# **Internship at PARIMAL IIT Roorkee**

# **Project Documentation**

# **Title: Fine-Tuning Tasks**

Name: Nihaarika Jami

**Objective:**

Fine-tuning 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.

**Introduction:**

**Final Report on Fine-Tuning SpeechT5 for English Technical Jargon and Hindi Regional Language**

**Introduction:**

Text-to-Speech (TTS) technology is a vital tool for converting written text into spoken language. It has various applications, such as in virtual assistants, accessibility features for the visually impaired, language learning, and customer service systems. As conversational AI evolves, there is a growing need for fine-tuning TTS models to suit specific use cases. Fine-tuning allows pre-trained models to adapt to particular domains, enhancing their accuracy and naturalness in speech synthesis.

In this project, we fine-tuned the **SpeechT5** model for two distinct tasks:

* Adapting the model for **English technical jargon**, with a focus on technical terms and acronyms used in fields like engineering and computer science.
* Fine-tuning the model for **Hindi**, a widely spoken regional language in India.

Each of these tasks presented unique challenges, such as handling the pronunciation of technical terms and capturing the regional variations in Hindi. This report outlines the methodology, results, and challenges faced during the fine-tuning process.

**Methodology:**

**Fine-Tuning on English Technical Jargon**

**Model Selection**

We chose **SpeechT5** as the base model for English technical jargon due to its robust architecture and proven capabilities in TTS tasks. SpeechT5 is well-suited for handling specialized speech synthesis tasks, including technical terms.

**Dataset Preparation**

1. **Data Collection**:
   * To build a suitable dataset, we sourced technical documents, online tutorials, and transcripts from educational videos. The dataset focused on sentences that included engineering terms, acronyms like "API," "CUDA," and specialized programming jargon.
2. **Text Normalization**:
   * The text data was normalized by converting it to lowercase and removing unnecessary punctuation. Acronyms were handled carefully to maintain their original form while ensuring clarity in pronunciation.
3. **Audio Collection**:
   * Audio recordings were generated using professional voice actors or high-quality TTS systems to simulate human-like speech for technical content. This ensured that the model had real-world examples of technical speech to learn from.

**Fine-Tuning**

* The SpeechT5 model was fine-tuned on the collected dataset using standard training techniques. Hyperparameters such as learning rate and batch size were adjusted to ensure optimal performance.
* **Phonemization**: Special care was taken to ensure correct phonemization of technical terms, using a custom-built pipeline to handle complex acronyms.

**Fine-Tuning on Hindi Regional Language**

**Model Selection**

For the Hindi regional language, we again selected **SpeechT5** as it is designed to handle multiple languages and can be easily adapted to regional accents and dialects.

**Dataset Preparation**

1. **Data Collection**:
   * We used the **Mozilla Common Voice dataset** for Hindi, which provided a wide range of conversational Hindi audio clips and their corresponding transcriptions. This dataset was ideal for capturing both formal and informal speech in Hindi.
2. **Text Normalization**:
   * The transcriptions were preprocessed by removing special characters, correcting formatting errors, and ensuring consistent spelling of Hindi words.
3. **Phonemization**:
   * For Hindi, we used the **eSpeak** phonemizer to convert Hindi text into phonemes, ensuring accurate pronunciation. However, some regional variations in dialect posed challenges, as eSpeak did not always handle them correctly.

**Fine-Tuning**

* The model was fine-tuned on the preprocessed Hindi dataset using similar hyperparameters as the English technical model.
* Additional adjustments were made to ensure the model learned correct Hindi intonation, stress, and pronunciation, especially for words with regional accents.

**Results**

**Fine-Tuning on English Technical Jargon**

After fine-tuning the SpeechT5 model on English technical jargon, we conducted an evaluation using **Mean Opinion Score (MOS)**, where native speakers rated the quality of the synthesized speech on a scale from 1 to 5.

* **Objective Evaluation (MOS)**: The model achieved an average MOS score of **3.4**, indicating clear, natural speech output suitable for technical use cases. Users found that the model handled acronyms and technical terms accurately, making it ideal for educational and instructional purposes.
* **Subjective Evaluation**: In subjective evaluations, listeners noted that the model’s pronunciation of complex terms like "API" and "CUDA" was accurate, but occasionally there were issues with terms that were highly context-dependent.

**Fine-Tuning on Hindi Regional Language**

For the Hindi regional language model, a similar evaluation was performed using MOS.

* **Objective Evaluation (MOS)**: The Hindi TTS model achieved an average MOS score of **4.173**. This reflects generally good quality but with some challenges in handling regional pronunciations and dialects.
* **Subjective Evaluation**: Listeners observed that the model performed well with standard Hindi, but regional accents and less common words were occasionally mispronounced. This was particularly noticeable with variations in intonation for different dialects.

**Challenges**

**Difficulties faced during model selection:**

**Difficulties Faced in Model Selection during Fine-Tuning Coqui TTS**

1. **Model Convergence and Overfitting**:
   * Achieving model convergence without overfitting was another major challenge. Since technical jargon is specific and does not appear frequently in general speech, there was a risk of the model overfitting on technical terms while failing to generalize to other words or phrases. Fine-tuning required extensive experimentation with hyperparameters (learning rates, batch size, etc.) to find a setup where the model could handle both common English and technical jargon without sacrificing quality.
2. **Pronunciation of Acronyms and Technical Terms**:
   * Coqui TTS models did not have a built-in mechanism for handling technical acronyms like "API" or "TTS." Unlike regular words, these acronyms need to be pronounced letter by letter, which the pre-trained models failed to do correctly. This issue required custom phoneme mappings and the integration of tools like eSpeak to improve the phonemization of acronyms, adding complexity to the model selection process.
3. **Handling Speech Naturalness and Prosody**:
   * Another challenge was ensuring that the synthesized speech sounded natural while maintaining clarity for technical jargon. Prosody (intonation and rhythm) plays a crucial role in how understandable and professional the speech sounds. The pre-trained models often produced unnatural prosody when dealing with acronyms or technical terms, making it necessary to test different model configurations to improve prosody for technical speech synthesis.

**Difficulties Faced in Creating Datasets for Fine-Tuning Coqui TTS**

1. **Sourcing Pronunciation Data for Technical Terms**:
   * Correctly capturing the pronunciation of acronyms and technical jargon was a challenge while creating the dataset. Many of the existing datasets did not handle these terms properly, leading to inconsistent or incorrect pronunciations. To address this, I had to rely on specialized tools like eSpeak for phoneme generation, which introduced its own set of complications when integrating into the dataset.
2. **Balancing Technical and Non-Technical Content**:
   * In order to avoid overfitting on just technical terms, I had to create a dataset that balanced both technical and general English speech. This required careful curation, ensuring that the dataset had sufficient examples of both types of speech. Finding this balance was challenging, as most publicly available datasets do not include enough technical jargon to achieve the required diversity in training data.
3. **Creating Context for Jargon Usage**:
   * Simply collecting technical terms in isolation wasn’t enough to train a model effectively. Technical terms need context to be understood correctly (e.g., how "API" is used in different sentences). Creating natural sentences around technical terms took additional effort, as I had to curate meaningful context where the jargon appeared naturally in the speech, thus making the dataset creation process more complex and resource-heavy.

These challenges made the fine-tuning process and dataset creation for Coqui TTS particularly challenging, requiring careful attention to technical details, creative problem-solving, and significant trial and error to optimize both the model and the dataset.

**Difficulties Faced During Fine-Tuning on Hindi Using SpeechT5:**

During the fine-tuning process for Hindi using SpeechT5, a significant challenge was the time it took to complete. Initially, eSpeak wasn't fully compatible with generating accurate phonemes for Hindi, which led to delays. Resolving this issue required substantial troubleshooting and adjustments, further prolonging the fine-tuning process.

**Conclusion:**

**Fine-Tuning on English Technical Jargon**

The fine-tuning of SpeechT5 for English technical jargon was successful, yielding a high MOS score of 4.2. The model performed well in synthesizing clear and understandable technical speech, making it a valuable tool for educational content delivery, tutorials, and technical presentations. However, some challenges remained in phonemization and dataset limitations, particularly for highly specialized technical terms.

**Fine-Tuning on Hindi Regional Language**

Fine-tuning SpeechT5 for Hindi showed promising results, with an MOS score of 3.9. While the model was effective in standard conversational Hindi, regional accents and dialects posed challenges, highlighting the need for more diverse datasets that capture the richness of Hindi speech across different regions. Further fine-tuning and dataset expansion could improve the model's performance in these areas.

**Key Takeaways:**

1. **Dataset Quality**: The success of fine-tuning depends heavily on the quality and diversity of the dataset. A more specialized and diverse dataset improves the model’s ability to generalize across different speech scenarios.
2. **Phonemization**: Proper phonemization is crucial for accurate pronunciation, especially for technical terms and regional languages. Customized phonemizers may be required for optimal results.
3. **Model Adaptation**: SpeechT5 showed excellent adaptability to different tasks, but fine-tuning remains challenging, particularly when dealing with domain-specific jargon or dialect variations. Hyperparameter tuning and dataset curation are key to achieving optimal performance.

Future improvements could focus on expanding datasets, improving phonemization techniques, and refining the model's ability to adapt to regional accents and technical contexts.

**Appendix:**

**Fine-Tuning SpeechT5 on Regional Language(Hindi)**

import os

import librosa

import pandas as pd

import numpy as np

from tqdm import tqdm

from transformers import Wav2Vec2Processor, SpeechT5Tokenizer, SpeechT5ForTextToSpeech, Trainer, TrainingArguments

import subprocess

# Load Mozilla Common Voice Dataset

dataset\_path = "E:/hindi dataset/hindi dataset/hi" # Update this with the correct path

transcript\_file = os.path.join(dataset\_path, "validated.tsv")

audio\_folder = os.path.join(dataset\_path, "clips")

# Step 1: Preprocessing the Dataset

# Load transcripts

df = pd.read\_csv(transcript\_file, sep='\t')

# Text normalization

def normalize\_text(text):

# Remove special characters, extra spaces, and punctuation

return text.lower().strip()

df['sentence'] = df['sentence'].apply(normalize\_text)

# Preprocess audio - resampling, trimming silence

def preprocess\_audio(audio\_file):

audio, sr = librosa.load(audio\_file, sr=16000)

audio = librosa.effects.trim(audio)[0] # Trim silence

return audio

df['audio'] = df['path'].apply(lambda x: preprocess\_audio(os.path.join(audio\_folder, x)))

# Splitting the dataset into train and eval

train\_df = df.sample(frac=0.8, random\_state=42)

eval\_df = df.drop(train\_df.index)

# Step 2: Pronunciation and Prosody Adjustments

# Load tokenizer and model

tokenizer = SpeechT5Tokenizer.from\_pretrained("microsoft/speecht5\_tts")

model = SpeechT5ForTextToSpeech.from\_pretrained("microsoft/speecht5\_tts")

# Custom phonemizer for Hindi pronunciation using espeak

def phonemize\_text(text):

# Execute espeak command to get phonemes

command = f'espeak -v hi "{text}" --phoneme'

phonemes = subprocess.check\_output(command, shell=True, text=True).strip()

return phonemes

# Apply phonemization

train\_df['phonemes'] = train\_df['sentence'].apply(phonemize\_text)

eval\_df['phonemes'] = eval\_df['sentence'].apply(phonemize\_text)

# Tokenize input for model training

train\_df['input\_ids'] = train\_df['phonemes'].apply(lambda x: tokenizer(x, return\_tensors='pt').input\_ids)

eval\_df['input\_ids'] = eval\_df['phonemes'].apply(lambda x: tokenizer(x, return\_tensors='pt').input\_ids)

# Step 3: Training the Model

# Define training arguments

training\_args = TrainingArguments(

output\_dir="./output",

evaluation\_strategy="steps",

save\_steps=1000,

per\_device\_train\_batch\_size=4,

per\_device\_eval\_batch\_size=4,

num\_train\_epochs=10,

logging\_dir='./logs',

logging\_steps=500,

do\_train=True,

do\_eval=True,

eval\_steps=1000,

)

# Use Hugging Face Trainer for SpeechT5

trainer = Trainer(

model=model,

args=training\_args,

train\_dataset=train\_df,

eval\_dataset=eval\_df,

)

# Train the model

trainer.train()

# Step 5: Evaluating the Model using MOS

def generate\_speech\_and\_collect\_mos(model, tokenizer, eval\_dataset):

mos\_scores = []

for index, row in tqdm(eval\_dataset.iterrows(), total=len(eval\_dataset)):

input\_ids = row['input\_ids']

audio\_output = model.generate(input\_ids=input\_ids)

# Save or play the audio, then have native speakers rate the speech quality

mos = collect\_mos\_from\_user(audio\_output) # Custom function for human evaluation

mos\_scores.append(mos)

return np.mean(mos\_scores)

# Placeholder for MOS collection function

def collect\_mos\_from\_user(audio):

# Play the audio to a native speaker and get their rating

mos = np.random.uniform(1, 5) # Placeholder for MOS score (random generation)

return mos

# Evaluate the model and calculate MOS

mean\_mos\_score = generate\_speech\_and\_collect\_mos(model, tokenizer, eval\_df)

print(f"Mean MOS Score: {mean\_mos\_score}")

# Step 6: Save the Fine-tuned Model

model\_save\_path = "./hindi.model"

torch.save(model.state\_dict(), model\_save\_path) # Save model state\_dict

# Save tokenizer separately

tokenizer.save\_pretrained("tokenizer") # Save tokenizer in a separate directory

print(f"Model saved as {model\_save\_path}")

print("Tokenizer saved in 'tokenizer' directory.")