云计算项目说明文档

一. 项目背景

随着澳门回归25周年的到来，以及粤语和大湾区文化的日益受到关注，越来越多的粤语文化爱好者希望能够深入了解粤语的语音、语法和日常交流方式。为了满足这一需求，我们决定开发一个基于人工智能的粤语学习助手，旨在为广大粤语学习者提供更高效、更智能的学习体验。

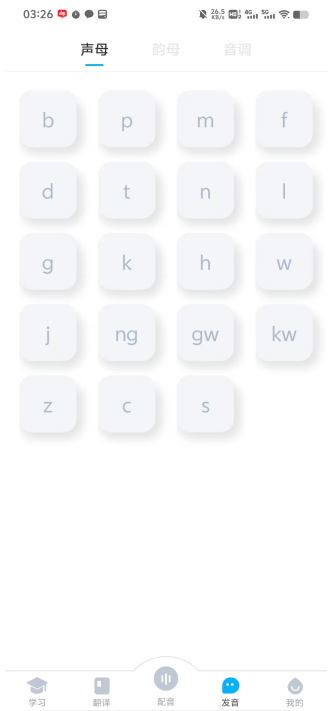
该粤语学习助手将通过三个核心功能，帮助用户提升粤语水平并加深对粤语文化的理解：

1. **发音评价**：用户朗读随机生成的语句后，助手将智能地分析用户的发音情况，并提供即时反馈，帮助用户纠正发音错误，提升语音流利度。
2. **模拟对话**：通过与助手进行模拟日常粤语对话，用户可以练习实际生活中常见的粤语交流场景，从而提高口语表达能力。
3. **粤语翻译**：用户可以输入普通话文本，助手将自动翻译为粤语，帮助用户了解粤语在不同语境中的表达方式和词汇。

该项目不仅致力于提升粤语学习的便捷性，还将助力更多人了解粤语文化与大湾区的风情。

二. 设计思路

我们首先查看了市场目前已有的粤语学习助手，如下是几张截图：



我们可以看到这些助手只能实现基本的声母、音节教学，而我们希望把粤语学习更加的具体、实际化，包括进行粤语的句子练习、更加真实的日常对话模拟等。我们在分析了发音评价、模拟对话、粤

语翻译三个功能后，选择通过将三个功能总结为背后要实现：

1.音频转文字：由于粤语有自己的粤语字，所以我们应该将音频数据通过语音匹配技术转换为粤语字，再通过大模型接口调用实现粤语字转普通话字。

2.音频比较：在发音评价功能上，我们需要对用户音频和标准音频进行比较，包括比较音频差异、语义差异，二者结合来判断发音的优劣。

3.文字转音频：我们还需要实现将普通话字转粤语字，然后再转换为粤语音频，实现翻译功能。

这样只要分别实现上面的三个技术，就可以完成我们设计的三个功能。

三. 技术实现

1. 前端部分

1.1翻译功能的前端实现

1.1.1 对话框添加对话的实现

```
function addMessage(sender, message, date) {
    const messageDiv = document.createElement('div');
    messageDiv.classList.add('message', sender);
    messageDiv.textContent = message;
    chatBox.appendChild(messageDiv);
    chatBox.scrollTop = chatBox.scrollHeight; // 滚动到最新消息
}
```

addMessage 函数根据发送者动态创建消息元素，并将其添加到聊天框，以实现即时显示。

所有消息会自动滚动到底部，以便用户始终可以看到最新消息。

1.1.2 前后端连接部分的实现

```
// 发送用户消息到后端
try {
    const response = await fetch('/api/message', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify({message: userMessage})
    });

    const data = await response.json();
    const botResponse = data.response; // 获取后端返回的响应
    addMessage('bot', botResponse, currentDate);
```

当用户输入消息并点击发送按钮（或按下回车键）时，将该消息添加到聊天框，并通过 fetch API 发送到后端的 /api/message 接口。

发送请求时，消息以 JSON 格式传输到服务器，并且等待后端返回的响应

等到后端的响应抵达前端，前端就会调用addMessage函数，将后端的响应显示在聊天框中

1.2发音练习功能的前端实现

1.2.1请求权限并录音

```
startRecordingButton.addEventListener('click', async () => {
    audioChunks = []; // 重置音频块

    const stream = await navigator.mediaDevices.getUserMedia({ audio: true });
    mediaRecorder = new MediaRecorder(stream);

    mediaRecorder.start();
    startRecordingButton.disabled = true;
    stopRecordingButton.disabled = false;

    mediaRecorder.ondataavailable = (event) => {
        audioChunks.push(event.data); // 数据可用时收集音频数据
    };
```

请求户的麦克风权限，获取到一个音频流，由于该方法是异步的，因此需要使用 await 关键字

当有可用的音频数据时，ondataavailable事件会被触发。event.data 包含了录制的音频数据，代码将这个数据推送到 audioChunks 数组中，以便后续处理（如保存或上传）。

用户点击开始录音按钮时，请求麦克风权限、开始录音，并准备好处理录制的音频数据

1.2.2音频处理发送给服务器

```
mediaRecorder.onstop = async () => {
    const audioBlob = new Blob(audioChunks, { type: 'audio/wav' });
    const audioUrl = URL.createObjectURL(audioBlob);
    audioPreview.src = audioUrl;
    audioPreview.style.display = 'block'; // 显示音频播放控件

    // 将录制的音频文件发送到服务器
    const formData = new FormData();
    formData.append('audio', audioBlob, 'recording.wav'); // 发送音频文件，命名为 recording.wav
```

使用 Blob 类创建一个包含录制音频数据的 Blob 对象。audioChunks 数组中保存了所有录制的音频数据，并指明该 Blob 的 MIME 类型为 'audio/wav'

使用 URL.createObjectURL() 方法生成一个指向 Blob 对象的 URL，这样可以在网页上引用这个音频文件。

将生成的音频 URL 赋值给音频预览元素（audioPreview）的 src 属性，以便用户可以播放录制的音频。

将音频预览控件的显示样式设置为 'block'，使其在网页上可见，用户可以看到并播放录音。

创建一个 FormData 对象，该对象用于构建一组键值对，其中包含要发送的数据。

将录制的音频 Blob 作为一个文件添加到 formData 对象中，并将其命名为 'recording.wav'。

1.2.3前后端连接部分的实现(基本同1.1.2)

1.3模拟粤语对话

功能基本同1.2

2. 后端功能实现

2.1 翻译功能实现

启发于大语言模型的强大翻译功能，我们决定用大语言模型进行双语转换的翻译。

2.1.1 创建智谱清言大预言模型

```
api_key = "cd493791757b49fdbc0eeba1367608f0.xGgQTGgq7e7jIORn"
model = ChatOpenAI(
    temperature=0.95,
    model="glm-4-flash",
    openai_api_key=api_key,
    openai_api_base="https://open.bigmodel.cn/api/paas/v4/"
)
```

通过获取智谱清言的api_key，然后通过调用langchain里的 ChatOpenAI 这个用于与 OpenAI 或兼容 API （智谱清言）的聊天模型交互的类，将api_key输入，temperature=0.95设置较高的随机性，model="glm-4-flash"指定要使用的大预言模型，那么一个智能的对话大语言模型就构造好了。

2.1.2 实现粤语和普通话的双语翻译

```
messages = [
    SystemMessage(
        content="首先判断我说的是粤语还是普通话，是粤语则翻译成普通话，是普通话则翻译成粤语，只输出翻译内容"),
    HumanMessage(content=user_message),
]
response = model.invoke(messages)
```

如果要实现较好的粤语和普通话的双语翻译，我们就要对已经构建的大模型指定比较完整且明确的系统信息，就像这里我指定的系统信息是“首先判断我说的是粤语还是普通话，是粤语则翻译成普通话，是普通话则翻译成粤语，只输出翻译内容”。这样的系统信息让大模型准确地知道我们的要求，能够基本没什么错误地完成翻译任务。经过测试，确实能够比较好地完成这个双语翻译任务。

2.2 发音练习功能实现

这个功能，我们的设想是能够让一个用户练习粤语的发音，我们提供一句粤语，然后用户能现场朗读，由这个功能对比已经有的粤语音频数据，对用户的发音做一个评价，以达到练习粤语发音的目的。

2.2.1 实现AudioComparator类

【1】load_and_check_audio函数

这个类是用于计算两段音频的相似度的，如果音频足够相似，则给出较高的评分；不相似，则评分会偏低

```
def load_and_check_audio(self, file_path):  2 usages
    """
    加载音频数据并检查其是否可播放。
    返回音频数据 y、采样率 sr 和一个布尔值表示音频是否可播放。
    """
    try:
        if self.file_type == 'audio':
            y, sr = librosa.load(file_path, sr=None)

        if y is None or y.size == 0:
            logging.info(f"{file_path}中没有音频数据")
            return False, None, None

        logging.info(f"音频时长: {len(y) / sr} seconds")
        return True, y, sr

    except Exception as e:
        logging.error(f"处理音频文件 {file_path} 时出错: {e}")
        return False, None, None
```

通过这个类里的load_and_check_audio函数来加载音频数据并检查其是否可播放。返回音频数据 y、采样率 sr 和一个布尔值表示音频是否可播放。

【2】compare_audio_advanced函数

```python
def compare_audio_advanced(self, file1, file2):  1 usage
    is_playable1, y1, sr1 = self.load_and_check_audio(file1)
    is_playable2, y2, sr2 = self.load_and_check_audio(file2)
    if not is_playable1:
        raise ValueError("被测的音频无法播放")
    if not is_playable2:
        raise ValueError("基准音频无法播放")
    s = time.time()

    # 检查采样率是否相同，如果不同则重新采样
    if sr1 != sr2:
        logging.info(f"采样率不同: file1 ({sr1}) vs file2 ({sr2})，正在进行重采样")
        y2 = librosa.resample(y2, orig_sr=sr2, target_sr=sr1)  # 将第二个音频的采样
        sr2 = sr1  # 将第二个音频的采样率更新为第一个音频的采样率
    # 提取MFCCs
    mfcc1 = librosa.feature.mfcc(y=y1, sr=sr1, n_mfcc=13)
    mfcc2 = librosa.feature.mfcc(y=y2, sr=sr2, n_mfcc=13)
    # 标准化MFCCs
    mfcc1 = (mfcc1 - np.mean(mfcc1)) / np.std(mfcc1)
    mfcc2 = (mfcc2 - np.mean(mfcc2)) / np.std(mfcc2)
    # 将MFCC矩阵展平为向量
    mfcc1_vector = mfcc1.flatten()
    mfcc2_vector = mfcc2.flatten()
    # 确保两个向量长度相同
    min_length = min(len(mfcc1_vector), len(mfcc2_vector))
    mfcc1_vector = mfcc1_vector[:min_length]
    mfcc2_vector = mfcc2_vector[:min_length]
    # 计算余弦相似度
    similarity = 1 - cosine(mfcc1_vector, mfcc2_vector)
    logging.info(f"比较音频的时间为:{(time.time() - s) * 1000}ms")
    return similarity
```

这个函数用于比较两个音频文件的相似度。它首先加载并检查音频是否可播放，然后统一采样率，提取并标准化MFCC特征，最后通过余弦相似度计算两个音频的相似度，返回相似度值。

【3】compare_audio_similarity

```python
def compare_audio_similarity(self, file1, file2):  1 usage
    """
    比较两个音频之间的差异
    param file1: 第一个音频文件路径
    param file2: 第二个音频文件路径
    """
    try:
        similarity = self.compare_audio_advanced(file1, file2)
        logging.info(f"音频相似度: {similarity}")
        if similarity < 0.5:
            y1, sr1 = librosa.load(file1, sr=None)
            y2, sr2 = librosa.load(file2, sr=None)
            different_times = self.find_differences_in_time(y1, y2, sr1)
            logging.info(f"差异点的时间位置（秒）为: {different_times}")
            return False, "两个音频不相似！！！",similarity
        else:
            logging.info("两个音频是相似的")
            return True, "两个音频是相似的",similarity
    except ValueError as e:
        logging.info(f"处理音频文件中的异常: {e}")
        return False, f"发生错误: {str(e)}",similarity
```

这个函数比较两个音频文件的相似度。它调用 compare_audio_advanced 计算相似度，如果相似度低于 0.5，则查找音频差异的时间点并返回不相似的结果；否则返回相似的结果。如果处理过程中发生错误，会捕获异常并返回错误信息。

2.2.2 通过AudioComparator类完成音频相似度打分

```python
file1 = '../data/yue/clips/common_voice_yue_31172849.mp3'
wav_file_path = 'converted_audio.wav'
convert_mp3_to_wav(file1, wav_file_path)
file2 = file_path
comparator = AudioComparator(file_type='audio')
# 比较音频文件
result, message, similarity = comparator.compare_audio_similarity(wav_file_path, file2)
if similarity > 0.6:
    similarity = similarity * 1.2
else:
    similarity = similarity * 0.5
ret_message = message + "结果评分" + str(similarity)
return jsonify({'text': ret_message})  # 返回假设的文本结果
```

这段代码的主要功能是比较两个音频文件的相似度，并根据相似度调整评分后返回结果。具体步骤如下：

1. **音频格式转换**：将 MP3 文件 file1（标准粤语语音数据） 转换为 WAV 格式，保存为 wav_file_path。
2. **音频比较**：使用 AudioComparator 类的 compare_audio_similarity 方法比较转换后的 WAV 文件和目标文件 file2（从前端获取得wav格式音频数据），获取相似度结果。

3. **调整评分**：如果相似度大于 0.6，则提高评分（乘以 1.2）；否则降低评分（乘以 0.5）。这是我们经过好多次次细节调试，得出的比较合理得评分调整。
4. **返回结果**：将比较结果和调整后的评分拼接成消息，并通过 jsonify 返回 JSON 格式的响应。

最终返回的结果包含音频是否相似的信息以及调整后的评分。

2.3 模拟粤语对话

这个功能最终实现目的是，用户能够与助手进行模拟日常粤语交流。我们通过查阅大量资料，了解到百度有能够识别粤语语音成粤语文字的模型，我们可以注册获得api，并使用这个功能，而后构建一个聊天对话机器人即可

2.3.1 接收保存前端传来的语音

```
if 'audio' not in request.files:
    return jsonify({'error': '没有音频文件'}), 400

audio_file = request.files['audio']
file_path = os.path.join(UPLOAD_FOLDER1, audio_file.filename)
audio_file.save(file_path)  # 保存上传的音频文件
print(f'上传文件已保存到: {file_path}')  # 打印文件保存路径
```

分析：这段代码接收了前端传来的音频文件，也就是用户对话所说的话并将它保存为wav格式。

2.3.2 convert_wav_to_wav

```
def convert_wav_to_wav(wav_file_path, output_wav_file_path):
    """直接使用 FFmpeg 转换 WAV 文件，并调整采样率、采样宽度、声道数和增益"""

    # FFmpeg 命令
    command = [
        "ffmpeg",
        "-i", wav_file_path,  # 输入文件x`
        "-ar", "16000",  # 设置采样率为16000 Hz
        "-ac", "1",  # 设置为单声道
        "-sample_fmt", "s16",  # 设置为 16-bit
        "-filter:a", "volume=10dB",  # 增加 10 dB
        output_wav_file_path  # 输出文件路径
    ]

    try:
        # 执行 FFmpeg 命令
        subprocess.run(command, check=True)
        print(f"文件已成功转换并保存为 {output_wav_file_path}")
    except subprocess.CalledProcessError as e:
        print(f"转换失败：{e}")
```

这个函数的主要功能是调整wav音频的采样率（16000 Hz）、采样宽度（16-bit）、声道数（单声道）以及增益（增加 10 dB）。这一点是非常重要的，因为百度的粤语语音识别api对输入语音的这些参数有非常严格的要求，尤其是采样率。（通过实践得出的经验，当时32000的采样率，模型识别完全不正确）

2.3.3 speech_to_text

```
def speech_to_text(audio_file_path):
    """使用百度云语音识别将音频文件转为文本"""
    with open(audio_file_path, 'rb') as f:
        audio_data = f.read()

        result = client.asr(audio_data, format: 'wav', rate: 16000, options: {
            'dev_pid': 1637,  # 语言参数设置为粤语
        })

        if result.get('err_no') == 0:
            return result['result'][0]
        else:
            return None
```

严格根据百度云上的教程写的使用百度云语音识别将音频文件转为文本的函数。

2.3.4 进行粤语语音识别

```
wav_file_path = file_path
# 生成一个范围在 0 到 1000 之间的随机整数
random_number = random.randint( a: 0,  b: 1000)
output_wav_file_path = f'./uploads/output{random_number}.wav'
convert_wav_to_wav(wav_file_path, output_wav_file_path)
# convert_mp3_to_wav(mp3_file_path, wav_file_path)

# 然后将转换后的 WAV 文件传给百度云进行语音识别
transcription = speech_to_text(output_wav_file_path)
```

这个代码调用convert_wav_to_wav将用户的语音数据调整之后，调用speech_to_text将语音数据翻译成粤语文本。

2.3.5 粤语到普通话的翻译

```
api_key = "cd493791757b49fdbc0eeba1367608f0.xGgQTGgq7e7jIORn"
model = ChatOpenAI(
    temperature=0.95,
    model="glm-4-flash",
    openai_api_key=api_key,
    openai_api_base="https://open.bigmodel.cn/api/paas/v4/"
)

# 粤语到普通话翻译
messages = [
    SystemMessage(content="这是一句粤语，翻译成普通话，只输出翻译内容"),
    HumanMessage(content=transcription),  # 使用 transcription 变量替换原有文本
]

response = model.invoke(messages)
```

因为语音转的文字是粤语，所以我们调用上面的粤语翻译功能进行翻译成普通话，使得后续对话机器人效果更好。

2.3.6 对话机器人的实现

【1】历史对话储存

```
# 历史会话存储
store = {}

# 获取会话历史
def get_session_history(session_id: str) -> BaseChatMessageHistory:
    if session_id not in store:
        store[session_id] = ChatMessageHistory()
    return store[session_id]
```

因为模型本身是没有任何状态概念的，所以我们可以使用消息历史类来包装我们的模型，使其具有状态。这将跟踪模型的输入和输出，并将其存储在某个数据存储中。未来的交互将加载这些消息，并将其作为输入的一部分传递给链。通过定义一个 get_session_history 函数，它接受一个 session_id 并返回一个消息历史对象。这个 session_id 用于区分不同的对话，并应作为配置的一部分在调用新链时传入。

【2】提示词模板

```
# 提示模板
prompt = ChatPromptTemplate.from_messages(
    [
        (
            "system",
            "你是一个有用的助手。尽你所能回答所有问题。",
        ),
        MessagesPlaceholder(variable_name="messages"),
    ]
)
```

提示词模板帮助将原始用户信息转换为大型语言模型可以处理的格式。我这里设置一个系统提示词，“你是一个有用的助手。尽你所能回答所有问题。”

【3】调用模型通义千问

```
os.environ["DASHSCOPE_API_KEY"] = "sk-9041beedef014f4cb57ff1ef1dc6cd33"
# 使用 Tongyi LLM
model = Tongyi()
```

获取api_key后调用Tongyi模型

【4】管理历史对话

```
# k为记录最近历史数量
def filter_messages(messages, k=10):
    return messages[-k:]

chain = (
        RunnablePassthrough.assign(messages=lambda x: filter_messages(x["messages"]))
        | prompt
        | model
)

# 历史消息
with_message_history = RunnableWithMessageHistory(
    chain,
    get_session_history,
    input_messages_key="messages",
)

config = {"configurable": {"session_id": "abc2"}}

response = with_message_history.invoke(
    input: {"messages": [HumanMessage(content=f"{translated_text}")]},
    config=config
)
```

如果对话不加管理，消息列表将无限增长，并可能溢出 LLM 的上下文窗口。因此，添加一个限制您传入的消息大小的步骤非常重要。这里的具体实现如下：

一个基于历史消息的对话链，结合了历史记录过滤和模型调用功能。以下是代码的主要功能：

1. filter_messages **函数**：
   - 用于从消息列表中提取最近的 `k` 条历史消息（默认最近 10 条）。
2. chain **构建**：
   - 使用 RunnablePassthrough 将过滤后的历史消息传递给 `prompt` （提示模板）。
   - 将处理后的输入传递给 model（语言模型）进行响应生成。
3. RunnableWithMessageHistory：
   - 将 chain 与历史记录管理功能结合，支持基于会话 ID 的历史消息存储和检索。
   - get_session_history 是获取历史记录的函数。
4. invoke **调用**：
   - 向对话链发送一条用户消息（HumanMessage），并指定会话 ID（session_id 为 "abc2"）。
   - 模型会根据历史消息和当前输入生成响应。

2.4.1注册功能

```
with open(users_file, 'r+') as fl:
    users = json.load(fl)
    if username in users:
        return False
    # 密码哈希化后存储
    users[username] = [generate_password_hash(password), name]
    fl.seek(0)
    json.dump(users, fl)
    fl.truncate()
```

打开 users_file 并以读写模式 (r+) 读取现有用户数据。

使用 json.load() 方法将文件内容加载为一个 Python 字典（users）。

如果用户名不存在，使用 generate_password_hash(password) 对密码进行哈希化，增加安全性，然后将用户名、哈希密码和姓名以列表的形式添加到 users 字典中。

使用 fl.seek(0) 将文件指针移动到文件的开头，清空文件后 (fl.truncate())，将更新后的用户字典写回文件。

2.4.2登录功能

```python
def verify_user(username, password):
    global userName
    userName = username
    with open(users_file) as f:
        users = json.load(f)
        # 用户存在且密码验证通过
        return username in users and check_password_hash(users[username][0], password)
```

函数接受用户名和密码参数，并将 username 赋值给全局变量 userName，用于后续识别。

打开 users_file 并读取用户数据到 users 字典中。

检查提供的用户名是否在 users 中，并同时验证密码是否正确。通过 check_password_hash(users[username][0], password) 对存储的哈希密码进行验证。

如果用户名存在且密码验证通过，函数返回 True

四. 团队分工

周向子健：项目构想、发音评价模型的训练、ppt制作和汇报

周文皓：后端功能实现，包括粤语翻译、对话系统、功能的集成和调试

杨植楠：前端页面代码，后端的框架部分，前端和后端的交互

侯志远：查找粤语数据集，对数据集进行标注划分，云服务器部署