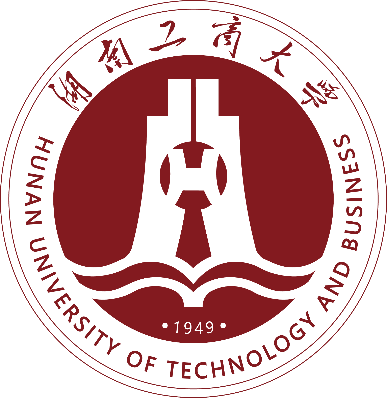
****

**湖南工商大学**

**《数字语音处理》课程考核报告**

**题　目： 基于**PaddleSpeech**的中文语音识别与合成系统实现**

|  |  |
| --- | --- |
| **姓 名:** | **孙叶倩 郭玉涵 吴霆宇** |
| **学 号:** | **2102010122**   |  |  | | --- | --- | |  | **2109020164 2123020010** | |
| **专 业­­­­­­­­­­­­­­­­­­­:** | **数字媒体与人工智能** |
| **班 级:** | **智媒体2101、2102** |
| **指导教师:** | **张军号** |
| **职 称­­­­­­­­­­­­­­­­­­­:** | **副教授** |

**前沿交叉学院**

**2023年6月**

**课程报告评审表**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 等级  成绩  组成 | 优秀 | 良好 | 中等 | 及格 | 不及格 |
| 程序演示（50分） | 完成语音识别功能15分，完成语音合成功能15分，性能提升各10分。 | | | | |
|  |  |  |  |  |
| 报  告  撰  写  （50分） | 内容详实（10分）；  算法模型描述清晰（20分）；  代码规范，有注释（10分）；  报告格式排版必须规范（10分）； | | | | |
|  |  |  |  |  |
| 综合成绩评定： 评阅老师（签章）：    年 月 日 | | | | | |

**目录**

[1 引言 1](#_Toc75534273)

[1.1 项目背景与需求 1](#_Toc341690934)

[1.1.1 项目背景 1](#_Toc428255660)

[2 系统环境配置 5](#_Toc1475176523)

[2.1 系统环境要求 5](#_Toc593117446)

[2.1.1 操作系统 5](#_Toc2053309195)

[2.1.2 Python版本 5](#_Toc2052916722)

[2.1.3 硬件设备 5](#_Toc1899073952)

[2.2 安装与配置 Python 虚拟环境 5](#_Toc1833949550)

[2.3 安装 PaddleSpeech 6](#_Toc357301459)

[3 模型介绍 7](#_Toc801344401)

[3.1 语音识别模型：DeepSpeech2 7](#_Toc1325397270)

[3.2 语音合成模型：TTS (Text-to-Speech) 8](#_Toc104046559)

[4 数据准备 9](#_Toc658828455)

[4.1 语音数据集介绍 9](#_Toc504159253)

[4.1.1 AIShell数据集 9](#_Toc1581577756)

[4.1.2 CSMSC数据集 10](#_Toc24762526)

[4.1.3 成员采样数据集 10](#_Toc1719430611)

[4.2 数据集下载与处理 11](#_Toc1930325045)

[4.2.1 语音识别-AlShell数据集 11](#_Toc937576086)

[4.2.2 语音合成-CSMSC数据集 11](#_Toc1753759363)

[5 数据预处理 12](#_Toc1220558866)

[5.1生成清单文件（Manifest） 12](#_Toc1169064718)

[5.2 特征提取与规范化 13](#_Toc1142829023)

[5.3构建词汇表 15](#_Toc433650793)

[5.4格式化清单文件 18](#_Toc1956863680)

[5.5数据预处理框架概述 20](#_Toc295815955)

[6 语音识别模型训练 22](#_Toc354112880)

[6.1 模型配置文件解读 22](#_Toc897988323)

[6.1.1 deepspeech2.yaml解读 22](#_Toc2122157192)

[6.1.2 chunk\_decode.yaml 解读 25](#_Toc1687516568)

[6.1.3 decode.yaml 解读 26](#_Toc274432447)

[6.2 语音识别模型训练 27](#_Toc1738746620)

[6.2.1 模型训练命令执行流程介绍 27](#_Toc156973964)

[6.2.2 模型训练命令流程 28](#_Toc1151494432)

[6.2.3 启动训练train.sh脚本 32](#_Toc44291860)

[6.2.4训练过程展示 35](#_Toc1383949158)

[6.3 模型优化与调优 37](#_Toc638117849)

[7 语音合成模型训练 37](#_Toc313355025)

[7.1 模型配置文件解读 37](#_Toc928002731)

[7.1.1 cnndecoder.yaml解读 37](#_Toc1915655403)

[7.1.2 conformer.yaml 解读 40](#_Toc1345522397)

[7.1.3 default.yaml 解读 41](#_Toc1192123469)

[7.2 语音识别模型训练 42](#_Toc2144200620)

[7.2.1 模型训练命令执行流程介绍 42](#_Toc656740033)

[7.2.2 模型训练命令脚本介绍 43](#_Toc1911272698)

[7.2.2 启动训练train.sh脚本 47](#_Toc699843460)

[8 模型评估与测试 48](#_Toc501097601)

[8.1 模型评估指标介绍 48](#_Toc1664000120)

[8.2 模型性能测试 50](#_Toc170481959)

[8.2.1语音识别性能测试 50](#_Toc547099815)

[8.2.2 语音合成性能测试 56](#_Toc1729097898)

[9 模型导出与部署 56](#_Toc1199660482)

[9.1 静态图与动态图介绍 56](#_Toc2117242938)

[9.1.1 静态图介绍 57](#_Toc698028176)

[9.1.2 动态图介绍 57](#_Toc56390471)

[9.1.3 静态图与动态图对比 58](#_Toc714357770)

[9.1.4 模型静态图与动态图导出转换意义 58](#_Toc1777453660)

[9.2 语音模型导出脚本阐述 58](#_Toc18650203)

[9.2.1 将模型导出为静态图 59](#_Toc2068833006)

[9.2.2 模型动态推理 60](#_Toc968603265)

[10 语音人机交互系统设计 61](#_Toc1389030595)

[10.1 语音交互系统架构整体设计逻辑 61](#_Toc142483628)

[10.1.1 Client-Server (C/S) 模式 62](#_Toc278069391)

[10.1.2 客户端设计 62](#_Toc587838665)

[10.1.2 服务器端设计 63](#_Toc1379666455)

[10.2 客户端运行流程阐述 63](#_Toc1673172526)

[10.3 服务器端运行流程阐述 67](#_Toc1859770664)

[10.4 总体运行流程阐述 70](#_Toc541067763)

[10.4.1 脚本阐述 70](#_Toc1280131343)

[10.4.2 流程运行展示 73](#_Toc1676306155)

[10.4.3 运行问题解决方案汇总 74](#_Toc839582092)

[10.5 总体效果展示 77](#_Toc1888659454)

[10.5.1 客户端页面展示 77](#_Toc743657071)

[10.5.2 后端展示 79](#_Toc289566757)

[11 系统交互结果分析 80](#_Toc550540797)

[11.1 语音识别项目结果 80](#_Toc1579623903)

[11.2 语音合成项目结果 81](#_Toc1546093507)

[11.2.1 效果展示 81](#_Toc641443449)

[11.2.2 合成音质与自然度方面 81](#_Toc372139403)

[12 总结与展望 81](#_Toc1074566157)

[12.1 项目总结 81](#_Toc2043413076)

[12.2 未来工作展望 82](#_Toc1085085508)

[参考文献 83](#_Toc601002632)

**基于PaddleSpeech 的中文语音识别与合成系统实现**

1. **引言**

## **项目背景与需求**

### **1.1.1 项目背景**

随着人工智能技术的蓬勃发展，中文语音识别与合成技术作为其重要分支，取得了显著的进步。这一进步主要得益于深度学习技术的不断成熟和完善，它极大地提升了语音识别与合成的准确性和自然度。这些技术现已广泛应用于智能客服、语音助手、自动翻译、教育辅助、智能家居控制等多个领域，极大地提高了人机交互的效率和便捷性。

在人机交互的领域中，语音识别技术扮演着将用户的语音指令转换为文本信息的桥梁角色，而语音合成技术则能将文本信息转化为流畅自然的语音输出，使得机器具备了“说话”的能力。这两种技术的相互融合，不仅为构建更加智能、自然的交互界面提供了强有力的技术支持，也进一步推动了人机交互技术的发展和创新。

对于中文这一使用人数众多的语言来说，其语音识别与合成技术的发展尤为重要。它不仅能够促进中文信息的广泛传播，还能提升中文用户的交互体验。然而，中文的语音特性，如音调的变化和同音字的存在等，也为中文语音识别与合成技术带来了一定的挑战。因此，需要更多的人去不断探索和创新，以克服这些挑战，推动中文语音识别与合成技术的进一步发展。

### **1.1.2 项目需求**

（1）中文语音识别模型构建

本项目将构建一个能够精准识别大于20个日常中文指令的语音识别模型。这些指令将涵盖诸如打开微信，打开浏览器搜索指定内容等日常用语，以确保模型在实际应用中的广泛适用性和准确性。

（2）中文语音合成模型研发

同时，本项目训练了一个能够将中文文本转换成流畅、自然语音的模型。所合成的语音将是项目人员自己采样音频并录入训练模型而成，以提供用户最佳的听觉体验。

## **目标与任务**

本项目的目标是开发训练深度学习模型，能够以高准确率识别大于20个常用的中文语音指令，实现对连续语音流的实时识别能力，以便于应用在动态交互场景中。同时训练一个语音合成模型，能够将中文文本转换成自然流畅的语音输出。合成语音应具有准确的语调、节奏和情感表达能力，以提供更自然的听觉体验。并且支持多种语音特性的调整，如语速、音量和语调，以满足不同场景的需求。

任务流程如下：

1. 数据收集：采集语音数据，并进行必要的清洗和预处理。
2. 特征提取：从语音信号中提取有用的特征，如梅尔频谱等。
3. 模型训练：使用提取的特征训练语音识别和合成模型。
4. 性能评估：对训练好的模型进行测试，评估识别准确率和合成自然度。
5. 系统整合：将语音识别和合成模块集成到统一的系统中。
6. 用户测试：进行实际场景下的用户测试，收集反馈并优化系统。

## **相关技术简介**

### **1.3.1 语音识别与合成的核心概念与主要技术**

语音识别基本原理及常用技术：

语音识别是以语音为研究对象，通过语音信号处理和模式识别让机器自动识别和理解人类口述的语言，其核心原理主要涉及信号处理、模式识别、概率论和信息论、发声机理和听觉机理等领域。

在语音识别的过程中，通常包括预处理、特征提取、模式匹配和后处理等步骤。预处理阶段主要是对输入的语音信号进行降噪、分帧等处理，以便提取出有效的特征。特征提取则是从语音信号中提取出反映其本质特征的信息，如梅尔频率倒谱系数（MFCC）等。模式匹配则是将提取出的特征与预先训练好的模型进行比对，以实现语音到文本的转换。后处理阶段则是对识别结果进行优化，以提高识别准确率。

语音合成基本原理及常用技术：

语音合成技术则是将文本转化为语音信号的技术。其核心原理主要涉及语言学、语音学和信号处理等领域。语音合成技术的目标在于产生自然、清晰的语音，以实现人与机器的自然交互。

在语音合成的实现过程中，通常包括文本分析、韵律建模和波形合成等步骤。文本分析主要是对输入的文本进行词法、句法、语义等方面的分析，以便正确理解其含义。韵律建模则是根据文本的情感、语速、语调等因素，对其发音的节奏、音高进行建模。波形合成则是根据韵律模型和声学模型，生成模拟人类语音的波形数据[2]。

### **1.3.2 PaddlePaddle框架及其特点**

PaddlePaddle是百度推出的深度学习平台，其不仅包含深度学习框架，还提供了一整套紧密关联、灵活组合的工具组件和服务平台，助力深度学习技术的应用落地。飞桨框架除了支持用户编写深度学习模型外，还对于自然语言处理、计算机视觉、语音和推荐有大量的预训练模型，可以通过PaddleHub工具直接使用。具有以下特点：

全栈支持：提供从数据准备到模型部署的完整流程。

易用性：拥有简洁的API和详尽的文档，便于模型开发和调试。

高性能：对多种硬件平台进行优化，支持多种计算设备。

灵活性：支持自定义算子和层，满足复杂模型结构和算法创新的需求。

产业级应用：基于百度的大规模应用实践，具备处理大规模数据和高并发请求的能力。

### **1.3.3 PaddleSpeech框架及其特色**

PaddleSpeech是PaddlePaddle生态下的一款开源语音识别工具包，支持多种语音识别任务，包括自动语音识别（ASR）、文本转语音（TTS）等。PaddleSpeech基于PaddlePaddle深度学习框架，提供了丰富的模型库和高效的计算性能，使得语音识别技术更加易于使用和部署。具有以下特点：

模型多样性：集成了多种先进的语音识别和合成模型。

端到端解决方案：提供从数据预处理到模型训练、评估、导出的完整流程。

易用性：简化了模型配置和训练流程，支持快速模型训练和测试。

高度可扩展性：支持用户自定义数据集和模型结构，便于个性化开发。

社区支持：拥有活跃的开发者社区，提供丰富的教程和案例，促进用户学习和交流。

1. **系统环境配置**

为了确保基于PaddleSpeech的中文语音识别与合成系统能够顺利开发和运行，此处将详细介绍所需的系统环境配置。

## **系统环境要求**

### **2.1.1 操作系统**

本项目选择macOS作为开发和运行的主要操作系统。macOS提供了一个稳定、安全且易于管理的开发环境，适合进行深度学习和其他计算密集型任务。

### **2.1.2 Python版本**

项目依赖Python编程语言，需要大于Python3.8以上的版本。本项目使用的是Python3.9版本，此版本包含了许多用于科学计算和数据处理的改进，同时确保了与PaddlePaddle和PaddleSpeech框架的兼容性。

### **2.1.3 硬件设备**

本项目使用macOS Sonoma 14.3.1配合Apple M2 Pro芯片和32 GB内存为中文语音识别与合成系统项目提供了一个强大、稳定和高效的开发环境。这种配置将支持开发过程中遇到的各种技术挑战，并有助于实现高质量的语音识别和合成结果。

## **安装与配置 Python 虚拟环境**

在进行Python项目开发时，使用虚拟环境可以有效地隔离项目依赖。在本项目中为了避免不同模块间的冲突。创建并激活两个独立的Python虚拟环境，一个用于语音合成（命名为padpy\_9），另一个用于语音识别（命名为paddle-env）。此处以创建padpy\_9虚拟环境，并激活为例：

1. 创建虚拟环境：



图2-1 创建padpy\_9虚拟环境

1. 激活虚拟环境：

2

图2-2 创建padpy\_9虚拟环境

## **安装 PaddleSpeech**

PaddleSpeech基于PaddlePaddle框架，因此需要先安装PaddlePaddle。本次项目是通过使用以下命令安装CPU版本的PaddlePaddle：

3

图2-3 安装CPU版本的PaddlePaddle

安装完成后，通过以下命令检查paddle是否安装成功

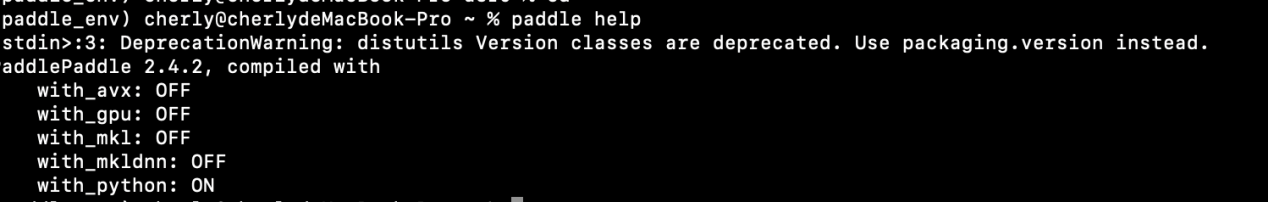


图2-4 命令检查paddle是否安装成功

安装PaddlePaddle之后，可以通过pip安装PaddleSpeech：

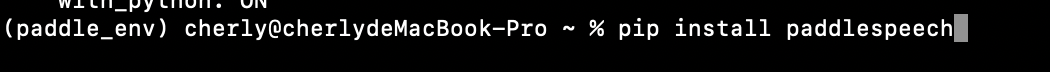


图2-5 安装PaddleSpeech

安装完成后，可以通过运行以下命令来验证PaddleSpeech是否正确安装：

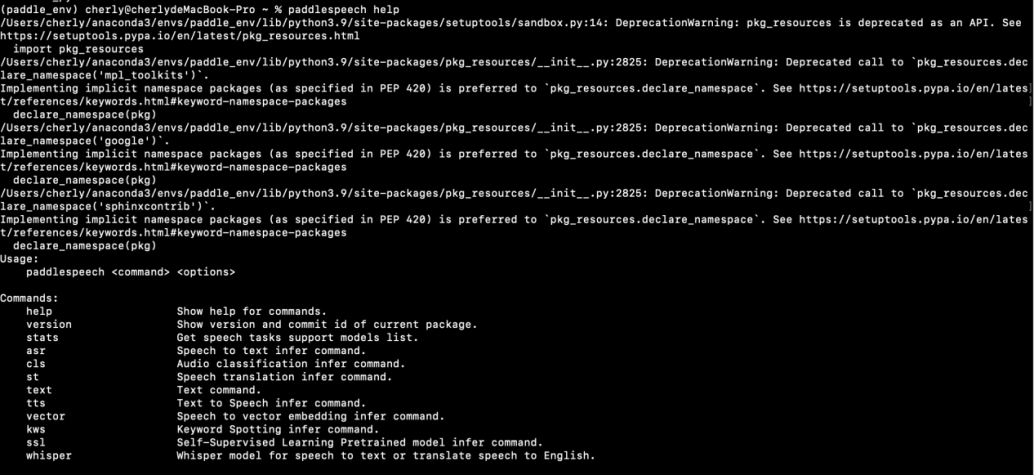


图2-6 验证PaddleSpeech是否正确安装

整体环境测试：

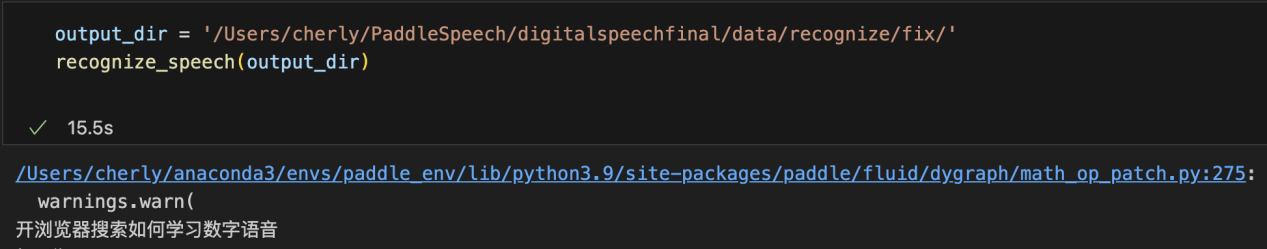


图2-7 整体环境测试

1. **模型介绍**

## **语音识别模型：DeepSpeech2**

DeepSpeech2 是一个基于深度学习的端到端语音识别系统，由Baidu Research团队开发。它使用循环神经网络(RNN)和连接时序分类(CTC)损失函数来实现从音频到文本的转换。

Deepspeech2 模型，其主要分为3个部分：

1. 特征提取模块

此处使用 linear 特征，也就是将音频信息由时域转到频域后的信息。

1. Encoder

Encoder 主要采用了 2 层降采样的 CNN（subsampling Convolution layer）和多层 RNN（Recurrent Neural Network）层组成。其中降采样的 CNN 主要用途在提取局部特征，减少模型输入的帧数，降低计算量，并易于模型收敛。

1. CTC Decoder

采用了 CTC 损失函数训练；使用CTC损失函数来训练模型，允许模型在没有明确时间对齐的情况下学习音频和文本之间的对应关系。

下述图3-1 为Deepspeech2 模型结构

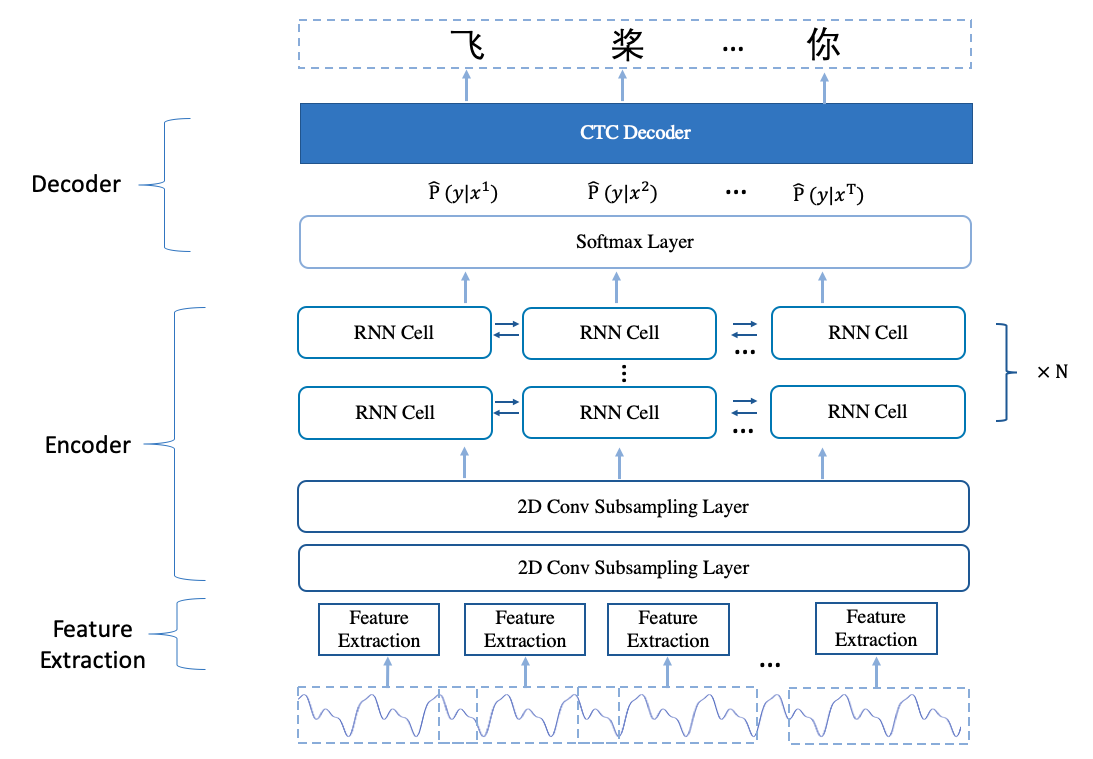


图3-1 Deepspeech2 模型结构

DeepSpeech2的训练和推理过程通常包括以下步骤：

特征提取：从原始音频中提取特征，如梅尔频率倒谱系数(MFCC)。

模型训练：使用大量的标注数据训练模型，直到收敛。

解码：在推理阶段，使用CTC解码器将模型输出转换为文本序列。

## **语音合成模型：TTS (Text-to-Speech)**

语音合成，通常称为文本到语音(TTS)，是将文本信息转换为口语化的语音的过程。TTS模型可由三个核心部分组成，分别是文本处理(Text Analysis)，声学模型(Acoustic Model)和声码器(Vocoder)。具体流程如下图所示：

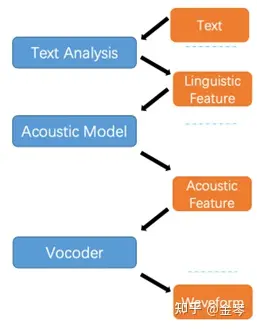


图3-2 TTS模型具体流程

文本处理模块接收文本输入，输出语言学特征至声学模型，这里的语言学特征可以是音素(phoneme)。声学模型负责将音素序列转换为声学特征，声学特征可以是梅尔谱(Mel-Spectrum)。最终由声码器将声学特征转换为可听音频信号。常见的声码器包括基于WaveNet的Parallel WaveGAN和其他基于神经网络的声码器。

1. **数据准备**

## **语音数据集介绍**

本项目在构建语音识别和语音合成功能时，分别使用了AIShell数据集和CSMSC数据集以及本项目设计人员音频生成的数据集。

### **4.1.1 AIShell数据集**

AIShell数据集是一个专为中文语音识别研究而设计的大规模、高质量语音数据库，其录音均在受控的录音室环境中进行，它包含了数千小时的录音，确保了声音的清晰度和减少背景噪声的干扰，能够为语音识别算法提供理想的输入。数据集中的录音按照严格的标准进行分段和标注，每个录音片段都与相应的文本脚本精确对应，包括标点符号和语调信息，能够支持更为精细的语音识别研究。由来自中国各地、不同年龄和性别的说话人所提供。这些录音覆盖了多样化的领域，如新闻、故事、指令等，以单声道和多声道的形式记录，确保了数据的丰富性和多样性。每个录音样本都配有精确的文本转录。



图4-1 AISHELL数据集介绍图

### **4.1.2 CSMSC数据集**

CSMSC（Chinese Standard Mandarin Speech Corpus）数据集是一个由中国科学院自动化研究所精心构建的普通话语音识别研究数据库，它收录了来自中国不同地区、具有不同年龄和性别背景的母语者提供的数千小时高质量语音录音。这些录音内容涵盖了日常对话、标准朗读、新闻广播等多种场景，每段录音都配有精确的文本转录和详细的说话人信息，包括口音、教育背景等，为语音识别、说话人识别、情感分析等研究领域提供了一个丰富、多样化且标注详尽的数据资源。CSMSC数据集以其卓越的录音品质、广泛的应用范围和开放的研究许可，已成为推动中文语音技术发展的关键工具，促进了语音识别系统性能的提升和语音处理技术的创新。



图4-2 csmcs数据介绍图

### **4.1.3 成员采样数据集**

本项目在选择数据时还采集了自己录制的音频，其中音频指令有“创建txt文件”、“打开记事本”、“打开浏览器”、“打开qq.mp3”、“微信发送消息”。数据架构如下：



图4-3自收集数据

## **数据集下载与处理**

下述为项目为训练模型而采用的数据集，并且给出了部分数据集的前期工作准备与处理。

### **4.2.1 语音识别-AlShell数据集**

语音识别模块数据集准备：

AIShell数据集下载链接：<https://openslr.elda.org/resources/33/>

### **4.2.2 语音合成-CSMSC数据集**

语音合成模块数据集准备：

Step1：CSMSC数据集下载链接：<https://test.data-baker.com/data/index/TNtts/>

Step2：使用[MFA](https://github.com/MontrealCorpusTools/Montreal-Forced-Aligner

获得 fastspeech2 的音素持续时间

Step3：生成Manifest文件，具体介绍在本文数据预处理部分详细展开

1. **数据预处理**

## **生成清单文件（Manifest）**

收集音频文件和对应的文本转录，编写脚本遍历音频文件并生成包含所需元数据的JSON对象。清单文件是一个文本文件，每行记录一个音频片段的元数据，这些元数据以JSON格式组织，包括：

• audio\_filepath：音频文件的路径。

• duration：音频片段的持续时间。

• text：音频片段的文本转录。

示例：

{"audio\_filepath":"/home/work/.cache/paddle/Libri/134686/1089-134686-0001.flac", "duration": 3.275, "text": "stuff it into you his belly counselled him"}

表5-1 生成清单文件脚本执行命令

|  |
| --- |
| 生成清单文件脚本执行命令 |
| python script.py --json-file data.json --manifest-file output\_manifest.txt  #script.py是脚本文件的名称，data.json是ESPnet数据JSON文件的路径，output\_manifest.txt是生成的清单文件的路径 |

下表为清单文件生成代码

表5-2 espnet\_json\_to\_manifest.py

|  |
| --- |
| espnet\_json\_to\_manifest.py |
| #!/usr/bin/env python  import argparse  import json  def main(args):  with open(args.json\_file, 'r') as fin:  data\_json = json.load(fin)  with open(args.manifest\_file, 'w') as fout:  for key, value in data\_json['utts'].items():  value['utt'] = key  fout.write(json.dumps(value, ensure\_ascii=False))  fout.write("\n")  if \_\_name\_\_ == '\_\_main\_\_':  parser = argparse.ArgumentParser(description=\_\_doc\_\_)  parser.add\_argument(  '--json-file', type=str, default=None, help="espnet data json file.")  parser.add\_argument(  '--manifest-file',  type=str,  default='manifest.train',  help='manifest data json line file.')  args = parser.parse\_args()  main(args) |

## **特征提取与规范化**

Z-score标准化是一种特征缩放技术，将特征转换为均值为0，标准差为1的标准正态分布。

为了进行有效的特征规范化，本项目首先需要计算训练数据集上语音特征的均值和标准差。这些统计数据是实现Z-score标准化（零均值和单位标准差）的基础，能够提高模型的泛化能力和收敛速度。

使用以下命令计算音频特征的均值和标准差：

表5-3特征提取与规范化

|  |
| --- |
| 特征提取与规范化执行命令 |
| python3 utils/compute\_mean\_std.py \  --num\_samples 2000 \  --spectrum\_type linear \  --manifest\_path path/to/manifest.train \  --output\_path path/to/mean\_std.npz |

执行这个命令将从训练清单文件中随机选择2000个样本，计算它们的功率谱特征的均值和标准差，并保存结果

其中compute\_mean\_std.py代码如下所需示：

表5-4 compute\_mean\_std.py

|  |
| --- |
| compute\_mean\_std.py |
| def compute\_cmvn(manifest\_path="data/librispeech/manifest.train",  output\_path="data/librispeech/mean\_std.npz",  num\_samples=2000,  num\_workers=0,  spectrum\_type="linear",  feat\_dim=13,  delta\_delta=False,  stride\_ms=10,  window\_ms=20,  sample\_rate=16000,  use\_dB\_normalization=True,  target\_dB=-20):  augmentation\_pipeline = AugmentationPipeline('{}')  audio\_featurizer = AudioFeaturizer(  spectrum\_type=spectrum\_type,  feat\_dim=feat\_dim,  delta\_delta=delta\_delta,  stride\_ms=float(stride\_ms),  window\_ms=float(window\_ms),  n\_fft=None,  max\_freq=None,  target\_sample\_rate=sample\_rate,  use\_dB\_normalization=use\_dB\_normalization,  target\_dB=target\_dB,  dither=0.0)  def augment\_and\_featurize(audio\_segment):  augmentation\_pipeline.transform\_audio(audio\_segment)  return audio\_featurizer.featurize(audio\_segment)  normalizer = FeatureNormalizer(  mean\_std\_filepath=None,  manifest\_path=manifest\_path,  featurize\_func=augment\_and\_featurize,  num\_samples=num\_samples,  num\_workers=num\_workers)  normalizer.write\_to\_file(output\_path)  def define\_argparse():  parser = argparse.ArgumentParser(description=\_\_doc\_\_)  add\_arg = functools.partial(add\_arguments, argparser=parser)  # yapf: disable  add\_arg('manifest\_path', str,  'data/librispeech/manifest.train',  "Filepath of manifest to compute normalizer's mean and stddev.")  add\_arg('output\_path', str,  'data/librispeech/mean\_std.npz',  "Filepath of write mean and stddev to (.npz).")  add\_arg('num\_samples', int, 2000, "# of samples to for statistics.")  add\_arg('num\_workers',  default=0,  type=int,  help='num of subprocess workers for processing')  add\_arg('spectrum\_type', str,  'linear',  "Audio feature type. Options: linear, mfcc, fbank.",  choices=['linear', 'mfcc', 'fbank'])  add\_arg('feat\_dim', int, 13, "Audio feature dim.")  add\_arg('delta\_delta', bool, False, "Audio feature with delta delta.")  add\_arg('stride\_ms', int, 10, "stride length in ms.")  add\_arg('window\_ms', int, 20, "stride length in ms.")  add\_arg('sample\_rate', int, 16000, "target sample rate.")  add\_arg('use\_dB\_normalization', bool, True, "do dB normalization.")  add\_arg('target\_dB', int, -20, "target dB.")  # yapf: disable  args = parser.parse\_args()  return args  def main():  args = define\_argparse()  print\_arguments(args, globals())  compute\_cmvn(\*\*vars(args))  if \_\_name\_\_ == '\_\_main\_\_':  main() |

处理过程在终端显示，如下图所示：



图 5-1 数据预处理过程

## **构建词汇表**

为了将文本转录转换为模型训练的输入特征并且在模型推理时将预测的标记索引转换回可读文本。本项目将构建词汇表。

词汇表是一个字符集合，用于将文本转录转换为模型训练所需的标记索引列表，解码时再将索引转换回文本

使用以下命令将根据训练清单文件中的所有文本转录构建一个完整的词汇表，并保存到指定路径

表5-5 词汇表构建命令

|  |
| --- |
| 词汇表构建命令 |
| python3 utils/build\_vocab.py \  --count\_threshold 0 \  --vocab\_path path/to/vocab.txt \  --manifest\_paths path/to/manifest.train |

下面为build\_vocab.py具体函数介绍

count\_manifest：统计清单文件中文本数据的字符或词汇使用频率。

dump\_text\_manifest：从清单文件中转储文本数据到一个文件对象。

build\_vocab：构建词汇表的主要函数。

内部逻辑：

打开或创建词汇表文件，并写入一些特殊标记，如BLANK、UNK、SOS。根据unit\_type决定是使用字符级别还是词汇级别或SentencePiece模型进行词汇表构建。使用SentencePiece进行词汇切分时，会训练SentencePiece模型并使用该模型编码文本数据。统计文本数据中的字符或词汇使用频率，并根据阈值过滤。将统计结果排序并写入到词汇表文件。

表5-6 词汇表构建命令代码

|  |
| --- |
| build\_vocab.py主要函数代码 |
| def count\_manifest(counter, text\_feature, manifest\_path):  manifest\_jsons = []  with jsonlines.open(manifest\_path, 'r') as reader:  for json\_data in reader:  manifest\_jsons.append(json\_data)  for line\_json in manifest\_jsons:  if isinstance(line\_json['text'], str):  tokens = text\_feature.tokenize(  line\_json['text'], replace\_space=False)  counter.update(tokens)  else:  assert isinstance(line\_json['text'], list)  for text in line\_json['text']:  tokens = text\_feature.tokenize(text, replace\_space=False)  counter.update(tokens)  def dump\_text\_manifest(fileobj, manifest\_path, key='text'):  manifest\_jsons = []  with jsonlines.open(manifest\_path, 'r') as reader:  for json\_data in reader:  manifest\_jsons.append(json\_data)  for line\_json in manifest\_jsons:  if isinstance(line\_json[key], str):  fileobj.write(line\_json[key] + "\n")  else:  assert isinstance(line\_json[key], list)  for line in line\_json[key]:  fileobj.write(line + "\n")  def build\_vocab(manifest\_paths="",  vocab\_path="examples/librispeech/data/vocab.txt",  unit\_type="char",  count\_threshold=0,  text\_keys='text',  spm\_mode="unigram",  spm\_vocab\_size=0,  spm\_model\_prefix="",  spm\_character\_coverage=0.9995):  manifest\_paths = [manifest\_paths] if isinstance(manifest\_paths,  str) else manifest\_paths  fout = open(vocab\_path, 'w', encoding='utf-8')  fout.write(BLANK + "\n") # 0 will be used for "blank" in CTC  fout.write(UNK + '\n') # <unk> must be 1  if unit\_type == 'spm':  # tools/spm\_train --input=$wave\_data/lang\_char/input.txt  # --vocab\_size=${nbpe} --model\_type=${bpemode}  # --model\_prefix=${bpemodel} --input\_sentence\_size=100000000  import sentencepiece as spm  fp = tempfile.NamedTemporaryFile(mode='w', delete=False)  for manifest\_path in manifest\_paths:  \_text\_keys = [text\_keys] if type(  text\_keys) is not list else text\_keys  for text\_key in \_text\_keys:  dump\_text\_manifest(fp, manifest\_path, key=text\_key)  fp.close()  # train  spm.SentencePieceTrainer.Train(  input=fp.name,  vocab\_size=spm\_vocab\_size,  model\_type=spm\_mode,  model\_prefix=spm\_model\_prefix,  input\_sentence\_size=100000000,  character\_coverage=spm\_character\_coverage)  os.unlink(fp.name)  # encode  text\_feature = TextFeaturizer(unit\_type, "", spm\_model\_prefix)  counter = Counter()  for manifest\_path in manifest\_paths:  count\_manifest(counter, text\_feature, manifest\_path)  count\_sorted = sorted(counter.items(), key=lambda x: x[1], reverse=True)  tokens = []  for token, count in count\_sorted:  if count < count\_threshold:  break  # replace space by `<space>`  token = SPACE if token == ' ' else token  tokens.append(token)  tokens = sorted(tokens)  for token in tokens:  fout.write(token + '\n')  fout.write(SOS + "\n") # <sos/eos>  fout.close() |

## **格式化清单文件**

本项目在数据处理的最后一部分将包含语音数据及其转录的清单文件（manifest file）解析并转换为一系列特定格式的文件，这些文件包括音频文件列表（wav.scp）、文本转录（text.word）、音节转录（text.syllable）和音素转录（text.phone），从而为语音识别系统的数据输入提供结构化的数据格式。通过命令行参数指定输入的清单文件路径和输出目录，脚本会读取清单文件中的每个条目，提取相关字段，并将其写入到相应的输出文件中，完成数据的格式化处理。

表5-7 格式化清单文件

|  |
| --- |
| 格式化清单文件执行命令 |
| python script.py --manifest-path path/to/manifest.jsonl --output-dir path/to/output |

该脚本使用jsonlines.open以读取模式打开清单文件，这是一个JSON线条格式的文件，每一行都是一个独立的JSON对象。通过迭代读取器，将所有JSON对象加载到列表manifest\_jsons中。之后从清单文件的第一行中获取所有可能的键，这些键代表不同的数据类型（如feat, text, syllable, phone）。如果键在key\_whitelist中，则根据filename字典创建对应的输出文件，并在file\_map中建立键与文件对象的映射。遍历清单文件中的每一行（即每个JSON对象）。对于每个JSON对象中的每个键值对，如果键在key\_whitelist中，则使用utt字段（通常是唯一的标识符）和对应的值构建一行文本，并写入到相应的文件中。每一行后面添加换行符。

具体代码如下

表5-8dump\_manifest.py

|  |
| --- |
| dump\_manifest.py |
| import argparse  from pathlib import Path  from typing import Union  import jsonlines  key\_whitelist = set(['feat', 'text', 'syllable', 'phone'])  filename = {  'text': 'text.word',  'syllable': 'text.syllable',  'phone': 'text.phone',  'feat': 'wav.scp',  }  def dump\_manifest(manifest\_path, output\_dir: Union[str, Path]):  output\_dir = Path(output\_dir).expanduser()  manifest\_path = Path(manifest\_path).expanduser()  with jsonlines.open(str(manifest\_path), 'r') as reader:  manifest\_jsons = list(reader)  first\_line = manifest\_jsons[0]  file\_map = {}  for k in first\_line.keys():  if k not in key\_whitelist:  continue  file\_map[k] = open(output\_dir / filename[k], 'w')  for line\_json in manifest\_jsons:  for k in line\_json.keys():  if k not in key\_whitelist:  continue  file\_map[k].write(line\_json['utt'] + ' ' + line\_json[k] + '\n')  for \_, file in file\_map.items():  file.close()  if \_\_name\_\_ == "\_\_main\_\_":  parser = argparse.ArgumentParser(  description="dump manifest to wav.scp text.word ...")  parser.add\_argument("--manifest-path", type=str, help="path to manifest")  parser.add\_argument(  "--output-dir",  type=str,  help="path to save outputs(audio and transcriptions)")  args = parser.parse\_args()  dump\_manifest(args.manifest\_path, args.output\_dir) |

## **数据预处理框架概述**

Run.sh脚本是用于训练、测试和导出本项目模型的流程脚本，在stage为0的阶段会执行data.sh脚本文件，此脚本实现了Aishell数据准备的完整流程。

执行以下命令运行该脚本

表5-9 data.sh运行

|  |
| --- |
| chmod +x prepare\_data.sh # 使脚本具有执行权限  ./prepare\_data.sh # 运行脚本 |

其主要流程如下：

1. 初始化阶段变量

stage：定义开始执行的阶段、stop\_stage：定义结束执行的阶段。

1. 设置字典目录

dict\_dir：设置存放字典文件的目录路径。

1. 导入解析选项脚本

导入parse\_options.sh脚本来解析命令行选项。

1. 创建必要的目录

创建data和dict\_dir等目录以存放数据和字典文件。

1. 下载数据和生成清单文件（Stage -1）

如果stage值小于等于-1，执行下载Aishell数据集并生成清单文件的操作。将生成的清单文件重命名为.raw后缀。

1. 计算特征的均值和标准差（Stage 0）

如果stage值小于等于0，执行计算训练数据特征的均值和标准差，用于特征归一化。

1. 构建词汇表（Stage 1）

如果stage值小于等于1，使用训练和验证数据集构建词汇表。

1. 格式化清单文件（Stage 2）

如果stage值小于等于2，对训练、验证和测试数据集的清单文件进行格式化，包括应用CMVN和转换文本为token IDs。

1. 错误检查

在每个主要步骤后，脚本会检查前一个命令的执行状态。如果命令失败（返回非零值），脚本将打印错误消息并终止执行。

1. 并行处理

在格式化清单文件阶段，脚本使用后台进程并行处理不同的数据集，并使用wait命令等待所有进程完成。

1. 脚本结束

如果所有步骤成功完成，脚本将打印“Data preparation done”并正常退出。

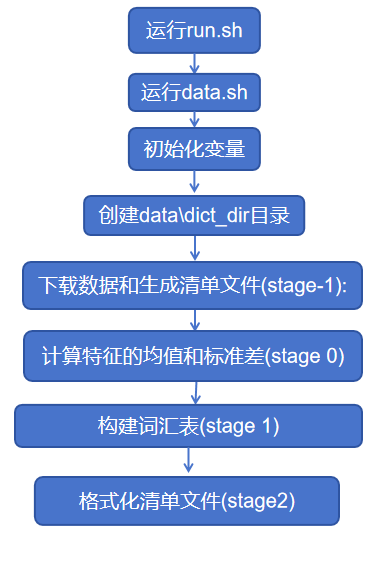


图5-3 data.sh框架流程图

通过上述处理得到如图5-2，5-3，5-4所示框架数据集列表：

|  |  |  |
| --- | --- | --- |
| 截屏2024-06-22 18.33.36  图5-3数据集展示1 | 截屏2024-06-22 18.34.00  图5-4数据集展示2 | 截屏2024-06-22 18.34.37  图5-5数据集展示3 |

1. **语音识别模型训练**

## **模型配置文件解读**

### **6.1.1 deepspeech2.yaml解读**

该文件是用来定义训练模型的参数的配置文件，分为一下四个大部分：

(1) 模型训练数据

定义训练中使用的数据文件和数据约束，如下图6-1所示：

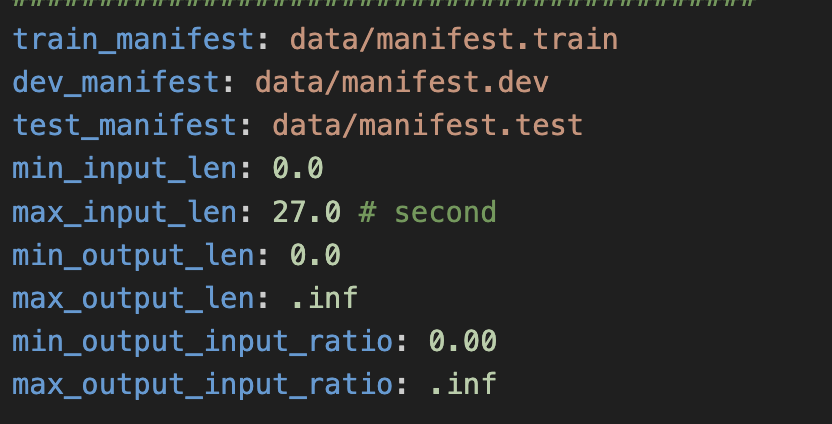


图 6-1 deepspeech.yaml\_data部分

其中参数解释如下：

• train\_manifest, dev\_manifest, test\_manifest：训练、开发和测试数据集的清单文件路径。这些文件列出音频文件及其对应的转录内容。

• min\_input\_len：输入音频序列的最小0.0秒

• max\_input\_len：27.0 表示输入最长音频为27秒。

• min\_output\_len, max\_output\_len：输出序列的最小和最大长度。.inf 表示没有上限。

• min\_output\_input\_ratio, max\_output\_input\_ratio：输出与输入序列长度的比例。这些设置为 .inf 意味着没有约束。

（2）数据加载器部分

该部分指定了数据的加载和处理方式，如下图5-2所示：

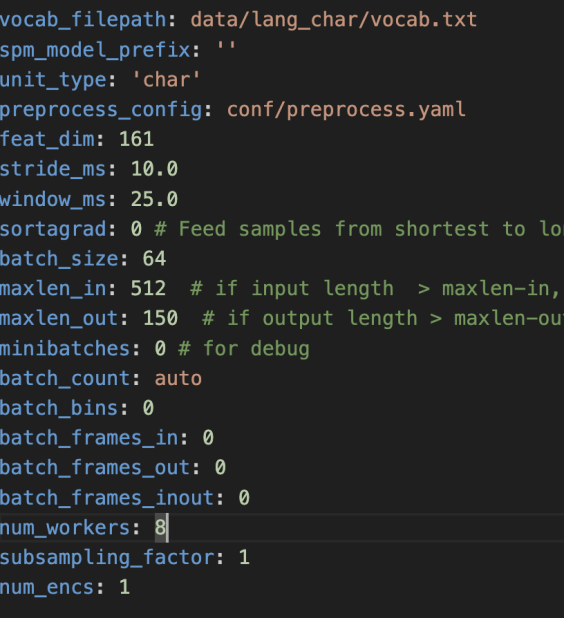


图 6-2 deepspeech.yaml\_dataloader部分

其中参数解释如下：

• vocab\_filepath：词汇表文件的路径，包含用于编码文本的字符或标记集。

• spm\_model\_prefix：SentencePiece模型的前缀。如果为空，表示不使用SentencePiece模型。

• unit\_type：使用的单元类型，例如 ‘char’ 表示字符。

• preprocess\_config：预处理配置文件的路径。

• feat\_dim：从音频中提取的特征维度，通常是梅尔频率倒谱系数（MFCC）或梅尔频谱的频带数量。

• stride\_ms：特征提取的步长大小（毫秒），例如 10ms。

• window\_ms：特征提取的窗口大小（毫秒），例如 25ms。

• sortagrad：是否按长度对训练样本进行排序。0 表示禁用，-1 表示对所有epoch启用，其他正数表示启用的初始epoch数量。

• batch\_size：每个批次的样本数量。

• maxlen\_in, maxlen\_out：输入和输出序列的最大长度。如果序列超过这些长度，批量大小将会减少。

• minibatches：用于调试，通常设置为 0。

• batch\_count：批量计数方法，通常设置为 ‘auto’。

• batch\_bins, batch\_frames\_in, batch\_frames\_out, batch\_frames\_inout：基于帧计数或其他标准的批量大小高级设置。

• num\_workers：数据加载的工作线程数。

• subsampling\_factor：输入序列的子采样因子。

• num\_encs：模型中使用的编码器数量。

（3）网络结构部分

该部分定义神经网络的结构,如下图5-3所示：

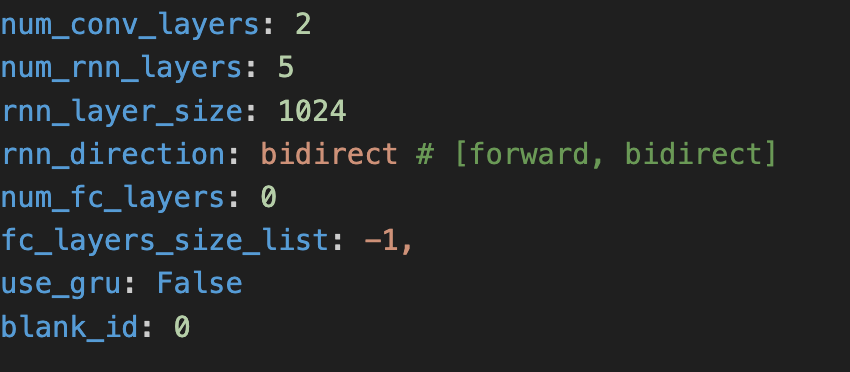


图 6-3 deepspeech2.yaml-network部分

其中参数解释如下：

• num\_conv\_layers：卷积层的数量。

• num\_rnn\_layers：循环层的数量。

• rnn\_layer\_size：每个RNN层的大小。

• rnn\_direction：RNN的方向，‘forward’ 表示前向，‘bidirect’ 表示双向。

• num\_fc\_layers：全连接层的数量。

• fc\_layers\_size\_list：全连接层的大小列表，-1 表示占位符。

• use\_gru：是否使用GRU（门控循环单元）代替LSTM。

• blank\_id：CTC（连接时序分类）中空白标记的ID。

（4）训练部分

该部分是定义训练模型的参数，如下图5-4所示：

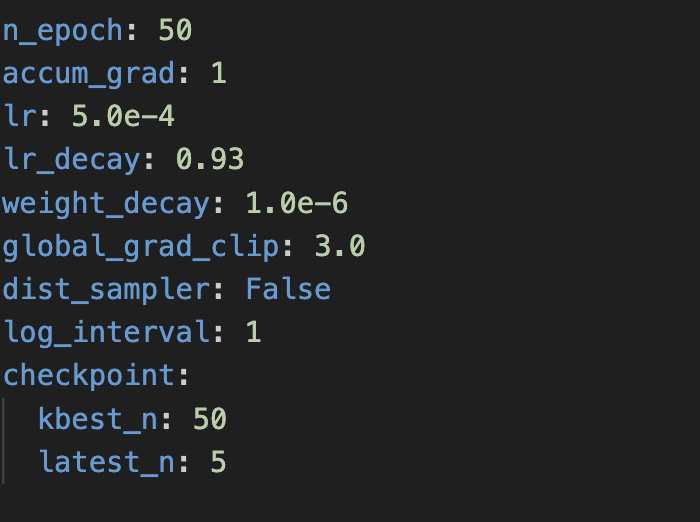


图 6-4 deepspeech2.yaml-training部分

其中参数解释如下：

• n\_epoch：训练的轮数。

• accum\_grad：梯度累积步数。

• lr：学习率，设置为 5.0e-4。

• lr\_decay：学习率衰减因子，设置为 0.93。

• weight\_decay：权重衰减（L2正则化），设置为 1.0e-6。

• global\_grad\_clip：梯度裁剪值，设置为 3.0。

• dist\_sampler：是否使用分布式采样，通常设置为 False。

• log\_interval：记录训练进度的间隔，设置为 1。

• checkpoint：

• kbest\_n：保留的最佳检查点数量，设置为 50。

• latest\_n：保留的最新检查点数量，设置为 5。

### **6.1.2 chunk\_decode.yaml 解读**

该文件用于配置模型解码过程的参数。在使用连接时序分类（CTC）算法进行束搜索（Beam Search）解码时，解码过程通常在语音识别任务中，将音频信号转换为文本。

chunk\_decode.yaml解码配置文件的参数，如下图5-5所示：

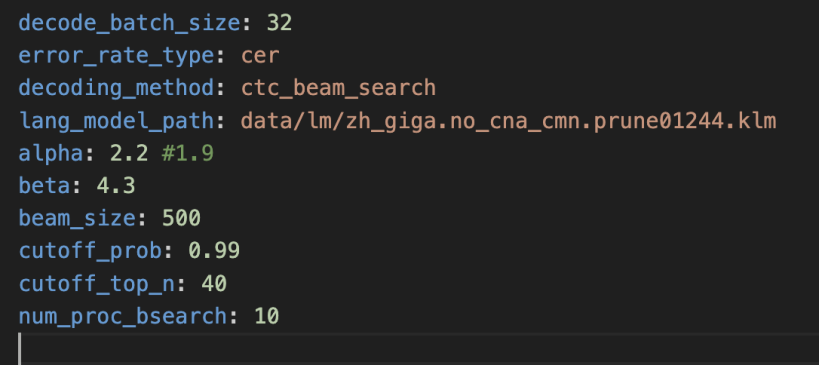


图 6-5 chunk\_decode.yaml配置信息

详细的参数信息解释如下：

• decode\_batch\_size：每批次解码的样本数量。

• error\_rate\_type：错误率的计算类型。cer 表示字符错误率。

• decoding\_method：解码方法。

•ctc\_beam\_search 表示使用CTC（连接时序分类）束搜索解码。

• lang\_model\_path：语言模型的路径。

• alpha：语言模型的权重参数。

• beta：字数长度的权重参数。

• beam\_size：束搜索的束宽，即同时保留的候选路径数量。

• cutoff\_prob：剪枝概率，表示保留的候选路径的概率阈值。

• cutoff\_top\_n：剪枝时保留的最高得分候选路径数量。

• num\_proc\_bsearch：束搜索时使用的并行进程数量。

### **6.1.3 decode.yaml 解读**

decode.yaml 文件用于配置语音识别系统中的解码过程。在语音识别中，解码是将模型预测的概率分布转换为实际的文本。具体内容如下图5-6所示：

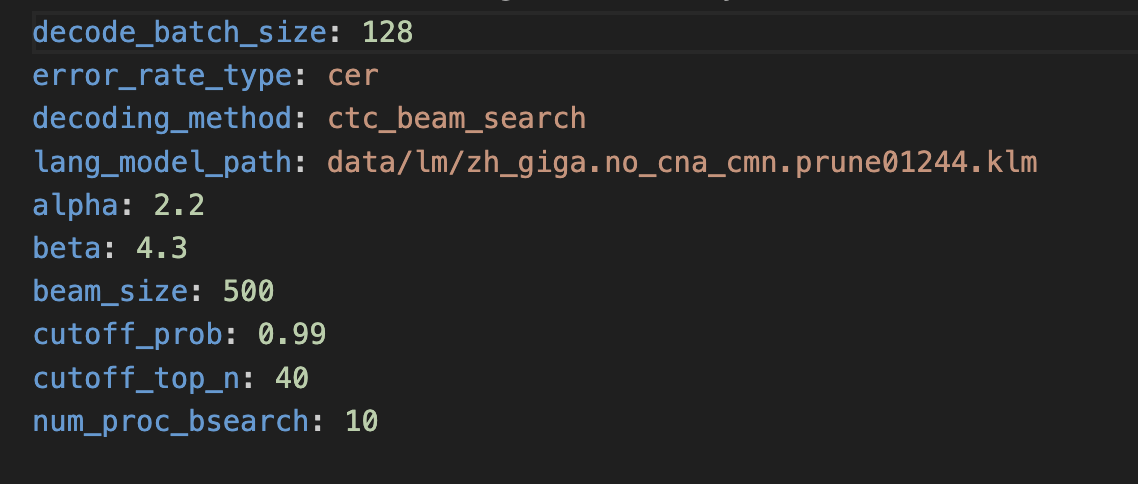


图 6-6 decode.yaml配置信息

该文件指定了解码时的各种参数，包括批量大小、错误率类型、解码方法、语言模型路径和其他相关参数。以下是这些参数的详细解释。

• decode\_batch\_size：每批次解码的样本数量。

• error\_rate\_type：指定评估解码结果时使用的错误率类型。

• decoding\_method：解码方法。

• lang\_model\_path指定用于解码的语言模型文件路径

• alpha：语言模型的权重参数。

• beta：字数长度的权重参数。

• beam\_size：束搜索的束宽，即同时保留的候选路径数量。

• cutoff\_prob：剪枝概率，表示保留的候选路径的概率阈值。

• cutoff\_top\_n：剪枝时保留的最高得分候选路径数量。

• num\_proc\_bsearch：束搜索时使用的并行进程数量。

## **语音识别模型训练**

### **6.2.1 模型训练命令执行流程介绍**

1. 找到语音识别模型训练目录

截屏2024-06-21 22.40.31

图 6-7 语音识别项目目录

1. 激活语音识别虚拟环境paddle\_env

截屏2024-06-21 22.39.49

图 6-8 激活虚拟环境paddle\_env

1. 配置环境文件path.sh生效

截屏2024-06-21 22.39.57

图 6-9 生效环境文件

1. 执行训练文件run.sh

截屏2024-06-21 22.40.21

图 6-10 执行训练文件

### **6.2.2 模型训练命令流程**

（1）path.sh环境文件

执行“source path.sh”命令可以使得path.sh文件中的环境变量生效。“source ${MAIN\_ROOT}/utils/parse\_options.sh”它将支持在shell脚本中使用‘--variable value’的方式。具体内容图下图5-10所示：



图 6-11 path.sh文件内容

（2）run.sh执行文件

所有需要的脚本都在`run.sh`文件中。`run.sh`中包含几个阶段，每个阶段都有其功能对应的应用功能，如下表6-1所示

表 6-1 run.sh阶段功能

|  |  |
| --- | --- |
| 阶段 | 功能 |
| 0 | 数据处理  包括： (1) 下载数据集  (2) 计算训练数据集的CMVN  (3) 获取词汇表文件  (4) 获取训练、开发和测试数据集的manifest文件 |
| 1 | 训练模型 |
| 2 | 通过平均前k个模型来获取最终模型，设置k=1表示选择最佳模型 |
| 3 | 测试最终模型性能 |
| 4 | 导出静态图模型 |
| 5 | 测试静态图模型 |
| 6 | 推断单个音频文件 |

在终端中，可以通过设置`stage`和`stop\_stage`来选择运行的阶段范围，能够一步一步的训练模型，具体命令如下图6-2所示：

截屏2024-06-21 22.40.21

图 6-12 执行具体stage

上图展示了，可具体执行第0阶段的数据处理部分，能够更清晰直观的看到处理进度，如下图6-13所示：



图 6-6 数据处理执行结果

在run.sh文件中先设置一些本地变量介绍，如下图6-14所示：

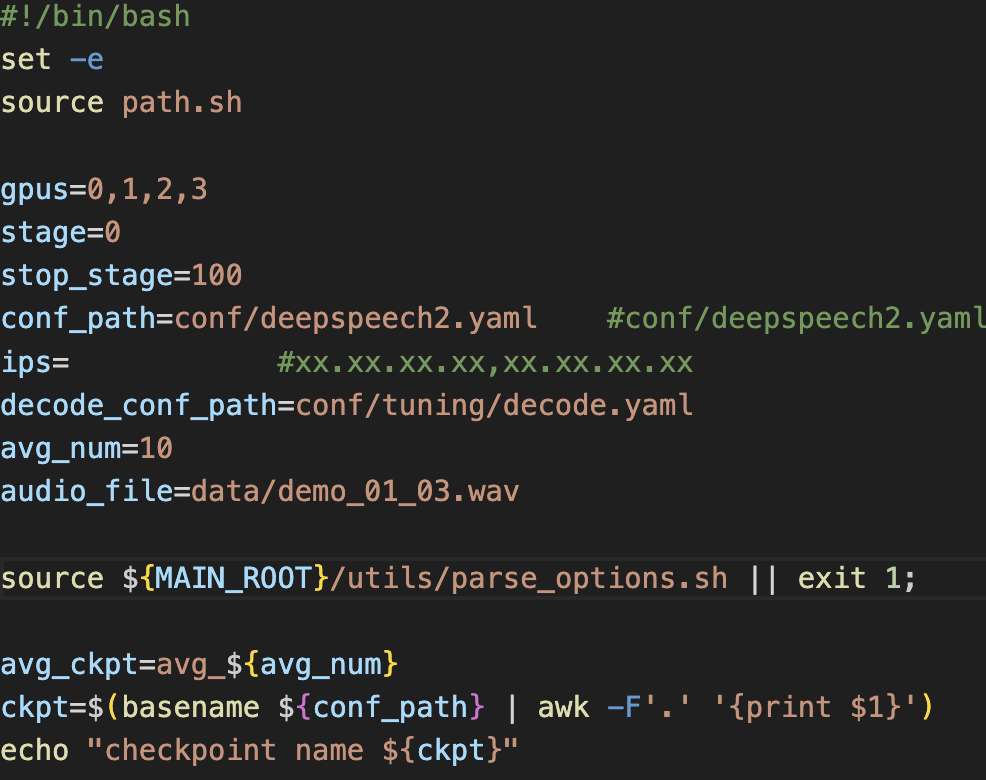


图 6-7 run.sh本地变量设置

其中上述图片6-14中的变量详细解释如下：

• gpus：表示要使用的GPU数量。如果设置`gpus=`，则表示只使用CPU。

• stage：表示实验中要开始的阶段编号。

• stop\_stage：表示实验中要结束的阶段编号。

• conf\_path：表示模型的配置路径。

• avg\_num：表示要平均以获得最终模型的前K个模型数量。

• model\_type：表示模型类型：离线或在线。

• audio\_file：表示在阶段6中要推断的单个文件路径。

• ckpt：表示模型的检查点前缀，例如"deepspeech2"。

设置完所有的本地变量后，run.sh文件配置了不同阶段的执行命令，其中包含的内容如下：

**阶段0：数据处理**

使用此示例，需要首先处理数据，可以在`run.sh`中使用阶段0。代码如下表6-2所示：

表 6-2 数据处理

|  |
| --- |
| 数据处理 |
| if [ ${stage} -le 0 ] && [ ${stop\_stage} -ge 0 ]; then  # 准备数据  bash ./local/data.sh || exit -1  fi |

处理后的结果如上文中提到的5.5数据预处理框架概述中展示的图5-2，5-3，5-4所示框架数据集列表。

**阶段1：模型训练**

训练模型，可以在`run.sh`中使用阶段1。代码如下表6-3所示：

表 6-3 模型训练

|  |
| --- |
| 模型训练 |
| if [ ${stage} -le 1 ] && [ ${stop\_stage} -ge 1 ]; then  # 训练模型，所有`ckpt`在`exp`目录下  CUDA\_VISIBLE\_DEVICES=${gpus} ./local/train.sh ${conf\_path} ${ckpt}  fi |

**阶段2：前k个模型平均**

训练模型后，需要获取用于测试和推断的最终模型。在每个epoch中，模型检查点会被保存，因此可以基于验证损失选择最佳模型，或者可以对前k个模型的参数进行平均来获得最终模型。可以使用阶段2来完成此操作，代码如下表6-4所示：

表 6-4 选择最佳模型

|  |
| --- |
| 前k个模型平均 |
| if [ ${stage} -le 2 ] && [ ${stop\_stage} -ge 2 ]; then  # 平均n个最佳模型  avg.sh best exp/${ckpt}/checkpoints ${avg\_num}  fi |

**阶段3：模型性能评估**

测试阶段用于评估模型性能。测试阶段的代码如下表6-5所示：

表 6-5 评估模型性能

|  |
| --- |
| 模型测试 |
| if [ ${stage} -le 3 ] && [ ${stop\_stage} -ge 3 ]; then  # 测试ckpt avg\_n  CUDA\_VISIBLE\_DEVICES=0 ./local/test.sh ${conf\_path} exp/${ckpt}/checkpoints/${avg\_ckpt} || exit -1  fi |

**阶段4：静态图模型导出**

此阶段用于将动态图转换为静态图，具体代码如下表6-6所示：

表 6-6 静态图模型导出

|  |
| --- |
| 静态图模型导出 |
| if [ ${stage} -le 4 ] && [ ${stop\_stage} -ge 4 ]; then  # 导出ckpt avg\_n  CUDA\_VISIBLE\_DEVICES=0 ./local/export.sh ${conf\_path} exp/${ckpt}/checkpoints/${avg\_ckpt} exp/${ckpt}/checkpoints/${avg\_ckpt}.jit ${model\_type}  fi |

**阶段5：静态图模型测试**

类似于阶段3，静态图模型也可以测试。

表 6-7 静态图模型测试

|  |
| --- |
| 静态图模型测试 |
| if [ ${stage} -le 5 ] && [ ${stop\_stage} -ge 5 ]; then  # 测试导出的ckpt avg\_n  CUDA\_VISIBLE\_DEVICES=0 ./local/test\_export.sh ${conf\_path} exp/${ckpt}/checkpoints/${avg\_ckpt}.jit ${model\_type}|| exit -1  fi |

### **6.2.3 启动训练train.sh脚本**

（1）train.sh流程

运行train.sh脚本先检查参数数量是否正确，再计算使用的 GPU 数量并输出。读取命令行参数，创建输出目录 exp并设置随机种子和内存分配策略。根据 GPU 数量选择单机或分布式训练模式，并执行相应的 Python 脚本进行训练。如果设置了随机种子，清理设置。检查训练是否成功，如果失败则输出错误信息并退出。

train.sh的bash执行流程如下图所示：

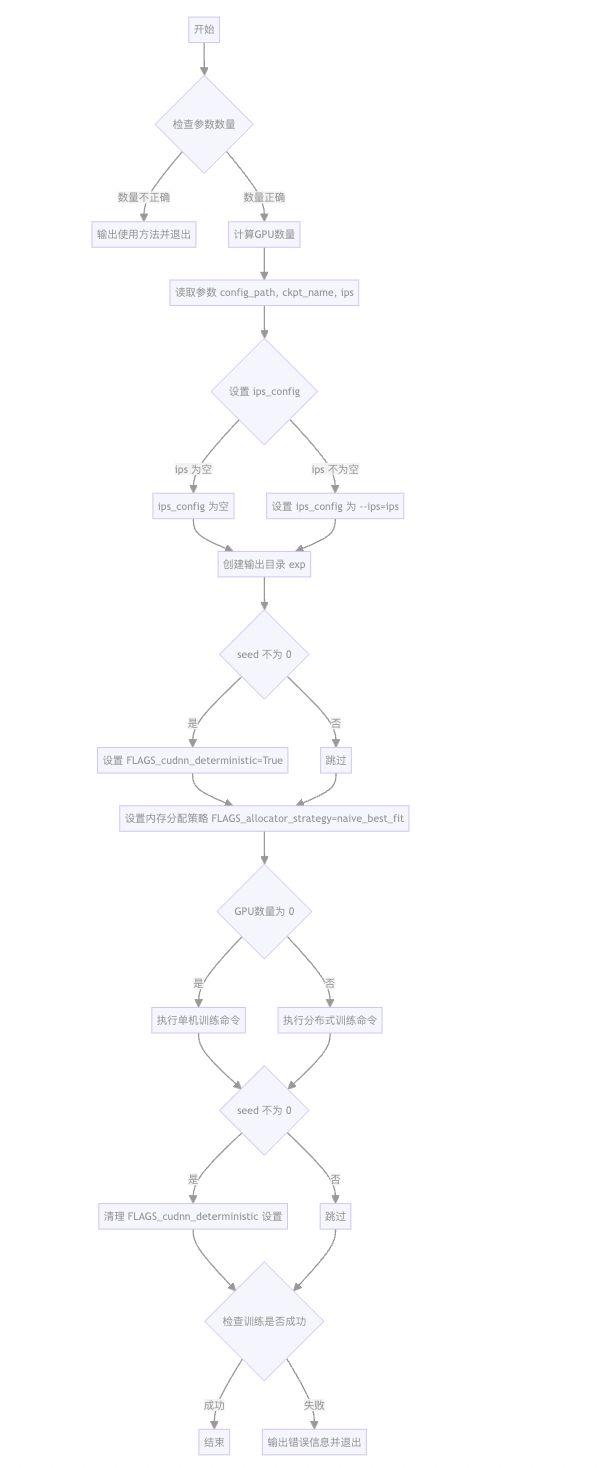


图 6-8 trian.sh文件执行流程图

（2）代码展示

|  |
| --- |
| train.sh文件内容 |
| if [ $# -lt 2 ] || [ $# -gt 3 ];then  echo "usage: CUDA\_VISIBLE\_DEVICES=0 ${0} config\_path ckpt\_name ips(optional)"  exit -1  fi  ngpu=$(echo $CUDA\_VISIBLE\_DEVICES | awk -F "," '{print NF}')  echo "using $ngpu gpus..."  config\_path=$1  ckpt\_name=$2  ips=$3  if [ ! $ips ];then  ips\_config=  else  ips\_config="--ips="${ips}  fi  mkdir -p exp  seed=10086  if [ ${seed} != 0 ]; then  export FLAGS\_cudnn\_deterministic=True  fi  export FLAGS\_allocator\_strategy=naive\_best\_fit  if [ ${ngpu} == 0 ]; then  python3 -u ${BIN\_DIR}/train.py \  --ngpu ${ngpu} \  --config ${config\_path} \  --output exp/${ckpt\_name} \  --seed ${seed}  else  python3 -m paddle.distributed.launch --gpus=${CUDA\_VISIBLE\_DEVICES} ${ips\_config} ${BIN\_DIR}/train.py \  --ngpu ${ngpu} \  --config ${config\_path} \  --output exp/${ckpt\_name} \  --seed ${seed}  fi  if [ ${seed} != 0 ]; then  unset FLAGS\_cudnn\_deterministic  fi  if [ $? -ne 0 ]; then  echo "Failed in training!"  exit 1  fi  exit 0 |

（3）参数详解

·CUDA\_VISIBLE\_DEVICES：指定要使用的 GPU 设备

·config\_path：配置文件的路径，用于指定训练的超参数和数据路径等。

·ckpt\_name：保存模型检查点的名称。

·ips (可选)：用于分布式训练的 IP 地址列表。

### **6.2.4训练过程展示**

训练过程如下图10～21所示：

|  |  |
| --- | --- |
| 截屏2024-06-28 12.08.22  图 6-9 训练1 | /Users/cherly/Library/Containers/com.kingsoft.wpsoffice.mac/Data/tmp/picturecompress_20240628121153/output_4.pngoutput_4  图 6-10 训练2 |
| 截屏2024-06-28 12.09.22  图 6-11 训练3 | /Users/cherly/Library/Containers/com.kingsoft.wpsoffice.mac/Data/tmp/picturecompress_20240628121153/output_3.pngoutput_3  图 6-12 训练4 |
| 截屏2024-06-28 12.09.41  图 6-13 训练5 | 截屏2024-06-28 12.09.53  图 6-14 训练6 |
| 截屏2024-06-28 12.10.00  图 6-15 训练7 | 截屏2024-06-28 12.10.10  图 6-16 训练8 |
| 截屏2024-06-28 12.10.17  图 6-17 训练9 | 截屏2024-06-28 12.10.26  图 6-18 训练10 |
| /Users/cherly/Library/Containers/com.kingsoft.wpsoffice.mac/Data/tmp/picturecompress_20240628121153/output_2.pngoutput_2  图 6-19 训练11 | /Users/cherly/Library/Containers/com.kingsoft.wpsoffice.mac/Data/tmp/picturecompress_20240628121153/output_1.pngoutput_1  图 6-20 训练12 |

共计训练21次，保存21个模型迭代结果，如下图所示：

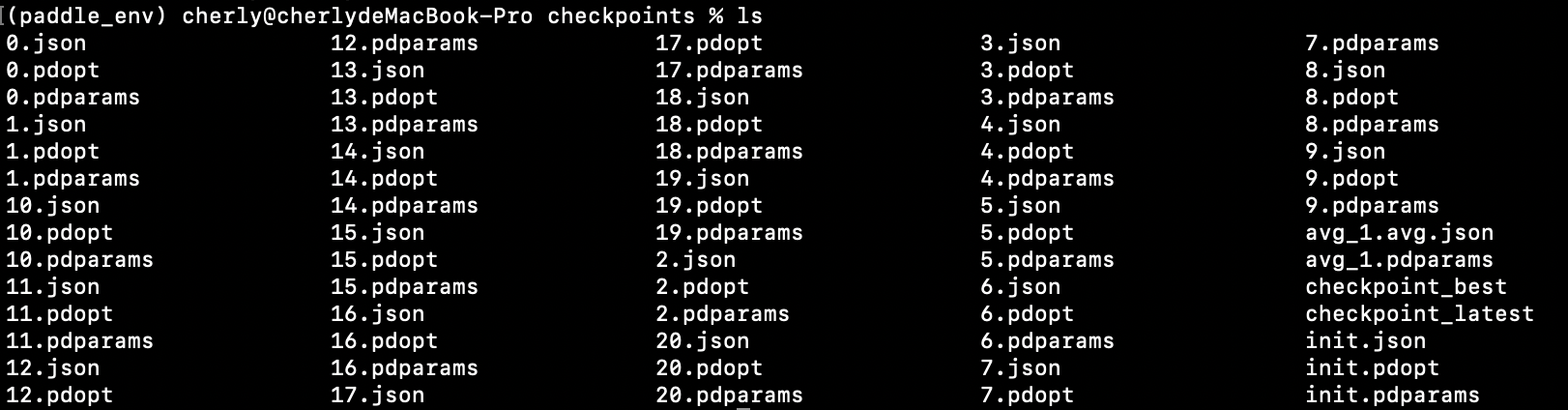


图 6-21 迭代模型结果文件保存list

## **模型优化与调优**

（1）在确保数据文件格式正确且路径有效情况下，根据数据集调整 max\_input\_len 和 max\_output\_len大小

（2）根据可用的GPU内存优化 batch\_size。调整 num\_workers 以匹配CPU核心数量。

（3）根据任务的复杂性和数据集大小调整层数和大小 num\_conv\_layers, num\_rnn\_layers, rnn\_layer\_size。

（4）尝试不同的学习率 (lr) 和衰减率 (lr\_decay) 以获得更好的收敛效果。使用梯度裁剪 (global\_grad\_clip) 避免梯度爆炸，尤其是对于深层网络。

1. **语音合成模型训练**

## **模型配置文件解读**

### **7.1.1 cnndecoder.yaml解读**

该文件是用来定义深度学习训练模型的参数的配置文件，分为一下五个大部分：

(1) 特征提取设置

定义特侦提取中的各项设置，如下图7-1所示：

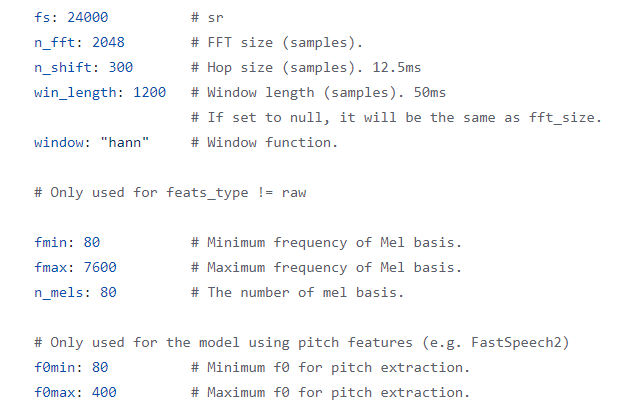


图 7-1 deepspeech.yaml\_FEATURE部分

其中参数解释如下：

• fs: 采样率，这里是24000Hz。

• n\_fft: FFT（快速傅里叶变换）大小，这里是2048个样本。

• n\_shift: 跳跃大小，即相邻两个FFT窗口的重叠部分，这里是300个样本。

• win\_length: 窗口长度，这里是1200个样本，如果设为null，则与FFT大小相同。

• window: 窗口函数类型，这里是"hann"窗。

(2) 数据集设置部分

定义训练的参数，如下图7-2所示：

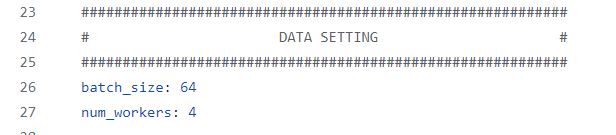


图 7-2 deepspeech.yaml\_DATA部分

其中参数解释如下：

• batch\_size: 每个批次的样本数量，这里是64。

• num\_workers: 加载数据时使用的子进程数量，这里是4

（3）模型设置部分

该部分指定了模型设置，如下图7-3所示：



图 7-3 deepspeech.yaml\_MODEL 部分

其中参数解释如下：

• model: 包含多个子设置，定义了模型的架构和参数。

• adim: 注意力维度。

• aheads: 注意力头的数量。

• elayers, eunits: 编码器层数和编码器前馈网络单元数。

• dlayers, dunits: 解码器层数和解码器前馈网络单元数。

• positionwise\_layer\_type, positionwise\_conv\_kernel\_size: 位置感知层的类型和卷积核大小。

其他设置定义了持续时间预测器、后网(postnet)、位置编码、归一化、dropout率、卷积层参数等。

（4）优化器设置部分

该部分定义优化器设置 ,如下图7-4所示：

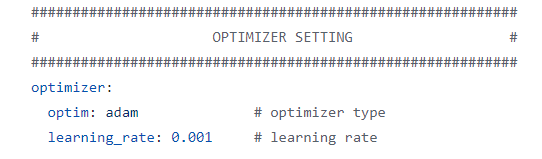


图 7-4 deepspeech2.yaml-OPTIMIZER 部分

其中参数解释如下：

• optim: 优化器类型，这里是"adam"。

• learning\_rate: 学习率。

（5）训练部分

该部分是定义训练模型的参数，如下图7-5所示：

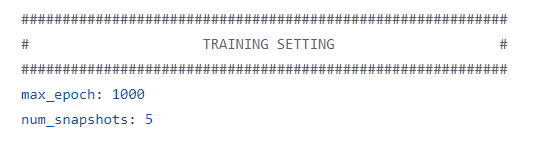


图 7-5 deepspeech2.yaml-TRAINING 部分

其中参数解释如下：

• max\_epoch: 最大训练周期数，这里是1000。

• num\_snapshots: 保存快照的数量，这里是5。

### **7.1.2 conformer.yaml 解读**

该文件是 PaddleSpeech 中用于设置深度学习模型训练参数的配置文件，专门用于配置基于 Conformer 架构的 TTS 模型。具体如下图7-6所示：

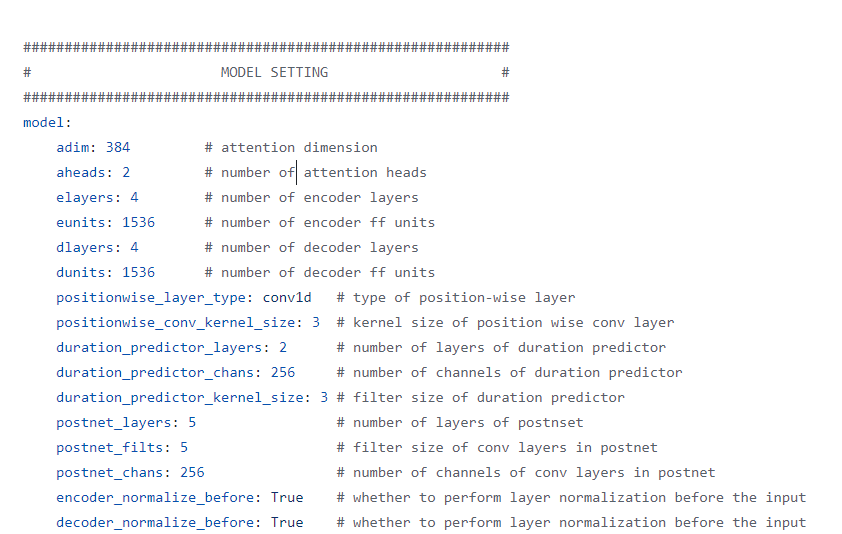


图 7-6 conformer.yaml配置信息

详细的参数信息解释如下：

• 模型设置 (MODEL SETTING)

• adim: 注意力机制中的维度，这里是384。

• aheads: 注意力头的数量，这里是2。

• elayers 和 eunits: 编码器的层数和每层的单元数，编码器有4层，每层有1536个单元。

• dlayers 和 dunits: 解码器的层数和每层的单元数，解码器同样有4层，每层1536个单元。

• positionwise\_layer\_type 和 positionwise\_conv\_kernel\_size: 位置感知层的类型和卷积核大小。

•duration\_predictor\_layers,duration\_predictor\_chans,duration\_predictor\_kernel\_size: 持续时间预测器的层数、通道数和卷积核大小。

• postnet\_layers, postnet\_filts, postnet\_chans: 后网的层数、滤波器大小和通道数。

• encoder\_normalize\_before 和 decoder\_normalize\_before: 是否在编码器和解码器的输入前进行归一化。

### **7.1.3 default.yaml 解读**

default.yaml 是PaddleSpeech 中用于配置文本到语音（TTS）模型的配置文件，详细定义了模型训练和推理过程中的参数设置，具体内容如下图7-7所示：

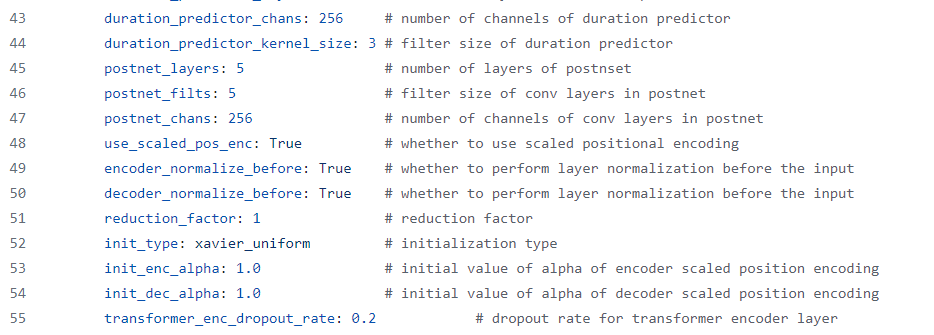


图 7-7 conformer.yaml配置信息

• postnet\_layers, postnet\_filts, postnet\_chans: 后网的层数、滤波器大小和通道数。

• use\_scaled\_pos\_enc: 是否使用缩放的位置编码，这里是True。

• encoder\_normalize\_before 和 decoder\_normalize\_before: 是否在编码器和解码器的输入前进行归一化。

• reduction\_factor: 降采样因子，这里是1。

• init\_type: 参数初始化类型，这里是xavier\_uniform。

• 各种dropout\_rate: 不同组件的dropout率，用于正则化。

• pitch\_predictor\_layers, pitch\_predictor\_chans, 等：音高预测器和嵌入层的设置。

• energy\_predictor\_layers, energy\_predictor\_chans, 等：能量预测器和嵌入层的设置。

## **语音识别模型训练**

### **7.2.1 模型训练命令执行流程介绍**

1. 找到语音识别模型训练目录

93e442481bbcbf9f841d053a1c61b35

图 7-8 语音合成项目目录

1. 激活语音识别虚拟环境paddle\_env

039ed82f2914f77b014f66016fce68d

图7-9 激活虚拟环境padpy\_9

1. 配置环境文件path.sh生效

2f5d36c9b73dfc4dfad7dea1c368746

图 7-10 生效环境文件

1. 执行训练文件run.sh

99d12973d74416a6f5866de7fcfbb55

图 7-11 执行训练文件

### **7.2.2 模型训练命令脚本介绍**

（1）path.sh环境文件

执行“source path.sh”命令可以使得path.sh文件中的环境变量生效。“source ${MAIN\_ROOT}/utils/parse\_options.sh”它将支持在shell脚本中使用‘--variable value’的方式。具体内容图下图7-12所示：



图 7-12 path.sh文件内容

（2）run.sh执行文件

所有需要的脚本都在`run.sh`文件中。`run.sh`中包含几个阶段，每个阶段都有其功能对应的应用功能，如下表7-8所示

表 7-8 run.sh阶段功能

|  |  |
| --- | --- |
| 阶段 | 功能 |
| 0 | 数据处理  包括： (1) 下载数据集  (2) 计算训练数据集的CMVN  (3) 获取词汇表文件  (4) 获取训练、开发和测试数据集的manifest文件 |
| 1 | 训练模型 |
| 2 | 进行语音合成 |
| 3 | 执行端到端语音合成 |
| 4 | 使用静态模型进行推理 |
| 5 | 将 PaddlePaddle 模型转换为 ONNX 格式 |
| 6 | 使用 ONNX Runtime 进行模型推理 |
| 7 | 将模型转换为 Paddle-Lite 格式，适用于在移动设备上运行 |
| 8 | 使用 Paddle-Lite 进行模型推理 |
| 9 | 执行动态量化 |
| 10 | 执行静态量化 |

在终端中，可以通过设置`stage`和`stop\_stage`来选择运行的阶段范围，能够一步一步的训练模型，具体命令如下图7-13所示：

7e2dc342b90468599c9540793cdca43

图 7-13 执行具体stage

上图展示了，可具体执行第0阶段的数据处理部分，能够更清晰直观的看到处理进度，如下图7-14所示：



图 7-14 数据处理执行结果

在run.sh文件中先设置一些本地变量介绍，如下图7-15所示：

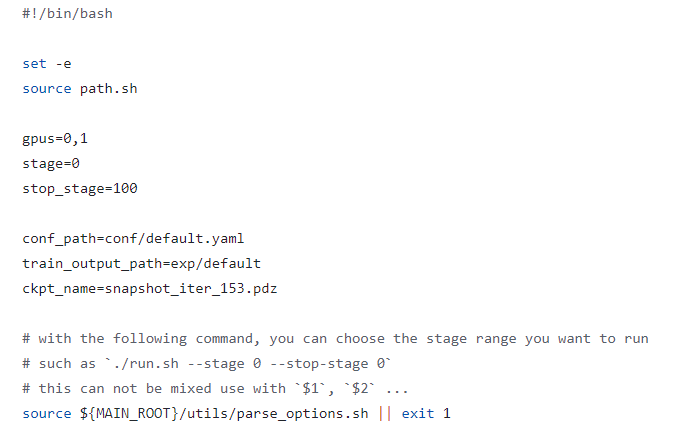


图 7-15 run.sh本地变量设置

其中上述图片5-13中的变量详细解释如下：

• gpus：表示要使用的GPU数量。如果设置`gpus=`，则表示只使用CPU。

• stage：表示实验中要开始的阶段编号。

• stop\_stage：表示实验中要结束的阶段编号。

• conf\_path：表示模型的配置路径。

• train\_output\_path=exp/default：定义了模型训练输出的目录，包括日志、检查点等。

• ckpt\_name=snapshot\_iter\_153.pdz：指定了一个特定的检查点文件名，这个检查点可能在训练或推理时被使用。

设置完所有的本地变量后，run.sh文件配置了不同阶段的执行命令，其中包含的内容如下：

**阶段0：数据处理**

使用此示例，需要首先处理数据，可以在`run.sh`中使用阶段0。代码如下表7-9所示：

表 7-9 数据处理

|  |
| --- |
| 数据处理 |
| if [ ${stage} -le 0 ] && [ ${stop\_stage} -ge 0 ]; then  # 准备数据  bash ./local/data.sh || exit -1  fi |

**阶段1：模型训练**

训练模型，可以在`run.sh`中使用阶段1。代码如下表7-10所示：

表 7-10 模型训练

|  |
| --- |
| 模型训练 |
| if [ ${stage} -le 1 ] && [ ${stop\_stage} -ge 1 ]; then  # 训练模型，所有`ckpt`在`exp`目录下  CUDA\_VISIBLE\_DEVICES=${gpus} ./local/train.sh ${conf\_path} ${ckpt}  fi |

**阶段2：前k个模型平均**

训练模型后，需要获取用于测试和推断的最终模型。在每个epoch中，模型检查点会被保存，因此可以基于验证损失选择最佳模型，或者可以对前k个模型的参数进行平均来获得最终模型。可以使用阶段2来完成此操作，代码如下表7-11所示：

表 7-11 选择最佳模型

|  |
| --- |
| 前k个模型平均 |
| if [ ${stage} -le 2 ] && [ ${stop\_stage} -ge 2 ]; then  # 平均n个最佳模型  avg.sh best exp/${ckpt}/checkpoints ${avg\_num}  fi |

**阶段3：模型性能评估**

测试阶段用于评估模型性能。测试阶段的代码如下表7-12所示：

表 7-12 评估模型性能

|  |
| --- |
| 模型测试 |
| if [ ${stage} -le 3 ] && [ ${stop\_stage} -ge 3 ]; then  # 测试ckpt avg\_n  CUDA\_VISIBLE\_DEVICES=0 ./local/test.sh ${conf\_path} exp/${ckpt}/checkpoints/${avg\_ckpt} || exit -1  fi |

**阶段4：静态图模型导出**

此阶段用于将动态图转换为静态图，具体代码如下表7-13所示：

表 7-13 静态图模型导出

|  |
| --- |
| 静态图模型导出 |
| if [ ${stage} -le 4 ] && [ ${stop\_stage} -ge 4 ]; then  # 导出ckpt avg\_n  CUDA\_VISIBLE\_DEVICES=0 ./local/export.sh ${conf\_path} exp/${ckpt}/checkpoints/${avg\_ckpt} exp/${ckpt}/checkpoints/${avg\_ckpt}.jit ${model\_type}  fi |

**阶段5：静态图模型测试**

类似于阶段3，静态图模型也可以测试。

表 7-14 静态图模型测试

|  |
| --- |
| 静态图模型测试 |
| if [ ${stage} -le 5 ] && [ ${stop\_stage} -ge 5 ]; then  # 测试导出的ckpt avg\_n  CUDA\_VISIBLE\_DEVICES=0 ./local/test\_export.sh ${conf\_path} exp/${ckpt}/checkpoints/${avg\_ckpt}.jit ${model\_type}|| exit -1  fi |

### **7.2.2 启动训练train.sh脚本**

（1）train.sh流程

运行train.sh脚本先将脚本的第一、二个参数赋值给变量 config\_path和train\_output\_path，接下来脚本使用 python3 命令来执行位于环境变量 BIN\_DIR 所指向目录下的 train.py 脚本，脚本中的 python3 命令后面跟着如下参数：

-train-metadata=dump/train/norm/metadata.jsonl：这是传递给 Python 训练脚本的一个命令行参数，指定了训练数据的元数据文件路径。

--dev-metadata=dump/dev/norm/metadata.jsonl：这是指定验证数据集元数据文件路径的命令行参数。

--config=${config\_path}：此参数将 config\_path 变量的值传递给训练脚本，用于指定配置文件。

--output-dir=${train\_output\_path}：此参数将 train\_output\_path 变量的值传递给训练脚本，用于指定输出目录。

--ngpu=1：这个参数告诉训练脚本使用 1 个 GPU 进行训练。

--phones-dict=dump/phone\_id\_map.txt：这个参数指定了音素到ID映射的字典文件路径，这在语音相关的训练中是常见的需求。

train.sh的bash执行流程如下图7-16所示：

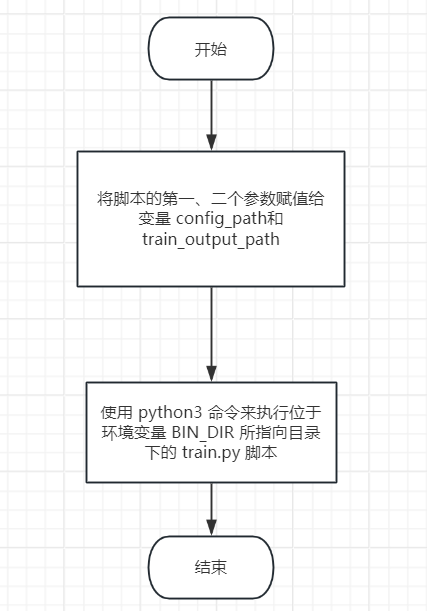


图 7-16 trian.sh文件执行流程图

1. 代码展示

如下表7-15所示

表 7-15 train.sh文件内容

|  |
| --- |
| train.sh文件内容 |
| #!/bin/bash  config\_path=$1  train\_output\_path=$2  python3 ${BIN\_DIR}/train.py \  --train-metadata=dump/train/norm/metadata.jsonl \  --dev-metadata=dump/dev/norm/metadata.jsonl \  --config=${config\_path} \  --output-dir=${train\_output\_path} \  --ngpu=1 \  --phones-dict=dump/phone\_id\_map.txt |

1. **模型评估与测试**

## **模型评估指标介绍**

语音模型评估指标有很多，本文接下来将介绍9种评估指标。

（1）准确率 (Accuracy)：

模型正确识别的样本数占总样本数的比例。它是评估模型整体性能的一个简单指标，但在处理不平衡数据集时可能会有局限性。

（2）错误率 (Error Rate)：

模型错误识别的样本数占总样本数的比例。通常用更具体的指标来描述错误情况，例如词错误率。

（3）词错误率 (Word Error Rate, WER)：

语音识别中最常用的评估指标。WER = (插入错误数 + 删除错误数 + 替换错误数) / 总词数。它衡量了识别结果和实际文本之间的差异。

• 插入错误 (Insertion Errors)：识别出实际不存在的词。

• 删除错误 (Deletion Errors)：漏掉了实际存在的词。

• 替换错误 (Substitution Errors)：将一个词识别为另一个错误的词。

（4）句错误率 (Sentence Error Rate, SER)：

表示识别出的句子中有多少句子包含至少一个错误。SER = (错误句子数 / 总句子数)。

（5）音素错误率 (Phoneme Error Rate, PER)：

类似于WER，但用于评估音素级别的错误。PER = (插入错误数 + 删除错误数 + 替换错误数) / 总音素数。

（6）字符错误率 (Character Error Rate, CER)：

类似于WER，但用于评估字符级别的错误。CER = (插入错误数 + 删除错误数 + 替换错误数) / 总字符数。这在处理汉语等字符语言时尤为重要。

（7）混淆矩阵 (Confusion Matrix)：

用于展示不同类别（或词）之间的混淆情况，帮助理解模型在哪些类别上容易出错。

（8）正确率和召回率 (Precision and Recall)：

• 正确率 (Precision)：正确识别的样本数占识别出的样本数的比例。

• 召回率 (Recall)：正确识别的样本数占实际样本数的比例。

（9）F1 Score：

• 是正确率和召回率的调和平均数，用于综合评估模型的表现。

## **模型性能测试**

本文将根据模型损失率和top-k模型的参数进行平均选择最佳模型。其中，在本文中的损失率和top-k详情介绍如下：

（1）验证损失选择最佳模型：

训练过程中，使用验证集来评估模型的性能。通常在每个epoch结束时计算一次验证损失（validation loss）。验证损失越低，表示模型在验证集上的表现越好。

可以在所有保存的模型检查点中选择验证损失最低的模型作为最终模型。这种方法确保所选的模型在验证数据上有最佳的性能，从而有助于在实际测试和推理时取得更好的结果。

（2）对top-k模型的参数进行平均（Model Averaging）：

对验证损失最低的k个模型进行参数平均。也就是说，选择验证损失最低的前k个模型，将它们的参数进行平均，以获得一个最终模型。这种方法可以减小单个模型的偶然性，提高模型的鲁棒性和泛化能力。

这种参数平均的方法可以帮助缓解训练过程中的波动，避免过拟合，提高模型在不同数据集上的表现。

### **8.2.1语音识别性能测试**

1. 运行测试脚本

在数据预处理、训练模型前两个步骤之后，需要测试模型找到最佳语音识别模型。本项目运行下表所述命令：

表 8-1 语音识别模型测试

|  |
| --- |
| 语音识别模型测试 |
| if [ ${stage} -le 2 ] && [ ${stop\_stage} -ge 2 ]; then  # avg n best model  avg.sh best exp/${ckpt}/checkpoints ${avg\_num}  fi |

执行命令运行avg.sh命令文件，即测试模型以找到最佳模型。下述阐述avg.sh脚本内容：

表 8-2 avg.sh脚本内容

|  |
| --- |
| avg.sh |
| #! /usr/bin/env bash  if [ $# != 3 ]; then  echo "usage: ${0} [best|latest] ckpt\_dir avg\_num"  exit -1  fi  avg\_mode=${1} # best,latest  ckpt\_dir=${2}  average\_num=${3}  decode\_checkpoint=${ckpt\_dir}/avg\_${average\_num}.pdparams  if [ $avg\_mode == best ];then  # best  avg\_model.py \  --dst\_model ${decode\_checkpoint} \  --ckpt\_dir ${ckpt\_dir} \  --num ${average\_num} \  --val\_best  else  # latest  avg\_model.py \  --dst\_model ${decode\_checkpoint} \  --ckpt\_dir ${ckpt\_dir} \  --num ${average\_num}  fi  if [ $? -ne 0 ]; then  echo "Failed in avg ckpt!"  exit 1  fi  exit 0 |

avg.sh文件逻辑为当执行命令为表16所述内容时，即测试模型命令执行找到best模型。否则找到latset模型作为结果模型。

为了更详细的展示模型测试过程，本文将详细阐述在avg.sh命令脚本文件中所提及的avg\_model.py，文件脚本内容从paddlespeech.dataset.s2t框架下导入了avg\_ckpts\_main脚本文件，以测试模型。avg\_ckpts\_main.py脚本文件主要功能内容展示如下所示：

表 8-3 avg\_ckpts\_main.py脚本文件主要功能

|  |
| --- |
| avg\_ckpts\_main.py脚本文件主要功能 |
| def average\_checkpoints(dst\_model="",  ckpt\_dir="",  val\_best=True,  num=5,  min\_epoch=0,  max\_epoch=65536):  paddle.set\_device('cpu')  val\_scores = []  jsons = glob.glob(f'{ckpt\_dir}/[!train]\*.json')  jsons = sorted(jsons, key=os.path.getmtime, reverse=True)  for y in jsons:  with open(y, 'r') as f:  dic\_json = json.load(f)  loss = dic\_json['val\_loss']  epoch = dic\_json['epoch']  if epoch >= min\_epoch and epoch <= max\_epoch:  val\_scores.append((epoch, loss))  assert val\_scores, f"Not find any valid checkpoints: {val\_scores}"  val\_scores = np.array(val\_scores)  if val\_best:  sort\_idx = np.argsort(val\_scores[:, 1])  sorted\_val\_scores = val\_scores[sort\_idx]  else:  sorted\_val\_scores = val\_scores  beat\_val\_scores = sorted\_val\_scores[:num, 1]  selected\_epochs = sorted\_val\_scores[:num, 0].astype(np.int64)  avg\_val\_score = np.mean(beat\_val\_scores)  print("selected val scores = " + str(beat\_val\_scores))  print("selected epochs = " + str(selected\_epochs))  print("averaged val score = " + str(avg\_val\_score))  path\_list = [  ckpt\_dir + '/{}.pdparams'.format(int(epoch))  for epoch in sorted\_val\_scores[:num, 0]  ]  print(path\_list)  avg = None  num = num  assert num == len(path\_list)  for path in path\_list:  print(f'Processing {path}')  states = paddle.load(path)  if avg is None:  avg = states  else:  for k in avg.keys():  avg[k] += states[k]  # average  for k in avg.keys():  if avg[k] is not None:  avg[k] /= num  paddle.save(avg, dst\_model)  print(f'Saving to {dst\_model}')  meta\_path = os.path.splitext(dst\_model)[0] + '.avg.json'  with open(meta\_path, 'w') as f:  data = json.dumps({  "mode": 'val\_best' if val\_best else 'latest',  "avg\_ckpt": dst\_model,  "val\_loss\_mean": avg\_val\_score,  "ckpts": path\_list,  "epochs": selected\_epochs.tolist(),  "val\_losses": beat\_val\_scores.tolist(),  })  f.write(data + "\n") |

主要作用是对多个模型检查点（checkpoints）的参数进行平均，以得到一个最终的模型。该函数根据验证损失（val\_loss）选择验证损失最低的前几个模型，或者选择最近保存的模型，然后对这些模型的参数进行平均。核心思想是通过对多个模型检查点的参数进行平均，以提高最终模型的鲁棒性和泛化能力。这样可以减小训练过程中可能出现的波动和过拟合问题，确保最终模型在测试和推理阶段的性能更为稳定和优秀。

其中，参数解释如下：

• dst\_model: 最终平均后的模型保存路径。

• ckpt\_dir: 检查点文件的目录。

• val\_best: 是否根据验证损失选择最佳模型。默认为True。

• num: 选择进行平均的模型数量。默认为5。

• min\_epoch: 最小的epoch限制。默认为0。

• max\_epoch: 最大的epoch限制。默认为65536。

1. 分析测试结果并展示

本项目共训练迭代20次，共计20个迭代过程中保存的模型模型执行“avg.sh best exp/deepspeech2/checkpoints 1”评估找到最优模型，执行过程如下图所示：

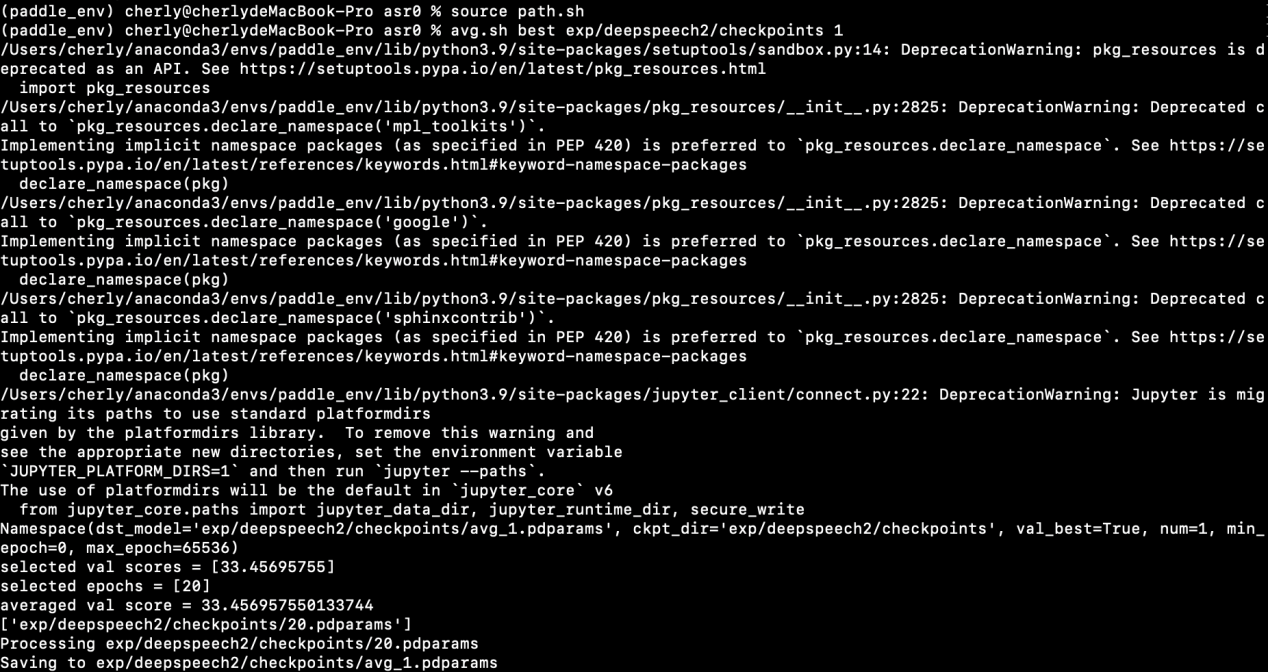


图 8-1 寻找最优模型

由上图可知：

Namespace(dst\_model='exp/deepspeech2/checkpoints/avg\_1.pdparams', ckpt\_dir='exp/deepspeech2/checkpoints', val\_best=True, num=1, min\_epoch=0, max\_epoch=65536)

selected val scores = [33.45695755]

selected epochs = [20]

averaged val score = 33.456957550133744

['exp/deepspeech2/checkpoints/20.pdparams']

Processing exp/deepspeech2/checkpoints/20.pdparams

Saving to exp/deepspeech2/checkpoints/avg\_1.pdparams

在选中的20次迭代当中，最优模型是第二十次迭代模型，其平均验证集损失率为33.45695755。在这里，第20个epoch的模型参数文件被选中并保存为avg\_1.pdparams。

### **8.2.2 语音合成性能测试**

（1）运行测试脚本

整体过程同语音识别模型过程一致，这里将不再赘述。

（2）分析测试结果并展示

部分结果展示如下图所示：

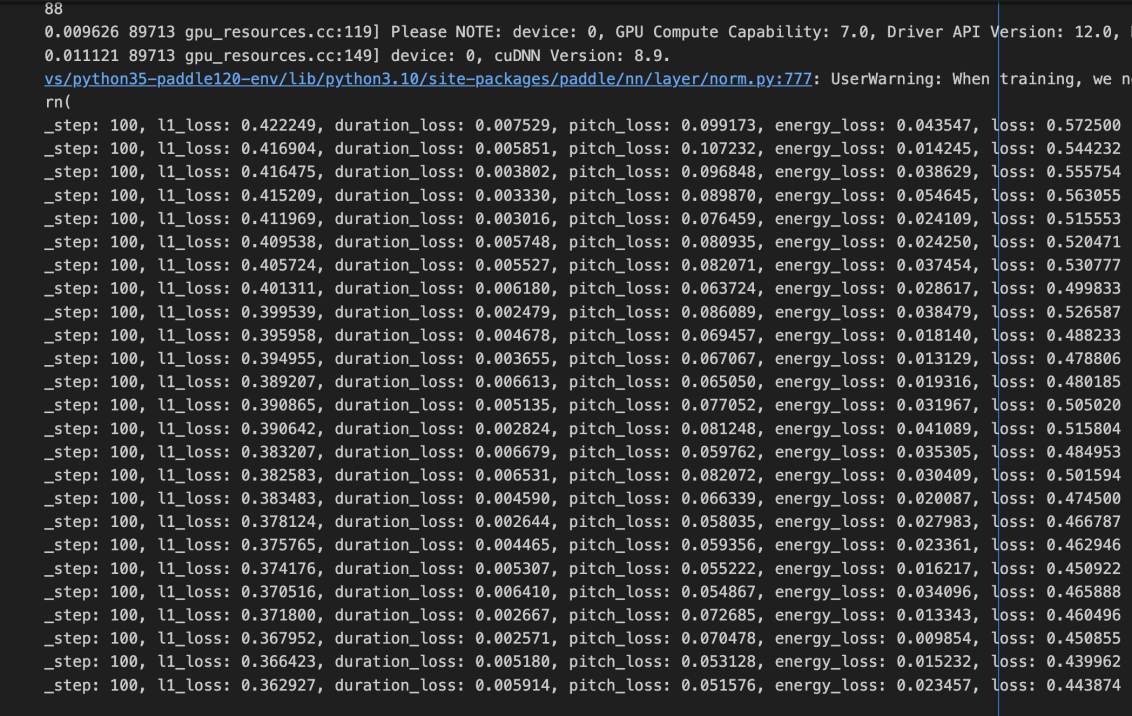


图 8-2 tts3模型测试

如上图所示模型在训练过程中损失率在不断下降，说明训练过程顺利且训练有效。本项目中选择最低损失率为0.443874为最优模型。

1. **模型导出与部署**

## **静态图与动态图介绍**

目前神经网络框架分为静态图框架和动态图框架，PyTorch 和 TensorFlow框架最大的区别就是他们拥有不同的计算图表现形式。 TensorFlow 使用静态图，这意味着需要先定义计算图，然后不断使用它，而在 PyTorch 中，每次都会重新构建一个新的计算图。

可以理解为静态图需要先构建再运行，动态图是一边运行一边构建。

### **9.1.1 静态图介绍**

静态图是指在深度学习模型构建过程中，先定义整个计算图（Computational Graph），然后再将数据逐步送入计算图中进行计算的方式。计算图在构建好之后就不再改变，每次输入数据时就按照图中定义的计算流程进行计算，最终得到结果。

在模型训练和测试期间，静态图允许 TensorFlow 在计算之前进行图形优化和编译，从而提高执行效率。在模型部署中，通常需要将 TensorFlow 模型导出为可运行的二进制文件，以便在生产环境中进行部署。这个导出过程称为 TensorFlow 模型的“冻结”，因为它会将静态图中的变量和操作转换为一个固定的计算图，以便稍后进行推理和预测。在模型的冻结期间，还可以应用各种优化技术，以进一步提高模型的执行效率和准确性，例如量化、剪枝和混合精度等。一旦模型被冻结并导出就可以将其集成到生产环境中，并使用它来进行实时推理。

### **9.1.2 动态图介绍**

在深度模型部署中，动态图通常指的是一种动态计算模型，其计算图的结构可以在运行时动态地改变。这种计算图由计算节点和数据节点组成，其中计算节点表示模型中的各种计算操作，数据节点表示输入、输出和中间结果等数据。

动态图是一种在运行时动态构建计算图的编程模式。在动态图中，整个计算图的结构可以在程序运行时动态地构建和修改，这种灵活性使得动态图在深度学习模型的训练、调试和部署中具有很大的优势。在深度学习框架中，使用动态图来进行模型构建和训练的常见库包括 PyTorch 和 Chainer 等。这些库使用动态图来构建模型，可以更加方便地进行模型训练和调试，同时也支持将模型转换为静态图形式，以便在生产环境中进行部署和优化。

### **9.1.3 静态图与动态图对比**

对于使用者来说，两种形式的计算图有着非常大的区别，同时静态图和动态图都有他们各自的优点，比如动态图比较方便debug，使用者能够用任何他们喜欢的方式进行debug，同时非常直观，而静态图是通过先定义后运行的方式，之后再次运行的时候就不再需要重新构建计算图，所以速度会比动态图更快。如下表9-1优缺点对比所示

表 9-1 静态图与动态图对比

|  |  |  |
| --- | --- | --- |
|  | 动态图框架 | 静态图框架 |
| 灵活性 | 运行时构建和修改神经网络结构，灵活性高 | 计算图在编译时确定，灵活性相对较低 |
| 性能优化 | 每次运行时重新构建计算图，可能导致性能不如静态图框架 | 编译器全局优化，性能更加高效，适合大规模部署和跨平台运行 |
| 可扩展性 | 可能不如静态图框架稳定和可扩展 | 跨平台兼容性较高，支持多种编程接口和硬件平台 |
| 调试和开发效率 | 实时观察和修改变量，便于调试和开发 | 可视化和调试工具相对完善，方便观察和调试模型 |
| 适用场景 | 适合模型开发和调试阶段，需要动态修改模型结构的场景 | 适合大规模部署和跨平台运行，对性能要求较高的应用场景 |

### **9.1.4 模型静态图与动态图导出转换意义**

将模型从动态图导出成静态图，可以在推理性能、跨平台兼容性、调试与优化、模型压缩与量化以及分布式训练等方面带来诸多好处。对于部署到生产环境的模型，静态图是更好的选择。

在本项目中，模型首先导出为动态图方便后续针对训练结果调整语音速度。在微调后达到满意的需求标准之后，将模型导出为静态图部署在项目本地语音交互系统下，方便后续可实时识别。

## **语音模型导出脚本阐述**

语音识别与语音合成模型导出的方法相同，下述将不再赘述两者共同部分。本文将以语音合成为例讲述语音模型导出脚本。

### **9.2.1 将模型导出为静态图**

运行tts模型目录下的run.sh文件中的第7阶段可以将导出Paddle Lite格式的静态图模型，其中run.sh第7阶段内容如下表9-2所示：

表 9-2 run.sh第7阶段

|  |
| --- |
| tts/run.sh/stage=7-stop\_stage=7导出静态图模型 |
| if [ ${stage} -le 7 ] && [ ${stop\_stage} -ge 7 ]; then      # NOTE by yuantian 2022.11.21: please compile develop version of Paddle-Lite to export and run TTS models,      #                   cause TTS models are supported by https://github.com/PaddlePaddle/Paddle-Lite/pull/9587      #                   and https://github.com/PaddlePaddle/Paddle-Lite/pull/9706      ./local/export2lite.sh ${train\_output\_path} inference pdlite fastspeech2\_csmsc x86      ./local/export2lite.sh ${train\_output\_path} inference pdlite pwgan\_csmsc x86      # ./local/export2lite.sh ${train\_output\_path} inference pdlite mb\_melgan\_csmsc x86      # ./local/export2lite.sh ${train\_output\_path} inference pdlite hifigan\_csmsc x86  fi |

运行上述stage=7的阶段可以运行export2lite.sh文件从而将训练出来的模型优化并导出为 Paddle Lite 格式的静态图模型，具体内容如下表9-3所示：

表 9-3 export2lite.sh

|  |
| --- |
| export2lite.sh |
| train\_output\_path=$1  model\_dir=$2  output\_dir=$3  model=$4  valid\_targets=$5  model\_name=${model%\_\*}  echo model\_name: ${model\_name}  suffix=${valid\_targets%,\*}  mkdir -p ${train\_output\_path}/${output\_dir}  paddle\_lite\_opt \  --model\_file ${train\_output\_path}/${model\_dir}/${model}.pdmodel \  --param\_file ${train\_output\_path}/${model\_dir}/${model}.pdiparams \  --optimize\_out ${train\_output\_path}/${output\_dir}/${model}\_${suffix} \  --valid\_targets ${valid\_targets} |

整个脚本的流程是从参数中获取路径和名称信息，然后创建必要的目录，最后使用 paddle\_lite\_opt 工具将指定的 Paddle 模型文件和参数文件导出为 Paddle Lite 格式的静态图模型，并存放在指定的输出目录中。针对上述表中的参数做下述具体概念阐述：

• train\_output\_path: 训练输出路径，这是模型训练结果存放的目录。

• model\_dir: 模型目录，这是包含模型文件的目录。

• output\_dir: 输出目录，用于存放导出后的模型。

• model: 模型的名称前缀，通常包含模型和参数文件的公共部分名称。

• valid\_targets: 有效目标，表示模型导出的目标平台，可以是 arm, x86 等。

• model\_name=${model%\_\*}: 提取模型名称，model\_name 变为 fastspeech2。

• echo model\_name: ${model\_name}: 打印模型名称。

• suffix=${valid\_targets%,\*}: 提取目标平台，去掉平台名称后的逗号及其后的部分。例如，如果 valid\_targets 是 arm, 则 suffix 变为 arm。

• model\_file: 指定模型文件的路径。

• param\_file: 指定模型参数文件的路径。

• optimize\_out: 指定优化后模型的输出路径。

• valid\_targets: 指定有效目标平台。

### **9.2.2 模型动态推理**

在step=8中，执行的是模型的动态图导出。主要命令对应执行的对象如下表9-4所示：

表 9-4 动态图导出

|  |
| --- |
| 动态图导出 |
| finetuneTTS(label\_path, exp\_name, start\_step=8, end\_step=8, max\_finetune\_step=100, batch\_size=32, learning\_rate=0.001) |

在动态图模式下，量化过程可以在模型运行时进行。动态量化可以在每次前向传递时动态地将浮点数转换为定点数。这样可以根据合成效果调整语速快慢，如下图9-1所示

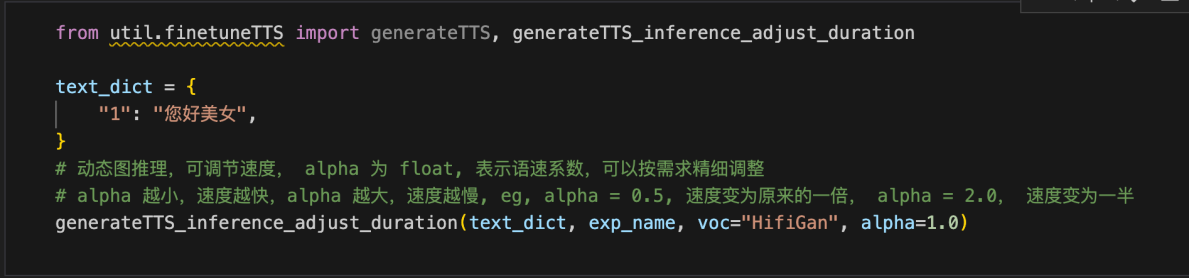


图 9-1 模型动态推理

模型导出为动态图进行动态推理后输出调整结果如下图9-2所示：

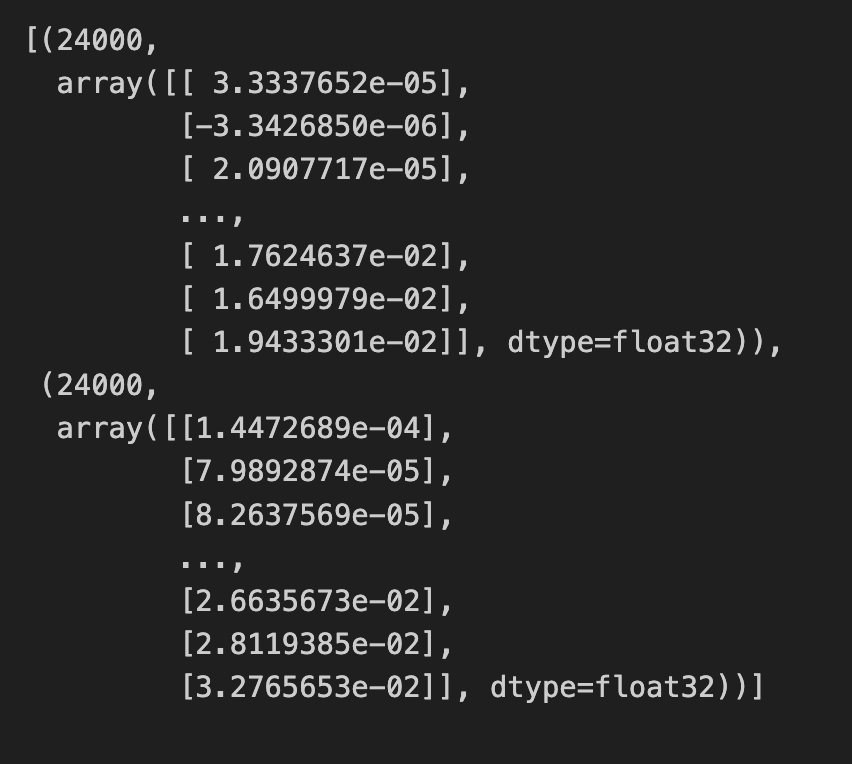


图 9-2 输出调整结果

1. **语音人机交互系统设计**

## **语音交互系统架构整体设计逻辑**

为通过语音指令输入控制多设备并通过语音合成播报结果的系统，本组设计一个基于客户端-服务器（C/S）架构的项目，整体设计逻辑如下图所示：

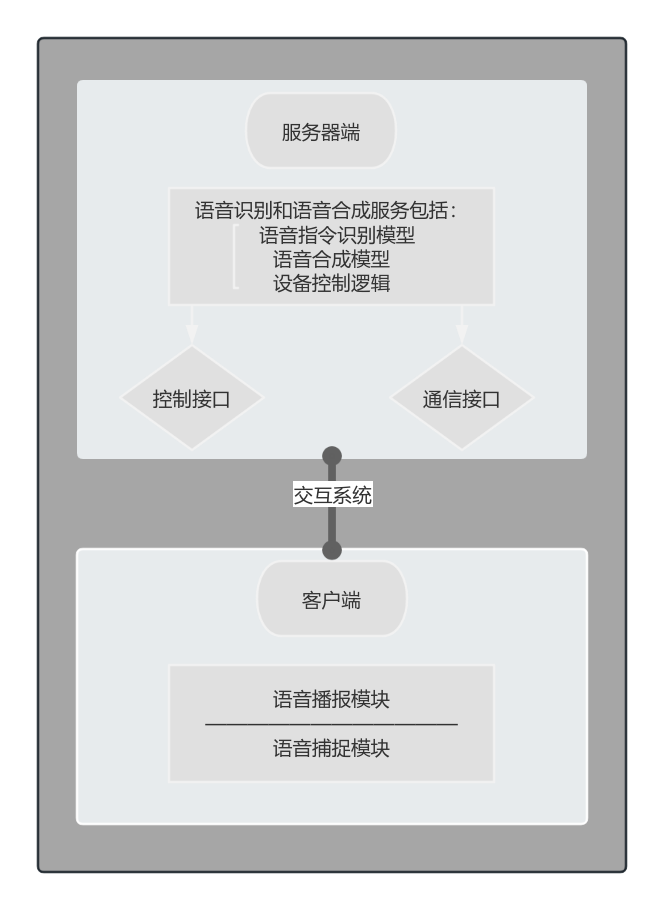


图 10-1 系统设计架构

### **10.1.1 Client-Server (C/S) 模式**

（1）Client-Server (C/S) 模式简介

（2）系统项目C/S模式作用

本项目采取在客户端PC\_2上运行本项目设计的xx语音助手应用程序，其功能是主要负责捕捉用户语音输入，传送到服务器端进行处理，接收处理结果并进行语音合成播报。

当服务器PC\_1端接收到客户端ipad传来的信息，集中处理信息并完成语音指令识别、控制其他设备以及生成语音合成结果的功能。

### **10.1.2 客户端设计**

客户端包含两个模块，分别为语音捕捉模块和语音播报模块。其可轻松实现人机语音交互，方便用户深度体验交互功能。

(1)语音捕捉模块

捕捉用户的语音指令，实时发送到服务器端进行处理。

(2)语音播报模块

接收服务器端返回的语音合成结果，通过设备的扬声器进行播报。

### **10.1.2 服务器端设计**

服务器端包含三种服务，分别为语音识别和语音合成服务、控制接口服务与通信接口服务。三者共同工作奠定了人机语音交互实现的基础

（1）语音识别和语音合成服务

1. 语音指令识别模型

部署训练好的端到端中文语音识别模型，处理客户端发送的语音指令，转换成文本指令。

1. 语音合成模型

部署训练好的端到端中文语音合成模型，根据文本生成自然、有情感的语音播报。

1. 设备控制逻辑

解析识别出的文本指令，确定需要控制的设备和操作，通过通信接口发送控制命令。

（2）控制接口服务

提供API用于客户端和其他设备的控制，处理设备的状态和执行控制命令。

（3）通信接口服务

处理客户端和服务器之间的通信，确保指令和结果能够快速、稳定地传递。

## **客户端运行流程阐述**

client\_design目录包含客户端设计的文件，包括index.html网页、styles.css样式表和script、app两个JavaScript脚本。因代码过长，本文暂不展示代码内容，具体实现功能见下文

index.html: 客户端的主页面，包含了网页的文本和框架

表 10-1 index.html解析

|  |
| --- |
| index.html解析 |
| Step1 文档类型声明：<!DOCTYPE html>告诉浏览器这是一个HTML5文档。  Step2 HTML标签：<html lang="en">定义了文档的根元素，并设置页面语言为英语。  Step3 头部信息：  <meta>标签：定义了字符集为UTF-8，并设置了视口的初始比例为1.0，这有助于在不同设备上进行响应式设计。  <title>：定义了页面的标题为“欢迎体验 GenuispeechAI助手”。  Step4 Bootstrap CSS链接：  通过<link>标签引入了Bootstrap的CSS文件，用于页面样式设计。  Step5 自定义CSS链接：  引入了自定义的CSS文件styles.css，这可能包含了页面特有的样式。  Step6 页面主体：  <body>标签内包含了页面的主要内容。  使用了Bootstrap的栅格系统来布局页面，包括container, row, col-md-6等类。  Step7 标题：  <h1>标签定义了页面的主标题，并应用了center-title类，这个类可能在styles.css中定义了居中和样式效果。  Step8 卡片组件：  使用了Bootstrap的card组件来创建一个包含语音命令的区域。  card-title定义了卡片的标题。  一个按钮元素，设置了id为start-record-btn，类为btn btn-primary btn-block，表示这是一个全宽度的主要操作按钮，用户可以点击它开始录音。  Step9 转录区域：  <div>元素，id为transcript，用于显示语音识别的转录文本。  Step10 JavaScript文件链接：  引入了Bootstrap的JavaScript依赖，包括jQuery, Popper.js和Bootstrap的JavaScript插件。  引入了自定义的JavaScript文件app.js，这可能是前面提到的录音和语音识别功能的实现代码。 |

styles.css用于渲染网页的样式

表 10-2 styles.css解析

|  |
| --- |
| styles.css解析 |
| body 选择器：设置了整个网页的背景颜色为#f8f9fa，这是一种浅灰色。  .card 选择器：为.card类设置了margin-top属性为50px，这通常用于给卡片组件添加顶部外边距。  .center-title 选择器：定义了一个居中对齐的标题样式，具有以下特点：  文本居中对齐（text-align: center;）  上边距为3rem（rem是相对于根元素的字体大小的单位）  字体颜色为#3498db，这是一种鲜艳的蓝色  字体大小为2.5rem  字体家族为Times New Roman，如果该字体不可用，则回退到Times或无衬线字体serif  字体加粗（font-weight: bold;）  文字阴影效果，水平和垂直偏移2px，模糊半径5px，颜色为半透明的黑色（rgba(0, 0, 0, 0.3)）  #transcript 选择器：为ID为transcript的元素设置了样式，包括：  边框为1px实线，颜色为#ccc（浅灰色）  内边距为10px  边框圆角为5px  最小高度为100px  背景颜色为#fff（白色）  button 选择器：定义了按钮的样式，包括：  字体大小为16px  内边距为10px 20px（上下10px，左右20px）  背景颜色为#4CAF50，这是一种绿色  字体颜色为白色  边框为无（border: none;）  边框圆角为5px  鼠标悬停时变为手形（cursor: pointer;）  button:hover 伪类：定义了鼠标悬停在按钮上时的样式，背景颜色变为#45a049，这是一种较深的绿色。 |

js目录包含script和app两个JavaScript脚本，用于网页的交互功能。

app.js包含与服务器交互的逻辑

表 10-3 app.js解析

|  |
| --- |
| app.js解析 |
| Step1获取DOM元素：  startRecordBtn：开始录音按钮。  stopRecordBtn：停止录音按钮。  statusDisplay：用于显示状态信息的元素。  audioPlayback：用于播放音频的元素  Step2 WebSocket连接：  创建一个新的WebSocket连接到服务器，地址为ws://your-server-address:your-port。需要将your-server-address和your-port替换为实际的服务器地址和端口。  Step3处理WebSocket消息：  当WebSocket接收到消息时，会创建一个新的Blob对象，包含从服务器接收到的音频数据，并设置其类型为audio/wav。  使用URL.createObjectURL创建一个可以用于播放的音频URL。  将audioPlayback元素的src属性设置为新创建的URL，并调用play方法播放音频。  更新statusDisplay元素的文本内容为“状态: 播放服务器响应...”。  Step4 捕捉用户语音：  为开始录音按钮添加点击事件监听器，当点击时执行以下操作：  使用navigator.mediaDevices.getUserMedia请求访问用户的麦克风。  创建MediaRecorder实例，用于录音。  设置MediaRecorder的事件处理函数：  onstart：录音开始时，清空audioChunks数组，更新状态显示，启用停止按钮，禁用开始按钮。  ondataavailable：每当有新的音频数据可用时，将其添加到audioChunks数组中。  onstop：录音停止时，将audioChunks数组转换为Blob对象，调用sendAudioToServer函数发送音频数据到服务器，更新状态显示，禁用停止按钮，启用开始按钮。  调用mediaRecorder.start开始录音。  Step5停止录音：  为停止录音按钮添加点击事件监听器，当点击时调用mediaRecorder.stop停止录音。  Step6发送音频数据到服务器：  定义sendAudioToServer函数，用于将音频Blob发送到服务器：  创建FileReader实例。  设置FileReader的onload事件处理函数，当读取完成时，将结果（即音频数据的ArrayBuffer）通过WebSocket发送到服务器。  调用readAsArrayBuffer方法将音频Blob读取为ArrayBuffer。 |

Script.js: 网页运行的脚本。

表 10-4 Script.js解析

|  |
| --- |
| Script.js解析 |
| Step1 添加事件监听器：  为ID为start-record-btn的元素添加点击事件监听器，当点击时调用startRecording函数。  Step2 定义startRecording函数：  首先检查浏览器是否支持navigator.mediaDevices和getUserMedia方法，如果不支持，则弹出警告并返回。  Step3 获取用户媒体：  使用navigator.mediaDevices.getUserMedia方法请求访问用户的麦克风，并获取一个MediaStream对象。  创建MediaRecorder实例并开始录音：  使用获取到的MediaStream创建MediaRecorder实例，并调用start方法开始录音。  Step4 存储音频块：  定义一个数组audioChunks用于存储录音过程中的数据块。  监听dataavailable事件：  当MediaRecorder实例的dataavailable事件触发时，将事件的数据（event.data）添加到audioChunks数组中。  Step5 监听stop事件：  当MediaRecorder实例的stop事件触发时，将audioChunks数组转换为一个Blob对象。  创建FormData对象并发送POST请求：  创建一个FormData对象，并将音频Blob作为表单数据的一部分。  使用fetch函数向服务器的/recognize端点发送POST请求，包含音频数据。  Step6 处理语音识别响应：  服务器响应后，解析响应为JSON格式，并获取识别的文本。  将识别的文本设置为ID为transcript的元素的文本内容。  Step7 语音合成：  创建另一个fetch请求，向服务器的/synthesize端点发送POST请求，包含要合成的文本。  Step8 服务器响应后，解析响应为JSON格式，并获取合成的音频URL。  Step9 播放合成的音频：  使用获取到的音频URL创建一个Audio对象，并调用play方法播放音频。  Step10停止录音：  使用setTimeout函数在5秒后停止录音。 |

## **服务器端运行流程阐述**

server\_design:这个目录包含服务器端的三个Python脚本（main.py、Server\_recognize.py以及Server\_answer.py），用于处理客户端的请求和执行语音识别与合成任务。

main.py:控制整个流程的脚本，包括启动服务器、接收客户端请求、调用识别和合成功能。

表 10-5 main.py代码

|  |
| --- |
| main.py |
| import os  import subprocess  from flask import Flask, request, jsonify  import datetime  app = Flask(\_\_name\_\_)  #语音监听与识别  @app.route('/recognize', methods=['POST'])  def recognize():  # 获取音频文件  audio\_data = request.files['audio']    # 生成唯一文件名并保存在指定目录  timestamp = datetime.datetime.now().strftime("%Y%m%d\_%H%M%S")  audio\_filename = f"recording\_{timestamp}.wav"  audio\_path = os.path.join('/Users/cherly/PaddleSpeech/digitalspeechfinal/genuis/data\_temp/recordingdata', audio\_filename)  audio\_data.save(audio\_path)  # 调用识别服务  result = subprocess.run(['conda', 'run', '-n', 'paddle\_env', 'python', 'server\_recognize.py', audio\_path], capture\_output=True, text=True)  recognized\_text = result.stdout.strip()  return jsonify({'text': recognized\_text})  #语音合成  @app.route('/synthesize', methods=['POST'])  def synthesize():  # 获取文本数据  text = request.json['text']    # 调用合成服务  result = subprocess.run(['conda', 'run', '-n', 'padpy\_9', 'python', 'server\_answer.py', text], capture\_output=True, text=True)  response\_audio = result.stdout.strip()  return jsonify({'audio': response\_audio})  if \_\_name\_\_ == '\_\_main\_\_':  app.run(host='0.0.0.0', port=5000) |

Server\_recognize.py: 实现语音识别功能的脚本，包含调用语音识别模型、处理识别结果等逻辑。

表 10-6 Server\_recognize.py代码

|  |
| --- |
| Server\_recognize.py |
| import sys  import subprocess  import paddle  import paddlespeech.cli.asr.infer as infer  import csv  import datetime  # 初始化ASR执行器  asr\_executor = infer.ASRExecutor()  # 音频实时处理ffmpeg  def ffmpeg\_run(input\_filepath):  output\_filepath = "/Users/cherly/PaddleSpeech/digitalspeechfinal/data/check.wav"  try:  # 使用FFmpeg进行转换  subprocess.run(['ffmpeg', '-i', input\_filepath, '-ar', '16000', '-ac', '1', output\_filepath], check=True)  print(f"成功转换: {input\_filepath} -> {output\_filepath}")  return output\_filepath  except subprocess.CalledProcessError as e:  print(f"转换过程中出现错误: {input\_filepath} -> {output\_filepath}: {e}")  return None  def recognize\_audio(audio\_path):  # 使用ASR执行器进行语音识别  result = asr\_executor(  audio\_file=audio\_path,  model='conformer\_wenetspeech',  lang='zh',  sample\_rate=16000  )  return result  def save\_to\_csv(timestamp, text, csv\_path):  with open(csv\_path, mode='a', newline='') as file:  writer = csv.writer(file)  writer.writerow([timestamp, text])  print(f"已保存识别结果到 {csv\_path}")  if \_\_name\_\_ == '\_\_main\_\_':  audio\_path = sys.argv[1]  csv\_path = "/Users/cherly/PaddleSpeech/digitalspeechfinal/genuis/data\_temp/recognition\_results.csv"    input\_filepath = ffmpeg\_run(audio\_path)  if input\_filepath:  recognized\_text = recognize\_audio(input\_filepath)  print(recognized\_text)    # 获取当前时间  timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")    # 保存到CSV文件  save\_to\_csv(timestamp, recognized\_text, csv\_path) |

Server\_answer.py:实现语音合成并播报功能的脚本，包含调用语音合成模型、生成语音文件、播放语音等逻辑。

表 10-7 Server\_answer.py代码

|  |
| --- |
| Server\_answer.py |
| import sys  import paddle  import paddlespeech  from paddle.nn.layer import layers  from paddlespeech.cli.tts.infer import TTSExecutor  import datetime  #from paddlespeech.t2s.models.fastspeech2 import FastSpeech2Model  # 初始化TTSExecutor  tts = TTSExecutor()  def synthesize\_speech(text):  #model = FastSpeech2Model.from\_pretrained('fastspeech2\_cnndecoder\_csmsc')  timestamp = datetime.datetime.now().strftime("%Y%m%d\_%H%M%S")  audio\_path = f"/Users/cherly/PaddleSpeech/digitalspeechfinal/genuis/data\_temp/answerdata/output{timestamp}.wav"  tts(text, output= audio\_path, am="fastspeech2\_male", voc="pwgan\_male")  return audio\_path  if \_\_name\_\_ == '\_\_main\_\_':  text = sys.argv[1]  response\_audio = synthesize\_speech(text)  print(response\_audio) |

## **总体运行流程阐述**

### **10.4.1 脚本阐述**

整体流程代码运行集成为genuis\_run.sh

表 10-7 genuis\_run.sh代码

|  |
| --- |
| genuis\_run.sh |
| #!/bin/bash  # Written by Sun Yeqian  # 语音识别模型预训练: paddle\_env (deepspeech2)  # 语音合成: padpy\_9 (fastspeech)  # 激活conda  source ~/miniconda3/etc/profile.d/conda.sh  # 安装依赖  conda activate paddle\_env  pip install -r /Users/cherly/PaddleSpeech/digitalspeechfinal/genuis/requirement.txt  conda activate padpy\_9  pip install -r /Users/cherly/PaddleSpeech/digitalspeechfinal/genuis/requirement.txt  # 启动服务器  conda activate paddle\_env  python /Users/cherly/PaddleSpeech/digitalspeechfinal/genuis/genuis\_main.py &  MAIN\_PID=$!  # 启动客户端  cd /Users/cherly/PaddleSpeech/digitalspeechfinal/genuis/client\_design  python -m http.server 8000 &  # 打开默认浏览器并访问客户端页面  sleep 5  if command -v xdg-open >/dev/null; then  xdg-open http://localhost:8000  elif command -v open >/dev/null; then  open http://localhost:8000  else  echo "请手动打开浏览器并访问 http://localhost:8000"  fi解析这个代码的具体流程 |

具体流程概括：

**Step1 激活conda环境**

source ~/miniconda3/etc/profile.d/conda.sh 这行命令用于激活conda环境，切换到PaddleSpeech环境：conda activate paddle\_env 命令用于激活名为paddle\_env的conda环境，这个环境可能已经配置了PaddleSpeech所需的所有依赖。

**Step2 安装依赖**

pip install -r /Users/cherly/PaddleSpeech/digitalspeechfinal/genius/requirement.txt 命令用于安装一个名为requirement.txt的文件中列出的Python依赖。这个文件是为了方便项目整体三个环境的包的安装，其中包含项目所需的必要库

**Step3 再次激活PaddleSpeech环境**

脚本再次执行conda activate paddle\_env，确保环境是激活状态。

**Step4 启动服务器**

python /Users/cherly/PaddleSpeech/digitalspeechfinal/genius/genius\_main.py & 命令启动了一个名为genius\_main.py的Python脚本，并将其放入后台运行。MAIN\_PID=$! 这行命令将后台进程的PID（进程ID）保存到变量MAIN\_PID中。

**Step5 切换到客户端目录**

cd /Users/cherly/PaddleSpeech/digitalspeechfinal/genius/client\_design 命令切换到客户端设计的目录。

**Step6 启动客户端服务器**

python -m http.server 8000 & 命令启动了一个简单的HTTP服务器，监听8000端口，并将客户端的文件提供给浏览器访问。

**Step7 等待服务器启动**

sleep 5 命令让脚本暂停5秒钟，以确保服务器有足够的时间启动。

**Step8 打开浏览器访问客户端页面**

接下来的几行代码尝试使用系统默认的浏览器打开http://localhost:8000。如果系统支持xdg-open或open命令，脚本将自动打开浏览器；如果不支持，会提示用户手动打开浏览器访问指定的URL。

运行流程图如下所示



图 10-2 总体运行流程图

### **10.4.2 流程运行展示**

下面将展示c/s模式下终端运行反馈展示：

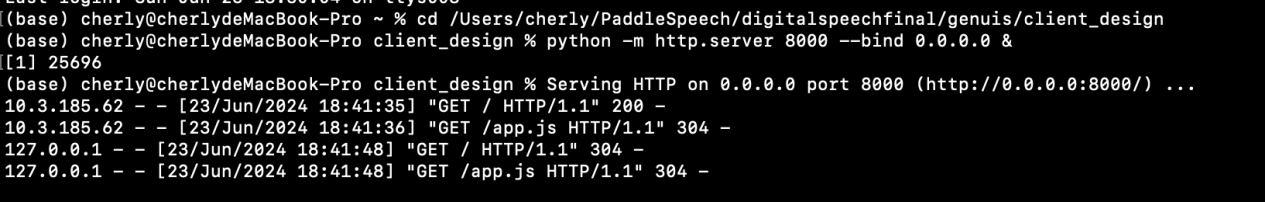


图 10-3 运行整体

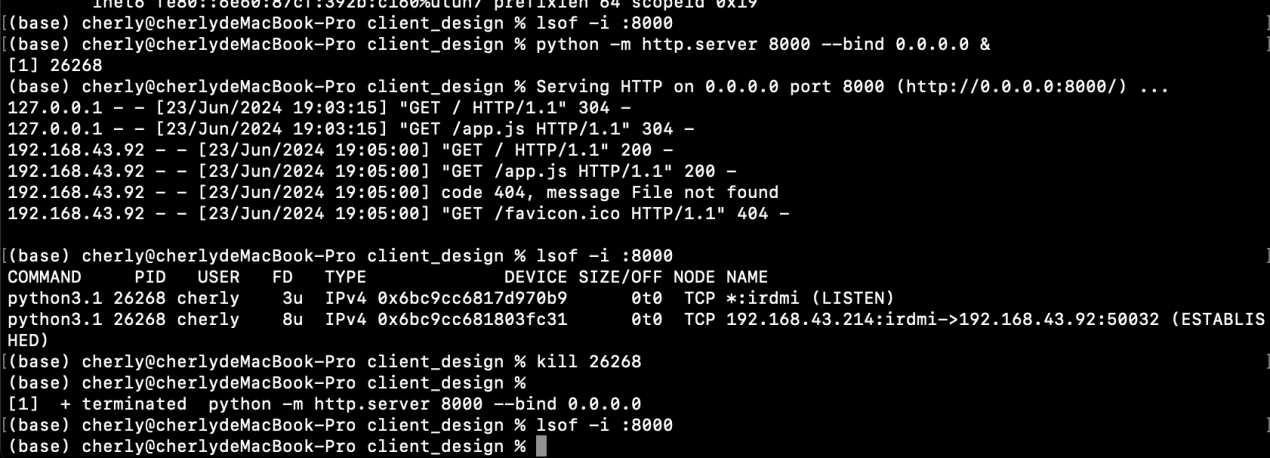


图 10-4 防止冲突

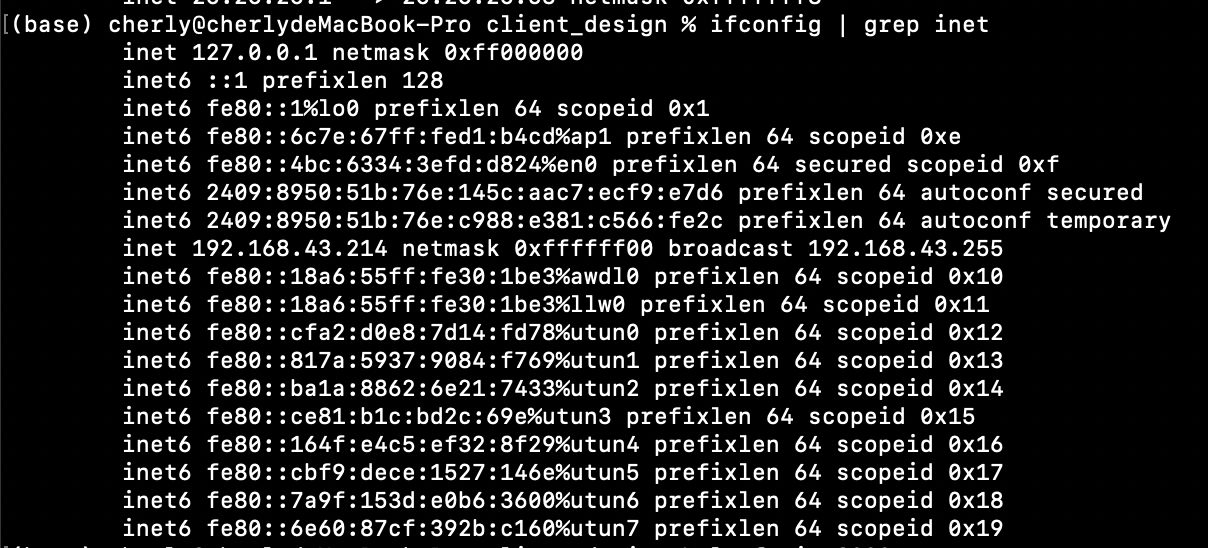


图 10-5 地址查询

### **10.4.3 运行问题解决方案汇总**

**（1）要检查客户端界面：**

1. 确保服务器正在运行

首先，确保你已经执行了 `run.sh` 文件，并且所有服务器进程（语音识别服务器和语音合成服务器）以及 HTTP 服务器已经成功启动。

1. 运行 `run.sh` 脚本

在终端中，导航到 `run.sh` 文件所在的目录，然后运行以下命令：

```bash ./run.sh

```

这个脚本会自动激活虚拟环境、启动服务器，并在本地启动 HTTP 服务器。

1. 检查本机浏览器

如果 `run.sh` 脚本成功运行，它会自动在你的默认浏览器中打开客户端界面。如果自动打开浏览器失败，可以手动打开浏览器并访问以下URL：

```

http://localhost:8000

```

1. 测试客户端功能

确保麦克风已连接并可以正常使用。

在客户端界面中，测试语音指令输入和相应的反馈功能。

验证语音识别和语音合成是否按预期工作。

（2）额外调试步骤

如果在运行过程中遇到问题，可以通过以下步骤进行调试：

1. 检查服务器日志

在运行 `run.sh` 的终端窗口中，检查服务器输出日志，看看是否有错误消息。

如果有问题，按 `Ctrl+C` 停止脚本并修复任何错误。

1. 手动启动服务器

如果 `run.sh` 没有按预期启动服务器，你可以手动启动每个服务器。

打开一个新的终端窗口，激活相应的虚拟环境并运行服务器脚本。例如：

```

conda activate paddle\_env

python /Users/cherly/PaddleSpeech/digitalspeechfinal/genuis/sever\_design/server\_recognize.py

```

然后，打开另一个终端窗口：

```

conda activate padpy\_9

python /Users/cherly/PaddleSpeech/digitalspeechfinal/genuis/sever\_design/server\_answer.py

```

再打开一个新的终端窗口启动 HTTP 服务器：

```

cd /Users/cherly/PaddleSpeech/digitalspeechfinal/genuis/client\_design

python -m http.server 8000

```

1. 检查网络设置

确保防火墙或网络设置不会阻止本地服务器的运行。

确认使用的端口（8000）没有被其他应用程序占用。

（3）手机端打开本地服务器页面：

实现多终端互通，能够让手机访问电脑上的本地服务器

1. 确保服务器在运行

首先，确认本地服务器已经成功启动，并且可以通过电脑访问。

在电脑浏览器中访问以下地址，确保页面正常显示：

```

http://localhost:8000

```

1. 获取电脑本地网络 IP 地址

为了让手机访问电脑上的本地服务器，需要使用电脑在局域网中的 IP 地址。

在 macOS 上获取本地 IP 地址：

1. 打开“系统偏好设置”。

2. 点击“网络”。

3. 选择当前连接的网络（如 Wi-Fi）。

4. 在右侧可以看到IP 地址

（4）在手机上使用本地网络 IP 地址

在手机浏览器中，访问你在上一步中获取到的 IP 地址：

```

http://192.168.1.100:8000

```

（5）确保电脑和手机在同一网络

确保电脑和手机都连接到同一个 Wi-Fi 网络。如果它们在不同的网络（例如一个在 Wi-Fi，另一个在蜂窝数据），手机将无法访问电脑上的本地服务器。

（6）检查防火墙设置

macOS 的防火墙设置可能会阻止手机访问本地服务器。你需要允许本地网络上的设备访问你的服务器：

- 打开“系统偏好设置”。

- 点击“安全性与隐私”。

- 选择“防火墙”选项卡。

- 如果防火墙已启用，点击“防火墙选项”。

- 确保 Python 解释器（或运行 HTTP 服务器的程序）被允许通过防火墙。

（7）检查服务器绑定的 IP 地址

确保 HTTP 服务器绑定到 0.0.0.0，而不仅仅是 localhost。这样，它会监听所有网络接口，而不仅仅是本地接口。

（8）确保没有端口冲突

确认端口 `8000` 未被其他服务占用。如果有其他服务使用这个端口，可以选择更改端口号，并在访问时使用新的端口号。

如果被占用，则检查端口占用：

要终止运行中的 HTTP 服务器以及所有相关的进程，可以按照以下步骤操作：

1.查找并终止后台进程

如果有后台运行的进程（如通过 `&` 符号启动），你需要找到这些进程并终止它们。

2.查找8000端口是否占用

在终端中运行以下命令来查找占用端口 8000 的进程

‘’‘lsof -i :8000’‘’

3.找到进程并kill

‘’‘kill 12345 # 使用实际的PID替换1234’‘’

## **总体效果展示**

### **10.5.1 客户端页面展示**

**（1）PC端页面展示**

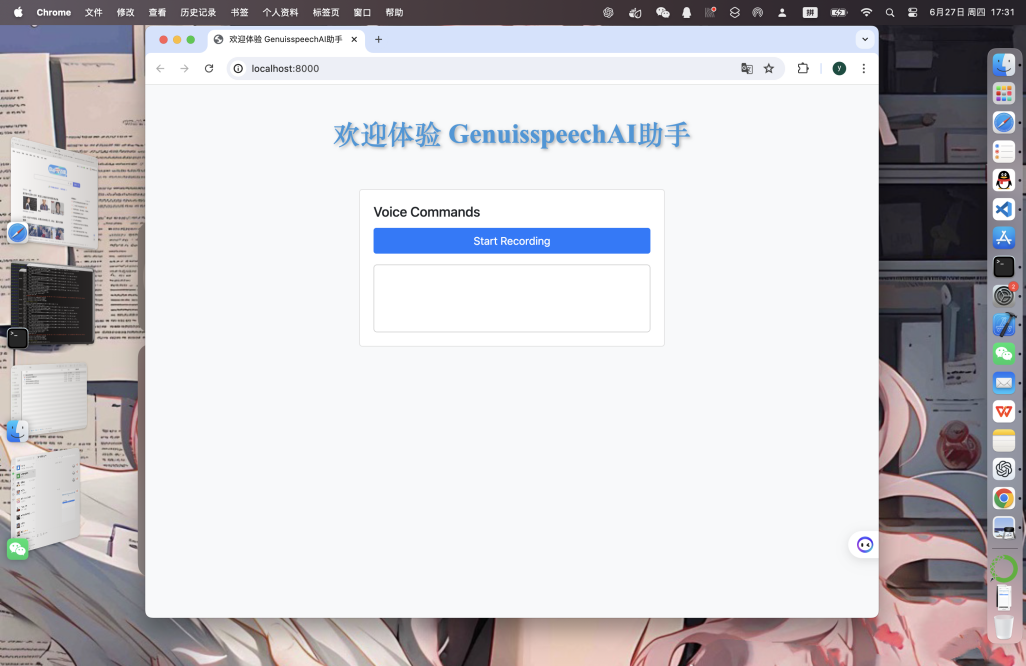
****

图 10-6 PC端页面展示

**（2）手机端页面展示**

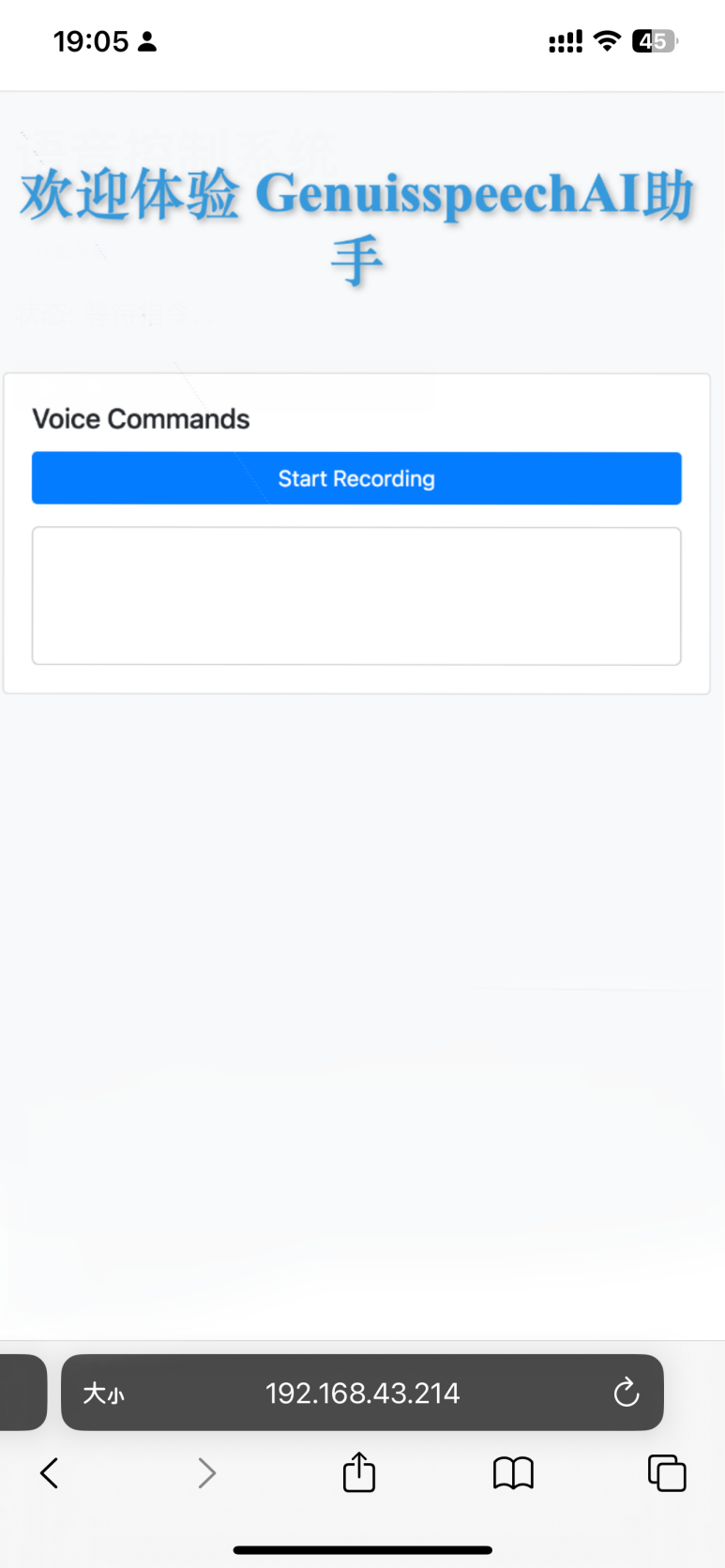
****

图 10-7 手机端页面展示

### **10.5.2 后端展示**

在终端中反馈后端信息，如下图所示

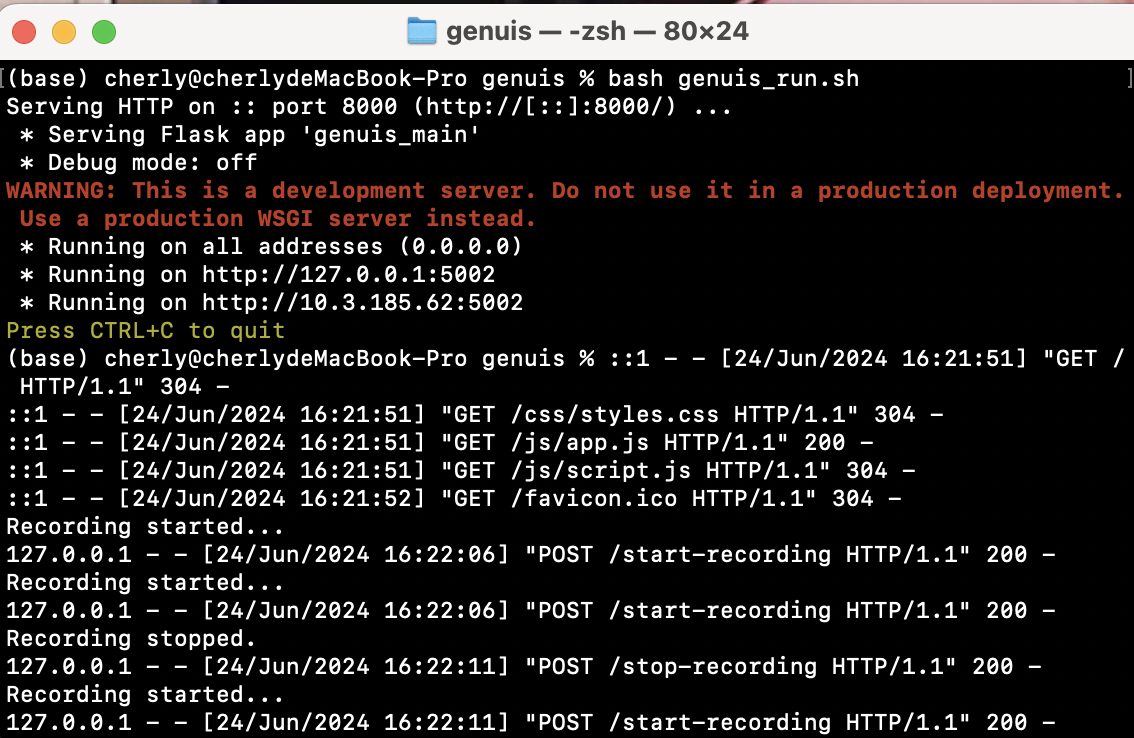


图 10-8 后端展示

1. **系统交互结果分析**

## **语音识别项目结果**

本项目训练的Deepspeech2模型可以实时识别语音文本。为测试模型在两种情况下的准确率，本项目准备了6条噪声场景下的语音文件，如下图所示识别率达到95%符合要求的在噪声场景下（SNR<10dB）识别率不低于80%，更无需讨论在清晰语音环境下识别率是否能够达到100%。结果展示图11-1如下



图 11-1 语音识别项目结果

## **语音合成项目结果**

### **11.2.1 效果展示**

本组的项目中可以实时根据语音识别的请求信息，反馈给语音合成脚本的借口并下达对应文本内容合成的语音来播报实时反馈给使用者。下图展示了Fastspeech2模型实时反馈合成的语音演示图：



图 11-2 语音合成项目结果

### **11.2.2 合成音质与自然度方面**

在合成音质与自然度方面，本组实验过程中先是用成员自行录制的语音进行训练，经过不断的调整和改进模型动态推理的参数合成的音效十分类似于成员个人音色。但由于采样设备的不充分与有限，故在自然度方面有所欠缺。

故本组又紧急采取公开音频数据集进行采样训练，模型训练效果十分优秀，合成的音质有音调与音色，十分的自然流畅。

1. **总结与展望**

## **项目总结**

基于PaddleSpeech框架，本项目通过深度学习技术的精湛应用，成功实现了中文语音识别与合成系统。在技术上，该项目构建了能够精准识别日常中文指令的语音识别模型，并研发了能将中文文本转换成流畅、自然语音的合成模型，展现了在语音识别与合成领域的深厚实力。

在数据集应用方面，项目充分利用了AIShell和CSMSC等大规模中文语音数据集，为模型训练提供了丰富的数据资源。同时，还包含了团队成员自行录制的音频数据，这些数据进一步增强了数据集的多样性，有助于模型适应更多实际场景，提升了泛化能力。

在模型优化方面，项目团队通过精心调整超参数、优化数据预处理流程，并采用合适的正则化技术，显著提升了模型的准确率和泛化能力。这些优化措施确保了模型的高效性和稳定性，使得系统在实际应用中表现出色。

此外，该项目还将语音识别和合成模块无缝集成到一个统一的系统中，实现了从语音输入到文本识别，再到语音输出的完整交互流程。这一成果不仅展示了在系统整合方面的能力，也凸显了项目在中文语音识别与合成领域的全面能力和显著成果。

## **未来工作展望**

展望未来，研发工作将聚焦于多个关键领域，力求实现技术的新突破。将不断探索前沿的神经网络架构，以提升语音识别和合成的精确度与品质。积极研发新型数据增强技术，确保模型在各种语音环境下都能表现出色。为拓宽应用范围，将增强系统的多语言和特殊方言的处理能力[3]，以适应更多场景。

为满足实时语音交互的高标准，将持续优化模型，减少响应时间。还将融入先进的语言理解技术，使系统交互更加自然智能[4]，为用户提供更出色的体验。此外，计划将技术应用于教育、医疗、智能家居、VR导游等多个领域[4]，以满足不同行业的实际需求。同时将专注于模型压缩和加速技术，确保模型在资源有限的设备上依然能够高效运行。这些努力将推动技术革新，为社会带来前所未有的便利与价值。

**参考文献**

1. 肖爱民.基于语音识别技术的智能家居控制系统的设计[D].南昌大学,2018.
2. 宋志浩.基于完全端到端方法的藏语拉萨话语音合成技术研究[D].西北民族大学,2023.DOI:10.27408/d.cnki.gxmzc.2023.000290.
3. 孟凌霄.“秒懂”30种方言中国电信用人工智能守护方言[N].中国科学报,2024-05-27(003).DOI:10.28514/n.cnki.nkxsb.2024.001096.
4. 凌志程,陈宇军,王珊珊,等.开展智能语音交互系统能力验证项目的可行性研究[J].日用电器,2024,(04):69-72.
5. 刘鹏.智能语音识别、合成、评测技术运用在导游智能教学中的效果分析[J].中国新通信,2023,25(17):107-109.