2024 edition

Deep Learning for Music Analysis and Generation

MIDI Generation

(x → MIDI)

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

Objectives

- **Language modeling** (LM) for symbolic-domain music generation
  - To account for ***long-range dependency***
  - Get a **token** representation of music
  - Then build an LM
    - By **RNN**
    - By **Transformer**

```
bach piano_strings start tempo90 piano:v72:G1
piano:v72:G2 piano:v72:B4 piano:v72:D4 violin:v80:G4
piano:v72:G4 piano:v72:B5 piano:v72:D5 wait:12
piano:v0:B5 wait:5 piano:v72:D5 wait:12 piano:v0:D5
wait:4 piano:v0:G1 piano:v0:G2 piano:v0:B4 piano:v0:D4
violin:v0:G4 piano:v0:G4 wait:1 piano:v72:G5 wait:12
piano:v0:G5 wait:5 piano:v72:D5 wait:12 piano:v0:D5
wait:5 piano:v72:B5 wait:12
```

Reference: Four ISMIR Tutorials

- "**Machine-Learning for Symbolic Music Generation"** by Pierre Roy (Spotify) and Jean-Pierre Briot (Sony CSL) at **ISMIR 2017**
  https://ismir2017.ismir.net/tutorials/index.html#T4
- "**Generating Music with GANs—An Overview and Case Studies**" by Hao-Wen Dong and Yi-Hsuan Yang (Academia Sinica) at **ISMIR 2019**
  https://salu133445.github.io/ismir2019tutorial/
- "**Designing Generative Models for Interactive Co-creation**" by Anna Huang (Google), Jon Gillick (UC Berkeley), Chris Donahue (Stanford) at **ISMIR 2021**
  https://github.com/chrisdonahue/music-cocreation-tutorial
- "**Transformer-based Symbolic Music Generation**" by Berker Banar (QMUL), Pedro Sarmento (QMUL), Sara Adkins (infinitealbum) at **ISMIR 2023**
  https://ismir2023.ismir.net/tutorials/

Reference: ACM Multimedia 2021 Tutorial

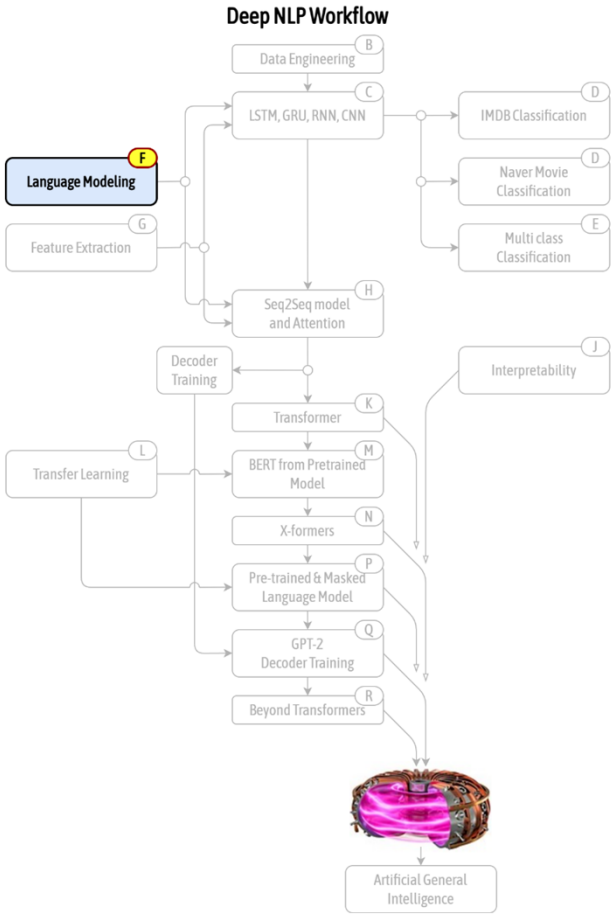
https://www.microsoft.com/en-us/research/uploads/prod/2021/10/Tutorial-on-AI-Music-Composition-@ACM-MM-2021.pdf

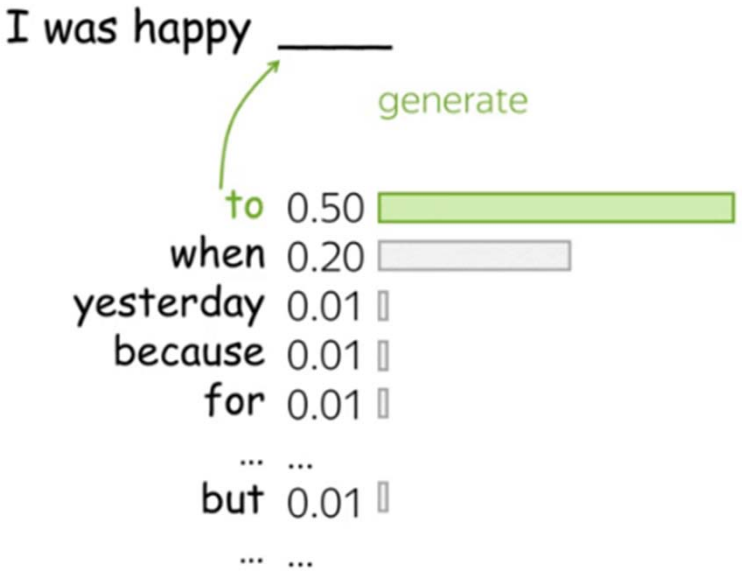
A Tutorial on AI Music Composition

Xu Tan & Xiaobing Li
Microsoft Research Asia & Central Conservatory of Music, China

Reference: Deep Learning Bible – NLP

https://wikidocs.net/178448

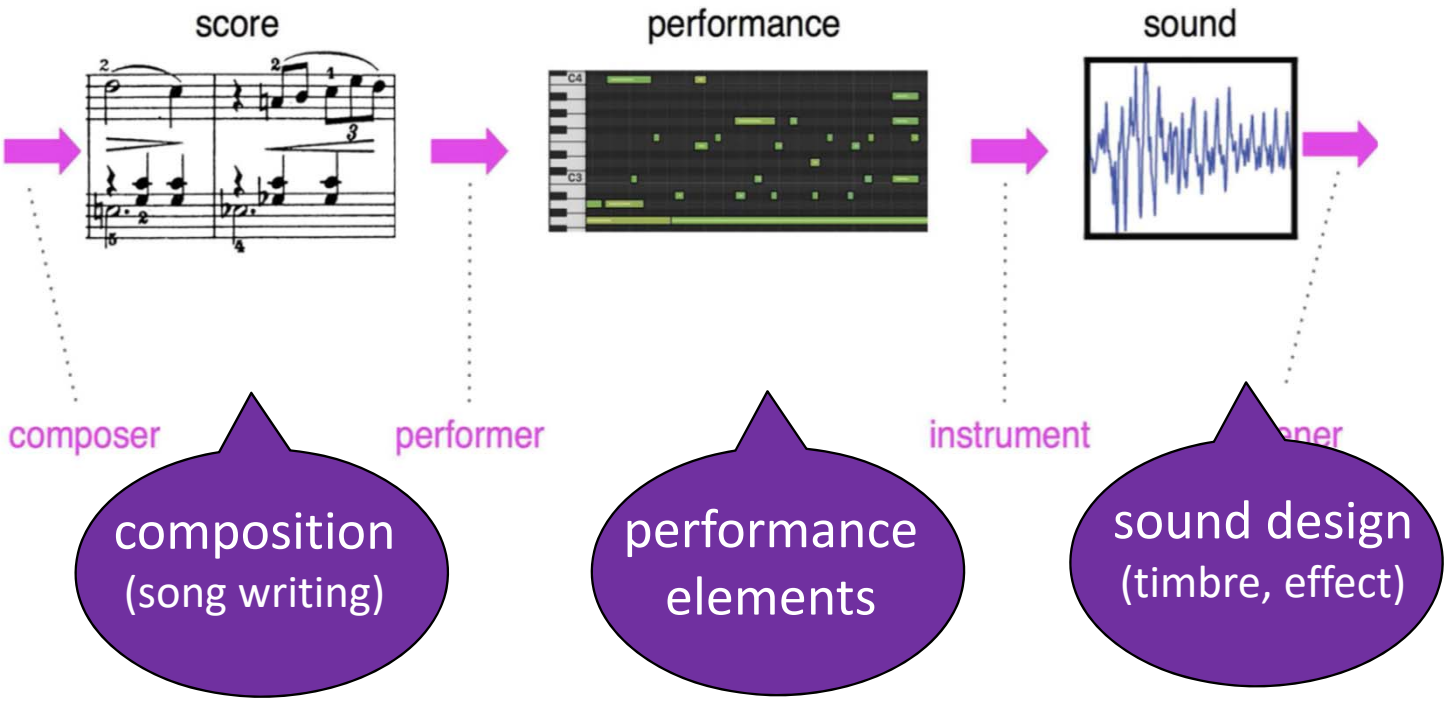




Outline

- **General ideas**
- Image-based approach for symbolic music generation
- Text-based approach for symbolic music generation
  - RNN
  - Transformer
- Token design
- Building your first MIDI Transformer
- Advanced models

Music Production Pipeline



Music Composition

基礎班學習重點

- **曲式學習**
  - 介紹世界各地的Pop Music
  - Rock Music的結構
  - Country Music的結構
  - 流行曲曲式的多種可能性
- **和弦理論 及 和聲鋪排**
  - 學習編排和弦進程的規則
  - 學習簡單好聽的Chord Progression幫助創作旋律
  - 學習加入調外和弦來製造色彩
  - 認識不同音樂類型所慣常使用的和弦進程
- **旋律創作**
  - 學習如何在Chord Progression之上創作旋律
  - 學習構思動機（短樂句 Short Phrase）
  - 學習把發展動機成一整段旋律
  - 學習旋律如何製造主歌與副歌的對比
  - 前奏與間奏的創作

https://anthelionmusic.com/musical-composition

Music Composition

國立清華大學音樂系

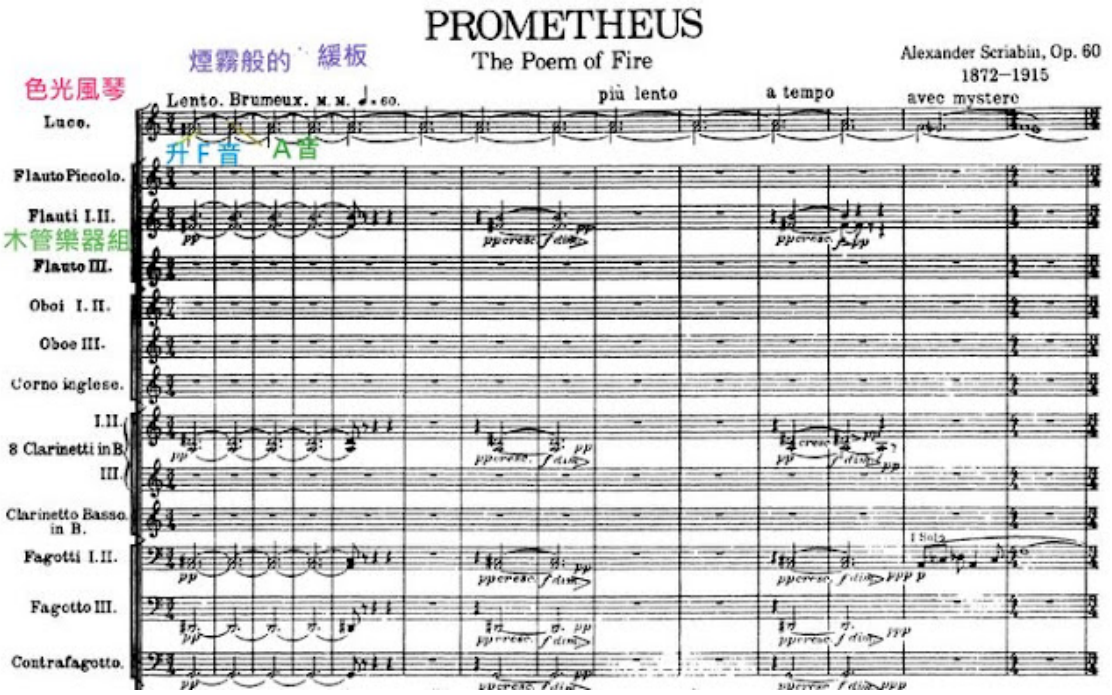
理論作曲一期末會考繳交曲目與規則

2018.02.01 修正
經 1022 第七次系務會議通過
經 1081 第五次系務會議通過

主修考試範圍				
	一 年 級	二 年 級	三 年 級	四 年 級
上學期	不考試（主修老師依學生的個別能力加強音樂理論）	1. 混聲四部合唱曲 2. 自由創作	1. 復格曲（巴赫風格） 2. 室內樂曲(7人以上編制）	自曲創作兩首樂曲(不同編制)
下學期	1. 器樂獨奏曲(非鋼琴獨奏曲) 2. 自由創作	1. 古典奏鳴曲式（一個樂章） 2. 弦樂四重奏	1. 管弦樂曲（兩管編制，曲長:12 至 15 分鐘之間）	1. 畢業製作（至少 30 分鐘） 2. 畢業論文（樂曲解析）

https://music.site.nthu.edu.tw/p/405-1113-122293,c12754.php?Lang=zh-tw

Music Composition



(images are from the internet)

Music Generation: Multiple aspects

https://www.eslite.com/product/1001117242682044820004

Music “Performance Generation”

https://www.upmedia.mg/news_info.php?SerialNo=83273

Types of Music Generation Research

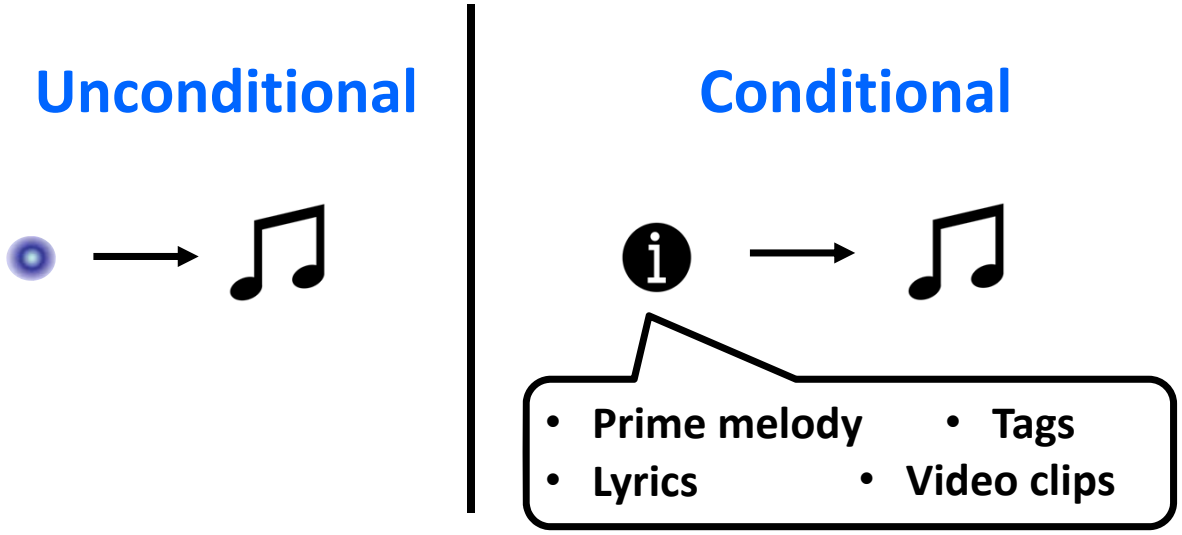
1. **Composition** (symbolic): MIDI scores (or other types of symbolic format)
2. **Performance** (symbolic): MIDI performances (scores + expression)
   a. Generate MIDI performances *from scratch* (i.e., generate scores and their expression at the same time)
   b. Given MIDI score, generate MIDI performance (e.g., VirtuosoNet [1])
3. **Audio**
   a. Generate audio *from scratch*
   b. Given MIDI, generate audio

Will talk about **1** and **2a** in this lecture

[1] Jeong et al, "VirtuosoNet: A hierarchical RNN-based system for modeling expressive piano performance," ISMIR 2019

Unconditional or Conditional Generation

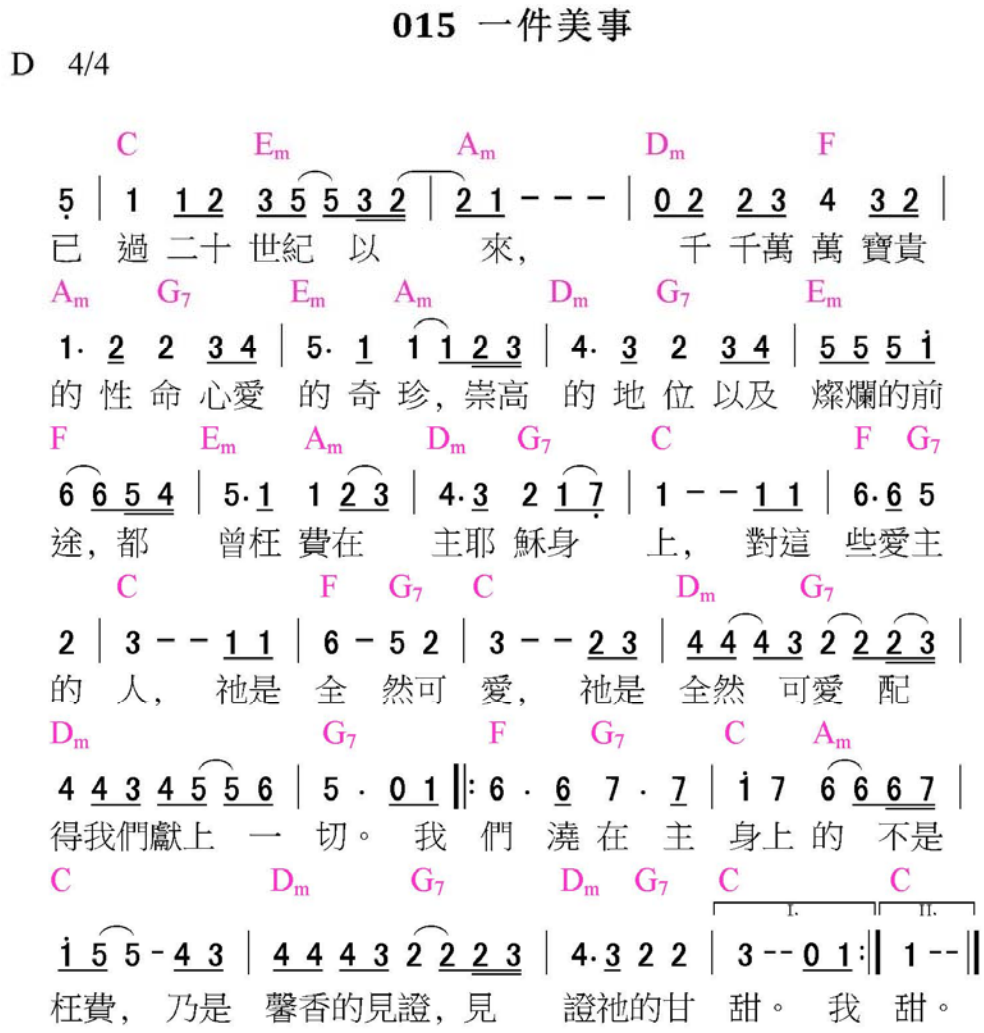
- The generation of MIDI scores can also be conditional



(Figure made by Hao-Min Liu)

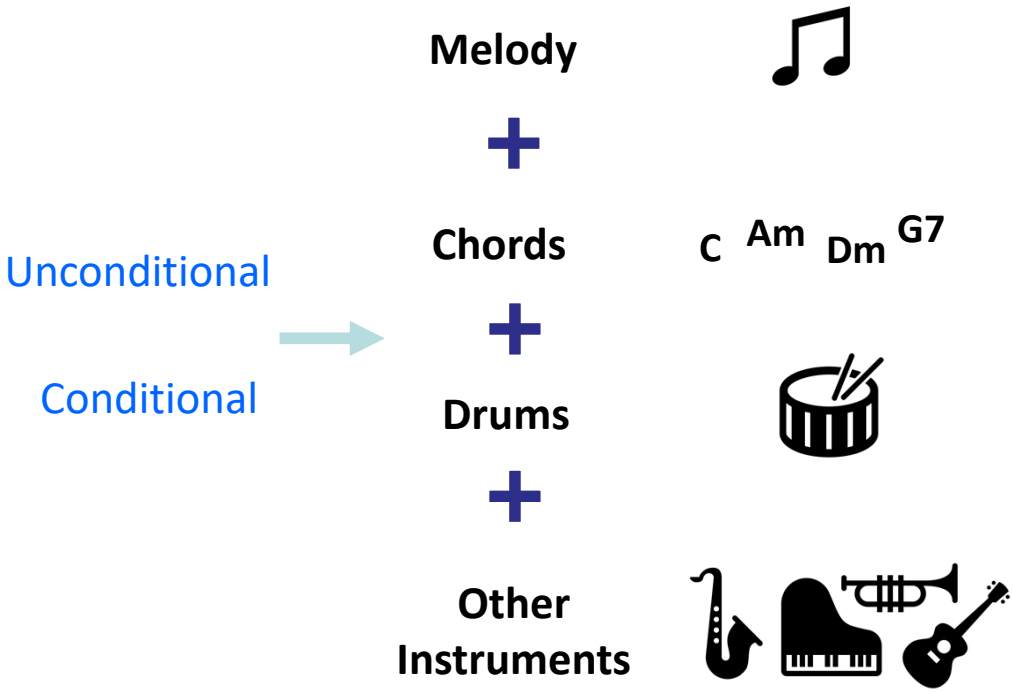
Lead Sheet Generation

- Lead sheet
  - melody
  - chord
  - (lyrics)
- Possible generation tasks
  - Melody generation
  - Given chord, generate melody
  - Given lyrics, generate melody
  - Melody+chord generation
  - Given melody, generate chord
  - Given melody, generate lyrics
  - Lyrics generation



Task Complexity: From Single Track to Multiple Tracks

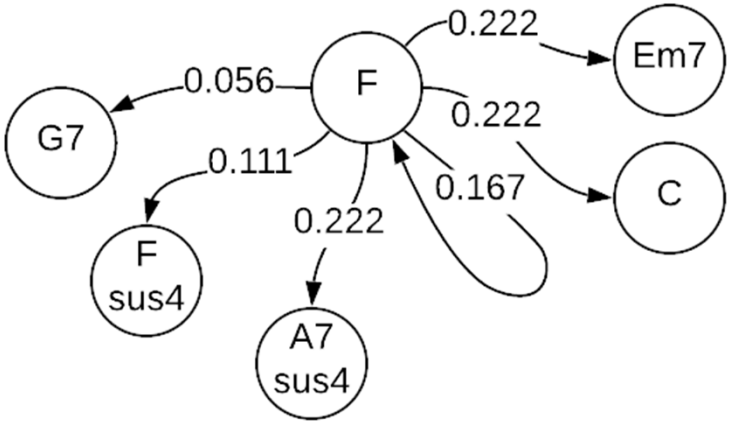
- Melody generation → piano generation → accompaniment generation



(Figure made by Hao-Min Liu)

Algorithmic Composition in the Old Days

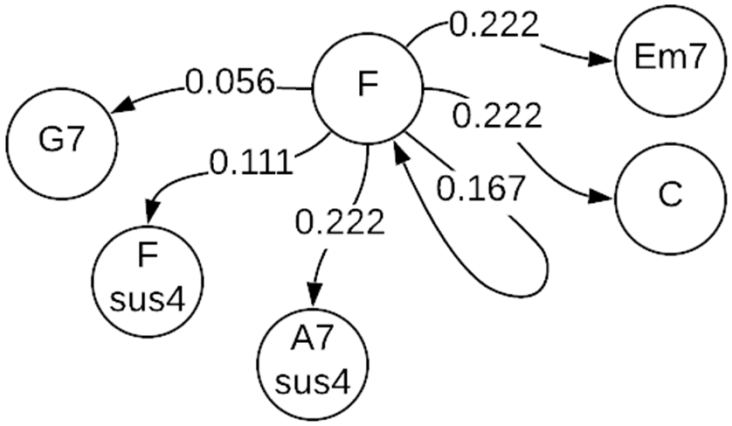
- Based on **probabilities**
- Given "x_{t-1}" (the last one), predict "x_t" (the current one)



https://towardsdatascience.com/markov-chain-for-music-generation-932ea8a88305

Algorithmic Composition in the Old Days

- **Rule-based**
  - safe but rigid
- **Statistical-based (e.g., Markov chain)**
  - limited flexibility



https://towardsdatascience.com/markov-chain-for-music-generation-932ea8a88305

Automatic Music Composition by Neural Networks

- The relationship between the input and output is modeled through a large number of layers, with “hidden states” that may not have a physical meaning

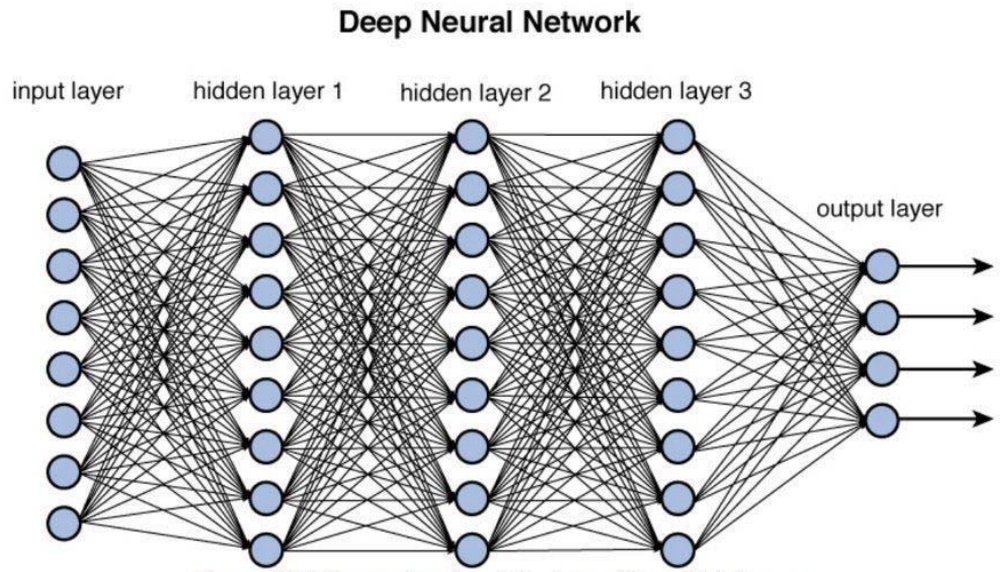
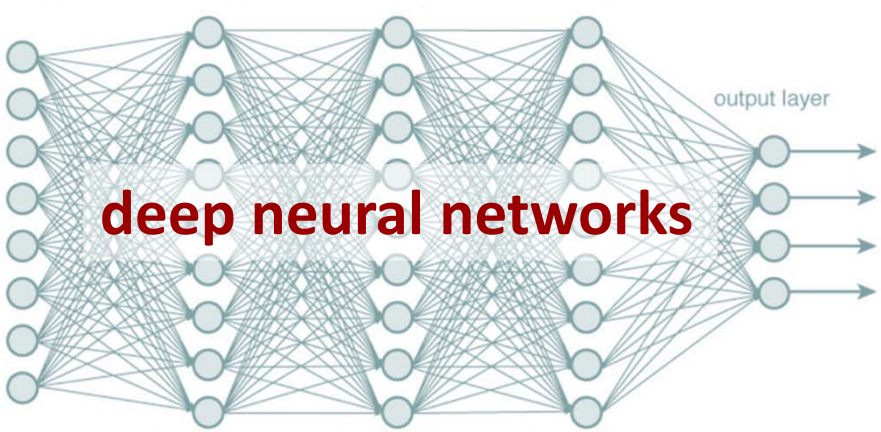


Figure 12.2 Deep network architecture with multiple layers.

https://towardsdatascience.com/training-deep-neural-networks-9fdb1964b964

Autoregressive Models for Music Generation

- Based on **probabilities**
- ~~Given "x_{t-1}" (the last one), predict "x_t" (the current one)~~



- Given "x_1, …, x_{t-2}, x_{t-1}" (all the past), predict "x_t" (the current one)

Autoregressive Models for Music Generation

- A piece of music is considered as a sequence of "**events**" (i.e., tokens)
- A model is trained to predict "**what comes next**", given the history of previous events

$$P(x_t \mid x_{t-1}, x_{t-2}, x_{t-3}, \ldots), \; x_t \in S$$

$x_t \in S$: a finite collection of possible events

"As we listen to melodies, our brain also guesses what's next"
https://bigthink.com/surprising-science/music-brain-predict

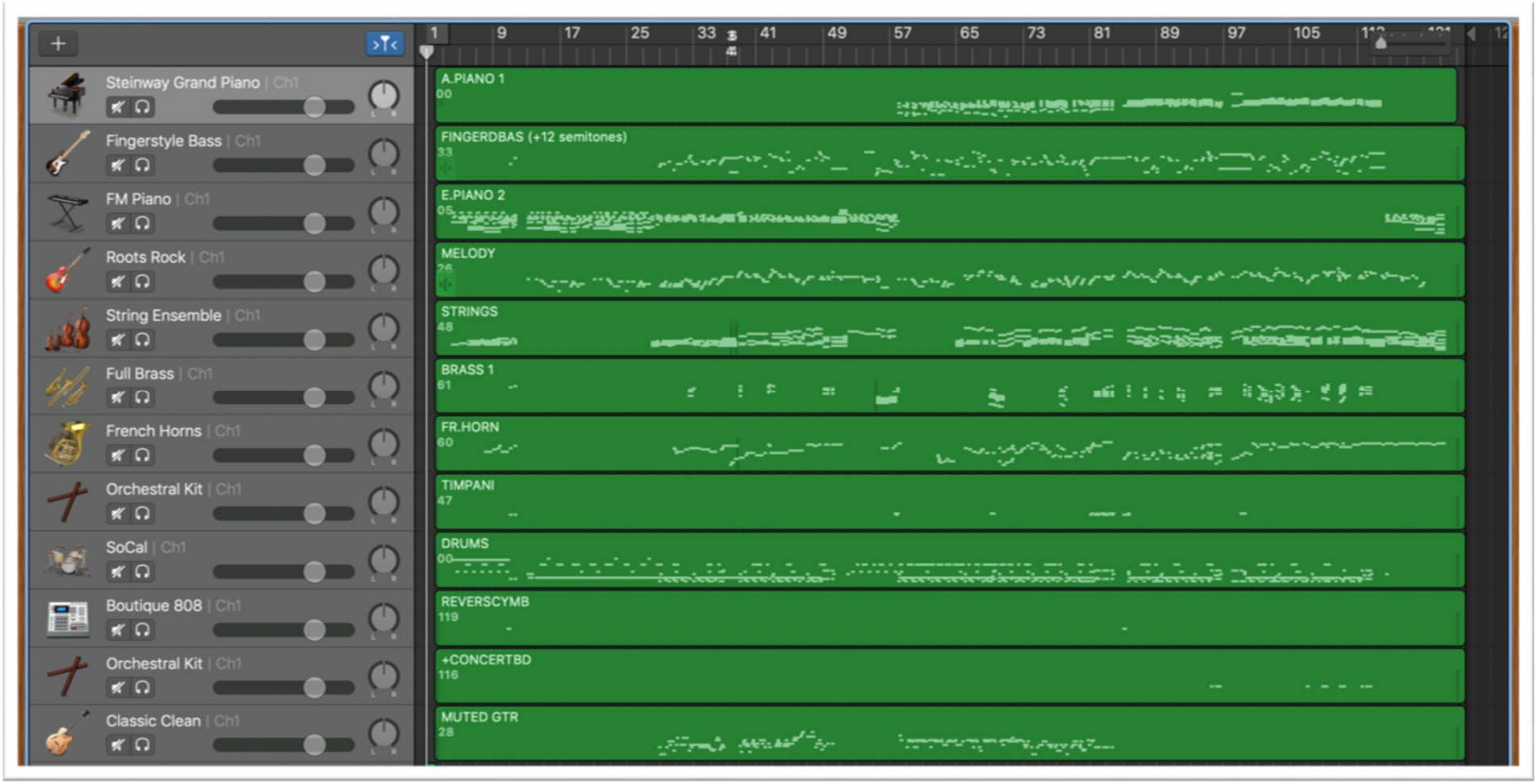
- Given a **prompt**, generate **continuation**
- But there are also *non-autoregressive* approaches

Outline

- General ideas
- **Image-based approach for symbolic music generation**
- Text-based approach for symbolic music generation
  - RNN
  - Transformer
- Token design
- Building your first MIDI Transformer
- Advanced models

Two Main Approaches

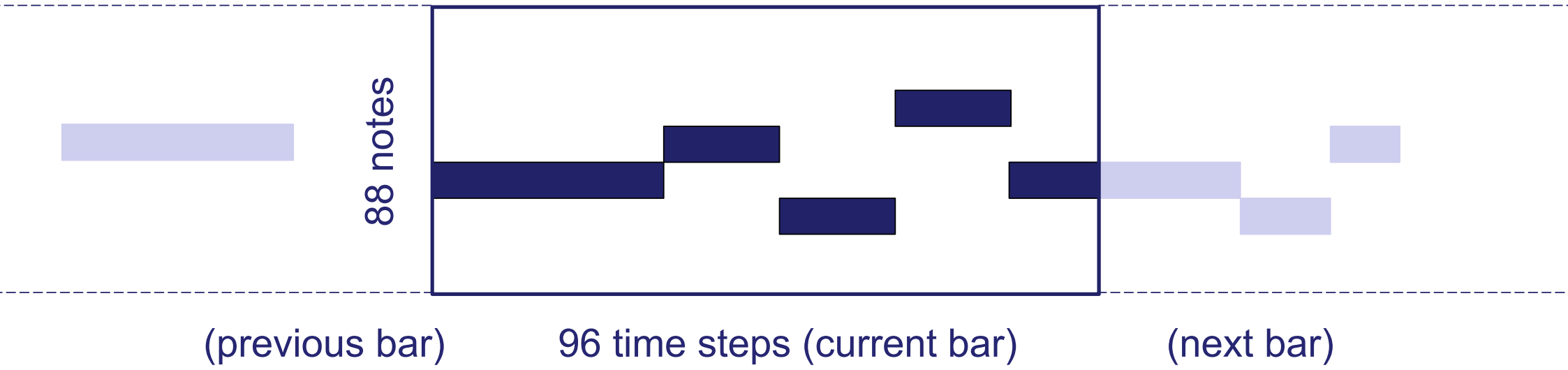
(1) Consider MIDI as **image** using the piano roll representation



(image from the internet)

Piano Roll

Represent each bar of a single-track MIDI as a fixed-size matrix (thus an image)

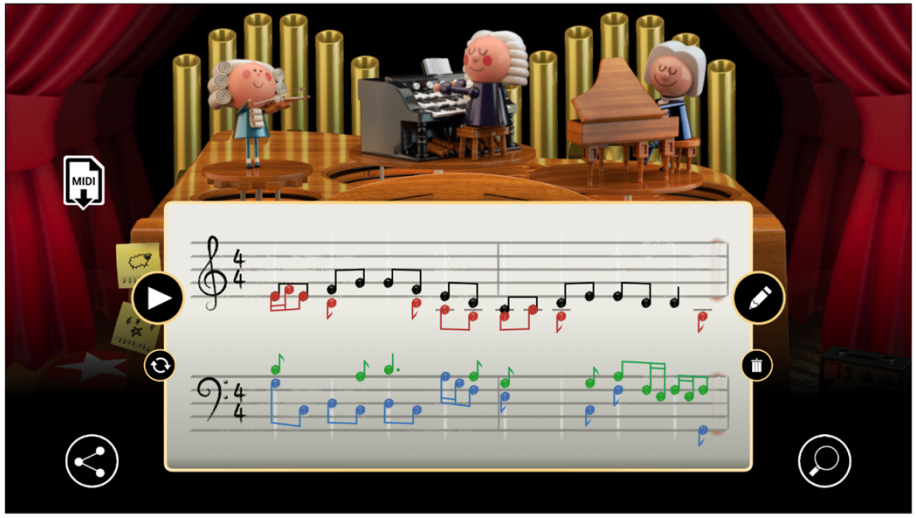


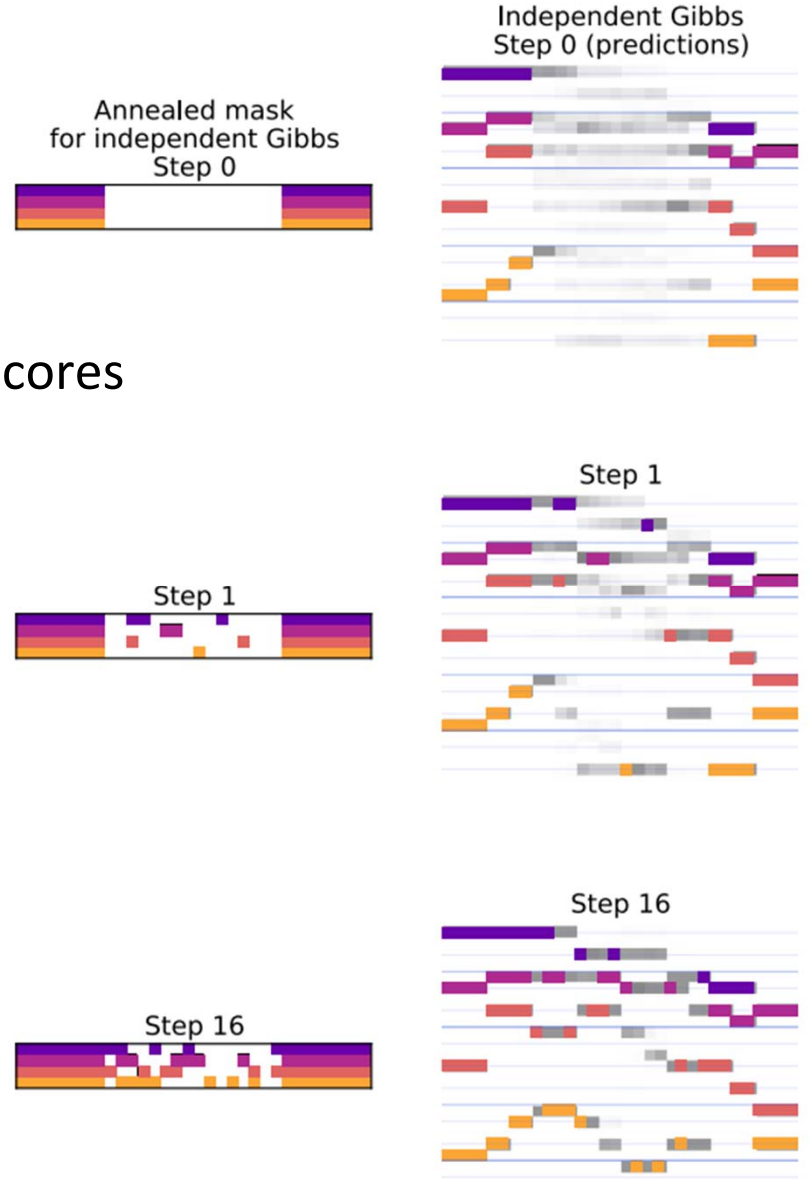
Ref: Dong et al, “Pypianoroll: Open source Python package for handling multitrack pianorolls”, ISMIR-LBD 2018

Coconet & Bach Doodle: Non-GAN CNN-based Approach

https://magenta.tensorflow.org/coconet

- For 2-bar piano roll "**infilling**": to complete *partial* scores
- Blocked-Gibbs sampling (non-autoregressive)



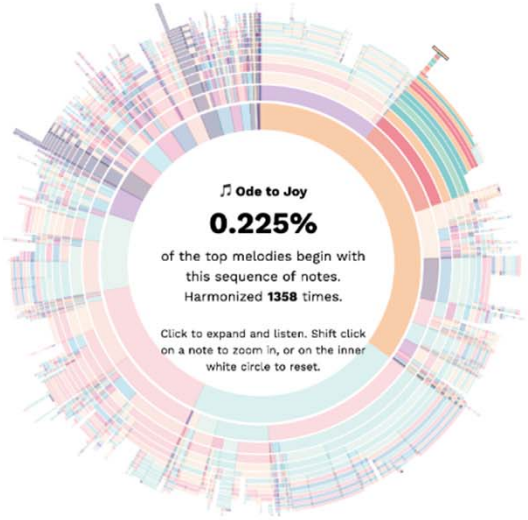


Ref: Huang et al, "Counterpoint by convolution", ISMIR 2017

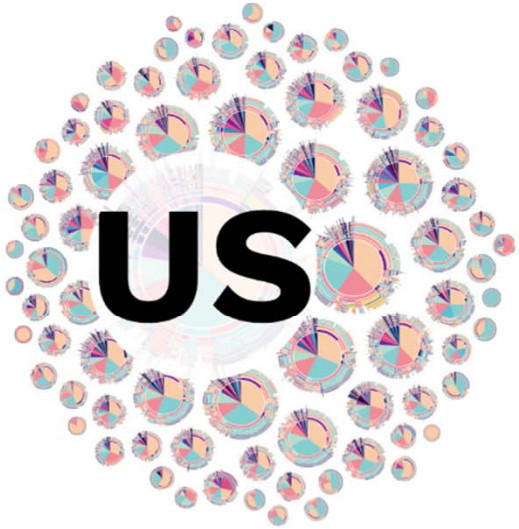
Coconet & Bach Doodle:
Non-GAN CNN-based Approach

https://magenta.tensorflow.org/datasets/bach-doodle

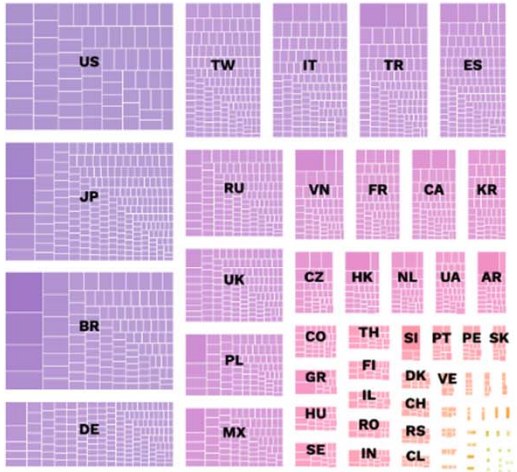
- "In three days, people spent 350 years worth of time playing with the Bach Doodle, and Coconet received more than 55 million queries"



Top overall repeated melodies



Top repeated melodies per country

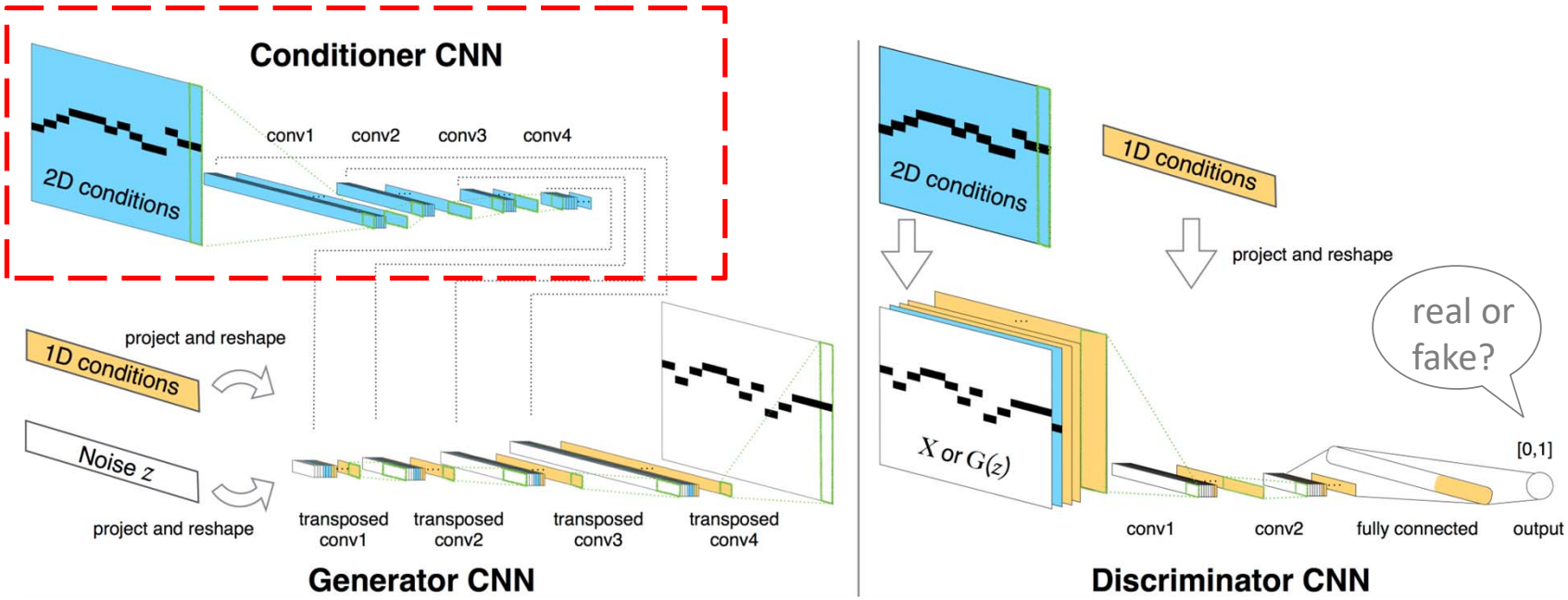


Unique regional hits

Ref: Huang et al, "The Bach Doodle: Approachable music composition with machine learning at scale", ISMIR 2019

MidiNet [yang17ismir]

- Convolutional GAN for **one-bar melody** generation
  - Turn a random vector into a piano-roll like matrix (non-autoregressive note-wise)
  - Conditioned on the previous bar (2D condition), or on the chord (1D condition)

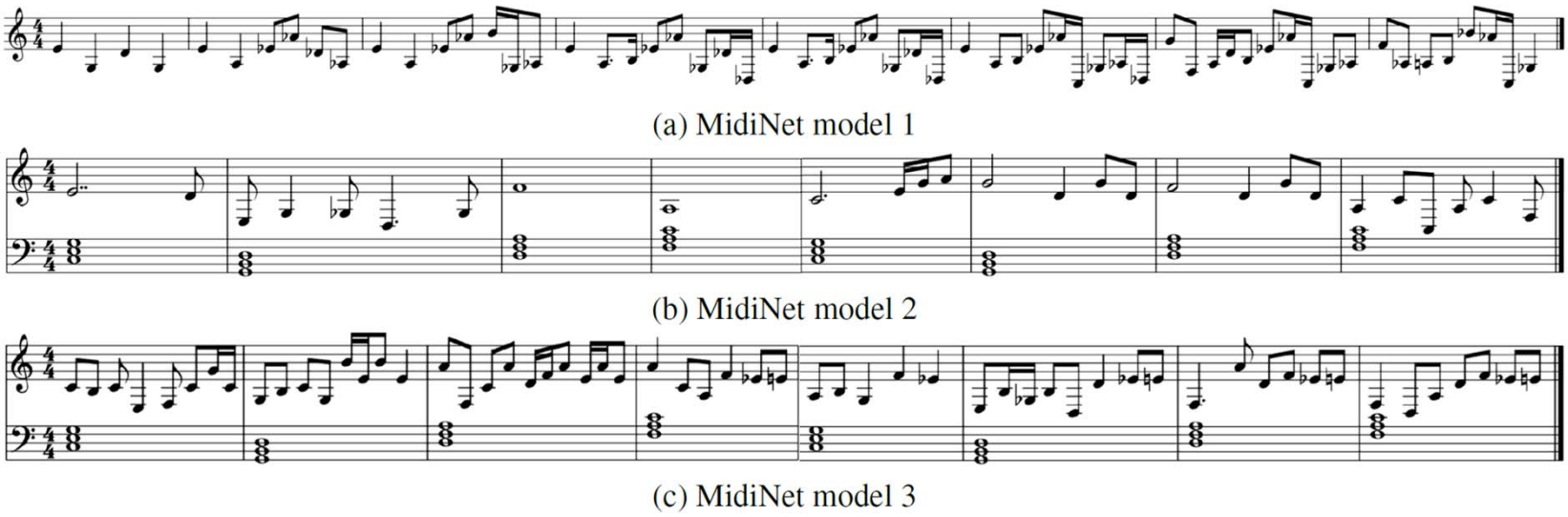


Ref: Yang et al, "MidiNet: A convolutional generative adversarial network for symbolic-domain music generation", ISMIR 2017

MidiNet: Examples

https://richardyang40148.github.io/TheBlog/midinet_arxiv_demo.html

- Variants of MidiNet 1 2 3



(a) MidiNet model 1

(b) MidiNet model 2

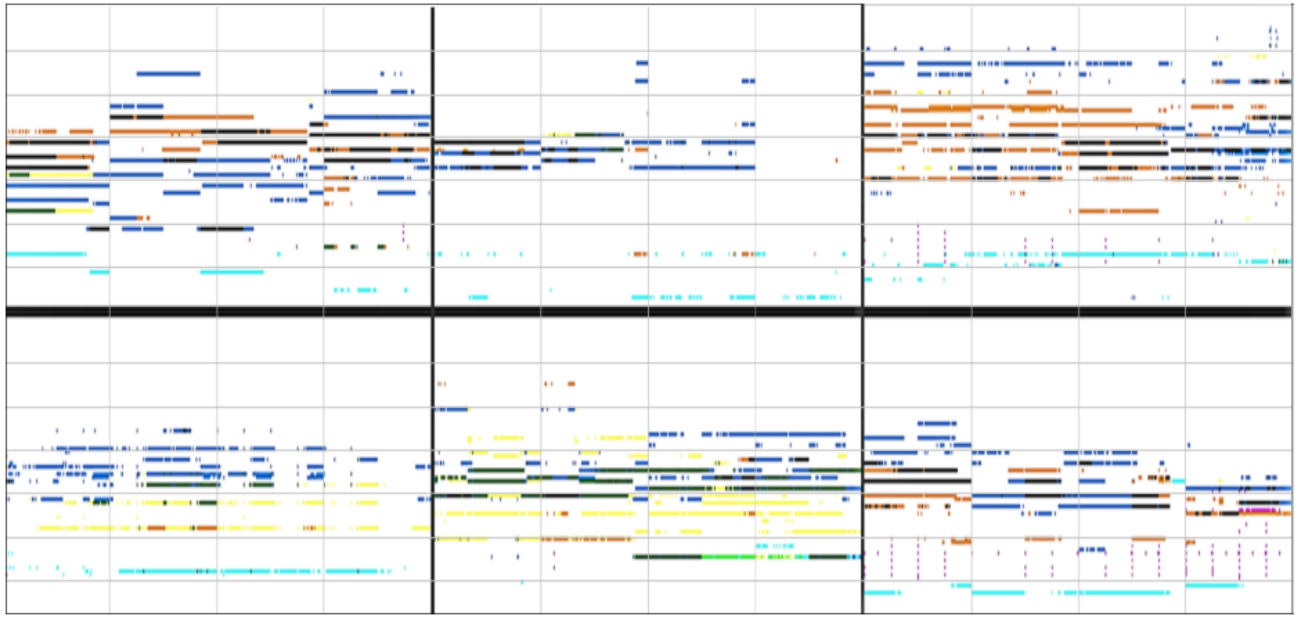
(c) MidiNet model 3

- With drums

Ref: Yang et al, "MidiNet: A convolutional generative adversarial network for symbolic-domain music generation", ISMIR 2017

Multi-track Piano Roll

- Represent difference voices (tracks) in different channels of a fixed-size tensor

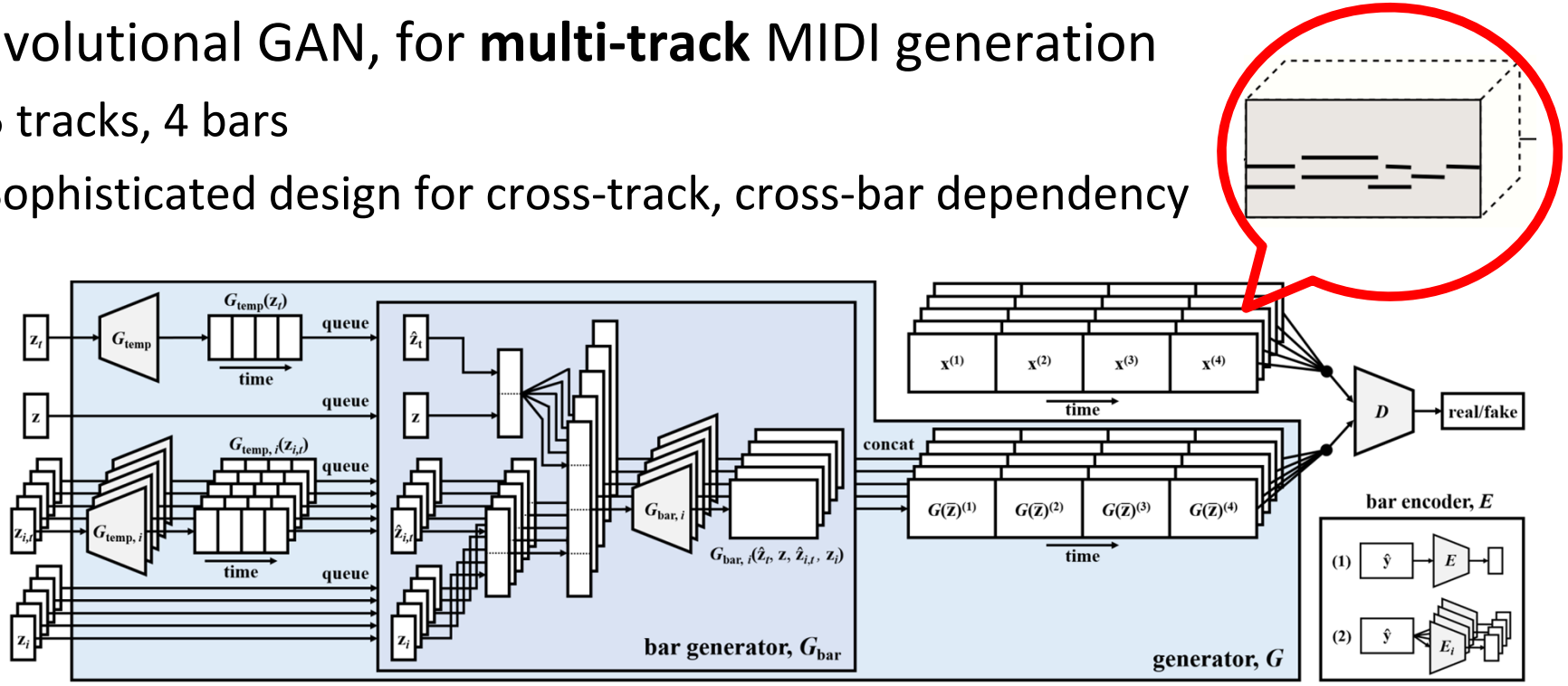


(different colors represent different tracks in this visualization)

MuseGAN [dong18aaai]

https://salu133445.github.io/musegan/

- Convolutional GAN, for **multi-track** MIDI generation
  - 5 tracks, 4 bars
  - Sophisticated design for cross-track, cross-bar dependency

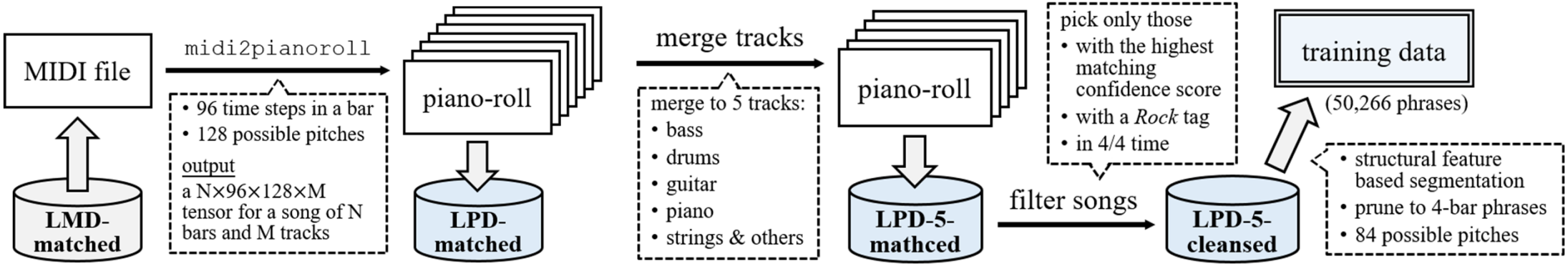


Ref: Dong et al, "MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment", AAAI 2018

MuseGAN: Data

https://salu133445.github.io/musegan/dataset

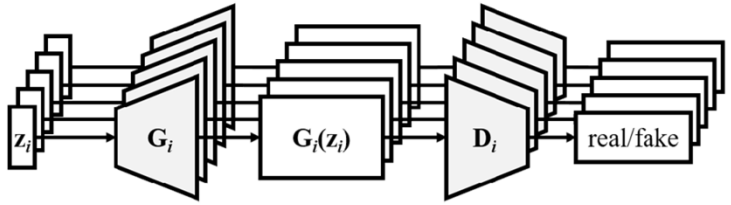
- LPD dataset: **128K MIDIs (piano-rolls)** from LMD (http://colinraffel.com/projects/lmd/)



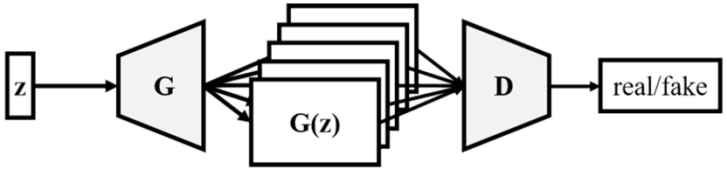
Ref: Dong et al, "MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment", AAAI 2018

MuseGAN: Intra- & Inter-tracks

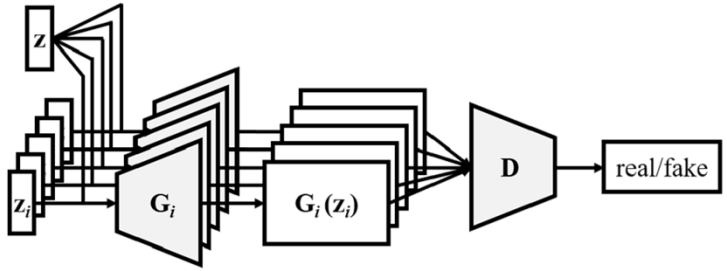
- Multi-track
  - Piano, guitar, bass, strings, drums
- **Hybrid model**
  - One "**shared**" (inter) z
  - Five "**private**" (intra) z_i
  - **Five** generators
  - **One** discriminator



(a) the jamming model



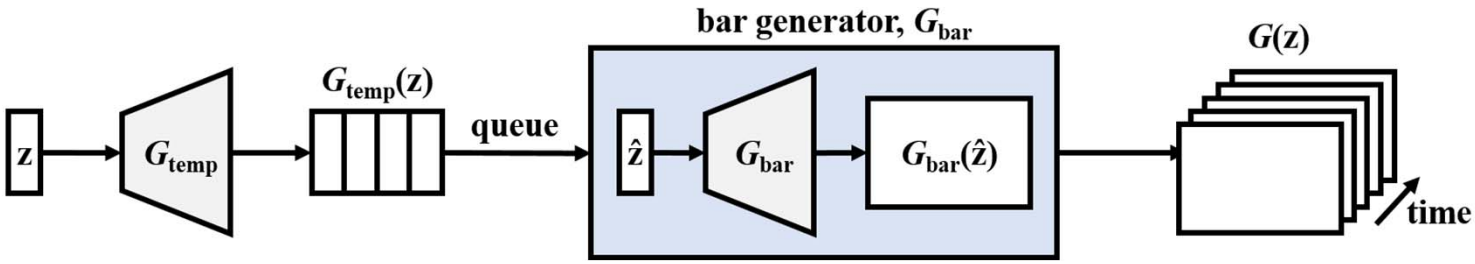
(b) the composer model



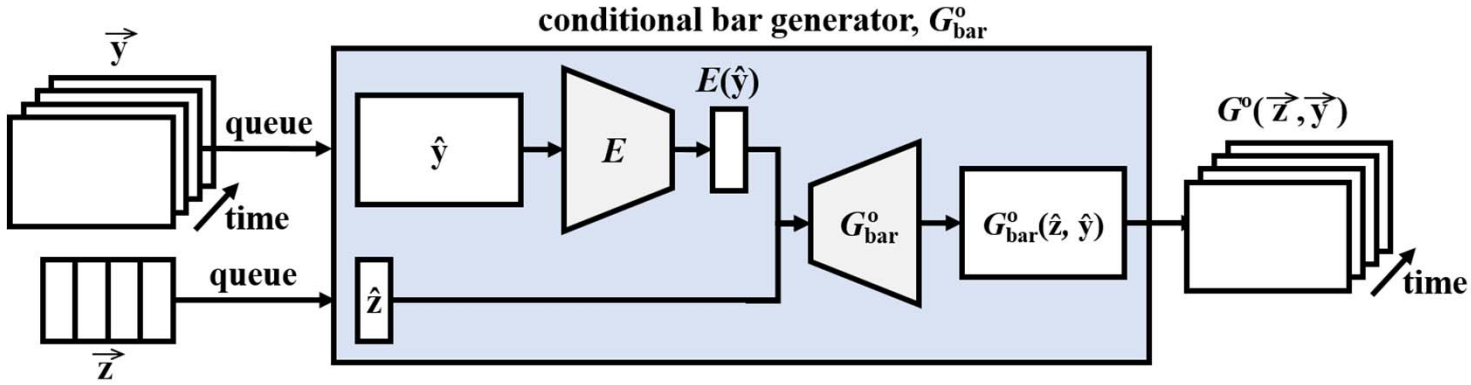
(c) the hybrid model

Ref: Dong et al, "MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment", AAAI 2018

MuseGAN: Temporal Model



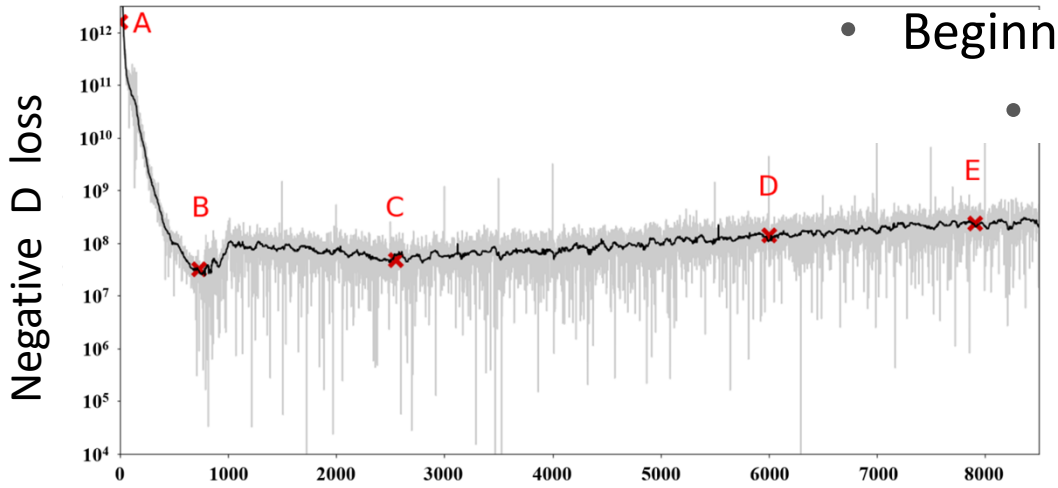
(a) generation from scratch



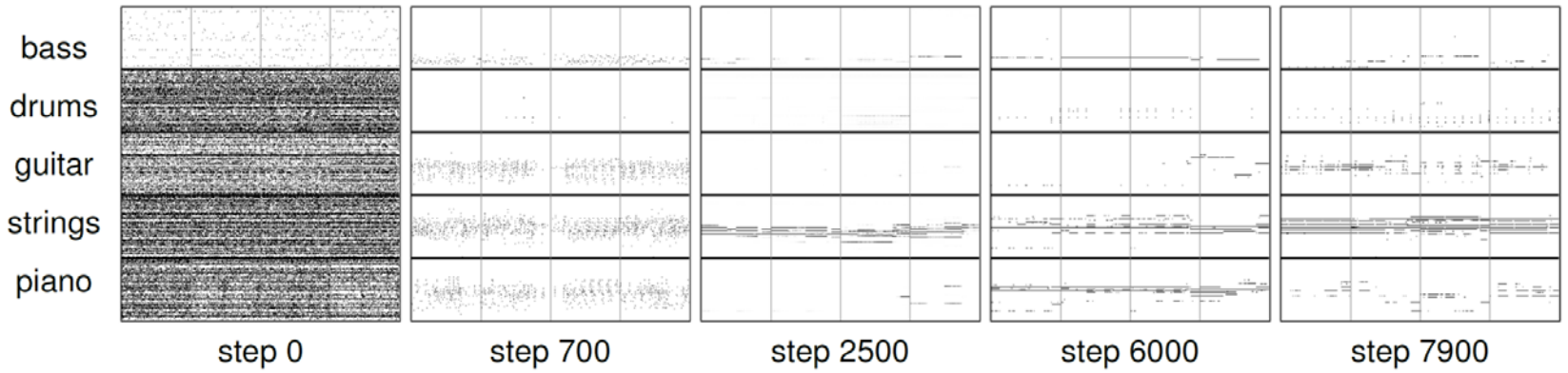
(b) track-conditional generation

Ref: Dong et al, "MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment", AAAI 2018

MuseGAN: WGAN-gp



- Beginning – 500:1 D/G updates
- Later – 5:1 D/G updates
- Training time: <24 hours



Ref: Dong et al, "MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment", AAAI 2018

MuseGAN: G & D

Input: $\mathbf{z} \in \Re^{128}$					
reshaped to $(1, 1) \times 128$ channels					
transconv	1024	2×1	$(2, 1)$	BN	ReLU
transconv	256	2×1	$(2, 1)$	BN	ReLU
transconv	256	2×1	$(2, 1)$	BN	ReLU
transconv	256	2×1	$(2, 1)$	BN	ReLU
transconv	128	3×1	$(3, 1)$	BN	ReLU
transconv	64	1×7	$(1, 7)$	BN	ReLU
transconv	K_{bar}	1×12	$(1, 12)$	BN	tanh
Output: $G_{\text{bar}}(\mathbf{z}) \in \Re^{96 \times 84 \times K_{\text{bar}}}$ (K_{bar}-track piano-roll)					

(b) the bar generator G_{bar}

- Grow **time steps** first
  - 1 → 2 → 4 → 8 → 16 → 32 → 96
- Then **notes** (freq)
  - octave (7)
  - then, pitch (84)

Input: $\tilde{\mathbf{x}} \in \Re^{4 \times 96 \times 84 \times 5}$ (real/fake piano-rolls of 5 tracks)				
reshaped to $(4, 96, 84) \times 5$ channels				
conv	128	$2 \times 1 \times 1$	$(1, 1, 1)$	LReLU
conv	128	$3 \times 1 \times 1$	$(1, 1, 1)$	LReLU
conv	128	$1 \times 1 \times 12$	$(1, 1, 12)$	LReLU
conv	128	$1 \times 1 \times 7$	$(1, 1, 7)$	LReLU
conv	128	$1 \times 2 \times 1$	$(1, 2, 1)$	LReLU
conv	128	$1 \times 2 \times 1$	$(1, 2, 1)$	LReLU
conv	256	$1 \times 4 \times 1$	$(1, 2, 1)$	LReLU
conv	512	$1 \times 3 \times 1$	$(1, 2, 1)$	LReLU
fully-connected	1024			LReLU
fully-connected	1			
Output: $D(\tilde{\mathbf{x}}) \in \Re$				

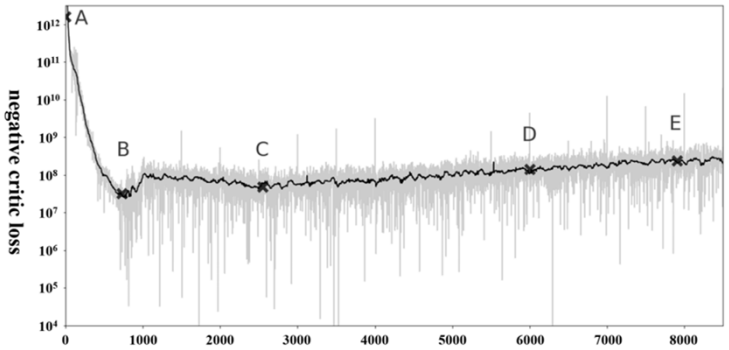
(c) the discriminator D

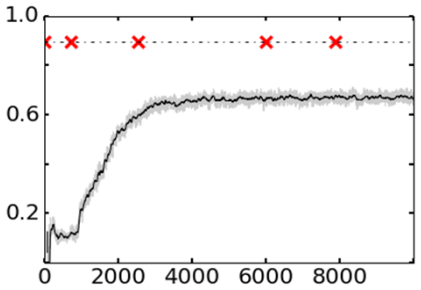
Ref: Dong et al, "MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment", AAAI 2018

MuseGAN: Objective Metrics

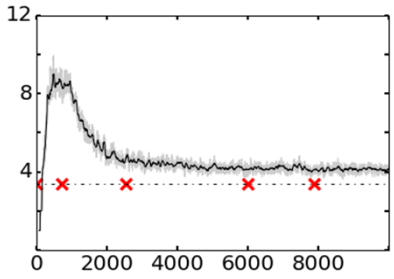
(implemented in https://github.com/salu133445/muspy)

- EB: **ratio of empty bars** (in %)
- UPC: **number of used pitch classes per bar** (from 0 to 12)
- QN: **ratio of "qualified" notes** (in %); we consider a note no shorter than three time steps (i.e. a 32th note) as a qualified note; QN shows if the music is overly fragmented
- DP, or **drum pattern**: ratio of notes in 8- or 16-beat patterns, common ones for Rock songs in 4/4 time
- TD: or **tonal distance**; the hamornicity between a pair of tracks; larger TD implies weaker inter-track harmonic relations



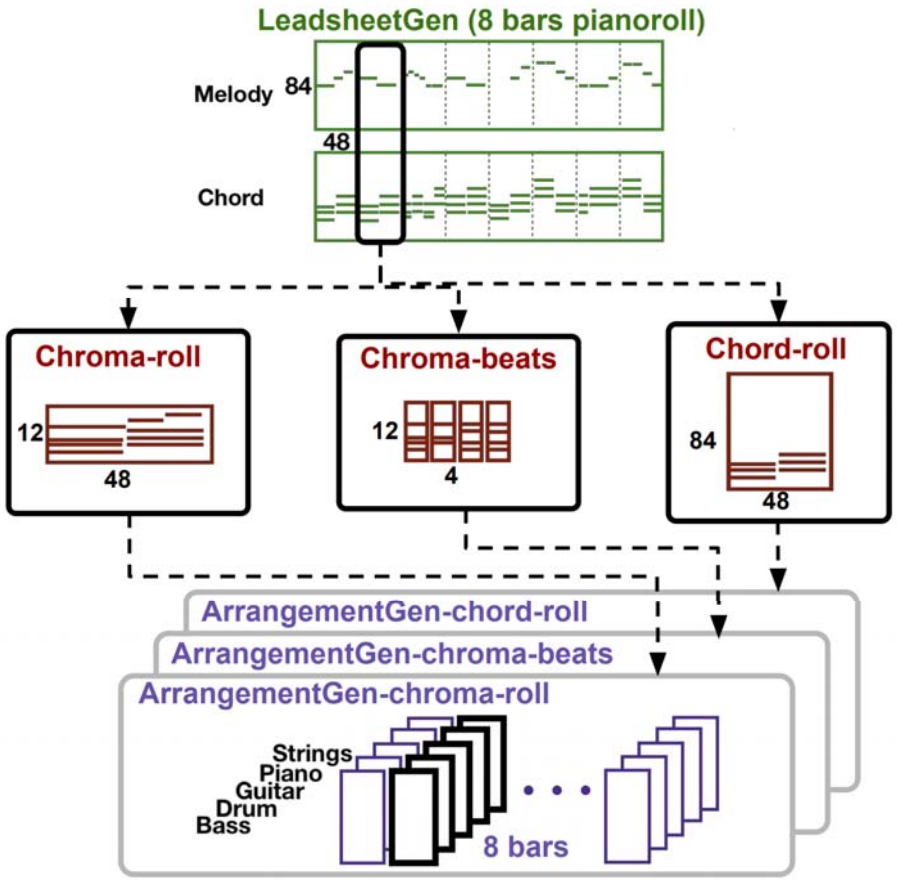


UPC

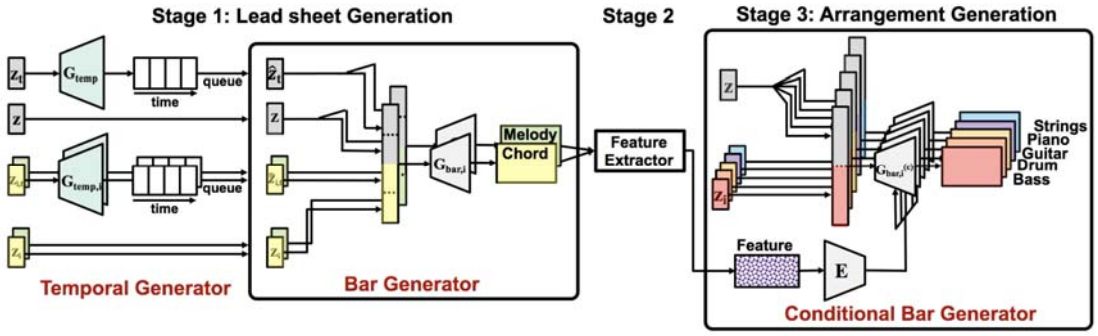


QN

LeadsheetGAN [liu18icmla]

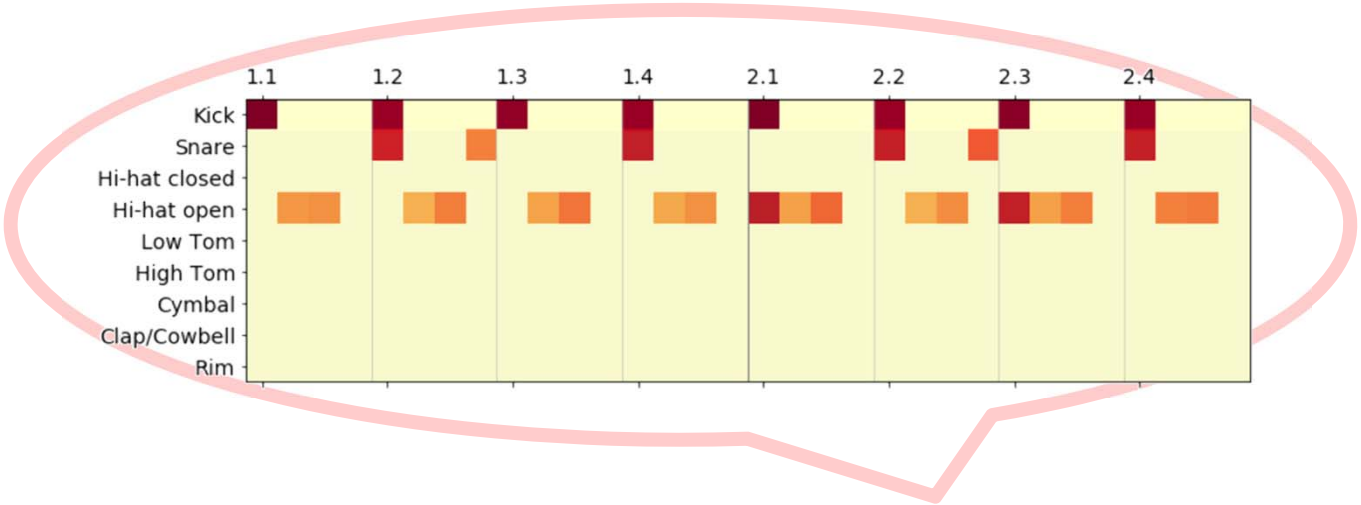


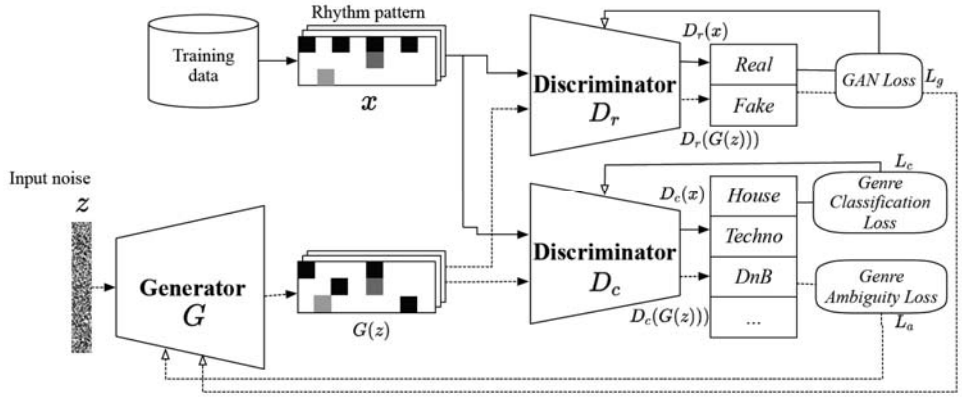
- Extension of MuseGAN
- Generate the *lead sheet* **first**
- Use features extracted from the lead sheet as condition to generate the multi-track arrangement



Ref: Liu & Yang, “Lead sheet generation and arrangement by conditional generative adversarial network”, ICMLA 2018

RhythmCAN [tokui20arxiv]





Tokui, “Can GAN originate new electronic dance music genres?—Generating novel rhythm patterns using GAN with genre ambiguity loss”, arXiv 2020

Two Main Approaches

(1) Consider MIDI as **image** using the piano roll representation

- Works better for generating **patterns** (and spectrograms)
- **Less effective for modeling long sequences**

Outline

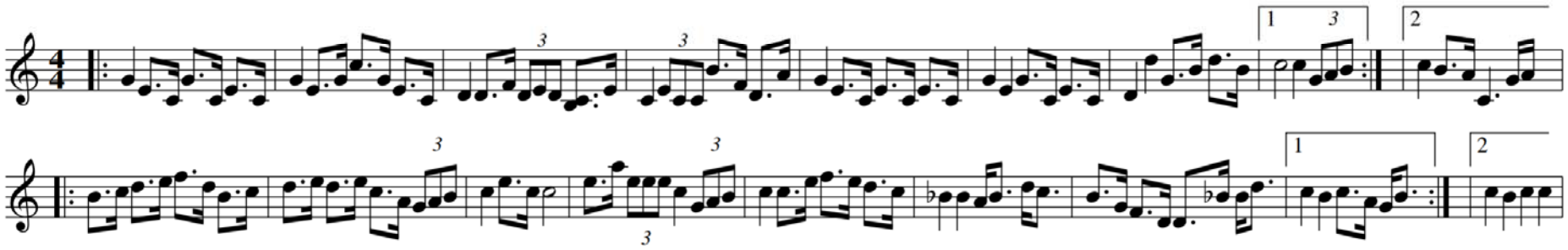
- General ideas
- Image-based approach for symbolic music generation
- **Text-based approach for symbolic music generation**
  - **RNN**
  - Transformer
- Token design
- Building your first MIDI Transformer
- Advanced models

Two Main Approaches

(2) Consider music as **text**

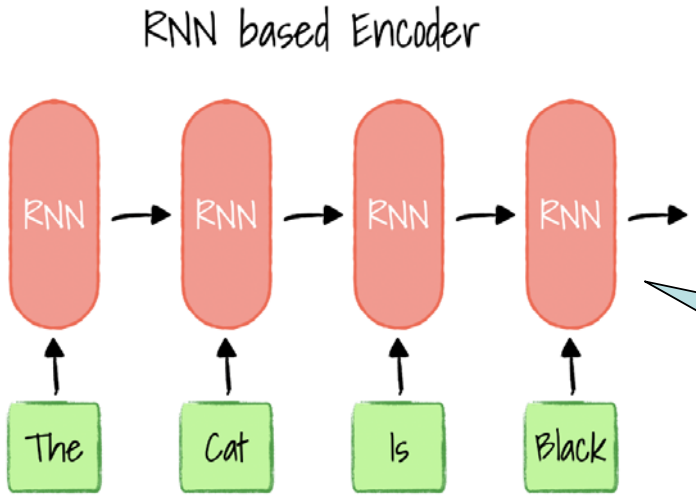
- Use “tokens” to represents “events” in music
- Becomes a “natural language generation” (NLG) problem
  - e.g., Folk RNN [sturm15ismir-lbd]

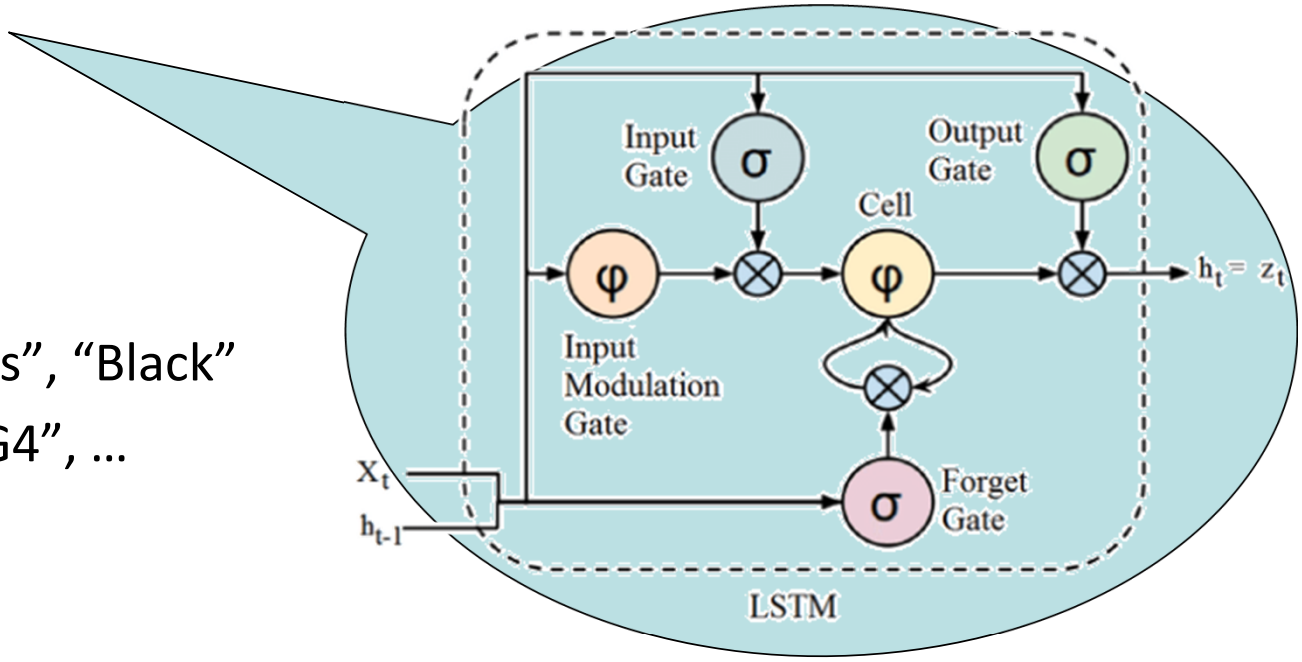
```
M:4/4
K:Cmaj
|: G 2 E > C G > C E > C | G 2 E > G c > G E > C | D 2 D > F (3 D E D [ B, C ] > E | C 2 (3 E C C B > F D > A |
G 2 E > C E > C E > C | G 2 E 2 G > C E > C | D 2 d 2 G > B d > B |1 c 4 c 2 (3 G A B :| |2 c 2 B > A C 3 G /2 A /2
|: B > c d > e f > d B > c | d > e d > e c > A (3 G A B | c 2 e > c c 4 | e > a (3 e e e c 2 (3 G A B |
c 2 c > e f > e d > c | _B 2 B 2 A < B d < c | B > G F > D D > _B B < d |1 c 2 B 2 c > A G < B :| |2 c 2 B 2 c 2 c 2 |
```



Ref: Sturm et al, “Folk music style modelling by recurrent neural networks with long short term memory units,” ISMIR-LBD 2015

RNNs





- Text tokens: “The”, “Cat”, “Is”, “Black”
- Music tokens: “C4”, “E4”, “G4”, …

RNN-based Symbolic Music Generation

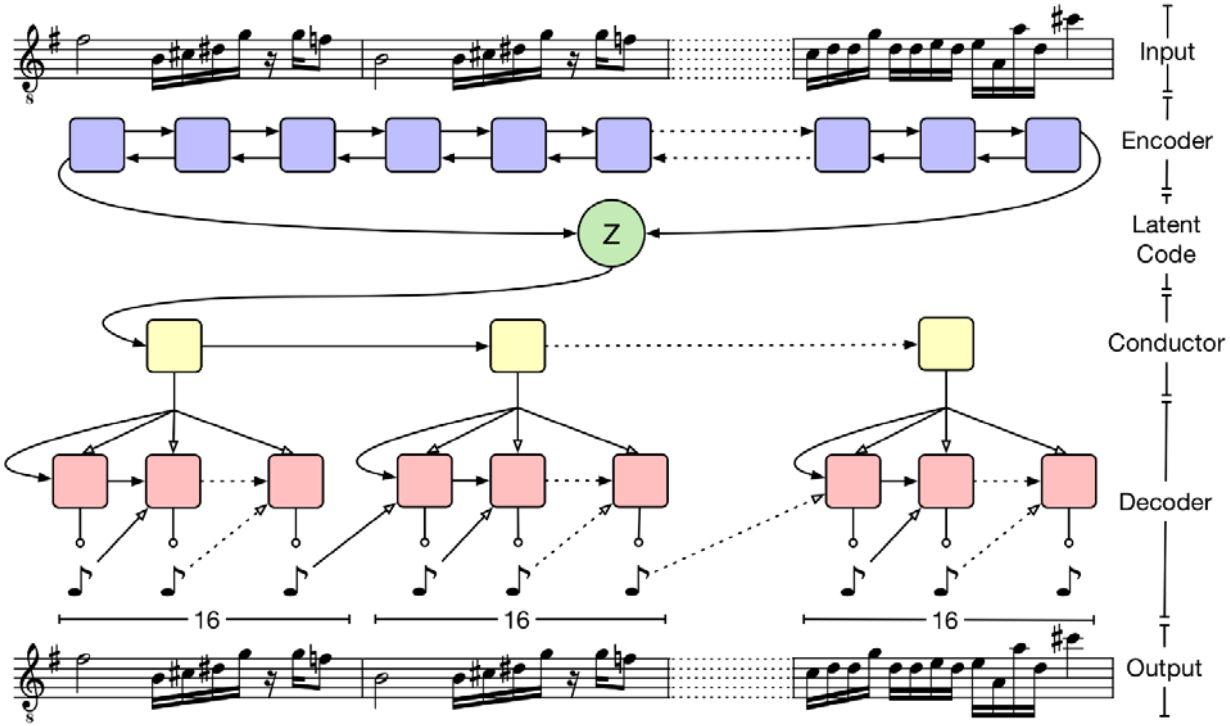
- Used to be the state-of-the-art
- Good for not-very-long sequences (e.g., 16 bars)

- Examples
  - **MelodyRNN** (2016): https://magenta.tensorflow.org/2016/06/10/recurrent-neural-network-generation-tutorial
  - **C-RNN-GAN** (2016)
  - **Song From PI** (2017)
  - **DeepBach** (ICML 2017)

RNN Example: Music VAE [roberts18icml]

https://magenta.tensorflow.org/music-vae

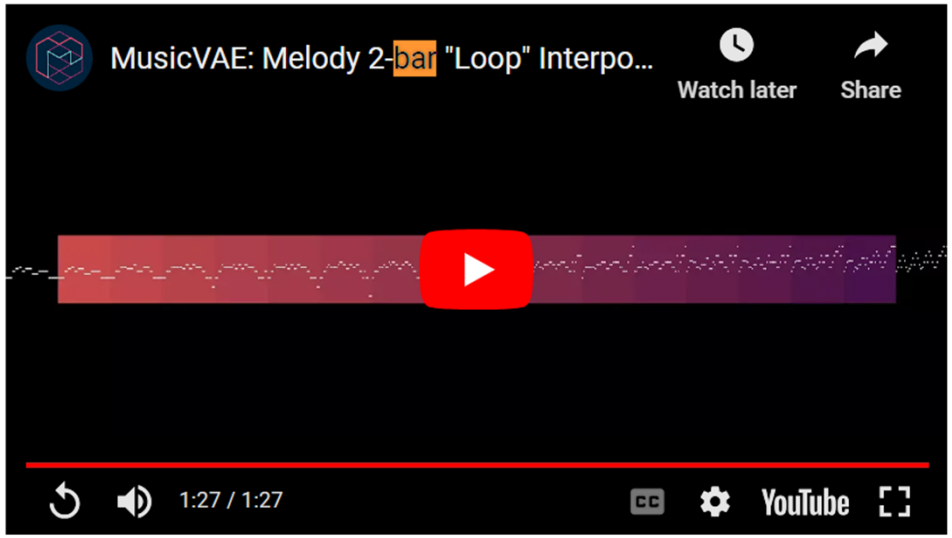
- For **melody** generation
- **Bidirectional encoder**
- **Hierarchical decoder**
  - **Bar-level** conductor
  - **Note-level** decoder
- **Recurrent VAE**
  - Two-layer LSTM for the encoder, conductor, decoder
  - Spherical Gaussian prior for the latent code *z*

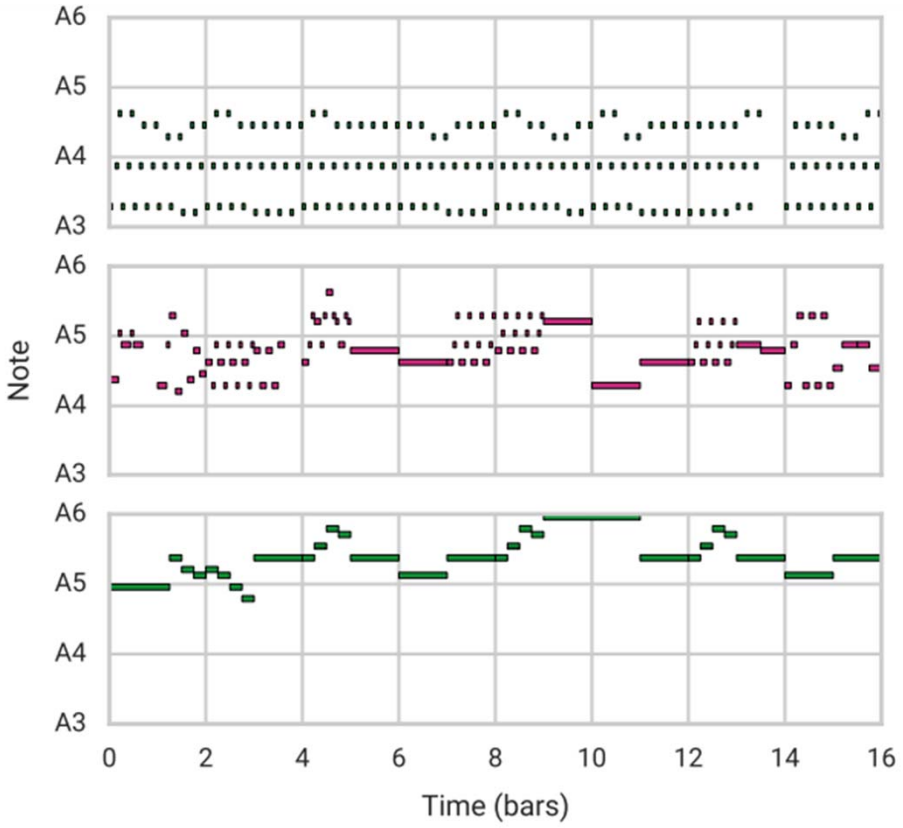


Ref: Roberts et al, "A hierarchical latent vector model for learning long-term structure in music", ICML 2018

Music VAE [roberts18icml]

- Latent space manipulation
  - Attribute vectors
  - **Spherical interpolation**

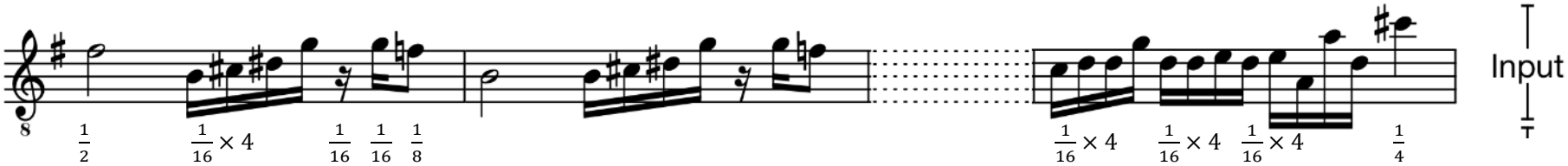




Ref: Roberts et al, "A hierarchical latent vector model for learning long-term structure in music", ICML 2018

Music VAE [roberts18icml]

- Data representation
  - 2- and 16-bar melodies (monophonic note sequences)
  - **130- dimensional output space** (categorical distribution over tokens)
    - 128 "**note-on**" tokens for the 128 MIDI pitches
    - plus single tokens for "**note-off**" and "**rest**"

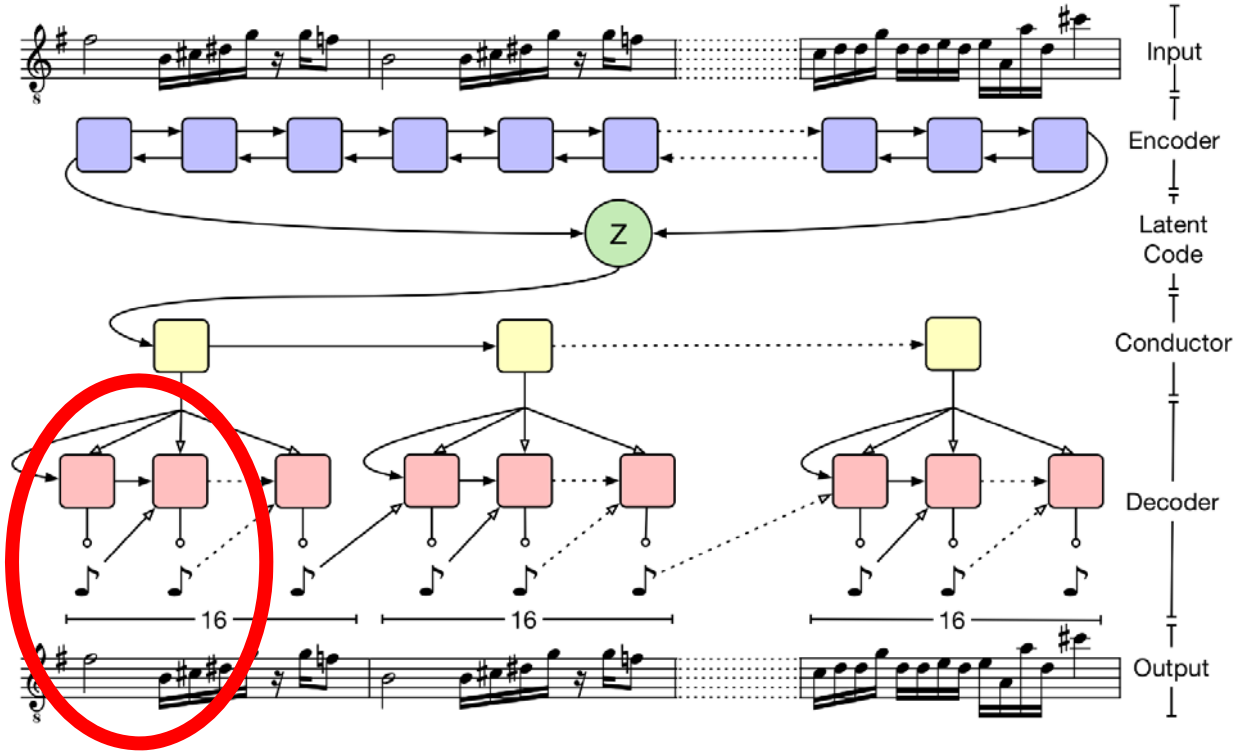


  - Model the melodies as sequences of 16th note events
    - Only consider 4/4 signature—each bar consists of 16 events
    - 2-bar data: sequence length 32
    - 16-bar data: sequence length 256

Ref: Roberts et al, "A hierarchical latent vector model for learning long-term structure in music", ICML 2018

Music VAE [roberts18icml]

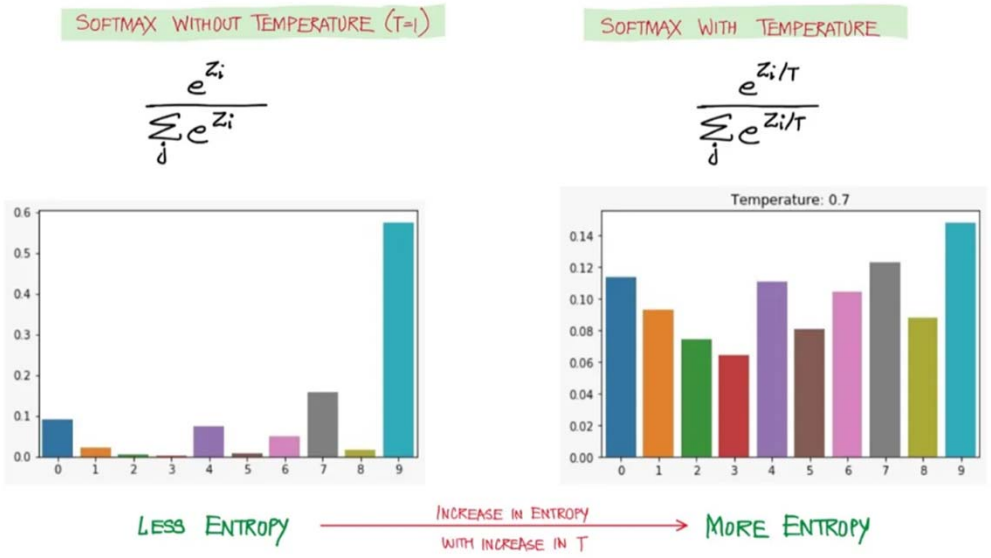
- Training
  - **Cross-entropy loss**
    - For each time step (token)
  - **Teacher forcing**
    - During *training* time, use observed sequence values (i.e. ground-truth samples) back into the RNN after each step
  - **Scheduled sampling**
    - To reduce "**exposure bias**," the discrepancy between training time and inference time sampling

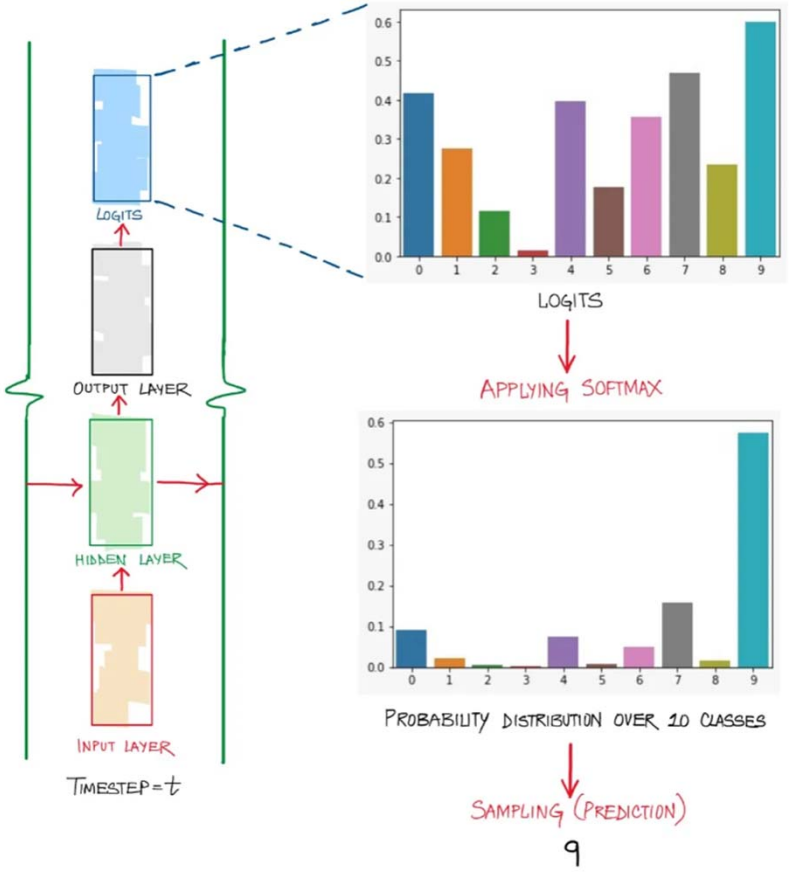


Ref: Roberts et al, "A hierarchical latent vector model for learning long-term structure in music", ICML 2018

Music VAE [roberts18icml]

- Inference
  - **Softmax temperature**
  - https://medium.com/mlearning-ai/softmax-temperature-5492e4007f71



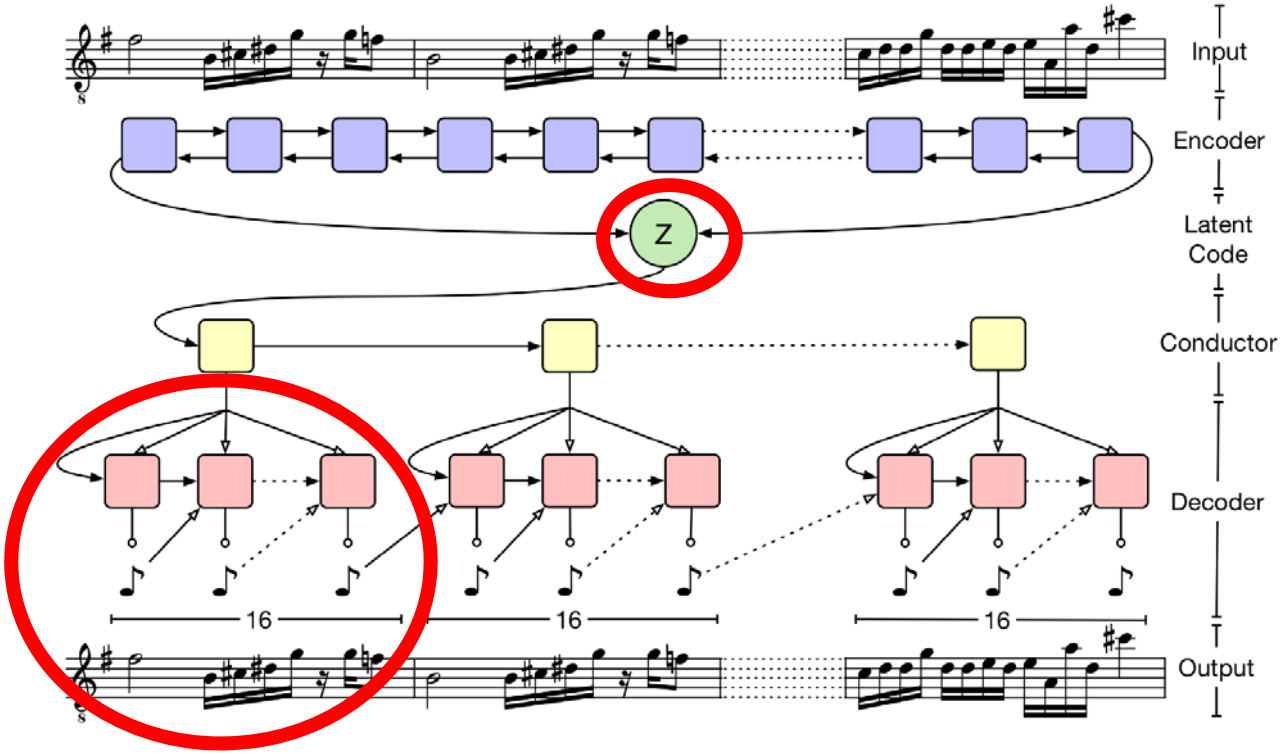


Training/Inference for Sequence Models (RNNs, Transformers)

- **Training**
  - Output a **softmax** distribution (sum-to-one)
  - Compare that softmax distribution (continuous-valued) with the groundtruth one-hot distribution (discrete) using **cross entropy**
  - *Teacher forcing*: **Do NOT really sample from that distribution**, but use the **groundtruth token** as the input for the next time step
- **Inference**
  - Output a **softmax** distribution (sum-to-one)
  - **Do sampling**
    - There are many sampling methods
    - May apply temperature
  - Use the **sampled token** as the input for the next time step
  - *Exposure bias:* The discrepancy between training time and inference time sampling

Music VAE [roberts18icml]

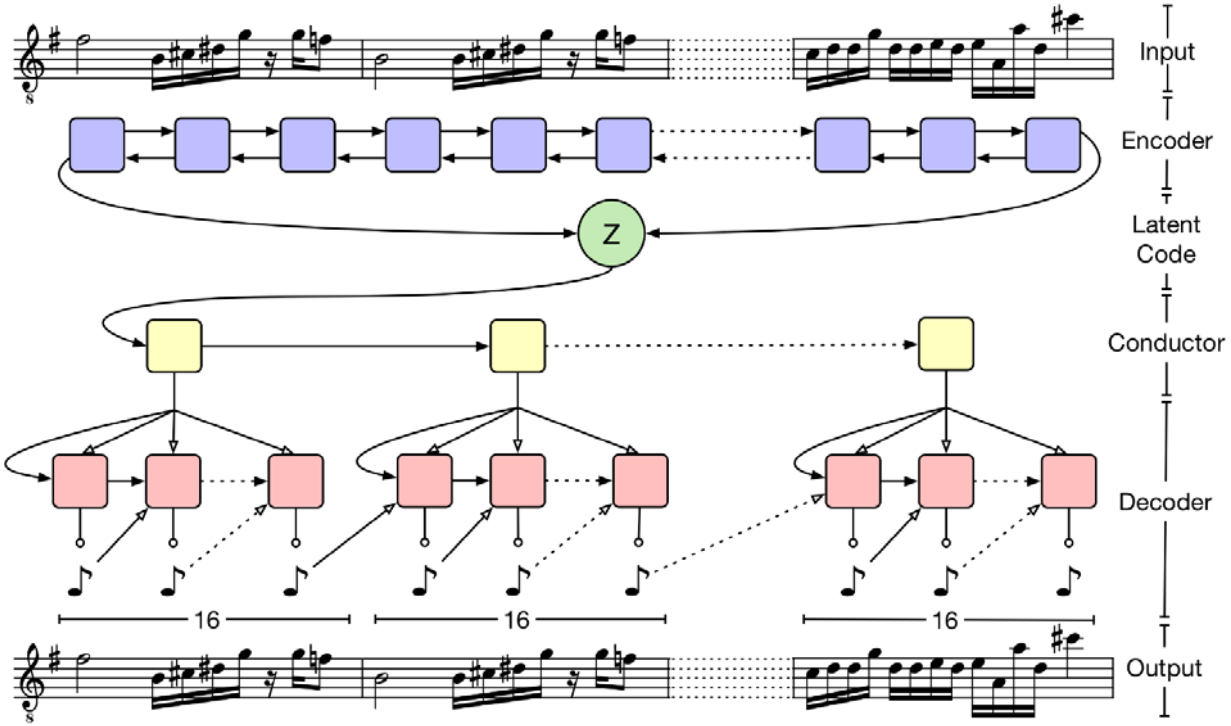
- Training
  - Cross-entropy loss
  - Teacher forcing
  - Avoid **“posterior collapse”**
    - Where the model learns to ignore the latent state
    - Reduce the effective scope of the decoder RNN by only allowing it to propagate state within an output subsequence
  - VAE training techniques: ***β*-VAE and *free bits***



Ref: Roberts et al, “A hierarchical latent vector model for learning long-term structure in music”, ICML 2018

Music VAE [roberts18icml]

- For **melody** generation
- For **bass** generation
- For **drum** generation
  - "For drum patterns, we mapped the 61 drum classes defined by the General MIDI standard to **9 canonical classes** and represented **all possible combinations of hits with 512 categorical tokens**"



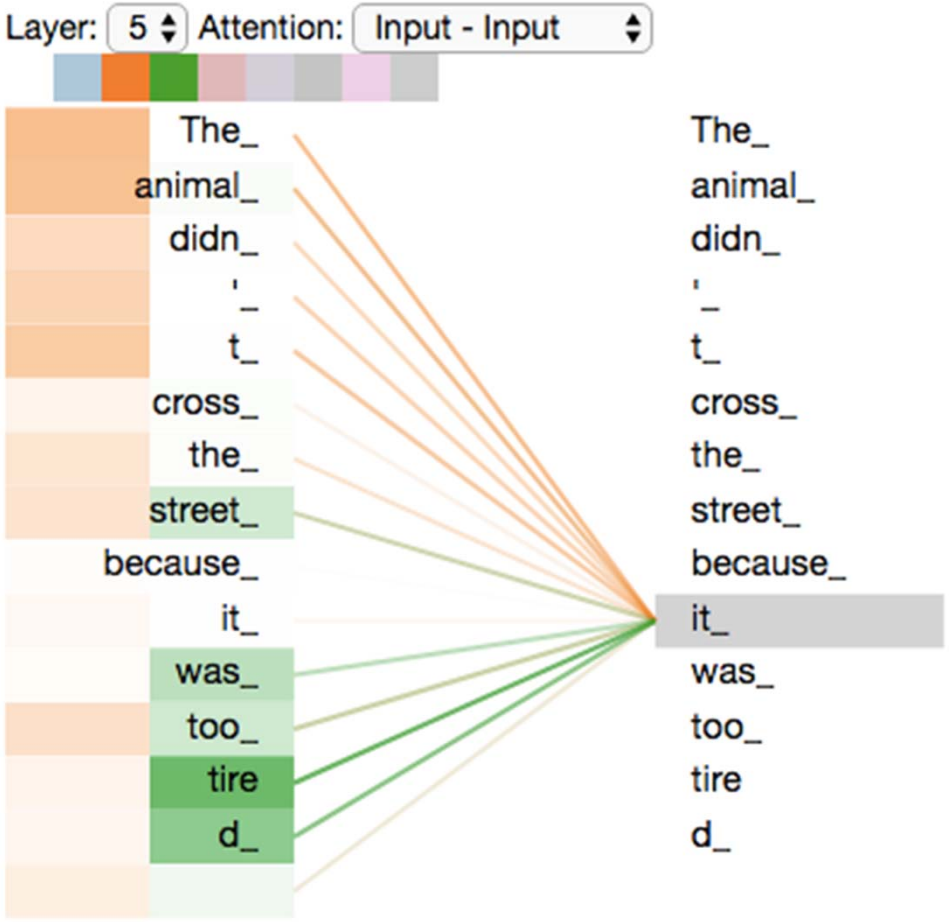
Ref: Roberts et al, "A hierarchical latent vector model for learning long-term structure in music", ICML 2018

Outline

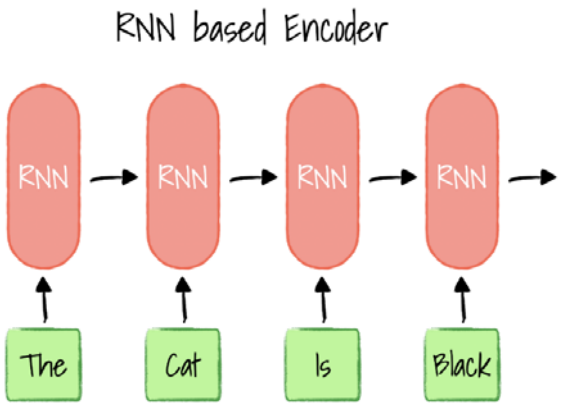
- General ideas
- Image-based approach for symbolic music generation
- Text-based approach for symbolic music generation
  - RNN
  - **Transformer**
- Token design
- Building your first MIDI Transformer
- Advanced models

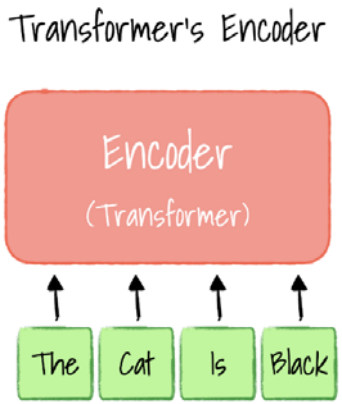
Transformers

- Attention mechanism
  - Maintain a latent vector for ***every*** token in the history
  - Compute the correlation between the input with all the latents in the history
  - Need "**positional encoding**"
- In contrast, there is only **one** latent vector for the entire sequence in RNNs



RNNs vs. Transformers





- **RNNs**
  - use only a latent vector
  - $y_t = f(y_{t-1}, h_{t-1})$

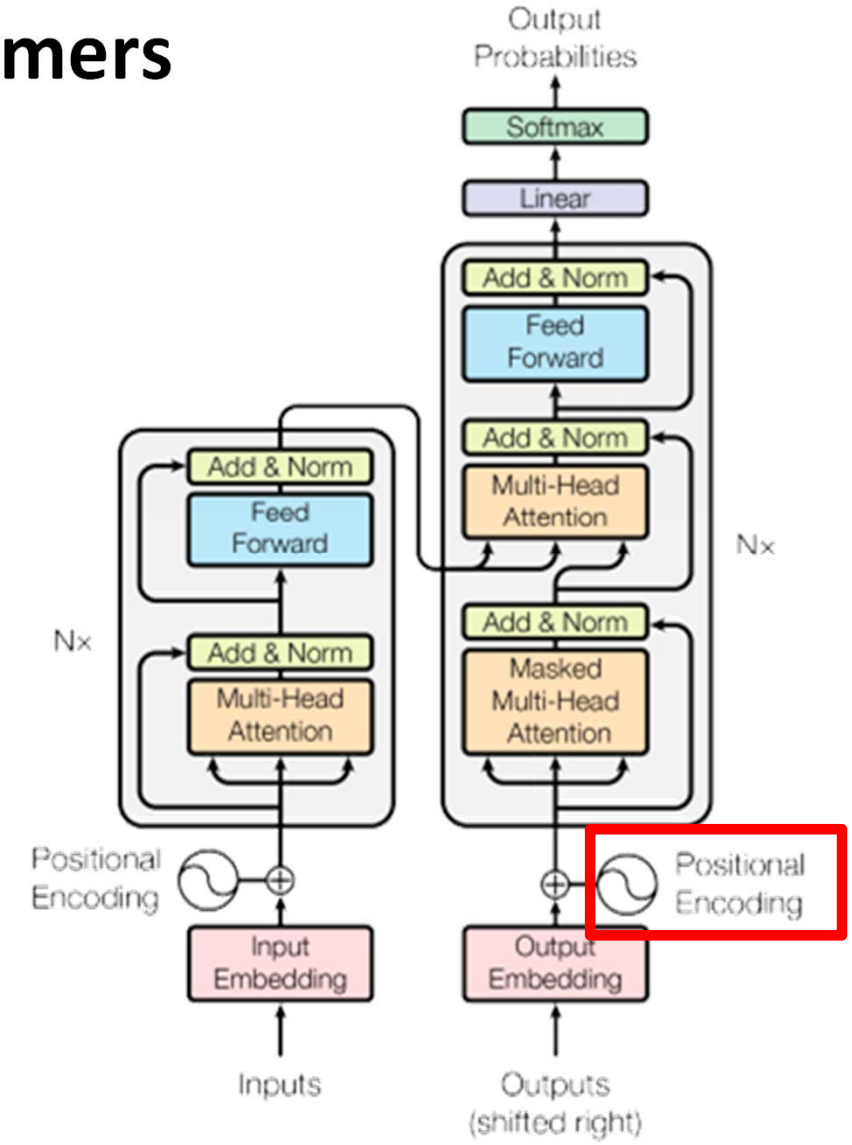
current output — last output — latent representation of the "history"

- **Transformers**
  - multiple latent vectors
  - $y_t = f(y_{t-1}, [h_{t-1}, h_{t-2}, \ldots, h_{t-L}])$

latent representation of t-1 — latent representation of t-L

Transformers

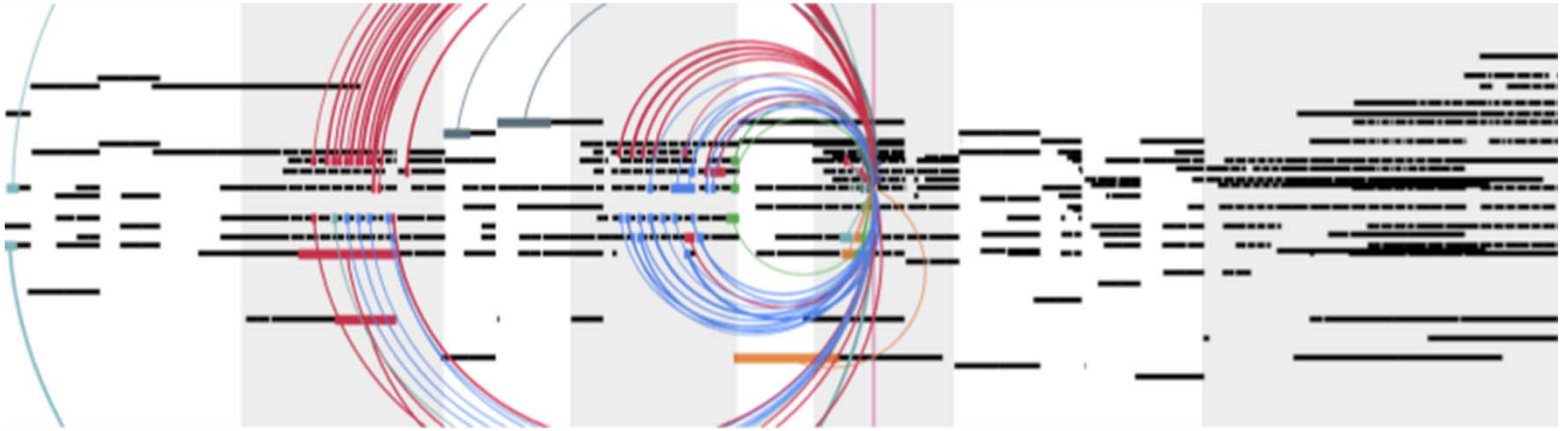
- There are encoders & decoders
  - Encoder: cross-attention
  - Decoder: self-attention
- **Decoder-only** architecture
  - Usually used for unconditional, or prompt-based generation
  - Mainly talk about this one today
- **Encoder/decoder** architecture
  - Usually used for sequence-to-sequence generation



Transformer Example 1: Music Transformer [huang19iclr]

https://magenta.github.io/listen-to-transformer/

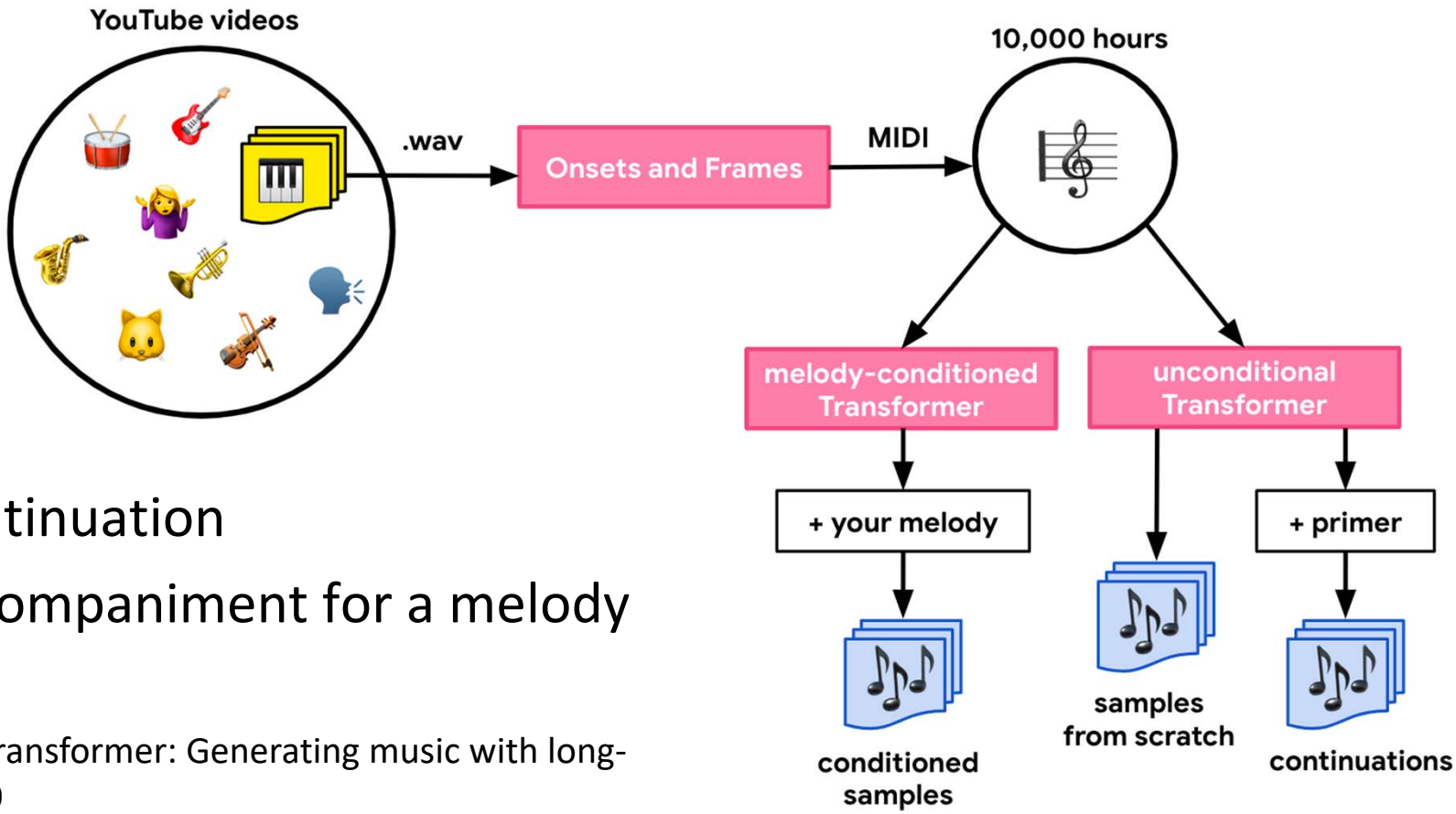
- Generate full piano performances (one note at a time)
  - "… can generate **minute-long** compositions (thousands of steps) with compelling structure, generate continuations that coherently elaborate on a given motif, and in a seq2seq setup generate accompaniments conditioned on melodies."



Ref: Huang et al, "Music Transformer: Generating music with long-term structure", ICLR 2019

Music Transformer [huang19iclr]

https://magenta.tensorflow.org/piano-transformer



- Generate continuation
- Generate accompaniment for a melody

Ref: Huang et al, "Music Transformer: Generating music with long-term structure", ICLR 2019

Transformer Example 2: MuseNet [payne19]

https://openai.com/blog/musenet/

- Generate multi-track MIDIs by a **72-layer** sparse Transformer decoder
  - *Instrument-related* tokens

```
bach piano_strings start tempo90 piano:v72:G1 piano:v72:G2
piano:v72:B4 piano:v72:D4 violin:v80:G4 piano:v72:G4
piano:v72:B5 piano:v72:D5 wait:12 piano:v0:B5 wait:5
piano:v72:D5 wait:12 piano:v0:D5 wait:4 piano:v0:G1 piano:v0:G2
piano:v0:B4 piano:v0:D4 violin:v0:G4 piano:v0:G4 wait:1
piano:v72:G5 wait:12 piano:v0:G5 wait:5 piano:v72:D5 wait:12
piano:v0:D5 wait:5 piano:v72:B5 wait:12
```

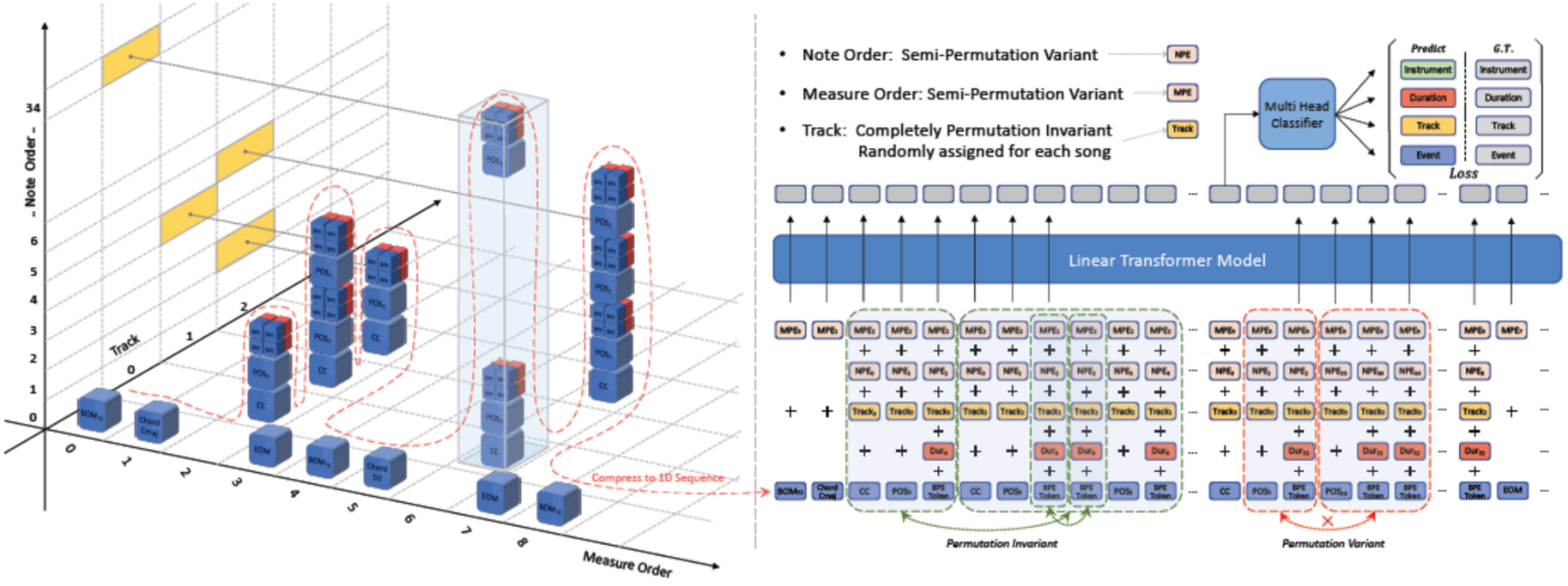
  - Another early multitrack Transformer: LakhNES [donahue19ismir]

Ref 1: Payne et al, “MuseNet,” OpenAI blog, 2019
Ref 2: Donahue et al, “LakhNES: Improving multi-instrumental music generation with cross-domain pre-training”, ISMIR 2019

Transformer Example 3: SymphonyNet [liu22ismir]

https://symphonynet.github.io/

- 12-layer Transformer decoder trained on 46,359 symphonic MIDI music



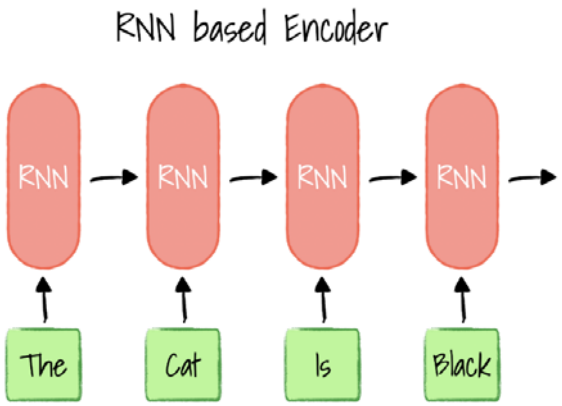
Ref: Liu et al, "Symphony generation with permutation invariant language model", ISMIR 2022

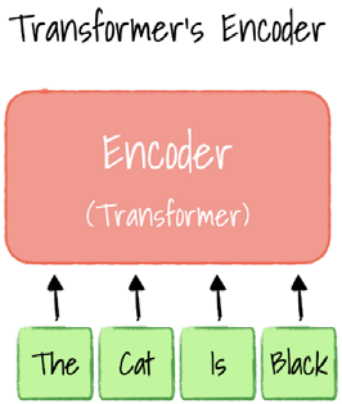
RNNs vs. Transformers

- **RNNs** work well when the sequence is ***not that long***
  - Melodies (e.g., Folk RNN, MusicVAE)
  - 8-bar piano performance (e.g., JazzRNN)

- **Transformers** work better for ***longer*** *sequences*
  - At the expense of computational power
    - An RNN-based model may contain **3** layers or so
    - A Transformer-based model may use up to >**72** layers … (e.g., MuseNet)
    - Usually, a **12**-layer Transformer decoder (with about 40M learnable parameters) works okay
  - Represent **the SOTA** approach for sequence modeling, and for building **LLMs**

RNNs vs. Transformers





- **RNNs**
  - use only a latent vector
  - $y_t = f(y_{t-1}, h_{t-1})$
  - **training: slower**
  - **inference: faster**

- **Transformers**
  - multiple latent vectors
  - $y_t = f(y_{t-1}, [h_{t-1}, h_{t-2}, \ldots, h_{t-L})$
  - **training: faster** (teacher forcing → parallel training)
  - **inference: slower**

Key to the Text-based Approach

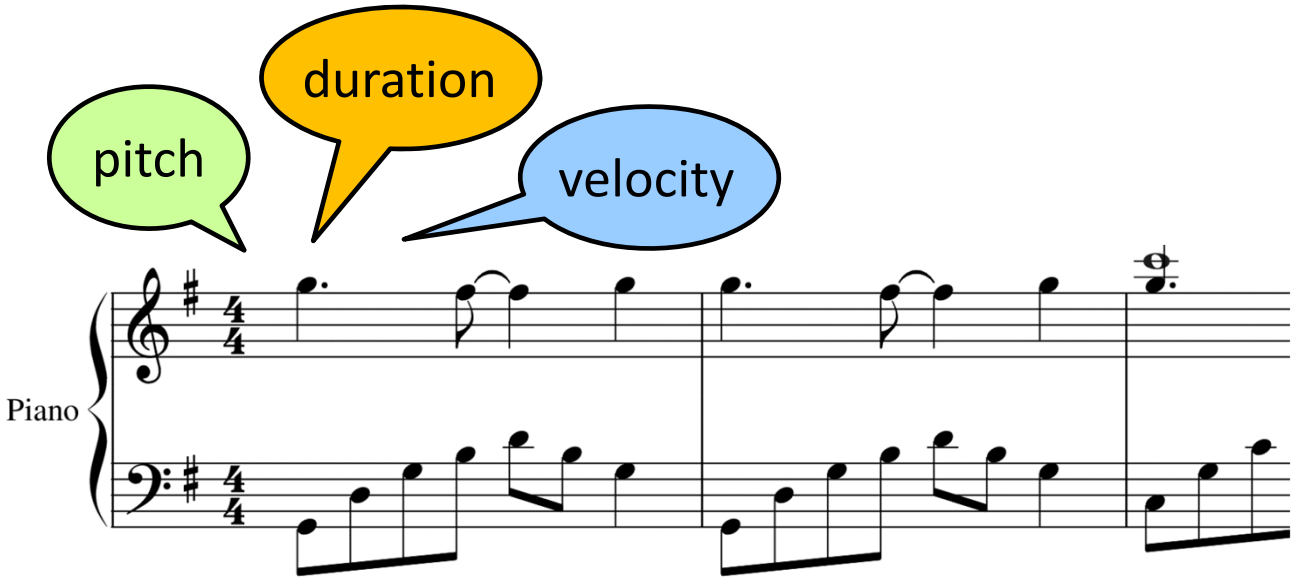
- **Neural sequence model**
  - RNNs, hierarchical RNN, Transformers, state-space models (Mamba)
- **Token representation of music**
  - Token design

Outline

- General ideas
- Image-based approach for symbolic music generation
- Text-based approach for symbolic music generation
- **Token design**
- Building your first MIDI Transformer
- Advanced models

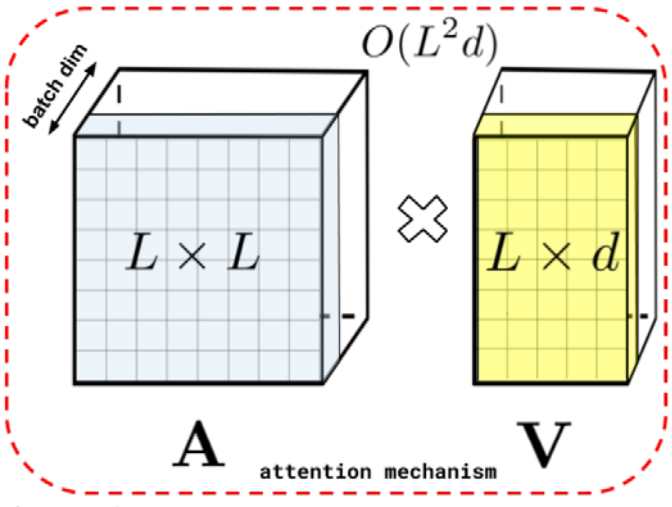
Issue 1: Long Sequence Length of MIDI-like and REMI

- Both MIDI-like and REMI use multiple tokens to represent a musical note; isn't that inefficient?



The Problem: Transformers are Expensive

- **Transformers**
  - multiple latent vectors
  - $y_t = f(y_{t-1}, [h_{t-1}, h_{t-2}, \ldots, h_{t-L})$
  - Memory complexity: **O(L^2)**, where *L* denotes the length of the attention window (i.e., the history) because of the *L* x *L* similarity matrix
  - Need to cut a music pieces into **segments** to fit the VRAM



(Figure from the Performer paper)

The Problem: Transformers are Expensive

	Representation	Model	Attn. window
Music Transformer (Huang et al. 2019)	MIDI-like	Transformer	2,048
MuseNet (Payne 2019)	MIDI-like*	Transformer	4,096
LakhNES (Donahue et al. 2019)	MIDI-like*	Transformer-XL	512
(Choi et al. 2020)	MIDI-like	Transformer	2,048
Pop Music T. (Huang and Yang 2020)	REMI	Transformer-XL	512
Transformer VAE (Jiang et al. 2020)	MIDI-like	Transformer	128
Guitar Transformer (Chen et al. 2020)	REMI*	Transformer-XL	512
Jazz Transformer (Wu and Yang 2020)	REMI*	Transformer-XL	512
MMM (Ens and Pasquier 2020)	MIDI-like*	Transformer	2,048

- Most people use **512-token** attention window before 2020
  - But, a piano cover of a pop song can contain up to **5k** tokens
  - **Segmentation** of music pieces makes it hard to learn **long-term patterns**

Compound Word Transformer [hsiao21aaai]

https://github.com/YatingMusic/compound-word-transformer

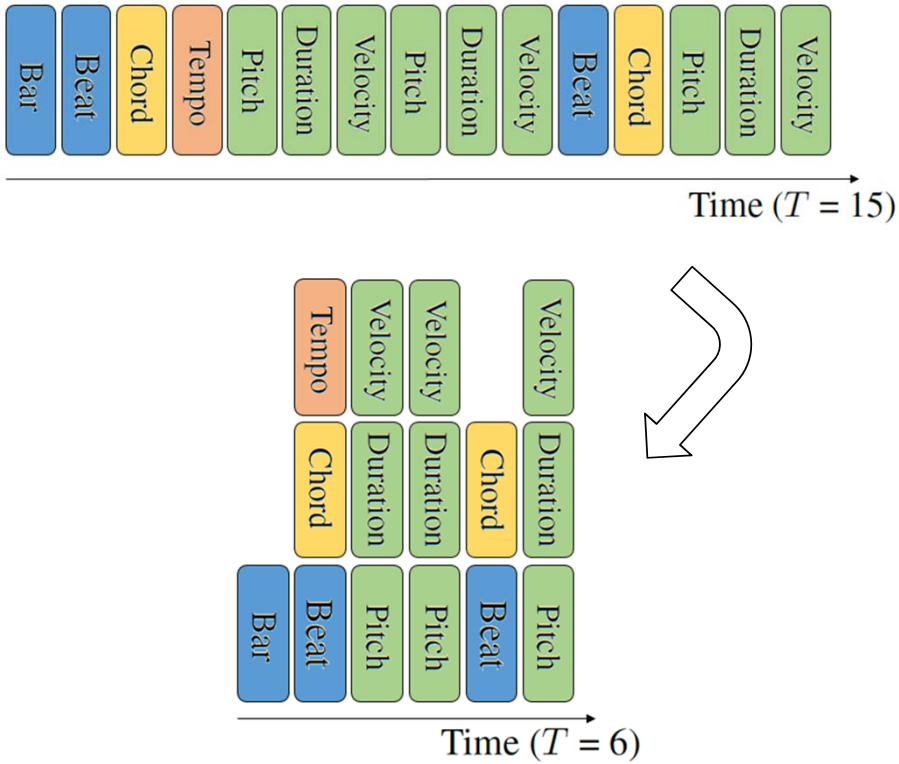
	GPU memory	Training time (single GPU)	Inference time (on GPU)	Quality MOS
Transformer-XL	4-17 GB	3-7 days	2x real time	3.0-**3.3**
CP Transformer (ours)	**4 GB**	**1 day**	**8x real time**	**3.3**

- **Compound Word**: grouping of tokens
- Shorter training time → easier to try different ideas
- Shorter inference time → good for real-time applications

Ref: Hsiao et al, "Compound Word Transformer: Learning to compose full-song music over dynamic directed hypergraphs," AAAI 2021

Compound Word Transformer [hsiao21aaai]

- Unlike the case in text, the vocabulary of music involves tokens of various token types (e.g., note-related, metric-related)



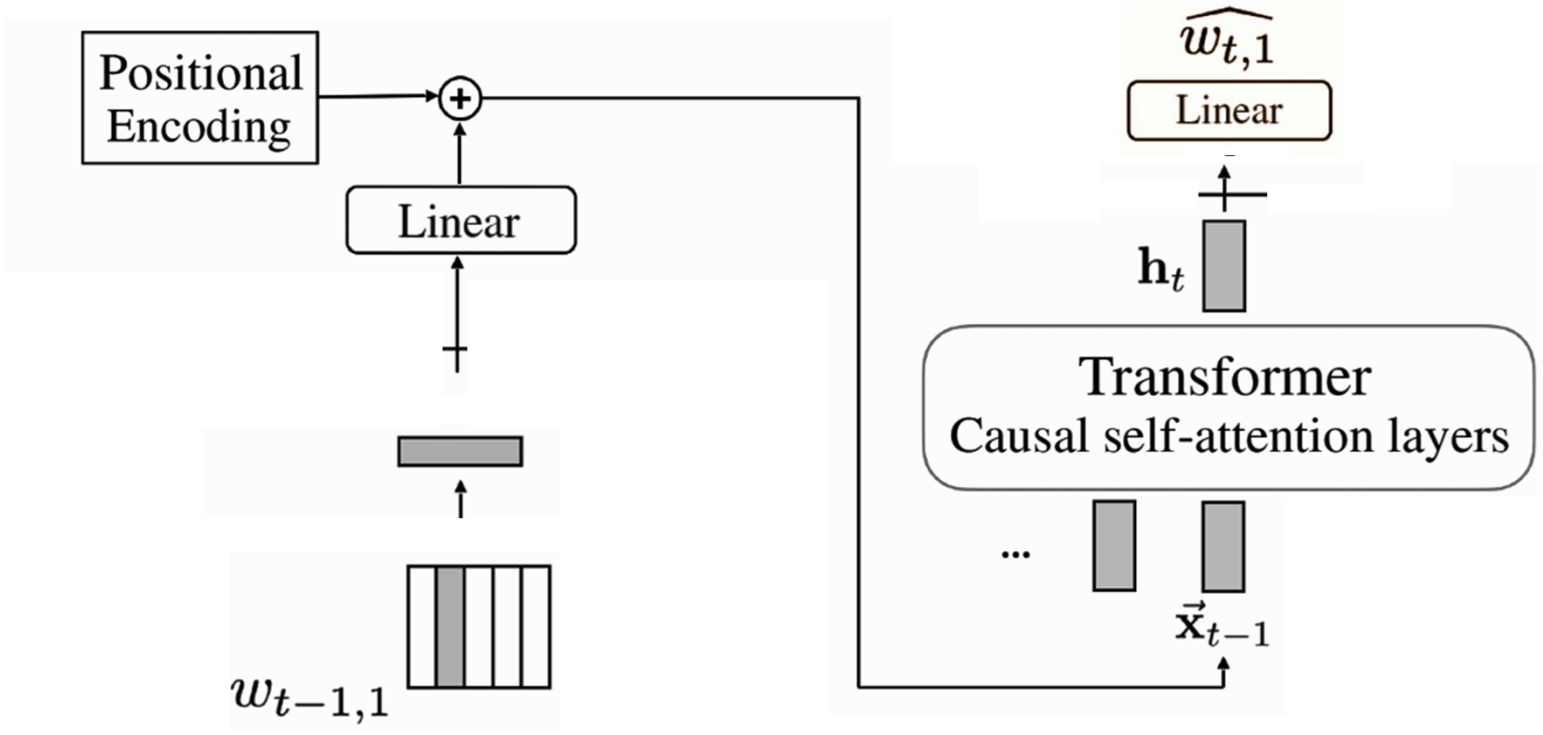
- Therefore, we can

1. **group tokens** into "compound words" (e.g., pitch + duration + velocity)
2. **predict multiple tokens** of various types **at once in a single time step**
3. the embeddings of the predicted tokens are combined to be used as input to the next time step

Ref: Hsiao et al, "Compound Word Transformer: Learning to compose full-song music over dynamic directed hypergraphs," AAAI 2021

Original Transformer

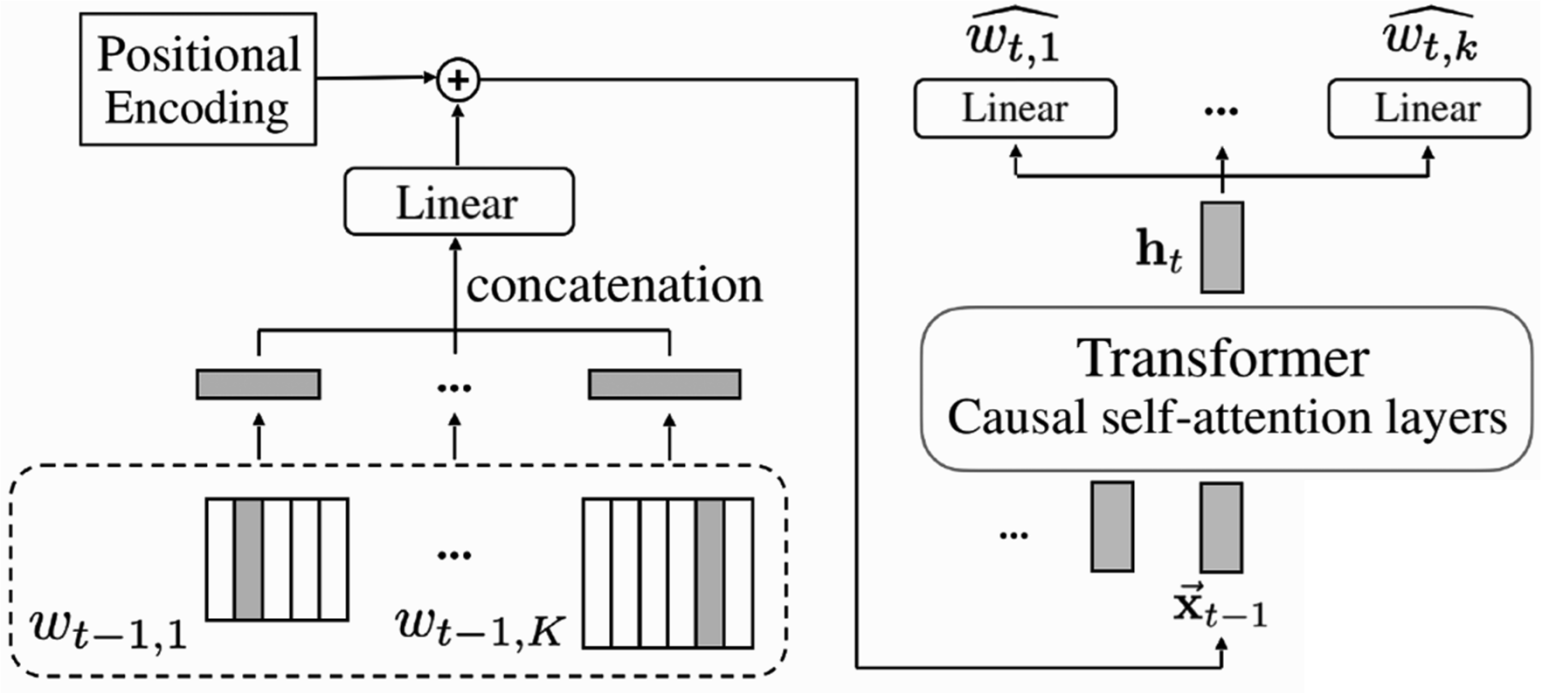
single output at each time step



single input at each time step

Compound Word Transformer

multiple output at each time step



multiple input at each time step

Transformer vs. CP Transformer

- **Transformer**
  - $y_t = f(y_{t-1}, [h_{t-1}, h_{t-2}, \ldots, h_{t-L})$
  - attention window: ***L* tokens**

- **CP Transformer**
  - multiple output: $\text{cp}_t = [y_t^{(1)}, y_t^{(2)}, \ldots, y_t^{(K)}]$
  - multiple input: $\text{cp}_{t-1} = [y_{t-1}^{(1)}, y_{t-1}^{(2)}, \ldots, y_{t-1}^{(K)}]$
  - $\text{cp}_t = f(\text{cp}_{t-1}, [h_{t-1}, h_{t-2}, \ldots, h_{t-L})$
  - `cp` is a super token
  - The latent vectors `h` are associated with each `cp`
  - Attention window: ***L* super-tokens**, or *LK* tokens

Compound Word Transformer [hsiao21aaai]

- We can now feed a **whole song** (up to 10,240 tokens) to our Transformer decoder for training on a single GPU with **11GB** VRAM
- The model converges within 1.5 days using an NVIDIA RTX 2080 Ti

Task	Representation + model@loss	Training time	GPU memory	Inference (/song) time (sec)	Inference (/song) tokens (#)
Conditional	Training data	—	—	—	—
	Training data (randomized)	—	—	—	—
	REMI + XL@0.44	3 days	4 GB	88.4	4,782
	REMI + XL@0.27	7 days	4 GB	91.5	4,890
	REMI + linear@0.50	3 days	17 GB	48.9	4,327
	CP + linear@0.27	0.6 days	10 GB	29.2	18,200
Unconditional	REMI + XL@0.50	3 days	4 GB	139.9	7,680
	CP + linear@0.25	1.3 days	4 GB	19.8	9,546

Ref: Hsiao et al, "Compound Word Transformer: Learning to compose full-song music over dynamic directed hypergraphs," AAAI 2021

Compound Word Transformer [hsiao21aaai]

- **Linear Transformer** [katharopoulos20icml]: **O(*L*)**
  → save GPU space
- **CP**: shorter sequence length
  → save GPU space further, and converge much faster

Representation + model@loss	Training time	GPU memory
Training data	—	—
Training data (randomized)	—	—
REMI + XL@0.44	3 days	4 GB
REMI + XL@0.27	7 days	4 GB
REMI + linear@0.50	3 days	17 GB
CP + linear@0.27	0.6 days	10 GB

Ref: Katharopoulos et al, "Transformers are RNNs: Fast autoregressive Transformers with linear attention", ICML 2020

Customed Design for Different Token Types

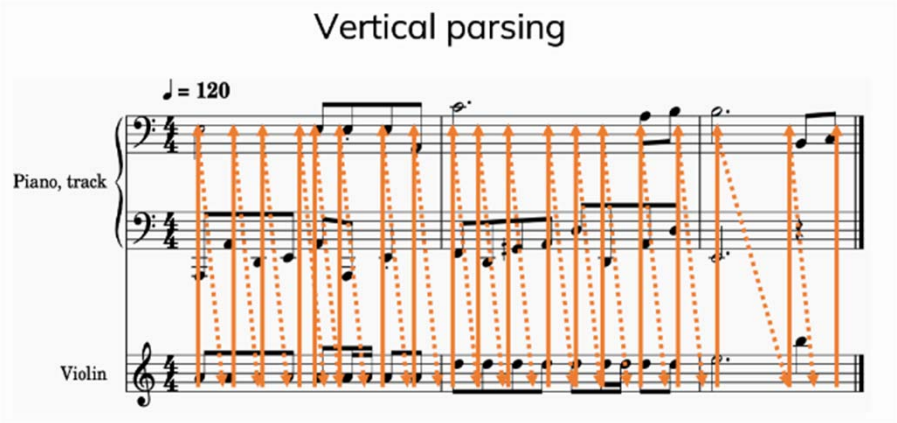
Different embedding size & sampling policy

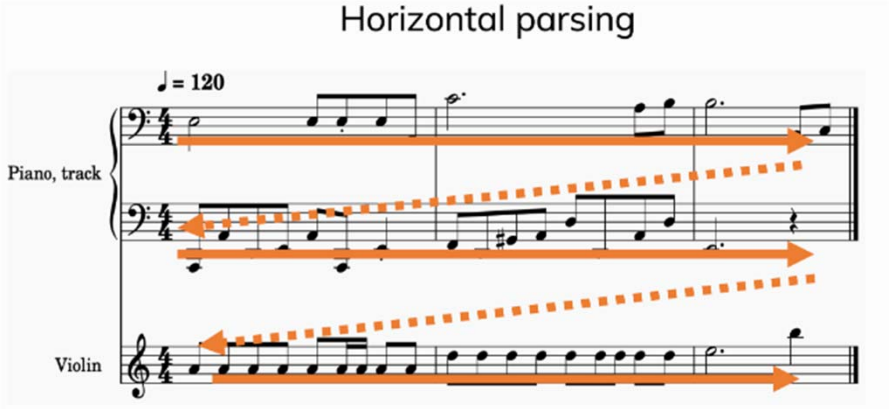
Repre.	Token type	Voc. size $\lvert\mathcal{V}_k\rvert$	Embed. size (d_k)	Sample$_k(\cdot)$ τ	ρ
CP	[track]	2 (+1)	3	1.0	0.90
	[tempo]	58 (+2)	128	1.2	0.90
	[position/bar]	17 (+1)	32	1.2	0.90
	[chord]	133 (+2)	256	1.0	0.90
	[pitch]	86 (+1)	512	1.0	0.90
	[duration]	17 (+1)	128	2.0	0.90
	[velocity]	24 (+1)	128	5.0	1.00
	[family]	4	32	1.0	0.99
	total	341 (+9)	—	—	—
REMI	total	338	512	1.2	0.90

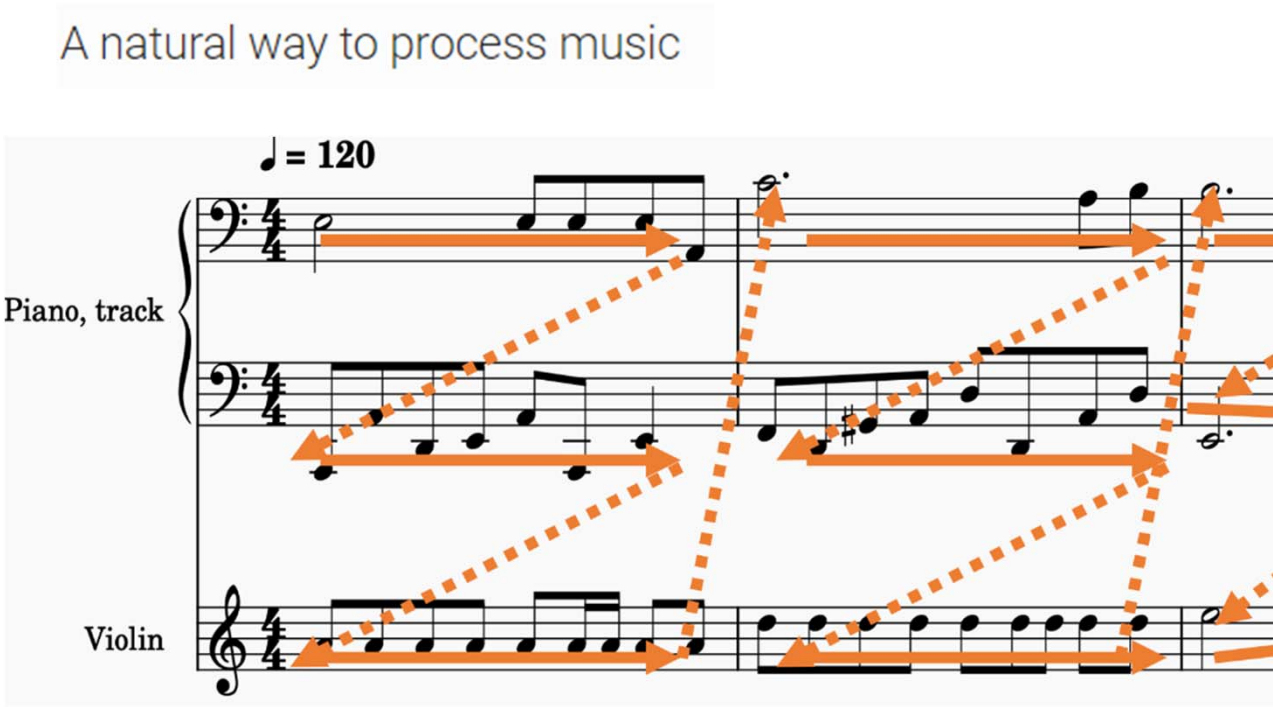
Table 3: Details of the CP representation in our implementation, including that of the sampling policy (τ-tempered top-ρ sampling). For the vocabulary size, the values in the parentheses denote the number of special tokens such as [ignore].

Issue 2: Token Order in Representing Multi-Track MIDI Music

https://musiclang.github.io/tokenizer/







Issue 3: Token Representation for Chords

https://musiclang.github.io/tokenizer/

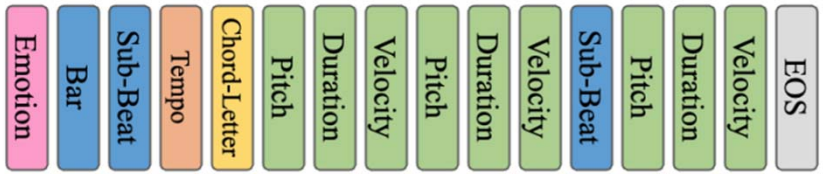
Each chord is depicted using seven tokens (six control tokens plus one "CHORD_CHANGE" token):

- **Scale degrees**: I, II, III, IV, V, VI, VII - represent the chord's degree within the scale. This does not specify the chord's type; the mode and extension are needed for that.
- **Tonality root**: 0 to 12, corresponding to each pitch class possible (C to B).
- **Tonality mode**: m for harmonic minor, M for major.
- **Chord octave**: -4 to 4, indicating the root note's octave relative to the 4th octave.
- **Extension**: Uses figured bass notation to detail the chord's bass and any extensions (e.g., '' (triad in root position), '6', '64', '7', '65', '43', '2'), including variations with sus2 or sus4 for suspended chords.
- **Duration numerator/denominator**: Numerical values representing the chord's fractional duration, providing control over the time signature (since one chord equals one bar in MusicLang).

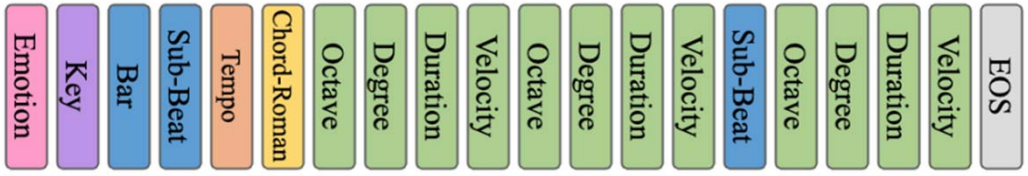
Issue 4: Token Representation for Key

https://emo-disentanger.github.io/

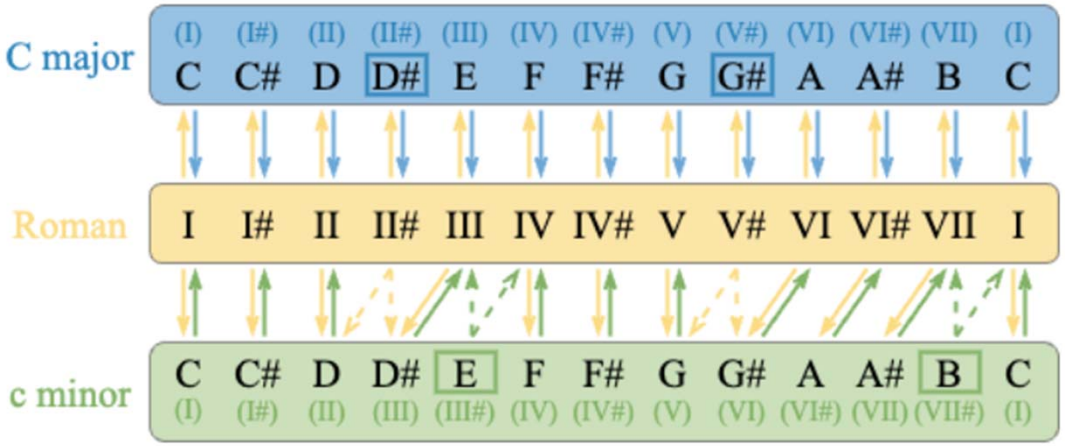
- **Functional** representation
  - Encode both melody and chords with Roman numerals relative to musical keys, to consider the interactions among notes, chords and tonalities



(a) REMI representation



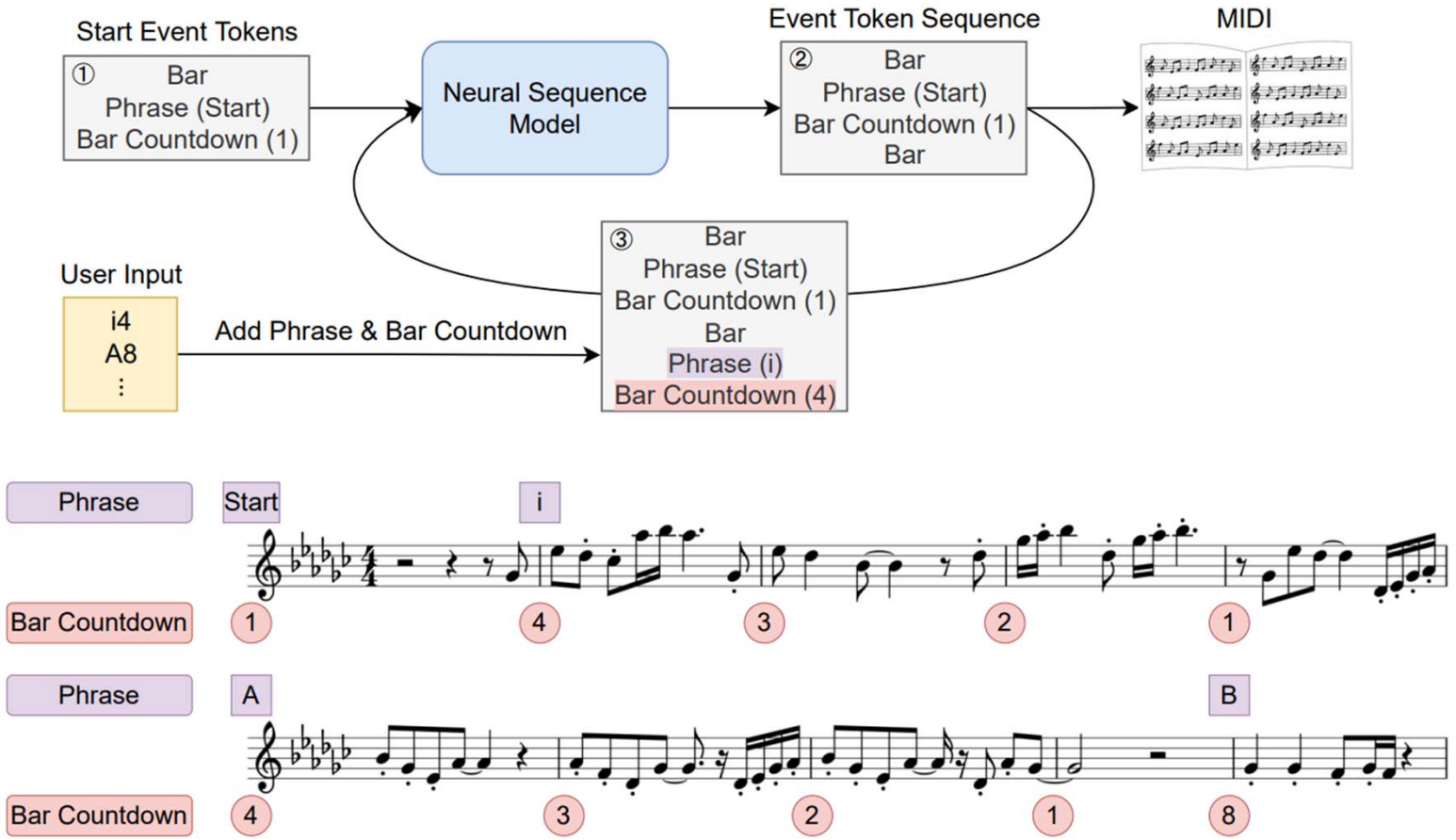
(b) Functional representation



Ref: Huang et al, “Emotion-driven piano music generation via two-stage disentanglement and functional representation”, ISMIR 2024

Issue 5: Token Representation for Musical Structure

https://mil-tokyo.github.io/phrase-length-designated-music-generation/



Outline

- General ideas
- Image-based approach for symbolic-domain music generation
- Text-based approach for symbolic-domain music generation
- Token design
- **Building your first MIDI Transformer**
- Advanced models

Building Your First MIDI Transformer Decoder

- Step (1): Task & Data
- Step (2): Token representation
- Step (3): Choose a model
- Step (4): Train the model till the loss is sufficiently low
- Step (5): Do inference
- Step (6): Listen to the generated music!
- Step (7): Do evaluation

DIY Step (1): Task & Data

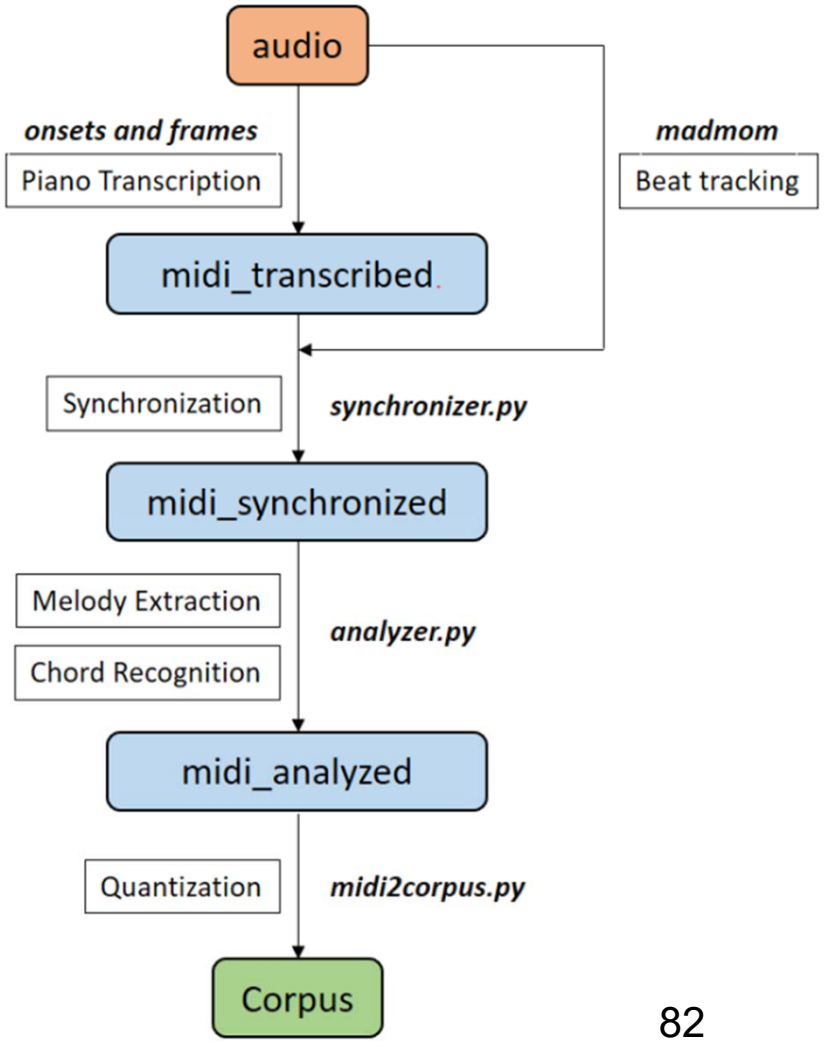
- Choose a task, and a dataset
  - https://github.com/affige/genmusic_demo_list
- List of symbolic musical datasets
  - https://github.com/wayne391/symbolic-music-datasets
- Three relatively more frequently studied cases
  - **Lead sheets** (MIDI score) — Jazz Transformer (ISMIR'20), Compose & Embellish (ICASSP'23)
  - **Multi-track MIDI** (MIDI score) — Multitrack Music Transformer (ICASSP'23)
  - **Piano performance** — Pop Music Transformer (MM'20), Compose & Embellish (ICASSP'23)

DIY: To Prepare a Dataset of Piano Performances

https://github.com/YatingMusic/compound-word-transformer/blob/main/dataset/Dataset.md

- Piano transcription
  - ("Onsets and frames" is no longer the SOTA → lecture 7)
- Beat and downbeat tracking
- Melody extraction
- Chord recognition
- Quantization

- Additional steps
  - Auto-tagging
  - Source separation



DIY Step (2): Choose A Token Representation

- Single-track
  - MIDI-like, REMI, CP, etc
- Multi-track
  - REMI+, Octuple, MMM, MuMIDI, etc
- Depend on your GPU VRAM, you may need to decide a sequence length and cut your musical pieces into segments

DIY Step (3): Choose A Model

- PyTorch 2 Transformers
  https://pytorch.org/blog/accelerated-pytorch-2/

- Huggingface Transformer
  https://huggingface.co/docs/transformers/index

- x-Transformers
  https://github.com/lucidrains/x-transformers

DIY Step (4): Train the Model

- Train the model till the loss is ***sufficiently*** low
- Our experience on pop piano
  - Training loss around **0.30** gives better result
  - The model overfits and generates overly repeating content when the loss is too low
  - Keep several checkpoints and listen to the result to choose one

- No need to consider validation data
  - Unless you are building a conditional generation model with, for example, an encoder/decoder architecture

DIY Step (5): Do Inference

https://towardsdatascience.com/decoding-strategies-that-you-need-to-know-for-response-generation-ba95ee0faadc

- Greedy
- Beam search
- Random sampling
- Temperature
- **Top-K Sampling**
  - Only top K probable tokens should be considered for a generation
- **Nucleus Sampling**
  - Focuses on the smallest possible sets of Top-V words such that the sum of their probability is ≥ p

DIY Step (5): Do Inference

- **Tweak the sampling parameters** to strike a better balance between **diversity** and **quality**
  - The sampling parameters can influence the generation result **A LOT**
    - But, oftentimes people did not describe how they choose the sampling parameters for their proposed models and the evaluated baseline models, leading to unfair performance comparison
  - **A good practice in our Compose & Embellish (ICASSP'23) paper**

> Nucleus sampling [21] with temperatured softmax is employed during inference. We discover that the temperature τ and probability mass truncation point p greatly affect the intra-sequence repetitiveness and diversity of generated lead sheets. Thus, we follow [22] and search within $\tau = \{1.2, 1.3, 1.4\}$ and $p = \{.95, .97, .98, .99\}$ to find a combination with which our lead sheet model generates outputs with the closest mean perplexity (measured by the model itself) to that of validation real data. Finally, $\tau = 1.2$ and $p = .97$ are cho-

Ref: Wu et al, "Compose & Embellish: Well-structured piano performance generation via a two-stage approach", ICASSP 2023

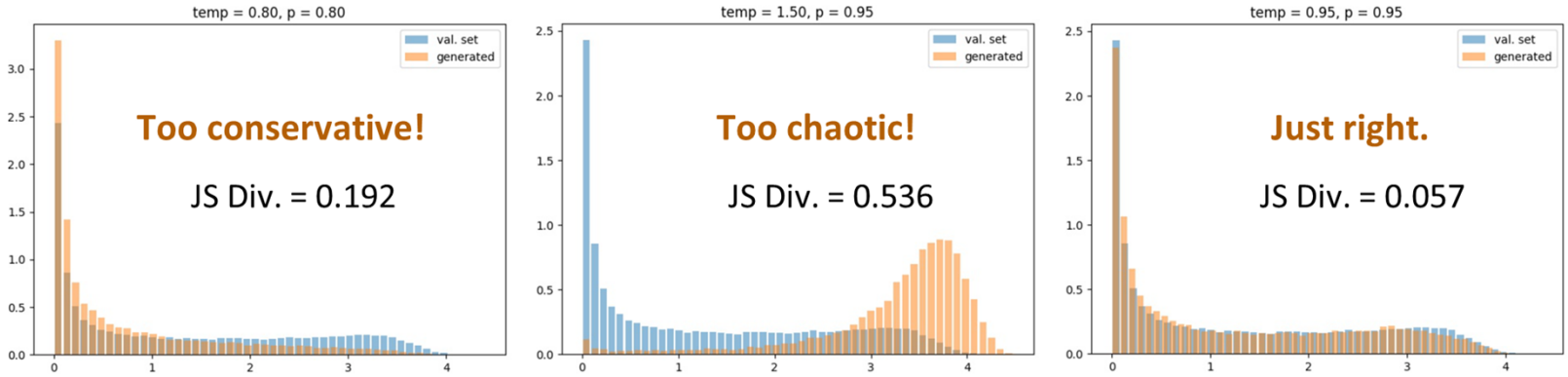
DIY Step (5): Do Inference

- **Nucleus Sampling w/ Temperature** [Holtzman et al., 2019]
  - Temperature -- **Reshape distribution**
  - Top-*p* -- **Truncate long tail**

→ collectively control how **aggressive** the sampling is

(Slide made by Shih-Lun Wu)

- Goal: match the **entropy histogram** during **sampling** to that computed on **val. set** by tuning temp. and top-*p*



Ref: Wu et al, “Compose & Embellish: Well-structured piano performance generation via a two-stage approach”, ICASSP 2023

DIY Step (6): Listen to the Generated Music!

https://www.fluidsynth.org/

https://github.com/bzamecnik/midi2audio

- Use a synthesizer if you prefer to listen to audio

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')
```

DIY Step (7): Do Evaluation

- Objective evaluation
- Subjective evaluation

Outline

- General ideas
- Image-based approach for symbolic-domain music generation
- Text-based approach for symbolic-domain music generation
- Token design
- Building your first MIDI Transformer
- **Advanced models**
  - Theme Transformer
  - MuseMorphose and VQVAE-Transformer
  - Compose-and-embellish

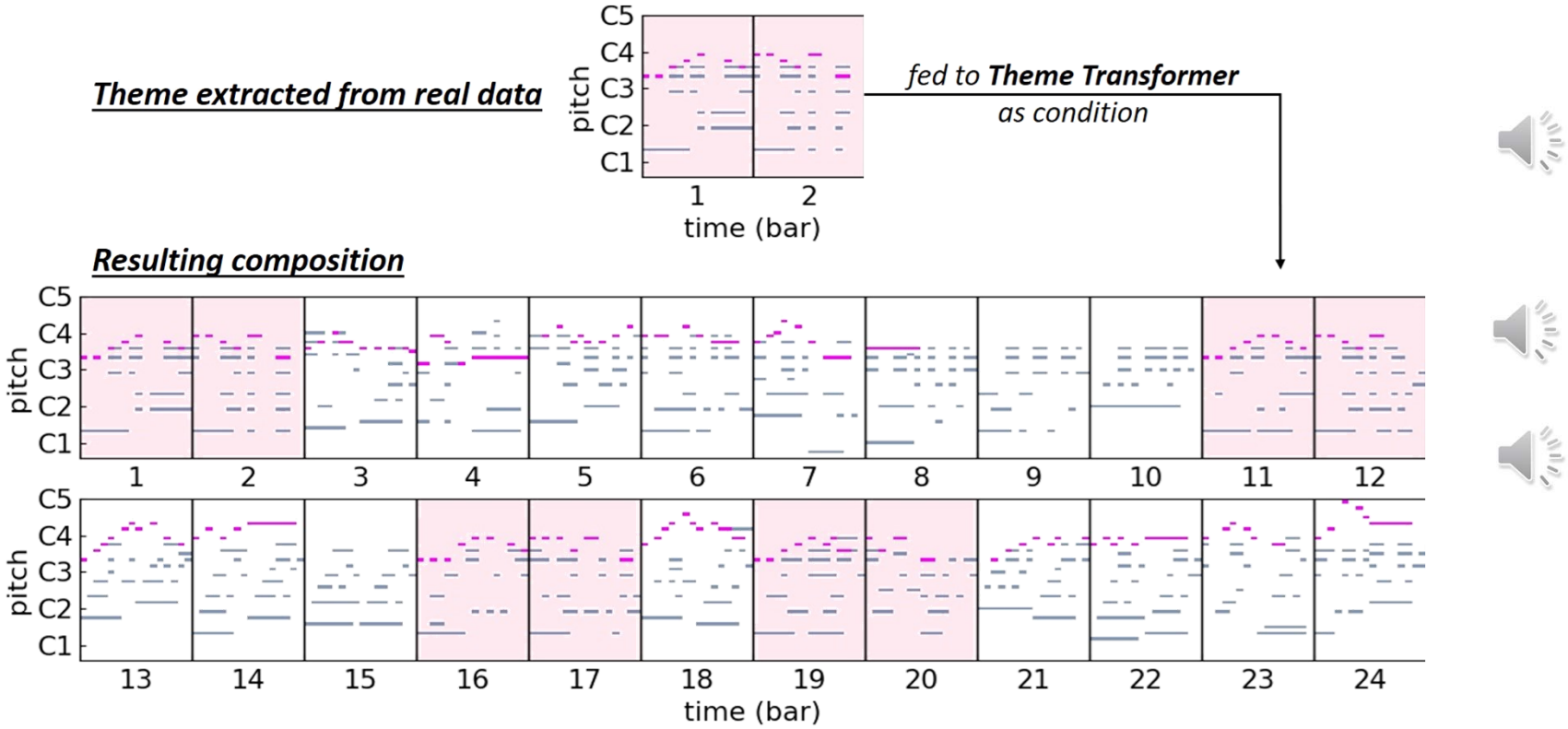
Theme Transformer: Theme-based Conditioning

- Conventional **prompt-based conditioning**
  - Cannot guarantee that the conditioning sequence would repeat or develop in the created continuation
  - Moreover, the influence of the prompt would diminish over time and the generated music *drifts away* from its beginning as the music gets longer
- Proposal: **theme-based conditioning**
  - **Automatic theme finding**: use fragments that have multiple occurrences in a music piece as the conditioning sequence at training time
  - Use **separate memory network** so that the model learns to **exploit the theme condition** when appropriate

Ref: Shih et al, “Theme Transformer: Symbolic music generation with theme-conditioned Transformer”, TMM 2023

Theme Transformer: Theme-based Generation

https://atosystem.github.io/ThemeTransformer/



Ref: Shih et al, “Theme Transformer: Symbolic music generation with theme-conditioned Transformer”, TMM 2023

Theme Transformer: Automatic Theme Finding

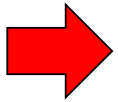
- Use **contrastive learning** to get **melody embedding**
  - **Negative pair**: fragments from different pieces
  - **Positive pair**: to be invariant to the following variations
    - Pitch shift on scale
    - Last note duration variation
    - Note splitting and combination

- Use DBSCAN to find clusters for each piece of music
- Pick the **largest cluster** as the theme for a piece of music

Theme Transformer: Token Design

- Use **THEME-START** and **THEME-END** tokens to indicate the beginning and ending of a ***theme region***

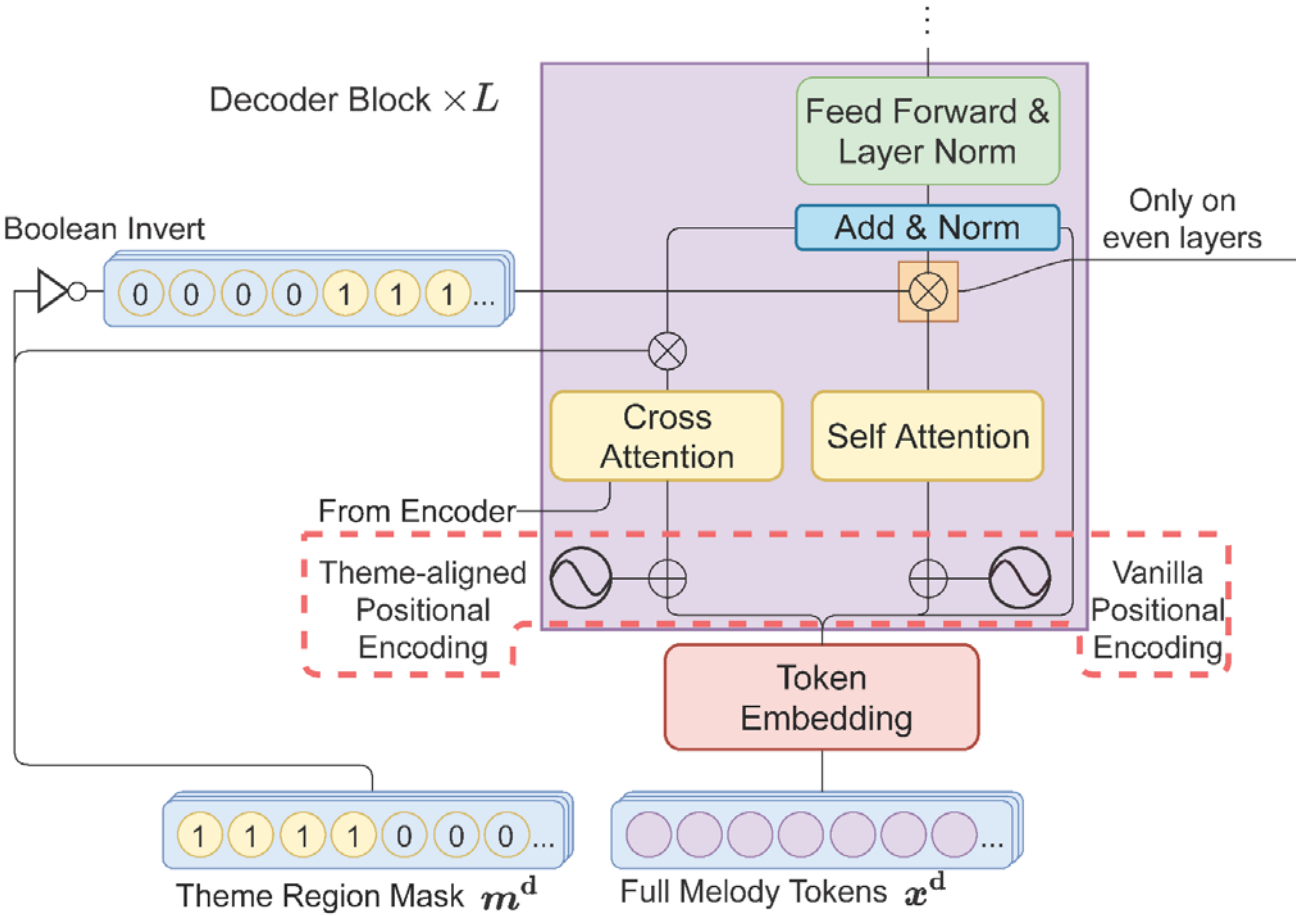
- At inference time, the model decides on its own when to enter a theme region



Token type	Piano Representation
NOTE-PITCH	127×2
NOTE-DURATION	64×2
NOTE-VELOCITY	126×2
TEMPO	76
BAR	1
SUBBEAT	16
THEME	2
PADDING	1
Total	730

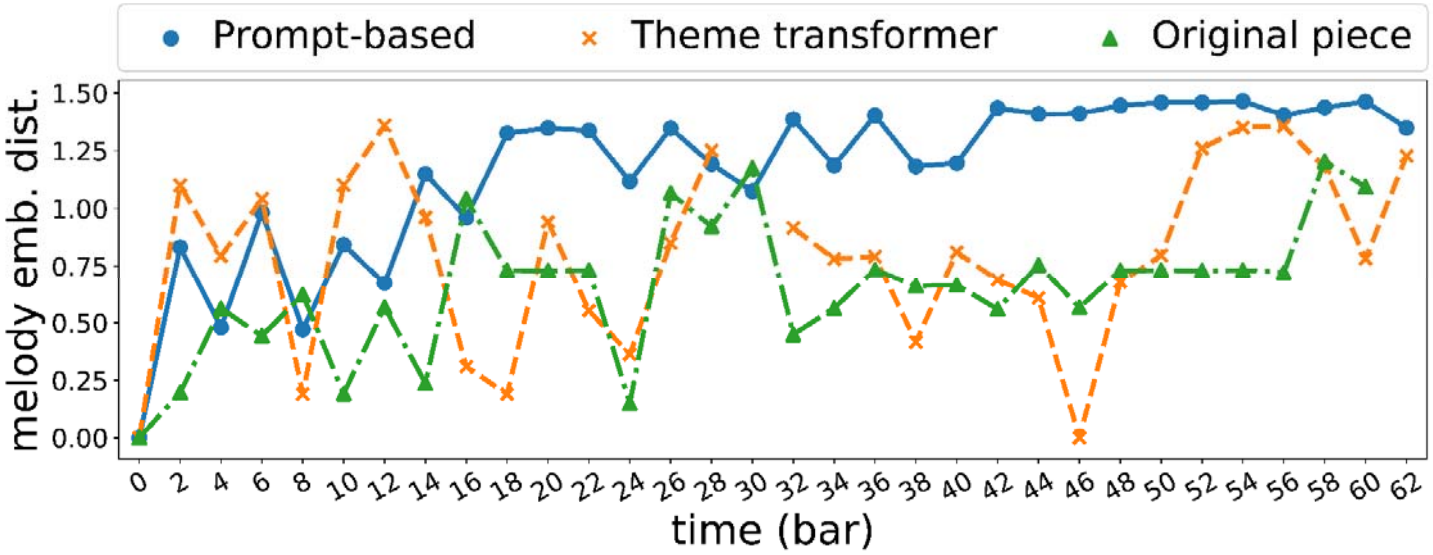
Theme Transformer: Attention Design

- **Separate memory network**
  - **Self-attention:** to attend to the past (local coherence)
  - **Cross-attention**: to attend to the theme
- **"Switch on"** cross-attention **only during theme regions**
- Dedicated **theme-aligned positional encoding**



Ref: Shih et al, "Theme Transformer: Symbolic music generation with theme-conditioned Transformer", TMM 2023

Theme Transformer: Evaluation



	Pitch class consistency↑	Melody inconsistency↓	Grooving consistency↑	Theme inconsistency↓	Theme uncontrollability↓
Baseline (Huang et al. 2019)	.59±.07	.33±.38	.84±.09	—	—
Seq2seq Transformer (proposed)	.61±.04	.46±.28	.90±.06	.22±.03	.45±.19
Theme Transformer (proposed)	.61±.06	.13±.24	.92±.07	.15±.06	.34±.13
Original pieces	.65±.05	.09±.18	.74±.10	.11±.06	.09±.06

Ref: Shih et al, “Theme Transformer: Symbolic music generation with theme-conditioned Transformer”, TMM 2023

Theme Transformer: Possible Extensions

- Extensions
  - Multiple themes
  - Application to other music genres
  - Use melody as the theme prompt (and make a UI)
  - CP token representation
  - Memory-based RNN
  - Variants of gating
  - To-copy pretrain
  - Better theme retrieval method
  - Better segmentation method

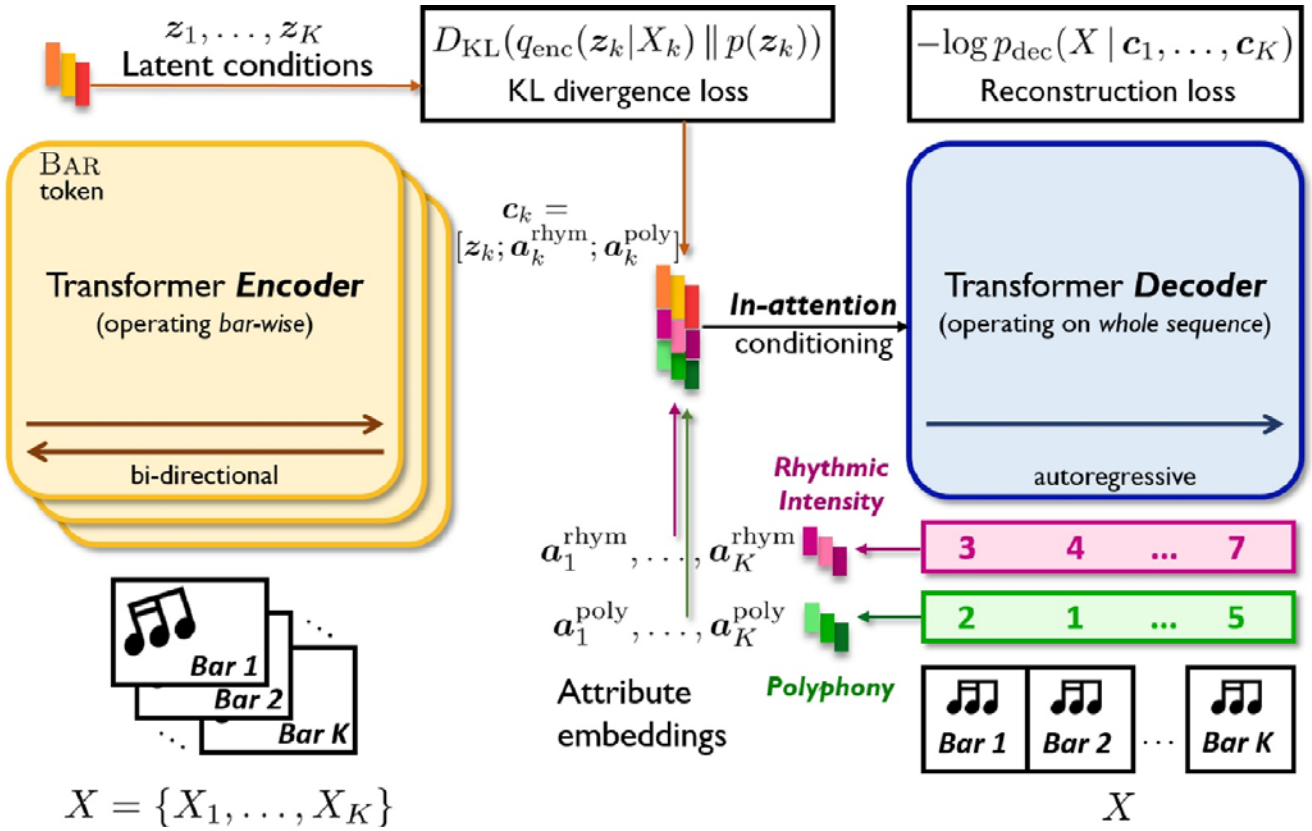
Theme Transformer: Strength and Weakness

- Allow us to repeat something over
- But
  - No development
  - Cannot control the structure

- Question: Can we generate a **skeleton/outline** of the music first, before we fill the details?

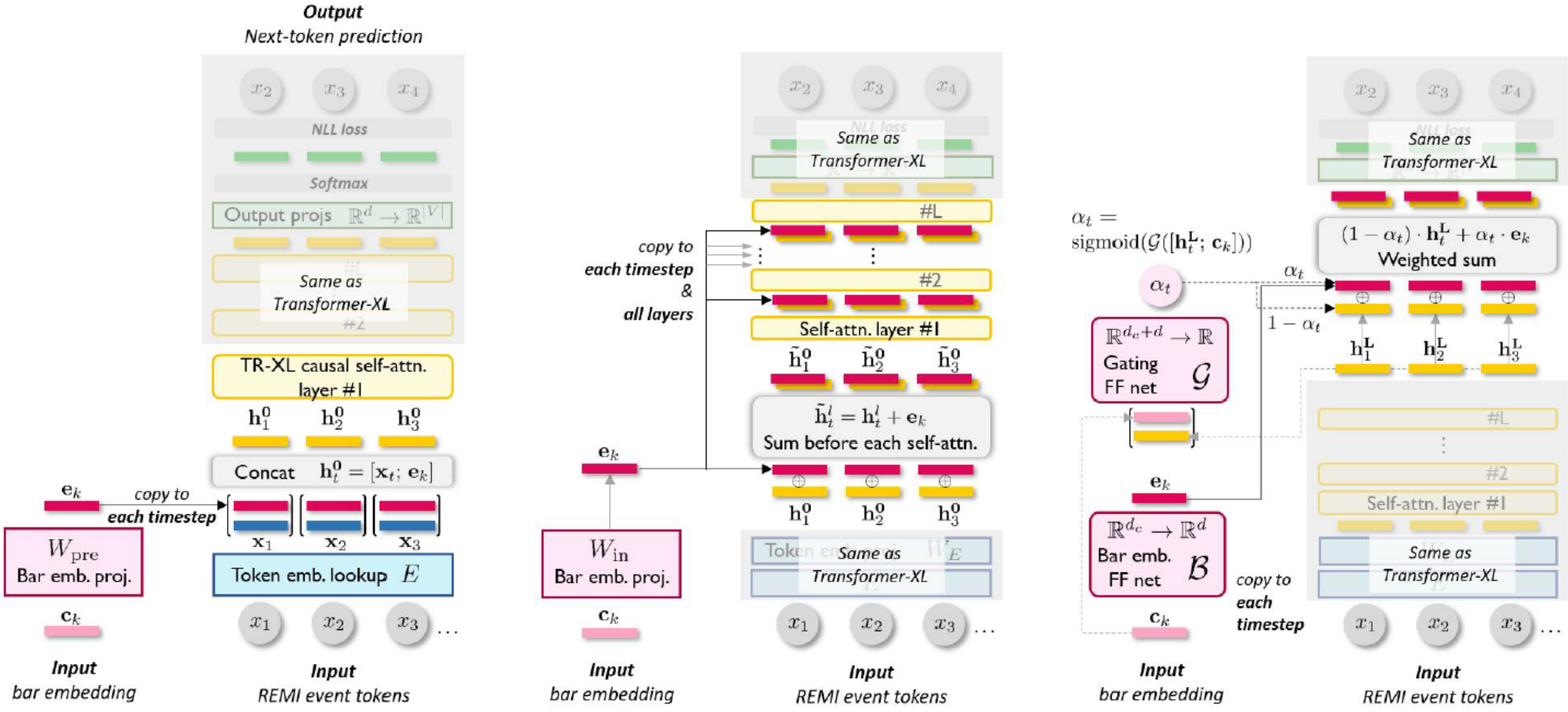
MuseMorphose: VAE-Transformer for Style Transfer

https://slseanwu.github.io/site-musemorphose/



Ref: Wu & Yang, "MuseMorphose: Full-song and fine-grained piano music style transfer with one Transformer VAE", TASLP 2023

MuseMorphose: Various Conditoining Mechansims



(a) Pre-attention conditioning (b) In-attention conditioning (c) Post-attention conditioning

Ref: Wu & Yang, "MuseMorphose: Full-song and fine-grained piano music style transfer with one Transformer VAE", TASLP 2023

MuseMorphose: VAE-Transformer for Style Transfer

Model	Best ckpt. at step	**Recons. NLL** *Train*	*Val.*	**KL divergence** *Train*	*Val.*
MIDI-VAE [19]	65K	0.676	0.894	0.697	0.682
Attr-Aware VAE [20]	190K	0.470	0.688	0.710	0.697
MuseMorphose (Ours), AE objective	90K	0.130	0.139	6.927	6.928
MuseMorphose (Ours), VAE objective	75K	0.697	0.765	0.485	0.484
MuseMorphose (Ours), preferred settings	140K	0.457	**0.584**	0.636	**0.636**

bold: leads the baselines under the same latent space constraints

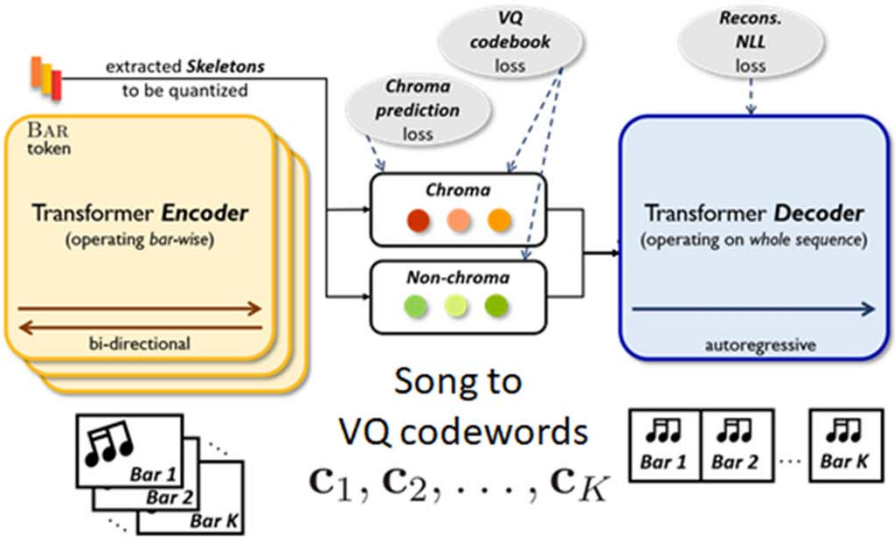
Model	**Fidelity** $sim_{\text{chr}}\uparrow$	$sim_{\text{grv}}\uparrow$	**Control** $\rho_{\text{rhym}}\uparrow$	$\rho_{\text{poly}}\uparrow$	**Fluency** PPL$\downarrow$	**Diversity** $sim_{\text{chr}}\downarrow$	$sim_{\text{grv}}\downarrow$
MIDI-VAE [19]	75.6	85.4	.719	.261	8.84	74.8	86.5
Attr-Aware VAE [20]	85.0	76.8	**.997**	.781	10.7	86.5	**84.7**
Ours, AE objective	**98.5**	**95.7**	.181	.154	**6.10**	97.9	95.4
Ours, VAE objective	78.6	80.7	.931	.884	7.89	**73.2**	84.9
Ours, preferred settings	91.2	84.5	.950	**.885**	7.39	87.1	87.6

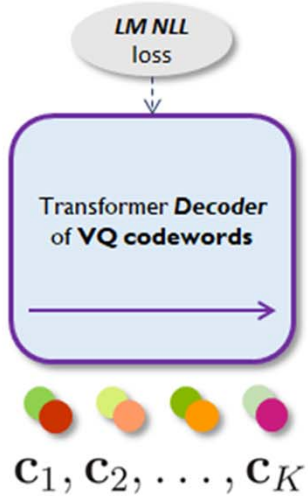
Ref: Wu & Yang, "MuseMorphose: Full-song and fine-grained piano music style transfer with one Transformer VAE", TASLP 2023

VQVAE Transformer for Structured Generation

Low-level Model:
Structure Extractor + Conditional Decoder

High-level Model:
Structure Generator





(Slide made by Shih-Lun Wu; unpublished)

VQVAE Transformer Result

- **Conditional generation:** given VQs from songs in test set
- **Better structure?** -- note n-gram repetition stats

		Short (3∼6 notes)	Mid (7∼15 notes)	Long (16∼40 notes)
Unique %	**Real data**	42.67	60.85	75.49
	VQ model	45.30	64.05	76.95
	Uncond. model	9.32	15.23	27.13
Reappearance interval (in bars)	**Real data**	16.61	23.68	29.22
	VQ model	16.33	23.74	29.08
	Uncond. model	2.74	3.56	4.70

- **Possible plagiarism?** -- len of longest common note string
  - **VQ gen** vs. **Ref real song:** 6.24 +/- 1.98 notes
  - Random real song pairs: 3.79 +/- 1.49 notes

(Slide made by Shih-Lun Wu; unpublished results)

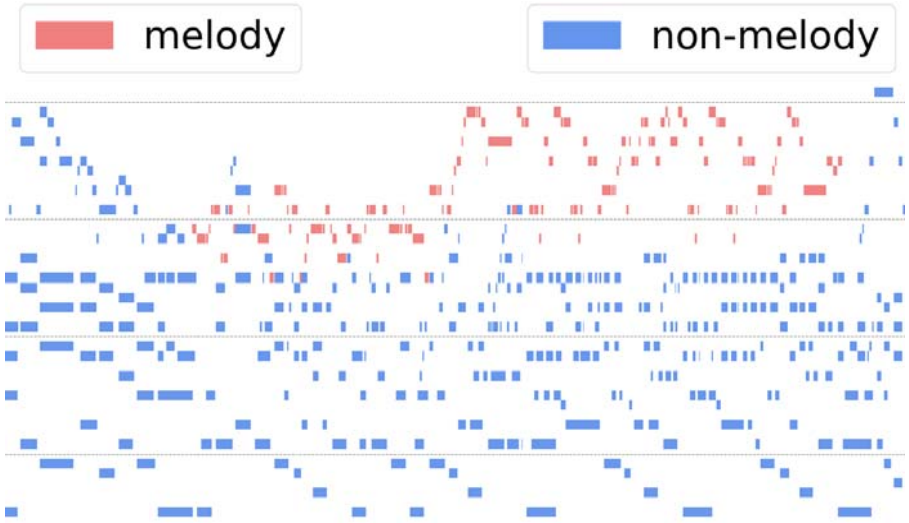
VQVAE Transformer

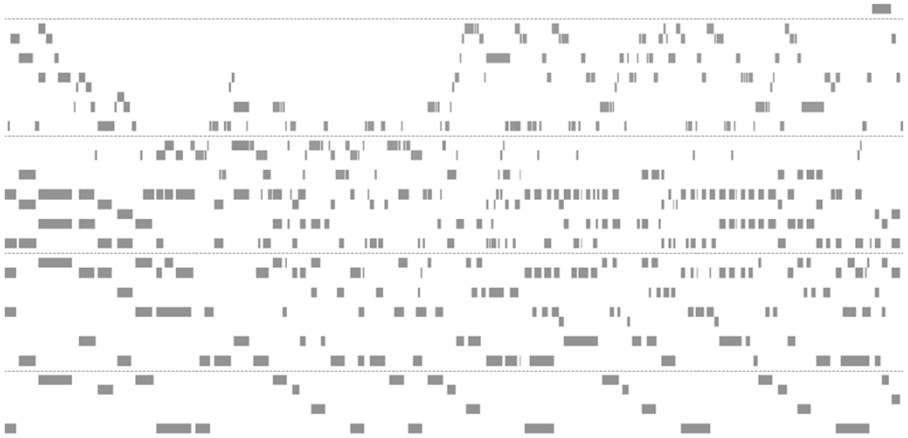
- May allow us to generate a **skeleton/outline** of the music first, before we fill the details
- But
  - Empirical result incomplete (not yet finished)

- Question: The VQVAE approach sounds quite brute-force, is there a **musically-inspired** approach?

Domain Knowledge Inspired Approach

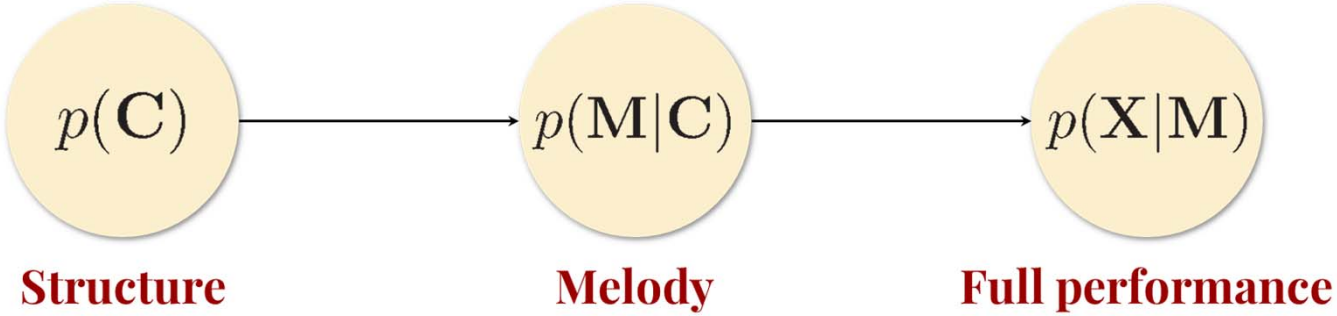
Modular approach > End-to-end approach





Generate melody first,
then the piano accompaniment

Generate Melody First, then the Piano

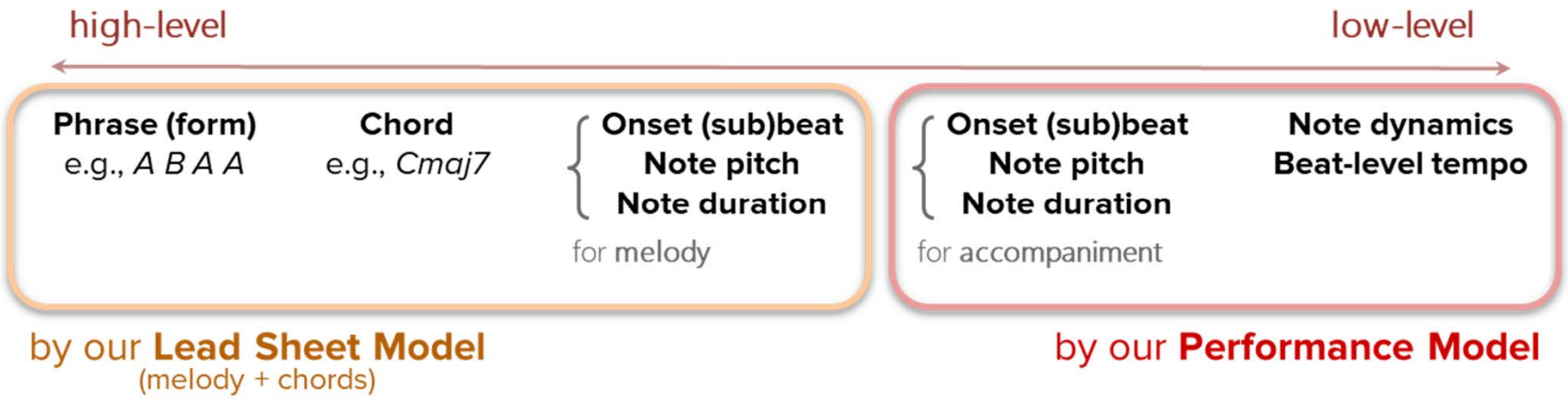


- End-to-end full-song generation is hard; will a 2-stage model make things easier? → Generate **lead sheet** (melody & chords) first, and then the **full piano performance**
- Can we pretrain the melody stage with non-piano data to do better?

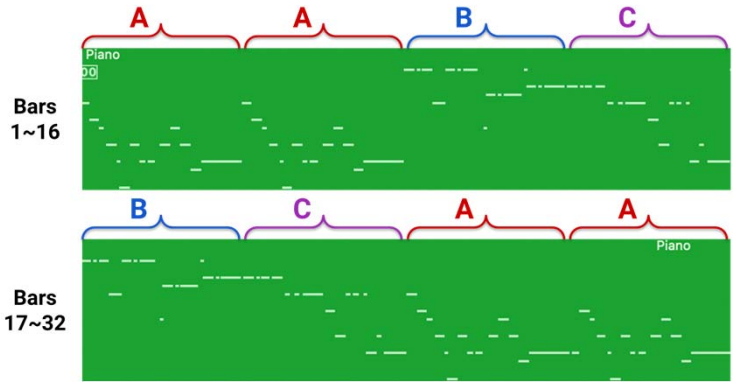
(Slide made by Shih-Lun Wu)

Compose & Embellish

(Figure made by Shih-Lun Wu)



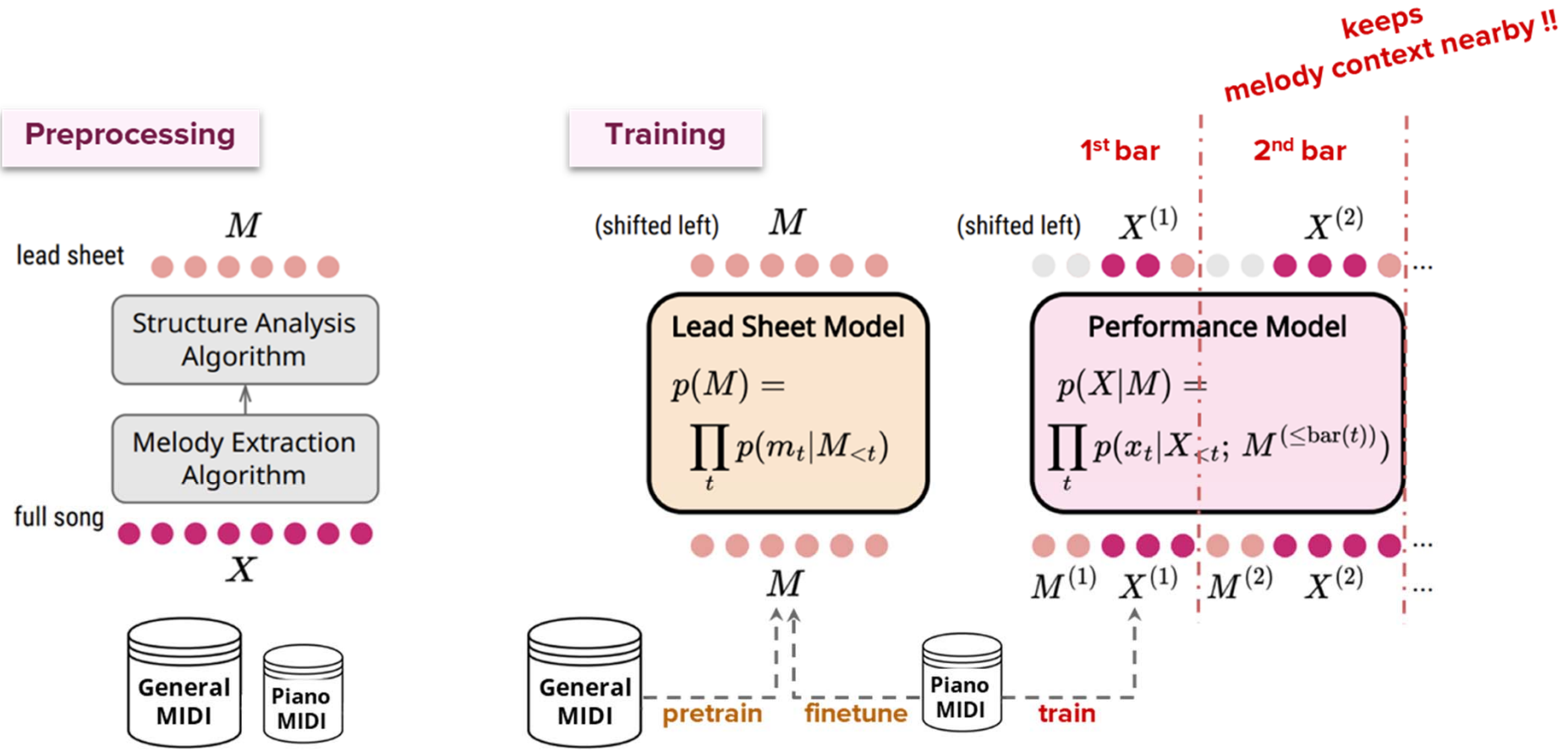
- A two-stage model
  - **Compose** lead sheet & form
  - **Embellish** it to create piano performance



Ref: Wu & Yang, "Compose & Embellish: Well-structured piano performance generation via a two-stage approach", ICASSP 2023

Compose & Embellish

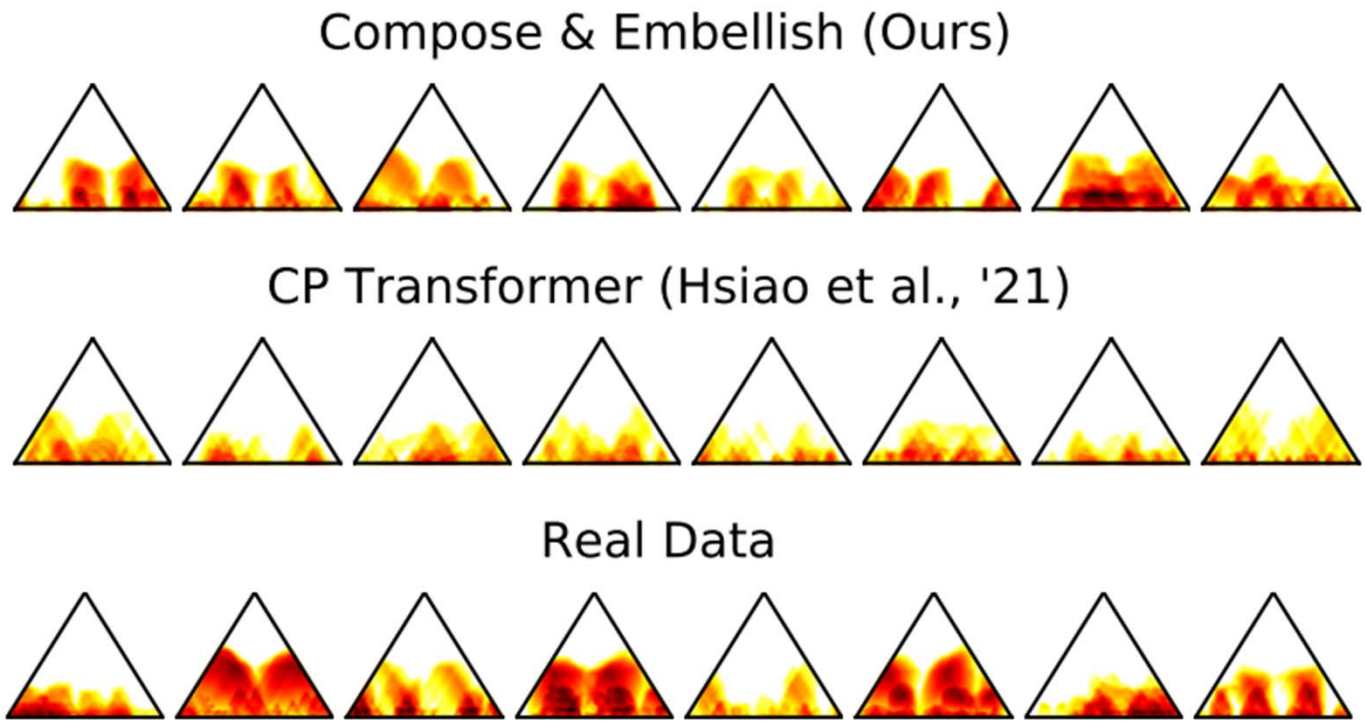
(Figure made by Shih-Lun Wu)



Ref: Wu & Yang, "Compose & Embellish: Well-structured piano performance generation via a two-stage approach", ICASSP 2023

Compose & Embellish: Result

https://github.com/slSeanWU/Compose_and_Embellish



Ref: Wu & Yang, “Compose & Embellish: Well-structured piano performance generation via a two-stage approach”, ICASSP 2023

Compose & Embellish: Result

- Greatly outperforms CP Transformer in structureness
- Ablations show that both the use of structure tokens ("struct") and the lead sheet data pre-training help

	Structureness (in %)			**Melody-line Diversity**		
	$\mathcal{SI}_{\text{short}}$	$\mathcal{SI}_{\text{mid}}$	$\mathcal{SI}_{\text{long}}$	$\mathcal{DN}_{\text{short}}$	$\mathcal{DN}_{\text{mid}}$	$\mathcal{DN}_{\text{long}}$
Naive repeats (1-bar)	83.6 ±8.6	88.3 ±5.4	75.7 ±11	1.2 ±0.6	1.2 ±0.6	1.3 ±0.6
Naive repeats (1-phrase)	75.5 ±15	86.2 ±6.8	73.9 ±11	8.2 ±4.8	10.0 ±5.8	10.8 ±6.5
CP Transformer [4]	32.5 ±3.3	29.9 ±4.4	17.9 ±6.3	82.0 ±8.9	99.6 ±0.7	100 ±0.0
COMPOSE & EMBELLISH	**36.8** ±6.7	**35.1** ±7.7	**25.8** ±12	**49.7** ±19	69.9 ±20	81.6 ±17
w/o struct	**36.8** ±6.8	34.2 ±9.2	23.8 ±11	48.0 ±17	68.7 ±17	82.7 ±14
w/o pretrain	36.6 ±7.5	33.1 ±8.8	19.6 ±10	53.2 ±19	**74.9** ±18	**87.9** ±14
w/o struct & pretrain	36.3 ±6.0	34.1 ±6.7	23.0 ±8.4	52.5 ±18	76.1 ±16	89.0 ±11
Real data	43.8 ±7.1	43.1 ±8.4	34.8 ±12	50.0 ±14	74.1 ±14	88.3 ±11

Ref: Wu & Yang, "Compose & Embellish: Well-structured piano performance generation via a two-stage approach", ICASSP 2023

Compose & Embellish: Result

- 5-point Likert scale on 5 aspects
  - coherence (Ch), correctness (Cr), structureness (S), richness (R), overall (O)

	Ch	Cr	S	R	O
CPT	2.38±0.9	2.49±0.9	2.33±0.9	2.64±0.9	2.33±0.9
C&E (ours)	3.53±0.9	3.11±1.0	3.36±1.2	3.29±1.0	3.18±0.9
Real data	4.42±0.7	4.13±0.8	4.44±0.8	4.24±0.8	4.40±0.7

(StDevs of *individual data points* shown after ±)

- Large (>1 point) gain over CP Transformer on coherence & structureness
- Still a long way to go to rival humans

(Figure made by Shih-Lun Wu)

Recap

- **Theme Transformer**
  - Allow for explicit **conditioning of "theme"** thru customized cross attention design
  - The idea is quite general and applicable to various types of music (melody, single-track piano, multi-track MIDI etc)

- **VAE Transformer (MuseMorphose)**
  - Allow for **time-varying condition**
  - For **style transfer**, and variation generation (as it needs a reference MIDI)
  - The idea can be extended to multi-track MIDI
  - The input can take audio files instead

Recap

- **VQVAE Transformer**
  - A brute-force approach for unconditional MIDI generation that generates a **blueprint** first, before filling the details
  - Can be extended to multi-track MIDI

- **Compose-and-embellish**
  - A musically-inspired approach for unconditional MIDI generation that generates a ***lead sheet* as the blueprint** first, before filling the details
  - More interpretable, interactive; can use unpaired lead sheet data for pretraining
  - Can be extended to multi-track MIDI
  - Can be combined with Theme Transformer
  - The input can take audio files instead