Task 1

Fine-tuning TTS for English with a Focus on Technical Vocabulary

Saurabh Kumar

Introduction

Fine-tuning a text-to-speech (TTS) model to accurately handle technical jargon commonly used in English technical interviews, such as "API," "CUDA," and "TTS." These terms often pose challenges for standard TTS models, which may mispronounce or fail to properly handle them. The objective of this task is to ensure that the model can fluently and correctly pronounce such jargon, providing more accurate and professional-sounding speech outputs in scenarios like technical interviews, presentations, or educational content. Fine-tuning a pre-trained TTS model on a specialized dataset containing these technical terms is essential to improve its accuracy and usability in such contexts.

Handling technical jargon effectively in real-world communication is crucial, particularly in technical interviews where precise understanding and articulation of concepts is vital. Mispronunciations of terms like "API" or "CUDA" can lead to misunderstandings, reducing the clarity of communication. In technical interviews, candidates must convey their knowledge efficiently, and accurate pronunciation ensures professionalism and credibility. Moreover, in educational settings or automated systems, incorrect pronunciations can confuse listeners and hinder learning. Fine-tuning TTS models to handle jargon enhances their applicability in technical contexts, enabling them to support smoother, more effective communication.

Dataset Description

Dataset Name: English Technical Jargon Vocabulary

Data Source: Custom Self – Recorded Audio (Audio, Text), sourced from Hugging Face and Sample set from - https://github.com/AI4Bharat/NPTEL2020-Indian-English-Speech-Dataset?tab=readme-ov-file (Downloads, Sample Data (Pure-Set) Train, Test and Dev sets downloader scripts)

Dataset Format: [Text = String], [Audio = mono, 16khz], [Metadata_file.csv]

```
filename,text
D:/Study_Material/Python/Computer_Vision/Fine_Tuning_TTS_Model/TTS_Custom/Audio/output_audio_0.wav,so this point one i just
D:/Study_Material/Python/Computer_Vision/Fine_Tuning_TTS_Model/TTS_Custom/Audio/output_audio_1.wav,of chat GPT or maybe it i
D:/Study_Material/Python/Computer_Vision/Fine_Tuning_TTS_Model/TTS_Custom/Audio/output_audio_2.wav,"Twilio does have one of
D:/Study_Material/Python/Computer_Vision/Fine_Tuning_TTS_Model/TTS_Custom/Audio/output_audio_3.wav,plus 3 and so it's just b
D:/Study_Material/Python/Computer_Vision/Fine_Tuning_TTS_Model/TTS_Custom/Audio/output_audio_4.wav,"There are people, you mi
D:/Study_Material/Python/Computer_Vision/Fine_Tuning_TTS_Model/TTS_Custom/Audio/output_audio_5.wav,are the differences pret
D:/Study_Material/Python/Computer_Vision/Fine_Tuning_TTS_Model/TTS_Custom/Audio/output_audio_6.wav,beginning it's all zero t
D:/Study_Material/Python/Computer_Vision/Fine_Tuning_TTS_Model/TTS_Custom/Audio/output_audio_7.wav,Discord Community to actu
D:/Study_Material/Python/Computer_Vision/Fine_Tuning_TTS_Model/TTS_Custom/Audio/output_audio_8.wav,we're making a new line a
D:/Study_Material/Python/Computer_Vision/Fine_Tuning_TTS_Model/TTS_Custom/Audio/output_audio_9.wav,run so we see that this r
D:/Study_Material/Python/Computer_Vision/Fine_Tuning_TTS_Model/TTS_Custom/Audio/output_audio_10.wav,"and we think that this
D:/Study_Material/Python/Computer_Vision/Fine_Tuning_TTS_Model/TTS_Custom/Audio/output_audio_11.wav,"We did that, we did tha
D:/Study_Material/Python/Computer_Vision/Fine_Tuning_TTS_Model/TTS_Custom/Audio/output_audio_12.wav,stream. columns and then
D:/Study_Material/Python/Computer_Vision/Fine_Tuning_TTS_Model/TTS_Custom/Audio/output_audio_13.wav,One thing I want you to
D:/Study_Material/Python/Computer_Vision/Fine_Tuning_TTS_Model/TTS_Custom/Audio/output_audio_14.wav,and ydev to evaluate the
```

Ex. Metadata.csv

Number of Samples: 11347 Self-recorded audio samples with its corresponding text pairs.

Technical Terms List:

Binary Search Tree (BST),
Depth-First Search (DFS),
Breadth-First Search (BFS),
VPN (Virtual Private Network),
REST (Representational State Transfer),
API (Application Programming Interface),
SaaS (Software as a Service),
PaaS (Platform as a Service),
IaaS (Infrastructure as a Service),
Cache,
OAuth,
GraphQL,
SQL (Structured Query Language),
NoSQL,
MongoDB,
Object-Oriented Programming (OOP),
GitHub,
Machine Learning (ML),
Deep Learning (DL),
Artificial Intelligence (AI),
Convolutional Neural Network (CNN),
Recurrent Neural Network (RNN),
Long Short-Term Memory (LSTM),
BERT (Bidirectional Encoder Representations from Transformers),
GPT (Generative Pre-trained Transformer),
Natural Language Processing (NLP),
Natural Language Understanding (NLU)
Word2Vec,
Principal Component Analysis (PCA),
K-Means Clustering,
Support Vector Machine (SVM),
XGBoost,
ROC Curve,
AUC (Area Under the Curve),
GPU (Graphics Processing Unit),
TPU (Tensor Processing Unit),
CUDA (Compute Unified Device Architecture)
FPGA (Field-Programmable Gate Array),
ASIC (Application-Specific Integrated Circuit),
RAM (Random Access Memory),
ROM (Read-Only Memory),
Intrusion Detection System (IDS),
Kubernetes

Fine-Tuning Process of the SpeechT5 TTS Model

The steps include data preparation, text normalization, speaker embedding extraction, model setup, and training. Below is a detailed breakdown of each stage in the process.

1. Initialization and Setup
First, the Hugging Face `SpeechT5ForTextToSpeech` model and `SpeechT5Processor` were loaded using the pre-trained checkpoint `"microsoft/speecht5_tts"`. This model is specialized for converting text into speech, making it a suitable choice for fine-tuning on the target dataset.

2. Loading and Processing the Dataset
The dataset, stored as a CSV file, was loaded using the `datasets` library. The dataset contained columns for audio data and corresponding transcriptions. The audio was cast to a standard sampling rate of 16 kHz for consistency across all audio samples.

3. Text Normalization
To ensure the text input was in a clean and consistent format, normalization steps were performed. This involved converting all text to lowercase, removing punctuation (except apostrophes), and stripping extra whitespace. The normalized text was added as a new column in the dataset. Additionally, numeric characters and underscores in the text were replaced with their respective word forms (e.g., "0" to "zero", "1" to "one") to make the speech output clearer for technical terms that might include numbers or underscores.

4. Speaker Embedding Extraction
To enhance the speech generation, speaker embeddings were generated using SpeechBrain's pre-trained `EncoderClassifier` model (`speechbrain/spkrec-xvect-voxceleb`). These embeddings capture speaker-specific characteristics, allowing the TTS model to generate speech that mimics specific speakers. The speaker embeddings were then added to the dataset during the preprocessing step, alongside the text and audio data.

5. Data Filtering and Splitting
To ensure efficient training, sequences that were too long (with more than 200 tokens) were filtered out to prevent training difficulties. The dataset was then split into training and testing sets using an 90-10 split.

6. Custom Data Collator
A custom data collator was created to handle padding of both input text and audio labels. Additionally, the speaker embeddings were incorporated into the batch, and special care was taken to handle padding correctly by masking the padded tokens in the labels to avoid affecting the loss computation.

7. Training Setup
The training setup involved specifying hyperparameters such as the:

- learning rate (0.0001),
- batch size (4),
- gradient accumulation steps (8),
- gradient checkpointing to optimize memory usage during training,
- maximum of 1000 steps,
- with evaluation and logging occurring every 100 steps.
- The training was set up to push the best model to the Hugging Face Hub.

8. Fine-Tuning the Model

The model was trained and evaluated on the technical dataset. The training process involved feeding the text and audio data through the model, using the speaker embeddings to help generate speech that reflected speaker characteristics.

9. Post-Training Inference and Speech Generation

Once the model was fine-tuned, it was used to generate speech from a given text input. The `SpeechT5HifiGan` vocoder was employed to convert the model's outputs into high-fidelity waveforms.

Model Performance

Training Loss	Step	Validation Loss
0.589100	100	0.512154
0.500500	200	0.509590
0.523000	300	0.475684
0.512100	400	0.474158
0.500500	500	0.464333
0.503700	600	0.467483
0.497200	700	0.458246
0.484300	800	0.454266
0.486300	900	0.452102
0.477300	1000	0.451052

Objective Metrics

Model	MOS	Inference Times (s)
SpeechT5_English_Technical	4.1	5.7
SpeechT5_Base	3.9	6.1

Technical Terms and their Pronunciation

The corresponding audio files for the technical terms that are mentioned above is present in the Github repository to which you can directly jump by clicking the below link.

Link - https://github.com/Saurabh-Kumar-0/TTS_IITR/tree/main/Technical_Terms_Audio

Conclusion

This report outlines the process of fine-tuning two Text-to-Speech (TTS) models, focusing on English technical speech and a regional language model. TTS is an essential technology used in various applications such as virtual assistants, accessibility tools for the visually impaired, and content creation. Fine-tuning allows pre-trained models to adapt to specific domains, improving their performance on specialized tasks like handling technical jargon or regional accents.