Multilingual Dubbing from Subtitle

This is not accurate

Subtitle Dubbing Tool is a convenient utility designed to assist in the process of dubbing subtitles for videos. This tool streamlines the workflow by integrating audio upload, automatic subtitle generation, translation, and text-to-speech (TTS) capabilities.

How to Use:

1. **Upload Audio:**
   - Begin by uploading the audio file corresponding to the video content you wish to dub.
2. **Whisper to Generate Subtitle (.srt) Format:**
   - Utilize the whisper feature to automatically generate subtitles in the .srt format. This feature transcribes the spoken content of the uploaded audio into text, ensuring accurate representation.
3. **Google Translate for Translation:**
   - Translate the generated subtitles into your desired language using Google Translate. This step ensures that your subtitles are accessible to a broader audience by providing translations in multiple languages.
4. **Edge TTS for Multilingual TTS:**
   - Employ Edge TTS (Text-to-Speech) functionality to convert the translated subtitles into speech. This enables the creation of dubbed audio tracks in various languages, enhancing the accessibility and reach of your video content.

Usage Instructions:

- Follow the numbered steps listed above to sequentially navigate through the subtitle dubbing process.
- Ensure that the uploaded audio file is clear and of sufficient quality to facilitate accurate transcription and dubbing.
- Review and refine the generated subtitles and translations as necessary to maintain accuracy and coherence.
- Experiment with different languages and TTS voices offered by Edge TTS to customize the dubbing experience according to your preferences and audience demographics.

Notes:

- Subtitle Dubbing Tool is intended to streamline the dubbing process and enhance the accessibility of video content by providing automated transcription, translation, and text-to-speech functionalities.
- While the tool aims to produce accurate results, it is advisable to review the generated subtitles and translations for any errors or discrepancies.
- Feedback and suggestions for improvement are welcome to continually enhance the functionality and usability of the tool.

Thank you for using Subtitle Dubbing Tool!

Install

```
#@title Install
!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()
```

If you don't have the srt File . First Generate the .srt File.

```
import uuid
import string
import os
import whisper
import torch

```

whisper_model_choice: base

```
# 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.make
        return filename_prefix  # Return the generated filen
    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)  # E
    return f"{hours:02d}:{minutes:02d}:{int(seconds):02d},{m

# Function to generate SRT subtitle file from transcribed re
def create_srt_subtitles(transcription_result, subtitle_file
    srt_content = ""

    for index, segment in enumerate(transcription_result['se
        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)} -->
        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", "sm
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(audi
    else:
        transcription_result = whisper_model.transcribe(audi

    # Save the transcription result for review
    with open('transcription_output.txt', 'w') as output_fil
        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/{genera

    # Generate and save the subtitle file
    create_srt_subtitles(transcription_result, subtitle_file

    return subtitle_filepath, transcription_result["text"].s

# Example usage:
# transcribe_audio_to_text("/content/sample_audio.MP3")

```

```
100%|██████████████████████████████████████| 139M/139M [00:01<00:00, 105MiB/s]
/usr/local/lib/python3.10/dist-packages/whisper/__init__.py:150: FutureWarning: You are using `torch.load` with `weights_onl
  checkpoint = torch.load(fp, map_location=device)
```

```
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]

```

```
'/content/user_upload/@English YouTube shortsshorts shortsindia trendingshorts youtubeshorts viralshorts newshorts.mp4'
```

```
# Define the path to the uploaded audio file
audio_file_path = "/content/user_upload/@English YouTube sho

# Convert the audio to text and generate the subtitle file p
subtitle_file_path, _ = transcribe_audio_to_text(audio_file_

# Return the path of the generated subtitle file
subtitle_file_path

```

audio_file_path: "/content/user_upload/@English YouTı"

```
'/content/whisper_subtitles/This_is_important_W.srt'
```

If you already have the srt . Start from here.

Use DownaSub to generate Subtitle from youtube video

[downsub]

Edge tts Config and demo

```
#@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",
```

text: "Hi, How are you ."

Language: Japanese

Gender: Male

speed: 1

translate_text_flag: ☐

save_path: "/content/edge.wav"

auto_download: ☐

```
    "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 == "English" :
    #   target_language='en'
    # if Language == "Hindi":
    #   target_language='hi'
    # if Language == "Bengali":
    #   target_language='bn'
    if Language == "Chinese":
          target_language='zh-CN'
    translator = Translator()
    translation = translator.translate(text, dest=target_lan
    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]
    # if language == "English":
    #     temp_list = input_text.strip().split(".")
    #     filtered_list = [element.strip() + '.' for element
    #     if temp_list[-1].strip():
    #         filtered_list.append(temp_list[-1].strip())
    #     return filtered_list

    # elif language == "Hindi" or language == "Bengali":
    #     temp_list = input_text.strip().split("।")
    #     filtered_list = [element.strip() + '।' for element
    #     if temp_list[-1].strip():
    #         filtered_list.append(temp_list[-1].strip())
    #     return filtered_list
    # else:
    #   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 i
    random_string = uuid.uuid4().hex[:8].upper()
    file_name = f"/content/edge_tts_voice/{truncated_text}_{
    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_stri
      # edge_command=f'edge-tts  --rate={calculate_rate_stri

      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}.
      k+=1
    print(chunk_audio_list)
    merge_audio_files(chunk_audio_list, save_path)
```

```
  else:
    edge_command=f'edge-tts  --rate={calculate_rate_string(s
    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 = "Japanese" # @param ['Afrikaans', 'Amharic', 'Ara
Gender = "Male"# @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,



# remove_slience  = True # @param {type:"boolean"}
# 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
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_s
  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)
# text="hi how are you"
# speed=1
# audio_path='/content/test.mp3'
# Language='English'
# Gender='Male'
# long_sentence=True
# translate_text_flag=True
# process_tts(text,speed,audio_path,Language,Gender,long_sen
# Audio(audio_path, autoplay=True)
```

0:02 / 0:02

```
import pysrt

input_srt_path = '/content/whisper_subtitles/This_is_importa

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 versio
with open(output_srt_path, "w", encoding='utf-8') as output_
    for subtitle in subtitles:
        output_file.write(f"{subtitle.index}\n")
        output_file.write(f"{subtitle.start} --> {subtitle.e
        output_file.write(f"{clean_subtitle_text(subtitle.te
        output_file.write(f"\n")

print(f"Cleaned subtitle file saved at: {output_srt_path}")
```

input_srt_path: "/content/whisper_subtitles/This_is_imp"

```
Cleaned subtitle file saved at: /content/cleaned_subtitles.srt
```

If your subtitle already in Given Language uncheck `translate_text_flag`

```
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):
```

srt_file_path: "/content/cleaned_subtitles.srt"

language: Japanese

gender: Female

speed: 1

long_sentence_flag: ☐

should_translate: ☐

```
    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 = "Japanese"  # @param ['Afrikaans', 'Amharic', 'Arabic', 'Azerbaijani', 'Bulgarian', 'Bengali', 'Bosnian', 'Catalan
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 = 1  # @param {type: "number"}
    long_sentence_flag = False  # @param {type:"boolean"}
    should_translate = False  # @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)

```

```
Japanese Dubbed Audio File Saved At: /content/TTS_DUB/Japanese_cleaned_subtitles.wav
```

```
from moviepy.editor import VideoFileClip, AudioFileClip

# Load the video and audio files
video_clip = VideoFileClip("/content/user_upload/@English YouTube shortsshorts shortsindia trendingshorts youtubeshorts vira
dubbed_audio = AudioFileClip("/content/TTS_DUB/Japanese_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)

# Optional: Adjust the duration of the video to match the audio length
# video_with_dubbed_audio = video_with_dubbed_audio.subclip(0, dubbed_audio.duration)

# 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")

```

```
Moviepy - Building video /content/TTS_DUB/Japanese_final_video.mp4.
MoviePy - Writing audio in Japanese_final_videoTEMP_MPY_wvf_snd.mp4
MoviePy - Done.
Moviepy - Writing video /content/TTS_DUB/Japanese_final_video.mp4

Moviepy - Done !
Moviepy - video ready /content/TTS_DUB/Japanese_final_video.mp4
```