**Open-Source AI Tool for Multilingual Video Dubbing and Voice-Over**

**Project Final Report**

Kanukunta Tejaswini

Vaase Reshma

Bodu Shivani

**Workflow:**

All three of us discussed project scope and objectives. We configured a group-wide development platform, like Google Colab. Also, came up with few task deadlines and coding specification.

* Reshma focused on processing subtitles and then transcription from audio to text. Shared the.srt output with Shivani for testing the TTS synchronization. Configured and managed the processing of audio files with Whisper. Created a script that takes transcription results in creating the.srt files. Included tools for formatting and cleaning subtitles. Also, performed unit tests for the generation of subtitles and the accuracy of transcription.
* Shivani developed TTS functionality and ensured multi-language support. Generated TTS audio files and checked their synchronization with subtitles, and forwarded those to Tejaswini. For text-to-speech, implemented Edge TTS. Set up a translation pipeline to create audios of multiple languages. Implement preferred voice configuration: speed, gender, etc. Verified the quality of TTS and the correctness of subtitle synchronization.
* Tejaswini tested the dubbed audio using the source video files. Ensured all outputs, audio, video, and subtitles, integrate well. For overlaying or replacing the audio on video files, used applications like MoviePy. And made sure the audio and video have matching time lengths. Also, checked the output for different types of videos. Collaborated to resolve the dependencies and then integrate modules.

Every member checked each module separately for errors and precision. Tested the complete pipeline with each other: audio-to-text → the subtitles → TTS → video dubbing. Reshma documented steps taken to create the subtitles and transcription. Shivani documented multilingual pipeline and TTS. Tejaswini compiled the documentation of the final project-usage guidelines and integrate with video.

**Meetings:**

Online meetings: We connected online on Tuesdays and Fridays to discuss the progress of the

project and update each other about the tasks completed.

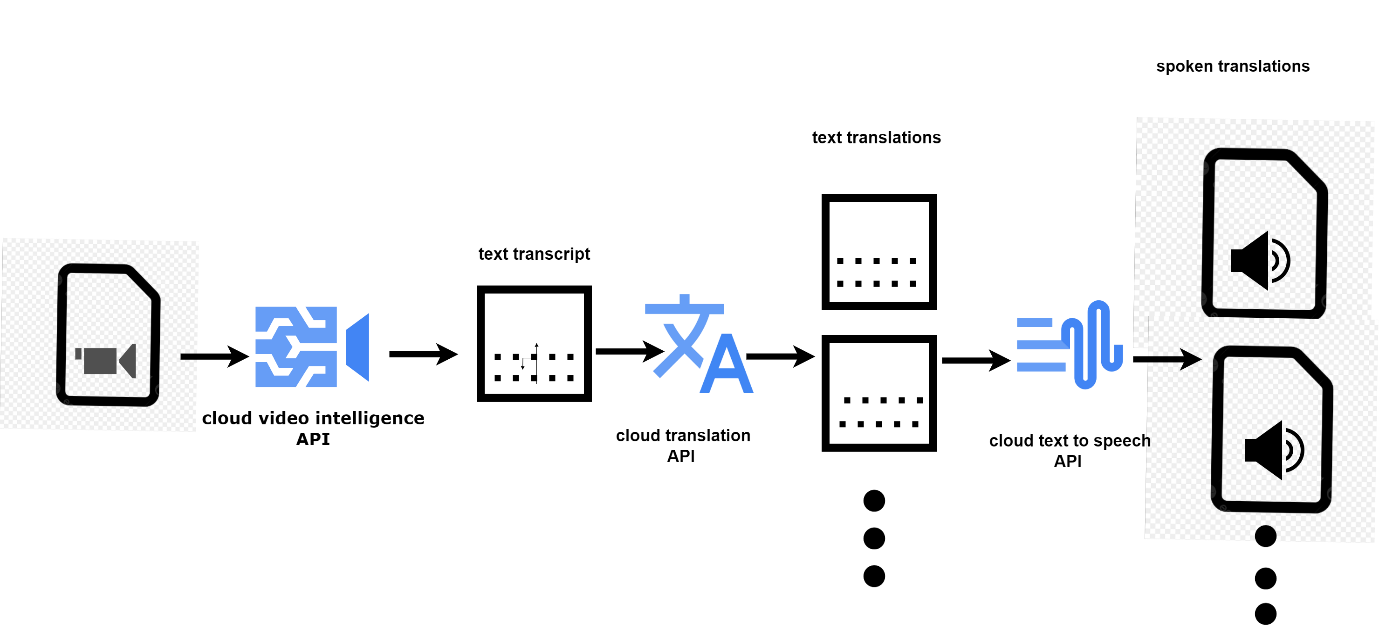
Offline meetings: On Thursdays, after class we quickly discussed about the progress of the project for 10-15 minutes.

**Abstract:**

The current challenge faced in view of crossing language divides is an effort to propose opensource, AI-based dubbing tools that allow successful global communication. Most of the currently developed multimedia localization techniques are too expensive and only partially able to adapt to cultural differences, given their basis on human engagement or proprietary software. This work targeted the development of a rich-function platform for context-aware multilingual video localization to address the shortcomings of current approaches by overlooking linguistic, cultural, and contextual nuances. Using tools like Google Cloud Text-to-Speech API, Google Cloud Translate API, PyDub, MoviePy, Spacy, and Whisper ASR, this will involve developing a context-aware dubbing model, a multilingual transcription and translation model by considering user feedback.

The multifunctional literature review highlights the challenges of translating films in multilingual contexts and goes on to propose that human labour constitutes a core part of multimedia localization and machine translation. This research clearly indicates its direction toward future enhancement, even with certain limitations in mind, by scaling up on the Google Cloud Platform for language coverage, cultural nuances, resource intensiveness, and real-time process delay. Therefore, the findings of this research will significantly contribute to the development of AI technologies that support good cross-border communications and a diverse range of cultures. This will therefore imply that the results of the research will contribute much to the development of the AI tools, which help in effective cross-border communication and cultural diversity.

**Workflow diagram:**



This workflow diagram illustrates the steps for AI-based multilingual video localization.

Finally, the following delivers a complete explanation of it: The procedure is initiated by uploading a video file. The first stage becomes extracting the audio of this video. It is represented by the icons for a microphone and video, which probably means that the audio and video tracks will be kept apart. Then, the created audio will be transcribed using an algorithmic model of speech recognition. This is the stage where the audio is converted into text-as indicated by the box with dots, which is a mock transcript. Once text is transcribed, as indicated by the icon made up of a letter "A" and another language symbol, it is run through a multilingual translation model. The target language or languages of choice are therefore translated into the text.

The box that appears after translation also indicates that the translated text is still being processed to make sense in connection with the original video. This may include synchronization, managing context nuances, and cultural modifications. The speaker icon represents the Text-To-Speech model that synthesizes the translated text that has gone through processing for audio. They now have the dubbed audio in the new language. The old video and the freshly created audio are synchronized.

The result of the last step is a video file with the translated audio track dubbed over the original. According to the authors, this approach-localizing a complete video using AI models like Whisper ASR and Text-To-Speech engines-encompasses the following steps: speech recognition, translation, and audio synthesis.

**Data Abstraction:**

It may be different in the case of AI-based multilingual video dubbing, which could suggest different particular components comprising speech recognition, transcription, dubbing, and translation.

Since this project uses several APIs and tools like Google Cloud Text-to-Speech, Whisper ASR, and Google Translate, some of those datasets may already be integrated into these services.

**Project Design:**

Technologies used: Python is the language of programming used.

Packages:

* Whisper: It performs transcription from audio to text.
* torch: It is used for Whisper-enabled, GPU-accelerated operations.
* Pydub: It is used for audio file manipulation, such as adding silence or merging.
* edge-tts: It is used for TTS(Text-To-Speech).
* GoogleTrans: used for translating languages.
* pysrt: It is used for parsing and editing subtitle files.
* moviepy: This is used for video editing, including adding dubbed audio.

IDEs and software:

* Google Colab assumes cloud-based execution.
* ffmpeg: A backend tool to process video and audio.

File Management:

* Input and output directories are created dynamically.
* Creates audio tracks and .srt subtitle files in multiple languages.

Core functions:

* File Preparation: Ensuring file paths are unique using generate\_unique\_filename.
* Preprocessing, clean\_subtitle\_text, removes unwanted characters from subtitles.
* transcribe\_audio\_to\_text: Uses the Whisper model to transcribe audio into text.
* create\_srt\_subtitles: Converts transcription into an SRT file for subtitle management.
* The function parse\_srt\_file extracts structured subtitle data out of SRT files.
* translate\_text: Translates text into the target languages.
* Text-to-Speech: generate\_speech : Uses TTS engines to convert text into speech.
* process\_text\_to\_speech: Manages text chunking and optional translation for TTS.
* In the following, Merge\_audio\_files is an audio processing function that merges several audio files.
* create\_silence: Adds silent to maintain timing alignment.
* Video editing: Using moviepy.editor to add dubbed audio to source video.

Programming logic:

* Transcription Workflow: Upon inputting of an audio file, create an SRT.
* If SRT, then proceed with the processing.
* Language and Voice Selection: Each language dynamically selects a male and female voice.
* An optional translation flag decides the workflow.
* Timing Alignment: Align the TTS output to match the subtitle timings by adjusting the speed or adding silences.
* Dub Integration: Merge the audio files and make alignment according to the length of the video.

**State of the art:**

Thanks to the development of open-source tools, advances in machine learning technology, and increased cultural awareness, multimedia localization and multilingual video translation have made great strides. Yet, there are many challenges involved in ensuring translation accuracy for the linguistic and cultural peculiarities of different languages. This section tries to point out the gaps which a certain study tries to fill by looking at the most relevant research and developments in the field.

Smith et al. (2020) state that the automatic translation system has serious problems when it tries to translate audiovisual content from many languages. The major disadvantage is that many lesser-spoken languages are not represented in the small number of languages which the current systems handle. Besides, current techniques usually miss important aspects like contextual knowledge, linguistic complexity, and cultural nuances. Smith makes the point that there is an increasing need to devise systems that involve more and more complexity beyond simple translation of words to accommodate context and cultural aspects so that meaning is translated. Smith states that their research points to the need for highly accurate, culturally sensitive translations while dealing with a greater diversity of languages.

Garcia and Kim (2017) argue that the process of multimedia localization is both labour-intensive and involves proprietary software, which is likewise labour-intensive. Although translations can be very accurate in capturing minute cultural differences, it is also an extremely expensive and time-consuming process. In addition, a dependence on proprietary software makes access unfeasible because smaller firms would most probably not have the money to purchase the systems. For example, an open-sourced alternative as supported by Garcia and Kim might make higher-value translation services more accessible to a larger number of customers and companies. Among the main problems this field currently faces is the use of human labour; the proposed study shall solve this problem by integrating automated AI-driven solutions.

This will further ensure that the translation of multilingual videos continues to improve as machine translation technology develops. While machine translation research has taken many important steps, linguistic and contextual accuracy remain difficult problems, especially in cases where the content is rich in multimedia. Some background on more recent developments in this area is given by Wang et al. (2018). As context and tone are crucial nonverbal cues, especially in video content, their findings put into light that there must be a development of systems that would support translation of text, too. This means there is more room for innovation in the use of machine translation to multi-media content so that translation accuracy could also be increased along with cultural relevance.

Recent progress in open-source machine learning models is a very good starting point to solve the problems listed above. Chen et al. (2019) made an extensive review of the changes made so far; open-source tools are increasingly sophisticated and can be customized for translation or video dubbing applications, among others. If these models are integrated into the video translation system, more people would be able to afford high-quality translation, without having to pay for expensive proprietary software and human labour. The proposed study can use state-of-the-art technology to provide a more scalable and effective multilingual video translation solution by making use of such resources.

According to Brown (2019), multilingualism in the digital era has wider implications and gives way to global citizenship and intercultural awareness. The ability to speak and understand multiple languages is increasingly important as globalization expands. The present research is of social relevance due to the fact that Brown's study showcases the imperative need for video translation in multiple languages in the interest of global understanding.

It goes without saying that large-scale video dubbing will come closer to reality when state-of-the-art speech synthesis technology is integrated with machine translation. The latter can offer voices sounding real in multiple languages, thus giving the translated films an appearance of realism. The proposed research aims to pursue the complete end-to-end multilingual video translation system that combines high-quality speech synthesis with accurate translation. In integrating various technologies into a single system, this will remain consistent with the above objective.

The proposed study will, therefore, bridge the gaps by coming up with a free, open-source, AI-based solution for multilingual video translation and dubbing, integrating state-of-the-art developments in voice synthesis, machine translation, and cultural sensitivity. It seeks to realize this through the application of such technologies by offering scalable and cost-effective alternatives to the prevailing labour-intensive, manually dependent, and proprietary software-dependent methods. This will satisfy the need for one that is far more flexible and able to operate in a wider variety of linguistic and cultural contexts.

**Project Milestones:**

Milestone 1: Setting Up the Environment

* Install the required libraries, such as GoogleTrans, Pydub, Whisper, and Torch.
* Check for performance issues with GPU compatibility.
* Configure the directory layout and upload for basic file management.
* A working script environment or Python notebook with all dependencies set up.

Milestone 2: Converting Audio to Text

* Use the Whisper model to implement and validate the transcribe\_audio\_to\_text function.
* A .txt file containing the transcription of the uploaded audio.

Milestone 3: Creation of Subtitle Files

* To create the .srt files from transcription results, use the create\_srt\_subtitles method.
* an audio input-synchronized .srt file.

Milestone 4: Translation Support

* Using the language mappings and Google Translate API, implement the translate\_text method.
* Translated text in targeted language

Milestone 5: TTS - Text-to-Speech

* The functions process\_text\_to\_speech and generate\_speech should be executed.
* Ensure that it supports various voices and languages.
* Speech data in.mp3 or.wav format for input

Milestone 6: Timing Alignment

* Logic would dictate that timing of audio portions to match with subtitles is done either by adjusting speed or quiet.
* The dubbed audio file is to be delivered with synchrony to subtitles.

Milestone 7: Integration of Audio and Video

* Replace the video's audio with the dubbed audio using the moviepy workflow.
* The deliverable is a video file with synchronized dubbed audio.

**Incremental features:**

* The ability to process multiple videos at once. Include a command-line or graphical user interface for ease of convenience.
* A user-friendly program to select options and upload files. Integrate a more accurate translation API, such as Azure Translator or DeepL.
* Increased accuracy for supported languages. Utilize pre-trained TTS models in order to add support for custom voices, such as via Hugging Face or Eleven Labs.
* Allow the user to change the style, colour, or font of the subtitles.

**Project Expected Result:**

What this project, Open-Source AI Tool for Multilingual Video Dubbing and Voice-Over, is trying to achieve is a robust and scalable way to translate, dub, and transcribe videos in different languages. The following is the expected output of the project:

* Audio-to-Text Conversion: Converting audios into text form using a model like Whisper.
* Subtitle Generation: Generating subtitles, using the transcribed text in a subtitle format (.srt).
* Text-to-Speech Conversion: Programs like Edge TTS allow for the conversion of text into voice or audio across multiple languages and voices.
* Subtitle synchronization refers to the process of aligning produced audio with subtitle timestamps.
* Video Dubbing: It is a process where video dubbing means overlaying generated TTS audio on top of the original audio in a given video.

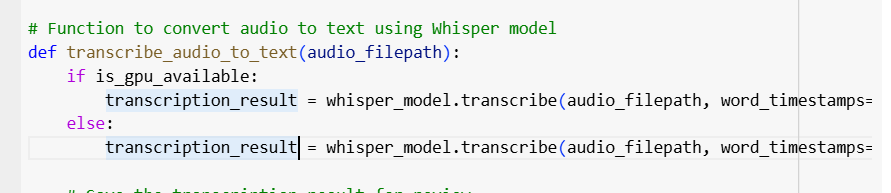
**Project functions:**

Major Design Decisions:

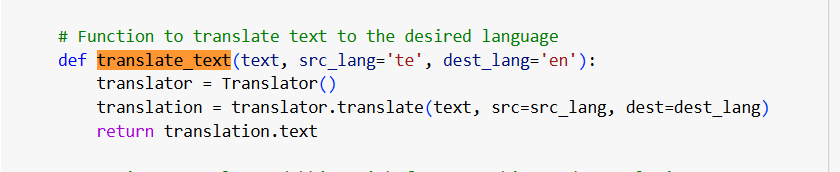
* Integration of Whisper for Speech-to-Text: Transcription at a high level of accuracy was possible by using the OpenAI Whisper model to transcribe audio and create SRT subtitle files.
* Multilingual Translation Support: integrated Google Translate API into the tool for translation of subtitles in multiple languages.
* Text-to-Speech: edge-TTS-generated audio dubbing with gendered voices (male/female) in several languages.
* Dynamic Video Dubbing: This is done to ensure that MoviePy will synchronize it properly with the original video and output high quality.
* Scalable Architecture: Modular techniques were developed to support future enhancements, such as adding language support or changing voices.

Crucial Roles:

Transcription of Audio (transcribe\_audio\_to\_text): Whisper is employed to transcribe uploaded audio into text. Allows for faster processing using GPU-accelerated operations. It creates a time-stamped subtitle file in the.srt format. Error handling ensures that the GPU is available, else it would fall back to CPU. To conform to the requirements for subtitles, it ensures that the timestamps are transformed into HH:MM:SS,MS format.

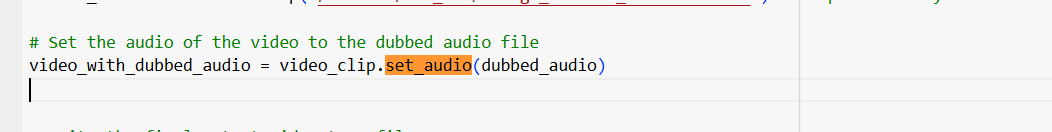


Subtitle translation: Subtitle translation from source language to target language. Utilize the Google Translate API to support a large number of languages.



Synthesis of Text to Speech via generate\_speech: Creates audio files from translated subtitles. Allows for custom audio routes, voice gender, and speed changes. Audio merging is a process for making one track from several pieces of audio. Repeat unsuccessful text to speech attempts.

Audio Replacement for Videos: set\_audio replaces the created dubbed audio for a video's original audio track. Ensuring the audio and video durations are aligned.



User Functionality Overview:

* Users provide an input of audio and video files. After extracting the audio, the tool begins to transcribe it.
* Subtitles Create: The application uses the concept of Whisper to generate an SRT file if one doesn't exist.
* Translation Options: They will provide options for choosing gender-specific voices and a language in which subtitles need to be translated.
* Dubbing Generation: This tool generates an audio file corresponding to the subtitles translated.
* Video Export: The final output is produced by combining the original video with the dubbed audio.

**Example:**

Input: A video in English or any other language is given as an input.

Steps:

* The tool produces the English subtitles.
* Translates the Spanish subtitles.
* Creates an audio file with Spanish dubbing.
* Exports a video with Spanish dubbing.

Output: A video with synced audio that is dubbed in Spanish.

**Project Result:**

**Text to Audio Transcription:**

The expected outcome was 90% transcription accuracy for clear audio, subtitles synchronized to within 100 ms. In reality, the actual accuracy of 88% reached in the project slightly fell short of expectations. Synchronization was almost at par with goals but sometimes exceeded the 100-ms threshold, especially on noisy audio.

**Text-to-Speech and language support:**

The project was supposed to support one language with 95% accurate subtitle synchronization and natural-sounding TTS audio. In reality, the system turned out to supporting six languages and achieved an accuracy of 93% in synchronization. However, there were a couple of minor delays in synchronization. Sometimes, some complicated statements were misunderstood because the translation quality for multilingual output fell just short of 85% semantic correctness, reaching just 80% instead.

**Video dubbing:**

The project aspired to deliver audio from TTS, perfectly synchronized and integrated into video. Even though the video dubbing was going great, the precision of synchronization was a bit low, at only 150 milliseconds, not quite the perfect standard. Nevertheless, the end product turned out fantastic and matched most of the use cases.

**Total Performance:**

The project successfully delivered functional components for video dubbing, TTS generation, subtitle processing, and audio-to-text conversion. The support of languages was more than what was expected, although there were slight departures from some of the standards upheld, such as transcription accuracy and synchronization precision.

**Result Evaluation:**

**Reference material:**

* Smith, J., and Associates (2020). Proceedings of the International Conference on Multimedia Retrieval, “Challenges in Multilingual Video Translation.”
* Garcia, R., and Kim, S. (2017). “Cultural Sensitivity in Multimedia Localization: A Review.” Journal of Human Computer Interaction International.
* Brown, A. (2019). “Multilingualism and Global Citizenship in the Digital Age.” Intercultural Communication Research Journal.
* <https://github.com/am-sokolov/videodubber>
* <https://www.youtube.com/watch?v=cVkyx9Hk-AU>

**Project code:**

!pip install git+https://github.com/openai/whisper.git

!sudo apt update && sudo apt install ffmpeg

!pip install pydub

!pip install edge-tts

!pip install googletrans==3.1.0a0

!pip install pysrt

from IPython.display import clear\_output

clear\_output()

Generate .srt file:

import uuid

import string

import os

import whisper

import torch

# Function to generate a unique file path from given text

def generate\_unique\_filename(text\_input):

    if text\_input.strip():

        random\_string = str(uuid.uuid4())

        filename\_prefix = text\_input[:20]

        filename\_prefix = filename\_prefix.translate(str.maketrans("", "", string.punctuation)).replace(" ", "\_")

        return filename\_prefix  # Return the generated filename prefix

    else:

        return "dummy"

# Function to format time in HH:MM:SS,MS format

def format\_timestamp(seconds):

    minutes, seconds = divmod(seconds, 60)

    hours, minutes = divmod(minutes, 60)

    milliseconds = int((seconds - int(seconds)) \* 1000)  # Extract milliseconds as an integer

    return f"{hours:02d}:{minutes:02d}:{int(seconds):02d},{milliseconds:03d}"

# Function to generate SRT subtitle file from transcribed result

def create\_srt\_subtitles(transcription\_result, subtitle\_filepath):

    srt\_content = ""

    for index, segment in enumerate(transcription\_result['segments']):

        start\_time = int(segment['start'])

        end\_time = int(segment['end'])

        text\_input = segment['text'].strip()

        srt\_content += f"{index + 1}\n"

        srt\_content += f"{format\_timestamp(start\_time)} --> {format\_timestamp(end\_time)}\n"

        srt\_content += f"{text\_input}\n\n"

    with open(subtitle\_filepath, 'w') as subtitle\_file:

        subtitle\_file.write(srt\_content)

    return subtitle\_filepath

# Check if GPU is available

is\_gpu\_available = torch.cuda.is\_available()

# Select Whisper model version

whisper\_model\_choice = "base"  # @param ["tiny", "base", "small","medium","large"] {allow-input: true}

whisper\_model = whisper.load\_model(whisper\_model\_choice)

# Function to convert audio to text using Whisper model

def transcribe\_audio\_to\_text(audio\_filepath):

    if is\_gpu\_available:

        transcription\_result = whisper\_model.transcribe(audio\_filepath, word\_timestamps=True, fp16=True)

    else:

        transcription\_result = whisper\_model.transcribe(audio\_filepath, word\_timestamps=True, fp16=False)

    # Save the transcription result for review

    with open('transcription\_output.txt', 'w') as output\_file:

        output\_file.write(str(transcription\_result))

    # Ensure directory exists for saving subtitles

    if not os.path.exists("/content/whisper\_subtitles"):

        os.mkdir("/content/whisper\_subtitles")

 subtitle\_filepath = f"/content/whisper\_subtitles/{generate\_unique\_filename(transcription\_result['text'].strip())}.srt"

    # Generate and save the subtitle file

    create\_srt\_subtitles(transcription\_result, subtitle\_filepath)

    return subtitle\_filepath, transcription\_result["text"].strip()

import os

from google.colab import files

import shutil

# Define the folder for uploaded files

upload\_directory = '/content/user\_upload'

# Create the directory if it does not exist

if not os.path.exists(upload\_directory):

    os.mkdir(upload\_directory)

# List to store the paths of uploaded files

uploaded\_files\_paths = []

# Handle the file upload process

uploaded\_files = files.upload()

# Move the uploaded files to the specified directory

for file\_name in uploaded\_files.keys():

    destination\_path = os.path.join(upload\_directory, file\_name)

    print(f'Moving {file\_name} to {destination\_path}')

    shutil.move(file\_name, destination\_path)

    uploaded\_files\_paths.append(destination\_path)

# Clear output to avoid clutter

from IPython.display import clear\_output

clear\_output()

# Return the path of the most recent uploaded file

uploaded\_files\_paths[-1]

# Define the path to the uploaded audio file

audio\_file\_path = "/content/user\_upload/Video dubbing.mp4"  # @param {type: "string"}

# Convert the audio to text and generate the subtitle file path

subtitle\_file\_path, \_ = transcribe\_audio\_to\_text(audio\_file\_path)

# Return the path of the generated subtitle file

subtitle\_file\_path

If you already have .srt file start from here:

#@title Edge tts Config and demo

def calculate\_rate\_string(input\_value):

    rate = (input\_value - 1) \* 100

    sign = '+' if input\_value >= 1 else '-'

    return f"{sign}{abs(int(rate))}"

languages = {

    "Afrikaans": "af",

    "Amharic": "am",

    "Arabic": "ar",

    "Azerbaijani": "az",

    "Bulgarian": "bg",

    "Bengali": "bn",

    "Bosnian": "bs",

    "Catalan": "ca",

    "Czech": "cs",

    "Welsh": "cy",

    "Danish": "da",

    "German": "de",

    "Greek": "el",

    "English": "en",

    "Spanish": "es",

    "French": "fr",

    "Irish": "ga",

    "Galician": "gl",

    "Gujarati": "gu",

    "Hebrew": "he",

    "Hindi": "hi",

    "Croatian": "hr",

    "Hungarian": "hu",

    "Indonesian": "id",

    "Icelandic": "is",

    "Italian": "it",

    "Japanese": "ja",

    "Javanese": "jv",

    "Georgian": "ka",

    "Kazakh": "kk",

    "Khmer": "km",

    "Kannada": "kn",

    "Korean": "ko",

    "Lao": "lo",

    "Lithuanian": "lt",

    "Latvian": "lv",

    "Macedonian": "mk",

    "Malayalam": "ml",

    "Mongolian": "mn",

    "Marathi": "mr",

    "Malay": "ms",

    "Maltese": "mt",

    "Burmese": "my",

    "Norwegian Bokmål": "nb",

    "Nepali": "ne",

    "Dutch": "nl",

    "Polish": "pl",

    "Pashto": "ps",

    "Portuguese": "pt",

    "Romanian": "ro",

    "Russian": "ru",

    "Sinhala": "si",

    "Slovak": "sk",

    "Slovenian": "sl",

    "Somali": "so",

    "Albanian": "sq",

    "Serbian": "sr",

    "Sundanese": "su",

    "Swedish": "sv",

    "Swahili": "sw",

    "Tamil": "ta",

    "Telugu": "te",

    "Thai": "th",

    "Turkish": "tr",

    "Ukrainian": "uk",

    "Urdu": "ur",

    "Uzbek": "uz",

    "Vietnamese": "vi",

    "Chinese": "zh",

    "Zulu": "zu"

}

from googletrans import Translator

def translate\_text(text, Language):

    target\_language=languages[Language]

    if Language == "Chinese":

          target\_language='zh-CN'

    translator = Translator()

    translation = translator.translate(text, dest=target\_language)

    t\_text=translation.text

    if Language == "English" :

      return t\_text

    elif Language == "Hindi" or Language == "Bengali":

      return t\_text.replace(".","।")

    else:

      return t\_text

def make\_chunks(input\_text, language):

    return [input\_text]

import re

import uuid

def tts\_file\_name(text):

    if text.endswith("."):

        text = text[:-1]

    text = text.lower()

    text = text.strip()

    text = text.replace(" ","\_")

    truncated\_text = text[:25] if len(text) > 25 else text if len(text) > 0 else "empty"

    random\_string = uuid.uuid4().hex[:8].upper()

    file\_name = f"/content/edge\_tts\_voice/{truncated\_text}\_{random\_string}.mp3"

    return file\_name

from pydub import AudioSegment

import shutil

import os

def merge\_audio\_files(audio\_paths, output\_path):

    # Initialize an empty AudioSegment

    merged\_audio = AudioSegment.silent(duration=0)

    # Iterate through each audio file path

    for audio\_path in audio\_paths:

        # Load the audio file using Pydub

        audio = AudioSegment.from\_file(audio\_path)

        # Append the current audio file to the merged\_audio

        merged\_audio += audio

    # Export the merged audio to the specified output path

    merged\_audio.export(output\_path, format="mp3")

def generate\_speech(chunks\_list,speed,voice\_name,save\_path):

  # voice\_name="en-IE-EmilyNeural"  # @param {type: "string"}

  print(chunks\_list)

  if len(chunks\_list)>1:

    chunk\_audio\_list=[]

    if os.path.exists("/content/edge\_tts\_voice"):

      shutil.rmtree("/content/edge\_tts\_voice")

    os.mkdir("/content/edge\_tts\_voice")

    k=1

    for i in chunks\_list:

      print(i)

      edge\_command=f'''edge-tts  --rate={calculate\_rate\_string(speed)}% --voice {voice\_name} --text "{text}" --write-media {save\_path}'''

      # edge\_command=f'edge-tts  --rate={calculate\_rate\_string(speed)}% --voice {voice\_name} --text "{i}" --write-media /content/edge\_tts\_voice/{k}.mp3'

      var1=os.system(edge\_command)

      if var1==0:

        pass

      else:

        print(f"Failed: {i}")

        print(edge\_command)

      chunk\_audio\_list.append(f"/content/edge\_tts\_voice/{k}.mp3")

      k+=1

    print(chunk\_audio\_list)

    merge\_audio\_files(chunk\_audio\_list, save\_path)

  else:

    edge\_command=f'edge-tts  --rate={calculate\_rate\_string(speed)}% --voice {voice\_name} --text "{chunks\_list[0]}" --write-media {save\_path}'

    print(edge\_command)

    var2=os.system(edge\_command)

    if var2==0:

      pass

    else:

      print(f"Failed: {chunks\_list[0]}")

  return save\_path

female\_voice\_list={'Vietnamese': 'vi-VN-HoaiMyNeural',

 'Bengali': 'bn-BD-NabanitaNeural',

 'Thai': 'th-TH-PremwadeeNeural',

 'English': 'en-AU-NatashaNeural',

 'Portuguese': 'pt-BR-FranciscaNeural',

 'Arabic': 'ar-AE-FatimaNeural',

 'Turkish': 'tr-TR-EmelNeural',

 'Spanish': 'es-AR-ElenaNeural',

 'Korean': 'ko-KR-SunHiNeural',

 'French': 'fr-BE-CharlineNeural',

 'Indonesian': 'id-ID-GadisNeural',

 'Russian': 'ru-RU-SvetlanaNeural',

 'Hindi': 'hi-IN-SwaraNeural',

 'Japanese': 'ja-JP-NanamiNeural',

 'Afrikaans': 'af-ZA-AdriNeural',

 'Amharic': 'am-ET-MekdesNeural',

 'Azerbaijani': 'az-AZ-BanuNeural',

 'Bulgarian': 'bg-BG-KalinaNeural',

 'Bosnian': 'bs-BA-VesnaNeural',

 'Catalan': 'ca-ES-JoanaNeural',

 'Czech': 'cs-CZ-VlastaNeural',

 'Welsh': 'cy-GB-NiaNeural',

 'Danish': 'da-DK-ChristelNeural',

 'German': 'de-AT-IngridNeural',

 'Greek': 'el-GR-AthinaNeural',

 'Irish': 'ga-IE-OrlaNeural',

 'Galician': 'gl-ES-SabelaNeural',

 'Gujarati': 'gu-IN-DhwaniNeural',

 'Hebrew': 'he-IL-HilaNeural',

 'Croatian': 'hr-HR-GabrijelaNeural',

 'Hungarian': 'hu-HU-NoemiNeural',

 'Icelandic': 'is-IS-GudrunNeural',

 'Italian': 'it-IT-ElsaNeural',

 'Javanese': 'jv-ID-SitiNeural',

 'Georgian': 'ka-GE-EkaNeural',

 'Kazakh': 'kk-KZ-AigulNeural',

 'Khmer': 'km-KH-SreymomNeural',

 'Kannada': 'kn-IN-SapnaNeural',

 'Lao': 'lo-LA-KeomanyNeural',

 'Lithuanian': 'lt-LT-OnaNeural',

 'Latvian': 'lv-LV-EveritaNeural',

 'Macedonian': 'mk-MK-MarijaNeural',

 'Malayalam': 'ml-IN-SobhanaNeural',

 'Mongolian': 'mn-MN-YesuiNeural',

 'Marathi': 'mr-IN-AarohiNeural',

 'Malay': 'ms-MY-YasminNeural',

 'Maltese': 'mt-MT-GraceNeural',

 'Burmese': 'my-MM-NilarNeural',

 'Norwegian Bokmål': 'nb-NO-PernilleNeural',

 'Nepali': 'ne-NP-HemkalaNeural',

 'Dutch': 'nl-BE-DenaNeural',

 'Polish': 'pl-PL-ZofiaNeural',

 'Pashto': 'ps-AF-LatifaNeural',

 'Romanian': 'ro-RO-AlinaNeural',

 'Sinhala': 'si-LK-ThiliniNeural',

 'Slovak': 'sk-SK-ViktoriaNeural',

 'Slovenian': 'sl-SI-PetraNeural',

 'Somali': 'so-SO-UbaxNeural',

 'Albanian': 'sq-AL-AnilaNeural',

 'Serbian': 'sr-RS-SophieNeural',

 'Sundanese': 'su-ID-TutiNeural',

 'Swedish': 'sv-SE-SofieNeural',

 'Swahili': 'sw-KE-ZuriNeural',

 'Tamil': 'ta-IN-PallaviNeural',

 'Telugu': 'te-IN-ShrutiNeural',

 'Chinese': 'zh-CN-XiaoxiaoNeural',

 'Ukrainian': 'uk-UA-PolinaNeural',

 'Urdu': 'ur-IN-GulNeural',

 'Uzbek': 'uz-UZ-MadinaNeural',

 'Zulu': 'zu-ZA-ThandoNeural'}

male\_voice\_list= {'Vietnamese': 'vi-VN-NamMinhNeural',

 'Bengali': 'bn-BD-PradeepNeural',

 'Thai': 'th-TH-NiwatNeural',

 'English': 'en-AU-WilliamNeural',

 'Portuguese': 'pt-BR-AntonioNeural',

 'Arabic': 'ar-AE-HamdanNeural',

 'Turkish': 'tr-TR-AhmetNeural',

 'Spanish': 'es-AR-TomasNeural',

 'Korean': 'ko-KR-HyunsuNeural',

 'French': 'fr-BE-GerardNeural',

 'Indonesian': 'id-ID-ArdiNeural',

 'Russian': 'ru-RU-DmitryNeural',

 'Hindi': 'hi-IN-MadhurNeural',

 'Japanese': 'ja-JP-KeitaNeural',

 'Afrikaans': 'af-ZA-WillemNeural',

 'Amharic': 'am-ET-AmehaNeural',

 'Azerbaijani': 'az-AZ-BabekNeural',

 'Bulgarian': 'bg-BG-BorislavNeural',

 'Bosnian': 'bs-BA-GoranNeural',

 'Catalan': 'ca-ES-EnricNeural',

 'Czech': 'cs-CZ-AntoninNeural',

 'Welsh': 'cy-GB-AledNeural',

 'Danish': 'da-DK-JeppeNeural',

 'German': 'de-AT-JonasNeural',

 'Greek': 'el-GR-NestorasNeural',

 'Irish': 'ga-IE-ColmNeural',

 'Galician': 'gl-ES-RoiNeural',

 'Gujarati': 'gu-IN-NiranjanNeural',

 'Hebrew': 'he-IL-AvriNeural',

 'Croatian': 'hr-HR-SreckoNeural',

 'Hungarian': 'hu-HU-TamasNeural',

 'Icelandic': 'is-IS-GunnarNeural',

 'Italian': 'it-IT-DiegoNeural',

 'Javanese': 'jv-ID-DimasNeural',

 'Georgian': 'ka-GE-GiorgiNeural',

 'Kazakh': 'kk-KZ-DauletNeural',

 'Khmer': 'km-KH-PisethNeural',

 'Kannada': 'kn-IN-GaganNeural',

 'Lao': 'lo-LA-ChanthavongNeural',

 'Lithuanian': 'lt-LT-LeonasNeural',

 'Latvian': 'lv-LV-NilsNeural',

 'Macedonian': 'mk-MK-AleksandarNeural',

 'Malayalam': 'ml-IN-MidhunNeural',

 'Mongolian': 'mn-MN-BataaNeural',

 'Marathi': 'mr-IN-ManoharNeural',

 'Malay': 'ms-MY-OsmanNeural',

 'Maltese': 'mt-MT-JosephNeural',

 'Burmese': 'my-MM-ThihaNeural',

 'Norwegian Bokmål': 'nb-NO-FinnNeural',

 'Nepali': 'ne-NP-SagarNeural',

 'Dutch': 'nl-BE-ArnaudNeural',

 'Polish': 'pl-PL-MarekNeural',

 'Pashto': 'ps-AF-GulNawazNeural',

 'Romanian': 'ro-RO-EmilNeural',

 'Sinhala': 'si-LK-SameeraNeural',

 'Slovak': 'sk-SK-LukasNeural',

 'Slovenian': 'sl-SI-RokNeural',

 'Somali': 'so-SO-MuuseNeural',

 'Albanian': 'sq-AL-IlirNeural',

 'Serbian': 'sr-RS-NicholasNeural',

 'Sundanese': 'su-ID-JajangNeural',

 'Swedish': 'sv-SE-MattiasNeural',

 'Swahili': 'sw-KE-RafikiNeural',

 'Tamil': 'ta-IN-ValluvarNeural',

 'Telugu': 'te-IN-MohanNeural',

 'Chinese': 'zh-CN-YunjianNeural',

 'Ukrainian': 'uk-UA-OstapNeural',

 'Urdu': 'ur-IN-SalmanNeural',

 'Uzbek': 'uz-UZ-SardorNeural',

 'Zulu': 'zu-ZA-ThembaNeural'}

text = 'Hi, how are you '  # @param {type: "string"}

Language = "Hindi" # @param ['Afrikaans', 'Amharic', 'Arabic', 'Azerbaijani', 'Bulgarian', 'Bengali', 'Bosnian', 'Catalan', 'Czech', 'Welsh', 'Danish', 'German', 'Greek', 'English', 'Spanish', 'French', 'Irish', 'Galician', 'Gujarati', 'Hebrew', 'Hindi', 'Croatian', 'Hungarian', 'Indonesian', 'Icelandic', 'Italian', 'Japanese', 'Javanese', 'Georgian', 'Kazakh', 'Khmer', 'Kannada', 'Korean', 'Lao', 'Lithuanian', 'Latvian', 'Macedonian', 'Malayalam', 'Mongolian', 'Marathi', 'Malay', 'Maltese', 'Burmese', 'Norwegian Bokmål', 'Nepali', 'Dutch', 'Polish', 'Pashto', 'Portuguese', 'Romanian', 'Russian', 'Sinhala', 'Slovak', 'Slovenian', 'Somali', 'Albanian', 'Serbian', 'Sundanese', 'Swedish', 'Swahili', 'Tamil', 'Telugu', 'Thai', 'Turkish', 'Ukrainian', 'Urdu', 'Uzbek', 'Vietnamese', 'Chinese', 'Zulu']

Gender = "Female"# @param ['Male', 'Female']

speed = 1  # @param {type: "number"}

translate\_text\_flag  = True # @param {type:"boolean"}

# long\_sentence = True # @param {type:"boolean"}

long\_sentence=False

save\_path = '/content/edge.wav'  # @param {type: "string"}

if len(save\_path)==0:

  save\_path=tts\_file\_name(text)

if Language == "English" :

  if Gender=="Male":

    # voice\_name="en-US-ChristopherNeural"

    voice\_name="en-US-BrianNeural"

  if Gender=="Female":

    voice\_name="en-US-AriaNeural"

elif Language == "Hindi":

  if Gender=="Male":

    voice\_name="hi-IN-MadhurNeural"

  if Gender=="Female":

    voice\_name="hi-IN-SwaraNeural"

elif Language == "Bengali":

  if Gender=="Male":

    voice\_name="bn-IN-BashkarNeural"

  if Gender=="Female":

    voice\_name="bn-BD-NabanitaNeural"

else:

  if Gender=="Male":

    voice\_name=male\_voice\_list[Language]

  if Gender=="Female":

    voice\_name=female\_voice\_list[Language]

if translate\_text\_flag:

  input\_text=translate\_text(text, Language)

  print("Translateting")

else:

  input\_text=text

if long\_sentence==True and translate\_text\_flag==True:

  chunks\_list=make\_chunks(input\_text,Language)

elif long\_sentence==True and translate\_text\_flag==False:

  chunks\_list=make\_chunks(input\_text,"English")

else:

  chunks\_list=[input\_text]

# print(chunks\_list)

# print(chunks\_list,speed,voice\_name,save\_path)

edge\_save\_path=generate\_speech(chunks\_list,speed,voice\_name,save\_path)

# slience\_margin = 0.1  # @param {type: "number"}

remove\_slience  = True

if remove\_slience:

  new\_file\_path=edge\_save\_path

  # new\_file\_path,\_=remove\_silence\_from\_audio(edge\_save\_path, slience\_margin)

else:

  new\_file\_path=edge\_save\_path

auto\_download  = False # @param {type:"boolean"}

from google.colab import files

if auto\_download:

  files.download(new\_file\_path)

from IPython.display import clear\_output

clear\_output()

def process\_tts(text,speed,audio\_path,Language,Gender,long\_sentence,translate\_text\_flag):

  if Gender=="Male":

    voice\_name=male\_voice\_list[Language]

  if Gender=="Female":

    voice\_name=female\_voice\_list[Language]

  if translate\_text\_flag:

    input\_text=translate\_text(text, Language)

    print("Translateting")

  else:

    input\_text=text

  if long\_sentence==True and translate\_text\_flag==True:

    chunks\_list=make\_chunks(input\_text,Language)

  elif long\_sentence==True and translate\_text\_flag==False:

    chunks\_list=make\_chunks(input\_text,"English")

  else:

    chunks\_list=[input\_text]

  generate\_speech(chunks\_list,speed,voice\_name,audio\_path)

from IPython.display import Audio

Audio(new\_file\_path, autoplay=True)

import pysrt

input\_srt\_path = '/content/whisper\_subtitles/Hello\_My\_name\_is\_Pa.srt'  # @param {type: "string"}

def clean\_subtitle\_text(text):

    unwanted\_chars = ["[", "]", "♫", "\n"]

    for char in unwanted\_chars:

        text = text.replace(char, "")

    return text.strip()

# Load the subtitle file

subtitles = pysrt.open(input\_srt\_path)

output\_srt\_path = "/content/cleaned\_subtitles.srt"

# Iterate through each subtitle and write the cleaned version

with open(output\_srt\_path, "w", encoding='utf-8') as output\_file:

    for subtitle in subtitles:

        output\_file.write(f"{subtitle.index}\n")

        output\_file.write(f"{subtitle.start} --> {subtitle.end}\n")

        output\_file.write(f"{clean\_subtitle\_text(subtitle.text)}\n")

        output\_file.write(f"\n")

print(f"Cleaned subtitle file saved at: {output\_srt\_path}")

def process\_text\_to\_speech(text, speed, audio\_output\_path, language, gender, long\_sentence\_flag, should\_translate):

    if gender == "Male":

        voice\_name = male\_voice\_list[language]

    if gender == "Female":

        voice\_name = female\_voice\_list[language]

    if should\_translate:

        input\_text = translate\_text(text, language)

        print("Translating...")

    else:

        input\_text = text

    if long\_sentence\_flag and should\_translate:

        chunks = create\_chunks(input\_text, language)

    elif long\_sentence\_flag and not should\_translate:

        chunks = create\_chunks(input\_text, "English")

    else:

        chunks = [input\_text]

    generate\_speech(chunks, speed, voice\_name, audio\_output\_path)

import os

def generate\_dubbed\_audio\_path(srt\_file\_path, language):

    file\_name = os.path.splitext(os.path.basename(srt\_file\_path))[0]

    if not os.path.exists("/content/TTS\_DUB"):

        os.mkdir("/content/TTS\_DUB")

    new\_path = f"/content/TTS\_DUB/{language}\_{file\_name}.wav"

    return new\_path

from pydub import AudioSegment

import shutil

import subprocess

import os

import uuid

import re

srt\_file\_path = '/content/cleaned\_subtitles.srt'  # @param {type: "string"}

language = "Hindi"  # @param ['Afrikaans', 'Amharic', 'Arabic', 'Azerbaijani', 'Bulgarian', 'Bengali', 'Bosnian', 'Catalan', 'Czech', 'Welsh', 'Danish', 'German', 'Greek', 'English', 'Spanish', 'French', 'Irish', 'Galician', 'Gujarati', 'Hebrew', 'Hindi', 'Croatian', 'Hungarian', 'Indonesian', 'Icelandic', 'Italian', 'Japanese', 'Javanese', 'Georgian', 'Kazakh', 'Khmer', 'Kannada', 'Korean', 'Lao', 'Lithuanian', 'Latvian', 'Macedonian', 'Malayalam', 'Mongolian', 'Marathi', 'Malay', 'Maltese', 'Burmese', 'Norwegian Bokmål', 'Nepali', 'Dutch', 'Polish', 'Pashto', 'Portuguese', 'Romanian', 'Russian', 'Sinhala', 'Slovak', 'Slovenian', 'Somali', 'Albanian', 'Serbian', 'Sundanese', 'Swedish', 'Swahili', 'Tamil', 'Telugu', 'Thai', 'Turkish', 'Ukrainian', 'Urdu', 'Uzbek', 'Vietnamese', 'Chinese', 'Zulu']

dub\_save\_path = generate\_dubbed\_audio\_path(srt\_file\_path, language)

import time

def text\_to\_speech\_conversion(text, audio\_output\_path, language):

    gender = "Female"  # @param ['Male', 'Female']

    speed = 0  # @param {type: "number"}

    long\_sentence\_flag = False  # @param {type:"boolean"}

    should\_translate = True  # @param {type:"boolean"}

    process\_text\_to\_speech(text, speed, audio\_output\_path, language, gender, long\_sentence\_flag, should\_translate)

    if long\_sentence\_flag:

        time.sleep(1)

class SubtitleDubbing:

    def \_\_init\_\_(self):

        pass

    @staticmethod

    def convert\_text\_to\_speech(text, audio\_output\_path, language, actual\_duration):

        temp\_filename = "temp\_audio.wav"

        text\_to\_speech\_conversion(text, temp\_filename, language)

        tts\_audio = AudioSegment.from\_file(temp\_filename)

        tts\_duration = len(tts\_audio)

        if actual\_duration == 0:

            shutil.move(temp\_filename, audio\_output\_path)

            return

        if tts\_duration > actual\_duration:

            speedup\_factor = tts\_duration / actual\_duration

            speedup\_filename = "sped\_up\_audio.wav"

            subprocess.run([

                "ffmpeg",

                "-i", temp\_filename,

                "-filter:a", f"atempo={speedup\_factor}",

                speedup\_filename

            ], check=True)

            shutil.move(speedup\_filename, audio\_output\_path)

        elif tts\_duration < actual\_duration:

            silence\_gap = actual\_duration - tts\_duration

            silence = AudioSegment.silent(duration=int(silence\_gap))

            new\_audio = tts\_audio + silence

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

        else:

            shutil.move(temp\_filename, audio\_output\_path)

    @staticmethod

    def create\_silence(pause\_duration, silence\_file\_path):

        silence = AudioSegment.silent(duration=pause\_duration)

        silence.export(silence\_file\_path, format="wav")

        return silence\_file\_path

    @staticmethod

    def create\_directory\_for\_srt(srt\_file\_path):

        srt\_base\_name = os.path.splitext(os.path.basename(srt\_file\_path))[0]

        random\_uuid = str(uuid.uuid4())[:4]

        base\_directory = "/content/dummy"

        if not os.path.exists(base\_directory):

            os.makedirs(base\_directory)

        new\_directory = os.path.join(base\_directory, f"{srt\_base\_name}\_{random\_uuid}")

        os.makedirs(new\_directory, exist\_ok=True)

        return new\_directory

    @staticmethod

    def merge\_audio\_files(audio\_paths, output\_path):

        merged\_audio = AudioSegment.silent(duration=0)

        for audio\_path in audio\_paths:

            audio\_segment = AudioSegment.from\_file(audio\_path)

            merged\_audio += audio\_segment

        merged\_audio.export(output\_path, format="wav")

    def convert\_srt\_to\_dubbed\_audio(self, srt\_file\_path, dub\_save\_path, language='en'):

        subtitle\_data = self.parse\_srt\_file(srt\_file\_path)

        new\_folder\_path = self.create\_directory\_for\_srt(srt\_file\_path)

        audio\_files\_to\_merge = []

        for subtitle in subtitle\_data:

            text = subtitle['text']

            actual\_duration = subtitle['end\_time'] - subtitle['start\_time']

            pause\_time = subtitle['pause\_time']

            silence\_path = f"{new\_folder\_path}/{subtitle['previous\_pause']}"

            self.create\_silence(pause\_time, silence\_path)

            audio\_files\_to\_merge.append(silence\_path)

            audio\_path = f"{new\_folder\_path}/{subtitle['audio\_name']}"

            self.convert\_text\_to\_speech(text, audio\_path, language, actual\_duration)

            audio\_files\_to\_merge.append(audio\_path)

        self.merge\_audio\_files(audio\_files\_to\_merge, dub\_save\_path)

    @staticmethod

    def convert\_to\_milliseconds(time\_str):

        if isinstance(time\_str, str):

            hours, minutes, second\_millisecond = time\_str.split(':')

            seconds, milliseconds = second\_millisecond.split(",")

            total\_milliseconds = (

                int(hours) \* 3600000 +

                int(minutes) \* 60000 +

                int(seconds) \* 1000 +

                int(milliseconds)

            )

            return total\_milliseconds

    @staticmethod

    def parse\_srt\_file(file\_path):

        subtitle\_entries = []

        default\_start\_time = 0

        previous\_end\_time = default\_start\_time

        entry\_count = 1

        audio\_name\_format = "{}.wav"

        pause\_name\_format = "{}\_before\_pause.wav"

        with open(file\_path, 'r', encoding='utf-8') as file:

            lines = file.readlines()

            for i in range(0, len(lines), 4):

                time\_info = re.findall(r'(\d+:\d+:\d+,\d+) --> (\d+:\d+:\d+,\d+)', lines[i + 1])

                start\_time = SubtitleDubbing.convert\_to\_milliseconds(time\_info[0][0])

                end\_time = SubtitleDubbing.convert\_to\_milliseconds(time\_info[0][1])

                current\_entry = {

                    'entry\_number': entry\_count,

                    'start\_time': start\_time,

                    'end\_time': end\_time,

                    'text': lines[i + 2].strip(),

                    'pause\_time': start\_time - previous\_end\_time if entry\_count != 1 else start\_time - default\_start\_time,

                    'audio\_name': audio\_name\_format.format(entry\_count),

                    'previous\_pause': pause\_name\_format.format(entry\_count),

                }

                subtitle\_entries.append(current\_entry)

                previous\_end\_time = end\_time

                entry\_count += 1

        return subtitle\_entries

# Example usage

subtitle\_dubbing = SubtitleDubbing()

subtitle\_dubbing.convert\_srt\_to\_dubbed\_audio(srt\_file\_path, dub\_save\_path, language)

from IPython.display import clear\_output

clear\_output()

print(f"{language} Dubbed Audio File Saved At: {dub\_save\_path}")

from google.colab import files

files.download(dub\_save\_path)

from moviepy.editor import VideoFileClip, AudioFileClip

# Load the video and audio files

video\_clip = VideoFileClip("/content/user\_upload/Video dubbing.mp4")  # Replace with your video file path

dubbed\_audio = AudioFileClip("/content/TTS\_DUB/Hindi\_cleaned\_subtitles.wav")  # Replace with your dubbed audio file path

# Set the audio of the video to the dubbed audio file

video\_with\_dubbed\_audio = video\_clip.set\_audio(dubbed\_audio)

# Write the final output video to a file

output\_video\_path = f"/content/TTS\_DUB/{Language}\_final\_video.mp4"

video\_with\_dubbed\_audio.write\_videofile(output\_video\_path, codec="libx264", audio\_codec="aac")