Speech-to-Text and Sentiment Analysis Using Google Multimodal LLM API

Introduction

This project implements a web-based application that processes recorded or uploaded audio files by using the Google Multimodal LLM API**.** The application uses a single API call to perform both transcription and sentiment analysis on the audio files. The results, including the transcriptions and sentiment analysis, are stored in text files on Google Cloud Storage and displayed to the user through the interface.

Goals and Objectives

- **User-Friendly Interface**: Provide an intuitive interface for recording, uploading, and displaying results.
- **Unified Processing:** Use the Google Multimodal LLM API to handle transcription and sentiment analysis in a single call.
- **Efficient Storage:** Save the Audios and API's response (transcription and sentiment analysis) as text files in Google Cloud Storage.

Features Implemented

- **Audio Recording and Upload:** Allow users to record audio in their browser or upload pre-recorded audio files.
- **Single API Call for Processing:** Use the Google Multimodal LLM API to perform transcription and sentiment analysis on audio files in a single call.
- **Dynamic File Storage and Display:** Store the API responses as text files in Google Cloud Storage, dynamically display stored files and results in the user interface.

Architecture:

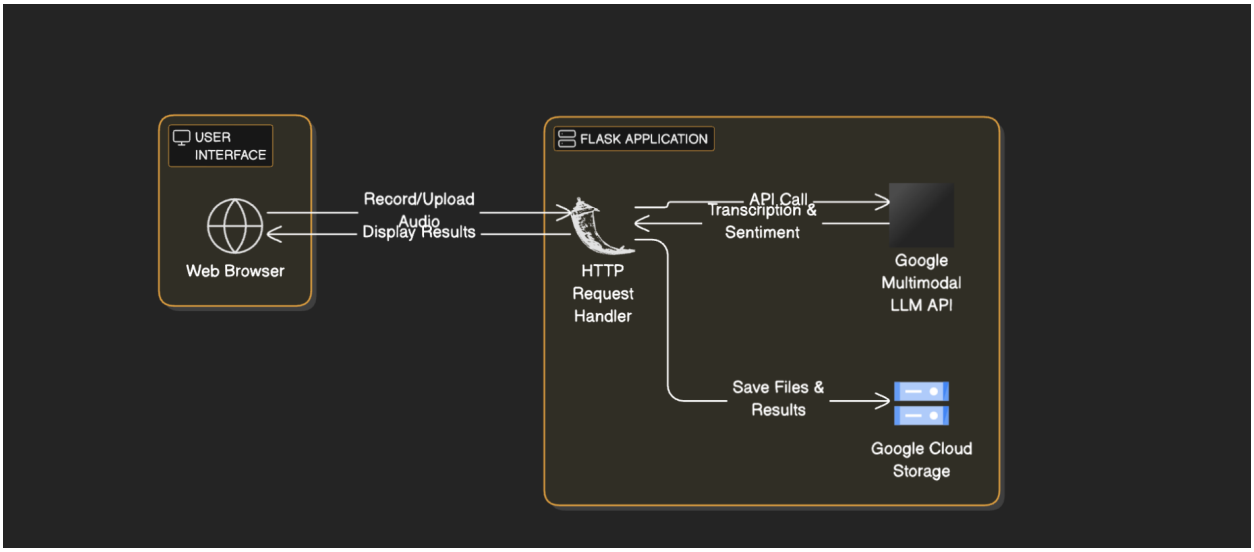


Fig: Application Architecture Diagram

Components

- **Flask Application**: Handles recording, file uploads, API requests, and result displays.
- **Google Multimodal LLM API**: Processes audio files to provide both transcription and sentiment analysis in one API call.
- **Google Cloud Storage**: Stores audio files and the corresponding API response text files.

Implementation

1. **Audio Recording and Upload**:

    - Users can record audio directly via their browser or upload existing audio files through the web interface.

2. **Unified Processing via Google Multimodal LLM API**:

    - A single API call is made to the Google Multimodal LLM API, configured with the **Gemini model**, to process audio files.

- The API performs both transcription and sentiment analysis in one step, returning a JSON response with the transcription and sentiment analysis results.

3. **API Request Workflow**:
   - The audio file (either recorded or uploaded) is sent to the API using an inline data configuration.
   - The API processes the audio and returns the transcription and sentiment analysis score in the following JSON format:

```
{
  "transcription": "Transcribed text here",
  "sentiment_analysis": "Positive/Neutral/Negative"
}
```

4. **File Management**:

   - **Audio Storage**:
     The uploaded or recorded audio file is saved in the static/audio/ or static/recordings/ directories of the Google Cloud Storage bucket.
   - **Transcription and Sentiment Analysis Results**:
     - The JSON response from the API is processed to extract the transcription and sentiment analysis score.
     - The combined output is saved as a text file in the static/atranscripts/ or static/rtranscripts/ directories of the bucket.

5. **Dynamic File Display**:

   - All stored files (audio and text) are dynamically listed on the user interface using Flask endpoints.
   - Separate lists are maintained for recordings, uploaded audio files, and their respective transcriptions.

6. **Error Handling**:

   - Comprehensive error handling ensures that any issues during API calls, file uploads, or storage operations are logged and returned as user-friendly messages.

7. **Key Features**:

   - The API key is securely read from a config.json file.
   - The sentiment analysis score is computed based on the transcription's emotional tone (Positive, Neutral, Negative).

- Both transcription and sentiment results are provided in real-time for user convenience.

Deployment

Deploy to Google Cloud Run:

Run the below command to deploy the project

gcloud run deploy

ak_akshayajay@cloudshell:~/prj1 (convaiproject1)$ gcloud run deploy

Command response:

Verify Deployment:

Visit https://prj1-1091022759046.us-east1.run.app/ to interact with the deployed application.

Pros and Cons

Advantages:

- Simplifies the architecture by consolidating tasks into a single API call.
- Provides users with both transcription and sentiment results in one step.
- Efficiently stores and organizes results for easy access.

Challenges:

- Dependence on the Multimodal LLM API may lead to cost considerations for large-scale usage.
- Latency for processing large audio files might impact user experience.

Problems Encountered and Solutions

Problem 1: Latency during API processing.

- Solution: Implemented efficient file handling and asynchronous processing to minimize delays.

Problem 2: Displaying API results in real-time.

- Solution: Streamlined result parsing and dynamically updated the user interface with API responses.

Application Instructions

Recording and Storing Files:

- Click "Start Recording" to record audio through the browser.
- Once recording is stopped, the audio file is automatically uploaded to Google Cloud Storage.
- The application retrieves the transcription and sentiment analysis from the API and displays the results on the webpage.

Uploading Audio for Transcription and Sentiment Analysis:

- Use the "Upload" option to upload an audio file.

- The file is sent to the Google Multimodal LLM API, which processes it in a single call.
- The transcription and sentiment analysis results are stored in Google Cloud Storage and displayed on the webpage.

Viewing Stored Files:

- Previously recorded or uploaded files, along with their corresponding transcriptions and sentiment analysis, are listed on the webpage.
- Users can view or download the files directly.

Lessons Learned

Integration with Google Multimodal LLM API:

- Simplified the application's workflow by leveraging a single API call for transcription and sentiment analysis.
- Learned to configure and use the **Gemini model** for multimodal tasks.

Efficient File Management:

- Gained experience in handling audio and text files, including secure storage and retrieval using Google Cloud Storage.

Error Handling and Debugging:

- Developed robust error-handling mechanisms to manage potential issues during API calls, file uploads, and processing.

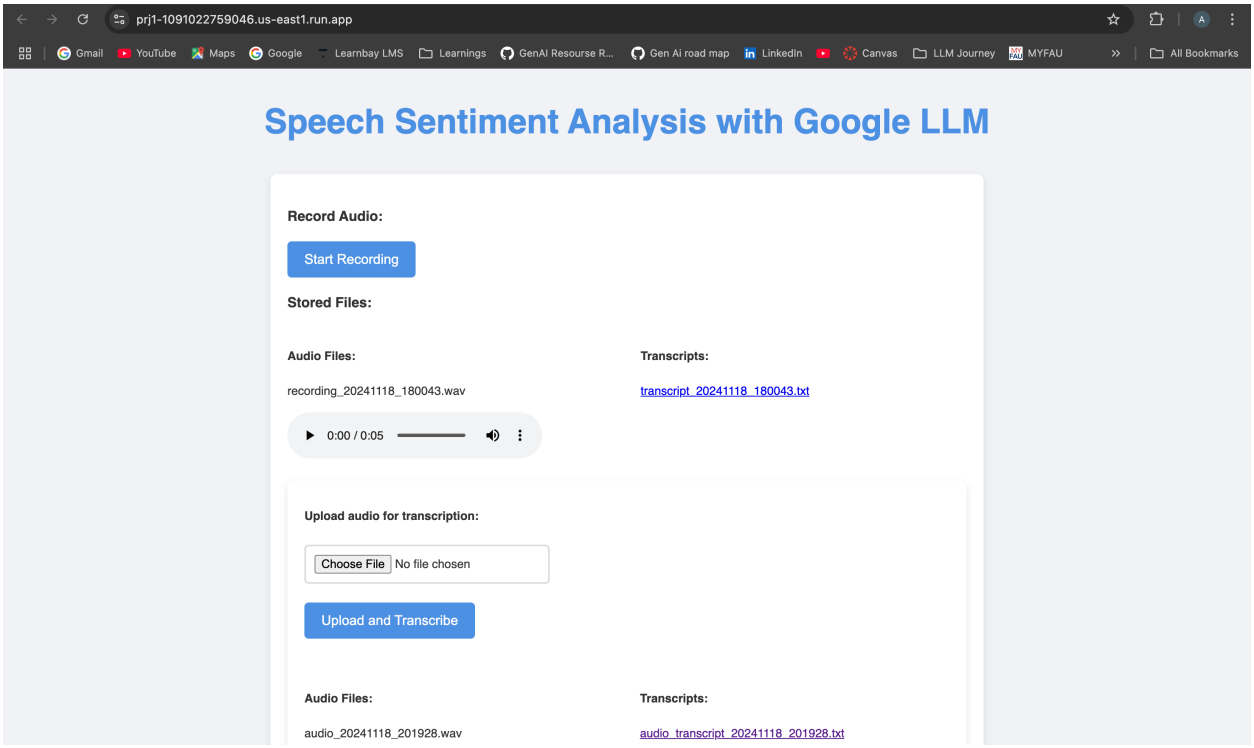
User-Centric Application Design:

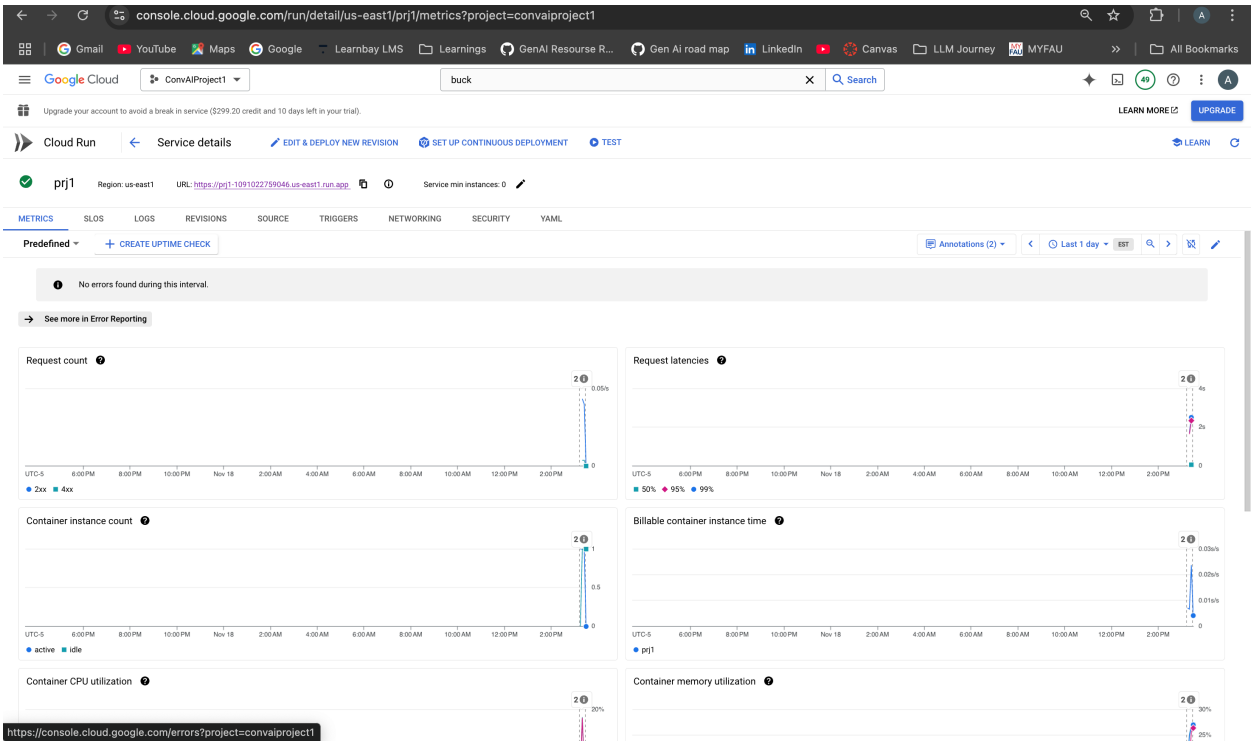
- Improved the application's usability by streamlining the interface for recording, uploading, and displaying results dynamically.

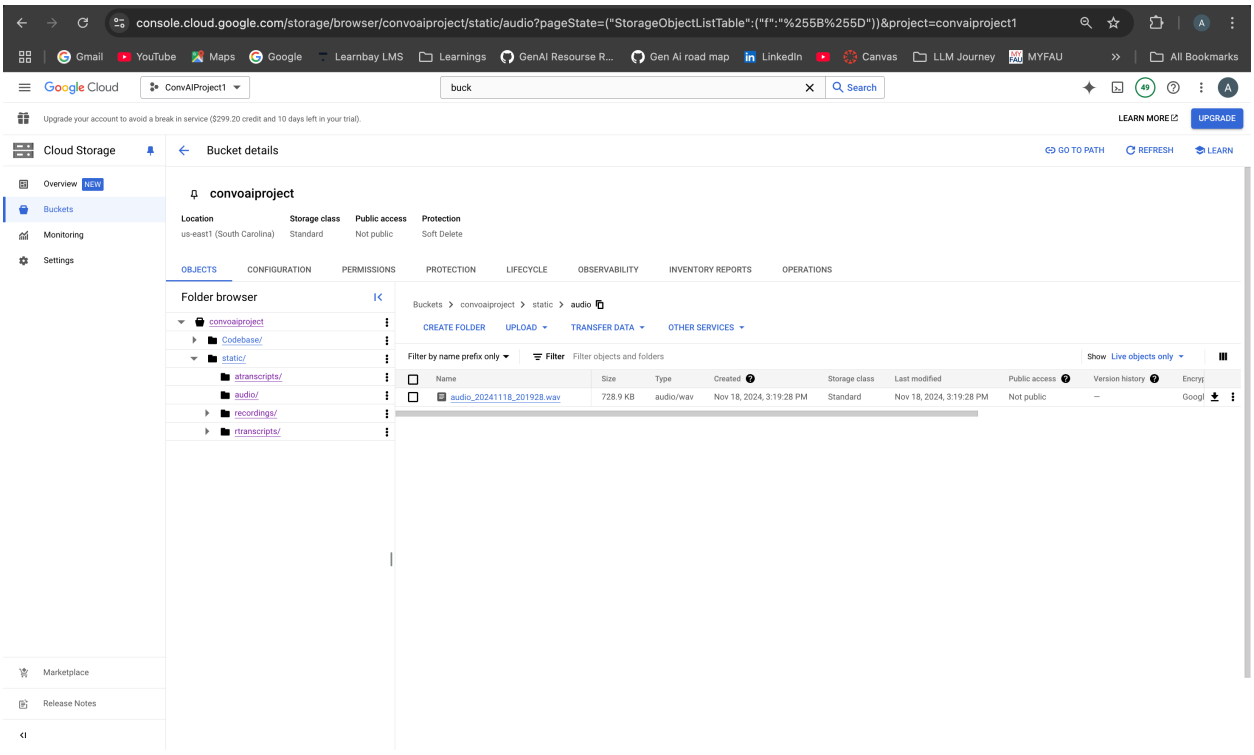
Scalability and Cloud Deployment:

- Understood the benefits of deploying applications on Google Cloud Run for scalability and reliability.

Results:


prj1-1091022759046.us-east1.run.app
Gmail
YouTube
Maps
Google
Learnbay LMS
Learnings
GenAI Resourse R...
Gen Ai road map
LinkedIn
Canvas
LLM Journey
MYFAU
All Bookmarks
Speech Sentiment Analysis with Google LLM
Record Audio:
Start Recording
Stored Files:
Audio Files:
Transcripts:
recording_20241118_180043.wav
transcript_20241118_180043.txt
0:00 / 0:05
Upload audio for transcription:
Choose File
No file chosen
Upload and Transcribe
Audio Files:
Transcripts:
audio_20241118_201928.wav
audio_transcript_20241118_201928.txt


console.cloud.google.com/run/detail/us-east1/prj1/metrics?project=convaiproject1
Google Cloud
ConvAIProject1
buck
Search
Cloud Run
Service details
EDIT & DEPLOY NEW REVISION
SET UP CONTINUOUS DEPLOYMENT
TEST
prj1
Region: us-east1
METRICS
SLOS
LOGS
REVISIONS
SOURCE
TRIGGERS
NETWORKING
SECURITY
YAML
Predefined
CREATE UPTIME CHECK
No errors found during this interval.
See more in Error Reporting
Request count
Request latencies
Container instance count
Billable container instance time
Container CPU utilization
Container memory utilization
https://console.cloud.google.com/errors?project=convaiproject1


console.cloud.google.com/storage/browser/convoaiproject/static/audio?pageState=("StorageObjectListTable":("f":"%255B%255D"))&project=convaiproject1
Google Cloud
ConvAIProject1
buck
Search
Cloud Storage
Bucket details
convoaiproject
Location
us-east1 (South Carolina)
Storage class
Standard
Public access
Not public
Protection
Soft Delete
OBJECTS
CONFIGURATION
PERMISSIONS
PROTECTION
LIFECYCLE
OBSERVABILITY
INVENTORY REPORTS
OPERATIONS
Folder browser
Buckets > convoaiproject > static > audio
CREATE FOLDER
UPLOAD
TRANSFER DATA
OTHER SERVICES
Codebase/
static/
atranscripts/
audio/
recordings/
rtranscripts/
audio_20241118_201928.wav
728.9 KB
audio/wav
Nov 18, 2024, 3:19:28 PM
Standard
Nov 18, 2024, 3:19:28 PM
Not public

Appendix

main.py

```
from flask import Flask, render_template, request, jsonify,send_file
from google.cloud import storage
import google.generativeai as genai
import json
import io
import os
import datetime

app = Flask(__name__,static_folder='static')

storage_client = storage.Client()
bucket_name = 'convoaiproject'
bucket = storage_client.bucket(bucket_name)
TRANSCRIPTS_FOLDER = 'static/rtranscripts/'
AUDIOS_FOLDER = 'static/recordings/'

#read api key
configfile="static/config.json"
with open(configfile, 'r') as file:
    data = json.load(file)
    key=data.get("API_KEY")

genai.configure(api_key=key)

@app.route('/')
def index():
    return render_template('index.html')


def upload_audio_text(file_content,filename,content_type):
    blob = bucket.blob(filename)
    blob.upload_from_string(
    file_content,
    content_type=content_type
    )

@app.route('/upload_audio', methods=['POST'])
def upload_audio():
    try:
        if 'audio_file' not in request.files:
            return "No audio file provided.", 400

        timestamp = datetime.datetime.now().strftime("%Y%m%d_%H%M%S")
        audio_file = request.files['audio_file']
        audio_content = audio_file.read()
        audio_filename = f'static/audio/audio_{timestamp}.wav'
```

```
        upload_audio_text(audio_content,audio_filename,audio_file.content_type)
        text_filename = f'static/atranscripts/audio_transcript_{timestamp}.txt'
        response_text=analyze_audio(audio_filename)

json_response=response_text.strip().removeprefix('```json').removesuffix('```')
        data = json.loads(json_response)
        print(data)
        transcript = data.get("transcription")
        score=data.get("sentiment_analysis")
        content = f"Transcript:\n{transcript}\n\nSentiment: {score}"
        upload_audio_text(content,text_filename,'text/plain')
        return content

    except Exception as e:
        return str(e), 500


@app.route('/upload_recording', methods=['POST'])
def upload_recording():
    try:
        if 'audio' not in request.files:
            return jsonify({'error': 'No audio file provided'}), 400

        audio_file = request.files['audio']

        if audio_file.filename == '':
            return jsonify({'error': 'No selected file'}), 400

        audio_content = audio_file.read()
        if len(audio_content) == 0:
            return jsonify({'error': 'File is empty'}), 400

        # Save audio file to Google Cloud Storage
        timestamp = datetime.datetime.now().strftime("%Y%m%d_%H%M%S")
        audio_filename = f'static/recordings/recording_{timestamp}.wav'
        upload_audio_text(audio_content,audio_filename,audio_file.content_type)
        text_filename = f'static/rtranscripts/transcript_{timestamp}.txt'
        # Save the transcript to Google Cloud Storage
        audio_file_uri=f"static/recordings/recording_{timestamp}.wav"
        response_text=analyze_audio(audio_file_uri)

json_response=response_text.strip().removeprefix('```json').removesuffix('```')
        data = json.loads(json_response)
        transcript = data.get("transcription")
        score=data.get("sentiment_analysis")
        content = f"Transcript:\n{transcript}\n\nSentiment : {score}"
        upload_audio_text(content,text_filename,'text/plain')
```

```
        return jsonify({'message': 'Success', 'audio_url': audio_filename,
'transcript_url': text_filename}), 200

    except speech.exceptions.GoogleAPICallError as api_error:
        print(f"Google Speech API Error: {str(api_error)}")
        return jsonify({'error': 'Speech-to-Text processing failed'}), 500

    except Exception as e:
        print(f"Error in upload_recording: {str(e)}")
        return jsonify({'error': str(e)}), 500

@app.route('/list_files')
def list_files():
    transcripts = []
    audios = []
    u_trans=[]
    u_audios=[]

    # List  transcript files
    blobs = bucket.list_blobs(prefix='static/rtranscripts/')
    for blob in blobs:
        if blob.name.endswith(".txt"):
            transcripts.append({
                'name': blob.name.split("/")[-1],
                'url': f'/serve_text/{blob.name}'
            })

    # List recording  files
    ablobs = bucket.list_blobs(prefix='static/recordings/')
    for blob in ablobs:
        if blob.name.endswith(".wav"):
            audios.append({
                'name': blob.name.split("/")[-1],
                'url': f'/serve_audio/{blob.name}'
            })

    # List uploaded text files
    ublobs = bucket.list_blobs(prefix='static/atranscripts/')
    for blob in ublobs:
        if blob.name.endswith(".txt"):
            u_trans.append({
                'name': blob.name.split("/")[-1],
                'url': f'/serve_text/{blob.name}'
            })

    # List uploaded audio files
    ua_blobs = bucket.list_blobs(prefix='static/audio/')
    for blob in ua_blobs:
```

```
        if blob.name.endswith(".wav"):
            u_audios.append({
                'name': blob.name.split("/")[-1],
                'url': f'/serve_audio/{blob.name}'
            })

    return jsonify({
        'transcripts': transcripts,
        'audios': audios,
        'u_trans' : u_trans,
        'u_audios' : u_audios
    })

@app.route('/serve_text/<path:filename>')
def serve_txt(filename):
    try:
        blob = bucket.blob(filename)
        content = blob.download_as_text()
        return content, 200, {'Content-Type': 'text/plain'}
    except Exception as e:
        return jsonify({'error': 'An error occurred while serving the transcript
file'}), 500

# Add this new route to serve audio files
@app.route('/serve_audio/<path:filename>')
def serve_audio(filename):
    blob = bucket.blob(filename)

    file_bytes = blob.download_as_bytes()
    return send_file(
        io.BytesIO(file_bytes),
        mimetype='audio/wav',
        as_attachment=True,
        download_name=filename.split('/')[-1]
    )

##audio to transcript

def analyze_audio(audio_file_uri):
    try:
        blob_path = audio_file_uri
        bucket = storage_client.bucket(bucket_name)
        blob = bucket.blob(audio_file_uri)

        # Download the audio file content
        audio_data = blob.download_as_bytes()

        # Configure Gemini
```

```
        generation_config = {
            "temperature": 0.1,
            "top_p": 0.95,
            "top_k": 64,
            "max_output_tokens": 8192,
        }

        # Initialize Gemini model
        model = genai.GenerativeModel(
            model_name="gemini-1.5-pro",
            generation_config=generation_config
        )
        contents = {
            "parts": [
                {
                    "text":"""transcribe this WAV format audio and provide the
transcript and sentiment analysis score in JSON format. If no transcript is available,
                            - return 0 for the sentiment analysis score,if transcript
was available return the esntiment like Possitive,Negative,Nuetral. The JSON response
format should be as follows
                            -json reponse:{
                                            "transcription": "full transcribed text
here",
                                            "sentiment_analysis": score
                                        }

                        - output should just json

                    """

                },
                {
                    "inline_data": {
                        "mime_type": "audio/wav",
                        "data": audio_data
                    }
                }
            ]
        }

        response = model.generate_content(contents)
        print(response)
        return response.text

    except Exception as e:
        return json.dumps({
            "error": str(e),
            "location": "Error accessing file in GCS bucket"
```

```
        })


# Example usage
if __name__ == '__main__':
    app.run(host="0.0.0.0", port=8080, debug=True)
```

HTML
templates/index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Text-to-Speech and Speech-to-Text</title>
    <style>
        body {
            font-family: 'Open Sans', sans-serif;
            margin: 0;
            padding: 0;
            background-color: #f0f2f5;
            color: #333;
        }
        h1 {
            text-align: center;
            font-size: 2.5rem;
            color: #4A90E2;
            margin-top: 30px;
        }
        .section {
            margin: 20px auto;
            padding: 20px;
            max-width: 800px;
            background-color: #fff;
            border-radius: 8px;
            box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
        }
        label {
            font-weight: bold;
            color: #333;
```

```
}
.text-audio-container {
    display: flex;
    align-items: flex-start;
    justify-content: space-between;
    margin-bottom: 20px;
}
textarea {
    width: 50%;
    height: 100px;
    padding: 10px;
    border: 2px solid #ddd;
    border-radius: 6px;
    transition: border-color 0.3s;
}
textarea:focus {
    border-color: #4A90E2;
    outline: none;
}
button {
    padding: 12px 20px;
    background-color: #4A90E2;
    color: white;
    border: none;
    border-radius: 5px;
    font-size: 1rem;
    cursor: pointer;
    transition: background-color 0.3s;
}
button:hover {
    background-color: #357ABD;
}
button:disabled {
    background-color: #B0C4DE;
    cursor: not-allowed;
}
.audio-player, .transcription-player {
    margin-top: 20px;
    margin-left: 10px;
    width:50%;
}
.audio-player h3, .transcription-player h3 {
    font-size: 1.1rem;
    margin-bottom: 5px;
    color: #333;
}
audio {
    width: 100%;
```

```
    max-width: 300px;
}
input[type="file"] {
    padding: 10px;
    background-color: #fff;
    border: 2px solid #ddd;
    border-radius: 5px;
    cursor: pointer;
    transition: border-color 0.3s;
}
input[type="file"]:hover {
    border-color: #4A90E2;
}
.upload-transcription-container {
    display: flex;
    flex-direction: column;
    gap: 20px;
}
.upload-section, .transcription-section {
    flex: 1;
    margin-top: 10px;
}
.transcription-section h3 {
    color: #4A90E2;
}
.recording-section {
    margin-top: 30px;
}
#recorded_audio_player {
    display: none;
    margin-top: 10px;
    align-items: center;
}
#transcription_text {
    font-family: monospace;
    background-color: #f7f7f7;
    padding: 10px;
    border-radius: 4px;
    white-space: pre-wrap;
}
#upload_recording {
    margin-left: 20px;
    padding: 8px 15px;
    background-color: #28a745;
    border-radius: 4px;
}
#upload_recording:hover {
    background-color: #218838;
```

```
        }
        footer {
            text-align: center;
            margin: 20px 0;
            font-size: 0.9rem;
            color: #777;
        }
        .file-container {
            display: flex;
            justify-content: space-between;
            margin-top: 20px;
        }

        .file-container > div {
            width: 48%;
        }
        ul {
        list-style-type: none;
        padding: 0;
        margin: 0;
    }
    </style>
</head>
<body>
    <h1>Speech Sentiment Analysis with Google LLM</h1>

     <!-- Audio Recording Section -->
     <div class="section recording-section">
        <h3>Record Audio:</h3>
        <button id="start_recording">Start Recording</button>
        <button id="stop_recording" style="display:none;">Stop Recording</button>
        <div id="recorded_audio_player" class="audio-player">
            <h3>Recorded Audio:</h3>
            <audio id="recorded_audio" controls></audio>
            <button id="upload_recording">Upload Recording</button>
        </div>
        <h3>Stored Files:</h3>
    <div class="file-container">
        <div>
            <h4>Audio Files:</h4>
            <ul id="audio_list"></ul>
        </div>
        <div>
            <h4>Transcripts:</h4>
            <ul id="transcript_list"></ul>
        </div>
    </div>
    <!-- Upload Audio Section -->
```

```
    <div class="section">
        <form action="/upload_audio" method="POST" enctype="multipart/form-data">
            <div class="upload-transcription-container">
                <div class="upload-section">
                    <label for="audio_file">Upload audio for
transcription:</label><br><br>
                    <input type="file" id="audio_file" name="audio_file"
accept="audio/wav"><br><br>
                    <button type="submit">Upload and Transcribe</button>
                </div>
                <div class="file-container">
                    <div>
                        <h4>Audio Files:</h4>
                        <ul id="uaudio_list"></ul>
                    </div>
                    <div>
                        <h4>Transcripts:</h4>
                        <ul id="utranscript_list"></ul>
                    </div>
                </div>

            </div>
        </form>
    </div>
    <script>
        let mediaRecorder;
        let recordedChunks = [];

        document.getElementById('start_recording').addEventListener('click', async ()
=> {
            const stream = await navigator.mediaDevices.getUserMedia({ audio: true });
            mediaRecorder = new MediaRecorder(stream);
            mediaRecorder.start();

            mediaRecorder.ondataavailable = event => {
                recordedChunks.push(event.data);
            };

            mediaRecorder.onstop = () => {
                const audioBlob = new Blob(recordedChunks, { type: 'audio/wav' });
                const audioUrl = URL.createObjectURL(audioBlob);
                document.getElementById('recorded_audio').src = audioUrl;
                document.getElementById('recorded_audio_player').style.display =
'flex';
            };

            document.getElementById('stop_recording').style.display = 'inline-block';
            document.getElementById('start_recording').style.display = 'none';
```

```
        });

        document.getElementById('stop_recording').addEventListener('click', () => {
            mediaRecorder.stop();
            document.getElementById('stop_recording').style.display = 'none';
            document.getElementById('start_recording').style.display = 'inline-block';
        });

        document.getElementById('upload_recording').addEventListener('click', async ()
=> {
            if (recordedChunks.length === 0) {
                alert('No recording available. Please record audio first.');
                return;
            }

            const audioBlob = new Blob(recordedChunks, { type: 'audio/wav' });
            const formData = new FormData();
            formData.append('audio', audioBlob, 'recording.wav');

            try {
                const response = await fetch('/upload_recording', {
                    method: 'POST',
                    body: formData
                });
                const data = await response.json();

                if (data.audio_url) {
                    alert(`Recording uploaded! Access it at: ${data.audio_url}`);
                    window.location.href = '/';
                } else {
                    alert('Error uploading recording: ' + (data.error || 'Unknown
error'));
                    window.location.href = '/';
                }
            } catch (error) {
                console.error('Error:', error);
                alert('Error uploading recording: ' + error.message);
            }

            recordedChunks = [];
        });

document.querySelector('form[action="/upload_audio"]').addEventListener('submit',
function (e) {
            e.preventDefault();
            const formData = new FormData(this);

            fetch('/upload_audio', {
```

```
            method: 'POST',
            body: formData
        })
        .then(response => {
            if (response.ok) {
                window.location.href = '/';
                return response.text();
            } else {
                return Promise.reject('Upload failed: ' + response.statusText);
            }
        })
        .catch(error => console.error('Error:', error));
    });

fetch('/list_files')
    .then(response => response.json())
    .then(data => {
        const transcriptList = document.getElementById('transcript_list');
        const audioList = document.getElementById('audio_list');
        const utranscriptList = document.getElementById('utranscript_list');
        const uaudio_list = document.getElementById('uaudio_list');
        data.transcripts.forEach(file => {
            const listItem = document.createElement('li');
            const link = document.createElement('a');
            link.href = file.url
            link.textContent = file.name;
            link.target = "_blank";
            listItem.appendChild(link);
            transcriptList.appendChild(listItem);
        });
        data.audios.forEach(file => {
            const listItem = document.createElement('li');
            const audioPlayer = document.createElement('audio');
            audioPlayer.controls = true;
            audioPlayer.src = file.url;
            const fileName = document.createElement('p');
            fileName.textContent = file.name;

            listItem.appendChild(fileName);
            listItem.appendChild(audioPlayer);
            audioList.appendChild(listItem);
        });

        data.u_trans.forEach(file => {
            const listItem = document.createElement('li');
            const link = document.createElement('a');
            link.href = file.url
            link.textContent = file.name;
```

```
                link.target = "_blank";
                listItem.appendChild(link);
                utranscriptList.appendChild(listItem);
            });

            data.u_audios.forEach(file => {
                const listItem = document.createElement('li');
                const audioPlayer = document.createElement('audio');
                audioPlayer.controls = true;
                audioPlayer.src = file.url;
                const fileName = document.createElement('p');
                fileName.textContent = file.name;
                listItem.appendChild(fileName);
                listItem.appendChild(audioPlayer);
                uaudio_list.appendChild(listItem);
            });
        })
        .catch(error => console.error('Error fetching files:', error));
    </script>
</body>
</html>
```

Requirements.txt

```
Flask==3.0.2
gunicorn==20.1.0
google-cloud-storage==2.18.2
google-generativeai==0.5.0
```

procfile

web: gunicorn --bind :$PORT --workers 1 --threads 4 --timeout 120 main:app

config.json

stati/config.json

```
{
  "API_KEY": "INSERT_YOUR_API_KEY_HERE"
}
```