**Text but no speech**  
  
package com.example.altsclient  
  
import android.Manifest  
import android.content.pm.PackageManager  
import android.media.MediaPlayer  
import android.media.MediaRecorder  
import android.os.Bundle  
import android.util.Log  
import android.widget.Button  
import android.widget.EditText  
import android.widget.Toast  
import androidx.appcompat.app.AppCompatActivity  
import androidx.core.app.ActivityCompat  
import androidx.core.content.ContextCompat  
import okhttp3.\*  
import java.io.File  
import java.io.IOException  
import java.util.concurrent.TimeUnit  
import okhttp3.MediaType.Companion.toMediaType  
import okhttp3.RequestBody.Companion.asRequestBody  
  
class MainActivity : AppCompatActivity() {  
 private lateinit var recordButton: Button  
 private lateinit var serverUrl: EditText  
 private var recorder: MediaRecorder? = null  
 private lateinit var fileName: String  
 private lateinit var responseFileName: String  
 private val client = OkHttpClient.Builder()  
 .connectTimeout(60, TimeUnit.*SECONDS*)  
 .readTimeout(60, TimeUnit.*SECONDS*)  
 .writeTimeout(60, TimeUnit.*SECONDS*)  
 .build()  
 private val RECORD\_REQUEST\_CODE = 101  
 private var isRecording = false  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.*activity\_main*)  
  
 recordButton = findViewById(R.id.*recordButton*)  
 serverUrl = findViewById(R.id.*serverUrl*)  
  
 fileName = "${*externalCacheDir*?.*absolutePath* ?: *filesDir*.*absolutePath*}/audio\_record.m4a"  
 responseFileName = "${*externalCacheDir*?.*absolutePath* ?: *filesDir*.*absolutePath*}/response.wav"  
  
 requestPermissions()  
  
 recordButton.*text* = "Start Recording"  
 recordButton.setOnClickListener **{** if (!isRecording) {  
 if (hasPermission()) {  
 startRecording()  
 recordButton.*text* = "Stop Recording"  
 isRecording = true  
 } else {  
 Toast.makeText(this, "Microphone permission not granted", Toast.*LENGTH\_SHORT*).show()  
 }  
 } else {  
 stopRecording()  
 recordButton.*text* = "Start Recording"  
 isRecording = false  
 }  
 **}** }  
  
 private fun requestPermissions() {  
 if (ContextCompat.checkSelfPermission(this, Manifest.permission.*RECORD\_AUDIO*) != PackageManager.*PERMISSION\_GRANTED* ||  
 ContextCompat.checkSelfPermission(this, Manifest.permission.*INTERNET*) != PackageManager.*PERMISSION\_GRANTED*) {  
 ActivityCompat.requestPermissions(this, *arrayOf*(Manifest.permission.*RECORD\_AUDIO*, Manifest.permission.*INTERNET*), RECORD\_REQUEST\_CODE)  
 }  
 }  
  
 override fun onRequestPermissionsResult(requestCode: Int, permissions: Array<out String>, grantResults: IntArray) {  
 super.onRequestPermissionsResult(requestCode, permissions, grantResults)  
 if (requestCode == RECORD\_REQUEST\_CODE && grantResults.*isNotEmpty*() && grantResults[0] == PackageManager.*PERMISSION\_GRANTED*) {  
 // Permission granted  
 } else {  
 Toast.makeText(this, "Permissions required for recording and network", Toast.*LENGTH\_SHORT*).show()  
 }  
 }  
  
 private fun hasPermission(): Boolean {  
 return ContextCompat.checkSelfPermission(this, Manifest.permission.*RECORD\_AUDIO*) == PackageManager.*PERMISSION\_GRANTED* }  
  
 private fun startRecording() {  
 try {  
 recorder = MediaRecorder().*apply* **{** setAudioSource(MediaRecorder.AudioSource.*MIC*)  
 setOutputFormat(MediaRecorder.OutputFormat.*MPEG\_4*)  
 setOutputFile(fileName)  
 setAudioEncoder(MediaRecorder.AudioEncoder.*AAC*)  
 setAudioSamplingRate(16000) // Match Whisper's preferred rate  
 setAudioChannels(1) // Mono for compatibility  
 prepare()  
 start()  
 **}** Toast.makeText(this, "Recording started", Toast.*LENGTH\_SHORT*).show()  
 } catch (e: Exception) {  
 Log.e("RecordingError", "Failed to start recording: ${e.message}", e)  
 Toast.makeText(this, "Recording failed: ${e.message}", Toast.*LENGTH\_SHORT*).show()  
 }  
 }  
  
 private fun stopRecording() {  
 try {  
 recorder?.*apply* **{** stop()  
 reset() // Reset recorder state  
 **}** recorder?.release()  
 recorder = null  
 Toast.makeText(this, "Recording stopped", Toast.*LENGTH\_SHORT*).show()  
 val audioFile = File(fileName)  
 Log.d("AudioFile", "Recorded file size: ${audioFile.length()} bytes")  
 if (audioFile.exists() && audioFile.length() > 1024) {  
 // Verify file is valid M4A  
 try {  
 val player = MediaPlayer().*apply* **{** setDataSource(fileName)  
 prepare()  
 release()  
 **}** playRecordedAudio() // Test playback before sending  
 sendAudioToServer()  
 } catch (e: Exception) {  
 Log.e("AudioValidation", "Invalid audio file: ${e.message}", e)  
 Toast.makeText(this, "Invalid audio file: ${e.message}", Toast.*LENGTH\_LONG*).show()  
 }  
 } else {  
 Toast.makeText(this, "Recorded file is empty or too small", Toast.*LENGTH\_LONG*).show()  
 }  
 } catch (e: Exception) {  
 Log.e("StopError", "Error stopping recorder: ${e.message}", e)  
 Toast.makeText(this, "Failed to stop recording: ${e.message}", Toast.*LENGTH\_SHORT*).show()  
 } finally {  
 recorder?.release()  
 recorder = null  
 }  
 }  
  
 private fun playRecordedAudio() {  
 try {  
 val player = MediaPlayer().*apply* **{** setDataSource(fileName)  
 prepare()  
 start()  
 Log.d("MediaPlayer", "Playing recorded audio, duration: ${getDuration()} ms")  
 **}** player.setOnCompletionListener **{** Log.d("MediaPlayer", "Recorded audio playback completed")  
 **it**.release()  
 **}** } catch (e: Exception) {  
 Log.e("PlayError", "Error playing recorded audio: ${e.message}", e)  
 Toast.makeText(this, "Failed to play recorded audio: ${e.message}", Toast.*LENGTH\_LONG*).show()  
 }  
 }  
  
 private fun sendAudioToServer() {  
 val file = File(fileName)  
 if (!file.exists() || file.length() == 0L) {  
 Toast.makeText(this, "Recorded file not found or empty", Toast.*LENGTH\_SHORT*).show()  
 return  
 }  
 Log.d("AudioFile", "Sending audio file, size: ${file.length()} bytes")  
  
 val requestBody = MultipartBody.Builder()  
 .setType(MultipartBody.FORM)  
 .addFormDataPart("file", "audio\_record.m4a", file.*asRequestBody*("audio/mp4".*toMediaType*()))  
 .build()  
  
 val url = "http://172.22.64.1:8000/process\_audio"  
 Log.d("Request", "Sending to $url")  
  
 val request = Request.Builder()  
 .url(url)  
 .post(requestBody)  
 .build()  
  
 client.newCall(request).enqueue(object : Callback {  
 override fun onFailure(call: Call, e: IOException) {  
 Log.e("NetworkError", "Failed to send: ${e.message}", e)  
 runOnUiThread **{** Toast.makeText(this@MainActivity, "Network error: ${e.message}", Toast.*LENGTH\_LONG*).show()  
 **}** }  
  
 override fun onResponse(call: Call, response: Response) {  
 Log.d("Response", "Response code: ${response.code}")  
 if (!response.isSuccessful) {  
 Log.e("ServerError", "Error: ${response.code} - ${response.message}")  
 runOnUiThread **{** Toast.makeText(this@MainActivity, "Server error: ${response.code}", Toast.*LENGTH\_LONG*).show()  
 **}** return  
 }  
  
 try {  
 val responseFile = File(responseFileName)  
 response.body?.byteStream()?.*use* **{** input **->** responseFile.*outputStream*().*use* **{** output **->** input.*copyTo*(output)  
 **}  
 }** Log.d("FileSize", "Response file size: ${responseFile.length()} bytes")  
 if (responseFile.length() == 0L) {  
 Log.e("ResponseError", "Received empty response file")  
 runOnUiThread **{** Toast.makeText(this@MainActivity, "Empty response from server", Toast.*LENGTH\_LONG*).show()  
 **}** return  
 }  
 runOnUiThread **{** playResponse() **}** } catch (e: Exception) {  
 Log.e("ResponseError", "Error handling response: ${e.message}", e)  
 runOnUiThread **{** Toast.makeText(this@MainActivity, "Failed to process response: ${e.message}", Toast.*LENGTH\_LONG*).show()  
 **}** }  
 }  
 })  
 }  
  
 private fun playResponse() {  
 try {  
 val responseFile = File(responseFileName)  
 if (!responseFile.exists() || responseFile.length() == 0L) {  
 Log.e("PlayError", "Response file not found or empty")  
 Toast.makeText(this, "Response file not found or empty", Toast.*LENGTH\_LONG*).show()  
 return  
 }  
 val player = MediaPlayer().*apply* **{** setDataSource(responseFileName)  
 prepare()  
 start()  
 Log.d("MediaPlayer", "Playing response audio, duration: ${getDuration()} ms")  
 **}** player.setOnCompletionListener **{** Log.d("MediaPlayer", "Response audio playback completed")  
 **it**.release()  
 **}** } catch (e: Exception) {  
 Log.e("PlayError", "Error playing response: ${e.message}", e)  
 Toast.makeText(this, "Failed to play response: ${e.message}", Toast.*LENGTH\_LONG*).show()  
 }  
 }  
  
 override fun onDestroy() {  
 super.onDestroy()  
 recorder?.release()  
 recorder = null  
 }  
}

--------------------------------------------------------------------------------------

from fastapi import FastAPI, UploadFile, File, Response

from fastapi.middleware.cors import CORSMiddleware

from TTS.api import TTS

import whisper

from litellm import completion

import tempfile

import os

import yaml

from dotenv import load\_dotenv

from pydub import AudioSegment

# Load environment variables and configuration

load\_dotenv()

with open('config.yaml', 'r') as file:

config = yaml.safe\_load(file)

# Initialize AI models

stt = whisper.load\_model(config["whisper"]["model"])

tts = TTS(model\_name=config["tts"]["model"], progress\_bar=False)

llm\_config = config["llm"]

# Set up FastAPI app with CORS

app = FastAPI()

app.add\_middleware(

CORSMiddleware,

allow\_origins=["\*"],

allow\_credentials=True,

allow\_methods=["\*"],

allow\_headers=["\*"],

)

@app.post("/process\_audio")

async def process\_audio(file: UploadFile = File(...)):

# Save uploaded audio to a temporary file

with tempfile.NamedTemporaryFile(suffix=".m4a", delete=False) as temp\_audio:

audio\_data = await file.read()

temp\_audio.write(audio\_data)

audio\_path = temp\_audio.name

print(f"Received audio file: {audio\_path}, size: {len(audio\_data)} bytes")

# Convert audio to WAV using pydub

try:

audio = AudioSegment.from\_file(audio\_path, format="m4a")

print(f"Audio file details: channels={audio.channels}, sample\_rate={audio.frame\_rate}, duration={len(audio)/1000:.2f}s")

wav\_path = audio\_path.replace(".m4a", ".wav")

audio = audio.set\_channels(1).set\_frame\_rate(16000) # Ensure mono and 16kHz

audio.export(wav\_path, format="wav")

print(f"Saved audio as WAV: {wav\_path}, size: {os.path.getsize(wav\_path)} bytes")

except Exception as e:

print(f"Audio preprocessing error: {str(e)}")

os.remove(audio\_path)

with tempfile.NamedTemporaryFile(suffix=".wav", delete=False) as temp\_synth:

tts.tts\_to\_file(

text="Audio processing failed. Please try again.",

speaker=config["tts"]["speakerId"] if tts.is\_multi\_speaker else None,

file\_path=temp\_synth.name

)

with open(temp\_synth.name, "rb") as f:

audio\_data = f.read()

os.remove(temp\_synth.name)

return Response(content=audio\_data, media\_type="audio/wav")

# Transcribe audio using Whisper

try:

transcription = stt.transcribe(wav\_path, language="en")

text = transcription["text"]

language = transcription["language"]

print(f"Transcribed text: '{text}', Language: {language}")

except Exception as e:

print(f"Transcription error: {str(e)}")

text = "Transcription failed. Please try again."

language = "en"

# Clean up audio files

os.remove(audio\_path)

if os.path.exists(wav\_path):

os.remove(wav\_path)

# Prepare LLM request

messages = [{"role": "system", "content": llm\_config["system"]}]

if not text.strip():

text = "No input detected. Please ask a question."

messages.append({"role": "user", "content": text})

print(f"LLM input messages: {messages}")

# Get LLM response

try:

response = completion(

model=llm\_config["model"],

messages=messages,

api\_base=llm\_config["url"],

stream=False

)

llm\_response = response["choices"][0]["message"]["content"]

except Exception as e:

llm\_response = f"Error processing LLM: {str(e)}"

print(f"LLM response: '{llm\_response}'")

# Synthesize response to audio using TTS

with tempfile.NamedTemporaryFile(suffix=".wav", delete=False) as temp\_synth:

try:

tts.tts\_to\_file(

text=llm\_response,

speaker=config["tts"]["speakerId"] if tts.is\_multi\_speaker else None,

language=language if tts.is\_multi\_lingual and language in tts.languages else None,

file\_path=temp\_synth.name

)

except Exception as e:

print(f"TTS error: {str(e)}")

tts.tts\_to\_file(

text="Sorry, I couldn't generate a response. Please try again.",

speaker=config["tts"]["speakerId"] if tts.is\_multi\_speaker else None,

file\_path=temp\_synth.name

)

synth\_path = temp\_synth.name

# Read and return synthesized audio

with open(synth\_path, "rb") as f:

audio\_data = f.read()

print(f"Synthesized audio size: {len(audio\_data)} bytes")

os.remove(synth\_path)

return Response(content=audio\_data, media\_type="audio/wav")

Working Model:  
package com.example.altsclient  
  
import android.Manifest  
import android.content.pm.PackageManager  
import android.media.MediaPlayer  
import android.media.MediaRecorder  
import android.os.Bundle  
import android.util.Log  
import android.widget.Button  
import android.widget.EditText  
import android.widget.Toast  
import androidx.appcompat.app.AppCompatActivity  
import androidx.core.app.ActivityCompat  
import androidx.core.content.ContextCompat  
import okhttp3.\*  
import java.io.File  
import java.io.IOException  
import java.util.concurrent.TimeUnit  
import okhttp3.MediaType.Companion.toMediaType  
import okhttp3.RequestBody.Companion.asRequestBody  
  
class MainActivity : AppCompatActivity() {  
 private lateinit var recordButton: Button  
 private lateinit var serverUrl: EditText  
 private var recorder: MediaRecorder? = null  
 private lateinit var fileName: String  
 private lateinit var responseFileName: String  
 private val client = OkHttpClient.Builder()  
 .connectTimeout(30, TimeUnit.*SECONDS*)  
 .readTimeout(30, TimeUnit.*SECONDS*)  
 .writeTimeout(30, TimeUnit.*SECONDS*)  
 .build()  
 private val RECORD\_REQUEST\_CODE = 101  
 private var isRecording = false  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.*activity\_main*)  
  
 recordButton = findViewById(R.id.*recordButton*)  
 serverUrl = findViewById(R.id.*serverUrl*)  
  
 fileName = "${*externalCacheDir*?.*absolutePath* ?: *filesDir*.*absolutePath*}/audio\_record.m4a"  
 responseFileName = "${*externalCacheDir*?.*absolutePath* ?: *filesDir*.*absolutePath*}/response.wav"  
  
 requestPermissions()  
  
 recordButton.*text* = "Start Recording"  
 recordButton.setOnClickListener **{** if (!isRecording) {  
 if (hasPermission()) {  
 startRecording()  
 recordButton.*text* = "Stop Recording"  
 isRecording = true  
 } else {  
 Toast.makeText(this, "Microphone permission not granted", Toast.*LENGTH\_SHORT*).show()  
 }  
 } else {  
 stopRecording()  
 recordButton.*text* = "Start Recording"  
 isRecording = false  
 }  
 **}** }  
  
 private fun requestPermissions() {  
 if (ContextCompat.checkSelfPermission(this, Manifest.permission.*RECORD\_AUDIO*) != PackageManager.*PERMISSION\_GRANTED* ||  
 ContextCompat.checkSelfPermission(this, Manifest.permission.*INTERNET*) != PackageManager.*PERMISSION\_GRANTED*) {  
 ActivityCompat.requestPermissions(this, *arrayOf*(Manifest.permission.*RECORD\_AUDIO*, Manifest.permission.*INTERNET*), RECORD\_REQUEST\_CODE)  
 }  
 }  
  
 override fun onRequestPermissionsResult(requestCode: Int, permissions: Array<out String>, grantResults: IntArray) {  
 super.onRequestPermissionsResult(requestCode, permissions, grantResults)  
 if (requestCode == RECORD\_REQUEST\_CODE && grantResults.*isNotEmpty*() && grantResults[0] == PackageManager.*PERMISSION\_GRANTED*) {  
 // Permission granted  
 } else {  
 Toast.makeText(this, "Permissions required for recording and network", Toast.*LENGTH\_SHORT*).show()  
 }  
 }  
  
 private fun hasPermission(): Boolean {  
 return ContextCompat.checkSelfPermission(this, Manifest.permission.*RECORD\_AUDIO*) == PackageManager.*PERMISSION\_GRANTED* }  
  
 private fun startRecording() {  
 try {  
 recorder = MediaRecorder().*apply* **{** setAudioSource(MediaRecorder.AudioSource.*MIC*)  
 setOutputFormat(MediaRecorder.OutputFormat.*MPEG\_4*)  
 setOutputFile(fileName)  
 setAudioEncoder(MediaRecorder.AudioEncoder.*AAC*)  
 setAudioSamplingRate(16000) // Match Whisper's preferred rate  
 setAudioChannels(1) // Mono for compatibility  
 prepare()  
 start()  
 **}** Toast.makeText(this, "Recording started", Toast.*LENGTH\_SHORT*).show()  
 } catch (e: Exception) {  
 Log.e("RecordingError", "Failed to start recording: ${e.message}", e)  
 Toast.makeText(this, "Recording failed: ${e.message}", Toast.*LENGTH\_SHORT*).show()  
 }  
 }  
  
 private fun stopRecording() {  
 try {  
 recorder?.*apply* **{** stop()  
 reset() // Reset recorder state  
 **}** recorder?.release()  
 recorder = null  
 Toast.makeText(this, "Recording stopped", Toast.*LENGTH\_SHORT*).show()  
 val audioFile = File(fileName)  
 Log.d("AudioFile", "Recorded file size: ${audioFile.length()} bytes")  
 if (audioFile.exists() && audioFile.length() > 1024) {  
 // Verify file is valid M4A  
 try {  
 val player = MediaPlayer().*apply* **{** setDataSource(fileName)  
 prepare()  
 release()  
 **}** playRecordedAudio() // Test playback before sending  
 sendAudioToServer()  
 } catch (e: Exception) {  
 Log.e("AudioValidation", "Invalid audio file: ${e.message}", e)  
 Toast.makeText(this, "Invalid audio file: ${e.message}", Toast.*LENGTH\_LONG*).show()  
 }  
 } else {  
 Toast.makeText(this, "Recorded file is empty or too small", Toast.*LENGTH\_LONG*).show()  
 }  
 } catch (e: Exception) {  
 Log.e("StopError", "Error stopping recorder: ${e.message}", e)  
 Toast.makeText(this, "Failed to stop recording: ${e.message}", Toast.*LENGTH\_SHORT*).show()  
 } finally {  
 recorder?.release()  
 recorder = null  
 }  
 }  
  
 private fun playRecordedAudio() {  
 try {  
 val player = MediaPlayer().*apply* **{** setDataSource(fileName)  
 prepare()  
 start()  
 Log.d("MediaPlayer", "Playing recorded audio, duration: ${getDuration()} ms")  
 **}** player.setOnCompletionListener **{** Log.d("MediaPlayer", "Recorded audio playback completed")  
 **it**.release()  
 **}** } catch (e: Exception) {  
 Log.e("PlayError", "Error playing recorded audio: ${e.message}", e)  
 Toast.makeText(this, "Failed to play recorded audio: ${e.message}", Toast.*LENGTH\_LONG*).show()  
 }  
 }  
  
 private fun sendAudioToServer() {  
 val file = File(fileName)  
 if (!file.exists() || file.length() == 0L) {  
 Toast.makeText(this, "Recorded file not found or empty", Toast.*LENGTH\_SHORT*).show()  
 return  
 }  
 Log.d("AudioFile", "Sending audio file, size: ${file.length()} bytes")  
  
 val requestBody = MultipartBody.Builder()  
 .setType(MultipartBody.FORM)  
 .addFormDataPart("file", "audio\_record.m4a", file.*asRequestBody*("audio/mp4".*toMediaType*()))  
 .build()  
  
 val url = "http://172.22.64.1:8000/process\_audio"  
 Log.d("Request", "Sending to $url")  
  
 val request = Request.Builder()  
 .url(url)  
 .post(requestBody)  
 .build()  
  
 client.newCall(request).enqueue(object : Callback {  
 override fun onFailure(call: Call, e: IOException) {  
 Log.e("NetworkError", "Failed to send: ${e.message}", e)  
 runOnUiThread **{** Toast.makeText(this@MainActivity, "Network error: ${e.message}", Toast.*LENGTH\_LONG*).show()  
 **}** }  
  
 override fun onResponse(call: Call, response: Response) {  
 Log.d("Response", "Response code: ${response.code}")  
 if (!response.isSuccessful) {  
 Log.e("ServerError", "Error: ${response.code} - ${response.message}")  
 runOnUiThread **{** Toast.makeText(this@MainActivity, "Server error: ${response.code}", Toast.*LENGTH\_LONG*).show()  
 **}** return  
 }  
  
 try {  
 val responseFile = File(responseFileName)  
 response.body?.byteStream()?.*use* **{** input **->** responseFile.*outputStream*().*use* **{** output **->** input.*copyTo*(output)  
 **}  
 }** Log.d("FileSize", "Response file size: ${responseFile.length()} bytes")  
 if (responseFile.length() == 0L) {  
 Log.e("ResponseError", "Received empty response file")  
 runOnUiThread **{** Toast.makeText(this@MainActivity, "Empty response from server", Toast.*LENGTH\_LONG*).show()  
 **}** return  
 }  
 runOnUiThread **{** playResponse() **}** } catch (e: Exception) {  
 Log.e("ResponseError", "Error handling response: ${e.message}", e)  
 runOnUiThread **{** Toast.makeText(this@MainActivity, "Failed to process response: ${e.message}", Toast.*LENGTH\_LONG*).show()  
 **}** }  
 }  
 })  
 }  
  
 private fun playResponse() {  
 try {  
 val responseFile = File(responseFileName)  
 if (!responseFile.exists() || responseFile.length() == 0L) {  
 Log.e("PlayError", "Response file not found or empty: $responseFileName")  
 Toast.makeText(this, "Response file not found or empty", Toast.*LENGTH\_LONG*).show()  
 return  
 }  
 Log.d("PlayResponse", "Attempting to play response file: $responseFileName, size: ${responseFile.length()} bytes")  
  
 // Validate WAV file format  
 val player = MediaPlayer().*apply* **{** setDataSource(responseFileName)  
 try {  
 prepare()  
 } catch (e: Exception) {  
 Log.e("PlayError", "MediaPlayer prepare failed: ${e.message}", e)  
 Toast.makeText(this@MainActivity, "Invalid audio format: ${e.message}", Toast.*LENGTH\_LONG*).show()  
 return  
 }  
 start()  
 Log.d("MediaPlayer", "Playing response audio, duration: ${getDuration()} ms")  
 **}** player.setOnCompletionListener **{** Log.d("MediaPlayer", "Response audio playback completed")  
 **it**.release()  
 **}** player.setOnErrorListener **{** mp, what, extra **->** Log.e("MediaPlayer", "Error during playback: what=$what, extra=$extra")  
 Toast.makeText(this@MainActivity, "Playback error: what=$what, extra=$extra", Toast.*LENGTH\_LONG*).show()  
 mp.release()  
 true  
 **}** } catch (e: Exception) {  
 Log.e("PlayError", "Error playing response: ${e.message}", e)  
 Toast.makeText(this, "Failed to play response: ${e.message}", Toast.*LENGTH\_LONG*).show()  
 }  
 }  
  
 override fun onDestroy() {  
 super.onDestroy()  
 recorder?.release()  
 recorder = null  
 }  
}

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:orientation="vertical"  
 android:padding="16dp"  
 android:gravity="center">  
  
 <EditText  
 android:id="@+id/serverUrl"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:hint="Enter server IP:Port (e.g., 172.22.64.1:8000)"  
 android:inputType="textUri"  
 android:padding="8dp" />  
  
 <Button  
 android:id="@+id/recordButton"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:text="Start Recording"  
 android:layout\_marginTop="24dp"  
 android:padding="16dp" />  
</LinearLayout>

prmaram@Prameela-Maram:~/alts\_env$ cat alts\_server.py

from fastapi import FastAPI, UploadFile, File, Response

from fastapi.middleware.cors import CORSMiddleware

from TTS.api import TTS

import whisper

from litellm import completion

import tempfile

import os

import yaml

from dotenv import load\_dotenv

from pydub import AudioSegment

# Load environment variables and configuration

load\_dotenv()

with open('config.yaml', 'r') as file:

config = yaml.safe\_load(file)

# Initialize AI models

stt = whisper.load\_model(config["whisper"]["model"])

tts = TTS(model\_name=config["tts"]["model"], progress\_bar=False)

llm\_config = config["llm"]

# Set up FastAPI app with CORS

app = FastAPI()

app.add\_middleware(

CORSMiddleware,

allow\_origins=["\*"],

allow\_credentials=True,

allow\_methods=["\*"],

allow\_headers=["\*"],

)

@app.post("/process\_audio")

async def process\_audio(file: UploadFile = File(...)):

# Save uploaded audio to a temporary file

with tempfile.NamedTemporaryFile(suffix=".m4a", delete=False) as temp\_audio:

audio\_data = await file.read()

temp\_audio.write(audio\_data)

audio\_path = temp\_audio.name

print(f"Received audio file: {audio\_path}, size: {len(audio\_data)} bytes")

# Convert audio to WAV using pydub

try:

audio = AudioSegment.from\_file(audio\_path, format="m4a")

print(f"Audio file details: channels={audio.channels}, sample\_rate={audio.frame\_rate}, duration={len(audio)/1000:.2f}s")

wav\_path = audio\_path.replace(".m4a", ".wav")

audio = audio.set\_channels(1).set\_frame\_rate(16000).set\_sample\_width(2) # Mono, 16kHz, 16-bit

audio.export(wav\_path, format="wav")

print(f"Saved audio as WAV: {wav\_path}, size: {os.path.getsize(wav\_path)} bytes")

except Exception as e:

print(f"Audio preprocessing error: {str(e)}")

os.remove(audio\_path)

with tempfile.NamedTemporaryFile(suffix=".wav", delete=False) as temp\_synth:

tts.tts\_to\_file(

text="Audio processing failed. Please try again.",

file\_path=temp\_synth.name

)

# Ensure fallback audio is also in correct format

audio = AudioSegment.from\_file(temp\_synth.name)

audio = audio.set\_channels(1).set\_frame\_rate(16000).set\_sample\_width(2)

audio.export(temp\_synth.name, format="wav")

with open(temp\_synth.name, "rb") as f:

audio\_data = f.read()

os.remove(temp\_synth.name)

return Response(content=audio\_data, media\_type="audio/wav")

# Transcribe audio using Whisper

try:

transcription = stt.transcribe(wav\_path, language="en")

text = transcription["text"]

language = transcription["language"]

print(f"Transcribed text: '{text}', Language: {language}")

except Exception as e:

print(f"Transcription error: {str(e)}")

text = "Transcription failed. Please try again."

language = "en"

# Clean up audio files

os.remove(audio\_path)

if os.path.exists(wav\_path):

os.remove(wav\_path)

# Prepare LLM request

messages = [{"role": "system", "content": llm\_config["system"]}]

if not text.strip():

text = "No input detected. Please ask a question."

messages.append({"role": "user", "content": text})

print(f"LLM input messages: {messages}")

# Get LLM response

try:

response = completion(

model=llm\_config["model"],

messages=messages,

api\_base=llm\_config["url"],

stream=False

)

llm\_response = response["choices"][0]["message"]["content"]

except Exception as e:

llm\_response = f"Error processing LLM: {str(e)}"

print(f"LLM response: '{llm\_response}'")

# Synthesize response to audio using TTS

with tempfile.NamedTemporaryFile(suffix=".wav", delete=False) as temp\_synth:

try:

tts.tts\_to\_file(

text=llm\_response,

speaker=config["tts"]["speakerId"] if tts.is\_multi\_speaker else None,

language=language if tts.is\_multi\_lingual and language in tts.languages else None,

file\_path=temp\_synth.name

)

# Post-process TTS output to ensure correct format

audio = AudioSegment.from\_file(temp\_synth.name)

audio = audio.set\_channels(1).set\_frame\_rate(16000).set\_sample\_width(2) # Mono, 16kHz, 16-bit

audio.export(temp\_synth.name, format="wav")

print(f"Synthesized audio details: channels={audio.channels}, sample\_rate={audio.frame\_rate}, sample\_width={audio.sample\_width\*8}-bit, duration={len(audio)/1000:.2f}s")

except Exception as e:

print(f"TTS error: {str(e)}")

tts.tts\_to\_file(

text="Sorry, I couldn't generate a response. Please try again.",

file\_path=temp\_synth.name

)

audio = AudioSegment.from\_file(temp\_synth.name)

audio = audio.set\_channels(1).set\_frame\_rate(16000).set\_sample\_width(2)

audio.export(temp\_synth.name, format="wav")

synth\_path = temp\_synth.name

# Read and return synthesized audio

with open(synth\_path, "rb") as f:

audio\_data = f.read()

print(f"Synthesized audio size: {len(audio\_data)} bytes")

os.remove(synth\_path)

return Response(content=audio\_data, media\_type="audio/wav")

**COMPLETE WORKING:**

from fastapi import FastAPI, UploadFile, File, Response

from fastapi.middleware.cors import CORSMiddleware

from TTS.api import TTS

import whisper

from litellm import completion

import tempfile

import os

import yaml

from dotenv import load\_dotenv

from pydub import AudioSegment

import base64

from fastapi.responses import JSONResponse

# Load environment variables and configuration

load\_dotenv()

with open('config.yaml', 'r') as file:

config = yaml.safe\_load(file)

# Initialize AI models

stt = whisper.load\_model(config["whisper"]["model"])

tts = TTS(model\_name=config["tts"]["model"], progress\_bar=False)

llm\_config = config["llm"]

# Set up FastAPI app with CORS

app = FastAPI()

app.add\_middleware(

CORSMiddleware,

allow\_origins=["\*"],

allow\_credentials=True,

allow\_methods=["\*"],

allow\_headers=["\*"],

)

@app.post("/process\_audio")

async def process\_audio(file: UploadFile = File(...)):

# Save uploaded audio to a temporary file

with tempfile.NamedTemporaryFile(suffix=".m4a", delete=False) as temp\_audio:

audio\_data = await file.read()

temp\_audio.write(audio\_data)

audio\_path = temp\_audio.name

print(f"Received audio file: {audio\_path}, size: {len(audio\_data)} bytes")

# Convert audio to WAV using pydub

try:

audio = AudioSegment.from\_file(audio\_path, format="m4a")

print(f"Audio file details: channels={audio.channels}, sample\_rate={audio.frame\_rate}, duration={len(audio)/1000:.2f}s")

wav\_path = audio\_path.replace(".m4a", ".wav")

audio = audio.set\_channels(1).set\_frame\_rate(16000) # Ensure mono and 16kHz

audio.export(wav\_path, format="wav")

print(f"Saved audio as WAV: {wav\_path}, size: {os.path.getsize(wav\_path)} bytes")

except Exception as e:

print(f"Audio preprocessing error: {str(e)}")

os.remove(audio\_path)

with tempfile.NamedTemporaryFile(suffix=".wav", delete=False) as temp\_synth:

tts.tts\_to\_file(

text="Audio processing failed. Please try again.",

speaker=config["tts"]["speakerId"] if tts.is\_multi\_speaker else None,

file\_path=temp\_synth.name

)

with open(temp\_synth.name, "rb") as f:

audio\_data = f.read()

os.remove(temp\_synth.name)

return JSONResponse(content={

"text": "Audio processing failed. Please try again.",

"audio": base64.b64encode(audio\_data).decode('utf-8')

})

# Transcribe audio using Whisper

try:

transcription = stt.transcribe(wav\_path, language="en")

text = transcription["text"]

language = transcription["language"]

print(f"Transcribed text: '{text}', Language: {language}")

except Exception as e:

print(f"Transcription error: {str(e)}")

text = "Transcription failed. Please try again."

language = "en"

# Clean up audio files

os.remove(audio\_path)

if os.path.exists(wav\_path):

os.remove(wav\_path)

# Prepare LLM request

messages = [{"role": "system", "content": llm\_config["system"]}]

if not text.strip():

text = "No input detected. Please ask a question."

messages.append({"role": "user", "content": text})

print(f"LLM input messages: {messages}")

# Get LLM response

try:

response = completion(

model=llm\_config["model"],

messages=messages,

api\_base=llm\_config["url"],

stream=False

)

llm\_response = response["choices"][0]["message"]["content"]

except Exception as e:

llm\_response = f"Error processing LLM: {str(e)}"

print(f"LLM response: '{llm\_response}'")

# Synthesize response to audio using TTS

with tempfile.NamedTemporaryFile(suffix=".wav", delete=False) as temp\_synth:

try:

tts.tts\_to\_file(

text=llm\_response,

speaker=config["tts"]["speakerId"] if tts.is\_multi\_speaker else None,

language=language if tts.is\_multi\_lingual and language in tts.languages else None,

file\_path=temp\_synth.name

)

except Exception as e:

print(f"TTS error: {str(e)}")

tts.tts\_to\_file(

text="Sorry, I couldn't generate a response. Please try again.",

speaker=config["tts"]["speakerId"] if tts.is\_multi\_speaker else None,

file\_path=temp\_synth.name

)

synth\_path = temp\_synth.name

# Read and encode synthesized audio as base64

with open(synth\_path, "rb") as f:

audio\_data = f.read()

audio\_base64 = base64.b64encode(audio\_data).decode('utf-8')

print(f"Synthesized audio size: {len(audio\_data)} bytes")

os.remove(synth\_path)

# Return JSON with text and audio

return JSONResponse(content={

"text": llm\_response,

"audio": audio\_base64

}) **-------------------------------------------------------------------------------------------------------------**

package com.example.altsclient  
  
import android.Manifest  
import android.content.pm.PackageManager  
import android.media.MediaPlayer  
import android.media.MediaRecorder  
import android.os.Bundle  
import android.util.Base64  
import android.util.Log  
import android.widget.Button  
import android.widget.EditText  
import android.widget.TextView  
import android.widget.Toast  
import androidx.appcompat.app.AppCompatActivity  
import androidx.core.app.ActivityCompat  
import androidx.core.content.ContextCompat  
import androidx.core.content.res.ResourcesCompat  
import okhttp3.\*  
import java.io.File  
import java.io.IOException  
import java.util.concurrent.TimeUnit  
import okhttp3.MediaType.Companion.toMediaType  
import okhttp3.RequestBody.Companion.asRequestBody  
import org.json.JSONObject  
  
class MainActivity : AppCompatActivity() {  
 private lateinit var recordButton: Button  
 private lateinit var serverUrl: EditText  
 private lateinit var responseTextView: TextView  
 private var recorder: MediaRecorder? = null  
 private lateinit var fileName: String  
 private val client = OkHttpClient.Builder()  
 .connectTimeout(120, TimeUnit.*SECONDS*)  
 .readTimeout(120, TimeUnit.*SECONDS*)  
 .writeTimeout(120, TimeUnit.*SECONDS*)  
 .build()  
 private val RECORD\_REQUEST\_CODE = 101  
 private var isRecording = false  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.*activity\_main*)  
  
 recordButton = findViewById(R.id.*recordButton*)  
 serverUrl = findViewById(R.id.*serverUrl*)  
 responseTextView = findViewById(R.id.*responseTextView*)  
  
 fileName = "${*externalCacheDir*?.*absolutePath* ?: *filesDir*.*absolutePath*}/audio\_record.m4a"  
  
 requestPermissions()  
  
 // Set initial button color (green for "Start Recording")  
 recordButton.*text* = "Start Recording"  
 recordButton.setBackgroundColor(ContextCompat.getColor(this, android.R.color.*holo\_green\_light*))  
  
 recordButton.setOnClickListener {  
 if (!isRecording) {  
 if (hasPermission()) {  
 startRecording()  
 recordButton.*text* = "Stop Recording"  
 recordButton.setBackgroundColor(ContextCompat.getColor(this, android.R.color.*holo\_red\_light*))  
 isRecording = true  
 } else {  
 Toast.makeText(this, "Microphone permission not granted", Toast.*LENGTH\_SHORT*).show()  
 }  
 } else {  
 stopRecording()  
 recordButton.*text* = "Start Recording"  
 recordButton.setBackgroundColor(ContextCompat.getColor(this, android.R.color.*holo\_green\_light*))  
 isRecording = false  
 }  
 }  
 }  
  
 private fun requestPermissions() {  
 if (ContextCompat.checkSelfPermission(this, Manifest.permission.*RECORD\_AUDIO*) != PackageManager.*PERMISSION\_GRANTED* ||  
 ContextCompat.checkSelfPermission(this, Manifest.permission.*INTERNET*) != PackageManager.*PERMISSION\_GRANTED*) {  
 ActivityCompat.requestPermissions(this, *arrayOf*(Manifest.permission.*RECORD\_AUDIO*, Manifest.permission.*INTERNET*), RECORD\_REQUEST\_CODE)  
 }  
 }  
  
 override fun onRequestPermissionsResult(requestCode: Int, permissions: Array<out String>, grantResults: IntArray) {  
 super.onRequestPermissionsResult(requestCode, permissions, grantResults)  
 if (requestCode == RECORD\_REQUEST\_CODE && grantResults.*isNotEmpty*() && grantResults[0] == PackageManager.*PERMISSION\_GRANTED*) {  
 // Permission granted  
 } else {  
 Toast.makeText(this, "Permissions required for recording and network", Toast.*LENGTH\_SHORT*).show()  
 }  
 }  
  
 private fun hasPermission(): Boolean {  
 return ContextCompat.checkSelfPermission(this, Manifest.permission.*RECORD\_AUDIO*) == PackageManager.*PERMISSION\_GRANTED* }  
  
 private fun startRecording() {  
 try {  
 recorder = MediaRecorder().*apply* {  
 setAudioSource(MediaRecorder.AudioSource.*MIC*)  
 setOutputFormat(MediaRecorder.OutputFormat.*MPEG\_4*)  
 setOutputFile(fileName)  
 setAudioEncoder(MediaRecorder.AudioEncoder.*AAC*)  
 setAudioSamplingRate(16000)  
 setAudioChannels(1)  
 prepare()  
 start()  
 }  
 Toast.makeText(this, "Recording started", Toast.*LENGTH\_SHORT*).show()  
 } catch (e: Exception) {  
 Log.e("RecordingError", "Failed to start recording: ${e.message}", e)  
 Toast.makeText(this, "Recording failed: ${e.message}", Toast.*LENGTH\_SHORT*).show()  
 }  
 }  
  
 private fun stopRecording() {  
 try {  
 recorder?.*apply* {  
 stop()  
 reset()  
 }  
 recorder?.release()  
 recorder = null  
 Toast.makeText(this, "Recording stopped", Toast.*LENGTH\_SHORT*).show()  
 val audioFile = File(fileName)  
 Log.d("AudioFile", "Recorded file size: ${audioFile.length()} bytes")  
 if (audioFile.exists() && audioFile.length() > 1024) {  
 try {  
 val player = MediaPlayer().*apply* {  
 setDataSource(fileName)  
 prepare()  
 release()  
 }  
 sendAudioToServer()  
 } catch (e: Exception) {  
 Log.e("AudioValidation", "Invalid audio file: ${e.message}", e)  
 Toast.makeText(this, "Invalid audio file: ${e.message}", Toast.*LENGTH\_LONG*).show()  
 }  
 } else {  
 Toast.makeText(this, "Recorded file is empty or too small", Toast.*LENGTH\_LONG*).show()  
 }  
 } catch (e: Exception) {  
 Log.e("StopError", "Error stopping recorder: ${e.message}", e)  
 Toast.makeText(this, "Failed to stop recording: ${e.message}", Toast.*LENGTH\_SHORT*).show()  
 } finally {  
 recorder?.release()  
 recorder = null  
 }  
 }  
  
 private fun sendAudioToServer() {  
 val file = File(fileName)  
 if (!file.exists() || file.length() == 0L) {  
 Toast.makeText(this, "Recorded file not found or empty", Toast.*LENGTH\_SHORT*).show()  
 return  
 }  
 Log.d("AudioFile", "Sending audio file, size: ${file.length()} bytes")  
  
 val requestBody = MultipartBody.Builder()  
 .setType(MultipartBody.FORM)  
 .addFormDataPart("file", "audio\_record.m4a", file.*asRequestBody*("audio/mp4".*toMediaType*()))  
 .build()  
  
 val url = if (serverUrl.*text*.toString().*startsWith*("http")) {  
 "${serverUrl.*text*}/process\_audio"  
 } else {  
 "http://${serverUrl.*text*}/process\_audio"  
 }  
 Log.d("Request", "Sending to $url")  
  
 val request = Request.Builder()  
 .url(url)  
 .post(requestBody)  
 .build()  
  
 client.newCall(request).enqueue(object : Callback {  
 override fun onFailure(call: Call, e: IOException) {  
 Log.e("NetworkError", "Failed to send: ${e.message}", e)  
 runOnUiThread {  
 Toast.makeText(this@MainActivity, "Network error: ${e.message}", Toast.*LENGTH\_LONG*).show()  
 }  
 }  
  
 override fun onResponse(call: Call, response: Response) {  
 Log.d("Response", "Response code: ${response.code}")  
 if (!response.isSuccessful) {  
 Log.e("ServerError", "Error: ${response.code} - ${response.message}")  
 runOnUiThread {  
 Toast.makeText(this@MainActivity, "Server error: ${response.code}", Toast.*LENGTH\_LONG*).show()  
 }  
 return  
 }  
  
 try {  
 val responseBody = response.body?.string()  
 if (responseBody.*isNullOrEmpty*()) {  
 Log.e("ResponseError", "Received empty response")  
 runOnUiThread {  
 Toast.makeText(this@MainActivity, "Empty response from server", Toast.*LENGTH\_LONG*).show()  
 }  
 return  
 }  
  
 val json = JSONObject(responseBody)  
 val llmText = json.getString("text")  
 val audioBase64 = json.getString("audio")  
  
 // Update UI with LLM response  
 runOnUiThread {  
 responseTextView.*text* = llmText  
 }  
  
 // Decode and play audio  
 val audioData = Base64.decode(audioBase64, Base64.*DEFAULT*)  
 playResponse(audioData)  
 } catch (e: Exception) {  
 Log.e("ResponseError", "Error handling response: ${e.message}", e)  
 runOnUiThread {  
 Toast.makeText(this@MainActivity, "Failed to process response: ${e.message}", Toast.*LENGTH\_LONG*).show()  
 }  
 }  
 }  
 })  
 }  
  
 private fun playResponse(audioData: ByteArray) {  
 try {  
 // Create a temporary file to store the audio  
 val tempFile = File.createTempFile("response", ".wav", *cacheDir*)  
 tempFile.*writeBytes*(audioData)  
  
 val player = MediaPlayer().*apply* {  
 setDataSource(tempFile.*absolutePath*)  
 prepare()  
 start()  
 Log.d("MediaPlayer", "Playing response audio, duration: ${getDuration()} ms")  
 }  
 player.setOnCompletionListener {  
 Log.d("MediaPlayer", "Response audio playback completed")  
 it.release()  
 tempFile.delete() // Clean up temporary file  
 }  
 } catch (e: Exception) {  
 Log.e("PlayError", "Error playing response: ${e.message}", e)  
 runOnUiThread {  
 Toast.makeText(this, "Failed to play response: ${e.message}", Toast.*LENGTH\_LONG*).show()  
 }  
 }  
 }  
  
 override fun onDestroy() {  
 super.onDestroy()  
 recorder?.release()  
 recorder = null  
 }  
}

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:orientation="vertical"  
 android:padding="16dp"  
 android:gravity="center">  
  
 <EditText  
 android:id="@+id/serverUrl"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:hint="Enter server IP:Port (e.g., 172.22.64.1:8000)"  
 android:inputType="textUri"  
 android:padding="8dp" />  
  
 <Button  
 android:id="@+id/recordButton"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:text="Start Recording"  
 android:layout\_marginTop="24dp"  
 android:padding="16dp"  
 android:background="@drawable/button\_background" />  
  
 <TextView  
 android:id="@+id/responseTextView"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:layout\_marginTop="24dp"  
 android:text="LLM Response will appear here"  
 android:textSize="16sp"  
 android:padding="8dp"  
 android:gravity="center" />  
</LinearLayout>