Leveraging Autoencoders and Attention Mechanisms for Improved GAN Music Generation

Isotton Gloria†, Munafo' Sara†

***Abstract*—Melody generation in the symbolic domain using deep learning is a rapidly evolving field, with significant implications for music composition and AI-driven creativity. In this paper, we explore the use of Convolutional Neural Networks (CNNs) within a Generative Adversarial Network (GAN) framework to generate sequences of MIDI notes one bar at a time. We introduce self-attention in the GAN architecture, which allows attention-driven, long-range dependency modeling of the song bars, enabling the model to capture more complex relationships within the music. Our approach includes a conditional mechanism that allows the model to generate melodies from scratch, conditioning on the melody of previous bars. To evaluate the effectiveness of our approach, we conduct a user study comparing eight-bar melodies generated by our model and other melodies from training set. Additionally, we propose two novel metrics for melody quality evaluation, one focused on the note frequency distribution, the other, more general, leverages embeddings from a pre-trained autoencoder. Through this set of metrics we are able to evaluate the impact of self-attention layers in the training phase, showing an increase in similarity between real and generated data.**

***Index Terms*—Melody generation, Generative Adversarial Networks, Convolutional Neural Networks, Autoencoders, music evaluation metrics.**

I. Introduction

Algorithmic composition has been a topic of research since the late 1950s [1], with early efforts leveraging rule-based systems and traditional neural networks [2]. However, the advent of deep learning has revitalized this field, enabling the generation of complex and realistic music through models trained on large datasets ([3], [4], [5], [6] [7], [8]). In particular, Generative Adversarial Networks (GANs) have shown promise in capturing intricate patterns in music data, producing melodies that are not only coherent but also creatively diverse.

Despite these advancements, most existing models for music generation, such as those based on Recurrent Neural Networks (RNNs) ([9], [10], [11], [12]), focus on generating sequences by conditioning each note on the previous ones. While effective, these models often struggle with long-range dependencies and require significant computational resources. Moreover, few attempts have been made to explore the potential of Convolutional Neural Networks (CNNs) in this domain, particularly for symbolic music generation where the structure of music can be naturally represented as a two-dimensional matrix. In this paper, we investigate the novel approach to melody generation using a CNN-based architecture within a Self-Attention GAN framework [13], which we refer to as Attention-MidiNet. Unlike traditional sequence generation models, this method generates melodies one bar at a time, allowing for a more modular and flexible composition process. This approach not only facilitates the use of 2D convolutional operations but also enables the model to incorporate various forms of prior knowledge, such as priming melodies, through a conditional mechanism. Moreover, the Self-Attention layers allow the model to capture long-range dependencies and intricate relationships across the entire sequence.

Additionally, we use a feature matching technique [14] that allows us to control the influence of these conditions on the generated result. For example, this enables us to regulate how much the current bar should resemble the previous ones, fostering both coherence and diversity in the music. Our CNN-based model can be easily extended to handle tensors instead of matrices, allowing for the generation of multi-channel MIDI tracks and music with multiple parts. This flexibility makes Attention-MidiNet a highly adaptive and generic alternative to traditional RNN-based designs.

During training, we utilize the Pitch Class Histogram (PCH) metric to guide the GAN's learning process. The core concept of PCH is to analyze and compare the pitch distributions of both generated and real samples, with the goal of achieving distributions that are as similar as possible. However, the PCH metric conducts a point-by-point comparison, which does not capture the full semantic and stylistic complexity of music. To address this limitation, we employ autoencoders to derive a latent representation of each song. The autoencoder, trained on a dataset of real songs, encodes both real and generated music into a shared embedding space. This allows us to assess the similarity between the two by comparing their reconstruction loss rather than their raw, point-by-point data.

In addition to this metric, we conducted a user study involving non-professional listeners to assess the aesthetic quality of the generated music. The study compared melodies produced by two variants of the GAN model-one with attention mechanisms and one without. These melodies were evaluated against melodies extracted from the original training data to determine the perceived quality and authenticity of the generated music.

Our contributions can be summarized in three major points:

1) we introduced Self-Attention layers in the CNN-based

†Department of Physics and Astronomy, University of Padova, gloria.isotton@studenti.unipd.it, sara.munafo@studenti.unipd.it

Our sincere thanks to Li-Chia Yang, Szu-Yu Chou, and Yi-Hsuan Yang from the Research Center for IT Innovation at Academia Sinica, Taipei, for their groundbreaking work on MIDINET and contributions to symbolic-domain music generation.

GAN model;
2) we proposed a metric based on the pitch range (PCH) to assess the generated melodies' similarity to the real ones;
3) we proposed a novel metrics for evaluating the quality of generated melodies, which utilizes embeddings from a pre-trained autoencoder.

While the pitch range metric has its limitations, it provides valuable insight into the quality of melodies during the training phase, helping to monitor and refine the model's performance.

This paper is organized as follows: Section II reviews related work in neural network-based music generation. Section III outlines the processing pipeline, while Section IV discusses the datasets and their pre-processing. In Section V, we describe the learning model. Section VI presents the results and discussion, and finally, Section VII offers concluding remarks and potential directions for future work.

II. Related Work

In recent years, the field of music generation has seen significant advancements, largely driven by the application of deep learning techniques. Among these, MIDI melody generation has garnered substantial interest, with models like Recurrent Neural Networks (RNNs) and Generative Adversarial Networks (GANs) being employed to create melodies that exhibit both coherence and creativity. One of the pioneering models in MIDI melody generation is MelodyRNN [15], developed as part of Google's Magenta project. MelodyRNN employs a sequence-to-sequence learning framework with LSTM networks to generate coherent melodies based on an input sequence. This model laid the groundwork for subsequent RNN-based music generation approaches, demonstrating the potential of recurrent architectures in capturing the temporal dependencies in music sequences. MelodyRNN featured the implementation of three RNN-based models, two of them aiming to learn longer term structures in the melodies. Building on the MelodyRNN model, other more complex models based on RNNs were developed, deepening the exploration of music generation by including chords, such as the Song from PI model [16], a hierarchical RNN model that exploits a hierarchy of recurrent layers to generate multi-track pop songs; this model, however requires some prior knowledge about how pop music is composed, which is incorporated in the layers of the Network. The first implementation based on GANs is the C-RNN-GAN model [17], which employs the generative adversarial paradigm to sequentially generate bars, leading to the generation of classical songs. The MidiNet model [18], presented one year later, further delved into the use of GANs for music generation, introducing the use of a Conditioner CNN in the generation process, which allows the Generator to condition its generation on previous bars. In the MidiNet model, whose architecture was used as a reference for our paper, creativity of the generated songs is taken into account via the tuning of the model's hyperparameters, and different models are built based on the level of creativity and the use of chords. In the analysis of the MidiNet performance, however, only a qualitative metric is employed to determine the goodness and realism of the generation process, which is what drove us to investigate further in the direction of a more objective similarity metric, while building up on the model by adding the attention mechanism, to allow the Network to learn longer term features.

III. Processing Pipeline

A system diagram of Attention-MidiNet is shown in Figure 1. Below, we present the technical details of each major component of our model.

A. Symbolic Representation for Convolution

MIDI files containing melodies are used as the basis for our model's symbolic representation of music. Each MIDI file is structured to have a fixed length, organized into 8 bars, with each bar containing 16 time steps or notes. During preprocessing, we ensure that only the longest note in each time step is retained, simplifying the representation. Following the same path, we also omit the velocity (volume) of the note events.

To make these melodies readable and processable by our GAN architecture, each note is one-hot encoded in a space of 127 possible pitch values, with an additional dimension for silence. This one-hot encoding converts each note into a binary vector in $\mathbb{R}^{128}$, where only one element is "1" (indicating the presence of a specific note), and the rest are "0".

After encoding, each melody in the MIDI format is represented as a tensor with the shape $[8, 16, 128]$, where the first dimension (8) corresponds to the 8 bars of the song, and the second dimension (16) represents the 16 notes, or time steps, within each bar. The third dimension (128) includes the one-hot encoded 128-dimensional vectors representing each possible note or silence.

This representation is limited because it doesn't account for the duration of notes or the possibility of overlapping notes or chords. However, it allows us to test our algorithm at an initial level and assess its basic functionality.

B. Generator CNN and Discriminator CNN

The foundation of our model is a deep convolutional generative adversarial network. The objective of this architecture is dual: the discriminator D is trained to differentiate between real (authentic) and generated (synthetic) data, while the generator G is designed to produce artificial data that can deceive the discriminator.

The generative model G maps a latent vector $\boldsymbol{z} \in \mathbb{R}^l$, typically modeled as a simple Gaussian noise $p(\boldsymbol{z}) = N(\boldsymbol{z}|0, I)$, to the tensor $G(\boldsymbol{z}) \in \mathbb{R}^{1\times16\times128}$ using a deep CNN with learnable parameters. If the generator G is functioning effectively, the output tensor should be indistinguishable from real data $\boldsymbol{\mathcal{X}}$ to the discriminator D.

The discriminator D distinguishes between real and generated (synthetic) data, providing feedback to improve the generator. At each step, the discriminator takes as input a

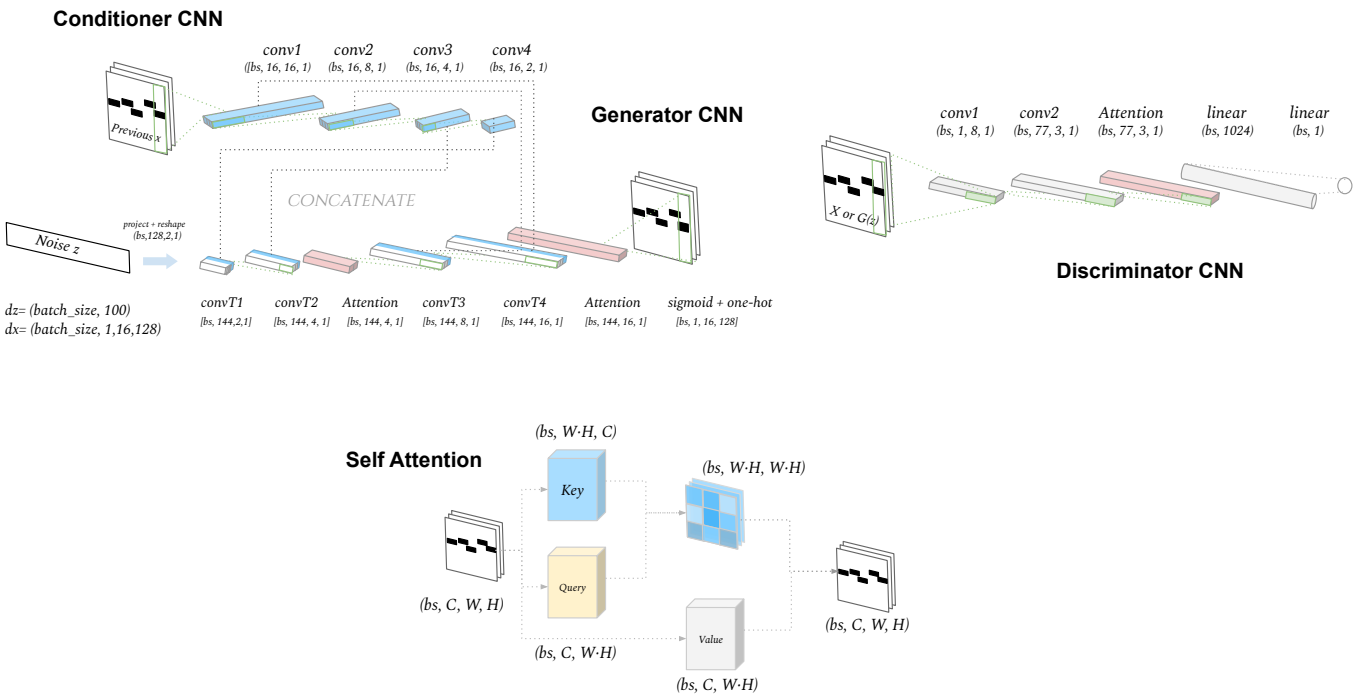

Fig. 1: System diagram of the proposed Attention-MidiNet model for symbolic-domain music generation. The top section illustrates the model architecture, including the generator, conditioner (colored in blue), and discriminator, with specified dimensions for each feature map. The bottom section depicts the general scheme of self-attention, applied to different layers (colored in red) of either the generator or the discriminator.

sample from the training set and a generated sample from the generator, classifying them by labeling real data as 1 and synthetic data as 0. The discriminator is trained to minimize classification errors, while the generator is trained to maximize them, making the problem a zero-sum game:

$$\min_G \max V(D,G) = \mathbb{E}_{\boldsymbol{\mathcal{X}} \sim p_{\text{data}}(\mathbf{X})}[\log(D(\boldsymbol{\mathcal{X}}))] + \mathbb{E}_{\mathbf{z} \sim p_z(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))] \quad (1)$$

where $\boldsymbol{\mathcal{X}} \sim p_{\text{data}}(\boldsymbol{X})$ denotes sampling from real data, and $\boldsymbol{z} \sim p_z(\boldsymbol{z})$ denotes sampling from a random distribution.

Training GANs often encounters challenges such as instability and mode collapse. To address these issues, various methods have been proposed. In our approach, we utilize feature matching and one-sided label smoothing [14]. Feature matching aims to minimize the difference between real and generated data distributions by incorporating additional L2 regularization terms into Eq. 1. Specifically, the following terms are added during the training of G:

$$\lambda_1 \|\mathbb{E}[\boldsymbol{\mathcal{X}}] - \mathbb{E}[G(\boldsymbol{z})]\|_2^2 + \lambda_2 \|\mathbb{E}[f(\boldsymbol{\mathcal{X}})] - \mathbb{E}[f(G(\boldsymbol{z}))]\|_2^2, \quad (2)$$

where f represents the first convolutional layer of the discriminator, and λ_1 and λ_2 are hyperparameters that are determined experimentally.

In one-sided label smoothing, real samples are assigned a label slightly less than 1 (in our case 0.9) instead of the standard label of 1. This technique reduces the risk of the discriminator becoming overly confident in its predictions, encouraging it to generalize more effectively. As a result, the generator is driven to produce data that more closely resembles the real distribution.

C. Conditioner CNN

In our work, we extend the generator within the GAN architecture to a conditional model, allowing for the integration of additional information during the generation process. Specifically, we condition the generation of the current bar on the melody of the previous bar, which is represented as an $h \times w$ matrix. This matrix is reshaped into smaller vectors with varying shapes to be incorporated into different intermediate layers of the generator G. To achieve this, we introduce a conditioner CNN that functions as the inverse of the generator CNN. The conditioner CNN processes the $h \times w$ conditional matrix using a series of convolutional layers. Notably, both the conditioner and generator CNNs utilize identical filter shapes in their convolutional layers, ensuring that the outputs from these layers are "compatible"

in shape. This design allows us to concatenate the output from a convolutional layer of the conditioner CNN with the input of the corresponding transposed convolutional layer of the generator CNN, effectively influencing the generation process. During training, the conditioner and generator CNNs are trained concurrently, sharing the same gradients.

D. Self-Attention

Convolutional layers, with their limited receptive field, require multiple layers to capture long-range dependencies, which can impede the learning of such dependencies in sequential data. To address this limitation, an attention mechanism was integrated into the intermediate layers of both the generator and discriminator networks, aiming to enhance the quality of the generated songs. This technique, which has proven effective in GAN-based image generation tasks [19], allows the self-attention module to complement the convolutional layers. By doing so, the model can effectively capture long-range, multi-level dependencies across different parts of the songs, thereby improving the overall generation process.

We incorporated two self-attention layers into the generator, specifically after the second and fourth convolutional layers, and one self-attention layer into the discriminator, placed after the second convolutional layer. Applying self-attention to these specific layers was chosen for its convenience, as the convolutional maps at these points are already in 4D, making them well-suited for the self-attention mechanism.

The input of the Self-Attention layer is a mini-batch of feature maps, indicated with $\overline{\boldsymbol{\mathcal{X}}}$, with dimensions (batch size$\times$ $c \times w \times h$). $\overline{\boldsymbol{\mathcal{X}}}$ is initially transformed into three separate feature spaces: $\boldsymbol{\mathcal{K}} \in \mathbb{R}^{bs \times w \cdot h \times c}$, $\boldsymbol{\mathcal{Q}} \in \mathbb{R}^{bs \times c \times w \cdot h}$, and $\boldsymbol{\mathcal{V}} \in \mathbb{R}^{bs \times c \times w \cdot h}$. These transformations are given by:

$$\begin{aligned} \boldsymbol{\mathcal{K}}(\overline{\boldsymbol{\mathcal{X}}}) &= W_k \overline{\boldsymbol{\mathcal{X}}}; \\ \boldsymbol{\mathcal{Q}}(\overline{\boldsymbol{\mathcal{X}}}) &= W_q \overline{\boldsymbol{\mathcal{X}}}; \\ \boldsymbol{\mathcal{V}}(\overline{\boldsymbol{\mathcal{X}}}) &= W_v \overline{\boldsymbol{\mathcal{X}}}, \end{aligned} \tag{3}$$

where W_k, W_q, and W_v are learned weight matrices. To compute the attention, we first calculate the attention scores by performing a dot product between the transformed feature spaces $\boldsymbol{\mathcal{K}}$ and $\boldsymbol{\mathcal{Q}}$ and then then we normalize the score using a softmax function:

$$\boldsymbol{\mathcal{A}}_{j,i} = \frac{\exp(\boldsymbol{\mathcal{S}}_{i,j})}{\sum_{i=1}^{N} \exp(\boldsymbol{\mathcal{S}}_{i,j})}, \quad \text{where } \boldsymbol{\mathcal{S}}_{i,j} = \boldsymbol{\mathcal{K}}(\boldsymbol{x}_i)^T \boldsymbol{\mathcal{Q}}(\boldsymbol{x}_j). \tag{4}$$

This produces an attention map $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{bs \times w \cdot h \times w \cdot h}$, which is used to weight the importance of each spatial location in the feature maps. This attention map is applied to the transformed feature space $\boldsymbol{\mathcal{V}}$ to produce the weighted sum of feature values, effectively aggregating information from different spatial locations according to their importance. The output of the attention layer is:

$$\overline{\boldsymbol{\mathcal{O}}}_j = \sigma \left(\sum_{i=1}^{w \cdot h} \boldsymbol{\mathcal{A}}_{i,j} \boldsymbol{\mathcal{V}}_i \right) + \overline{\boldsymbol{\mathcal{X}}}_j \tag{5}$$

E. Evaluation metric: reconstruction loss from a pre-trained autoencoder

To evaluate the quality of the generated melodies, we utilize a novel metric based on embeddings from a pre-trained autoencoder. This metric assesses how closely the generated melodies resemble human-composed music from dataset, providing a measure of the model's performance. This approach involves a neural network with an equal number of output units and input units, trained to produce an output $\boldsymbol{\mathcal{Y}}$ that closely matches the input $\boldsymbol{\mathcal{X}}$. During training, the network learns to encode the input $\boldsymbol{\mathcal{X}}$ into a latent compressed representation $\boldsymbol{z}(\boldsymbol{\mathcal{X}})$ via an encoder, and then decode this representation to reconstruct the input via a decoder. The latent space representation allows for direct comparison of the overall semantic and stylistic characteristics of two input songs, rather than a point-by-point analysis.

The encoder is a CNN with four convolutional layers followed by a linear layer, designed to produce a feature representation $\boldsymbol{z}$ of the input $\boldsymbol{\mathcal{X}}$ in the latent space $\mathbb{R}^{z_{size}}$. All layers utilize Leaky ReLU activations. The decoder mirrors the encoder architecture, consisting of four convolutional transpose layers that upsample the latent vector $\boldsymbol{z}$ back to the original size of the input tensor $\mathbb{R}^{c \times w \times h}$.

The training of the autoencoder is performed by minimizing the mean squared error loss on a training set that is also used for training the GAN model:

$$\begin{gathered} L(\theta, \phi) = \frac{1}{N} \sum_{i=1}^{N} \| \boldsymbol{\mathcal{X}}_i - D_\theta(E_\phi(\boldsymbol{\mathcal{X}}_i)) \|_2^2, \\ \text{where } E_\phi : \boldsymbol{\mathcal{X}} \mapsto \boldsymbol{z} \quad D_\theta : \boldsymbol{z} \mapsto \boldsymbol{\mathcal{Y}} \end{gathered} \tag{6}$$

Additionally, random noise is introduced during training, which involves randomly shifting the order of the song notes.

IV. Signals and Features

For this project, we gathered data from the Lakh MIDI dataset, a collection of $176,581$ heterogeneous MIDI files. Since our implementation focuses on the generation of melodies (i.e. one note is played at a time), we preprocessed the dataset as follows:

1) We selected only the MIDI files containing a 'MELODY' track, making sure that the track was in fact characterized by one note played at a time and removing corrupted files;
2) We chose a specific tempo $(4/4)$ and ticks per beat (384) to reduce the heterogeneity of the dataset and work with standardized data; for the same scope, we decided to consider only 8 bars per song, removing excess bars from longer songs;
3) We selected a timestep of 96 ticks per beat, so that each bar was composed by $w = 16$ notes; for each timestep, we associated a single note, corresponding to the longest played note in that timestep.
4) We formatted each note into a 128-dimensional one-hot encoded array, with all zeroes except for the active note.

At the end of this preprocessing phase, our dataset consisted of a $[2698, 8, 16, 128]$-shaped array ([songs $\times$ bars $\times$ notes $\times$ pitch range]). Since the dimension of the dataset was significantly smaller than the original dataset, we then proceeded to augment the dataset with the following strategies:

- Transposition: each song is transposed of a random number of semitones in the range $[-12, +12]$;
- Rotation: for each songs, notes are shifted clockwise of a random number of steps in the range $[8, 120]$.

After data augmentation, our dataset consisted of 43168 melodies, which we then split into 30217 melodies for training and 12951 for testing. From the training set, two datasets were obtained: the `train_data` dataset, and the `prev_data` datasets. The latter is obtained by adding an empty bar before the first bar of each song in the `train_data` (i.e. a bar where each note event is represented by all zeroes) and considering 8 bars starting from this empty one (effectively removing the last bar of each song). This necessity arises from the architecture itself, in particular for the need of the Conditioner to have as input the previous bar than the one given to the Discriminator.

V. Learning Framework

As introduced before, the core of the learning algorithm is the minmax game between discriminator and generator. For each training epoch, two updates of the generator and one of the discriminator are performed, to weaken the discriminator.

A. Training the GAN

The Conditioner CNN architecture is implemented in a way that it could be considered a *reverse* of the Generator CNN: while G starts from a lower-dimensional input ($[72, 100]$) and aims to obtain a higher-dimensional input $G(z)$ ($[72, 1, 16, 128]$), C starts from a higher-dimensional input ($[72, 1, 16, 128]$) and applies convolutions to reduce its dimension. Each random vector $\boldsymbol{z}$ first passes through two fully-connected layers, with 1024 neurons each. The output is then reshaped into a 2-by-1 matrix with 128 channels.

- **Generator** : the Generator consists of four transposed convolutional layers:
  - The first three transposed convolutional layers use filters of shape 2-by-1 and stride 2-by-2, with the number of output channels equal to the pitch range (128). These layers gradually upsample the input vector while reducing the number of channels.
  - The final transposed convolutional layer uses a filter of shape 1-by-128 and stride 1-by-2, producing a 1-channel output with the desired pitch range.
- **Conditioner** : In parallel to the Generator, the input tensor `prev_x` is processed by a series of four convolutional layers by the Conditioner CNN.
  - The first convolutional layer uses filters of shape 1-by-128 and stride 1-by-2, reducing the pitch dimension while increasing the number of channels to 16.
  - The next three convolutional layers use filters of shape 2-by-1 and stride 2-by-2, further reducing the spatial dimensions.

  At each stage, the output of the transposed convolutional layers is concatenated with the corresponding output from the convolutional layers of C. Additionally, if attention is enabled, two self-attention layers are applied to the concatenated tensors after the first and second transposed convolutional layers, enhancing the model's focus on global features.
  A sigmoid activation is applied to the output of the last transposed convolutional layer to generate the final output, which is then followed by a layer that turns off per time step all notes but the one with the highest activation, ensuring that the output $G(\boldsymbol{z})$ represents a monophonic melody; $G(\boldsymbol{z})$ will be given as input to the Discriminator, which will have to determine how likely it is to be a real melody.
- **Discriminator**: the input tensor (either real or generated) of the Discriminator CNN passes through two convolutional layers:
  - The first convolutional layer uses a filter of shape 2-by-128 with a stride of 2-by-2, reducing the pitch dimension and preserving a single channel.
  - The second convolutional layer uses a filter of shape 4-by-1 with a stride of 2-by-2, increasing the number of channels to 77.

  The output is batch-normalized and, if attention is enabled, passed through a self-attention mechanism. The output of the self-attention layer (or the convolutional layer if attention is not used) is then activated with a Leaky ReLU and flattened into a vector. This vector is processed by two fully-connected layers: the first one reduces the dimensionality to 1024, while the second reduces it further to a single output, representing the raw prediction score of whether the input is real or fake. Finally, a sigmoid activation function is applied to the output to obtain the probability score, indicating the likelihood that the input is real.

To address the issue of training instability, which is common when training GANs, a few regularization techniques were implemented during training. First of all, as introduced in V, we added two feature matching terms to the Generator loss, that enforce the closeness between real and generated data (see Eq. 2). The parameters λ_1, λ_2 need to be set empirically: the first one weighs the importance we give to the 'closeness' between the real bar $\boldsymbol{\mathcal{X}}$ and the generated bar $G(\boldsymbol{z})$, while the latter enforces closeness between their respective feature maps in the first layer of the Discriminator. For both Generator and Discriminator, Adam optimizer was employed, with parameters $\{\beta_1, \beta_2\} = \{0.5, 0.999\}$, however we decided to opt for an adaptive learning rate to further weaken the Discriminator: we started with a higher learning rate for the Generator, and then we halved both learning rates each 10 training epochs. In addition, we employed one-sided

label smoothing for the Discriminator, setting the 'True' label to 0.9, and substituted the generator loss function as follows:

$$\log(1 - D(G(\boldsymbol{z}))) \longrightarrow -\log(D(G(\boldsymbol{z}))) \tag{7}$$

which holds the same meaning (we minimize the negative log-probability of D being wrong instead of maximizing the probability of D being wrong), but avoids the issue of vanishing gradients for the Generator.

B. The testing stage

For testing, our goal is to observe the performance of the trained generator on the inference task, and the process is done one bar at a time.

- For each sample in the `test_data` dataset, only the first bar is given as input to the trained Conditioner.
- The Generator, starting as usual from a random noise vector $\boldsymbol{z}$ (this time of shape $[1, 100]$) will generate a bar conditioned on the real one.
- From the third bar on, the Conditioner will use as previous bar the one generated from G at the previous iteration, and the generation process goes on iteratively until eight bars are reached.

In this way, the model has to rely on its own outputs to condition subsequent generations. This simulates the actual use-case where only the first bar might be given by a user or another system, and the model must autonomously generate the rest. This process tests the model's ability to maintain coherence over longer sequences, even when its own outputs serve as inputs. It's a more challenging scenario and closer to real-world applications, where perfect inputs (i.e., real bars) aren't continuously available.

C. Monitoring the training process

Because of the adversarial nature of the learning process, only monitoring the loss values during training can be misleading. For example, if D is very strong, G's loss might be high even if it is improving. Conversely, if D is weak, G's loss might be low even though it is not generating realistic data. Furthermore, D and G's losses are not efficient at highlighting training problems such as mode collapse or vanishing gradients. For this reason, we decided to implement an additional metric that would allow us to have more insight on how the training process is actually evolving, the Pitch Class Histogram (or PCH) [20]. The PCH main idea is to analyze the distribution of pitches both of the generated and real samples, and compare the two distributions, which ideally we would want to be as close as possible; for each sample, the frequency of each pitch (12 in total) is computed, and then normalized, obtaining an histogram representing the distribution of pitches over the sample song. To measure the *closeness* between the two distribution, a numeric evaluation of the Kullback-Leibler divergence:

$$D_{\mathrm{KL}}(\mathrm{PCH}_{\mathrm{real}} \parallel \mathrm{PCH}_{\mathrm{generated}}) = \\ = \sum_i \mathrm{PCH}_{\mathrm{real}}(i) \log\left(\frac{\mathrm{PCH}_{\mathrm{real}}(i)}{\mathrm{PCH}_{\mathrm{generated}}(i)}\right) \tag{8}$$

is performed. It is worth mentioning that, while adding insight on the similarity of our generated data, this metric only focuses on the pitch distribution, which is one of the many aspects contributing to such similarity. Our usage of the PCH is, therefore, not aimed at obtaining an absolute and objective *similarity score*, but rather at deepening our analysis of the model's performance. Due to the different nature in the procedures of training and testing, simply applying this metric would have not given us insightful information to compare the two. Instead, we adapted the metric's application in a way that we deemed more meaningful:

- Training stage: during training, the frequency of each pitch is computed for both real and generated samples, and averaged out over the minibatch; the minibatch average is then used to compute the K-L divergence at each epoch.
- Testing stage: during testing, we split each (real and generated) song in half, and computed the pitch frequencies (and K-L divergences) separately for the first and second half.

With this adaptation on the testing stage, our goal is to investigate on the possible degeneration of songs: as the generated new bar strays further away from the original real bar, it might become more and more complex for the model to maintain similarity with real bars without degenerating onto the same few notes.

D. Pre-trained AutoEncoder for further analysis

Finally, we exploited embedding through a pre-trained AutoEncoder to further compare the generated melodies with the real ones. The AutoEncoder was trained on the same dataset on which the GAN was trained on, `train_data`. The model we implemented is composed as follows:

1) **Encoder**:
   - The input is passed through a convolutional layer with a filter of size 1-by-128 and a stride of 1-by-2; the output of this layer has 16 channels.
   - Three more convolutional layers follow, each with a filter size of 2-by-12-by-1 and a stride of 2-by-2. Each layer maintains 16 channels while gradually reducing the spatial dimensions of the feature maps. After each convolutional layer, Leaky ReLU with a negative slope of 0.2 is applied.
   - After the last convolutional layer, the output is flattened into a 2D vector of size $[72, 32]$ and passed through a fully connected (linear) layer that projects it into a lower-dimensional latent space $[72, \texttt{z\_size}]$; this latent vector `z_latent` is the compressed representation of the input data, and we set $\texttt{z\_size} = 16$.
2) **Decoder**:
   - The first three layers mirror the encoder's convolutional layers, using 2-by-12-by-1 filters with a stride of 2-by-2. After each convolutional layer, Leaky ReLU with a negative slope of 0.2 is applied.

- The final layer uses a 1-by-128 filter with a stride of 1-by-2, which restores the input dimensions. The output channel is set to 1, matching the input data's original channel number.
- A sigmoid activation function is applied at the final layer to ensure that the output values are in the range $[0, 1]$.

The AutoEncoder was pre-trained on 30 epochs and a learning rate of 0.0005.

During testing, we applied the pre-trained AutoEncoder to two separate datasets: the test set (real songs) and the generated set that matches the test set in dimension. Reconstruction loss was then calculated separately for both the real and generated datasets by applying mean squared error (MSE) across each. By comparing the reconstruction losses, we could evaluate the similarity between the two sets. If the generated songs closely resemble the real ones, the AutoEncoder should easily reconstruct them, indicating minimal divergence between the distributions of real and generated data. However, if the generated songs lack realism, the AutoEncoder, encountering unfamiliar inputs, is expected to produce a higher reconstruction error. Thus, a significantly higher reconstruction loss for the generated set, compared to the real one, could signal that the generated songs are less realistic.

VI. Results

We explored various strategies for optimizing the music generation process using our GAN model. Initially, we experimented with restricting the generated notes to two octaves, as suggested by prior research [18]. However, after adapting the code accordingly and conducting several tests, we observed no significant differences in the generated music, leading us to proceed with the full range of octaves.

We then focused on tuning key parameters of our model, adjusting the initial values of the adaptive learning rates β_1 and β_2, regularization parameters λ_1 and λ_2 and the probability of song inversion during training. Both qualitative assessments and quantitative metrics, including PCH and reconstruction loss, were used to evaluate the outcomes. Despite the reconstruction error remaining largely consistent across all experiments, the optimal parameters were identified by qualitatively evaluating the sampled songs, in addition to these metrics. The identified optimal parameters are shown in table 1.

With these parameters established, we compared the model's performance with and without self-attention mechanisms. The K-L divergence between pitch class histograms computed over the songs generated during training displays a better alignment between real and generated songs in the presence of self-attention, with a value of 0.1152, against the 0.3288 obtained without self attention. However, we observed in both cases that there were no significant improvements in the PCH metric as the number of epochs increased (Figure 2). This lack of improvement suggests that the model may not be specifically targeting pitch frequency as a key factor in the learning process. Instead, it might be focusing on other musical elements that contribute more significantly to the overall similarity between generated and real songs, potentially sacrificing some degree of accuracy in pitch distribution to improve other qualities. However, the absence of an increase in KL divergence during training could also be interpreted as a promising sign, as it suggests that the model does not suffer from mode collapse -a common issue in GAN training where the generator produces a limited variety of outputs, often resulting in highly repetitive or identical samples. During testing we observed a different behavior: the KL divergence between the generated and target distributions tended to increase towards the latter half of the generated songs, reflecting the qualitative degeneration of the music, as shown in Figure 3.

Parameter	Value
β_1	0.00001
β_2	0.0005
λ_1	0.8
λ_2	10
Training Epochs	50
p_{invert}	0.3

TABLE 1: Optimal Parameters for Training Attention-MidiNet

	Train	Test (first half)	Test (second half)
Off	0.3288	0.7619	11.6468
On	0.1152	0.4799	6.8600

TABLE 2: K-L divergence between PCHs during training and testing, without ('OFF' row) and with ('ON'row) self-attention.

Initially, when our model lacked robust regularization techniques and methods to weaken the discriminator, the degeneration of the generated music was more pronounced, often collapsing into a single repeated note. After incorporating these improvements, degeneration was reduced to sequences of two or three repeating notes. While this degeneration occurs in both scenarios (with and without self attention), it is less severe when self-attention is applied and the KL divergence values during testing more closely align with those observed during training (see Table 2). Despite these enhancements, the persistent degradation highlights the need for further refinement in the model, particularly in sustaining musicality throughout the entire length of a generated piece.

In our evaluation, we applied a metric based on autoencoders to analyze the generated songs with self-attention mechanisms. We used a pretrained autoencoder to transform both the training and generated songs into a latent space. To facilitate comparison, we applied Principal Component Analysis and represented these embeddings in a 3D space for visual inspection, aiming to assess whether it was plausible

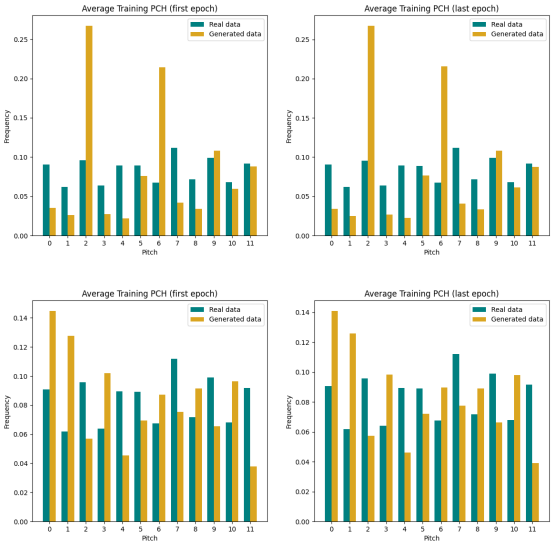

Fig. 2: **PCHs for first and last training epochs**. This figure illustrates the average over all minibatches of the PCH values for the generated songs during training in the first and last epoch, both without self-attention (top) and with self-attention(bottom). In both cases, the difference in pitch values between generated and real songs does not exhibit significant change throughout the training epochs, indicating that the model may not be prioritizing pitch frequency as a key factor in the learning process.

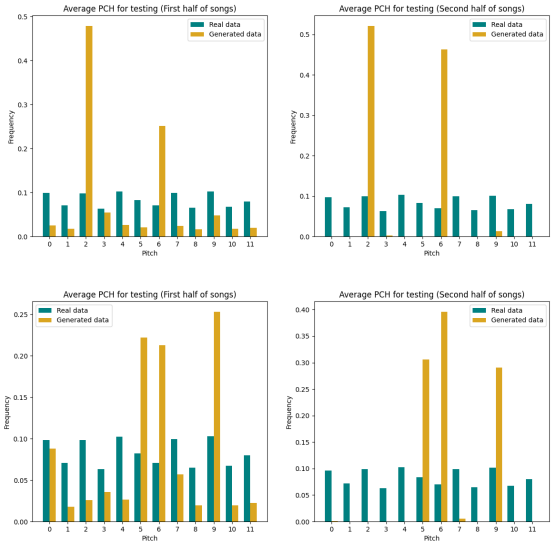

Fig. 3: **PCHs for test generated songs**. This figure illustrates PCH values for the generated songs during testing, split into halves, both without self-attention (top) and with self-attention(bottom). In both cases, a degeneration into only a few specific pitches is observed in the second half of the songs, even if the effect is slightly reduced in the presence of self-attention layers.

that both sets of embeddings share the same probability density function.

Initially, we considered clustering the songs within this latent space to compare outcomes. However, this approach proved challenging as predicting the number of clusters was difficult due to the lack of control over the style of the training data, leading us to abandon the clustering attempt.

To quantify the similarity between the two classes of embeddings, generated songs with attention and real (test) songs, we computed the reconstruction loss. This numerical comparison allowed us to evaluate the quality of the generated songs in relation to the real ones, providing insight into the effectiveness of the attention mechanism in preserving musical characteristics during generation; in both models, the reconstruction loss associated to generated data remained close to the value associated to real data, as shown in Table 3.

	Real	**Generated**
Off	0.007442	0.007774
On	0.007442	0.007641

TABLE 3: Recontstruction loss of real and generated songs, without ('OFF' row) and with ('ON'row) self-attention.

The visual representation of the real and generated embeddings is shown in the accompanying image (Figure 4). Although this metric shows promise, it did not yield the clear results we had hoped for in this case. It would be interesting to refine the model's training, perhaps with a more varied and larger initial dataset, and reapply this metric. Nonetheless, the idea of visualizing embeddings is valuable for understanding the data and remains a good approach despite the results.

Finally, we conducted a user study to assess the aesthetic quality of the generated music, involving a panel of ten non-professional listeners. This blind qualitative evaluation compared melodies generated by two variants of a GAN model -one with attention mechanisms and one without- against melodies extracted from the original training data. Participants rated the pleasantness, interest, and realism of a mix of real songs, songs generated with self-attention, and songs generated without self-attention, using a scale from 1 to 10. The results of these evaluations are summarized in the accompanying Figure 5.

VII. Concluding Remarks

During the development of our model, we enhanced the MidiNet architecture proposed by Yang et al. [18] by integrating Self-Attention layers and implementing additional regularization techniques, such as adaptive learning rates and one-sided label smoothing. Furthermore, we addressed a significant challenge in working with GAN architectures, the lack of reliable quantitative measures of model quality, by incorporating two quantitative metrics: Pitch Class Histogram (PCH) and Autoencoder embedding.

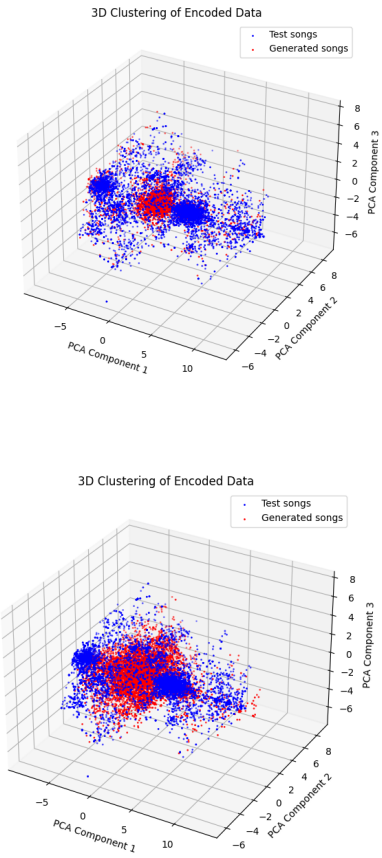

Fig. 4: **3D visualization of latent space embeddings.** 3D visualization of latent space embeddings for real songs from test set and generated songs without (top) and with (bottom) attention mechanisms. The embeddings, reduced via PCA, are compared to assess the similarity in distribution between the two sets. The distribution of the test set embeddings is quite peculiar, with patterns of points forming lines or distributed on the surface of a cone, likely due to the data augmentation process. The embeddings of the generated songs blend well with the real ones.

Through this implementation, we deepened our understanding of GANs architectures, and of the possible ways in which generated data can be quantitatively compared to real data in a way that aligns with human objective evaluation. The main challenges we faced regarded:

- the pre-processing phase, in which we needed to devise a way to extrapolate melodies from a polyphonic dataset, and this required deep exploration of the dataset;
- the overpowering of the Discriminator, which in the preliminary phases of our tests was much more dominant, and pushed us to explore a variety of different techniques to improve the Generator's performance.

The inclusion of the Self-Attention mechanism consistently improved performance across all our tests, although the extent of this improvement was less than anticipated. This discrepancy might be attributed to the significant reduction of the original dataset during the pre-processing phase, which, when combined with extensive data augmentation, could have overly limited the diversity of our training data. Looking forward, we believe that using a larger and more diverse dataset could further amplify the positive impact of Self-Attention on the generated melodies.

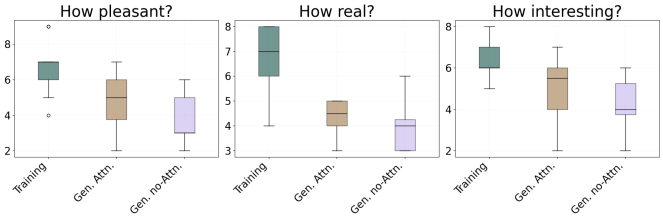

Fig. 5: **Results of the human evaluation.** This plot shows the comparison of human evaluations for the questions "How pleasant?", "How real?", and "How interesting?" across three conditions: songs sampled from training set, songs generated with attention mechanism and songs generated without attention mechanism. The boxplots display the distribution of responses for each condition. The boxplots reveal that the generated songs generally perform worse compared to the true ones. However, the model without Attention mechanism shows significantly poorer performance, indicating that this mechanism plays an important role in the quality of the generated songs.

Author contribution statement The authors confirm contribution to the paper as follows. Both authors equally contributed to the code development and the training tasks for this project. To optimize time, certain tasks were divided between the authors. Specifically:

- **Gloria Isotton** was responsible for designing and implementing the Autoencoder model;
- **Sara Munafo** was responsible for the developement and implementation of the Pitch Class Histogram algorithm.

Both authors actively collaborated on the review and integration of the various components.

REFERENCES

[1] L. Hiller and L. M. Isaacson, *Experimental Music: Composition with an Electronic Computer*. New York: McGraw-Hill, 1959.

[2] P. M. Todd, "A connectionist approach to algorithmic composition," *Computer Music Journal*, vol. 13, no. 4, pp. 27–43, 1989.

[3] M. Bretan, G. Weinberg, and L. Heck, "A unit selection methodology for music generation using deep neural networks," *arXiv preprint arXiv:1612.03789*, 2016.

[4] H. Chu, R. Urtasun, and S. Fidler, "Song from pi: A musically plausible network for pop music generation," *arXiv preprint arXiv:1611.03477*, 2016.

[5] J. Engel, C. Resnick, A. Roberts, S. Dieleman, D. Eck, K. Simonyan, and M. Norouzi, "Neural audio synthesis of musical notes with wavenet autoencoders," *arXiv preprint arXiv:1704.01279*, 2017.

[6] O. Mogren, "C-rnn-gan: Continuous recurrent neural networks with adversarial training," *arXiv preprint arXiv:1611.09904*, 2016.

[7] S. Mehri, K. Kumar, I. Gulrajani, R. Kumar, S. Jain, J. Sotelo, A. C. Courville, and Y. Bengio, "Samplernn: An unconditional end-to-end neural audio generation model," *arXiv preprint arXiv:1612.07837*, 2016.

[8] N. Jaques, S. Gu, R. E. Turner, and D. Eck, "Tuning recurrent neural networks with reinforcement learning," *arXiv preprint arXiv:1611.02796*, 2016.

[9] D. Eck and J. Schmidhuber, "Finding temporal structure in music: Blues improvisation with lstm recurrent networks," in *Proceedings of the IEEE Workshop on Neural Networks for Signal Processing*, pp. 747–756, IEEE, 2002.

[10] A. Graves, "Generating sequences with recurrent neural networks," *arXiv preprint arXiv:1308.0850*, 2013.

[11] J. J. Bharucha and P. M. Todd, "Modeling the perception of tonal structure with neural nets," *Computer Music Journal*, vol. 13, no. 4, pp. 44–53, 1989.

[12] N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent, "Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription," *arXiv preprint arXiv:1206.6392*, 2012.

[13] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena, "Self-attention generative adversarial networks," 2019.

[14] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," 2016.

[15] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *arXiv preprint arXiv:1409.3215*, vol. 27, 2014.

[16] S. F. Hang Chu, Raquel Urtasun, "Song from pi: A musically plausible network for pop music generation," *arXiv preprint arXiv:1611.03477*, 2016.

[17] O. mogren, "C-rnn-gan: Continuous recurrent neural networks with adversarial training," *arXiv preprint arXiv:1611.09904*, 2016.

[18] Y.-H. Y. Li-Chia Yang, Szu-Yu Chou, "Midinet: A convolutional generative adversarial network for symbolic-domain music generation," *arXiv preprint arXiv:1703.10847*, 2017.

[19] H. Tang, H. Liu, D. Xu, P. H. S. Torr, and N. Sebe, "Attentiongan: Unpaired image-to-image translation using attention-guided generative adversarial networks," 2021.

[20] A. E. George Tzanetakis and P. Cook, "Pitch histograms in audio and symbolic music information retrieval," *Journal of New Music Research*, vol. 32, no. 2, pp. 143–152, 2003.