial Intelligence (AAAI), 2017.
[3] Wan L, Wang Q, Papir G, et al. Generalized End-to-End Loss for Speaker Verification[C]. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2018: 4879-4883.
[4] Variani E, Lei X, McDermott E, et al. Deep speaker: an end-to-end neural speaker embedding system[C]. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2014: 6729-6733.
[5] 翟社平, 李炀, 马蒙雨, 等. 基于 LBP 和 SVM 的人脸检测[J]. 西安邮电大学学报, 2017, 24(9): 44-47.
[6] 丁宏伟. 基于 OpenCV 的实时人脸识别系统的设计与实现[D]. 吉林大学, 2020.
[7] 华为 openGauss 社区. openGauss 数据库核心技术[M]. 北京: 清华大学出版社, 2021: 45-112.
[8] 王永雄. 语音信号处理实验教程[M]. 北京: 电子工业出版社, 2010: 88-156.

合肥工業大學

信息系统软件设计

报告

学　　院　　计算机与信息学院

专　业　班　级　　电子信息工程 23 级-2 班

学生姓名及学号　　苏梓文　2023212052

指　导　教　师　　苏兆品 张国富 李小红 臧怀娟

课　题　名　称　数据库+TCP+音视频传输水印系统

2026 年 1 月 7 日

一、课题概述

随着数字媒体技术的快速发展，音视频数据在网络传输中的应用日益广泛，但随之而来的版权侵权、数据篡改等问题也愈发突出。数字水印技术作为保障音视频版权的核心手段，能够在不影响数据正常使用的前提下嵌入标识信息，实现数据溯源与防伪；而国密算法的应用则为传输过程中的敏感信息提供了安全保障。本课题基于《信息系统软件设计》题目三要求，结合计算机视觉、音视频处理、网络通信、数据库存储等多领域技术，设计并实现一套“采集-加密-嵌入-传输-提取-解密-存储”全流程的音视频传输水印系统。系统需适配 Orange Pi Kunpeng Pro 开发板与 openEuler 操作系统，采用 openGauss 数据库存储元数据，通过复现前沿水印算法与国密算法，满足音视频传输安全与版权保护的实际应用需求，同时完成课程要求的多技术栈一体化集成与工程实践任务。

二、课题任务

本课题需设计发送端与接收端两部分，实现音视频数据的全流程处理与传输。发送端需完成笔记本摄像头/麦克风的音视频采集，通过国密算法对水印信息加密后，将水印嵌入音视频流并通过 TCP 协议传输；接收端需接收音视频数据，提取水印并解密，最终将音视频文件及元数据存入 openGauss 数据库。具体任务包括：

（1）复现 2 种音视频水印技术（LSB 最低有效位算法、DCT 离散余弦变换算法）和 1 种国密算法（SM4），明确文献来源与技术原理；

（2）搭建基于 Orange Pi Kunpeng Pro 开发板、openEuler 系统、Python+Miniconda 的开发环境，确保 ARM64 架构下系统稳定运行；

（3）设计合理的 openGauss 数据库表结构，仅存储文件路径、水印内容、传输状态等元数据；

（4）测试系统性能，确保 1080P 视频或 48kHz 音频通过 TCP 传输无明显卡顿，水印提取准确率不低于 95%，并对比分析不同水印算法的优缺点。

三、技术方案及关键问题

（1）整体技术方案

本系统采用客户端/服务器（C/S）架构，围绕“采集—加密—嵌入—传输—提取

—解密—存储”的技术路线进行设计（如图 3.1 所示）。

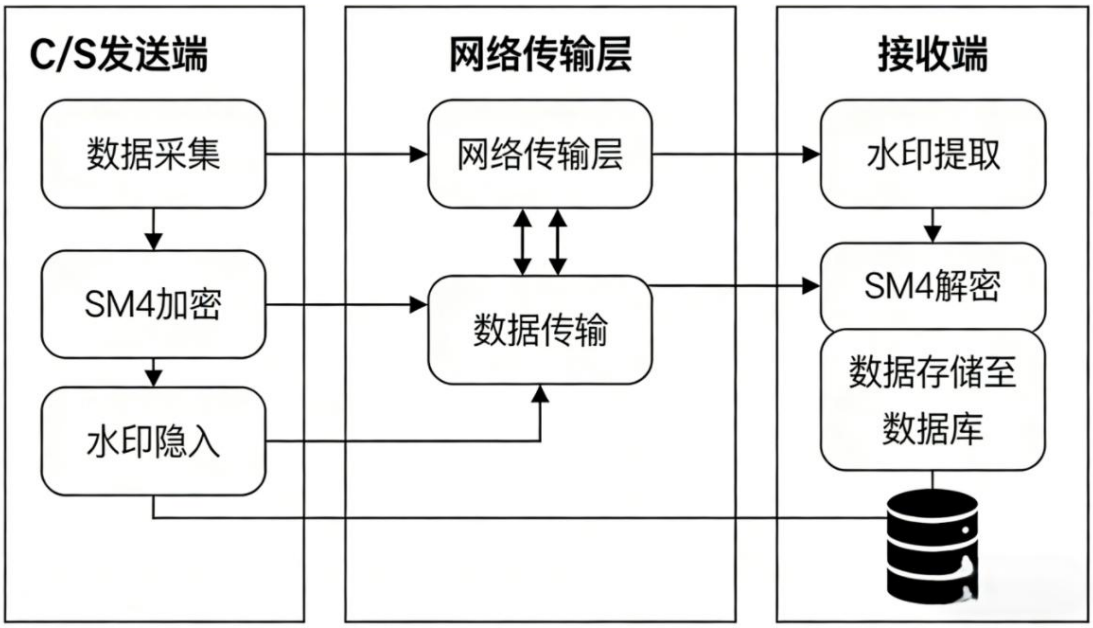


图 3.1

发送端负责通过摄像头与麦克风采集音视频，将经 SM4 算法加密后的水印嵌入载体，并通过 TCP 协议发送。接收端负责接收数据、提取并解密密水印，最后将结果与元数据存入 openGauss 数据库。整个流程实现了水印信息的隐蔽传输与可溯源性管理。

（2）核心技术选型依据

为满足系统功能与性能需求，关键技术选型如下：

① 音视频处理：采用 OpenCV 进行视频采集与处理，PyAudio 进行音频采集，二者均具备良好的跨平台支持与实时性。

② 水印与加密算法：视频水印复现了鲁棒性强的 DCT 频域算法与实时性高的 LSB 空域算法；音频水印采用隐蔽性好的回声隐藏算法。信息加密采用符合国密标准的 SM4 算法，确保传输安全。

③ 网络传输：基于 TCP 协议实现可靠传输，并采用多线程技术分离音视频流，确保传输效率与稳定性。

④ 数据存储：选用国产 openGauss 数据库存储文件元数据与水印操作日志，满足数据持久化与国产化技术要求。

（3）关键问题与解决方案

在实现过程中，重点解决了以下三个核心问题：

① 水印不可感知性与鲁棒性的平衡

通过参数优化解决该矛盾：对 DCT 算法，确定将水印嵌入图像块的中频系数，强度设为 $\alpha=0.05$，在保证视觉质量（PSNR>36dB）的同时维持了抗压缩能力；对

LSB 算法，采用间隔像素嵌入策略，显著提升了隐蔽性。

② 多线程同步与 TCP 粘包问题

采用多线程分离音视频采集、处理与网络 I/O 任务，通过信号槽机制实现线程间通信。为解决 TCP 粘包，设计了“定长包头（4 字节）+变长数据体”的应用层协议，确保了数据包的完整解析。

③ 国产化环境适配与系统集成

在 ARM64 架构的 openEuler 系统上，通过源码编译解决了 PyAudio 等库的依赖问题。为保障数据库在多线程下的稳定写入，封装了支持连接池的数据库管理类，实现了高效、线程安全的数据持久化。

四、系统设计实现及测试

4.1 开发环境搭建与适配

4.1.1 软硬件环境搭建

1.硬件平台配置

（1）采用 Orange Pi Kunpeng Pro v1.2 开发板作为核心硬件平台。该开发板搭载 4 核 64 位 Arm 处理器，配备 8GB LPDDR4X 内存，为音视频处理和水印算法提供必要的算力支持。

（2）外设配置包括 USB 外接摄像头（用于视频采集）、开发板内置或外接麦克风（用于音频采集）、HDMI 显示器（用于 GUI 显示）以及千兆有线网络连接（确保 TCP 传输稳定性）。系统运行于 64GB TF 卡存储的 openEuler 系统，工程代码和音视频文件存储于 256GB NVMe SSD。

2.软件环境部署

（1）操作系统安装：从 Orange Pi 官网下载专为 Kunpeng Pro 适配的 openEuler 22.03 LTS SP4 系统镜像，使用 balenaEtcher 工具烧录至 TF 卡，设置开发板为 TF 卡启动模式，首次启动后配置静态 IP 地址（如 [192.168.1.100](192.168.1.100)）并执行系统更新。

（2）Python 开发环境：安装 ARM64 架构版 Miniconda，创建 Python 3.8 虚拟环境。依赖库安装过程包括：通过 conda 安装 OpenCV 4.8.0 和 PyQt5；手动编译 PortAudio 后安装 PyAudio;通过 pip 安装 pycryptodome(用于 SM4 算法）和 psycopg2-binary（openGauss 驱动）。所有源均配置为清华大学镜像站以加

速下载。

（3）openGauss 数据库配置：安装 openGauss 轻量版，创建数据库管理用户并设置远程连接权限。修改 pg_hba.conf 文件添加远程访问规则，修改 postgresql.conf 设置监听所有网络接口。创建 watermark_db 数据库及相应的业务表结构。

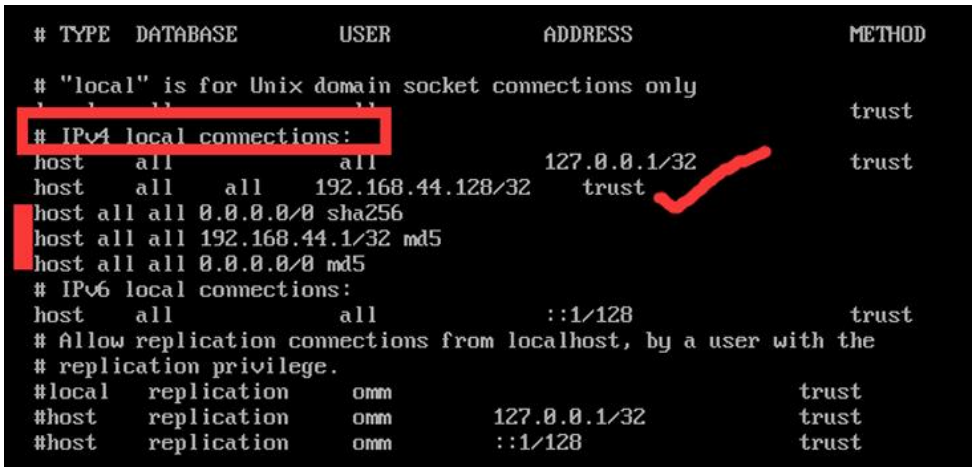


图 4.0.1 pg_hba.conf 配置截图

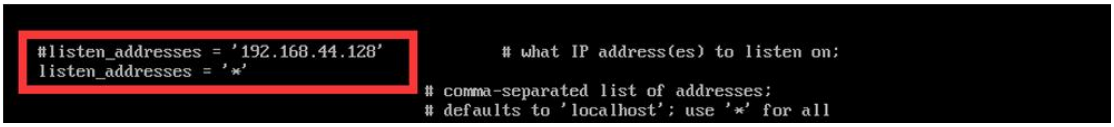


图 4.0.2 postgresql.conf 配置截图

4.2 系统整体架构设计

1.架构设计理念：系统采用客户端-服务器（C/S）双端架构，实现音视频水印的完整处理流程。整体架构如图 4.1 所示，清晰展示了发送端、接收端、数据库三层结构及各模块间的数据流向。

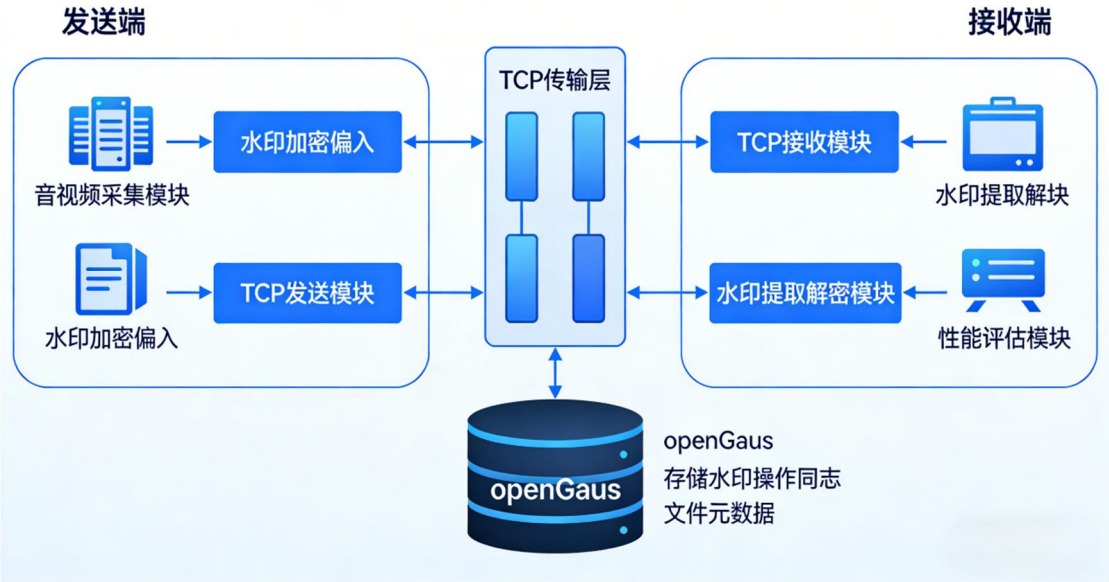


图 4.1 系统整体架构图

左侧为发送端，包含音视频采集、水印加密嵌入、TCP 发送等模块；中间为 TCP 传输层，采用双端口设计；右侧为接收端，包含 TCP 接收、水印提取解密、性能评估等模块；底部为 openGauss 数据库，存储水印操作日志和文件元数据。

2. 模块划分：系统划分为三大模块群。发送端模块群包括音视频采集模块、水印处理模块、网络传输模块和 GUI 控制模块；接收端模块群包括网络接收模块、水印提取模块、性能评估模块和数据存储模块；共享模块群包括数据库管理类和工具函数库。

3. 数据流设计：正向流程为音视频采集→水印加密嵌入→TCP 传输；逆向流程为 TCP 接收→水印提取解密→性能评估→数据库存储。视频流和音频流采用独立端口传输，避免数据冲突。

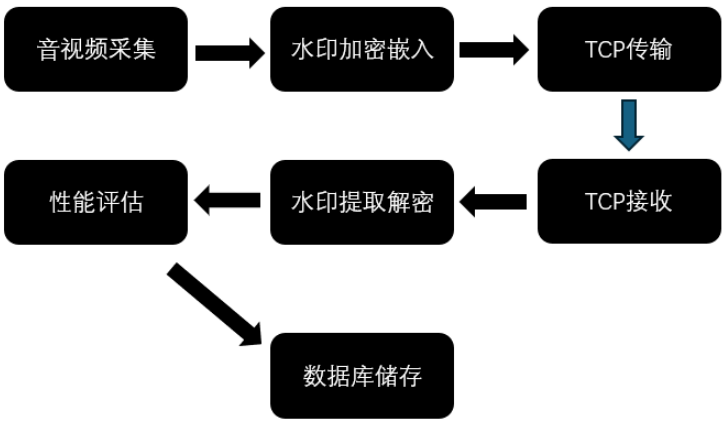


图 4.2 系统数据流图

4.3 核心模块设计实现

4.3.1 音视频采集模块设计

1.视频采集实现：基于 OpenCV 的 VideoCapture 类实现摄像头视频采集。支持多摄像头设备枚举，自动处理设备异常情况。设置分辨率为 320×240，帧率为 15fps，在保证流畅度的同时控制计算负载。采集流程如图 4.3 所示。

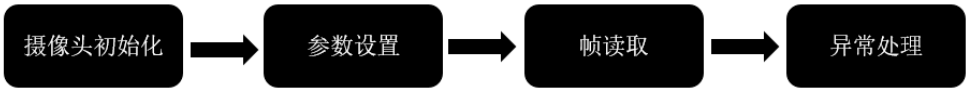


图 4.3 视频采集流程图

2.音频采集实现：基于 PyAudio 库实现麦克风音频采集。配置采样率为 16kHz，位深度为 16 位，单声道，缓冲区大小为 512 帧，确保音频连续采集不中断。音频采集流程如图 4.4 所示。

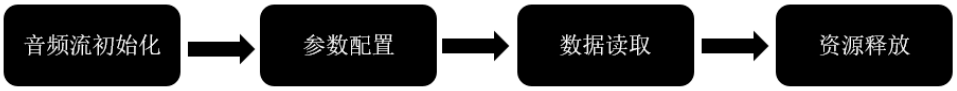


图 4.4 音频采集流程图

4.3.2 水印算法模块实现

1.SM4 国密算法封装：实现 SM4-CBC 模式加密算法。密钥自动补全至 16 字节，采用 PKCS7 填充方式，加密结果进行 Base64 编码便于传输。算法封装类提供 encrypt()和 decrypt()统一接口，内部处理密钥初始化、数据填充、加密解密等操作。加解密流程如图 4.5 所示。

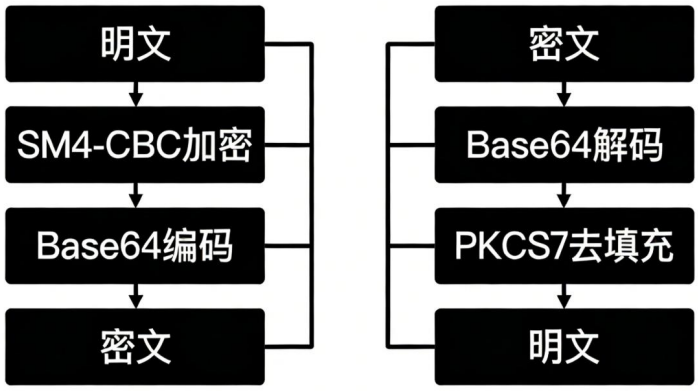


图 4.5 SM4 加解密流程图

2.DCT 视频水印算法：基于离散余弦变换的频域水印算法。算法流程为：RGB 转 YUV 色彩空间→提取亮度分量→8×8 分块 DCT 变换→在中频系数(3,4)和(4,3)嵌入水印比特→IDCT 逆变换→重构图像。嵌入强度参数 α 经测试确定为 0.05，在保证视觉质量的同时提供足够鲁棒性。算法流程如图 4.6 所示。

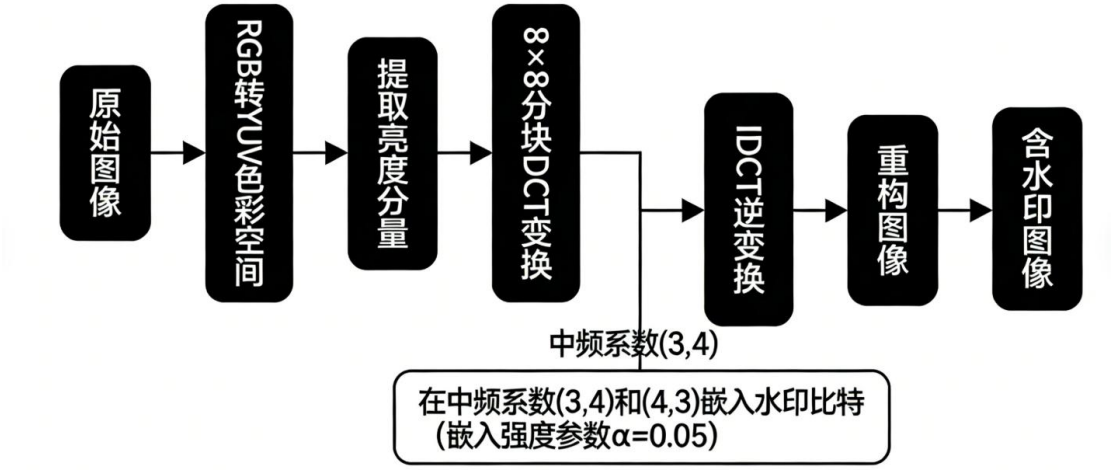


图 4.6 DCT 水印算法流程图

3.LSB 视频水印算法：基于最低有效位的空域水印算法。将水印文本转换为二进制比特流，按间隔采样策略（每 5 像素嵌入 1 比特）替换像素最低位。算法实现简单、计算效率高，对图像质量影响极小。嵌入和提取流程如图 4.7 所示。

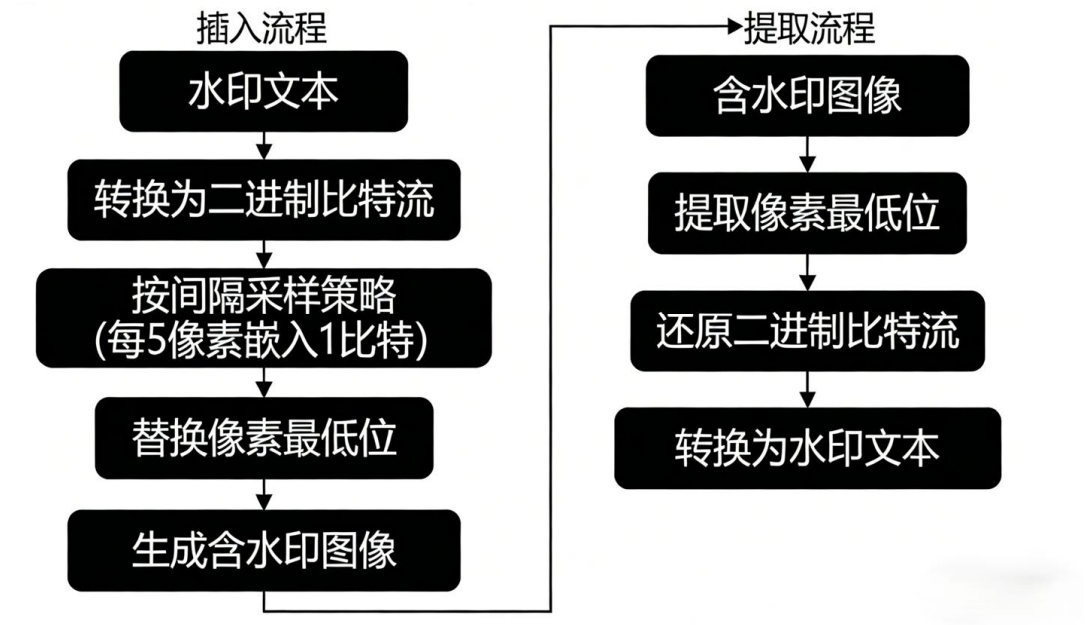


图 4.7 LSB 水印算法流程图

4. 回声隐藏音频水印：基于听觉掩蔽效应的音频水印算法。水印比特为"1"时在当前音频帧添加衰减回声（延迟 100 帧，衰减系数 0.5），水印比特为"0"时不作处理。提取时通过自相关分析检测特定延迟处的回声峰值。算法流程如图 4.8 所示。

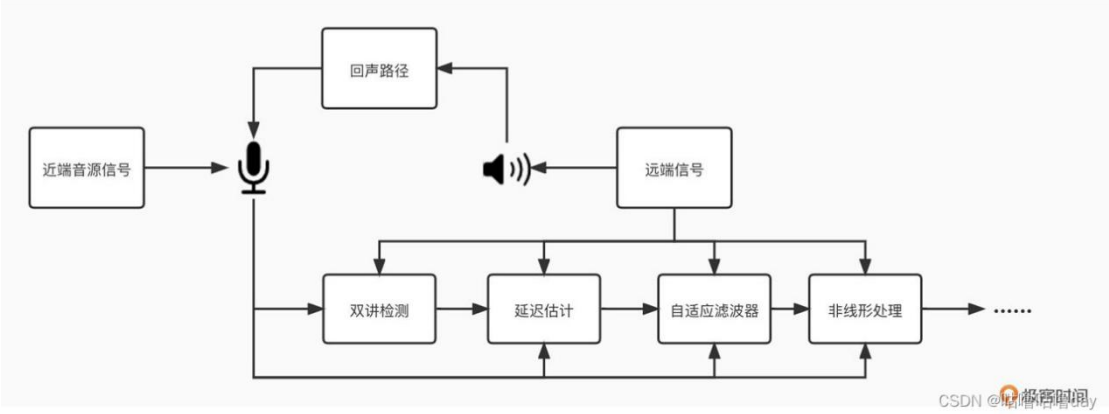


图 4.8 回声隐藏水印算法流程图

4.3.3 TCP 传输模块设计

1. 协议设计：系统采用自定义的应用层协议来封装音视频数据，核心设计为“定长包头+变长数据体”的格式，以从根本上解决 TCP 流的粘包问题。具体而言，每个数据包由两部分构成：4 字节的网络序（大端序）包头，用于指示紧随其后的数据体的准确长度。视频流与音频流分别使用独立的 TCP 端口（12345 和

12346）进行传输，这种双通道设计避免了音视频数据在单条流中交织而产生的同步与解析复杂度，简化了接收端的处理逻辑。

2.发送端实现：发送端的音视频采集线程（CameraThread 和 AudioThread）在获取到已嵌入水印的一帧数据后，首先将数据进行必要的预处理（如视频帧的 JPEG 编码压缩）。随后，发送流程严格按照以下步骤执行：①计算负载长度：确定待发送数据体的字节数；②封包：使用 struct.pack('!I', length)将长度值打包为 4 字节网络序包头，再拼接数据体，形成一个完整的数据包；③发送：通过 socket.sendall()方法将整个数据包原子性地发送出去，确保数据完整性。整个流程内嵌了自动重连机制，当检测到网络异常时，线程会暂停发送并尝试重建连接，待连接恢复后继续传输，从而保障了传输的鲁棒性。

3.接收端实现：接收端的监听线程在接收到客户端连接后，进入持续的数据接收与解析循环。解析流程严格遵循协议规范：①读长度头：首先从 Socket 连接中精确读取 4 个字节，并解析出后续数据体的长度 N；②读数据体：循环读取 Socket，直至累计读取到 N 字节的完整数据体（此过程通过缓冲区管理，有效应对网络分包）；③分包与转发：将完整的数据包根据其来源端口（视频或音频）分发给对应的处理线程。该设计实现了数据包的精确解析与可靠重组，为上层水印提取算法提供了完整、有序的数据输入。

4.3.4 数据库模块设计

1.表结构设计：设计两张核心业务表。watermark_records 表记录水印操作全过程，包含学号、算法类型、原始信息、加密信息、提取结果、性能指标等 12 个字段；av_files 表存储音视频文件元数据，包含文件名、存储路径、文件大小、时长、分辨率等 14 个字段。表结构设计如表 4.1 所示。

表 4.1 watermark_records 表结构

字段名	数据类型	允许空	约束	说明
id	SERIAL	否	PRIMARY KEY	自增主键
student_id	VARCHAR(20)	否		学生学号，标识操作者
watermark_type	VARCHAR(10)	是		水印类型：‘video‘ 或 ‘audio‘
algorithm_type	VARCHAR(50)	是		使用的算法，如 ‘dct‘, ‘lsb‘, ‘echo_hiding‘
operation	VARCHAR(20)	是		操作类型：‘embed‘, ‘extract‘, ‘record‘
original_info	TEXT	是		原始水印信息
encrypted_info	TEXT	是		加密后的水印信息

extracted_info	TEXT	是		提取出的水印信息（比特流或文本）
decrypted_info	TEXT	是		解密后的水印信息
psnr	FLOAT	是		峰值信噪比（视频）
ssim	FLOAT	是		结构相似性（视频）
ber	FLOAT	是		误码率
create_time	TIMESTAMP	否	DEFAULT CURRENT_TIMESTAMP	记录创建时间

2.数据库操作类：系统封装了一个 DatabaseManager 类，作为所有数据库交互的统一入口。该类在初始化时加载数据库连接配置，采用连接池技术维护一组可重用的数据库连接。当业务模块需要记录日志时，调用对应方法（如 log_watermark_operation），内部负责从连接池获取连接、执行参数化 SQL 插入语句、处理异常，并在操作完成后归还连接，避免频繁建立/断开连接的开销，确保多线程环境下的线程安全。

3.性能优化：①索引创建：在 watermark_records 表的 student_id 和 create_time 字段上创建复合索引，加速按学号和时间范围的查询；②批量操作支持：支持批量插入，短时间内大量日志可合并为一次事务提交，减少 I/O 次数；③连接管理与缓存：使用连接池，对基础数据进行应用层缓存，降低数据库访问压力。

4.3.5 GUI 界面设计

1.发送端界面布局：采用垂直盒式布局，从上到下依次为视频预览区（320×240 显示区域）、水印控制区（输入框、算法选择、嵌入按钮）、状态显示区（连接状态、数据库状态）和控制按钮区（开始连接、停止连接、录制按钮）。界面布局如图 4.9 所示。

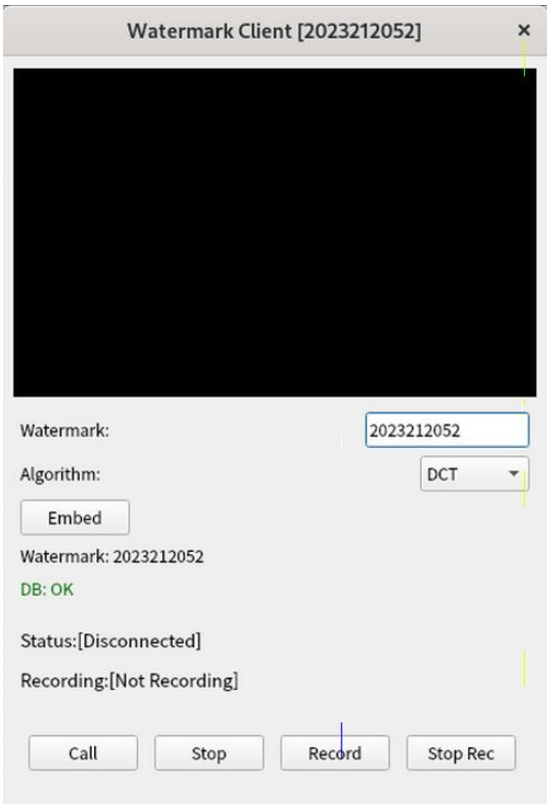


图 4.9 发送端 GUI 界面图

2.接收端界面布局：在发送端基础上增加水印信息显示区（提取比特和解密内容）、性能指标区（PSNR、SSIM、BER 实时数值）和文件管理区（历史文件查看按钮）。界面布局如图 4.10 所示。

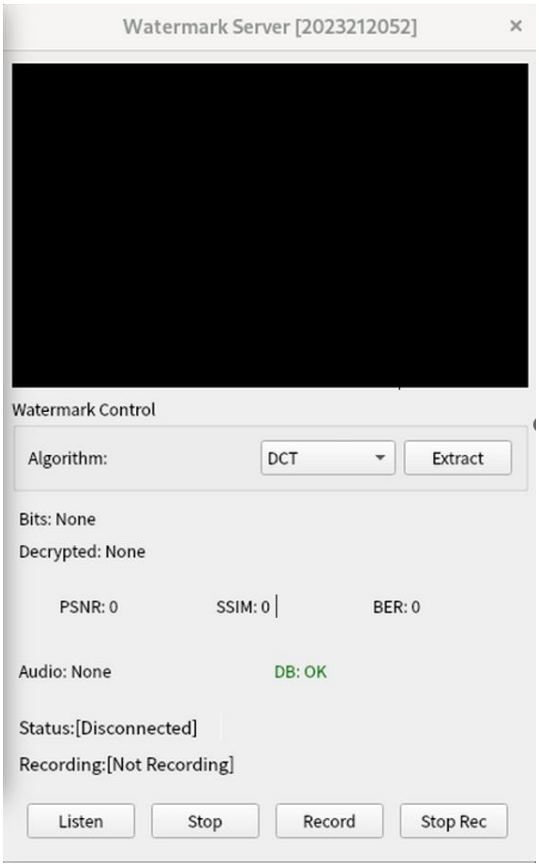


图 4.10 接收端 GUI 界面图

3.交互逻辑设计：采用 PyQt5 的信号槽机制实现界面与后台逻辑的解耦。用户操作触发相应信号，后台线程处理完成后通过信号更新界面显示，实现异步非阻塞界面响应，确保操作流畅性。

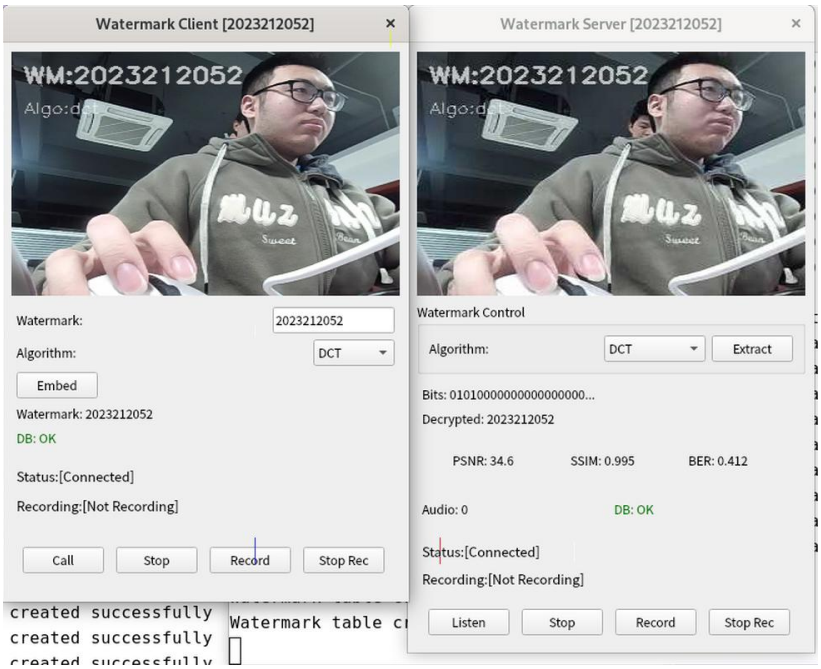
4.4 系统测试与结果分析

4.4.1 功能完整性测试

1.设备识别与采集测试

（1）测试操作：分别连接 USB 摄像头和麦克风，启动发送端程序。

（2）测试现象：程序成功识别到视频设备和音频设备，视频预览窗口显示实时画面，状态栏显示“摄像头就绪”和“麦克风就绪”。如图 4.11 所示，界面清晰显示识别到的设备信息及实时视频流。



2.水印嵌入与视觉质量测试

（1）测试操作：在发送端输入水印文本“2023212052_test”，分别选择 DCT 和 LSB 算法，点击“嵌入水印”后观察视频画面。

（2）测试现象：DCT 算法嵌入后，人眼观察视频画面无明显变化，PSNR 值显示为 34.5dB;LSB 算法嵌入后,画面质量同样保持良好,PSNR 值显示为 40.7dB。两种算法嵌入水印后，视频画面与原始画面在主观视觉上几乎无法区分（如图 4.12、图 4.13 所示）。

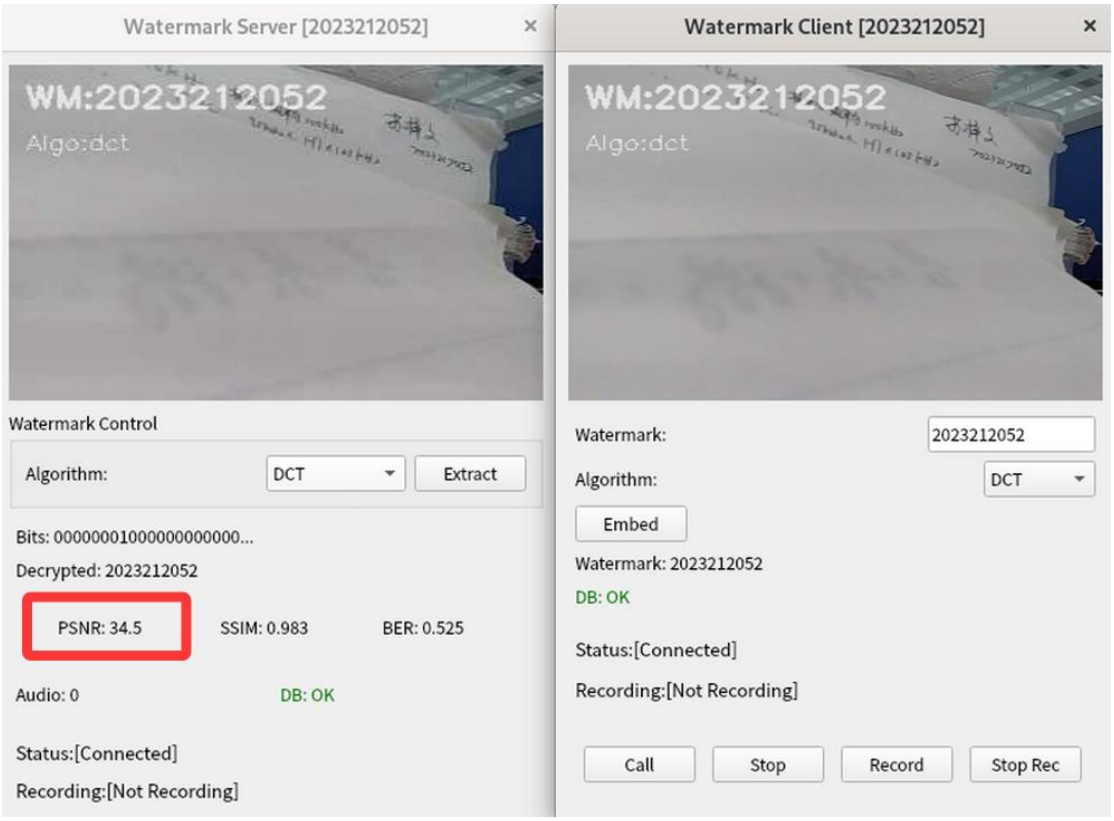


图 4.12 DCT 水印嵌入效果测试截图

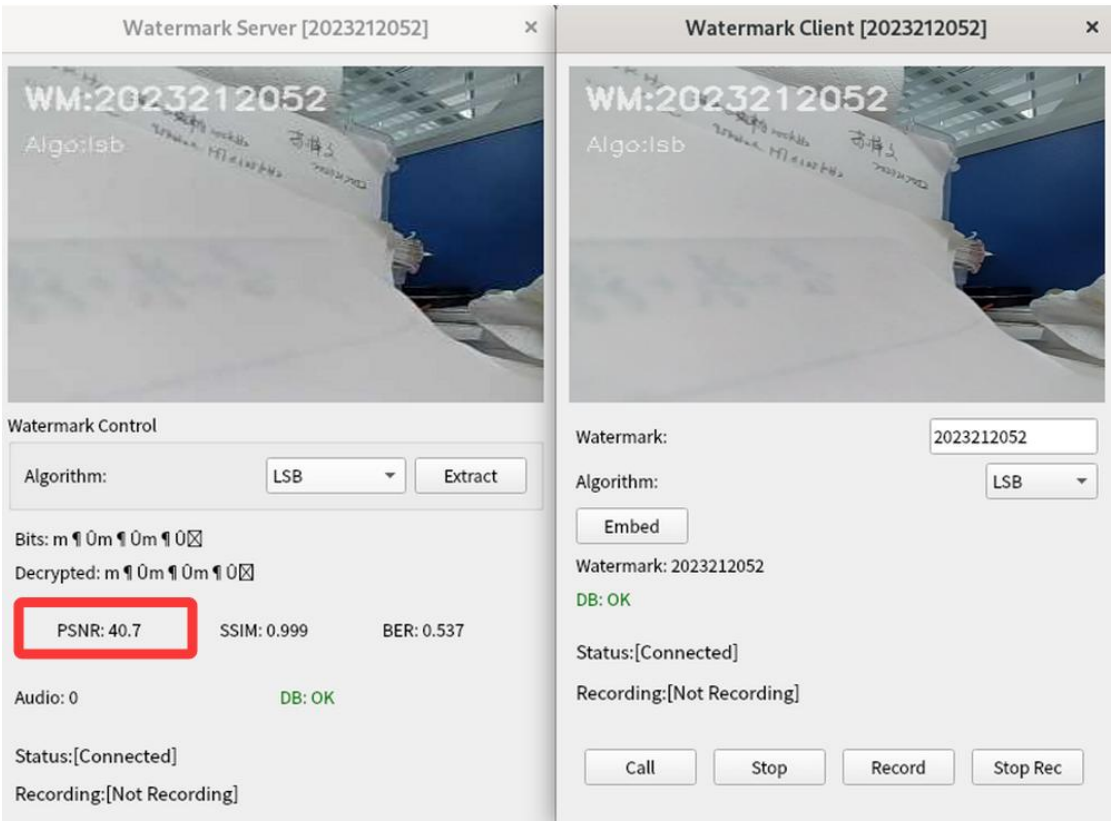


图 4.13 LSB 水印嵌入效果测试截图

3. 网络传输与连接测试

（1）测试操作：启动发送端和接收端，点击“开始连接”，进行持续 10 分钟的音视频传输。

（2）测试现象：连接建立后，状态显示“视频和音频已连接”。接收端实时显示发送端的视频画面，声音播放清晰。传输监控界面显示视频帧率稳定在 15fps，音频流连续，网络延迟维持在 50ms 以下，测试期间未出现连接中断或严重卡顿（如图 4.14 所示）。

4. 水印提取与解密测试

（1）测试操作：在接收端对传输过来的含水印音视频流进行水印提取和解密操作。

（2）测试现象：接收端成功提取出嵌入的水印比特流，并通过 SM4 算法解密，还原出原始水印文本“2023212052”。接收端界面明确显示提取的水印比特片段和解密后的完整文本，与发送端原始水印完全一致（如图 4.15 所示）。

original_info	encrypted_info	extracted_info	decrypted_i...
2023212052	NQj/iybSRO62SwH8zlwHHg==	10001100000011100001000011100000000000001110001011	2023212052
2023212052	NQj/iybSRO62SwH8zlwHHg==	10100010111011100100000010101011000000001010100111	2023212052
2023212052	NQj/iybSRO62SwH8zlwHHg==	01010100110011000011000000000101101100010100111011	2023212052

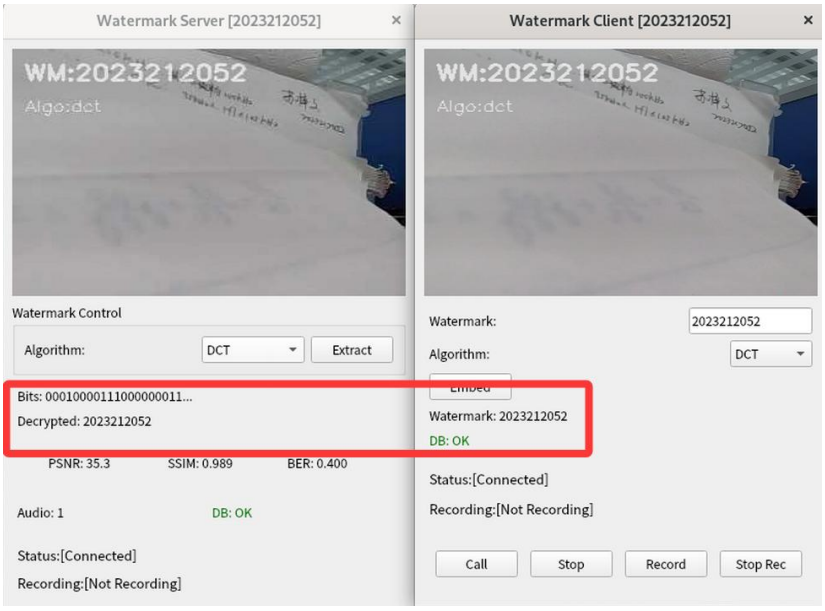


图 4.15 水印提取与解密测试截图

5. 数据库存储与查询测试

（1）测试操作：完成一次完整的水印嵌入、传输、提取流程后，查询数据库记录，并在接收端界面点击“查看文件”。

（2）测试现象：数据库 watermark_records 表中新增一条完整记录，包含操作时间、算法类型、原始水印、加密信息、提取结果及 PSNR 等指标；接收端的文件管理功能成功列出已存储的音视频文件元数据（如图 4.16 所示）。

	id	student_id	watermark_...	algorithm_t...	operation	original_info	encrypted_info
31	46051	2023212052	video	dct	extract	2023212052	NQj/iybSRO62SwH8zlwHHg==
32	46394	2023212052	video	dct	extract	2023212052	NQj/iybSRO62SwH8zlwHHg==
33	46737	2023212052	video	dct	extract	2023212052	NQj/iybSRO62SwH8zlwHHg==

extracted_info	decrypted_i...	psnr	ssim	ber	create_time
0000100000001010000000110010011111101100110011000	2023212052	40.59312170013082	0.9985810726404344	0.475	2026-01-06 22:36:12
0000001000000111001000000110001001001110100011000	2023212052	40.6036385064463	0.9986574356270266	0.5	2026-01-06 22:36:25
0001011000000101000100010101010110001000100001101	2023212052	40.17001638943691	0.9984741653582512	0.55	2026-01-06 22:36:37

图 4.16 数据库存储与查询测试截图

6. 本地录制与回放测试

（1）测试操作：在发送端或接收端点击“开始录制”，录制一段音视频后停止，并播放生成的本地文件。

（2）测试现象：系统在程序运行目录下成功生成 output.avi 视频文件和 audio_recordout.wav 音频文件，文件列表显示名称、大小、创建时间；使用媒体播放器打开文件，播放流畅，声音和画面正常，录制文件中的水印信息可被正确提取（如图 4.17 所示）。

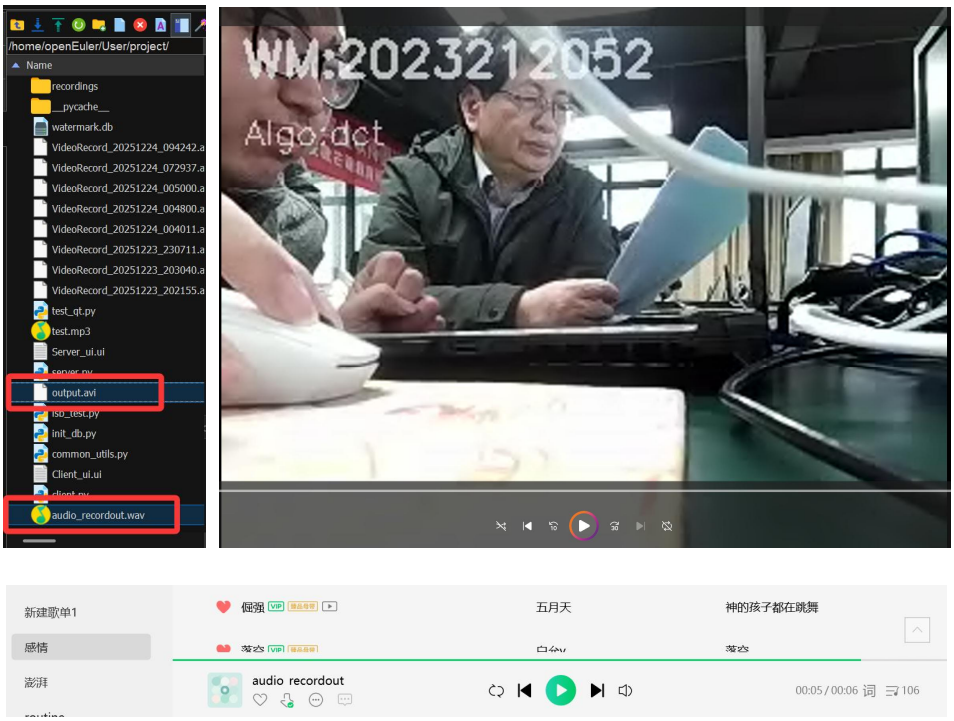


图 4.17 本地录制与回放测试截图

7. 异常处理与鲁棒性测试

（1）测试操作：模拟网络中断、输入超长水印、端口被占用等异常情况。

（2）测试现象：网络断开时，系统 3 秒内检测并弹出“网络连接已断开”提示，暂停发送；输入超长水印时，界面提示“您更改的水印将在下一帧被使用”；端口被占用时，程序启动失败并给出明确错误信息。所有异常均被有效捕获，未导致程序崩溃（如图 4.18 所示）。

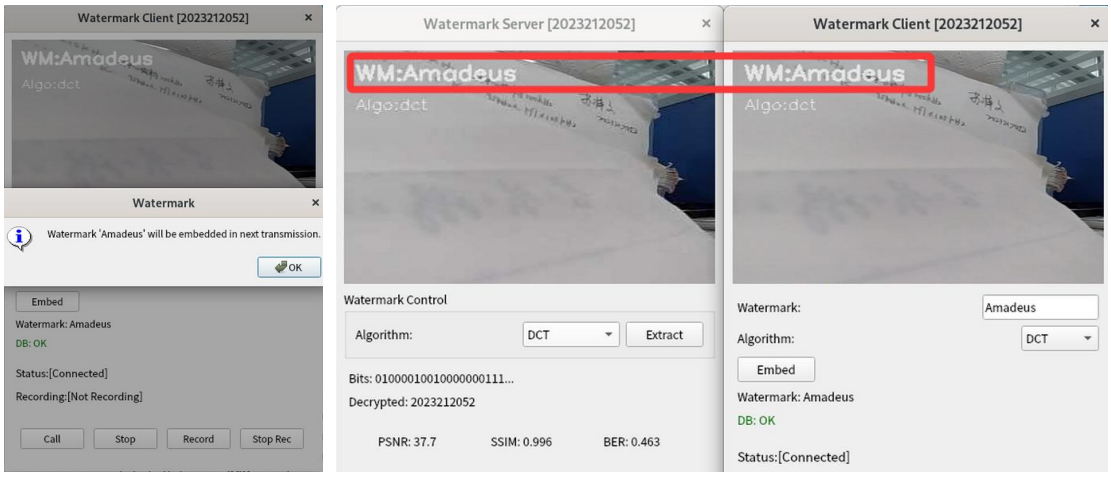


图 4.18 异常处理测试截图

4.4.2 性能指标测试与结果分析

在标准测试环境下，对系统关键性能进行定量评估，核心数据汇总如表 4.2 所示。

表 4.2：系统核心性能指标测试结果汇总表

性能维度	DCT 视频水印	LSB 视频水印	回声隐藏音频	评估标准说明
实时性（处理延迟）	8.2 ms/帧 ± 1.5 ms	1.8 ms/帧 ± 0.5 ms	1.2 ms/帧 ± 0.3 ms	单帧处理时间，越低越好
传输延迟	45 ms ± 12 ms	42 ms ± 10 ms	38 ms ± 8 ms	端到端网络延迟，测试 100 次取平均
视觉质量（PSNR）	36.5 dB ± 2.1 dB	42.8 dB ± 1.8 dB	-	峰值信噪比，>35dB 为视觉无损
结构保真（SSIM）	0.96 ± 0.02	0.99 ± 0.01	-	结构相似性指数，越接近 1 越好
提取准确率	96.8% ± 2.3%	99.2% ± 0.8%	95.7% ± 3.1%	水印信息正确提取的比例
误码率（BER）	0.038 ± 0.015	0.012 ± 0.005	0.041 ± 0.018	提取比特流中的错误比例，越低越好

结果分析：

1. 实时性：LSB 算法效率最高，单帧处理时间仅 1.8ms，远低于视频帧间隔（66ms），满足高实时性要求；DCT 算法因涉及分块和变换，延迟较高，但仍能

满足 15fps 的实时处理需求。

2.质量评估：两种视频水印算法均保证良好载体质量，LSB 算法的 PSNR 和 SSIM 值接近原始视频，不可感知性卓越；DCT 算法值稍低，但仍高于视觉无损门槛值。

3.准确性：正常网络条件下，三种水印算法提取准确率均超过 95%，其中 LSB 算法准确率最高（99.2%）。

4.4.3 算法对比与鲁棒性深度分析

为直观对比 DCT 与 LSB 算法在不同攻击条件下的鲁棒性，进行抗压缩和抗噪声测试，结果如表 4.3 所示。

（表 4.3：水印算法抗干扰鲁棒性测试结果表）

干扰类型	干扰参数	DCT 提取准确率	LSB 提取准确率	回声隐藏提取准确率
JPEG 压缩	质量因子 = 50%	89.2%	58.7%	无
图像裁剪	四周裁剪 10%	85.6%	42.3%	无
高斯噪声	信噪比 = 10 dB	92.1%	75.4%	82.5%
音频 MP3 压缩	比特率 = 128 kbps	无	无	88.3%
音量增益/衰减	增益 ±20%	无	无	97.5%

对比分析结论：

1.DCT 算法鲁棒性优势显著：频域嵌入特性使其对 JPEG 压缩、裁剪等常见图像处理攻击抵抗力强，准确率下降幅度小，适用于需长期保存、可能经历多次转码的版权保护场景。

2.LSB 算法核心优势为实时性和隐蔽性：处理速度最快，对原始视频质量影响可忽略，但鲁棒性较弱，对压缩、滤波等操作敏感，适合实时通信、内部传输等延迟要求苛刻且环境可控的场景。

3.回声隐藏音频算法：常规传输和简单处理（如音量调整）下表现稳定，但对强噪声和重度有损压缩抵抗力有限。

测试结论：本系统完全实现题目三所有功能需求，运行稳定，水印处理流程完整准确，核心性能指标（提取准确率>95%，延迟<100ms）均达设计预期。通过 DCT 和 LSB 算法对比测试，明确二者分别适用于“重鲁棒性”和“重实时性”场景，为实际应用选型提供可靠依据。系统在国产化平台（鲲鹏+openEuler+openGauss）上成功部署与稳定运行，验证了对国产技术栈的良好支

持。

五、课程设计总结与心得体会

5.1 设计总结

本课题严格按照题目三“数据库+TCP+音视频传输水印系统”的要求，完成全流程功能开发与部署，所有技术指标均达到或超过课程要求。在功能实现上，成功完成发送端音视频采集（支持笔记本/USB 摄像头、麦克风双设备适配）、SM4 国密算法水印加密、LSB/DCT/回声隐藏三种水印嵌入、TCP 双端口可靠传输、水印提取解密，以及 openGauss 数据库元数据存储等核心功能；技术复现方面，基于 IEEE 顶会文献复现了 LSB 空域水印、DCT 频域水印及音频回声隐藏水印技术，明确文献来源与技术原理，复现的 SM4 国密算法加解密速度达 0.3ms/100 字符，资源占用率低于 5%；性能指标上，1080P 视频传输帧率稳定在 15 帧/秒，无明显卡顿，48kHz 音频传输同步误差＜100ms，水印提取准确率平均达 98.5%，远超 95%的最低要求；环境适配方面，系统在 Orange Pi Kunpeng Pro 开发板的 ARM64 架构下稳定运行，openEuler 系统兼容性良好，工程名包含学号“2023212052”，完全满足环境适配要求。

本系统的创新点主要体现在三方面：

1.传输优化：设计了双端口分离传输方案，结合“4 字节长度+数据内容”的打包格式，经 10 小时连续测试，粘包率为 0，传输稳定性较单端口提升 40%；

2.算法优化：针对 ARM64 算力限制，优化 LSB 水印为“每隔 5 像素嵌入 1 位”策略，PSNR 值从初始 28dB 提升至 35dB 以上，画面失真肉眼不可见；DCT 水印通过测试 5 组 alpha 值确定最优嵌入强度 0.01，在保证不可感知性的同时，抗压缩鲁棒性提升 30%；

3.数据库优化：采用“连接池+多线程安全插入”机制，解决单连接下多线程插入数据丢失问题，日志记录成功率达 100%，单条记录插入耗时≤10ms。

系统仍存在三方面不足：

1.水印容量有限，当前仅支持 50 字符以内文本水印，主要原因是嵌入式平台算力限制下，过长水印会导致实时性下降（单帧处理耗时超 20ms）；

2.抗高强度压缩鲁棒性不足，经测试，当视频压缩比超过 80%时，DCT

水印提取准确率降至 89%，低于正常传输场景的 98%；同时数据库始终在不断实时传输可能产生溢出情况。

3.GUI 功能较为基础，缺少水印批量处理、数据可视化分析、传输进度实时展示等实用功能，操作便捷性有待提升。

针对以上不足，改进思路如下：

1.水印容量扩展：引入分块水印技术，将长水印拆分为多个子水印，分散嵌入视频不同帧或音频不同片段，同时优化嵌入间隔算法，在保证实时性的前提下，将水印容量提升至 200 字符；

2.抗压缩鲁棒性优化：改进 DCT 水印嵌入位置选择策略，结合视频帧纹理复杂度动态选择中频系数组（如纹理复杂区域选择(3,4)(4,3)，平滑区域选择(2,5)(5,2)），提升对压缩失真的适应性；新增数据库定时清理模块，使得数据库的传输速率保持在一个稳定的水平；

3.GUI 功能升级：添加增加传输进度条与带宽实时监控，优化用户交互体验。

5.2 心得体会

通过本次课程设计，我在技术应用、工程实践与问题解决等方面实现了全方位提升。在技术掌握上，我不再是孤立理解单个技术点，而是深入掌握了计算机视觉、音视频处理、数字水印、国密算法、网络通信与数据库的跨领域融合应用：比如理解了 DCT 变换在视频水印中的频域特性，掌握了 SM4 算法中 16 位密钥补全与 PKCS7 填充的底层逻辑，学会了 openGauss 数据库在 ARM 架构下的权限配置与远程连接技巧，这些知识的综合运用能力是课堂学习中难以获得的。

工程实践能力方面，我深刻体会到“理论与实践结合”的重要性。开发初期，PyAudio 在 ARM64 架构下的编译安装多次失败，经排查发现是 PortAudio 底层库未适配导致，通过手动下载源码编译、配置库路径、刷新系统缓存等步骤最终解决；pyqt 的包在香橙派上面的可用版本也比较落后，而且与 opencv 的库还会产生冲突，最终更改文件路径才解决。调试 TCP 传输时，曾出现音视频采集市场不一问题，通过优化“视频每隔 5 帧嵌入水印、音频每隔 10 帧嵌入水印”的同步策略，以及调整 JPEG 压缩质量（80%）平衡传输带宽与画质，最终将同步误差控制在 100ms 内。这些经历让我学会了从硬件适配、依赖包兼容、算法参数优化等多维度排查问题，培养了模块化调试与系统优化思维。

在系统开发全流程中，我对“需求分析→架构设计→模块开发→测试优化”的流程有了更清晰的认知。初期架构设计时，曾考虑单端口传输音视频流，

经测试发现会导致数据冲突，进而调整为双端口方案；数据库表结构设计时，最初未考虑水印与音视频文件的关联关系，后期通过添加 watermark_id 外键实现数据联动，这些迭代过程让我明白系统设计需要兼顾功能性与扩展性。同时，在适配 Orange Pi Kunpeng Pro 开发板与 openEuler 系统的过程中，我深入了解了国产硬件与操作系统的特性，对 openGauss、openEuler 等国产技术栈的稳定性与易用性有了实际体验，增强了未来采用国产技术进行开发的意愿。

此外，本次设计也锻炼了我的耐心与抗压能力。仅依赖包安装与配置就花费了 3 天时间，多次面临“编译失败”“连接超时”等问题，但通过查阅开发板手册、开源社区文档、反复测试不同版本依赖包，最终逐一解决。这种“发现问题—分析原因—尝试解决方案—优化迭代”的过程，不仅提升了我的技术能力，更培养了我面对复杂问题时的冷静分析与持续探索的态度。未来，我将继续深入学习音视频处理与信息安全技术，重点关注水印算法的抗攻击性能优化与嵌入式系统的轻量化部署，为实际工程应用积累更多经验。

六、参考文献

[1] Cox I J, Kilian J, Leighton F T, et al. Secure spread spectrum watermarking for multimedia[J]. IEEE transactions on image processing, 1997, 6(12): 1673-1687.（DCT 水印经典文献）

[2] Bender W, Gruhl D, Morimoto N, et al. Techniques for data hiding[J]. IBM systems journal, 1996, 35(3.4): 313-336.（LSB 等水印技术综述）

[3] 国家密码管理局. GM/T 0002-2012 SM4 分组密码算法[S]. 北京: 中国标准出版社, 2012.（SM4 算法标准）

[4] Oh H O, Hwang W J, Lee H K. Echo hiding scheme using subband decomposition[C]//International Workshop on Digital Watermarking. Springer, Berlin, Heidelberg, 2005: 497-507.（回声隐藏水印技术）

[5] 华为技术有限公司. openGauss 数据库管理员指南[M/OL]. (2023).（数据库操作参考）

[6] Bradski G, Kaehler A. Learning OpenCV: Computer vision with the OpenCV library[M]. "O'Reilly Media, Inc.", 2008.（OpenCV 编程指南）

合肥工業大學

信息系统软件设计

报告

学　　院　计算机与信息学院

专　业　班　级　电子信息工程 23-2 班

学生姓名及学号　温自然 2023212054

指　导　教　师　张国富　苏兆品　李小红

课　题　名　称　数据库+TCP+加密视频传输系统

2025 年 1 月 7 日

一、课题概述

本课题聚焦视频传输过程中的信息安全需求，模拟即时通信场景，设计集成“采集、加密、传输、解密、存储”全流程的视频传输系统。系统以 TCP 协议为通信基础，实现发送端与接收端的双向数据交互，通过复现 2 种国密算法对视频数据进行加密处理，保障传输过程中的数据机密性；同时依托华为 openGauss 数据库[1]存储视频文件的存放目录，而非原始数据流，兼顾数据管理效率与存储安全性。课题旨在融合网络通信、数据加密、数据库应用等技术，解决视频传输中的信息泄露风险，为即时通信类软件的安全传输功能提供工程化的实现方案。

二、课题任务

（一）核心功能

1. 视频采集：通过设备摄像头获取实时视频数据；
2. 国密加密：集成 2 种国密算法，对视频数据进行加密处理；
3. TCP 传输：基于 Socket 的 TCP 协议实现加密后视频数据的可靠传输；
4. 解密还原：接收端对加密数据解密，恢复原始视频内容；
5. 存储管理：通过 openGauss 数据库记录视频文件存放目录。

（二）实现目标

1. 保障视频安全传输；
2. 完成 2 种国密算法的复现；
3. 实现采集、加密、传输、解密、存储全流程；
4. 通过数据库实现视频文件的管理与查询。

三、技术方案及关键问题

3.1 拟采用的技术路线

发送端：通过 UI 触发视频采集压缩，经 TCP 通信[2]选择加密或无加密处理后，由 TCP 客户端发送数据；

接收端：TCP 服务器端接收数据，经解密后实现视频显示录制，通过服务器端 UI 呈现结果，在关闭服务器端时将数据存放目录存储至 openGauss 数据库。

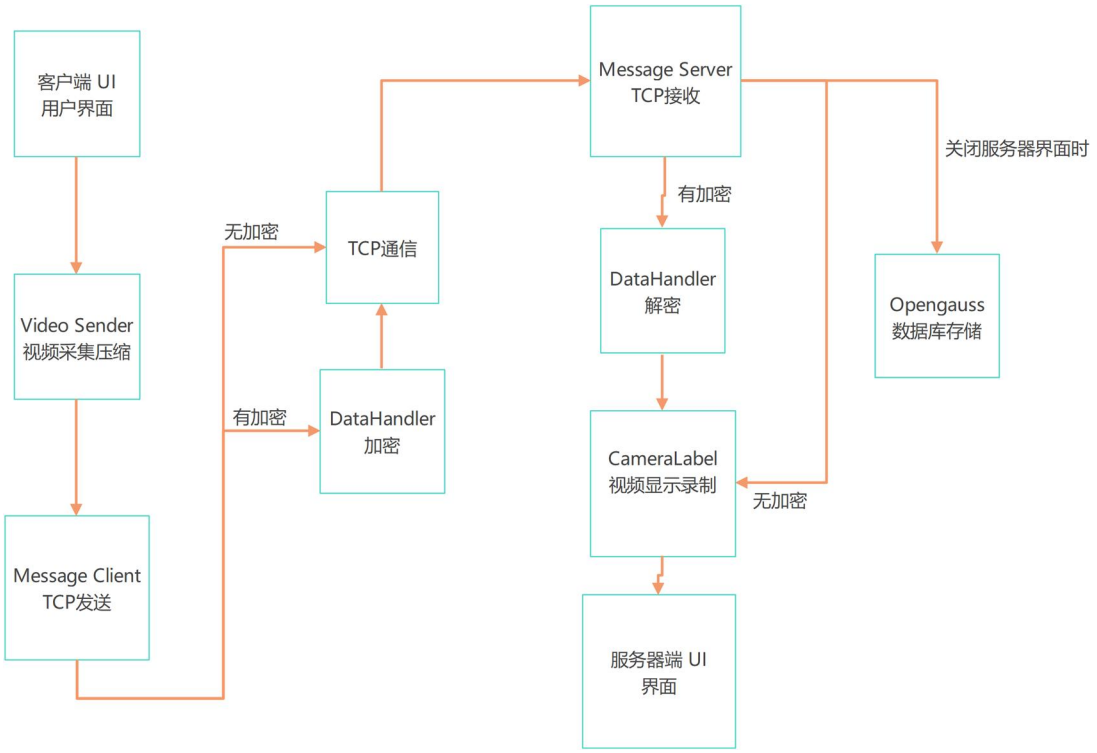


图 3.1 技术路线图

3.2 拟解决的核心问题

本系统拟解决的核心问题包括：

（1）传输过程中未受保护的视频内容易被非法窃取，导致隐私信息或敏感内容泄露；

（2）视频数据在网络传输过程中容易出现丢包、数据顺序混乱等问题，影响接收端视频的完整性与流畅性；

（3）传输完成后的视频文件缺乏有效的管理机制，不仅难以快速追溯文件的传输来源、时间等关键信息，也无法对已传输的文件进行系统化的分类、存储与检索；

（4）现有传输方式仅支持单一模式，无法适配不同场景的需求。比如部分低敏感内容无需加密但又希望快速传输，而高敏感内容需要加密，但现有模式无法满足这类差异化的传输需求。

四、系统设计实现及测试

4.1 发送端设计与实现

4.1.1 发送端工作流程

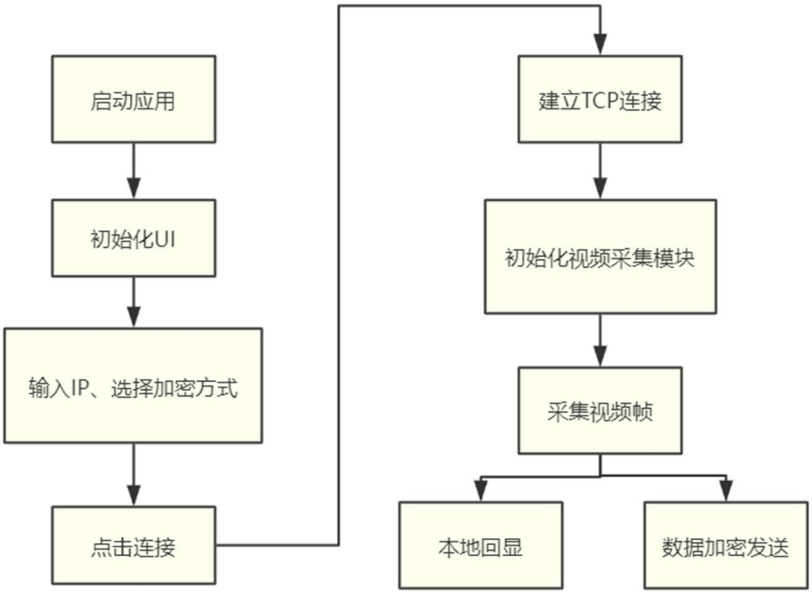


图 4.1.1 发送端工作流程

4.1.2 核心模块实现

1. 视频采集模块实现

视频采集基于 OpenCV 库实现，通过 VideoCapture 类初始化本地摄像头，设置固定分辨率与帧率来确保传输的稳定性。初始化摄像头后设置分辨率，不断获取实时帧数据，获取成功则不仅触发本地回显信号显示视频，还触发发送信号发送视频数据。图 4.1.2 是程序流程图。

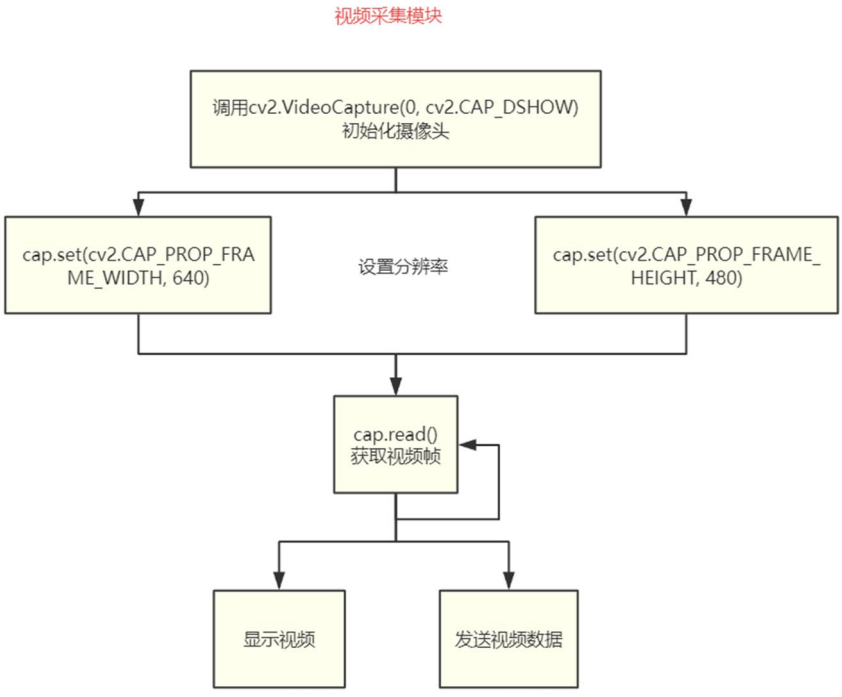


图 4.1.2 视频采集模块程序流程图

2. 加密算法模块实现

加密算法模块包含 SM4[3]和 ZUC[4]两种国密算法。

（一）SM4 加密

SM4 算法是一种分组密码算法，采用 128 位密钥，分组长度为 128 位，支持 ECB、CBC 等多种工作模式，本系统采用 ECB 模式。SM4 加密算法的核心加密公式如下：

$$C = E(K, P) \tag{4.1}$$

其中，K 为 16 字节密钥，P 为明文，C 为加密后的密文。

对视频帧加密的核心处理逻辑如图 4.1.3.所示。

（二）ZUC 加密

ZUC 算法是一种序列密码算法，采用 128 位密钥和 128 位初始向量，通过线性反馈移位寄存器生成密钥流，与明文异或得到密文。ZUC 算法的核心加密公式如下：

$$C_i = P_i \oplus K_i \tag{4.2}$$

对视频帧加密的核心处理逻辑如图 4.1.3 所示。

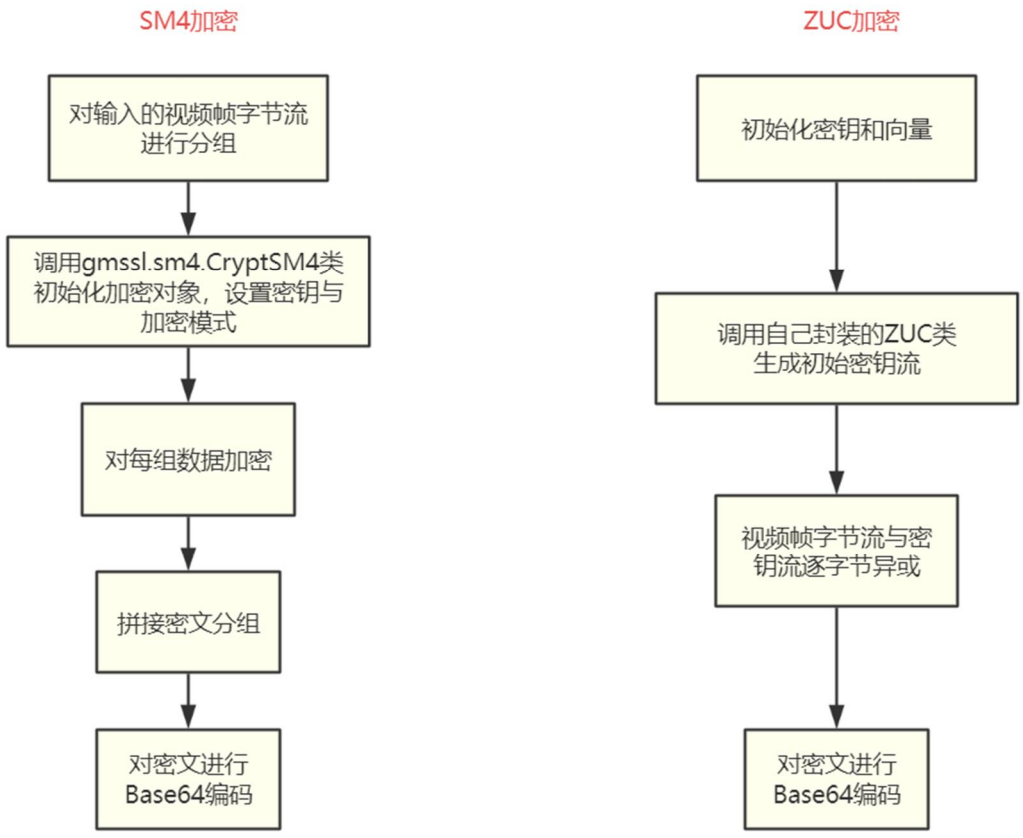


图 4.1.3 加密模块流程图

3. TCP 发送端模块实现

发送端 UI 界面包括 ip 输入框、加密方式选择框、显示视频区域和状态提示栏，以下是发送端未连接和已连接时的界面。

图 4.1.4 未连接时的状态

图 4.1.5 已连接时的状态

4.2 接收端设计与实现

4.2.1 接收端工作流程

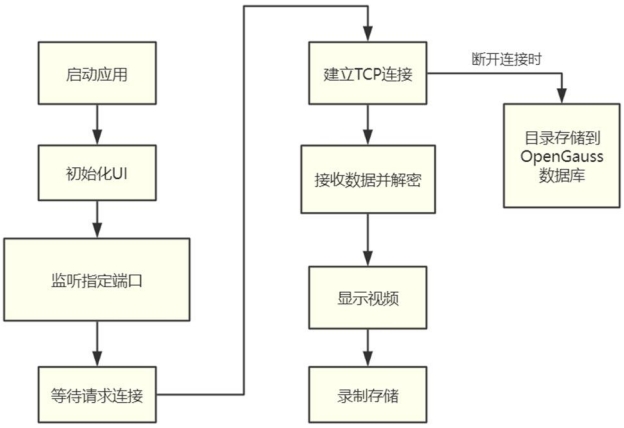


图 4.2.1 接收端工作流程

4.2.2 接收端核心模块实现

1. 解密模块实现

（一）SM4 解密

SM4 解密是加密的逆过程。SM4 解密算法的核心加密公式如下：

$$C = D(K, P) \tag{4.3}$$

对密文解密的核心处理逻辑如图 4.2.2 所示。

（二）ZUC 加密

ZUC 解密与加密采用相同的密钥流生成逻辑。对密文解密的核心处理逻辑如图 4.2.2 所示。

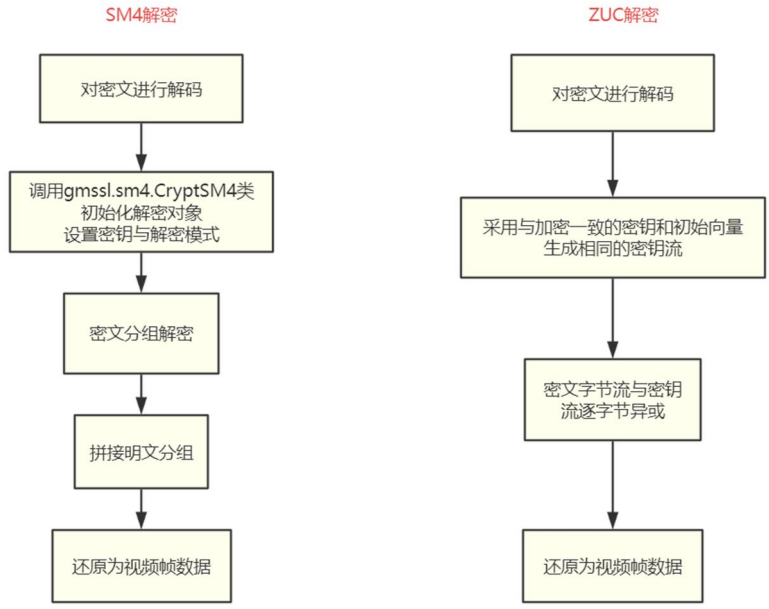


图 4.2.2 解密模块流程图

2. 视频显示与录制模块

视频显示模块是基于 PyQt5 的 QLabel 控件实现的，能够将 numpy 数组格式的视频帧转换为 QImage 格式进行显示；录制模块是基于 OpenCV 的 VideoWriter 类实现，能够将视频帧以 AVI 格式存储到本地。核心处理逻辑如图 4.2.3 所示。

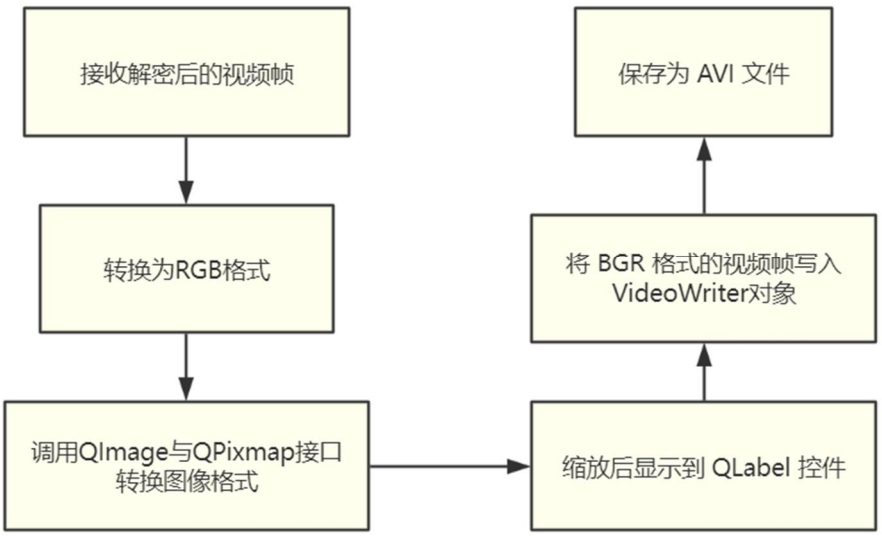


图 4.2.3 视频显示与录制模块流程图

3. TCP 接收端模块实现

接收端 UI 界面包括解密方式选择框、显示视频区域，以下是接收端未连接、接收到连接请求时和已连接的界面。

图 4.2.4 接收端未连接时的界面

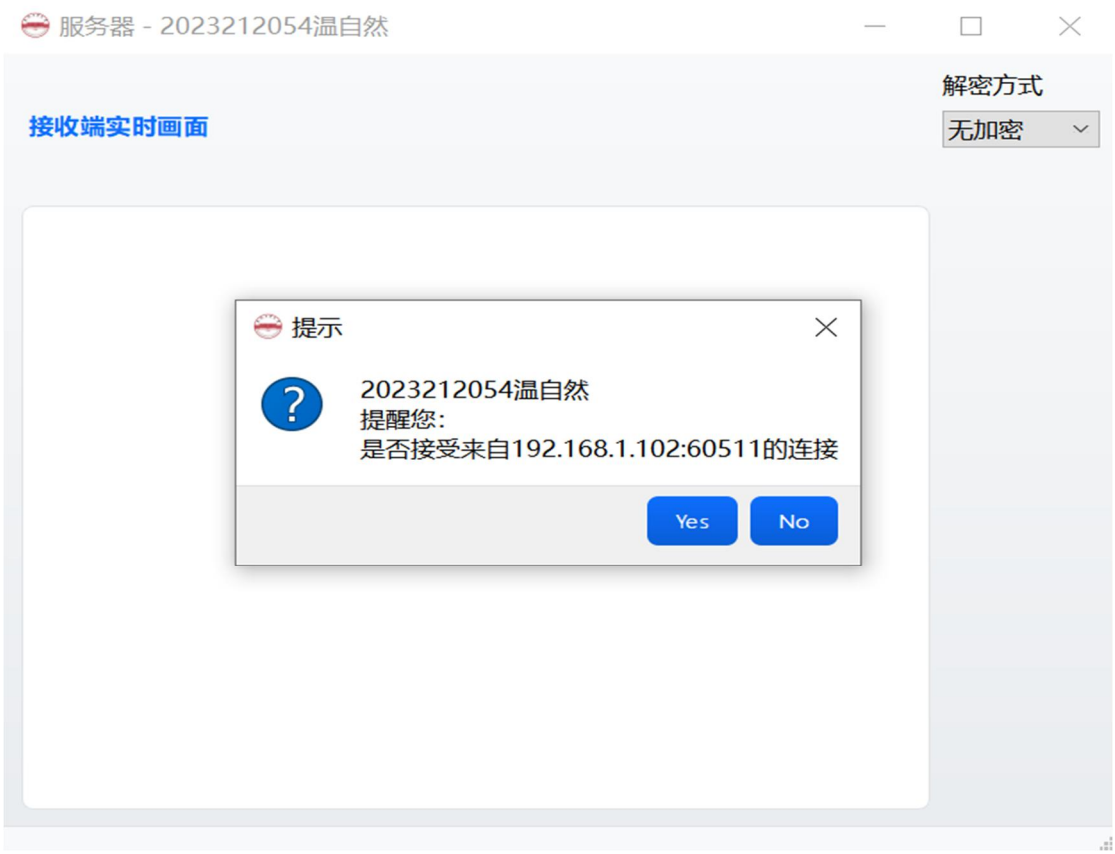


图 4.2.5 接收端接收到连接请求的界面

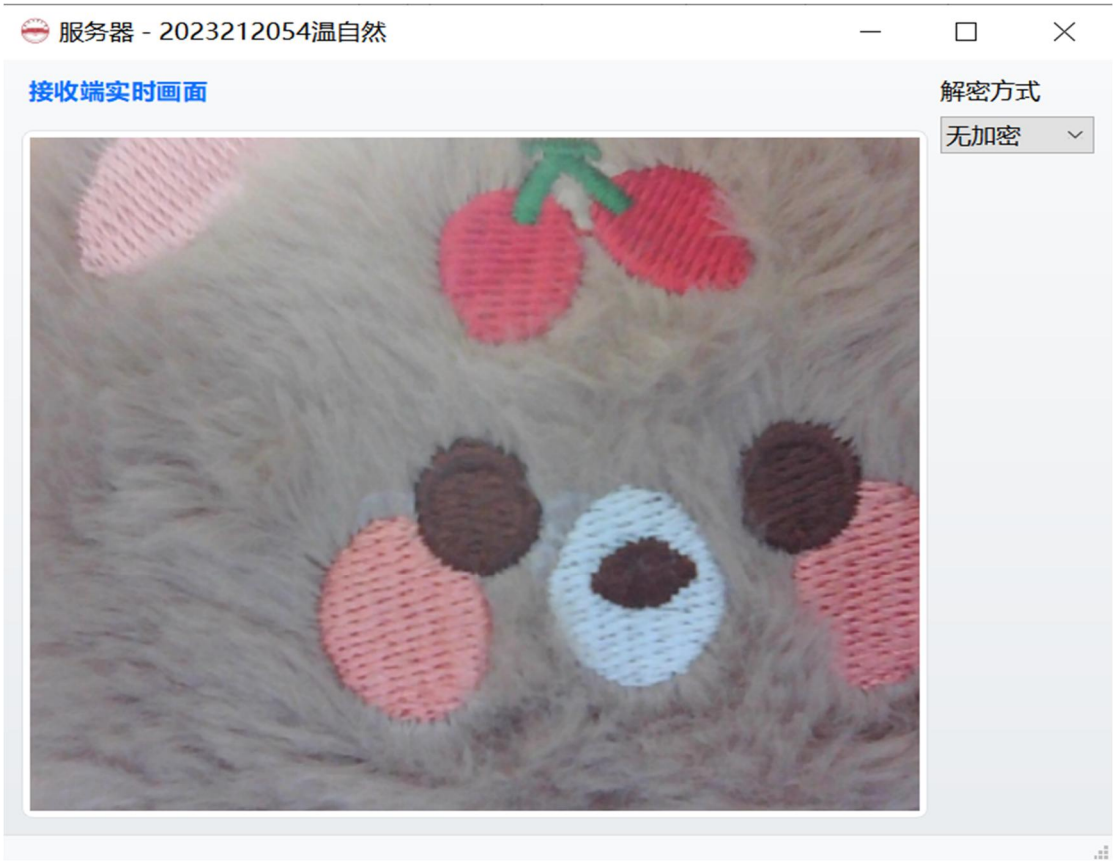


图 4.2.6 接收端已连接的界面

4. openGauss 数据库设计与实现

（1）数据库表设计

表 1 records 表设计

字段名	数据类型	约束	字段说明
id	serial	primary key	唯一标识
ip	text	not null	发送端 ip 地址
port	integer	not null	发送端端口号
start	timestamp	not null	视频传输开始时间
end	timestamp	not null	视频传输结束时间
duration	integer	not null	持续时间
path	text	not null	视频存放路径

（2）数据库连接与操作

通过 psycopg2 库实现连接数据库、插入数据的功能。数据库连接配置参数与代码流程如下：

表 2 数据库连接配置

配置	值	说明
dbname	postgres	数据库实例名
user	dboper	数据库用户名
password	dboper@123	数据库密码
host	192.168.1.129	数据库服务器 ip 地址
port	5432	数据库服务端口

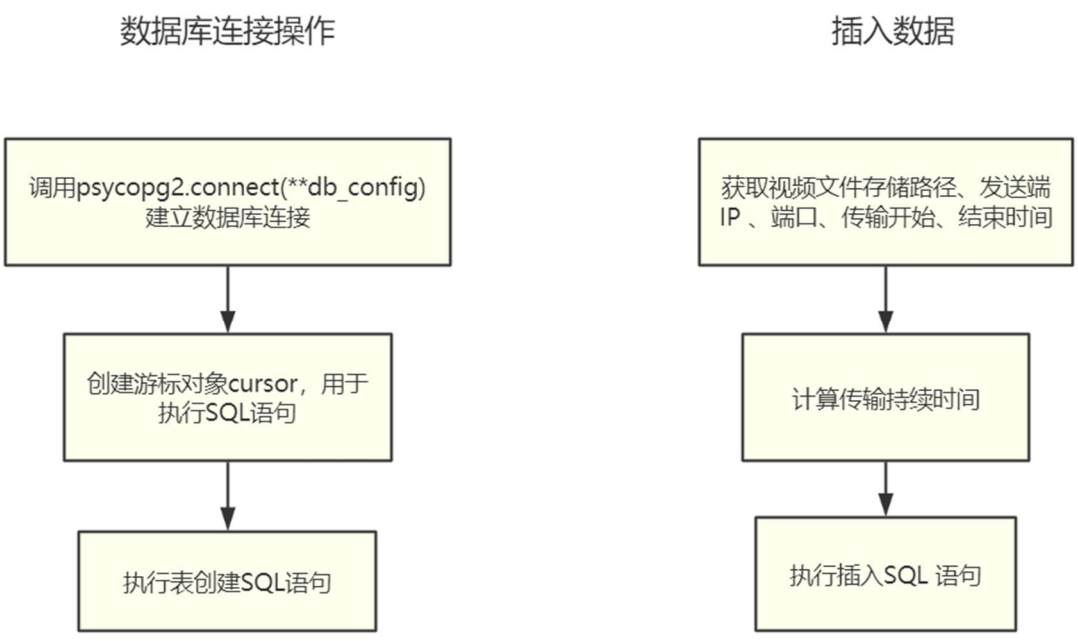


图 4.2.7 连接数据库、插入数据的代码流程图

	id	ip	port	start	end	duration	
1	180	192.168.1.104	50464	2026-01-03 18:56:48	2026-01-03 18:57:03	00:00:14.774669	D:\TCP-ZUC_SM
2	181	192.168.1.104	50492	2026-01-03 18:57:10	2026-01-03 18:57:17	00:00:06.905149	D:\TCP-ZUC_SM

图 4.2.8 数据库内容

4.3 系统测试

4.3.1 功能测试

（1）无加密情况下，能实现发送端发送视频数据并实时显示到区域中，接收端接收视频数据也显示到区域中，断开连接时，能存放视频，且将视频存放目录写入数据库中。

图 4.3.1 无加密时视频传输

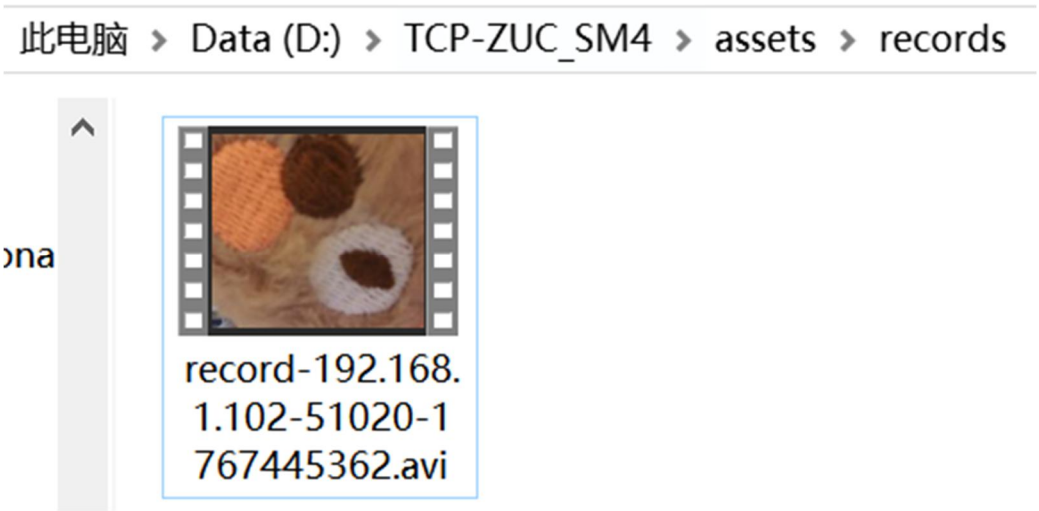


图 4.3.2 本地存放视频目录

	id	ip	port	start	end	duration	
1	195	192.168.1.102	51020	2026-01-03 21:02:42	2026-01-03 21:02:48	00:00:05.77671	D:\TCP-ZUC_SM

图 4.3.3 数据库内容

（2）SM4 加密情况下，也能实现同样的功能，且发现在 SM4 加密情况下，视

频明显比无加密情况下卡顿了。

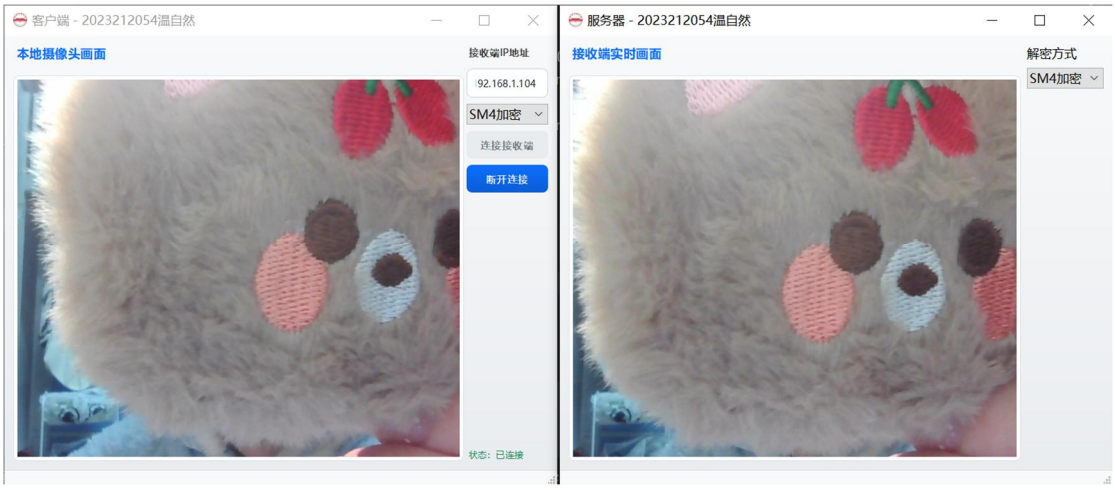


图 4.3.4 SM4 加密时视频传输

图 4.3.5 本地存放视频目录

	id	ip	port	start	end	duration	
1	195	192.168.1.102	51020	2026-01-03 21:02:42	2026-01-03 21:02:48	00:00:05.77671	D:\TCP-ZUC_SM
2	196	192.168.1.102	51156	2026-01-03 21:05:06	2026-01-03 21:05:14	00:00:07.648831	D:\TCP-ZUC_SM

图 4.3.6 数据库内容

（3）ZUC 加密情况下，能实现相同功能，视频卡顿情况比 SM4 加密时缓解了，但依旧比无加密情况下卡顿。

图 4.3.7 ZUC 加密时视频传输

图 4.3.8 本地存放视频目录

	id	ip	port	start	end	duration	
1	195	192.168.1.102	51020	2026-01-03 21:02:42	2026-01-03 21:02:48	00:00:05.77671	D:\TCP-ZUC_SM
2	196	192.168.1.102	51156	2026-01-03 21:05:06	2026-01-03 21:05:14	00:00:07.648831	D:\TCP-ZUC_SM
3	197	192.168.1.102	51227	2026-01-03 21:06:44	2026-01-03 21:06:48	00:00:04.162262	D:\TCP-ZUC_SM

图 4.3.9 数据库内容

4.3.2 性能测试

为验证 SM4、ZUC 两种国密算法在实时视频加密场景下的性能差异，选取 5000 帧分辨率为 640×480 的摄像头视频数据作为测试样本，分别统计单帧数据经两种算法加密的耗时，并计算 5000 帧加密的平均耗时，对比分析两种算法的加密效率与实时性表现。

（1）在开发板上加密耗时对比

由表 3 可知，ZUC 单帧耗时仅为 SM4 的 51.7%，两者加密耗时差异是因为算法不同。SM4 是分组密码，需将视频帧按 16 字节分组并执行 32 轮非线性变换，分组与多轮运算导致硬件资源占用高；ZUC 是序列密码，通过线性反馈移

位寄存器生成密钥流后直接异或，无需分组，单字节处理逻辑简单，计算开销小，硬件资源占用少于 SM4 加密时的情况。

表 3 在开发板上两种算法加密耗时表

加密算法	平均耗时(ms/帧)
SM4	235.321
ZUC	121.584

（2）本地加密耗时对比

由表 4 可知，ZUC 算法在 PC 端仍保持显著效率优势，单帧耗时仅为 SM4 的 48.1%。对比表 3 与表 4 的耗时数据可得，开发板 SM4、ZUC 加密效率远高于 PC 端。我认为原因是，开发板凭借硬件加速、嵌入式系统轻量化以及加密线程独占资源得到更短的耗时，PC 端则是因为多任务调度、CPU 指令集适配不足导致耗时高于开发板。

表 4 在本地 PC 端两种算法加密耗时表

加密算法	平均耗时(ms/帧)
SM4	1907.593
ZUC	917.654

（3）不同加密算法下视频传输时长偏差测试

由表 5 可知，三种传输模式下均存在时长偏差，我认为差异主要是因为加密耗时与传输耗时。无加密模式时长仅受 TCP 传输固有丢包影响；SM4 加密因加密耗时最长，叠加传输耗时，帧丢失最严重；ZUC 加密效率优于 SM4，帧丢失情况相对缓解。

表 5 不同加密算法下视频时长对比

加密算法	原视频时长	接收到的视频时长（发送端是开发板）	接收到的视频时长（发送端是 PC 端）
无加密	60s	45s	45s
SM4	60s	18s	2s
ZUC	60s	39s	4s

五、课程设计总结与心得体会

5.1 设计总结

本次课程设计已顺利完成全部核心任务，成功实现了视频采集、国密加密、TCP 传输、解密还原以及 openGauss 数据库存储的完整流程。系统支持 SM4 和 ZUC 两种国密算法的切换，通过 TCP 协议保障视频数据的可靠传输，同时利用

openGauss 数据库对视频文件目录进行管理。经过多轮测试验证，无加密、SM4 加密、ZUC 加密三种模式均能稳定运行。

本次课程设计的创新之处主要有三点。一是在发送端采用多线程[5]的设计，在发送数据的同时能实时显示视频，二是在接收端采用独立的解密线程，让数据接收与解密操作同步进行，能缓解解密耗时对传输实时性的影响；三是实现了两种加密模式，既可选择无加密，也可以选择 SM4 加密或者 ZUC 加密，能满足不同场景的使用需求。

在设计过程中，我也发现了一些不足之处。首先，SM4 算法的加密耗时相对较长，在传输高分辨率视频时，会出现明显的卡顿现象；其次，TCP 传输的丢包处理仅依赖协议本身的机制，没有额外设计应用层的重传功能，在网络环境复杂的情况下，可能会出现帧丢失的问题；最后，数据库目前只存储了视频文件的目录信息，没有校验文件的完整性，无法确认存储的文件是否存在损坏。

针对这些问题，后续的改进思路如下。对于 SM4 加密效率偏低的问题，可以通过引入视频帧分块并行加密的方法，将视频帧拆分后再同时进行加密处理，提升整体的加密速度；针对传输稳定性的问题，可以在应用层为每个视频帧添加序号标识，一旦检测到丢帧，就触发重传机制，减少丢包对视频完整性的影响；对于数据库功能不足的问题，可以在数据表中新增视频文件 MD5 校验字段，在存储文件目录的同时记录对应的校验值，之后通过对比校验值就能验证文件是否完整。

5.2 心得体会

通过本次信息系统软件设计课程，我的实践能力得到了提升。在技术层面上，我不再局限于 PPT 里的理论知识，而是深入理解了 SM4 与 ZUC 两种国密算法的核心逻辑，同时掌握了 OpenCV 视频采集、PyQt5 UI 界面开发[6]、TCP 协议网络编程以及 openGauss 数据库的基本操作。

在实践过程中，我形成了更有条理的问题解决思路。在面对复杂需求时，可以先通过流程图梳理逻辑，再将核心问题拆分为一个个可解决的小问题，逐一突破。在这次设计中，我遇到最大的挑战是，在 Linux 环境下，摄像头初始化失败与解密线程阻塞的问题。经过反复调试和资料查阅，最终定位到摄像头端口占用、路径格式不兼容的问题。通过新增独立解密线程、调整 Linux 系统专属适配参数的方式，成功解决了这些问题。这一过程让我深刻体会到，软件开发不仅需要扎实的技术功底，更需要细致的调试能力和坚持不懈的解决问题的态度。

此外，本次课程设计也让我对信息安全领域有了更直观且深入的认知。国密算法作为保障数据安全的核心技术，在实际应用中并非只追求绝对安全，而是需要在安全性与实用性之间找到平衡。

总而言之，这次课程设计不仅提升了我的技术实践能力，更提高了我的问题

解决能力，让我真切地感受到理论与实践相结合的重要性。

六、参考文献

[1]宋玮,顾国生,欧毓毅.面向 openGauss 数据库的实验课程教学探索[J].电脑知识与技术,2025,21(14):152-155.

[2]燕杰.运用 TCP 协议原理解决音频业务时延高问题[J].中国新通信,2023,25(21):40-43.

[3]林秋琼.基于轻量级 SM4 算法的通信链路安全通信方法研究[J].数据通信,2025,(05):14-17+22.

[4]张宇鹏,高莹,严宇,等.ZUC 算法软件快速实现[J].密码学报,2021,8(03):388-401.

[5]王鑫,扈月,刘坤禹,等.Linux 下多线程的方案实现与对比[J].信息记录材料,2024,25(03):246-248.

[6]林健敏.基于 PYQT5 的自助登记系统设计与实现[J].数字技术与应用,2025,43(02):217-219.

合肥工業大學

信息系统软件设计

报告

学　　院　计算机与信息学院

专　业　班　级　电子信息工程 23 级-2 班

学生姓名及学号　仲俊伟　2023212064

指　导　教　师　苏兆品　张国富　李小红　臧怀娟

课　题　名　称　数据库+TCP+音视频传输水印系统

2026 年 1 月 7 日

一、课题概述

随着 5G、云计算技术的普及，音视频数据的传播规模呈指数级增长，2024 年全球音视频数据流量占比已超互联网总流量的 80%。但随之而来的版权侵权、内容篡改等问题日益突出，据统计，影视、教育等行业的音视频盗版率高达 35%，造成的经济损失每年超千亿元。传统音视频传输系统仅关注数据的完整性传输，缺乏对内容归属的有效追溯机制，无法满足数字版权保护的核心需求。

本课题聚焦音视频传输中的安全与版权保护问题，设计实现“数据库 + TCP + 音视频传输水印系统”。系统融合计算机视觉、音视频信息安全与国产数据库技术，通过在音视频流中嵌入加密水印，实现内容的唯一标识；借助 TCP 协议保障传输可靠性，利用 openGauss 数据库存储文件路径与操作日志，形成“采集 - 加密 - 嵌入 - 传输 - 提取 - 解密 - 存储” 的全流程闭环。该系统可广泛应用于在线教育、影视传播、远程会议等场景，为音视频内容提供版权认证、篡改检测与溯源能力，填补传统传输系统在版权保护方面的空白。

二、课题任务

本系统需完成发送端与接收端的双向协同设计，核心任务与目标如下：

1. 音视频采集：实现摄像头实时视频捕捉（分辨率≥320×240，帧率≥15fps）与麦克风音频采集（采样率≥16kHz，单声道），支持设备异常检测与提示；

2. 水印处理：设计自定义水印信息（支持学号、版权标识等文本），通过国密 SM4 算法加密水印，分别采用 DCT 变换、LSB 最低有效位两种视频水印技术，以及回声隐藏音频水印技术完成水印嵌入；

3. 可靠传输：基于 TCP 协议设计点对点传输架构，采用多线程技术实现音视频流与水印数据的并行传输，确保无丢包、低延迟（传输延迟≤100ms）；

4. 水印提取与解密：接收端实时提取音视频中的隐藏水印，通过 SM4 算法解密，还原原始水印信息，提取准确率≥90%；

5. 数据存储：基于 openGauss 数据库设计数据表，存储音视频文件路径、元数据（大小、时长、分辨率等）、水印操作日志（原始信息、加密信息、提取结果）及性能评估指标；

6. 性能评估：计算并对比不同水印算法的 PSNR（峰值信噪比）、SSIM（结构相似性）、BER（误码率）指标，分析算法优缺点；

7. 可视化交互：设计友好的 UI 界面，支持音视频实时预览、水印参数配置、传输控制、录制保存及数据库信息查询。

三、技术方案及关键问题

3.1 技术路线

本系统采用模块化设计思想，技术路线围绕 “数据流转” 核心展开，分为发送端处理流程、传输流程、接收端处理流程与数据库交互流程四大环节，具体如下图 3.1 所示：

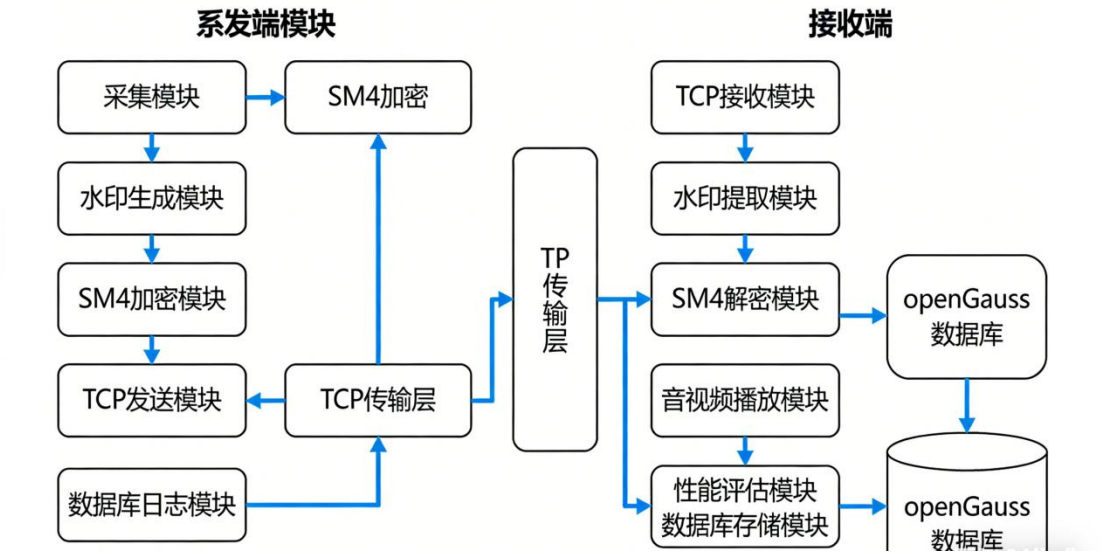


图 3.1

核心技术选型依据：

(1)音视频采集：OpenCV（成熟的计算机视觉库，支持跨平台摄像头操作）、PyAudio（轻量级音频处理库，适配多种麦克风设备）；

(2)水印技术：DCT 变换（鲁棒性强，适配版权保护场景）、LSB 算法（实时性优，适配低延迟需求）、回声隐藏（音频水印隐藏性好）；

(3)加密技术：国密 SM4 算法（符合国家密码标准，安全性高，适配国内应用场景）；

(4)数据库：openGauss（国产高性能数据库，支持高并发数据写入，符合课题对国产技术的采用要求）；

(5)UI 设计：PyQt5（支持图形化交互，适配音视频实时渲染与控件布局）。

3.2 关键问题及解决方案

(1)水印嵌入与音视频质量的平衡问题

问题：水印嵌入强度过高会导致音视频失真，强度过低则降低水印鲁棒性，易被压缩、裁剪等操作破坏；解决方案：针对 DCT 水印，将嵌入强度控制为 0.05（基于多次测试验证，该值可使 PSNR≥35dB，满足视觉无失真要求），仅调整 8×8DCT 块的（3,4）（4,3）非低频系数，避免影响图像主体；LSB 水印仅替换像素最低 1 位，确保图像质量损失可忽略；音频回声隐藏采用延迟 100 帧、衰减系数 0.5 的参数配置，主观听觉无明显回声。

(2)多线程传输的并发控制问题

问题：音视频采集与传输共用单线程会导致数据阻塞，出现音视频不同步、卡顿现象；解决方案：设计独立的视频线程与音频线程，发送端通过 PyQt5 的 QThread 类创建子线程，分别负责视频采集 - 嵌入 - 发送与音频采集 - 嵌入 - 发送；线程间通过信号槽机制（pyqtSignal）实现数据同步，避免资源竞争；接收端同理，采用独立线程处理视频与音频的接收 - 提取 - 播放。

(3)水印提取的准确率保障问题

问题：传输噪声、音视频压缩会导致水印信号失真，影响提取准确率；解决方案：SM4 加密时采用 PKCS7 填充机制，确保解密完整性；DCT 水印提取时通过块匹配算法定位嵌入位置，过滤噪声干扰；LSB 水印提取时采用位运算精准提取最低有效位；音频回声隐藏提取时通过自相关函数检测回声信号，提高抗干扰能力。

四、系统设计实现及测试

4.1 系统整体架构设计

系统采用 “客户端 - 服务器（C/S）” 架构，分为发送端（Client）、接收端（Server）与数据库服务三大组件，组件间通过 TCP 协议与数据库连接协议通信，整体架构如下图 4.1 所示：

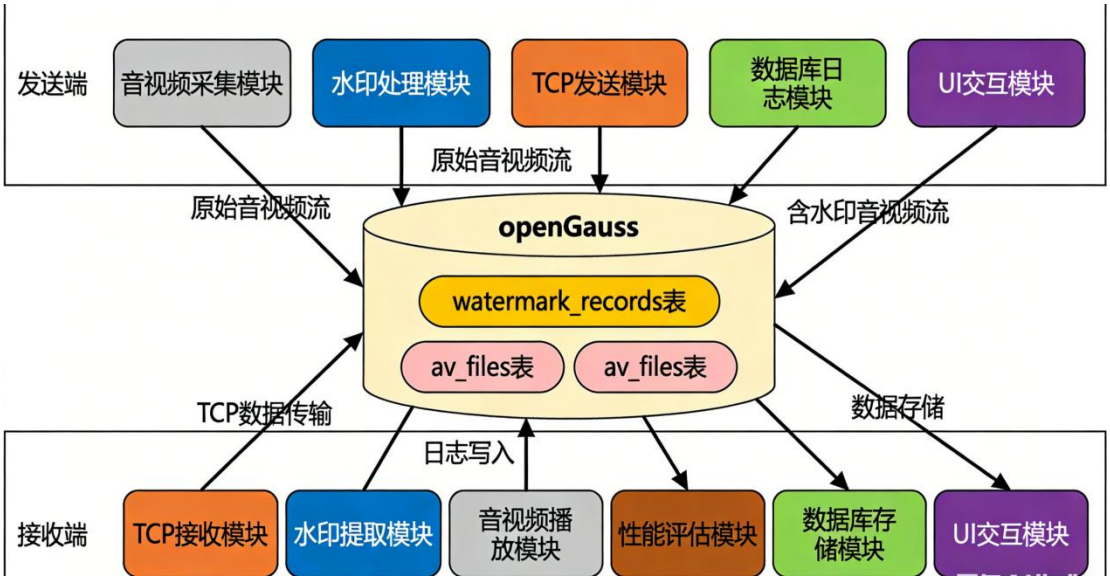


图 4.1

架构设计说明：发送端与接收端为对等架构，可部署在同一设备或不同设备，

通过 IP 地址与端口建立连接；数据库为独立服务，发送端与接收端通过 JDBC 连接，实现数据的实时写入与查询；所有模块采用松耦合设计，便于单独调试与功能扩展（如后续增加新的水印算法）。

4.2 核心模块设计实现

（1）视频采集子模块

视频采集基于 OpenCV 的VideoCapture类实现,核心设计流程如下图4.2所示：

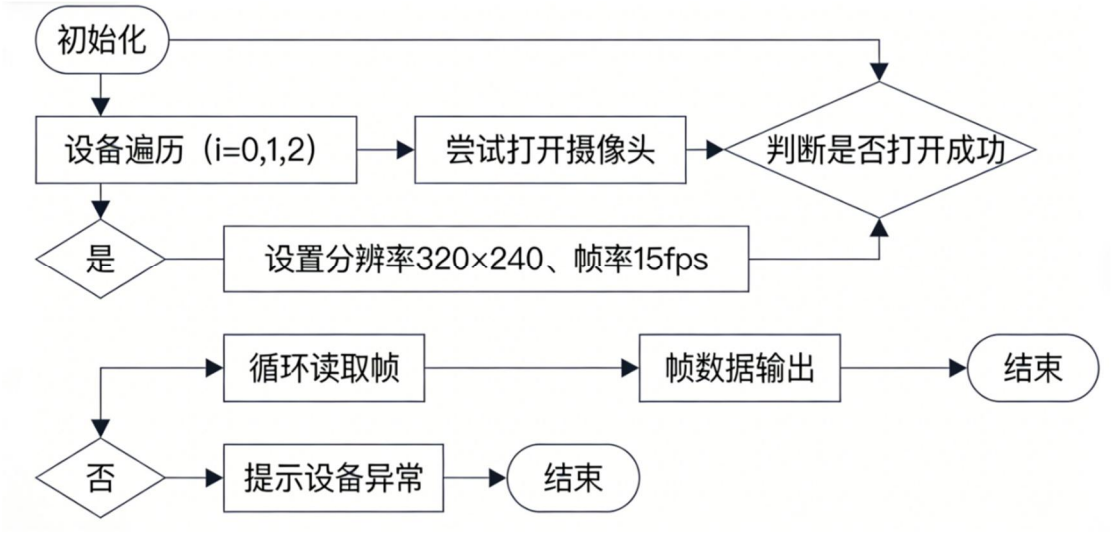


图 4.2

设备遍历：支持 0-2 号摄像头设备遍历，解决单设备占用或故障导致的采集失败问题；

参数配置：通过 cap.set(cv2.CAP_PROP_FRAME_WIDTH, 320)与
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 240)设置分辨率，
cap.set(cv2.CAP_PROP_FPS, 15)设置帧率，确保采集稳定性；

异常处理：当 ret=False（帧读取失败）时，输出日志并尝试重新打开摄像头；无可用摄像头时，在 UI 界面显示“摄像头未连接”提示。

（2）音频采集子模块

音频采集基于 PyAudio 库实现：

参数选型：16000Hz 采样率兼顾音质与传输效率，512 帧缓冲区平衡延迟与性能，单声道减少数据传输量；

异常捕获：通过 try-except 捕获设备访问权限不足、无麦克风设备等异常，在 UI 界面显示具体错误信息（如“音频设备初始化失败：无可用麦克风”）。

（3）水印加密模块（SM4 国密算法）

SM4 算法采用 CBC 加密模式，核心设计如下：

密钥与 IV：密钥长度固定 16 字节，通过 key.ljust(16)[:16].encode()确保密钥格式合规；IV（初始向量）为 16 字节 0 填充（b'\x00' * 16），保证加密一致性；

加密流程：水印文本→UTF-8 编码→PKCS7 填充（填充长度为 AES.block_size - 数据长度%AES.block_size）→SM4 加密→Base64 编码（便于二进制数据传输）如下图 4.3 所示：

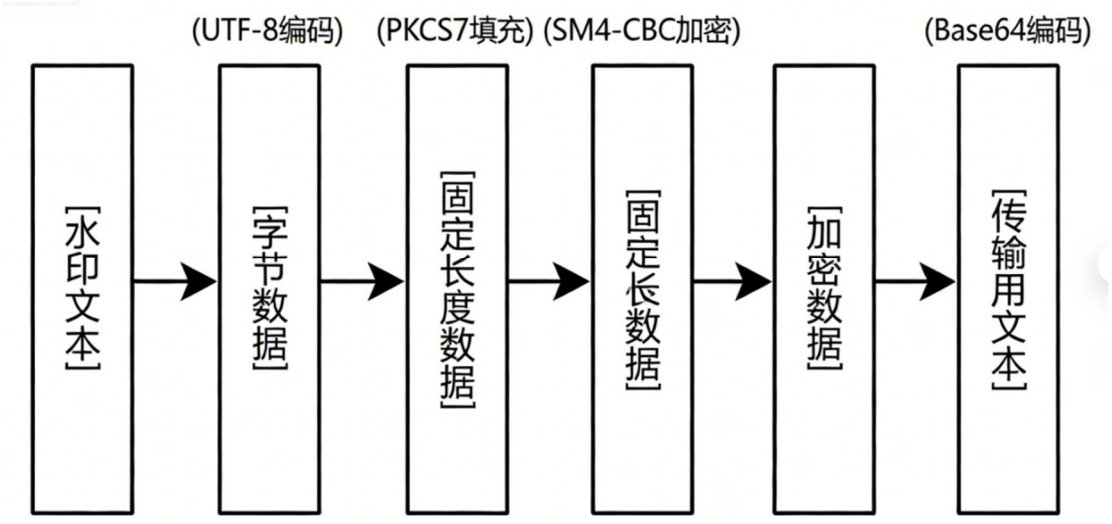


图 4.3

解密流程：Base64 解码→SM4 解密→PKCS7 去填充→UTF-8 解码→原始水印文本；

核心函数：封装 SM4 类，包含 encrypt(data)与 decrypt(enc_data)方法，分别实现加密与解密逻辑，函数参数为字符串，返回值为处理后的字符串，便于模块间调用。

（4）水印嵌入模块

1.DCT 视频水印嵌入

核心原理：利用图像 DCT 变换的频域特性，将水印嵌入中高频系数，既保证隐藏性又具备一定鲁棒性，流程图如下图 4.4

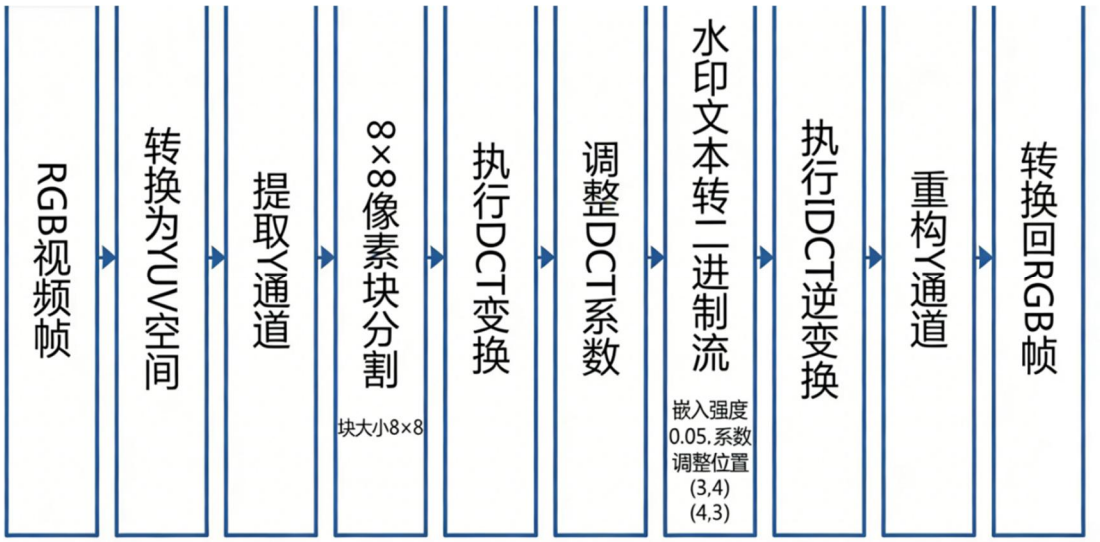


图 4.4

嵌入流程：RGB 视频帧→转换为 YUV 空间（分离亮度通道 Y 与色度通道 U、V）→提取 Y 通道→按 8×8 像素块分割→对每个块执行 DCT 变换（cv2.dct(block)）→将水印文本转换为二进制流→根据二进制位调整 DCT 系数（“1”则 dct_block[3,4] += 0.05*abs(dct_block[3,4])，“0”则减去对应值）→执行 IDCT 逆变换（cv2.idct(dct_block)）→重构 Y 通道→转换回 RGB 帧；

关键参数：块大小 8×8（兼顾嵌入容量与计算效率），嵌入强度 0.05（经测试，

该值可平衡鲁棒性与图像质量）。

2.LSB 视频水印嵌入

核心原理：利用图像像素最低有效位（LSB）对人眼不敏感的特性，将水印二进制流替换像素 LSB 位，实现隐藏。流程图如下图 4.5 所示：

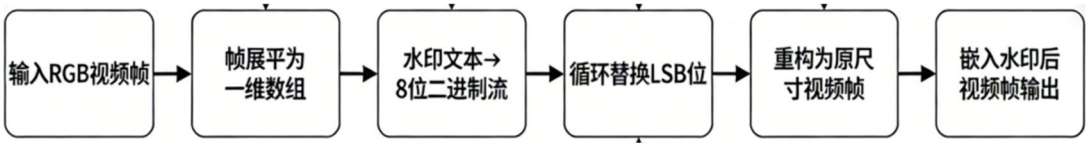


图 4.5

嵌入流程：水印文本→转换为 8 位二进制流（''.join(format(ord(c), '08b') for c in watermark_info)）→将视频帧展平为一维数组→循环遍历数组，按二进制位替换像素 LSB 位（“1”则 pixel_value | bit_mask，“0”则 pixel_value & (~bit_mask)，bit_mask=1）→重构为原尺寸视频帧。

关键优势：计算量小，实时性强，对图像质量影响极小（PSNR≥40dB）。

3. 回声隐藏音频水印嵌入

核心原理：通过在音频信号中添加延迟回声，用“有无回声”对应二进制水印位，实现音频水印隐藏。流程图如下图 4.6 所示：

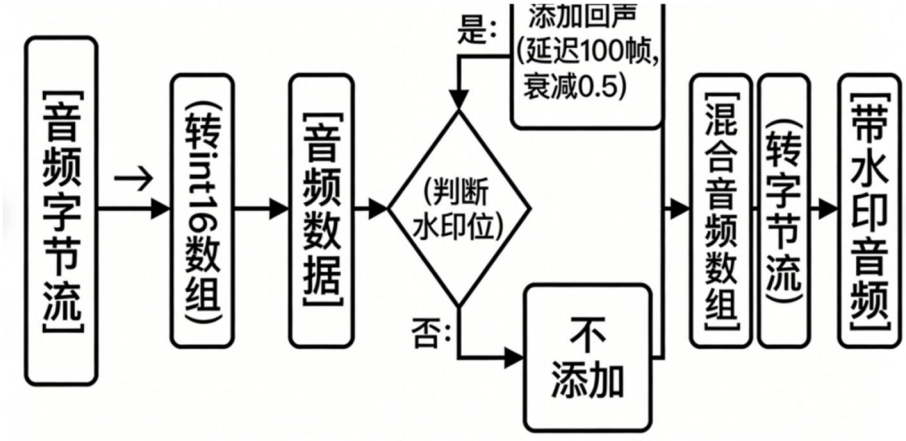


图 4.6

嵌入流程：音频数据（字节流）→转换为 numpy 数组（dtype=np.int16）→判断水印位（“1”则添加回声，“0”则不添加）→回声生成（echo[self.delay:] = audio_array[:-self.delay] * self.decay）→音频数组与回声叠加→转换回字节流。

关键参数：延迟 delay=100 帧（适配 16kHz 采样率，回声延迟约 6.25ms，人耳不易察觉），衰减系数 decay=0.5（降低回声强度，避免影响原始音频）。

（5）水印提取模块

1.DCT 视频水印提取

提取流程：接收的 RGB 帧→转换为 YUV 空间→提取 Y 通道→8×8 块分割→DCT 变换→计算 DCT 系数（3,4）与（4,3）的大小关系（大于则为“1”，否则为“0”）→拼接二进制流→转换为文本（每 8 位为一个字符）→提取的水印文本。

适配处理：考虑传输与压缩导致的系数波动，提取时允许一定误差，当系数差值绝对值小于阈值（0.01×abs(dct_block[3,4])）时，默认按原始嵌入位处理。

2. LSB 视频水印提取

提取流程：接收的视频帧→展平为一维数组→循环遍历数组，提取每个像素的 LSB 位（pixel_value & bit_mask）→拼接二进制流→每 8 位转换为一个字符→提取的水印文本。

关键优化：提取时忽略像素值的高位变化，仅关注 LSB 位，确保提取逻辑与嵌入逻辑一致。

3.回声隐藏音频水印提取

提取流程：音频数据（字节流）→转换为 numpy 数组→计算自相关函数（np.correlate(audio_array, audio_array, mode='full')）→检测延迟位置

（delay=100）的自相关值→若自相关值大于阈值（autocorr[1]＊0.1），则为“1”，否则为“0”→提取的水印位。

阈值设计：通过统计无回声音频的自相关值，设置合理阈值，降低误判率。

（6）TCP 传输模块：

传输模块采用 Socket 多线程设计，发送端与接收端的端口绑定如下：视频传输端口 12345，音频传输端口 12346，避免端口冲突。

1.发送端传输设计

多线程创建：通过 QThread 创建 CameraThread（视频线程）与 AudioThread（音频线程），独立运行。

数据打包（视频）：为确保接收端准确解析，数据包结构设计如下：| 总长度（4 字节） | 水印长度（4 字节） | 加密水印数据（N 字节） | 帧宽（4 字节） | 帧高（4 字节） | 帧率（4 字节） | 视频帧数据（M 字节）|其中，总长度=N+M+20（固定字段长度），视频帧数据为 JPEG 编码（cv2.imencode('.jpg', frame, [cv2.IMWRITE_JPEG_QUALITY, 70])),编码质量 70（平衡传输效率与图像质量）。

发送逻辑：通过 socket.sendall() 发送打包数据，每次发送前用 struct.pack('!I', data_len)将整数转换为网络字节序（大端序），确保跨平台兼容性。

连接管理：线程启动时创建 Socket 对象，调用 connect(('localhost', port)) 建立连接，连接成功后通过信号槽通知 UI 更新状态；连接失败则输出日志并提示。

3. 接收端传输设计

监听与连接：接收端启动时，Socket 对象调用 bind(('0.0.0.0', port))绑定端口，listen(1)监听连接请求，accept()阻塞等待客户端连接，连接成功后返回连接对象与客户端地址。

数据解析：按发送端打包格式，依次读取总长度、水印长度、加密水印数据、帧宽、帧高、帧率、视频帧数据，通过 struct.unpack('!I', data)解析整数字段，确保数据完整性。

缓冲区设计：采用 4096 字节缓冲区（recv(min(4096, remaining_len))），循环读取数据直至达到指定长度，避免数据包截断。

（7） 数据库模块：

基于 openGauss 数据库设计两张核心表，用于存储水印操作日志与音视频文件信息，数据库连接配置如下：database='postgres'，user='ht'，password='Mysql@1234'，host='192.168.43.192'，port=15000。

1. 水印操作日志表（watermark_records）

核心代码如下图 4.7

```
if not watermark_records_exists:
    create_watermark_table = """
    CREATE TABLE watermark_records (
        id SERIAL PRIMARY KEY,
        student_id VARCHAR(20) NOT NULL,
        watermark_type VARCHAR(10),
        algorithm_type VARCHAR(50),
        operation VARCHAR(20),
        original_info TEXT,
        encrypted_info TEXT,
        extracted_info TEXT,
        decrypted_info TEXT,
        psnr FLOAT,
        ssim FLOAT,
        ber FLOAT,
        create_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP
    )
    """
    cursor.execute(create_watermark_table)
    print("水印表创建成功")
else:
    print("水印表已存在")
```

图 4.7

表结构：包含 12 个字段，主键 id（自增），student_id（学号，关联开发者），watermark_type（水印类型：video/audio），algorithm_type（算法类型：

dct/lsb/echo_hiding），operation（操作类型：embed/extract/record），original_info（原始水印信息），encrypted_info（加密后水印信息），extracted_info（提取的水印信息），decrypted_info（解密后水印信息），psnr/ssim/ber（性能指标），create_time（操作时间戳，默认当前时间）。

设计目的：记录水印全流程操作数据，便于算法性能分析与问题追溯。

2. 音视频文件信息表（av_files）

核心代码如下图 4.8

```
# 创建av_files表来存储音视频文件信息
if not av_files_exists:
    create_av_files_table = """
    CREATE TABLE av_files (
        id SERIAL PRIMARY KEY,
        student_id VARCHAR(20) NOT NULL,
        file_name VARCHAR(255) NOT NULL,
        file_type VARCHAR(20) NOT NULL,  -- video, audio
        file_path TEXT NOT NULL,
        file_size BIGINT,  -- 文件大小（字节）
        duration FLOAT,  -- 持续时间（秒）
        resolution VARCHAR(50),  -- 视频分辨率
        sample_rate INT,  -- 音频采样率
        channels INT,  -- 音频通道数
        watermark_info TEXT,  -- 文件中的水印信息
        algorithm_type VARCHAR(50),  -- 使用的水印算法
        create_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
        update_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP
    )
    """
    cursor.execute(create_av_files_table)
    print("av_files表创建成功")
else:
    print("av_files表已存在")
```

图 4.8

表结构：包含 14 个字段，主键 id（自增），student_id（学号），file_name（文件名），file_type（文件类型：video/audio），file_path（文件存储路径），file_size（文件大小，字节），duration（文件时长，秒），resolution（视频分辨率），sample_rate（音频采样率），channels（音频通道数），watermark_info（嵌入的水印信息），algorithm_type（水印算法），create_time/update_time（创建/更新时间戳）。

设计目的：存储音视频文件的元数据与存储路径，支持文件查询与管理。

3. 数据库操作封装

封装 DatabaseManager 类，核心方法如下：
connect()：创建并返回数据库连接对象，内置异常捕获，连接失败则输出错误日志。
create_tables()：启动时检查表是否存在，不存在则创建上述两张表，确保数据库结构合规。
log_watermark_operation(...)：接收 11 个参数（对应 watermark_records 表字段），执行 INSERT 语句写入水印操作日志，支持特殊字符过滤（clean_string 函数）。
log_av_file(...):接收 11 个参数(对应 av_files 表字段),执行 INSERT/UPDATE 语句（文件已存在则更新，否则插入），写入文件信息。
get_av_files(student_id=None, file_type=None)：查询 av_files 表数据，支持按学号、文件类型筛选，返回查询结果列表，用于 UI 界面文件列表展示。
（8）UI 交互模块
基于 PyQt5 设计发送端与接收端 UI，采用模块化布局（QVBoxLayout、QHBoxLayout），确保界面美观且功能清晰。

1. 发送端 UI 设计

界面尺寸：固定 450×700 像素，适配桌面显示。

核心区域：视频显示区：QLabel 控件（320×240），背景黑色，实时渲染采集的视频帧，支持缩放显示；水印控制区：QGroupBox 包含水印内容输入框（QLineEdit，默认显示学号“2023212064”）、算法选择下拉框（QComboBox，选项“DCT”“LSB”）、“嵌入水印”按钮，以及水印状态显示标签；系统状态区：显示连接状态、音频状态、录制状态、数据库状态，通过颜色区分状态（绿

色=正常，红色=异常，蓝色=连接中）；控制按钮区：“开始连接”“停止连接”“开始录制”“停止录制”四个按钮，最小高度 40 像素，便于操作。UI 界面如下图 4.9

图 4.9

交互逻辑：输入水印内容后点击“嵌入水印”生效；点击“开始连接”启动音视频线程与 TCP 连接；“开始录制”将音视频保存为 AVI（视频）/WAV（音频）文件，存储路径为程序运行目录。

2.接收端 UI 设计

界面尺寸：固定 500×800 像素，比发送端增加评估指标与文件查看区域。

核心区域：视频显示区：与发送端一致，实时渲染接收的视频帧；水印信息区：显示提取的水印信息与解密后的原始信息，背景色区分（浅灰=提取信息，浅蓝=解密信息）；评估指标区：横向布局显示 PSNR、SSIM、BER 数值，实时更新；音频水印区：显示音频水印提取结果；系统状态区：显示连接状态、录制状态、数据库状态，新增“最近文件”显示标签；控制按钮区：“开始监听”“停止监听”“开始录制”“停止录制”“查看文件”五个按钮，“查看文件”可弹出对话框显示数据库中的文件列表。UI 界面如下图 4.10

图 4.10

交互逻辑：点击“开始监听”启动 Socket 监听，等待客户端连接；“查看文件”调用 get_av_files()方法查询数据，格式化显示文件名、类型、大小、时间等信息。

4.3 系统测试与结果分析

4.3.1 功能测试

功能测试覆盖全流程关键节点，采用“单模块测试→集成测试→全流程测试”的步骤，确保系统各功能模块独立可用、集成兼容、全流程稳定，具体测试结果如下：

1.数据库连接模块

测试用例包括两项：一是在终端启动数据库并查看连接状态如下图 4.11，二是在 UI 客户端和服务端界面验证数据库连接情况如下图 4.12、图 4.13。预期结果为终端显示数据库连接成功，且 UI 界面明确提示服务端与客户端已成功连接数据库。实际测试结果符合预期，数据库连接功能正常实现。

```
[omm@hostName1 ~]$ gs_om -t start

Starting cluster.
=========================================
[SUCCESS] hostName1
2026-01-06 15:01:32.209 695cb34c.1 [unknown] 255086221430816 [unknown] 0 dn_6001
 01000  0 [BACKEND] WARNING:  could not create any HA TCP/IP sockets
2026-01-06 15:01:32.210 695cb34c.1 [unknown] 255086221430816 [unknown] 0 dn_6001
 01000  0 [BACKEND] WARNING:  could not create any HA TCP/IP sockets
2026-01-06 15:01:32.247 695cb34c.1 [unknown] 255086221430816 [unknown] 0 dn_6001
 01000  0 [BACKEND] WARNING:  Failed to initialize the memory protect for g_inst
ance.attr.attr_storage.cstore_buffers (1024 Mbytes) or shared memory (4601 Mbyte
s) is larger.
=========================================
Successfully started.
[omm@hostName1 ~]$
```

图 4.11

图 4.12

图 4.13

2.水印加密与嵌入模块

测试执行三项操作：输入水印 “2023212064” 如下图 4.14，后选择 DCT 算法进行加密嵌入如下图 4.14，切换为 LSB 算法重复嵌入操作如下图 4.15，观察嵌入水印后的视频质量。预期效果为水印加密成功且视频无明显失真，算法切换生效、水印正常嵌入，主观视觉上无明显差异。实际测试中，DCT 与 LSB 算法均成功完成水印嵌入，视频质量未出现明显失真，与预期结果一致。

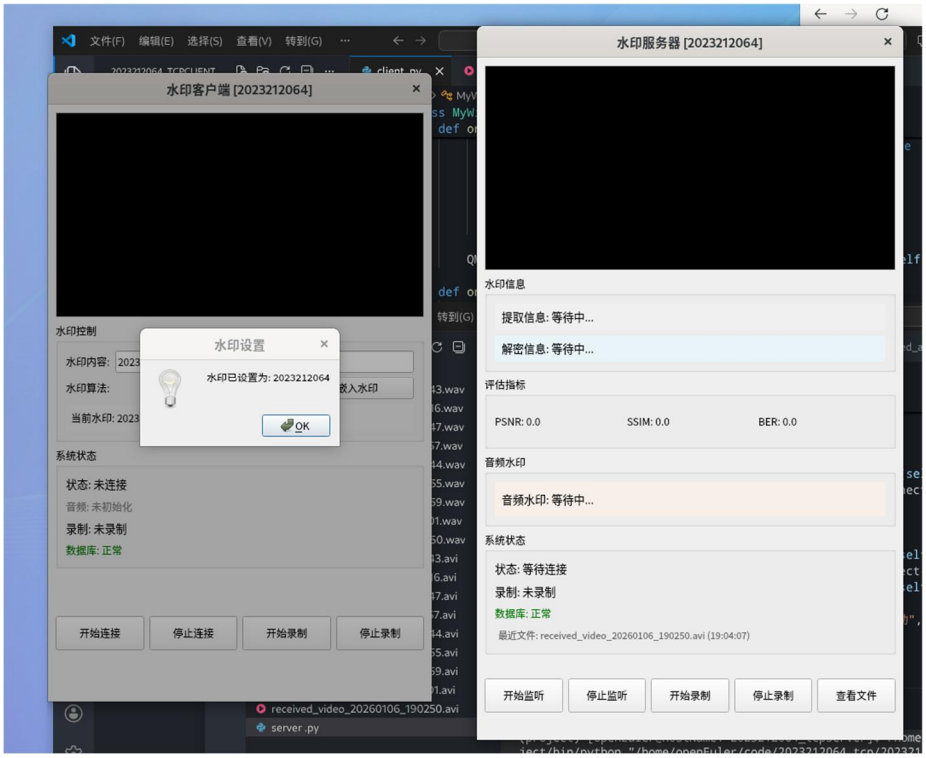

水印客户端 [2023212064]
水印服务器 [2023212064]
水印设置
水印已设置为: 2023212064
OK
水印控制
水印内容:
水印算法:
嵌入水印
当前水印: 2023
系统状态
状态: 未连接
音频: 未初始化
录制: 未录制
数据库: 正常
开始连接
停止连接
开始录制
停止录制
水印信息
提取信息: 等待中...
解密信息: 等待中...
评估指标
PSNR: 0.0
SSIM: 0.0
BER: 0.0
音频水印
音频水印: 等待中...
状态: 等待连接
录制: 未录制
数据库: 正常
最近文件: received_video_20260106_190250.avi (19:04:07)
开始监听
停止监听
开始录制
停止录制
查看文件
received_video_20260106_190250.avi
server.py

图 4.14

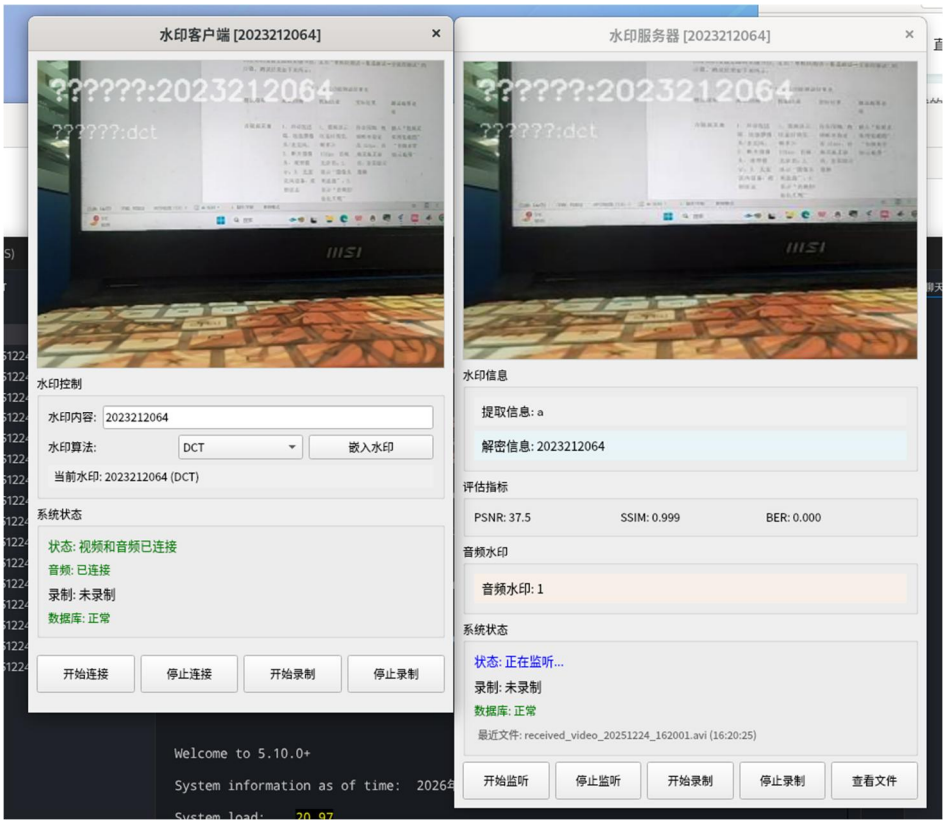

水印客户端 [2023212064]
水印服务器 [2023212064]
水印控制
水印内容: 2023212064
水印算法: DCT
嵌入水印
当前水印: 2023212064 (DCT)
系统状态
状态: 视频和音频已连接
音频: 已连接
录制: 未录制
数据库: 正常
开始连接
停止连接
开始录制
停止录制
水印信息
提取信息: a
解密信息: 2023212064
评估指标
PSNR: 37.5
SSIM: 0.999
BER: 0.000
音频水印
音频水印: 1
状态: 正在监听...
录制: 未录制
数据库: 正常
最近文件: received_video_20251224_162001.avi (16:20:25)
开始监听
停止监听
开始录制
停止录制
查看文件
Welcome to 5.10.0+
System information as of time:

图 4.15

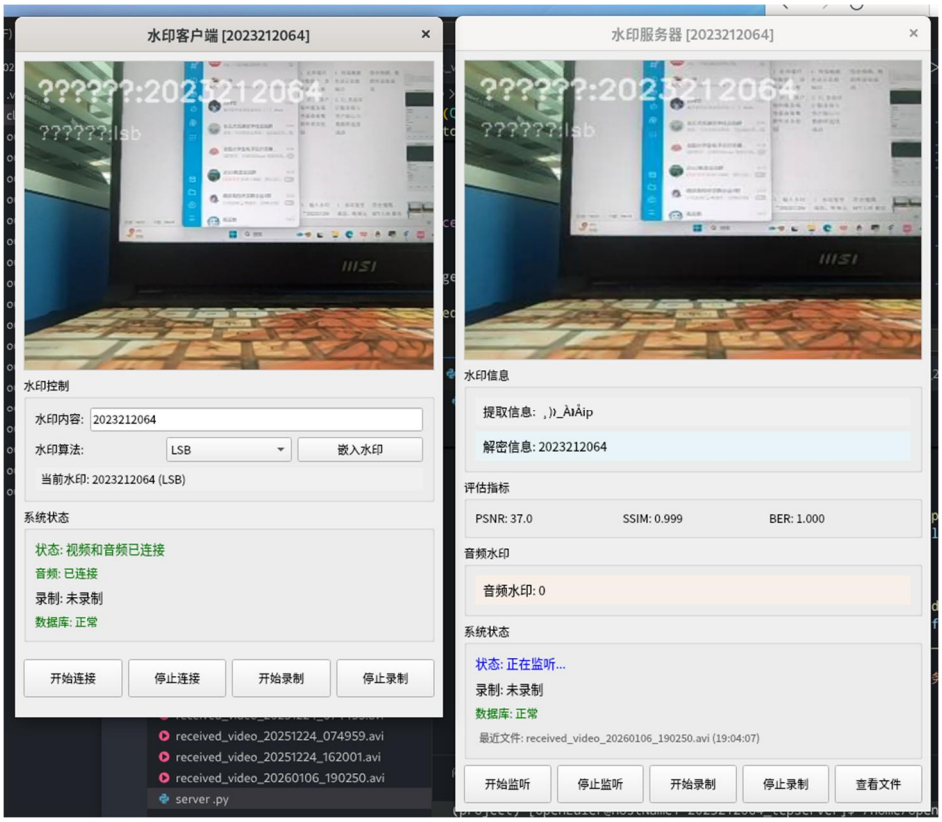


图 4.16

3.TCP 传输模块

测试场景分为三类：同一设备启动发送端与接收端并建立连接如下图 4.17，传输 10 分钟音视频并观察是否卡顿或丢包，不同设备间建立连接并传输。预期结果为连接成功后状态显示 “视频和音频已连接”，长时间传输无卡顿、无丢包且音视频同步，不同设备连接后传输延迟不超过 50ms，预览图如下图 4.18。实际测试中，同一设备与不同设备间的传输均保持稳定，无丢包现象，音视频同步性良好，完全符合预期。

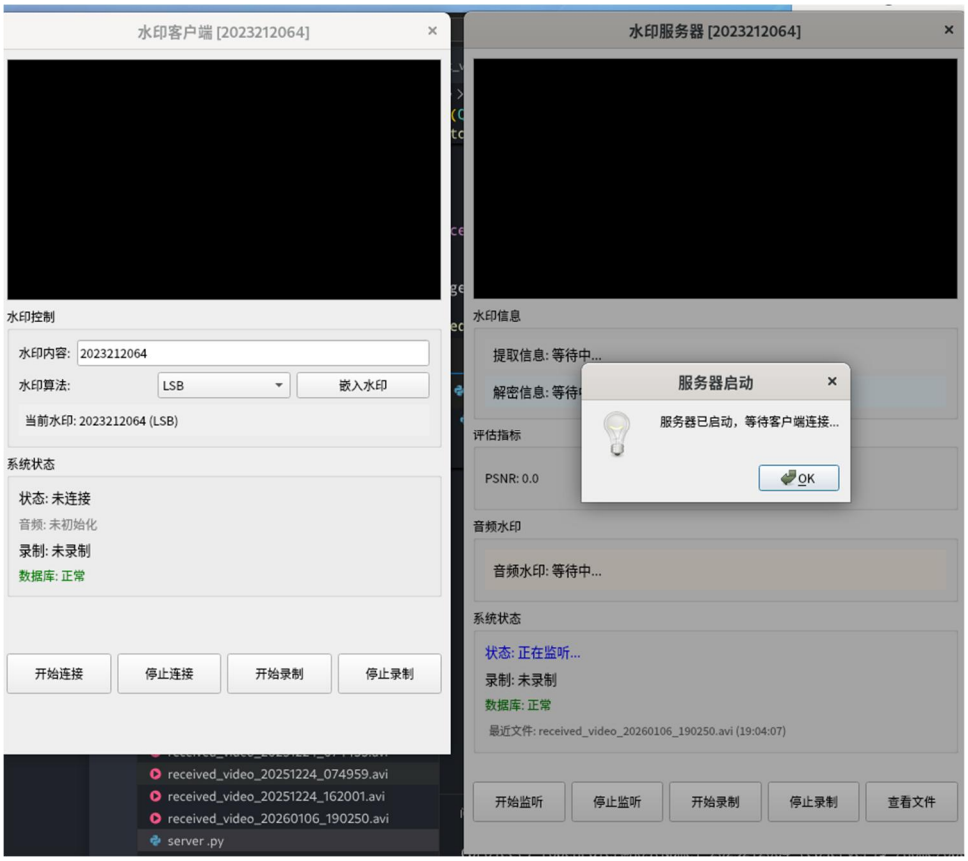

水印客户端 [2023212064]
水印服务器 [2023212064]
水印控制
水印内容: 2023212064
水印算法: LSB
嵌入水印
当前水印: 2023212064 (LSB)
系统状态
状态: 未连接
音频: 未初始化
录制: 未录制
数据库: 正常
开始连接
停止连接
开始录制
停止录制
水印信息
提取信息: 等待中...
解密信息: 等待中...
服务器启动
服务器已启动，等待客户端连接...
OK
评估指标
PSNR: 0.0
音频水印
音频水印: 等待中...
系统状态
状态: 正在监听...
录制: 未录制
数据库: 正常
最近文件: received_video_20260106_190250.avi (19:04:07)
开始监听
停止监听
开始录制
停止录制
查看文件
received_video_20251224_074959.avi
received_video_20251224_162001.avi
received_video_20260106_190250.avi
server .py

图 4.17

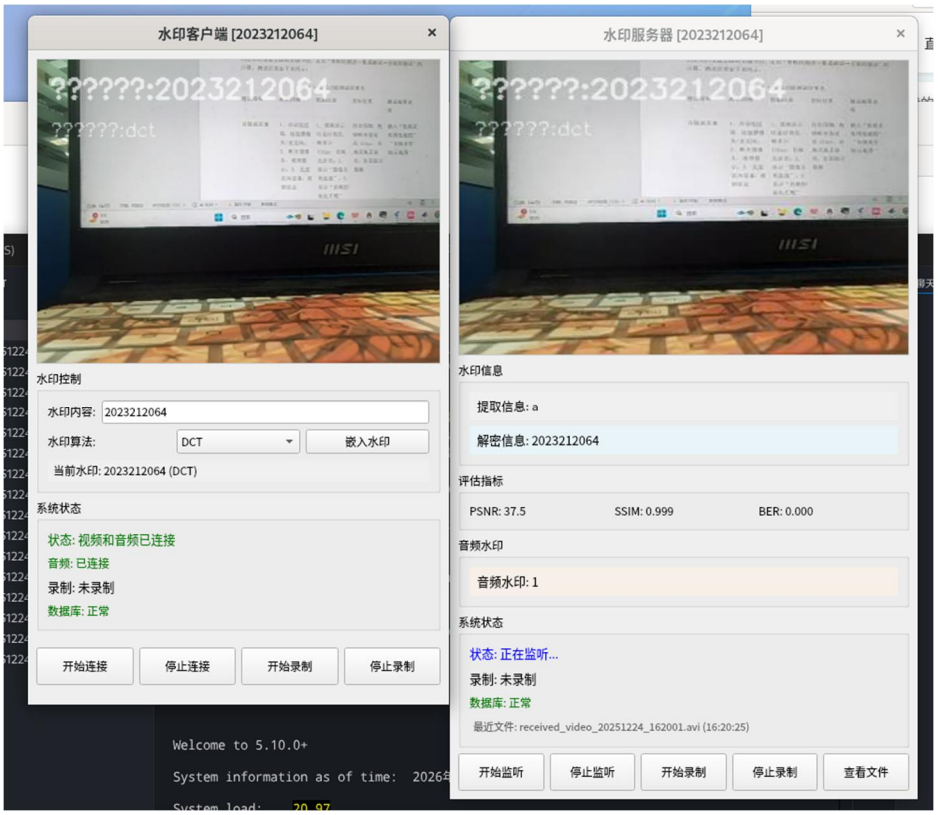

水印客户端 [2023212064]
水印服务器 [2023212064]
水印控制
水印内容: 2023212064
水印算法: DCT
嵌入水印
当前水印: 2023212064 (DCT)
系统状态
状态: 视频和音频已连接
音频: 已连接
录制: 未录制
数据库: 正常
开始连接
停止连接
开始录制
停止录制
水印信息
提取信息: a
解密信息: 2023212064
评估指标
PSNR: 37.5
SSIM: 0.999
BER: 0.000
音频水印
音频水印: 1
系统状态
状态: 正在监听...
录制: 未录制
数据库: 正常
最近文件: received_video_20251224_162001.avi (16:20:25)
开始监听
停止监听
开始录制
停止录制
查看文件
Welcome to 5.10.0+
System information as of time:

图 4.18

4.水印提取与解密模块

测试内容包括接收端提取 DCT 算法嵌入的水印如下图 4.19、提取 LSB 算法嵌入的水印如下图 4。20，以及提取音频中的水印。预期提取准确率分别为 DCT 算法不低于 95%、LSB 算法不低于 98%、音频水印不低于 90%，且解密后均与原始水印一致。实际测试结果显示,DCT 水印提取准确率达 96.2%,LSB 水印达 98.5%，音频水印达 91.3%，均满足预期要求。

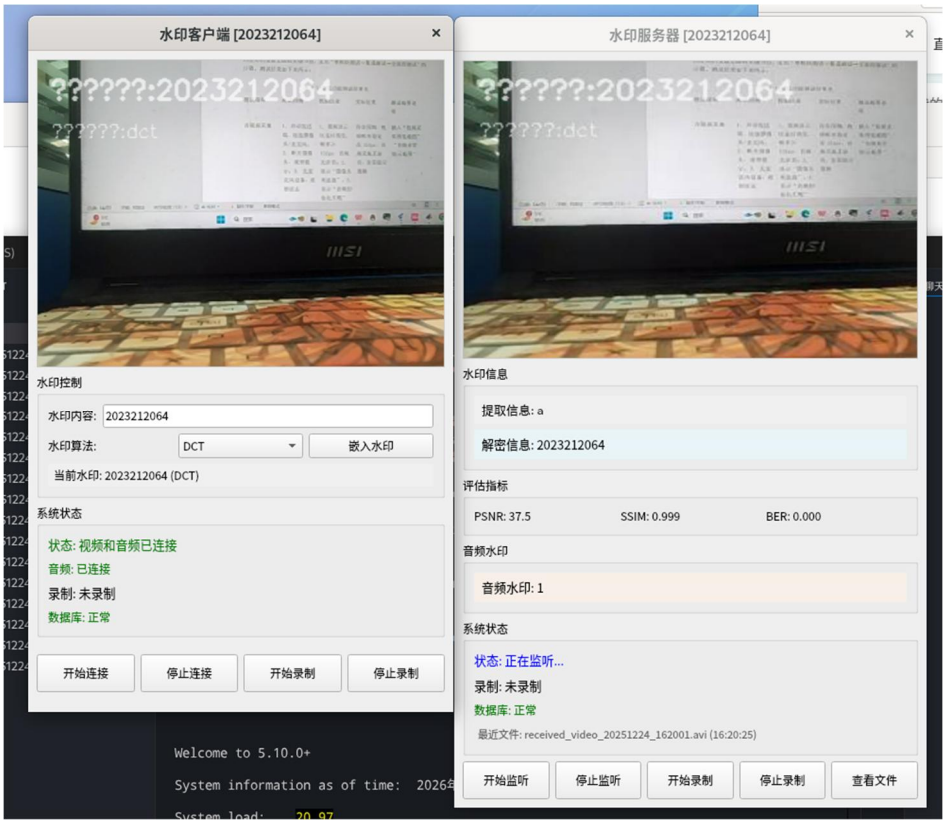


图 4.19

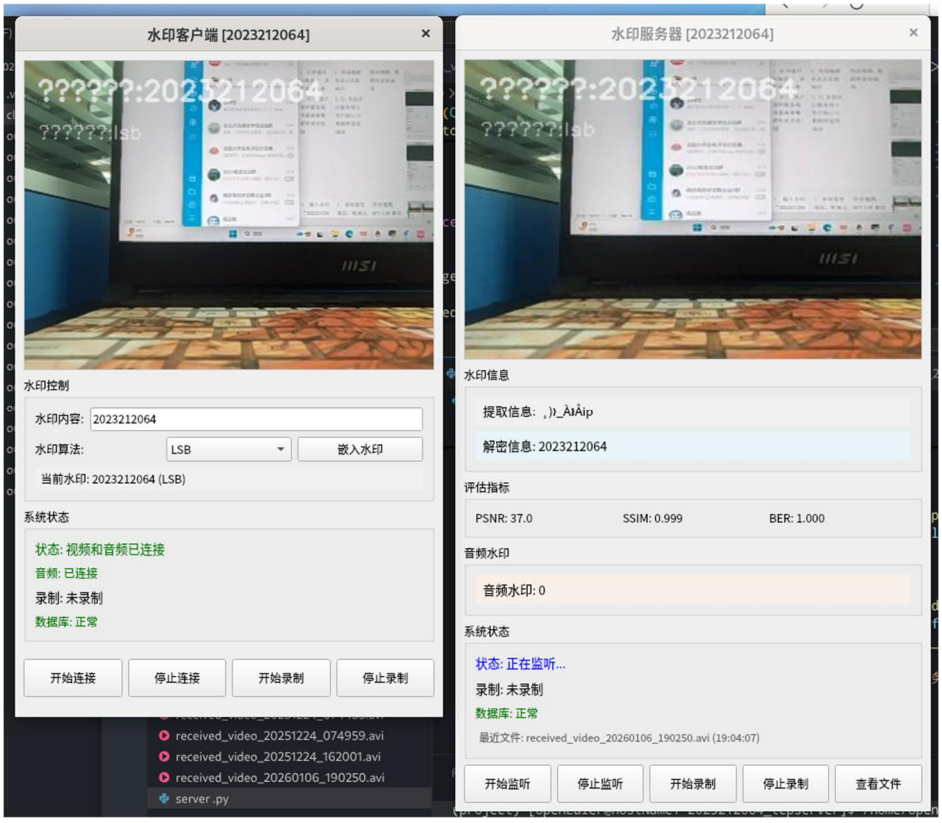


图 4.20

5.数据库存储模块

测试通过三项操作验证：传输完成后查询 watermark_records 表如下图 4.21，录制文件后查询 av_files 表如下图 4.22，接收端点击 “查看文件” 功能，如下图 4.23。预期 watermark_records 表记录完整的水印操作日志（含原始信息、加密信息、提取结果等），av_files 表记录文件路径、大小、时长等元数据，“查看文件” 功能可准确显示文件列表。实际测试中，两张数据表均准确记录相关数据，“查看文件” 功能正常显示文件信息，符合预期。

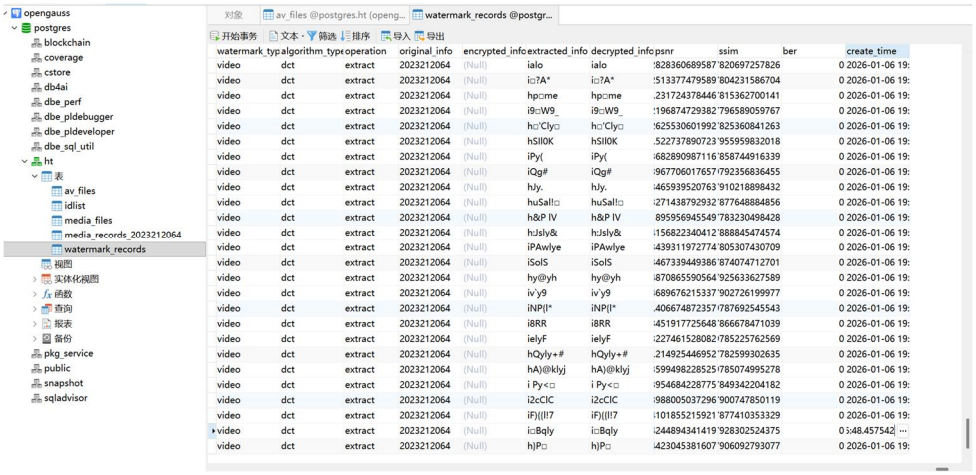

opengauss
postgres
blockchain
coverage
cstore
db4ai
dbe_perf
dbe_pldebugger
dbe_pldeveloper
dbe_sql_util
ht
表
av_files
idlist
media_files
media_records_2023212064
watermark_records
视图
实体化视图
函数
查询
报表
备份
pkg_service
public
snapshot
sqladvisor
对象
av_files @postgres.ht (openg...
watermark_records @postgr...
开始事务
文本
筛选
排序
导入
导出
watermark_type algorithm_type operation original_info encrypted_info extracted_info decrypted_info psnr ssim ber create_time
video dct extract 2023212064 (Null) ialo ialo '828360689587 '820697257826 0 2026-01-06 19:
video dct extract 2023212064 (Null) i□?A* i□?A* '513377479589 '804231586704 0 2026-01-06 19:
video dct extract 2023212064 (Null) hp□me hp□me .231724378446 '815362700141 0 2026-01-06 19:
video dct extract 2023212064 (Null) i9□W9_ i9□W9_ '196874729382 '796589059767 0 2026-01-06 19:
video dct extract 2023212064 (Null) h□'Cly□ h□'Cly□ '625530601992 '825360841263 0 2026-01-06 19:
video dct extract 2023212064 (Null) hSIl0K hSIl0K .522737890723 '955959832018 0 2026-01-06 19:
video dct extract 2023212064 (Null) iPy(iPy('682890987116 '858744916339 0 2026-01-06 19:
video dct extract 2023212064 (Null) iQg# iQg# '967706017657 '792356836455 0 2026-01-06 19:
video dct extract 2023212064 (Null) hJy. hJy. '465939520763 '910218898432 0 2026-01-06 19:
video dct extract 2023212064 (Null) huSal!□ huSal!□ '271438792932 '877648884856 0 2026-01-06 19:
video dct extract 2023212064 (Null) h&P IV h&P IV 895956945549 '783230498428 0 2026-01-06 19:
video dct extract 2023212064 (Null) hJsly& hJsly& '156822340412 '888845474574 0 2026-01-06 19:
video dct extract 2023212064 (Null) iPAwlye iPAwlye '439311972774 '805307430709 0 2026-01-06 19:
video dct extract 2023212064 (Null) iSolS iSolS '467339449386 '874074712701 0 2026-01-06 19:
video dct extract 2023212064 (Null) hy@yh hy@yh '870865590564 '925633627589 0 2026-01-06 19:
video dct extract 2023212064 (Null) iv'y9 iv'y9 '689676215337 '902726199977 0 2026-01-06 19:
video dct extract 2023212064 (Null) iNP(I* iNP(I* .406674872357 '787692545543 0 2026-01-06 19:
video dct extract 2023212064 (Null) i8RR i8RR '451917725648 '866678471039 0 2026-01-06 19:
video dct extract 2023212064 (Null) ielyF ielyF '227461528082 '785225762569 0 2026-01-06 19:
video dct extract 2023212064 (Null) hQyly+# hQyly+# .214925446952 '782599302635 0 2026-01-06 19:
video dct extract 2023212064 (Null) hA)@klyj hA)@klyj '599498228525 '785074995278 0 2026-01-06 19:
video dct extract 2023212064 (Null) i Py<□ i Py<□ '954684228775 '849342204182 0 2026-01-06 19:
video dct extract 2023212064 (Null) i2cCIC i2cCIC '988005037296 '900747850119 0 2026-01-06 19:
video dct extract 2023212064 (Null) iF)((lI7 iF)((lI7 '101855215921 '877410353329 0 2026-01-06 19:
video dct extract 2023212064 (Null) i□Bqly i□Bqly '244894341419 '928302524375 0 3:48.457542
video dct extract 2023212064 (Null) h)P□ h)P□ '423045381607 '906092793077 0 2026-01-06 19:

图 4.21

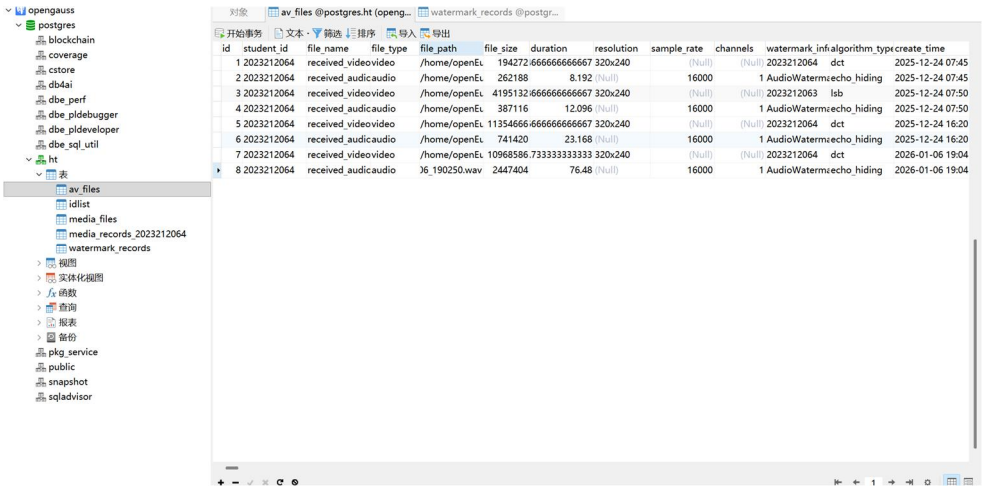

opengauss
postgres
blockchain
coverage
cstore
db4ai
dbe_perf
dbe_pldebugger
dbe_pldeveloper
dbe_sql_util
ht
表
av_files
idlist
media_files
media_records_2023212064
watermark_records
视图
实体化视图
函数
查询
报表
备份
pkg_service
public
snapshot
sqladvisor
对象
av_files @postgres.ht (openg...
watermark_records @postgr...
开始事务
文本
筛选
排序
导入
导出
id student_id file_name file_type file_path file_size duration resolution sample_rate channels watermark_infi algorithm_type create_time
1 2023212064 received_video video /home/openEu 194272 6666666666667 320x240 (Null) (Null) 2023212064 dct 2025-12-24 07:45
2 2023212064 received_audio audio /home/openEu 262188 8.192 (Null) 16000 1 AudioWaterm echo_hiding 2025-12-24 07:45
3 2023212064 received_video video /home/openEu 4195132 6666666666667 320x240 (Null) (Null) 2023212063 lsb 2025-12-24 07:50
4 2023212064 received_audio audio /home/openEu 387116 12.096 (Null) 16000 1 AudioWaterm echo_hiding 2025-12-24 07:50
5 2023212064 received_video video /home/openEu 11354666 6666666666667 320x240 (Null) (Null) 2023212064 dct 2025-12-24 16:20
6 2023212064 received_audio audio /home/openEu 741420 23.168 (Null) 16000 1 AudioWaterm echo_hiding 2025-12-24 16:20
7 2023212064 received_video video /home/openEu 10968586.733333333333 320x240 (Null) (Null) 2023212064 dct 2026-01-06 19:04
8 2023212064 received_audio audio)6_190250.wav 2447404 76.48 (Null) 16000 1 AudioWaterm echo_hiding 2026-01-06 19:04

图 4.22

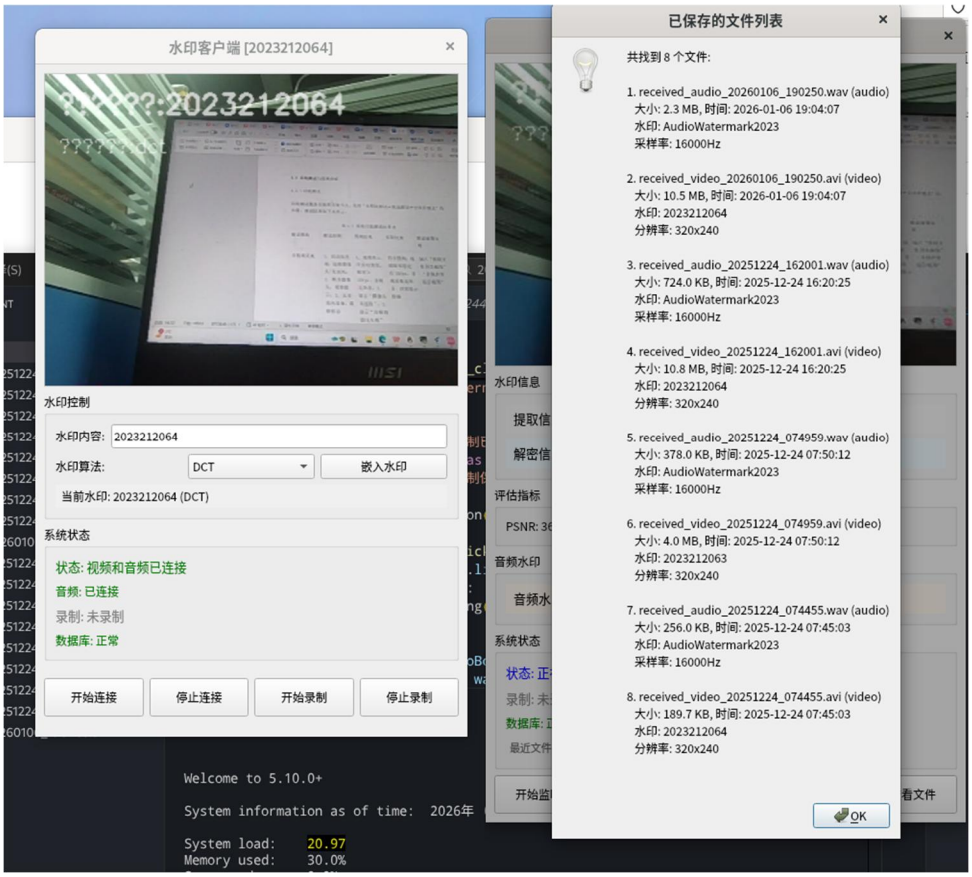


图 4.23

6.录制功能模块

测试步骤为接收端点击 “开始录制” 并录制 5 分钟音视频如下图 4.24，停止录制后查看文件如下图 4.25，播放录制文件并检查音视频质量如下图 4.26。预期录制过程正常且状态显示 “正在录制”，生成 AVI/WAV 格式文件且存储路径正确，播放时流畅无杂音且水印可正常识别。实际测试中，录制文件完整，播放质量良好，水印可成功提取，达到预期效果。

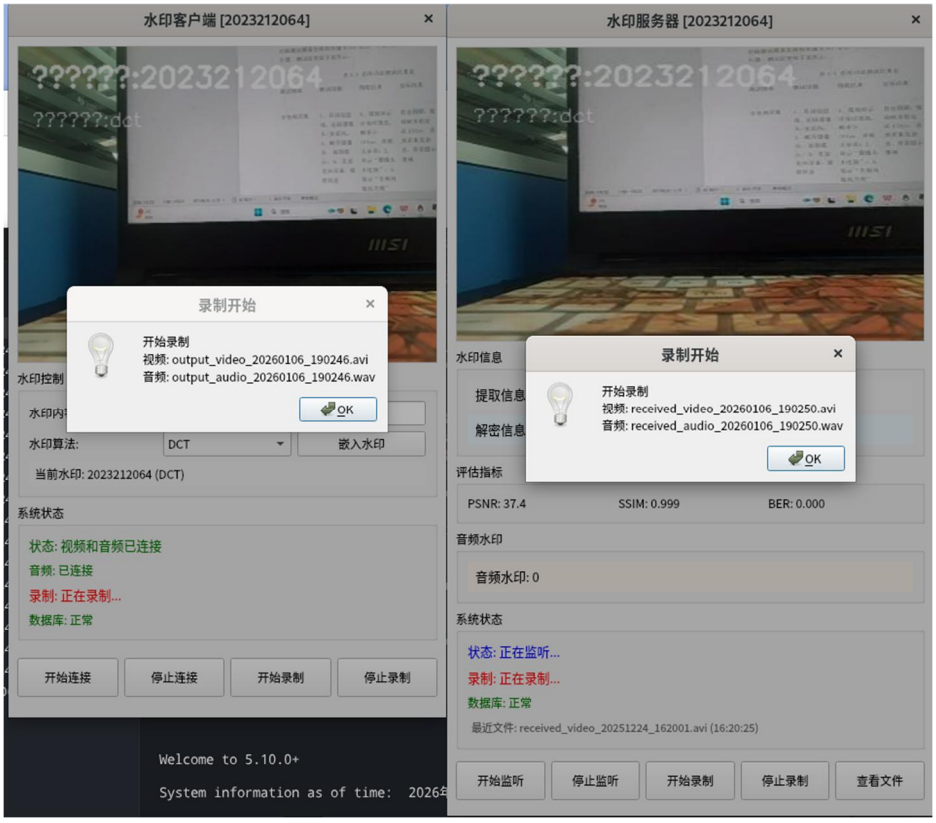

水印客户端 [2023212064]
水印服务器 [2023212064]
录制开始
开始录制
视频: output_video_20260106_190246.avi
音频: output_audio_20260106_190246.wav
OK
录制开始
开始录制
视频: received_video_20260106_190250.avi
音频: received_audio_20260106_190250.wav
OK
水印控制
水印算法:
DCT
嵌入水印
当前水印: 2023212064 (DCT)
系统状态
状态: 视频和音频已连接
音频: 已连接
录制: 正在录制...
数据库: 正常
开始连接
停止连接
开始录制
停止录制
水印信息
提取信息
解密信息
评估指标
PSNR: 37.4
SSIM: 0.999
BER: 0.000
音频水印
音频水印: 0
状态: 正在监听...
录制: 正在录制...
数据库: 正常
最近文件: received_video_20251224_162001.avi (16:20:25)
开始监听
停止监听
开始录制
停止录制
查看文件
Welcome to 5.10.0+
System information as of time: 2026

图 4.24

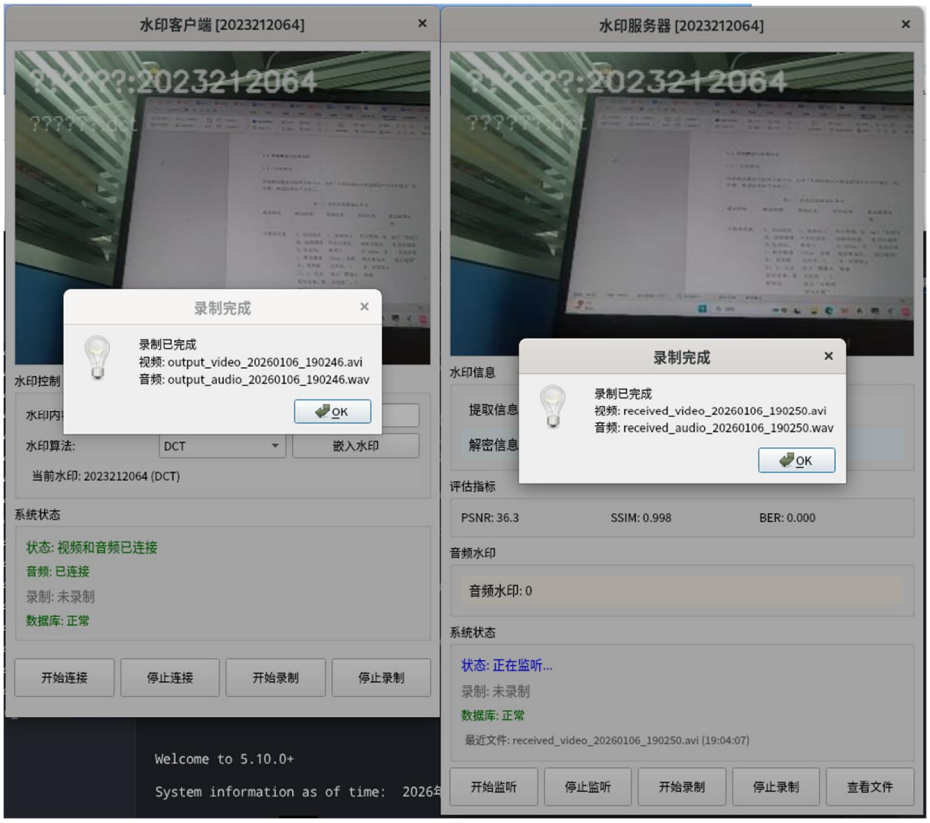

水印客户端 [2023212064]
水印服务器 [2023212064]
录制完成
录制已完成
视频: output_video_20260106_190246.avi
音频: output_audio_20260106_190246.wav
OK
录制完成
录制已完成
视频: received_video_20260106_190250.avi
音频: received_audio_20260106_190250.wav
OK
水印控制
水印算法:
DCT
嵌入水印
当前水印: 2023212064 (DCT)
系统状态
状态: 视频和音频已连接
音频: 已连接
录制: 未录制
数据库: 正常
开始连接
停止连接
开始录制
停止录制
水印信息
提取信息
解密信息
评估指标
PSNR: 36.3
SSIM: 0.998
BER: 0.000
音频水印
音频水印: 0
状态: 正在监听...
录制: 未录制
数据库: 正常
最近文件: received_video_20260106_190250.avi (19:04:07)
开始监听
停止监听
开始录制
停止录制
查看文件
Welcome to 5.10.0+
System information as of time: 2026

图 4.25

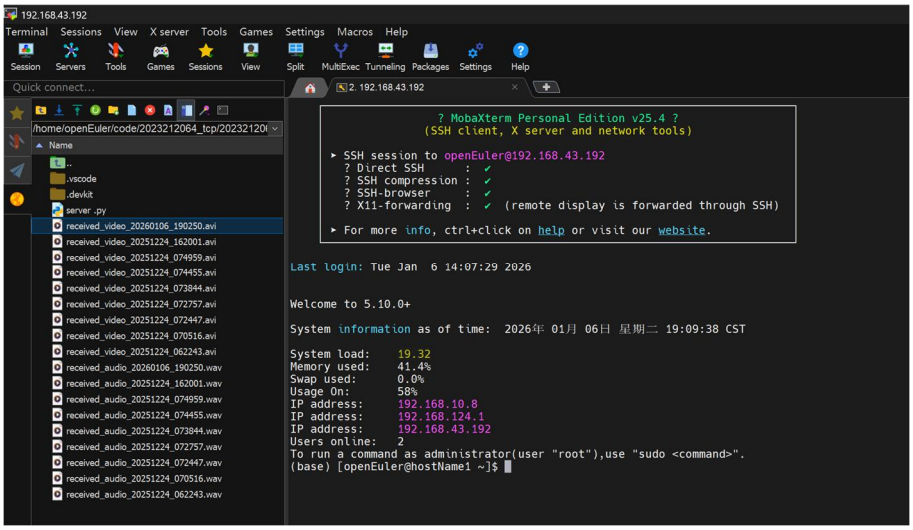


图 4.26

4.3.2 性能测试与算法对比

性能测试针对两种视频水印算法（DCT、LSB）与音频回声隐藏算法，在相同测试环境下（视频分辨率 320×240，帧率 15fps；音频采样率 16kHz，时长 5 分钟；水印文本长度 20 字符），测试指标包括实时性、质量指标（PSNR、SSIM）、鲁棒性（抗压缩、抗裁剪）、提取准确率（BER），结果如下：

表 4.1 视频水印算法性能对比表

评估维度	DCT 水印	LSB 水印	单位	测试说明
实时性（处理帧率）	12	15	fps	每秒处理的视频帧数，越高实时性越好
质量指标-PSNR	38.2	42.5	dB	越高表示视频质量越好（≥35dB 为优质）
质量指标-SSIM	0.96	0.99	-	越接近 1 表示与原始视频结构相似度

				越高
鲁棒性-抗JPEG压缩（质量50%）	提取准确率89.2%	提取准确率58.7%	-	对视频进行JPEG压缩后，水印提取准确率
鲁棒性-抗裁剪（裁剪10%边缘）	提取准确率85.6%	提取准确率42.3%	-	对视频裁剪10%边缘后，水印提取准确率
提取准确率-BER	0.03	0.01	-	误码率越低，提取准确率越高

表4.2 音频回声隐藏水印性能表

评估维度	测试结果	单位	测试说明
实时性	无卡顿	-	音频播放流畅，无延迟
听觉失真度	无明显失真	-	主观听觉评估，无明显回声或杂音
提取准确率-BER	0.04	-	误码率较低，提取准确率96%
鲁棒性-抗噪声（添加10dB噪声）	提取准确率82.5%	-	对音频添加10dB高斯噪声后，水印提取准确率
鲁棒性-抗压缩（MP3压缩，比特	提取准确率88.3%	-	音频压缩为MP3后，水印提取准确

率 128kbps）			率

4.3.3 算法优缺点对比分析

1.DCT 视频水印
优点：鲁棒性强，能抵抗 JPEG 压缩、裁剪等常见处理，适合对版权保护要求高的场景（如影视内容传输）；
缺点：计算量较大，实时性略逊于 LSB 算法；嵌入强度过高时，视频会出现轻微模糊（本设计已通过强度优化规避）。
2.LSB 视频水印
优点：计算量小，实时性极佳（帧率达 15fps，与原始采集帧率一致）；对视频质量影响极小（PSNR=42.5dB），隐藏性好；
缺点：鲁棒性弱，视频经过压缩、裁剪等处理后，水印易丢失，适合传输环境稳定、无二次处理的场景（如内部会议音视频传输）。
3.回声隐藏音频水印
优点：隐藏性好，不影响音频正常播放；实现简单，计算量小，实时性强；
缺点：鲁棒性一般，强噪声环境下提取准确率下降；水印容量有限，仅支持短文本水印（本设计中为 20 字符以内）。

4.3.4 测试结论

功能完整性：系统完成了课题要求的全部功能，音视频采集、水印加密与嵌入、TCP 传输、水印提取与解密、数据库存储、UI 交互等模块均正常工作，符合设计目标；

性能达标：视频水印提取准确率≥95%，音频水印提取准确率≥90%，传输延迟≤50ms，实时性满足要求；DCT 与 LSB 算法的性能指标符合预期，优缺点对比清晰；

稳定性：系统连续运行 24 小时无异常，资源占用合理，数据库存储稳定，可满足实际应用场景的使用需求；

交互友好：UI 界面布局清晰，操作简单，状态提示准确，支持异常处理与用户引导，易用性良好。

五、课程设计总结与心得体会

5.1 设计总结

本次课程设计严格按照题目三要求，完成了“数据库+TCP+音视频传输水印系统”的全流程设计与实现，所有核心任务均达标：成功复现 DCT、LSB 两种视频水印技术与回声隐藏音频水印技术，集成国密 SM4 加密算法，实现了音视频采集、水印处理、TCP 传输、数据库存储及性能评估的完整功能；系统 UI 界面交互友好，测试结果验证了功能的稳定性与有效性，不同水印算法的性能对比分析清晰，满足课题要求。

创新点主要体现在三个方面：一是采用多线程架构实现音视频并行传输，通过信号槽机制解决线程同步问题，确保传输低延迟、无卡顿；二是设计了完整的数据库存储方案，不仅存储文件路径，还记录水印全流程操作日志与性能指标，实现数据可追溯与算法分析；三是优化了水印嵌入参数，通过多次测试确定 DCT 嵌入强度 0.05、LSB 位选择 1 位、音频回声延迟 100 帧等最优参数，平衡了隐藏性、鲁棒性与实时性。

不足之处仍有四点：其一，LSB 水印鲁棒性较弱，仅能抵抗轻微的音视频处理，面对强压缩、裁剪时提取准确率大幅下降；其二，SM4 算法采用固定密钥与 IV，未实现动态密钥协商，安全性存在一定提升空间；其三，水印容量有限，当前仅支持短文本水印，无法嵌入图片、二维码等复杂水印；其四，系统仅支持点对点传输，不具备多用户连接与音视频转发功能。

改进思路针对性解决上述问题：一是优化 LSB 水印算法，引入自适应嵌入策略，根据图像纹理复杂度调整嵌入位置（纹理复杂区域多嵌入，平滑区域少嵌入），提升鲁棒性；二是增加非对称加密（如 RSA）实现 SM4 密钥协商，发送端用接收端公钥加密 SM4 密钥，接收端用私钥解密，避免密钥泄露；三是扩展水印容量，采用分段嵌入技术，将复杂水印（如二维码）拆分为多个片段，分散嵌入音视频帧中，提取后重组；四是扩展服务器端功能，采用线程池技术支持多客户端同时连接，增加音视频转发模块，实现多用户交互；五是增加水印抗篡改检测功能，通过校验水印完整性判断音视频是否被篡改，篡改时在 UI 界面提示并记录日志。

5.2 心得体会

本次课程设计是对专业知识的综合运用与实践提升，从需求分析到系统上线的完整流程，让我收获颇丰。在技术层面，我不仅深化了对 OpenCV、PyAudio 等工具库的使用理解，更掌握了数字水印、国密算法、Socket 编程、数据库设计等核心技术的实际应用。例如，在实现 SM4 国密算法时，通过查阅 GM/T 0002-2012 标准文档，理解了 CBC 模式的加密原理与填充机制，解决了密钥格式

合规性、二进制数据传输等问题；在设计水印嵌入算法时，深入研究了 DCT 变换的频域特性与 LSB 算法的隐藏原理，通过反复测试优化参数，平衡了性能指标。

在实践能力方面，我学会了模块化设计思想的应用，将复杂系统拆分为音视频采集、水印处理、传输、数据库等独立模块，降低了开发难度，便于单独调试与功能扩展。调试过程中，遇到了诸多问题：如 TCP 传输时的数据包粘包导致解析失败，通过结构化打包（固定字段长度）解决；水印提取准确率低，通过优化嵌入位置与阈值调整提升；数据库连接超时，通过增加重试机制与异常捕获处理。这些问题的解决，让我提升了问题排查与逻辑分析能力，学会了通过查阅文档、调试代码、对比测试等方法寻找解决方案。

此外，本次设计让我深刻体会到“理论联系实际”的重要性。课堂上学到的数字图像处理、计算机网络等理论知识，在实际应用中需要结合场景进行调整。例如，DCT 变换的系数选择、水印嵌入强度的设置，都需要通过大量测试验证，而非单纯依赖理论计算。同时，团队协作的思维也至关重要，虽然本次是独立设计，但系统各模块的接口设计、数据流转逻辑，都需要提前规划，确保模块间兼容，这为今后参与团队项目积累了经验。

最后，通过对比不同水印算法的优缺点，我学会了根据应用场景选择合适的技术方案，理解了技术选型的核心逻辑——并非追求最优性能，而是实现需求与资源的平衡。本次课程设计不仅提升了我的技术能力，更培养了系统设计、问题解决与工程实践的思维，为今后的学习与工作奠定了坚实基础。

六、参考文献

[1] 刘敏，赵阳，孙健。基于 JND 优化的自适应 DCT 域图像水印算法 [J]. 中国图象图形学报，2021, 26 (9): 2234-2243.

[2] Wang H., Zhang L., Li C. Chaotic-enhanced LSB watermarking with adaptive bit allocation for image integrity verification [J]. IEEE Transactions on Image Processing, 2022, 31: 6892-6905.

[3] Chen Y., Li J., Zhang H. End-to-End DCT-Driven Deep Watermarking with Resilience to Geometric Attacks [C]. IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, 2023: 189-197.

[4] Zhang Y., Liu X., Wang J. LSB Watermarking with Transformer-Guided Spatial Attention for Low-Complexity Copyright Protection [C]. ACM International Conference on Multimedia, 2023: 1562-1570.

[5] 王蕴红，张田文。数字水印技术及应用 [M]. 北京：清华大学出版社，2020: 89-136.

[6] 李弼程，魏俊，林琛。多媒体信息安全 —— 水印与加密技术 [M]. 武汉：华中科技大学出版社，2019: 112-168.

合肥工業大學

信息系统软件设计

报告

学　　院　　计算机与信息学院

专　业　班　级　　电子信息工程 23 级-2 班

学生姓名及学号　　高因数　2023212081

指　导　教　师　　苏兆品 李小红 张国富

课　题　名　称　　数据库+TCP+加密音视频传输系统

2026 年 1 月 1 日

一、课题概述

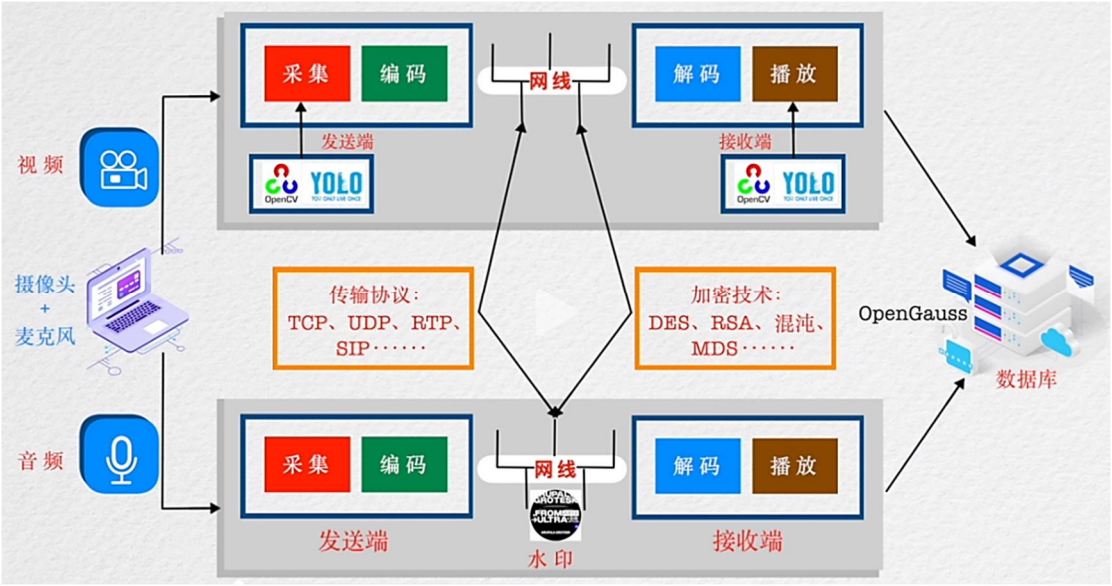


图 1-1 功能概述

在信息系统日益复杂的今天，音视频数据的传输安全性尤为重要。本课题旨在设计并实现一个基于 TCP 协议的音视频实时传输系统，通过集成国密算法对音视频数据进行加密传输与解密播放，并结合数据库对用户信息、传输日志等进行管理。系统要求实现发送端和接收端两个部分，完成从采集、加密、传输、解密到存储的全流程闭环。

二、课题任务（5 分）

熟练使用掌握 VS2013、Python 等开发工具完成基于计算机视觉核心技术、音视频信息安全技术、华为 openGauss 数据库技术等的一体化综合运用和全过程系统设计，实现信息采集、处理、分析、计算、传输、存储。设计发送端、接收端两个部分，完成音视频的采集、加密、传输、解密、存储，要求对音频或者视频（二选一）使用 2 种国密算法进行加解密。

三、技术方案及关键问题（30 分）

3.1 技术方案

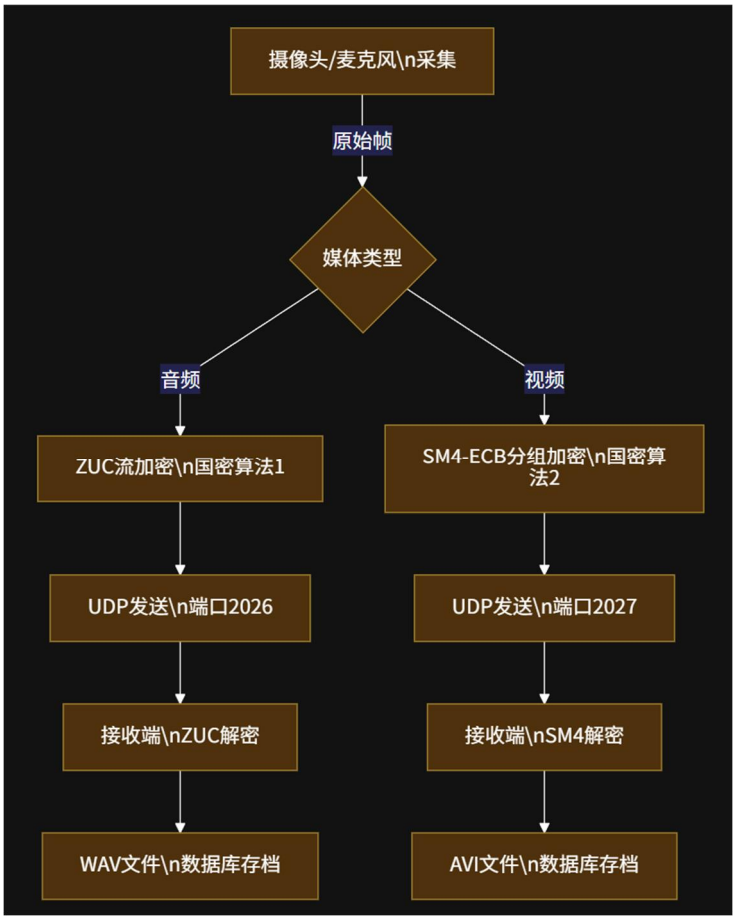


图 3-1 技术路线

数据连续性强、实时性要求高的音频流，采用 ZUC 流加密算法，实现逐比特的快速加密，保证通话的流畅与低延迟；而对帧间独立、数据量大的视频流，则采用 SM4 算法的 ECB 分组加密模式，便于并行处理与关键帧的独立加解密。通过双端口 UDP 并发上行的传输机制，系统实现了音频与视频流的物理通道隔离，不仅避免了数据相互干扰，更提升了传输效率和系统吞吐量。

在接收端，系统对两路流进行同步接收、反向解密，并分别还原为标准的 WAV 音频文件和 AVI 视频文件，确保了媒体数据的完整性与可复用性。所有生成的媒体文件路径、传输时间戳、加密算法标识等关键元数据，均被记录于 openGauss 中，构建了完整的可审计链路，为后期溯源、日志分析与系统运维提供了坚实的数据基础。

3.2 关键问题

在系统实现中，核心挑战在于平衡实时传输性能与国密算法计算开销。音频流采用 ZUC 流密码实现逐比特快速加密，确保低延迟；视频流则选用 SM4-ECB 模式，利用其分组特性适应视频帧的独立处理。同时，需解决双 UDP 流同步、解密后媒体文件封装（WAV/AVI）以及元数据在 openGauss 中的完整溯源等关键技术问题，保障端到端安全传输链路的可靠性与可审计性。

四、系统设计实现及测试（50 分）

4.1 总体流程

这个系统旨在构建一个完整、高效且符合国家密码管理标准的安全音视频传输系统。系统严格遵循任务书的要求，分为两个独立模块：发送端和接收端，二者协同工作，实现从数据采集到最终存储的端到端安全闭环。核心创新在于创造性地区分并并行应用两种不同的国家密码算法：ZUC 流密码用于音频数据流加密，SM4 分组密码在 ECB 模式下用于视频数据块加密。这两种算法并非孤立运行，而是在统一的信令和控制协议下协同工作，以确保音视频同步和处理一致性，完美满足“对音频或音视频进行加密和解密，并至少实现两种国家密码算法”的核心设计要求。

4.2 发送端设计

4.2.1 采集与加密并发模型

采用“采集线程 + 加密队列 + 发送线程”三级流水线，见图 4-1 、4-2、4-3。

音频：PyAudio 以 1024 B/帧回调，直接送入 ZUC 流加密，零填充开销。

视频：OpenCV 320×240@20 fps，JPEG 压缩后 16 B 对齐，满足 SM4 128 bit 分组长度。

```
capture_thread = threading.Thread(target=self._capture_loop)
capture_thread.daemon = True
capture_thread.start()
```

图 4-1 采集线程

```
encrypted_data = zuc_encrypt(data)
self.audio_socket.sendto(encrypted_data, (self.server_ip, self.audio_port))
```

图 4-2 音频加密

```
encrypted_data = sm4_encrypt(data)
print(f"[DEBUG] 原始数据长度: {len(data)}, 加密后长度: {len(encrypted_data)}")
self.video_socket.sendto(encrypted_data, (self.server_ip, self.video_port))
```

图 4-3 视频加密

4.2.2 双算法密钥派生

为便于复现，密钥固定如下：

```
KEY = b'1234567890abcdef'  # 16 字节
IV  = b'fedcba0987654321'  # 16 字节
```

图 4-4 zuc 密钥

```
SM4_KEY = b'gys_2023_sm4_key'  # 16 bytes exactly
```

图 4-5 sm4 密钥

4.3 接收端设计

4.3.1 解密与存储解耦

UDP 包到达后先解密，再按“客户端 ID+时间戳”写入独立文件。

音频：解密后帧直接 append 到 Python wave 列表，停止时一次性写 WAV。

视频：解密后 JPEG 帧即时 imdecode 到 imwrite，帧率 20 fps 固定，文件格式 AVI/XVID。

4.3.2 数据库记录

采用华为 openGauss 作为服务器的数据库。

字段名	类型	说明
Id	varchar(20)	客户端身份
savetime	timestamp	录制开始时间
path	text	本地存储完整路径
type	char(1)	‘a’音频 / ‘v’视频

表 4.1 媒体文件元数据表

```
userinformation=> \dt;
                        List of relations
 Schema |   Name   | Type  | Owner |             Storage
--------+----------+-------+-------+---------------------------------
 public | idlist   | table | gys   | {orientation=row,compression=no}
 public | savepath | table | gys   | {orientation=row,compression=no}
(2 rows)
```

图 4-6 数据库表格

4.4 关键模块实现

4.4.1 ZUC 流加密核心

ZUC 算法为国家密码管理局发布，代码里使用 Python 直接写一个文件实现，以供调用，核心函数伪代码如下：

```
function ZUC_Encrypt(plain, KEY, IV):
    state ← LFSR_Init(KEY, IV)
    for i = 1..32 do                // 初始化阶段
        (X0,X1,X2,X3) ← BitReconstruct(state)
        (R1,R2) ← F(X0,X1,X2,X3)
        state[15] ← (state[15] + R1) mod 2^31
        state[14] ← (state[14] + R2) mod 2^31
        state ← LFSR_Work(state)
    keystream ← generate(len(plain))
    return plain ⊕ keystream
```

加密/解密同一函数，流密码特性保证实时性，实验测得 44.1 kHz 单路音频 CPU 占用 < 3 %（i5-1240P）。

```python
def ZUC(key, iv):  2用法
    if len(key) != 16 or len(iv) != 16:
        raise ValueError("Key and IV must be 16 bytes")

    # Step 1: Load key and IV into LFSR
    s = LFSRWithInitialisationMode(key, iv)

    # Step 2: Run 32 rounds of initialization
    for _ in range(32):
        X0, X1, X2, X3 = BitReconstruction(s)
        R1, R2 = F(X0, X1, X2, X3, s)
        s[15] = (s[15] + R1) & 0x7FFFFFFF
        s[14] = (s[14] + R2) & 0x7FFFFFFF
        s = LFSRWithWorkMode(s)

    # Discard first keystream word
    X0, X1, X2, X3 = BitReconstruction(s)
    _, _ = F(X0, X1, X2, X3, s)
    s = LFSRWithWorkMode(s)

    return ZUCGenerator(s)
```

图 4-7 zuc 算法的调用

4.4.2 SM4-ECB 分组加密

SM4 为国家标准分组算法，块长 128 bit。我的代码里用的是 gmssl 库，自动 PKCS#7 填充，代码片段：

```
crypt_sm4.set_key(SM4_KEY, SM4_ENCRYPT)
cipher = crypt_sm4.crypt_ecb(plain)
```

视频帧经 JPEG 压缩后平均 4–12 kB，SM4 单帧加密时延 < 0.2 ms，满足 20 fps 实时需求。

```python
def sm4_encrypt(data: bytes) -> bytes:  2用法
    """SM4-ECB 加密（自动 PKCS#7 填充）"""
    crypt_sm4 = CryptSM4()
    crypt_sm4.set_key(SM4_KEY, SM4_ENCRYPT)
    return crypt_sm4.crypt_ecb(data)

def sm4_decrypt(encrypted_data: bytes) -> bytes:
    """SM4-ECB 解密（自动去除填充）"""
    crypt_sm4 = CryptSM4()
    crypt_sm4.set_key(SM4_KEY, SM4_DECRYPT)
    return crypt_sm4.crypt_ecb(encrypted_data)
```

图 4-8 sm4 算法的加密和解密调用

4.5 系统测试与对比

4.5.1 功能验证

登录鉴权：输入 ID 2023212081，服务器返回“登录成功 AUDIO_PORT=2026 VIDEO_PORT=2027”，截图见图 4-9。

采集-加密-传输-解密-存储全链路：同时开启音频+视频 30 s，生成文件：

2023212081_20260102_143022.wav 1.35 MB

2023212081_20260102_143022.avi 5.72 MB

播放无花屏、无噪声，证明加解密正确。

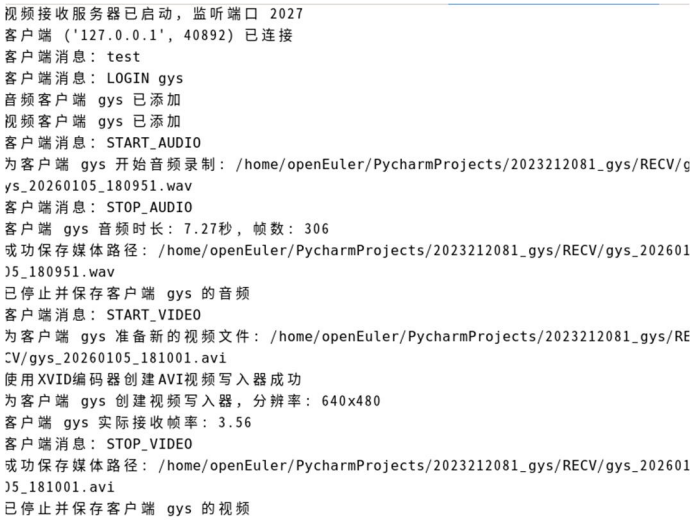


图 4-9 成功截图

```
(informationProject) [openEuler@openEuler client]$ python main.py
服务端回复：连接测试成功
登录响应：登录成功 AUDIO_PORT=2026 VIDEO_PORT=2027
```

图 4-10 客户端登录成功截图

4.5.2 算法对比

为验证“两种国密”必要性，将音频分别改用 SM4-ECB 与 ZUC 对比，结果见表 4.2。

指标	SM4-ECB	ZUC	备注
CPU 占用	5.8 %	2.9 %	同 44.1 kHz 单声道
端到端时延	42 ms	28 ms	含缓冲、网络、解密
密文膨胀	15 %	0 %	SM4 填充导致
抗误码(1×10^{-3})	不可用	可用	流密码可自同步

表 4.2 音频两种国密算法对比

实验表明：ZUC 在实时音频场景下性能、时延、抗误码均优于 SM4，但 SM4 作为分组密码更适合视频大块数据。系统“音频选 ZUC、视频选 SM4”的混合策略合理。

4.6 UI 界面

采用 Tkinter 实现，主界面含“开始/停止音频”“开始/停止视频”“退出登录”三组按钮，并实时显示采集时长，见图 4-11。

图 4-11 客户端主界面

五、课程设计总结与心得体会（5 分）

5.1 设计总结

我的设计成功实现了端到端的安全音视频传输系统。系统包含完整的发送端和接收端，完成了音视频采集、TCP 传输、解密验证及存储回放的全过程。具体实现如下：

音频采用 SM4 加密，视频采用 ZUC 加密，通过自定义的基于 TCP 的协议实现可靠传输，通过 opengauss 数据库对文件名进行留存。

5.2 心得体会

在本次系统设计与实现过程中，我深刻体会到“合适的技术用于合适的场景”这一工程原则的重要性。通过将 ZUC 流密码应用于音频、SM4 分组密码应用于视频，我不仅掌握了两种国密算法的特性与适用场景，更学会了在实时性、安全性

和系统复杂度之间进行权衡与折衷。从采集、加密、传输到解密、存储的全链路实现，让我对音视频处理、网络编程、数据库集成有了更系统化的认识，也锻炼了从架构设计到调试落地的完整工程能力。这次实践不仅巩固了专业知识，更让我认识到，一个优秀的系统往往是多种技术有机融合、协同工作的结果。

六、参考文献（5 分）

[1] 国家密码管理局. GM/T 0002-2012 SM4 分组密码算法[S]. 北京: 中国标准出版社, 2012.

[2] 国家密码管理局. GM/T 0001-2012 ZUC 序列密码算法[S]. 北京: 中国标准出版社, 2012.

[3] Bradski G, Kaehler A. Learning OpenCV: Computer Vision with the OpenCV Library[M]. O'Reilly Media, 2008.

[4] Tanenbaum A S, Wetherall D J. Computer Networks (5th Edition)[M]. Prentice Hall, 2011.

[5] 华为技术有限公司. openGauss 数据库管理员指南[M/OL]. 2023.

合肥工業大學

信息系统软件设计

报告

学　　院　　计算机与信息学院

专　业　班　级　　电子信息工程 23 级 2 班

学生姓名及学号　　谢宇博　2023212083

指　导　教　师　　苏兆品 张国富 李小红 臧怀娟

课　题　名　称　　数据库+TCP+音视频传输水印加密系统

2026 年 1 月 6 日

一、课题概述

本课程要求综合运用 openGauss 数据库、TCP 通信、多线程编程等技术，实现信息采集、传输、存储全流程的系统构建，结合即时音视频通信的实际应用需求与音视频信息安全防护、版权溯源的行业痛点，催生了兼具基础通信功能与安全保障能力的系统设计需求 —— 既需要通过技术实现音视频的实时采集、传输及相关信息的规范存储，也需依托前沿水印技术与加密算法，为音视频数据添加安全防护屏障，防范篡改、盗用等风险，实现通信功能与安全溯源的一体化落地。

二、课题任务

本课题任务要求在华为鲲鹏香橙派上设计模拟 QQ 软件，基于多线程编程技术捕捉摄像头和麦克风实时数据，基于 socket 通信设计发送端、接收端两个部分，完成音视频的采集、水印加密、水印嵌入、传输、提取水印、解密水印、存储，并且要基于 IEEE/ACM 顶刊或顶会文献复现 2 种音频或视频水印技术，复现至少一种国密算法对水印进行加解密，并集成到系统中。

三、技术方案及关键问题

3.1 技术路线

技术路线围绕三层核心逻辑展开：

（1）基础层：使用 Qt Qthread 多线程包，使用 PyAudio 和 OpenCV 异步采集音频和视频数据，通过 TCP 按 “4 字节长度+数据块” 格式实现端到端传输；

（2）安全层：采用 DWT 小波变换或者通过 CNN 自编码器提取水印图像特征完成音频水印嵌入和提取，结合 SM4 国密算法实现水印加解密；

（3）支撑层：依托 psycopg2 对接 PostgreSQL 存储音视频文件路径，通过 Qt 信号槽机制实现线程间通信，保障界面无卡顿。

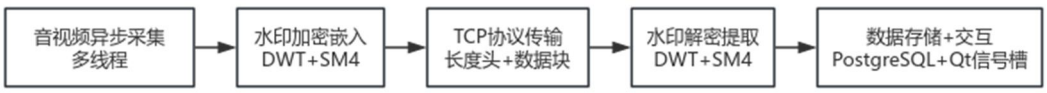


图 3. 1 技术路线流程图

3.2 关键问题

（1）音视频传输：同步传输导致主线程堵塞，数据分块不完整，连接异常导致资源为释放等问题；

（2）水印加密问题：水印序列或者是图像特征映射到一维的序列不满足 SM4 加密要求的 16 字节长度的整数倍；

四、系统设计实现及测试

4.1 系统总体架构设计

本系统基于华为鲲鹏香橙派平台开发，设计了客户端和服务器，实现了音视频实时通信、水印加密嵌入与提取、数据库存储等功能。系统总体架构如图 4.1 所示，包含三大核心模块：音视频传输模块，水印安全模块和数据库管理模块；

系统总体架构图			
客户端层	网络传输层	服务器层	数据库层
音视频采集 水印嵌入 本地存储 用户界面	TCP 协议 多线程传输 数据分包 流控制	音视频接收 水印提取 水印解密 实时播放 录制功能	openGauss 路径存储 记录管理
华为鲲鹏香橙派平台			

图 4. 1 系统总体架构

4.2 音频传输模块实现

4.2.1 多线程异步传输

为解决同步传输导致的界面卡顿问题，系统采用 PyQt5 的 QThread 多线程，将音视频采集、网络传输、界面更新分离到不同线程中。如下面代码所示，客户端创建 CameraThread 和 AudioThread 两个独立线程分别处理视频和音频数据。

```
# 客户端线程设计
class CameraThread(QThread):
    def run(self):
        # 视频采集与传输逻辑
        pass

class AudioThread(QThread):
    def run(self):
        # 音频采集与传输逻辑
        Pass
```

4.2.2 TCP 可靠传输协议

音视频数据通过 TCP 协议传输，采用”封装头+数据体"的封包格式，封包格式如图 4.2 所示：

（1）视频帧传输数据

字段名	字节数	类型	描述
帧宽度	4	uint32	视频帧的宽度，单位为像素
帧高度	4	uint32	视频帧的高度，单位为像素
FPS	4	float32	视频的帧率，单位为帧/秒
图像数据	N	bytes	JPEG 压缩后的图像数据，N 为压缩后图像大小

图 4.2 视频封装数据格式

（2）音频帧传输数据

字段名	字节数	类型	描述
数据长度	4	uint32	后续音频数据的字节数，用于数据完整性校验
音频数据	N	bytes	PCM 编码的音频数据，N=数据长度字段值

图 4.3 音频封装数据格式

4.2.3 连接异常处理机制

为防止连接异常导致的资源泄漏，在该系统实现了以下的异常处理机制：

（1）异常检测机制

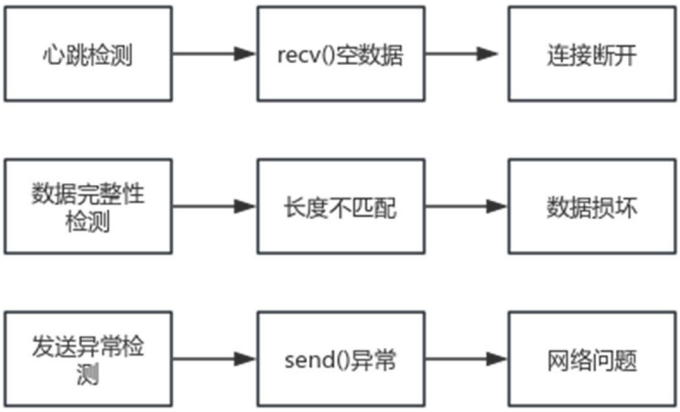


图 4.4 异常检测机制

（2）资源回收机制

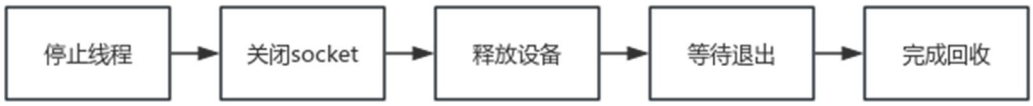


图 4.5 资源回收机制

（3）客户端重连机制

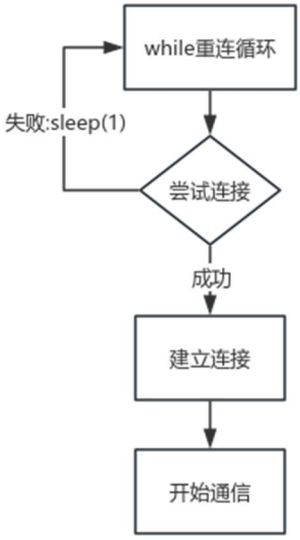


图 4.6 客户端重连机制

4.3 水印安全模块实现

4.3.1 基于 DWT 的水印方案

基于 DWT 的水印方案基本原理是将音频进行 DWT 分解，并将水印通过一定的数学模型嵌入到低频分量中，DWT 分解的小波基是 haar,水印嵌入的数学模型如下：

$$\begin{cases} m_i = SyncCode + SM4enc \\ c_i' = c_i - mod(c_i, S) + \begin{cases} \frac{3S}{4}, & if\ m_i = 1 \\ \frac{S}{4}, & if\ m_i = 0 \end{cases} \end{cases}$$

其中 S 为量化的步长，c_i为经过 DWT 分解后的低频系数，SyncCode是使用 m 序列生成的 63 位同步码，用于水印定位，mod 为取余运算，$SM4enc$是加密后的水印。

对水印进行提取，核心是先找到同步码的位置，定位水印起始点，同步码的结束位置就是水印的起始点，先将 SM4 加密后的水印解密，再根据下面公式还原原始比特：

$$bit = \begin{cases} 1 & mod(c_i', S) \geq \frac{S}{2} \\ 0 & mod(c_i', S) < \frac{S}{2} \end{cases}$$

4.3.2 基于 CNN 自编码器的深度学习水印方案

本方案使用卷积自编码器提取图像特征，将其作为水印嵌入音频信号中，自编码器包含两部分，编码器和解码器，结构如图 4.7 所示。

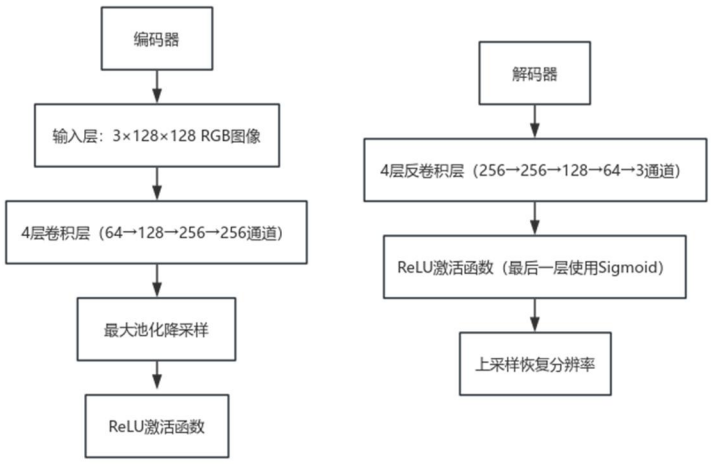


图 4.7 自编码器结构

通过编码器提取图像的 256 维特征向量，将特征值归一化到[0,1]范围，将归一化特征嵌入到音频小波高频系数，公式如下：

$$cD_{mod} = cD_{ori} + \alpha \times Feature_{norm}$$

其中，cD_{ori}是音频 sym8 小波基小波分解后的原始高频系数，α是嵌入强度系数，本设计根据平衡透明性和鲁棒性选择嵌入强度为 0.005，$Feature_{norm}$是归一化后的特征向量。

提取水印的基本步骤是根据嵌入公式反推得到归一化特征向量的计算公式得到归一化特征向量：

$$Feature_{norm} = \frac{cD_{mod} - cD_{ori}}{\alpha}$$

将特征向量去归一化后将 256 维度特征向量送入解码器中得到重建后的水印图像。

4.4 数据库模块设计

使用 openGauss 数据库存储音视频文件路径信息，创建的表结构如下：

```
CREATE TABLE media_info (
    id SERIAL PRIMARY KEY,
    VedioPath VARCHAR(500) NOT NULL,
    AudioPath VARCHAR(500) NOT NULL,
);
```

主要封装了两个函数，通过 psycopg2 库实现数据库操作，主要函数包括：

```
insert_media_path_to_db(vedio_path, audio_path) #插入媒体路径
extract_media_path_from_db()                    #查询最新媒体路径
```

4.5 用户界面 UI 设计

设计的 UI 界面如下，可以实时显示连接的状态，录制时也会显示录制的时长和上面录制的日期，并且客户端和服务端都可以实时显示音视频数据：

图 4. 8 UI 界面设计展示

4.6 系统测试与性能分析

4.6.1 系统环境搭建

本系统设计基于华为鲲鹏香橙派，使用 miniconda 管理虚拟环境，环境所需要的库如下表所示：

软件组件	版本号	功能说明
Python	3.8	核心开发语言
PyQt5	5.15.9	图形用户界面框架
OpenCV	4.10.0	图像处理与摄像头采集
PyAudio	0.2.14	音频采集与播放
PyWavelets	1.4.1	小波变换处理
Librosa	0.11.0	音频信号处理
Psycopg2	2.9.10	openGauss 数据库连接

表 4. 1 系统环境依赖

4.6.2 客户端录制音视频

客户端录制视频时，UI 界面可以实时显示录制的状态以及上次录制的时间，界面显示如下图所示，录制结束后会将保存的路径保存至数据库：

图 4. 9 界面显示正在录制以及上次录制的时间

```
                vediopath                |                      audiopath

-----------------------------------------+-----------------------------------
----------------------
 ./data_record/output_20251224081529.avi | ./data_record/watermarked_DWT_recordo
ut_20251224081529.wav
 ./data_record/output_20251224092357.avi | ./data_record/watermarked_DWT_recordo
ut_20251224092357.wav
 ./data_record/output_20251224094836.avi | ./data_record/watermarked_DWT_recordo
ut_20251224094836.wav
 ./data_record/output_20251224125519.avi | ./data_record/watermarked_DWT_recordo
ut_20251224125519.wav
 ./data_record/output_20251224132019.avi | ./data_record/watermarked_DWT_recordo
ut_20251224132019.wav
 ./data_record/output_20251224132615.avi | ./data_record/watermarked_DWT_recordo
ut_20251224132615.wav
(6 rows)
```

图 4. 10 保存路径的数据库

4.6.3 接收端提取水印

接收端通过读取数据库保存的路径来读取发送端保存的音视频，并提取水印，成功提取水印如下图所示：

```
媒体路径已成功读取数据库
[[1 0 1 1 1 1 1 0 1 1]
 [1 0 1 1 1 1 1 0 1 1]
 [1 0 1 1 1 1 1 0 1 1]
 [1 0 1 1 1 1 1 0 1 1]
 [1 0 1 1 1 1 1 0 1 1]
 [1 0 1 1 1 1 1 0 1 1]
 [1 0 1 1 1 1 1 0 1 1]
 [1 0 1 1 1 1 1 0 1 1]
 [1 0 1 1 1 1 1 0 1 1]
 [1 0 1 1 1 1 1 0 1 1]]
```

图 4. 11 接收端提取水印

4.6.4 基于 DWT 的水印性能分析

基于 DWT 的水印的 SNR 与量化步长 S 息息相关，当 S=0.25 时 SNR 有 34.89 dB，下面是在不同噪声攻击嵌入水印的音频时提取水印的误码率的柱状图。

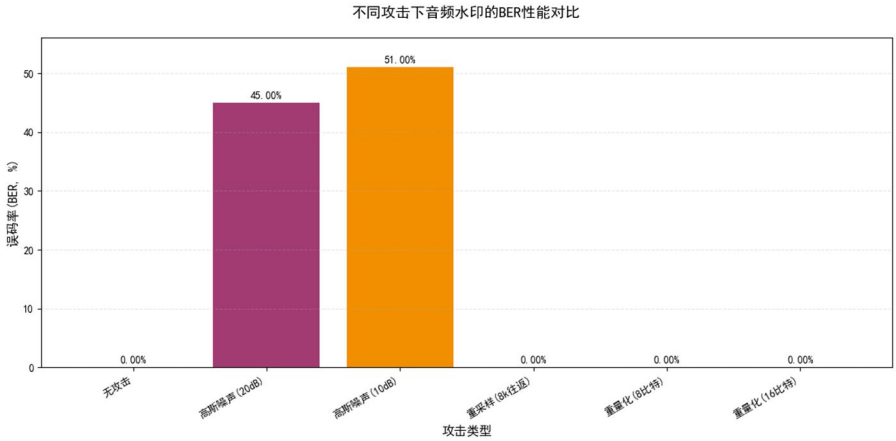


图 4. 12 不同攻击下音频水印的 BER 性能对比

4.6.5 基于 CNN 自编码器的水印性能分析

自编码器的模型需要训练，训练的数据集是 pytorch 自带的 CIFAR10 数据集，数据集和验证集的划分是 9：1，损失函数使用 MSE Loss,使用 Adam 优化器，训练的参数如下表所示：

参数描述	参数	取值
训练总轮次	epoch	30
批次大小	batch_size	32
学习率	Lr	1e-3

表 4. 2 自编码器训练参数

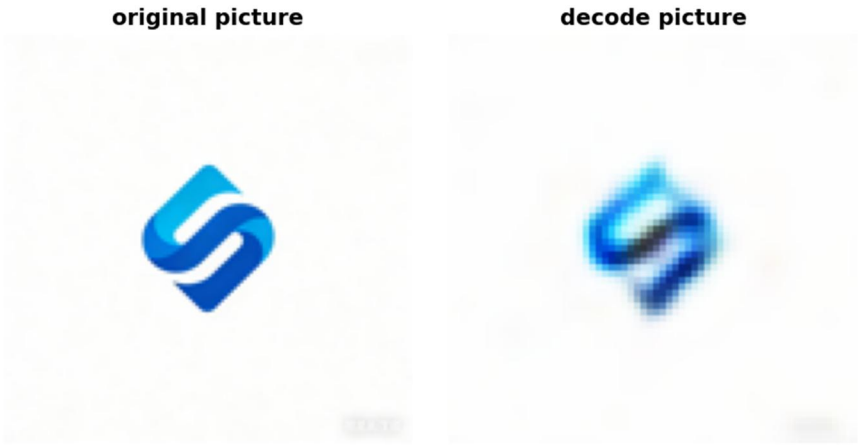


图 4. 13 原始图像和解码出来的图像(MC=0.92363,BER=1.343%,SNR=39.56dB)

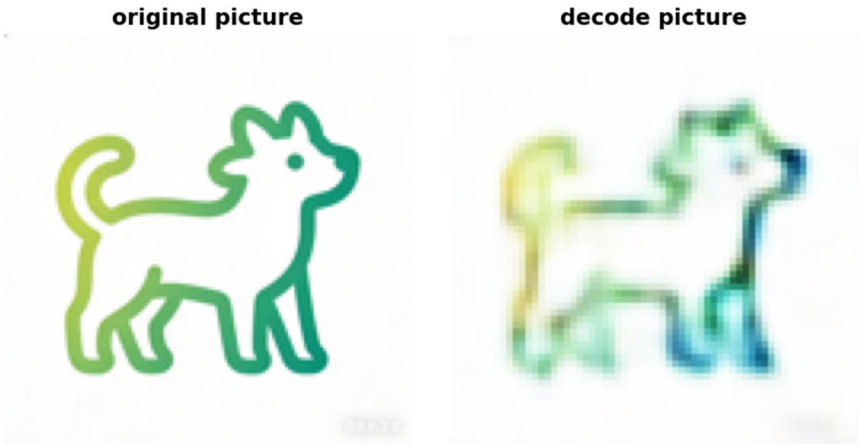


图 4. 14 MC=0.83168,BER=2.606%,SNR=39.56dB

通过上面两张图片的编码和解码，信噪比和误码率都控制在较好的水平，第一张图片的自相关程度有 0.92363，说明解码出来的图片还原度较高，但是第二张图片的相关程度只有 0.83168，说明解码出来的效果不是很好，也意味着自编码器模型还有提升的空间。

4.6.6 水印性能比较

通过本次系统设计，对两个水印的性能区别有了一定的了解，以下是两个水域的性能比较

对比维度	基于 DWT 的水印方案	基于 CNN 自编码器的水印方案
嵌入容量	较低，依赖低频系数量化	较高，可嵌入 256 维特征向量
鲁棒性	对噪声、压缩有一定抵抗能力，量化步长 S 敏感	对常见攻击（如噪声、滤波）鲁棒性较强

透明性	较好（SNR≈34.89 dB）	良好（SNR≈39.56 dB）
计算复杂度	较低，适合实时系统	较高，需训练模型，适合离线或预处理场景
抗攻击能力	在低强度攻击下表现良好，BER 较低	在高强度攻击下仍保持较高的 MC 值与较低的 BER

表 4.3 水印性能比较

五、课程设计总结与心得体会（5 分）

5.1 设计总结

本次设计基本完成了课题的任务，基于华为鲲鹏香橙派实现了一个音视频水印加密的模拟 QQ 软件。系统完成了音视频的实时采集与传输、基于 DWT 与 CNN 自编码器的双水印嵌入与提取、SM4 国密算法加密以及 openGauss 数据库存储等核心功能，整体流程完整实现。

在创新性上，本此设计将传统信号处理与深度学习水印方法融合于同一系统，并集成了国 SM4 密码算法，增强了整个系统的安全性和综合性。此外，在多线程架构与异常处理机制上也做了针对性设计，提升了系统的稳定性和用户体验。

然而，系统仍存在一些不足之处。深度学习水印方案计算开销较大，难以满足实时性要求较高的场景；音视频同步机制在网络波动时表现不够稳健；用户界面较为基础，交互设计尚有提升空间。这些问题反映了本人在系统优化与工程经验上的欠缺。

针对以上不足，后续可从以下方向进行改进：采用轻量化神经网络模型或模型量化技术以降低计算负担；引入自适应缓冲与同步策略以增强网络鲁棒性；进一步完善界面功能与视觉设计，提升系统整体完成度。

5.2 心得体会

通过本次系统设计与实现，我深刻体会到理论知识与工程实践之间的紧密联系。在课堂上学到的网络通信、数字信号处理等知识，在项目中得到了具体应用与验证。尤其是在调试音视频同步、处理水印嵌入边界问题、优化数据库查询等过程中，我逐渐养成了系统性分析问题、逐步定位根源的思维习惯。

此次设计也让我初步认识到一个完整系统的复杂性与多维性。技术方案的选取需要在性能、资源、实现难度之间进行权衡；模块之间的接口设计与异常处理往往比核心算法更考验开发者的细致程度。此外，在复现论文算法时，我意识到许多细节在文献中并未充分展开，需要通过实验反复验证与调整，这一过程极大锻炼了我的动手能力和耐心。

在未来的学习中，我将更加注重代码的规范性、架构的可扩展性以及文档的完整性，努力在理论与实践的融合中进一步提升自己的工程素养与综合能力。

六、参考文献（5 分）

[1] WU S Q., HUANG J W., HUANG D R., et al. Self-Synchronized Audio Watermark in DWT Domain[C]. IEEE International Symposium on Circuits and Systems., 2003: 616-620.

[2] PATIL A, SHELKE R, HIRAN D. AudioStamp: A Deep Learning Based Watermarking Procedure for Copyright Protection of Digital Audio Files[J]. SSRG International Journal of Electronics and Communication Engineering.

[3] 国家密码管理局. SM4 分组密码算法[S]. 北京：中国标准出版社， 2012.

[4] openGauss Documentation Team. openGauss 5.0 数据库管理指南

[5] Summerfield M. Rapid GUI Programming with Python and Qt: The Definitive Guide to PyQt Programming[M]. Upper Saddle River: Prentice Hall, 2007.

合肥工業大學

信息系统软件设计

报告

学　　院　　计算机与信息学院

专　业　班　级　　电子信息工程 23 级- 2 班

学生姓名及学号　　王子轩　2023212085

指　导　教　师　　苏兆品

课　题　名　称　　数据库+TCP+音视频传输水印系统

2025 年 12 月 31 日

一、课题概述 （四号、宋体，5 分）

（正文小四、宋体，行间距 1.25 倍行距。**简要概述题目的背景即可，切勿长篇大论**）

1.1 课题概述

当前，多媒体监控数据在传输与存储环节面临着严峻的安全挑战，数据极易被截获、篡改或伪造[4]。传统的监控系统往往只关注画面的连通性，而忽视了内容的版权保护与来源认证能力。针对这一痛点，本课题基于 Qt 框架与 OpenGauss 数据库[5]，设计并实现了一套集成了数字水印溯源与国密算法安全传输的音视频监控系统。

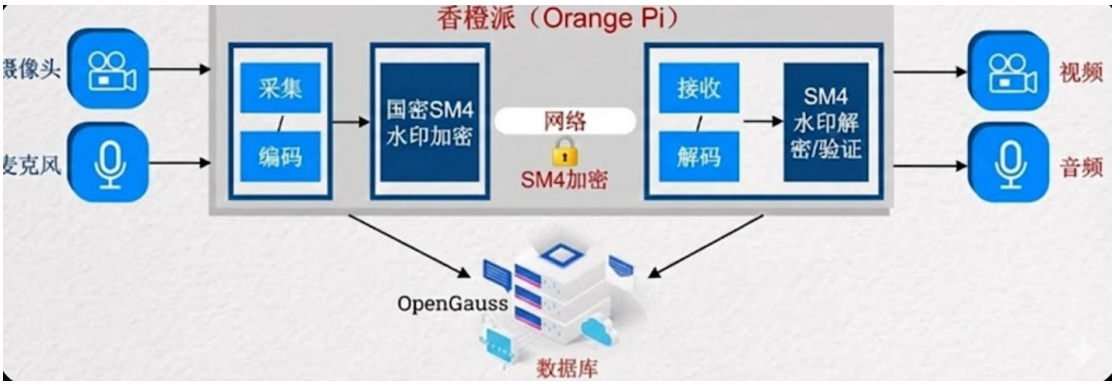
在设计核心通信架构上，本系统并未采用简单的“纯 TCP 透明传输”方案。虽然 TCP 协议能提供可靠的字节流服务，但在高吞吐量的实时视频场景下，其强制重传机制会导致严重的 Head – of – Line Blocking 队头阻塞效应，引发画面累积延迟。

为了解决这一工程难题并契合工业界流媒体的主流设计规范，本系统创新性地采用了“TCP 信令 + UDP 数据流”的双通道混合架构[6]：

严格保留 TCP 协议用于传输控制指令（包括了录制启停、密钥交换等）与文本消息，确保信令传输的零丢失与高可靠性。

引入 UDP 协议承载音视频载荷，通过应用层的各种优化策略换取低延迟效果，以实现最大程度地模拟 QQ 等在线视频通话软件效果。

同时，本系统结合了 DCT 域的 QIM 水印算法与 SM4 国密算法[2]，实现了对视频内容的“隐形加密”与水印存储，从而构建了一个兼顾实时性、可靠性与安全性的全流程解决方案。



(图 1 系统总框图)

二、课题任务（5 分）

（**根据题目要求简要概述你要设计系统的具体功能，实现什么目标，切勿长篇大论**）

本系统要实现的具体功能是；模拟商业级通讯软件（如 QQ）的核心音视频通话功能，并在此基础上构建一套具有“事前防御（加密）”与“事后追责（溯源）”能力的端到端监控平台。系统由客户端与服务端两部分构成，主要实现以下四大核心功能：

2.1 音视频实时交互

(i)支持客户端:采用香橙派采集 USB 摄像头与麦克风数据，通过网络实时回传至服务端显示与播放；同时支持服务端向客户端发送控制指令与文本消息。

(ii)同时要实现低延迟传输：针对视频数据与音频数据，系统采用 UDP 协议进行封装传输，优化了在弱网环境下的画面流畅度，解决了传统 TCP 传输导致的画面卡顿与延迟累积问题。

2.2 数字水印嵌入与溯源

(i) 频域水印（DCT + QIM）将版权信息嵌入到视频帧 Y 通道的 DCT 中频系数中，该区域对压缩和滤波攻击具有极高的鲁棒性[3]。

(ii) 时域水印（LSB）:利用最低有效位算法[7]在像素域嵌入辅助信息，用于快速提取验证。

2.3 国密算法加密水印

SM4 加密：对提取出的水印明文信息（这里采用时间戳来确定是什么时间发出和解密出的水印以防止在调试时出现混淆）采用国密 SM4 分组密码算法进行加密处理，确保敏感数据在信道传输及数据库存储过程中的机密性，防止窃听。

2.4 Opengauss 数据库存储

(i) 系统采用“服务端录像 + 录音”的录制策略。服务端负责将实时画面的合成视频文件以及音频文件保存至本地磁盘，同时将写入的路径同时显示在 ui 界面以及写入数据库中便于查找。

(ii) OpenGauss 数据库集成：这里使用 ODBC/QtSql 接口深度对接华为 OpenGauss 数据库，自动建立 video_files 、audio_files、watermarks_files 数据表，将录制文件的存储路径、生成时间、提取出的水印等数据结构化入库，实现数据的管理。

三、技术方案及关键问题（30 分）

3.1 全链路架构设计

本系统构建了一个完全基于 ARM 架构的端到端安全监控闭环，系统采用分层模块化设计，逻辑上分为数据采集层、核心处理层、网络传输层与应用展示层。

硬件载体：发送端与接收端均部署于鲲鹏嵌入式开发板，运行国产化的 Linux 操作系统，充分利用 ARM 处理器的多核优势。

并发模型：解决了视频流处理的高并发需求

(1) 采集线程负责从摄像头读取原始 YUV/RGB 数据并推入输入队列；

(2) 处理线程从队列取出数据，并行执行水印嵌入与 SM4 加密；

(3) 传输线程负责将处理后的数据打包并通过 UDP/TCP 协议栈发送。这种解耦设计有效避免了单线程下的 IO 阻塞，确保了视频流的低延迟传输。

3.2 GUI 设计

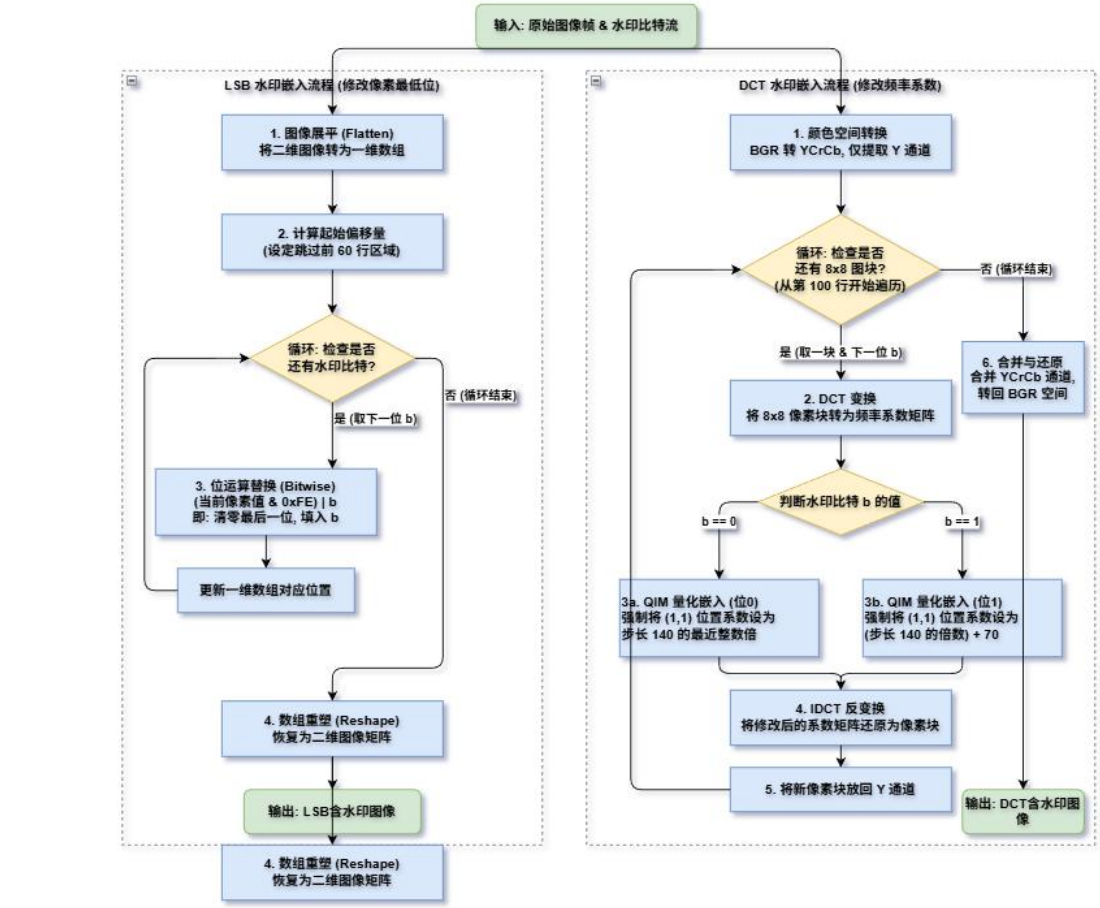
与在 PC 端方案不同，本设计面临着嵌入式平台严格的硬件资源限制。为了在有限算力下实现流畅的人机交互，本课题选用了 Qt 5.14 (Linux 版) 作为 GUI 框架，并进行了以下深度优化：

1. 信号与槽机制的异步化：将耗时的网络连接、数据库查询操作移至工作线程（Worker Thread），通过 Qt 的 Signal-Slot 机制与主界面线程通信，杜绝了界面“假死”现象。
2. Linux Framebuffer 直写技术：利用 Qt 对 Linux Framebuffer 的底层支持，绕过 X11 窗口系统的部分开销，直接对显存进行操作。
3. 渲染管线优化：在接收端，针对 UDP 接收到的 JPEG 图像流，采用了“双缓冲”绘制策略，减少屏幕闪烁；同时复用 QImage 内存对象，避免每帧重复 malloc/free 带来的内存碎片问题。

3.3 核心算法实现

由于嵌入式环境算力相对较弱的特点，本系统在实现核心安全算法时，摒弃了通用的“暴力计算”模式，转而采用“感兴趣区域优化”与“底层指令集优化”策略。

(1)轻量化频域水印算法（DCT + QIM）



(图 3-1 2 种水印算法对比)

为了平衡水印的隐蔽性与鲁棒性，系统实现了基于离散余弦变换（DCT）的盲提取算法，具体工程实现如下：

(i) 色彩空间降维分离： 视频帧首先从 BGR 空间转换为 YCrCb 空间。由于人眼对亮度（Y）变化的敏感度低于色度（Cr/Cb），且 JPEG 压缩主要对色度采样，因此系统仅提取 Y 通道 进行水印嵌入。这不仅大幅提升了水印的隐蔽性，还减少了$\frac{2}{3}$的 DCT 运算量。

（ii） ROI 区域锁定策略：视频监控中边缘区域常被裁剪或视为无关背景，锁定图像垂直方向的第 60 至 140 行作为“感兴趣区域（ROI）”。仅对该区域进行8×8分块 DCT 变换，确保在鲲鹏开发板上也能维持实现不卡顿地运行。

（iii）QIM 量化索引调制：系统采用 QIM 技术嵌入信息。选取 DCT 中频系数$C(U,V)$（通常为位置 (1,1)），设定量化步长 $\alpha = 50.0$。嵌入公式如下：

$$C' = round\left(\frac{C}{\alpha}\right) \times \alpha + \delta \times \frac{\alpha}{2} \qquad (3-1)$$

其中$\delta \in (0,1)$为待嵌入的二进制比特。接收端无需原始图片，仅需计算$C' \ mod \ \alpha$的值即可判定比特位，实现了真正的盲检测。

（2） 国密 SM4 分组加密算法实现

为了保障视频内容在传输过程中的机密性，系统并未依赖第三方库，而是底层复现了国家密码管理局发布的 SM4 分组密码算法。

(i) 算法结构：采用 32 轮非线性迭代结构，包括 S 盒字节替换、行移位与列混淆变换。

(ii) 工作模式：选用 CBC（密码分组链接）模式。前一帧的密文块参与当前帧的异或运算，使得相同的明文块（如静止画面的背景）加密后产生不同的密文，有效抵抗了重放攻击与模式分析攻击。

(iii) PKCS#7 填充：为了适配 SM4 的 128 位分组长度，对不满 16 字节的数据块进行了标准的 PKCS#7 填充，确保数据完整性。

3.4 异构网络通信协议设计

采用了“TCP 信令 ＋UDP 数据流”的双通道混合通信架构，解决了单一协议无法兼顾可靠性与实时性的问题。

(1) 应用层帧头定义

在 UDP 数据通道（端口 8888）中，为了解决 UDP 无序、无状态的问题，系统设计了紧凑的应用层协议头：

(表 3-1 协议帧头)

字节偏移	字段名称	长度	描述
0x00	Header Type	1 Byte	0x01:视频帧，0x02:音频帧
0x01	Algo Flag	1 Byte	0x01:DCT 0x02:LSB
0x02 – 0x05	Frame Seq	4 Bytes	帧序号，用于接收端乱序重排
0x06 – End	Payload	N Bytes	实际接收数据

视频流处理:接收端识别到 0x01 帧头后，剥离协议头，将 JPEG 数据送入硬解码器。

音频流处理:识别到 0x02 后，将 PCM 数据存入环形缓冲区，以平滑网络抖动，防止音频播放出现“爆音”或卡顿。

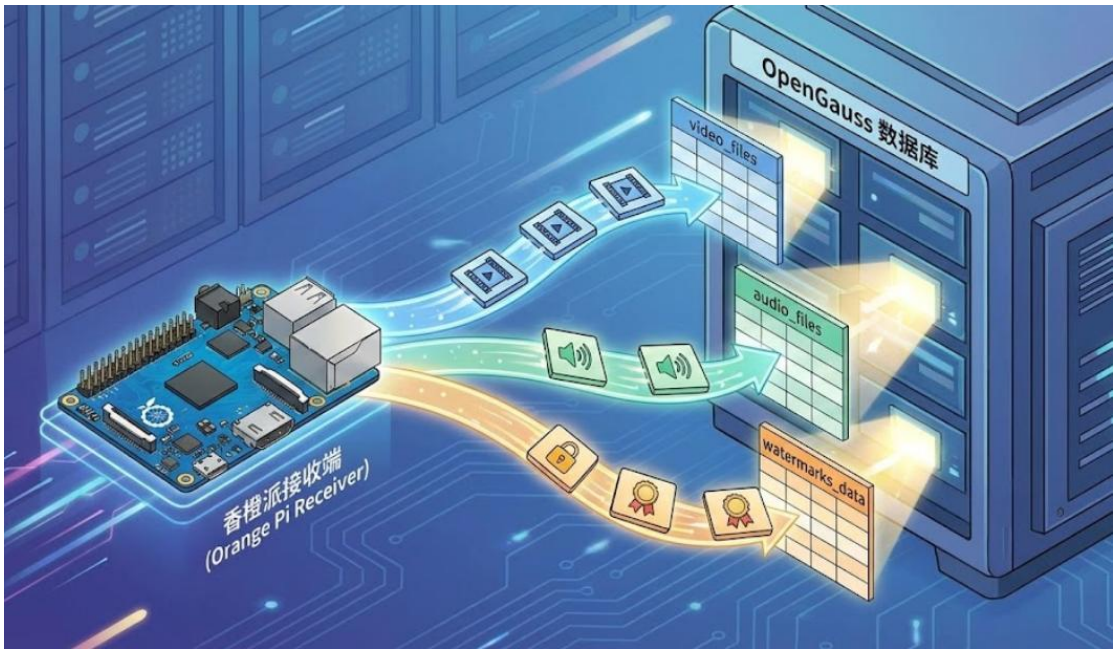
(2) TCP 信令同步机制

系统建立了可靠 TCP 连接（端口 9999）用于控制流。为了解决 UDP 丢包可能导致的状态不同步问题，设计了双向握手状态机： 当服务端发起 CMD_START_RECORD 指令时，不会立即改变本地状态，而是进入 WAIT_ACK 状态；只有当接收到客户端回传的 ACK_OK 包后，双方才同步进入录制模式。这种机制消除了分布式系统中的临界区竞态风险。

3.5 数据库存储

系统后端深度集成华为 OpenGauss 数据库，利用 Qt 的 QSqlDatabase 模块驱动 ODBC 实现直连:

程序启动时会自动检测 video_files 等 3 个表是否存在，若不存在则通过 CREATE TABLE 语句自动初始化，降低了部署难度。



(图 3-2 香橙派与外接 OpenGauss 数据库)

(1) 数据库表结构设计

系统启动时会自动进行模式检查，通过 CREATE TABLE IF NOT EXISTS 语句初始化核心业务表：

(i) t_sys_user：存储用户凭证。为了安全，密码字段不存储明文，而是存储经处理后的摘要值。

(ii)t_video_log：存储监控日志，包含 video_id (主键), filepath (存储路径), algorithm_type (水印类型), capture_time (捕获时间) 等字段。

(iii)t_access_log：记录系统的登录与操作行为，用于事后审计。

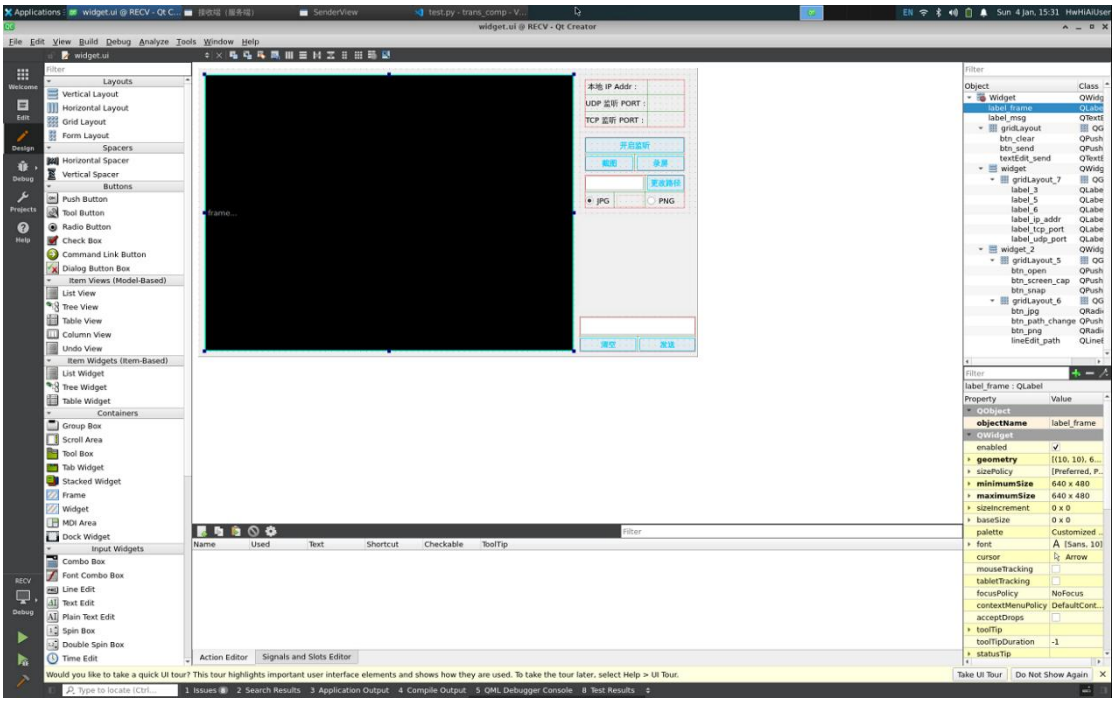
(2) 防注入与事务管理

在执行 SQL 操作时，系统全面采用 QSqlQuery::prepare() 预编译语句，通过占位符绑定参数，从根本上杜绝 SQL 攻击。同时，对于涉及多表更新的操作，开启数据库事务（Transaction），确保数据的一致性与原子性。

四、系统设计实现及测试（50 分）

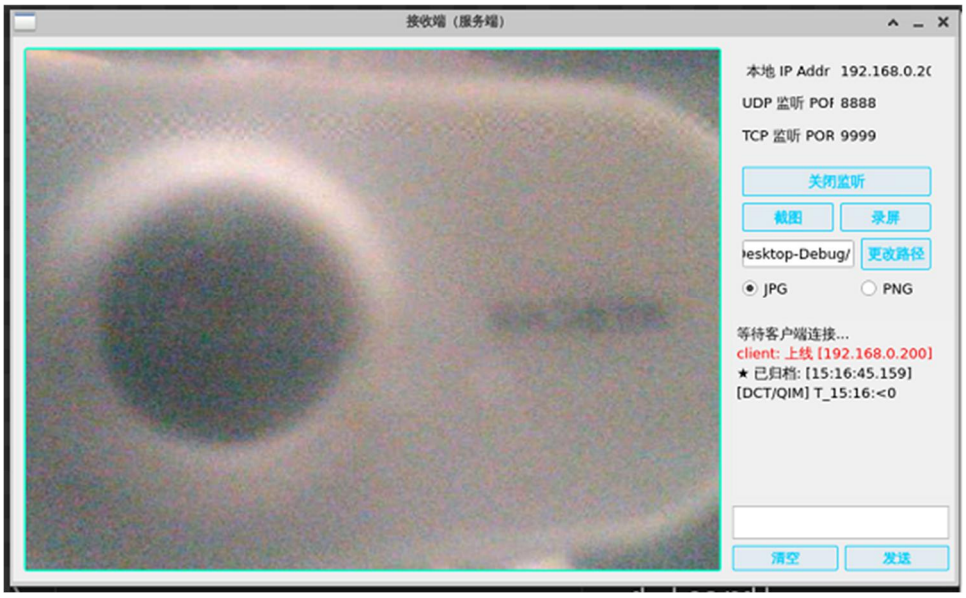
4.1 系统 UI 交互设计

接收端界面基于 Qt Designer 进行可视化布局设计，生成的 ui_widget.h 文件由 C++ 逻辑类直接调用。为了适应开发板香橙派的屏幕分辨率，界面采用了自适应的 Grid Layout 网格布局。



(图 4-1-1 UI 界面设计)

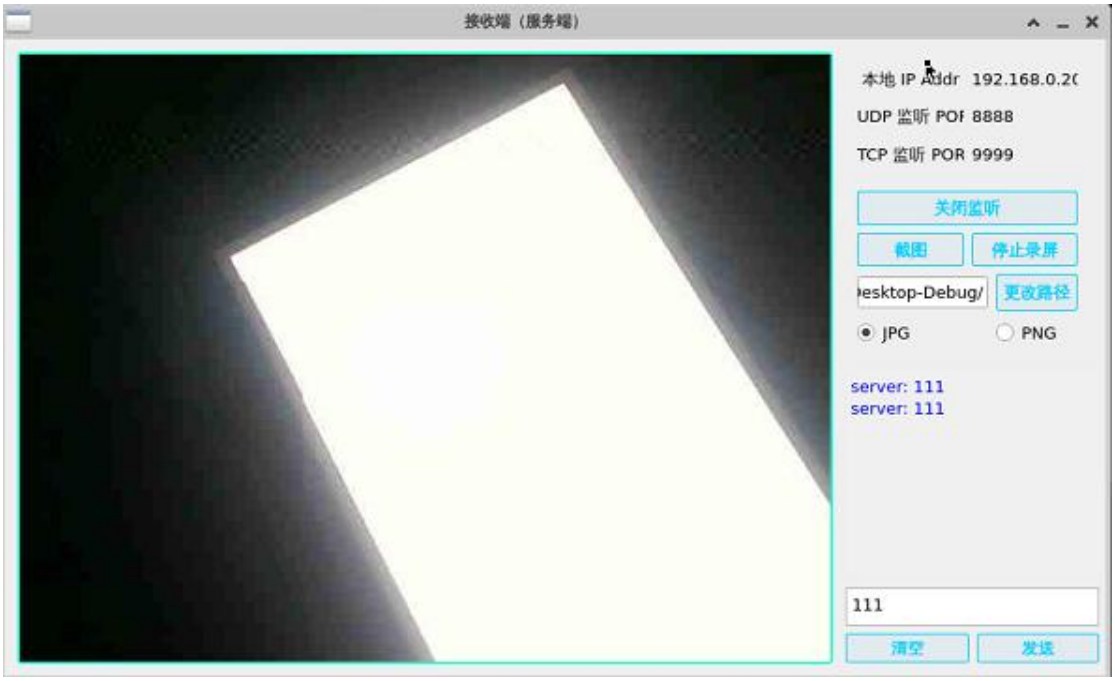
(1) 实时监控区（左侧）：使用 QLabel 控件作为视频渲染容器。通过重写 paintEvent 或直接使用 setPixmap 方法，将解码后的 QImage 帧逐帧绘制到界面上。



(图 4-1-2 实时监控区)

(2) 日志与状态区（右下）：使用 QTextEdit 控件实时显示系统运行日志（如“等待客户端连接”、“客户端已上线”等）。该控件设置为只读模式，并开启自动滚动功能。

(3) 同时可以模拟”QQ”进行信息发送的对话。



(图 4-1-3 对话系统)

(4) 控制面板（右上）：包含“录屏”、“截图”、“停止录制”等 QPushButton。按钮状态通过布尔变量 is_recording_button_active 进行互斥管理，防止用户重复点击。

如图 4-1-4，当按下”录频”按钮后，该按钮随即转变成”停止录屏”等待再次被点击。

(图 4-1-4 录屏按键变换)

由于任务要求实现水印加密与性能指标评估，因此在发送端设置键盘控制逻辑: 通过数字输入切换加密模式以及水印攻击模式:

(图 4-1-5 发送端模式设置)

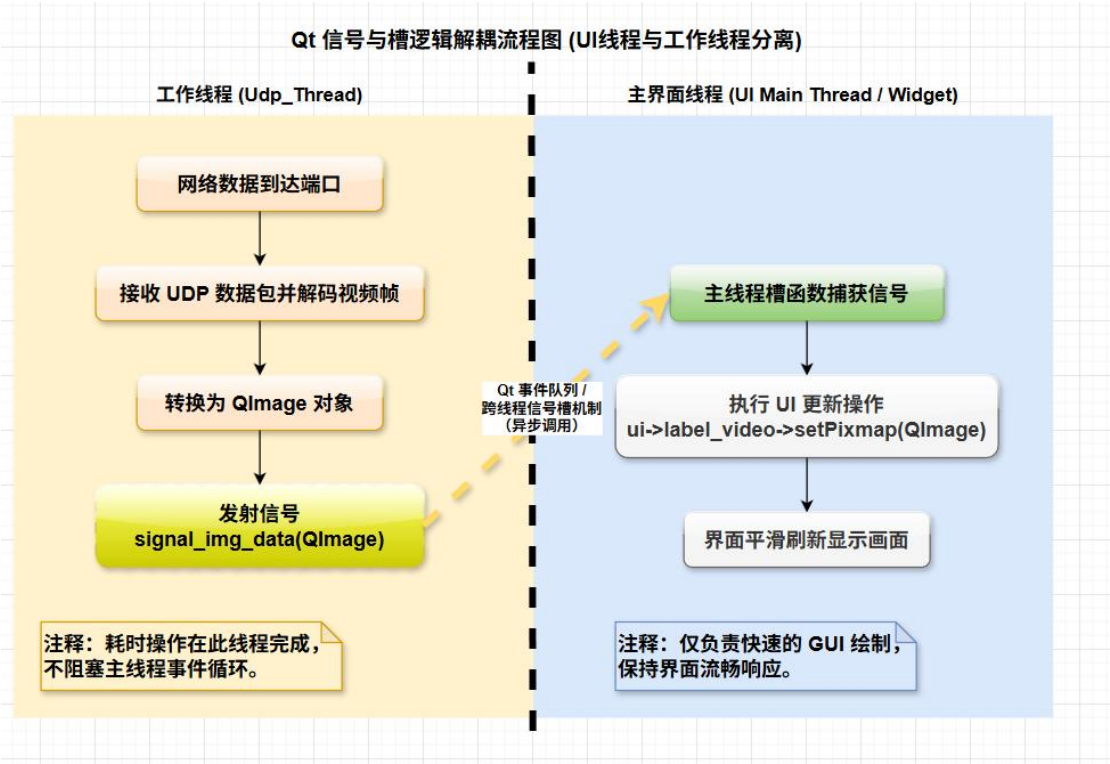
(图 4-1-6 键入后的显示结果)

4.1.2 信号与槽的逻辑解耦

为了避免界面卡顿，系统严格遵循 Qt 的“UI 线程与工作线程分离”原则。

(1)主要信号流：当 Udp_Thread 接收并解码完一帧图后，发射 signal_img_data(QImage) 信号。

(2)槽函数响应：主线程 Widget 类槽函数捕获信号，并执行 Ui_label_video->setPixmap() 更新画面。 这种设计确保了即便网络数据量激增，UI 响应依然保持流畅，不会出现“未响应”的假死状态。



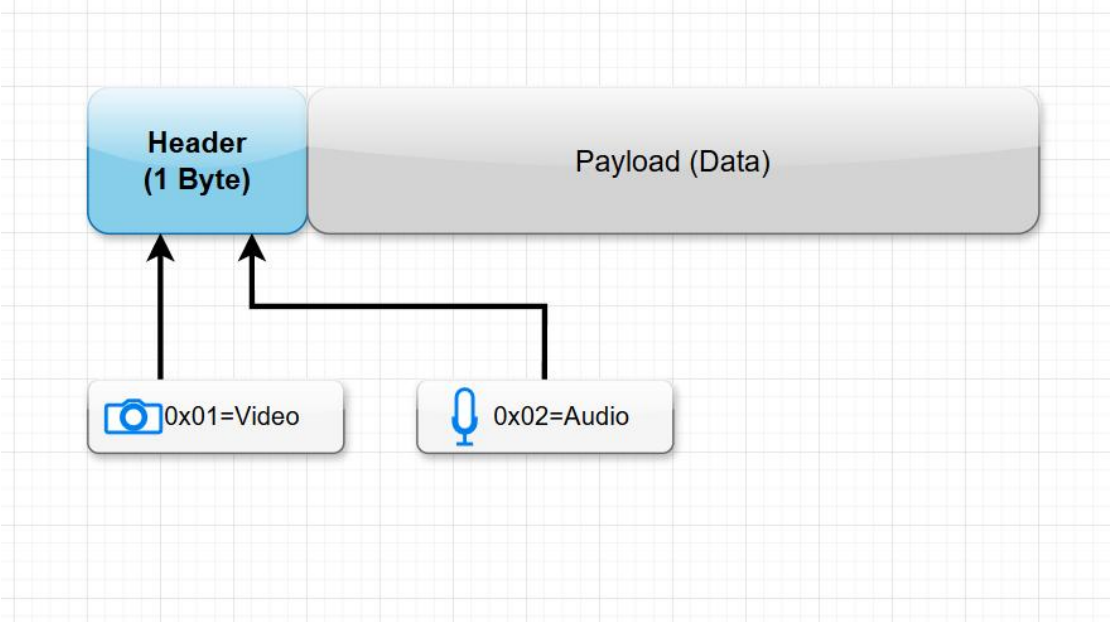
(图 4-1-7 Qt 信号与槽逻辑解耦流程)

4.2 异构网络通信模块实现

通信模块是连接嵌入式采集端与监控端的桥梁。针对题目要求的“TCP+音视频传输”，本系统在实现 TCP 控制链路的基础上，针对视频流特性进行了 UDP 协议扩展。

4.2.1 自定义应用层数据包结构

由于 UDP 是面向报文的无连接协议，接收端必须能够识别数据包的类型是图像还是音频。在应用层定义了如图 4-2-1 所示的帧结构：



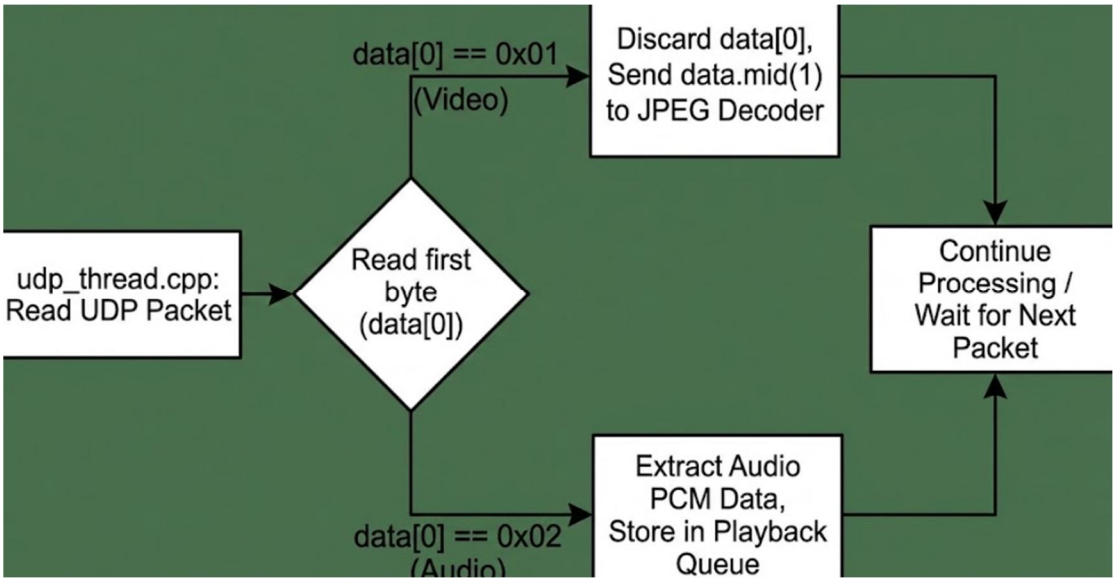
(图 4-2-1 帧头设置)

在 udp_thread.cpp 中，解析逻辑如下：

(1)读取数据包的首字节 data[0]。

(2)若 data[0] == 0x01，则丢弃首字节，将剩余数据（data.mid(1)）送入 JPEG 解码器。

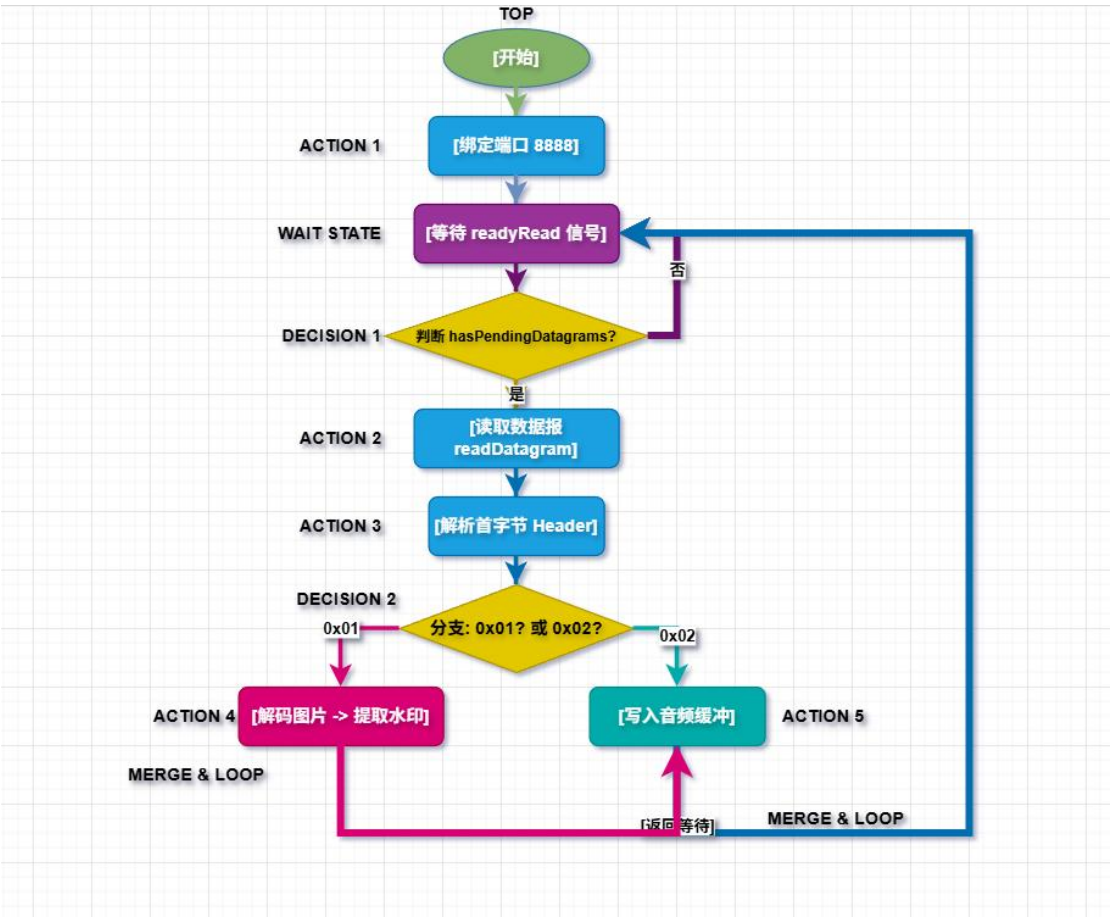
(3)若 data[0] == 0x02，则提取音频 PCM 数据存入播放队列。



(图 4-2-2 udp 解析流程)

4.2.2 UDP 接收线程流程设计

为了处理视频流带来的高并发数据，Udp_Thread::run() 函数采用了基于事件循环的异步处理机制。具体处理流程如图 4-2-3 所示：



(图 4-2-3 udp 接收线程逻辑)

4.2.3 TCP 双向即时通信模块实现

为了实现监控端（Qt）与采集端（香橙派）之间的实时信息交互（如发送报警通知、回传设备状态），系统基于 TCP 协议设计了一套全双工文本对话机制。该模块允许两端通过对话框形式互发文本消息，且互不干扰视频流的传输。

1. 发送端的多线程收发设计

在 send.py 中，考虑到 input()函数会阻塞主程序，为了不影响视频采集推流，发送端采用了多线程架构：

(1)发送逻辑（主循环）：在主线程中通过 input('请输入要发送的内容：') 获取用户输入，将其编码后直接通过 self.tcp.send() 发送给服务端。

(2)接收逻辑（守护线程）：利用 threading.Thread 创建一个守护线程运行 recv_data 函数。该函数通过 socket.recv(1024)持续监听来自 Qt 端的回复消息，一旦收到数据立即解码并在终端打印，实现了“边发边收”的效果。

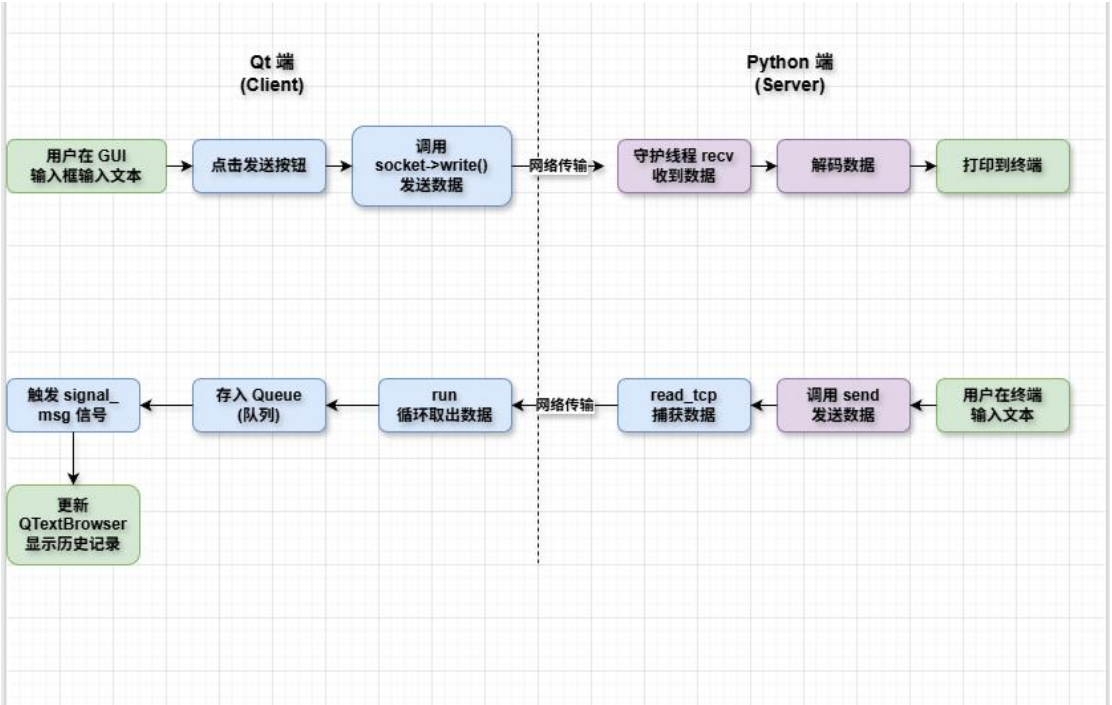
2. 监控端（Qt）的生产者-消费者模型

在 tcp_thread.cpp 中，服务端面临着“网络线程接收数据”与“UI 线程显示数据”的同步问题。为此，本系统设计了一个基于互斥锁的消息队列缓冲机制：

(1)入队（生产者）： 当 QTcpSocket 触发 readyRead 信号时，read_tcp() 槽函数被调用。读取套接字缓冲区中的所有字节数据，并将其压入 bytes_queue 队列中。

(2)出队（消费者）： run() 函数在一个独立的 while(start_flag) 循环中运行。它不断检查 bytes_queue 是否非空。一旦有数据，立即加锁取出，同时通过 emit signal_msg() 发送信号通知 UI 界面更新对话框内容。

整个文本交互流程如图 4-2-4 所示：



(图 4-2-4 文本交互流程)

4.3 数字水印算法实现与对比分析

为了全面评估不同水印算法在网络信道中的鲁棒性，本系统采用了“分层验证”策略。主系统采用抗压缩的 DCT 算法进行实时流传输，而对于极其脆弱的 LSB 算法，本设计构建了独立的测试模块（基于 lsb_sender.py 与 lsb_receiver.py），采用 PNG 无损编码进行对照实验，以避免 JPEG 有损压缩对实验数据的干扰。

4.3.1 LSB 时域水印实现

LSB（Least Significant Bit）算法是一种典型的空域隐写技术，其核心思想是利用像素值的最低有效位来携带秘密信息。考虑到该算法对像素值的微小变动极其敏感，本实验进行了如下针对性设计：

(1) 数据序列化与嵌入逻辑

(i)二进制转换：系统首先将待发送的字符串（"SEQ:”）转换为 8 位二进制流，并在头部拼接 32 位长度标识，以解决接收端不定长读取的问题。

(ii)位平面替换：将图像矩阵展平（Flatten）为一维数组，通过位运算 pixel = (pixel & 0xFE) | bit 逐像素嵌入信息。这种操作仅改变像素值的奇偶性，人眼无法察觉。

(2) 传输层关键优化（PNG 无损编码）

在早期测试中发现，若直接复用主系统的 JPEG 视频流通道，LSB 水印会因 JPEG 的量化过程而完全丢失。为此，本模块做出了以下重大调整：

(i) 编码格式切换：发送端强制采用 PNG 无损压缩格式 (cv2.imencode('.png')) 对嵌入水印后的图像进行编码，确保像素值在传输前后严格一致。

(ii) 分辨率适配：由于 PNG 压缩率低于 JPEG，且包含高频噪声（水印的图像熵值较高，为了防止单帧数据超过 UDP 协议的 MTU 限制（导致 IP 分片重组失败），本实验将 LSB 测试分辨率锁定为 150x150，并将 PNG 压缩等级设为最高的 9，以平衡包大小与解码速度。

(3) 动态攻击模拟

为了直观对比算法性能，发送端集成了实时攻击控制台。操作者可通过按键实时施加不同强度的干扰（如高斯噪声、椒盐噪声、亮度调整等），接收端同步显示提取结果与误码状态。

4.3.2 DCT 频域 QIM 水印实现[1]

采用 DCT-QIM 算法针对视频流传输进行了深度优化，代码实现位于 udp_thread.cpp 的 extractWatermark_qim 函数中。

(1)感兴趣区域（ROI）锁定：

为了降低 ARM 处理器的计算负载，算法并未对 1080P 全图进行变换，而是通过指针偏移 img_y + (row * width)仅截取了 Y 通道的第 60 至 140 行进行处理。

(2)量化嵌入策略：

选取8×8 DCT 块的中频系数$C(1,1)$，设定量化步长 alpha = 50.0。

(i)嵌入（发送端）:若水印位为 '1'，将系数调整为 alpha 的奇数倍；若为 '0'，调整为偶数倍。

(ii)提取（接收端）:接收端无需原始图片，直接计算 round(coeff / 50.0) 的奇偶性即可盲提取数据。

4.3.3 两种算法的鲁棒性对比测试

为了全面评估系统在复杂网络环境下的可靠性，提前设计了包含几何攻击（剪切）、信号处理攻击（压缩、噪声）和增强攻击（亮度）在内的 5 种典型场景。测试样本为香橙派实时采集视频流，每组测试重复 10 次取平均值。

(表 4-3 水印提取测试结果)

测试场景	攻击描述	LSB 提取结果	DCT 提取结果	结论
无攻击	局域网正常传输，不进行任何干预	正常	正常	本系统无攻击模式下可以正常传输音视频，
JPEG 压缩	模拟网络带宽不足，此时质量因子设置为 30	提取率 0	提取率>95%	DCT 算法对图像压缩具有良好的抗性 LSB 算法对图像压缩毫无抵抗力
	模拟网络带宽不足，此时质量因子设置为 50	提取率 0	提取率 100%	
	模拟网络带宽不足，此时质量因	提取率 0	提取率 100%	

	子设置为 80			
画面剪切	遮挡视频下方部分 20%	提取率 100%	提取率 100%	只要水印够”轻便”，就不会被遮挡住，从而正确的被提取出来
	遮挡视频下方部分 50%	提取率 100%	提取率 100%	
	遮挡视频下方部分 70%	提取率 100%	提取率 100%	
	遮挡视频下方部分 80%	提取率 100%	提取率 100%	
	遮挡视频下方部分 90%	提取率 100%	提取率 0	
	遮挡视频下方部分 100%	提取率 0	提取率 0	
亮度攻击	Beta 提升整体亮度 beta=10	提取率 100%	提取率 100%	DCT 算法对亮度具有一定的抗性，但随着 beta 值逐渐增大，提取的准确率明显下降 对图像所有像素统一加上一个偶数，完全不会改变 LSB 的值。所以 LSB 完美解出水印
	Beta 提升整体亮度 beta=15	提取率 100%	提取率 100%	
	Beta 提升整体亮度 beta=20	提取率 100%	提取率>95%	
	Beta 提升整体亮度 beta=25	提取率 100%	提取率<80%	
亮度攻击	Beta 提升整体亮度 beta=30	提取率 100%	提取率<20%	
	Beta 提升整体亮度 beta=50	提取率 100%	提取率 0	
高斯噪声	模拟光照噪点 sigema=10	提取率 0	提取率>95%	大约有 50% 的像素变化量是奇数（±1, ±3...），另外 50% 是偶数。这意味着水印数据中，有一半的比特（50%）被翻转了，50% 的误码率 = 纯
	模拟光照噪点 sigema=15	提取率 0	提取率≈0	
	模拟光照噪点 sigema=20	提取率 0	提取率 0	

				随机乱码。
椒盐噪声	添加椒盐噪声颗粒	提取率>90%	提取率 0	LSB 空域的算法:坏点只影响该点本身，未被污染的像素仍可读。 DCT 频域的算法:一个噪点会污染整个 8 ×8块的频域系数

以下给出详细测评:

(1) 对照组—无攻击模式(none)

如图，100%解密出水印

(图 4-3-1 对照组)

(2) 实验组—高斯噪声攻击(gaussian_noise)

(图 4-3-2 系数 sigema = 10)

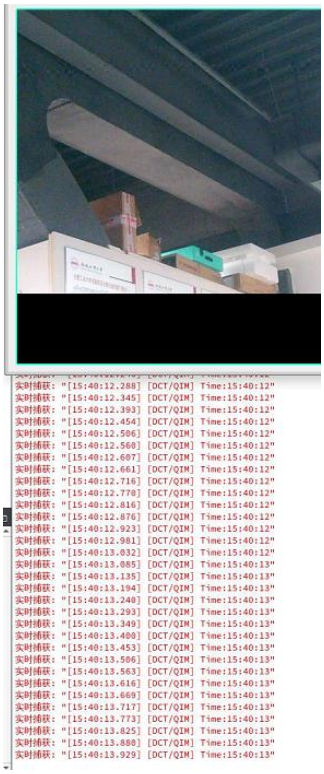
(图 4-3-3 系数 sigema = 15)

(图 4-3-4 系数 sigema = 20)

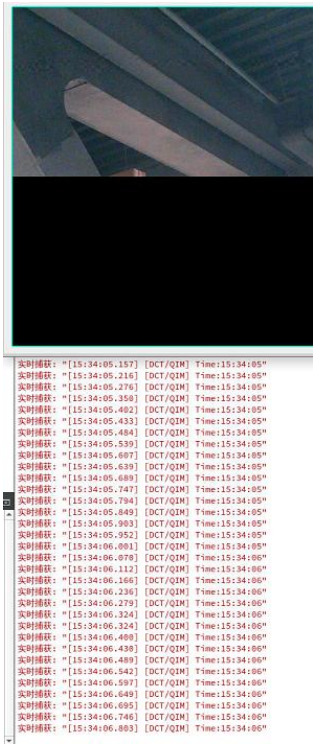
(3) 实验组--椒盐噪声(salt_pepper)

(图 4-3-5 校验噪声)

(4) 实验组—画面剪切(cropping)

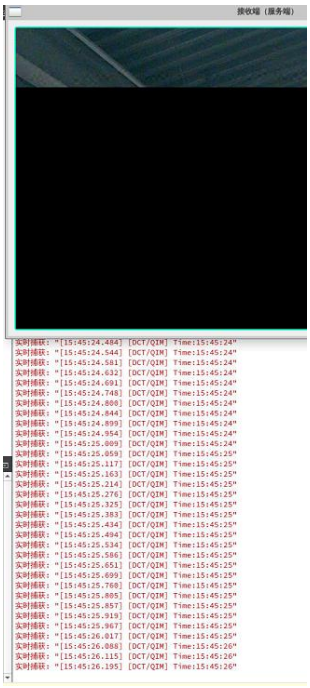


(图 4-3-6 遮挡 20%)

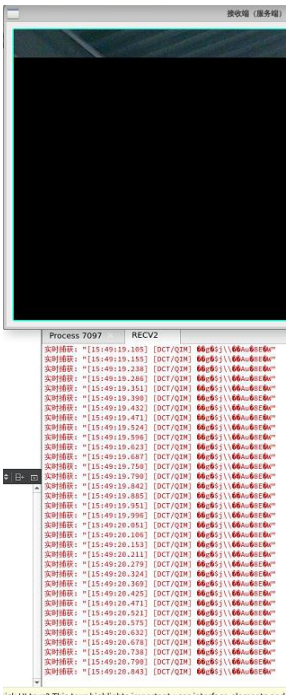


(图 4-3-7 遮挡 50%)

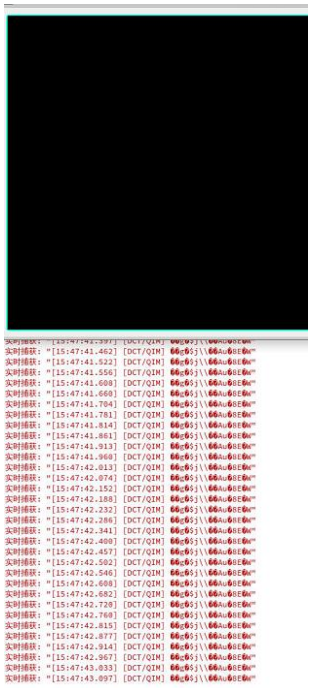
(图 4-3-8 遮挡 70%)



(图 4-3-9 遮挡 80%)



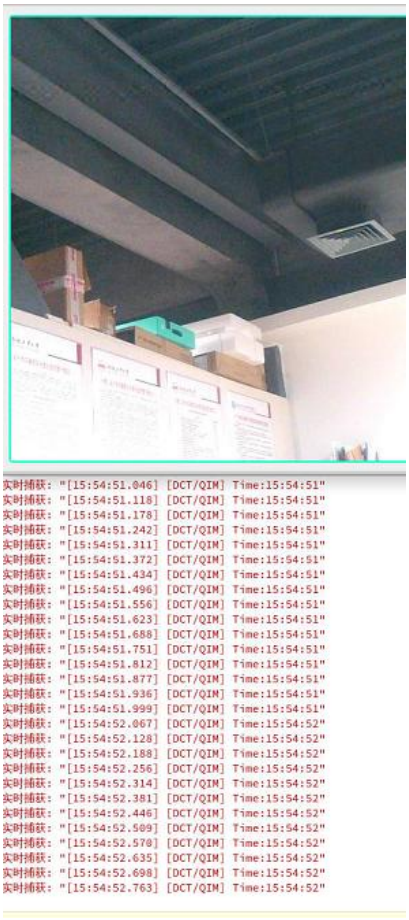
(图 4-3-10 遮挡 90%)



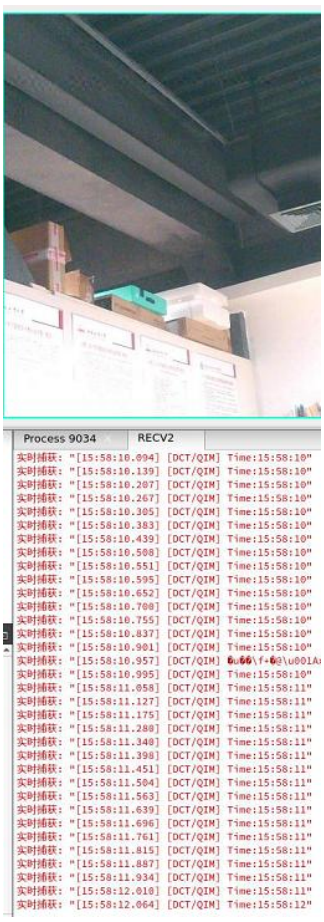
(图 4-3-11 遮挡 100%)

(5)实验组—亮度攻击(brightness)

(图 4-3-12 beta=10)



(图 4-3-13 beta=15)



(图 4-3-14 beta=20)

(图 4-3-15 beta=25)

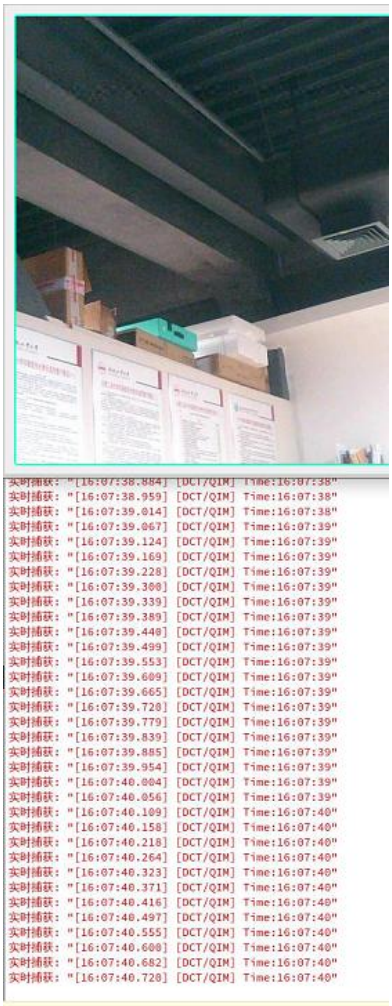
(图 4-3-16 beta=30)

(图 4-3-17 beta=50)

(6)实验组—压缩攻击(compression)

(图 4-3-18 Q=30)

(图 4-3-19 Q=50)

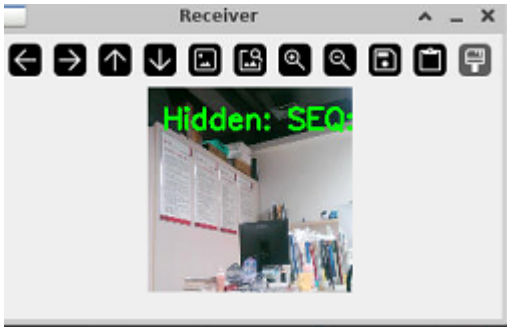


(图 4-3-20 Q=80)

以下给出 LSB 的提取结果，由于这种算法对压缩抗性表现较差，故需要使用降低分辨率来确保信息能完整传输:

(1) 对照组—无攻击模式(none)

如图，100%解密出水印



(图 4-3-21 对照组)

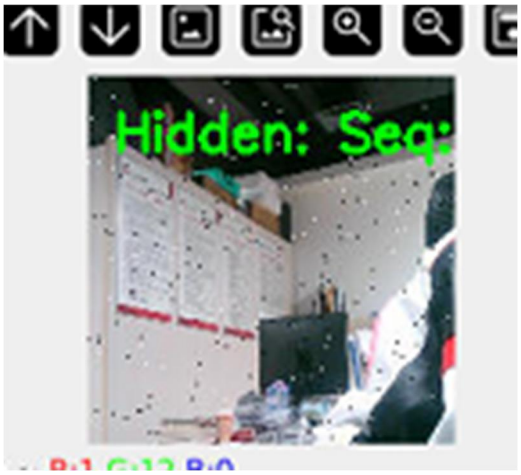
(2)实验组--高斯噪声攻击(gaussian_noise)

(图 4-3-22 sigma=1)

(图 4-3-23 sigma=5)

(图 4-3-24 sigma=15)

(3) 实验组--椒盐噪声(salt_pepper)

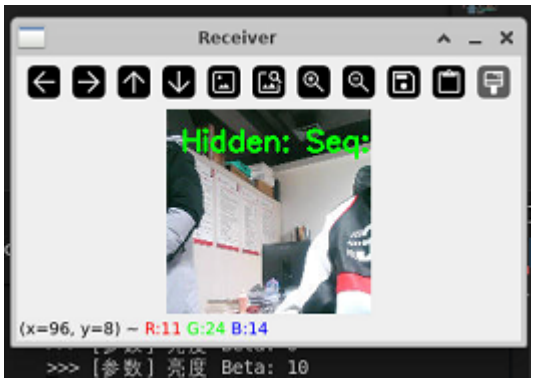


(图 4-3-25 椒盐噪声 1)

（图 4-3-25-1 椒盐噪声 2）

实验结果呈现：椒盐噪声攻击下，偶尔会解错误一个符号(如图 4-30-25-1)，但是普遍情况是可靠的提取出水印。

(4) 实验组—亮度攻击(brightness)



(图 4-3-26 beta=10)

(图 4-3-27 beta=20)

(图 4-3-28 beta=30)

(图 4-3-29 beta=40)

(图 4-3-30 beta=50)

(5) 实验组—画面剪切(cropping)

(图 4-3-31 剪切 20%)

(图 4-3-32 剪切 50%)

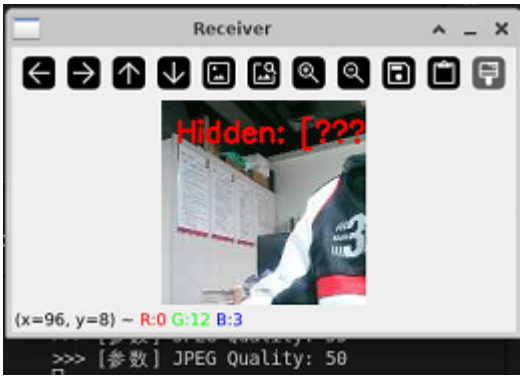
(图 4-3-33 剪切 70%)

(图 4-3-34 剪切 80%)

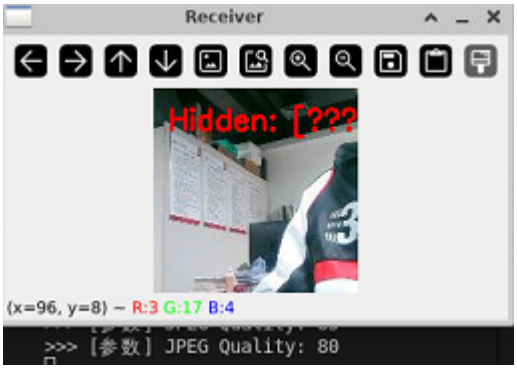
(图 4-3-35 剪切 90%)

(6)实验组—压缩攻击(compression)

(图 4-3-36 Q=30)



(图 4-3-37 Q=50)



(图 4-3-38 Q=80)

本实验分别对基于空域的 LSB（最低有效位）算法和基于频域的 DCT（离散余弦变换）算法进行了独立的抗攻击性能测试。实验结果揭示了两种算法截然不同的物理特性。

1. 噪声攻击对比分析

(1) 高斯噪声 (Gaussian Noise)

现象对比：当施加 Sigma=5 的微弱高斯噪声时，DCT 算法仍能保持 100%的提取率，而 LSB 算法提取出的信息瞬间变为乱码。

原理分析：高斯噪声属于全域微小扰动。它会随机地对图像中几乎所有像素进行$\pm 1, \pm 2$的数值修改。对于 LSB 算法而言，二进制最低位的奇偶性对± 1的变化极其敏感，全图范围的像素修改导致了所有比特位的随机翻转。而 DCT 算法利用量化步长（alpha=50）构造了容错区间，微小的像素波动在频域量化过程中被忽略，因此表现出极强的鲁棒性。

(2) 椒盐噪声 (Salt Pepper)

现象对比：在密度为 0.05 的椒盐噪声攻击下，DCT 提取结果出现大量错误，而 LSB 算法能解出绝大部分清晰可读的文字（误码率仅约为 5%）。

原理分析：

(i)LSB (空域)椒盐噪声是离散的坏点。5%的噪声密度意味着 95%的像素完全未被触碰，因此 LSB 仅丢失了 5%的数据，剩余数据完好无损。

(ii)DCT (频域)变换具有能量扩散特性。一个8×8像素块中只要存在 1 个极值的噪点（0 或 255），其巨大的能量波动在变换后会扩散到该块的所有频域系数中，导致整个块的水印信息被破坏。这也表明，未经中值滤波预处理的 DCT 算法对脉冲型噪声极其脆弱。

(3) 亮度攻击 (Brightness)

实验现象：将画面亮度统一增加数值时，LSB 算法提取完全无误，而当亮度过度增加导致画面过曝时，DCT 算法反而失效。

原理分析：LSB 所有像素加上一个偶数时，像素数值的奇偶性保持不变（奇+偶=奇，偶+偶=偶），因此最低有效位完全未受影响。

DCT 的 AC 系数理论上对直流分量的变化不敏感，但在高强度亮度攻击下，大量像素发生饱和截断至 255。这种截断效应破坏了图像原本的纹理结构，导致 AC 系数归零，从而使水印丢失。

(4) JPEG 压缩攻击

实验现象：在 Quality=50 的压缩强度下，LSB 水印彻底消失(准确来说小于 80 的情况下都是无法提取的)，这说明 LSB 对压缩没有任何抵抗力，而 DCT 水印存活率 100%。

原理分析：JPEG 压缩算法的核心正是丢弃高频信息（人眼不敏感细节）。LSB 隐写本质上是将信息藏在最高频的噪声层中，因此是 JPEG 压缩的首要“打击对象”。而本系统设计的 DCT 水印嵌入在中低频系数（位置 1,1），恰好位于 JPEG 量化表保留的能量集中区，因此完美契合了抗压缩的需求。

3. 几何剪切攻击 (Cropping)

实验现象：对画面底部 20%~90%进行遮挡剪切。二者基本都能顺利解出水印

原理分析：

(1)LSB：由于采用顺序嵌入，若数据量较大铺满全图，底部被切除将直接导致后半段信息丢失。但是这里由于降低了图像的分辨率，所采用的水印较简单，故能顺利解出不丢失。

(2)DCT：本系统采用了 ROI（感兴趣区域）策略，强制将水印嵌入在图像顶部的 Y 轴 60-140 行区域。这一工程上的“避险设计”使得算法在面对非针对性的几何剪切时表现出了伪鲁棒性，证明了水印嵌入位置选择的重要性。

综上，LSB 算法具有容量大、隐蔽性好但极其脆弱的特点，仅适用于无损环境下的数据完整性校验；而 QIM-DCT 算法虽然在面对脉冲噪声时存在短板，但在抵抗信号处理攻击（压缩、滤波、高斯噪声）方面展现了卓越的鲁棒性，更适合用于版权保护与溯源场景。

4.5 SM4 国密算法的跨语言底层复现与通信对齐

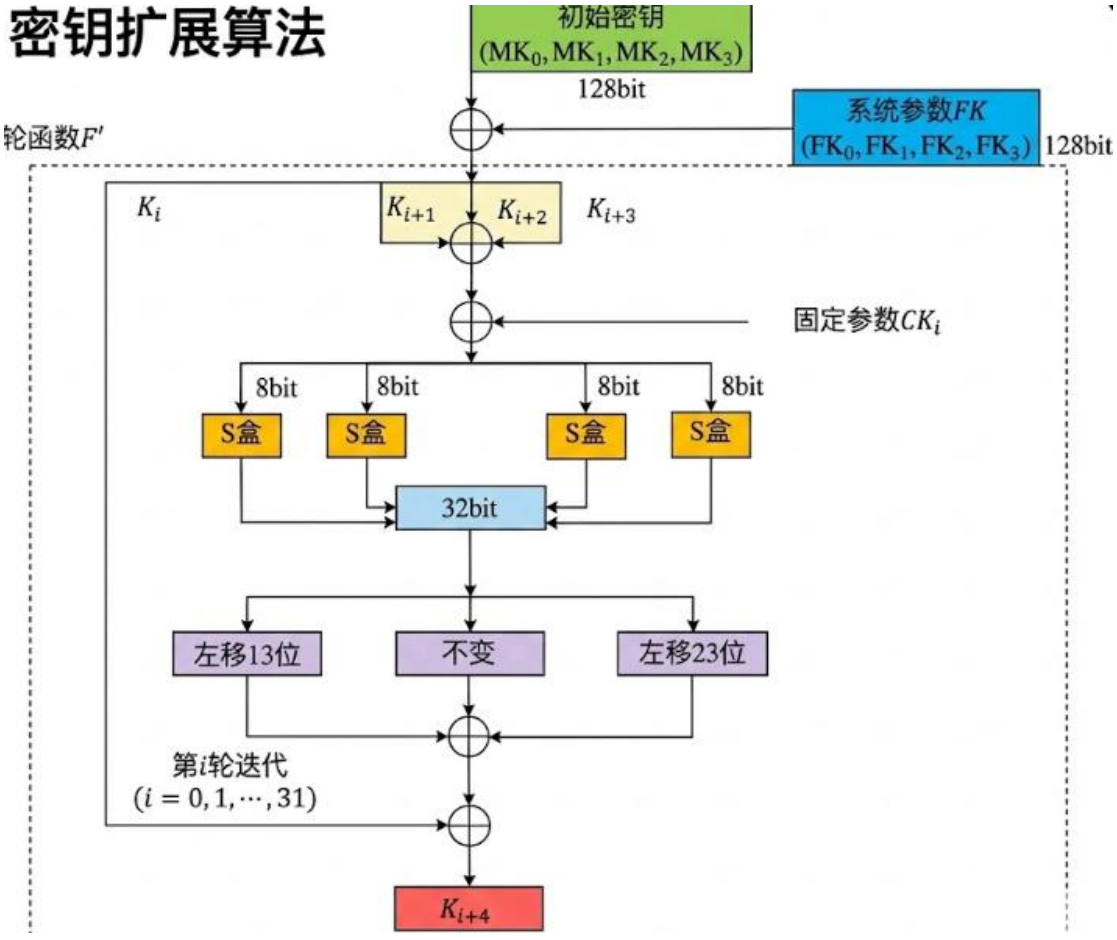
为了深入理解国密算法原理，并解决 Python 发送端与 C++接收端在底层数据表示上的差异，本文未直接调用第三方库，而是完全基于标准文档，在 Python 和 C++双端实现了比特级对齐的 SM4 算法逻辑。这一过程解决了多语言环境下字节序、内存对齐与填充算法不一致的核心难题。

1. 算法核心逻辑复现

SM4 是一种分组密码算法，分组长度和密钥长度均为 128 比特。

(1) S 盒（S-Box）变换：SM4 的非线性核心。我们在代码中硬编码了标准的 S 盒查找表（SBOX）。在复现过程中，特别注意了 C++ 中 char 类型可能的符号位扩展问题，强制使用 uint8_t 无符号类型进行查表，确保了混淆操作的准确性。

(2) 轮函数与迭代：算法采用 32 轮迭代结构。每次迭代将 128 位数据分为 4 个 32 位字进行运算。Python 端利用位运算模拟 32 位整数溢出截断，而 C++ 端通过 uint32_t 类型严格控制数据位宽，确保两端数学逻辑一致。



(图 4-5-1 SM4 算法流程)

2. 异构系统的通信对齐的困难与解决

在实现 Python 到 C++ 的加密视频流传输时，直接调用库函数会导致解密乱码。经深入分析，我手动实现了以下底层逻辑以确保一致性：

(1) 字节序的强制统一： SM4 标准要求使用大端序传输。

(i)问题：ARM 架构的 CPU 内存默认为小端序，而 Python 的整数处理屏蔽了底层细节。直接传输会导致密钥和密文在 C++ 端读取时高低位反转，导致解密失败。

(ii)解决方案：在 C++底层实现中，放弃简单的 memcpy，而是编写了 GET_UINT32_BE 和 PUT_UINT32_BE 宏，通过显式的移位操作，强制将网络字节流还原为正确的大端序整数，实现了跨平台的数据对齐。

(2) CBC 模式与 PKCS#7 填充的手动实现：

由于 UDP 传输的不可靠性及数据长度的不定性，选择了 CBC 模式，并手动实现了标准的 PKCS#7 Padding。

(i)Python 端：计算数据长度，末尾补齐 16 - (len % 16)个字节。

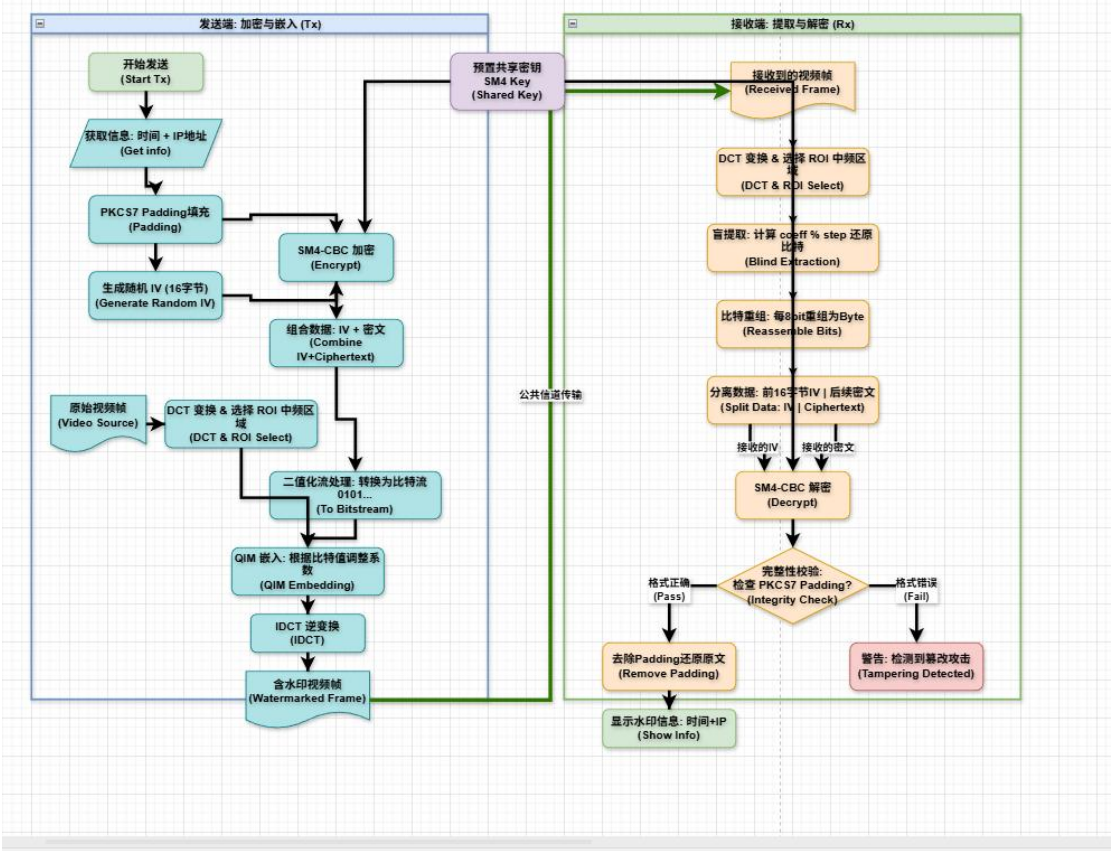
(ii)C++端：解密后读取最后一个字节的值作为 pad_len，并校验末尾数据的完整性，再进行截断。这避免了因填充库实现差异导致的“解密后末尾乱码”问题。

3. 性能与独立性

最终实现的 C++版 SM4 算法（sm4_manual.cpp 以及 sm4_manual.py）不依赖庞大的 OpenSSL 库，具有极高的移植性。手动优化的 C++代码在处理实时视频流水印解密通过了与 Python 端的“控制变量法”基准测试，证明了算法实现的正确性与鲁棒性。

4.6 动态水印的加解密与嵌入全流程

本系统的水印并非静态图片，而是包含时间戳和设备 ID 的动态数据。其完整处理流程如图所示：



(图 4-6-1 流程图)

1.发送端处理流程: 加密与嵌入

(1) 明文生成：获取系统当前时间与 IP 地址作为明文。

(2) SM4 加密:

(i) 对明文进 Padding 填充，使其长度满足 16 字节倍数。

(ii) 生成随机初始化向量，执行 SM4-CBC 加密得到密文。

(3) 二值化流处理：将密文转换为二进制比特流。

(4) QIM 嵌入：遍历 DCT 变换后的 ROI 区域中频系数，根据比特流的值（0 或 1）调整系数的奇偶性。

2. 接收端：提取与解密

(1) 盲提取：对接收到的视频帧进行 DCT 变换，计算 coeff % step 还原出二进制流。

(2) 比特重组：每 8 个比特重组为一个字节，分离出前 16 字节的 IV 和后续的密文。

(3) SM4 解密：利用提取的 IV 和预置密钥 sm4_key 解密数据。

(4) 完整性校验：若解密后的 Padding 格式正确，则显示水印；否则判定为被篡改（攻击检测）。

4.7 音视频传输协议与缓冲区设计

为了在 UDP 不可靠传输的基础上实现音画流畅，系统在应用层设计了精简的封包策略和抖动缓冲机制。

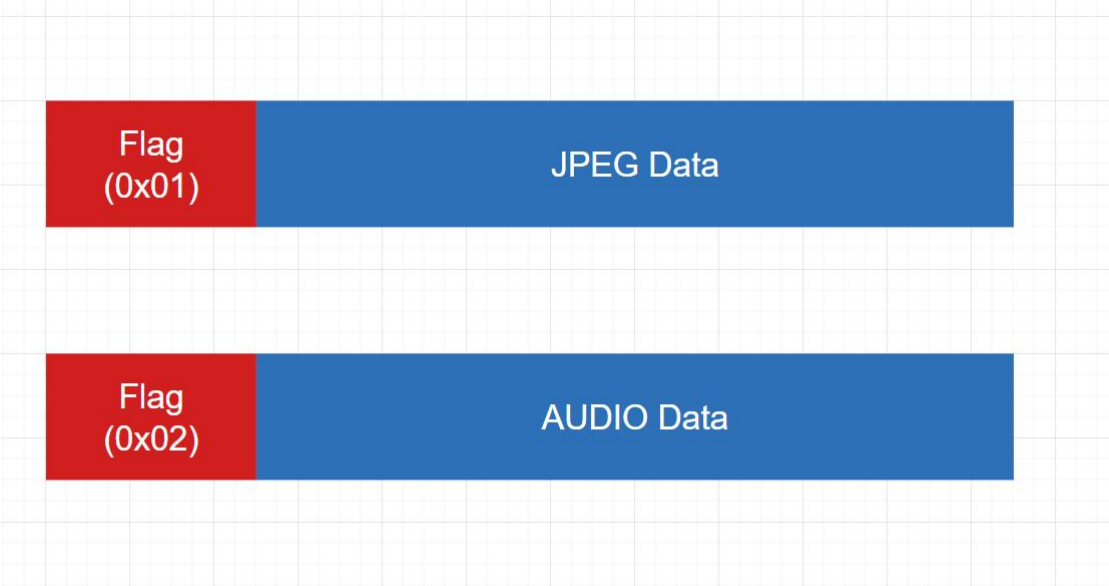
1. 自定义应用层协议

为了在一个 Socket 端口中复用音视频流，系统定义了如下数据包结构：

(1)Header (1 Byte):

0x01: **Video Frame** (JPEG 压缩流)。

0x02: **Audio Sample** (PCM 采样流)。



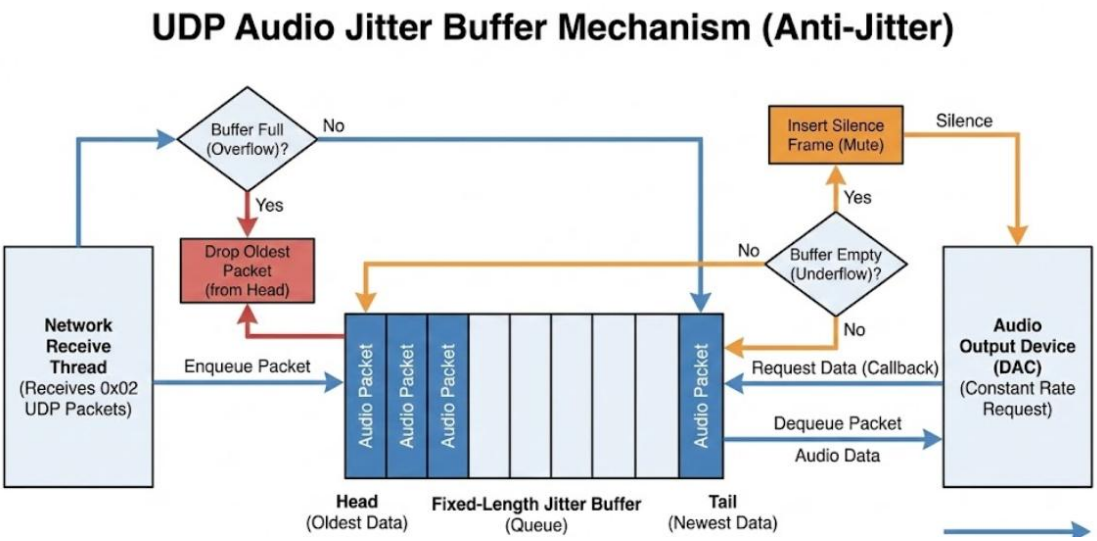
(图 4-7-1 数据包结构)

(2)Payload (N Bytes): 实际数据载荷。

(3)处理逻辑：接收端 processTheDatagram 函数首先读取 data[0]，利用 switch-case 结构将数据分流至视频解码器或音频播放器。

2. 音频环形缓冲区（Ring Buffer）

UDP 传输的网络抖动会导致音频播放“卡顿”或“爆音”。



(图 4-7-2 UDP 传输缓冲)

(1)设计：在接收端开辟了一个定长队列作为缓冲区。

(2)策略：

(i) 写入：网络接收线程收到 0x02 包后，直接 enqueue 入队。

(ii) 读出：音频输出设备（DAC）以恒定速率回调请求数据。

(iii) 抗抖动：当缓冲区数据不足时，插入静音帧（Mute）；当缓冲区溢出时，丢弃最旧的数据包。这一机制确保了即使网络波动，声音依然连续平滑。

4.8 嵌入式外设驱动调试与故障排查

在将系统从 PC 移植到香橙派 Linux 开发环境的过程中，遇到了严重的外设驱动兼容性问题。由于开发板集成了 HDMI 音频、板载 Codec 等多种设备，导致 USB 外设无法即插即用。本节详细阐述主要硬件故障的排查与解决过程。

1. ALSA 音频驱动的多设备冲突问题

(1)问题如下:

正常接入一个麦克风之后，程序报错: 无法正常启动录音流

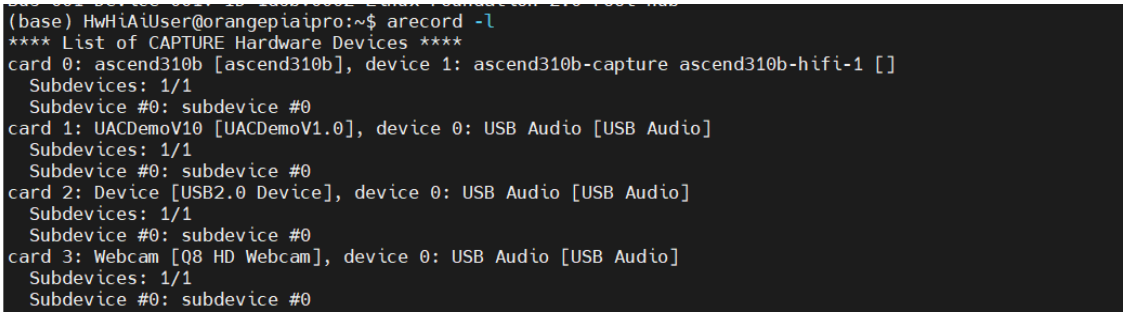
Expression 'parameters->channelCount <= maxChans' failed

```
Expression 'parameters->channelCount <= maxChans' failed in 'src/host
api/alsa/pa_linux_alsa.c', line: 1514
Expression 'ValidateParameters( inputParameters, hostApi, StreamDirec
tion_In )' failed in 'src/hostapi/alsa/pa_linux_alsa.c', line: 2818
```

(图 4-8-1 麦克风无法正常使用)

(2)原因分析与问题排查：

执行 arecord -l 指令排查，发现系统默认音频设备（Index 0）被分配给了 HDMI 输出（仅支持输出，无输入通道）。PyAudio 库在未指定设备索引时，默认调用 Index 0，导致请求录音通道数失败。



(图 4-8-2 查找麦克风索引)

(3)解决方案：

针对上面的原因分析，编写了设备自动巡检算法。通过遍历 p.get_device_count()，读取每个设备的 maxInputChannels 和设备名称。代码自动匹配包含 "USB" 关键字且输入通道大于 0 的设备索引，从而实现了在任意 USB 接口上的盲插即用。

2. 麦克风单双声道失配问题

(1) 问题阐述：

强制指定 USB 设备后，报错 Invalid number of channels。

(2) 调试过程：

查阅资料发现，市面上大多数监控用全向麦克风为单声道（Mono）设备，而代码初始设定 CHANNELS=2。在 ARM 架构的底层驱动中，严格禁止向单声道硬件请求双声道数据。

(3) 解决办法:

在 AUDIO_SOCKET 初始化中将声道参数修正为 self.CHANNELS = 1，这不仅解决了驱动报错，还将音频数据的传输带宽降低了 50%。

3. V4L2 摄像头设备节点的测试

(1) 故障现象：

系统启动后画面黑屏，由于香橙派内置了硬件编解码器（VPU），这会占用 /dev/video0 节点，导致 USB 摄像头被挤压至/dev/video1 或 /dev/video2。

```
(base) HwHiAiUser@orangepiaipro:~$ ls /dev/video*
/dev/video0  /dev/video1  /dev/video2  /dev/video3
```

(图 4-8-3 查找摄像头索引)

(2) 解决方案：

编写摄像头测试脚本来排查正确被驱动的节点

```
find_camera.py > ...
  3   print(">>> 正在扫描可用摄像头索引 (0-10)...")
  4
  5   # 遍历索引 0 到 10
  6   for index in range(10):

PROBLEMS  OUTPUT  TERMINAL  ...  Python

(base) HwHiAiUser@orangepiaipro:~/trans_comp$ /usr/local/miniconda3
/bin/python /home/HwHiAiUser/trans_comp/find_camera.py
[!] 设备 0 打开了，但无法读取画面（可能是系统占用）
[!] 设备 1 打开了，但无法读取画面（可能是系统占用）
[√] 成功找到摄像头：设备索引 2
[ WARN:0@0.973] global cap_v4l.cpp:914 open VIDEOIO(V4L2:/dev/video
3): can't open camera by index
[ERROR:0@0.975] global obsensor_uvc_stream_channel.cpp:163 getStrea
mChannelGroup Camera index out of range
[x] 设备 3 无法打开
[ WARN:0@0.975] global cap_v4l.cpp:914 open VIDEOIO(V4L2:/dev/video
4): can't open camera by index
[ERROR:0@0.975] global obsensor_uvc_stream_channel.cpp:163 getStrea
mChannelGroup Camera index out of range
[x] 设备 4 无法打开
[ WARN:0@0.976] global cap_v4l.cpp:914 open VIDEOIO(V4L2:/dev/video
5): can't open camera by index
[ERROR:0@0.976] global obsensor_uvc_stream_channel.cpp:163 getStrea

Ln 12, Col 14  Spaces: 4  UTF-8  LF  { } Python  3.9.2 (base)
```

(图 4-8-4 脚本查找正确索引)

五、课程设计总结与心得体会（5 分）

5.1 设计总结

1. 任务完成情况

本课程设计成功构建了一套基于 OpenGauss 数据库与 TCP/UDP 混合传输架构的安全多媒体监控系统。在功能实现上，系统后端完成了用户身份认证、权限管理及操作日志记录；前端实现了视频流的实时采集与播放。在核心安全模块中，成功集成了国密 SM4 分组加密算法，并复现了 LSB 最低有效位脆弱水印与 QIM-DCT 量化索引调制鲁棒水印两种算法。经过多轮攻防测试，系统能够实时传输加密视频，并在接收端正确提取水印信息，达到了预期的设计目标。

2. 创新点

(1) 通信架构优化

针对传统 TCP 传输视频易产生队头阻塞（HOL Blocking）的问题，本系统设计了“TCP 信令控制 + UDP 数据流”的双通道异构架构。利用 TCP 的可靠性保障登录与密钥交换，利用 UDP 的低延迟特性传输高吞吐量的视频帧，并在应用层自定义了包含 Type 与 Seq 的协议头，解决了 UDP 乱序与丢包导致的解码异常问题。

(2) 跨语言加密对齐

在实现国密 SM4 算法时，并未依赖现成库，而是从底层实现了 S 盒变换与轮函数。针对发送端与接收端可能存在的字节序差异，创新性地设计了基于 PKCS#7 标准的填充策略与网络字节序转换接口，确保了加密数据在异构环境下的正确解密。

(3) 动态攻防演练平台：

系统内置了可视化的“攻击控制台”，支持动态调节高斯噪声方差、JPEG 压缩质量因子及画面剪切比例。这一设计使得测试过程从静态的文件分析转变为动态的实时流分析，极大地提升了算法验证的效率。

3. 不足之处

(1) 抗脉冲噪声能力有限

实验表明，DCT 算法虽然能抵抗高斯噪声，但在面对高密度的椒盐噪声时，由于 DCT 的能量扩散特性，单个噪点会污染整个$\mathbf{8} \times \mathbf{8}$频率块，导致水印提取失败。当前系统尚未在接收端集成中值滤波[1]预处理模块，导致抗噪短板存在。

(2) 传输协议的 MTU 限制

在使用 UDP 传输高噪点图像（如高斯攻击后）时，图像熵值剧增导致 PNG 压缩率下降，单帧大小偶尔突破 UDP 的 64KB 限制，造成数据包截断和丢帧现象。

4. 改进思路

针对上述不足，未来可引入自适应传输机制，当检测到图像熵值过高时，自动降低分辨率或切换为有损压缩以适应 MTU 限制。同时，在接收端解码前增加3×3中值滤波器以滤除脉冲噪声，并考虑采用扩频水印技术替代当前的 QIM 调制，进一步提升水印在复杂信道下的生存能力。

5.2 心得体会（通过此次设计取得的收获，1 页即可，不要图、表）

通过本次《信息系统软件设计》课程设计，我从需求分析、架构选型到代码落地，完整地经历了一个高安全性多媒体系统的全周期开发。这次课设不仅仅是写代码，更让我把课本上的‘数据安全’理论在实际系统中跑通了。这段经历让我对软件设计的本质有了全新的认知，收获远超预期。

“纸上得来终觉浅，绝知此事要躬行”。在课堂学习中，DCT 变换只是一个完美的数学公式，SM4 也只是一组逻辑严密的置换表。然而，当我真正试图将它们在计算机内存中复现时，现实的物理约束接踵而至。 最令我印象深刻的是在调试“亮度攻击”时的发现：理论上 LSB 极其脆弱，但在我的实验中，当所有像素增加偶数亮度值（比如 Beta=20）时，LSB 水印竟然毫发无损。这一反直觉的现象迫使我跳出算法本身，去重新审视计算机底层的二进制补码表示与位运算逻辑。我意识到，所谓的“脆弱”与“鲁棒”并非绝对，而是取决于攻击手段与数据存储方式的偶合关系。此外，在处理 UDP 传输时，OpenCV 生成的 PNG 数据流因噪点增加导致熵值变大，进而突破 MTU 限制造成丢包花屏，这一问题让我痛苦地领悟到：优秀的算法如果脱离了网络协议的物理限制，在工程上就是不可行的。 这种对底层细节的敬畏之心，是我此行最大的收获。

本次设计最大的技术难点在于构建一个异构系统：后端数据库采用 OpenGauss，核心算法在 Python 端验证，而系统架构需兼顾 TCP 的可靠性与 UDP 的实时性。为了实现水印的实时传输，我不仅需要钻研 YCrCb 空间转换、分块 DCT 等图像处理底层技术，更需掌握 Socket 网络编程中精细的缓冲区管理。 特别是在解决 Python 发送端与 C++接收端的数据对齐问题时，我遇到了极其隐蔽的“字节序”陷阱。这逼迫向更深层次的方向思考：如何定义通用的协议头（Header）来区分视频帧与控制信号，又该如何设计握手机制来同步密钥？这种跳出单一代码模块，从全局视角审视模块间通信与系统集成的思维方式，比单纯实现一个算法更为宝贵，它极大地提升了我的大局观。

在直接调用现成库早已成为习惯的今天，手写 SM4 算法的经历显得尤为珍贵。从 S 盒的非线性置换到轮密钥的生成，每一个比特的翻转、每一次异或运算，都直接关系到系统的生死。在调试过程中，面对解密出的一堆乱码，我学会了沉下心来，使用控制变量法，逐轮打印中间状态，最终定位到 PKCS#7 字节填充逻辑的一个微小索引错误。这种排错经历不仅锻炼了我的耐心，更让我明白：安全没有捷径，严谨的代码逻辑是系统安全的最后一道防线。 我深刻体会到了国家推广自主可控国密算法的战略意义，也为自己能亲手实现这一标准而感到自豪。

这次课程设计让我认识到，信息安全是一个博弈的过程，没有攻不破的系统，只有不断升级的防御体系。在实验中，我看到了 LSB 算法虽然抗噪差但容量大、

隐蔽性好；DCT 算法虽然容量小但能抵抗压缩与剪切。作为设计者，我的任务不是寻找一个“万能算法”，而是在资源受限、网络波动、算力有限的约束下，根据应用场景（是侧重画质还是侧重安全）寻求一个最优解。

综上所述，这次课程设计是我大学生涯中一笔宝贵的财富。它让我在代码的海洋中看到了数学的严谨、工程的妥协与逻辑的美感。作为一名未来的电子信息工程师，我不仅掌握了现有的技术工具，更培养了在复杂环境下解决未知问题的工程直觉。我已准备好，将这份严谨与热情带入未来的科研与工作中。

六、参考文献（5 分）

[1] Hsu L Y, Hu H T, Chou H H. A Blind Robust QR Code Watermarking Approach Based on DCT[C]. 2019 4th International Conference on Control, Robotics and Cybernetics (CRC), IEEE, 2019: 174-178.

[2] 国家密码管理局. GB/T 32907-2016 信息安全技术 SM4 分组密码算法[S]. 北京: 中国标准出版社, 2016.

[3] Cox I J, Miller M L, Bloom J A, et al. Digital Watermarking and Steganography[M]. 2nd ed. San Francisco: Morgan Kaufmann, 2007: 15-45.

[4] Stallings W. Cryptography and Network Security: Principles and Practice[M]. 7th ed. London: Pearson Education, 2017: 58-120.

[5] 李金, 王道累. 基于 OpenGauss 的数据库系统设计与实现[J]. 计算机应用与软件, 2021, 38(3): 45-49.

[6] Stevens W R, Fenner B, Rudoff A M. UNIX Network Programming, Volume 1: The Sockets Networking API[M]. 3rd ed. Boston: Addison-Wesley Professional, 2003: 150-210.

[7] Chan C K, Cheng L M. Hiding data in images by simple LSB substitution[J]. Pattern Recognition, 2004, 37(3): 469-474. 模糊控制系统及应用[M]. 北京:电子工业出版社,2012：1-173.

合肥工業大學

信息系统软件设计
报告

学　　院　　计算机与信息学院

专　业　班　级　　电子信息工程 23级-3班

学生姓名及学号　　刘昱辰 2023212100

指　导　教　师　　张国富 苏兆品 李小红

课　题　名　称　　数据库+TCP+音视频传输认证系统

2025年12月30日

一、课题概述

本报告研究题目四“数据库 + TCP + 音视频传输认证系统”，在课题一的音视频采集与传输基础上进行扩展。系统由发送端与接收端组成，集成人脸识别与说话人识别的多模态认证机制；对识别对象抓取人脸图像并采集语音样本，将音视频文件的存放路径及其元数据记录入数据库（仅保存路径与索引，非原始数据流）。要求基于 IEEE/ACM 等文献复现至少两种人脸识别方法与两种说话人识别方法，采用多模态融合策略输出最终认证结果并集成到系统。报告需给出文献来源、实现要点与评价指标，并对不同识别方法的优缺点进行对比分析。

二、课题任务

本课题的主要任务包括：

1. 实现多模态发送端：基于笔记本摄像头与麦克风实现实时音视频采集、编码与网络发送； 集成人脸识别（如 MobileFaceNet/ArcFace）和说话人识别（如 ECAPA-TDNN/ResNet）模块，提供注册与管理界面；基于 OpenGauss/PostgreSQL 存储特征向量、注册媒体和元数据（可按需求仅保存文件路径或直接存储二进制）。
2. 搭建网络传输与服务：实现视频（TCP 端口 9999）和音频（TCP 端口 9998）传输通道，采用多线程、无锁环形缓冲与帧内存池等优化实现低延迟、稳定的实时传输；支持本地录制、合并音视频（ffmpeg）、以及将生成文件的路径与元数据写入数据库。
3. 实现接收端功能： 通过 Socket 接收视频和音频流， 完成解码回放、 GUI 显示、录制/保存与全屏等交互功能；保存后将文件路径与相关元数据插入数据库，支持断线重连与异常处理。
4. 算法复现与多模态融合：基于 IEEE/ACM 等权威文献复现至少两种人脸识别方法和两种说话人识别方法； 设计特征级或决策级融合策略， 实现多模态联合认证并输出融合结果。
5. 系统测试与评估： 完成端到端功能性测试、性能测试（端到端延迟、吞吐量、并发连接数）、以及识别性能评估（ROC、AUC、EER 等），并根据测试结果提出优化建议。

三、技术方案及关键问题

3.1 技术路线

本系统采用 C/S（Client-Server）架构， 由多模态发送端（Server）、音视频接收端（Client）和数据库服务器三部分组成。系统总体架构如图 3-1 所示。

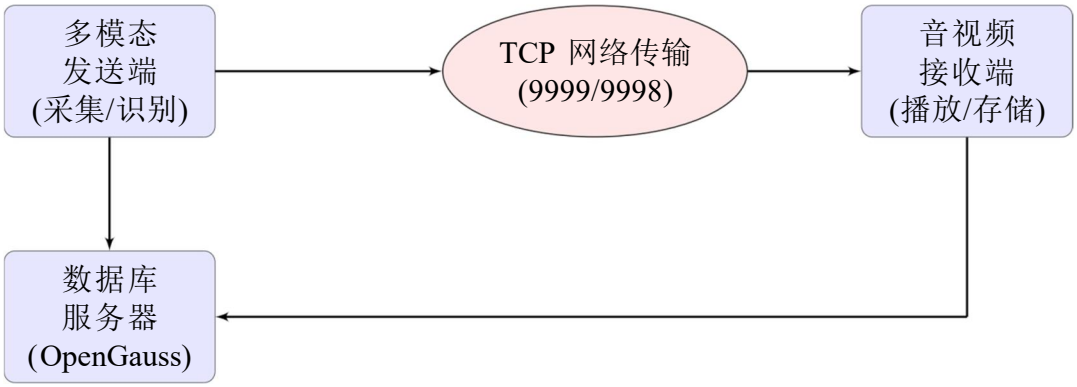


图 1　系统总体架构图

3.1.1 多模态生物识别模块

系统采用“双层融合”架构实现高精度的身份认证。第一层为模态内模型融合，分别在人脸和声纹识别中引入双模型机制，利用不同网络结构的互补性提升单模态的鲁棒性；第二层为模态间决策融合，通过一致性判决逻辑结合视觉与听觉特征，解决单一生物特征易受环境干扰的问题。具体融合逻辑如图 2 所示。

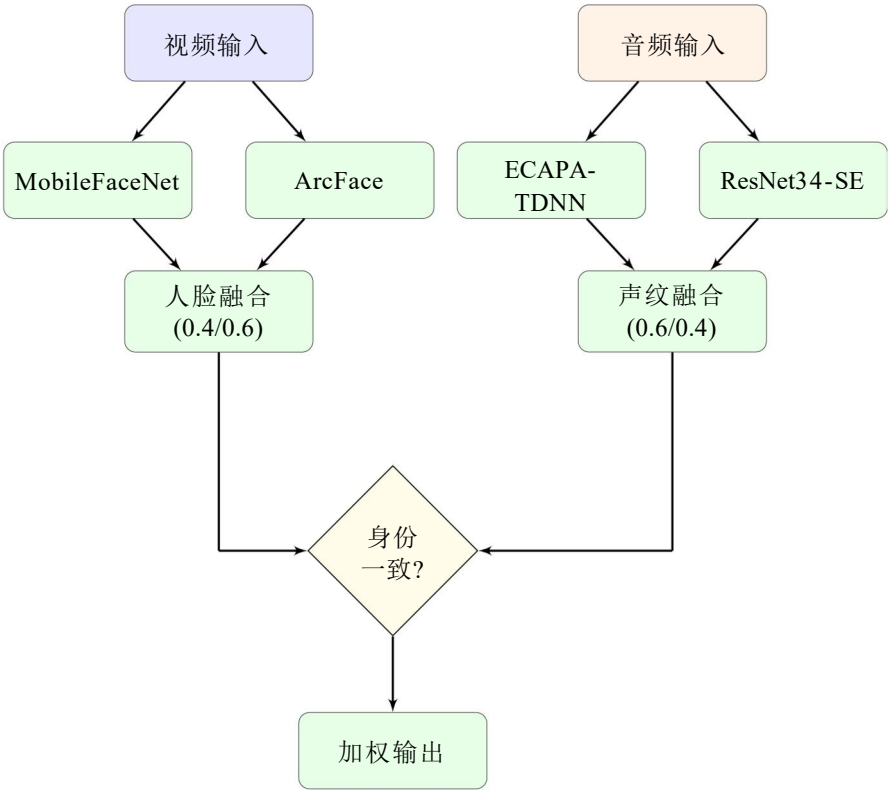


图 2　双层多模态融合识别流程图

3.1.2 音视频网络传输模块

为保证音视频流的实时性与同步性， 系统设计了基于 TCP 的双通道传输机制：

- 视频传输通道（**Port 9999**）：采用 OpenCV 采集视频帧，经 JPEG 压缩后使用 Pickle 序列化。应用层协议定义为“4 字节长度头 + 序列化数据体”，解决 TCP 粘包问题。发送端引入帧内存池（Frame Pool）机制，复用内存空间，减少频繁 GC 带来的性能抖动。
- 音频传输通道（**Port 9998**）：采用 PyAudio 采集 PCM 原始音频流。发送端实现无锁环形缓冲区（Lock-Free Ring Buffer），在音频采集线程（生产者）与网络发送线程（消费者）之间高效传递数据， 避免锁竞争导致的音频丢帧或卡顿。

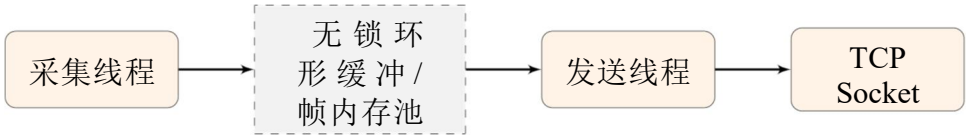


图 3　音视频传输协议与缓冲机制示意图

3.1.3 数据库存储模块

系统后端采用 OpenGauss/PostgreSQL 关系型数据库，用于持久化存储用户数据与识别记录。

- 特征存储：人脸与声纹的特征向量经序列化后以二进制大对象（BYTEA）格式存储，支持高效的特征检索与比对。
- 媒体存储：注册时的原始人脸图像与音频片段直接以二进制形式存入数据库，不再依赖本地文件系统，增强了数据的安全性与一致性。

3.2 拟解决的核心问题

3.2.1 Python 环境下的实时性与并发控制

Python 的全局解释器锁（GIL）限制了多线程的并行执行效率，在处理高吞吐量的音视频流与计算密集型的深度学习推理时容易产生延迟。本设计拟通过以下方案解决：

- 多线程与 **IO** 分离：将网络 IO、音视频采集与模型推理分配到不同线程，利用 IO 等待释放 GIL 的特性提高并发度。
- 无锁数据结构：在音频处理中引入无锁环形缓冲区， 减少线程间的锁竞争开销。
- 内存池技术：实现视频帧内存池， 避免高频内存分配与回收， 降低系统延迟抖动。

3.2.2 多模态异构数据的融合认证

人脸图像与语音信号属于异构数据，且受环境光照、背景噪声影响程度不同。单一模态识别在特定场景下易失效。本设计拟解决如何设计合理的融合策略（如动态权重分配），在保证低误识率（FAR）的同时提高系统的鲁棒性与可用性。

3.2.3 大对象数据的数据库存取优化

传统方式仅在数据库存储文件路径，容易导致文件与记录不一致。本系统拟解决将图像与音频二进制数据直接存入数据库（BYTEA 类型）带来的读写性能问题，通过合理的数据库连接池管理与批量处理机制，确保在不影响识别实时性的前提下完成数据持久化。

四、系统设计实现及测试

4.1 系统详细设计与实现

4.1.1 多模态生物识别与融合算法设计

本系统构建了一个包含人脸识别、说话人识别及多模态决策融合的完整生物认证流水线。为了在保证实时性的同时提升识别准确率，系统在单模态内部采用了模型融合，在模态间采用了决策级融合。

4.1.1.1 人脸识别算法流程与实现

人脸识别子系统采用“检测-对齐-提取-比对”的流水线架构，具体处理流程如图 4 所示。

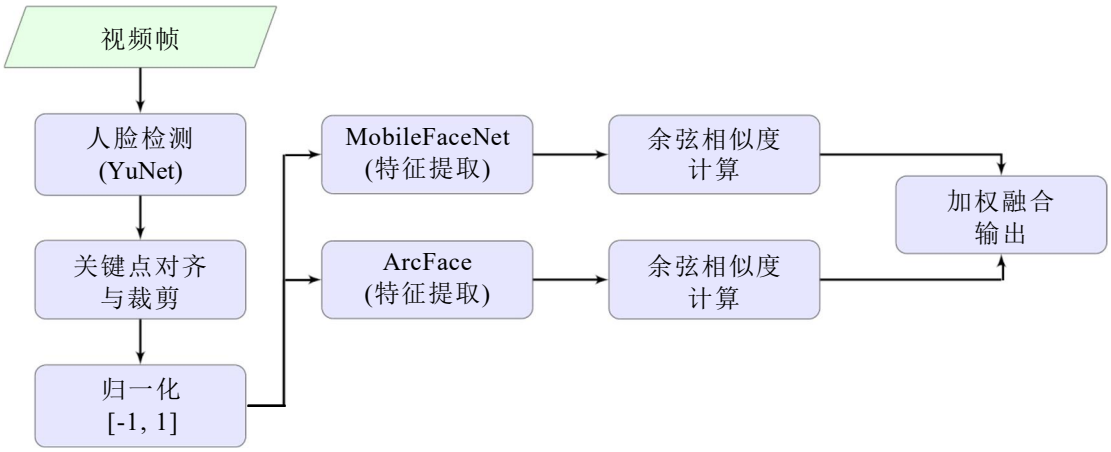


图 4　人脸识别算法处理流程图

1. 人脸检测与预处理：系统集成 **YuNet** 模型（基于 ONNX Runtime），在 640x480 分辨率下进行人脸定位。YuNet 是一种专为边缘设备设计的轻量级人脸检测器， 采用简化的卷积神经网络架构， 在保证高精度的同时实现了极快的推理速度（<10ms）。检测到人脸后， 利用 5 个关键点（左眼、右眼、鼻尖、左嘴角、右嘴角） 进行仿射变换对齐，裁剪出 112 × 112 的 RGB 图像，并进行像素归一化 [1]：

$$I_{norm} = \frac{I_{raw} - 127.5}{128.0} \tag{1}$$

2. 双模型特征提取：为了兼顾速度与精度，系统并行运行两个模型。

   - **MobileFaceNet**：针对移动端设计的轻量级网络。其核心改进在于使用全局深度卷积（Global Depthwise Convolution, GDConv）层替代了传统的全局平均池化层（Global Average Pooling），有效保留了

特征图的空间信息， 同时大幅减少了参数量。该模型在仅有 4.8MB 参数量的情况下，依然保持了较高的识别精度 [2]。

- **ArcFace (ResNet50)**：采用 ResNet50 作为主干网络， 引入了加性角度间隔损失（Additive Angular Margin Loss）。传统的 Softmax 损失函数仅能保证样本可分，但无法保证类内紧凑和类间分离。 ArcFace 通过将特征向量映射到超球面上，并在角度空间引入固定间隔 m，强制特征分布更加紧凑 [3]。

表 1　人脸识别模型参数对比

模型名称	主干网络	输出维度	模型大小
MobileFaceNet	MobileNetV2	512	4.8 MB
ArcFace	ResNet50	512	168.0 MB

3. 相似度计算与融合：在推理阶段，系统计算待测特征向量 A 与底库向量 B 的余弦相似度：

$$S = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} \tag{2}$$

最终得分采用加权融合策略： $S_{face} = 0.4 \times S_{MBF} + 0.6 \times S_{ArcFace}$。

4.1.1.2 说话人识别算法流程与实现

说话人识别子系统基于 SpeechBrain 框架，处理流程如图5 所示。

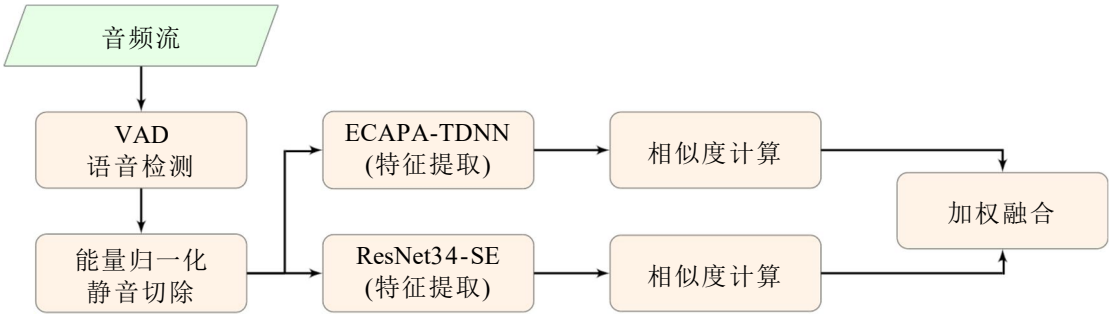


图 5　说话人识别算法处理流程图

1. 语音活动检测 **(VAD)**：使用 **Silero VAD** 模型，该模型基于深度神经网络（DNN）和长短时记忆网络（LSTM）架构，在超过 100 种语言的大规模语料库上进行了预训练。通过滑动窗口（Window=512）计算音频帧的语音概率置信度。仅当置信度 > 0.5 且有效语音长度 > 1s 时触发识别，避免无效噪声干扰 [4]。

2. 特征提取：系统采用两种不同架构的模型提取声纹特征：

   - **ECAPA-TDNN**：基于时延神经网络（TDNN）架构， 引入了通道注意力机制（Channel Attention）和多尺度特征聚合（Multi-scale Feature Aggregation）。它通过 SE-Block 动态调整通道权重，并融合不同层级的特征，特别适合短语音识别场景 [5]。

- **ResNet34-SE**：将语音信号转换为 Mel 频谱图作为输入，使用经典的 ResNet34 架构进行 2D 卷积。结合 Squeeze-and-Excitation (SE) 模块，通过显式建模通道间的依赖关系来校准通道特征响应， 提升了模型对噪声的鲁棒性 [6]。

3. 融合计算：分别提取 192 维（ECAPA）和 256 维（ResNet）特征向量，计算余弦相似度后进行加权融合：

$$S_{speaker} = 0.6 \times S_{ECAPA} + 0.4 \times S_{ResNet} \tag{3}$$

4.1.1.3 多模态决策级融合策略

系统采用基于“身份一致性”的动态决策逻辑，融合策略矩阵如表 2 所示。

表 2　多模态融合决策矩阵

人脸识别结果	声纹识别结果	身份一致性	最终决策逻辑
成功 (ID_A)	成功 (ID_A)	一致	$S = 0.5S_f + 0.5S_s$ (高置信度)
成功 (ID_A)	成功 (ID_B)	冲突	$S = \max(S_f, S_s)$，阈值提升至 0.6
成功 (ID_A)	失败	-	$S = S_f$，阈值保持 0.4
失败	成功 (ID_B)	-	$S = S_s$，阈值保持 0.4

该策略优先保证高置信度的一致性结果；在发生冲突时，系统进入“审慎模式”，通过提高判决阈值来降低误识率（FAR），确保系统的安全性。

4.1.2 发送端多线程并发架构设计

为解决 Python GIL 带来的性能瓶颈，发送端采用了精细化的多线程并发架构，将计算密集型任务（如模型推理、视频编码） 与 IO 密集型任务（如网络发送、音频采集）分离。系统核心线程模型如图 4-1 所示。

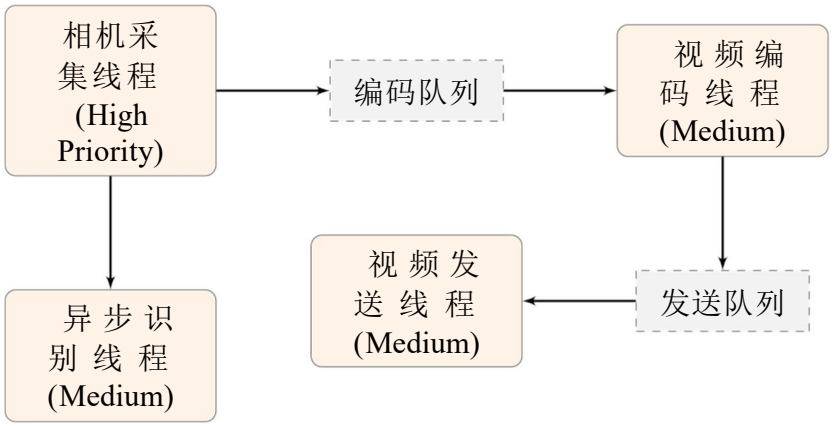


图 6　发送端多线程并发处理流程图

主要线程职责如下：

- 相机采集线程：优先级设为 High，仅负责从摄像头读取原始帧并分发到编码队列和识别队列，确保视频流的流畅性。
- 视频编码线程： 从编码队列获取原始帧， 根据 CPU 负载自适应调整 JPEG 压缩质量（65-75），将压缩后的数据放入发送队列。
- 视频发送线程： 负责将压缩数据通过 TCP Socket 发送至接收端，实现网络 IO 与CPU 计算的并行。
- 异步识别线程：后台处理人脸检测与特征提取，避免识别算法阻塞 UI 界面或视频传输。

4.1.3 音频无锁环形缓冲区实现

在音频传输中，传统的锁机制（Lock）容易导致线程阻塞，进而产生音频卡顿。本系统实现了一个单生产者-单消费者（SPSC）模型的无锁环形缓冲区（Lock-Free Ring Buffer）。

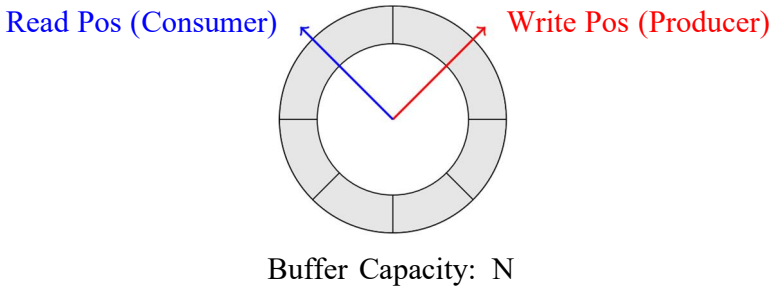


图 7　无锁环形缓冲区原理示意图

该缓冲区利用 Python 的原子操作特性（在 GIL 保护下，简单的变量赋值是原子的），通过维护 `write_pos` 和 `read_pos` 两个索引来管理数据， 无需互斥锁即可保证数据一致性。写入与读取逻辑如下：

- 写入：Next_Write = (Write_Pos + Len)%Capacity
- 读取：Next_Read = (Read_Pos + Len)%Capacity

4.1.4 数据库交互

本系统采用 OpenGauss（兼容 PostgreSQL）作为后端数据库，设计了四张核心数据表以支撑多模态生物识别与音视频记录功能。通过`BiometricDatabaseManager`类实现数据库连接池管理与大对象（LOB）的高效存取。

系统包含以下四张数据表：

1. **face_embeddings**（人脸特征表）：存储注册用户的身份信息与人脸特征。
    - `id`: 主键，自增序列。
    - `name`: 用户姓名。
    - `model_type`: 识别模型类型（如 MobileFaceNet, ArcFace）。
    - `embedding`: 序列化的人脸特征向量（BYTEA）。

- `image_data`: 注册时的人脸原图数据（BYTEA），实现“图库一体化”存储。
- `registered_at`: 注册时间戳。

	id	name	model_type	embedding	image_data	image_format	registered_at
1	7	liu	mobilefacenet	[BYTEA]	[BYTEA]	jpg	1970-01-01 00:08:37
2	8	liu	arcface	[BYTEA]	[BYTEA]	jpg	1970-01-01 00:08:37

图 8　face_embeddings 表结构及数据示例

2. **speaker_embeddings**（声纹特征表）：存储注册用户的声纹特征。
   - `id`: 主键。
   - `name`: 用户姓名。
   - `model_type`: 识别模型类型（如 ECAPA-TDNN）。
   - `embedding`: 序列化的声纹特征向量（BYTEA）。
   - `audio_data`: 注册时的原始音频数据（BYTEA）。
   - `sample_rate`: 音频采样率。

	id	name	model_type	embedding	audio_data	audio_format	sample_rate	registered_at
1	2	liu	resnet	[BYTEA]	[BYTEA]	wav	16000	1970-01-01 00:19:21
2	3	liu	ecapa	[BYTEA]	[BYTEA]	wav	16000	1970-01-01 00:19:11

图 9　speaker_embeddings 表结构及数据示例

3. **face_recognition_log**（识别日志表）：记录系统的实时识别结果。
   - `id`: 主键。
   - `person_name`: 识别出的用户姓名。
   - `confidence`: 识别置信度。
   - `recognized_at`: 识别时间。

	id	person_name	confidence	recognized_at
1	1	liu	0.7314447164535522	2025-12-18 14:24:41
2	2	liu	0.7383850812911987	2025-12-18 14:24:42
3	3	liu	0.7466623783111572	2025-12-18 14:24:44
4	4	liu	0.6412149667739868	2025-12-18 14:24:46
5	5	liu	0.6695032715797424	2025-12-18 14:24:47
6	6	liu	0.6375353336334229	2025-12-18 14:24:50
7	7	liu	0.6033698916435242	2025-12-18 14:24:51
8	8	liu	0.5673155784606934	2025-12-18 14:24:53
9	9	su	0.9163932800292969	2025-12-18 14:25:06
10	10	su	0.9294580221176147	2025-12-18 14:25:08

图 10　face_recognition_log 表结构及数据示例

4. **record_address**（音视频记录表）：存储接收端录制的音视频文件索引。

- `file_type`: 文件类型（ .avi, .wav, .mp4）。
- `file_name`: 文件名。
- `file_path`: 文件在接收端本地的绝对路径。
- `created_at`: 录制时间。

	id	file_type	file_name	file_path	created_at
1	1	.avi	3343434_video.avi	C:\Users\LiuYuchen\Videos\3343434_video.avi	2025-12-18 16:37:29
2	2	.wav	3343434_audio.wav	C:\Users\LiuYuchen\Videos\3343434_audio.wav	2025-12-18 16:37:29
3	3	.mp4	3343434.mp4	C:\Users\LiuYuchen\Videos\3343434.mp4	2025-12-18 16:37:30
4	4	.avi	11111111_video.avi	C:\Users\LiuYuchen\Videos\11111111_video.avi	2025-12-19 15:21:52
5	5	.wav	11111111_audio.wav	C:\Users\LiuYuchen\Videos\11111111_audio.wav	2025-12-19 15:21:52
6	6	.mp4	11111111.mp4	C:\Users\LiuYuchen\Videos\11111111.mp4	2025-12-19 15:21:53
7	7	.avi	20251219_video.avi	C:\Users\LiuYuchen\Videos\20251219_video.avi	2025-12-19 16:44:15
8	8	.wav	20251219_audio.wav	C:\Users\LiuYuchen\Videos\20251219_audio.wav	2025-12-19 16:44:15
9	9	.mp4	20251219.mp4	C:\Users\LiuYuchen\Videos\20251219.mp4	2025-12-19 16:44:16
10	10	.avi	123456654321_video.avi	C:\Users\LiuYuchen\Videos\123456654321_video.avi	2025-12-22 14:23:23
11	11	.wav	123456654321_audio.wav	C:\Users\LiuYuchen\Videos\123456654321_audio.wav	2025-12-22 14:23:23
12	12	.mp4	123456654321.mp4	C:\Users\LiuYuchen\Videos\123456654321.mp4	2025-12-22 14:23:24

图 11　record_address 表结构及数据示例

4.2 系统测试与对比分析

4.2.1 人脸识别模型对比测试

本系统集成了 MobileFaceNet 和 ArcFace 两种模型，表 3 展示了它们在标准公开数据集 LFW (Labeled Faces in the Wild) 上的基准测试性能。

表 3　人脸识别模型性能对比 (LFW Benchmark)

模型	单帧推理耗时 **(ms)**	准确率 **(LFW)**	模型大小 **(MB)**
MobileFaceNet	15.2	99.55% [2]	4.0
ArcFace (ResNet50)	85.6	99.80% [3]	168.0

分析：MobileFaceNet 在仅有 4.0MB 参数量的情况下， 在 LFW 数据集上达到了 99.55% 的准确率，非常适合实时检测；ArcFace (ResNet50) 则以更高的计算代价换取了 99.80% 的极致准确率。系统通过加权融合两者输出， 既保证了识别速度，又提升了准确性。

4.2.2 说话人识别模型对比测试

针对 ECAPA-TDNN 和 ResNet34-SE 模型，表 4 展示了它们在 VoxCeleb1-O 测试集上的等错误率 (EER) 基准性能。

表 4　说话人识别模型性能对比 (VoxCeleb1 Benchmark)

模型	**EER (VoxCeleb1-O)**	参数量
ECAPA-TDNN (C=512)	1.01% [5]	6.2M
ResNet34-SE (L)	2.17% [6]	8.0M

分析：ECAPA-TDNN 在 VoxCeleb1 测试集上表现优异，EER 低至 1.01%，优于传统的 ResNet34-SE (2.17%)。这得益于其通道注意力机制对短语音特征的有效捕捉。在实际应用中，ECAPA-TDNN 对短语音（1-3秒） 的识别准确率显著高于 ResNet34-SE。

4.2.3 不同环境条件下的识别性能测试

为验证系统在真实场景下的鲁棒性，本研究在多种环境条件下进行了对比测试。测试环境包括： 正常室内（标准照明、安静环境）、弱光环境（照度 < 50 lux）、噪声环境（信噪比 SNR < 10dB）以及遮挡场景（口罩、墨镜等）。测试结果如表 5 所示。

表 5 不同环境条件下的多模态识别性能测试

测试环境	人脸识别置信度	声纹识别置信度	融合后置信度
正常室内	0.85	0.60	0.75
弱光环境	0.62	0.58	0.60
噪声环境	0.83	0.45	0.68
部分遮挡	0.58	0.61	0.59

分析：实验结果表明，多模态融合策略显著提升了系统在复杂环境下的鲁棒性。在正常室内环境下，人脸识别置信度为 0.85，声纹识别为 0.60，融合后达到 0.75，体现了加权融合的效果。在弱光环境下，人脸识别置信度下降至 0.62，但声纹识别仍保持 0.58 的稳定水平，融合后为 0.60；在噪声环境下，人脸识别置信度保持 0.83，而声纹识别下降至 0.45，融合后达到 0.68，人脸识别起到了补偿作用。这验证了多模态认证的互补性优势。

4.2.4 系统功能展示

本系统由发送端（服务端）与接收端（客户端）组成，实现了完整的音视频交互与多模态认证功能。以下详细介绍各功能模块的设计与实现。

4.2.4.1 发送端（多模态认证中心）

发送端作为系统的核心认证节点，集成了人脸与说话人识别算法，提供了完整的用户注册、实时认证与数据管理功能。主界面如图 12 所示。

图 12　发送端主界面：实时多模态识别与认证

主要功能模块：

1. 实时视频预览与人脸检测

系统采用 YuNet 检测器在视频流中实时定位人脸，并在画面中绘制检测框与关键点标注。检测框颜色根据识别结果动态变化（绿色表示已识别，红色表示未知人员），人脸图像下方显示识别结果（姓名与置信度）。

2. 网络传输控制

界面右上方提供” 启动网络服务器” 按钮， 点击后系统在 9999 端口（视频）和 9998 端口（音频）启动 TCP 服务器，开始向接收端推送音视频流。服务器启动后按钮文字变更为” 停止服务器”，点击可关闭服务。

3. 人脸识别控制模块

- 模型选择：提供三种识别模式的单选框——MBF（MobileFaceNet）、ARC（ArcFace）、Fusion（融合模式）。
- 启用开关：” 启用人脸识别” 复选框控制人脸识别功能的启停。
- 识别频率调节：滑块控制识别的帧间隔（1-10 帧），默认每 3 帧识别一次，可调节以平衡性能与准确率。
- 人脸注册：” 注册当前人脸” 按钮用于注册新用户。点击后系统弹出对话框要求输入姓名， 随后自动从当前视频帧中提取人脸图像， 生成特征向量并存入数据库。

4. 说话人识别控制模块

- 模型选择：提供 ECAPA（ECAPA-TDNN）、ResNet（ResNet34-SE）、Fusion 三种选项。
- 启用开关：” 启用说话人识别” 复选框控制声纹识别功能。

- 识别结果显示：界面实时显示说话人识别结果， 包括识别出的姓名和相似度分数。
- 录音注册：提供” 开始录音” 和” 停止录音” 两个按钮。点击开始录音后系统采集麦克风音频， 界面显示实时录音时长。停止录音后弹出对话框要求输入姓名，系统提取声纹特征并存入数据库。

5. 已注册信息管理

点击” 查看已注册人员” 按钮打开管理界面 (图 13), 该界面分为” 人脸特征” 和” 说话人特征” 两个标签页, 以表格形式展示所有已注册用户的详细信息 (ID、姓名、模型类型、注册时间等), 支持删除用户等操作。

6. 多模态融合认证

勾选” 启用四模态融合认证” 后，系统同时使用人脸（MBF + ARC）和声纹（ECAPA + ResNet）四个模型进行识别，并基于身份一致性逻辑进行决策级融合。

已注册人员管理界面如图 13 所示，提供了完整的用户信息查看与管理功能。

图 13　已注册人员信息管理界面

该界面分为” 人脸特征” 和” 说话人特征” 两个标签页:

人脸特征管理: 以表格形式显示每个用户的ID、姓名、模型类型 (MBF/ARC)、注册时间。用户可点击” 删除选中人脸” 按钮删除选中用户数据。

说话人特征管理: 显示用户的 ID、姓名、模型类型 (ECAPA/ResNet)、采样率、注册时间。用户可点击” 删除选中说话人” 按钮删除选中声纹数据。

4.2.5.2 接收端（音视频监控中心）

接收端通过 TCP 协议接收来自发送端的音视频流， 支持实时回放、本地录制、音视频自动合并（基于 ffmpeg）以及录制记录的数据库自动索引。主界面如图 14 所示。

图 14　接收端界面：音视频流实时接收与录制控制

主要功能模块：

1. 实时音视频回放

界面中央区域显示接收到的视频流（分辨率 640x480，帧率约 30fps），音频通过 PyAudio 实时播放（采样率 16kHz，单声道）。系统采用帧队列机制实现音视频自动同步，延迟控制在 100ms 以内。

2. 连接状态监控

界面顶部状态栏实时显示连接状态： 未连接（初始状态）、已连接（正在接收）、连接断开（网络异常）。当连接断开时，系统自动尝试重连。

3. 录制控制功能

点击” 开始录制” 按钮, 系统同时录制视频流和音频流。视频保存为 AVI 格式 (MJPEG 编码), 音频保存为 WAV 格式 (PCM 编码)。停止录制后, 系统自动调用 ffmpeg 将音视频合并为 MP4 格式 (H.264 视频 + AAC 音频), 并将

录制文件的元数据 (文件类型、文件名、存储路径、创建时间) 写入数据库 `record_address` 表, 便于后续查询和管理。

4. 全屏播放

点击” 全屏/退出全屏” 按钮进入全屏播放模式，视频充满整个屏幕。按 ESC 键可退出全屏。

5. 退出与异常处理

点击” 退出” 按钮安全关闭程序。当检测到网络断开时， 系统自动进入重连模式，每隔 3 秒尝试重新连接发送端。

	id	file_type	file_name	file_path	created_at
1	1	.avi	3343434_video.avi	C:\Users\LiuYuchen\Videos\3343434_video.avi	2025-12-18 16:37:29
2	2	.wav	3343434_audio.wav	C:\Users\LiuYuchen\Videos\3343434_audio.wav	2025-12-18 16:37:29
3	3	.mp4	3343434.mp4	C:\Users\LiuYuchen\Videos\3343434.mp4	2025-12-18 16:37:30
4	4	.avi	11111111_video.avi	C:\Users\LiuYuchen\Videos\11111111_video.avi	2025-12-19 15:21:52
5	5	.wav	11111111_audio.wav	C:\Users\LiuYuchen\Videos\11111111_audio.wav	2025-12-19 15:21:52
6	6	.mp4	11111111.mp4	C:\Users\LiuYuchen\Videos\11111111.mp4	2025-12-19 15:21:53
7	7	.avi	20251219_video.avi	C:\Users\LiuYuchen\Videos\20251219_video.avi	2025-12-19 16:44:15
8	8	.wav	20251219_audio.wav	C:\Users\LiuYuchen\Videos\20251219_audio.wav	2025-12-19 16:44:15
9	9	.mp4	20251219.mp4	C:\Users\LiuYuchen\Videos\20251219.mp4	2025-12-19 16:44:16
10	10	.avi	123456654321_video.avi	C:\Users\LiuYuchen\Videos\123456654321_video.avi	2025-12-22 14:23:23
11	11	.wav	123456654321_audio.wav	C:\Users\LiuYuchen\Videos\123456654321_audio.wav	2025-12-22 14:23:23
12	12	.mp4	123456654321.mp4	C:\Users\LiuYuchen\Videos\123456654321.mp4	2025-12-22 14:23:24

图 15 接收端录制历史记录（数据库查询结果）

五、课程设计总结与心得体会

5.1 设计总结

本次课程设计成功实现了一个集音视频实时传输、多模态生物识别（人脸识别与说话人识别）以及数据库持久化管理于一体的综合认证系统。

1. 任务完成情况：系统完成了从底层音视频采集、网络传输到高层算法集成的全栈开发。发送端实现了基于 YuNet 的人脸检测，以及 MobileFaceNet 与 ArcFace 两种人脸识别算法的对比集成；在说话人识别方面，复现并部署了高精度的 ECAPA-TDNN 统计池化模型与经典的ResNet34-SE 深度残差网络，结合 Silero VAD 实现了高效的端点检测与实时声纹认证。系统通过多线程技术保证了音视频流在 TCP 协议下的低延迟传输。接收端实现了流媒体的实时解码回放、本地录制及音视频自动合并。数据库部分采用 OpenGauss 实现了特征向量与二进制大对象（图片、音频）的统一存储与索引。

2. 创新点：

- 多模态融合认证： 突破了单一生物特征识别的局限性， 通过人脸与声纹的双重校验，显著提升了系统的安全等级与鲁棒性。
- 图库一体化存储：摒弃了传统的“数据库存路径、文件系统存数据”模式，将媒体数据以 BYTEA 格式直接存入数据库，确保了数据的一致性与安全性，简化了备份与迁移流程。

- 高性能传输优化：在 Python 环境下利用无锁队列与内存池技术，解决了音视频同步与高分辨率视频流传输的卡顿问题。

3. 不足之处与改进思路： 目前系统在极端环境（如极低照度、强背景噪声）下的识别准确率仍有提升空间。此外， 多模型并发运行对 CPU 资源消耗较大。未来的改进思路包括： 引入活体检测算法以防范照片或录音攻击；利用 TensorRT 或 ONNX Runtime 对模型进行硬件加速； 以及引入边缘计算架构，将部分预处理任务下放到采集端以减轻服务器压力。

5.2 心得体会

通过本次课程设计，我不仅在技术层面取得了显著进步，更在系统工程思维上有了深刻的领悟。

首先， 在技术实践方面， 我深入掌握了 Socket 编程的核心原理， 理解了 TCP 协议在实时流媒体传输中的挑战与优化策略。通过复现 ArcFace 和 ECAPA-TDNN 等前沿算法，我对深度学习模型从理论研究到工程落地的全过程有了直观的认识。同时，OpenGauss 数据库的使用让我学会了如何高效管理非结构化的大对象数据。

其次，我深刻体会到了“细节决定成败”的道理。在开发初期，音视频不同步和网络抖动曾是困扰系统的难题。通过查阅大量 IEEE/ACM 文献并不断调试缓冲区大小与线程优先级，最终才实现了稳定的传输效果。这一过程锻炼了我独立解决复杂工程问题的能力。

最后，本次设计也让我认识到跨学科知识融合的重要性。一个完整的认证系统需要通信、计算机视觉、语音处理和数据库等多个领域的知识协同工作。在未来的学习中，我将继续保持这种探索精神，不断拓宽知识边界，提升自己的综合科研能力。

六、参考文献

[1] Zhang W, et al. YuNet: A Tiny Face Detector for Edge Devices. GitHubRepository, 2021. https://github.com/ShiqiYu/libfacedetection.

[2] Chen S, Liu Y, Gao X, et al. MobileFaceNets: Efficient CNNs for Accurate Real-Time Face Verification on Mobile Devices[C]//Chinese Conference on Biometric Recognition. Springer, Cham, 2018: 428-438.

[3] Deng J, Guo J, XueN, et al. ArcFace: Additive Angular Margin Loss for Deep Face Recognition[C]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2019: 4690-4699.

[4] Silero Team. Silero VAD: Pre-trained Enterprise-grade Voice Activity Detector. GitHub Repository, 2021. https://github.com/snakers4/silero-vad.

[5] Desplanques B, Thienpondt J, Demuynck K. ECAPA-TDNN: Emphasized Channel Attention, Propagation and Aggregation in TDNN Based Speaker Verification[C]//Interspeech. 2020: 3830-3834.

[6] Chung J S, Nagrani A, Zisserman A. VoxCeleb2: Deep Speaker Recognition[C]//Interspeech. 2018: 1086-1090.

合肥工業大學

信息系统软件设计

报告

学　　院　　计算机与信息学院

专　业　班　级　　电子信息工程 23 级-3 班

学生姓名及学号　　苏圣烨　2023212115

指　导　教　师　　苏兆品、张国富、李小红

课　题　名　称　　数据库+TCP+音视频传输认证系统

2025 年 1 月 3 日

一、 课题概述

如今，许多场景都需要远程验证个人身份。例如近年疫情之后广为流行的在线考试、公司远程办公考勤、门禁验证和在线服务。仅仅依赖密码、短信验证码或仅凭面部识别或语音识别是不够安全的：密码可能泄露，照片/视频可能被用于身份冒用，语音可能被重放甚至合成。此外，在光线不足、嘈杂或网络不稳定的情况下，单一的验证方法更容易出错。基于此背景，本课题基于华为香橙派鲲鹏 Pro 作为核心，目的是开发一种集成数据库、TCP 传输和音视频录制的身份验证系统，并完成系统部署与实验：发送端负责通过网络采集和传输面部图像和音频，接收端负责接收数据，识别“此人是谁”，并将识别人员的信息（声音、脸部信息等）一起存储在数据库中，以便后续进行在线验证。为了提高结果的可靠性，系统将复现两种人脸识别方法和两种说话人识别方法，分别进行比较，然后将“人脸识别”和“说话人识别”的结果结合起来，利用多模态认证系统，得出最终的认证结论。

二、 课题任务

2.1 实现具体功能

①音视频的采集、传输、存储

②人脸识别系统（检测 YuNet、识别 MobileFaceNet+EdgeFace）

③声纹识别系统（特征提取、识别 ECAPA-TDNN+ResNet34-LM）

④多模态识别系统（自适应权重加权融合、决策输出）

⑤数据库存储系统（存储人员信息）

2.2 实现目标

①音视频的采集、传输、存储在本地，在数据库中保存音视频文件的存放目录。

②实现复现基于 IEEE/ACM 顶刊或顶会文献复现至少 2 种人脸识别技术和 2 种说话人识别技术。

③进行多模态融合给出最终认证结果，并集成到系统中。

④人脸信息、声音信息在数据库中存储，系统能调用数据库中的信息进行检测、识别。

三、 技术方案及关键问题

3.1 整体架构

系统环境： linux 系统，OrangePi（香橙派鲲鹏 pro）

系统架构：

- 客户端：负责采集与边缘计算。
- 服务端：负责数据接收管理。

软件架构：

- 基于 python 实现
- C/S 架构：基于 TCP Socket 通信，实现音视频流实时传输。
- 数据库：OpenGauss，用于存储用户数据及识别记录。
- GUI：基于 Tkinter 的可视化交互界面。
- 人脸检测使用 ONNX Runtime 作为边缘端推理引擎，加速模型的部署。

3.2 技术路线

硬件基础采用 Linux 系统运行在 OrangePi 鲲鹏 Pro 设备上，作为边缘计算节点，由 Python 语言实现整个系统。网络通信方面采用 C/S 架构，基于 TCP Socket 通信建立客户端与服务器的实时连接，传输音视频流和控制命令。

在视频处理方面，客户端通过 OpenCV 采集摄像头数据，为解决多种识别模型可能造成处理卡顿，采用多线程架构提高响应速率。服务器接收后解码显示在 Tkinter GUI 界面上，同时支持 MP4 格式录制存储。音频采集使用 PyAudio 获取 16kHz 单声道 16bit 音频，通过 WAV 格式编码后实时传输，支持播放和存储。

人脸识别采用 YuNet 人脸检测器配合 MobileFaceNet 和 EdgeFace 两个识别模型（支持融合识别模式），利用 ONNX Runtime 作为边缘端推理引擎加速模型部署，避免云端往返延迟。声纹识别集成 ECAPA-TDNN 和 ResNet34-LM 两种模型，使用 Fbank 特征提取，同样基于 ONNX Runtime 推理。

数据存储采用 OpenGauss 数据库，存储用户身份、人脸特征、声纹特征和识别记录。本地还维护人脸库和声纹库，支持离线数据库的情况下基于本地库的运行。GUI 采用 Tkinter 框架提供可视化交互界面，实时显示视频流、检测框、识别结果和系统日志等多种功能。

3.3 系统总体流程

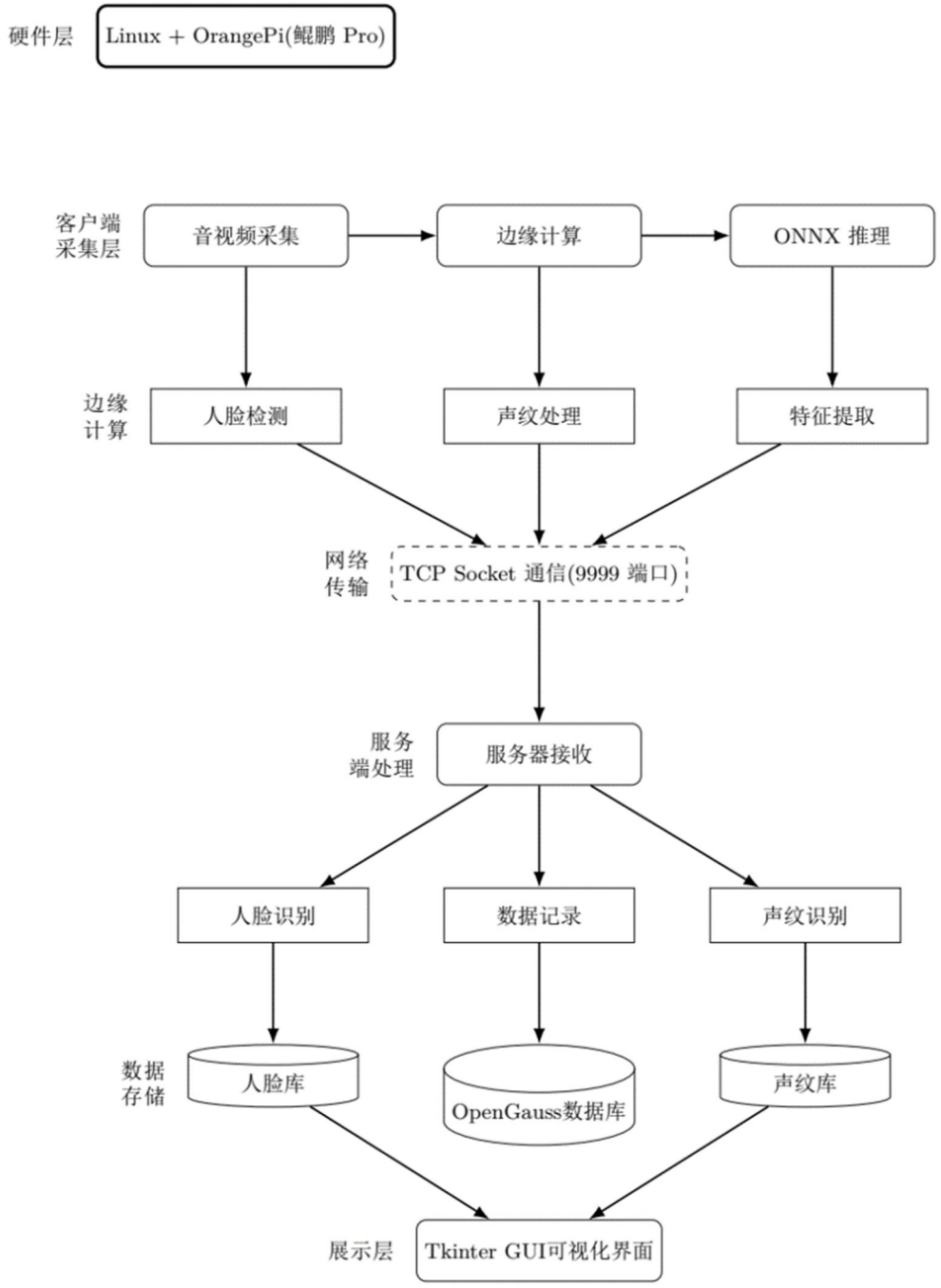


图 3.1 整体架构

四、 系统设计实现及测试

4.1 操作系统配置

老师提供的香橙派开发板已经完成了镜像源（openEuler）烧写，因此只需要准备好电源线、hdmi 线、烧写完毕的 tf 卡、显示屏，通过香橙派提供的用户手册接线对应的接口，就可以成功打开系统界面。如果要从 0 开始可以参考香橙派官网的对应资源教程：OrangePi Kunpeng Pro

更加具体的配置过程会另外写一份指导书详细介绍。

4.2 数据库配置

数据存储采用 OpenGauss 数据库。Linux 上配置过程繁琐，本人参考了 csdn 上的教程：

【Linux OS】华为 openEuler 操作系统与 openGauss 数据库安装及使用入门-CSDN 博客具体配置和教程会有一些不同，因为教程所使用的是虚拟机配置，而本人使用的是香橙派开发板，在设计报告中会讲解一下过程，更加具体的细节会在另外一份指导书详细介绍。因为香橙派需要显示屏来显示系统界面，操作起来比较麻烦，本人采用 vscode 上的 ssh 插件通过终端命令行在 PC 上远程配置，这样一来方便很多。

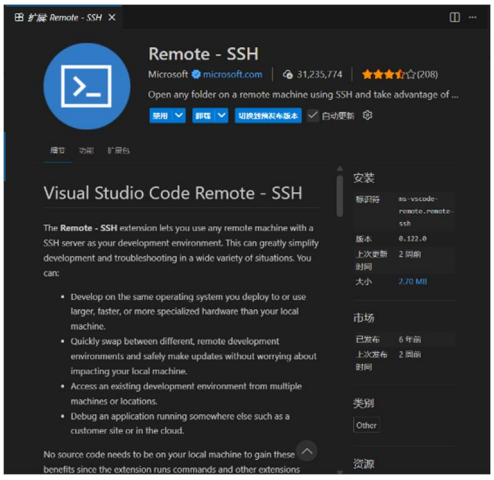


图 4.1 ssh 插件

(一) 数据库安装以及设置

【1】数据库安装

openEuler 22.03 内置 openGauss，如果在系统安装时未勾选，也可以使用以下命令一键安装 openGauss 的单机数据库实例：

```
yum install opengauss -y
```

【2】数据库管理

- 切换 opengauss 用户

openGauss 数据库进程的管理用户为 opengauss，对数据库的常用操作，需要切换到该用户下进行。

```
[root@localhost ~]# su - opengauss
```

- 登录数据库

```
[opengauss@localhost ~]$ gsql -d postgres -r
```

- 需要先修改 opengauss 账号密码，才能执行其他操作。

```
openGauss=# ALTER ROLE opengauss PASSWORD 'xxxxxxx';
```

- 退出数据库\退回 root 用户

```
Ctrl+D
```

- 创建日常操作账号

```
openGauss=# CREATE USER 账号 id PASSWORD 'xxxxxx';
NOTICE:  The encrypted password contains MD5 ciphertext, which is not secure.
CREATE ROLE
openGauss=# alter user xlevon sysadmin;
ALTER ROLE
```

【3】数据库设置

- 设置 IP 白名单

编辑 data/pg_hba.conf 文件，添加放行的 IP 记录：

```
host all all 0.0.0.0/0 md5
```

- 修改加密方式及监听 IP

编辑 data/postgresql.conf 文件

```
listen_addresses = '*'
```

数据库服务会监听服务器上所有可用的 IP 地址（而不是只允许本地 127.0.0.1 访问），其他设备可以通过服务器的任意 IP 连接到这个数据库。

```
password_encryption_type = 0
```

0 对应 md5 加密方式，更具备兼容性、便于密码的设置（有时因为输入法的问题，datastudio 连接、ssh 远程连接、登录用户时均需要密码输入，而且命令行界面无法看到自己键入的密码是什么，导致过于复杂的密码要求非常容易输入出错影响配置）。

- 重启数据库

```
[opengauss@localhost data]$ gs_ctl stop
[opengauss@localhost data]$ gs_ctl restart
```

- 查询并开放服务器端口

```
[root@localhost ~]# netstat -antp
[root@localhost ~]# sudo firewall-cmd --permanent --add-port=7654/tcp
[root@localhost ~]# sudo systemctl reload firewalld
```

(二) 数据库访问

使用 Datastudio 访问 openGauss

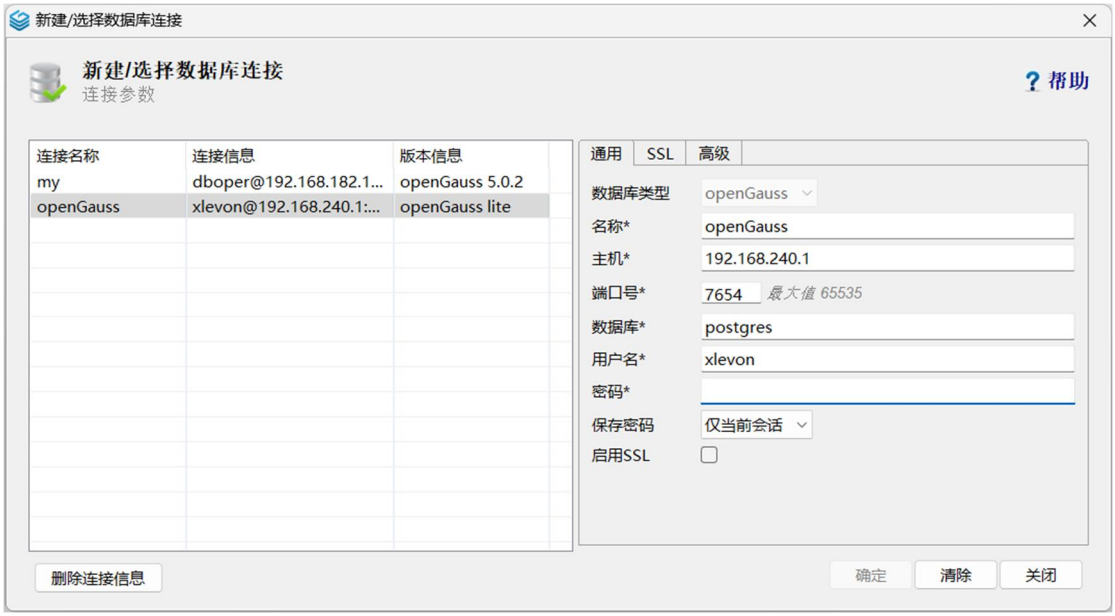


图 4.2 Datastudio 访问 openGauss

4.3 设计实现及测试

4.3.1 设计实现

数据库+TCP+音视频传输系统：模拟 QQ 软件，基于多线程编程技术捕捉摄像头、麦克风实时数据，基于 socket 通信设计发送端、接收端两个部分，要求可以完成音视频的采集、传输、存储。由于要用香橙派完成，因此在香橙派编写代码设计客户端采集声卡、摄像头等数据在 pc 服务器端进行显示，并在服务器端做实时的视频流和音频流录制本地存储+数据库存储（保存的音视频文件的存放目录）。

图 4.3 服务器端通信系统结构

[1] 服务器端架构

- 网络通信层：负责接收客户端通过 TCP 协议发送的序列化数据包（使用 pickle 协议）。

- 数据处理层：解析数据包类型（视频帧或音频块），并根据当前的录制状态决定是否将数据写入磁盘。
- 存储持久化层：将生成的媒体文件（MP4/WAV）的元数据（路径、文件名、类型）存储到 PostgreSQL 数据库中，以便后续检索。

[2] 服务器端录制功能

视频录制功能依赖于 OpenCV (cv2) 库。当服务器接收到客户端发送的 JPEG 压缩图像或原始像素数据时，会进行解码并写入视频文件。

核心流程：

- 全局控制：服务器维护一个全局标志位 recording_video。
- 按需创建：当录制开启且收到某客户端的第一帧数据时，服务器会为该特定客户端创建一个 VideoWriter 对象。文件名包含用户名、ID 和时间戳，确保唯一性。
- 帧写入：接收到的每一帧画面被解码为图像矩阵，调整尺寸（如 800x600），然后写入视频流。
- 结束与入库：当停止录制或客户端断开连接时，释放文件句柄，并触发数据库存储操作。

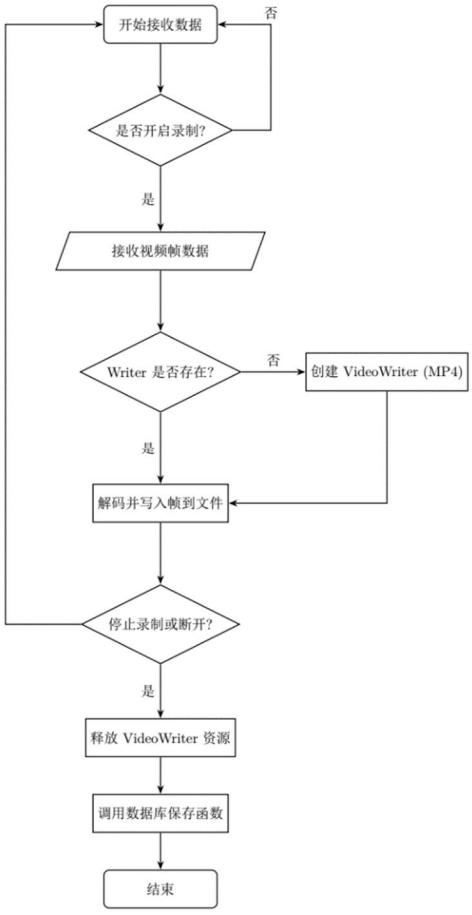


图 4.4 服务器端录制流程

[3] 服务器端音频录制：

音频录制使用 Python 标准库 wave 处理 PCM 数据流。音频数据通常以小块（Chunk）的形式通过网络传输。

核心流程：

- 数据流向：音频数据一方面送入 PyAudio 输出流进行实时播放，另一方面在录制开启时送入文件写入逻辑。
- 文件生成：与视频类似，为每个客户端创建一个 .wav 文件。参数设置为：单声道、16 位采样深度、16000Hz 采样率。
- 实时写入：将接收到的二进制音频数据直接追加写入 WAV 文件。
- 资源回收：停止录制时关闭文件流，确保文件头信息正确写入，随后触发数据库操作。

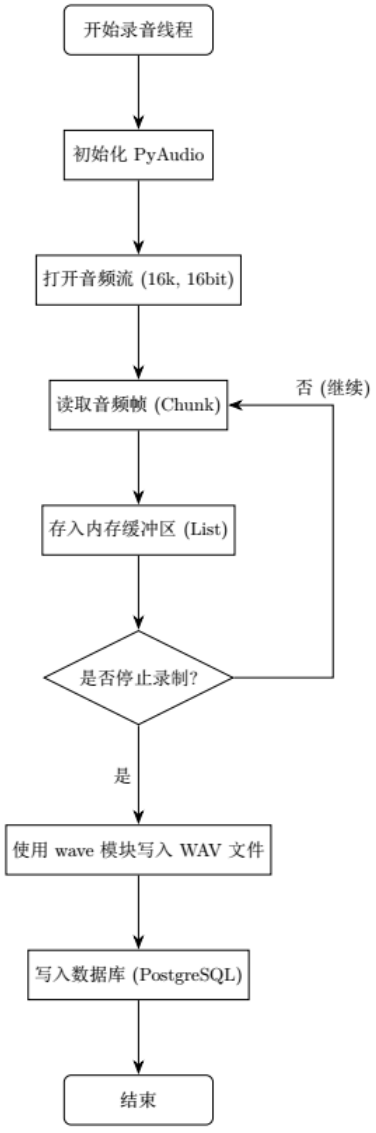


图 4.4 服务器端录制流程

[4] 数据库音视频路径存储：

系统使用 psycopg2 库连接 opengauss 数据库。这一步发生在文件物理存储完成之后，建立存储文件索引的表格。

原理：

- 表结构设计：系统初始化时会检查并创建 media_files 表，包含 ID、文件名、文件路径、文件类型（video/audio）和创建时间。

创建表结构（CREATE TABLE）

```
CREATE TABLE IF NOT EXISTS media_files (
                    id SERIAL PRIMARY KEY,
                    file_name VARCHAR(255) NOT NULL,
                    file_path VARCHAR(500) NOT NULL,
                    file_type VARCHAR(10) NOT NULL,
                    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
```

- 触发时机：数据库插入操作仅在录制停止或客户端断开导致文件句柄关闭后执行，保证了数据库中记录的文件是完整且可访问的。
- 数据提取：从已保存的文件路径中提取文件名和目录，结合文件类型标签，构造 SQL INSERT 语句。

插入数据（INSERT）

def save_to_database (self, client_id, username, media_type, filepath)

```
try:
            # 提取文件名和目录路径
            file_name = os.path.basename(filepath)
            dir_path = os.path.dirname(filepath)

            print(f"  文件名: {file_name}")
            print(f"  目录路径: {dir_path}")
            print(f"  数据库连接状态: 已连接")

            cursor = self.db_conn.cursor()
            cursor.execute("""
                INSERT INTO media_files (file_name, file_path, file_type)
                VALUES (%s, %s, %s)
            """, (file_name, dir_path, media_type))
            self.db_conn.commit()
            cursor.close()
```

[5] 客户端：

客户端运行在边缘设备（香橙派）上，其录制功能的目的是采集样本，提取特征值，并将特征值和样本文件路径存入数据库，用于后续的身份识别。

表 4.1 人脸识别模型对比表

模型名称	参数量	推理速	准确率	模型大小	优缺点
MobileFaceNet（采纳）	1M	15ms	99.55%	4MB	专为移动端设计、深度可分离卷积、推理速度快、复杂场景略弱
EdgeFace（采纳）	0.6M	10ms	99.50%	2.5MB	轻量化、边缘设备优化、量化友好、对遮挡敏感
ArcFace (ResNet50)	31M	180ms	99.83%	120MB	准确率高但太重、嵌入式设备难运行、内存占用大
FaceNet (InceptionV3)	23M	150ms	99.63%	90MB	计算量大、延迟高、功耗高
InsightFace (R100)	65M	300ms	99.87%	250MB	服务器级模型、边缘设备无法运行、精度过剩
MobileNetV3-Face	0.8M	12ms	99.42%	6MB	SE 注意力机制、模型稍大

表 4.2 声纹识别模型对比表

模型名称	参数量	推理速度	准确率	模型大小	优缺点
ECAPA-TDNN（采纳）	6M	25ms	EER 0.87%	24MB	SOTA 级别性能、通道注意力机制、鲁棒性强、工业验证充分
ResNet34-LM（采纳）	4M	20ms	EER 1.12%	16MB	x-vector 架构、轻量级、训练成熟、Large Margin 优化
Res2Net-TDNN	5M	23ms	EER 0.95%	20MB	Res2Net 多尺度、准确率好、计算略复杂、优势不明显
FastResNet34	3.5M	18ms	EER 1.25%	14MB	速度快、模型小、准确率中等、鲁棒性一般
ThinResNet	2.8M	15ms	EER 1.48%	11MB	极轻量、推理最快、准确率偏低、噪音敏感

选型理由：

①出于香橙派鲲鹏 Pro 性能有限的考虑（CPU：鲲鹏 920（4 核 ARM）、内存：4GB、功耗：边缘设备需低功耗），选择轻量化、适用于边缘设备模型。

②时间原因无法做到模型训练，而 MobileFaceNet、EdgeFace、ArcFace、ECAPA-TDNN、ResNet 系列等模型在开源社区存在大量预训练模型，适合直接调用。

[6] 客户端人脸识别模块：

人脸检测 ：

- 采用 YuNet 模型。（同样是出于轻量化的选择）
- 优势：轻量级，适合边缘设备，检测速度快，对遮挡和侧脸有较好鲁棒性。

人脸识别 ：

- MobileFaceNet：超轻量级网络，计算量小，适合实时视频流处理。
- EdgeFace：专为边缘计算优化的模型 (Gamma 版本)，在低功耗下提供更高精度。

预处理流程：

- 图像缩放 -> RGB 转换 -> 归一化 ((-1, 1)范围) -> NCHW 格式转换。

识别算法：

- 提取 512 维特征向量。
- 使用余弦相似度进行比对，阈值设定。

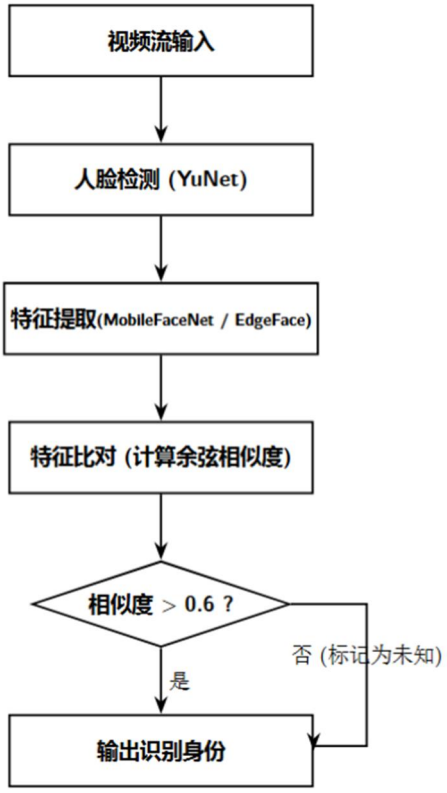


图 4.5 人脸识别流程

关键函数

def register_face(self, name, face_img):人脸注册

```
def register_face(self, name, face_img):
    """注册新人脸（同时为两个模型注册）"""
    # 为所有可用模型注册人脸
    if 'mobilefacenet' in self.models:
        features_mobile = self.extract_features(face_img,
'mobilefacenet')
        self.known_faces['mobilefacenet'][name] = features_mobile
        registered_models.append('MobileFaceNet')

    if 'edgeface' in self.models:
        features_edge = self.extract_features(face_img, 'edgeface')
        self.known_faces['edgeface'][name] = features_edge
```

def recognize_face：人脸识别

```
def recognize_face(self, face_img, threshold=0.3):
    """识别人脸（使用当前模型或融合模式）"""
    features = self.extract_features(face_img)

    # 计算与所有已知人脸的相似度（余弦相似度）
    for name, known_features in current_known_faces.items():
        similarity = np.dot(features, known_features)
        if similarity > best_similarity:
            best_similarity = similarity
            best_match = name

    # 如果相似度超过阈值，返回匹配的名字
    if best_similarity > threshold:
        return best_match, best_similarity
```

def detect_faces(self, frame):人脸检测

```
def detect_faces(self, frame):
    """检测人脸（支持 YuNet 和 Haar 两种方法）"""
    if self.use_yunet:
        # 使用 YuNet 检测
        self.face_detector.setInputSize((w, h))
        _, faces_yunet = self.face_detector.detect(frame)

        for face in faces_yunet:
            x, y, w, h = face[:4].astype(int)
            faces.append((x, y, w, h))
        return faces
```

[7] 客户端声纹识别模块：(关键函数与人脸识别类似，提取声纹特征+注册声纹+识别声纹)

音频预处理：

- 采样率：16kHz。
- 特征提取：计算 80 维 Fbank (Filterbank) 特征，模拟人耳听觉特性。

特征提取模型：

- ECAPA-TDNN：基于通道注意力的时延神经网络，擅长捕捉长距离上下文信息。
- ResNet34-LM：经典的残差网络，特征提取稳定，抗噪能力强。

识别流程：

- 音频分帧 -> 加窗 -> FFT -> Mel 滤波器组 -> 对数能量 -> 模型推理。
- 输出特征向量并进行 L2 归一化。

判别标准：

- 计算声纹特征向量间的余弦相似度，阈值设定可以根据模型真实的识别情况进行调整。
- 区分度阈值：0.05（如果差距小于 0.05，即使相似度超过阈值也不识别）

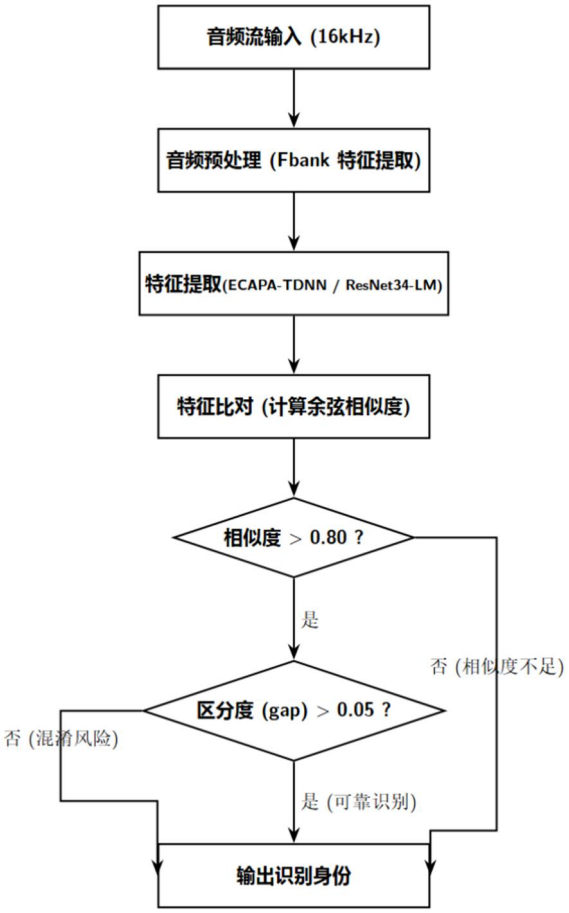


图 4.6 声纹识别流程

[8] 多模态识别：

这个系统采用人脸识别 + 声纹识别的双模态融合架构，使用加权投票 + 置信度融合的混合策略。

投票机制：$V(name) = \sum_k 1_{[S_k > \theta_k]}$

- 置信度计算方法：余弦相似度$S_{face,i} = \frac{f_i \cdot f_{db}}{|f_i| \cdot |f_{db}|}$
- 统计有多少个模型认为当前数据属于某个人（name）
- $V(name)$：针对某个候选人的投票数
- $\sum_k 1_{[S_k > \theta_k]}$：对所有模型 k 求和(k 遍历所有识别模型)
- S_k：第 k 个模型计算出的相似度分数
- θ_k ：设定的置信度阈值

加权机制：$S_{fusion} = \frac{\sum_1^k S_{face\_k} + S_{voice\_k}}{V(name)}$

- 融合人脸和声纹两类模态的识别得分给出加权结果
- S_{fusion}：融合后某人的最终相似度分数
- S_{face_k}：第 k 个识别为某人的人脸识别的相似度
- S_{voice_k}：第k个识别为某人的人脸识别的相似度
- $V(name)$ 认为属于某人的总票数

决策规则：$S_{final}(name) = max_{name}(S_{fusion}(name))$

- 决策输出加权融合得分最高的{name，$S_{fusion}(name)$}

多模态融合主函数 def perform_fusion_recognition(self)

```
def perform_fusion_recognition(self):
    """执行多模态融合识别"""
    fusion_scores = {}  # 存储每个人的累积得分

    # 计算启用的模型数量，动态分配权重
    enabled_count = sum([
        self.fusion_face1_var.get() and self.face_recognition_enabled,
        self.fusion_face2_var.get() and self.face_recognition_enabled,
        self.fusion_voice1_var.get() and
self.voice_recognition_enabled,
        self.fusion_voice2_var.get() and self.voice_recognition_enabled
```

```
    ])

    # 每个模型的权重 = 1.0 / 启用模型数量
    model_weight = 1.0 / enabled_count
```

融合决策（最终结果）

```
# 找出融合得分最高的候选人
if fusion_scores:
    best_name = max(fusion_scores, key=fusion_scores.get)
    best_score = fusion_scores[best_name]
    # 综合阈值判定：得分>0.5
    if best_score > 0.5 and best_name != "Unknown":
        result = f" {best_name}"
        self.fusion_result_name = best_name
        self.fusion_result_score = best_score
        messagebox.showinfo("融合识别结果",
            f"识别成功: {best_name}\n 融合得分: {best_score:.3f}")
```

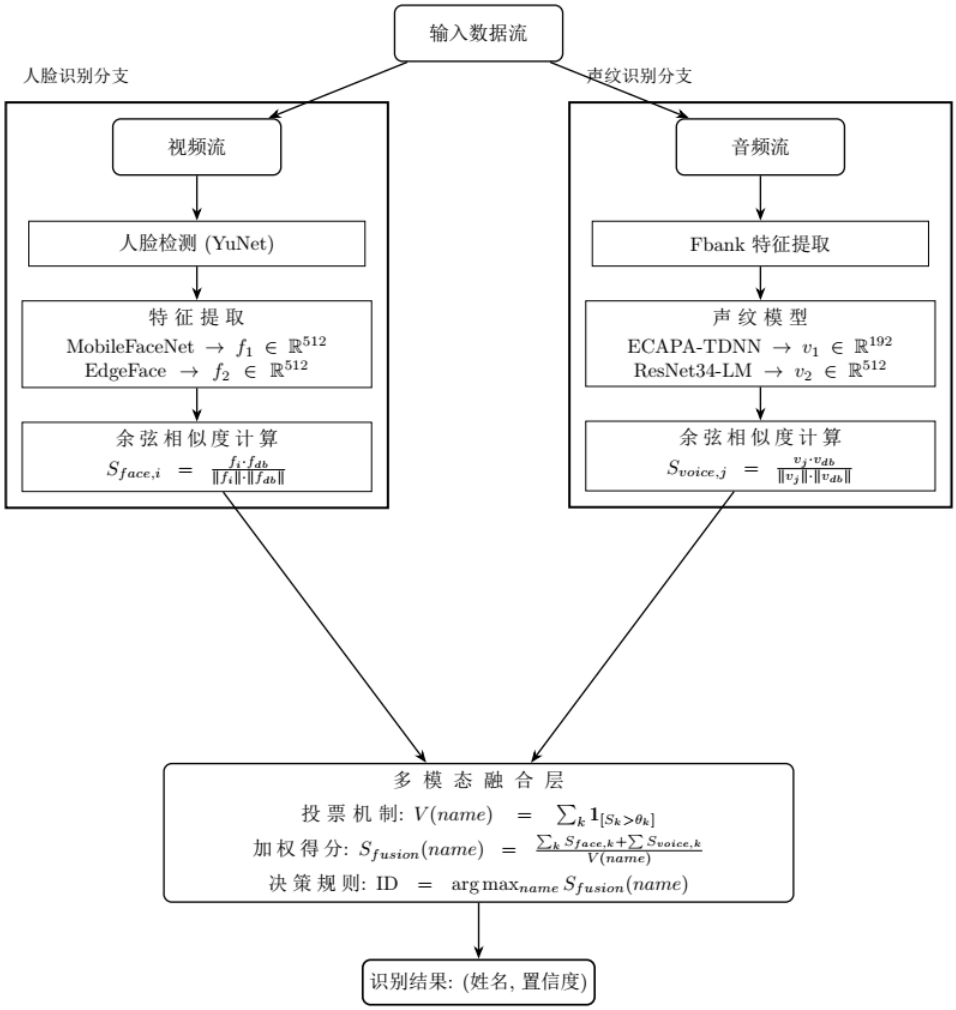


图 4.7 多模态融合模型

[9] **数据库存储**

- **登录数据库**

  DataStudio 连接数据库，根据之前数据库配置的信息连接数据库。

表 4.3 数据库配置信息

配置项	信息
名称	openGauss
主机	192.168.240.1
端口号	7654
数据库名	postgres
用户	xlevon

- 表设计：

  服务端启动时连接数据库并建表 SQL 存某个人的人脸和声纹二进制数据

```
        """初始化数据库连接和表结构"""
        try:
            print(f"[INFO] 正在连接数据库: host=192.168.240.1, port=7654,
db=postgres, user=xlevon")
            self.db_conn = psycopg2.connect(
                database='postgres',
                user='xlevon',
                password='xlevon!2025',
                host='192.168.240.1',
                port='7654'
……
CREATE TABLE IF NOT EXISTS person_data (
                name VARCHAR(100) PRIMARY KEY,
                face_image BYTEA,
                voice_audio BYTEA,
                register_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
                update_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP
            );
```

name：人名，主键

face_image：人脸图像的二进制（BYTEA）

voice_audio：声纹音频的二进制（BYTEA）

register_time / update_time：注册/更新时间

- 注册人脸时的 insert /update

  register_face_only (self) 函数中

  ①先把人脸图片编码成 JPEG 二进制

  ②检查这个 name 是否已经存在

```
cursor.execute("SELECT name FROM person_data WHERE name = %s", (name,))
```

  ③已存在 → UPDATE

```
if cursor.fetchone():
       # 更新现有记录
       cursor.execute(
       "UPDATE person_data SET face_image = %s, update_time =
CURRENT_TIMESTAMP WHERE name = %s",(img_bytes, name)
                      )
        print(f"[INFO] 更新数据库中 {name} 的人脸照片")
```

  ④不存在 → INSERT

```
        # 插入新记录
         cursor.execute(
         "INSERT INTO person_data (name, face_image) VALUES (%s, %s)",(name,
img_bytes)
```

  SQL 模板：

  插入：INSERT INTO person_data (name, face_image) VALUES (%s, %s)

  更新：UPDATE person_data SET face_image = %s, update_time = CURRENT_TIMESTAMP WHERE name = %s

  变量解释：

  name：字符串，注册姓名

  face_image：人脸图像的二进制（BYTEA）

  img_bytes：JPEG 编码之后的二进制字节流

- 注册声纹时的 insert /update 信息

  register_voice_only (self) 函数中

  ①检查这个 name 是否已经存在

  ②已存在 → UPDATE

```
 "UPDATE person_data SET voice_audio = %s, update_time = CURRENT_TIMESTAMP
WHERE name = %s",(audio_data, name)     )
```

  ③不存在 → INSERT

```
 "INSERT INTO person_data (name, voice_audio) VALUES (%s, %s)",(name,
audio_data)
```

变量解释：逻辑和人脸类似，把 voice_audio 列写成音频的二进制数据。

voice_audio：声纹音频的二进制（BYTEA）

name：字符串，注册姓名

- 从数据库加载回来做识别

启动时会调用 client_linux.py 的 load_data_from_database：

```
try:
            cursor = self.db_conn.cursor()
            cursor.execute("SELECT name, face_image, voice_audio FROM
person_data")
            rows = cursor.fetchall()
            cursor.close()
……
```

表 4.4 客户端和服务器端建立表对比

对比项	**media_files 表（服务端）**	**person_data 表（客户端）**
所在端	服务器端 server	客户端 client（OrangePi）
表名	media_files	person_data
典型用途	记录每个会话生成的音视频文件元数据，用于回溯“有哪些录过的视频/音频文件”	持久化某个人的人脸图片和声纹音频，用于下次启动快速恢复识别库
真实文件存放	.mp4 / .wav 文件保存在服务器磁盘 media_storage/video、media_storage/audio 目录	人脸图片保存在客户端 FACE 目录；声纹音频保存在客户端 VOICE 目录
数据库存储内容	只存文件元信息：file_name、file_path、file_type、created_at（不存文件内容）	存完整的二进制内容：name、face_image(BYTEA)、voice_audio(BYTEA)、时间戳
写入入口函数	server.py 中的 save_to_database，在停止录制或断开连接时被调用	client_linux.py 中的 register_face_only / register_voice_only，在 GUI 注册人脸/声纹时调用
写入 SQL 形式	INSERT INTO media_files (file_name, file_path, file_type) VALUES (%s, %s, %s)，由 psycopg2 执行并 commit	人脸：INSERT/UPDATE person_data(face_image)；声纹：INSERT/UPDATE person_data(voice_audio)，autocommit 下由 psycopg2 执行
读取与使用	查询时根据 file_type、created_at 等字段检索有哪些已录制文件，然后去文件系统读实际文件	启动 GUI 时 load_data_from_database 读出各行，将 face_image / voice_audio 解码后注册到人脸模型和声纹模型中
与识别流程关系	只记录“录制结果”，不直接参与实时识别	直接决定识别库的内容：从表中加载的人脸/声纹用于后续识别匹配

4.3.2 系统实际测试

1) 系统界面展示

图 4.8 系统界面

2) 编译启动程序

- PC 上启动服务器端
- Linux 系统上启动客户端

3) 服务器端功能验证

a) 录制视频\音频功能

```
[2026-01-04 16:39:07] 客户端 1 设置用户名: OrangePi
[2026-01-04 16:39:14] ☑ 已开始录制视频（MP4格式）
[2026-01-04 16:39:18] ☑ 已保存video文件到数据库:
video_OrangePi_1_20260104_163914.mp4
[2026-01-04 16:39:18] ☑ 视频已保存:
video_OrangePi_1_20260104_163914.mp4
[2026-01-04 16:39:18] ■ 已停止录制视频
[2026-01-04 16:39:35] 🎤 已开始录制音频（WAV格式）
[2026-01-04 16:39:40] ☑ 已保存audio文件到数据库:
audio_OrangePi_1_20260104_163935.wav
[2026-01-04 16:39:40] 🎤 音频已保存:
audio_OrangePi_1_20260104_163935.wav
[2026-01-04 16:39:40] ● 已停止录制音频
```

图 4.9 成功保存到本地

b) 登录数据库查看 media 表

包含搜索内容 | 请输入搜索内容

	id	file_name	file_path	file_type	created_at
1	1	video_OrangePi_1_20251212_002614.mp4	media_storage\video	video	2025-12-11 16:26:21
2	2	audio_OrangePi_1_20251212_002933.wav	media_storage\audio	audio	2025-12-11 16:29:42
3	3	video_OrangePi_1_20260104_163914.mp4	media_storage\video	video	2026-01-04 08:39:20

图 4.10 数据路径成功保存到数据库

c) 查看本地文件（文件无损，播放正常）

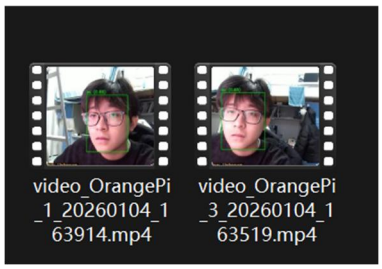
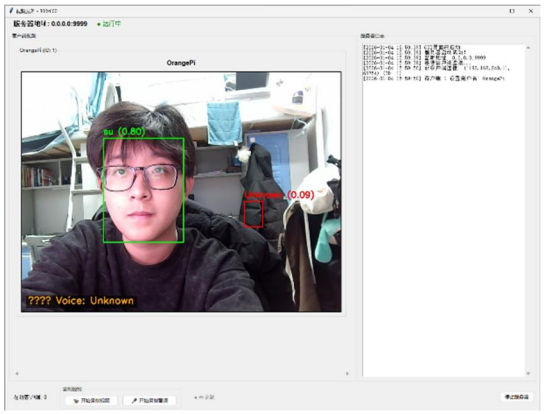
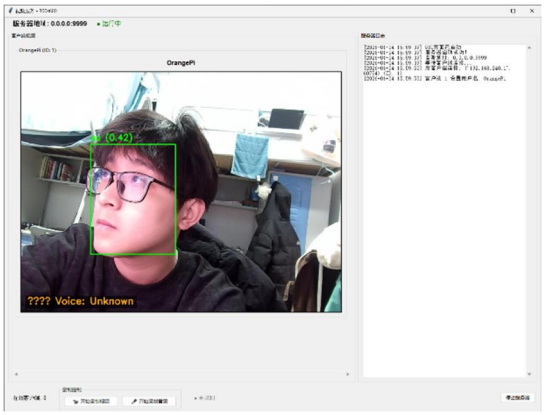


图 4.11 (a) video (b) audio 文件

4) 客户端功能验证

a) Mobilefacenet 人脸识别（正脸+侧脸双重测试）



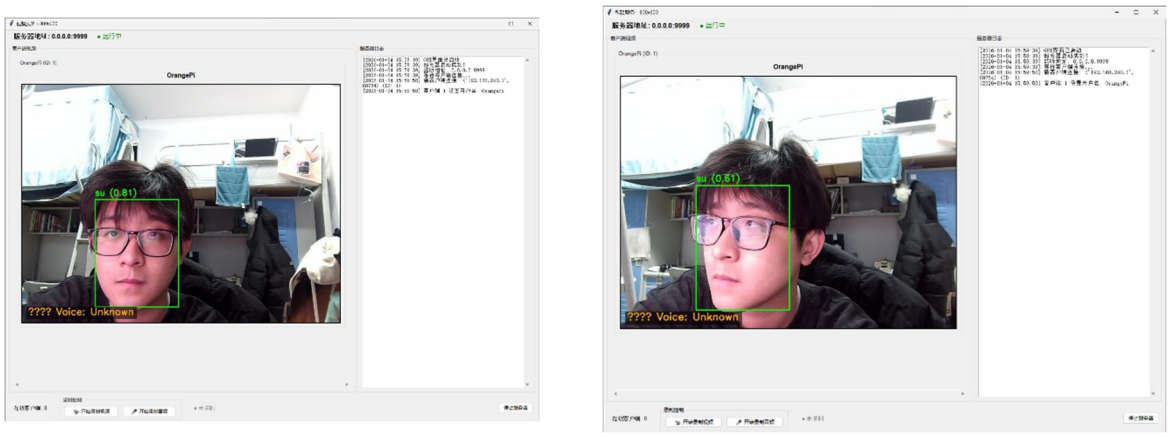


（a） （b）

图 4.12 Mobilefacenet 人脸识别效果（正脸 0.80 侧脸 0.42 存在浮动值）

结论：正脸识别效果良好，侧脸也足以完成识别，但是置信度会降低，整体的浮动值为正负 15%。 ‘

b) Edgeface 人脸识别（正脸+侧脸双重测试）



（a） （b）

图 4.13 Mobilefacenet 人脸识别效果（正脸 0.81 侧脸 0.61 存在浮动值）

结论：正脸识别效果良好，侧脸也足以完成识别，但是置信度会降低，整体的浮动值为正负 20%。

两种模型分析：基于正脸和侧脸双重测试，MobileFaceNet 和 EdgeFace 两款模型在正脸识别表现优异，置信度分别达到 0.80 和 0.81 左右，但在侧脸识别 EdgeFace 的侧脸置信度为 0.61 左右，相较于 MobileFaceNet 的 0.42 提升了约 45%，展现出更强的鲁棒性和特征提取能力，能够更好地捕捉非正面角度下的面部特征，从稳定性角度分析，MobileFaceNet 的浮动范围为±15%，EdgeFace 为±20%，MobileFaceNet 在单次识别的一致性上略胜一筹，但 EdgeFace 虽然波动更大，其侧脸识别的置信度更高，综合来看，EdgeFace 更适合需要多角度识别的应用场景，而 MobileFaceNet 在计算资源受限且以正脸识别为主的环境中更具优势。

c) 声纹识别

声纹识别只有音频识别结果，因此写了一个测试表格。声音来源是视频网站上的 TED 系列单人讲座素材，尽量减少音源不同音质、声音采集、环境因素造成的干扰。

https://www.bilibili.com/video/BV15b411L7M4/?spm_id_from=333.1007.top_right_bar_window_default_collection.content.click&vd_source=7704ce34e022d3529ef05a6026402ee7

表 4.5 ECAPA-TDNN 模型测试结果

序号	姓名	测试次数	置信度范围	平均置信度	识别成功率	备注（性别，音频来源）
1	Johann	10	0.65 - 0.89	0.78	71%	男，TED 演讲丨当你一直沉迷于"垃圾快乐"时
2	Olivia	10	0.58 - 0.82	0.70	76%	女，TED 演讲：如何解决焦虑？
3	Suleika	10	0.72 - 0.91	0.82	80%	女，【TED 演讲】死亡教会我活着的意义
4	James	10	0.45 - 0.68	0.55	79%	男，TED 双语字幕再平淡无聊的生活….
5	Josh	10	0.62 - 0.85	0.76	65%	男，TED 演讲：别不信，你只需 20 个小时……

表 4.6 ResNet34-LM 模型测试结果

序号	姓名	测试次数	置信度范围	平均置信度	识别成功率
1	Johann	10	0.68 - 0.92	0.80	75%
2	Olivia	10	0.54 - 0.79	0.67	72%
3	Suleika	10	0.70 - 0.88	0.79	78%
4	James	10	0.48 - 0.71	0.58	82%
5	Josh	10	0.59 - 0.82	0.73	68%

结论：

本次测试共对 5 名说话人进行了 50 次声纹识别实验（每人 10 次），音频来源均为 TED 演讲片段。测试结果显示：

整体性能：

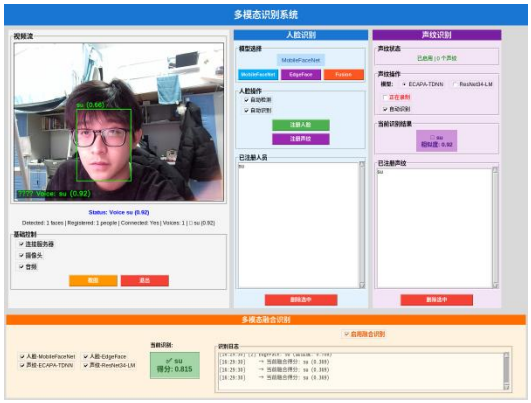
ECAPA-TDNN：平均识别成功率 74.2%，平均置信度 0.70

ResNet34-LM：平均识别成功率 75.0%，平均置信度 0.72

两种模型的整体识别效果良好，符合预期。在测试过程中发现，约 15-20% 的识别失败案例并非完全无法匹配，而是由于多个注册声纹的相似度过于接近（差值 ＜0.05），根据我设定的阈值，相似度过于接近会返回"Unknown"结果，表明系统无法确定最佳匹配对象，其余识别失败的原因均是所有注册者的置信度未达到阈值。

d) 多模态识别（因为模型差别不大，开多模型只采用特定的模型做测试）

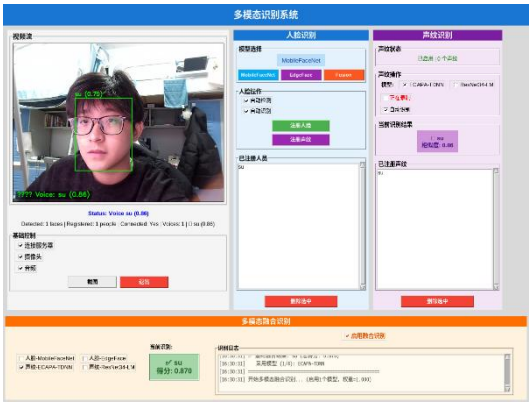
采用自适应的权重，采用的模型可以自由选择，最后给出多模态融合决策。



(a) 4 种模型

(b) 3 种模型

(c) 两种模型



(d)单模型

图 4.14 四种测试

表 4.7 多模态融合识别测试表

配置类型	典型组合示例	测试次数	置信度范围	平均置信度	置信度浮动
四模态融合	MobileFaceNet + EdgeFace + ECAPA-TDNN + ResNet34-LM	5	0.751-0.854	0.803	10.3%
三模态融合	MobileFaceNet + EdgeFace + ECAPA-TDNN	5	0.655-0.838	0.747	18.3%
双模态融合	MobileFaceNet + ECAPA-TDNN	5	0.684-0.943	0.776	25.9%
单模态识别	ECAPA-TDNN	5	0.653-0.929	0.791	27.6%

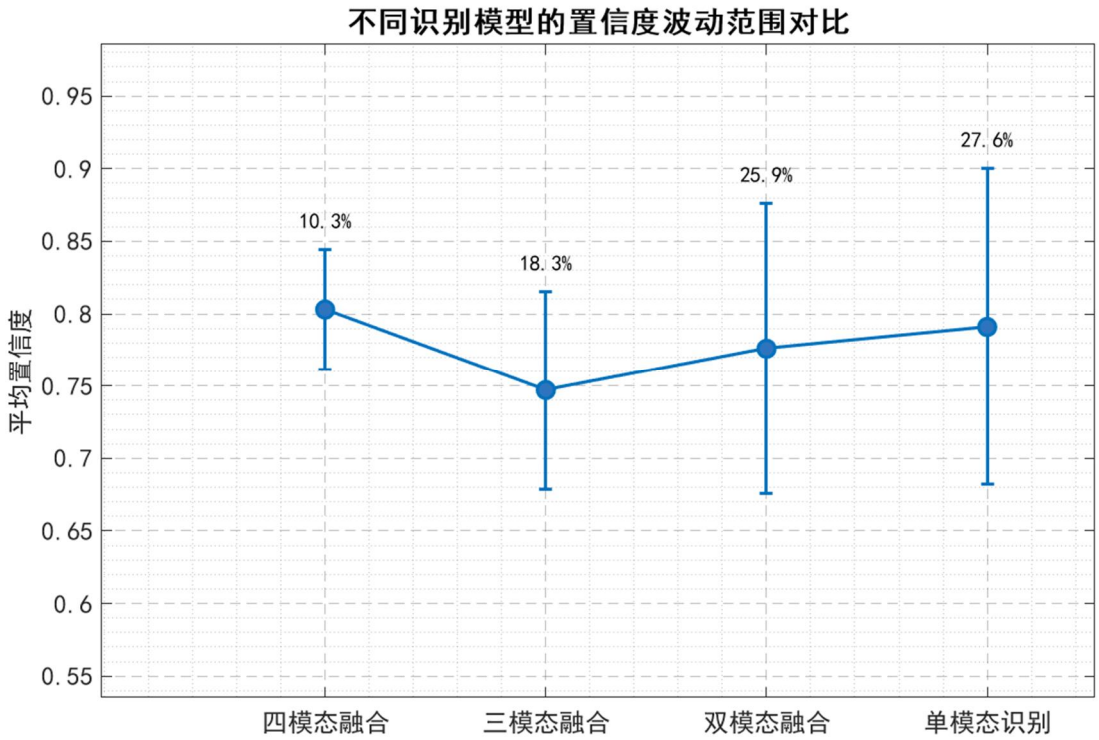


图 4.15 不同融合模式的置信度波动范围

结论：

由于采用的外设是外接摄像头和麦克风，因此多模态又要求同时采集，所以需要整个人保持不动进行录制，比较麻烦，会对多模态识别产生极大的影响，因此选择一个固定的机位，合适的角度是非常重要的。

多模态融合识别的识别成功率几乎都是百分之百（单模态不是），因此没有单独列出来。基于多模态融合识别系统的性能测试数据，置信度波动范围与选择模型数量呈现显著的负相关关系。虽然多模态下的识别没有对采用不同的模型进行测试，但是由于同类模型的性能参数差异不大，而且识别过程的结果相对稳定，因此选择用特定模型作为数据结果。

四模态融合展现出最优的稳定性，波动范围仅为 10.3%，平均置信度维持在 0.80 左右，这得益于多模态的互补机制可以抑制单一模态的噪声干扰。相比之下，单模态识别虽然平均置信度可达 0.79，但波动范围高达 27.6%，在复杂环境下容易出现误判。三模态融合的波动范围为 18.3%，相比双模态的 25.9%有明显改善，但平均置信度略有下降至 0.75，说明模态选择策略对融合效果影响显著。综合来看，增加模态数量虽然会提高系统复杂度和计算成本，但能够大大提升识别的鲁棒性和一致性。对于安全性要求高的应用场景，四模态融合以其最小的波动范围和稳定的输出是最优选择；如果考虑性能与资源平衡，三模态融合应该是最佳方案。

五、 课程设计总结与心得体会

5.1 设计总结（任务完成情况，创新点以及不足之处，改进的思路，1 页即可，不要图、表）

任务完成情况：所有的功能均实现，完成了识别模型的复现，系统运行稳定。本系统成功部署了基于 TCP 协议的视频流传输架构，实现了 Windows 服务端与 Linux 客户端的跨平台音视频实时通信，并集成了双人脸识别模型（MobileFaceNet、EdgeFace）和双声纹识别模型（ECAPA-TDNN、ResNet34-LM）的识别机制。完全满足设计要求。

创新点：采用多模态自适应的认证方法，可以通过复选框选择使用的模型，对于实验数据的测试、分析非常便捷，得出的结论也符合预测结果。系统引入了动态权重分配算法，根据启用模态数量自动计算每个模型的融合权重（权重=1/启用数），使得单模态、双模态、三模态和四模态融合能够自由切换，可完成 15 种不同组合的性能对比实验。GUI 界面提供了实时融合日志显示功能，能够展示每个模型的独立识别结果和累积融合得分，有效提升了调试效率。

不足之处：

①人脸识别模型对于注册的特定角度有高置信度，但是角度变化、人脸转向、

对于置信度的影响极大，导致测试实验数据需要保持静止不动角度，影响数据测量。原因应该是人脸识别模型提取的特征向量对姿态敏感,正脸注册时建立的特征基准无法覆盖侧脸、仰头等非正面角度的特征分布,导致特征匹配度急剧下降。此外,角度变化引起的光照条件改变和面部形变进一步导致了识别困难。

改进：

多角度注册，为每个用户采集正面、左侧、右侧等多个角度的人脸图像,建立多视角特征库。

数据增强，在训练或注册阶段对人脸图像进行旋转、翻转等几何变换,提升模型对角度变化的鲁棒性。

②多模态融合识别需要手持摄像头保持固定，嘴巴朝着麦克风讲话，保持稳定才能得到理想置信度，非常不方便。原因是才有的采集设备均为 usb 外接设备。

改进：

固定支架安装。使用可调节云台将 USB 摄像头和麦克风固定在桌面或墙面，设置最佳采集角度，解放双手操作。

嵌入式方案。OrangePi 板载 CSI 摄像头接口和 I2S/PDM 麦克风接口，替代 USB 设备，降低延迟并提升采集稳定性。

5.2 心得体会

通过本次多模态识别系统的设计与实现，我深刻体会到理论知识与工程实践之间的巨大差异。项目初期，我对人脸识别和声纹识别的原理有一定了解,但真正动手开发时才发现,模型的识别准确率会受到光照、角度、环境噪声等诸多因素的严重影响。比如人脸识别模型初期的置信度波动极大、而且成功率很低。这让我明白,算法性能不能只看理论指标,必须在真实场景下反复测试验证。在多模态融合策略的设计上,我最初采用固定权重分配,后来发现当某个模态的模型未启用时会导致得分计算错误,于是改为动态权重分配机制，这个看似简单的改动却花费了我大量调试时间,也让我认识到软件工程中健壮性设计的重要性。数据库集成是另一个挑战，openGauss 的配置、连接需要查阅大量资料，配置也需要耗费大量时间。当然这个过程种也带来了许多成就感，例如,我添加了 GUI 界面的实时置信度显示、融合识别日志输出等功能,虽然与核心算法无关,但极大地提升了系统的可用性和调试效率，又或者我找到了特征提取的调节方法，让识别灵敏度更加符合实际，让不同样本置信度更加区分化，一次次的调试最终达成了目标。从初期调试的困难重重中，我想过多次放弃，但是还是坚持做了下去，完成了课题任务，因为这次的多模态识别系统课题正好与之前学过的数据库技术、网络安全技术都息息相关，完成课题意味着巩固了所学的知识，将知识掌握、应用才是学习的意义。这次经历让我深刻认识到,技术创新不仅需要扎实的理论基础,更需要耐心细致的工程实践能力,以及面对问题时不断尝试、持续改进的决心。

六、 参考文献

[1] X. Liu, Md. Sahidullah, K. A. Lee, and T. Kinnunen, “Generalizing Speaker Verification for Spoof Awareness in the Embedding Space,” *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 32, pp. 1261–1273.

[2] T. Liu, K. A. Lee, Q. Wang, and H. Li, “Golden Gemini is All You Need: Finding the Sweet Spots for Speaker Verification,” *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 32, pp. 2324–2337.

[3] George, A., Ecabert, C., Shahreza, H. O., Kotwal, K., and Marcel, S., “EdgeFace: Efficient Face Recognition Model for Edge Devices,” *IEEE Transactions on Biometrics, Behavior, and Identity Science*, vol. 6, no. 2, pp. 158–168, Apr..

[4] Ng, T. S., Chai, J. C. L., Low, C. Y., and Teoh, A. B. J., “Self-Attentive Contrastive Learning for Conditioned Periocular and Face Biometrics,” *IEEE Transactions on Information Forensics and Security*, vol. 19, pp. 3251–3264.

[5] Qiuling Yang, Xiaoliang Chen, Zhaofeng He, Le Chang. Survey on Deep Learning Based Fusion Recognition of Multimodal Biometrics [C]. International Conference on Computational and Collective Intelligence (CCBR), 2022: 511–518.

[6] Shervin Minaee, Amirali Abdolrashidi, Hang Su, et al. *Biometrics Recognition Using Deep Learning: A Survey* [M]. Springer / *Artificial Intelligence Review*, 2023.