2025 edition

Deep Learning for Music Analysis and Generation

# Synthesis & MIDI-to-Audio Generation

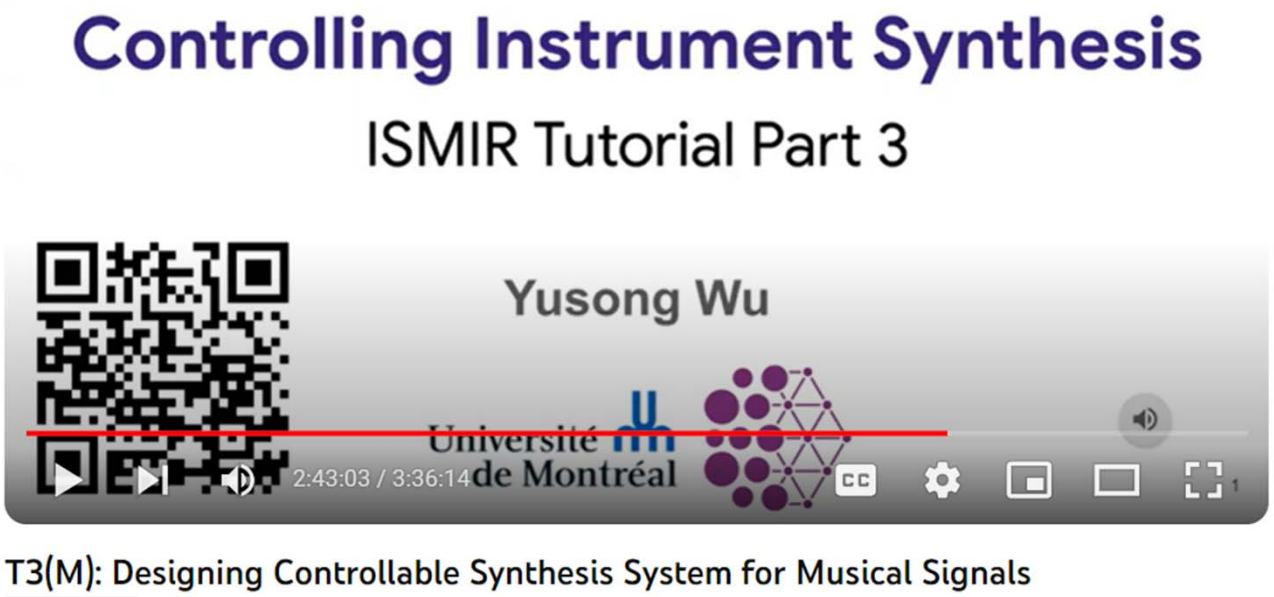
(midi → audio; X → audio)

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

# Reference: ISMIR 2022 Tutorial

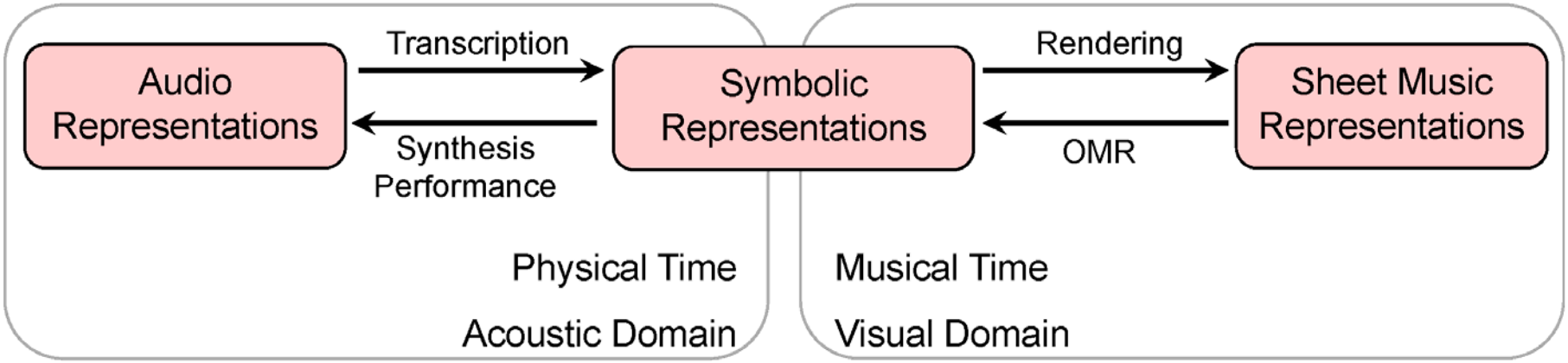
https://github.com/lukewys/ISMIR2022-tutorial

https://youtu.be/7U-zDL5con8?si=HcD7YDN66YPlyGCN&t=9783



# Data Representation of Music

(Figure source: https://www.mdpi.com/2624-6120/2/2/18)




- **Musical audio** (waveforms, spectrograms)
  - what we "hear"
  - involve sounds
- **Symbolic representations** (e.g., MIDI, MusicXML)
  - the way music can be represented in a Digital Audio Workstation (DAW)
  - still relevant today as it's *interpretable* and *editable*
- **Sheet music** (visual representations of a musical score)
  - what musicians "compose"

# MIDI-to-Audio Generation

- **Note-by-note** (synthesizers)
  - Take pitch and duration as condition
  - Generate single notes

- MIDI **sequence** to audio sequence
  - More general than single-note generation

- **Text-to-MIDI-to-music** generation
  - Little work has been done

# Outline

- **DSP-based music synthesizers**
- DL-based generation of single notes
- DL-based generation of note sequences
- DL-based generation of loops

# 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

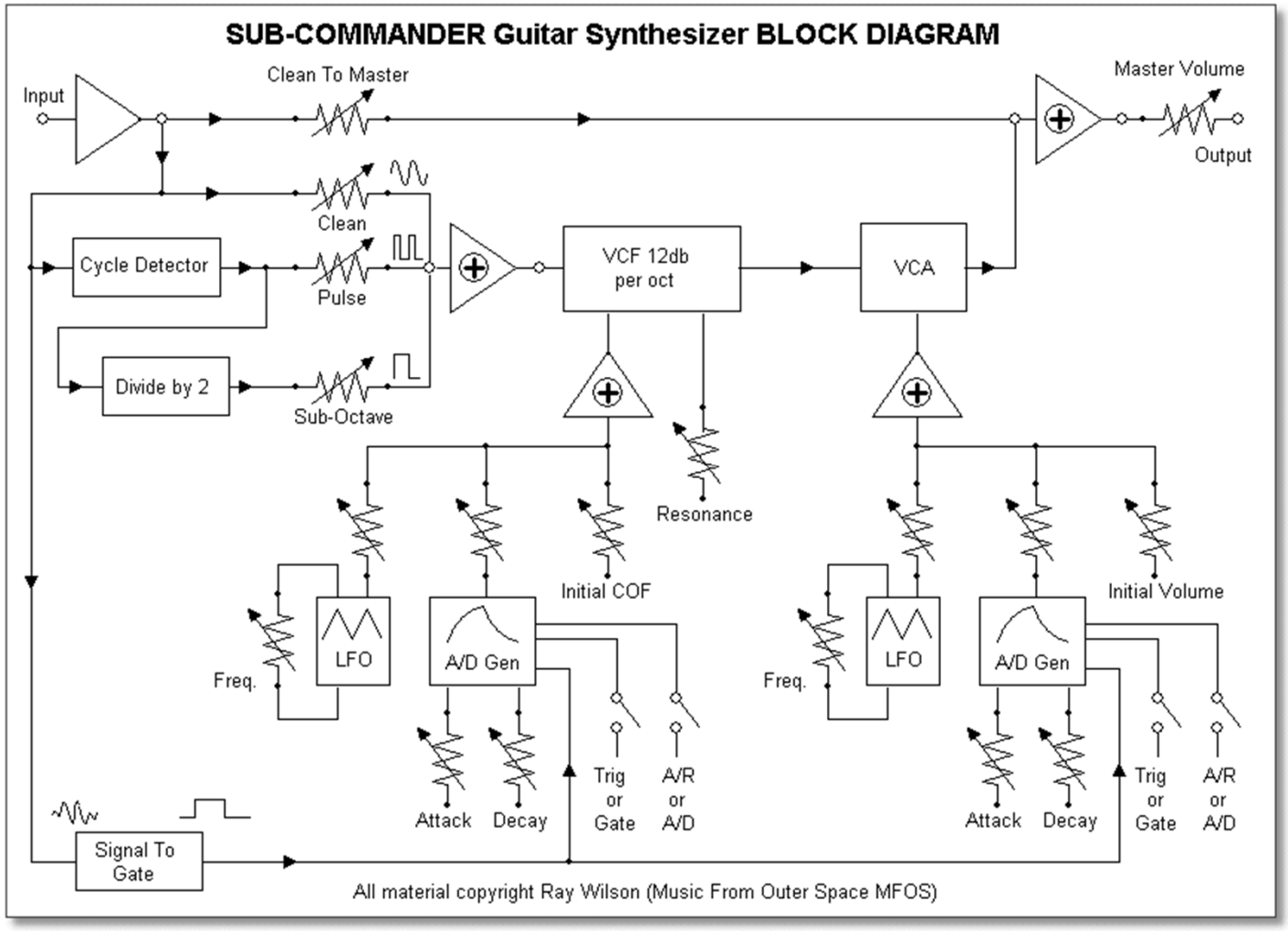



Figure from:
http://musicfromouterspace.com/analogsynth_new/GUITARSYNTHAUG2007/GUITARSYNTHAUG2007.html

# 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)

# 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**

# Sample-based Synthesis: Ample Sound

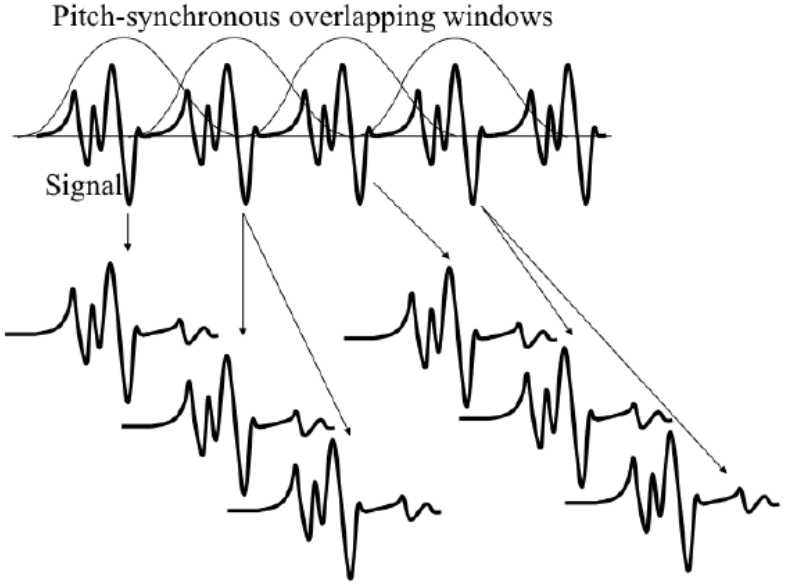
https://x.com/Reo5419233/status/1853347613116072218

# 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

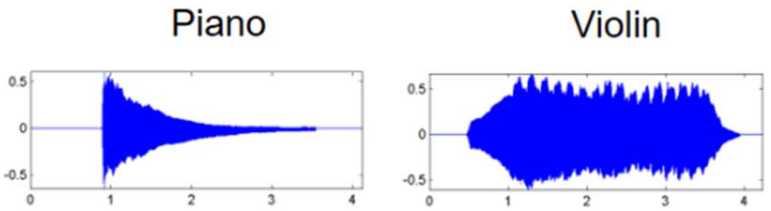
- Overlap-Add (OLA)
  - OLA is the most simple TSM algorithm that changes the length of the signal through modifying the hop size between analysis frame and synthesis frame.
- Pitch Synchronous Overlap-Add (TD-PSOLA)
  - TD-PSOLA is the algorithm that analyzes the orignal waveforms to create pitich-synchronous 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

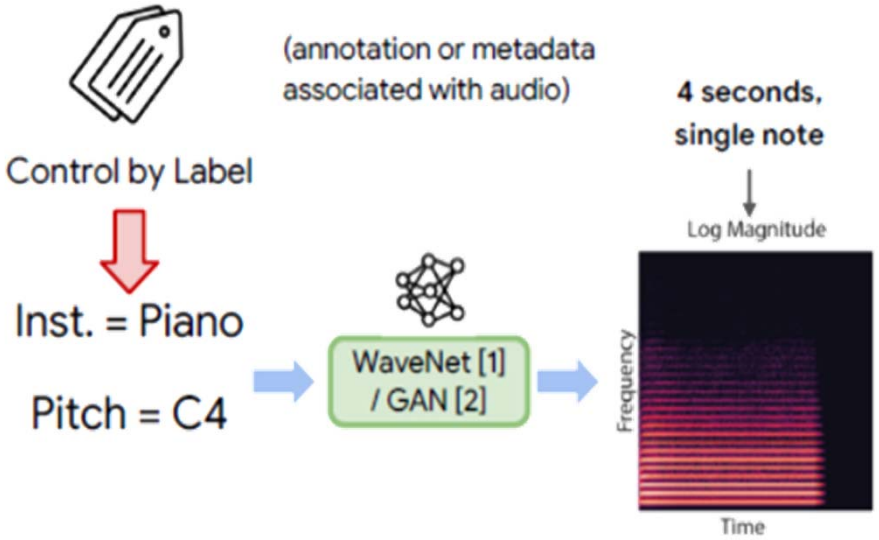
- DSP-based music synthesizers
- **DL-based generation of single notes**
  - WaveNet Autoencoder
  - GANSynth
  - InstrumentGen
- DL-based generation of note sequences
- DL-based generation of loops

# Neural Generation of Single Notes

- "**MIDI pitch → audio**"
- Condition on pitch and instrument
- Modeling mainly **timbre**




- Can we learn such a mapping in a *data-driven* way via DL?




Ref: https://github.com/lukewys/ISMIR2022-tutorial

# Autoregressive Models for Unconditional Audio Generation

- Generate **raw waveforms**
- **Unconditional** generation (not music synthesis yet)
  - **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:

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

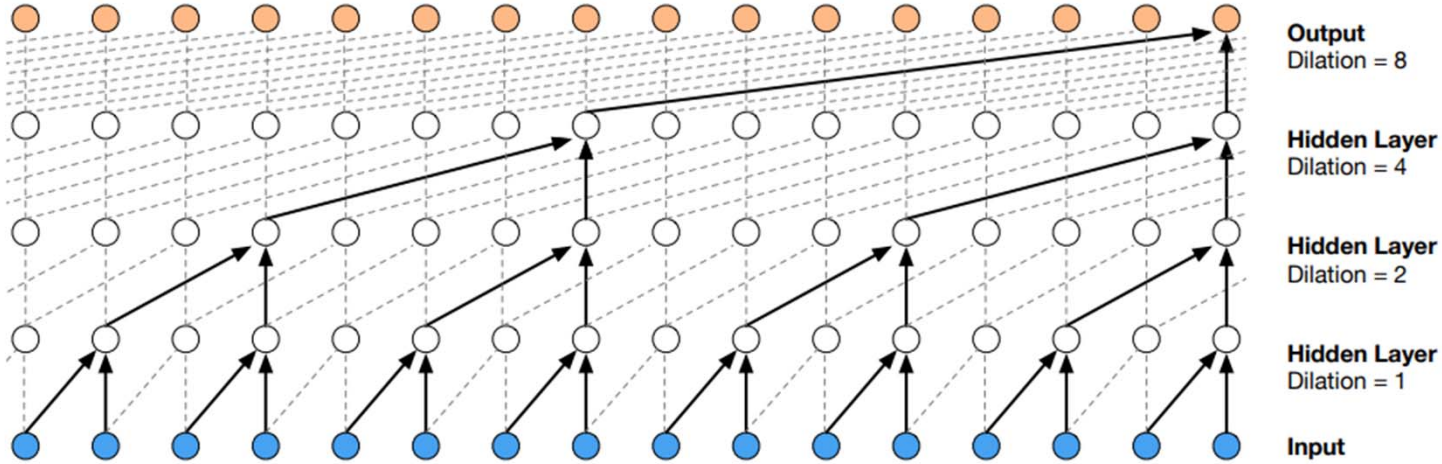
Each audio sample $x_t$ is therefore conditioned on the samples at all previous timesteps.

- **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"

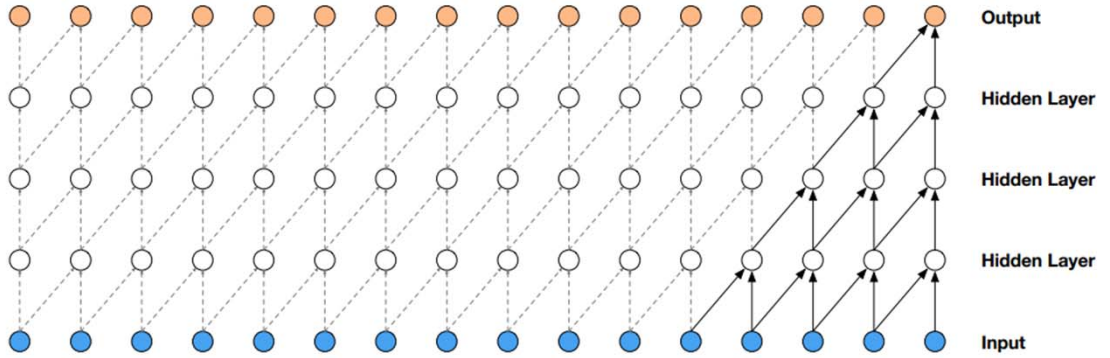
Ref: van den Oord et al, "WaveNet: A generative model for raw audio," arXiv 2016

# 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



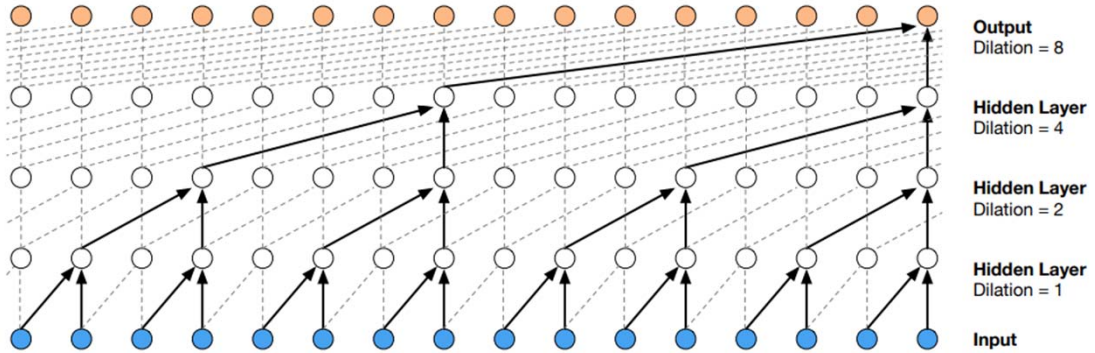
# WaveNet: Output Design

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 $\mu$-law companding transformation (ITU-T, 1988) to the data, and then quantize it to 256 possible values:

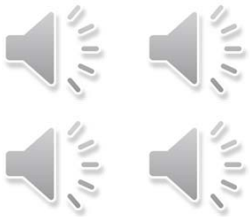
$$f(x_t) = \text{sign}(x_t) \frac{\ln(1 + \mu|x_t|)}{\ln(1 + \mu)},$$

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: 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)

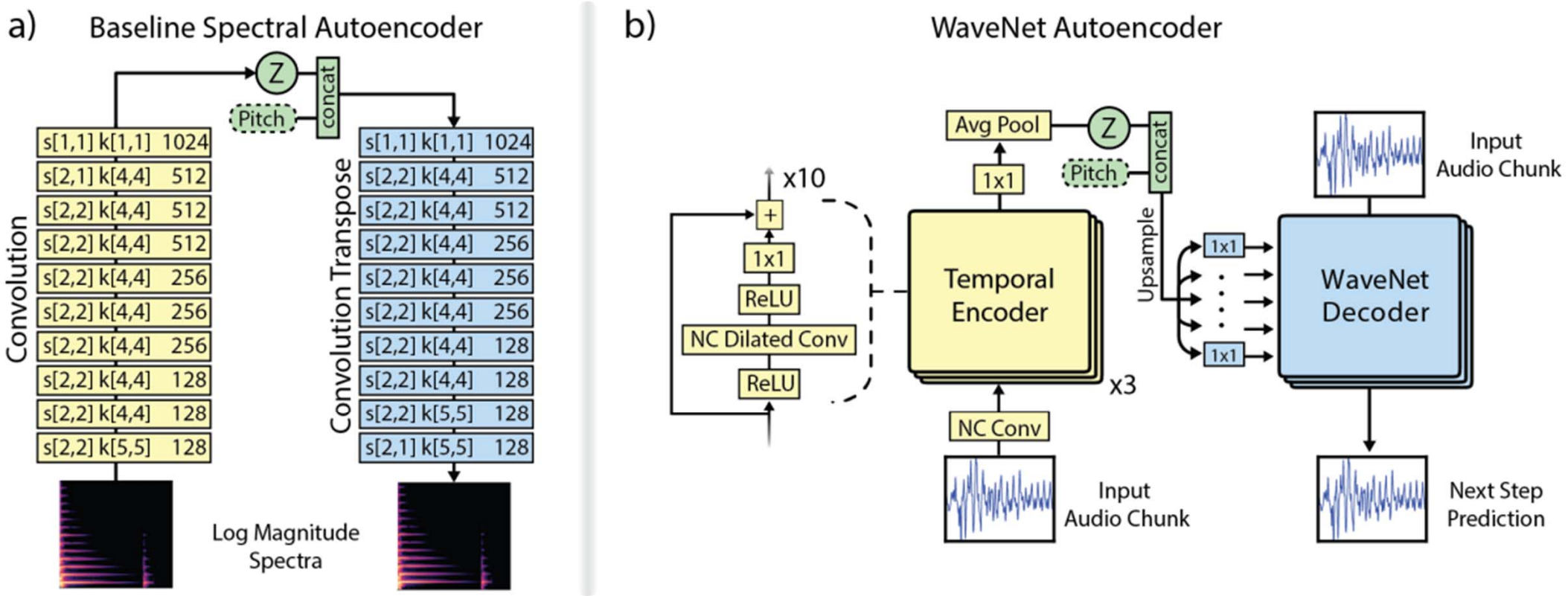
- Extension of WaveNet for "**MIDI pitch → audio**"
  - Encoder-decoder; conditional generation
- **WaveNet Autoencoder**
  - WaveNet-like **encoder** that infers hidden embeddings $Z = f(x)$ distributed in time
  - WaveNet **decoder** that uses $Z$ to reconstruct the original audio autoregressively

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

  - 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

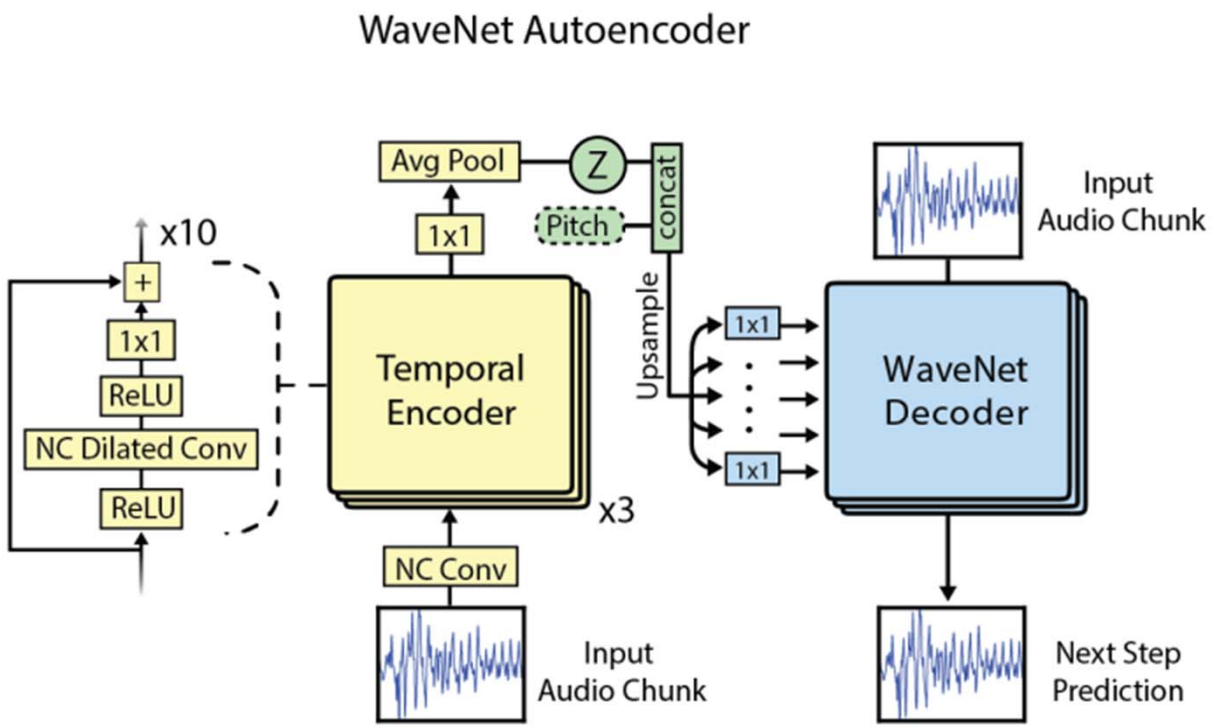
# WaveNet AutoEncoder




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

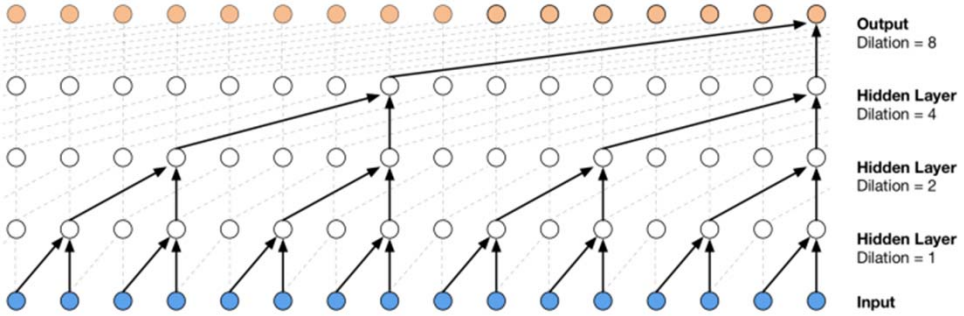
# WaveNet AutoEncoder

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

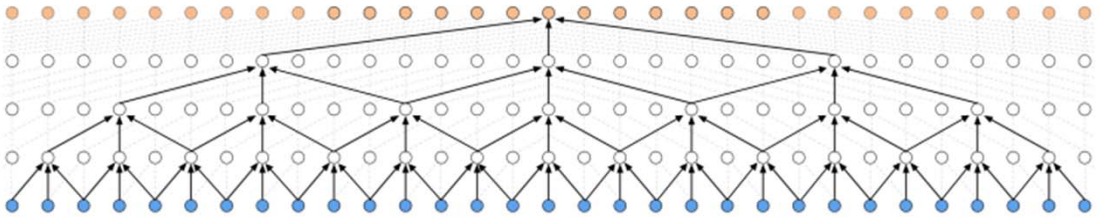


# WaveNet AutoEncoder

- *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*

# 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

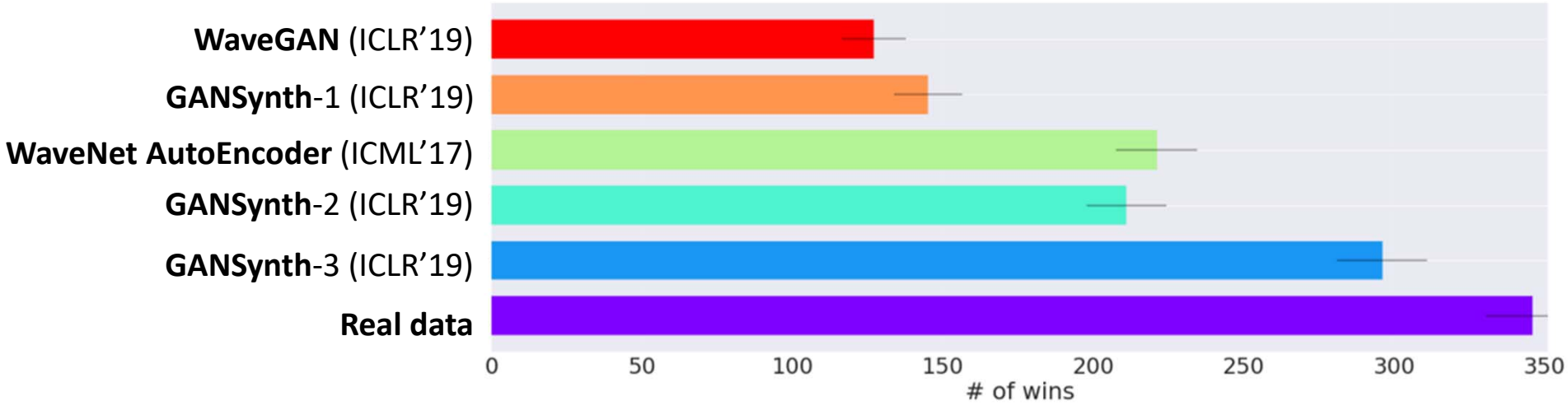



Figure 2: Number of wins on pair-wise comparison across different output representations and baselines. Ablation comparing highest performing models of each type. Higher scores represent better

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

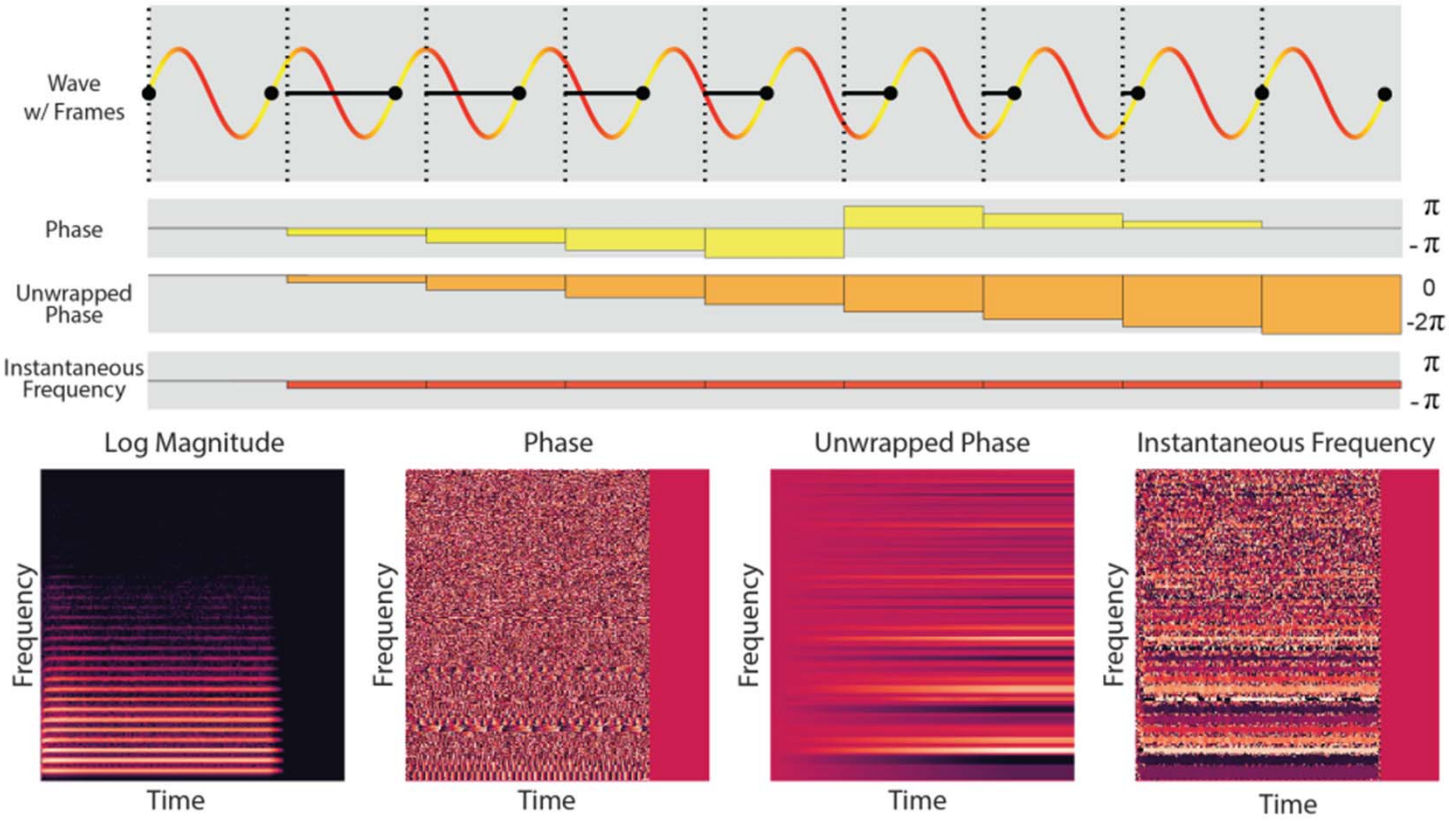
# GANSynth (ICLR'19)

- **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

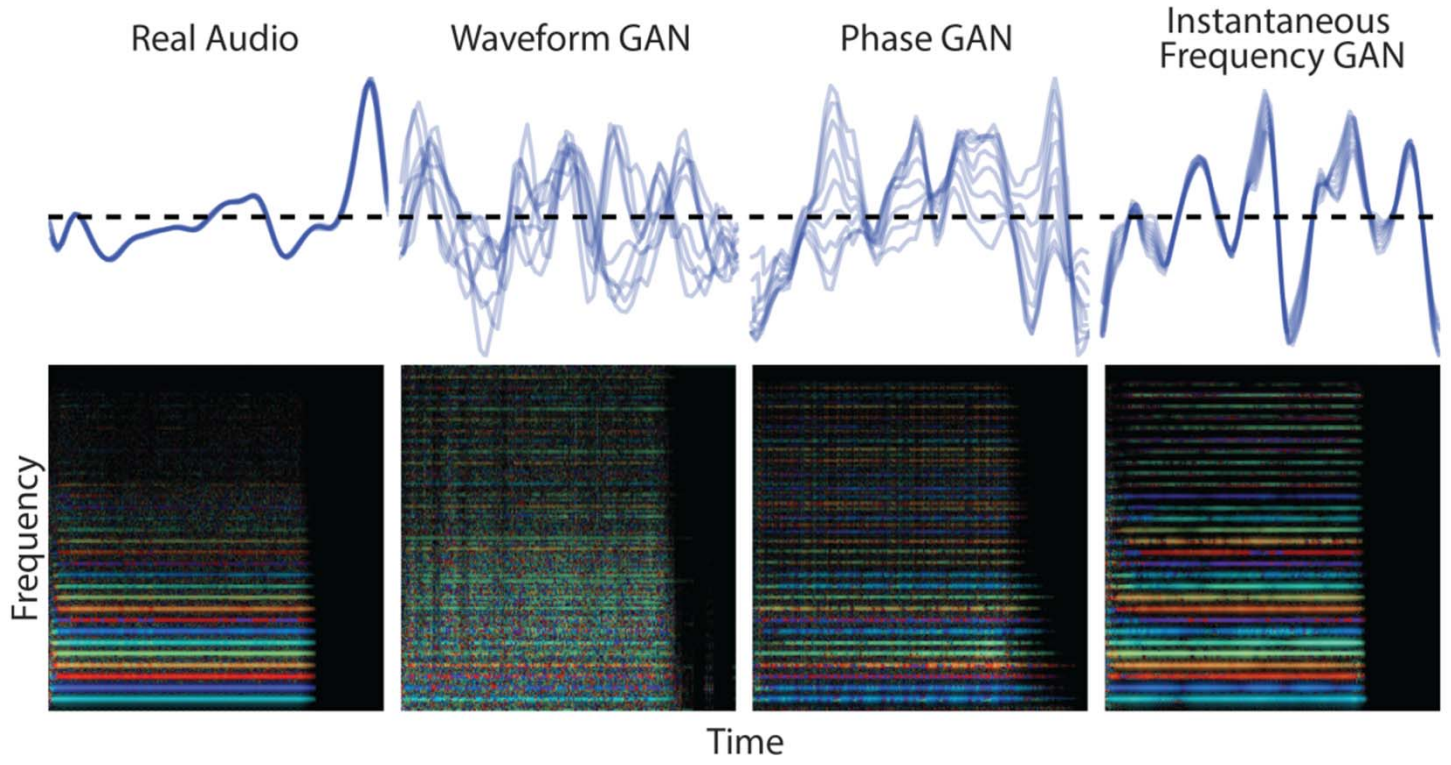
# GANSynth

- Actually model the **instantaneous radial frequency (IF)**, not the phase
  - IF: the **derivative of phase** (unwrapped over the $2\pi$ boundary)



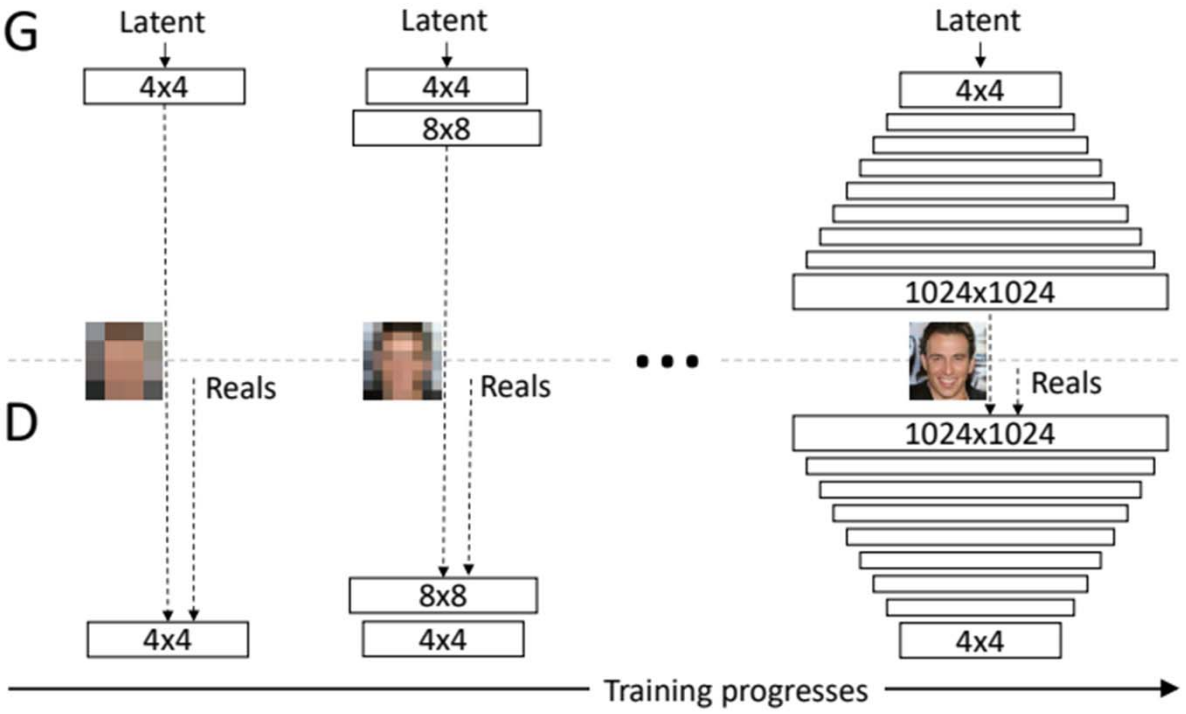
# GANSynth

- Actually model the **instantaneous radial frequency (IF)**, not the phase
  - IF: the **derivative of phase** (unwrapped over the $2\pi$ boundary)



# GANSynth

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

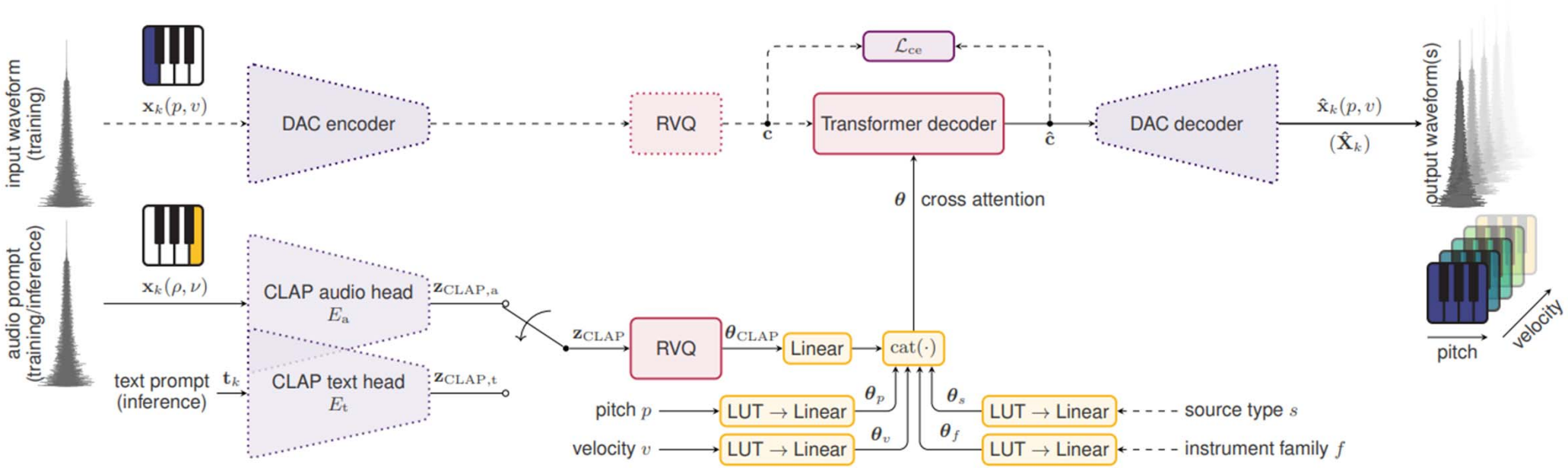



Ref: Karras et al, "Progressive growing of GANs for improved quality, stability, and variation," ICLR 2018

# InstrumentGen (ISMIR'24)

https://instrumentgen.netlify.app/

- Take **pitch**, **velocity**, and **text** as input




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

# InstrumentGen (ISMIR'24)

https://instrumentgen.netlify.app/

## Advanced Descriptive Prompts/Limitations

To showcase the capabilities and current limitations of the model in response to advanced descriptive prompts, we present the following examples.

Select MIDI velocity: 100

| Prompt | Generated audio (2 octaves) |
|---|---|
| A string ensemble characterized by high harmonics, light bowing, and sparse vibrato, yielding an airy, floating tonal quality. | 0:00 / 1:40 |
| A bass synth with a distorted sawtooth waveform and high resonance, delivering a gritty, aggressive sonic texture. | 0:00 / 1:40 |
| Staccato piano notes augmented with a synthetic overlay and digital delay, producing a crisp, rhythmically precise tonal effect. | 0:00 / 1:40 |

## Sample-to-Instrument

Although it is not the central focus of our paper, our system inherently accommodates a sample-to-instrument functionality, whereby a musical instrument can be generated from a single audio reference as input. We notionally demonstrate this with an out-of-domain audio sample used as the prompt.

| Prompt | Generated audio (2 octaves) |
|---|---|
| 0:00 / 0:01 | 0:00 / 1:40 |

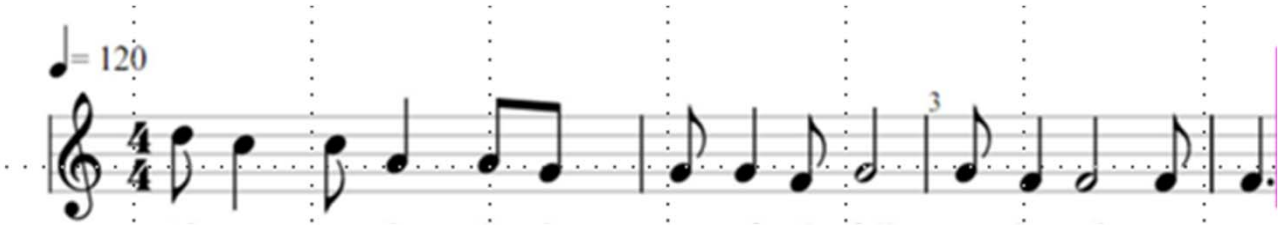
Ref: Nercessian et al, "Generating sample-based musical instruments using neural audio codec language models," ISMIR 2024

# Outline

- DSP-based music synthesizers
- DL-based generation of single notes
- **DL-based generation of note sequences**
  - Deep Performer
  - MIDI-DDSP
- DL-based generation of loops

# Phrase Generation: Sequence of MIDI Pitches → Audio

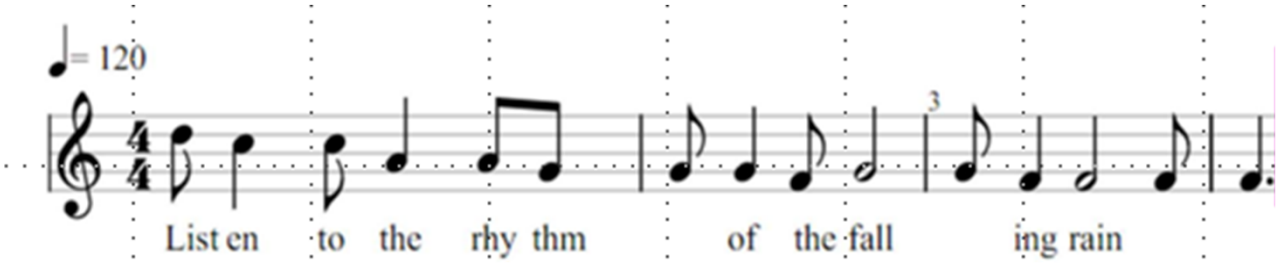
- Note-to-audio
  - "Single MIDI pitch → audio"
  - Like a **synthesizer**, or an "**instrument**"
  - Let the human musician play it

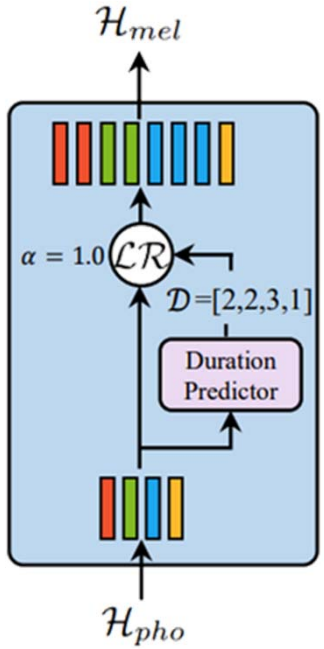



- Score-to-audio
  - "Sequence of MIDI pitches → audio"
  - More autonomous
  - Generate timbre and **expressive performance**
    - Need to account for **note-note transition**, position of a note in a phrase, etc

# Monophonic Phrase Generation

- We have actually learned this before, in **singing voice synthesis** (SVS), for **monophonic** note sequence
  - Conversion between **symbolic timing** to **physical timing** via mechanisms such as duration predictor
  - Use neural networks to learn note-note transitions, and the way to sing each note depending on their location in a phrase

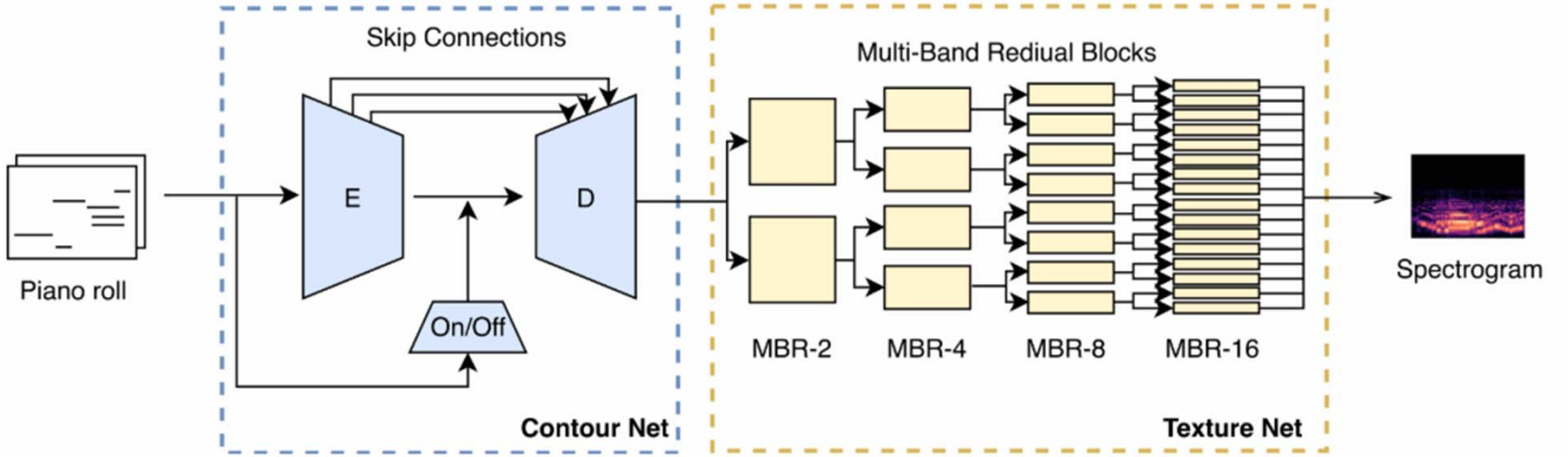



- How about the **polyphonic** case?

# Polyphonic Phrase Generation

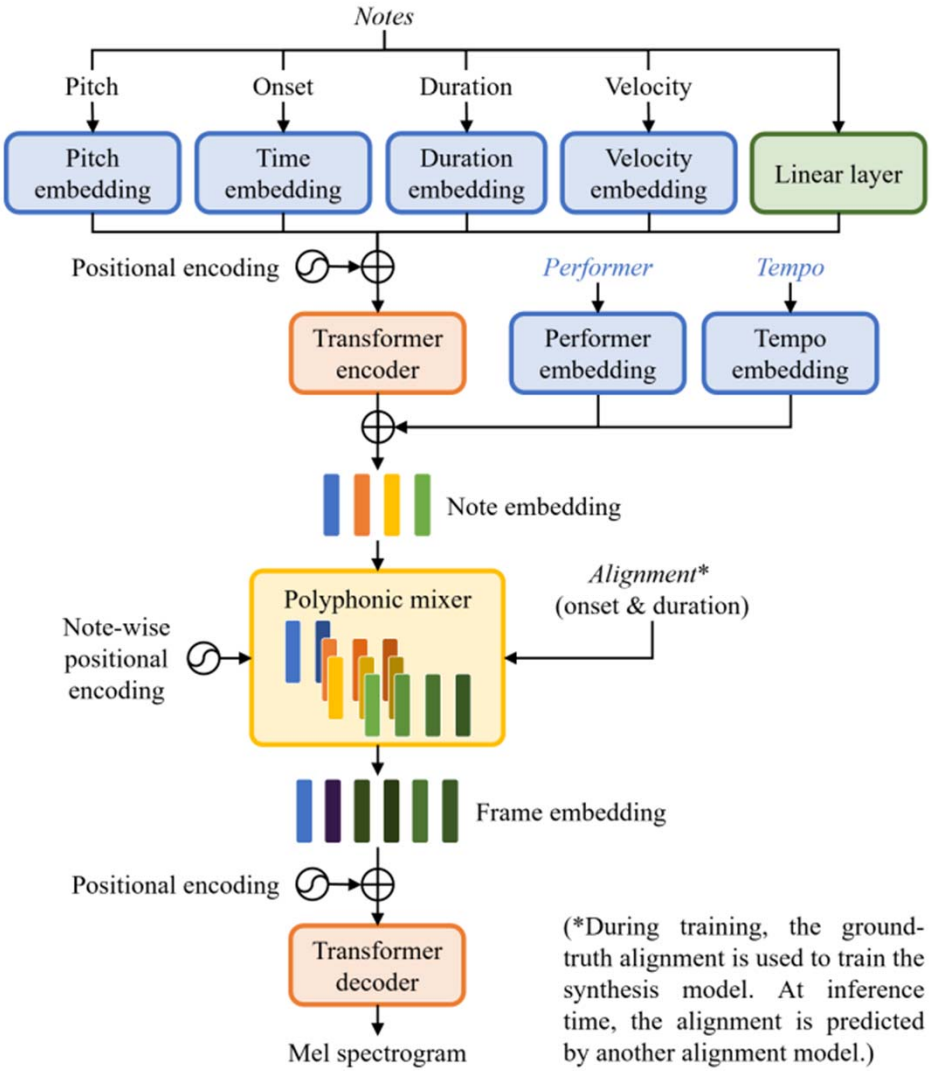
- **PerformanceNet** (AAAI'18)
  - **CNN** approach converting pianorolls (MIDI performances) into Mel spectrograms
  - Limits: assume the input and output are of the same length (no need to care about symbolic-to-physical timing conversion)




Ref: Wang & Yang, "PerformanceNet: Score-to-audio music generation with multi-band convolutional residual network," AAAI 2018

# Polyphonic Phrase Generation

- **Deep Performer** (ICASSP'22)
  - **Transformer**-based approach being inspired by **SVS** models
  - Use an **alignment** module for symbolic-to-physical timing conversion, by predicting the **onset** and **duration** of each note
    - NOTE: onset prediction is not needed for monophonic cases such as SVS
  - Use a **"polyphonic mixer"** to handle polyphonic inputs
    - sum up the note embeddings for the same frame according to onsets, durations




Ref: Dong et al, "Deep Performer: Score-to-audio music performance synthesis," ICASSP 2022

# Deep Performer

https://hermandong.com/deepperformer/

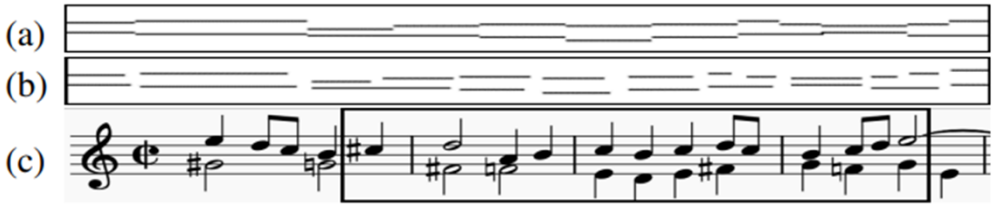



**Fig. 4**: Examples of the alignments predicted by (a) the constant-tempo baseline model and (b) Deep Performer, our proposed model. (c) shows the input score.

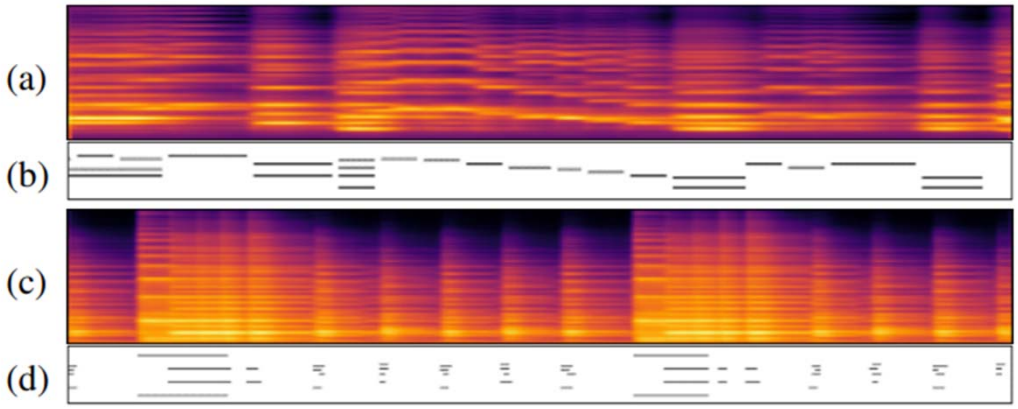


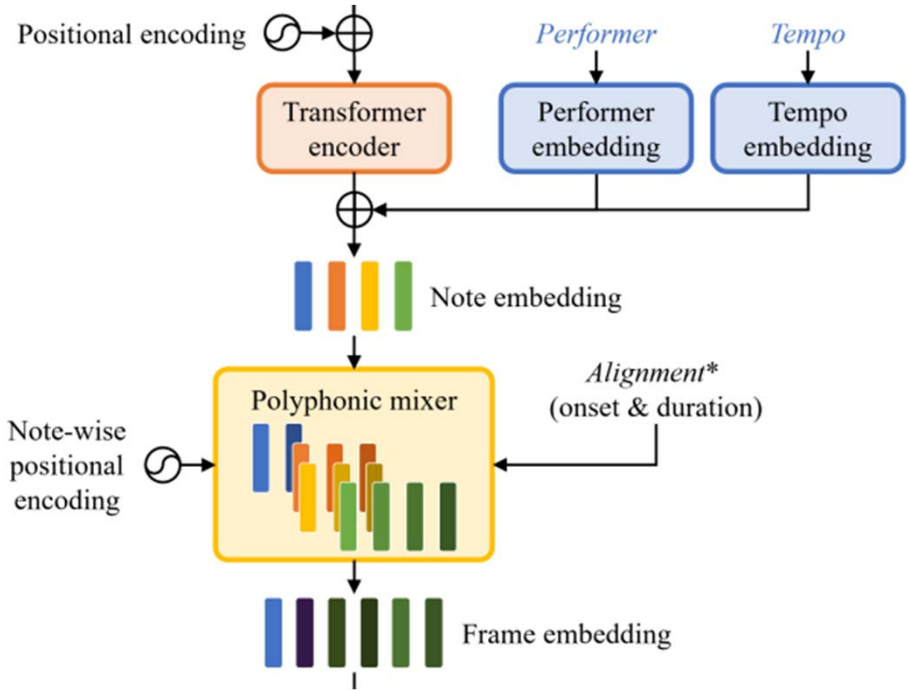

**Fig. 5**: Examples of the mel spectrograms, in log scale, synthesized by our proposed model for (a) violin and (c) piano. (b) and (d) show the input scores for (a) and (c), respectively.

- Future work mentioned
  - Use articulation marks and ornaments on scores to model playing techniques
  - disentangle timbre from room acoustics
  - incorporate adversarial losses

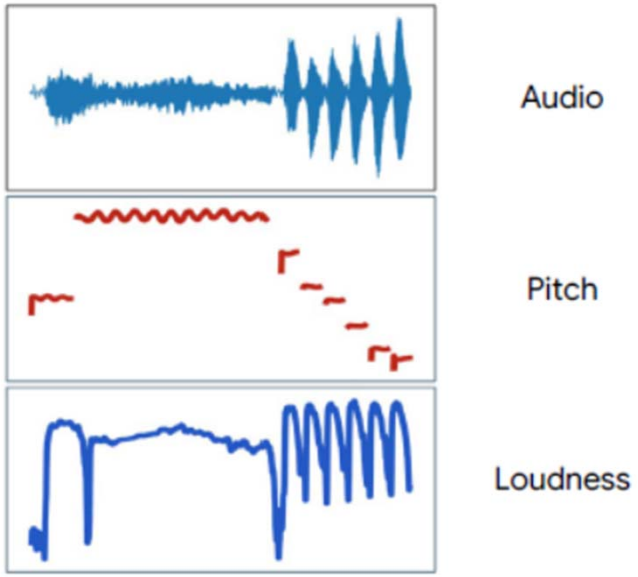
Ref: Dong et al, "Deep Performer: Score-to-audio music performance synthesis," ICASSP 2022

# Controllability of MIDI-to-Audio Generation

- Deep Performer offers control of **onset** and **duration**

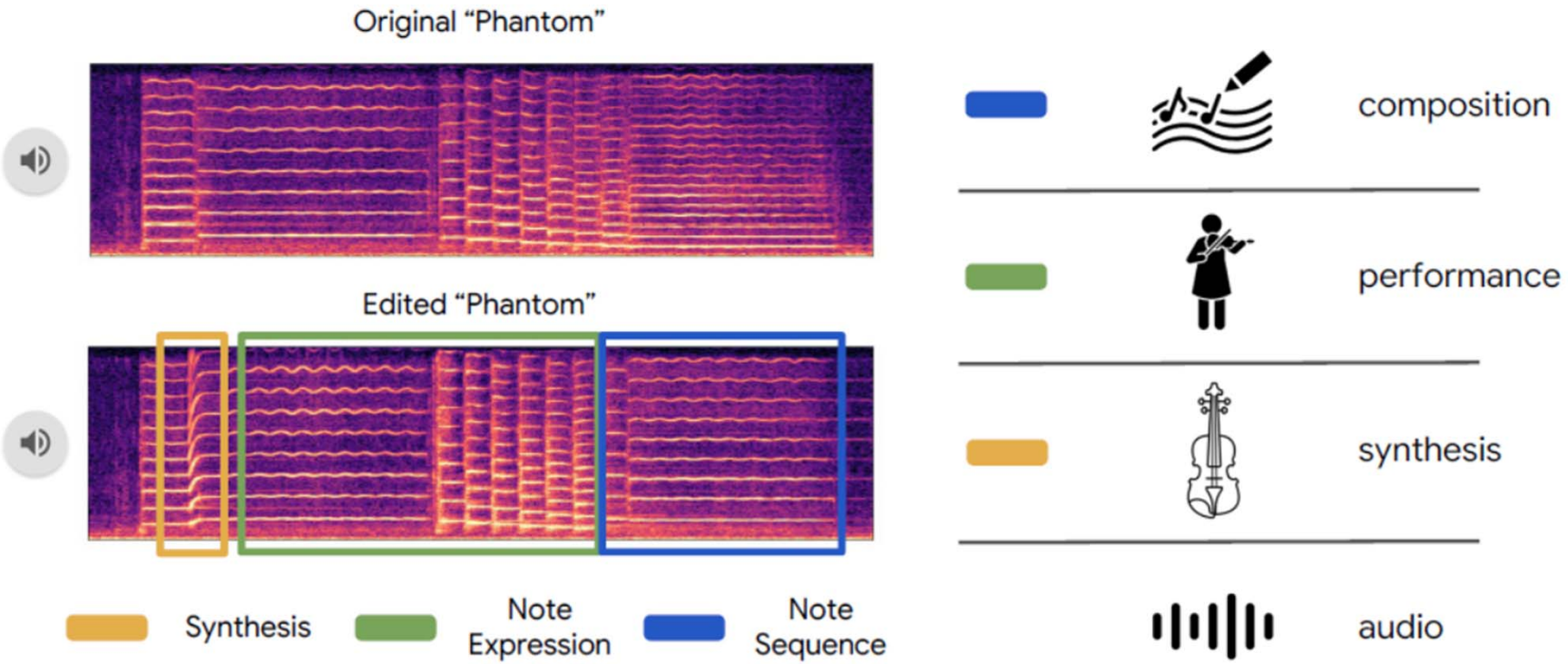



- But there are other **low-level, time-varying** quantities that we may want to control
- E.g., **F0 contour**, **loudness**



# MIDI-DDSP: Controlling Instrument Synthesis (ICLR'22)

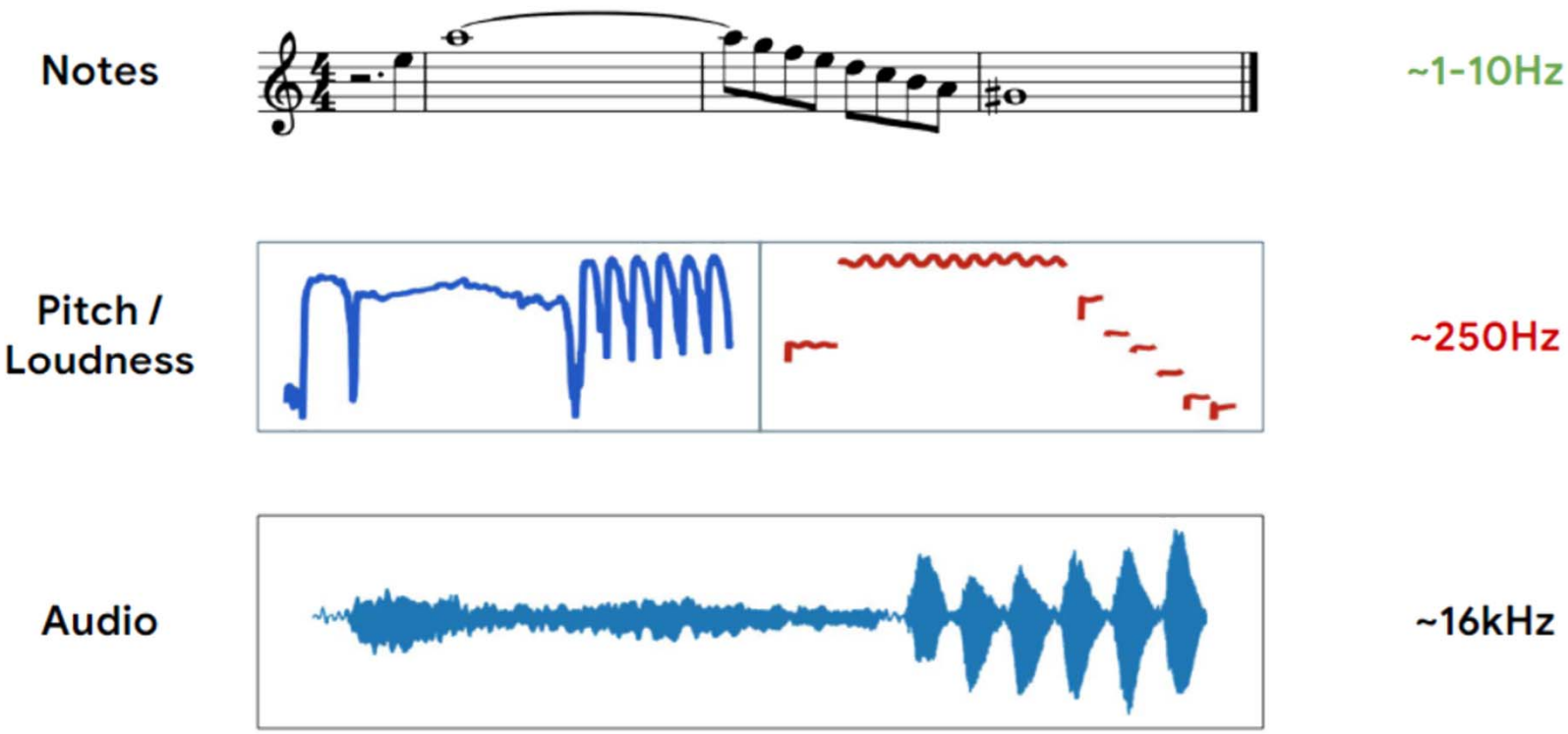
https://youtu.be/7U-zDL5con8?si=HcD7YDN66YPlyGCN&t=9783




Ref: Wu et al, "MIDI-DDSP: Detailed control of musical performance via hierarchical modeling," ICLR 2022

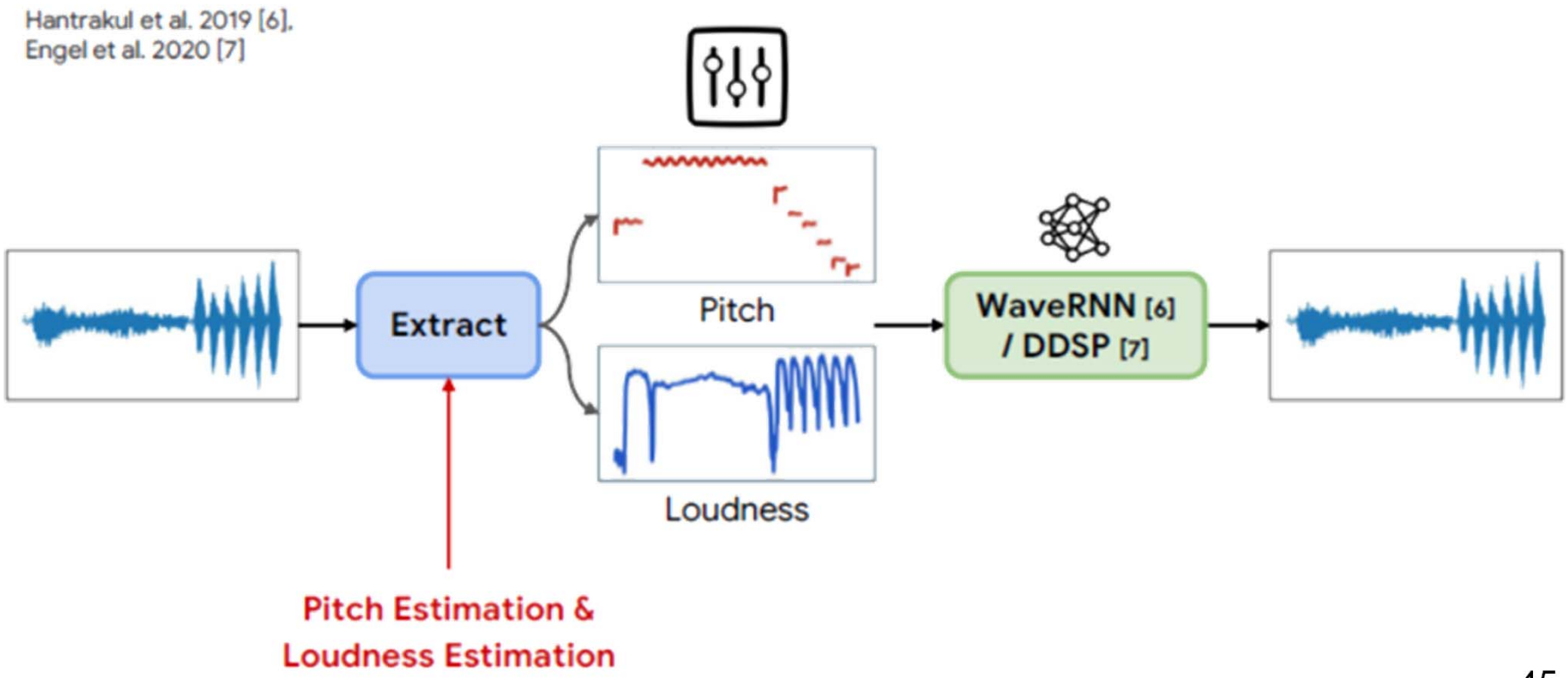
# Low-level Quantities

https://github.com/lukewys/ISMIR2022-tutorial



# Extract the Label: Pitch and Loudness

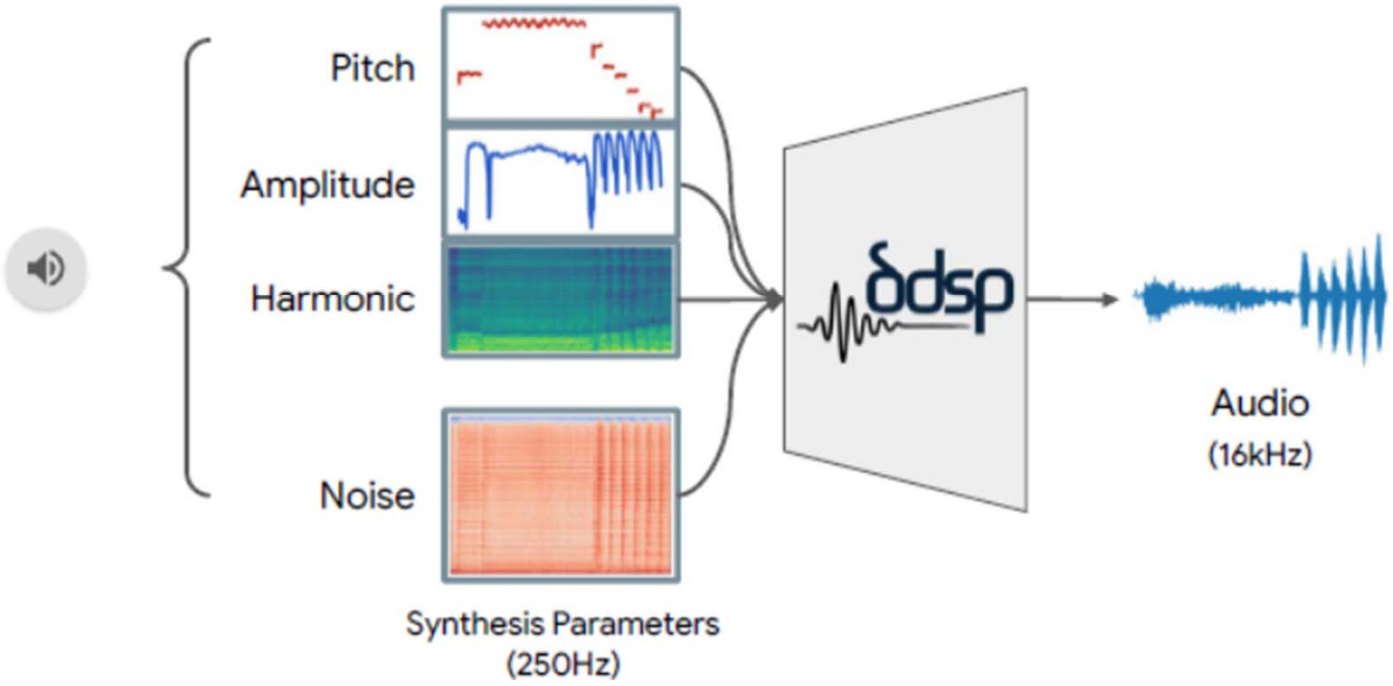
https://github.com/lukewys/ISMIR2022-tutorial



# Learn to Extract Synthesis Parameters: DDSP

https://github.com/lukewys/ISMIR2022-tutorial

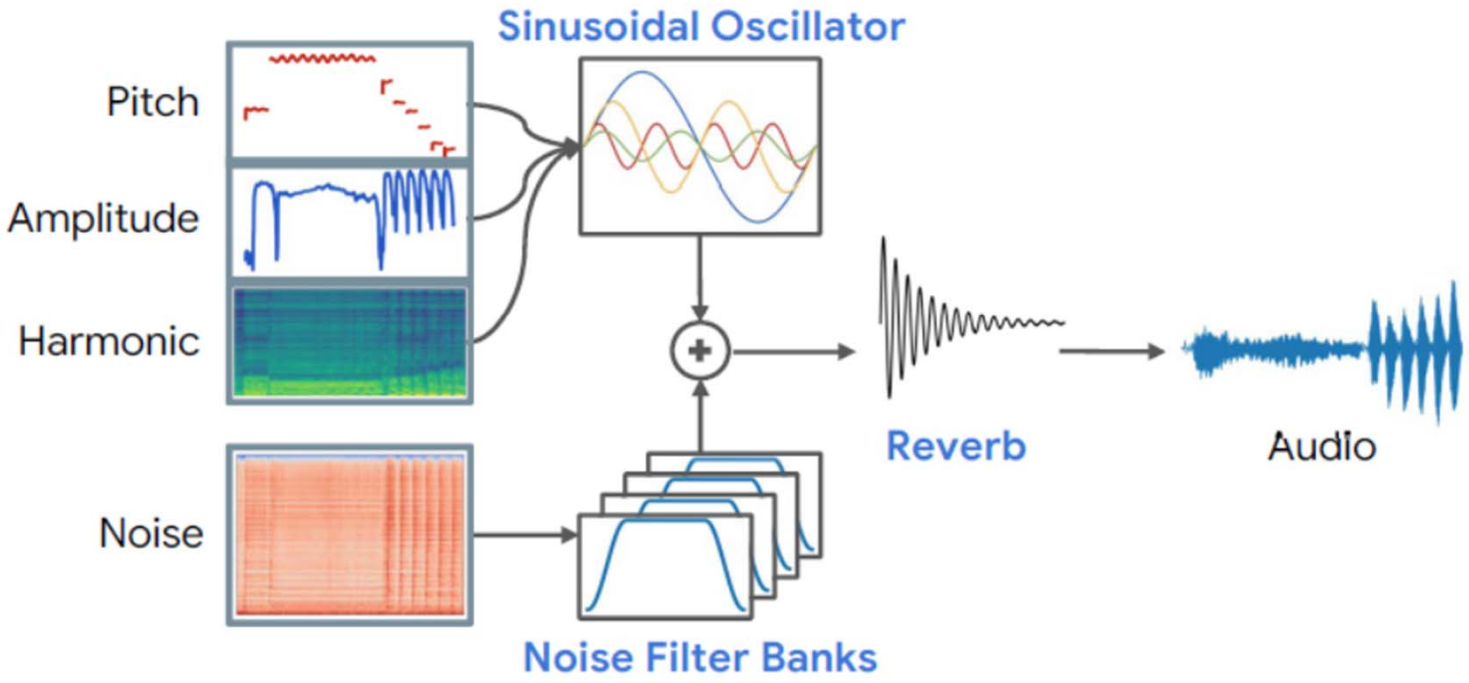
Engel et al. 2020 [7]



# DDSP: Differentiable Digital Signal Processing

https://github.com/lukewys/ISMIR2022-tutorial

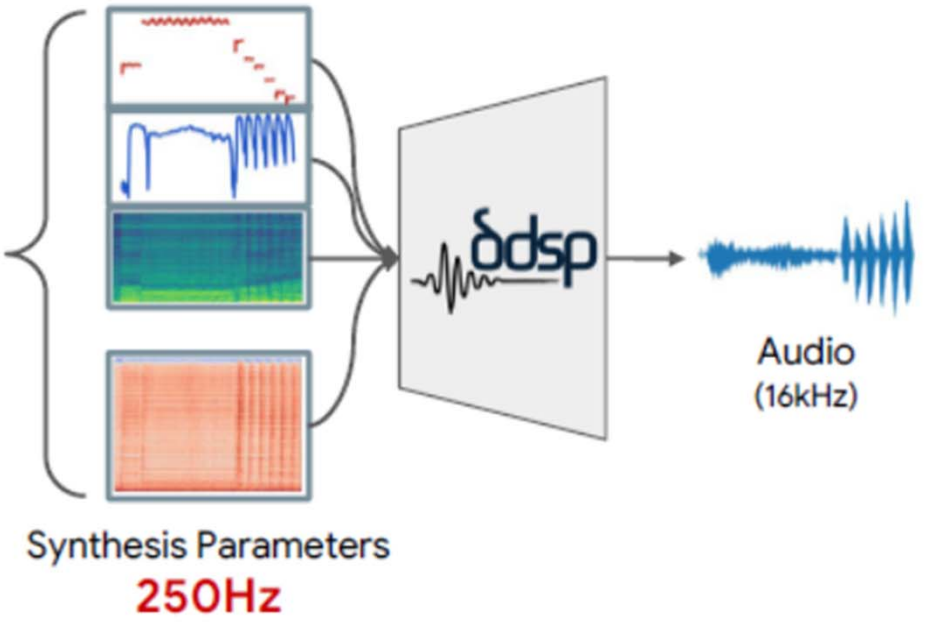
Engel et al. 2020 [7]



# Problem of Low-level Control

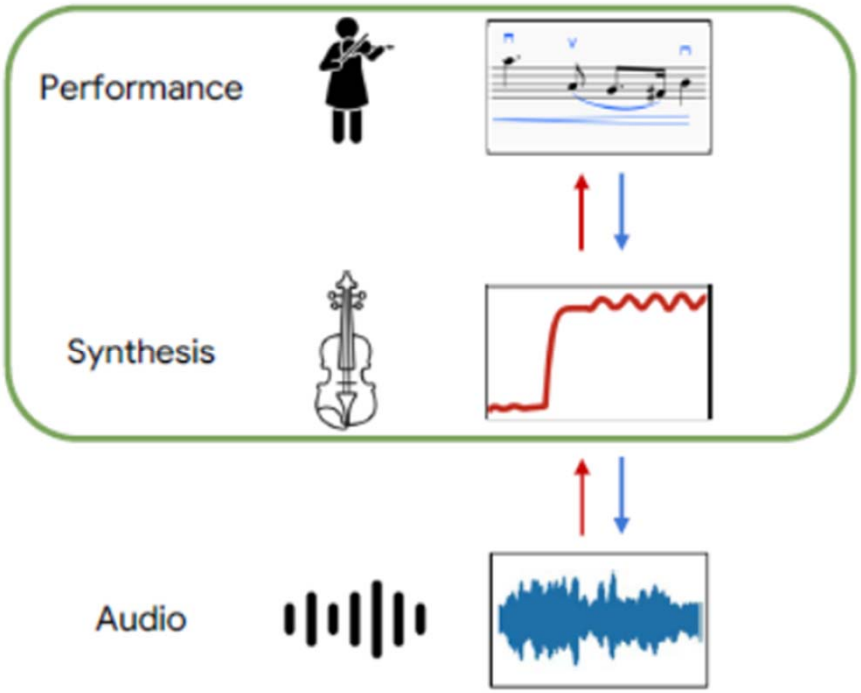
https://github.com/lukewys/ISMIR2022-tutorial

1 sec: **127 x 250 = ~32k**
**parameters to control!!**
Need high-level Control



# Extract Performance Parameter

https://github.com/lukewys/ISMIR2022-tutorial

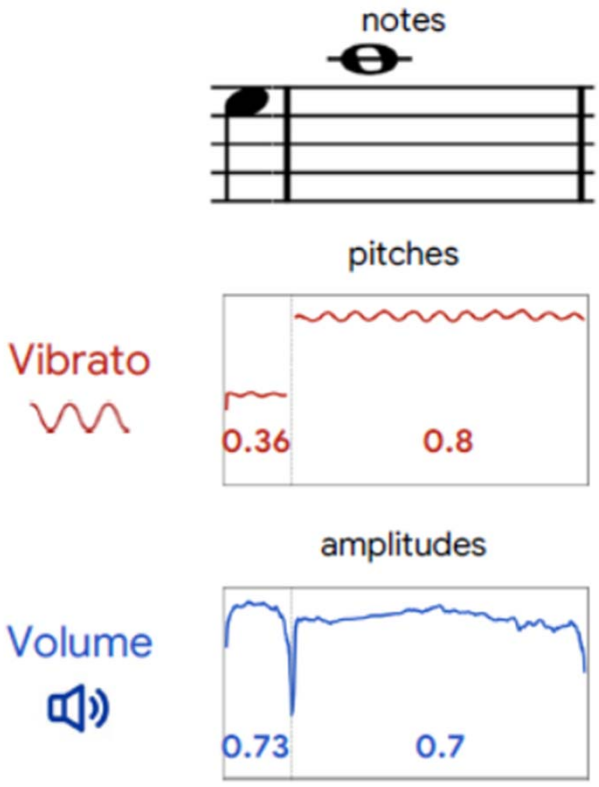



Summary statistics pooled over notes
6-D **scalar features**, scaled [0,1]:

- Volume
- Vibrato
- Brightness
- Attack Noise
- Volume Peak Position
- Volume Fluctuation

# Extract Performance Parameter

https://github.com/lukewys/ISMIR2022-tutorial

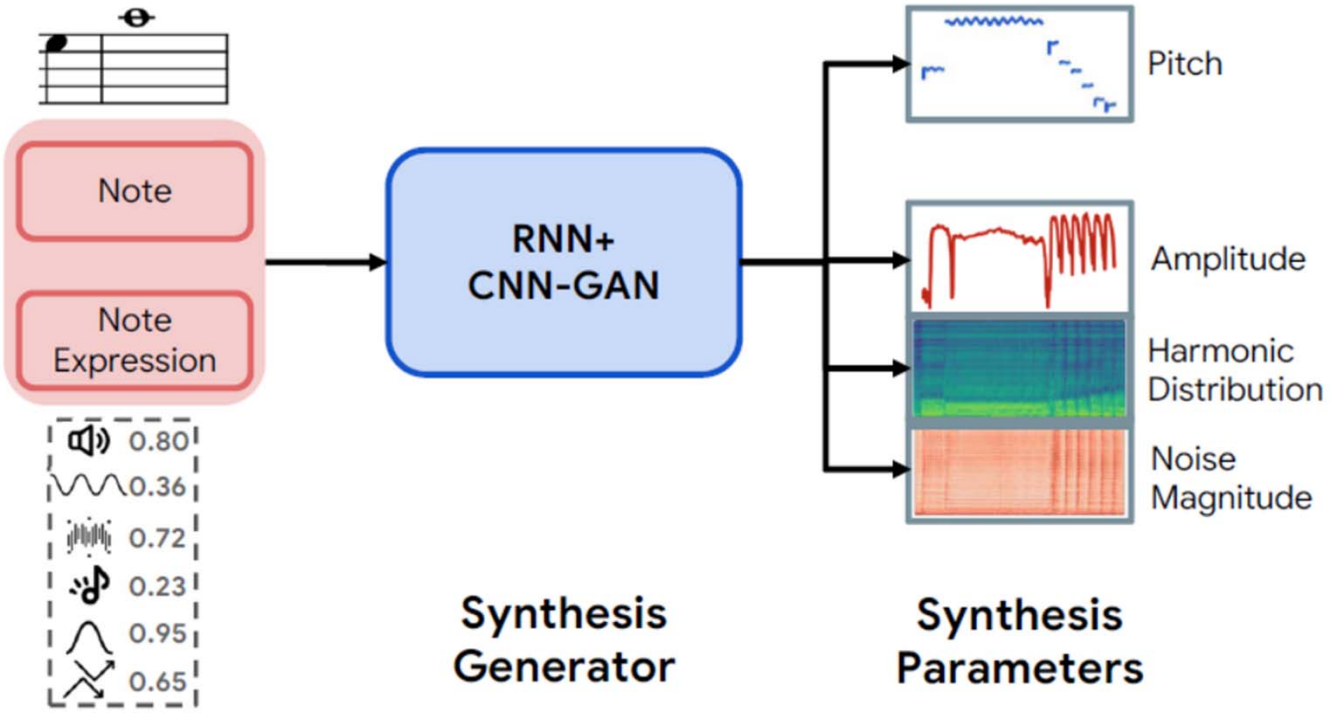



Summary statistics pooled over notes
6-D **scalar features**, scaled [0,1]:

- Volume
- Vibrato
- Brightness
- Attack Noise
- Volume Peak Position
- Volume Fluctuation

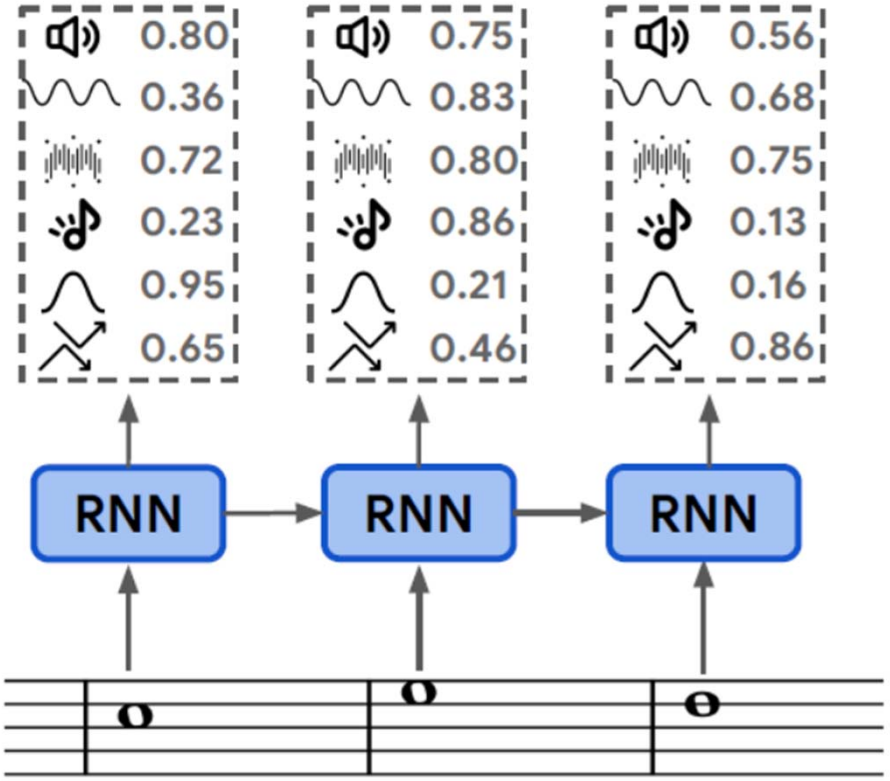
# Synthesis Generator

https://github.com/lukewys/ISMIR2022-tutorial



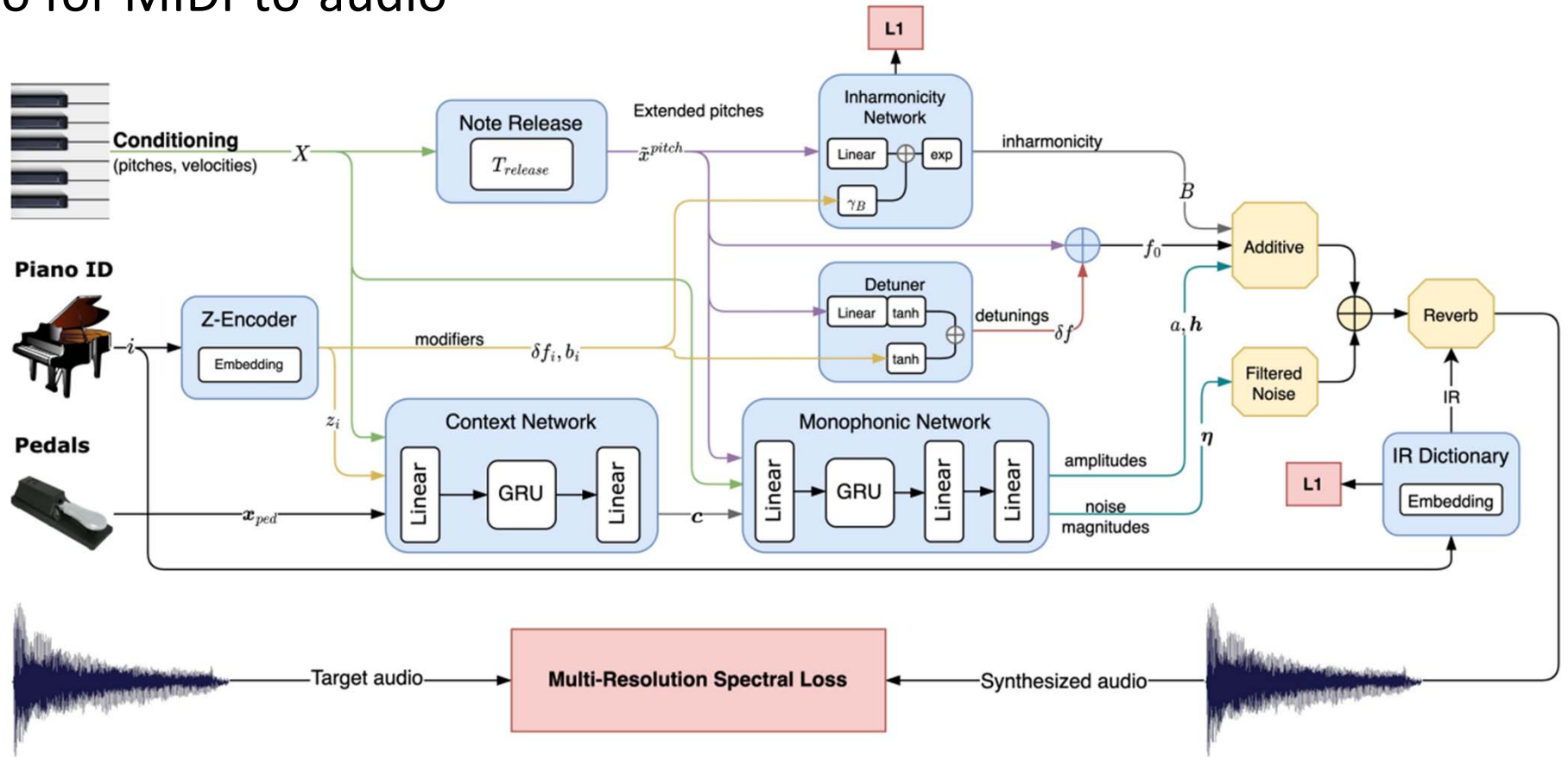
# Autoregressive Prior on Expression Controls

https://github.com/lukewys/ISMIR2022-tutorial



# DDSP-Piano (AES'23)

- Also for MIDI-to-audio

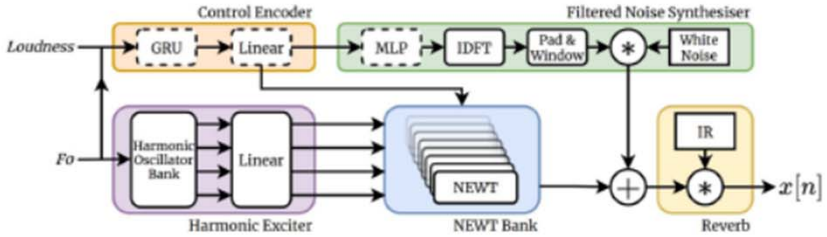



Ref: Renault et al, "DDSP-Piano: a neural sound synthesizer informed by instrument knowledge," AES 2023

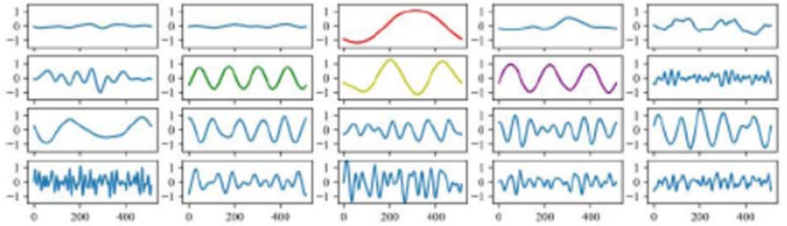
# Other Differentiable Synthesis Works

- Also for MIDI-to-audio

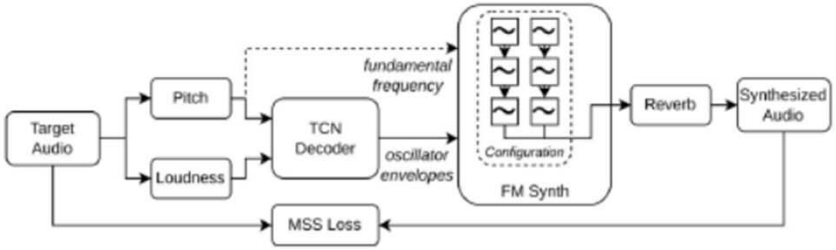
Waveshaping Synthesis [8]




Wavetable Synthesis [9]



FM Synthesis [10]




Ref: https://github.com/lukewys/ISMIR2022-tutorial

# Outline

- DSP-based music synthesizers
- DL-based generation of single notes
- DL-based generation of note sequences
- **DL-based generation of loops**
  - StyleGAN

# 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/

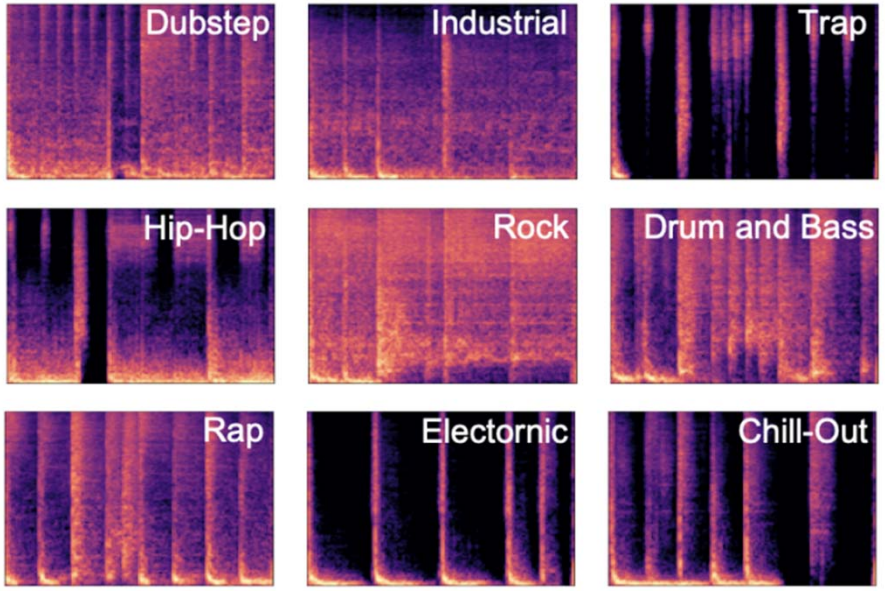
- More challenging than note synthesis
  - Longer
  - May not have associated MIDI
- Less challenging than *general* unconditional music generation
  - Fixed number of bars (e.g., 4-bar)
  - Basic building block of music

# Loop Generation by StyleGAN2 (ISMIR'21)

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*

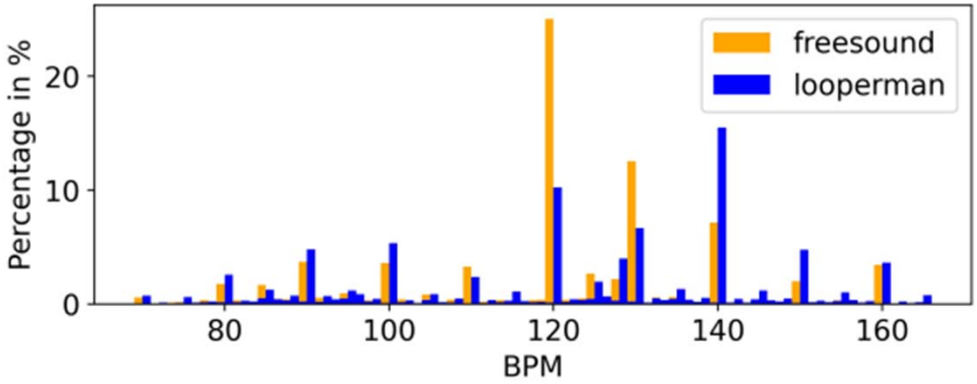



Ref: Hung et al, "A benchmarking initiative for audio-domain music generation using the FreeSound loop dataset," ISMIR 2021

# 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

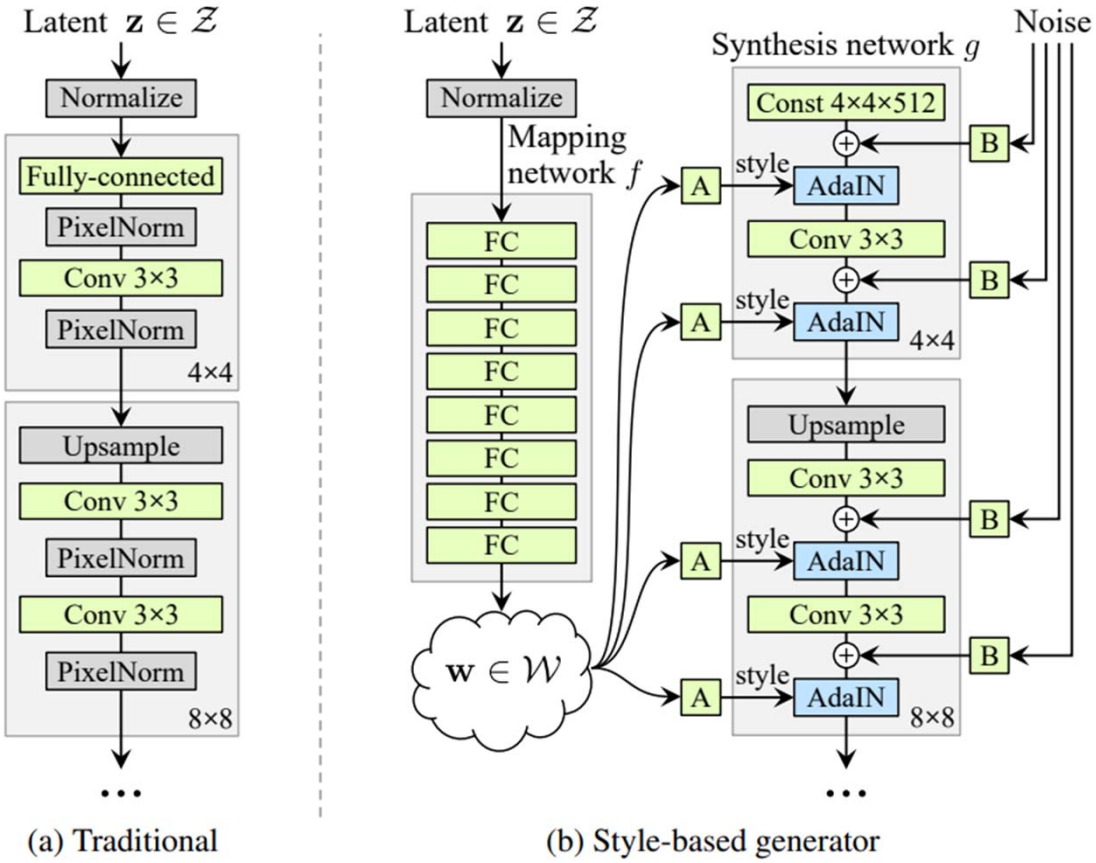



- All time-scaled to **120 BPM**
  - "We use pyrubberband to temporally stretch all the loops to **2-second** long […] most sounded plausible with little perceptible artifacts"
    - "This, however, may not be the case if the loops are not drum loops. Some data filtering might be needed then, e.g., to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



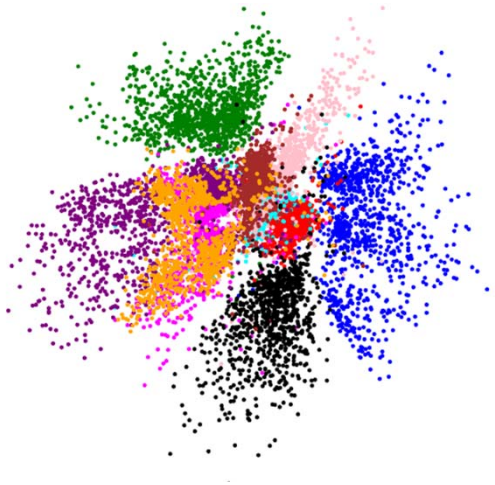


Ref 1: Karras et al, "A style-based generator architecture for generative adversarial networks," CVPR 2019
Ref 2: Karras et al, "Analyzing and improving the image quality of StyleGAN," CVPR 2020

# AdaIN: Adaptive Instance Normalization

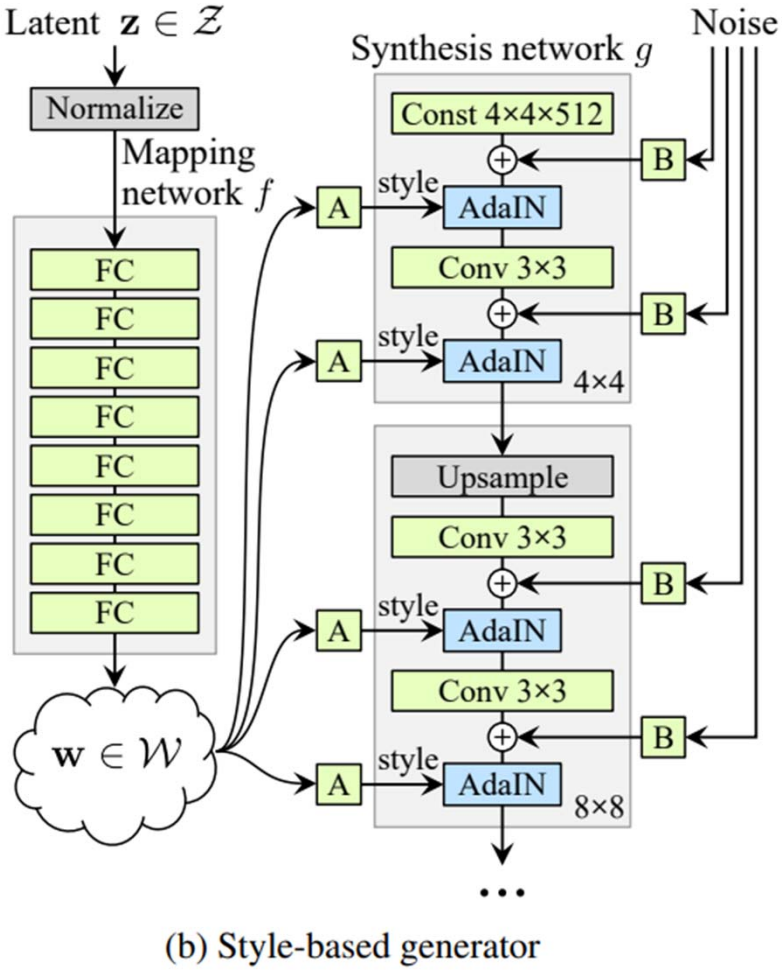
$$\mathrm{AdaIN}(\mathbf{x}_i, \mathbf{y}) = \mathbf{y}_{s,i} \frac{\mathbf{x}_i - \mu(\mathbf{x}_i)}{\sigma(\mathbf{x}_i)} + \mathbf{y}_{b,i},$$

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



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

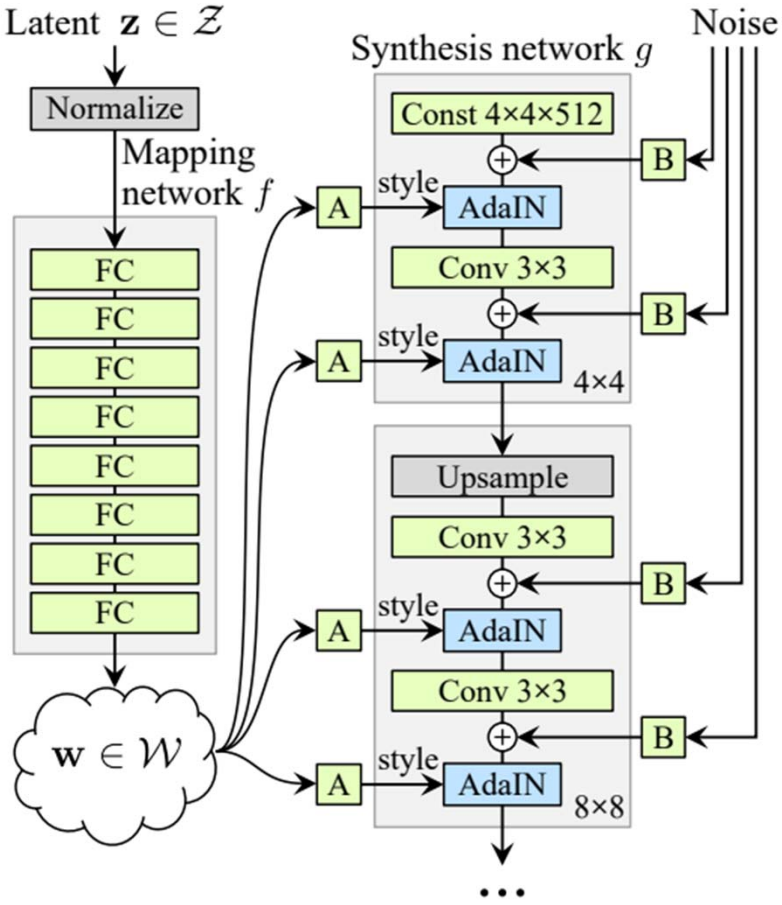
# StyleGAN: Interpolation

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

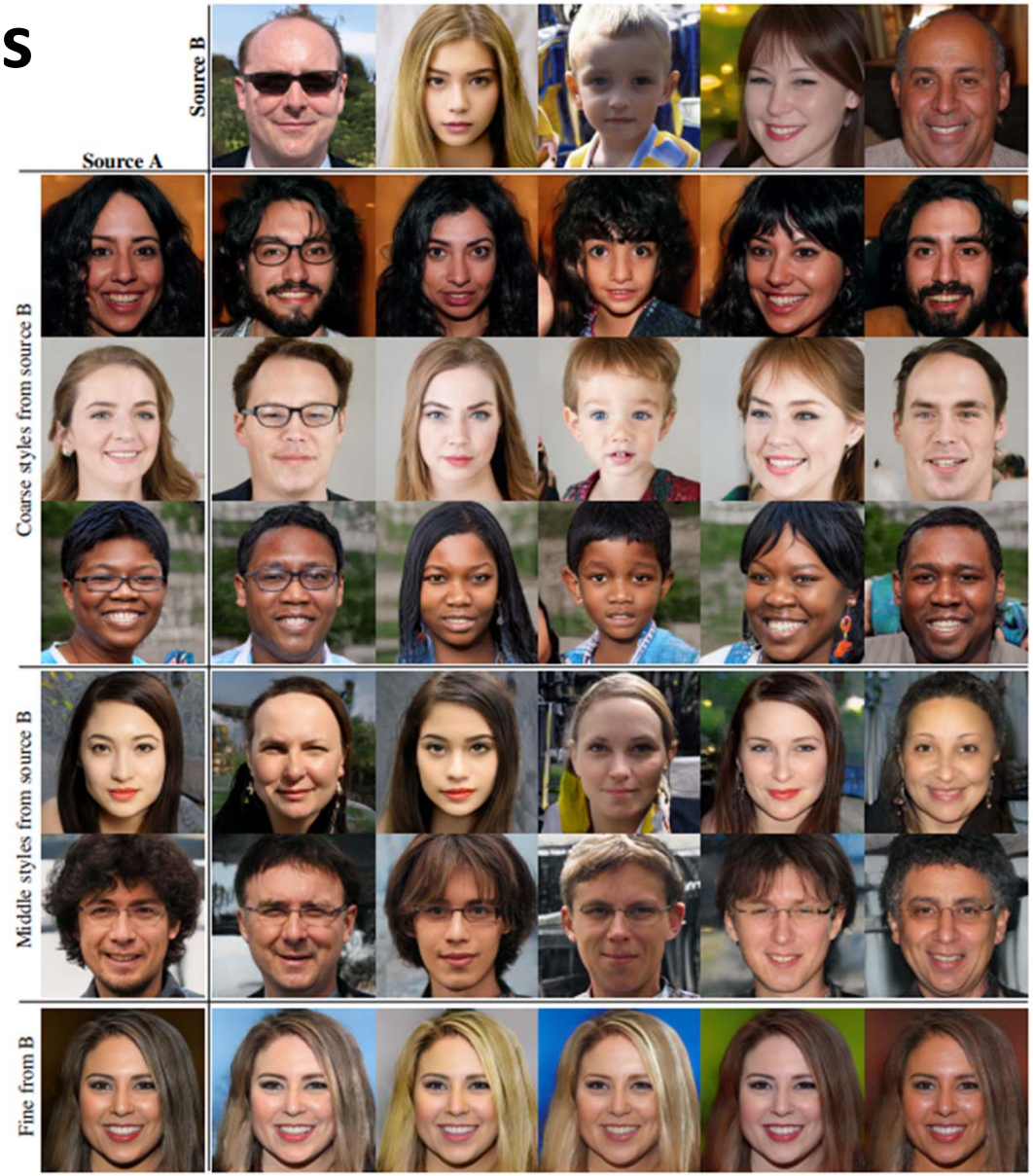
- $\{A_1^{low}, A_1^{mid}, A_2^{high}\} \rightarrow$ ?




(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

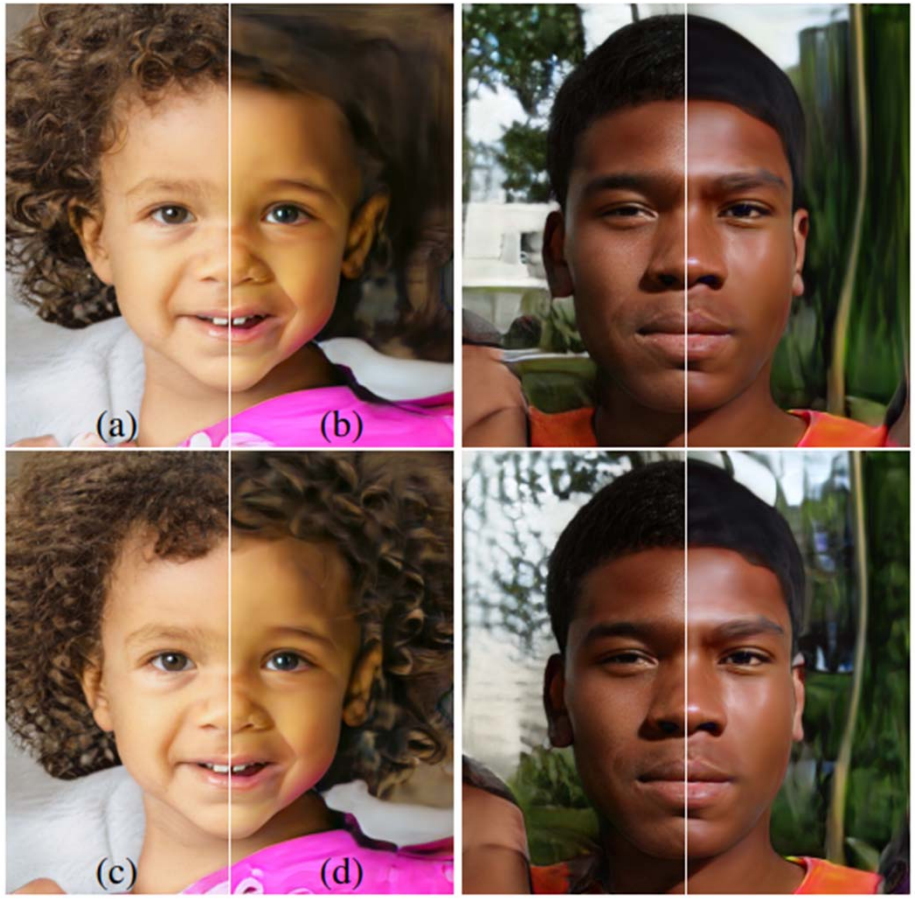



Figure 5. Effect of noise inputs at different layers of our generator. (a) Noise is applied to all layers. (b) No noise. (c) Noise in fine layers only ($64^2$ – $1024^2$). (d) Noise in coarse layers only ($4^2$ – $32^2$). We can see that the artificial omission of noise leads to featureless "painterly" 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.

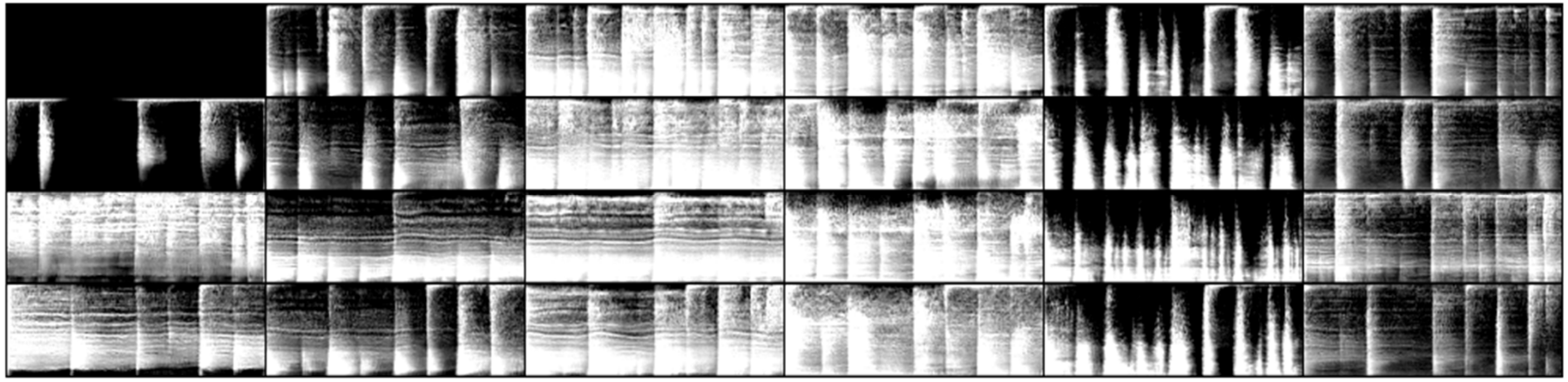
| | Configuration | FID ↓ |
|---|---|---|
| A | Baseline StyleGAN [24] | 4.40 |
| B | + Weight demodulation | 4.39 |
| C | + Lazy regularization | 4.38 |
| D | + Path length regularization | 4.34 |
| E | + No growing, new G & D arch. | 3.31 |
| F | + Large networks (StyleGAN2) | **2.84** |
| | Config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

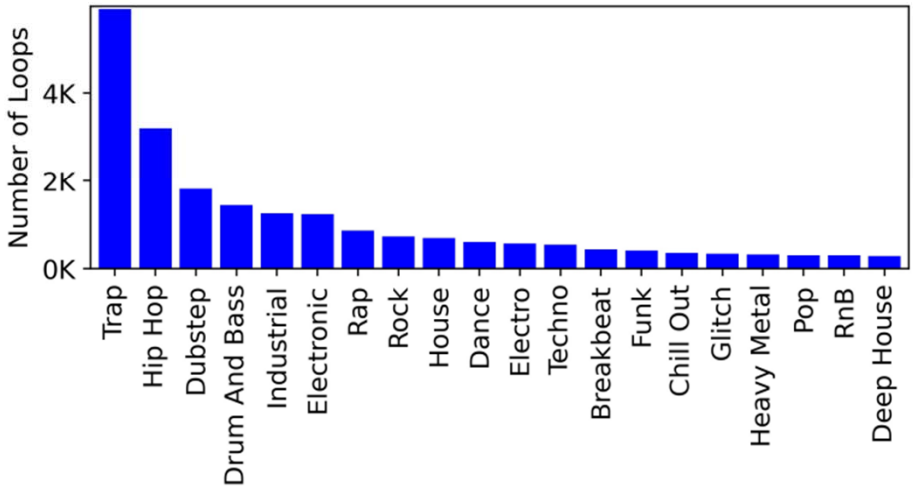
Ref: Hung et al, "A benchmarking initiative for audio-domain music generation using the FreeSound loop dataset," ISMIR 2021

# 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 $p(y)$,

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

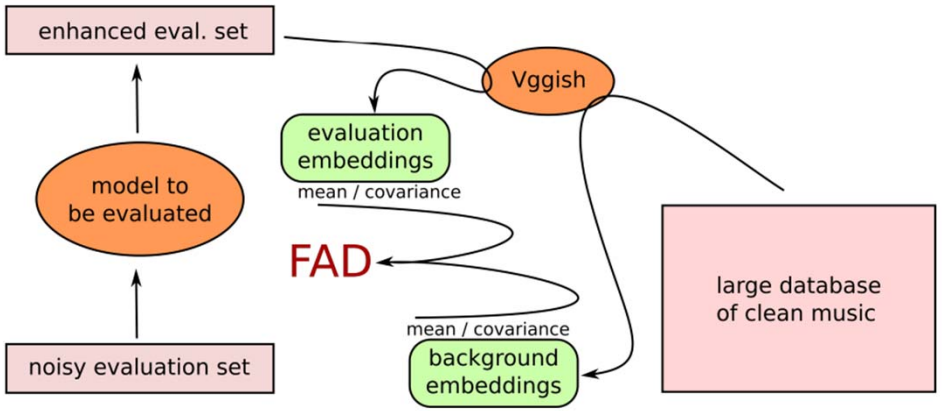
- To compute the IS, we use short-chunk CNN to train a 66-class classifier for **loop genre** classification




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



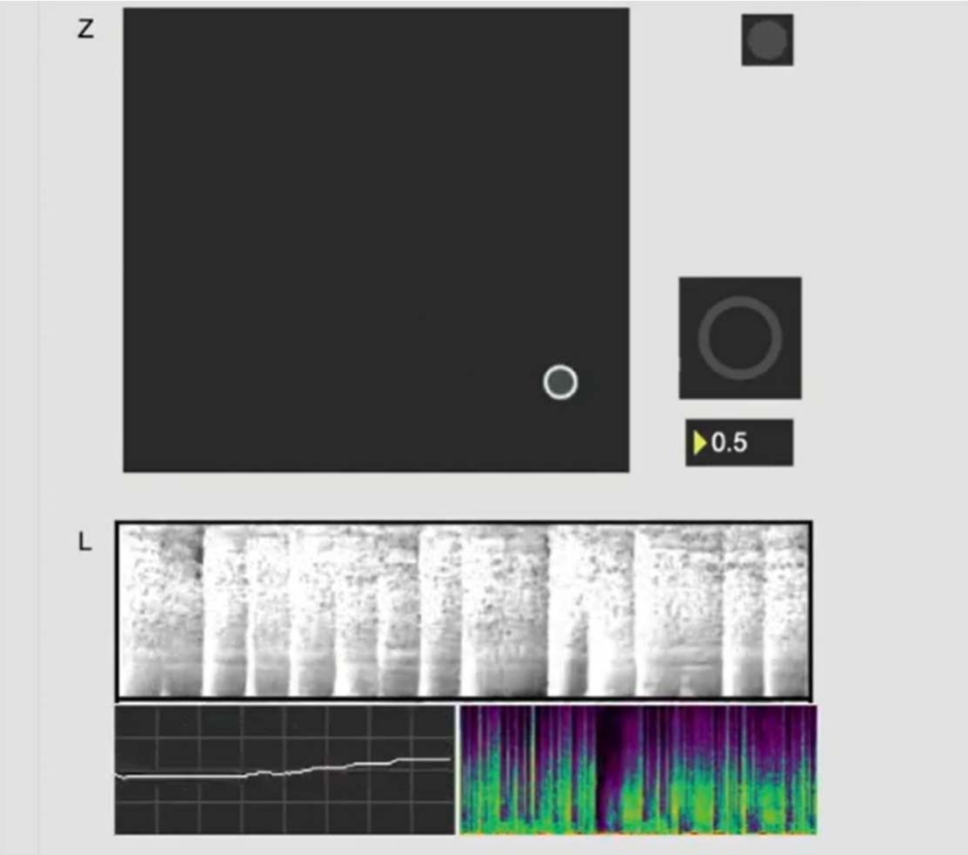

$$\mathbf{F}(\mathcal{N}_b, \mathcal{N}_e) = \|\mu_b - \mu_e\|^2 + tr(\Sigma_b + \Sigma_e - 2\sqrt{\Sigma_b \Sigma_e})$$

  - 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



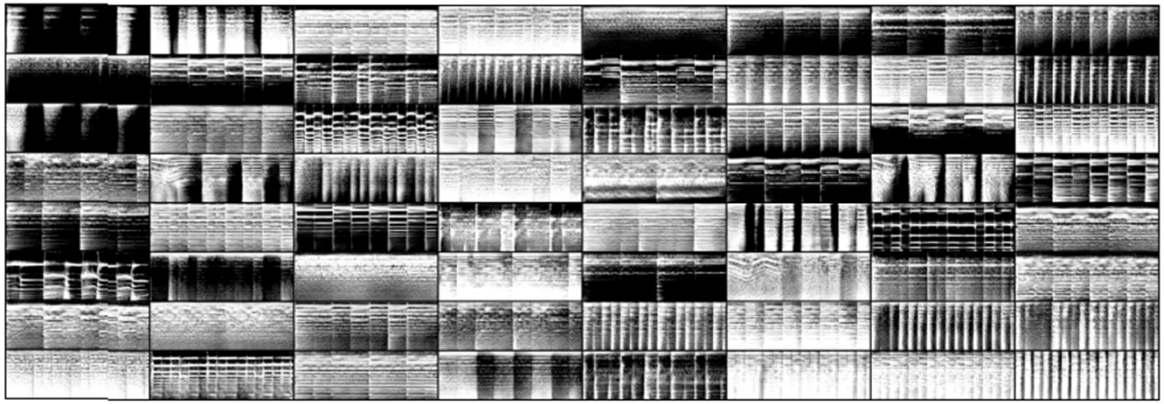

By Nao Tokui (2023)

# Loop Generation by Projected GAN (ISMIR'22)

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

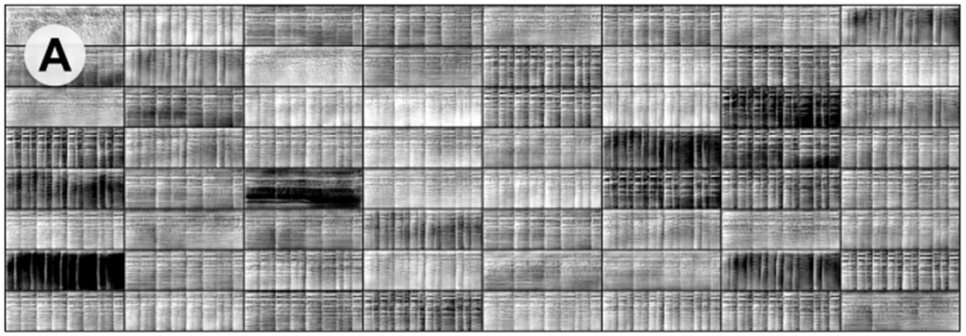
- Consider the generation of not only drum loops but also **synth loops**

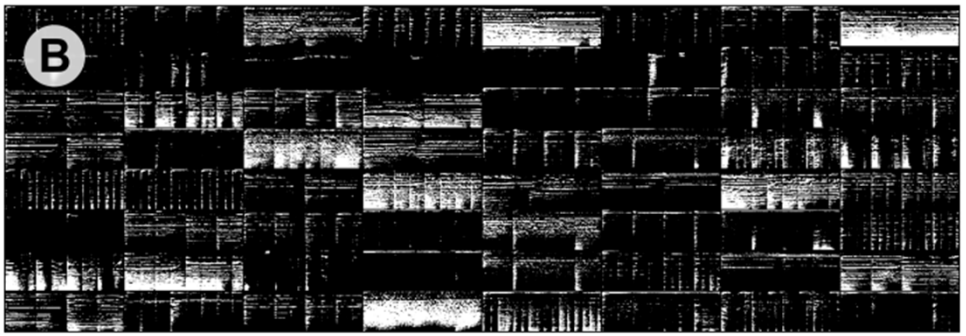


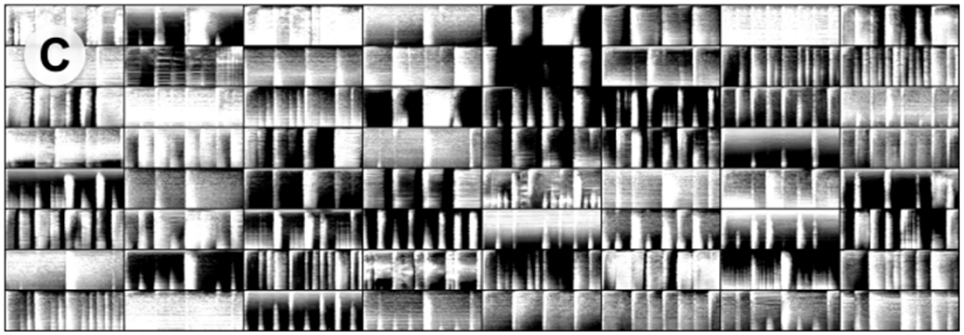
Ref: Yeh et al, "Exploiting pre-trained feature networks for generative adversarial networks in audio-domain loop generation," ISMIR 2022

# Different Cases of GAN Training

- Also studied in the Projected GAN paper for loop generation



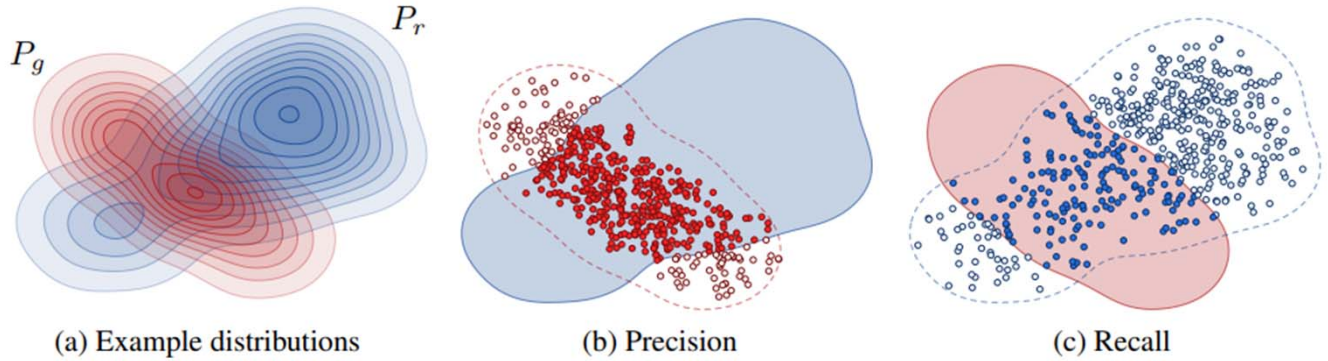



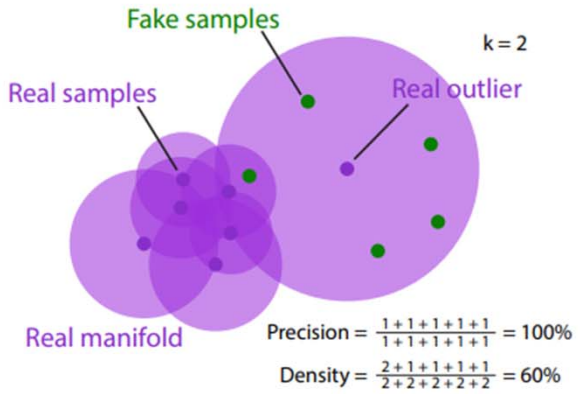

- Examples from Mel-spectrogram generation (not Mel-vocoder)
  - (A) Mode collapse
  - (B) Gradient vanishing
  - (C) Successful training

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

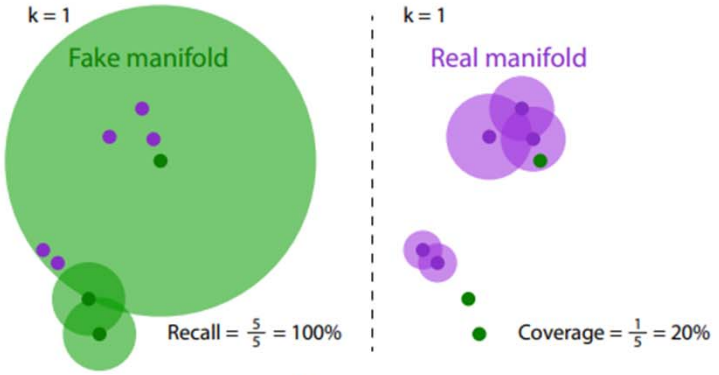
# More Evaluation Metrics for Unconditional Generation




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




(a) Precision versus density.



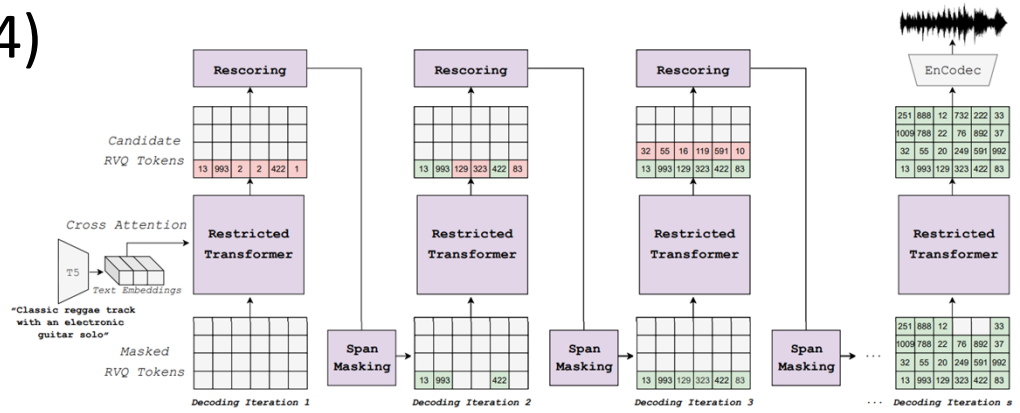

(b) Recall versus coverage.

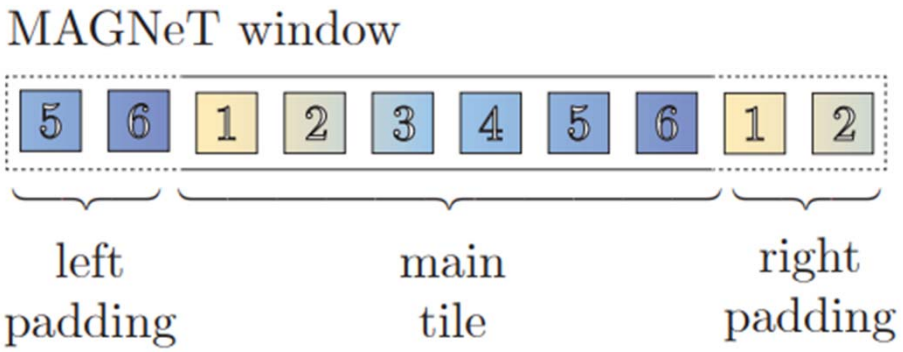
Ref 1: Kynkäänniemi et al., "Improved precision and recall metric for assessing generative models," NeurIPS 2019
Ref 2: Naeem et al., "Reliable fidelity and diversity metrics for generative models," ICML 2020.

# Train-Free Text-Conditioned Loop Generation (ISMIR'25)

https://gladia-research-group.github.io/loopgen-demo/

- Model backbone: MAGNET (ICLR'24)
  - **NAR**, Transformer-based
- **Training-free**
  - Generate tokens in a ***circular*** pattern, attending to the beginning of the audio when creating ending
  - Require no training, and no specialized loop datasets




Ref: Marincione et al, "LoopGen: Training-free loopable music generation," ISMIR 2025