2024 edition

Deep Learning for Music Analysis and Generation

Music Synthesis & Timbre Transfer

generating notes, loops, or short clips
({x, label, audio} → audio)

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

Objectives

- Previously
  - **Source separation**: “audio (mixture) → audio (stem)” [*perfect alignment*]
  - **Mel-vocoder**: “mel → audio (stem)” [*output length is a factor of the input length*]
    - Conditional audio generation tasks with **strong, temporally-aligned** conditions
- Now
  1. **Music synthesis:** “MIDI pitch → audio”
     - Conditional audio generation with only **class-level condition**
  2. **Unconditional generation** of loops**:** “x → audio”
     - Unconditional audio generation (**no conditions**)
     - *Weaker* condition: **No “ground-truth” target**
     - Cannot use reconstruction loss
  3. **Timbre transfer** of short clips**:** “audio → audio”

Outline

- **DSP-based music synthesizers**
- DL-based generation of single notes
- DL-based generation of loops
- DL-based timbre transfer of short audio clips

Synthesizers

https://en.wikipedia.org/wiki/Synthesizer

- An electronic musical instrument that generates audio signals
  - **Analog**: circuits, voltage-controlled oscillator
  - **Digital**: through DSP
    - Yamaha DX7: based on frequency modulation (FM) synthesis technology

Synthesizers via Circuits

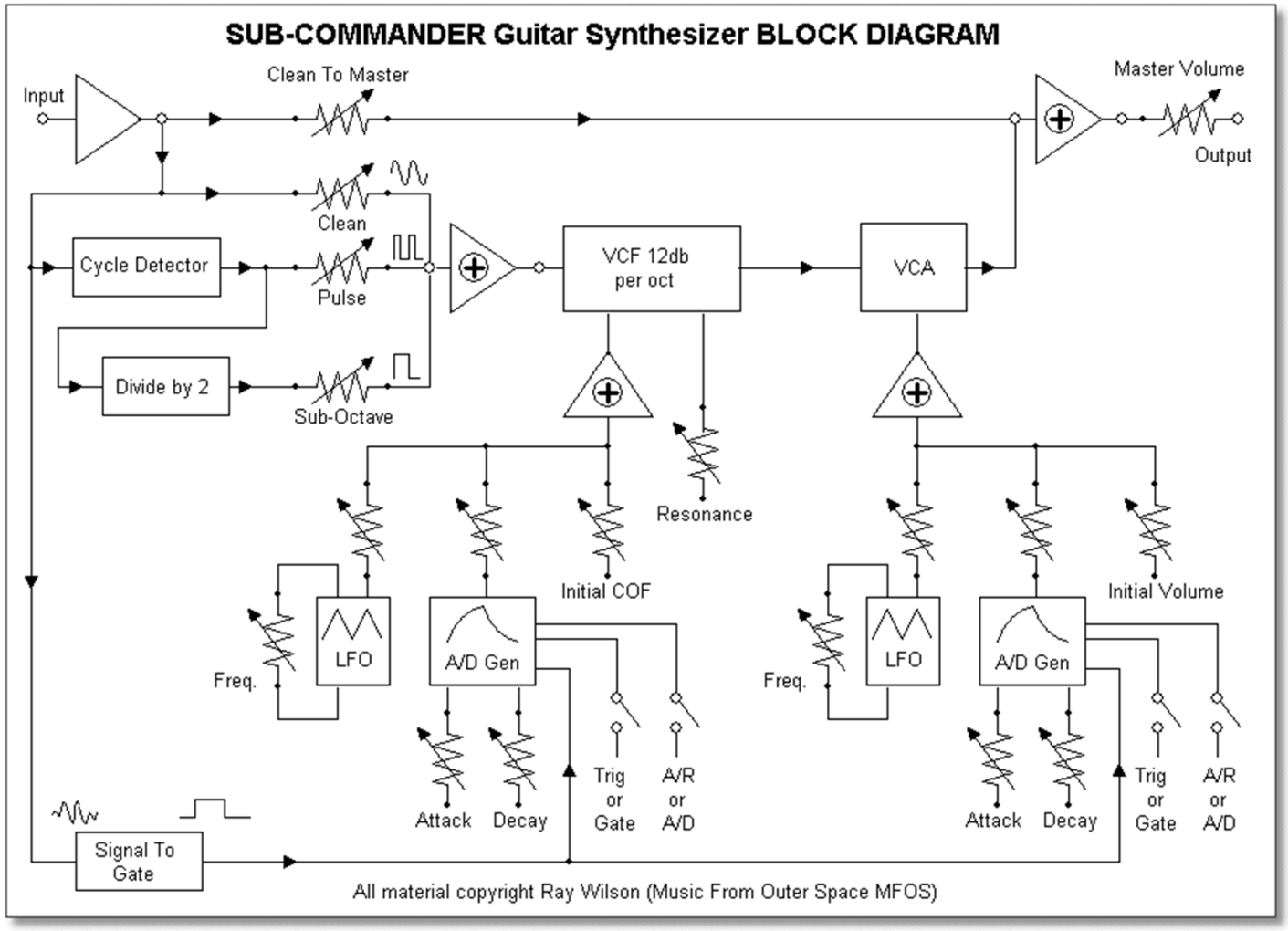


Figure from:
http://musicfromouterspace.com/analogsynth_new/GUITARSYNTHAUG2007/GUITARSYNTHAUG2007.html

Synthesizers for MIDI-to-Audio Rendering

https://www.fluidsynth.org/

https://github.com/bzamecnik/midi2audio

- Useful when working on symbolic music generation

FluidSynth

A SoundFont Synthesizer

FluidSynth is a real-time software synthesizer based on the SoundFont 2 specifications and has reached widespread distribution. FluidSynth itself does not have a graphical user interface, but due to its powerful API several applications utilize it and it has even found its way onto embedded systems and is used in some mobile apps.

```
from midi2audio import FluidSynth
```

Play MIDI:

```
FluidSynth().play_midi('input.mid')
```

Synthesize MIDI to audio:

```
# using the default sound font in 44100 Hz sample rate
fs = FluidSynth()
fs.midi_to_audio('input.mid', 'output.wav')
```

Synthesizers for Sound Design

https://www.youtube.com/watch?v=NJLIS2MkFe4

- Go beyond simulating the sounds of an acoustic instrument
- Widely used in EDM

Figure from: https://professionalcomposers.com/synth-pad-quick-guide/

Sound Design and Synth Fundamentals

DSP-based Synthesizers

https://en.wikipedia.org/wiki/Synthesizer

- Different types of synthesizers
  - **Subtractive synthesis:** complex waveforms are generated by oscillators and then shaped with filters to remove or boost specific frequencies
  - **Additive synthesis:** a large number of waveforms, usually sine waves, are combined into a composite sound
  - **FM synthesis:** a carrier wave is modulated with the frequency of a modulator wave
  - **Wavetable synthesis:** synthesizers modulate smoothly between digital representations of different waveforms, changing the shape and timbre
  - **Granular synthesis**
  - **Sample-based synthesizer**
  - **Physical modelling synthesis**

DSP-based Synthesizers

https://www.coursera.org/learn/audio-signal-processing

https://github.com/MTG/sms-tools

https://cs.gmu.edu/~sean/book/synthesis/

Week 1: Introduction; basic mathematics
Week 2: Discrete Fourier transform
Week 3: Fourier transform properties
Week 4: Short-time Fourier transform
Week 5: Sinusoidal model
Week 6: Harmonic model
Week 7: Sinusoidal plus residual modeling
Week 8: Sound transformations
Week 9: Sound/music description
Week 10: Concluding topics; beyond audio signal processing

Topics in the Book

1. **Introduction.** Synthesizer basics, usage environments, basic history.
2. **Representations of sound.** Units of measure, issues in digitization of signals.
3. **Additive synthesis.** Additive implementation, monophony and polyphony.
4. **Modulation.** LFOs, envelopes, sequencers and drum machines, arpeggiation, modulation matrices, MIDI modulation.
5. **Subtractive synthesis.** Oscillators, antialiasing and Nyquist, wave shaping, wave folding, phase distortion, combination and amplification.
6. **Digital filters.** Transfer functions, poles and zeros, magnitude and phase response, Laplace domain, Z domain and the Bilinear transform, second-order filters, filter composition, first-order filters, fourth-order ladder filters, formants.
7. **Frequency Modulation synthesis.** Phase modulation, sidebands, Bessel functions and reflection, operators and algorithms.
8. **Sampling.** Pulse code modulation, wavetable synthesis, granular synthesis, resampling techniques, real-time interpolation.
9. **Effects.** Delays, flangers, chorus, reverb, phasers, physical modeling synthesis.
10. **Controllers.** MIDI protocols.
11. **The Fourier Transform.** DFT and FFT, windowing, STFT.

Sample-based Synthesis

- **Vocaloid** by Xavier Serra
  https://en.wikipedia.org/wiki/Xavier_Serra

Daisy was a collaboration project with Yamaha. The aim of the project was to develop a singing voice synthesizer in which the user would input the lyrics and the notes of a vocal melody and obtain a synthetic performance of a virtual singer. To synthesize such performance the system concatenates a chain of elemental synthesis units. These units are obtained by transposing and time-scaling samples from singers databases. These databases are created out of recording, analyzing, labeling and storing singers performing in as many different musical and phonetic contexts as possible.

Based on Daisy's research, Yamaha released a product named Vocaloid. Vocaloid was presented at the 114th Audio Engineering Society (AES) Convention in Amsterdam in March 2003 and had a big media impact leaded by The New York Times article "Could I Get That Song in Elvis, Please?". The Vocaloid synthesizer was Nominee for the European IST (Information Society Technologies) Prize 2005 and is the choice of leading artists including Mike Oldfield.

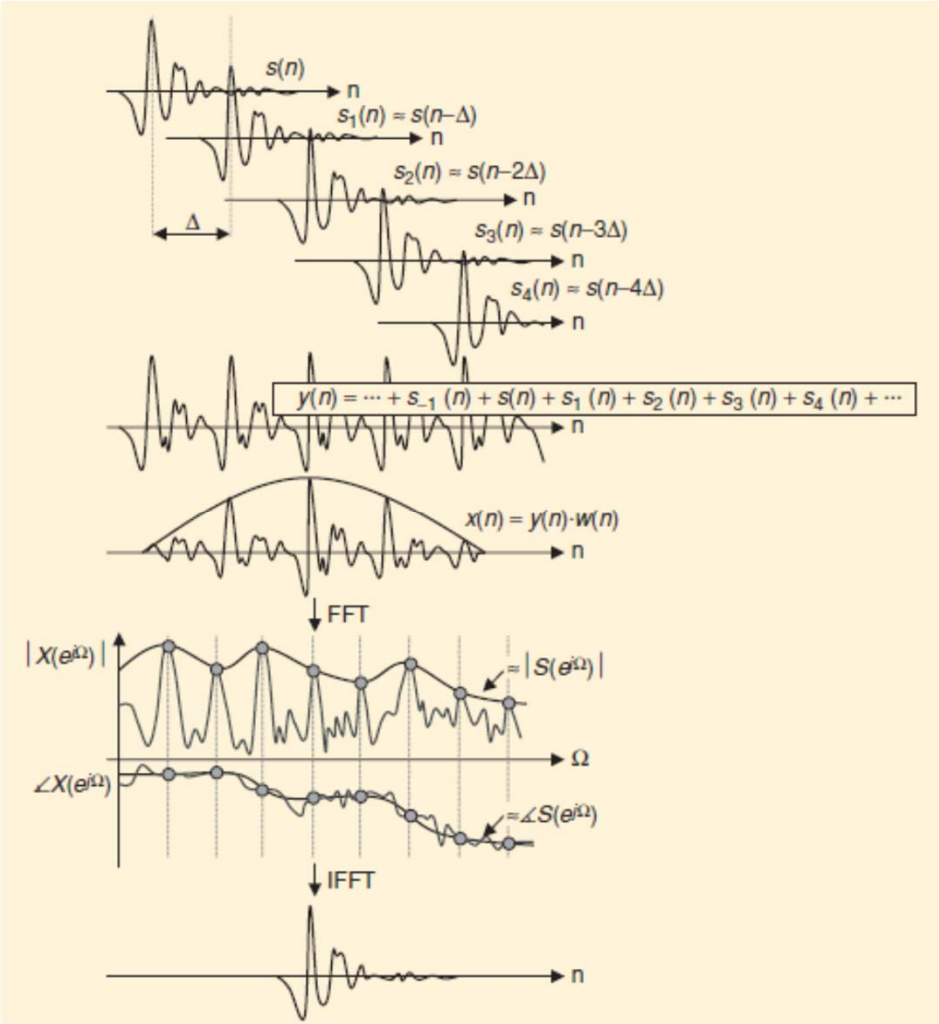
The research carried out for Daisy is mainly described in Bonada and Loscos (2003), Bonada et al. (2003), and Bonada et al (2001).

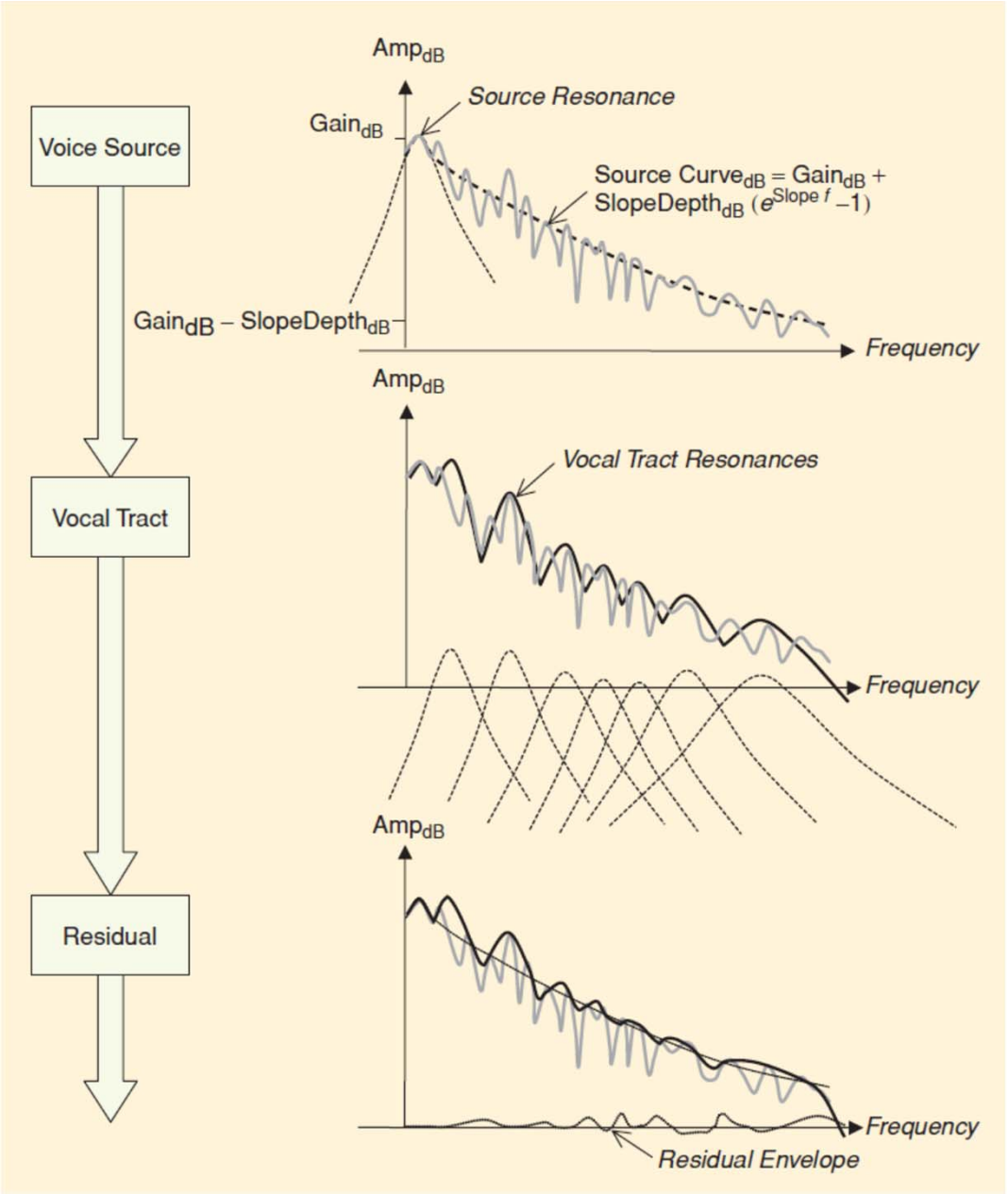
Source:
Alex Loscos, *Spectral Processing of the Singing Voice*, PhD Thesis, UPF, Spain (supervised by Xavier Serra)

(Bonada and Loscos, 2003) “Sample-based singing voice synthesizer by spectral concatenation”

```The singing synthesis system we present generates a performance of an artificial singer out of the musical score and the phonetic transcription of a song using a frame-based frequency domain technique. This performance mimics the real singing of a singer that has been previously recorded, analyzed and stored in a database, in which we store his voice characteristics (phonetics) and his low-level expressivity (attacks, releases, note transitions and vibratos). **To synthesize such performance the systems concatenates a set of elemental synthesis units (phonetic articulations and stationeries). These units are obtained by transposing and time-scaling the database samples**. The concatenation of these transformed samples is performed by spreading out the spectral shape and phase discontinuities of the boundaries along a set of transition frames that surround the joint frames. […]```

Source: http://mtg.upf.edu/node/322
```





https://ieeexplore.ieee.org/document/4117930

Sample-based Synthesis: Ample Sound

http://www.amplesound.net/en/index.asp

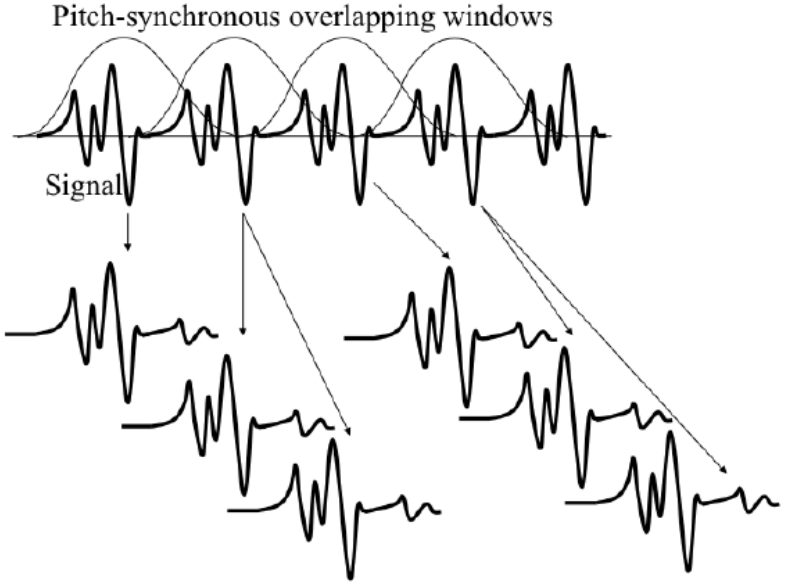
- Challenges:
  - Great manual effort in recording
    - Every note, velocity, playing technique, etc
  - Note-note transition

Overlap-and-Add for Audio Time Stretching and Pitch Scaling

https://en.wikipedia.org/wiki/Audio_time_stretching_and_pitch_scaling

https://en.wikipedia.org/wiki/PSOLA

- **PSOLA**: Pitch synchronous overlap-and-add
  - Divide the waveform into small overlapping segments
  - To **change the pitch**, the segments are *moved further apart* (to decrease the pitch) or *closer* together (to increase the pitch)
  - To **change the duration**, the segments are then *repeated* multiple times (to increase the duration) or some are *eliminated* (to decrease the duration)



Ref: Figure from https://www.researchgate.net/figure/Pitch-Synchronous-Overlap-Add-PSOLA_fig2_264006670

Audio Signal Processing by PyTSMod

(Audio Time Stretching and Pitch Scaling)

https://seyong92.github.io/PyTSMod-ISMIR2020LBD/

Available TSM algorithms

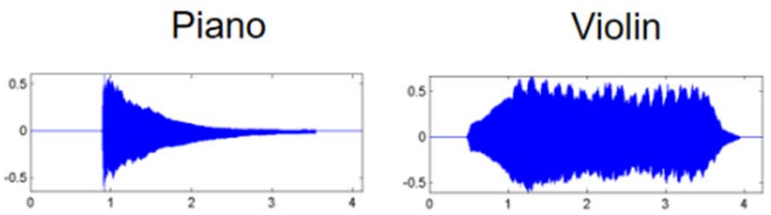
- Overlap-Add (OLA)
  - OLA is the most simple TSM algorithm that changes the length of the signal through modifying the hop size between analysis frame and synthesis frame.
- Pitch Synchronous Overlap-Add (TD-PSOLA)
  - TD-PSOLA is the algorithm that analyzes the orignal waveforms to create pitich-synchronous analysis windows and synthesize the output signal both for modifying time-scale and pitch-scale.
- Waveform Similarity Overlap-Add (WSOLA)
  - WSOLA maximizes the waveform similarity by allowing timing tolerance to analysis frame to find the most similar position through cross correlation.
- Phase Vocoder (PV)
  - Phase vocoder estimates instantaneous frequency, and it is used to update phases of input signal's frequency components in short-time Fourier transform. Although TSM results with phase vocoder has high phase continuity, it causes an transient smearing for percussive audio sources and a coloring artifact called phasiness.
- TSM based on harmonic-percussive source separation (HPTSM)
  - A novel TSM algorithm that applying phase vocoder to only harmonic sources and applying OLA to only percussive sources.

Outline

- DSP-based music synthesizers
- **DL-based generation of single notes**
  - WaveNet Autoencoder
  - GANSynth
  - InstrumentGen
- DL-based generation of loops
- DL-based timbre transfer of short audio clips

Neural Generation of Single Notes

- “**MIDI pitch → audio**”
- Synthesis of **timbre**
- Can be used in **MIDI-to-audio** rendering



- Can we learn such a mapping in a *data-driven* way via DL?
- Can we learn a *latent embedding* space of sounds that allow for creative applications such as interpolation or extrapolation of sounds?

Figure from: https://crazyoscarchang.github.io/2018/11/13/agent-embeddings/

Autoregressive Models for Unconditional Audio Generation

- **WaveNet** (Van Den Oord et al., 2016)
- **SampleRNN** (Mehri et al., 2016)
- **WaveRNN** (Kalchbrenner et al., 2018)
- Inference with these models is inherently slow and inefficient because audio samples must be generated sequentially

WaveNet (arXiv'16)

In this paper we introduce a new generative model operating directly on the raw audio waveform. The joint probability of a waveform $\mathbf{x} = \{x_1, \ldots, x_T\}$ is factorised as a product of conditional probabilities as follows:

$$p(\mathbf{x}) = \prod_{t=1}^{T} p(x_t \mid x_1, \ldots, x_{t-1}) \quad (1)$$

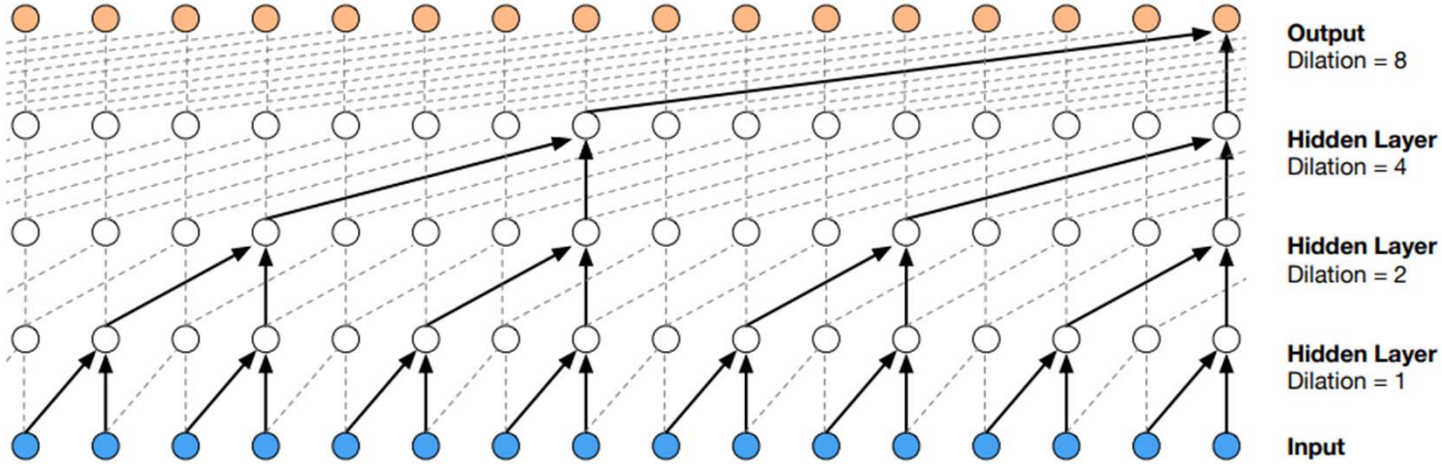
Each audio sample x_t is therefore conditioned on the samples at all previous timesteps.

- **Autoregressive; causal** convolutions
  - The prediction at timestep *t* cannot depend on any of the future timesteps
  - "At training time, the conditional predictions for all timesteps can be made in **parallel** because all timesteps of ground truth x are known"
  - "When generating with the model, the predictions are **sequential**: after each sample is predicted, it is fed back into the network to predict the next sample"

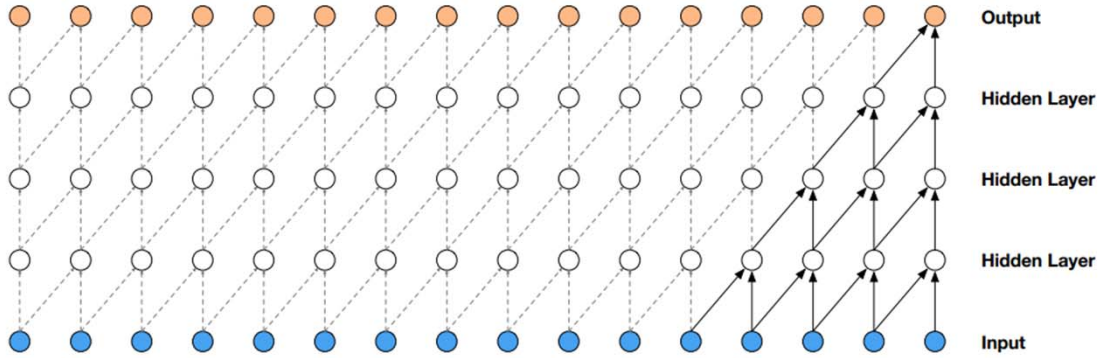
Ref: van den Oord et al, "WaveNet: A generative model for raw audio," arXiv 2016

WaveNet: Dilated Convolutions

- *Convolution with holes*
  - "A convolution where the filter is applied over an area larger than its length by skipping input values with a certain step"
  - Exponentially increasing the dilation factor results in exponential receptive field growth with depth



Usual convolution



WaveNet: Output Design

Because raw audio is typically stored as a sequence of 16-bit integer values (one per timestep), a softmax layer would need to output 65,536 probabilities per timestep to model all possible values. To make this more tractable, we first apply a μ-law companding transformation (ITU-T, 1988) to the data, and then quantize it to 256 possible values:

$$f(x_t) = \text{sign}(x_t) \frac{\ln(1 + \mu |x_t|)}{\ln(1 + \mu)},$$

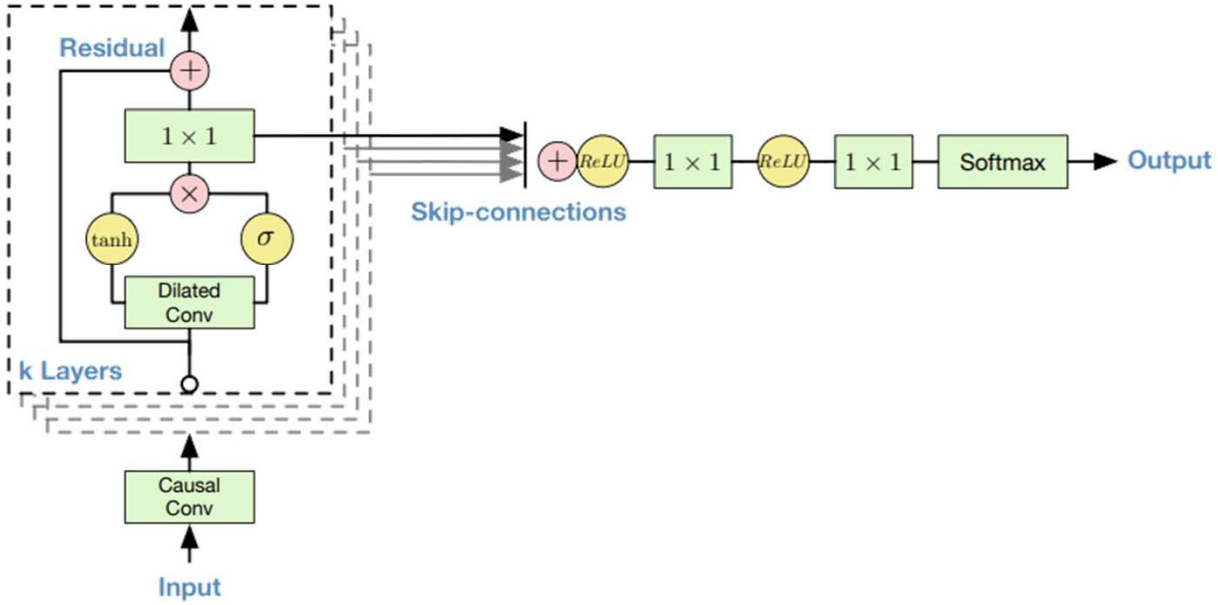
where $-1 < x_t < 1$ and $\mu = 255$. This non-linear quantization produces a significantly better reconstruction than a simple linear quantization scheme. Especially for speech, we found that the reconstructed signal after quantization sounded very similar to the original.

WaveNet: Gating

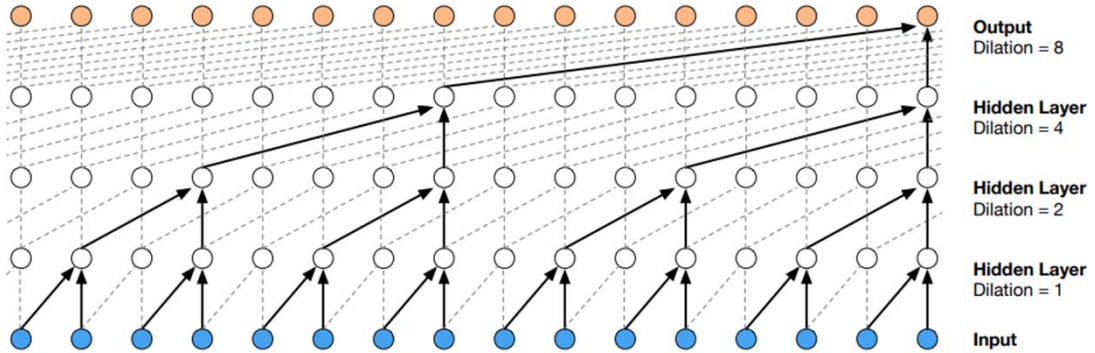
We use the same gated activation unit as used in the gated PixelCNN (van den Oord et al., 2016b):

$$\mathbf{z} = \tanh\left(W_{f,k} * \mathbf{x}\right) \odot \sigma\left(W_{g,k} * \mathbf{x}\right), \tag{2}$$

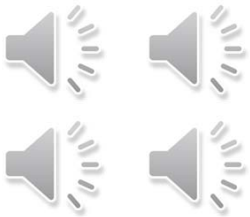
where $*$ denotes a convolution operator, $\odot$ denotes an element-wise multiplication operator, $\sigma(\cdot)$ is a sigmoid function, k is the layer index, f and g denote filter and gate, respectively, and W is a learnable convolution filter. In our initial experiments, we observed that this non-linearity worked significantly better than the rectified linear activation function (Nair & Hinton, 2010) for modeling audio signals.



WaveNet: Generated Samples



$$p(\mathbf{x}) = \prod_{t=1}^{T} p(x_t \mid x_1, \ldots, x_{t-1})$$



- Unconditional generation from this model manifests as “babbling” due to the lack of longer term structure
  - https://download.magenta.tensorflow.org/audio_examples/nsynth/UnconditionalWaveNetGeneration_WARNING_HIGH_VOLUME/0.mp3
  - https://download.magenta.tensorflow.org/audio_examples/nsynth/UnconditionalWaveNetGeneration_WARNING_HIGH_VOLUME/1.mp3
  - https://download.magenta.tensorflow.org/audio_examples/nsynth/UnconditionalWaveNetGeneration_WARNING_HIGH_VOLUME/2.mp3
  - https://download.magenta.tensorflow.org/audio_examples/nsynth/UnconditionalWaveNetGeneration_WARNING_HIGH_VOLUME/3.mp3

WaveNet AutoEncoder (ICML'17)

- Extension of WaveNet for "**MIDI pitch → audio**"
  - *WaveNet*: decoder-only; unconditional generation
  - *WaveNet Autoencoder*: encoder-decoder; conditional generation

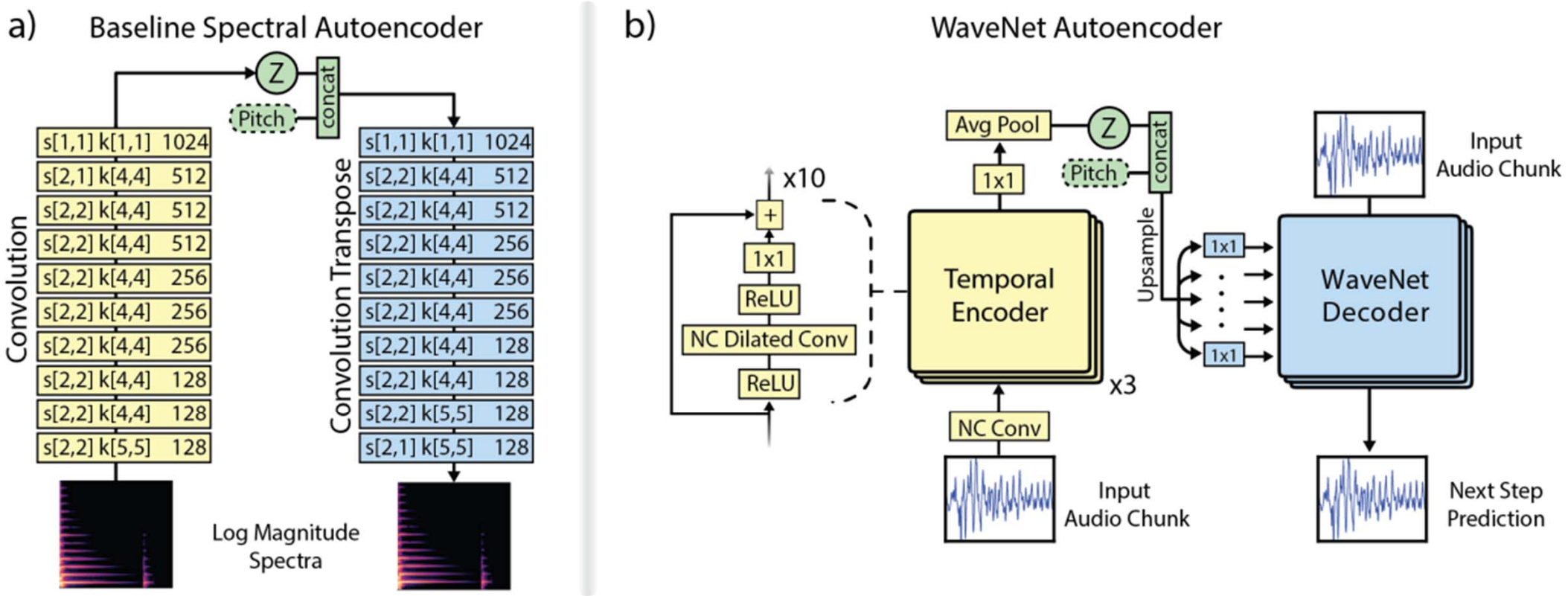
- **WaveNet Autoencoder**
  - WaveNet-like **encoder** that infers hidden embeddings $Z = f(x)$ distributed in time
  - WaveNet **decoder** that uses Z to reconstruct the original audio autoregressively

$$p(x) = \prod_{i=1}^{N} p(x_i | x_1, ..., x_{N-1}, f(x))$$

  - At inference time, the embedding can be inferred deterministically from audio or drawn from other points in the embedding space, e.g., through interpolation

Ref: Engel et al, "Neural audio synthesis of musical notes with WaveNet autoencoders," ICML 2017

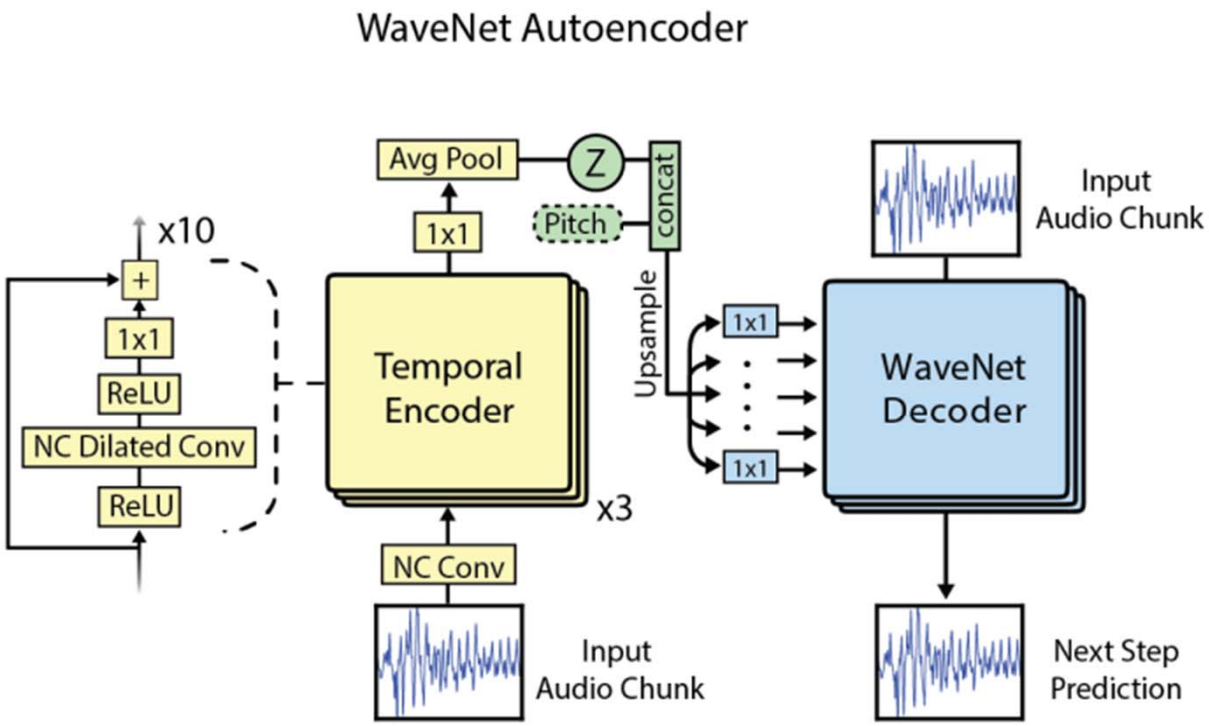
WaveNet AutoEncoder



- **Pitch**: one-hot pitch representation **(an embedding)**
- For the spectral autoencoder, Griffin-Lim is used to reconstruct phase

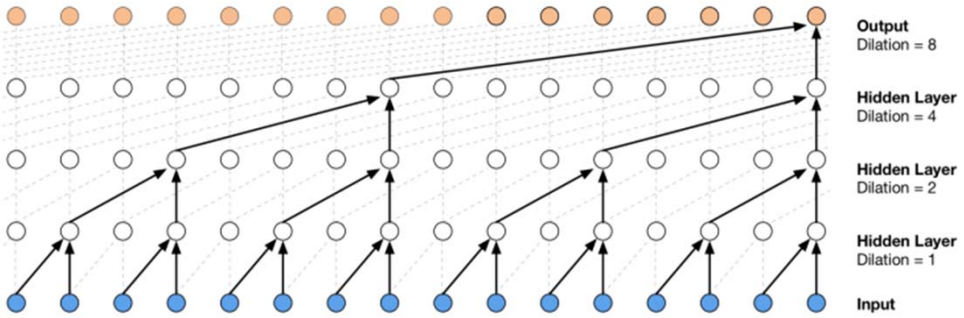
WaveNet AutoEncoder

- *WaveNet-like encoder*: **non-causal** (NC)**, non-autoregressive**
- *WaveNet decoder*: **causal, autoregressive**

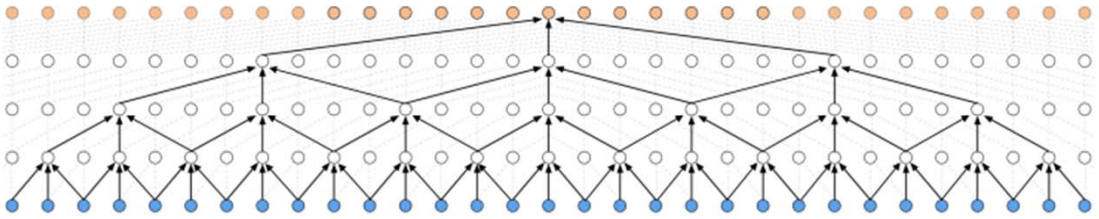


WaveNet AutoEncoder

- *WaveNet-like encoder*: **non-causal** (NC)**, non-autoregressive**
- *WaveNet decoder*: **causal, autoregressive**



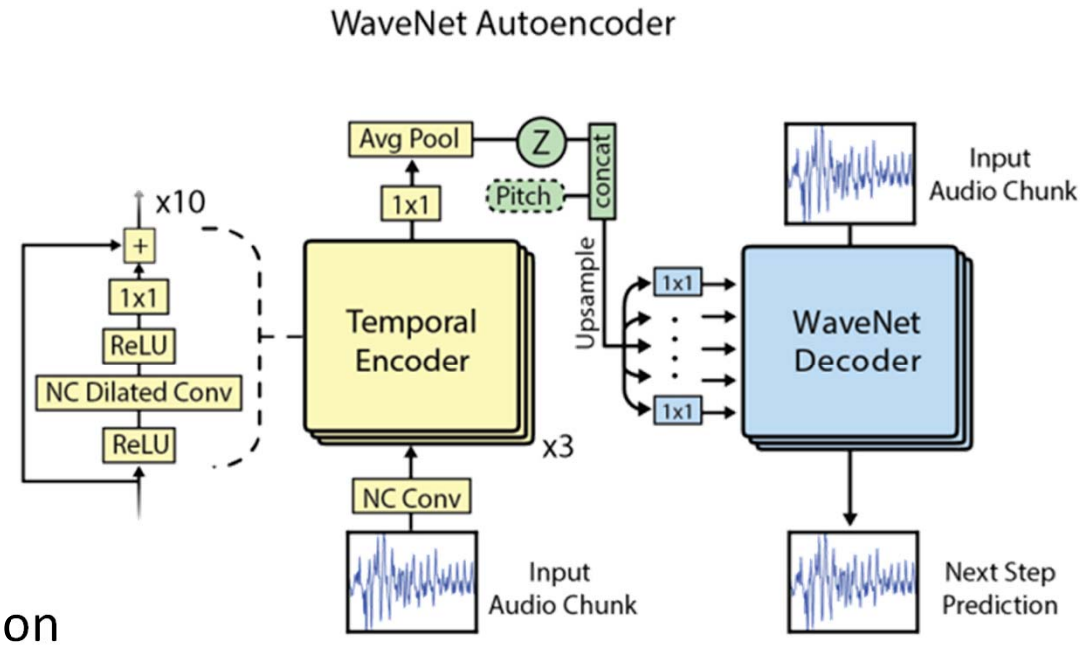
a stack of dilated causal convolutional layers



a stack of dilated non-causal convolutional layers

WaveNet AutoEncoder: Details

- Encoder
  - 30-layer nonlinear residual network
  - The result Z is a sequence of hidden codes with separate dimensions for time and channel
  - A stride of 512 (32ms) with 16 dimensions per timestep
- Decoder
  - Upsample Z to the original audio rate with nearest neighbor interpolation
  - Use that as a condition to bias every layer with a different 1x1 linear projection



Dataset: NSynth

https://magenta.tensorflow.org/nsynth

- Across a range of pitches, timbres, and volumes

	Source			
Family	ACOUST	ELECTR	SYNTH	**Total**
BASS	200	8387	60 368	68 955
BRASS	13 760	70	0	13 830
FLUTE	6572	70	2816	9458
GUITAR	13 343	16 805	5275	35 423
KEYBOARD	8508	42 709	3838	55 055
MALLET	27 722	5581	1763	35 066
ORGAN	176	36 401	0	36 577
REED	14 262	76	528	14 866
STRING	20 510	84	0	20 594
SYNTH LEAD	0	0	5501	5501
VOCAL	3925	140	6688	10 753
Total	108 978	110 224	86 777	306 043

Performance Evaluation on NSynth

- WaveNet is slow (autoregressive generation)
- And is outperformed by a later model called **GANSynth** in perceptual quality

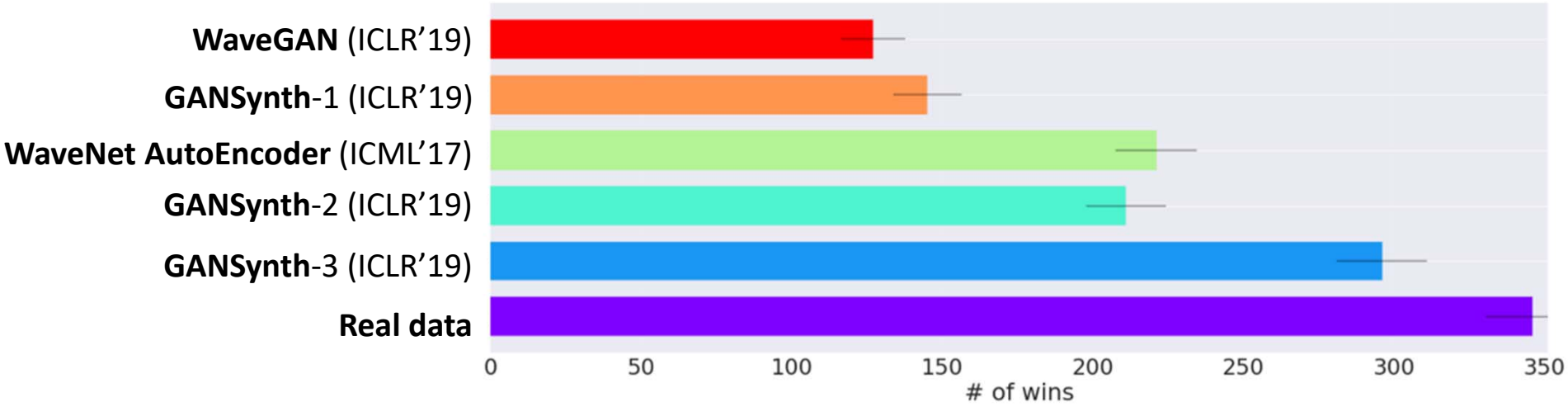


Figure 2: Number of wins on pair-wise comparison across different output representations and baselines. Ablation comparing highest performing models of each type. Higher scores represent better

Ref: Engel et al, "GANSynth: Adversarial neural audio synthesis," ICLR 2019

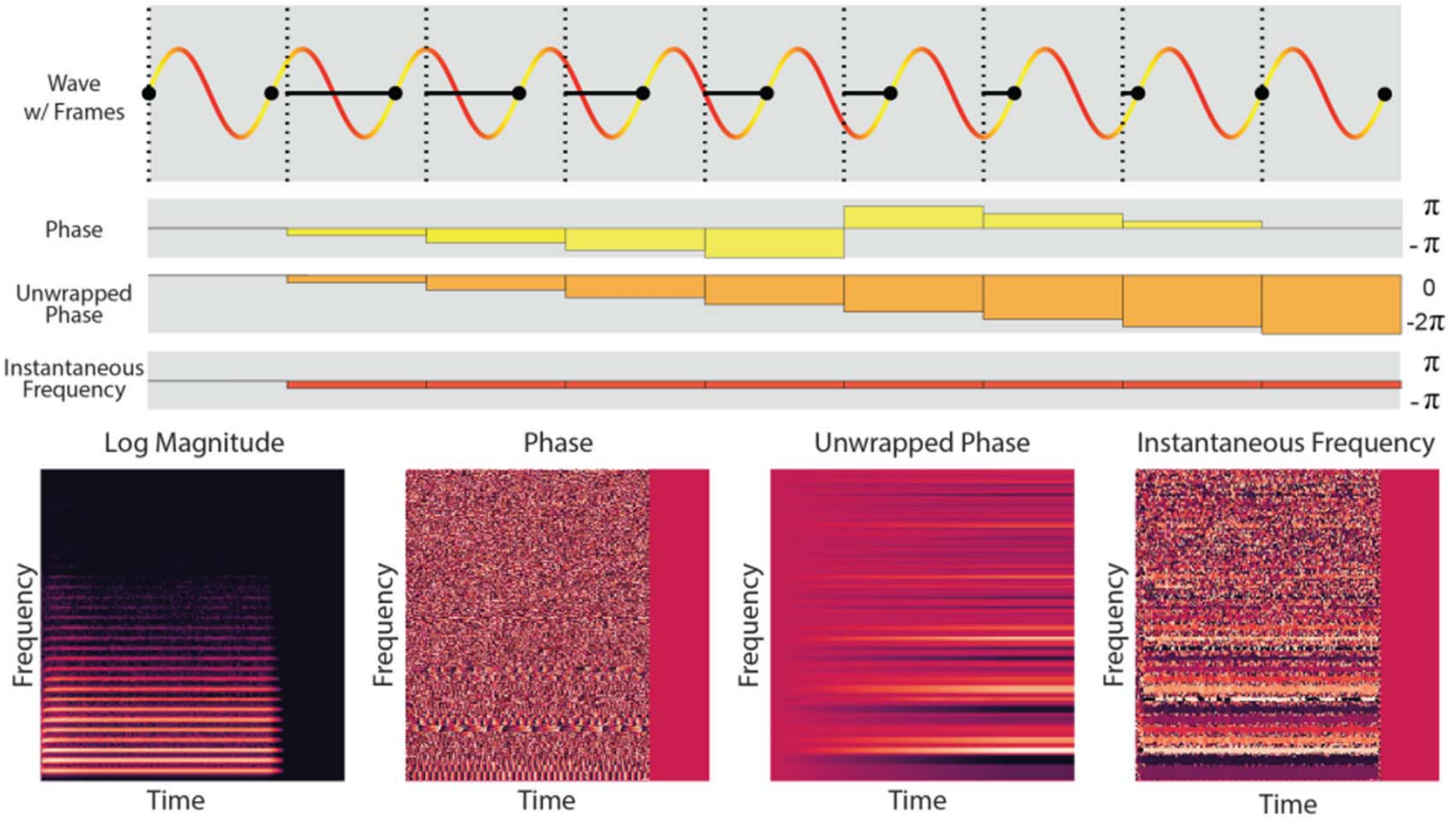
GANSynth (ICLR'19)

- **Non-autoregressive** generation using GAN
- Generate **magnitude spectrogram** & **phase**, instead of waveforms
- Use *Progressive growing GAN* (ICLR'18)
- "On the NSynth dataset, GANs can outperform a strong WaveNet baseline in automatic and human evaluations, and generate examples **~54,000** times faster"

Ref: Engel et al, "GANSynth: Adversarial neural audio synthesis," ICLR 2019

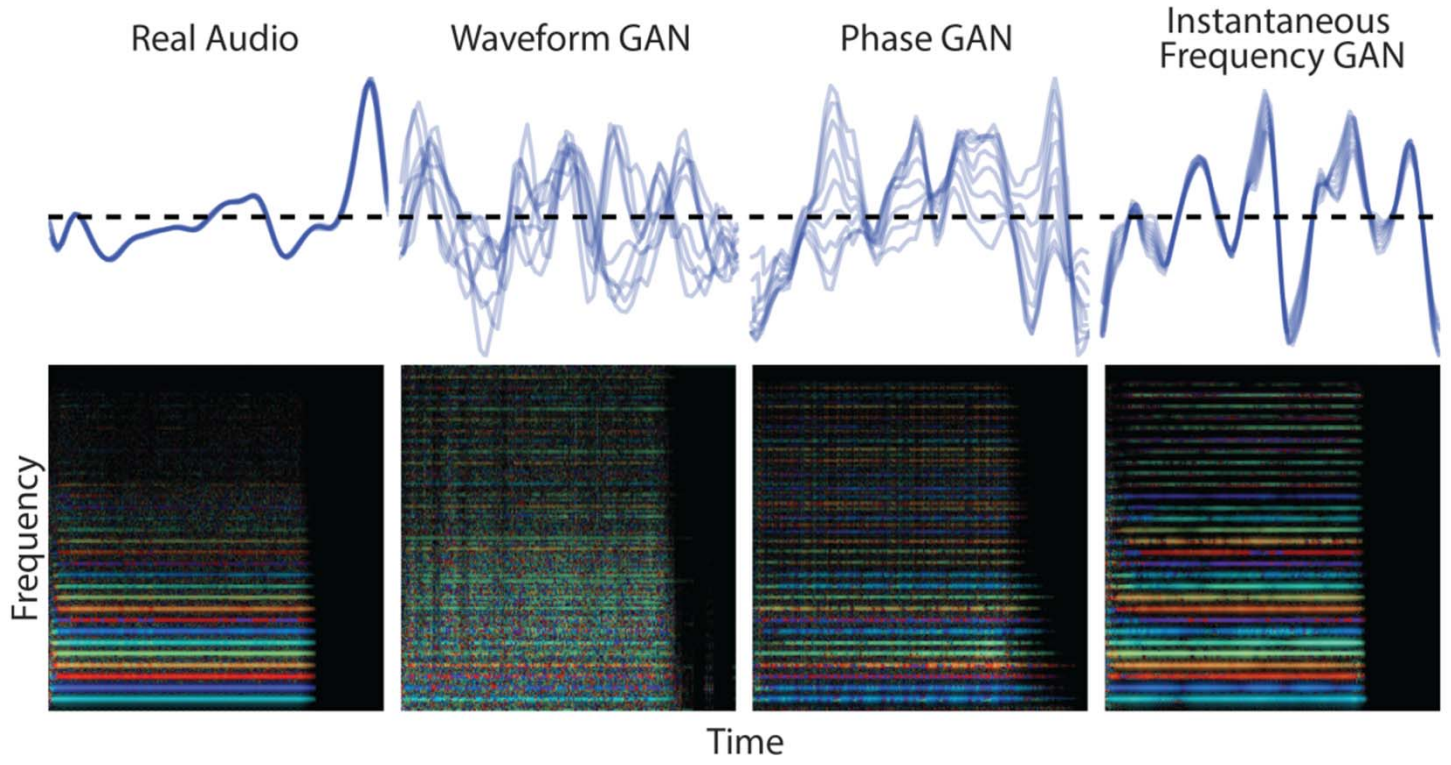
GANSynth (ICLR'19)

- Actually model the **instantaneous radial frequency (IF)**, not the phase
  - IF: unwrap the phase over the 2π boundary (orange boxes) and take its derivative



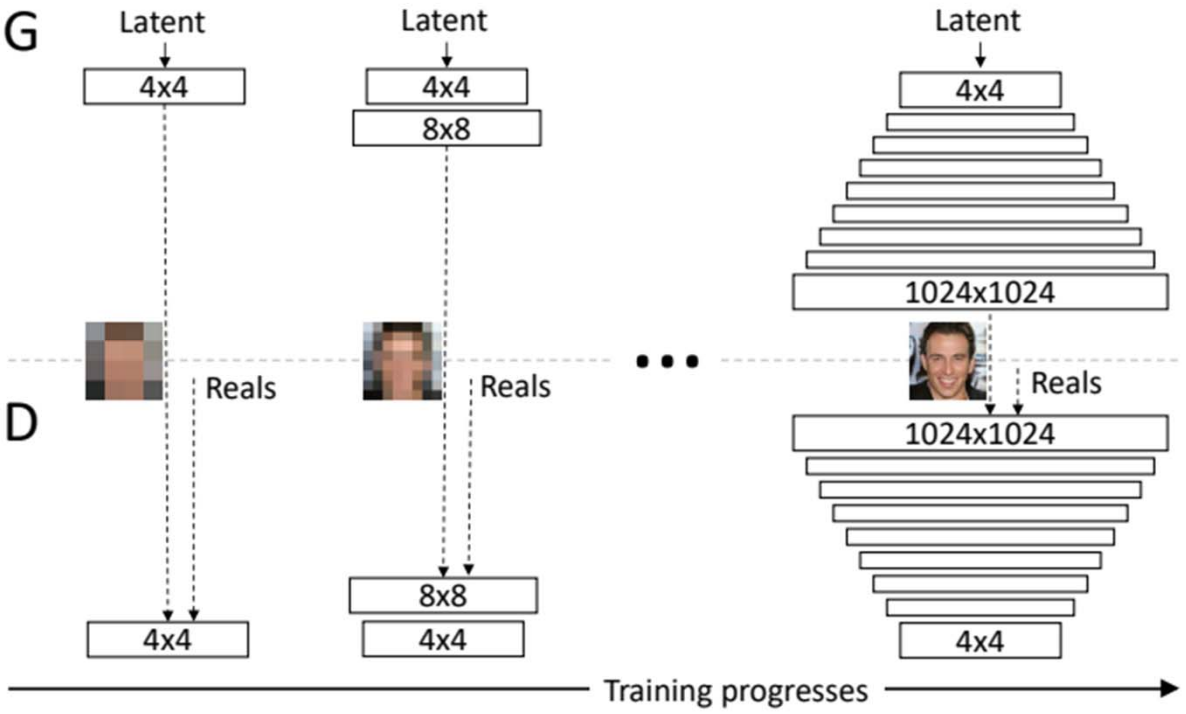
GANSynth (ICLR'19)

- Actually model the **instantaneous radial frequency (IF)**, not the phase
  - IF: unwrap the phase over the 2π boundary (orange boxes) and take its derivative



GANSynth (ICLR'19)

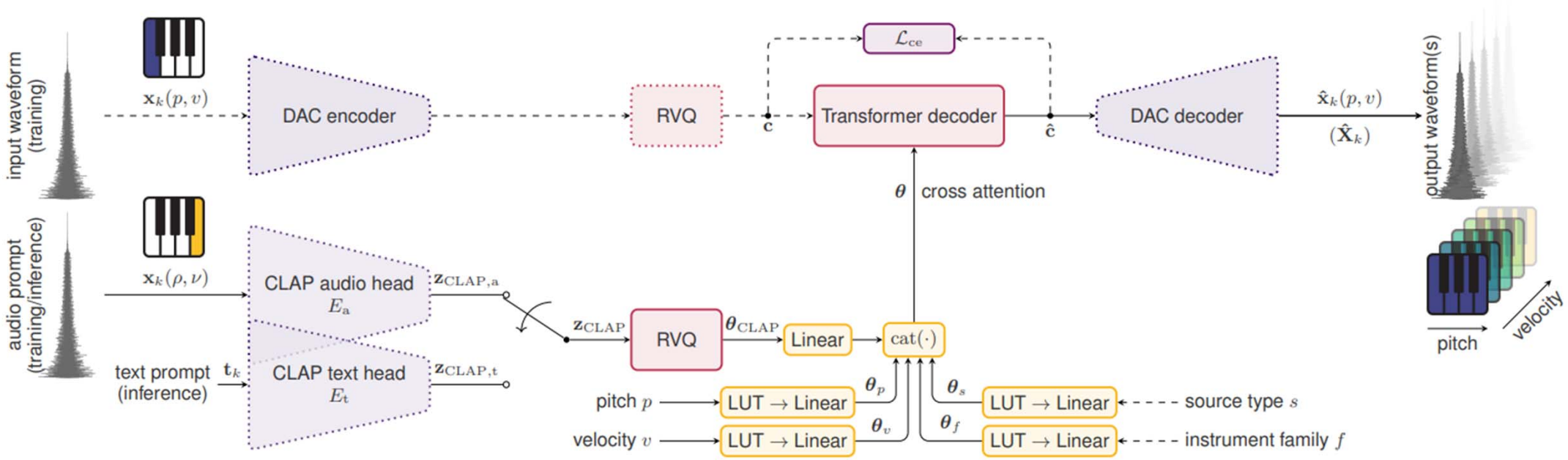
- Use *Progressive growing GAN* (PGGAN; ICLR'18)



Ref: Karras et al, "Progressive growing of GANs for improved quality, stability, and variation," ICLR 2018

InstrumentGen (ISMIR'24)

- Take pitch, velocity, and text as input



Ref: Nercessian et al, "Generating sample-based musical instruments using neural audio codec language models," ISMIR 2024

Recap: Generation of *single notes*

- "**Single MIDI pitch → audio**" (note generation)
  - **WaveNet autoencoder** (ICML'17): dilated convolution in waveform domain
  - **GANSynth** (ICLR'19): CNN on spectrograms
  - **InstrumentGen** (ISMIR'24): waveform domain

- "**Sequence of MIDI pitches → audio**" (MIDI-to-audio)
  - **PerformanceNet** (AAAI'19): CNN based
  - **Deep Performer** (ICASSP'22): treat it as singing voice synthesis → lecture 10
  - **MIDI-DDSP** (ICLR'22): use differential DSP (DDSP) → lecture 9
  - **DDSP-piano** (AES'23): use differential DSP (DDSP)

Outline

- DSP-based music synthesizers
- DL-based generation of single notes
- **DL-based generation of loops**
  - StyleGAN
- DL-based timbre transfer of short audio clips

Neural Generation of Loops

- **“x → audio”**
- Loops
  - Repeating section of sound material
  - Widely used in many genres (e.g., hip hop)

Figure from: https://www.magix.com/us/music-editing/music-production/effects/loops/

- More challenging than note synthesis
  - Longer
  - May not have associated MIDI
- Less challenging than *general* unconditional music generation
  - Fixed number of bars (e.g., 4-bar)
  - Basic building block of music

Progress in Image Generation

- **"x → image"**
  - PGGAN (ICLR'18)
  - StyleGAN (CVPR'19)
  - StyleGAN 2 (CVPR'20)
  - StyleGAN 3 (NeurIPS'21)
  - VQGAN (CVPR'21)
- **"class → image"**
  - BigGAN (ICLR'19)
  - ADM (diffusion; NeurIPS'21)
- **"text → image"**
  - DALL·E (ICML'21)
  - DALL·E 2 (arXiv'22)
  - Imagen (NeurIPS'22)
  - Stable Diffusion (CVPR'22)
  - MidJourney

Why GAN?

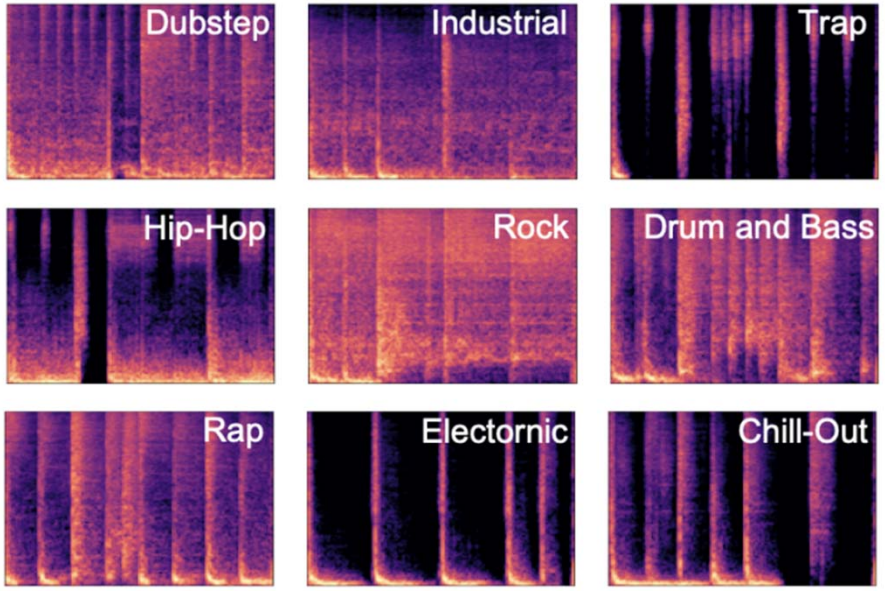
- GANs are advantageous when…
  1. **Hand-crafted loss functions are not good enough**
     - For example, reconstruction loss functions such as MSE can easily lead to over-smoothing result
     - Let the model (i.e., the discriminator) ***learn*** the loss function
     - This is the case for source separation and Mel-vocoders, for we don't want to rely on reconstruction loss functions only
  2. **There is no paired data; or there is no ground truth target**
     - This is the case for generation problems with weaker conditions
     - Such as "x → image", "class → image", "text → image", and **"x → audio"**
- Downside of GANs
  - Work well for **fixed-length data** (e.g., images) but not that good for **sequential data**
  - Need to build a **language model** (e.g., via **Transformers**) instead

Loop Generation by StyleGAN2 (ISMIR'21) (from our lab)

https://loopgen.github.io/

- **"x → audio"**
- Use StyleGAN2 (a non-autoregressive model) to generate **fixed-length** Mel-spectrograms
- Then use a Mel-vocoder (e.g., MelGAN) to get waveforms

- Here the Mel-spectrograms are not from real data, but are *generated*

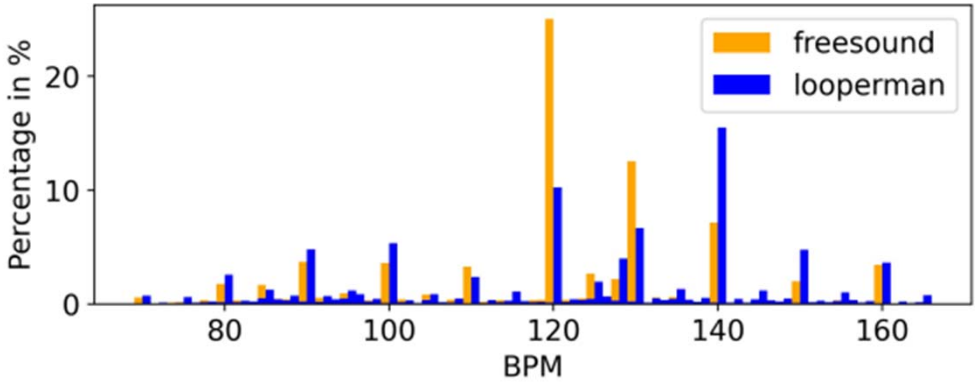


Ref: Hung et al, "A benchmarking initiative for audio-domain music generation using the FreeSound loop dataset," ISMIR 2021

Dataset: FreeSound Loop Dataset (FSLD)

https://zenodo.org/record/3967852

- 9,455 production-ready, public-domain CC-licensed loops
- 2,608 drum loops
- 13,666 **one-bar** loops

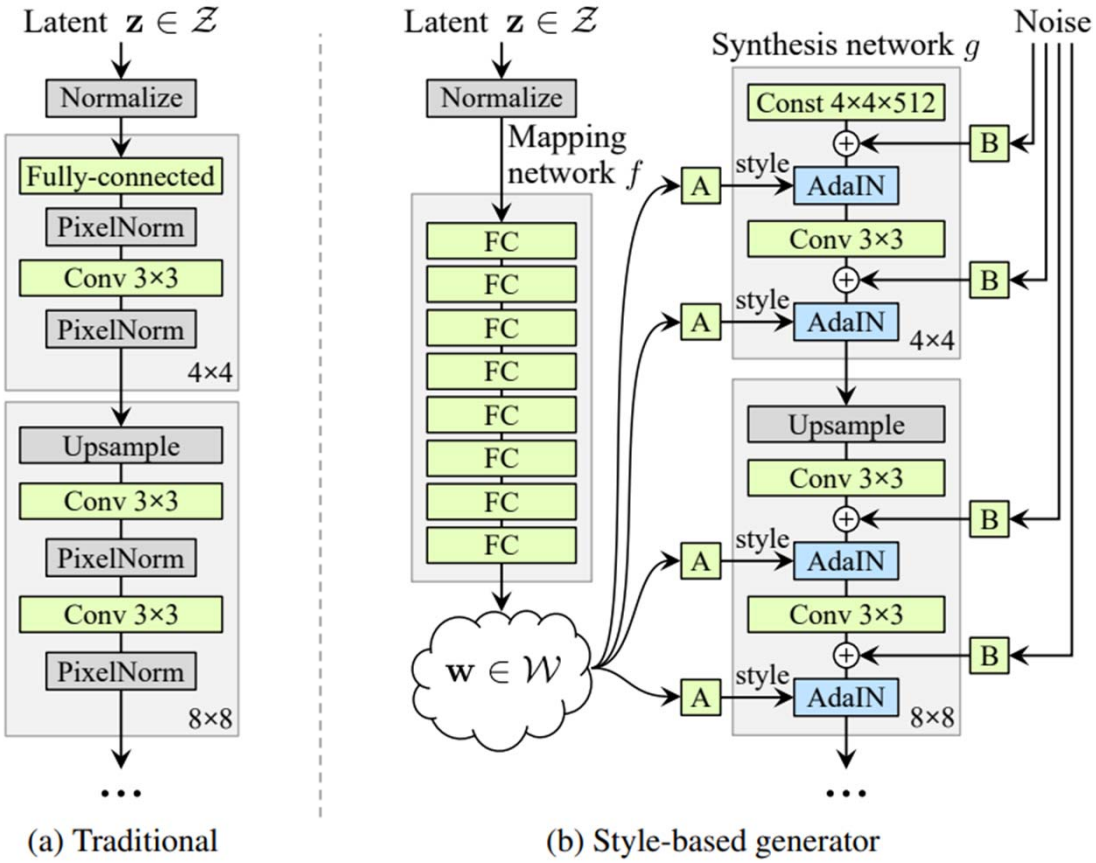


- All time-scaled to **120 BPM**
  - "We use pyrubberband to temporally stretch all the loops to **2-second** long […] most sounded plausible with little perceptible artifacts"
    - "This, however, may not be the case if the loops are not drum loops. Some data filtering might be needed then, e.g., to remove those whose tempo are much away from 120 BPM."

Ref: Hung et al, "A benchmarking initiative for audio-domain music generation using the FreeSound loop dataset," ISMIR 2021

StyleGAN & StyleGAN2

- Traditionally the latent code is provided to the generator through an input layer
- In StyleGAN, the latent code ***z*** is transformed into another latent ***w*** thru fully-connected layers and then used to control **AdaIN** operations after each convolution layer

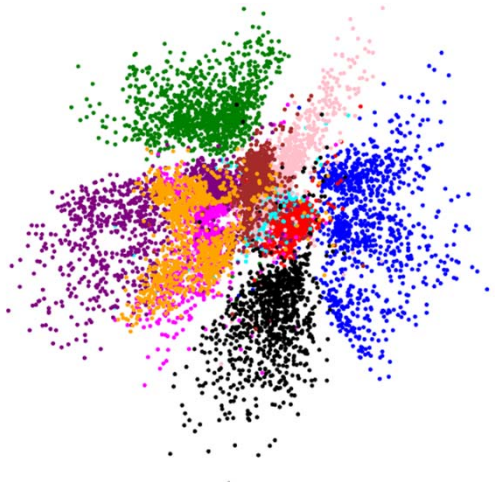


Ref 1: Karras et al, "A style-based generator architecture for generative adversarial networks," CVPR 2019
Ref 2: Karras et al, "Analyzing and improving the image quality of StyleGAN," CVPR 2020

AdaIN: Adaptive Instance Normalization

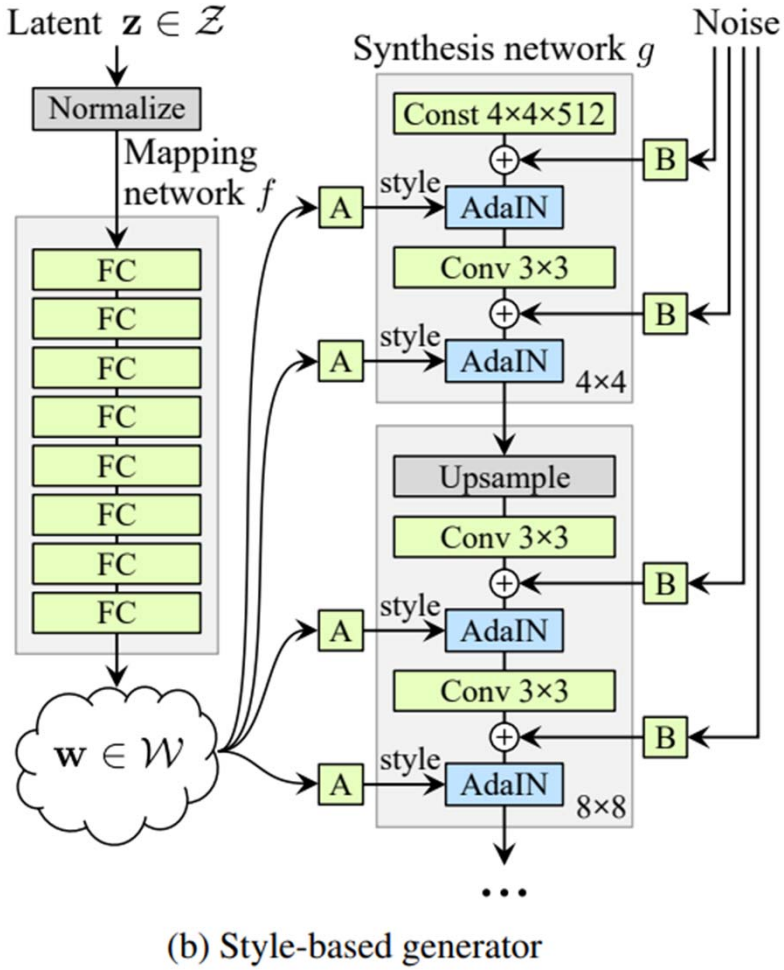
$$\mathrm{AdaIN}(\mathbf{x}_i, \mathbf{y}) = \mathbf{y}_{s,i} \frac{\mathbf{x}_i - \mu(\mathbf{x}_i)}{\sigma(\mathbf{x}_i)} + \mathbf{y}_{b,i},$$

- Each feature map $\boldsymbol{x_i}$ is **normalized** separately, and then **scaled** and **biased** using the corresponding scalar components from style $\boldsymbol{y}$



Latent space visualization of the 10 MNIST digits in 2 dimensions

https://www.researchgate.net/figure/Latent-space-visualization-of-the-10-MNIST-digits-in-2-dimensions-of-both-N-VAE-left_fig2_324182043



(b) Style-based generator

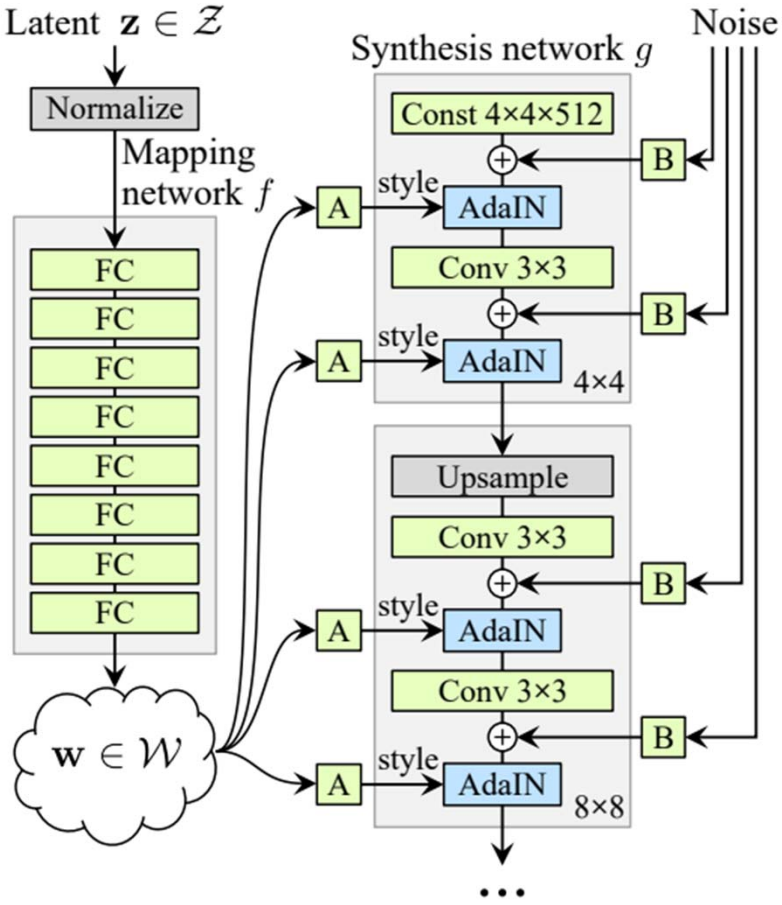
StyleGAN: Interpolation

https://github.com/justinpinkney/stylegan-matlab-playground/blob/master/examples/interpolation.md

“Style Mixing” w/ Latent Codes in Different Resolutions

- $\boldsymbol{z}_1 \rightarrow \{A_1^{low}, A_1^{mid}, A_1^{high}\} \rightarrow$ image 1
- $\boldsymbol{z}_2 \rightarrow \{A_2^{low}, A_2^{mid}, A_2^{high}\} \rightarrow$ image 2

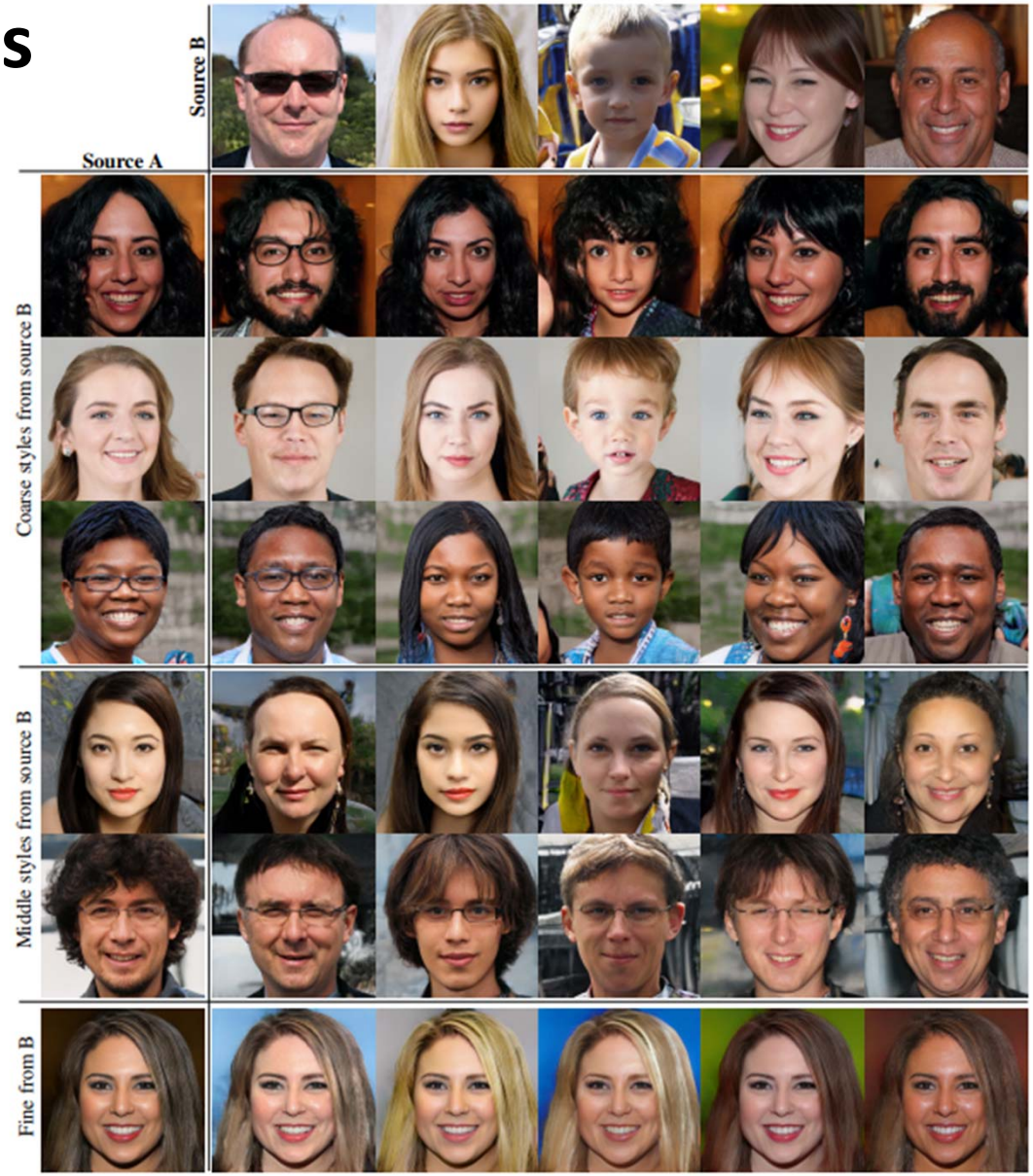
- $\{A_1^{low}, A_1^{mid}, A_2^{high}\} \rightarrow$?



(b) Style-based generator

“Style Mixing” w/ Latent Codes in Different Resolutions

- Copying the styles from coarse spatial resolutions (4^2– 8^2):
  - high-level aspects such as pose, general hair style, face shape, and eyeglasses
- Copying the styles of middle resolutions (16^2–32^2):
  - smaller scale facial features, hair style, eyes open/closed
- Copying the fine styles (64^2–1024^2)
  - color scheme and microstructure



StyleGAN: Effect of Noise

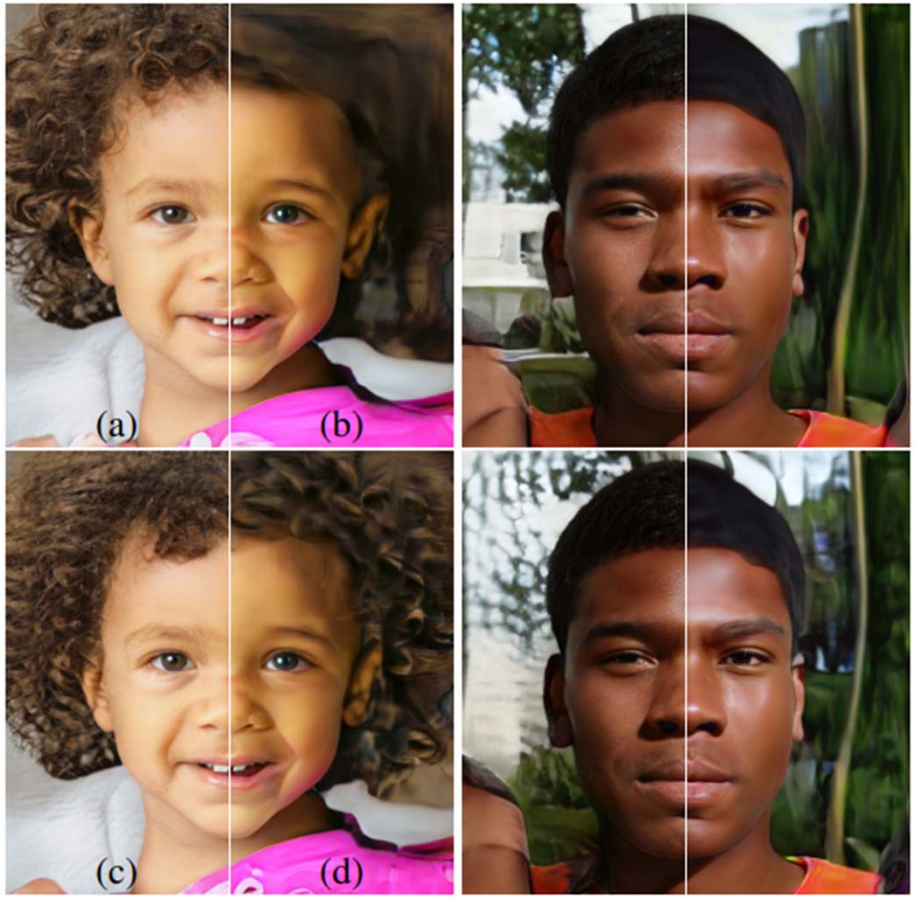


Figure 5. Effect of noise inputs at different layers of our generator. (a) Noise is applied to all layers. (b) No noise. (c) Noise in fine layers only (64^2 – 1024^2). (d) Noise in coarse layers only (4^2 – 32^2). We can see that the artificial omission of noise leads to featureless "painterly" look. Coarse noise causes large-scale curling of hair and appearance of larger background features, while the fine noise brings out the finer curls of hair, finer background detail, and skin pores.

StyleGAN & StyleGAN2: Ablations

	Method	CelebA-HQ	FFHQ
A	Baseline Progressive GAN [30]	7.79	8.04
B	+ Tuning (incl. bilinear up/down)	6.11	5.25
C	+ Add mapping and styles	5.34	4.85
D	+ Remove traditional input	5.07	4.88
E	+ Add noise inputs	**5.06**	4.42
F	+ Mixing regularization	5.17	**4.40**

Table 1. Fréchet inception distance (FID) for various generator designs (lower is better). In this paper we calculate the FIDs using 50,000 images drawn randomly from the training set, and report the lowest distance encountered over the course of training.

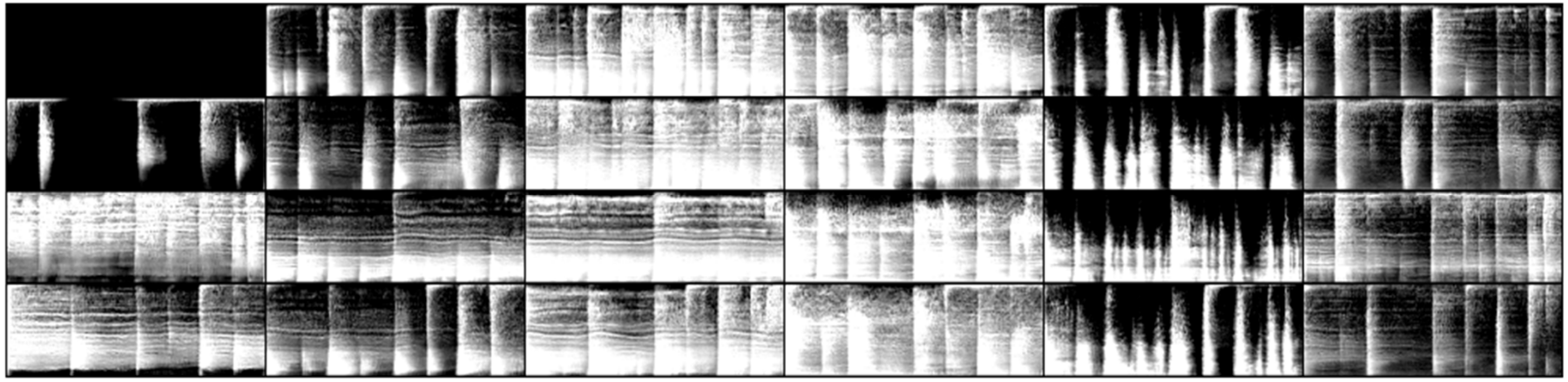
	Configuration	FID ↓
A	Baseline StyleGAN [24]	4.40
B	+ Weight demodulation	4.39
C	+ Lazy regularization	4.38
D	+ Path length regularization	4.34
E	+ No growing, new G & D arch.	3.31
F	+ Large networks (StyleGAN2)	**2.84**
	Config A with large networks	3.98

Ref 1: Karras et al, "A style-based generator architecture for generative adversarial networks," CVPR 2019
Ref 2: Karras et al, "Analyzing and improving the image quality of StyleGAN," CVPR 2020

Loop Generation by StyleGAN2

https://loopgen.github.io/

- Random generation
- *Style mixing*
  - The interpolated loop resembles the source audio in terms of **timbre**, and resembles the target audio in its **rhythmic pattern**



Ref: Hung et al, “A benchmarking initiative for audio-domain music generation using the FreeSound loop dataset,” ISMIR 2021

Loop Generation by StyleGAN2

https://github.com/allenhung1025/LoopTest

- ***Easy to try*** *(recommended)*
  - "The total training time is 100 hr on an **NVIDIA GTX1080 GPU** with 8GB memory."

- We also provide code for objective evaluation metrics
  - **Inception score**
  - **FAD**
  - Two other **diversity metrics**
    - Number of statistically-different bins (**NDB**)
    - Jensen-Shannon divergence (**JSD**)

	IS ↑	FAD ↓	JS ↓	NDB/K ↓
Looperman	11.9±3.21	0.11	0.01	0.01
Freesound	6.30±1.82	0.72	0.08	0.46
StyleGAN	1.31±1.95	13.78	0.43	0.94
StyleGAN2	**5.24±1.84**	7.91	**0.09**	**0.59**
UNAGAN	3.33±1.65	**4.32**	0.16	0.73

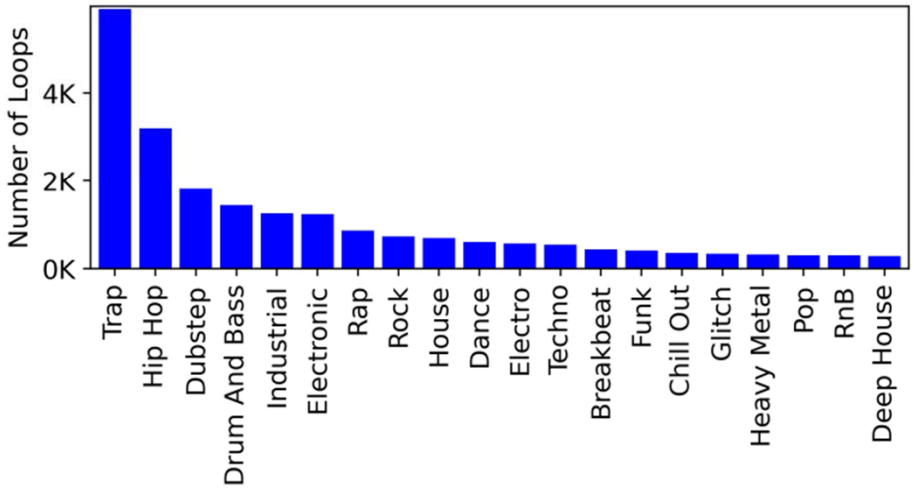
Ref: Hung et al, "A benchmarking initiative for audio-domain music generation using the FreeSound loop dataset," ISMIR 2021

Objective Evaluation Metric: Inception Score (IS)

- A ***reference-free*** evaluation metric for audio generation models
- Measures the **quality** + **diversity** of the generated data and detects ***mode collapse*** by using a pre-trained *domain-specific* classifier
  - Computed as the KL divergence between the conditional probability $p(y|x)$ and marginal probability $p(y)$,

$$\mathrm{IS} = \exp\left(E_x[\mathrm{KL}(p(y|x)\|p(y)]\right),$$

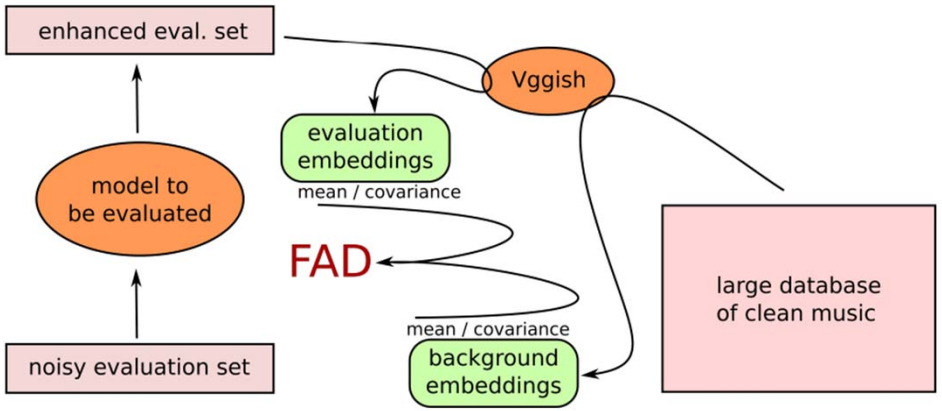
- To compute the IS, we use short-chunk CNN to train a 66-class classifier for **loop genre** classification



Ref: Salimans et al, "Improved techniques for training GANs," NeurIPS 2016

Objective Evaluation Metric: Fréchet Audio Distance (FAD)

- Also *reference-free*
- The discrepancy of the ***multivariate Gaussians*** **distributions** of real and generated data in an embedding space computed by a pretrained VGGish-based audio classifier



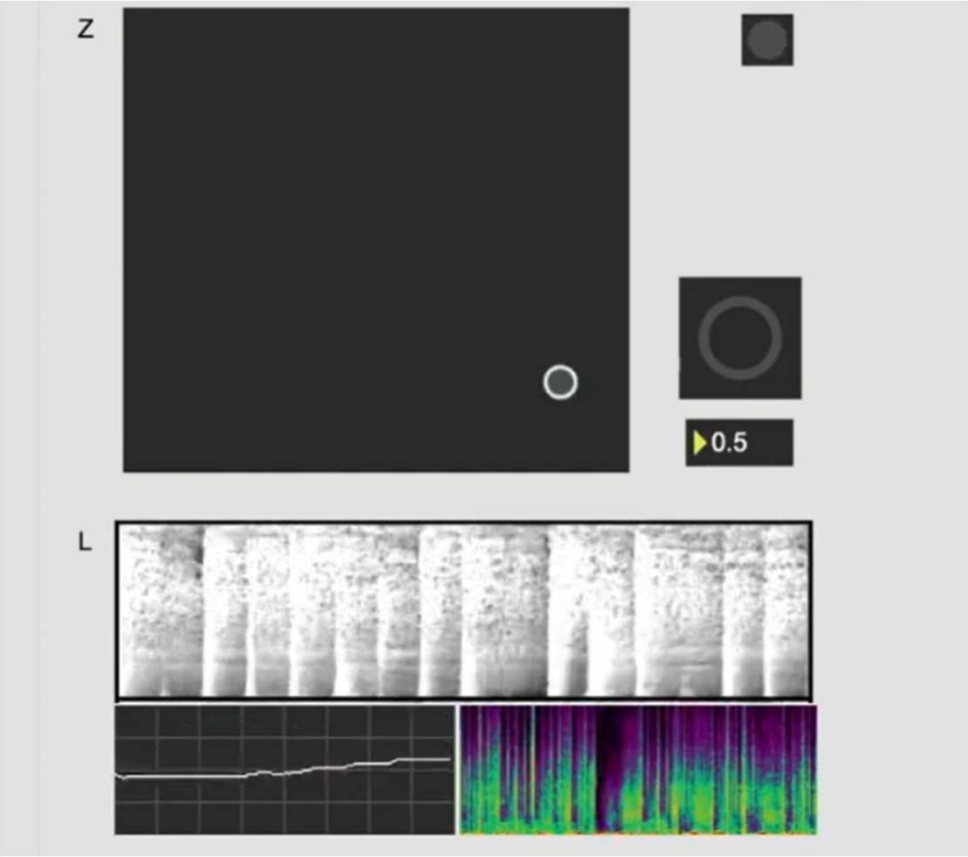
$$\mathbf{F}(\mathcal{N}_b, \mathcal{N}_e) = \|\mu_b - \mu_e\|^2 + tr(\Sigma_b + \Sigma_e - 2\sqrt{\Sigma_b \Sigma_e})$$

  - The VGGish is an audio classifier over 3,000 classes trained on a large dataset of YouTube videos (i.e., *not domain-specific*). The activations from the 128 dimensional layer prior to the final classification layer are used as the embedding.
  - Measures **quality** + **diversity**

Ref: Kilgour et al, "Fréchet Audio Distance: A metric for evaluating music enhancement algorithms," arXiv 2016

Loop Generation: Exploring the Latent Space

https://www.youtube.com/watch?app=desktop&v=Sg3tIwPjPBI



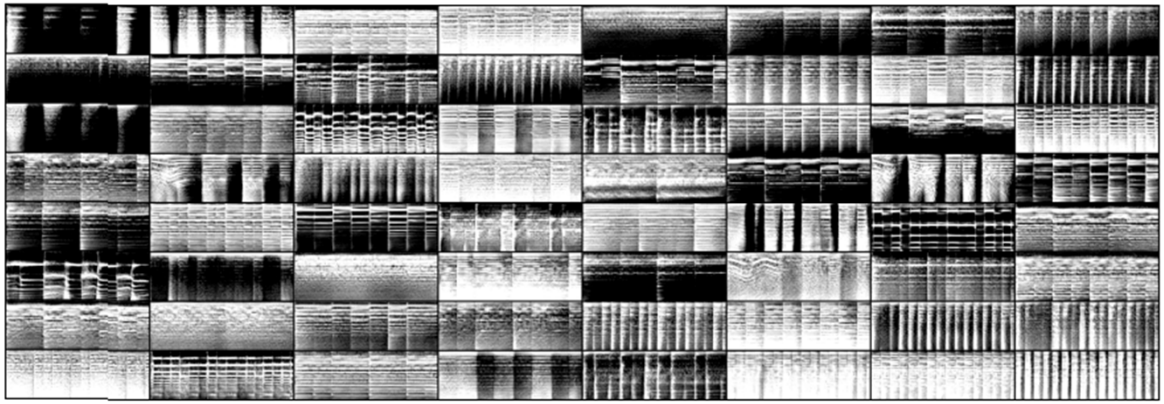
By Nao Tokui (2023)

Loop Generation by Projected GAN (ISMIR'22) (from our lab)

https://arthurddd.github.io/PjLoopGAN/

- Use **projected GAN** to speed up and stabilize GAN training
  - With Projected GAN, the loop generation models can converge around 5 times faster without performance degradation

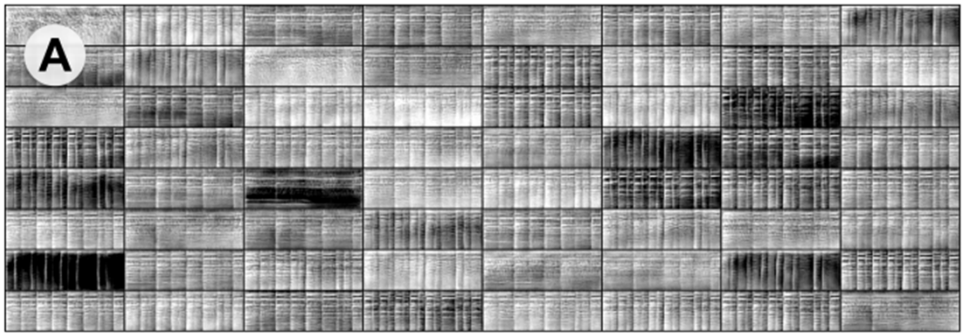
- Consider the generation of not only drum loops but also **synth loops**

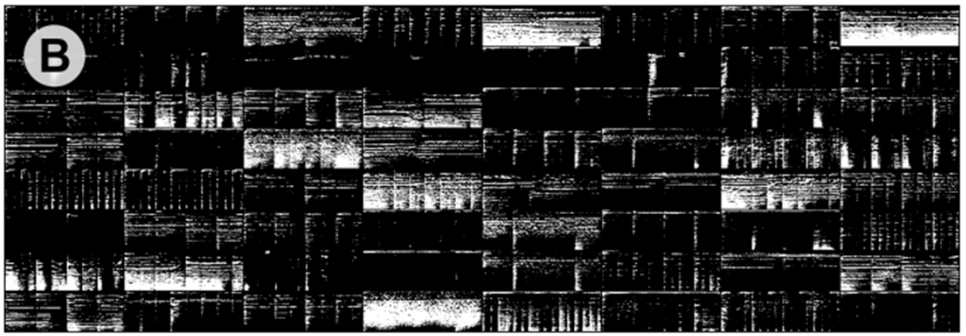


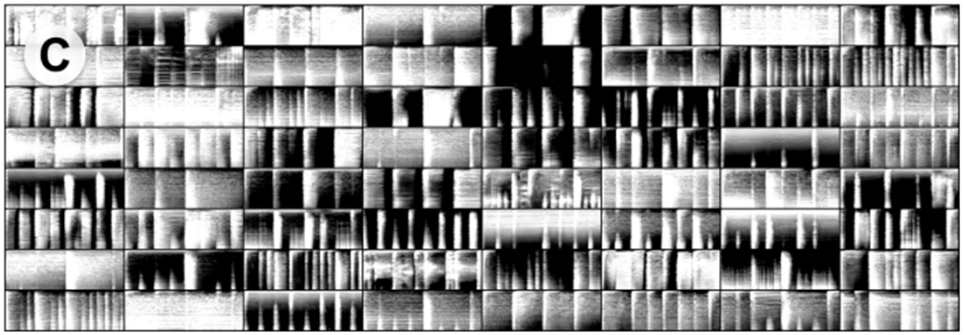
Ref: Yeh et al, "Exploiting pre-trained feature networks for generative adversarial networks in audio-domain loop generation," ISMIR 2022

Different Cases of GAN Training

- Also studied in the Projected GAN paper for loop generation



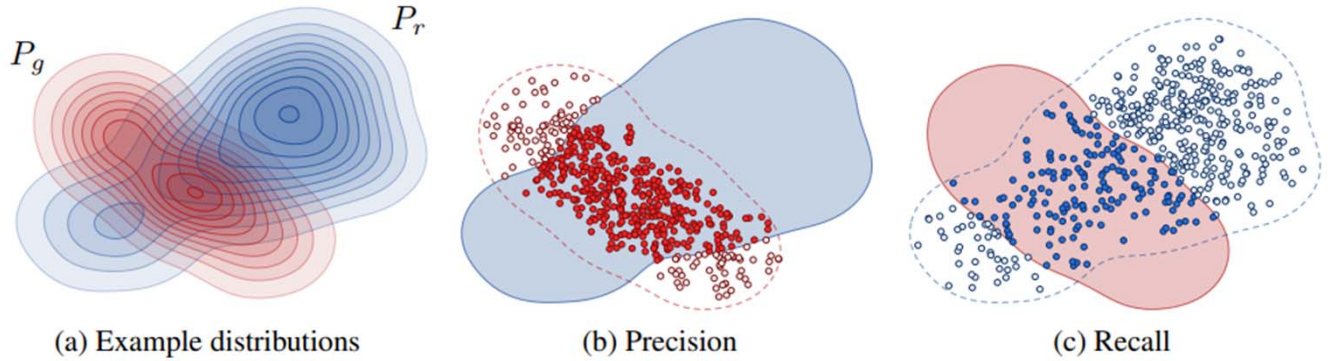




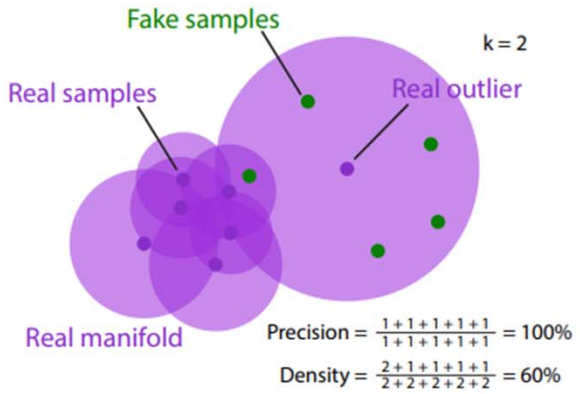
- Examples from Mel-spectrogram generation (not Mel-vocoder)
  - (A) Mode collapse
  - (B) Gradient vanishing
  - (C) Successful training

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

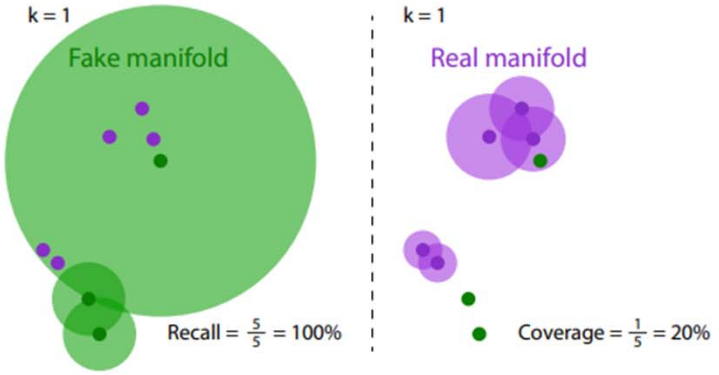
More Evaluation Metrics for Unconditional Generation



(a) Example distributions (b) Precision (c) Recall



(a) Precision versus density.



(b) Recall versus coverage.

Ref 1: Kynkäänniemi et al., "Improved precision and recall metric for assessing generative models," NeurIPS 2019
Ref 2: Naeem et al., "Reliable fidelity and diversity metrics for generative models," ICML 2020.

Advanced Topics for Loop Generation

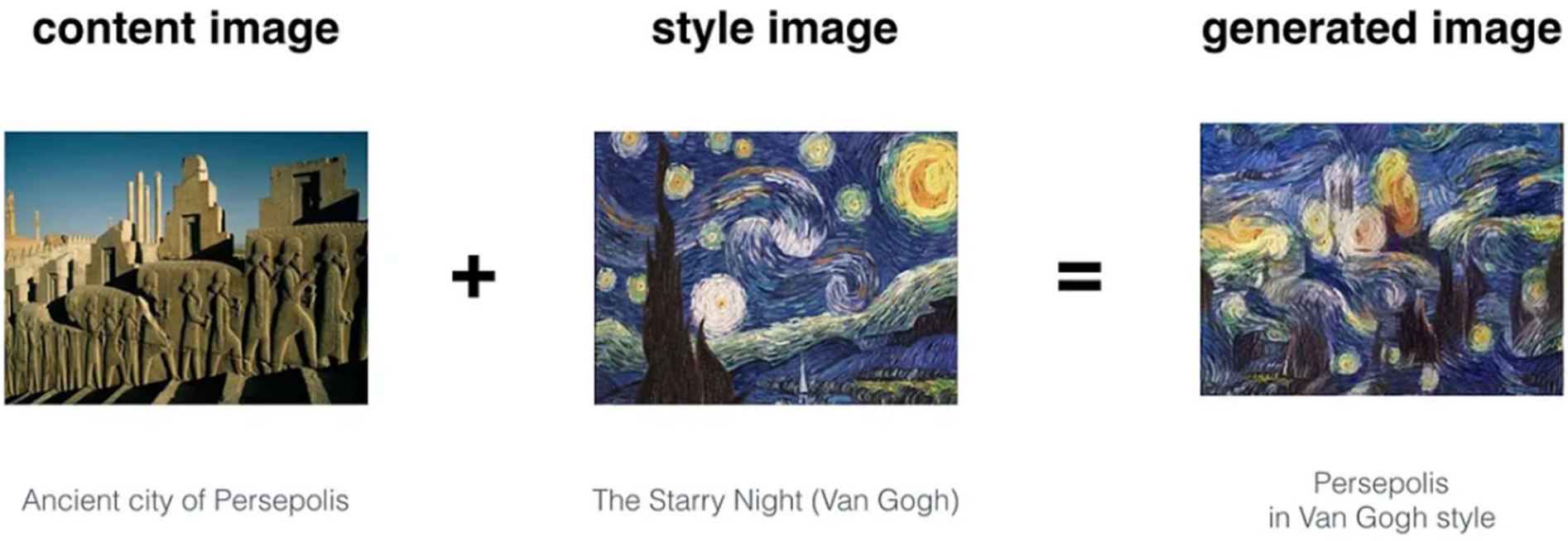
- We have only considered loop generation as a fixed-length, unconditional audio generation problem
  - "**x → audio of loop**"
- Loop generation can also be ***class-conditioned***
  - "**MIDI pitch → audio of single note**"
  - "**genre class** (e.g., rap, trap, dubstep) **→ audio of loop**"
  - "**loop role** (e.g., percussive, melody, bass, chord, FX) [1] **→ audio of loop**"
- Loop generation can also be of ***variable-length*** (various BPM, or #bars)
  - May need a **language model** (LM)
  - See forthcoming lectures

[1] Ching et al, "Instrument role classification: Auto-tagging for loop based music," CSMC 2020

Outline

- DSP-based music synthesizers
- DL-based generation of single notes
- DL-based generation of loops
- **DL-based timbre transfer of short audio clips**
  - DDSP → lecture 9
  - RAVE
  - FiLM-TCN
  - Hyper-RNN

Style Transfer: “Content” vs “Style”



https://towardsdatascience.com/neural-style-transfer-on-real-time-video-with-full-implementable-code-ac2dbc0e9822

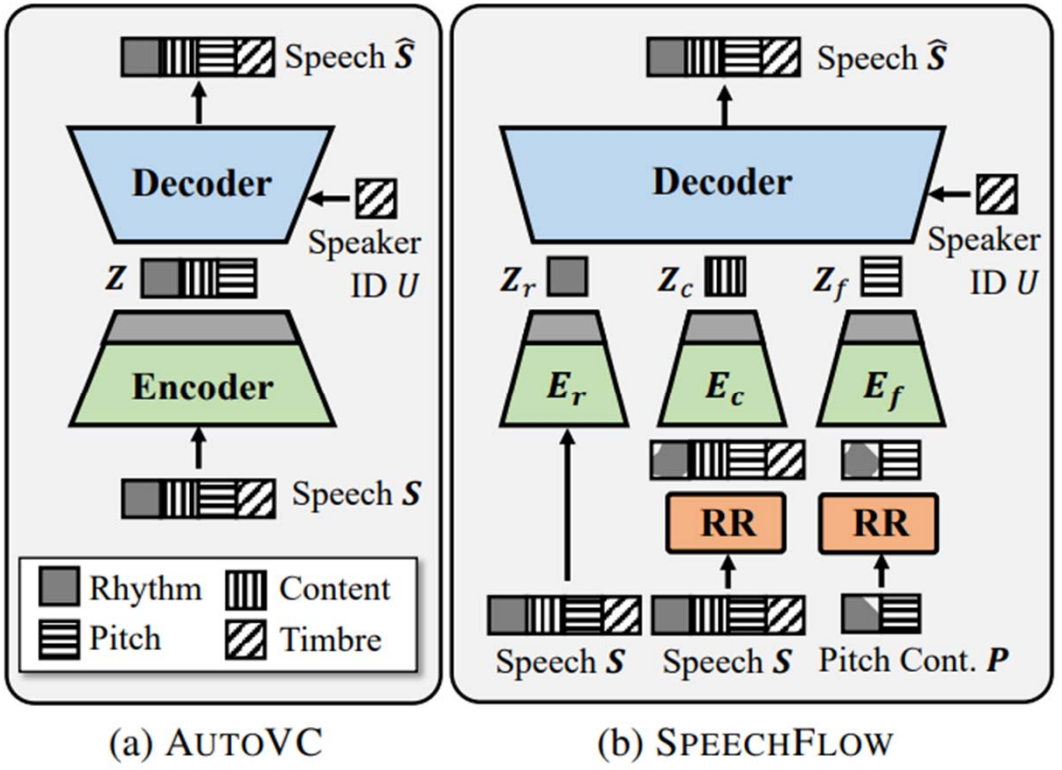
Style Transfer: “Paired Data” vs “Non-paired Data”

- Paired data
  - Examples
    - We have images of the same content but different styles
    - We have different speakers speaking the same sentence
  - In such cases, we can use them as input & output respectively to train the model

- Non-paired data
  - Can only use reconstruction loss to train the encoder/decoder
  - The encoder/decoder never learns to do style transfer
  - However, due to some mechanisms for **content/style disentanglement**, style transfer can still be achieved

Decomposition of Elements of Speech

- Speech = pitch + rhythm + content (phoneme) + timbre
- AutoVC
  - decoder can readily get timbre (style) information from the speaker ID
  - decoder only need non-style information from **z** (the encoder output) so as to reconstruct the input audio
  - encoder learns to be timbre invariant



Ref1: Qian et al., "AUTOVC: Zero-shot voice style transfer with only autoencoder loss," ICML 2019

Ref2: Qian et al., "Unsupervised speech decomposition via triple information bottleneck," ICML 2020

Representation of Timbre: Timbre Decoder
(embeddings → audio)

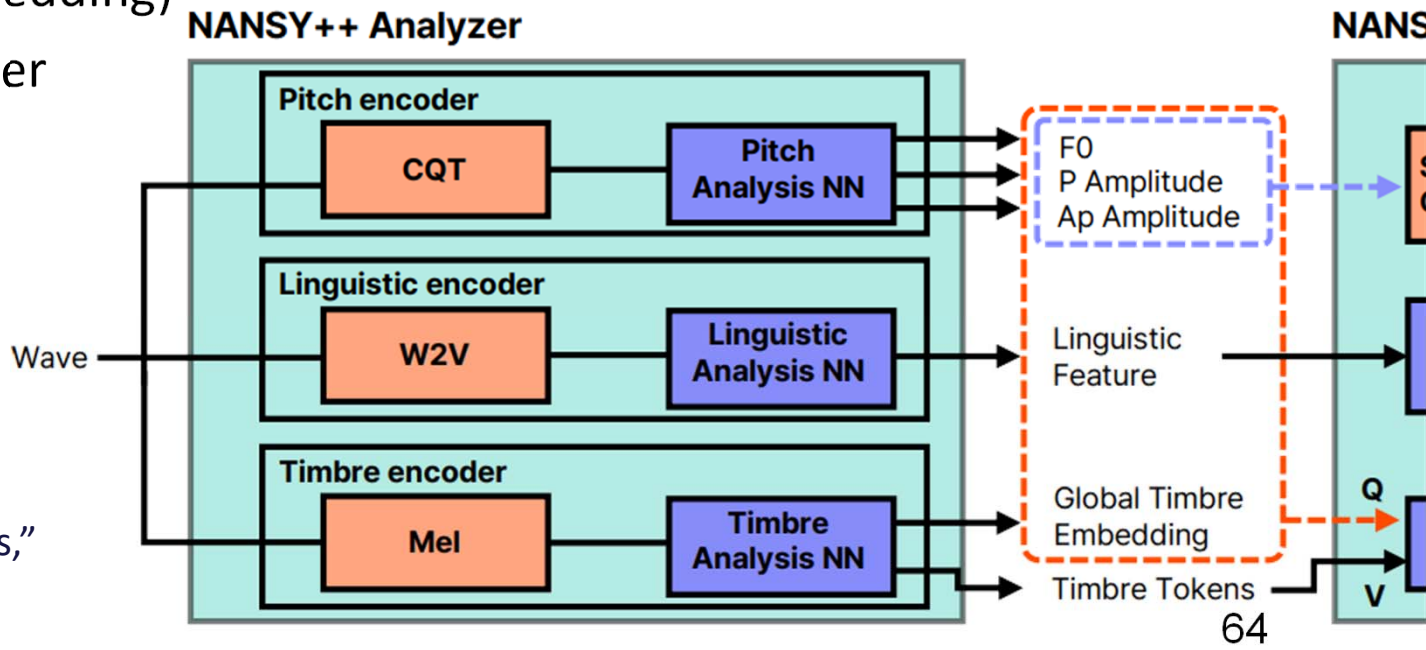
- Generate audio given some **“timbre-invariant” embeddings**
- Example: **“many-to-one” voice conversion**
  - Use output of automatic speech recognition (ASR) models, a.k.a., phonetic posteriorgrams (PPG), as the model input
    - Note: ASR should focus on the linguistic content, rather than the speaker timbre
  - The decoder **“injects” timbre information** while creating the output

Ref: Liu et al., “Any-to-many voice conversion with location-relative sequence-to-sequence modeling,” TASLP 2021

Representation of Timbre: Timbre Encoder

(audio → embeddings)

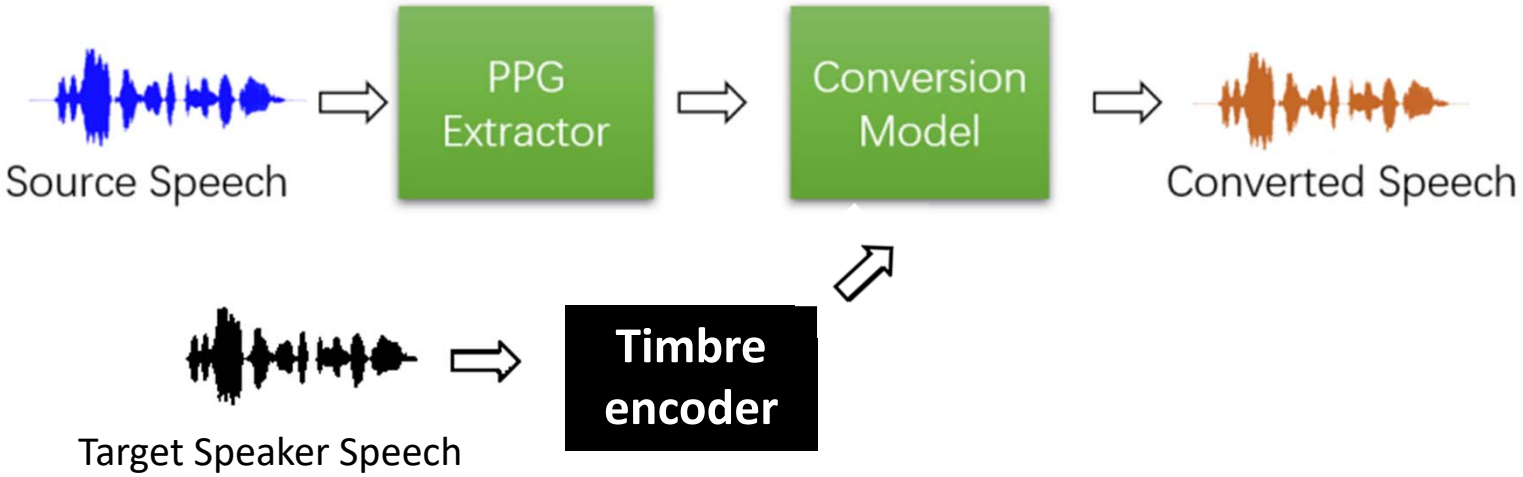
- Turn audio into **timbre embeddings**
- *Only* encode timbre information, while ignoring the others
- Several methods
  - speaker ID (one-hot embedding)
  - embedding from a speaker verification model
  - embedding from contrastive learning



Ref: Choi et al., "NANSY++: Unified voice synthesis with neural analysis and synthesis," ICLR 2023

Timbre Decoder vs. Timbre Encoder

- **Many-to-one** voice conversion
  - **Content** encoder
  - **Timbre** decoder
- **Many-to-many voice** conversion
  - **Content** encoder
  - **Timbre** encoder (for style condition)
  - Decoder



DDSP-based Timbre Transfer

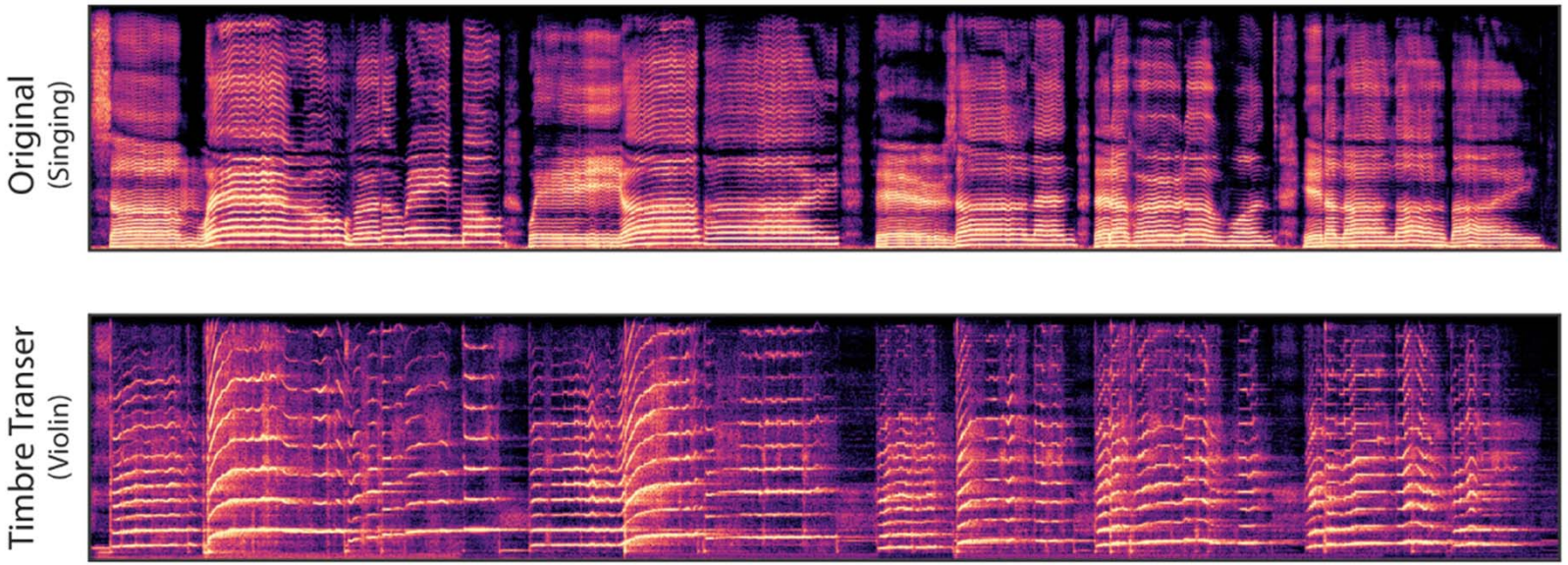
https://sites.research.google/tonetransfer

Tone Transfer — Behind the Scenes

https://magenta.tensorflow.org/tone-transfer

https://magenta.tensorflow.org/ddsp

- Keeping the **frequency** and **loudness** *the same* and changing the sound character



Ref: Engel et al., “DDSP: Differentiable digital signal processing,” ICLR 2020

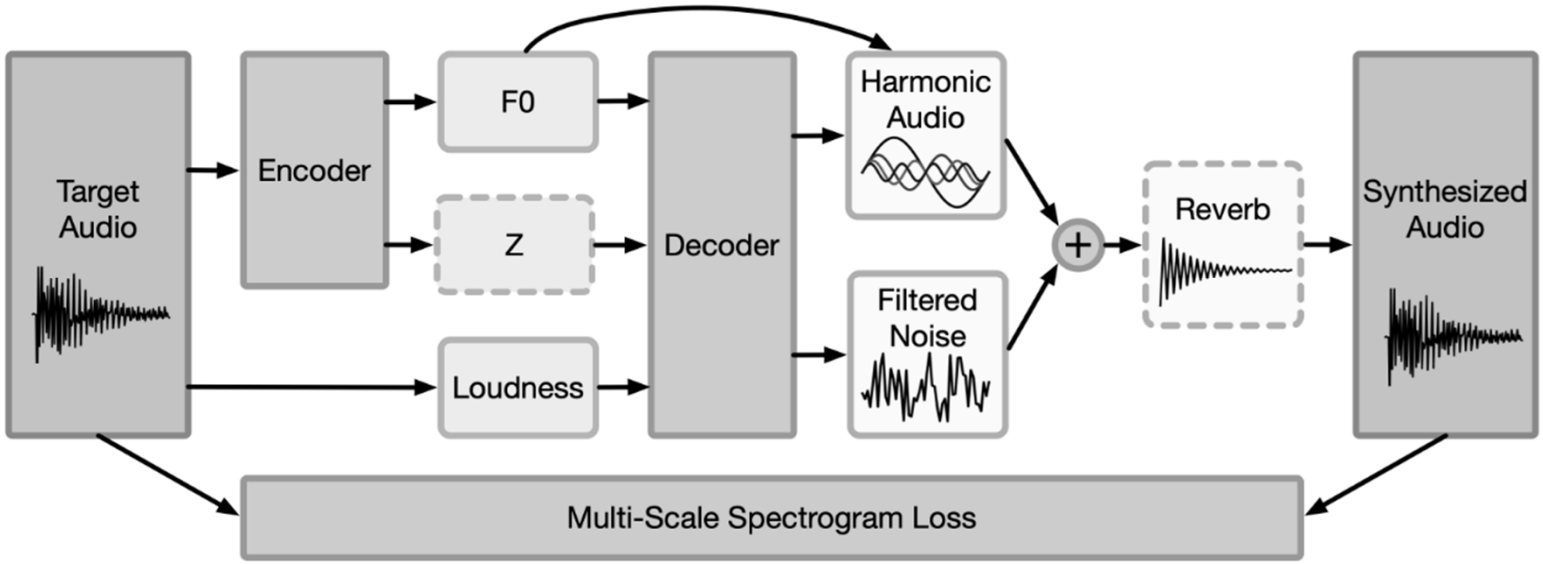
Decomposition of Elements of Music (ICLR'20)

(for monophonic signals)

https://magenta.tensorflow.org/ddsp

- **Music = F0 (pitch + duration) + loudness + "z" (timbre)**

content *style*

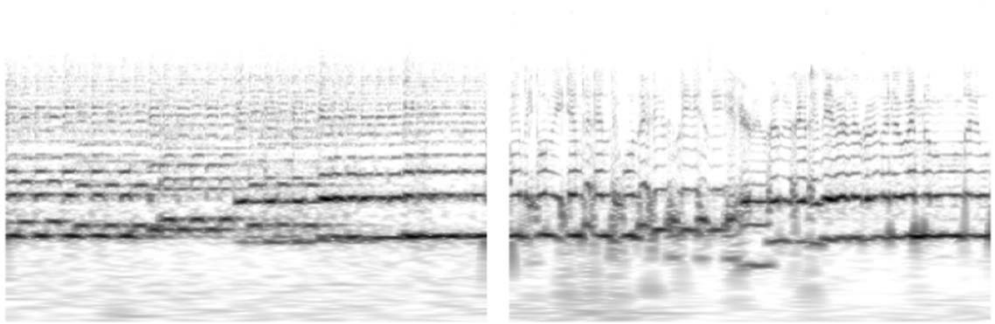


Ref: Engel et al., "DDSP: Differentiable digital signal processing," ICLR 2020

RAVE (arXiv'21)

https://anonymous84654.github.io/RAVE_anonymous/

Strings to speech transfer



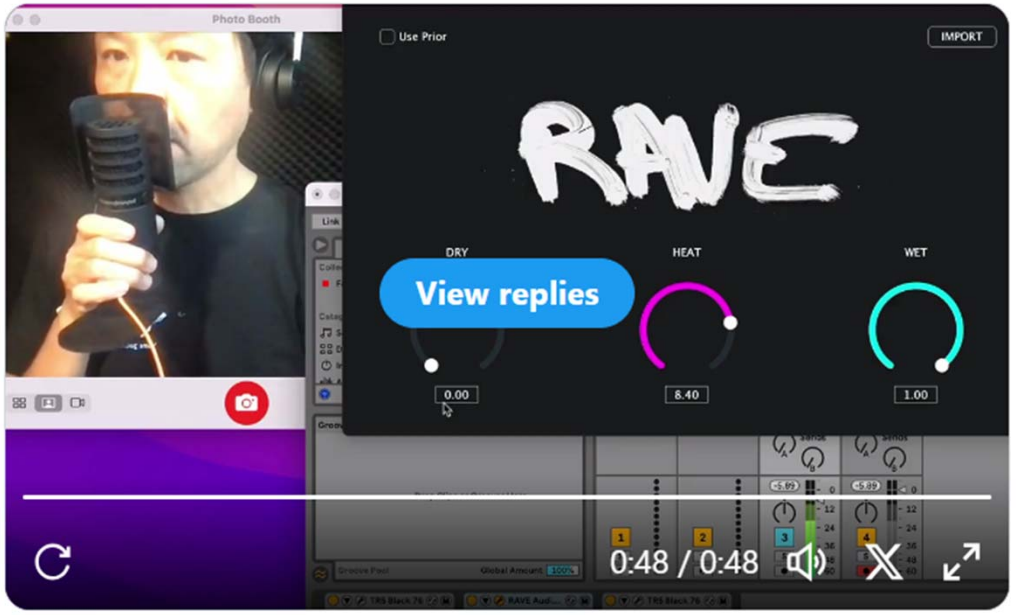
original

0:00 / 0:04

reconstructed

0:00 / 0:04

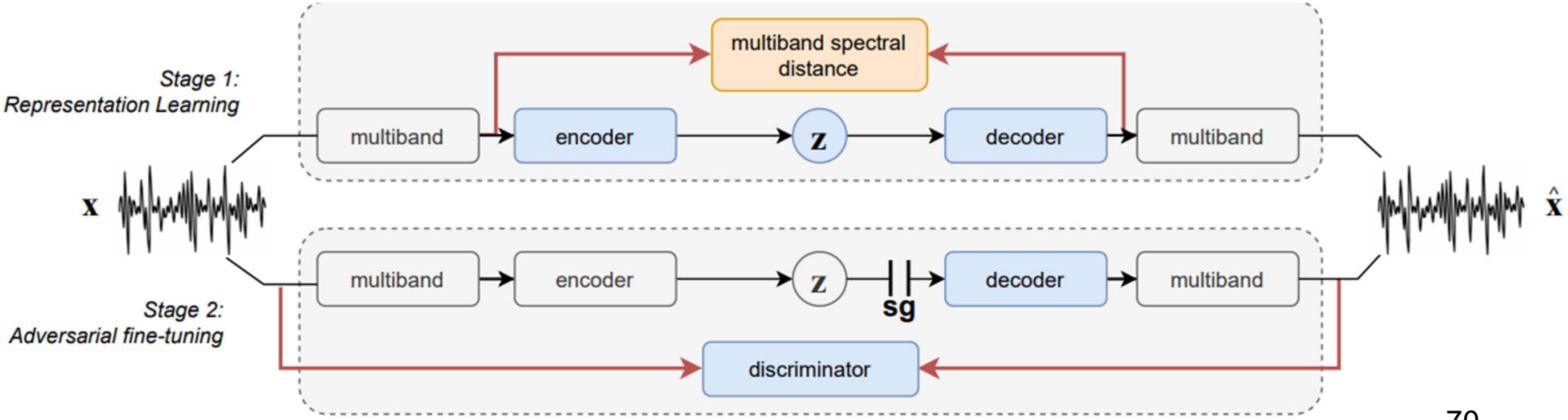
Realtime Voice to Darbuka transfer



Ref: Caillon et al, "RAVE: A variational autoencoder for fast and high-quality neural audio synthesis," arXiv 2021

RAVE: VAE+GAN

- Only “**z**”
- Two-stage training procedure
  - First trained as a **regular VAE** for representation learning
  - Then fine-tuned with an **adversarial** generation objective in order to achieve high quality audio synthesis



RAVE: Style Transfer

5.4 Timbre transfer

We demonstrate that RAVE can be used to perform domain transfer even if it has not been specifically trained to address this particular task. We perform domain transfer by simply providing to a pretrained RAVE model audio samples coming from outside of its original training distribution (e.g violin samples are reconstructed with a model trained on speech, see section F for more details).

Neural audio style transfer (and messing with latent ...

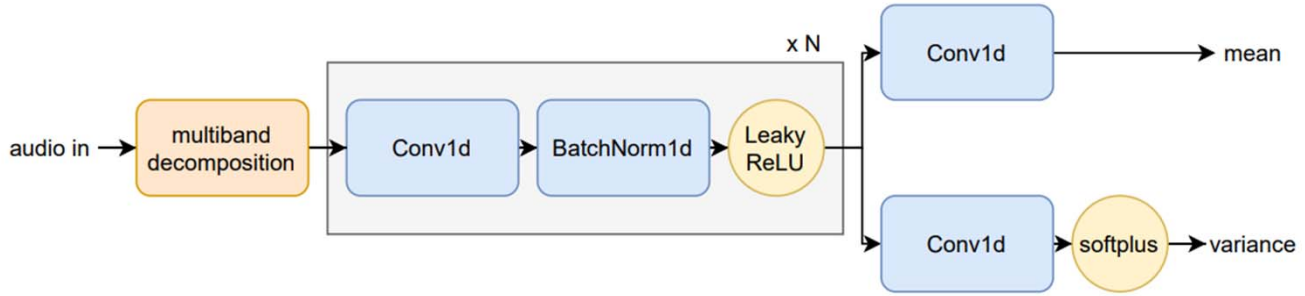
YouTube · martsman (MARTSM_N)
1 個月前

Neutone RAVE AI audio style transfer

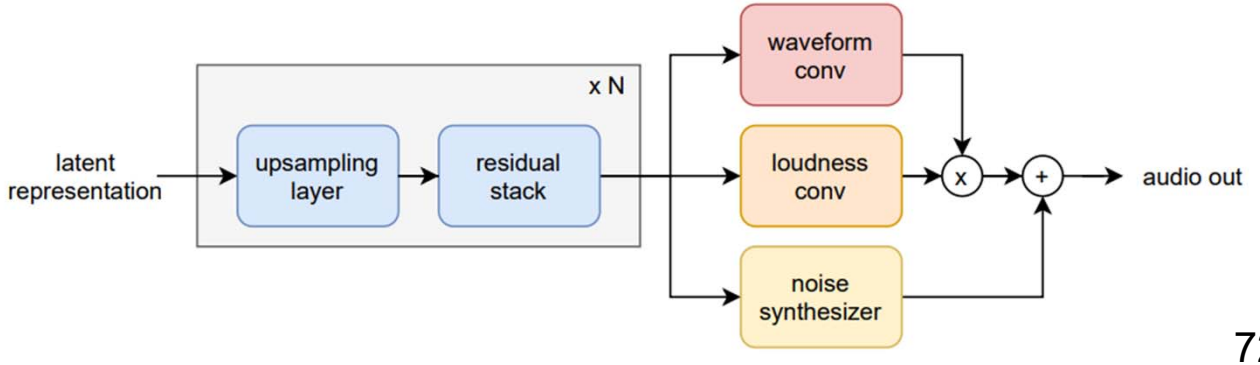
YouTube · Night Shining
2022年9月7日

RAVE: VAE+GAN

- **Encoder**: multiband decomposition and outputs a 128-dim latent



- **Decoder** is a modified version of the generator of **MelGAN**
- **Discriminator**: Exactly the same discriminator as in **MelGAN**



RAVE: Real-time Generation

- Good for creative applications

Table 2: Comparison of the synthesis speed for several models

Model	CPU synthesis	GPU synthesis
NSynth	18 Hz	57 Hz
SING	304 kHz	9.8 MHz
RAVE (Ours) w/o multiband	38 kHz	3.7 MHz
RAVE (Ours)	**985 kHz**	**11.7 MHz**

Realtime usage

This section presents how RAVE can be loaded inside `nn~` in order to be used live with Max/MSP or PureData.

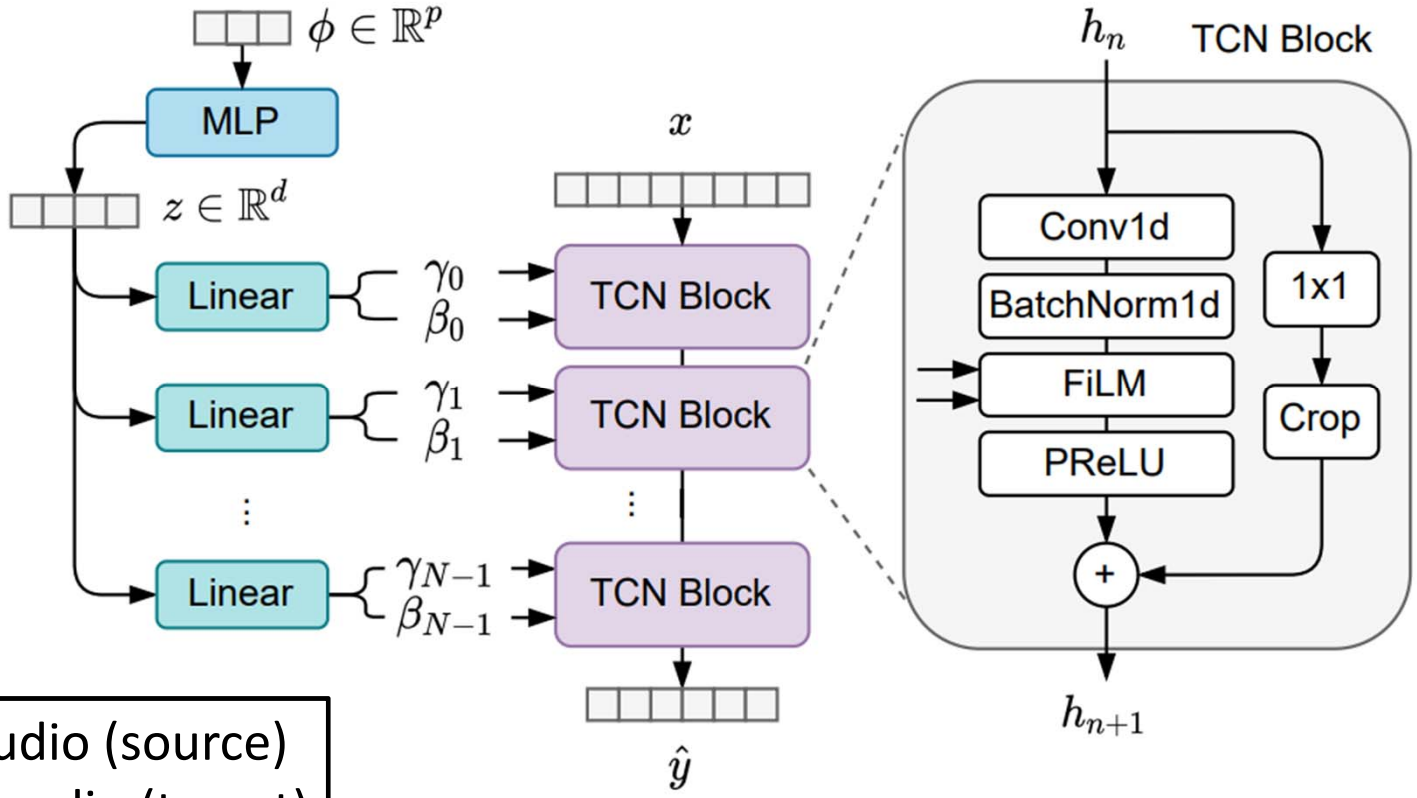
RAVE: Being Actively Updated at IRCAM

(Institut de recherche et coordination acoustique/musique)

https://github.com/acids-ircam/RAVE

Type	Name	Description
Architecture	v1	Original continuous model
	v2	Improved continuous model (faster, higher quality)
	v3	v2 with Snake activation, descript discriminator and Adaptive Instance Normalization for real style transfer
	discrete	Discrete model (similar to SoundStream or EnCodec)
	onnx	Noiseless v1 configuration for onnx usage
	raspberry	Lightweight configuration compatible with realtime RaspberryPi 4 inference
Regularization (v2 only)	default	Variational Auto Encoder objective (ELBO)
	wasserstein	Wasserstein Auto Encoder objective (MMD)
	spherical	Spherical Auto Encoder objective
Discriminator	spectral_discriminator	Use the MultiScale discriminator from EnCodec.
Others	causal	Use causal convolutions
	noise	Enable noise synthesizer V2

FiLM-TCN (AES'22)

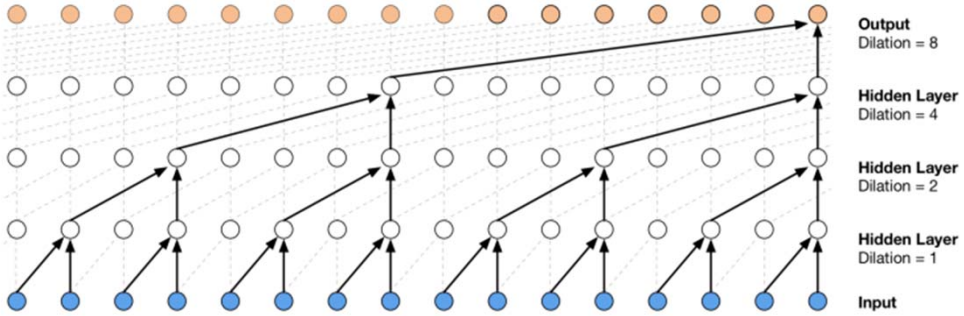


x: input audio (source)
y: output audio (target)
ϕ: condition

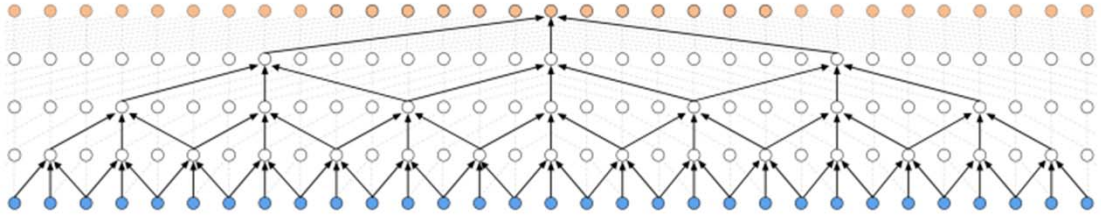
Ref: Steinmetz & Reiss, "Efficient neural networks for real-time modeling," AES 2022

Temporal Convolution Network (TCN)

- *WaveNet*-like sample-level **dilated** convolution
  - Convolutions across sub-sampled input representations



a stack of dilated causal convolutional layers



a stack of dilated non-causal convolutional layers

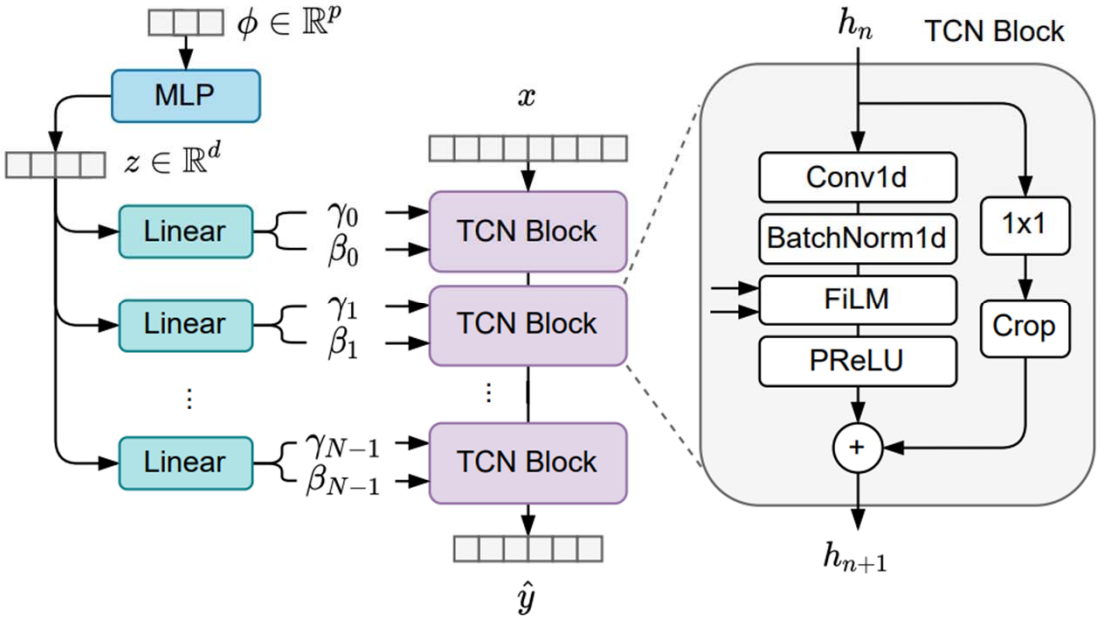
Ref 1: van den Oord et al, “Wavenet: A generative model for raw audio,” ISCA 2016
Ref 2: Lemaire & Holzapfel, “Temporal convolutional networks for speech and music detection in radio broadcast,” ISMIR 2019

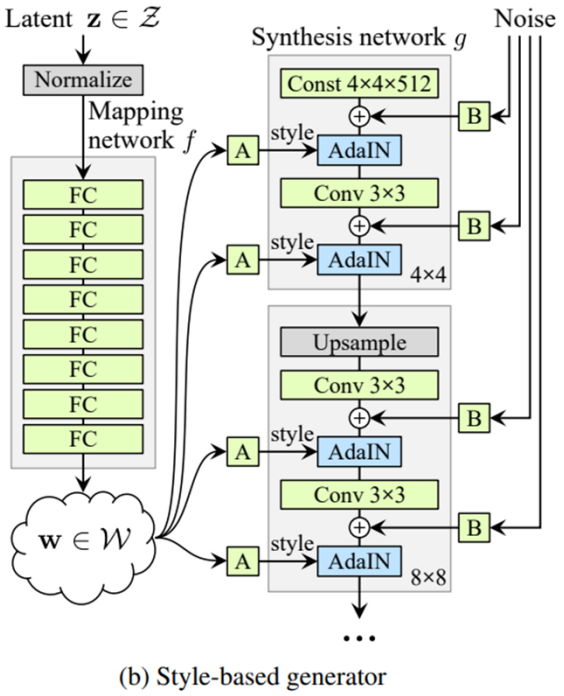
FiLM vs AdaIN

- They are different "**conditioning mechanisms**"

$$F(h_{n,c}, \gamma_{n,c}, \beta_{n,c}) = \gamma_{n,c} h_{n,c} + \beta_{n,c}$$

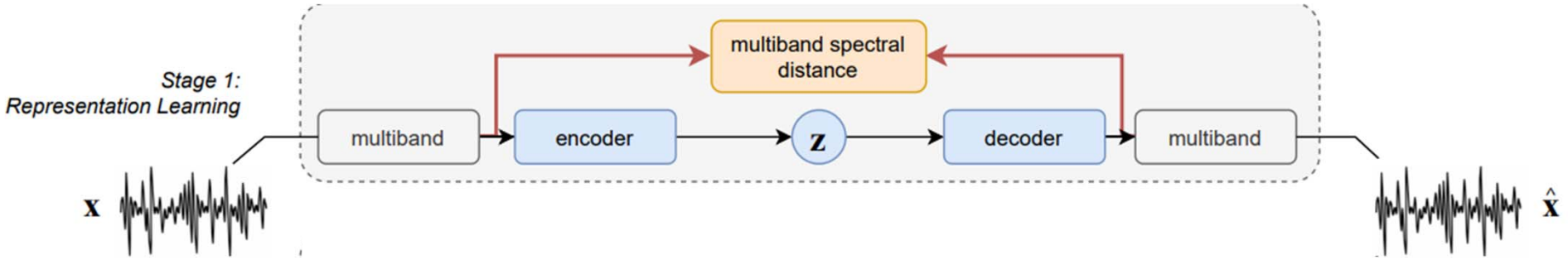
$$\mathrm{AdaIN}(\mathbf{x}_i, \mathbf{y}) = \mathbf{y}_{s,i} \frac{\mathbf{x}_i - \mu(\mathbf{x}_i)}{\sigma(\mathbf{x}_i)} + \mathbf{y}_{b,i},$$

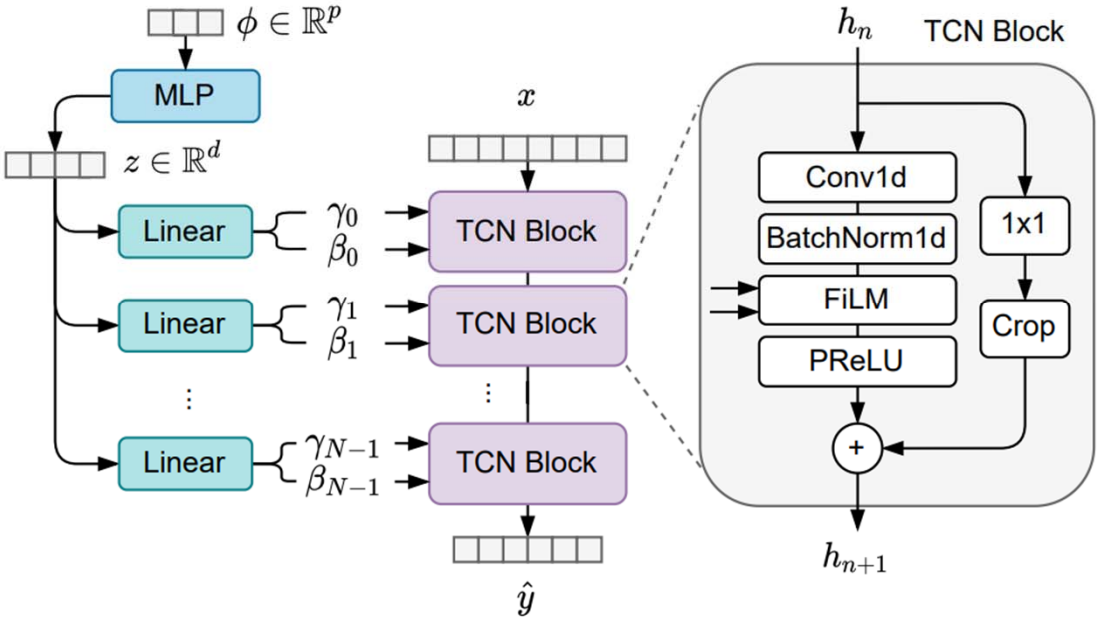




(b) Style-based generator

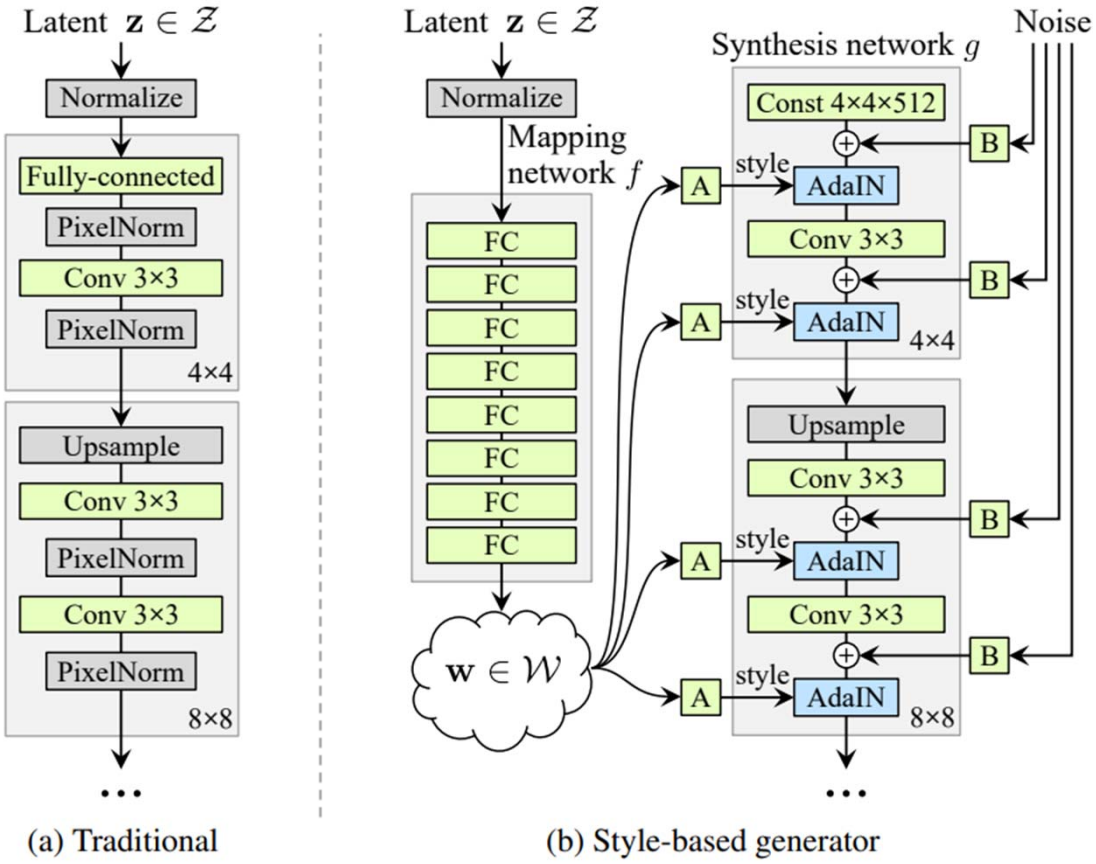
RAVE vs FiLM-TCN



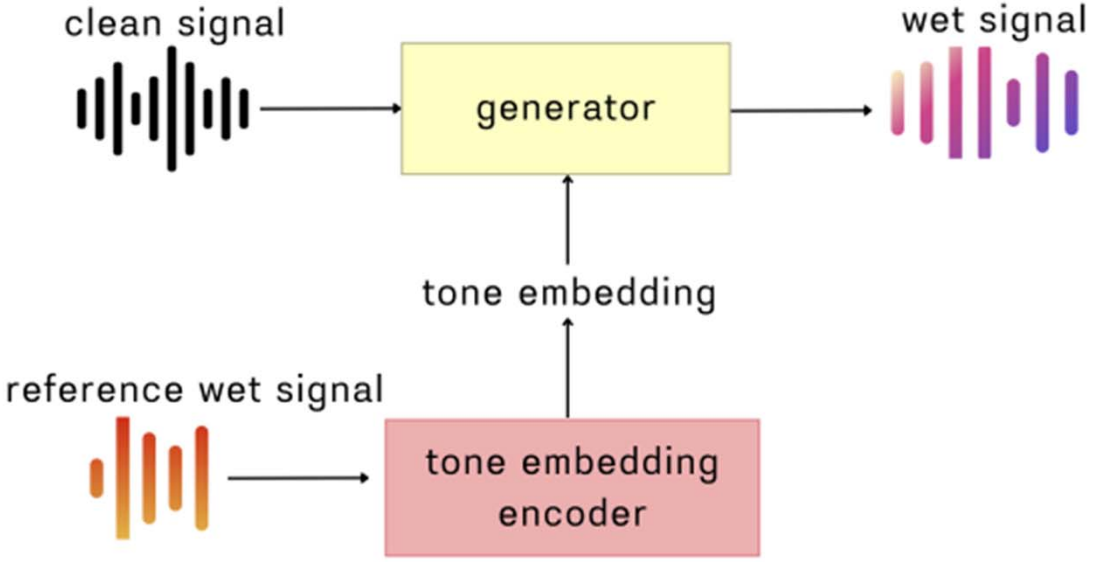


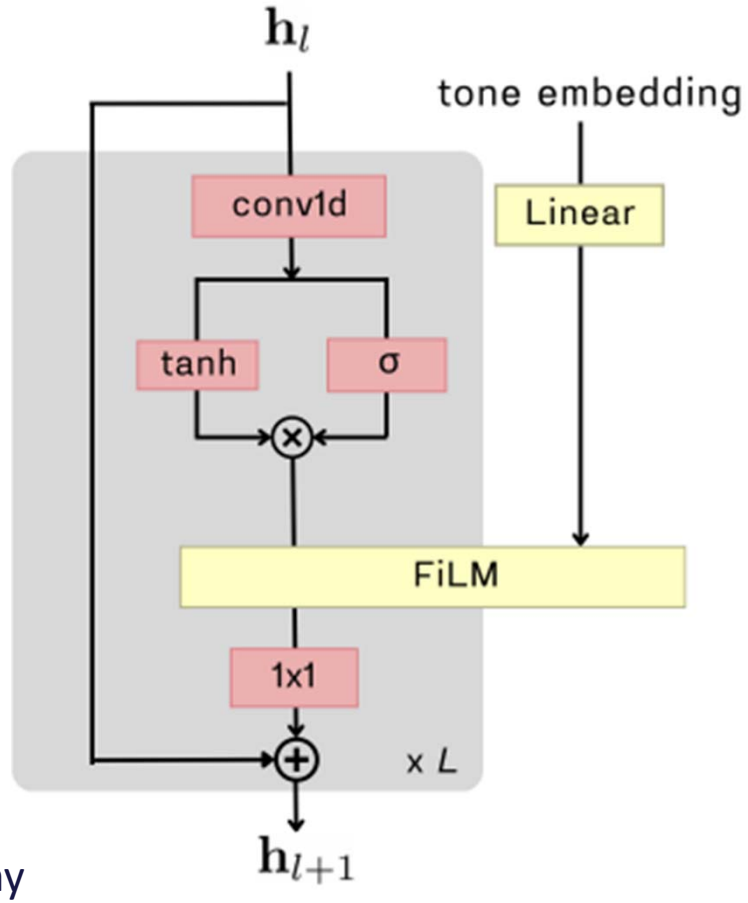
Recap: StyleGAN & StyleGAN2

- Approach 1 (left)
  - Use the latent **z** as the **input** to the decoder
- Approach 2 (right)
  - Use the latent **z** as the **condition** for the decoder



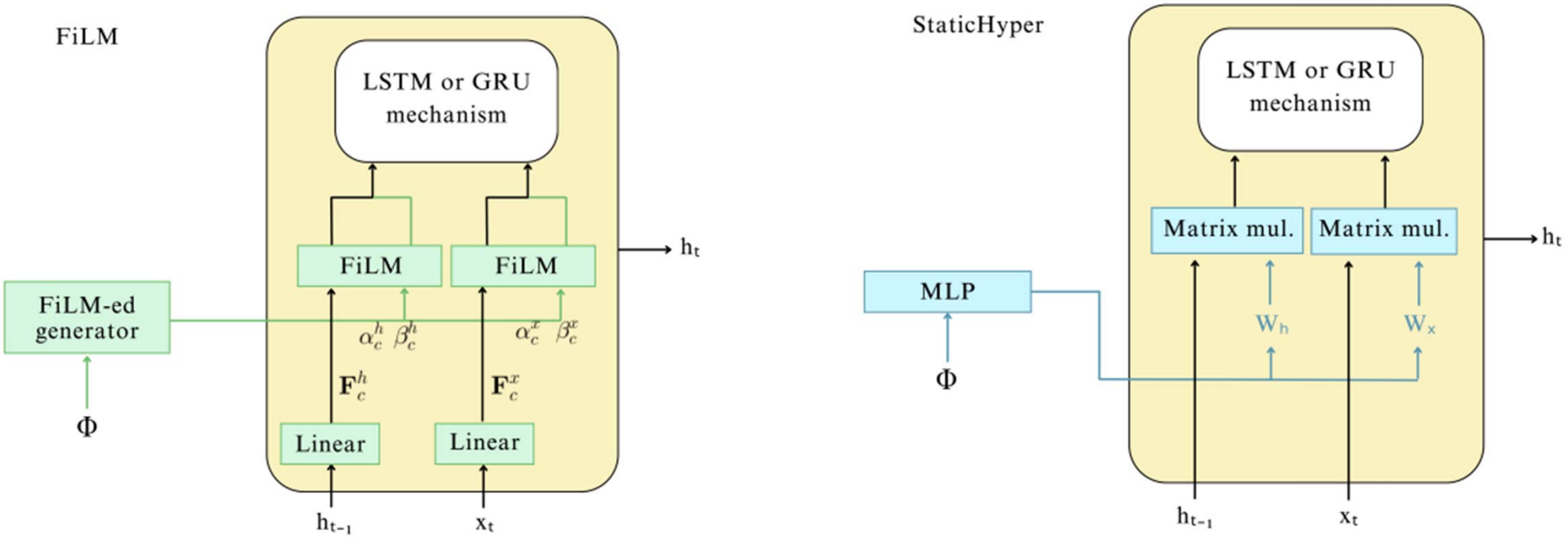
One-to-Many Tone Transfer (ISMIR'24) (from our lab)





Ref: Chen et al, "Towards zero-shot amplifier modeling: One-to-many amplifier modeling via tone embedding control," ISMIR 2024.

Hyper-RNN (DAFx'24) (from our lab)



Ref: Yeh et al, "Hyper recurrent neural network: Condition mechanisms for black-box audio effect modeling," DAFx 2024

Conditional Mechanisms for Audio Generation

Model	Condition	OD-3 (Overdrive)						Params
		L1	STFT	LUFS	CF	RMS	Transient	
LSTM	Concat	0.123	1.901	0.524	2.982	1.259	25.997	4769
	FiLM	0.145	1.057	0.248	1.834	0.552	20.322	22561
	StaticHyper	0.146	1.031	0.221	2.051	0.451	20.106	40449
	DynamicHyper	0.149	0.695	0.199	2.431	0.402	12.968	21857
GRU	Concat	0.120	1.933	0.455	2.932	1.096	27.338	3585
	FiLM	**0.011***	**0.536**†	0.176	**0.676***	0.401	12.504	17217
	StaticHyper	0.017	0.698	0.165	1.650	**0.318**†	**12.347**†	30369
	DynamicHyper	0.150	**0.428***	**0.075***	0.883	**0.153***	**11.308***	20289
TCN	Concat	0.033	0.928	0.305	1.177	0.671	27.634	21769
	FiLM	0.044	0.698	0.338	0.894	0.842	33.678	29849
GCN	Concat	**0.013**†	0.792	0.202	**0.776**†	0.447	19.103	19824
	FiLM	0.149	0.672	**0.141**†	1.200	0.276	24.474	32368