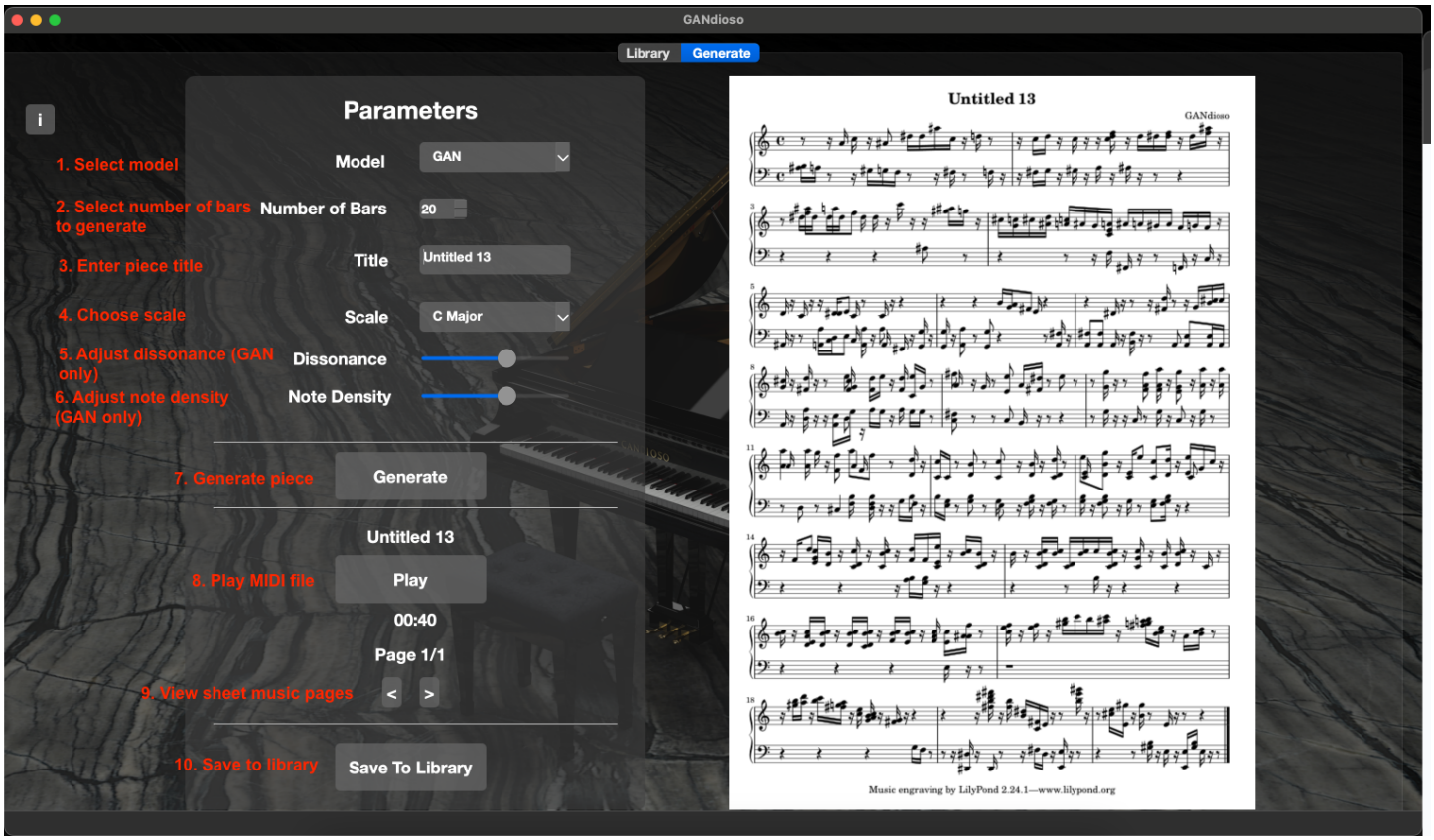
GANdioso User Manual

Daniel Wang

1 How To Use GANdioso

Using GANdioso is very simple: the GUI consists of two tabs, Library and Generate. The Generate tab is where you can generate new pieces, and generated pieces are saved to the Library tab, where you can open the PDF scores and listen to the MIDI files.

1.1 Generating A Piece



1. Go to the 'Generate' tab.

2. Adjust the parameters to your preference. Here is an explanation of what each Parameter does:

   - Number of Bars: The number of bars of music to generate, where each bar is written in 4/4. Accepts values from 10-100.
   - Title: The title of the piece that will be displayed in the Library. An error message will appear if the title is already used or if it exceeds 15 characters.
   - Scale: The scale, or key, of the generated piece. The Generative Adversarial Network (GAN) itself does not generate a piece in the specified key; the GAN output is processed to match the desired key.
   - Dissonance: How dissonant the piece is (the proportion of notes that are not within the selected key).
   - Note density: The number of notes in each section of music.

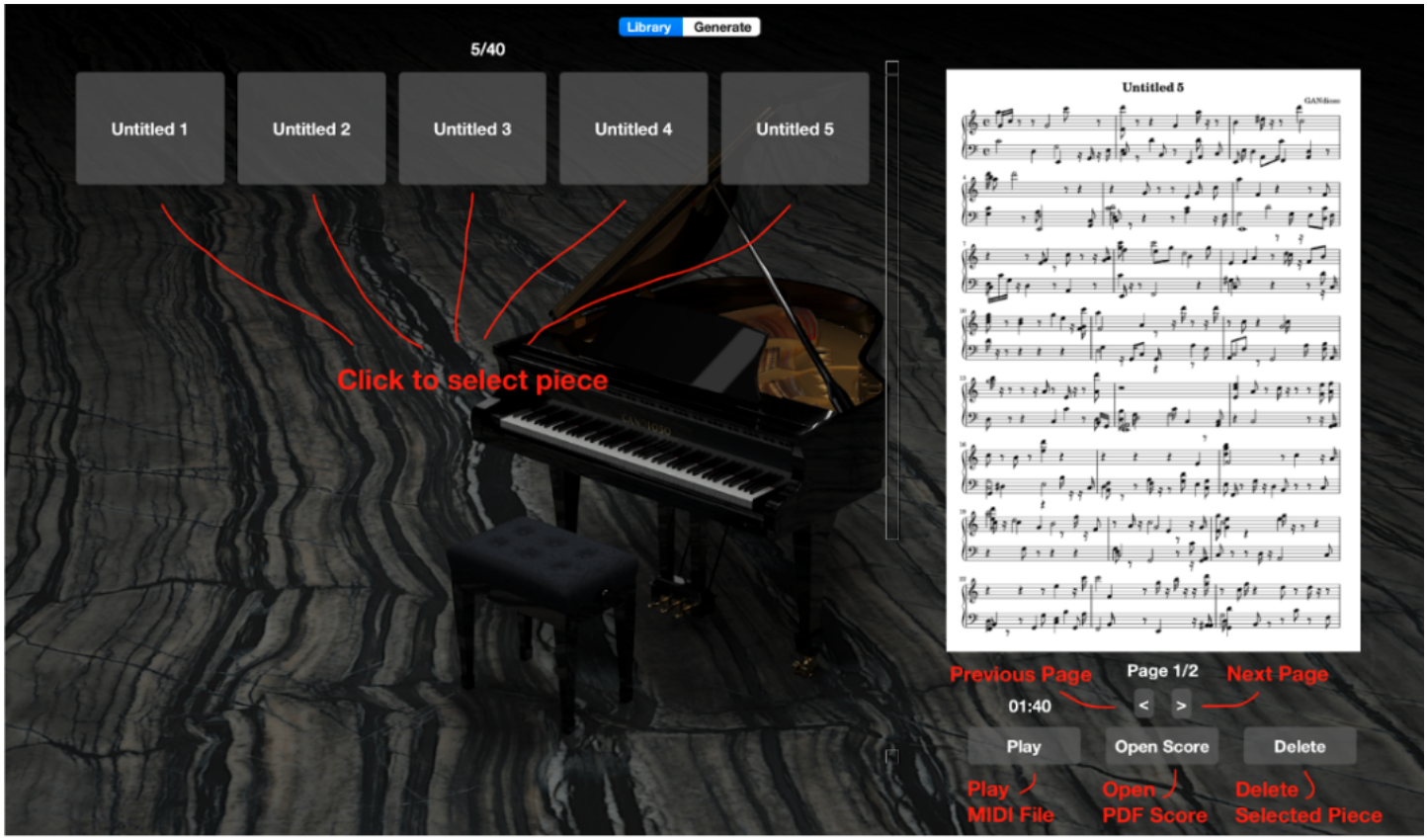
3. Press 'Generate' to generate a new piece.

4. (Optional) Press 'Play' to listen to the piece and 'Stop' to stop listening, and use the arrows to flip through pages of

the sheet music.

5. Press ‘Save To Library’ to save the piece to the library, or ‘Generate’ to generate another piece. Pressing ‘Generate’ will delete the current piece if it has not been saved to the library.

1.2 View And Listen To Saved Pieces



1. Go to the ‘Library’ tab.

2. Click on the button with the name of a piece on it to select that piece.

3. Press ‘Play’ to listen to the piece and ‘Stop’ to stop listening, and press ‘¡’ and ‘¿’ to view all pages of the sheet music.

4. (Optional) Press ‘Open Score’ to open the PDF sheet music of the selected piece.

5. (Optional) Press ‘Delete’ to delete the selected piece.

2 Deep Learning Models For Music Generation In GANdioso

GANdioso currently offers two deep learning models for generating piano music: A Generative Adversarial Network (GAN), and a Transformer model.

2.1 Training Data

Both models are trained using the MAESTRO Dataset (https://magenta.tensorflow.org/datasets/maestro), which contains over 1200 MIDI files of classical piano performances captured with fine alignment (3 ms), lasting about 200 hours in total.

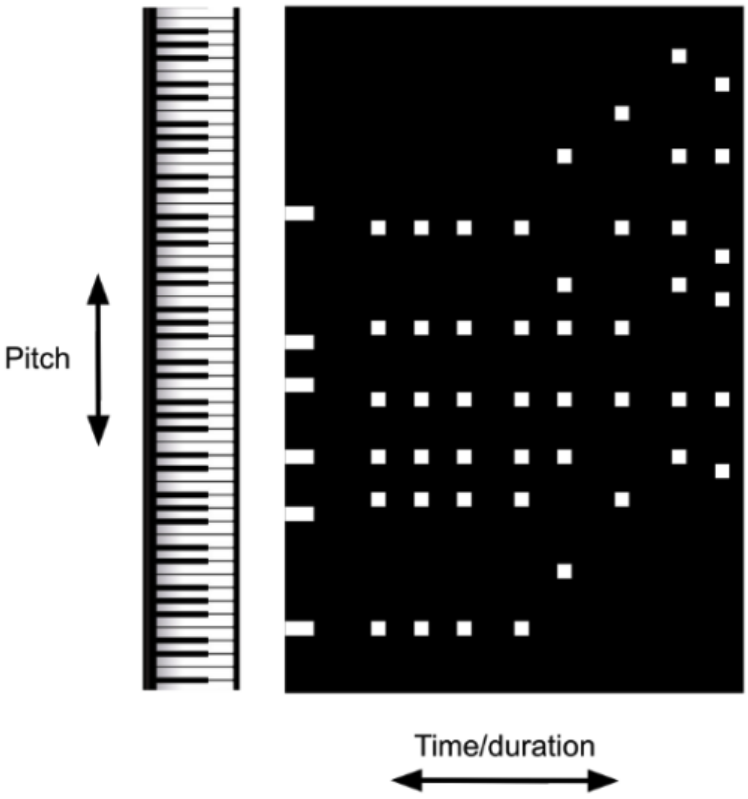
2.2 Generative Adversarial Network (GAN)

The original version of GANdioso generates music using a Generative Adversarial Network (GAN).

Data Preprocessing

Each midi file is encoded as an image.

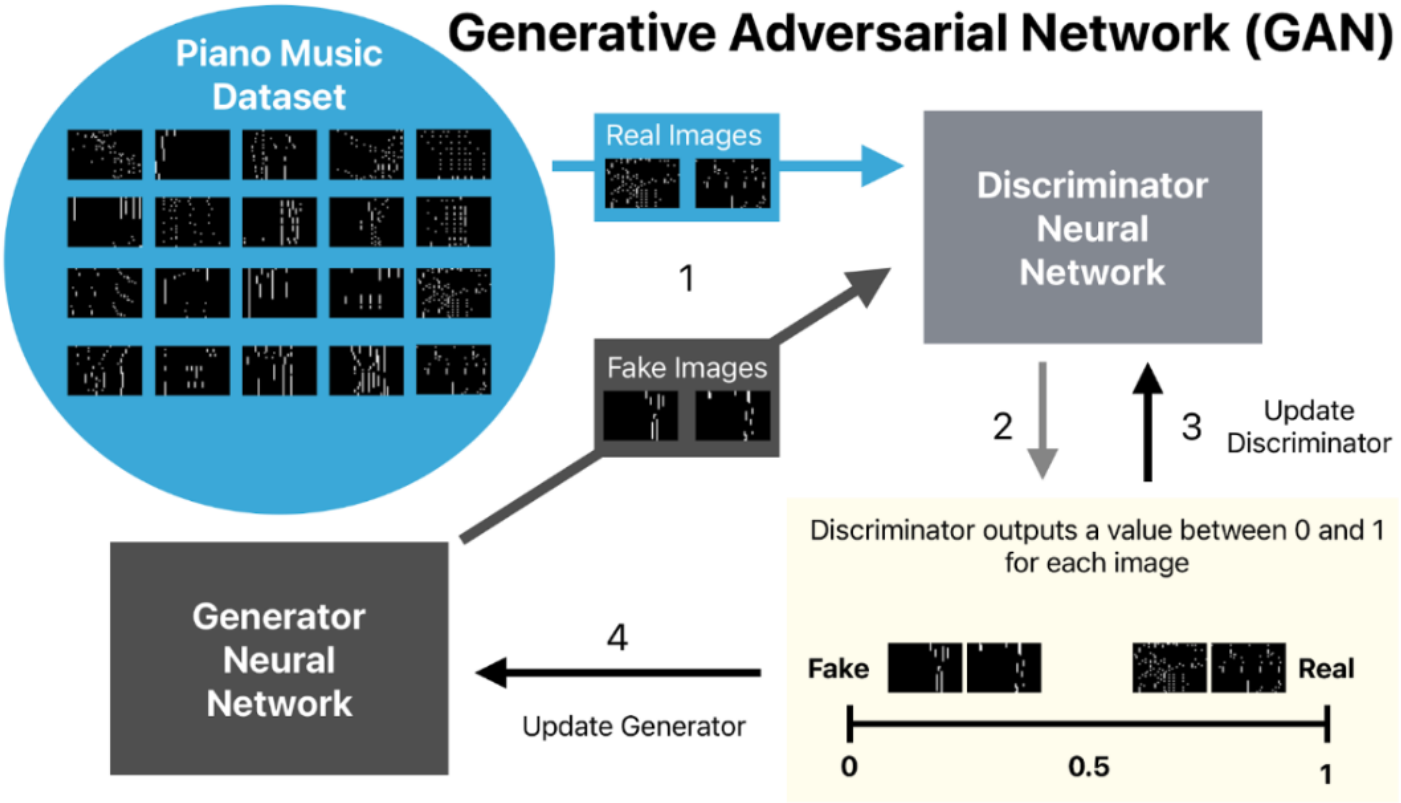
The entire dataset is split into around 200 000 images similar to the one shown here. The horizontal axis represents duration and the vertical axis represents the pitch.



Due to computational constraints, the images only capture the middle 48 keys of the piano where most notes are played. This greatly reduces the computational cost of training the network, but sacrifices a considerable amount of information in the training data.

Model

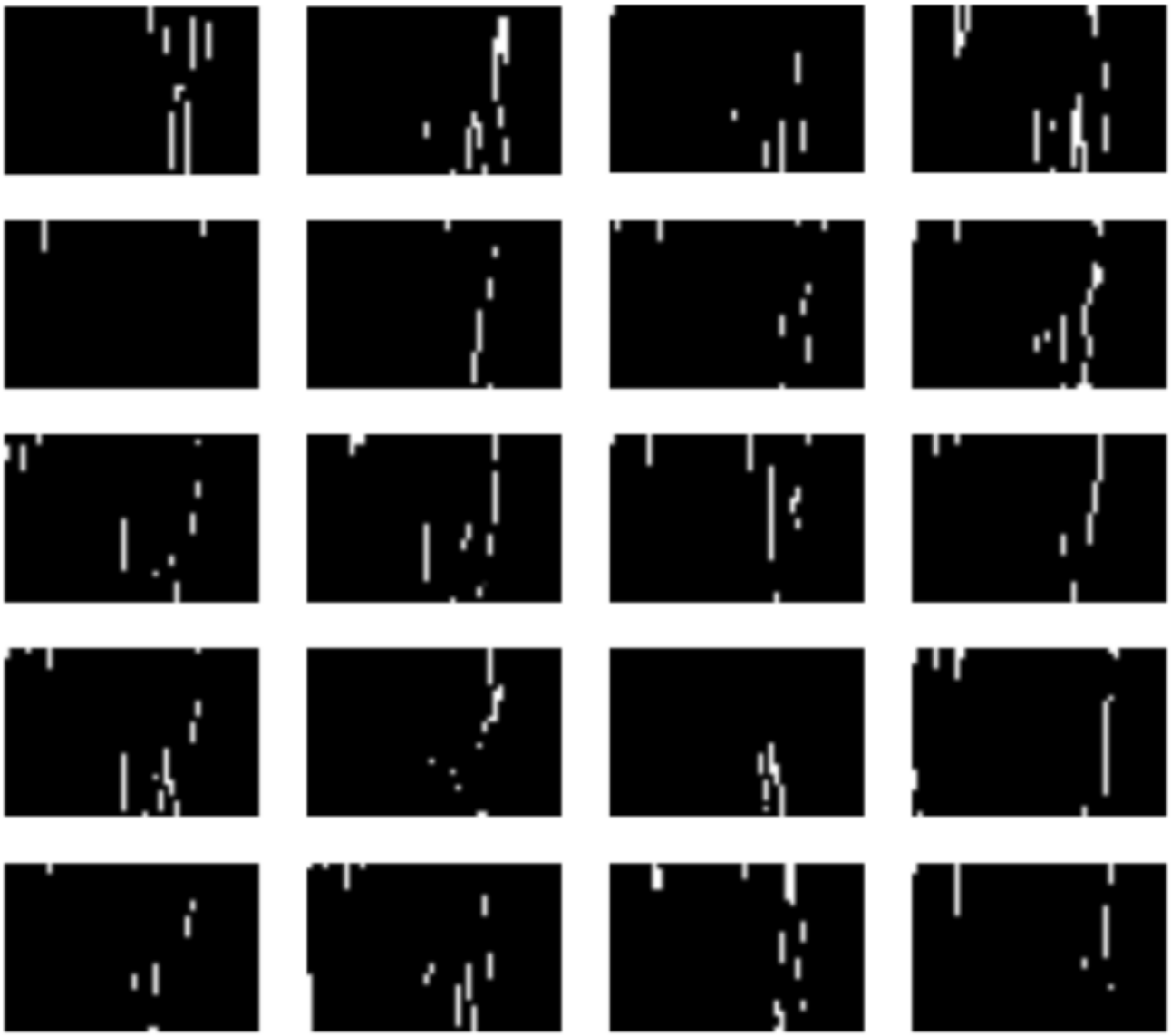
A GAN consists of two neural networks: a Generator and a Discriminator. The generator uses transpose convolutional layers to create an image from a latent vector, and the discriminator is a convolutional binary classifier network that outputs the probability of an image being real (from the training dataset) or fake (created by the generator). Below is a simplified overview of how a GAN works.



1. Real images taken from the dataset and fake images generated by the generator are fed into the discriminator.
2. The discriminator determines whether a given image is real or fake by calculating a value between 0 and 1 for each image, where 0 means that the discriminator thinks the image is fake and 1 means that it thinks the image is real. This value is used to determine how accurate the discriminator is.
3. The discriminator parameters are updated so that it learns to more accurately distinguish between real and fake images.
4. The generator is updated so that it can generate more realistic images that trick the discriminator into thinking they are real when they are actually fake.

These steps are repeated many times, and both the generator and discriminator improve during this process until the generator is eventually able to generate realistic images.

Once the GAN has been trained, it is able to generate images like these:

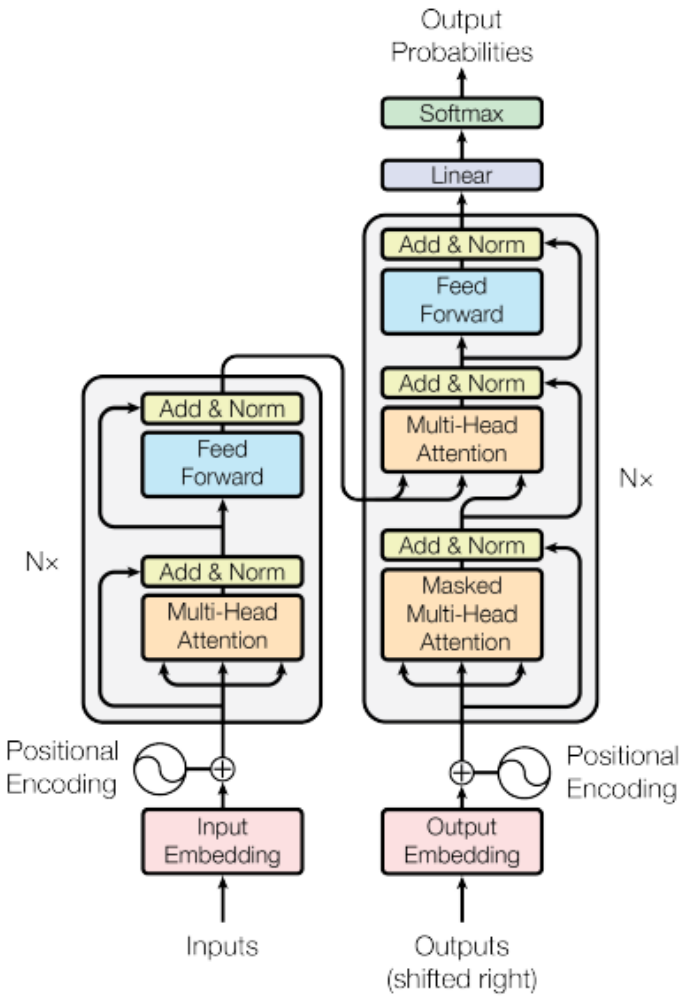


The process of turning these images into MIDI and PDF files are described in Section 3.

While the GAN is able to generate partially realistic music, it has the fundamental problem that each image is generated independently of one another. This means that there is no deliberate long-term structure, and the images are essentially randomly concatenated together to form a full piece of music. To draw an analogy, this is like writing a bunch of unrelated sentences and putting them together to form a paragraph - it wouldn't make any sense. A different architecture is required to achieve longer phrases of coherent music.

2.3 Transformer

The Transformer architecture, introduced in *Attention Is All You Need* (Vaswani et al., 2017) for sequence processing tasks, is responsible for the rapid progress in natural language processing in recent years, and LLMs such as ChatGPT.



Music is similar to language in the sense that both have a vocabulary of notes/alphabet letters, which are arranged into sequences to create coherent music or text. For these reasons, Transformers have been widely adopted in recent music generation models, where researchers have modified the original Transformer architecture to address domain-specific problems; most notably, the long-term structure of music. *Pop Music Transformer: Beat-based Modeling and Generation of Expressive Pop Piano Compositions* (Huang & Yang, 2020) implements the Transformer-XL architecture, which makes use of segment-level recurrence and relative positional encoding to allow longer contexts to be utilised. More recently, *Museformer: Transformer with Fine- and Coarse-Grained Attention for Music Generation* (Yu et al., 2022) implements attention in a way such that tokens of a specific bar attend to all tokens of structure-related bars (fine-grained attention) and single summary tokens of all other bars (coarse-grained attention) to mitigate the quadratic complexity of the attention mechanism. The structure-related bars are set to be, perhaps not so elegantly, the previous 1st, 2nd, 4th, 8th, 12th, 16th, 24th, and 32nd bar based on similarity statistics over the whole dataset.

In GANdioso, a vanilla autoregressive Transformer Decoder has been implemented. Although it does not make use of the

techniques mentioned above, it achieves noticeably better results than the GAN.

Data Preprocessing

The data preprocessing is inspired by the REMI representation introduced in *Pop Music Transformer: Beat-based Modeling and Generation of Expressive Pop Piano Compositions* (Huang & Yang, 2020). Each MIDI file is represented as a sequence of three types of tokens:

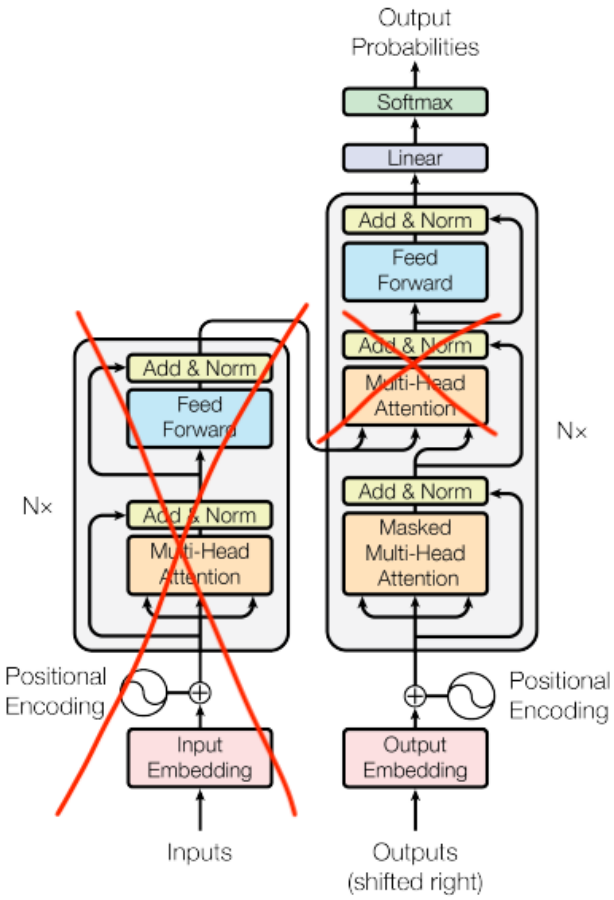
- Note tokens: Note tokens are in the format n{pitch}, where pitch is an integer from 21-108 corresponding to one of the 88 keys on a piano. Note tokens indicate that a specific key is being pressed.
- Duration tokens: Duration tokens are in the format d{duration}, where duration is an integer representing the number of semiquavers that a note is held down for. Duration tokens always follow immediately after Note tokens.
- Time tokens: Time tokens are in the format t{duration} where duration is an integer representing the number of semiquavers between the onset of the previous note and the next one. If there is no Time token between successive notes, then the notes are played simultaneously as a chord.

Start and Pad tokens are also added at the start and end of pieces. In this representation, piano pieces contains around 10000 tokens on average. The model is trained on sequences of 512 tokens. The tokenized representation of a short phrase of piano music looks like this:

... 'n78', 'd4', 't3', 'n82', 'd1', 't2', 'n70', 'd11', 'n68', 'd9', 't2', 'n74', 'd4', 't3', 'n82', 'd3', 't4', ...

Model

The model is an autoregressive Transformer Decoder. This architecture omits the encoder block (left) from original Transformer architecture and only uses the decoder block (right) of the with the encoder-decoder cross-attention removed.



The model receives the start token as input, and generates the first token. Then, it uses the start token and first token as inputs to generate the second token, and so on. In this autoregressive procedure, the next token is conditioned on all previous tokens, enabling the model to generate all 512 tokens beginning with just the start token. During training, entire 512 token sequences are shifted right by one position and input into the model, with the unshifted 512 token sequences as labels. The model then predicts all 512 tokens in a single forward pass. To prevent the model from using future tokens to predict the current token, an upper triangular mask is applied during the attention computation that zeroes out the attention value between any given token and all of its successive tokens. Because the model produces random outputs early on during training, the teacher-forcing strategy of inputting 512 token sequences from the dataset as opposed to sampling them autoregressively saves the model from learning to generate new tokens given random tokens, and instead learns to generate new tokens given a coherent sequence of tokens throughout the entire training process.

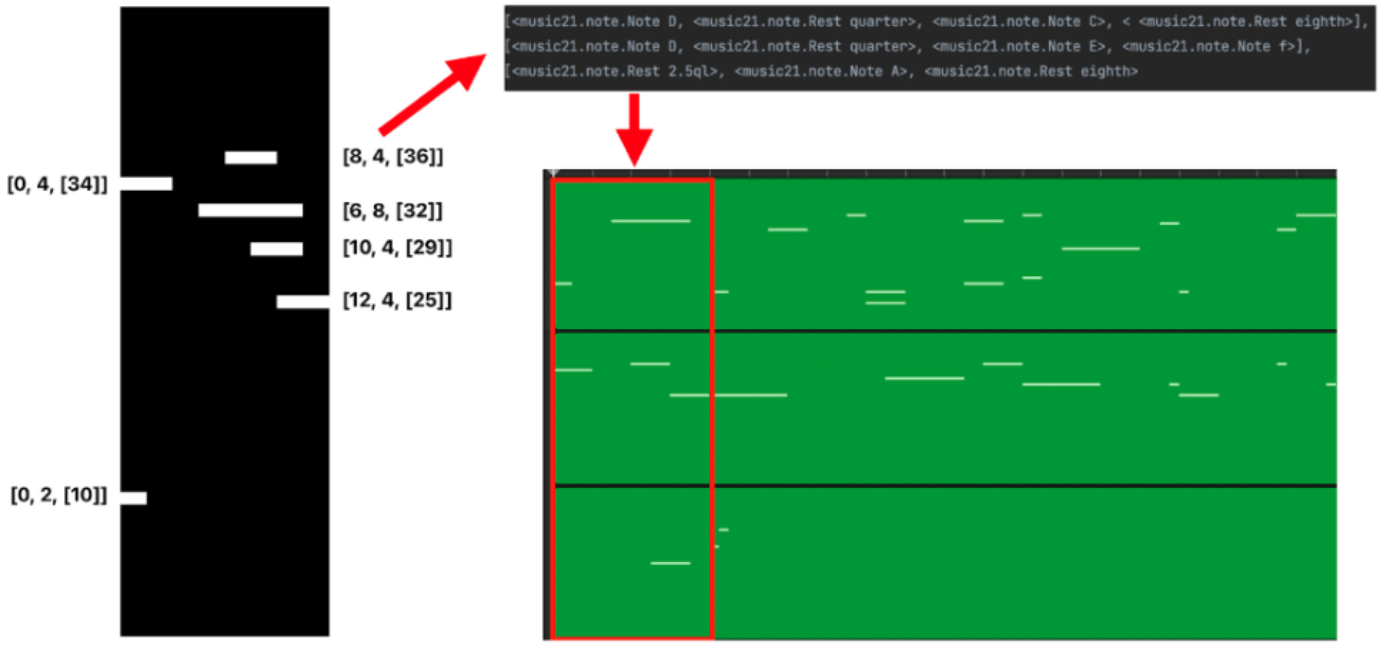
3 From Model Outputs to Sheet Music and MIDI Audio

GANdioso makes use of the Music21 library for MIDI file generation and Abjad/Lilypond for sheet music. The GAN outputs a series of images, and the Transformer outputs a sequence of tokens.

The outputs from both models are processed to ensure that no errors occur during the sheet music and MIDI file genera-

tion process, and that the music matches the parameters selected by the user. This involves transposing the piece to the selected key, removing bars that exceed the desired length, altering the durations and offsets of notes so that the sheet music appears neater.

For both models, the outputs are encoded numerically and then written as Music21 objects, which are then turned into MIDI files by the Music21 library.



To generate the PDF sheet music, the notes in the images are written as strings. The Abjad library passes these strings into the Lilypond music engraving software which generates the PDF sheet music file.

```
Score("{ { { { { r4 r4 e''4 f'4 } { s4 s4 s8 a'4 s8 } { d''4 s8 c''2 s8 } } { { d''16 g''16 r8 r8 a'4 r8 r4 } { f'2 s4 s8 d''8 } } { { r8 b'2 c''4 r8 } { s4 s4 s4 e''4 } }
 { { d''8 r8 r4 r4 r8 r16 a'16 } { a'2 s4 s8 b'8 } } { { f'4 r4 r8 a'8 d''4 } { s4 s4 s8 e''16 s16 s8 s16 a'16 } } { { d''2 r4 r8 r16 <a' b'>16 } } { { r16 a'4 r16 r8 r4 r4
 } { f'8 s8 s4 s4 s4 } } { { e''8 c''4 g'4 c''4 r8 } { s4 s8 b'8 s4 s4 } } { { f''4 r8 e''4 r8 r4 } { d''8 s8 <a' c''>2 s4 } } { { c''8 r8 f'8 r8 r8 d''4 r8 } { s4 b'2 s4
 } { g'2 f'2 } } } } { { { { d8 r8 r4 r4 r4 } } { { b,8 r8 r4 r4 r4 } } { { <g, b,>4 r4 r8 d4 r8 } } { { e8 r8 d'2 r4 } } { { b,16 r16 r8 r4 r4 r4 } } { { r8 e,16 r16 c8 r8
 r4 r4 } { s8 d8 s4 s4 s4 } } { { <b, d f>8 a,4 r8 r4 r4 } } { { r4 c'8 r8 r4 r4 } } { { e,4 r4 r4 r4 } } { r1 } } } }", name='Score', simultaneous=True)
```