SynthSing: A Singing Voice Synthesizer

James Bunning
Shiyu Mou
Sharada Murali
Mu Yang
Yixin Yang

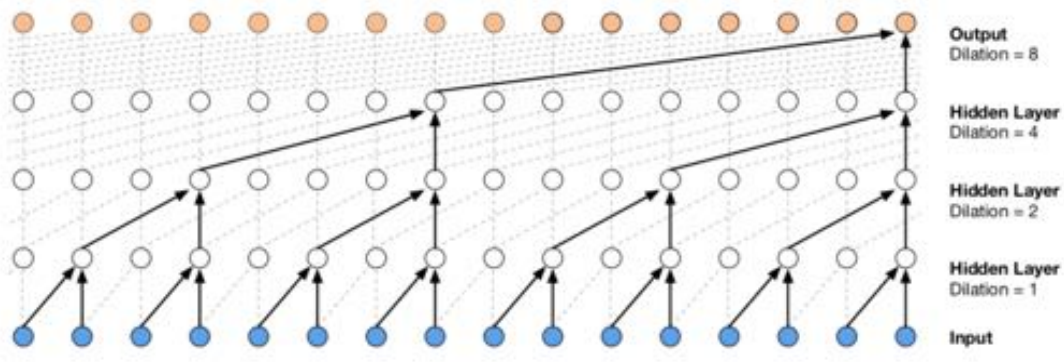
Introduction

Background

- Machine learning for TTS has greatly increased the quality of generated speech.
- WaveNet - deep generative model capable of producing very realistic-sounding speech.
- These advancements in TTS can be applied to singing synthesis to produce results far better than concatenative methods.

Model Details

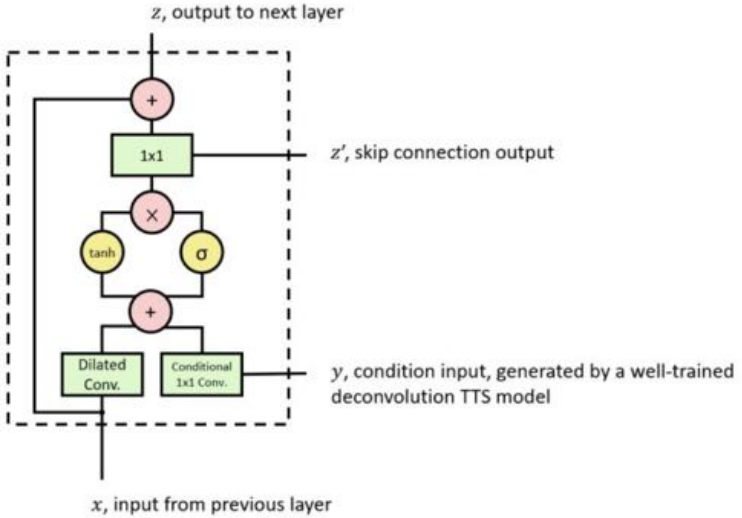
Network Architecture Details: WaveNet



- Autoregressive - uses past predicted values as input to predict the next value.
- Dilation convolution to increase receptive field.

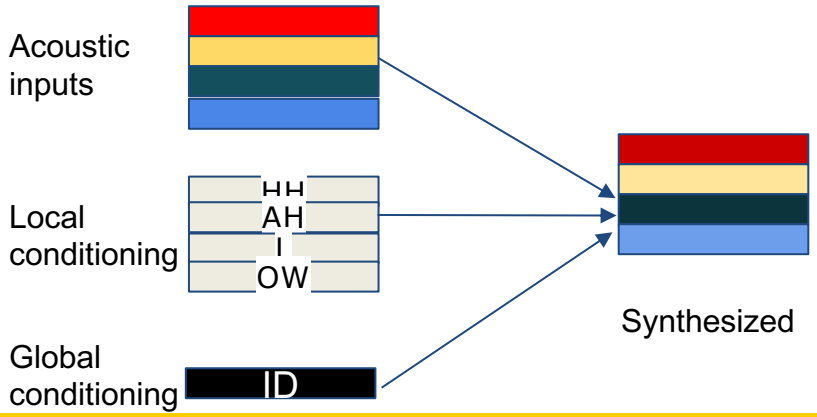
Network Architecture Details: WaveNet

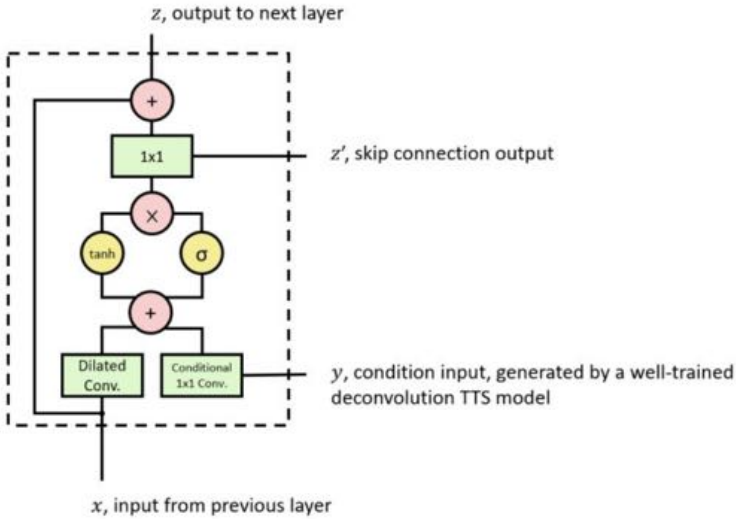
- Conditioned on control inputs
  - Global conditioning, i.e. singer identity
  - Existing implementation
  - Audio samples generated by a Vanilla Wavenet trained with global conditioning on MIR-1k, a Chinese Acapella singing dataset.



Network Architecture Details: WaveNet

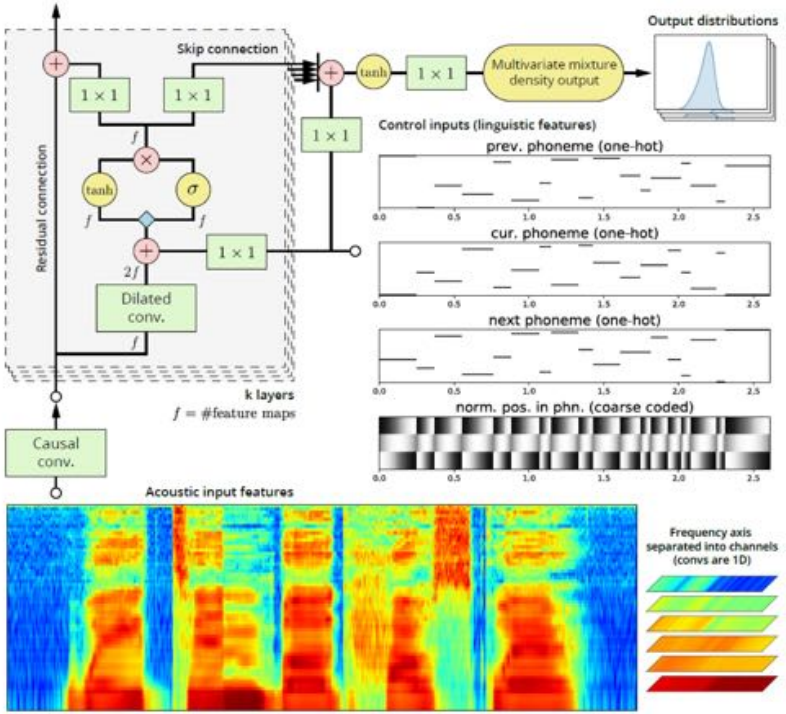
- Conditioned on control inputs
  - Local conditioning: phonetic features
  - Implemented by ourselves

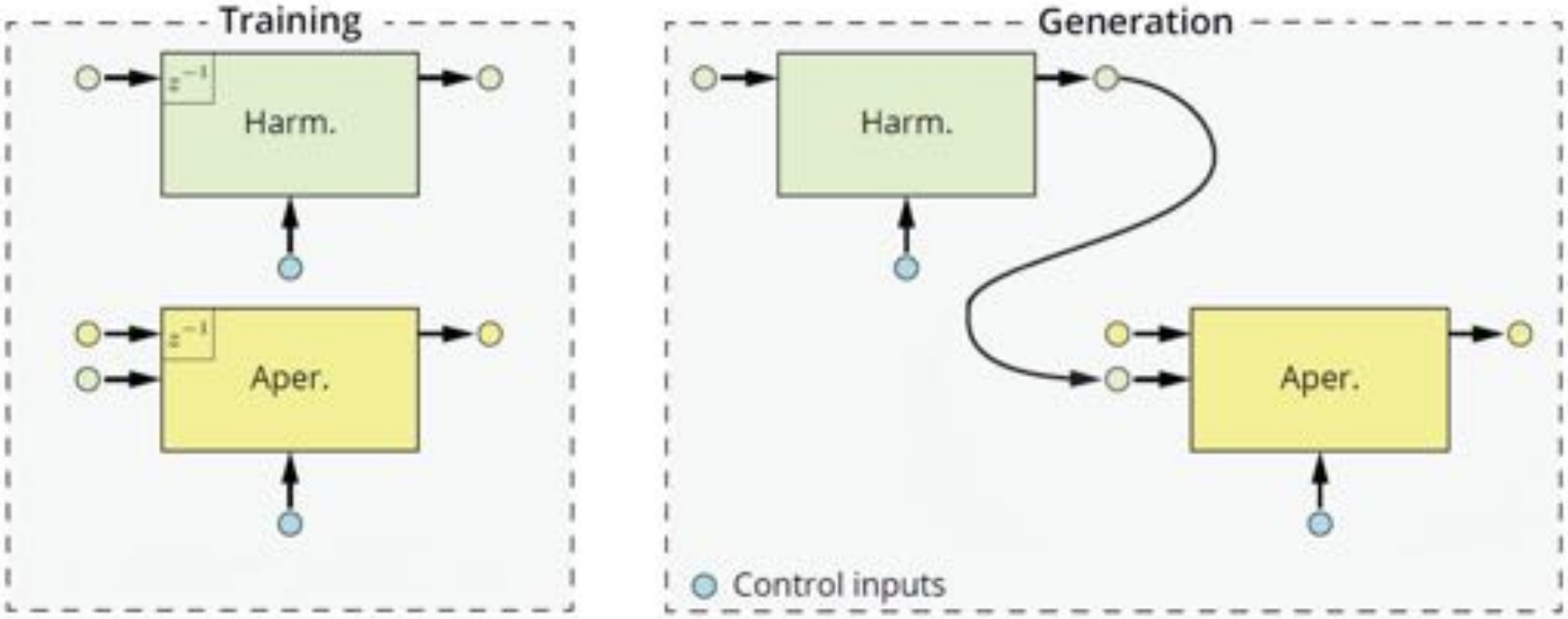




Features of the Proposed Model

- Acoustic inputs:
  - 60 log-MFSC
  - 4 Band Aperiodicity Coefficients
- Control inputs:
  - F0 (coarse coded)
  - Previous, current, next phoneme identities (one-hot coded)
  - Normalized phoneme position (coarse coded)




Training
Harm.
Aper.
Generation
Harm.
Aper.
Control inputs

Building the Model

Dataset

Two datasets:

1) Nagoya Institute of Technology (Nitech) Japanese nursery rhyme dataset:
   - 31 raw audio clips with time-aligned phoneme-level labels.
   - 34 phonemes (including “pau”).

2) Coldplay dataset:
   - 10 Coldplay songs with isolated vocals downloaded from YouTube.
   - Time-aligned phonemes generated by Gentle.
   - 39 phonemes (including “pau”).

Phoneme Generation & Alignment

Phoneme Preprocessing

Original

```
0 11800000 pau
11800000 12250000 d
12250000 16350000 e
16350000 17800000 N
17800000 18350000 d
18350000 20950000 e
20950000 22600000 N
22600000 23950000 m
23950000 27200000 u
27200000 28250000 sh
28250000 28950000 i
28950000 29650000 m
29650000 31800000 u
31800000 33000000 sh
```

Current-Previous-Next Phonemes

	beg	end	current	before	next
0	0	23800000	pau	None	n
1	23800000	24400000	n	pau	e
2	24400000	30450000	e	n	N
3	30450000	35800000	N	e	n
4	35800000	36200000	n	N	e

Time to Frame conversion

	beg	end	current	before	next
0	0	233	pau	None	n
1	233	239	n	pau	e
2	239	299	e	n	N
3	299	353	N	e	n
4	353	357	n	N	e

One-hot-encoding and positioning coarse coding

	beg	end	before	current	next	coarse
0	0	233	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	[1, 0, 0]
1	0	233	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	[0.9871609871609871, 0.008584008584008583, 0.0...
2	0	233	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	[0.9744671403197158, 0.017095914742451153, 0.0...
3	0	233	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	[0.961915688959621, 0.025537584662644806, 0.01...
4	0	233	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	[0.949503924181845, 0.03391085443506589, 0.016...

Phoneme Alignment w/ GENTLE

- Robust yet lenient forced-aligner built on Kaldi. A tool for aligning speech with text.
- We input music files and corresponding lyrics to Gentle, then we get aligned phonemes in Json format.
- The Json file is converted to the same format as the NIT Japanese dataset.


```
{
 "transcript": "When you try your best but you don't succeed \r\n
what you need \r\nWhen you feel so tired but you can't sleep ",
 "words": [
  {
   "alignedWord": "when",
   "case": "success",
   "end": 0.35,
   "endOffset": 4,
   "phones": [
    {
     "duration": 0.1,
     "phone": "w_B"
    },
    {
     "duration": 0.07,
     "phone": "eh_I"
    },
    {
     "duration": 0.07,
     "phone": "n_E"
    }
   ],
   "start": 0.11,
   "startOffset": 0,
   "word": "When"
```

Acoustic Feature Extraction Using WORLD Vocoder

WORLD Vocoder - F0

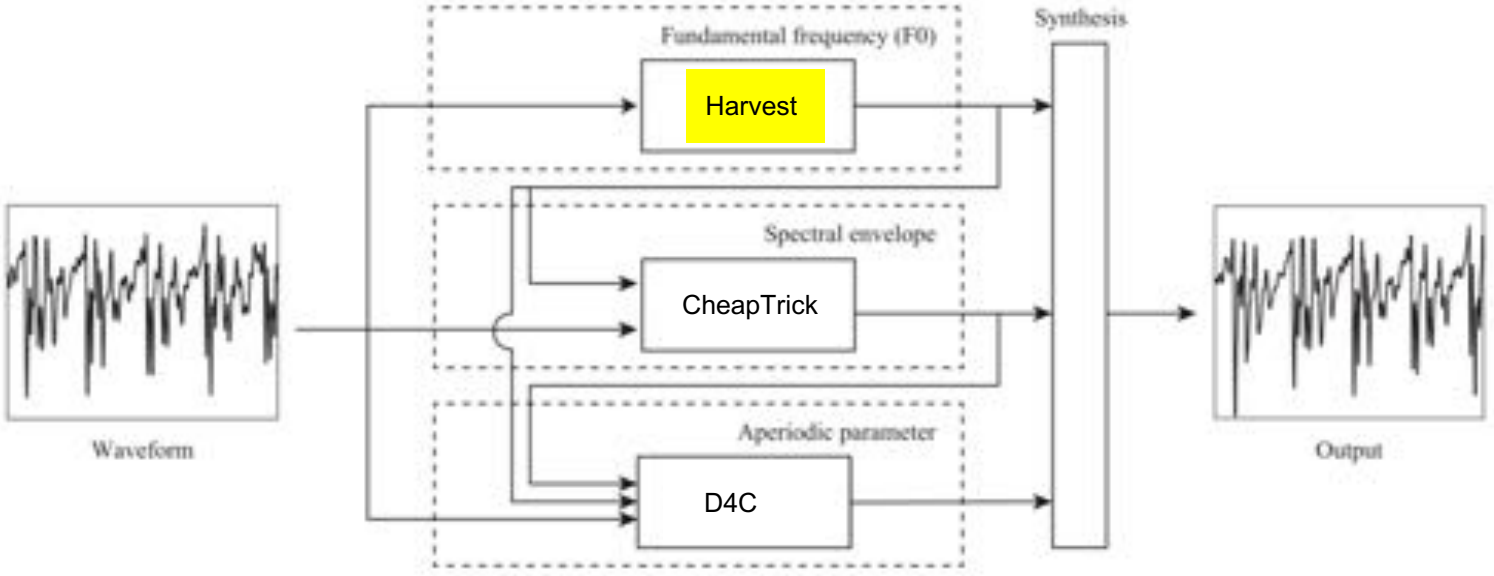


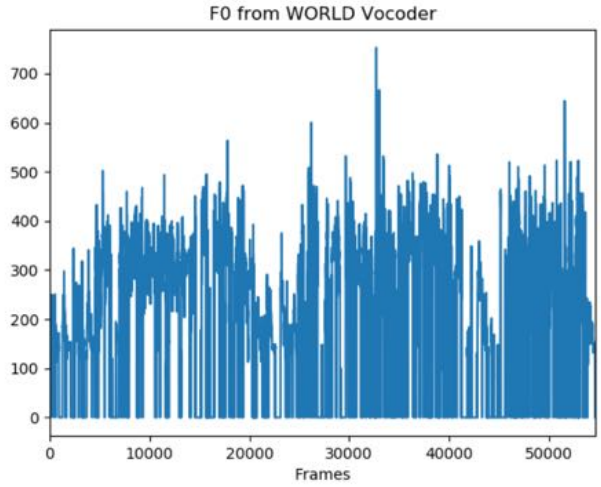
Fig. 1 Overview of the developed system. WORLD consists of three analysis algorithms for determining the F0, spectral envelope, and aperiodic parameters and a synthesis algorithm incorporating these parameters.

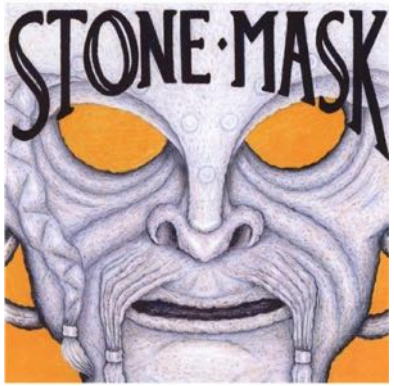
F0 extraction

Harvest Algorithm

1) Obtain F0 candidates using several band-pass filters with different center frequencies.
2) Refine and score basic F0 candidates using the instantaneous frequency, and then estimate several F0 candidates in each frame.
3) Run a connection algorithm using neighboring F0s to generate a reliable F0 contour based on the candidates.

```
f0, t = pw.harvest(x, fs)
```





F0 refinement

```
f0_ref = pw.stonemask(x, f0, t, fs)
```

StoneMask Algorithm

Refine voiced/unvoiced decision.

WORLD Vocoder - Spectral Envelope

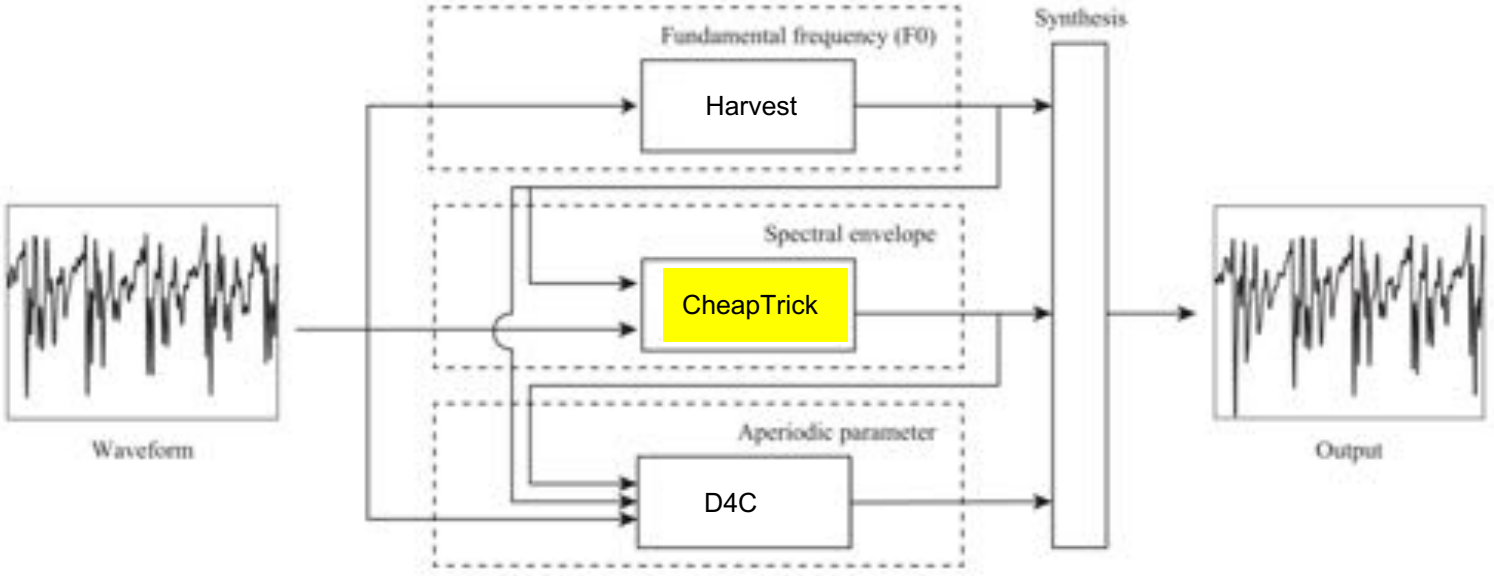
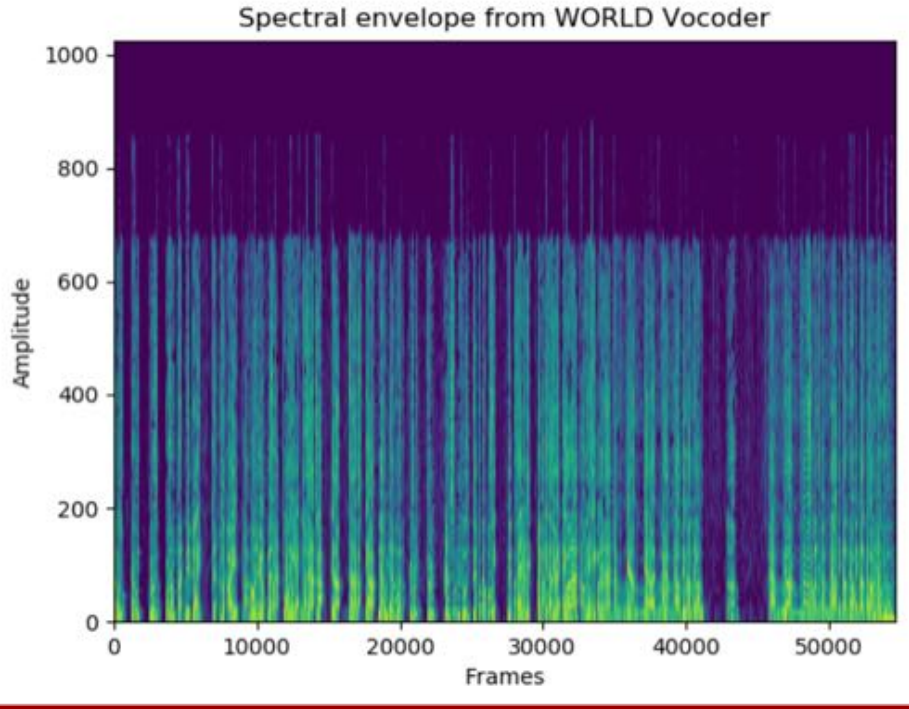


Fig. 1 Overview of the developed system. WORLD consists of three analysis algorithms for determining the F0, spectral envelope, and aperiodic parameters and a synthesis algorithm incorporating these parameters.

Harmonic Spectral Envelope estimation

```
sp = pw.cheaptrick(x, f0, t, fs)
```



CheapTrick Algorithm

1) Adaptively window F0.

2) Smooth frequency domain of the power spectrum.

3) Lifter in the quefrency domain for spectral recovery.

WORLD Vocoder - Aperiodicity Envelope

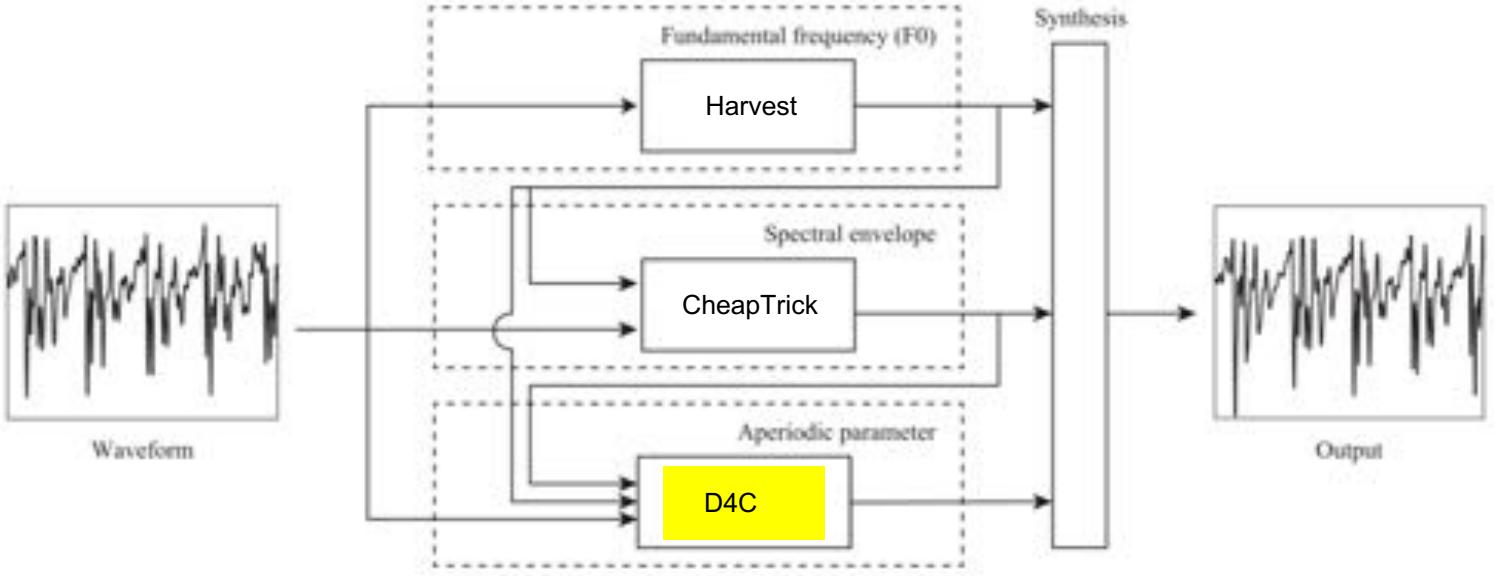
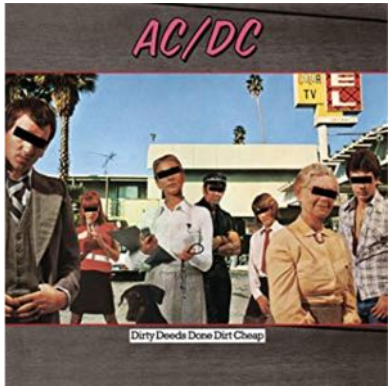


Fig. 1 Overview of the developed system. WORLD consists of three analysis algorithms for determining the F0, spectral envelope, and aperiodic parameters and a synthesis algorithm incorporating these parameters.



Band aperiodicity estimation

```
ap = pw.d4c(x, f0, t, fs)
```

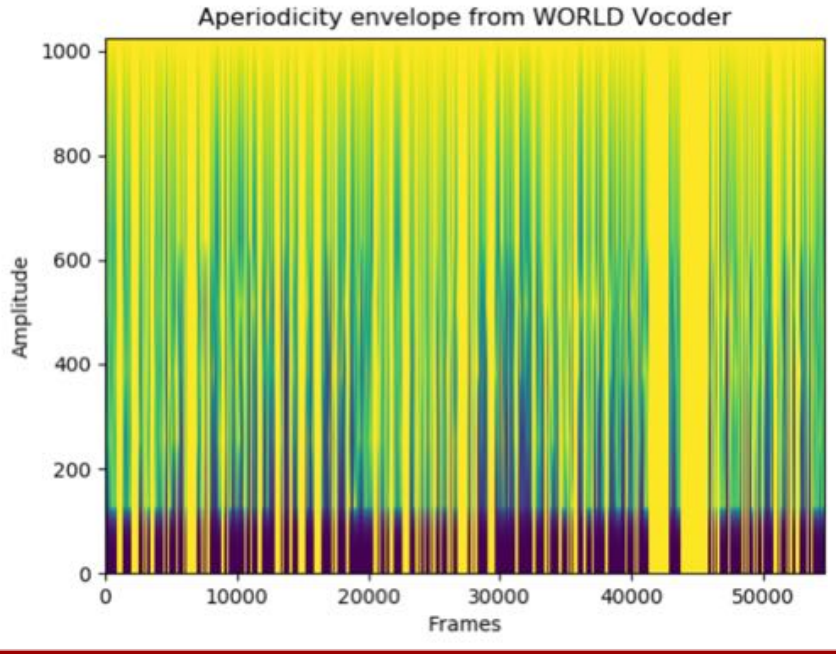
D4C Algorithm

1) Calculate temporally static parameter on basis of group delay.

2) Calculate parameter shaping.

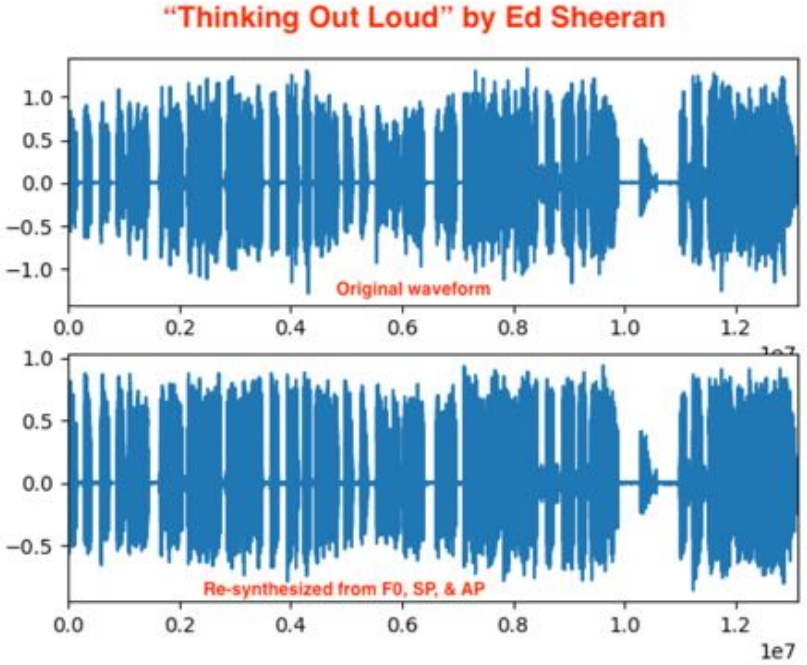
3) Estimate band-aperiodicity.

*Reduce dimensionality from 1025 to 4 coefficients.



Re-synthesis

```
y_h = pw.synthesize(f0, sp, ap, fs, frame_period)
```



F0 Refinement and Coarse Coding

- Refined F0 from WORLD is further interpolated; stray zeros removed.
- Conversion to cents using reference F0 for the speaker.

$$f_{cents} = \mathbf{1200} \times log_2(f/f_{ref})$$

- These values are used to coarse code F0 into 3 states.

	0	1	2
3163	0	0.759692	0.240308
3164	0	0.703291	0.296709
3165	0	0.790704	0.209296
3166	0	0.880443	0.119557
3167	0.0400257	0.959974	0
3168	0.0258179	0.974182	0
3169	0.0636846	0.936315	0
3170	0.0637071	0.936293	0
3171	0.0280956	0.971904	0
3172	0.122027	0.877973	0
3173	0.162933	0.837067	0
3174	0.137404	0.862596	0
3175	0.0171952	0.982805	0
3176	0	0.991889	0.00811072
3177	0	0.936596	0.0634037
3178	0	0.906007	0.0939933
3179	0	0.906007	0.0939933

Spectral Envelope to log-MFSC

- Dimensionality of the spectrum (2048) is reduced to 60.
- Converted using frequency warping in the cepstral domain with warping coefficient α=$\mathbf{0.45}$.
- Using Speech Signal Processing Toolkit (SPTK) for conversion to cepstrum.

```
mc = pysptk.conversion.sp2mc(sp, order, alpha)     # Mel cepstrum
```

Changes to existing WaveNet Architecture

Softmax vs. CGM

- Traditional Wavenet: **μ**-law encoding with 256-way softmax prediction.
- Vocoder features vary smoothly; modeled to be continuous.
- GMM with 4 components = 12 parameters per input feature.
- Reduced further to 4 using Constrained Gaussian Mixtures (CGMs).

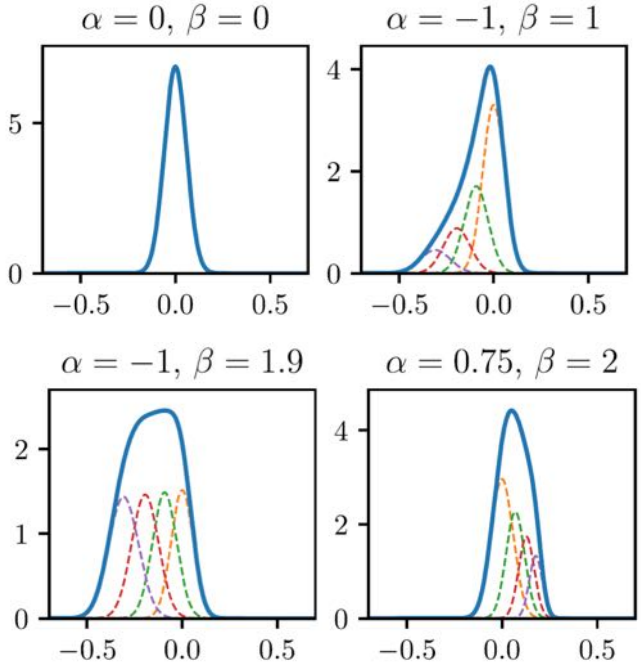
$$softmax(z_i) = \frac{exp(z_i)}{\sum_j exp(z_j)}$$

$$\begin{bmatrix} 0 \\ 0 \\ . \\ . \\ . \\ . \\ 1 \\ 0 \\ . \\ 0 \end{bmatrix}$$

Each sample in Wavenet after encoding

Softmax vs. CGM

- Used to predict skewness, shape, location and scale.
- Result: Huge reduction in parameter space.
- Speeds up generation time.
  - Softmax: 20+ minutes for 10s of 16KHz audio
  - CGM: 2-3 minutes for 10s of 32KHz audio



Softmax v. CGM: Modifying the Loss function

- As GMM parameters are predicted, cannot use existing loss function anymore (softmax cross-entropy).
- Instead, use negative log-likelihood:

$$\mathcal{L}(y \,/\, x) = -log\left[\sum_{k}^{K} \Pi_k(x)\phi(y, \mu(x), \sigma(x))\right]$$

Experimentation and Results

Regularization

- Training the model on ground truth data can lead to the model being overly biased towards the non-control inputs and ignore the control inputs.
- As a way around this, we add noise (modeled as a standard normal) with level 0.4.
- During training, input to the network is noisy and min-max normalized, and the target is the uncorrupted values (without normalization).

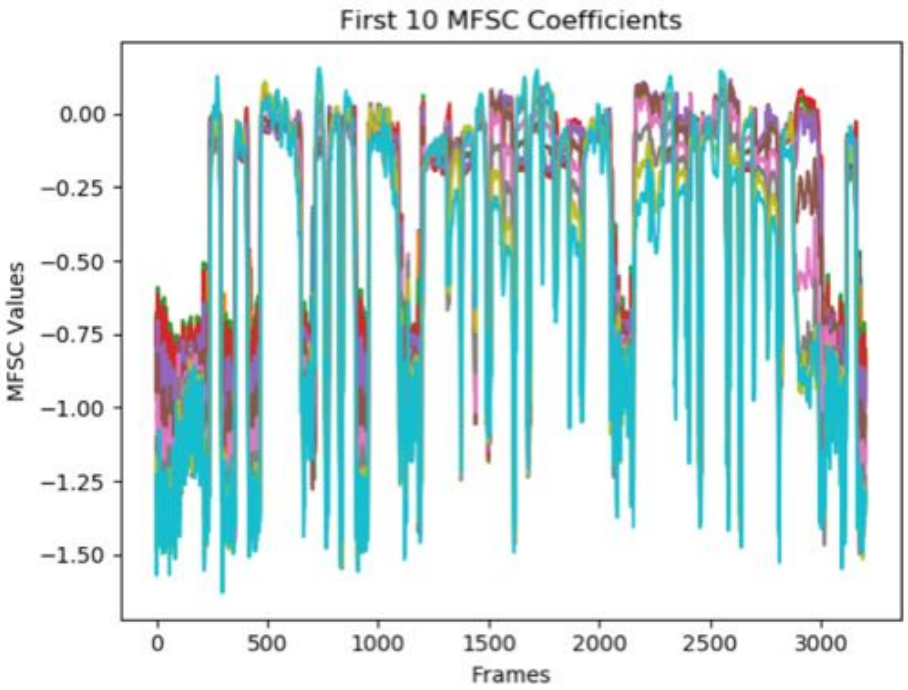
Training the Models

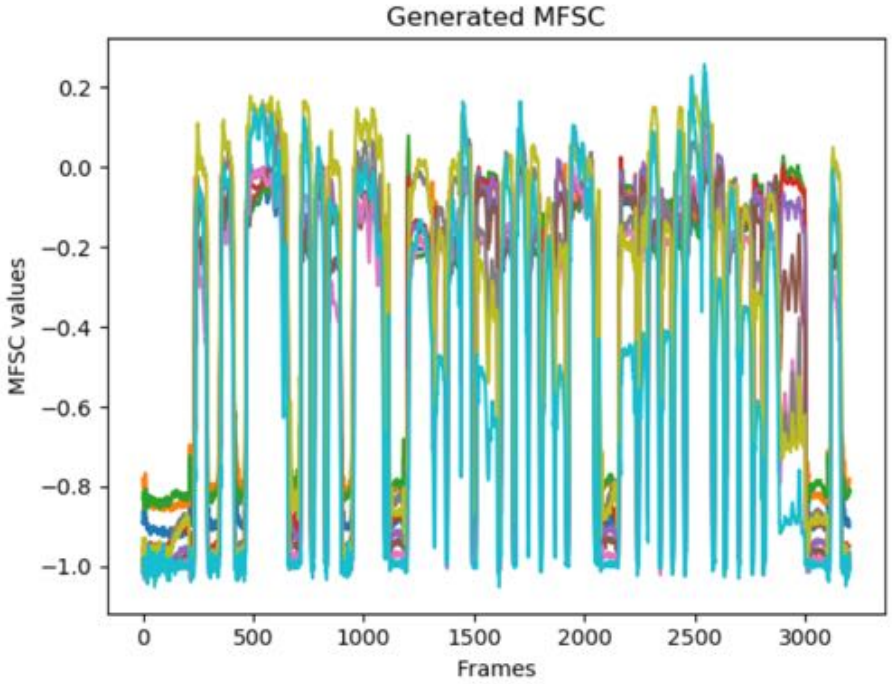
- Total number of trainable variables:
  - Harmonic Model: 925,714
  - Aperiodicity Model: 54,860
- Each sub-model is trained for 10M timesteps.
- Training time: 3-4 hours per sub-model.

1. Replicating results from the dataset

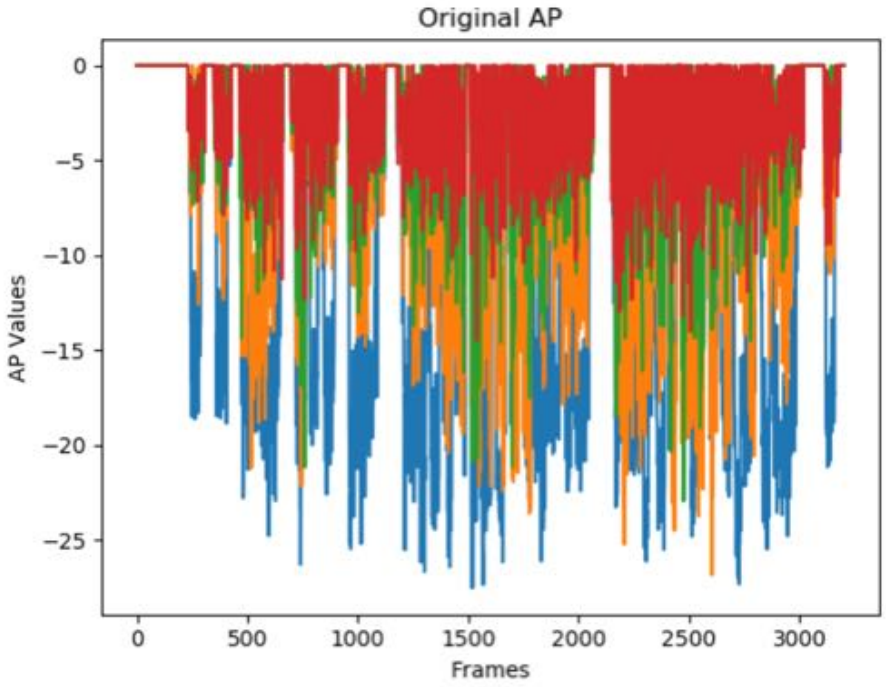
- Generated samples from the Japanese dataset from the MFSC and AP models.
- Resynthesized using true F0, generated MFSC and generated AP in WORLD vocoder.

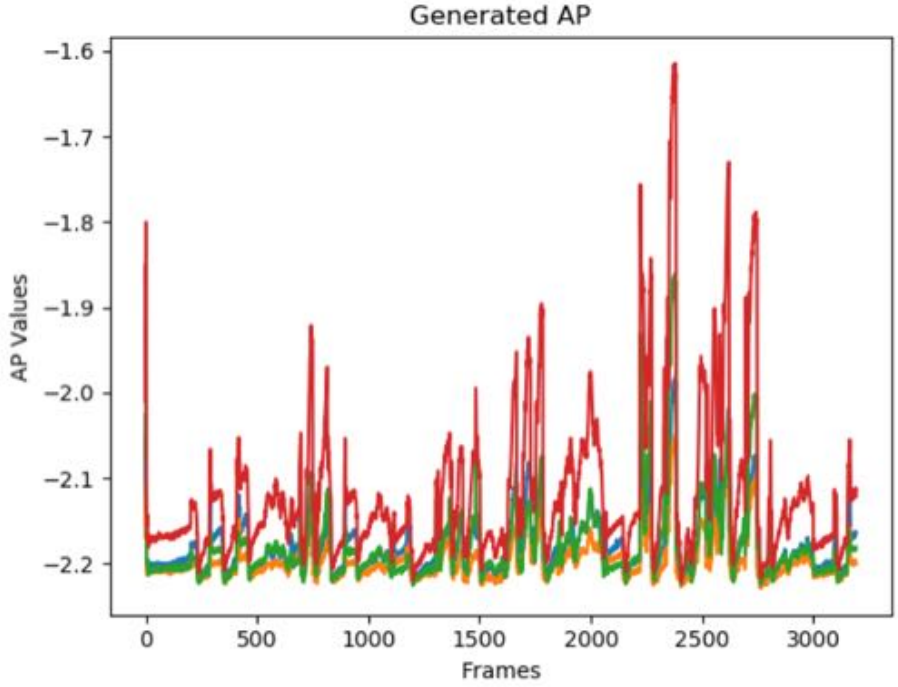
1. Replicating results from the dataset





1. Replicating results from the dataset





1. Replicating results from the dataset

- Due to the autoregressive nature of the model, errors compound over time.
- Generated AP diverges from ground truth values.
- However, experimentation showed that quality of synthesis is not adversely affected.

Original

Synthesized

2. Generating previously unseen sequences

- By concatenating random sequences of phonemes together, created data that the model had not seen before.
- Generated singing sounds natural, but some phonemes are not distinguishable.

Original

Synthesized

3. Using self-created dataset

- Coldplay dataset
  - Generated samples from the Coldplay dataset using the MFSC model.
  - Resynthesized using true F0 and AP coefficients and generated MFSC via WORLD vocoder.

Original

Synthesized

Tacotron

Tacotron is an integrated end-to-end TTS model.

1. The input data is just sequences of word + aligned corresponding wav audio files.
2. When generating, 10 seconds of audio file will be generated for each input text sentence.
3. The output sounds like it’s patching samples together.

Tacotron Dataset

- Self-generated dataset.
- Pairs of line-by-line trimmed audio and lyrics.
- Back to 00's. Utilized the .LRC lyrics files searched online.
- Clean voice audio is segmented using the timing information provided in .LRC file.
- Then the data is organized in the same manner of the LJ Speech Dataset.

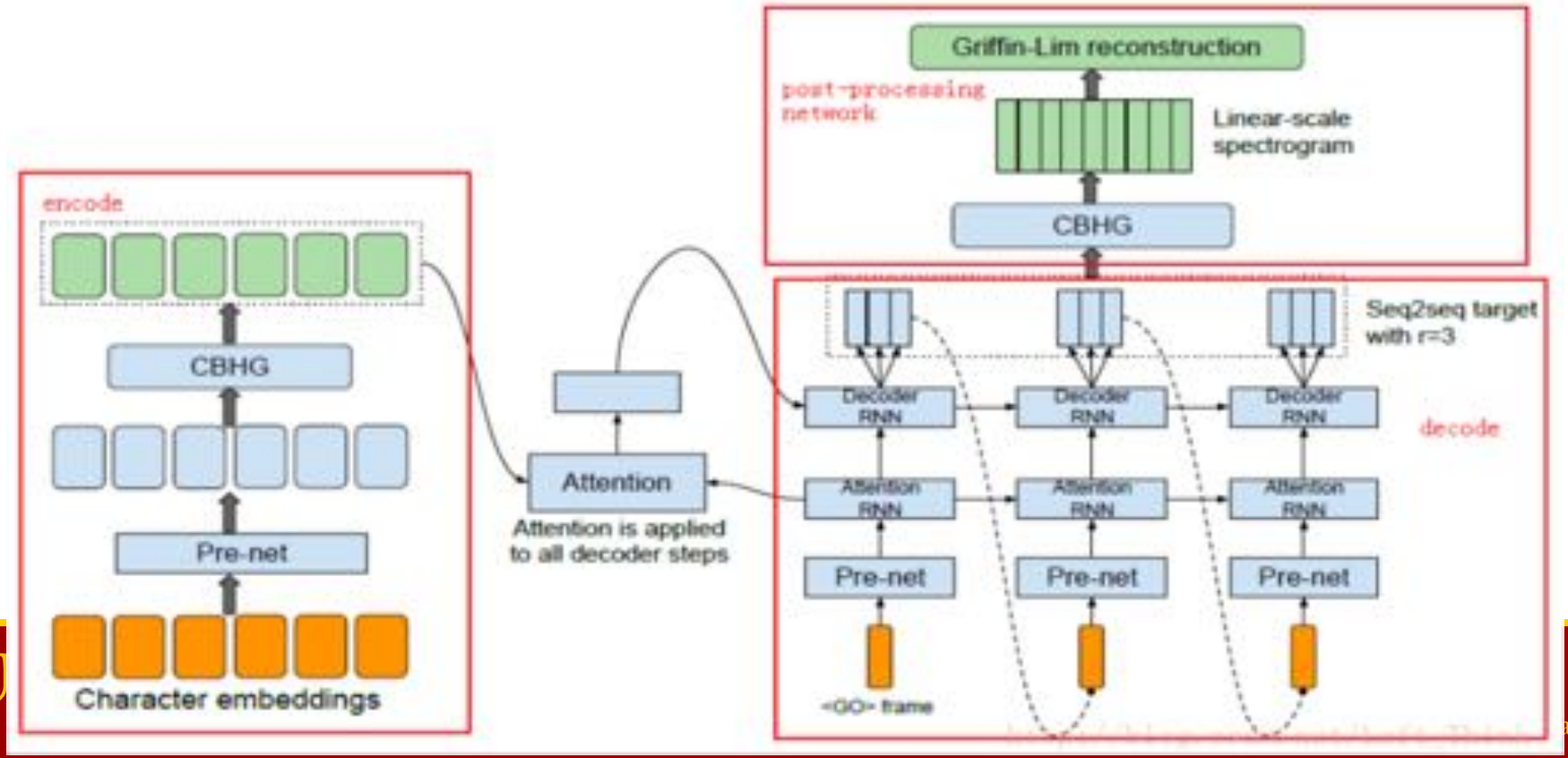
```
[01:10.280]Ooh coloured crimson in
[01:13.160]
[01:13.910]One or two could free my
[01:17.400]
[01:21.580]This is how it ends I fe
[01:24.770]Chemicals burn in my
[01:26.720]
[01:27.700]Bloodstream
[01:28.300]
[01:31.540]Fading out again
[01:33.470]
[01:34.070]I feel the chemicals
[01:36.000]Burn in my
[01:36.880]
[01:38.020]Bloodstream
[01:38.530]
[01:40.090]So tell me when it kicks
[01:41.340]
[01:50.020]So tell me when it kicks
[01:51.720]
[01:52.500]Nononono
[01:54.040]
```

Tacotron Dataset (contd)

Problem:

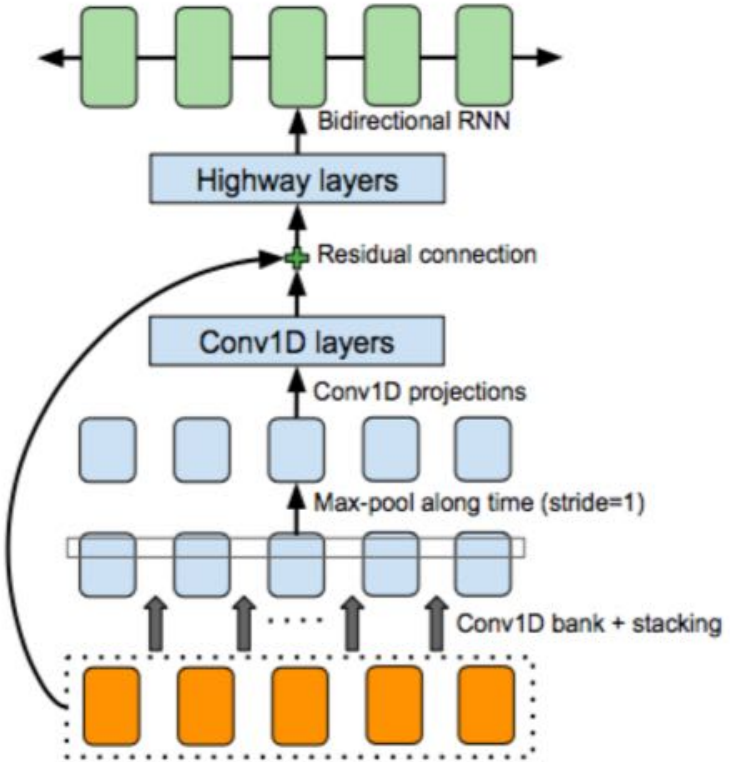
- The .lrc files are not 100 percent accurate. After line-by-line segmentation, you can hear parts of the beginning word of the next sentence in the current line's audio. This is because for displaying synced lyrics, perfect timing information is not needed.
- .lrc files are group-sourced by various people online. Many transcriptions are "lazy". For example, someone writes "takin'" instead of "taking".
- Since we're using clean voice audio generated by various people online, certain sections might be dropped (as a result of removing instrumental sections where no signing is present). This causes mismatching with the .lrc files. In this case, we set an offset by hand.

Model Architecture



CBHG

- A bunch of 1-D convolutional filters
  + highway networks
  + bidirectional RNN (GRU)

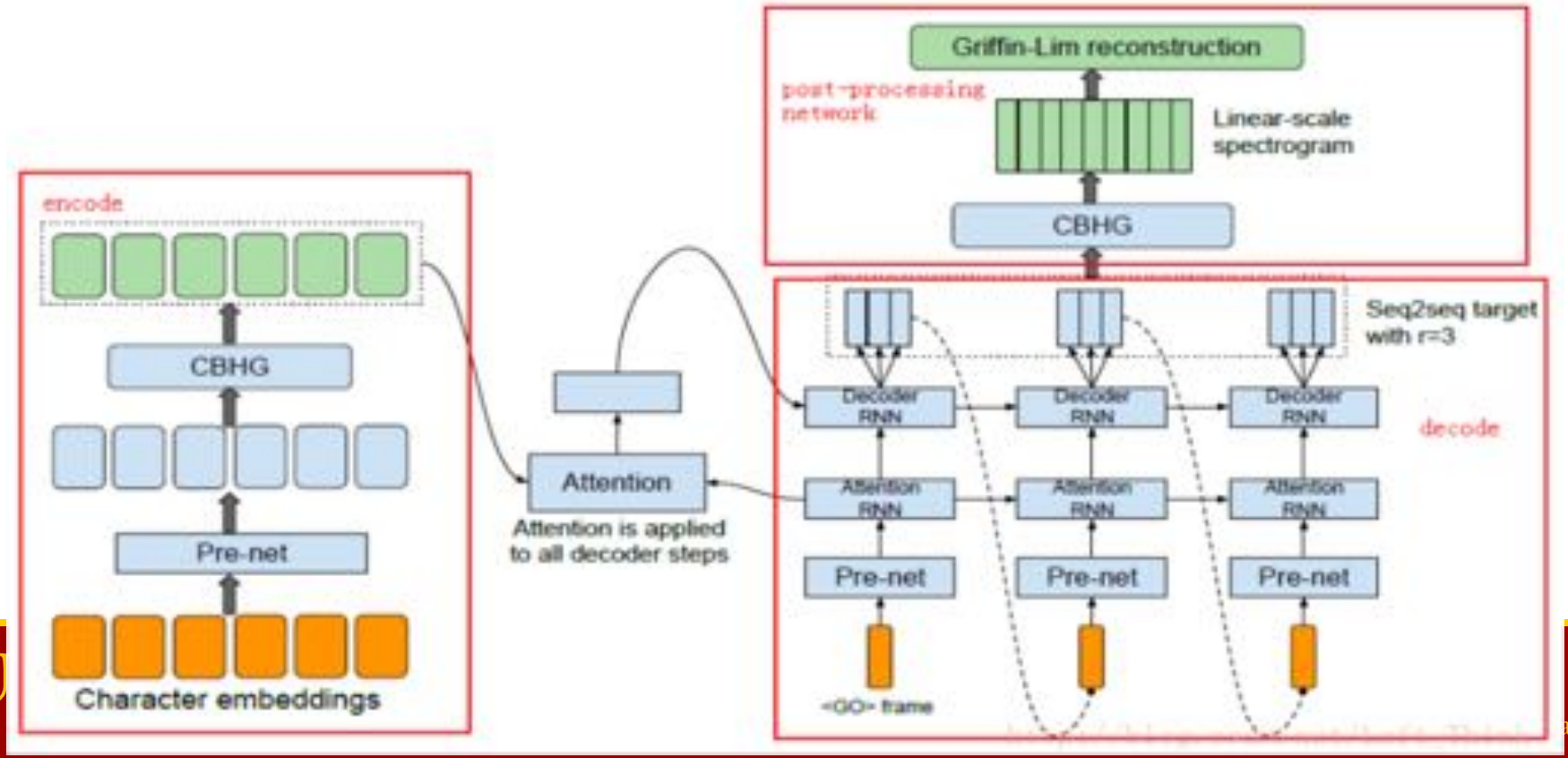


Highway networks

The structure of each layer of highway nets is: put the input into the two-layer fully connected network at the same time. The activation functions of the two networks use ReLu and sigmoid functions respectively, assuming the output of ReLu is output1. The output of sigmoid is output2, then the output of the highway layer is

$$output = output1 * output2 + input * (1 - output2)$$

Model Architecture

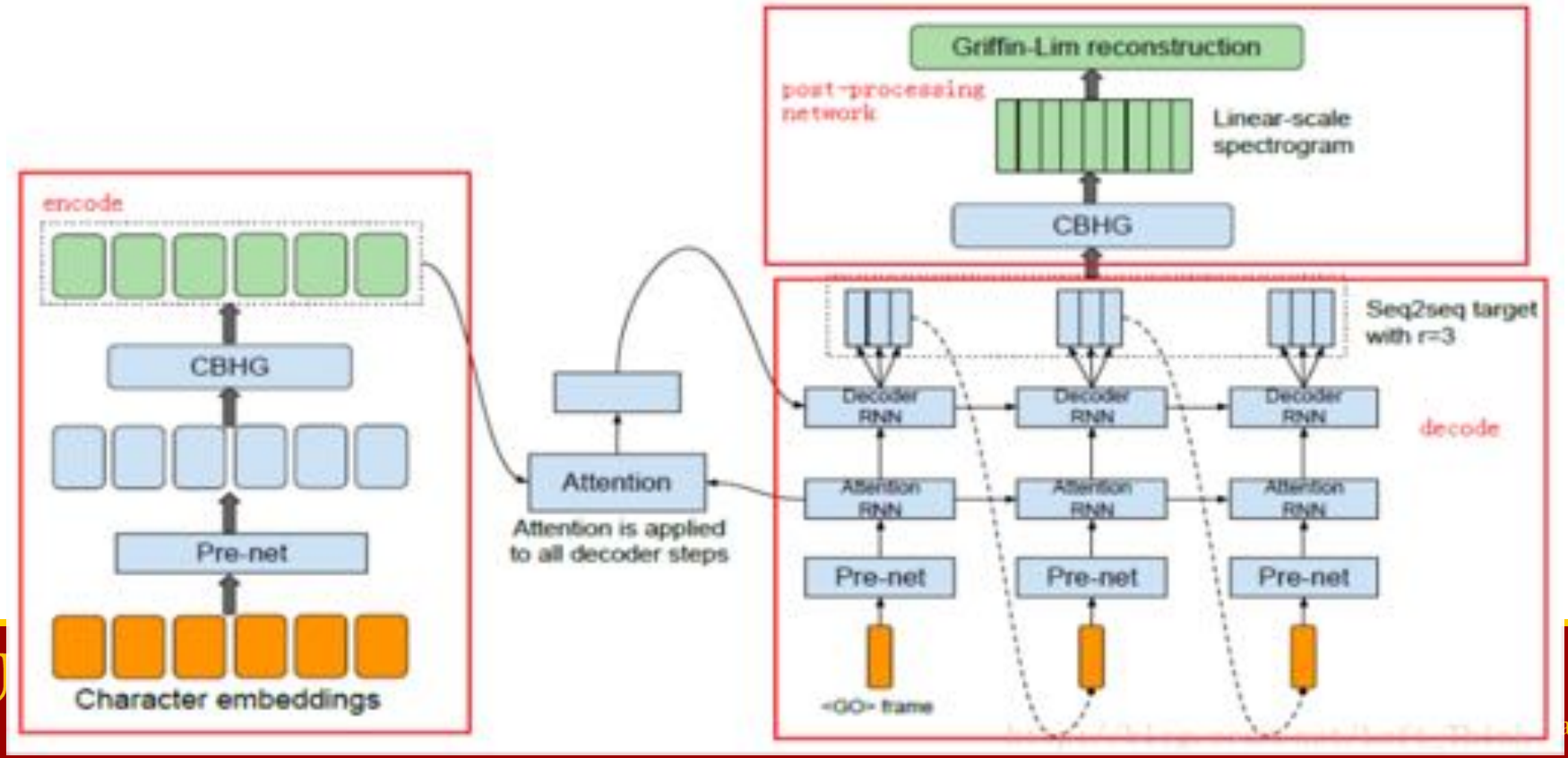


Attention Mechanism

- A disadvantage of this architecture is that the output context of the encoder is too small, so it is difficult to generalize all the semantic details of a long sequence. One solution is to introduce an attention mechanism.

- Each time a word is predicted, the output of the encoder is given different weights, weighted and summed, and then input into the decoder.

Model Architecture



Comparing SynthSing with Tacotron

- SynthSing generation sounds more natural, though both were trained with similar dataset sizes.
- Tacotron requires a lot more data than SynthSing for comparable performance.
- No control over pitch output in Tacotron.
- Data preparation for Tacotron is a lot easier.
- Tacotron's architecture is easier to generate.

Individual Contributions

- Data Preparation
  - WORLD Vocoder - James
  - Dataset creation - James
  - F0 coarse coding - Sharada
  - SP to log-MFSC conversion - Sharada
  - GENTLE phoneme alignment - James
  - Phoneme extraction and Encoding - Shiyu
  - Normalized position coarse coding - Shiyu

Individual Contributions

- Wavenet Model
  - Implementing Harmonic and Aperiodicity model - Mu
  - CGM Implementation - Mu, Sharada
  - Generation - Mu, Sharada
  - Fast generation from CGMs - Sharada
- Tacotron Model
  - Data preparation - Shiyu, Yixin
  - Model architecture - Yixin
  - Attention Mechanism optimization - Yixin
  - Training and Generation - Shiyu

Conclusions

Overall Observations

- Our synthesized MFSCs from Japanese training recordings sounded close to the ground-truth ones.
- While there were a few phonemes mis-pronounced, the synthesis was intelligible and the singing sounded natural.
- Our experiment with mismatched phonemes is an indication that the model scales well to outside data.
- The expanded receptive field of our WaveNet architecture allows us to train on more modestly-sized datasets.
- Our parametric approach allowed for significantly reduced training and generation times versus the sample-by-sample WaveNet model.

Comparison with NPSS

- Only timbre model implemented; no conversion of notes to F0.
- SynthSing can be used to train **multiple** singers simultaneously using global conditioning, unlike NPSS.

Challenges

- Preparation of English dataset
  - Finding enough clean audio data
  - Struggles with GENTLE
- Implementing coarse coding
- Implementing local conditioning
- Implementing CGM predictions
  - Hazy details
  - Tuning temperatures for training and generation
  - Going from slow generation (100 frames in 2 hours) to fast generation (100 frames in a few seconds)

Further Work

- Implementing the NPSS pitch model.
- Transfer learning between singers.
- Using source separation to extract clean audio for training.

References

[1] Blaauw, Merlijn, and Jordi Bonada, “A Neural Parametric Singing Synthesizer Modeling Timbre and Expression from Natural Songs,” Applied Sciences, 2017.

[2] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, “Wavenet: A generative model for raw audio,” 2016.

[3] M. Morise, F. Yokomori, and K. Ozawa: WORLD: a vocoder-based high-quality speech synthesis system for real-time applications, IEICE transactions on information and systems, vol. E99-D, no. 7, pp. 1877-1884, 2016.

[4] Y. Wang , RJ Skerry-Ryan∗ , D. Stanton, “TACOTRON: Towards end-to-end speech synthesis”, 6 April 2017.