2025 edition

Deep Learning for Music Analysis and Generation

Transformer-based Text-to-music Generation

(text → audio)

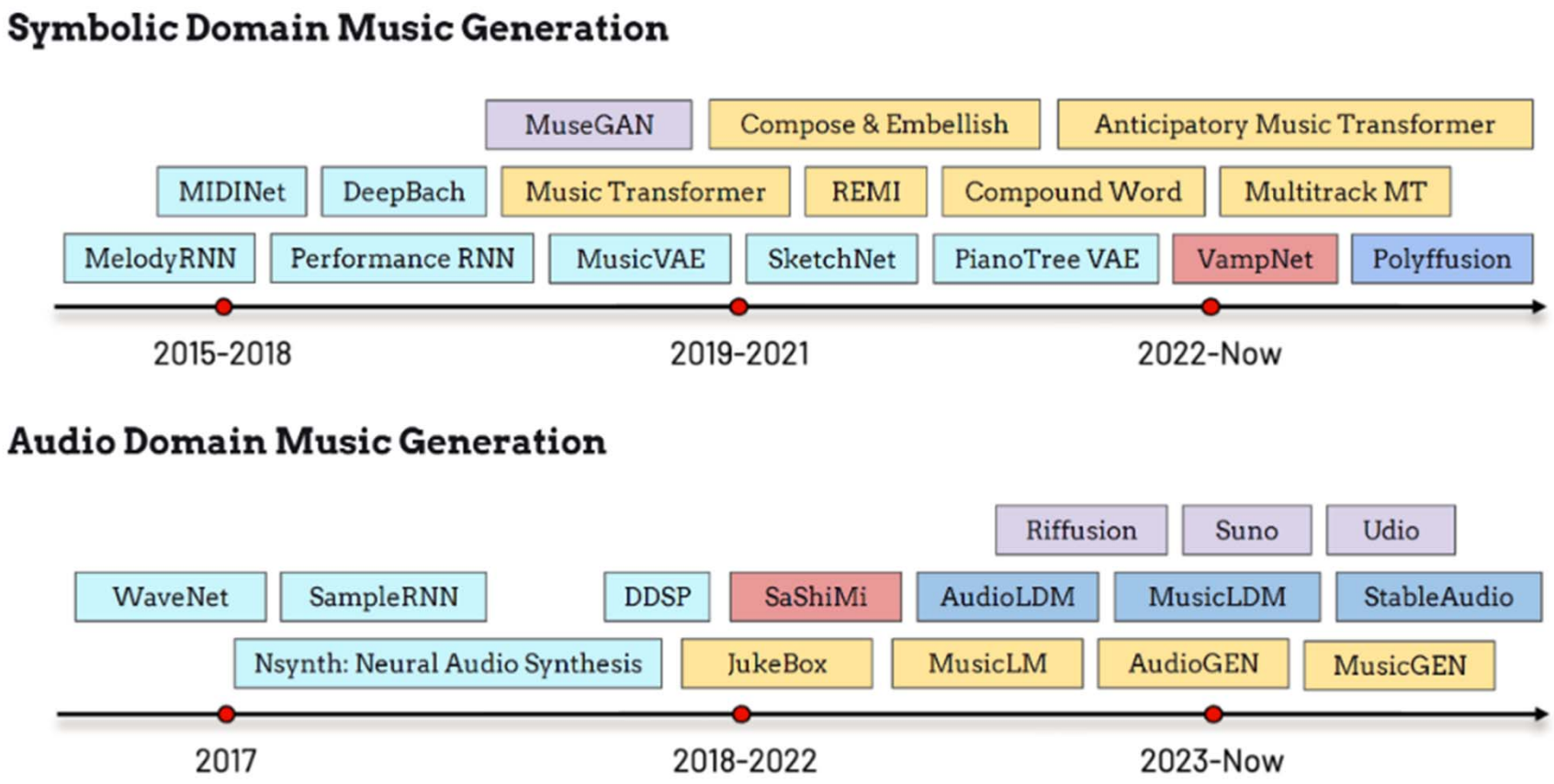
Yi-Hsuan Yang Ph.D.
yhyangtw@ntu.edu.tw

Outline

- **Basics**
- VQ-VAE
- Audio codec models
- Transformer
- Transformer-based text-to-music generation

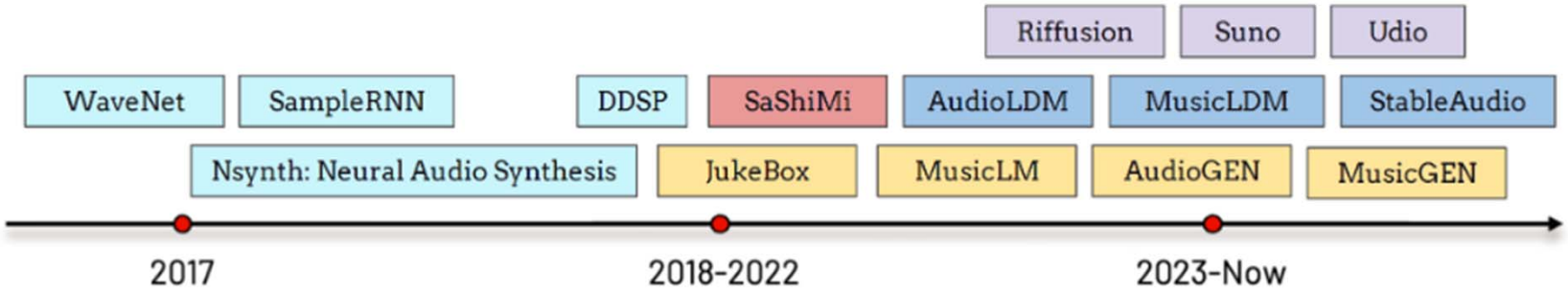
Rapid Progress in Music Generation

https://mulab-mir.github.io/music-language-tutorial/generation/intro.html



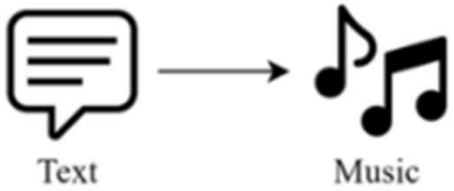
Rapid Progress in Music Generation

- Single note generation, single phrase generation (e.g., WaveNet)
- **Text-to-music generation**: generating instrumental music (e.g., MusicLM)
- **Song generation**: generating vocal+backings (e.g., JukeBox, Suno)



Text-to-Music Generation: Problem Definition

https://mulab-mir.github.io/music-language-tutorial/generation/intro.html



$P(\text{music}|\text{text})$

Text:

Keyword of music:

[pop, heart-warming, piano solo, wedding purpose]

Description:

A haunting piano melody weaves through lush string harmonies over a subtle 6/8 rhythm, with a delicate chord progression moving from C minor to F major, creating an ethereal soundscape. Syncopated rhythms in the bass add tension, as the piece gradually builds in intensity before resolving into a soft, contemplative cadence.

The Birth of Text-to-Music Generation Research

- ***Transformer***-based text-to-music (TTM) generation
  - Google's **MusicLM** model (**2023.01**): https://google-research.github.io/seanet/musiclm/examples/
  - ByteDance's MeLoDy model (2023.05): https://Efficient-MeLoDy.github.io/
  - Facebook's MusicGen model (2023.06): https://github.com/facebookresearch/audiocraft
- ***Diffusion***-based text-to-music (TTM) generation
  - **AudioLDM** (**2023.01**), AudioLDM2 (2023.08)
  - Stable audio (2024.02)
- Also great progress in "**text-to-audio**" (text-to-sound) but omitted here

Elements of a Text-to-Music Generation Model

- **1. Audio encoder/decoder**
  - So that we can build an LM over the (discrete or continuous) decoder input
- **2. Music LM**
  - Can do unconditional generation
- **3. Text encoder** or **text-music joint embedding space**
  - Provides conditions for the music LM to realize text-to-music generation

Key Technologies that Enable Text-to-Music Generation

- **Language model (LM) for music**: RNN, Transformers
  - Model long term dependency
  - Initially applied to MIDI music generation as it's easier to convert MIDI events into tokens
  - **Significance**: A few bars of MIDI music → full-length MIDI music
- **Audio waveform encoder/decoder**
  - Thanks to advances in Mel-vocoders, source separation, audio codec models, etc
  - Tokenize audio files (spectrograms or waveforms)
    - Enable us to build LM for musical audio
    - And learn the dependency between text tokens and audio tokens
  - **Significance:** LM for MIDI music → LM for musical audio

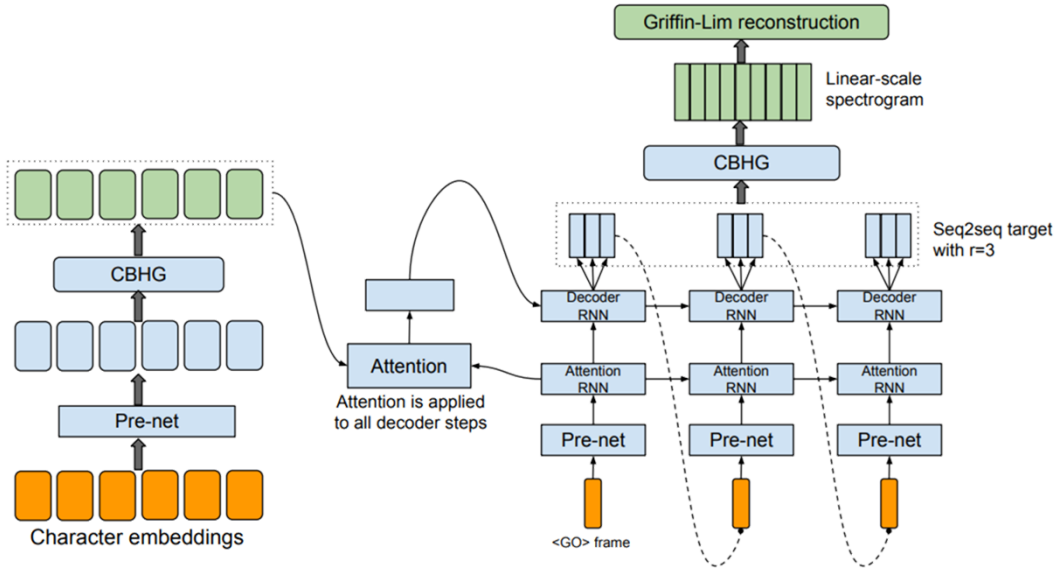
Outline

- Basics
- **VQ-VAE**
- Audio codec models
- Transformer
- Transformer-based text-to-music generation

Building an LM for Audio

- MIDI events are by nature **discrete**; easy to build an LM for MIDI
- Audio files (waveforms or spectrograms) are **continuous** so it's trickier
- We can still build an autoregressive model for continuous audio
  - Predict the next frame given the previous frames (e.g., TacoTron, Transformer-TTS)
  - Use MSE loss

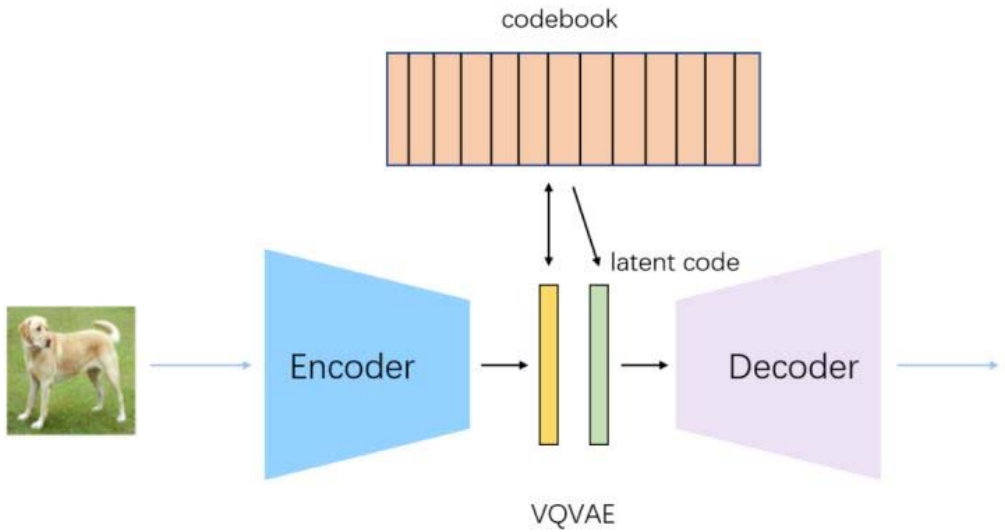
- But, LLMs excel at working with **discrete data**
  - Use cross entropy loss



(Figure from Wang et al, "Tacotron: Towards end-to-end speech synthesis," INTERSPEECH 2017)

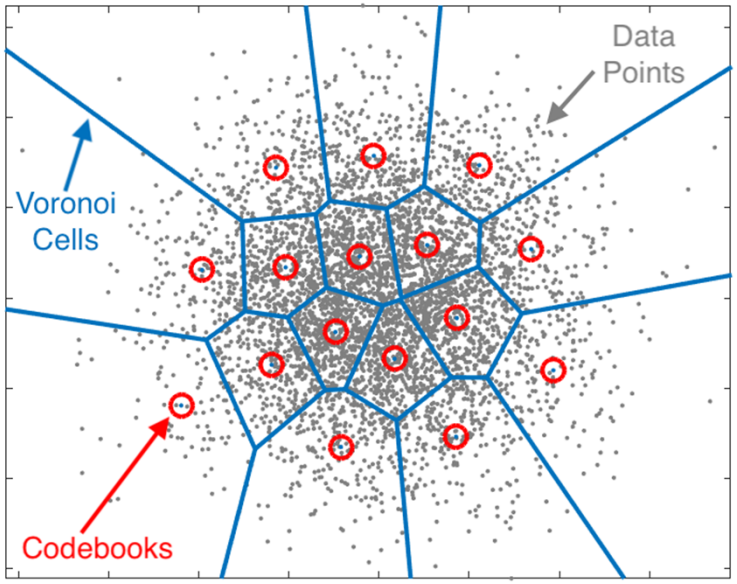
Language Modeling (LM) in Music Audio

- **Audio domain**
  - use some encoder/decoder to convert audio into "**audio tokens**"
  - **for example: VQ-VAE**
  - then build an LM



https://zhuanlan.zhihu.com/p/388299884

Illustration of vector quantization (VQ)

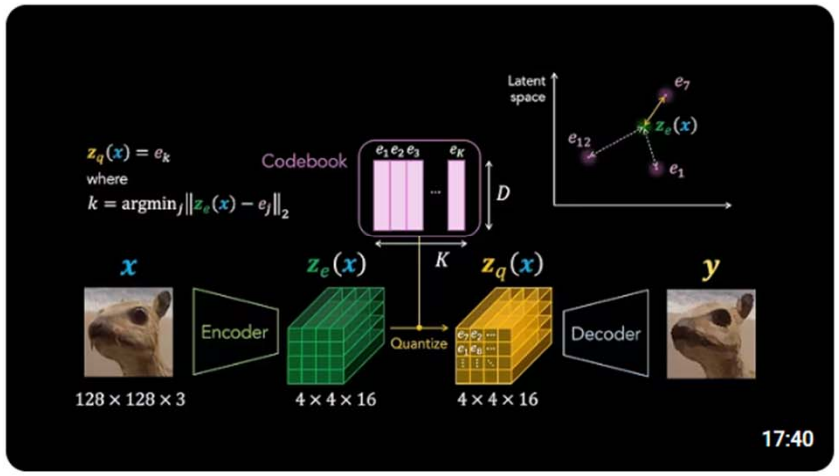


https://towardsdatascience.com/improving-vector-quantization-in-vector-quantized-variational-autoencoders-vq-vae-915f5814b5ce

What is VQ-VAE

https://www.youtube.com/watch?v=yQvELPjmyn0

- Before VQ-VAE: **Infinite** number of **continuous** latent vectors
- After VQ-VAE: choose from a **limited** collection, known as the **codebook**
- Strong constraint of the latent representation



Vector-Quantized Variational Autoencoders (VQ-VAEs) | Deep Learning

19K views • 1 year ago

DeepBean

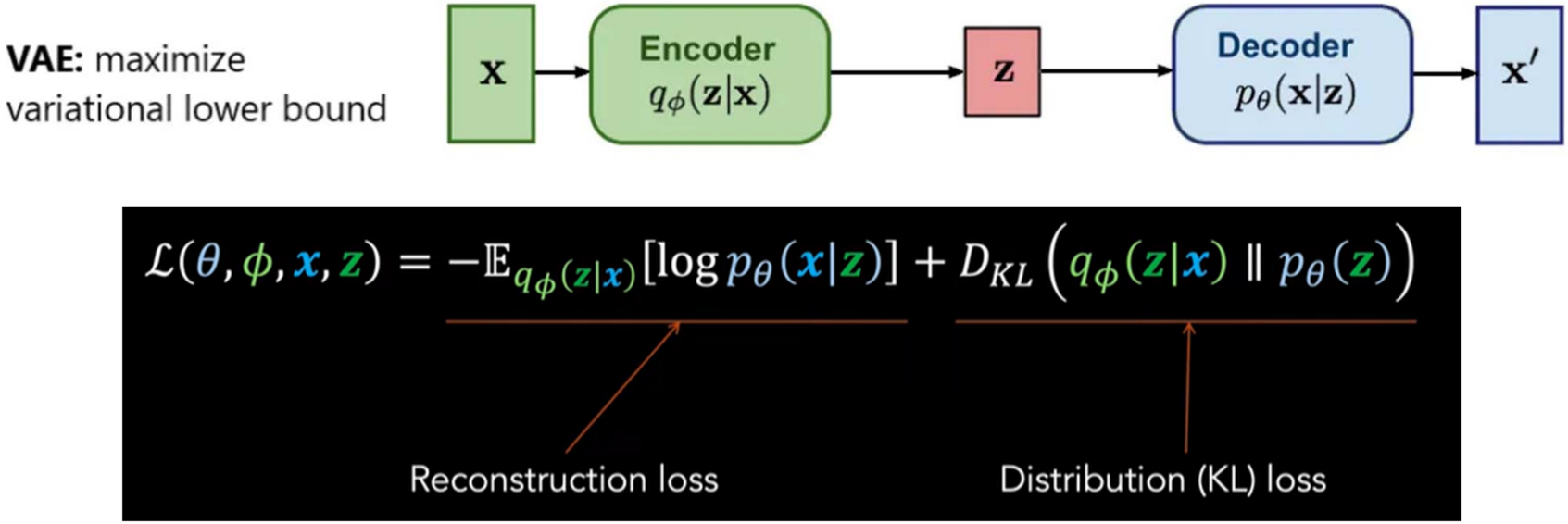
The Vector-Quantized Variational Autoencoder (VQ-VAE) forms discrete latent representat...

4K Lecture

12 chapters Introduction | VAE refresher | Quantization | Posterior |...

Varitaional Autoencoder (VAE) vs VQ-VAE

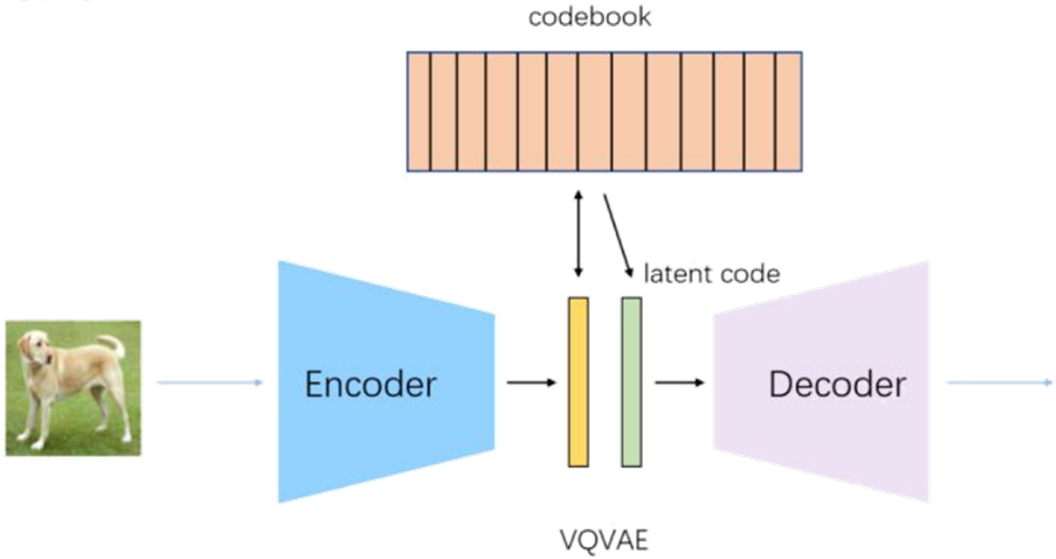
- VAE produces **continuous latent,** while VQVAE produces **discrete codes** (indices for a fixed number of latents)
- Both have an "**information bottleneck**"



(Figure from: https://pub.towardsai.net/diffusion-models-vs-gans-vs-vaes-comparison-of-deep-generative-models-67ab93e0d9ae)

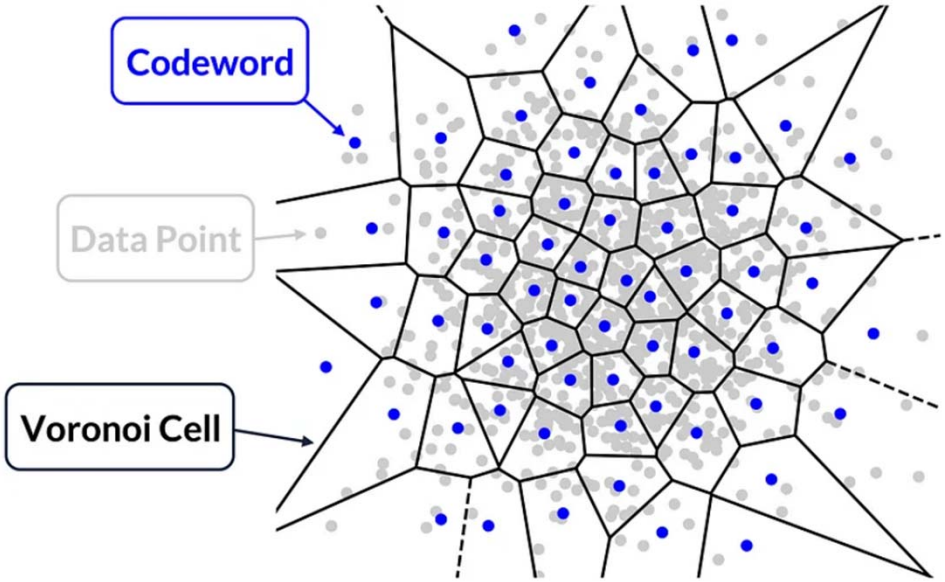
Codebook: Two Important Hyper-parameters

- The **dimension of the latent space**, *D*
  - we perform clustering in this latent space
  - the space can be different from the output space of the encoder
- The **number of unique latent vectors** (or codewords), *K*
  - the vocabulary size

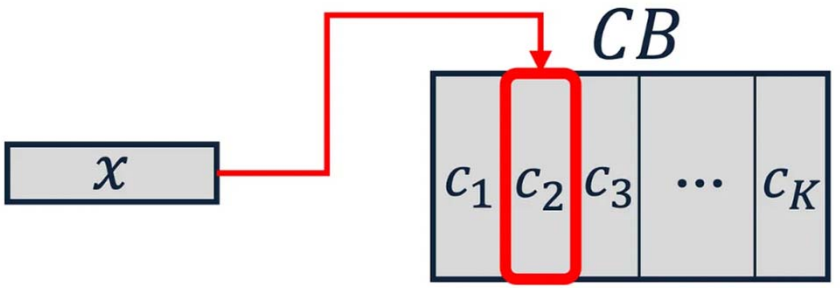


Vector Quantization

- Form a codebook by **clustering** in an embedding space from an encoder



Vector Quantization Operation (image by author)



$$x_{quantized} = c_{i^*} \; ; \; i^* = \arg\min_i \|x - c_i\|^2 \; ; \; i \in \{1, ..., K\}$$

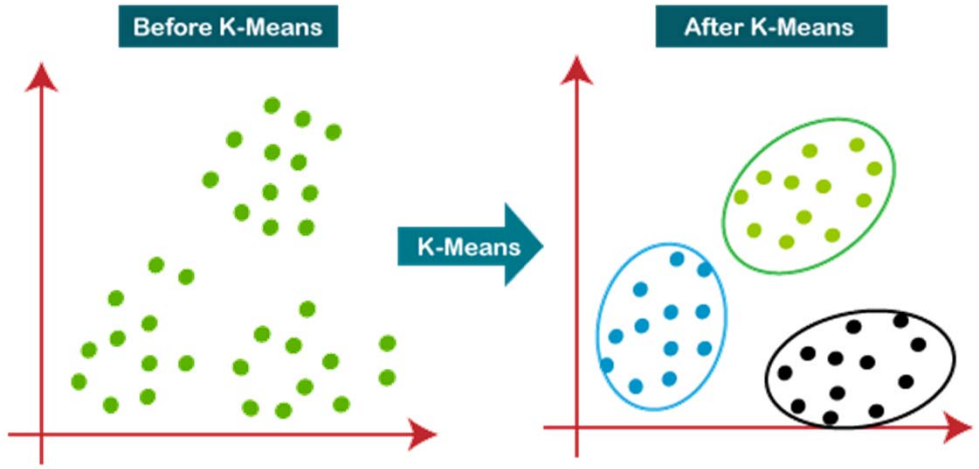
$$in\ this\ case : x_{quantized} = c_2$$

(Figure from: https://towardsdatascience.com/optimizing-vector-quantization-methods-by-machine-learning-algorithms-77c436d0749d)

K-Means Clustering

https://towardsdatascience.com/clear-and-visual-explanation-of-the-k-means-algorithm-applied-to-image-compression-b7fdc547e410

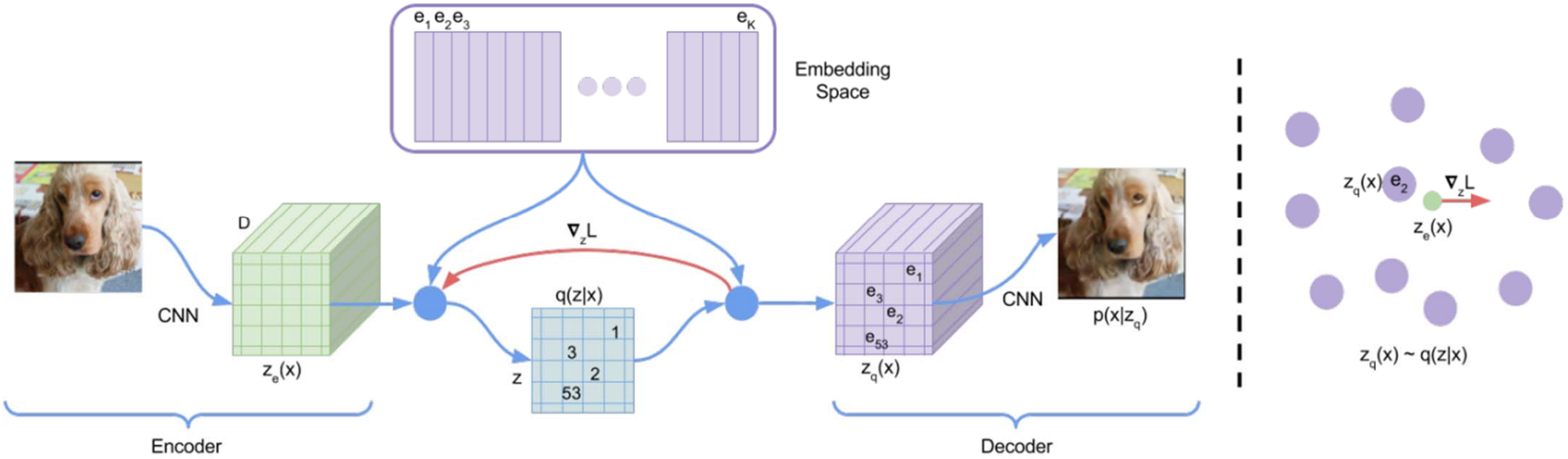
- K-Means algorithm
  - Initializing a set of cluster centroids
  - Assigning observations to clusters
  - Updating the clusters



https://keytodatascience.com/k-means-clustering-algorithm/

- Difference between K-Means and VQVAE
  - In K-Means, the feature space is ***given***
  - In VQVAE, the feature space is ***learned*** (i.e., the encoder)

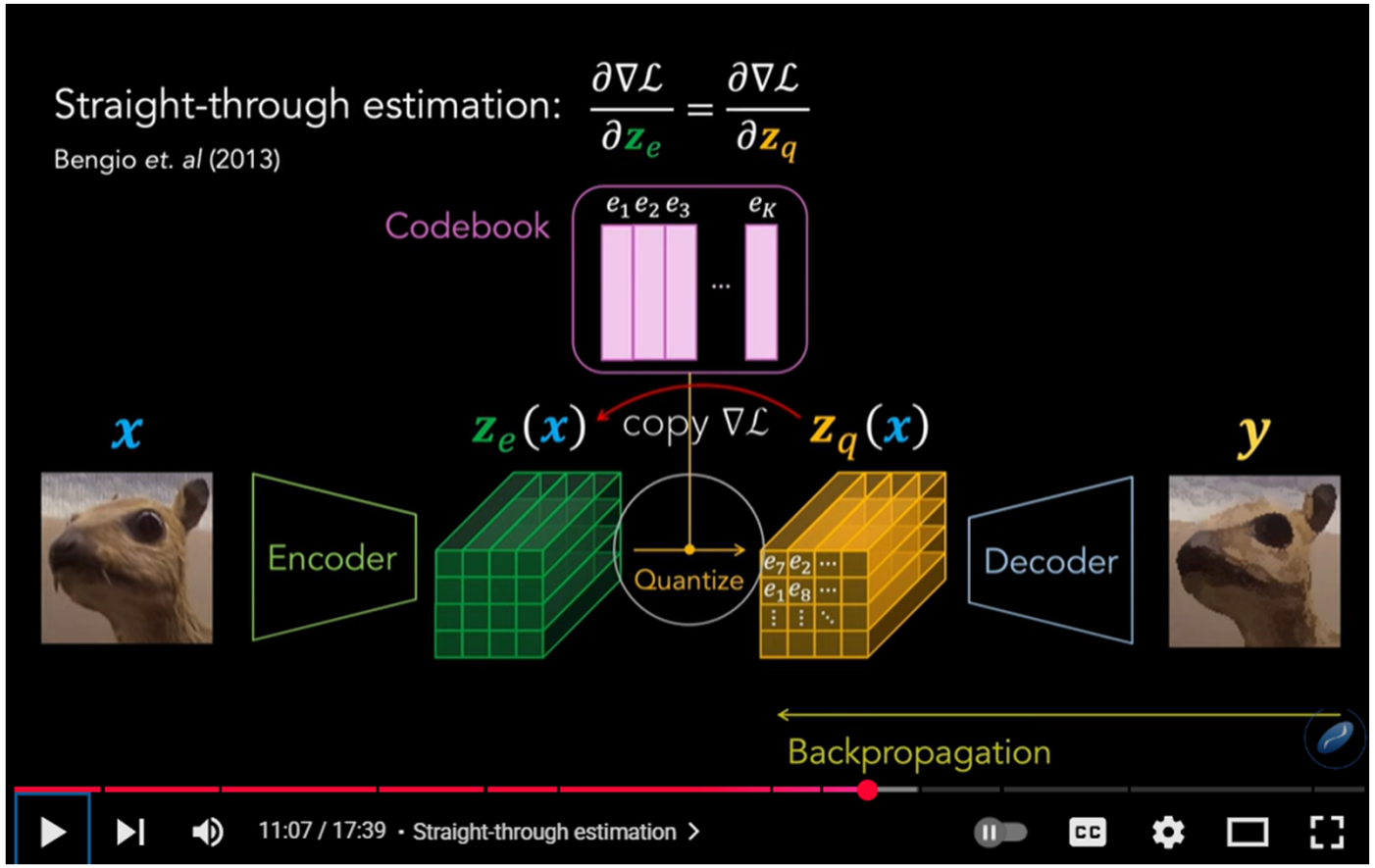
VQVAE



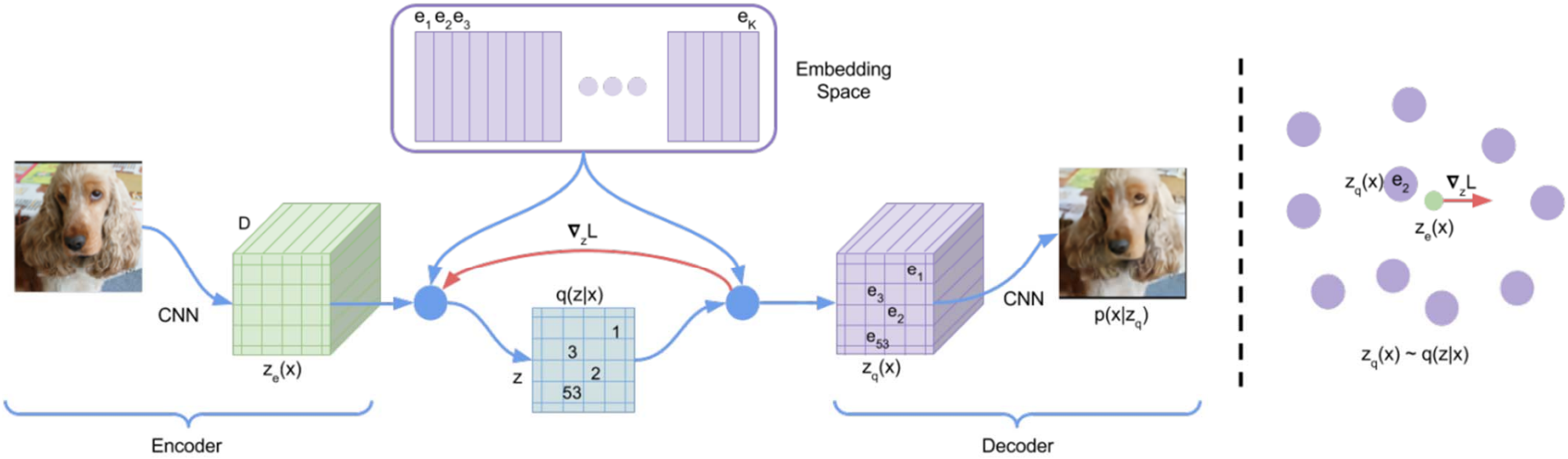
- Encoder output: $z_e(\mathbf{x})$
- VQ: find the codeword $\mathbf{e}$ from a learned codebook $\{\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_K\}$ that is closest to the encoder output to represent it
- Decoder input: $z_q(\mathbf{x}) \triangleq \mathbf{e}$

Straight-Through Estimation (STE)

- The quantization step is not differentiable



VQVAE



$$L = \log p(x|z_q(x)) + \|\text{sg}[z_e(x)] - e\|_2^2$$

Reconstruction loss

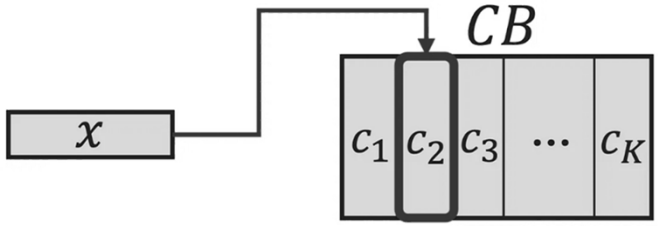
Optimizes the ***encoder*** and ***decoder***

Codebook loss

Optimizes the ***codebook*** (move the codewords towards the encoder outputs)

VQVAE: Summary

- **VQVAE produces discrete codes** (indices for a fixed number of latents)
  - **Encoder output**: an arbitrary continuous latent $q_\phi(\mathbf{z}|\mathbf{x})$
  - **Decoder input**: a *continuous* latent *selected* from the codebook
  - **Codebook size** K → only *K* possible decoder input
  - **The latent z** → one-hot index vector of dimension K

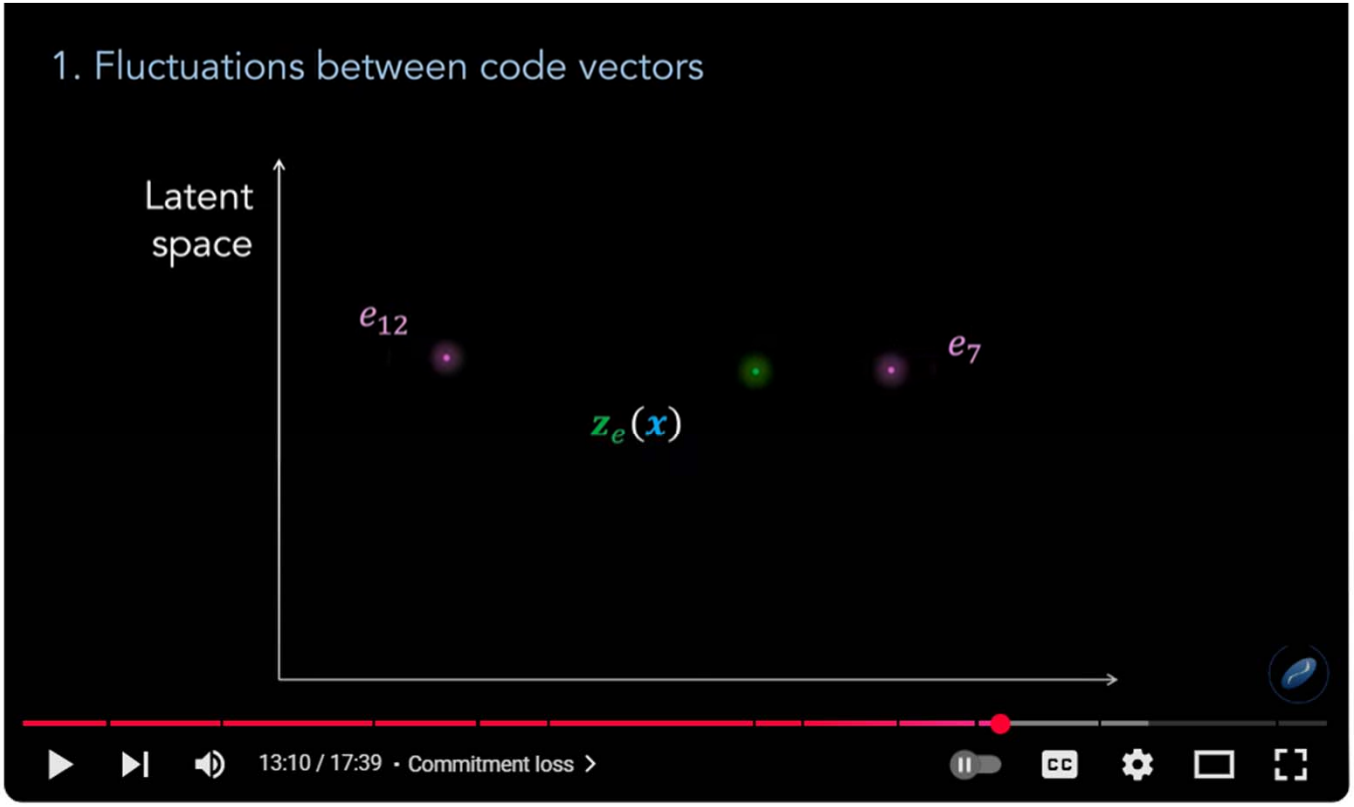


$$x_{quantized} = c_{i^*} \; ; \; i^* = \arg\min_i \|x - c_i\|^2 \; ; \; i \in \{1, ..., K\}$$

$$in\ this\ case : x_{quantized} = c_2$$

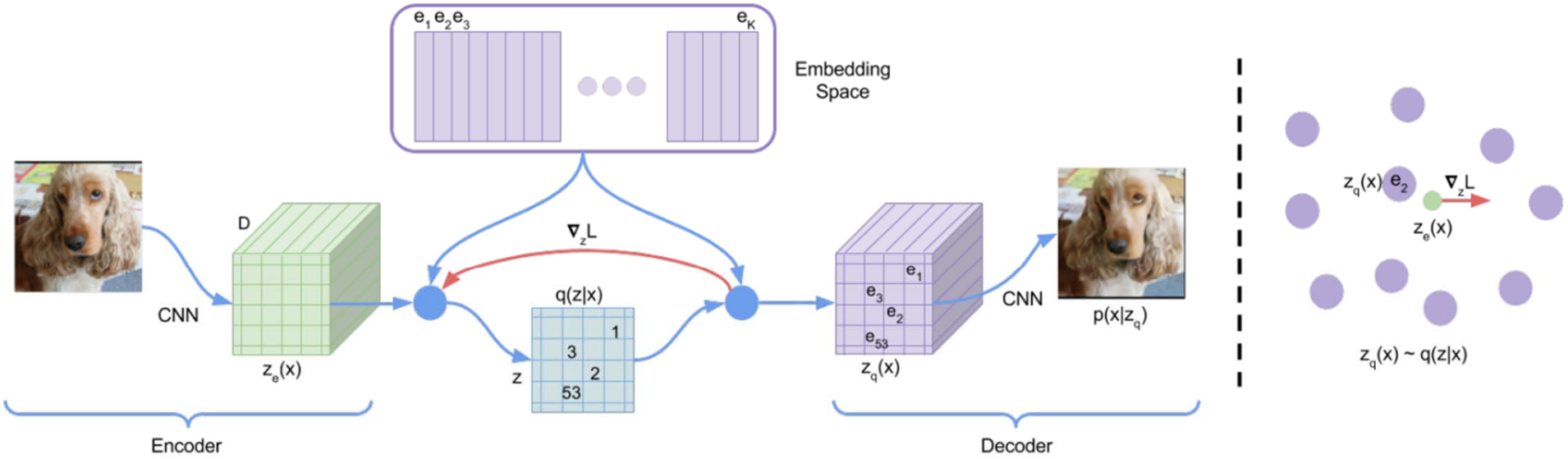
Issue 1: Unstable Training
(The Need of a "Commitment Loss")

- To avoid fluctuations between code vectors during training



Vector-Quantized Variational Autoencoders (VQ-VAEs) | Deep Learning

VQVAE



$$L = \log p(x|z_q(x)) + \|\mathrm{sg}[z_e(x)] - e\|_2^2 + \beta\|z_e(x) - \mathrm{sg}[e]\|_2^2,$$

Reconstruction loss

Optimizes the ***encoder*** and ***decoder***

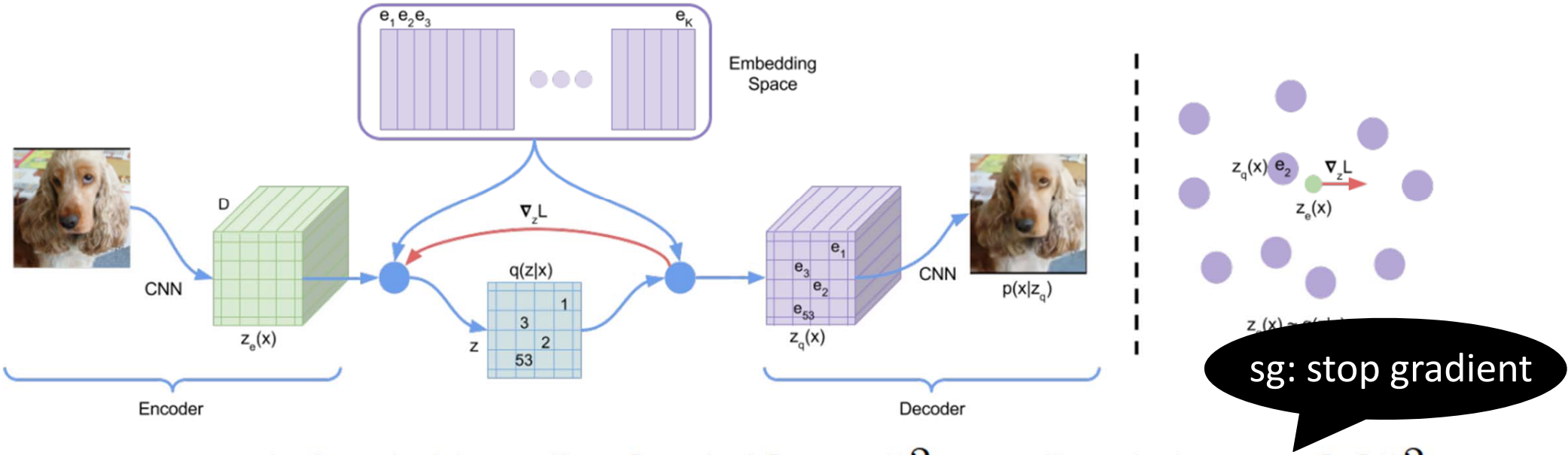
Codebook loss

Optimizes the ***codebook*** (move the codewords towards the encoder outputs)

commitment loss

Optimizes the ***encoder*** (move the encoder outputs towards the codewords)

VQVAE



$$L = \log p(x|z_q(x)) + \|\mathrm{sg}[z_e(x)] - e\|_2^2 + \beta\|z_e(x) - \mathrm{sg}[e]\|_2^2,$$

Reconstruction loss

Codebook loss

commitment loss

Decoder needs to recover the input from a quantized latent

Commitment loss helps improves convergence because the encoder output would “commit” to some codewords instead of changing the codewords too frequently

VQVAE Training Tips: Exponential Moving Average (EMA)

- Instead of directly updating codebook vectors with encoder gradients
- We do **"soft memory" of assignments:** accumulates information about which latent vectors get assigned to a code over time

$$N_i^{(t)} := N_i^{(t-1)} * \gamma + n_i^{(t)}(1-\gamma), \quad m_i^{(t)} := m_i^{(t-1)} * \gamma + \sum_j^{n_i^{(t)}} E(x)_{i,j}^{(t)}(1-\gamma), \quad e_i^{(t)} := \frac{m_i^{(t)}}{N_i^{(t)}}$$

$$L = \log p(x|z_q(x)) + \|\mathrm{sg}[z_e(x)] - e\|_2^2 + \beta\|z_e(x) - \mathrm{sg}[e]\|_2^2,$$

Codebook loss

Optimizes the ***codebook***

(move the codewords towards the encoder outputs)

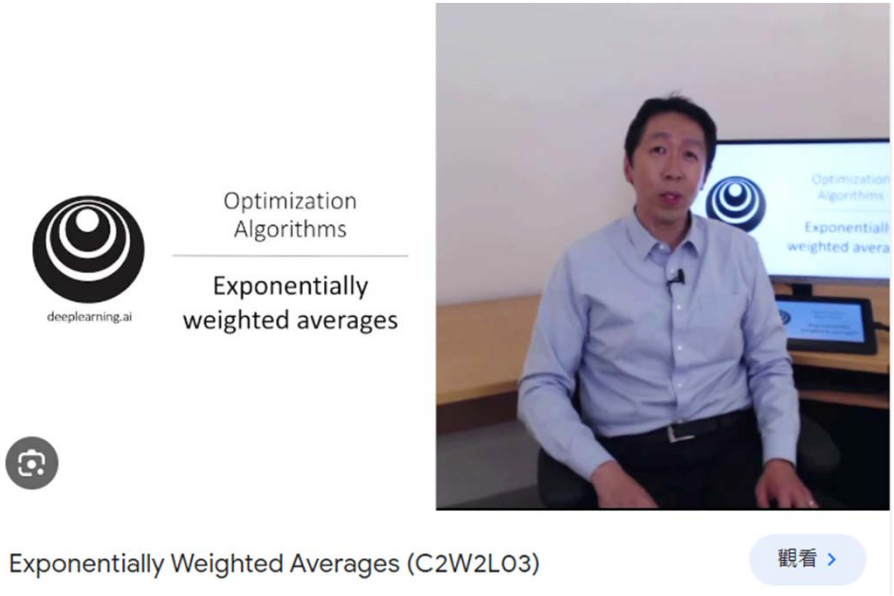
Ref: Razavi et al, "Generating diverse high-fidelity images with VQ-VAE-2," NeurIPS 2019

VQVAE Training Tips: Exponential Moving Average (EMA)

$$L = \log p(x|z_q(x)) + \|\mathrm{sg}[z_e(x)] - e\|_2^2 + \beta\|z_e(x) - \mathrm{sg}[e]\|_2^2,$$

Reconstruction loss | Codebook loss | commitment loss

(EMA update)



https://www.youtube.com/watch?v=lAq96T8FkTw

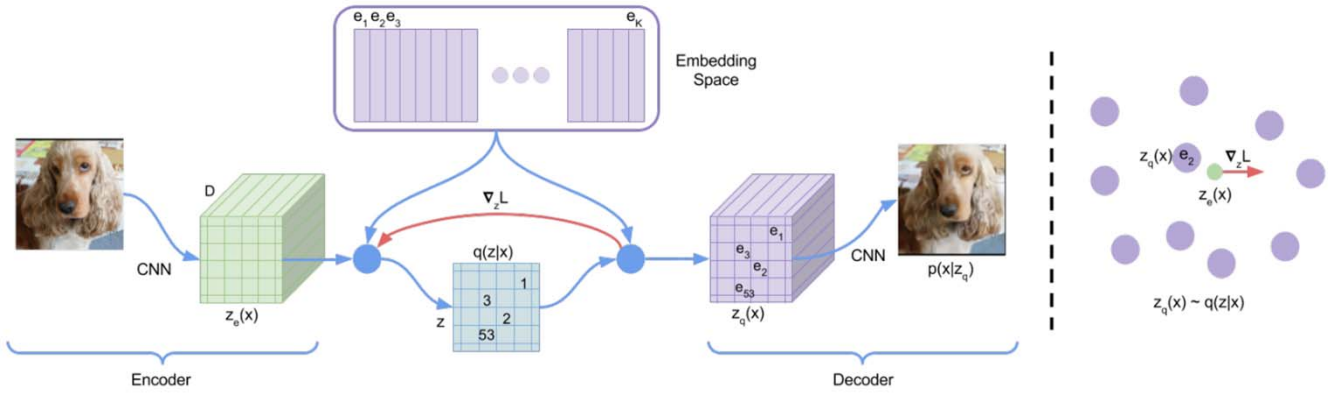
pedrocg42 commented on Oct 27, 2022 Author

Right now I am following **@lucidrains** recommendation of not using any commitmen_loss and just using EMA to update the VQ codebooks. At the time I tried several configurations of both decay and commitment_cost with the same outcome,

Some people said that we don't need the commitment loss if we use EMA
(https://github.com/lucidrains/vector-quantize-pytorch/issues/27)

Issue 2: Low Codeword Utilization Rate

- **Codebook/index collapse:** Some codewords are never used at all (dead)
- Solution 1: choose a **smaller codebook size**
- Solution 2: choose a **smaller latent dimension**
  - *perform clustering in a lower-dimensional space*
  - the encoder values are projected down and then projected back to a high-dimensional after quantization

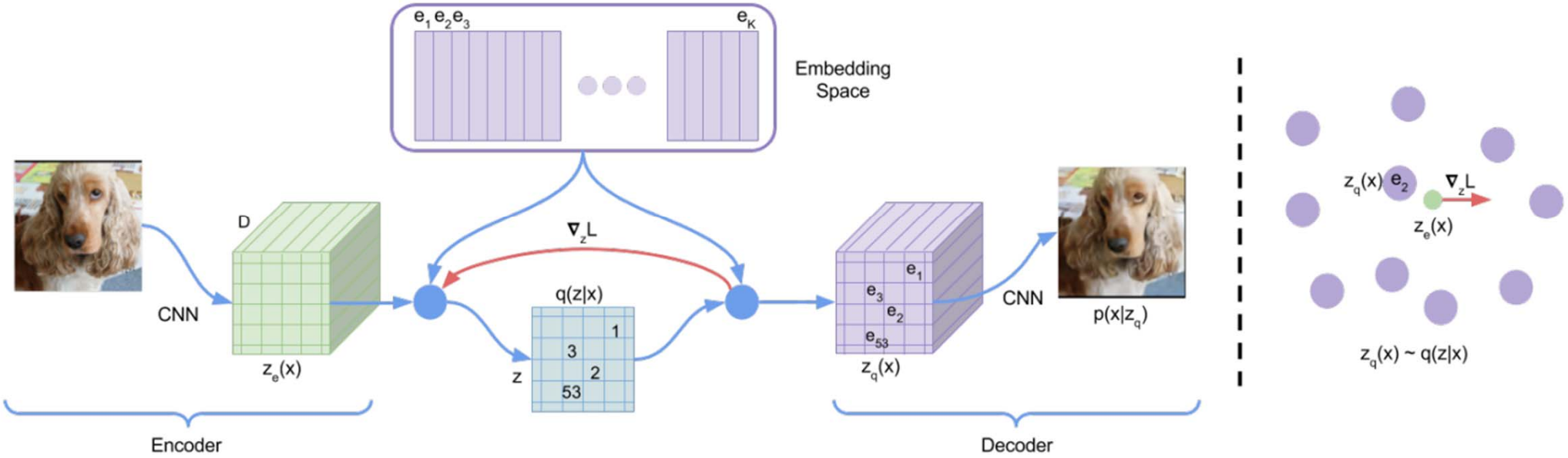


Issue 3: Poor Reconstruction Quality

- Increase codebook size or latent dimensionality
- Use **multiple codebooks**
  - hierarchical VQ-VAEs
  - Residual VQs

Image Tokenization by VQVAE

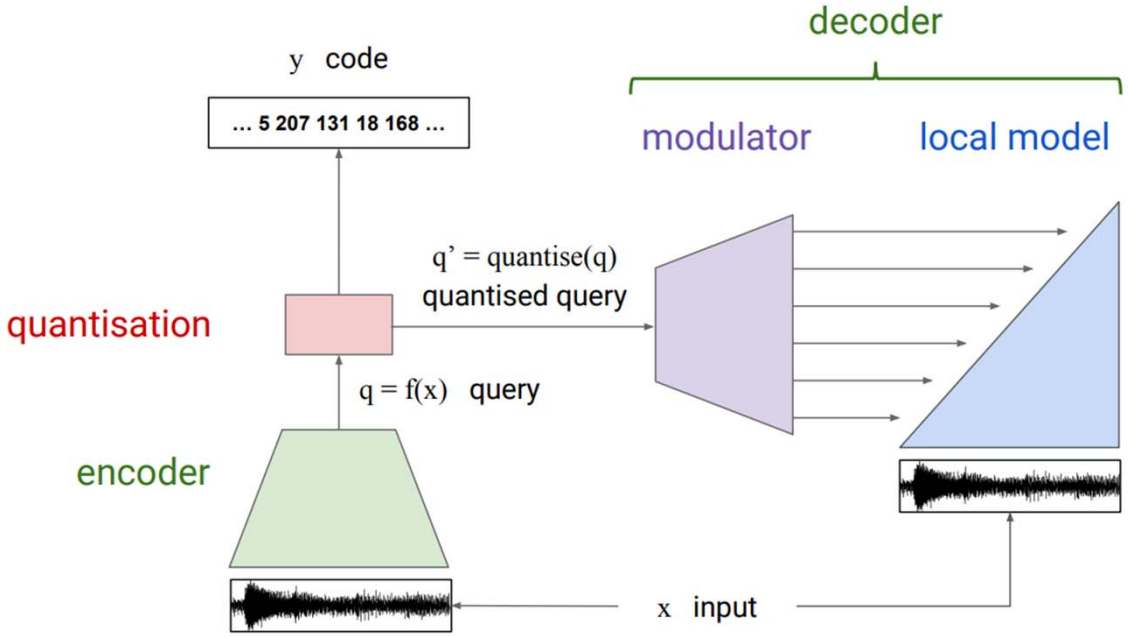
- We can obtain **discrete tokens of continuous data** by doing **vector quantization** at the output of an image **encoder** (Oord et al., 2017)



Ref: Oord et al, "Neural discrete representation learning, NeurIPS 2017

Audio Tokenization by VQVAE

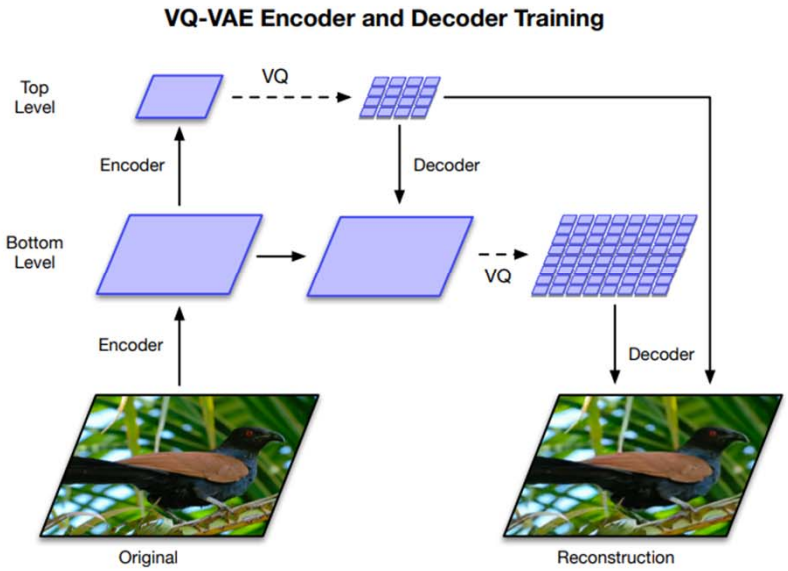
- Similarly, we can do **vector quantization** at the output of an audio **encoder** (Dieleman et al., 2018)

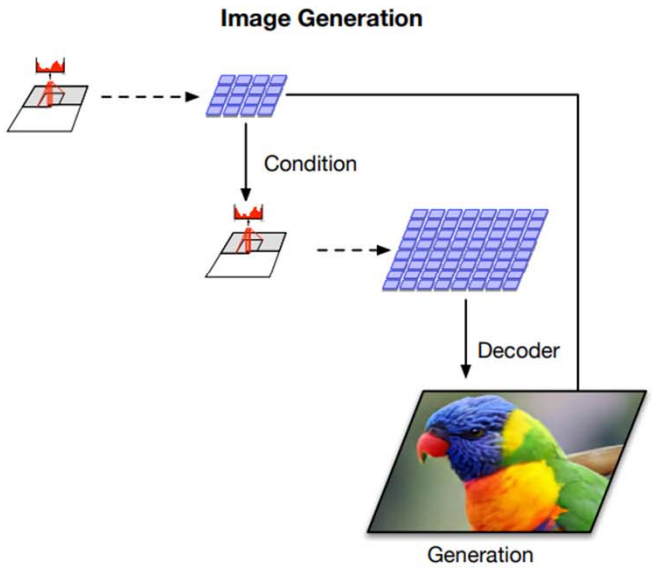


Ref: Dieleman et al, "The challenge of realistic music generation: modelling raw audio at scale," NeurIPS 2018

VQVAE 2: Hierarchical VQ

- **Two codebooks**
  - the input to the model is a 256×256 image that is compressed to quantized latent maps of size 64×64 and 32×32 for the bottom and top levels, respectively
  - the decoder reconstructs the image from the two latent maps.

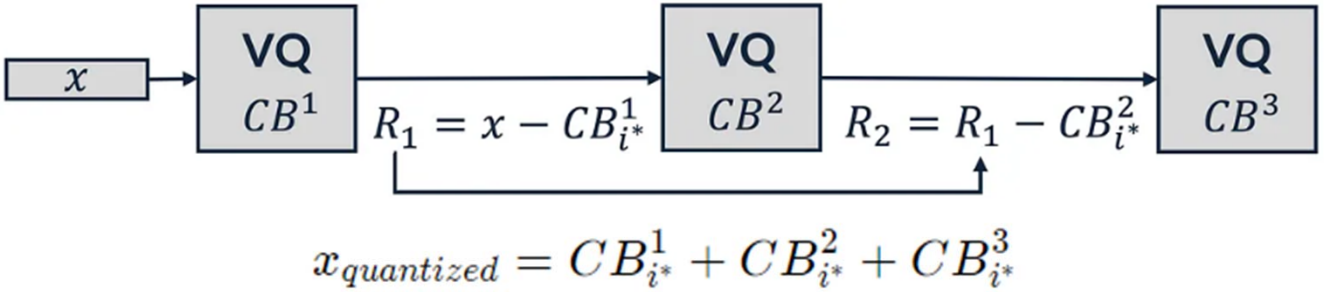




Ref: Razavi et al, "Generating diverse high-fidelity images with VQ-VAE-2," NeurIPS 2019

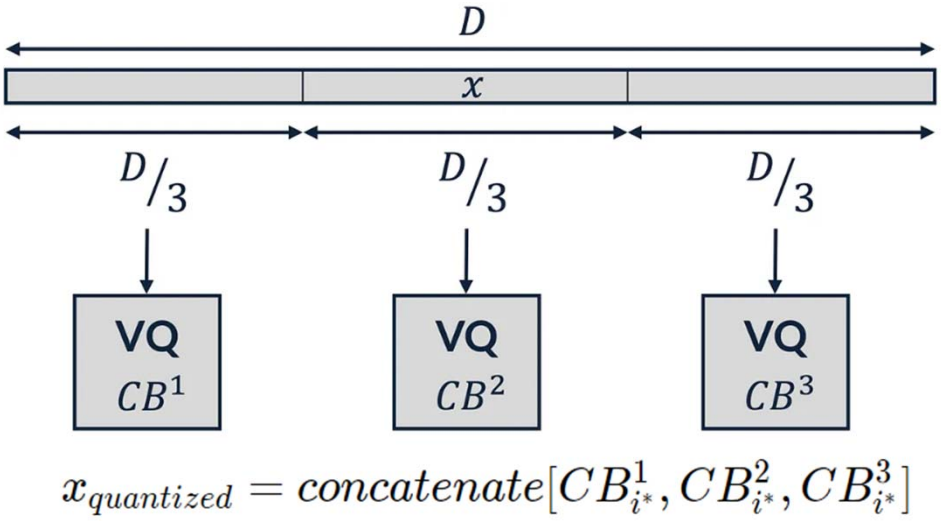
Variants of VQ

- Residual VQ (**RVQ**)



- Product VQ (PVQ)

  (a.k.a., decomposed VQ; DVQ)



- More… (e.g., additive VQ)

(Figure from: https://towardsdatascience.com/optimizing-vector-quantization-methods-by-machine-learning-algorithms-77c436d0749d)

VQVQE Library

https://github.com/lucidrains/vector-quantize-pytorch

vector-quantize-pytorch

- Initialization
- **Lower codebook dimension**
- Cosine similarity
- **Expiring stale codes**
- Orthogonal regularization loss
- Multi-headed VQ
- Random Projection Quantizer
- Finite Scalar Quantization
- Lookup Free Quantization

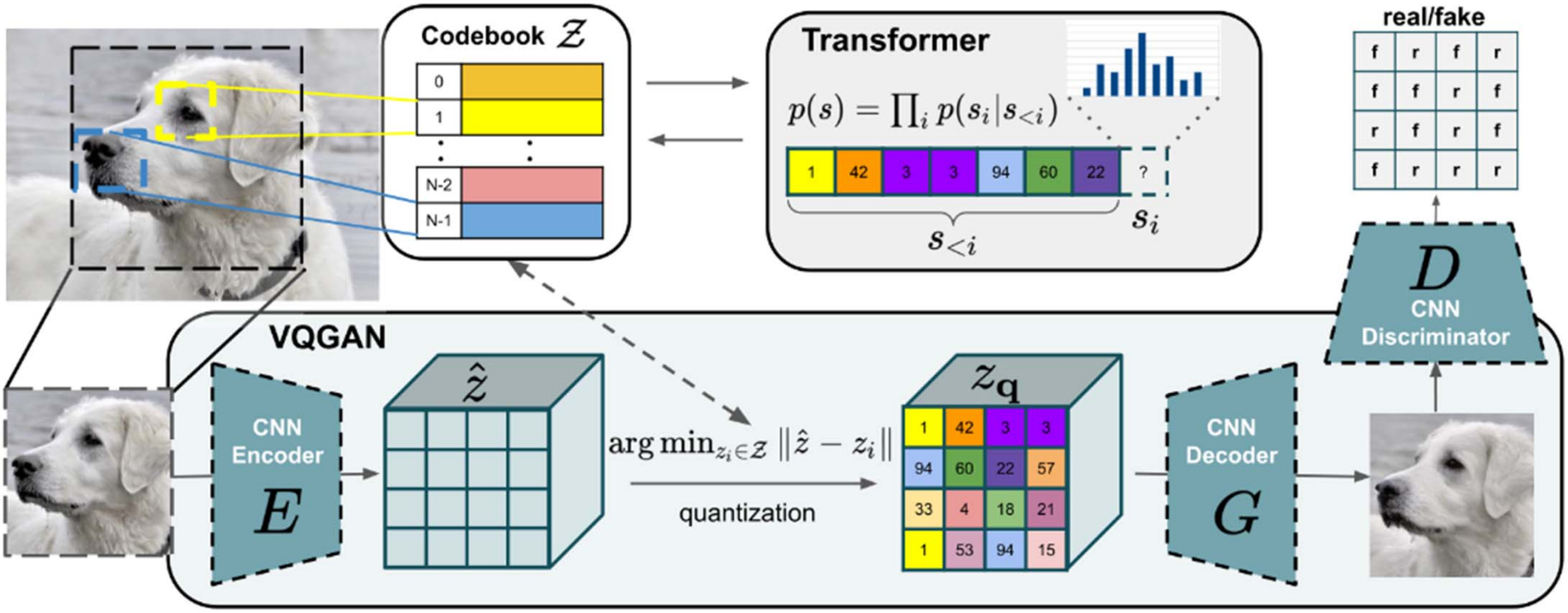
Summary

- We can use **VQVAE** to **convert continuous data to discrete tokens**
- What we get from VQVAE: **encoder**, **decoder**, and **codebook**

- We can then train a Transformer **LM** over the codeword sequences
  - Image generation: VQGAN
  - Music generation: Jukebox, KaraSinger, JukeDrummer, SingSong

- The VQVAE and the Transformer LM are trained separately

VQVAE-based Image Generation: VQGAN

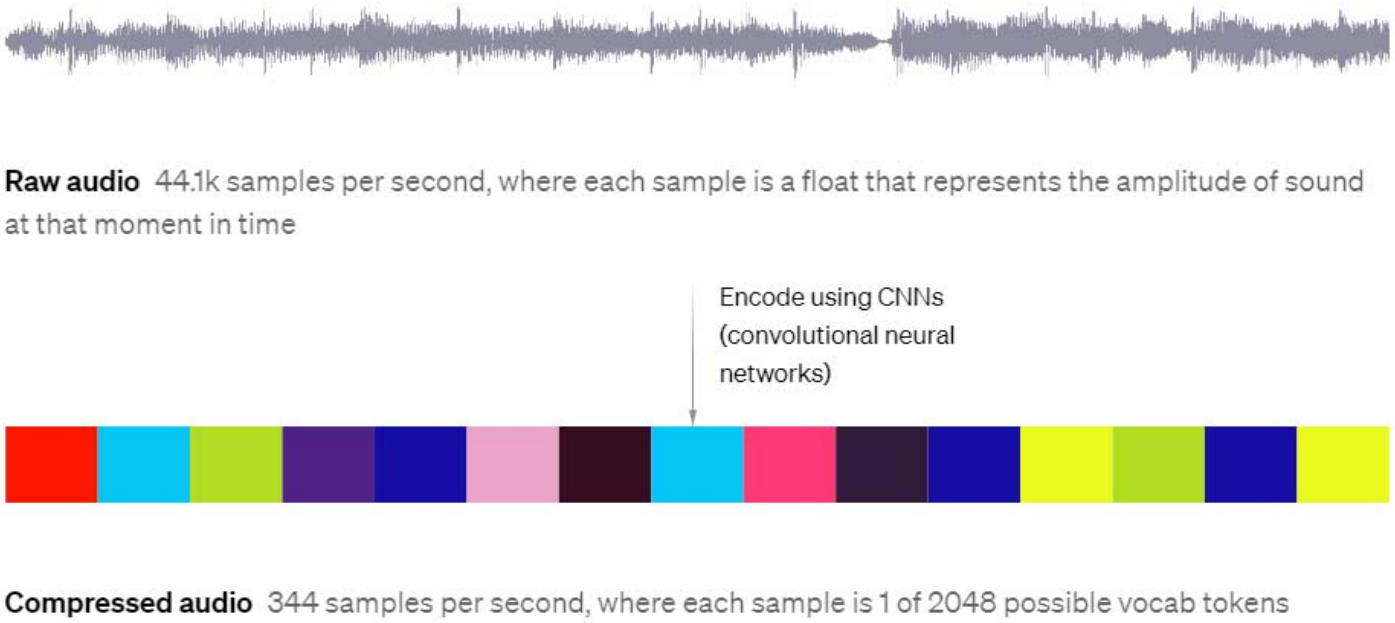


Ref: Esser et al, "Taming Transformers for high-resolution image synthesis," CVPR 2021
Ref: Yu et al, "Vector-quantized Image Modeling with Improved VQGAN," ICLR 2022

VQVAE-based Music Generation: JukeBox

https://openai.com/research/jukebox

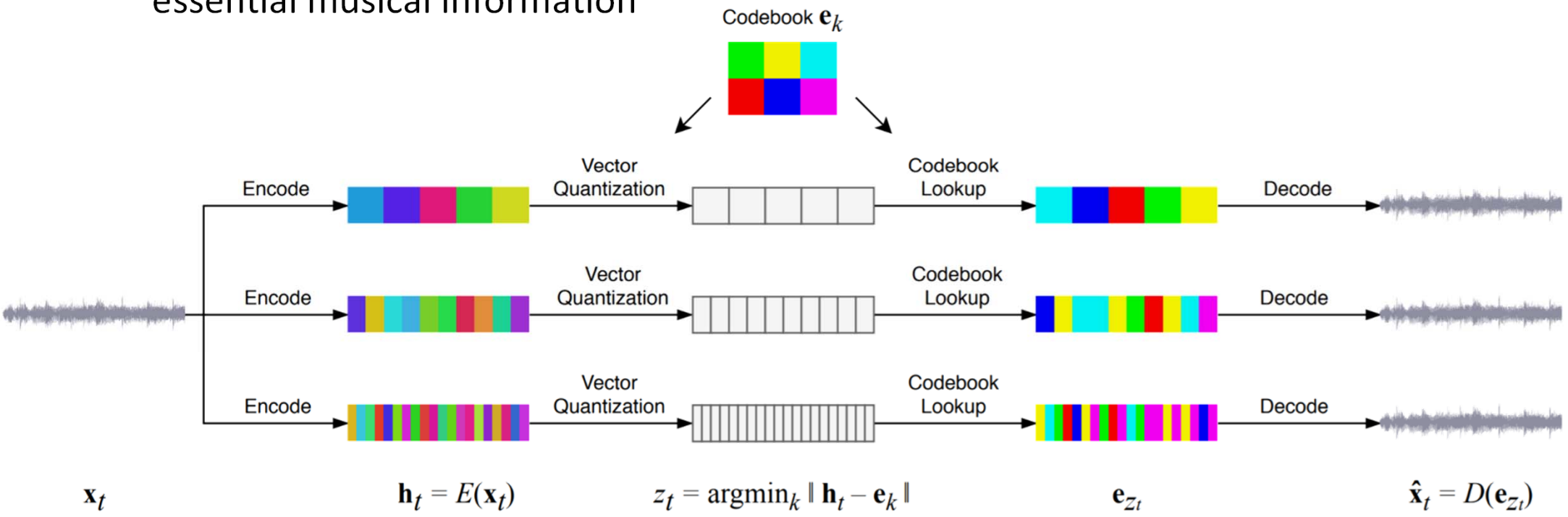
- Use **vector quantization** (VQ) to model waveforms as "**acoustic tokens**"



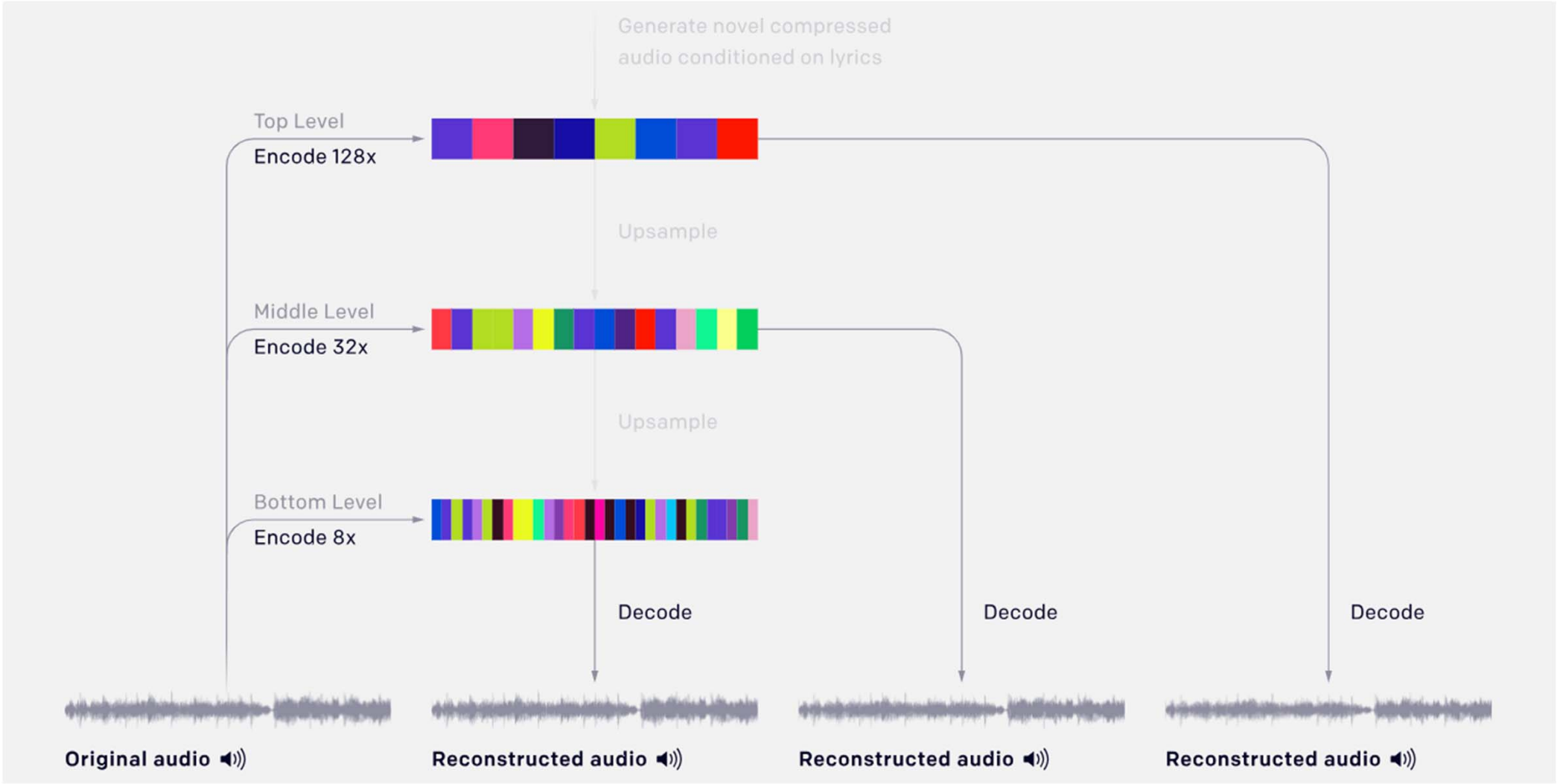
Ref: Dhariwal et al, "Jukebox: A generative model for music," arXiv 2020

VQVAE-based Music Generation: JukeBox

- Three layers of VQ for fidelity (like VQVAE 2): **top**, **middle**, **bottom**
  - Each VQ-VAE level independently encodes the input
  - **Bottom** produces the highest quality reconstruction, while **top** retains only the essential musical information

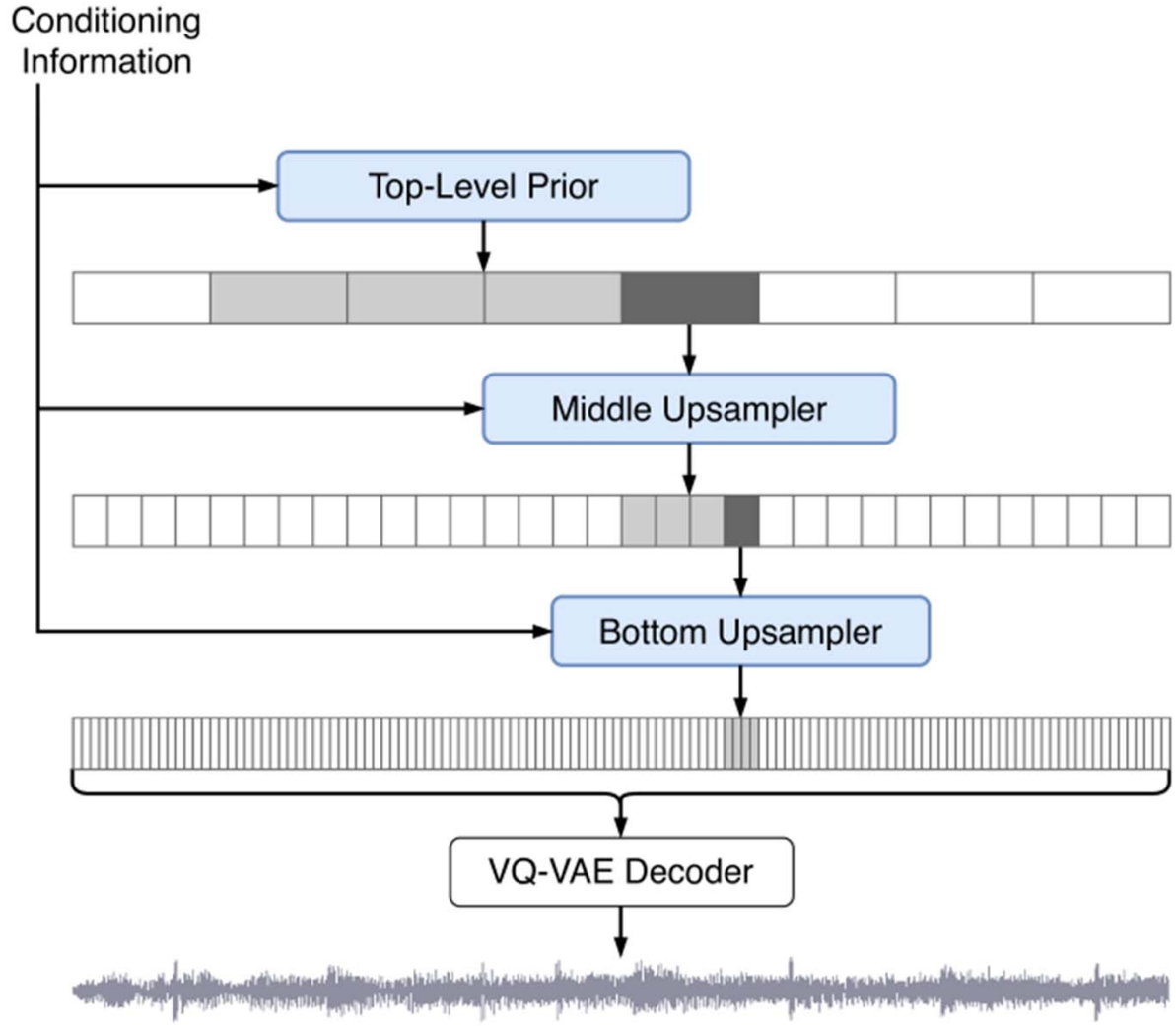


VQVAE-based Music Generation: JukeBox

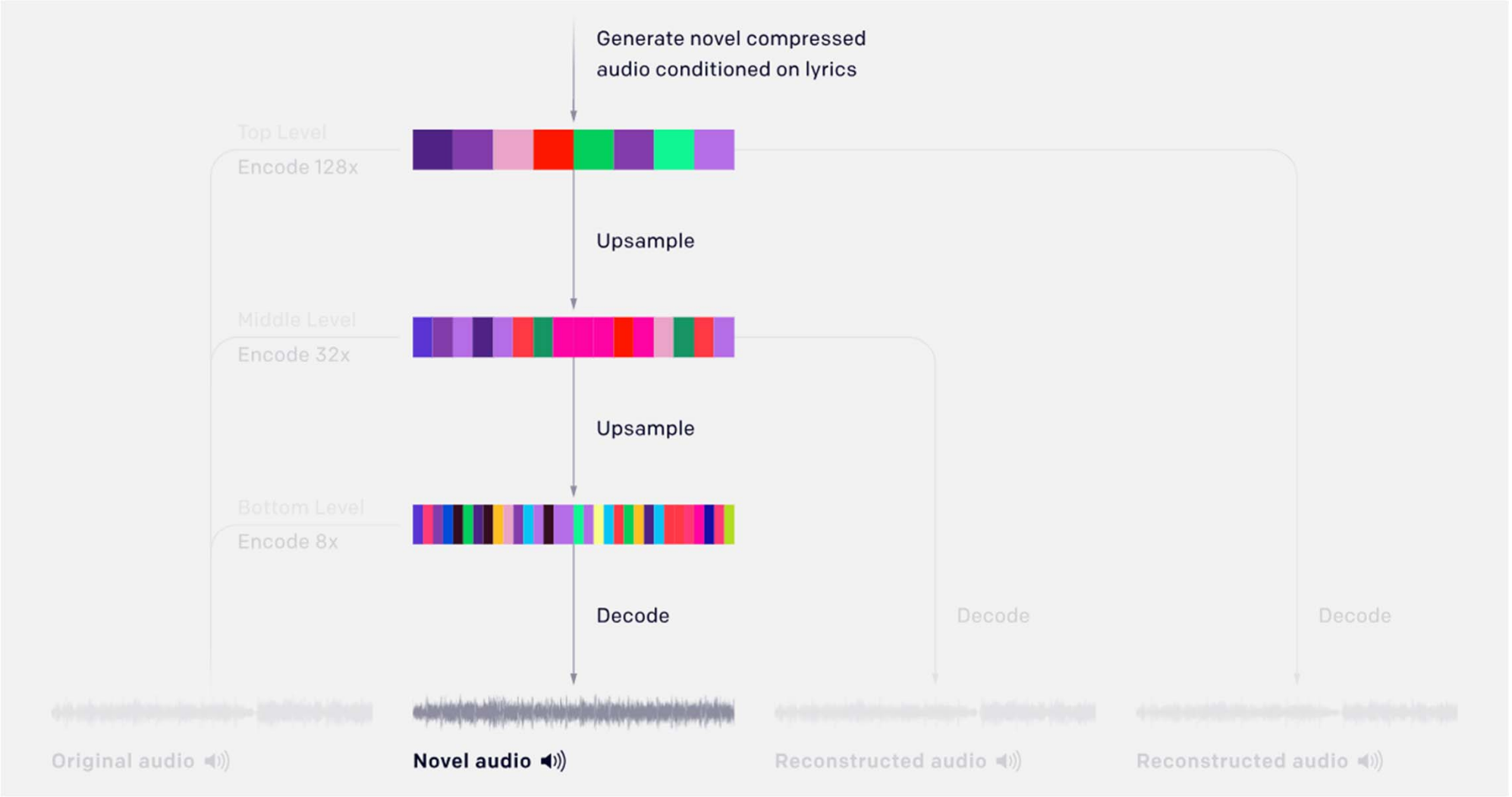


VQVAE-based Music Generation: JukeBox

- Take **lyrics** as condition
  - Given lyrics, generate **top-level** tokens
  - Given lyrics+top, generate **mid-level** tokens
  - Given lyrics+mid, generate **bottom-level** tokens
- Generate music waveforms that contain both **vocal** and **instrumental background** (mixed together)

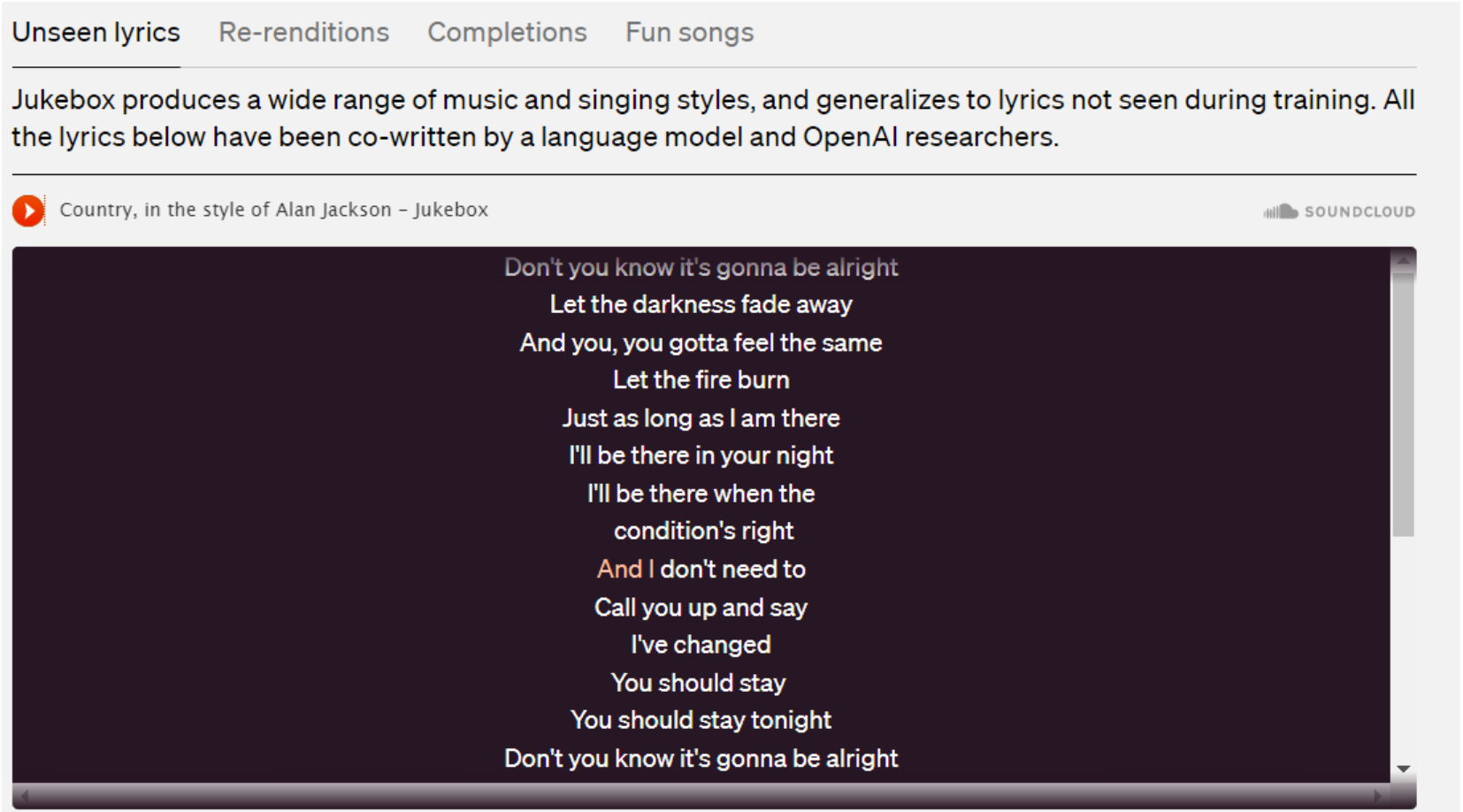


VQVAE-based Music Generation: JukeBox



VQVAE-based Music Generation: JukeBox

https://openai.com/research/jukebox

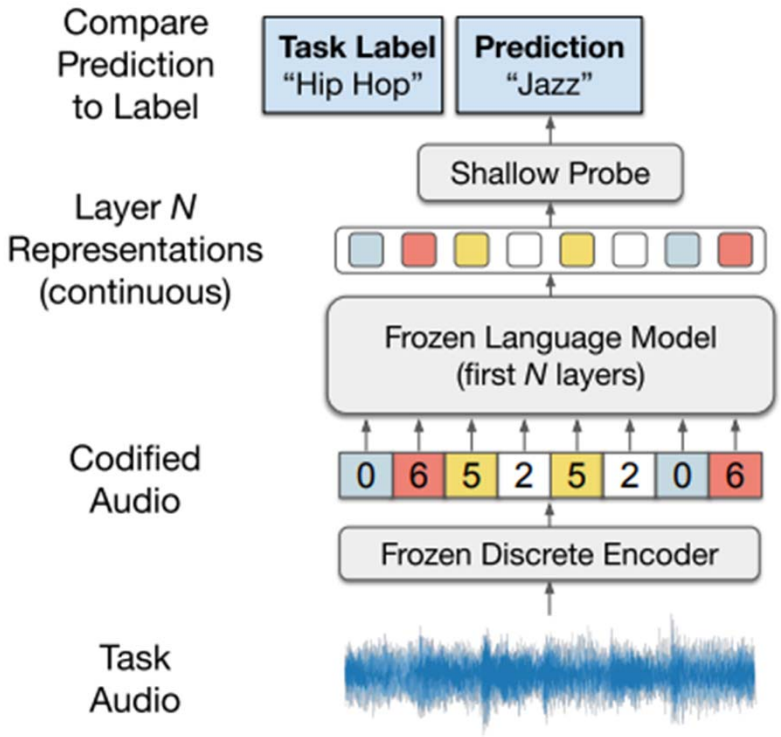


ps. JukeBox as a Music Foundation Model

- For **LALM** (e.g., LLark)

	LLARK
LLM	Llama2-7b
audio encoder	Jukebox
audio token frequency(Hz)	10
max train duration(s)	25
tune modules	projection, LLM

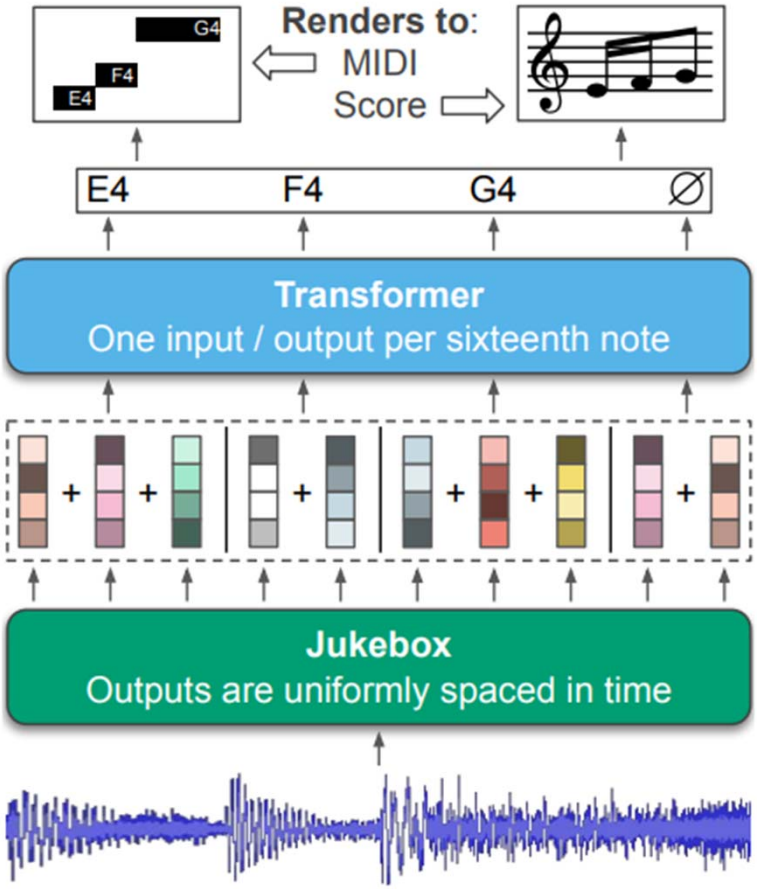
- For **music classification** (JukeMIR; ISMIR 2021)



ps. JukeBox as a Music Foundation Model

https://github.com/chrisdonahue/sheetsage

- For **lead sheet transcription** (SheetSage; ISMIR 2022)
  - SOTA model for transcribing the melody and chord progression
  - Do transfer learning
  - Use **Transformer** to learn the language model (LM) for melody and chords
  - Heavy but accurate



Ref: Donahue et al., "Melody transcription via generative pre-training," ISMIR 2022

Issues of OpenAI's Jukebox

- **GPU-intensive**
  - **1M** proprietary songs as training data & **512 V100s** & total training time **>6 weeks** (i.e., around >60 years of single V100 time)

trained on 128 V100s for 2 weeks, and the top-level prior has 5 billion parameters and is trained on 512 V100s for 4 weeks. We use Adam with learning rate 0.00015 and weight decay of 0.002. For lyrics conditioning, we reuse the prior and add a small encoder, after which we train the model on 512 V100s for 2 weeks. The detailed hyperparameters for our models and training are provided in Appendix B.3.

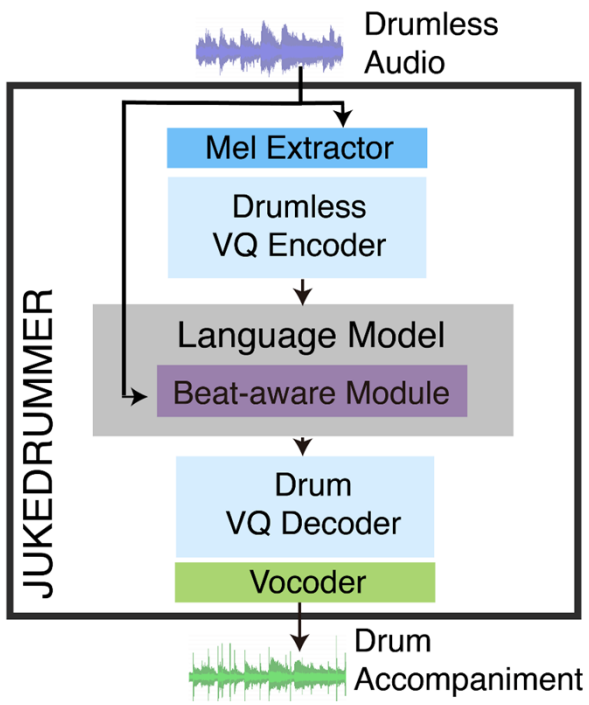
The current model takes around an hour to generate 1 minute of top level tokens. The upsampling process is very slow, as it proceeds sequentially through the sample. Currently it takes around 8 hours to upsample one minute of top level tokens. We can create a human-in-the-loop co-composition process at the top level only, using the VQ-VAE decoders

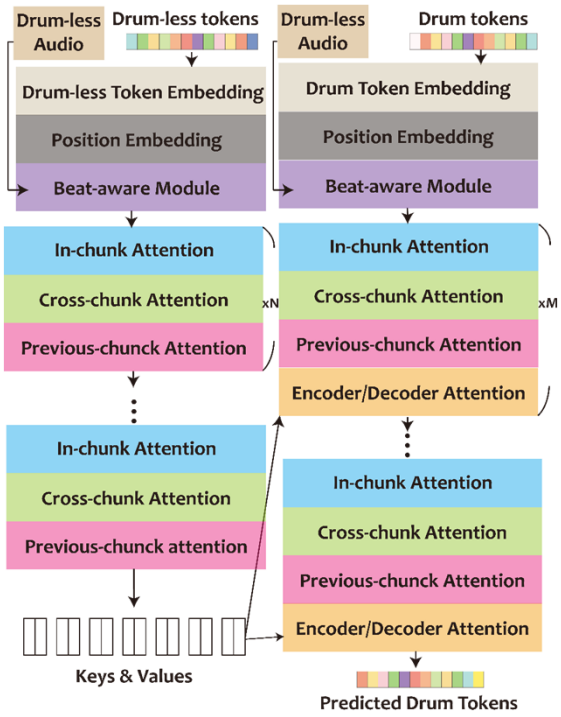
- **Low controllability**
  - The only control signal is the lyrics

Appendix: JukeDrummer, VQVAE on Mel-spectrograms

https://legoodmanner.github.io/jukedrummer-demo/

- **Given no-drum music, generate drum accompaniment**
  - Apply blind source separation to get pairs of drum/no-drum
  - Seq2seq: **no-drum tokens** as the source, **drum** tokens as the target to be generated
- Using Mel-spectrogram
  - Train on **1** GTX 1080-Ti for **2** days





Ref: Wu et al, "JukeDrummer: Conditional beat-aware drum accompaniment generation in the audio domain using Transformer VQ-VAE," ISMIR 2022

BTW, the Pursuit of "Audio Words" is Actually Not New…

Scale	Formulation	Notation	Dimension
Frame level (alphabet; e.g., 46 ms) (elementary unit)	$\hat{\alpha}_{\mathbf{s,t}} = \arg\min_{\alpha_{s,t}} \|\mathbf{x}_{s,t} - \mathbf{D}_1\alpha_{s,t}\|_2^2 + \lambda f(\alpha_{s,t})$	$\mathbf{D}_1$: first-layer codebook of audio-alphabets $\mathbf{x}_{s,t}$: frame-level acoustic feature $\alpha_{s,t}$: frame-level encoding over audio-alphabets	$\mathbf{D}_1 \in \mathbb{R}^{m\times k_1}$ $\mathbf{x}_{s,t} \in \mathbb{R}^m$ $\alpha_{s,t} \in \mathbb{R}^{k_1}$
Segment level (word; e.g., 2.5 sec) (combination of alphabets)	$\hat{\beta}_{\mathbf{s}} = \arg\min_{\beta_s} \|\sum_t \alpha_{s,t} - \mathbf{D}_2\beta_s\|_2^2 + \lambda f(\beta_s)$	$\mathbf{D}_2$: second-layer codebook of audio-words level $\overline{\alpha}_s = \sum_t \alpha_{s,t}$: pooled frame-level encoding β_s: segment-level encoding over audio-words	$\mathbf{D}_2 \in \mathbb{R}^{k_1\times k_2}$ $\overline{\alpha}_s \in \mathbb{R}^{k_1}$ $\beta_s \in \mathbb{R}^{k_2}$
Clip level (document; e.g., 30 sec)	$\gamma_1 = \sum_s \overline{\alpha}_s = \sum_{s,t} \alpha_{s,t}$ $\gamma_2 = \sum_s \beta_s$	γ_1: bag-of-audio alphabet (BoAA) γ_2: bag-of-audio word (BoAW)	$\gamma_1 \in \mathbb{R}^{k_1}$ $\gamma_2 \in \mathbb{R}^{k_2}$

- Codebook construction methods
  - Exemplar-based method
  - k-means
  - Online dictionary learning
- Encoding methods
  - l_1-regularized sparse coding
  - Top- Vector Quantization
  - Triangle k-means

TABLE III
VARIANTS OF TERM FREQUENCY-INVERSE DOCUMENT FREQUENCY

Abbr.	Formulation	Abbr.	Formulation
tf_b	$f_{d,t}$	idf_b	$\ln \|\mathcal{D}\|/F_t$
tf_{l1}	$\ln(1+f_{d,t})$	idf_p	$\ln (\|\mathcal{D}\| - F_t)/F_t$
tf_{l2}	$1+\log_2 f_{d,t}$	idf_{e1}	$(\max_{t'\in\mathcal{T}} n_{t'}) - n_t$
tf_{o1}	$\frac{f_{d,t}}{f_{d,t}+\kappa\|d\|/\|\Delta d\|}$	idf_{e2}	$1 - \frac{n_t}{\ln\|\mathcal{D}\|}$
tf_{o2}	$\frac{(\kappa+1)f_{d,t}}{f_{d,t}+\kappa\left((1-b)+b\cdot\frac{\|d\|}{\|\Delta d\|}\right)}$	idf_o	$\ln\left(\frac{\|\mathcal{D}\|-F_t+0.5}{F_t+0.5}\right)$

$f_{d,t}$ — the number of occurrences of term t in document d
F_t — the number of documents in $\mathcal{D}$ that contain t
$|d|$ — cardinality (length) of d; $|\Delta d|$ — average document length in $\mathcal{D}$
n_t — entropy of term t in $\mathcal{D}$; $n_t = -\sum_d (f_{d,t}/F_t)\ln(f_{d,t}/F_t)$
$\mathcal{T}$ — the universe of terms; $\mathcal{D}$ — the universe of documents

Ref 1: Yeh et al, "AWtoolbox: Characterizing audio information using audio words," MM 2014

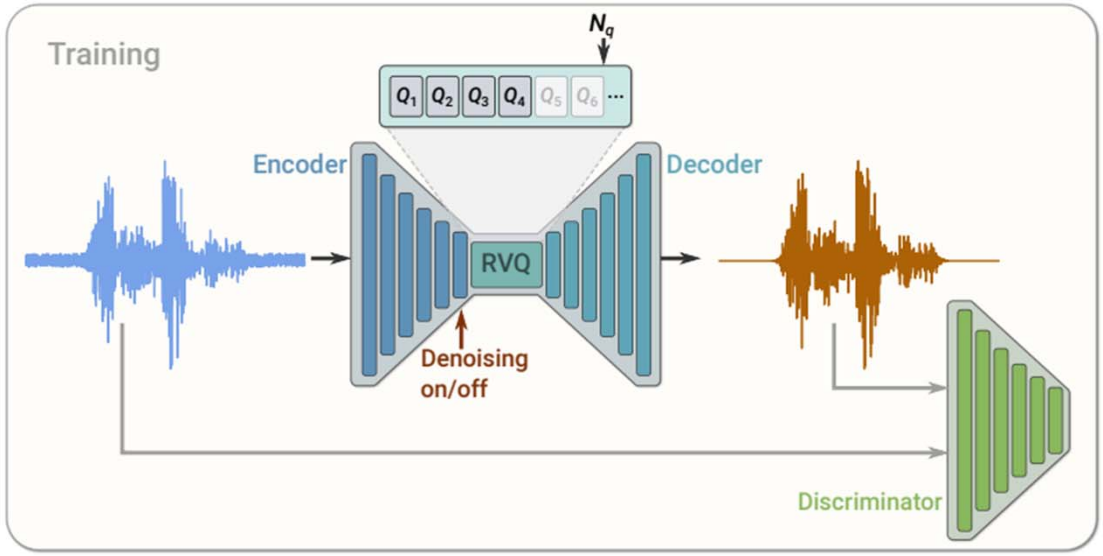
Ref 2: Su et al, "A systematic evaluation of the bag-of-frames representation for music information retrieval," TMM 2014

Outline

- Basics
- VQ-VAE
- **Audio codec models**
- Transformer
- Transformer-based text-to-music generation

Audio Codec

- **Use raw waveforms for VQ-VAE** (like OpenAI's JukeBox)
- Initially developed for **audio compression**
  - ***Bitrate*** vs ***fidelity***
  - Computational complexity vs coding efficiency

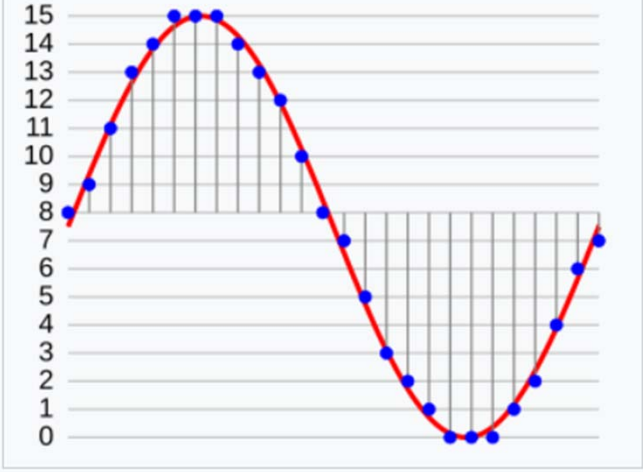


- Use vector quantization (VQ) for quantization
  - Produce **discrete tokens** that can be used for building a Transformer LM, though training LMs is not the focus of these audio compression papers

Ref: Zeghidour et al, "SoundStream: An end-to-end neural audio codec," IEEE TASLP 2021

Bitrate

- Pulse-code modulation (PCM)
  - 4bits: 16 possible values
  - 16bits: 65,536 possible values

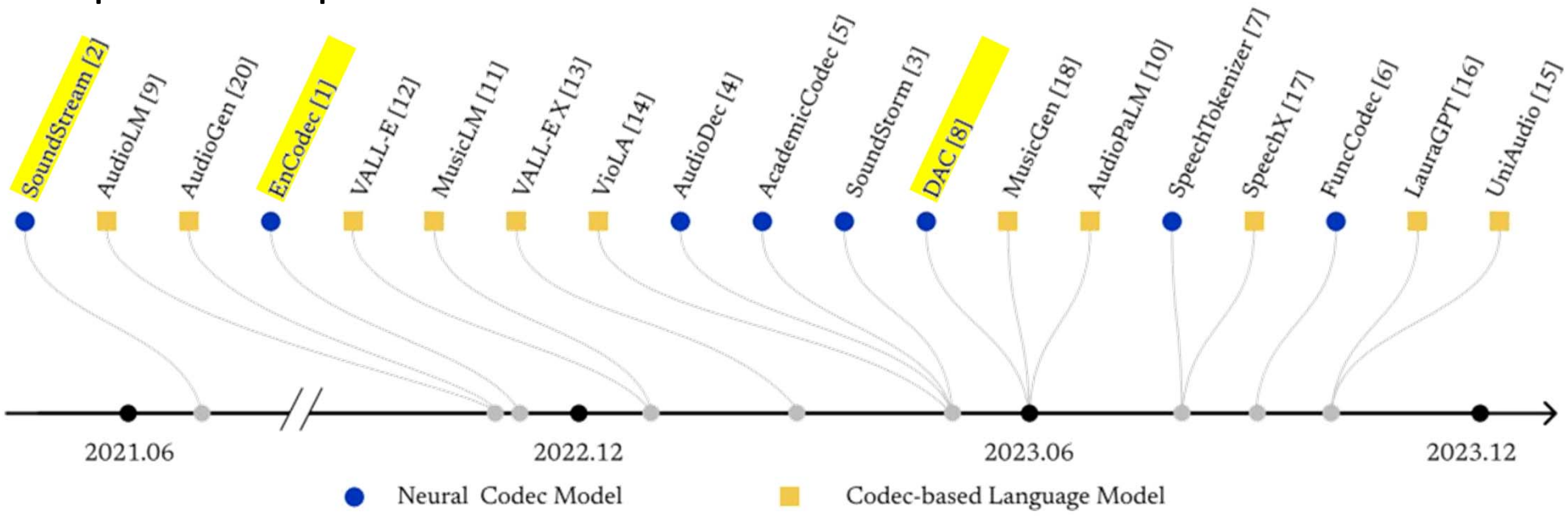


Sampling and quantization of a signal (red) for 4-bit LPCM over a time domain at specific frequency

- For 16bits 44.1kHz audio
  - 705.6 kbits per second (**kbps**)
  - 42.336 Mbits per minute

Audio Codec

- Rapid development



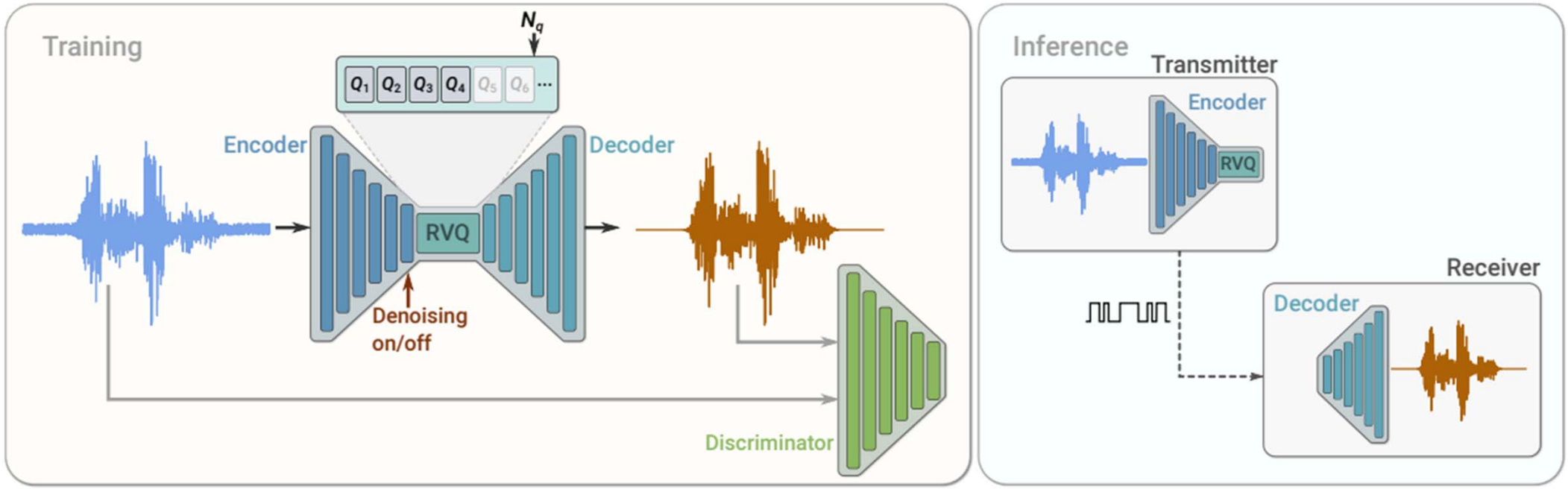
Towards audio language modeling - an overview

Haibin Wu[1], Xuanjun Chen[1*], Yi-Cheng Lin[1*], Kai-wei Chang[1], Ho-Lam Chung[1],
Alexander H. Liu[2], Hung-yi Lee[1]

https://arxiv.org/pdf/2402.13236

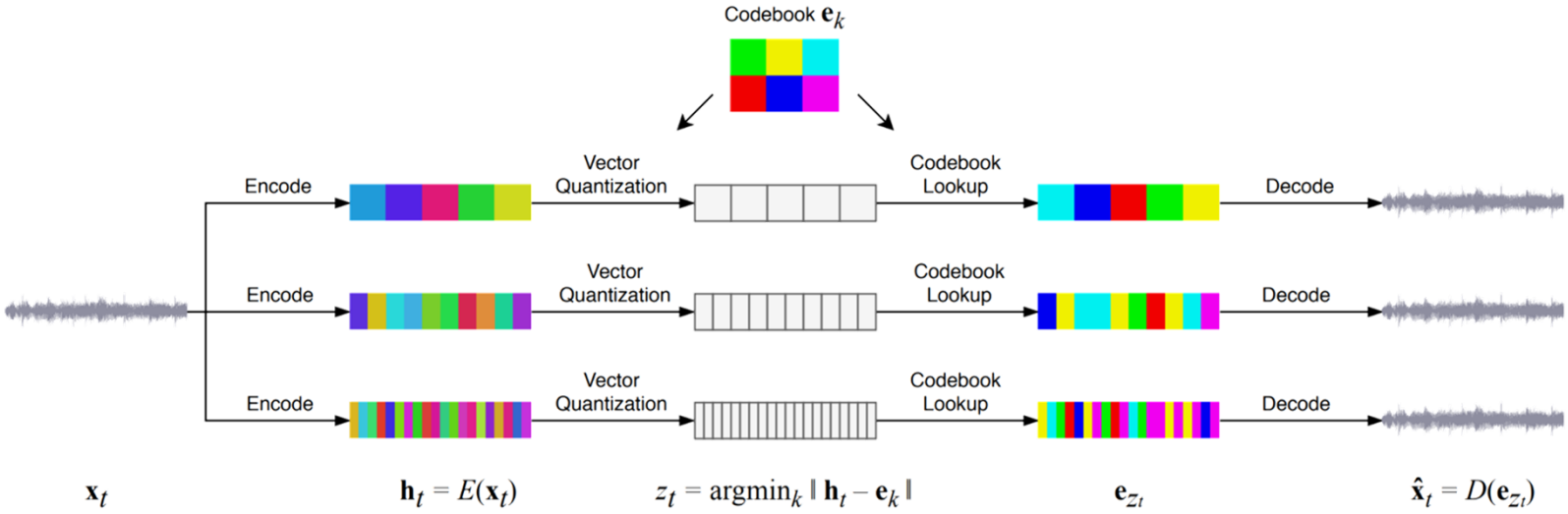
SoundStream (from Google)

- Use RVQ
  - The same audio chunk is **recursively** VQed by different codebooks

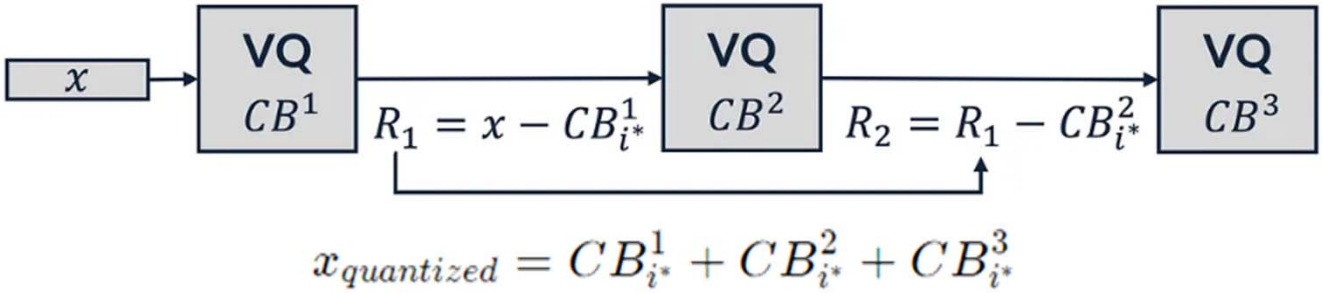


Ref: Zeghidour et al, "SoundStream: An end-to-end neural audio codec," IEEE TASLP 2021

Hierarchical VQ vs RVQ



- **RVQ** is found to be superior in audio codec research



RVQ Demonstration

- Use descript audio codec (DAC) → see later slides

- Adding more quantizers (codebook layers) improve fidelity
  - At the cost of increased bitrate (therefore less compression rate)

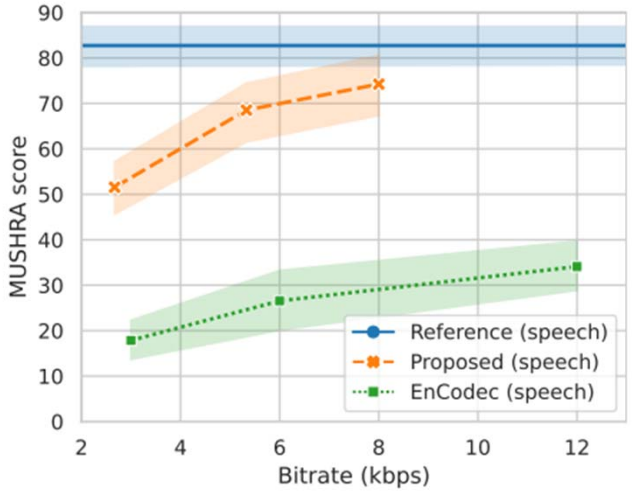
9 layers

6 layers

3 layers

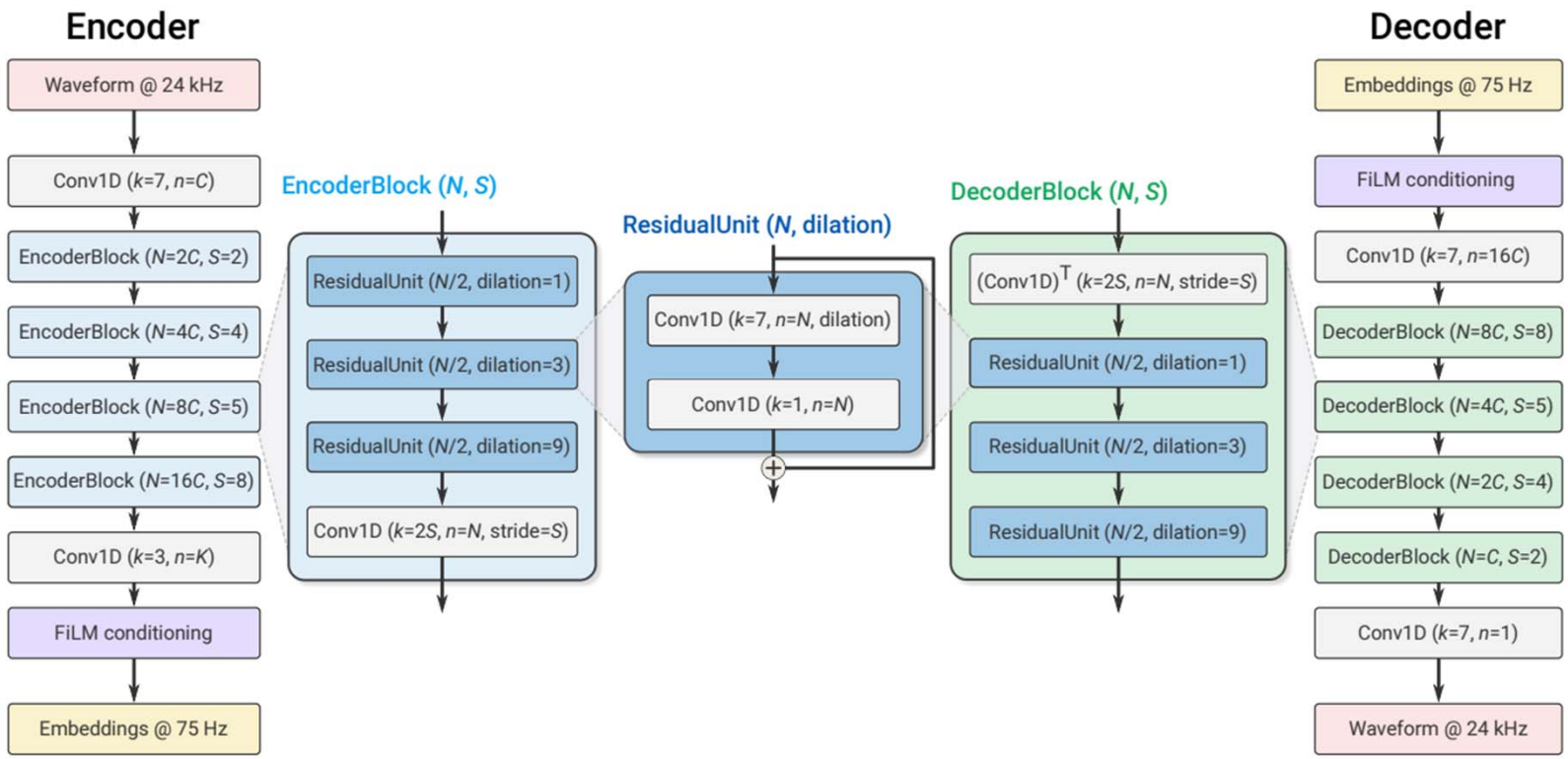
2 layers

1 layer



SoundStream (from Google)

- **Convolutional** encoder/decoder



Ref: Zeghidour et al, "SoundStream: An end-to-end neural audio codec," IEEE TASLP 2021

SoundStream (from Google)

- Use GAN
  - use both a waveform-domain **discriminator** and an STFT-based **discriminator**, which receives as input the complex-valued STFT of the input waveform, expressed in terms of real and imaginary parts
- Both discriminators are fully **convolutional**

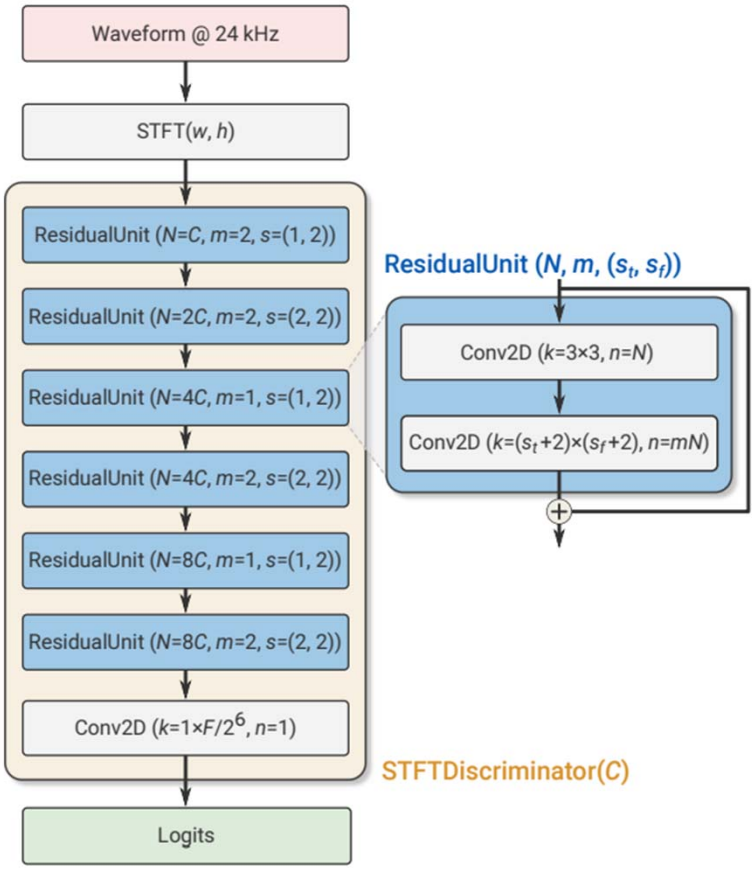


Fig. 4: STFT-based discriminator architecture.

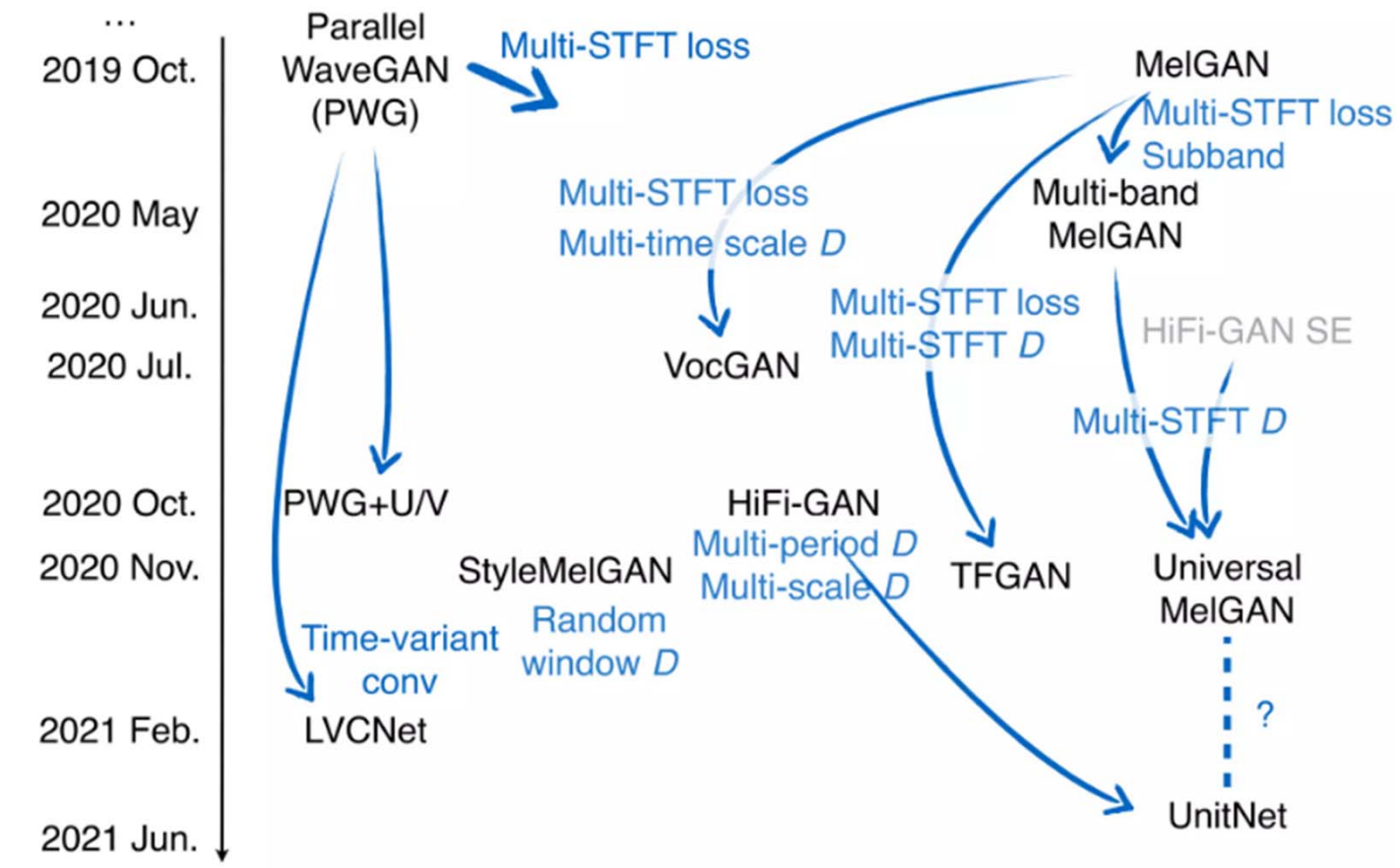
Ref: Zeghidour et al, "SoundStream: An end-to-end neural audio codec," IEEE TASLP 2021

Why GAN instead of Waveform-domain Loss?

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

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

GAN-based Models Developed in Vocoder Research



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

SoundStream (from Google)

- Use **8** layers of RVQ
- Codebook size set to **1024** (2^{10}) for each quantizer
- Evaluation metric: Google's **ViSQOL** (Virtual Speech Quality Objective Listener)
- **Not open source**

TABLE II: Trade-off between residual vector quantizer depth and codebook size at 6 kbps.

Number of quantizers N_q	8	16	80
Codebook size N	1024	32	2
ViSQOL	4.01 ± 0.03	3.98 ± 0.03	3.92 ± 0.03

(the effective number of unique codewords of these three configurations is all $\mathbf{2^{80}}$, why?)

Ref: Zeghidour et al, "SoundStream: An end-to-end neural audio codec," IEEE TASLP 2021

EnCodec (from Meta)

Open source https://github.com/facebookresearch/encodec

- A causal model operating at 24 kHz on monophonic audio trained on a variety of audio data.
- A non-causal model operating at 48 kHz on stereophonic audio trained on music-only data.

The 24 kHz model can compress to 1.5, 3, 6, 12 or 24 kbps, while the 48 kHz model support 3, 6, 12 and 24 kbps. We also provide a pre-trained language model for each of the models, that can further compress the representation by up to 40% without any further loss of quality.

Compression

```
encodec [-b TARGET_BANDWIDTH] [-f] [--hq] [--lm] INPUT_FILE [OUTPUT_FILE]
```

Decompression

```
encodec [-f] [-r] ENCODEC_FILE [OUTPUT_WAV_FILE]
```

Ref: Défossez et al, “High fidelity neural audio compression,” TMLR 2023

EnCodec (from Meta)

Extracting discrete representations

```python
from encodec import EncodecModel
from encodec.utils import convert_audio
```

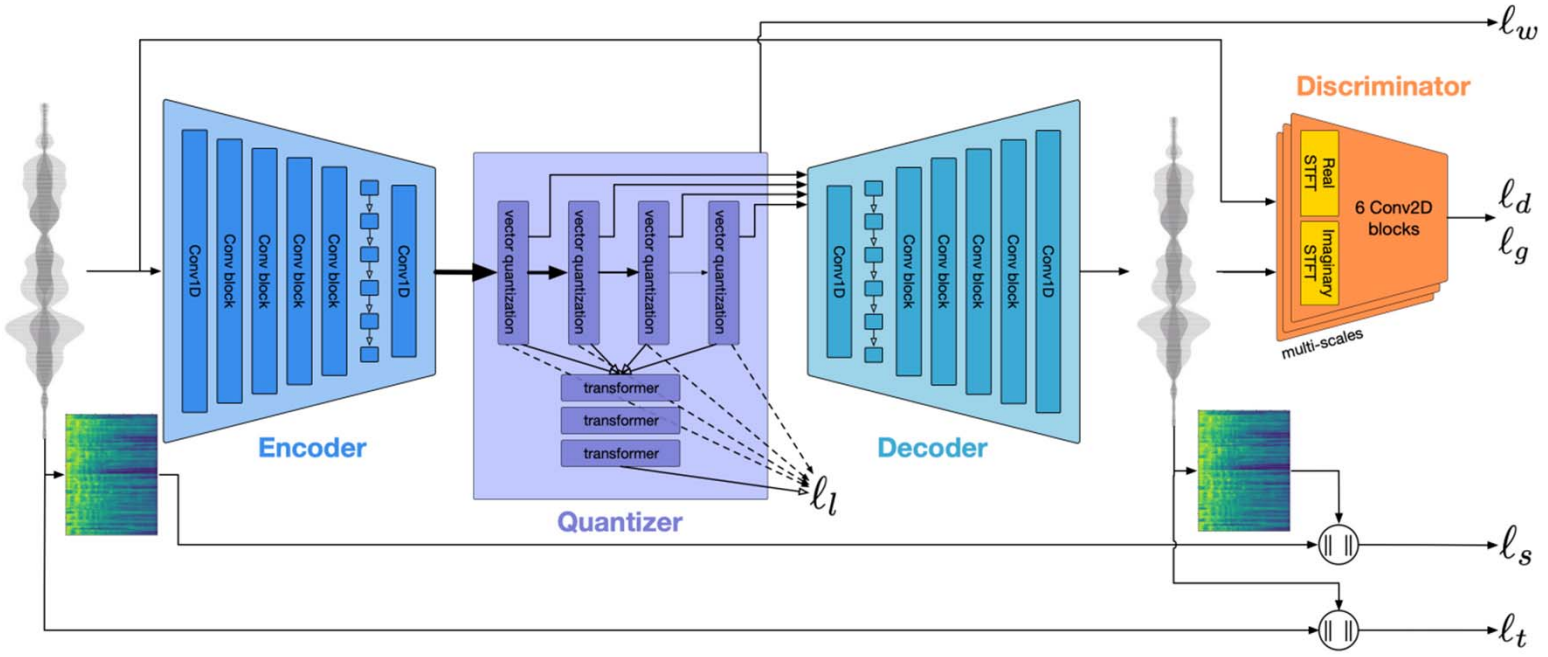
```python
# Instantiate a pretrained EnCodec model
model = EncodecModel.encodec_model_24khz()
# The number of codebooks used will be determined bythe bandwidth selected.
# E.g. for a bandwidth of 6kbps, `n_q = 8` codebooks are used.
# Supported bandwidths are 1.5kbps (n_q = 2), 3 kbps (n_q = 4), 6 kbps (n_q = 8) and 12 kbps (n_q =16) and 24
# For the 48 kHz model, only 3, 6, 12, and 24 kbps are supported. The number
# of codebooks for each is half that of the 24 kHz model as the frame rate is twice as much.
model.set_target_bandwidth(6.0)

# Load and pre-process the audio waveform
wav, sr = torchaudio.load("<PATH_TO_AUDIO_FILE>")
wav = convert_audio(wav, sr, model.sample_rate, model.channels)
wav = wav.unsqueeze(0)

# Extract discrete codes from EnCodec
with torch.no_grad():
    encoded_frames = model.encode(wav)
codes = torch.cat([encoded[0] for encoded in encoded_frames], dim=-1)  # [B, n_q, T]
```

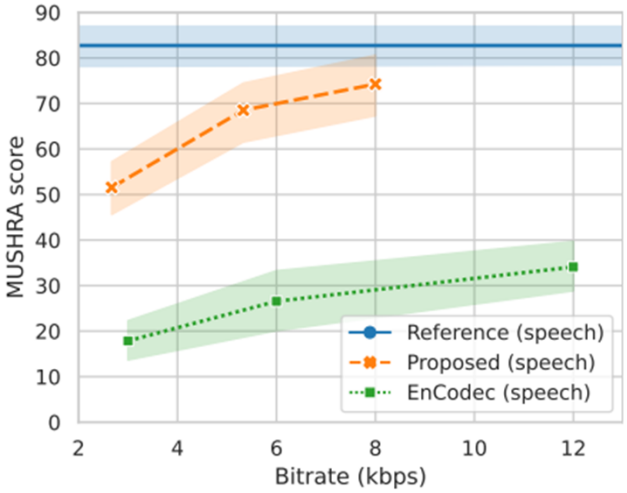
Encodec

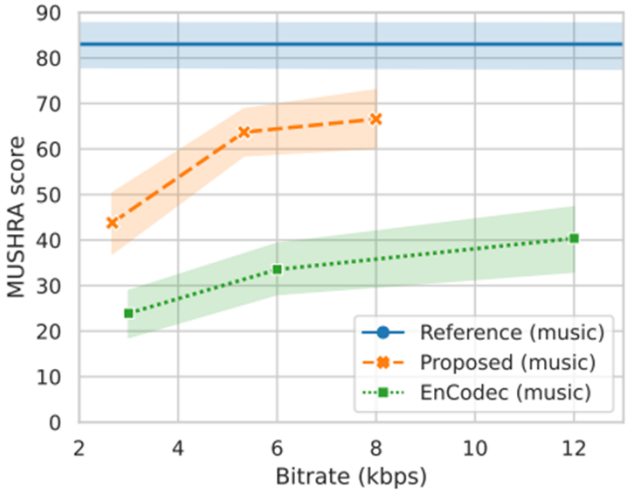
- RVQVAE+GAN, convolutional encoder/decoder
- Trained using **8 A100 GPUs** for 300 epochs
- Use the Jamendo dataset for the music version of Encodec



Descript Audio Codec (DAC)

- Can compress 44.1 KHz audio into discrete codes at **8 kbps** bitrate (~90x compression)
  - Why 90x? (16bits 44.1kHz audio: 705.6 kbps)
- Outperform EnCodec in both objective and subjective evaluations





Codec	Bitrate (kbps)	Bandwidth (kHz	Mel distance ↓	STFT distance ↓	ViSQOL ↑	SI-SDR ↑
Proposed	1.78	22.05	1.39	1.95	3.76	2.16
	2.67	22.05	1.28	1.85	3.90	4.41
	5.33	22.05	1.07	1.69	4.09	8.13
	8	22.05	0.93	1.60	4.18	10.75
EnCodec	1.5	12	2.11	4.30	2.82	-0.02
	3	12	1.97	4.19	2.94	2.94
	6	12	1.83	4.10	3.05	5.99
	12	12	1.70	4.02	3.13	8.36
	24	12	1.61	3.97	3.16	9.59
Lyra	9.2	8	2.71	4.86	2.19	-14.52
Opus	8	4	3.60	5.72	2.06	5.68
	14	16	1.23	2.14	4.02	8.02
	24	16	0.88	1.90	4.15	11.65

Ref: Kumar et al, “High-fidelity audio compression with improved RVQGAN,” NeurIPS 2023

Descript Audio Codec (DAC)

Improved RVQGAN

Lower codebook dimension

Ablation on	Decoder dim.	Activation	Multi-period	Single-scale	# of STFT bands	Multi-scale mel.	Latent dim	Quant. method	Quant. dropout	Bitrate (kbps)	Balanced samp.	Mel distance ↓	STFT distance ↓	ViSQOL ↑	SI-SDR ↑	Bitrate efficiency ↑
	1536	snake	✓	✗	5	✓	8	Proj.	1.0	8	✓	1.09	1.82	3.96	9.12	99%
Architecture	512	snake	✓	✗	5	✓	8	Proj.	1.0	8	✓	1.11	1.83	3.91	8.72	99%
	1024	snake	✓	✗	5	✓	8	Proj.	1.0	8	✓	1.07	1.82	3.96	9.07	99%
	1536	relu	✓	✗	5	✓	8	Proj.	1.0	8	✓	1.17	1.81	3.83	6.92	99%
Discriminator	1536	snake	✗	✗	✗	✓	8	Proj.	1.0	8	✓	1.13	1.92	4.12	1.07	62%
	1536	snake	✓	✗	1	✓	8	Proj.	1.0	8	✓	1.07	1.80	3.98	9.07	99%
	1536	snake	✗	✗	5	✓	8	Proj.	1.0	8	✓	1.07	1.81	3.97	9.04	99%
	1536	snake	✗	✓	5	✓	8	Proj.	1.0	8	✓	1.08	1.82	3.95	8.51	99%
Reconstruction loss	1536	snake	✓	✗	5	✗	8	Proj.	1.0	8	✓	1.10	1.87	4.01	7.68	99%
Latent dim	1536	snake	✓	✗	5	✓	2	Proj.	1.0	8	✓	1.44	2.08	3.65	2.22	84%
	1536	snake	✓	✗	5	✓	4	Proj.	1.0	8	✓	1.20	1.89	3.86	7.15	97%
	1536	snake	✓	✗	5	✓	32	Proj.	1.0	8	✓	1.10	1.84	3.95	9.05	98%
	1536	snake	✓	✗	5	✓	256	Proj.	1.0	8	✓	1.31	1.97	3.79	5.09	59%
Quantization setup	1536	snake	✓	✗	5	✓	8	EMA	1.0	8	✓	1.11	1.84	3.94	8.33	97%
	1536	snake	✓	✗	5	✓	8	Proj.	0.0	8	✓	0.98	1.70	4.09	10.14	99%
	1536	snake	✓	✗	5	✓	8	Proj.	0.25	8	✓	0.99	1.69	4.04	10.00	99%
	1536	snake	✓	✗	5	✓	8	Proj.	0.5	8	✓	1.01	1.75	4.03	9.74	99%
	1536	snake	✓	✗	5	✓	8	Proj.	1.0	24	✓	0.73	1.62	4.16	13.83	99%
Data	1536	snake	✓	✗	5	✓	8	Proj.	1.0	8	✗	1.09	1.94	3.89	8.89	99%

Descript Audio Codec (DAC)

https://github.com/descriptinc/descript-audio-codec

```
python3 -m dac download --model_type 44khz # downloads the 44kHz variant
python3 -m dac download --model_type 24khz # downloads the 24kHz variant
python3 -m dac download --model_type 16khz # downloads the 16kHz variant
```

```
# Load audio signal file
signal = AudioSignal('input.wav')

# Encode audio signal as one long file
# (may run out of GPU memory on long files)
signal.to(model.device)

x = model.preprocess(signal.audio_data, signal.sample_rate)
z, codes, latents, _, _ = model.encode(x)
```

DAC vs Encodec

Q: why 8 kbps?
A: 9*10*86

Codec	Sampling rate (kHz)	Target bitrate (kbps)	Striding factor	Frame rate (Hz)	# of 10-bit codebooks	Compression factor
Proposed	44.1	8	512	86	9	91.16
EnCodec	24	24	320	75	32	16
	48	24	320	150	16	32
SoundStream	24	6	320	75	8	64

Sampling rate (kHz)

Target bitrate (kbps)

Striding factor

Frame rate (Hz)

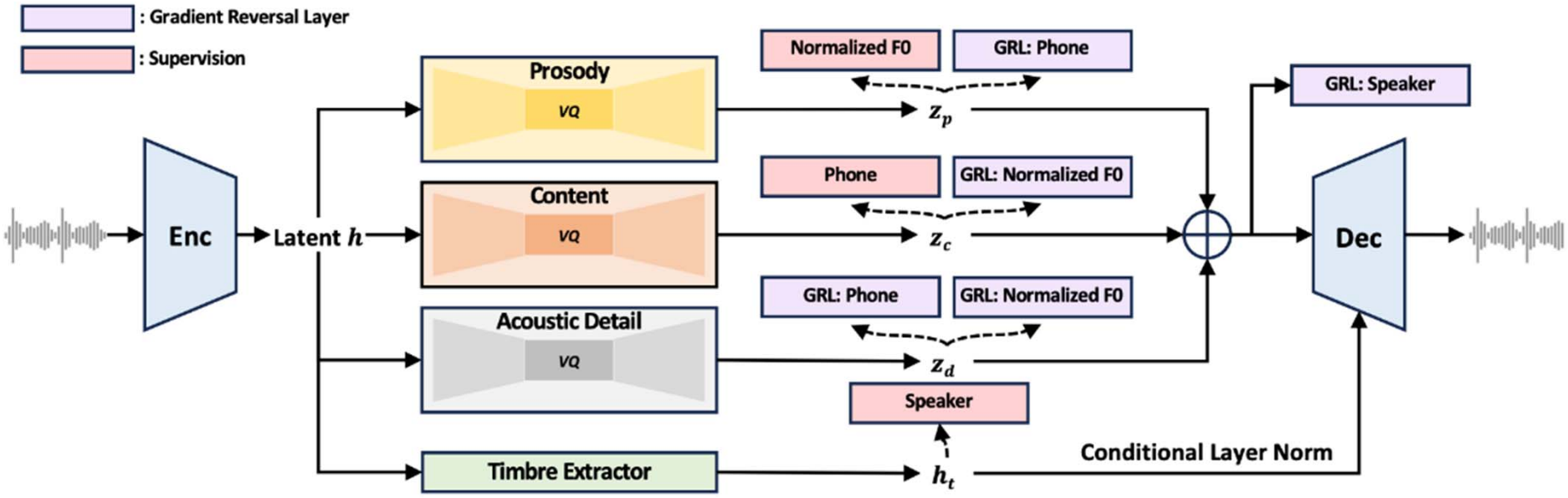
# of 10-bit codebooks

Compression factor

(Frame rate is related to the design of the encoder/decoder in VQVAE)

Many More Codec Models…

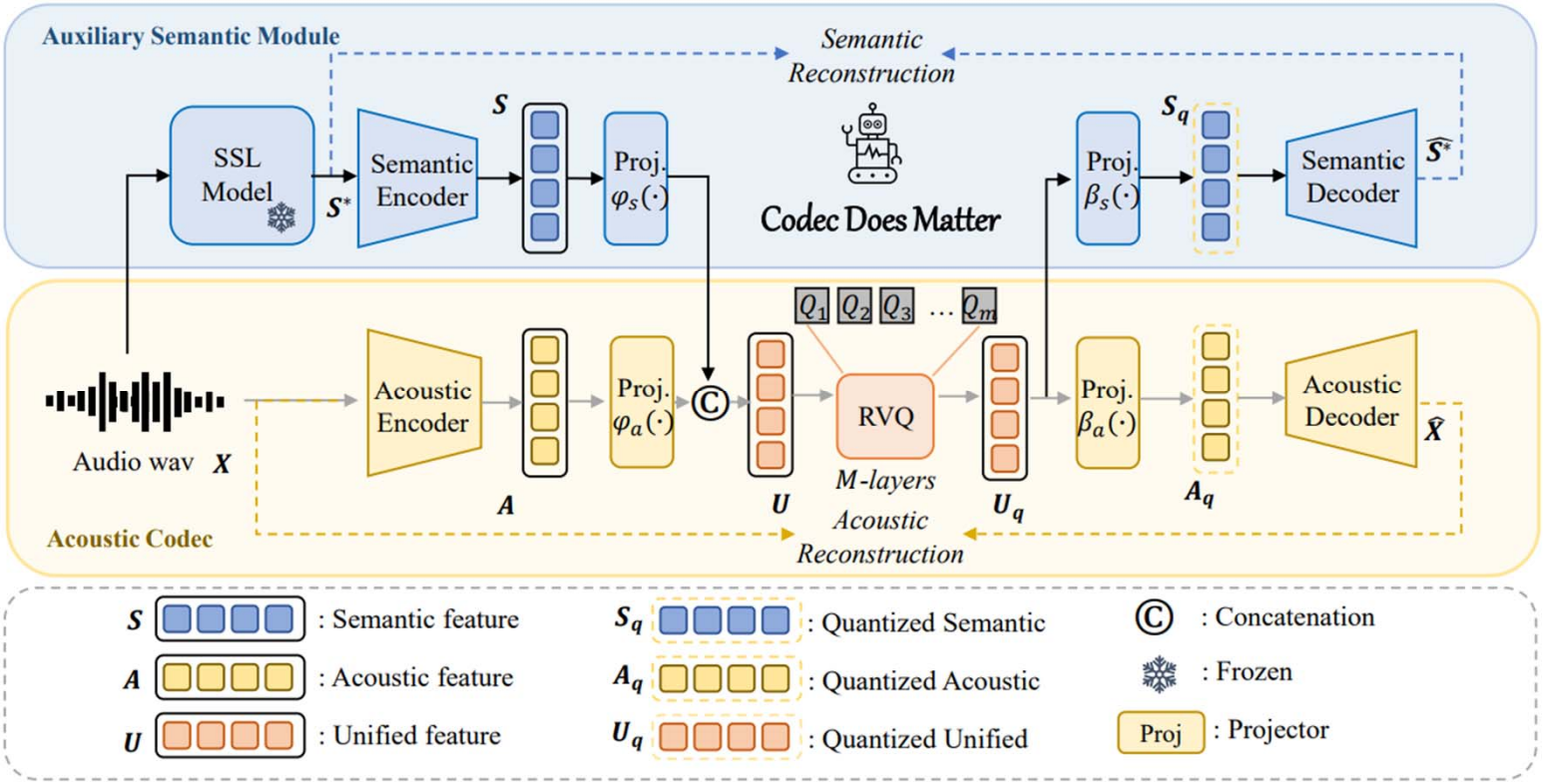
- **FACodec:** factorized neural speech codec (ICML 2024)



Ref: Ju et al, "NaturalSpeech 3: Zero-shot speech synthesis with factorized codec and diffusion models," ICML 2024

Many More Codec Models…

- **X-Codec:** improved capability of semantic modeling (arxiv:2408.17175)

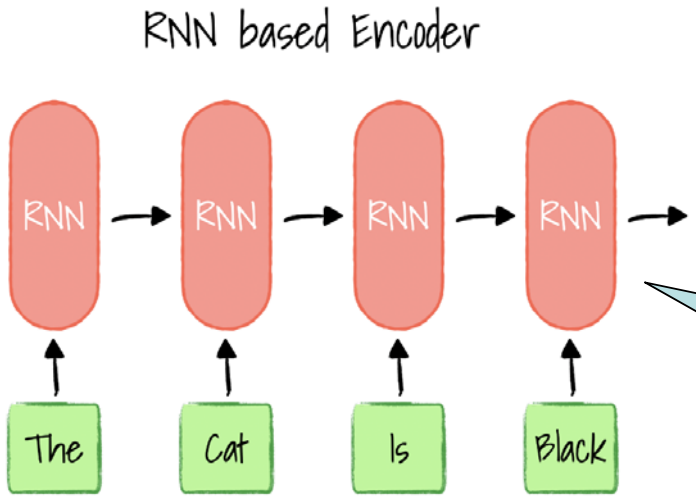


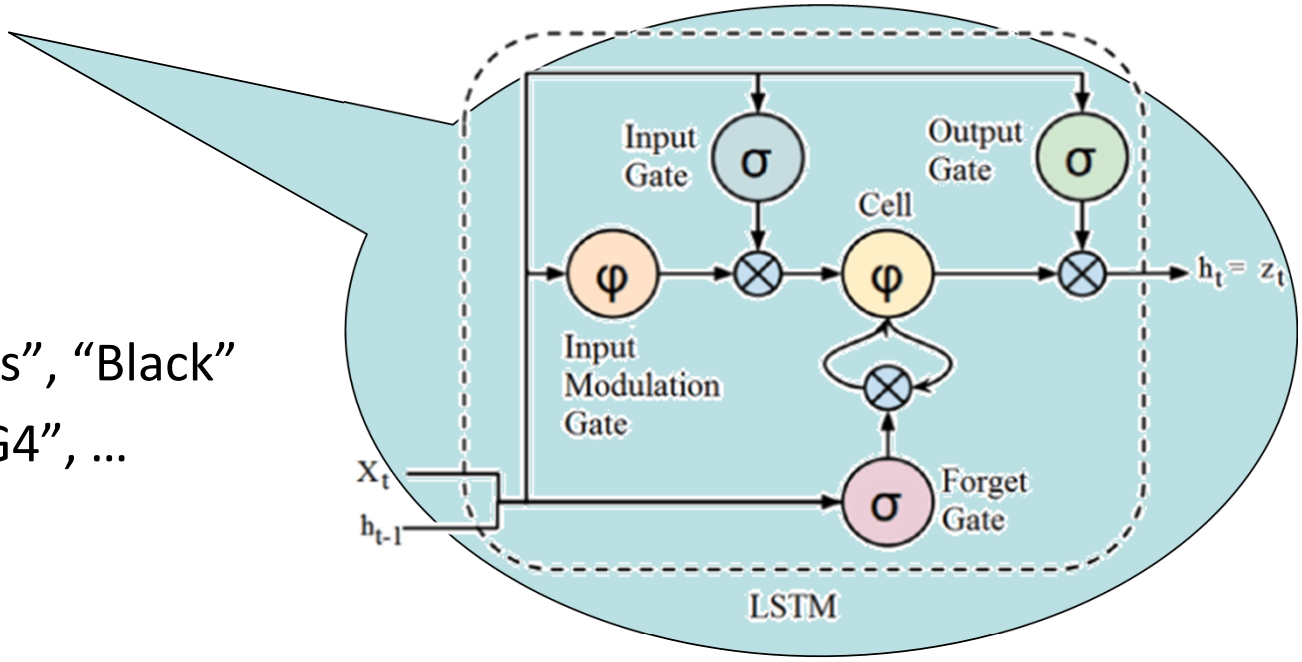
Ref: Ye et al, "Codec does matter: Exploring the semantic shortcoming of codec for audio language model," arXiv 2024

Outline

- Basics
- VQ-VAE
- Audio codec models
- **Transformer**
- Transformer-based text-to-music generation

RNNs

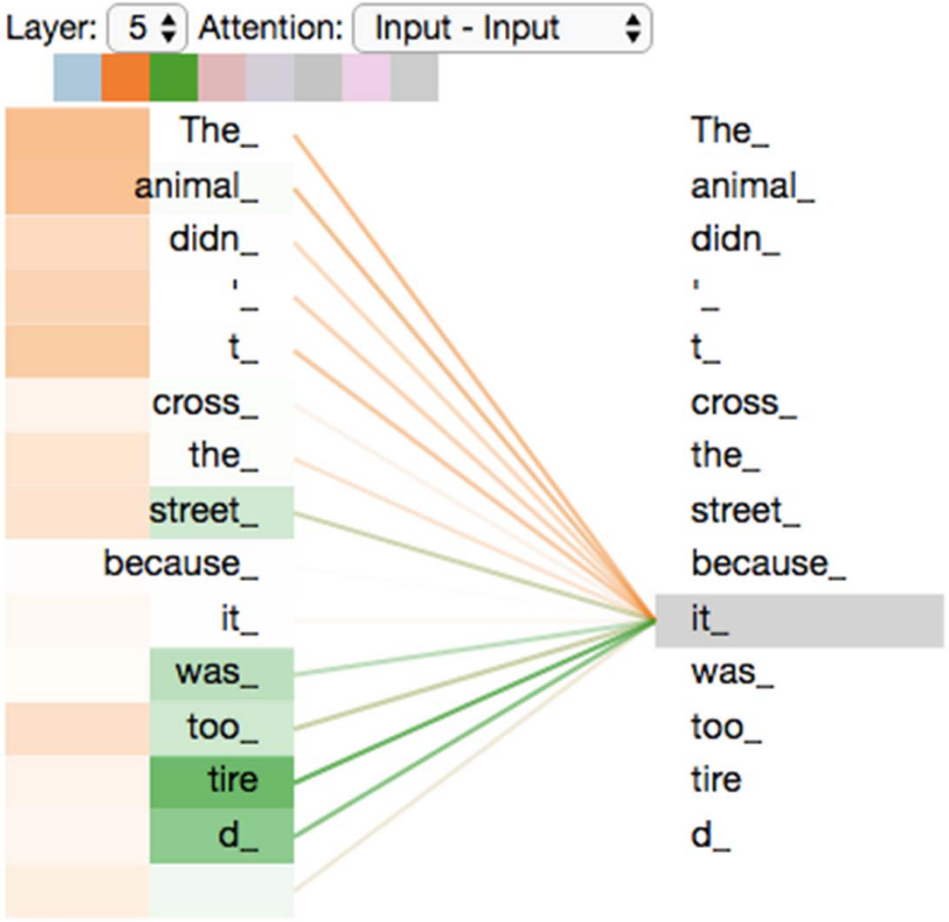




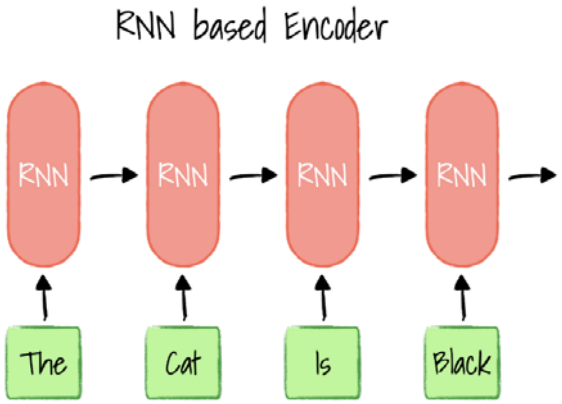
- Text tokens: “The”, “Cat”, “Is”, “Black”
- Music tokens: “C4”, “E4”, “G4”, …

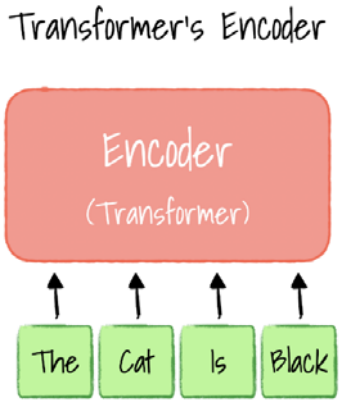
Transformers

- 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
  - $y_t = f(y_{t-1}, h_{t-1})$

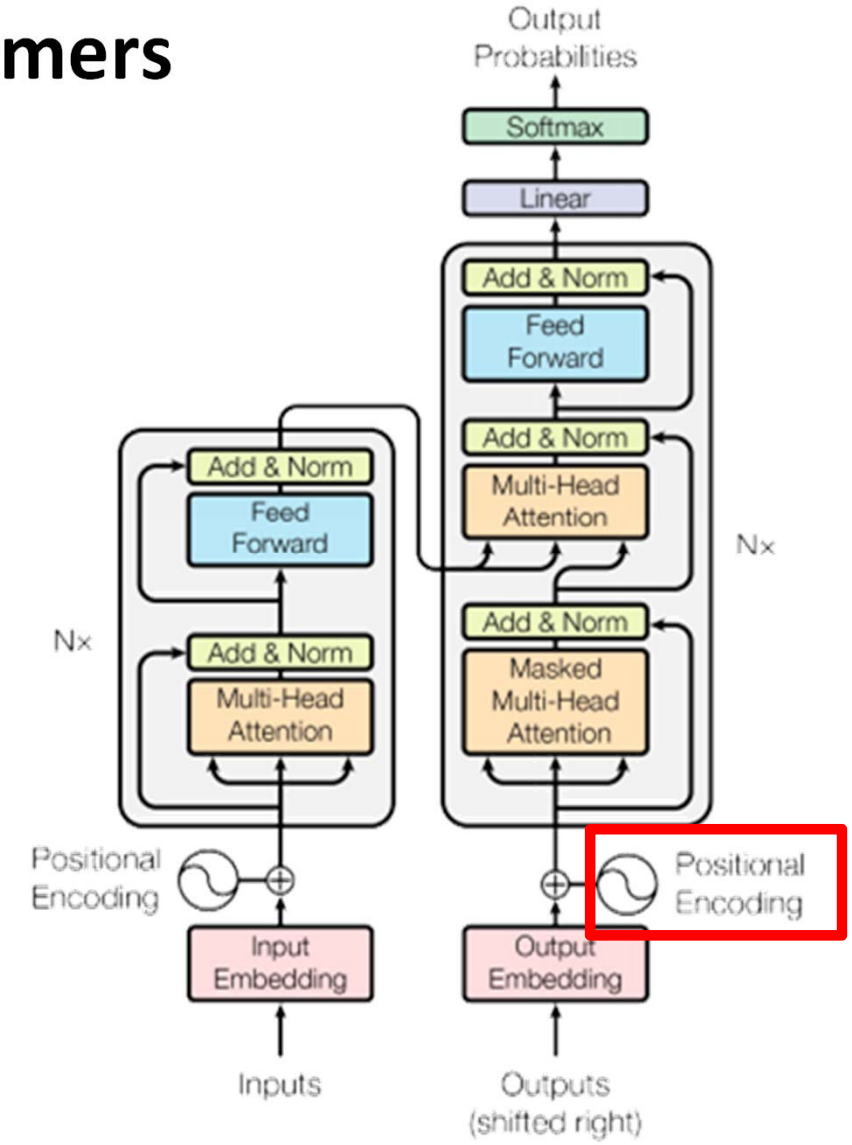
current output — last output — latent representation of the "history"

- **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 (from Prof. Jia-Bin Huang)

https://www.youtube.com/watch?v=rcWMRA9E5RI

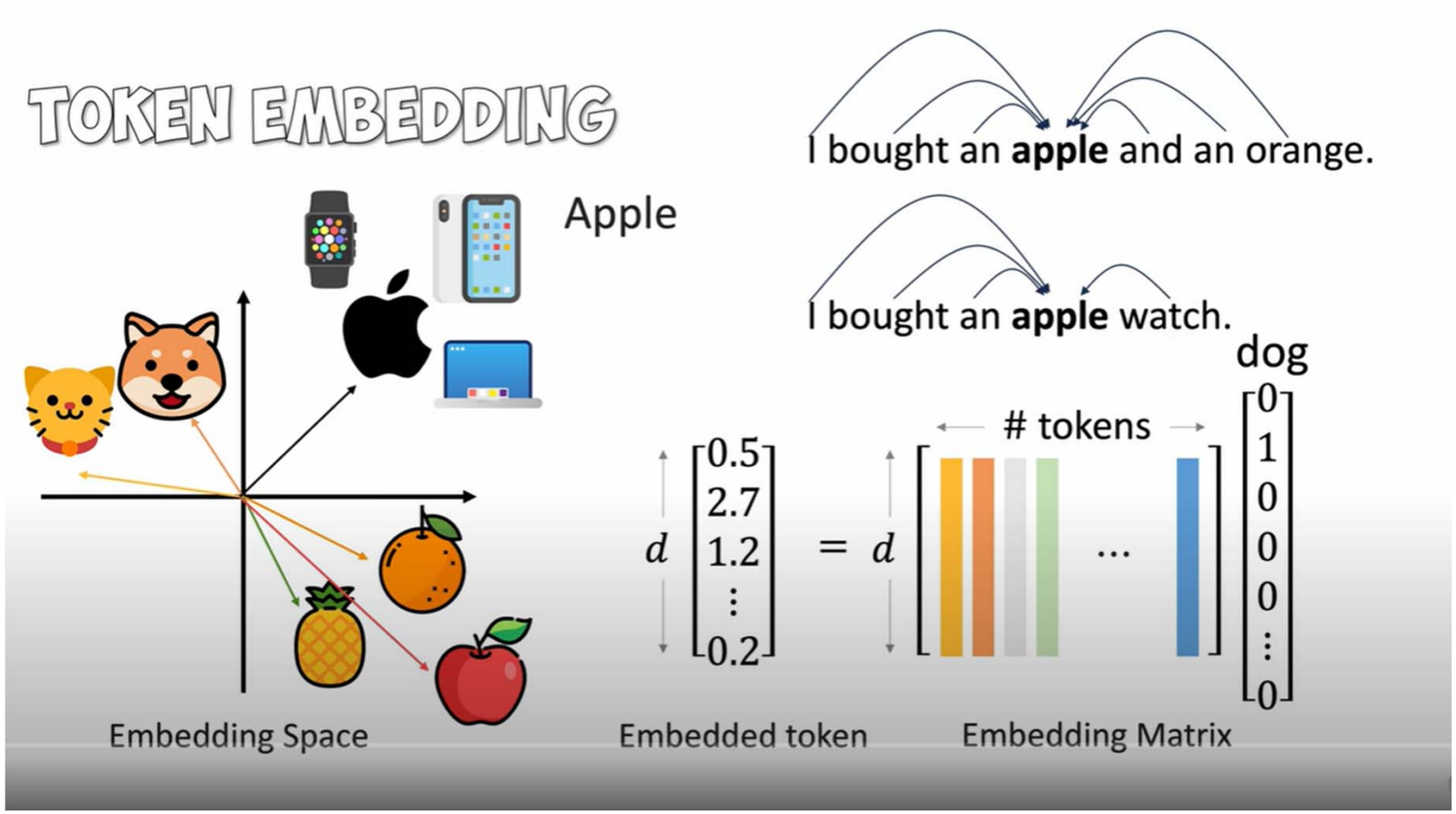
But What Are Transformers?

Jia-Bin Huang • 6.9K views

Transformers is arguably the most influential neural network architecture in the last decade, powering the current boom of...

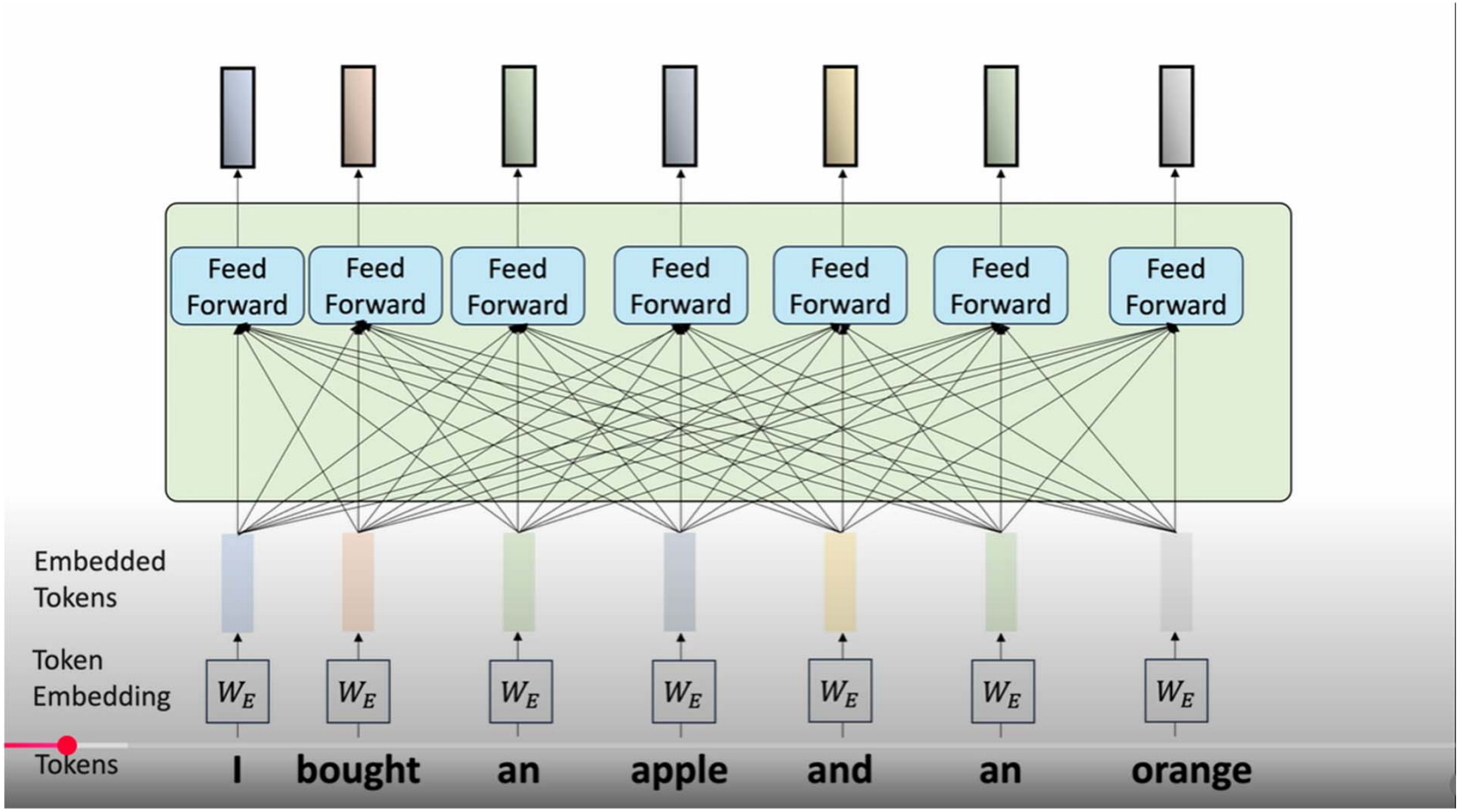
Transformer (from Prof. Jia-Bin Huang)

https://www.youtube.com/watch?v=rcWMRA9E5RI



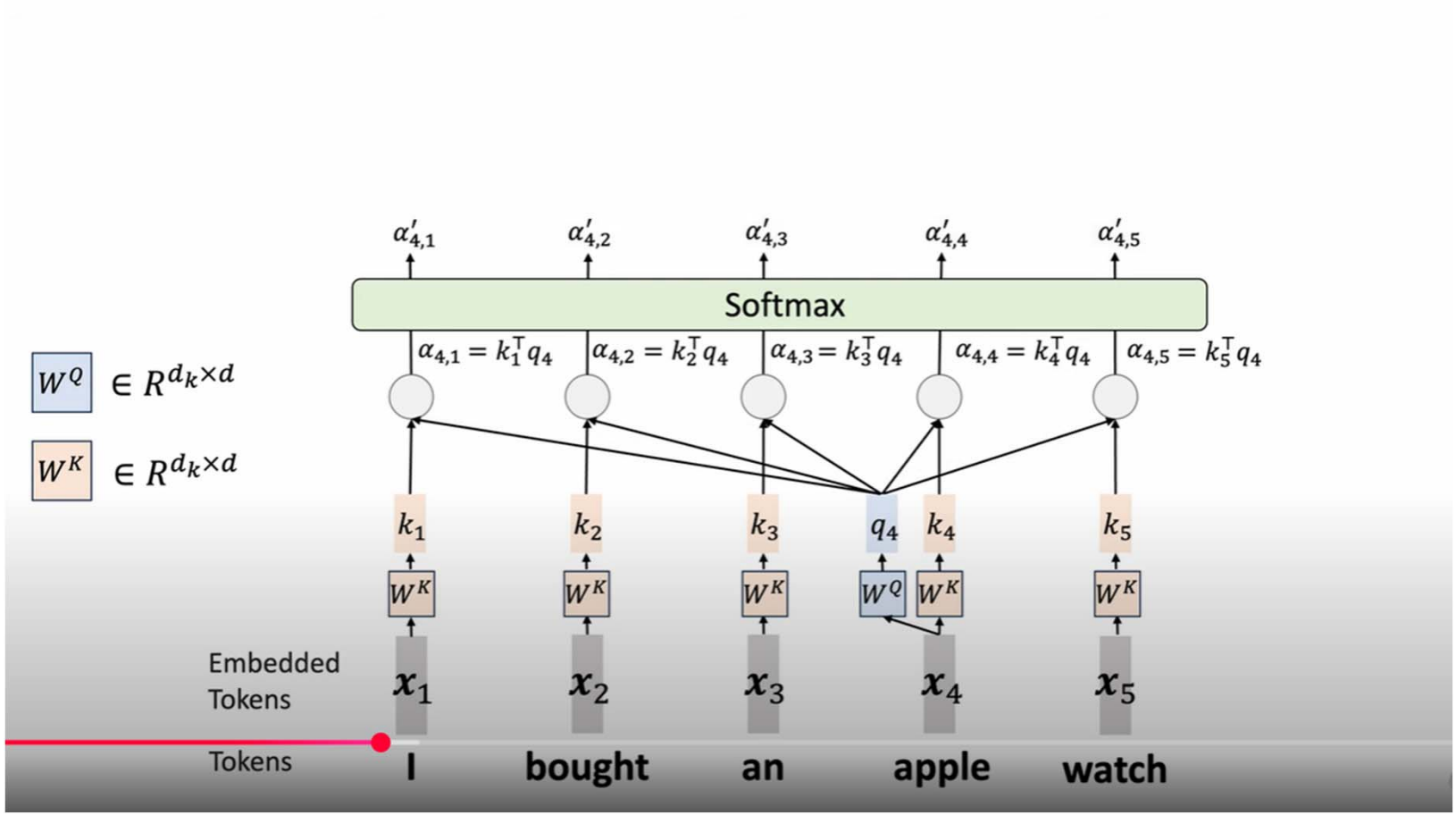
Transformer (from Prof. Jia-Bin Huang)

https://www.youtube.com/watch?v=rcWMRA9E5RI



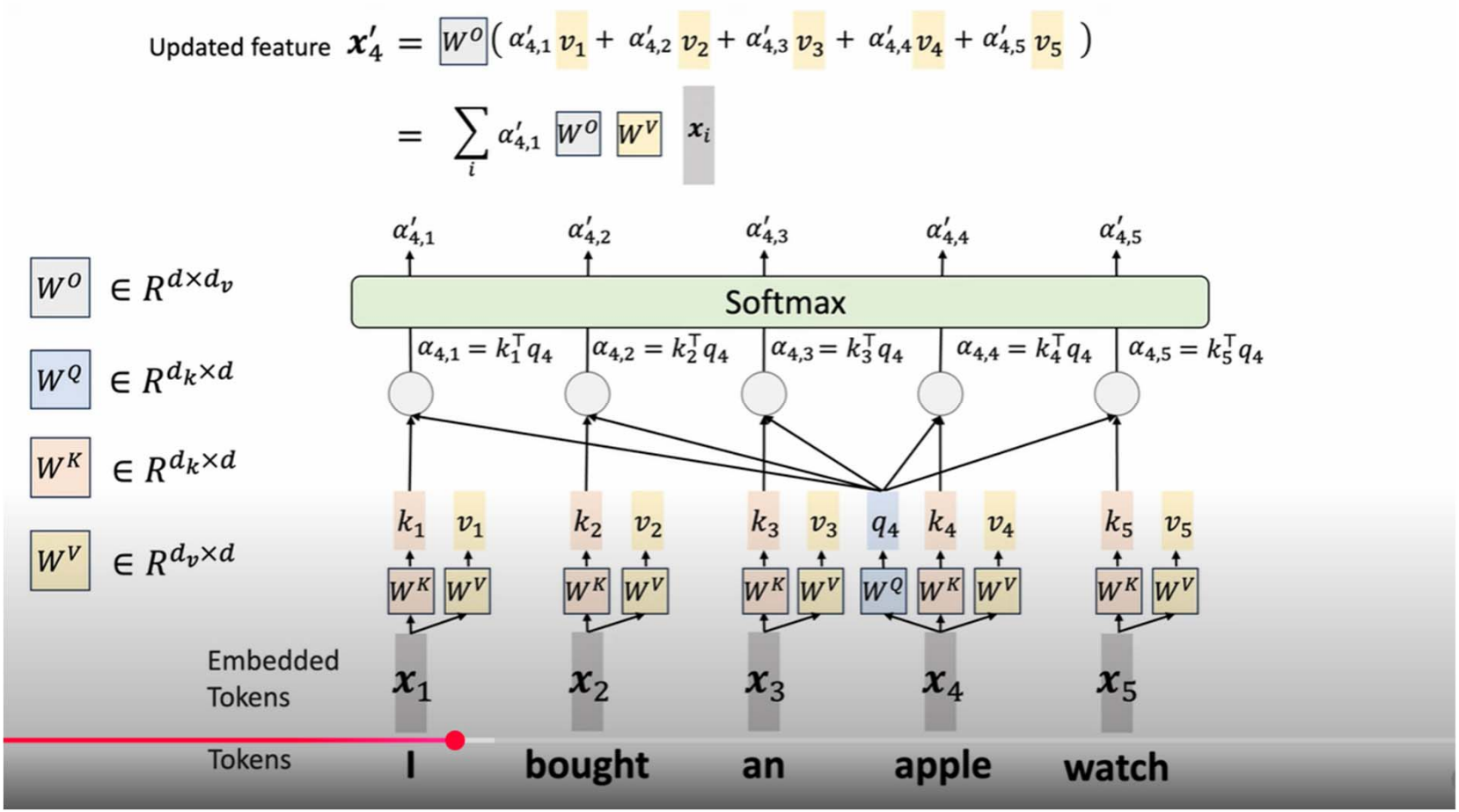
Transformer (from Prof. Jia-Bin Huang)

https://www.youtube.com/watch?v=rcWMRA9E5RI



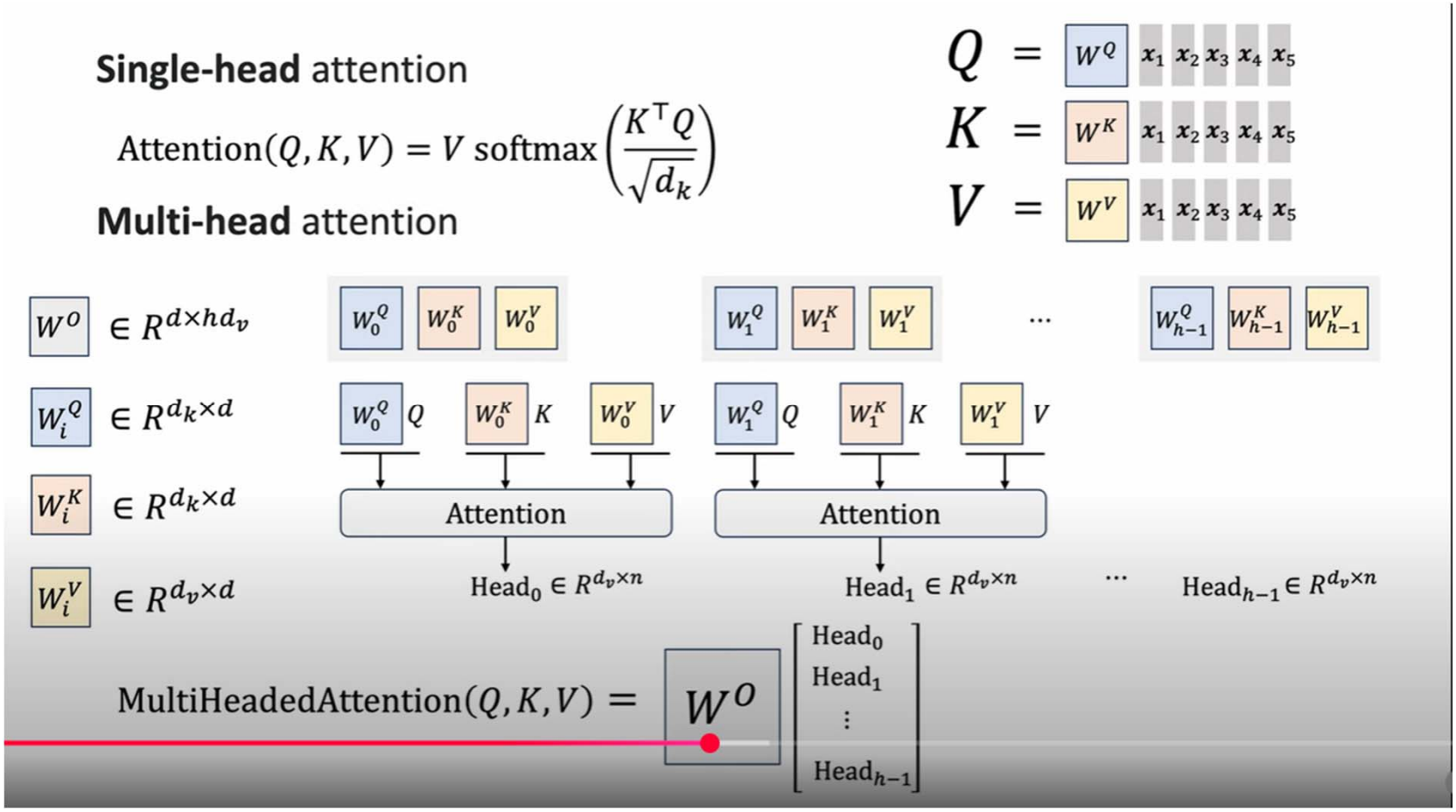
Transformer (from Prof. Jia-Bin Huang)

https://www.youtube.com/watch?v=rcWMRA9E5RI



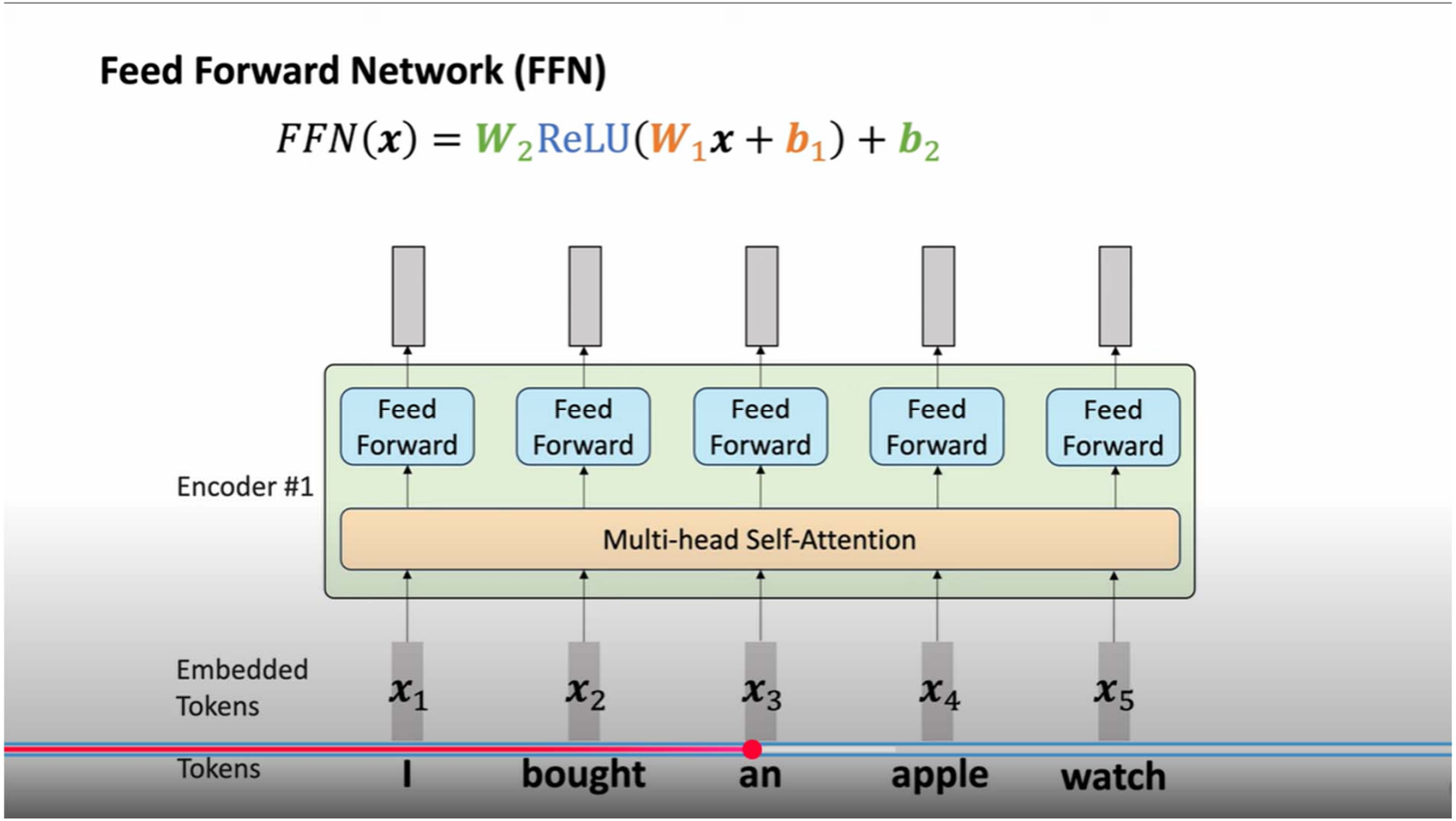
Transformer (from Prof. Jia-Bin Huang)

https://www.youtube.com/watch?v=rcWMRA9E5RI



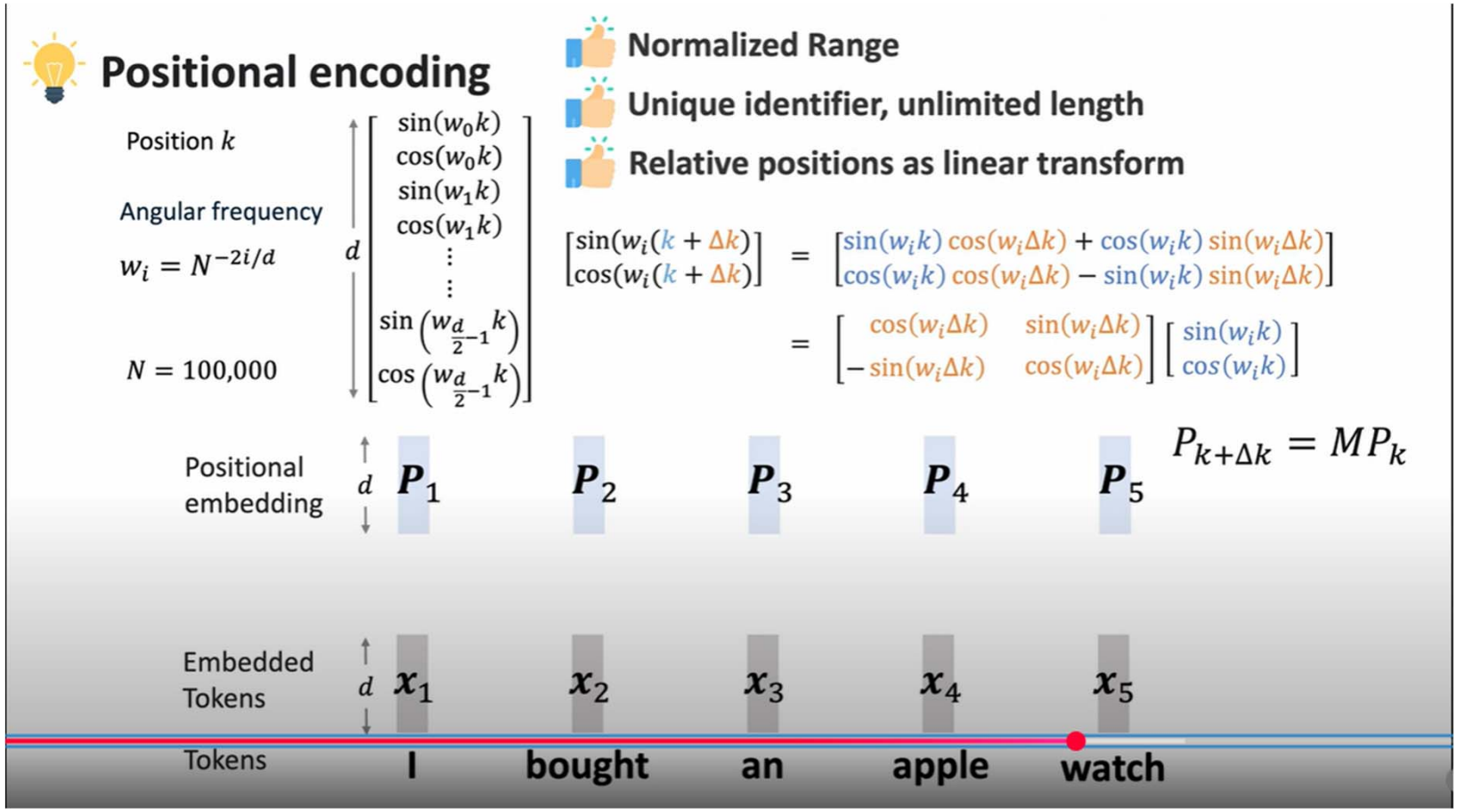
Transformer (from Prof. Jia-Bin Huang)

https://www.youtube.com/watch?v=rcWMRA9E5RI



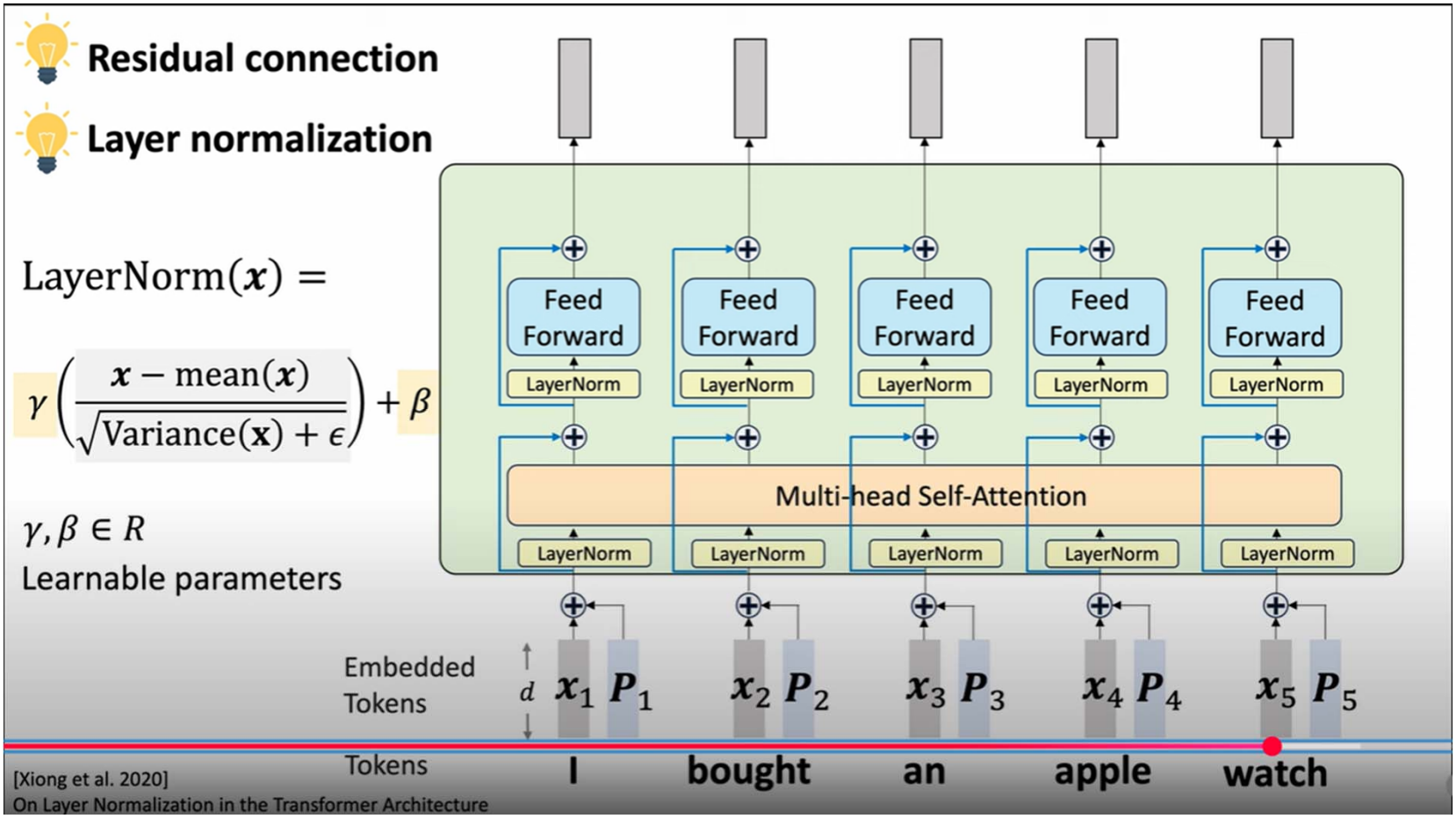
Transformer (from Prof. Jia-Bin Huang)

https://www.youtube.com/watch?v=rcWMRA9E5RI



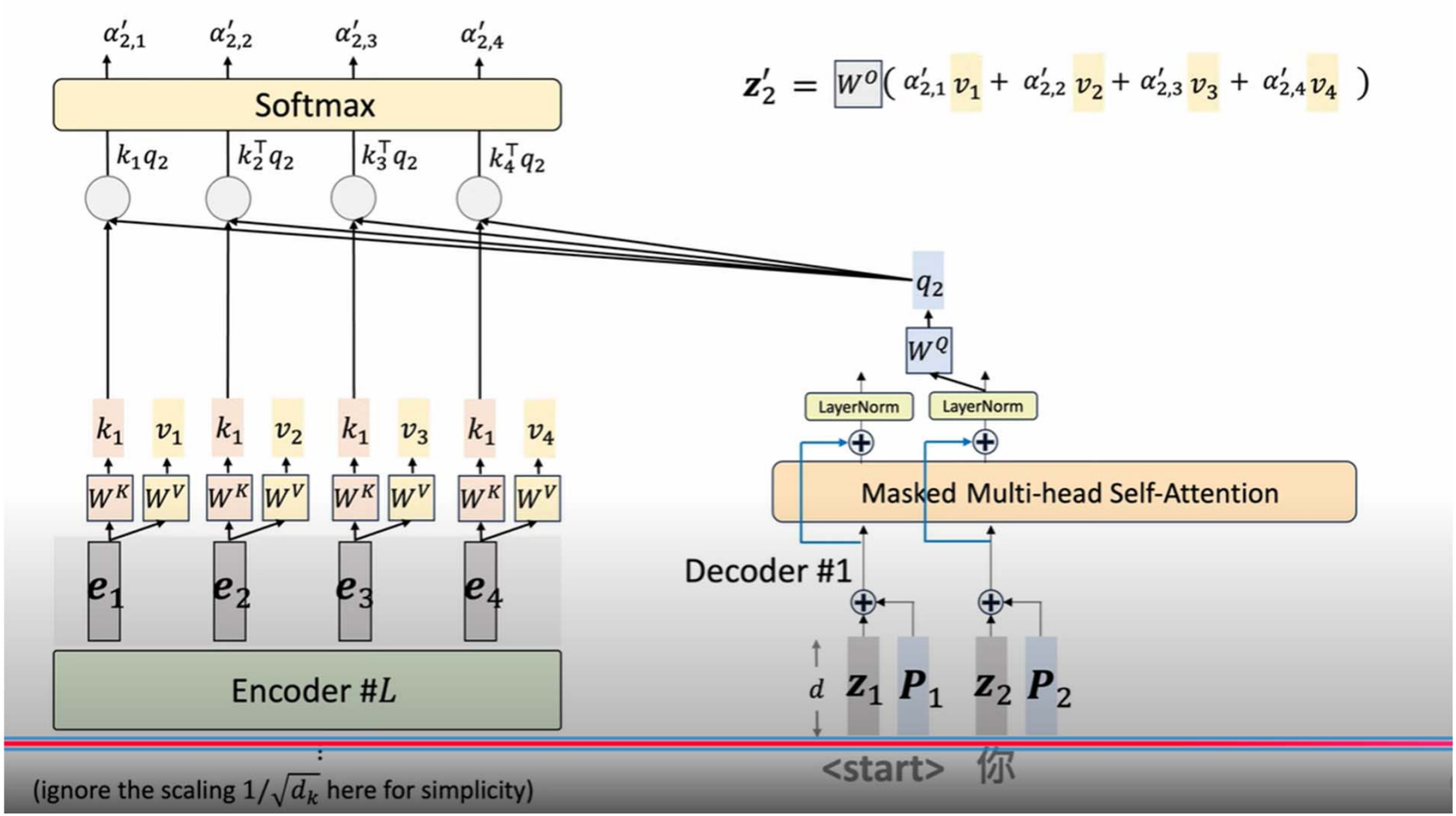
Transformer (from Prof. Jia-Bin Huang)

https://www.youtube.com/watch?v=rcWMRA9E5RI



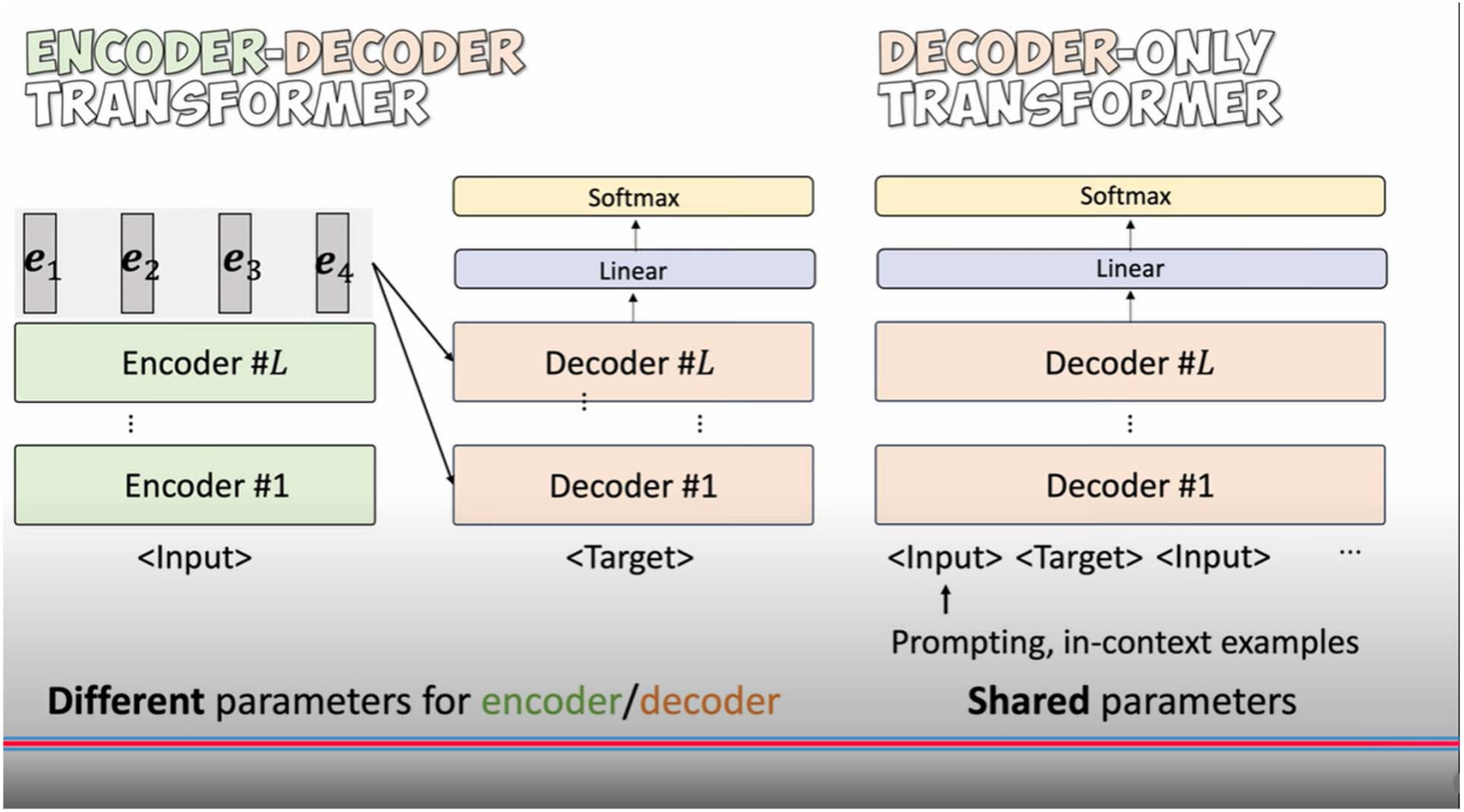
Transformer (from Prof. Jia-Bin Huang)

https://www.youtube.com/watch?v=rcWMRA9E5RI



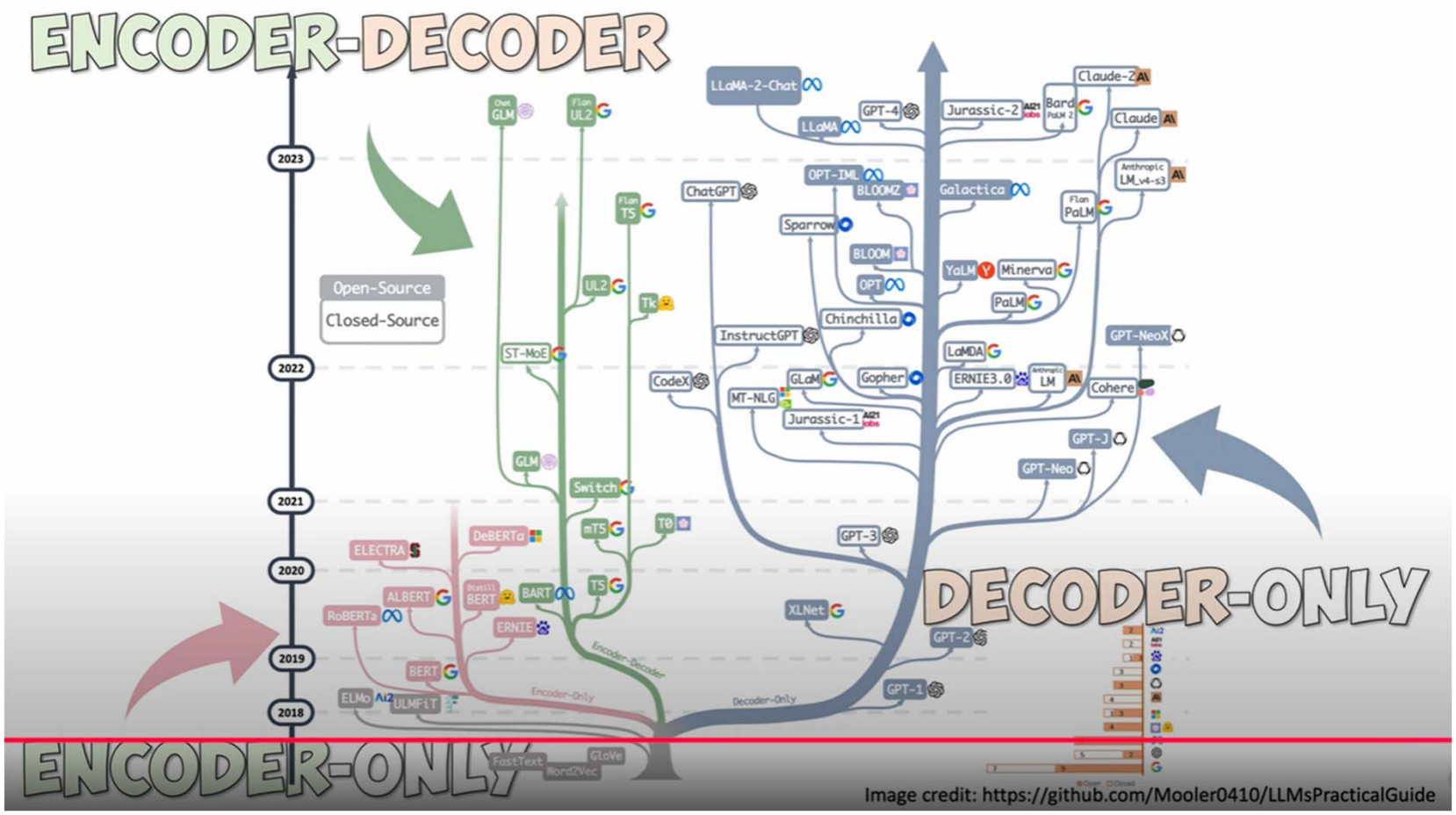
Transformer (from Prof. Jia-Bin Huang)

https://www.youtube.com/watch?v=rcWMRA9E5RI



Transformer (from Prof. Jia-Bin Huang)

https://www.youtube.com/watch?v=rcWMRA9E5RI



Positioning Embedding (from Prof. Jia-Bin Huang)

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

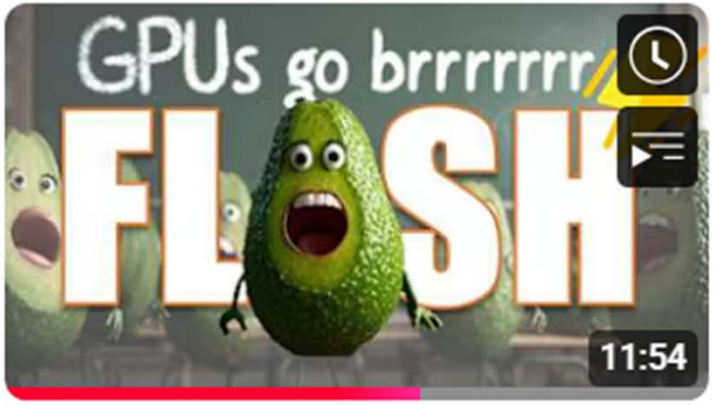
How Rotary Position Embedding Supercharges Modern LLMs...

Jia-Bin Huang • 16K views

Positional information is critical in transformers' understanding of sequences and their ability to generalize beyond training context...

FlashAttention (from Prof. Jia-Bin Huang)

https://www.youtube.com/watch?v=gBMO1JZav44



How FlashAttention Accelerates Generative AI Revolution

Jia-Bin Huang • 16K views

FlashAttention is an IO-aware algorithm for computing attention used in Transformers. It's fast, memory-efficient, and exact. It has...

Outline

- Basics
- VQ-VAE
- Audio codec models
- Transformer
- **Transformer-based text-to-music generation**

MusicLM (from Google)

https://google-research.github.io/seanet/musiclm/examples/

- Given a **text description of music**, generate **waveforms** directly
  - "Piano for study" | "Hip-hop song with violin solo" | "Latin workout" | "Feel-good mandopop indie" | "Salsa for broken hearts"

- **Story mode**

Text prompts

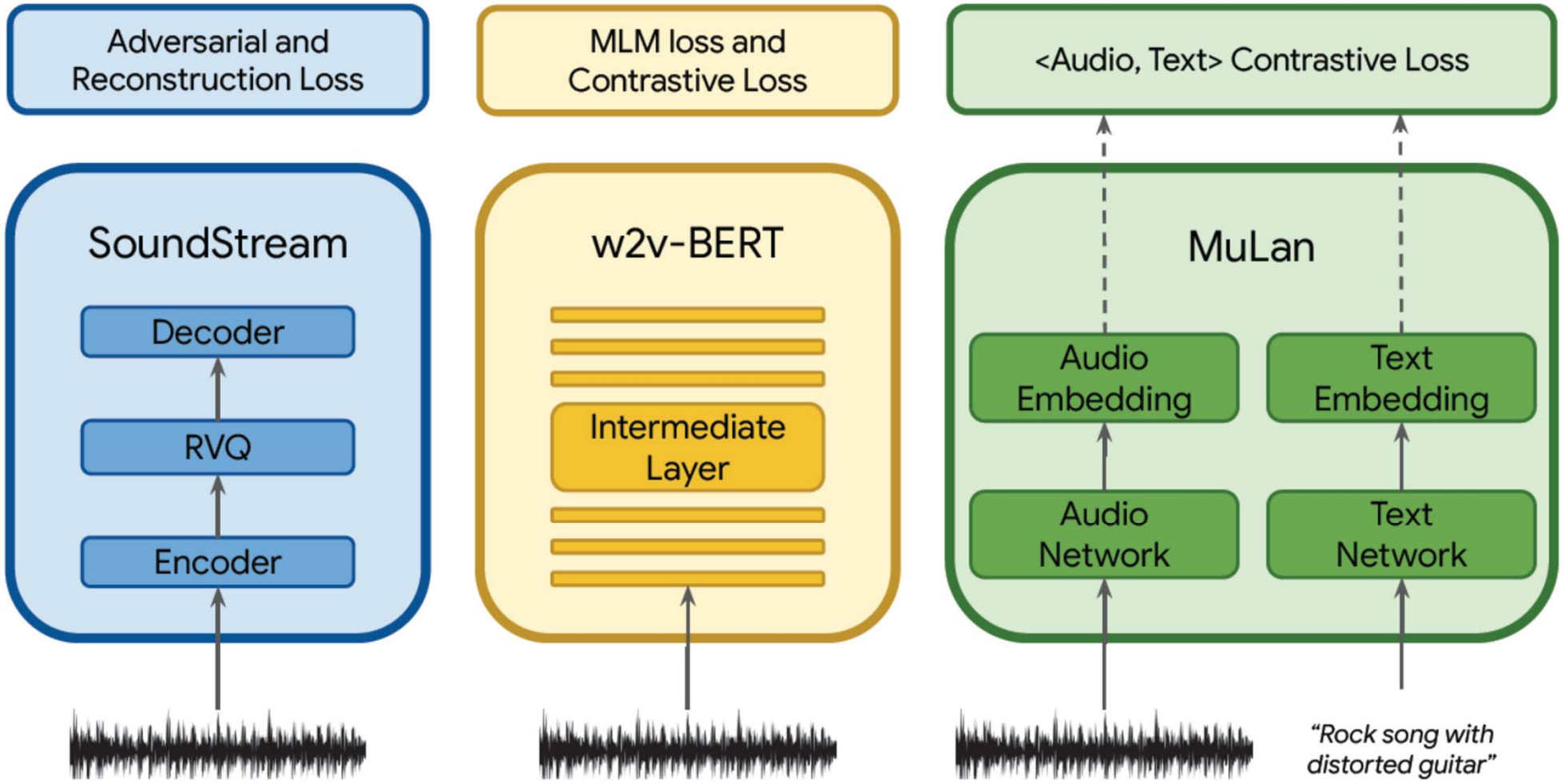
time to meditate (0:00-0:15)
time to wake up (0:15-0:30)
time to run (0:30-0:45)
time to give 100% (0:45-0:60)

- **Text and melody conditioning**
  - Text-conditioned continuation of the provided melody input

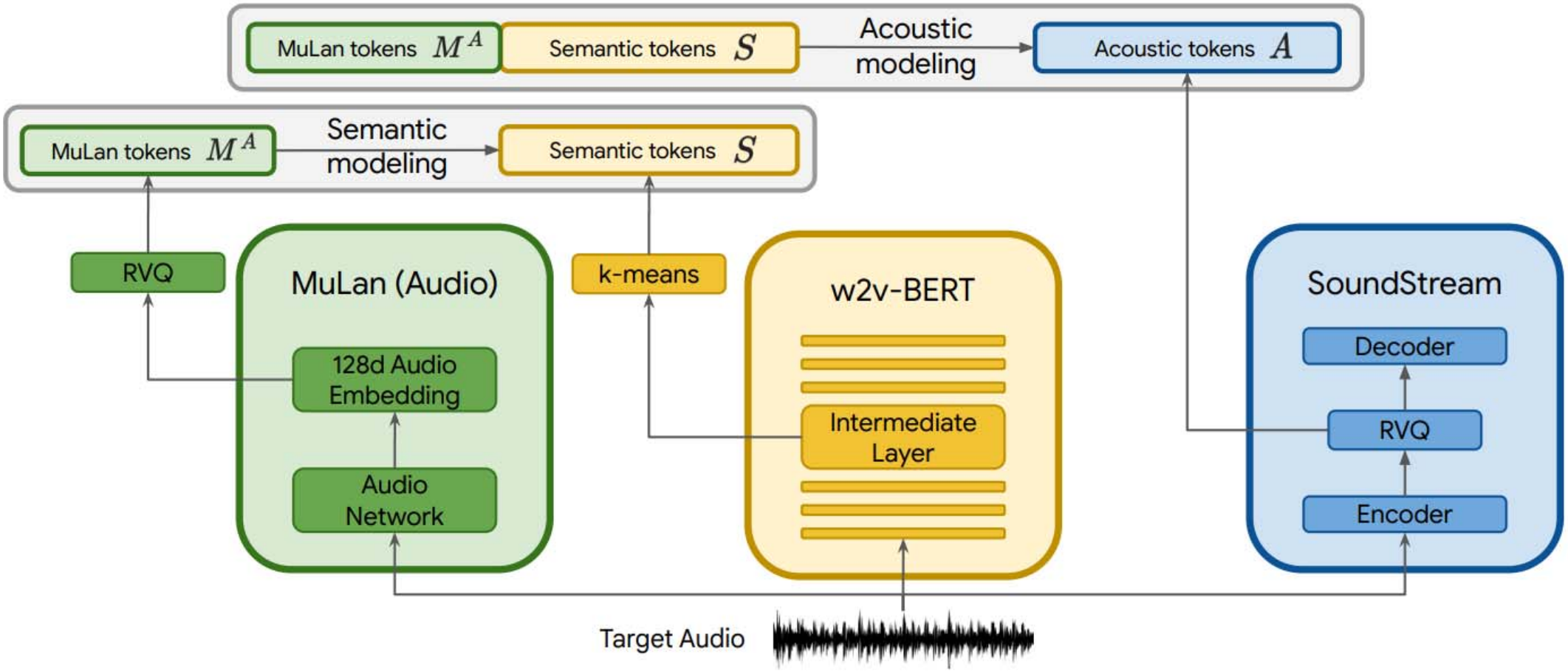
- **Image-to-music**
  - Painting caption conditioning

Ref: Agostinelli et al, "MusicLM: Generating music from text," arXiv 2023

MusicLM's Pretrained Networks



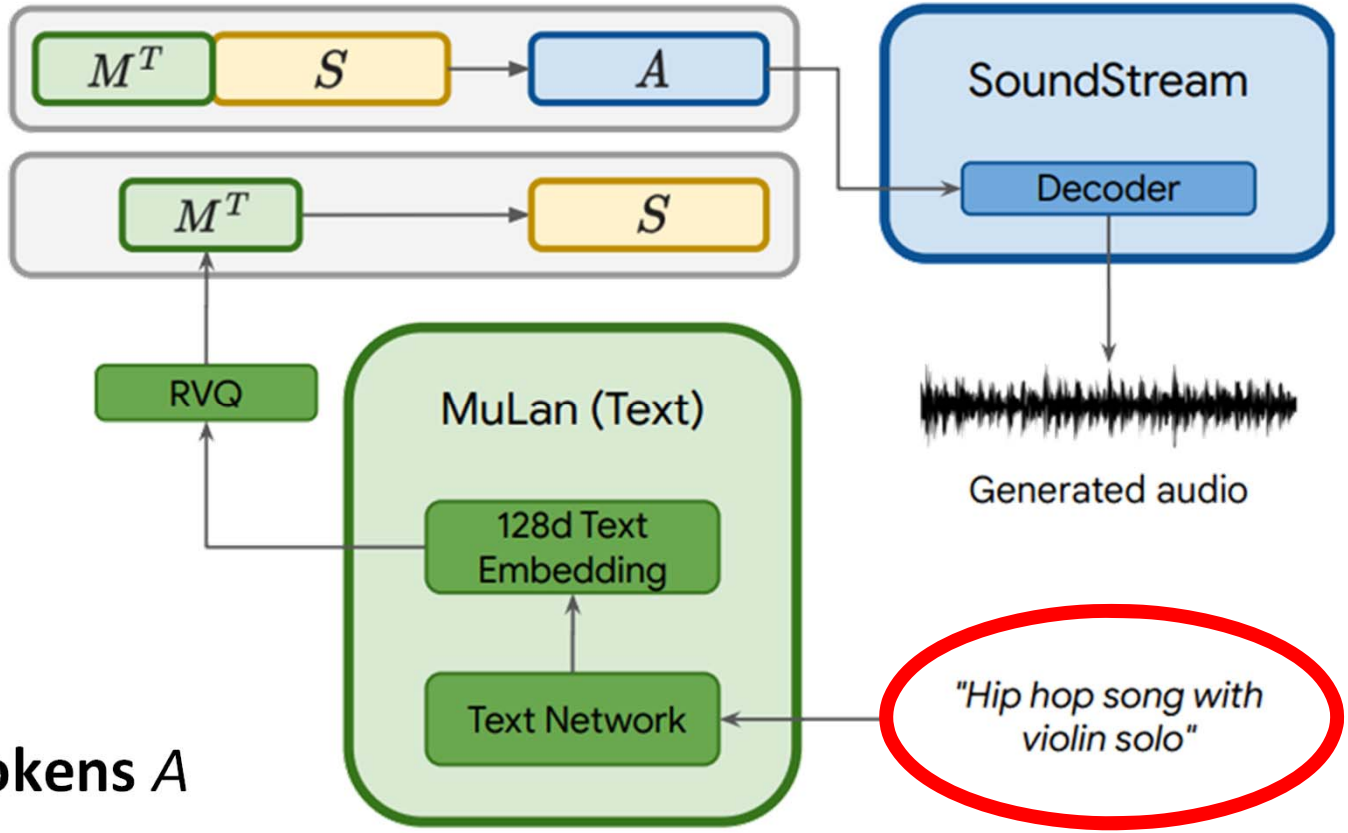
MusicLM Training Procedure



Ref: Agostinelli et al, "MusicLM: Generating music from text," arXiv 2023

MusicLM Inference Procedure

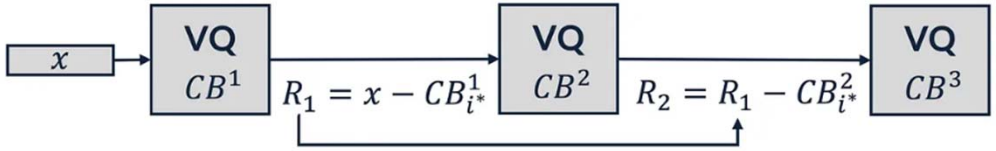
- Given **text tokens** M^T from the input text description
- Generate high-level **semantic tokens** *S* that model the long-term structure of the music to be generated
- Then generate **acoustic tokens** *A* that model the fine details



Ref: Agostinelli et al, "MusicLM: Generating music from text," arXiv 2023

Jukebox vs MusicLM

- **Jukebox**
  - 3 layers of **acoustic tokens**: high, mid, bottom
  - The three layers operate in **different** temporal resolution
  - About **7,234** tokens per second



- **MusicLM**
  - **600 acoustic tokens** per second
    - 12 layers of SoundStream RVQ tokens (each with a vocabulary size of 1024)
    - All the 12 layers operate in the **same** temporal resolution (50 Hz)
  - Plus **25 semantic tokens** per second
    - Semantic tokens from w2v-BERT (codebook size also 1024)

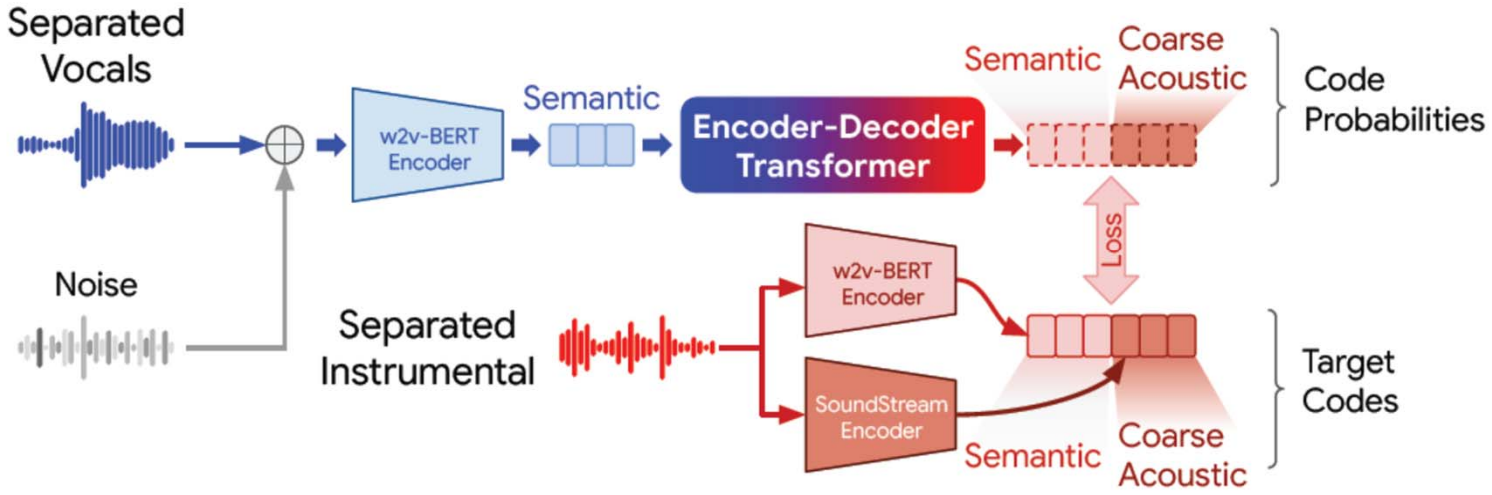
Issues of MusicLM

- **Not** open source; but there are unofficial implementations
  - https://github.com/lucidrains/musiclm-pytorch
  - https://github.com/zhvng/open-musiclm
- Slow (one second 625 tokens; i.e., **625** times of autoregressive sampling)
- Limited controllability
  - The main control signal is the text description (which is usually about the genre and instrumentation of music)
  - Advanced users may want to control, e.g., the **melody** of the singing vocals and the **grooving** (rhythm) of the backing instrumentals

Ref: Agostinelli et al, "MusicLM: Generating music from text," arXiv 2023

SingSong (from Google)

https://storage.googleapis.com/sing-song/index.html



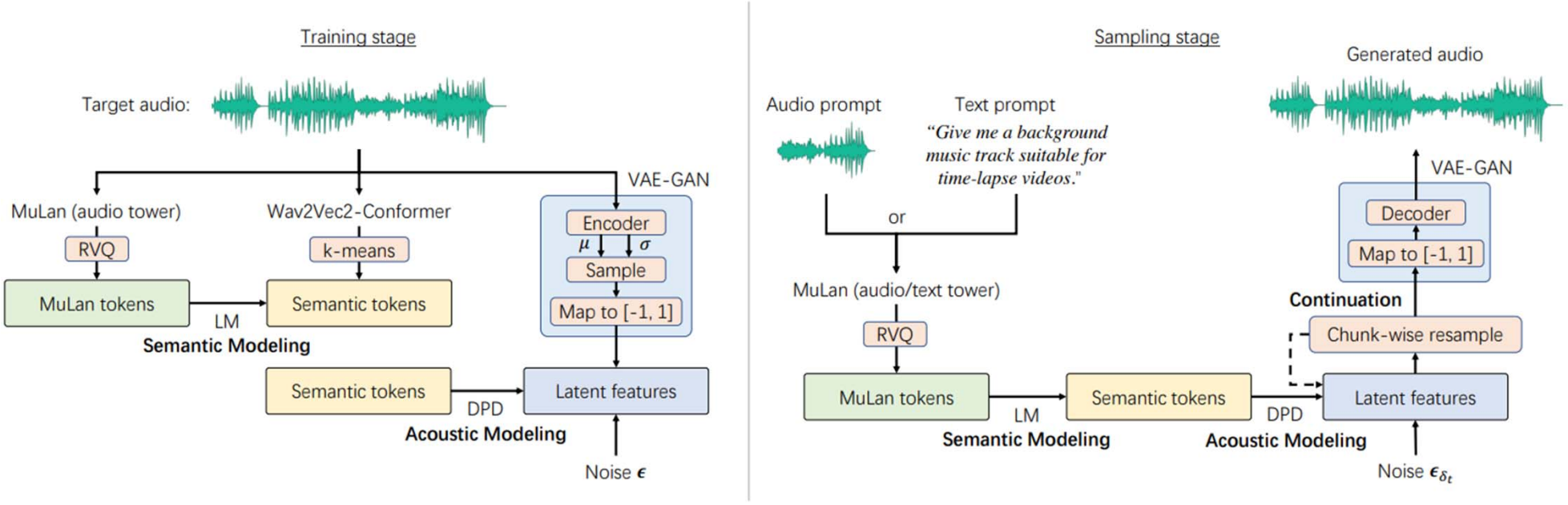
- **Given vocal, generate multi-instrument accompaniment**
  - Apply "blind source separation" to get pairs of vocal/backing tracks
  - Seq2seq: **vocal tokens** as the source, generate the backing tokens
- **Not** open source

Ref: Donahue et al, "SingSong: Generating musical accompaniments from singing," arXiv 2023

MeLoDy (from ByteDance)

https://Efficient-MeLoDy.github.io/

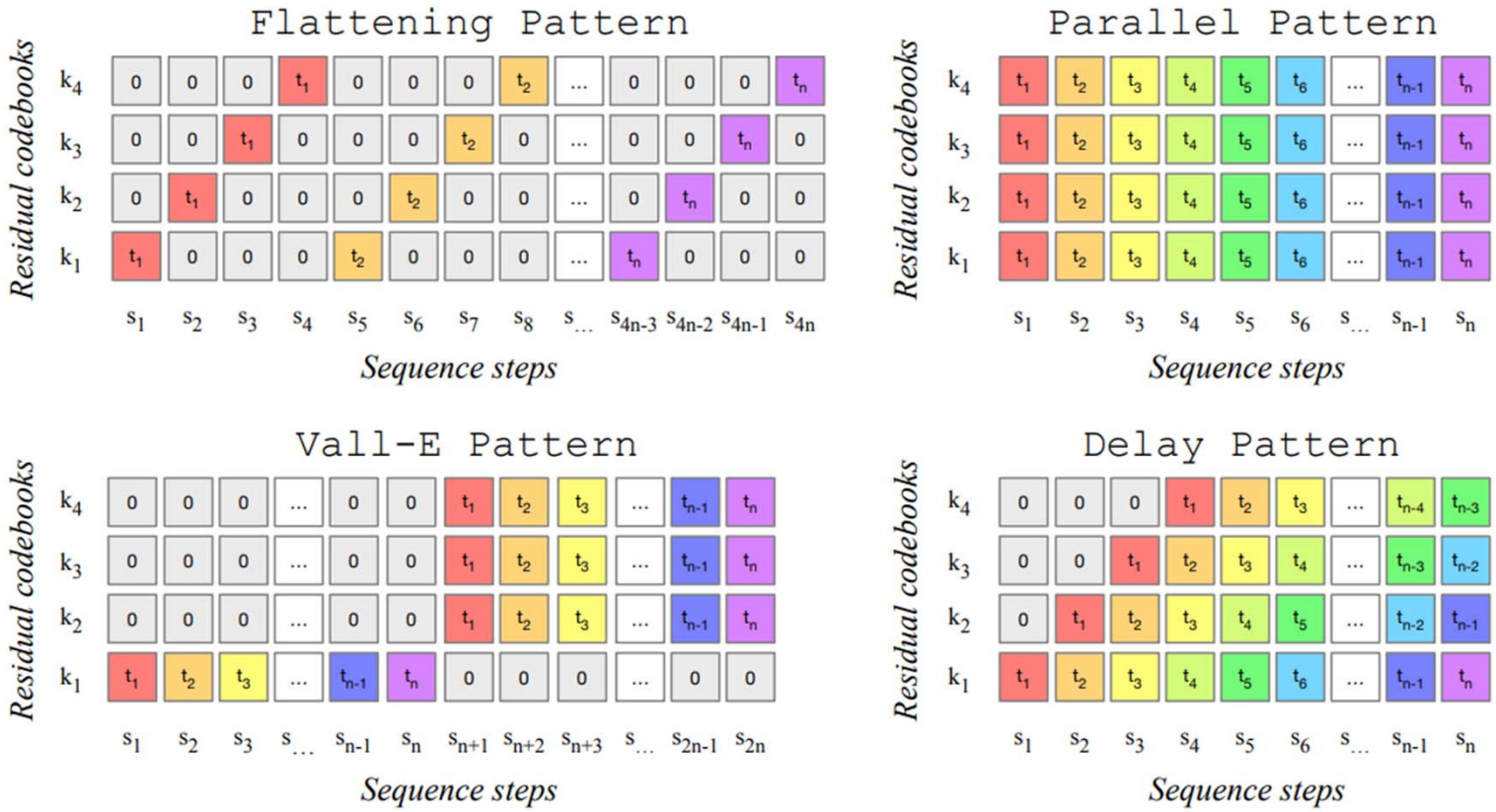
- Replace the LM for acoustic tokens in MusicLM by **latent diffusion**
- **Not** open source



Ref: Lam et al, "Efficient neural music generation," arXiv 2023

MusicGen (from Meta)

- **Parallel prediction** of the Encodec's RVQ tokens
- Open source!

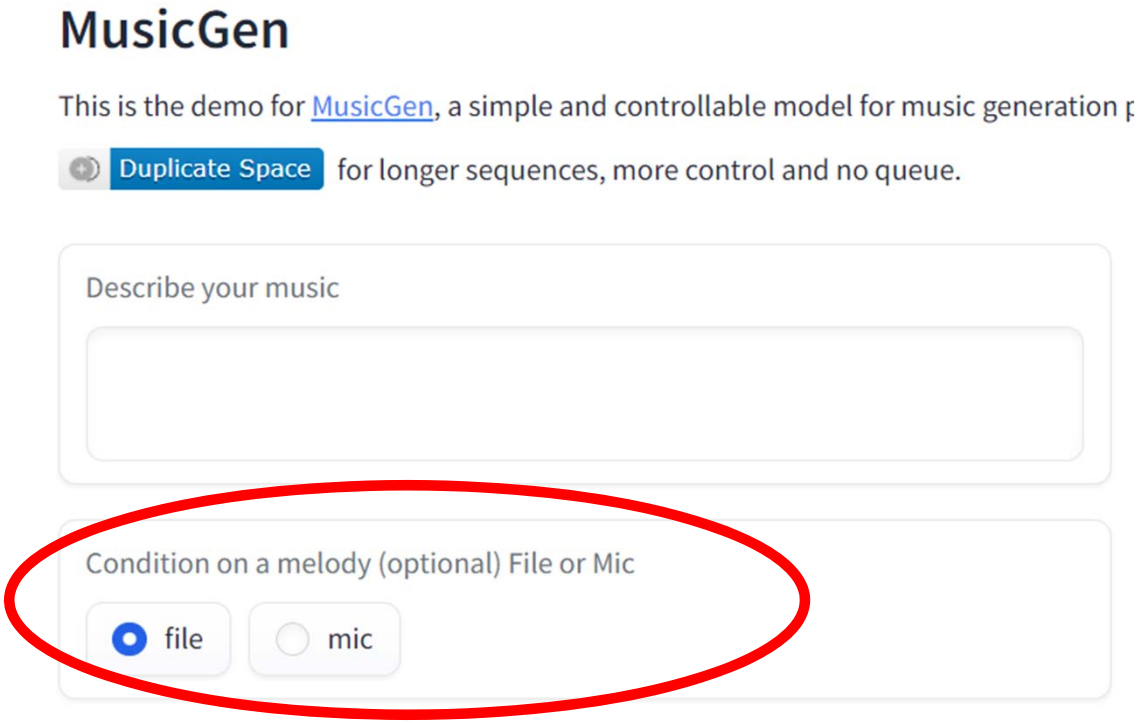


MusicGen (from Meta)

https://huggingface.co/spaces/facebook/MusicGen

https://github.com/facebookresearch/audiocraft

- **Melody conditioning:** Not just as a prompt
  - Tried using **chromagram** of the melody as input but this leads to overfitting
  - Introduce an **information bottleneck** by choosing the **dominant time-frequency bin** in each time step instead



Ref: Copet et al, "Simple and controllable music generation," arXiv 2023

MusicGen: Audio Encoder

- Audio encoder
  - We use a non-causal five layers **EnCodec** model for 32 kHz monophonic audio with a stride of 640, resulting in a frame rate of **50 Hz**
  - The embeddings are quantized with an RVQ with **4 quantizers**, each with a codebook size of **2048**
  - Train the model on one-second audio segments cropped at random
  - One second → **50** autoregressive sampling with *Parallel* or *Delay* pattern
- Transformer model
  - Flash attention from the xFormers
  - Train on 30-sec clips sampled at random from the full track (batch 192), for 1M steps
  - Train 300M, 1.5B and 3.3B parameter models, using respectively 32, 64 and 96 GPUs
  - Top-k sampling with keeping the top 250 tokens and a temperature of 1.0

MusicGen

- While **flattening** improves generation, it comes at a **high computational cost** and similar performance can be reached for a fraction of this cost using a simple **delay** approach

		In Domain Test Set				
CONFIGURATION	Nb. steps	$\text{FAD}_{\text{vgg}} \downarrow$	KL $\downarrow$	$\text{CLAP}_{\text{scr}} \uparrow$	OVL. $\uparrow$	REL. $\uparrow$
Delay	1500	0.96	0.52	0.35	**79.69**$\pm$1.46	79.67$\pm$1.41
Partial Delay	1500	1.51	0.54	0.32	79.13$\pm$1.56	79.67$\pm$1.46
Parallel	1500	2.58	0.62	0.27	72.21$\pm$2.49	80.30$\pm$1.43
VALL-E	3000	1.98	0.56	0.30	74.42$\pm$2.28	76.55$\pm$1.67
Partial Flattening	3000	1.32	0.54	0.34	78.56$\pm$1.86	79.18$\pm$1.49
Flattening	6000	**0.86**	**0.51**	**0.37**	**79.71**$\pm$1.58	**82.03**$\pm$1.1

				In Domain Test Set					
Dim.	Heads	Depth	# Param.	PPL$\downarrow$	$\text{FAD}_{\text{vgg}} \downarrow$	KL $\downarrow$	$\text{CLAP}_{\text{scr}} \uparrow$	OVL. $\uparrow$	REL. $\uparrow$
1024	16	24	300M	56.1	0.96	0.52	0.35	78.3$\pm$1.4	82.5 $\pm$1.6
1536	24	48	1.5B	48.4	0.86	**0.50**	0.35	**81.9**$\pm$1.4	82.9$\pm$1.5
2048	32	48	3.3B	**46.1**	**0.82**	**0.50**	**0.36**	79.2$\pm$1.3	**83.5**$\pm$1.3

MusicGen: Text Encoder

- **No text-audio joint embedding**; simply use a **pure text encoder** such as T5 or Flan-T5
- And **no semantic tokens**; just RVQ acoustic tokens → simple architecture
- No diffusion

Table A.1: Text encoder results. We report results for T5, Flan-T5, and CLAP as text encoders. We observe similar results for T5 and Flan-T5 on all the objective metrics. CLAP encoder performs consistently worse on all the metrics but CLAP score.

	MusicCaps Test Set				
Model	FAD_{vgg} ↓	KL ↓	$CLAP_{scr}$ ↑	Ovl. ↑	Rel. ↑
T5	**3.12**	**1.29**	0.31	84.89±1.78	**82.52±1.31**
Flan-T5	3.36	1.30	**0.32**	**86.25±1.75**	80.83±1.88
CLAP	4.23	1.53	**0.32**	79.79±1.76	77.28±1.54

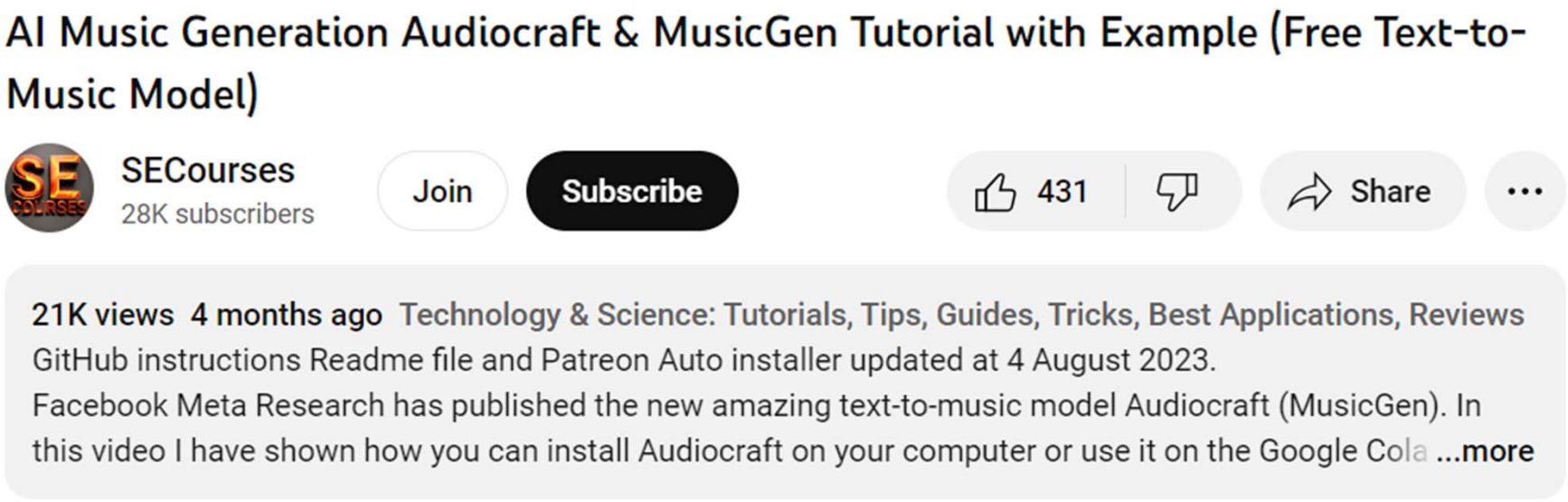
Ref: Copet et al, "Simple and controllable music generation," arXiv 2023

Jukebox vs. MusicLM vs. MusicGen

	Jukebox (2020)	MusicLM (2023.01)	MusicGen (2023.06)
Audio tokens	**7,234** *acoustic tokens* (3-layer hierarchical VQVAE: high, mid, bottom)	**600** *acoustic tokens* (12-layer SoundStream codec) + **25** *semantic tokens* (w2v-BERT) per second	**200** *acoustic tokens* (4-layer EnCodec codec)
Text tokens	No text tokens	**Text tokens representing the user input** (via VQed MuLan: contrastive learning based joint text-audio embedding)	**Text tokens representing the user input** (via T5 or FLAN-T5)
LM	72-layer Transformer	24-layer Transformer	24- or 48- layer Transformer
#Param.	>6B	N/A	300M, 1.5B, or 3.3B

MusicGen Tutorial

https://github.com/FurkanGozukara/Stable-Diffusion/blob/main/Tutorials/AI-Music-Generation-Audiocraft-Tutorial.md



https://www.youtube.com/watch?v=v-YpvPkhdO4

Issues of MusicGen

- Even for MusicGen-small (300M), the authors report its training involves up to **1M training steps** on **32 GPUs** over **20K hours** of private data
  - The specific GPU hardware is not disclosed
  - Assuming **a single RTX 3090 GPU** and BP16 training precision, replicating the training process would require approximately **4,044 GPU hours**

- Other TTM models are even more expensive to train

- From-scratch training is expensive, what can we do? → ***Fine-tuning***

Fine-Tune MusicGen

https://github.com/ylacombe/musicgen-dreamboothing

https://replicate.com/blog/fine-tune-musicgen

- "The aim is to provide tools to easily fine-tune and dreambooth the MusicGen model suite on **small consumer GPUs**, with **little data** and to leverage a series of optimizations & tricks to reduce resource consumption"
  - For example, fine-tune MusicGen Melody on **27 minutes** of Punk music
- Using a few tricks, this fine-tuning run used **10GB** of GPU memory and ran in under **15 minutes** on an A100 GPU
  - Those tricks are LoRA, half-precision, gradient accumulation, gradient checkpointing
  - The largest memory saving comes from **LoRA**

LoRA

- Update only a small subset of parameters
  - By decomposing weights into **low-rank matrices**
  - Then **add back**
  - Possible targets:
    - Query, key, value, and output matrices
    - Feed-forward layer matrices
    - Especially the query and key matrices, for they directly affect the attention scores
- Main benefits
  - **Low training cost**
  - **Avoiding catastrophic forgetting** (preserves pre-trained knowledge)
  - **Modularity**, flexibility, portability

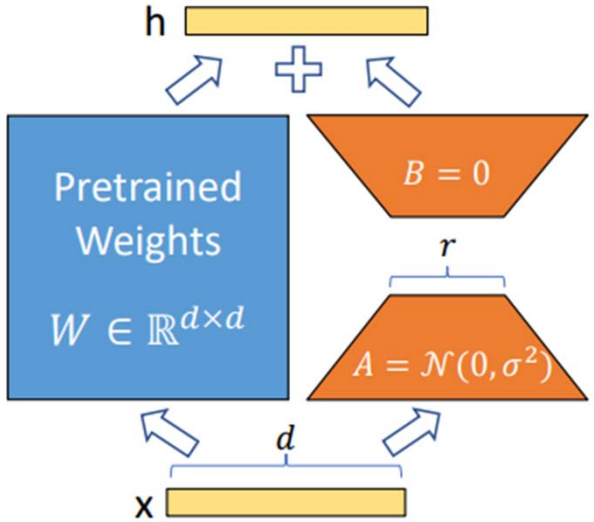


Figure 1: Our reparametrization. We only train A and B.

Ref: Hu et al, "LoRA: Low-rank adaptation of large language models," ICLR 2022

LoRA

- Important hyper-parameters of LoRA
  - **Rank**
    - Common values range from 4 to 64
    - Setting it to 8 or 16 may be enough
  - **Scaling factor alpha**
    - To combine the pretrained weights with the newly added ones
    - Often set to a value like 8, 16, or 32
  - Dropout (optional)
  - Target modules

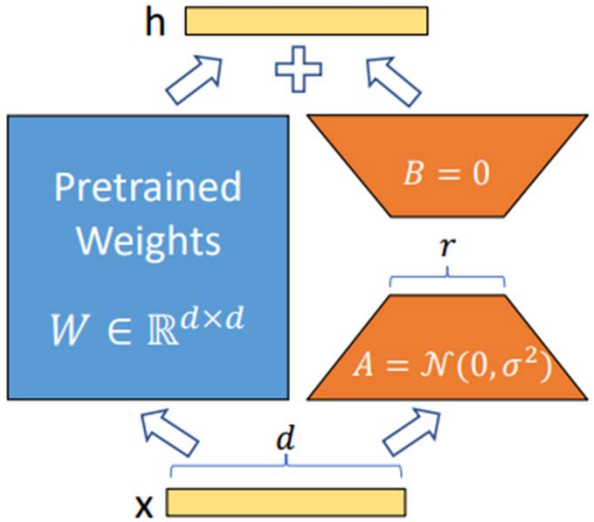


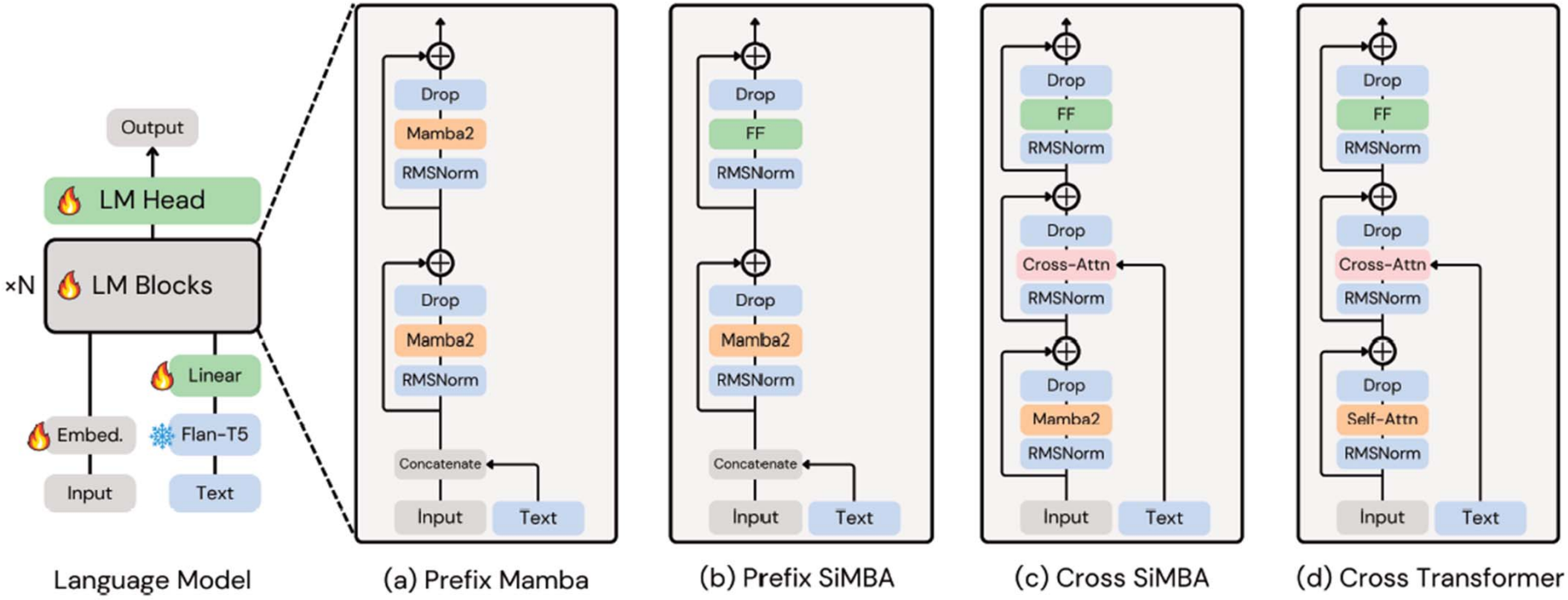
Figure 1: Our reparametrization. We only train A and B.

Ref: Hu et al, "LoRA: Low-rank adaptation of large language models," ICLR 2022

Advanced Topic: Towards Low-Cost Training of TTMs

https://lonian6.github.io/web-exploring-ssm/

- Use state-space models (SSMs) as the backbone instead



Ref: Lee et al, "Exploring state-space-model based language model in music generation," ISMIR-LBD 2025

Advanced Topic: State-Space Models (SSMs) vs Transformers

https://goombalab.github.io/blog/2025/tradeoffs/

- SSMs are modern RNNs