# CS 224S / Linguist 285
# Spoken Language Processing

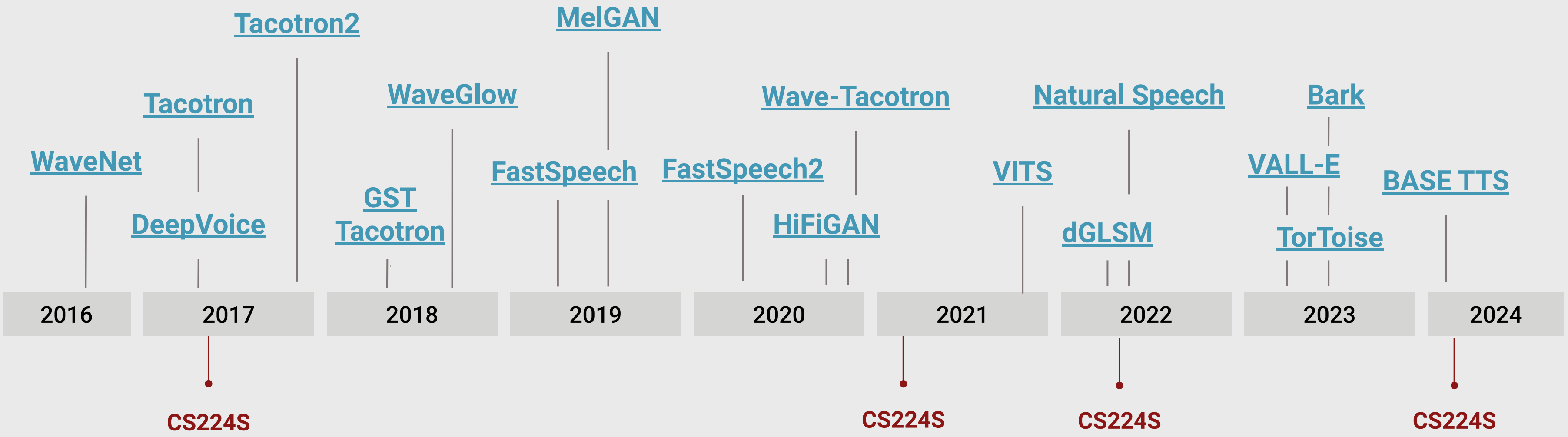
Andrew Maas | Stanford University | Spring 2024

## Lecture 6: Deep Learning for TTS



Slide contributions by Alex Barron & Mike Wu

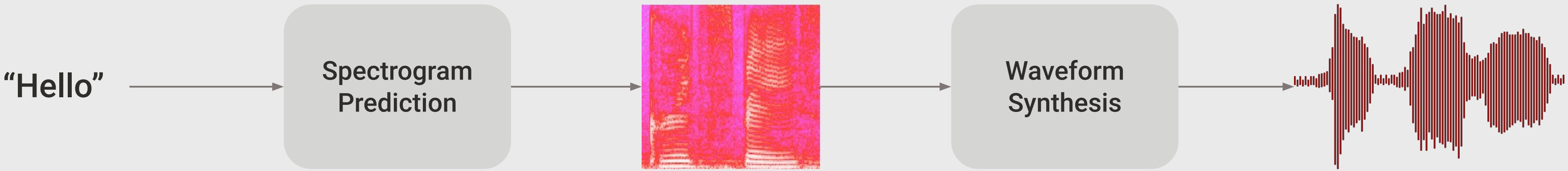
# Rapid Progress in TTS Over the Last Few Years



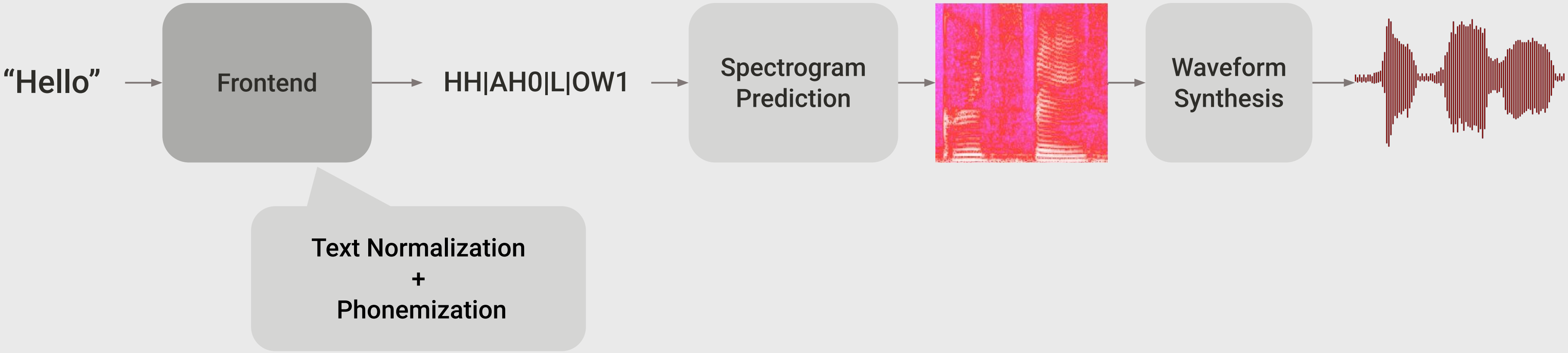
# Outline

- Text to Spectrogram Models
- Speaker and Style Embeddings
- Spectrogram to Audio Models (Vocoders)

# Neural TTS Paradigm



# Neural TTS Paradigm



# Why Neural TTS?

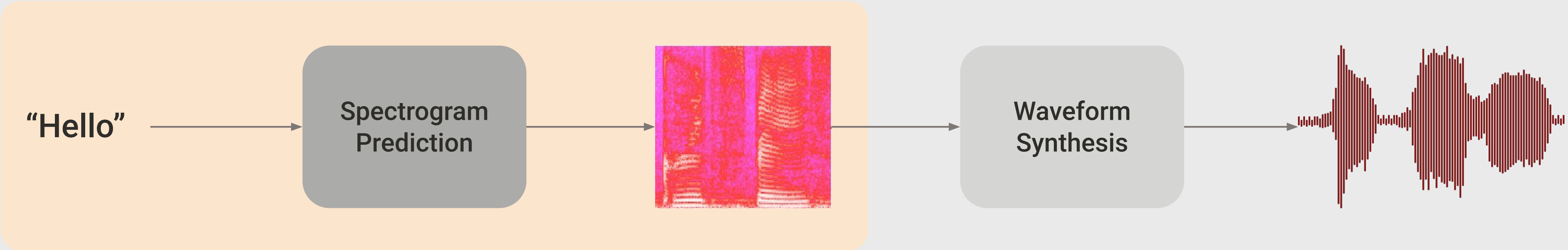
- Upper bound on quality is higher
- Works with a wider variety of datasets
- Much more easily extended for speaker/style customization
- Many fewer components to train than traditional TTS
- Single speaker datasets are 1-10Gb e.g. LJSpeech
- You can get decent results in a night on a solid GPU with most models

# Why Use Intermediate Spectrograms?

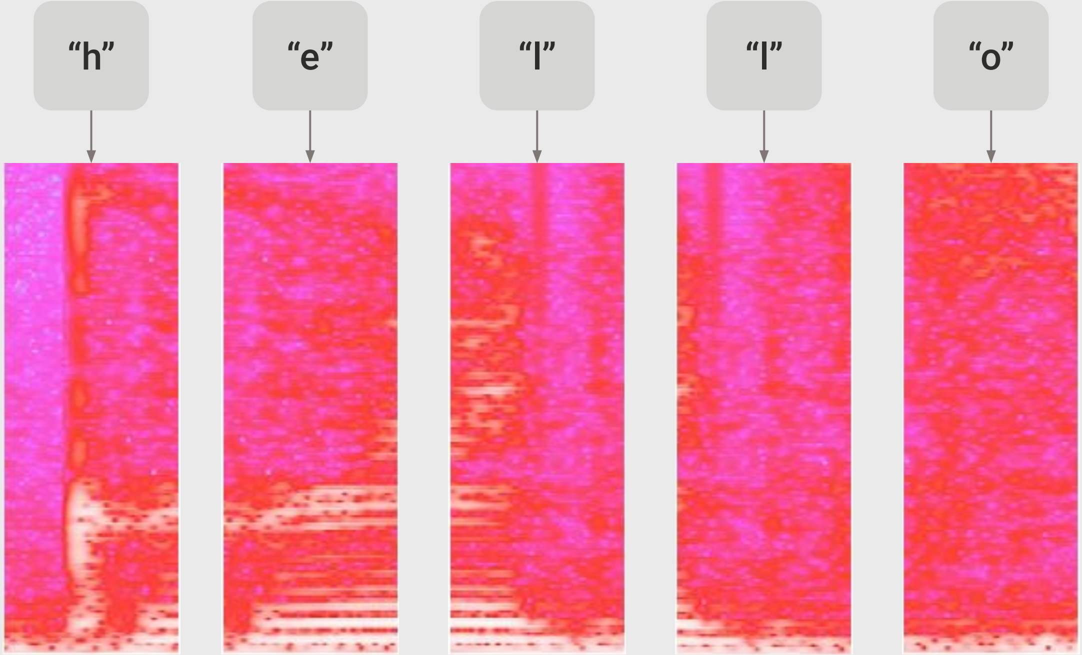
- Prosodic/phonemic aspects of speech can be modelled without phase information
- Allows focus on human speech frequency bands with mel filters
- STFT chunks speech into frames of a useful duration for phoneme and prosody modeling
- Fast to generate thanks to FFT
- Separate model can be used to fill in the phase

# Text to Spectrogram Models

# Neural TTS Paradigm

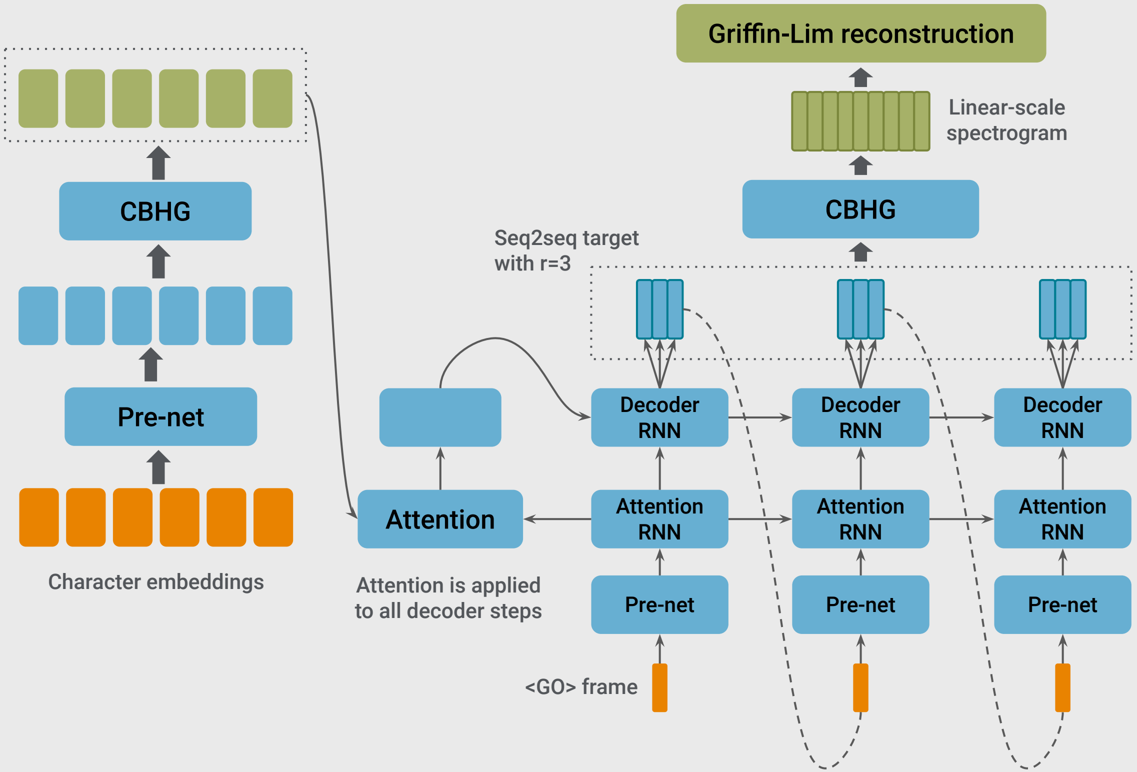


# Sequence to Sequence Problem



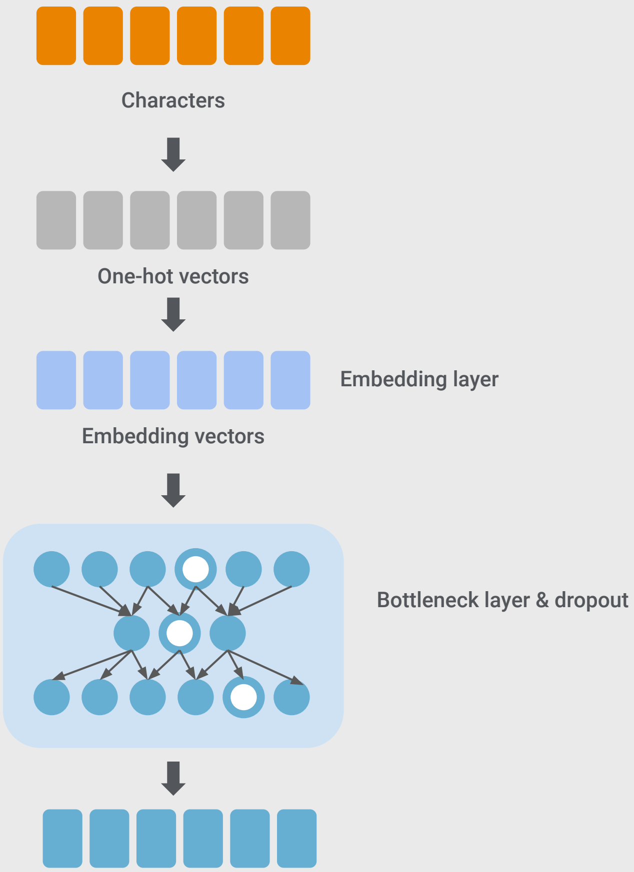
# Tacotron

- Encoder decoder model with attention
- Predicts mel spectrograms from character inputs
- “Information bottleneck” in
  pre-net crucial for regularization



# Pre-Net

Maps characters into a continuous vector

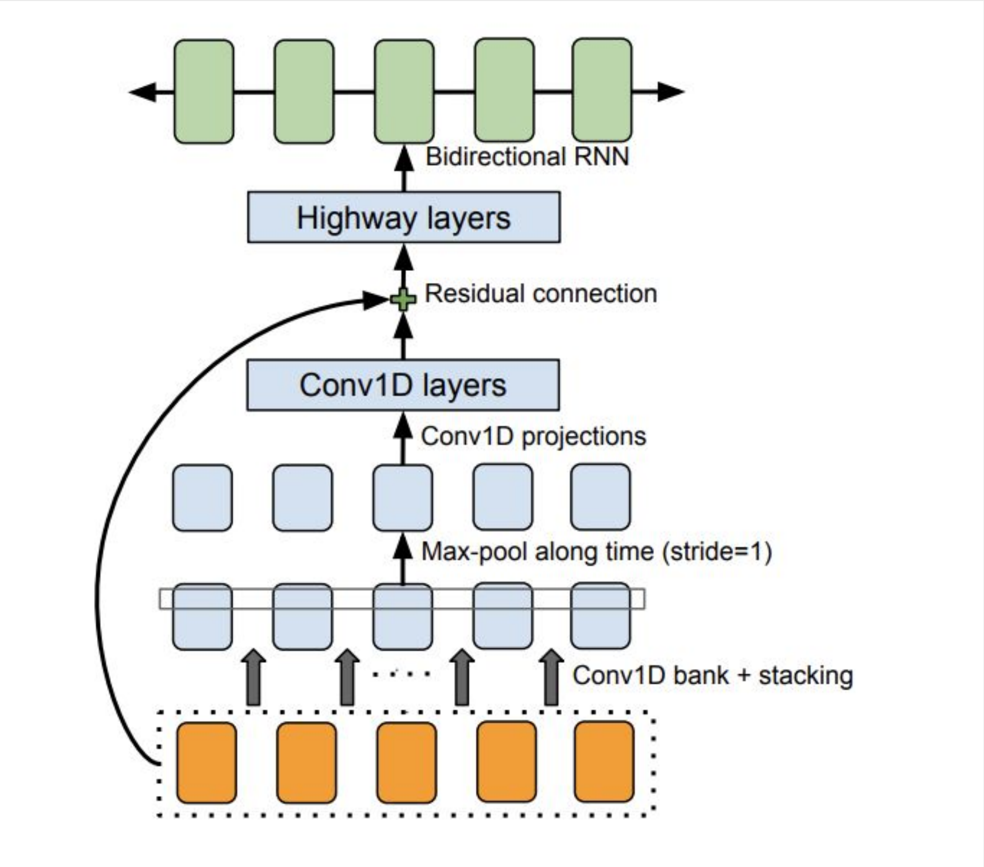


# CBHG Encoder

This is a special multi-layer RNN that includes a convolutional layer.

Mixes information 2 ways:

1. With neighboring characters via conv1d
2. Throughout entire sequence with GRU



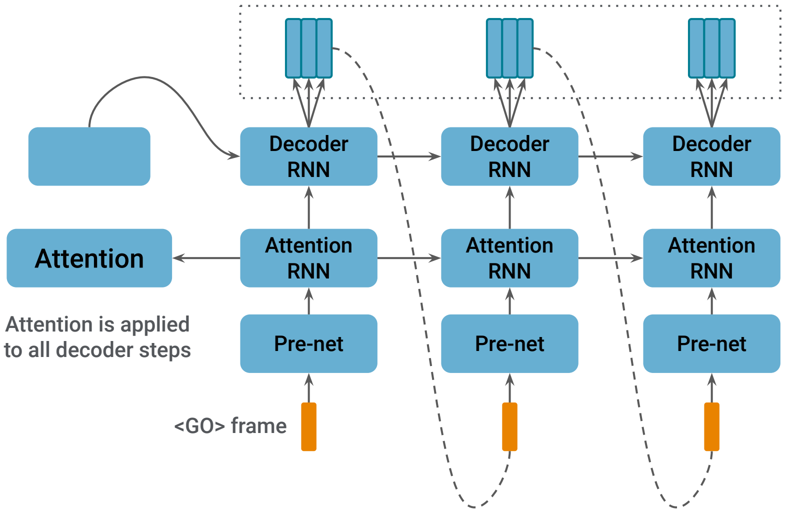
# Tacotron Decoder

$$\{\boldsymbol{h}_j\}_{j=1}^{L} = \text{Encoder}(\{\boldsymbol{x}_j\}_{j=1}^{L})$$

$$\boldsymbol{s}_i = \text{RNN}_{\text{Att}}(\boldsymbol{s}_{i-1}, \boldsymbol{c}_{i-1}, \boldsymbol{y}_{i-1})$$

$$\boldsymbol{\alpha}_i = \text{Attention}(\boldsymbol{s}_i, \ \ldots) \qquad \boldsymbol{c}_i = \sum_j \alpha_{i,j} \boldsymbol{h}_j$$

$$\boldsymbol{d}_i = \text{RNN}_{\text{Dec}}(\boldsymbol{d}_{i-1}, \boldsymbol{c}_i, \boldsymbol{s}_i) \qquad \boldsymbol{y}_i = f_{\text{o}}(\boldsymbol{d}_i)$$



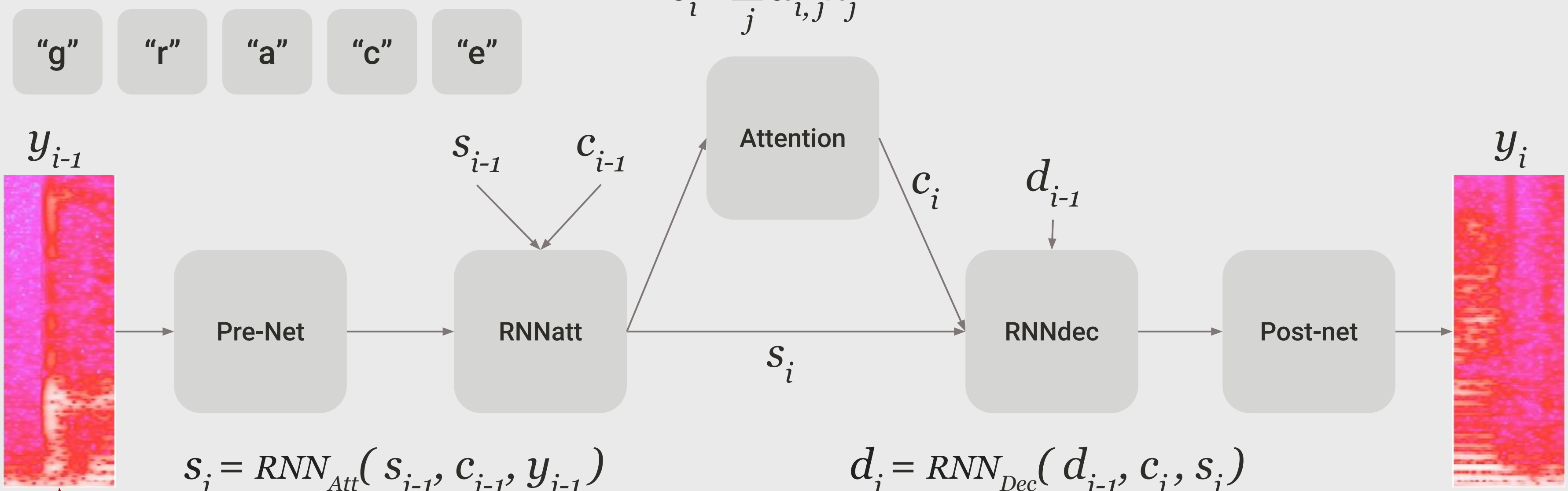

Wang et al 2017

Optionally takes previous alignment, encoder states

$$\{h_j\}_{j=1}^{L} = \text{Encoder}\left(\{x_j\}\right)_{j=1}^{L}$$

$$\alpha_i = \text{Attention}\,(s_i, \ldots)$$

$$c_i = \sum_j \alpha_{i,j} h_j$$




$$s_i = RNN_{Att}(\, s_{i-1}, c_{i-1}, y_{i-1}\,)$$

$$d_i = RNN_{Dec}(\, d_{i-1}, c_i, s_i)$$

Can be 1-5 mel frames (reduction factor)

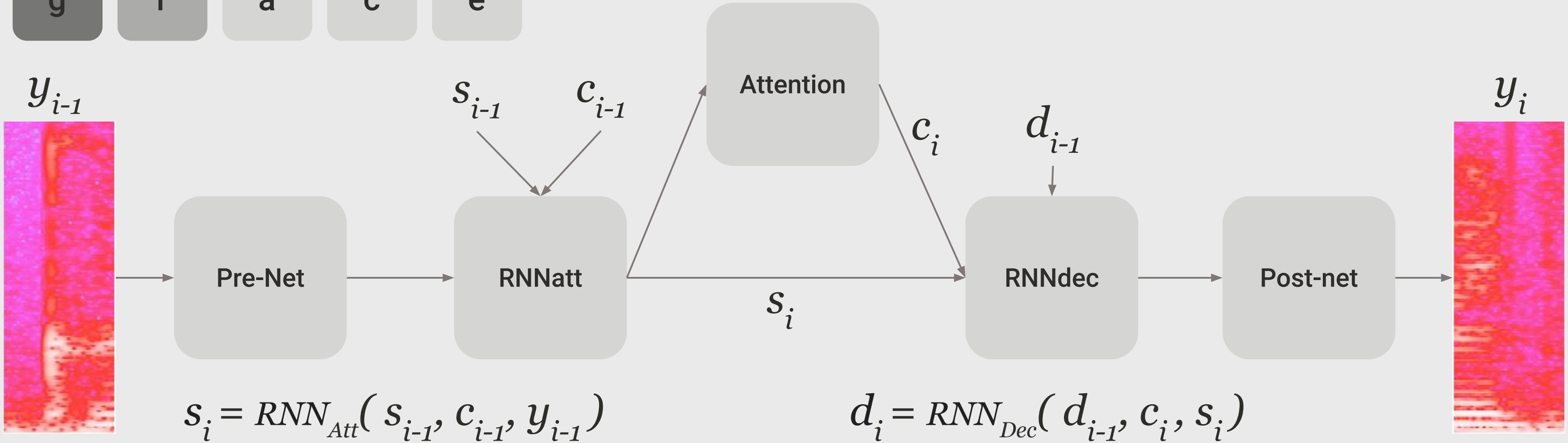
# j = 0

$$\{h_j\}_{j=1}^{L} = \text{Encoder}\left(\{x_j\}_{j=1}^{L}\right)$$

$$\alpha_i = \text{Attention}\,(s_i, \ldots)$$

$$c_i = \sum_j \alpha_{i,j} h_j$$

"g" "r" "a" "c" "e"




$$s_i = RNN_{Att}(s_{i-1}, c_{i-1}, y_{i-1})$$

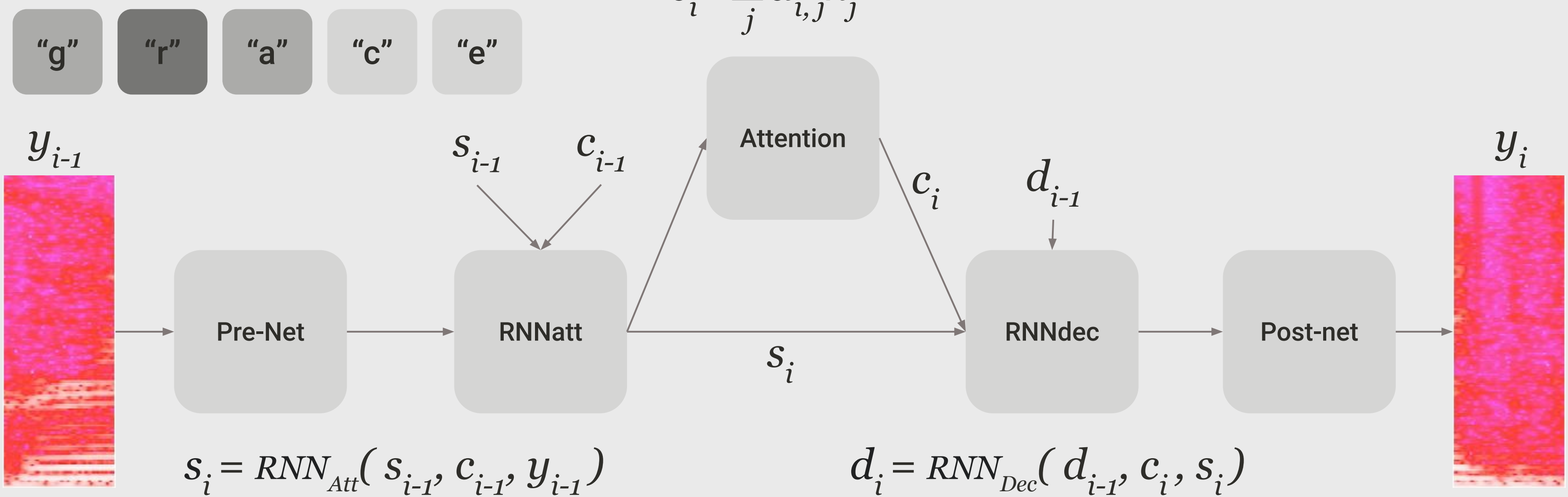
$$d_i = RNN_{Dec}(d_{i-1}, c_i, s_i)$$

# j = 1

$$\{h_j\}_{j=1}^{L} = \text{Encoder}\left(\{x_j\}_{j=1}^{L}\right)$$

$$\alpha_i = \text{Attention}\,(s_i, \ldots)$$

$$c_i = \sum_j \alpha_{i,j} h_j$$




$$s_i = RNN_{Att}(s_{i-1}, c_{i-1}, y_{i-1})$$

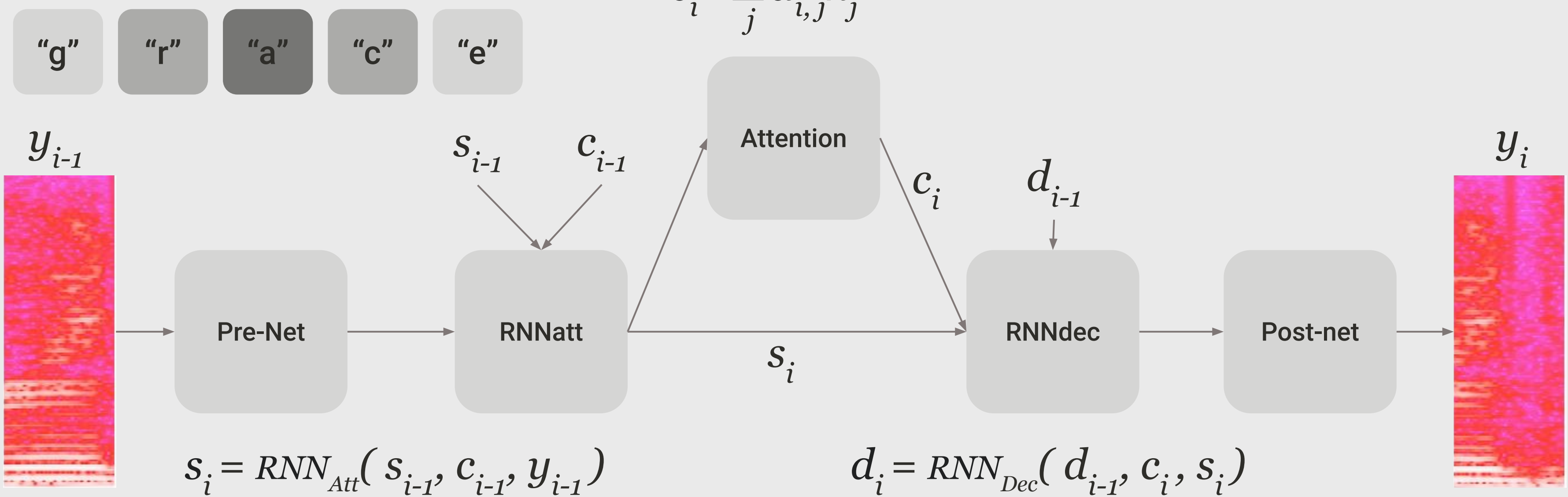
$$d_i = RNN_{Dec}(d_{i-1}, c_i, s_i)$$

# j = 2

$\{h_j\}_{j=1}^{L} = \text{Encoder}\ (\{x_j\})_{j=1}^{L}$

$\alpha_i = \text{Attention}\ (s_i, \ldots)$

$c_i = \sum_j \alpha_{i,j} h_j$




$s_i = RNN_{Att}(s_{i-1}, c_{i-1}, y_{i-1})$

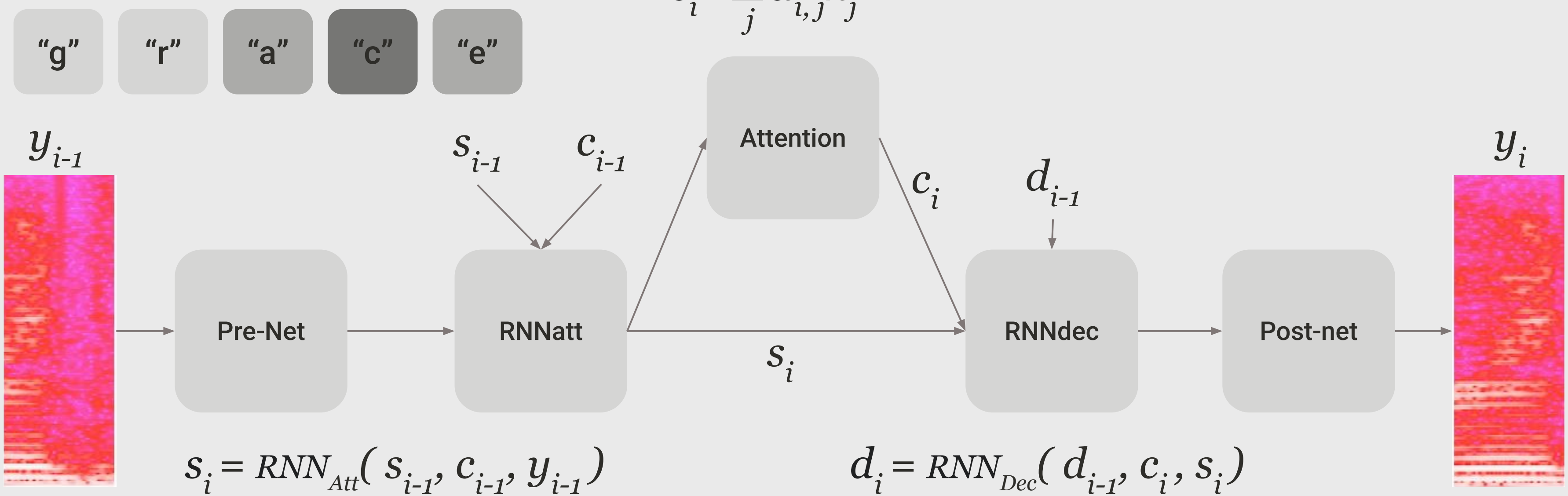
$d_i = RNN_{Dec}(d_{i-1}, c_i, s_i)$

# j = 3

$$\{h_j\}_{j=1}^{L} = \text{Encoder}(\{x_j\}_{j=1}^{L})$$

$$\alpha_i = \text{Attention}(s_i, \ldots)$$

$$c_i = \sum_j \alpha_{i,j} h_j$$




$$s_i = RNN_{Att}(s_{i-1}, c_{i-1}, y_{i-1})$$

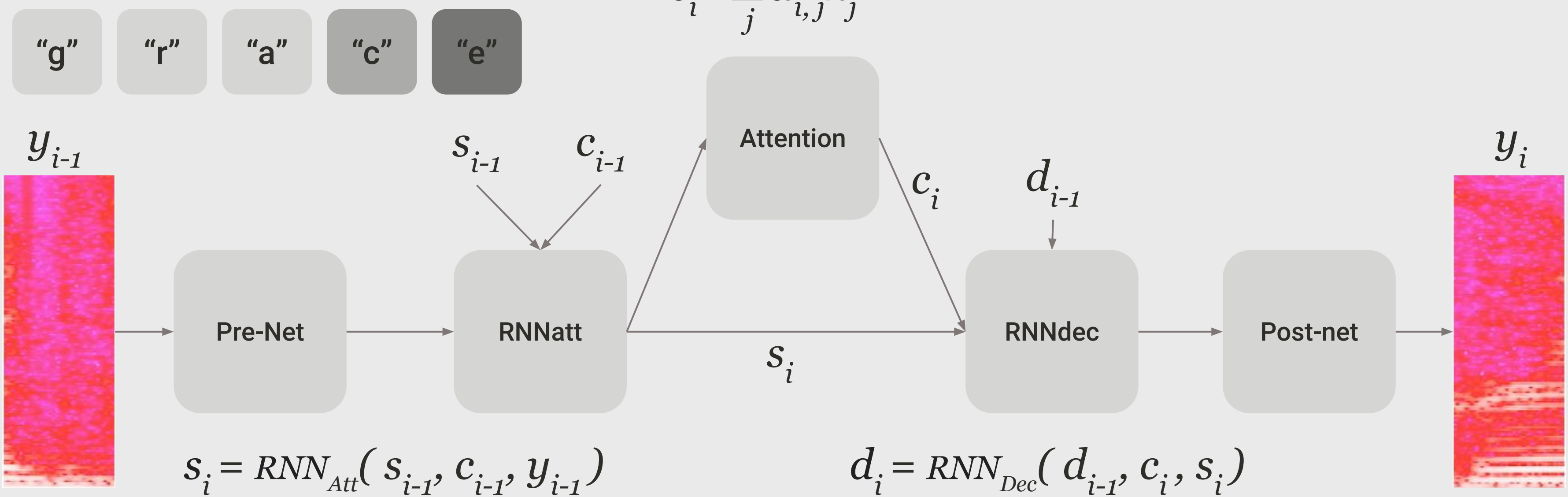
$$d_i = RNN_{Dec}(d_{i-1}, c_i, s_i)$$

# j = 4

$$\{h_j\}_{j=1}^{L} = \text{Encoder}\,(\{x_j\})_{j=1}^{L}$$

$$\alpha_i = \text{Attention}\,(s_i, \ldots)$$

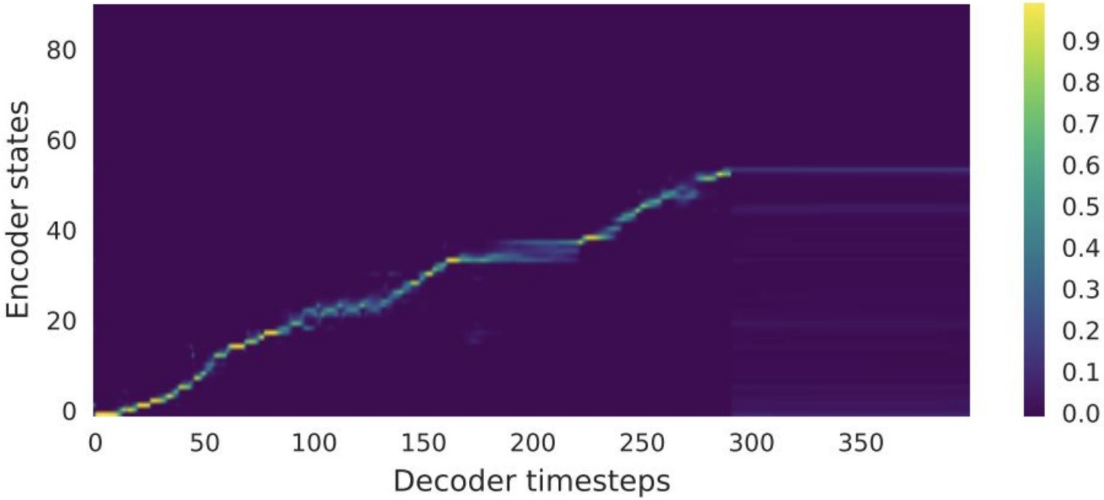
$$c_i = \sum_j \alpha_{i,j} h_j$$




$$s_i = RNN_{Att}(\,s_{i-1}, c_{i-1}, y_{i-1}\,)$$

$$d_i = RNN_{Dec}(\,d_{i-1}, c_i, s_i)$$

# Attention Reveals Alignment

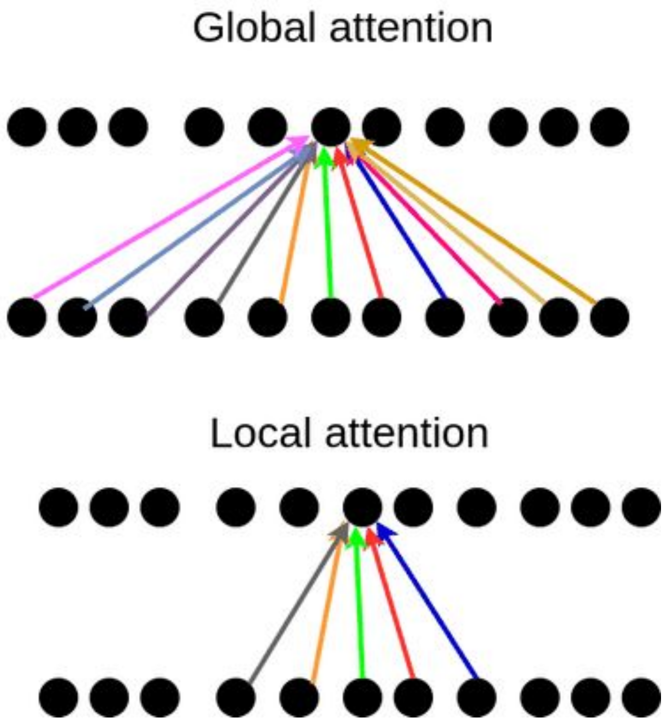



Wang et al 2017

[Tacotron samples]

# Many Forms of Attention

- Content Based (Bahdanau)
- Location Sensitive
- Location Relative (GMM, DCA)



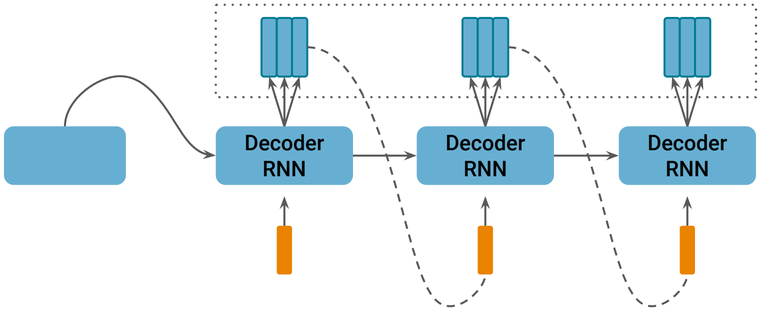

Wang et al 2017
https://theaisummer.com/attention

# Deep dive: FastSpeech 1 & 2

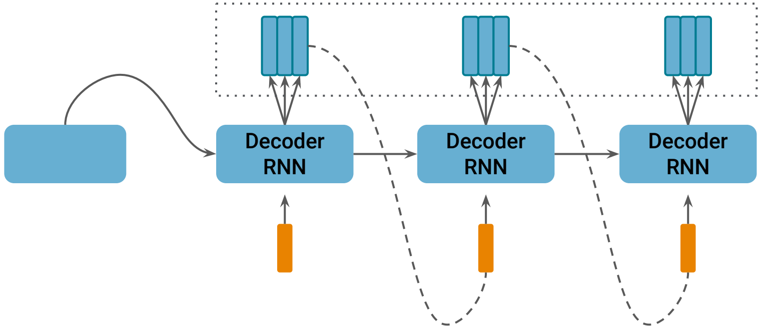
# Motivation

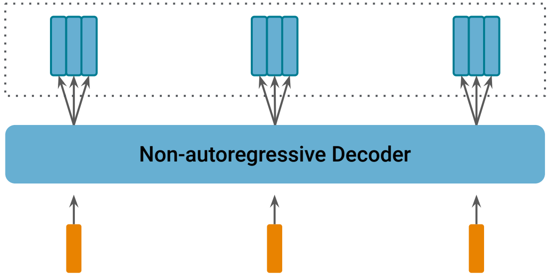
Disadvantages of auto-regressive generation:

- <u>*Slow:*</u> It is by definition step by step. Spectrogram sequences can be 1000s of steps long.
- <u>Error prone:</u> If an error happens on step 5, then it affects steps 6 to 1000.



# Motivation



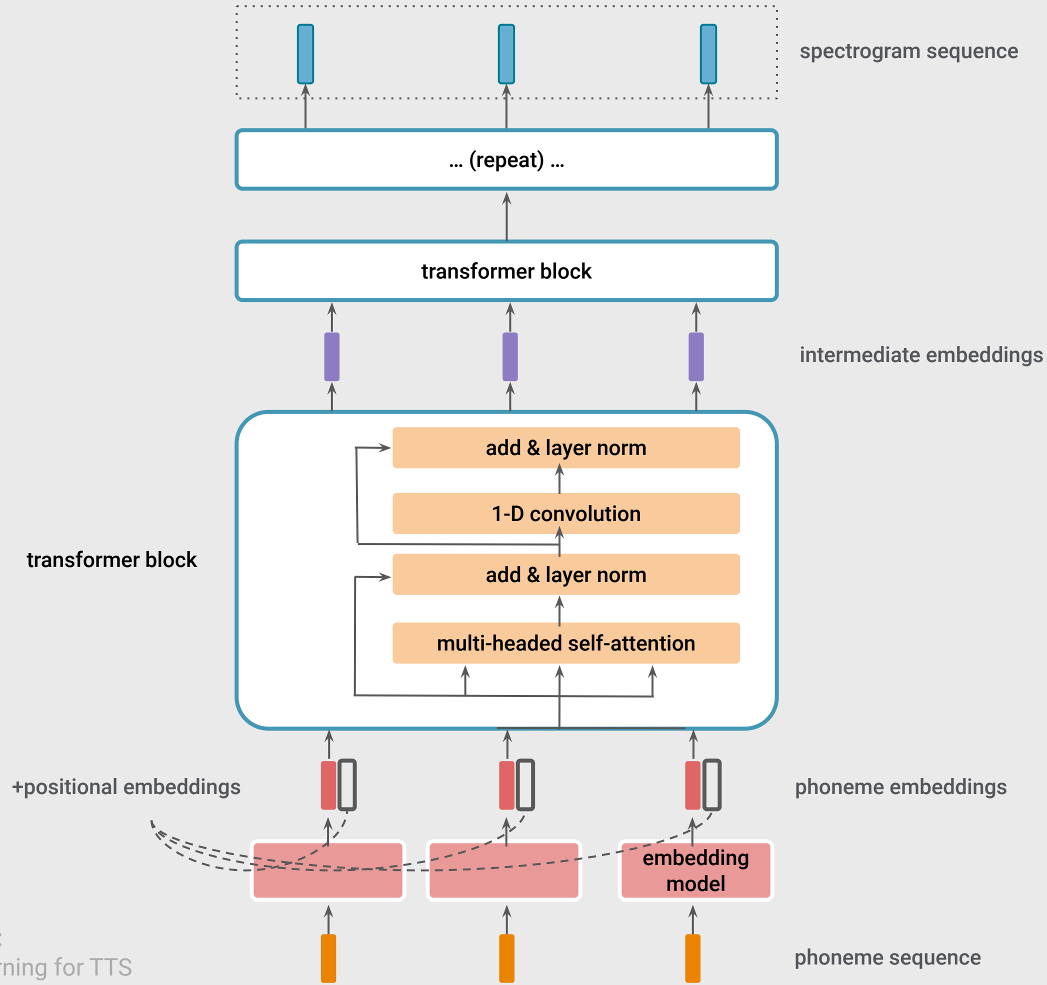

What if we could do this?

- ***Fast***: everything happens at once.
- **Less error prone**: no forward propagation.

# Feed forward Transformers

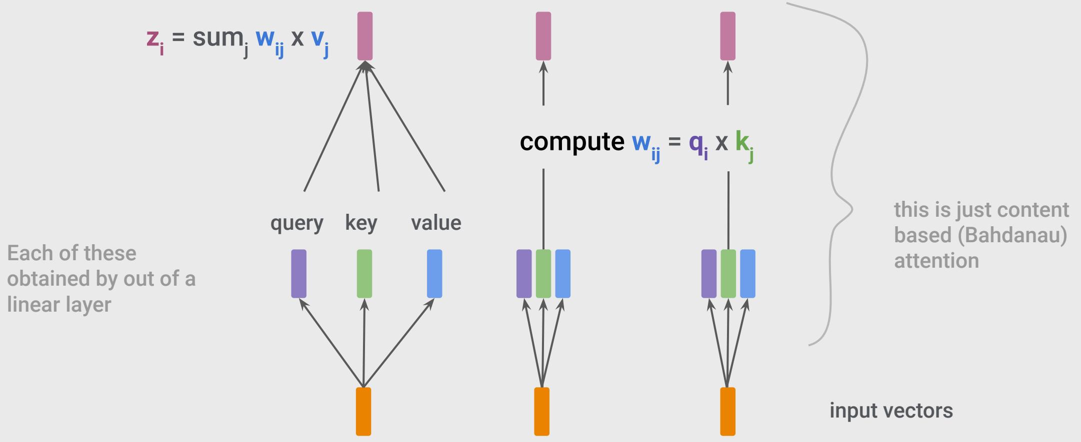
Borrow recent NLP breakthroughs for non-autoregressive modeling: **transformers**.

- Same architecture as LLMs e.g. GPT-4, Llama, etc.
- Stacked attention layers to mix information in the input phonemes
- Each phoneme input is mapped to a predicted mel-spectrogram



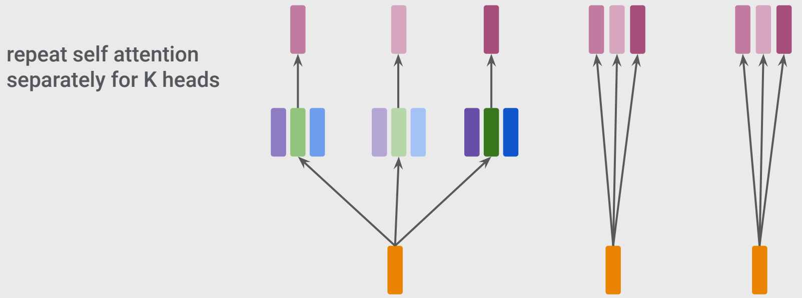
# Self attention

A self attention layer mixes information between a sequence of embeddings.




# Multi-headed attention

Goal is to provide diversity in what is being focused on

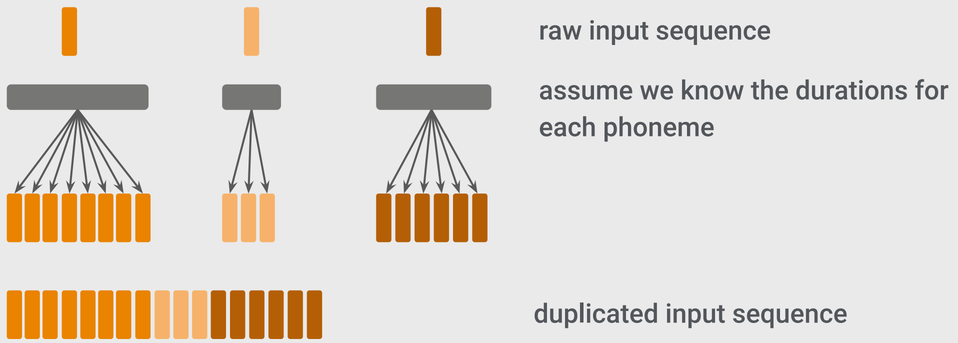


# Length Regulator

One of the properties of a transformer is that a sequence of N input tokens produces N output tokens.

**But this doesn't work for us!** Spectrogram sequences are often much longer than phoneme sequences.

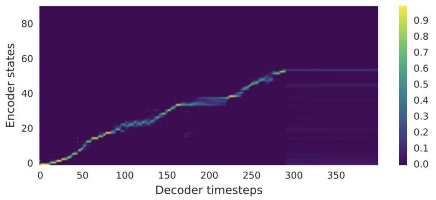
The trick is that we will **duplicate** the phonemes based on their duration. This way, a longer lasting phoneme will produce a longer sequence of spectrograms.



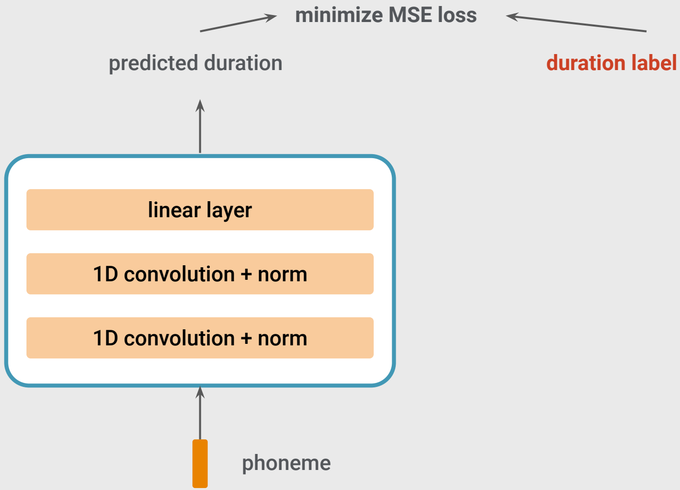
# Duration Predictor

To do the length correction, we need to know the duration from phonemes alone.

How do we do this?




Train a separate model jointly to predict the length of the mel-spectrograms for each phoneme.

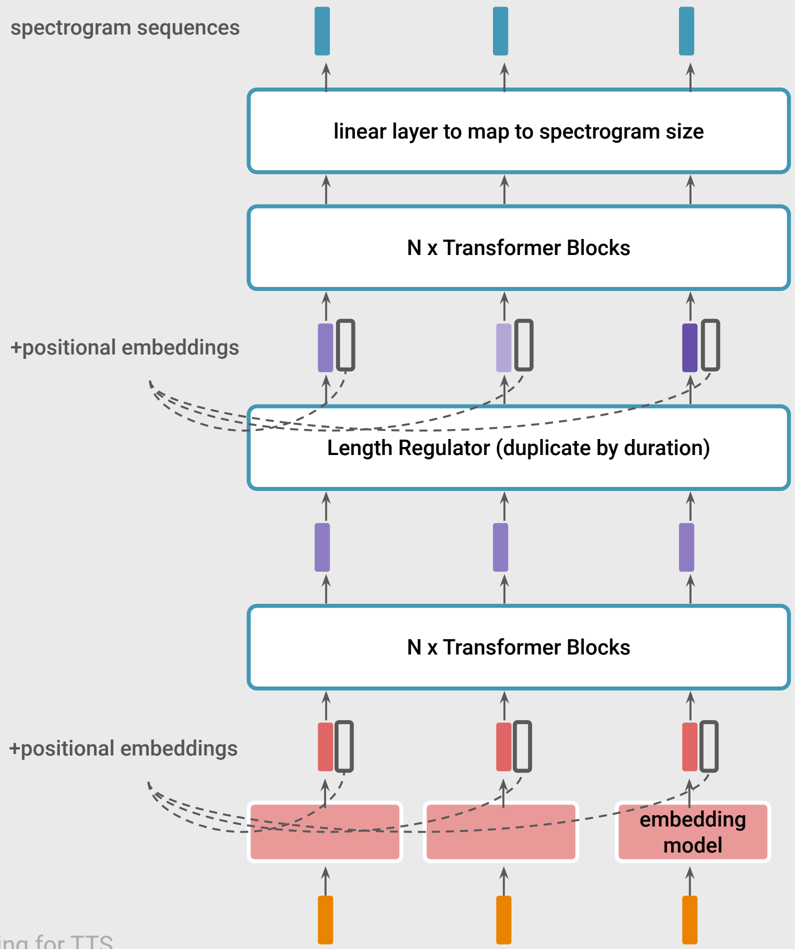



## How do we get the duration label?

HACK: take a pretrained autoregressive TTS model, and estimate duration using attention from phonemes to spectrograms. Use that as label for this model.

# FastSpeech

Put together all the components and stack transformer layers.



[Fastspeech2 samples]

# Wait... One-to-Many?

There are lots of ways to say the same phonemes, depending on the speed, pitch, energy of the speaker. All of these possible answers are "right", and having multiple right answers is bad for training.

## Which one should the transformer model generate?

# Hack: Pretrained TTS Model

Like getting durations, we again rely on a pretrained autoregressive TTS model. Mel spectrograms are generated from the autoregressive model and use as ground truth for training FastSpeech.

## Why does this solve the one-to-many problem?

We basically ask the model to favor whatever spectrogram the autoregressive model chose. This is equivalent to *knowledge distillation*.

# FastSpeech2

FastSpeech used a few hacks that rely on a pretrained autoregressive model. FastSpeech2 makes a few small changes to replace that dependency and some extra bells and whistles.

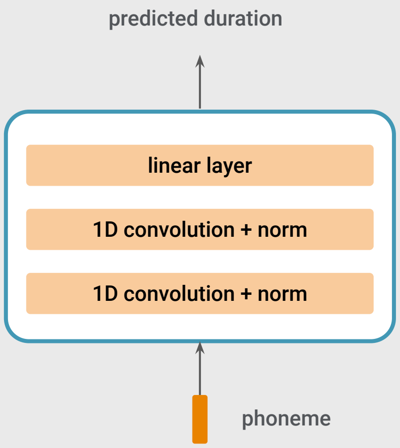
# Variance Adaptors

Decide "variance  information"

- Duration
- Pitch
- Energy

using the phoneme so that the one-to-many problem becomes one-to-one.

## Duration / Energy / Pitch Predictor



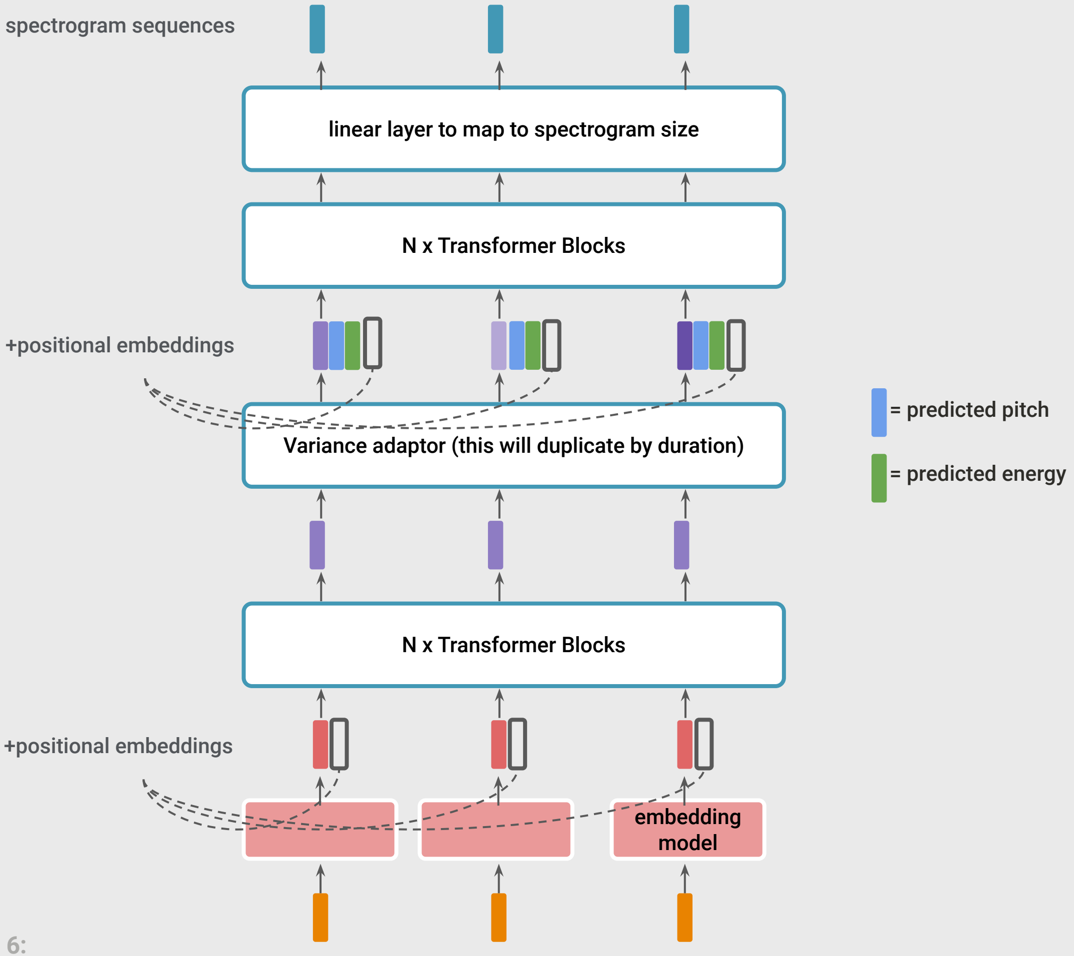

For each of three, we train a separate small model to predict them from the phoneme sequence. The labels are extracted using deterministic tools.

- **Duration** Use the Montreal forced alignment tool rather than a pretrained autoregressive model.
- **Energy** Treat $L_2$ norm of amplitude of the STFT of the frame as label.
- **Pitch** Use continuous wavelet transforms (CWT) to produce pitch spectrograms.

# FastSpeech2

Add extra variance information to the phoneme embeddings by concatenating them.

- No more reliance on a pretrained model.
- Extracts auxiliary labels from phoneme itself.



# Attention vs Duration Based Models

## Attention-based

- No alignments needed
- Adaptable to diverse, noisy datasets
- Capable of more natural prosody

## Duration-based

- Fast parallel inference
- Less chance of alignment problems
- Easier to train
- More robust to silence in training data

# A Compromise: FastSpeech with Soft Attention

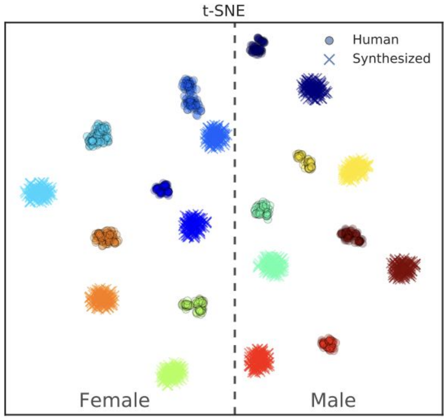
- Add a soft attention module to FastSpeech style TTS
- Compute a softmax across all pairs of text and spectrogram frames
- Use forward sum algorithm to compute the optimal alignment
- Can reuse CTC loss from ASR
- Examples: JETS

# An Alternative: Flow-based Models

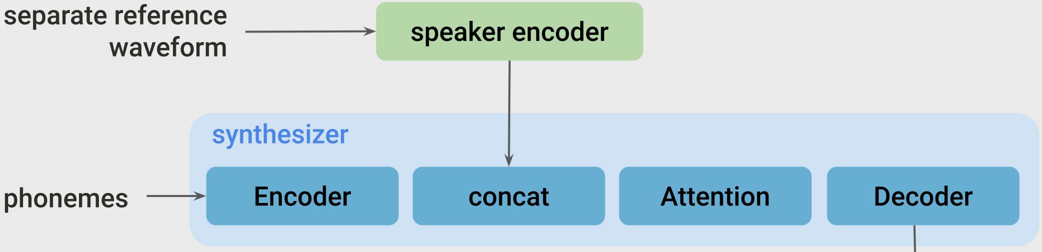
- Flow-based models combine transformer backbones with learned duration/attention
- All the benefits of FastSpeech – fast parallel inference, high quality
- All the benefits of Tacotron – no alignments needed, more flexible
- Examples included Glow-TTS, VITS, NaturalSpeech

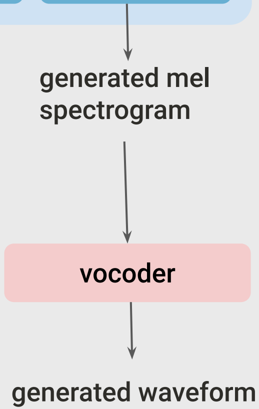
# Speaker and Style Embeddings

# Multispeaker TTS




Low dimensional projections of speaker embeddings




Edit Synthesizer architecture to take as input a "speaker embedding".

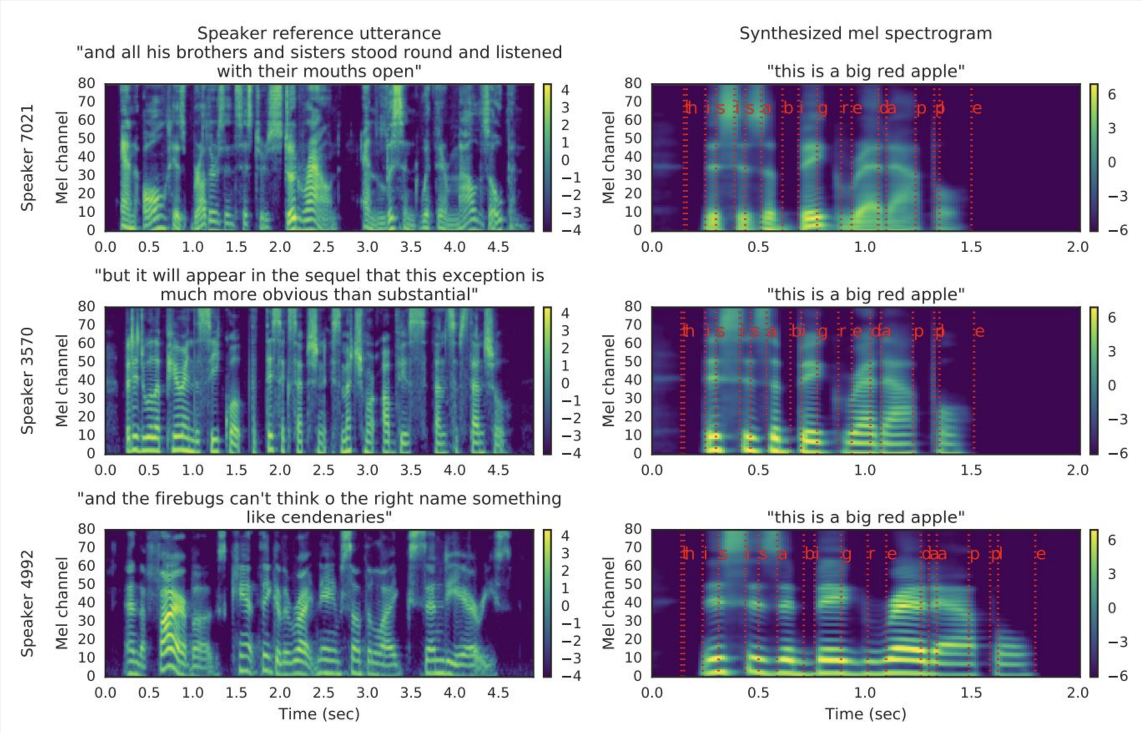
The speaker embedding is trained to maximize cosine similarity of embeddings of utterances from the same speaker.

[Transfer Learning from Speaker Verification to Multispeaker Text-To-Speech Synthesis]

# Multispeaker TTS

**The same sentence produces different spectrograms based on the reference utterance.**

Shape of the generated spectrograms are similar but some are stretched out more, representing slower speakers.

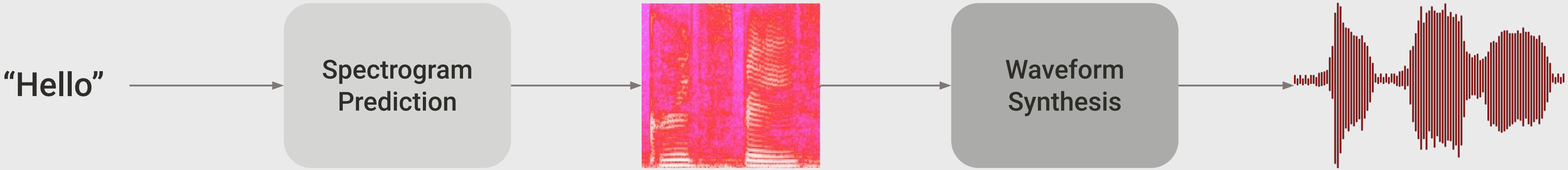



[Transfer Learning from Speaker Verification to Multispeaker Text-To-Speech Synthesis]

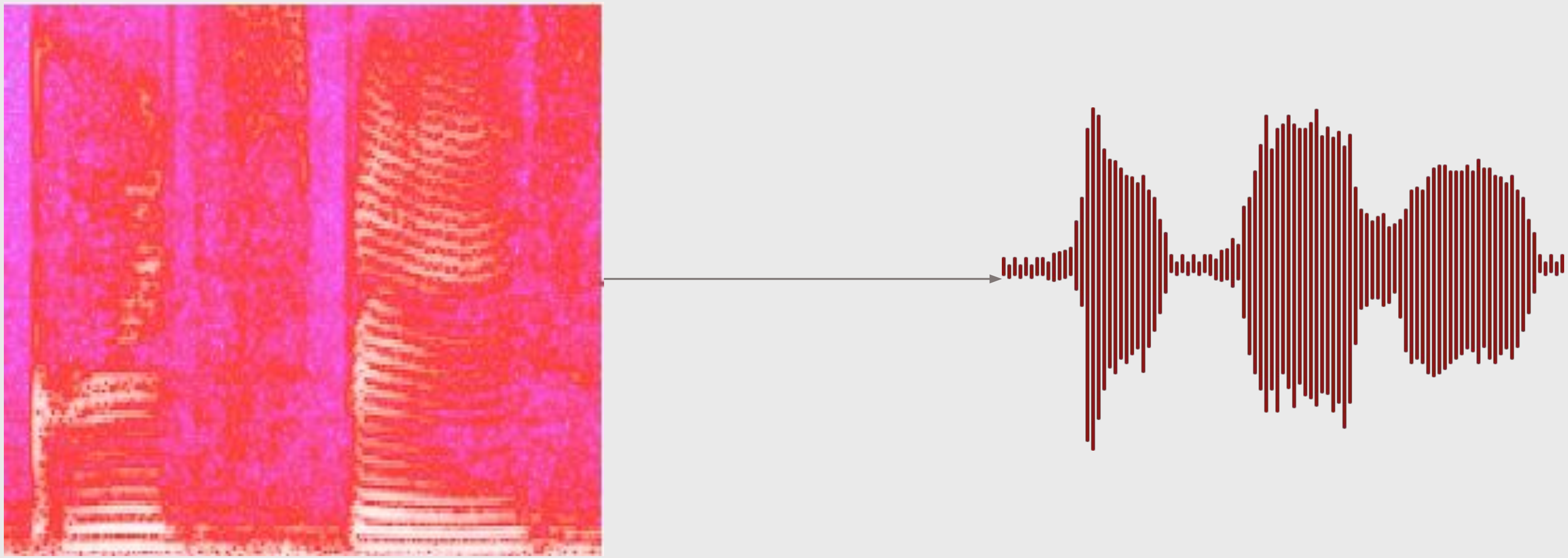
[Speaker samples]

# Vocoders

# Neural TTS Paradigm



# Spectrogram to Waveform Conversion



# Phase Prediction

- We have a magnitude log mel spectrogram from Tacotron/FastSpeech etc.
- We need to fill in the phase to get clear audio, the spectrogram does not represent phase

# Griffin-Lim Wave Reconstruction

- Pure signal processing approach to phase reconstruction
- No learned parameters
- Iteratively reconstructs phase information from just the magnitude spectrogram
- Used in the original Tacotron paper
- How it works:
    - Start with a random prediction for the phase
    - Iteratively apply istft and stft to generate more “consistent” spectrograms
- These sound much clearer than random/zero phase in practice
- Limitations:
    - Since it has no parameters, Griffin-Lim can only provide a coarse reconstruction of the phase
    - Neural models trained on spectrogram/audio pairs are needed for higher quality outputs

# WaveNet

- One of the initial modern deep learning methods for neural net waveform synth
- Generating 16k audio samples per second is a challenge – specialized architectures often used

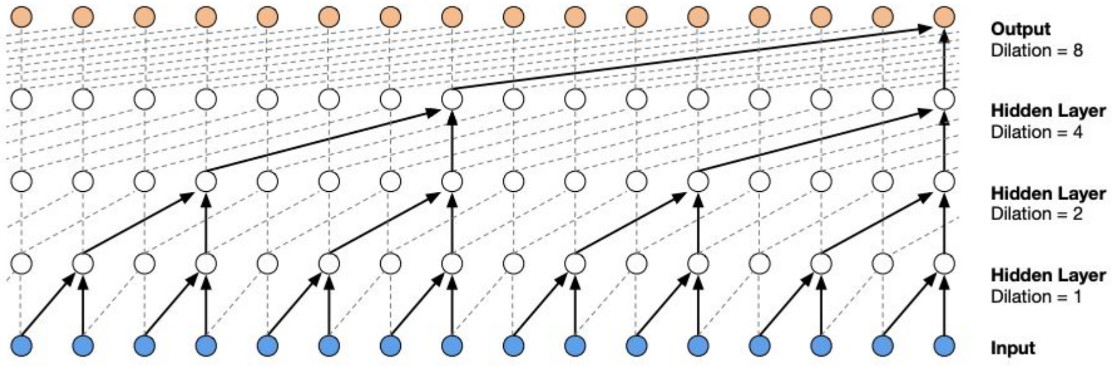
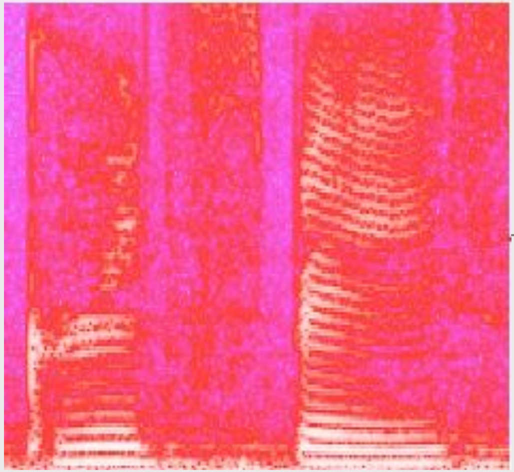



Figure 3: Visualization of a stack of *dilated* causal convolutional layers.

[DeepMind Post]

# WaveNet



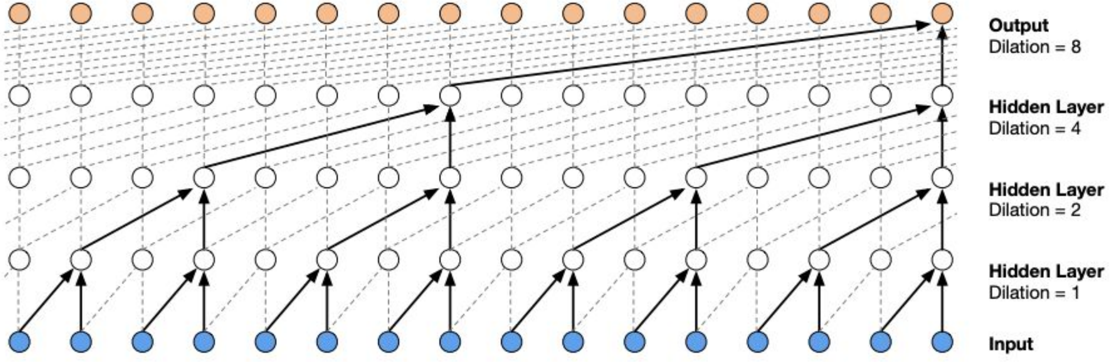



Figure 3: Visualization of a stack of *dilated* causal convolutional layers.

DeepMind Post

# HiFiGAN

- **GAN based vocoders have some of the best quality/latency trade offs currently**
  - Active development in this area as hardware and compute-efficient neural architectures improve
  - Ideally vocoders can run much faster than realtime, and on-device

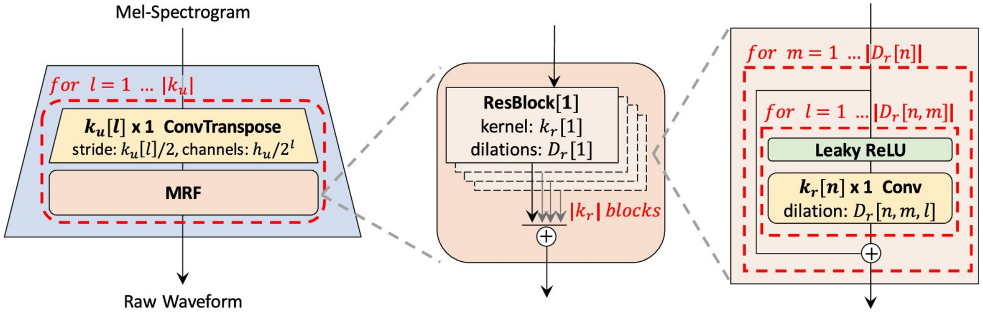



Figure 1: The generator upsamples mel-spectrograms up to $|k_u|$ times to match the temporal resolution of raw waveforms. A MRF module adds features from $|k_r|$ residual blocks of different kernel sizes and dilation rates. Lastly, the $n$-th residual block with kernel size $k_r[n]$ and dilation rates $D_r[n]$ in a MRF module is depicted.

# Conclusion

# Ethical TTS

- Modern TTS is a powerful tool
- People have and will continue to be fooled by great TTS
- Only synthesize someone's voice with permission
- Disclose that your dialog system is a bot

# Appendix

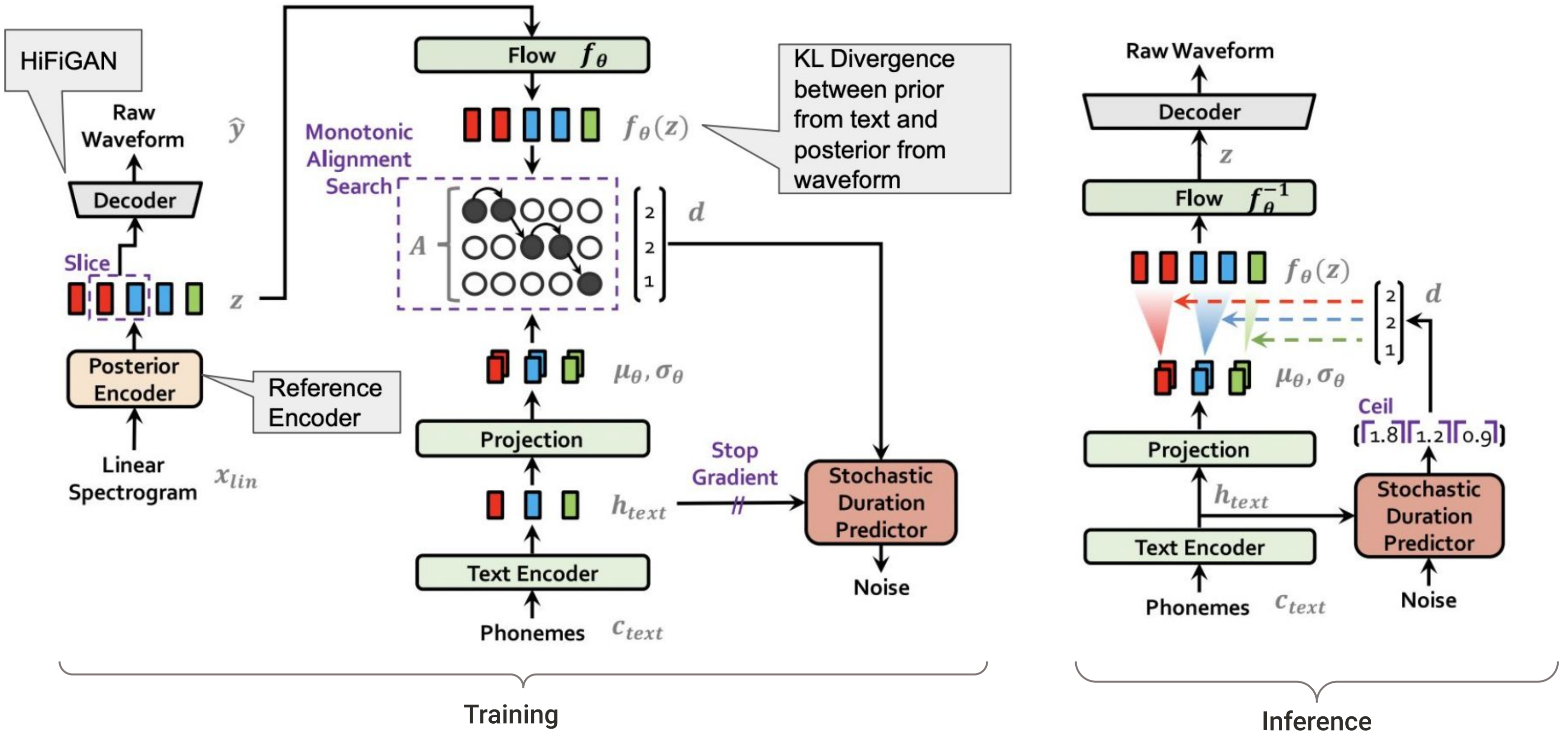
# End-to-end: VITS

# VITS

- Glow-TTS style flow model with monotonic alignment search
- Reference encoder with VAE latent space
- Flow model to produce varied duration modeling
- HifiGAN inspired waveform decoder
- Fully end to end training

Kim et al. 2021

# VITS




Kim et al. 2021

# VITS Loss

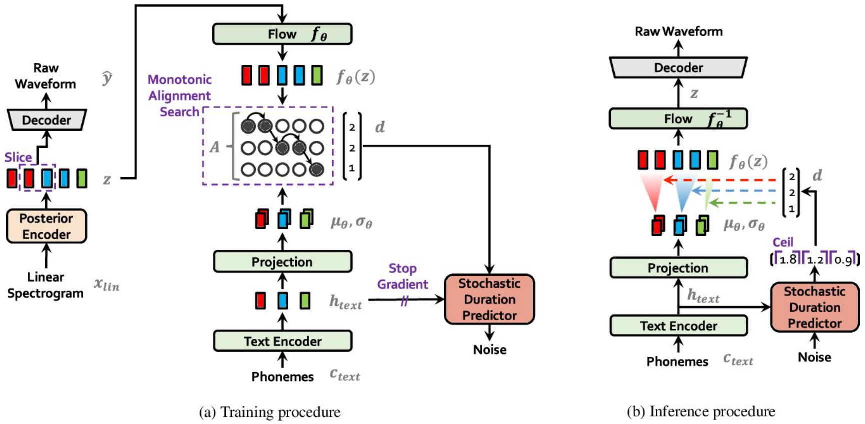
$$L_{vae} = L_{recon} + L_{kl} + L_{dur} + L_{adv}(G) + L_{fm}(G)$$

Kim et al. 2021

# Motivation

Parallel TTS systems are two-staged: (1) first learn to generate spectrograms, and then (2) generate waveforms.

There should be a technique leveraging recent generative techniques to be end-to-end.

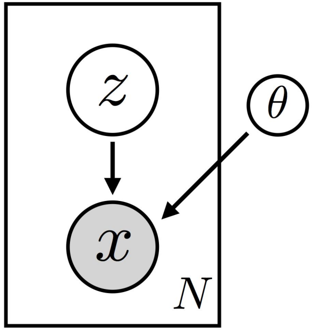


(a) Training procedure


(b) Inference procedure

The VITS model. Next we will discuss each component in this system.

# Background: VAEs




A variational autoencoder is a latent variable model of the form

$$p_\theta(x,z) = p_\theta(x|z)p(z)$$

where x is our observed data (phoneme), and z is a high dimensional latent variable. Pick θ to maximize log $p_\theta(x)$ , called "evidence".

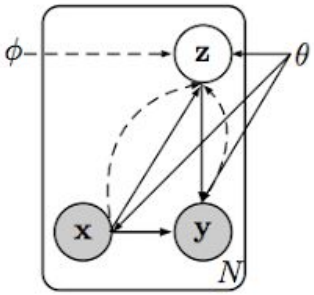
This is hard but we can derive a lower bound. We introduce a inference network $q_\varphi(z|x)$, mapping x to a Gaussian. Then:

$$\begin{aligned} \log p_\theta(x) &= \log \int_z p_\theta(x,z)\, dz = \log \int_z p_\theta(x|z)p(z)\, dz \\ &= \log \int_z p_\theta(x|z)p(z)\, q_\varphi(z|x) / q_\varphi(z|x)\, dz \\ &= \log E_{q(z|x)}[\, p_\theta(x|z)p(z) / q_\varphi(z|x)\, ] \\ &\geq E_{q(z|x)}[\, \log p(x \mid z) + \log p(z) - \log q_\varphi(z|x)\, ] \qquad \text{(Jensen's Inequality)} \\ &= E_{q(z|x)}[\, \log p(x \mid z)\, ] - KL[\, q_\varphi(z|x), p(z)\, ] \end{aligned}$$

This is called the "evidence lower bound" or ELBO. We want to maximize it w.r.t parameters θ and φ. $q_\varphi(z|x)$ and $p_\theta(x,z)$ are parameterized by neural networks.

# Background: CVAEs

We want to condition the latent variable on another variable c




A conditional variational autoencoder looks similar but with an additional variable:

$$p_\theta(x,z|c) = p_\theta(x|z)p(z|c)$$

where $p_\theta(x|z,c) = p_\theta(x|z)$. In this case, the ELBO becomes:

$$\log p_\theta(x|c) \geq E_{q(z|x)}[\log p(x|z)] - KL[q_\varphi(z|x), p(z|c)]$$

Again, we want to maximize it w.r.t parameters θ and φ. $q_\varphi(z|x)$ and $p_\theta(x|z)$ are neural networks.

We may sometimes parameterize $p_\theta(z|c)$ as a third neural network.

# VITS: Model

In the VITS context

$x$ = waveform

$x_{mel}$ = mel spectrogram

$z$ = high dimensional vector

Take the ELBO one term at a time.

1 Reconstruction loss, or $E_{q(z|x)}[\log p(x|z)]$

1. Fix an audio sample x from dataset. Compute $x_{mel}$ from x.
2. Sample $z' \sim q_\varphi(z|x)$. Sample $x' \sim p_\theta(x|z')$. Compute $x'_{mel}$ from x'.
3. Compute $||x_{mel} - x'_{mel}||_1$. This is proportional to $E_{q(z|x)}[\log p(x|z)]$.

# VITS: Model

In the VITS context

$x$ = waveform

$x_{mel}$ = mel spectrogram

$z$ = high dimensional vector

$c = \{ c_{text}, A \}$

$c_{text}$ = phonemes fron text

$A$ = alignment matrix
(shape: $|c_{text}| \times |z|$)

Take the ELBO one term at a time.

1 Reconstruction loss, or $E_{q(z|x)}[\log p(x|z)]$

1. Fix an audio sample $x$ from dataset. Compute $x_{mel}$ from $x$
2. Sample $z' \sim q_\varphi(z|x)$. Sample $x' \sim p_\theta(x|z')$. Compute $x'_{mel}$ from $x'$
3. Compute $||x_{mel} - x'_{mel}||_1$. This is proportional to $E_{q(z|x)}[\log p_\theta(x|z)]$

2 KL divergence, or $E_{q(z|x)}[\log q_\varphi(z|x) - \log p_\theta(z|c)]$

1. Fix an audio sample $x$ from dataset. Fetch context $c$ for $x$.
2. Sample $z' \sim q_\varphi(z|x)$
3. Evaluate $\log q_\varphi(z|x)$ and $\log p_\theta(z|c_{text}, A)$

# VITS: Model

The VITS graphical model:

x = waveform

$x_{mel}$ = mel spectrogram

z = high dimensional vector

c = { $c_{text}$, A }

$c_{text}$ = phonemes fron text

A = alignment matrix
(shape: $|c_{text}|$ x $|z|$)

Take the ELBO one term at a time.

1 Reconstruction loss, or $E_{q(z|x)}[\log p(x|z)]$

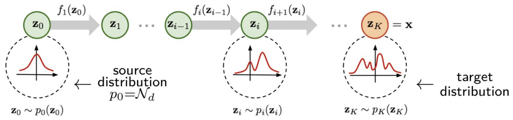
1. Fix an audio sample x from dataset. Compute $x_{mel}$ from x
2. Sample $z' \sim q_\varphi(z|x)$. Sample $x' \sim p_\theta(x|z')$. Compute $x'_{mel}$ from x'
3. Compute $||x_{mel} - x'_{mel}||_1$. This is proportional to $E_{q(z|x)}[\log p_\theta(x|z)]$

2 KL divergence, or $E_{q(z|x)}[\log q_\varphi(z|x) - \log p_\theta(z|c)]$

1. Fix an audio sample x from dataset. Fetch context c for x.
2. Sample $z' \sim q_\varphi(z|x)$. (Use same sample as in Step 1).
3. Evaluate $\log q_\varphi(z|x)$ and $\log p_\theta(z|c_{text}, A)$

3 Add the two.

# Background: Invertible Flows




A method to build a complex distribution from a simple one using <u>invertible functions</u>.

Example of image flow samples

Define an invertible function f. Assume a simple distribution p(z). Given a sample z~p(z). Now, compute z'= f(z).

Normalizing flows guarantee that distribution

$$\log p(z') = \log p(z) - \log | \det \partial f/\partial z |$$

In other words, there's a closed form expression for the resulting distribution! You can then stack multiple flows together.

If $z_K = f_K \ldots f_1(z_0)$ and $z_0 \sim p(z_0)$. Then

$$\log p_K(z_K) = \log q_0(z_0) - \text{sum}_k ( \log | \det \partial f_k/\partial z_{k-1} | )$$

We often parameterize the function $f_\theta$ with a neural network.

# VITS: Priors

In VITS, the prior $p_\theta(z|c)$ is parameterized to be more expressive. This is done using normalizing flows.

Start with a simple distribution $p_0(z|c)$ as a Gaussian distribution

$$N(\mu_\theta(c), \sigma(c))$$

where the mean/stdev are outputs from a neural network. If we want a more expressive distribution, we can define an invertible mapping $f_\theta$ where $f_\theta(z) \sim p_\theta(z|c)$. Then, by rules of normalizing flows

$$p_\theta(f_\theta(z)|c) = p_0(z|c) \mid \det \partial f_\theta(z)/\partial z \mid$$

Using a strong prior turns out to be important for sample quality.

*Table 2.* MOS comparison in the ablation studies.

| Model | MOS (CI) |
|---|---|
| Ground Truth | 4.50 ($\pm$0.06) |
| Baseline | 4.50 ($\pm$0.06) |
| without Normalizing Flow | 2.98 ($\pm$0.08) |
| with Mel-spectrogram | 4.31 ($\pm$0.08) |

# Background: GANs

In image generation, adversarial losses typically improve sample quality.

Example of image GAN samples

Define a discriminator **D** and a generator **G** as two neural networks. The generator maps a latent variable **z** to a waveform **x**. The discriminator classifies an input **x** as real or generated.

The loss function is:

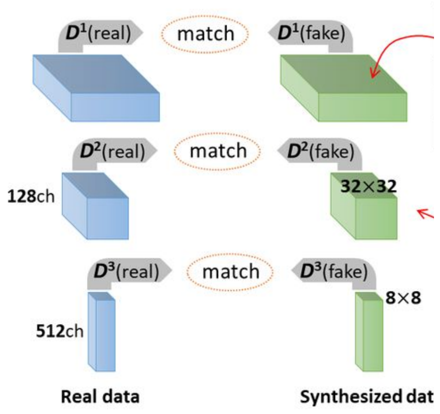
$$\min_\theta \max_\varphi E_x [ \log D_\varphi(x) ] + E_z [ \log(1 - D(G_\theta(z))) ]$$

This is a minimax game.

- A perfect discriminator would separate generated examples from real ones in the dataset.
- A perfect generator would produce samples indistinguishable by the discriminator.

Over time, both push each other to be better. You end up with a powerful generator.

# VITS: Adversarial training




Visualization of feature matching

- Let the decoder $p_\theta(x|z)$ be the generator G.
- Introduce a discriminator D.
- Optimize a variation of the GAN objective

  x = waveform, z = latent

  $$L_{adv} = E_{(x,z)} [ (D_\varphi(x)-1)^2 + (D(G(z)))^2 ] + E_z [ (D(G(z))-1)^2 ]$$

- Add an additional feature matching loss

  $$L_{fm} = E_{(y,z)}[ \mathrm{sum}_l\ 1/N_l \| D^l(x) - D^l(G(z)) \|_1 \ ]$$

T = # of layers in discriminator
$D^l$ = feature map of l-th layer with $N_l$ features
This is like a reconstruction loss for intermediate layers

# VITS: Alignment

In the VITS model, we assume access to an alignment matrix A (shape: $|c_{text}| \times |z|$).

**How do we get this?**

Alignment is between input text and target speech audio. We want to find the best matrix A such that we

$$\max_A \text{ELBO} = \max_A \log p_\theta(x_{mel}|z) + \log p(z|c_{text}, A) - \log q(z|x)$$

$$= \max_A \log p_\theta(z|c_{text}, A)$$

$$= \log N(f_\theta(x) \mid \mu_\theta(c_{text}, A), \sigma_\theta(c_{text}, A))$$

This is a <u>search</u> problem over all possible alignments. We don't have any labels for this so it's generally hard,

- To make this problem simpler, limit candidate alignments to be **monotonic** and **non-skipping**.
- This makes it possible to do dynamic programming to find the best alignment.

[Glow-TTS]

# VITS: Duration Prediction

Use alignment to compute duration prediction.

Add some randomness to make it sound realistic.

**Simple approach**

Given an alignment A, duration for the i-th token $d_i = \text{sum}_j A_{ij}$. Can compute all durations summing across columns in A.

But, this doesn't capture variability of speaking rates. For more realistic rhythm, VITS adds a model to introduce stochasticity.

**VITS approach**

Generative model to output duration $\mathbf{d \sim p_\theta(d|c_{text})}$ from the input text. Use variational dequantization since d is discrete. Optimize a lower bound on $\mathbf{\log p_\theta(d|c_{text})}$

flow++

# VITS: Objective

VITS objective= (1) VAE reconstruction loss +
(2) VAE KL divergence +
(3) Duration prediction loss +
(4) Adversarial loss +
(5) Feature matching loss

## VITS Architectures

- WaveNet residual blocks for encoder $q_\varphi(z|x)$
- $c_{text}$ is ingested using a hidden layer from a transformer
- normalizing flow $f_\theta$ is a stack of affine coupling layers where the Jacobian determinant is 1.
- Decoder $p_\theta(x|z)$ is a HiFi-GAN. The discriminator is also the one used in HiFi-GAN.
- The duration predictor is a network of stacked residual blocks and convolutional layers.

# VITS: Summary

- Combines the best features of flows, VAEs and GANs
- End to end training
- No alignments required
- Controllable prosody through VAE
- Fast parallel inference: 67 RTF on 1 V100
- Very high MOS scores

# NaturalSpeech

- Similar structure to VITS with prior/posterior flow model
- Adds phoneme pretraining, differential duration modeling and a memory VAE
- Matches MOS of human speaker on LJSpeech dataset
- Samples

Tan et al. 2022

# Recent Methods

# Tortoise

Better speech synthesis through scaling

# VALL-E / VALL-E-X

Neural Codec Language Models are Zero-Shot Text to Speech Synthesizers

# Bark

https://github.com/suno-ai/bark

# More Examples of Neural Networks on Raw Audio

- Generative model of audio
- Autoregressive: generates one sample of audio at a time
- Many layers of dilated convolutions for a high receptive field
- Very high output quality
- Extremely Slow
- Can be conditioned on linguistic features or spectrograms to generate speech for specific utterances

Oord et al. 2016

# WaveRNN

- Hyper-optimize a simple, autoregressive GRU model instead of WaveNet
- Up to 96% (!) weight sparsification and subsampling
- Runs ~4x real time even on smartphone CPUs
- Diverse applications in audio (see LPCNet, Lyra codec / WaveNetEQ packet loss smoothing)

Kalchbrenner et al. 2018

# WaveRNN

- Hyper-optimize a simple, autoregressive GRU model instead of WaveNet
- Up to 96% (!) weight sparsification and subsampling
- Runs ~4x real time even on smartphone CPUs
- Diverse applications in audio (see LPCNet, Lyra codec / WaveNetEQ packet loss smoothing)

Kalchbrenner et al. 2018

# Parallelizing WaveNet

WaveNet

Slow
Autoregressive
High Quality

→

???

Fast
Parallel
High Quality

# Parallelizing WaveNet

WaveNet

Slow
Autoregressive
High Quality

→

???

Sequential
Training Parallel
Inference

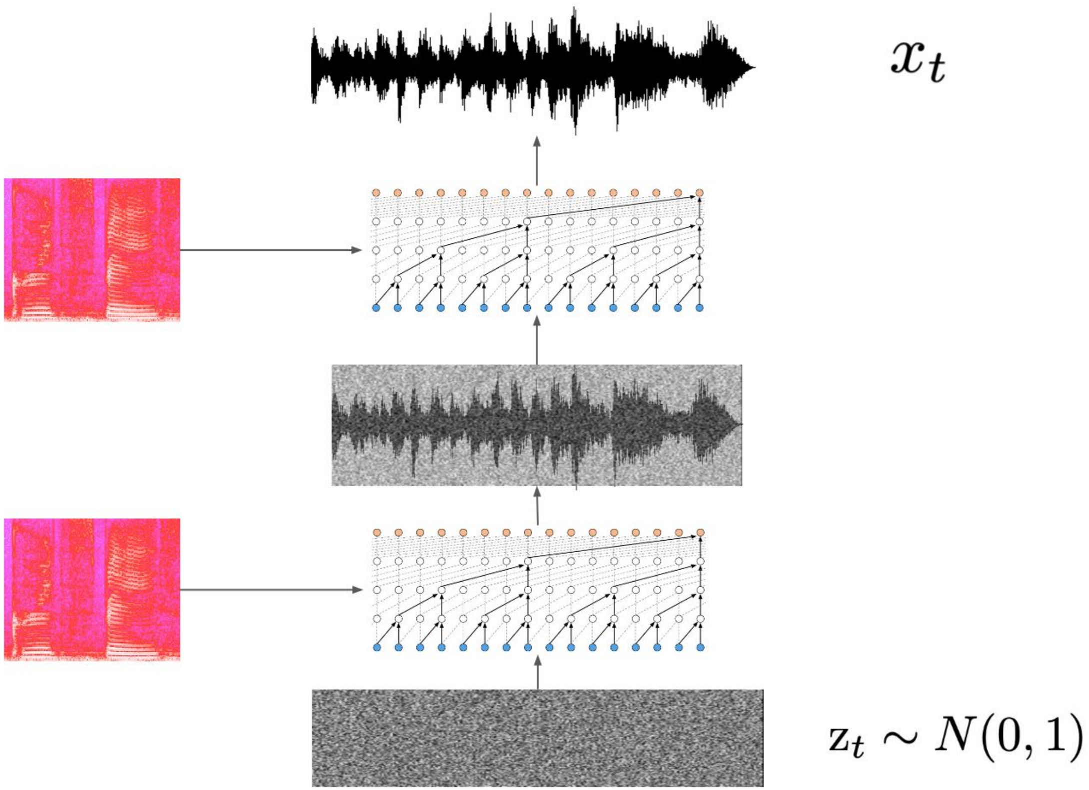
# Inverse Autoregressive Flows

- Sample the number of audio samples we want to generate from a unit gaussian distribution
- Transform those samples by a mean and variance predicted by a neural net
- This produces the full waveform in parallel
- Each step is as follows:

$$x_t = z_t \cdot s(\boldsymbol{z}_{<t}, \boldsymbol{\theta}) + \mu(\boldsymbol{z}_{<t}, \boldsymbol{\theta})$$

where s and μ are produced by running a WaveNet on z

# Inverse Autoregressive Flows



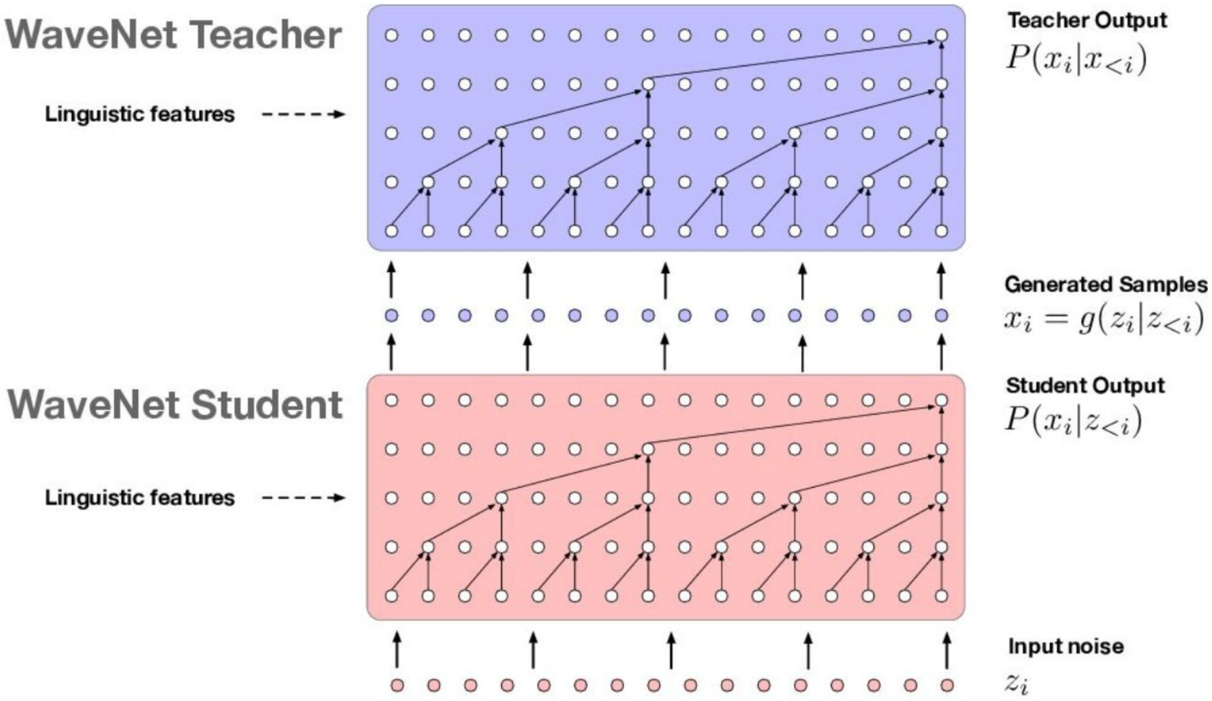
# Inverse Autoregressive Flows

- Fast, parallel sampling
- Closed form for gradient update requires an autoregressive calculation
- This makes directly training the flow intractable
- In a sense, the inverse of WaveNet

# Parallel WaveNet: Student and Teacher

- Use a trained normal WaveNet model as a “teacher” for an IAF
- Minimize the KL divergence between the output distribution of the IAF and teacher wavenet
- This can be done in parallel, so training is fast
- Once trained, the student IAF can then perform inference in parallel on its own

# Parallel WaveNet: Student and Teacher




[Oord et al. 2016]

# Parallel WaveNet Issues

- Have to train two separate models
- Even with Clarinet, training the student distribution to match the teacher is extremely finicky
- Perceptual losses required which are hand tuned
- In practice, very hard to replicate the quality of the original WaveNet

# Parallelizing WaveNet

WaveNet

Slow
Autoregressive
High Quality

→

IAF

Parallel Training
Inference

# Parallelizing WaveNet

WaveNet

Slow
Autoregressive
High Quality

→

IAF

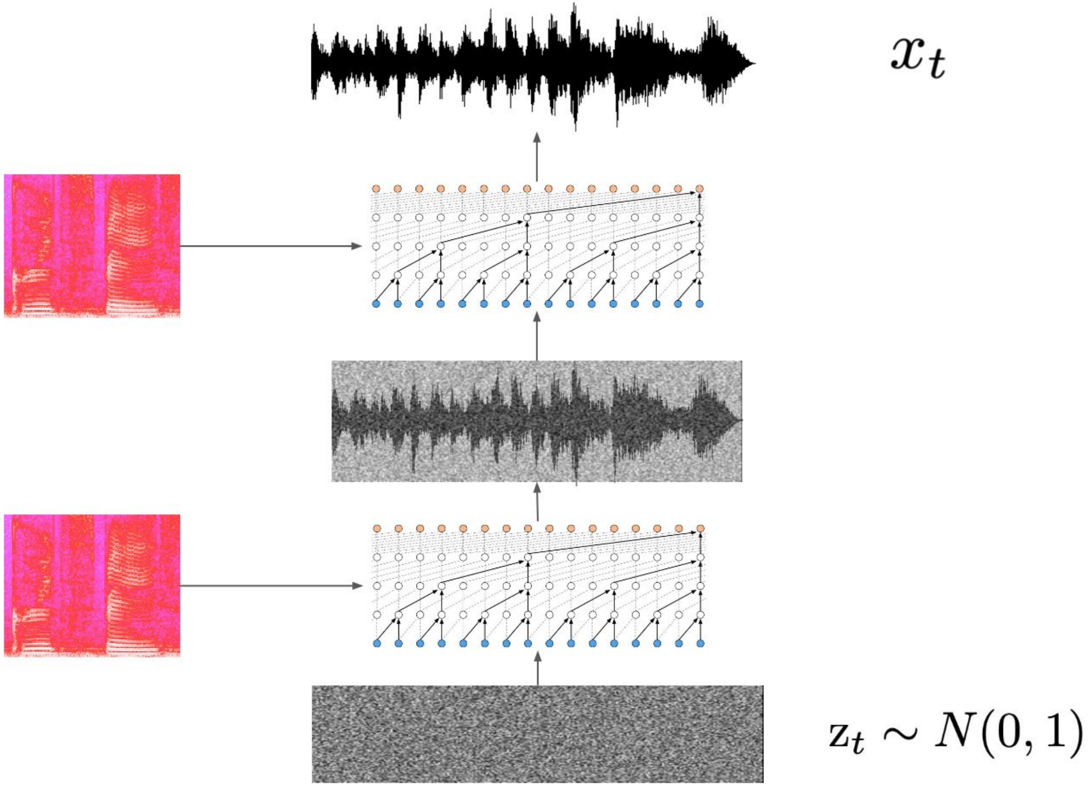
Parallel Training
Inference

→

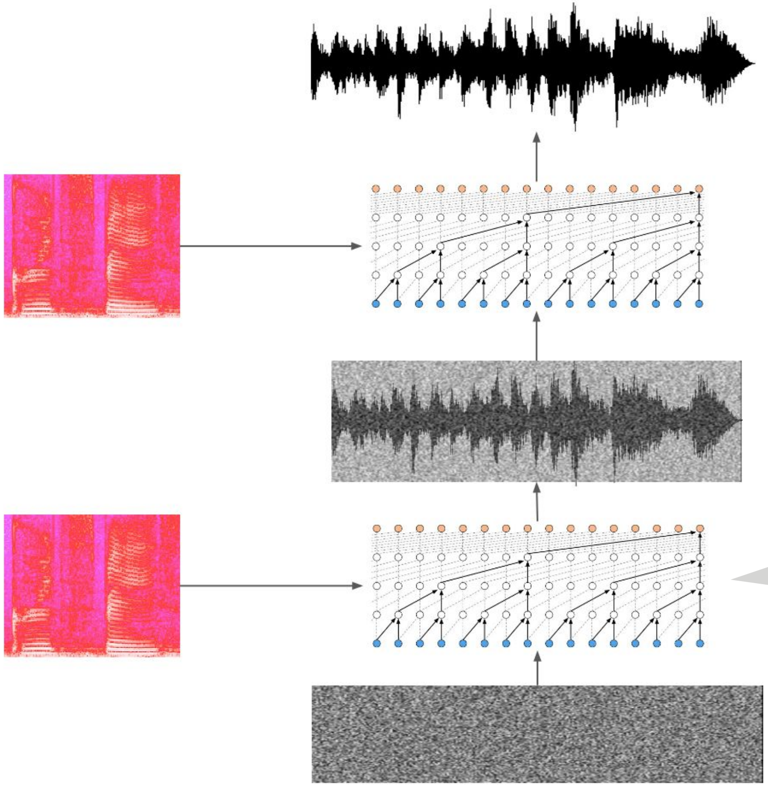
???

Parallel Training
Parallel Inference

# Inverse Autoregressive Flows



# Inverse Autoregressive Flows



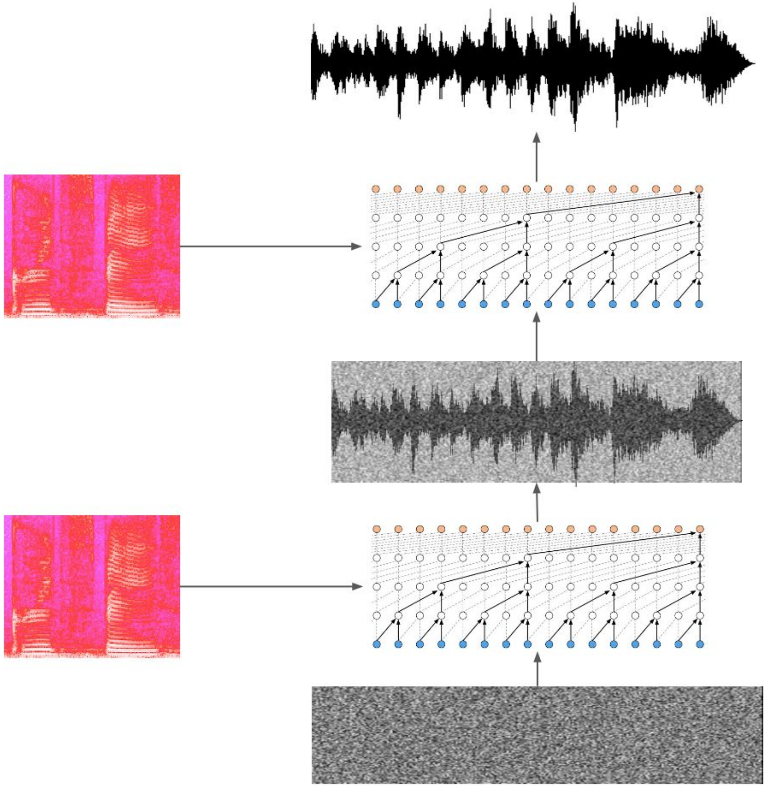
Autoregressive WaveNet => intractable log likelihood

Flow Training ⇔ WaveNet Inference => SLOW

$$z_t \sim N(0, 1)$$

# Inverse Autoregressive Flows

What if this was invertible?

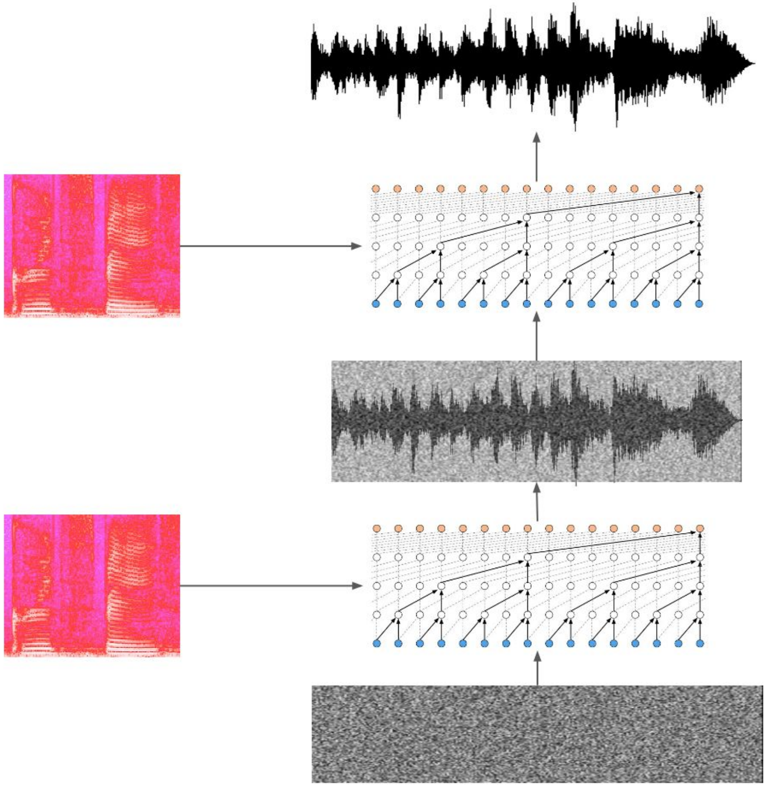


$x_t$

$z_t \sim N(0, 1)$

# Inverse Autoregressive Flows

What if this was invertible?



$x_t$

Inference:
sample z and
transform to x

Training: transform x to z
and enforce a normal
distribution on z

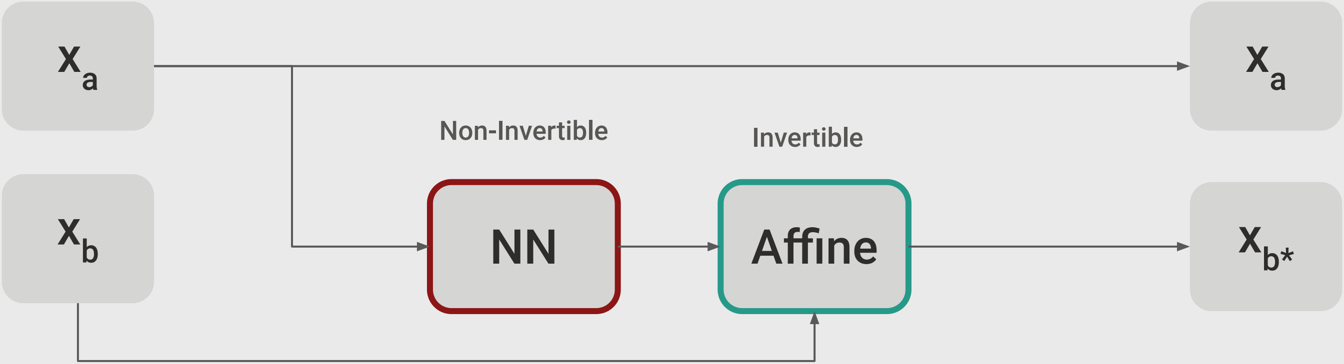
$z_t \sim N(0, 1)$

# Glow

- Invertible flow based model
- Originally applied to image generation by OpenAI
- Quickly repurposed for audio generation with WaveGlow

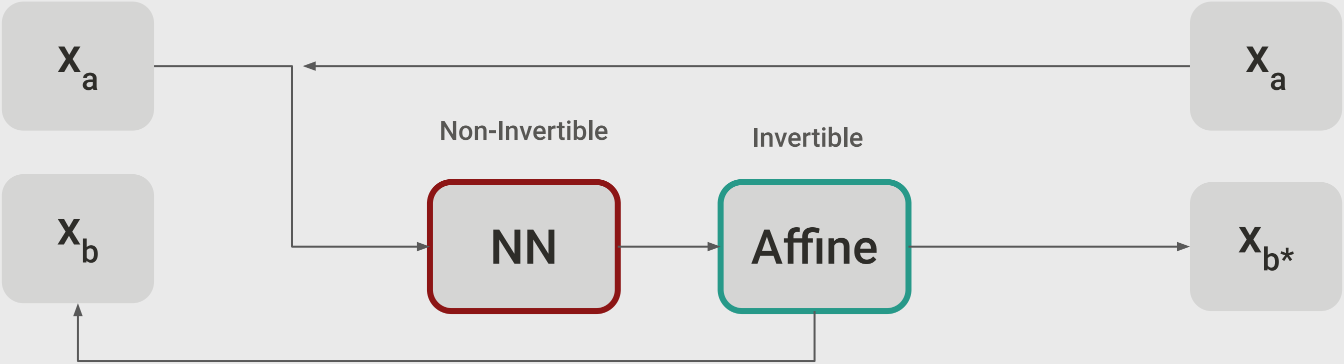
Kingma, Dhariwal 2018, Prenger et al. 2018

# Affine Coupling Layer - Forward




In the forward pass, $x_a$ is unchanged and used to transform $x_b$ into $x_{b*}$

# Affine Coupling Layer - Backward




In the backwards pass, WN produces the same scale and bias for the affine transformation since $x_a$ is the same. This means we can just invert the affine transformation to transform $x_{b*}$ to $x_b$

# Affine Coupling Layers

$$\boldsymbol{x}_a, \boldsymbol{x}_b = split(\boldsymbol{x})$$

$$(\log \boldsymbol{s}, \boldsymbol{t}) = WN(\boldsymbol{x}_a, mel\text{-}spectrogram)$$

$$\boldsymbol{x}_b\prime = \boldsymbol{s} \odot \boldsymbol{x}_b + \boldsymbol{t}$$

$$\boldsymbol{f}^{-1}_{coupling}(\boldsymbol{x}) = concat(\boldsymbol{x}_a, \boldsymbol{x}_b\prime)$$

Prenger et al. 2018

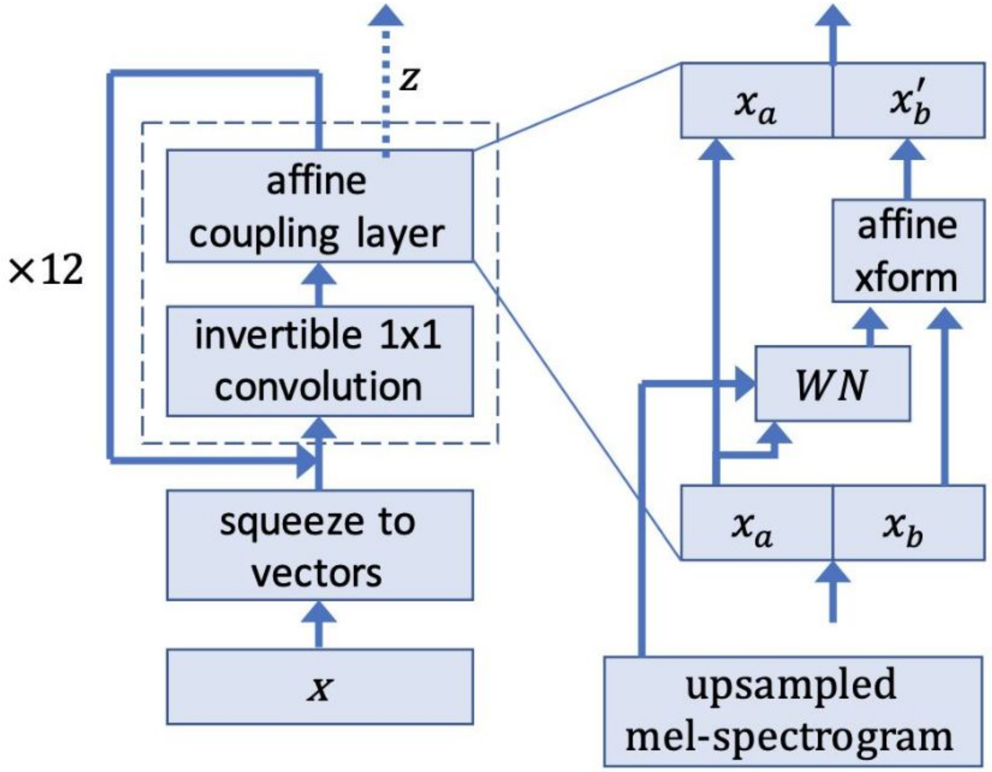
# Mixing Channels

- Affine Coupling Layers can only transform half the input at a time
- Need a way to mix the channels between coupling layers

# Invertible 1x1 Convolution

- 1x1 Convolution with a square kernel
- Initialize the kernel to be an invertible, orthonormal matrix
- Add a term to the loss to ensure it stays invertible in training
- For the backwards pass we just invert the kernel
- Now the channels are mixed between coupling layers

# WaveGlow Architecture




[Prenger et al. 2018](#)

# WaveGlow Loss Function

$$\log p_\theta(\boldsymbol{x}) = -\frac{\boldsymbol{z}(\boldsymbol{x})^T \boldsymbol{z}(\boldsymbol{x})}{2\sigma^2}$$

Fit z to a unit Gaussian Distribution

$$+ \sum_{j=0}^{\#coupling} \log \boldsymbol{s}_j(\boldsymbol{x}, \mathit{mel\text{-}spectrogram})$$

Change of variables from coupling

$$+ \sum_{k=0}^{\#conv} \log \det |\boldsymbol{W}_k|$$

Ensure 1x1 conv kernels remain invertible

Prenger et al. 2018

# WaveGlow

- Directly maximising likelihood makes training much more stable
- Eliminates the needs for perceptual losses
- Only have to train one model
- Quality equal to WaveNet
- Synthesize audio in parallel

# Parallelizing WaveNet

# Can we Go faster?

- WaveGlow requires a powerful GPU for fast inference
- WaveRNN requires heavy optimization to run real time on CPUs
- Is there an alternative?

# GAN-based Vocoders

- Generative adversarial networks applied to audio generation
- Simultaneously train two networks: a generator and a discriminator
- Generator produces audio from the spectrograms to be as close as possible to the ground truth audio
- Discriminator trained to distinguish generator outputs from real audio
- Examples include MelGAN, Parallel WaveGAN, HiFiGAN

# LSGAN Architecture



Discriminator Loss

$$\mathcal{L}_{Adv}(D;G) = \mathbb{E}_{(x,s)}\left[(D(x)-1)^2 + (D(G(s)))^2\right]$$

Generator Loss

$$\mathcal{L}_{Adv}(G;D) = \mathbb{E}_{s}\left[(D(G(s))-1)^2\right]$$

Mao et al. 2016

# Additional Losses for Audio GANs

- Direct reconstruction loss on mel spectrograms

$$\mathcal{L}_{Mel}(G) = \mathbb{E}_{(x,s)}\left[ ||\phi(x) - \phi(G(s))||_1 \right]$$

- Discriminator feature map L1 loss

$$\mathcal{L}_{FM}(G; D) = \mathbb{E}_{(x,s)}\left[ \sum_{i=1}^{T} \frac{1}{N_i} ||D^i(x) - D^i(G(s))||_1 \right]$$

Kumar et al 2019, Kong et al. 2020

# Multi-scale/multi-period Discriminators

- Multiple discriminators at different scales/periods are helpful
- Capture long term dependencies

Kumar et al 2019, Kong et al. 2020

# GANs

- Very fast parallel GPU and CPU synthesis
- Quality approaching or matching WaveNet/WaveGlow/WaveRNN
- Require carefully designed additional losses to perform well
- Good open source implementations

# Summary

- GAN based vocoders have the best quality/latency trade offs currently
- HiFiGAN is a great choice – high performance and high quality

# End to End Glow TTS Model

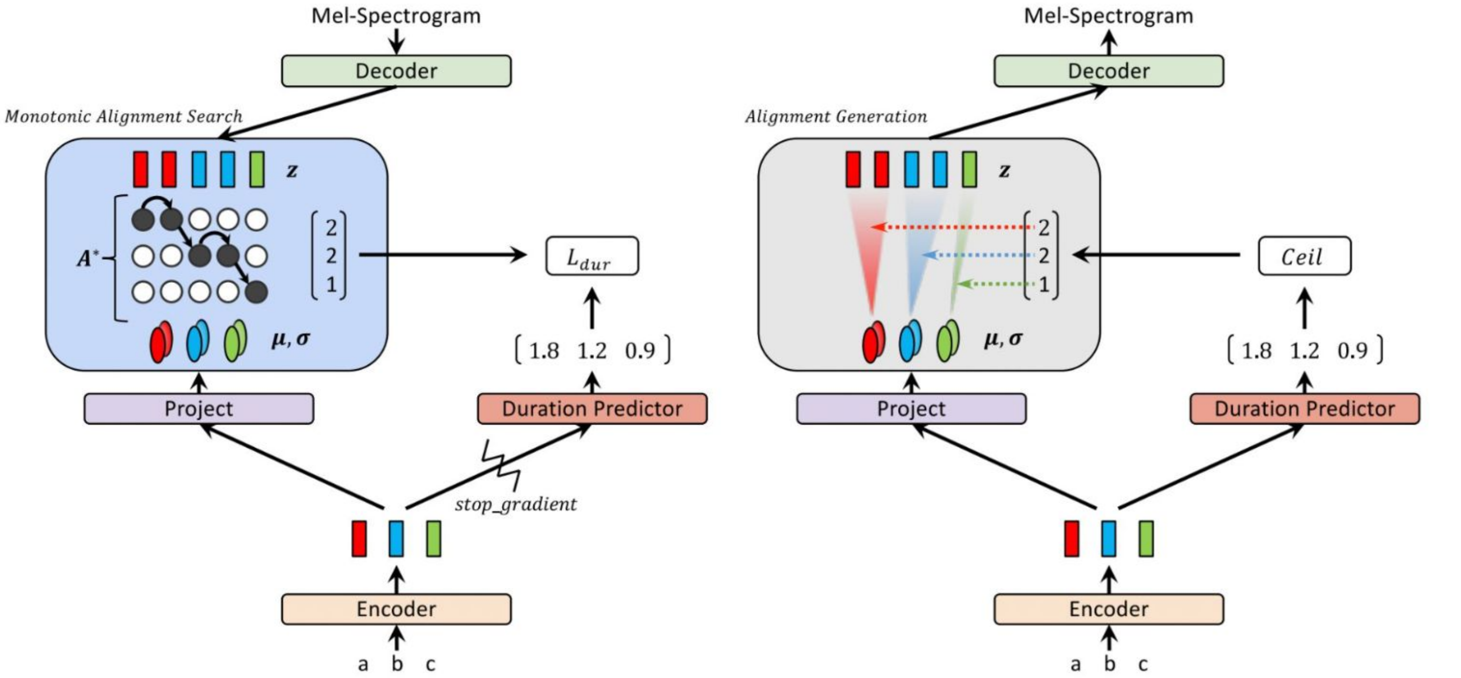
# The Ideal TTS Model

- Expressive, flexible duration modeling like Tacotron
- Fast parallel inference like FastSpeech
- Reference encoder to account for one-to-many mapping
- Trained end to end – no separate vocoder

# Glow-TTS

- Use a flow model for posterior from mel spectrograms to text
- Use a transformer text encoder to parametrize the prior
- Train using maximum likelihood to match prior and posterior distributions
- Since we have maximum likelihood, use dynamic programming to find the most likely alignment during training
- For inference, train a separate duration predictor to match the most likely alignment

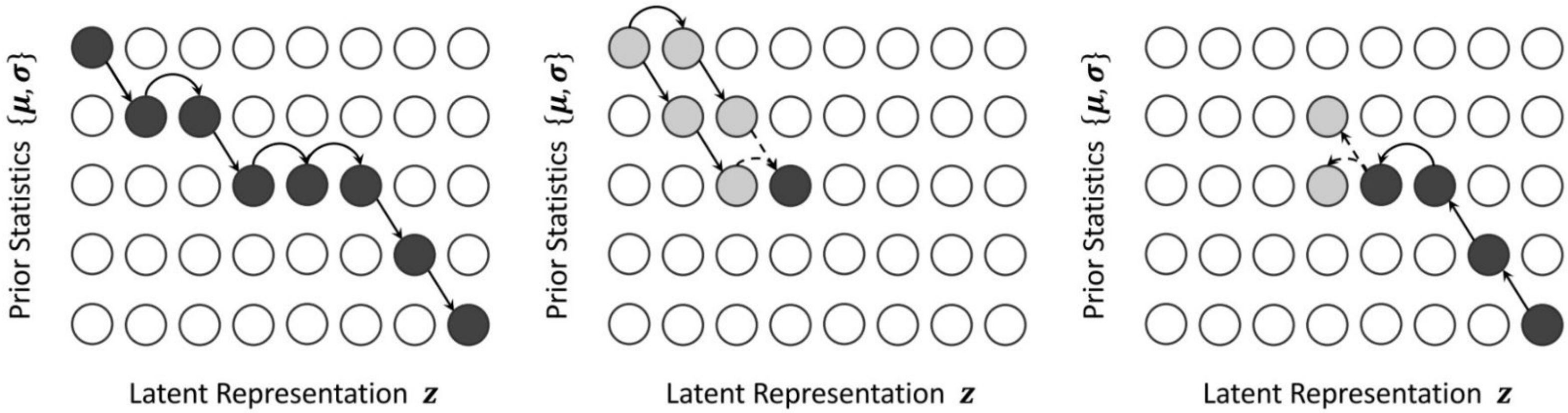
# Glow-TTS




(a) An abstract diagram of the training procedure. (b) An abstract diagram of the inference procedure.

Kim et al. 2020

# Glow-TTS – Monotonic Alignment Search




(a) An example of monotonic alignments

(b) Calculating the maximum log-likelihood $Q$.

(c) Backtracking the most probable alignment $A^*$.

**Figure:** Illustration of monotonic alignment.

[Kim et al. 2020]

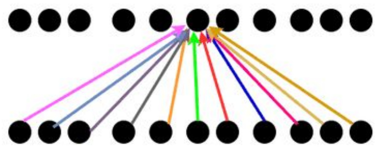
# Glow-TTS Shortcomings

- Not trained end to end – still uses a mel spectrogram output
- No reference encoder – less prosodic variation/controllability
- Direct duration prediction – less natural prosody

# TacoTron Attention Variations

# Content Based Attention

Global attention



RNN_att Query

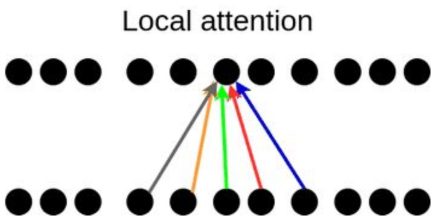
Encoder Keys

$$e_{i,j} = v^T tanh(W s_i + V h_j)$$

$$\alpha_i = softmax(e_i)$$

Bahdanau et al. 2014

# Location Sensitive Attention




Convolution with previous alignment

$$f_{i,j} = F * \alpha_{i-1}$$

$$e_{i,j} = v^T tanh(W s_i + V h_j + U f_{i,j})$$

$$\alpha_i = softmax(e_i)$$

Shen et al. 2018

# Location Sensitive Attention

- Allows the model to explicitly use previous alignments for computing the next attention state
- Achieves much stronger alignments in practice than plain Bahdanau attention
- Enough model flexibility to learn a high quality text to spectrogram mapping

# Dynamic Convolutional Attention

$$f_{i,j} = F * \alpha_{i-1}$$

$$G(s_i) = v_g^T tanh(W_g s_i + b_g)$$

$$g_{i,j} = G(s_i) * \alpha_{i-1}$$

$$p_i = \log(P * \alpha_{i-1})$$

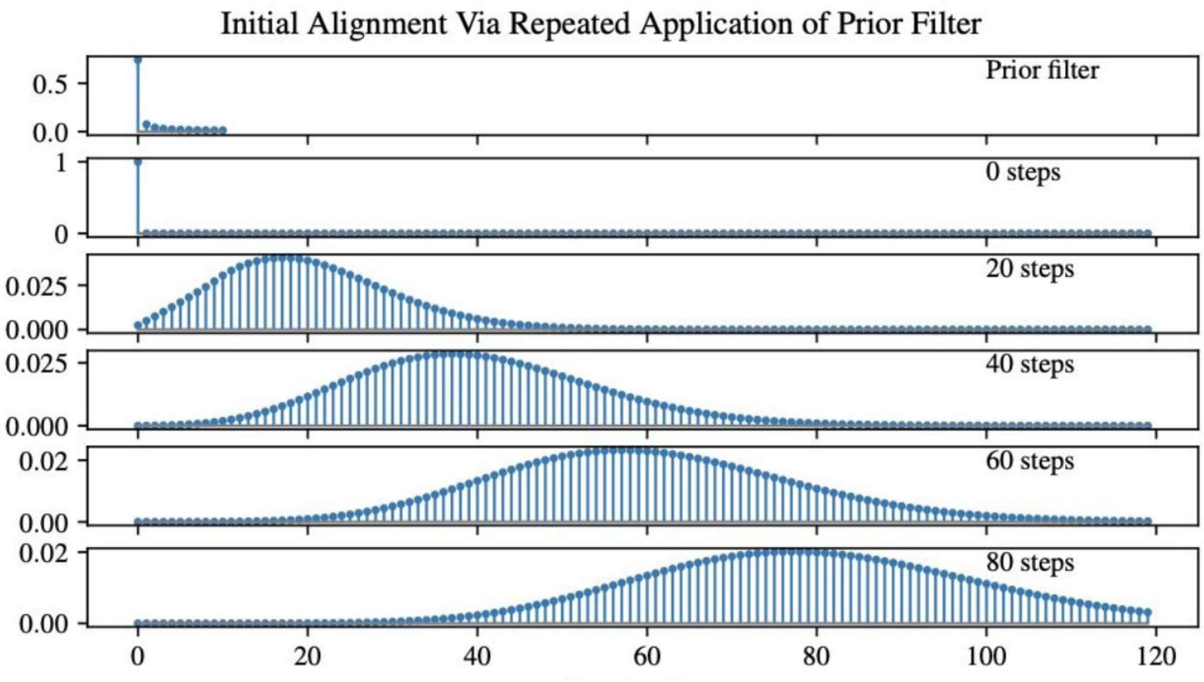
$$e_{i,j} = v^T tanh(U f_{i,j} + T g_{i,j}) + p_{i,j}$$

Dynamic filters computed from attention state

Prior bias to encourage monotonicity

Battenberg et al. 2019

# Dynamic Convolutional Attention




$$p_i = \log(P * \alpha_{i-1})$$

Battenberg et al. 2019

# Dynamic Convolutional Attention

- Dynamic filters on previous alignment instead of directly using the encoder outputs and query
- Add a prior bias which softly encourages monotonicity
- Learns even more consistent alignments than location sensitive attention
- Better generalization to long utterances
- Tends to reach an alignment faster

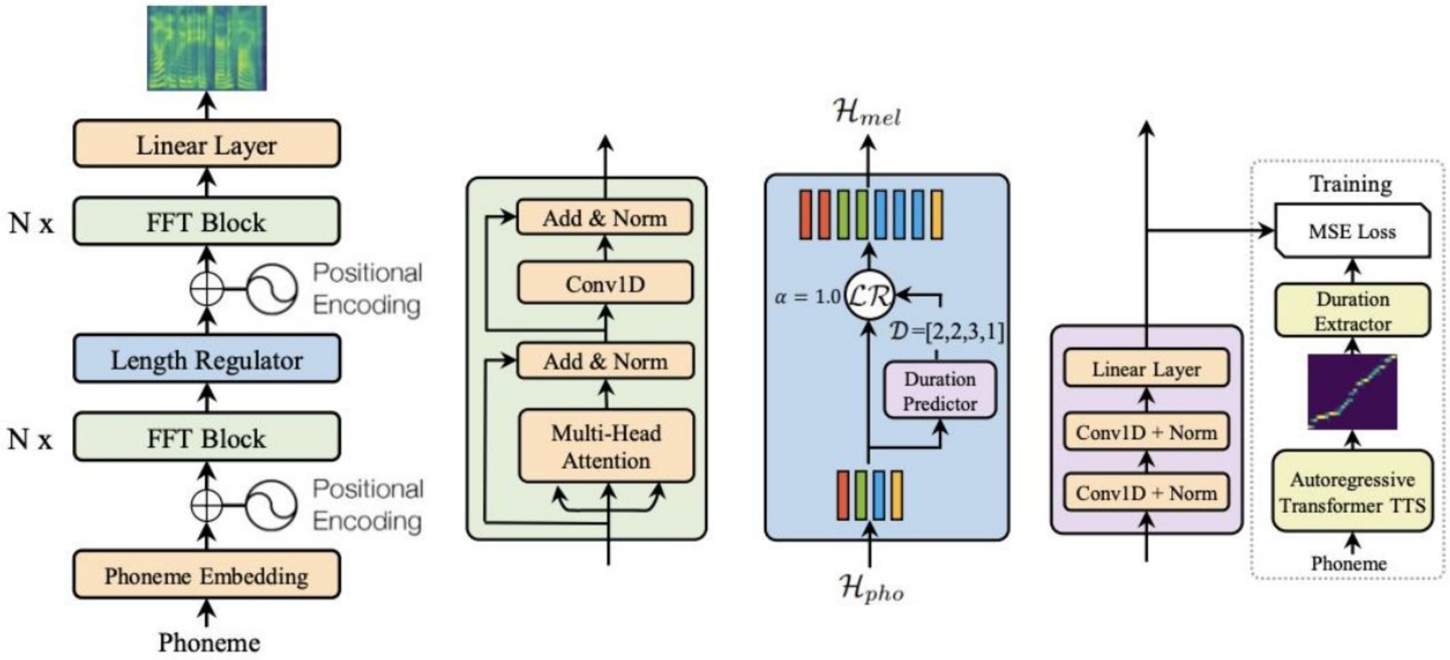
# Tips for Training Attention TTS Models

- Alignments are everything, a good alignment in training almost certainly means good generalization
- Make sure your examples are well trimmed, consider normalizing volume and removing especially noisy samples
- Use a location based attention. LSA is simple and works well. DCA/GMM can be even better
- Make sure your log mel spectrograms are well normalized
- Fine tuning from existing models can be useful for small/noisy datasets
- Reduction factor is your friend if you're struggling to get an alignment

# Attention Model Drawbacks

- Autoregressive => Slow
- Occasionally prone to skipping, repeating etc even with LSA, DCA

# An Alternative: Explicit Duration Modelling




Ren et al 2019

# FastSpeech 1/2

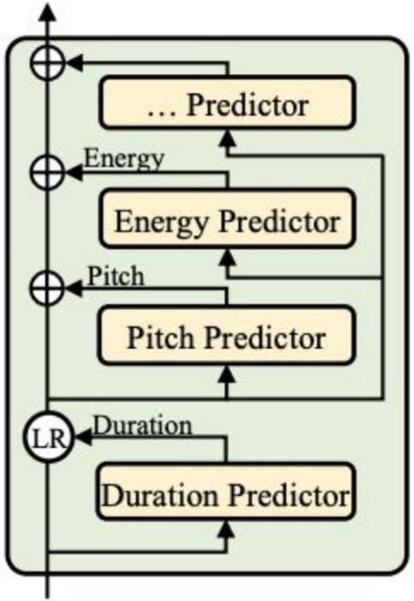
- Similar to earlier DNN TTS systems
- Explicitly predict phoneme durations, f0 and pitch
- Durations for training come from an autoregressive model (e.g. tacotron) or from traditional HMM forced alignments
- To match the input and output lengths, repeat input states according to phoneme durations
- Use a transformer to predict in parallel rather than frame by frame

Ren et al 2019, Ren et al. 2020

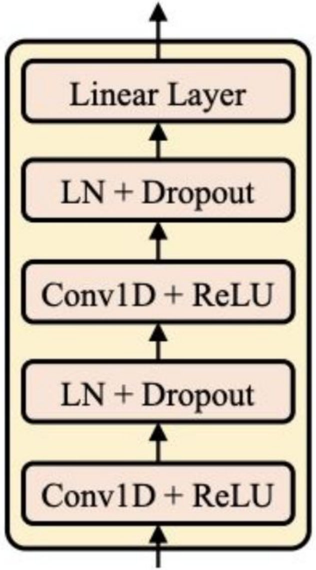
# FastSpeech 2 Variance Predictors

At training time use the ground truth duration, energy, f0 and pitch for synthesis and train predictors with MSE

(FFT = Feed Forward Transformer not Fast Fourier Transform)




Variance Predictor Structure:




Ren et al. 2020

# Generative Spoken Language Modeling

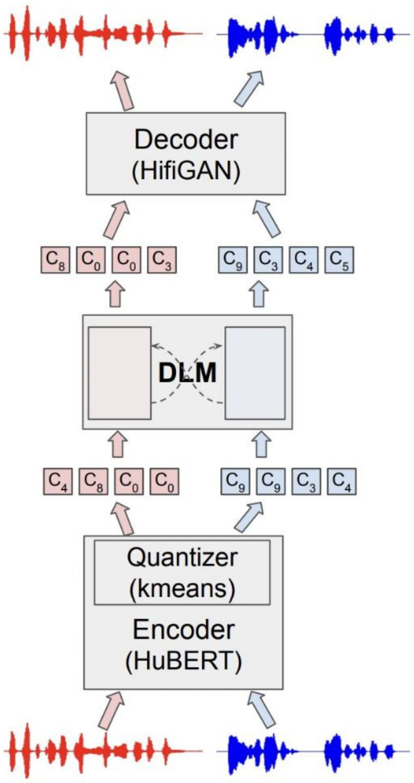
# The Future of TTS

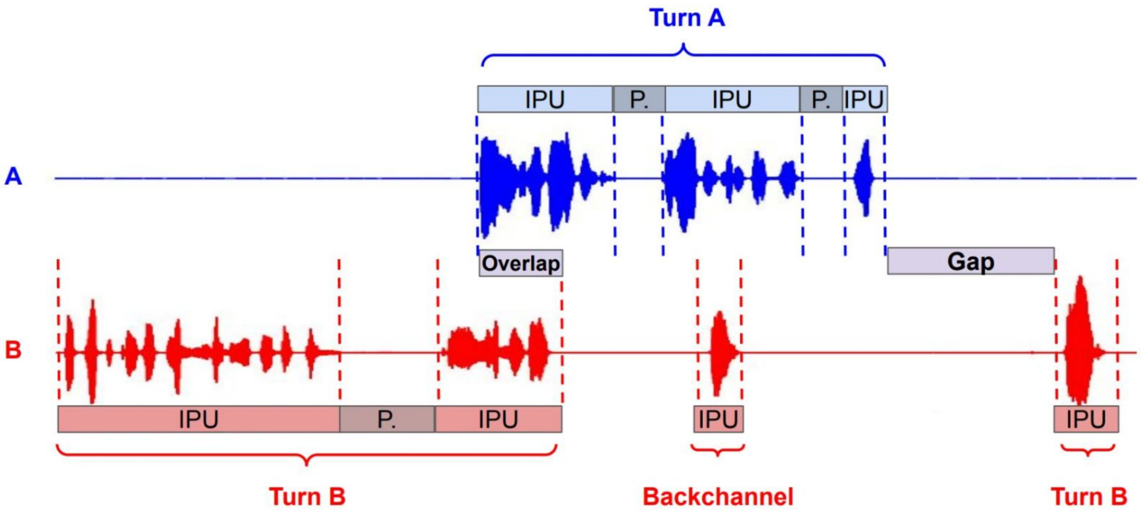
- On high quality datasets, TTS has reached parity with humans in MOS
- These systems operate on a single utterance at a time
- Systems that handle long form context and dynamically adjust their tone are the future

# Generative Spoken Language Modeling

- Obtain discrete audio codes from Wav2Vec-2, HUBERT etc.
- Train a GPT style transformer LM on the codes
- Train a speech synthesis model to convert codes to speech
- Can simulate turn-taking and backchannels when training on two channels
- Samples

# Generative Spoken Language Modeling




Nguyen et al 2022

# More on style and speaker embeddings

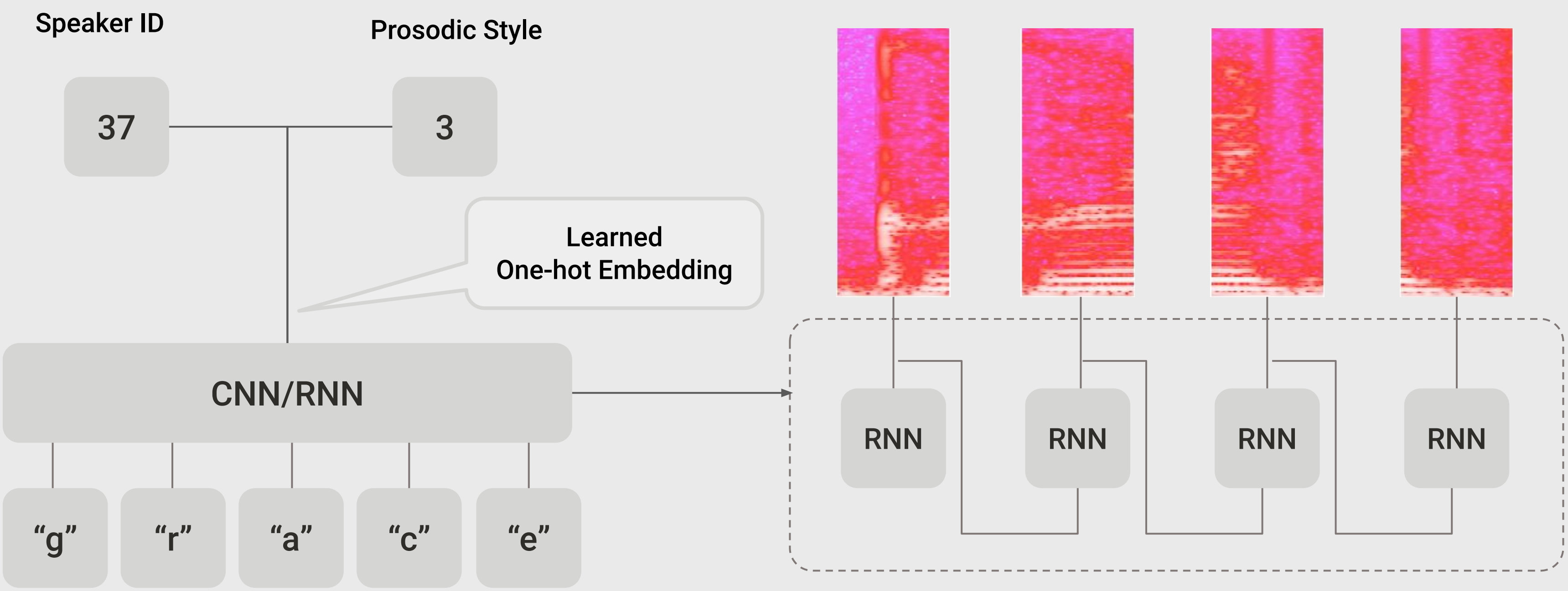
# Expanding the “text” in TTS

- TTS is fundamentally a one-to-many mapping
- The same text has infinitely many voicings
- Controllable speaker and prosody is very useful in dialog systems and elsewhere

# Speaker/Style with One Hot Labels

- Enumerate your speakers and/or styles and label the training data with them
- During training, learn an embedding for each speaker/style by passing a one hot encoding to the encoder
- At inference, pass in the corresponding speaker/style embedding
- Simple and easy to train but constrained by the breadth of your labels

# Sequence to Sequence Problem

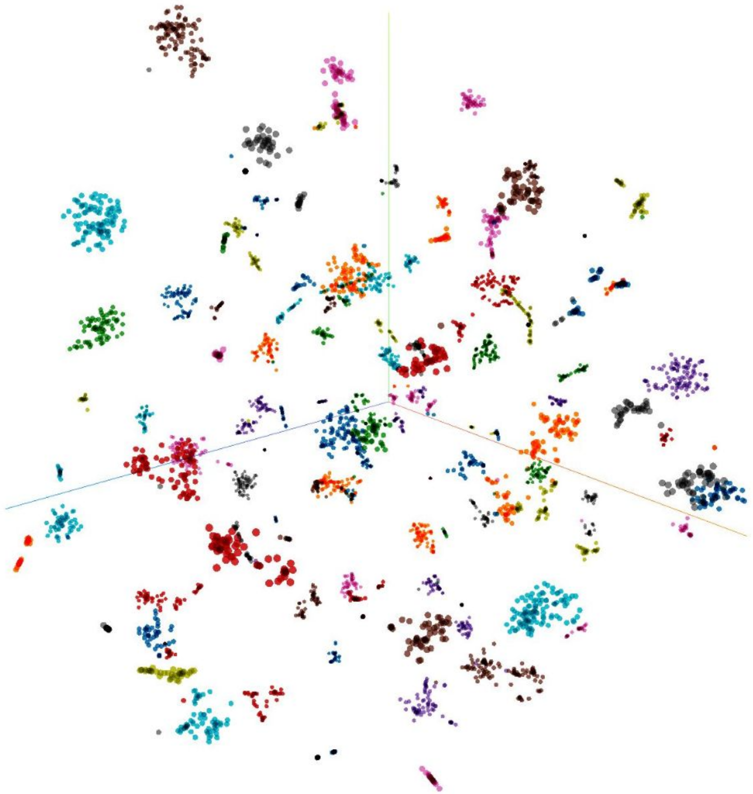


# Learned Speaker Embeddings

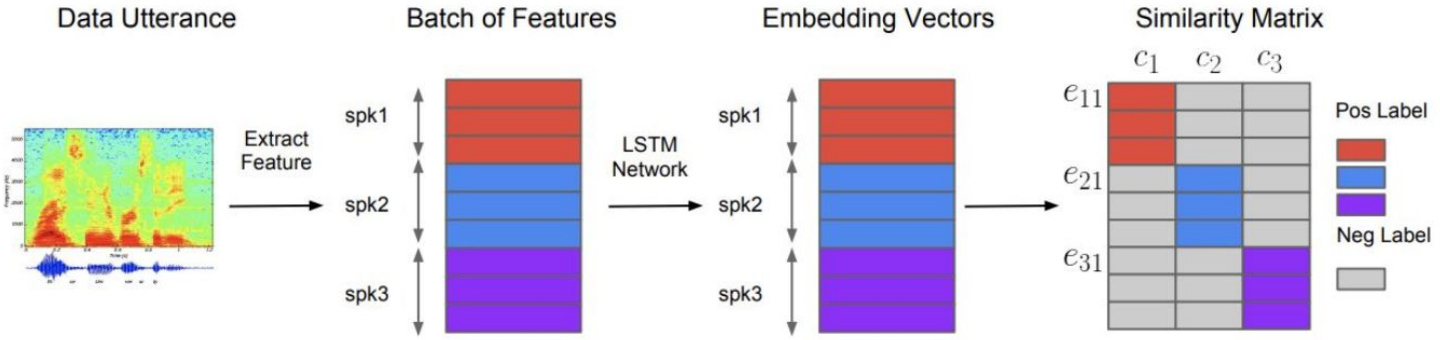
- Train with large datasets of speaker-labelled audio
- Feed frozen embeddings to TTS model at training and inference time
- If the training dataset is sufficiently diverse, zero shot synthesis is possible for new speakers with a single utterance
- Audio Samples

Jia et al. 2018

# Learned Speaker Embeddings



# Learned Speaker Embeddings



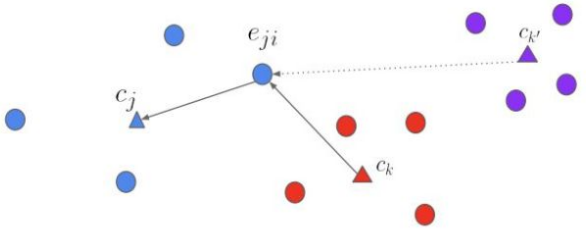
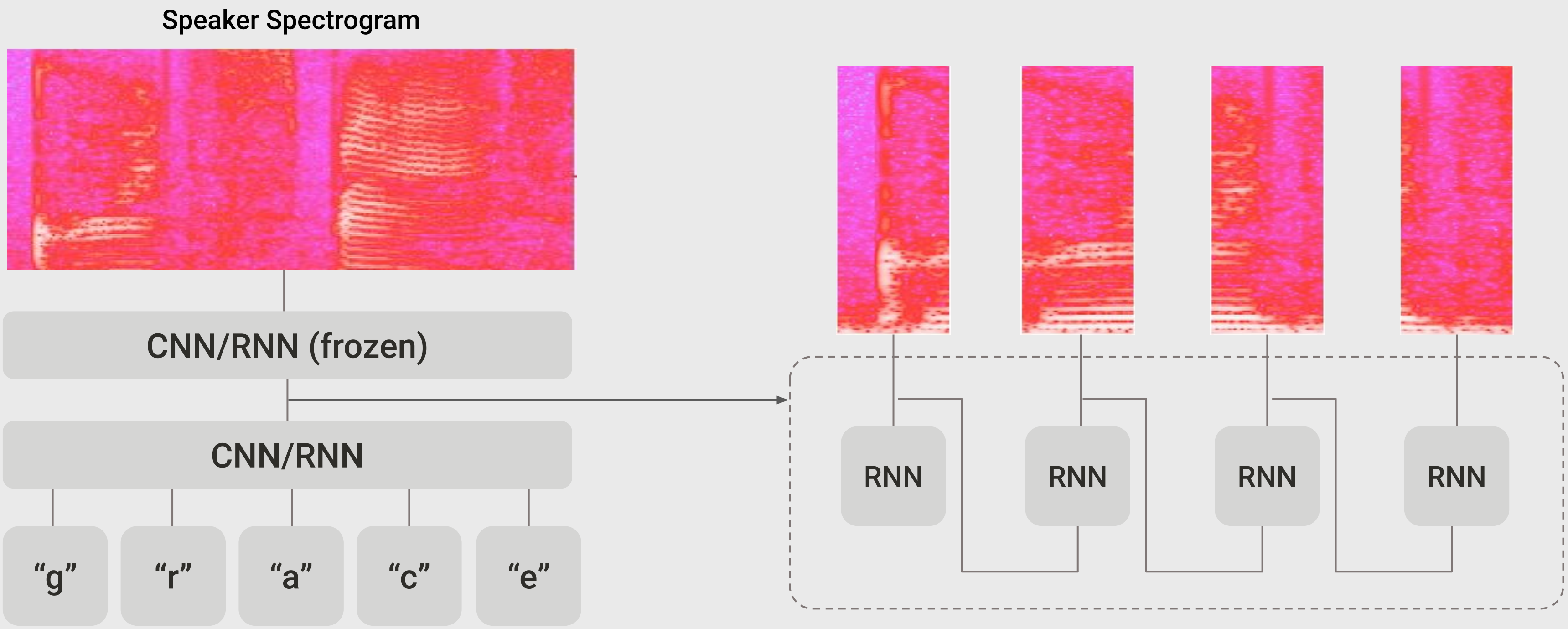




**Figure 1:** System overview. Different colors indicate utterances / embeddings from different speakers.

**Figure 2:** GEE loss pushes the embedding towards the centroid of the true speaker, and away from the centroid of the most similar different speaker. [Wang et al 2017]

# Learned Speaker Embeddings



# Learned Style Embeddings

- Instead of explicitly labelling style, can we get the model to learn structure in the audio data organically?
- Feed in the mel spectrogram as an input to a style module at training time
- Compress with conv/lstm to prevent trivial reconstruction
- At inference time feed in a reference mel spectrogram or sample from the latent space
- Can be achieved with token embeddings or a VAE
- Known as a “reference encoder” in the literature

# GST Tacotron

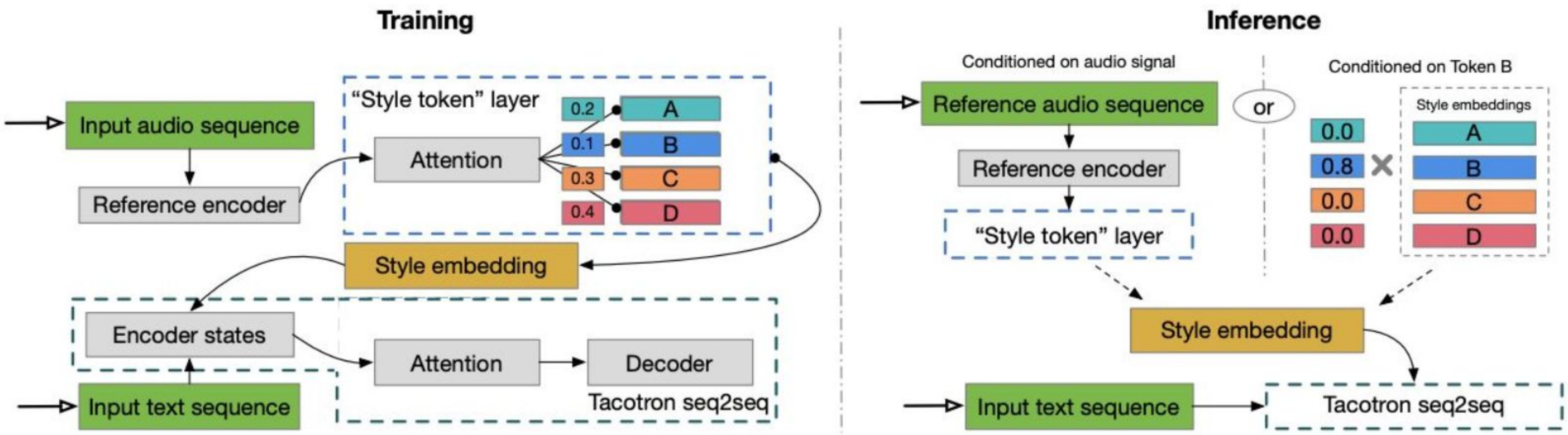



**Figure:** Model diagram. During training, the log-mel spectrogram of the training target is fed to the reference encoder followed by a style token layer. The resulting style embedding is used to condition the Tacotron text encoder states. During inference, we can feed an arbitrary reference signal to synthesize text with its speaking style. Alternatively, we can remove the reference encoder and directly control synthesis using the learned interpretable tokens.

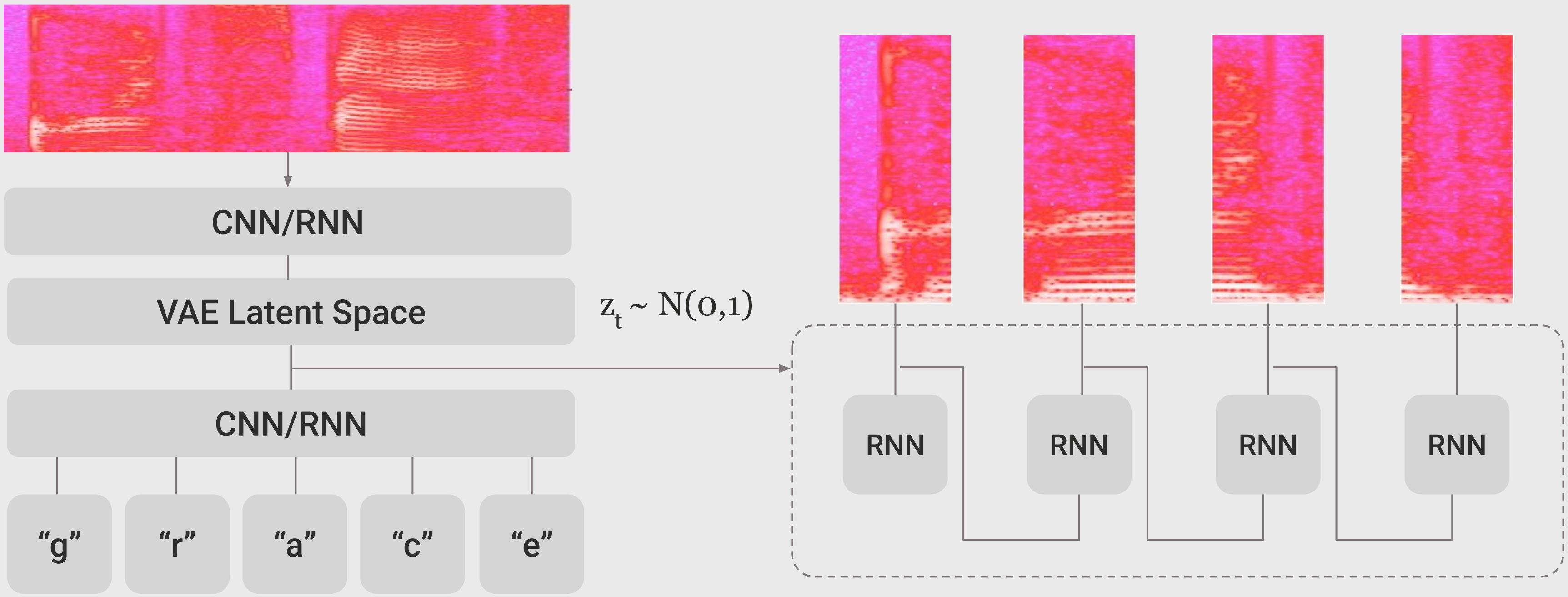
[Wang et al 2017](#) | [Samples](#)

# VAE Tacotron

- Use variational auto encoder for style latent space
- Latent space then encouraged to follow a gaussian distribution
- Sample prosodies from latent space at inference time
- GMVAE Tacotron uses a hierarchical mixture of gaussians so each component learns a different prosodic component of the data
- Fine-grained VAEs learn the variability in the model's prosody. This can be useful when generating data for semi-supervised ASR

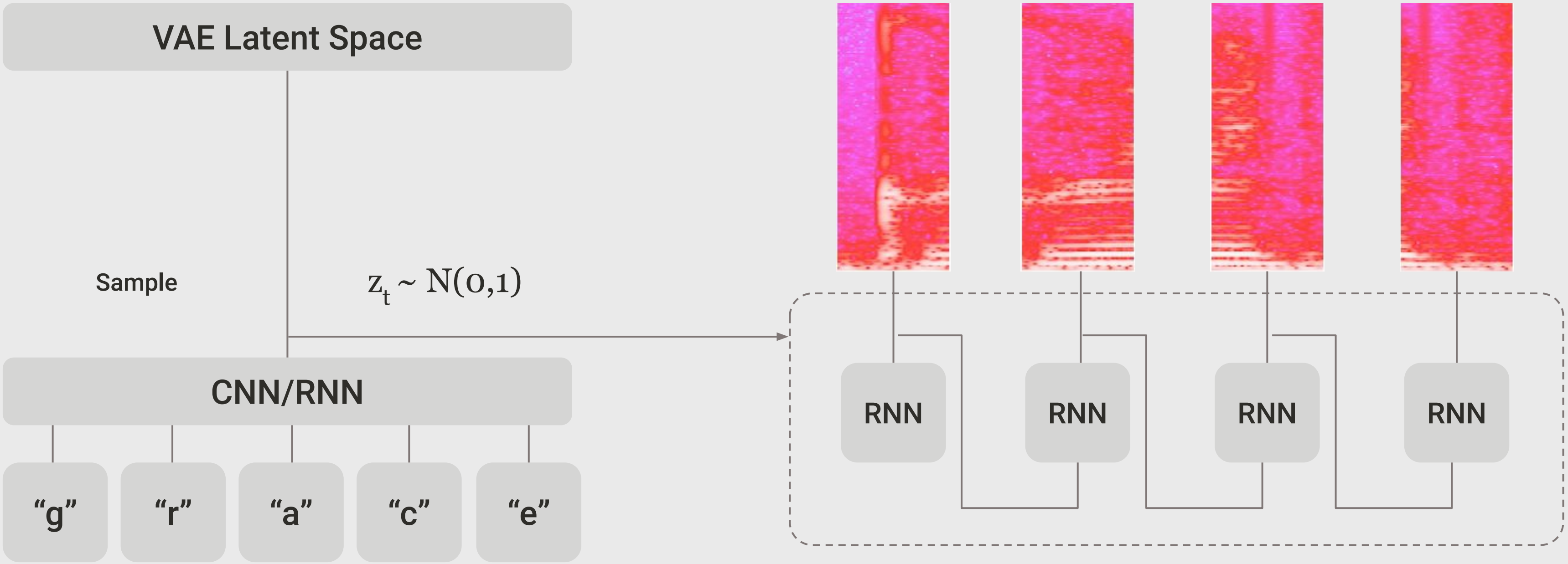
Hsu et al. 2018, Sun et al. 2020

# Training



CNN/RNN
VAE Latent Space
$z_t \sim N(0,1)$
CNN/RNN
"g"
"r"
"a"
"c"
"e"
RNN
RNN
RNN
RNN

# Inference


VAE Latent Space
Sample
$z_t \sim N(0,1)$
CNN/RNN
"g"
"r"
"a"
"c"
"e"
RNN
RNN
RNN
RNN