2024 edition

Deep Learning for Music Analysis and Generation

Vocoders

(mel → audio)

Yi-Hsuan Yang Ph.D.
yhyangtw@ntu.edu.tw

Objectives

- Traditional DSP-based **source-filter** model for audio
- **Upsampling** layers in DL models for audio
- The design of **GANs** for audio

- It's a type of *conditional* audio **generation** task (with *strong condition*)
  - **"mel → audio"**

Reference 1: ICASSP 2022 Short Course

https://www.slideshare.net/jyamagis/advancements-in-neural-vocoders

https://colab.research.google.com/drive/1EO-ggi1U9f2zXwTiqg7AEljVx11JKta7

Tutorial on neural vocoder.

By Xin Wang, National Institute of Informatics, 2022. See more in LICENSE Section.

This was used for ICASSP 2022 short course **Inclusive Neural Speech Synthesis**, **LECTURE 1, vocoder**.

Full Table of contents:

- Introduction & basics:
  - chapter_1_introduction.ipynb: entry point, and Python/Pytorch conventions used in this hands-on session;
  - chapter_2_DSP_tools_Python.ipynb: selected DSP tools for speech processing;
  - chapter_3_DSP_tools_in_DNN_Pytorch.ipynb: selected DSP tools implemented as layers in neural networks;
- DSP-based Vocoder:
  - chapter_4_DSP-based_Vocoder: traditional DSP-based vocoder included in SPTK toolkit;
- Neural vocoders:
  - chapter_5_DSP+DNN_NSF.ipynb: a simple neural vocoder without autoregressive, GAN, or flow;
  - chapter_6_AR_WaveNet.ipynb: the autogressive WaveNet;
  - chapter_7_AR_iLPCNet.ipynb: the autogressive iLPCNet;
  - chapter_8_Flow_WaveGlow.ipynb: the flow-based WaveGlow model;
  - chapter_9_GAN_HiFiGAN_NSFw/GAN.ipynb: the simple but high-quality HiFiGAN vocoder. We also show the NSF combi

Reference 2: Aalto University Course

https://speechprocessingbook.aalto.fi/Modelling/Vocoder.html

Introduction to Speech Processing

1. Preface
2. Introduction
3. Basic Representations
4. Pre-processing
5. Modelling tools in speech processing
6. Evaluation of speech processing methods
7. Speech analysis
8. Recognition tasks in speech processing
9. Speech Synthesis
10. Transmission, storage and telecommunication
11. Speech enhancement
12. Computational models of human language processing
13. Speech data and experiment design

Reference 3: ISMIR 2019 Tutorial

https://salu133445.github.io/ismir2019tutorial/

Generating Music with GANs

Hao-Wen Dong, Yi-Hsuan Yang

UC San Diego & Academia Sinica

Taiwan AI Labs & Academia Sinica

Welcome to the website for the tutorial "Generating Music with GANs—An Overview and Case Studies" at ISMIR 2019 (November 4th at Delft, The Netherlands).

Slides

- ismir2019-tutorial-slides.pdf

Notebooks

- GAN for images (MNIST Database): `gan.ipynb` Open in Colab
- GAN for piano rolls (Lakh Pianoroll Dataset): `musegan.ipynb` Open in Colab

Related projects

- MuseGAN
- LeadSheetGAN

Outline

- **General idea**
- DL components: upsampling layers and GAN
- GAN-based Mel-vocoders
- More recent work

What is a Vocoder?

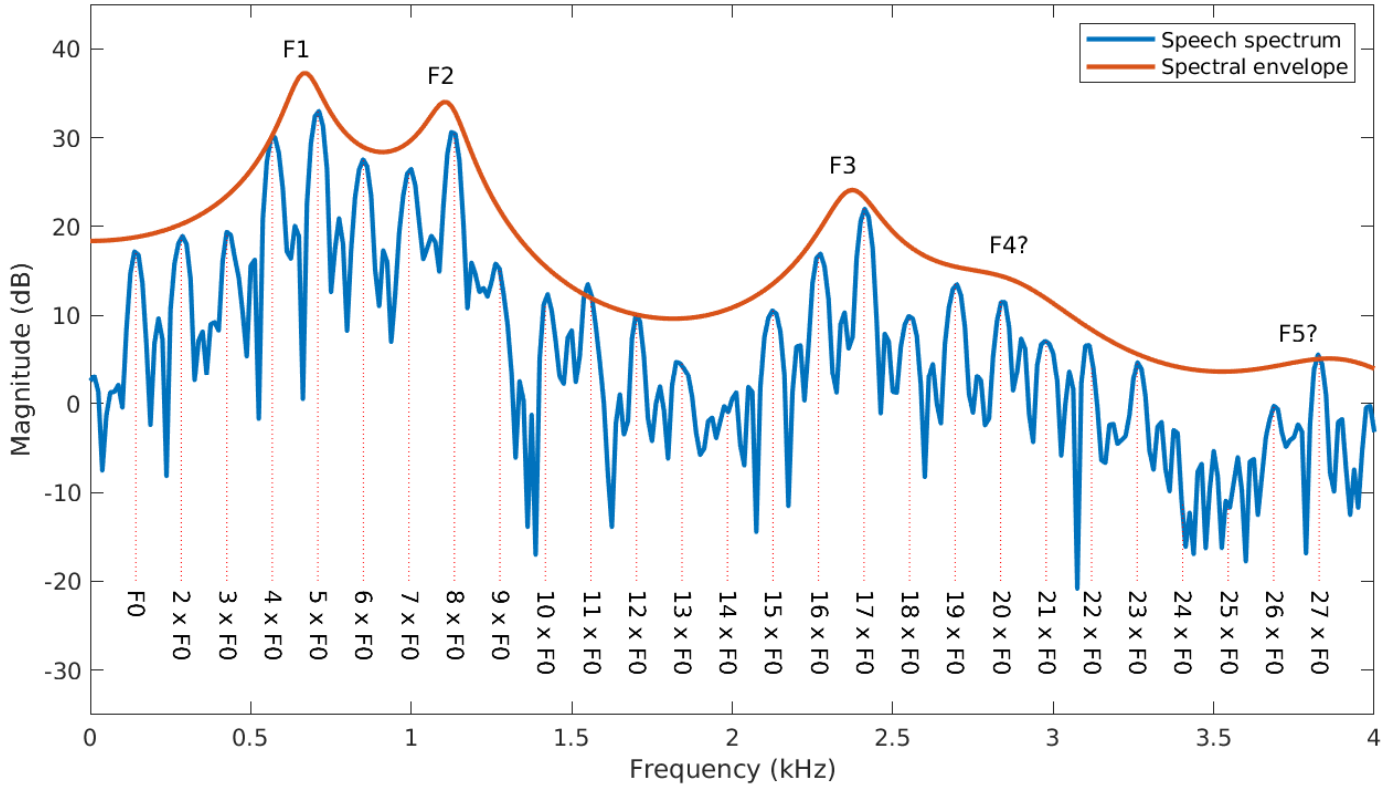
https://speechprocessingbook.aalto.fi/Modelling/Vocoder.html

- **Vocoder** = voice and encoder
  - "Refers to a process of applying a **speech-like** characteristic to a sound"
  - "The basic idea of the vocoder is that any sound which has a **formant** structure [i.e., ***peaks in the spectrum***], will be perceived as a **speech sound**."
  - "Thus if we modify a sound such that it has peaks in the spectrum similar to a speech sound, then it will be perceived as a speech sound. Many of the original features of the sound will be preserved, but it will then have the added interpretation as a speech sound. It is as if the original sound would become the carrier tone of the speech signal."

Ref: Figure from https://speechprocessingbook.aalto.fi/Introduction/Speech_production_and_acoustic_properties.html#content-acoustic-properties

What is a Vocoder?

https://speechprocessingbook.aalto.fi/Introduction/Speech_production_and_acoustic_properties.html#acoustic-properties-of-speech-signals



What is a Vocoder?

https://speechprocessingbook.aalto.fi/Modelling/Vocoder.html

- Formant analysis via Mel-spectral envelope
- Applying the envelope ("*vocoded*")

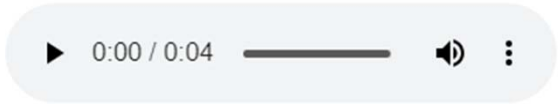
We can then extract the mel-envelopes of the speech and other signals, remove the envelope shape from the other by division, and apply the speech envelope by multiplication as

$$\mathrm{Spectrum}[\mathrm{Vocoder}] = \mathrm{Spectrum}[\mathrm{Other}] \times \frac{\mathrm{Envelope}[\mathrm{Speech}]}{\mathrm{Envelope}[\mathrm{Other}]}$$

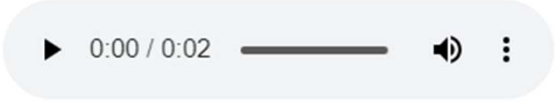
where $\mathrm{Spectrum}[\mathrm{X}]$ and $\mathrm{Envelope}[\mathrm{X}]$ refer to the spectrum and magnitude envelope of X, respectively.

Speech

Other

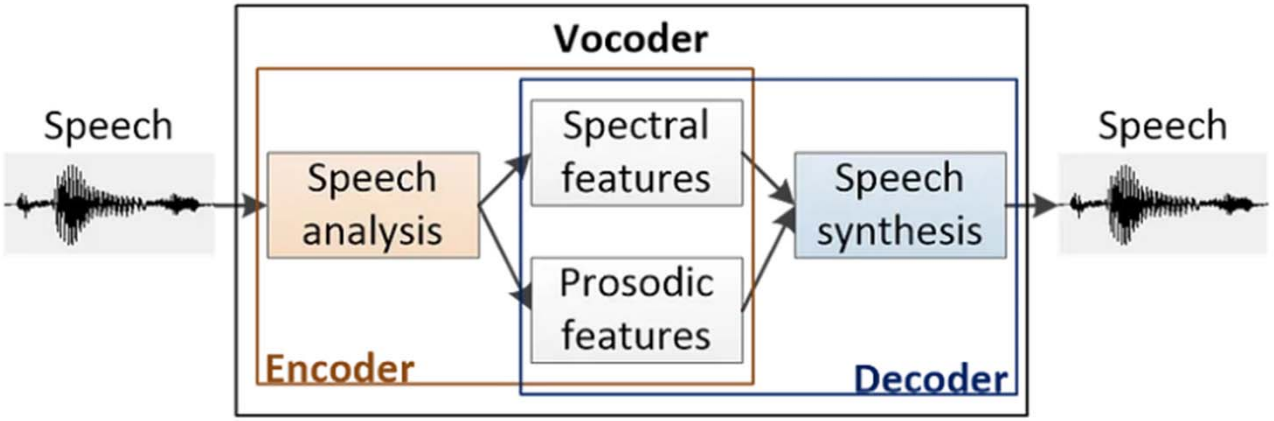


Vocoder



What is a Vocoder?

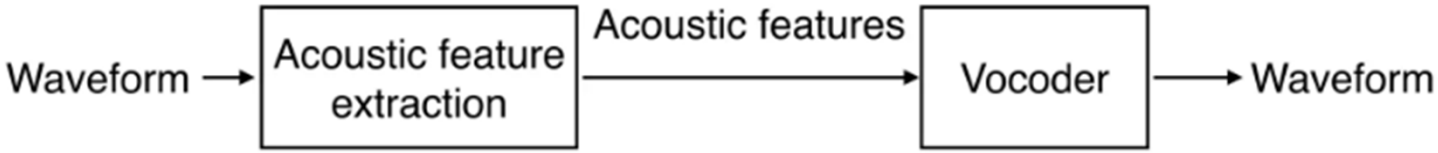
- **Vocoder**: acoustic features → audio waveforms
  - Mostly for **speech** audio initially
  - Early approaches mainly reply on **DSP**
  - SOTA approaches use DL
  - The term is now used more broadly to refer to such a conversion for **any sounds**



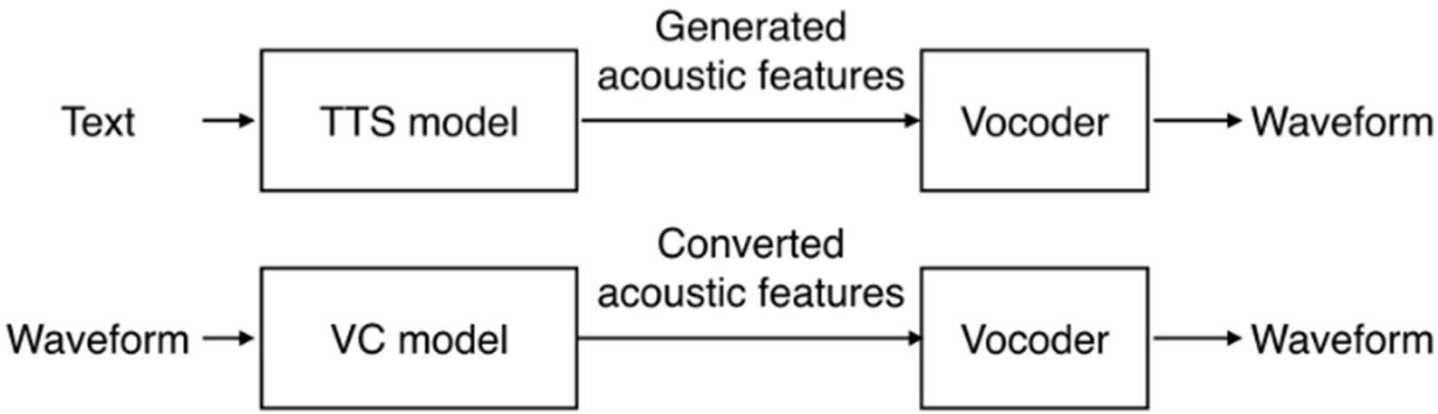
Ref: Figure from https://medium.com/@bigpon517/2020-speech-generation-0-vocoder-and-rnn-and-cnn-based-speech-waveform-generative-models-c324c88e789a

Copy-Synthesis, TTS, and VC

- Copy-synthesis

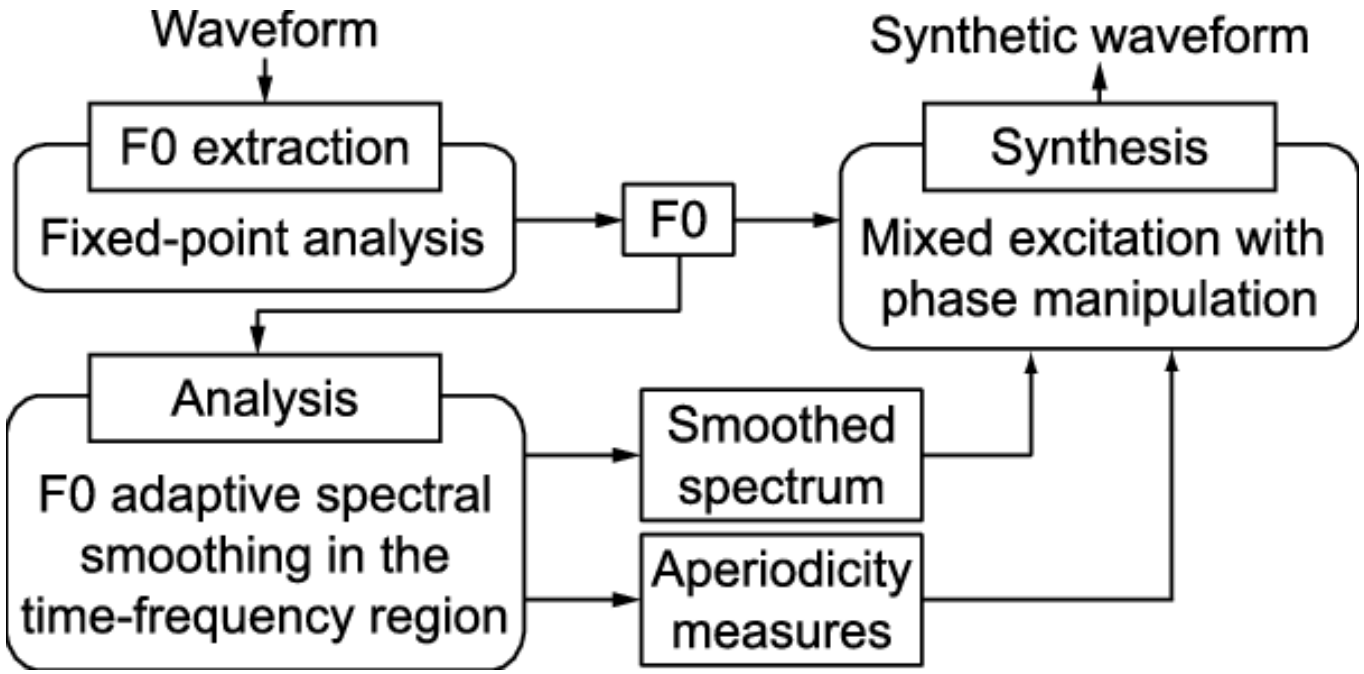


- TTS and VC (in the general form)



Ref: Figure from https://www.slideshare.net/jyamagis/advancements-in-neural-vocoders (page 11)

STRAIGHT Vocoder: F0 + Spectral Envelop + Aperiodicity



Ref 1: Kawahara et al, “Restructuring speech representations using a pitch-adaptive time–frequency smoothing and an instantaneous-frequency-based F0 extraction: Possible role of a repetitive structure in sounds,” Speech Communication 1999.

Ref 2: Figure from https://www.researchgate.net/figure/A-block-diagram-of-STRAIGHT-vocoding-method_fig6_220239312

WORLD Vocoder : F0 + Spectral Envelop + Aperiodicity

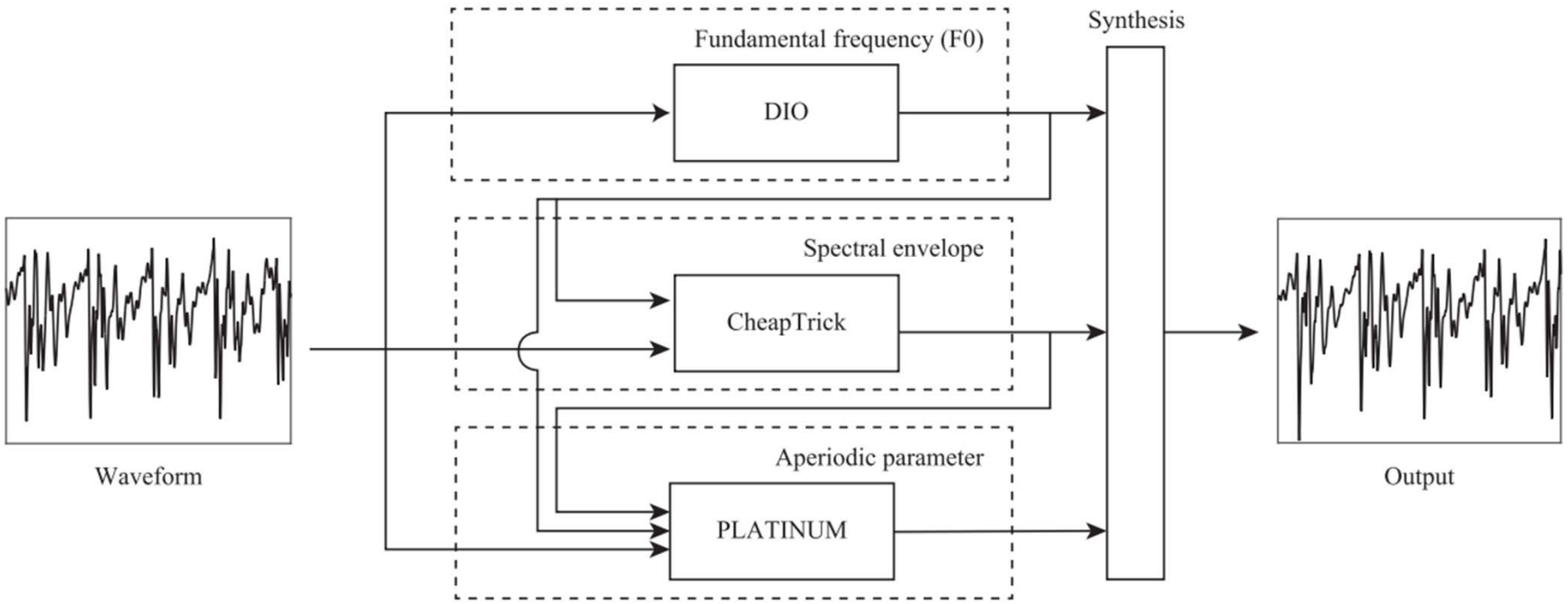
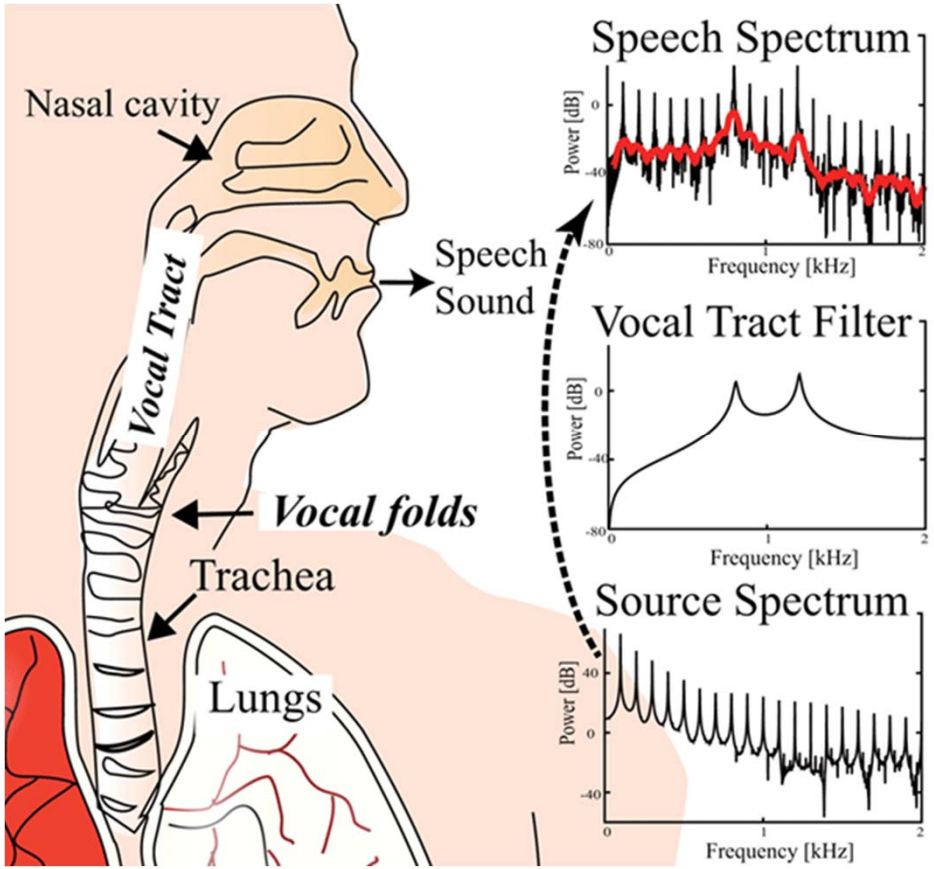


Fig. 1 Overview of the developed system. WORLD consists of three analysis algorithms for determining the F0, spectral envelope, and aperiodic parameters and a synthesis algorithm incorporating these parameters.

Ref: Morise et al, "World: A vocoder-based high-quality speech synthesis system for real-time applications," IEICE Transactions on Information and Systems, 2016.

Source-Filter Model

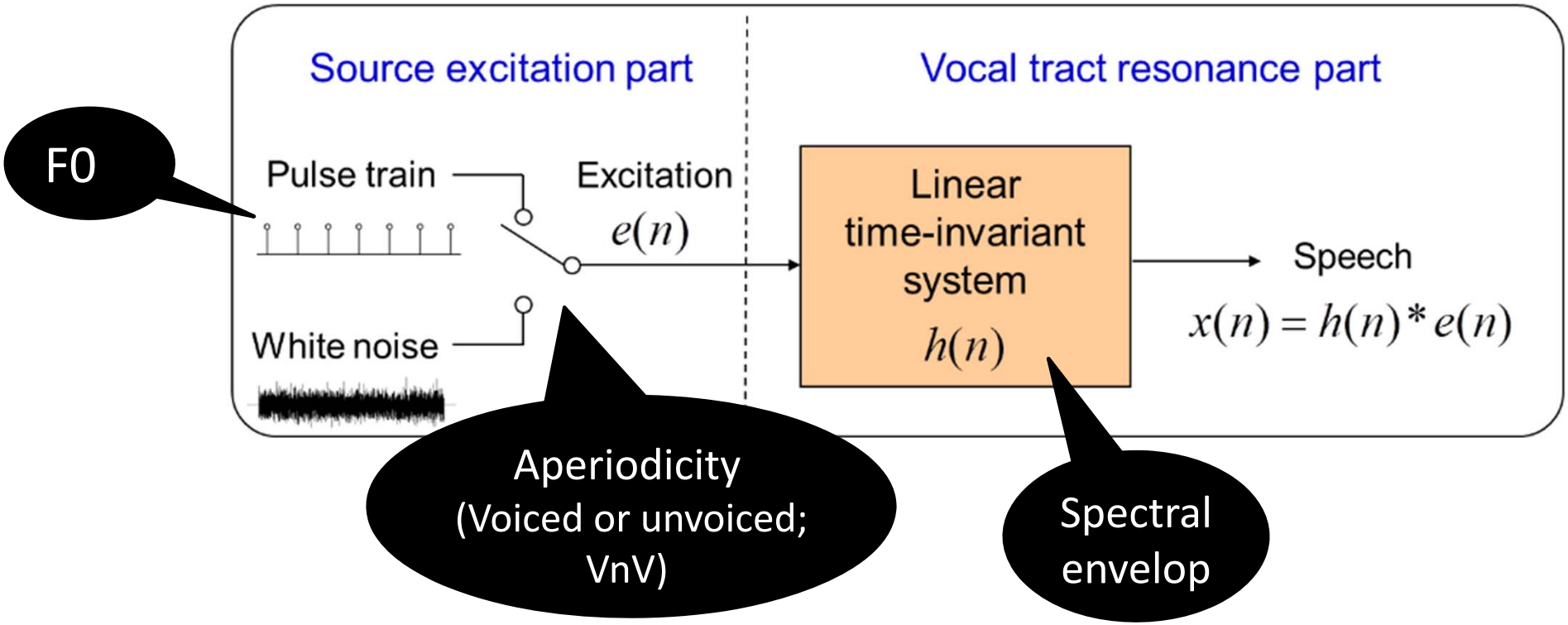
- Assumes that speech is a convolution between a **source signal** (*periodic* or *noise-like*) and an LTI **filter**
- Excitation source signals
  - *Periodic* sound: vowels; nasal consonants
  - *Aperiodic* sound: fricative consonants (e.g., “s”, “f”), plosives (e.g., “p”, “t”), or affricates (e.g., “ch”)
  - A mixture of periodic and aperiodic sound (example: [z])



Ref: Figure from https://oxfordre.com/linguistics/display/10.1093/acrefore/9780199384655.001.0001/acrefore-9780199384655-e-894

Source-Filter Model

- Assumes that speech is a convolution between an **excitation signal** (*periodic* or *noise-like*) and an LTI **filter**

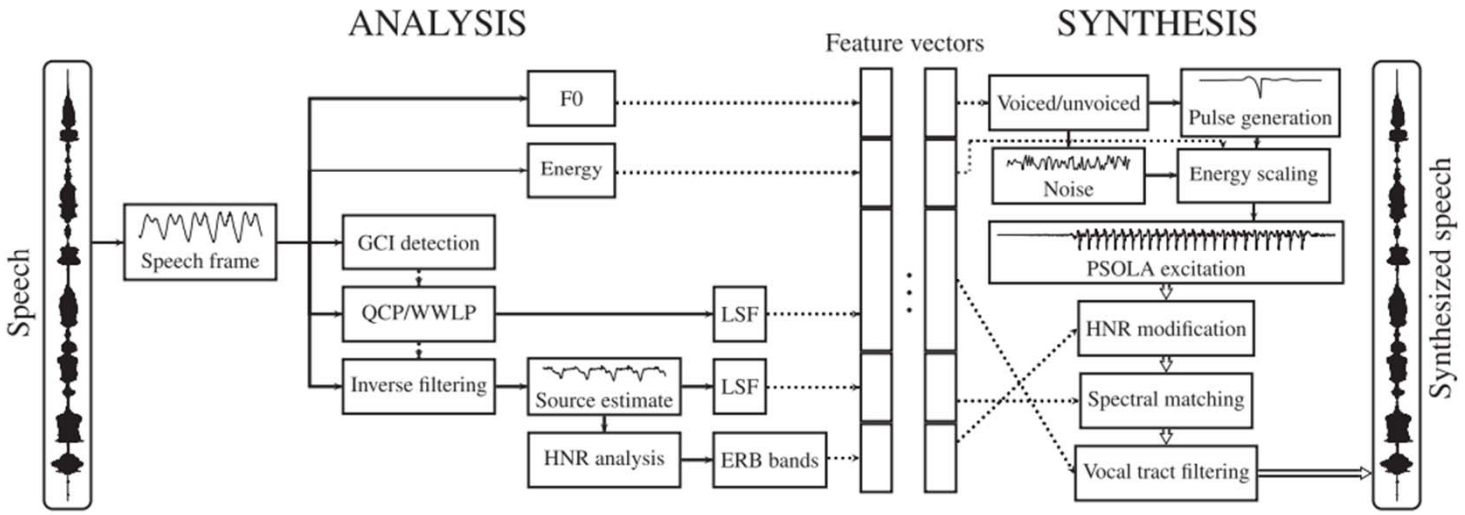


Ref: Figure from https://blog.csdn.net/xmdxcsj/article/details/72419803

Many Other Vocoders

https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8357921

- "The vocoders developed can be categorized roughly into three groups"
  - Mixed excitation with a spectral envelope (e.g., STRAIGHT, WORLD)
  - Sinusoidal vocoders
  - Glottal vocoders



Ref: Airaksinen et al, "A Comparison Between STRAIGHT, glottal, and sinusoidal vocoding in statistical parametric speech synthesis", TASLP 2018

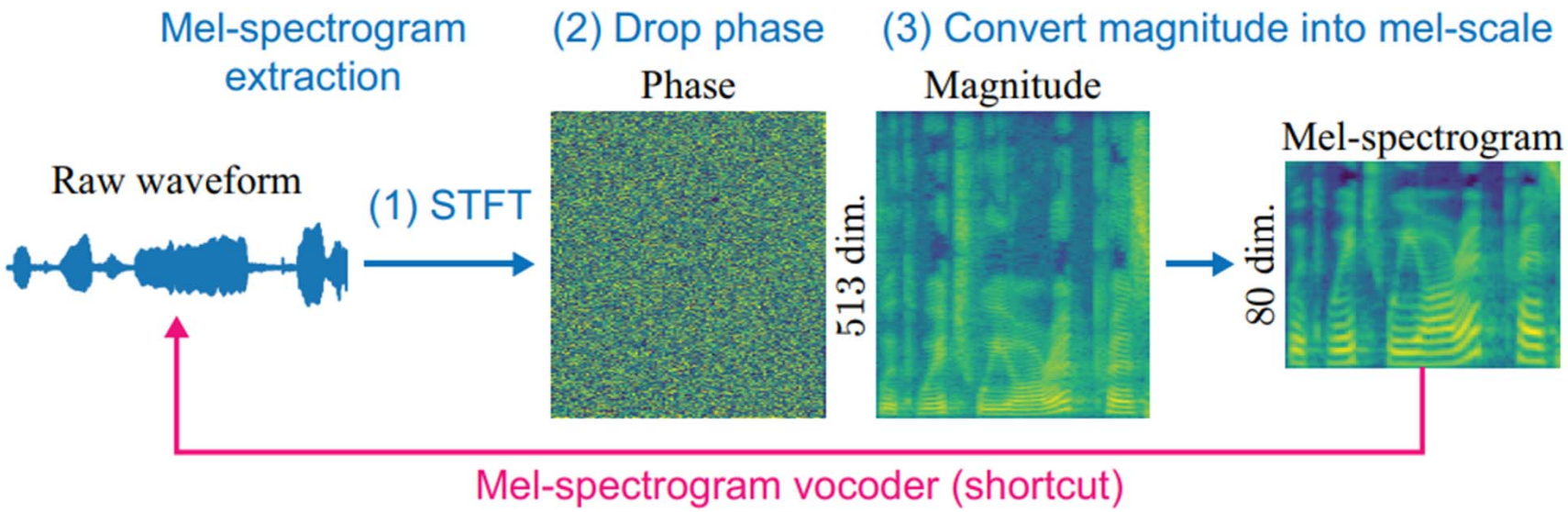
Library: Speech Signal Processing Toolkit

https://github.com/sp-nitech/SPTK

- Aperiodicity extraction (`ap`)
- Conversion from/to log area ratio (`lar2par` and `par2lar`)
- Dynamic range compression (`drc`)
- Entropy calculation (`entropy`)
- Huffman coding (`huffman` , `huffman_encode` , and `huffman_decode`)
- Magic number interpolation (`magic_intpl`)
- Median filter (`medfilt`)
- Mel-cepstrum postfilter (`mcpf`)
- Mel-filter-bank extraction (`fbank`)
- Nonrecursive MLPG (`mlpg -R 1`)
- Pitch adaptive spectrum estimation (`pitch_spec`)
- Pitch extraction by DIO used in WORLD (`pitch -a 3`)
- Pole-zero plot (`gpolezero`)
- Scalar quantization (`quantize` and `dequantize`)
- Spectrogram plot (`gspecgram`)
- Stability check of LPC coefficients (`lpccheck`)
- Subband decomposition (`pqmf` and `ipqmf`)
- WORLD synthesis (`world_synth`)
- Windows build support

Mel-Vocoder

- **Mel-vocoder**: Mel-spectrogram → audio waveforms



Ref: Figure from https://arxiv.org/pdf/2203.02395.pdf

Mel-Vocoder

- **Mel-vocoder**: Mel-spectrogram → audio waveforms
- For not only speech but **any sounds**
- We can divide an audio synthesis model into two steps:
  - The **generator** generates acoustic features, or the Mel-spectrogram
  - The **vocoder** converts Mel-spectrogram into waveforms
- Why Mel-vocoder becomes popular in the DL era
  - More information than F0 + spectral envelop + aperiodicity
  - Modern computing devices can deal with acoustic features with a larger dimension

Mel-Vocoder: Approaches

- Pure **signal processing** techniques
  - Griffin & Lim algorithm
- **Autoregressive** neural networks
  - WaveNet, SampleRNN, WaveRNN
  - "Inference with these models is inherently **slow** and inefficient because audio samples must be **generated sequentially**"
- **Non-autoregressive** neural networks
  - "Orders of magnitudes faster than their auto-regressive counterparts because they are highly parallelizable and can fully exploit modern deep learning hardware"

Ref: Quotes from the MelGAN paper (https://arxiv.org/pdf/1910.06711.pdf)

Griffin & Lim Algorithm

- Initially for magnitude spectrogram → audio waveforms
- Iterative optimization
- No training is needed
- Robotic artifacts

算法步骤：

- 随机初始化一个相位谱
- 用这个相位谱与已知的幅度谱（来自MEL谱）经过ISTFT（逆傅里叶变换）合成新的语音波形
- 用合成语音做STFT， 得到新的幅度谱和新的相位谱
- 丢弃新的幅度谱，用已知幅度谱与新的相位谱合成新的语音
- 重复2,3,4多次，直至合成的语音达到满意的效果或者迭代次数达到设定的上限

https://blog.csdn.net/CSDN_71560364126/article/details/103968034

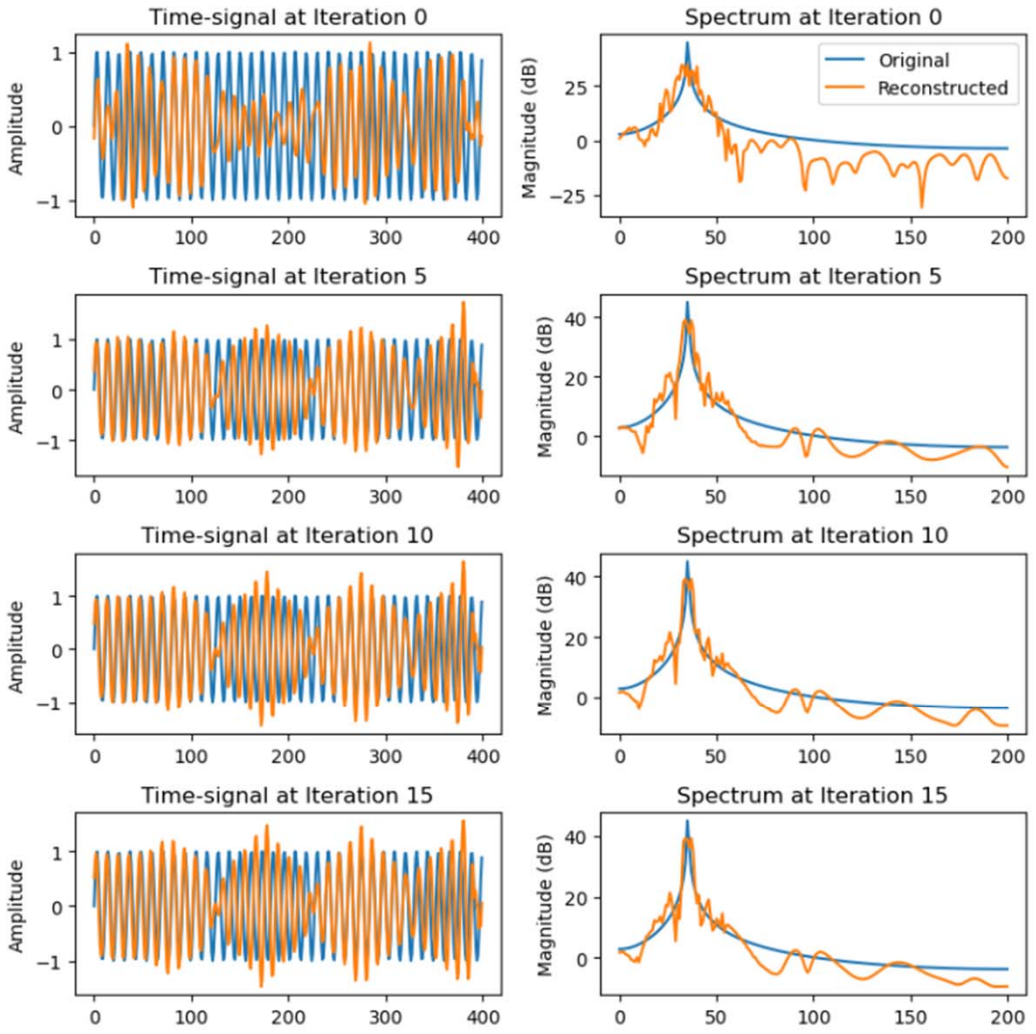
Ref: Griffin & Lim, “Signal estimation from modified short-time Fourier transform,” TASLP 1984.

Griffin & Lim Algorithm

https://speechprocessingbook.aalto.fi/Modelling/griffinlim.html

```
# example signal
original_waveform = np.sin(np.linspace(0,221,400))
_,_,original_spectrogram = scipy.signal.stft(original_waveform,nperseg=100, return_onesided=True)
frames = original_spectrogram.shape[1]

# initialize target magnitude spectrogram
magnitude_spectrogram = np.abs(original_spectrogram)

# initial guess of complex spectrogram = random phase
spectrogram = magnitude_spectrogram*np.exp(1j*2*np.pi*np.random.rand(51,frames))
```

```
plt.figure(figsize=(8,8))
N = 4
K = 5
for h in range(N*K - K+1):
    _,waveform = scipy.signal.istft(spectrogram, nperseg=100, input_onesided =True, boundary=True)
    waveform = np.real(waveform)
    _,_,spectrogram = scipy.signal.stft(waveform,    nperseg=100, return_onesided=True)
    spectrogram = np.exp(1j*np.angle(spectrogram))*magnitude_spectrogram

    if h % K == 0:
        k = h//K
        plt.subplot(N,2,2*k+1)
        plt.plot(original_waveform)
        plt.plot(waveform)
        plt.title(f"Time-signal at Iteration {h}")
        plt.ylabel('Amplitude')

        plt.subplot(N,2,2*k+2)
        plt.plot(20*np.log10(np.abs(scipy.fft.rfft(original_waveform))),label='Original')
        plt.plot(20*np.log10(np.abs(scipy.fft.rfft(waveform))),label='Reconstructed')
        plt.title(f"Spectrum at Iteration {h}")
        plt.ylabel('Magnitude (dB)')
        if h==0: plt.legend()
```



Griffin & Lim-based Mel-Vocoder via LibROSA

https://librosa.org/doc/main/generated/librosa.feature.inverse.mel_to_audio.html#librosa.feature.inverse.mel_to_audio

librosa.feature.inverse.mel_to_audio(*M, *, sr=22050, n_fft=2048, hop_length=None, win_length=None, window='hann', center=True, pad_mode='constant', power=2.0, n_iter=32, length=None, dtype=<class 'numpy.float32'>, **kwargs*) [source]

Invert a mel power spectrogram to audio using Griffin-Lim.

This is primarily a convenience wrapper for:

```
>>> S = librosa.feature.inverse.mel_to_stft(M)
>>> y = librosa.griffinlim(S)
```

librosa.util.nnls(*A, B, **kwargs*) [source]

Non-negative least squares.

Given two matrices A and B, find a non-negative matrix X that minimizes the sum squared error:

```
err(X) = sum_i,j ((AX)[i,j] - B[i, j])^2
```

- **mel-to-stft**: approximate STFT magnitude from a Mel power spectrogram using a non-negative least squares (NNLS) algorithm

Autoregressive vs Non-autoregressive Generation

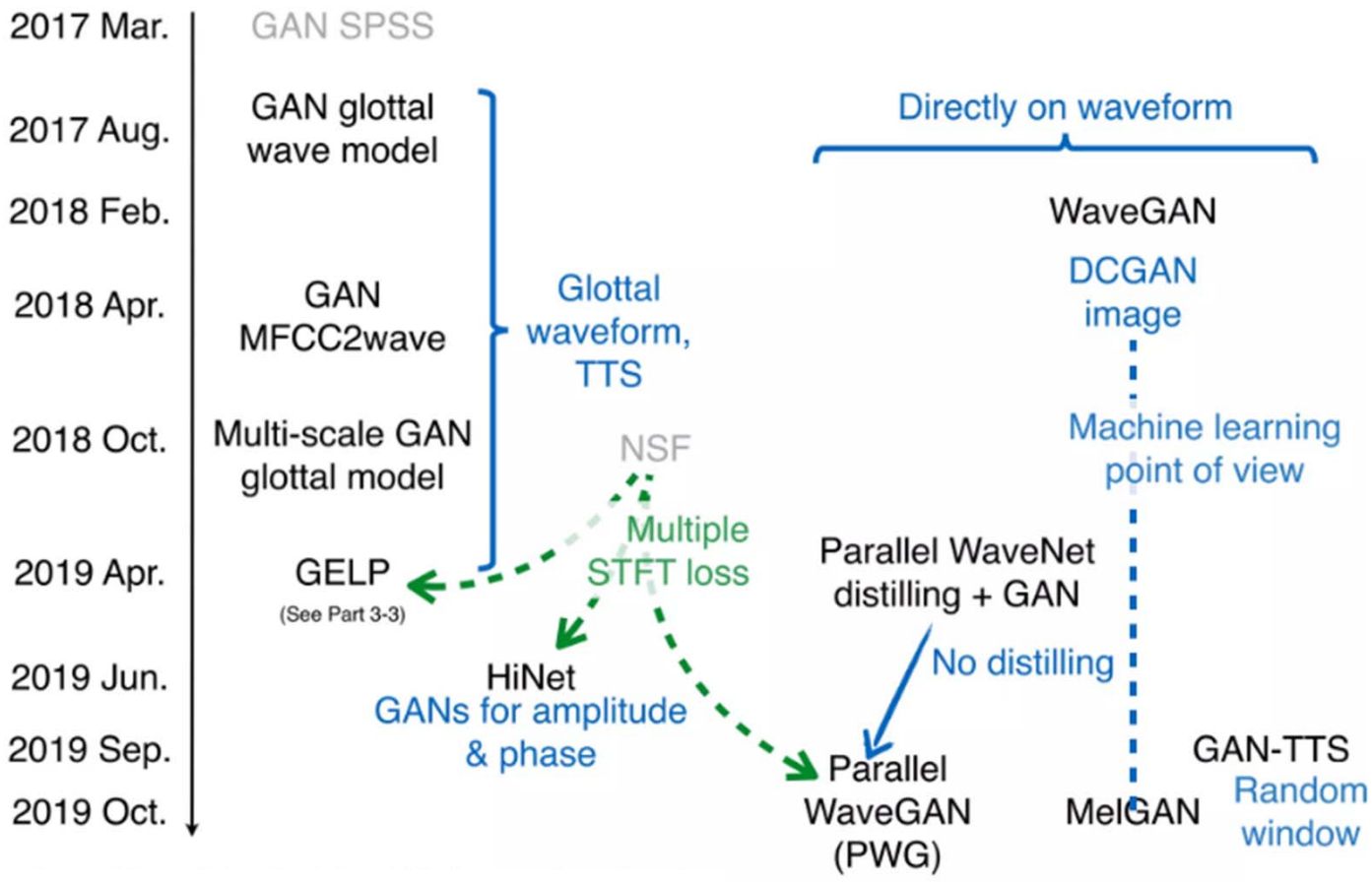
- **Autoregressive**
  - Causal (no information about the future)
  - One sample at a time
  - Good for *online* application
  - Too slow for offline application
  - And good for *unconditional audio generation*
  - Examples: WaveNet, Transformer
- **Non-autoregressive**
  - Non-causal
  - Good for Mel-vocoding because we have seen the entire input

Non-autoregressive Neural Vocoders

- Via **knowledge-distillation** of autoregressive models
  - Parallel WaveNet (Oord et al., 2017) and Clarinet (Ping et al., 2018)
- Via **flow-based** models
  - WaveGlow (Prenger et al., 2019)
- Via **GAN**-based models
  - Reconstruction loss + adversarial loss
  - SOTA

Ref: Quotes from the MelGAN paper (https://arxiv.org/pdf/1910.06711.pdf)

GAN-based Models



Ref: Figure from https://www.slideshare.net/jyamagis/advancements-in-neural-vocoders (page 111)

Why GAN? (Or, Why Reconstruction Loss Alone is Not Enough)

This new methodology … has allowed for waveforms to be generated … at … a higher real-time speed than in models with autoregressive constraints

MelGAN is substantially faster than other mel-spectrogram inversion alternatives

Multi-band MelGAN runs at more than 10x faster than real-time on CPU.

- Not only about fast inference speed
- Also on combating with the over-smoothing

*(Conventional) models are limited due to the point-wise regression in the time domain, which results in **smoothing and loss of high frequencies**.*

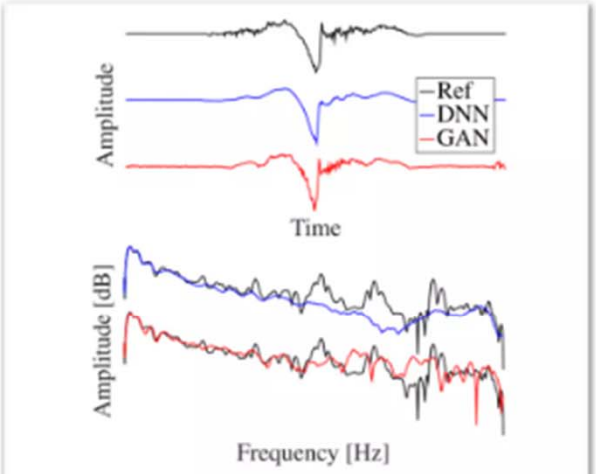
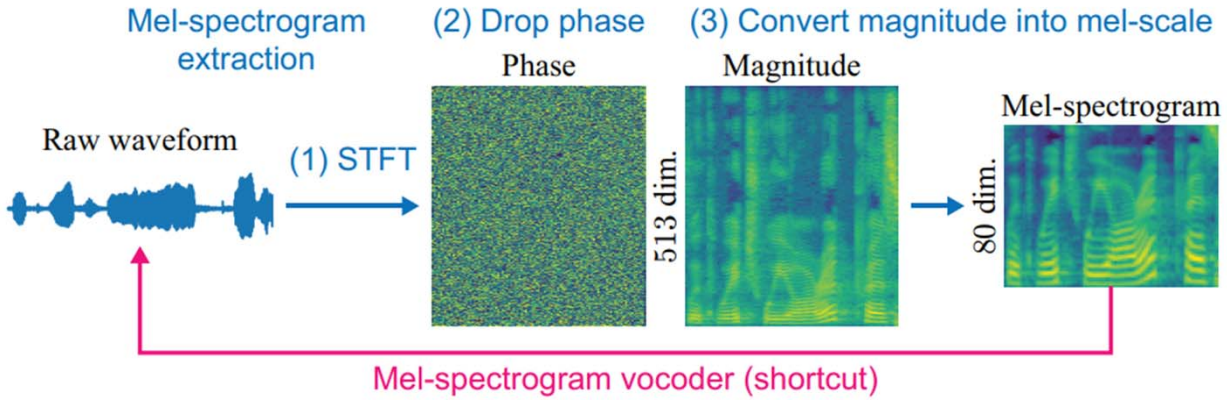


Fig. 2. Excitation pulses shown in the time and frequency domain. DNN pulse model (blue) fits the overall reference pulse (black) shape, but high-frequency stochastic components are smoothed out, whereas GAN (red) is able to generate realistic high-frequency components.

Ref: Figure from https://www.slideshare.net/jyamagis/advancements-in-neural-vocoders (page 113)

Mel-Vocoder

- **Mel-vocoder**: Mel-spectrogram → audio waveforms
  - Upsampling
  - Time-frequency conversion
    - Inverse the Mel-scale
    - Predict the phase



  - There is a “ground truth” output
  - The input (“Mel-spectrogram”) is temporally aligned with the expected output
  - Paired data for training such a model can be easily obtained

Ref: Figure from https://arxiv.org/pdf/2203.02395.pdf

Source Separation vs Mel-Vocoder

Task	Input/Output	Conditioning type	Data availability
Source separation	audio (mixture) → audio (stem)	*strong & time-aligned* conditioning (perfect alignment)	Harder to get paired data
Mel-vocoder	mel → audio	*strong & time-aligned* conditioning (output sequence length is a factor of the input sequence length)	Extremely easy to get paired data

- *Strong* condition: it can be said that there is a "ground-truth" output
- *Time-aligned*: the condition is time-varying and is temporally aligned with the input
- From encoder/decoder perspective
  - Timbre classifier: encoder (downsampling)
  - Source separation : encoder/decoder
  - Mel-vocoder: decoder (upsampling)

Other Types of Audio Generation Problems

Task	Input/Output	Conditioning type	Data availability
Source separation	audio (mixture) → audio (stem)	*strong & time-aligned* conditioning	Harder to get paired data
Mel-vocoder	mel → audio	*strong & time-aligned* conditioning	Extremely easy to get paired data
Voice conversion	audio (source) → audio (target)		
Accomp. generation	audio (singing) → audio (accomp.)		
Audio synthesis	MIDI → audio		
Audio generation	x → audio		

Recap: Approaches to Deal with Phase for STFT-based Models

https://source-separation.github.io/tutorial/basics/phase.html

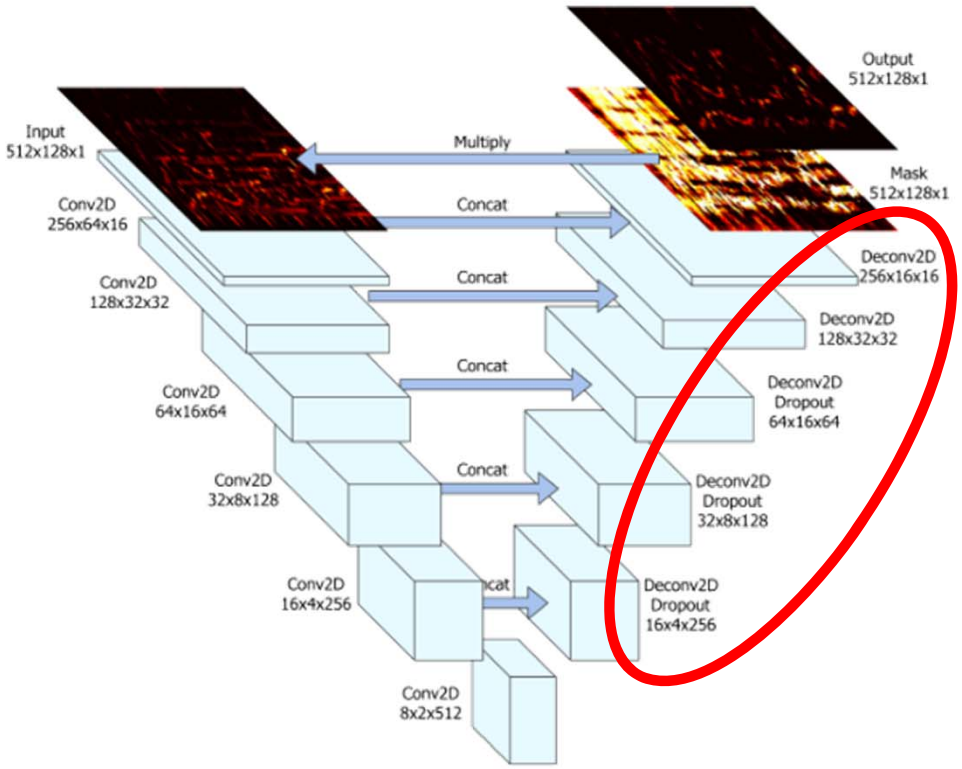
- **Copy the phase (from the input)**
  - Work fine for source separation (but NOT for Mel-vocoders)
- **Given the magnitude, estimate the phase**
  - Griffin & Lim
  - DL-based Mel-vocoders
- **Work on complex-valued spectrograms**
- **Work on audio waveforms, not spectrograms**

Outline

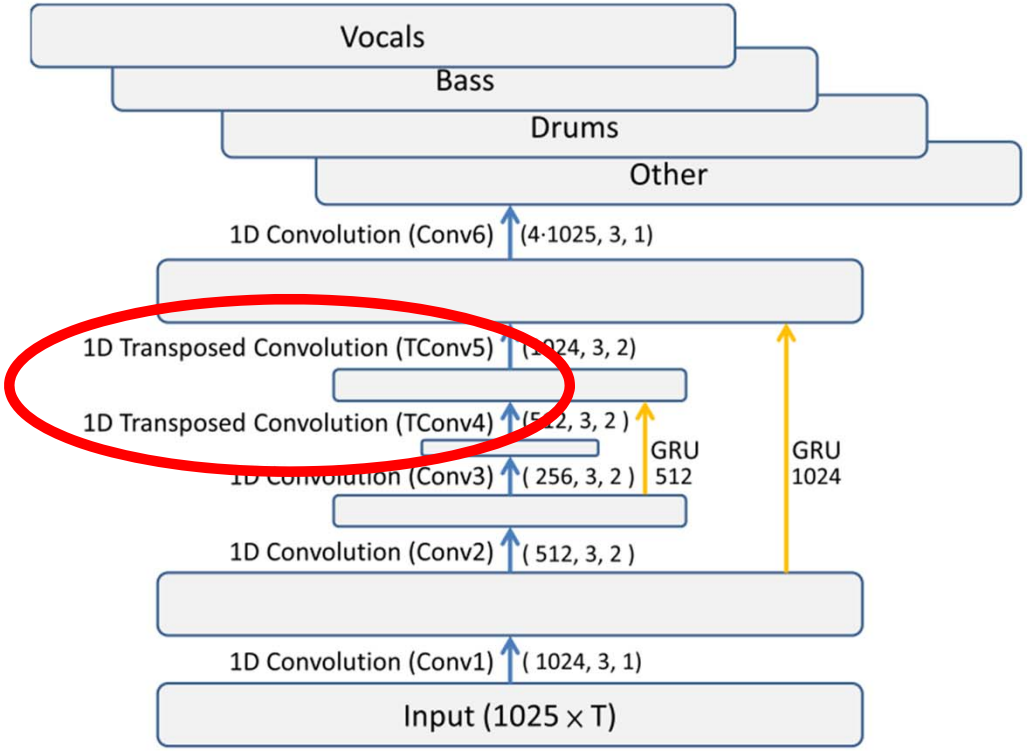
- General idea
- **DL components: upsampling layers and GAN**
- GAN-based Mel-vocoders
- More recent work

Upsampling Layers: Transposed Convolution

- Also known as "**deconvolution** layers"

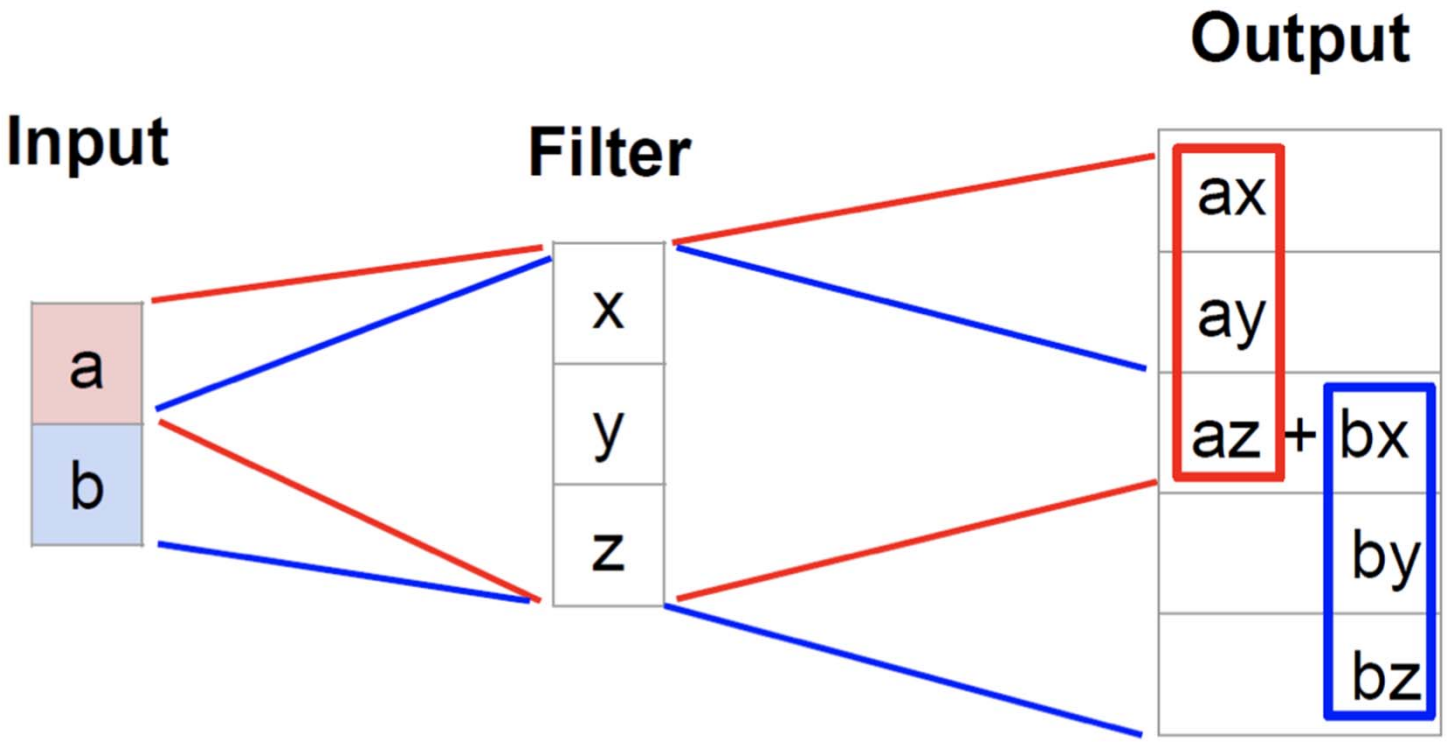


U-Net-based source separation model [jansson17ismir]



ARC [liu18icmla]

Convolution and Transposed Convolution



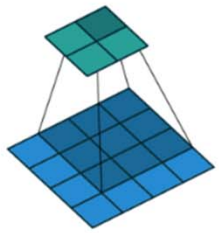
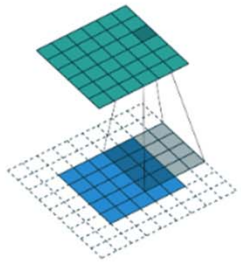
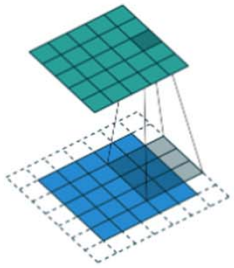
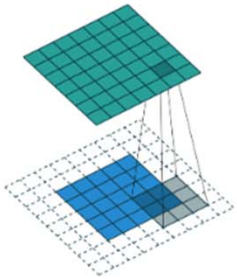
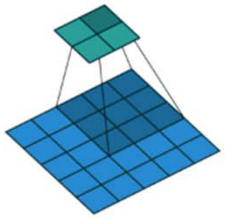
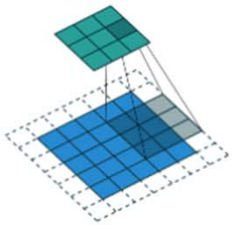
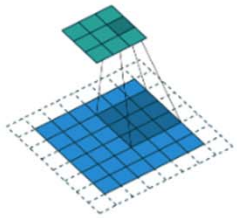
Ref: Figure from https://medium.com/jun94-devpblog/dl-12-unsampling-unpooling-and-transpose-convolution-831dc53687ce

Downsampling Layers: Convolution

https://github.com/vdumoulin/conv_arithmetic

Convolution animations

N.B.: Blue maps are inputs, and cyan maps are outputs.

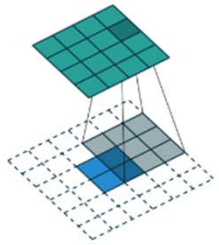
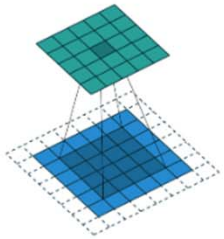
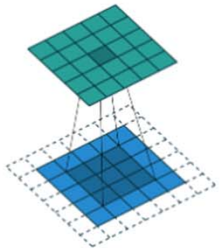
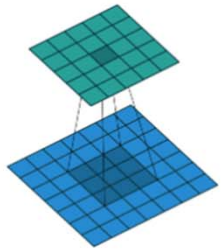
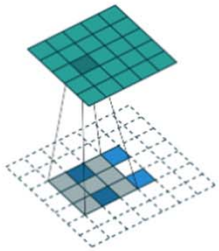
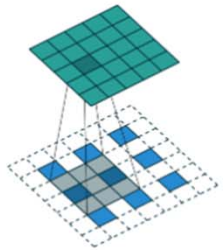
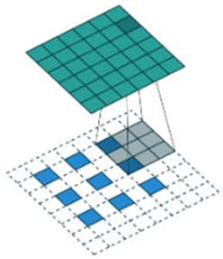
			
No padding, no strides	Arbitrary padding, no strides	Half padding, no strides	Full padding, no strides
			
No padding, strides	Padding, strides	Padding, strides (odd)	

Upsampling Layers: Transposed Convolution

https://github.com/vdumoulin/conv_arithmetic

Transposed convolution animations

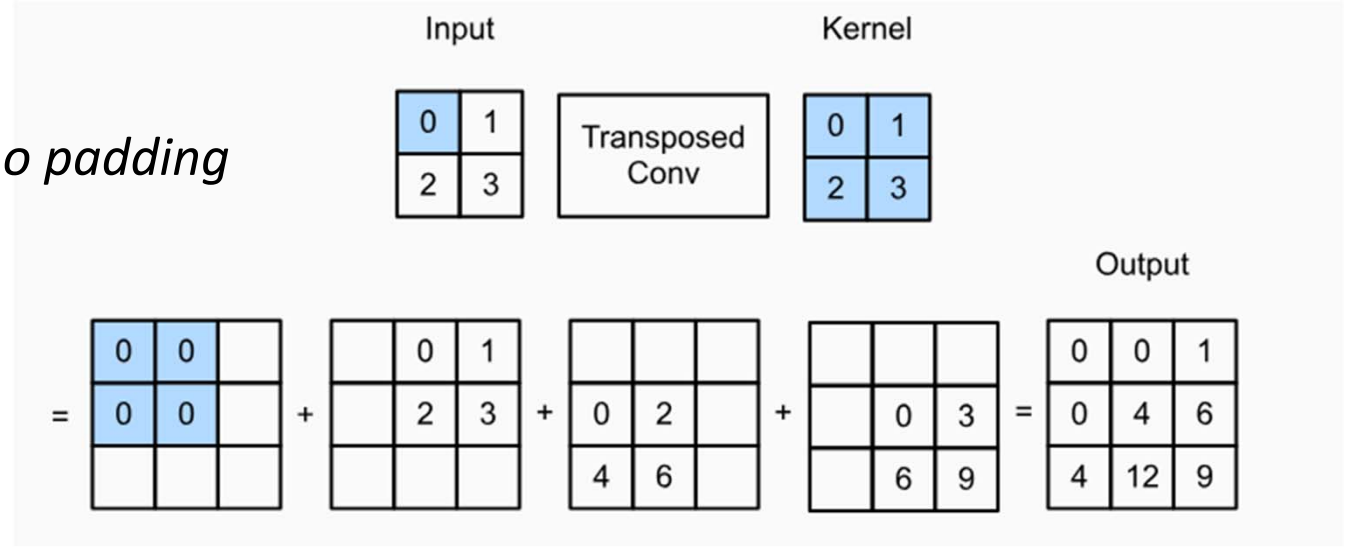
N.B.: Blue maps are inputs, and cyan maps are outputs.

			
No padding, no strides, transposed	Arbitrary padding, no strides, transposed	Half padding, no strides, transposed	Full padding, no strides, transposed
			
No padding, strides, transposed	Padding, strides, transposed	Padding, strides, transposed (odd)	

Upsampling Layers: Transposed Convolution

https://d2l.ai/chapter_computer-vision/transposed-conv.html

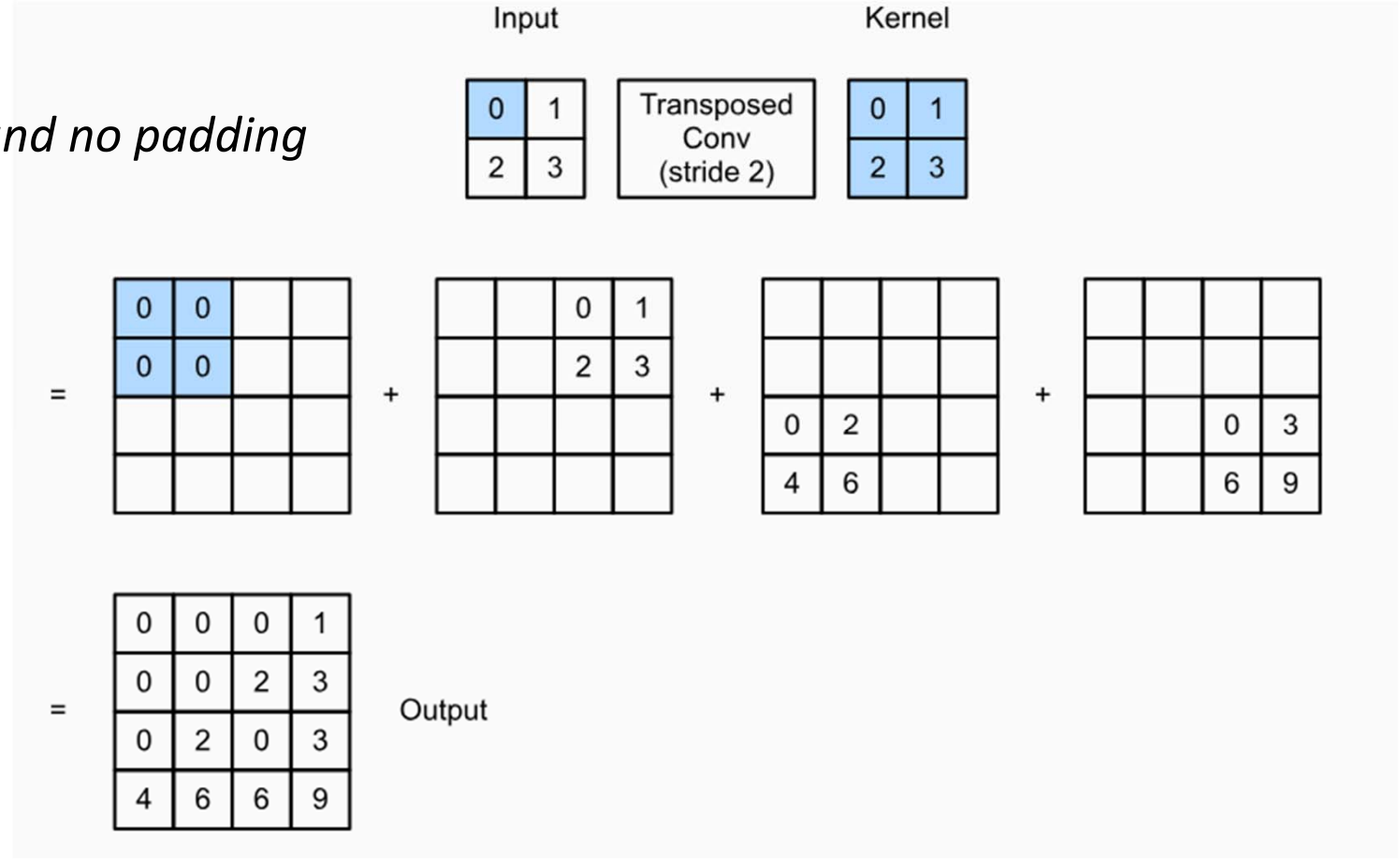
*stride of **1** and no padding*



Upsampling Layers: Transposed Convolution

https://d2l.ai/chapter_computer-vision/transposed-conv.html

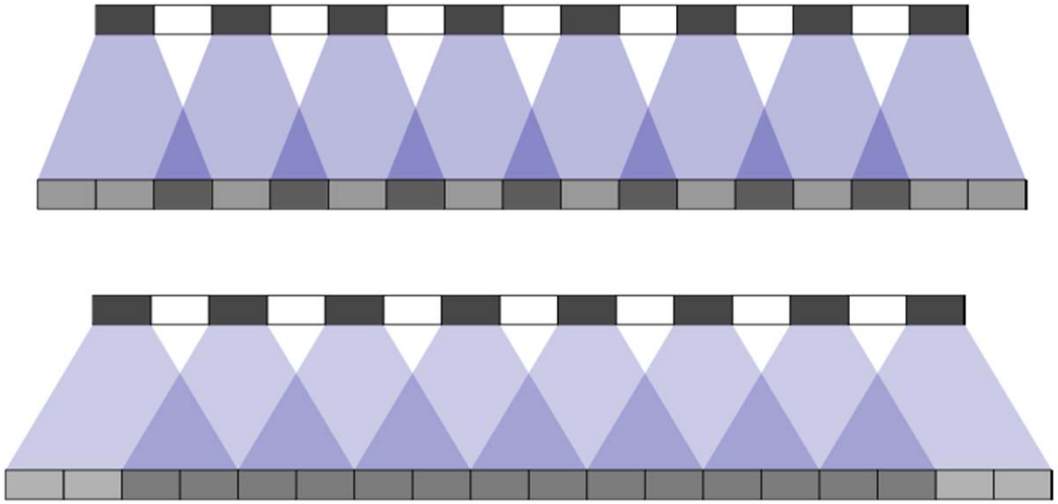
*stride of **2** and no padding*



Upsampling Layers: The Checkerboard Artifacts in Images

https://distill.pub/2016/deconv-checkerboard/

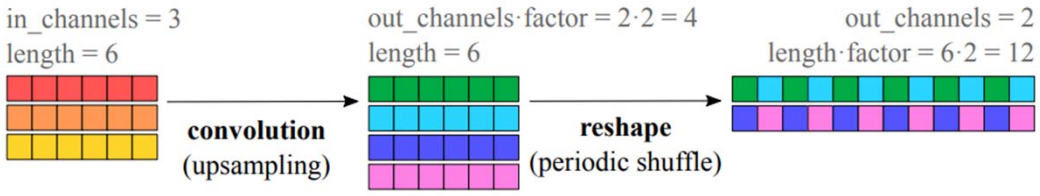
- Due to uneven overlaps: the kernel size (the output window size) is not divisible by the stride (the spacing between points on the top)
- Solution: Use a kernel size that is divided by your stride



Upsampling Layers: The Checkerboard Artifacts in Audio

http://www.jordipons.me/apps/upsamplers/

- Explore other neural upsampler approaches
  - **Transposed convolution**: no overlap (length=stride) or full overlap (length is multiple of the stride), not partial overlap
    - "Even under no overlap and full overlap setups, the loss function and the weights' initialization issues remain"
  - **Interpolation upsamplers**
    - Nearest neighbor interpolation
    - Linear interpolation
    - Stretch interpolation
    - Sinc interpolation
  - **Subpixel convolution**



Ref: Pons et al, "Upsampling layers for music source separation," EUSIPCO 2023

Upsampling Layers: The Checkerboard Artifacts in Audio

http://www.jordipons.me/apps/upsamplers/

- Explore other neural upsampler approaches
- In the context of **source separation**

Music source separation (MUSDB [27] benchmark)	SDR ↑	epoch	#parm
Demucs-like (Fig. 1, b): transposed CNN (full-overlap)	5.35	319 s	703M
Demucs-like (Fig. 1, c): nearest neighbor interpolation	5.17	423 s	716M
Demucs-like: linear interpolation	4.62	430 s	716M
Demucs-like (Fig. 1, d): subpixel CNN	5.38	311 s	729M
Modified (Fig. 9, a): transposed CNN (full-overlap)	5.37	326 s	703M
Modified (Fig. 9, b): subpixel CNN	5.38	315 s	729M
WaveUnet [2]: linear intepolation	3.23	-	10M
Demucs [3]: transposed convolution (full-overlap)	5.34[4]	-	648M
OpenUnmix [30]: spectrogram-based	5.36	-	8.9M
Sams-net [31]: spectrogram-based	5.65	-	3.7M

"Informal listening, however, reveals that **tonal artifacts** can emerge even after training, **especially in silent parts and with out-of-distribution data** (e.g., with sounds and conditions not seen during training)"

"We find that **nearest neighbor and linear interpolation models do not have this disadvantage, although they achieve worse SDR scores**."

Ref: Pons et al, "Upsampling artifacts in neural audio synthesis," ICASSP 2021

Upsampling Layers “for Waveforms”: Snake Activation

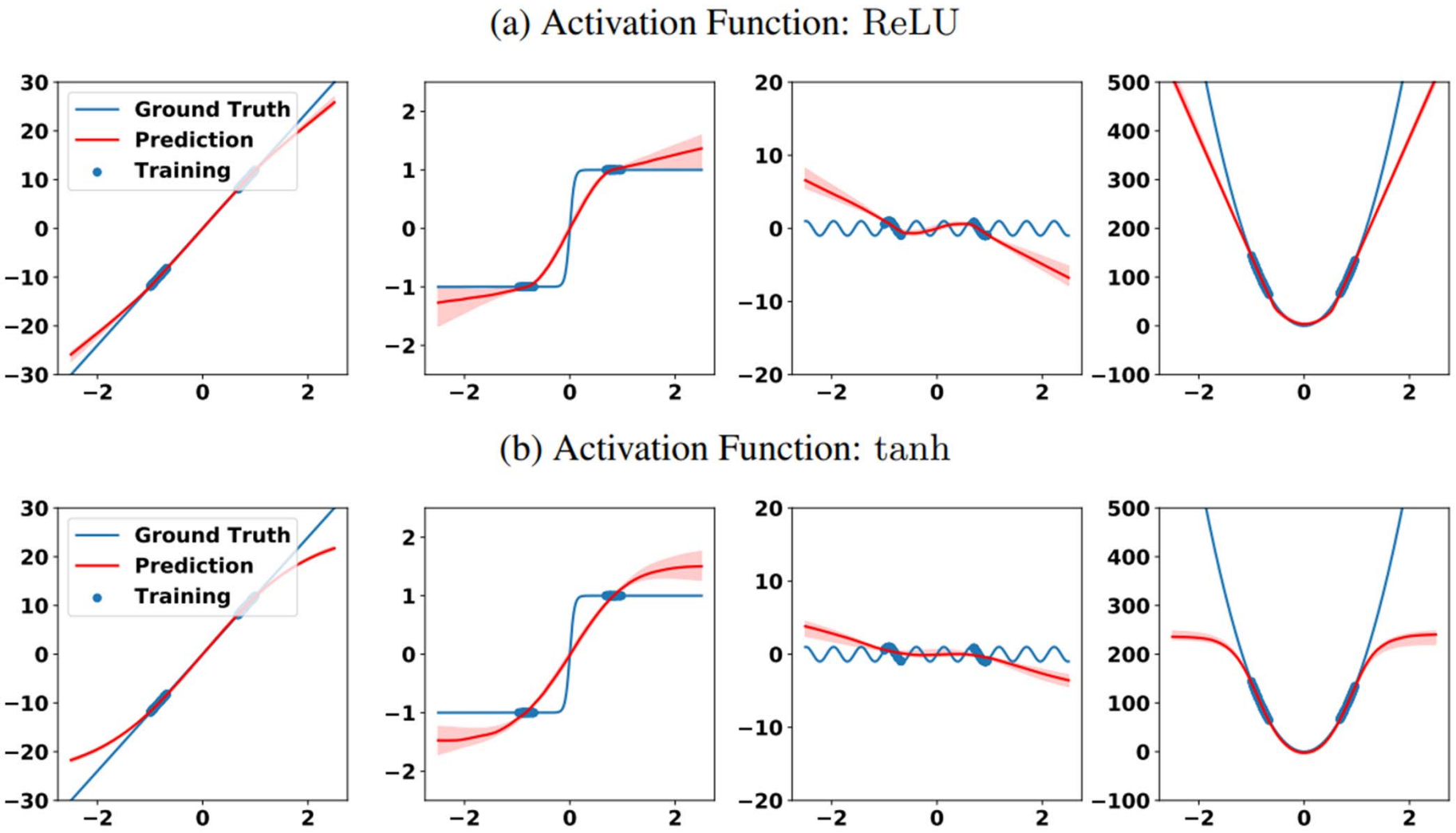
- **Mel-vocoder**: Mel-spectrogram → audio waveforms

Neural Networks Fail to Learn Periodic Functions and How to Fix It

Liu Ziyin[1], **Tilman Hartwig**[1,2,3], **Masahito Ueda**[1,2,4]
[1]Department of Physics, School of Science, The University of Tokyo
[2]Institute for Physics of Intelligence, School of Science, The University of Tokyo
[3]Kavli IPMU (WPI), UTIAS, The University of Tokyo
[4]RIKEN CEMS

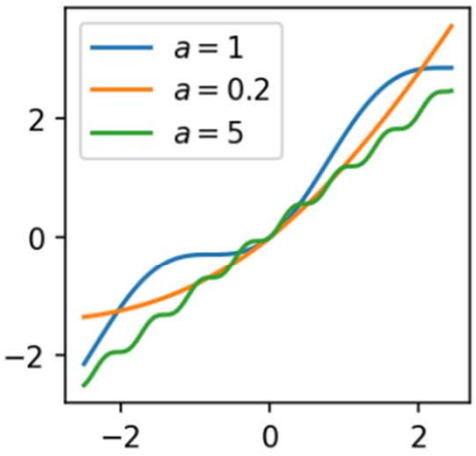
34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

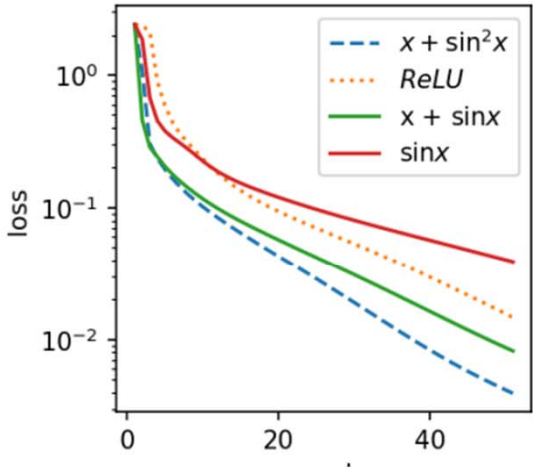
Upsampling Layers “for Waveforms”: Snake Activation

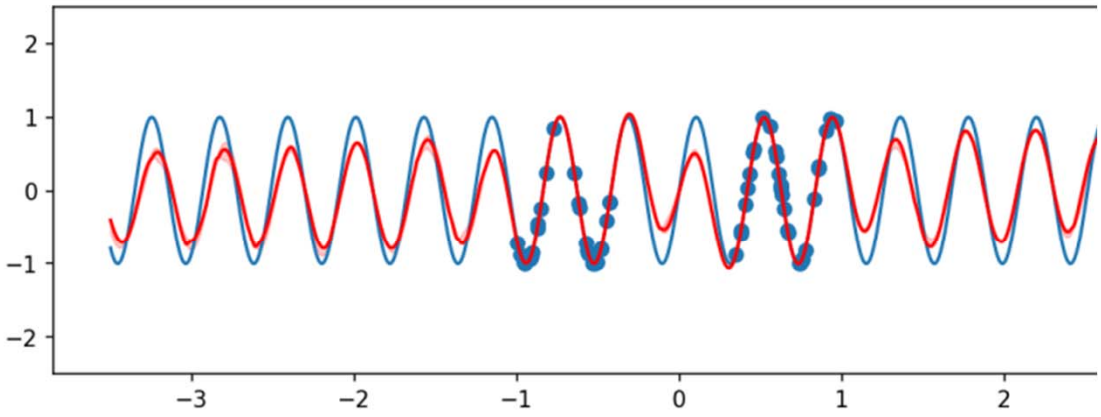


Upsampling Layers “for Waveforms”: Snake Activation

$$\text{Snake}_a := x + \frac{1}{a}\sin^2(ax) = x - \frac{1}{2a}\cos(2ax) + \frac{1}{2a},$$





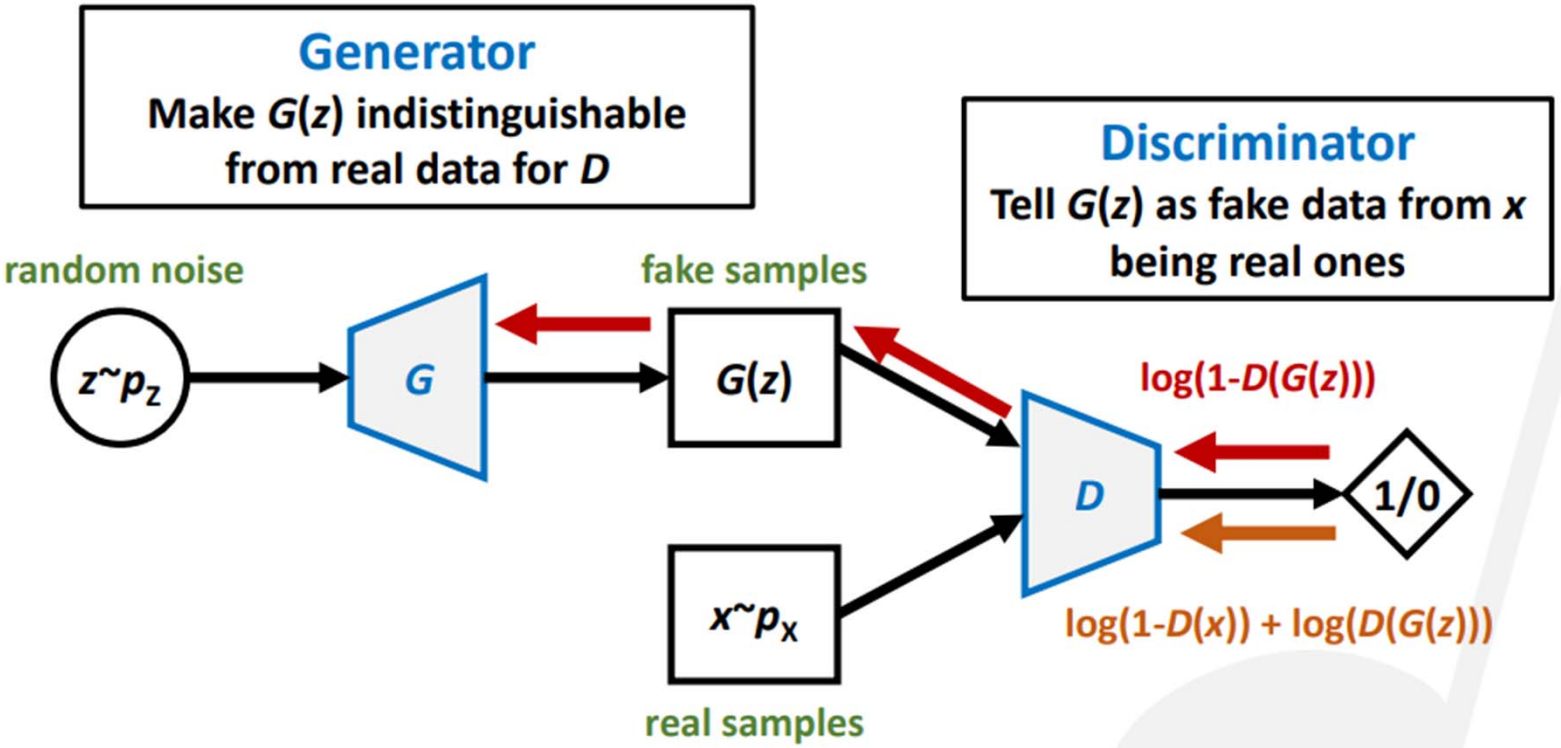


	ReLU	Swish	Tanh	$\sin(x)$	$x + \sin(x)$	$x + \sin^2(x)$
monotonic	✓	✗	✓	✗	✓	✓
(semi-)periodic	✗	✗	✗	✓	✓	✓
first non-linear term	-	$\frac{x^2}{4}$	$-\frac{x^3}{3}$	$-\frac{x^3}{6}$	$-\frac{x^3}{6}$	x^2

Table 1: Comparison of different periodic and non-periodic activation functions.

GAN: Generative Adversarial Network

https://salu133445.github.io/ismir2019tutorial/



GAN: Classic Minimax Formulation

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))].$$

- Both terms are related to the discriminator D
  - The goal is to make $D(x)$ close to 1
  - And to make $D(G(z))$ close to 0
- Only the second term is related to the generator G
  - The goal is to make $D(G(z))$ close to 1
- Update D when G is fixed; and update G when D is fixed

Ref: Goodfellow et al, "Generative adversarial nets," NeurIPS 2014

GAN: The Hinge Loss Formulation

$$\min_G \max_D V(D,G)$$

$$V_D(\hat{G}, D) = \mathop{\mathrm{E}}_{\boldsymbol{x}\sim q_{\mathrm{data}}(\boldsymbol{x})} [\min(0, -1 + D(\boldsymbol{x}))] + \mathop{\mathrm{E}}_{\boldsymbol{z}\sim p(\boldsymbol{z})} \left[\min\left(0, -1 - D\left(\hat{G}(\boldsymbol{z})\right)\right)\right]$$

$$V_G(G, \hat{D}) = -\mathop{\mathrm{E}}_{\boldsymbol{z}\sim p(\boldsymbol{z})} \left[\hat{D}\left(G(\boldsymbol{z})\right)\right],$$

- "Hinge loss" for the discriminator D
  - $D(x)$ the larger the better
  - $D(G(z))$ the smaller the better
- "Linear loss" for the generator G
  - $D(G(z))$ the larger the better

Ref: Lim & Ye, "Geometric GAN," arXiv 2017

Ref: Miyato et al, "Spectral normalization for generative adversarial networks," ICLR 2018

GANs vs VAEs

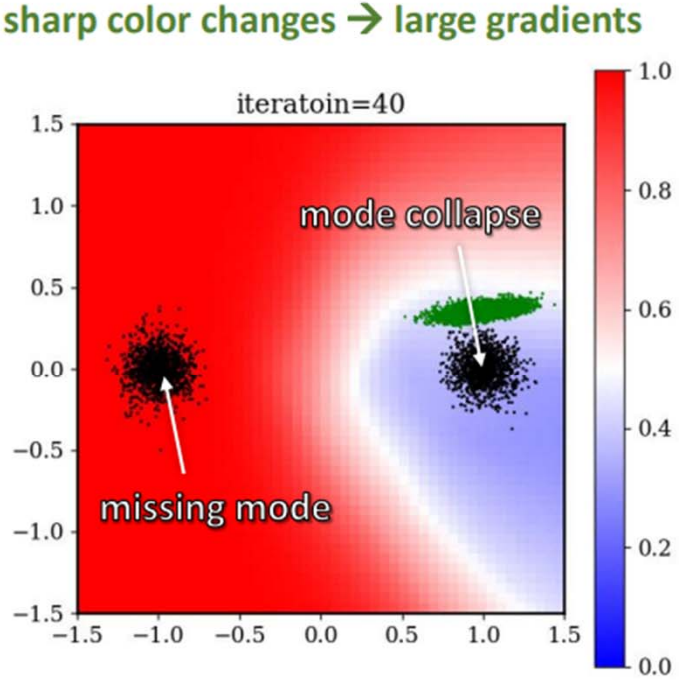
https://salu133445.github.io/ismir2019tutorial/

	GAN	VAE
Objective	(generator) fool the discriminator (discriminator) tell real data from fake ones	reconstruct real data using pixel-wise loss
Results	tend to be sharper 	tend to be more blurred 
Diversity	Higher	Lower
Stability	Lower	Higher

GAN: Mode Collapse and Missing Mode

https://salu133445.github.io/ismir2019tutorial/

- **Key**—discriminator provides generator with gradients as a guidance for improvement
  - Discrimination is easier than generation
  - Discriminator tends to provide large gradients
  - Result in unstable training of the generator
- Common failure cases
  - Mode collapse
  - Missing modes

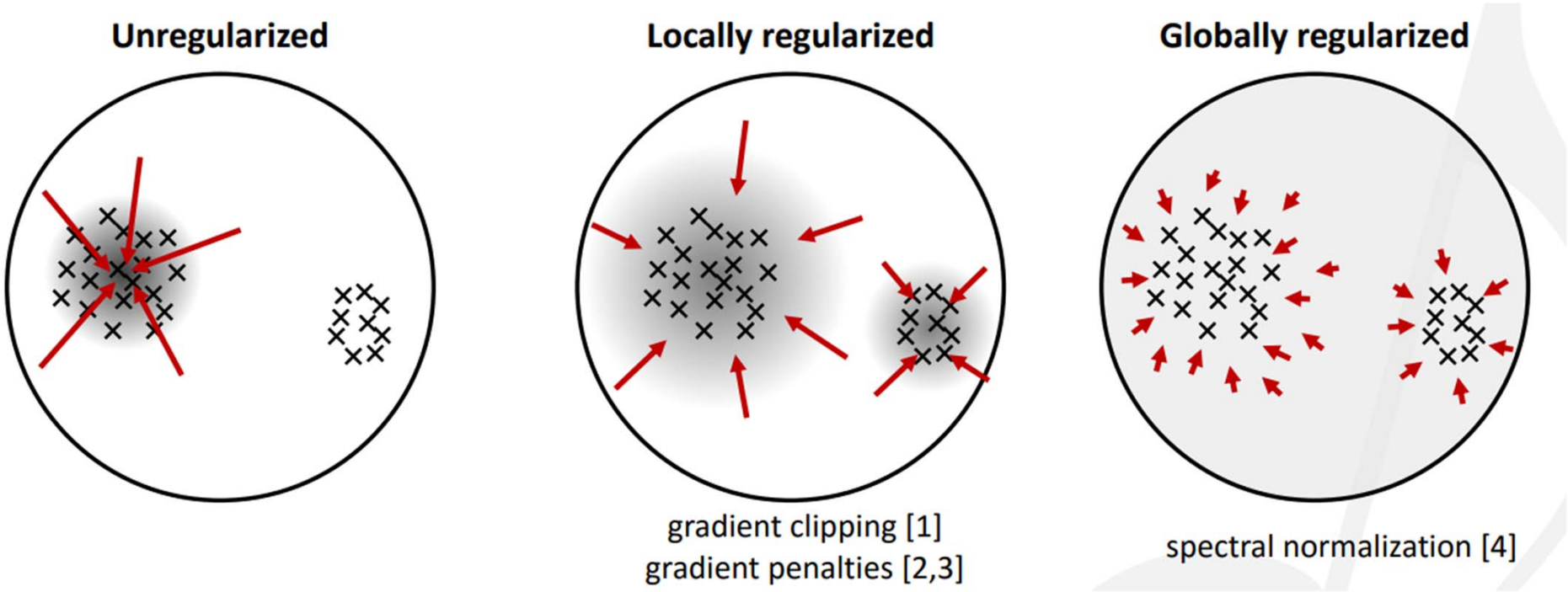


(Colors show the outputs of the discriminator)

- Also see: https://github.com/soumith/ganhacks

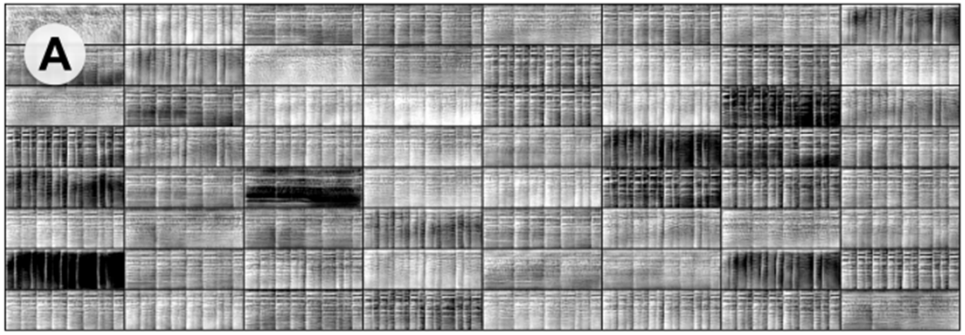
GAN: Regularizing GANs

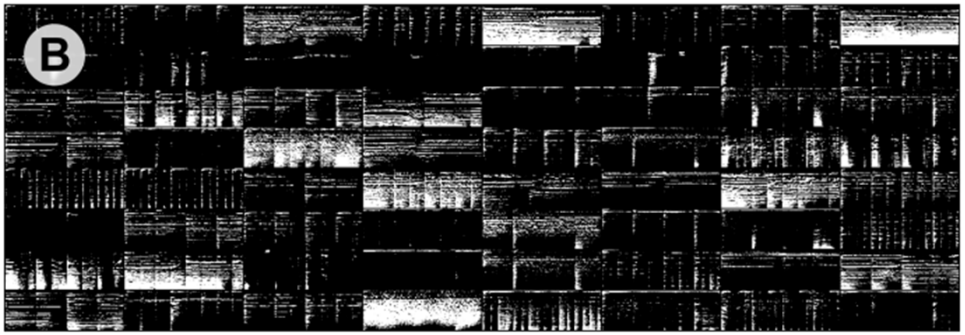
https://salu133445.github.io/ismir2019tutorial/

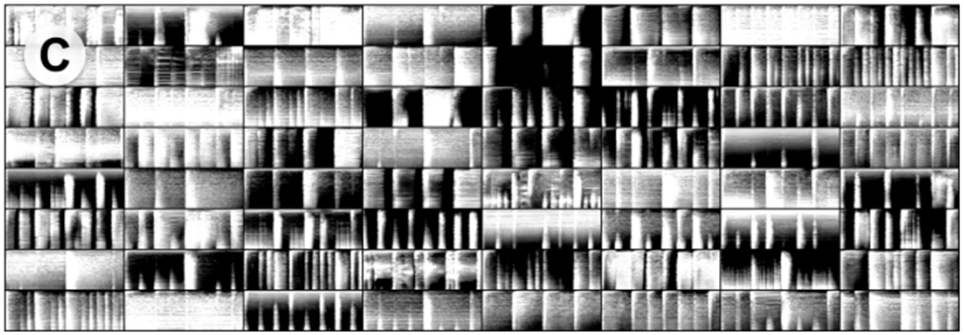


GAN: Gradient Vanishing

- Gradient *vanishes* when the discriminator is too strong







- Examples from Mel-spectrogram generation (not Mel-vocoder)
  - (A) Mode collapse
  - (B) Gradient vanishing
  - (C) Successful training

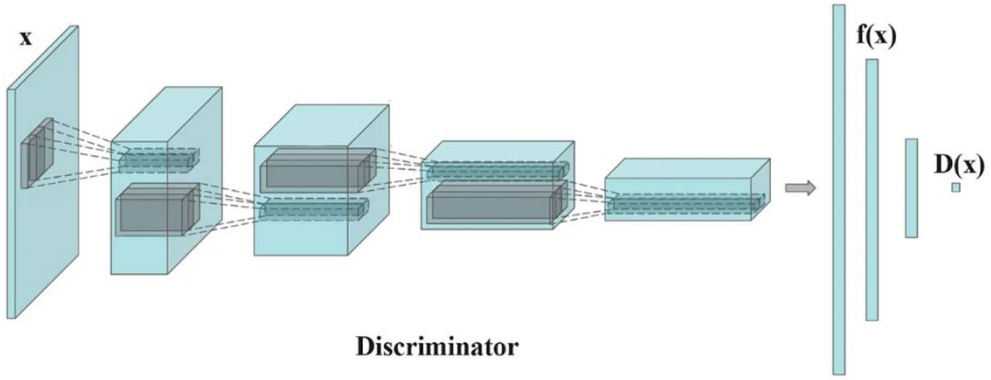
Ref: Yeh et al, "Exploiting pre-trained feature networks for generative adversarial networks in audio-domain loop generation," ISMIR 2022

GAN: Feature Matching Loss

https://paperswithcode.com/method/feature-matching

- A regularizing objective for a generator in GANs that prevents it from **overtraining** on the current discriminator
- Train the generator to match the expected value of the features on an **intermediate layer** of the discriminator
  - measure the L2-distance between the **means** of the feature vectors of the real data and the generated ones
  - $f(.)$ is the feature vector extracted in an immediate layer by the discriminator

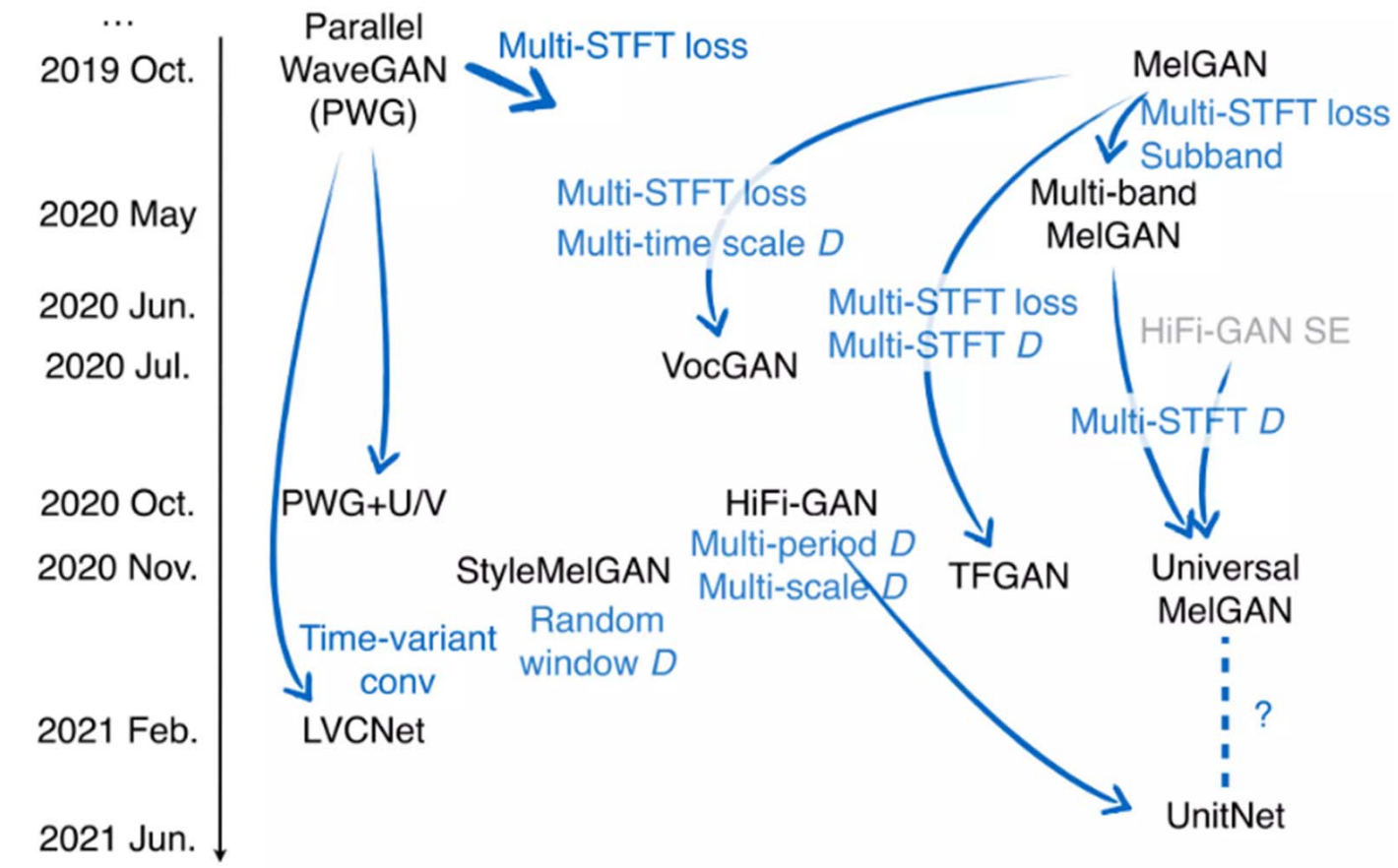
$$||\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}\mathbf{f}(\boldsymbol{x}) - \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}\mathbf{f}(G(\boldsymbol{z}))||_2^2$$



Outline

- General idea
- DL components: upsampling layers and GAN
- **GAN-based Mel-vocoders**
- More recent work

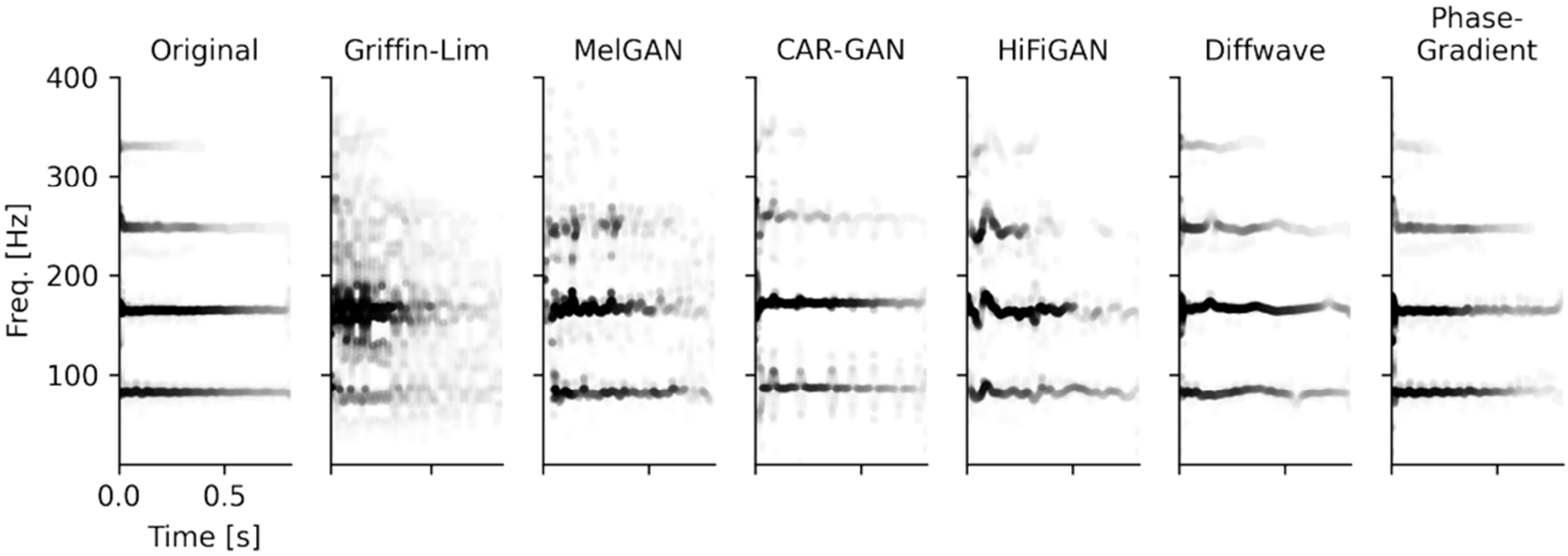
GAN-based Models



Ref: Figure from https://www.slideshare.net/jyamagis/advancements-in-neural-vocoders (page 112)

GAN-based Models

- Great improvement over the years



Ref: Di Giorgi et al., “Mel spectrogram inversion with stable pitch,” ISMIR 2022

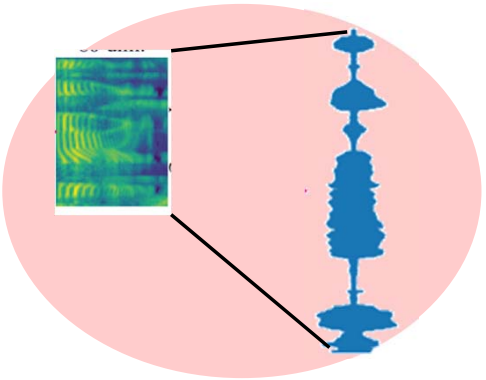
GAN-based Models

- Let's talk about the following models
  - MelGAN (arXiv:1910.06711; NeurIPS'19)
  - MB-MelGAN (arXiv:2005.05106; INTERSPEECH'20)
  - HiFiGAN (arXiv:2010.05646; NeurIPS'20)
  - iSTFTNet (arXiv:2203.02395; ICASSP'22)

MelGAN (NeurIPS'19)

https://github.com/descriptinc/melgan-neurips/blob/master/melgan_slides.pdf

- First GAN-based, non-autoregressive Mel-Vocoder
  - ***Feedforward***: upsamples the input to create the output
- Smaller and **much faster**



Model	Number of parameters (in millions)	Speed on CPU (in kHz)	Speed on GPU (in kHz)
Wavenet (Shen et al., 2018)	24.7	0.0627	0.0787
Clarinet (Ping et al., 2018)	10.0	1.96	221
WaveGlow (Prenger et al., 2019)	87.9	1.58	223
MelGAN (ours)	**4.26**	**51.9**	**2500**

Ref: Kumar et al., "MelGAN: Generative adversarial networks for conditional waveform synthesis," NeurIPS 2019

MelGAN

https://github.com/descriptinc/melgan-neurips/blob/master/melgan_slides.pdf

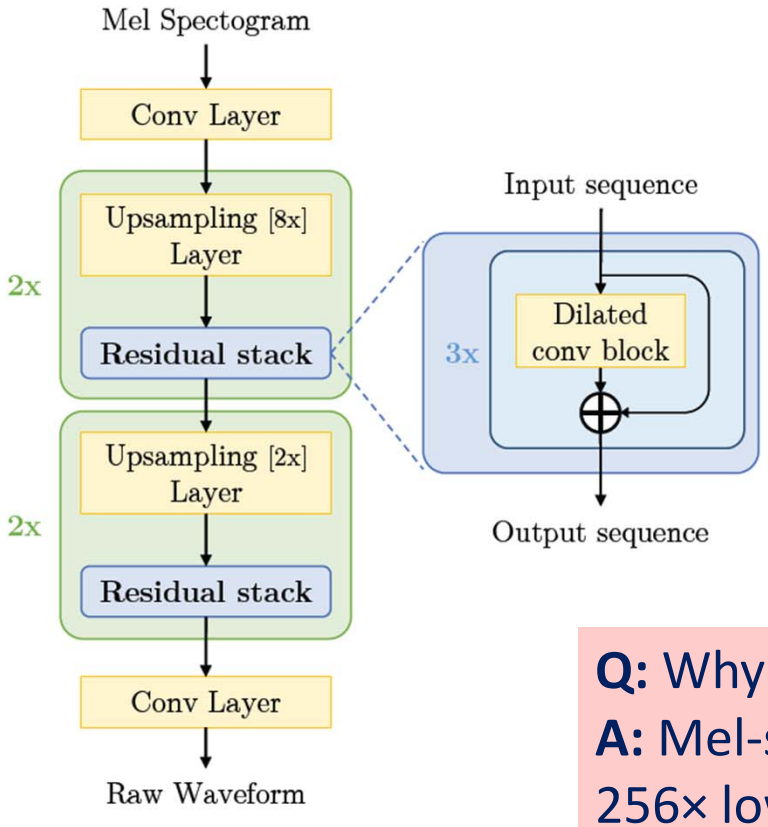
- First GAN-based, non-autoregressive Mel-Vocoder
- Smaller and much faster than competing methods at the time
- But a bit less accurate than WaveNet and flow-based models such as WaveGlow

Model	MOS	95% CI
Griffin Lim	1.57	±0.04
WaveGlow	4.11	±0.05
WaveNet	4.05	±0.05
MelGAN	3.61	±0.06
Original	**4.52**	± **0.04**

Ref: Kumar et al., "MelGAN: Generative adversarial networks for conditional waveform synthesis," NeurIPS 2019

MelGAN: Generator

https://github.com/descriptinc/melgan-neurips/blob/master/melgan_slides.pdf



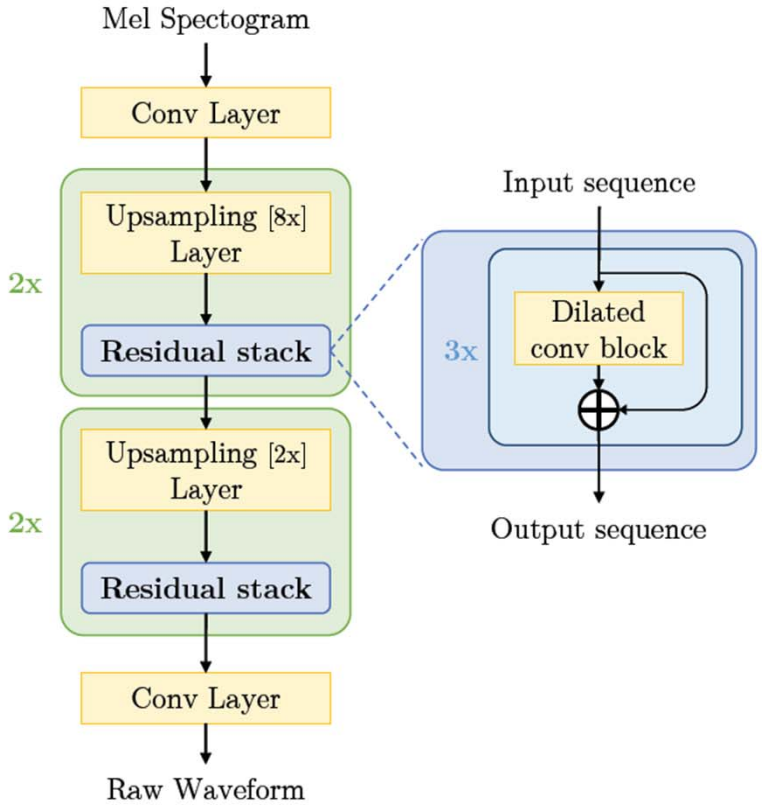
Architecture

- Stack of transposed convolutional layers to upsample the input sequence.
- Each transposed convolutional layer followed by a stack of residual blocks.

Q: Why [8x8x2x2]?
A: Mel-spectrogram (used for all experiments) is at a 256× lower temporal resolution

MelGAN: Generator

https://github.com/descriptinc/melgan-neurips/blob/master/melgan_slides.pdf

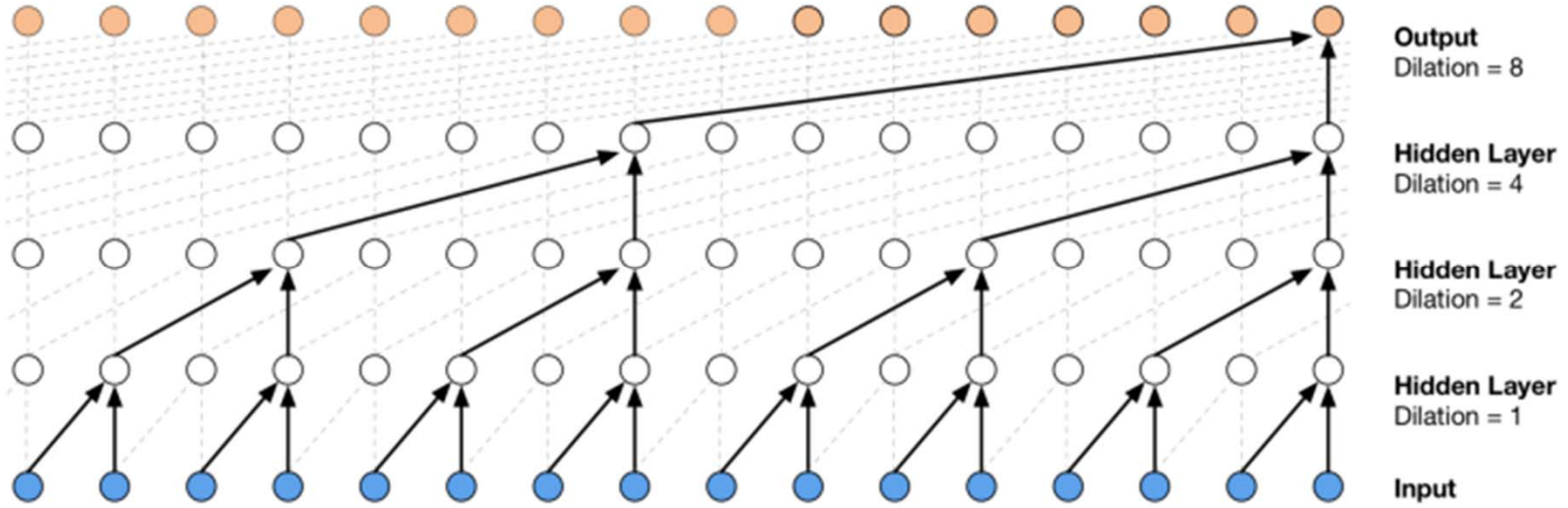


Induced Receptive Field

- **Residual blocks with dilations** so temporally far output activations of each layer has significant overlapping inputs.
- Receptive field of a stack of dilated convolution layers **increases exponentially** with the number of layers.

Dilation?

- **Dilated convolution** (≠ transposed convolution)
  - Increase reception field exponentially with linearly increasing number of parameters
  - See the WaveNet paper
- Stride ≠ dilation



Ref: van den Oord et al, "Wavenet: A generative model for raw audio," ISCA 2016

MelGAN: Generator

https://github.com/descriptinc/melgan-neurips/blob/master/melgan_slides.pdf

Checkerboard Artifacts

- Kernel-size as a **multiple** of stride
- Dilation grows as a **power** of the kernel-size
- Receptive field of the stack looks like a fully balanced (seeing input uniformly) and symmetric tree

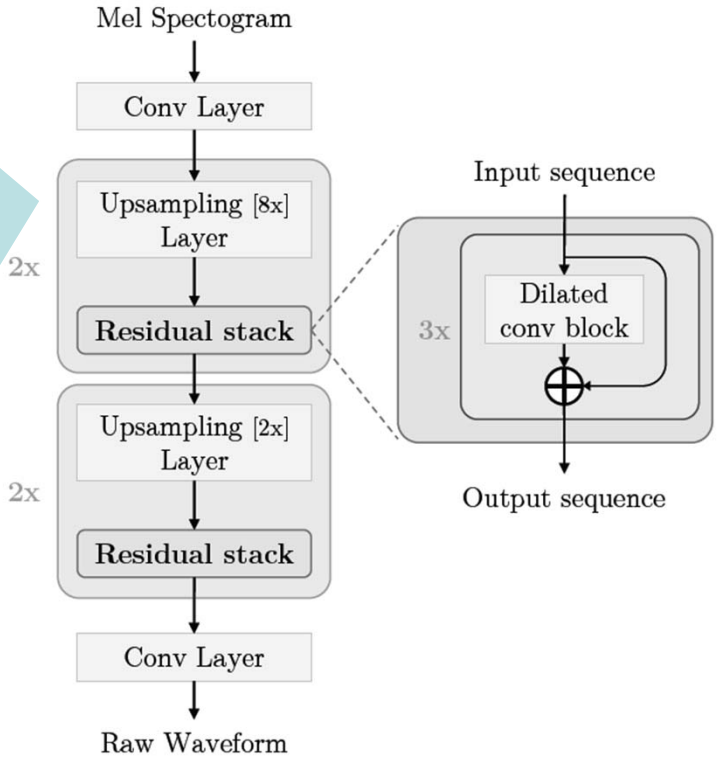
Normalization

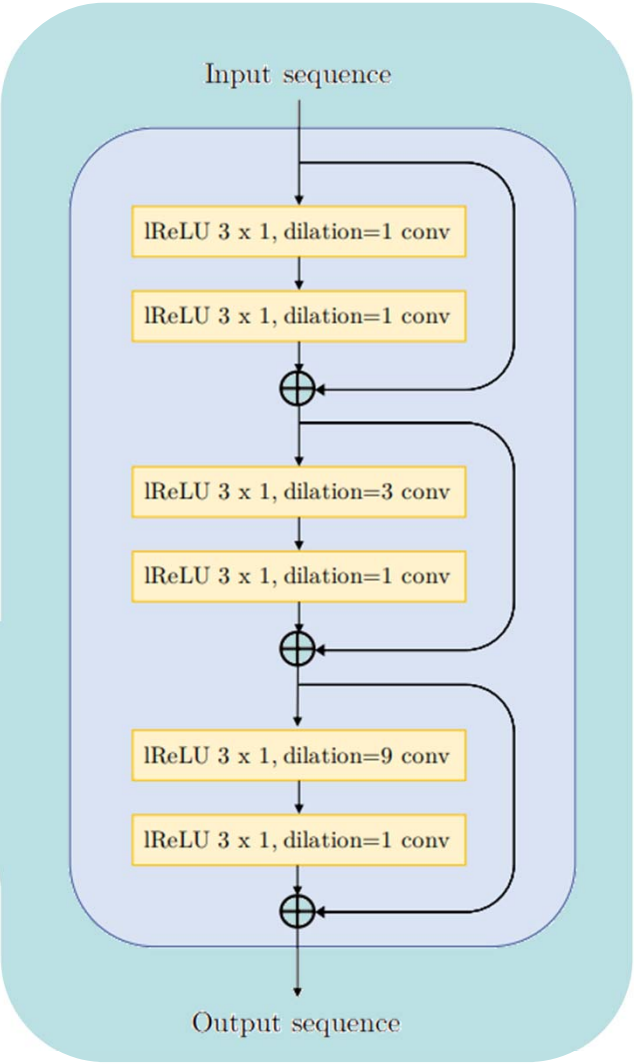
- Instance-norm washes away pitch information, making audio sound metallic
- Spectral-norm's strong Lipshitz constraint badly impacts feature matching objective
- **Weight-norm** works best

MelGAN: Generator

7×1, stride=1 conv 512
lReLU 16×1, stride=8 conv transpose 256
Residual Stack 256
lReLU 16×1, stride=8 conv transpose 128
Residual Stack 128
lReLU 4×1, stride=2 conv transpose 64
Residual Stack 64
lReLU 4×1, stride=2 conv transpose 32
Residual Stack 32
lReLU 7×1, stride=1 conv 1 Tanh

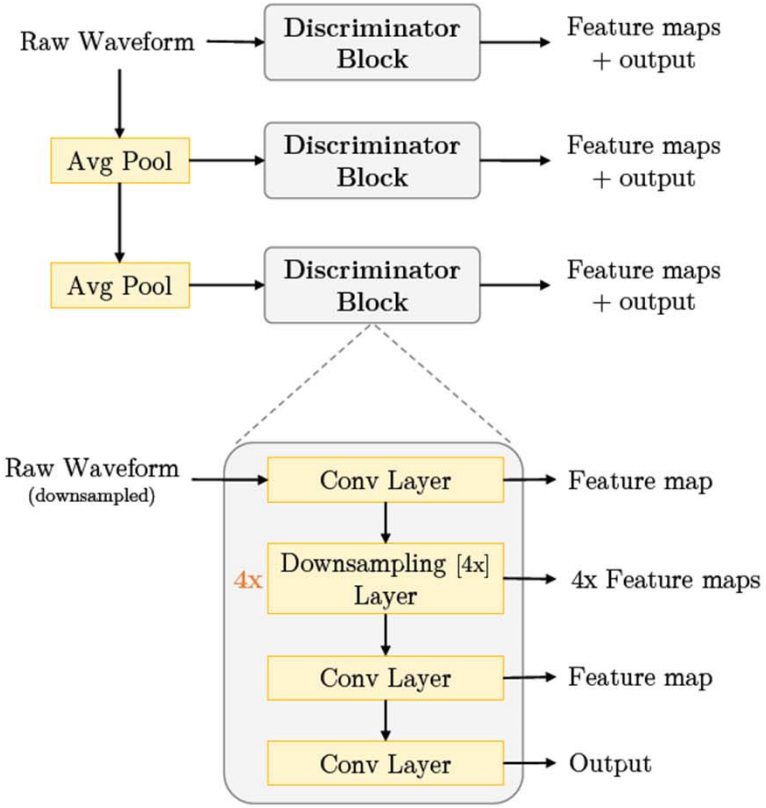
(a) Generator Architecture





MelGAN: Discriminator

https://github.com/descriptinc/melgan-neurips/blob/master/melgan_slides.pdf



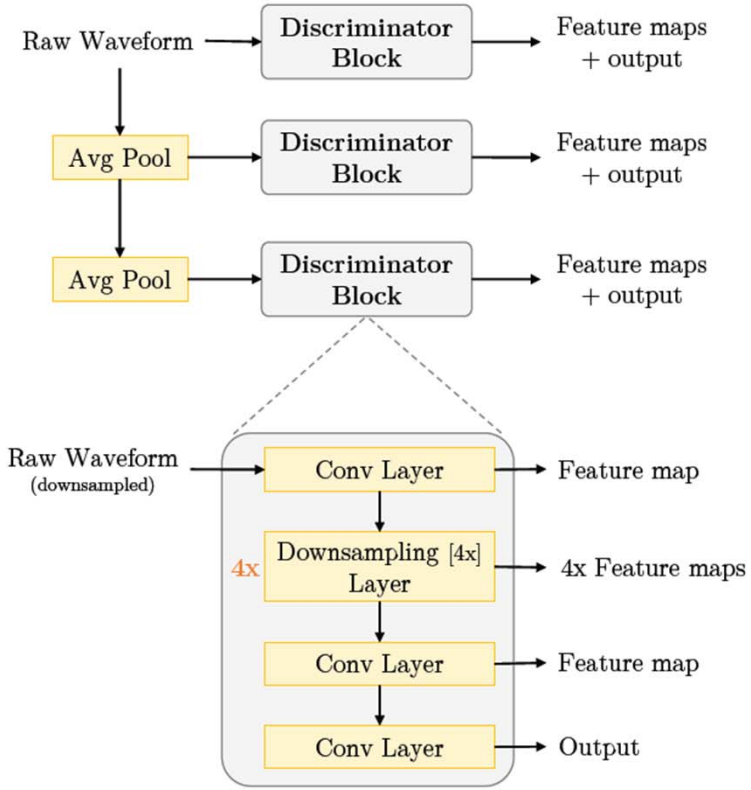
Multiscale Architecture

- 3 discriminators (identical structure) operate on different audio scales -- original scale, 2x and 4x downsampled.
- Each discriminator biased to learn features for **different frequency range** of the audio.

"For example, the discriminator operating on downsampled audio does not have access to high frequency component, hence, it is biased to learn discriminative features based on low frequency components only."

MelGAN: Discriminator

https://github.com/descriptinc/melgan-neurips/blob/master/melgan_slides.pdf



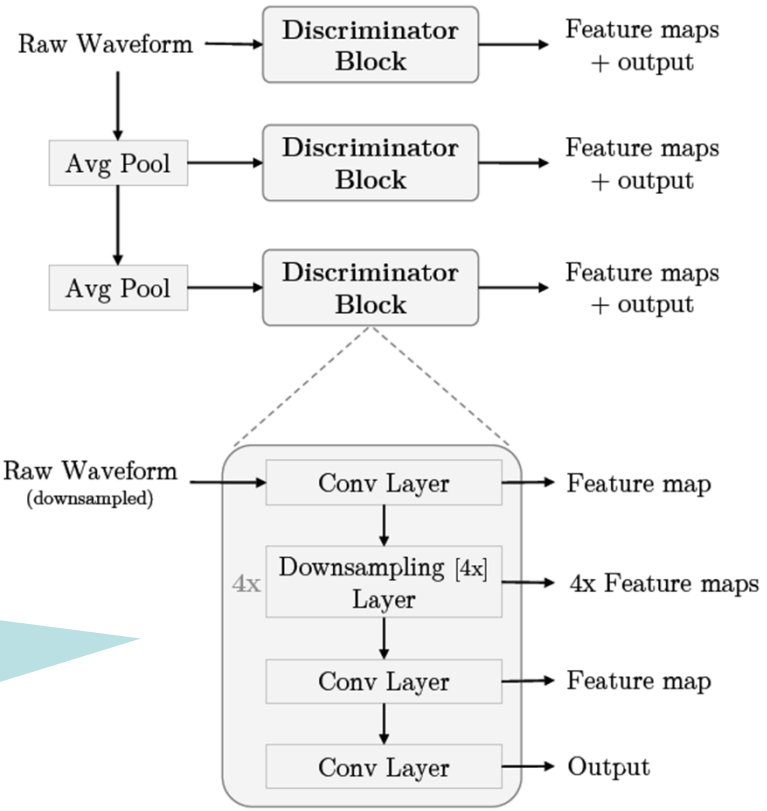
Window-based objective

- Each individual discriminator is a Markovian window-based discriminator (analogues to image patches, Isola et al. (2017))
- Discriminator learns to classify between distributions of **small audio chunks**.
- Overlapping large windows maintain coherence across patches

MelGAN: Discriminator

15 × 1, stride=1 conv 16 lReLU
41 × 1, stride=4 groups=4 conv 64 lReLU
41 × 1, stride=4 groups=16 conv 256 lReLU
41 × 1, stride=4 groups=64 conv 1024 lReLU
41 × 1, stride=4 groups=256 conv 1024 lReLU
5 × 1, stride=1 conv 1024 lReLU
3 × 1, stride=1 conv 1

(b) Discriminator Block Architecture



MelGAN: Loss Functions

https://github.com/descriptinc/melgan-neurips/blob/master/melgan_slides.pdf

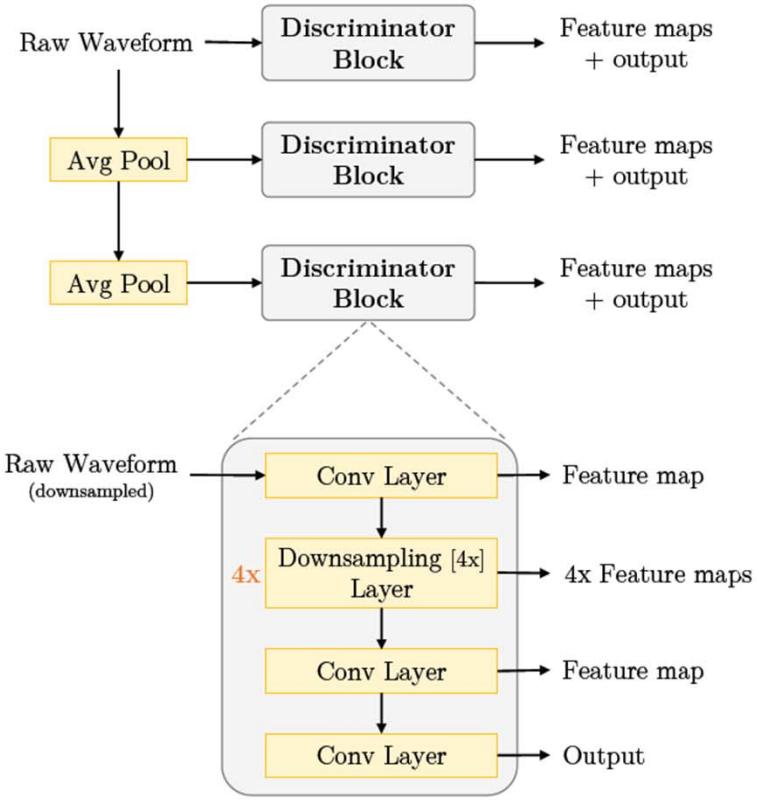
We use the hinge loss formulation (Lim & Ye, 2017; Miyato et al., 2018):

$$\min_{D_k} \mathbb{E}_x \Big[\min(0, 1 - D_k(x)) \Big] + \mathbb{E}_{s,z} \Big[\min(0, 1 + D_k(G(s,z))) \Big], \ \forall k = 1, 2, 3$$

$$\min_{G} \mathbb{E}_{s,z} \Bigg[\sum_{k=1,2,3} -D_k(G(s,z)) \Bigg]$$

MelGAN: Loss Functions

https://github.com/descriptinc/melgan-neurips/blob/master/melgan_slides.pdf



We additionally use a feature matching objective (Larsen et al., 2015) to train the generator:

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

Overall Generator Objective:

$$\min_G\left(\mathbb{E}_{s,z}\left[\sum_{k=1,2,3} -D_k(G(s,z))\right] + \lambda\sum_{k=1}^{3}\mathcal{L}_{\text{FM}}(G, D_k)\right)$$

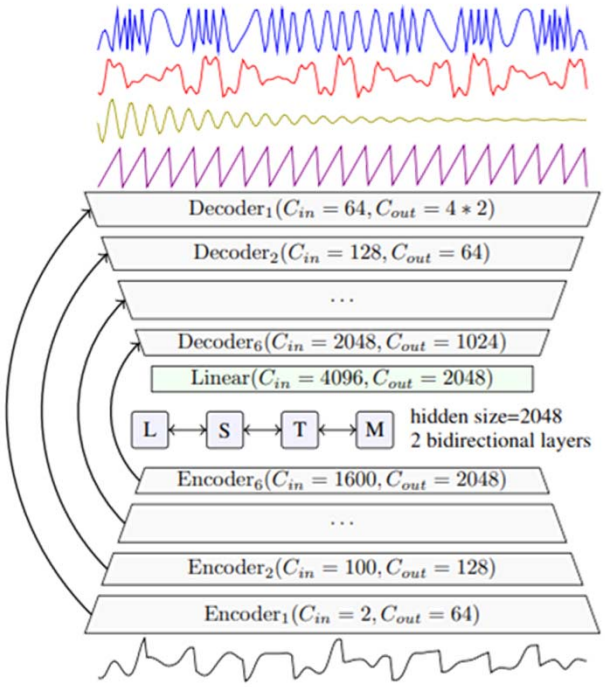
Question: Why Not Waveform-domain Loss Functions?

https://www.soundsandwords.io/audio-loss-functions/

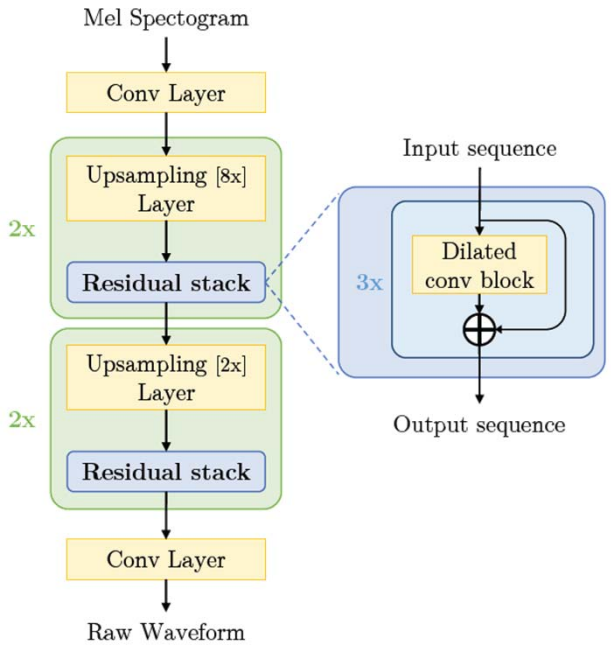
- **Not shift invariant**
  - "If your models comes up with the exact correct output, but it's shifted just 1 millisecond too early or late, a human would consider it a perfect match but error would likely spike. This can especially manifest as phase shift invariance, where two waves have similar frequencies but out-of-sync phases, leading to a large gap in waveform distance."
- Not reflect our **perception** of audio
  - "If training a model for human ears, with L1/L2 you risk overweighting the importance of low frequency sounds"

Question: Why not U-Net?

- **Source separation**
  - Use U-Net
  - Use waveform domain loss

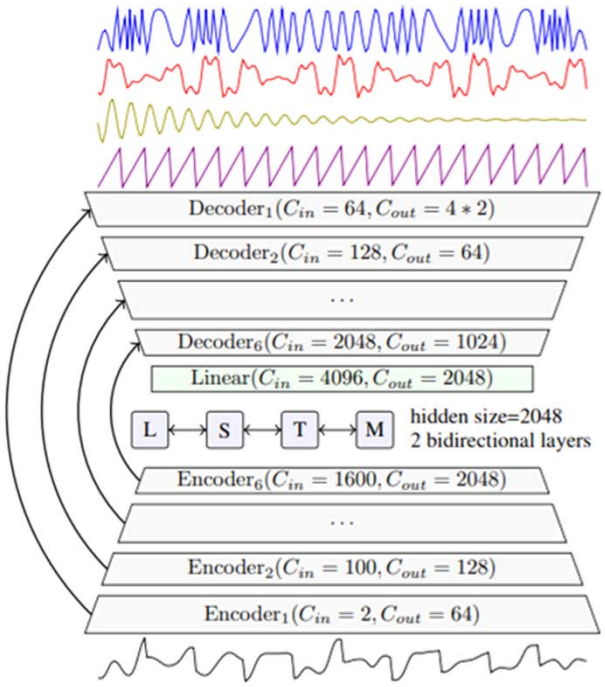


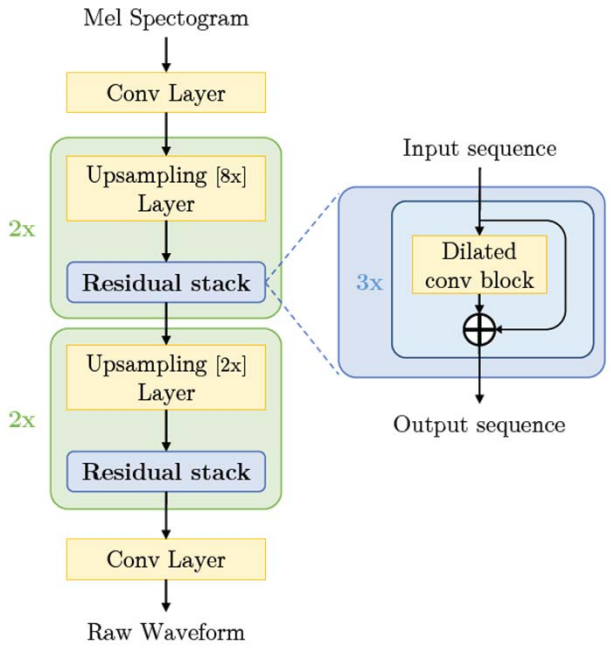
- **Mel-vocoder**
  - Do not use U-Net
  - Dot not use waveform domain loss



Question: Why Not GAN for Source Separation?

- **Source separation**
  - Seldom use GAN
- **Mel-vocoder**
  - Use GAN a lot





MelGAN: Ablation Study

Model	MOS	95% CI
w/ Spectral Normalization	1.33	±0.07
w/ L1 loss (audio space)	2.59	±0.11
w/o Window-based Discriminator	2.29	±0.10
w/o Dilated Convolutions	2.60	±0.10
w/o Multi-scale Discriminator	2.93	±0.11
w/o Weight Normalization	3.03	±0.10
Baseline (MelGAN)	**3.09**	± **0.11**

- "Instance normalization washes away important pitch information, making the audio sound metallic. We also obtained poor results when applying spectral normalization"
- "Adding L1 loss in audio space introduces audible noise that hurts audio quality"

Ref: Kumar et al., "MelGAN: Generative adversarial networks for conditional waveform synthesis," NeurIPS 2019

MB-MelGAN

- Improved version of MelGAN

Index	Model	MOS
F0	MelGAN [20]	3.98±0.04
F1	+ Pretrain G	4.04±0.03
F2	+ $L_{mr\_stft}(G)$	4.06±0.04
F3	+ Deepen ResStack	**4.35±0.05**

1. Increase the receptive field of the generator (F3)
2. Substitute the feature matching loss with the **multi-resolution STFT loss** (F1, F2)
   - Avoid the problems of **time-domain loss**
   - **Pre-train** the generator using STFT loss, then adversarial loss
3. Extend MelGAN with **multiband processing**
   - The generator takes mel-spectrograms as input and produces sub-band signals which are subsequently summed back to full-band signals as discriminator input.

Ref: Yang et al, "Multi-band MelGAN: Faster Waveform Generation for High-Quality Text-to-Speech," INTERSPEECH 2020

Multi-resolution STFT Loss

- Firstly explored elsewhere (e.g., in Parallel WaveGAN)
- Avoid the problems of time-domain loss
- **Multiple resolutions**: different window sizes and hop sizes

$$L_{sc}(x, \widetilde{x}) = \frac{\| |STFT(x)| - |STFT(\widetilde{x})| \|_F}{\| |STFT(x)| \|_F}$$

$$L_{mag}(x, \widetilde{x}) = \frac{1}{N} \| log|STFT(x)| - log|STFT(\widetilde{x})| \|_1$$

$$L_{mr\_stft}(G) = \mathbb{E}_{x,\widetilde{x}}[\frac{1}{M} \sum_{m=1}^{M} (L_{sc}^m(x, \widetilde{x}) + L_{mag}^m(x, \widetilde{x}))]$$

- **Pre-train** the generator using STFT loss, then adversarial loss

HiFiGAN (NeurIPS'20)

https://github.com/jik876/hifi-gan

- A great improved version of MelGAN that outperforms WaveNet, WaveGlow
- Widely-used

	Model	MOS (CI)	Speed on CPU (kHz)	Speed on GPU (kHz)	# Param (M)
	Ground Truth	4.45 (±0.06)	–	–	–
	WaveNet (MoL)	4.02 (±0.08)	–	0.07 (×0.003)	24.73
	WaveGlow	3.81 (±0.08)	4.72 (×0.21)	501 (×22.75)	87.73
	MelGAN	3.79 (±0.09)	145.52 (×6.59)	14,238 (×645.73)	4.26
Highest MOS →	HiFi-GAN $V1$	**4.36** (±0.07)	31.74 (×1.43)	3,701 (×167.86)	13.92
Most light-weight →	HiFi-GAN $V2$	4.23 (±0.07)	214.97 (×9.74)	16,863 (×764.80)	**0.92**
Fastest →	HiFi-GAN $V3$	4.05 (±0.08)	**296.38 (×13.44)**	**26,169 (×1,186.80)**	1.46

Ref: Kong et al., "HiFi-GAN: Generative adversarial networks for efficient and high fidelity speech synthesis," NeurIPS 2020

HiFiGAN: Generator

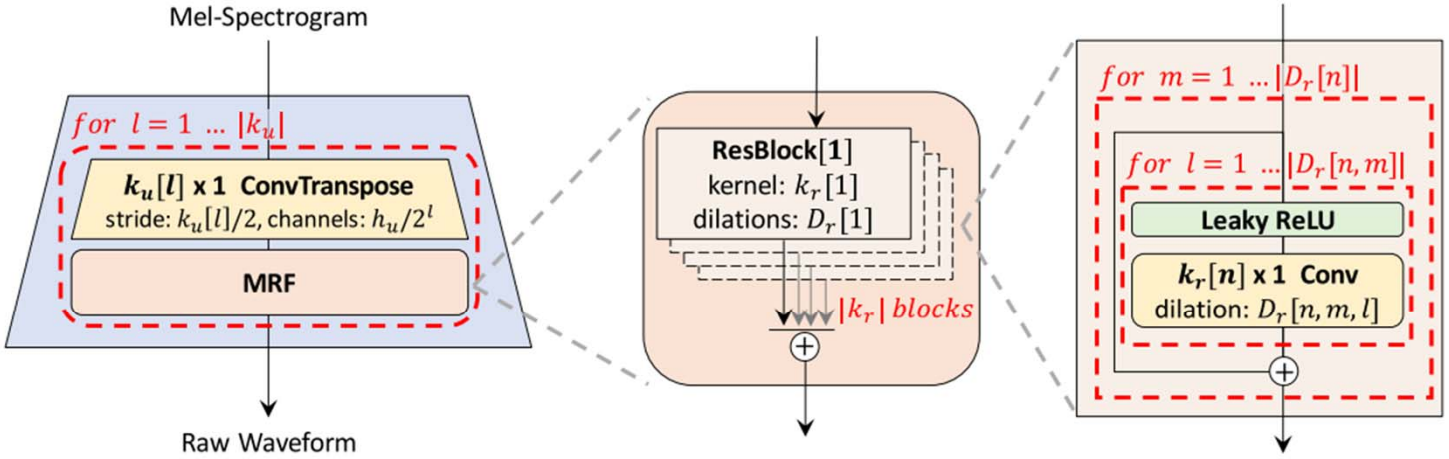


Figure 1: The generator upsamples mel-spectrograms up to $|k_u|$ times to match the temporal resolution of raw waveforms. A MRF module adds features from $|k_r|$ residual blocks of different kernel sizes and dilation rates. Lastly, the n-th residual block with kernel size $k_r[n]$ and dilation rates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

- Multi-receptive field fusion (MRF)
- Different **kernel sizes** and **dilation rates** are selected for each **residual block** to form diverse receptive field patterns

HiFiGAN: Generator

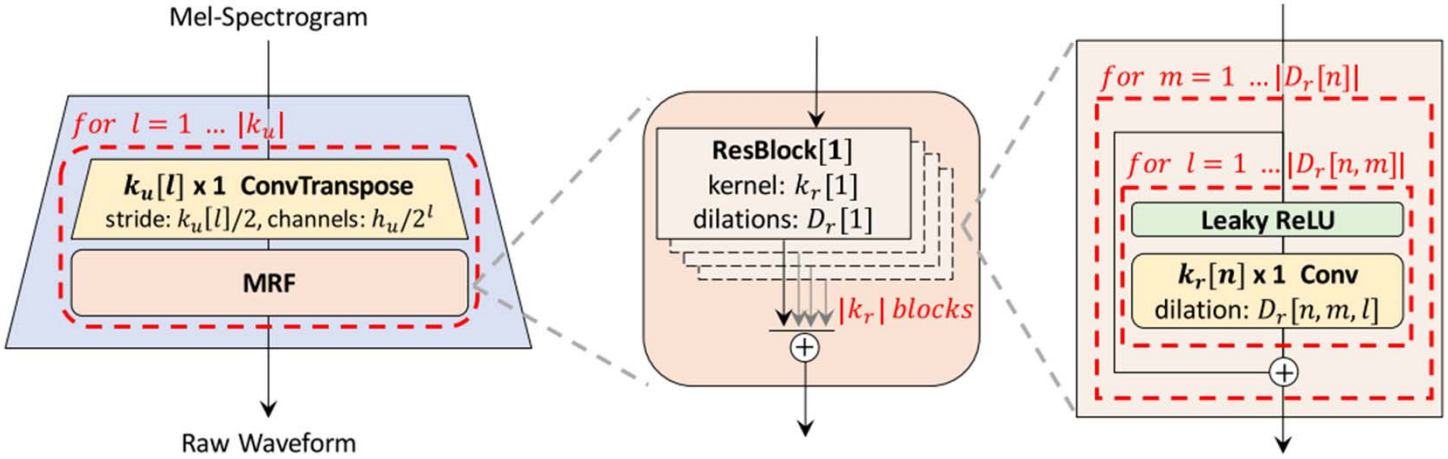
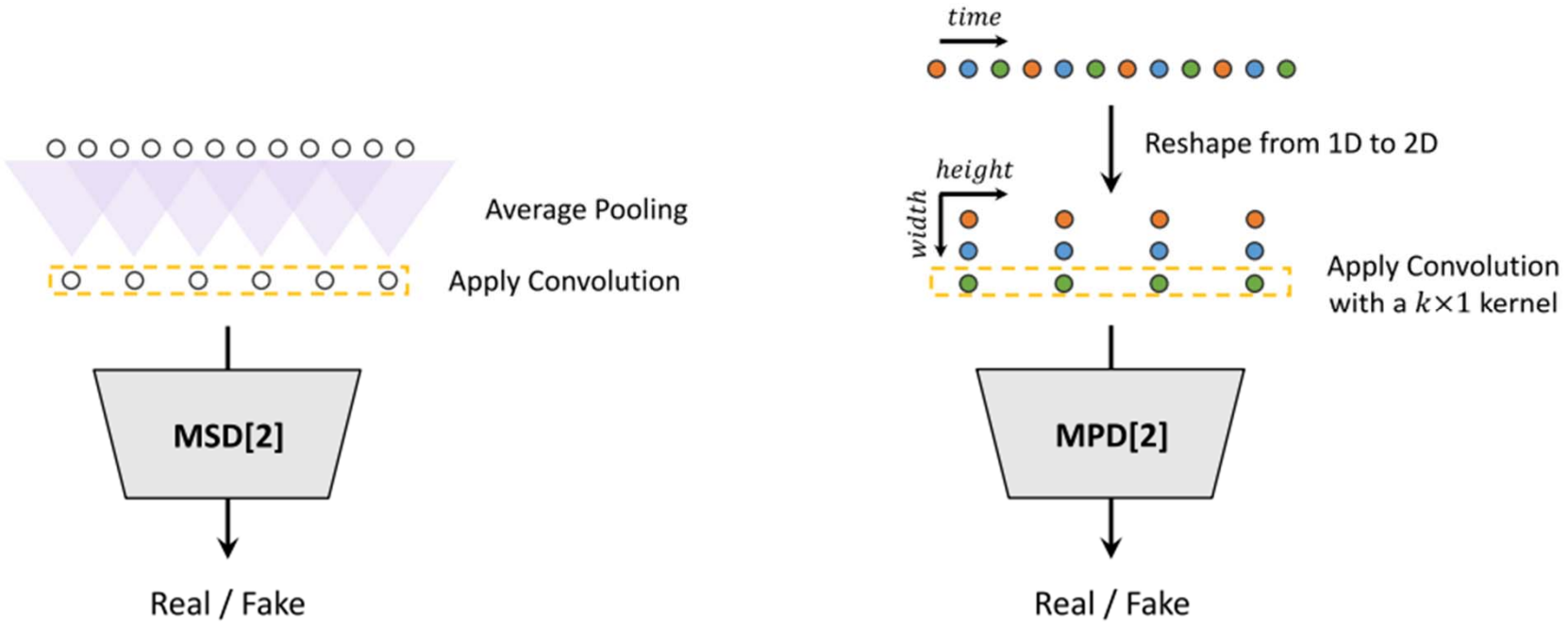


Table 5: Hyper-parameters of three generator $V1$, $V2$, and $V3$.

Model	Hyper-Parameters			
	h_u	k_u	k_r	D_r
$V1$	512	[16, 16, 4, 4]	[3, 7, 11]	[[1, 1], [3, 1], [5, 1]] $\times$3
$V2$	128	[16, 16, 4, 4]	[3, 7, 11]	[[1, 1], [3, 1], [5, 1]] $\times$3
$V3$	256	[16, 16, 8]	[3, 5, 7]	[[1], [2]], [[2], [6]], [[3], [12]]

HiFiGAN: Discriminator



- Multi-scale discriminator (MSD) + a new **multi-period discriminator** (MPD)
  - "MPD is a mixture of sub-discriminators, each of which only accepts equally spaced samples of an input audio [and therefore] looking at different parts of an input audio"
  - "We set the periods to [2, 3, 5, 7, 11] to avoid overlaps as much as possible."

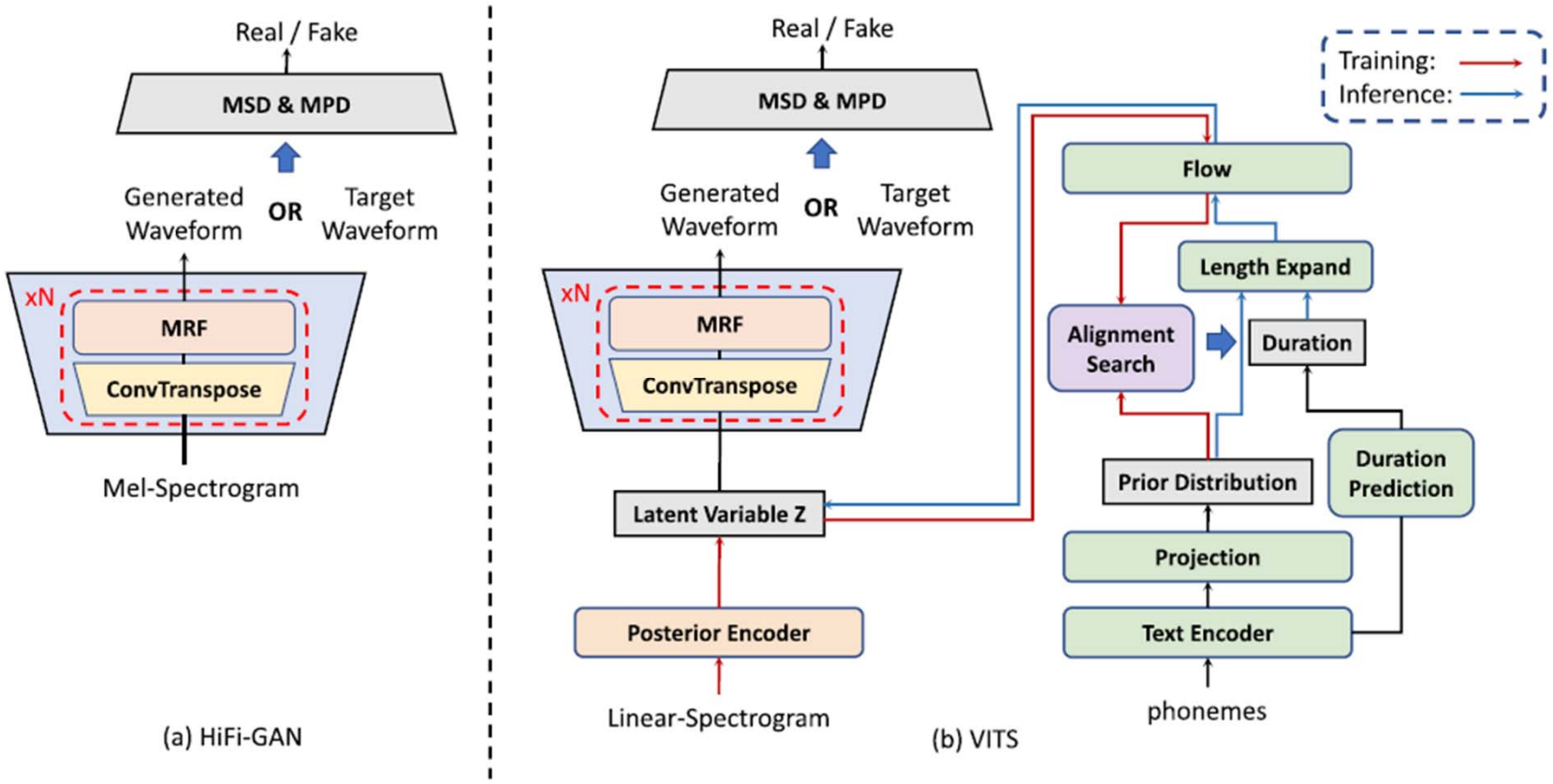
HiFiGAN: Ablation Study

Model	MOS (CI)
Ground Truth	4.57 ($\pm$0.04)
Baseline (HiFi-GAN $V3$)	4.10 ($\pm$0.05)
w/o MPD	2.28 ($\pm$0.09)
w/o MSD	3.74 ($\pm$0.05)
w/o MRF	3.92 ($\pm$0.05)
w/o Mel-Spectrogram Loss	3.25 ($\pm$0.05)
MPD p=[2,4,8,16,32]	3.90 ($\pm$0.05)
MelGAN	2.88 ($\pm$0.08)
MelGAN with MPD	3.35 ($\pm$0.07)

Ref: Kong et al., “HiFi-GAN: Generative adversarial networks for efficient and high fidelity speech synthesis,” NeurIPS 2020

VITS (HiFiGAN-based Architecture for Speech Synthesis)

- By the same group of authors



Ref: Kim et al., “Conditional variational autoencoder with adversarial learning for end-to-end text-to-speech,” ICML 2021

GAN-based Models

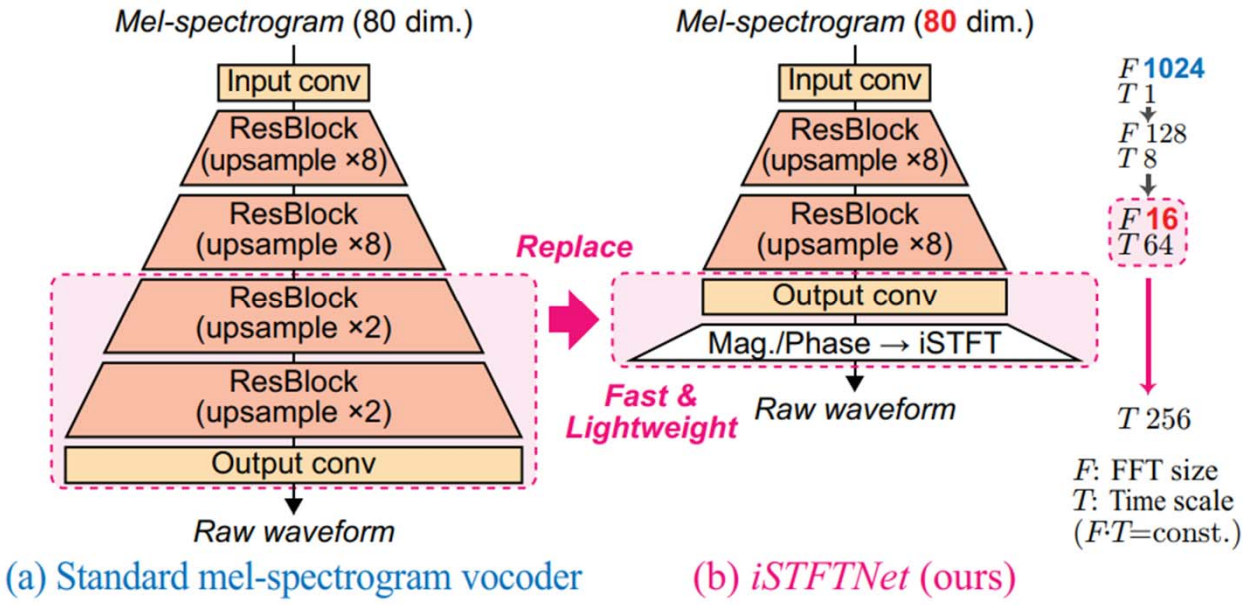
	Generator (*G*)	Discriminator (*D*) loss	Auxiliary loss
Parallel WaveGAN	Single scale	Single scale (target waveform)	**Multi**-reso STFT loss
MelGAN	**Multi-scale** (multiple up-sampling stages) e.g., duplication, conv1d w/ stride	**Multi-scale time domain** (average pooling)	**Multi**-scale feature matching
TFGAN		**Multi-scale time & frequency domain**	**Multi**-reso STFT & **multi**-scale time domain loss
StyleMelGAN		**Multiple** random framing window **&** subband *D*	**Multi**-reso STFT loss
VocGAN		**Multi-scale time domain** (ave. pooling & output at different scale)	
Multi-band MelGAN		**Multi-scale time domain** (ave. pooling)	
UnivNet		**Multi-scale time domain** (ave. pooling & squeeze)	
HiFi-GAN			Single spectral loss

Ref: Figure from https://www.slideshare.net/jyamagis/advancements-in-neural-vocoders (page 127)

iSTFTNet

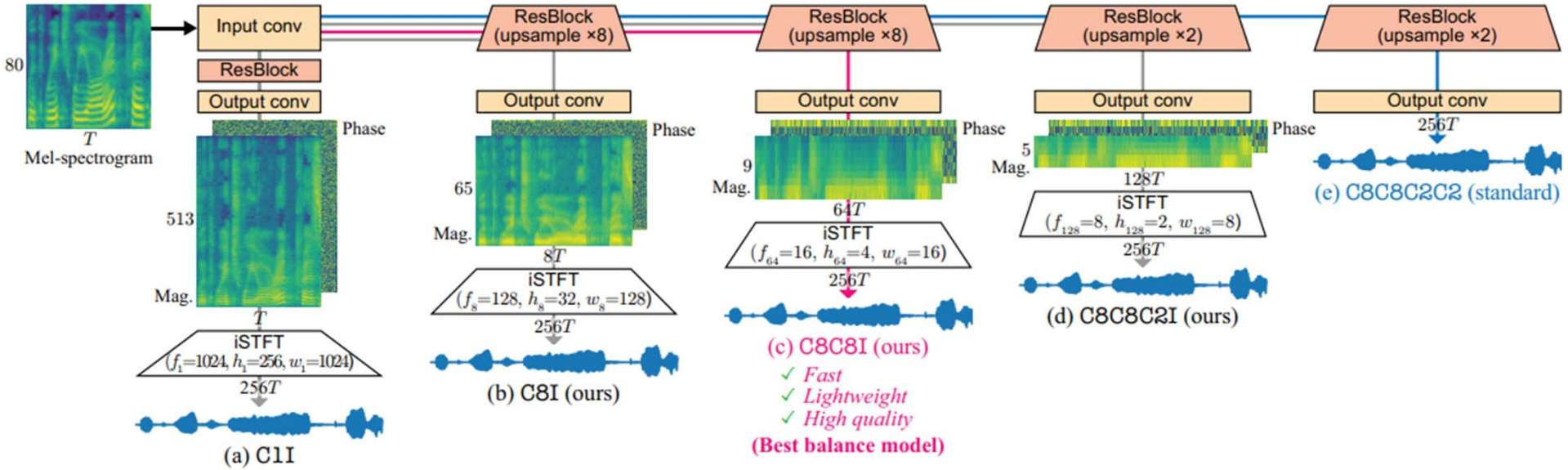
https://www.kecl.ntt.co.jp/people/kaneko.takuhiro/projects/istftnet/

- Improved version of HiFiGAN
- Use inverse STFT layers
  - replaces some layers after sufficiently reducing the frequency dimension using upsampling layers
  - reduces the computational cost from black-box modeling



Ref: Kaneko et al, "iSTFTNet: Fast and lightweight Mel-spectrogram vocoder incorporating Inverse Short-Time Fourier Transform," ICASSP 2022

iSTFTNet



No.	Model	MOS↑	cFW2VD↓	Speed↑ (GPU)	Speed↑ (CPU)	# Param↓ (M)
1	Ground truth	4.46 ±0.14	–	–	–	–
2	V1 (original) [12]	4.22 ±0.17	0.020	×143.59 (100)	×1.34 (100)	13.94 (100)
3	V1-C8C8C2I	4.22 ±0.17	0.018	×179.42 (125)	×1.63 (122)	13.80 (99)
4	V1-C8C8I	4.26 ±0.17	0.020	×245.68 (171)	×2.33 (174)	13.26 (95)
5	V1-C8I	3.32 ±0.22	0.073	×609.43 (424)	×7.57 (565)	10.89 (78)
6	V1-C8C1I	3.82 ±0.17	0.033	×326.39 (227)	×3.97 (296)	19.15 (137)

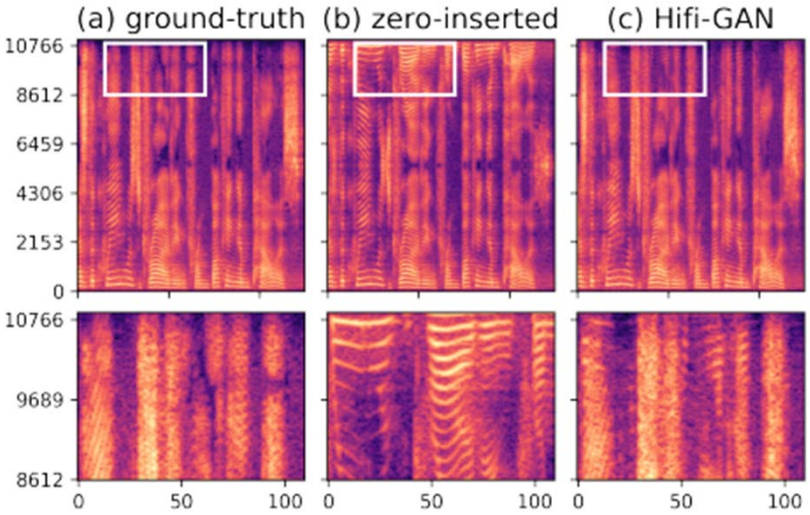
Ref: Kaneko et al, "iSTFTNet: Fast and lightweight Mel-spectrogram vocoder incorporating Inverse Short-Time Fourier Transform," ICASSP 2022

Outline

- General idea
- DL components: upsampling layers and GAN
- DL-based Mel-vocoders
- **Advanced topics**
  - **More on downsampling, upsampling layers**
  - More Mel-Vocoders
  - New trend: Audio codec models

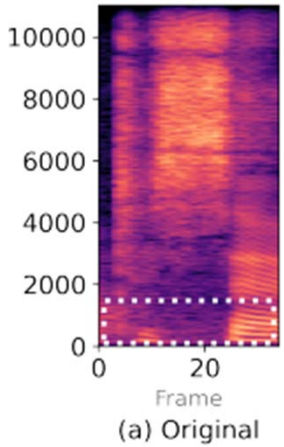
Problems in Neural Upsampling and Downsampling

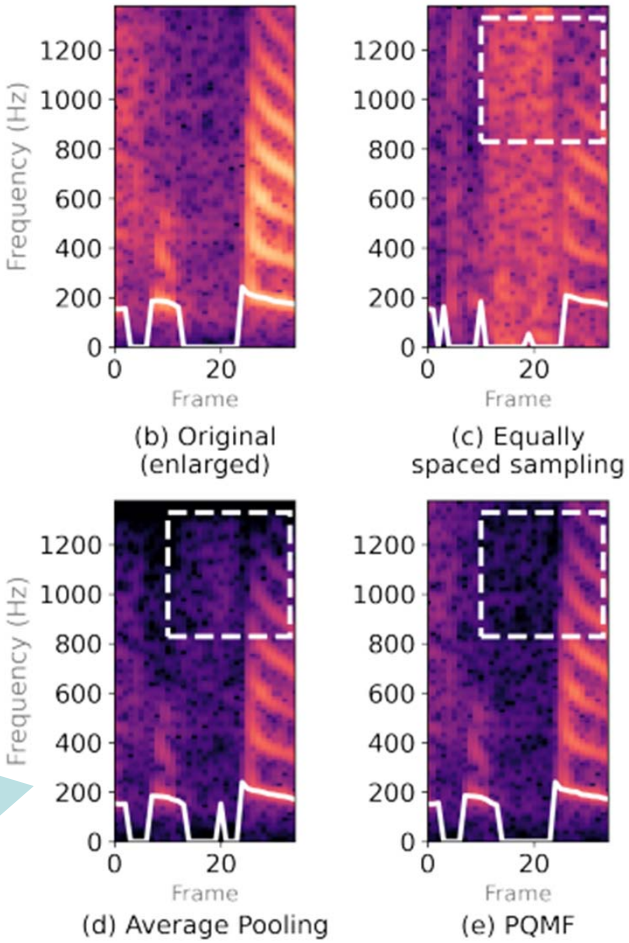
- Upsampling



Artefacts due to “spectral replicas”

- Downsampling





Less aliasing because it’s an LPF + decimation

High-frequency components, which are supposed to be removed, fold back and distort the harmonic frequency components at low-frequency bands

Ref: Bak et al, “Avocodo: Generative adversarial network for artifact-free vocoder,” AAAI 2023

1D Convolution from DSP's Perspective

$$y_{c'}[n'] = \downarrow_H \left(\sum_{c=0}^{C^{(\mathrm{in})}-1} \texttt{xcorr}(w_{c,'c}, x_c)[n] \right)$$

where $\texttt{xcorr}(\cdot,\cdot)$ denotes the cross-correlation of two inputs and $\downarrow_H(\cdot)$ stands for the decimation with a factor of H.

$$y_{c'}[n'] = \downarrow_H \left(\sum_{c=0}^{C^{(\mathrm{in})}-1} (b_{c,'c} * x_c)[n] \right),$$

where $*$ denotes the convolution and $b_{c,'c}[n]$ is given as

$$b_{c,'c}[n] = w_{c,'c}[-n].$$

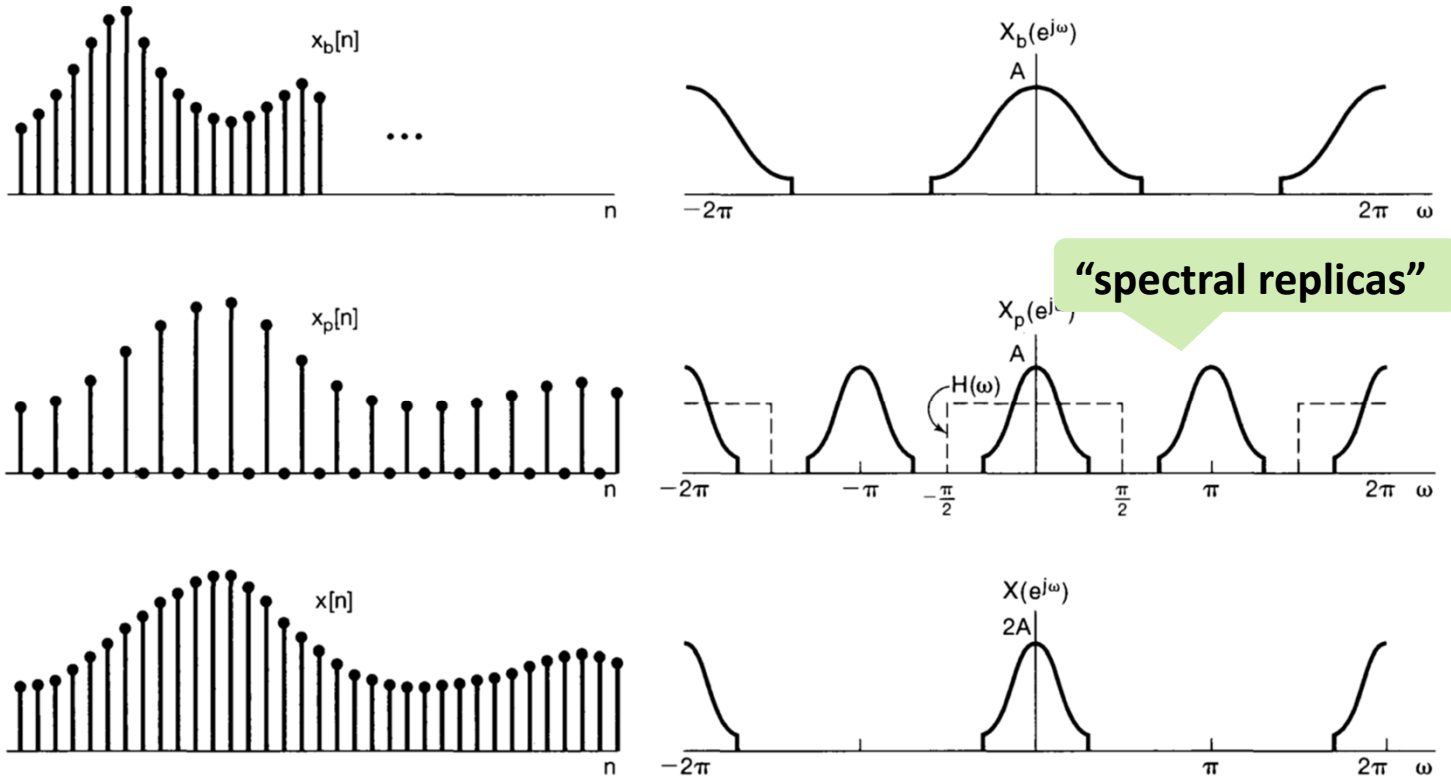
Ref: Saito et al, "Sampling-frequency-independent convolutional layer and its application to audio source separation," TASLP 2022

Upsampling from DSP's Perspective

- **Upsampling = zero insertion + low pass filtering** (LPF)

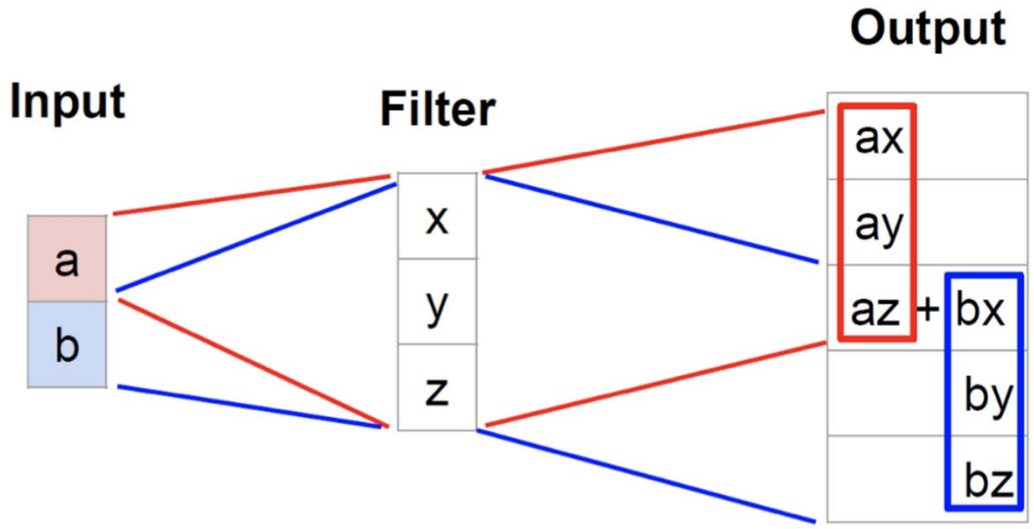
- **Step 1**: zero-insertion
  - Time: $1 \rightarrow N$
  - Freq: $2\pi \rightarrow \frac{2\pi}{N}$

- **Step 2**: DT low-pass filtering
  - remove copies in between (2π)s



Transposed Convolution-based Neural Upsampler

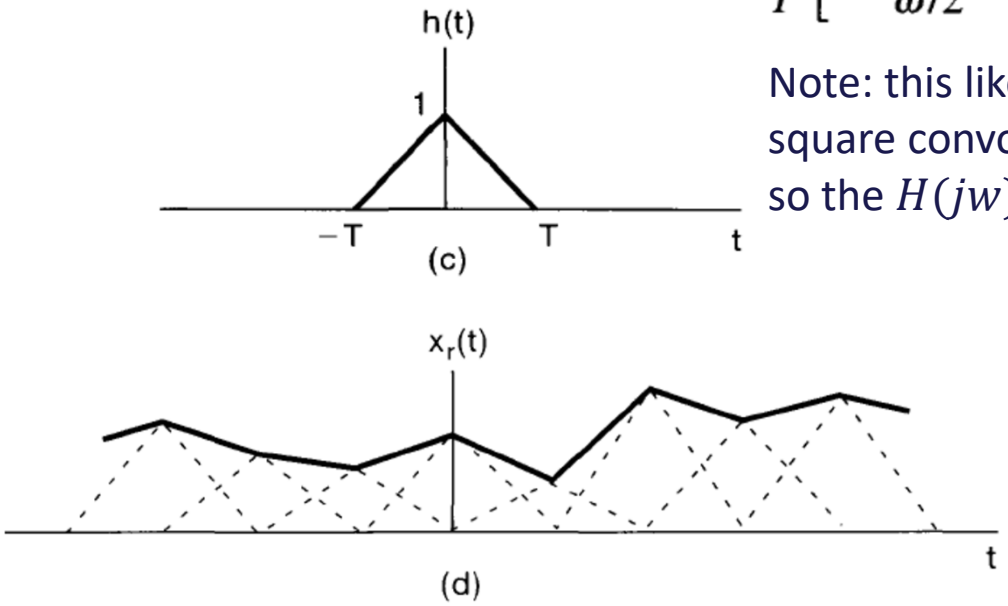
- Upsampling = zero insertion + low pass filtering
- **Transposed convolution = zero insertion + vanilla convolution**



Linear Interpolation-based Neural Upsampler

- Upsampling = zero insertion + low pass filtering
- **Linear interpolation = zero insertion + first-order hold**

$$H(j\omega) = \frac{1}{T}\left[\frac{\sin(\omega T/2)}{\omega/2}\right]^2$$



Note: this like the result of a square convolves by a square; so the $H(jw)$ is like sinc^2

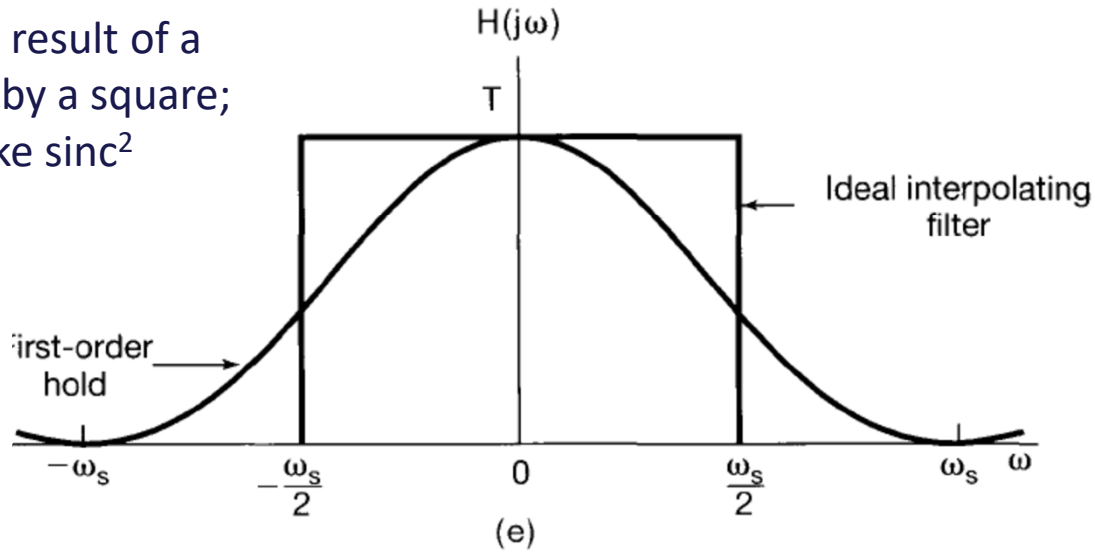
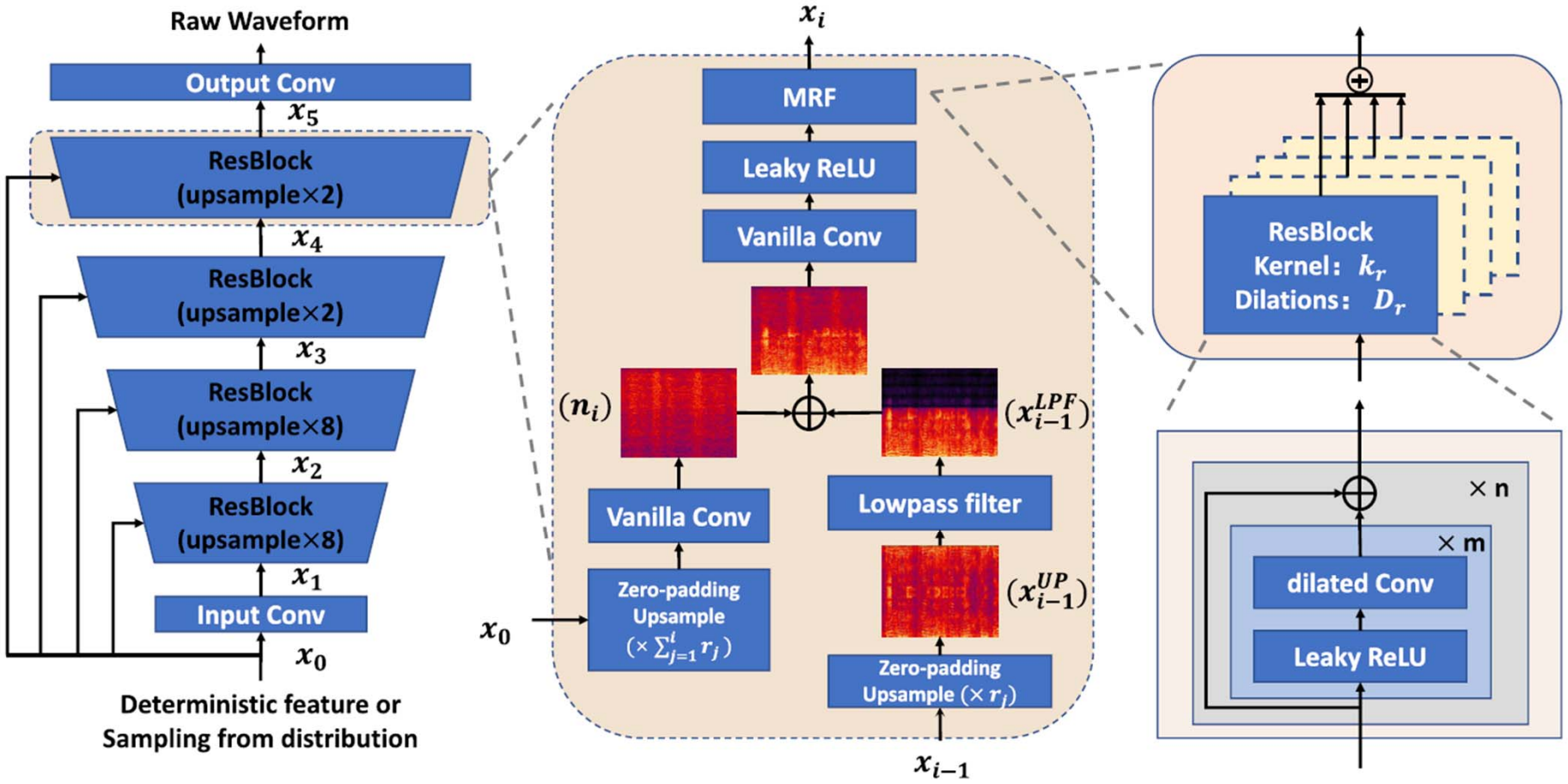


Figure source: Oppenheim et al, *Signals and Systems*

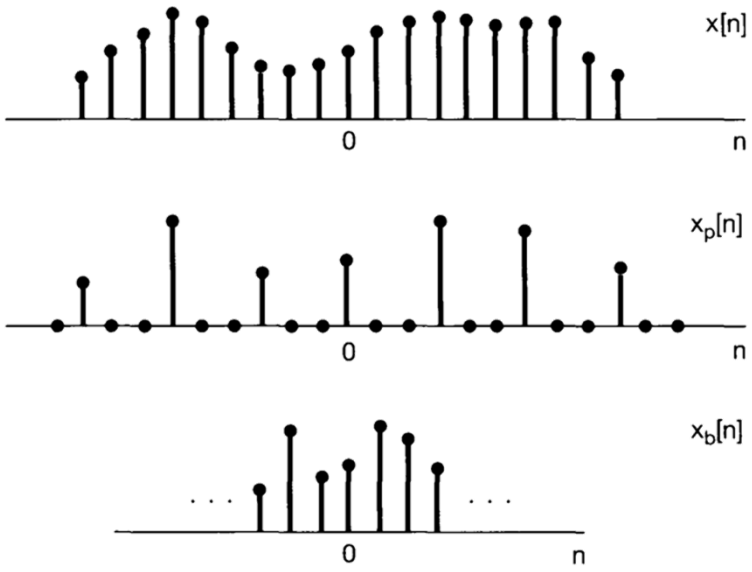
Anti-Aliasing Generator Design



Ref: Shang et al, "Analysis and solution to aliasing artifacts in neural waveform generation models," AA 2023

Downsampling from DSP's Perspective

- **Downsampling = discrete-time sampling + zero removal**
- Sampling rate insufficient → ***aliasing***



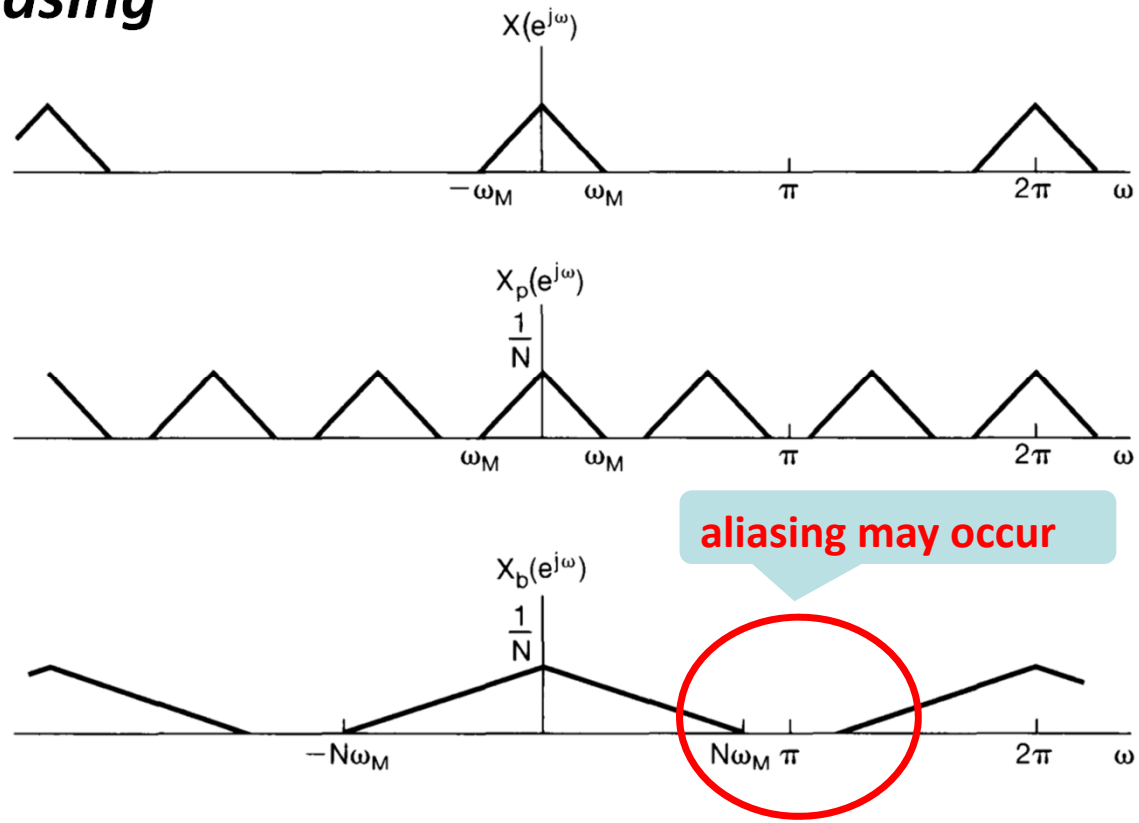
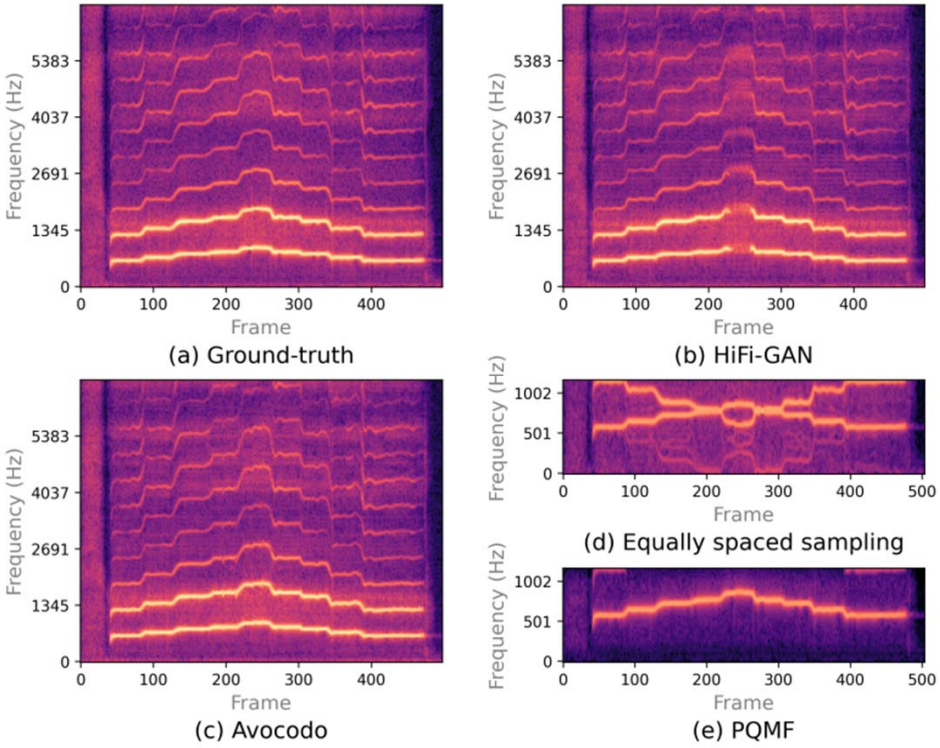
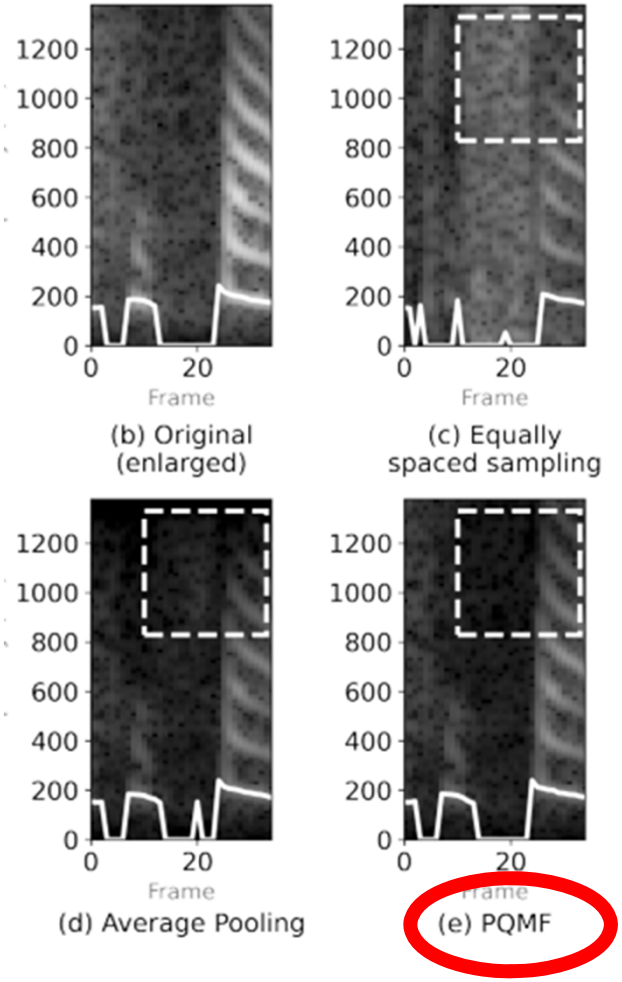


Figure source: Oppenheim et al, *Signals and Systems*

Anti-Aliasing Discriminator Design

- Use a differentiable **pseudo quadrature mirror filter bank** (PQMF) as LPF for down sampling

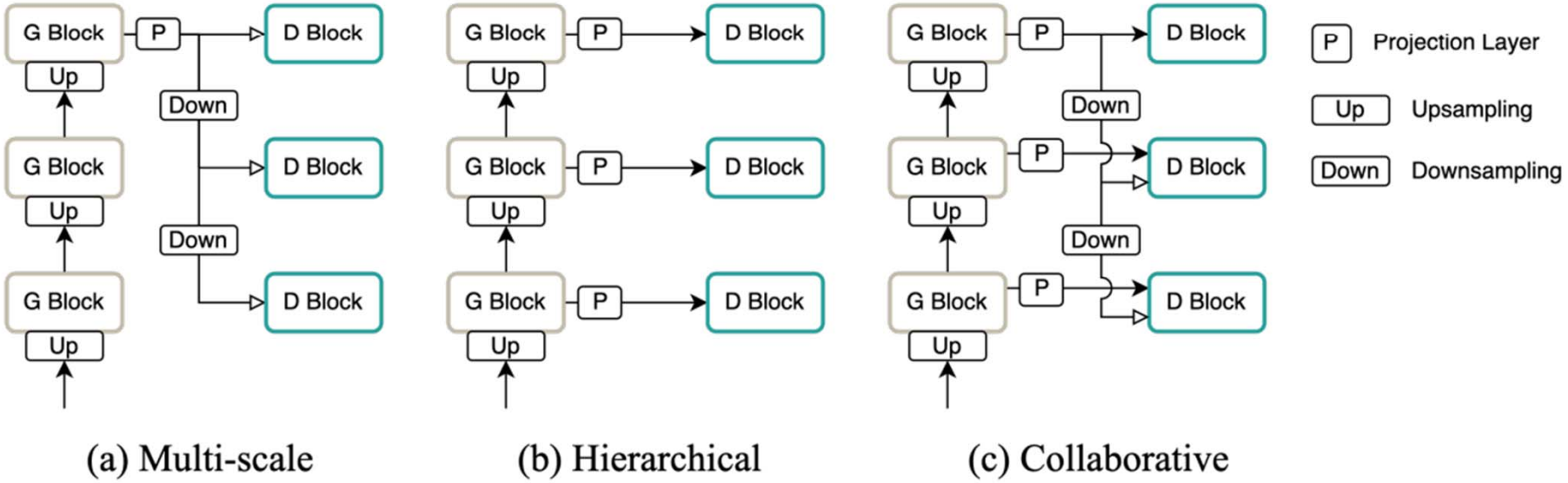




Ref: Bak et al, "Avocodo: Generative adversarial network for artifact-free vocoder," AAAI 2023

Anti-Aliasing Discriminator Design

- **Collaborative multi-band discriminator** (CoMBD)



- **Sub-band discriminator** (SBD)

Ref: Bak et al, "Avocodo: Generative adversarial network for artifact-free vocoder," AAAI 2023

More On Aliasing: StyleGAN3

https://github.com/NVlabs/alias-free-gan

- "We describe a comprehensive overhaul of all signal processing aspects of the StyleGAN2 generator"
- "Current upsampling filters are simply not aggressive enough in suppressing aliasing, and that extremely high-quality filters with over 100dB attenuation are required."
- "We present a principled solution to aliasing caused by pointwise nonlinearities by considering their effect in the continuous domain and appropriately low-pass filtering the results"

	Configuration	FID ↓	EQ-T ↑	EQ-R ↑
A	StyleGAN2	5.14	–	–
B	+ Fourier features	4.79	16.23	10.81
C	+ No noise inputs	4.54	15.81	10.84
D	+ Simplified generator	5.21	19.47	10.41
E	+ Boundaries & upsampling	6.02	24.62	10.97
F	+ Filtered nonlinearities	6.35	30.60	10.81
G	+ Non-critical sampling	4.78	43.90	10.84
H	+ Transformed Fourier features	4.64	45.20	10.61
T	+ Flexible layers (StyleGAN3-T)	4.62	63.01	13.12
R	+ Rotation equiv. (StyleGAN3-R)	**4.50**	**66.65**	**40.48**

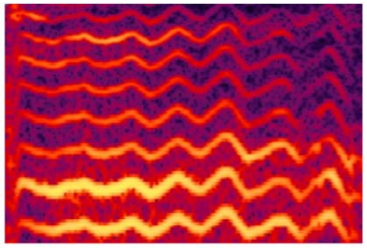
Ref: Karras et al, "Alias-free generative adversarial networks," NeurIPS 2021

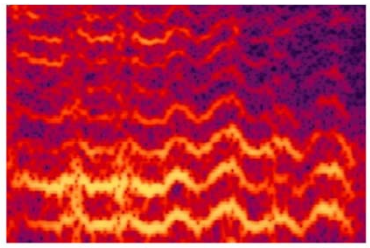
Outline

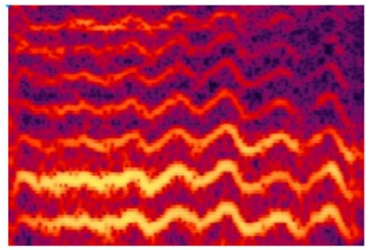
- General idea
- DL components: upsampling layers and GAN
- DL-based Mel-vocoders
- **More recent work**
  - More on downsampling, upsampling layers
  - **More Mel-Vocoders**
  - New trend: Audio codec models

BigVGAN

- A **universal** vocoder that generalizes well for *zero-shot* (out-of-distribution) scenarios without fine-tuning
  - "We introduce **periodic activation function** and **anti-aliased representation** into the GAN generator, which brings the desired inductive bias for audio synthesis and significantly improves audio quality"
  - "We train our GAN vocoder at the largest scale up to **112M** parameters, which is unprecedented in the literature"
  - "BigVGAN trained only on **clean speech** (LibriTTS), achieves the SOTA performance for various zero-shot (out-of-distribution) conditions, including unseen speakers, languages, recording environments, **singing voices**, **music**, and **instrumental audio**"

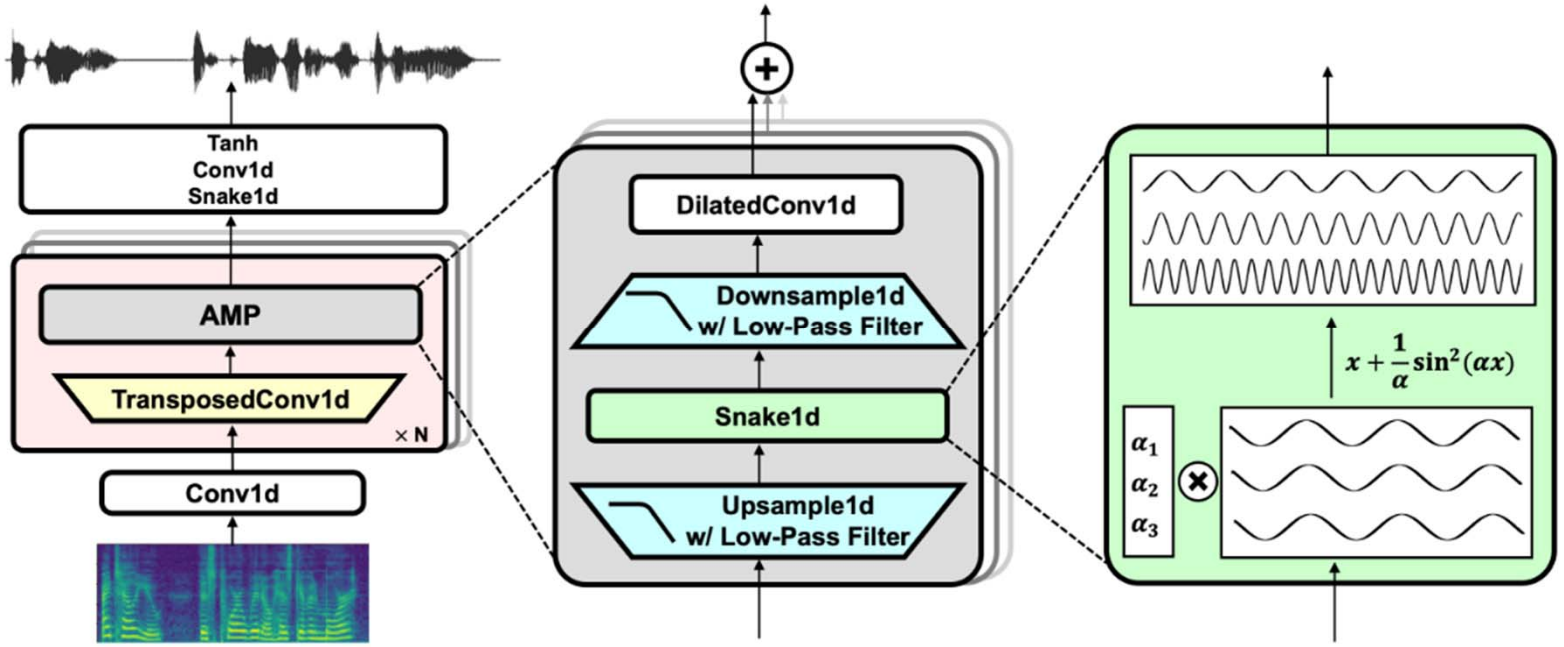

Ground Truth


HiFi-GAN


BigVGAN

Ref: Lee et al, "A universal neural vocoder with large-scale training," ICLR 2023

BigVGAN



- Anti-aliased multi-periodicity composition (AMP)
  - Uses **Snake** function for providing **periodic inductive bias**, and **low-pass filter** for **anti-aliasing** purpose.

Ref: Lee et al, "A universal neural vocoder with large-scale training," ICLR 2023

Diffusion-based Mel-Vocoders

- **DiffWave**, WaveGrad, PriorGrad

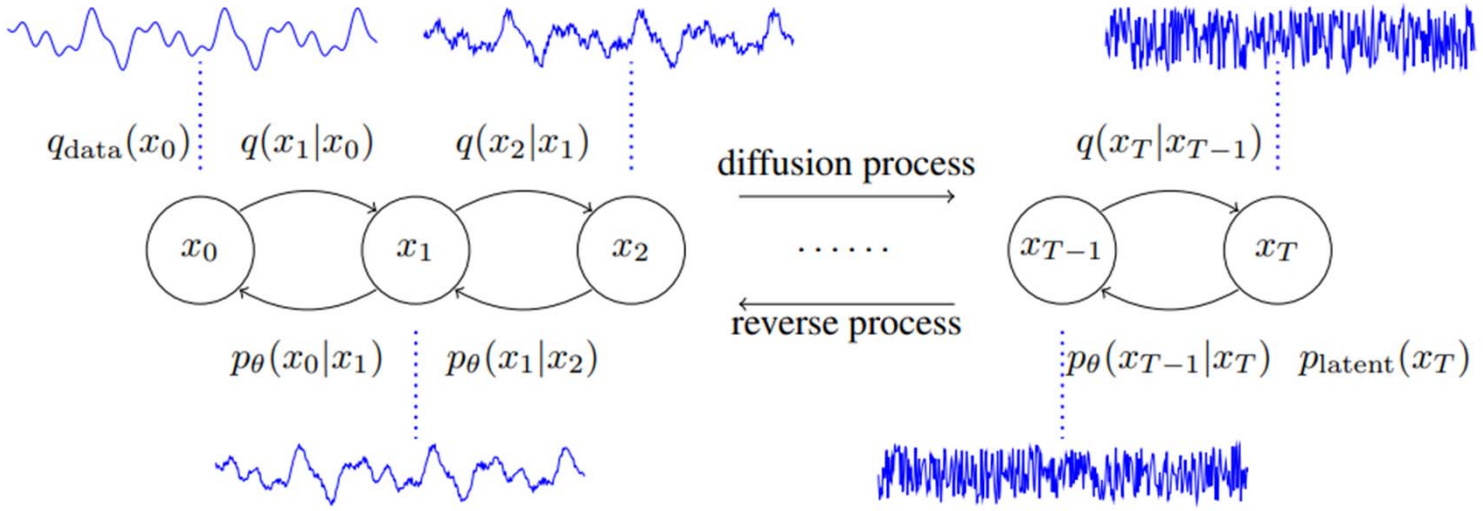
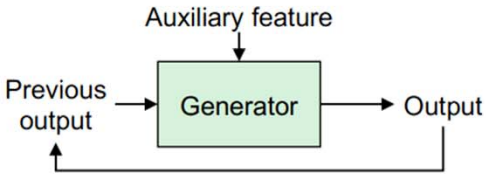


Figure 1: The diffusion and reverse process in diffusion probabilistic models. The reverse process gradually converts the white noise signal into speech waveform through a Markov chain $p_\theta(x_{t-1}|x_t)$.

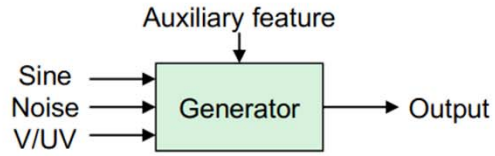
Ref: Kong et al, "DiffWave: A versatile diffusion model for audio synthesis," ICLR 2021

Source-Filter based Mel-Vocoders

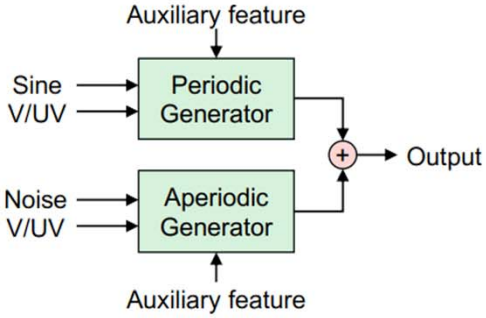
- **NSF** (ICASSP'19)
- **PeriodNet** (ICASSP'21)
- **DSPGAN** (ICASSP'23)



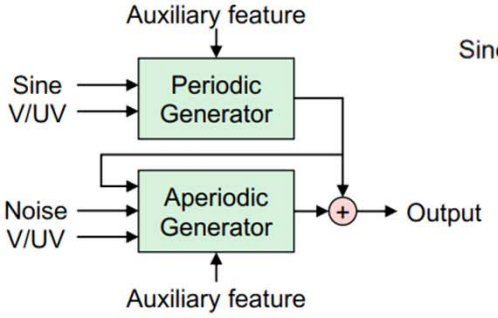
(a) AR model



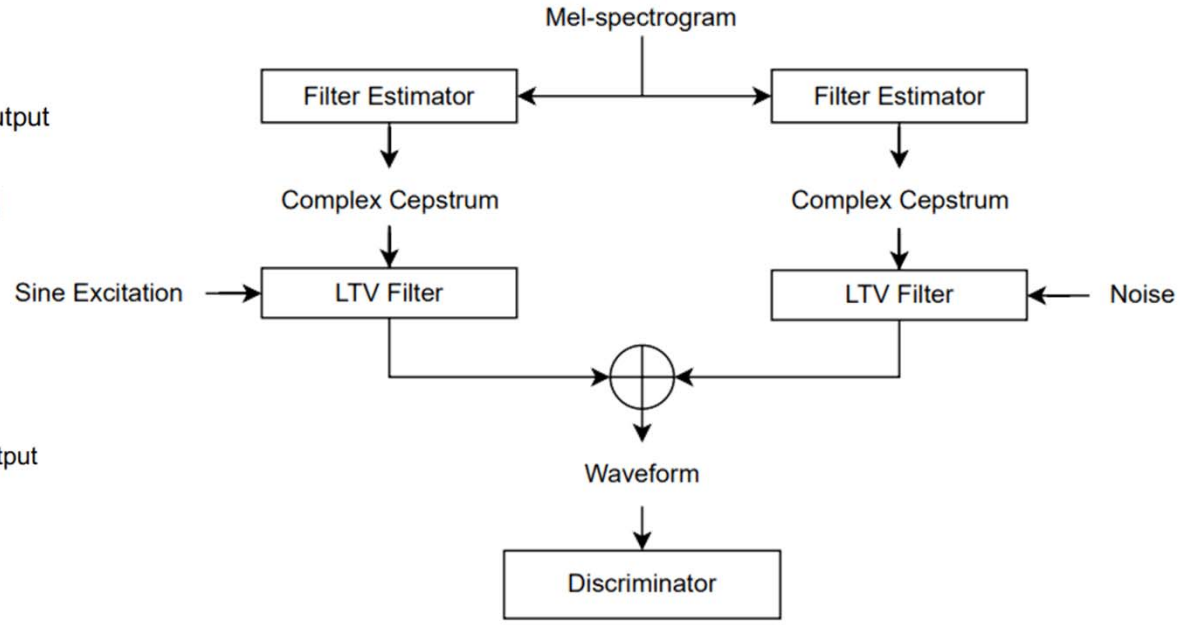
(b) Non-AR baseline model



(c) Non-AR parallel model

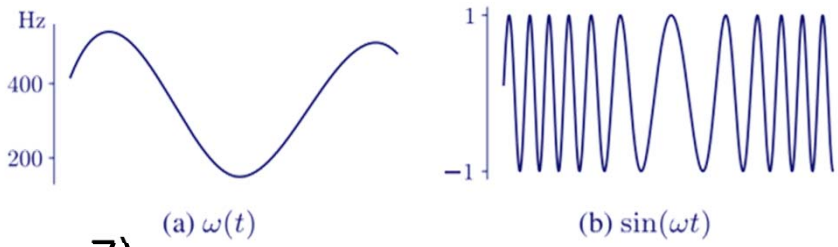


(d) Non-AR series model



Source-Filter based Mel-Vocoders

Figure from: https://arxiv.org/pdf/2306.00814.pdf



- **F0 to sine**
  - F0 estimation by YIN, WORLD, or CREPE (see Lecture 7)
  - Convert F0 (instantaneous frequency) to **sine**

- **NSF** (ICASSP'19)

The input F0 sequence is $\boldsymbol{f}_{1:T}$, where $f_t \in \mathbb{R}_{\geq 0}$ is the F0 value of the t-th time step, and $f_t > 0$ and $f_t = 0$ denote being voiced and unvoiced, respectively. A sine waveform $\boldsymbol{e}_{1:T}^{<0>}$ with the fundamental frequency can be generated as

$$e_t^{<0>} = \begin{cases} \alpha \sin(\sum_{k=1}^{t} 2\pi \frac{f_k}{N_s} + \phi) + n_t, & \text{if } f_t > 0 \\ \frac{\alpha}{3\sigma} n_t, & \text{if } f_t = 0 \end{cases}, \quad (13)$$

- Code examples [ref 1, ref 2]

ddsp-singing-vocoders / ddsp / modules.py

```
# harmonic synth
n_harmonic = amplitudes.shape[-1]
phase  = torch.cumsum(2 * np.pi * f0 / self.fs, axis=1) + initial_phase
phases = phase * torch.arange(1, n_harmonic + 1).to(phase)
```

ParallelWaveGAN / parallel_wavegan / layers / sine.py

```
 7    class SineGen(torch.nn.Module):
48        def _f02sine(self, f0_values):
56            rad_values = (f0_values / self.sampling_rate) % 1
73            tmp_over_one = torch.cumsum(rad_values, 1) % 1
```

Ref 1: https://github.com/YatingMusic/ddsp-singing-vocoders/blob/f72157cdf9738bb0a14179a0cab13a73f56f5238/ddsp/modules.py#L31
Ref 2: https://github.com/kan-bayashi/ParallelWaveGAN/blob/c68b4590ab20eaf55e0b96b82325a90177fffd5c/parallel_wavegan/layers/sine.py#L8

Source-Filter based Mel-Vocoders

https://nii-yamagishilab.github.io/samples-nsf/

https://colab.research.google.com/drive/1ekU2YlG-05FaMvWGPEpYIRFvqxSaMsqE?usp=sharing#scrollTo=-X8c2_ufjpNH

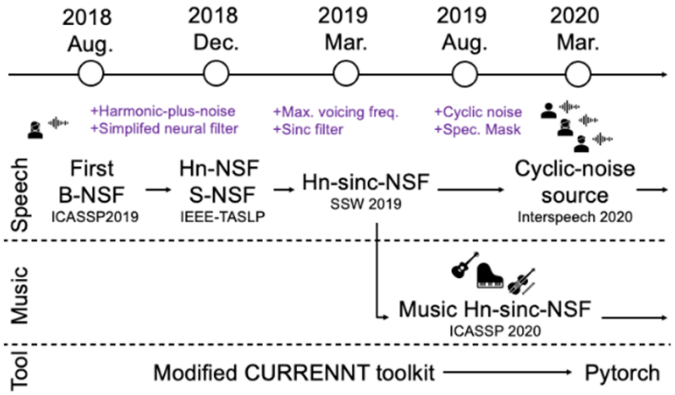
- **NSF-HiFiGAN**

Sec.3 Combine HiFiGAN Discriminator with NSF (Generator)

It is possible to replace the HiFiGAN generator with other types of neural waveform model.

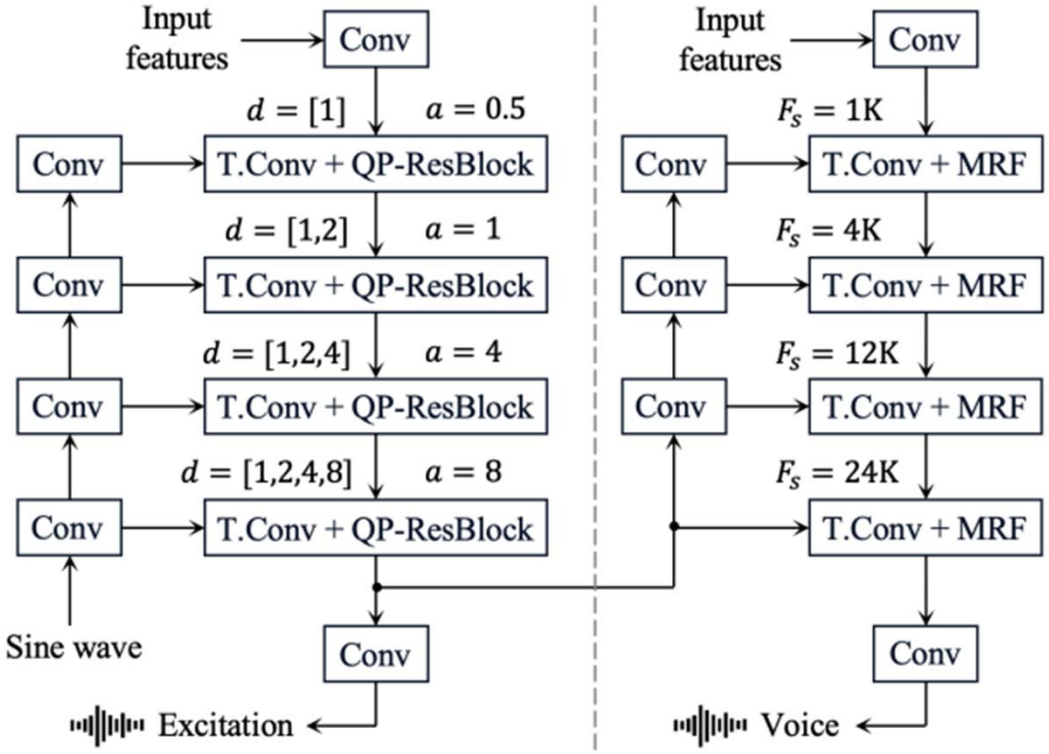
NSF is a good choice since it is trained using STFT loss only.

Here, we load the Hn-NSF trained with HiFiGAN discriminator and produce some audio samples.



Source-Filter based Mel-Vocoders

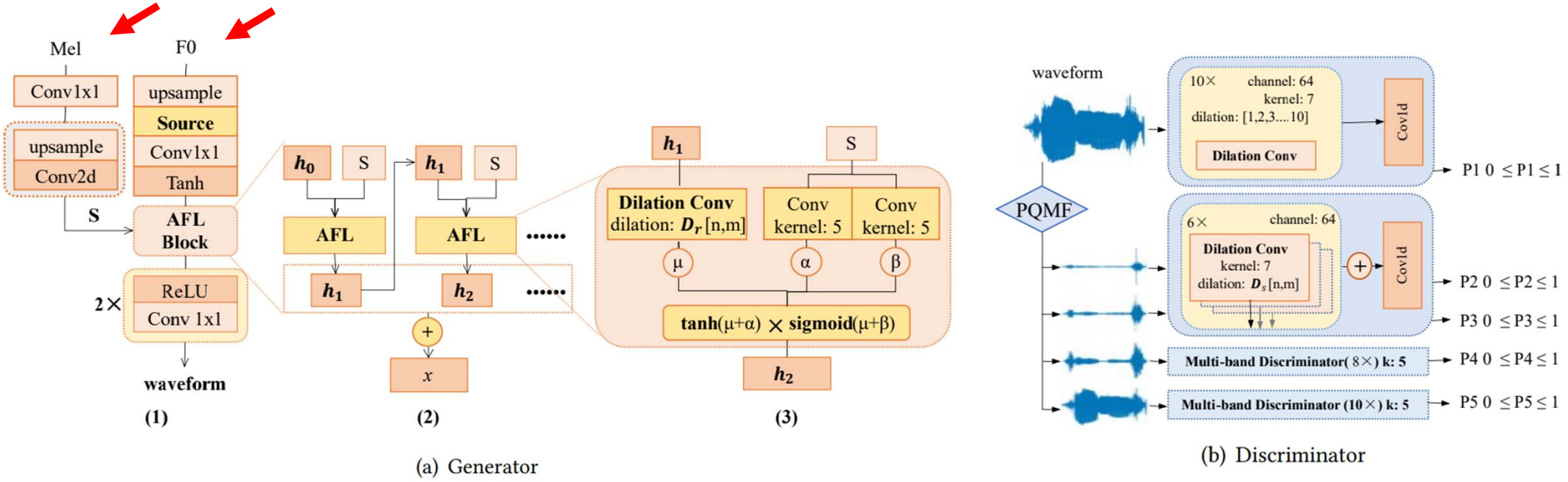
- **SiFiGAN** (ICASSP'23)**:** HiFi-GAN + F0 controllability
  - The given F0 can be different from the F0 in Mel-spectrogram
  - Examples:
    - 0.5 * F0
    - 2 * F0



Ref: Yoneyama et al, "Source-Filter HiFi-GAN: Fast and pitch controllable high-fidelity neural vocoder," ICASSP 2023

GAN-based Mel-Vocoders with F0 Input

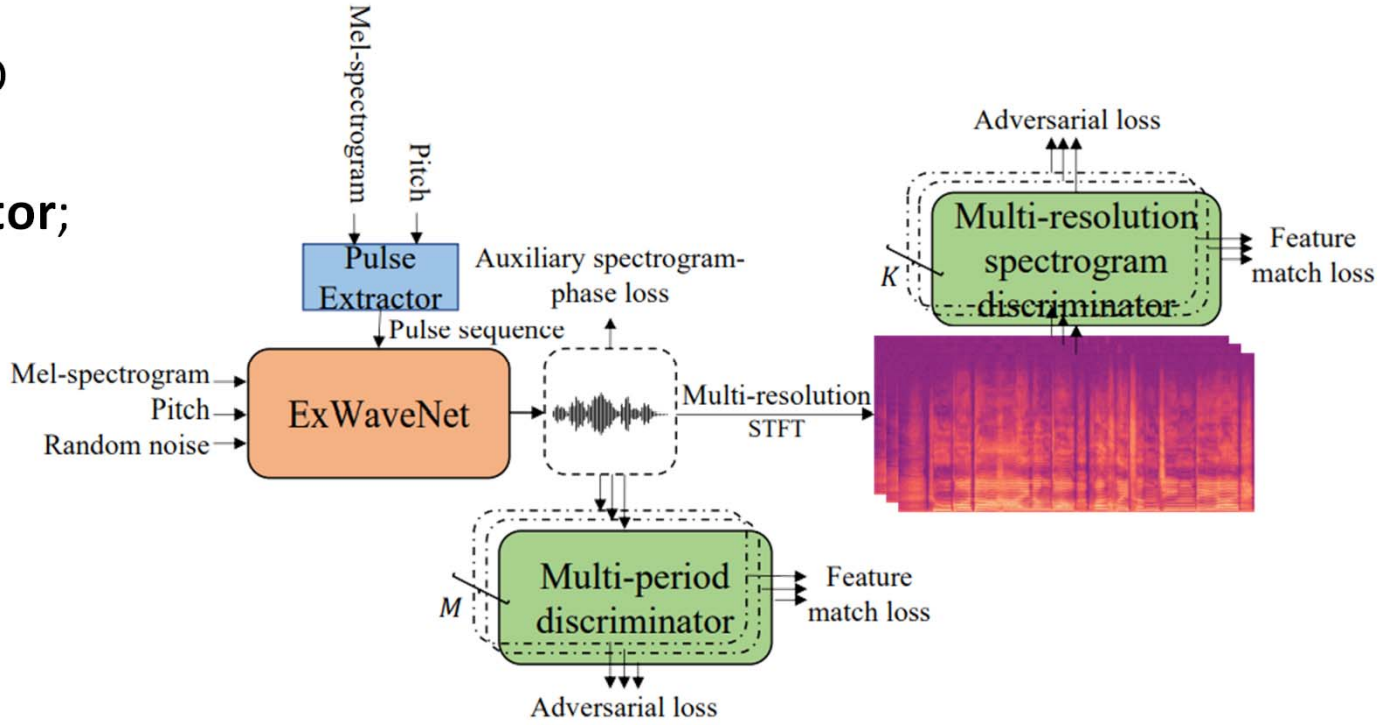
- **SingGAN** (MM'22)



Ref: Huang et al, "SingGAN: Generative adversarial network for high-fidelity singing voice generation," MM 2022

GAN-based Mel-Vocoders with F0 Input

- **HiFi-WaveGAN** (arXiv'22)
  - Use MPD (multi-period discriminator) and MRSD (**multi-resolution spectrogram discriminator**; proposed in UnivNet)



Ref: Wang et al, "HiFi-WaveGAN: Generative adversarial network with auxiliary spectrogram-phase loss for high-fidelity singing voice generation," arXiv 2022

GAN-free, DDSP-based Mel-Vocoders

https://github.com/YatingMusic/ddsp-singing-vocoders

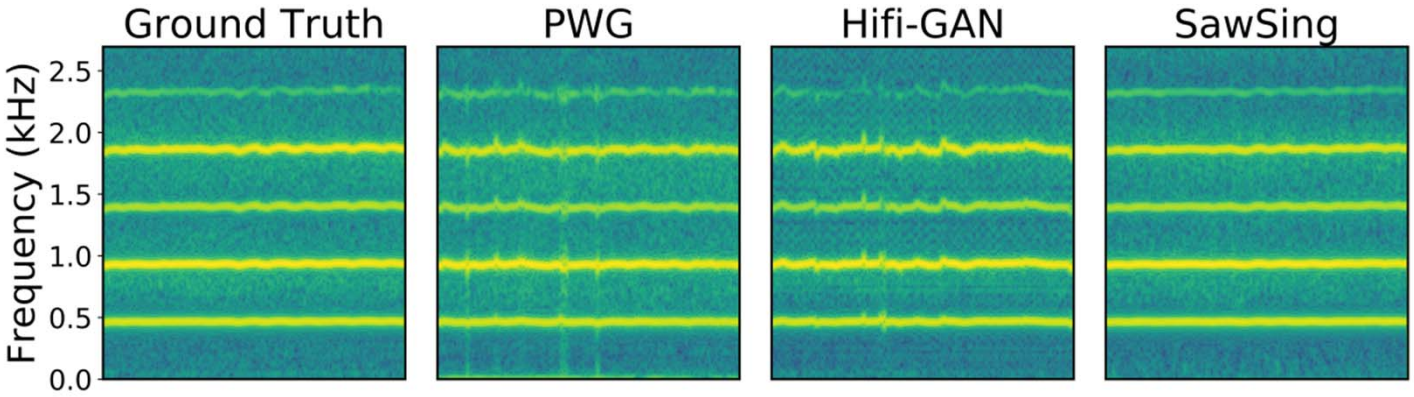
- Use the **source filter** model as an inductive bias for training vocoders
- **No GAN**

All the vocoders we presented are '**harmonic plus noise**' models[4]. According to their ways to modeling harmonic parts of singing voices, we have the following models:

Model Name	Synthesizer	Note
SawSinSub	Filtered Sawtooth Synthesizer (Appproximated by Harmonic Synthesizer)	proposed SawSing[1]
Sins	Harmonic Synthesizer	[2]
DWS	Wavetable Synthesizer	[3]
Full	Filtered Harmonic Synthesizer	modified from [2]
SawSub	Filtered Sawtooth Synthesizer	modified from [1]

GAN-free, DDSP-based Mel-Vocoders

- The use of the sine activation avoids the **glitches** in black-box models such as PWG and HiFiGAN



- It’s also possible to avoid the glitches for HiFiGAN by using F0 as an additional condition

Ref: Wu et al, “DDSP-based singing vocoders: A new subtractive-based synthesizer and a comprehensive evaluation,” ISMIR 2022

GAN-free, DDSP-based Mel-Vocoders

- The source-filter assumption provides an inductive bias that allows the vocoder to be trained on a **small** amount of data

Model	Para-meters	RTF
FastDiff [21]	15.3M	0.017
HiFi-GAN [15]	13.9M	0.004
PWG [14]	1.5M	0.007
NSF [12]	1.2M	0.006
DDSP-Add [44]	0.5M	0.003
DWTS [27]	0.5M	0.019
SawSing	0.5M	0.003

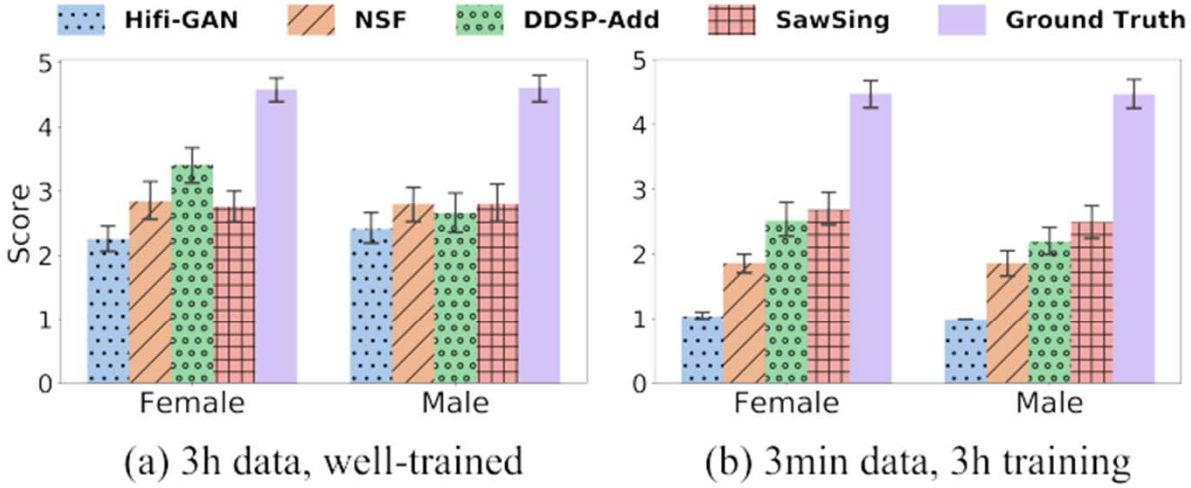


Figure 3: MOS with 95% confidence intervals for subjective evaluation of vocoders trained in the two scenarios.

Ref: Wu et al, "DDSP-based singing vocoders: A new subtractive-based synthesizer and a comprehensive evaluation," ISMIR 2022

Outline

- General idea
- DL components: upsampling layers and GAN
- DL-based Mel-vocoders
- **Advanced topics**
  - More on downsampling, upsampling layers
  - More Mel-Vocoders
  - **New trend: Audio codec models** → see lecture on text-to-music