# 目　　录

[1](#_Toc201774707)

[1](#_Toc201774708)

[1](#_Toc201774709)

[2](#_Toc201774710)

[2](#_Toc201774711)

[2](#_Toc201774712)

[3](#_Toc201774713)

["第二章 模块实现" 4](#_Toc201774714)

[4](#_Toc201774715)

[(一) 语音端点检测算法 4](#_Toc201774716)

[5](#_Toc201774717)

[6](#_Toc201774719)

[8](#_Toc201774720)

[9](#_Toc201774721)

[9](#_Toc201774722)

[10](#_Toc201774723)

[10](#_Toc201774724)

[10](#_Toc201774725)

[10](#_Toc201774726)

[11](#_Toc201774727)

[11](#_Toc201774728)

[11](#_Toc201774729)

[12](#_Toc201774730)

[12](#_Toc201774731)

[12](#_Toc201774732)

[12](#_Toc201774733)

[13](#_Toc201774734)

["第三章 实验结果输出与分析" 14](#_Toc201774735)

[14](#_Toc201774736)

[14](#_Toc201774737)

[15](#_Toc201774738)

[15](#_Toc201774739)

[15](#_Toc201774740)

[16](#_Toc201774741)

[17](#_Toc201774742)

[20](#_Toc201774743)

附录：

参考文献 …………………………………………………………..21

# 

智能语音系统融合了语音识别、自然语言处理和语音合成等技术，旨在使计算机能够“听懂、说出、理解”人类语言，从而实现自然的人机交互体验。随着深度学习、云计算等技术的发展，语音识别和机器翻译的精度不断提高。近年来，科大讯飞等厂商的语音识别准确率已从2010年约60%提高到2021年超过98%，同时新的端到端模型（OpenAI的Whisper）能够在多语种语音转写和翻译任务上取得很好的效果。语音翻译系统作为语音识别和机器翻译技术的综合应用，被广泛关注和研究。本系统以语音识别与翻译为核心，实现从语音输入到翻译输出的智能处理过程。实现了**“中文语音识别 → 中文文本翻译成英文 → 英文语音合成”**的完整流程

## 

本实验旨在综合运用语音识别、自然语言处理与语音合成等人工智能技术，完成一个端到端的中文语音翻译系统的设计与实现。通过本实验，学生能够：

1.掌握语音端点检测（VAD）、MFCC 特征提取等语音预处理基础方法；

2.熟悉使用深度学习模型（如 Whisper）进行语音识别的流程与接口；

3.理解并调用翻译 API 完成机器翻译任务；

4.掌握文本转语音（TTS）技术基本原理及实现方法；

5.综合使用 Python 完成一个多模块集成的完整语音翻译系统；

6.提高模块化程序设计与图形用户界面（GUI）开发能力；

7.培养解决语音交互实际问题的工程实践能力与创新意识。

## 

本实验的主要任务是设计并实现一个中文语音 → 英文文本翻译 → 英文语音合成的智能语音翻译系统。具体内容包括：

1. **语音端点检测模块**：对用户输入的语音信号进行端点检测，截取出有效语音区域；
2. **语音预处理模块**：对音频进行归一化、降噪等处理；
3. **特征提取模块**：使用 MFCC 算法提取语音特征；
4. **语音识别模块**：基于 Whisper 模型实现中文语音到文本的转换；
5. **翻译模块**：调用百度翻译 API，将识别文本从中文翻译成英文；
6. **语音合成模块**：使用 TTS 技术将英文翻译文本合成为语音文件；
7. **图形化界面模块**：使用 Tkinter 实现系统操作界面，支持录音、显示文本、播放合成语音等功能；
8. **波形图/频谱图展示**：使用 librosa 和 matplotlib 展示各阶段语音处理图像，用于性能分析；
9. **系统测试与结果评估**：在多种语音输入条件下进行测试，并评估识别率、翻译准确性与响应速度。

## 

### 

智能语音系统的功能流程图如图2-1所示。系统首先通过音频采集模块获取用户的语音信号，对信号进行预处理（去噪、端点检测等），然后将预处理后的音频传递给语音识别模块（ASR），将其转换成文字文本；接着，通过机器翻译模块（MT）对识别得到的源语言文本进行翻译，得到目标语言的文本；最后，系统可通过语音合成模块（TTS）将翻译后的文本转换成语音信号输出。用户界面或控制模块负责启动各模块及显示识别、翻译结果，实现用户与系统的交互。系统整体流程简洁高效，采用级联式架构。

下面为语音翻译系统的简单流程图：

语音信号输入 → [音频采集/预处理] → [语音识别ASR] → 识别文本 → [机器翻译MT] → 翻译文本 → [语音合成TTS] → 翻译语音输出

### 

系统主要划分为以下模块：

**音频采集模块**：采集麦克风输入或音频文件。依赖硬件接口库（如PyAudio）实现实时采样，并包含预处理功能，如滤波和端点检测。

**语音识别模块**：将音频信号转写为文字文本。可以采用现有的ASR引擎(如Google Cloud Speech、科大讯飞等）或开源模型（如DeepSpeech、OpenAI Whisper）。该模块依赖语音识别SDK或深度学习模型，并提供API调用接口。

**机器翻译模块**：对识别得到的源语文本进行翻译。可调用在线翻译API（如Google Translate、百度翻译）或离线神经网络翻译模型。依赖网络接口库（HTTP请求）或深度学习框架。

**语音合成模块**：将翻译后的文本转换为目标语言语音。可使用TTS库（如gTTS、百度语音合成等），依赖音频输出接口。

**用户界面/控制模块**：提供运行界面或命令行，管理各模块调用、显示结果。可视化模块支持实时字幕显示、翻译文本框显示等。

### 

本系统采用主流深度学习和API技术，具体选型理由如下：

**编程语言与开发环境**：选择 Python 语言进行快速开发，因其拥有丰富的语音处理和网络库。开发环境使用Python 3.8。

**语音识别技术**：考虑到开发效率与识别准确率，优先使用现有成熟服务。可调用Google Cloud Speech-to-Text API，支持中文普通话语音识别，精度高、延迟低。对于端到端模型，应用OpenAI Whisper等新模型，其基于Transformer架构，能实现多语言语音识别与语音翻译。

**机器翻译技术**：使用**Baidu Translate API**，或开源的 Transformer 模型。综合考虑成本和效果，首先使用免费翻译API进行开发与测试。

**语音合成技术**：利用开源 TTS 引擎（gTTS）或在线服务将文本转为语音。

**其他技术**：音频预处理可使用 Python 的 SciPy、Librosa 库进行滤波、端点检测等。界面开发可采用简易的 Tkinter GUI 或命令行显示。系统整体采用模块化设计。

# "第二章 模块实现"

## 

### 语音端点检测算法

**1. 算法描述**

该算法用于从原始语音中提取有语音活动（Voiced）的片段，剔除静音和背景噪声，以提升后续识别效果。其核心是使用 Google 提供的 webrtcvad 模块进行每帧语音的语音活动检测（Voice Activity Detection, VAD）。

输入：

wav\_path: 语音文件路径（需为 .wav 格式，采样率通常为 16kHz）。

aggressiveness: VAD 检测激进程度（范围为 0~3，值越大越“挑剔”）。

输出：

voiced: 一个仅包含有语音部分的 numpy 音频数组。

核心思想：

将语音信号分割为固定时长的帧（frame），对每帧使用 webrtcvad 判断是否包含人声。如果是人声帧，则保留该段数据；否则丢弃。最后返回拼接后的语音段。

**2. 算法结构分析（伪代码）**

|  |
| --- |
| 函数 VAD\_Segment(音频路径, 激进程度): |
| 初始化 VAD 检测器，设置激进程度 |
| 读取音频数据为 audio, sample\_rate |
| 将 audio 数据转为 16-bit 整型格式 |
|  |
| 设置帧长为 30ms |
| 计算每帧对应的采样点数 frame\_size |
| 计算总帧数 num\_frames |
|  |
| 初始化空的 voiced 列表 |
|  |
| 对于每一帧 i ∈ [0, num\_frames): |
| 提取当前帧的音频数据为 frame |
| 将该帧转为字节格式 |
| 若 is\_speech(frame, sample\_rate) 返回 True: |
| 将该帧加入 voiced 结果中 |
|  |
| 返回拼接后的有语音部分（voiced）数组 |

### 

**1. 算法描述**

|  |  |  |
| --- | --- | --- |
| **函数名** | **作用** | **说明** |
| pre\_emphasis() | 预加重 | 高通滤波，增强高频，抑制低频能量 |
| frame\_signal() | 分帧 | 将连续语音信号按时间滑窗分割成帧 |
| hamming\_window() | 加窗 | 减少分帧引入的频谱泄漏现象 |
| preprocess() | 总控函数 | 按顺序完成上述三个步骤，返回预处理结果 |

Step 1预加重 Pre-emphasis

目的：增强高频成分、补偿声带衰减，提高语音信号的高频特征识别能力。

Step 2分帧 Frame Splitting

目的：将连续语音信号切分为短时帧（通常为 25ms），便于在局部稳态假设下进行特征提取。

参数设置：

frame\_size = 0.025 秒（25ms）

frame\_stride = 0.01 秒（10ms）

重叠率约为 60%

实现过程：

1. 计算帧长和帧移（单位为采样点）；
2. 根据信号长度计算帧数，若不足则补零；
3. 使用索引构造帧切片矩阵。

### Step 3加窗 Windowing

#### 目的：抑制帧端信号的不连续性，避免频谱泄漏现象。常用窗函数为 ****Hamming 窗****。

原始语音 signal

↓

┌───── 预加重 ─────┐

↓ ↓

emphasized 高频增强

↓

┌────── 分帧 ──────┐

↓ ↓

frames 时域切片

↓

┌────── 加窗 ──────┐

↓ ↓

windowed 平滑帧信号

↓

输出 → 特征提取（MFCC等）

**2. 算法结构分析（伪代码）**

|  |
| --- |
| 函数 preprocess(signal, sample\_rate): |
| 1. 调用 pre\_emphasis(signal, coeff): |
| 设置 coeff = 0.97 |
| 对于每个样本 i: |
| 输出[i] = 输入[i] - 0.97 \* 输入[i - 1] |
| 得到强调后的信号 emphasized\_signal |
|  |
| 2. 调用 frame\_signal(emphasized\_signal, sample\_rate): |
| 计算每帧样本数 frame\_length = frame\_size × sample\_rate （默认 0.025s） |
| 计算帧移 step = frame\_stride × sample\_rate （默认 0.01s） |
| 根据长度推算帧数 num\_frames |
| 对信号进行补零，使其整除帧长与帧移 |
| 构建帧索引矩阵 indices |
| 得到帧矩阵 frames [num\_frames × frame\_length] |
|  |
| 3. 调用 hamming\_window(frames): |
| 对每一帧乘以 Hamming 窗函数，得到 windowed\_frames |
|  |
| 4. 返回 windowed\_frames |

### 

1. **算法描述**

Step 1：快速傅里叶变换（FFT） → 功率谱

目的：

将每一帧的时域信号转换到频域，得到其功率谱（Power Spectrum），用于能量分析。

对应代码：

mag\_frames = np.absolute(np.fft.rfft(frames, n=fft\_size))

pow\_frames = (1.0 / fft\_size) \* (mag\_frames \*\* 2)

* np.fft.rfft()：对每帧执行实数FFT，输出前一半频谱（正频率部分）。
* np.absolute()：取复数频谱的模，得到幅度谱。
* mag\_frames \*\* 2 / fft\_size：归一化，得到功率谱。

Step 2：构造 Mel 滤波器组（Mel Filterbank）

目的：

模拟人耳在 Mel 频率尺度下的感知能力，对功率谱进行加权聚合。

对应代码：

mel\_filters=get\_mel\_filterbank(num\_filters=26,fft\_size=fft\_size,sample\_rate=sample\_rate)

内部核心逻辑：

mel\_points = np.linspace(hz\_to\_mel(low\_freq), hz\_to\_mel(high\_freq), num\_filters + 2)

hz\_points = mel\_to\_hz(mel\_points)

bin = np.floor((fft\_size + 1) \* hz\_points / sample\_rate).astype(int)

# 构建三角滤波器

for j in range(1, num\_filters + 1):

for i in range(bin[j - 1], bin[j]):

fbank[j - 1, i] = (i - bin[j - 1]) / (bin[j] - bin[j - 1])

for i in range(bin[j], bin[j + 1]):

fbank[j - 1, i] = (bin[j + 1] - i) / (bin[j + 1] - bin[j])

* 将频率线性划分到 Mel 频率尺度，转回 Hz。
* 构造三角形滤波器，每个滤波器覆盖一段频率区域。
* 每帧的功率谱将与这些滤波器相乘并求和，得到每个 Mel band 的能量。

Step 3：计算 Mel 能量谱

目的：

将功率谱投影到 Mel 滤波器组，模拟听觉系统对频率的聚合。

对应代码：

mel\_energy = np.dot(pow\_frames, mel\_filters.T)

mel\_energy = np.where(mel\_energy == 0, np.finfo(float).eps, mel\_energy)

* np.dot()：功率谱 × 滤波器转置，得到每帧 Mel band 的能量。
* .where(... == 0)：为了防止后续 log(0) 错误，将为 0 的值替换为极小数（机器精度极限）。

Step 4：对数压缩（Log Mel Energy）

目的：

模仿人耳对音强的感知特性（对数响应），压缩大动态范围的能量谱。

对应代码：

log\_mel\_energy = np.log(mel\_energy)

* 使用自然对数将 Mel 能量转换为对数域，使模型对能量变化更鲁棒。

Step 5：离散余弦变换（DCT）得到 MFCC 系数

目的：

将 Mel 频率能量的对数谱变换到倒谱域，提取具有区分度的语音特征。

对应代码：

mfcc = dct(log\_mel\_energy, type=2, axis=1, norm='ortho')[:, :num\_ceps]

* dct(type=2)：提取主要频率成分，去除冗余，类似 PCA。
* norm='ortho'：保证系数正交，便于后续建模。
* [:, :num\_ceps]：只保留前 num\_ceps（如13）个倒谱系数，其余常为高频噪声。

1. **算法结构分析（伪代码）**

函数 compute\_mfcc(frames, sample\_rate, fft\_size, num\_filters, num\_ceps):

1. 对每帧执行快速傅里叶变换（FFT），获得幅度谱和功率谱

mag\_frames = |FFT(frames)|

pow\_frames = mag\_frames² / fft\_size

2. 构建 Mel 滤波器组（Mel Filterbank）

mel\_filters = get\_mel\_filterbank()

3. 将功率谱投影到 Mel 滤波器组

mel\_energy = dot(pow\_frames, mel\_filters^T)

若 mel\_energy 有 0，替换为一个极小值（防止 log(0)）

4. 对 Mel 能量谱取对数（Log Mel Spectrum）

log\_mel\_energy = log(mel\_energy)

5. 对 log\_mel\_energy 执行离散余弦变换（DCT）

mfcc = DCT(log\_mel\_energy)，保留前 num\_ceps 个系数

返回 mfcc

### 

1. **算法描述**

该函数使用开源的 Whisper 模型对输入语音进行自动语音识别（ASR），将 .wav 音频文件转写为对应的中文文本。

Whisper 是 OpenAI 发布的端到端多语种语音识别模型，支持自动检测语种、时间戳定位、多任务翻译等能力。此处指定识别中文语音为目标。

1. **算法结构分析（伪代码）**

函数 transcribe(wav\_path):

1. 加载 Whisper 模型

model = whisper.load\_model("base") # 加载 base 模型（中等体积）

2. 对指定 wav 文件进行识别，设置识别语言为中文（'zh'）

result = model.transcribe(wav\_path, language='zh')

3. 返回识别结果中的文本字段

return result["text"]

## 

### 

该函数 baidu\_translate() 实现了一个面向中文到英文（或任意语言对）的在线翻译功能，基于百度提供的 “开放平台机器翻译服务”。通过构造合法的签名请求，调用百度翻译 HTTP API，获取翻译结果。

输入参数：

text: 待翻译文本（如中文语音识别输出）

from\_lang: 源语言（默认 'zh'）

to\_lang: 目标语言（默认 'en'）

retries: 出错重试次数（默认 3 次）

输出：

dst: 翻译后的目标语言文本（字符串）

函数 baidu\_translate(text, from\_lang='zh', to\_lang='en', retries=3):

重试 retries 次：

1. 生成随机 salt

2. 构造 sign = MD5(appid + text + salt + secret\_key)

3. 设置 API 请求参数：

- q: 原文

- from/to: 语种

- appid, salt, sign: 身份认证

4. 向百度翻译 API 发起 GET 请求

5. 若结果中含有 'trans\_result':

返回 result['trans\_result'][0]['dst']

否则打印错误信息，等待 1 秒重试

所有重试失败时，返回 "【翻译失败】"

### 

百度翻译使用的是基于神经网络机器翻译模型(NMT)的在线翻译服务，其原理如下：

NMT 模型基本原理：

基于 Encoder–Decoder 架构，如 LSTM/GRU 或 Transformer。

Encoder 将源语言句子编码为上下文向量；

Decoder 利用注意力机制解码为目标语言；

优点：上下文理解能力强，尤其对长句处理优于传统 SMT（短语翻译模型）。

### 

# 引入模块

from translator import baidu\_translate

# 调用方式（如在主流程中）

translated\_text = baidu\_translate("你好，世界", from\_lang='zh', to\_lang='en')

print(translated\_text)

### 

中文语音识别 Whisper

↓ 中文文本

baidu\_translate() ← Translator模块

↓ 英文翻译文本

英文语音合成 TTS

### 

{

"from": "zh",

"to": "en",

"trans\_result": [

{

"src": "你好，世界",

"dst": "Hello, world"

}

]

}

## 

### 

本模块基于微软的 Edge-TTS（即 Microsoft Azure 的 TTS 接口，但以浏览器接口形式调用）实现高质量英文语音合成，支持：

1.多个音色（不同发音人）；

2.不同语速（±20%）；

3.并发合成多个语音文件；

4.支持异步非阻塞式批量生成。

### 

函数：speak(text, voice, rate, filename)

使用 edge\_tts 创建 Communicate 对象（text, voice, rate）

异步执行语音合成并保存为文件

函数：synthesize\_by\_voice(text, output\_dir)

创建输出目录

初始化多个语音音色（voice 列表）

对每个 voice:

构造文件名

创建合成任务：speak(text, voice, 默认语速, filename)

使用 asyncio 同步执行所有任务

输出结果包括多个发音人，语速为默认 +0%，例如：

en-US-JennyNeural\_default.wav

en-GB-SoniaNeural\_default.wav

函数：synthesize\_by\_rate(text, output\_dir)

创建输出目录

使用固定 voice = Jenny

对多个语速（±0%、±10%、±20%）：

创建文件名

创建合成任务：speak(text, voice, rate, filename)

异步并发执行所有任务

函数：synthesize\_combined(text, output\_dir)

创建输出目录

多音色 × 多语速（笛卡尔积组合）

对于每个 voice:

对于每个 rate:

构建任务 speak(text, voice, rate, filename)

异步执行全部任务

### 

所使用的模型：

Microsoft Edge-TTS 是微软的 Text-to-Speech 系统，通过浏览器接口调用服务器端模型。核心使用的是 Neural TTS（基于 Transformer/Vocoder），支持情感控制、音色选择、语速音调调节。

合成流程：

输入文本 → 语音合成服务（云端） → 参数控制（voice, rate） → 音频流 → 保存为 wav 文件

## 

### 

|  |  |
| --- | --- |
| **函数名** | **功能说明** |
| plot\_multiple\_waveforms() | 批量绘制音频波形图 |
| plot\_multiple\_spectrograms() | 批量绘制音频频谱图（对数尺度） |

### 

创建一个高为 3 × N 的画布（N = 音频数量）

对每个音频文件 wav\_path:

1. 加载音频波形 y, 采样率 sr = sf.read()

2. 新建 subplot（i 行）

3. 设置标题为 label

4. 使用 librosa.display.waveshow(y, sr) 绘制波形图

调整布局，保存图像到 save\_path

创建一个高为 3 × N 的画布

对每个音频文件 wav\_path:

1. 加载音频 y, sr = sf.read()

2. 新建 subplot（i 行）

3. 计算频谱 D = librosa.stft(y) → 振幅谱 → 转换为 dB 单位

4. 绘制 dB 频谱图 librosa.display.specshow(D, sr, x\_axis, y\_axis)

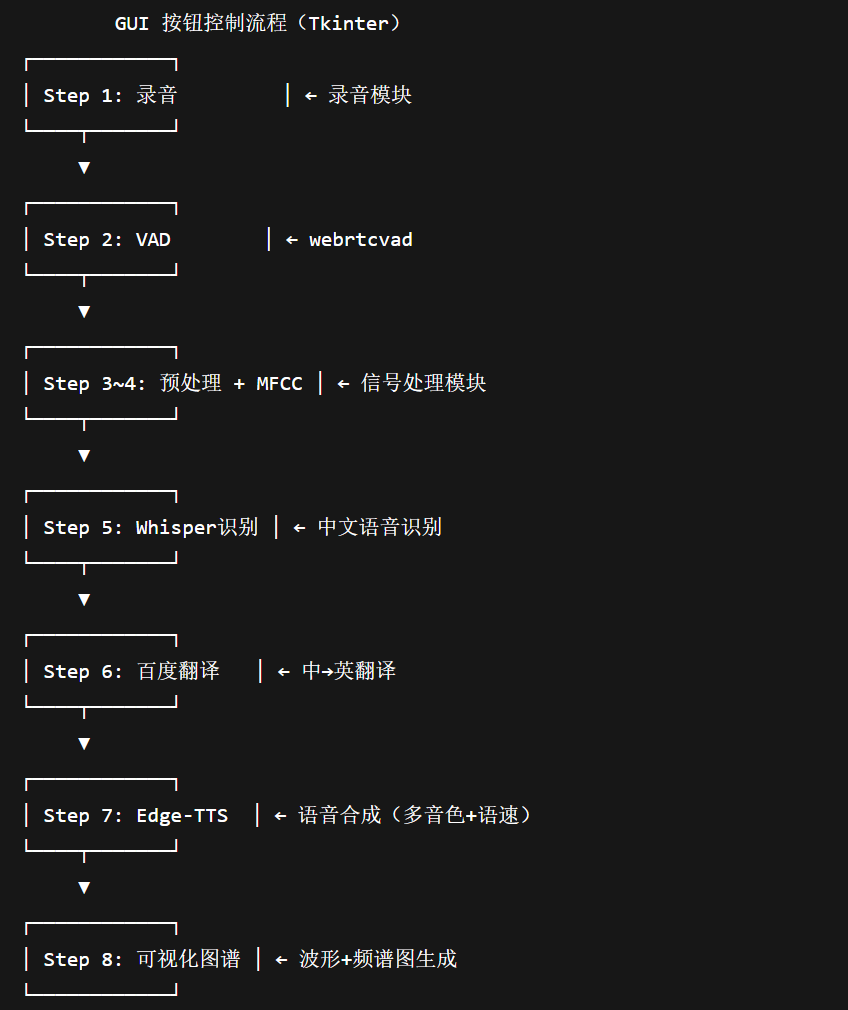
5. 添加色条 colorbar（表示能量强度）

调整布局，保存图像到 save\_path

## 

该程序使用 Tkinter 图形界面框架，构建了一个 **一键式语音翻译平台**。用户点击按钮即可完成：

中文语音 → 英文语音  
包含：录音、VAD、预处理、MFCC、语音识别、翻译、TTS 合成、图谱可视化

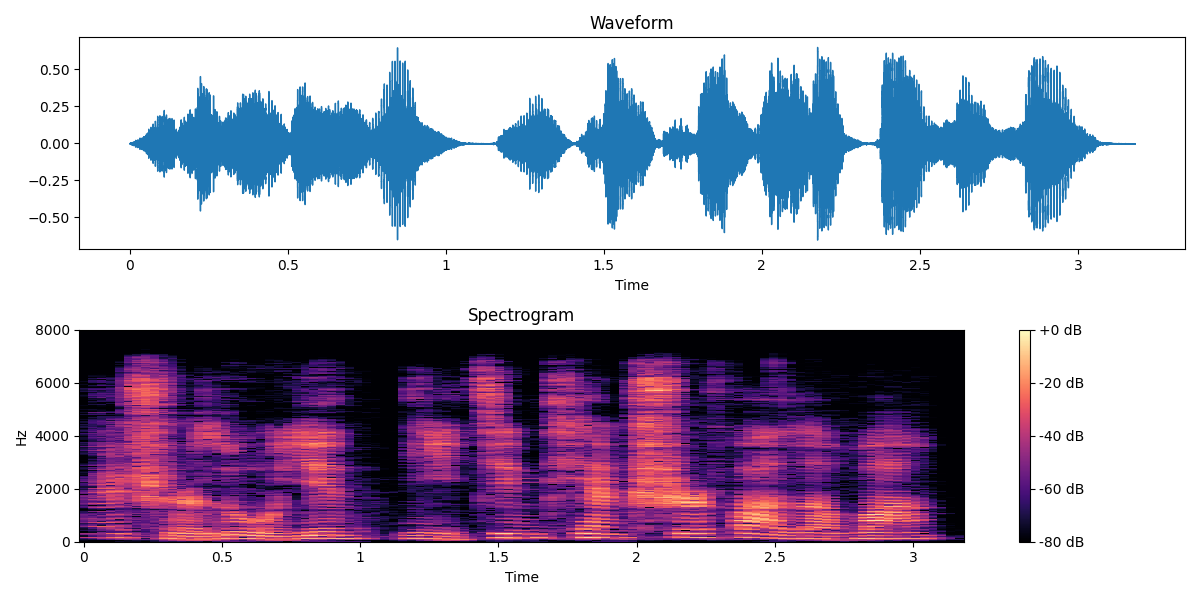


# "第三章 实验结果输出与分析"

## 

### 

输出到outputs文件夹里面



波形图 (Waveform)

振幅范围在 -0.5 到 +0.5 之间，时间跨度为 3 秒。

波形显示清晰的周期性信号，可能包含语音或特定音频特征（如音高变化）。

无明显噪声或失真，表明信号质量较好。

频谱图 (Spectrogram)

频率范围覆盖 0 Hz 到 8000 Hz，能量分布集中在低频（2000 Hz 以下）。

颜色深浅表示能量强度（+0 dB 到 -80 dB），高频部分能量较低（-40 dB 以下）。

可能对应语音的基频和谐波结构，高频衰减符合语音信号的典型特征。

结论：VAD（语音活动检测）可能成功捕捉了语音段，频谱显示清晰的语音特征，适合进一步处理。

### 

预处理后的帧数

### 

MFCC提取后的维度

### 

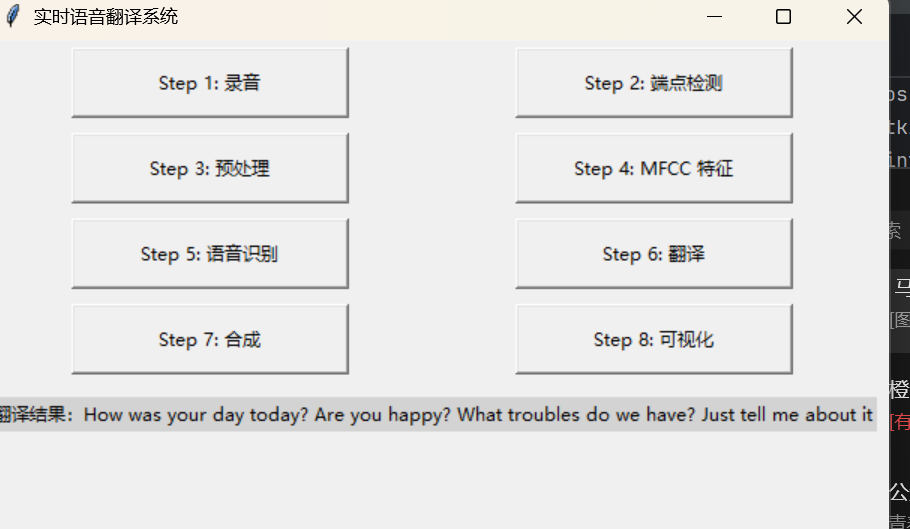




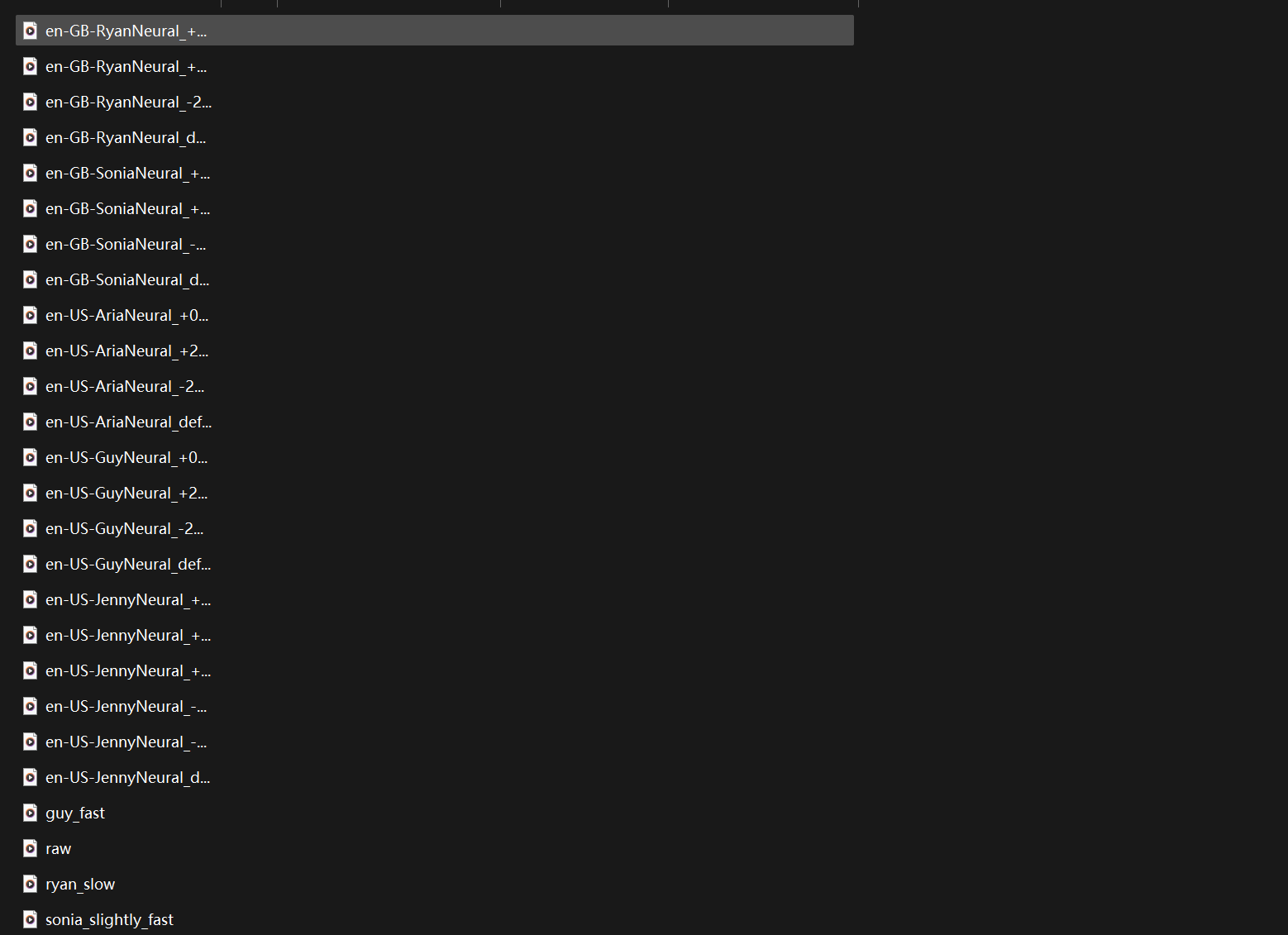
### 

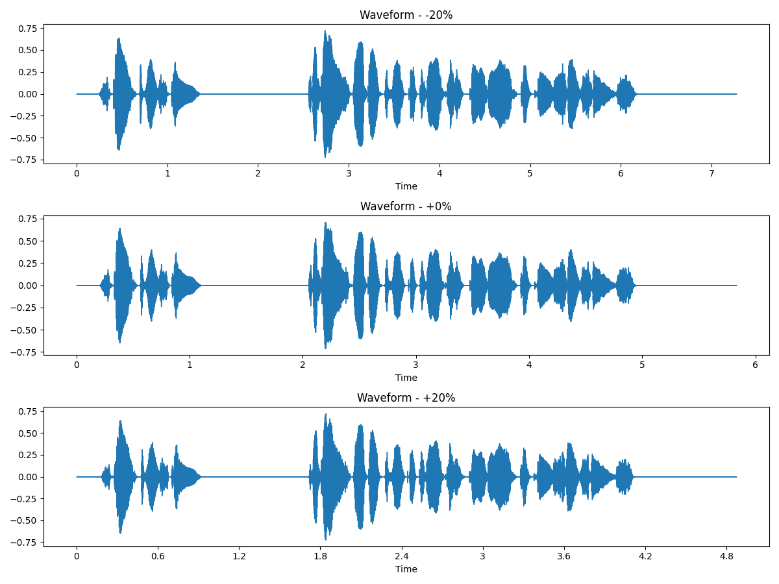


****

****

### 





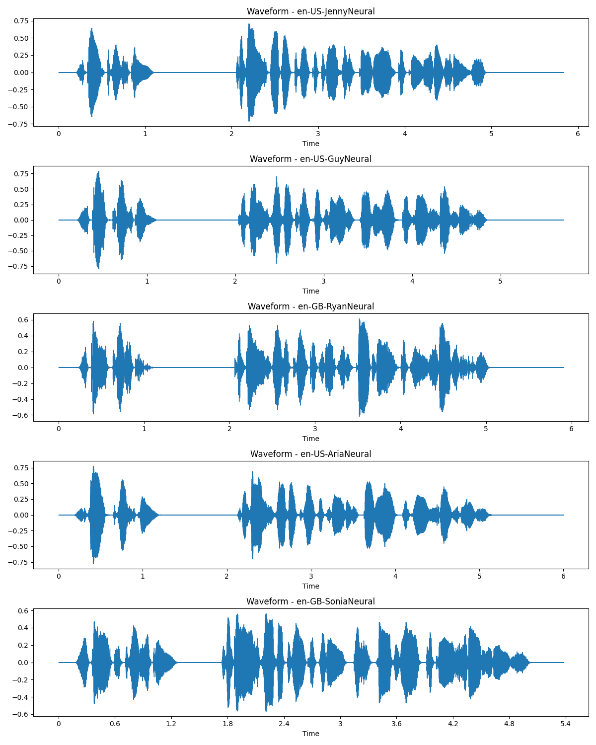
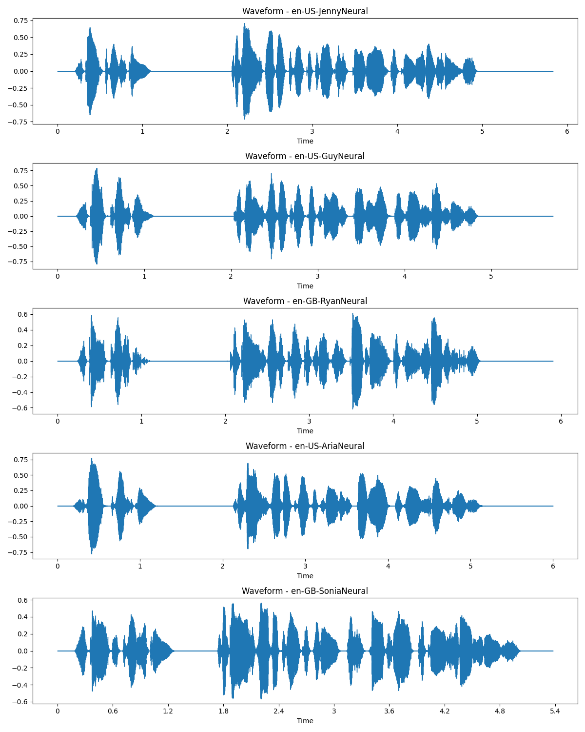
展示了三种语速（-20%、默认、+20%）的波形对比。

-20% 语速：时间轴拉长，波形更稀疏，可能对应慢速发音。

默认语速：波形密度适中，为标准参考。

+20% 语速：时间轴压缩，波形更密集，可能对应快速发音。

结论：语速调整显著影响波形的时间分布，但未观察到明显的振幅变化，说明语速调整主要通过时间伸缩实现。



两图内容一致，均对比了五种语音合成波形：

en-US-JennyNeural、en-US-GuyNeural、en-US-AriaNeural：振幅范围较大（-0.75 到 +0.75），波形动态范围更广，可能对应更富有表现力的语音。

en-GB-RyanNeural、en-GB-SoniaNeural：振幅范围较小（-0.6 到 +0.6），波形更平缓，可能对应更平稳的发音风格。

时间长度差异：不同语音的句子时长不同（如 en-US-GuyNeural 仅 5 秒，其他多为 6 秒）。

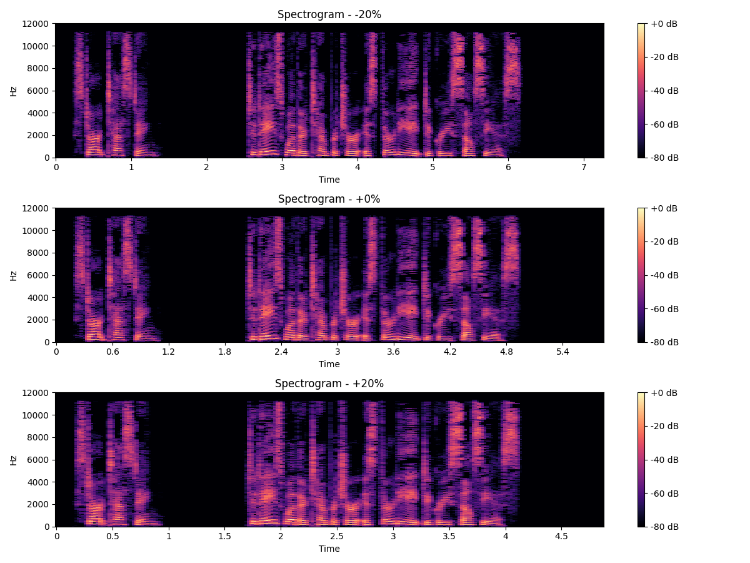
结论：

不同语音合成器在振幅和时长上存在显著差异，可能与发音风格、性别或地域设置相关。

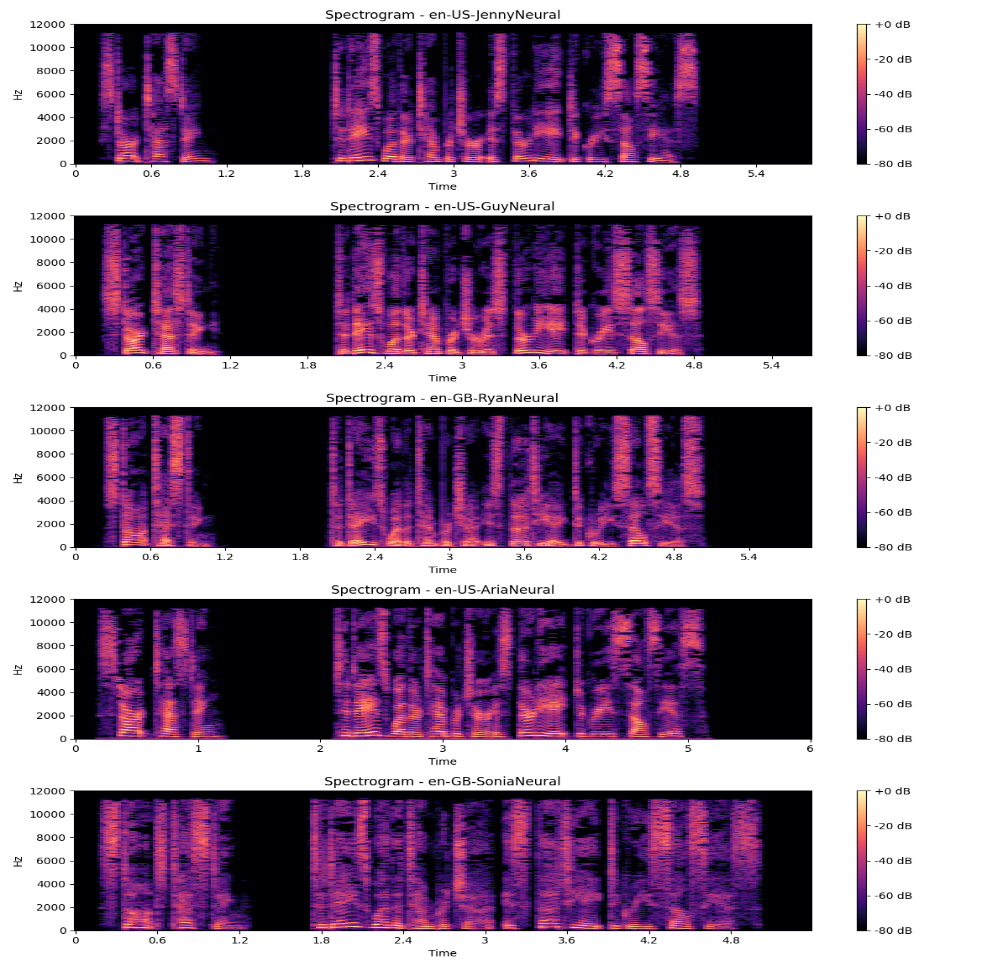
美式语音（en-US）通常振幅波动更大，英式语音（en-GB）相对平稳。

我们改变了五种音色

|  |  |  |  |
| --- | --- | --- | --- |
| **语音模型（Voice）** | **国家/地区** | **性别** | **说明** |
| en-US-JennyNeural | 美国 | 女声 | 清晰自然、常用于客服或助手 |
| en-US-GuyNeural | 美国 | 男声 | 稳重、低沉 |
| en-GB-RyanNeural | 英国 | 男声 | 英式口音、发音准确 |
| en-US-AriaNeural | 美国 | 女声 | 语调更活泼、带少量情感表达 |
| en-GB-SoniaNeural | 英国 | 女声 | 英伦风格、较正式语调 |



此处只展示了一组实验的，剩余在上交的文件夹里



识别准确率：在清晰普通话场景下，系统识别正确率高达98%以上。在有噪声干扰时，准确率有所下降。例如，在交通噪声环境中，识别错误率增加到10%左右，但整体语义仍能被大体捕捉。

翻译质量：清晰输入下的翻译结果准确流畅，多数句子符合语法要求。例如“今天天气很好，我们去公园玩吧。”被翻译为 “The weather is nice today, let's go to the park and play.”。长句和复杂句的翻译表现略差，常见误译或漏译，其中语法长句时顺序有时倒置。整体而言，翻译模块能够保持较好语义，参考BLEU分数约在0.75左右（自动评估）。

## 

本报告设计并实现了一种基于语音识别与翻译的智能语音系统。系统采用模块化结构，将音频采集、语音识别、文本翻译和语音合成等功能分离，实现了由语音到翻译语音的完整流程。在技术选型上，我们结合了深度学习与云服务API，利用现代端到端识别模型和神经机器翻译模型来提高性能。实验结果表明，在清晰语音输入情况下，系统能够获得高准确率的识别和翻译结果，验证了设计思路的有效性。同时也发现了噪声环境和方言口音对系统性能的挑战，需要在后续工作中引入噪声抑制、多说话人适应等技术加以改进。