

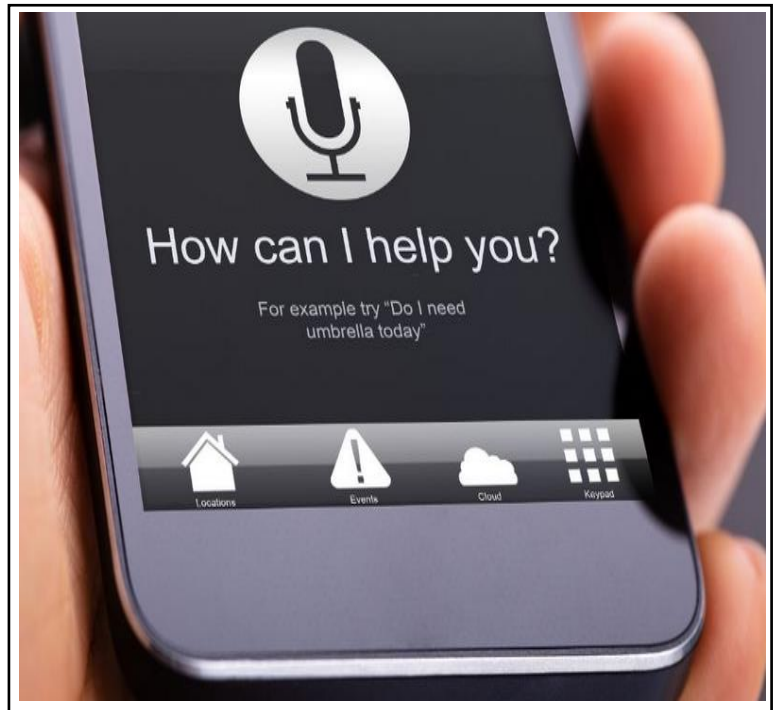
# AI-Driven Speech Recognition Models

## Speech Recognition with End-to-End ASR

**Topic Areas:** Speech-to-Text Speaker Diarization Automatic Speech Recognition (ASR) Noise-Robustness Data Augmentation Deep Learning

Matthew Harper  
*University of Houston Victoria*

**Abstract:** AI-based speech recognition is a rapidly growing field of research which has transformed the way humans interact not only with machines but with each other. Speech recognition is considered a technology that enables machines to understand and interpret human speech [1]. Also known as automatic speech recognition (ASR), computer speech recognition or speech-to-text, is a capability that enables a program to process human speech into a written format. While speech recognition is commonly confused with voice recognition, speech recognition focuses on the translation of speech from a verbal format to a text one whereas voice recognition just seeks to identify an individual user's voice [6]. This report provides a summary introduction into speech recognition models and illustrates a basic ASR model with the lab practicum.



**Figure 1** – Example of a Speech-to-Text model utilized by common smart phone applications [4].

### Introduction

Speech recognition refers to the technology that enables machines to understand and interpret human speech. This technology involves converting spoken language into text through a series of complex processes, including audio signal processing, feature extraction, and pattern recognition. [1].

Early development of speech recognition systems began in the early 1950s at Bell Laboratories. In 1952, a pioneering system named "Audrey" was developed. This innovative creation, a testament to the early days of speech recognition, demonstrated remarkable capabilities. Audrey could accurately recognize spoken digits, from zero to nine, with an impressive accuracy rate exceeding 90%. The system was trained on the voice of its developer, HK David, showcasing the early limitations of speech recognition technology that relied on specific voices and limited vocabularies [6].

Building upon the foundation laid by Audrey, researchers continued to push the boundaries of speech recognition technology. In 1962, IBM introduced the "Shoebbox," a significant step forward in the field. This innovative machine demonstrated the ability to recognize a vocabulary of 16 spoken English words, marking a substantial increase in the scope of speech recognition capabilities. Simultaneously, Soviet scientists made remarkable progress in developing algorithms capable of recognizing over 200 words, further expanding the potential applications of speech recognition technology. These advancements in the 1960s laid the groundwork for future developments and demonstrated the growing potential of speech recognition as a tool for human-computer interaction [4].

Today, deep learning models combined with cloud computing have largely reinvigorated the advanced field of text-to-speech technology. Cloud-based platforms offer scalable solutions for processing large volumes of data, enabling the development of more sophisticated and accurate models. Additionally, the availability of vast amounts of data has fueled the development of deep learning techniques, which have significantly improved the quality of machine generated and synthesized speech. These innovations have made speech recognition technology more accessible and versatile, finding applications in various domains, including virtual assistants, language learning, and accessibility tools [4].

Even today, speech recognition is a complex field within computer science, which involves converting spoken language into written text. Overall, speech recognition (SR) is a complex field involving linguistics, mathematics, and statistics. Speech recognizers consist of components like speech input, feature extraction, feature vectors, a decoder, and word output [6].

#### **Components of Speech Recognition:**

Audio Signal Processing: The first step in speech recognition is capturing the audio signal using microphones. The audio is then processed to remove noise and enhance the clarity of the speech signal.

Feature Extraction: The processed audio signal is analyzed to extract key features, such as pitch, tone, and frequency. These features are crucial for distinguishing different sounds and phonemes in the speech.

Pattern Recognition: The extracted features are then compared against a database of known patterns (e.g., phonemes, words, and phrases). The system uses these patterns to interpret the spoken words and convert them into text.

Language Modeling: Language models help improve the accuracy of speech recognition by predicting the likelihood of certain words or phrases occurring in a given context. These models are trained on large datasets of text and speech to understand the structure and grammar of the language.

#### **Speech Recognition Model Accuracy and Word Error Rate (WER)**

The decoder uses acoustic models, pronunciation dictionaries, and language models to determine the appropriate output. Accuracy is measured by word error rate (WER). Factors like pronunciation, accent, pitch, volume, and background noise affect WER and is one of the primary challenges in speech recognition along with spoken language accents and semantics. The goal is to achieve human parity in WER. Research suggests a WER of around 4% is possible, but replication has been difficult. It requires a sophisticated combination of linguistics, mathematics, and statistics to achieve high accuracy [1, 6].

## **AI Driven Speech Recognition**

Artificial intelligence has revolutionized speech recognition (SR) by addressing the limitations of traditional rule-based systems, which were often constrained by predefined patterns and struggled to adapt to variations in accents, dialects, and speech patterns. AI-driven systems, powered by deep learning models and natural language processing techniques, have overcome these challenges, enabling more accurate, versatile, and user-friendly speech recognition capabilities. As a result, AI-based systems have revolutionized speech recognition by addressing the limitations of traditional rule-based systems, which were often constrained by predefined patterns and struggled to adapt to variations in accents, dialects, and speech patterns. These AI-driven systems, powered by deep learning models and natural language processing techniques, have overcome these challenges, enabling more accurate, versatile, and user-friendly speech recognition capabilities [1].

Deep learning models, and in particular neural networks, have been instrumental in improving the accuracy and adaptability of speech recognition systems. These models can learn from vast amounts of data, allowing them to recognize a wider range of speech patterns, accents, and dialects. By leveraging the power of deep learning, AI-driven systems can effectively capture the nuances of human speech, making them more robust and reliable in diverse environments [1].

Moreover, the integration of natural language processing (NLP) techniques has significantly enhanced the ability of speech recognition systems to understand the context and meaning behind spoken words. NLP algorithms can analyze the semantic and syntactic structure of language, allowing systems to not only transcribe speech but also comprehend the intent and context of the utterance. This capability enables more natural and intuitive interactions, as systems can respond appropriately to user queries and commands, even in complex or ambiguous situations [1].

### **Examples of AI-Driven Enhancements in Speech Recognition [1]:**

Deep Learning and Neural Networks: AI-driven systems use deep learning models to improve accuracy and adaptability, allowing them to recognize a wider range of speech patterns and accents.

Natural Language Processing (NLP): AI integrates NLP techniques to understand the context and meaning behind spoken words, enabling more natural and intuitive interactions.

Real-time Adaptation: AI enables systems to adapt to individual users over time, improving their ability to recognize specific speech patterns and preferences.

Multilingual Support: AI has significantly improved the ability of speech recognition systems to support multiple languages and dialects.

Overall these AI aided enhancements are enabling the development of more accurate, versatile, and user-friendly speech recognition systems with a wider range of applications, thereby, transforming the way we interact with technology now and in the future.

### **Applications in AI Enhanced Speech Recognition: Transforming Industries**

Artificial intelligence has significantly advanced speech recognition (SR) technology, enabling a wide range of application across various industries. By leveraging the power of AI, speech recognition systems have become more accurate, efficient, and versatile, transforming the way we interact with machines [1].

One of the most notable impacts of AI-driven speech recognition is in the healthcare sector. AI-powered systems can automatically transcribe medical dictation, reducing the administrative burden on healthcare providers and improving the accuracy of medical records. This enables doctors and other healthcare professionals to focus more on patient care and less on time-consuming documentation tasks [1].

In the customer service industry, AI-powered speech recognition has revolutionized the way businesses interact with their customers. Virtual assistants and chatbots powered by AI can understand and respond to customer queries in real time, providing immediate assistance without the need for human intervention. This improves customer satisfaction and reduces wait times, leading to more efficient and effective customer service operations.

The automotive industry has also benefited significantly from AI-driven speech recognition. Voice-controlled systems in vehicles allow drivers to control navigation, make phone calls, and adjust settings using their voice, reducing distractions and promoting safer driving. By minimizing the need for manual interaction with in-car systems, speech recognition technology enhances the overall driving experience.

#### **Example Applications of AI-Driven SR Systems [1]:**

Healthcare: AI-powered speech recognition automates medical transcription, reducing administrative burdens and improving the accuracy of medical records.

Customer Service: Virtual assistants and chatbots powered by AI enhance customer service efficiency, reduce wait times, and improve overall customer experiences.

Automotive Industry: Voice-controlled systems in vehicles, enabled by speech recognition, contribute to safer driving experiences by minimizing distractions.

Education: AI-driven speech recognition tools make education more accessible, facilitating language learning and providing support for students with disabilities.

Entertainment: Voice-controlled gaming, virtual reality, and content search enhance user experiences, making entertainment more interactive and engaging.

The recent developments in AI-driven speech recognition is transforming industries by enabling more natural and intuitive human-computer interactions. This technology improves accuracy, efficiency, and accessibility, opening up new possibilities for innovation in various fields and continues to advance the field of speech recognition and tis applications[1].

#### **Speech-to-Text with Deep Learning**

Deep learning neural network based speech recognition (SR) systems address limitations of traditional rule-based systems, which are often constrained by predefined patterns and struggle to adapt to variations in accents, dialects, and speech patterns. Moreover, AI-driven systems, powered by deep learning models and natural language processing techniques, have overcome these challenges, enabling more accurate, versatile, and user-friendly speech recognition capabilities. Additionally, this is a critical advancement in

the field of natural language processing (NLP) which has revolutionized how we convert spoken language into text [1].

The following items include the process steps of converting speech to text, all of which can be efficiently handled by deep learning models [1]:

Audio Input Processing: The first step is to process the incoming audio signal, which typically involves splitting the audio into small segments or "slices." These segments might be as short as 20 milliseconds, as shown in the diagram on the slide.

Feature Extraction: Each audio slice is then transformed into a set of features that can be processed by the neural network. Common features include Mel-frequency cepstral coefficients (MFCCs) and spectrograms, which represent the frequency and intensity of the audio signal over time.

Modeling Sequential Data: Speech is inherently sequential, meaning the order of the audio slices matters. Deep learning models, particularly those designed for sequential data, are used to capture this temporal relationship. The model processes each slice in the context of previous slices to maintain the continuity of speech.

Prediction of Phonemes or Words: As the model processes each audio slice, it generates a probability distribution over possible phonemes or words. The model's current state, influenced by previous audio slices, guides the prediction for the current slice.

Decoding the Output: The final step involves decoding the sequence of predicted phonemes or words into coherent text. This might involve additional processing, such as using language models to improve the grammaticality and coherence of the output.

An important feature of any Deep Learning Model is the architecture utilized for the neural network. Although many, there are a few prominent models more commonly used for SR and speech-to-text purposes. These include the following deep learning models [1]:

Recurrent Neural Networks (RNNs): RNNs are designed to handle sequential data by maintaining a hidden state that is passed from one time step to the next. This makes them suitable for speech recognition, where the sequence of sounds is crucial.

Long Short-Term Memory (LSTMs): A form of RNN that includes special gates to control the flow of information, allowing them to maintain long-term dependencies. This helps in handling sequences where context from earlier in the sequence is important.

Transformers: Transformers rely on self-attention mechanisms to process sequences in parallel, rather than sequentially. This allows for more efficient computation and the ability to capture long-range dependencies more effectively.

### **Speech-to-Text Speaker Diarization**

Speaker diarization is a technique used to identify and separate the speech of different speakers in an audio recording. By analyzing the audio stream, the system can determine who spoke at what time, providing valuable information for applications like transcription, analysis, or indexing. In addition, This

technique is especially useful in scenarios with multiple speakers, such as meetings, interviews, or broadcast news, where distinguishing between different speakers is crucial for accurate understanding and analysis of the content [1].

The primary purpose of speaker diarization is to enhance transcription, understanding, and usefulness of audio data by detecting, identifying, and separating speakers. This enhances transcription, provides better speaker-specific analysis, and improved interaction with the data [1].

**Speaker diarization has various practical applications, including [1]:**

Improved Transcription: In transcription settings, diarization provides valuable information by labeling the speech of different speakers, making transcripts more understandable and useful.

Meeting Analysis: Diarization tools can track participation, identify dominant speakers, and analyze the content of each speaker's speech, aiding in meeting summarization and understanding group dynamics.

Audio Indexing: For large datasets of audio or video content, diarization helps index data by speaker, facilitating easier searching and retrieval of information.

Enhanced User Experience: In voice-activated systems, diarization can improve the user experience by distinguishing between different users and responding appropriately based on the speaker's identity.

These applications demonstrate the versatility and value of speaker diarization in various domains, from transcription to meeting analysis and user experience enhancement.

**Generalized Inference with Deep Learning SR models**

Deep learning models have significantly outperformed traditional methods in speech recognition due to their ability to capture and model complex patterns in speech data. Unlike traditional methods, which rely on handcrafted features and statistical models, deep learning models can automatically learn relevant features from raw audio data, leading to more accurate and robust recognition [1].

**Noise Robustness Filtering in SR**

Deep learning methods utilized for SR enable the ability to handle and process variability in speech data. Traditional methods struggled to cope with variations in accents, speaking styles, and noisy environments, often leading to degraded performance. Deep learning models, on the other hand, can learn to generalize across these variations, making them more adaptable to different speech inputs. This robustness is particularly important in real-world applications where speech data can be noisy or non-standard [1].

Moreover, noise is typically a challenging barrier with speech recognition systems which limits accurate transcription and understanding of speech.

A few challenges include the following [1]:

Variety of Sound Sources – Unwanted sounds or noise can come from many varying sources including background sound, traffic, industrial machinery, and even environmental factors such as rain and wind.

Signal Degradation – Noise can appear in form of sound signal transform quality which translates into incorrect feature extraction.

Overlapping Speech – Constructive and destructive interference can occur when multiple people are speaking simultaneously in similar frequency ranges. This creates an overlapping effect and is challenging to filter out unwanted noisy interference in the wanted sound.

Variability in Noise Levels – Varying noise levels can change rapidly within the same environment or across different environments requiring speech recognition systems to adapt quickly and effectively to maintain accuracy.

There are several prominent methods utilized to improve noise reduction and noise robustness within speech recognition systems including [1]:

Noise Reduction Algorithms – Algorithms designed to filter out noise from the audio signal before it is processed by the speech recognition system. Common methods include spectral subtraction, Wiener filtering, and adaptive filtering.

Data Augmentation – This method involves artificially creating variations in the training data to improve the model's ability to generalize to different noise conditions.

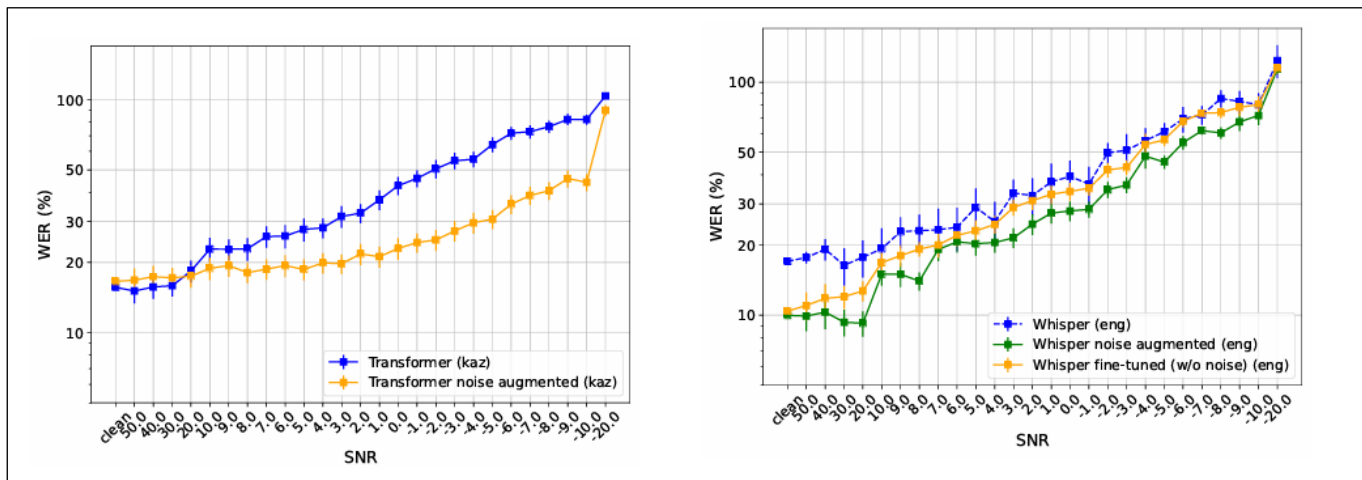
Training with Noisy Data – The objective of training with noisy audio samples is to train the model to distinguish between noise and speech.

In addition, it is not uncommon for multiple combinations of the above methods to be utilized to train and fine-tune noise-robust automatic speech recognition systems. Moreover, recent research has demonstrated significant progress by creating noisy training models with the use of data augmentation to create and embed the noisy audio files, combined with the use of adaptive filtering algorithms [1, 5].

Noisy audio files were generated by adding noise to original audio with a particular signal-to-noise ratio (SNR) which is the ratio between signal and noise [5]. The higher the SNR the better signal quality as illustrated in the formula below where convolution was utilized to add or embed the noisy signals into the original signal [5].

$$training\_audio = orig\_audio + a * noise \quad a = \sqrt{10^{\frac{-SNR}{10}} \frac{\|orig\_audio\|_2^2}{\|noise\|_2^2}} \quad [5]$$

The graphs below in the figure indicate a significant reduction in % WER in both monolingual and multilingual models.



**Figure 2** – Results of ASR model trained with noise embedded augmented audio data set combined with [5].

### Training SR Models

Training of deep learning models can be performed end-to-end, meaning the entire process from audio input to text output is optimized simultaneously. This contrasts with traditional methods, which often required separate models for different stages of the process (e.g., feature extraction, pattern recognition). End-to-end training simplifies the development pipeline, allowing for faster iteration and improvement of speech recognition systems [1].

In addition, training models on noisy data can have a significant effect on accuracy and usefulness of ASR models. Unfortunately, there is a limited amount of noisy data for training even for the high-resource languages that have a large speech corpora. However, as other have demonstrated, training with data augmented data sets embedded with noise can be one possible solution to creating or generating noisy data sets to significantly reduce word error rates exhibited with inferential models [1,5].

Fortunately, there is now a few prominent noise-robust or noisy audio training sets for use in existing training data sets including [1]:

DeepSpeech – Developed by Mozilla, DeepSpeech is an open-source speech recognition model that incorporates noise robustness techniques into the training set.

Google's ASR (Automatic Speech Recognition) System – Google ASR is used in well known ASR models such as Google Assistant and Google Translate and is known for its ability to function well in noisy environments.

Kaldi ASR – An open-sourced toolkit that is widely used in academic and industrial research which includes noise-robust features for training models in noisy environments.

NeMo (Neural Modules) ASR - Is an open-sourced NVIDIA end-to-end ASR toolkit for developing conversational AI models. Lab 6 utilizes this toolkit specifically to fine-tune a pre-trained model to improve noise robustness, and demonstrate the efficient way to train for various noisy conditions [7].



## End-to-End ASR Models

End-to-end Automatic Speech Recognition (ASR) models represent a significant advancement over traditional methods. Unlike traditional systems that break down the process into multiple steps, end-to-end models directly map raw audio inputs to text outputs, streamlining the entire process. This integration leads to improved accuracy and efficiency, as the model can learn to optimize all aspects of the task simultaneously [1]. In addition, end-to-end ASR models simplify speech recognition pipeline by eliminating the need for multiple or separate models by utilizing the following techniques:

### End-to-end ASR models offer several advantages over traditional methods, including [1]:

Direct Mapping: By directly mapping raw audio to text, end-to-end models eliminate the need for intermediate representations, simplifying the system and allowing for simultaneous optimization.

Unified Model Architecture: The use of a single neural network architecture streamlines the process and reduces complexity.

Reduced Dependency on Handcrafted Features: End-to-end models can learn relevant features directly from raw audio data, reducing reliance on handcrafted features like MFCCs.

Simplified Training Process: The integrated approach eliminates the need for training and fine-tuning multiple separate components, leading to faster development and deployment.

Improved Performance: End-to-end models often achieve better recognition accuracy and are more robust to variations in speech and background noise.

## Scalability in SR models

Finally, deep learning models are highly scalable and can be adapted to different languages, dialects, and domains with relative ease. This flexibility is essential for building speech recognition systems that can be deployed in various contexts and cater to diverse user needs. By leveraging the power of deep learning, speech recognition systems can be made more accessible and inclusive, breaking down language barriers and enabling communication across different cultures [1].

Improved Accuracy: Deep learning models, especially those using architectures like LSTMs and transformers, can capture more complex patterns in speech data, leading to higher accuracy in recognition and transcription.

Handling Variability: Deep learning models are more robust to variations in accents, speaking styles, and noisy environments, making them more adaptable to different speech inputs.

End-to-End Training: Deep learning models can be trained end-to-end, simplifying the development process and allowing for faster iteration and improvement.

Scalability and Flexibility: Deep learning models are highly scalable and can be adapted to different languages, dialects, and domains with relative ease.

These advantages have enabled AI-driven speech recognition systems to achieve remarkable accuracy and versatility, transforming the way we interact with technology.

## Lab 6 – Speech Recognition with End-to-End ASR

**Code Reference:** Google Colab. (n.d.).

[https://colab.research.google.com/github/NVIDIA/NeMo/blob/stable/tutorials/asr/ASR\\_with\\_NeMo.ipynb](https://colab.research.google.com/github/NVIDIA/NeMo/blob/stable/tutorials/asr/ASR_with_NeMo.ipynb) [8]

### The process for the Transformer Classification:

- Step 1 – Load the Python Libraries huggingface\_hub and NeMo toolkit ASR from Github
- Step 2 – Create Path to data directory
- Step 3 – Load AN4 Dataset
- Step 4 – Convert to .wav Audio Files
- Step 5 – Verify Dataset Loaded & Converted (display audio waveform)
- Step 6 – Install NeMo ASR Toolkit
- Step 7 – Instantiate QuartzNet ASR Model
- Step 8 – Create Training/Evaluation Manifest of Metadata/Descriptions
- Step 9 – Load YAML Model Parameter Configurations
- Step 10 – Train Model with PyTorch Lightning
- Step 11 – Inference

**Course:** COSC 6370 Advanced Topics in AI

**Student:** Matthew Harper

**Assignment:** Lab 6 - End-to-End ASR with NeMo Toolkit

**Reference:** [https://colab.research.google.com/github/NVIDIA/NeMo/blob/stable/tutorials/asr/ASR\\_with\\_NeMo.ipynb](https://colab.research.google.com/github/NVIDIA/NeMo/blob/stable/tutorials/asr/ASR_with_NeMo.ipynb)

### Step 1 – Load the Python Libraries huggingface\_hub and NeMo toolkit ASR from Github

The first of the lab is to import the required libraries.

Revert to the 0.23.2 version of huggingface\_hub for this execute properly

```
#Need to install or revert to the correct revision of huggingface_hub 0.23.2
!pip install huggingface_hub==0.23.2
```

Install the NVIDIA NeMo github repository for nemo\_toolkit

```
[ ] #Install the NeMo github repository for nemo toolkit and change [asr] to [all] for this to work correctly
    #hash out the exit() as we are using notebook and will not need to restart the kernel
    """
    You can run either this notebook locally (if you have all the dependencies and a GPU) or on Google Colab.

    Instructions for setting up Colab are as follows:
    1. Open a new Python 3 notebook.
    2. Import this notebook from GitHub (File -> Upload Notebook -> "GITHUB" tab -> copy/paste GitHub URL)
    3. Connect to an instance with a GPU (Runtime -> Change runtime type -> select "GPU" for hardware accelerator)
    4. Run this cell to set up dependencies.
    5. Restart the runtime (Runtime -> Restart Runtime) for any upgraded packages to take effect

    NOTE: User is responsible for checking the content of datasets and the applicable licenses and determining if suitable
    """
    # If you're using Google Colab and not running locally, run this cell.

    ## Install dependencies
    !pip install wget
    !apt-get install sox libsndfile1 ffmpeg
    !pip install text-unidecode
    !pip install matplotlib>=3.3.2

    ## Install NeMo
    BRANCH = 'r2.0.0rc0'
    !python -m pip install git+https://github.com/NVIDIA/NeMo.git@$BRANCH#egg=nemo_toolkit[all]

    """
    Remember to restart the runtime for the kernel to pick up any upgraded packages (e.g. matplotlib)!
    Alternatively, you can uncomment the exit() below to crash and restart the kernel, in the case
    that you want to use the "Run All Cells" (or similar) option.
    """
    # exit()
```

```
[ ] x python setup.py bdist_wheel did not run successfully.
    exit code: 1
    -> See above for output.

    note: This error originates from a subprocess, and is likely not a problem with pip.
    Building wheel for causal-conv1d (setup.py) ... error
    ERROR: Failed building wheel for causal-conv1d
    Running setup.py clean for causal-conv1d
    Successfully built nemo_toolkit
    Failed to build causal-conv1d
    ERROR: ERROR: Failed to build installable wheels for some pyproject.toml based projects (causal-conv1d)
    '\nRemember to restart the runtime for the kernel to pick up any upgraded packages (e.g. matplotlib)!'\n
    , in the case\nthat you want to use the "Run All Cells" (or similar) option.\n'
```

Because the causal convolution failed to install we will install it manually as this is required for the audio file encoding.

```
#Error received for causal 1D convolution so we need to install this manually
#This is important and required for the convolution of the audio data set files
!pip install causal-conv1d
```

```
Requirement already satisfied: causal-conv1d in /usr/local/lib/python3.10/dist-packages (1.4.0)
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages (from causal-conv1d) (2.4.1+cu121)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from causal-conv1d) (24.1)
Requirement already satisfied: ninja in /usr/local/lib/python3.10/dist-packages (from causal-conv1d) (1.11.1.1)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch->causal-conv1d) (3.16.1)
Requirement already satisfied: typing-extensions>=4.8.0 in /usr/local/lib/python3.10/dist-packages (from torch->causal-conv1d) (4.12.2)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch->causal-conv1d) (1.13.3)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch->causal-conv1d) (3.3)
Requirement already satisfied: Jinja2 in /usr/local/lib/python3.10/dist-packages (from torch->causal-conv1d) (3.1.4)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch->causal-conv1d) (2024.6.1)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from Jinja2->torch->causal-conv1d) (2.1.5)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from sympy->torch->causal-conv1d) (1.3.0)
```

## Step 2 – Create Path to data directory

Before loading the AN4 dataset we need to set path and the data directory path so we can download the dataset into the correct location directory.

```
[ ] #Import operating system library and create a data directory for path % cd if needed

import os
# This is where the an4/ directory will be placed.
# Change this if you don't want the data to be extracted in the current directory.
data_dir = '.'

if not os.path.exists(data_dir):
    os.makedirs(data_dir)
```

## Step 3 – Load AN4 Dataset

The AN4 dataset is a small dataset recorded and distributed by Carnegie Mellon University, and

### ✦ Taking a Look at Our Data (AN4)

The AN4 dataset, also known as the Alphanumeric dataset, was collected and published by Carnegie Mellon University. It consists of recordings of people spelling out addresses, names, telephone numbers, etc., one letter or number at a time, as well as their corresponding transcripts. We choose to use AN4 for this tutorial because it is relatively small, with 948 training and 130 test utterances, and so it trains quickly.

Before we get started, let's download and prepare the dataset. The utterances are available as `.sph` files, so we will need to convert them to `.wav` for later processing. If you are not using Google Colab, please make sure you have [Sox](#) installed for this step—see the "Downloads" section of the linked Sox homepage. (If you are using Google Colab, Sox should have already been installed in the setup cell at the beginning.)

consists of recordings of people spelling out addresses, names, etc.

## Step 4 – Convert to .wav Audio Files

We now download the dataset and will use glob to convert the data files into .wav audio files from .sph.

```
[ ] #Use glob to convert audio files to .wav files

import glob
import subprocess
import tarfile
import wget

# Download the dataset. This will take a few moments...
print("*****")
if not os.path.exists(data_dir + '/an4_sphere.tar.gz'):
    an4_url = 'https://dldata-public.s3.us-east-2.amazonaws.com/an4_sphere.tar.gz'
    an4_path = wget.download(an4_url, data_dir)
    print(f"Dataset downloaded at: {an4_path}")
else:
    print("Tarfile already exists.")
    an4_path = data_dir + '/an4_sphere.tar.gz'

if not os.path.exists(data_dir + '/an4/'):
    # Untar and convert .sph to .wav (using sox)
    tar = tarfile.open(an4_path)
    tar.extractall(path=data_dir)

    print("Converting .sph to .wav...")
    sph_list = glob.glob(data_dir + '/an4/**/*.sph', recursive=True)
    for sph_path in sph_list:
        wav_path = sph_path[:-4] + '.wav'
        cmd = ["sox", sph_path, wav_path]
        subprocess.run(cmd)
    print("Finished conversion.\n*****")
```



```
*****
Tarfile already exists.
Finished conversion.
*****
```

## Step 5 – Verify Dataset Loaded & Converted (display audio waveform)

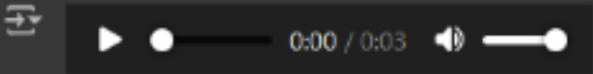
To verify the files were successfully downloaded and converted lets load and decode a .wav audio file and then play the file with ipd.Audio in which the letter are G L E N N.

```
[ ] #Import librosa to load and decodes the audio as a time series with sample rate

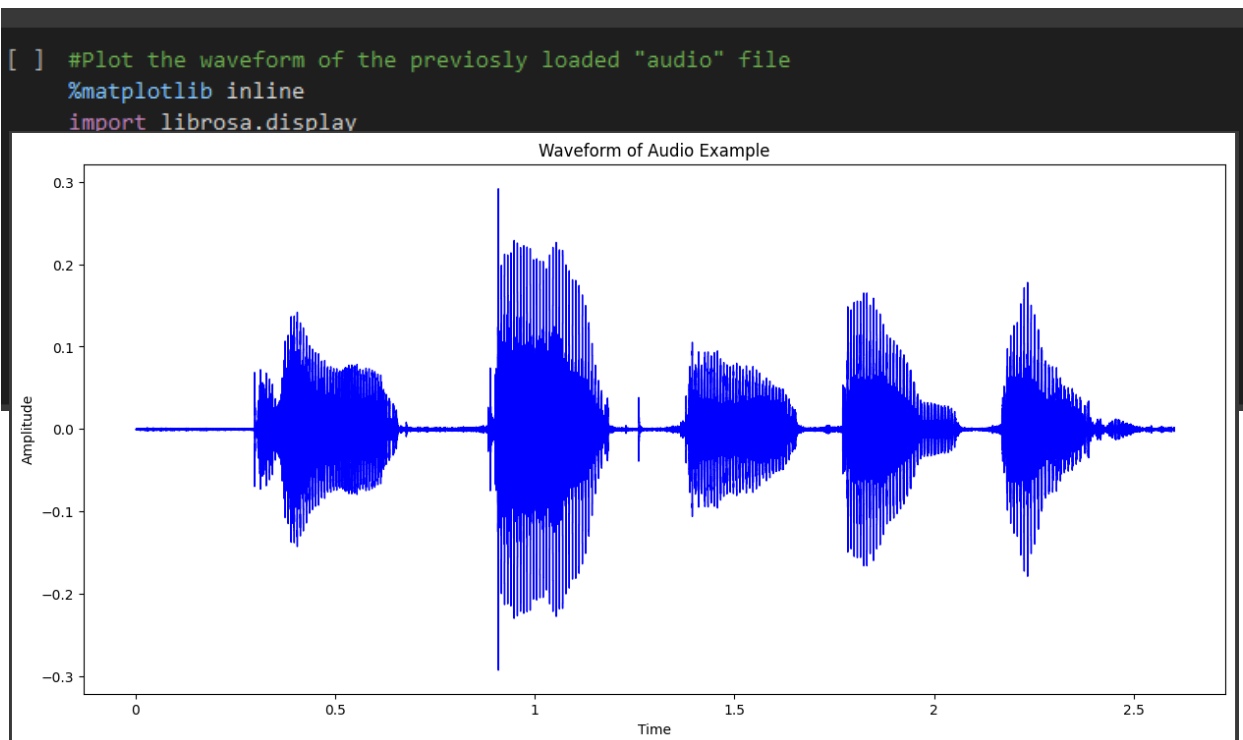
import librosa
import IPython.display as ipd

# Load and listen to the audio file
example_file = data_dir + '/an4/wav/an4_clstk/mgah/cen2-mgah-b.wav'
audio, sample_rate = librosa.load(example_file)

ipd.Audio(example_file, rate=sample_rate)
```



To visualize this we can also display or plot the audio waveform with the `librosa.display.waveshow` where `matplotlib` is for the borders and figure size container.

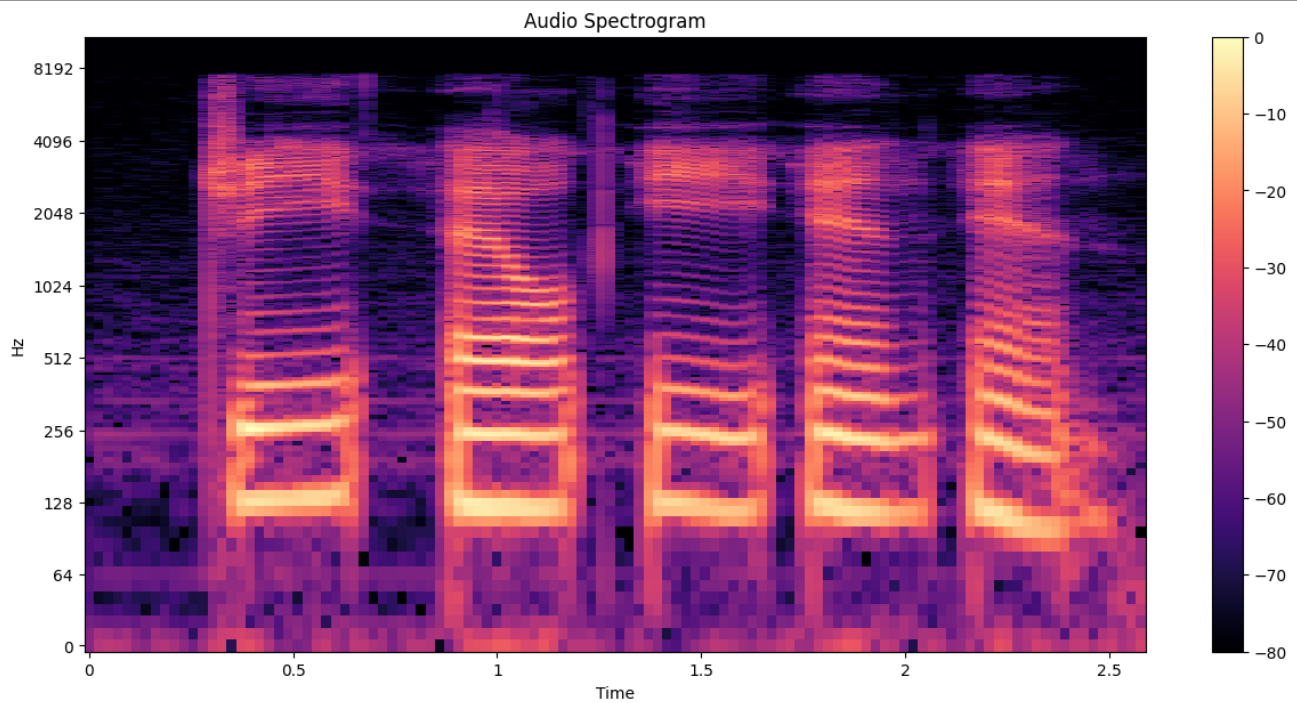


```
#create and display the fourier transform of the audio file

import numpy as np

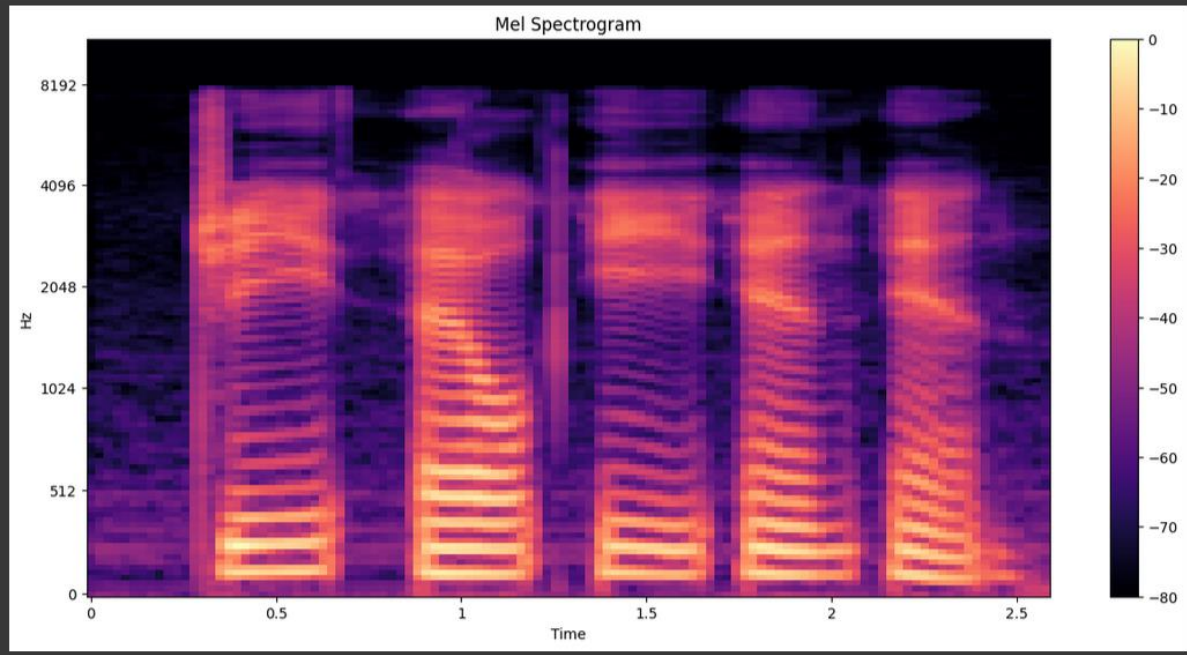
# Get spectrogram using Librosa's Short-Time Fourier Transform (stft)
spec = np.abs(librosa.stft(audio))
spec_db = librosa.amplitude_to_db(spec, ref=np.max) # Decibels

# Use log scale to view frequencies
librosa.display.specshow(spec_db, y_axis='log', x_axis='time')
plt.colorbar()
plt.title('Audio Spectrogram');
```



```
# Plot the mel spectrogram of our sample
mel_spec = librosa.feature.melspectrogram(y=audio, sr=sample_rate)
mel_spec_db = librosa.power_to_db(mel_spec, ref=np.max)

librosa.display.specshow(
    mel_spec_db, x_axis='time', y_axis='mel')
plt.colorbar()
plt.title('Mel Spectrogram');
```



## Convolutional ASR Models

This lab uses both the Jasper and QuartzNet models which consist of the following layers:

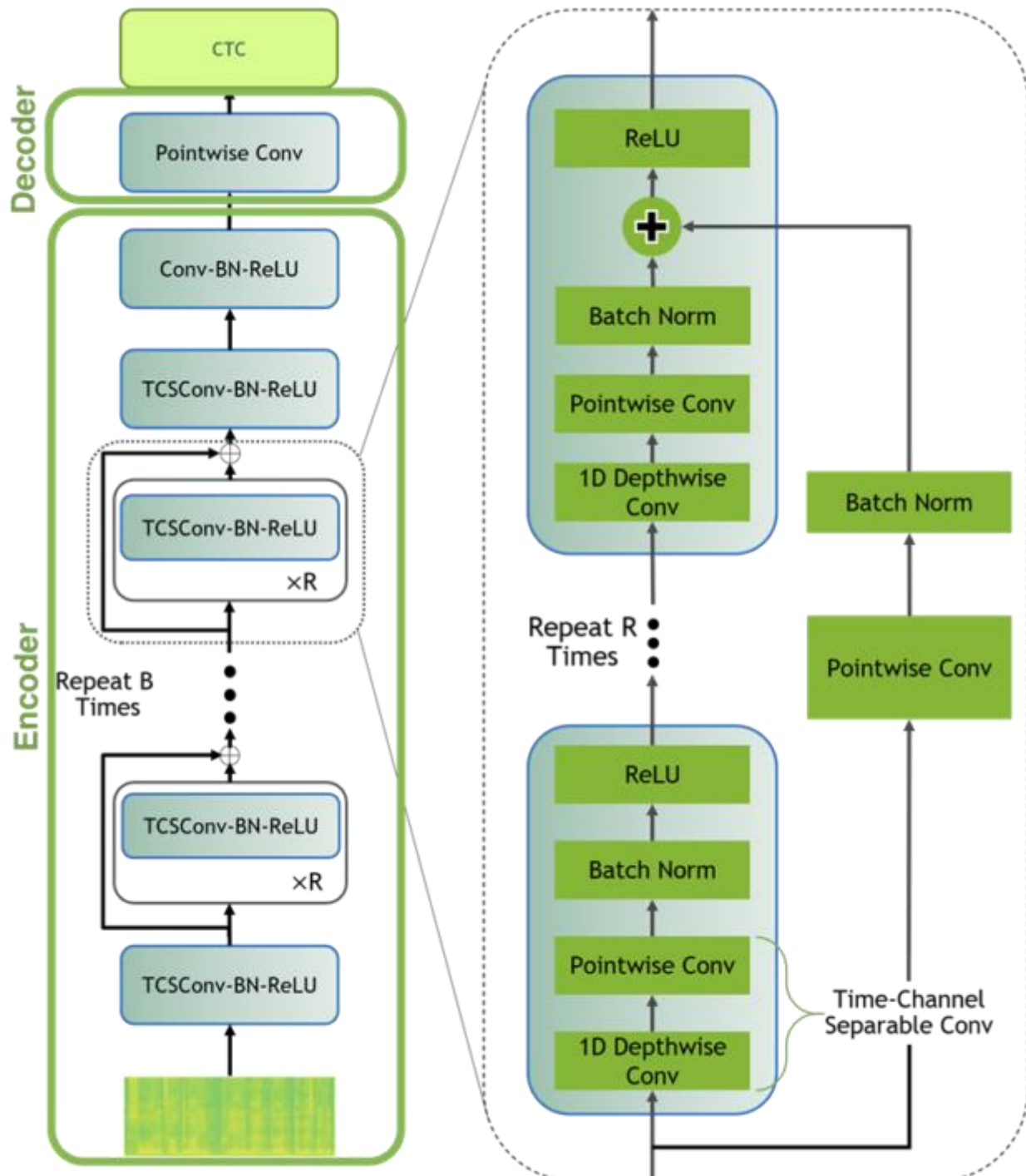
- 1D Convolution
- Batch Normalization
- Rectified Linear Unit (ReLU)
- Dropout

Lab training utilizes a small Jasper (Just Another SPeech Recognizer) model from scratch (e.g. initialized randomly). In brief, Jasper architectures consist of a repeated block structure that utilizes 1D convolutions. In a Jasper\_KxR model, R sub-blocks (consisting of a 1D convolution, batch norm, ReLU, and dropout) are grouped into a single block, which is then repeated K times. We also have a one extra block at the beginning and a few more at the end that are invariant of K and R, and we use CTC loss.

### The QuartzNet Model



The QuartzNet is better variant of Jasper with a key difference that it uses time-channel separable 1D convolutions. This allows it to dramatically reduce number of weights while keeping similar accuracy.



## Step 6 – Install NeMo ASR Toolkit

Install the full Neural Modules (NeMo) toolkit

```
[ ] #Install the full nemo toolkit for ASR
!pip install nemo_toolkit[all]
```

```
[ ] # NeMo's "core" package
import nemo
# NeMo's ASR collection - this collections contains complete ASR models and
# building blocks (modules) for ASR
import nemo.collections.asr as nemo_asr
```

## Step 7 – Insantiate QuartzNet ASR Model

Instantiate the QuartzNet Model as follows:

```
# This line will download pre-trained QuartzNet15x5 model from NVIDIA's NGC cloud and instantiate it for you
quartznet = nemo_asr.models.EncDecCTCModel.from_pretrained(model_name="QuartzNet15x5Base-En")

[NeMo I 2024-10-07 03:23:04 cloud:58] Found existing object /root/.cache/torch/NeMo/NeMo_1.23.0/QuartzNet15x5Ba
[NeMo I 2024-10-07 03:23:04 cloud:64] Re-using file from: /root/.cache/torch/NeMo/NeMo_1.23.0/QuartzNet15x5Base
[NeMo I 2024-10-07 03:23:04 common:924] Instantiating model from pre-trained checkpoint
[NeMo I 2024-10-07 03:23:04 features:289] PADDING: 16
[NeMo W 2024-10-07 03:23:05 nemo_logging:349] /usr/local/lib/python3.10/dist-packages/nemo/core/connectors/save
```

Add the paths to the audio files we want to transcribe from the data directory path.

```
[ ] #Create a path to files we want to transcribe into the list and pass to the model
#change the audio=files to paths2audio_files=files
files = [os.path.join(data_dir, 'an4/wav/an4_clstk/mgah/cen2-mgah-b.wav')]
for fname, transcription in zip(files, quartznet.transcribe(paths2audio_files=files)): # Changed 'audio' to 'paths2audio_files'
    print(f"Audio in {fname} was recognized as: {transcription}")

Transcribing: 100% 1/1 [00:04<00:00, 4.41s/it]
[NeMo W 2024-10-07 03:24:12 nemo_logging:349] /usr/local/lib/python3.10/dist-packages/nemo/collections/asr/parts/preprocessing/fe
with torch.cuda.amp.autocast(enabled=False):

Audio in ./an4/wav/an4_clstk/mgah/cen2-mgah-b.wav was recognized as: g l e n n
```

## Step 8 – Create Training/Evaluation Manifest of Metadata/Descriptions

The next step is to create training descriptions or metadata of our training audio files for training and validation or WER.

In NeMo the metadata manifest must include the path to the audio file, duration of audio file, and the transcript as follows:

```
{"audio_filepath": "path/to/audio.wav", "duration": 3.45, "text": "this is a nemo tutorial"}
```

Use the following code script to build training and evaluation manifests for the audio files.

```
# --- Building Manifest Files --- #
import json

# Function to build a manifest
def build_manifest(transcripts_path, manifest_path, wav_path):
    with open(transcripts_path, 'r') as fin:
        with open(manifest_path, 'w') as fout:
            for line in fin:
                # Lines look like this:
                # <s> transcript </s> (fileID)
                transcript = line[: line.find('(')-1].lower()
                transcript = transcript.replace('<s>', '').replace('</s>', '')
                transcript = transcript.strip()

                file_id = line[line.find('(')+1 : -2] # e.g. "cen4-fash-b"
                audio_path = os.path.join(
                    data_dir, wav_path,
                    file_id[file_id.find('-')+1 : file_id.rfind('-')],
                    file_id + '.wav')

                duration = librosa.core.get_duration(filename=audio_path)

                # Write the metadata to the manifest
                metadata = {
                    "audio_filepath": audio_path,
                    "duration": duration,
                    "text": transcript
                }
                json.dump(metadata, fout)
                fout.write('\n')

# Building Manifests
print("*****")
train_transcripts = data_dir + '/an4/etc/an4_train.transcription'
train_manifest = data_dir + '/an4/train_manifest.json'
if not os.path.isfile(train_manifest):
    build_manifest(train_transcripts, train_manifest, 'an4/wav/an4_clstk')
    print("Training manifest created.")

test_transcripts = data_dir + '/an4/etc/an4_test.transcription'
test_manifest = data_dir + '/an4/test_manifest.json'
if not os.path.isfile(test_manifest):
    build_manifest(test_transcripts, test_manifest, 'an4/wav/an4test_clstk')
    print("Test manifest created.")
print("****Done****")
```

## Step 9 – Load YAML Model Parameter Configurations

Next we load the YAML configuration file for the model to the configuration path. YAML is commonly used for extension configuration files and applications where data is being stored or transmitted. It essentially allows for easier configurations of model parameters.

```
# --- Config Information ---#
try:
    from ruamel.yaml import YAML
except ModuleNotFoundError:
    from ruamel_yaml import YAML
config_path = './configs/config.yaml'

if not os.path.exists(config_path):
    # Grab the config we'll use in this example
    BRANCH = 'r2.0.0rc0'
    !mkdir configs
    !wget -P configs/ https://raw.githubusercontent.com/NVIDIA/NeMo/$BRANCH/examples/asr/conf/config.yaml

yaml = YAML(typ='safe')
with open(config_path) as f:
    params = yaml.load(f)
print(params)
```

## Step 10 – Train Model with PyTorch Lightning

NeMo models are based on PyTorch Lightning module and makes training and fine-tuning easier for mixed precision and distributed training.

```
[ ] import pytorch_lightning as pl
    trainer = pl.Trainer(devices=1, accelerator='gpu', max_epochs=50)
```

Prior to training we need to instantiate the ASR model based on the previously configured parameters with config.yaml.

```
from omegaconf import DictConfig
params['model']['train_ds']['manifest_filepath'] = train_manifest
params['model']['validation_ds']['manifest_filepath'] = test_manifest
first_asr_model = nemo_asr.models.EncDecCTCModel(cfg=DictConfig(params['model']), trainer=trainer)
```

Start Training with the `trainer.fit()` method. Recall and observe that we assigned the trainer to 50 epochs.

```
[ ] # Start training!!!
trainer.fit(first_asr_model)

max_steps: 1500
)
INFO:pytorch_lightning.callbacks.model_summary:
| Name | Type | Params
-----
0 | preprocessor | AudioToMelSpectrogramPreprocessor | 0
1 | encoder | ConvASREncoder | 1.2 M
2 | decoder | ConvASRDecoder | 29.7 K
3 | loss | CTCLoss | 0
4 | spec_augmentation | SpectrogramAugmentation | 0
5 | wer | WER | 0
-----
1.2 M Trainable params
0 Non-trainable params
1.2 M Total params
4.836 Total estimated model params size (MB)

[NeMo W 2024-10-07 03:25:59 nemo_logging:349] /usr/local/lib/python3.10/dist-packa
warnings.warn(_create_warning_msg(

[NeMo W 2024-10-07 03:26:01 nemo_logging:349] /usr/local/lib/python3.10/dist-packa
with torch.cuda.amp.autocast(enabled=False):

[NeMo W 2024-10-07 03:26:01 nemo_logging:349] /usr/local/lib/python3.10/dist-packa
with torch.cuda.amp.autocast(enabled=False):

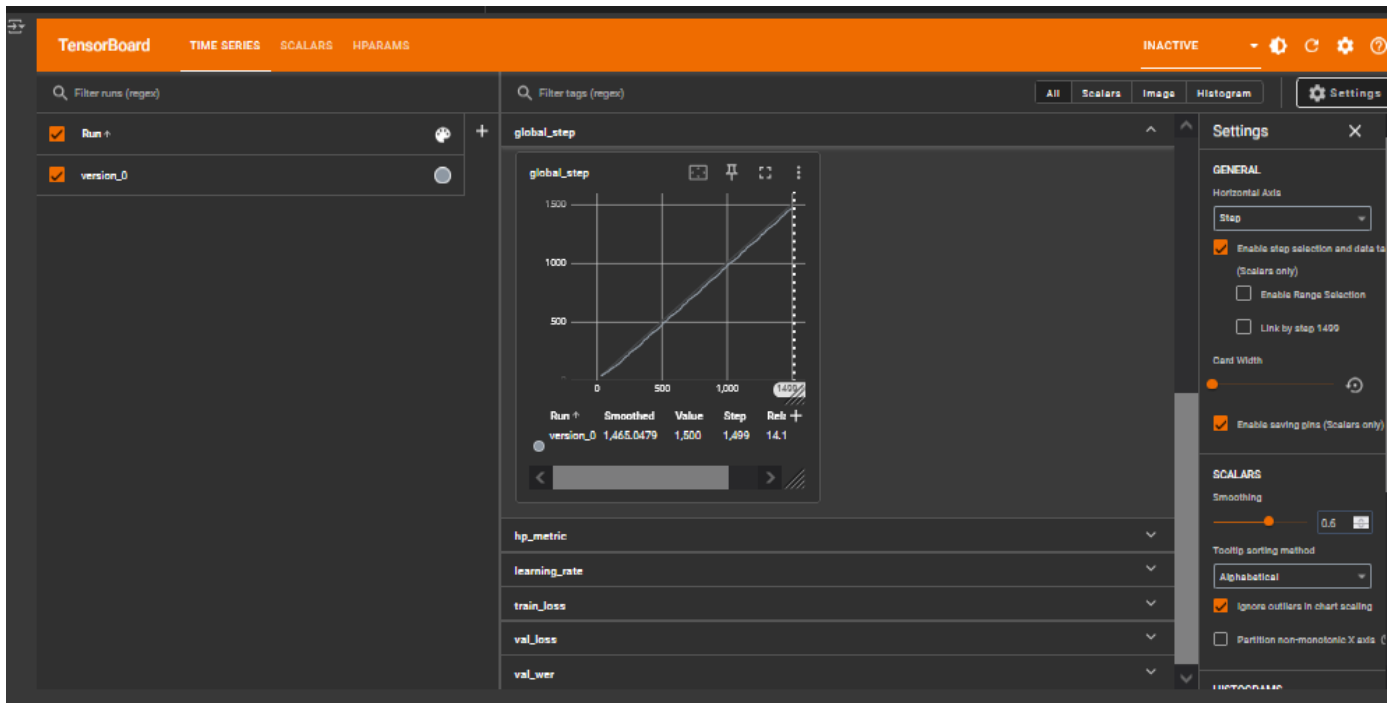
[NeMo W 2024-10-07 03:26:04 nemo_logging:349] /usr/local/lib/python3.10/dist-packa
rank_zero_warn(

Epoch 49: 100% 
```

Training progress and results can be visualized with the Tensorboard (Dashboard) as follows.

```
[ ] #OPTIONAL - to start a tensor board we can monitor training
try:
    from google import colab
    COLAB_ENV = True
except (ImportError, ModuleNotFoundError):
    COLAB_ENV = False

# Load the TensorBoard notebook extension
if COLAB_ENV:
    %load_ext tensorboard
    %tensorboard --logdir lightning_logs/
else:
    print("To use tensorboard, please use this notebook in a Google Colab environment.")
```



## Step 11 – Inference

Now its time for Inference by assigning the audio files paths for transcription

also has `batch_size` argument to improve performance.

```
[ ] #Now we are ready to use our model "Inference" by calling the 4 audio files from path for analysis

audio = [os.path.join(data_dir, 'an4/wav/an4_clstk/mgah/cen2-mgah-b.wav'),
          os.path.join(data_dir, 'an4/wav/an4_clstk/fmjd/cen7-fmjd-b.wav'),
          os.path.join(data_dir, 'an4/wav/an4_clstk/fmjd/cen8-fmjd-b.wav'),
          os.path.join(data_dir, 'an4/wav/an4_clstk/fkai/cen8-fkai-b.wav')]
print(first_asr_model.transcribe(paths2audio_files=audio,
                                 batch_size=4))
```



Transcribing: 100%  1/1 [00:00<00:00, 1.02IVs]

```
[NeMo W 2024-10-07 03:42:46 nemo_logging:349] /usr/local/lib/python3.10/dist-packages/nemo/collections/asr/parts/
with torch.cuda.amp.autocast(enabled=False):
```

```
[NeMo W 2024-10-07 03:42:46 nemo_logging:349] /usr/local/lib/python3.10/dist-packages/nemo/collections/asr/parts/
with torch.cuda.amp.autocast(enabled=False):
```

```
[' e ', ' fi s', ' f oe', ' s']
```

Now we loop through the inference batch and print the Word Error Rate as the final result.

```
# Print the WER for the previous inference set

# Bigger batch-size = bigger throughput
params['model']['validation_ds']['batch_size'] = 16

# Setup the test data loader and make sure the model is on GPU
first_asr_model.setup_test_data(test_data_config=params['model']['validation_ds'])
first_asr_model.cuda()
first_asr_model.eval()

# We will be computing Word Error Rate (WER) metric between our hypothesis and predictions
# WER is computed as numerator/denominator.
# We'll gather all the test batches' numerators and denominators.
wer_nums = []
wer_denoms = []

# Loop over all test batches.
# Iterating over the model's `test_dataloader` will give us:
# (audio_signal, audio_signal_length, transcript_tokens, transcript_length)
# See the AudioToCharDataset for more details.
for test_batch in first_asr_model.test_dataloader():
    test_batch = [x.cuda() for x in test_batch]
    targets = test_batch[2]
    targets_lengths = test_batch[3]
    log_probs, encoded_len, greedy_predictions = first_asr_model(
        input_signal=test_batch[0], input_signal_length=test_batch[1]
    )
    # Notice the model has a helper object to compute WER
    first_asr_model.wer.update(predictions=greedy_predictions, predictions_lengths=targets_lengths)
    _, wer_num, wer_denom = first_asr_model.wer.compute()
    first_asr_model.wer.reset()
    wer_nums.append(wer_num.detach().cpu().numpy())
    wer_denoms.append(wer_denom.detach().cpu().numpy())

    # Release tensors from GPU memory
    del test_batch, log_probs, targets, targets_lengths, encoded_len, greedy_predictions

# We need to sum all numerators and denominators first. Then divide.
print(f"WER = {sum(wer_nums)/sum(wer_denoms)}")
```

```
[NeMo I 2024-10-07 03:42:53 audio_to_text_dataset:49] Model level config does not contain `sample_rate`, please
[NeMo I 2024-10-07 03:42:53 audio_to_text_dataset:49] Model level config does not contain `labels`, please exp
[NeMo I 2024-10-07 03:42:53 collections:196] Dataset loaded with 130 files totalling 0.10 hours
[NeMo I 2024-10-07 03:42:53 collections:197] 0 files were filtered totalling 0.00 hours
[NeMo W 2024-10-07 03:42:53 nemo_logging:349] /usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloaders.py:144: UserWarning: The `collate_fn` argument is deprecated and will be removed in a future PyTorch version. Please use the `collate_fn` argument of the `DataLoader` constructor instead.
warnings.warn(_create_warning_msg(

WER = 0.9573091849935317
```

Observe that our model performed extremely poorly with a WER of 96% and needs further training.



**Conclusion**

Overall, this exercise provides students with a comprehensive introduction and tutorial for deep learning speech recognition model development using the NeMo ASR toolkit. The lab provided students with an introductory to integrated end-to-end SR models which can be tuned for use in specific tasks and domains with speech-to-text applications. Results of the lab also demonstrated the effectiveness of the model to encode and decode audio files for processing into textual recognition output files. While the model demonstrated poor performance overall demonstrated the model worked as developed but required further fine-tuning.

## References

- [1] Gohel, Hardik. Multi-modal Models. COSC 6370 – Advanced Topics in AI Module 1-8.  
<https://hpedsi.uh.edu/people/hardik-gohel>Links to an external site.
- [2] Tutorials — NVIDIA NeMo Framework User Guide latest documentation. (n.d.).  
<https://docs.nvidia.com/nemo-framework/user-guide/latest/nemotoolkit/starthere/tutorials.html>
- [3] Speech recognition. (n.d.). <https://www.ibm.com/history/voice-recognition>
- [4] iMerit Technology Services. (2022, March 21). The Past, Present, and Future of Speech-to-Text and AI Transcription. iMerit. <https://imerit.net/blog/the-past-present-and-future-of-speech-to-text-and-ai-transcription-all-una/>
- [5] Noise-Robust automatic speech recognition for industrial and urban environments. (2023, October 16). IEEE Conference Publication | IEEE Xplore. <https://ieeexplore.ieee.org/document/10312708>
- [6] Ibm. (2024, August 13). Speech Recognition. What Is Speech Recognition.  
<https://www.ibm.com/topics/speech-recognition>
- [7] NeMo speech models. (n.d.). <https://catalog.ngc.nvidia.com/orgs/nvidia/models/nemospeechmodels>
- [8] Google Colab. (n.d.-b).  
[https://colab.research.google.com/github/NVIDIA/NeMo/blob/stable/tutorials/asr/ASR\\_with\\_NeMo.ipynb](https://colab.research.google.com/github/NVIDIA/NeMo/blob/stable/tutorials/asr/ASR_with_NeMo.ipynb)