

中科院声学所离线识别引擎

工程使用说明

目录

[1 概述 4](#_Toc83915106)

[2 编译环境 5](#_Toc83915107)

[3 整体框架 6](#_Toc83915108)

[4 主要源代码介绍 7](#_Toc83915109)

[5 函数调用流程 9](#_Toc83915110)

[5.1“单线程单解码器”函数调用流程 9](#_Toc83915111)

[5.2“多线程多解码器”函数调用流程 9](#_Toc83915112)

[5.3 解码器函数调用流程 10](#_Toc83915113)

[如何正确运行 11](#_Toc83915114)

[附录 11](#_Toc83915115)

[A 接口函数调用流程 11](#_Toc83915116)

版 本 历 史

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本/状态 | 作者 | 参与者 | 起止日期 | 备注 |
| 1.0 | 陈向东 | 龚云波  黎塔 | 2021/09/30 |  |
|  |  |  |  |  |
|  |  |  |  |  |

## 1 编译环境

（1）编译器要求：

安装4.7以上版本的gcc和g++

（2）机器环境配置：

要求64位linux系统，Centos或Redhat 6.3以上企业开发版

（3）编译步骤：

DNN版本，进入离线/离线语音识别引擎-总模块/TBNR\_APII目录执行：

make – f Makefile\_TBNR\_new\_dnn

即可生成TBNR动态库：libTBNR\_API.so

（4）依赖库

依赖解码器生成的静态库，位于TBNR\_API\release\_lib\_wfst\_dnn目录下：

librecengine.a：解码器生成的静态库，已提供源码

libsrilm.a：解码器生成的静态库，已提供源码

libtools.a：解码器生成的静态库，已提供源码

libtshare.a：解码器生成的静态库，已提供源码

libwfstdecoder.a：解码器生成的静态库，已提供源码

（5）第三方依赖库

依赖intel 加速库，对应的so位于TBNR\_API/RelyOn/ipp\_lib/linux下，包括：

libiomp5.so： Intel公司发布的通用IPP函数库，用于加速计算，我们没有源码。

libippcoreem64t.so：Intel公司发布的通用IPP函数库，用于加速计算，我们没有源码。

libippcoreem64t.so.6.0：Intel公司发布的通用IPP函数库，用于加速计算，我们没有源码。

libippscem64t.so.6.0： Intel公司发布的通用IPP函数库，用于加速计算，我们没有源码。

libippsem64t.so：Intel公司发布的通用IPP函数库，用于加速计算，我们没有源码。

libippsem64t.so.6.0： Intel公司发布的通用IPP函数库，用于加速计算，我们没有源码。

libippsmx.so.6.0：Intel公司发布的通用IPP函数库，用于加速计算，我们没有源码。

libippsrem64t.so： Intel公司发布的通用IPP函数库，用于加速计算，我们没有源码。

libippsrem64t.so.6.0：Intel公司发布的通用IPP函数库，用于加速计算，我们没有源码。

libippsry8.so.6.0：Intel公司发布的通用IPP函数库，用于加速计算，我们没有源码。

## 2 整体框架



如上图所示，首先对输入的音频数据进行端点检测，获取逐句的有效语音数据，然后将其送入特征提取模块，接着对于提取的声学特征进行解码，解码过程采用transformer的decoder输出进行字符同步的束搜索算法，在搜索空间内寻找匹配概率最大的最优路径，得到识别结果。

## 3 主要源代码介绍

TBNR\_DNN目录：用于编译生成libTBNR\_API.so

|──Makefile.common.variables：设置TBNR\_API下Makefile\_TBNR\_new\_dnn编译选项

|──include\_wfst：链接解码器需要头文件

| |──onepass.h：定义解码器类

|──TBNR\_API目录：编译目录

| |──Makefile\_TBNR\_new\_dnn：Makefile文件

| |──TBNR\_API.cpp : 实现TBNR各功能模块的组合，对外接口API。

| |──CTSRBuffer\_new.cpp： 通过类CTSBuffer控制数据流程。

| |──DNN目录， 该目录实现主要是基于深度歇息的端点检测功能

| | ├──DNN\_VAD.h DNN端点检测头文件

| | |──libDnnVad.a DNN端点检测实现库，该库的实现在”离线识别子模块-DNNVAD”中进行信息描述

| |──BackDetector目录，对分段数据进行二次判决，增加有限分段数据的可靠性

| | |──VAD\_Interface.cpp 二次判决源文件

| |──DTMFDetect按键音检测源文件功能目录

| | |──DTMF\_Detection.cpp 按键音检测源文件

| |──RingDetect目录，该目录实验彩铃检测

| | |──RingDetect.cpp彩铃检测接口源文件

| |──CreateFeature目录， 该目录实现主要功能是提取特征

| | |──CreateFeature.cpp:实现提取语音特征；

| | |──onlinecms.cpp: 进行倒谱均值减

| | |──PLP\_NCC.cpp：提取PLP及基频残差。

| | |──rsrfft.cpp：实现分裂基fft算法

| | |──plp.cpp：提取plp特征（FBank方法）

| | |──PitchTracker.cpp：提取基频模块，利用频域算法提取基频

| | |──mfcc\_hires.cpp：提取MFCC模块，实现提取MFCC特征

| |──highpass，该目录主要功能是进行高通滤波

| | |──Highfilter.cpp：实现高通滤波器相关函数

| | |──RecResultDeal，该目录主要为识别结果后处理

| |──GBK2UTF8：识别结果GBK转UTF8编码

| | |──CodeForamtConversion.cpp 转码函数源文件

| | |──libiconv.a GNU一部分基于[GPL](https://baike.baidu.com/item/GPL)公开源代码进行编码格式之间的转换库

| |──release\_lib\_wfst\_dnn，该目录实现解码依赖的静态库

| | |──librecengine.a：解码器编译生成，已提供源码

| | |──libsrilm.a：解码器编译生成，已提供源码

| | |──libtools.a：解码器编译生成，已提供源码

| | |──libtshare.a：解码器编译生成，已提供源码

| | |──libwfstdecoder.a：解码器编译生成，已提供源码

| |──RelyOn：该目录下依赖的intel ipp库及头文件

| | |──RelyOn\ipp\_header\emt64t：64位用该目录头文件，引擎目前用的该目录。

| | |──RelyOn\ipp\_lib\linux：ipp动态库，详细介绍参照依赖库介绍。

| |──resample:该目录实现重采样相关算法

| | |──filter.cpp：实现滤波相关算法

| | |──Rtresample.cpp：实现重采样相关算法

| |──HotWord：热词功能模块

| | |──Word2PhnIdx.cpp：热词功能实现源文件

| | |──DTW.cpp：DTW算法实现源文件

| | |──Tools：字符串处理文件目录

| |──Punctuation：标点功能模块

| | |──Punctuation.h：标点功能接口头文件

| | |──libPunctuation.so：标点功能接口库

| | |──libcrflearn.so：标点功能依赖crf库

| |──ITN：ITN功能模块

| | |──translate-api.h：ITN功能头文件

| | |──libthraxrewrite.so：ITN功能实现库

| | |──thrax-1.2.3：ITN功能依赖的thrax库

| | |──openfst-1.6.3：ITN功能依赖的openfst库

## 4 函数调用流程

### 4.1“单线程单解码器”函数调用流程



说明：

（1）图中虚线部分为可选操作；

### 5.2“多线程多解码器”函数调用流程

这里多线程指的是“多个发送数据线程”，下图分别展示在线模式下多线程多解码器函数调用流程。

Fig.b 离线模式“多线程多解码器”函数调用流程

如如图Fig.b所示，外层通过TBNR\_SendData()接口函数完成送入语音任务，内层对进来的语音任务依次进行高通滤波，特征提取，端点检测，然后将分段的特征压入任务池，在处理多任务时，所有会话任务公用一个任务池，解码线程依次从任务池中依次获取特征任务，完成解码并将解码后的结果各自的结果池中，并通过结果回调返回结果。

说明1：在DNNVAD检测过程中，检测到尾点时，将起点到尾点之间的特征数据压入任务池中等待解码；也就是说离线识别环境是在检测到尾点之后开始进行解码。

说明2：结果池的目的：由于在解码特征任务时，不能保证按照特征任务顺序依次解码完成，可能存在同一语音任务中的后一段比前一段解码先完成，所以需解码结果进行排序，防止返回结果时结果出现乱序。

说明3：外层的会话数和内层的解码线程数没有关系，并且会话ID(SessionID)与解码ID(DecodeID)也没有对应关系，解码线程从公共任务池中依次去取特征任务，完成解码后，根据特征任务中会话ID信息，压入与之ID对应的结果池中并返回与之ID对应的回调函数。

### 5.3 解码器函数调用流程



## 如何正确运行

按照章节2的步骤编译生成动态库libTBNR\_API.so后，将该so到测试环境bin\lib目录下，即可进行识别。

## 附录

### A 接口函数调用流程

