Configuration Manual

MSc Research Project
Data Analytics

Zeba Siddique
Student ID: x20227086

School of Computing
National College of Ireland

Supervisor: Dr. Abubakr Siddig

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Zeba Siddique
Student ID:	x20227086
Programme:	Data Analytics
Year:	2022
Module:	MSc Research Project
Supervisor:	Dr. Abubakr Siddig
Submission Due Date:	15/08/2022
Project Title:	Configuration Manual
Word Count:	1126
Page Count:	29

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Zeba Siddique
Date:	13th August 2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Zeba Siddique
x20227086

1 Introduction

This document is a configuration manual that contains all the necessary information required to achieve a child speech synthesis artefact. It comprises of the minimum and must-have system requirements for reproducing the thesis work. Along with the necessary system configurations and pre-requisites, this document with the help of code snippets describes the main blocks of the thesis. The document details the step-by-step instructions from data collection to generating results to executing the artefact.

2 Required Specifications

2.1 Hardware requirements

Figure 1 describes the hardware specifications (device and windows) used to carry out the thesis.

Device Specifications

HP Spectre x360 Convertible 14-ea0xxx

Device name	DESKTOP-KHUORMC
Processor	11th Gen Intel(R) Core (TM) i7-1165G7 @ 2.80GHz 2.80 GHz
Installed RAM	16.0 GB (15.6 GB usable)
Device ID	B3992509-FEF9-4008-A80E-1FD4E8A47312
Product ID	00325-97258-57122-AAOEM
System type	64-bit operating system, x64-based processor
Pen and touch	Pen and touch support with 10 touch points

Windows Specifications

Windows 10

Edition	Windows 10 Home
Version	21H2
Installed on	23-Sep-21
OS build	19044.1889
Experience	Windows Feature Experience Pack 120.2212.4180.0

Figure 1: Hardware Specifications

2.2 Software requirements

For execution of the artefacts, the below mentioned list of software must be installed on the system.

1. Anaconda Navigator for Windows (Version 4.12.0)
2. Python 3.9.13
3. Visual Studio Code (Version 1.70.0 [user setup])
4. VS Extensions:
   (a) Jupyter (v2022.7.1102252217)
   (b) Jupyter Keymap (v1.0.0)
   (c) Jupyter Notebook Renderers (v1.0.9)
   (d) Pylance (v2022.8.20)
   (e) Python (v2022.12.0)
5. Google Chrome (Version 104.0.5112.81)

2.3 Storage requirements/ Products/ Subscriptions

Following are the additional and essential requirements used to carry out the thesis:

1. Google Colaboratory Pro+
2. Google One/ Drive - 2TB storage

3 Data Collection

This research uses a freely available multi-speaker child speech dataset: My Science Tutor (MyST). This dataset was obtained through a shared drive after receiving necessary permissions from concerned authorities and agreeing to terms and conditions for the research license agreement. Access to MyST Corpus can be requested from the official website of the dataset [1]. The dataset contains audio references (.flac) and their transcripts (.trn) if any for each speaker based on the recording phase. The MyST corpus consists of 456 hours speech data from 1,371 students.

4 Data Cleaning and Pre-processing

Out of the total child speech data available in the MyST corpus only 45% of the audio references are transcribed. This thesis focuses on the audio references that have a transcript file. The data is sampled using the non-probability based purposive sampling to use the transcribed audio references for phase 2 sessions. Based on the works of Jain et al. (2022) on the same corpus, the MyST corpus is processed. The data cleaning and pre-processing steps undertaken for MyST corpus for this research are described below:

[1] MyST Corpus: `https://boulderlearning.com/request-the-myst-corpus/`

4.1 Import Libraries for preparing the MyST corpus

Figure 2 demonstrates the libraries to be imported to pre-process the MyST corpus. The figure also mentions the different .py files that facilitate the cleaning and pre-processing of MyST corpus.

```
MyST Preprocessing > DataPrep > libs > flags.py > ...
from enum import Enum

MyST Preprocessing > DataPrep > extract_readings.py > ...
import asyncio
import logging
from typing import Any

from libs.flags import Dataset
from libs.data_placement import DataSetChanges

MyST Preprocessing > DataPrep > libs > data_placement.py > ...
import os
import string
import librosa
import matplotlib.pyplot as plt
from multimethod import RETURN
import numpy as np
import pandas as pd
import soundfile as sf
import librosa.display

from glob import glob
from typing import Dict, List
from libs.flags import Dataset
from asyncio.log import logger
```

Figure 2: Required Python Libraries for MyST corpus

4.2 Creating Constants

Figure 3 demonstrates the constants used during the cleaning and pre-processing of the MyST corpus.

```
MyST Preprocessing > DataPrep > libs > flags.py > ...
from enum import Enum

class Dataset(Enum):
    NAME = "My Science Tutor"
    URL = ""
    ACTUAL_PATH  = r'C:\Users\zzeba\Documents\Zeba\NCI\Sem3\Thesis\myst-v0.4.2\data'
    MODIFIED_PATH = r'C:\Users\zzeba\Documents\Zeba\NCI\Sem3\Thesis\MySTDataset'
    DEV_RUN = "development"
    TRAIN_RUN = "train"
    TEST_RUN = "test"
    RECORDING_CSV_FILE_PATH = r'C:\Users\zzeba\Desktop\Child TTS - MyST\MyST Preprocessing\DataPrep\InitialAudioDetails.csv'
    METADATA_FILE_PATH = r'MyST Preprocessing\MyST_DataSet\metadata.txt'
```

Figure 3: Defining Constants for MyST corpus

4.3 Data Cleaning

Figure 4 demonstrates the execution point for cleaning the MyST corpus.

MyST Preprocessing > DataPrep > extract_readings.py > ...

```python
async def extract_readings(run_type: str) -> None:
    """Pre-process the MyST Speech corpus
    """
    try:
        await DataSetChanges._read_dataset(run_type)
    except:
        logger.exception('Get Dataset Details: %s')

if __name__ == '__main__':
    params_myst = {
        'run_type' : 'development'
    }
    handler(params_myst, None)


```

MyST Preprocessing > DataPrep > libs > data_placement.py > _extract_recording_details

```python
class DataSetChanges():

    async def _read_dataset(run_type: str) -> None:
        """Navigate to the file path and wait for processing to finish"""
        await _extract_recording_details() # step 1
        recording_df = await _read_recording_details_from_csv() # step 2
        await _remove_silence_from_audio_files(recording_df=recording_df) # step 3


```

Figure 4: Code Execution Point

Figure 5 demonstrates the selection of transcribed audio references that are 10-15 seconds in length and free from dis-fluency markers. The punctuation in the transcripts are replaced. Additionally, the details of the selected transcribed audio files are saved in a CSV file for further processing

```
MyST Preprocessing > DataPrep > libs > data_placement.py > ...
async def _extract_recording_details():
    """ Get the valid audio references and transcripts from the
    MyST corpus
    """
    duration_of_recordings = []
    root_directory = Dataset.ACTUAL_PATH.value
    batch_counter = 1

    speech_data_details = []

    for dir_path, sub_directories, files in os.walk(root_directory):
        for filename in files:
            is_valid_session = await _check_for_phase_two_sessions(directory_path=dir_path)
            if is_valid_session:
                if filename.endswith(".flac"):
                    flac_file = os.path.join(dir_path,filename)
                    trn_file = flac_file.replace(".flac", ".trn")
                    is_valid_transcript, transcript_content = await _check_for_valid_transcript(trn_file=trn_file)
                    if is_valid_transcript :
                        speech_data = await _get_data_details_in_row(
                            flac_file=flac_file, trn_file=trn_file,
                            transcript_content=transcript_content
                        )
                        if speech_data != []:
                            speech_data_details.append(speech_data)
                            if (len(speech_data_details) % 100 == 0):
                                await _write_speech_details_to_file(speech_data_details)
                                speech_data_details = []
                                print("Batch completed : ", batch_counter)
                                batch_counter += 1
                    else:
                        continue
    if len(speech_data_details) > 0:
        await _write_speech_details_to_file(speech_data_details)
        speech_data_details = []
        print("Batch completed as last : ", batch_counter)

    print ("function finished execution")
```

Figure 5: Selection of Audio-Transcripts from MyST corpus

Figures 6, 7, and 8 demonstrates the transcript cleaning, audio selection, and creation of CSV file respectively.

MyST Preprocessing > DataPrep > libs > data_placement.py > _get_data_details_in_row

```python
async def _check_for_phase_two_sessions(directory_path):
    """Pick audio references that are recorded
    for Phase 2 sessions of the MyST Corpus
    Purposive sampling: Non-probability"""
    isValid = False
    phase_two_sessions = ['_EE_', '_MX_', '_SMP_', '_SRL_', '_LS_']
    for session_name in phase_two_sessions:
        if session_name in directory_path:
            isValid = True
    return isValid

async def _check_for_valid_transcript(trn_file):
    """ Check for transcripts that do not have any
    disfluency markers. Return False if a the transcript contains
    disfluency else remove the punctuations from the transcript
    if exist and return True
    """
    isValid = True
    transcript_content = None
    disfluency_markers = ['<NO_SIGNAL>',
    '<SILENCE>', '<BREATH>', '<LAUGH>', '<COUGH>',
    '<NOISE>', '<SIDE_SPEECH>', '<SNIFF>', '<ECHO>',
    '<DISCARD>', '<INDISCERNIBLE>', '(*)', '(())']
    if os.path.exists(trn_file):
        with open(trn_file, 'r', encoding='ascii', errors='ignore') as transcript_file:
            transcript_content = transcript_file.read()
            transcript_content = transcript_content.upper()
            #print(transcript_content)
            if any(ele in transcript_content for ele in disfluency_markers):
                isValid = False
                transcript_content = None
    else:
        isValid = False

    if isValid :
        transcript_content = ''.join([i for i in transcript_content if i not in string.punctuation])
        #print("Removed Punctuation")
    return isValid, transcript_content
```

Figure 6: Purposive Sampling and Transcript Cleaning

MyST Preprocessing > DataPrep > libs > data_placement.py > ...

```python
async def _get_data_details_in_row(flac_file : str, trn_file : str, transcript_content : str):
    """ Get valid audio reference and transcript file names along
    with additional details in an array
    """
    speech_details = []

    samples, sample_rate = librosa.load(flac_file)
    duration_of_recording= round(librosa.get_duration(y = samples, sr = sample_rate),2)
    if duration_of_recording >= 10 and duration_of_recording <= 15:
        speech_details = [
            _get_speaker_id(file_name=flac_file),
            duration_of_recording,
            flac_file,
            trn_file,
            transcript_content.strip(),
            sample_rate
            ]
    return speech_details

def _rows_to_dataframe(speech_data_details : List) -> pd.DataFrame:
    """Convert the array into a pandas dataframe"""
    speech_details_df = pd.DataFrame(
            speech_data_details,
            columns=[
                'SpeakerID',
                'AudioLength',
                'AudioFilePath',
                'TranscriptFilePath',
                'TranscriptText',
                'AudioSampleRate'
            ]
        )

    return speech_details_df


def _get_speaker_id(file_name : str):
    """Get the speaker ID from a file name"""
    return ((file_name.split("myst_"))[1]).split("_")[0]
```

Figure 7: Selection of Audio and Creation of DataFrame

MyST Preprocessing > DataPrep > libs > data_placement.py > ...

```python
async def _write_speech_details_to_file(speech_data_details):
    """Put the details of the valid files in a text file"""
    speech_data_dataframe = _rows_to_dataframe(speech_data_details)
    file_name = r'C:\Users\zzeba\Desktop\MyST Preprocessing\DataPrep\InitialAudioDetails.csv'
    if not os.path.exists(file_name):
        speech_data_dataframe.to_csv('DataPrep/InitialAudioDetails.csv',
        index=False, encoding='utf-8-sig'
        )
        print("Created CSV file")
    else:
        speech_data_dataframe.to_csv('DataPrep/InitialAudioDetails.csv',
        mode='a', header=False, index=False, encoding='utf-8-sig'
        )
        print("Appended to CSV file")
```

Figure 8: MyST CSV File Creation

4.4 Data Pre-processing

For performing data pre-processing on the MyST corpus, the CSV file is retrieved to obtain the cleaned audio references. The following figures 9 and 10 demonstrate the steps carried on the MyST corpus.

MyST Preprocessing > DataPrep > libs > data_placement.py > ...

```python
async def _read_recording_details_from_csv():
    """Read the CSV file consisting details of recordings and return a pandas dataframe"""
    if os.path.exists(Dataset.RECORDING_CSV_FILE_PATH.value):
        recording_df = pd.read_csv(Dataset.RECORDING_CSV_FILE_PATH.value)
    return recording_df

async def _remove_silence_from_audio_files(recording_df):
    """Remove all the silence between the audio files"""
    if len(recording_df) > 0:
        try:
            for row in recording_df.itertuples(index=True, name='Pandas'):
                if os.path.exists(row.AudioFilePath):
                    wav_file_name = (row.AudioFilePath.split("\\")[-1]).split(".flac")[0]
                    wav_file_name_with_extension = wav_file_name + ".wav"
                    wav_file_path = r'MyST Preprocessing\MyST_DataSet\wavs' + "\\" + wav_file_name_with_extension

                    audio_file, sample_rate = librosa.load(row.AudioFilePath, sr= row.AudioSampleRate, mono=True)
                    # remove the silence from between the voice signals
                    clips = librosa.effects.split(audio_file, top_db=100)

                    # combine audio clips without silence
                    wav_data = []
                    if(len(clips) > 0):
                        for clip in clips:
                            audio_data = audio_file[clip[0]: clip[1]]
                            wav_data.extend(audio_data)
                        # save the wav file in the folder
                        sf.write(wav_file_path, wav_data, row.AudioSampleRate)
                    else:
                        print("no clips")
                        sf.write(wav_file_path, audio_file, row.AudioSampleRate)
                    # write the audio, transcript, and speaker details into a txt file
                    await _write_transcript_to_metadata(row, wav_file_name)
        except:
            logger.exception('Error in _remove_silence_from_audio_files: %s')
        finally:
            logger.exception('Finished executing _remove_silence_from_audio_files')
```

Figure 9: MyST Data Pre-processing

```
MyST Preprocessing > DataPrep > libs > data_placement.py > ...
async def _write_transcript_to_metadata(audio_details, audio_file_name):
    """Write details of the audio recording to a txt file"""
    file_path = r'MyST Preprocessing\MyST_DataSet\metadata.txt'
    audio_info = audio_file_name + "|" + audio_details.TranscriptText + "|" + str(audio_details.SpeakerID) + "\n"
    file_mode = ""
    try:
        if not os.path.exists(file_path):
            file_mode = "w"
        else:
            file_mode = "a"
        with open(file_path, mode = file_mode) as f:
            f.write(audio_info)
    except:
        logger.exception('Error in _write_transcript_to_metadata: %s')
```

Figure 10: MyST Data Metadata

Figure 11 demonstrates the distribution of duration of the pre-processed audio reference files.

```
MyST Preprocessing > DataPrep > libs > data_placement.py > ...
async def _plot_audio_length_distribution(speech_data_dataframe : pd.DataFrame):
    # Plot audio duraiton distribution
    n, bins, patches = plt.hist(x=speech_data_dataframe['AudioLength'], bins='auto', color='#0504aa',
                                alpha=0.7, rwidth=0.85)
    plt.grid(axis='y', alpha=0.75)
    plt.xlabel('Duration in seconds')
    plt.ylabel('Number of Audio')
    plt.title('Distribution of audio duration')
    maxfreq = n.max()
    # Set a clean upper y-axis limit.
    plt.ylim(ymax=np.ceil(maxfreq / 10) * 10 if maxfreq % 10 else maxfreq + 10)
    plt.show()
```

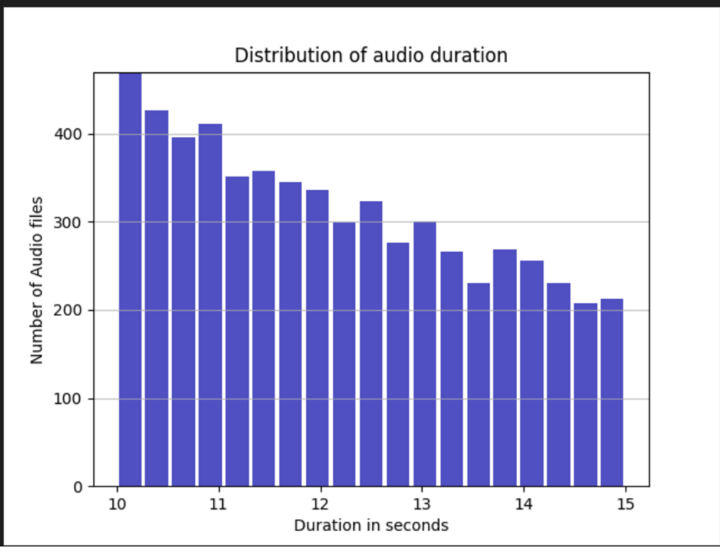


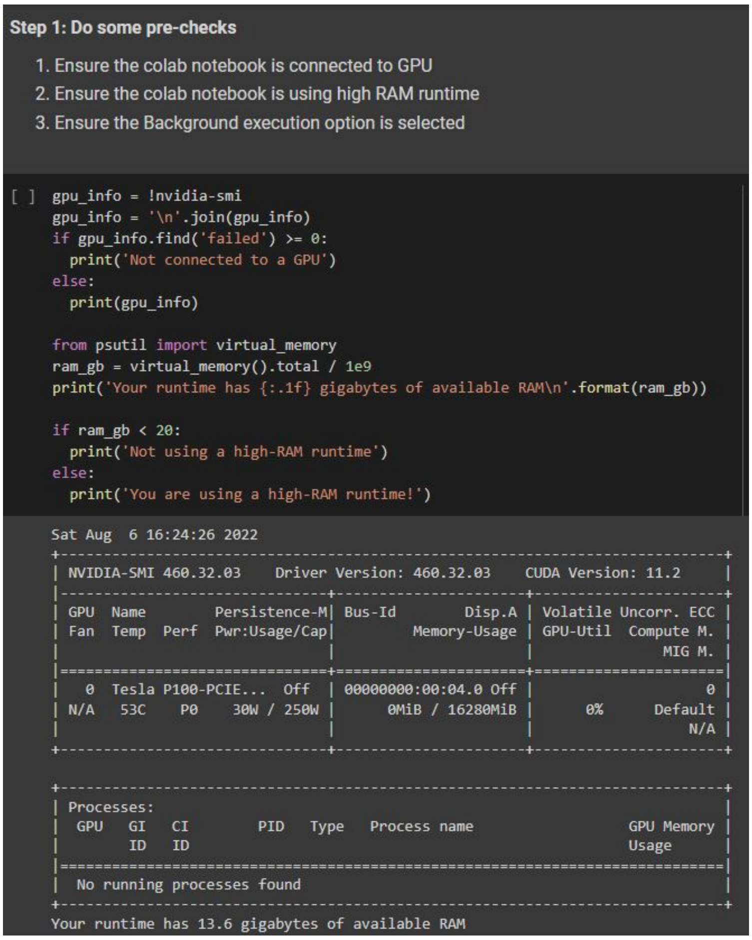
Figure 11: Pre-processed Audio Distribution

5 Implementation of the HiFi-GAN vocoder

This section describes the implementation of the HiFi-GAN vocoder that uses the python TTS library (for training from scratch) [2] and the coqui-ai TTS Github repository (for fine-tuning) [3]. The rest of the artefeacts are executed on Google Colab Pro+ with hardware accelerator selected as 'GPU', with High Run-time and background execution enabled. Note: The pre-processed MyST corpus having folder structure as

- Folder: DataSetName
  - Folder: wavs (containing all pre-processed files)
  - Folder: wavs1 (containing 10% files from the wavs folder)
  - File: metadata.txt

5.1 Run pre-requisites



Step 1: Do some pre-checks

1. Ensure the colab notebook is connected to GPU
2. Ensure the colab notebook is using high RAM runtime
3. Ensure the Background execution option is selected

```
[ ] gpu_info = !nvidia-smi
    gpu_info = '\n'.join(gpu_info)
    if gpu_info.find('failed') >= 0:
      print('Not connected to a GPU')
    else:
      print(gpu_info)

    from psutil import virtual_memory
    ram_gb = virtual_memory().total / 1e9
    print('Your runtime has {:.1f} gigabytes of available RAM\n'.format(ram_gb))

    if ram_gb < 20:
      print('Not using a high-RAM runtime')
    else:
      print('You are using a high-RAM runtime!')
```

```
Sat Aug  6 16:24:26 2022
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.32.03    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla P100-PCIE...  Off  | 00000000:00:04.0 Off |                    0 |
| N/A   53C    P0    30W / 250W |      0MiB / 16280MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
Your runtime has 13.6 gigabytes of available RAM
```

Figure 12: Colab Pre-requisites for HiFi-GAN

[2]Python TTS library https://pypi.org/project/TTS/
[3]Python TTS library https://github.com/coqui-ai/TTS

Step 2: Download the My Science Tutor Children Speech Corpus

Step 3: Clean and pre-process the MyST dataset by executing extract_readings.py file

Step 4: Upload the preprocessed .wav files and the metadata.txt file obtained from Step 3 to Google Drive

Folder Structure:

1. MyST_DataSet
    1. wavs (folder)
    2. wavs1 (pick roughly 500 files from wavs folder into this folder)
    3. metadata.txt

Step 5: Mount the drive folder

```
[ ] from google.colab import drive
    drive.mount('/content/gdrive', force_remount=True)

    Mounted at /content/gdrive
```

Figure 13: Mount Google Drive

5.2 Import Libraries for implementing the HiFi-GAN vocoder

The following Figure demonstrates all the necessary libraries required to implement the HiFi-GAN vocoder to be trained on the MyST corpus.

Install & Import Required Libraries

Step 6: Install & Import necessary libraries for training the HiFi-GAN vocoder model on MyST corpus

Note that: For training the HiFi-GAN, the Python TTS library is used.

```
[ ] !pip install TTS

    Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
    Requirement already satisfied: TTS in /usr/local/lib/python3.7/dist-packages (0.7.1)

!pip install trainer

    Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
    Requirement already satisfied: trainer in /usr/local/lib/python3.7/dist-packages (0.0.13)
    Requirement already satisfied: protobuf<3.20,>=3.9.2 in /usr/local/lib/python3.7/dist-packages (from trainer) (3.17.3)

!pip install IPython

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: IPython in /usr/local/lib/python3.7/dist-packages (5.5.0)
Requirement already satisfied: traitlets>=4.2 in /usr/local/lib/python3.7/dist-packages (from IPython) (5.1.1)

[ ] import os

[ ] from trainer import Trainer, TrainerArgs

[ ] from TTS.utils.audio import AudioProcessor
    from TTS.vocoder.configs import HifiganConfig
    from TTS.vocoder.datasets.preprocess import load_wav_data
    from TTS.vocoder.models.gan import GAN

[ ] from TTS.tts.configs.shared_configs import BaseDatasetConfig
    from TTS.tts.datasets import load_tts_samples

[ ] import librosa
    import librosa.display
    import matplotlib.pyplot as plt
    import numpy as np
```

Figure 14: Import Libraries for HiFi-GAN vocoder

5.3 Training the HiFi-GAN vocoder on MyST corpus

The implementation of the HiFi-GAN vocoder for this research uses the HiFi-GAN generator [4], discriminator [5] and the training config file [6]. The modifications in the configurations are demonstrated in the subsequent figures.

Training the HiFi-GAN vocoder model from scratch on the pre-processed MyST children's speech corpus

Step 7: Provide the folder path to save the checkpoints for training the model from scratch

```
[ ] output_path = "/content/gdrive/MyDrive/Zeba_TTS/TTS_CP_HiFiGan" #"/content/gdrive/MyDrive/Zeba_TTS"
```

Step 8: Set the training parameters for the HiFi-GAN vocoder of the TTS library Note that: The parameters

1. data_path: Provide the folder path where the wavs1 or wavs folder is present
2. output_path: Provide the folder path where the checkpoints, training log and config will be stored
3. epochs: Set the number of epoch to train the model. As background execution option along with high RAM and GPU was available, the training epoch size was straight away set to 500. The checkpoints at regular intervals (100, 200, and so on) can be later evaluated

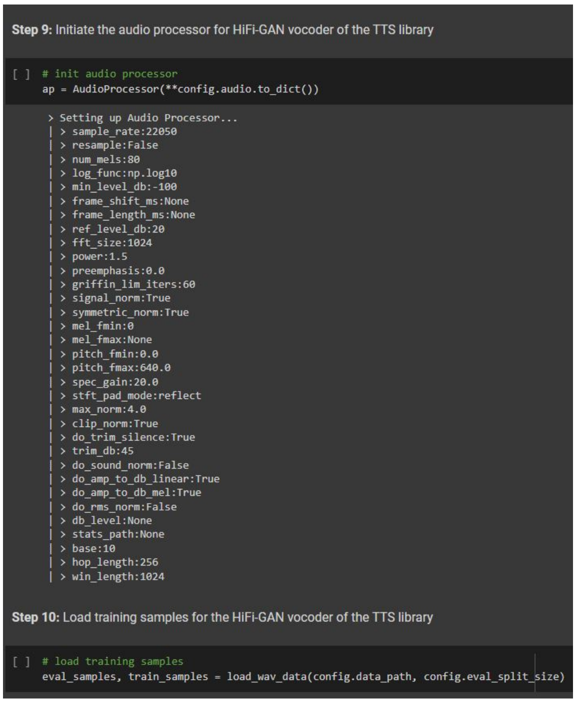
```
[ ] config = HifiganConfig(
        batch_size=32,
        eval_batch_size=16,
        num_loader_workers=4,
        num_eval_loader_workers=4,
        run_eval=True,
        test_delay_epochs=5,
        epochs=500,
        seq_len=8192,
        pad_short=2000,
        use_noise_augment=True,
        eval_split_size=100,
        print_step=25,
        print_eval=False,
        mixed_precision=False,
        lr_gen=1e-4,
        lr_disc=1e-4,
        data_path="/content/gdrive/MyDrive/Zeba_TTS/MyST_DataSet/wavs1",
        output_path=output_path,
    )
```

Figure 15: Modifying the parameters for HiFi-GAN vocoder

[4] HiFi-GAN Generator `https://github.com/coqui-ai/TTS/blob/dev/TTS/vocoder/models/hifigan_generator.py`

[5] HiFi-GAN Discriminator `https://github.com/coqui-ai/TTS/blob/dev/TTS/vocoder/models/hifigan_discriminator.py`

[6] HiFi-GAN Config `https://github.com/coqui-ai/TTS/blob/dev/TTS/vocoder/configs/hifigan_config.py`



Step 9: Initiate the audio processor for HiFi-GAN vocoder of the TTS library

```
[ ] # init audio processor
    ap = AudioProcessor(**config.audio.to_dict())

 > Setting up Audio Processor...
 | > sample_rate:22050
 | > resample:False
 | > num_mels:80
 | > log_func:np.log10
 | > min_level_db:-100
 | > frame_shift_ms:None
 | > frame_length_ms:None
 | > ref_level_db:20
 | > fft_size:1024
 | > power:1.5
 | > preemphasis:0.0
 | > griffin_lim_iters:60
 | > signal_norm:True
 | > symmetric_norm:True
 | > mel_fmin:0
 | > mel_fmax:None
 | > pitch_fmin:0.0
 | > pitch_fmax:640.0
 | > spec_gain:20.0
 | > stft_pad_mode:reflect
 | > max_norm:4.0
 | > clip_norm:True
 | > do_trim_silence:True
 | > trim_db:45
 | > do_sound_norm:False
 | > do_amp_to_db_linear:True
 | > do_amp_to_db_mel:True
 | > do_rms_norm:False
 | > db_level:None
 | > stats_path:None
 | > base:10
 | > hop_length:256
 | > win_length:1024
```

Step 10: Load training samples for the HiFi-GAN vocoder of the TTS library

```
[ ] # load training samples
    eval_samples, train_samples = load_wav_data(config.data_path, config.eval_split_size)
```

Figure 16: Initiate Audio Processor for HiFi-GAN vocoder

Step 10: Load training samples for the HiFi-GAN vocoder of the TTS library

```
[ ] # load training samples
    eval_samples, train_samples = load_wav_data(config.data_path, config.eval_split_size)
```

Step 10: Initiate the HiFi-GAN vocoder for the TTS library Note that: The details of the generator and discriminator are displayed to ensure the files that are being used from the TTS library

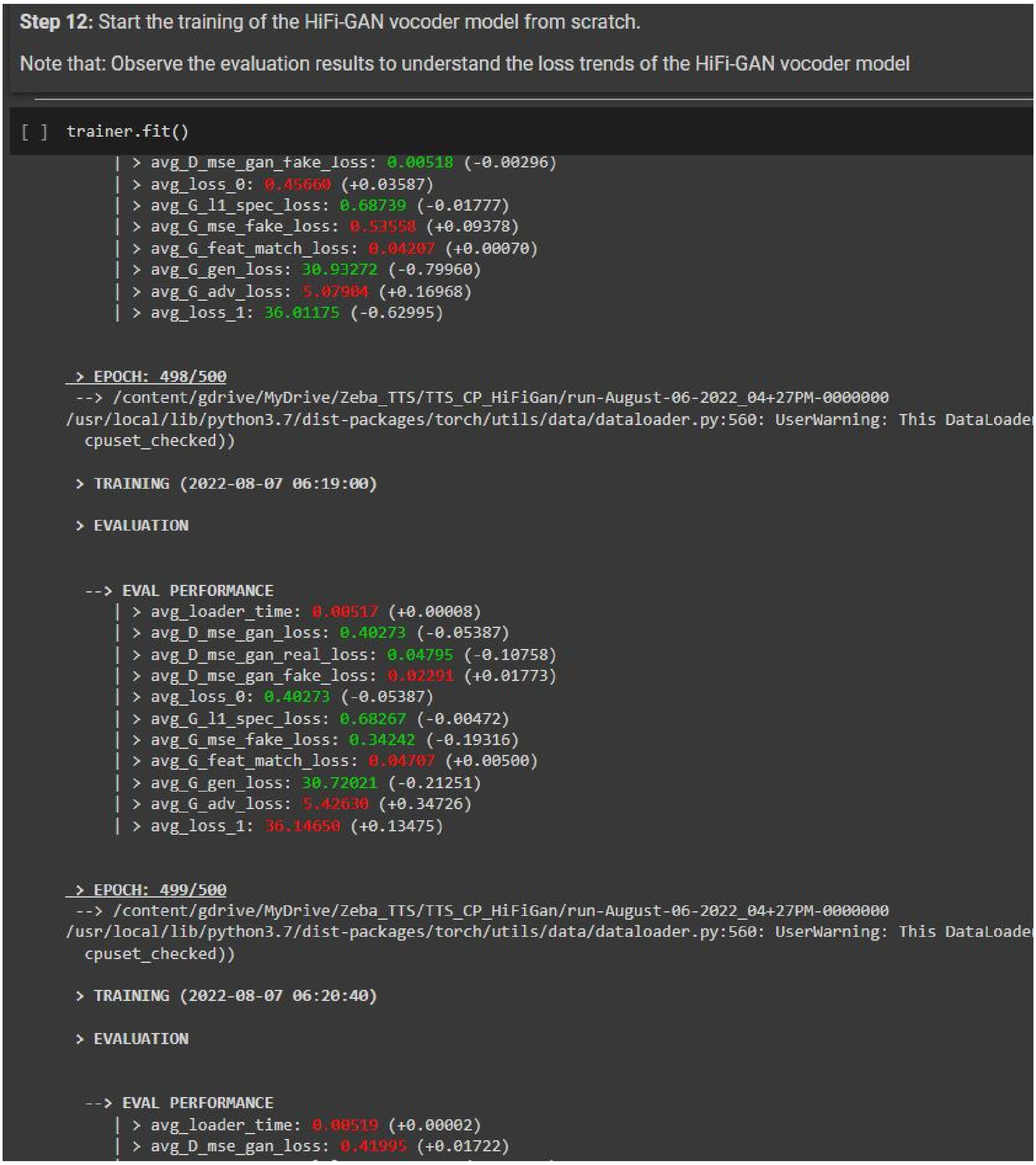
```
[ ] # init model
    model = GAN(config, ap)

 > Generator Model: hifigan_generator
 > Discriminator Model: hifigan_discriminator
```

Step 11: Initiate the Trainer model to train the HiFi-GAN vocoder for the TTS library

```
[ ] # init the trainer and
    trainer = Trainer(
        TrainerArgs(), config, output_path, model=model, train_samples=train_samples, eval_samples=eval_samples
    )

 > Using CUDA: True
 > Number of GPUs: 1

 > Model has 84660721 parameters
```

Figure 17: Initiate Trainer Class for HiFi-GAN vocoder

The following figure displays the HiFi-GAN vocoder trained from scratch on pre-processed MyST corpus for 500 epochs



Step 12: Start the training of the HiFi-GAN vocoder model from scratch.

Note that: Observe the evaluation results to understand the loss trends of the HiFi-GAN vocoder model

```
[ ]  trainer.fit()
         | > avg_D_mse_gan_fake_loss: 0.00518 (-0.00296)
         | > avg_loss_0: 0.45660 (+0.03587)
         | > avg_G_l1_spec_loss: 0.68739 (-0.01777)
         | > avg_G_mse_fake_loss: 0.53558 (+0.09378)
         | > avg_G_feat_match_loss: 0.04207 (+0.00070)
         | > avg_G_gen_loss: 30.93272 (-0.79960)
         | > avg_G_adv_loss: 5.07904 (+0.16968)
         | > avg_loss_1: 36.01175 (-0.62995)


  > EPOCH: 498/500
  --> /content/gdrive/MyDrive/Zeba_TTS/TTS_CP_HiFiGan/run-August-06-2022_04+27PM-0000000
/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:560: UserWarning: This DataLoade
  cpuset_checked))

  > TRAINING (2022-08-07 06:19:00)

  > EVALUATION


   --> EVAL PERFORMANCE
         | > avg_loader_time: 0.00517 (+0.00008)
         | > avg_D_mse_gan_loss: 0.40273 (-0.05387)
         | > avg_D_mse_gan_real_loss: 0.04795 (-0.10758)
         | > avg_D_mse_gan_fake_loss: 0.02291 (+0.01773)
         | > avg_loss_0: 0.40273 (-0.05387)
         | > avg_G_l1_spec_loss: 0.68267 (-0.00472)
         | > avg_G_mse_fake_loss: 0.34242 (-0.19316)
         | > avg_G_feat_match_loss: 0.04707 (+0.00500)
         | > avg_G_gen_loss: 30.72021 (-0.21251)
         | > avg_G_adv_loss: 5.42630 (+0.34726)
         | > avg_loss_1: 36.14650 (+0.13475)


  > EPOCH: 499/500
  --> /content/gdrive/MyDrive/Zeba_TTS/TTS_CP_HiFiGan/run-August-06-2022_04+27PM-0000000
/usr/local/lib/python3.7/dist-packages/torch/utils/data/dataloader.py:560: UserWarning: This DataLoade
  cpuset_checked))

  > TRAINING (2022-08-07 06:20:40)

  > EVALUATION


   --> EVAL PERFORMANCE
         | > avg_loader_time: 0.00519 (+0.00002)
         | > avg_D_mse_gan_loss: 0.41995 (+0.01722)
```

Figure 18: Training the HiFi-GAN vocoder from scratch on the MyST corpus

5.4 Generating Audio References from the trained HiFi-GAN vocoder

The audio references were generated as shown in Figure 19 from the HiFi-GAN vocoder model trained on MyST corpus for 100 and 200 epochs for preliminary analysis. Figure 20 demonstrates the use of IPython library to hear the audio file generated from the vocoder

Step 13: Generate audio from the trained HiFi-GAN vocoder

1. Check the audio for 100 and 200 epochs and draw inferences from it

```
[ ] # Using checkpoint for 100 epochs to generate an output
    !tts --text "My dataset is on test for 100 epochs" \
    --model_name "tts_models/en/ljspeech/tacotron2-DDC" \
    --vocoder_name "vocoder_models/en/ljspeech/hifigan_v2" \
    --vocoder_path  "/content/gdrive/MyDrive/Zeba_TTS/TTS_CP_HiFiGan/run-August-06-2022_04+27PM-0000000/best_model_1400.pth" \
    --vocoder_config_path "/content/gdrive/MyDrive/Zeba_TTS/TTS_CP_HiFiGan/run-August-06-2022_04+27PM-0000000/config.json" \
    --out_path scratch100.wav
```

```
['My dataset is on test for 100 epochs']
 > Processing time: 2.375121593475342
 > Real-time factor: 0.6293432889844656
 > Saving output to scratch100.wav
```

```
[ ] !tts --text "My dataset is on test for 200 epochs" \
    --model_name "tts_models/en/ljspeech/tacotron2-DDC" \
    --vocoder_name "vocoder_models/en/ljspeech/hifigan_v2" \
    --vocoder_path  "/content/gdrive/MyDrive/Zeba_TTS/TTS_CP_HiFiGan/run-August-06-2022_04+27PM-0000000/best_model_1624.pth" \
    --vocoder_config_path "/content/gdrive/MyDrive/Zeba_TTS/TTS_CP_HiFiGan/run-August-06-2022_04+27PM-0000000/config.json" \
    --out_path scratch200.wav
```

```
  > Generator Model: hifigan_generator
  > Discriminator Model: hifigan_discriminator
 Removing weight norm...
  > Text: My dataset is on test for 200 epochs
  > Text splitted to sentences.
 ['My dataset is on test for 200 epochs']
  > Processing time: 2.422065496444702
  > Real-time factor: 0.6359133191632417
  > Saving output to scratch200.wav
```

Figure 19: Generating audio references from the HiFi-GAN vocoder for 100 and 200 epochs

Playing the sound generated by training the HiFi-GAN vocoder model purely on MyST corpus

```
[ ] # Gnerated voice for HiFi-GAN trained for 100 epochs from scratch on the MyST corpus
    import IPython
    IPython.display.Audio("/content/scratch100.wav")
```

0:00 / 0:03

```
[ ] # Gnerated voice for HiFi-GAN trained for 200 epochs from scratch on the MyST corpus
    import IPython
    IPython.display.Audio("/content/scratch200.wav")
```

0:00 / 0:03

Figure 20: Playing the audio generated from HiFi-GAN vocoder for 100 and 200 epochs

As the audio generated from the HiFi-GAN vocoder trained from the scratch was purely metallic, the transfer learning approach was adopted.

6 Transfer Learning approach for the HiFi-GAN vocoder

This section describes the implementation of the transfer learning approach of the HiFi-GAN vocoder. A pre-trained HiFi-GAN model was used to be further fine-tuned on the MyST dataset. The generator and the discriminator is the same as used in 5. The configuration parameters changed are described in the subsequent figures.

6.1 Import/ Install repository for implementing the transfer learning for the pre-trained HiFi-GAN vocoder

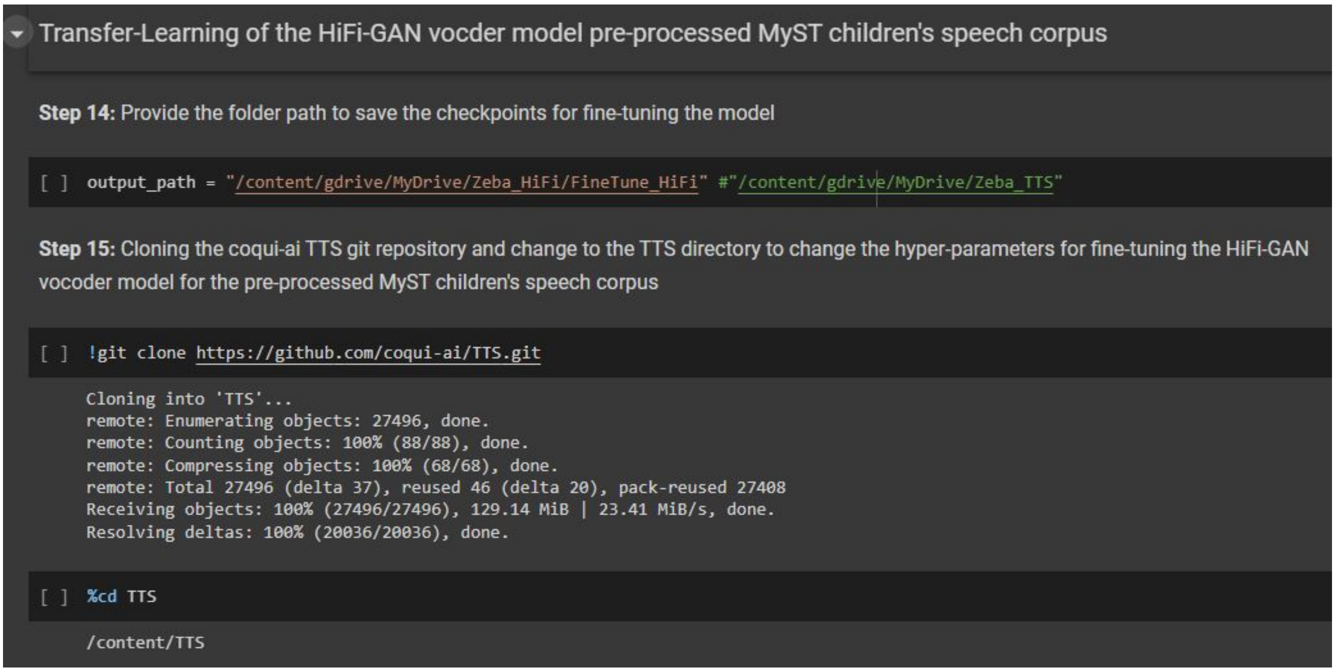


Figure 21: Cloning the TTS git repository

6.2 Modify the hyper-parameters in the HiFi-GAN vocoder configuration file

Step 16: Locate the train_hifigan.py file for the LJSpeech dataset (/content/TTS/recipes/ljspeech/hifigan/train_hifigan.py)

Change the necessary parameters. The parameters changed to fine-tune HiFi-GAN for MYST dataset are:

1. data_path = "/content/gdrive/MyDrive/Zeba_TTS/MyST_DataSet/wavs1"
2. output_path = "/content/gdrive/MyDrive/Zeba_HiFi/FineTune_HiFi"
3. restore_path = -- pre-trained HiFi-GAN path (for 1st run) then our checkpoint paths for next runs
4. eval_split_size = 100
5. epochs = 100 each time
6. resblock_type = 1
7. upsample_initial_channel = 512
8. resblock_kernel_sizes = [3,7,11]
9. resblock_dilation_sizes = [[1,3,5],[1, 3, 5],[1, 3, 5]]
10. upsample_factors = [8,8,2,2]
11. upsample_kernel_sizes = [16,16,4,4]
12. n_fft: 1024
13. hop_length: 256
14. win_length: 1024
15. n_mels: 80

Figure 22: Modifying the parameters for fine-tuning the HiFi-GAN vocoder

6.3 Start Fine-tuning and Training the HiFi-GAN Vocoder

Step 17: Start fine-tuning the HiFi-GAN vocoder model based on the hyper-parameters mentioned above

```
[ ] !CUDA_VISIBLE_DEVICES="0" python "/content/TTS/recipes/ljspeech/hifigan/train_hifigan.py" \
        --output_path="/content/gdrive/MyDrive/Zeba_HiFi/FineTune_HiFi"\
        --save_all_best = True
```

```
 > Setting up Audio Processor...
 | > sample_rate:22050
 | > resample:False
 | > num_mels:80
 | > log_func:np.log10
 | > min_level_db:-100
 | > frame_shift_ms:None
 | > frame_length_ms:None
 | > ref_level_db:20
 | > fft_size:1024
 | > power:1.5
 | > preemphasis:0.0
 | > griffin_lim_iters:60
 | > signal_norm:True
 | > symmetric_norm:True
 | > mel_fmin:0
 | > mel_fmax:None
 | > pitch_fmin:0.0
 | > pitch_fmax:640.0
 | > spec_gain:20.0
 | > stft_pad_mode:reflect
 | > max_norm:4.0
 | > clip_norm:True
 | > do_trim_silence:True
 | > trim_db:45
 | > do_sound_norm:False
 | > do_amp_to_db_linear:True
 | > do_amp_to_db_mel:True
 | > do_rms_norm:False
 | > db_level:None
 | > stats_path:None
 | > base:10
 | > hop_length:256
 | > win_length:1024
 > Generator Model: hifigan_generator
 > Discriminator Model: hifigan_discriminator
 > Using CUDA: True
 > Number of GPUs: 1

 > Model has 84660721 parameters

 > EPOCH: 0/100
 --> /content/gdrive/MyDrive/Zeba_HiFi/FineTune_HiFi/run-August-10-2022_12+38PM-d46fbc24

 > TRAINING (2022-08-10 12:38:04)
```

Figure 23: Fine-tune and train the HiFi-GAN vocoder on MyST corpus

The following figure demonstrates the repetition of the fine-tuning and training process for desired number of epochs (here 200).

Step 18: Repeat the training process for various epochs by changing the --restore_path parameter for the latest checkpoint available in each run

```
[ ]  !CUDA_VISIBLE_DEVICES="0" python "/content/TTS/recipes/ljspeech/hifigan/train_hifigan.py" \
        --output_path="/content/gdrive/MyDrive/Zeba_HiFi/FineTune_HiFi"\
        --restore_path="/content/gdrive/MyDrive/Zeba_HiFi/FineTune_HiFi/run-August-10-2022_12+38PM-d46fbc24/best_model_980.pth"\
        --save_all_best = True
```

```
 --> EVAL PERFORMANCE
    | > avg_loader_time: 0.00421 (-0.00006)
    | > avg_D_mse_gan_loss: 0.36248 (-0.01204)
    | > avg_D_mse_gan_real_loss: 0.04637 (-0.05039)
    | > avg_D_mse_gan_fake_loss: 0.01144 (+0.00312)
    | > avg_loss_0: 0.36248 (-0.01204)
    | > avg_G_l1_spec_loss: 0.67334 (-0.01766)
    | > avg_G_mse_fake_loss: 0.50474 (-0.01965)
    | > avg_G_feat_match_loss: 0.05486 (+0.00170)
    | > avg_G_gen_loss: 30.30023 (-0.79476)
    | > avg_G_adv_loss: 6.42967 (+0.16438)
    | > avg_loss_1: 36.72990 (-0.63039)
```

```
!CUDA_VISIBLE_DEVICES="0" python "/content/TTS/recipes/ljspeech/hifigan/train_hifigan.py" \
    --output_path="/content/gdrive/MyDrive/Zeba_HiFi/FineTune_HiFi"\
    --restore_path="/content/gdrive/MyDrive/Zeba_HiFi/FineTune_HiFi/run-August-10-2022_01+49PM-d46fbc24/best_model_1111.pth"\
    --save_all_best = True
```

```
    | > D_mse_gan_loss: 0.31493  (0.32491)
    | > D_mse_gan_real_loss: 0.04014  (0.03069)
    | > D_mse_gan_fake_loss: 0.00761  (0.02420)
    | > loss_0: 0.31493  (0.32491)
    | > grad_norm_0: 4.23894  (5.86320)
    | > G_l1_spec_loss: 0.53196  (0.52914)
    | > G_mse_fake_loss: 0.53478  (0.50658)
    | > G_feat_match_loss: 0.06495  (0.06166)
    | > G_gen_loss: 23.93812  (23.81123)
    | > G_adv_loss: 7.54901  (7.16551)
    | > loss_1: 31.48713  (30.97674)
    | > grad_norm_1: 1360.89795  (880.02283)
    | > current_lr_0: 0.00009
    | > current_lr_1: 0.00009
    | > step_time: 2.07910  (2.13160)
    | > loader_time: 0.00350  (0.00204)


 > CHECKPOINT : /content/gdrive/MyDrive/Zeba_HiFi/FineTune_HiFi/run-August-10-2022_03+03PM-d46fbc24/checkpoint_2100.pth

 > EVALUATION


 --> EVAL PERFORMANCE
    | > avg_loader_time: 0.00453 (-0.00001)
    | > avg_D_mse_gan_loss: 0.33334 (-0.03052)
    | > avg_D_mse_gan_real_loss: 0.04228 (-0.03455)
    | > avg_D_mse_gan_fake_loss: 0.00998 (+0.00377)
    | > avg_loss_0: 0.33334 (-0.03052)
    | > avg_G_l1_spec_loss: 0.67433 (-0.01866)
    | > avg_G_mse_fake_loss: 0.46126 (-0.01942)
```

Figure 24: Restoring the training process for fine-tuning the HiFi-GAN vocoder

7 Synthesizing voice from the proposed model

This section describes the implementation of the proposed model (Speaker encoder - Tacotron 2 - HiFi-GAN) . The implementations uses the implementation of the speaker encoder and Tacotron model [7] and the pre-trained model weights from [8] and [9]. Following figures demonstrate the steps to synthesize voice from the proposed model.

7.1 Import/ Install repository for implementing the proposed model that contains transfer learning for the pre-trained HiFi-GAN vocoder

Step 19: To synthesize voice from the proposed TTS model that uses:

1. Speaker Encoder Model (pre-trained on 4 corpus)
2. Tacotron 2 (pre-trained on 2 corpus)
3. Fine-tuned HiFi-GAN vocoder model

Clone the github repo https://github.com/CorentinJ/Real-Time-Voice-Cloning.git Allow the following code to download the repo's c models

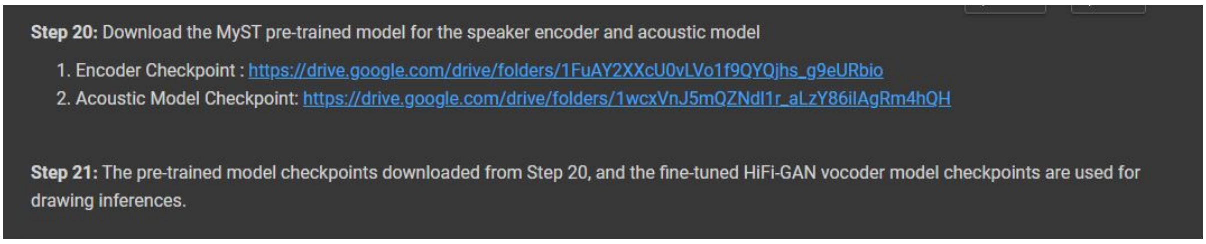
```
[ ] #@title Setup CorentinJ/Real-Time-Voice-Cloning

    #@markdown * clone the project
    #@markdown * download pretrained models

    %tensorflow_version
    import os
    from os.path import exists, join, basename, splitext

    git_repo_url = 'https://github.com/CorentinJ/Real-Time-Voice-Cloning.git'
    project_name = splitext(basename(git_repo_url))[0]
    if not exists(project_name):
      # clone and install
      !git clone -q --recursive {git_repo_url}
      # install dependencies
      !cd {project_name} && pip install -q -r requirements.txt
      !pip install -q --upgrade gdown
      !apt-get install -qq libportaudio2
      !pip install -q https://github.com/tugstugi/dl-colab-notebooks/archive/colab_utils.zip

      # download pretrained model
      #!cd {project_name} && wget https://github.com/blue-fish/Real-Time-Voice-Cloning/releases/download/v1.0/pretrai
      !cd {project_name} && mkdir -p saved_models/default/
      !cd {project_name}/saved_models/default/ && gdown https://drive.google.com/uc?id=1q8mEGwCkFy23KZsinbuvdKAQLqNKt
      !cd {project_name}/saved_models/default/ && gdown https://drive.google.com/uc?id=1EqFMIbvxffxtjiVrtykroF6_mUh-5
      !cd {project_name}/saved_models/default/ && gdown https://drive.google.com/uc?id=1cf2NO6FtI0jDuy8AV3Xgn6leO6dHj

    import sys
    sys.path.append(project_name)

    from IPython.display import display, Audio, clear_output
    from IPython.utils import io
    import ipywidgets as widgets
    import numpy as np
    from dl_colab_notebooks.audio import record_audio, upload_audio

    from synthesizer.inference import Synthesizer
```

Figure 25: Cloning the git repository and load default models

[7] Speaker encoder and Tacotron 2 Implementation `https://github.com/CorentinJ/Real-Time-Voice-Cloning`

[8] Speaker encoder checkpoints `https://drive.google.com/drive/folders/1FuAY2XXcU0vLVo1f9QYQjhs_g9eURbio`

[9] Acoustic model checkpoints `https://drive.google.com/drive/folders/1wcxVnJ5mQZNdl1r_aLzY86iIAgRm4hQH`



Step 20: Download the MyST pre-trained model for the speaker encoder and acoustic model

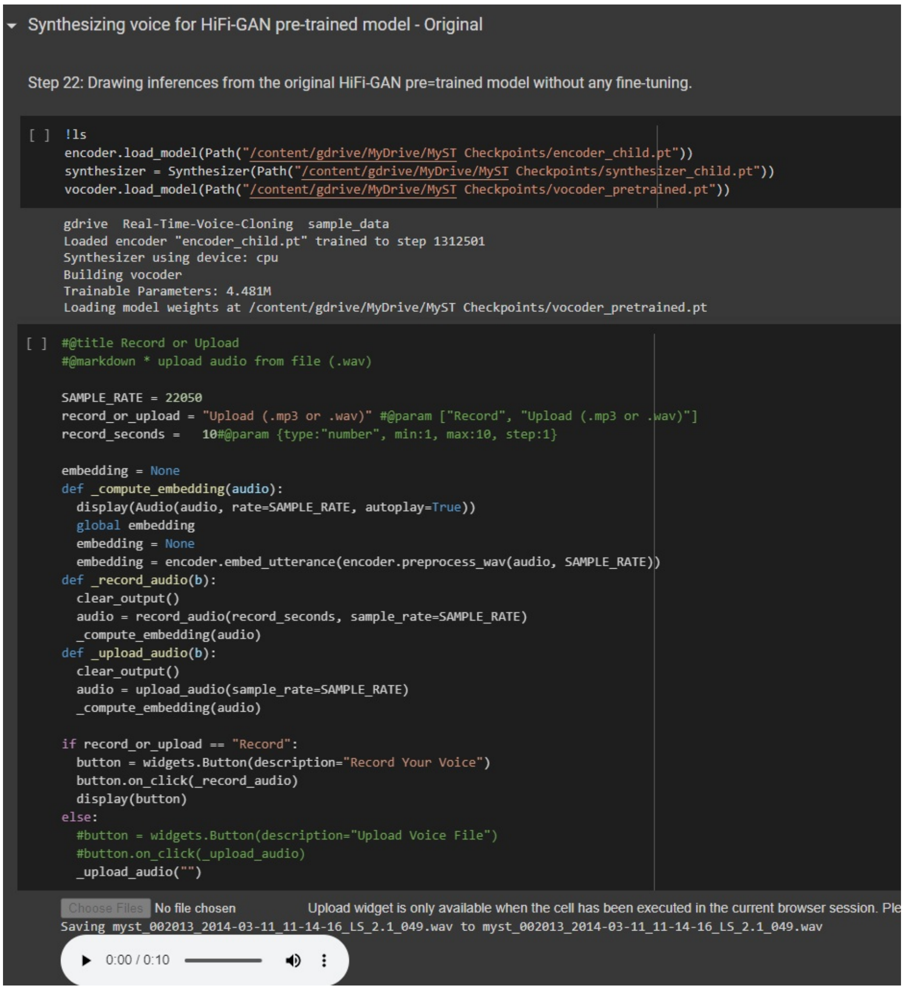
1. Encoder Checkpoint : https://drive.google.com/drive/folders/1FuAY2XXcU0vLVo1f9QYQjhs_g9eURbio
2. Acoustic Model Checkpoint: https://drive.google.com/drive/folders/1wcxVnJ5mQZNdl1r_aLzY86ilAgRm4hQH

Step 21: The pre-trained model checkpoints downloaded from Step 20, and the fine-tuned HiFi-GAN vocoder model checkpoints are used for drawing inferences.

Figure 26: Download pre-trained models and use HiFi-GAN pre-trained model

7.2 Synthesize voice using pre-trained HiFi-GAN vocoder in the proposed model

The following figures demonstrate loading the pre-trained HiFi-GAN vocoder model and synthesize voice from the proposed model.



Synthesizing voice for HiFi-GAN pre-trained model - Original

Step 22: Drawing inferences from the original HiFi-GAN pre=trained model without any fine-tuning.

```
!ls
encoder.load_model(Path("/content/gdrive/MyDrive/MyST Checkpoints/encoder_child.pt"))
synthesizer = Synthesizer(Path("/content/gdrive/MyDrive/MyST Checkpoints/synthesizer_child.pt"))
vocoder.load_model(Path("/content/gdrive/MyDrive/MyST Checkpoints/vocoder_pretrained.pt"))
```

```
gdrive  Real-Time-Voice-Cloning  sample_data
Loaded encoder "encoder_child.pt" trained to step 1312501
Synthesizer using device: cpu
Building vocoder
Trainable Parameters: 4.481M
Loading model weights at /content/gdrive/MyDrive/MyST Checkpoints/vocoder_pretrained.pt
```

```
#@title Record or Upload
#@markdown * upload audio from file (.wav)

SAMPLE_RATE = 22050
record_or_upload = "Upload (.mp3 or .wav)" #@param ["Record", "Upload (.mp3 or .wav)"]
record_seconds =   10#@param {type:"number", min:1, max:10, step:1}

embedding = None
def _compute_embedding(audio):
  display(Audio(audio, rate=SAMPLE_RATE, autoplay=True))
  global embedding
  embedding = None
  embedding = encoder.embed_utterance(encoder.preprocess_wav(audio, SAMPLE_RATE))
def _record_audio(b):
  clear_output()
  audio = record_audio(record_seconds, sample_rate=SAMPLE_RATE)
  _compute_embedding(audio)
def _upload_audio(b):
  clear_output()
  audio = upload_audio(sample_rate=SAMPLE_RATE)
  _compute_embedding(audio)

if record_or_upload == "Record":
  button = widgets.Button(description="Record Your Voice")
  button.on_click(_record_audio)
  display(button)
else:
  #button = widgets.Button(description="Upload Voice File")
  #button.on_click(_upload_audio)
  _upload_audio("")
```

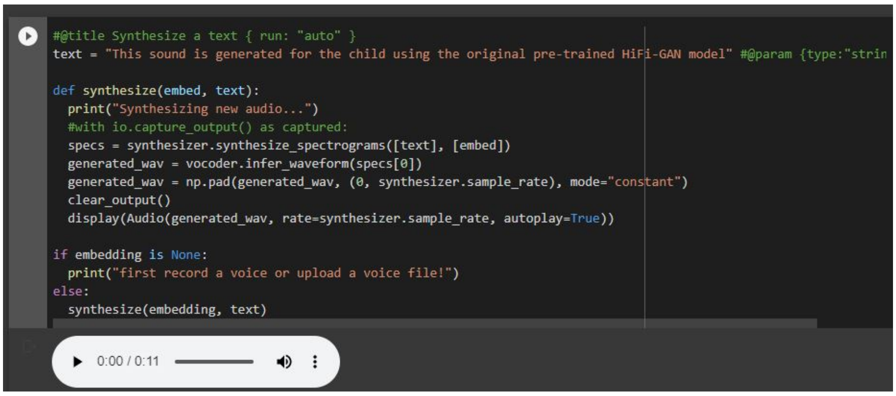
Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Ple

```
Saving myst_002013_2014-03-11_11-14-16_LS_2.1_049.wav to myst_002013_2014-03-11_11-14-16_LS_2.1_049.wav
```

0:00 / 0:10

Figure 27: Load pre-trained HiFi-GAN vocoder model and upload an audio file to synthesize

As shown in the figure below, enter the desired text to generate the synthesized voice. Additionally, save the input and the synthesized file for evaluation

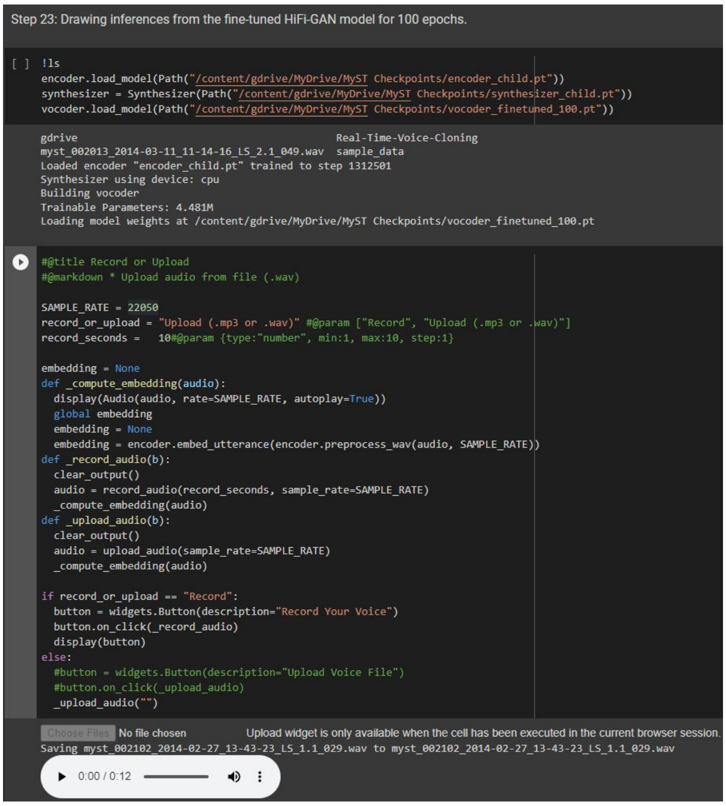


```
#@title Synthesize a text { run: "auto" }
text = "This sound is generated for the child using the original pre-trained HiFi-GAN model" #@param {type:"strin

def synthesize(embed, text):
  print("Synthesizing new audio...")
  #with io.capture_output() as captured:
  specs = synthesizer.synthesize_spectrograms([text], [embed])
  generated_wav = vocoder.infer_waveform(specs[0])
  generated_wav = np.pad(generated_wav, (0, synthesizer.sample_rate), mode="constant")
  clear_output()
  display(Audio(generated_wav, rate=synthesizer.sample_rate, autoplay=True))

if embedding is None:
  print("first record a voice or upload a voice file!")
else:
  synthesize(embedding, text)
```


Figure 28: Enter desired text to generate the synthesized voice

7.3 Synthesize voice using fine-tuned HiFi-GAN vocoder for 100 epochs in the proposed model

The following figures demonstrate the generation of synthesized voice for the fine-tuned HiFi-GAN vocoder model for 100 epochs.



Step 23: Drawing inferences from the fine-tuned HiFi-GAN model for 100 epochs.

```
!ls
encoder.load_model(Path("/content/gdrive/MyDrive/MyST Checkpoints/encoder_child.pt"))
synthesizer = Synthesizer(Path("/content/gdrive/MyDrive/MyST Checkpoints/synthesizer_child.pt"))
vocoder.load_model(Path("/content/gdrive/MyDrive/MyST Checkpoints/vocoder_finetuned_100.pt"))
```

```
gdrive                                         Real-Time-Voice-Cloning
myst_002013_2014-03-11_11-14-16_LS_2.1_049.wav  sample_data
Loaded encoder "encoder_child.pt" trained to step 1312501
Synthesizer using device: cpu
Building vocoder
Trainable Parameters: 4.481M
Loading model weights at /content/gdrive/MyDrive/MyST Checkpoints/vocoder_finetuned_100.pt
```

```
#@title Record or Upload
#@markdown * Upload audio from file (.wav)

SAMPLE_RATE = 22050
record_or_upload = "Upload (.mp3 or .wav)" #@param ["Record", "Upload (.mp3 or .wav)"]
record_seconds =   10#@param {type:"number", min:1, max:10, step:1}

embedding = None
def _compute_embedding(audio):
  display(Audio(audio, rate=SAMPLE_RATE, autoplay=True))
  global embedding
  embedding = None
  embedding = encoder.embed_utterance(encoder.preprocess_wav(audio, SAMPLE_RATE))
def _record_audio(b):
  clear_output()
  audio = record_audio(record_seconds, sample_rate=SAMPLE_RATE)
  _compute_embedding(audio)
def _upload_audio(b):
  clear_output()
  audio = upload_audio(sample_rate=SAMPLE_RATE)
  _compute_embedding(audio)

if record_or_upload == "Record":
  button = widgets.Button(description="Record Your Voice")
  button.on_click(_record_audio)
  display(button)
else:
  #button = widgets.Button(description="Upload Voice File")
  #button.on_click(_upload_audio)
  _upload_audio("")
```


Figure 29: Load fine-tuned HiFi-GAN vocoder model (100 epochs) and upload an audio file to synthesize

As shown in the figure below, enter the desired text to generate the synthesized voice. Additionally, save the input and the synthesized file for evaluation

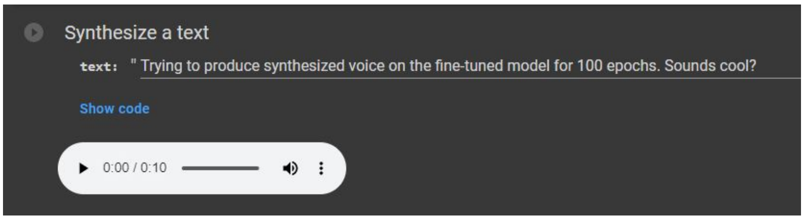


Figure 30: Enter desired text to generate the synthesized voice

7.4 Synthesize voice using fine-tuned HiFi-GAN vocoder for 200 epochs in the proposed model

The following figures demonstrate the generation of synthesized voice for the fine-tuned HiFi-GAN vocoder model for 200 epochs.

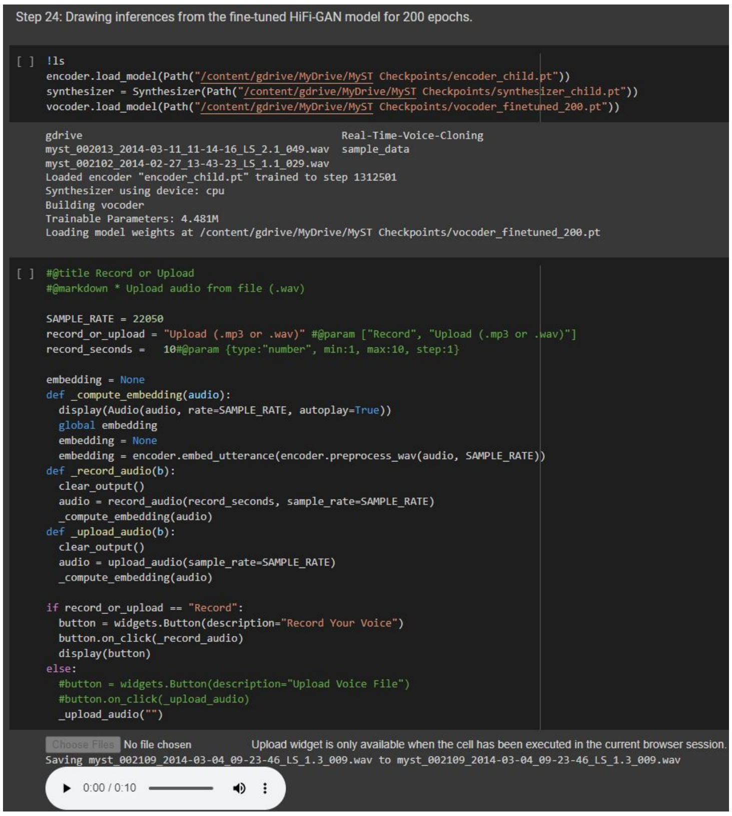


Figure 31: Load fine-tuned HiFi-GAN vocoder model (200 epochs) and upload an audio file to synthesize

As shown in the figure below, enter the desired text to generate the synthesized voice. Additionally, save the input and the synthesized file for evaluation

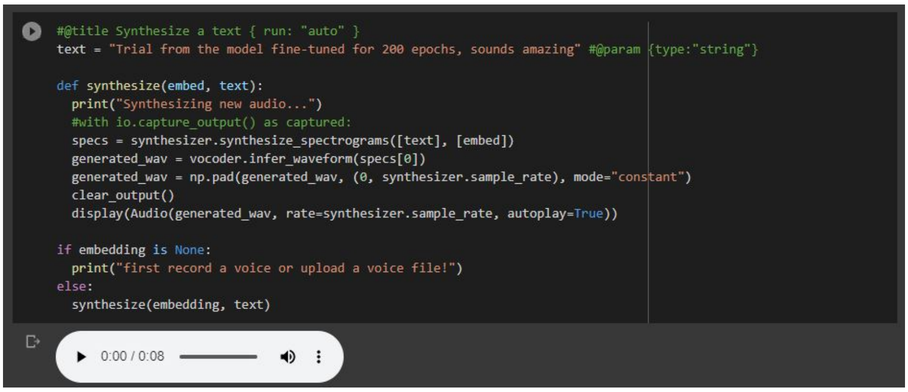


Figure 32: Enter desired text to generate the synthesized voice

8 Evaluation: Generate MOSNet scores

This section describes the generation of MOSNet scores for the original and the synthesized audio files produced in 7. The MOSNet for this research is implemented from [10]. The following figure demonstrates the generation of MOSNet scores for this research.

8.1 Import/ Install repository for implementing and generating MOSNet scores

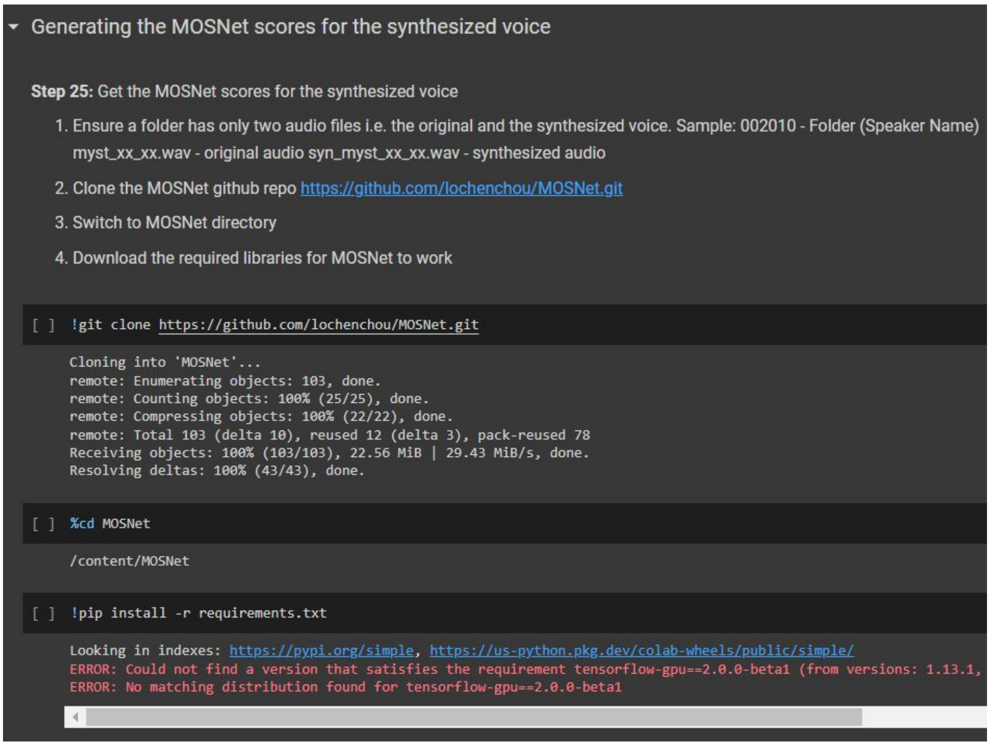


Figure 33: Cloning the git repository, change directory, and install requirements

[10] MOSNet Implementation `https://github.com/lochenchou/MOSNet.git`

8.2 Get Valid Directories for MOSNet to run

```
[ ] """Loop through each speaker folders to get their MOSNet scores
    using the MOSNet github repo code
    """
    root_directory = "/content/gdrive/MyDrive/Zeba_TTS/Synthesized_Voices"
    previous_directory = ""

    for dir_path, sub_directories, files in os.walk(root_directory):
      for filename in files:
        if filename.endswith(".wav") and ("Pretrained" in dir_path):
          if previous_directory == "" or previous_directory != dir_path:
            previous_directory = dir_path
            print("Valid directories :", previous_directory)

    Valid directories : /content/gdrive/MyDrive/Zeba_TTS/Synthesized_Voices/Pretrained_0/002013_Pretrained
    Valid directories : /content/gdrive/MyDrive/Zeba_TTS/Synthesized_Voices/Pretrained_0/002033_Pretrained
    Valid directories : /content/gdrive/MyDrive/Zeba_TTS/Synthesized_Voices/Pretrained_100/002026_Pretrained
    Valid directories : /content/gdrive/MyDrive/Zeba_TTS/Synthesized_Voices/Pretrained_100/002030_Pretrained
    Valid directories : /content/gdrive/MyDrive/Zeba_TTS/Synthesized_Voices/Pretrained_100/002102_Pretrained
    Valid directories : /content/gdrive/MyDrive/Zeba_TTS/Synthesized_Voices/Pretrained_200/002109_Pretrained
    Valid directories : /content/gdrive/MyDrive/Zeba_TTS/Synthesized_Voices/Pretrained_200/002113_Pretrained
    Valid directories : /content/gdrive/MyDrive/Zeba_TTS/Synthesized_Voices/Pretrained_200/002269_Pretrained
    Valid directories : /content/gdrive/MyDrive/Zeba_TTS/Synthesized_Voices/Pretrained_200/002274_Pretrained
```

Figure 34: Get Valid Directories for MOSNet

8.3 Generate MOSNet scores for synthesized voices from pre-trained HiFi-GAN vocoder

```
[ ] # HiFi-GAN purely pre-trained model
    !python ./custom_test.py \
    --rootdir '/content/gdrive/MyDrive/Zeba_TTS/Synthesized_Voices/Pretrained_0/002013_Pretrained'
    print("\n")
    !python ./custom_test.py \
    --rootdir '/content/gdrive/MyDrive/Zeba_TTS/Synthesized_Voices/Pretrained_0/002033_Pretrained'

    Loading model weights
    CNN_BLSTM init
    Start evaluating 2 waveforms...
    100% 2/2 [00:04<00:00,  2.07s/it]
    Average: 2.3395


    Loading model weights
    CNN_BLSTM init
    Start evaluating 2 waveforms...
    100% 2/2 [00:03<00:00,  1.99s/it]
    Average: 2.3425000000000002
```

Figure 35: MOSNet scores for synthesized voices from pre-trained HiFi-GAN vocoder

8.4 Generate MOSNet scores for synthesized voices from fine-tuned HiFi-GAN vocoder (100 epochs)

```
[ ] # HiFi-GAN fine-tuned model for 100 epochs
    !python ./custom_test.py \
    --rootdir '/content/gdrive/MyDrive/Zeba_TTS/Synthesized_Voices/Pretrained_100/002026_Pretrained'

    print("\n")

    !python ./custom_test.py \
    --rootdir '/content/gdrive/MyDrive/Zeba_TTS/Synthesized_Voices/Pretrained_100/002030_Pretrained'

    print("\n")

    !python ./custom_test.py \
    --rootdir '/content/gdrive/MyDrive/Zeba_TTS/Synthesized_Voices/Pretrained_100/002102_Pretrained'
```

```
Loading model weights
CNN_BLSTM init
Start evaluating 2 waveforms...
100% 2/2 [00:03<00:00,  1.71s/it]
Average: 2.904


Loading model weights
CNN_BLSTM init
Start evaluating 2 waveforms...
100% 2/2 [00:04<00:00,  2.08s/it]
Average: 2.6870000000000003


Loading model weights
CNN_BLSTM init
Start evaluating 2 waveforms...
100% 2/2 [00:03<00:00,  1.93s/it]
Average: 2.6405
```

Figure 36: MOSNet scores for synthesized voices from fine-tuned HiFi-GAN vocoder (100 epochs)

8.5 Generate MOSNet scores for synthesized voices from fine-tuned HiFi-GAN vocoder (200 epochs)

```
[ ]  # HiFi-GAN fine-tuned model for 200 epochs
     !python ./custom_test.py \
     --rootdir '/content/gdrive/MyDrive/Zeba_TTS/Synthesized_Voices/Pretrained_200/002109_Pretrained'

     print("\n")

     !python ./custom_test.py \
     --rootdir '/content/gdrive/MyDrive/Zeba_TTS/Synthesized_Voices/Pretrained_200/002113_Pretrained'

     print("\n")


     !python ./custom_test.py \
     --rootdir '/content/gdrive/MyDrive/Zeba_TTS/Synthesized_Voices/Pretrained_200/002269_Pretrained'

     print("\n")


     !python ./custom_test.py \
     --rootdir '/content/gdrive/MyDrive/Zeba_TTS/Synthesized_Voices/Pretrained_200/002274_Pretrained'

     Loading model weights
     CNN_BLSTM init
     Start evaluating 2 waveforms...
     100% 2/2 [00:03<00:00,  1.77s/it]
     Average: 2.928


     Loading model weights
     CNN_BLSTM init
     Start evaluating 2 waveforms...
     100% 2/2 [00:03<00:00,  1.75s/it]
     Average: 2.9215


     Loading model weights
     CNN_BLSTM init
     Start evaluating 2 waveforms...
     100% 2/2 [00:03<00:00,  1.76s/it]
     Average: 2.721


     Loading model weights
     CNN_BLSTM init
     Start evaluating 2 waveforms...
     100% 2/2 [00:03<00:00,  2.00s/it]
     Average: 2.3899999999999997
```

Figure 37: MOSNet scores for synthesized voices from fine-tuned HiFi-GAN vocoder (200 epochs)

9 Evaluation: Plot Visualization

This section describes the generation of mel spectrograms and waveforms for the original and synthesized voices produced in 7. The following figures demonstrate the visualizations done in this research.

9.1 Plot Mel Spectrograms and Waveforms

Plot visualizations for inferences

Step 27: Plotting spectrograms and waveforms for original and synthesized voices

1. Ensure that the original and synthesized voice are the only files present in the folder.
2. For ease of inference, the code follows the following directory structure: Synthesized Voices Output > Pretained_0 or Pretrained_100 or Pretrained_200 > Speaker > myst_xx_xx (original audio file) syn_mys_xx_xx (synthesized audio file)

```
[ ] """Loop through each wav file in different speaker folders and
    save their respective mel spectrogram and waveform using the
    Librosa library
    """
    root_directory = "/content/gdrive/MyDrive/Zeba_TTS/Synthesized_Voices"
    for dir_path, sub_directories, files in os.walk(root_directory):
      for filename in files:
        if filename.endswith(".wav"):
          audio_file = os.path.join(dir_path,filename)
          audio_data, sr = librosa.load(audio_file)

          # Mel-frequency spectrogram generation for each wav file
          melspectrum = librosa.feature.melspectrogram(y=audio_data, sr=sr, hop_length= 512, window='hann', n_mels=80)
          fig, ax = plt.subplots()
          S_dB = librosa.power_to_db(melspectrum, ref=np.max)
          img = librosa.display.specshow(S_dB, x_axis='time',
                                  y_axis='mel', sr=sr,
                                  ax=ax)
          fig.colorbar(img, ax=ax, format='%+2.0f dB')
          fig_title = 'Mel-frequency spectrgoram of ' + filename
          pltFileName = dir_path + "/Mel_" + filename.split(".wav")[0] + ".png"
          ax.set(title=fig_title)
          plt.show()
          plt.savefig(pltFileName)
          print("Mel-frequency spectrogram saved in : ", pltFileName)

          # Waveform generation for each wav file
          y, sr = librosa.load(audio_file, duration = 4)
          fig_title = 'Waveform of ' + filename
          pltFileName = dir_path + "/Wav_" + filename.split(".wav")[0] + ".png"
          plt.plot(y);
          plt.title(fig_title);
          plt.xlabel('Time in seconds (samples)');
          plt.ylabel('Amplitude');
          plt.show()
          plt.savefig(pltFileName)
          print("Waveform saved in : ", pltFileName)
```

Figure 38: Generate and Store Mel Spectrograms and Waveforms

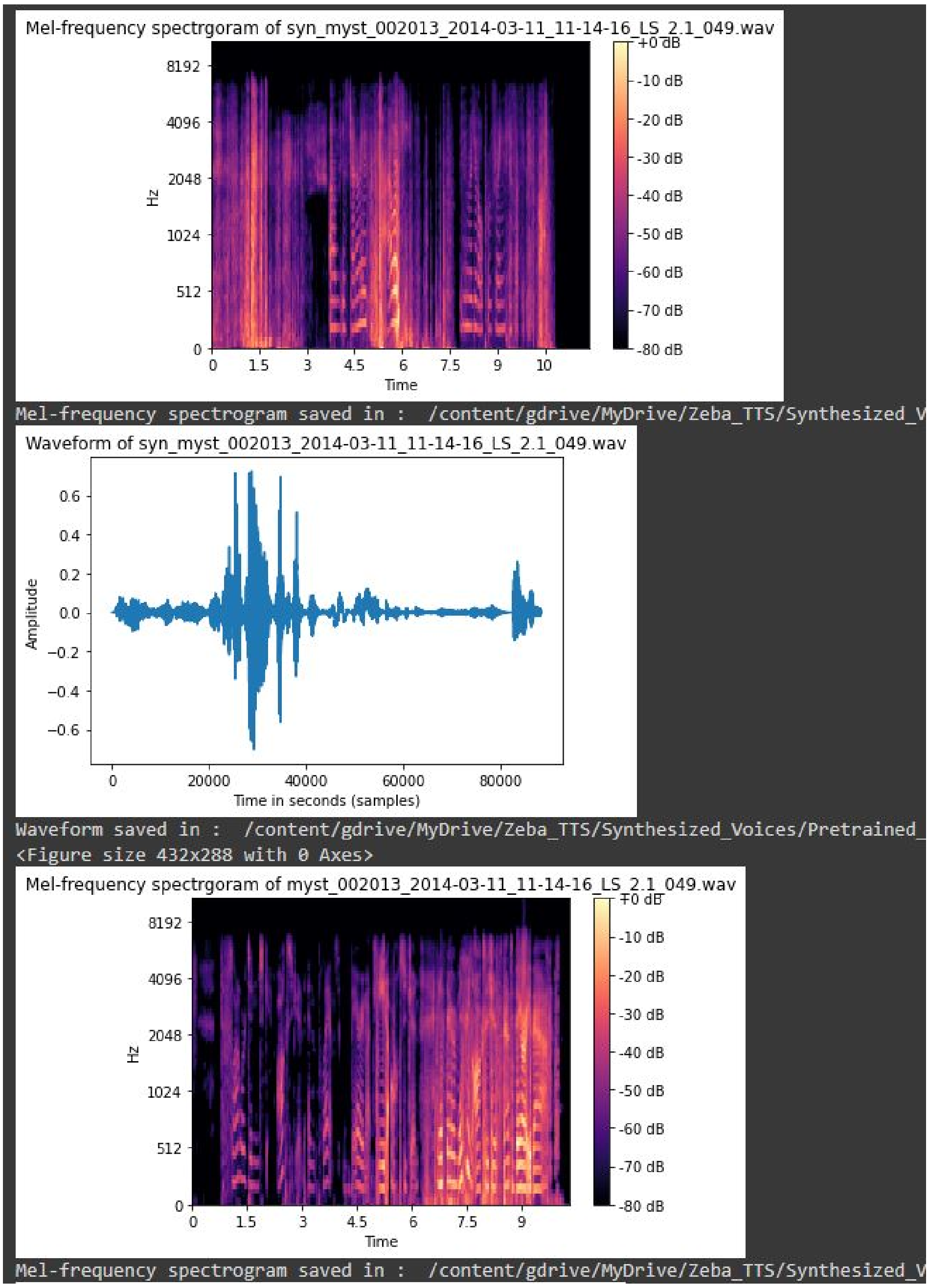


Figure 39: Produced Mel Spectrograms and Waveforms

References

Jain, R., Yiwere, M. Y., Bigioi, D., Corcoran, P. and Cucu, H. (2022). A text-to-speech pipeline, evaluation methodology, and initial fine-tuning results for child speech synthesis, *IEEE Access* **10**: 47628–47642.