

# 派蒙语音助手

## 设计目标

如今，诸如“天猫精灵”，“小爱同学”，“小度”等智能音箱对于大多数人来说应该不陌生。2022年底，由OpenAI研发的聊天机器人ChatGPT，短短5天，注册用户数就超过100万。ChatGPT能够通过理解和学习人类的语言来进行对话，还能根据聊天的上下文进行互动，真正像人类一样来聊天交流。因为其易用性，以及生成文本的高质量，ChatGPT也成为了大家日常功能和学习中的好助手，甚至可以平替百度。就我和我的同学而言，ChatGPT在日常的文书类任务的撰写，代码的debug以至于情感交流都提供了很大的帮助。

如果能把ChatGPT接入语音助手就好了！

春学期开始不久的3月2日，OpenAI开放了官方的API，允许第三方开发者通过API将ChatGPT集成到他们的应用程序和服务中。在语音助手中使用ChatGPT来生成对话的梦想变得“轻而易举”（只要解决翻墙的问题，然后参照官方文档调用API就可以）。综上，把ChatGPT接入语音助手的任务就解决了。

此外，因为我是一个重度的原神（一款开放世界游戏）玩家，并且非常喜欢其中的派蒙（派蒙是游戏中的NPC之一），所以，我希望用派蒙的音效来进行最终的语音合成，并且可以集成一些有利于游戏玩家的功能。所以我集成和实现的功能如下：

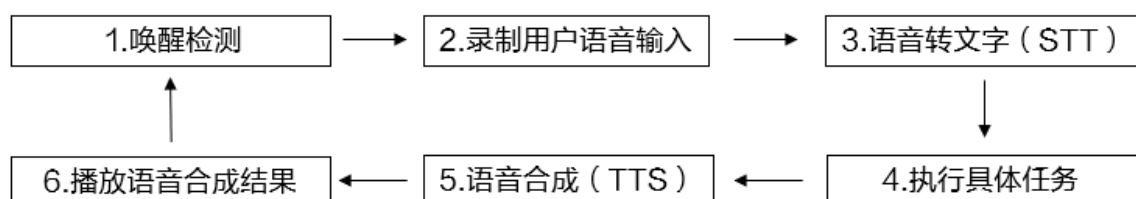
- 天气查询
- 游戏一键签到
- 游戏社区热帖查看
- 派蒙语音播报

整个项目已在[github](#)上开源。

整活视频链接为[https://youtu.be/PoGvY\\_eOYAc](https://youtu.be/PoGvY_eOYAc)。

## 设计过程

### 系统框架



以上是我整个系统的主体框架：

整个系统首先需要通过预先设置的唤醒词来进行唤醒整个语音交互的功能。

检测到唤醒词后，通过USB麦克风录制用户的输入。然后调用百度api实现语音转文字。

获得返回的文字后，我通过关键词检测来进行判断，决定这轮是否执行语音助手集成的具体的任务。如果都不是的话，则调用接口将任务交给ChatGPT。

然后，我们将需要播报的内容通过我在WSL上部署的语音合成服务进行派蒙音效的语音合成。

最后，我将合成的音频文件传回树莓派，并通过蓝牙音箱播放。

## 环境及外设

树莓派信号：树莓派 4b4g

操作系统：Ubuntu 22.04

Python版本：Python 3.10

音频输入设备：USB麦克风 [淘宝链接](#)

音频输出设备：蓝牙音箱 [淘宝链接](#)

显示屏：7寸IPS高清触摸屏 [淘宝链接](#)

### 外设图片



USB麦克风



蓝牙音箱



7寸IPS高清触摸屏




## 唤醒检测

在唤醒检测上，不同于大多数同学直接通过百度api语音转文字并进行判断，我使用了snowboy进行离线的唤醒检测。

但实现离线的唤醒检测的功能，并不是直接clone仓库并运行这么轻松。

## 唤醒词模型的训练

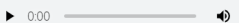
网上可以搜到的教程都直接通过snowboy官网来训练模型，或者直接下载别人训练好的模型，但是，**snowboy官网在2020.03.18日后就已经停止维护**。通过访问官网来训练模型显然不现实。好在有大佬自己又搭了一个训练的web端<https://snowboy.hahack.com/>，通过电脑麦克风录制三条语言，即可生成适用于snowboy的唤醒词模型。

 Record

Model Name:

Examples (3 required)

1.  

2.  

3.  

 Submit

通过以上步骤，我得到了"paimeng.pmdl"文件，用于后续的唤醒模块。

## 编译动态库

clone仓库后，snowboy并不能直接运行示例脚本“开箱即用”。需要基于操作系统编译对应语言的动态库。

首先，需要安装对应的依赖

```
sudo apt install python3-pyaudio
sudo apt install swig
sudo apt install libatlas-base-dev
```

❗ 安装依赖库后，直接make仍然会报错：

```
make: python3-config: Command not found
make: python3-config: Command not found
swig -I../.. -c++ -python -o snowboy-detect-swig.cc snowboy-detect-swig.i
g++ -I../.. -O3 -fPIC -D_GLIBCXX_USE_CXX11_ABI=0 -std=c++0x -c snowboy-detect-swig.cc
snowboy-detect-swig.cc:181:11: fatal error: Python.h: No such file or directory
  181 | # include <Python.h>
      |           ^~~~~~
compilation terminated.
make: *** [Makefile:70: snowboy-detect-swig.o] Error 1
```

这是因为Linux 发行版通常会把类库的头文件和相关的 pkg-config 分拆成一个单独的 xxx-dev。以python为例，python-dev 包括了一些用 C / Java / C# 等编写的 Python 扩展，在编译的时候依赖的头文件等信息。在编译一个用 C 语言编写的 Python 扩展模块时，因此需要先安装 python-dev开发包。

```
sudo apt install python3-dev
```

之后再make，问题解决，生成了\_snowboydetect.so文件。在这之后，我们就可以运行官方提供的脚本进行测试。以下为本项目中，基于snowboy语音唤醒模块的测试代码。

```
import snowboydecoder
import sys
import signal
```

```

interrupted = False

def signal_handler(signal, frame):
    global interrupted
    interrupted = True

# 检查是否有中断
def interrupt_callback():
    global interrupted
    return interrupted

# 模型路径
model = "resources/models/paimeng.pmdl"

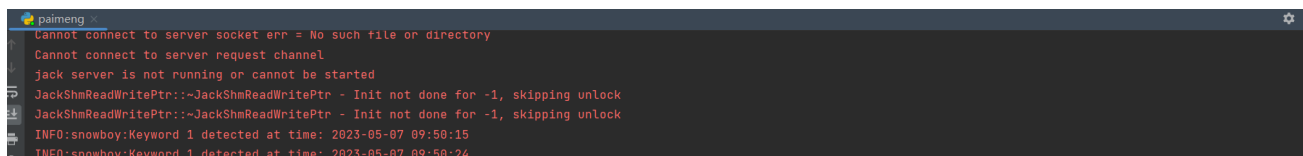
# 捕获SIGINT信号
signal.signal(signal.SIGINT, signal_handler)

# 如果有语音输入，调用snowboy的HotwordDetector函数进行检测
detector = snowboydecoder.HotwordDetector(model, sensitivity=0.5)
print('Listening... Press Ctrl+C to exit')

# 主循环
detector.start(detected_callback=snowboydecoder.play_audio_file,
               interrupt_check=interrupt_callback,
               sleep_time=0.03)

detector.terminate()

```



```

paimeng
Cannot connect to server socket err = No such file or directory
Cannot connect to server request channel
jack server is not running or cannot be started
JackShmReadWritePtr::~JackShmReadWritePtr - Init not done for -1, skipping unlock
JackShmReadWritePtr::~JackShmReadWritePtr - Init not done for -1, skipping unlock
INFO:snowboy:Keyword 1 detected at time: 2023-05-07 09:50:15
INFO:snowboy:Keyword 1 detected at time: 2023-05-07 09:50:24

```

唤醒词检测模块工作正常。

## 用户语音输入

为了录制用户的语音，我使用了pyaudio这个库。本模块具体的工作流程为：

- 打开一个音频流
- 等待用户开始说话
- 录制语音并检测是否结束
- 保存音频文件

首先，我尝试了固定时长的语音录制，具体代码如下（在本测试文件中，我设定录制时长为固定的5秒 RECORD\_SECONDS = 5）：

```

import pyaudio
import wave

CHUNK = 1024
FORMAT = pyaudio.paInt16
CHANNELS = 1

```

```

RATE = 16000
RECORD_SECONDS = 5
WAVE_OUTPUT_FILENAME = "output.wav"

# 创建 pyaudio 对象
p = pyaudio.PyAudio()

# 打开一个音频流
stream = p.open(format=FORMAT,
                 channels=CHANNELS,
                 rate=RATE,
                 input=True,
                 frames_per_buffer=CHUNK)

print("开始录音...")

# 从音频流中读取数据
frames = []
for i in range(0, int(RATE / CHUNK * RECORD_SECONDS)):
    data = stream.read(CHUNK)
    frames.append(data)

print("录音结束! ")

# 将录制的音频保存为 wav 文件
wf = wave.open(WAVE_OUTPUT_FILENAME, 'wb')
wf.setnchannels(CHANNELS)
wf.setsampwidth(p.get_sample_size(FORMAT))
wf.setframerate(RATE)
wf.writeframes(b''.join(frames))
wf.close()

# 关闭音频流和 pyaudio 对象
stream.stop_stream()
stream.close()
p.terminate()

```

但显然，采用固定时长并不够灵活，所以我对上述代码进行了改进，以音频强度为参照，进行静音检测以及用户输入检测的判断。其中，THRESHOLD就是一个超参数，是用来区分是否用户在说话的阈值。silent\_counter则是用来统计，开始录制后，用户停顿的时间。这是为了提供一个裕度，因为很显然，我们平常说话的时候也不是一字不停，一口气说到底。在说话时，会有短暂的停顿。改进后，代码如下：

```

import pyaudio
import wave
import time
import struct

CHUNK = 1024
FORMAT = pyaudio.paInt16
CHANNELS = 1
RATE = 16000
WAVE_OUTPUT_FILENAME = "output.wav"

THRESHOLD = 300

# 创建 pyaudio 对象
p = pyaudio.PyAudio()

```

```

# 打开一个音频流
stream = p.open(format=FORMAT,
                 channels=CHANNELS,
                 rate=RATE,
                 input=True,
                 frames_per_buffer=CHUNK)

print("等待用户开始说话...")

# 等待用户开始说话
while True:
    data = stream.read(CHUNK)

    max_value = 0
    data_int = struct.unpack(f'{CHUNK}h', data)
    max_value = max(data_int)
    # print(max_value) # debug用

    # 如果声音够大, 说明用户开始输入, 结束等待状态, 开始录制
    if max_value > THRESHOLD:
        break

print("开始录音...")

# 开始录音
frames = [data]
silent_counter = 0
while True:
    data = stream.read(CHUNK)
    frames.append(data)

    data_int = struct.unpack(f'{CHUNK}h', data)
    max_value = max(data_int)

    # 如果此时音量较小, 则silent_counter + 1
    # 但如果超过阈值的音量, 那么说明用户停顿后又继续开始说话, 那么需要将silent_counter清零
    silent_counter = silent_counter + 1 if max_value < THRESHOLD else 0
    # 此处设置为, 超过三秒, 视作用户输入结束
    if silent_counter > RATE // CHUNK * 3:
        break

print("录音结束! ")

# 将录制的音频保存为 wav 文件
wf = wave.open(WAVE_OUTPUT_FILENAME, 'wb')
wf.setnchannels(CHANNELS)
wf.setsampwidth(p.get_sample_size(FORMAT))
wf.setframerate(RATE)
wf.writeframes(b''.join(frames))
wf.close()

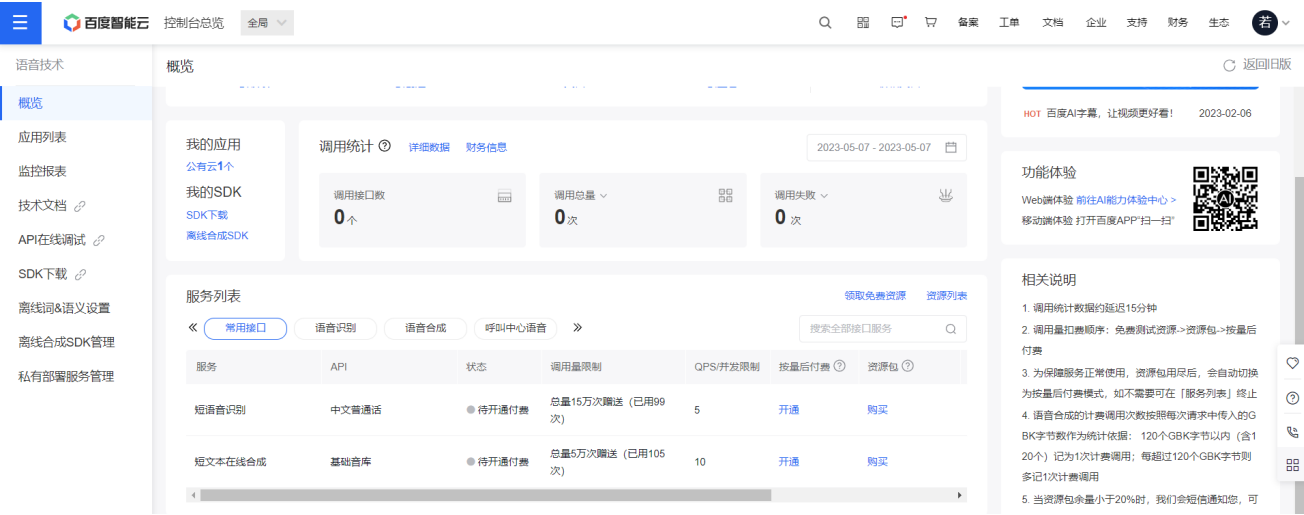
# 关闭音频流和 pyaudio 对象
stream.stop_stream()
stream.close()
p.terminate()

```

通过检查保存的音频文件output.wav，可以判断本模块正常工作。

## 语音转文字 (STT)

在本模块，我使用了百度的api。注册账号后，可以领取免费资源。有15万次语音识别，和5万次语音合成。在本项目的调试过程中，我才用了一百条左右，所以不需要付费，用官方赠送的免费资源也是戳戳有余的。



阅读文档可知，我们可以通过修改pid参数，指定具体的语音识别任务的语言：

1537	普通话(纯中文识别)	语音近场识别模型	有标点	支持自定义词库
1737	英语		无标点	不支持自定义词库
1637	粤语		有标点	不支持自定义词库
1837	四川话		有标点	不支持自定义词库
1936	普通话远场	远场模型	有标点	不支持

本模块的测试代码如下（出于隐私考虑，我把我的账号信息和api key隐去了）：

```
from aip import AipSpeech

# 替换为您的百度 API 密钥
BAIDU_APP_ID = 'xxx'
BAIDU_API_KEY = 'xxx'
BAIDU_SECRET_KEY = 'xxx'

# 创建一个 AipSpeech 对象
client = AipSpeech(BAIDU_APP_ID, BAIDU_API_KEY, BAIDU_SECRET_KEY)

def recognize_wav_file(filename):
    with open(filename, 'rb') as file:
        audio_data = file.read()

    # 设置dev_pid为1537，即普通话(纯中文识别)
```

```

response = client.asr(audio_data, 'wav', 16000, {'dev_pid': 1537})

if response['err_no'] == 0:
    text = response['result'][0]
    return text
else:
    print(f"识别错误: {response['err_no']} - {response['err_msg']}")
    return None

# 设置保存的 WAV 文件路径
wav_file_path = "output.wav"

recognized_text = recognize_wav_file(wav_file_path)

if recognized_text:
    print(f"识别结果: {recognized_text}")
else:
    print("识别失败")

```

## 执行具体任务

在得到语音识别的结果后，我通过判断分支的语句，让语音助手执行不同的任务。分支的判断比较简单粗暴，就是通过关键词检测

```

# 判断是否为特殊任务
if text == "一键签到。":
    response_text = u_tasks.signYS()
elif text == "我要看帖子。":
    response_text = u_tasks.getYSArticle()
elif u_tasks.is_query_weather(text):
    response_text = u_tasks.query_weather(text)
else:
    # 若都不是，则调用chatgpt的api生成回复
    response_text = generate_response(text)

```

## 天气查询

天气查询功能，我使用了知心天气的api。心知天气免费版访问频限20次/分钟但无限访问量，无过期时间。具体代码如下（出于隐私考虑，我把我的账号信息和api key隐去了）：

在判断用户的输入是否为天气查询的任务时，我采用了关键词检测的方式。先通过jieba这个库对查询语句进行分词，然后检测是否存在'天气'、'气温'、'温度'等关键词。这里需要注意的是：jieba分词有三种不同的分词模式：**精确模式**、**全模式**和**搜索引擎模式**。

```

jieba.lcut(sentence, cut_all=False) # 精确模式
jieba.lcut(sentence, cut_all=True) # 全模式
jieba.lcut_for_search(sentence) # 搜索引擎模式

```

对于“今天天气真好”，三者划分结果如下：



```
'今天天气 真好' # 精确模式
'今天 今天天气 天天 天气 真好' # 全模式
'今天 天天 天气 今天天气 真好' # 搜索引擎模式
```

其中，精确模式会将“今天天气”合并在一起，导致我简单粗暴的关键词判断方法失效，而后两种模式都可以划分出“天气”，在实际过程中，我使用了搜索引擎模式。

在查询天气时我先通过jieba这个库对查询语句进行分词，并找出其中的地名。如果没有，则默认是杭州。然后通过requests库发起请求，最后对返回的json文件进行解析。

```
def is_query_weather(text):
    # if(text=="查询天气")
    # return True
    if text == "查询天气":
        return True

    # 定义关键词列表
    keywords = ['天气', '气温', '温度']
    words = jieba.lcut_for_search(text)
    print(words)
    # 检查是否包含关键词
    if any(word in keywords for word in words):
        return True
    else:
        return False

def query_weather(text):
    words = pseg.cut(text)
    city = "杭州"
    for word, flag in words:
        if flag == 'ns': # ns 表示地名
            city = word
            break

    params = {
        # "key": "你的API密钥",
        "key": "xxx",
        "location": city, # 设置查询地点
        "language": "zh-Hans",
        "unit": "c",
    }
    url = "https://api.seniverse.com/v3/weather/now.json"
    r = requests.get(url, params=params)
    # 解析数据
    data = r.json()["results"]

    ret = ""
    ret += data[0]["location"]["name"]
    ret += "的天气是"
    ret += data[0]["now"]["text"]

    print(ret)
    return ret
```

```
test03-pyaudio.py
test04-paimeng.py
test05-crawler.py
test06-playground.py
test07-riandao.py
u_tasks
107     text = "北京天气怎么样?"
108     print(is_query_weather("北京天气怎么样?"))
109     query_weather(text)

if __name__ == "__main__":

/usr/bin/python3 /mnt/d/Desktop/embeddedProj/u_tasks.py
[test]
Building prefix dict from the default dictionary ...
2023-05-07T10:49:39 DEBUG Building prefix dict from the default dictionary ...
Loading model from cache /tmp/jieba.cache
2023-05-07T10:49:39 DEBUG Loading model from cache /tmp/jieba.cache
['北京', '天气', '怎么', '怎么样', '?']
True
Loading model cost 0.465 seconds.
2023-05-07T10:49:39 DEBUG Loading model cost 0.465 seconds.
Prefix dict has been built successfully.
2023-05-07T10:49:39 DEBUG Prefix dict has been built successfully.
[{'location': {'id': 'WX4FBXXFKE4F', 'name': '北京', 'country': 'CN', 'path': '北京,北京,中国', 'timezone': 'Asia/Shanghai', 'timezone_offset': '+08:00'}, 'now': {'text': '晴', 'code':
```

可以发现，心知天气虽然免费，但是返回的结果比较简单'now': {'text': '晴', 'code': '0', 'temperature': '22'}。后来，受到老师上课通过杭州气象官网爬取天气数据的启发，我将对于杭州本地的天气的查询，通过杭州气象官方进行爬取。而不是统一的调用心知天气的api。

最初，我尝试直接通过在网站源码中定位，获得数据，核心代码如下：

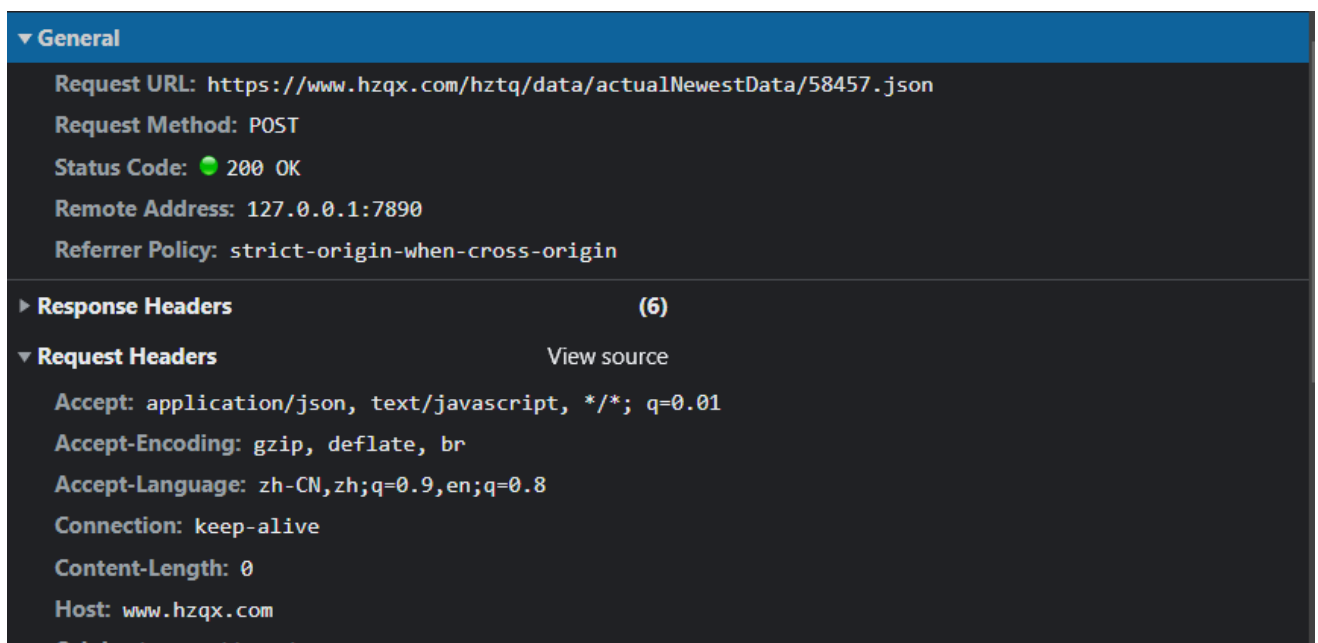
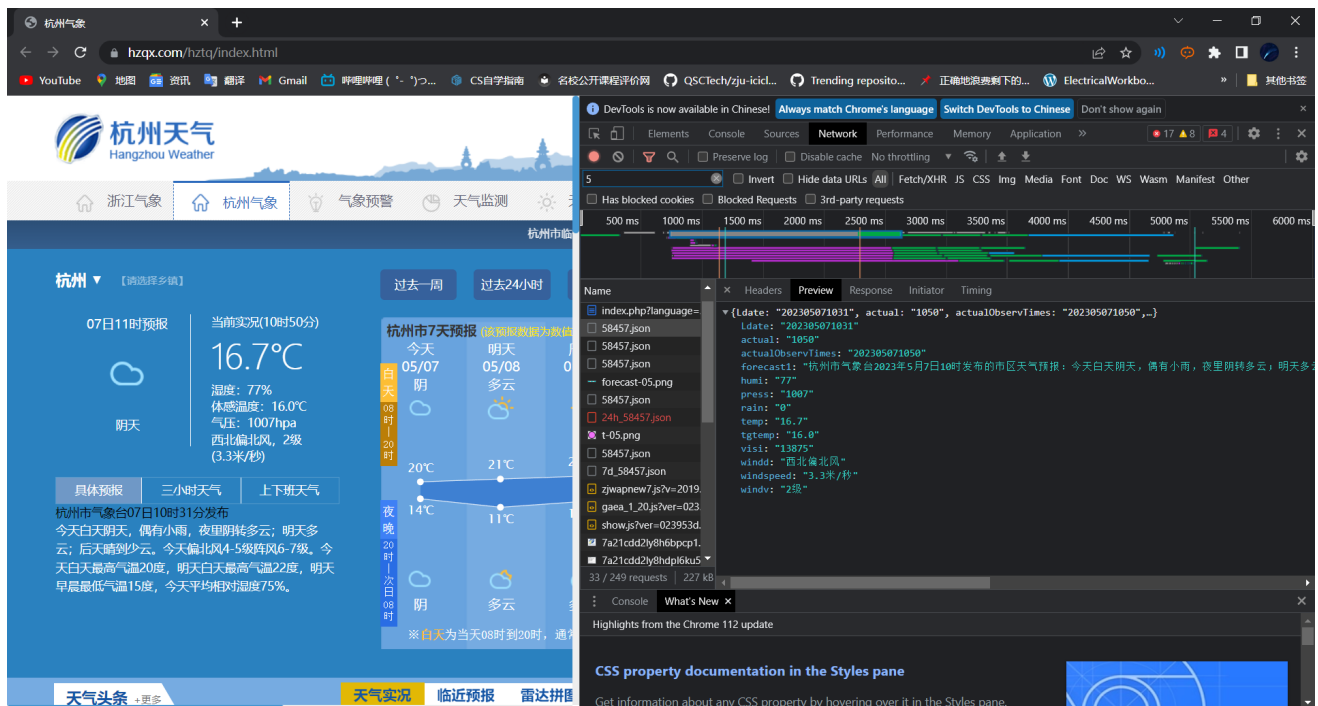
```
response = requests.get(url, headers=headers)
content_type = response.headers.get('Content-Type')

if response.status_code == 200:
    soup = BeautifulSoup(response.content, 'html.parser', from_encoding='utf-8')
```

但是以失败告终，因为requests获取网站源码，只能获得静态的html文件。其中，与“具体预报”相关的部分，是通过js脚本获取的。这部分代码在静态html中为，<li id="forecastDetail"></li>这部分，显然这在静态的html中是不存在的，该部分内容是由JavaScript动态加载的：

```
<div class="weather_tab">
  <ul class="tab">
    <li class="current" style="border-right:1px solid #FFF; width:30%">
      具体预报
    </li>
    <li style="border-right:1px solid #FFF; width:35%">
      三小时天气
    </li>
    <li style="width:34%">
      上下班天气
    </li>
  </ul>
  <div class="weather0_con">
    <ul class="block" >
      <li id="forecastDetail"></li>
    </ul>
    <ul>
      <li id="forecast3Detail"></li>
    </ul>
    <ul>
      <li id="sxbDetail"></li>
    </ul>
  </div>
</div>
```

所以，只能转变思路，通过chrome的开发者工具的network选项卡，找到具体预报请求的目标地址，然后通过requests库发起请求以获得内容。



特别的，务必需要写请求头，否则服务器会拒绝访问，并返回403。具体代码如下：

```
import requests

url = 'https://www.hzqx.com/hztq/data/actualNewestData/58457.json'

headers = {
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/93.0.4577.63 Safari/537.36",
    "Accept":
    "text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9",
    "Content-type": "application/x-www-form-urlencoded",
    "Accept-Language": "zh-CN,zh;q=0.9",
    "Accept-Encoding": "gzip, deflate, br",
    "Cookie": ""
}

response = requests.get(url, headers=headers)
```

```

if response.status_code == 200:
    content_json = response.json()
    print(content_json)
    print(content_json['forecast1'])

    text = content_json['forecast1']
    text = text.split(": ", 1)
    text = text[1]
    text = text.replace('%', '')
    print(text)
else:
    print(f'请求失败, 状态码: {response.status_code}')

```

```

/usr/bin/python3 /mnt/d/Desktop/embeddedProj/test09-3-hangzhoutq.py
{'ldate': '202305071031', 'actual': '1055', 'actualObservTimes': '202305071055', 'forecast1': '杭州市气象台2023年5月7日10时发布的市区天气预报: 今天白天阴天, 偶有小雨, 夜里阴转多云; 明天多云; 后天晴。杭州市气象台2023年5月7日10时发布的市区天气预报: 今天白天阴天, 偶有小雨, 夜里阴转多云; 明天多云; 后天晴到少云。今天偏北风4-5级阵风6-7级。今天白天最高气温20度, 明天白天最高气温22度, 明天早晨最低气温15度, 今天平均相对湿度75。今天白天阴天, 偶有小雨, 夜里阴转多云; 明天多云; 后天晴到少云。今天偏北风4-5级阵风6-7级。今天白天最高气温20度, 明天白天最高气温22度, 明天早晨最低气温15度, 今天平均相对湿度75。'}

```

这样，我们就获得了更加具体精确的杭州的天气。

## 一键签到&查看帖子

也是同样的使用了requests库实现爬虫，但是为了实现签到和帖子的查看，需要先模拟登录。此部分，我参考了GitHub上开源的游戏助手工具“米游币”：

首先，向["https://webapi.account.mihoyo.com/Api/cookie\\_accountinfo\\_by\\_loginticket?login\\_ticket={}"](https://webapi.account.mihoyo.com/Api/cookie_accountinfo_by_loginticket?login_ticket={})发起请求拿到stuid

然后向[https://api-takumi.mihoyo.com/auth/api/getMultiTokenByLoginTicket?login\\_ticket={}&token\\_types=3&uid={}发](https://api-takumi.mihoyo.com/auth/api/getMultiTokenByLoginTicket?login_ticket={}&token_types=3&uid={})起请求，拿到stoken

参考项目“米游币”的代码将整个模拟登录的过程封装为一个类，在类的初始化的时候调用以下与登录相关的函数拿到Cookie值：

```

# md5加密
def md5(text):
    md5 = hashlib.md5()
    md5.update(text.encode())
    return md5.hexdigest()

# 生成指定位数随机字符串
def randomStr(n):
    return (''.join(random.sample(string.ascii_lowercase, n))).upper()

# 生成DS
def DSGet():
    n = salt
    i = str(int(time.time()))
    r = randomStr(6)
    c = md5("salt=" + n + "&t=" + i + "&r=" + r)
    return "{},{},{}".format(i, r, c)

def DSGet2(q: str="", b: str="") -> str:
    n = salt2
    i = str(int(time.time()))
    r = str(random.randint(100001, 200000))
    add = f'&b={b}&q={q}'

```

```

c = md5("salt=" + n + "&t=" + i + "&r=" + r + add)
return f"{i},{r},{c}"

# 获取Cookie
def getCookie(cookie):
    Cookie = {}
    if "login_ticket" in cookie:
        cookie = cookie.split(";")
        for i in cookie:
            if i.split("=")[0] == "login_ticket":
                Cookie["login_ticket"] = i.split("=")[1]
                break
        req = requests.get(url=cookieUrl.format(Cookie["login_ticket"]))
        data = json.loads(req.text.encode('utf-8'))
        if "成功" in data["data"]["msg"]:
            Cookie["stuid"] = str(data["data"]["cookie_info"]["account_id"])
            req = requests.get(url=cookieUrl2.format(Cookie["login_ticket"], Cookie["stuid"]))
            data = json.loads(req.text.encode('utf-8'))
            Cookie["stoken"] = data["data"]["list"][0]["token"]
            print("登录成功!")
            return [1, Cookie]
        else:
            print("cookie已失效,请重新登录米游社抓取cookie")
            return [0, "cookie已失效,请重新登录米游社抓取cookie"]
    else:
        print("cookie中没有'login_ticket'字段,请重新登录米游社, 重新抓取cookie!")
        return [0, "cookie中没有'login_ticket'字段,请重新登录米游社, 重新抓取cookie!"]

```

之后将该内容加入到请求头中, 使用requests库既可实现签到和查询帖子的功能, 本部分较为常规。签到&查看帖子具体代码如下:

```

def signInYS(self):
    log.info("正在签到.....")
    header = {}
    header.update(self.headers)
    ys = gameList[1]

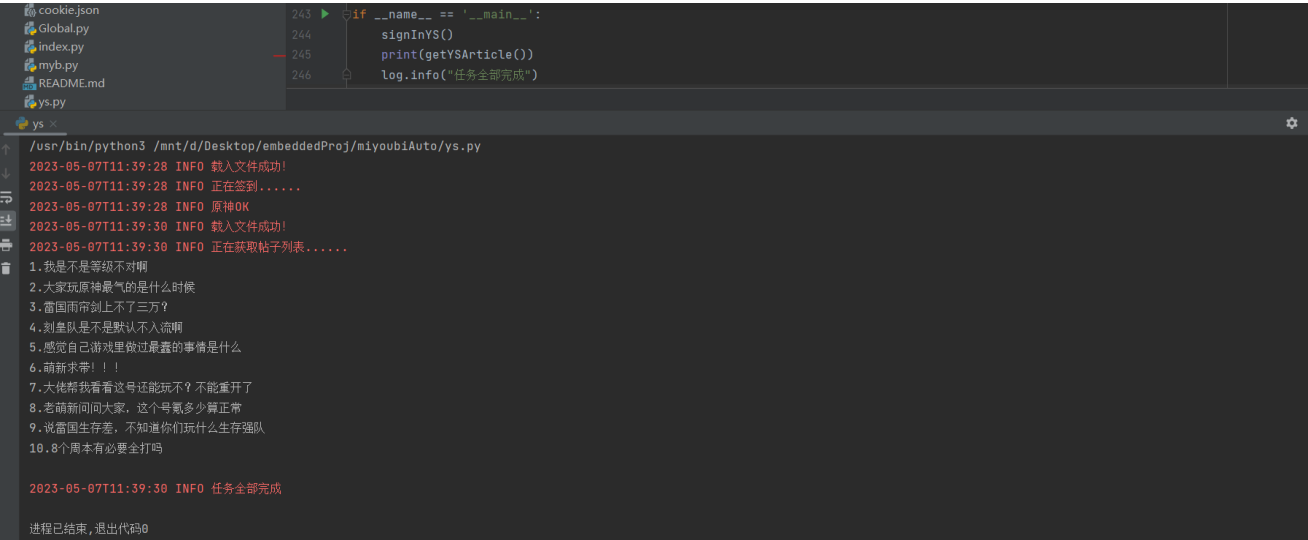
    header["DS"] = DSGet2("", json.dumps({"gids": ys["id"]}))
    req = requests.post(url=signUrl, json={"gids": ys["id"]}, cookies=self.Cookie,
headers=header)
    data = json.loads(req.text.encode('utf-8'))
    if "err" not in data["message"]:
        log.info(str(ys["name"])+ data["message"]))
        time.sleep(2)
        if "成功" in data["message"]:
            return True
        else:
            return False
    else:
        log.info("签到失败, 你的cookie可能已过期, 请重新设置cookie。")
        with open(f"{PATH}/cookie.json", "w") as f:
            f.write('')
            f.close()
            return False

def returnArticles(self):
    List = []

```

```
ret_str = ""
log.info("正在获取帖子列表.....")
ys = gameList[1]

req = requests.get(url=listUrl.format(ys["forumId"]), headers=self.headers)
data = json.loads(req.text.encode('utf-8'))
# 获取十个帖子
for n in range(10):
    List.append(data["data"]["list"][n]["post"]["subject"])
    ret_str += str(n+1)+"."+data["data"]["list"][n]["post"]["subject"] + "\n"
return ret_str
```



使用chatgpt生成对话

如果用户的输入不是以上任何特殊命令的话，语音助手将调用open ai的接口，生成对话。 GPT-4 目前处于 Limited beta 阶段，只有获得访问权限的人才能访问。GPT-3.5 模型可以理解 and 生成自然语言或代码。网上可以查到的大多数教程都使用的是text-davinci-002。在本项目中，我使用了在 GPT-3.5 系列中功能最强大、最具成本效益的型号gpt-3.5-turbo，它已针对聊天进行了优化，但也适用于传统的补全（Completion）任务。以下摘自官方文档：

MODEL	描述	最大Token数	训练数据
gpt-3.5-turbo	功能最强大的 GPT-3.5 型号，针对聊天进行了优化，成本仅为 text-davinci-003 的 1/10。将使用我们最新的模型迭代进行更新。	4096 Token	截至 2021 年 9 月
gpt-3.5-turbo-0301	gpt-3.5-turbo 2023 年 3 月 1 日的快照。与 gpt-3.5-turbo 不同，此模型不会更新，并且仅在 2023 年 6 月 1 日结束的三个月内提供支持。	4096 Token	截至 2021 年 9 月
text-davinci-003	可以以比 curie、babbage、ada 模型更好的质量、更长的输出和一致的指令遵循来完成任何语言任务。还支持在文本中 插入补全。	4097 Token	截至 2021年6月
text-davinci-002	与 text-davinci-003 类似的功能，但使用监督微调而不是强化学习进行训练	4097 Token	截至 2021年6月
code-davinci-	针对代码完成任务进行了优化	8001 Token	截至 2021年6



接口返回的数据格式如下：

```
{
  "choices": [
    {
      "finish_reason": "stop",
      "index": 0,
      "message": {
        "content":
"\u4f60\u597d\uff01\u6709\u4ec0\u4e48\u6211\u53ef\u4ee5\u5e2e\u52a9\u4f60\u7684\u5417\uff1f",
        "role": "assistant"
      }
    }
  ],
  "created": 1683438625,
  "id": "chatcmpl-7DRZp3IWnHvfXtKl4KOGCgrVsFX",
  "model": "gpt-3.5-turbo-0301",
  "object": "chat.completion",
  "usage": {
    "completion_tokens": 18,
    "prompt_tokens": 23,
    "total_tokens": 41
  }
}
```

以下为对话生成模块的测试代码（调用open ai 的接口需要翻墙以及充值，翻墙和充值的具体细节就不在本报告中展开了）：

```
import openai

# 设置 OpenAI API 密钥
openai.api_key = 'sk-xxx'

# 设置prompt（用于提示的文本）
prompt = "你好呀"

# 使用 OpenAI API 生成对话回复
def generate_reply(prompt):
    response = openai.ChatCompletion.create(
        model="gpt-3.5-turbo",
        messages=[
            {"role": "system", "content": "You are a helpful assistant."},
            {"role": "user", "content": prompt}
        ],
        max_tokens=60,
        n=1,
        temperature=0.5,
    )
    # print(response)
    return response.choices[0].message.content.strip()

# 打印开始信息及生成的回复，用于debug
print("start")
print(generate_reply(prompt))
```

```
> paimon
> ref
> robot-master
> snowboy-master
> VITS-main
  API key.txt
  audio.mp3
  geckodriver.log
  main.py
  output.wav
  report.md
  requirements.txt
  result.mp3
  test01-baidu.py
  test02-openai.py
  test03-pyaudio.py
  test04-paimeng.py
  test05-crawler.py
  test06-playsound.py
  test07-qianbao.py
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
model="gpt-3.5-turbo",
messages=[
    {"role": "system", "content": "You are a helpful assistant."},
    {"role": "user", "content": prompt}
],
max_tokens=60,
n=1,
temperature=0.5,
)
# print(response)
return response.choices[0].message.content.strip()

# 打印生成的回复
print("start")
print(generate_reply(prompt))
```

运行: test02-openai

```
/usr/bin/python3 /mnt/d/Desktop/embeddedProj/test02-openai.py
start
你好! 有什么我可以帮助你的吗?
进程已结束,退出代码0
```

## 语音合成TTS

这个部分是本项目较大的创新点，因为没有直接调用现成的api。而是在本地部署VITS模型进行生成。

VITS (Variational Inference with adversarial learning for end-to-end Text-to-Speech) 是一种结合变分推理 (variational inference)、标准化流 (normalizing flows) 和对抗训练的高表现力语音合成模型。VITS通过隐变量而非频谱串联起来语音合成中的声学模型和声码器，在隐变量上进行随机建模并利用随机时长预测器，提高了合成语音的多样性，输入同样的文本，能够合成不同声调和韵律的语音。

在本项目中，考虑到我笔记本算力的局限性，我直接下载了网络上别人训练好的派蒙音效的模型文件，超过400MB的Paimon.pth。并且截取了VITS中推理部分。本模块测试代码如下：

```
import socket
import torch
import utils
from models import SynthesizerTrn
from text.symbols import symbols
import soundfile as sf

# 配置文件名，保存路径等参数
text = "旅行者，我在"
length_scale = 1
filename = 'results/result123'
audio_path = f'{filename}.mp3'

# 创建模型，加载派蒙音效模型参数
hps = utils.get_hparams_from_file("./configs/biaobei_base.json")
model = SynthesizerTrn(
    len(symbols),
    hps.data.filter_length // 2 + 1,
    hps.train.segment_size // hps.data.hop_length,
    **hps.model.cuda()
)
model.eval()
utils.load_checkpoint('model/Paimon.pth', model)
```



```

stn_tst = utils.get_text(text, hps)
with torch.no_grad():
    x_tst = stn_tst.cuda().unsqueeze(0)
    x_tst_lengths = torch.LongTensor([stn_tst.size(0)]).cuda()
    audio = model.infer(x_tst, x_tst_lengths, noise_scale=.667, noise_scale_w=0.8,
length_scale=length_scale)[0][
    0, 0].data.cpu().float().numpy()
    # 将语音合成的结果写入到指定mp3文件中
    sf.write(audio_path, audio, samplerate=hps.data.sampling_rate)

print("已完成语音生成")

```

😞 那么，只要将所有模块的代码进行整合，然后在树莓派上运行就可以了？

😞 并不是，我在测试这个推理模型的时候，是使用的笔记本WSL的环境，当我把相同的代码拷贝到树莓派上的时候，出现了Illegal instruction (core dump)的报错。我尝试了安装对应版本的pytorch，并且删除了所有涉及把数据转移到gpu的代码，都无果。一度想过放弃。在最初提交展示作品的时候，我最完整的视频也是直接调用百度api进行语音合成。在预约课程展示当天早上，我突然想到，可以通过网络来传递文件。就把我的笔记本作为一个语音合成的服务器，然后把笔记本上合成的结果发给树莓派不就可以了？于是，便有了一下的解决方案，也是我实现派蒙语音助手最关键的一步（毕竟，没有派蒙音效的语音助手怎么能称作派蒙语音助手呢？）：



当测试demo跑通的时候，我非常激动。但是，我马上发现了问题：因为以太网的MTU（Maximum Transmission Unit 最大传输单元）是1500字节，这对于我们的音频文件来说，一次send显然是不足够的。这导致，我在树莓派上接收到的音频，只能播放不完整的一小段。为了解决socket的大文件传输通过sendall拆分成多个小包发送。并且在发送前，首先向树莓派通知文件的完整大小，此后，服务端通过sendall发送数据，而客户端（树莓派）则核验已经接收到的文件大小，在未达到先前告知的文件的完整大小之前，循环调用recv，以append的方式进行接收。

因为是本地部署的模型，所以肯定没有百度的语音合成那么智能。在测试中，我发现，VITS并不能将0-9的阿拉伯数字转变为语音，而是选择直接略过。所以我写了辅助函数convert\_number\_to\_chinese来帮助将数字转换为汉字的格式。有一些特别需要注意的是对于二位数77，70等，我们不应该直接转换成七七，而应该转换成七十七。

```

# 树莓派
import socket
from playsound import playsound

server_addr = '172.23.116.224'
server_port = 8989

sk_pi = socket.socket() # 实例化一个socket对象
sk_pi.connect((server_addr, server_port)) # 作为client向windows发起请求

text = "大家好，我是派蒙"
# 发送需要合成的文字
sk_pi.send(text.encode('utf-8'))

```

```

ret = sk_pi.recv(1024).decode('utf-8')
len_file = int(ret)
sk_pi.send("Ack from pi: 已知需要接收大小为{}的文件".format(len_file).encode('utf-8'))

save_filename = 'audio_from_win.mp3'
already_receive = 0
with open(save_filename, 'wb+') as f:
    while True:
        ret = sk_pi.recv(1024)
        f.write(ret)
        already_receive += len(ret)
        if already_receive >= len_file:
            break

playsound(save_filename)

sk_pi.send('Receive from pi: 接收成功'.encode('utf-8'))
sk_pi.close()

# windows
import socket
import re

def convert_number_to_chinese(num):
    num = int(num)
    units = ['', '十', '百', '千']
    nums = '零一二三四五六七八九'
    result = []

    if num < 10:
        return nums[num]

    num_str = str(num)
    num_len = len(num_str)

    for i, n in enumerate(num_str):
        if n == '0':
            if result and result[-1] != '零':
                result.append('零')
        else:
            result.append(nums[int(n)])
            result.append(units[num_len - i - 1])

    return ''.join(result).rstrip('零')

import re
def replace_numbers_with_chinese(text):
    def replace(match):
        number = match.group(0)
        return convert_number_to_chinese(number)

    return re.sub(r'\d+', replace, text)

server_addr = '172.23.116.224'
server_port = 8989
listen_addr = '10.181.243.251'

```

```

listen_port = 6666

sk_s = socket.socket() # 实例化一个对象sk
sk_s.bind((listen_addr, listen_port)) # 把地址绑定到套接字
sk_s.listen() # 监听链接
print("windows开始监听")

while True:
    conn_pi, addr_pi = sk_s.accept() # 接收客户端链接
    print(addr_pi) # 打印出客户端的地址+端口

    # 先接受等待转换成语音的文本
    text = conn_pi.recv(1024).decode('utf-8')
    print('从pi接收: ',text)

    # 2023/05/06 去除换行符
    text = text.replace('\n', '')
    # print("去除换行符后结果:")
    # print(text)
    pattern = re.compile(r"^[^da-zA-Z0-9\u4e00-\u9fa5]+")
    text = pattern.sub("", text)
    text = replace_numbers_with_chinese(text)
    print(text)

    # 向wsl请求生成语音
    # 作为client向wsl发起请求
    sk_c = socket.socket() # 创建客户端套接字
    sk_c.connect((server_addr, server_port)) # 连接服务器(ip地址+端口)

    sk_c.send(text.encode('utf-8'))

    ret = sk_c.recv(1024) # 等待wsl完成任务的消息
    print("从wsl接收: ",ret.decode('utf-8'))
    sk_c.send('windows收到消息!'.encode('utf-8'))
    sk_c.close() # 关闭客户端套接字

    filename = './results/result123.mp3'
    with open(filename,"rb") as f:
        rdata = f.read()

    len_data = len(rdata)
    print("向pi发送的文件大小为: ",len_data)
    conn_pi.send(str(len_data).encode('utf-8'))
    ret = conn_pi.recv(1024).decode('utf-8')
    print(ret)

    conn_pi.sendall(rdata)

    ret = conn_pi.recv(1024)
    print(ret.decode('utf-8'))
    conn_pi.close() # 关闭客户端套接字

sk_s.close() # 关闭服务器套接字

# WSL
import socket
import torch

```

```

import utils
from models import SynthesizerTrn
from text.symbols import symbols
import soundfile as sf

text = "旅行者，我在"
length_scale = 1
filename = 'results/result123'
audio_path = f'{filename}.mp3'
# 创建模型，加载派蒙语音合成模型参数
hps = utils.get_hparams_from_file("./configs/biaobei_base.json")
model = SynthesizerTrn(
    len(symbols),
    hps.data.filter_length // 2 + 1,
    hps.train.segment_size // hps.data.hop_length,
    **hps.model).cuda()
model.eval()
utils.load_checkpoint('model/Paimon.pth', model)

listen_addr = '172.23.116.224'
listen_port = 8989

sk = socket.socket() # 实例化一个socket对象
sk.bind((listen_addr, listen_port)) # 把地址绑定到套接字
sk.listen() # 监听链接
print("开始监听")
while True:
    # 等待windows用户发送请求
    conn, addr = sk.accept() # 接收客户端链接
    print(addr) # 打印出客户端的地址+端口

    text = conn.recv(1024).decode('utf-8')
    text.replace('\n', ' ')
    print("从windwos收到，需要转换的文本为: ",text)

    # 进行推理（语音合成）
    stn_tst = utils.get_text(text, hps)
    with torch.no_grad():
        x_tst = stn_tst.cuda().unsqueeze(0)
        x_tst_lengths = torch.LongTensor([stn_tst.size(0)]).cuda()
        audio = model.infer(x_tst, x_tst_lengths, noise_scale=.667, noise_scale_w=0.8,
length_scale=length_scale)[0][
        0, 0].data.cpu().float().numpy()
        sf.write(audio_path, audio, samplerate=hps.data.sampling_rate)

    conn.send('已完成语音生成'.encode('utf-8')) # 向客户端(windows)发送合成完成的信息
    print("已完成语音生成")
    ret = conn.recv(1024) # 接收客户端信息(1024字节)
    print(ret.decode('utf-8')) # 打印客户端信息(bytes类型需要decode)
    conn.close() # 关闭客户端套接字
sk.close() # 关闭服务器套接字

```

```
('10.181.243.176', 35906)
从pi接收： 原神是一款动作冒险类游戏，由PlatinumGames开发，由Nintendo发行。游戏的背景设定在一个叫做“原神”的神秘世界，玩家将扮演一个叫做“坠落者”的角色，在神秘的世界中探索，解开谜团，并与强大的怪物和其他坠落者作战。游戏中有丰富的战斗模式，玩家可以使用各种武器，技能和技巧来击败敌人。游戏中还有一个叫做“圆神”的角色，它是游戏中最强大的敌人，玩家需要面对它，才能攻克游戏的最终关卡。
从wsl接收： 已完成语音生成
向pi发送的文件大小为： 224347
收到大小为224347的文件

('10.181.243.176', 49718)
从pi接收： 杭州的天气是多云，气温是23摄氏度。
从wsl接收： 已完成语音生成
向pi发送的文件大小为： 22194
收到大小为22194的文件

('10.181.243.176', 58758)
从pi接收： 很抱歉，我不能做你的女朋友。我认为我们之间应该建立一种友谊，而不是一种爱情关系。我们可以一起分享快乐，交流想法，支持彼此，但我不能答应你的要求。我希望我们能够建立更深厚的友谊，而不是一种短暂的爱情关系。
从wsl接收： 已完成语音生成
向pi发送的文件大小为： 113243
收到大小为113243的文件

|
```

## 播放语音合成的结果

在项目中，我是用了playsound来播放音频。代码也非常简单。

```
playsound('audio.mp3') # 传入的参数为需要播放的音频文件名
```

但是需要注意，和pyaudio一样，我们在pip install playsound之前需要安装以下依赖项

```
sudo apt install python3-gi python3-gi-cairo gir1.2-gtk-3.0
```

## 框架整合

以上介绍了本语音助手的各个模块，最后，我编写了pmain.py代码，作为程序的主循环，按序调用各个模块。至此，派蒙语音助手全部完成。

```
import u_tasks
from hotword import snowboydecoder
import sys
import signal
import speech_recognition as sr
from playsound import playsound
from aip import AipSpeech
import openai
import time
import u_record
import socket
from playsound import playsound

server_addr = '10.181.243.251'
server_port = 6666

# 替换为您的百度 API 密钥
BAIDU_APP_ID = 'xxx'
BAIDU_API_KEY = 'xxx'
BAIDU_SECRET_KEY = 'xxx'
```

```

# 替换为您的 OpenAI API 密钥
OPENAI_API_KEY = 'sk-xxx'

MAX_LENGTH = 100

openai.api_key = OPENAI_API_KEY

# 创建一个 AipSpeech 对象
baidu_client = AipSpeech(BAIDU_APP_ID, BAIDU_API_KEY, BAIDU_SECRET_KEY)

interrupted = False

# 将用户的语音保存为文件
def save_audio_to_file(audio, filename):
    with open(filename, "wb") as file:
        file.write(audio.get_wav_data())

def signal_handler(signal, frame):
    global interrupted
    interrupted = True

def interrupt_callback():
    global interrupted
    return interrupted

def detected_callback():
    playsound("paimon_response.mp3")
    time.sleep(1)
    # print("唤醒词已检测到!")
    recognize_and_respond()

def recognize_and_respond():

    print("正在倾听您的问题...")
    u_record.record_wav("output.wav")
    # try:
    print("正在识别语音...")
    # 将音频转换为可识别的格式
    # audio_data = audio.get_wav_data()
    audio_data = open("output.wav", "rb").read()
    # 1537表示中文
    response = baidu_client.asr(audio_data, 'wav', 16000, {'dev_pid': 1537})

    # print(response)
    if response['err_no'] == 0:
        text = response['result'][0]
        print(f"您说了: {text}")

        response_text = ""
        # 判断是否为特殊任务
        if text == "一键签到。":
            response_text = u_tasks.signYS()
        elif text == "我要看帖子。":
            response_text = u_tasks.getYSArticle()
        elif u_tasks.is_query_weather(text):
            response_text = u_tasks.query_weather(text)

```

```

else:
    # 若都不是，则调用chatgpt的api生成回复

    # add length limit
    text += "请用中文回答，并控制在{}字以内。".format(MAX_LENGTH)
    response_text = generate_response(text)

    # chatgpt 回复文字 -> 语音(调用百度api)
    # tts_result = baidu_client.synthesis(response_text, 'zh', 1, {'vol': 5, 'per': 0,})
    # if not isinstance(tts_result, dict):
    #     with open('audio.mp3', 'wb') as f:
    #         f.write(tts_result)

    sk_pi = socket.socket()
    sk_pi.connect((server_addr, server_port))

    text = response_text
    sk_pi.send(text.encode('utf-8'))

    ret = sk_pi.recv(1024).decode('utf-8')
    len_file = int(ret)
    sk_pi.send("即将接收大小为{}的文件".format(len_file).encode('utf-8'))

    save_filename = 'audio.mp3'
    already_receive = 0
    with open(save_filename, 'wb+') as f:
        while True:
            ret = sk_pi.recv(1024)
            f.write(ret)
            already_receive += len(ret)
            if already_receive >= len_file:
                break

    print(f"助手回答: {response_text}")
    playsound('audio.mp3')

else:
    print("抱歉，我没有听清楚。")

def generate_response(prompt):
    print("等待chatgpt生成回复...")
    response = openai.ChatCompletion.create(
        model="gpt-3.5-turbo",
        messages=[
            {"role": "system", "content": "You are a helpful assistant."},
            {"role": "user", "content": prompt}
        ],
        max_tokens=60,
        n=1,
        temperature=0.5,
    )

    message = response.choices[0].message.content.strip()
    return message

# 替换为您的模型文件路径
model_path = './hotword/resources/models/paimeng.pmdl'

```

```
signal.signal(signal.SIGINT, signal_handler)

detector = snowboydecoder.HotwordDetector(model_path, sensitivity=0.5)
print("正在监听唤醒词，请说话...")

# main loop
detector.start(detected_callback=detected_callback,
              interrupt_check=interrupt_callback,
              sleep_time=0.03)

detector.terminate()
```

## 参考资料

[Snowboy唤醒](#)

[pyaudio语音录制](#)

[百度AI语音技术api](#)

[jieba分词原理](#)

[Python爬虫天气查询](#)

[米游币脚本](#)

[openAI api中文文档](#)

[VITS语音合成](#)

[Socket编程](#)