End-To-End Speech AI Pipelines

NVIDIA

Table of Contents

Preface

Who is this e-book series for?

This e-book series is intended for business decision owners and developers within an enterprise who would like to understand the core concepts of speech AI and how to build and deploy applications for different use-cases.

The series consists of three parts:

Part 1: Introduction to Speech AI
Part 2: End-To-End Speech AI Pipelines
Part 3: Building Speech AI Applications

Over the course of this e-book series, you will learn about:

- Speech AI and its major building blocks.
- The role of speech AI in industries.
- Different components and technologies that are involved in building end-to-end speech AI pipelines
- How to get started with speech AI in your business with NVIDIA Riva.

Part 2: End-To-End Speech AI Pipelines

In Part 1 of this e-book series, Introduction to Speech AI, we explored how speech AI applications can be leveraged in our day-to-day activities and at business enterprises, and we discussed the challenges in building these applications. In this second part of the e-book series, we will explain the different technologies that are involved in building end-to-end speech AI pipelines. An insight into the underlying components that make up the various stages of speech processing can empower developers to customize speech AI systems and solutions to suit their needs.

We will look into the various components in the automatic speech recognition (ASR) and text-to-speech (TTS) pipelines that make up a Speech AI system.

Automatic Speech Recognition

Automatic Speech Recognition (ASR), also known as speech-to-text, refers to the task of getting a program to automatically transcribe spoken language to text. It takes human voice as input and converts it into readable text with the goal of minimizing transcription errors typically measured with a Word Error Rate (WER) metric. In other words, given an audio file (for example, a WAV file) containing speech, the challenge is: how do we transform this into the corresponding text with as few errors as possible?

ASR can be used in popular applications including multi-speaker meeting transcription, understanding voice commands, and automatic call routing, to name a few. Speech recognition has been around for some decades. It dates back to the early 1950s - the first speech recognition system, called *"Audrey" system,* was designed by Bell Laboratories to recognize digits by a single voice. Over the years, speech recognition has advanced from being a U.S. Department of Defense's ARPA (Advanced Research Projects Agency) project in the '70s, to a dictation system in the '90s to a versatile technology today. Almost 71 percent of US consumers prefer to use voice searches to conduct a query over the traditional method of typing. Voice search results are six times more accurate as compared with text-based search.

With the application of artificial intelligence in speech recognition systems and increased speech data, ASR has found its way back to the mainstream. By 2023, it is predicted that customers will prefer to use speech interfaces to initiate 70% of self-service customer interactions, a number rising from 40% in 2019. Deep learning has made ASR less expensive, more accessible, more powerful, and highly accurate.

Traditional Speech Recognition Pipeline

Traditional speech recognition takes a generative approach, modeling the full pipeline of how speech sounds are produced in order to evaluate a speech sample. It starts from a language model that encapsulates the most likely and frequently occurring orderings of words that are generated (e.g., an n-gram model), to a pronunciation model for each word in that ordering (e.g., a pronunciation table), to an acoustic model that translates those pronunciations to audio waveforms (e.g., a Gaussian Mixture Model). Then, given a spoken input, the goal is to find the most likely sequence of text that would result in the given audio according to our generative pipeline of models.

With advances in the capabilities of neural networks, each component of the traditional speech recognition model can be replaced by a neural model that has better performance and greater potential for generalization. For example, an n-gram model can be replaced by a neural language model, a pronunciation table can be replaced by with a neural pronunciation model, and so on. Note that each of these neural network models need to be trained individually on different tasks, and that errors in any model in the pipeline can throw off the whole prediction.

Deep Learning End-To-End Speech Recognition Pipeline

More recently, deep learning has replaced these traditional statistical methods, such as Hidden Markov Models and Gaussian Mixture Models, as it offers better accuracy when identifying phonemes. Recurrent Neural Networks (RNNs) are used to deal with this sequential information - audio data over time that corresponds to a sequence of letters. However, this poses a problem - how do we match each time step from the audio data to the correct output characters given that the input sequence (number of audio timesteps) is not the same length as the desired output (transcript length)?

Earlier approaches to solve this problem relied on temporally aligned data, in which each segment of time in an audio file was matched with a corresponding speech sound such as a phoneme or word and squashed repeated letters in the word for example, “LLLLAAAPPPPTTTOOOOPPPP” becomes “LAPTOP”. However, it turns out that this idea has some problems: not only does alignment make the dataset incredibly labor-intensive to label, it does not work with words like "book" that contain consecutive repeated letters.

Modern end-to-end approaches get around these problems using methods that do not require manual alignment at all; as a result, the input-output pairs are really just the raw audio and the transcript - no extra data or labeling is required. Popular approaches are Connectionist Temporal Classification (CTC) and sequence-to-sequence models with attention.

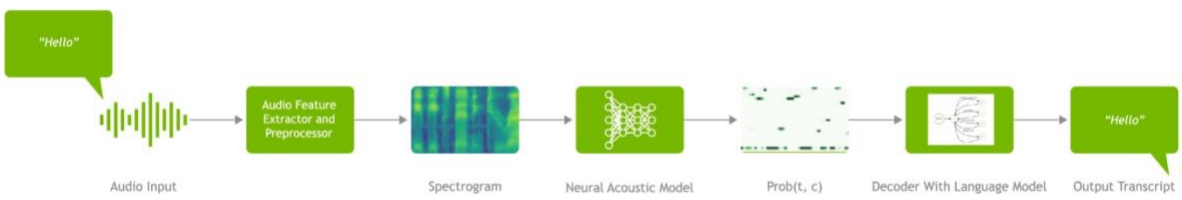

Figure 1: Example of an ASR Pipeline

Audio Preprocessing

The first step of a typical ASR pipeline is to extract useful audio features from the input audio. Audio information is more useful in the context of frequencies of sound over time. We can get a better representation by applying a Fourier Transform on the audio signal to get something more useful: a **spectrogram**, which is a representation of the energy levels (i.e., amplitude, or "loudness") of each frequency (i.e., pitch) of the signal over the duration of the file. A spectrogram, which can be viewed as a heat map, is a good way of visualizing how the strengths of various frequencies in the audio vary over time. The heat map is obtained by breaking up the signal into smaller, usually overlapping, chunks and performing a Short-Time Fourier Transform (STFT) on each chunk.

In practice, the Mel Spectrogram is used instead of a normal spectrogram. The mel spectrogram changes the frequency scale from a linear to a logarithmic scale and produces frequencies that are better suited for human hearing.

In other words, the mel transformation of the frequencies is more aligned to what humans perceive; a change of +1000Hz from 2000Hz->3000Hz is better perceived than a change in the 9000Hz->10000Hz range. We use the mel spectrogram because we are processing and transcribing human speech, and in this frequency range, it transforms the scale to better match what we hear.

Acoustic Model

Spectrograms or Mel Spectrograms from the feature extraction block are passed to a deep learning based acoustic model. The acoustic model is a network that predicts the probability distributions over vocabulary characters (or more generally, text tokens) per each time step, as illustrated in Figure 2. The acoustic model output can contain repeated characters over multiple time steps, based on how a word is pronounced, for example “LLLLAAAPPPPTTTOOOOPPPP”. Thanks to a post processing step, these repeated characters are “collapsed” to make the final words. The acoustic model is trained on datasets such as LibriSpeech ASR Corpus, Mozilla Common Voice datasets, Voxpopuli datasets, each consisting of hundreds to thousands of hours of audio and transcriptions in the target language.

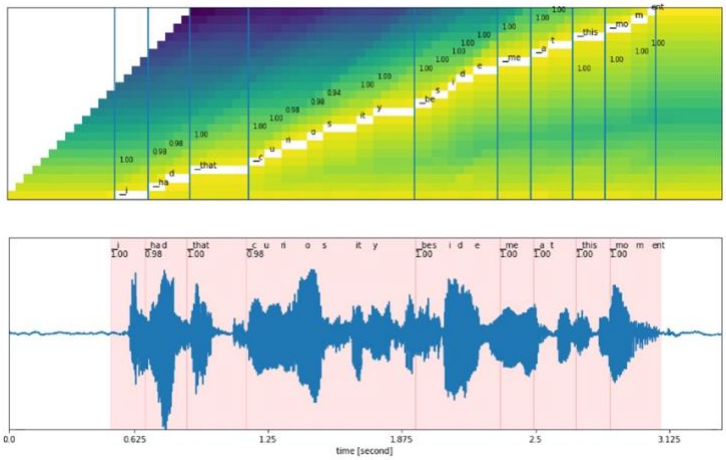

Figure 2: Acoustic Model Output (Source: Moto Hira, “Forced Alignment with WAV2VEC2”)

The output of the acoustic model shown in Figure 2 predicts the probabilities over characters at each time step of the audio signal.

Popular deep learning models for ASR include Wav2letter, Deepspeech, LAS (Listen, Attend, and Spell), and NVIDIA's high performing acoustic models - Jasper, QuartzNet and Citrinet and Conformer-CTC, which belong to a family of CTC-based end-to-end models and can predict a transcript directly from an audio input without additional alignment information.

Jasper ("Just Another Speech Recognizer") is a deep time delay neural network (TDNN) consisting of blocks of 1D-convolutional layers. QuartzNet improves on Jasper by replacing 1D convolutions with 1D time-channel separable convolutions. Citrinet is a version of QuartzNet that is smaller and uses sub-word encoding through word piece tokenization and Squeeze-and-Excitation mechanism to obtain highly accurate audio transcripts while using an autoregressive, CTC-based decoding scheme for efficient inference. The resulting architecture significantly reduces the quality gap between non-autoregressive and sequence-to-sequence and transducer models, making non-autoregressive models more appealing as they are generally more computationally efficient.

Conformer is a recent architecture that combines two prominent architectures: the transformer, which is a powerful and popular architecture used in natural language processing, and convolutional neural network (CNN), which is commonly employed for computer vision. Transformers are good at capturing global interactions thanks to itself attention mechanism, while CNNs are good at exploiting local features. The NVIDIA Conformer-CTC model is a non-autoregressive variant of the Conformer model for ASR which uses CTC loss/decoding instead of a transducer [1].

Decoder and Language Model

The decoder converts the matrix of probabilities over characters at each audio time step into a sequence of words. Greedy decoding, the simplest strategy for a decoder, simply takes the most probable character at each time step. The letter with the highest probability is chosen at each time-step, without regard to any semantic understanding of what was being communicated. Then, the repeated characters are removed or collapsed, and blank tokens are discarded.

Language models have shown to help the accuracy of ASR models. A language model (LM) can produce a score indicating the likelihood of a sentence appearing in its training corpus. For example, an LM trained on an English corpus will judge "Recognize speech" as more likely than "Wreck a nice peach", while also judging "Je suis un étudiant" as quite unlikely, since that is a sentence in French.

A language model can be used to add linguistic prior knowledge and therefore correct mistakes in the acoustic model. For example, an acoustic model might interpret what it hears from the audio as "I have rose beef for lunch", whereas in combination with a language model, it has a better chance to output the correct text as "I have roast beef for lunch", since the latter would have a much higher language model score than the former.

A beam search decoder is an advanced decoder, which can concurrently inspect multiple possible characters at each time step in searching for the best possible overall output, that is, one with the highest combined score given by the acoustic model and the language model.

Punctuation and Capitalization Model

Automatic Speech Recognition (ASR) systems typically generate text with no punctuation and capitalization of the words. Besides being hard to read for humans, this may cause difficulties in other

automated processing steps, as the ASR output could be an input to named entity recognition, machine translation, or text-to-speech models. If the input text has punctuation and words are capitalized correctly, this could potentially boost the performance of such models.

For each word in the input text, the punctuation and capitalization model predict the following:

- a punctuation mark that should follow the word (if any); the model supports commas, periods, and question marks.
- whether the word should be capitalized or not.

(Inverse) Text Normalization Model

Text Normalization converts text from written form into its verbalized form. It is used as a preprocessing step before Text-to-Speech (TTS) and for preprocessing Automatic Speech Recognition (ASR) training transcripts. For example, “at 10:00” generates “at ten o’clock” and “it weighs 10kg.” generates “it weighs ten kilograms”.

In the reverse direction, Inverse Text Normalization (ITN) rules are applied to transform the text in verbal format into a desired written format, for example, “ten o’clock” converts to “10:00”, and “ten dollars” converts to “$10”.

Evaluation of ASR Systems

Word error rate (WER) is a common performance metric of ASR systems. It is defined as the number of errors divided by the total number of spoken words, as follows:
Word Error Rate = (Substitutions + Insertions + Deletions) / Number of Words Spoken

The three types of error are defined as follows:

- Substitution is the error made when a word is replaced, for example “rose” instead of “roast”.
- Insertion is the error made when a word is added but was not said, for example, “wreck a nice” instead of “recognize”.
- Deletion is the error made when a word is omitted from the transcript, for example, “let see” instead of “let me see”.

Character error rate (CER) is another performance metric of ASR systems. CER is similar to WER but operates on characters instead of words. It is most commonly used for languages such as Japanese, Mandarin, and Korean, where a “word” is not separated by a specific marker like spaces in English).

Text-to-Speech (TTS)

Text-to-Speech (TTS) or Speech Synthesis involves transforming the system’s response in the form of text to a phonetic transcription that you can hear. It involves taking the text response generated by the dialog system (which manages the conversation with the user), and converting it to natural-sounding speech. This vocal clarity is achieved using deep neural networks that produce human-like intonation and a clear articulation of words. Some of the open source datasets for TTS are LJ Speech, Nancy, TWEB, and LibriTTS that have a text file associated with the audio.

The Text-to-Speech step is commonly achieved using two different approaches: a two-stage pipeline and an end-to-end pipeline.

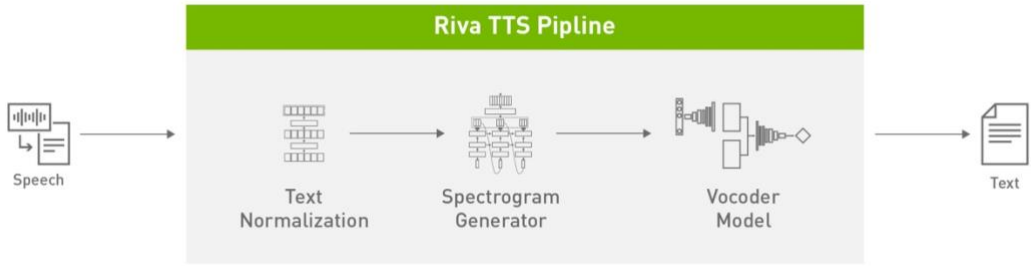

Figure 3: Example of a Text-To-Speech Pipeline

In a two-stage pipeline, two separate networks are trained separately for converting speech-to-text - the Spectrogram generator network and the Vocoder Network. The "end-to-end" approach uses one model to generate audio straight from text. The end-to-end approach is explored by several research works [2], [3], [4]. The network is trained directly from text-audio pairs, without depending on intermediate representations. This approach reduces the cost of manual annotation of duration information and prevents error propagation between networks.

The Two-stage Pipeline

In the two stage pipeline, first, a model is used to generate a mel spectrogram from text. Second, a vocoder model is used to generate audio from a mel spectrogram.

Spectrogram Generator Network

The first stage of the TTS pipeline uses a neural network to generate a spectrogram from text. Preparing the input text for synthesis requires:

- Text analysis, such as converting text into words and sentences
- Text Normalization converts text from written form into its verbalized form, including identifying and expanding abbreviations. Some examples mentioned before are: "at 10:00" generates "at ten o'clock" and "it weighs 10kg." generates "it weighs ten kilograms".
- Recognizing and analyzing expressions.

Expressions include dates, amounts of money, and airport codes. The normalized text is then fed to a speech synthesis neural network model, which converts the text to Mel Spectrograms.

Popular deep learning models for synthesis include autogressive models such as: Tacotron 2, Deep Voice 1 and Deep Voice 2. Other non-autoregressive transformer-based and convolution-based models include: FastSpeech 2, FastPitch, TalkNet and so on. The difference between non-autoregressive models and auto-regressive models is in the decoding of the tokens. Auto-regressive models decode tokens one after another, whereas non-autoregressive models assume token-wise independence at the decoding stage to facilitate parallel implementations, typically resulting in faster inference.

Vocoder Network

The second stage of the TTS pipeline takes the spectrogram from the spectrogram generator network as an input into a neural Vocoder Network that generates a waveform of natural sounding speech from the spectrogram in time domain.

Popular vocoder models include: Wavenet, WaveGlow, UniGlow, HiFiGAN, and MelGAN.

The End-to-End Pipeline

Examples of the end-to-end approach are ClariNet and FastPitch-HifiGan-E2E, which are non-autoregressive models that generate audio from text.

FastPitch-HifiGan-E2E, as the name suggests, combines FastPitch speech synthesis model and HiFiGAN vocoder into one model and is trained jointly in an end-to-end manner.

The FastPitch portion consists of the same transformer-based encoder, pitch predictor, and duration predictor as the original FastPitch model. The HiFiGAN portion takes the discriminator from HiFiGAN and uses it to generate audio from the output of the FastPitch portion. No spectrograms are used in the training of the model. The training losses are taken from HiFiGan plus additional losses for the pitch and duration predictors.

Evaluation of the Pipeline

Mean Opinion Score (MOS) is the most frequently used method to evaluate the quality of TTS systems [5].

The MOS metric has its origins in the telecommunications field and is defined as the arithmetic mean over ratings given by human evaluators for a given stimulus in a subjective quality evaluation test. In TTS, a common evaluation setup is for a group of people to listen to the generated samples, and give each a score from 0 to 5. MOS is then calculated as the average score over all evaluators and test samples. Real human speech scores between 4.5 to 4.8, whereas competitive Deep Learning based TTS systems nowadays routinely score above 4.

Conclusion

Modern state of the art ASR and TTS systems make extensive use of deep neural networks. Each pipeline contains one to several such networks, each consisting of tens of millions to hundreds of millions of parameters. These deep neural networks are trained on massive labeled speech datasets. Recent advances in end-to-end deep learning based approaches have pushed ASR error rates down to a level that makes ASR systems not only practical but also pleasant to use, while pushing TTS quality close to that of the human voice.

In the next part, Part 3 of this e-book series, we will describe how developers within enterprises can develop real-time Speech AI applications and deploy models in production for high-performance inference rapidly and with minimal effort.

References

- [1] Fu-Hao Yu, Kuan-Yu Chen, "Non-autoregressive Transformer-based End-to-End ASR Using BERT", National Taiwan University of Science and Technology, Taiwan.
- [2] Ye Jia, Yu Zhang, Ron J. Weiss, Quan Wang, Jonathan Shen, Fei Ren, Zhifeng Chen, Patrick Nguyen, Ruoming Pang, Ignacio Lopez Moreno, Yonghui Wu, "Transfer Learning from Speaker Verification to Multispeaker Text-To-Speech Synthesis", NeurIPS 2018.
- [3] Dabiao Ma, Zhiba Su, Wenxuan Wang, Yuhao Lu, "FPETS: Fully Parallel End-to-End Text-to-Speech System".
- [4] Wei Ping, Kainan Peng, Jitong Chen, "ClariNet: Parallel Wave Generation in End-to-End Text-to-Speech", ICLR 2019.
- [5] Sergios Karagiannakos, "Speech synthesis: A review of the best text to speech architectures with Deep Learning".