Music generation with Diffusion models

Bojescu Mihai, 2025

Agenda

- Short history
- Data representations
- Existing work
- Proposed solution

Short history

Pre computers era

- Precomputed, randomly-aligned bars
  Example: Mozart Dice Game, 1787
- The parts were composed by an artist ahead of time
- Each and every part was composed such that it was sound with the other ones
- Generation was done by rolling a dice and selecting the parts indicated by it
  Example: Pick part 5, then 8, then 11, then 17

Early computers era

- Rule-based systems
  Example: David Cope (1987). Experiments in music intelligence. University of California.

- Markov chains and Genetic algorithms
  Example: de Mantaras, R. L., & Arcos, J. L. (2002). AI and Music: From Composition to Expressive Performance. AI Magazine, 23(3), 43. https://doi.org/10.1609/aimag.v23i3.1656

Modern computers era

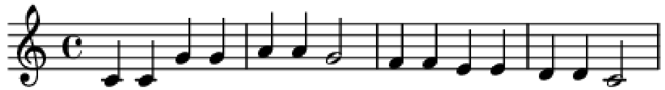
- Neural Networks, with different architectures:
  - GANs
  - VAEs
  - RNNs
  - Transformers
  - Diffusion models

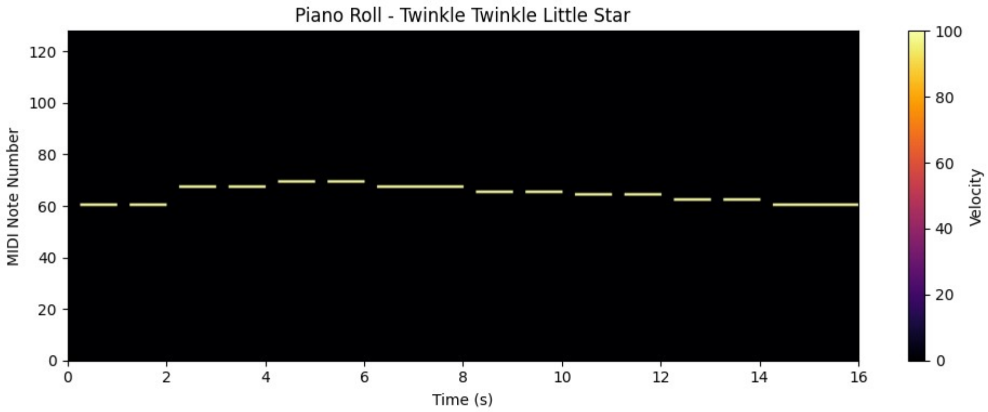
Data representations

Data representations

- Partitures, piano rolls
- Grammars, MIDIs
- Waveforms
- Spectrograms, mel-spectrograms

Partitures, piano rolls



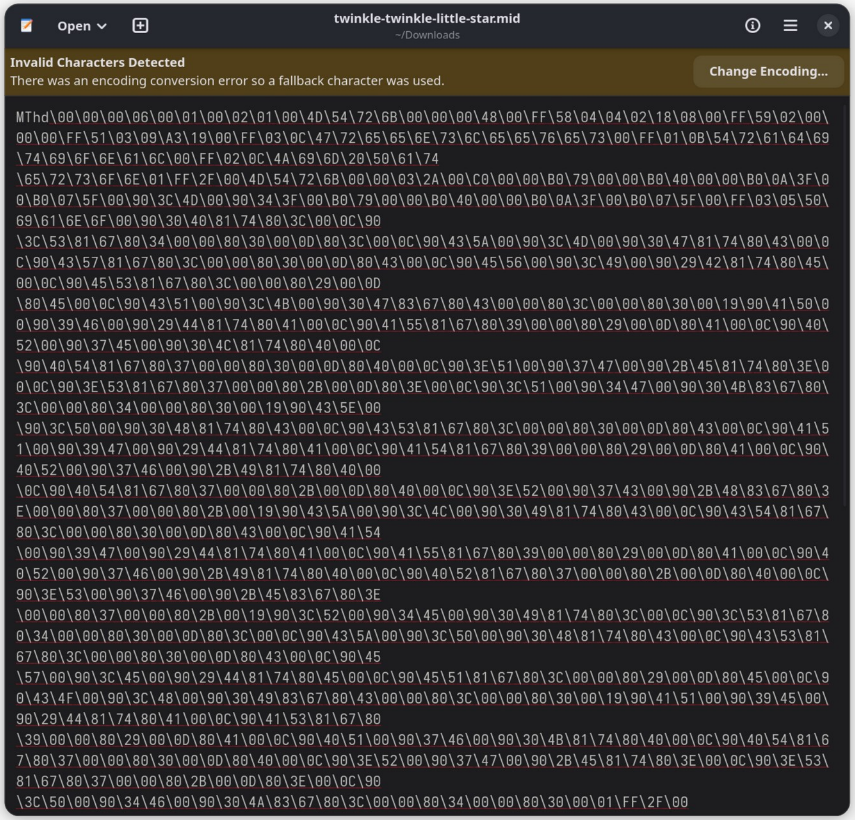


Partitures, piano rolls

- Built by artists as blueprints for music, based on music theory
- Can express tempo, pitch, energy of one or more instruments
- Can be processed by computers after preprocessing
- Can be used by ML algorithms
- Cannot express the imperfections of the music intepreter if interpreted by a computer program

Grammars, MIDIs

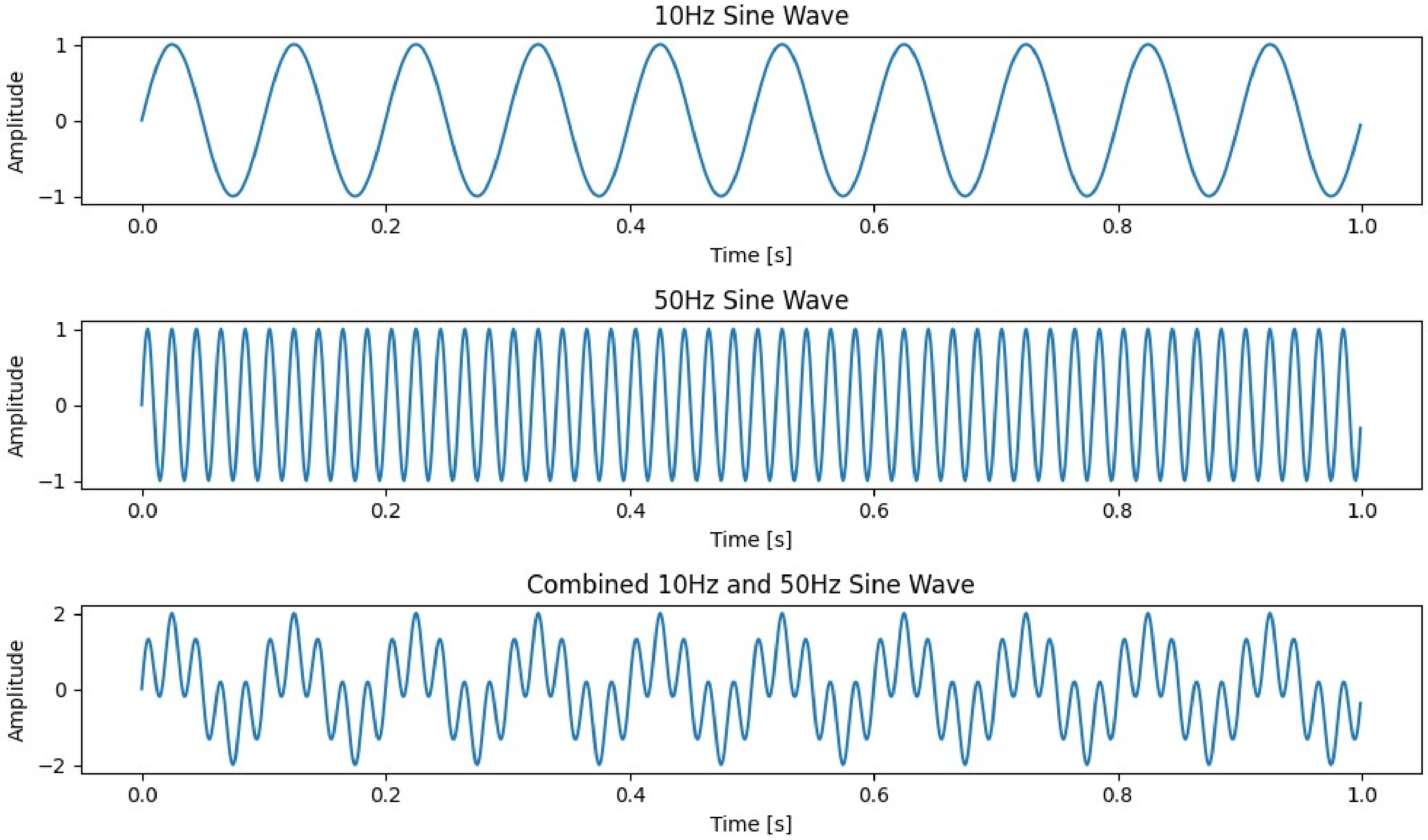
```
# Notes for "Twinkle Twinkle Little Star"
notes = [
    "c'4", "c'", "g'", "g'",
    "a'", "a'", "g'2",
    "f'4", "f'", "e'", "e'",
    "d'", "d'", "c'2",
]
```



Grammars, MIDIs

- Represent another way to represent musical partitures. They are more specific to computers rather than artists
- Can express tempo, pitch, energy of one or more instruments
- Can be processed by computers after preprocessing
- Can be used by ML algorithms
- Cannot express the imperfections of the music intepreter if interpreted by a computer program

Waveforms

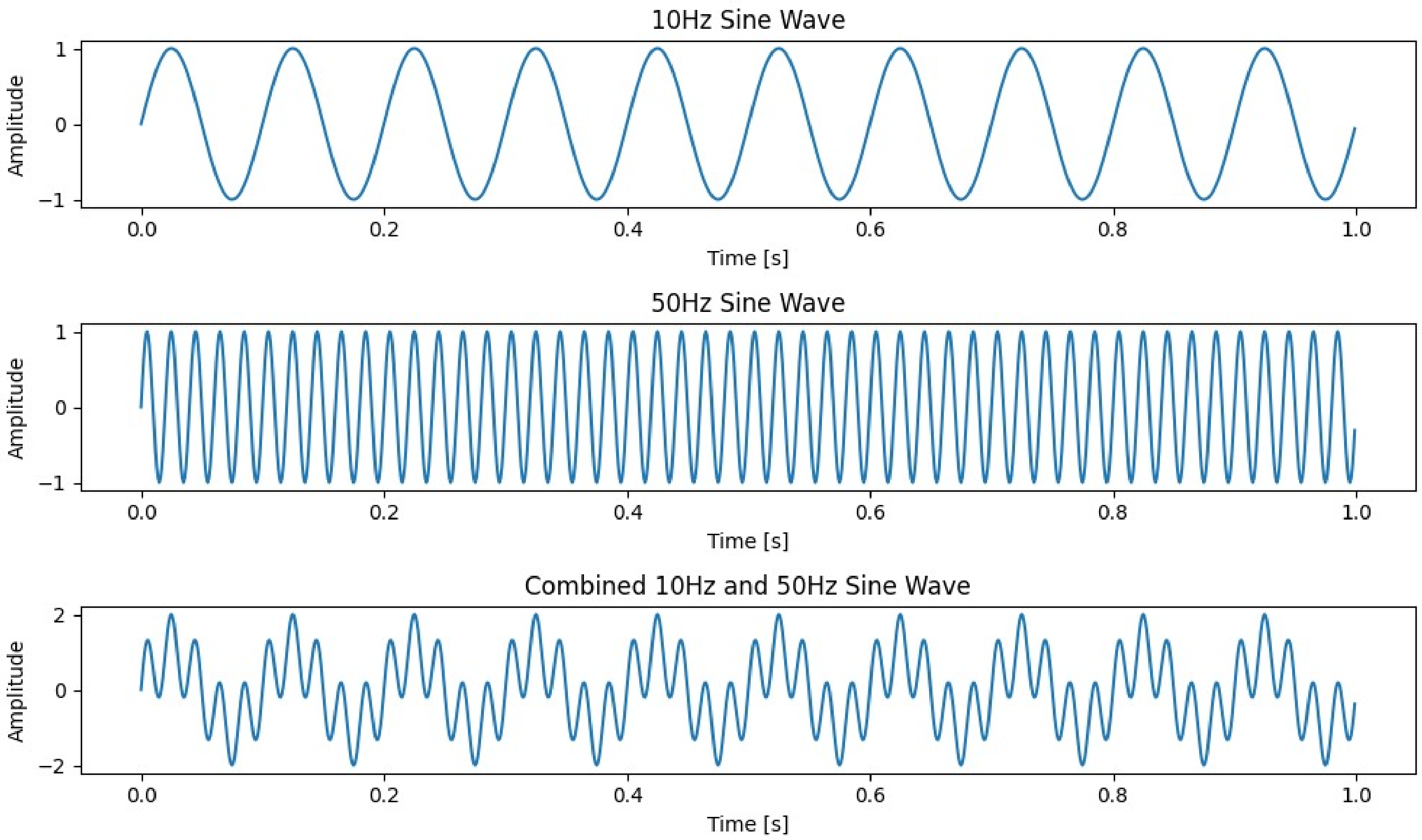


10Hz Sine Wave
Amplitude
1
0
−1
0.0
0.2
0.4
0.6
0.8
1.0
Time [s]
50Hz Sine Wave
Amplitude
1
0
−1
0.0
0.2
0.4
0.6
0.8
1.0
Time [s]
Combined 10Hz and 50Hz Sine Wave
Amplitude
2
0
−2
0.0
0.2
0.4
0.6
0.8
1.0
Time [s]

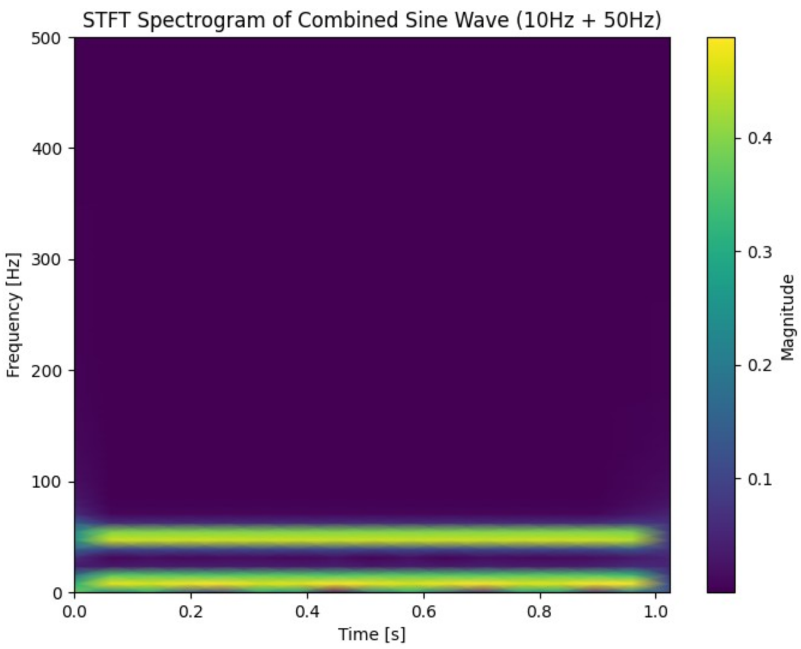
Waveforms

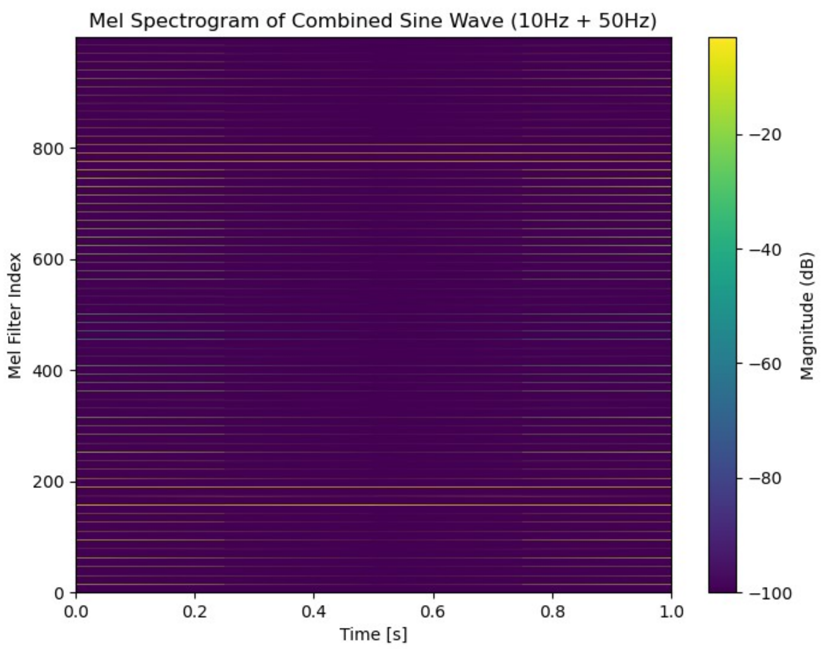
- Represent the way the medium vibrates in order to produce the music
- Can express any sound, doesn't need to follow music theory
  Example: speech
- Can express tempo, pitch, energy of one or more instruments
- Can express the emotion of the artist
- Can be processed by computers after processing
- Often used by ML algorithms

Waveforms


10Hz Sine Wave
Amplitude
Time [s]
50Hz Sine Wave
Amplitude
Time [s]
Combined 10Hz and 50Hz Sine Wave
Amplitude
Time [s]
0.0
0.2
0.4
0.6
0.8
1.0
1
0
−1
2
0
−2

Spectrograms, mel-spectrograms





Spectrograms, mel-spectrograms

- Created by applying Fourier transforms on a waveform in order to extract the frequencies and their contribution from it
- Can express tempo, pitch, energy of one or more instruments
- Can express the imperfections of the interpreter
- Can be processed by computers after processing
- Often used by ML algorithms

Existing work

MuseGAN

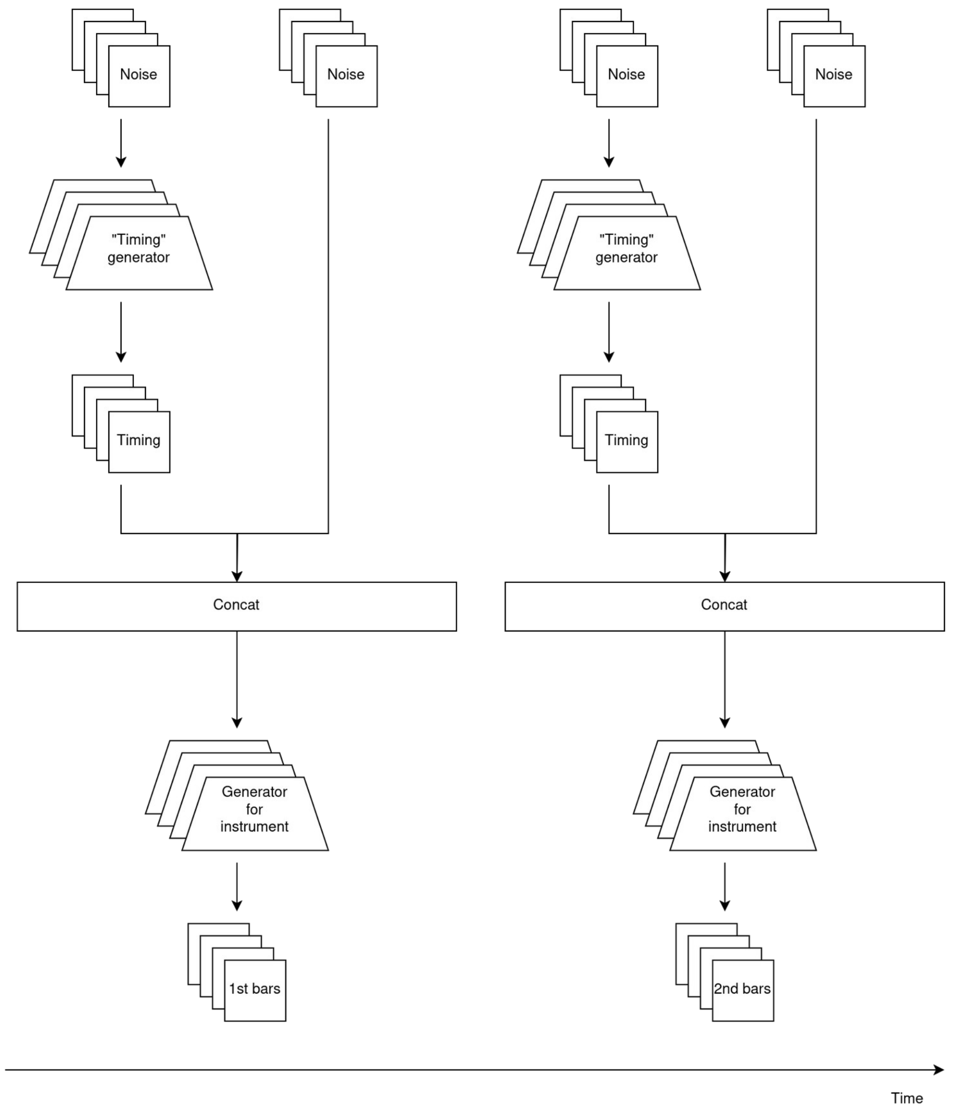
- **Architecture**: Generative-adversarial networks, with CNN generators and CNN discriminators
- **Priming data**: Noise
- **Output data**: Multi-track piano rolls

MuseGAN

- The MIDIs are categorised, filtered and merged:
  - The MIDIs are split into the following 5 categories: bass, drums, guitar, piano, strings & other. The ones that are not in any of these categories are discarded
  - The MIDIs that have only a few notes are merged with MIDIs of similar instruments, in order to prevent data sparsity
    - Even if this introduces noise, the authors preferred it instead of empty bars
  - The MIDIs that are not in 4/4 time and don’t have the “Rock” tag are discarded
    Example: Pink Floyd – Money gets discarded due to being in 7/8 time, even if it is a “Rock” song, with the categories above
- The MIDIs are then transformed into piano rolls

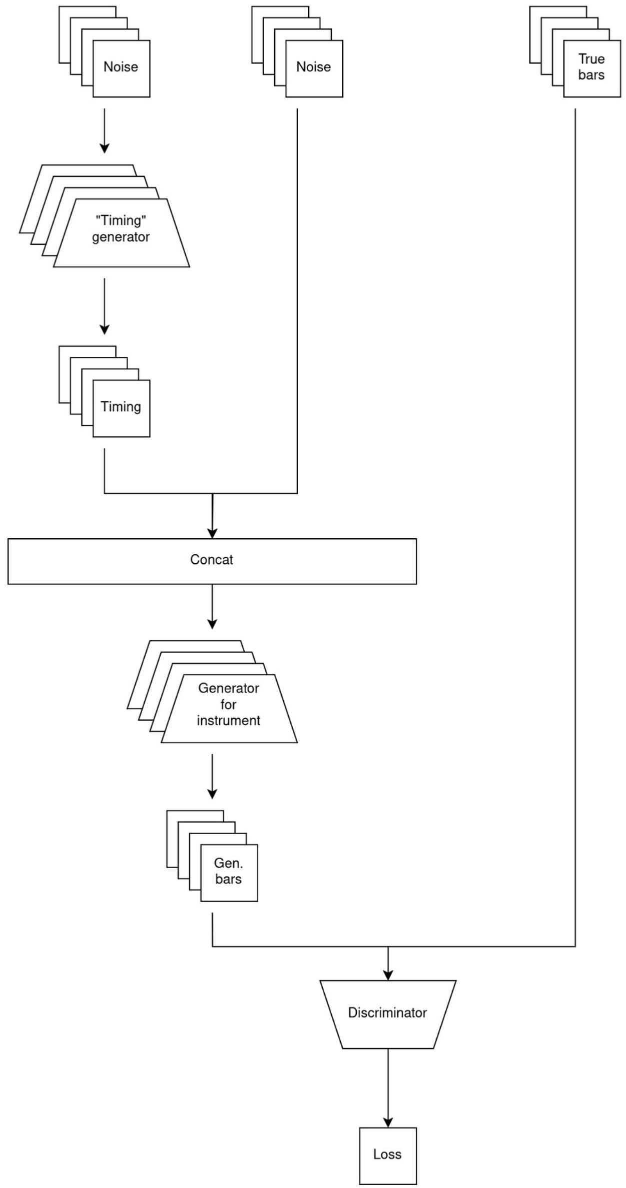
MuseGAN

- Each generator must generate a bar for the instrument it was trained on
- After the bars are generated, the process is repeated
- The generator is trained once every 5 discriminator updates



MuseGAN

- Discriminators are trained 5 times more than the generators. This is done in order to:
  - Prevent mode collapse
  - Stabilise training
  - Ensure that the discriminator provides meaningful gradients to the generator
- The paper proposed using only a single discriminator, regardless of the number of generators
  - This is in order to generalise better



MuseGAN

Advantages

- Produces coherent, mult-track musical pieces
- The outputs can easily be interpreted by humans and machines alike

Disadvantages

- Complex
- High computation cost
- Can produce only a limited amount of musical pieces (inherited from GANs)

Music Transformer

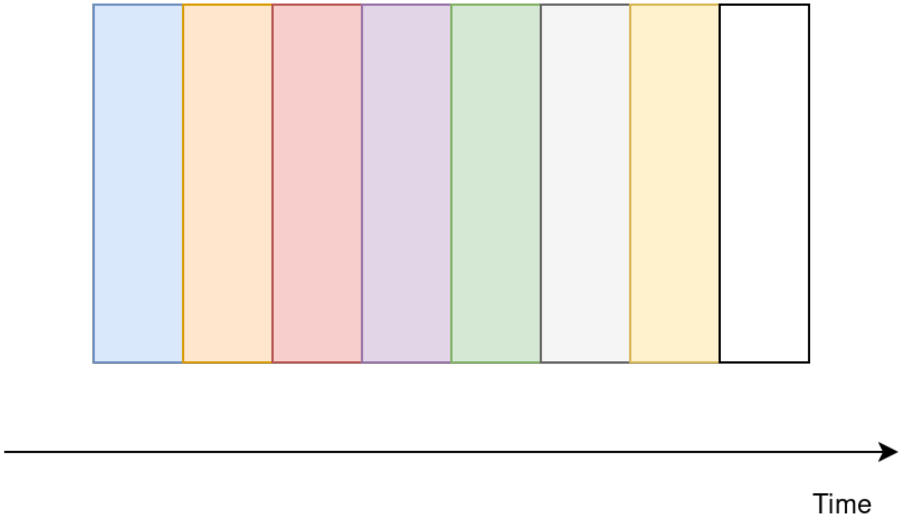
- **Architecture**: Transformers, using Relative Local Attention
- **Priming data**: A sequence of tokens
  Example: Time shift, velocity, first note
- **Output data**: MIDIs

Music Transformer

- The authors observed that scales, arpeggios and other motifs express a grammar specific to a song or genre, and are recurrent periodically
- This observation consists the motivation Relative Local Attention was used, as it could capture these relationships

Music Transformer

- The authors took a language-modelling approach for the symbolic data representation
- Thus the data is expressed a sequence of tokens



Music Transformer

- After the MIDIs are converted into discrete token sequences, the model is trained to predict the next token, given previous tokens
- Thus, the training process is similar to GPTs

Music Transformer

Advantages

- The context-window is large
- Training is faster compared to a GAN
- Good generalisation

Disadvantages

- Works with symbolic data rather than waveforms
- Implementation complexity

Jukebox

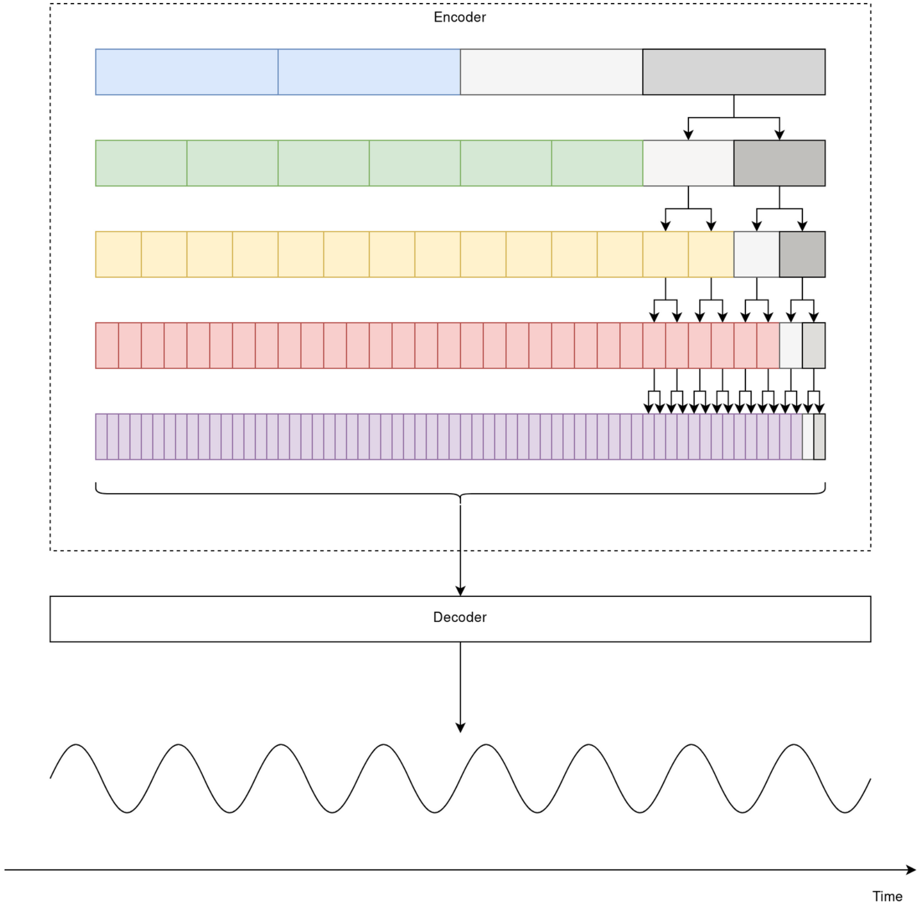
- **Architecture**: VAE, 1D CNNs for encoders, Transformers for decoders
- **Priming data**: A start token, followed by embeddings
  Example: <START>, then genre, artist, timing
- **Output data**: Waveforms

Jukebox

- The audio data:
  - Is standardized to 32 bits floats, and resampled to 44.1 kHz
  - Then it is converted to values between [-1, 1]
  - Then it is randomly augmented by converting to mono, either by selecting one channel, or averaging the two channels
  - Then it is split into 9-second segments, with overapping or non-overlapping chunks
- The text:
  - Is converted into text embeddings, similar to other language models

Jukebox

- Multiple encoders are trained at different hop lengths
- After each encoder generates its output, it is given to the encoders below them:
  - The first encoder produces "coarse" tokens for the next encoder
  - The next encoders upsamples the tokens upsamples the tokens (green, yellow, red)
  - The last encoder upsamples the tokens further, producing "fine-grained" tokens
- The tokens from the last encoder are given to the decoder, which produces high-fidelity audio



Jukebox

- In order to prevent the model to map most of the latent space to only a few codebook embeddings (codebook collapse), the authors chose to randomly restart the embeddings.
- The restart is performed once the mean usage of a codebook vector falls below a threshold

Jukebox

- Training was performed:
  - On 512 V100s for 4 weeks
  - Using Adam optimiser
- Evaluation metrics:
  - Perplexity
  - Musicality
  - Coherence
  - Spectral convergence
  - Codebook utilisation
  - Human rating

Jukebox

Advantages

- Produces waveforms directly
- Produces high-quality audio
- Takes text as inputs
- Allows for flexible conditioning
  Example: genre, artist

Disadvantages

- Highly-complex
- Slow and immensely expensive to train, option being open only to few
- Slow to infer
- Struggles with blending genres

JEN

- **Architecture**: VAE, built with Attention blocks and 1D CNNs, in the shape of a U-Net
- **Priming data**: Text embeddings
  Example: Text embeddings for “Blues song in the style of Robert Johnson”
- **Output data**: Waveforms

JEN

- The paper aims to use image diffusion to generate both autoregressively and non-autoregressively
- This is done in order to improve the generation speed, while also learning the sequential structure of the data

JEN

- In order for the model to be both autoregressive and non-autoregressive, a multi-task training stategy was applied:
  - Text-guided Music Generation: The entire song is masked with Gaussian Noise
  - Music In-Painting: Parts of the song are masked with Gaussian Noise
  - Music Continuation: The end of the song is masked with Gaussian noise
- Two modes are integrated:
  - Unidirectional: To gather sequential dependency
  - Bidirectional: To gather comprehensive context

JEN

- Training was performed:
  - On 8 x A100s for 200k steps, on a dataset of 5000 hours of high-quality private music data
  - Using AdamW optimizer
- Evaluation metrics:
  - Perplexity
  - Fréchet Audio Distance
  - Kullback-Leibler Divergence
  - CLAP Score, using a pre-trained CLAP model
  - Human rating

JEN

Advantages

- High-fidelity audio generation
- More efficient than MuseGAN, Jukebox
- Strong text-music alignment
- The architecture is efficient

Disadvantages

- Partially trained on proprietary datasets
- The paper has some gaps with respect to zero-shot generation, as it is not shown in the metrics
- Inference is not that fast

Proposed solution

Proposed solution

- **Architecture**: Diffusion, built with Attention blocks
- **Priming data**: A start token, followed by text embeddings
  Example: <START>, then genre, artist, timing
- **Output data**: Spectrograms, which can be converted to music

Proposed solution

- The dataset will consist of 1113 copyright-free musical pieces with textual descriptions, from HuggingFace
- The audio data will be transformed:
  - Into spectrograms
  - Then into tokens
- The textual data will be transformed into text embeddings

Proposed solution

Two modes of generation:

- Filling: Pieces of the spectrogram will be randomly masked with Gaussian noise
- Continuation: Gaussian noise is added at the end of the spectrogram

For each mode, the model must remove the noise.

Proposed solution

- In order to mitigate conversion loss between spectrograms and waveforms, the loss from conversion will also be considered during training

Proposed solution

Proposed experiments:

- Try to mask different parts of the spectrograms, similar to JEN
- Try to use relative positioning, along with absolute positioning, similar to Music Transformer
- Investigate with different hop lengths, similar to Jukebox

Thank you!