Automated Speech Recognition and Text-to-Speech Synthesis

Natalie Parde

UIC CS 421

Automated Speech Recognition

"RADIO REX" by Leo Reynolds is licensed with CC BY-NC-SA 2.0. To view a copy of this license, visit https://creativecommons.org/licenses/by-nc-sa/2.0/

"Amazon Echo" by adambowie is licensed with CC BY-NC-SA 2.0.

Automated Speech Recognition

I love natural language processing!

What makes speech harder or easier to recognize?

- Vocabulary size
  - Digit recognition: relatively easy
  - Open-ended transcription: much harder
- Speech recipient
  - Human speaking to machines tends to be easier to recognize than humans speaking to other humans
  - Read speech (reading aloud from written content): relatively easy
  - Conversational speech: much harder
- Channel and noise
  - Quiet room with good microphones: relatively easy
  - Noisy environment with distant microphone: much harder
- Speaker-class characteristics
  - Speaker using same dialect as the training data: relatively easy
  - Speaker using different dialect (or from different speaker population, e.g., children): much harder

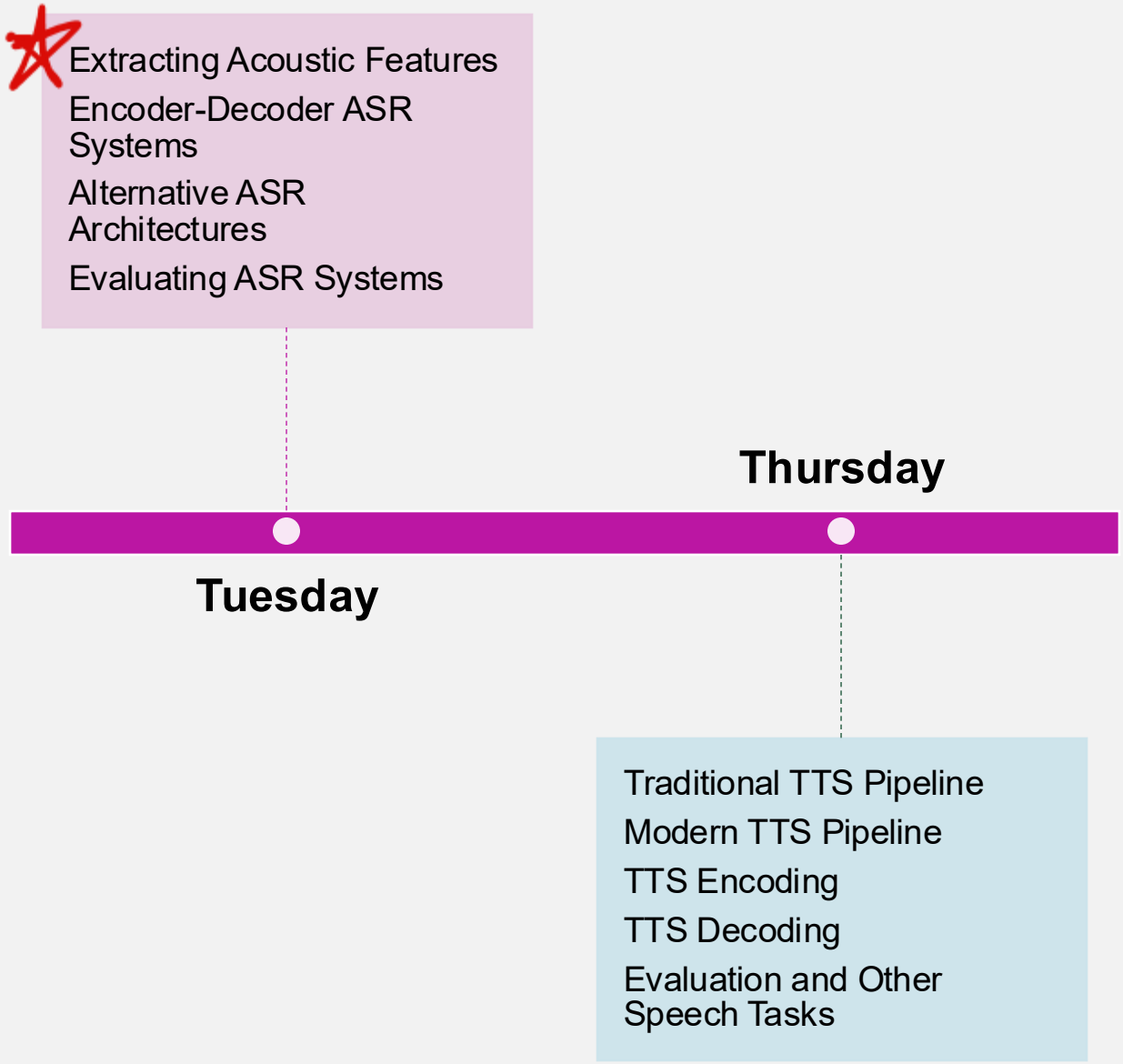
Publicly Available ASR Corpora

- **LibriSpeech:** 1000 hours of English audiobooks
  - https://www.openslr.org/12
- **Switchboard:** 240 hours of English telephone conversations between strangers
  - https://catalog.ldc.upenn.edu/LDC97S62
- **CALLHOME:** 60 hours of English telephone conversations between friends
  - https://ca.talkbank.org/access/CallHome/eng.html
- **Santa Barbara Corpus of Spoken American English:** Conversations between English speakers in a wide range of settings
  - https://www.linguistics.ucsb.edu/research/santa-barbara-corpus
- **CORAAL:** Sociolinguistic interviews with speakers of African American Language
  - http://lingtools.uoregon.edu/coraal/
- **CHiME:** English conversations in difficult or noisy settings
  - https://chimechallenge.github.io/chime6/
- **HKUST:** 200 hours of Mandarin telephone conversations between friends or strangers
  - https://catalog.ldc.upenn.edu/LDC2005S15
- **AISHELL:** 170 hours of Mandarin read speech from various domains
  - https://www.openslr.org/33/

What about multilingual speech corpora?

- Recently much more common!
- Common Voice: 33,150 hours of speech contributed by volunteers from the crowd, spanning 133 languages
  - https://www.kaggle.com/datasets/mozillaorg/common-voice
- FLEURS: Parallel speech dataset built on a machine translation benchmark dataset, with three speakers each for 102 languages reading a subset of 2009 sentences (~12 hours of speech per language)
  - https://huggingface.co/datasets/google/fleurs

This Week's Topics

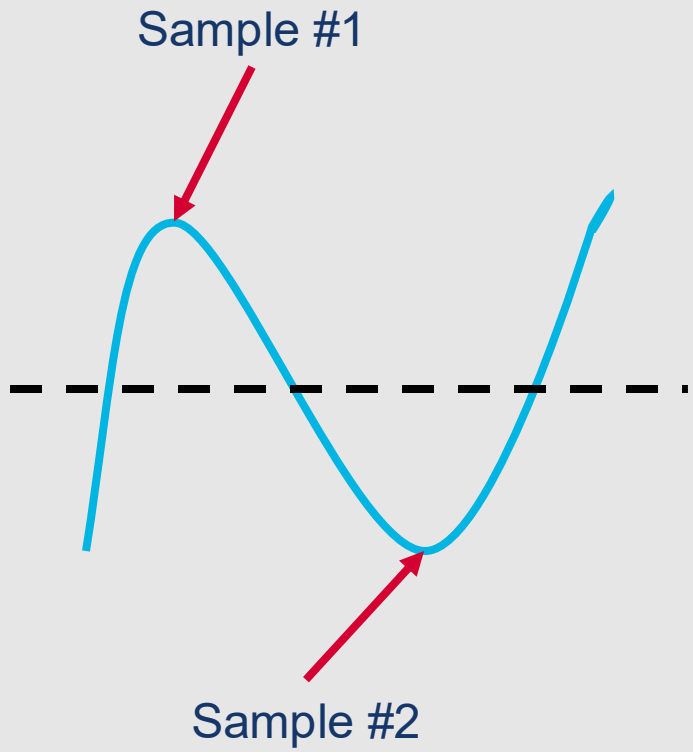


How do we train an ASR system?

- Acoustic waveforms are commonly converted into sequences of acoustic feature vectors through a process of **sampling** and **quantization**
- Each vector represents a small slice of time in the audio sequence
- Common feature type: **Log mel spectrum vectors**

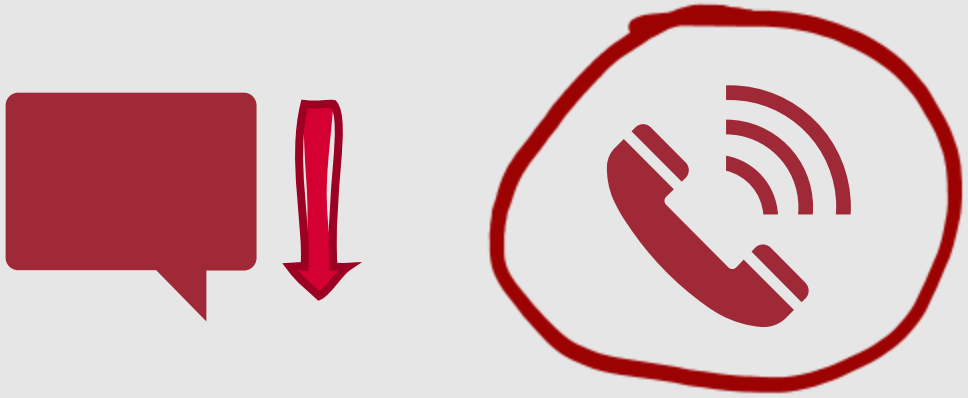
Sampling

- Measure the signal's amplitude at a specific time point
- **Sampling rate:** Number of samples taken per second
- At least two samples per cycle are needed to measure an audio wave (more samples → higher accuracy)
- Maximum frequency wave for a given sampling rate = half the sampling rate
  - Known as the **Nyquist frequency**
    - For example, a sampling rate of 20,000 Hz would be needed for speech at frequencies of ≤ 10,000 Hz
  - 1 Hz = once per second



Sampling

- Can’t combine sampling rates when training the same system!
- Downsample sources with higher sampling rates to match the source with the lowest sampling rate



Common Sampling Rates

- Human Speech
  - ≤ 10,000 Hz
- Telephone Speech
  - ≤ 4,000 Hz

Quantization

- Real-valued amplitudes are represented as integers
- **Quantum size:**
  - 8 bit (-128 through 127)
  - 16 bit (-32,768 through 32,767)
- Values closer together than the specified granularity are represented identically
- Sample amplitude of waveform x at time index n is $x[n]$

Windowing

- Extracting features from a roughly stationary window of the quantized waveform (a **speech frame**)
- Three windowing parameters:
  - **Frame size:** Width of the window in milliseconds
  - **Frame Stride:** Offset between successive windows
  - **Shape:** Behavior of the window near its boundaries
- Windowed waveform at time n, $y[n]$, is obtained by multiplying the signal by the window value at that time
  - $y[n] = w[n]s[n]$

Window Shape

- **Rectangular**
  - Segment of the original signal without any further changes
  - $w[n] = \begin{cases} 1, & 0 \le n \le L-1 \\ 0, & \text{otherwise} \end{cases}$
  - Abrupt cutoffs at start and end points can pose challenges for Fourier analysis (necessary for feature extraction)
- **Hamming**
  - Shrinks the signal values near the start and end points
  - $w[n] = \begin{cases} 0.54 - 0.46\cos(\frac{2\pi n}{L}), & 0 \le n \le L-1 \\ 0, & \text{otherwise} \end{cases}$

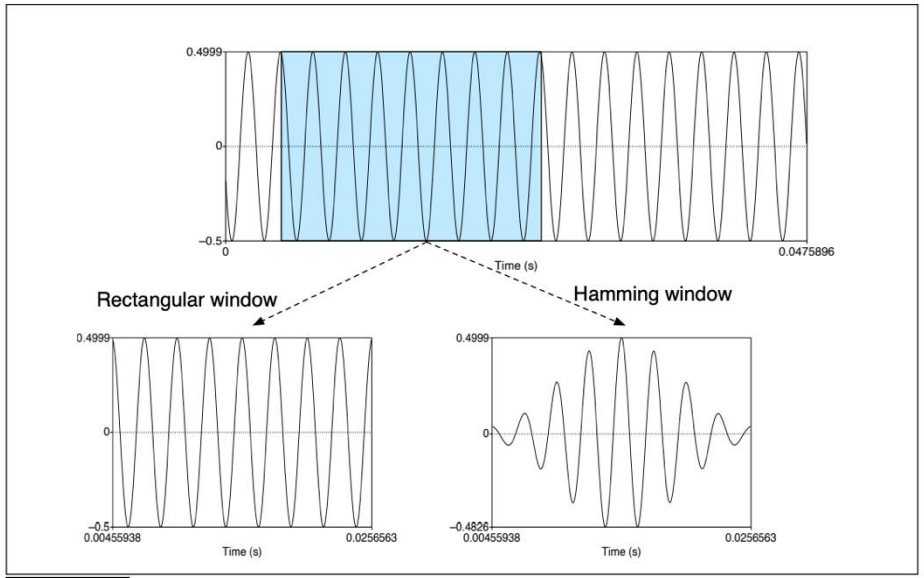


Figure 26.3 Windowing a sine wave with the rectangular or Hamming windows.

Source: Dan Jurafsky and James H. Martin, *Speech and Language Processing*, Chapter 26: https://web.stanford.edu/~jurafsky/slp3/26.pdf

Discrete Fourier Transform

- Extracts **spectral information** (how much energy the signal contains at different frequency bands) from windowed signals
  - $X[k] = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi}{N}kn}$, where k is a given frequency band and $x[n]$ is a windowed signal at time n (j is the imaginary unit from Euler's formula: $e^{j\theta} = \cos(\theta) + j\sin(\theta)$)
- Can be computed using the **fast Fourier transform**
  - Divide-and-conquer approach for computing the discrete Fourier transform

Mel Frequencies

- Humans do not perceive differences in pitch equally across all frequency bands
  - Low frequencies → easy to detect differences in pitch
  - High frequencies → harder to detect differences in pitch
- This can be modeled using the **mel scale**
- **Mel:** Unit of pitch
  - Pairs of sounds that are *perceived* as being equidistant in pitch are positioned equidistant to one another on the mel scale

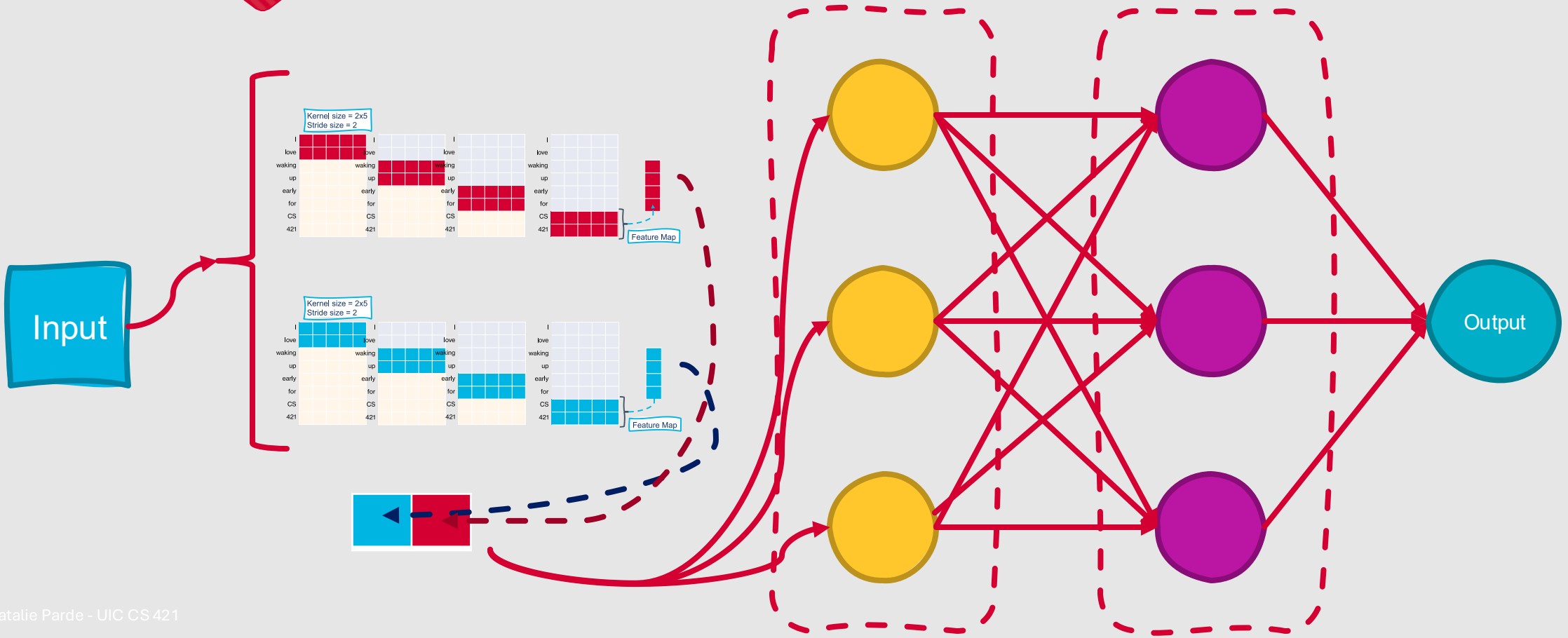
Mel Frequencies

- To convert from raw acoustic frequency to **mel frequency**:
  - $\text{mel}(f) = 1127 \ln\left(1 + \frac{f}{700}\right)$
- Then, compute energy at each mel frequency band
  - Fine-grained resolution at low frequencies
  - Coarser-grained resolution at high frequencies
- Take the log of mel spectrum values
  - Humans are also less sensitive to changes in higher amplitudes
  - Makes feature estimates less sensitive to input variations (e.g., power variations resulting from variations in microphone distance)

Convolutional Neural Networks for Extracting Speech Features

- Input: Audio representation (e.g., Mel spectra or a raw audio signal)
- Output: Sequence of latent (encoded) representations of the input audio
- Popular architectures using CNN features:
  - Whisper
  - wav2vec2.0
  - HuBERT

Recall the CNN architecture….



How do we adapt this for speech?

Slide the kernel over the acoustic signal in the time dimension

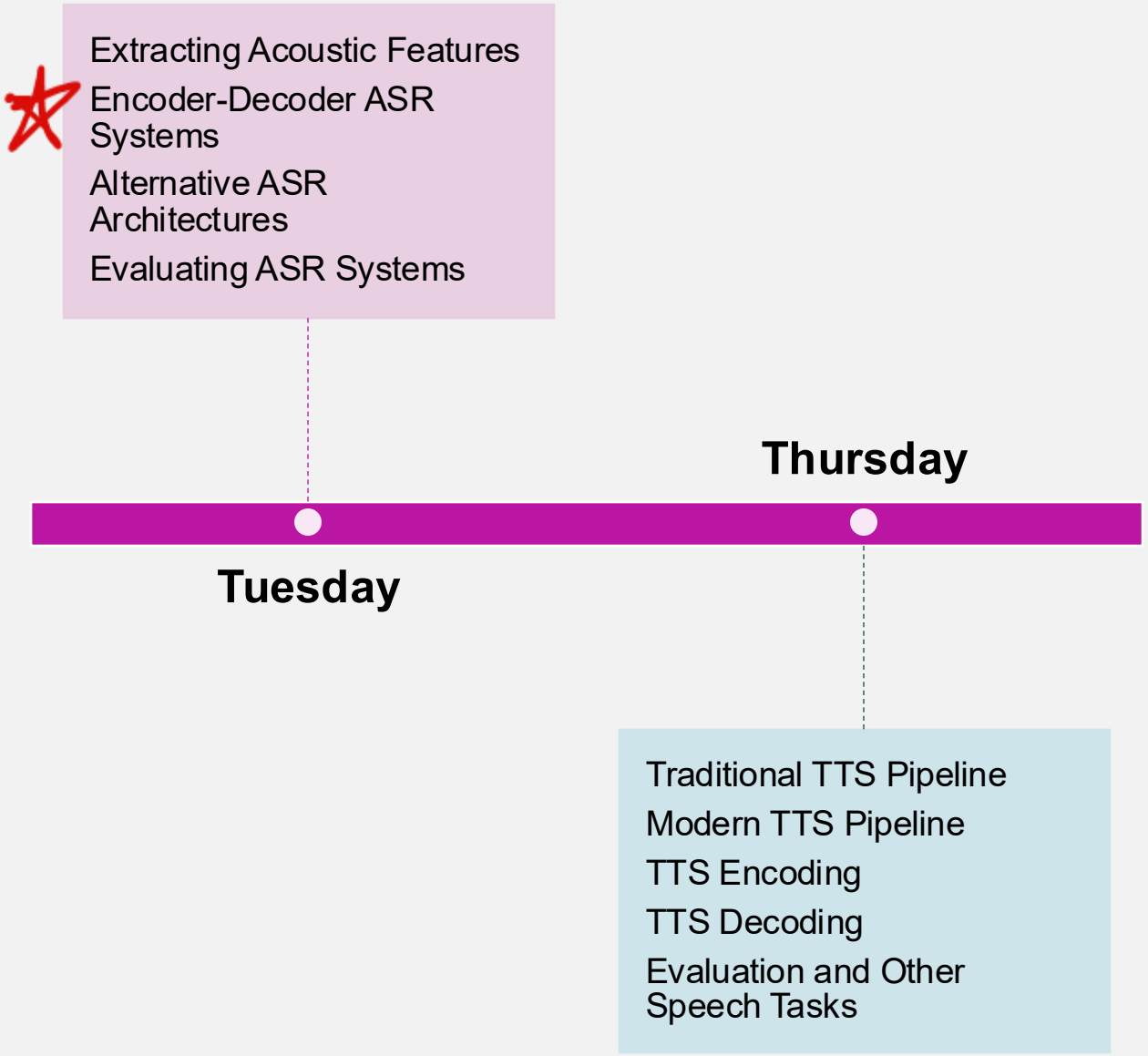
At each temporal frame, calculate the dot product of the kernel with the local input values

This convolution process, as adapted for speech, is sometimes referred to as **cross-correlation**

Other Speech-Specific CNN Details

- Kernel size (sometimes called the **receptive field**) should be a small number of frames compared to the signal
  - Whisper has one frame every 10 ms, with each frame representing a window of 25 ms of speech information
- Try to make kernels just long enough to extract meaningful phonetic features (e.g., stop closures or aspiration)
- Usually, the input to a convolutional layer is the output from a log mel spectrum, so the kernel will have separate vectors for each of the channels from that spectrum and then the overall output for the kernel will integrate information from all the input channels
  - Number of channels = **kernel depth**

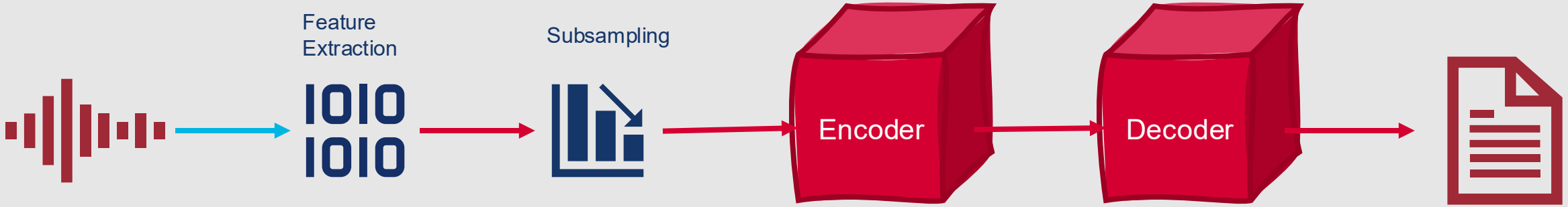
This Week's Topics



How do we build speech recognizers?

- Goal: Map from log mel spectral features to letters or morphemes
- Basic architecture: **Attention-based encoder-decoder (AED)** model
  - Well-suited for tasks in which input and output sequences are expected to be of different lengths (e.g., long sequences of acoustic features being mapped to relatively small sequences of letters)
  - Implemented using RNNs or Transformers
  - Very similar to machine translation!

Encoder-Decoder Speech Recognizer



Subsampling

- Difference between input and output sequence lengths is *very* large for speech recognition!
- Subsampling compresses the input sequence before it is passed into the encoder
- Simplest method: **Low frame rate**[1]
  - At time i, concatenate the acoustic feature vector f_i with the previous two acoustic feature vectors, f_{i-1} and f_{i-2}
  - Delete f_{i-1} and f_{i-2}
  - Repeat at f_{i+3} and so forth to convert the original sequence of t vectors, each with a dimensionality of d, to a new sequence of $\frac{t}{3}$ vectors, each with a dimensionality of $3d$

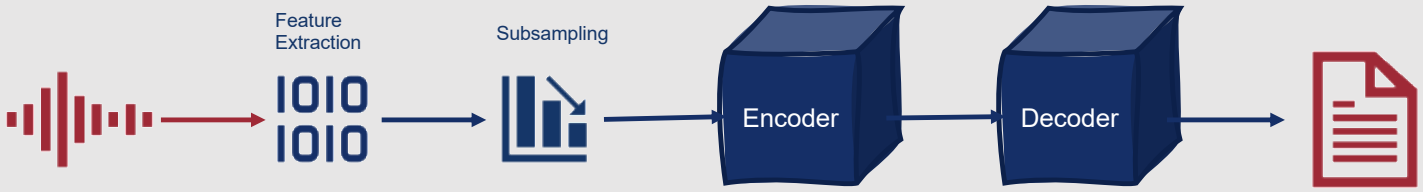
[1]Pundak, Golan, and Tara N. Sainath. "Lower Frame Rate Neural Network Acoustic Models." *Interspeech 2016* (2016): 22-26.

Subsampling Using CNNs

- We can also subsample using CNN layers
  - Set the stride size in the convolutional layer to greater than 1
  - Then, the output sequence will become shorter than the input sequence
- Whisper uses two convolutional layers, with the second having a stride size of 2 (meaning that the output sequence is half the length of the input sequence)

How can we improve speech recognition performance?

- Encoder-decoder models for speech recognition implicitly learn character-level language models, but not necessarily good ones
- It's much easier to find text-only training data than paired text-speech training data!
- Solution: We can use large text-based language models to improve speech recognition performance



Adding Language Models to Speech Recognizers

- Run beam search to get an **n-best list** of recognized sentences
- Use a large, text-based language model to rescore each hypothesis, Y, for the input speech, X
- Interpolate the original speech recognizer score, $P(Y|X)$, with the text-based language model score, $P_{LM}(Y)$, using a finetuned weight parameter, λ, and normalize by the number of characters in the hypothesis, $|Y|_c$
  - $$\text{score}(Y|X) = \frac{1}{|Y|_c} \log P(Y|X) + \lambda \log P_{LM}(Y)$$

Training Speech Recognizers

- Normal cross-entropy loss
- **Teacher forcing** is generally employed to improve training efficiency
  - Pass the gold standard previous token into the decoder at a given time step rather than the predicted token

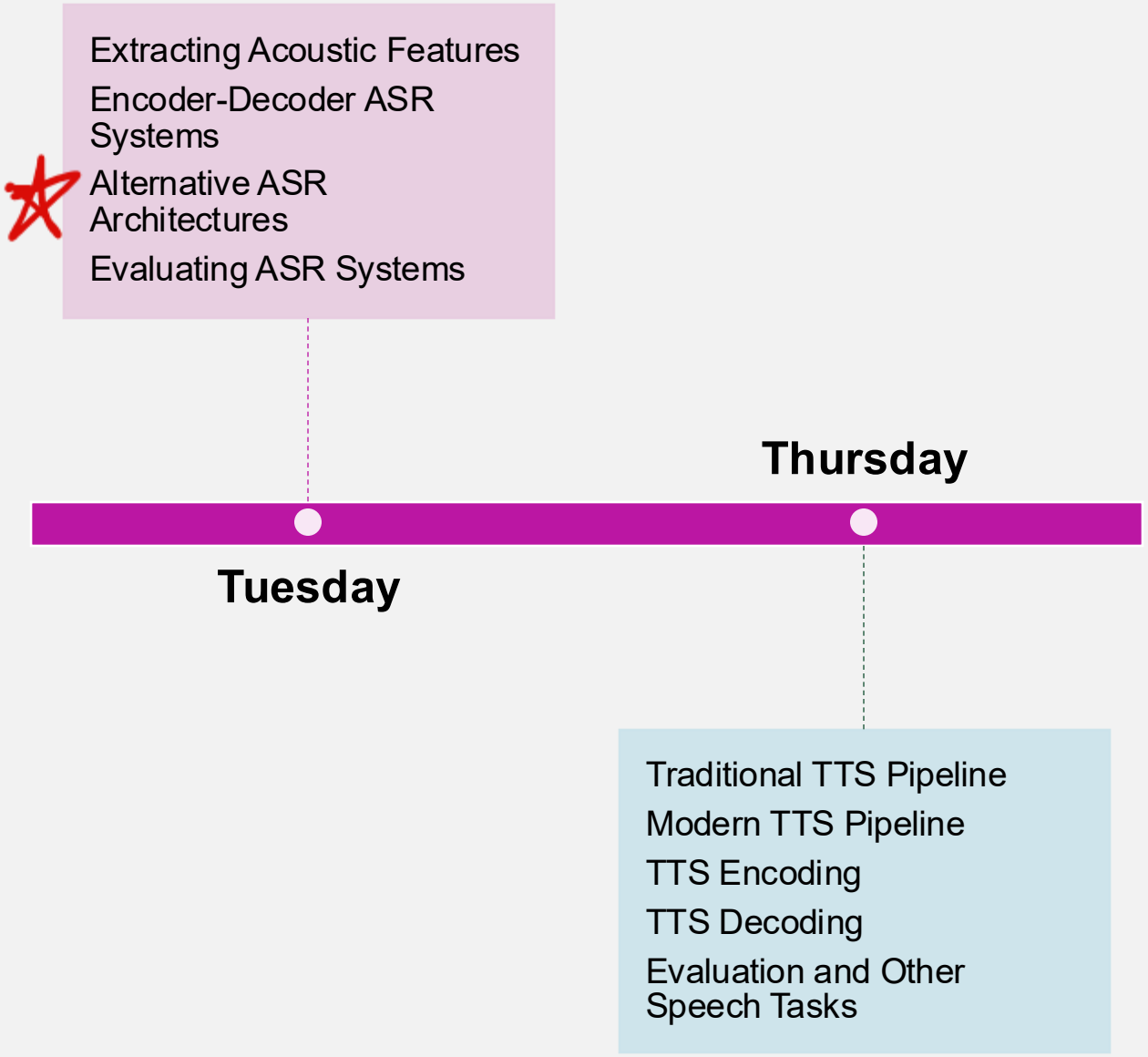
How much data is needed to train encoder-decoder ASR systems?

Whisper-v2 is currently trained on 680k hours of speech (mostly from English, but 118k hours are distributed across 96 other languages)

Another model, OWSM, is trained on 180k hours of primarily manually transcribed publicly available data, including LibriSpeech, Multilingual LibriSpeech, Common Voice, FLEURS, Switchboard, AMI, and others

In general, higher-quality data can enable reliance on lower overall quantities of data

This Week's Topics

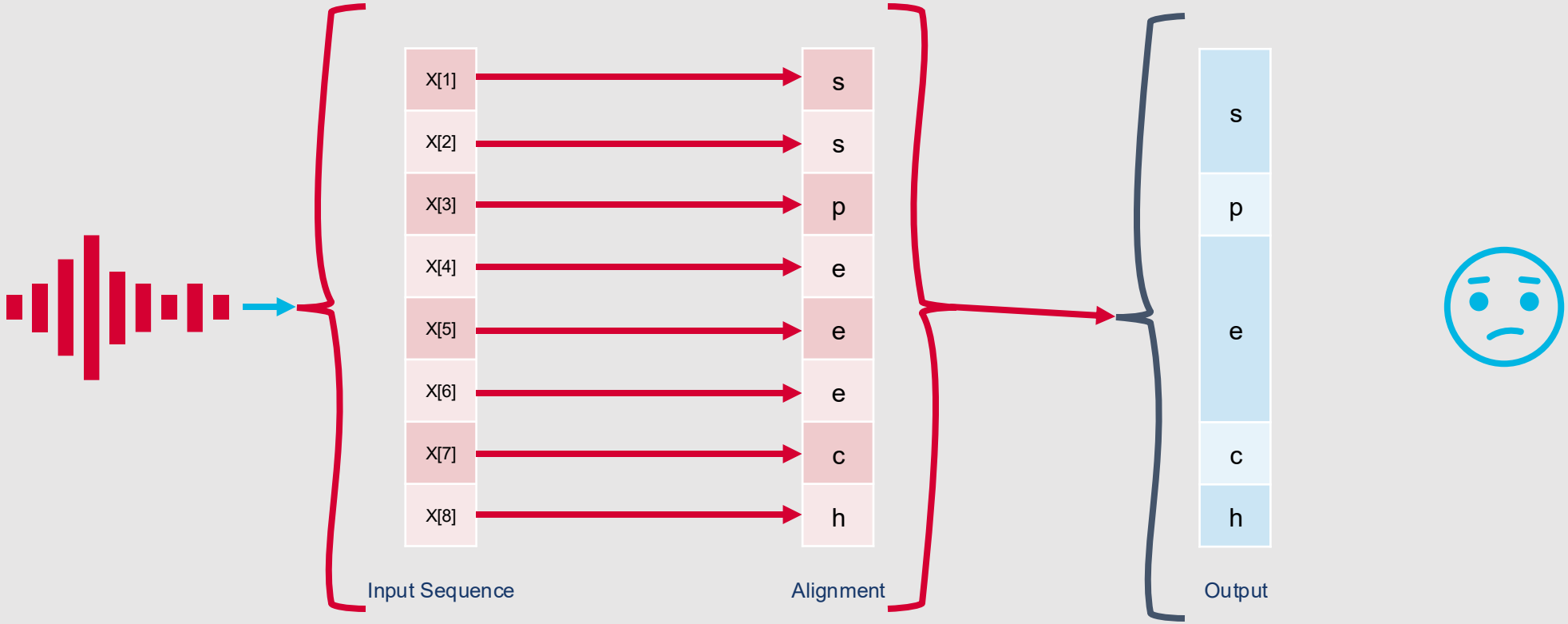


A Classic Alternative: Connectionist Temporal Classification

Alignment-based alternative to encoder-decoder models

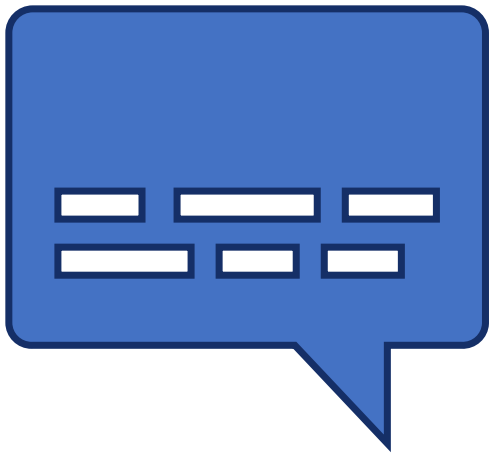
Works by predicting a single character for each element of the input sequence, and then merging redundant characters

Naïve Alignment with Collapsing Function



Limitations of Naïve Alignment

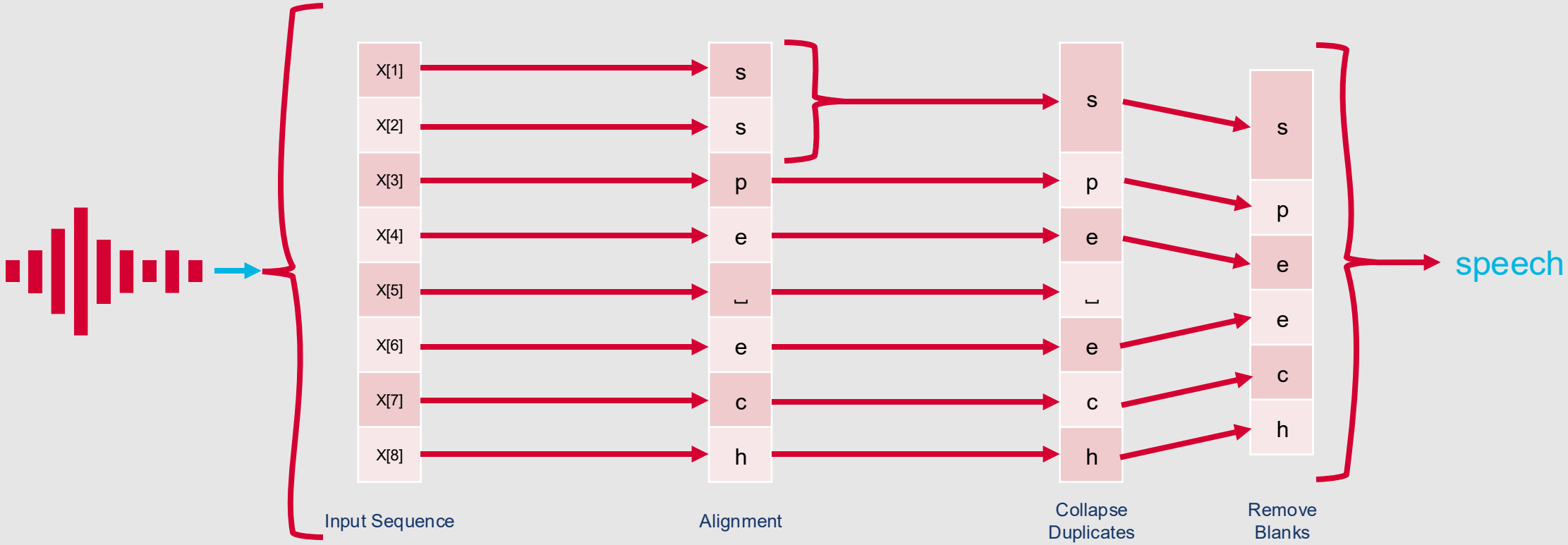
- Can’t handle repeated letters
- Can’t handle silence



- Adds a blank (“␣”) character to the transcription alphabet
  - Can be inserted between repeated characters to prevent them from being collapsed together
  - Can be inserted when no letter needs to be transcribed (e.g., silence)
- Collapsing function then:
  - Collapses repeated letters
  - *Then* removes blanks

How does CTC address these issues?

Connectionist Temporal Classification Algorithm



Alignment Selection

- Many input alignments can exist!
- How can CTC select from among many possible alignments?
  - Assume conditional independence, letting $A = \{a_1, \dots, a_T\}$ represent an alignment for X
    - $P_{\text{CTC}}(A|X) = \prod_{t=1}^{T} P(a_t|X)$
  - Greedily select the best alignment, $\hat{A} = \{\widehat{a_1}, \dots, \widehat{a_T}\}$, by choosing the character, c, that maximizes the probability at each time t
    - $\widehat{a_t} = \underset{c \in C}{\text{argmax}}\, P_t(c|X)$

Identifying the Most Probable Alignment

- The most probable output sequence $\hat{Y}$ is the output sequence Y that has the highest sum over the probability of all of its possible alignments, $A \in B^{-1}(Y)$
  - $P_{\text{CTC}}(Y|X) = \sum_{A \in B^{-1}(Y)} P(A|X)$
  - $\hat{Y} = \underset{Y}{\text{argmax}}\, P_{\text{CTC}}(Y|X)$
- We can approximate this using a Viterbi extension[1] of beam search to avoid summing over all alignments
- Just like with encoder-decoder models, we can interpolate a text-based language model score, $P_{\text{LM}}(Y)$, with the CTC probability and a length factor, $L(Y)$
  - $\text{score}_{\text{CTC}}(Y|X) = \log P_{\text{CTC}}(Y|X) + \lambda_1 \log P_{\text{LM}}(Y) \lambda_2 L(Y)$

[1]Hannun, A. Y., Maas, A. L., Jurafsky, D., & Ng, A. Y. (2014). First-pass large vocabulary continuous speech recognition using bi-directional recurrent DNNs. *arXiv preprint arXiv:1408.2873*.

Training CTC-based ASR Systems

- CTC loss function for a dataset D:
  - $L_{\text{CTC}} = \sum_{(X,Y)\in D} -\log P_{\text{CTC}}(Y|X)$
- We can efficiently merge the alignments to help compute this sum using a variation of the forward-backward algorithm[1]

[1]Hannun, A. (2017). Sequence modeling with ctc. *Distill*, *2*(11), e8.

If we want, we can even combine CTC with an encoder-decoder architecture.

- Weight the two losses using a finetuned weight parameter, λ
  - $L = -\lambda \log P_{\text{encoder-decoder}}(Y|X) - (1-\lambda) \log P_{\text{CTC}}(Y|X)$
- Also weight the contributions of the encoder-decoder, CTC, and text-based language model using finetuned weights when making predictions
  - $\hat{Y} = \underset{Y}{\text{argmax}}[\lambda \log P_{\text{encoder-decoder}}(Y|X) - (1-\lambda) \log P_{\text{CTC}}(Y|X) + \gamma \log P_{\text{LM}}(Y)]$

How do CTC speech recognizers compare with encoder-decoder speech recognizers?

- **CTC speech recognizers**
  - Generally lower accuracy
  - Can be used for streaming audio signals
- **Encoder-decoder speech recognizers**
  - Generally higher accuracy
  - Cannot be used for streaming since they need to compute attention over the entire input before passing information to the decoder

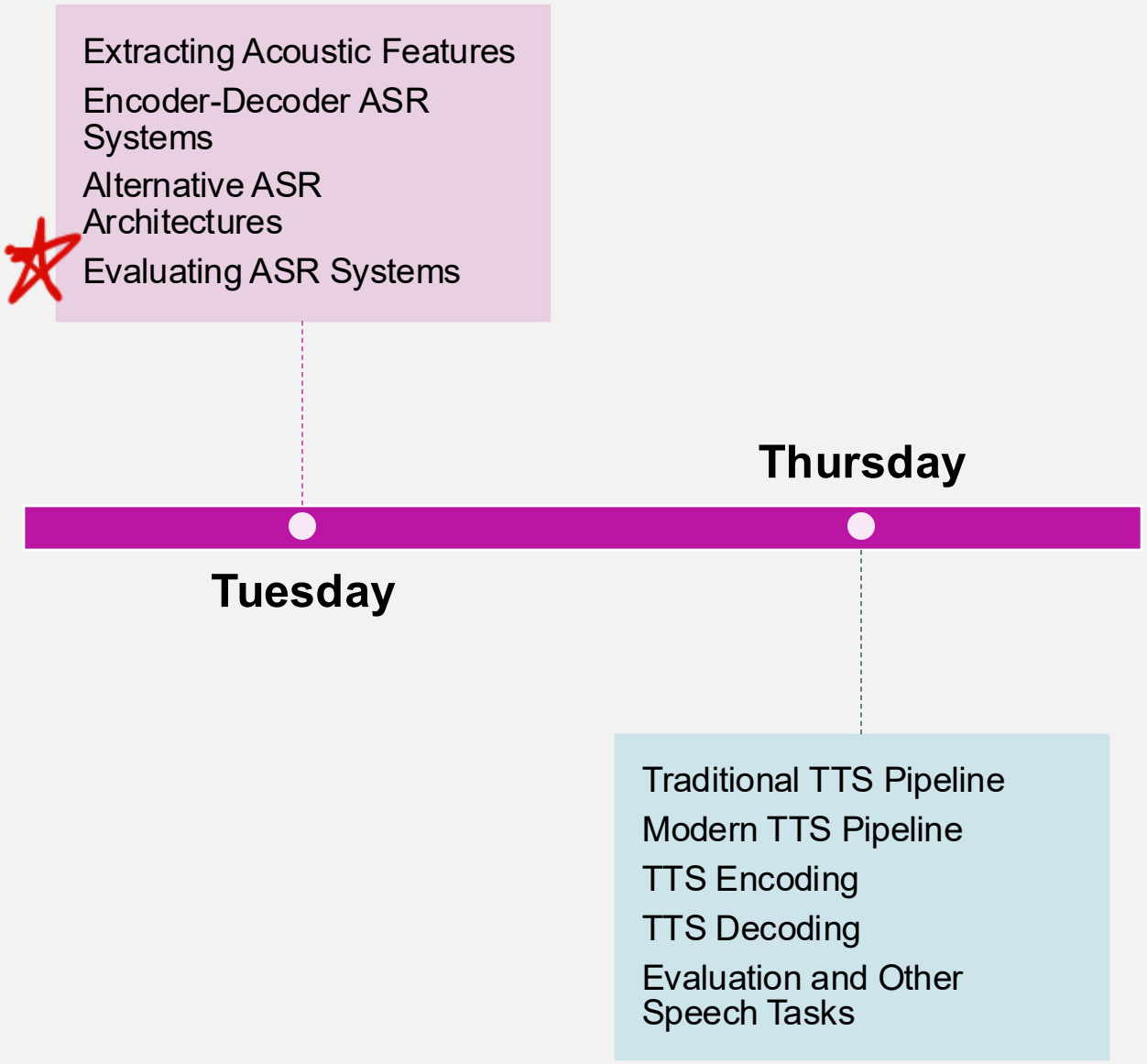
What if we don't have large quantities of labeled training data, but still want to use more modern architectures?

- Self-supervised speech models!
- Popular example: HuBERT
  - https://huggingface.co/docs/transformers/en/model_doc/hubert
  - https://dl.acm.org/doi/10.1109/TASLP.2021.3122291
- Rather than mapping an acoustic input to a string of letters or tokens, these models:
  - First, bootstrap discrete phonetic units from the acoustic input, learning to map from waveforms to these units
    - Requires unlabeled speech files, but no transcripts
  - Next, finetune these pretrained models using a smaller set of labeled data

How can we bootstrap phonetic units?

- Start with mel frequency cepstral coefficients and extract these from the entire acoustic training set
- Cluster the MFCCs using k-means clustering
  - This produces a **codebook** of *k* acoustic units
- These acoustic units are then used as targets for the remainder of the pretraining process

This Week's Topics



Word Error Rate

Measures the difference between the predicted and gold standard transcriptions

Based on minimum edit distance, setting penalties for insertions, deletions, and substitutions to 1

Word Error Rate

- Normalize the total number of word **substitutions**, **insertions**, and **deletions** that are necessary to map the predicted transcript to the gold standard transcript by the number of words in the gold standard transcript
  - $$\text{Word Error Rate} = 100 * \frac{\text{Insertions+Substitutions+Deletions}}{\text{\# Words in Gold Standard Transcript}}$$

Word Error Rate: Case Example

I really love to see its fort on a run

I really love CS 421

run						
a						
on						
fort						
its						
see						
to						
love						
really						
I						
#						
	#	**I**	**really**	**love**	**CS**	**421**

Word Error Rate: Case Example

I really love to see its fort on a run

I really love CS 421

run	10	9	8	7	7	7
a	9	8	7	6	6	6
on	8	7	6	5	5	5
fort	7	6	5	4	4	4
its	6	5	4	3	3	3
see	5	4	3	2	2	2
to	4	3	2	1	1	2
love	3	2	1	0	1	2
really	2	1	0	1	2	3
I	1	0	1	2	3	4
#	0	1	2	3	4	5
	#	**I**	**really**	**love**	**CS**	**421**

Word Error Rate: Case Example

I really love to see its fort on a run

I really love CS 421

run	10	9	8	7	7	7
a	9	8	7	6	6	6
on	8	7	6	5	5	5
fort	7	6	5	4	4	4
its	6	5	4	3	3	3
see	5	4	3	2	2	2
to	4	3	2	1	1	2
love	3	2	1	0	1	2
really	2	1	0	1	2	3
I	1	0	1	2	3	4
#	0	1	2	3	4	5
	#	**I**	**really**	**love**	**CS**	**421**

Word Error Rate: Case Example

I really love to see its fort on a run

I really love CS 421

$$\text{WER} = 100 * \frac{\text{I} + \text{S} + \text{D}}{\text{\# Words in Gold Standard}}$$

run	10	9	8	7	7	7
a	9	8	7	6	6	6
on	8	7	6	5	5	5
fort	7	6	5	4	4	4
its	6	5	4	3	3	3
see	5	4	3	2	2	2
to	4	3	2	1	1	2
love	3	2	1	0	1	2
really	2	1	0	1	2	3
I	1	0	1	2	3	4
#	0	1	2	3	4	5
	#	**I**	**really**	**love**	**CS**	**421**

D=5

S=2

#=5

Word Error Rate: Case Example

I really love to see its fort on a run

I really love CS 421

$$\text{WER} = 100 * \frac{\text{I} + \text{S} + \text{D}}{\text{\# Words in Gold Standard}}$$

$$\text{WER} = 100 * \frac{7}{5} = 140\%$$

run	10	9	8	7	7	7
a	9	8	7	6	6	6
on	8	7	6	5	5	5
fort	7	6	5	4	4	4
its	6	5	4	3	3	3
see	5	4	3	2	2	2
to	4	3	2	1	1	2
love	3	2	1	0	1	2
really	2	1	0	1	2	3
I	1	0	1	2	3	4
#	0	1	2	3	4	5
	#	**I**	**really**	**love**	**CS**	**421**

D=5

S=2

#=5

sclite

- Standard package for computing word error rates
- Freely available from the National Institute of Standards and Technologies (NIST): https://github.com/usnistgov/SCTK
- Also performs other useful speech evaluation tasks:
  - Shows which words are often misrecognized for others using confusion matrices
  - Summarizes statistics of words that are often inserted or deleted
  - Can provide error rates by speaker and the percentage of sentences with at least one word error

Preparing Text for Evaluation

- Common to standardize text before computing word error rate by:
  - Removing metalanguage (e.g., notes or transcription comments appearing between brackets)
  - Removing or standardizing interjections or filled pauses
  - Standardizing contracted and non-contracted forms of language (e.g., “I’m” versus “I am”)
  - Normalizing numbers, quantities, dates, and times
  - Unifying regional spelling differences (e.g., US versus UK English spellings)

Matched-Pair Sentence Segment Word Error (MAPSSWE)

- Test for determining whether the difference in word error rate between two ASR systems is statistically significant
- Averages across word error rates for different segments
- Letting N_A^i be the number of errors made on segment i by system A, N_B^i be the number of errors made on segment i by system B, and $Z_i = N_A^i - N_B^i$, we would expect the average of all Z values when comparing identical systems to be 0
- We can estimate the true average from our sample as:
  - $\widehat{\mu_z} = \frac{\sum_{i=1}^{n} Z_i}{n}$
- We can estimate the variance as:
  - $\sigma_z^2 = \frac{1}{n-1}\sum_{i=1}^{n}(Z_i - \mu_z)^2$
- Finally, we can compute:
  - $W = \frac{\widehat{\mu_z}}{\sigma_Z / \sqrt{n}}$
- Assuming a large enough n, we can reject the null hypothesis ($\mu_z = 0$) if $P(Z \geq |w|) \leq 0.05$, where Z is the standard normal distribution and w is the realized value W

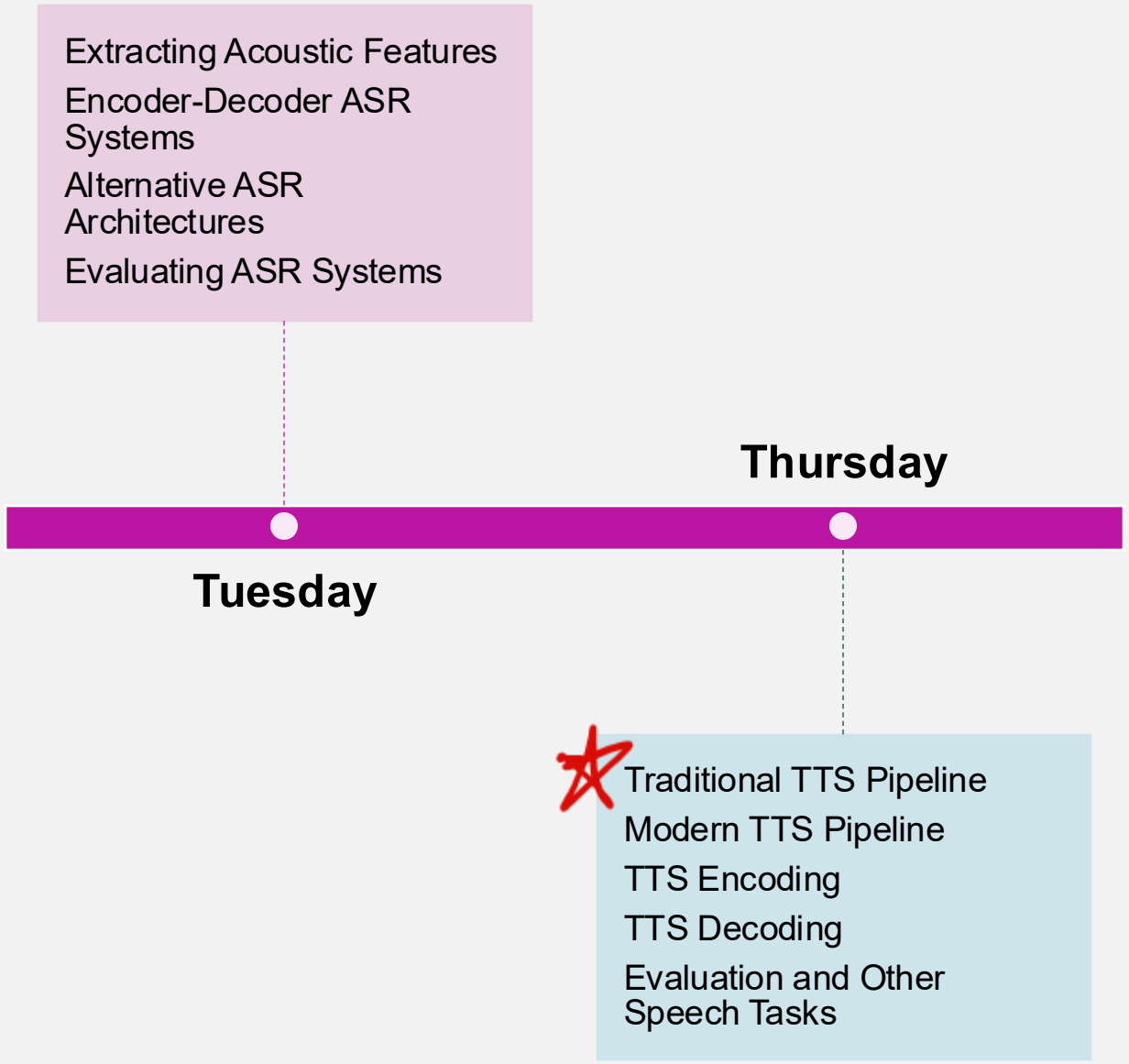
Remaining Challenges in ASR

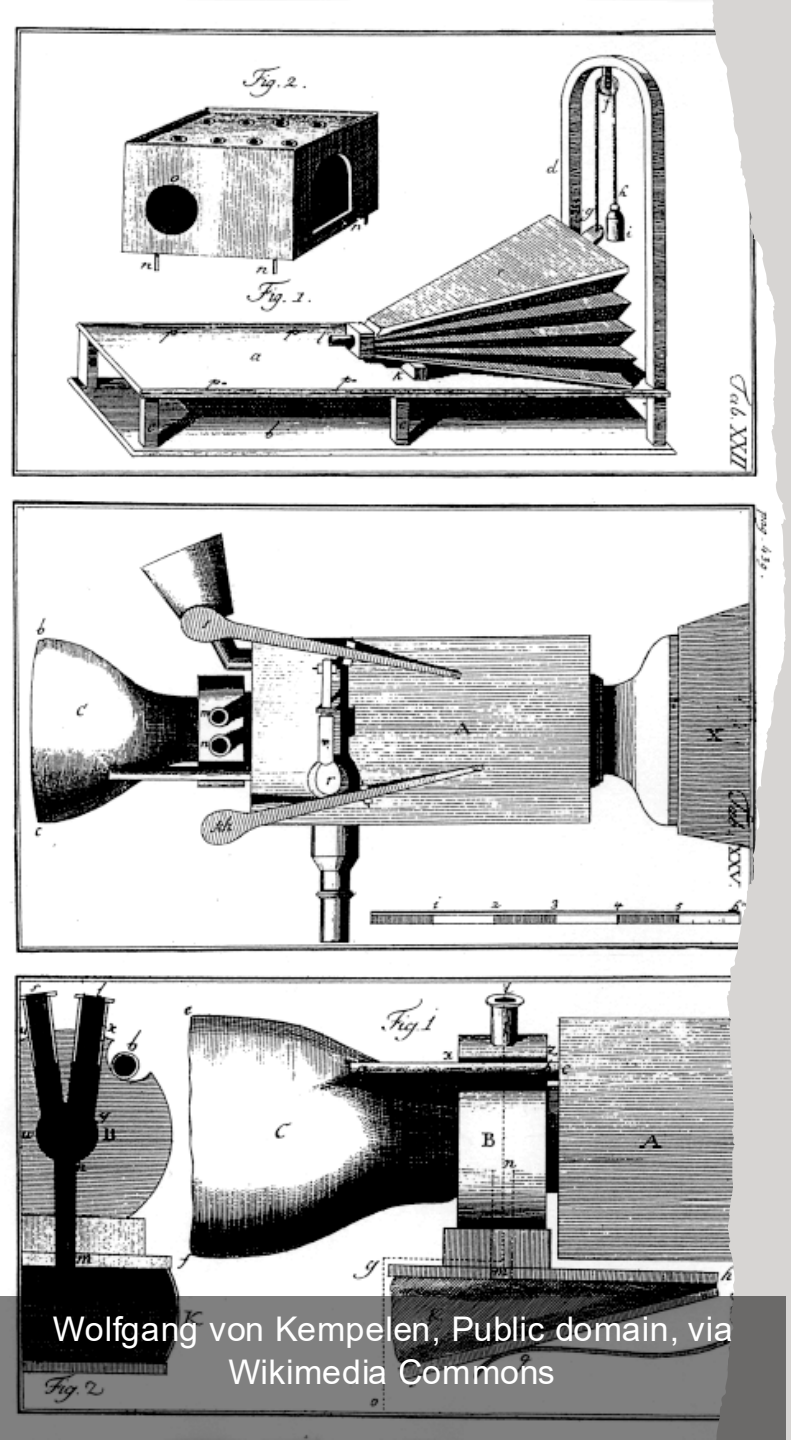
- Better metrics needed
  - Would be nice to have metrics that didn't give equal weight to every word, although it is difficult to agree on how to weight different word types for different ASR applications
- Lingering performance issues when handling non-standard language dialects and rapid switching between languages
- Sub-optimal performance in noisy, real-world environments
- Model degradation when moving from training domains (e.g., read speech) to new settings (e.g., clinical transcription)
- Low or no consideration of prosody or other paralinguistic information

Summary: ASR

- **Automatic speech recognition** is the task of mapping an input acoustic signal to a string of text
- This is often achieved by converting the acoustic signal to an acoustic feature vector through a process of **sampling** and **quantization**, and then passing that feature vector into an encoder-decoder model
- An alternative to using encoder-decoder models is **connectionist temporal classification**, which outputs a character for each input frame and then collapses these together to produce an alignment between input signal and output text
- Self-supervised ASR approaches like HuBERT bootstrap **phonetic units** from the input acoustic signal and then can be fine-tuned with much smaller amounts of labeled data for specialized tasks
- ASR performance is often evaluated using **word error rate**

This Week's Topics





Wolfgang von Kempelen, Public domain, via Wikimedia Commons

Text-to-Speech

- Goal: Map from sequences of letters to acoustic signals
- Many practical applications in chatbots and assistive devices
- Earliest speech synthesizer was built in the late 1700s!

How can we create TTS systems?

Traditional approach:

- Train a model using data from recording studios to replicate a single speaker

More modern approach:

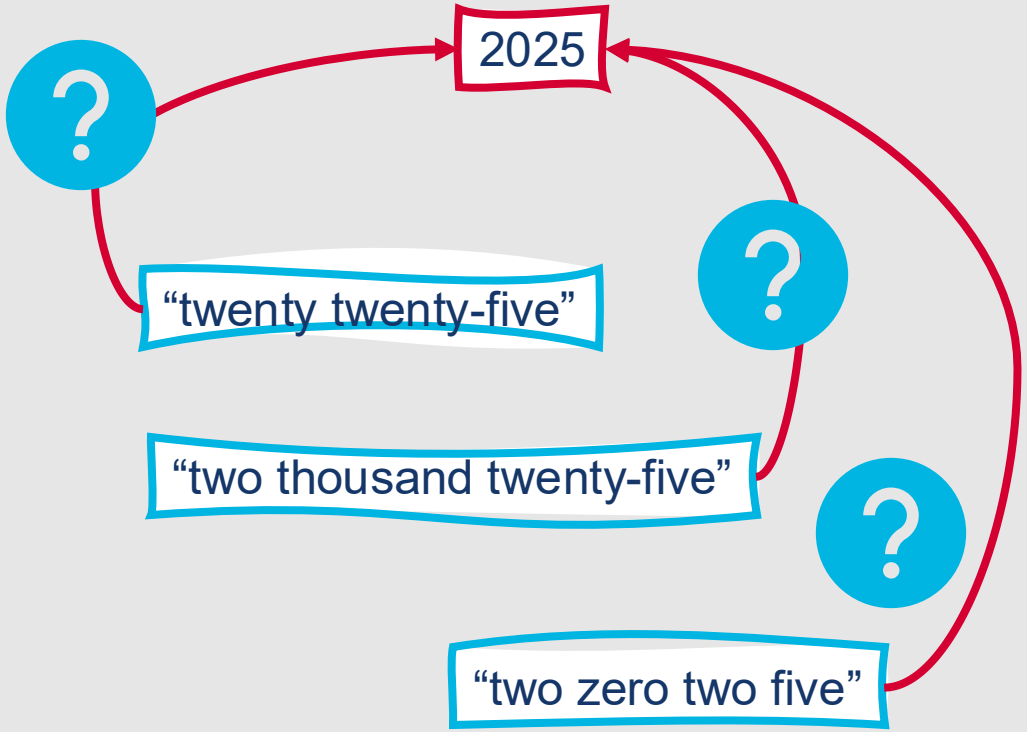
- Train a model on huge datasets containing multiple speakers
- Use voice prompts to guide the creation of a new voice not previously seen during training (sometimes called **zero-shot TTS**)

Breakdown of a Single-Voice TTS System

- **Encoder-decoder** architecture
- Two subtasks:
  - **Spectrogram prediction:** Mapping from sequences of letters to sequences of mel spectral values
  - **Vocoding:** Mapping from sequences of mel spectral values to waveforms

TTS Preprocessing

- Text sequences are preprocessed before predicting spectrograms to handle words with pronunciations that vary depending on context
- Pronunciations often depend on the word's **semiotic class**
  - Abbreviation
  - Date
  - Year
  - Money
  - Percentage
- This process is generally referred to as **text normalization**



Text Normalization

- Can be done using rule-based techniques or an encoder-decoder model
- **Rule-based normalization:**
  - Write regular expressions to detect non-standard words
  - Write rules specifying how different semiotic classes should be verbalized
- **Encoder-decoder normalization:**
  - Train the system to map from an initial realization to a target verbalization

Rule-based vs. encoder-decoder normalization?

Rule-based normalization

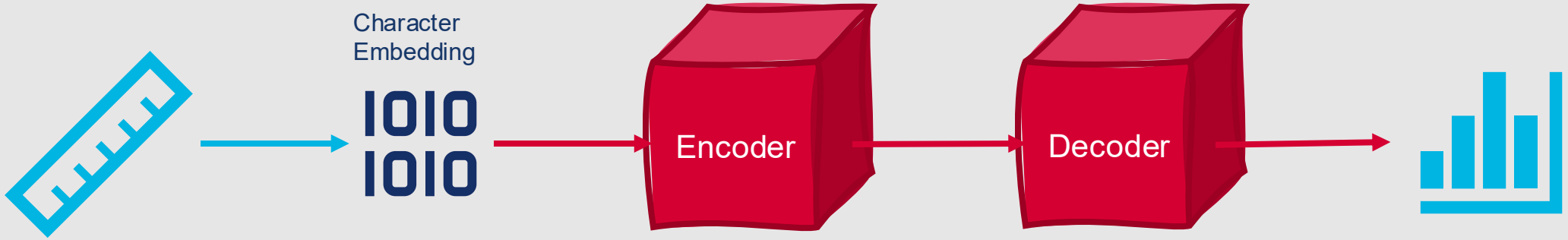
- Doesn’t require any training data
- Requires potentially time-consuming rule creation by experts

Encoder-decoder normalization

- Requires a labeled training set
- Generally achieves higher performance

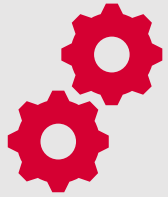
Spectrogram Prediction

- Can be performed using an encoder-decoder with attention
- Very similar to ASR systems!



Vocoding

Goal: Convert a log mel spectrogram to an acoustic waveform

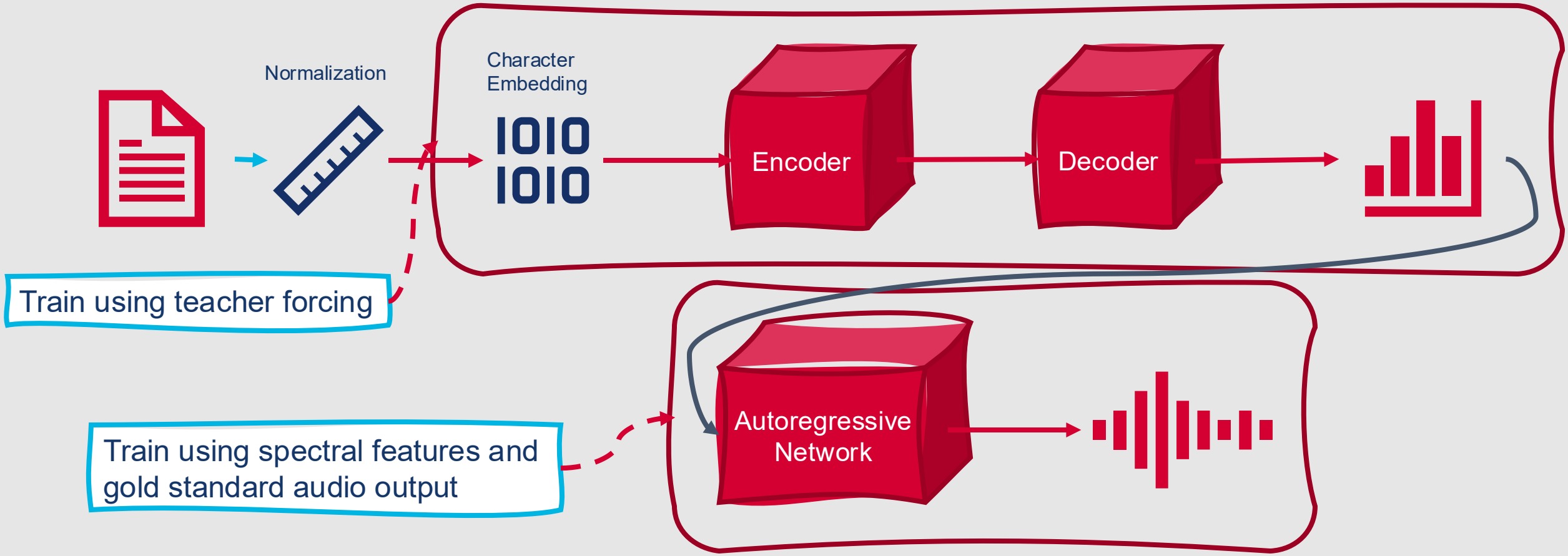


Can be done using architectures like those used for autoregressive language modeling

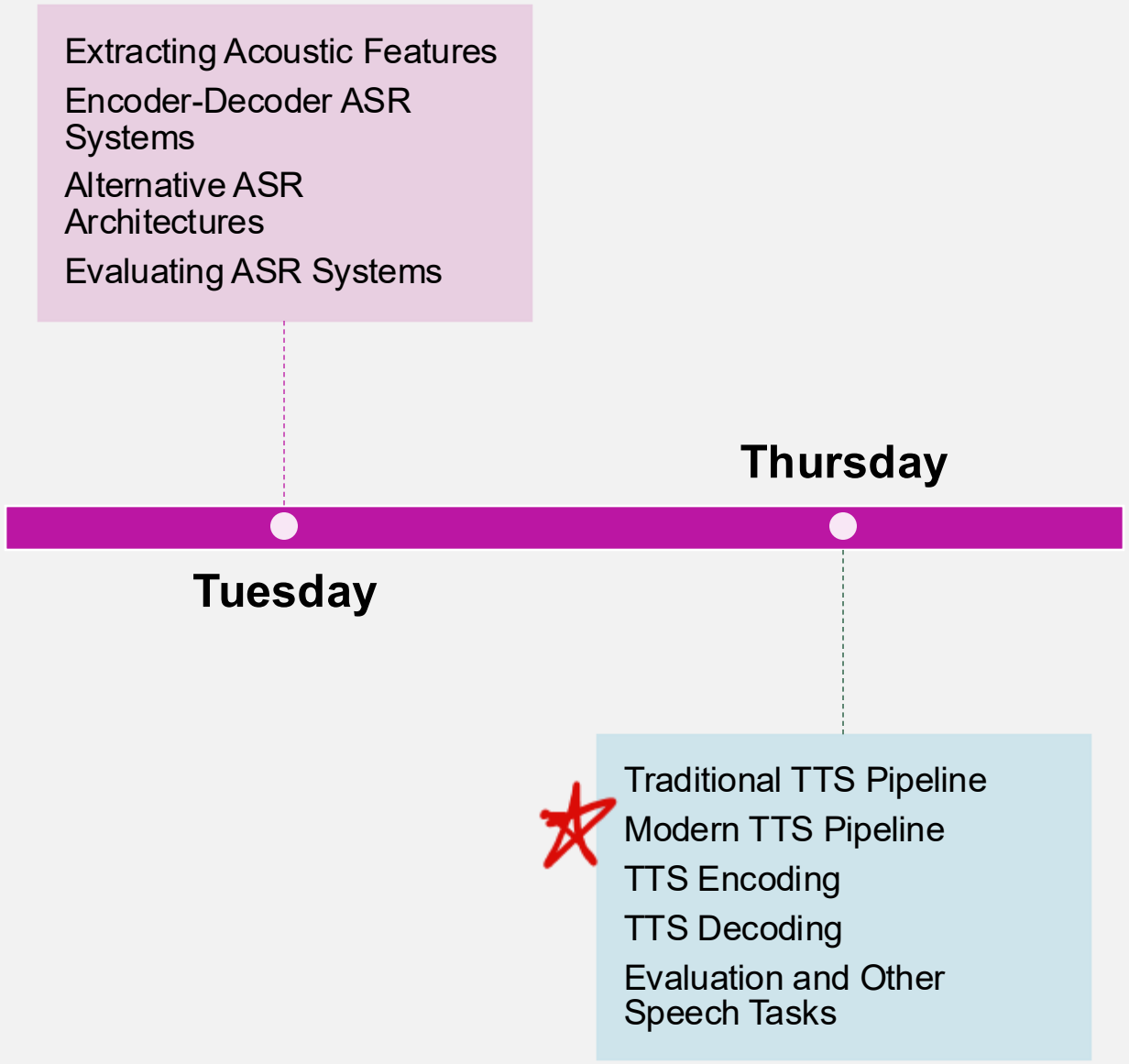
Generally predict mu-law audio samples at each timestep

Compressed 8-bit acoustic samples

Putting it all together….



This Week's Topics



Breakdown of a Modern Multi-Voice TTS System

Large language model architectures

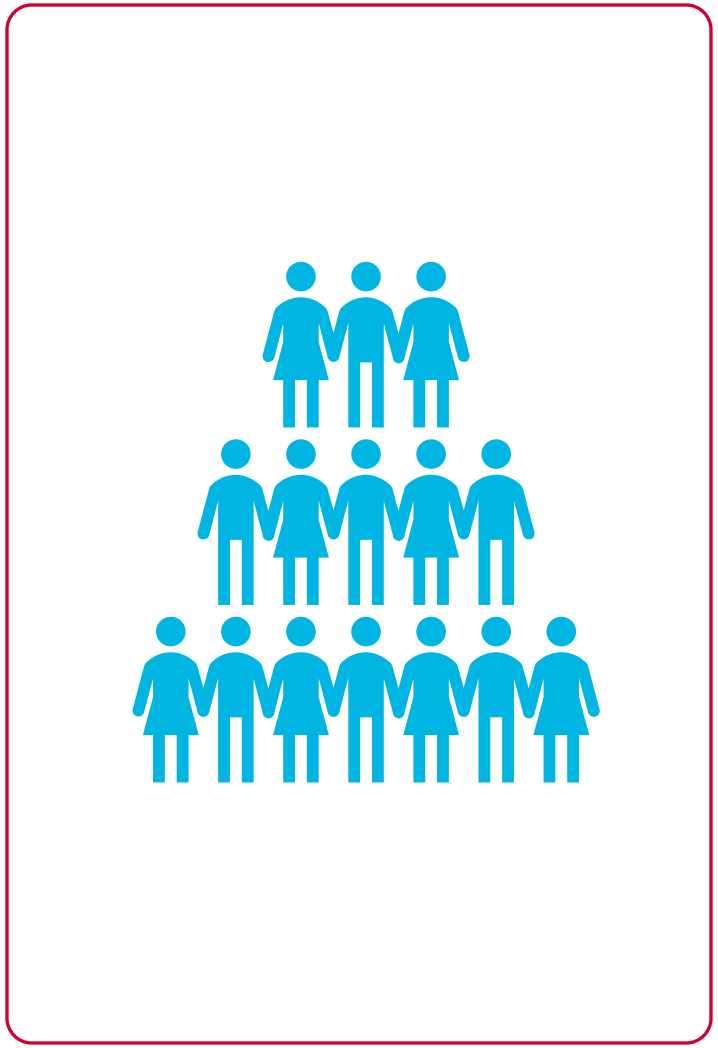
Typically leverage:

Discrete audio tokens

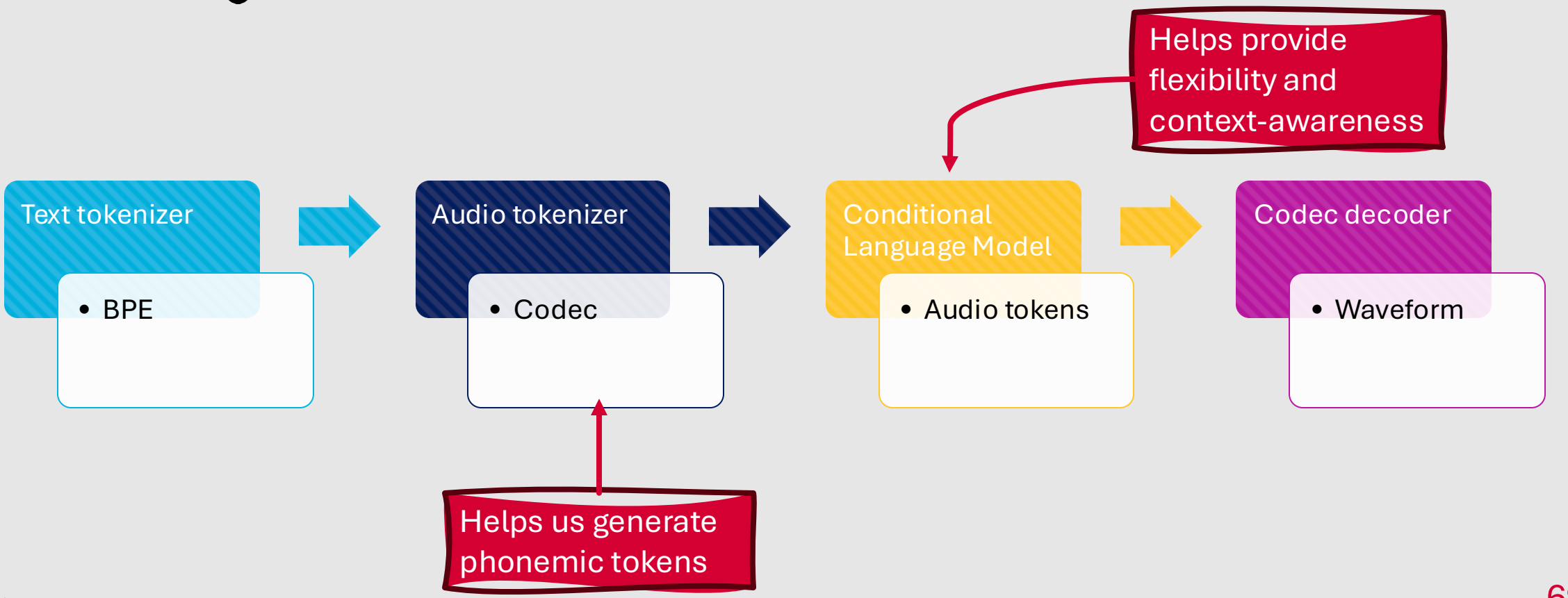
Conditional generation

Why the ongoing shift to this approach?

- Traditional systems required large quantities of data (hundreds of hours) from a single speaker; obtaining such quantities of data is costly
- It's easier to acquire less data from many speakers, and allows for the development of more flexible generated speech



The Modern TTS Pipeline



Zero-Shot TTS

Input:

- Text prompt
- A few seconds of a voice sample

Output:

- Spoken form of the specified text, mimicking the voice sample

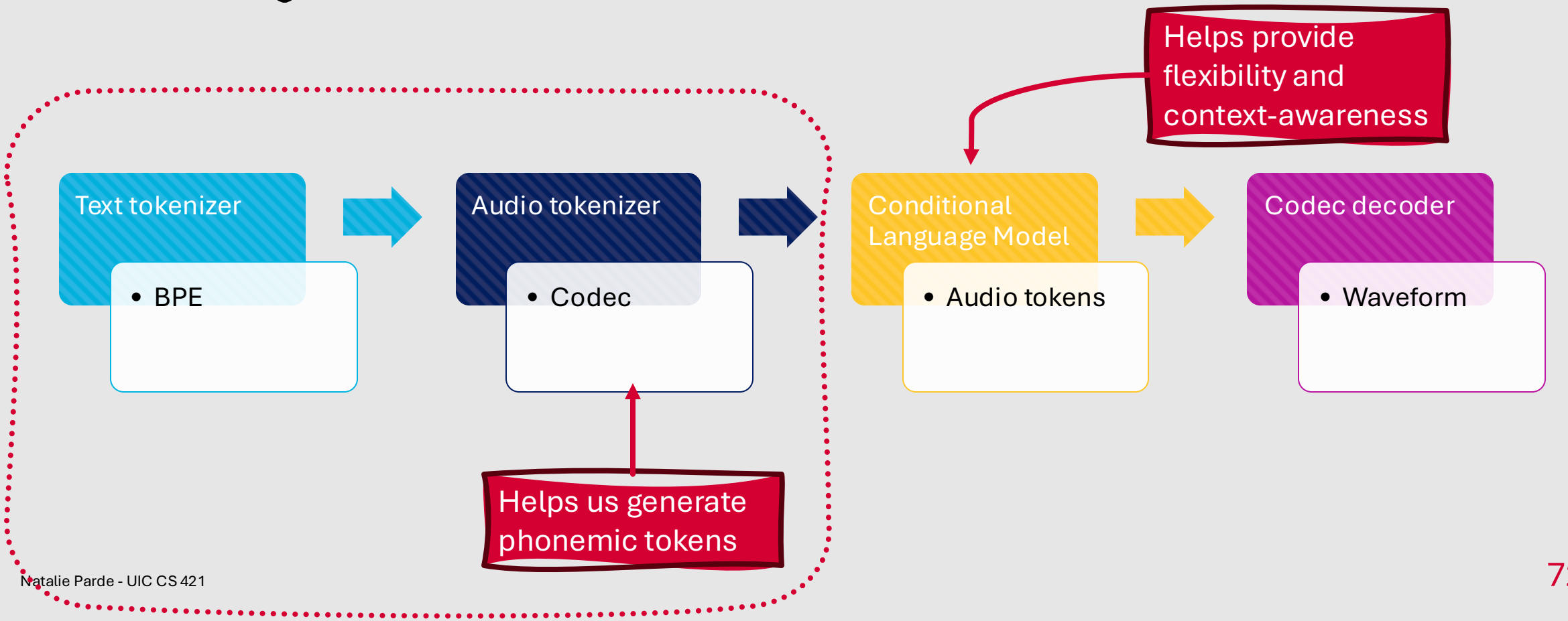
Uses in-context learning to map the input voice sample to new generated output

Will typically imitate vocal qualities, prosody, and even background noise

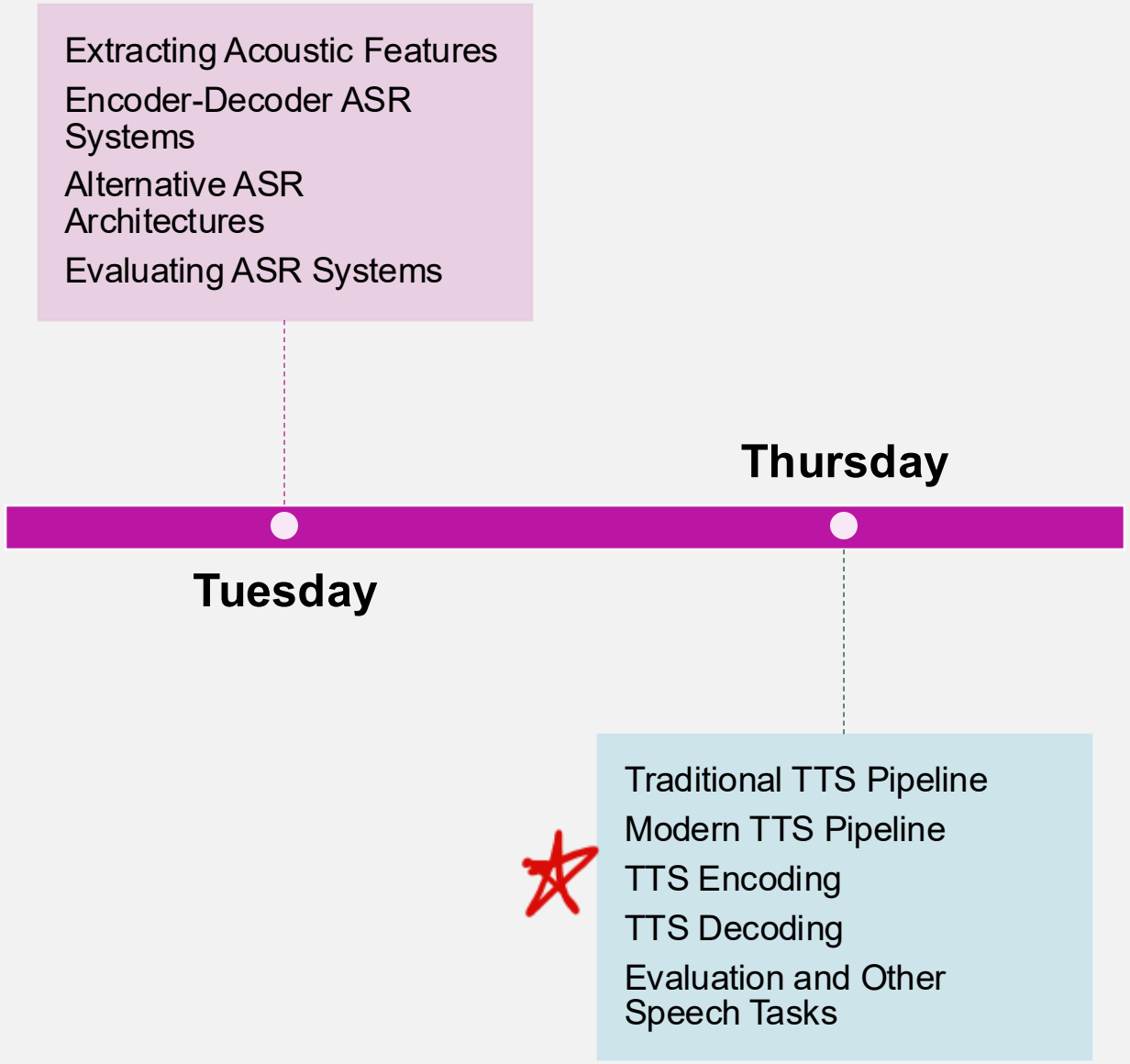
Can language models directly model audio waveforms?

- No (not yet!)
- If speech is sampled at 24 kHz, then we have 24,000 data samples per second
- Current language models struggle to handle that many timesteps
- Audio codecs help us reduce this dramatically, creating discrete, compressed tokens
  - ~75 tokens/second

The Modern TTS Pipeline



This Week's Topics



Byte-Pair Encoding (BPE) Refresher

- Merge frequent character pairs to form subwords
- Learn which character pairs are frequent using large text corpora

Word Tokenizer

- ["I", "can't", "believe", "it's", "already", "November", "!"]

Subword Tokenizer

- ["I", "can", "##’”, "t", "believe", "it", "##'", "s", "already", "November", "!"]

Byte-Pair Encoding

- ["I", " can", ”’”, “t”, " believe", " it", "'", "s", " already", " November", "!"]

Neural Audio Codecs

- Word is formed from **co**der/**dec**oder
- Historically, codecs were hardware devices that digitized analog symbols
- In this case, we're encoding analog speech signals into digitized, compressed representations
  - Helps us create discrete tokens from a speech signal!
- Generally, codecs are trained as **autoencoders**
  - Goal: Reconstruct the waveform from a compressed representation

ENCODEC

- Widely used neural codec that was originally designed for efficient audio compression
  - https://huggingface.co/docs/transformers/main/en/model_doc/encodec
- Downsamples audio by 320x
- Produces embeddings at 75 Hz
- Discrete tokens preserve acoustic quality and speaker identity

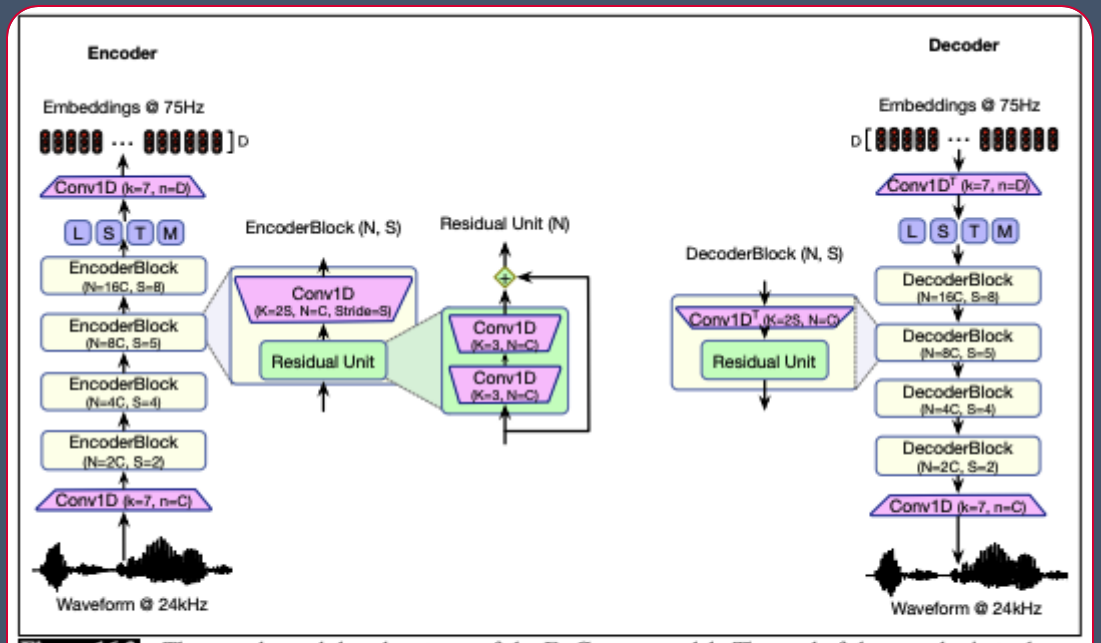


Figure 16.3 The encoder and decoder stages of the ENCODEC model. The goal of the encoder is to downsample an input waveform by encoded it as a series of embeddings $\mathbf{z}_t$ at 75Hz, i.e. 75 embeddings a second. Because the original signal was represented at 24kHz, this is a downsampling of $\frac{24000}{75} = 320$ times. Between the encoder and decoder is a quantization step producing a lossy embedding $\mathbf{z}_{\mathbf{q}_t}$. The goal of the decoder is to take the lossy embedding $\mathbf{z}_{qt}$ and upsample it, converting it back to a waveform.

ENCODEC Encoder

- CNN layers and residual blocks
- Large stride size to support downsampling
- Each vector represents ~13 ms of sound
- Embeddings capture pitch, timbre, and phonetic content

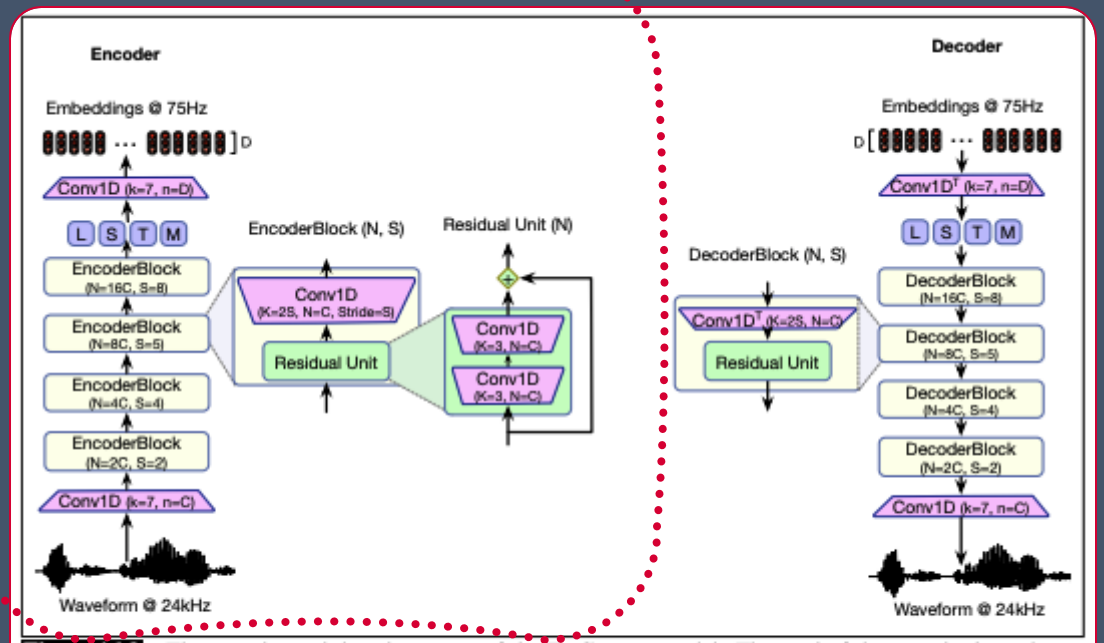


Figure 16.3 The encoder and decoder stages of the ENCODEC model. The goal of the encoder is to downsample an input waveform by encoded it as a series of embeddings $\mathbf{z}_t$ at 75Hz, i.e. 75 embeddings a second. Because the original signal was represented at 24kHz, this is a downsampling of $\frac{24000}{75} = 320$ times. Between the encoder and decoder is a quantization step producing a lossy embedding $\mathbf{z}_{\mathbf{q}_t}$. The goal of the decoder is to take the lossy embedding $\mathbf{z}_{qt}$ and upsample it, converting it back to a waveform.

ENCODEC Decoder

- Upsamples CNN layers
- Reconstructs waveform from embeddings
- Works with quantizer output

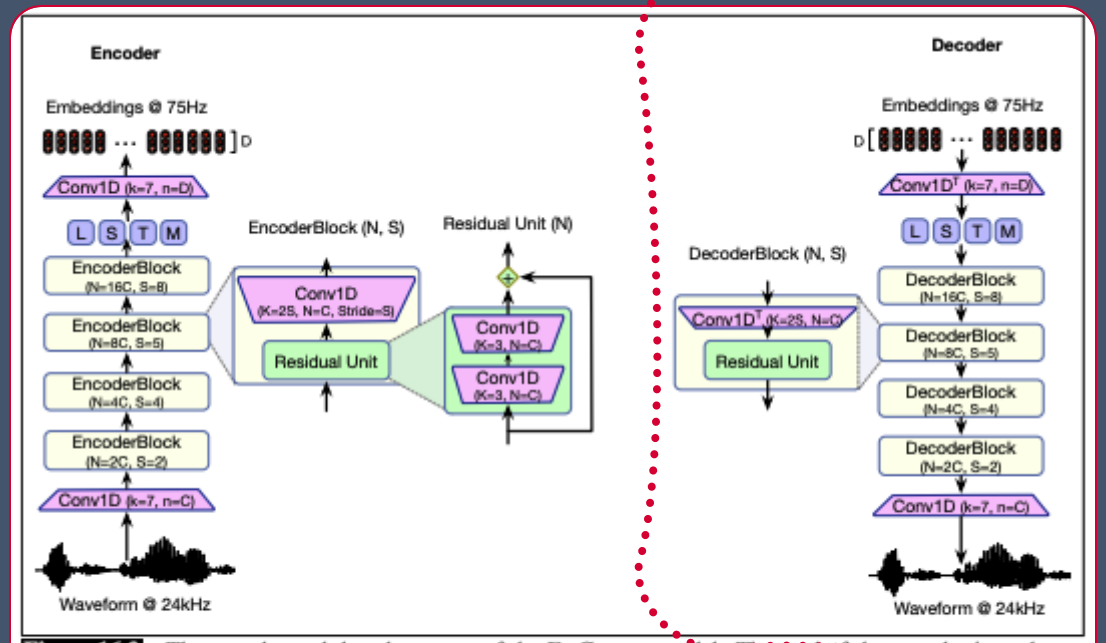


Figure 16.3 The encoder and decoder stages of the ENCODEC model. The goal of the encoder is to down sample an input waveform by encoded it as a series of embeddings $\mathbf{z}_t$ at 75Hz, i.e. 75 embeddings a second. Because the original signal was represented at 24kHz, this is a downsampling of $\frac{24000}{75} = 320$ times. Between the encoder and decoder is a quantization step producing a lossy embedding $\mathbf{z}_{\mathbf{q}_t}$. The goal of the decoder is to take the lossy embedding $\mathbf{z}_{qt}$ and upsample it, converting it back to a waveform.

What is the quantizer?

- The intermediate component of the encoder-decoder architecture that:
  - Takes each embedding corresponding to part of the waveform
  - Represents it by a sequence of discrete tokens each taken from one of the codebooks
- Helps enforce consistency across speakers and acoustic conditions

Vector Quantization (VQ)

- Historically, used to compress a speech signal to reduce the bit rate for transmission or storage
- Basic idea:
  - Turn each vector into an integer index representing a class or cluster
  - Then, instead of transmitting a big vector of floating point numbers, transmit the integer index
  - At the other end of the transmission, reconstitute the vector from the index
- In modern speech applications, we use vector quantization not necessarily for transmission purposes, but because it conveniently creates discrete tokens!
  - Fits well into the language modeling paradigm

How does vector quantization work?

Typically implemented using k-means clustering

In essence, we:

- Run a large set of speech wavefiles through an encoder to generate *n* vectors, each corresponding to some frame of speech
- Cluster all of these *n* vectors into *k* clusters
- Use the cluster index as a discrete signal
  - Associate each cluster with a codeword (centroid of all the vectors in the cluster) that is part of a codebook (cluster IDs together with their codewords)

Vector Quantization at Inference Time

1

When a new vector arrives, compare it with each vector in the codebook

2

Assign the new vector to whichever codeword's cluster is closest

3

Output the discrete symbol/integer for that codeword

Limitations of Simple Vector Quantization

- Difficult to represent complex acoustic detail
- Can lead to subpar reconstructions
- In many cases, speech is too rich to try to compress into one discrete code per frame
- How is this addressed?
  - **Residual Vector Quantization**

Residual Vector Quantization (RVQ)

- ENCODEC and many other audio tokenizers used residual vector quantization instead of simple VQ
- Key idea behind RVQ: Multiple codebooks arranged in a hierarchy
- How does this work?
  - Run standard VQ with a codebook
  - Then for an input embedding, take the codeword vector that is produced and find the difference between the input embedding and the codeword vector
  - Take the residual that results, and pass it through another vector quantizer
  - Repeat until we have the original codeword and seven residuals
  - Add these eight items together

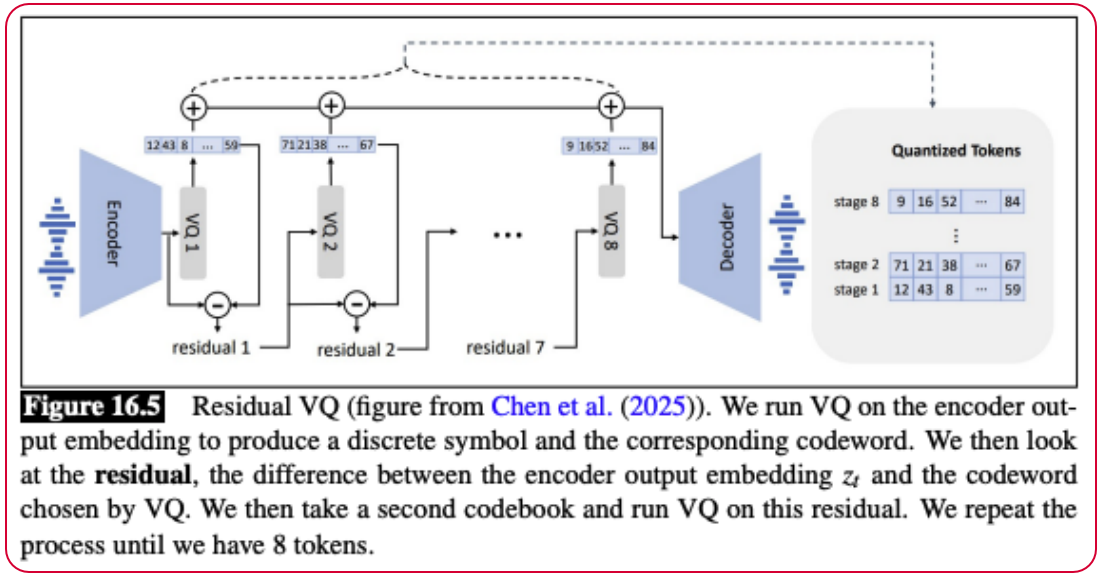
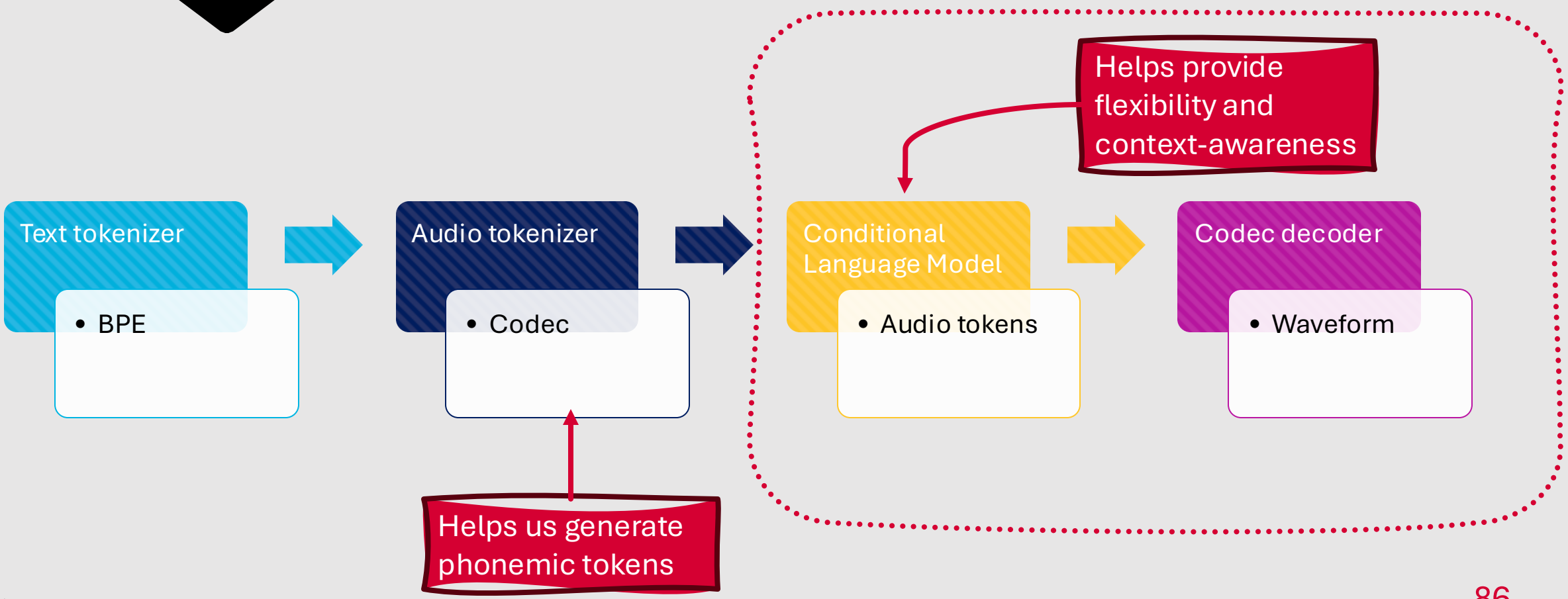


Figure 16.5 Residual VQ (figure from Chen et al. (2025)). We run VQ on the encoder output embedding to produce a discrete symbol and the corresponding codeword. We then look at the **residual**, the difference between the encoder output embedding z_t and the codeword chosen by VQ. We then take a second codebook and run VQ on this residual. We repeat the process until we have 8 tokens.

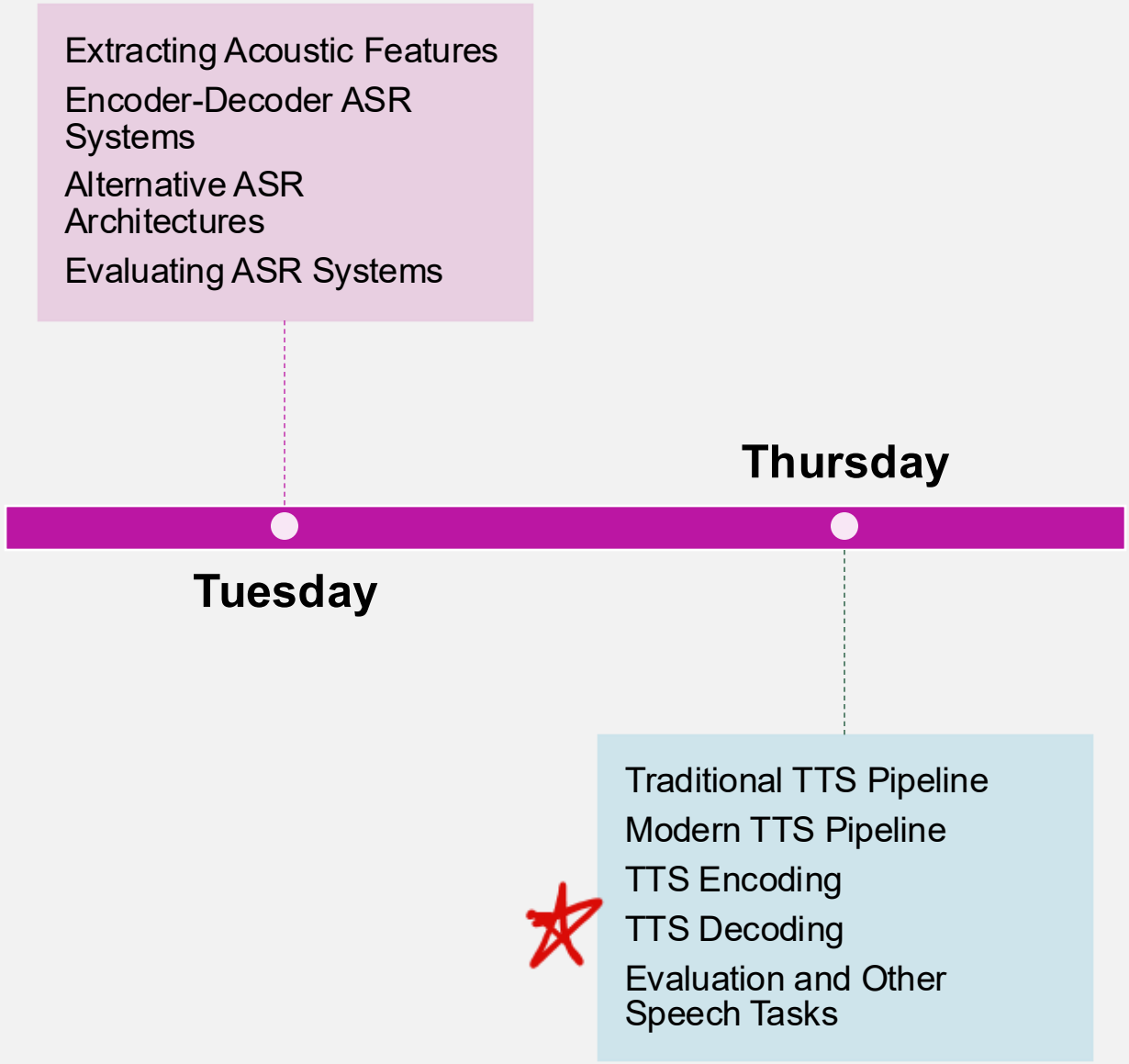
How do we train audio tokenizers?

- Typically done using an end-to-end process
- Input:
  - Audio waveform (typically 1 or 10 seconds extracted from the longer original waveform)
- Desired output:
  - Same waveform span (model is an autoencoder that learns to map to itself)
- Usually trained on large speech datasets
  - Common Voice
  - Audio Set: https://research.google.com/audioset/dataset/index.html

The Modern TTS Pipeline



This Week's Topics



Modern TTS systems perform as follows:

- Take an input text to synthesize and a sample of the voice to be used ✅
- Tokenize both, using BPE for the text and an audio codec for the speech ✅
- Use a language model to conditionally generate discrete audio tokens corresponding to the text prompt ❓

Why incorporate language modeling in TTS?

- Language models:
  - Learn long-range dependencies
  - Scale with dataset size and quality
  - Have been successfully applied in other multimodal settings
- As we've seen, a dominant trend in NLP right now is to treat most problems as sequence labeling/prediction tasks
  - This follows that trend!

However, some modifications to standard LLM setups may promote improved performance.

VALL-E: Popular TTS system that uses a two-stage conditional language model to generate audio tokens corresponding to the desired text

Why two stages?

- Architectural choice influenced by the hierarchical nature of the RVQ quantizer
  - Output of the first RVQ quantizer is the most important to the final speech
  - Subsequent quantizers contribute less and less residual information to the final signal

How does two-stage language modeling work?

- First, an autoregressive language model generates the first-quantizer codes to determine the coarse acoustic structure and timing
- Given those codes, a non-autoregressive language model is run seven times (one for each remaining quantizer) to generate residual codes in parallel, refining the audio with increasing detail

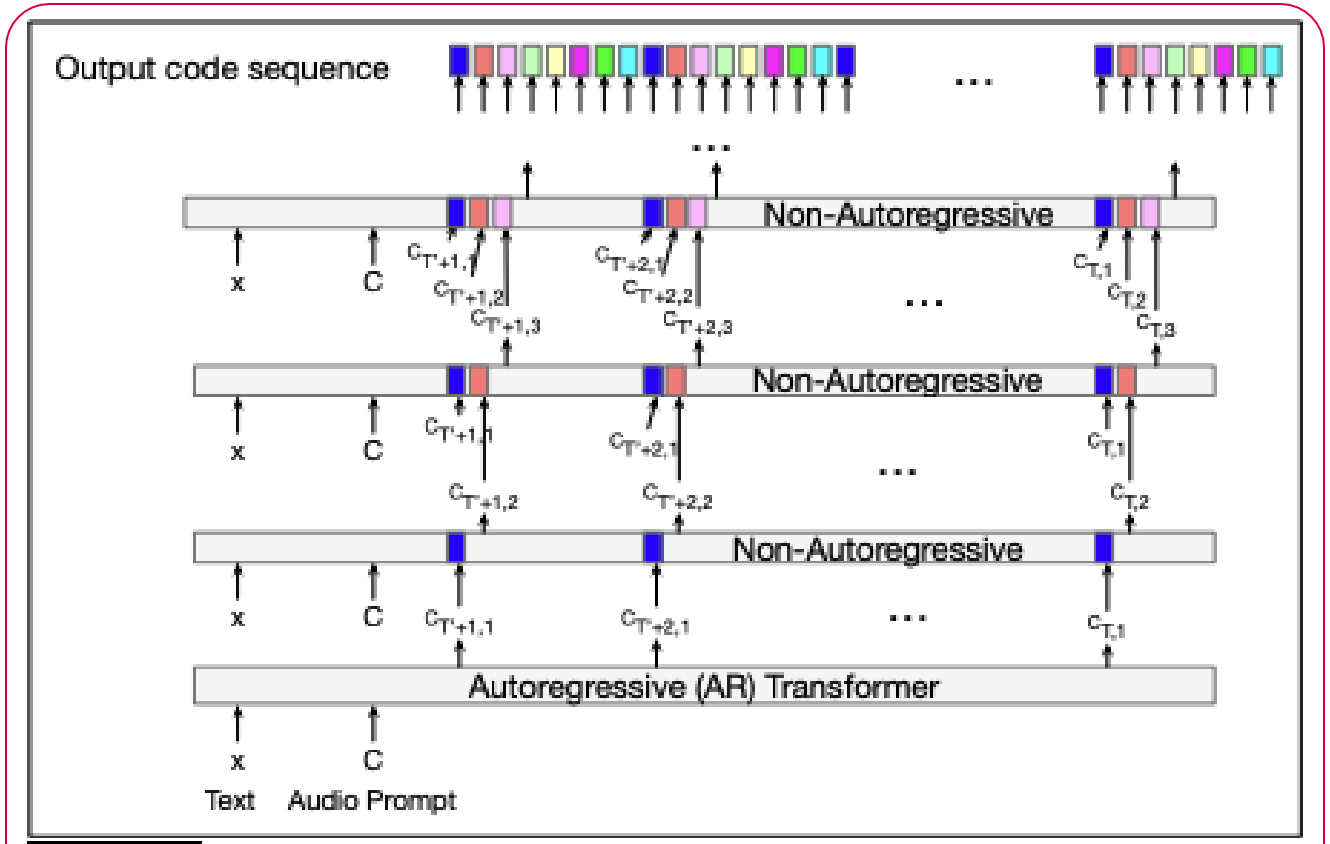


Figure 16.7 The 2-stage language modelling approach for VALL-E, showing the inference stage for the autoregressive transformer and the first 3 of the 7 non-autoregressive transformers. The output sequence of discrete audio codes is generated in two stages. First the autoregressive LM generates all the codes for the first quantizer from left to right. Then the non-autoregressive model is called 7 times to generate the remaining codes conditioned on all the codes from the preceding quantizer, including conditioning on the codes to the right.

How does two-stage language modeling work?

- First, an autoregressive language model generates the first-quantizer codes to determine the coarse acoustic structure and timing
- Given those codes, a non-autoregressive language model is run seven times (one for each remaining quantizer) to generate residual codes in parallel, refining the audio with increasing detail

Predicts one token at a time (powerful, but slow) to determine timing, phonetic content, prosody, utterance length, and most speaker characteristics

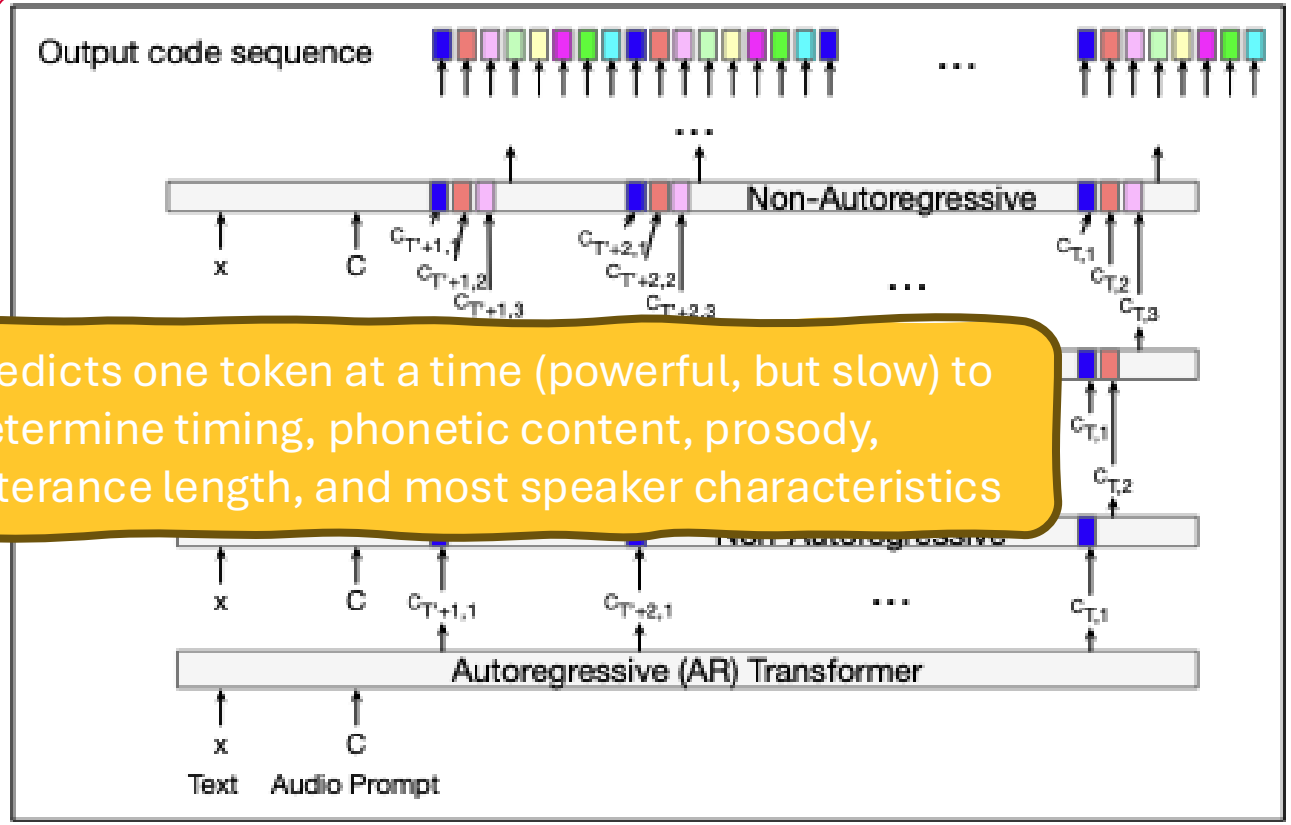
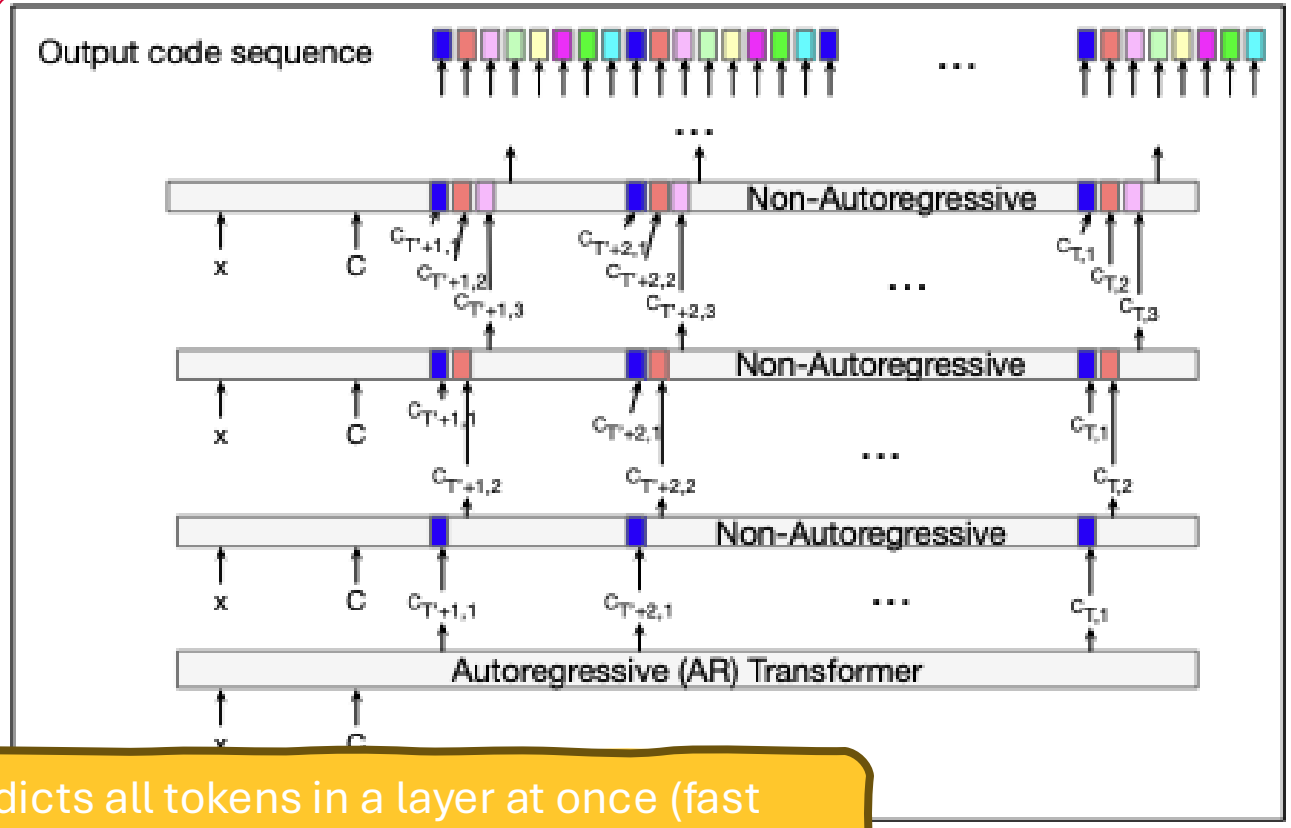


Figure 16.7 The 2-stage language modelling approach for VALL-E, showing the inference stage for the autoregressive transformer and the first 3 of the 7 non-autoregressive transformers. The output sequence of discrete audio codes is generated in two stages. First the autoregressive LM generates all the codes for the first quantizer from left to right. Then the non-autoregressive model is called 7 times to generate the remaining codes conditioned on all the codes from the preceding quantizer, including conditioning on the codes to the right.

How does two-stage language modeling work?

- First, an autoregressive language model generates the first-quantizer codes to determine the coarse acoustic structure and timing
- Given those codes, a non-autoregressive language model is run seven times (one for each remaining quantizer) to generate residual codes in parallel, refining the audio with increasing detail



Predicts all tokens in a layer at once (fast and efficient) to add more fine detail

ALL-E, showing the inference non-autoregressive transform-
ers. The output sequence of discrete audio codes is generated in two stages. First the autoregressive LM generates all the codes for the first quantizer from left to right. Then the non-autoregressive model is called 7 times to generate the remaining codes conditioned on all the codes from the preceding quantizer, including conditioning on the codes to the right.

Training Procedure

- Given an audio sample and its tokenized transcription, use a pretrained ENCODEC to convert the audio sample to its code matrix with eight codes per downsampled output vector
- Given the tokenized transcription and the code matrix, train the TTS as a conditional code language model to maximize the likelihood of the code matrix **C** conditioned on the tokenized transcription **x**:
  - $L = -\log p(\mathbf{C}|\mathbf{x})$

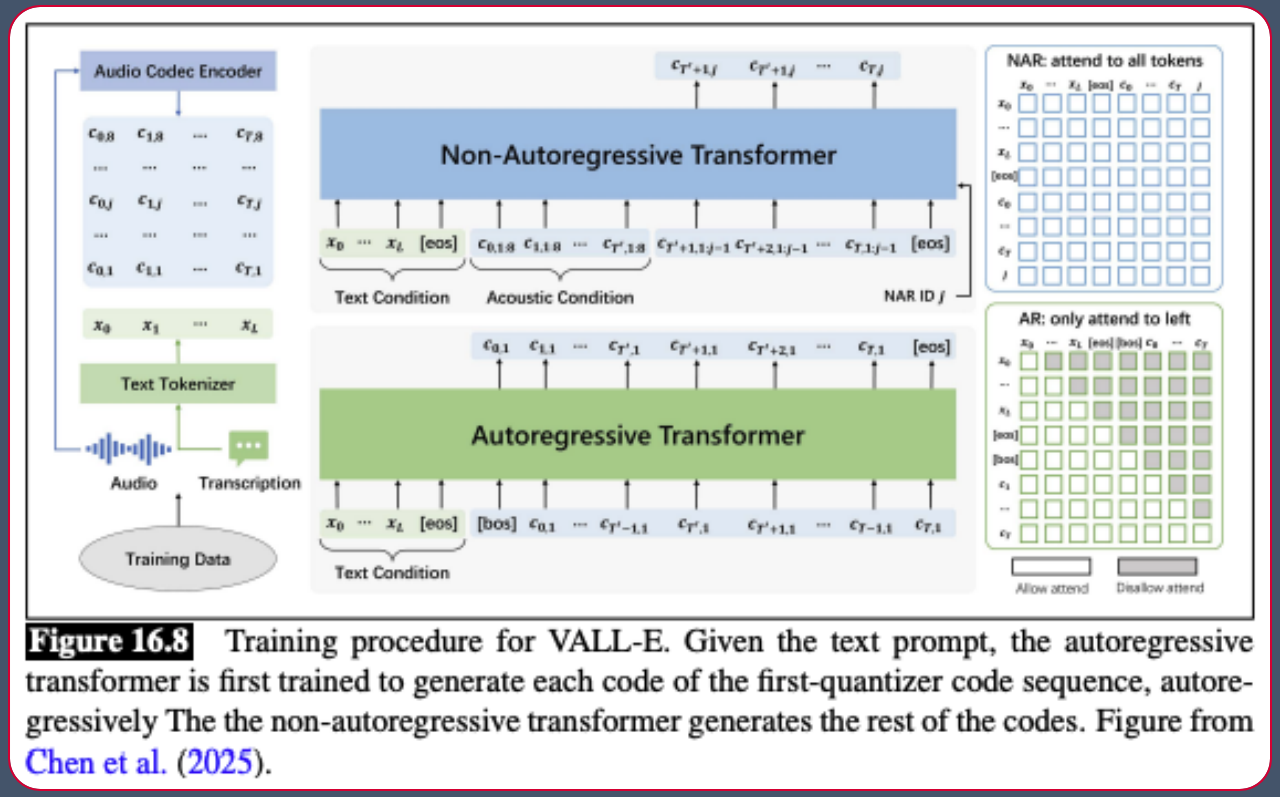


Figure 16.8 Training procedure for VALL-E. Given the text prompt, the autoregressive transformer is first trained to generate each code of the first-quantizer code sequence, autoregressively The the non-autoregressive transformer generates the rest of the codes. Figure from Chen et al. (2025).

Inference Procedure

- Given a text sequence to be spoken and a speech sample from some new speaker, for which we have the transcript:
  - First run the codec to get an acoustic code matrix for the speech sample
  - Concatenate the transcription of the speech sample to the text sequence to be spoken to create the full input prompt
  - Tokenize the prompt
  - Generate an acoustic code matrix conditioned on the text sequence and the full input prompt
  - Then, use the ENCODEC decoder to convert the generated tokens to an acoustic waveform

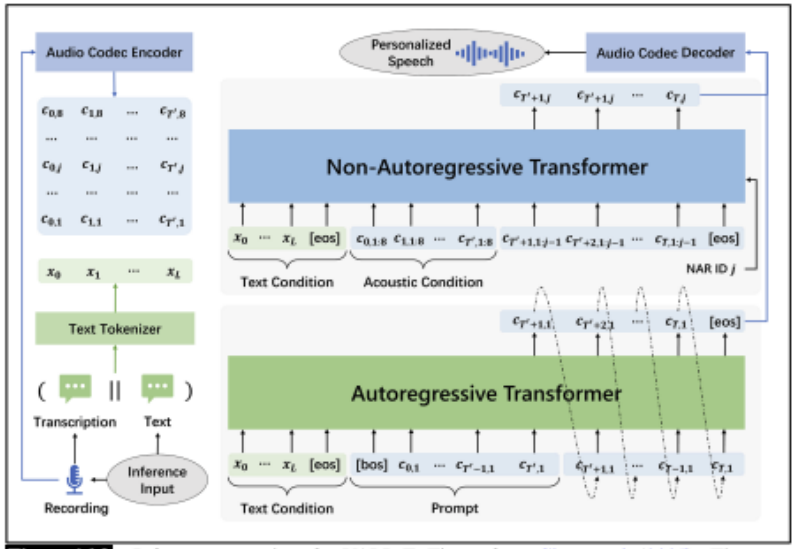
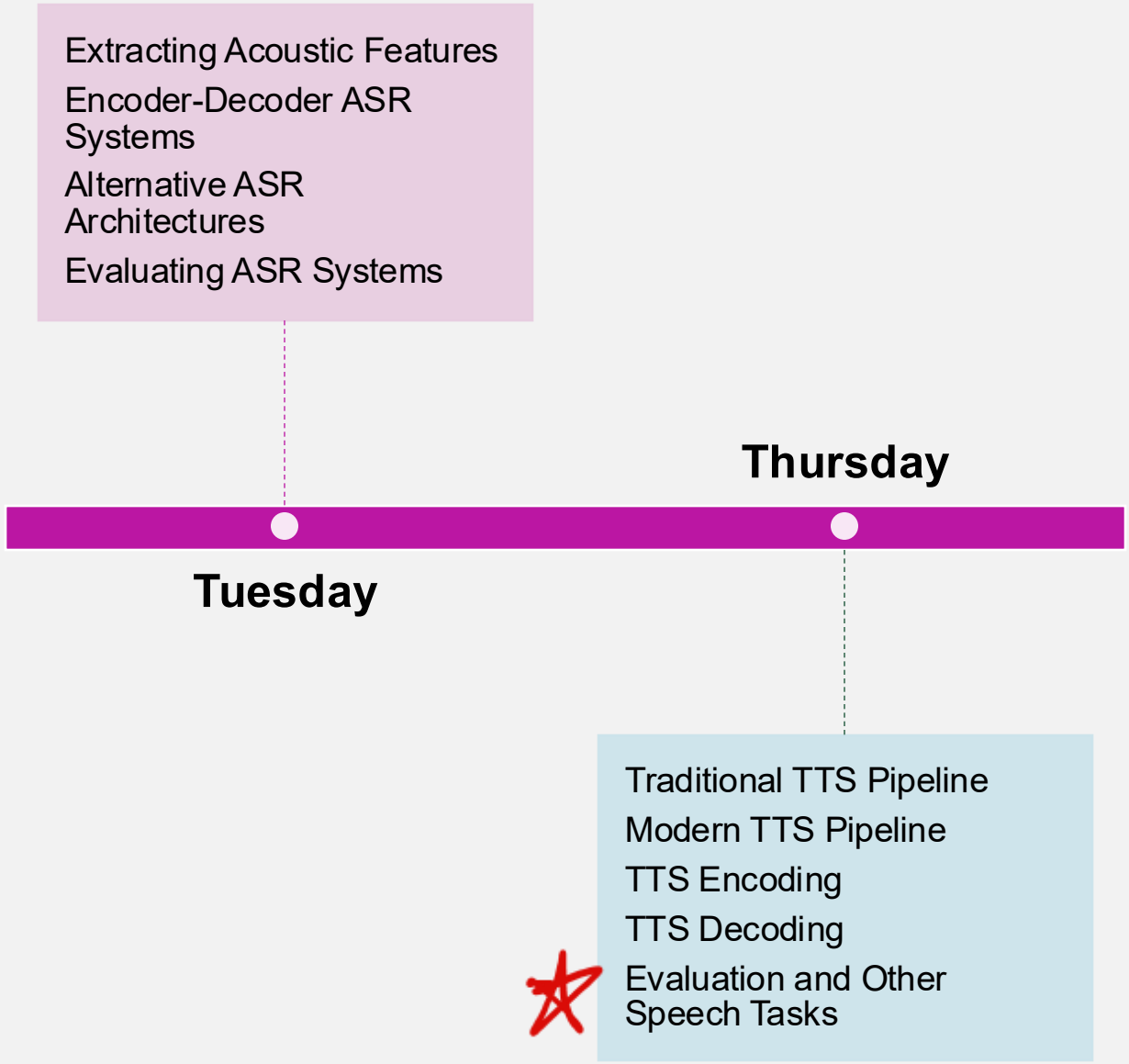


Figure 16.9 Inference procedure for VALL-E. Figure from Chen et al. (2025). The transcript for the 3 seconds of enrolled speech is first prepended to the text to be generated, and both the speech and text are tokenized. Next the autoregressive transformer starts generating the first codes $\mathbf{c}_{t'+1,1}$ conditioned on the transcript and acoustic prompt.

This Week's Topics



TTS Evaluation

- TTS systems are generally evaluated by human listeners
  - Play audio clip
  - Request a **mean opinion score (MOS)**
  - Compare to other systems based on their MOS scores on the same sentences
- **AB testing** can also be used
  - Play a synthesized sentence from system A
  - Play the same synthesized sentence from system B
  - Ask listener to rate which one is better
  - Compare the two systems to see which one has more sentences rated as being better

What about automated metrics?

- Disclaimer: Human evaluation works best for TTS!
- Automated metrics are sometimes added to provide more perspective
  - Generated output can be run through an ASR system and word error rate can be calculated for the ASR output
  - The voice used in the prompt (for modern TTS systems) and the output voice can be passed through a speaker verification system, and the resulting score can be used as a similarity score

Speech processing isn't limited to ASR and TTS!

Application-specific tasks:

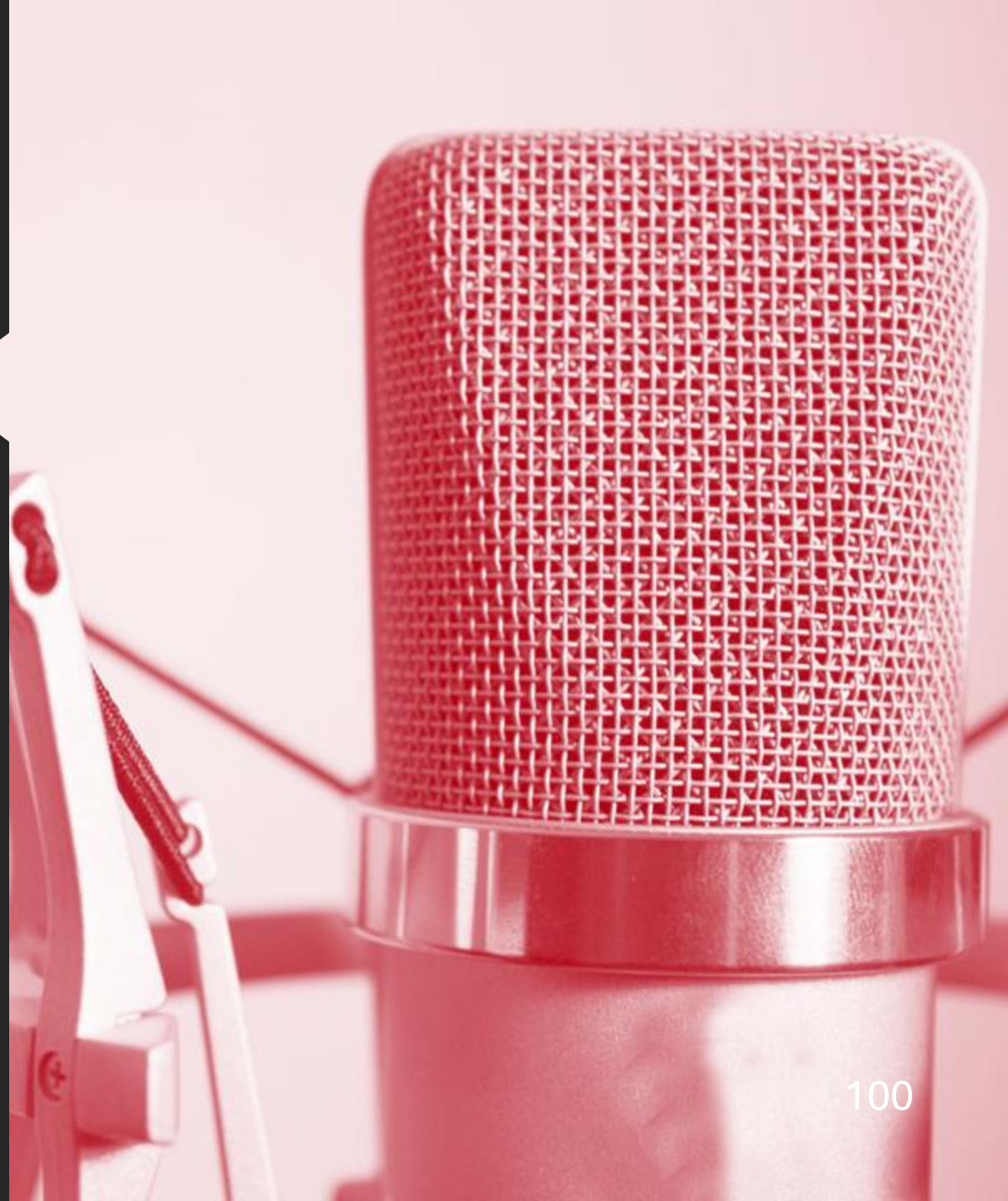
- Wake word detection
- Language identification

Multiparty interaction tasks:

- Speaker diarization
- Speaker recognition

Wake Word Detection

- Allows for small-scale ASR to be incorporated into edge devices
  - Maintains privacy by transmitting as little speech as possible
- Goal: Detect a word or short phrase (usually to wake up a voice-enabled assistant)
- Typically uses a scaled-down ASR model, followed by a whole-word classifier



Language Identification

Goal: Given an acoustic file, identify which language is being spoken

Important for creating multilingual ASR models and scraping/organizing multilingual speech data

Challenging to do with small segments especially (solving this can help lead to improved ASR in code-switched settings!)

Speaker Diarization

- Goal: Determine who spoke when in multi-speaker audio recordings
- Good speaker diarization systems should be able to mark the start and end of each speaker's turns in an interaction
- Useful for transcribing meetings, classroom speech, or medical interactions
- Often these systems work by:
  - Using voice activity detection models to find segments of continuous speech
  - Extracting speaker embedding vectors
  - Clustering the vectors to group segments likely from the same speaker
- Recent work has also explored the use of end-to-end approaches to map directly from input speech to a sequence of speaker labels for each frame

Speaker Recognition

- Involves two subtasks:
  - **Speaker verification:** Determine whether this is speaker X or not
  - **Speaker identification:** Assign the speaker to one of a potentially large database of users
- Often useful for enhancing security in spoken language systems, for instance those involving the access of personal information over the phone

What's next in TTS?

- Remaining challenges:
  - Implementing multilingual TTS
  - Generating natural prosody
- Increasing push (but not much success so far) towards unified spoken language models

Summary: TTS

- **Text-to-speech synthesizers** map strings of text to acoustic signals
- Earlier systems often did that by predicting a **spectrogram** for the text string, and then **vocoding** from sequences of spectral values to acoustic waveforms
- More recent TTS systems heavily incorporate language modeling into their approach, often by first tokenizing input text and audio and then using the tokenized content to prompt a language model and generate acoustic waveforms for the predicted audio tokens
- **Vector quantization** is a way to turn a series of vectors into discrete symbols
- **Residual vector quantization** is a hierarchical version of vector quantization that first quantizes a vector, and then repeatedly quantizes the residual between the resulting codeword and the input vector
- TTS systems are often evaluated by **human raters**