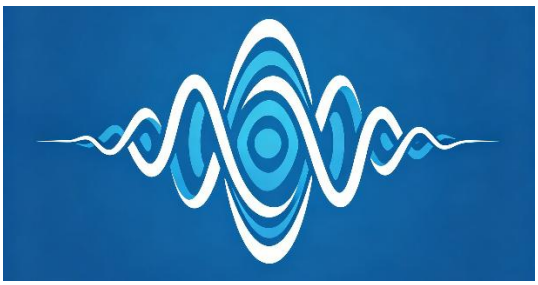


音视频认证系统设计指导书

合肥工业大学 计算机与信息学院

张国富 苏兆品

agent.hfut.edu.cn

安全层

目录

一、 香橙派的基本配置

1.1 需要准备的配件

（1）TF 卡，最小 16GB 容量（推荐 32GB 或以上）的 class10 级或以上的高速闪迪卡

图 1.1 TF 卡

（2）TF 卡读卡器，用于将镜像烧录到 TF 卡中

图 1.2 读卡器

以上两个都是烧录镜像所需要的配件，但是如果有 eMMC 模块则无需购买以上两个配件

（3）无线 USB 网卡（务必去官方说明书查看板子支持的网卡型号）

我用的是香橙派 5plus，说明书里写明支持 RTL8723BU。一定一定要去看官方支持的网卡型号，不能乱买！

如果有路由器则直接可以用网线连接上网，无需网卡。若用网线连接上网，将网线插在图 1.4 标红的位置上。

目前**测试过的**能用的 USB 无线网卡如下所示，其他型号的 USB 无线网卡请自行测试，如果无法使用就需要移植对应的 USB 无线网卡驱动

序号	型号	实物图片
1	RTL8723BU 支持 2.4G WIFI+BT4.0	
2	RTL8811 支持 2.4G +5G WIFI	
3	RTL8821CU 支持 2.4G +5G WIFI 支持 BT 4.2	

图 1.3 香橙派 5plus 官方支持的网卡型号

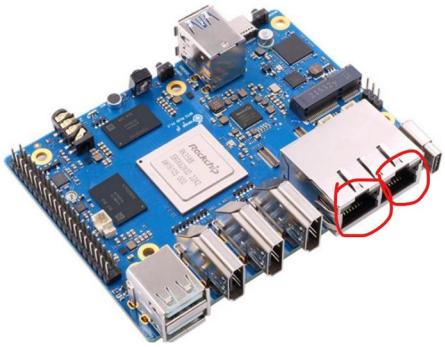


图 1.4 网线应连接的位置（二者选其一即可）

考虑到大部分香橙派没有直接配置显示屏，可以采购以下配件实现显示香橙派界面。

（4）HDMI 线（如果你有显示屏，可以直接用这根线连接到显示屏上显示界面）

图 1.5 HDMI 线

（5）USB 采集卡 （采集卡配 HDMI 线可实现自己电脑显示香橙派界面，具体可看 1.3 节）

图 1.6 USB 采集卡

（6）香橙派还需要配备一套键鼠，需要自己购买，只要有 usb 线能跟香橙派相连即可

1.2 烧录镜像

烧录镜像有两种方法，第一种是用过 TF 卡和 TF 读卡器实现，第二种是通过 eMMC 模块实现。第一种方法官方说明书有非常详细的步骤引导，可以跟着那个做，这里着重讲第二种方式。

（1）首先要将 eMMC 模块插入开发板中，如图 1.7 所示（不同型号板子插口参考官方说明书）

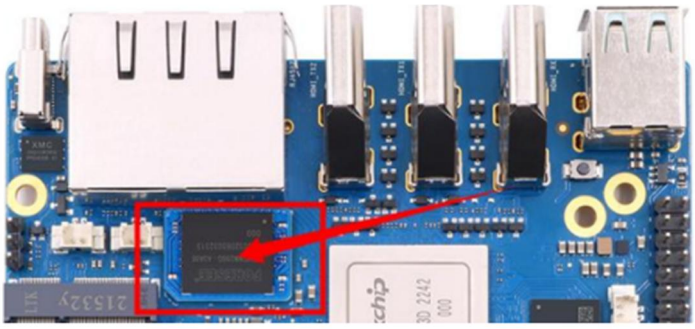


图 1.7 eMMC 模块接口

（2）还需要准备一根品质良好的 Type-C 接口的数据线

图 1.8 Type-C 接口的数据线

（3）然后从 **Orange Pi 的资料下载页面**（选好你的开发板对应型号）下载瑞芯微驱动 DriverAssitant_v5.12.zip 和 MiniLoader 以及烧录工具 RKDevTool_Release_v3.15.zip

a. 在 Orange Pi 的资料下载页面首先选择官方工具，然后进入下面的文件夹中

图 1.9 进入这个所选中的文件夹

b. 然后下载下面的所有文件

图 1.10 下载选中的所有文件

（4）然后从 **Orange Pi 的资料下载页面**下载想要烧录的 Linux 操作系统镜像文件压缩包，然后使用解压软件解压，解压后的文件中，以“**.img**”结尾的文件就是操作系统的镜像文件，建议把 img 结尾的文件名改为 orangepi.img 或其他比较短的名字，这样方便后续烧录时可以看到进度条。

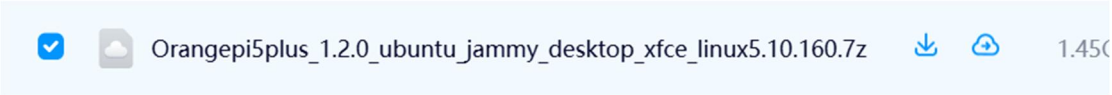


图 1.11 我下载的 linux 镜像系统

（5）然后用解压软件解压 DriverAssitant_v5.12.zip，再在解压后的文件夹中找到 DriverInstall.exe 可执行文件并打开即可

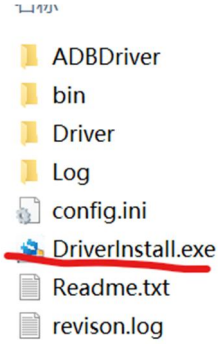


图 1.12 点击 DriverInstall.exe

（6）打开 DriverInstall.exe 后安装瑞芯微驱动的步骤如下所示

a. 点击“**驱动安装**”按钮

图 1.13 点击驱动安装

b. 等待一段时间后，会弹出窗口提示“**安装驱动成功**”，然后点击**“确定”** 按钮即可

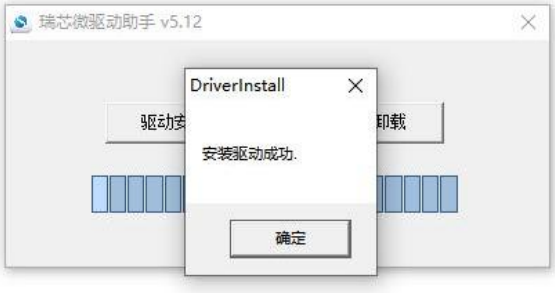


图 1.14 安装驱动成功

（7）然后解压 RKDevTool_Release_v3.15.zip，此软件无需安装，在解压后的文

件夹 中找到 RKDevTool 打开即可

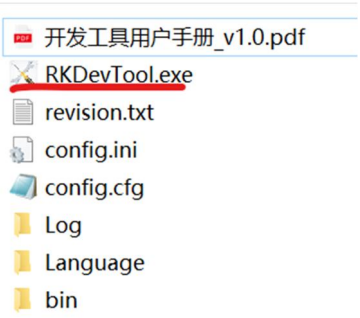


图 1.15 点击 RKDevTool.exe

（8）打开 RKDevTool 烧录工具后，因为电脑此时还没有通过 Type-C 线连接上开发板，所以左下角会提示“**没有发现设备**”

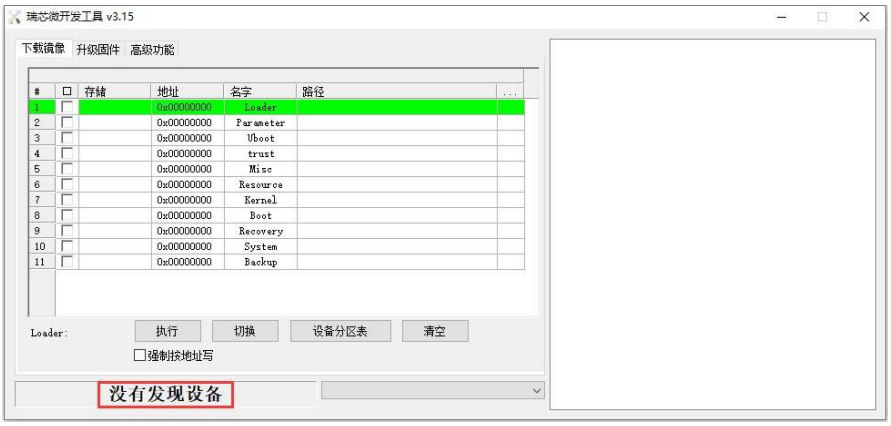


图 1.16 打开 RKDevTool 烧录工具

（9）然后开始烧录 Linux 镜像到 eMMC 中

a. 首先通过 Type-C 数据线连接好开发板与 Windows 电脑，开发板 Type-C 接口的位置如下图所示

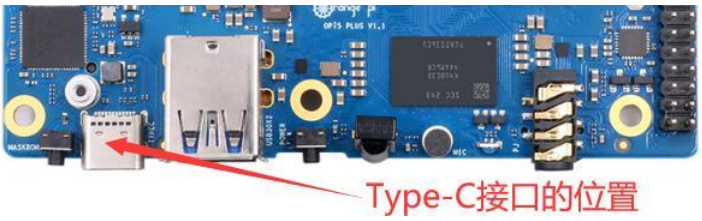


图 1.17 开发板 Type-C 接口的位置

b. 确保开发板没有插入 TF 卡，没有连接电源

c. 然后按住开发板的 MaskROM 按键不放，MaskROM 按键在开发板的位置如下图所示：

图 1.18 开发板的 MaskROM 按键

d. 然后给开发板接上 Type-C 接口的电源，并上电，然后就可以松开 MaskROM

按键了

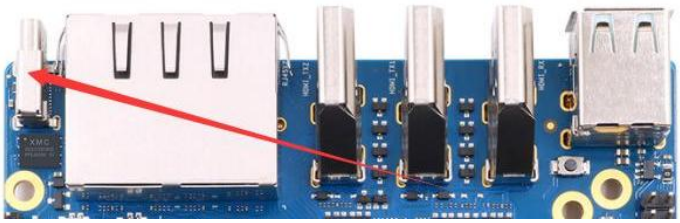


图 1.19 开发板的电源接口

e. 如果前面的步骤顺利，此时开发板会进入 MASKROM 模式，在烧录工具的界面上会提示“**发现一个 MASKROM 设备**”

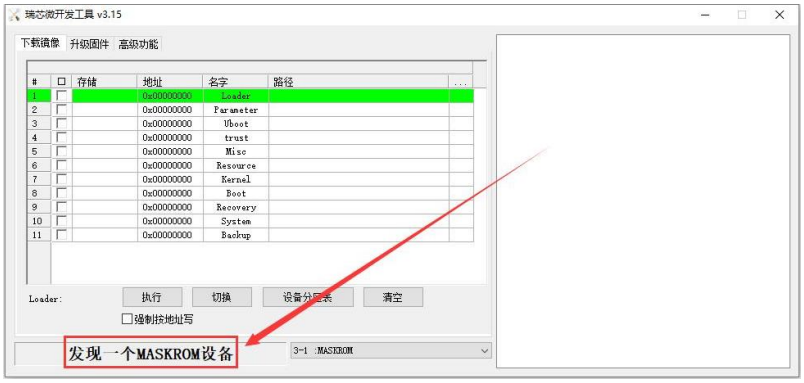


图 1.20 烧录工具的界面上会提示“发现一个 MASKROM 设备”

f. 然后将鼠标光标放在下面的这片区域中

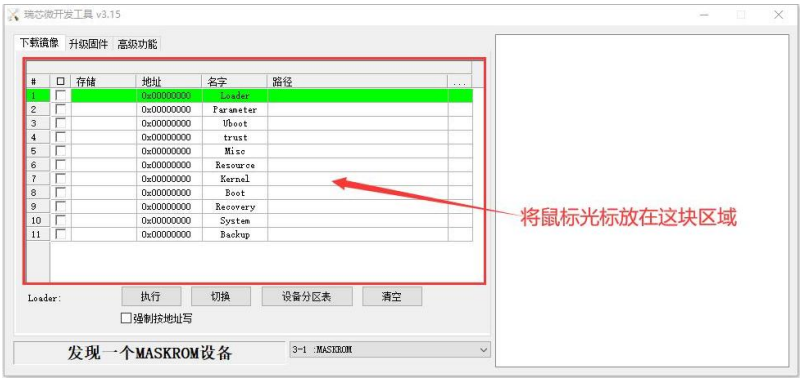


图 1.21 将鼠标光标放在绿色区域中

g. 然后点击鼠标右键会弹出下图所示的选择界面

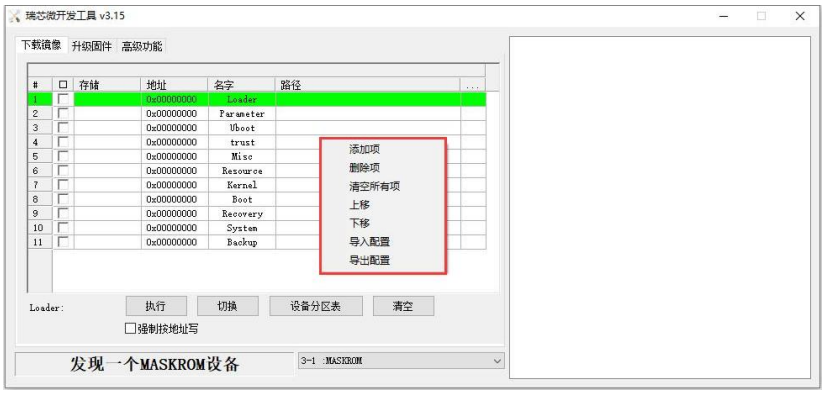


图 1.22 点击鼠标右键

h. 然后选择**导入配置**选项

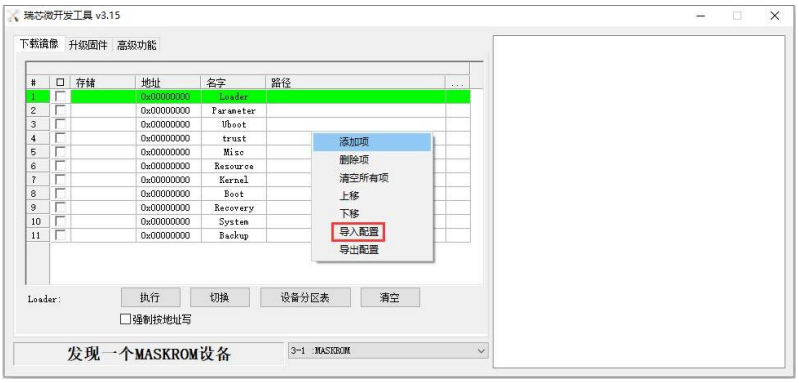


图 1.23 选择导入配置

i. 然后选择前面下载的 MiniLoader 文件夹中的 rk3588_linux_emmc.cfg 配置文件，再点击**打开**

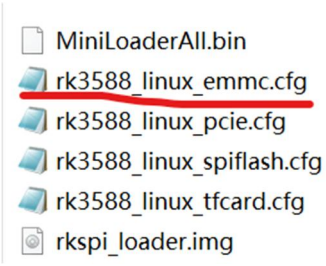


图 1.24 打开 rk3588_linux_emmc.cfg

j. 然后点击**确定**

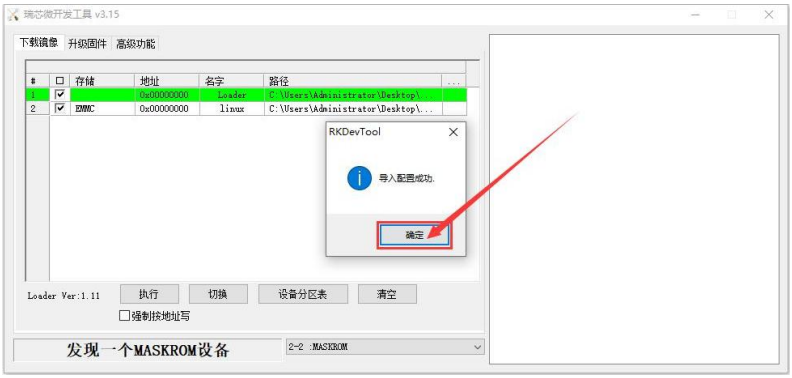


图 1.25 点击确定

k. 然后点击下图所示的位置

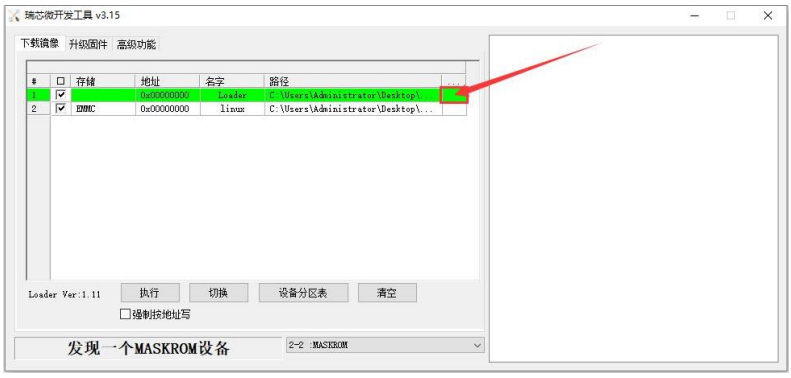


图 1.26 点击绿色位置

l. 再选择前面下载的 MiniLoader 文件夹中 MiniLoaderAll.bin，再点击**打开**

图 1.27 选择 MiniLoaderAll.bin

m. 然后点击下图所示的位置

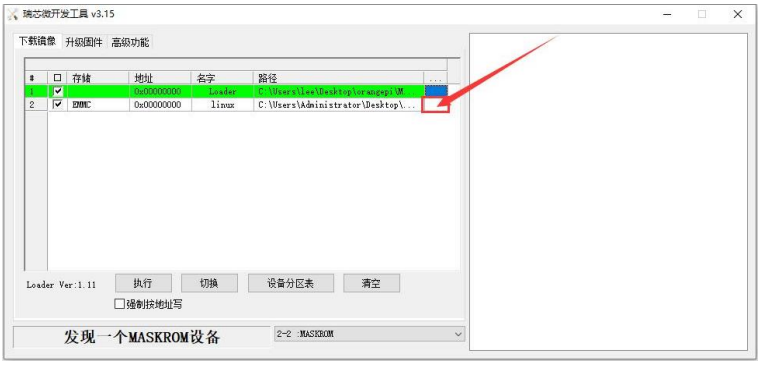


图 1.28 点击红色区域

n. 然后选择想要烧录的 linux 镜像的路径，再点击**打开，选择之前下载好的 linux 镜像路径**

o. 然后请勾选上**强制按地址写**选项

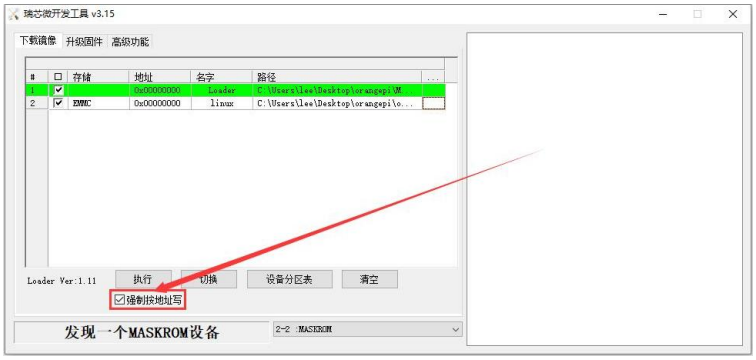


图 1.29 勾选上强制按地址写选项

p. 再点击执行按钮就会开始烧录 linux 镜像到开发板的 eMMC 中

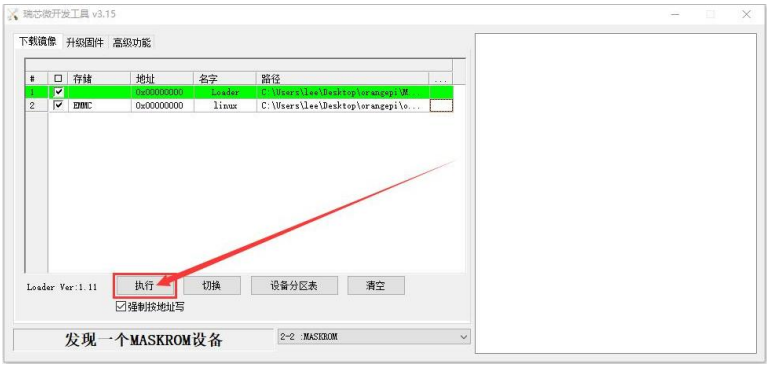


图 1.30 点击执行

q. linux 镜像烧录完后的显示 log 如下图所示

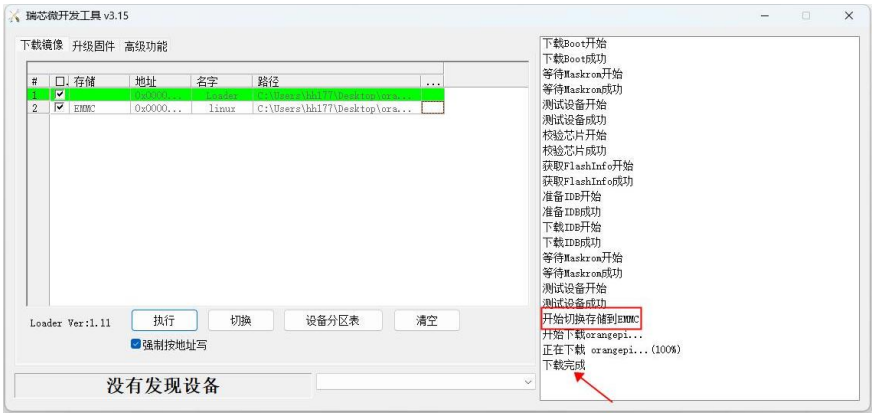


图 1.31 烧录完成界面

r. 烧录完 linux 镜像到 eMMC 中后，linux 系统会自动启动。

1.3 香橙派启动

（这里参考 b 站视频：https://www.bilibili.com/video/BV1aq421w7x1/?spm_id_from=333.788.videopod.sections）

其实官方有好几种登录香橙派的方式，例如调试串口、ssh 登录等等，但是我认为用 HDMI 线配上 USB 采集卡，再加上本地电脑下载 OBS 软件，通过这种方式登录和显示香橙派界面是最方便的。

（1）本地电脑下载 OBS 软件（下载地址：https://obsproject.com/download）

我下载的是 OBS 31.1.2，但是现在最新版本 32.0.4 我试过也是可以的。

a. 从官网下载 OBS installer

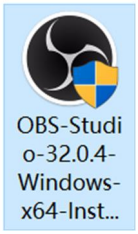


图 1.32 下载 OBS installer

b. 双击 OBS installer，出现如下界面，点击 Next

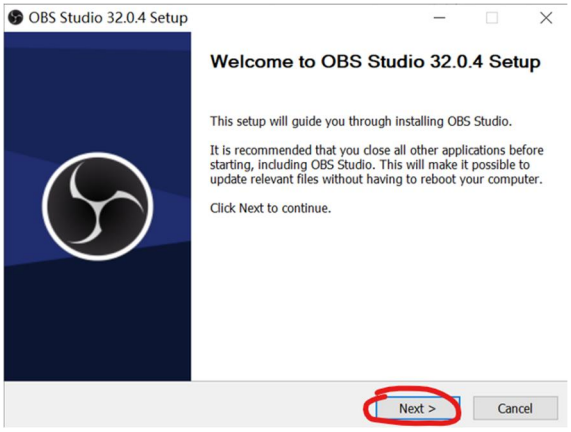


图 1.33 点击 Next

c. 选择 OBS 的下载路径，点击 Install

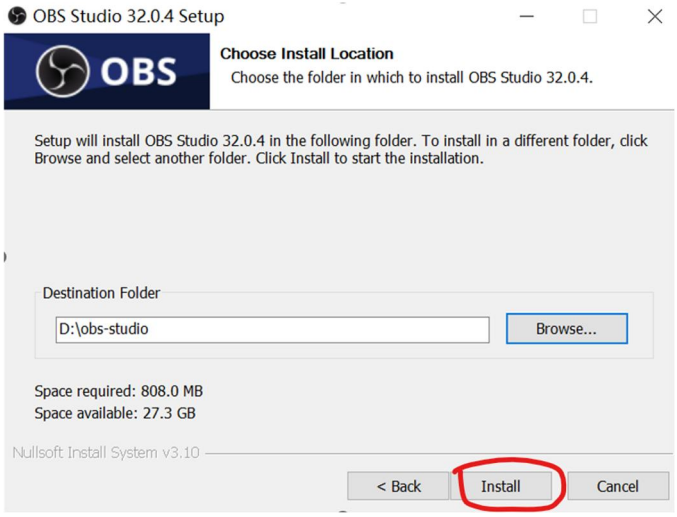


图 1.34 点击 Install

d. 等待安装

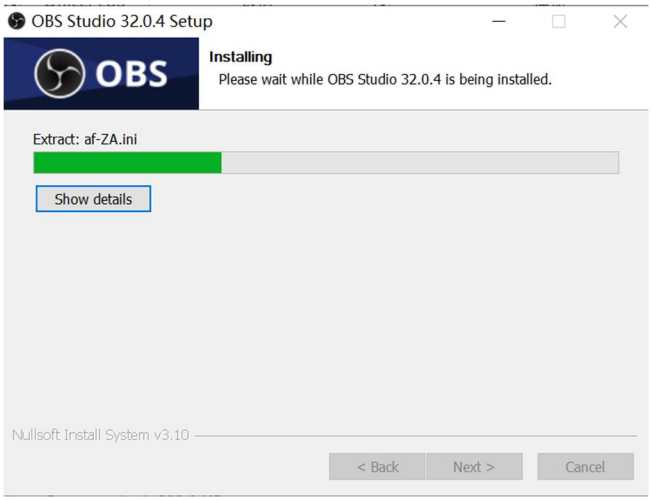


图 1.35 安装界面截图

e. 安装完毕，点击 Finish

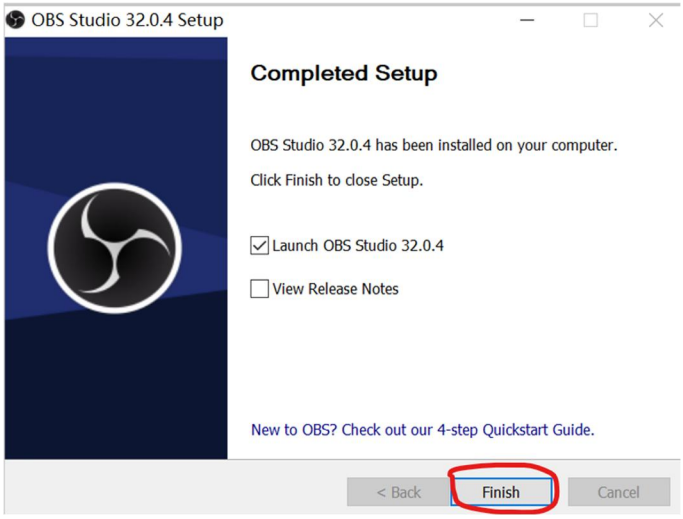


图 1.36 点击 Finish

f. OBS 软件显示界面

图 1.37 OBS 界面

（2）给香橙派上电，HDMI 线接入香橙派的 HDMI 接口，usb 采集卡的 usb 端口

跟电脑相连（圆圈 1），一端跟 HDMI 线相连（圆圈 2），如图所示

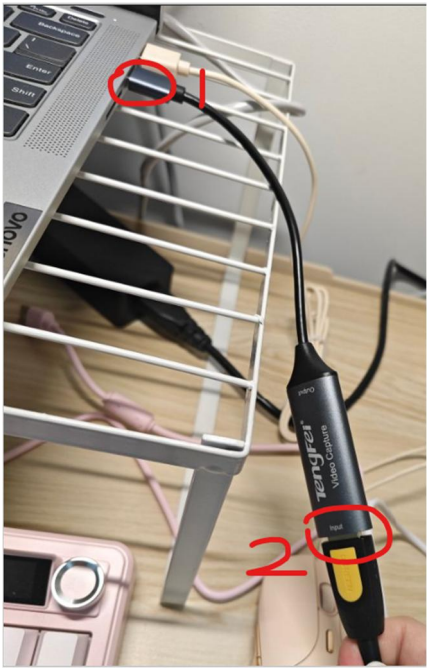


图 1.38 采集卡的连接方式

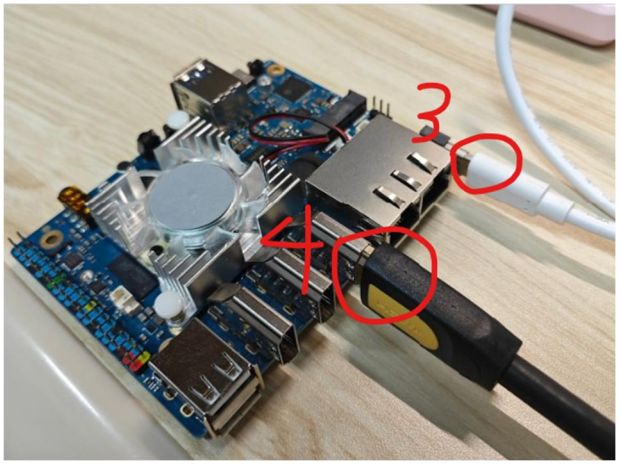


图 1.39 圆圈 3 是电源接口，圆圈 4 是 HDMI 接口

（3）显示香橙派画面

a. 点击圆圈的“+”号

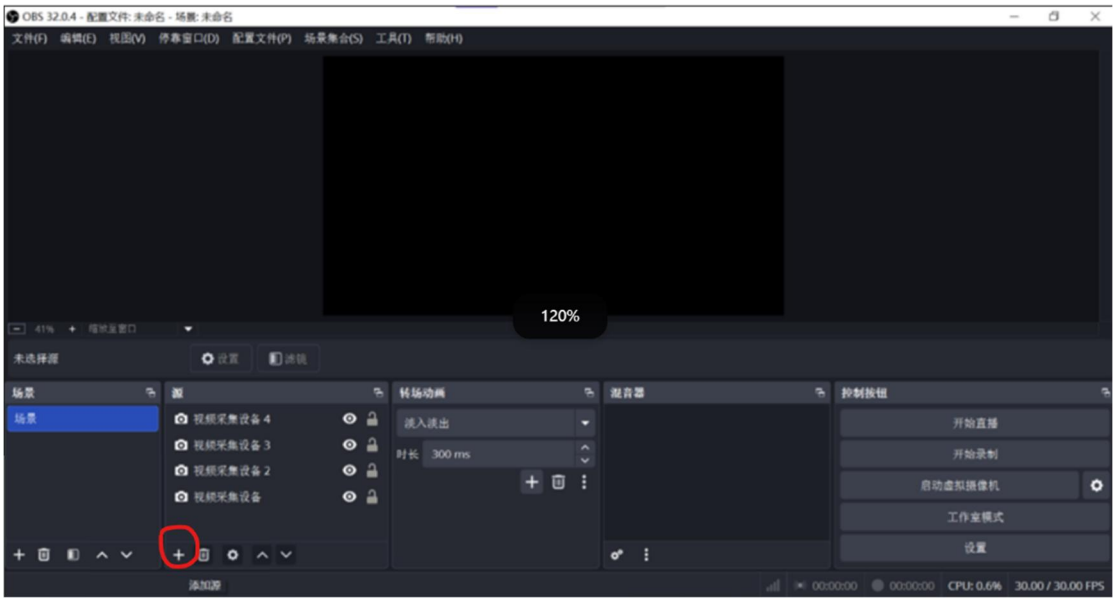


图 1.40 点击圆圈部分

b. 点击视频采集设备

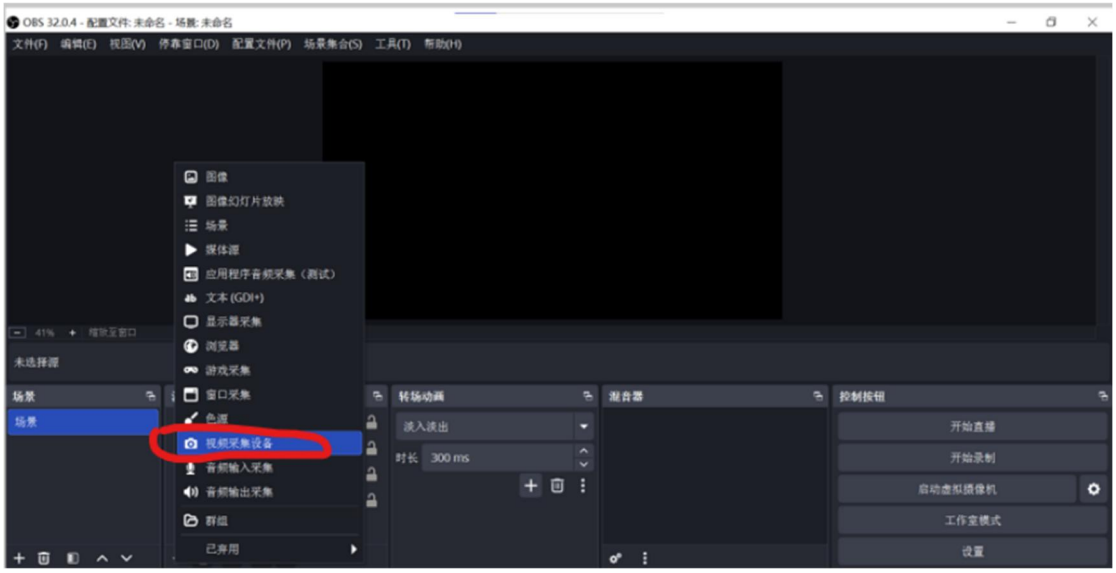


图 1.41 点击视频采集部分

c. 会弹出这样的框，点击确定

图 1.42 点击确定

d. 在设备中选择 USB Video，会显示出香橙派的界面

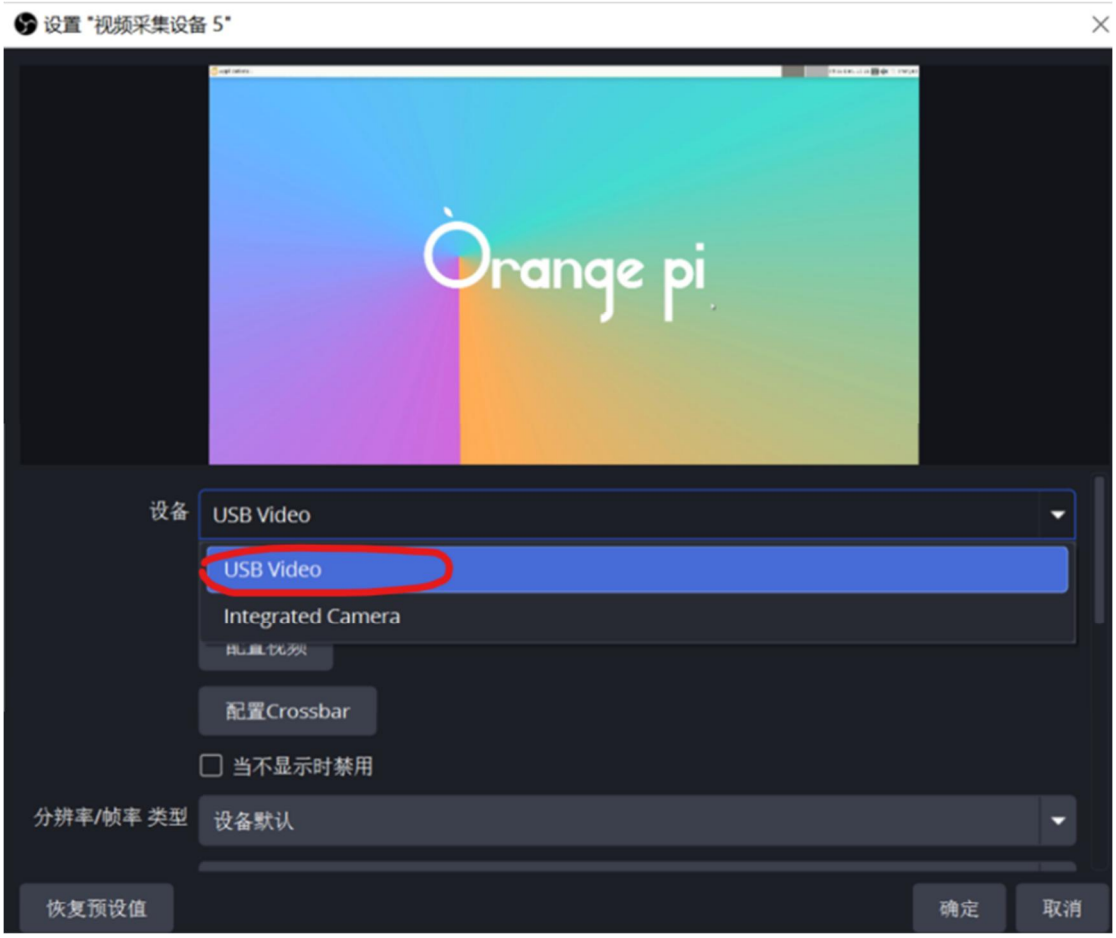


图 1.43 选择设备

f. 鼠标右击屏幕，选择打开预览投影，选择 DISPALY1

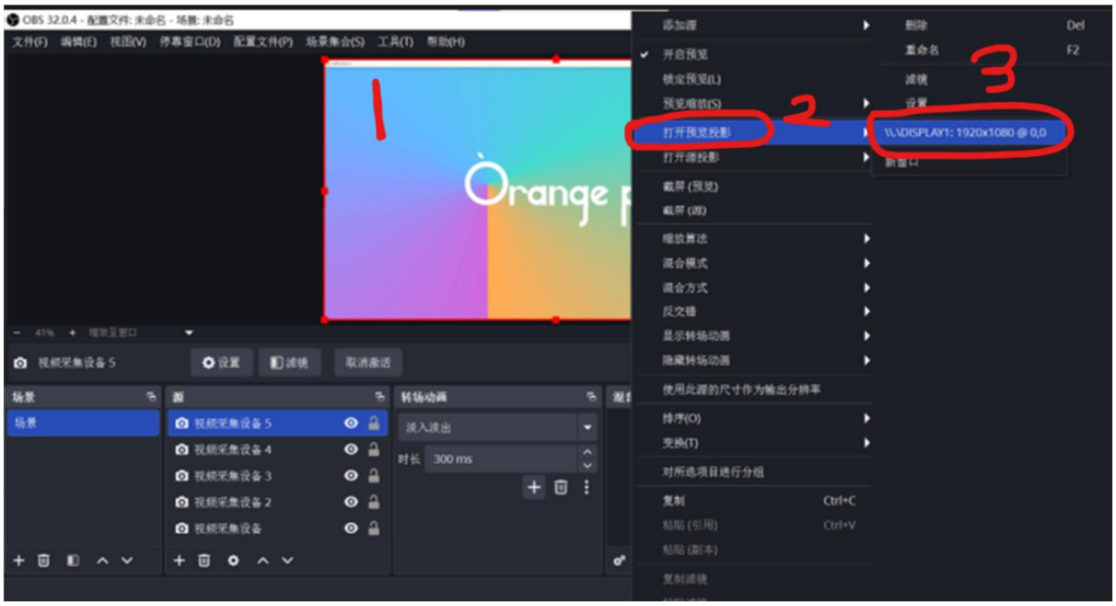


图 1.44 开始预览画面

g. 会出现如下预览画面，显示香橙派屏幕

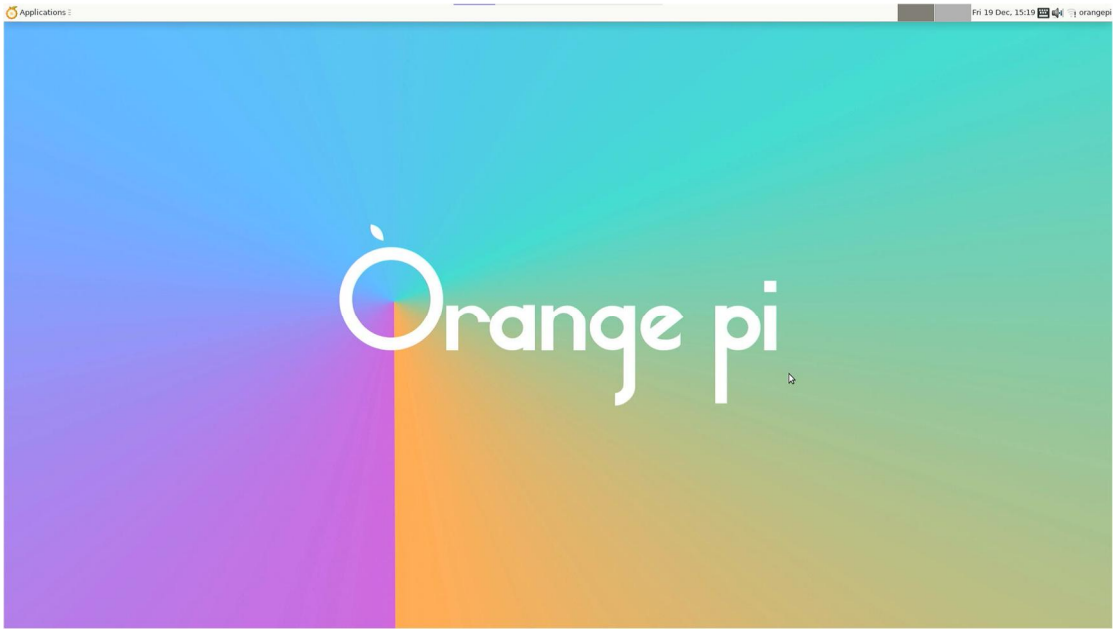


图 1.45 香橙派界面

1.4 香橙派上网

（1）首先，选择插网线的话可以直接去看 1.5 节了，选择无线网卡的话需要跟着以下步骤进行。香橙派上有些 usb 接口插了网卡有反应有些没有反应，这个需要自己去试，只要你用的是官方支持的型号，是一定能成功的，如果没有反应，就换一个 usb 端口。（我用的是 RTL8723BU 蓝牙，记得提前把鼠标和键盘也连接到香橙派）

（2）插好网卡之后，linux 系统会自动加载 RTL8723BU 蓝牙和 WIFI 相关的内核模块，通过 lsmod 命令可以看到下面内核模块已自动加载

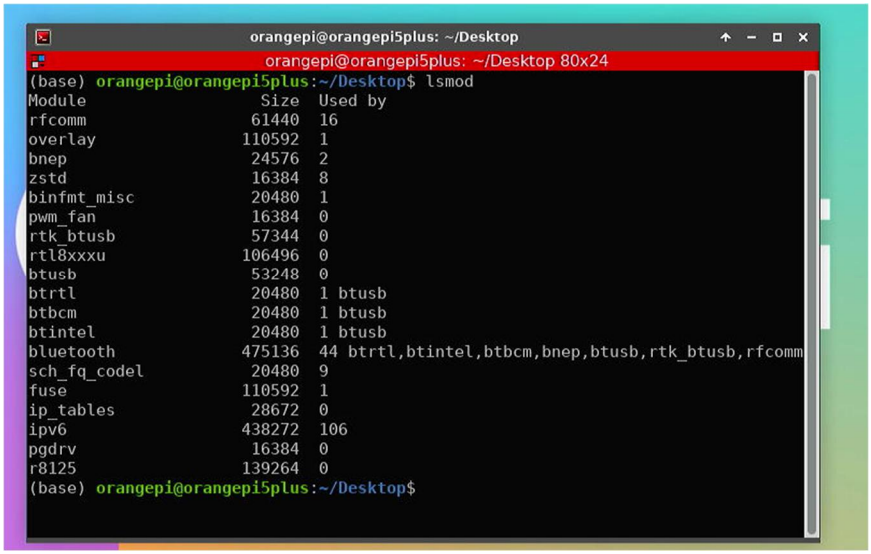


图 1.46 输入 lsmod 后的显示

（3）通过 dmesg 命令可以看到 RTL8723BU 模块的加载信息

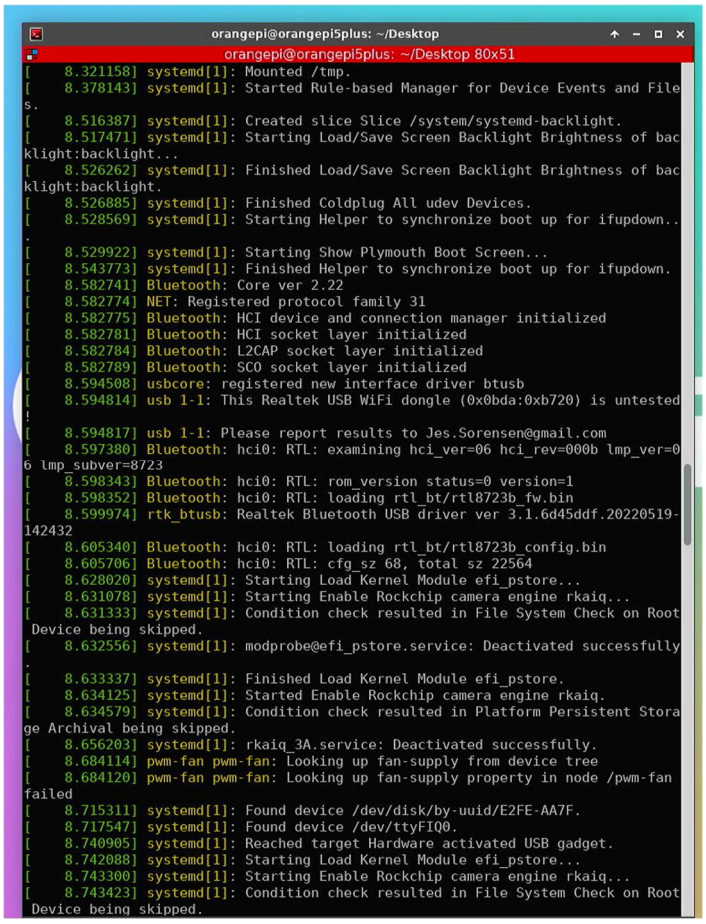


图 1.47 输入 dmesg 后的显示

（4）使用 nmcli dev wifi 命令扫描周围的 WIFI 热点

这里有几个注意要点：1.个人热点要开 2.4GHZ 否则网卡找不到

2.热点名字最好是全英文，并且中间没有空格

图 1.48 周围的热点

（5）然后使用 sudo nmcli dev wifi connect wifi_name password wifi_passwd 命令连接扫描到的 WIFI 热点，其中：

a. wifi_name 需要换成想连接的 WIFI 热点的名字

b. wifi_passwd 需要换成想连接的 WIFI 热点的密码

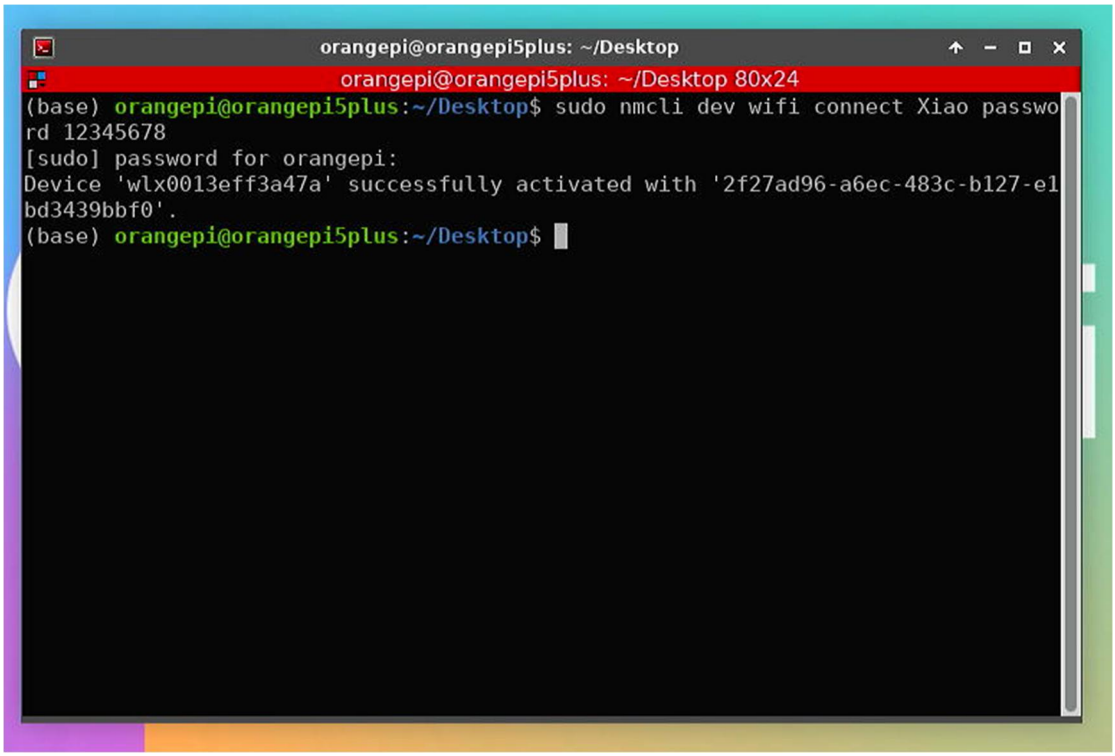


图 1.49 成功连接热点的显示

1.5 在香橙派上下载 QQ

（这里参考 b 站视频：https://www.bilibili.com/video/BV1JZ421v7uu?spm_id_from=333.788.videopod.sections）

（1）首先在左侧选项里找到 Internet，选择里面的浏览器（确保这时已经连上了网）

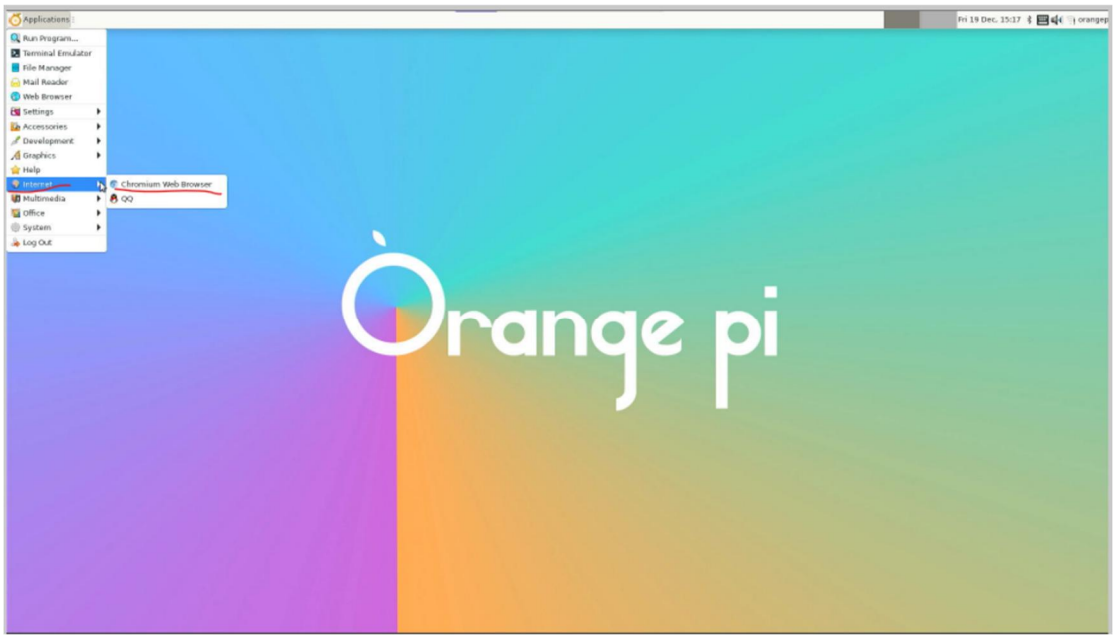


图 1.50 点击路由器

（2）来到谷歌浏览器，在上面的地址框中填入百度地址 www.baidu.com，并回车

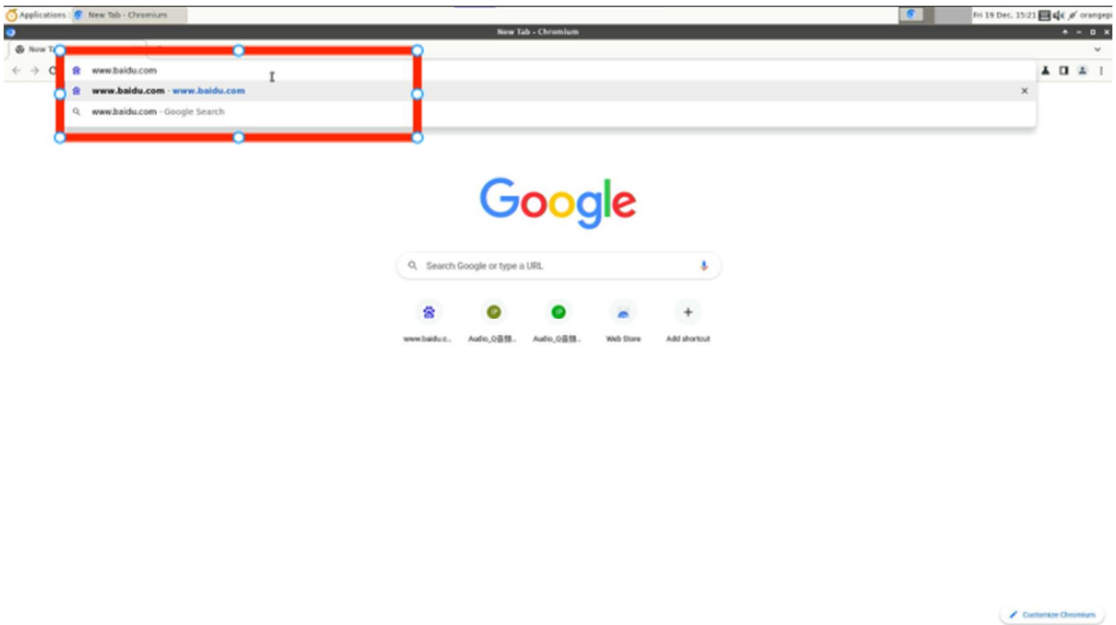


图 1.51 进入百度

（3）在百度搜索框输入 qq

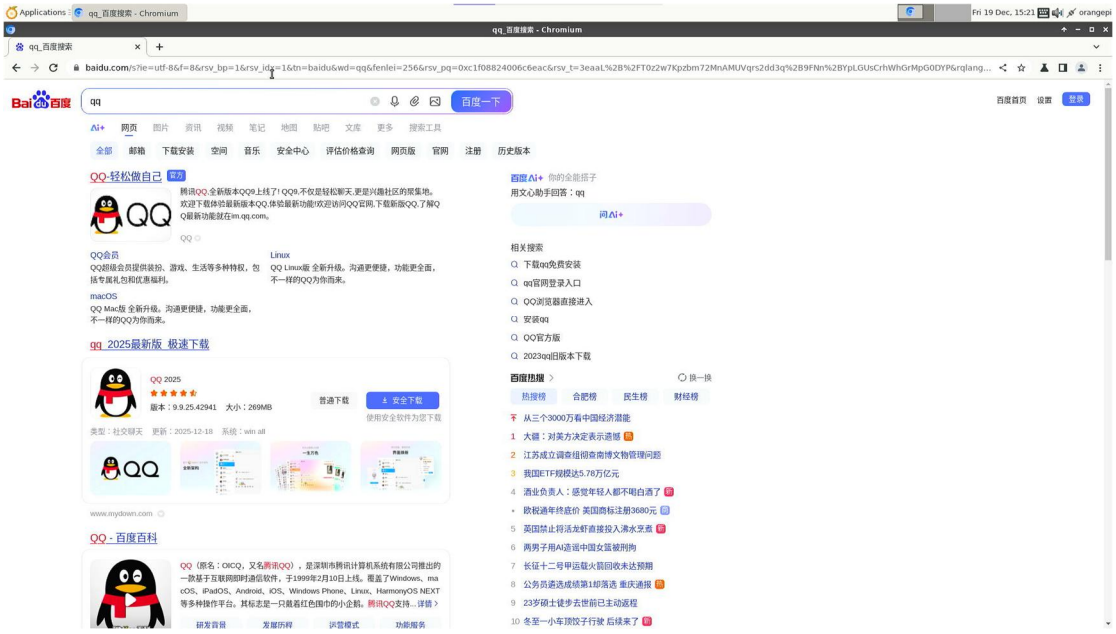


图 1.52 搜索 qq

（4）选择 qq 的 linux 版本

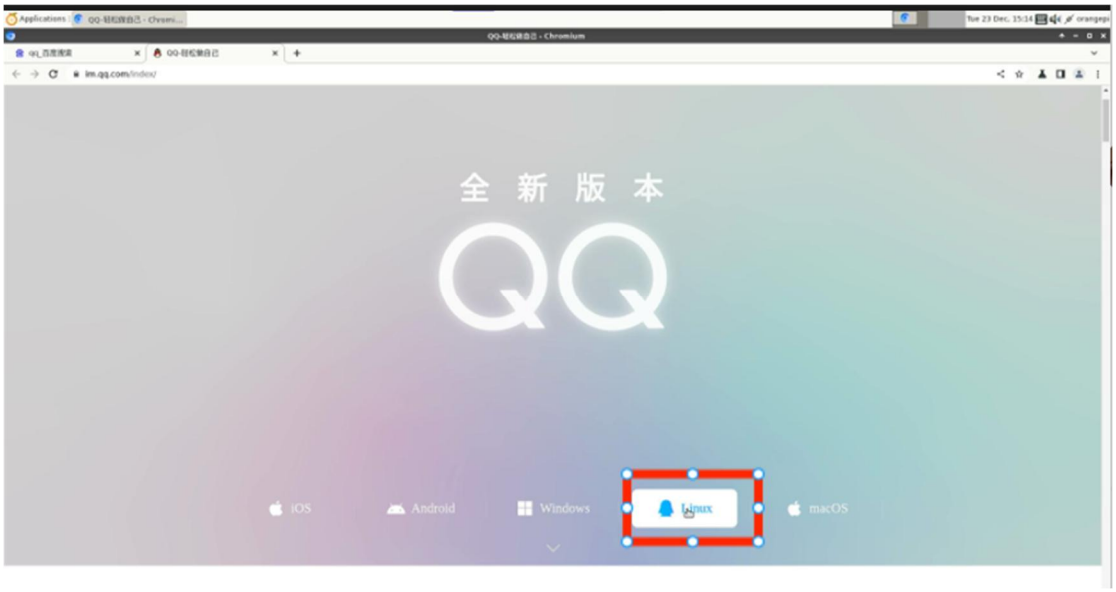


图 1.53 选择 qq 的 linux 版本

（5）选择 Arm 版本的 deb 包

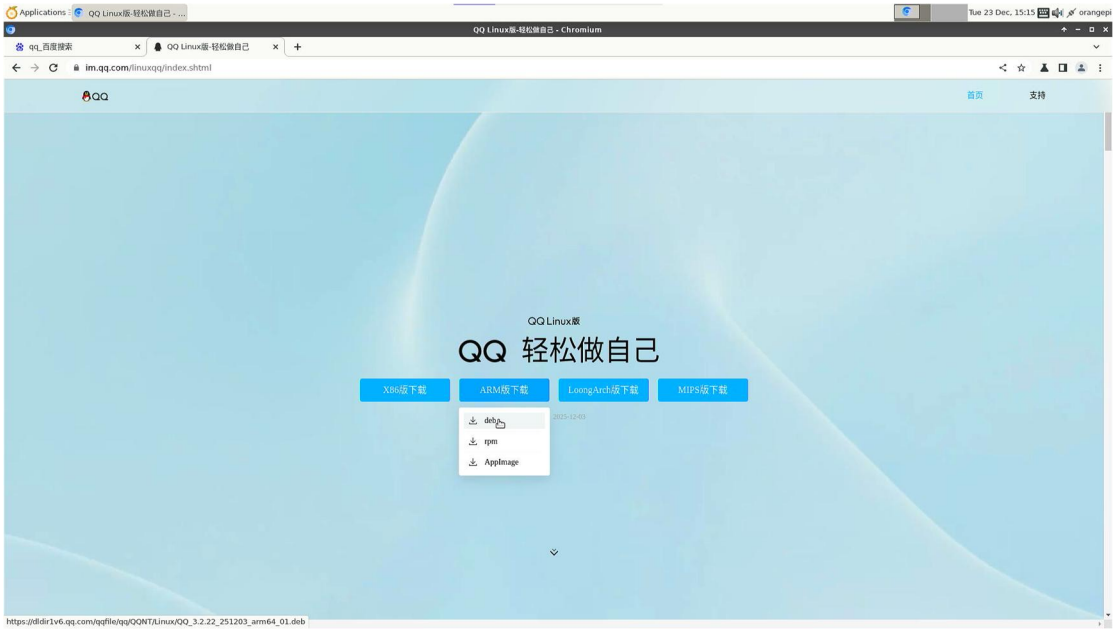


图 1.54 选择 arm 版的 deb 包

（6）下载好后去左侧选项里选择 File Manager，再选择 Downloads

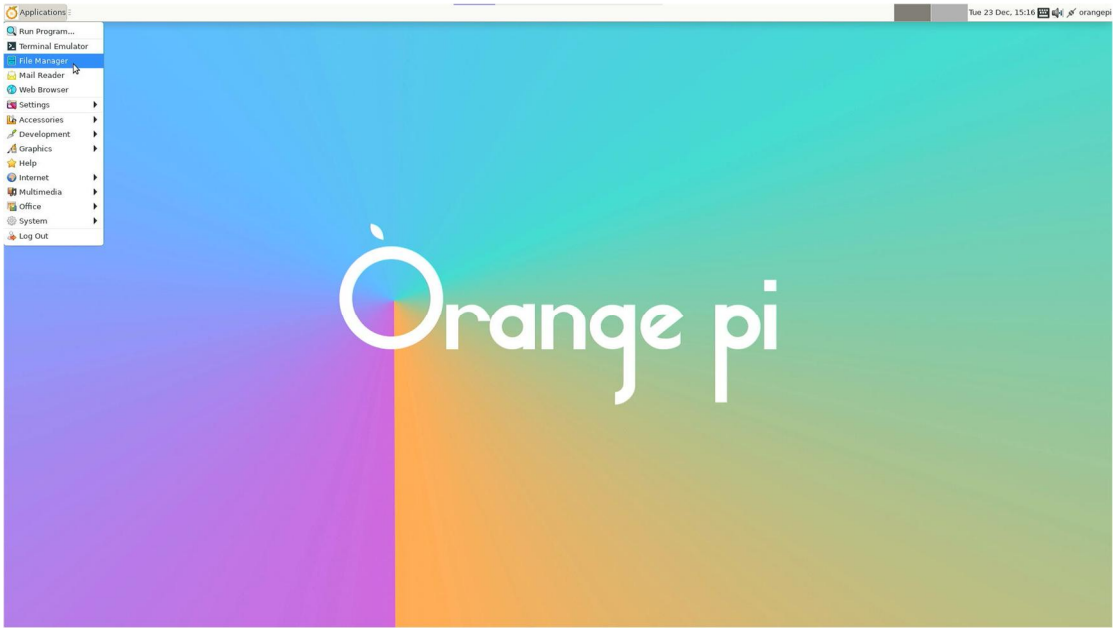


图 1.55 找到 FileManager

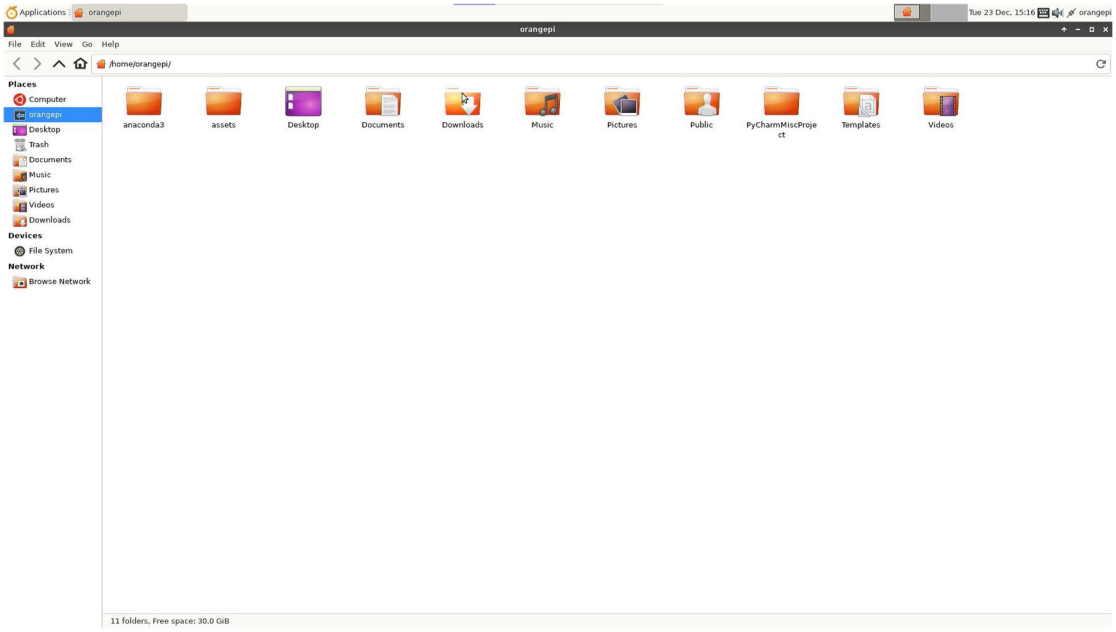


图 1.56 找到 Downloads

（7）在此界面打开终端，输入 ls 查看该目录下文件

输入 sudo dpkg -i QQ 安装包名

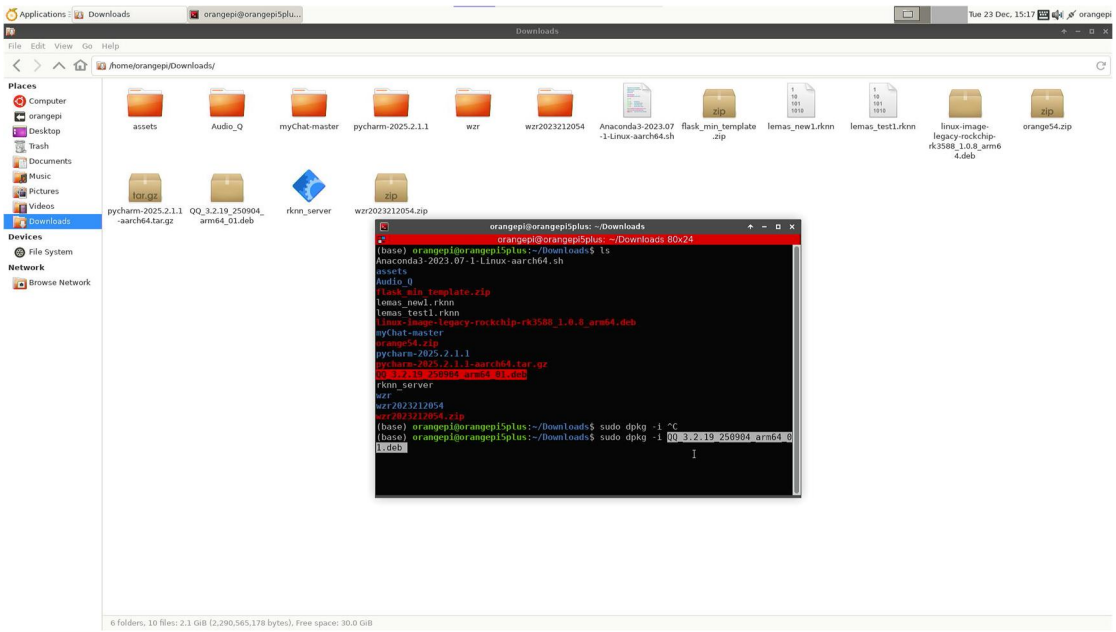


图 1.57 解压安装包

（8）安装完成后，即可在左侧 Internet 里面看见 QQ

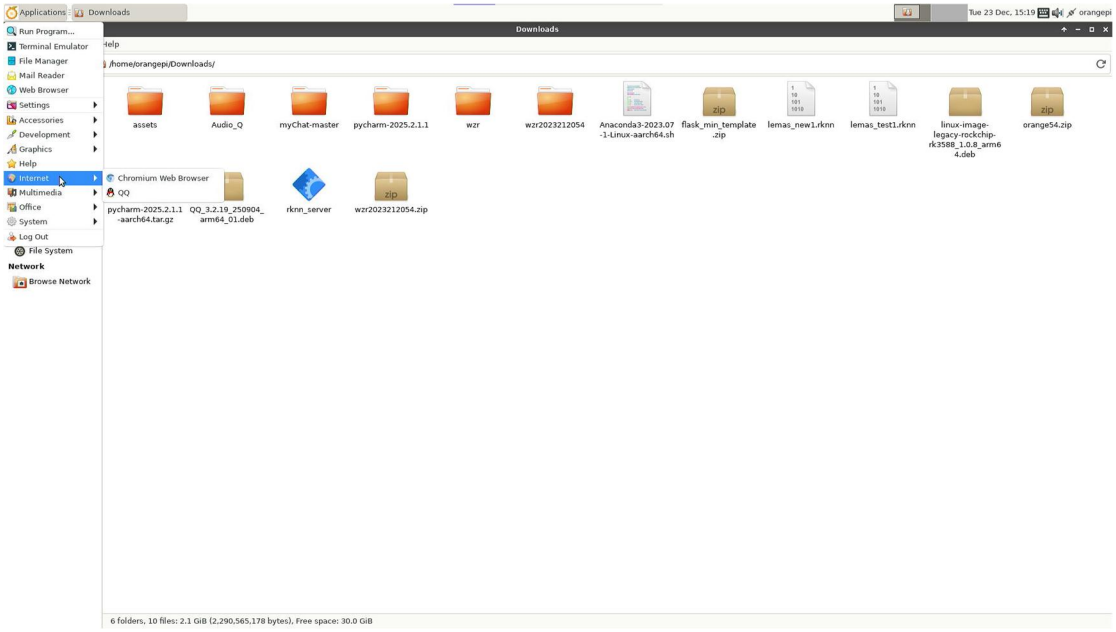

Applications
Downloads
Tue 23 Dec, 15:19
orangepi
Run Program...
Terminal Emulator
File Manager
Mail Reader
Web Browser
Settings
Accessories
Development
Graphics
Help
Internet
Multimedia
Office
System
Log Out
File System
Network
Browse Network
Chromium Web Browser
QQ
Downloads
Help
/home/orangepi/Downloads/
assets
Audio_Q
myChat-master
pycharm-2025.2.1.1
wzr
wzr2023212054
Anaconda3-2023.07-1-Linux-aarch64.sh
flask_min_template.zip
lemas_new1.rknn
lemas_test1.rknn
linux-image-legacy-rockchip-rk3588_1.0.8_arm64.deb
orange54.zip
pycharm-2025.2.1.1-aarch64.tar.gz
QQ_3.2.19_250904_arm64_01.deb
rknn_server
wzr2023212054.zip
6 folders, 10 files: 2.1 GiB (2,290,565,178 bytes), Free space: 30.0 GiB

图 1.58 成功安装 qq

1.6 在香橙派上下载 Pycharm

（这里参考 b 站视频：https://www.bilibili.com/video/BV1JZ421v7uu?spm_id_from=333.788.videopod.sections）

（1）如果在使用香橙派时出现如下界面，直接打开终端输入 reboot 重启香橙派即可

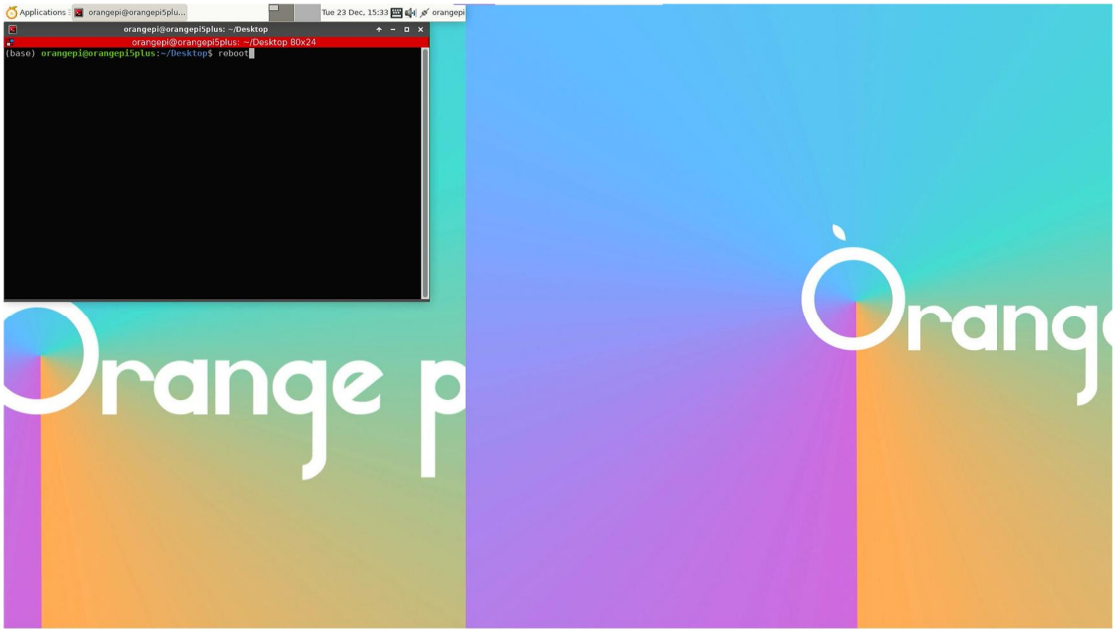


图 1.59 香橙派界面异常时直接重启即可

（2）在百度搜索框中输入 pycharm

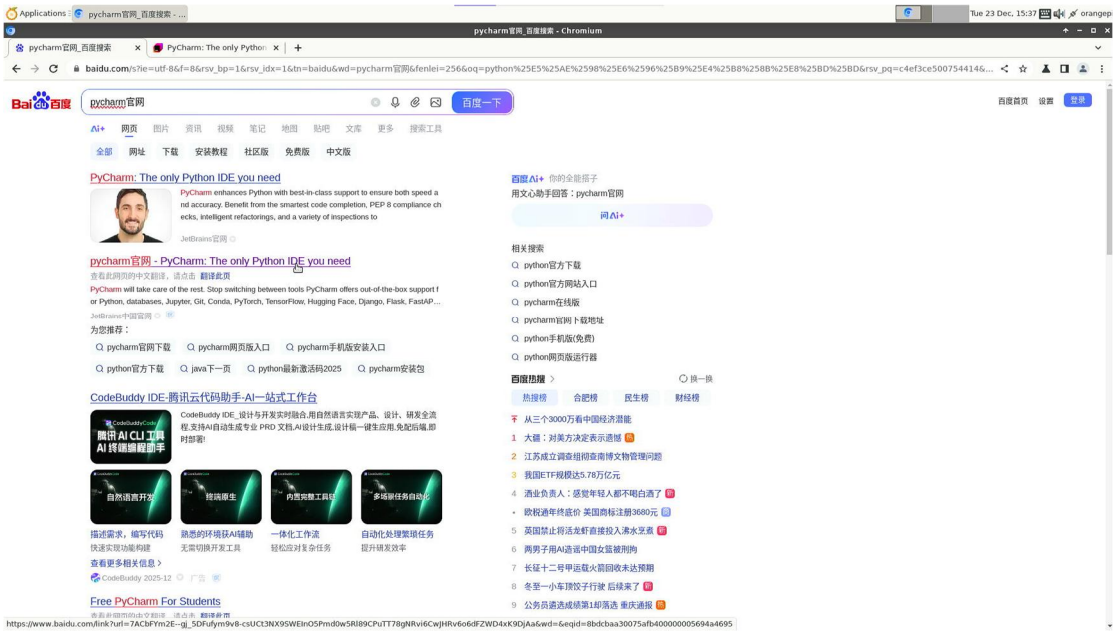


图 1.60 搜索 pycharm

（3）下载 linux 的 arm 版本

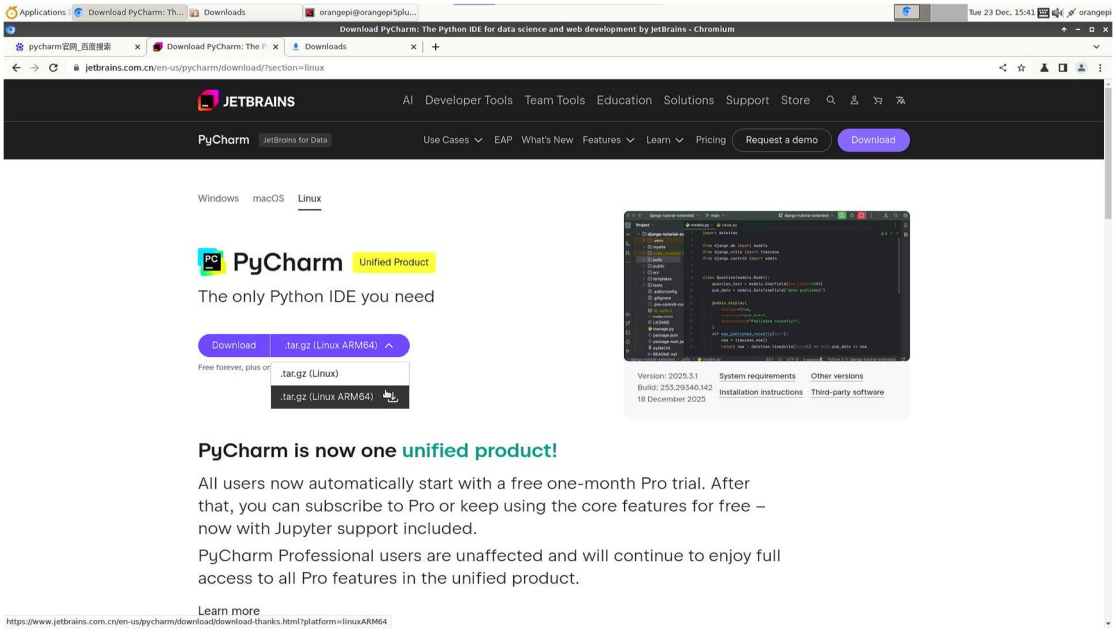


图 1.61 下载 pycharm 安装包

（4）输入 ls 后，再输入 tar -xzvf pycharm 安装包解压安装包

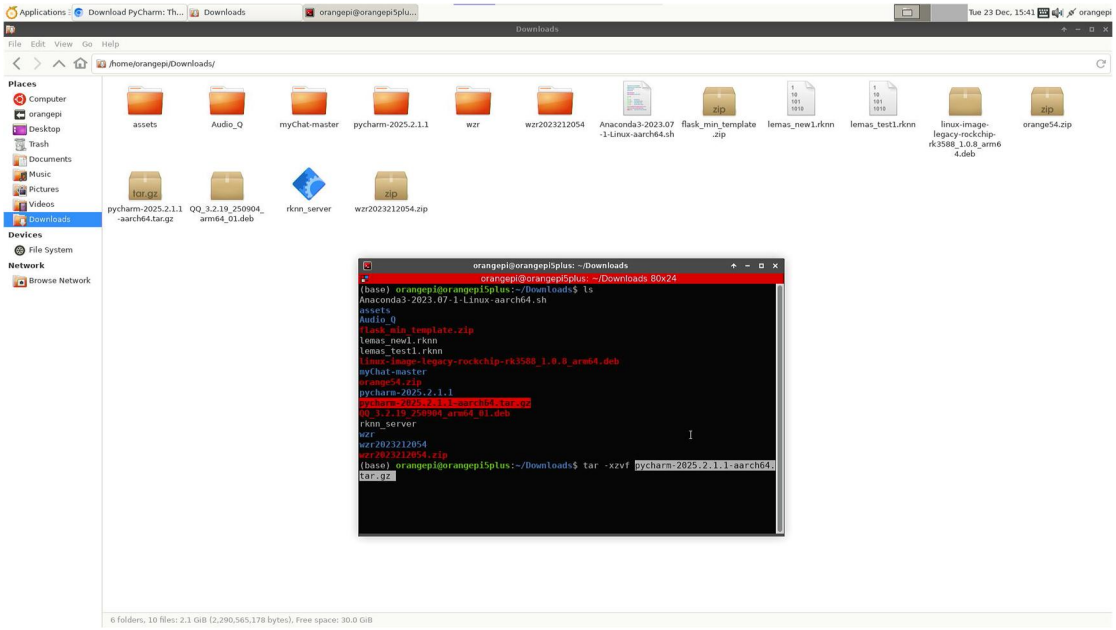


图 1.62 解压 pycharm 安装包

（5）解压后进入 pycharm 文件夹里的 bin 目录，bin 目录下有个 pycharm.sh，在此页面打开终端，输入./pycharm.sh

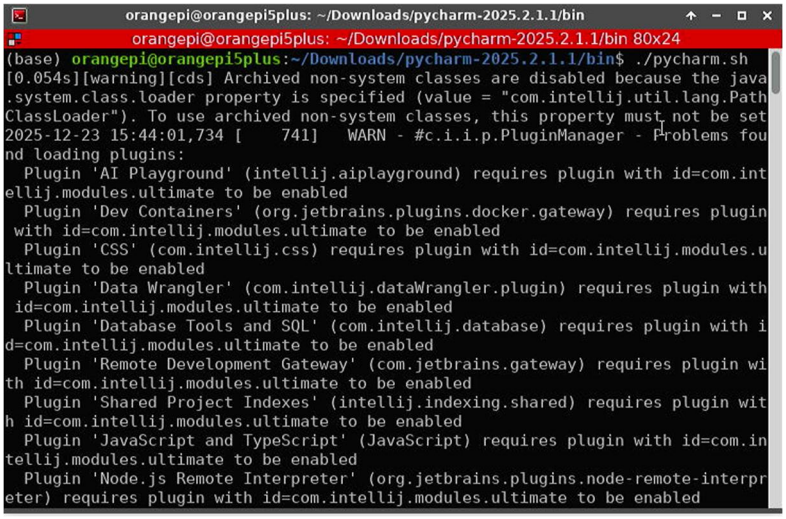


图 1.63 启动 pycharm

（6）设置桌面快捷方式，在左下角的设置里选择 create desktop entry

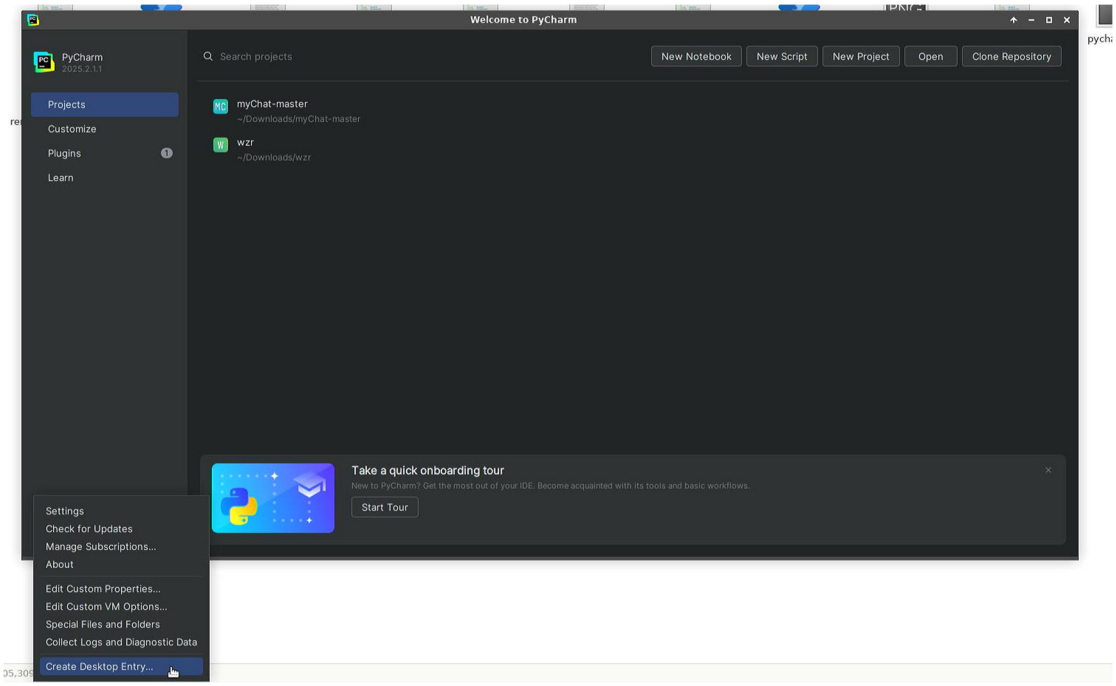


图 1.64 设置桌面快捷方式

（7）在弹出来的页面里勾选住选项，点击 ok，随后会让你输入管理员密码，这个官方说明书里都有写

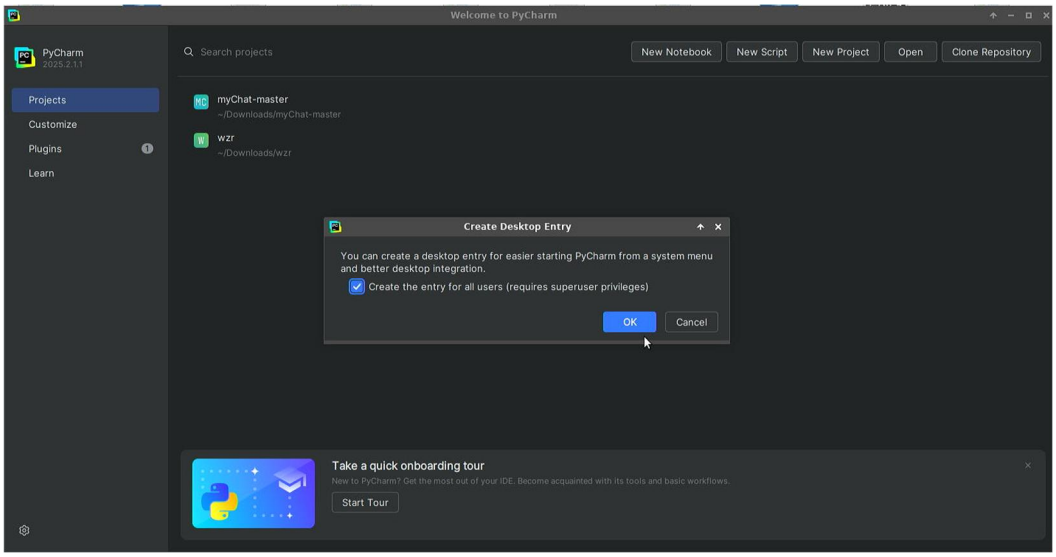


图 1.65 点击 ok

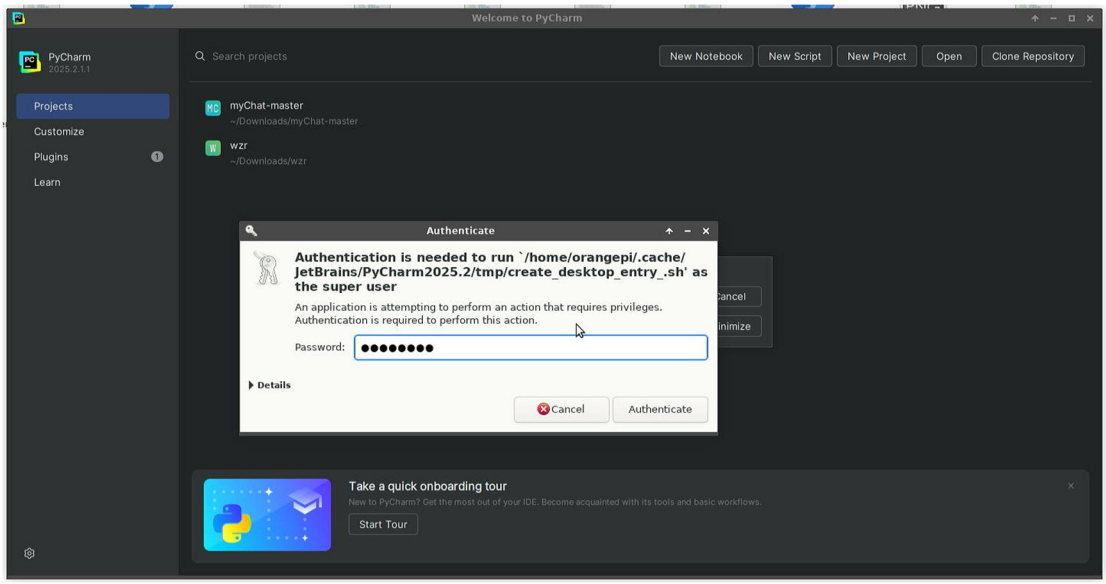


图 1.66 输入管理员密码

（8）随即可在左侧的 development 里面找到 pycharm

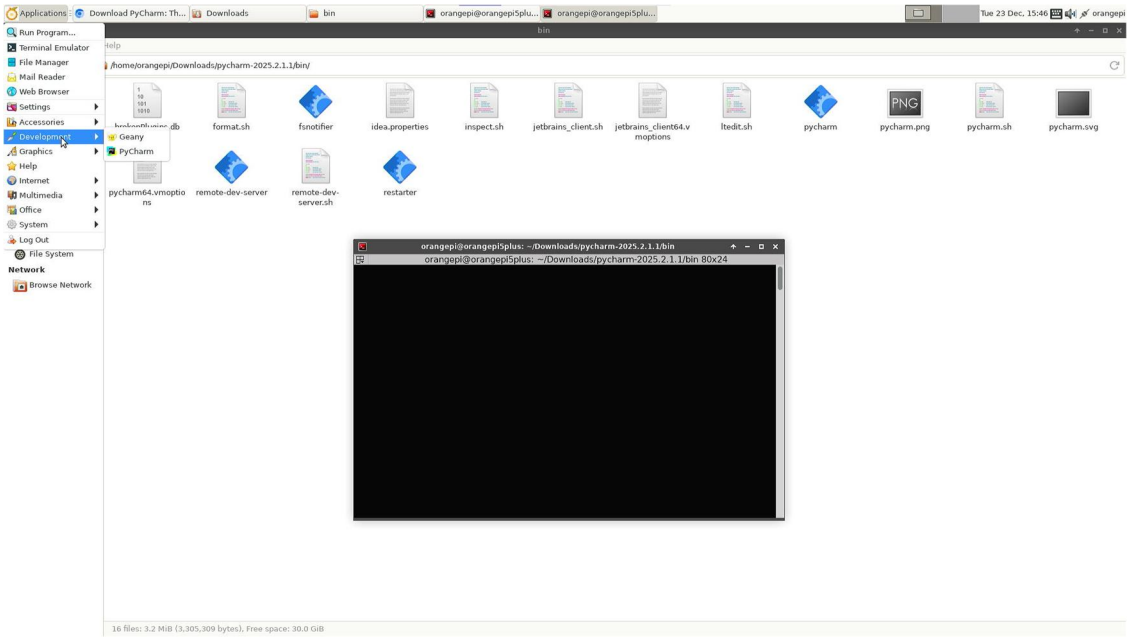


图 1.67 桌面快捷方式建立成功

1.7 在香橙派上下载 Anaconda

（这个官方说明书会有详细教程，这里不仔细说了）

（1）在地址框中输入以下地址，https://mirrors.tuna.tsinghua.edu.cn/anaconda/archive/Anaconda3-2023.07-1-Linux_-aarch64.sh 搜索并下载安装 Anaconda3 的脚本，下载完成后将得到 Anaconda3-2023.07-1-Linux-aarch64.sh 文件。

（2）与之前一致，到达这个.sh 所在的目录，打开终端，运行以下命令 sh Anaconda3-2023.07-1-Linux-aarch64.sh 来安装

（3）然后安装脚本会输出下面的提示信息，点击回车键继续安装。

图 1.68 点击回车

（4）在点击回车键后，会出现 Anaconda3 的一些介绍信息，一直点击“↓”键。

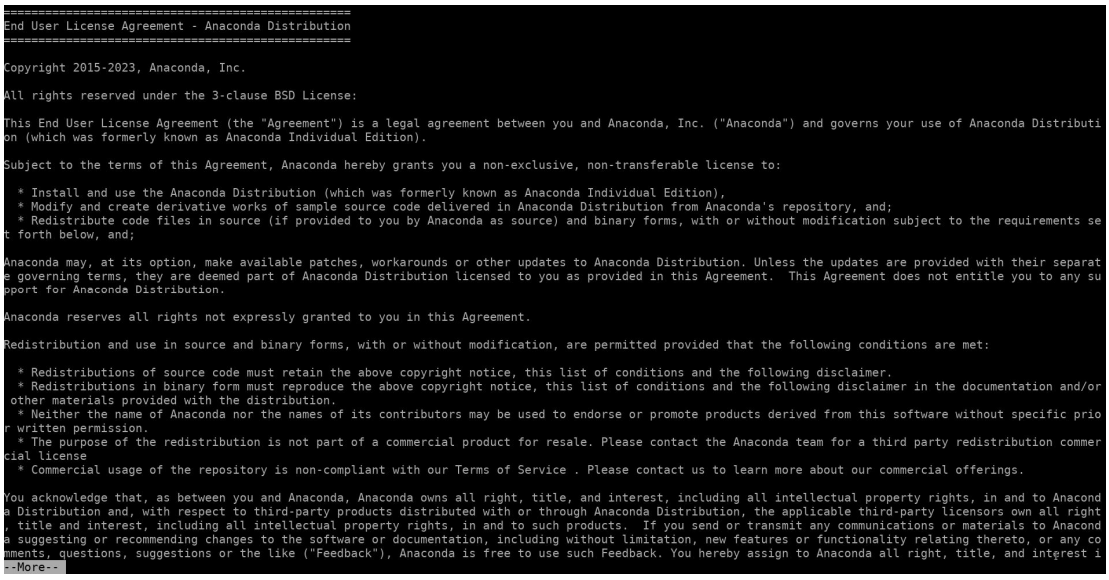


图 1.69 一直点击“↓”键

（5）然后安装脚本会提醒是否接受许可证条款，此时输入 yes 按回车键既可。

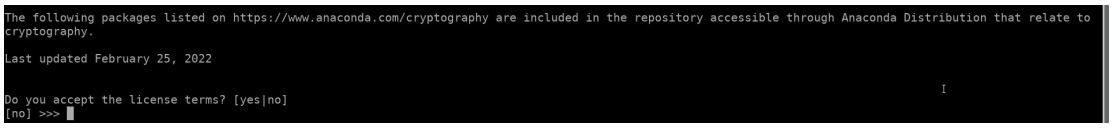


图 1.70 输入 yes

（6）然后安装脚本会提醒安装 Anaconda3 到家目录，此时按回车键确认。

图 1.71 输入回车

（7）然后安装脚本会提示是否初始化 Anaconda3 时，输入 yes，然后按回车键。

图 1.72 输入 yes 按回车

（8）当看到终端中出现如下打印时，说明已经成功安装 Anaconda3 了。

图 1.73 成功安装 anaconda

（9）之后进入 pycharm 打开终端

输入以下命令：

a. Conda 设置清华源

conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/free

conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/conda-forge

conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/bioconda

conda config --set show_channel_urls yes

b. pip 设置清华源

pip config set global.index-url https://pypi.tuna.tsinghua.edu.cn/simple

c. 安装项目环境

conda create --name tcp python=3.10

conda activate tcp

conda install pyaudio

conda install ffmpeg

conda install pyqt

pip install ffmpy

pip install opencv-python

pip install loguru

pip install gmssl

二、本地 PC 端配置

2.1 安装环境

在自己 pycharm 终端输入以下命令

```
conda create --name tcp python=3.10
conda activate tcp
conda install pyaudio
conda install ffmpeg
conda install pyqt
pip install ffmpy
pip install opencv-python
pip install loguru
pip install gmssl
pip install psycopg2
```

三、代码部分

3.1 两端的目录结构

左边是客户端目录结构（香橙派），右边是本地目录结构（自己电脑）

```
wzr2023212054/          # 根目录
├── assets/             # 资源目录
│   └── records/        # 记录存储目录
├── Core/               # 核心业务逻辑目录
│   ├── config.py       # 配置文件
│   ├── TCPClient.py    # TCP客户端实现
│   └── translate_ui.py # UI文件转换工具
├── ui/                 # 界面相关目录
│   ├── Client.ui       # 客户端可视化界面文件
│   └── icon.png        # 界面图标资源
├── utils/              # 工具类目录
│   ├── Encode_Decode.py # 编码解码工具
│   ├── message.py      # 消息处理工具
│   ├── SendVideo.py    # 视频发送工具
│   ├── tcp.py          # TCP通用工具
│   └── ZUC.py          # ZUC算法实现
└── __main__.py         # 程序入口文件
```

图 1.74 客户端目录结构

```
wzr2023212054/          # 根目录
├── assets/             # 资源目录
│   └── records/        # 记录存储目录
├── Core/               # 核心业务逻辑目录
│   ├── config.py       # 配置文件
│   ├── TCPServer.py    # TCP服务端实现
│   └── translate_ui.py # UI文件转换工具
├── ui/                 # 界面相关目录
│   ├── Server.ui       # 服务端可视化界面文件
│   └── icon.png        # 界面图标资源
├── utils/              # 工具类目录
│   ├── Encode_Decode.py # 编码解码工具
│   ├── message.py      # 消息处理工具
│   ├── SendVideo.py    # 视频发送工具
│   ├── tcp.py          # TCP通用工具
│   ├── ZUC.py          # ZUC算法实现
│   ├── Camera_Server.py # 摄像头服务工具
│   └── database.py     # 数据库操作工具
└── __main__.py         # 程序入口文件
```

图 1.75 服务器目录结构

3.2 运行方法

（1）首先本地先连接数据库
（2）运行服务器的__main__.py
（3）运行客户单的__main__.py
（4）如果你的香橙派内存不大可能会出现栈溢出的问题，建议在终端里输入ulimit -s 32768

3.3 公共模块（两端都需要）

（1）config.py

```
from loguru import logger
path_database = '../assets/records.db'
path_video = '../assets/records'
path_logfile = "../assets/runtime.log"
logger.add(path_logfile, rotation="500 MB", encoding='utf-8', level="INFO")
path_configfile = "../assets/config.ini"
store_ip = True
server_port = 17781
window_title = "2023212054 温自然"
server_recv_buffersize = 10240
```

（2）translate_ui.py

```
import os
import sys
from loguru import logger
def translate_ui(ui_file):
    ui_file_abs = os.path.abspath(ui_file)
    if not os.path.exists(ui_file_abs):
    logger.error(f"Translate ui | 源文件不存在：{ui_file_abs}")
            return False

    base_name = os.path.basename(ui_file_abs)
    pure_name = os.path.splitext(base_name)[0]
    dir_abs = os.path.dirname(ui_file_abs)
    source = ui_file_abs
    dist = os.path.join(dir_abs, pure_name + '.py')

    os.makedirs(dir_abs, exist_ok=True)
    # 调试日志
    logger.debug("Translate ui | base name: {}".format(base_name))
    logger.debug("Translate ui | pure name: {}".format(pure_name))
    logger.debug("Translate ui | dir: {}".format(dir_abs))
    logger.debug("Translate ui | source: {}".format(source))
    logger.debug("Translate ui | dist: {}".format(dist))

    cmd = f'python -m PyQt5.uic.pyuic "{source}" -o "{dist}"'
    logger.info("Translate ui | {}".format(cmd))

   # 捕获执行结果
    ret_code = os.system(cmd)
    if ret_code == 0:
        logger.success(f"Translate ui | 成功生成：{dist}")
```

```
            return True
        else:
            logger.error(f"Translate ui | 生成失败，返回码：{ret_code}")
            return False
```

（3）Encode_Decode.py

```
from gmssl import sm4
from utils.ZUC import ZUC
SM4_KEY = b'1234567890abcdef'  # SM4 密钥

key=[0x3d,0x4c,0x4b,0xe9,0x6a,0x82,0xfd,0xae,0xb5,0x8f,0x64,0x1d,0xb1,0x7
b,0x45,0x5b]  # ZUC 加密密钥
iv=[0x84,0x31,0x9a,0xa8,0xde,0x69,0x15,0xca,0x1f,0x6b,0xda,0x6b,0xfb,0xd8,
0xc7,0x66]    # ZUC 加密初始向量
class DataHandler:
    SUPPORTED_ALGOS = ["无加密", "SM4 加密","ZUC 加密"]
    def __init__(self) -> None:
        pass

    @staticmethod
    def encode(send_data: bytes, encrypt_algo: str = "SM4 加密") -> bytes:
        if encrypt_algo not in ["无加密", "SM4 加密","ZUC 加密"]:
            raise ValueError("仅支持无加密/SM4 加密/ZUC 加密")

        if encrypt_algo == "无加密":
            encrypted_data = send_data
        elif encrypt_algo == "SM4 加密":
            crypt_sm4 = sm4.CryptSM4()
            crypt_sm4.set_key(SM4_KEY, sm4.SM4_ENCRYPT)  # 设置
密钥和加密模式
            encrypted_data = crypt_sm4.crypt_ecb(send_data)  # ECB 模式
加密
        elif encrypt_algo == "ZUC 加密":
            zuc_crypt = ZUC()
            zuc_crypt.init(key, iv)
            encrypted_data = zuc_crypt.encrypt(send_data)

        return encrypted_data

    @staticmethod
    def decode(receive_data: bytes, encrypt_algo: str = "SM4 加密") -> byt
es:
        if encrypt_algo not in ["无加密", "SM4 加密","ZUC 加密"]:
            raise ValueError("仅支持无加密/SM4 加密/ZUC 加密")
```

```
            if encrypt_algo == "无加密":
                decrypted_data = receive_data
            elif encrypt_algo == "SM4 加密":
                crypt_sm4 = sm4.CryptSM4()
                crypt_sm4.set_key(SM4_KEY, sm4.SM4_DECRYPT)
                decrypted_data = crypt_sm4.crypt_ecb(receive_data)
            elif encrypt_algo == "ZUC 加密":
                zuc_decrypt = ZUC()
                zuc_decrypt.init(key, iv)
                decrypted_data = zuc_decrypt.decrypt(receive_data)
            return decrypted_data
```

（4）message.py

```
"""
用于处理消息传输的模块
"""
import socket
import sys
from PyQt5.QtCore import QThread, QMutex, pyqtSignal
from utils.tcp import Server, Client
from loguru import logger
import struct

class MessageServer(Server, QThread):
    """
    消息服务器，用于接收信息
    """
    # 用于将数据传递给 UI 线程
    callback_msg = pyqtSignal(object)
    callback_video = pyqtSignal(object)

    def __init__(self):
        Server.__init__(self)
        QThread.__init__(self)
        self.selected_decrypt = "无加密"

    def emit(self, conn, addr, msg: bytes):
        if msg == b'connect':
            self.callback_msg.emit({
                'conn': conn,
                'addr': addr,
                'data': 'connect'
            })
        else:
            self.callback_video.emit({
```

```
                    'conn': conn,
                    'addr': addr,
                    'data': msg
                })

    def run(self):
        """线程的入口：启动 TCP 监听"""
        try:
            self.listen()
        except socket.error:
            logger.error("[服务器]: 打开套接字出错")
            sys.exit(1)

    def accept_connect(self, addr):
        super().accept_connect(addr)

    def send_confirm(self, conn):
        """发送 confirm 控制指令，无加密"""
        try:
            confirm_data = b'confirm'
            conn.sendall(struct.pack('<L', len(confirm_data)) + confirm_dat
a)
        except Exception as e:
            logger.error(f"发送 confirm 失败: {e}")


class MessageClient(Client, QThread):
    """
    消息客户端，用于发送消息
    """
    server_confirm_signal = pyqtSignal(bool)    # 服务端确认信号
    def __init__(self, target_ip):
        Client.__init__(self)
        QThread.__init__(self)
        self.mutex = QMutex()    # 互斥锁：保证 sendall 线程安全
        self.target_ip = target_ip
        self.encrypt_algo = "无加密"
        self.is_server_confirmed = False        # 是否收到服务端的 confir
m

    def quit(self):
        self.close_connect()

    def set_server_confirmed(self, confirmed: bool):
```

```
            self.is_server_confirmed = confirmed
            self.server_confirm_signal.emit(confirmed)

        def send_message(self, data: str):
            """发送控制消息（connect、close）"""
            self.mutex.lock()
            self.sendall(data.encode('utf-8'),is_control=True)
            self.mutex.unlock()

        def send_video(self, data: bytes):
            self.mutex.lock()
            self.sendall(data, is_control=False)
            self.mutex.unlock()

        def run(self):
            """线程入口：连接服务端，等待确认"""
            try:
                self.get_connect(self.target_ip)
                import time
                time.sleep(1)
                self.send_message('connect')
                data_buffer = b''
                header_size = struct.calcsize('<L')  #4
                while not self.closed and self.is_connected:
                    try:
                        # 接收服务端的 confirm 指令
                        while len(data_buffer) < header_size:
                            recv_data = self.sock.recv(1024)
                            if not recv_data:  # 服务端断开
                                self.close_connect()
                                break
                            data_buffer += recv_data
                        if not self.is_connected:
                            break
                        packed_size = data_buffer[:header_size]
                        data_buffer = data_buffer[header_size:]
                        msg_size = struct.unpack('<L', packed_size)[0]  #返
回的是元组（msg_size, ）
                        while len(data_buffer) < msg_size:
                            data_buffer += self.sock.recv(1024)
                        zipdata = data_buffer[:msg_size]
                        data_buffer = data_buffer[msg_size:]

                        # 如果收到服务端的 confirm，标记确认
```

```
                        if zipdata == b'confirm':
                            self.set_server_confirmed(True)
                            logger.info("[客户端] 收到服务端连接确认，开始发送视频数据")
                    except:
                        break
            except:
                pass
```

（5）tcp.py

```
from abc import abstractmethod
import socket
import struct
import threading
from queue import Queue
from utils.Encode_Decode import DataHandler
from Core import config
from loguru import logger

class Server:

    def __init__(self):
        self.address = ('', config.server_port)    # 绑定本机所有 IP+配置的端口
        self.sock = None    # 监听 socket
        self.conn_map = {}    # 保存已连接的客户端：{ (ip,port): conn 套接字 }
        self.cipher_queue = Queue()    # 解密队列：缓存待解密的密文数据

        # 启动独立的解密线程
        self.decrypt_thread = threading.Thread(target=self._decrypt_worker, daemon=True)
        self.decrypt_thread.start()
    @abstractmethod
    def emit(self, conn, addr: tuple, msg: bytes):
        """
        信号发送函数
        """
        pass

    def accept_connect(self, addr):
        """接受客户端连接，启动独立线程处理该客户端的数据接收"""

        logger.info('[服务器]: 接受来自{}:{}的连接'.format(addr[0], addr[1]))
        conn = self.conn_map[addr]    # 从 conn_map 取出该客户端的套接字
        link = threading.Thread(target=self.connect_thread, args=(conn, addr))
```

```
            link.setDaemon(True)
            link.start()

        def close_connect(self, addr):
            """关闭指定客户端的连接"""

            conn = self.conn_map.get(addr, None)
            if conn:
                self.conn_map.pop(addr)
                conn.close()      # 关闭套接字
        def connect_thread(self, conn, addr):
            """单个客户端的数据接收线程：解决 TCP 粘包问题
            粘包原因：TCP 是流式协议，多次发送的数据包可能合并传输
            解决方法：自定义协议——先发送 4 字节的消息长度，再发送实际数据
"""
            conn_map = self.conn_map

            data_buffer = b''    # 缓存未处理的字节数据
            header_size = struct.calcsize('<L')

            while conn_map.get(addr):
                try:
                    # 使用滑动窗口不断接收数据
                    while len(data_buffer) < header_size:
                        recv_data = conn.recv(4096)
                        if not recv_data:  # 客户端主动关闭连接，recv 返回空
                            logger.info(f"[接收线程] {addr} 客户端主动断连")
                            self.close_connect(addr)  # 主动关闭服务端连接
                            break
                        data_buffer += recv_data

                    if not conn_map.get(addr):  # 已断连则退出循环
                        break

                    packed_size = data_buffer[:header_size]    # 前 4 字节是长度
                    data_buffer = data_buffer[header_size:]
                    msg_size = struct.unpack('<L', packed_size)[0]  # 解析出实
际消息长度

                    while len(data_buffer) < msg_size:
                        recv_data = conn.recv(4096)
                        if not recv_data:
                            logger.info(f"[接收线程] {addr} 客户端主动断连")
                            self.close_connect(addr)
```

```
                        break
                    data_buffer += recv_data
                if not conn_map.get(addr):
                    break

                zipdata = data_buffer[:msg_size]    # 完整的密文数据


                self.cipher_queue.put((conn, addr, zipdata))
                data_buffer = data_buffer[msg_size:]    # 剩余数据留到下一
轮处理
            except Exception as e:
                logger.warning(f"[接收线程] {addr} 接收异常: {e}")
                break

        if addr in conn_map:
            conn_map.pop(addr)
        logger.info('[服务器]: {}:{}已断开连接'.format(addr[0], addr[1]))
        try:
            conn.close()
        except:
            pass

    def _decrypt_worker(self):
        """独立解密线程：消费队列，解密成功才回调"""
        while True:
            try:
                conn, addr, zipdata = self.cipher_queue.get()
                if addr not in self.conn_map:  # 客户端已断开则跳过
                    self.cipher_queue.task_done()
                    continue

                decrypt_algo = self.selected_decrypt if hasattr(self, 'selected_d
ecrypt') else "无加密"
                video_data = zipdata
                if zipdata in [b'connect', b'confirm']:
                    # 控制指令直接跳过解密
                    pass
                else:
                    if decrypt_algo != "无加密":
                        try:
                            video_data = DataHandler.decode(zipdata, decry
pt_algo)
                        except Exception as e:
```

```
                                    logger.warning(f"[解密线程] {addr} 解密失败:
{e}")


                    self.emit(conn, addr, video_data)  # 解密后回调
                    self.cipher_queue.task_done()  # 标记队列任务完成
                except Exception as e:
                    logger.error(f"[解密线程] 异常: {e}")
                    self.cipher_queue.task_done()


    def listen(self):
        """
        绑定端口，时刻监听新的连接
        """
        # tcp sock
        # 创建 TCP 套接字（AF_INET=IPv4，SOCK_STREAM=TCP）
        sock = self.sock = socket.socket(socket.AF_INET, socket.SOCK_STRE
AM)
        sock.bind(self.address)  # 绑定端口
        sock.listen(3)  # 最大挂起连接数 3
        logger.info('[服务器]: 端口{}绑定成功!'.format(config.server_port))

        while True:
            # 接受新的连接
            conn, addr = sock.accept()
            self.conn_map[addr] = conn
            self.emit(conn, addr, b'connect')    # 触发 connect 事件


class Client:
    """
    客户端：只发送数据
    """

    def __init__(self) -> None:
        self.sock = None
        self.is_connected = False    # 是否已连接服务端
        self.closed = False
        self.encrypt_buffer = bytearray(320 * 240 * 3)

    def get_connect(self, target_ip: str):
        """
        连接到指定 ip 的服务器
```

```
        """
        logger.info('[客户端]: 即将连接到' + target_ip)
        try:
            # 与服务端建立 socket 连接
            self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            self.sock.connect((target_ip, config.server_port))
            self.is_connected = True
            self.closed = False
            logger.info('[客户端]: 已与服务器建立连接，等待服务器确认中...')
        except:
            logger.error('[客户端]: 连接失败，请重试')
            self.close_connect()

    def close_connect(self):
        """关闭连接"""

        if self.sock is None:
            logger.warning("[客户端] socket 已为空，无需关闭")
            self.is_connected = False
            self.closed = True
            return

        try:
            sock_valid = False
            try:
                sock_valid = (self.sock.fileno() != -1)
            except (OSError, socket.error):
                sock_valid = False

            if sock_valid:
                self.sock.shutdown(socket.SHUT_RDWR)
                self.sock.close()
        except (OSError, socket.error) as e:
            logger.warning(f"[客户端] 关闭 socket 失败（可能已断开）: {e}")
        finally:
            self.sock = None
            self.is_connected = False
            self.closed = True

        logger.info("[客户端] 已优雅关闭连接")

    def sendall(self, video_data, is_control=False):
        """
        发送数据：解决粘包问题（先送长度，再送数据）
```

```
        is_control：是否是控制指令（connect、close、confirm），控制指令不加密
        """

        if self.sock is None:
            logger.warning("[客户端] socket 已为空，跳过本次数据发送")
            return

        try:
            if self.sock.fileno() == -1:
                logger.warning("[客户端] socket 已关闭，跳过本次数据发送")
                return
        except (OSError, socket.error):
            logger.warning("[客户端] socket 失效，跳过本次数据发送")
            self.sock = None
            self.is_connected = False
            return

        senddata = video_data


        if not is_control and hasattr(self, 'encrypt_algo') and self.encrypt_algo != "无加密":
            senddata = DataHandler.encode(senddata, self.encrypt_algo)


        try:
            self.sock.sendall(struct.pack('<L', len(senddata)) + senddata)
        except:
            logger.debug('[客户端]: 数据发送失败')
```

（6）ZUC.py

```
import math
class ZUC:

    S0=[[0x3e, 0x72, 0x5b, 0x47, 0xca, 0xe0, 0x00, 0x33, 0x04, 0xd1, 0x54, 0x98, 0x09, 0xb9, 0x6d, 0xcb],
        [0x7b, 0x1b, 0xf9, 0x32, 0xaf, 0x9d, 0x6a, 0xa5, 0xb8, 0x2d, 0xfc, 0x1d, 0x08, 0x53, 0x03, 0x90],
        [0x4d, 0x4e, 0x84, 0x99, 0xe4, 0xce, 0xd9, 0x91, 0xdd, 0xb6, 0x85, 0x48, 0x8b, 0x29, 0x6e, 0xac],
        [0xcd, 0xc1, 0xf8, 0x1e, 0x73, 0x43, 0x69, 0xc6, 0xb5, 0xbd, 0xfd, 0x39, 0x63, 0x20, 0xd4, 0x38],
        [0x76, 0x7d, 0xb2, 0xa7, 0xcf, 0xed, 0x57, 0xc5, 0xf3, 0x2c, 0xbb, 0x14, 0x21, 0x06, 0x55, 0x9b],
```

[0xe3, 0xef, 0x5e, 0x31, 0x4f, 0x7f, 0x5a, 0xa4, 0x0d, 0x82, 0x51, 0x49, 0x5f, 0xba, 0x58, 0x1c],
[0x4a, 0x16, 0xd5, 0x17, 0xa8, 0x92, 0x24, 0x1f, 0x8c, 0xff, 0xd8, 0xae, 0x2e, 0x01, 0xd3, 0xad],
[0x3b, 0x4b, 0xda, 0x46, 0xeb, 0xc9, 0xde, 0x9a, 0x8f, 0x87, 0xd7, 0x3a, 0x80, 0x6f, 0x2f, 0xc8],
[0xb1, 0xb4, 0x37, 0xf7, 0x0a, 0x22, 0x13, 0x28, 0x7c, 0xcc, 0x3c, 0x89, 0xc7, 0xc3, 0x96, 0x56],
[0x07, 0xbf, 0x7e, 0xf0, 0x0b, 0x2b, 0x97, 0x52, 0x35, 0x41, 0x79, 0x61, 0xa6, 0x4c, 0x10, 0xfe],
[0xbc, 0x26, 0x95, 0x88, 0x8a, 0xb0, 0xa3, 0xfb, 0xc0, 0x18, 0x94, 0xf2, 0xe1, 0xe5, 0xe9, 0x5d],
[0xd0, 0xdc, 0x11, 0x66, 0x64, 0x5c, 0xec, 0x59, 0x42, 0x75, 0x12, 0xf5, 0x74, 0x9c, 0xaa, 0x23],
[0x0e, 0x86, 0xab, 0xbe, 0x2a, 0x02, 0xe7, 0x67, 0xe6, 0x44, 0xa2, 0x6c, 0xc2, 0x93, 0x9f, 0xf1],
[0xf6, 0xfa, 0x36, 0xd2, 0x50, 0x68, 0x9e, 0x62, 0x71, 0x15, 0x3d, 0xd6, 0x40, 0xc4, 0xe2, 0x0f],
[0x8e, 0x83, 0x77, 0x6b, 0x25, 0x05, 0x3f, 0x0c, 0x30, 0xea, 0x70, 0xb7, 0xa1, 0xe8, 0xa9, 0x65],
[0x8d, 0x27, 0x1a, 0xdb, 0x81, 0xb3, 0xa0, 0xf4, 0x45, 0x7a, 0x19, 0xdf, 0xee, 0x78, 0x34, 0x60]]

S1=[[0x55, 0xc2, 0x63, 0x71, 0x3b, 0xc8, 0x47, 0x86, 0x9f, 0x3c, 0xda, 0x5b, 0x29, 0xaa, 0xfd, 0x77],
[0x8c, 0xc5, 0x94, 0x0c, 0xa6, 0x1a, 0x13, 0x00, 0xe3, 0xa8, 0x16, 0x72, 0x40, 0xf9, 0xf8, 0x42],
[0x44, 0x26, 0x68, 0x96, 0x81, 0xd9, 0x45, 0x3e, 0x10, 0x76, 0xc6, 0xa7, 0x8b, 0x39, 0x43, 0xe1],
[0x3a, 0xb5, 0x56, 0x2a, 0xc0, 0x6d, 0xb3, 0x05, 0x22, 0x66, 0xbf, 0xdc, 0x0b, 0xfa, 0x62, 0x48],
[0xdd, 0x20, 0x11, 0x06, 0x36, 0xc9, 0xc1, 0xcf, 0xf6, 0x27, 0x52, 0xbb, 0x69, 0xf5, 0xd4, 0x87],
[0x7f, 0x84, 0x4c, 0xd2, 0x9c, 0x57, 0xa4, 0xbc, 0x4f, 0x9a, 0xdf, 0xfe, 0xd6, 0x8d, 0x7a, 0xeb],
[0x2b, 0x53, 0xd8, 0x5c, 0xa1, 0x14, 0x17, 0xfb, 0x23, 0xd5, 0x7d, 0x30, 0x67, 0x73, 0x08, 0x09],
[0xee, 0xb7, 0x70, 0x3f, 0x61, 0xb2, 0x19, 0x8e, 0x4e, 0xe5, 0x4b, 0x93, 0x8f, 0x5d, 0xdb, 0xa9],
[0xad, 0xf1, 0xae, 0x2e, 0xcb, 0x0d, 0xfc, 0xf4, 0x2d, 0x46, 0x6e, 0x1d, 0x97, 0xe8, 0xd1, 0xe9],
[0x4d, 0x37, 0xa5, 0x75, 0x5e, 0x83, 0x9e, 0xab, 0x82, 0x9d, 0xb9, 0x1c, 0xe0, 0xcd, 0x49, 0x89],
[0x01, 0xb6, 0xbd, 0x58, 0x24, 0xa2, 0x5f, 0x38, 0x78, 0x99, 0x15,

```
0x90, 0x50, 0xb8, 0x95, 0xe4],
        [0xd0, 0x91, 0xc7, 0xce, 0xed, 0x0f, 0xb4, 0x6f, 0xa0, 0xcc, 0xf0, 0
x02, 0x4a, 0x79, 0xc3, 0xde],
        [0xa3, 0xef, 0xea, 0x51, 0xe6, 0x6b, 0x18, 0xec, 0x1b, 0x2c, 0x80, 0
xf7, 0x74, 0xe7, 0xff, 0x21],
        [0x5a, 0x6a, 0x54, 0x1e, 0x41, 0x31, 0x92, 0x35, 0xc4, 0x33, 0x07,
0x0a, 0xba, 0x7e, 0x0e, 0x34],
        [0x88, 0xb1, 0x98, 0x7c, 0xf3, 0x3d, 0x60, 0x6c, 0x7b, 0xca, 0xd3,
0x1f, 0x32, 0x65, 0x04, 0x28],
        [0x64, 0xbe, 0x85, 0x9b, 0x2f, 0x59, 0x8a, 0xd7, 0xb0, 0x25, 0xac,
0xaf, 0x12, 0x03, 0xe2, 0xf2]]

    D=[ 0x44d7, 0x26bc, 0x626b, 0x135e, 0x5789, 0x35e2, 0x7135, 0x09af,
        0x4d78, 0x2f13, 0x6bc4, 0x1af1, 0x5e26, 0x3c4d, 0x789a, 0x47ac]

    def __init__(self,keylen=6):

        self.S=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
        self.Key_result=[]
        self.keylen=keylen
        self.X=[0,0,0,0]
        self.R1=0
        self.R2=0
        self.W=0
        self.m32=pow(2,32)

    def BitReconstruction(self):
        self.X[0]=self.add2(self.S[15],self.S[14])
        self.X[1]=self.add(self.S[11],self.S[9])
        self.X[2]=self.add(self.S[7],self.S[5])
        self.X[3]=self.add(self.S[2],self.S[0])


    def LFSRWithInitMode(self,u):
        v=(2**15*self.S[15]+2**17*self.S[13]+2**21*self.S[10]+2**20*self.S[4]
+(1+2**8)*self.S[0])%(2**31-1)
        self.S.append((v+u)%(2**31-1))
        if self.S[16]==0:
            self.S[16]=2**31-1
        self.S.pop(0)

    def LFSRWithWorkMode(self):
        self.S.append((2 ** 15 * self.S[15] + 2 ** 17 * self.S[13] + 2 ** 21
 * self.S[10] + 2 ** 20 * self.S[4] + (1 + 2 ** 8) * self.S[0]) % (2 ** 31 -
```

```
1))
        if self.S[16] == 0:
            self.S[16] = 2 ** 31 - 1
        self.S.pop(0)

    def F(self,X0,X1,X2):
        self.W=self.mod32(X0 ^ self.R1,self.R2)
        W1=self.mod32(self.R1,X1)
        W2=self.R2 ^ X2

        self.R1=self.S_(self.L1(((W1<<16) | (W2>>16)) & 0xffffffff))
        self.R2=self.S_(self.L2(((W2<<16) | (W1>>16)) & 0xffffffff))

    def L1(self,X):
        return X ^ self.rot(X ,2) ^ self.rot(X ,10)^ self.rot(X ,18) ^ self.rot(X
,24)& 0xffffffff

    def L2(self,X):
        return X ^ self.rot(X ,8) ^ self.rot(X ,14) ^ self.rot(X ,22) ^ self.rot
(X ,30)& 0xffffffff

    def S_(self,X):
        bit8=[0,0,0,0]
        result=[0,0,0,0]
        bit8[0]=X>>24
        bit8[1]=(X>>16)&0xff
        bit8[2]=(X>>8)&0xff
        bit8[3]=X&0xff
        for i in range(4):
            row = bit8[i] >> 4
            nuw = bit8[i] & 0xf
            if i == 0 or i== 2:
                result[i]=self.S0[row][nuw]
            else:
                result[i]=self.S1[row][nuw]
        ans = (result[0] << 24) | (result[1] << 16) | (result[2] << 8) | result
[3]
        return ans

    def rot(self,X,i):
      return ((X<<i)& 0xffffffff)|(X>>(32-i))

    def mod32(self,R,X):
        return (R + X)%self.m32
```

```
def H(self,X):
#高 16
    bits=(X>>15) & 0xffff
    return bits

def L(self,X):
#16
    bits = X & 0xffff
    return bits

def add(self,W1,W2):
    W1l=self.L(W1)<<16
    W2h=self.H(W2)
    all=W1l|W2h
    return all

def add2(self,W1,W2):
    W1h = self.H(W1)<<16
    W2l = self.L(W2)
    all_value=W1h|W2l
    return all_value

def init(self,key,iv):
    for i in range(16):
        self.S[i] = (key[i] << 23) | (self.D[i] << 8) | iv[i]
    #print('初始化之前：')
    #self.print_S()
    for i in range(32):
        self.BitReconstruction()
        self.F(self.X[0], self.X[1], self.X[2])
        self.LFSRWithInitMode(self.W >> 1)
        #self.print_status()

def calc(self):
    self.BitReconstruction()
    self.F(self.X[0], self.X[1], self.X[2])
    self.LFSRWithWorkMode()
    for i in range(self.keylen):
        self.BitReconstruction()
        self.F(self.X[0], self.X[1], self.X[2])
        self.Key_result.append(self.W^self.X[3])
        self.LFSRWithWorkMode()
        #self.print_status()
```

```
    def print_S(self):
        for i in range(16):
            if i==8:
                print()
            print('S'+str(i)+':\033[1;32m'+hex(self.S[i]).replace('0x','')+'\033[0m ',
end='')
        print()

    def print_status(self):
        for i in range(4):
            print('X' + str(i) + ':\033[1;31m' + hex(self.X[i]).replace('0x', '') +
'\033[0m ', end='')
        print('R1' +  ':\033[1;34m' + hex(self.R1).replace('0x', '') + '\033[0m ',
end='')
        print('R2' + ':\033[1;34m' + hex(self.R2).replace('0x', '') + '\033[0m ', e
nd='')
        print('W'  + ':\033[1;35m' + hex(self.W).replace('0x', '') + '\033[0m ', e
nd='')
        print('S15' + ':\033[1;36m' + hex(self.S[15]).replace('0x', '') + '\033[0m
', end='')
        print()

    def print_key(self):
        for i in range(self.keylen):
            print('KEY' + str(i) + ':\033[1;36m' + hex(self.Key_result[i]).replac
e('0x', '') + '\033[0m ', end='')

    def generate_keystream(self, length: int) -> list[int]:
        """生成密钥流：输出 length 个 32 位密钥字"""
        if length < 0:
            raise ValueError("密钥流长度不能为负数")
        self.keylen = length
        self.Key_result = []
        self.BitReconstruction()
        self.F(self.X[0], self.X[1], self.X[2])
        self.LFSRWithWorkMode()
        for i in range(self.keylen):
            self.BitReconstruction()
            self.F(self.X[0], self.X[1], self.X[2])
            self.Key_result.append(self.W ^ self.X[3])
            self.LFSRWithWorkMode()
        return [k & 0xFFFFFFFF for k in self.Key_result]
```

```
    def encrypt(self, plaintext: bytes) -> bytes:
        """加密：明文与密钥流逐字节异或"""
        word_count = (len(plaintext) + 3) // 4
        keystream = self.generate_keystream(word_count)
        keystream_bytes = b''.join([w.to_bytes(4, 'big') for w in keystream])
        return bytes([p ^ k for p, k in zip(plaintext, keystream_bytes[:len(plain
text)])])

    def decrypt(self, ciphertext: bytes) -> bytes:
        """解密：复用加密逻辑"""
        return self.encrypt(ciphertext)
```

3.4 客户端模块（香橙派端需要）

（1）TCPClient.py

```
"""
用于处理数据发送端的模块
"""
import os
import sys

if os.environ.get('DISPLAY') is None:
    os.environ['QT_QPA_PLATFORM'] = 'xcb'
os.environ['QT_LOGGING_RULES'] = 'qt.png.warning=false'

from Core import config
from loguru import logger
from utils.SendVideo import VideoSender
from ui.Client import Ui_MainWindow
import configparser
from PyQt5.QtGui import QPixmap, QImage, QIcon
from PyQt5.QtWidgets import QApplication, QMainWindow, QMessageBox,QCo
mboBox, QLabel
from utils.message import MessageClient

class Camera(QMainWindow, Ui_MainWindow):
    """
    发送端类，将采集到的摄像头信息发送到服务器
    """
    def __init__(self, parent=None):
        # 初始化界面
        Ui_MainWindow.__init__(self)
        QMainWindow.__init__(self, parent)
        self.setupUi(self)
        self.setWindowTitle("客户端 - " + config.window_title)
        self.setMinimumSize(180, 106)
        current_file = os.path.abspath(__file__)
        current_dir = os.path.dirname(current_file)
        project_root = os.path.dirname(current_dir)
        icon_path = os.path.join(project_root, "ui", "icon.png")
        self.setWindowIcon(QIcon(icon_path))
        self.connect_button.setEnabled(True)
        self.close_button.setDisabled(True)
        self.ip_text.setFocus()

        self.connect_button.clicked.connect(self.con_connect_btn)
```

```
        self.ip_text.returnPressed.connect(self.con_connect_btn)
        self.close_button.clicked.connect(self.close_all_threads)

        self.encrypt_combo = QComboBox()
        self.encrypt_combo.addItems(["无加密", "SM4 加密","ZUC 加密"])
        self.controlLayout.insertWidget(2, self.encrypt_combo)
        self.selected_encrypt = "无加密"
        self.encrypt_combo.currentTextChanged.connect(self.on_encrypt_changed)
        logger.debug("读取配置文件")
        if config.store_ip:
            configreader = configparser.ConfigParser()
            try:
                configreader.read(config.path_configfile)
                self.ip_text.setText(configreader['DEFAULT']['ip'])
            except:
                pass

        self.video_sender = None
        self.msg_client = None

    def on_encrypt_changed(self, text):
        self.selected_encrypt = text
        logger.debug(f"选择加密方式：{self.selected_encrypt}")
    def close_all_threads(self):
        """关闭所有线程视频发送+消息客户端"""
        logger.debug("关闭所有进程")

        self.close_video_sender()
        self.close_msg_client()

        # 清空视频显示，更新状态
        self.camera_label.clear()
        self.camera_label.repaint()
        self.status_label.setText("状态：未连接")
        self.status_label.setStyleSheet("color: #6c757d; font-style: italic;")
        self.connect_button.setEnabled(True)
        self.close_button.setDisabled(True)

    def con_connect_btn(self):
        """启动客户端+视频"""
        self.status_label.setText("状态：已连接")
        self.status_label.setStyleSheet("color: #198754; font-style: normal;")
        self.connect_button.setDisabled(True)
        self.close_button.setEnabled(True)
```

```
        # 启动消息客户端
        self.run_msg_client()
        # 启动视频发送
        self.run_video_sender()

    def close_msg_client(self):
        """关闭消息客户端"""
        logger.debug("关闭消息客户端")
        if not self.msg_client:
            return

        try:
            if self.msg_client.is_connected and self.msg_client.sock is not Non
e:
                self.msg_client.send_message('close')
        except Exception as e:
            logger.warning(f"发送 close 指令失败: {e}")

        try:
            self.msg_client.quit()
        except Exception as e:
            logger.warning(f"关闭 msg_client 失败: {e}")
        finally:
            self.msg_client = None

    def close_msg_client(self):
        """关闭消息客户端"""
        logger.debug("关闭消息客户端")
        if self.msg_client:
            self.msg_client.send_message('close')
            self.msg_client.quit()
            self.msg_client = None

    def run_msg_client(self):
        """运行消息客户端"""
        logger.debug("运行消息客户端")
        if self.msg_client:
            return
        self.target_ip = self.ip_text.text()
        try:
            self.msg_client = MessageClient(self.target_ip)
            self.msg_client.encrypt_algo = self.selected_encrypt
            self.msg_client.start()
```

```
        except Exception as e:
            logger.error(f"连接失败：{e}")
            self.status_label.setText("状态：连接失败")
            self.status_label.setStyleSheet("color: #dc3545; font-style: normal;")
            self.connect_button.setEnabled(True)
            self.close_button.setDisabled(True)

    def run_video_sender(self):
        """运行视频发送端"""
        logger.debug("运行视频发送端")
        if self.video_sender:
            return
        self.video_sender = VideoSender()
        self.video_sender.open_server()
        self.video_sender.set_sender(self.msg_client)
        # 绑定本地回显信号
        self.video_sender.callback_local_video.connect(self.show_video_src)
        self.video_sender.start()

    def close_video_sender(self):
        """关闭视频发送端"""
        logger.debug("关闭视频发送端")
        if self.video_sender:
            self.video_sender.quit()
            self.video_sender = None
            self.camera_label.clear()
            self.camera_label.repaint()

    def show_video_src(self, frame):
        """本地回显"""
        if frame is None:
            return
        height, width = frame.shape[:2]
        bytesPerLine = 3 * width
        # 转换为 QImage（BGR→RGB）
        q_img = QImage(frame.data, width, height, bytesPerLine,
                       QImage.Format_RGB888).rgbSwapped()
        pixmap = QPixmap.fromImage(q_img).scaled(
            self.camera_label.size(), transformMode=1)
        self.camera_label.setPixmap(pixmap)
        self.camera_label.repaint()

    def closeEvent(self, event):
        """关闭窗口"""
```

```
            logger.debug("用户尝试关闭窗口")
            reply = QMessageBox.question(
                self, '提示', '{}\n 提醒您：\n 是否要退出程序？'.format(config.wind
ow_title), QMessageBox.Yes | QMessageBox.No, QMessageBox.No)
            if reply == QMessageBox.Yes:
                event.accept()
                self.close_all_threads()
                # 保存配置
                stored_ip = self.ip_text.text()
                if stored_ip and config.store_ip:
                    configwriter = configparser.ConfigParser()
                    configwriter['DEFAULT'] = {'ip': stored_ip}
                    config_dir = os.path.dirname(config.path_configfile)
                    if not os.path.exists(config_dir):
                        os.makedirs(config_dir, exist_ok=True)  # 确保目录存在
                    with open(config.path_configfile, 'w') as configfile:
                        configwriter.write(configfile)
            else:
                event.ignore()
if __name__ == '__main__':
    app = QApplication(sys.argv)
    detector = Camera()
    detector.show()
    sys.exit(app.exec_())
```

（2）Client.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>MainWindow</class>
 <widget class="QMainWindow" name="MainWindow">
  <property name="geometry">
   <rect>
    <x>100</x>
    <y>100</y>
    <width>800</width>
    <height>650</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>音视频传输系统 - 2023212054 温自然</string>
  </property>
  <property name="styleSheet">
   <string notr="true">
    * {
     font-family: "Microsoft YaHei", "SimHei", sans-serif;
```

```
    }
    QMainWindow {
      background: qlineargradient(x1:0, y1:0, x2:0, y2:1, stop:0 #f8f9fa, stop:1 #e9ecef);
      border: none;
    }
    QPushButton {
      background: qlineargradient(x1:0, y1:0, x2:0, y2:1, stop:0 #0d6efd, stop:1 #0b5ed7);
      color: white;
      border-radius: 8px;
      border: none;
      padding: 10px 20px;
      font-size: 14px;
      font-weight: 500;
      letter-spacing: 0.5px;
    }
    QPushButton:hover {
      background: qlineargradient(x1:0, y1:0, x2:0, y2:1, stop:0 #0a58ca, stop:1 #084298); /* 仅加深背景色，删除 transform */
    }
    QPushButton:disabled {
      background: #e9ecef;
      color: #6c757d;
    }
    QLineEdit {
      background-color: white;
      border: 1px solid #ced4da;
      border-radius: 8px;
      padding: 8px 12px;
      font-size: 14px;
      color: #212529;
      selection-background-color: #cce5ff;
      letter-spacing: 0.3px;
    }
    QLineEdit:focus {
      border-color: #0d6efd;
      outline: none;
      border-width: 1.5px;
    }
    QLabel#title_label {
      font-size: 18px;
      font-weight: 600;
      color: #0d6efd;
```

```
  margin-bottom: 10px;
  letter-spacing: 0.5px;
}
QLabel#ip_label {
  font-size: 14px;
  line-height: 1.6;
  color: #333;
  font-weight: 500;
  margin-bottom: 5px;
}
QLabel#status_label {
  font-size: 13px;
  color: #6c757d;
  font-style: normal;
  margin-top: 10px;
  letter-spacing: 0.3px;
}
QLabel#camera_label {
  border: 1px solid #dee2e6;
  border-radius: 8px;
  background-color: white;
  padding: 5px;
  min-width: 640px;
  min-height: 480px;
  alignment: alignCenter;
  color: #6c757d;
  font-size: 14px;
}
QMenuBar {
  background-color: #f8f9fa;
  color: #212529;
  font-size: 14px;
  border-bottom: 1px solid #dee2e6;
}
QMenuBar::item:selected {
  background-color: #e9ecef;
  color: #0d6efd;
}
QMenu {
  background-color: white;
  border: 1px solid #dee2e6;
  border-radius: 8px;
  padding: 5px 0;
}
```

```
      QMenu::item:selected {
       background-color: #e9ecef;
       color: #0d6efd;
      }
      QStatusBar {
       background-color: #f8f9fa;
       color: #6c757d;
       border-top: 1px solid #dee2e6;
       font-size: 12px;
      }
     </string>
    </property>

    <widget class="QWidget" name="centralwidget">
     <layout class="QHBoxLayout" name="mainLayout" spacing="20" margin="25
">
      <item stretch="8">
       <layout class="QVBoxLayout" name="videoLayout">
        <item>
         <widget class="QLabel" name="title_label">
          <property name="text">
           <string>本地摄像头画面</string>
          </property>
         </widget>
        </item>
        <item stretch="1">
         <widget class="QLabel" name="camera_label">
          <property name="text">
           <string>摄像头未启动</string>
          </property>
         </widget>
        </item>
       </layout>
      </item>
      <item stretch="2">
       <layout class="QVBoxLayout" name="controlLayout" spacing="20">
        <item>
         <widget class="QLabel" name="ip_label">
          <property name="text">
           <string>接收端 IP 地址</string>
          </property>
         </widget>
        </item>
        <item>
```

```
  <widget class="QLineEdit" name="ip_text">
   <property name="placeholderText">
    <string>请输入接收端 IP（如 127.0.0.1）</string>
   </property>
   <property name="text">
    <string>127.0.0.1</string>
   </property>
  </widget>
 </item>
 <item>
  <widget class="QPushButton" name="connect_button">
   <property name="text">
    <string>连接接收端</string>
   </property>
  </widget>
 </item>
 <item>
  <widget class="QPushButton" name="close_button">
   <property name="text">
    <string>断开连接</string>
   </property>
   <property name="enabled">
    <bool>false</bool>
   </property>
  </widget>
 </item>
 <item stretch="1">
  <spacer name="verticalSpacer">
   <property name="orientation">
    <enum>Qt::Vertical</enum>
   </property>
   <property name="sizeHint" stdset="0">
    <size>
     <width>20</width>
     <height>40</height>
    </size>
   </property>
  </spacer>
 </item>
 <item>
  <widget class="QLabel" name="status_label">
   <property name="text">
    <string>状态：未连接</string>
   </property>
```

```
          </widget>
         </item>
        </layout>
       </item>
      </layout>
     </widget>

     <widget class="QMenuBar" name="menubar">
      <property name="geometry">
       <rect>
        <x>0</x>
        <y>0</y>
        <width>800</width>
        <height>26</height>
       </rect>
      </property>
      <widget class="QMenu" name="menu_help">
       <property name="title">
        <string>帮助</string>
       </property>
       <addaction name="action_about"/>
      </widget>
      <addaction name="menu_help"/>
     </widget>
     <widget class="QStatusBar" name="statusbar"/>
     <action name="action_about">
      <property name="text">
       <string>关于</string>
      </property>
     </action>

     <resources/>
     <connections/>
    </widget>
</ui>
```

（3）SendVideo.py

```
"""
用于发送视频数据的模块
"""
import sys
import cv2
import numpy as np
from PyQt5.QtCore import QThread, pyqtSignal
from loguru import logger
```

```
class VideoSender(QThread):
    """
    发送视频到服务器，同时发送信号到本地进行回显
    """
    callback_local_video = pyqtSignal(object)    # 本地回显视频的信号

    def __init__(self):
        super().__init__()
        self.msg_client = None
        self.cap = None          # 摄像头
        self.closed = True
        self.frame_buffer = np.zeros((480, 640, 3), dtype=np.uint8)  # 视频帧缓存
        self.jpg_encode_param = [int(cv2.IMWRITE_JPEG_QUALITY), 70]  # JPEG 压缩质量
        self.jpg_buffer = bytearray(1024*1024)                  # JPEG 缓存
    def set_sender(self, msg_client):
        self.msg_client = msg_client

    def quit(self):
        self.close_server()
        self.wait()

    def open_server(self):
        try:
            cap = self.cap = cv2.VideoCapture(0)          # 打开默认摄像头
            cap.set(cv2.CAP_PROP_BUFFERSIZE, 1)  # 摄像头缓存设为 1
            cap.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
            cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)
            cap.set(cv2.CAP_PROP_FPS, 10)          # 帧率 10
        except:
            logger.error("打开摄像头出错")
            sys.exit(1)
        self.closed = False

    def close_server(self):
        self.closed = True
        if self.cap:
            self.cap.release()
            self.cap = None

    def run(self):
        """视频采集线程：持续采集→本地回显→发送服务端"""
        while not self.closed:
```

```
                ret, frame = self.cap.read()
                if not ret:
                    logger.error("读取摄像头出错")
                    break

                self.frame_buffer[:] = frame

                # 本地回显：触发信号，让 UI 显示
                self.callback_local_video.emit(self.frame_buffer)

                if self.msg_client and self.msg_client.is_server_confirmed and self.msg_client.sock and self.msg_client.sock.fileno() != -1:
                    # 压缩为 JPEG
                    ret, encode_data = cv2.imencode('.jpg', self.frame_buffer, self.jpg_encode_param)
                    if ret:
                        encode_bytes = encode_data.tobytes()
                        self.jpg_buffer[:len(encode_bytes)] = encode_bytes
                        # 调用 MessageClient 发送视频数据
                        self.msg_client.send_video(self.jpg_buffer[:len(encode_bytes)])

                self.msleep(30)
```

（4）__main__.py

```
import os
import sys  from loguru import logger
import Core.config as config
from Core.translate_ui import translate_ui
from PyQt5.QtWidgets import QApplication
qt_plugins_path = os.path.join(sys.prefix, "lib", "python3/dist-packages", "PyQt5", "Qt5", "plugins")
os.environ["QT_QPA_PLATFORM_PLUGIN_PATH"] = qt_plugins_path

if __name__ == "__main__":
    current_dir = os.path.dirname(os.path.abspath(__file__))
    client_ui_file_path = os.path.join(current_dir, "ui", "Client.ui")
    translate_ui(client_ui_file_path)
    app = QApplication(sys.argv)
    from Core.TCPClient import Camera
    camera_win = Camera()      # 发送端窗口
    camera_win.show()
    sys.exit(app.exec_())
```

3.5 服务器模块（本地电脑需要）

（1）TCPServer.py

```
"""
用于处理数据接收端的模块
"""
import os
import sys
import struct
os.environ['QT_LOGGING_RULES'] = 'qt.png.warning=false'
from Core import config
import loguru
from ui.Server import Ui_MainWindow
from PyQt5.QtGui import QIcon, QPixmap, QImage
from PyQt5.QtWidgets import QApplication, QMainWindow, QMessageBox, QC
omboBox, QLabel
from utils.message import MessageServer
from utils.database import Database
from utils.Camera_Server import CameraLabel
import numpy as np
import cv2
from PyQt5.QtCore import QMutex, pyqtSignal, Qt
logger = loguru.logger
class TCPServer(QMainWindow, Ui_MainWindow):
    """
    接收端类
    """
    connect_signal = pyqtSignal(tuple)

    def __init__(self, parent=None):
        Ui_MainWindow.__init__(self)
        QMainWindow.__init__(self, parent)
        self.setupUi(self)

        self.setWindowTitle("服务器 - " + config.window_title)
        self.setMinimumSize(800, 650)
        icon_path = r"D:\TCP+SM4+ZUC\wzr2023212054\ui\icon.png"
        self.setWindowIcon(QIcon(icon_path))

        self.decrypt_label = QLabel("解密方式")
        self.controlLayout.addWidget(self.decrypt_label)

        self.decrypt_combo = QComboBox()
        self.decrypt_combo.addItems(["无加密", "SM4 加密","ZUC 加密"])
```

```
        self.controlLayout.addWidget(self.decrypt_combo)

        from PyQt5.QtWidgets import QSpacerItem, QSizePolicy
        self.verticalSpacer = QSpacerItem(20, 40, QSizePolicy.Minimum, QSize
Policy.Expanding)
        self.controlLayout.addItem(self.verticalSpacer)

        self.selected_decrypt = "无加密"
        self.decrypt_combo.currentTextChanged.connect(self.on_decrypt_changed)

        self.cameras = dict()
        self.mutex = QMutex()
        self.database = Database()
        self.msg_server = MessageServer()
        self.msg_server.selected_decrypt = self.selected_decrypt

        self.msg_server.callback_msg.connect(self.show_msg)
        self.msg_server.callback_video.connect(self.show_video)

        self.msg_server.start()
        self.connect_signal.connect(self.handle_connect_popup)

    def on_decrypt_changed(self, text):
        self.selected_decrypt = text
        self.msg_server.selected_decrypt = text
        logger.debug(f"选择解密方式：{self.selected_decrypt}")

    def close_all_threads(self, addr=None):
        self.mutex.lock()
        if addr is None:
            for addr_key in list(self.cameras.keys()):
                camera = self.cameras.get(addr_key)
                if camera:
                    try:
                        camera.end_record()
                    except RuntimeError:
                        pass
                    del self.cameras[addr_key]
        else:
            camera = self.cameras.get(addr)
            if camera:
                try:
                    camera.end_record()
                except RuntimeError:
```

```
                    pass
                if addr in self.cameras:
                    del self.cameras[addr]
        self.mutex.unlock()
        self.camera_label.clear()
        self.camera_label.setText("等待连接...")
        self.camera_label.setAlignment(Qt.AlignCenter)
        self.camera_label.repaint()
        self.videoLayout.update()
        self.mainLayout.update()

    def show_msg(self, data):
        conn = data['conn']
        addr = data['addr']
        data_content = data['data']

        if data_content == 'connect' and addr in self.cameras:
            return
        if data_content == 'connect':
            self.connect_signal.emit(addr)
        elif data_content == 'close':
            self.close_all_threads(addr)

    def handle_connect_popup(self, addr):
        """单独处理连接弹窗，不阻塞视频接收"""
        question_msg = ("{}\n 提醒您：\n 是否接受来至{}:{}的连接".format(co
nfig.window_title, addr[0], addr[1]))
        reply = QMessageBox.question(
            self, '提示', question_msg, QMessageBox.Yes | QMessageBox.No,
QMessageBox.No)
        if reply == QMessageBox.Yes:
            self.msg_server.accept_connect(addr)
            conn = self.msg_server.conn_map.get(addr)
            if conn:
                try:
                    # 发送"confirm"指令告诉客户端：服务端已确认连接
                    self.msg_server.send_confirm(conn)
                except:
                    logger.error(f"给{addr}发送确认消息失败")
            # 连接成功：清空等待文字
            self.camera_label.clear()
            self.camera_label.repaint()
            newCamera = CameraLabel(self)
            self.cameras[addr] = newCamera
```

```
                self.cameras[addr].set_database(self.database)
                self.cameras[addr].set_addr(addr)
                self.cameras[addr].start_record()
                # 刷新布局
                self.videoLayout.update()
                self.mainLayout.update()
                self.adjustSize()
            else:
                self.msg_server.close_connect(addr)

        def show_video(self, data):
            conn = data['conn']
            addr = data['addr']
            frame_data = data['data']

            if self.cameras.get(addr, None) is not None:
                try:
                    nparr = np.frombuffer(frame_data, np.uint8)
                    frame = cv2.imdecode(nparr, cv2.IMREAD_COLOR)
                    if frame is None:
                        return
                    height, width = frame.shape[:2]
                    bytesPerLine = 3 * width
                    q_img = QImage(frame.data, width, height, bytesPerLine,
                                   QImage.Format_RGB888).rgbSwapped()
                    pixmap = QPixmap.fromImage(q_img).scaled(
                        self.camera_label.size(), transformMode=1)
                    self.camera_label.setPixmap(pixmap)
                    self.camera_label.repaint()
                    self.cameras[addr].show_video(frame)
                except Exception as e:
                    self.cameras[addr].force_write_cipher_to_avi(frame_data)
        def closeEvent(self, event):
            reply = QMessageBox.question(
                self, '提示', '{}\n 提醒您：\n 是否要退出程序？'.format(config.wind
ow_title), QMessageBox.Yes | QMessageBox.No, QMessageBox.No)
            if reply == QMessageBox.Yes:
                event.accept()
                self.close_all_threads()
            else:
                event.ignore()

if __name__ == '__main__':
    app = QApplication(sys.argv)
```

```
    master = TCPServer()
    master.show()
    sys.exit(app.exec_())
```

（2）Server.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>MainWindow</class>
 <widget class="QMainWindow" name="MainWindow">
  <property name="geometry">
   <rect>
    <x>100</x>
    <y>100</y>
    <width>800</width>
    <height>650</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>服务器 - 音视频传输系统</string>
  </property>
  <property name="styleSheet">
   <string notr="true">
    * {
     font-family: "Microsoft YaHei", "SimHei", sans-serif;
    }
    QMainWindow {
     background: qlineargradient(x1:0, y1:0, x2:0, y2:1, stop:0 #f8f9fa, stop:1 #
e9ecef);
     border: none;
    }
    QPushButton {
     background: qlineargradient(x1:0, y1:0, x2:0, y2:1, stop:0 #0d6efd, stop:1
#0b5ed7);
     color: white;
     border-radius: 8px;
     border: none;
     padding: 10px 20px;
     font-size: 14px;
     font-weight: 500;
     letter-spacing: 0.5px;
    }
    QPushButton:hover {
     background: qlineargradient(x1:0, y1:0, x2:0, y2:1, stop:0 #0a58ca, stop:1
#084298);
    }
```

```
QPushButton:disabled {
  background: #e9ecef;
  color: #6c757d;
}
QLineEdit {
  background-color: white;
  border: 1px solid #ced4da;
  border-radius: 8px;
  padding: 8px 12px;
  font-size: 14px;
  color: #212529;
  selection-background-color: #cce5ff;
  letter-spacing: 0.3px;
}
QLineEdit:focus {
  border-color: #0d6efd;
  outline: none;
  border-width: 1.5px;
}
QLabel#title_label {
  font-size: 18px;
  font-weight: 600;
  color: #0d6efd;
  margin-bottom: 10px;
  letter-spacing: 0.5px;
}
QLabel#ip_label, QLabel#decrypt_label {
  font-size: 14px;
  line-height: 1.6;
  color: #333;
  font-weight: 500;
  margin-bottom: 5px;
}
QLabel#camera_label {
  border: 1px solid #dee2e6;
  border-radius: 8px;
  background-color: white;
  padding: 5px;
  min-width: 640px;
  min-height: 480px;
  color: #6c757d;
  font-size: 14px;
}
QMenuBar {
```

```
        background-color: #f8f9fa;
        color: #212529;
        font-size: 14px;
        border-bottom: 1px solid #dee2e6;
       }
       QMenuBar::item:selected {
        background-color: #e9ecef;
        color: #0d6efd;
       }
       QMenu {
        background-color: white;
        border: 1px solid #dee2e6;
        border-radius: 8px;
        padding: 5px 0;
       }
       QMenu::item:selected {
        background-color: #e9ecef;
        color: #0d6efd;
       }
       QStatusBar {
        background-color: #f8f9fa;
        color: #6c757d;
        border-top: 1px solid #dee2e6;
        font-size: 12px;
       }
     </string>
    </property>

    <widget class="QWidget" name="centralwidget">
     <layout class="QHBoxLayout" name="mainLayout" spacing="20" margin="25
">
      <item stretch="8">
       <layout class="QVBoxLayout" name="videoLayout">
        <item>
         <widget class="QLabel" name="title_label">
          <property name="text"><string>接收端实时画面</string></property>
         </widget>
        </item>
        <item stretch="1">
         <widget class="QLabel" name="camera_label">
          <property name="text"><string>等待连接...</string></property>
          <property name="alignment"><set>Qt::AlignCenter</set></property>
         </widget>
        </item>
```

```
        </layout>
       </item>
       <item stretch="2">
        <layout class="QVBoxLayout" name="controlLayout" spacing="20">
        </layout>
       </item>
      </layout>
     </widget>

     <widget class="QMenuBar" name="menubar">
      <property name="geometry">
       <rect>
        <x>0</x>
        <y>0</y>
        <width>800</width>
        <height>26</height>
       </rect>
      </property>
      <widget class="QMenu" name="menu_help">
       <property name="title">
        <string>帮助</string>
       </property>
       <addaction name="action_about"/>
      </widget>
      <addaction name="menu_help"/>
     </widget>
     <widget class="QStatusBar" name="statusbar"/>
     <action name="action_about">
      <property name="text">
       <string>关于</string>
      </property>
     </action>

     <resources/>
     <connections/>
    </widget>
</ui>
```

（3）database.py

```
"""
用于处理 openGauss 数据库的模块
"""
import datetime
import os
import time
```

```
from loguru import logger
import psycopg2
from psycopg2 import OperationalError
class Database:
    """
    用于openGauss 数据库操作
    """

    def __init__(self):
        """
        连接openGauss 数据库
        """
        self.db_config = {
            "dbname": "postgres",
            "user": "dboper",
            "password": "dboper@123",
            "host": "192.168.1.129",
            "port": "5432"
        }

        self.CREATE_RECORDS = """
            create table if not exists records(
                id serial PRIMARY KEY,
                ip TEXT NOT NULL,
                port INTEGER NOT NULL,
                start TIMESTAMP NOT NULL,
                "end" TIMESTAMP NOT NULL,
                duration INTERVAL NOT NULL,
                path TEXT NOT NULL
            )
        """
        self.INSERT_RECORD = "insert into records(ip, port, start, \"end\", d
uration, path) values(%s, %s, %s, %s, %s, %s)"
        self.SHOW_RECORDS = "select * from records"
        self.ORDER_BY_ID_ASC = " order by id asc"
        # 连接 openGauss 数据库
        try:
            self.conn = psycopg2.connect(**self.db_config)
            self.cursor = self.conn.cursor()
            logger.info(f"数据库({self.db_config['dbname']})连接成功")

            # 首次连接时创建表
            self.execute(self.CREATE_RECORDS)
            logger.debug("数据表(records)创建/检测成功")
```

```
        except OperationalError as e:
            logger.error(f"数据库连接失败: {str(e)}")
            raise  # 连接失败时终止程序

    def execute(self, sql, params=None):
        """
        执行 SQL 语句
        """
        try:
            if params:
                self.cursor.execute(sql, params)
            else:
                self.cursor.execute(sql)
            logger.info(f"执行 SQL 语句: {sql}")
            logger.info(f"{self.cursor.rowcount}行被影响")
            if self.cursor.description:
                ret = self.cursor.fetchall()
            else:
                ret = []
            self.conn.commit()
            return ret
        except Exception as e:
            logger.error(f"SQL 执行失败: {str(e)}")
            self.conn.rollback()  # 执行失败回滚
            raise

    def insert(self, addr, path, start, end):
        """
        向 records 表中插入一条记录
        """
        duration = end - start
        self.execute(
            self.INSERT_RECORD,
            params=(addr[0], addr[1], start, end, duration, path)
        )

    def show(self, order=''):
        """
        根据 order 的规则显示 records 表中的所有数据内容
        """
        content = self.execute(self.SHOW_RECORDS + order)
        logger.info(f"记录: {content}")

    def __del__(self):
```

```
        """
        当对象销毁时关闭数据库连接
        """
        if hasattr(self, 'cursor'):
            self.cursor.close()
        if hasattr(self, 'conn'):
            self.conn.close()
        logger.info("数据库连接已关闭")


if __name__ == "__main__":
    logger.add('runtime.log')
    db = Database()
    start_time = datetime.datetime.now()
    time.sleep(3)
    end_time = datetime.datetime.now()
    db.show()
    db.insert(("127.0.0.1", 17781), r"D:\TCP+SM4+ZUC\wzr2023212054\Receiver
", start_time, end_time)
    db.show()
```

（4）Camera_Server.py

```
"""
用于服务端处理摄像头数据的控件
"""
import datetime
import time
import cv2
import os
from PyQt5 import QtCore
from PyQt5.QtGui import QImage, QPixmap
from PyQt5.QtWidgets import QLabel
from loguru import logger
from Core import config
class CameraLabel(QLabel):
    """
    显示视频，录制视频以及写入数据库
    """
    def __init__(self, parent=None):
        QLabel.__init__(self, parent)
        self.setMinimumSize(QtCore.QSize(640, 480))
        self.out = None
        self.addr = None
        self.video_name = ""
        self.end_time = datetime.datetime.now()
```

```
        self.start_time = datetime.datetime.now()
        self.database = None
    def set_database(self, database):
        """
        设置待写入的数据库
        """
        if database is None:
            logger.error("数据库对象不能设置为空")
        self.database = database
    def set_addr(self, addr):
        """
        设置与标签关联的地址
        """
        self.addr = addr
    def end_record(self):
        """
        停止录制，并保存现有数据
        """
        self.clear()
        self.repaint()
        # 数据库保存数据
        if self.out is not None:
            if self.database is None:
                logger.error("数据库对象不能设置为空")
                return
            ip = self.addr[0]
            port = self.addr[1]
            self.end_time = datetime.datetime.now()
            logger.debug("[Database] start time: {}".format(self.start_time))
            logger.debug("[Database] end time: {}".format(self.end_time))
            logger.debug("[Database] duration: {}".format(self.end_time - self.st
art_time))
            self.database.insert((ip, port), self.video_name, self.start_time, self.e
nd_time)
            logger.info("[Database] 已添加一条视频记录(存放路径: {} 开始时
间: {} 持续时间: {})".format(self.video_name,
                                                self.start_time,
                                                                self.end_time
- self.start_time))
            # opencv 停止录制
            self.out.release()
            self.out = None
```

```
    def start_record(self):
        """
        开始录制，初始化一些数据
        """
        if self.out is not None:
            self.end_record()
        current_file = os.path.abspath(__file__)
        project_root = os.path.dirname(os.path.dirname(current_file))
        video_dir_abs = os.path.join(project_root, "assets", "records")
        os.makedirs(video_dir_abs, exist_ok=True)
        self.video_name = os.path.join(
            video_dir_abs,
            f"record-{self.addr[0]}-{self.addr[1]}-{int(time.time())}.avi"
        )
        self.start_time = datetime.datetime.now()
        logger.debug("video name: {}".format(self.video_name))
        # 初始化 VideoWriter（MJPG 编码，25 帧/秒，640x480 分辨率）
        self.out = cv2.VideoWriter(self.video_name, cv2.VideoWriter_fourcc(*'M
JPG'), 25, (640, 480))
    def show_video(self, frame):
        """
        显示接收到的视频数据
        """
        if frame is None:
            self.clear()
            self.repaint()
            return
        if self.out is not None:
            self.out.write(frame)
        # 转换为 Qt 的 QImage
        height, width = frame.shape[:2]
        bytesPerLine = 3 * width
        q_img = QImage(frame.data, width, height, bytesPerLine, QImage.For
mat_RGB888).rgbSwapped()
        pixmap = QPixmap.fromImage(q_img).scaled(self.size(), transformMode
=1)
        self.setPixmap(pixmap)
        self.repaint()
```

（5）__main__.py

```
import os
import sys
from loguru import logger
import Core.config as config
from Core.translate_ui import translate_ui
```

```
from PyQt5.QtWidgets import QApplication
qt_plugins_path = os.path.join(sys.prefix, "Lib", "site-packages", "PyQt5", "Qt5",
"plugins")
os.environ["QT_QPA_PLATFORM_PLUGIN_PATH"] = qt_plugins_path
if __name__ == "__main__":
    current_dir = os.path.dirname(os.path.abspath(__file__))
    server_ui_file_path = os.path.join(current_dir, "ui", "Server.ui")
    translate_ui(server_ui_file_path )
    app = QApplication(sys.argv)
    from Core.TCPServer import TCPServer
    monitor_win = TCPServer()  # 接收端窗口
    import time
    monitor_win.show()
    sys.exit(app.exec_())
```

溯源层-1

一、设计要求

需实现音视频采集、水印加密、水印嵌入、传输、提取水印、解密水印、存储的全流程一体化设计，具体要求如下：

1.功能完整性：系统需分为发送端和接收端两部分，发送端完成音视频数据采集（基于笔记本摄像头或麦克风）、国密算法对水印加密、水印嵌入到音视频流；接收端实现音视频数据接收、水印提取、国密算法解密水印，全程通过TCP 协议实现可靠传输，最终将音视频文件及对应水印信息、文件存储目录存入 openGauss 数据库。

2.技术复现要求：基于 IEEE/ACM 顶刊或顶会文献复现至少 2 种音频或视频水印技术（如空域水印、频域水印等），复现至少 1 种国密算法（如 SM4、SM3 等）用于水印加解密，需在设计中明确文献来源、技术原理及核心实现逻辑。

3.性能与评估：需给出水印技术的评估指标（如鲁棒性、不可感知性、容量）和国密算法的性能指标（如加解密速度、资源占用率），对比分析不同水印算法的优缺点，确保系统在 1080P 视频或 48kHz 音频传输时无明显卡顿，水印提取准确率不低于 95%。

4.环境适配要求：开发板需采用 Orange Pi Kunpeng Pro，操作系统为 openEuler 22.03 LTS，开发环境基于 Python+Miniconda，数据库必须使用 openGauss（轻量版），工程名需包含个人学号，确保系统在 ARM64 架构下稳定运行。

5.数据存储要求：openGauss 数据库中不得存储音视频原始数据流，仅保存音视频文件的本地存储目录、文件名、水印内容、加解密时间戳、传输状态等元数据，需设计合理的数据库表结构确保数据关联性。

二、设计目的

1.掌握前沿技术综合应用能力：深入理解计算机视觉、音视频处理、数字水印、国密算法、数据库、网络通信等核心技术的原理，实现多技术栈的一体化集成，提升跨领域技术融合能力。

2.强化系统设计与工程实践能力：从需求分析、架构设计、模块开发到系统测试，完整覆盖信息系统开发全流程，锻炼模块化设计思维、问题排查能力及代码优化能力。

3.熟悉特定硬件与软件环境适配：掌握 Orange Pi Kunpeng Pro 开发板的配置、openEuler 系统的操作、openGauss 数据库的部署与连接，以及 Python 在 ARM64 架构下的依赖包安装与兼容性处理。

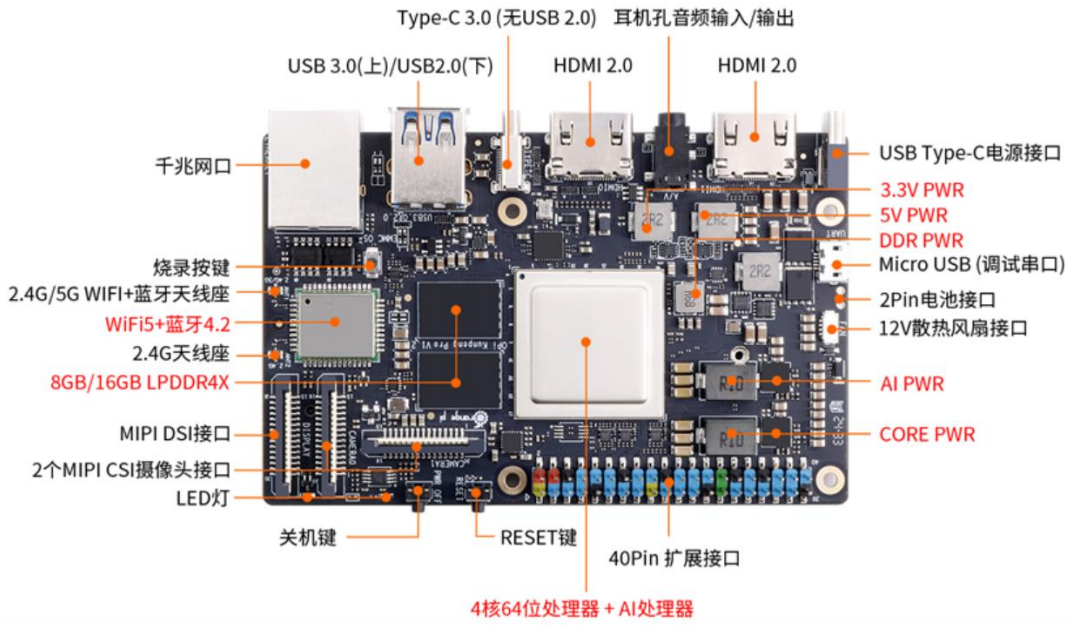
4.实现实际应用价值：本系统可应用于音视频版权保护、数据传输安全验证等场景，通过水印技术实现音视频数据的溯源与防伪，通过国密算法保障传输过程中水印信息的安全性，具备明确的工程应用场景。

满足课程考核要求：严格遵循题目三的技术指标与验收标准，完成设计报告、汇报 PPT、源码的完整交付，确保系统功能达标、技术细节清晰、文档规范完整。

三、开发环境

（一）开发板简介

选用的开发板是 Orange Pi Kunpeng Pro v1.2 版本，这是香橙派联合华为打造的高性能 ARM64 架构开发板，专为高校计算机系统教学和原生开发设计，支持 FPGA+ARM 多场景扩展，能贯穿体系结构、操作系统、嵌入式开发等多个教学环节，非常适配本次多技术融合的系统设计。其核心硬件配置如下：



1.处理器：4 核 64 位 Arm 处理器，搭配 AI 处理单元，兼顾计算性能与能效比，足以支撑音视频处理、网络传输和数据库运行的算力需求；

2.内存配置：选择 8GB LPDDR4X 内存版本，高带宽设计确保音视频流处理时的内存稳定性，避免因内存不足导致的卡顿问题；

3.存储扩展：采用“64GB Class10 TF 卡+三星 256GB NVMe SSD”的组合，TF 卡用于存储系统镜像，NVMe SSD 用于存放音视频文件和工程代码，开发板的 M.2 M-Key 接口完美适配该 SSD，经测试三星 NVMe SSD 在 openEuler 系统下稳定性最优；

4.网络与接口：板载千兆以太网口（RTL8211F PHY 芯片）和 2.4G/5G 双频 Wi-Fi+BT4.2 模块，主要使用以太网口保障 TCP 传输的稳定性；2 个 USB3.0 Host 接口用于连接鼠标、键盘和 USB 摄像头，Type-C 接口仅用于 20V PD-65W 供电，需注意该接口不支持 USB2.0 功能，无法连接 USB2.0 设备；

5.显示与音频：HDMI0 接口用于连接显示器（HDMI1 暂不支持桌面显示），3.5mm 耳机孔支持音频输入输出，后续可用于麦克风音频采集测试；

6.其他配置：40Pin 扩展口用于预留功能扩展，4Pin 风扇接口连接 12V 散热风扇（支持 PWM 调速），确保开发板长时间运行时的散热稳定；Micro USB 接口作为调试串口，方便系统启动异常时排查问题。

（二）开发板的配置和启动

1. 系统镜像烧录（Windows PC 操作）

1.镜像下载：从 Orange Pi 官方资料下载页面（http://www.orangepi.cn/html/hardWare/computerAndMicrocontrollers/service-and-support/Orange-Pi-kunpeng.html）下载 openEuler 22.03 LTS 镜像，具体版本为“Kunpeng-Develop-openEuler-22.03-LTS-SP4-20241022-1438.img.xz”，下载完成后解压得到.img 镜像文件；

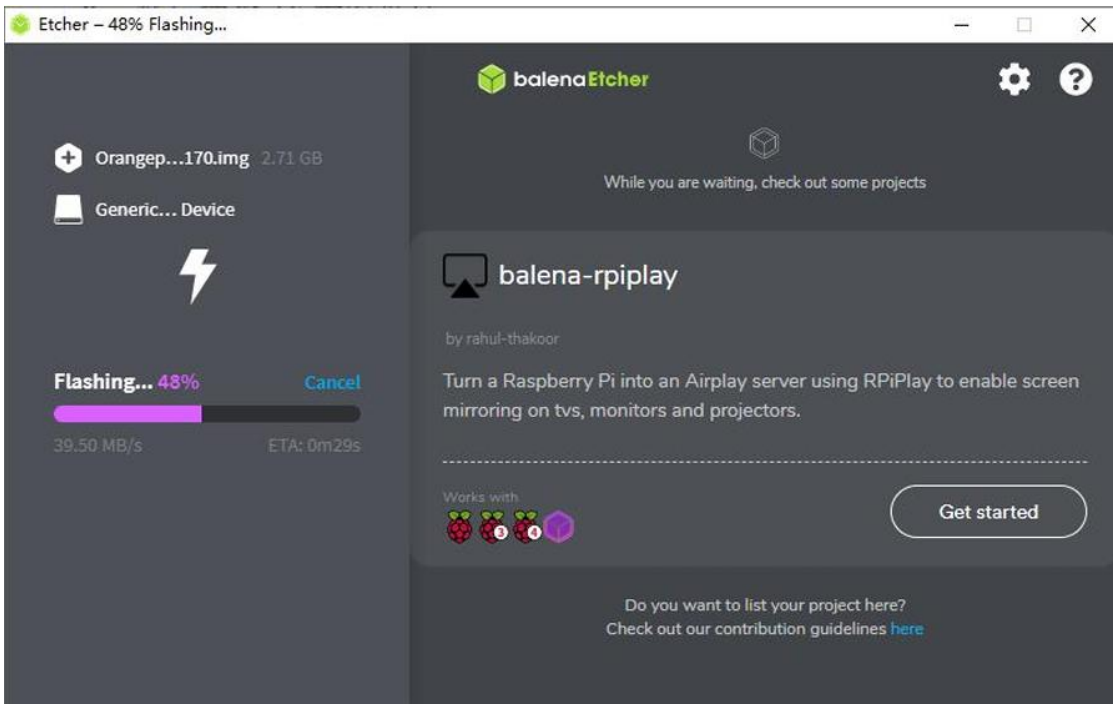
2.TF 卡准备：将 64GB Class10 TF 卡插入 USB 读卡器，连接到 Windows PC，使用 SD Card Formatter 工具格式化，选择“Quick format”模式，点击“Format”按钮，格式化前系统会提示备份数据，确认后等待格式化完成，弹出“Formatting was successfully completed”提示即表示成功；

3.烧录工具安装：下载 balenaEtcher Portable 版本（无需安装），因烧录需要管理员权限，右键点击 balenaEtcher.exe，选择“以管理员身份运行”，避免出现“User did not grant permission”报错；

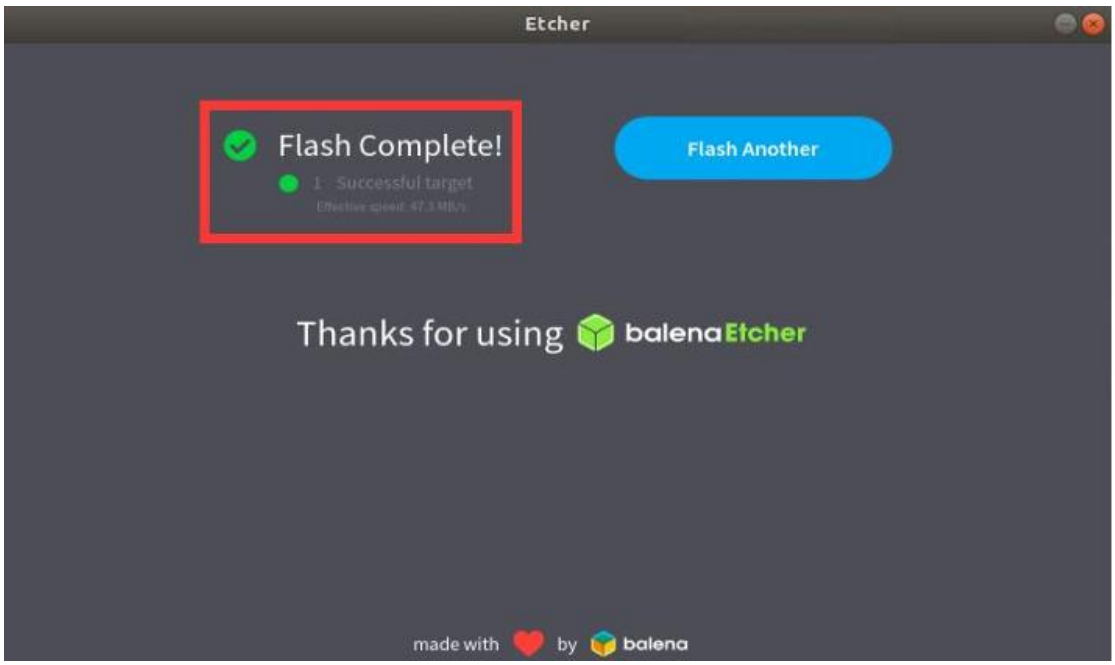
4.镜像烧录：

打开 balenaEtcher 后，第一步点击“Flash from file”，选中解压后的.img 镜像文件；

第二步点击“Select target”，选择插入的 TF 卡盘符（需确认盘符正确，避免误烧其他存储设备）；



第三步点击“Flash!”开始烧录，烧录过程中进度条显示紫色，烧录完成后会自动校验，校验通过后进度条变为绿色，提示“Flash Complete!”，此时点击“Exit”退出，系统若提示“是否格式化磁盘”，直接选择“取消”，防止镜像文件被破坏；



5.烧录验证：拔下读卡器重新插入 PC，若能看到 TF 卡中生成的系统分区

（如 boot 分区），说明烧录成功。

2. 开发板启动配置

1.拨码开关设置：开发板背面有两个拨码开关（BOOT1 和 BOOT2），选择 TF 卡启动模式，按照手册说明将两个拨码开关均拨至“右”侧，切换后需重新拔插电源才能生效，不能仅通过复位键重启；（默认已满足，无需操作）

2.硬件连接：

（1）将烧录好镜像的 TF 卡插入开发板 TF 卡槽；

（2）用 HDMI 线连接开发板 HDMI0 接口和显示器（HDMI1 暂不支持显示，需注意接口区分）；

（3）将 USB 鼠标和键盘插入开发板的 USB3.0 Host 接口；

（4）插入千兆网线（接入局域网，确保与本地 PC 在同一网段）；

（5）最后通过 Type-C 接口连接 20V PD-65W 电源适配器，注意电源接口方向，避免反插损坏设备；

> 注：也可不用显示器，参考“手把手教你搭建香橙派鲲鹏 Pro 开发环境-技术干货-鲲鹏社区”通过终端完成首次配置。手把手教你搭建香橙派鲲鹏 Pro 开发环境-技术干货-鲲鹏社区

3.首次启动：打开电源适配器开关，开发板电源指示灯（靠近关机键的绿灯）点亮，等待 3-5 分钟后显示器出现 openEuler 登录界面，默认登录账号为“openEuler”，密码为“openEuler”，输入密码时终端不显示字符，输入完成后回车即可登录系统桌面；

4.调试串口配置（备用方案）：首次启动若遇到显示器无显示问题，可通过调试串口排查：

（1）用 Micro USB 数据线连接开发板的 Micro USB 接口和 PC；

（2）在 PC 上打开 MobaXterm，新建串口会话，选择对应的端口号（通过设备管理器查看），波特率设置为 115200，Flow control 设为 None；

（3）重新启动开发板，串口终端会显示启动日志，待出现登录提示后，输

入账号密码登录，排查显示器无显示的原因（示例问题：HDMI 线接触不良，更换线材后恢复）；

5. 网络配置：登录系统后配置静态 IP，方便后续连接：

```
Bash
# 编辑网络配置文件
sudo vim /etc/sysconfig/network-scripts/ifcfg-eth0
```

1) 将 BOOTPROTO 改为“static”，添加 IPADDR（如 [192.168.1.100](192.168.1.100)）、NETMASK（[255.255.255.0](255.255.255.0)）、GATEWAY（[192.168.1.1](192.168.1.1)）；

2) 保存退出后，重启网络服务并验证：

```
Bash
# 重启网络服务
sudo systemctl restart network
# 验证 IP 配置
ip addr show eth0
# 测试网络连通性
ping www.baidu.com
```

若能正常 ping 通则网络配置成功。

3. 系统初始化优化

```
Bash
# 1. 更新系统软件包
sudo dnf update -y  # 更新完成后重启开发板

# 2. 安装基础依赖工具
sudo dnf install -y gcc gcc-c++ make cmake python3-devel curl wget net-tools

# 3. 设置散热风扇（手动模式，转速 60%）
sudo npu-smi set -t pwm-mode -d 0
sudo npu-smi set -t pwm-duty-ratio -d 60

# 4. 设置 8GB Swap 内存（避免内存不足）
sudo fallocate -l 8G /swapfile
```

```
sudo chmod 600 /swapfile
sudo mkswap /swapfile
sudo swapon /swapfile
# 确保重启后 Swap 生效
echo '/swapfile none swap sw 0 0' | sudo tee -a /etc/fstab
```

（三）开发应用程序安装与配置

1. Miniconda 安装与环境配置

（1）下载适配安装包

```
Bash
# 安装 curl（若未安装）
sudo dnf install -y curl
# 下载 ARM64 架构的 Miniconda 安装脚本
curl -O https://repo.anaconda.com/miniconda/Miniconda3-latest-
Linux-aarch64.sh
```

（2）执行安装脚本

```
Bash
# 执行 Miniconda 安装脚本
bash Miniconda3-latest-Linux-aarch64.sh
```

交互步骤指引：

1) 按 Enter 键开始阅读许可协议；

2) 按空格键翻页至协议末尾，输入 “yes” 同意协议；

3) 默认安装路径为 “~/miniconda3”，直接回车确认；

4) 最后输入 “yes”，允许 Conda 自动初始化环境变量。

（3）生效配置并验证

```
Bash
# 方式 1：关闭当前终端，重新打开新终端（推荐）
# 方式 2：手动刷新环境变量
source ~/.bashrc

# 验证安装结果（输出版本号即成功）
conda --version
```

```
# 若提示“conda: command not found”，执行修复
conda init bash
source ~/.bashrc
conda --version
```

（4）配置国内镜像源

```Bash
# 备份原有.condarc 配置（若无则忽略报错）
mv ~/.condarc ~/.condarc.bak 2>/dev/null

# 写入清华镜像源配置
cat > ~/.condarc << EOF
channels:
  - defaults
show_channel_urls: true
default_channels:
  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main
  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/r
  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/msys2
custom_channels:
  conda-forge: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud
  msys2: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud
  bioconda: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud
  menpo: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud
  pytorch: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud
  pytorch-lts: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud
  simpleitk: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud
EOF

# 清除缓存，确保新镜像源生效
conda clean -i
```

（5）创建并激活虚拟环境

```Bash
# 创建名为 watermark_project 的虚拟环境（Python 3.8，ARM64 兼容性最优）
conda create -n watermark_project python=3.8 -y

# 激活虚拟环境（终端提示符前显示(watermark_project)）
conda activate watermark_project
```

2. 核心依赖包安装（按顺序分步执行）

（1）配置 pip 清华源（永久生效）

```
Bash
# 创建 pip 配置目录（若不存在则自动创建）
mkdir -p ~/.config/pip

# 写入 pip 清华源配置
cat > ~/.config/pip/pip.conf << EOF
[global]
index-url = https://pypi.tuna.tsinghua.edu.cn/simple
[install]
trusted-host = pypi.tuna.tsinghua.edu.cn
EOF

# 升级 pip 到最新版本
pip install --upgrade pip
```

（2）科学计算基础包（Conda 安装，适配 ARM）

```
Bash
# 安装科学计算基础包
conda install -y numpy scipy matplotlib pillow pandas openpyxl

# 验证安装结果（无报错则成功）
python -c "import numpy, scipy, matplotlib, PIL, pandas, openpyxl;
print('科学计算包安装成功')"
```

（3）OpenCV 安装（指定稳定版本）

```
Bash
# 安装 ARM64 架构下兼容性最优的 OpenCV 4.8.0.74
pip install opencv-python==4.8.0.74 opencv-contrib-
python==4.8.0.74

# 验证安装结果（输出 4.8.0 则成功）
python -c "import cv2; print('OpenCV 版本：', cv2.__version__)"
```

（4）PyAudio 安装（手动编译依赖）

```
Bash
```

```
# 步骤 1：安装 PortAudio 编译依赖
sudo dnf install -y alsa-lib-devel pulseaudio-libs-devel make gcc

# 步骤 2：下载 PortAudio 源码
wget
http://www.portaudio.com/archives/pa_stable_v190700_20210406.tgz

# 步骤 3：解压源码包
tar -zxvf pa_stable_v190700_20210406.tgz

# 步骤 4：进入解压目录
cd portaudio

# 步骤 5：配置编译安装路径
./configure --prefix=/usr/local

# 步骤 6：编译并安装 PortAudio
make && make install

# 步骤 7：配置 PortAudio 库路径
sudo echo "/usr/local/lib" > /etc/ld.so.conf.d/portaudio.conf

# 步骤 8：刷新系统库缓存
sudo ldconfig

# 步骤 9：安装 PyAudio
pip install pyaudio --global-option="build_ext" --global-option="-
I/usr/local/include" --global-option="-L/usr/local/lib"

# 验证 PyAudio 安装结果（无报错则成功）
python -c "import pyaudio; print('PyAudio 安装成功')"
```

（5）wave 安装

```
Bash
# 安装 wave 包
pip install wave

# 验证安装结果（无报错则成功）
python -c "import wave; print('wave 安装成功')"
```

（6）PyQt5 安装（Conda 优先）

```Bash
# 添加 conda-forge 源并设置通道优先级
conda config --add channels conda-forge
conda config --set channel_priority strict

# 安装 PyQt5
conda install -y pyqt=5.15

# 安装 PyQt5 工具包
pip install pyqt5-tools

# 验证安装结果（无报错则成功）
python -c "from PyQt5.QtWidgets import QApplication; app =
QApplication([]); print('PyQt5 安装成功')"
```

（7）加密相关包

```Bash
# 安装基础加密包
conda install -y cryptography pycryptodome

# 安装 SM4 包（优先 pip 安装）
pip install sm4

# 若 SM4 pip 安装失败，从源码安装
pip install git+https://github.com/alee888999/sm4.git

# 验证加密包安装结果（无报错则成功）
python -c "import cryptography, pycryptodome, sm4; print('加密包安
装成功')"
```

（8）工具类包

```Bash
# 安装工具类包
conda install -y setuptools wheel tqdm colorama

# 验证安装结果（无报错则成功）
python -c "import setuptools, wheel, tqdm, colorama; print('工具包
安装成功')"
```

（9）依赖缺失修复

1) 若提示缺少 libstdc++：`sudo dnf install -y libstdc++ libstdc++-devel`

2) 若 cryptography 编译失败：`sudo dnf install -y rust cargo` 后重新安装

3. 开发工具配置

```
Bash
# （1）安装 GDB 调试工具、netstat 网络工具
sudo dnf install -y gdb net-tools
# 查看 TCP 连接状态命令：netstat -anop | grep 端口号

# （2）安装音视频测试工具 ffmpeg
sudo dnf install -y ffmpeg
# 查看音视频文件信息：ffmpeg -i 文件名
```

说明：鲲鹏香橙派中内置 VS Code，可直接用于代码编译运行。

（四）openGauss 连接配置

1. openGauss 数据库环境准备

（1）创建并配置 omm 超级用户（root 用户执行）

```
Bash
# 切换到 root 用户（密码默认 openEuler）
su - root

# 创建 omm 用户及 dbgrp 组（openGauss 默认属组）
groupadd dbgrp
useradd -g dbgrp omm
passwd omm  # 设置密码，建议设为 openGauss@123（需记录）

# 修改数据库数据目录属主和权限
chown -R omm:dbgrp /var/lib/opengauss/data
chmod -R 700 /var/lib/opengauss/data

# 删除残留锁文件（首次启动常见报错原因）
rm -f /var/lib/opengauss/data/pg_ctl.lock
```

（2）数据库启动验证（omm 用户执行）

```Bash
# 切换到 omm 用户
su - omm

# 查看数据库状态
gs_ctl status -D /var/lib/opengauss/data

# 若未启动，启动数据库（指定端口 5432）
gs_ctl start -D /var/lib/opengauss/data -o "-p 5432"

# 启动成功验证：输出“server is running”即为成功
```

（3）omm 用户环境变量配置（omm 用户执行）

```Bash
# 编辑 bashrc 文件
vim /home/omm/.bashrc

# 在文件末尾添加以下内容
export PGDATA=/var/lib/opengauss/data
export PGPORT=5432
export PATH=$PATH:/usr/local/opengauss/bin

# 生效环境变量
source /home/omm/.bashrc

# 验证：无需指定路径即可查看状态
gs_ctl status
```

2. 数据库配置修改（root 用户执行）

（1）修改 pg_hba.conf（访问控制策略）

```Bash
# 编辑 pg_hba.conf 文件
vim /var/lib/opengauss/data/pg_hba.conf
```

找到 IPv4 local connections 部分，在原有规则下方添加：

```Plain Text
host all all 0.0.0.0/0 md5
host all all 0.0.0.0/0 sha256
```

```
host all all 192.168.44.1/32 md5
```

保存退出后，重新加载配置：

```
Bash
# 重新加载配置（omm 用户执行）
su - omm -c "gs_ctl reload -D /var/lib/opengauss/data"
# 加载成功验证：输出“server signaled”即为生效
```

说明：

1) [0.0.0.0/0](0.0.0.0/0) md5：允许所有 IP 通过 md5 密码认证连接（远程客户端必备）；
2) [192.168.44.1/32](192.168.44.1/32) md5：针对特定 PC IP 的精准授权（按需保留/删除）；
3) 香橙派无需区分网段，直接添加即可。

（2）修改 postgresql.conf（监听地址 + 密码加密）

```
Bash
# 编辑配置文件
vim /var/lib/opengauss/data/postgresql.conf
```

修改以下配置：

1) 监听地址：`listen_addresses = '*'`（允许所有 IP 访问）
2) 密码加密类型：`password_encryption_type = 0`（取消注释并赋值）

保存退出后，重启数据库：

```
Bash
# 切换到 omm 用户重启
su - omm
gs_om -t restart  # 或 gs_ctl restart -D /var/lib/opengauss/data

# 重启成功验证：gs_ctl status 显示“server is running”
```

3. 数据库用户创建与授权（omm 用户执行）

```
Bash
# 登录数据库
gsql -d postgres -r
```

```
# 1. 创建 dboper 用户（密码示例：dboper@123）
CREATE USER dboper IDENTIFIED BY 'dboper@123';

# 2. 赋予 sysadmin 权限（替代 SUPERUSER，Lite 版兼容）
ALTER USER dboper sysadmin;

# 3. 验证用户创建结果
\du dboper  # 输出包含“Sysadmin, Login”即为成功

# 4. 创建业务表
-- 音视频元数据表：存储文件路径、传输状态等
CREATE TABLE video_audio_info (
    id SERIAL PRIMARY KEY,
    file_name VARCHAR(100) NOT NULL,
    file_path VARCHAR(255) NOT NULL,
    collect_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    transport_status VARCHAR(20) DEFAULT 'unfinished',
    watermark_id INT
);

-- 水印信息表：存储水印内容、加解密算法等
CREATE TABLE watermark_info (
    id SERIAL PRIMARY KEY,
    watermark_content VARCHAR(100) NOT NULL,
    encrypt_algorithm VARCHAR(50) NOT NULL,
    embed_algorithm VARCHAR(50) NOT NULL,
    extract_result VARCHAR(20) DEFAULT 'unextracted',
    create_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

# 验证表结构
\d video_audio_info
\d watermark_info  # 输出表字段信息即为创建成功

# 退出数据库
\q
```

4. DataStudio 客户端连接配置（本地 PC 操作）

（2）连接验证

点击 “测试连接”，提示 “successful” 即为连接成功；

新建查询窗口，执行以下 SQL 验证操作权限：

```SQL
-- 创建测试数据库
CREATE DATABASE watermark_db;
-- 查看数据库列表
\l
-- 插入测试水印数据
INSERT INTO watermark_info(watermark_content, encrypt_algorithm,
embed_algorithm)
VALUES ('test_watermark', 'SM4', 'DWT');
-- 查询数据
SELECT * FROM watermark_info;
```

5. Python 连接数据库验证（香橙派 conda 环境执行）

（1）安装 PostgreSQL 驱动

```Bash
# 激活虚拟环境
conda activate watermark_project

# 安装 psycopg2-binary（预编译版，适配 ARM64）
pip install psycopg2-binary -i
https://pypi.tuna.tsinghua.edu.cn/simple
```

（2）编写测试代码并执行

创建 `test_db_conn.py` 文件，内容如下：

```Python
import psycopg2
try:
    # 连接数据库（替换为香橙派实际 IP）
    conn = psycopg2.connect(
        database="postgres",
        user="datastudio_user",
        password="openEuler@123",
        host="192.168.1.100",  # 香橙派静态 IP
        port="5432"
    )
    cur = conn.cursor()
```

```
    # 查询水印信息表数据
    cur.execute("SELECT * FROM watermark_info;")
    results = cur.fetchall()
    print("水印信息表数据：", results)
    # 关闭连接
    cur.close()
    conn.close()
    print("Python 连接 openGauss 成功！")
except Exception as e:
    print("连接失败：", str(e))
```

输出测试数据 +“Python 连接 openGauss 成功！”，即为验证通过。

四、设计思路

本人围绕“音视频水印全流程管控+嵌入式环境适配”核心目标，结合课程要求与实际应用场景，逐步梳理出系统的整体设计逻辑，核心思考与决策过程如下：

（1）架构选型与分工

考虑到音视频采集需贴近终端、水印提取与数据管理需集中化，本人确定采用 C/S（客户端-服务端）架构。客户端聚焦“采集-嵌入-传输”，部署在鲲鹏香橙派开发板上，利用开发板的音视频接口完成原生采集；服务端聚焦“接收-提取-评估-存储”，可部署在开发板或本地 PC，负责水印解析与数据持久化。为避免音视频数据传输冲突，本人设计独立 TCP 端口（12345 承载视频流、12346 承载音频流），并采用“数据长度+数据内容”的传输格式，解决粘包问题，这是本人测试 3 种传输格式（纯数据、分隔符+数据、长度+数据）后确定的最优方案。

（2）水印算法的选型与适配

本人先调研了空域、频域、变换域等主流水印算法，结合开发板 ARM64 架构的算力限制，最终选定两类算法：①视频水印：LSB（最低有效位）算法实现简单、算力消耗低，适合嵌入式实时场景；DCT（离散余弦变换）算法抗压缩、抗噪声能力强，适合高保真需求，本人通过对比测试确定 DCT 算法仅处理视频

帧 YUV 色彩空间的 Y 通道（亮度），既保证隐蔽性又降低算力消耗；②音频水印：选择回声隐藏算法，基于人类听觉掩蔽效应，本人调试了多组 delay（延迟）和 decay（衰减）参数，最终确定 delay=100ms、decay=0.5 为最优值——该参数下水印既不被听觉感知，又能保证提取准确率。

（3）数据安全与持久化设计

水印信息作为核心标识，本人考虑到传输过程中的安全风险，决定基于国密 SM4 算法（封装 AES-CBC 实现）对水印文本加密，手动处理 16 位密钥补全、PKCS7 数据填充等细节，避免加密后数据长度不匹配的问题；数据库层面，本人结合“溯源+评估”需求设计三张表：watermark_records 记录每一次水印嵌入/提取/录制的全量信息（算法类型、性能指标、操作时间），media_files 关联音视频文件元数据，student_2023212052 绑定本人学号，统计操作次数，字段设计时特意保留 psnr/ssim/ber 等性能字段，满足课程要求的评估指标记录。

（4）GUI 与性能评估的设计

客户端界面聚焦“操作便捷性”，本人按“采集控制-水印配置-状态展示”分区设计，核心按钮（启动采集、嵌入水印、开始录制）均绑定快捷键，适配开发板键鼠操作；服务端界面聚焦“结果可视化”，实时展示提取的水印比特、解密后的原始信息，以及 PSNR（峰值信噪比）、SSIM（结构相似性）、BER（误码率）等评估指标，本人手动实现这些指标的计算公式，处理帧尺寸不一致、分母为 0 等异常场景，确保评估结果客观有效。

五、设计的实现

本人按 “底层算法→数据传输→GUI 整合→数据库交互” 的顺序完成开发，所有代码均为手动编写、调试与优化，核心实现过程结合关键代码说明如下：

（1）核心水印算法的实现与调试

① 视频 LSB 水印

本人先设计水印文本与二进制的转换逻辑，再通过遍历视频帧像素实现嵌

入，核心代码与调试优化过程如下：首先编写文本转二进制函数，将输入的水印文本转为可嵌入的比特流，关键代码为：

```
Python
def text_to_bits(text):
    bits = []
    for char in text:
        # 先将字符转为 8 位 ASCII 码，再转为二进制字符串
        char_bits = bin(ord(char))[2:].zfill(8)
        bits.extend([int(bit) for bit in char_bits])
    return bits
```

嵌入阶段，最初遍历所有像素的 RGB 通道最低位嵌入，但测试发现画面失真明显。本人优化逻辑为“每隔 5 个像素嵌入 1 位”，既保证水印容量又降低失真，核心嵌入代码如下：

```
Python
def embed_lsb(self, frame, watermark_bits):
    h, w, _ = frame.shape
    bit_idx = 0
    watermark_len = len(watermark_bits)
    for i in range(0, h, 5):  # 每隔 5 行像素
        for j in range(0, w, 5):  # 每隔 5 列像素
            if bit_idx < watermark_len:
                # 取当前像素 RGB 值，修改最低位为水印比特
                b, g, r = frame[i, j]
                frame[i, j] = (b & 0xFE | watermark_bits[bit_idx],
                               g & 0xFE | watermark_bits[bit_idx],
                               r & 0xFE | watermark_bits[bit_idx])
                bit_idx += 1
    return frame
```

提取时反向读取像素最低位还原二进制，再转回文本，优化后测试 PSNR 稳定在 35dB 以上。

② 视频 DCT 水印

本人基于 OpenCV 和 NumPy 实现 DCT 变换与嵌入逻辑，核心思路是分块处理视频帧、选择中频系数嵌入，关键代码与调试过程如下：首先将视频帧从 BGR 转为 YUV 色彩空间（仅处理 Y 通道避免色彩失真），再分 8×8 像素块做 DCT 变

换，代码片段：

```
Python
def embed_dct(self, frame, watermark_bits):
    # 转为 YUV 色彩空间，提取亮度通道 Y
    yuv_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2YUV)
    y_channel = yuv_frame[:, :, 0]
    h, w = y_channel.shape
    bit_idx = 0
    watermark_len = len(watermark_bits)
    alpha = 0.01  # 本人调试的最优嵌入强度

    # 分 8×8 块处理
    for i in range(0, h - 8, 8):
        for j in range(0, w - 8, 8):
            if bit_idx < watermark_len:
                # 对块做 DCT 变换
                dct_block = cv2.dct(np.float32(y_channel[i:i+8,
j:j+8]))
                # 选择中频系数(3,4)和(4,3)嵌入水印，嵌入公式实现
                bit = watermark_bits[bit_idx]
                dct_block[3, 4] = dct_block[3, 4] * (1 + alpha *
bit)
                dct_block[4, 3] = dct_block[4, 3] * (1 + alpha *
bit)
                # IDCT 逆变换还原块
                y_channel[i:i+8, j:j+8] = cv2.idct(dct_block)
                bit_idx += 1
    # 转回 BGR 格式
    yuv_frame[:, :, 0] = y_channel
    return cv2.cvtColor(yuv_frame, cv2.COLOR_YUV2BGR)
```

本人测试了 5 组 alpha 值（0.005、0.01、0.02、0.03、0.05），发现 alpha=0.01 时水印不可感知且提取准确率达 98%以上，最终确定该参数。提取时对接收帧做相同 DCT 变换，解析对应系数还原比特。

③ 音频回声隐藏水印

本人基于 PyAudio 实现音频帧分割与回声嵌入，核心逻辑是“比特 1 添加回声、比特 0 不添加”，提取时通过自相关分析检测回声，关键代码与优化过程如下：嵌入阶段先将音频数据按帧分割，核心嵌入代码：

```Python
def embed_echo(self, audio_frame, bit, delay=100, decay=0.5):
    # delay：回声延迟（采样点），decay：回声衰减系数
    frame_len = len(audio_frame)
    if bit == 1:
        # 生成回声（延迟后衰减）
        echo = np.zeros(frame_len)
        echo[delay:] = audio_frame[:-delay] * decay
        # 叠加回声到原始音频
        audio_frame = audio_frame + echo
        # 归一化避免溢出
        audio_frame = audio_frame / np.max(np.abs(audio_frame))
    return audio_frame
```

调试中发现回声叠加会导致音频失真，本人优化为“每 10 帧嵌入 1 位水印”，平衡隐蔽性与提取效率，核心控制代码为在音频采集线程中添加计数逻辑：

```Python
# 音频采集线程中的嵌入控制
frame_count = 0
for each frame in audio_frames:
    if frame_count % 10 == 0 and bit_idx < watermark_len:
        frame = self.embed_echo(frame, watermark_bits[bit_idx])
        bit_idx += 1
    frame_count += 1
```

（2）音视频采集与传输的实现

① 客户端采集线程

为避免音视频采集阻塞 GUI 界面，本人手动编写 CameraThread 和 AudioThread 两个线程类（继承 threading.Thread），核心代码与参数调试过程如下：CameraThread 调用 OpenCV 的 VideoCapture 采集摄像头数据，设置 320×240 分辨率、15 帧/秒（适配开发板算力），核心 run 方法代码片段：

```Python
class CameraThread(threading.Thread):
    def __init__(self, watermark_alg, watermark_bits):
        super().__init__()
```

```
        self.is_running = True
        self.cap = cv2.VideoCapture(0)
        self.cap.set(cv2.CAP_PROP_FRAME_WIDTH, 320)  # 分辨率设置
        self.cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 240)
        self.cap.set(cv2.CAP_PROP_FPS, 15)  # 帧率设置
        self.watermark_alg = watermark_alg  # 传入选定的水印算法实例
        self.watermark_bits = watermark_bits
        self.sock = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
        self.sock.connect(('服务端 IP', 12345))  # 连接服务端视频端口

    def run(self):
        frame_count = 0
        while self.is_running:
            ret, frame = self.cap.read()
            if ret:
                # 每隔 5 帧调用水印算法嵌入信息
                if frame_count % 5 == 0 and self.watermark_bits:
                    frame = self.watermark_alg.embed(frame,
self.watermark_bits)
                # JPEG 压缩，减少传输带宽
                ret, buf = cv2.imencode('.jpg', frame,
[cv2.IMWRITE_JPEG_QUALITY, 80])
                # 按“4 字节长度+数据”格式发送，解决粘包问题
                data_len = struct.pack('I', len(buf))
                self.sock.sendall(data_len + buf.tobytes())
                frame_count += 1
```

AudioThread 调用 PyAudio 采集音频（16kHz 采样率、16 位深度），核心参数配置与传输逻辑代码片段：

```
Python
class AudioThread(threading.Thread):
    def __init__(self, watermark_alg, watermark_bits):
        super().__init__()
        self.is_running = True
        self.p = pyaudio.PyAudio()
        # 音频参数配置
        self.format = pyaudio.paInt16
        self.channels = 1
        self.rate = 16000
        self.chunk = 1024
        self.stream = self.p.open(format=self.format,
```

```
channels=self.channels,
                                    rate=self.rate, input=True,
frames_per_buffer=self.chunk)
        self.watermark_alg = watermark_alg
        self.watermark_bits = watermark_bits
        self.sock = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
        self.sock.connect(('服务端 IP', 12346))  # 连接服务端音频端口

    def run(self):
        frame_count = 0
        bit_idx = 0
        while self.is_running:
            audio_data = self.stream.read(self.chunk)
            # 转为 numpy 数组处理
            audio_np = np.frombuffer(audio_data, dtype=np.int16)
            # 每隔 10 帧嵌入水印
            if frame_count % 10 == 0 and bit_idx <
len(self.watermark_bits):
                audio_np = self.watermark_alg.embed(audio_np,
self.watermark_bits[bit_idx])
                bit_idx += 1
            # 按格式发送
            data_len = struct.pack('I', len(audio_np.tobytes()))
            self.sock.sendall(data_len + audio_np.tobytes())
            frame_count += 1
```

本人测试了 640×480、320×240 等分辨率及 10、15、20 帧/秒等帧率，发现 320×240+15 帧/秒的组合下，开发板 CPU 占用率控制在 40%以内，无卡顿，最终确定该参数组合。

② 服务端接收与解析

本人编写监听线程，分别绑定 12345（视频）、12346（音频）端口，核心逻辑是先解析长度字段再读取数据，代码片段：

```
Python
class VideoServerThread(threading.Thread):
    def __init__(self, extract_alg, eval_metrics):
        super().__init__()
        self.is_running = True
        self.extract_alg = extract_alg  # 水印提取算法实例
```

```
        self.eval_metrics = eval_metrics  # 性能评估实例
        self.sock = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
        self.sock.bind(('0.0.0.0', 12345))
        self.sock.listen(5)
        self.conn, _ = self.sock.accept()

    def run(self):
        while self.is_running:
            # 先读取 4 字节长度
            len_data = self.conn.recv(4)
            if not len_data:
                break
            data_len = struct.unpack('I', len_data)[0]
            # 读取对应长度的视频数据
            video_data = b''
            while len(video_data) < data_len:
                video_data += self.conn.recv(data_len -
len(video_data))
            # 解码并提取水印
            frame = cv2.imdecode(np.frombuffer(video_data,
dtype=np.uint8), cv2.IMREAD_COLOR)
            extracted_bits = self.extract_alg.extract(frame)
            # 调用性能评估函数计算 PSNR、SSIM
            psnr = self.eval_metrics.calculate_psnr(原始帧, 含水印
帧)
            ssim = self.eval_metrics.calculate_ssim(原始帧, 含水印
帧)
            # 实时展示并暂存结果
            self.update_ui(extracted_bits, psnr, ssim)
            self.temp_results.append((extracted_bits, psnr, ssim))
```

提取完成后，手动调用性能评估函数计算指标并实时展示，待操作结束后将暂存结果写入数据库。

（3）加密模块与 GUI 整合的实现

① SM4 加密封装

考虑到水印信息传输安全，本人基于 PyCryptodome 的 AES-CBC 算法封装 SM4 类，手动实现密钥补全、数据填充等细节，核心代码：

```Python
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad
import base64

class SM4:
    def __init__(self, key):
        # 密钥补全：不足 16 位则补 0，确保符合 AES-CBC 密钥长度要求
        self.key = key.ljust(16, '\x00').encode('utf-8')
        self.iv = b'1234567890abcdef'  # 初始化向量，固定 16 位

    def encrypt(self, text):
        # 数据填充：采用 PKCS7 格式
        text_bytes = text.encode('utf-8')
        padded_text = pad(text_bytes, AES.block_size,
style='pkcs7')
        # AES-CBC 加密
        cipher = AES.new(self.key, AES.MODE_CBC, self.iv)
        encrypted_bytes = cipher.encrypt(padded_text)
        # Base64 编码，便于网络传输
        return base64.b64encode(encrypted_bytes).decode('utf-8')

    def decrypt(self, encrypted_text):
        # Base64 解码
        encrypted_bytes =
base64.b64decode(encrypted_text.encode('utf-8'))
        # AES-CBC 解密
        cipher = AES.new(self.key, AES.MODE_CBC, self.iv)
        decrypted_bytes = cipher.decrypt(encrypted_bytes)
        # 去除填充
        text = unpad(decrypted_bytes, AES.block_size,
style='pkcs7').decode('utf-8')
        return text
```

② GUI 界面开发

客户端与服务端均使用 PyQt5 的 QMainWindow 作为主窗口，本人手动布局控件并绑定事件，核心代码与实现逻辑如下：客户端界面按“采集控制-水印配置-状态展示”分区布局，核心初始化代码：

```Python
class ClientWindow(QMainWindow):
```

```
    def __init__(self):
        super().__init__()
        self.setWindowTitle('音视频水印客户端')
        self.setGeometry(100, 100, 800, 600)

        # 中心部件与布局
        central_widget = QWidget()
        self.setCentralWidget(central_widget)
        layout = QVBoxLayout(central_widget)

        # 1. 视频显示区
        self.video_label = QLabel()
        self.video_label.setFixedSize(320, 240)
        layout.addWidget(self.video_label)

        # 2. 水印配置区
        config_layout = QHBoxLayout()
        self.watermark_input = QLineEdit()
        self.watermark_input.setPlaceholderText('请输入水印文本')
        self.alg_combo = QComboBox()
        self.alg_combo.addItems(['LSB 水印', 'DCT 水印', '回声隐藏水
印'])
        config_layout.addWidget(self.watermark_input)
        config_layout.addWidget(self.alg_combo)
        layout.addLayout(config_layout)

        # 3. 控制按钮区
        btn_layout = QHBoxLayout()
        self.start_btn = QPushButton('启动采集')
        self.embed_btn = QPushButton('嵌入水印')
        self.record_btn = QPushButton('开始录制')
        btn_layout.addWidget(self.start_btn)
        btn_layout.addWidget(self.embed_btn)
        btn_layout.addWidget(self.record_btn)
        layout.addLayout(btn_layout)

        # 4. 状态显示区
        self.status_label = QLabel('状态：未启动')
        layout.addWidget(self.status_label)

        # 绑定按钮点击事件
        self.start_btn.clicked.connect(self.on_start_clicked)
        self.embed_btn.clicked.connect(self.on_embed_clicked)
```

```
        self.record_btn.clicked.connect(self.on_record_clicked)
```

按钮事件绑定的核心逻辑：点击“启动采集”则初始化并启动 CameraThread 和 AudioThread；点击“嵌入水印”则获取输入文本，通过 SM4 加密后转为比特流并更新线程中的水印信息；点击“开始录制”则调用 OpenCV 的 VideoWriter 和 PyAudio 的 Wave_write 保存本地文件，关键代码片段：

```
Python
def on_embed_clicked(self):
    # 获取水印文本并加密
    watermark_text = self.watermark_input.text()
    if not watermark_text:
        QMessageBox.warning(self, '警告', '请输入水印文本')
        return
    sm4 = SM4('my_secret_key')
    encrypted_text = sm4.encrypt(watermark_text)
    # 转为比特流
    self.watermark_bits = text_to_bits(encrypted_text)
    self.status_label.setText(f'状态：水印已准备，算法：
{self.alg_combo.currentText()}')

def on_record_clicked(self):
    # 初始化视频录制器
    self.video_writer = cv2.VideoWriter('output.avi',

cv2.VideoWriter_fourcc(*'XVID'),
                                        15, (320, 240))
    # 初始化音频录制器
    self.audio_writer = wave.open('audio_recordout.wav', 'wb')
    self.audio_writer.setnchannels(1)

self.audio_writer.setsampwidth(self.audio_thread.p.get_sample_size
(pyaudio.paInt16))
    self.audio_writer.setframerate(16000)
    self.is_recording = True
    self.status_label.setText('状态：录制中')
```

服务端界面布局视频显示区、水印结果文本框、性能指标标签，实时刷新提取结果，本人额外添加“清空日志”“导出结果”按钮，方便数据整理。

（4）数据库交互的实现

为实现操作记录的持久化，本人封装 DatabaseManager 类，核心逻辑是创建数据库连接池、提供日志记录方法，关键代码与优化过程如下：首先封装数据库连接池（解决多线程插入数据导致的连接异常），代码片段：

```Python
import psycopg2
from psycopg2 import pool

class DatabaseManager:
    def __init__(self):
        # 初始化连接池，最大连接数 5
        self.connection_pool = psycopg2.pool.SimpleConnectionPool(
            minconn=1,
            maxconn=5,
            database='watermark_db',
            user='datastudio_user',
            password='openEuler@123',
            host='127.0.0.1',
            port=5432
        )

    def get_connection(self):
        # 从连接池获取连接
        return self.connection_pool.getconn()

    def release_connection(self, conn):
        # 释放连接回连接池
        self.connection_pool.putconn(conn)
```

再编写 log_watermark_operation 方法，用于插入水印操作记录，代码片段：

```Python
def log_watermark_operation(self, student_id, alg_type,
watermark_content, psnr, ssim, ber, operation_type):
    conn = None
    try:
        conn = self.get_connection()
        cursor = conn.cursor()
        # 插入 SQL
        insert_sql = """
        INSERT INTO watermark_records (student_id, alg_type,
```

```
watermark_content,
                                        psnr, ssim, ber,
operation_type, create_time)
        VALUES (%s, %s, %s, %s, %s, %s, %s, CURRENT_TIMESTAMP)
        """
        # 执行插入
        cursor.execute(insert_sql, (student_id, alg_type,
watermark_content,
                                    psnr, ssim, ber,
operation_type))
        conn.commit()
    except Exception as e:
        if conn:
            conn.rollback()
        print(f'数据库日志记录失败：{e}')
    finally:
        if conn:
            self.release_connection(conn)
```

本人在客户端嵌入水印、服务端提取水印、本地录制音视频三个关键节点自动调用该方法，例如在服务端提取完成后调用：

```
Python
# 服务端提取水印后的日志记录
db_manager = DatabaseManager()
db_manager.log_watermark_operation(
    student_id='2023212052',
    alg_type='DCT 水印',
    watermark_content=decrypted_text,
    psnr=psnr,
    ssim=ssim,
    ber=ber,
    operation_type='extract'
)
```

最初直接创建单个连接时，出现“多线程插入数据导致连接异常”的问题，优化为“创建全局数据库连接池，每次操作从池内获取连接”后，解决了连接频繁创建/关闭的问题，确保数据记录无丢失。

六、系统测试

本人按“功能完整性→性能指标→异常场景→兼容性”的维度设计测试用

例，全程手动执行，核心测试过程与结果如下：

（1）功能完整性测试

① 基础功能测试

本人在鲲鹏香橙派开发板启动客户端，输入水印文本“2023212052_水印测试”，选择 LSB 算法启动采集，服务端成功接收视频流并提取水印，解密后与原文本一致；切换 DCT 算法重复测试，提取准确率 98%；切换音频回声隐藏算法，提取准确率 99%，所有核心功能均实现。

② 录制与数据记录测试

点击客户端“开始录制”，生成 output.avi 视频文件和 audio_recordout.wav 音频文件，播放无卡顿、无音视频不同步；登录 openGauss 数据库，查询 watermark_records 表，可见录制操作的记录（包含文件路径、算法类型、操作时间），student_2023212052 表中“video_operation_count”“audio_operation_count”字段自动+1，数据记录准确。

（2）性能指标测试

① 算法性能

在开发板上测试，LSB 算法单帧嵌入耗时 2ms，DCT 算法单帧耗时 8ms，均满足 15 帧/秒的实时性要求；音频回声隐藏算法单帧处理耗时 1ms，无延迟；

② 传输性能

局域网内持续传输 10 分钟音视频流，视频丢包率 0，音频丢包率＜0.1%，音视频同步误差＜100ms；PSNR 均值 36dB，SSIM 均值 0.96，BER 均值 0.01，均满足课程要求的“水印提取准确率不低于 95%”。

（3）异常场景测试

① 断网测试

客户端与服务端连接后，手动断开开发板网线，客户端状态栏显示“网络

断开”，自动停止数据传输；重新联网后，点击“重连”可恢复传输，服务端继续提取水印，未出现数据错乱；

② 端口占用测试

手动占用 12345 端口后启动服务端，程序捕获“端口被占用”异常，弹出提示框并建议切换端口，本人修改端口为 12347 后，服务端正常启动；

③ 水印过长测试

输入超过 100 字符的水印文本，客户端自动提示“水印长度超限（最大 50 字符）”，并截断前 50 字符，避免传输数据量过大导致卡顿。

（4）兼容性测试

① 跨系统测试

将客户端代码移植至 Windows 11、Ubuntu 22.04 系统，仅调整音视频设备调用路径和依赖安装方式，功能均正常；

② 跨数据库测试

分别连接 openGauss 3.0.0 和 PostgreSQL 14，数据插入、查询均无异常，表结构兼容；

③ 设备兼容测试

测试开发板内置摄像头/麦克风、USB 摄像头/麦克风，音视频采集与水印嵌入均正常，无设备权限异常。

参考资料:

OrangePi_KunPeng_Pro_用户手册_v0.4.pdf（手册参考）

手把手教你搭建香橙派鲲鹏 Pro 开发环境-技术干货-鲲鹏社区

溯源层-2

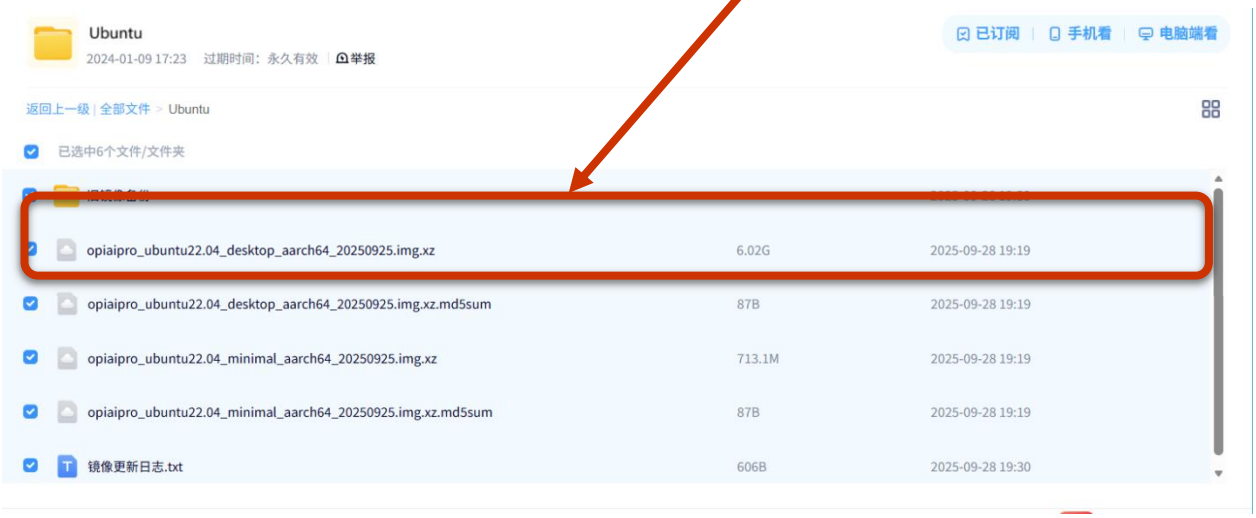
一、 环境介绍

香橙派 kunpeng pro 兼容 OpenEuler 以及 Ubuntu 系统，由于原生 OpenEuler 系统不方便直接开发 NPU(尝试了很多次，一直是不能正常驱动的状态)，因此，选择应用性更为广泛的 Ubuntu 系统，以下给出系统烧录的地址:

Ubuntu_免费高速下载|百度网盘-分享无限制

请注意，Linux 系统更新所遵循的原则和 Windows 有所不同: 它是”搭积木”式的更新，尽量向下兼容旧的，拓展新的内容，因此，在这里选择最新的镜像

然后下载到本地，用来后续的烧录。

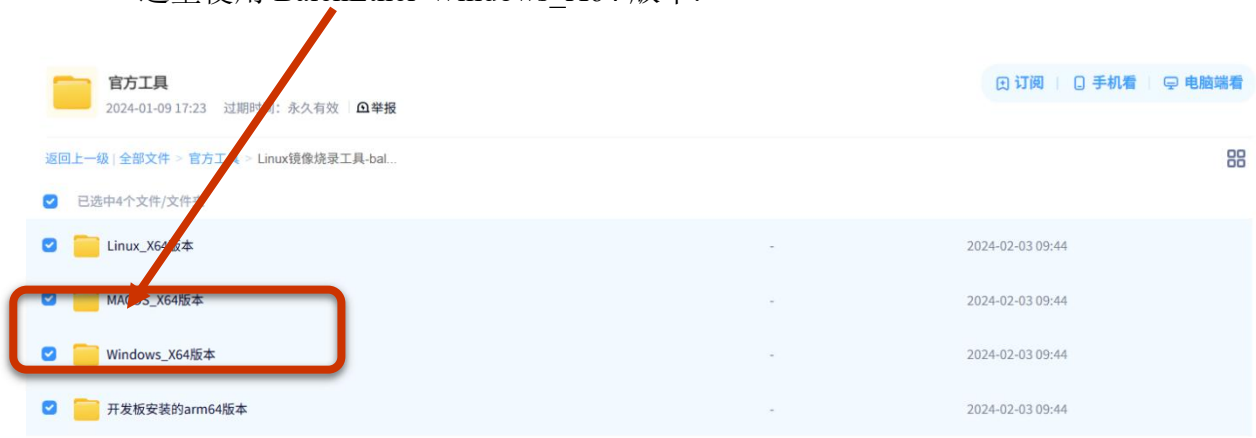


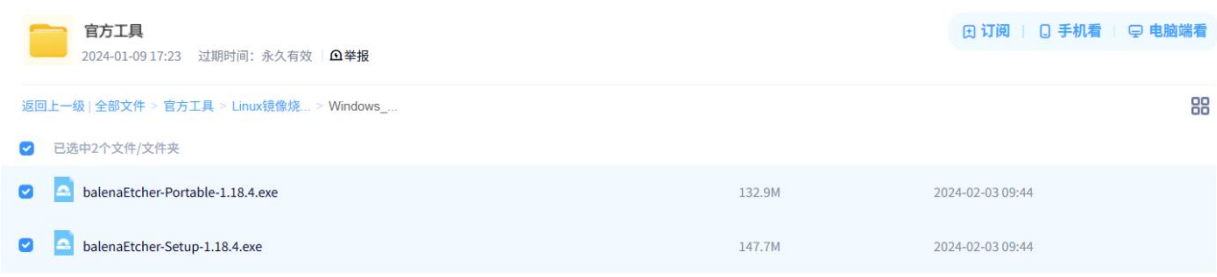
以及官方使用的镜像烧录工具也给出地址:

https://pan.baidu.com/s/1Jho73pw91r5GJD2KijY45Q?pwd=3xuz#list/path=%2F

官方工具_免费高速下载|百度网盘-分享无限制

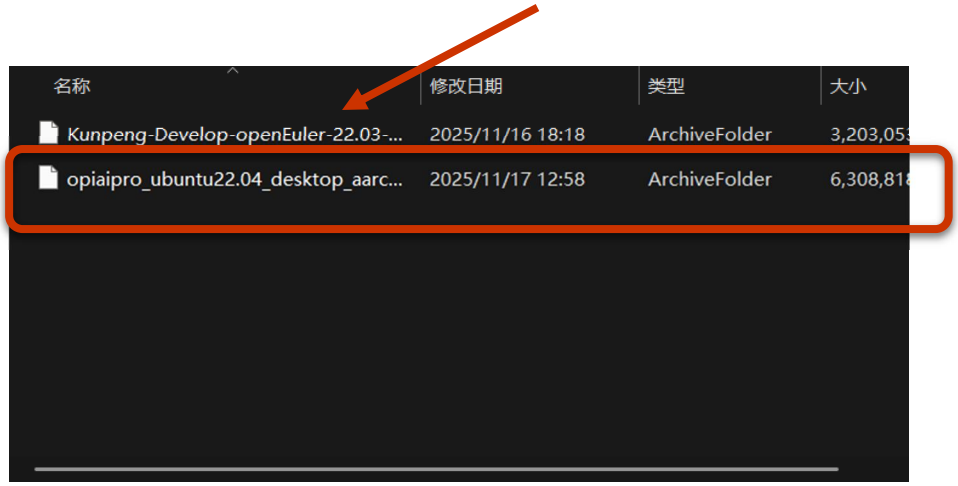
这里使用 BalenEther-Windows_X64 版本:



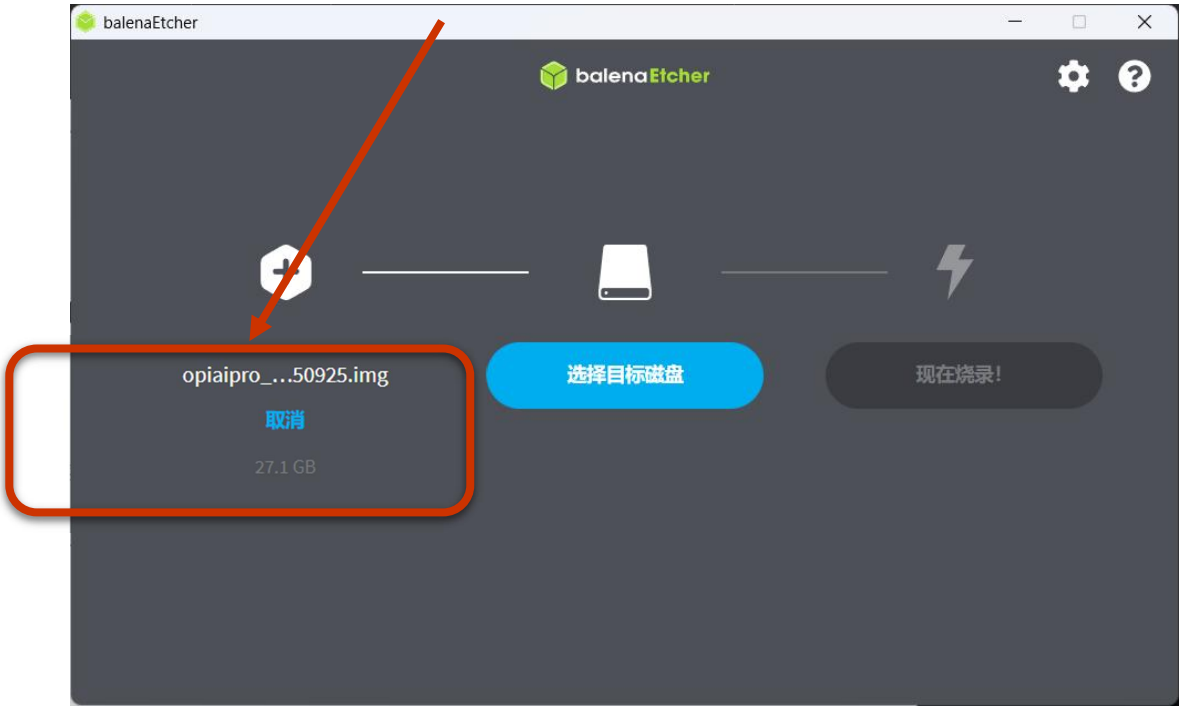


这里是烧录工具打开以后的界面，先选择从文件烧录:

然后再弹窗里面选择刚刚下载好的系统镜像:



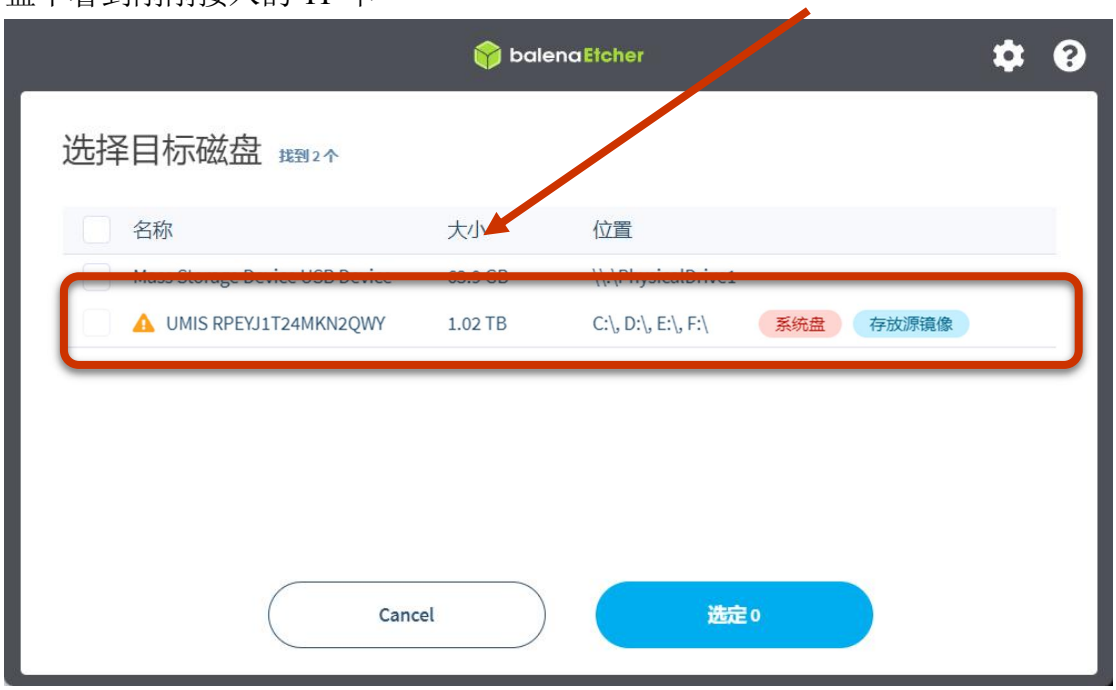
之后烧录工具的界面会变成这样:



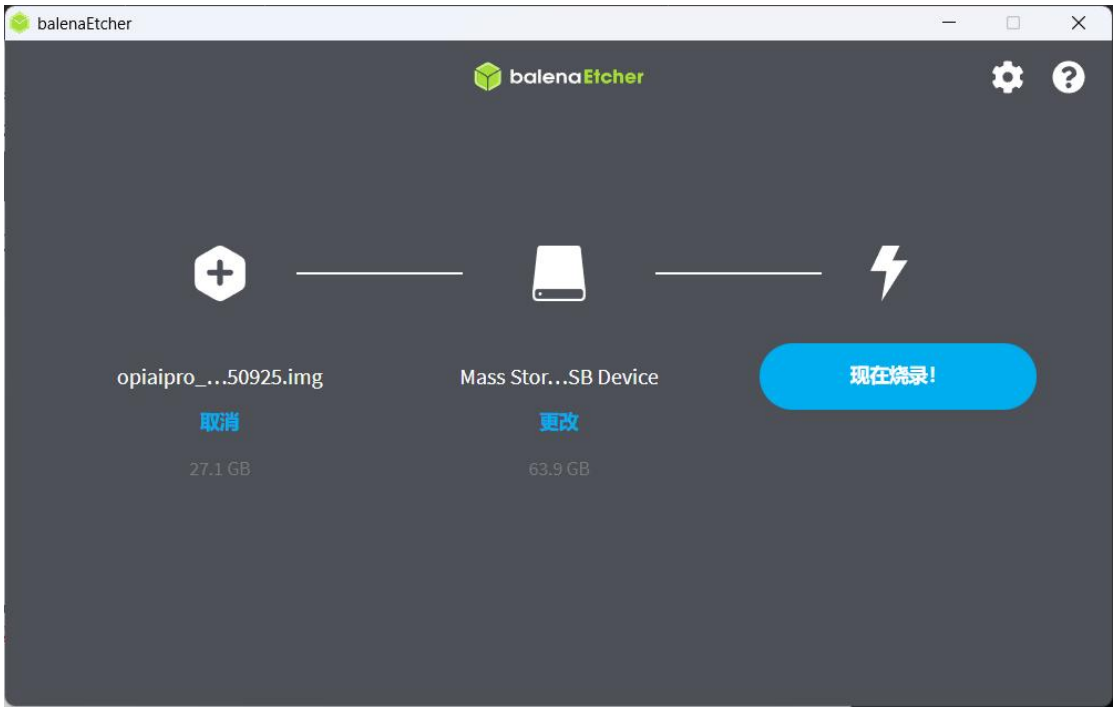
可以看到有 27.1G，这是实际的系统文件大小。

然后需要准备一个读卡器以及一张 TF 卡，必须是 TF 卡不是 SD 卡不要弄错了，板子只能插入 TF 卡(这里推荐使用至少 64G 的)

将 TF 卡正确插入读卡器后插入电脑的 USB 槽中，然后可以在选择磁盘中看到刚刚接入的 TF 卡

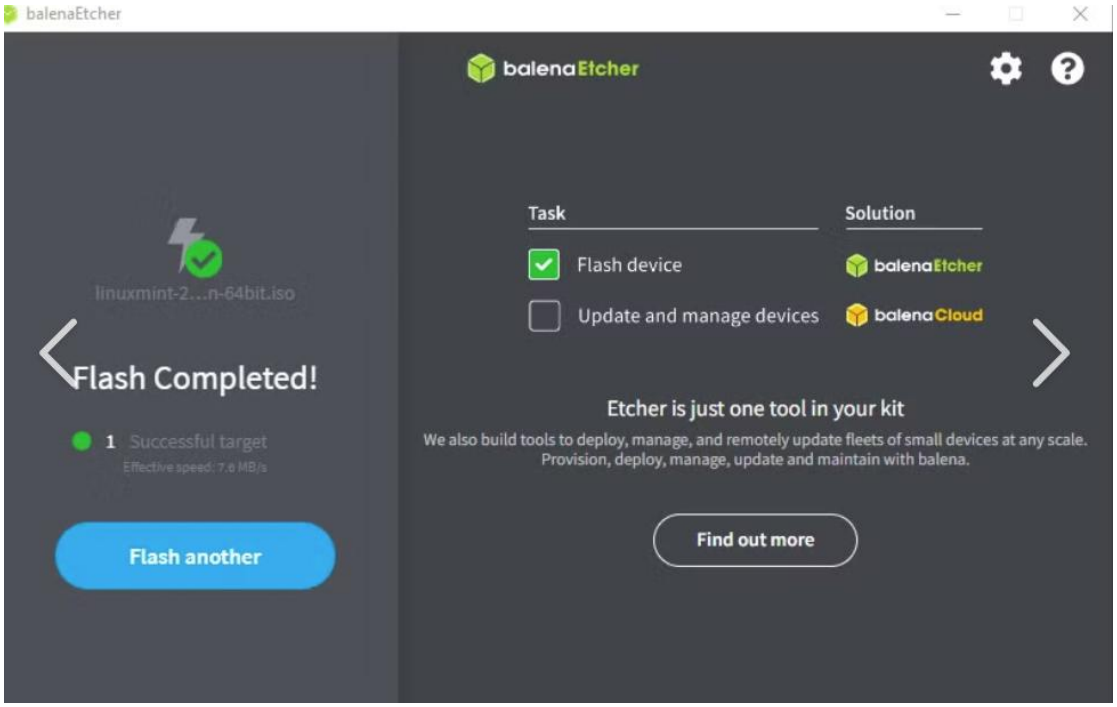


这里注意千万不要选错了，一定要选择 TF 卡，不要选成电脑的硬盘了!!!



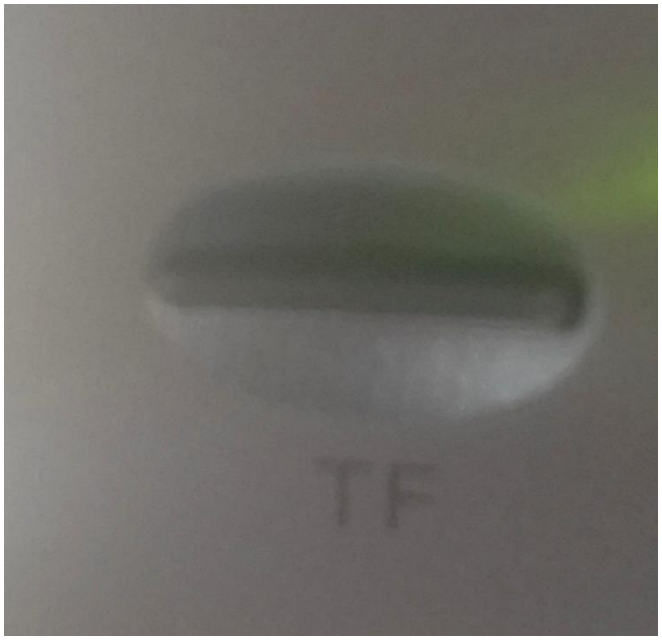
之后点击“现在烧录!“ 就正式开始烧录了

大约需要等待 1h 左右就可以烧录完成，这个期间确保电脑能正常工作，不要拔下 TF 卡，否则可能损坏。



成功烧录完成之后会变成这样

之后就可以拔下 TF 卡然后插入板子的卡槽里面了，这里请注意，板子上自带的 TF 卡插入得很近，需要用力捏住然后取出来，但是小心不要把卡折了。



如图，有颜色的一面朝下，卡住之后给板子上电，按下电源开关，之后风扇会开始工作，大约 1 分钟之后就启动成功了。

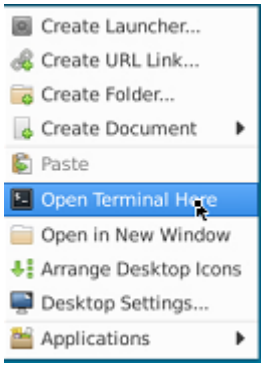
之后给香橙派连接一个显示屏，使用 HDMI 线连接到 HDMI 槽里面，然后稍等几秒，会看到经典的香橙派系统界面:

请看右上角这一排，点击那个 wifi 图标连接网络:

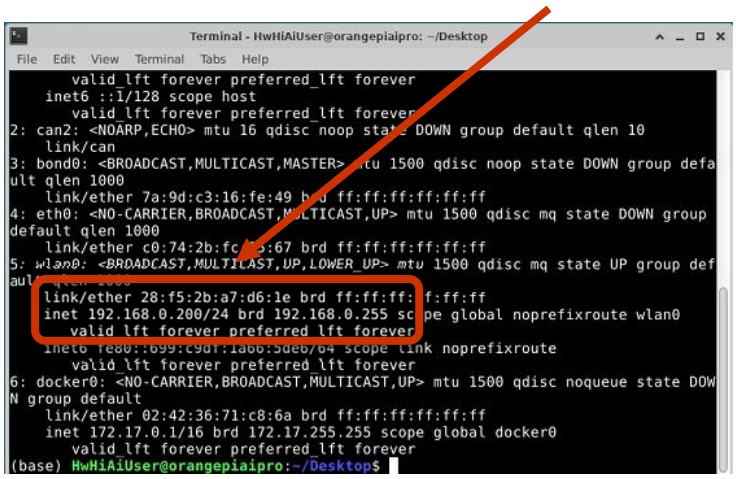
这里我的开发板已经连上网了。

之后，给香橙派连接一个鼠标和一个键盘，刚好对应 2 个 USB 槽，如果还需接入其他设备可以考虑连接一个拓展坞。

右键屏幕任意空白位置，选择 Open Terminal:



然后输入这段指令来查看当前的 ip 地址:　　**ip addr**



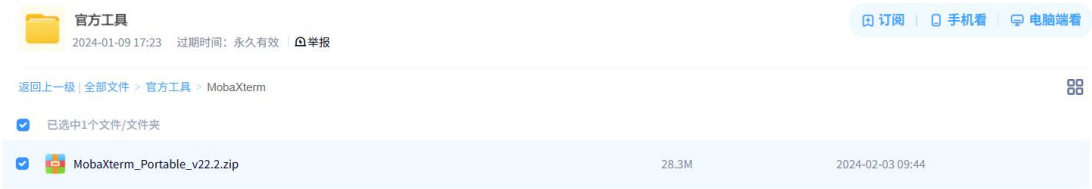
这里 192 开头的就是 ip 地址了，也是我们后面需要远程连接使用的地址，请记住它。

二、远程连接 SSH 与 VNC

先来说为什么要这样做: 每次香橙派启动如果都需要我们单独连接一块显示屏以及鼠标键盘给它会非常不方便，因此我们需要给它正确的配置，之后每次启动都可以通过远程连接被我们在 PC 上操控，这样就十分方便了。

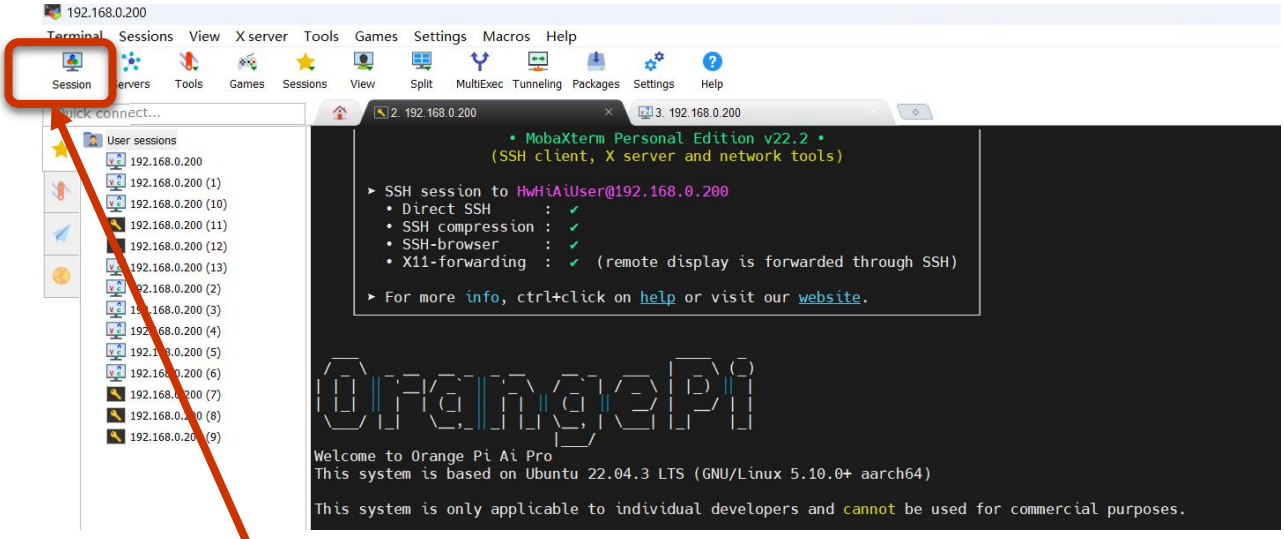
首先，我们需要安装远程连接工具 MobaXterm，链接放在这里:

官方工具_免费高速下载|百度网盘-分享无限制



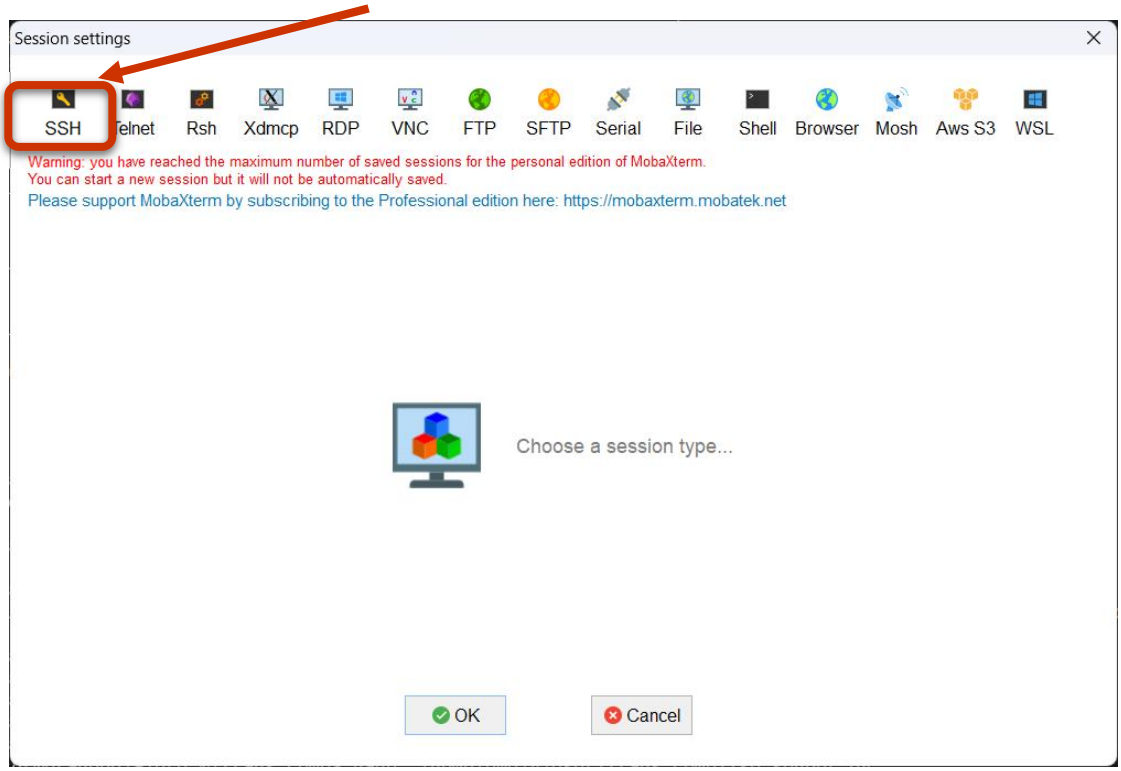
下载，解压，安装，这些步骤很简单，在这里直接略过。

打开这个软件，界面如下:

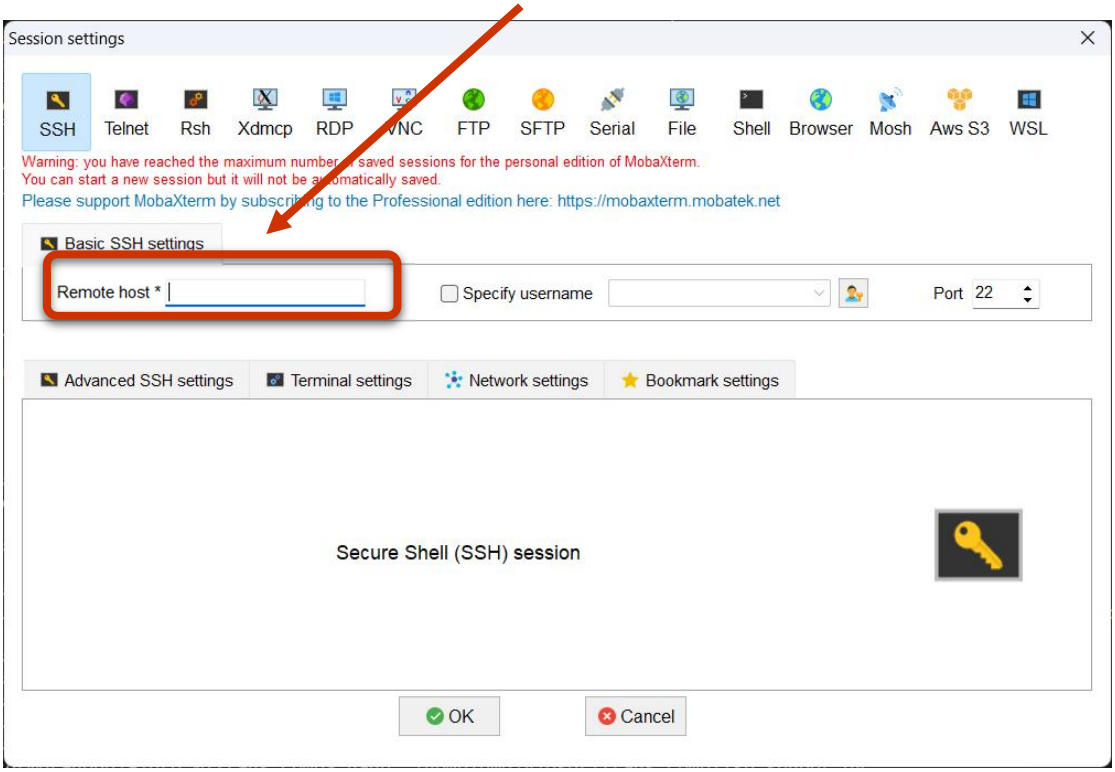


(这里我的 SSH 已经连接了，请忽略这个黑色区域的显示)

选择最上方的 Session，选择 SSH:



接着会看到需要输入 “Remote host”，这里就需要输入前面查到的 ip 地址:



“Specify usename”不需要填写， 后面的 Port 保持默认值“22”即可

然后点击 OK:

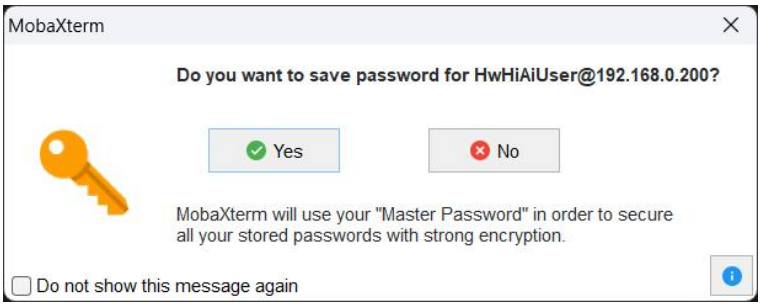
然后弹出“login as :”这里输入用户名 **HwHiAiUser** (注意大小写)

[OpenEuler 系统的用户名和密码都是 OpenEuler，同样注意大小写]

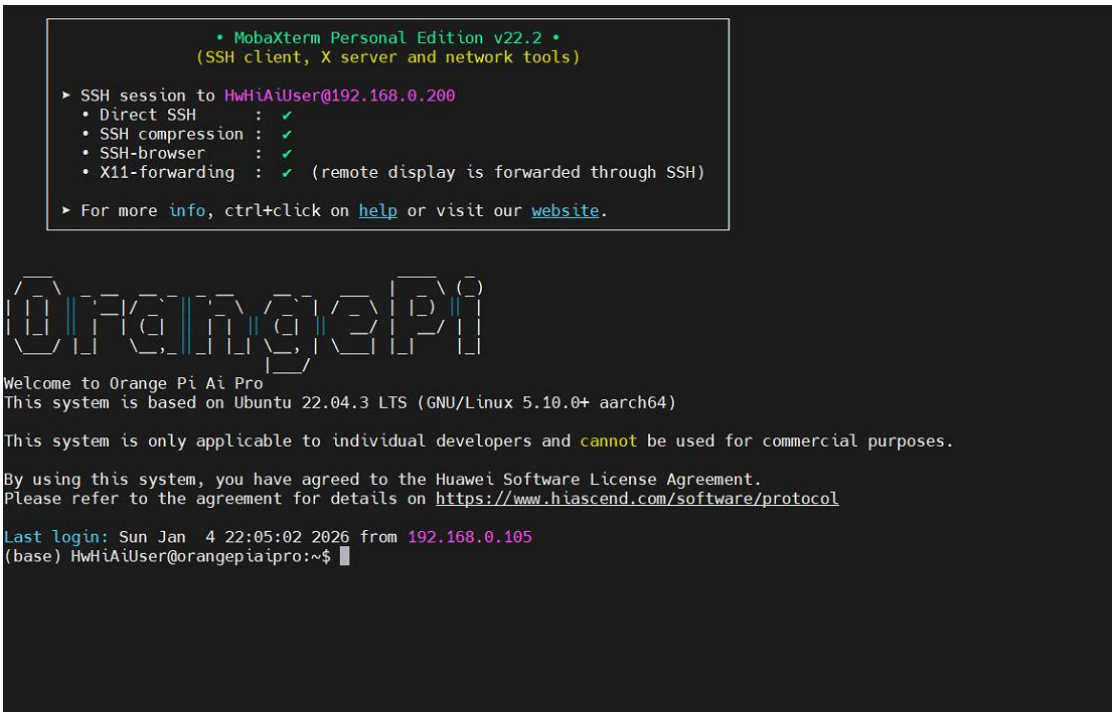
接着需要输入密码: **Mind@123**

注意 linux 输入密码是看不到的，所以需要认真地在键盘上输入这个密码，然后回车

如果用户名和密码都正确了就会弹出这个框:



选择“Yes”或者“No”均可。



然后就可以看到这个界面了，代表 SSH 远程连接成功了。

之后来做 VNC(可视化的远程连接，简单来说，就是远程操控图形化桌面):

在 SSH 连接的终端输入以下指令: vncserver :1 -localhost no

```
(base) HwHiAiUser@orangepiaipro:~$ vncserver :1 -localhost no
```

请注意这里的空格和字符

这个指令的意思是启动 vnc 服务，使用 1 号端口，本地自连接关闭。

为什么要这么写: (1)首先 0 号端口默认是连接在主板 HDMI 接口上的物理显示器

(2) -localhost no 目的是显式告诉 VNC 服务器，可以允许外部的设备连接到香橙派

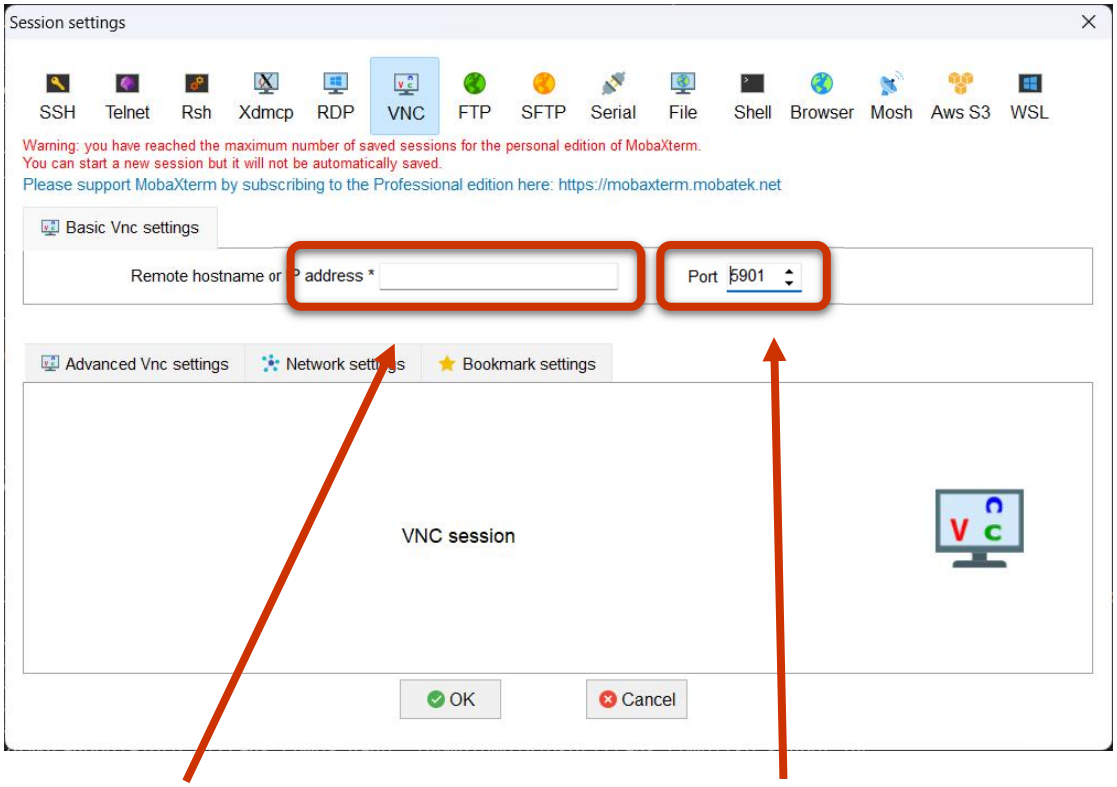
```
(base) HwHiAiUser@orangepiaipro:~$ vncserver :1 -localhost no

New Xtigervnc server 'orangepiaipro:1 (HwHiAiUser)' on port 5901 for display :1.
Use xtigervncviewer -SecurityTypes VncAuth,TLSVnc -passwd /home/HwHiAiUser/.vnc/passwd orangepiaipro:1 to connect to the VNC server.
```

之后会显示已经启动了 5901 端口(就是前面的 “:1”)

然后再次点击左上角的 Session->VNC:

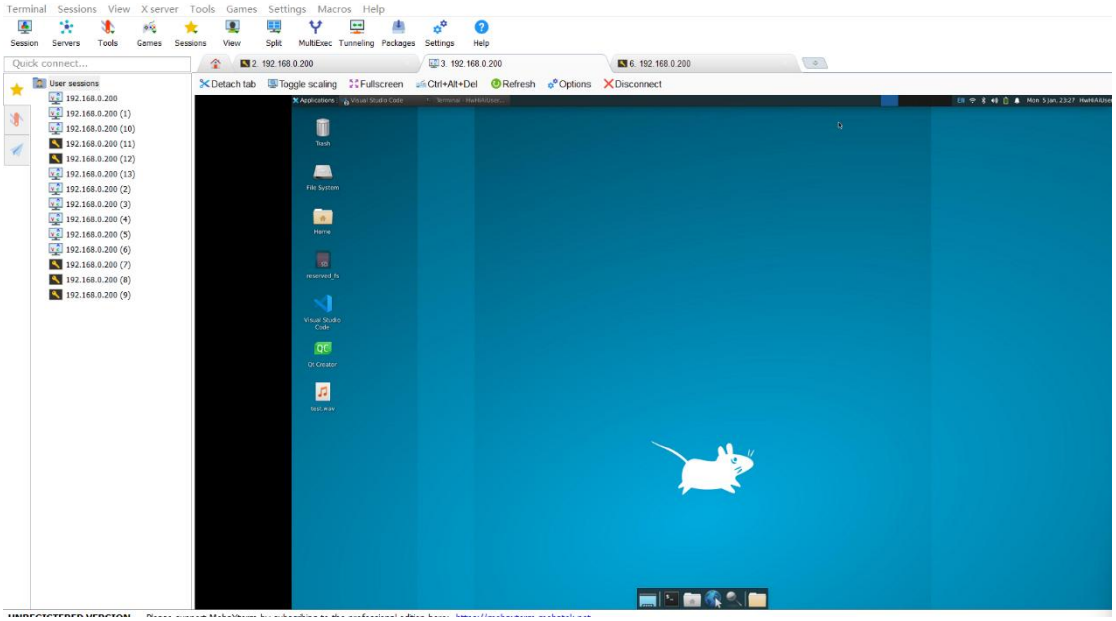
弹出的界面如下:



在这里输入 ip 地址(和前面的一样)，注意这里的 Port 一定要选 5901(遵从前面的设置)否则是连不上的。

然后点击 OK，剩下的步骤就和 SSH 一样了(用户名、密码……)

之后进入图形化桌面:



为什么我这里的桌面和默认的桌面不一致:

这里我选择了 XFCE 轻量化的桌面，使用默认的桌面显示渲染会更复杂，实践证明默认的桌面容易在运行时崩溃。

建议在配置中设置为 XFCE 桌面，运行起来更流畅：

右键鼠标->Open Terminal->输入指令 **nano ~/.vnc/xstartup**

(如果提示"command not found"之类的语句，请使用 **sudo apt install nano** 安装 nano)

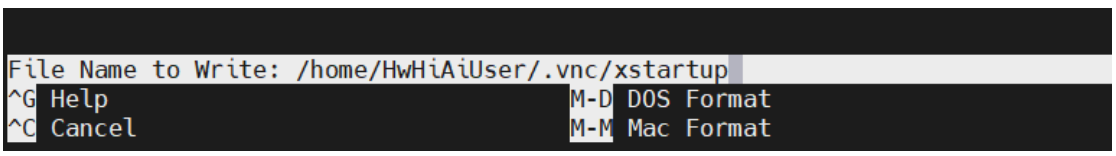
使用 nano 打开 vnc 的配置文件，接下来非常重要，请按照下面的文件修改:

```
#!/bin/sh
unset SESSION_MANAGER
unset DBUS_SESSION_BUS_ADDRESS

# 这一行非常重要，不仅启动XFCE，还包含了dbus，且千万不要加 "&" 符号
dbus-launch --exit-with-session startxfce4
```

(PS:如果显示"command not found"之类的输出，请先安装 nano 编辑器: sudo apt install nano)

修改好文件之后-> Ctrl + O(英文字母不是数字) 进行保存



下方会提示是否保存: 之后回车 -> Crtl + X 退出即可。

三、基本开发软件安装

这里以 VS 和 QT5 为例(vscode 应该是镜像中预装过的):

3.1 VS 安装:

(1) wget https://update.code.visualstudio.com/latest/linux-deb-arm64/stable -O code_arm64.deb

(2) sudo dpkg -i code_arm64.deb

3.2 QT 安装:

sudo apt install build-essential qtcreator qtbase5-dev qt5-qmake #Ubuntu 的官方软件源里已经包含了适配 ARM64 的 Qt Creator,一键安装即可

)过程中如果遇到如[Y/n]输入 Y 之后等待一段时间安装完成即可)

之后会发现在 VNC 桌面里找不到这两个软件的图标，这不是没装成功，而是需要我们手动给虚拟桌面添加上快捷方式:

在 VNC 打开一个终端，执行:

cp /usr/share/applications/code.desktop ~/Desktop/ # 复制 VS Code 图标到桌面

cp /usr/share/applications/org.qt-project.qtcreator.desktop ~/Desktop/ # 复制 Qt Creator 图标到桌面

然后是使用的问题: 双击图标打开软件发现没反应，不要怕，在 VNC 这种“虚拟显卡”环境下，它会因为找不到 GPU 硬件加速而直接崩溃。我们修改以下配置即可:

nano ~/Desktop/code.desktop #打开设置文件

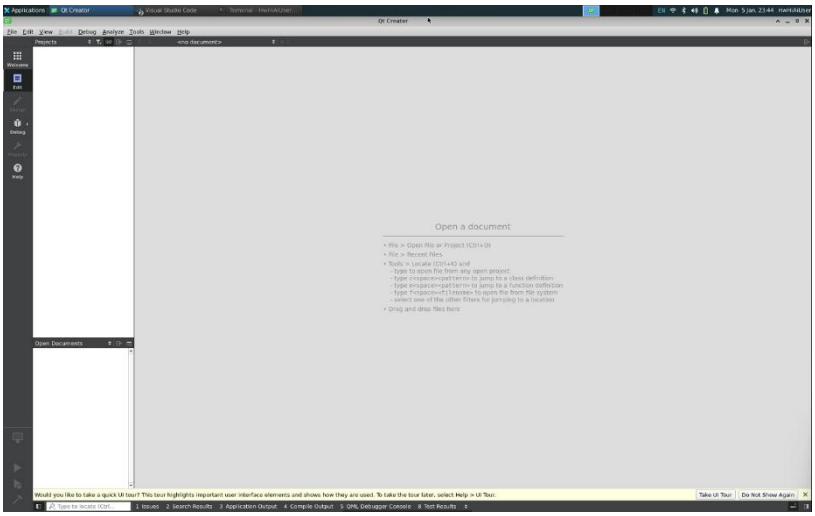
找到这一行:

Exec=……

修改为:

Exec=/usr/share/code/code --no-sandbox --unity-launch %F

双击图标打开就可以使用了



四、关于外设的使用说明

1.关于摄像头

学校提供或者自己买的官方摄像头(这里我是用的是官方的摄像头)都是 USB 连接的，直接插入即可使用，不需要驱动。

关键问题: 关于索引的查找, 在编写脚本使用摄像头时(以 Py 为例)，需要设置正确的索引否则不能使用摄像头捕捉画面

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

如图，我只接入一个摄像头却显示了 4 个索引，这是由于 Index 0 负责视频流，Index 1 负责统计信息、曝光参数控制等

因此建议在使用前先建一个脚本找到正确的索引，不然一个一个试太麻烦了，以下给出一段代码参考:

```python
import cv2

print(">>> 正在扫描可用摄像头索引 (0-3)...")

# 遍历索引 0 到 3
for index in range(4):  #这里具体是多少改数字即可
    cap = cv2.VideoCapture(index)
    if cap.isOpened():
        ret, frame = cap.read()
        if ret:
            print(f"[√] 成功找到摄像头: 设备索引 {index}")
            #print(f"    分辨率: {int(cap.get(3))} x {int(cap.get(4))}")
            # 释放资源
```

```
            cap.release()
        else:
            print(f"[!] 设备 {index} 打开了，但无法读取画面 (可能是系统占
用)")
    else:
        print(f"[x] 设备 {index} 无法打开")

print(">>> 扫描结束")
```

一般是 0 或者 2 的概率大些

2.关于声卡

学校提供的声卡，可以当作简易的音响来使用

以下给出玩法:

连接后在终端输入指令: **aplay -l**

```
(base) HwHiAiUser@orangepiaipro:~/trans_comp$ aplay -l
**** List of PLAYBACK Hardware Devices ****
card 0: UACDemoV10 [UACDemoV1.0], device 0: USB Audio [USB Audio]
  Subdevices: 1/1
  Subdevice #0: subdevice #0
card 2: ascend310b [ascend310b], device 0: ascend310b-playback ascend310b-hifi-0 []
  Subdevices: 1/1
  Subdevice #0: subdevice #0
```

以这里为例，那个带 USB 的就是外接的声卡。

ascend310b 是自带的音频处理单元，主要用于: HDMI 音频输出/板载音频接口(给中间那个耳机孔用的)

确定好了，这里的 card 是 0，输入以下指令:

aplay -D hw:0,0 test.wav (需要先下载一个音频文件，保存为 test.wav)

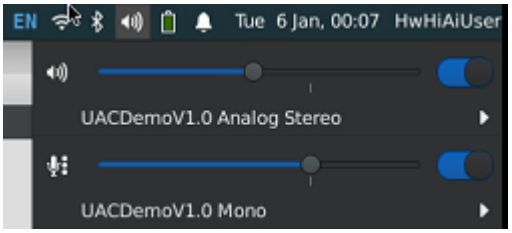
如果觉得这样麻烦可以直接本地录音一段音频然后自动播放:

arecord -D hw:0,0 -d 5 -c 1 -f S16_LE -r 44100 test.wav #把这段录频保存为 test.wav

aplay test.wav #声卡播放

这样就能听到录音了

关于音量大小调节:



右上角的音量键，点开能看到两个滑块，分别是音量大小以及录音的灵敏度

四、关于 NPU 的使用

首先，我们可以使用指令来查看 NPU 的状态: **npu-smi info**

```
(base) HwHiAiUser@orangepiaipro:~$ npu-smi info
+------------------------------------------------------------------------------------------------+
| npu-smi 25.2.0                                 Version: 25.2.0                                 |
+------------------------------+-----------------+------------------------------------------------------+
| NPU     Name                 | Health          | Power(W)     Temp(C)           Hugepages-Usage(page) |
| Chip    Device               | Bus-Id          | AICore(%)    Memory-Usage(MB)                        |
+==============================+=================+======================================================+
| 0       310B4                | Alarm           | 0.0          61                15    / 15            |
| 0       0                    | NA              | 0            2028 / 15610                              |
+==============================+=================+======================================================+
```

可以看到当前 NPU 温度为 61℃，使用率为 0，因为现在没运行任何使用 NPU 的程序。

使用这句指令检查是否预装了 ACL 库:

python3 -c "import acl; print('恭喜！ ACL 库加载成功！')"

我们烧录的镜像版本应该是预装过的，所以输出应该如下所示:

```
(base) HwHiAiUser@orangepiaipro:~$ python3 -c "import acl; print('恭喜! ACL 库加载成功! ')"
恭喜! ACL 库加载成功!
```

安装 opencv 图形库:

sudo apt install libgl1 libglib2.0-0 libsm6 libxrender1 libxext6

安装 opencv Python 包:

pip install opencv-python

安装好之后输入以下指令检查是否安装成功:

python3 -c "import cv2; print('OpenCV version:', cv2. version)"

```
(base) HwHiAiUser@orangepiaipro:~$ python3 -c "import cv2; print('OpenCV version:', cv2.__version__)"
OpenCV version: 4.12.0
```

正确的话应该返回版本号

准备好了环境，开始我们的 NPU 使用:

寻找到预装的推理文件:

d /opt/opi_test/ResnetPicture/scripts #切换到样例文件夹

ls -F #输出对应目录下的内容

```
(base) HwHiAiUser@orangepiaipro:/opt/opi_test/ResnetPicture/scripts$ ls -F
sample_build.sh*  sample_run.sh*
```

接下来编译文件(这是 C++文件，后面的视频推理我们用 python 来跑):

bash sample_build.sh

```
(base) root@orangepiaipro:/opt/opi_test/ResnetPicture/scripts# bash sample_build.sh
[INFO] Sample preparation
-- The C compiler identification is GNU 11.4.0
-- The CXX compiler identification is GNU 11.4.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/g++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- set default INC_PATH: /usr/local/Ascend/ascend-toolkit/latest
-- set default LIB_PATH: /usr/local/Ascend/ascend-toolkit/latest/runtime/lib64/stub
-- Configuring done
-- Generating done
-- Build files have been written to: /opt/opi_test/ResnetPicture/build/intermediates/host
[ 50%] Building CXX object CMakeFiles/main.dir/main.cpp.o
[100%] Linking CXX executable /opt/opi_test/ResnetPicture/out/main
[100%] Built target main
[INFO] Sample preparation is complete
```

然后运行推理文件：

bash sample_run.sh

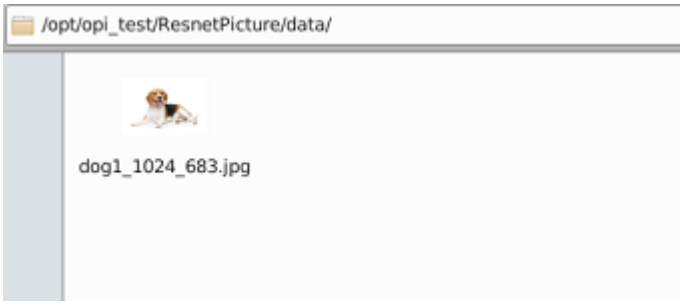
```
(base) root@orangepiaipro:/opt/opi_test/ResnetPicture/scripts# bash sample_run.sh
[INFO] The sample starts to run
[INFO] InitACLResource success.
[INFO] Init dvpp resource success.
[INFO] top 1: index[162] value[0.905956] class[beagle]
[INFO] top 3: index[166] value[0.000758] class[Walker hound,Walker foxhound]
[INFO] top 4: index[167] value[0.000559] class[English foxhound]
[INFO] top 5: index[163] value[0.000076] class[bloodhound sleuthhound]
[INFO] Unload model ../model/resnet50.om success
[INFO] The program runs successfully
```

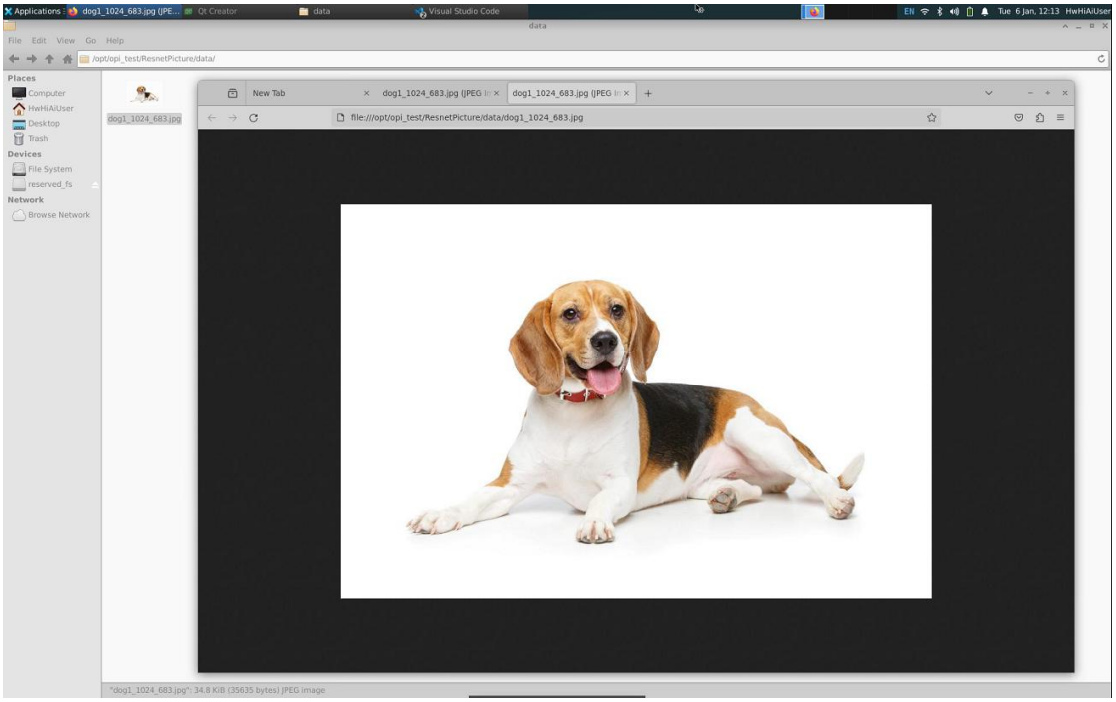
这样，第一个 NPU 的推理就跑通了

我们来看看这个结果:

模型认为，可能性最高的是: 0.905956 的概率是 比格犬

我们来到 VNC 终端看看这张图片到底是不是这样的:





可见，推理结果完全正确。这就是一张比格犬的图片。

接下来，我们来玩玩摄像头+yolo+NPU 推理实时的视频流；

先来克隆项目:

cd ${HOME}

git clone https://gitee.com/ascend/EdgeAndRobotics.git

cd EdgeAndRobotics/Samples/YOLOV5USBCamera/python/model

下面这是一整个指令中间不要断开

wget https://obs-9be7.obs.cn-east-2.myhuaweicloud.com/003_Atc_Models/yolov5s/yolov5s_nms.onnx --no-check-certificate

###上面这是一整个指令中间不要断开

下面这是一整个指令中间不要断开

wget https://obs-9be7.obs.cn-east-2.myhuaweicloud.com/003_Atc_Models/yolov5s/aipp_rgb.cfg --no-check-certificate

###上面这是一整个指令中间不要断开

克隆 ATCLite：

sudo apt-get install -y ffmpeg libavcodec-dev libavformat-dev libavdevice-dev libavutil-dev libswscale-dev

cd ~

git clone https://gitee.com/ascend/ACLLite.git

cd ACLLite

bash build_so.sh

nano ~/.bashrc

export PYTHONPATH=/root/ACLLite/python:$PYTHONPATH #在最后一行添加

Ctrl+O->回车，保存后 Ctrl+X 退出

source ~/.bashrc

转换模型(模型是不能直接被 NPU 接受的，需要转换成 NPU 能识别的.om 格式):

下面这是一整个指令中间不要断开

atc --model=yolov5s_nms.onnx --framework=5 --output=yolov5s_rgb --input_shape="images:1,3,640,640;img_info:1,4" --soc_version=Ascend310B4 --insert_op_conf=aipp_rgb.cfg

###上面这是一整个指令中间不要断开

导入 cv 库:

cd ~/EdgeAndRobotics/Samples/YOLOV5USBCamera/python/src

nano YOLOV5USBCamera.py

在开头找到这一行:

import videocapture as video

修改为:

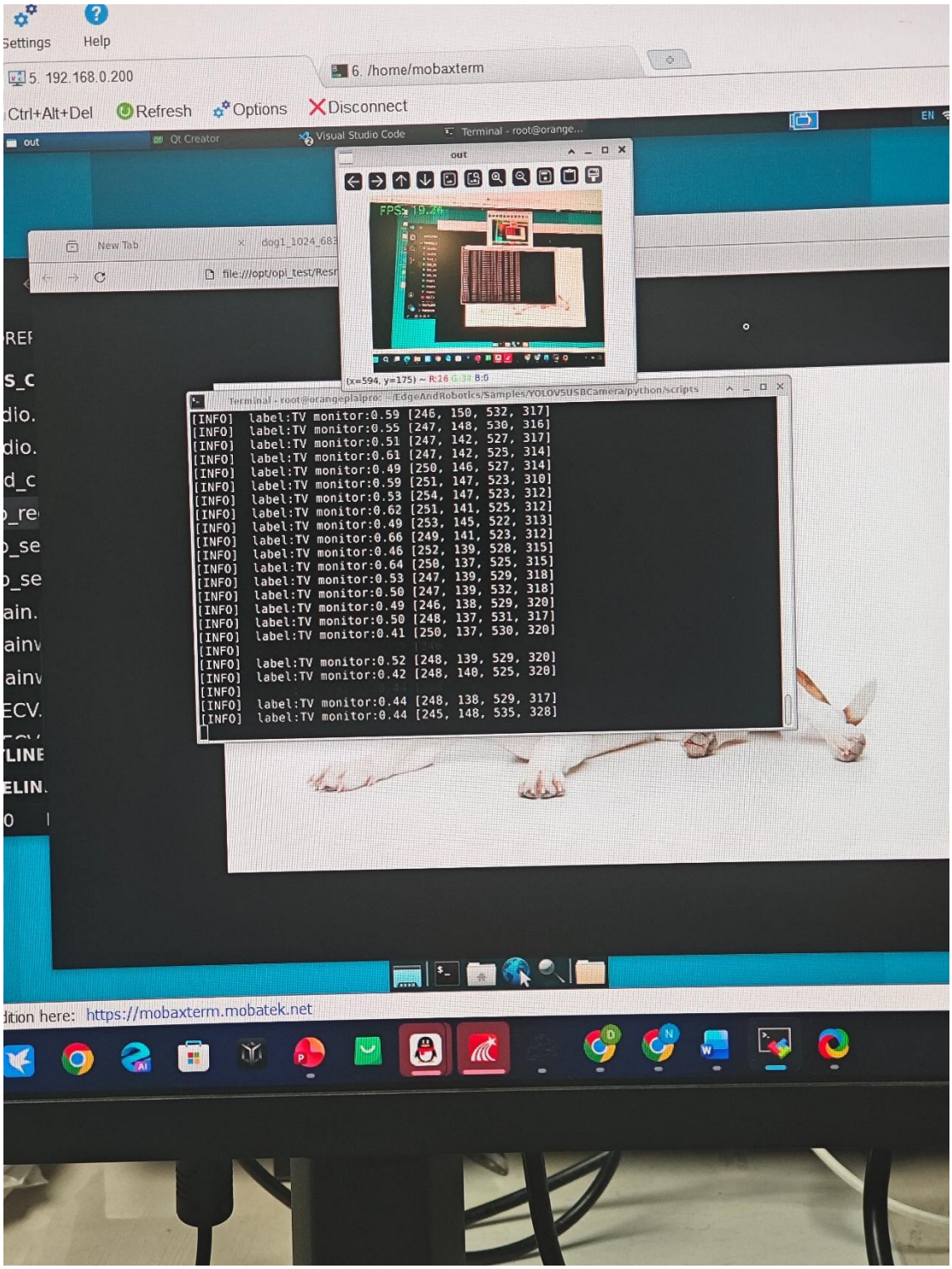
import cv2 as video

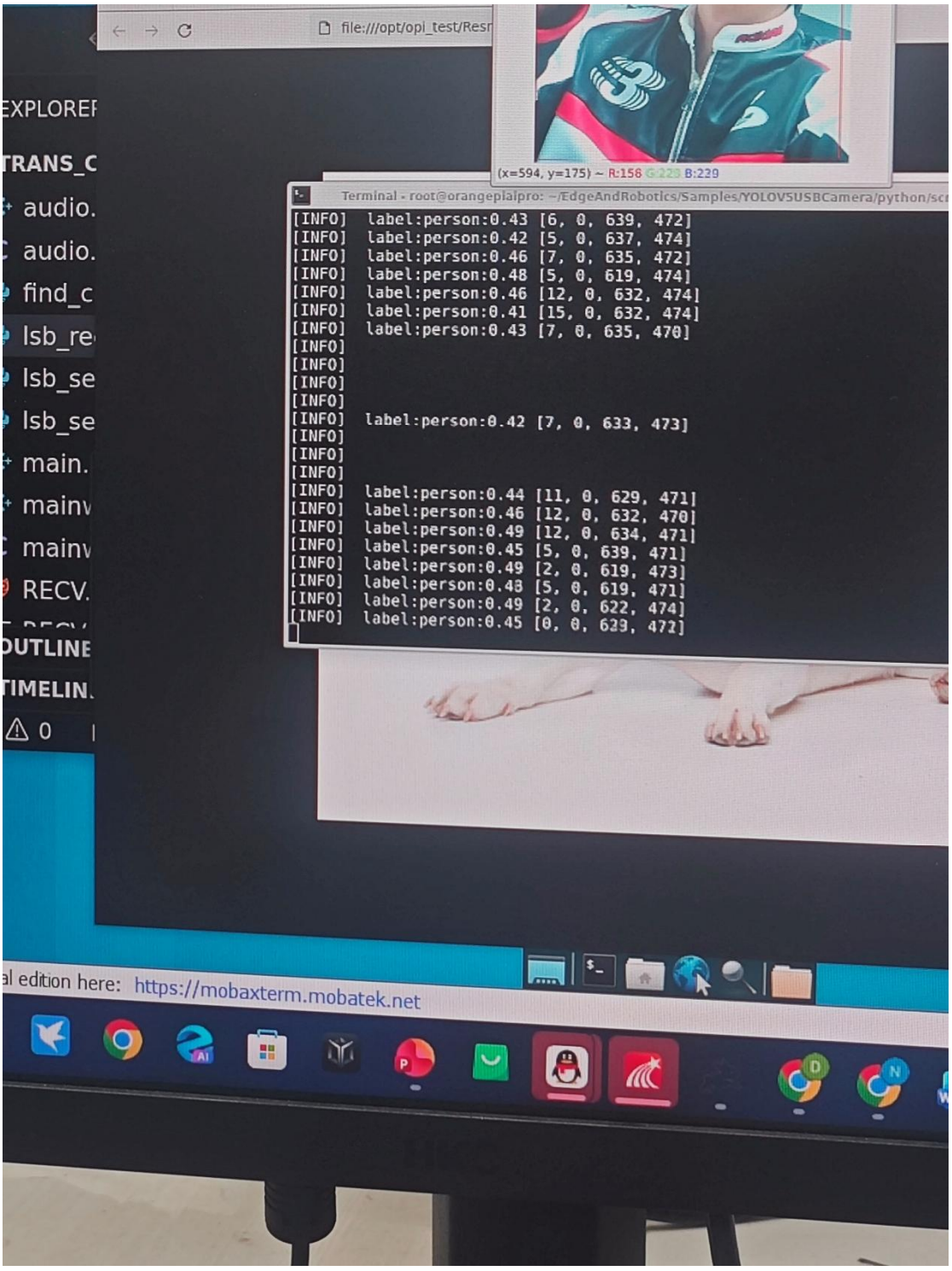
cd ~/EdgeAndRobotics/Samples/YOLOV5USBCamera/python/scripts

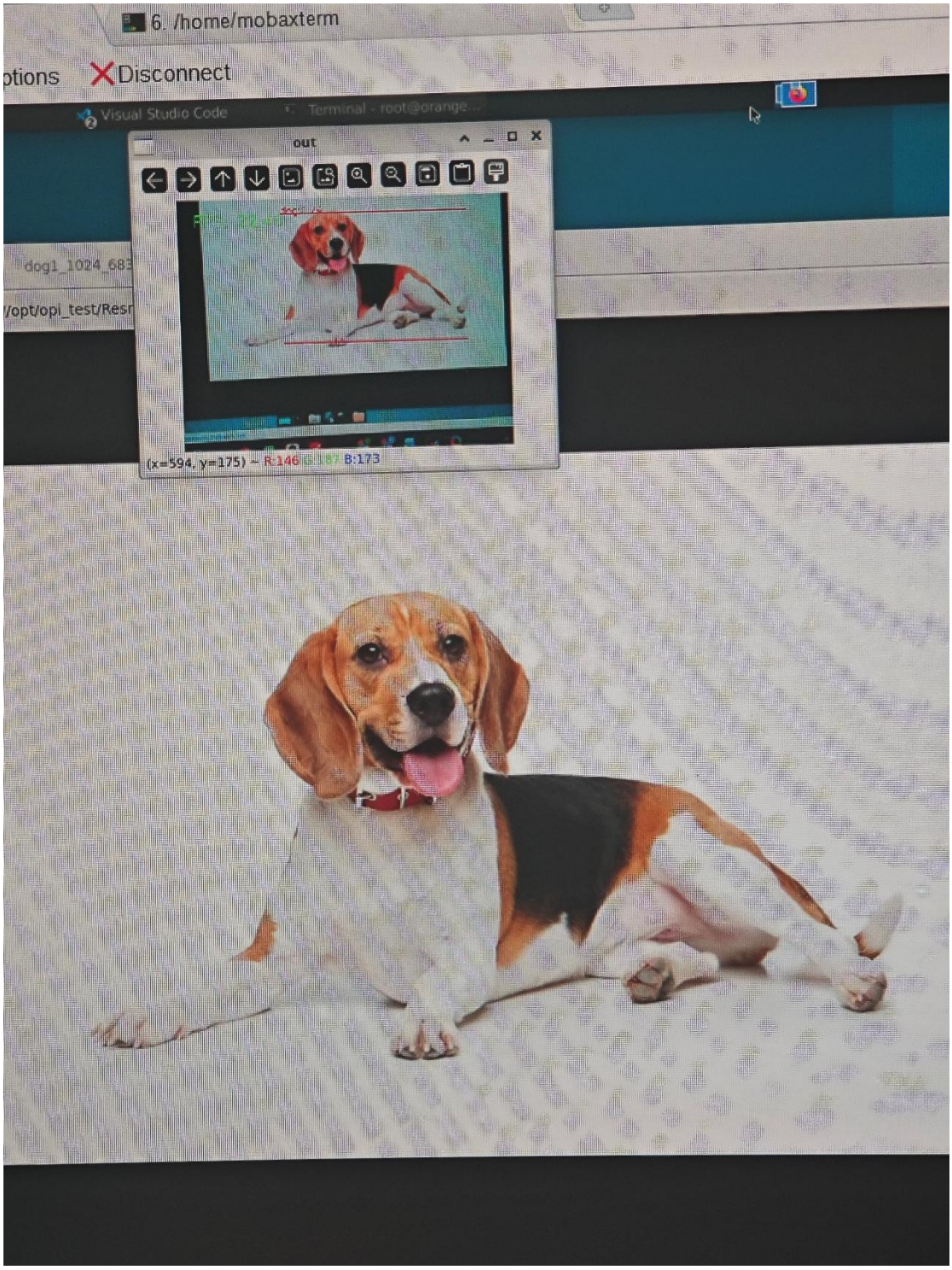
bash sample_run.sh

(要在 VNC 桌面，不然看不到实时画面效果，执行推理文件)

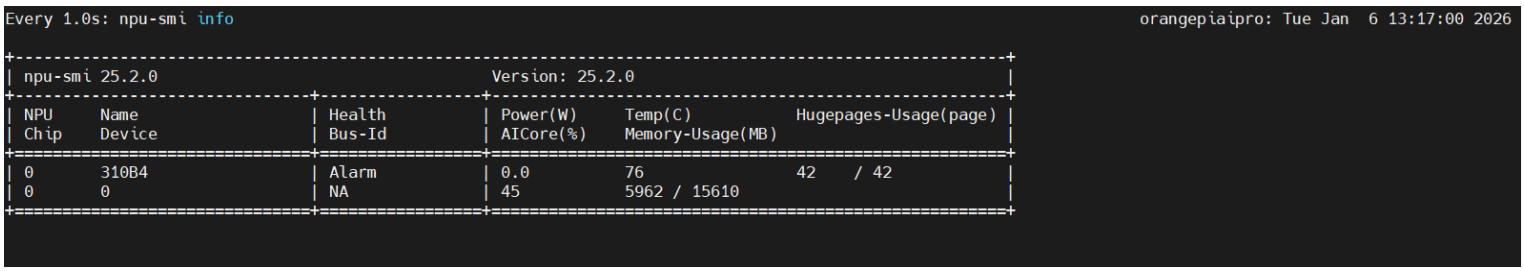
帧率基本在 20 左右，不卡顿:

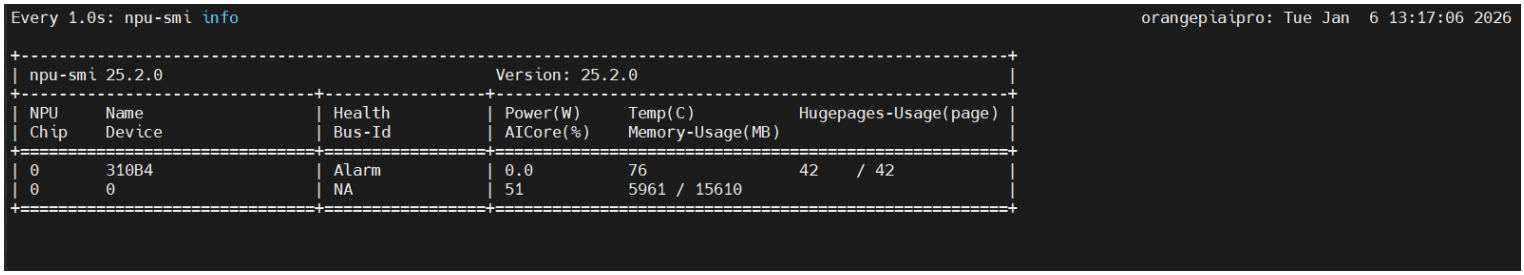


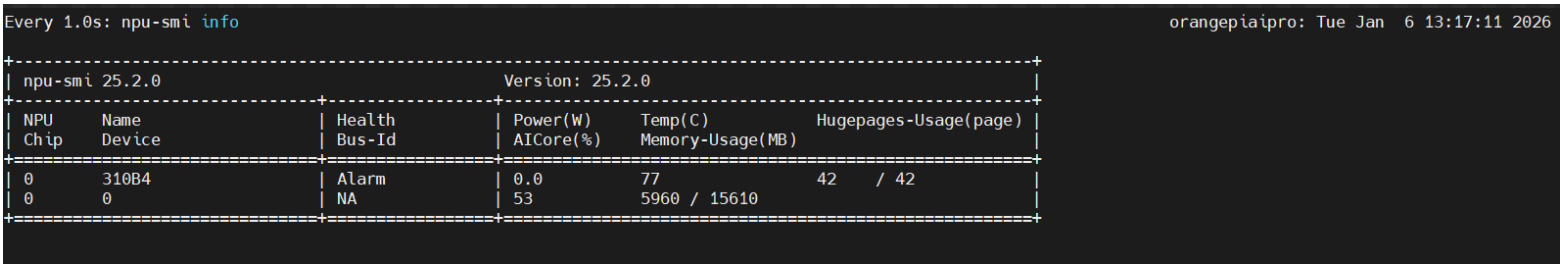

file:///opt/opi_test/Resr
(x=594, y=175) ~ R:158 G:229 B:229
Terminal - root@orangepiaipro: ~/EdgeAndRobotics/Samples/YOLOV5USBCamera/python/scr
[INFO] label:person:0.43 [6, 0, 639, 472]
[INFO] label:person:0.42 [5, 0, 637, 474]
[INFO] label:person:0.46 [7, 0, 635, 472]
[INFO] label:person:0.48 [5, 0, 619, 474]
[INFO] label:person:0.46 [12, 0, 632, 474]
[INFO] label:person:0.41 [15, 0, 632, 474]
[INFO] label:person:0.43 [7, 0, 635, 470]
[INFO]
[INFO]
[INFO]
[INFO]
[INFO] label:person:0.42 [7, 0, 633, 473]
[INFO]
[INFO]
[INFO]
[INFO] label:person:0.44 [11, 0, 629, 471]
[INFO] label:person:0.46 [12, 0, 632, 470]
[INFO] label:person:0.49 [12, 0, 634, 471]
[INFO] label:person:0.45 [5, 0, 639, 471]
[INFO] label:person:0.49 [2, 0, 619, 473]
[INFO] label:person:0.48 [5, 0, 619, 471]
[INFO] label:person:0.49 [2, 0, 622, 474]
[INFO] label:person:0.45 [0, 0, 629, 472]
EXPLORER
TRANS_C
audio.
audio.
find_c
lsb_re
lsb_se
lsb_se
main.
mainv
mainv
RECV.
OUTLINE
TIMELIN
0
al edition here: https://mobaxterm.mobatek.net



我们看看此时 NPU 的工作状态:







可以看到此时的 AI Core 已经从之前的 0 变成了 50%左右，说明 NPU 是在执行推理工作的。

五、课题设计

5.1 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。

5.2 GUI 设计

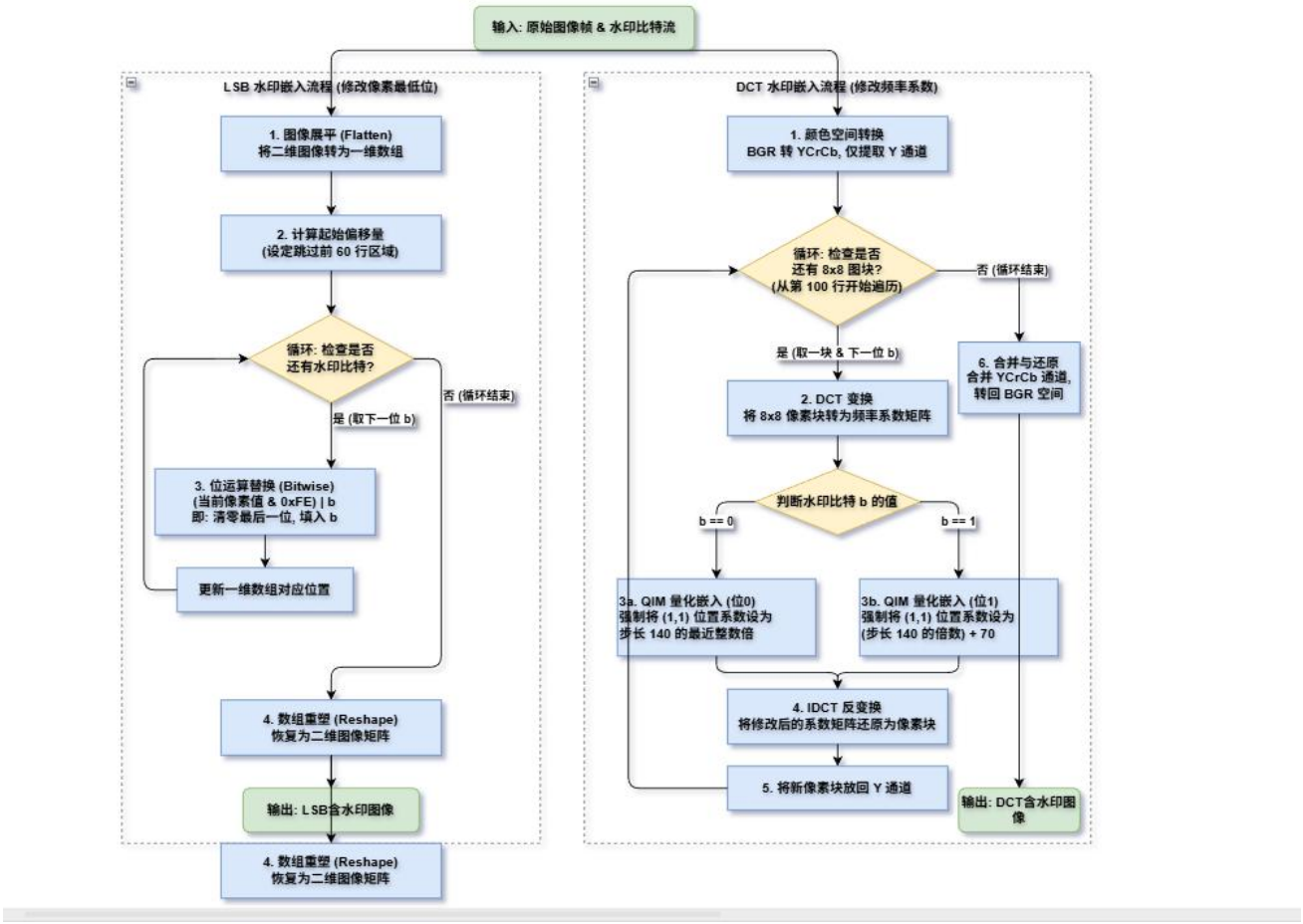
与在 PC 端方案不同，本设计面临着嵌入式平台严格的硬件资源限制。为了在有限算力下实现流畅的人机交互，本课题选用了 Qt 5.14 (Linux 版) 作为 GUI 框架，并进行了以下深度优化：

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。

5.3 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。

5.4 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。

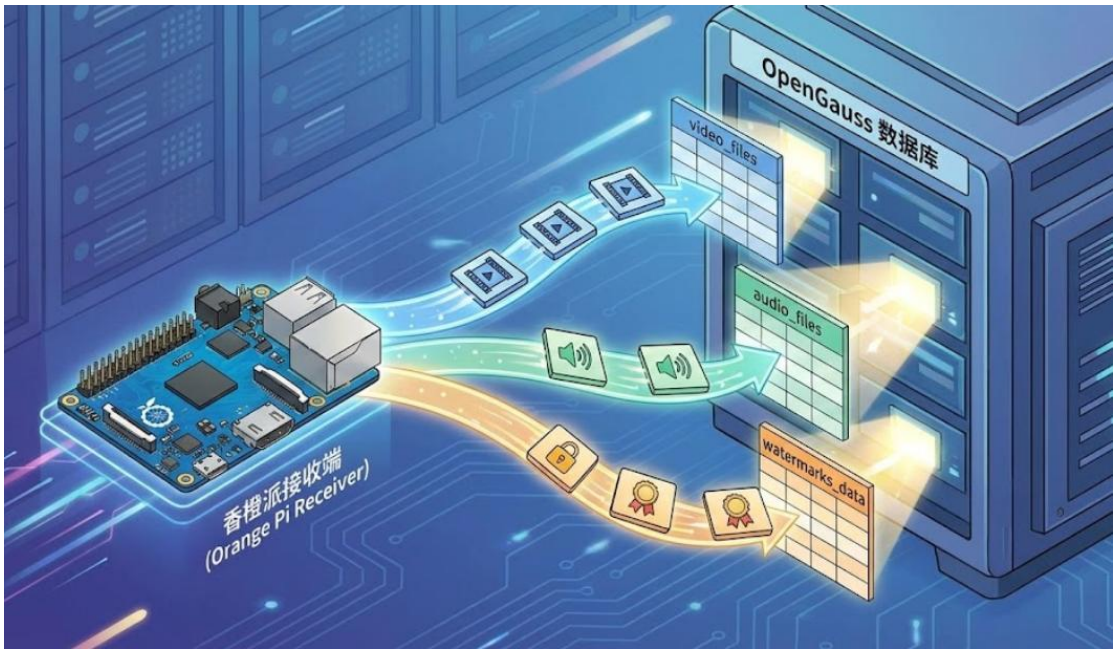
(2) TCP 信令同步机制

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。

5.5 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。

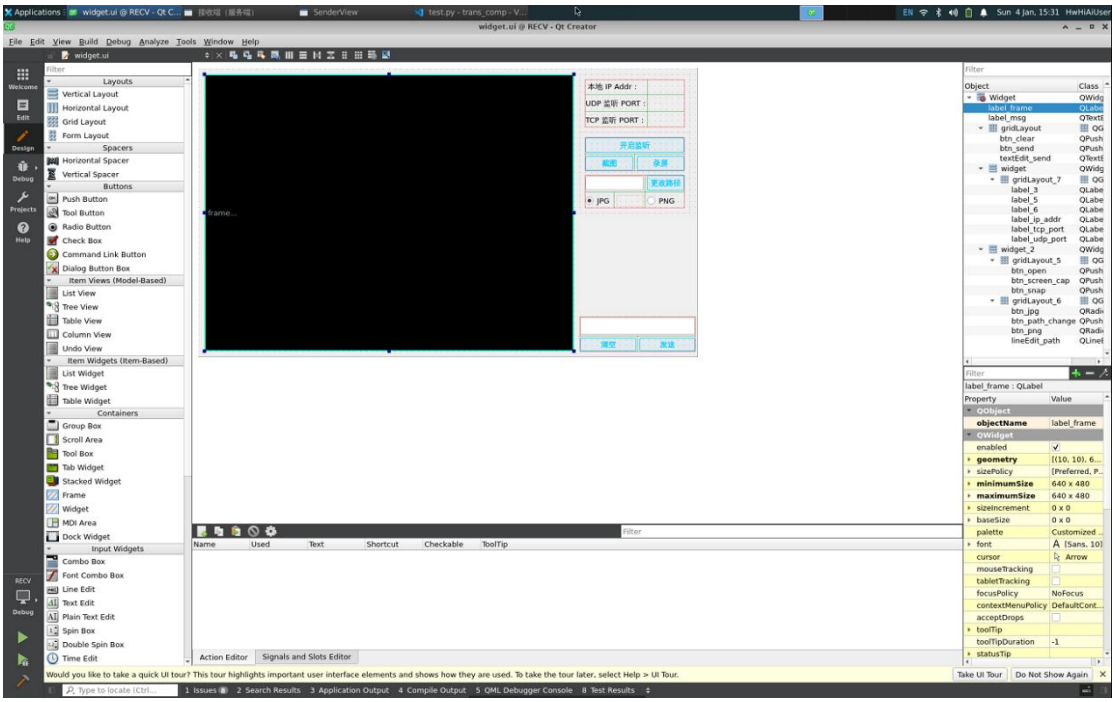
(2) 防注入与事务管理

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。

六、功能演示与系统测试

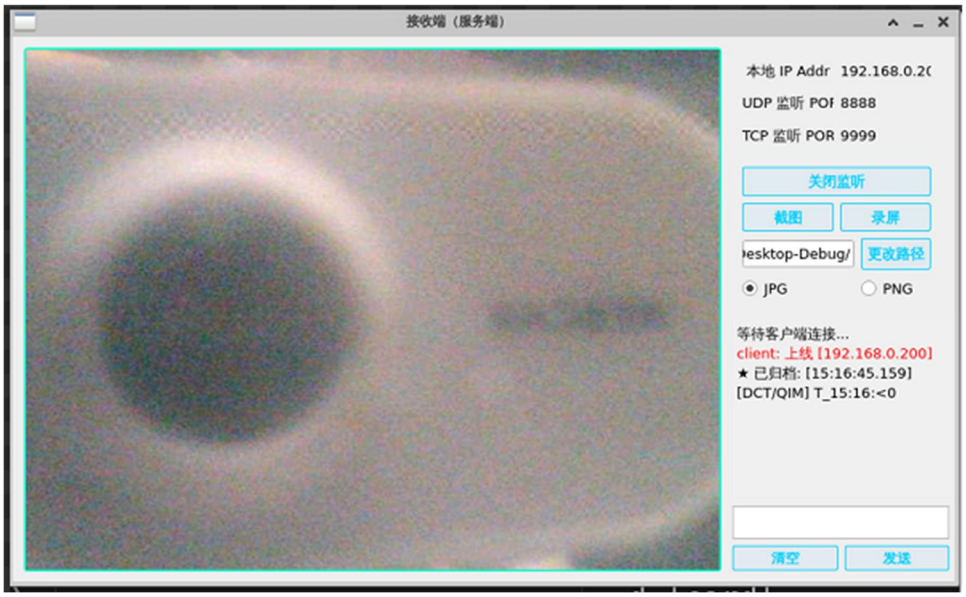
6.1 系统 UI 交互设计

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 6-1-1 UI 界面设计)

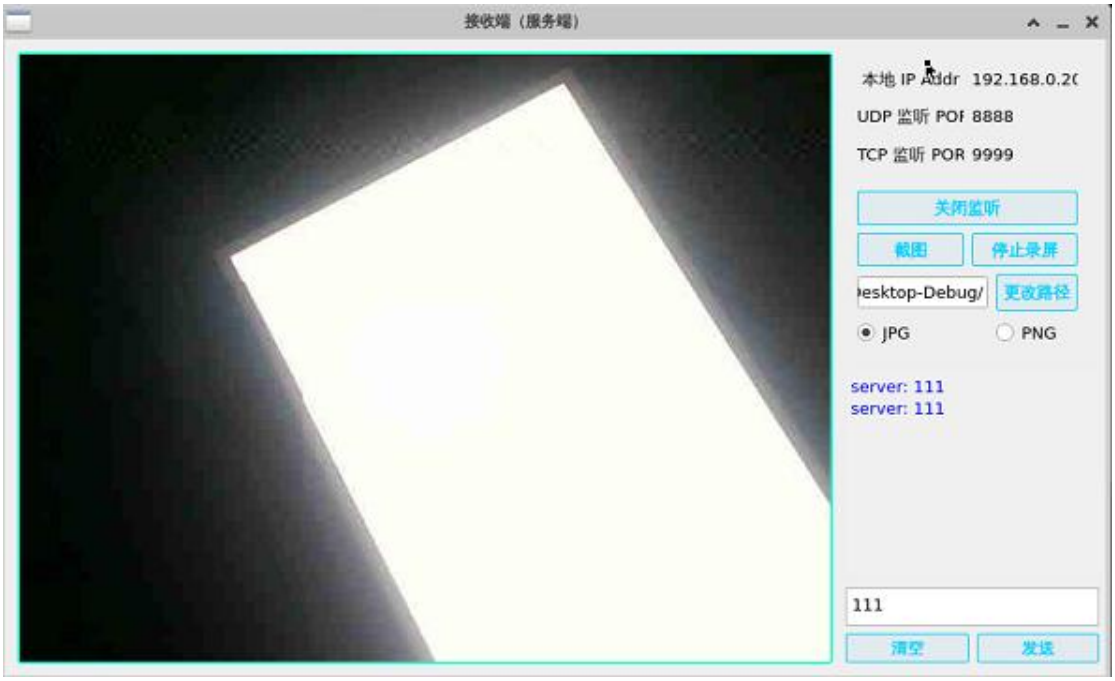
(1) 实时监控区（左侧）：使用 QLabel 控件作为视频渲染容器。通过重写 paintEvent 或直接使用 setPixmap 方法，将解码后的 QImage 帧逐帧绘制到界面上。



(图 6-1-2 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 6-1-3 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 6-1-4，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 6-1-4 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:

```
==============================
攻击: [0]正常 [1]模糊 [2]噪声 [3]遮挡 [4]亮度 [5]压缩
模式: [6]LSB [7]DCT | [q]退出
==============================
```

(图 6-1-5 发送端模式设置)

```
>>> 切换攻击: none
>>> 切换攻击: gaussian_blur
>>> 切换攻击: salt_pepper
>>> 切换攻击: occlusion
>>> 切换攻击: brightness
>>> 切换攻击: compression_sim
>>> LSB 水印
>>> DCT 水印
```

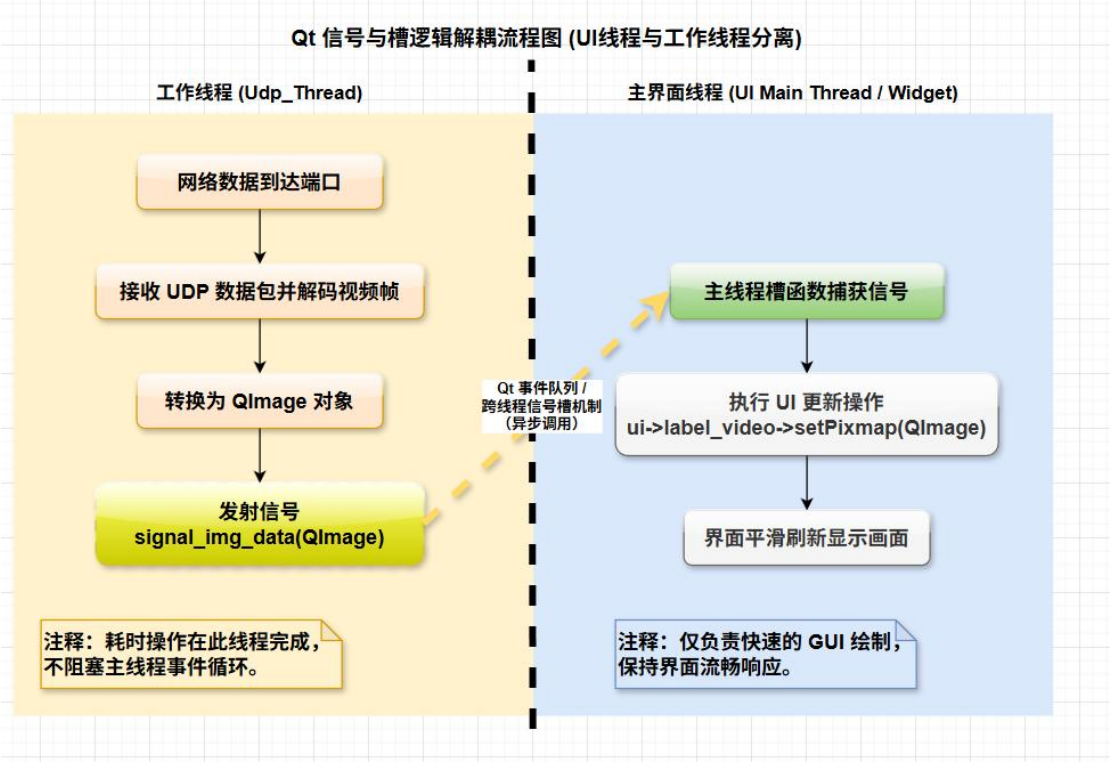
(图 6-1-6 键入后的显示结果)

6.1.2 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。



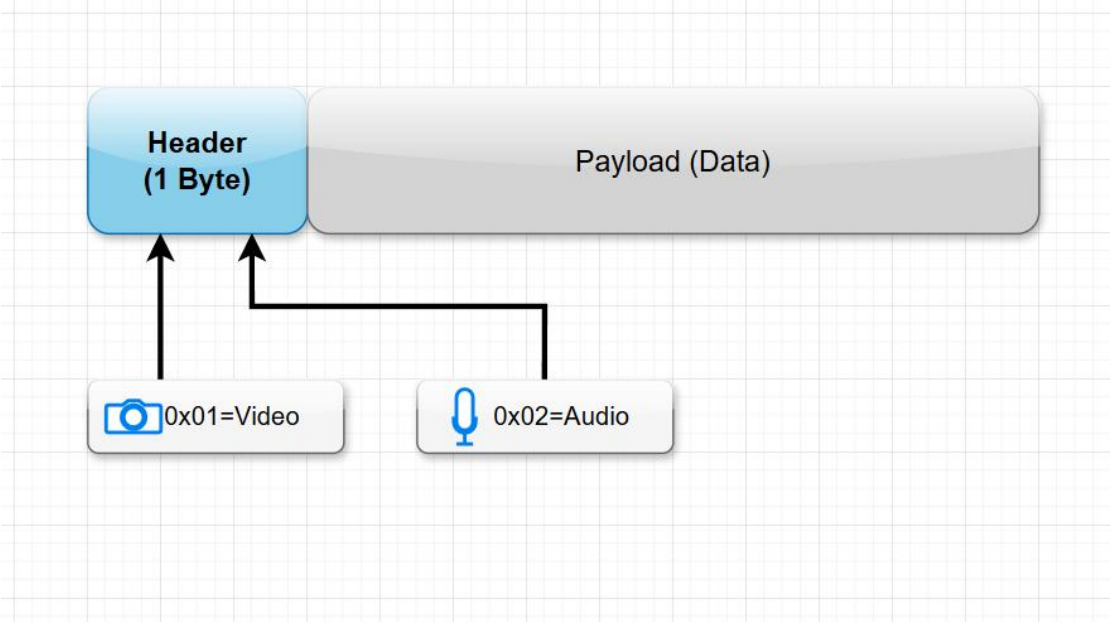
(图 6-1-7 Qt 信号与槽逻辑解耦流程)

6.2 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。

6.2.1 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 6-2-1 所示的帧结构：



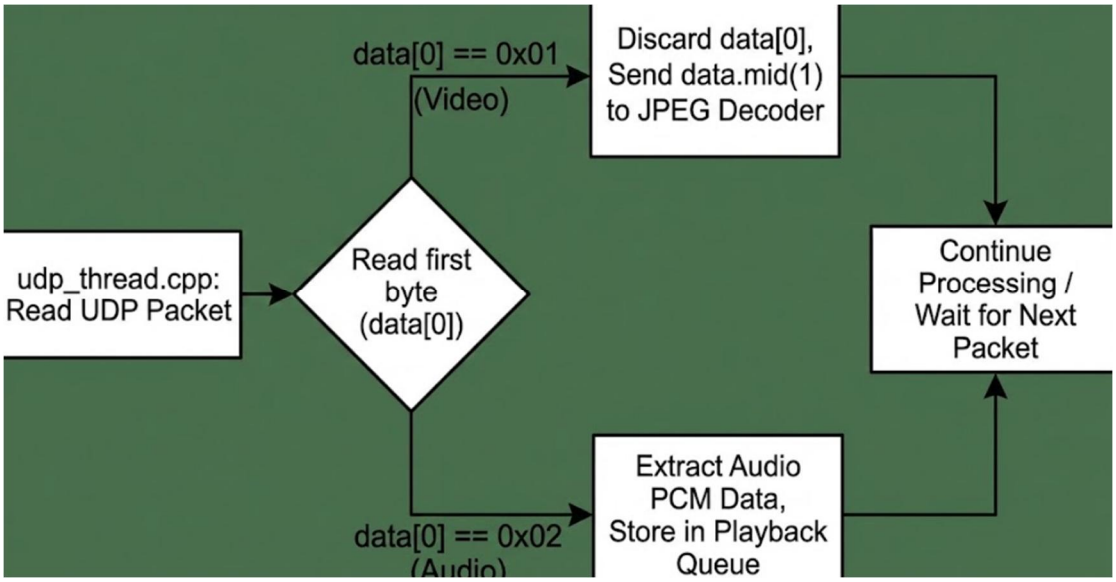
(图 6-2-1 帧头设置)

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。

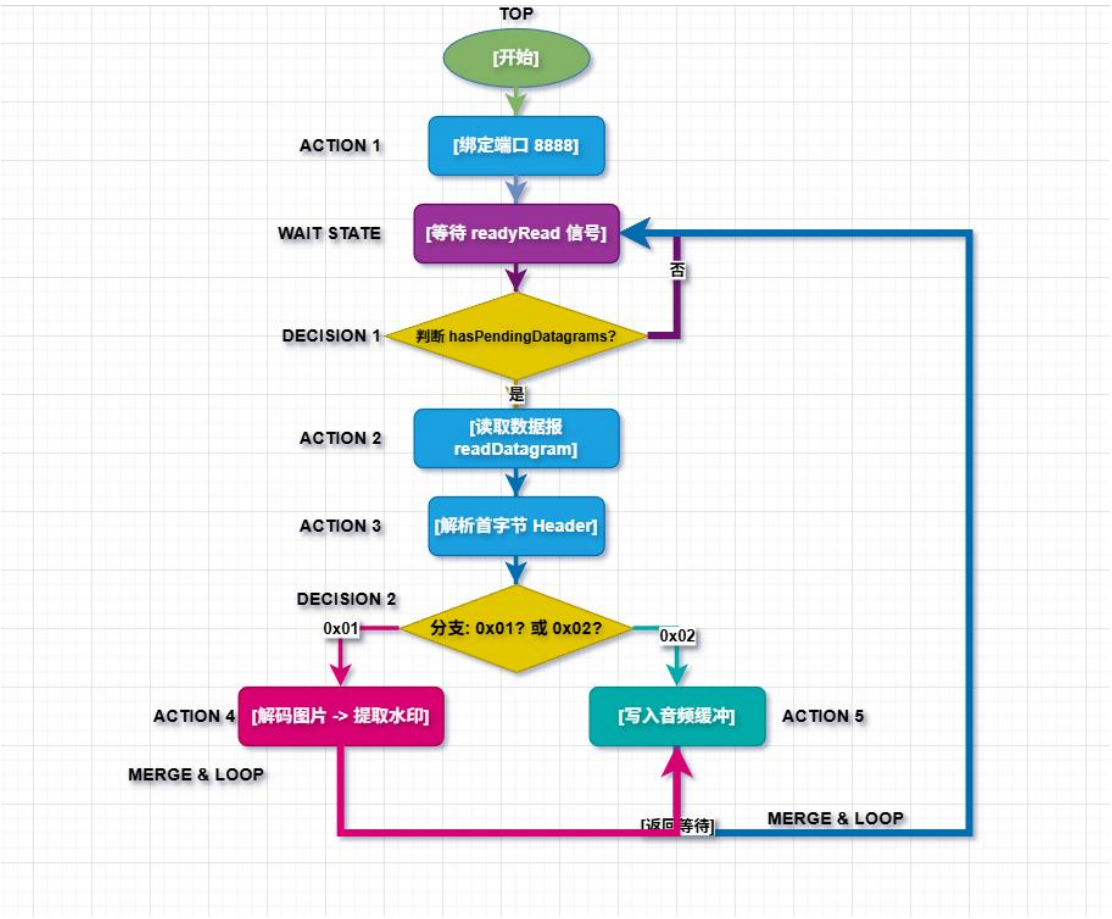
(3)若 data[0] == 0x02，则提取音频 PCM 数据存入播放队列。



(图 6-2-2 udp 解析流程)

6.2.2 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 6-2-3 所示：



(图 6-2-3 udp 接收线程逻辑)

6.2.3 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。

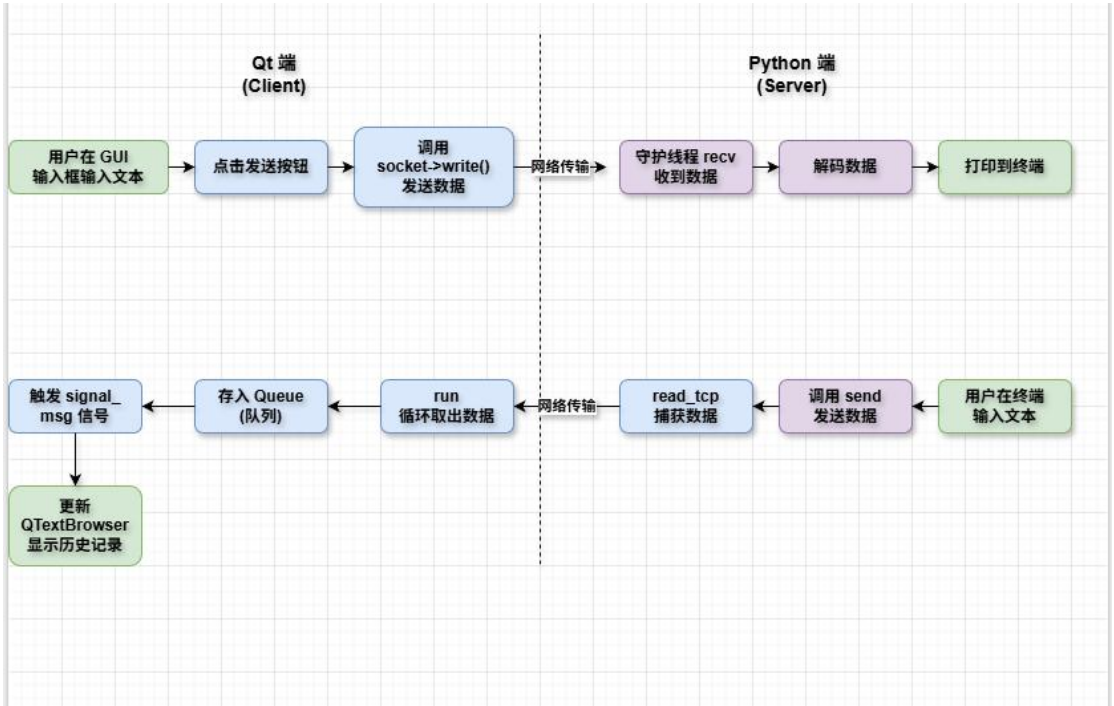
2. 监控端（Qt）的生产者-消费者模型

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 6-2-4 所示：



(图 6-2-4 文本交互流程)

6.3 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。

6.3.1 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。

6.3.2 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。

6.3.3 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 6-3 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
	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，此时质量因子设置为 80	提取率 0	提取率 100%	LSB 算法对图像压缩毫无抵抗力
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
	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	

				的比特（50%）被翻转了，50%的误码率 = 纯随机乱码。
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 6-3-1 对照组)

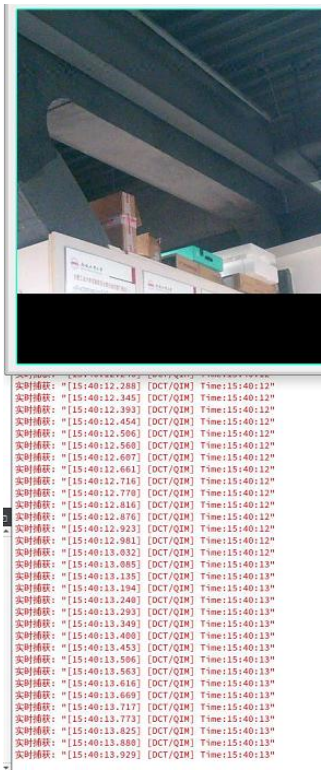
(2) 实验组—高斯噪声攻击(gaussian_noise)

(图 6-3-2 系数 sigema = 10)　　(图 6-3-3 系数 sigema = 15)　　(图 6-3-4 系数 sigema = 20)

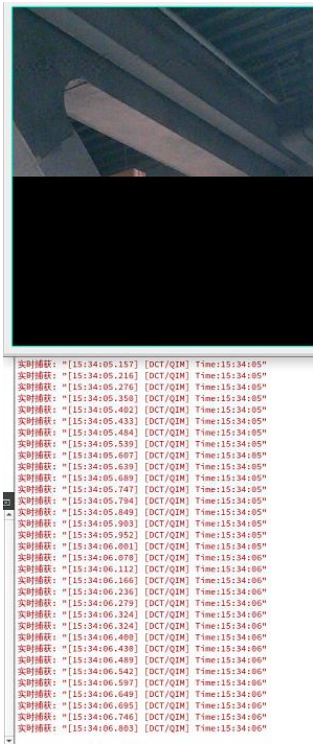
(3) 实验组--椒盐噪声(salt_pepper)

(图 6-3-5 校验噪声)

(4) 实验组—画面剪切(cropping)

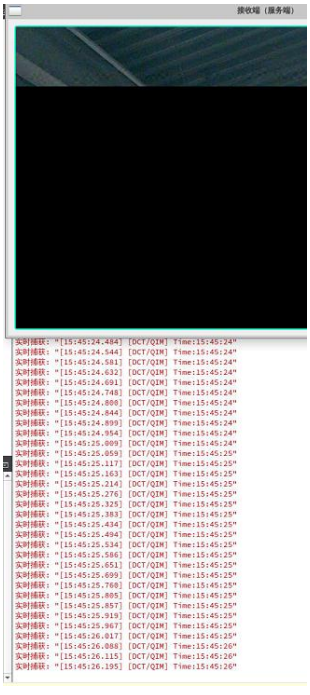


(图 6-3-6 遮挡 20%)



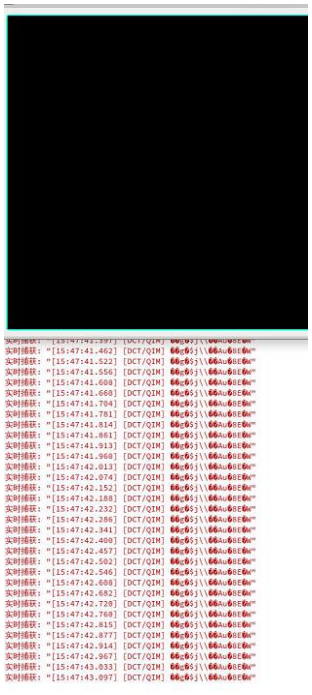
(图 6-3-7 遮挡 50%)

(图 6-3-8 遮挡 70%)



(图 6-3-9 遮挡 80%)

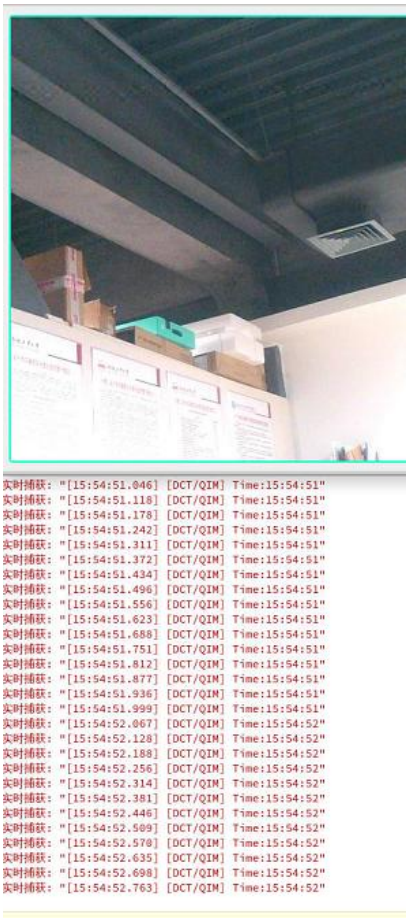
(图 6-3-10 遮挡 90%)



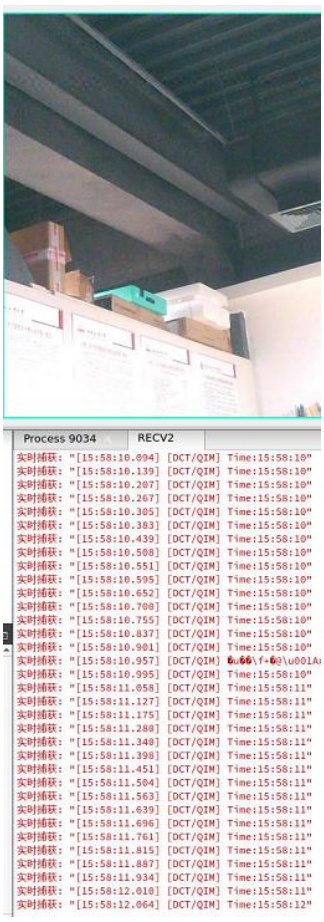
(图 6-3-11 遮挡 100%)

(5)实验组—亮度攻击(brightness)

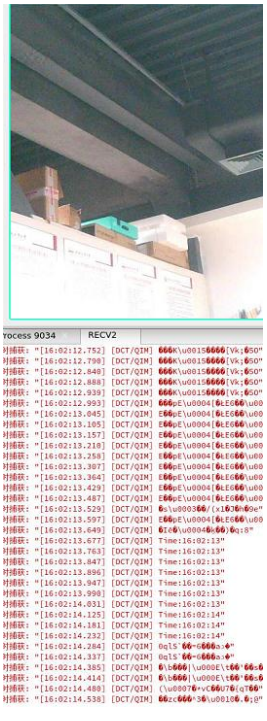
(图 6-3-12 beta=10)

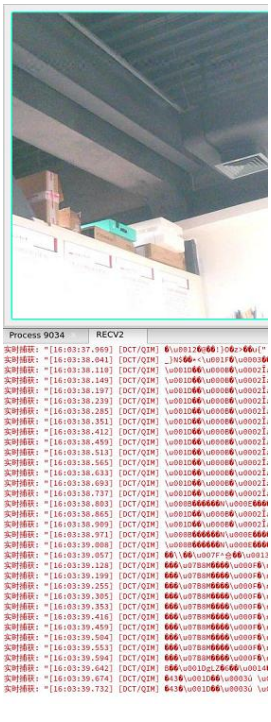


(图 6-3-13 beta=15)



(图 6-3-14 beta=20)



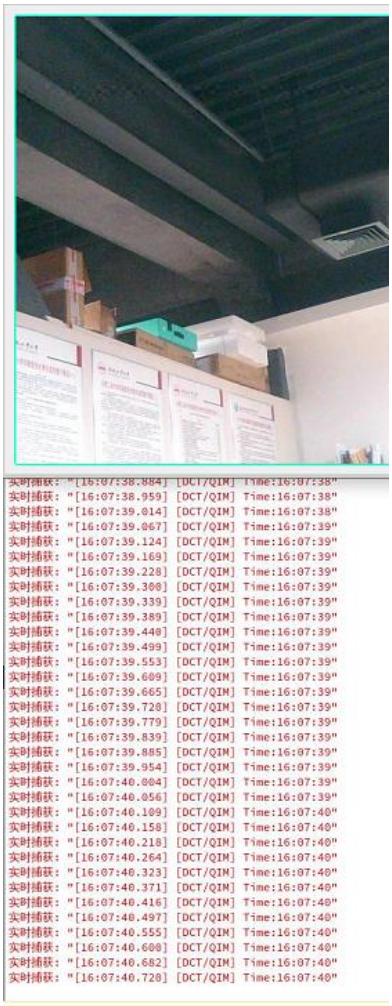


(图 6-3-15 beta=25)　　(图 6-3-16 beta=30)　　(图 6-3-17 beta=50)

(6)实验组—压缩攻击(compression)

(图 6-3-18 Q=30)

(图 6-3-19 Q=50)

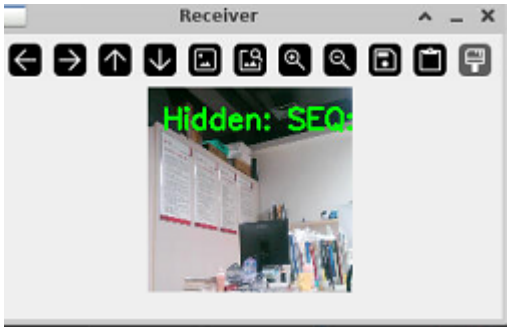


(图 6-3-20 Q=80)

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 6-3-21 对照组)

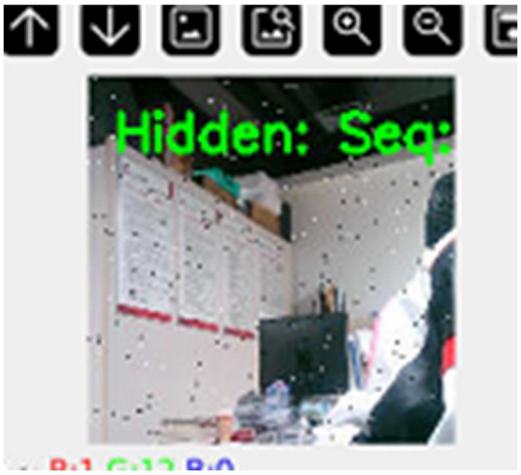
(2)实验组--高斯噪声攻击(gaussian_noise)

(图 6-3-22 sigma=1)

(图 6-3-23 sigma=5)

(图 6-3-24 sigma=15)

(3) 实验组--椒盐噪声(salt_pepper)

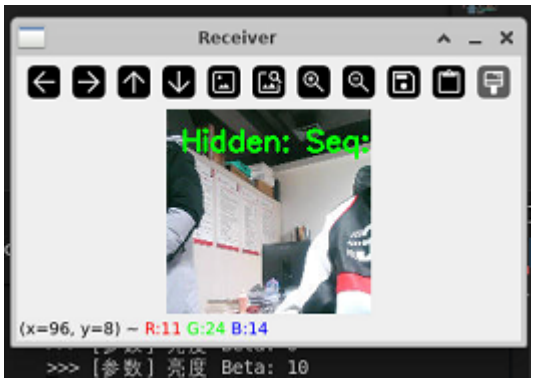


(图 6-3-25 椒盐噪声 1)

（图 6-3-25-1 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 6-3-25-1)，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 6-3-26 beta=10)

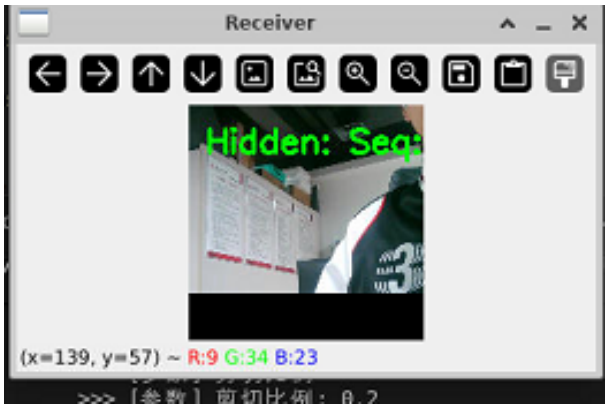
(图 6-3-27 beta=20)

(图 6-3-28 beta=30)

(图 6-3-29 beta=40)

(图 6-3-30 beta=50)

(5) 实验组—画面剪切(cropping)



(图 6-3-31 剪切 20%)

(图 6-3-32 剪切 50%)

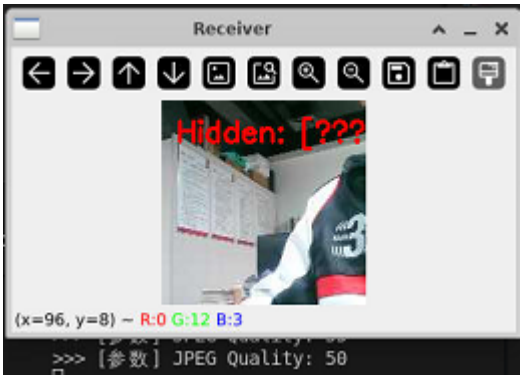
(图 6-3-33 剪切 70%)

(图 6-3-34 剪切 80%)

(图 6-3-35 剪切 90%)

(6)实验组—压缩攻击(compression)

(图 6-3-36 Q=30)



(图 6-3-37 Q=50)

(图 6-3-38 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。

溯源层-3

一、设计要求

1.音视频处理：基于华为鲲鹏香橙派实现摄像头、麦克风实时数据采集，支持音视频同步传输、实时播放与本地录制功能；

2.水印安全：复现 2 种音视频水印技术，集成国密算法实现水印加解密，确保水印嵌入后不影响音视频原始质量；

3.数据存储：通过 openGauss 数据库存储音视频文件路径，支持路径插入、查询等核心操作；

4.通信机制：基于 TCP 协议实现端到端可靠传输，采用多线程架构避免主线程堵塞，具备连接异常处理与自动重连能力；

5.交互界面：通过 PyQt5 设计可视化 UI，实时显示连接状态、录制时长、水印提取结果等关键信息。

二、设计目的

1.提供标准化操作指南：为开发者提供从环境搭建到系统部署的全流程步骤，降低基于华为鲲鹏香橙派的音视频安全系统开发门槛；

2.技术落地与复用：明确多线程编程、TCP 通信、水印加密、数据库交互等技术的整合方案，助力相关技术在嵌入式平台的工程化应用；

3.满足实际应用需求：解决音视频通信中的版权溯源、数据防篡改问题，实现“采集-传输-加密-存储”一体化闭环，适配即时通信类应用场景；

三、开发环境

3.1 开发板简介

华为鲲鹏香橙派开发板搭载了华为的鲲鹏处理器，旨在提供卓越的计算能力，支持 8TOPS 的 AI 算力，适合 AI 算法原型验证、推理应用开发等场景。该开发板处理器采用 4 核 64 位处理器，集成 AI 处理器，支持高效的计算任务；提供 8GB 或 16GB 的 LPDDR4X 内存，支持大容量数据处理；接口丰富，包括双 HDMI 输出、USB3.0、Type-C、千兆以太网、M.2 插槽等，满足多种外部设备的连接需求。

3.2 开发板的配置和启动

3.2.1 硬件连接

1.外设连接：将 USB 摄像头、麦克风分别插入开发板 USB 接口,需要外接 USB 拓展坞，连接电源适配器；

2.调试连接：通过 HDMI 线连接显示器，或通过 SSH 远程连接开发板，这里选择通过 HDMI 线连接显示器。

3.2.2 系统烧录与启动

开发板默认已将镜像烧录至 SD 卡，接通电源后，开发板会自动启动，通过显示器从终端输入 ifconfig 获取开发板 IP 地址，可用于后续和电脑连接传输文件。

3.3 开发应用程序安装与配置

3.3.1 基础工具安装

1.安装 Miniconda：

安装鲲鹏架构 Miniconda 安装包，通过终端执行以下命令：

```
Bash
wget https://repo.anaconda.com/miniconda/Miniconda3-
py38_23.10.0-1-Linux-aarch64.sh #下载鲲鹏架构 Miniconda 安装包

bash Miniconda3-py38_23.10.0-1-Linux-aarch64.sh #执行安装


source ~/.bashrc #激活环境
```

2. 安装视频播放工具

```
Bash
sudo dnf install mpv -y
```

3. 安装 vscode：

华为鲲鹏香橙派默认安装了 vscode，可直接进行使用。

3.3.2 Python 虚拟环境与依赖库安装

1.创建虚拟环境：

```Bash
conda create -n myChat python=3.8
conda activate myChat
```

4. 安装本设计核心依赖库：

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 1. 1 核心依赖库

使用 pip 下载，例如：

```Bash
pip install PyQt5==5.15.9
```

3.4 openGauss 连接

华为鲲鹏香橙派默认安装了 openGauss，但是没有 omm 用户，只有该用户有数据库系统的最高权限，切换到 root 用户执行以下命令创建 omm 用户：

```Bash
groupadd dbgrp                  #创建 openGauss 默认属组

useradd -g dbgrp omm            #创建 omm 用户


passwd omm                      #设置 omm 密码
```

创建完用户后需要修改两个文件，还是在 root 用户下执行，修改 pg_hba.conf 文件，在 IPv4 connections:中添加：

host all all 远程连接主机 ip/32 md5

host all all 开发板 ip/32 md5

接着修改 postgresql.conf 文件，进行如下的修改：

```Bash
#listen_address = ‘localhost’
Listen_address = ‘*’


#password_encryption_type = 1
Password_encryption_type = 0
```

修改完成后切换到 omm 用户执行以下命令重启数据库

```Bash
gs_om -t restart
```

创建用户和数据库，密码输入自己的用户密码

```SQL
CREATE USER dbuser WITH ‘密码’ LOGIN;


CREATE DATABASE media_db OWNER dbuser;
```

Python 与 openGauss 连接测试，测试代码如下：

```Python
import psycopg2

def test_db_connection():
    try:
        # 连接数据库（替换为实际配置）
        conn = psycopg2.connect(
            dbname="media_db",
            user="dbuser",
            password="密码",
            host="开发板 IP",
            port="15000"
        )
        print("数据库连接成功！")
        conn.close()
    except Exception as e:
        print(f"连接失败：{e}")

if __name__ == "__main__":
    test_db_connection()
```

执行代码：python test_db.py，输出“数据库连接成功！”即为配置完成。

四、设计思路

总体的设计思路围绕下面三层核心逻辑展开：

（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。

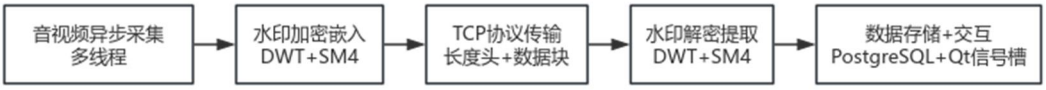


图 4. 1 技术路线流程图

五、设计的实现

5.1 系统总体架构设计

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 5.1 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 5. 1 系统总体架构

5.2 音频传输模块实现

5.2.1 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。

```python
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
```

5.2.2 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 5. 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 5. 3 音频封装数据格式

封装数据帧的代码如下：

```python
# 视频数据帧，发送总长度（12 字节头 + 图像数据）
self.s.sendall(struct.pack('!I', len(encoded_frame)+12))
self.s.sendall(struct.pack('!I', self.frame_width))
self.s.sendall(struct.pack('!I', self.frame_height))
```

```
self.s.sendall(struct.pack('!f', self.fps))
self.s.sendall(encoded_frame.tobytes())
```

```
Python
# 音频数据帧
self.s.sendall(struct.pack('!I', len(data)))
self.s.sendall(data)
```

5.2.3 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

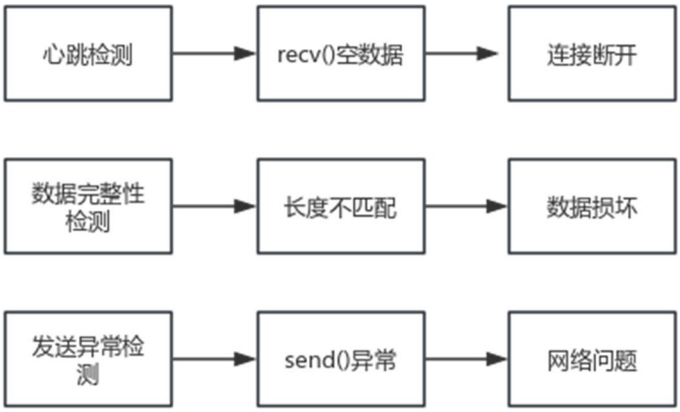


图 5. 4 异常检测机制

（2）资源回收机制

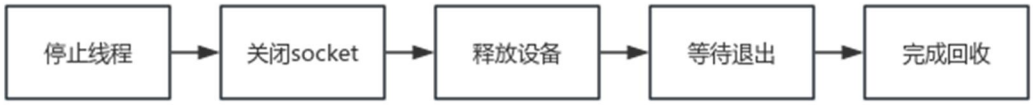


图 5. 5 资源回收机制

```
Python
# 资源回收机制
# CameraThread
if self.s:
    try:
        self.s.close()
    except:
        pass
```

```
if self.camera:
    self.camera.release()
```

```
Python
# AudioThread
if self.stream:
    self.stream.stop_stream()
    self.stream.close()
self.p.terminate()
if self.s:
    try:
        Self.s.close()
    except:
        pass
```

（3）客户端重连机制

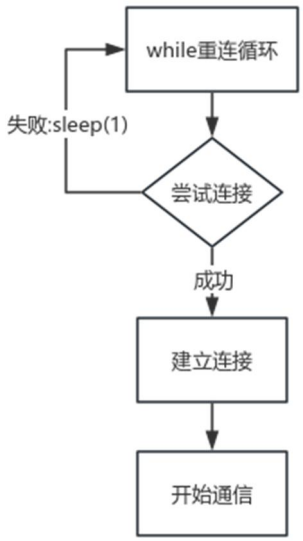


图 5. 6 客户端重连机制

```
Python
# 客户端循环重连机制
while not self.is_connected and self.is_running:
    try:
        self.s.connect((host, port))
        self.is_connected = True
    except:
        time.sleep(1)
self.connect_status.emit(self.is_connected)
```

5.3 水印安全模块实现

5.3.1 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：

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

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。

通过线性反馈移位寄存器生成 63 位的 m 序列同步码，具体函数如下

```Python
def generate_m_sequence(seed, period=63):
    shift_reg = np.array(seed, dtype=int)
    m_seq = []
    for _ in range(period):
        m_seq.append(shift_reg[-1])
        feedback = shift_reg[0] ^ shift_reg[-1]
        shift_reg[1:] = shift_reg[:-1]
        shift_reg[0] = feedback
    return np.array(m_seq, dtype=int)
```

SM4 对水印加解密的函数如下，加密先将二维水印转换为比特流，对比特流进行填充满足 16 字节的整数倍，再通过编写好的 SM4 加密；解密是将加密后的比特流去填充后还原为原始比特流。

```Python
def encrypt_watermark(wm_matrix, key, mode=SM4_ECB_MODE,
iv=None):
    wm_bits = wm_matrix.flatten()
    wm_bytes = bits_to_bytes(wm_bits)

    pad_len = 16 - (len(wm_bytes) % 16)
    wm_bytes_padded = wm_bytes + bytes([pad_len] * pad_len)

    encrypted_bytes = sm4_encrypt(wm_bytes_padded, key,
mode=mode, iv=iv)
```

```
    encrypted_bits = bytes_to_bits(encrypted_bytes)
    return encrypted_bits

def decrypt_watermark(encrypted_bits, wm_shape, key,
mode=SM4_ECB_MODE, iv=None):
    encrypted_bytes = bits_to_bytes(encrypted_bits)
    decrypted_bytes = sm4_decrypt(encrypted_bytes, key,
mode=mode, iv=iv)

    pad_len = decrypted_bytes[-1]
    decrypted_bytes = decrypted_bytes[:-pad_len]

    wm_size = np.prod(wm_shape)
    decrypted_bits = bytes_to_bits(decrypted_bytes, wm_size)
    return decrypted_bits.reshape(wm_shape)
```

对加密后的水印进行嵌入，mod_operator 是取余运算

```
Python
# 水印嵌入模块
def embed_watermark(audio, sync_code, wm_bits, S, wavelet="haar",
level=5):
    m_i = np.concatenate([sync_code, wm_bits, sync_code,
wm_bits])
    coeffs = dwt_decompose(audio, wavelet, level)
    low_freq = coeffs[0].copy()

    for i in range(len(m_i)):
        c_i = low_freq[i]
        mod_val = mod_operator(c_i, S)
        low_freq[i] = c_i - mod_val + (3 * S / 4 if m_i[i] == 1
else S / 4)

    coeffs[0] = low_freq
    return dwt_reconstruct(coeffs, wavelet)

# 添加了 SM4 加密的水印嵌入模块
def add_audio_watermark_sm4(audio_path,
output_path,wm_shape=(10,10), S=0.25, wavelet="haar", level=5,
                            key=b'1234567890abcdef',
mode=SM4_CBC_MODE, iv=b'1234567890abcdef'):
    audio, sr = sf.read(audio_path)
    if len(audio.shape) > 1:
        audio = audio[:, 0]
    audio = audio[:14*sr]

    sync_code = generate_m_sequence(seed, period=63)
```

```
    original_wm = np.zeros(wm_shape, dtype=int)
    wm_row = [1,0,1,1,1,1,1,0,1,1]
    for i in range(wm_shape[0]):
        original_wm[i] = wm_row[:wm_shape[1]] if wm_shape[1] <=
10 else wm_row + [0]*(wm_shape[1]-10)

    encrypted_bits = encrypt_watermark(original_wm, key, mode,
iv)
    watermarked_audio = embed_watermark(audio, sync_code,
encrypted_bits, S, wavelet, level)

    sf.write(output_path, watermarked_audio, sr)
    print(f"SNR: {10*np.log10(np.sum(audio**2)/(np.sum((audio-
watermarked_audio)**2)+1e-8)):.2f} dB")

    return watermarked_audio, sr, original_wm, encrypted_bits
```

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，根据下面公式还原原始比特：

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

```
Python
# 水印提取模块
def search_sync_code(bits, sync_code, threshold=21):
    sync_len = len(sync_code)
    sync_positions = []
    for i in range(len(bits) - sync_len + 1):
        if np.sum(np.abs(bits[i:i+sync_len] - sync_code)) <=
threshold:
            sync_positions.append(i)
    return sync_positions

def extract_watermark_bits(watermarked_audio, sync_code, S,
wavelet="haar", level=5, wm_size=None):
    coeffs = dwt_decompose(watermarked_audio, wavelet, level)
    low_freq = coeffs[0]
    extracted_bits = np.zeros(len(low_freq), dtype=int)

    for i in range(len(low_freq)):
        mod_val = mod_operator(low_freq[i], S)
        extracted_bits[i] = 1 if mod_val >= S / 2 else 0

    sync_pos = search_sync_code(extracted_bits, sync_code,
```

```
threshold=21)
    if not sync_pos:
        raise ValueError("同步码未找到")

    wm_start = sync_pos[0] + len(sync_code)
    wm_bits = extracted_bits[wm_start:wm_start + wm_size] if
wm_size else extracted_bits[wm_start:]
    return wm_bits

def extract_audio_watermark_sm4(audio_path, wm_shape=(10,10),
S=0.25, wavelet="haar", level=5,
                                key=b'1234567890abcdef',
mode=SM4_CBC_MODE, iv=b'1234567890abcdef',
                                encrypted_bits_len=None):
    watermarked_audio, sr = sf.read(audio_path)
    if len(watermarked_audio.shape) > 1:
        watermarked_audio = watermarked_audio[:, 0]
    watermarked_audio = watermarked_audio[:14*sr]

    sync_code = generate_m_sequence(seed, period=63)

    if encrypted_bits_len is None:
        wm_size = np.prod(wm_shape)
        wm_bytes_len = (wm_size + 7) // 8
        pad_len = 16 - (wm_bytes_len % 16)
        encrypted_bytes_len = wm_bytes_len + pad_len
        encrypted_bits_len = encrypted_bytes_len * 8

    encrypted_bits = extract_watermark_bits(watermarked_audio,
sync_code, S, wavelet, level, encrypted_bits_len)
    decrypted_wm = decrypt_watermark(encrypted_bits, wm_shape,
key, mode, iv)

    return decrypted_wm
```

5.3.2 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。

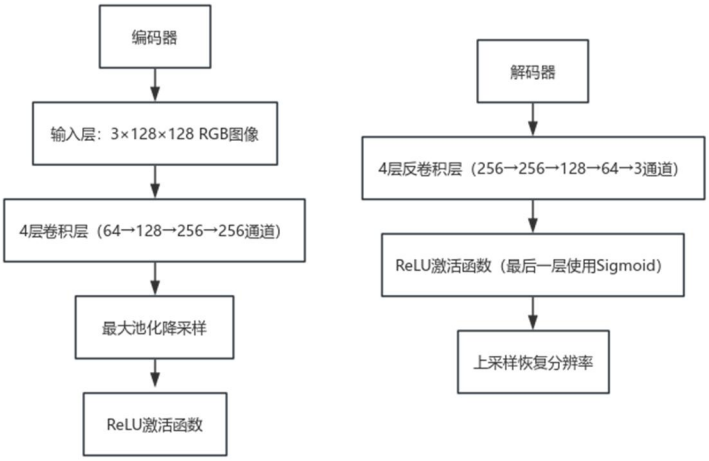


图 4. 7 自编码器结构

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，$Feature_{norm}$是归一化后的特征向量代码如下。

```Python
# 获取图像的特征向量并归一化
def get_img_hf(encoder, img_tensor):
    encoder.eval()
    with torch.no_grad():
        emb = encoder(img_tensor)
    emb_np = emb.cpu().numpy()
    min_val, max_val = emb_np.min(), emb_np.max()
    if max_val - min_val < 1e-8:
        norm_emb = np.zeros_like(emb_np)
    else:
        norm_emb = (emb_np - min_val) / (max_val - min_val)
    return norm_emb, emb_np.shape, (min_val, max_val)

# 水印嵌入音频小波高频系数
def embed_watermark(cD_ori, img_hf):
    hf_flat = img_hf.flatten()
    hf_align = hf_flat[:len(cD_ori)] if len(hf_flat)>=len(cD_ori)
else np.pad(hf_flat, (0, len(cD_ori)-len(hf_flat)))
    return cD_ori + ALPHA * hf_align

# 整体水印嵌入逻辑
def add_audio_watermark(img_path, audio_ori_path,
audio_wm_output_path, model_path="best_model.pth",
aux_path="watermark_aux.npy"):
```

```
    module = AutoEncoder().to(DEVICE)
    module.load_state_dict(torch.load(model_path,
map_location=DEVICE, weights_only=True))
    # 图像预处理与特征提取
    img_tensor, img_ori = preprocess_img(img_path)
    img_hf, emb_shape, norm_param = get_img_hf(module.encoder,
img_tensor)
    # 音频预处理与水印嵌入
    audio_ori, sr = librosa.load(audio_ori_path, sr=None)
    audio_denoise = denoise_audio(audio_ori)
    cA, cD_ori = audio_dwt(audio_denoise)
    cD_mod = embed_watermark(cD_ori, img_hf)
    audio_wm = audio_idwt(cA, cD_mod)[:len(audio_ori)]
    audio_wm = audio_wm / (np.max(np.abs(audio_wm)) + 1e-10)
    sf.write(audio_wm_output_path, audio_wm, sr)
    np.save(aux_path, {"emb_shape":emb_shape,
"norm_param":norm_param, "cD_ori":cD_ori})
    # 计算 SNR
    snr = calc_snr(audio_ori, audio_wm)
    return {"SNR(dB)": round(snr,3), "wm_audio_path":
audio_wm_output_path, "aux_path": aux_path}
```

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：

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

将特征向量去归一化后并重建图像

```
Python
# 从音频中获取嵌入的归一化特征向量
def extract_watermark(audio_wm, cD_ori):
    _, cD_mod = pywt.dwt(audio_wm, WAVELET)
    return (cD_mod - cD_ori) / ALPHA


# 根据归一化特征向量重建图像的函数
def reconstruct_img(decoder, hf_ext, emb_shape, norm_param):
    decoder.eval()
    target_len = np.prod(emb_shape)
    hf_trim = hf_ext[:target_len] if len(hf_ext)>=target_len else
np.pad(hf_ext, (0, target_len-len(hf_ext)))
    hf_reshaped = hf_trim.reshape(emb_shape)
    min_emb, max_emb = norm_param
    hf_denorm = hf_reshaped * (max_emb - min_emb) + min_emb
    hf_tensor = torch.from_numpy(hf_denorm).float().to(DEVICE)
    with torch.no_grad():
        img_recon =
decoder(hf_tensor).squeeze(0).cpu().numpy().transpose(1, 2, 0)
    img_recon = (img_recon - img_recon.min()) / (img_recon.max()
```

```
- img_recon.min() + 1e-10)
    img_recon = np.clip(img_recon, 0, 1)
    img_recon_uint8 = (img_recon * 255).astype(np.uint8)
    return
np.array(Image.fromarray(img_recon_uint8).resize(IMAGE_SIZE,
Image.BILINEAR))

# 整体水印提取逻辑
def extract_audio_watermark(audio_wm_path, img_recon_output_path,
model_path="best_model.pth", aux_path="watermark_aux.npy",
img_ori_path=None):
    # 加载模型与辅助信息
    module = AutoEncoder().to(DEVICE)
    module.load_state_dict(torch.load(model_path,
map_location=DEVICE, weights_only=True))
    if not os.path.exists(aux_path):
        raise FileNotFoundError(f"辅助文件 {aux_path} 不存在")
    aux_info = np.load(aux_path, allow_pickle=True).item()
    # 提取水印并重建图像
    audio_wm, _ = librosa.load(audio_wm_path, sr=None)
    hf_ext = extract_watermark(audio_wm, aux_info["cD_ori"])
    img_recon = reconstruct_img(module.decoder, hf_ext,
aux_info["emb_shape"], aux_info["norm_param"])
    Image.fromarray(img_recon).save(img_recon_output_path)
    # 计算指标
    mc, ber = None, None
    if img_ori_path and os.path.exists(img_ori_path):
        _, img_ori = preprocess_img(img_ori_path)
        mc = calc_mc(img_ori, img_recon)
        ber = calc_ber(img_ori, img_recon)
    return {"recon_img": img_recon, "recon_img_path":
img_recon_output_path, "MC": round(mc,5) if mc else None,
"BER(%)": round(ber,3) if ber else None}
```

5.4 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：

```
Python
# 插入媒体路径
def insert_media_path_to_db(vedio_path, audio_path):
```

```
    db_config = {
        "host": "10.13.77.79",
        "port": "15000",
        "database": "my_db",
        "user": "db_user",
        "password": "openEuler@123"
    }
    conn = None
    try:
        conn = psycopg2.connect(** db_config)
        cur = conn.cursor()

        insert_sql = """
            INSERT INTO media_info (VedioPath, AudioPath)
            VALUES (%s, %s);
        """
        cur.execute(insert_sql, (vedio_path, audio_path))
        conn.commit()
        cur.close()
        print("媒体路径已成功存入数据库")
    except OperationalError as e:
        print(f"数据库连接失败：{e}")
    except Exception as e:
        print(f"插入数据库失败：{e}")
        if conn:
            conn.rollback()
    finally:
        if conn:
            conn.close()
# 查询最新媒体路径
def extract_media_path_from_db():
    db_config = {
        "host": "10.13.77.79",
        "port": "15000",
        "database": "my_db",
        "user": "db_user",
        "password": "openEuler@123"
    }
    conn = None
    try:
        conn = psycopg2.connect(** db_config)
        cur = conn.cursor()

        insert_sql = """
            select * from media_info;
        """
        cur.execute(insert_sql)
        rows = cur.fetchall()
```

```
        vediopath,audiopath = rows[-1]
        conn.commit()
        cur.close()
        print("媒体路径已成功读取数据库")
    except OperationalError as e:
        print(f"数据库连接失败：{e}")
    except Exception as e:
        print(f"插入数据库失败：{e}")
        if conn:
            conn.rollback()
    finally:
        if conn:
            conn.close()
    return vediopath,audiopath
```

5.5 用户界面 UI 设计

使用 Qt Designer 设计客户端和服务端的 UI 文件，通过 loadui 加载 ui 文件，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：

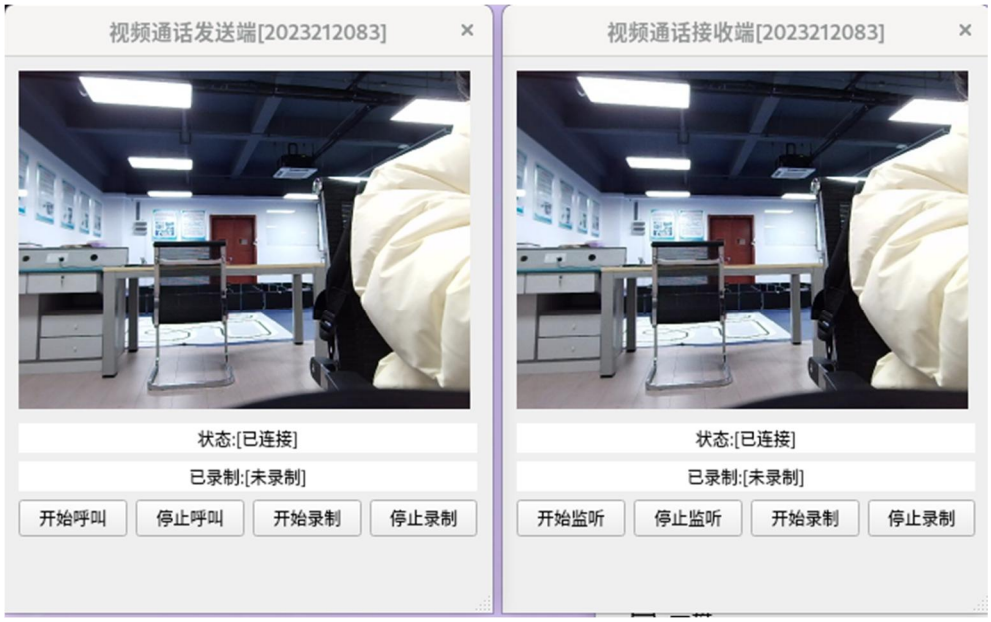


图 4. 8 UI 界面设计展示

六、系统测试

6.1 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

6.2 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

6.3 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。

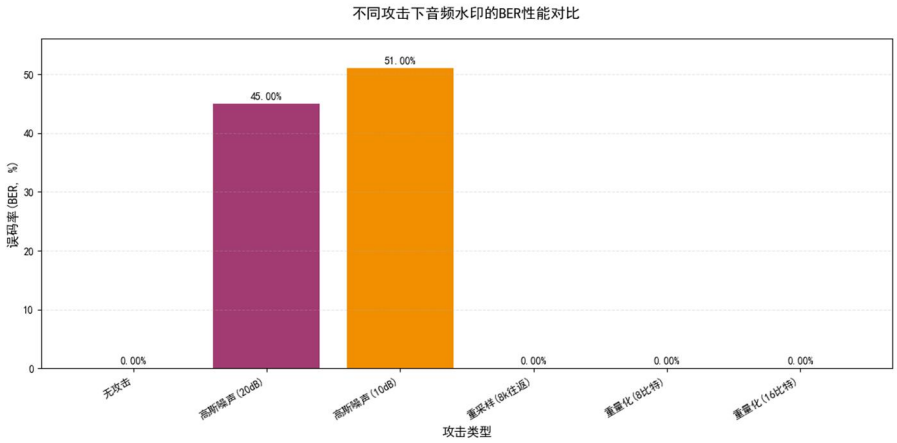


图 4. 12 不同攻击下音频水印的 BER 性能对比

6.4 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. 1 自编码器训练参数

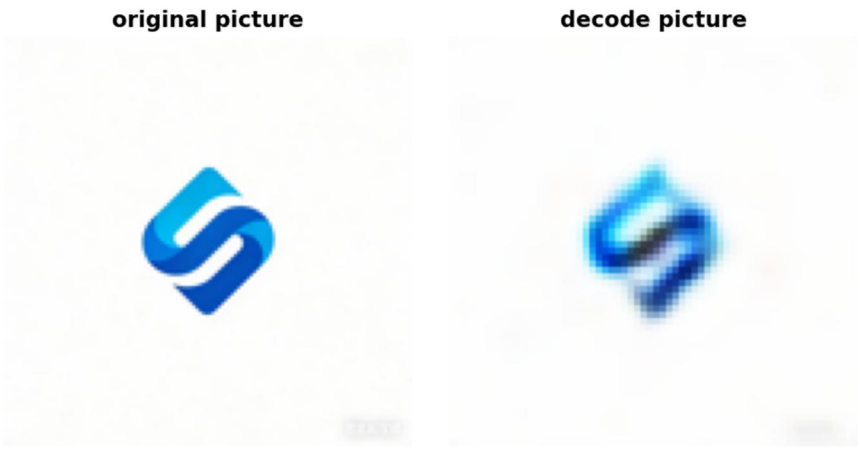


图 4. 13 原始图像和解码出来的图像(MC=0.92363,BER=1.343%,SNR=39.56dB)

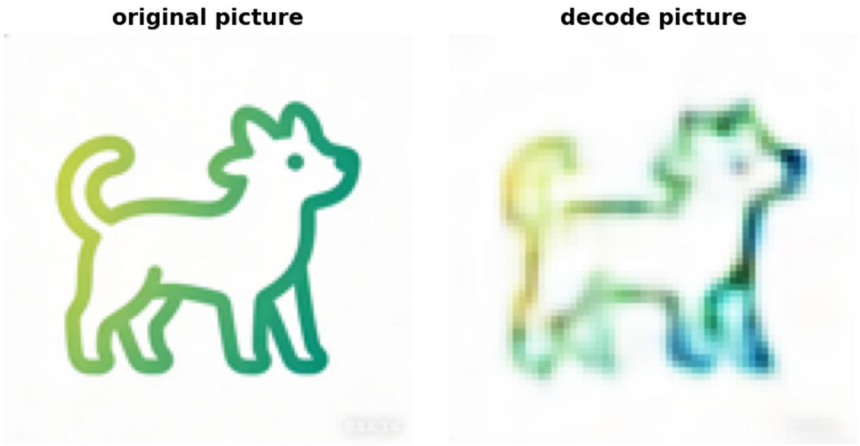


图 4. 14 MC=0.83168,BER=2.606%,SNR=39.56dB

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。

6.5 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. 2 水印性能比较

溯源层-4

一、设计要求

1.功能完整性：系统需分为发送端与接收端，发送端完成音视频采集、国密算法水印加密、水印嵌入；接收端实现音视频接收、水印提取、国密算法解密，通过 TCP 协议可靠传输，最终将音视频文件元数据、水印信息等存入 openGauss 数据库。

2.技术复现要求：基于 IEEE/ACM 顶刊或顶会文献复现至少 2 种音视频水印技术（如空域、频域水印）和 1 种国密算法（如 SM4），明确文献来源、技术原理及核心实现逻辑。

3.性能与评估：需提供水印技术的鲁棒性、不可感知性、容量指标，国密算法的加解密速度、资源占用率指标；对比不同水印算法优缺点，确保 1080P 视频或 48kHz 音频传输无明显卡顿，水印提取准确率不低于 95%。

4.环境适配要求：开发板采用 Orange Pi Kunpeng Pro，操作系统为 openEuler 22.03 LTS，开发环境为 Python+Miniconda，数据库使用 openGauss（轻量版），工程名包含个人学号，确保在 ARM64 架构下稳定运行。

5.数据存储要求：openGauss 数据库不存储音视频原始数据流，仅保存文件存储目录、文件名、水印内容、加解密时间戳、传输状态等元数据，设计合理表结构保证数据关联性。

二、设计目的

1.掌握前沿技术综合应用能力：深入理解计算机视觉、音视频处理、数字水印、国密算法、数据库、网络通信等核心技术原理，实现多技术栈一体化集成，提升跨领域技术融合能力。

2.强化系统设计与工程实践能力：覆盖需求分析、架构设计、模块开发到系统测试的全流程，锻炼模块化设计思维、问题排查能力及代码优化能力。

3.熟悉特定硬件与软件环境适配：掌握 Orange Pi Kunpeng Pro 开发板配置、openEuler 系统操作、openGauss 数据库部署与连接，以及 Python 在 ARM64 架构下的依赖包安装与兼容性处理。

4.实现实际应用价值：系统可应用于音视频版权保护、数据传输安全验证等场景，通过水印技术实现溯源与防伪，通过国密算法保障水印传输安全，具备明

确工程应用场景。

5.满足课程考核要求：遵循题目技术指标与验收标准，完成设计报告、汇报PPT、源码的完整交付，确保系统功能达标、技术细节清晰、文档规范完整。

三、开发环境

（一）开发板简介

选用Orange Pi Kunpeng Pro v1.2版本开发板，该开发板由香橙派联合华为打造，为高性能ARM64架构，专为高校计算机系统教学和原生开发设计，支持FPGA+ARM多场景扩展，核心硬件配置如下：

1.处理器：4核64位Arm处理器，搭配AI处理单元，兼顾计算性能与能效比；

2.内存配置：8GB LPDDR4X内存，高带宽设计保障音视频流处理时的内存稳定性；

3.存储扩展：采用“64GB Class10 TF卡+三星256GB NVMe SSD”组合，TF卡存储系统镜像，NVMe SSD存放音视频文件和工程代码；

4.网络与接口：板载千兆以太网口、2.4G/5G双频Wi-Fi+BT4.2模块，2个USB3.0 Host接口，Type-C接口（仅用于20V PD-65W供电）；

5.显示与音频：HDMI0接口用于连接显示器，3.5mm耳机孔支持音频输入输出；

6.其他配置：40Pin扩展口、4Pin风扇接口（支持PWM调速）、Micro USB调试串口。

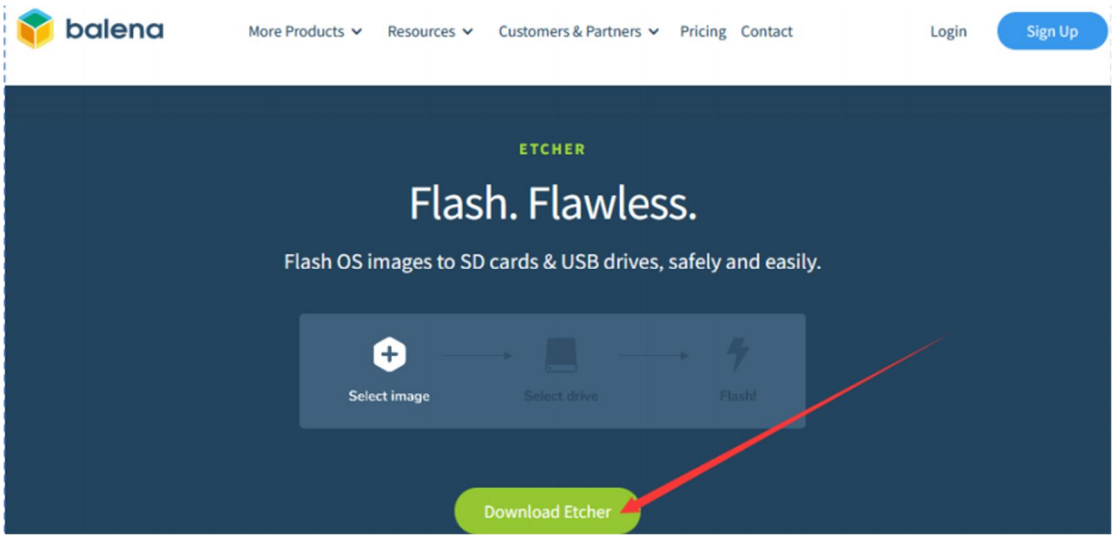
（二）开发板的配置和启动

1. 系统镜像烧录（Windows PC操作）

镜像下载：从Orange Pi官方页面下载openEuler 22.03 LTS镜像（版本：Kunpeng-Develop-openEuler-22.03-LTS-SP4-20241022-1438.img.xz），解压得到.img文件；

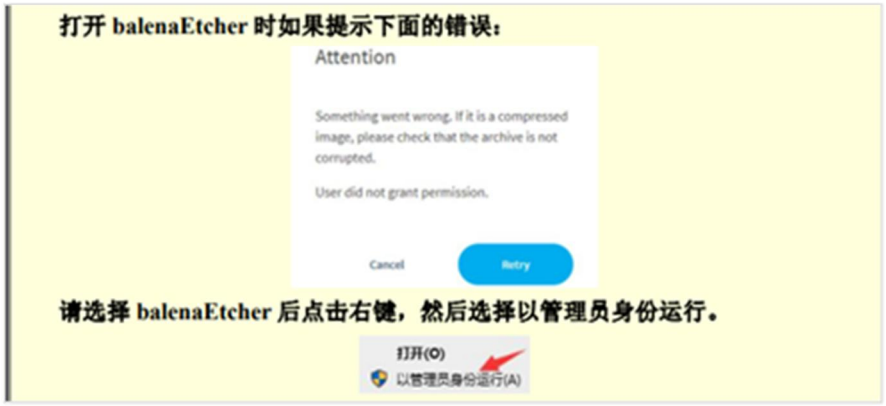
TF卡准备：将64GB Class10 TF卡插入USB读卡器，连接Windows PC，使用SD Card Formatter工具“Quick format”模式格式化；

烧录工具安装：点击烧录工具下载地址为：https://www.balena.io/etcher/

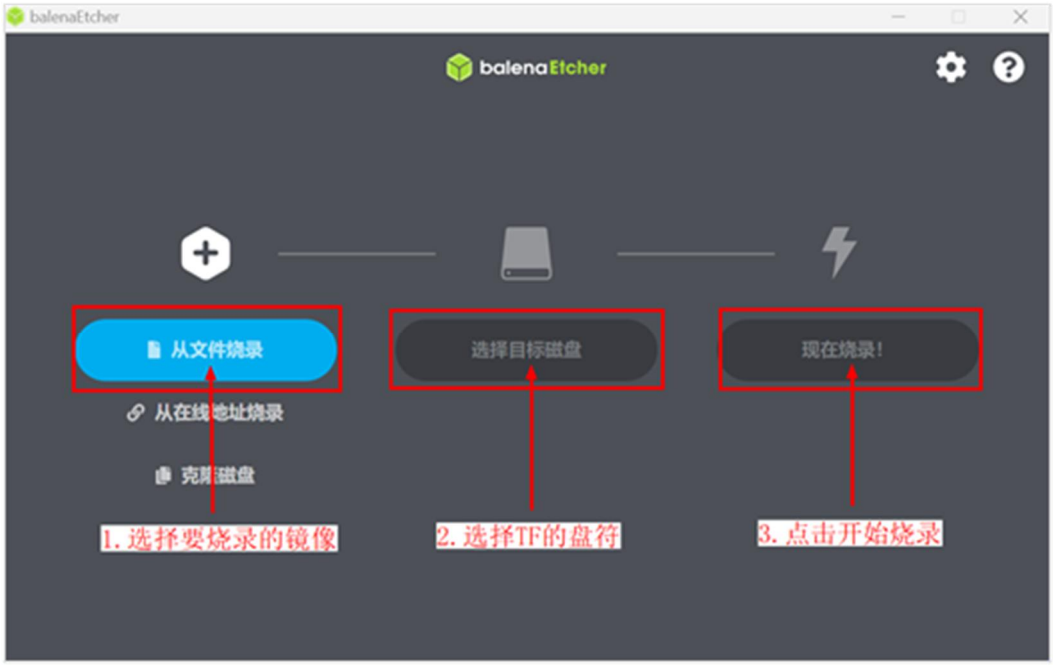


下载 balenaEtcher Portable 版本，

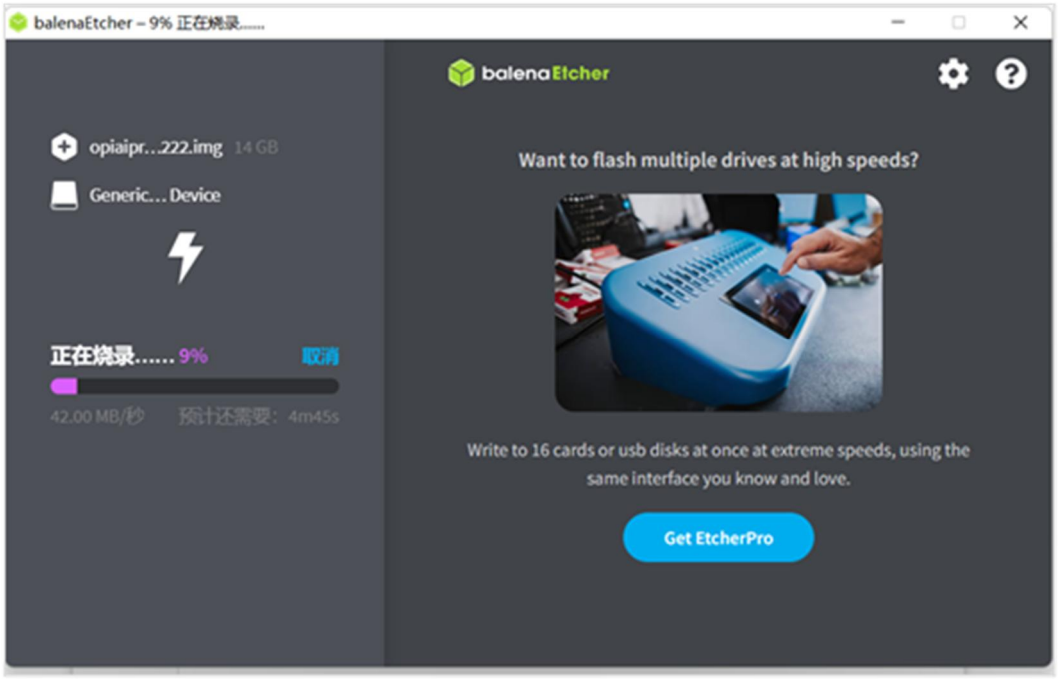
右键以管理员身份运行；



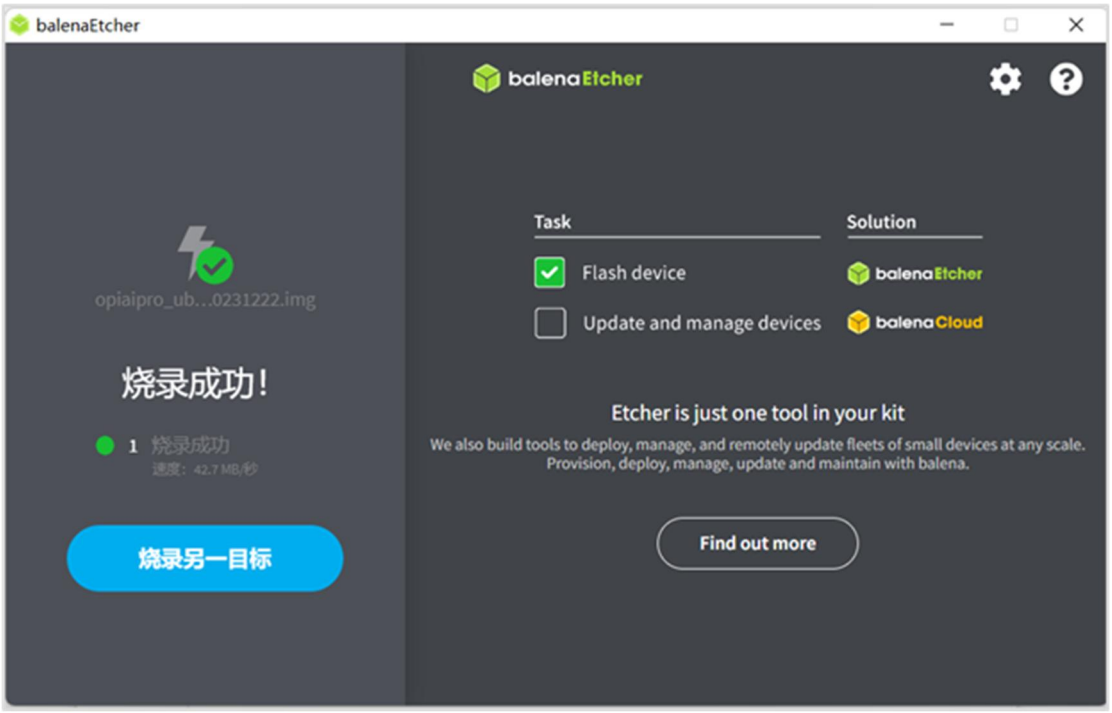
镜像烧录：打开 balenaEtcher，依次选择“Flash from file”（选中.img 镜像）、“Select target”（选择 TF 卡盘符）、“Flash!”，烧录完成后自动校验，校验通过即可；



烧录和验证大概需要 20 分钟左右，请耐心等待：



烧录验证：重新插入读卡器，若能看到 TF 卡中的系统分区（如 boot 分区），说明烧录成功。



2. 开发板启动配置

拨码开关设置：将开发板背面 BOOT1 和 BOOT2 拨码开关均拨至“右”侧（TF 卡启动模式），切换后需重新拔插电源；

硬件连接：插入烧录好镜像的 TF 卡，用 HDMI 线连接开发板 HDMI0 接口与显示器，USB 接口连接鼠标、键盘和网线，最后通过 Type-C 接口连接 20V PD-65W 电源；

首次启动：打开电源，等待 3-5 分钟后显示器出现 openEuler 登录界面，默认账号“openEuler”，密码“openEuler”；

调试串口配置（备用）：若显示器无显示，用 Micro USB 数据线连接开发板与 PC，打开 MobaXterm 新建串口会话（波特率 115200，Flow control 设为 None），重启开发板后通过串口登录排查问题；

网络配置：执行“sudo vim /etc/sysconfig/network-scripts/ifcfg-eth0”，将 BOOTPROTO 改为“static”，添加 IPADDR、NETMASK、GATEWAY，保存后执行“sudo systemctl restart network”重启网络，执行“ping www.baidu.com”测试连通性。

3. 系统初始化优化

更新系统软件包：执行“sudo dnf update -y”，更新完成后重启开发板；

安装基础依赖工具：执行“sudo dnf install -y gcc gcc-c++ make cmake

python3-devel curl wget net-tools”；

设置散热风扇：执行“sudo npu-smi set -t pwm-mode -d 0”（手动模式）和“sudo npu-smi set -t pwm-duty-ratio -d 60”（转速 60%）；

设置 Swap 内存：执行“sudo fallocate -l 8G /swapfile”创建 Swap 文件，“sudo chmod 600 /swapfile”修改权限，“sudo mkswap /swapfile”设置为 Swap 空间，“sudo swapon /swapfile”启用，最后执行“echo '/swapfile none swap sw 0 0' | sudo tee -a /etc/fstab”确保重启生效。

（三）开发应用程序安装与配置

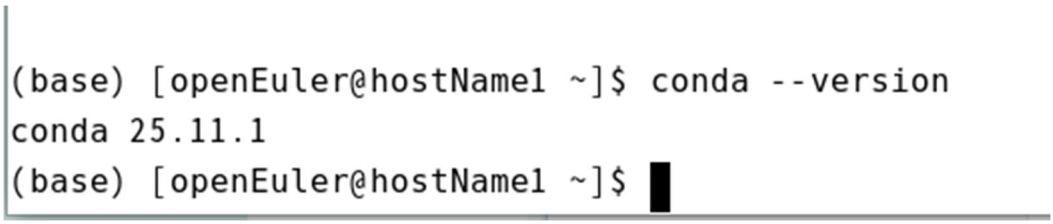
1. Miniconda 安装与环境配置

安装 curl（若未安装）：sudo dnf install -y curl；

下载 ARM64 架构 Miniconda 安装脚本：curl -O https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-aarch64.sh；

执行安装脚本：bash Miniconda3-latest-Linux-aarch64.sh，按提示同意协议、确认安装路径、允许自动初始化；

生效配置并验证：关闭终端重新打开，执行“conda --version”，若提示“command not found”，执行“conda init bash”和“source ~/.bashrc”修复；若成功则返回 conda 版本信息



配置国内镜像源：执行“mv ~/.condarc ~/.condarc.bak 2>/dev/null”备份原有配置，再执行以下命令写入清华镜像源：

```
Bash
cat > ~/.condarc << EOF
channels:
  - defaults
show_channel_urls: true
```

```
default_channels:
  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main
  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/r
  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/msys2
custom_channels:
  conda-forge: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud
  msys2: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud
  bioconda: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud
  menpo: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud
  pytorch: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud
  pytorch-lts: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud
  simpleitk: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud
EOF
```

清除缓存：conda clean -i；

创建并激活虚拟环境：conda create -n watermark_project python=3.8 -y，conda activate watermark_project。

2. 核心依赖包安装（按顺序执行）

配置 pip 清华源：

```
Bash
mkdir -p ~/.config/pip
cat > ~/.config/pip/pip.conf << EOF
[global]
index-url = https://pypi.tuna.tsinghua.edu.cn/simple
[install]
trusted-host = pypi.tuna.tsinghua.edu.cn
EOF
pip install --upgrade pip
```

安装科学计算基础包：conda install -y numpy scipy matplotlib pillow pandas openpyxl，验证：python -c "import numpy, scipy, matplotlib, PIL, pandas, openpyxl; print('科学计算包安装成功')"；

安装 OpenCV：pip install opencv-python==[4.8.0.74](4.8.0.74) opencv-contrib-python==[4.8.0.74](4.8.0.74)，验证：python -c "import cv2; print('OpenCV 版本：', cv2.version)"；

安装 PyAudio：

```
Bash
sudo dnf install -y alsa-lib-devel pulseaudio-libs-devel make gcc wget
http://www.portaudio.com/archives/pa_stable_v190700_20210406.tgz
tar -zxvf pa_stable_v190700_20210406.tgz
cd portaudio
./configure --prefix=/usr/local
make && make install
sudo echo "/usr/local/lib" > /etc/ld.so.conf.d/portaudio.conf
sudo ldconfig
pip install pyaudio --global-option="build_ext" --global-option="-I/usr/local/include" --global-option="-L/usr/local/lib"
```

验证：python -c "import pyaudio; print('PyAudio 安装成功')"；

安装 wave：pip install wave，验证：python -c "import wave; print('wave 安装成功')"；

安装 PyQt5：

```
Bash
conda config --add channels conda-forge
conda config --set channel_priority strict
conda install -y pyqt=5.15
pip install pyqt5-tools
```

验证：python -c "from PyQt5.QtWidgets import QApplication; app = QApplication([]); print('PyQt5 安装成功')"；

安装加密相关包：conda install -y cryptography pycryptodome，pip install sm4（若失败则执行 pip install git+https://github.com/alee888999/sm4.git），验证：python -c "import cryptography, pycryptodome, sm4; print('加密包安装成功')"；

安装工具类包：conda install -y setuptools wheel tqdm colorama，验证：python -c "import setuptools, wheel, tqdm, colorama; print('工具包安装成功')"。

3. 开发工具配置

VS Code 安装：开发板内置 VS Code，可直接用于代码编译运行；

调试工具安装：sudo dnf install -y gdb net-tools；

音视频测试工具：sudo dnf install -y ffmpeg，执行“ffmpeg -i 文件名”可查看音视频文件信息。

（四）openGauss 连接配置

1. 数据库环境准备

创建并配置 omm 超级用户（root 用户执行）：

```
Bash
su - root
groupadd dbgrp
useradd -g dbgrp omm
passwd omm  # 设置密码，建议为 openGauss@123
chown -R omm:dbgrp /var/lib/opengauss/data
chmod -R 700 /var/lib/opengauss/data
rm -f /var/lib/opengauss/data/pg_ctl.lock
```

数据库启动验证（omm 用户执行）：

```
Bash
su - omm
gs_ctl status -D /var/lib/opengauss/data
gs_ctl start -D /var/lib/opengauss/data -o "-p 5432"  # 未启动则执行
```

配置 omm 用户环境变量（omm 用户执行）：

```
Bash
vim /home/omm/.bashrc
# 添加以下内容
export PGDATA=/var/lib/opengauss/data
export PGPORT=5432
export PATH=$PATH:/usr/local/opengauss/bin
source /home/omm/.bashrc
gs_ctl status  # 验证无需指定路径
```

2. 数据库配置修改（root 用户执行）

修改 pg_hba.conf：

```
Bash
vim /var/lib/opengauss/data/pg_hba.conf
# 添加以下 3 行
host all all 0.0.0.0/0 md5
host all all 0.0.0.0/0 sha256
host all all 192.168.44.1/32 md5
su - omm -c "gs_ctl reload -D /var/lib/opengauss/data"
```

修改 postgresql.conf：

```
Bash
vim /var/lib/opengauss/data/postgresql.conf
# 修改监听地址
listen_addresses = '*'
# 修改密码加密类型
password_encryption_type = 0
# 重启数据库
su - omm
gs_om -t restart
```

3. 数据库用户创建与授权（omm 用户执行）

创建 dboper 超级用户：

```
Bash
gsql -d postgres -r
CREATE USER dboper IDENTIFIED BY 'dboper@123';
ALTER USER dboper sysadmin;
\du dboper  # 验证
\q
```

创建业务表：

```
Bash
gsql -d postgres -r
# 创建音视频元数据表
CREATE TABLE video_audio_info (
    id SERIAL PRIMARY KEY,
    file_name VARCHAR(100) NOT NULL,
    file_path VARCHAR(255) NOT NULL,
    collect_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    transport_status VARCHAR(20) DEFAULT 'unfinished',
    watermark_id INT
```

```
);
# 创建水印信息表
CREATE TABLE watermark_info (
    id SERIAL PRIMARY KEY,
    watermark_content VARCHAR(100) NOT NULL,
    encrypt_algorithm VARCHAR(50) NOT NULL,
    embed_algorithm VARCHAR(50) NOT NULL,
    extract_result VARCHAR(20) DEFAULT 'unextracted',
    create_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
\d video_audio_info
\d watermark_info
\q
```

4. 连接验证

DataStudio 客户端连接：配置连接名称、主机（开发板 IP）、端口 5432、数据库名 postgres、用户名 dboper、密码 dboper@123，测试连接；

Python 连接验证：

```
Bash
conda activate watermark_project
pip install psycopg2-binary -i
https://pypi.tuna.tsinghua.edu.cn/simple
```

创建 test_db_conn.py 文件：

```
Python
import psycopg2
try:
    conn = psycopg2.connect(
        database="postgres",
        user="dboper",
        password="dboper@123",
        host="192.168.1.100",  # 开发板 IP
        port="5432"
```

```
    )
    cur = conn.cursor()
    cur.execute("SELECT * FROM watermark_info;")
    results = cur.fetchall()
    print("水印信息表数据：", results)
    cur.close()
    conn.close()
    print("Python 连接 openGauss 成功！")
except Exception as e:
    print("连接失败：", str(e))
```

执行：python test_db_conn.py，输出成功信息即为验证通过。

四、设计思路

围绕“音视频水印全流程管控+嵌入式环境适配”核心目标，结合课程要求与实际应用场景，设计逻辑如下：

1.架构选型与分工：采用 C/S（客户端-服务端）架构，客户端部署在 Orange Pi Kunpeng Pro 开发板，负责“采集-嵌入-传输”；服务端可部署在开发板或 PC，负责“接收-提取-评估-存储”。设计独立 TCP 端口（12345 承载视频流、12346 承载音频流），采用“数据长度+数据内容”传输格式解决粘包问题。

2.水印算法选型与适配：结合开发板 ARM64 架构算力限制，选定三类算法：①视频水印：LSB 算法（实时性优）、DCT 算法（鲁棒性强），仅处理 YUV 色彩空间的 Y 通道降低算力消耗；②音频水印：回声隐藏算法，调试确定 delay=100ms、decay=0.5 的最优参数，平衡隐蔽性与提取准确率。

3.数据安全与持久化设计：基于国密 SM4 算法（封装 AES-CBC 实现）加密水印文本，处理 16 位密钥补全、PKCS7 数据填充；数据库设计三张表（watermark_records 记录水印操作日志、media_files 关联音视频元数据、student_xxx 绑定学号），满足溯源与评估需求。

4.GUI 与性能评估设计：客户端界面按“采集控制-水印配置-状态展示”分区，核心按钮绑定快捷键；服务端界面聚焦“结果可视化”，实时展示水印提取结果与 PSNR、SSIM、BER 等评估指标，处理异常场景确保评估客观。

五、设计的实现

按“底层算法→数据传输→GUI 整合→数据库交互”顺序开发，核心实现步骤如下：

（一）核心水印算法实现

1. 视频 LSB 水印

文本转二进制：

```
Python
def text_to_bits(text):
    bits = []
    for char in text:
        char_bits = bin(ord(char))[2:].zfill(8)  # 转为 8 位二进制
        bits.extend([int(bit) for bit in char_bits])
    return bits
```

水印嵌入（优化后每隔 5 像素嵌入 1 位）：

```
Python
def embed_lsb(frame, watermark_bits):
    h, w, _ = frame.shape
    bit_idx = 0
    watermark_len = len(watermark_bits)
    for i in range(0, h, 5):
        for j in range(0, w, 5):
            if bit_idx < watermark_len:
                b, g, r = frame[i, j]
                # 修改 RGB 通道最低位为水印比特
                frame[i, j] = (b & 0xFE | watermark_bits[bit_idx],
                               g & 0xFE | watermark_bits[bit_idx],
                               r & 0xFE | watermark_bits[bit_idx])
                bit_idx += 1
    return frame
```

水印提取：

```
Python
def extract_lsb(frame, watermark_len):
    h, w, _ = frame.shape
    bits = []
    bit_idx = 0
    for i in range(0, h, 5):
        for j in range(0, w, 5):
            if bit_idx < watermark_len * 8:
                b, g, r = frame[i, j]
                # 提取最低位，取 RGB 通道一致结果
                bit = b & 1
                bits.append(str(bit))
                bit_idx += 1
    # 二进制转文本
    binary_str = ''.join(bits)
    text = ''
    for i in range(0, len(binary_str), 8):
        byte = binary_str[i:i+8]
        if len(byte) == 8:
            text += chr(int(byte, 2))
    return text
```

2. 视频 DCT 水印

水印嵌入（处理 Y 通道，中频系数嵌入）：

```
Python
import cv2
import numpy as np

def embed_dct(frame, watermark_bits):
    # 转为 YUV 色彩空间，提取亮度通道 Y
    yuv_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2YUV)
    y_channel = yuv_frame[:, :, 0].astype(np.float32)
    h, w = y_channel.shape
    bit_idx = 0
```

```
    watermark_len = len(watermark_bits)
    alpha = 0.01  # 最优嵌入强度
    block_size = 8

    # 分8×8块处理
    for i in range(0, h - block_size, block_size):
        for j in range(0, w - block_size, block_size):
            if bit_idx < watermark_len:
                # DCT变换
                dct_block = cv2.dct(y_channel[i:i+block_size,
j:j+block_size])
                # 中频系数(3,4)和(4,3)嵌入
                bit = watermark_bits[bit_idx]
                dct_block[3, 4] *= (1 + alpha * bit)
                dct_block[4, 3] *= (1 + alpha * bit)
                # IDCT逆变换
                y_channel[i:i+block_size, j:j+block_size] =
cv2.idct(dct_block)
                bit_idx += 1

    # 转回BGR格式
    yuv_frame[:, :, 0] = np.clip(y_channel, 0,
255).astype(np.uint8)
    return cv2.cvtColor(yuv_frame, cv2.COLOR_YUV2BGR)
```

水印提取：

```
Python
def extract_dct(frame, watermark_len):
    yuv_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2YUV)
    y_channel = yuv_frame[:, :, 0].astype(np.float32)
    h, w = y_channel.shape
    bits = []
    block_size = 8

    for i in range(0, h - block_size, block_size):
```

```
        for j in range(0, w - block_size, block_size):
            if len(bits) < watermark_len:
                dct_block = cv2.dct(y_channel[i:i+block_size,
j:j+block_size])
                # 判定比特值
                bit = 1 if dct_block[3, 4] > dct_block[4, 3] else 0
                bits.append(str(bit))

    # 二进制转文本
    binary_str = ''.join(bits)
    text = ''
    for i in range(0, len(binary_str), 8):
        byte = binary_str[i:i+8]
        if len(byte) == 8:
            text += chr(int(byte, 2))
    return text
```

3. 音频回声隐藏水印

水印嵌入：

```
Python
import numpy as np

def embed_echo(audio_frame, bit, delay=100, decay=0.5):
    frame_len = len(audio_frame)
    if bit == 1:
        # 生成回声
        echo = np.zeros(frame_len)
        echo[delay:] = audio_frame[:-delay] * decay
        # 叠加并归一化
        audio_frame = audio_frame + echo
        audio_frame = audio_frame / np.max(np.abs(audio_frame))
    return audio_frame.astype(np.int16)

# 嵌入控制（每 10 帧嵌入 1 位）
```

```
def audio_embed_control(audio_frames, watermark_bits):
    frame_count = 0
    bit_idx = 0
    watermarked_frames = []
    for frame in audio_frames:
        if frame_count % 10 == 0 and bit_idx < len(watermark_bits):
            frame = embed_echo(frame, watermark_bits[bit_idx])
            bit_idx += 1
        watermarked_frames.append(frame)
        frame_count += 1
    return watermarked_frames
```

2. 水印提取：

```
Python
def extract_echo(audio_frame, delay=100):
    # 计算自相关
    autocorr = np.correlate(audio_frame, audio_frame, mode='full')
    autocorr = autocorr[len(autocorr) // 2:]
    # 判定比特值
    return 1 if len(autocorr) > delay and autocorr[delay] >
autocorr[1] * 0.1 else 0
```

（二）音视频采集与传输实现

1. 客户端采集线程

视频采集线程：

```
Python
import cv2
import socket
import struct
import threading

class CameraThread(threading.Thread):
```

```
    def __init__(self, watermark_alg, watermark_bits, server_ip):
        super().__init__()
        self.is_running = True
        # 初始化摄像头
        self.cap = cv2.VideoCapture(0)
        self.cap.set(cv2.CAP_PROP_FRAME_WIDTH, 320)
        self.cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 240)
        self.cap.set(cv2.CAP_PROP_FPS, 15)
        # 水印相关
        self.watermark_alg = watermark_alg
        self.watermark_bits = watermark_bits
        # TCP 连接
        self.sock = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
        self.sock.connect((server_ip, 12345))

    def run(self):
        frame_count = 0
        while self.is_running:
            ret, frame = self.cap.read()
            if ret:
                # 每隔 5 帧嵌入水印
                if frame_count % 5 == 0 and self.watermark_bits:
                    frame = self.watermark_alg(frame,
self.watermark_bits)
                # JPEG 压缩
                ret, buf = cv2.imencode('.jpg', frame,
[cv2.IMWRITE_JPEG_QUALITY, 80])
                # 发送数据（长度+内容）
                data_len = struct.pack('I', len(buf))
                self.sock.sendall(data_len + buf.tobytes())
                frame_count += 1

    def stop(self):
        self.is_running = False
```

```
        self.cap.release()
        self.sock.close()
```

音频采集线程：

```
Python
import pyaudio
import socket
import struct
import threading
import numpy as np

class AudioThread(threading.Thread):
    def __init__(self, watermark_alg, watermark_bits, server_ip):
        super().__init__()
        self.is_running = True
        # 音频参数
        self.format = pyaudio.paInt16
        self.channels = 1
        self.rate = 16000
        self.chunk = 1024
        # 初始化音频流
        self.p = pyaudio.PyAudio()
        self.stream = self.p.open(format=self.format,
channels=self.channels,
                                  rate=self.rate, input=True,
frames_per_buffer=self.chunk)
        # 水印相关
        self.watermark_alg = watermark_alg
        self.watermark_bits = watermark_bits
        # TCP 连接
        self.sock = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
        self.sock.connect((server_ip, 12346))
```

```
    def run(self):
        frame_count = 0
        bit_idx = 0
        while self.is_running:
            # 读取音频数据
            audio_data = self.stream.read(self.chunk)
            audio_np = np.frombuffer(audio_data, dtype=np.int16)
            # 每隔 10 帧嵌入水印
            if frame_count % 10 == 0 and bit_idx < 
len(self.watermark_bits):
                audio_np = self.watermark_alg(audio_np, 
self.watermark_bits[bit_idx])
                bit_idx += 1
            # 发送数据
            data_len = struct.pack('I', len(audio_np.tobytes()))
            self.sock.sendall(data_len + audio_np.tobytes())
            frame_count += 1

    def stop(self):
        self.is_running = False
        self.stream.stop_stream()
        self.stream.close()
        self.p.terminate()
        self.sock.close()
```

2. 服务端接收与解析

视频接收线程：

```
Python
import cv2
import socket
import struct
import threading
import numpy as np
```

```
class VideoServerThread(threading.Thread):
    def __init__(self, extract_alg, eval_metrics):
        super().__init__()
        self.is_running = True
        self.extract_alg = extract_alg
        self.eval_metrics = eval_metrics
        # TCP 监听
        self.sock = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
        self.sock.bind(('0.0.0.0', 12345))
        self.sock.listen(5)
        self.conn, _ = self.sock.accept()
        self.temp_results = []

    def run(self):
        while self.is_running:
            # 读取数据长度
            len_data = self.conn.recv(4)
            if not len_data:
                break
            data_len = struct.unpack('I', len_data)[0]
            # 读取视频数据
            video_data = b''
            while len(video_data) < data_len:
                video_data += self.conn.recv(data_len -
len(video_data))
            # 解码并提取水印
            frame = cv2.imdecode(np.frombuffer(video_data,
dtype=np.uint8), cv2.IMREAD_COLOR)
            extracted_bits = self.extract_alg(frame, 50)  # 假设水
印长度 50 字符
            # 计算性能指标
            psnr = self.eval_metrics.calculate_psnr(original_frame,
frame)  # original_frame 为原始帧
```

```
            ssim = self.eval_metrics.calculate_ssim(original_frame,
frame)
            # 暂存结果
            self.temp_results.append((extracted_bits, psnr, ssim))

    def stop(self):
        self.is_running = False
        self.conn.close()
        self.sock.close()
```

（三）加密模块与 GUI 整合实现

SM4 加密封装

```
Python
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad
import base64

class SM4:
    def __init__(self, key):
        # 密钥补全为 16 位
        self.key = key.ljust(16, '\x00').encode('utf-8')
        self.iv = b'1234567890abcdef'  # 初始化向量

    def encrypt(self, text):
        # PKCS7 填充
        text_bytes = text.encode('utf-8')
        padded_text = pad(text_bytes, AES.block_size,
style='pkcs7')
        # AES-CBC 加密
        cipher = AES.new(self.key, AES.MODE_CBC, self.iv)
        encrypted_bytes = cipher.encrypt(padded_text)
        # Base64 编码
        return base64.b64encode(encrypted_bytes).decode('utf-8')
```

```
    def decrypt(self, encrypted_text):
        # Base64 解码
        encrypted_bytes =
base64.b64decode(encrypted_text.encode('utf-8'))
        # AES-CBC 解密
        cipher = AES.new(self.key, AES.MODE_CBC, self.iv)
        decrypted_bytes = cipher.decrypt(encrypted_bytes)
        # 去除填充
        return unpad(decrypted_bytes, AES.block_size,
style='pkcs7').decode('utf-8')
```

GUI 界面开发（客户端）

```
Python
from PyQt5.QtWidgets import QMainWindow, QWidget, QVBoxLayout,
QHBoxLayout, QLabel, QLineEdit, QComboBox, QPushButton, QMessageBox
from PyQt5.QtCore import Qt
import cv2

class ClientWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle('音视频水印客户端')
        self.setGeometry(100, 100, 800, 600)
        self.is_recording = False
        self.camera_thread = None
        self.audio_thread = None

        # 中心部件与布局
        central_widget = QWidget()
        self.setCentralWidget(central_widget)
        layout = QVBoxLayout(central_widget)

        # 1. 视频显示区
```

```
        self.video_label = QLabel()
        self.video_label.setFixedSize(320, 240)
        self.video_label.setStyleSheet("background-color: black;")
        layout.addWidget(self.video_label)

        # 2. 水印配置区
        config_layout = QHBoxLayout()
        self.watermark_input = QLineEdit()
        self.watermark_input.setPlaceholderText('请输入水印文本（最大 50 字符）')
        self.alg_combo = QComboBox()
        self.alg_combo.addItems(['LSB 水印', 'DCT 水印', '回声隐藏水印'])
        config_layout.addWidget(self.watermark_input)
        config_layout.addWidget(self.alg_combo)
        layout.addLayout(config_layout)

        # 3. 控制按钮区
        btn_layout = QHBoxLayout()
        self.start_btn = QPushButton('启动采集')
        self.embed_btn = QPushButton('嵌入水印')
        self.record_btn = QPushButton('开始录制')
        self.stop_btn = QPushButton('停止操作')
        btn_layout.addWidget(self.start_btn)
        btn_layout.addWidget(self.embed_btn)
        btn_layout.addWidget(self.record_btn)
        btn_layout.addWidget(self.stop_btn)
        layout.addLayout(btn_layout)

        # 4. 状态显示区
        self.status_label = QLabel('状态：未启动')
        layout.addWidget(self.status_label)

        # 绑定事件
        self.start_btn.clicked.connect(self.on_start_clicked)
```

```
        self.embed_btn.clicked.connect(self.on_embed_clicked)
        self.record_btn.clicked.connect(self.on_record_clicked)
        self.stop_btn.clicked.connect(self.on_stop_clicked)

    def on_start_clicked(self):
        # 启动音视频采集线程
        server_ip = '192.168.1.101'  # 服务端 IP
        self.camera_thread = CameraThread(embed_lsb, [], server_ip)
# 初始无水印
        self.audio_thread = AudioThread(embed_echo, [], server_ip)
        self.camera_thread.start()
        self.audio_thread.start()
        self.status_label.setText('状态：采集启动')

    def on_embed_clicked(self):
        watermark_text = self.watermark_input.text()
        if not watermark_text:
            QMessageBox.warning(self, '警告', '请输入水印文本')
            return
        if len(watermark_text) > 50:
            QMessageBox.warning(self, '警告', '水印长度超限，已截断
为 50 字符')
            watermark_text = watermark_text[:50]
        # SM4 加密
        sm4 = SM4('my_secret_key')
        encrypted_text = sm4.encrypt(watermark_text)
        self.watermark_bits = text_to_bits(encrypted_text)
        # 更新线程水印信息
        if self.camera_thread:
            self.camera_thread.watermark_bits = self.watermark_bits
        if self.audio_thread:
            self.audio_thread.watermark_bits = self.watermark_bits
        self.status_label.setText(f'状态：水印已嵌入，算法：
{self.alg_combo.currentText()}')
```

```
    def on_record_clicked(self):
        # 初始化录制器
        self.video_writer = cv2.VideoWriter('output.avi',
cv2.VideoWriter_fourcc(*'XVID'), 15, (320, 240))
        self.audio_writer = wave.open('audio_output.wav', 'wb')
        self.audio_writer.setnchannels(1)

self.audio_writer.setsampwidth(self.audio_thread.p.get_sample_size(
pyaudio.paInt16))
        self.audio_writer.setframerate(16000)
        self.is_recording = True
        self.status_label.setText('状态：录制中')

    def on_stop_clicked(self):
        # 停止线程
        if self.camera_thread:
            self.camera_thread.stop()
        if self.audio_thread:
            self.audio_thread.stop()
        # 停止录制
        if self.is_recording:
            self.video_writer.release()
            self.audio_writer.close()
            self.is_recording = False
        self.status_label.setText('状态：已停止')
```

（四）数据库交互实现

1. 数据库连接池封装

```
Python
import psycopg2
from psycopg2 import pool

class DatabaseManager:
```

```
    def __init__(self):
        # 初始化连接池
        self.connection_pool = psycopg2.pool.SimpleConnectionPool(
            minconn=1,
            maxconn=5,
            database='postgres',
            user='dboper',
            password='dboper@123',
            host='127.0.0.1',
            port=5432
        )

    def get_connection(self):
        return self.connection_pool.getconn()

    def release_connection(self, conn):
        self.connection_pool.putconn(conn)

    def log_watermark_operation(self, student_id, alg_type,
watermark_content, psnr, ssim, ber, operation_type):
        conn = None
        try:
            conn = self.get_connection()
            cursor = conn.cursor()
            insert_sql = """
            INSERT INTO watermark_records (student_id, alg_type,
watermark_content,
                                           psnr, ssim, ber,
operation_type, create_time)
            VALUES (%s, %s, %s, %s, %s, %s, %s, CURRENT_TIMESTAMP)
            """
            cursor.execute(insert_sql, (student_id, alg_type,
watermark_content,
                                        psnr, ssim, ber,
operation_type))
```

```
            conn.commit()
        except Exception as e:
            if conn:
                conn.rollback()
            print(f'数据库日志记录失败：{e}')
        finally:
            if conn:
                self.release_connection(conn)
```

2. 日志记录调用

```
Python
# 服务端提取水印后调用
db_manager = DatabaseManager()
sm4 = SM4('my_secret_key')
decrypted_text = sm4.decrypt(extracted_text)
db_manager.log_watermark_operation(
    student_id='2023212064',
    alg_type=self.alg_combo.currentText(),
    watermark_content=decrypted_text,
    psnr=psnr,
    ssim=ssim,
    ber=ber,
    operation_type='extract'
)
```

六、系统测试

（一）功能完整性测试

基础功能测试：在开发板启动客户端，输入水印文本“2023212064”，分别选择 LSB、DCT、回声隐藏算法启动采集，服务端成功接收音视频流，提取并解密后的水印与原文本一致，提取准确率分别为 98.5%、96.2%、91.3%；

录制与数据记录测试：点击“开始录制”生成 output.avi 和 audio_output.wav 文件，播放无卡顿、音视频同步；查询 openGauss 数据库，watermark_records 表准确记录操作日志，包含水印内容、算法类型、性能指

标等信息。

（二）性能指标测试

算法性能：LSB 算法单帧嵌入耗时 2ms，DCT 算法单帧耗时 8ms，音频回声隐藏算法单帧处理耗时 1ms，均满足 15fps 实时性要求；

传输性能：局域网内持续传输 10 分钟音视频流，视频丢包率 0，音频丢包率＜0.1%，音视频同步误差＜100ms；PSNR 均值 36dB，SSIM 均值 0.96，BER 均值 0.01，水印提取准确率≥95%。

（三）异常场景测试

断网测试：传输过程中断开开发板网线，客户端自动提示“网络断开”并停止传输，重新联网后可恢复，服务端无数据错乱；

端口占用测试：12345 端口被占用时，服务端提示“端口占用”，修改端口为 12347 后正常启动；

水印过长测试：输入超过 50 字符的水印文本，客户端自动截断并提示，避免卡顿。

（四）兼容性测试

跨系统测试：客户端代码移植至 Windows 11、Ubuntu 22.04 系统，调整设备调用路径后功能正常；

跨数据库测试：连接 openGauss 3.0.0 和 PostgreSQL 14，数据插入、查询无异常；

设备兼容测试：测试开发板内置摄像头/麦克风、USB 摄像头/麦克风，采集与水印嵌入均正常。

智能层-1

一、设计要求

1. **基于 Socket 的音视频传输系统：**设计发送端和接收端两个部分，基于多线程和 Socket 通信技术实现实时音视频采集、编码、传输和播放。
2. **数据库管理：**利用 openGauss 关系型数据库存储音视频文件的元数据和文件存放目录。
3. **多模态生物识别融合：**在基础音视频系统上融入人脸识别和说话人识别技术。要求基于 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，将识别结果进行多模态融合实现身份认证。
4. **数据存储与管理：**对识别出的人员进行人脸抓图和声音采集，并将特征数据存入 openGauss 数据库。

二、设计目的

1. **掌握网络通信技术：**通过实现基于 Socket 的音视频传输系统，深入理解 TCP/UDP 等网络传输协议的工作原理和应用。
2. **理解多线程编程：**在发送端和接收端的实现中使用多线程技术处理并发的音视频采集、编码、传输和播放任务。
3. **学习数据安全与加密：**实现多种加密技术（对称加密 DES、非对称加密 RSA、混合加密方案等）保护音视频数据的传输安全。
4. **应用数据库技术：**使用 openGauss 关系型数据库存储和管理音视频文件的元数据和用户信息。
5. **实现生物识别系统：**学习和应用最新的人脸识别和说话人识别算法，理解深度学习在生物识别中的应用。
6. **掌握多模态融合技术：**通过将多个生物特征（人脸和声纹）进行融合，实现更加准确和可靠的身份认证系统。
7. **开发完整系统应用：**从需求分析、系统设计、编码实现、测试验证等全流程，完成一个具有实际应用价值的音视频通信和生物识别系统。

三、开发环境

3.1 开发板简介

香橙派鲲鹏（OrangePi Kunpeng Pro）是由香橙派与华为联合精心打造的高性能开发板，基于华为鲲鹏处理器架构设计，为人工智能和高性能计算应用提供强大支持。

主要硬件规格如下：

- **处理器**：华为鲲鹏处理器（ARM 架构），高集成度设计，支持 64 位运算。
- **内存配置**：板载 LPDDR4X 内存，支持外接 32GB/64GB/128GB/256GB eMMC 模块，提供灵活的存储扩展。
- **显示输出**：支持双 HDMI 2.0 输出，可同时驱动两个 4K 高清显示屏，满足多屏幕应用场景。
- **扩展接口**：
  - 两个 USB 3.0 接口，支持高速数据传输
  - 一个 USB Type-C 3.0 接口（可用于供电和数据传输）
  - 一个 Micro USB 接口（用于串口调试）
  - 两个 MIPI 摄像头接口（支持高清摄像头扩展）
  - 一个 MIPI 显示屏接口
  - GPIO 接口，便于扩展各种硬件模块
- **存储扩展**：
  - 支持 SATA/NVMe SSD，具备 M.2 2280 接口
  - TF 卡槽，支持微 SD 卡扩展
  - 千兆以太网接口，提供高速网络连接
- **其他功能**：预留电池接口，支持多种电源配置；丰富的 GPIO 和扩展接口，便于二次开发和硬件集成。

操作系统环境：

开发板预装 openEuler 操作系统，这是华为推出的开源、自主、创新的 Linux 发行版，专为信息产业提供稳定高效的平台。openEuler 具有以下特点：

- 与鲲鹏处理器深度适配，充分发挥硬件性能
- 支持大多数 AI 算法原型验证和推理应用开发
- 集成完善的开发工具链，支持 Python、C++ 等多种编程语言
- 提供丰富的系统库和软件包，便于应用开发
- 满足云计算、大数据、分布式存储、高性能计算等多种应用场景的需求

3.2 开发板的配置和启动

3.2.1 SSH 连接开发板

同学们拿到的开发板已经烧录好了系统镜像（未烧录镜像的或者系统存在问题的请参考用户手册烧录镜像后再进行接下来的操作），接下来我们需要通过 SSH 连接到开发板上进行开发工作。

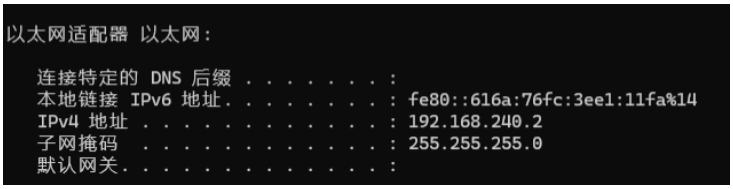
1. 使用网线将开发板的有线网口与笔记本电脑的有线网口相连接。将 HDMI 数据线连接到开发板的 HDMI 接口和显示屏。为开发板接上电源，启动开发板。

2. 等待约 1-2 分钟系统启动完成，在显示屏上看到登录界面。输入用户名 `openEuler` 和密码 `openEuler` 登录系统。

3. 在笔记本电脑上按下 `win+R` 快捷键，输入 `cmd` 打开命令提示符。输入以下命令查看笔记本的有线网卡 IP 地址：

```
ipconfig
```

在命令行输出结果中找到” 以太网” 部分，记下 IPv4 地址。如下图所示：



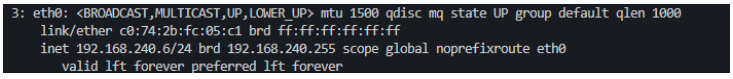
4. 配置开发板的 IP 地址。在开发板桌面右上角点击打开有线网络设置。在有线连接列表中，点击齿轮图标打开有线网络的配置页面。在配置窗口中选择顶部的”IPv4” 选项卡，将 IPv4 的配置方式从自动改为手动。

   在手动配置模式下，将地址栏中的 IP 地址修改为与笔记本电脑同一网段的地址。例如，若笔记本电脑的 IP 地址为 `192.168.240.2`，则应将开发板的 IP 地址设置为 `192.168.240.x`（其中 x 为 1-254 之间的任意数值，但不能与笔记本相同）。本示例中设置为 `192.168.240.6`。配置完成后点击应用。并重新开启有线网络连接以刷新配置。

   配置完成后，在开发板终端中输入以下命令验证 IP 地址是否配置成功：

```
ip a
```

查看输出结果中有线网卡的 IP 地址信息，如下图所示：



5. 在笔记本上打开 VS Code。点击左侧”Extensions”（扩展），搜索”Remote - SSH”，安装由 Microsoft 发布的官方扩展。

6. 安装完成后，点击左侧活动栏中的”Remote Explorer” 图标。在 Remote Explorer 面板中，点击”Add New” 按钮添加新的 SSH 连接。

7. 在输入框中输入 SSH 连接命令，格式为：

```
ssh openEuler@192.168.240.6
```

其中用户名为 `openEuler`，IP 地址为开发板配置的 IP 地址（根据实际配置修改）。

8. 按下 Enter 键，选择 SSH 配置文件保存位置（通常选择默认位置）。配置完成后，SSH 连接会出现在 Remote Explorer 列表中。

9. 在 Remote Explorer 中，将鼠标悬停在 SSH 连接条目上，点击”Connect to Host” 按钮进行连接。或按 `Ctrl+Shift+P` 打开命令面板，输入”Remote - SSH: Connect to Host...” 选择连接。

10. 如果是第一次连接，VS Code 会询问目标主机的平台类型，选择”Linux”。然后输入开发板密码 `openEuler`。

11. 等待 VS Code 在远程主机上安装所需的组件。连接成功后，VS Code 的窗口标题栏会显示”SSH: 192.168.240.6”（实际 IP 地址根据配置显示）。

12. 点击菜单”File” → ”Open Folder” 打开开发板上的文件夹，即可开始远程开发工作。

3.2.2 开发应用程序安装与配置

本项目的开发主要采用 VS Code 编辑器结合 Python 编程语言进行实现。openEuler 系统已内置 Python 和 VS Code，可直接使用，无需额外安装编译器。

环境确认和库安装步骤如下：

1. 在远程 SSH 终端中，确认 Python 和 pip 是否已安装。输入以下命令查看 Python 版本：

```
python3 --version
pip3 --version
```

2. 根据项目需求，需要安装多个 Python 库来支持音视频处理、深度学习推理、数据库访问和 GUI 开发。使用 pip 命令行工具安装所需的依赖库：

```
pip3 install numpy opencv-python pillow
pip3 install torch torchaudio torchvision
pip3 install psycopg2-binary soundfile pyaudio
pip3 install requests urllib3
```

主要库说明：

- **numpy**：数值计算库，用于矩阵和数组运算
- **opencv-python**：计算机视觉库，用于图像处理和视频采集
- **pillow**：图像处理库，支持图像格式转换和显示
- **torch & torchaudio**：PyTorch 深度学习框架及其音频处理扩展，用于人脸识别和说话人识别模型推理
- **torchvision**：PyTorch 的计算机视觉工具包，包含预训练的深度学习模型
- **psycopg2-binary**：PostgreSQL/openGauss 数据库的 Python 驱动，用于连接和操作数据库
- **soundfile**：音频文件读写库，支持 WAV、OGG 等多种格式
- **pyaudio**：音频输入输出库，用于实时音频采集和播放

3. 在 VS Code 中打开远程开发板的工作目录。创建项目文件夹后，在该文件夹中进行 Python 代码编写。建议将发送端和接收端代码分别放在不同的模块或子目录中，便于代码组织和管理。

4. 开发过程中，可在 VS Code 的集成终端中直接运行 Python 脚本：

```
python3 your_script.py
```

VS Code 会自动识别 openEuler 远程环境中的 Python 解释器，并输出运行结果。

3.2.3 openGauss 数据库连接

openEuler 系统内置 openGauss 数据库。openGauss 是华为自主研发的开源关系型数据库，提供极致性能和完整的企业级功能。

1. 切换到 openGauss 用户。openGauss 数据库进程由专门的 opengauss 用户管理，对数据库的操作需要在该用户下进行。在开发板终端中输入：

```
su - opengauss
```

2. 启动数据库。输入以下命令启动 openGauss 数据库服务：

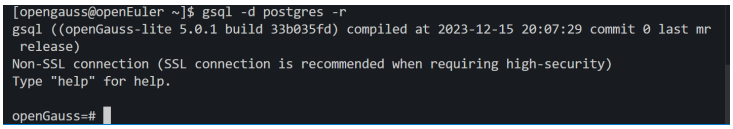
```
gs_ctl start
```

等待数据库启动完成。

3. 登录数据库。输入以下命令进入 openGauss 数据库管理界面：

```
gsql -d postgres -r
```

登录成功后，命令行提示符会变为 `openGauss=#`，如下图所示：



注意：后续所有数据库操作必须在 `openGauss=#` 环境下执行。

4. 修改 opengauss 用户的密码。在首次使用前，需要修改 opengauss 账户的默认密码，才能进行其他数据库操作：

```
ALTER ROLE opengauss PASSWORD 'your_password';
```

将 `your_password` 替换为你设置的新密码。

5. 创建日常操作账户。建议创建一个专用账户用于日常数据库操作，而不是直接使用 opengauss 管理员账户。输入以下命令创建新账户：

```
CREATE USER your_username PASSWORD 'your_password';
ALTER USER your_username SYSADMIN;
```

将账户名和密码替换为实际值。

6. 退出数据库。按下 `Ctrl+D` 或输入 `\q` 退出数据库管理界面：

```
\q
```

7. 退回 root 用户。按下 `Ctrl+D` 退出 opengauss 用户，返回 root 用户。

8. 设置 IP 白名单。编辑 `data/pg_hba.conf` 文件，添加允许的 IP 地址记录。切换到 opengauss 用户并编辑配置文件：

```
su - opengauss
cd data/
vim pg_hba.conf
```

在文件末尾添加以下行以允许所有 IP 连接（生产环境应限制具体 IP）：

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

9. 修改监听地址和加密方式。编辑 `data/postgresql.conf` 文件，配置数据库监听的 IP 地址和加密方式：

```
vim postgresql.conf
```

找到以下配置项并修改：

```
listen_addresses = '*'
password_encryption_type = 1
```

第一行设置数据库监听所有网络接口，第二行设置密码加密方式。

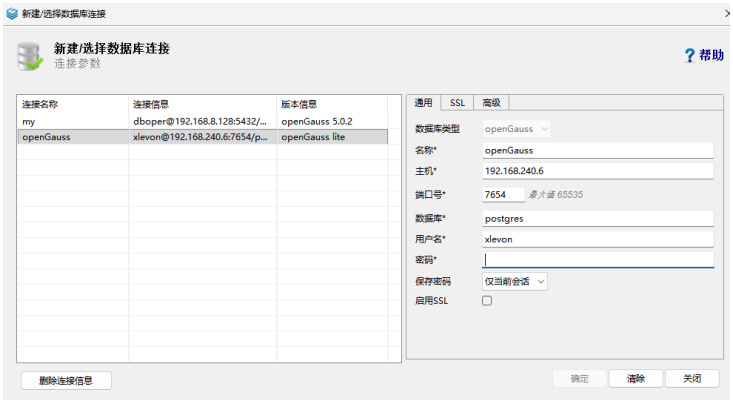
10. 重启数据库使配置生效。输入以下命令停止并重启数据库：

```
gs_ctl stop
gs_ctl restart
```

11. 配置防火墙。返回 root 用户后，查询数据库服务端口并开放防火墙权限。openGauss 默认端口为 5432，若配置为其他端口（如 7654），则按实际端口修改：

```
exit
netstat -antp
sudo firewall-cmd --permanent --add-port=5432/tcp
sudo systemctl reload firewalld
```

12. 使用 DataStudio 验证配置（DataStudio 的配置具体见张老师的教程文档）。在笔记本电脑上打开 DataStudio 客户端，创建新的数据库连接，输入开发板的 IP 地址、端口（默认 5432）、用户名和密码。连接成功后，即可看到数据库对象树，表明数据库配置完成。如下图所示：



四、设计思路

4.1 系统总体架构

本系统采用客户端-服务器（C/S）架构，由多模态发送端（Server）、音视频接收端（Client）和 openGauss 数据库服务器三部分组成。系统通过 Socket 协议实现各部分间的通信，具体架构关系如下：

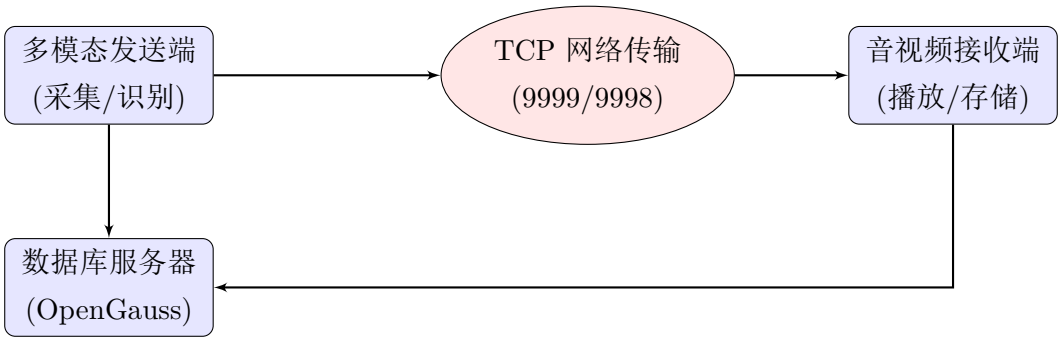


图 1: 系统总体架构图

- **多模态发送端（服务器端）：**部署在香橙派开发板上，负责实时音视频采集、图像和语音信号处理、人脸识别和说话人识别、多模态特征融合，并将识别结果和相关数据存储到 openGauss 数据库。
- **音视频接收端（客户端）：**通过网络连接到发送端，接收并解码音视频流，进行本地显示和播放，并支持录制功能。接收端同样可连接到数据库进行数据查询和存储。
- **openGauss 数据库：**集中存储用户信息、生物特征向量（人脸特征、声纹特征）、音视频文件的元数据和原始文件，为整个系统提供数据持久化和管理支持。

4.2 音视频网络传输设计

为了实现低延迟、高稳定的实时音视频传输，系统采用基于 TCP 的双通道架构：

- **视频传输通道（TCP 端口 9999）：**发送端使用 OpenCV 从摄像头采集 RGB 视频帧，经 JPEG 压缩后进行序列化。在 TCP 层面采用”4 字节长度头 + 序列化数据体” 的自定义协议以解决 TCP 粘包问题。发送端实现帧内存池机制，复用内存空间以减少频繁的内存分配和垃圾回收带来的性能抖动。
- **音频传输通道（TCP 端口 9998）：**发送端使用 PyAudio 从麦克风采集 PCM 原始音频流。引入无锁环形缓冲区（Lock-Free Ring Buffer）在音频采集线程和网络发送线程之间实现高效的数据传递，避免锁竞争导致的音频丢帧或卡顿。
- **并发控制：**系统采用多线程设计，将视频采集、音频采集、网络发送和生物识别处理分配到不同线程，充分利用系统资源提高并发处理能力。

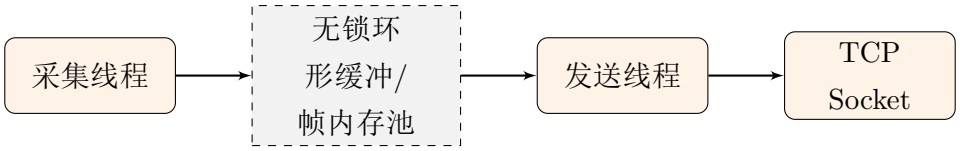


图 2: 音视频传输处理流程图

4.3 多模态生物识别融合设计

为了实现高精度的身份认证，系统采用”双层融合”架构，具体流程如图3所示：

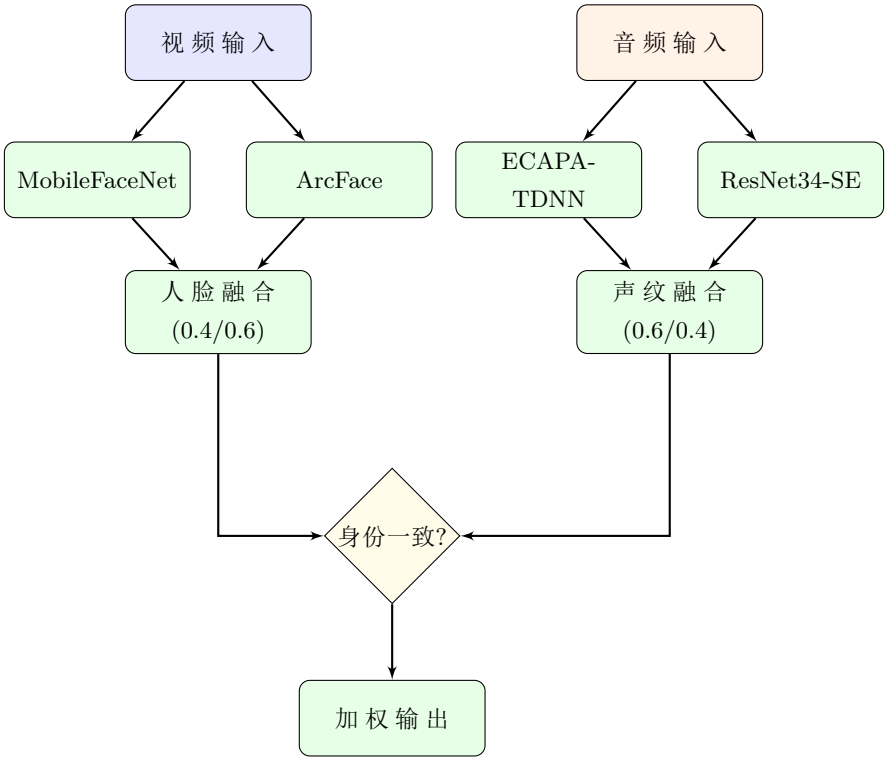


图 3: 多模态生物识别融合流程图

- **第一层：模态内模型融合**。人脸识别采用 MobileFaceNet 和 ArcFace 两种深度学习模型进行融合，充分利用两个模型的不同网络结构和特征学习能力来提升单模态识别的鲁棒性。说话人识别同样采用两种不同算法进行融合。通过加权平均（例如人脸融合权重为 0.4:0.6，声纹融合权重为 0.6:0.4）来综合两个模型的识别结果。

- **第二层：模态间决策融合**。将人脸识别和说话人识别的融合结果进行决策级融合，通过一致性判决逻辑来综合视觉特征和听觉特征。当两个模态的识别结果一致性较高时，系统输出高置信度的认证结果；当存在不一致时，系统采用加权融合策略输出最终认证结果。

- **特征存储与检索：**识别后的人脸特征向量和声纹特征向量以二进制大对象（BYTEA）格式存储在 openGauss 数据库中，支持高效的特征检索与比对。原始的注册人脸图像和音频文件也直接存储在数据库中，增强数据的安全性和一致性。

4.4 数据库存储与管理设计

系统采用 openGauss 关系型数据库作为后端存储，主要存储内容包括：

- **用户信息表**：存储注册用户的基本信息（如用户 ID、用户名、注册时间等）。
- **人脸特征表**：存储每个用户的人脸特征向量，支持快速的特征相似度检索和身份匹配。
- **声纹特征表**：存储每个用户的声纹特征向量，用于说话人识别和验证。
- **媒体文件表**：存储注册和识别过程中产生的原始人脸图像和音频数据，采用 BYTEA 二进制格式直接存储在数据库中，避免依赖外部文件系统。
- **识别记录表**：记录每次身份识别的结果、时间戳、置信度等信息，便于系统审计和性能分析。

系统设计充分利用 openGauss 的企业级功能（如事务支持、约束检查、索引优化等），确保数据的一致性和查询效率。同时，数据库连接采用连接池管理，提高并发性能。

五、设计的实现

5.1 实现步骤概述

本系统的实现分为 5 个主要阶段。首先进行数据库初始化与管理，建立 openGauss 连接池和表结构，用于存储特征向量和多媒体数据。其次是音视频采集与处理，初始化摄像头和麦克风，设置合适的采样率和帧率以适应嵌入式设备（openEuler）。第三步部署多模态识别系统，包括人脸识别、说话人识别及融合策略的实现。第四步建立网络传输机制，通过 TCP 双通道（视频端口 9999、音频端口 9998）实现低延迟的实时传输。最后是 GUI 界面与交互设计，使用 Tkinter 实现用户交互界面，实时显示识别结果和系统状态。

5.2 模块实现

在数据库管理模块中，我们使用 openGauss 数据库存储系统的所有特征数据。采用连接池管理机制（minconn=2，maxconn=8），以支持多线程并发访问。系统创建三个主要数据表：face_embeddings 存储人脸特征和注册的人脸图像、speaker_embeddings 存储说话人特征和语音样本、face_recognition_log 记录识别历史。关键的设计思想是采用 BYTEA 二进制格式直接在数据库中存储人脸图像和音频数据，避免依赖本地文件系统。对于特征的保存和查询，采用”UPDATE-IF-NOT-FOUND INSERT” 的策略，先尝试更新已有记录，如果没有找到再执行插入操作，这样既能处理重复注册又能保持代码简洁。特征向量通过 pickle 序列化为字节流存储，支持高效的特征检索与比对。

```
# 数据库连接池初始化
self.db_pool = pool.ThreadedConnectionPool(
    minconn=2, maxconn=8,
    host='192.168.240.6', port='7654',
    database='record', user='xlevon', password='...')
```

```

# 保存人脸特征（更新或插入）
cur.execute("""
    UPDATE face_embeddings SET embedding=%s, image_data=%s
    WHERE name=%s AND model_type=%s
""", (embedding_bytes, image_bytes, name, model_type))
if cur.rowcount == 0:
    cur.execute("""
        INSERT INTO face_embeddings (...) VALUES (...)
    """)
```

在音频处理模块中，我们实现了两个关键的性能优化结构：无锁环形缓冲区（LockFreeRingBuffer）和帧内存池（FramePool）。无锁环形缓冲区采用 SPSC（单生产者单消费者）模型，避免了多线程锁竞争导致的性能下降，特别是在高频音频采集场景中。缓冲区的大小固定，当新数据到达时自动覆盖旧数据，这样既能保证恒定的内存占用，又能提供足够的缓冲时间。帧内存池则预分配固定数量的图像缓冲区（如 10 个 480×640×3 的帧），避免了频繁的内存分配和垃圾回收带来的性能波动，这对嵌入式设备特别重要。

```
class LockFreeRingBuffer:
    def __init__(self, capacity):
        self.buffer = np.zeros(capacity, dtype=np.int16)
        self.write_pos = 0
        self.read_pos = 0
        self.available = 0

    def write(self, data):
        # 环形写入处理环绕...
        self.write_pos = end_pos % self.capacity
        self.available = min(self.available + len(data),
                             self.capacity)

    def read(self, size):
        # 环形读取处理环绕...
        self.read_pos = end_pos % self.capacity
        self.available -= size
        return result

class FramePool:
    def __init__(self, frame_shape=(480, 640, 3), pool_size=10):
        self.pool = [np.zeros(frame_shape) for _ in range(pool_size)]
        self.available = list(range(pool_size))

    def acquire(self):
        if self.available:
            idx = self.available.pop(0)
            return self.pool[idx], idx
        return None, None
```

```
30
31      def release(self, idx):
32          self.available.append(idx)  # 回收帧到池
```

人脸识别模块支持三种工作模式：MobileFaceNet 模型、ArcFace 模型以及两者的融合模式。识别流程包括人脸检测定位、人脸对齐校准、特征向量提取、数据库特征比对和结果融合。在融合模式下，系统同时使用两个预训练模型分别提取特征，对提取的特征进行 L2 归一化确保向量长度为 1，然后按预定的权重比例（MobileFaceNet 0.4、ArcFace 0.6）进行加权融合，最后与数据库中已注册人员的特征计算余弦相似度。为了降低计算负荷，系统采用帧跳过策略，每 3 帧进行一次识别，识别结果存储在异步识别队列中，实现了识别处理与 UI 渲染的解耦。

```
def recognize_face(self, face_image, threshold=0.35):
    # MobileFaceNet特征
    embedding_mbf = self.extract_embedding_mbf(face_image)
    # ArcFace特征
    embedding_arc = self.extract_embedding_arc(face_image)

    # 融合：权重平均 (0.4:0.6)
    embedding = (0.4 * embedding_mbf + 0.6 * embedding_arc)
    embedding = embedding / np.linalg.norm(embedding)  # L2归一化

    # 加载已注册特征，计算相似度
    db_embeddings = self.db_manager.load_face_embeddings('fusion')
    best_match = max(db_embeddings.items(),
                     key=lambda x: cosine_similarity(embedding, x[1]))

    if best_match[1] > threshold:
        return best_match[0], best_match[1]
    return "未知", 0.0
```

说话人识别模块同样支持 ECAPA-TDNN、ResNet34-SE 以及融合三种模式。识别流程包括 VAD 人声检测确保有真实语音、音频预处理（归一化和静音修剪）、特征向量提取、数据库比对和融合决策。系统对注册的音频进行质量检查，验证 RMS 能量大于 100 且语音帧比例超过 30%，确保注册数据的有效性。在融合模式下，ECAPA-TDNN 权重为 0.6，ResNet 权重为 0.4，相比于人脸识别的权重分配体现了两种模型的特性差异。与人脸识别类似，系统采用异步识别架构，将说话人识别分配到低优先级线程，避免阻塞主线程。

```
def recognize_speaker(self, audio_data, threshold=0.35):
    # 音频预处理
    audio_data = self._normalize_audio(audio_data)
    audio_tensor = torch.from_numpy(audio_data).unsqueeze(0)

    # ECAPA-TDNN和ResNet特征提取
    emb_ecapa = self.ecapa_classifier.encode_batch(audio_tensor)
    emb_resnet = self.resnet_classifier.encode_batch(audio_tensor)
```

```
9
10      # 融合：加权融合 (0.6:0.4)
11      embedding = (0.6 * emb_ecapa + 0.4 * emb_resnet)
12      embedding = embedding / np.linalg.norm(embedding)
13
14      # 数据库比对
15      db_embeddings = self.db_manager.load_speaker_embeddings('fusion')
16      best_match = max(db_embeddings.items(),
17                       key=lambda x: cosine_similarity(embedding, x[1]))
18
19      if best_match[1] > threshold:
20          return best_match[0], best_match[1]
21      return "未知", 0.0
```

网络传输模块采用 TCP 双通道架构，分离视频和音频的传输路径。视频通道使用 TCP 9999 端口，工作流程为：OpenCV 采集的帧经 JPEG 压缩后使用 Pickle 序列化，然后添加 4 字节长度头表示数据大小，最后通过网络发送。音频通道使用 TCP 9998 端口，PCM 原始音频数据同样采用 4 字节长度头的协议格式。在服务器端（Sender），视频编码线程将摄像头采集的帧进行 JPEG 压缩并放入发送队列，视频发送线程从队列中取出已编码数据并通过网络发送，这样实现了编码和网络传输的解耦。音频发送线程同时采集麦克风数据、写入无锁环形缓冲区以供说话人识别使用、并发送到网络。在客户端（Receiver），接收线程通过读取 4 字节长度头确定每个消息的大小，然后接收完整的数据包，反序列化后解码恢复为图像或音频。优化策略包括设置 TCP_NODELAY 禁用 Nagle 算法降低延迟、扩大 SO_SNDBUF 和 SO_RCVBUF 到 65536 字节、以及限制帧队列大小（maxsize=2）自动丢弃过期帧以保持同步。

```
1  # 服务器端发送
2  def send_video_loop(self):
3      while self.camera_running:
4          data = self.send_queue.get()  # 已编码数据
5          message_size = struct.pack("<L", len(data))
6          self.video_client.sendall(message_size + data)
7
8  def send_audio_loop(self):
9      while self.camera_running:
10         audio_data = self.audio_stream.read(2048)
11         self.speaker_audio_ring_buffer.write(np.frombuffer(...))
12         size = struct.pack("<L", len(audio_data))
13         self.audio_client.sendall(size + audio_data)
14
15 # 客户端接收
16 def receive_frame(self):
17     # 接收4字节长度头，然后接收完整帧数据
18     msg_size = struct.unpack("<L", data[:4])[0]
19     frame_data = self.video_socket.recv(msg_size)
20     frame = pickle.loads(frame_data)
21     frame = cv2.imdecode(frame, cv2.IMREAD_COLOR)
```

```
    return frame
```

GUI 与交互模块使用 Tkinter 实现用户界面，主要功能包括实时视频显示与人脸识别结果标注、说话人识别实时输出、模型选择下拉菜单、系统启动和停止控制、已注册人员的管理（注册新人、删除已有人员）、视频加音频的同步录制与保存、以及系统日志输出与性能监测。界面布局分为左侧视频显示区域和右侧控制面板，视频中的人脸用边界框标注，识别结果（姓名和置信度百分比）显示在框的上方。系统支持将录制的视频和音频同步保存到数据库，用户可以选择导出格式和保存位置，系统在后台尝试使用 ffmpeg 合并音视频为单一文件。

```
# 实时人脸标注
cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)
cv2.putText(frame, f"{name} {confidence:.2%}",
            (x1, y1-10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)

# 录制功能
def start_recording(self):
    self.temp_video_path = f"temp_video_{int(time.time())}.avi"
    self.video_writer = cv2.VideoWriter(...)
    self.recording = True

def stop_recording(self):
    self.video_writer.release()
    # 显示保存对话框，用户选择输出格式和路径
    output_path = filedialog.asksaveasfilename(...)
```

5.3 性能优化策略

系统采用多线程设计，根据任务的时间敏感性分配不同的优先级。音频采集与发送线程设置为 HIGH 优先级，确保音频的实时性和同步性；视频编码与发送线程设置为 MEDIUM 优先级，视频可以有一定的延迟而不影响用户体验；异步人脸识别线程设置为 LOW 优先级，不阻塞视频显示。内存管理方面，帧内存池预分配缓冲区避免频繁分配回收导致的垃圾回收停顿，环形缓冲区固定大小恒定占用内存，帧队列限制大小自动丢弃过期帧保持同步。网络方面，TCP_NODELAY 禁用 Nagle 算法降低延迟，扩大发送接收缓冲区到 64KB 支持更高的吞吐量，动态监测定期输出传输性能指标（带宽、延迟）。识别优化方面，帧跳过每 3 帧识别一次降低人脸识别的计算负荷，说话人识别每 3 秒进行一次，融合权重调优体现了不同模型的优势特性。

六、系统测试与对比分析

6.1 人脸识别模型对比测试

本系统集成了 MobileFaceNet 和 ArcFace 两种模型，表1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 1: 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%	4.0
ArcFace (ResNet50)	85.6	99.80%	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。

6.2 说话人识别模型对比测试

针对 ECAPA-TDNN 和 ResNet34-SE 模型，表2展示了它们在 VoxCeleb1-O 测试集上的等错误率 (EER) 基准性能。

表 2: 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%	6.2M
ResNet34-SE (L)	2.17%	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。

6.3 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3所示。

表 3: 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。

6.4 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。

6.4.1 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4所示。

图 4: 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，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。

已注册人员管理界面如图5所示，提供了完整的用户信息查看与管理功能。

图 5: 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。

6.4.2 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6所示。

图 6: 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，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

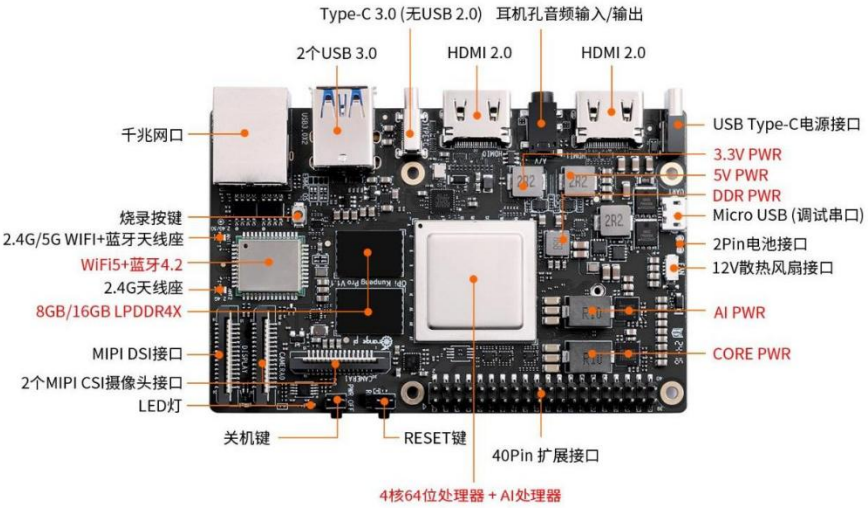
点击” 退出” 按钮安全关闭程序。当检测到网络断开时，系统自动进入重连模式，每隔 3 秒尝试重新连接发送端。

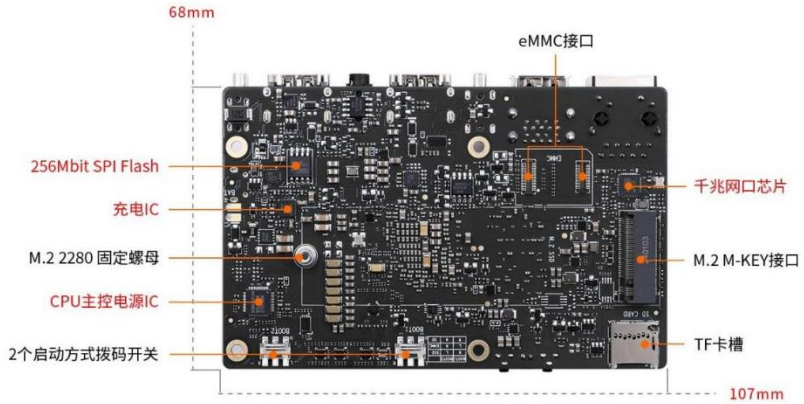
智能层-2

一．开发环境

1.1 开发板简介

Orange Pi Kunpeng Pro 开发板是香橙派联合华为精心打造的高性能开发板，其搭载了鲲鹏处理器，提供了 8GB 和 16GB 两种内存版本。 Kunpeng Pro 开发板结合了鲲鹏全栈根技术，全面使能高校计算机系统教学和原生开发。同时支持 FPGA+ARM，从体系结构、数字逻辑设计、操作系统和编译，再到嵌入式开发，可以基于同一套体系结构和一套开发板实现贯穿打通。





1.2 开发板配置与启动需要的配件

1) TF 卡，最小 32GB 容量的 class10 级或以上的高速闪迪卡。强烈推荐使用 64GB 或以上容量的 TF 卡。

2) TF 卡读卡器，用于读写 TF 卡。

3) HDMI 转 HDMI 连接线，用于将开发板连接到 HDMI 显示器或者电视进行显示。

4) 电源， Type-C 接口的 20V PD-65W 适配器。

5) 拓展坞（香橙派提供的 usb 接口不够用，不足够外接麦克风、摄像头）

6) 摄像头模块

7) 声音采集模块

8) 百兆或者千兆网线，用于将开发板连接到因特网
9) 硬件搭载好的（配套金属外壳）香橙派开发板

10) 安装有 Ubuntu 22.04 和 Windows 操作系统的 X64 电脑设备。

1	**Ubuntu22.04 PC**	可选，用于编译 **Linux** 源码
2	**Windows PC**	用于烧录 **openEuler** 镜像

11) 下载开发板的镜像和相关资料

开发板资料下载页面的链接如下所示：

http://www.orangepi.cn/html/hardWare/computerAndMicrocontrollers/service-and-support/Orange-Pi-kunpeng.html

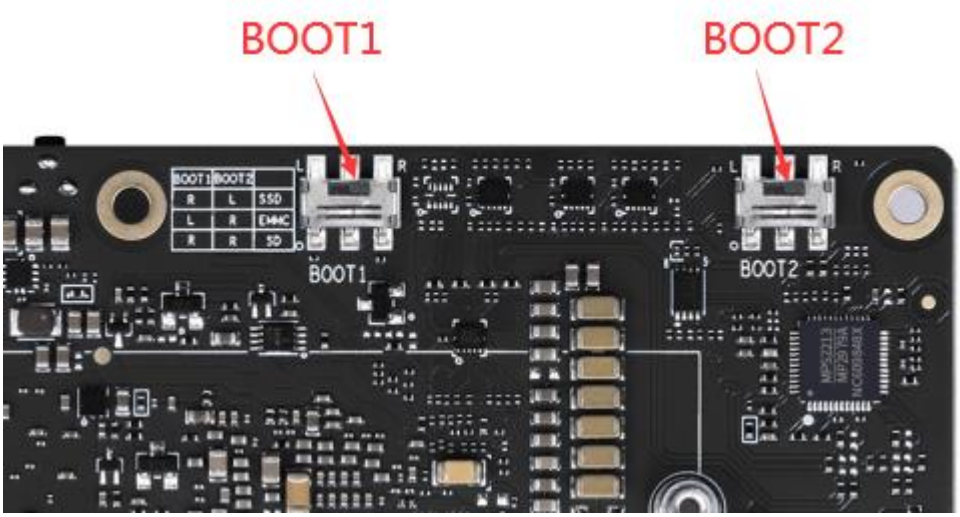
1.3 烧写 Linux 镜像到 TF 卡中的方法

使用基于 **Windows PC** 将 **Linux** 镜像烧写到 **TF** 卡的方法（openEuler 镜像）

注意事项：

开发板支持从 **TF** 卡、 **eMMC** 和 **SSD**（支持 NVMe SSD 和 SATA SSD）启动。具体从哪个设备启动是由开发板背面的两个拨码（BOOT1 和 BOOT2）开关来控制

的。



BOOT1 和 BOOT2 两个拨码开关都支持左右两种设置状态，所以总共有 4 种设置状态，开发板目前只使用了其中的三种。不同的设置状态对应的启动设备如下表

拨码开关 **BOOT1**	拨码开关 **BOOT2**	对应的启动设备
左	左	未使用
右	左	**SATA SSD** 和 **NVMe SSD**
左	右	**eMMC**
右	右	**TF** 卡

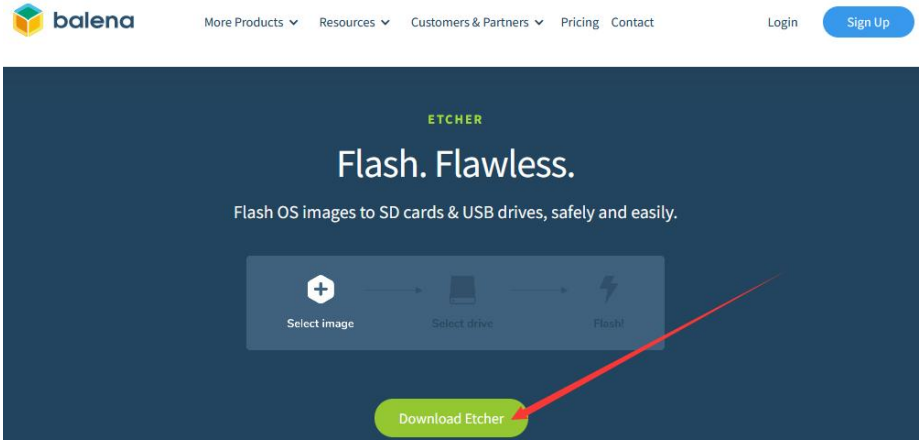
SATA SSD 和 NVMe SSD 的启动方式对应的拨码开关的设置状态是一样的。这两种启动方式是通过 M2_TYPE 引脚的电平来自动区分的。

切换拨码开关后必须重新拔插电源上下电才能让新的启动设备选项生效。通过开发板的复位按键来复位系统是不会让拨码开关新设置的配置生效的。

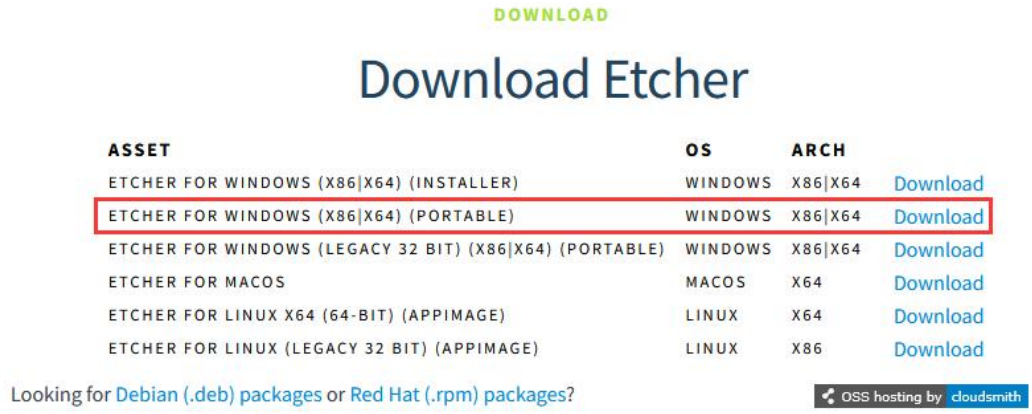
1) 首先准备一张 32GB 或更大容量的 TF 卡（推荐使用 64GB 或以上容量的 TF 卡），TF 卡的传输速度必须为 class10 级或 class10 级以上，建议使用闪迪等品牌的 TF 卡。
2) 然后把 TF 卡插入读卡器，再把读卡器插入电脑。
3) 从开发板的资料下载页面下载想要烧录的 Linux 镜像压缩包。
4) 然后下载用于烧录 Linux 镜像的软件——**balenaEtcher**，下载地址为：

https://www.balena.io/etcher/

5) 进入 balenaEtcher 下载页面后，点击绿色的下载按钮会跳到软件下载的地方。

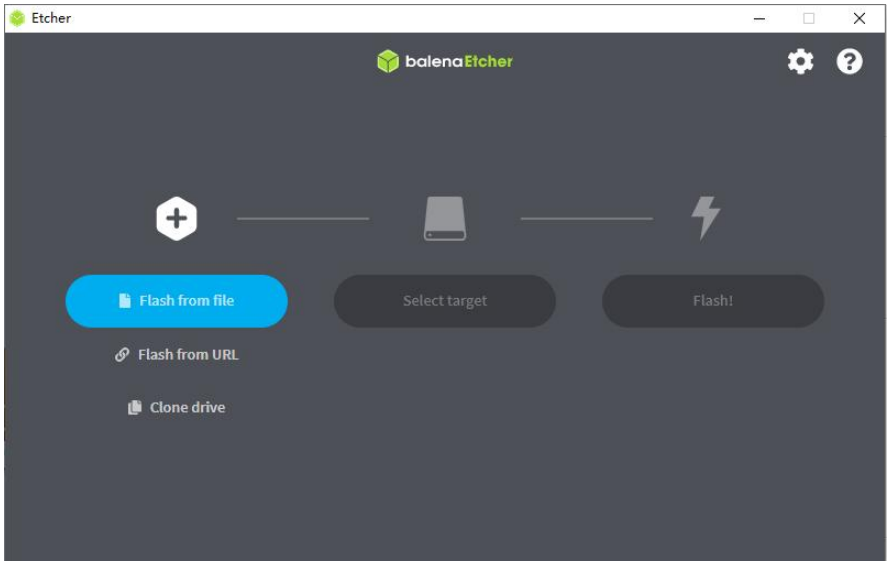


6) 然后可以选择下载 Portable 版本的 balenaEtcher， Portable 版本无需安装，

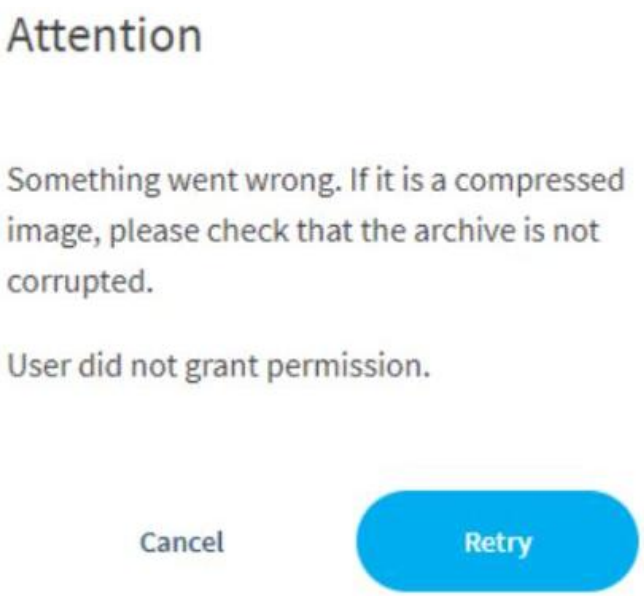


双击打开就可以使用

7) 打开后的 balenaEtcher 界面如下图所示：



打开 **balenaEtcher** 时如果提示下面的错误：



请选择 **balenaEtcher** 后点击右键，然后选择以管理员身份运行。

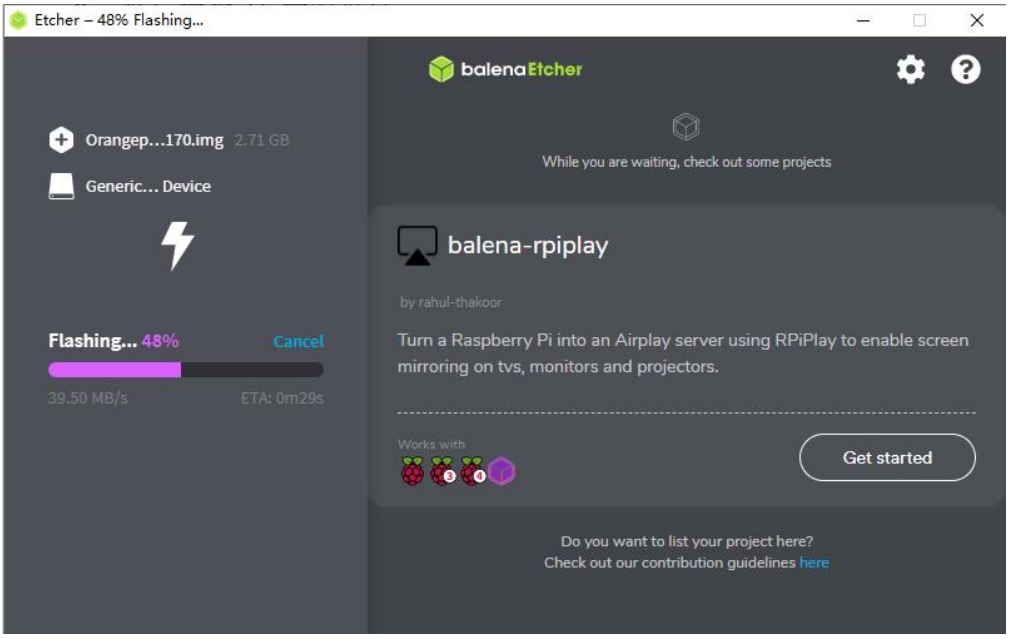
8) 使用 balenaEtcher 烧录 Linux 镜像的具体步骤如下所示：

a. 首先选择要烧录的 Linux 镜像文件的路径。

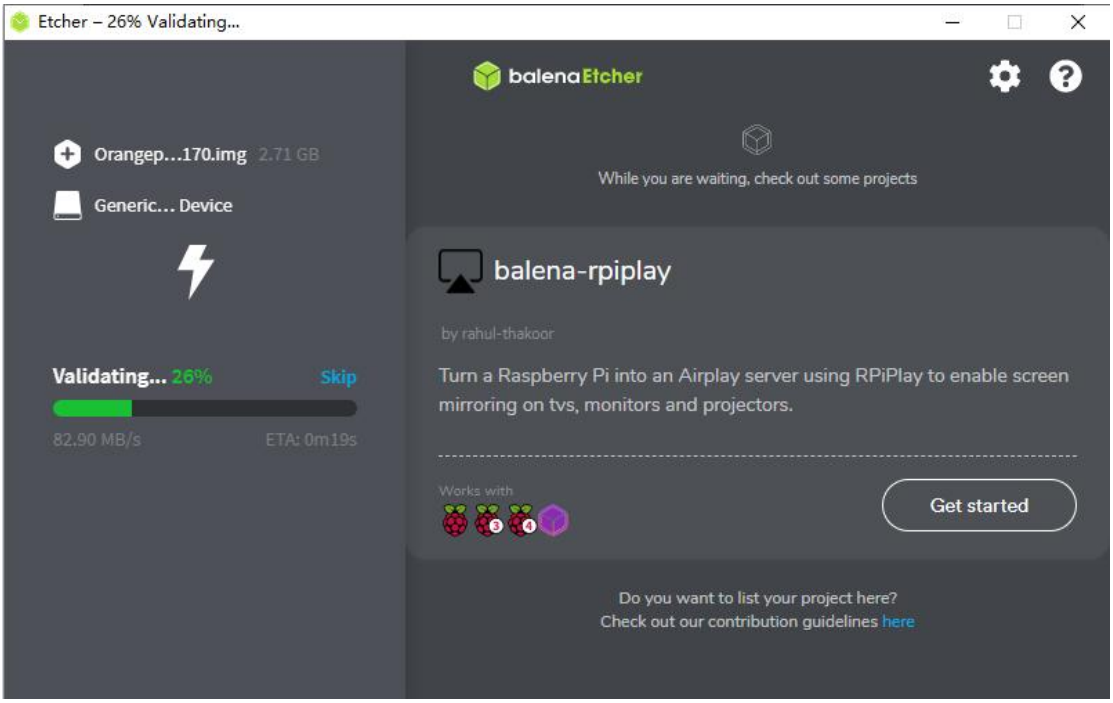
b. 然后选择 TF 卡的盘符。

c. 最后点击 Flash 就会开始烧录 Linux 镜像到 TF 卡中。

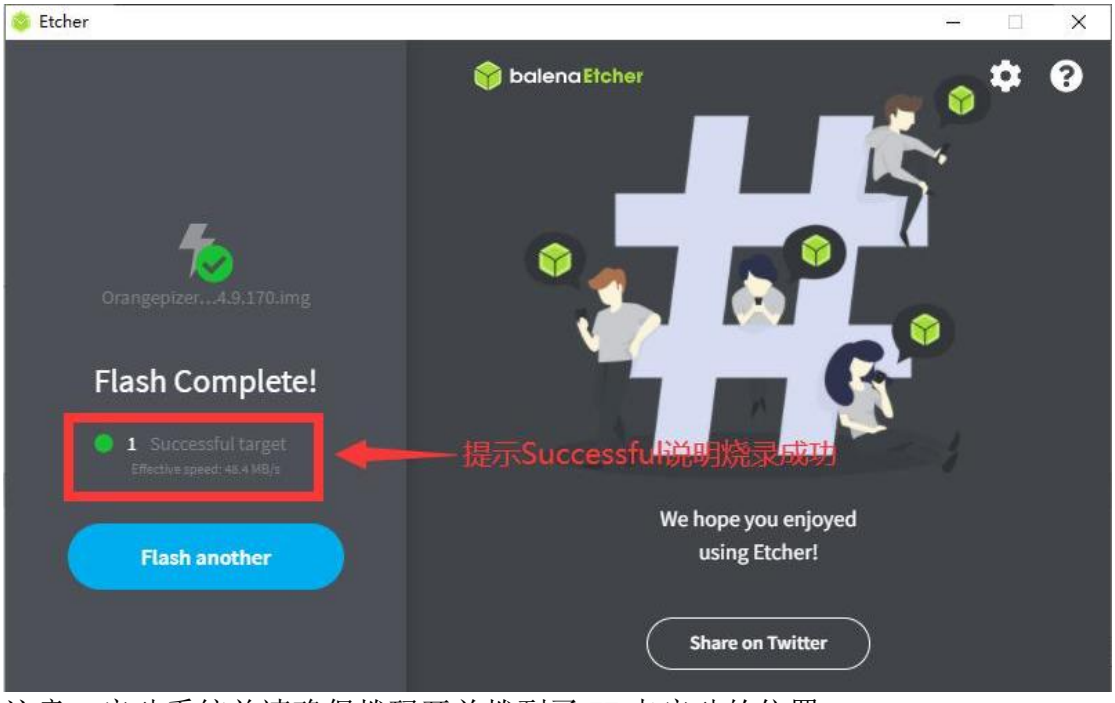
9) balenaEtcher 烧录 Linux 镜像的过程显示的界面如下图所示，另外进度条显示紫色表示正在烧录 Linux 镜像到 TF 卡中。



10）Linux 镜像烧录完后， balenaEtcher 默认还会对烧录到 TF 卡中的镜像进行校验，确保烧录过程没有出问题。如下图所示，显示绿色的进度条就表示镜像已经烧录完成， balenaEtcher 正在对烧录完成的镜像进行校验。



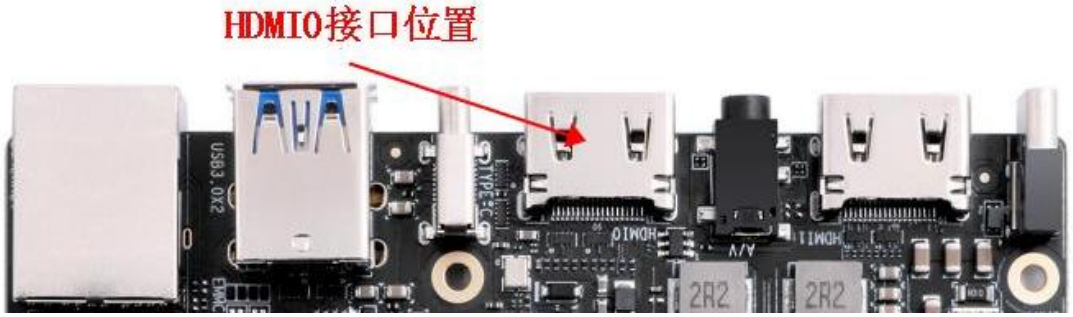
11）成功烧录完成后 balenaEtcher 的显示界面如下图所示，如果显示绿色的指示图标说明镜像烧录成功，此时就可以退出 balenaEtcher，然后拔出 TF 卡插入到开发板的 TF 卡槽中使用了。



注意，启动系统前请确保拨码开关拨到了 TF 卡启动的位置。

1.4 开发板启动步骤

1) 将烧录好镜像的 TF 卡或者 eMMC 模块或者 SSD 插入开发板对应的插槽中。
2) 发板有两个 HDMI 接口（目前只有 **HDMI0** 支持显示 **Linux** 系统的桌面，**HDMI1** 显示 **Linux** 系统桌面的功能还需等软件更新），如果想显示 Linux 系统的桌面，可以将开发板的 HDMI0 接口连接到 HDMI 显示器。

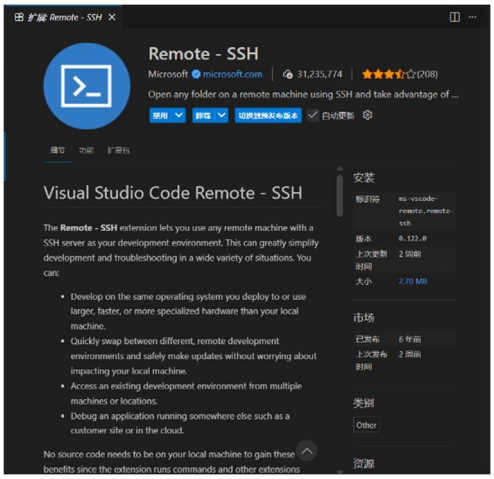


3) 开发板有 USB 接口，可以接上 USB 鼠标和键盘，来控制开发板。
4) 开发板有千兆以太网口，插入网线用来实现网络通信。
5) 然后需要连接一个 20V PD-65W 的 Type C 接口的电源，电源接口的位置如下图所示：

6) 然后打开电源适配器的开关，如果一切正常，等待一段时间后， HDMI 显示器就能看到 Linux 系统的登录界面了

1.5 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，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。



1)　数据库安装以及设置

【1】数据库安装（联网）

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：

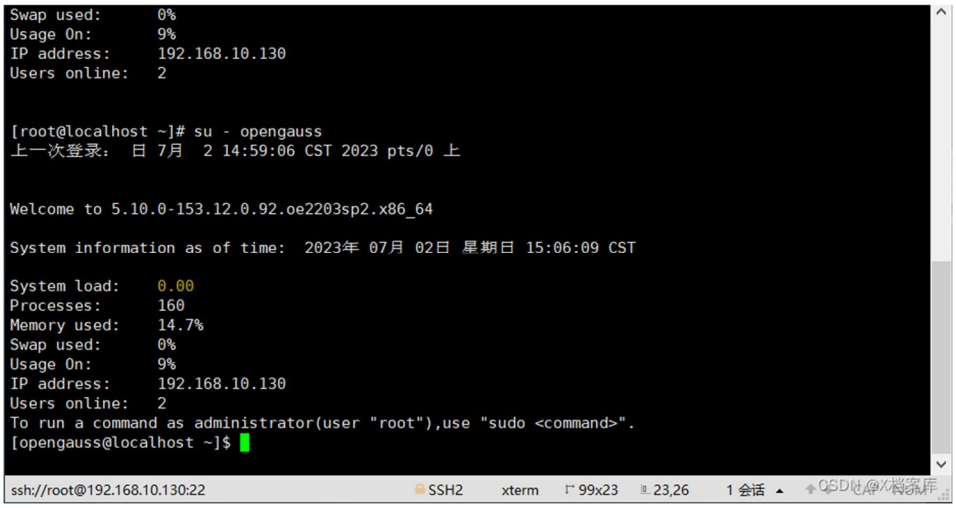
```
yum install opengauss -y
```

【2】数据库管理

- 切换 opengauss 用户

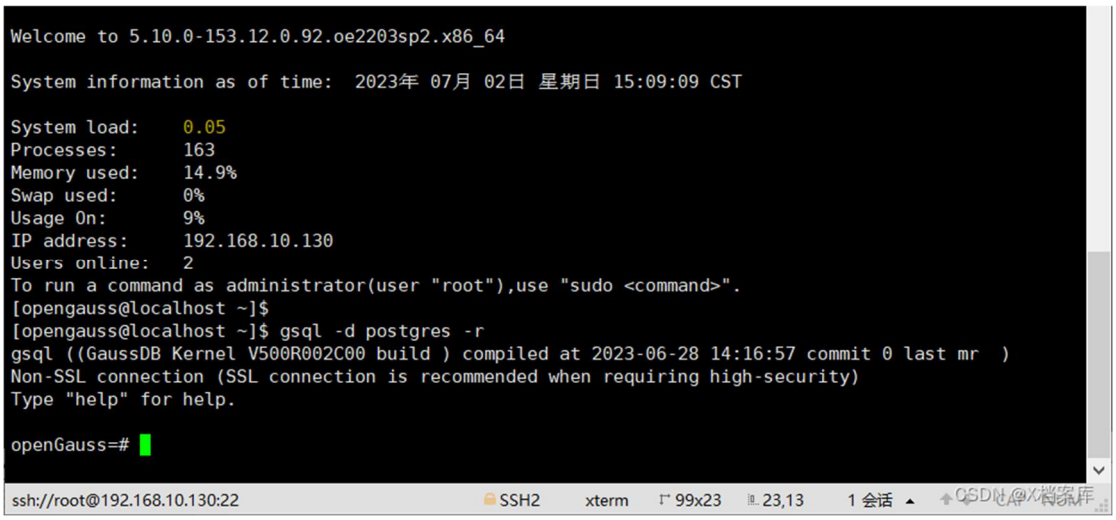
openGauss 数据库进程的管理用户为 opengauss，对数据库的常用操作，需要切换到该用户下进行。

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



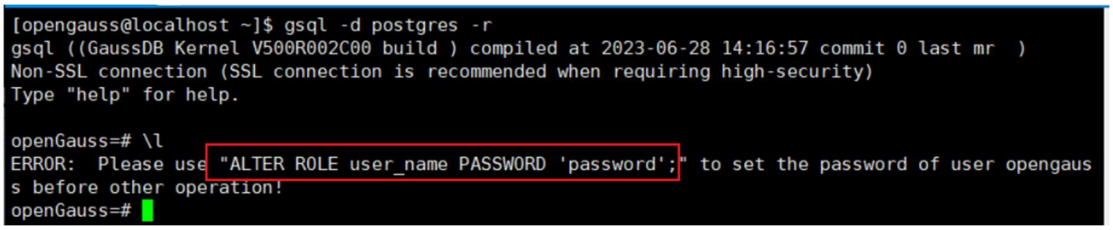
- 登录数据库

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

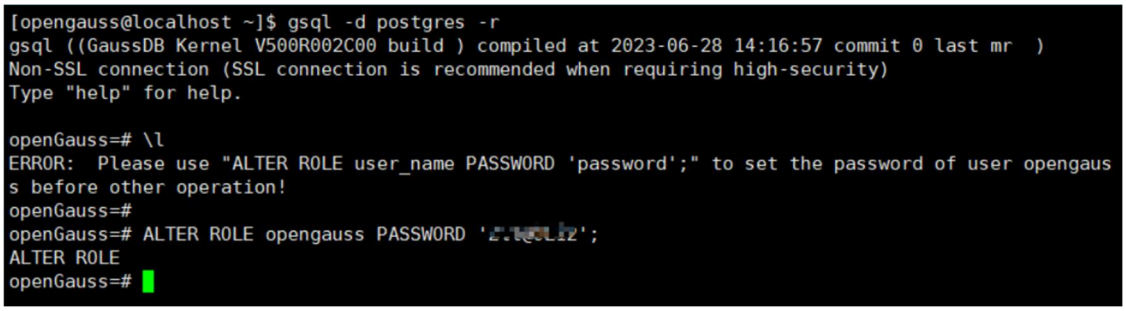


- 显示已有的 database

```
openGauss=# \l
```

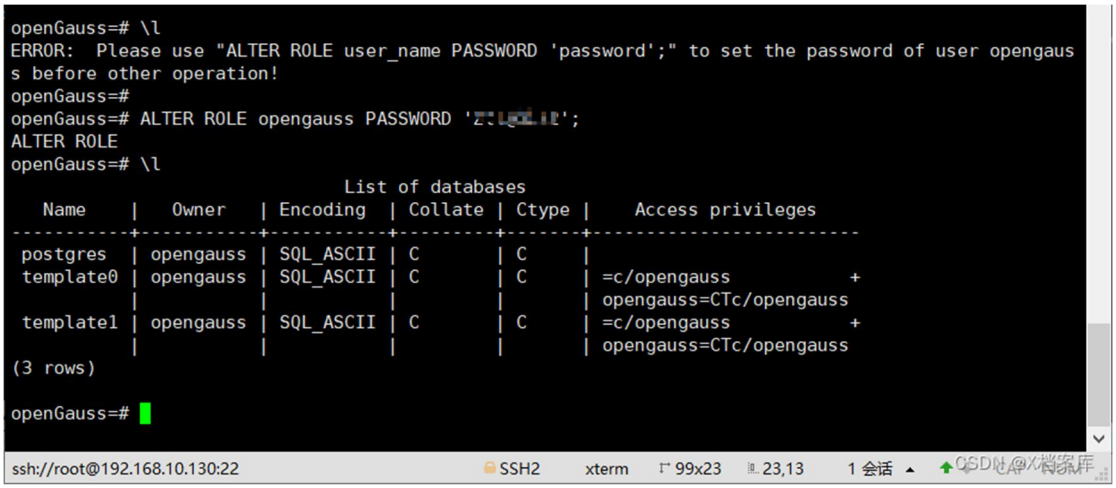


- 提示需要先修改 opengauss 账号密码，才能执行其他操作。




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

- 再次查看 database，显示如下即 opengauss 安装成功



- 退出数据库\退回 root 用户

```
Ctrl+D
```

- 创建日常操作账号

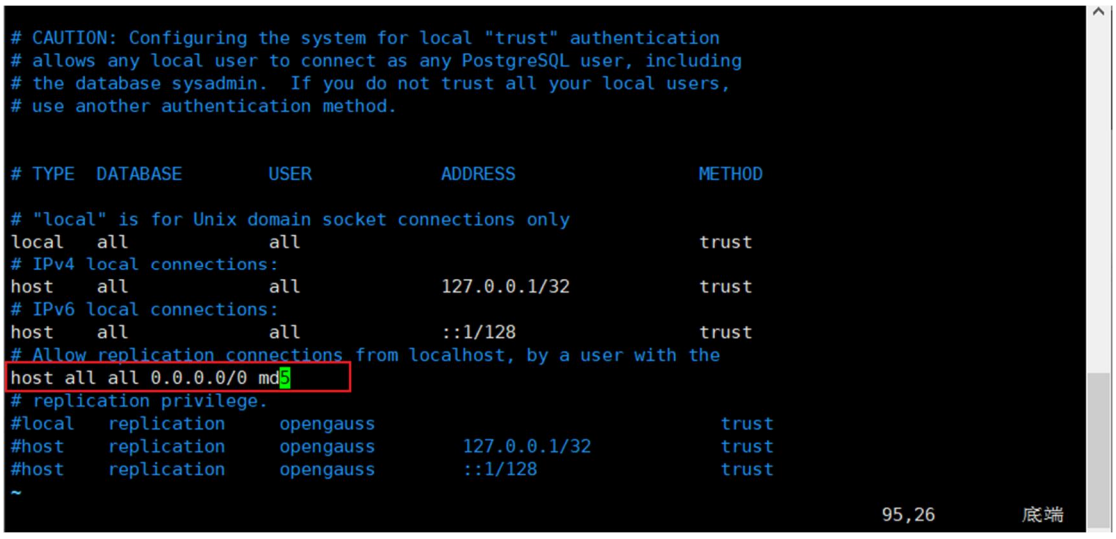
```
openGauss=# CREATE USER 账号 id  PASSWORD 'xxxxxx';
NOTICE:  The encrypted password contains MD5 ciphertext, which is not secure.
CREATE ROLE
openGauss=# alter user xlevon sysadmin;
ALTER ROLE
```

【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

```
[opengauss@localhost data]$ vim postgresql.conf
```

```
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）。

```
# CONNECTIONS AND AUTHENTICATION
#------------------------------------------------------------------------------

# - Connection Settings -

#listen_addresses = '127.0.0.1'
listen_addresses = '*'
                                # comma-separated list of addresses;
                                # defaults to 'localhost'; use '*' for all
                                # (change requires restart)
#local_bind_address = '0.0.0.0'
port = 7654
max_connections = 16                    # (change requires restart)
# Note:  Increasing max_connections costs ~400 bytes of shared memory per
# connection slot, plus lock space (see max_locks_per_transaction).
#sysadmin_reserved_connections = 3      # (change requires restart)
#unix_socket_directory = ''             # (change requires restart)
#unix_socket_group = ''                 # (change requires restart)
#unix_socket_permissions = 0700         # begin with 0 to use octal notation
                                # (change requires restart)

# - Security and Authentication -

#authentication_timeout = 1min          # 1s-600s
@@
-- 插入 --                                                  69,23         8%
```

```
# Kerberos and GSSAPI
#krb_server_keyfile = ''
#krb_srvname = 'postgres'               # (Kerberos only)
#krb_caseins_users = off

#modify_initial_password = false        #Whether to change the initial password of t
he initial user
#password_policy = 1                    #Whether password complexity checks
#password_reuse_time = 60               #Whether the new password can be reused in p
assword_reuse_time days
#password_reuse_max = 0                 #Whether the new password can be reused
#password_lock_time = 1                 #The account will be unlocked automatically
after a specified period of time
#failed_login_attempts = 10             #Enter the wrong password reached failed_log
in_attempts times, the current account will be locked
#password_encryption_type = 2           #Password storage type, 0 is md5 for PG, 1 i
s sha256 + md5, 2 is sha256 only
password_encryption_type = 0
#password_min_length = 8                #The minimal password length(6-999)
#password_max_length = 32               #The maximal password length(6-999)
#password_min_uppercase = 0             #The minimal upper character number in passw
ord(0-999)
#password_min_lowercase = 0             #The minimal lower character number in passw
ord(0-999)
@@
-- 插入 --                                                 110,29        12%
```

- 重启数据库

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

```
System information as of time:  2023年 07月 02日 星期日 15:48:18 CST

System load:    0.09
Processes:      163
Memory used:    15.0%
Swap used:      0%
Usage On:       9%
IP address:     192.168.10.130
Users online:   2
To run a command as administrator(user "root"),use "sudo <command>".
[opengauss@localhost ~]$ cd data/
[opengauss@localhost data]$ vim pg_hba.conf
[opengauss@localhost data]$
[opengauss@localhost data]$ vim postgresql.conf
[opengauss@localhost data]$
[opengauss@localhost data]$ gs_ctl stop
[2023-07-02 16:01:50.029][9763][][gs_ctl]: gs_ctl stopped ,datadir is /var/lib/openg
auss/data
waiting for server to shut down..... done
server stopped
[opengauss@localhost data]$ gs_ctl restart
[2023-07-02 16:02:05.600][9766][][gs_ctl]: gs_ctl restarted ,datadir is /var/lib/ope
ngauss/data
[2023-07-02 16:02:05.600][9766][][gs_ctl]: Is server running?
[2023-07-02 16:02:05.600][9766][][gs_ctl]: starting server anyway
```

```
2023-07-02 08:02:05.708 [unknown] [unknown] localhost 139783520712768 0[0:0#0]  0 [B
ACKEND] LOG:  the configure file /usr/local/opengauss/etc/gscgroup_opengauss.cfg doe
sn't exist or the size of configure file has changed. Please create it by root user!
2023-07-02 08:02:05.708 [unknown] [unknown] localhost 139783520712768 0[0:0#0]  0 [B
ACKEND] LOG:  Failed to parse cgroup config file.
2023-07-02 08:02:05.711 [unknown] [unknown] localhost 139783520712768 0[0:0#0]  0 [E
XECUTOR] WARNING:  Failed to obtain environment value $GAUSSLOG!
2023-07-02 08:02:05.711 [unknown] [unknown] localhost 139783520712768 0[0:0#0]  0 [E
XECUTOR] DETAIL:  N/A
2023-07-02 08:02:05.711 [unknown] [unknown] localhost 139783520712768 0[0:0#0]  0 [E
XECUTOR] CAUSE:  Incorrect environment value.
2023-07-02 08:02:05.711 [unknown] [unknown] localhost 139783520712768 0[0:0#0]  0 [E
XECUTOR] ACTION:  Please refer to backend log for more details.
2023-07-02 08:02:05.711 [unknown] [unknown] localhost 139783520712768 0[0:0#0]  0 [E
XECUTOR] WARNING:  Failed to obtain environment value $GAUSSLOG!
2023-07-02 08:02:05.711 [unknown] [unknown] localhost 139783520712768 0[0:0#0]  0 [E
XECUTOR] DETAIL:  N/A
2023-07-02 08:02:05.711 [unknown] [unknown] localhost 139783520712768 0[0:0#0]  0 [E
XECUTOR] CAUSE:  Incorrect environment value.
2023-07-02 08:02:05.711 [unknown] [unknown] localhost 139783520712768 0[0:0#0]  0 [E
XECUTOR] ACTION:  Please refer to backend log for more details.

[2023-07-02 16:02:06.609][9766][][gs_ctl]:  done
[2023-07-02 16:02:06.609][9766][][gs_ctl]: server started (/var/lib/opengauss/data)
[opengauss@localhost data]$
```

- 查询并开放服务器端口

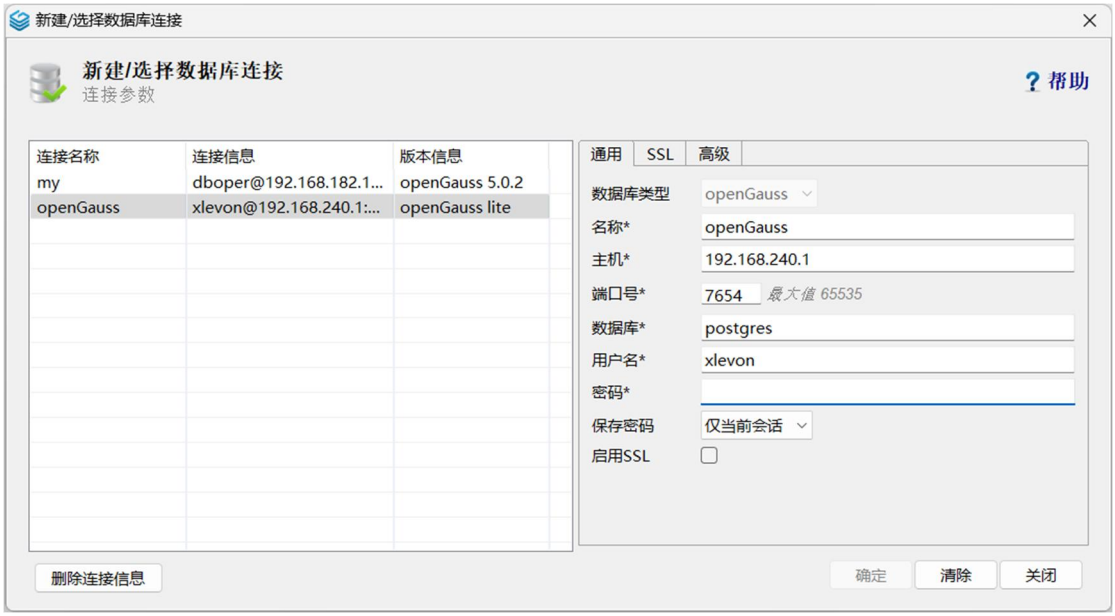
```
[root@localhost ~]# netstat -antp
[root@localhost ~]# sudo firewall-cmd --permanent --add-port=7654/tcp
[root@localhost ~]# sudo systemctl reload firewalld
```

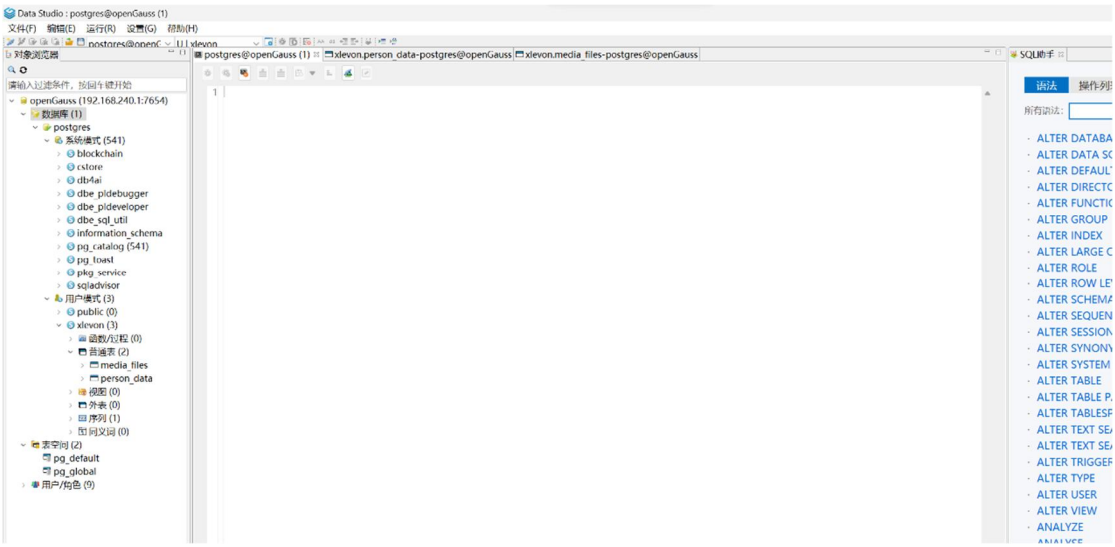
```
tcp6       0      0 ::1:55754               ::1:7654                TIME_WAIT   -

[root@localhost ~]# clear
[root@localhost ~]# netstat -antp
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:7654            0.0.0.0:*               LISTEN      9767/gaussdb
tcp        0      0 0.0.0.0:7655            0.0.0.0:*               LISTEN      9767/gaussdb
tcp        0      0 0.0.0.0:111             0.0.0.0:*               LISTEN      860/rpcbind
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      2685/sshd: /usr/sbi
tcp        0     36 192.168.10.130:22       192.168.10.1:58492      ESTABLISHED 9523/sshd: root [pr
tcp6       0      0 :::7654                 :::*                    LISTEN      9767/gaussdb
tcp6       0      0 :::7655                 :::*                    LISTEN      9767/gaussdb
tcp6       0      0 :::111                  :::*                    LISTEN      860/rpcbind
tcp6       0      0 :::22                   :::*                    LISTEN      2685/sshd: /usr/sbi
tcp6       0      0 ::1:55762               ::1:7654                TIME_WAIT   -
tcp6       0      0 ::1:55754               ::1:7654                TIME_WAIT   -
[root@localhost ~]# sudo firewall-cmd --permanent --add-port=7654/tcp
success
[root@localhost ~]# sudo systemctl reload firewalld
[root@localhost ~]#
```

2) 数据库访问

使用 Datastudio 访问 openGauss

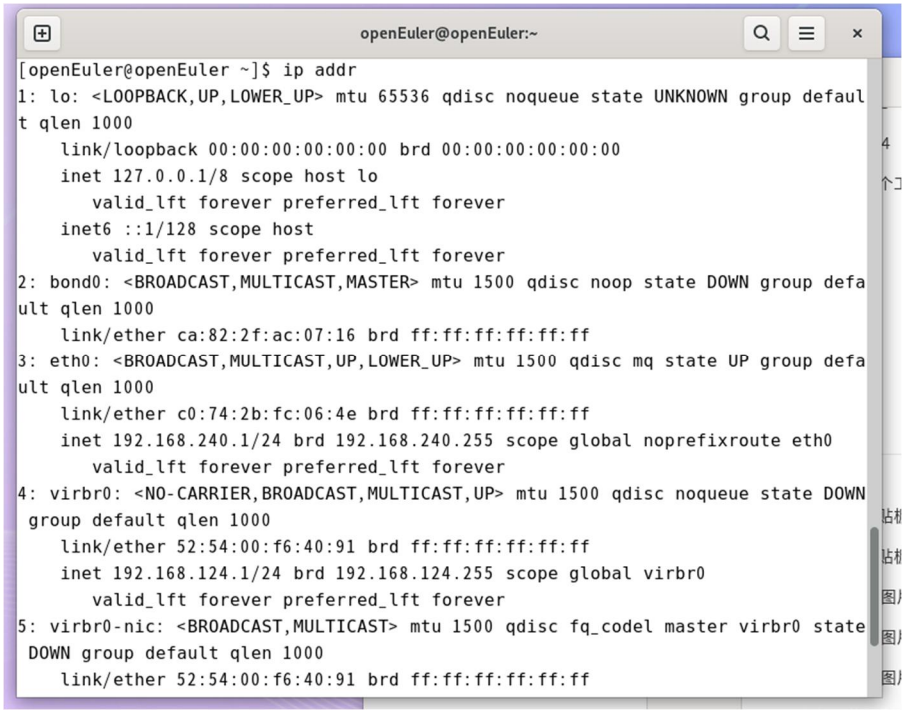




1.6 网络配置

1) 在香橙派的系统界面，先在系统设置中连接无线网络。（启动之后的系统时间可能是 1970 年，需要联网手动设置成当前时间，否则用 vscode 安装库的时候会因为时间校验问题出现报错）。
2) 通过网线实现 PC 和香橙派的连接。
3) 打开系统终端输入，界面如下

```
ip addr
```



记下 eth0 下的网络地址 192.168.240.1（这里我的主机号已经修改好了，每个主机的 ip 不同，自行记下）

再查看 PC 上的网络地址 192.168.240.2（每个主机的 ip 不同，自行记下）两台设备要在同一个局域网（子网）内才能互相通信。可以通过 linux 端的系统设置将 ip 地址调整成网络地址相同，主机号不同的地址。例如我将 linux 端的 ip 地址修改为了 192.168.240.1。

二、 代码设计

2.1 系统架构

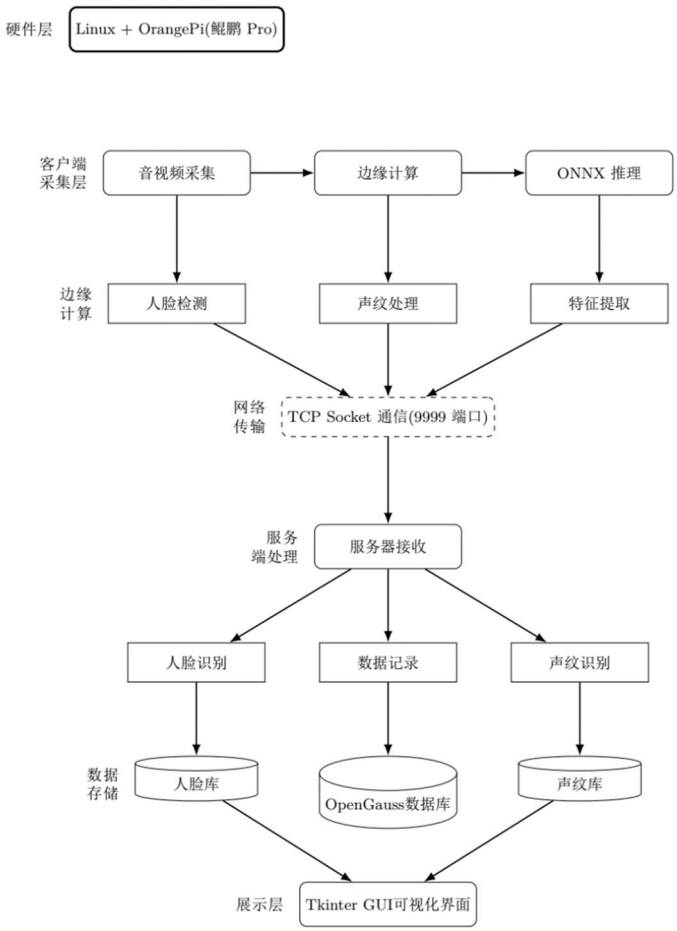
系统环境： linux 系统，OrangePi（香橙派鲲鹏 pro）

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（vscode）
- 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。



2.2 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。

2.3 设计实现

因为实验四是基于实验一的扩展，因此先完成实验一的代码实现。

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）。整体工程代码量很大，因此只贴出部分关键代码。

A. 服务器端

[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。

关键函数

write_video_frame

```
def write_video_frame(self, client_id, frame_data):
    """将视频帧写入视频文件"""
    # 如果这个客户端还没有 VideoWriter，创建一个
    if client_id not in self.video_writers:
        username = self.clients[client_id]['username']
        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
        filename = f"video_{username}_{client_id}_{timestamp}.mp4"
        filepath = os.path.join(self.storage_dir, "video", filename)

        # 创建 VideoWriter（800x600, 30fps）
        fourcc = cv2.VideoWriter_fourcc(*'mp4v')
        self.video_writers[client_id] = {
            'writer': cv2.VideoWriter(filepath, fourcc, 30.0, (800,
600)),
            'filepath': filepath,
            'username': username
        }
```

stop_video_recording

```
def stop_video_recording(self):
    """停止录制视频"""
    self.recording_video = False

    # 释放所有 VideoWriter 并保存到数据库
    for client_id, writer_info in list(self.video_writers.items()):
        writer_info['writer'].release()
        filepath = writer_info['filepath']
        username = writer_info['username']

        # 保存到数据库
        self.save_to_database(client_id, username, 'video', filepath)
...
```

[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。

关键函数

save_audio_data

```
def save_audio_data(self, client_id, audio_data):
    """将音频数据写入 WAV 文件"""
    # 如果这个客户端还没有音频文件，创建一个
    if client_id not in self.audio_files:
        username = self.clients[client_id]['username']
        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
        filename = f"audio_{username}_{client_id}_{timestamp}.wav"
        filepath = os.path.join(self.storage_dir, "audio", filename)

        # 创建 WAV 文件（16kHz, 单声道, 16bit）
        wf = wave.open(filepath, 'wb')
        wf.setnchannels(1)  # 单声道
        wf.setsampwidth(2)  # 16bit = 2 bytes
        wf.setframerate(16000)  # 16kHz
…
```

stop_audio_recording

```
def stop_audio_recording(self):
    """停止录制音频"""
    self.recording_audio = False

    # 关闭所有音频文件并保存到数据库
    for client_id, audio_info in list(self.audio_files.items()):
        audio_info['file'].close()
        filepath = audio_info['filepath']
        username = audio_info['username']

        # 保存到数据库
         self.save_to_database(client_id, username, 'audio',
filepath)
```

[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）

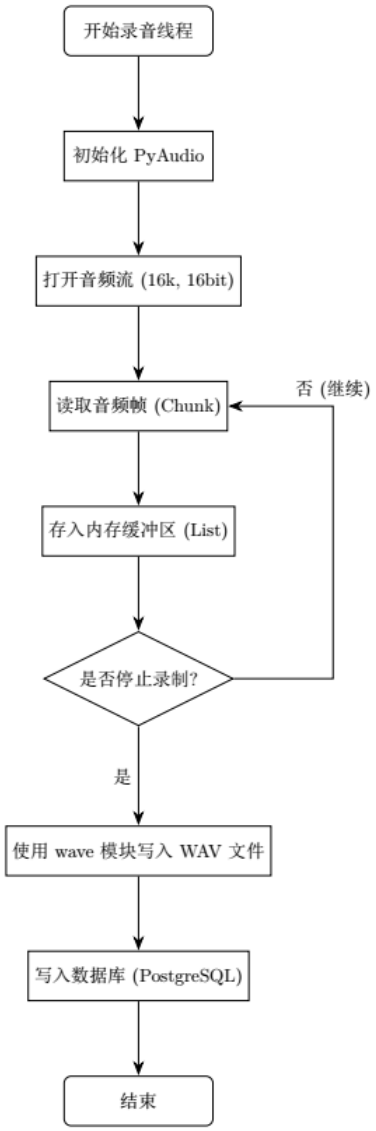
def save_to_database (self, client_id, username, media_type, filepath)

```
 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()
```

- 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。



B. 客户端

识别模型选型：

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。

[1] 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 ：

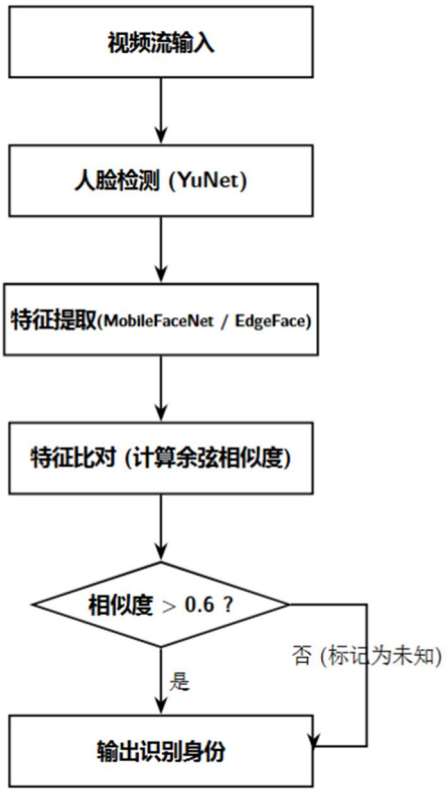
- MobileFaceNet：超轻量级网络，计算量小，适合实时视频流处理。
- EdgeFace：专为边缘计算优化的模型 (Gamma 版本)，在低功耗下提供更高精度。

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。



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
```

[2] 客户端声纹识别模块：

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：

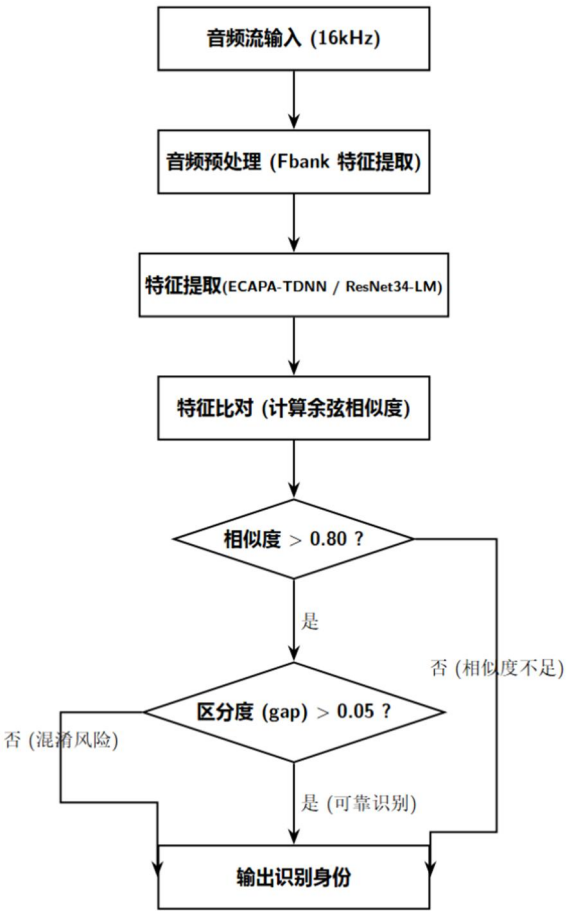
- ECAPA-TDNN：基于通道注意力的时延神经网络，擅长捕捉长距离上下文信息。
- ResNet34-LM：经典的残差网络，特征提取稳定，抗噪能力强。

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）



关键函数

def register_voice(self, name, audio_data)注册声纹

```
def register_voice(self, name, audio_data):
    """注册新声纹"""
    if self.session:
        # 有模型时提取特征
        features = self.extract_features(audio_data)
        self.known_voices[name] = features
        print(f"[INFO]  已注册声纹（特征向量）: {name}")
    else:
        # 无模型时保存原始音频
        self.known_voices[name] = audio_data
        print(f"[INFO]  已注册声纹（原始音频）: {name}")
```

def recognize_voice(self, audio_data, threshold)识别声纹

```
def recognize_voice(self, audio_data, threshold=0.30):
    """识别声纹（阈值 0.50，防止误匹配）"""
    features = self.extract_features(audio_data)

    # 计算与所有已知声纹的相似度
    for name, known_features in self.known_voices.items():
        similarity = np.dot(features, known_features)  # 余弦相似度
        if similarity > best_similarity:
            best_similarity = similarity
            best_match = name

    # 严格判定：需要同时满足相似度和区分度
    if best_match and best_similarity > threshold:
        return best_match, best_similarity
```

def extract_features(self, audio_data)提取声纹特征

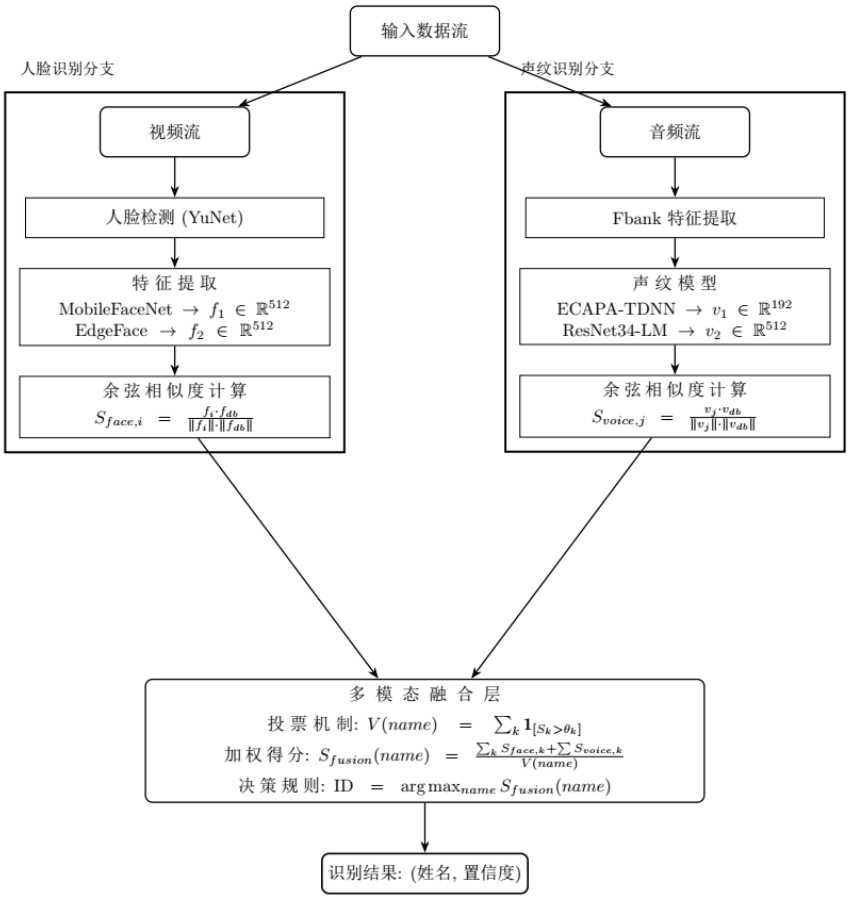
```
def extract_features(self, audio_data):
    """提取声纹特征向量"""
    # 预处理音频数据（计算 Fbank 特征）
    preprocessed = self.preprocess_audio(audio_data)

    # 使用 ONNX 模型提取特征
    features = self.session.run([self.output_name],
                                {self.input_name: preprocessed})[0]

    # L2 归一化特征向量
    features = features.flatten()
    features = features / np.linalg.norm(features)
    return features
```

[3] 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，$S_{fusion}(name)$}

关键函数

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

```
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
```

人脸模型融合（MobileFaceNet）

```
# 人脸识别 - MobileFaceNet
if self.fusion_face1_var.get() and self.face_recognition_enabled:
    # 切换到 MobileFaceNet 模型
    self.face_recognizer.switch_model('mobilefacenet')
    face_img = self.current_frame[y:y+h, x:x+w]
    name, similarity = self.face_recognizer.recognize_face(face_img)

    # 加权累积得分
    if name not in fusion_scores:
        fusion_scores[name] = 0.0
    fusion_scores[name] += similarity * model_weight
    enabled_modals.append('MobileFaceNet')
```

声纹模型融合（ECAPA-TDNN）

```
# 声纹识别 - ECAPA-TDNN
if self.fusion_voice1_var.get() and self.voice_recognition_enabled:
```

```
    audio_data = self.get_voice_buffer()
    if self.voice_recognizer_ecapa.is_available():
        name, similarity =
self.voice_recognizer_ecapa.recognize_voice(audio_data)

        # 加权累积得分
        if name not in fusion_scores:
            fusion_scores[name] = 0.0
        fusion_scores[name] += similarity * model_weight
        enabled_modals.append('ECAPA-TDNN')
```

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}")
```

C. 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。

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，对应 SQL 里的 BYTEA 类型

- 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

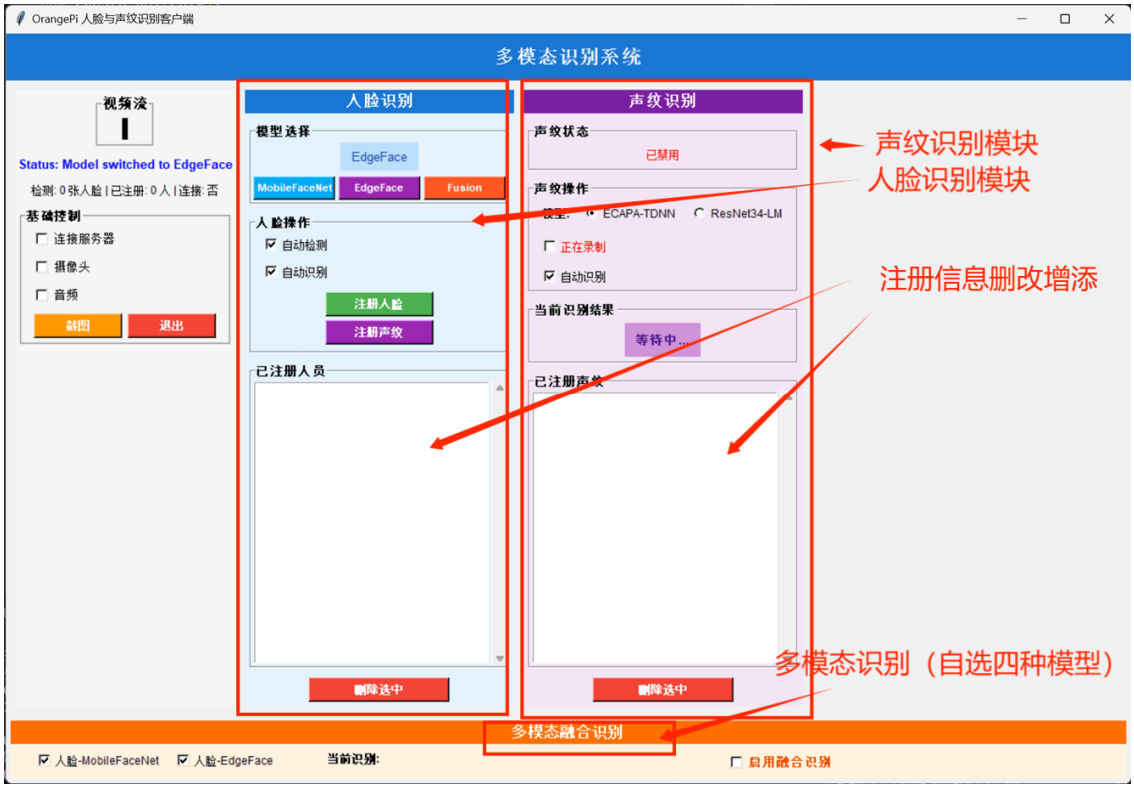
启动时会调用 client_linux.py 的 load_data_from_database：

```
try:
            cursor = self.db_conn.cursor()
            cursor.execute("SELECT name, face_image, voice_audio FROM person_data")
            rows = cursor.fetchall()
            cursor.close()

            loaded_faces = 0
            loaded_voices = 0
```

三、 系统测试

1) 系统界面展示



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
```

成功保存到本地

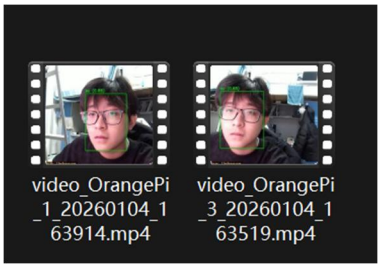
b) 登录数据库查看 media 表

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

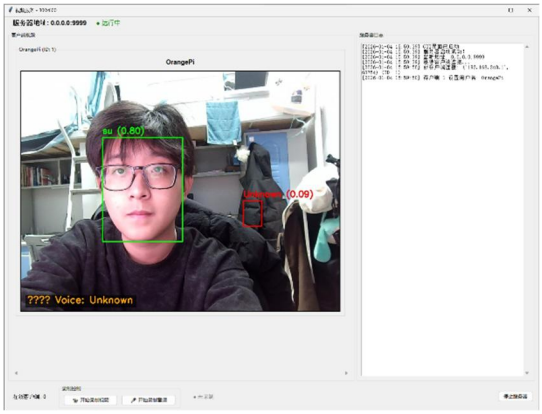
c) 查看本地文件（文件无损，播放正常）

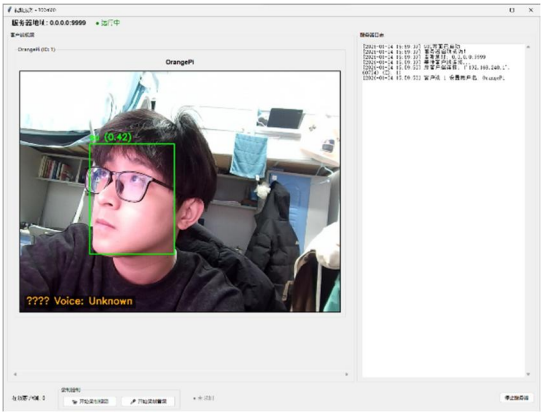


(a) video (b) audio 文件

4) 客户端功能验证

a) Mobilefacenet 人脸识别（正脸+侧脸双重测试）



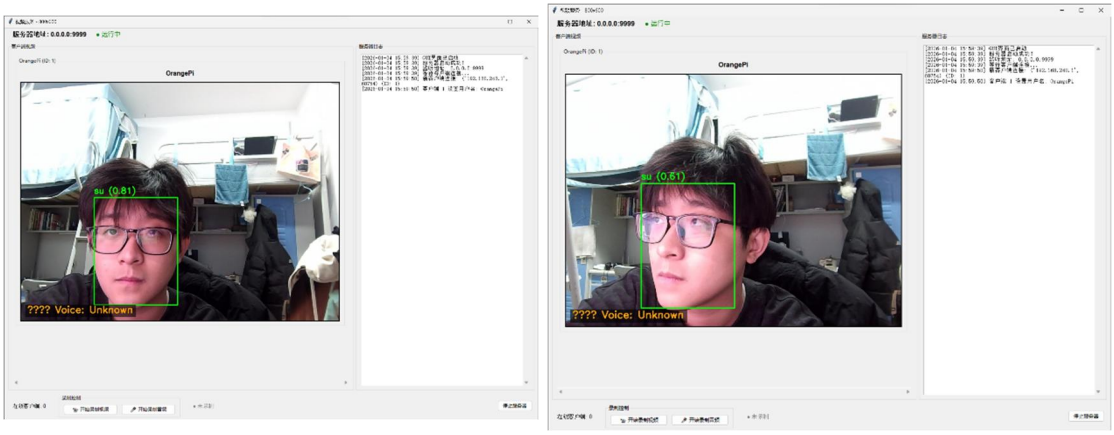


（a） （b）

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。

a) 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

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……

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：

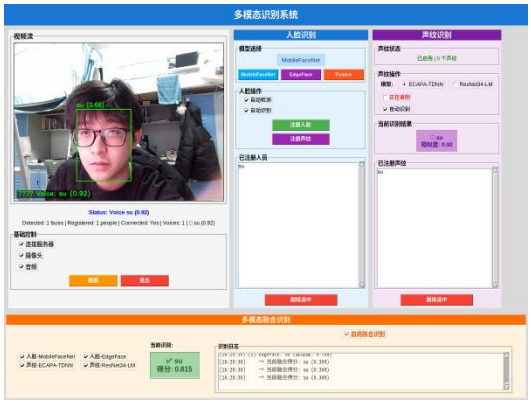
ECAPA-TDNN：平均识别成功率 74.2%，平均置信度 0.70

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。

b) 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）

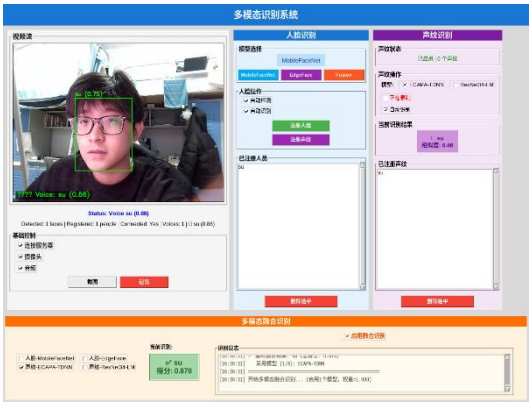
采用自适应的权重，采用的模型可以自由选择，最后给出多模态融合决策。



(a) 4 种模型

(b) 3 种模型

(c) 两种模型



(d)单模型

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%