Name: Niharika Amritkar **Email: a.niharika146@gmail.com**

Assignment 2: Fine-tuning Text-to-Speech (TTS) Models for English Technical Speech and Regional Languages

Objective:

The purpose of this assignment is to fine-tune two text-to-speech (TTS) models. One model will be optimized to handle technical jargon commonly used in English technical interviews, such as "API," "CUDA," and "TTS." The other model will be fine-tuned for a regional language of your choice. You will also explore ways to optimize the model for fast inference, and investigate techniques such as quantization to reduce model size without compromising performance.

Contents

- ## INTRODUCTION:

Text-to-Speech (TTS) is a technology which can convert any given text as an input into an audio which uses natural voices. It is primarily based on deep learning techniques which helps in converting any text input into audios or speech. It has widespread applications like one of the customer service sectors. The major application can be seen for helping visually impaired people or people having reading disability to understand the texts. They are implemented in audiobooks to help such people get access to the text and book contents.

Since it can generate audios, they are also seen in voice assistance like the famous Apple Siri, Amazon Alexa and Google Assistant. They have smart applications like understanding the pronunciations of words and smart responses in a particular system.

Importance of Fine-Tuning: there are various reasons for fine-tuning the model is beneficial. Firstly, it helps in understanding the language and accent adaptations as different regions have different accents. Secondly, it has the flexibility of selecting custom voice adaptation. Hence one can give any voice and the model can generate the audio in that particular voice. Fine- Tuning also helps in building models that are domain specific like the one we have is specific to technical terms. There may be models specific to other industries like medical, telecommunication, etc.

- ## METHODOLOGY:

TASK1:

Model selection: choosing the right model along with the right dataset is crucial for the success of the model's functionalities. Therefore, for task1, I chose SpeechT5 model. This model is versatile and can handle variety of text-to-speech tasks.

Dataset Preparation: since the main aim of the model is to be able to pronounce technical terms, it was crucial to find or prepare a dataset having such technical terms. I found one dataset on Hugging Face platform which included such terms. It was apt for the problem statement and contained a good amount of training data of about 9.95K rows. It has audios and transcription for the model training.

Fine-tuning:

```
from datasets import load_dataset, Audio
dataset = load_dataset("Yassmen/TTS_English_Technical_data", split="train")
dataset
```

```
README.md:   0%|          | 0.00/333 [00:00<?, ?B/s]
train-00000-of-00004.parquet:   0%|          | 0.00/469M [00:00<?, ?B/s]
train-00001-of-00004.parquet:   0%|          | 0.00/466M [00:00<?, ?B/s]
train-00002-of-00004.parquet:   0%|          | 0.00/468M [00:00<?, ?B/s]
train-00003-of-00004.parquet:   0%|          | 0.00/541M [00:00<?, ?B/s]
Generating train split:   0%|          | 0/9951 [00:00<?, ? examples/s]
Dataset({
    features: ['audio', 'transcription'],
    num_rows: 9951
})
```

```
[ ] dataset = dataset.cast_column("audio", Audio(sampling_rate=16000))
```

```
[ ] from transformers import SpeechT5Processor

    checkpoint = "microsoft/speecht5_tts"
    processor = SpeechT5Processor.from_pretrained(checkpoint)
```

```
[ ] tokenizer = processor.tokenizer
```

Let's normalize the dataset, create a column called "normalized_text"

```
def extract_all_chars(batch):
    all_text = " ".join(batch["transcription"])
    vocab = list(set(all_text))
    return {"vocab": [vocab], "all_text": [all_text]}


vocabs = dataset.map(
    extract_all_chars,
    batched=True,
    batch_size=-1,
    keep_in_memory=True,
    remove_columns=dataset.column_names,
)

dataset_vocab = set(vocabs["vocab"][0])
tokenizer_vocab = {k for k, _ in tokenizer.get_vocab().items()}
```

```
Map:   0%|          | 0/621 [00:00<?, ? examples/s]
```

```
[ ] dataset_vocab - tokenizer_vocab
```

```
{'\n', ' ', '%', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '@'}
```

```
import re

def normalize_text(text):
    # Convert to lowercase
    text = text.lower()

    # Remove punctuation (except apostrophes)
    text = re.sub(r'[^\w\s\']', '', text)

    # Remove extra whitespace
    text = ' '.join(text.split())

    return text

# Define a function to add the normalized_text column
def add_normalized_text(example):
    example['normalized_text'] = normalize_text(example['transcription'])
    return example

# Apply the function to the dataset
dataset = dataset.map(add_normalized_text)

# Print the first few examples to verify
print(dataset[2:5])
```

```
Map:   0%|          | 0/621 [00:00<?, ? examples/s]
{'audio': [{'path': 'YT2-9833.wav', 'array': array([-0.00598094,  0.03031809, -0.03093819, ..., -0.0726028 ,
       -0.07300073, -0.07294128]), 'sampling_rate': 16000}, {'path': 'YT2-7339.wav', 'array': array([0.00041708, 0.0048018 , 0.01064588, ..., 0.0
```

```
def extract_all_chars(batch):
    all_text = " ".join(batch["normalized_text"])
    vocab = list(set(all_text))
    return {"vocab": [vocab], "all_text": [all_text]}


vocabs = dataset.map(
    extract_all_chars,
    batched=True,
    batch_size=-1,
    keep_in_memory=True,
    remove_columns=dataset.column_names,
)

dataset_vocab = set(vocabs["vocab"][0])
tokenizer_vocab = {k for k, _ in tokenizer.get_vocab().items()}
```

```
Map:   0%|          | 0/621 [00:00<?, ? examples/s]
```

+ Code + Text

```
dataset_vocab - tokenizer_vocab
```

```
{' ', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'}
```

```python
replacements = [
    ('API', 'A-P-I'),       # API remains as API
    ('OAuth', 'o-auth'),  # OAuth to oauth
    ('REST', 'rest'),     # REST to rest
    ('SQL', 'S-Q-L'),       # SQL remains as SQL
    ('CUDA', 'koodaa'),  # CUDA to koodaa
    ('TTS', 'T-T-S'),       # TTS remains as TTS
    ('HTTP', 'H T T P'),    # HTTP remains as HTTP
    ('JSON', 'J-SON'),    # JSON remains as JSON
    ('XML', 'X-M-L'),       # XML remains as XML
    ('SSH', 'S-S-H'),       # SSH remains as SSH
    ('SSL', 'S-S-L'),       # SSL remains as SSL
    ('DNS', 'D-N-S'),       # DNS remains as DNS
    ('GPU', 'G-P-U'),       # GPU remains as GPU
    ('CPU', 'C-P-U'),       # CPU remains as CPU
    ('FTP', 'F-T-P'),       # FTP remains as FTP
    ('VPN', 'V-P-N'),       # VPN remains as VPN
    ('TCP/IP', 'T-C-P/I-P'),# TCP/IP remains as TCP/IP
    ('UI', 'U-I'),         # UI remains as UI
    ('UX', 'U-X'),         # UX remains as UX
    ('IDE', 'I-D-E'),       # IDE remains as IDE
    ('Git', 'Git'),       # Git remains as Git
    ('Docker', 'Docker'),# Docker remains as Docker
    ('Kubernetes', 'kubernetes'), # Kubernetes to kubernetes
    ('PyTorch', 'pytorch'), # PyTorch to pytorch
    ('TensorFlow', 'tensorflow'), # TensorFlow to tensorflow
    ('GraphQL', 'graphql'), # GraphQL to graphql
    ('CI/CD', 'ci-cd'),    # CI/CD to ci-cd
    ('JWT', 'J-W-T'),        # JWT remains as JWT
    ('WebSocket', 'websocket'), # WebSocket to websocket
    ('0', 'zero'),
```

```python
def cleanup_text(inputs):
    for src, dst in replacements:
        inputs["normalized_text"] = inputs["normalized_text"].replace(src, dst)
    return inputs

dataset = dataset.map(cleanup_text)
```

```
Map:   0%|          | 0/621 [00:00<?, ? examples/s]
```

```python
import os
import torch
from speechbrain.pretrained import EncoderClassifier

spk_model_name = "speechbrain/spkrec-xvect-voxceleb"

device = "cuda" if torch.cuda.is_available() else "cpu"
speaker_model = EncoderClassifier.from_hparams(
    source=spk_model_name,
    run_opts={"device": device},
    savedir=os.path.join("/tmp", spk_model_name),
)
```

```
def create_speaker_embedding(waveform):
    with torch.no_grad():
        speaker_embeddings = speaker_model.encode_batch(torch.tensor(waveform))
        speaker_embeddings = torch.nn.functional.normalize(speaker_embeddings, dim=2)
        speaker_embeddings = speaker_embeddings.squeeze().cpu().numpy()
    return speaker_embeddings
```

```
hyperparams.yaml:   0%|          | 0.00/2.04k [00:00<?, ?B/s]
embedding_model.ckpt:   0%|          | 0.00/16.9M [00:00<?, ?B/s]
mean_var_norm_emb.ckpt:   0%|          | 0.00/3.20k [00:00<?, ?B/s]
classifier.ckpt:   0%|          | 0.00/15.9M [00:00<?, ?B/s]
label_encoder.txt:   0%|          | 0.00/129k [00:00<?, ?B/s]
/opt/conda/lib/python3.10/site-packages/speechbrain/utils/checkpoints.py:147: FutureWarning: You are using `torch.load` with `weights_only=False` (the current default va
  torch.load(path, map_location=device), strict=False
/opt/conda/lib/python3.10/site-packages/speechbrain/processing/features.py:1215: FutureWarning: You are using `torch.load` with `weights_only=False` (the current default
  stats = torch.load(path, map_location=device)
```

```
[ ] def prepare_dataset(example):
        audio = example["audio"]

        example = processor(
            text=example["normalized_text"],
            audio_target=audio["array"],
            sampling_rate=audio["sampling_rate"],
            return_attention_mask=False,
        )

        # strip off the batch dimension
        example["labels"] = example["labels"][0]

        # use SpeechBrain to obtain x-vector
        example["speaker_embeddings"] = create_speaker_embedding(audio["array"])

        return example
```

```
[ ] processed_example = prepare_dataset(dataset[0])
    list(processed_example.keys())
```

```
['input_ids', 'labels', 'speaker_embeddings']
```

+ Code + Text

```
[ ] processed_example["speaker_embeddings"].shape
```

```
(512,)
```

```
[ ] dataset = dataset.map(prepare_dataset, remove_columns=dataset.column_names)
```

```
Map:   0%|          | 0/621 [00:00<?, ? examples/s]
```

```
[ ] def is_not_too_long(input_ids):
        input_length = len(input_ids)
        return input_length < 200

    dataset = dataset.filter(is_not_too_long, input_columns=["input_ids"])
    len(dataset)
```

```
Filter:   0%|          | 0/621 [00:00<?, ? examples/s]
621
```

```
[ ] dataset = dataset.train_test_split(test_size=0.1)
```

```
    def __call__(
        self, features: List[Dict[str, Union[List[int], torch.Tensor]]]
    ) -> Dict[str, torch.Tensor]:
        input_ids = [{"input_ids": feature["input_ids"]} for feature in features]
        label_features = [{"input_values": feature["labels"]} for feature in features]
        speaker_features = [feature["speaker_embeddings"] for feature in features]

        # collate the inputs and targets into a batch
        batch = processor.pad(
            input_ids=input_ids, labels=label_features, return_tensors="pt"
        )

        # replace padding with -100 to ignore loss correctly
        batch["labels"] = batch["labels"].masked_fill(
            batch.decoder_attention_mask.unsqueeze(-1).ne(1), -100
        )

        # not used during fine-tuning
        del batch["decoder_attention_mask"]

        # round down target lengths to multiple of reduction factor
        if model.config.reduction_factor > 1:
            target_lengths = torch.tensor(
                [len(feature["input_values"]) for feature in label_features]
            )
            target_lengths = target_lengths.new(
                [
                    length - length % model.config.reduction_factor
                    for length in target_lengths
                ]
            )
            max_length = max(target_lengths)
            batch["labels"] = batch["labels"][:, :max_length]

        # also add in the speaker embeddings
        batch["speaker_embeddings"] = torch.tensor(speaker_features)

        return batch
```

```
[ ] data_collator = TTSDataCollatorWithPadding(processor=processor)
```

```
[ ] from transformers import SpeechT5ForTextToSpeech

    model = SpeechT5ForTextToSpeech.from_pretrained(checkpoint)
```

```
config.json:   0%|          | 0.00/2.06k [00:00<?, ?B/s]
pytorch_model.bin:   0%|          | 0.00/585M [00:00<?, ?B/s]
```

```
from functools import partial

# disable cache during training since it's incompatible with gradient checkpointing
model.config.use_cache = False

# set language and task for generation and re-enable cache
model.generate = partial(model.generate, use_cache=True)
```

```python
from transformers import Seq2SeqTrainingArguments

training_args = Seq2SeqTrainingArguments(
    output_dir="./speecht5_finetuned_technical_data",  # change to a repo name of your choice
    per_device_train_batch_size=16,
    gradient_accumulation_steps=4,
    learning_rate=1e-5,
    warmup_steps=500,
    max_steps=4000,
    gradient_checkpointing=True,
    fp16=True,
    eval_strategy="steps",
    per_device_eval_batch_size=8,
    save_steps=1000,
    eval_steps=1000,
    logging_steps=25,
    report_to=["tensorboard"],
    load_best_model_at_end=True,
    greater_is_better=False,
    label_names=["labels"],
```

```python
    fp16=True,
    evaluation_strategy="steps",
    per_device_eval_batch_size=2,
    save_steps=100,
    eval_steps=100,
    logging_steps=25,
    report_to=["tensorboard"],
    load_best_model_at_end=True,
    greater_is_better=False,
    label_names=["labels"],
    push_to_hub=True,
)
```

```
/opt/conda/lib/python3.10/site-packages/transformers/training_args.py:1545: FutureWarning: `evaluation_strategy` is deprecated and will be removed in version 4.
  warnings.warn(
```

```python
[ ] from transformers import Seq2SeqTrainer

trainer = Seq2SeqTrainer(
    args=training_args,
    model=model,
    train_dataset=dataset["train"],
    eval_dataset=dataset["test"],
    data_collator=data_collator,
    tokenizer=processor,
)
```

```
/opt/conda/lib/python3.10/site-packages/accelerate/accelerator.py:494: FutureWarning: `torch.cuda.amp.GradScaler(args...)` is deprecated. Please use `torch.amp.
  self.scaler = torch.cuda.amp.GradScaler(**kwargs)
max_steps is given, it will override any value given in num_train_epochs
```

```python
trainer.train()
```

TASK 2:

Model selection: for the regional language fine-tuning, SpeechT5 model has been selected. This was selected because of its versatility and model flexibility to get trained on custom dataset.

Dataset selection: For this task, regional language “Hindi” is selected and therefore a Hindi language dataset has been selected. The dataset is available on Hugging face and is one of the Common Voice datasets. It consisted of training data with a good number of rows and approximately 3 speakers. The Dataset contains Hindi language sentences and hence the model has been trained as per the pronunciations of the Hindi letter and vocabulary. Since It was a challenge to find a pretrained Hindi speaker for the model, therefore, the model was trained on English speaker and the Phenomes were edited as per the requirements and needs of the language pronunciations.

Fine-Tuning:

```
from transformers import SpeechT5Processor, SpeechT5ForTextToSpeech

processor = SpeechT5Processor.from_pretrained("microsoft/speecht5_tts")
model = SpeechT5ForTextToSpeech.from_pretrained("microsoft/speecht5_tts")
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_token.py:89: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
preprocessor_config.json: 100%  433/433 [00:00<00:00, 5.67kB/s]
tokenizer_config.json: 100%  232/232 [00:00<00:00, 3.85kB/s]
spm_char.model: 100%  238k/238k [00:00<00:00, 2.60MB/s]
added_tokens.json: 100%  40.0/40.0 [00:00<00:00, 1.03kB/s]
special_tokens_map.json: 100%  234/234 [00:00<00:00, 4.66kB/s]
config.json: 100%  2.06k/2.06k [00:00<00:00, 32.4kB/s]
pytorch_model.bin: 100%  585M/585M [00:07<00:00, 98.3MB/s]
```

```
from datasets import load_dataset, Audio

dataset = load_dataset(
    "mozilla-foundation/common_voice_11_0", 'hi', split='train'
)
```

```
common_voice_11_0.py: 100%  8.13k/8.13k [00:00<00:00, 290kB/s]
README.md: 100%  14.4k/14.4k [00:00<00:00, 722kB/s]
languages.py: 100%  3.44k/3.44k [00:00<00:00, 193kB/s]
release_stats.py: 100%  60.9k/60.9k [00:00<00:00, 2.64MB/s]
The repository for mozilla-foundation/common_voice_11_0 contains custom code which must be executed t
You can avoid this prompt in future by passing the argument `trust_remote_code=True`.

Do you wish to run the custom code? [y/N] y
n_shards.json: 100%  12.2k/12.2k [00:00<00:00, 767kB/s]
hi_train_0.tar: 100%  114M/114M [00:00<00:00, 168MB/s]
hi_dev_0.tar: 100%  61.9M/61.9M [00:00<00:00, 167MB/s]
hi_test_0.tar: 100%  92.2M/92.2M [00:00<00:00, 196MB/s]
hi_other_0.tar: 100%  113M/113M [00:00<00:00, 174MB/s]
hi_invalidated_0.tar: 100%  23.4M/23.4M [00:02<00:00, 181MB/s]
train.tsv: 100%  1.30M/1.30M [00:00<00:00, 57.1MB/s]
dev.tsv: 100%  627k/627k [00:00<00:00, 5.57MB/s]
test.tsv: 100%  824k/824k [00:00<00:00, 38.4MB/s]
other.tsv: 100%  1.04M/1.04M [00:00<00:00, 42.1MB/s]
invalidated.tsv: 100%  201k/201k [00:00<00:00, 11.0MB/s]
```

```
[ ] dataset = dataset.cast_column("audio", Audio(sampling_rate=16000))
```

Let's quickly check how many examples are in this dataset.

```
[ ] len(dataset)
```

```
4361
```

```
[ ] from transformers import SpeechT5Processor

    checkpoint = "microsoft/speecht5_tts"
    processor = SpeechT5Processor.from_pretrained(checkpoint)
```

```
[ ] tokenizer = processor.tokenizer
```

```
def extract_all_chars(batch):
    all_text = " ".join(batch["sentence"])
    vocab = list(set(all_text))
    return {"vocab": [vocab], "all_text": [all_text]}

vocabs = dataset.map(
    extract_all_chars,
    batched=True,
    batch_size=-1,
    keep_in_memory=True,
    remove_columns=dataset.column_names,
)

dataset_vocab = set(vocabs["vocab"][0])
tokenizer_vocab = {k for k,_ in tokenizer.get_vocab().items()}
```

```
Map: 100% 4361/4361 [00:00<00:00, 33774.94 examples/s]
```

```
[ ] def extract_all_chars(batch):
    all_text = " ".join(batch["sentence"])
    vocab = list(set(all_text))
    return {"vocab": [vocab], "all_text": [all_text]}

# Get column names from the 'train' split (assuming it exists)
column_names = dataset.column_names

vocabs = dataset.map(
    extract_all_chars,
    batched=True,
    batch_size=-1,
    keep_in_memory=True,
    remove_columns=column_names,  # Remove columns present in the splits
)
```

Map: 100% 4361/4361 [00:00<00:00, 39682.91 examples/s]

Now we have two sets of characters, one with the vocabulary from the dataset and one with the vocabula
difference between these sets, we find the characters that are in the dataset but not in the tokenizer.

```
dataset_vocab - tokenizer_vocab
```

```
{' ',
 '&',
 '।',
 'ँ',
 'ं',
 'ः',
 'अ',
 'आ',
 'इ',
 'ई',
 'उ',
 'ऊ',
 'ऋ',
 'ए',
 'ऐ',
 'ऑ',
 'ओ',
 'औ',
 'क',
 'ख',
 'ग',
 'घ',
 'च',
 'छ',
 'ज',
 'झ',
 'ञ',
 'ट',
 'ठ',
 'ड',
 'ढ',
```

```
replacements = [
    ('अ', 'a'),      # Hindi 'अ' to English 'a'
    ('आ', 'aa'),      # Hindi 'आ' to English 'aa'
    ('इ', 'i'),      # Hindi 'इ' to English 'i'
    ('ई', 'ee'),      # Hindi 'ई' to English 'ee'
    ('उ', 'u'),      # Hindi 'उ' to English 'u'
    ('ऊ', 'oo'),      # Hindi 'ऊ' to English 'oo'
    ('ए', 'e'),      # Hindi 'ए' to English 'e'
    ('ऐ', 'ai'),      # Hindi 'ऐ' to English 'ai'
    ('ओ', 'o'),      # Hindi 'ओ' to English 'o'
    ('औ', 'au'),      # Hindi 'औ' to English 'au'

    # Consonants
    ('क', 'k'),      # Hindi 'क' to English 'k'
    ('ख', 'kh'),      # Hindi 'ख' to English 'kh'
    ('ग', 'g'),      # Hindi 'ग' to English 'g'
    ('घ', 'gh'),      # Hindi 'घ' to English 'gh'
    ('च', 'ch'),      # Hindi 'च' to English 'ch'
    ('छ', 'chh'),      # Hindi 'छ' to English 'chh'
    ('ज', 'j'),      # Hindi 'ज' to English 'j'
    ('झ', 'jh'),      # Hindi 'झ' to English 'jh'
    ('ट', 't'),      # Hindi 'ट' to English 't'
    ('ठ', 'th'),      # Hindi 'ठ' to English 'th'
    ('ड', 'd'),      # Hindi 'ड' to English 'd'
    ('ढ', 'dh'),      # Hindi 'ढ' to English 'dh'
    ('ण', 'n'),      # Hindi 'ण' to English 'n'
    ('त', 't'),      # Hindi 'त' to English 't'
    ('थ', 'th'),      # Hindi 'थ' to English 'th'
```

```python
    ('स', 's'),      # Hindi 'स' to English 's'
    ('ह', 'h'),      # Hindi 'ह' to English 'h'

    # Special characters and sounds
    ('क्ष', 'ksh'),  # Hindi 'क्ष' to English 'ksh'
    ('त्र', 'tr'),   # Hindi 'त्र' to English 'tr'
    ('ज्ञ', 'gy'),   # Hindi 'ज्ञ' to English 'gy'
    ('ङ', 'ng'),     # Hindi 'ङ' to English 'ng'

    # Vowel diacritics (Matras)
    ('ा', 'aa'),      # Hindi 'ा' to English 'aa'
    ('ि', 'i'),       # Hindi 'ि' to English 'i'
    ('ी', 'ee'),      # Hindi 'ी' to English 'ee'
    ('ु', 'u'),       # Hindi 'ु' to English 'u'
    ('ू', 'oo'),      # Hindi 'ू' to English 'oo'
    ('े', 'e'),       # Hindi 'े' to English 'e'
    ('ै', 'ai'),      # Hindi 'ै' to English 'ai'
    ('ो', 'o'),       # Hindi 'ो' to English 'o'
    ('ौ', 'au'),      # Hindi 'ौ' to English 'au'
    ('ं', 'n'),       # Hindi 'ं' to English 'n' (Anusvara)
    ('ः', 'h'),       # Hindi 'ः' to English 'h' (Visarga)

    # Punctuation
    ('।', '.'),       # Hindi danda (।) to period (.)
    (',', ','),       # Comma remains comma
    ('?', '?'),       # Question mark remains question mark
]

def cleanup_text(inputs):
    for src, dst in replacements:
        inputs["sentence"] = inputs["sentence"].replace(src, dst)
    return inputs

dataset = dataset.map(cleanup_text)
```

```
Map: 100% 4361/4361 [00:01<00:00, 3018.33 examples/s]
```

Speakers

```python
from collections import defaultdict
speaker_counts = defaultdict(int)

for speaker_id in dataset["client_id"]:
    speaker_counts[speaker_id] += 1
```

```python
print(speaker_counts )
```

```
defaultdict(<class 'int'>, {'0f018a99663f33afbb7d38aee281fb1afcfd07f9e7acd00383f604e1e17c38d6ed8adf1bd2ccbf927a52c5adefb
```

By plotting a histogram we can get a sense of how much data there is for each speaker.

```python
import matplotlib.pyplot as plt

plt.figure()
plt.hist(speaker_counts.values(), bins=20)
plt.ylabel("Speakers")
plt.xlabel("Examples")
plt.show()
```

```python
import os
import numpy
import torch
from speechbrain.pretrained import EncoderClassifier
import torch
import torchaudio
import glob
import numpy
import argparse
from tqdm import tqdm
import torch.nn.functional as F

spk_model_name = "speechbrain/spkrec-xvect-voxceleb"

device = "cuda" if torch.cuda.is_available() else "cpu"
speaker_model = EncoderClassifier.from_hparams(
    source=spk_model_name,
    run_opts={"device": device},
    savedir=os.path.join("/tmp", spk_model_name),
)


def create_speaker_embedding(waveform):
    with torch.no_grad():
        speaker_embeddings = speaker_model.encode_batch(torch.tensor(waveform))
        speaker_embeddings = torch.nn.functional.normalize(speaker_embeddings, dim=2)
        speaker_embeddings = speaker_embeddings.squeeze().cpu().numpy()
    return speaker_embeddings
```


```python
device = "cuda" if torch.cuda.is_available() else "cpu"
speaker_model = EncoderClassifier.from_hparams(
    source=spk_model_name,
    run_opts={"device": device},
    savedir=os.path.join("/tmp", spk_model_name)
)


def create_speaker_embedding(waveform):
    with torch.no_grad():
        speaker_embeddings = speaker_model.encode_batch(torch.tensor(waveform))
        speaker_embeddings = torch.nn.functional.normalize(speaker_embeddings, dim=2)
        speaker_embeddings = speaker_embeddings.squeeze().cpu().numpy()
    return speaker_embeddings
```


```
<ipython-input-20-35c782c042c4>:4: UserWarning: Module 'speechbrain.pretrained' was deprecated, redirecting to 'speechbrain.inference'
  from speechbrain.pretrained import EncoderClassifier
hyperparams.yaml: 100% 2.04k/2.04k [00:00<00:00, 79.3kB/s]
/usr/local/lib/python3.10/dist-packages/speechbrain/utils/autocast.py:68: FutureWarning: `torch.cuda.amp.custom_fwd(args...)` is depre
  wrapped_fwd = torch.cuda.amp.custom_fwd(fwd, cast_inputs=cast_inputs)
embedding_model.ckpt: 100% 16.9M/16.9M [00:00<00:00, 39.9MB/s]
mean_var_norm_emb.ckpt: 100% 3.20k/3.20k [00:00<00:00, 62.8kB/s]
classifier.ckpt: 100% 15.9M/15.9M [00:00<00:00, 113MB/s]
label_encoder.txt: 100% 129k/129k [00:00<00:00, 2.76MB/s]
/usr/local/lib/python3.10/dist-packages/speechbrain/utils/checkpoints.py:194: FutureWarning: You are using `torch.load` with `weights_
  state_dict = torch.load(path, map_location=device)
/usr/local/lib/python3.10/dist-packages/speechbrain/processing/features.py:1311: FutureWarning: You are using `torch.load` with `weigh
```

```
[ ] def prepare_dataset(example):
        # load the audio data; if necessary, this resamples the audio to 16kHz
        audio = example["audio"]

        # feature extraction and tokenization
        example = processor(
            text=example["sentence"],
            audio_target=audio["array"],
            sampling_rate=audio["sampling_rate"],
            return_attention_mask=False,
        )

        # strip off the batch dimension
        example["labels"] = example["labels"][0]

        # use SpeechBrain to obtain x-vector
        example["speaker_embeddings"] = create_speaker_embedding(audio["array"])

        return example
```

Let's verify the processing is correct by looking at a single example:

```
if len(dataset) > 0:
    processed_example = prepare_dataset(dataset[0])
else:
    print("Dataset is empty. Please check the dataset loading or filtering steps.")
```

```
[ ] list(processed_example.keys())
```

```
['input_ids', 'labels', 'speaker_embeddings']
```

The tokens should decode into the original text, with `</s>` to mark the end of the sentence.

```
[ ] tokenizer.decode(processed_example["input_ids"])
```

```
'hmne uskaa jn<unk> mdin mnaayaa.</s>'
```

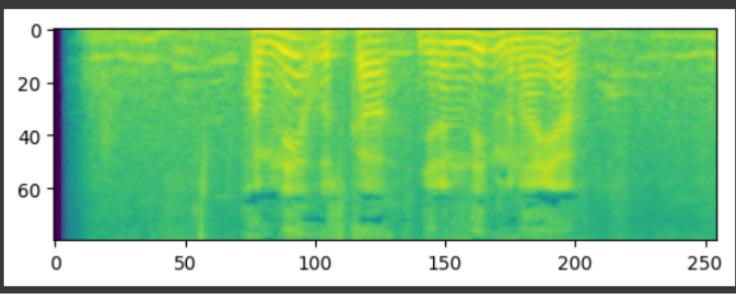
Speaker embeddings should be a 512-element vector:

```
[ ] processed_example["speaker_embeddings"].shape
```

```
(512,)
```

The labels should be a log-mel spectrogram with 80 mel bins.

```
[ ] import matplotlib.pyplot as plt
    plt.figure()
    plt.imshow(processed_example["labels"].T)
    plt.show()
```



```
[ ] def is_not_too_long(input_ids):
        input_length = len(input_ids)
        return input_length < 200

    dataset = dataset.filter(is_not_too_long, input_columns=["input_ids"])
```

Filter: 100% 4361/4361 [00:00<00:00, 19660.20 examples/s]

How many examples are left?

```
[ ] len(dataset)
```

```
4361
```

Train/test split

Create a basic train/test split. For our purposes, it's OK if the same speaker is part of both sets.

```
[ ] dataset = dataset.train_test_split(test_size=0.1)
```

What does the dataset look like now?

```
dataset
```

```
DatasetDict({
    train: Dataset({
        features: ['input_ids', 'labels', 'speaker_embeddings'],
        num_rows: 3924
    })
    test: Dataset({
        features: ['input_ids', 'labels', 'speaker_embeddings'],
        num_rows: 437
```

```
[ ] def is_not_too_long(input_ids):
        input_length = len(input_ids)
        return input_length < 200

    dataset = dataset.filter(is_not_too_long, input_columns=["input_ids"])
```

Filter: 100% 4361/4361 [00:00<00:00, 19660.20 examples/s]

How many examples are left?

```
[ ] len(dataset)
```

```
4361
```

Train/test split

Create a basic train/test split. For our purposes, it's OK if the same speaker is part of both sets.

```
[ ] dataset = dataset.train_test_split(test_size=0.1)
```

What does the dataset look like now?

```
dataset
```

```
DatasetDict({
    train: Dataset({
        features: ['input_ids', 'labels', 'speaker_embeddings'],
        num_rows: 3924
    })
    test: Dataset({
        features: ['input_ids', 'labels', 'speaker_embeddings'],
        num_rows: 437
```

```python
from dataclasses import dataclass
from typing import Any, Dict, List, Union

@dataclass
class TTSDataCollatorWithPadding:
    processor: Any

    def __call__(self, features: List[Dict[str, Union[List[int], torch.Tensor]]]) -> Dict[str, torch.Tensor]:

        input_ids = [{"input_ids": feature["input_ids"]} for feature in features]
        label_features = [{"input_values": feature["labels"]} for feature in features]
        speaker_features = [feature["speaker_embeddings"] for feature in features]

        # collate the inputs and targets into a batch
        batch = processor.pad(
            input_ids=input_ids,
            labels=label_features,
            return_tensors="pt",
        )

        # replace padding with -100 to ignore loss correctly
        batch["labels"] = batch["labels"].masked_fill(
            batch.decoder_attention_mask.unsqueeze(-1).ne(1), -100
        )

        # not used during fine-tuning
        del batch["decoder_attention_mask"]

        # round down target lengths to multiple of reduction factor
        if model.config.reduction_factor > 1:
            target_lengths = torch.tensor([
                len(feature["input_values"]) for feature in label_features
            ])
            target_lengths = target_lengths.new([
                length - length % model.config.reduction_factor for length in target_lengths
            ])
            max_length = max(target_lengths)
            batch["labels"] = batch["labels"][:, :max_length]
```


```python
        )

        # not used during fine-tuning
        del batch["decoder_attention_mask"]

        # round down target lengths to multiple of reduction factor
        if model.config.reduction_factor > 1:
            target_lengths = torch.tensor([
                len(feature["input_values"]) for feature in label_features
            ])
            target_lengths = target_lengths.new([
                length - length % model.config.reduction_factor for length in target_lengths
            ])
            max_length = max(target_lengths)
            batch["labels"] = batch["labels"][:, :max_length]
```


```python
        # also add in the speaker embeddings
        batch["speaker_embeddings"] = torch.tensor(speaker_features)

        return batch
```

```
[ ]  data_collator = TTSDataCollatorWithPadding(processor=processor)
```

Let's test the data collator.

```
[ ]  features = [
         dataset["train"][0],
         dataset["train"][1],
         dataset["train"][20],
     ]

     batch = data_collator(features)
```

```
from transformers import Seq2SeqTrainingArguments

training_args = Seq2SeqTrainingArguments(
    output_dir="./speecht5_tts_voxpopuli_hindi",  # change to a repo name of your choice
    per_device_train_batch_size=16,
    gradient_accumulation_steps=4,
    learning_rate=1e-5,
    warmup_steps=500,
    max_steps=4000,
    gradient_checkpointing=True,
    fp16=True,
    eval_strategy="steps",
    per_device_eval_batch_size=8,
    save_steps=1000,
    eval_steps=1000,
    logging_steps=25,
    report_to=["tensorboard"],
    load_best_model_at_end=True,
    greater_is_better=False,
    label_names=["labels"],
    push_to_hub=True,
)
```

Create the trainer object using the model, dataset, and data collator.

```
[47] from transformers import Seq2SeqTrainer

     trainer = Seq2SeqTrainer(
         args=training_args,
         model=model,
         train_dataset=dataset["train"],
         eval_dataset=dataset["test"],
         data_collator=data_collator,
         tokenizer=processor.tokenizer,
     )
```

```
trainer.train()
```

```
/usr/local/lib/python3.10/dist-packages/torch/utils/checkpoint.py:295: FutureWarning: `torch.cpu.amp.autocast(args...)` is deprecated. Please use `torch.amp.autocast('
  with torch.enable_grad(), device_autocast_ctx, torch.cpu.amp.autocast(**ctx.cpu_autocast_kwargs):  # type: ignore[attr-defined]
```

[2053/4000 1:06:41 < 1:03:18, 0.51 it/s, Epoch 33.37/66]

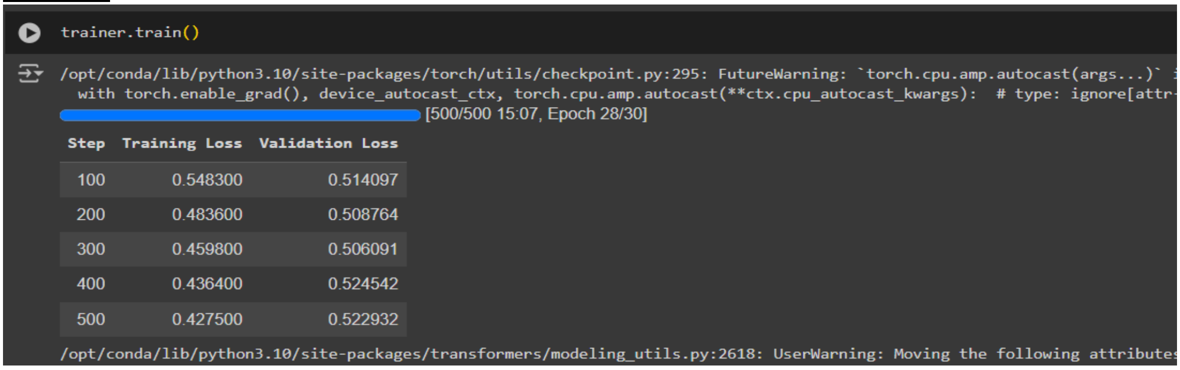
Step	Training Loss	Validation Loss
1000	2.073700	0.470535
2000	1.885600	0.456285

```
/usr/local/lib/python3.10/dist-packages/transformers/modeling_utils.py:2785: UserWarning: Moving the following attributes in the config to the generation config: {'max
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torch/utils/checkpoint.py:295: FutureWarning: `torch.cpu.amp.autocast(args...)` is deprecated. Please use `torch.amp.autocast('
  with torch.enable_grad(), device_autocast_ctx, torch.cpu.amp.autocast(**ctx.cpu_autocast_kwargs):  # type: ignore[attr-defined]
/usr/local/lib/python3.10/dist-packages/torch/utils/checkpoint.py:295: FutureWarning: `torch.cpu.amp.autocast(args...)` is deprecated. Please use `torch.amp.autocast('
  with torch.enable_grad(), device_autocast_ctx, torch.cpu.amp.autocast(**ctx.cpu_autocast_kwargs):  # type: ignore[attr-defined]
```

If we do one more `push_to_hub()` after training we can get a nice model card built for us. We simply have to set the appropriate keyword

- **<u>RESULTS:</u>**

<u>TASK1:</u>



```
trainer.train()

/opt/conda/lib/python3.10/site-packages/torch/utils/checkpoint.py:295: FutureWarning: `torch.cpu.amp.autocast(args...)`
  with torch.enable_grad(), device_autocast_ctx, torch.cpu.amp.autocast(**ctx.cpu_autocast_kwargs):  # type: ignore[attr
[500/500 15:07, Epoch 28/30]
```

Step	Training Loss	Validation Loss
100	0.548300	0.514097
200	0.483600	0.508764
300	0.459800	0.506091
400	0.436400	0.524542
500	0.427500	0.522932

```
/opt/conda/lib/python3.10/site-packages/transformers/modeling_utils.py:2618: UserWarning: Moving the following attributes
```

Outputs	Fine- Tuned Model Output	Technical Terms
Output 1	output1.wav	CUDA
Output 2	output1 (1).wav	REST API, HTTP
Output 3	output1 (2).wav	OAuth
Output 4	output1 (3).wav	SQL

Inference Time:

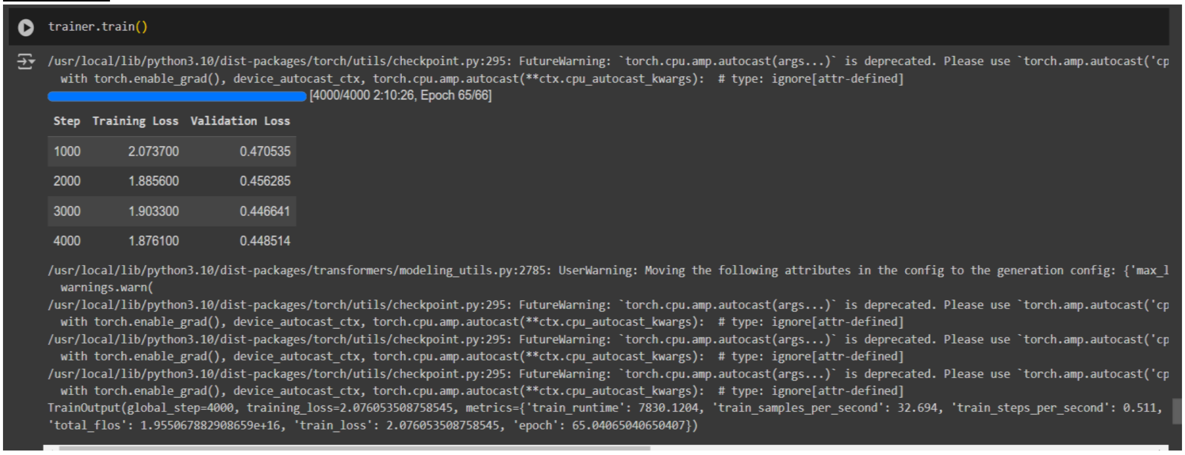
Output 1	0.0020 seconds
Output 2	0.0011 seconds
Output 3	0.0021 seconds
Output 4	00.20 seconds

MOS:

Output1	4
Output2	4.4

Output3	3.8
Output4	3.4

TASK2:



```
trainer.train()

/usr/local/lib/python3.10/dist-packages/torch/utils/checkpoint.py:295: FutureWarning: `torch.cpu.amp.autocast(args...)` is deprecated. Please use `torch.amp.autocast('cp
  with torch.enable_grad(), device_autocast_ctx, torch.cpu.amp.autocast(**ctx.cpu_autocast_kwargs):  # type: ignore[attr-defined]
[4000/4000 2:10:26, Epoch 65/66]
```

Step	Training Loss	Validation Loss
1000	2.073700	0.470535
2000	1.885600	0.456285
3000	1.903300	0.446641
4000	1.876100	0.448514

```
/usr/local/lib/python3.10/dist-packages/transformers/modeling_utils.py:2785: UserWarning: Moving the following attributes in the config to the generation config: {'max_l
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torch/utils/checkpoint.py:295: FutureWarning: `torch.cpu.amp.autocast(args...)` is deprecated. Please use `torch.amp.autocast('cp
  with torch.enable_grad(), device_autocast_ctx, torch.cpu.amp.autocast(**ctx.cpu_autocast_kwargs):  # type: ignore[attr-defined]
/usr/local/lib/python3.10/dist-packages/torch/utils/checkpoint.py:295: FutureWarning: `torch.cpu.amp.autocast(args...)` is deprecated. Please use `torch.amp.autocast('cp
  with torch.enable_grad(), device_autocast_ctx, torch.cpu.amp.autocast(**ctx.cpu_autocast_kwargs):  # type: ignore[attr-defined]
/usr/local/lib/python3.10/dist-packages/torch/utils/checkpoint.py:295: FutureWarning: `torch.cpu.amp.autocast(args...)` is deprecated. Please use `torch.amp.autocast('cp
  with torch.enable_grad(), device_autocast_ctx, torch.cpu.amp.autocast(**ctx.cpu_autocast_kwargs):  # type: ignore[attr-defined]
TrainOutput(global_step=4000, training_loss=2.076053508758545, metrics={'train_runtime': 7830.1204, 'train_samples_per_second': 32.694, 'train_steps_per_second': 0.511,
'total_flos': 1.955067882908659e+16, 'train_loss': 2.076053508758545, 'epoch': 65.04065040650407})
```

Outputs	**Fine-tuned model**	**Pre- trained model**
Output 1	output.wav	output2.wav
Output 2	output (1).wav	output2 (1).wav
Output 3	output (2).wav	output2 (2).wav
Output 4	output (3).wav	output2 (3).wav

Inference time:

Output 1	0.0014 seconds
Output 2	0.0016 seconds
Output 3	0.0009 seconds
Output 4	0.0081 seconds

MOS:

Output1	4.2
Output2	4.5
Output3	3.4
Output4	3.1

- **CHALLENGES:**

1. **Insufficient Data:** as per the problem statement and the domain, there wad limited amount of data for fine-tuning. Insufficient data could result in model overfitting and thus impacting the purpose and functionalities of the model.

2. **Data preparation:** even for custom data preparation, there were challenges to combine the audios and sentences for training due to its amount of trainable data and limited instruction for creating the data in an appropriate form.

3. **Limited computing resources:** for fine-tuning such model, there were challenges related to the computational powers like limited memory size, GPU resources and time. It was tough to train regional language model on available resources as all of the available resources could not satisfy the needs. This is the main reason to why there is a fusion solution for Task2 problem statement.

4. **Data Quality and size:** Challenges were faced specifically on the size of the datasets, especially for the TASK1 problem. The available 'English dataset' for model training was huge in size that the available computing resources failed. There is limitation when it comes to train a TTS model on English language Dataset due to its size and availability.

5. **Tokenization limitation:** This was a major challenge faced particularly in TASK2 as there is a limitation for selecting the tokenizer which is pretrained on a particular regional language like Hindi or Marathi. This created a problem for training the model for its pronunciations.

6. **Speaker model selection:** Other major challenge is the limitation for the selection of SPK model or the speaker for the model, particularly for TASK2 as there are no or limited speaker trained on regional languages like Hindi and Marathi.

   NOTE: The above challenge resulted in the development of a creative/ fusion solution for TASK2.

- ## CONCLUSION:

In this project, I explored Text-To-Speech models and their respective functionalities focusing on two different tasks: Technical Terms Pronunciations and Regional Language. This whole journey consisted a learning experience along with addressing the challenges that were faced on every step and phase for completion of the respective tasks. Fine-Tuning models on specific domains results on much better deliverables and high-quality speech.

Learnings:

1. Fine-Tuning TTS model for Technical Terms like API, CUDA etc., involved handling the phonemes carefully so that these words could be pronounced as letters and not just one word. This included providing proper and clear replacement direction to the model while fine- tuning. This task emphasized on the availability as well as the limitation of the dataset.
2. Fine- Tuning TTS model for Regional Language task involved careful selection of the dataset and analysing its phonemes so the model could give good results. However the speaker used in the model was trained on English language, it encouraged for thinking out of the box and coming up with an innovative solution to fuse Hindi language with English Speaker so as to get the desired results.
3. For each task, it was also important to make sure the models were coded as per the required needs in terms of embeddings, tokens, tensors etc., and this demanded to learn and understand the architecture of the models.

Future Scope:

1. The quality and availability of various factors like Dataset, speaker used in model significantly affects the delivering power of the text to speech model.
2. Heavy computing resources can result in a good training and fine tuning of the model thus resulting in a better output.
3. There is a need of more dataset and more pretrained models for further domain specific text-to speech models.
4. Since Text-to-Speech models have an increasing demand in various languages, therefore future scope would involve improving these TTS models so that they will include majority of the languages.
5. Future work would also include focusing on the model's architecture so that the model will not only work on giving the audio outputs from the text input but also making the model do contextual understanding. This will ensure that the pronunciations will be more accurate.

LINKS:

TASK1:

Dataset:(https://huggingface.co/datasets/Yassmen/TTS_English_Technical_data)
Model: (https://huggingface.co/Niha14/speecht5_finetuned_techincal_data)

TASK2:
Dataset: (https://huggingface.co/datasets/mozilla-foundation/common_voice_17_0/viewer/hi)
Model:
(https://huggingface.co/Niha14/speecht5_finetuned_techincal_data/blob/main/README.md)