5.3文本纠正样例

|  |  |
| --- | --- |
| 原始文本 | 那年冬天竹母死了父亲的差事也交谢了正是祸部单行的日子 |

|  |  |
| --- | --- |
| 纠正文本 | 那年冬天祖母死了父亲的差事也交谢了正是祸不单行的日子 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| target | candidate | cos\_sim | pinyin\_sim | ngram\_sim |
| 竹 | 祖 | 0.6100631 | 0.7692307692 |  |
| 父 | 0.5683848 | 0. |  |
| 异 | 0.42654333 |  |  |
| 母 | 0.55227005 |  |  |
| 亲 | 0.5160396 |  |  |
| 部 | 不 | 0.6238163 |  |  |
| 补 | 0.61413115 |  |  |
| 女 | 0.48703188 |  |  |

[{'

char': '祖', 'cos\_sim': 0.6100631, 'pinyin\_sim': 0.7692307692307692,

'ngram\_sim': 3.2518814545525276},

{'char': '父', 'cos\_sim': 0.5683848, 'pinyin\_sim': 0.3846153846153846, 'ngram\_sim': 3.9463048549934747},

{'char': '异', 'cos\_sim': 0.42654333, 'pinyin\_sim': 0.2777777777777778, 'ngram\_sim': 2.315970345456918},

{'char': '母', 'cos\_sim': 0.55227005, 'pinyin\_sim': 0.4347826086956522, 'ngram\_sim': 1.0791812460476249},

{'char': '亲', 'cos\_sim': 0.5160396, 'pinyin\_sim': 0.30303030303030304, 'ngram\_sim': 1.591064607026499}]

[{'char': '不', 'cos\_sim': 0.6238163, 'pinyin\_sim': 10.0, 'ngram\_sim': 2.69284691927723},

{'char': '补', 'cos\_sim': 0.61413115, 'pinyin\_sim': 3.3333333333333335, 'ngram\_sim': 0.3010299956639812},

{'char': '女', 'cos\_sim': 0.48703188, 'pinyin\_sim': 0.4347826086956522, 'ngram\_sim': 2.5224442335063197}]

：

5.4 标点符号添加

语义识别模块输出的识别结果为原始的文本序列，只有汉语文字，不包含标点符号，而本文的自动字幕功能是需要将生成的文本字幕显示到用户客户端供用户浏览的，若字幕语句中没有标点符号停顿，会极大的影响阅览体验，所以本文在后处理阶段对文本进行标点添加操作，提高字幕结果的可读性。

标点添加功能具体的实现流程为，搭建一个深度学习模型预测标点符号的添加位置以及具体添加哪种标点符号，之后通过标点添加逻辑对语音识别模块的流式文本输出进行标点添加。

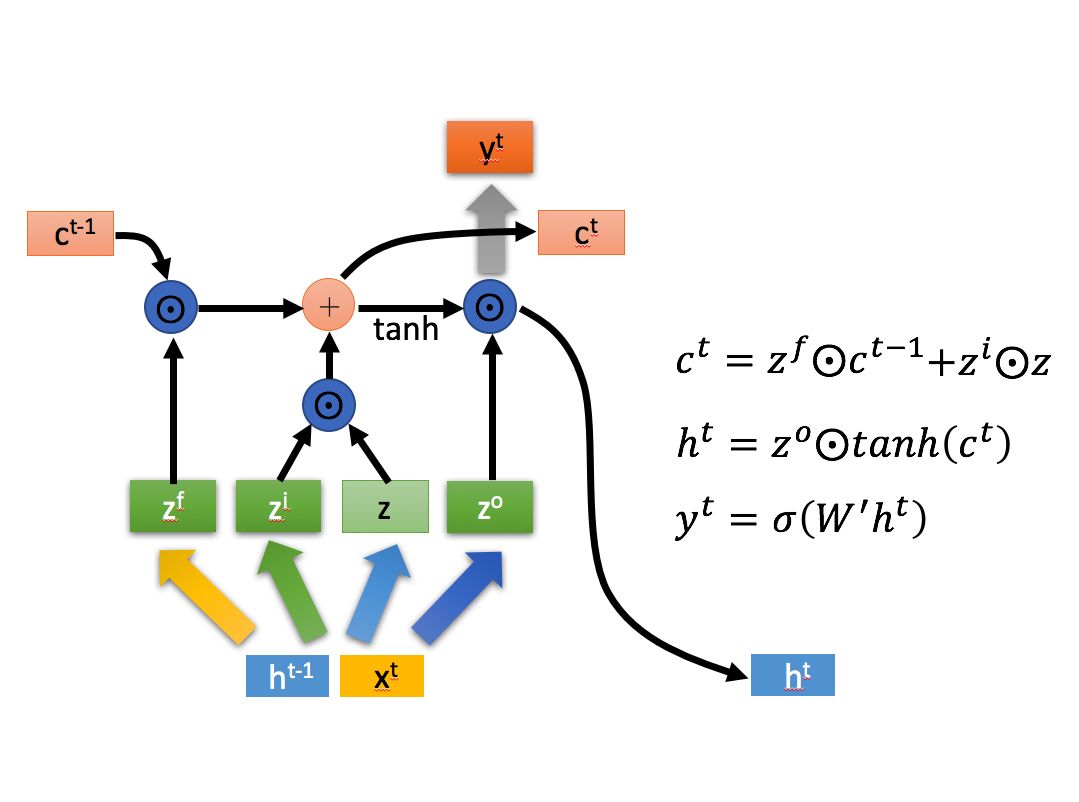
5.4.1 标点预测模型

首先是构建标点符号标注数据集，本文使用的语料数据同5.3章节用到的wiki中文语料相同，只是在其基础上进行了分词和标点符号标签标注工作。本文将标点符号添加任务抽象成序列标注问题，因为中文语句中是有字和词的区别，一般标点符号都是存在于一个完整词语的后面，所以首先对语句进行分词操作，一个词作为一个输入序列的token，对于每一个token，都有一个对应的符号标签，若这个token之后是一个标点符号，则将此标点符号作为此token的标签，若其之后不是标点符号，则将其标注为空白标签X，本文提供支持的标签序列为[**'X'**, **'，'**, **'。'**, **'！'**, **'？'**, **'、'**, **'：'**]，其标注数据集格式样例如表5.1所示。

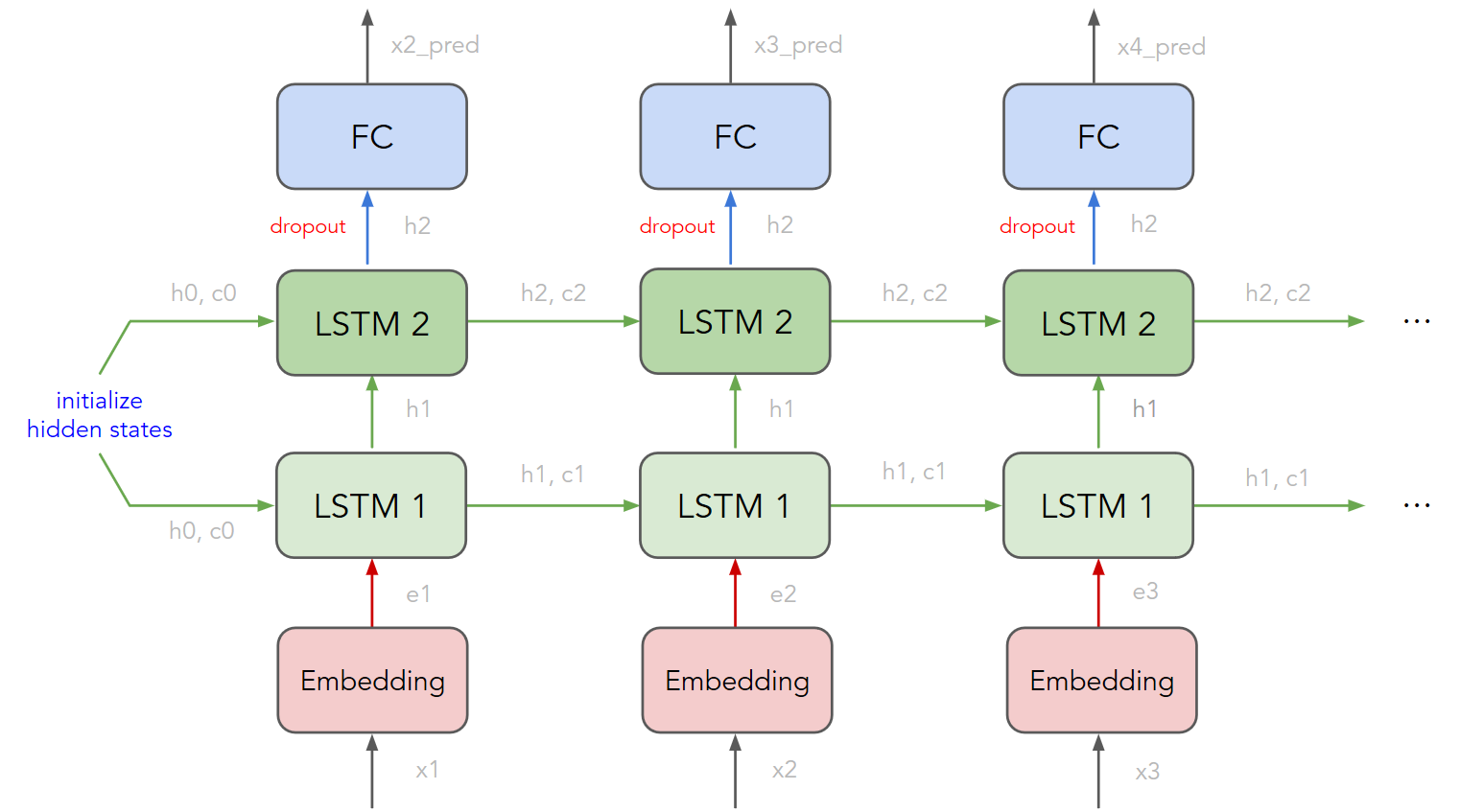
|  |  |
| --- | --- |
| 输入x | 标签y |
| 中国 河北省 一 烟花场 发生 爆炸 造成 多人 死亡 | X X X X X ， X X 。 |
| 第七届 世界 华商 大会 在 马来西亚 开幕 | X X X X X X 。 |
| 美军 占领 伊拉克 美军 今天 在 搜捕行动 中 遇袭 两人 受 重伤 | X X : X X X X X ，X X 。 |

本文使用的深度学习预测模型为BiLstm，对于序列标注问题双向循环神经网络RNN可以记忆提取上下文信息，考虑各个token之间的相关性，从而提高对当前时间步token的理解预测能力。但是传统的循环神经网络在考虑远距离历史信息时，由于训练过程中反向传播梯度消失问题，很容易将长期记忆特征丢失掉，不能考虑到全局语义，所以本文使用引入逻辑门控结构的长短期记忆单元Bilstm网络模型，其单元内部结构和计算公式如图5.7所示，主要是通过三个逻辑控制门输入门，遗忘门和输出门来控制时间步单元对当前信息，历史信息和当前输出的考虑权重，缓解RNN中各个时间步参数累乘造成的梯度消失问题，

提高模型的长期记忆能力和对序列的拟合理解能力。

本文基于wiki语料分词统计构造训练词典，共有352219个词条，如果和预测模型统一训练embedding，收敛会很慢，所以本文使用了开源项目Chinese Word Vectors（2018 Analogical Reasoning on Chinese Morphological and Semantic Relations）提供的预训练词向量sgns.wiki，其基于wiki中文语料使用word2vec模型训练所得，每一个词用300维向量表示，在本文的标点添加模型训练过程中，通过加载此预训练词向量，可以极大提高训练收敛速度。

本文的标点添加模型结构如下图5.8所示。

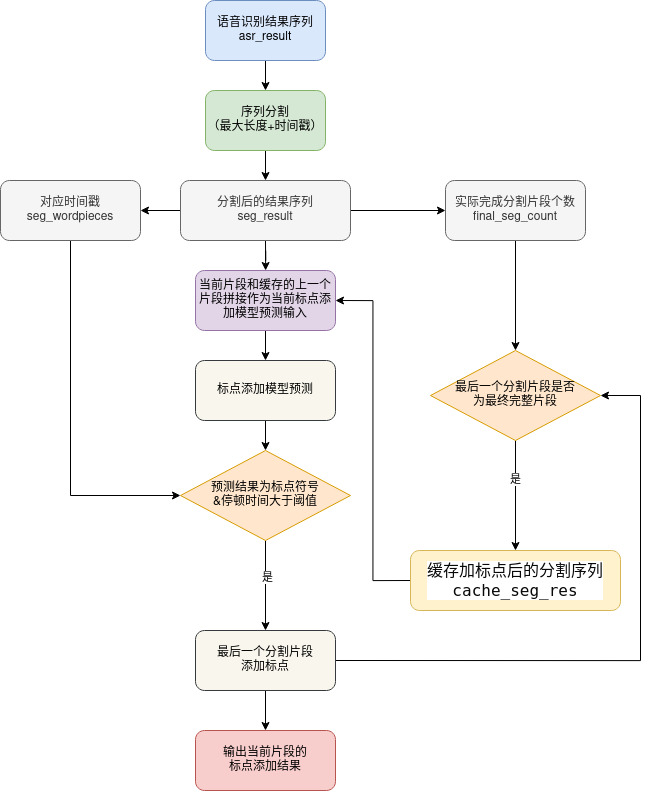


原始的输入token序列经过embedding层获得对应的嵌入向量表示，再经过两层Lstm层抽取上下文信息，输出高层稠密信息表示的隐状态向量，最终通过一个全连接层将输出映射到任务标签维度，得到预测结果。

5.4.2标点添加逻辑

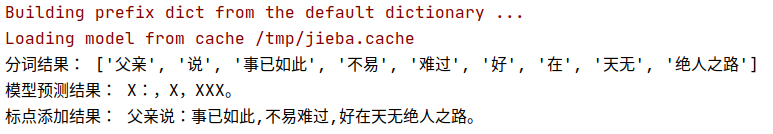
因为语音识别模块的输出方式是流式输出，每次返回结果会和上一次结果有重复的片段序列，而标点添加模型需要尽可能长的序列信息，才能考虑全局的上下文语境 ，得到较好的标点预测效果，但同时考虑模型推理的时间性能问题，输入也不能太长，所以折衷考虑两方面因素，本文设计了一些规则逻辑来实现流式序列标点添加功能。

具体的添加逻辑流程如图5.9所示。

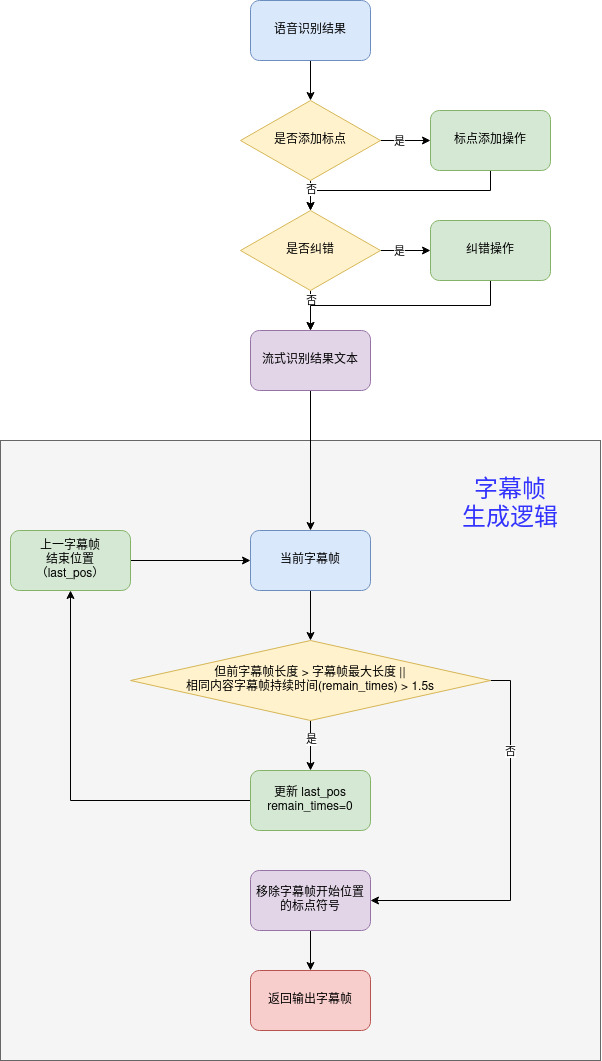


首先根据语音识别模块返回的时间戳信息计算每个字之间的时间间隔，根据此间隔大小和最大长度阈值等规则对识别结果进行分割，得到分割后片段的序列结果和对应的时间间隔信息，以及当前已经识别的完成的分割片段个数。然后将当前片段和缓存结果中最后一个片段拼接，目的是在标点添加模型预测过程中能够考虑到更多的语境信息，然后将此拼接结果序列通过模型预测得到预测结果，同时在预测为标点的位置加入停顿间隔时长判断，若停顿时长小于最低阈值，证明此处预测结果不可信的概率较大，则去除添加此标点，较少错误识别。接下来根据完整分割片段个数确定是否将当前片段加入缓存。最终将当前添加标点符号之后的序列片段返回输出。

下图5.9为测试样例的标点识别结果。



5.5 生成字幕帧

对于原始的语音识别结果，经过后处理的标点添加和纠错处理之后，得到了准确率较高的字幕文本结果，但是此结果是连续的文本序列，考虑到视频播放客户端的字幕显示实际效果，需要进行一些逻辑规则处理，将流式文本转化成文本字幕帧，以供播放器周期性加载显示。

具体的字幕帧生成逻辑如下图5.10所示。

首先根据记录的上一帧字幕的结束位置下标last\_pos生成当前字幕帧，然后进行一个循环判断，判断的逻辑为当前字幕帧长度大于播放客户端在设定字幕字体大小的条件下能显示的最大长度，或者当前内容的字幕帧重复保持的时间大于1.5s，则使用上一个字幕帧长度更新last\_pos，进而重新生成当前字幕帧，知道不满足判断条加跳出循环，然后判断当前字幕帧的开始位置是否为标点，若为标点则去除，防止影响字幕显示效果，最后将字幕帧发送给客户端播放器进行周期性的渲染显示。

5.6本章总结

第一章简单介绍了原始语音信号预处理操作，包括音频降噪和通道采样率对齐等。

第二章介绍了在自动字幕生成流程中对语音识别服务的请求调用逻辑和程序代码实现的简单流程。

第三章详细介绍了对原始语音识别结果的文本纠错处理，包括文本纠错的常用算法，本文实现的纠错模型和纠错逻辑，同时使用测试样例对纠错效果进行了展示论证。

第四章本文也详细介绍了标点添加模块，分析了标点添加的功能需求，描述了数据集构建和模型实现过程，以及标点添加的逻辑流程。

第五章本文整体梳理了自动字幕生成的基本流程和字幕帧生成的具体规则逻辑。

第六章 系统测试与分析 12

基于上述章节介绍的自动字幕模块需求分析及系统设计实现，本文从功能和性能两个角度设计了基本的测试实验，科学全面的验证本文的自动字幕模块的实现效果，进而发现问题，总结经验，为进一步的提高优化提供基础。

6.1 测试环境配置

本系统基于Docker容器进行语音识别服务后端部署，基于Python脚本语言进行后处理和字幕生成功能实现，能够跨平台灵活部署，具有可移植，轻量化的特点。为了验证本系统的最低配置运行环境，本文基于linux系统的本地环境展开测试验证实验，而具体的线上服务的运行环境配置比本地环境配置要高，所以其具体的运行效果比本测试实验效果要有更好的预期。

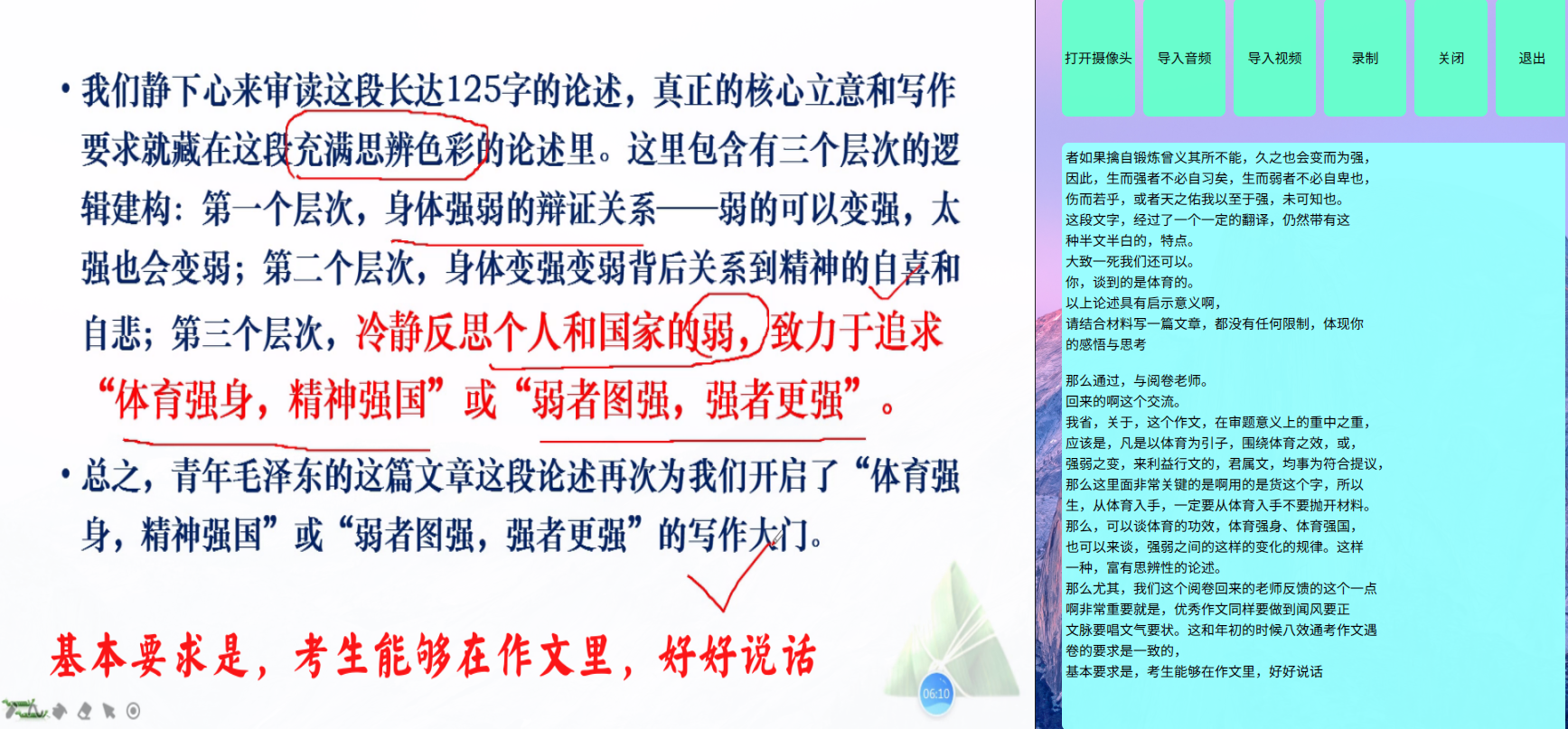
具体的环境参数配置如下表6.1所示。

|  |  |  |
| --- | --- | --- |
| 配置参数 | 本地测试环境 | 线上生产环境 |
| 操作系统 | Ubuntu18.04 | Ubuntu18.04 |
| CPU内核数 | 8 | 16 |
| 内存大小 | 16GB | 32GB |
| 文件存储系统 | EXT3 | FastDFS |
| Docker版本 | 20.10.12 | 20.10.12 |
| Python版本 | 3.7.8 | 3.7.8 |
|  |  |  |

6.2 GUI 测试程序

为了测试验证实验的展开以及功能效果演示，本文基于pyqt框架开发了一个图形界面测试程序，主要实现了在线语音识别，离线语音识别，离线视频字幕自动添加等功能。

下图6.1是测试程序的自动字幕生成功能展示图。



本测试程序的图形界面主要分为三个功能区，其中左侧是视频播放区域，主要完成在线摄像头打开，离线视频播放以及字幕帧展示等功能，右下侧是字幕文本的显示区域，右上侧是功能菜单选项区域。

6.3 功能测试

6.1.1语音信号预处理

本测试模块主要验证对原始的输入音频信号的降噪，通道和采样率转换操作是否正确，确保语音识别模块输入的语音信号具有统一的音频格式。

具体的测试用例如下表6.2所示。

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例名称 | 前置条件 | 预期结果 | 测试结果 |
| 音频降噪 | 包含白噪声的音频信号 | 过滤噪声后的高信噪比音频信号 | 通过 |
| 通道转换 | 单通道，双通道，多通道音频信号 | 统一转换为单通道音频信号 | 通过 |
| 采样率转换 | 不同采样率的音频信号 | 统一转换为16k采样率的音频信号 | 通过 |

6.1.2语音识别服务请求

按照4.3章节介绍的操作步骤，将语音识别程序进行Docker镜像打包和服务部署，验证其操作流程的是否正确，确保语音识别服务能正常启动。

具体的测试用例如下表6.3所示。

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例名称 | 前置条件 | 预期结果 | 测试结果 |
| 语音识别镜像打包 | 开发好的语音识别代码已上传至指定的远程代码仓库。 | 通过docker images命令可以查看到生成好的asr\_wenet\_lm镜像。 | 通过 |
| 语音识别服务部署 | 打包好的镜像文件。 | 运行语音识别服务启动脚本，可以看到服务成功启动。 | 通过 |
| 语音识别服务请求 | 部署成功的Docker语音识别服务 | 通过运行Grpc客户端测试程序，发送音频信号识别请求，返回对应的语音识别结果。 | 通过 |

6.1.3文本后处理和字幕生成

本测试模块主要是验证本系统对于语音识别结果的纠错，标点添加以及字幕帧生成操作是否正常。

具体的测试用例如下表6.4所示。

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例名称 | 前置条件 | 预期结果 | 测试结果 |
| 文本纠错 | 带有谐音错误的语音识别结果文本  训练好的文本纠错模型 | 对于常见的谐音错误可以准确识别并纠正。 | 部分通过 |
| 标点添加 | 无标点的原始语音识别结果文本 | 按照中文语法语义逻辑添加合适标点。 | 通过 |
| 字幕帧生成 | 添加标点后的语音识别结果文本 | 字幕帧和声音对齐，且无分割显示错误。 | 通过 |

6.1.4 在线字幕生成测试

本测试模块通过启动音频输入设备，在线测试对流式输入的实时识别效果，验证本自动字幕生成模块的整体功能是否正常。

测试用例如下表6.5所示。

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例名称 | 前置条件 | 预期结果 | 测试结果 |
| 在线识别测试 | 音频输入设备  前端测试程序 | 实时识别的结果文本和声音对齐，连续识别1小时后无异常 | 通过 |

6.4 性能测试

6.4.1空间性能

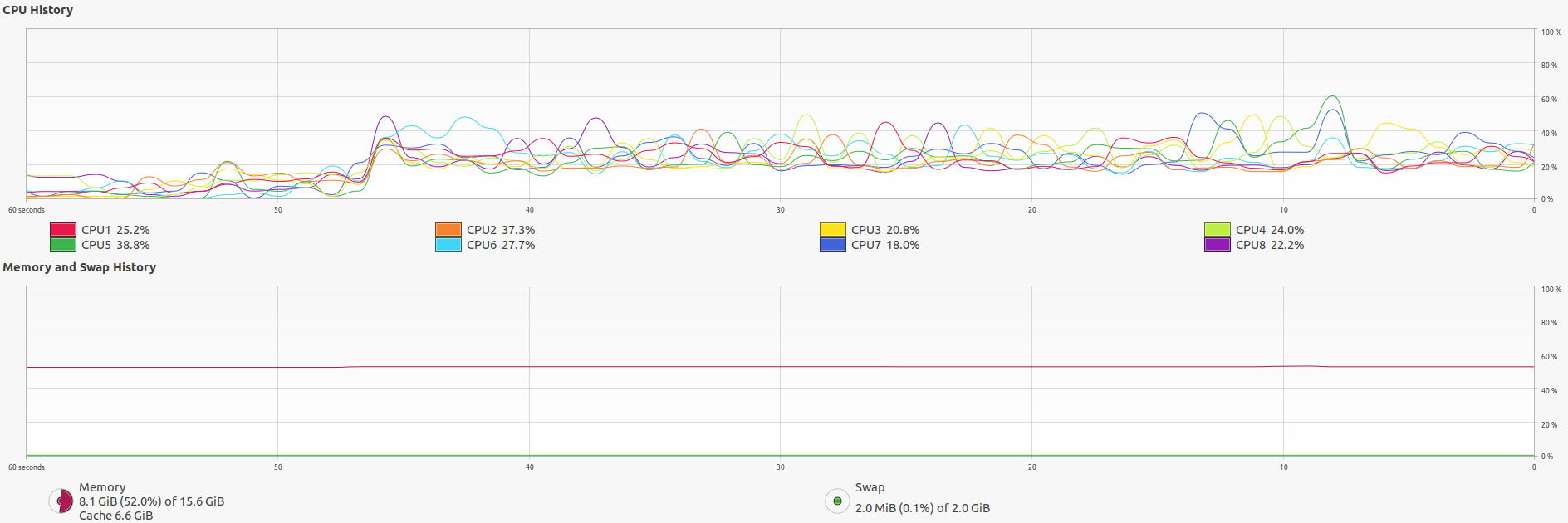
本测试模块主要考察在在线字幕自动生成场景下，服务程序运行对机器CPU和内存的使用情况。

本文使用htop命令工具进行机器cpu及内存的监控测试，其中显示界面上侧部分是当前机器cpu和内存的占用百分比，下侧部分是所有运行程序的资源使用情况。

具体的自动字幕生成程序在执行流式语音识别任务时的资源占用情况如下图6.2所示。



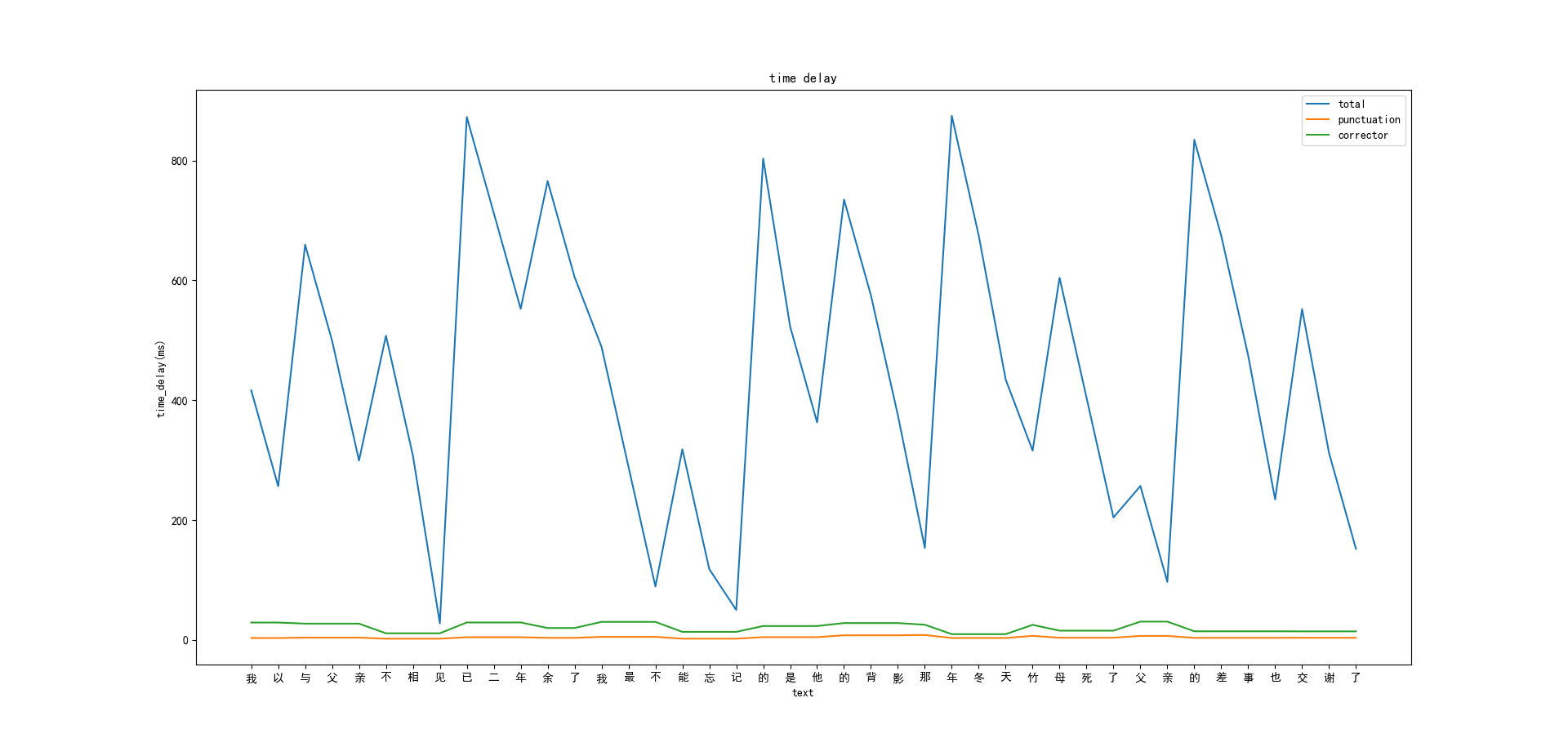
由上图6.2可知，本文的运行程序主要由两部分组成，一个进程是grpc\_server\_main，为语音识别服务程序，其cpu占用率为18.8%，内存占用率为8.6%。另一个进程是python脚本程序grpc\_client\_demo，是客户端测试程序，主要包括语音识别后处理模块和字幕生成模块，其单核cpu占用率为18.625%，内存占用率为18.0%。可以看出，语音识别服务程序和客户端测试程序的单核cpu占用率都低于20%，满足系统模块的部署要求。另外为了测试程序运行对系统资源占用情况的稳定性，我们对程序运行状态进行了连续观察，在持续1小时时间内无异常资源使用情况波动，程序稳定运行。下图6.3为程序运行1分钟内的cpu内存占用情况示意图。



6.4.2时间性能

对于实时在线语音识别任务，最重要的指标之一就是识别结果的时间延迟，只有较低的识别时延才能保证系统具有较好的实用性，从而使用户具有较好的使用体验。所以本文设计了字幕生成时间延迟测试实验来准确量化系统运行的时间性能，分析各个数据处理模块的延迟开销，从而为后续优化提供方向基础。

下图6.4为语音识别时间延迟测试结果样例图。



本文首先获取语音识别结果文本对应的原始时间戳，并在客户端程序中记录全局时间戳，从而计算每个识别结果文字的识别时间延迟，然后对测试结果时延进行统计计算，最终本语音识别系统的平均识别时延为455.76ms，时延的抖动标准差为232.80ms，而标点添加处理的平均耗时为5.16ms，纠错处理的平均时延为18.16ms，均满足系统设计要求，达到可用效果。

6.4本章小结

本章主要对本文设计的自动字幕模块进行系统测试，从功能和性能两个角度验证其在实际应用场景中的可用性和稳定性。

第一节简单介绍了测试实验运行的本地环境配置，同时和线上生产环境作了基本对比。

第二节简单展示了测试实验用到的图形界面运行程序及其基本功能。

第三节详细介绍了各个功能模块的具体测试用例及其测试结果。

第四节从空间性能和时间性能两个方面讲述了系统运行时的性能状态，并对测试结果进行简单分析。

第七章研究总结与启示

7.1研究总结

基于智慧教室中直播课程字幕添加的现实需求，本文设计并实现了自动字幕生成系统，并将其应用集成到智慧教室产品中，提升用户的使用体验，完善产品功能。

本文首先分析了当前语音识别技术和自动字幕生成技术的研究现状，最终选择基于深度学习的端到端语音识别框架Wenet作为基础语音识别模型，然后结合教育领域的特殊性质，引入了语言模型，热词增强模型，标点添加模型和纠错模型等各个优化处理模型，同时使用C++编程语言开发实现语音识别模型推理过程，完整构建了自动字幕生成链路，最终使其达到产品化应用水平。

以实际场景需求为目标，我完成了自动字幕模块从功能开发，接口设计到部署调试等完整产品设计实现流程，在这个将专业知识应用到产品开发的过程中，通过学习其他文献资料，我学到了很多新的专业技术和开发工具，包括语音识别技术，语音分割技术，流媒体技术以及容器化部署技术等，另外更重要的是，在老师和学长的指导帮助下，我掌握了现实产品的标准开发测试流程以及解决问题的科学思路方法，这对我之后的工作生活都收益良多。

7.2未来改进方向

经过半年的学习研发，本文已经基本实现了自动字幕生成的基本功能，但是由于个人技术水平和时间的限制，本模块中还有很多问题需要解决优化，在此总结了以下几个方面可以继续改进提高。

1.数据层面

目前语音识别模型训练用的是TAL数据集，语言模型和纠错模型用的是Wiki语料数据，这和我们的智慧教室业务数据特征有很大不同，所以需要花费精力对智慧教室课程进行人工字幕标注，构建自有业务数据集，使训练集和实际业务数据具有相同分布，从而提高各个模型的识别准确率。

2.模型算法层面

本文目前使用的语音识别模型为U2模型，但是对于实时字幕生成，第二阶段的重打分操作耗时较多且在字幕切帧后失去意义，需要优化重打分时机，同时使用动态Chunk机制，提高流式识别效率。另外当前语言模型和语音识别模型采用shallow fusion解耦设计，为了更好的提高两个模型的配合协调能力，可以考虑引入cold fusion技术，使用领域业务数据微调训练时就引入语言模型，加快模型训练过程，提高两个模型的关联融合效率。

3.实现部署层面

目前语音识别服务已经进行了Docker部署，而纠错和标点添加等后处理模块仍然耦合在字幕生成前端程序中，可以考虑将这些模块都进行单独的服务部署，这样可以将这些服务都作为一个可插拔的模块灵活使用，但也同时要考虑服务通信引入的时延对实时字幕生成产生的时间性能开销等问题。