**软件学院2032-2024学年度第二学期**

**《软件工程理论与实践课程设计》**

**实验报告**

|  |  |
| --- | --- |
| **班级：** | 软件工程2204班 |
| **项目名称：** | 语音识别ASR模型移动部署 |
| **项目组组长：** | 彭晓风 |
| **项目组成员：** | 学号：U202217245  姓名：彭晓风  学号：U202217247  姓名：舒雯浩 |
| **指导老师：** | 刘小峰 |

**目录**

[1. 项目概述 3](#_Toc168591146)

[2. 需求描述 3](#_Toc168591147)

[2.1 功能性需求 3](#_Toc168591148)

[2.2 非功能性需求 3](#_Toc168591149)

[3. 系统设计[深度学习部署类题目可以不写] 3](#_Toc168591150)

[3.1 架构设计 3](#_Toc168591151)

[3.2 界面原型设计 3](#_Toc168591152)

[3.3 详细设计 3](#_Toc168591153)

[4. 系统实现 3](#_Toc168591154)

[5. 系统界面展示 4](#_Toc168591155)

[6. 总结 4](#_Toc168591156)

## 项目概述

* 1. 项目基本介绍

UI界面



应用图标



项目介绍

该项目旨在把whisper模型部署到移动端，在移动端实现语音识别功能。

功能特性

移动端的Transform能实现英文的语音识别，并提供显示识别消耗时间的功能。

Transform的使用非常简单，只需点击录音按钮，Transform将自动完成语音录入、语音识别和结果输出。

* 1. github仓库地址

<https://github.com/fengqi121/Transform>

* 1. 人员基本分工
     1. 队长：彭晓风

作为项目队长，彭晓风主要负责安卓端应用前端和部分后端的编写及模型的转换。

1. ****Github 仓库管理****：他将负责整个 Github 仓库的管理工作，包括代码整理、归档以及版本的管理。他还将负责编写详实的 README 文档，以便于其他团队成员以及可能接触项目的人对项目有明确的了解。
2. ****用户界面设计：他****将负责用户界面的设计和开发，他会注重用户体验，致力于创建出既易于使用又美观的用户界面。
3. **文档撰写**：为了保证项目的可维护性和持续性，他将负责编写清晰且详细的开发文档，以便于后续的开发和维护工作。
4. **共同进行模型转换：**他将和其他成员一同进行模型转换，把whisper模型转换为安卓端可用的模型。
5. **模型调用：**他将编写录音逻辑的实现和模型调用，以实现模型响应和前后端的快速响应。
6. TDD 测试：他将采用测试驱动开发（TDD）的方法，为代码编写详尽的单元测试，以确保代码的质量和稳定性。
7. **编写文档：**为了保证项目的可维护性和持续性，他将负责编写清晰且详细的开发文档，以便于后续的开发和维护工作。
   * 1. 队员：舒雯浩

舒雯浩主要负责安卓端应用后端和模型前的处理以及参与模型转换。

1. **共同撰写需求：**一起参与需求讨论和构思，确保对项目的理解一致，从而能够产生详尽而有用的项目需求文档。在撰写需求过程中，努力保证每个要求都准确、清晰，易于理解且可行。
2. **模型前的处理：**对读取的音频进行处理，将其转换为模型能够识别的张量，以便模型能够被有效使用。
3. **共同进行模型转换：**他将与队长一起进行模型的转换。
4. **编写文档：**他将帮助队长进行开发文档、设计文档的撰写。

## 需求描述

### 功能性需求

2.1.1、用户需求

移动部署

将whisper模型部署到移动端，在移动端本地实现语音识别的功能。

2.1.2、系统需求

语音识别功能

1. 初始假设

用户希望模型部署到移动端后能实现语音识别点的功能。

1. 正常状态

用户点击录音按钮，系统开始录音。

用户再次点击录音按钮，录音结束，调用模型识别结果，识别完毕后处理结果输出到屏幕。

1. 有哪些会出错

录音进程被非正常关闭但录音还未关闭。

4、其他活动

系统应该保证用户隐私，不记录用户的语音输入内容。

5、完成的系统状态

用户可以通过录音输入方式进行语音识别。

系统可以对语音进行识别并将其转换为文本格式。

### 非功能性需求

2.2.1 性能需求：

语音识别的处理时间：对于录音，系统应当在录音结束后3秒内给出结果。

2.2.2 安全性需求：

用户的录音不能被应用非法保存。

2.2.3 可用性需求：

模型调用应当只在调用时打开，完毕后停止。

2.2.4 可维护性需求：

系统应具备良好的模块化和文档化，以便进行维护和升级。

系统应能够容易地添加新的功能。

2.2.5 可扩展性需求：

系统应设计成可扩展的架构，以便在未来可以添加更多的功能。

2.2.6 用户体验需求：

系统的用户界面应简洁易用，用户能够快速理解如何使用各项功能。

## 系统实现

开发语言：Kotlin+java

开发框架；Android8.0

开发环境：Adroid studio

3.1.模型量化转换

本项目量化方法是使用olive工具包量化转换，不同于普遍的转换方法。

参考链接：https://github.com/microsoft/Olive/tree/main/examples/whisper

**使用 ORT 工具链进行whisper模型优化**

3.1.1.量化管道

执行优化管道：

CPU，FP32：PyTorch 模型 -> Onnx 模型 -> Transformers 优化的 Onnx 模型 -> 插入光束搜索操作 -> 插入前/后处理操作

CPU，INT8：PyTorch 模型 -> Onnx 模型 -> Transformers 优化的 Onnx 模型 ->动态量化的 Onnx 模型 -> 插入光束搜索操作 -> 插入前/后处理操作

CPU，INT8：PyTorch 模型 -> Onnx 模型 -> Transformers 优化的 Onnx 模型 -> Intel® 神经压缩器动态量化 Onnx 模型 -> 插入光束搜索操作 ->插入前/后处理操作

GPU，FP32：PyTorch 模型 -> Onnx 模型 -> 转换器优化的 Onnx 模型 -> 插入光束搜索操作 -> 插入前/后处理操作

GPU，FP16：PyTorch 模型 -> Onnx 模型 -> 转换器优化的 Onnx 模型 ->混合精密模型 -> 插入光束搜索操作 -> 插入预/后处理操作

GPU，INT8：PyTorch 模型 -> Onnx 模型 -> Transformers 优化的 Onnx 模型 ->动态量化的 Onnx 模型 -> 插入光束搜索操作 -> 插入前/后处理操作

输出最终模型和延迟结果。

3.1.2.准备工作

先决条件

克隆存储库并安装 Olive。

打开终端运行：

**git clone https://github.com/microsoft/Olive.git**

**cd Olive**

**python -m pip install .**

从开源库里下载whisper量化示例文件夹，在文件夹里打开终端：

**python -m pip install -r requirements.txt**

准备工作流配置 json

**python prepare\_whisper\_configs.py --model\_name openai/whisper-tiny.en**

运行配置以优化模型

**首先，根据通行证安装所需的软件包。**

**olive run --config whisper\_cpu\_int8.json --setup**

**3.1.3.模型优化**

**olive run --config whisper\_cpu\_int8.json 2> $null**

测试优化模型的转录

**python test\_transcription.py --config whisper\_cpu\_int8.json**

3.2.src/main/java/whisper.transform

3.2.1.Mainactivity.kt

这个文件是 Android 应用的主活动。它包含了应用的主要逻辑，包括录音按钮的点击事件处理，录音的开始和结束，以及音频数据的处理。当录音按钮被点击时，它会启动一个新的线程来处理音频数据，然后将这些数据传递给语音识别器。

package whisper.transform

//导入所需要的包

import android.widget.ImageButton

import android.Manifest

import android.content.pm.PackageManager

import androidx.appcompat.app.AppCompatActivity

import android.os.Bundle

import android.util.Log

import android.widget.TextView

import android.widget.Toast

import androidx.core.app.ActivityCompat

import java.util.concurrent.Executors

import java.util.concurrent.atomic.AtomicBoolean

class MainActivity : AppCompatActivity() {

//应用程序的主要组件

private val recordAudioButton: ImageButton by lazy { findViewById(R.id.audio\_button) }

private val resultText: TextView by lazy { findViewById(R.id.result\_text) }

private val statusText: TextView by lazy { findViewById(R.id.status\_text) }

private val speechRecognizer: SpeechRecognizer by lazy {

resources.openRawResource(R.raw.whisper\_cpu\_int8\_model).use {

val modelBytes = it.readBytes()

SpeechRecognizer(modelBytes)

}

}

private var recording = false//录音标志

private val stopRecordingFlag = AtomicBoolean(false)//停止录音标志

//定义线程执行器

private val workerThreadExecutor = Executors.newSingleThreadExecutor()

//输出结果

private fun setSuccessfulResult(result: SpeechRecognizer.Result) {

runOnUiThread {

statusText.text = "语音识别花费(${result.inferenceTimeInMs} ms)。"

resultText.text = result.text.ifEmpty { "<无语音需识别。>" }

}

}

//处理异常

private fun setError(exception: Exception) {

Log.e(TAG, "Error: ${exception.localizedMessage}", exception)

runOnUiThread {

statusText.text = "Error"

resultText.text = exception.localizedMessage

}

}

//检查是否有录音权限

private fun hasRecordAudioPermission(): Boolean =

ActivityCompat.checkSelfPermission(

this,

Manifest.permission.RECORD\_AUDIO

) == PackageManager.PERMISSION\_GRANTED

//请求权限

override fun onRequestPermissionsResult(

requestCode: Int,

permissions: Array<out String>,

grantResults: IntArray

) {

super.onRequestPermissionsResult(requestCode, permissions, grantResults)

if (requestCode == RECORD\_AUDIO\_PERMISSION\_REQUEST\_CODE) {

if (!hasRecordAudioPermission()) {

Toast.makeText(

this,

"请打开录音权限",

Toast.LENGTH\_SHORT

).show()

}

}

}

//初始化

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

setContentView(R.layout.activity\_main)

//录音的点击事件

recordAudioButton.setOnClickListener {

//检查是否有录音权限

if (!hasRecordAudioPermission()) {

requestPermissions(

arrayOf(Manifest.permission.RECORD\_AUDIO),

RECORD\_AUDIO\_PERMISSION\_REQUEST\_CODE

)

return@setOnClickListener

}

recording=!recording//录音标志

stopRecordingFlag.set(!recording)

workerThreadExecutor.submit {

try {

if (recording) {

runOnUiThread {

recordAudioButton.setImageResource(R.drawable.micphone\_open)

}//录音状态

} else {

runOnUiThread {

recordAudioButton.setImageResource(R.drawable.micphone\_closed)

}//默认状态

}

runOnUiThread {

recordAudioButton.isEnabled = true

}//录音按钮可用

if(recording) {

val audioTensor = AudioTensorSource.fromRecording(stopRecordingFlag)

val result = audioTensor.use { speechRecognizer.run(audioTensor) }

setSuccessfulResult(result)

}

} catch (e: Exception) {

setError(e)

}

}

}

}

//后台挂起时

override fun onPause() {

super.onPause()

stopRecordingFlag.set(true)

}

//应用退出

override fun onDestroy() {

super.onDestroy()

workerThreadExecutor.shutdown()

speechRecognizer.close()

}

companion object {

const val TAG = "ORTSpeechRecognizer"//标识i当前模块

private const val RECORD\_AUDIO\_PERMISSION\_REQUEST\_CODE = 1

}

}

3.2.2.AudioTensorSource.kt

这个文件定义了一个名为 AudioTensorSource 的类，该类用于模型前的处理，该类负责从麦克风录音并将音频数据转换为 OnnxTensor 对象。这个类使用了 AudioRecord 类来从麦克风获取音频数据，并使用 OnnxTensor.createTensor 方法将这些数据转换为张量。

package whisper.transform

//导入工具包

import ai.onnxruntime.OnnxTensor

import ai.onnxruntime.OrtEnvironment

import android.annotation.SuppressLint

import android.media.AudioFormat

import android.media.AudioRecord

import android.media.MediaRecorder

import android.util.Log

import java.nio.FloatBuffer

import java.util.concurrent.atomic.AtomicBoolean

//声音处理类

class AudioTensorSource {

companion object {

private const val bytesPerFloat = 4//浮点数占4个字节

private const val sampleRate = 16000//采样率

private const val maxAudioLengthInSeconds = 30//最大录音时间

@SuppressLint("MissingPermission")

fun fromRecording(stopRecordingFlag: AtomicBoolean): OnnxTensor {

val recordingChunkLengthInSeconds = 1

//获取最小缓冲区大小

val minBufferSize = maxOf(

AudioRecord.getMinBufferSize(

sampleRate,

AudioFormat.CHANNEL\_IN\_MONO,

AudioFormat.ENCODING\_PCM\_FLOAT

),

2 \* recordingChunkLengthInSeconds \* sampleRate \* bytesPerFloat

)

//创建一个音频记录器

val audioRecord = AudioRecord.Builder()

.setAudioSource(MediaRecorder.AudioSource.MIC)

.setAudioFormat(

AudioFormat.Builder()

.setSampleRate(sampleRate)

.setEncoding(AudioFormat.ENCODING\_PCM\_FLOAT)

.setChannelMask(AudioFormat.CHANNEL\_IN\_MONO)

.build()

)

.setBufferSizeInBytes(minBufferSize)

.build()

try {

//创建一个浮点数数组，用于存储音频数据

val floatAudioData = FloatArray(maxAudioLengthInSeconds \* sampleRate) { 0.0f }

//浮点数数据偏移量

var floatAudioDataOffset = 0

//开始录音

audioRecord.startRecording()

while (!stopRecordingFlag.get() && floatAudioDataOffset < floatAudioData.size) {

//计算需要读取的浮点数

val numFloatsToRead = minOf(

recordingChunkLengthInSeconds \* sampleRate,

floatAudioData.size - floatAudioDataOffset

)

//读取音频数据

val readResult = audioRecord.read(

floatAudioData, floatAudioDataOffset, numFloatsToRead,

AudioRecord.READ\_BLOCKING

)

//打印日志

Log.d(MainActivity.TAG, "AudioRecord.read(float[], ...) returned $readResult")

//如果读取结果大于等于0

if (readResult >= 0) {

floatAudioDataOffset += readResult

} else {

throw RuntimeException("AudioRecord.read() returned error code $readResult")

}

}

//停止录音

audioRecord.stop()

//获取环境

val env = OrtEnvironment.getEnvironment()

val floatAudioDataBuffer = FloatBuffer.wrap(floatAudioData)

//返回一个音频张量

return OnnxTensor.createTensor(

env, floatAudioDataBuffer,

tensorShape(1, floatAudioData.size.toLong())

)

} finally {

//释放音频记录器

if (audioRecord.recordingState == AudioRecord.RECORDSTATE\_RECORDING) {

audioRecord.stop()

}

audioRecord.release()

}

}

}

}

3.2.3.SpeechRecognizer.kt

这个文件定义了一个名为SpeechRecognizer的类，该类负责运行语音识别模型，对模型输出进行处理，返回结果。

package whisper.transform

//导入必要包

import ai.onnxruntime.OnnxTensor

import ai.onnxruntime.OrtEnvironment

import ai.onnxruntime.OrtSession

import ai.onnxruntime.extensions.OrtxPackage

import android.os.SystemClock

//声音识别类

class SpeechRecognizer(modelBytes: ByteArray) : AutoCloseable {

private val session: OrtSession//会话

private val baseInputs: Map<String, OnnxTensor>//基础输入

//初始化

init {

val env = OrtEnvironment.getEnvironment()//获取环境

val sessionOptions = OrtSession.SessionOptions()//会话选项

sessionOptions.registerCustomOpLibrary(OrtxPackage.getLibraryPath())//注册自定义操作库

session = env.createSession(modelBytes, sessionOptions)//创建会话

//基础输入

baseInputs = mapOf(

"min\_length" to createIntTensor(env, intArrayOf(1), tensorShape(1)),

"max\_length" to createIntTensor(env, intArrayOf(200), tensorShape(1)),

"num\_beams" to createIntTensor(env, intArrayOf(1), tensorShape(1)),

"num\_return\_sequences" to createIntTensor(env, intArrayOf(1), tensorShape(1)),

"length\_penalty" to createFloatTensor(env, floatArrayOf(1.0f), tensorShape(1)),

"repetition\_penalty" to createFloatTensor(env, floatArrayOf(1.0f), tensorShape(1)),

)

}

//结果类

data class Result(val text: String, val inferenceTimeInMs: Long)

//运行类

fun run(audioTensor: OnnxTensor): Result {

//输入

val inputs = mutableMapOf<String, OnnxTensor>()

baseInputs.toMap(inputs)

inputs["audio\_pcm"] = audioTensor

val startTimeInMs = SystemClock.elapsedRealtime()//开始时间

val outputs = session.run(inputs)//运行

val elapsedTimeInMs = SystemClock.elapsedRealtime() - startTimeInMs//花费时间

val recognizedText = outputs.use {

@Suppress("UNCHECKED\_CAST")

(outputs[0].value as Array<Array<String>>)[0][0]

}//识别文本

return Result(recognizedText, elapsedTimeInMs)//返回结果

}

//关闭onnxtensor对象

override fun close() {

baseInputs.values.forEach {

it.close()

}

session.close()

}

}

3.2.4、TensorUtils.kt

这个文件包含了一些工具函数，用于创建 OnnxTensor 对象。这些函数接受一个 OrtEnvironment 对象、一个数据数组以及一个形状数组，然后使用 OnnxTensor.createTensor 方法创建一个新的 OnnxTensor 对象。

package whisper.transform

import ai.onnxruntime.OnnxTensor

import ai.onnxruntime.OrtEnvironment

import java.nio.FloatBuffer

import java.nio.IntBuffer

internal fun createIntTensor(env: OrtEnvironment, data: IntArray, shape: LongArray): OnnxTensor {

return OnnxTensor.createTensor(env, IntBuffer.wrap(data), shape)

}

internal fun createFloatTensor(

env: OrtEnvironment,

data: FloatArray,

shape: LongArray

): OnnxTensor {

return OnnxTensor.createTensor(env, FloatBuffer.wrap(data), shape)

}

internal fun tensorShape(vararg dims: Long) = longArrayOf(\*dims)

3.3.1.main/AndroidManifest.xml

这个文件包含该应用的主要信息

<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:tools="http://schemas.android.com/tools">

<uses-permission android:name="android.permission.RECORD\_AUDIO" />

<application

android:allowBackup="true"

android:dataExtractionRules="@xml/data\_extraction\_rules"

android:fullBackupContent="@xml/backup\_rules"

android:icon="@mipmap/ic\_launcher"

android:label="@string/app\_name"

android:supportsRtl="true"

android:theme="@style/Theme.SpeechRecognition"

tools:targetApi="31">

<activity

android:name="whisper.transform.MainActivity"

android:exported="true">

<intent-filter>

<action android:name="android.intent.action.MAIN" />

<category android:name="android.intent.category.LAUNCHER" />

</intent-filter>

</activity>

</application>

</manifest>

3.4.src/main/res

该文件夹为资源文件夹

3.4.1.layout/activity\_main.xml

定义了应用的主界面。

<?xml version="1.0" encoding="utf-8"?>

<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:app="http://schemas.android.com/apk/res-auto"

xmlns:tools="http://schemas.android.com/tools"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

tools:context="whisper.transform.MainActivity"

android:background="@drawable/background">

//录音按钮

<ImageButton

android:id="@+id/audio\_button"

android:layout\_width="0dp"

android:layout\_height="0dp"

android:layout\_margin="40dp"

android:layout\_marginEnd="172dp"

android:layout\_marginBottom="44dp"

android:scaleType="centerCrop"

app:layout\_constraintBottom\_toBottomOf="parent"

app:layout\_constraintEnd\_toEndOf="parent"

app:layout\_constraintHeight\_percent="0.15"

app:layout\_constraintHorizontal\_bias="0.543"

app:layout\_constraintStart\_toStartOf="parent"

app:layout\_constraintWidth\_percent="0.3"

app:srcCompat="@drawable/micphone\_closed" />

//结果显示文本

<TextView

android:id="@+id/result\_text"

android:layout\_width="0dp"

android:layout\_height="0dp"

android:text="TextView"

app:layout\_constraintBottom\_toTopOf="@id/status\_text"

app:layout\_constraintEnd\_toEndOf="parent"

app:layout\_constraintHeight\_percent="0.5"

app:layout\_constraintHorizontal\_bias="0.0"

app:layout\_constraintStart\_toStartOf="parent"

app:layout\_constraintTop\_toTopOf="parent"

app:layout\_constraintVertical\_bias="0.0"

app:layout\_constraintWidth\_percent="1"/>

//运行时间显示文本

<TextView

android:id="@+id/status\_text"

android:layout\_width="0dp"

android:layout\_height="0dp"

android:layout\_marginBottom="32dp"

android:text="TextView"

app:layout\_constraintBottom\_toTopOf="@id/audio\_button"

app:layout\_constraintEnd\_toEndOf="parent"

app:layout\_constraintHeight\_percent="0.2"

app:layout\_constraintHorizontal\_bias="0.0"

app:layout\_constraintStart\_toStartOf="parent"

app:layout\_constraintWidth\_percent="1" />

</androidx.constraintlayout.widget.ConstraintLayout>

3.5.src/java/whisper/transform/SpeechRecognitionInstrumentedTest.kt

package whisper.transform

import androidx.test.core.app.ActivityScenario

import androidx.test.espresso.Espresso.onView

import androidx.test.espresso.action.ViewActions.click

import androidx.test.espresso.matcher.ViewMatchers.withId

import androidx.test.ext.junit.runners.AndroidJUnit4

import androidx.test.platform.app.InstrumentationRegistry

import junit.framework.TestCase.assertEquals

import org.junit.Assert

import org.junit.Test

import org.junit.runner.RunWith

import java.lang.reflect.Field

/\*\*

\* Instrumented test, which will execute on an Android device.

\*

\* See [testing documentation](http://d.android.com/tools/testing).

\*/

@RunWith(AndroidJUnit4::class)

class SpeechRecognitionInstrumentedTest {

@Test

fun useAppContext() {

// Context of the app under test.

val appContext = InstrumentationRegistry.getInstrumentation().targetContext

assertEquals("whisper.transform", appContext.packageName)

}

fun testRecordingValueAfterButtonClick() {

// 启动 MainActivity

val scenario = ActivityScenario.launch(MainActivity::class.java)

// 获取 MainActivity 的 recording 字段的初始值

var recordingInitialValue = false

scenario.onActivity { activity ->

val field: Field = MainActivity::class.java.getDeclaredField("recording")

field.isAccessible = true

recordingInitialValue = field.get(activity) as Boolean

}

// 找到录音按钮并点击一次

onView(withId(R.id.audio\_button)).perform(click())

// 获取 MainActivity 的 recording 字段的值

var recordingValueAfterClick = false

scenario.onActivity { activity ->

val field: Field = MainActivity::class.java.getDeclaredField("recording")

field.isAccessible = true

recordingValueAfterClick = field.get(activity) as Boolean

}

// 验证 recording 字段的值是否已经改变

Assert.assertNotEquals(recordingInitialValue, recordingValueAfterClick)

}

}

## 系统界面展示

activity\_main(主界面）默认状态



录音开始后



点击录音按钮开始录音，再次点击后进行识别，输出结果。

## 总结

在这个课程设计项目中，我们的团队实现了语音识别whisper模型部署到移动端，在移动端本地实现了语音识别的功能。在这个过程中，我们深入了解了pytorch模型的结构、模型量化、模型输入和输出处理。

在代码实现方面，我们遵循了良好的编程习惯。我们的代码结构清晰，每个函数、每个模块都有其明确的职责；我们的命名规范，变量名、函数名都能准确地反映其功能；我们的注释详细，每一段重要的代码都有相应的注释，方便后续的维护和修改。这些都是我们在软件工程课程中学到的重要知识，也是我们在实际开发过程中得到应用的地方。

总的来说，这个课程设计项目是一次非常宝贵的实践经验。它不仅提升了我们的编程技能，也锻炼了我们的问题解决能力。同时，看到自己的作品能够真正地帮助到用户，也是一种非常满足的感觉。我们深感，软件开发不仅仅是编写代码，更是解决问题，满足用户需求的过程。我们将带着这次的经验和教训，继续在软件工程的道路上探索和前进。