

Adaptive Text-to-Speech in Low Computational Resource Scenarios

Flora Xue



Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2020-97

<http://www2.eecs.berkeley.edu/Pubs/TechRpts/2020/EECS-2020-97.html>

May 29, 2020

Copyright © 2020, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

Acknowledgement

I would like to thank Bichen Wu and Daniel Rothchild for their advice on this project along the way, to my co-workers Tianren Gao and Bohan Zhai for their collaboration on the SqueezeWave paper, and Prof. Joseph Gonzalez and Prof. Kurt Keutzer for their support and guidance throughout my master's career.

Adaptive Text-to-Speech in Low Computational Resource Scenarios

by Flora Xue

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences,
University of California at Berkeley, in partial satisfaction of the requirements for the
degree of **Master of Science, Plan II.**

Approval for the Report and Comprehensive Examination:

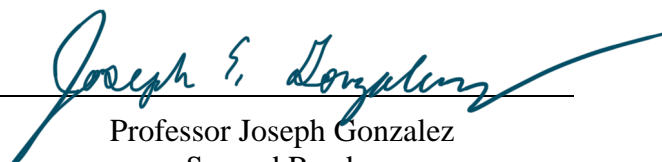
Committee:



Professor Kurt Keutzer
Research Advisor

May 29, 2020

(Date)



Professor Joseph Gonzalez
Second Reader

May 28, 2020

(Date)

Adaptive Text-to-Speech in Low Computational Resource Scenarios

Copyright 2020
by
Flora Xue

Abstract

Adaptive Text-to-Speech in Low Computational Resource Scenarios

by

Flora Xue

Master of Science in EECS

University of California, Berkeley

Professor Kurt Keutzer, Chair

Adaptive text-to-speech (TTS) system has a lot of interesting and useful applications, but most of the existing algorithms are designed for training and running the system in the cloud. This thesis proposes an adaptive TTS system designed for edge devices with a low computational cost based on generative flows. The system, which is only 7.2G MACs and 42x smaller than its baseline, has the potential to adapt and infer without exceeding the memory constraint and edge processor capacity. Despite its low-cost, the system can still adapt to a target speaker with the same similarity and no significant audio naturalness degradation as with baseline models.

To My Families

For your continued love and support along the way.

Contents

| | |
|---|------------|
| Contents | ii |
| List of Figures | iii |
| List of Tables | iv |
| 1 Introduction | 1 |
| 2 Related Works | 3 |
| 2.1 Adaptive TTS | 3 |
| 2.2 Vocoder Models | 3 |
| 2.3 Voice Conversion | 4 |
| 3 Preliminaries | 5 |
| 3.1 Flow-based Models | 5 |
| 3.2 Computational Complexity of Blow and Waveglow | 6 |
| 4 Methodology | 8 |
| 4.1 Flow-based Lightweight Mel-spectrogram Conversion: Blow-Mel | 8 |
| 4.2 Flow-based Lightweight Vocoder: SqueezeWave | 12 |
| 5 Evaluation | 16 |
| 5.1 Vocoder | 16 |
| 5.2 Mel-spectrogram Adaptation | 19 |
| 6 Conclusion | 25 |
| Bibliography | 26 |

List of Figures

| | | |
|-----|--|----|
| 4.1 | Overview of the Blow-mel model. | 9 |
| 4.2 | Structure of the coefficient version of the Blow-mel structure (in (b)), with its comparison to the vanilla Blow-mel in (a). The main difference is the way in computing the speaker’s embedding. In both of the graphs, the tables in green are learned during training (the speaker embedding table, with shape 108*128 in (a), and the speaker coefficient table, with shape 108*80 in (b)). The expert embedding table for (b) is extracted from pre-trained vanilla Blow-mel. | 11 |
| 4.3 | Normal convolutions vs. depthwise separable convolutions. Depthwise separable convolutions can be seen as a decomposed convolution that first combines information from the temporal dimension and then from the channel dimension. | 13 |
| 4.4 | Structure of the WN function in WaveGlow. | 15 |
| 4.5 | Structure of the WN function in SqueezeWave. | 15 |
| 5.1 | A detailed comparison on the naturalness MOS scores categorized by gender. M2M stands for Male to Male, M2F stands for Male to Female, F2M stands for Female to Male, and F2F stands for Female to Female. | 21 |
| 5.2 | A detailed comparison on the similarity MOS scores categorized by gender and confidence levels. M2M stands for Male to Male, M2F stands for Male to Female, F2M stands for Female to Male, and F2F stands for Female to Female. blow_baseline stands for the original Blow model. blow_mel_sw stands for our Blow-mel with SqueezeWave model, and blow_mel_wg stands for our Blow-mel with WaveGlow model. | 23 |

List of Tables

| | | |
|-----|--|----|
| 5.1 | A comparison of SqueezeWave and WaveGlow. SW-128L has a configuration of $L=128$, $C_g=256$, SW-128S has $L=128$, $C_g=128$, SW-64L has $L=64$, $C_g=256$, and SW-64S has $L=64$, $C_g=128$. The quality is measured by mean opinion scores (MOS). The main efficiency metric is the MACs needed to synthesize 1 second of 22kHz audio. The MAC reduction ratio is reported in the column "Ratio". The number of parameters are also reported. | 18 |
| 5.2 | Inference speeds (samples generated per second) on a Mackbook Pro and a Raspberry Pi. | 18 |
| 5.3 | A summarized comparison of the audio naturalness result of Blow, Blow-mel with WaveGlow and Blow-mel with SqueezeWave. The naturalness is measured by mean opinion scores (MOS). | 20 |
| 5.4 | A summarized comparison of the audio similarity result of Blow, Blow-mel with WaveGlow and Blow-mel with SqueezeWave. The similarity is measured by crowd-sourced responses from Amazon Mechanical Turk. Answering "Same speaker: absolutely sure" or "Same speaker: not sure" are counted as similar, while answering "Different speaker: not sure", and "Different speaker: absolutely sure" are counted as not similar. | 22 |

Acknowledgments

I would like to thank Bichen Wu and Daniel Rothchild for their advice on this project along the way, to my co-workers Tianren Gao and Bohan Zhai for their collaboration on the SqueezeWave paper, and Prof. Joseph Gonzalez and Prof. Kurt Keutzer for their support and guidance throughout my master's career.

Chapter 1

Introduction

Text-to-speech (TTS) system aims at generating a human voice based on a piece of text. An adaptive TTS system requires that in addition to generating speech based on text, the voice should also resemble that of a target speaker's. Adaptive TTS systems can enable a bunch of interesting and useful applications, especially on edge devices. Some examples may include: edge medical devices can synthesize original voices for those speech-impaired ¹; messaging apps can synthesize voice messages for the sender based on their text; nighttime reading apps can simulate a parent's voice when reading out stories for their baby and free their parent from this labor. Applications like these rely on efficient adaptive TTS algorithms.

Thanks to recent advances in deep learning, there already exist models that are related to the adaptive TTS task. Synthesizer models such as [17, 14] can predict acoustic features (such as mel-spectrograms) from pieces of text. Vocoder models such as [8, 18, 12] can generate audio waveforms from those predicted mel-spectrograms. Voice conversion models such as [13, 16] can adapt a source speaker's speech into a target speaker's. However, most of these models are designed to be complex models that can only be run in the cloud. Even if we can deploy pre-trained synthesizers (e.g. [14]) to edge devices for fast mel-spectrogram generation, we still face the computation bottleneck imposed by the latter half of the system (i.e. from mel-spectrogram to target speaker's speech). For example, [8] suffers from its slow audio generation speed; models such as [12, 16] have high MACs which far exceeds the capacity of edge processors; auto-regressive models such as [13, 18] have to run full backpropagations in order to adapt to unseen speakers – for auto-regressive models, the adaptation phase is not possible to be performed on edge devices due to memory constraint. Therefore, we need an adaptive TTS system that has the potential to adapt and infer on edge devices.

A number of trends happening nowadays suggest that moving systems such as adaptive TTS that were once cloud-based to the edge is becoming feasible and desirable. First, hardware used in mobile phones is becoming increasingly powerful, and making effective use of this computation could lead to significant reductions in cloud computing costs. Second,

¹<https://deepmind.com/blog/article/Using-WaveNet-technology-to-reunite-speech-impaired-users-with-their-original-voices>

consumers are becoming increasingly concerned about data privacy, especially concerning speech data. Smartphones, smart TVs, and home assistants have all been accused of sending sensitive data to the cloud without users' knowledge². Moving the machine learning computations to the edge would eliminate the need to send data to the cloud in the first place. Finally, consumers are becoming increasingly reliant on speech synthesis systems, to provide timely speech synthesis, to respond interactively to messages, etc. These applications must work with low latency and even without a reliable Internet connection – constraints that can only be satisfied when speech synthesis is done on-device. Responding to these trends requires moving the inference and even adaptation of TTS models to the edge. Therefore, we propose a low computational cost adaptive TTS system in this paper that can both generate audios from mel-spectrograms and convert/adapt the generated voice to a target speaker's, such that the aforementioned applications can be efficiently implemented on edge devices.

Assuming that a high-performance synthesizer (e.g. [14]) is present, we only need to convert a mel-spectrogram to a target speaker's voice. We base our system mostly on flow-based models. The main motivation is that, different from auto-regressive models, flow-based models have the potential to be trained on edge devices. Using the idea proposed in iRevNet [4], we can throw away all the forward-pass activations to save memory, and re-compute a small chunk of them as needed during backward pass. This approach can drastically decrease the memory usage during model training to $O(1)$, essentially keeping it well within the memory capacity of mobile devices.

In our proposed system, we first apply a flow-based model based on Blow to convert a source mel-spectrogram to a target mel-spectrogram. Then a carefully re-designed flow-based vocoder network based on WaveGlow can synthesize the target speaker's voice. Notably, our system is significantly smaller in terms of MACs compared with the original Blow and WaveGlow. Blow requires 53.87G MACs to convert 1 second of 16kHz speech (or 74.24G MACs if in 22kHz), and WaveGlow requires 229G MACs to generate 1 second of 22kHz speech. Our system only needs a total of 7.2G MACs (3.42G MACs for mel-spectrogram conversion, and 3.78G MACs for voice synthesis) to synthesize 1 second of speech in 22kHz, which is 42x smaller than using a concatenation of Waveglow and Blow (303G = 74.24G + 229G).

Section 4.1 of the paper will introduce the implementation of the mel-spectrogram conversion model. Section 4.2 will introduce the network optimizations based on Waveglow that can lead to significant efficiency improvements. Chapter 5 demonstrates that the proposed system, despite being extremely lightweight, can generate voices for different speakers without a significant loss to the audio quality and achieves a comparable similarity to the target speaker.

Our code, trained models, and generated samples are publicly available at <https://github.com/floraxue/low-cost-adaptive-tts>.

²<https://www.washingtonpost.com/technology/2019/05/06/alexa-has-been-eavesdropping-you-this-whole-time>

Chapter 2

Related Works

2.1 Adaptive TTS

Adaptive TTS is generally achieved either by retraining the whole vocoder network on a new speaker’s speech, or by employing separate speaker embeddings that can be quickly learned in the model. [7] introduces an adaptable TTS system with LPCNet. This system has an auto-regressive architecture, and is therefore not able to be trained on edge devices as explained in the Introduction. In addition, when adapting to new speakers, the network needs to be retrained on the new speaker’s voice using at least 5 minutes of audio samples, or 20 minutes of audio samples for optimal performance. Comparing to other works such as [1] which only uses several seconds of audio, the system is not very efficient in terms of adaptation. [1] learns an unseen speaker’s characteristics with only a few samples by introducing a model adaptation phase. While the idea of adding an adaptation phase is interesting and can be applied in our system, its results are still based on WaveNet [8], which means that its audio generation is slower than real-time.

2.2 Vocoder Models

In 2016, Oord et al. [8] proposed WaveNet, which achieves human-like audio synthesis performance. However, as we mentioned above, its slow synthesis speed (slower than real-time) makes it inefficient for online speech synthesis. Its successors such as WaveRNN [5] and LPCNet [18] can synthesize with much faster than real-time (on a GPU). In addition, the LPCNet [18] is also a lightweight model that can be run on mobile devices. Although WaveRNN and LPCNet have the above benefits, a major drawback of both models is that they are auto-regressive, making it impossible to adapt to new speakers on edge devices. Non-auto-regressive models are thus the best directions for us to build upon. Among these non-auto-regressive ones, Parallel WaveNet [9] and Clarinet [10] are harder to train and implement than the autoregressive ones due to their complex loss functions and the teacher-student network architecture, according to the argument from [12]. To the best of our knowledge,

Waveglow [12] is the current state-of-the-art vocoder that uses a fully feed-forward architecture to generate high-quality voices.

2.3 Voice Conversion

Voice Conversion, different from adaptive TTS, solves the audio to audio conversion problem instead of the text to audio generation problem. Works from this field is still highly related with adaptive TTS since the core problem of converting between identities is common. Recently, AutoVC [13] and Blow [16] proposed approaches to perform voice conversion with high audio quality and similarity to target speaker. While AutoVC is able to perform zero-shot and high-quality voice conversion to unseen speakers, it is still an auto-regressive model, which hinders its training on edge devices. On the other end of the spectrum, Blow is a promising model for edge devices deployment since its fully convolutional architecture. However, its huge computational cost (measured in term of MACs) makes it impossible to run inference on edge devices.

Chapter 3

Preliminaries

3.1 Flow-based Models

Flow-based models are first proposed in Glow [6]. Different from other generative models, flow-based models directly model the data distribution $p(x)$. While $p(x)$ is normally intractable in other generative models, flow-based models can still model it through its architectural design. These models can learn a series of invertible transformations that bijectively turn x from the data distribution into a latent variable z , where z is from a Gaussian distribution. During inference, the model draws a Gaussian sample and transforms it back to the data distribution.

Waveglow

WaveGlow is a flow-based model that generates an audio waveform conditioned on a mel-spectrogram. The architecture is similar to that of Glow [6], with changes introduced for speech synthesis. Its general architecture is explained in detail below, since the efficiency improvement introduced in Section 4.2 requires a detailed analysis of its original architecture.

Instead of convolving the waveforms directly, WaveGlow first groups nearby samples to form a multi-channel input $x \in \mathbf{R}^{L, C_g}$, where L is the length of the temporal dimension and C_g is the number of grouped audio samples per time step (The number of samples in the waveform is just $L \times C_g$). This grouped waveform \mathbf{x} is then transformed by a series of bijections, each of which takes $\mathbf{x}^{(i)}$ as input and produces $\mathbf{x}^{(i+1)}$ as output. Within each bijection, the input signal $\mathbf{x}^{(i)}$ is first processed by an invertible point-wise convolution, and the result is split along the channel dimension into $\mathbf{x}_a^{(i)}, \mathbf{x}_b^{(i)} \in \mathbf{R}^{L, C_g/2}$. $\mathbf{x}_a^{(i)}$ is then used to compute affine coupling coefficients $(\log \mathbf{s}^{(i)}, \mathbf{t}^{(i)}) = \text{WN}(\mathbf{x}_a^{(i)}, \mathbf{m})$. $\mathbf{s}^{(i)}, \mathbf{t}^{(i)} \in \mathbf{R}^{L, C_g/2}$ are the affine coupling coefficients that will be applied to $\mathbf{x}_b^{(i)}$, $\text{WN}(\cdot, \cdot)$ is a WaveNet-like function, or WN function for short, $\mathbf{m} \in \mathbf{R}^{L_m, C_m}$ is the mel-spectrogram that encodes the audio, L_m is the temporal length of the mel-spectrogram and C_m is the number of frequency components. Next, the affine coupling layer is applied: $\mathbf{x}_b^{(i+1)} = \mathbf{x}_b^{(i)} \otimes \mathbf{s}^{(i)} + \mathbf{t}^{(i)}$, $\mathbf{x}_a^{(i+1)} = \mathbf{x}_a^{(i)}$, where

\otimes denotes element-wise multiplication. Finally, $\mathbf{x}_a^{(i)}$ and $\mathbf{x}_b^{(i+1)}$ are concatenated along the channel dimension.

The majority of the computation of WaveGlow is in the WN functions $WN(\cdot, \cdot)$, illustrated in Figure 4.4. The first input to the function is processed by a point-wise convolution labeled *start*. This convolution increases the number of channels of $\mathbf{x}_a^{(i)}$ from $C_g/2$ to a much larger number. In WaveGlow, $C_g = 8$, and the output channel size of *start* is 256. Next, the output is processed by a dilated 1D convolution with a kernel size of 3 named *in_layer*. Meanwhile, the mel-spectrogram \mathbf{m} is also fed into the function. The temporal length of the mel-spectrogram L_m is typically much smaller than the length of the reshaped audio waveform L . In WaveGlow, $L_m = 63, C_m = 80, L = 2,000, C_g = 8$. So in order to match the temporal dimension, WaveGlow upsamples \mathbf{m} , and then passes it through a convolution layer named *cond_layer*. The output of *in_layer* and *cond_layer* are combined in the same way as WaveNet [8] through the *gate* function, whose output is then processed by a *res_skip_layer*. The output of this layer has a temporal length of $L = 2000$ and a channel size of 512 in the original WaveGlow. It is then split into two branches along the channel dimension. This structure is repeated 8 times and at the last one, the output of *res_skip_layer* is then processed by a point-wise convolution named *end*. This convolution computes the transformation factors $\mathbf{s}^{(i)}$ and $\mathbf{t}^{(i)}$ and compresses the channel size from 512 to $C_g = 8$.

Blow

Blow is also a flow-based model with a similar invertible architecture as WaveGlow. The model generates audio waveform conditioned on speaker embeddings learned during training. The model is largely based on Glow [6], but it introduced several changes that are critical to voice conversion: 1) it uses a single scale architecture, i.e. the intermediate latent representations are kept to the same dimension throughout the flows; 2) it organizes 12 flows into a block, and increases the number of blocks (8 blocks in its architecture) to create a deep architecture, therefore increasing the model’s receptive field; 3) it models the latent space z as a speaker-independent space, such that the speaker traits are all stored in the speaker embeddings; 4) it uses hyperconditioning to convolve the speaker’s identity with input waveform; 5) the speaker embeddings are shared across the model, and 6) it uses data augmentation to improve the performance. Our mel-spectrogram conversion model introduced in Section 4.1 inherits these changes.

3.2 Computational Complexity of Blow and Waveglow

According to the source code of Blow and WaveGlow, we calculate the computational cost of the two models. The details of the calculation can be found in our source code. To generate 1 second of 16kHz audio, Blow requires 53.87G MACs, or 74.24G MACs if the audio is sampled at 22kHz. To generate 1 second of 22kHz audio, WaveGlow requires 229G MACs.

In WaveGlow, among all the layers, *in_layers* accounts for 47%, *cond_layers* accounts for 39%, and *res_skip_layer* accounts for 14%.

Chapter 4

Methodology

4.1 Flow-based Lightweight Mel-spectrogram Conversion: Blow-Mel

Overview

The lightweight mel-spectrogram conversion model is similar to [16], but some improvements are introduced to make the model work efficiently with mel-spectrograms. A schematic of the model is plotted in Figure 4.2.

The input of the model is a mel-spectrogram x and a speaker id y . The mel-spectrogram is fed into several steps of flow to get a latent variable z . The steps of flow are a series of invertible transformations that transform the input mel-spectrogram from its data distribution into a latent Gaussian space represented by z . There are 96 steps of Flows in total in this architecture. Within each step of Flow, the input features first go through a convolutional layer with kernel width 1 to mix the input channels. Under such kernel width, this convolutional layer is inherently invertible. Then the convolutional output goes through an ActNorm layer. The ActNorm output is split into two halves, x_a and x_b , in its channel dimension. x_b is used as the input to the Coupling Net, and the output is again equally divided in channel to be a scaler s and a shifter t . The scaler and the shifter are used to perform affine transformation on x_a , such that $x'_a = s * (x_a + t)$. The final output of the Flow is a channel concatenation of the transformed x'_a and the original x_b . Note that even if the Coupling Net is not invertible, the Flow is still invertible: with knowledge of the Flow's output, the original value of x_a can be easily recovered after another forward pass on the Coupling Net.

The speaker's identity is integrated with the input mel-spectrogram within the Coupling Net. The speaker id y selects the corresponding speaker embedding in the embedding table. The embedding table is randomly initialized and learned during training. The speaker embedding goes through an Adapter, which is a fully connected layer. The produced vector becomes weights in the hyperconvolution layer (kernel width 3). The hyperconvolution can

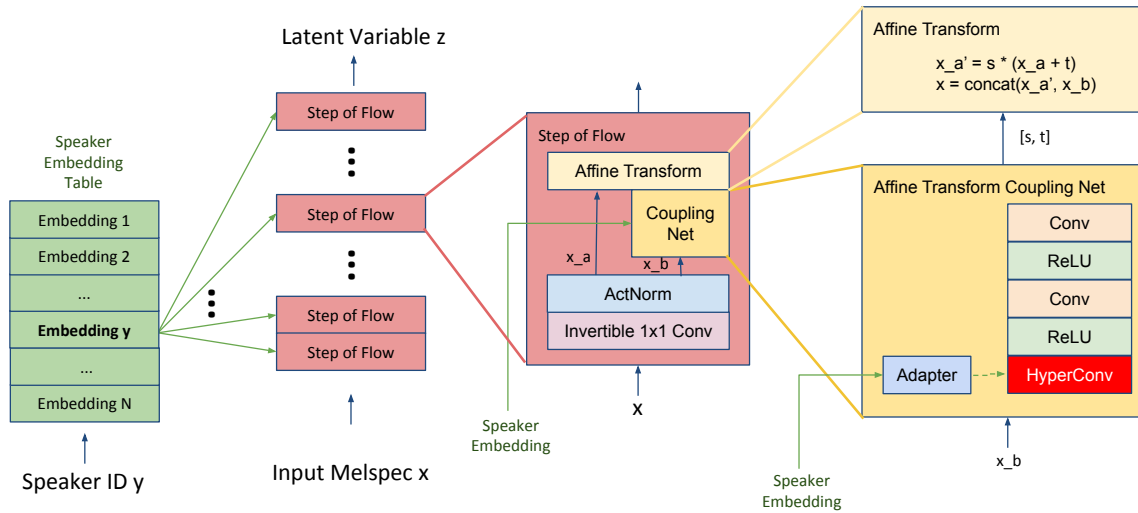


Figure 4.1: Overview of the Blow-mel model.

therefore convolve the speaker identity information with the input features. The output of hyperconvolution then goes through two convolutional layers, where the first one is 480 to 480 channels with a kernel width 1, and the second one is 480 to 160 channels with a kernel width 3.

The model is trained as a common flow-based model as introduced in [6] with a log likelihood that consists of two parts: the first part compares the z against a unit Gaussian, and the second part is the summed log determinant of each layer.

Increasing input frame size for a larger receptive field

A common issue in flow-based models applied in the speech field is to increase the receptive field size. This problem is less prominent in this mel-spectrogram conversion model, since mel-spectrogram is already a condensed representation of audio waveform, but it still exists. According to the original setup of Blow [16], the input frame size is 4096, which corresponds to around 256ms of audio at 16kHz. However, after the STFT operation with window size 256 to get mel-spectrograms, the input size becomes 16, which is too small and creates issue for deep network training. Therefore, we increased the input frame size to 16384 samples on 22kHz audio, which is roughly 743ms. This makes the receptive field much larger, and the model can thus learn the relationship between more phonemes (which are around 50ms to

180ms).

Removing squeeze operation between blocks

While the original Blow model proposes to squeeze the input into a progression of 2, 4, 8, ... 256 channels before passing into each block, the mel-spectrogram conversion model does not need a squeeze operation between blocks at all. The main reason is that it does not need to increase the receptive field and fold the audio for better training. Since the mel-spectrogram is extracted as an 80 channel input, the channel size between blocks is fixed at 80.

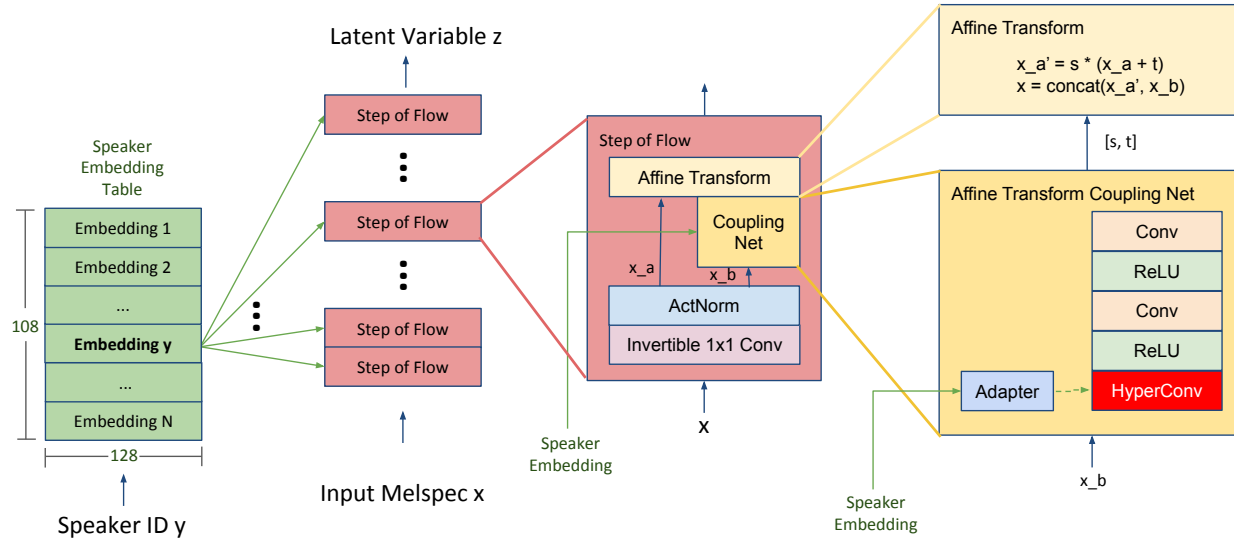
Following the removal of squeeze operations, the latent dimensions between Flows are fixed. Therefore, it is not necessary to use a different output channel size for different Coupling Nets, and the channel size is fixed at 160 to accommodate the mel-spectrogram channel size.

Learning speaker embeddings by using a dynamic combination of expert embeddings: Blow-mel-coeff

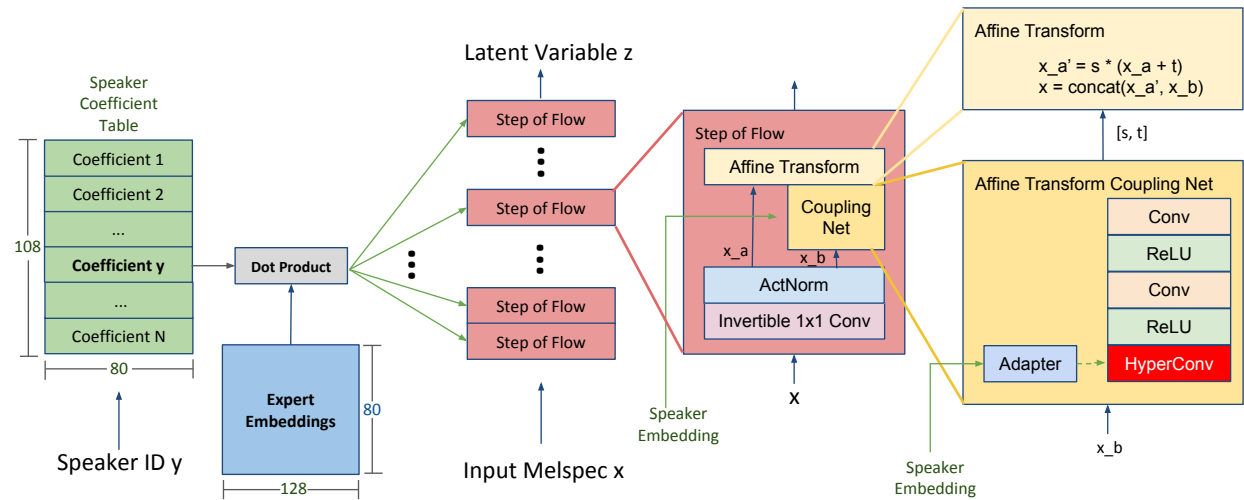
To promote the learning of shared attributes between certain speakers while also preserving the ability to learn distinct speaker embeddings, we also experimented with a novel way to dynamically combine expert embeddings to create an embedding for a given speaker.

To achieve this, we first extract all the learned embeddings after training the model on the full dataset. From this embedding table, we run principle component analysis and compute a number of principle components from the embeddings. The principle components are stored as expert embeddings in the subsequent training and inference stage. During the next training stage, each speaker from the training set corresponds to a vector of coefficients. The coefficients can be used to linearly combine expert embeddings (i.e. the principle components). These coefficients are randomly initialized and learned during training. The inference step is similar to the original one, where a forward pass is performed with a source mel-spectrogram and source coefficient to get the latent vector z , and the z flows reversely in the model with the target coefficient to generate target mel-spectrogram.

A schematic of the coefficient version of the model is plotted in Figure 4.2.



(a) The vanilla Blow-mel structure



(b) The coefficient version of the Blow-mel structure

Figure 4.2: Structure of the coefficient version of the Blow-mel structure (in (b)), with its comparison to the vanilla Blow-mel in (a). The main difference is the way in computing the speaker’s embedding. In both of the graphs, the tables in green are learned during training (the speaker embedding table, with shape 108*128 in (a), and the speaker coefficient table, with shape 108*80 in (b)). The expert embedding table for (b) is extracted from pre-trained vanilla Blow-mel.

4.2 Flow-based Lightweight Vocoder: SqueezeWave

Reshaping audio waveforms

After carefully examining the network structure of WaveGlow, we identified that a major source of the redundancy comes from the shape of the input audio waveform to the network. In the original WaveGlow, the input waveform is reshaped to have a large temporal dimension and small channel size ($L = 2000, C_g = 8$). This leads to high computational complexity in three ways: 1) WaveGlow is a 1D convolutional neural network, and its computational complexity is linear in L . 2) Mel-spectrograms have a much coarser temporal resolution than the grouped audio: in the original WaveGlow, $L = 2000$ but $L_m = 63$. In order to match the temporal dimensions of the two signals, WaveGlow upsamples the mel-spectrogram before passing it through *cond.layers*. The upsampled mel-spectrograms are highly redundant since new samples are simply interpolated from existing ones. Therefore, in WaveGlow, most of the computations in *cond.layers* are not necessary. 3) Inside each WN function, the 8-channel input is projected to have a large intermediate channel size, typically 256 or 512. A larger channel size is beneficial since it increases the model capacity. However, at the output of WN, the channel size is compressed to $C_g = 8$ to match the audio shape. Such drastic reduction creates an “information bottleneck” in the network and information encoded in the intermediate representation can be lost.

To fix this, we simply re-shape the input audio \mathbf{x} to have a smaller temporal length and a larger channel size, while keeping the internal channel sizes within the WN function the same. In our experiments, we implement two settings: $L = 64, C_g = 256$ or $L = 128, C_g = 128$. (The total number of samples are changed from 16,000 to 16,384.) When $L = 64$, the temporal length is the same as the mel-spectrogram, so no upsampling is needed. When $L = 128$, we change the order of operators to first apply *cond.layer* on the the mel-spectrogram and then apply nearest-neighbor upsampling. This way, we can further reduce the computational cost of the *cond.layers*.

Depthwise convolutions

Next, we replace 1D convolutions in the *in_layer* with depthwise separable convolutions. Depthwise separable convolutions are popularized by [2] and are widely used in efficient computer vision models, including [15, 20]. In this work we adopt depthwise separable convolutions to process 1D audio.

To illustrate the benefits of depthwise separable convolutions, consider a 1D convolutional layer that transforms an input with shape $C_{in} \times L_{in}$ into an output with shape $C_{out} \times L_{out}$, where C and L are the number of channels and temporal length of the signal, respectively. For a kernel size K , the kernel has shape $K \times C_{in} \times C_{out}$, so the convolution costs $K \times C_{in} \times C_{out} \times L_{out}$ MACs. A normal 1D convolution combines information in the temporal and channel dimensions in one convolution with the kernel. The depthwise separable convolution decomposes this functionality into two separate steps: (1) a temporal combining layer and

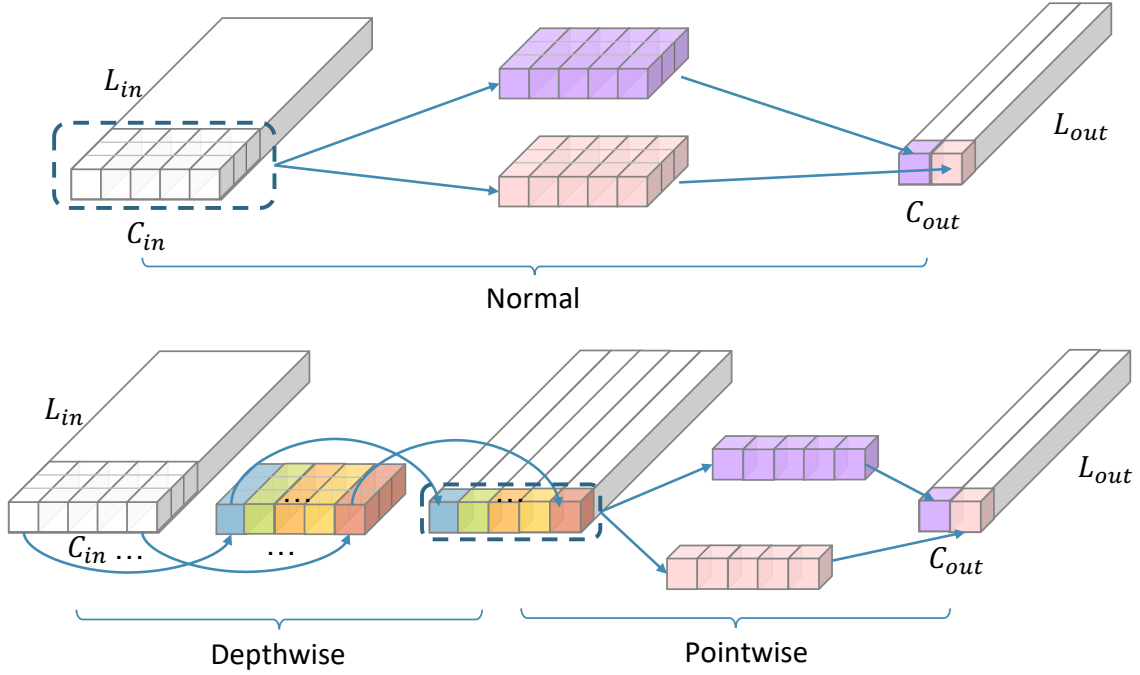


Figure 4.3: Normal convolutions vs. depthwise separable convolutions. Depthwise separable convolutions can be seen as a decomposed convolution that first combines information from the temporal dimension and then from the channel dimension.

(2) a channel-wise combining layer with a kernel of size 1. Step 1 is called a depthwise convolution, and step 2 is called a pointwise convolution. The difference between a normal 1D convolution and a 1D depthwise separable convolution is illustrated in Figure 4.3. After applying the depthwise separable convolution, the computational cost for step-1 becomes $K \times C_{in} \times L_{in}$ MACs and for step-2, $C_{in} \times C_{out} \times L_{in}$. The reduction of computation is therefore

$$\frac{C_{in} \times C_{out} \times L_{in} + K \times C_{in} \times L_{in}}{K \times C_{in} \times C_{out} \times L_{in}} = \frac{1}{C_{out}} + \frac{1}{K}.$$

In our setup, $K = 3$ and $C_{out} = 512$, so using this technique leads to around 3x MAC reduction in the *in_layers*.

Other improvements

In addition to the above two techniques, we also make several other improvements: 1) since the temporal length is now much smaller, WN functions no longer need to use dilated convolutions to increase the receptive fields, so we replace all the dilated convolutions with regular convolutions, which are more hardware friendly; 2) Figure 4.4 shows that the outputs of the *res_skip_layers* are split into two branches. Hypothesizing that such a split is not

necessary since the topologies of the two branches are almost identical, we merge them into one and reduce the output channel size of the *res_skip_layers* by half. The improved SqueezeWave structure is illustrated in Figure 4.5.

Chapter 5

Evaluation

The evaluation is divided into two sections. In the first section, we compare the lightweight vocoder SqueezeWave with WaveGlow in terms of efficiency and audio quality. In the second section, we compare the mel-spectrogram converter Blow-mel with Blow in terms of efficiency, audio quality and similarity.

5.1 Vocoder

Experimental Setup

Our experimental setup is similar to that of [12]: we use the LJSpeech dataset [3], which has 13,100 paired text/audio examples. We use a sampling rate of 22050Hz for the audio. We extract mel-spectrograms with librosa, using an FFT size of 1024, hop size 256 and window size 1024. We split the dataset into a training and a test set, and the split policy is provided in our source code. We reproduce the original WaveGlow model by training from scratch on 8 Nvidia V100 32GB RAM GPUs with a batch size 24. We train our lightweight vocoder with 24GB-RAM Titan RTX GPUs using a batch size of 96 for 600k iterations. Detailed configurations are available in our code. Table 5.1 summarizes the comparison in terms of audio quality and efficiency of the two models.

Results

We consider three metrics of computational efficiency: 1) MACs required per second of generated audio, 2) number of model parameters, and 3) actual speech generation speed, in generated samples per second, on a Macbook Pro and a Raspberry Pi 3b+.

In terms of the audio quality, we use Mean Opinion Score (MOS) as the metric as in [17, 8, 12, 11]. We crowd-source our MOS evaluation on Amazon Mechanical Turk. We use 10 fixed sentences for each system, and each system/sentence pair is rated by 100 raters. Raters are not allowed to rate the same sentence twice, but they are allowed to rate another sentence from the same or a different system. We reject ratings that do not pass a hidden

quality assurance test (ground truth vs. obviously unnatural audio). We report MOS scores with 95% confidence intervals.

According to the results table 5.1, WaveGlow achieves MOS scores comparable to those for ground-truth audio. However, the computational cost of WaveGlow is extremely high, as it requires 228.9 GMACs to synthesize 1 second of 22kHz audio. SqueezeWave models are much more efficient. The largest model, SW-128L, with a configuration of $L=128$, $C_g=256$ requires 61x fewer MACs than WaveGlow. With reduced temporal length or channel size, SW-64L (106x fewer MACs) and SW-128S (214x fewer MACs) achieves slightly lower MOS scores but significantly lower MACs. Quantitatively, MOS scores of the SqueezeWave models are lower than WaveGlow, but qualitatively, their sound qualities are similar, except that audio generated by SqueezeWave contains some background noise. Noise cancelling techniques can be applied to improve the quality. Readers can find synthesized audio of all the models from our source code. We also train an extremely small model, SW-64S, with $L=64$, $C_g=128$. The model only requires 0.69 GMACs, which is 332x fewer than WaveGlow. However, the sound quality is obviously lower, as reflected in its MOS score.

We deploy WaveGlow and SqueezeWave to a Macbook Pro with an Intel i7 CPU and a Raspberry Pi 3B+ with a Broadcom BCM2837B0 CPU. We report the number of samples generated per second by each model in Table 5.2. On a Macbook, SqueezeWave can reach a sample rate of 123K-303K, 30-72x faster than WaveGlow, or 5.6-13.8x faster than real-time (22kHz). On a Raspberry Pi computer, WaveGlow fails to run, but SqueezeWave can still reach 5.2k-21K samples per second. SW-128S in particular can reach near real-time speed while maintaining good quality.

| Models | MOS | GMACs | Ratio | Params |
|----------|-----------------|-------|-------|--------|
| GT | 4.62 ± 0.04 | – | – | – |
| WaveGlow | 4.57 ± 0.04 | 228.9 | 1 | 87.7 M |
| SW-128L | 4.07 ± 0.06 | 3.78 | 61 | 23.6 M |
| SW-128S | 3.79 ± 0.05 | 1.07 | 214 | 7.1 M |
| SW-64L | 3.77 ± 0.05 | 2.16 | 106 | 24.6 M |
| SW-64S | 2.74 ± 0.04 | 0.69 | 332 | 8.8 M |

Table 5.1: A comparison of SqueezeWave and WaveGlow. SW-128L has a configuration of $L=128$, $C_g=256$, SW-128S has $L=128$, $C_g=128$, SW-64L has $L=64$, $C_g=256$, and SW-64S has $L=64$, $C_g=128$. The quality is measured by mean opinion scores (MOS). The main efficiency metric is the MACs needed to synthesize 1 second of 22kHz audio. The MAC reduction ratio is reported in the column “Ratio”. The number of parameters are also reported.

| Models | Macbook Pro | Raspberry Pi |
|----------|-------------|--------------|
| WaveGlow | 4.2K | Failed |
| SW-128L | 123K | 5.2K |
| SW-128S | 303K | 15.6K |
| SW-64L | 255K | 9.0K |
| SW-64S | 533K | 21K |

Table 5.2: Inference speeds (samples generated per second) on a Mackbook Pro and a Raspberry Pi.

5.2 Mel-spectrogram Adaptation

Experimental Setup

We use a similar setup as in [16] to run our experiments. We use the VCTK [vctk] dataset, which contains 46 hours of audio spoken by 108 speakers ¹. Each speaker speaks a subset of all the sentences, where different subsets have intersections. We downsample the dataset from 48kHz to 22kHz for training our model such that it agrees with the vocoders. When reproducing Blow, we downsample the dataset to 16kHz following their setup. The dataset is randomly split into training, validation and testing set using a 8:1:1 ratio. In the splitting script, we only split on the utterances, not the speakers, meaning that all of the speakers are present in each split. In addition, we follow Blow to ensure that the same sentence do not appear in different splits, so that data leakage is prevented.

Since the model takes mel-spectrograms as inputs, we use the same mel-spectrogram extraction process as in Section 5.1. We train our model for three days on 4 Nvidia P100 GPUs, with a batch size of 1024, an Adam optimizer and an initial learning rate of $1e - 4$. We employ a learning rate annealing policy: if the model’s validation loss stops improving for 10 consecutive epochs, the learning rate will be timed by 0.2. If the learning rate annealing happens twice, the training is stopped.

The trained model is used to convert mel-spectrogram of a source speech to a target mel-spectrogram. The conversion is performed using the test set utterances as source speech, and the target speakers are randomly selected from the 108 speakers. We perform the conversion between all possible gender combinations.

The output mel-spectrogram is then plugged into downstream vocoders to transform into an audio waveform. We use our lightweight vocoder, which is trained as described in Section 5.1. As a way to perform abelation, we also present the audio generation result using a pretrained WaveGlow, which is available at its code repository ². Note that both of the pretrained vocoders are trained from the single speaker LJSpeech dataset.

As a baseline, we also reproduce Blow by following the experimental setup described in its paper. It is trained using the same dataset for 15 days on three GeForce RTX 2080-Ti GPUs. Since Blow is a voice to voice conversion model, its output will be directly used for comparison. Since the data set is the same our model’s, its voice conversion is performed on the same source sentences and the same set of randomly selected target speakers.

Audio Naturalness Results

To evaluate the audio naturalness, or the amount of artifacts and distortions in the audio, we use MOS scores with scale from 1 to 5 as defined in 5.1. Since the test set contains 4412 utterances, we only sample 100 random utterance from this set to get faster MOS rating results. For each Mechanical Turk worker, they see a random 10 utterances from the 100.

¹The dataset claims to contain 109 speakers, but the person "p315" does not have any utterances.

²<https://github.com/NVIDIA/waveglow>

We collect 500 responses from the workers, so that each of the 100 audios are rated for 50 times on average. Raters are only allowed to rate the same system once (i.e. only rate 10 different sentences) to ensure the diversity of raters on the same system. Quality assurance measures similar to 5.1 are implemented to ensure the quality of the responses. Invalid responses are rejected and excluded from the results. We also report the 95% confidence interval on the MOS scores.

| Models | MOS |
|---------------------------|-----------------|
| Blow | 2.89 ± 0.03 |
| Blow-mel with WaveGlow | 2.76 ± 0.03 |
| Blow-mel with SqueezeWave | 2.33 ± 0.03 |

Table 5.3: A summarized comparison of the audio naturalness result of Blow, Blow-mel with WaveGlow and Blow-mel with SqueezeWave. The naturalness is measured by mean opinion scores (MOS).

Table 5.3 shows a comparison of the baseline Blow model, our lightweight mel-spectrogram conversion model (Blow-mel) with two different downstream vocoder models (SqueezeWave introduced in Section 4.2 and WaveGlow). In general, the lightweight mel-spectrogram conversion model does not introduce a significant audio quality degradation.

We also summarize the MOS score results by gender categories in Figure 5.1 (i.e. Male to Male, Male to Female, Female to Male, Female to Female). As we can see from the graph, when the target speaker is a male, the results for our models (Blow-mel with SqueezeWave and Blow-mel with WaveGlow) are significantly lower than that for Blow. This is likely because of the fact that SqueezeWave and WaveGlow are both pre-trained on LJSpeech, which contains a single female speaker. Therefore, even if the model can be directly deployed to generate male’s voices, the quality may not be desirable. This issue can potentially be resolved by pre-training both models on the VCTK dataset, but we will leave this for further exploration due to time constraint. When the target speaker is a female, our models’ MOS scores are comparable to that of Blow’s, and Blow-mel with WaveGlow and even outperform Blow. This result suggests that our Blow-mel model itself potentially does not introduce any performance degradation, but only improvement (e.g. improved ability to model phoneme transitions due to the increased receptive field). When converting across genders, all three of the systems are performing worse than between genders. This trend is commonly found in most voice conversion systems and is considered normal due to the larger gap between female and male acoustic features (such as pitches).

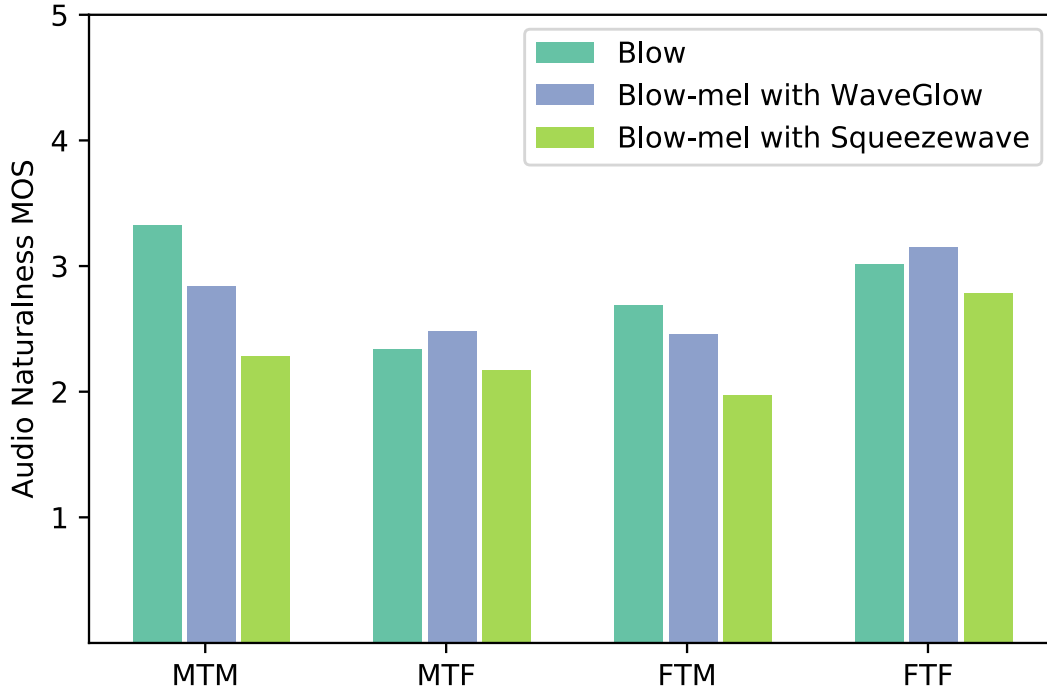


Figure 5.1: A detailed comparison on the naturalness MOS scores categorized by gender. M2M stands for Male to Male, M2F stands for Male to Female, F2M stands for Female to Male, and F2F stands for Female to Female.

Similarity Results

In order to properly evaluate the voice conversion quality, we also need to verify the model’s ability to generate a speech that is similar to the target speaker’s voice. We use crowd-sourced feedback from Amazon Mechanical Turk to evaluate the similarity. Our setup is similar to that of [16], which is based on [19]. We use the same 100 utterances as in the audio naturalness evaluation, and each Mechanical Turk worker see 10 random utterances from this set. Each utterance is placed next to its corresponding target speaker’s real speech recording, forming a pair of voice conversion audio and target audio. For each pair of recordings, the worker is asked to answer if or not these two recordings are possibly from the same speaker. Note that the worker cannot see which one is a generated voice conversion audio, and which one is a real speech recording. The worker is also asked to ignore possible distortions or artifacts in the audios. The worker can choose an answer from the following 4 choices: "Same speaker: absolutely sure", "Same speaker: not sure", "Different speaker:

not sure”, and ”Different speaker: absolutely sure”.

We collect 500 responses from the workers, with each response containing 10 pairs of ratings. So each pair of audios receive on average 50 ratings. Raters are not allowed to submit more than one response for each system. We also employ quality assurance measures on these responses. In addition to the 10 pairs of audios, a worker also needs to answer 4 additional pairs of hidden tests without knowing they are tests. 2 pairs of the tests are ”ground truth tests”, where each pair consists of two exact audios. 2 pairs of the tests are ”negative tests”, where each pair consists of a female’s real speech recording and a male’s real speech recording. If a worker choose any of the two ”different” for the ground truth tests, or any of the two ”same” for negative tests, the response is considered invalid. Invalid responses are rejected and excluded from the reported results.

| Models | Similarity to Target | Num Valid Raters |
|---------------------------|----------------------|------------------|
| Blow | 40.16% | 430 |
| Blow-mel with WaveGlow | 48.63% | 430 |
| Blow-mel with SqueezeWave | 40.94% | 427 |

Table 5.4: A summarized comparison of the audio similarity result of Blow, Blow-mel with WaveGlow and Blow-mel with SqueezeWave. The similarity is measured by crowd-sourced responses from Amazon Mechanical Turk. Answering ”Same speaker: absolutely sure” or ”Same speaker: not sure” are counted as similar, while answering ”Different speaker: not sure”, and ”Different speaker: absolutely sure” are counted as not similar.

Table 5.4 summarizes the similarity result. We can see that Blow-mel with both vocoders can achieve a comparable result as with the original Blow. This indicates that the reduction in computational complexity does not introduce degradation in terms of similarity.

Note that the similarity score from Blow is significantly lower than that reported in its paper. Given that we use the same question wording (i.e. instructions) and question setup (i.e. pairing converted audios with real target speech) for each rater, we deduct this is likely because of the difference in selecting utterances to rate and our rater groups. Blow reports that it uses 4 utterances selected from the 4412 test set to rate each system under comparison. The criteria for selecting the 4 utterances is not known. However, we randomly selected 100 utterances from the 4412 test set to rate each system. Our utterances set used for rating is more random and diverse, so we believe our results have more statistical significance than Blow’s. In addition, within Blow’s rater group, 8 out of the 33 raters have speech processing expertise, so it is possible that some of the raters may already know the expected performance when rating the audio pairs. However, our raters group of 500 people can be considered as a random sample from all Mechanical Turk workers, so they may not

have an expectation on the performance of the systems. We believe that our group of raters may better represent the general public, and our results are thus more plausible.

We also present a more detailed comparison with more granularity in Figure 5.2. We categorize the utterances into four categories, Male to Male, Male to Female, Female to Male, and Female to Female. We also present the degree of the workers’ confidence in these ratings. From the graph we can see that Blow-mel with WaveGlow outperforms Blow in all gender categories. In addition, Blow-mel with SqueezeWave shows a lower similarity when target speakers are males, which means that the audio quality might have adversely impacted the similarity rating. Even if we have clearly instructed the raters to ignore artifacts and distortions, the audio quality degradation may have influenced the worker’s ability to identify speaker characteristics in the audio. With a SqueezeWave pretrained on VCTK for better audio quality, it is possible that the similarity can be further improved.

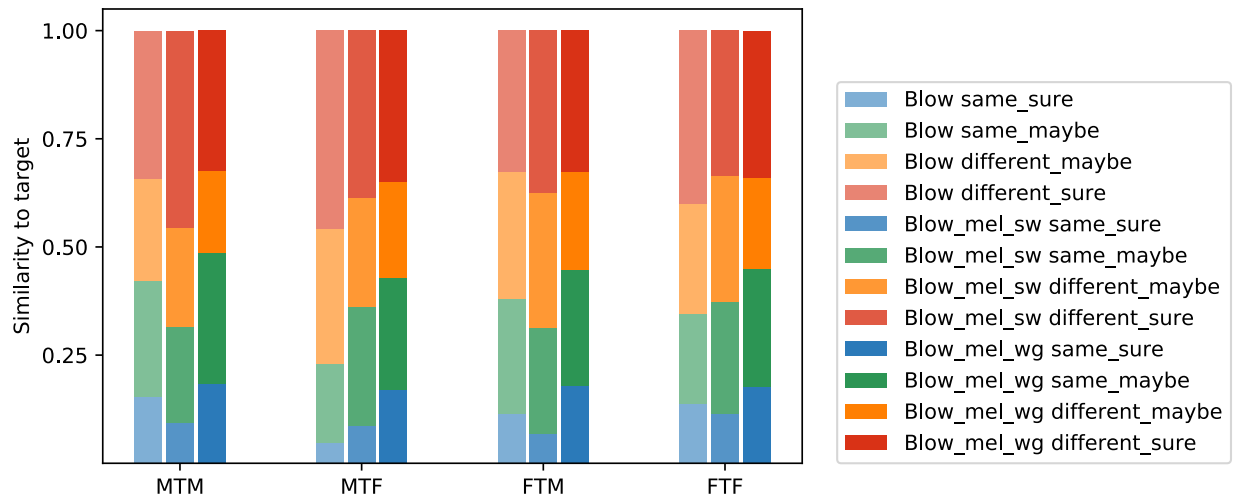


Figure 5.2: A detailed comparison on the similarity MOS scores categorized by gender and confidence levels. M2M stands for Male to Male, M2F stands for Male to Female, F2M stands for Female to Male, and F2F stands for Female to Female. blow_baseline stands for the original Blow model. blow_mel_sw stands for our Blow-mel with SqueezeWave model, and blow_mel_wg stands for our Blow-mel with WaveGlow model.

Results for Blow-mel with coefficient (Blow-mel-coeff)

We also run the adaptive TTS system with Blow-mel-coeff to convert mel-spectrograms and SqueezeWave/WaveGlow to generate audio waveform. However, the generated audios does not demonstrate that this method is effective enough in terms of generalizing to different speakers. With the principle components extracted from the 108 speaker embeddings, there are 2 speakers whose embedding cannot be spanned by a linear combination by the 108 speakers. This causes the generated audios to be silence for the two speakers, while all the other speakers can still generate legitimate outputs. To verify that the embedding space is diverse and therefore cannot be easily collapsed to lower dimensions, we also explore the percentage of the explained variance out of the total variance for each singular value during PCA. We find that the first singular value only explains 3.3% of the total variance. Overall, there are only 3 singular values that can each explain more than 3% of the total variance. If we look at the number of singular values that can each explain more than 1% of the total variance, there are still only 40. Although the Blow-mel-coefficient model is not feasible, the above findings indicate that the embedding space learned in the pre-trained Blow-mel is considerably different for each speaker.

Chapter 6

Conclusion

In this thesis, propose an adaptive TTS system that can be used for training and inference on edge devices where the memory and computational power is limited. We base our system on flow-based models to bypass the memory constraint, and re-design model architectures to compress the model to a fewer MACs. We evaluate our low-cost system both separately for audio generation (SqueezeWave vs WaveGlow) and mel-spectrogram conversion (Blow-mel with WaveGlow vs Blow), and together for mel-spectrogram to target speech generation (Blow-mel with SqueezeWave vs Blow). We demonstrate that our proposed system can generate speech with comparable similarity to the baseline model and no significant loss to the audio quality. As future work, we could potentially improve the audio quality by pretraining the vocoder on a multispeaker dataset, or exploring new ways of learning shared characteristics from speaker embeddings. We would also extend the system by employing an adaptation phase to learn the speaker embedding for unseen speakers, and finally deploying the system onto edge devices.

Bibliography

- [1] Yutian Chen et al. *Sample Efficient Adaptive Text-to-Speech*. 2018. arXiv: 1809.10460 [cs.LG].
- [2] Andrew G Howard et al. “Mobilenets: Efficient convolutional neural networks for mobile vision applications”. In: *arXiv preprint arXiv:1704.04861* (2017).
- [3] Keith Ito. *The LJ Speech Dataset*. <https://keithito.com/LJ-Speech-Dataset/>. 2017.
- [4] Jörn-Henrik Jacobsen, Arnold Smeulders, and Edouard Oyallon. *i-RevNet: Deep Invertible Networks*. 2018. arXiv: 1802.07088 [cs.LG].
- [5] Nal Kalchbrenner et al. “Efficient neural audio synthesis”. In: *arXiv preprint arXiv:1802.08435* (2018).
- [6] Diederik P. Kingma and Prafulla Dhariwal. *Glow: Generative Flow with Invertible 1x1 Convolutions*. 2018. arXiv: 1807.03039 [stat.ML].
- [7] Zvi Kons et al. “High quality, lightweight and adaptable TTS using LPCNet”. In: *arXiv preprint arXiv:1905.00590* (2019).
- [8] Aaron van den Oord et al. “Wavenet: A generative model for raw audio”. In: *arXiv preprint arXiv:1609.03499* (2016).
- [9] Aaron van den Oord et al. *Parallel WaveNet: Fast High-Fidelity Speech Synthesis*. 2017. arXiv: 1711.10433 [cs.LG].
- [10] Wei Ping, Kainan Peng, and Jitong Chen. *ClariNet: Parallel Wave Generation in End-to-End Text-to-Speech*. 2018. arXiv: 1807.07281 [cs.CL].
- [11] Wei Ping et al. “Deep Voice 3: 2000-Speaker Neural Text-to-Speech”. In: *International Conference on Learning Representations*. 2018. URL: <https://openreview.net/forum?id=HJtEm4p6Z>.
- [12] Ryan Prenger, Rafael Valle, and Bryan Catanzaro. “Waveglow: A flow-based generative network for speech synthesis”. In: *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2019, pp. 3617–3621.
- [13] Kaizhi Qian et al. *AUTOVC: Zero-Shot Voice Style Transfer with Only Autoencoder Loss*. 2019. arXiv: 1905.05879 [eess.AS].
- [14] Yi Ren et al. *FastSpeech: Fast, Robust and Controllable Text to Speech*. 2019. arXiv: 1905.09263 [cs.CL].

- [15] Mark Sandler et al. “MobileNetV2: Inverted Residuals and Linear Bottlenecks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 4510–4520.
- [16] Joan Serrà, Santiago Pascual, and Carlos Segura. *Blow: a single-scale hyperconditioned flow for non-parallel raw-audio voice conversion*. 2019. arXiv: 1906.00794 [cs.LG].
- [17] J. Shen et al. “Natural TTS Synthesis by Conditioning Wavenet on MEL Spectrogram Predictions”. In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Apr. 2018, pp. 4779–4783. DOI: 10.1109/ICASSP.2018.8461368.
- [18] Jean-Marc Valin and Jan Skoglund. “LPCNet: Improving neural speech synthesis through linear prediction”. In: *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2019, pp. 5891–5895.
- [19] Mirjam Wester, Zhizheng Wu, and Junichi Yamagishi. “Analysis of the Voice Conversion Challenge 2016 Evaluation Results”. English. In: *Interspeech 2016*. Interspeech. International Speech Communication Association, Sept. 2016, pp. 1637–1641. DOI: 10.21437/Interspeech.2016-1331.
- [20] Bichen Wu et al. “Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 10734–10742.