cambai-challenge

March 5, 2024

1 Data Engineering Pipeline for Creating a TTS Dataset for the Irish Language from YouTube

Data Science Position Challenge: Design and implement a data engineering pipeline for creating a Text-to-Speech (TTS) dataset for the Irish language from Youtube. Your submission should include a comprehensive document outlining your proposed pipeline, code, and a subset of the collected dataset formatted similarly to LibriTTS (yes we want you to actually execute your data pipeline and collect 1 Gb of data as representative sample of your pipeline actually working), showcasing your approach to data collection, processing, and management.

Introduction The aim of this project was to construct a data engineering pipeline dedicated to generating a Text-to-Speech (TTS) dataset for the Irish language, utilizing the rich array of content available on YouTube. This pipeline was specifically designed to automate the extraction, processing, and formatting of audio and textual data to adhere to LibriTTS specifications.

Approach and Design Decisions

1. **Data Collection & Filtering**: Initially, YouTube Data API V3 method for data collection was explored as documented in the "Extracting Irish Channel IDs and Videos using YT Data API V3" section of this notebook. However, due to the API rate limits and the inaccuracy of transcription for the Irish language, these methods proved unsatisfactory. Consequently, the project shifted towards utilizing `yt-dlp`, a command-line tool that surpassed the YouTube Data API's limitations, facilitating more extensive and flexible data collection.
2. **Transcript Extraction and Filtering**: After experimenting with different approaches, the pipeline successfully implemented a method bypassing the limitations encountered with the YouTube Data API and Google Cloud's Speech-to-Text API as mention in the section "Extracting Captions/ Text". YT Data API V3 support extracting captions ONLY for videos which the user is authorized to. I also tried extracting audio first and then using Speech-to-Text API to extract text from the audio, however that is not accurate. The final approach utilized `YouTubeTranscriptApi` to directly extract transcripts, ensuring a more reliable and focused retrieval of Irish language transcripts.
3. **Audio Extraction**: Leveraging `yt-dlp`, the pipeline efficiently extracted audio tracks from downloaded YouTube videos. This tool was chosen for its robustness and compatibility with a wide range of formats, ensuring the audio data maintained a consistent standard necessary for TTS applications.
4. **Data Processing and Formatting**: Following the extraction, audio files were aligned with

their respective transcripts, and the data was organized following the structure of the LibriTTS dataset. This involved detailed formatting to ensure compatibility with TTS systems, highlighting the necessity of precise alignment and standardized data representation.

5. **LibriTTS Format Creation**: In adherence to LibriTTS standards, the dataset was meticulously organized into structured directories, each containing segmented trasncript files and their corresponding audio and JSON and Text captions/transcriptions. This structure is essential for TTS model training, facilitating the easy location and association of text with spoken audio.

6. **Data Quality and Packaging**: The final phase involved conducting thorough quality checks to validate the dataset's integrity and usability. The completed dataset was then packaged, ready for distribution and utilization in TTS systems, ensuring a focus on the Irish language.

Challenges and Solutions Navigating API limitations was a significant challenge, resolved by transitioning to `yt-dlp` for video collection and direct transcript extraction. This adaptation was crucial in broadening the dataset and ensuring the inclusion of authentic Irish language content.

Another notable challenge was the accurate extraction and verification of Irish language transcripts. By opting for direct transcript fetching and setting stringent filtering criteria, the pipeline enhanced the relevance and quality of the textual data, crucial for a nuanced language such as Irish.

Conclusion This pipeline underscores the innovative use of online video content for linguistic resource development, particularly for underrepresented languages. While tailored for the Irish language, its adaptable framework can serve other languages and dialects. The project encapsulates a holistic approach to TTS dataset creation, from strategic data collection to meticulous formatting in line with LibriTTS standards, marking a significant stride in the realm of language technology.

2 Designing LibriTTS Data Format

2.0.1 Adaptation for Our Dataset

Due to the unique nature of our dataset, which consists of a single continuous audio file rather than individual clips for each transcript segment, and the lack of distinct speaker information, I have modified the conventional LibriTTS format to accommodate our specific requirements. This adaptation maintains the organizational structure of the original format while addressing the unique aspects of our data.

2.0.2 Directory and File Structure

The dataset is organized to reflect a single, continuous audio context. Files are organized based on a unique identifier, which in our case is the `Video_ID`, corresponding to the time of the dataset's creation or recording. This structure accommodates our continuous audio file and segmented transcript references efficiently.

Directory Structure Example: `LibriTTS-Irish/`

```
LibriTTS-Irish/
    V-Hmc-MRRiU/                # Unique Video ID directory
```

```
V-Hmc-MRRiU.wav          # The single, continuous audio file
segments/                # Directory for text segment files
    0000_V-Hmc-MRRiU_09700.txt          # Text file for a specific segment
    0000_V-Hmc-MRRiU_12440.txt          # Another segment text file
    ...
V-Hmc-MRRiU.json         # Complete JSON transcript
V-Hmc-MRRiU.txt          # Complete text transcript
```

File Naming Conventions:

- `V-Hmc-MRRiU` - Represents the unique `Video_ID`.
- `0000_V-Hmc-MRRiU_09700`: Here, `0000` is an arbitrary number reflecting a speaker ID (used when specific speaker information is absent),`V-Hmc-MRRiU` is the video_id, and `09700` represents the start time of the segment in milliseconds.
- `V-Hmc-MRRiU.wav`: This is the single audio file in .wav format.
- `V-Hmc-MRRiU.json` and `V-Hmc-MRRiU.txt`: These are the transcript files for the entire audio, provided in two different formats.

2.0.3 Text Files and Time Stamps:

For each segment, create a text file in the `segments` directory. These files should contain the transcript text and the corresponding time range from the continuous audio file.

Example:

- **File Name:** `0000_V-Hmc-MRRiU_09700.txt`
- **File Content:** This includes the transcript segment from the audio.

  ```
  Dia dhaoibh ar fad, a chairde Gaeil.
  ```

- **Associated JSON Transcript File (`V-Hmc-MRRiU.json`):** This file includes text segments, start times, and durations. Below is an example structure:

  ```json
  [
      {
        "text": "Dia dhaoibh ar fad, a chairde Gaeil.",
        "start": 9.7,
        "duration": 2.0
      },
      {
        "text": "Is c\u00fais \u00e1thais dom beannachta\u00ed na hAthbhliana a ghu\u00ed or
        "start": 12.24,
        "duration": 2.88
      },
      {
        "text": "ar bhur dteaghlaigh, agus ar ghach uile dhuine at\u00e1 bailithe le ch\u00e
        "start": 15.26,
        "duration": 5.96
      },
  ```

```
    {
      "text": "T\u00e1im an-sh\u00e1sta t\u00fas a chur le Bliain na Gaeilge,",
      "start": 22.26,
      "duration": 3.16
    },
  ...
]
```

- **Associated Text Transcript File (`V-Hmc-MRRiU.txt`):** This file includes the full transcript corresponding to the audio file segments:

```
  Dia dhaoibh ar fad, a chairde Gaeil.
  Is cúis áthais dom beannachtaí na hAthbhliana a ghuí oraibh,
  ar bhur dteaghlaigh, agus ar ghach uile dhuine atá bailithe le chéile anocht ar Oíche Ch
  Táim an-shásta tús a chur le Bliain na Gaeilge,
  bliain ina ndéanfar ceiliúradh ar 125 bliain d'athbheochan na Gaeilge in Éirinn.
  Ar ndóigh, bhí ról nach beag ag Uachtaráin na hÉireann i gcur chun cinn na Gaeilge
  - chomh fada siar leis an gcéad Uachtarán, Dubhghlas de hÍde,
  ceannródaí na hathbheochana, a bhí mar chathaoirleach ag an gcruinniú, i mí Iúil 1893
  nuair a bunaíodh Conradh na Gaeilge.
  Is é an chuspóir a leag de hÍde, Eoin Mac Néill, agus a chomhleacaithe amach dóibh fhéir
  nuair a chuireadar tús le hobair pobal-bhunaithe na hAthbheochana, ná an Ghaeilge a choi
...
```

“‘

3 Code implementation with comments

[1]:
```python
import pandas as pd
from googleapiclient import discovery
import json
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
from googleapiclient.discovery import build
from googleapiclient.errors import HttpError
import io
import os
import html
from google.auth.transport.requests import Request
from google.oauth2.credentials import Credentials
from googleapiclient.http import MediaIoBaseDownload
from google_auth_oauthlib.flow import InstalledAppFlow
from googleapiclient.discovery import build
from youtube_transcript_api import YouTubeTranscriptApi
from youtube_transcript_api.formatters import TextFormatter
from youtube_transcript_api.formatters import JSONFormatter
from pathlib import Path
```

```
import subprocess
import shutil
import time
```

3.1 Extracting Irish Channel IDs and Videos using YT Data API V3

[2]:
```
API_KEY = 'AIzaSyAHwxmd1IJS37O4vdurw52tdTMZ39kYQnU'

keyword_list = [
    'Gaeilge le haghaidh', 'Foghlaim Gaeilge',
#     'Cúrsa Gaeilge', 'Nuacht Gaeilge', 'Stair Éireann',
#     'Gaeilge Labhartha', 'Bunús na Gaeilge', 'Gaeilge Bheo', 'Cultúr na␣
↪hÉireann', 'Gaeilge Coláiste',
#     'Gaeilge Scoile', 'Éireannach', 'TG4 Gaeilge', 'RTÉ Gaeilge', 'Pobal␣
↪Gaeilge', 'Gaeilge Aclaí',
#     'Gaeilge Físeán', 'Ceachtanna Gaeilge', 'Gaeilge Amhrán'  # 'Gaeilge␣
↪Amhrán' cautiously included, can skip if music is often returned
]
```

[3]:
```
def find_irish_channels(api_key, keyword_list, max_results_per_keyword=10):
    youtube = build('youtube', 'v3', developerKey=api_key)
    all_channels = set()  # Use a set to avoid duplicates

    for keyword in keyword_list:
        try:
            search_response = youtube.search().list(
                q=keyword,
                part='id,snippet',
                maxResults=max_results_per_keyword,
                type='channel',
                regionCode='IE'
            ).execute()

            for search_result in search_response.get('items', []):
                if search_result['id']['kind'] == 'youtube#channel':
                    all_channels.add(search_result['id']['channelId'])

        except Exception as e:
            print(f"An error occurred with keyword '{keyword}': {e}")

    return list(all_channels)  # Convert the set back to a list
```

[4]:
```
irish_channels = find_irish_channels(API_KEY, keyword_list,␣
↪max_results_per_keyword=100)
```

[5]:
```
len(irish_channels)
```

[5]: 30

Here, I initially extracted Channel IDs specific to the Irish region, followed by the extraction of videos from these channels. This approach was adopted to ensure the procurement of videos exclusively in the Irish language, thereby eliminating content featuring a mixture of languages, such as English and Irish.

Additional Irish Channel IDs can be incorporated manually, or the search criteria can be expanded to identify more Channel IDs, consequently increasing the collection of Video URLs.

[6]:
```
from googleapiclient.discovery import build

def youtube_search(keywords, api_key, max_results=10, channel_ids=None):
    youtube = build('youtube', 'v3', developerKey=api_key)
    videos = []
    videoids = []
    try:
        for channel_id in (channel_ids or ['']):  # Iterate through channels if␣
 ↪provided, else search broadly
            search_response = youtube.search().list(
                q=keywords,
                part='id,snippet',
                maxResults=max_results,
                type='video',
                regionCode='IE',
                videoCaption='closedCaption',  # Ensure videos have closed␣
 ↪captions
                channelId=channel_id  # Search within a specific channel if␣
 ↪provided
            ).execute()

            for search_result in search_response.get('items', []):
                if search_result['id']['kind'] == 'youtube#video':
                    videoids.append(search_result['id']['videoId'])
                    videos.append(f"https://www.youtube.com/watch?
 ↪v={search_result['id']['videoId']}")

            if not channel_ids:  # Break early if not searching specific␣
 ↪channels
                break

    except Exception as e:
        print(f"An error occurred: {e}")
        return videos

    return videoids
```

```
[7]: irish_videos = youtube_search(keyword_list, API_KEY, 10, None)
```

```
[8]: irish_videos
```

```
[8]: ['Gpae1kmFtV8',
      'oLuinmqHZf4',
      '5OdhIXAAjbw',
      'yeMzwtNe0tU',
      'vzlIxabdzR4',
      'VdNIpSZwZ_0',
      'ZIfkUPQszPU',
      'LIghiqb4rpI',
      '9W6l_u8ef34',
      'jEBBKS5sn6E']
```

3.1.1 Final Method Implemented

```
[9]: def search_videos(keywords, ytdlp_path, region_code, max_results):
         """Search for videos on YouTube based on keywords."""
         video_ids = []

         for keyword in keywords:
             # Added '--geo-bypass-country' with 'IE' to simulate access from Ireland
             command = [ytdlp_path, '--geo-bypass-country', region_code, '--get-id',␣
      ↪f'ytsearch{max_results}:{keyword}']
             result = subprocess.run(command, stdout=subprocess.PIPE,␣
      ↪stderr=subprocess.PIPE, text=True)
             if result.stdout:
                 # Extend the video_ids list with the new ids found
                 video_ids.extend(result.stdout.strip().split('\n'))
         return video_ids
```

```
[10]: ytdlp_path = '/Users/nikhiljindal/opt/anaconda3/envs/myenv/bin/yt-dlp'
      ffmpeg_path = '/opt/homebrew/bin/ffmpeg'
      region_code = 'IE'
      max_results = 10
      video_ids = search_videos(keyword_list,ytdlp_path,region_code, max_results)
      print(video_ids)
```

```
['V-Hmc-MRRiU', 'LpVMYXWhErU', '3eC2XbN1-Bg', 'cQsm9JPe794', 'TZATuiLuTvQ',
'FXL2aE5uUXc', 'XQdGShFcDaQ', 'Q8UhNJw3bWE', 'TIb0FhBC1QM', 'PXhDlYREJyo',
'fs5fqq2YumA', 'i4ATd-q_YZM', 'uPqoKpyhzP8', 'CUM9T5K8lrI', 'sQgvS2A8Xz8',
'lDxkKsnB7cg', 'V3jUiApohGQ', 'CnrQmSIkP1E', 'wraRj_ch2rU', 'lP1FXSktMtg']
```

```
[11]: len(video_ids)
```

```
[11]: 20
```

The Video IDs extracted above have been filtered based on location and keywords. However, it is important to note that not all videos are in the Irish language, as the yt-dlp tool lacks a specific language filtering feature. In the subsequent section, I will address this issue.

3.2 Extracting Captions/ Text

3.2.1 Method 1: Using YT Data API V3

[12]:
```
def get_authenticated_service():
    creds = None
    # The file token.json stores the user's access and refresh tokens.
    if os.path.exists('token.json'):
        creds = Credentials.from_authorized_user_file('token1.json',␣
↪scopes=['https://www.googleapis.com/auth/youtube.force-ssl'])
    # If there are no (valid) credentials available, let the user log in.
    if not creds or not creds.valid:
        if creds and creds.expired and creds.refresh_token:
            creds.refresh(Request())
        else:
            flow = InstalledAppFlow.from_client_secrets_file('client_secrets.
↪json', scopes=['https://www.googleapis.com/auth/youtube.force-ssl'])
            creds = flow.run_local_server(port=0)
        # Save the credentials for the next run
        with open('token.json', 'w') as token:
            token.write(creds.to_json())

    return build('youtube', 'v3', credentials=creds)

# Use this authenticated service to make API calls
youtube = get_authenticated_service()
```

[13]:
```
def download_captions(youtube, video_id, language='en'):
    try:
        # Retrieve a list of available caption tracks
        caption_list = youtube.captions().list(part='snippet',␣
↪videoId=video_id).execute()
        print(caption_list)
        caption_id = None
        for item in caption_list.get('items', []):
            if item['snippet']['language'] == language and␣
↪item['snippet']['trackKind'] != 'ASR':
                caption_id = item['id']
                break

        if caption_id:
            # Download the caption track
            request = youtube.captions().download(id=caption_id)
            fh = io.FileIO(f"{video_id}.srt", mode='wb')
            downloader = MediaIoBaseDownload(fh, request)
            done = False
            while not done:
                _, done = downloader.next_chunk()
            fh.close()  # Ensure the file is closed after download
            print(f"Downloaded captions for {video_id}")
            return f"{video_id}.srt"
        else:
            print(f"No suitable caption track found for {video_id}")
            return None
    except HttpError as e:
        print(f"An error occurred: {e}")
        return None
```

[14]:
```
download_captions(youtube, 'jBN2_YuTclU')
```

```
{'kind': 'youtube#captionListResponse', 'etag': 'aDeprX_OcnL6B7s1iXY9xARqB34',
'items': [{'kind': 'youtube#caption', 'etag': 'LLr_yKS8UqCg2tHofDhw2YLW8U8',
'id': 'AUieDaaeetUJNobcsPOKKTl07gjyISnMbN26WnAoPh8sQdSKU1I', 'snippet':
{'videoId': 'jBN2_YuTclU', 'lastUpdated': '2023-11-13T21:21:27.926775Z',
'trackKind': 'asr', 'language': 'en', 'name': '', 'audioTrackType': 'unknown',
'isCC': False, 'isLarge': False, 'isEasyReader': False, 'isDraft': False,
'isAutoSynced': False, 'status': 'serving'}}]}
An error occurred: <HttpError 403 when requesting https://youtube.googleapis.com
/youtube/v3/captions/AUieDaaeetUJNobcsPOKKTl07gjyISnMbN26WnAoPh8sQdSKU1I?
returned "The permissions associated with the request are not sufficient to
download the caption track. The request might not be properly authorized, or the
video order might not have enabled third-party contributions for this caption.".
Details: "[{'message': 'The permissions associated with the request are not
sufficient to download the caption track. The request might not be properly
authorized, or the video order might not have enabled third-party contributions
for this caption.', 'domain': 'youtube.caption', 'reason': 'forbidden',
'location': 'id', 'locationType': 'parameter'}]">
```

The method previously described will not function due to modifications in the YouTube DATA API. Please consult the following articles for further information.

- https://issuetracker.google.com/issues/241669016?pli=1
- https://www.reddit.com/r/GoogleAppsScript/comments/186e4ot/is_it_even_possible_to_download_any
- https://stackoverflow.com/questions/30653865/downloading-captions-always-returns-a-403

3.2.2 Method 2: Using Google Cloud Speech to Text

[15]:
```
# # Ensure you have your correct path to the credentials file
# credentials = service_account.Credentials.
 ↪from_service_account_file('camb_StoT.json')

# def convert_to_mono(stereo_file_path):
#     sound = AudioSegment.from_wav(stereo_file_path)
#     mono_sound = sound.set_channels(1)
#     mono_sound.export(stereo_file_path, format="wav")

# def transcribe_audio(speech_file, lang="ga-IE"):
#     # Convert stereo audio to mono
#     convert_to_mono(speech_file)

#     # Check the file size after conversion
#     file_size = os.path.getsize(speech_file)
#     if file_size > 10485760:  # File size greater than 10 MB
#         print(f"Skipping large file: {speech_file} (Size: {file_size} bytes)")
#         return None

#     client = speech.SpeechClient(credentials=credentials)
#     with io.open(speech_file, 'rb') as audio_file:
#         content = audio_file.read()

#     audio = speech.RecognitionAudio(content=content)
#     config = speech.RecognitionConfig(
#         encoding=speech.RecognitionConfig.AudioEncoding.LINEAR16,
#         sample_rate_hertz=48000,  # Adjust based on your actual files
#         language_code=lang,
#         enable_word_time_offsets=True
#     )

#     operation = client.long_running_recognize(config=config, audio=audio)
#     print('Waiting for operation to complete...')
#     response = operation.result(timeout=90)  # Increase timeout as needed

#     for result in response.results:
#         print("Transcript: {}".format(result.alternatives[0].transcript))

# audio_path = 'data/uG1mplJey4M.wav'
# transcribe_audio(audio_path)
```

This method lacks precision and has been observed to produce incorrect English text instead of Irish.

3.2.3 Final Method Implemented: Using YT transcript API

https://pypi.org/project/youtube-transcript-api/

[16]:
```
#pip install youtube-transcript-api
```

[18]:
```
def create_transcript_text_files(video_ids, main_output_directory, language):
    irish_video_ids = []
    skip = 0
    for video_id in video_ids:
        output_directory = os.path.join(main_output_directory, video_id)

        # Initialize the path for the transcript file without creating the␣
 ↪directory yet
        output_file_path = os.path.join(output_directory, f"{video_id}.txt")

        # Check if the transcript file already exists and is not empty
        if os.path.exists(output_file_path) and os.path.
 ↪getsize(output_file_path) > 0:
            print(f"Text Transcript file already exists and is not empty for␣
 ↪{video_id}")
            continue  # Skip to the next file

        try:
            # Attempt to fetch the transcript in the Irish language
            transcript = YouTubeTranscriptApi.get_transcript(video_id,␣
 ↪languages=[language])
            if transcript:
                # If the transcript exists, create the directory (if it doesn't␣
 ↪exist already)
                if not os.path.exists(output_directory):
                    os.makedirs(output_directory)

                formatter = TextFormatter()

                # .format_transcript(transcript) turns the transcript into a␣
 ↪text string.
                text_formatted = formatter.format_transcript(transcript)

                # Write the formatted text content to a file
                with open(output_file_path, 'w', encoding='utf-8') as␣
 ↪output_file:
                    output_file.write(text_formatted)
                print(f"Text Transcript file created for video id: {video_id}")
                irish_video_ids.append(video_id)
        except Exception as e:
            skip += 1
```

```
#               print(f"An error occurred for videos_id {video_id}: {e}")
            # If there's an error, and directory was created without files,␣
↪remove it
            if os.path.exists(output_directory) and not os.
↪listdir(output_directory):
                os.rmdir(output_directory)

    print(f"Skip Counter: {skip}")
    return irish_video_ids
```

[19]:
```
output_directory = "Test/"
language = 'ga'
irish_video_ids =␣
↪create_transcript_text_files(video_ids,output_directory,language )
```

```
Text Transcript file created for video id: V-Hmc-MRRiU
Skip Counter: 19
```

[20]:
```
video_ids
```

[20]:
```
['V-Hmc-MRRiU',
 'LpVMYXWhErU',
 '3eC2XbN1-Bg',
 'cQsm9JPe794',
 'TZATuiLuTvQ',
 'FXL2aE5uUXc',
 'XQdGShFcDaQ',
 'Q8UhNJw3bWE',
 'TIb0FhBC1QM',
 'PXhD1YREJyo',
 'fs5fqq2YumA',
 'i4ATd-q_YZM',
 'uPqoKpyhzP8',
 'CUM9T5K8lrI',
 'sQgvS2A8Xz8',
 'lDxkKsnB7cg',
 'V3jUiApohGQ',
 'CnrQmSIkP1E',
 'wraRj_ch2rU',
 'lP1FXSktMtg']
```

[21]:
```
irish_video_ids
```

[21]:
```
['V-Hmc-MRRiU']
```

The function described above is designed to extract transcripts and filter video IDs based on the availability of Irish captions/transcripts. It should be noted that among

20 video IDs, only one contains Irish captions.

3.3 Extracting Audio from YT URLS using YT-DLP

[22]:
```python
# ytdlp_path = '/Users/nikhiljindal/opt/anaconda3/envs/myenv/bin/yt-dlp'
# ffmeg_path = '/opt/homebrew/bin/ffmpeg'
# output_directory='/dataset/video_id/'

def download_audio_with_ytdlp(irish_video_ids, ytdlp_path, ffmeg_path,␣
 ↪main_output_directory, output_format='wav', sample_rate=24000):

    for video_id in irish_video_ids:
        # Define output filename format

        output_directory = main_output_directory + video_id

        if not os.path.exists(output_directory):
            os.makedirs(output_directory)

        output_filepath = os.path.join(output_directory, f'{video_id}.
 ↪{output_format}')
        temp_filepath = os.path.join(output_directory, f'temp_{video_id}.
 ↪{output_format}')

        # Check if the audio file already exists to avoid re-downloading
        if not os.path.exists(output_filepath):
            # Run yt-dlp to download the best audio
            yt_dlp_command = [
                ytdlp_path,  # Make sure yt-dlp is installed and in your PATH
                '--extract-audio',
                '--audio-format', output_format,
                '--audio-quality', '0',  # Best quality
                '--output', f'{output_directory}/%(id)s.%(ext)s',
                video_id
            ]
            subprocess.run(yt_dlp_command, stdout=subprocess.DEVNULL,␣
 ↪stderr=subprocess.STDOUT)

            # Convert downloaded audio to desired sample rate and mono channel
            ffmpeg_command = [
                ffmeg_path,
                '-i', output_filepath,  # Input file
                '-ac', '1',  # Mono channel
                '-ar', str(sample_rate),  # Sample rate
                '-y',  # Overwrite output file if it exists
                temp_filepath  # Temporary output file
            ]
            subprocess.run(ffmpeg_command, stdout=subprocess.DEVNULL,␣
↪stderr=subprocess.STDOUT)

            # Move the converted file back to the original file path
            shutil.move(temp_filepath, output_filepath)
            print(f"Audio saved as {output_filepath}")
        else:
            print(f"Audio file already exists: {output_filepath}")
```

[23]:
```
download_audio_with_ytdlp(irish_video_ids, ytdlp_path, ffmpeg_path,␣
↪output_directory)
```

```
Audio saved as Test/V-Hmc-MRRiU/V-Hmc-MRRiU.wav
```

[24]:
```
# THIS WAS THE CODE FOR METHOD 2 (Speech to Text) OF "Extracting Captions/␣
↪Text", THAT DIDN'T WORK OUT

# import os
# from pydub import AudioSegment
# import subprocess

# def download_audio_with_ytdlp(youtube_url, output_format='wav',␣
↪sample_rate=24000):
#     # Define output filename format
#     output_template = 'data/' + '%(id)s.%(ext)s'
#     download_path = os.path.join('.', output_template)

#     # Run yt-dlp to download the best audio
#     command = [
#         '/Users/nikhiljindal/opt/anaconda3/envs/myenv/bin/yt-dlp',
#         '--extract-audio',
#         '--audio-format', output_format,
#         '--output', download_path,
#         youtube_url
#     ]
#     subprocess.run(command, check=True)

#     # Find downloaded file (assuming yt-dlp names files correctly)
#     downloaded_files = [f for f in os.listdir('.') if f.endswith('.wav')]
#     if downloaded_files:
#         downloaded_file = downloaded_files[0]  # Assuming we only downloaded␣
↪one file
#         # Convert audio to desired sample rate and mono channel using ffmpeg
#         output_file = f"{downloaded_file.split('.')[0]}_converted.
↪{output_format}"
#         ffmpeg_command = [
#             'ffmpeg',
#               '-i', downloaded_file,
#               '-ac', '1',  # Set audio channels to 1 (mono)
#               '-ar', str(sample_rate),  # Set sample rate to 24 kHz
#               '-y',  # Overwrite output file if exists
#               output_file
#           ]
#           subprocess.run(ffmpeg_command, check=True)

#           # Remove the original download to save space, if different
#           if downloaded_file != output_file:
#               os.remove(downloaded_file)
#           return output_file
#       else:
#           print("No audio file downloaded.")
#           return None


# for video in videos:
#     output_audio_file = download_audio_with_ytdlp(video)
#     if output_audio_file:
#         print(f"Audio saved as {output_audio_file}")
```

3.4 Data Processing and Formatting

[25]:
```
def create_transcript_json_files(irish_video_ids, main_output_directory,␣
 ↪language):

    for video_id in irish_video_ids:

        output_directory = main_output_directory + video_id

        if not os.path.exists(output_directory):
            os.makedirs(output_directory)

        output_file_path = os.path.join(output_directory, video_id + '.json') ␣
 ↪# Define path for transcript file

        # Check if the transcript file already exists and is not empty
        if os.path.exists(output_file_path) and os.path.
 ↪getsize(output_file_path) > 0:
            print(f"JSON Transcript file already exists and is not empty for␣
 ↪{video_id}")
            continue  # Skip to the next file

        try:
            # Attempt to fetch the transcript in the Irish language
            transcript = YouTubeTranscriptApi.get_transcript(video_id,␣
↪languages=[language])

            formatter = JSONFormatter()

            # .format_transcript(transcript) turns the transcript into a JSON␣
↪string.
            json_formatted = formatter.format_transcript(transcript, indent=2)

            # Write the formatted text content to a file
            with open(output_file_path, 'w', encoding='utf-8') as output_file:
                output_file.write(json_formatted)
            print(f"Text Transcript file created for video id: {video_id}")

        except Exception as e:
            print(f"An error occurred for video_id {video_id}: {e}")
            # Skip file if there's an error (you can decide to delete the audio␣
↪file or keep it for manual review)
```

[26]:
```
def create_text_files(main_directory, speaker_id='0000'):
    # Iterate over all subdirectories in the main directory
    for subdir in next(os.walk(main_directory))[1]:
        video_id = subdir
        json_path = Path(main_directory) / video_id / f"{video_id}.json"
        output_dir = Path(main_directory) / video_id / 'segments'

        # Ensure the output directory exists
        output_dir.mkdir(parents=True, exist_ok=True)

        # Load the transcript data from the JSON file if it exists
        if json_path.exists():
            with open(json_path, 'r', encoding='utf-8') as file:
                transcripts = json.load(file)

            # Iterate over the transcript entries and create a text file for␣
↪each
            for entry in transcripts:
                # Construct the filename using the speaker ID and start time as␣
↪unique identifier
                start_time_str = "{:05d}".format(int(entry['start'] * 1000))  #␣
↪Convert start time to milliseconds and format
                filename = f"{speaker_id}_{video_id}_{start_time_str}.txt"
                file_path = output_dir / filename

                # Write the transcript text to the file
                with open(file_path, 'w', encoding='utf-8') as text_file:
                    text_file.write(entry['text'])
            print(f"Text files created for video ID: {video_id}")
        else:
            print(f"No JSON found for video ID: {video_id}")
```

[27]:
```
print("Extracting JSON transcripts...")
create_transcript_json_files(irish_video_ids, output_directory, language)

print("Preparing Segments...")
create_text_files(output_directory)
```

```
Extracting JSON transcripts…
Text Transcript file created for video id: V-Hmc-MRRiU
Preparing Segments…
Text files created for video ID: V-Hmc-MRRiU
```