CompoNet: Toward Incorporating Human Perspective in Automatic Music Generation Using Deep Learning

SeyyedPooya HekmatiAthar
North Carolina Agricultural and Technical State University
Mohd Anwar (manwar@ncat.edu)
North Carolina Agricultural and Technical State University

Research Article

Keywords: CompoNet, Human, Deep Learning

Posted Date: September 7th, 2021

DOI: https://doi.org/10.21203/rs.3.rs-812168/v1

CompoNet: Toward Incorporating Human Perspective in Automatic Music Generation using Deep Learning

SeyyedPooya HekmatiAthar[1] **and Mohd Anwar**[1,*]

[1]North Carolina A&T State Univeristy, Department of Computer Science, Greensboro, 27411, United States
[*]Corresponding Author: Mohd Anwar (manwar@ncat.edu)

ABSTRACT

The art nature of music makes it difficult, if not impossible, to extract solid rules from composed pieces and express them mathematically. This has led to the lack of utilization of music expert knowledge in the AI literature for automation of music composition. In this study, we employ *intervals*, which are the building blocks of music, to represent musical data closer to human composers' perspectives. Based on intervals, we developed and trained OrchNet which translates musical data into and from numerical vector representation. Another model called CompoNet is developed and trained to generate music. Using *intervals* and a novel monitor-and-inject mechanism, we address the two main limitations of the literature: lack of orchestration and lack of long-term memory. The music generated by CompoNet is evaluated by Turing Test: whether human judges can tell the difference between the music pieces composed by humans versus. generated by our system. The Turing test results were compared using Mann-Whitney U Test, and there was no statistically significant difference between human-composed music versus what our system has generated.

Automatic Music Generation (AMG) started in 1957 by Ljaren Hiller and Leonard Isaacson. They employed Markov chains and rule-based logic and developed software capable of composing a string quartet[1]. In early 1960s, Iannis Xenakis generated numerical patterns using computers, which he later transcribed into sheet music[2]. After a period of silence in the literature which was probably due to AI winter, in early 2000s, Blackwell created an improvisational musical system by applying multi-swarms[3]. Concurrently, Eck et al. employed Deep learning for the first time in AMG. They used two Long Short-Term Memory (LSTM) models to create music[4]. Schulze and Merwe proposed a model which can handle two-instruments harmony. This model, which was based on Hidden Markov Model generated pieces for piano with melodies on the right hand and simple chord progressions on the left hand[5]. Kosta et al. achieved chord sequence generation based on feeding observed examples into an unsupervised multi-stage framework[6]. Colombo et al. composed melody using Recurrent Neural Network (RNN) architectures[7]. Similar to Colombo's work, Sturm et al. employed RNN to composite music in digital format[8]. Jaques et al. combined RNN and reinforcement learning to develop music generation models[9]. Dieleman et al. proposed multiple deep architectures for generating music in raw audio format[10,11]. Huang et al. applied deep sequential generative models with self-attention to generate structured compositions in synthesizing keyboard compositions[12]. Hadjeres et al. handled monophonic AMG based on Variational Auto-Encoder (VAE)[13]. Inspired by the exciting progress in image and video related tasks, some researchers have employed Generative Adversarial Networks (GAN)[14] to extract temporal features and applied them to AMG. Yang et al. proposed midinet which is a convolution GAN[15] with specified conditions in 1D and 2D during the generation process. Midinet effectively solves the problem of lack of context association in AMG. Considering the temporal structure of music, C-RNN-GAN and SeqGAN combined RNN and GAN to generate music[16,17]. SeqGan combined reinforcement learning with GAN for music generation[17,18]. SeqGAN tried to extract temporal features by RNN; however, SeqGAN is not suitable for multi-instrument symbolic AMG in pianoroll format because pianoroll for multi-instruments symbolic music is represented as high-dimensional data, and SeqGAN is difficult to train with high-dimensional data. Dong et al. proposed MusGan which is a multi-track symbolic AMG model based on a hybrid GAN architecture[19,20]. Table 1 summarizes the most relevant works in the literature.

The Gaps in the Existing Literature

The existing literature leaves two major challenges/limitations unanswered: lack of orchestration and lack of long-term memory.

Table 1. Summary of most relevant works in the literature

Reference	Model Name	Dataset	Feature Engineering
Guan *et al.*[21]	DMB-GAN	Lakh MIDI Dataset[22]	Convolution - Self Attention
Hadjeres *et al.*[13]	GLSR-VAE	music21?	RNN
Eck & Schmidhuber[4]	-	[23]	LSTM
Strum *et al.*[8]	char-rnn	[24]	LSTM
Strum *et al.*[8]	folk-rnn	[24]	LSTM
Jaques *et al.*[9]	Note RNN	Not given	RNN
Yu *et al.*[17]	SeqGAN	Nottingham Dataset[25]	Convolution
Yang *et al.*[15]	MidiNet	TheoryTab[26]	Convolution
Mogren[16]	C-RNN-GAN	Not Given	RNN
Dong *et al.*[19,20]	MuseGAN	Lakh MIDI Dataset[22]	Convolution

Lack of Orchestration

Numbers in music theory should be interpreted differently than in the algebra. This leads to extra challenges when it comes to musical data representation. In other words, since numerical patterns in music follow different rules than patterns in other domains, existing pattern recognition techniques fail to correctly and efficiently extract the patterns in a given piece of music. Since common practices in pattern recognition need to put extra effort to compensate for music-specific differences, the number of instruments and ensembles needs to be lowered.

Lack of Long-Term Memory

Existing methodologies can be categorized into generative and memory-based models. While memory-based models, unlike generative models, do have memory capabilities, they inherit LSTM's most famous limitation: lack of long term memory. It has been observed that in sequence generation based on LSTM, an object, name, or idea gets introduced in the beginning, but it vanishes as the sequence gets longer.

Methods

In this paper, we aim to address the aforementioned challenges/limitations by employing the embedding mechanism based on music theory[27]. The embedded data is then given to a Stacked AutoEncoder called Orchestration Network (OrchNet). OrchNet specializes in translating musical data to and from vector representation. In the latent space of OrchNet, Composer Network (CompoNet) handles generation while ensuring long-term dependencies are maintained using a novel monitor-and-inject mechanism.

Pre-Processing Data

Dataset

Classical pieces are chosen as a use case for this paper. A MIDI dataset of 3,920 pieces from 144 classical composers is available on https://www.kaggle.com/blanderbuss/midi-classic-music/[28]. There is no documentation on how this dataset is gathered; therefore, it needs to be validated and cleaned.

Data Validation

The first step was to check file formats. There were some compressed files (.zip) in the dataset. The content of these files are extracted and checked. Then, a number of files (n=100) were randomly selected and verified by a human music analyst. This was done by comparing the content of the files with the available recordings of the same piece. In addition to these files, some famous pieces were selected and verified. The list of these pieces is provided in Table 2.

Data Cleaning

Data cleaning was done using Pypianoroll[29] and pretty_midi[30] packages. The following criteria were followed for excluding a file:

1. if either Pypianoroll or pretty_midi cannot handle the file and throw an error,
2. if the file has more than one time signature change event,
3. if the file contains less than 10 bars (the number of bars was calculated as: $active\ length / beat\ resolution / time\ signature\ numerator$),

Table 2. List of verified pieces

Composer	Piece Name
J.S. Bach	Ave Maria
	Bwv 565 Toccata and Fugue In Dm
	Bwv 772 Invention n01
	Bwv 1041 Violin Concerto n1 3mov
	Unfinished Fugue
	Bwv 1053 Harpsichord Concerto n2 1 mov
Beethoven	Fur Elise
	String Quartet no 1 op 18 1 mov
	Egmont Overture
	Sonata No.14 Op 27 Moonlight Sonata
	Symphony No. 9
	Two rondos for piano
Brahms	Waltz No. 1
	Hungarian Dance No.5
Chopin	Etude No. 1
	Nocturne Opus 62 No.1
Fauré	Spleen
	Romance sans paroles
Haydn	Symphony No. 94 “Surprise”
	Symphony No. 96 “Miracle”
Pachelbel	werde munter mein gemüte
	Canon Fantasy
Liszt	Hungarian Rhapsody No. 2
	Dante Sonata
Mozart	A piece for Piano, K. 176
	Piano Concerto No. 18, K. 456, “Paradis”
	Symphony No. 25 K. 183
	Piano Concerto No. 26, K. 537 “Coronation”
	Symphony No. 40 K. 550
	Sonata for Piano Duet in C major, K.19d
Paganini	Witches Dance
	Moto Perpetuo
Rachmaninoff	Piano Concerto No.2 Op.18
	Prelude in A minor Op. 32 No. 8
R. Korsakov	Flight Of the Bumblebee
	Aria of Sadko
Schubert	Erlkönig . D328 Op. 1
	Symphony D. 944 No. 9 “Great”
J. Strauss	The Blue Danube Op.314
Wagner	Ride of the Valkyries
Tchaikovsky	Waltz of the Flowers
	Swan Lake
Vivaldi	The Four Seasons
	Violin Concerto Op. 9 No. 2 RV. 345

4. if the numerator of the first time signature event is 1.

As a result, 653 files were excluded due to the first criterion while the other criteria reduced the number of files from 3,267 to 2,416.

Data Embedding

The Music Embedding package[27] was used for data embedding. This package accepts inputs in the pianoroll format; therefore, Pypianoroll package is used to convert MIDI files into pianorolls and then input them to the embedder of the Music Embedding package. The embedder converts pianoroll into sequences of intervals. In music theory, an interval is defined as the difference in pitch between two notes. The results of this embedding is samples with five dimensions:

- Order: the ordinal number in the interval,
- Type: the type or mode of the interval,
- Direction: is interval ascending or descending,
- Octave: indicates the octave difference of the two notes of the interval,
- RLE: the value of RLE encoding, i.e. note duration.

OrchNet

Input and Output

The proposed OrchNet has music theory built into it and translates music data to and from vector representation. OrchNet, which is a Stacked AutoEncoder, receives data embedded with Music Embedding package. *Order, type,* and *direction* are categorical while *octave* and *RLE* are numerical; therefore, it is reasonable to have one-hot encoding for *Order, type,* and *direction* and integer encoding for *octave* and *RLE*. However, since OrchNet is an AutoEncoder, its input and output must be identical; and having different encoding schemes in the output makes the model extremely difficult to train as each encoding scheme requires a specific loss function and combining different loss functions is not always possible. For this reason, *octave* and *RLE* are also encoded with one-hot encoding. Hence, each time step of the data is encoded as:

- Order: a one-hot vector of 7,
- Type: a one-hot vector of 5,
- Direction: a one-hot vector of 2,
- Octave: a one-hot vector of 4,
- RLE: a one-hot vector of 72.

This encoding, which requires 90 bits, allows encoding all seven ordinal possibilities of an interval, all five type possibilities of an interval, both ascending and descending directions, and 72 different possibilities for note duration; while allowing to have identical loss functions, hence an easier and more stable training routine.

The above-mentioned 90-bit encoding represents a single interval. Yet, music is the sequence of intervals. Per music theory, it is reasonable to break a piece into bars. However, each bar can have a different time signature and different number of notes and intervals. To mitigate this variable-length issue, padding and truncating mechanisms can be used. Therefore, each data sample for OrchNet is a 64 by 90 tensor; if a data sample has more than 64 time steps, it is truncated to 64; while the vast majority of data samples, which have less than 64 time steps, are padded with zeros to make them 64 by 90. The decision for having 64 time steps in each data sample is made because (1) the chosen number needs to be a power of two due to (GPU) hardware considerations, (2) 32 and smaller numbers are too aggressive and they truncate lots of data, while 128 is too much and pads lots of zeros which in turn increases sparsity in the data. 64 is the sweet spot, it truncates only 2% of samples while keeps the number of padded zeros manageable.

Model Architecture

Since the data has temporal dependencies, OrchNet is built based on LSTM blocks. As explained earlier, the input is a 64 by 90 tensor in which the sequence is padded with zeros to make its length consistent with other sequences. Therefore, OrchNet starts with a masking layer which return a mask of *True* and *False* values which indicates which time steps are real data and which of them are the padded zeros. An LSTM layer consisting 64 units follows the masking layer. This layer has a dropout value of 0.2 to prevent over-fitting. Finally, a fully connected layer with 128 neurons is added to generate a vector of 128 as the latent space representation of the input.

The decoder starts with *repeat vector* operation to convert the 1D latent space into 2D which allows the following 64-unit LSTM layer reproduce the input. A fully connected layer with 90 neurons prepares the data to flow into the five final fully connected branches, each of which representing a dimension in the data. All branches have categorical cross-entropy loss function.

A music piece can be embedded in three different ways: *melodic*, *harmonic*, and *with-respect-to-bar*. After applying data validation and cleaning procedures, *melodic*, *harmonic*, and *with-respect-to-bar* embeddings were applied on the dataset. Then, the aforementioned model, which is illustrated in Fig. 1, was trained as the building block of OrchNet, irrespective of embedding type, i.e., all three embeddings were used equally as the training dataset. Then, three copies of the trained model were obtained and each of them were re-trained separately on a specific type of embedding.

The trained models can translate a single track (instrument) into vector representation. However, this is not enough for an orchestral piece. To address this, another AutoEncoder was trained to combine the vector representations of different tracks. As shown in Fig. 2, this fully connected model accepts up to 16 tracks. The number of tracks in the model is arbitrarily decided to be 16 as this number (1) satisfies hardware acceleration constraints, (2) does not lead to hard-to-handle sparsity in the data, and (3) covers 95% of the dataset without losing any data.

The building block of OrchNet has a latent space of 128. Combining the latent spaces of 16 copies of this model, each of which having three different embedding types, results in a vector of $3 \times 16 \times 128 = 6144$. Passing through two fully connected layers with 2048 and 1024 neurons respectively, OrchNet encodes each bar of an up-to-16 tracks piece of music into a vector of 1024.

CompoNet

LSTM

CompoNet works in the latent space of OrchNet and generates sequences which can be decoded with OrchNet and presented as symbolic music. The core component of this sequence generation is an LSTM model as shown in Fig. 4. The model starts with two layers of LSTM units arranged in the opposition direction of each other (bidirectional), each consisting of 128 units. The output of these two layers is then given to a fully connected layer of 1024 neurons to generate the output. A dropout of 0.2 is used as a regularization mechanism.

This model works based on a sliding window mechanism which is illustrated in Fig. 3. In this mechanism, the model receives 16 samples as the input and generates the next one. The generated sample is added to the sequence and the window moves one step forward to include the newly generated sample and exclude the oldest sample. The limitations of this solution are:

- it is not self-sufficient, the first 16 samples are needed from external sources,
- it suffers from the long-term memory problem which was mentioned earlier.

GAN

The model shown in Fig. 4 requires the first 16 samples to be given from another source. While this source can be an existing piece of music, it is also possible to employ generative models to generate the required 16 samples. As illustrated in Fig. 5, a model based on GAN architecture can be developed and trained out of OrchNet Encoder, a Generator, and a Discriminator. The generator receives a vector of 2048 which is filled with random values. Using a fully connected layer of 2048 neurons and two layers of 1024 LSTM units (bidirectional), it converts the random input into a 16 by 1024 vector which matches the size of 16 samples required by the LSTM model.

The input to the discriminator comes from either the generator or from OrchNet Encoder encoding 16 consecutive bars. The discriminator performs a binary classification task to determine the source of a given input. The training routine starts with training the discriminator. At this stage, since generator is producing random output, discriminator easily learns how to differentiate between inputs from OrchNet Encoder and the generator. The value of loss function is then back-propagated through discriminator to calculate the loss of the generator. This leads to training the generator to produce more realistic outputs. The cycle repeats and discriminator becomes better at differentiating the source and generator becomes better at producing outputs that it becomes hard for the discriminator to distinguish. At the end, the generator is trained enough to generate the required 16 samples similar to the existing samples in the dataset from a given random vector.

The Monitor-and-Inject Mechanism

The proposed architecture of CompoNet has two major issues:

- being based on LSTM, it suffers from lack of long-term memory,
- it follows a fixed sequential approach which makes it unsuitable to generate different styles and forms.

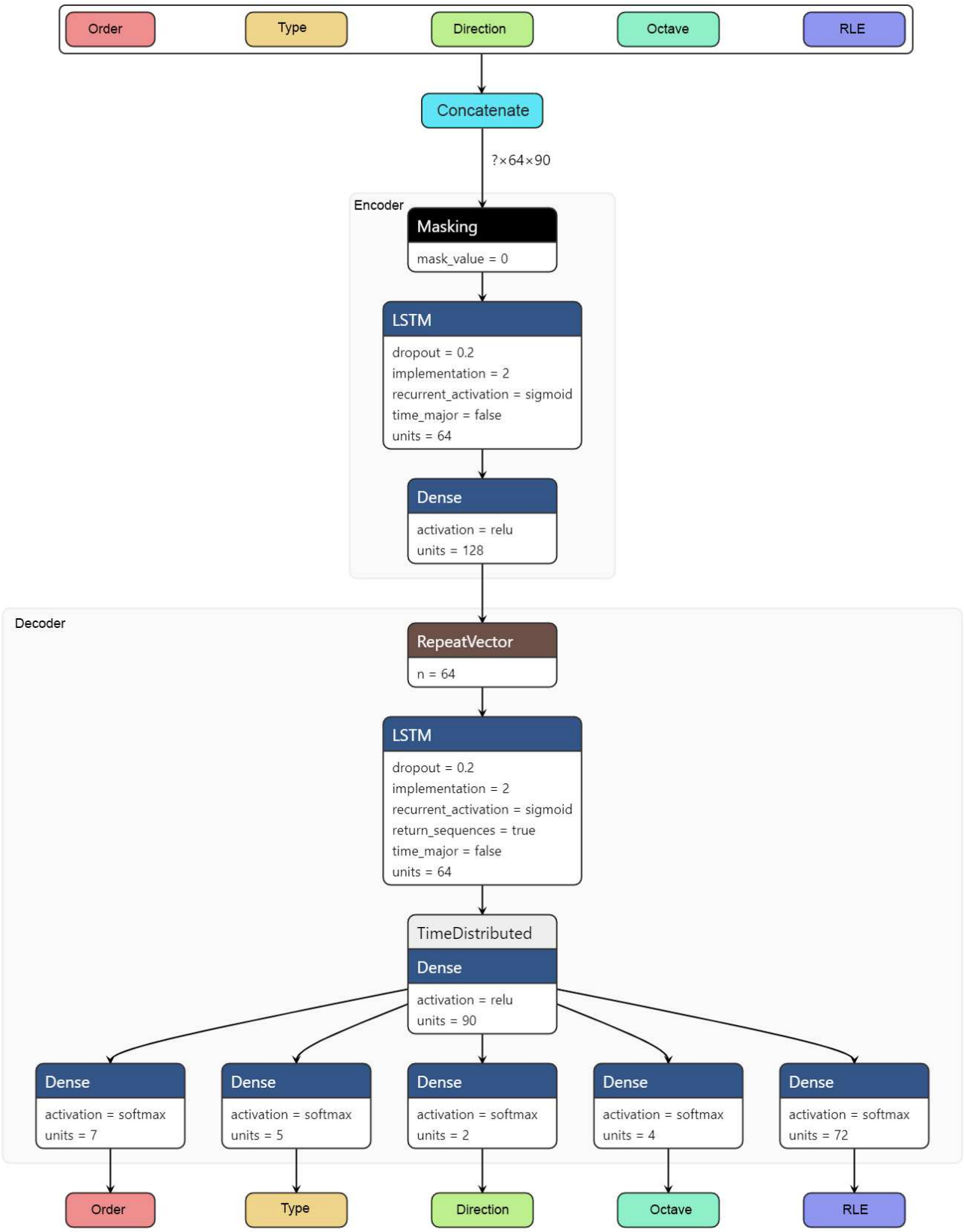


Figure 1. The building block of OrchNet

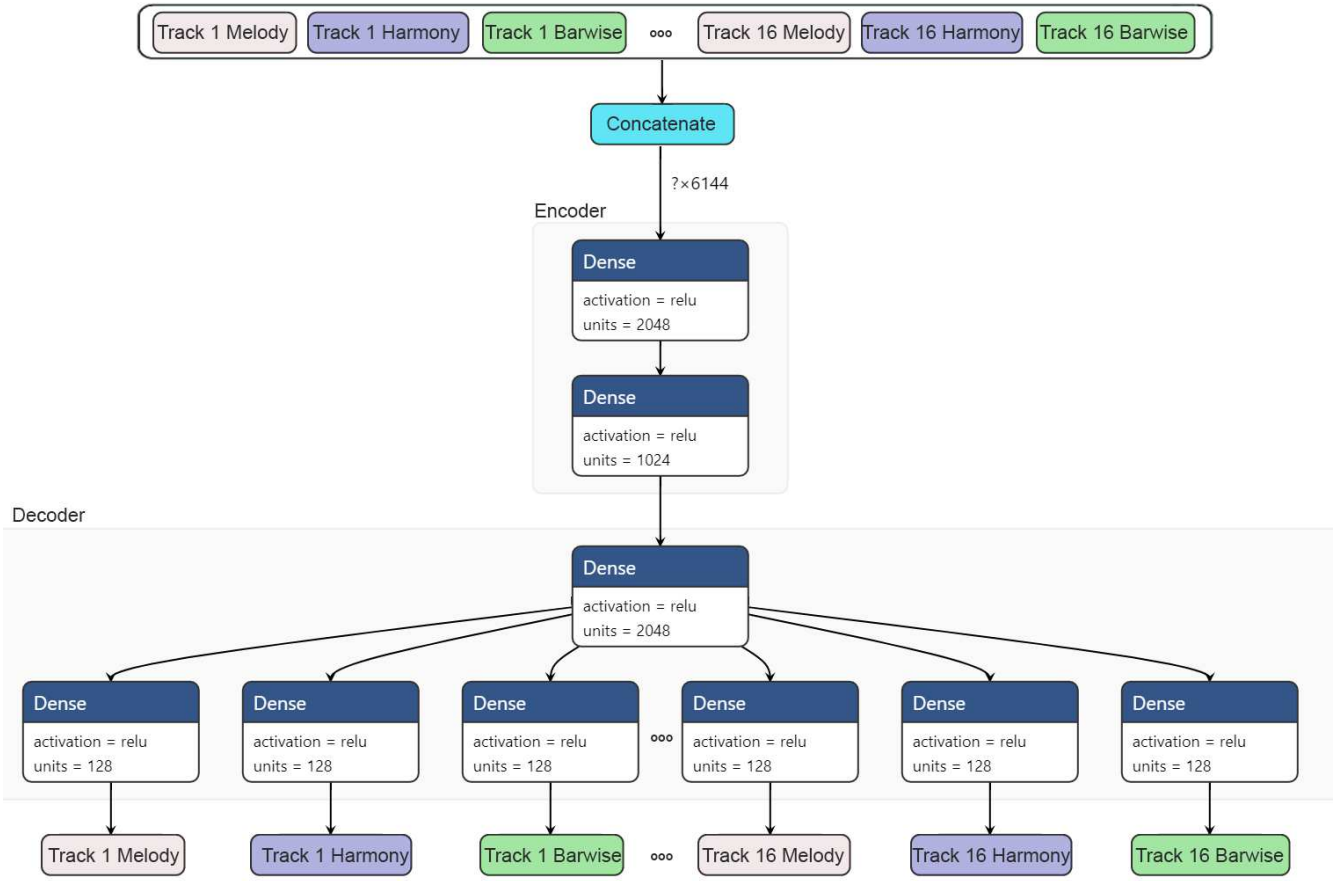
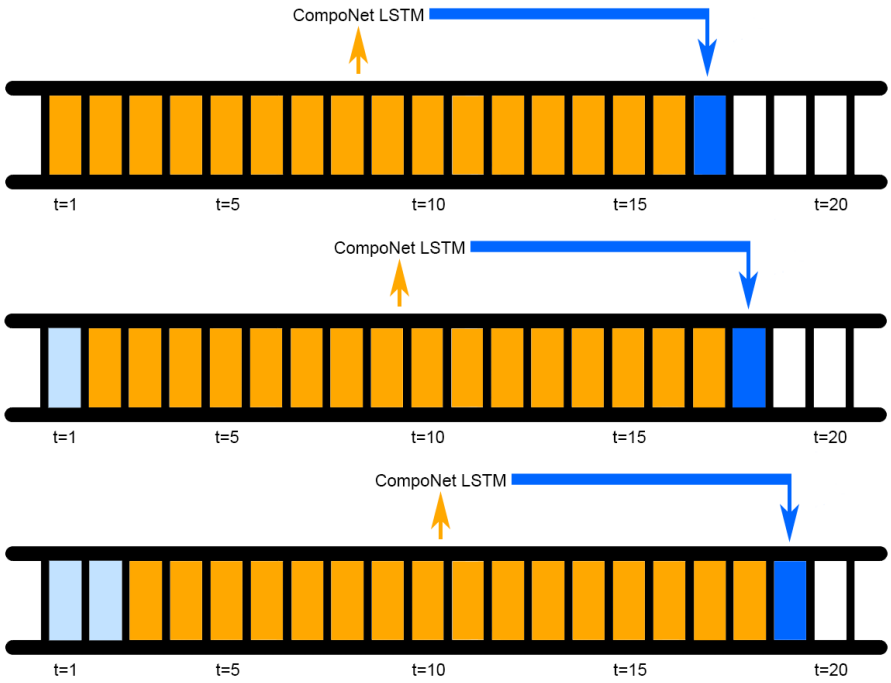


Figure 2. The full stack of OrchNet



window.png

Figure 3. The sliding window mechanism in CompoNet

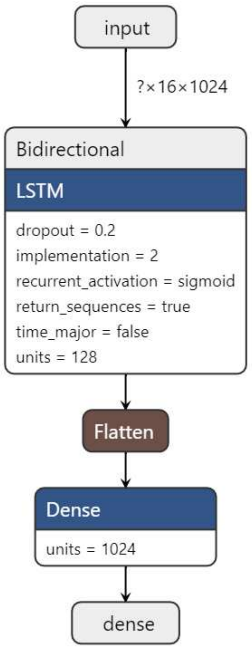


Figure 4. The LSTM model in CompoNet

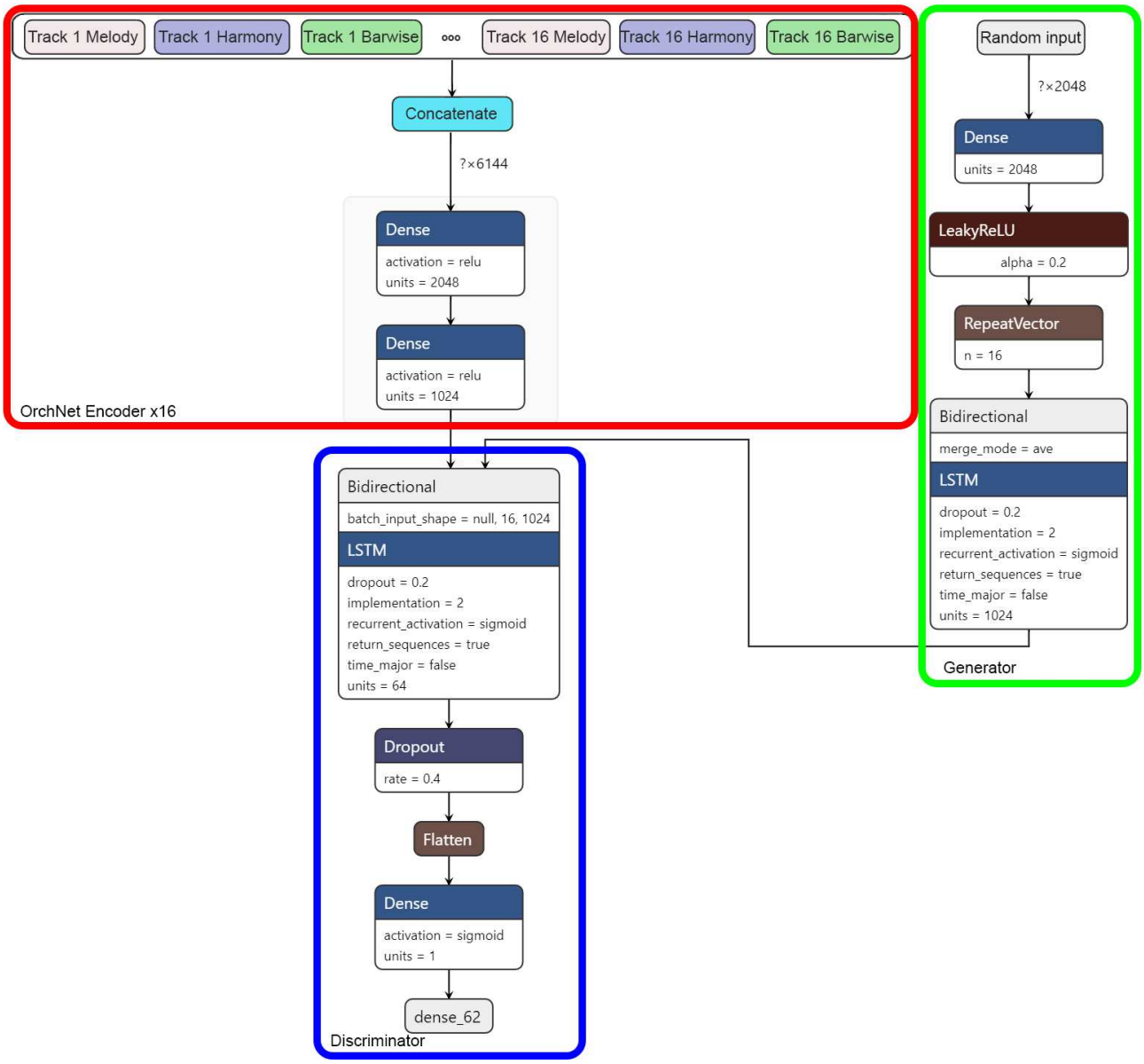


Figure 5. The GAN model in CompoNet

Simple.png

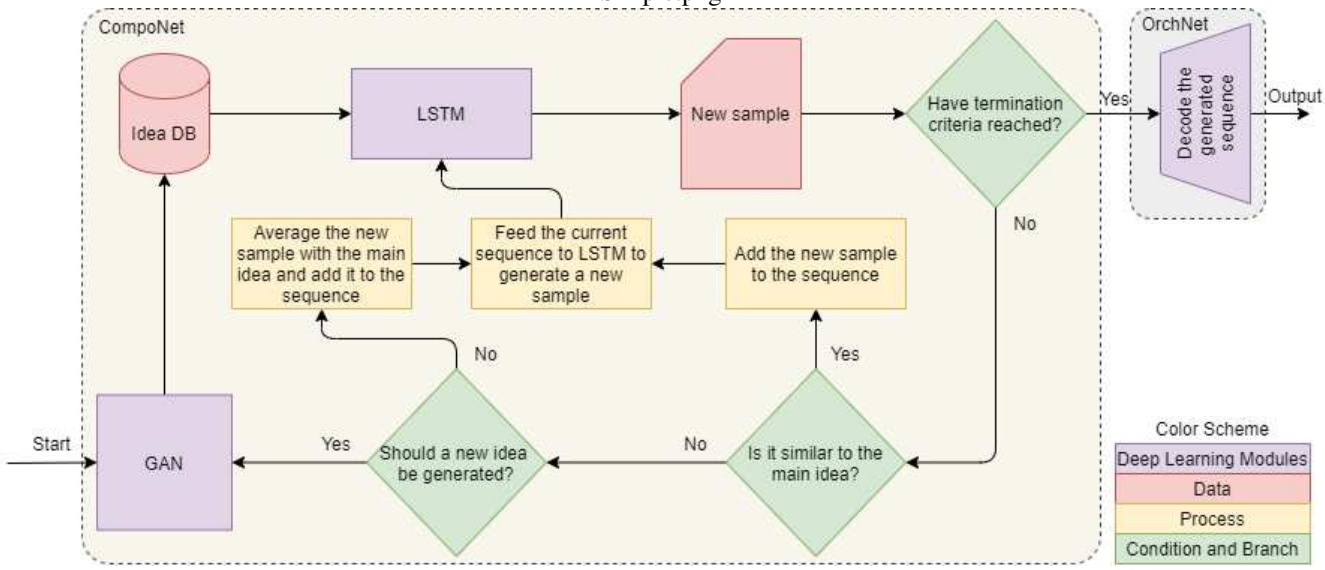


Figure 6. An example flowchart of the monitor-and-inject mechanism for maintaining long-term consistency and adopting different forms and styles

Both of these issue can be addressed by monitoring the output of CompoNet and altering it on the fly as needed. In other words, since CompoNet has a subject/idea generation engine (the GAN model) and a model which maintains the short-term temporal dependencies (the LSTM model), these two components can be combined in virtually infinite number of ways to to generate various styles and form such as *Sonata, Fugue, Nocturne* and so on, as well as new creative ways.

Fig. 6 illustrates an example of how the two components of CompoNet can be combined. While this example does not implement any predefined form or style of music and it has some level of randomness built into it, it ensures long-term dependencies are maintained. The process of generating a new piece of music starts by the GAN model generating the initial seed. The seed is stored in the idea database, then it is given to the LSTM model to generate a new sample. The monitoring process begins by *virtually* adding the new sample to the existing sequence. Then, the similarity of the newest 16 elements of the generated sequence is compared to the initial seed from the GAN. While the impact and application of using different metrics for measuring similarity is a topic for further investigation, here, Mean-Squared Error (MSE) is used. If the similarity meets a certain threshold value, then the new sample will be added to the sequence and the cycle repeats. Otherwise, a decision has to be made to either generate a new seed or reuse a previous one. While different logics can be implemented to determine how previously stored ideas can be reused, here the simple mechanism of roulette wheel is used to give both options of generating new seed and reusing the last seed equal chances. This is important because if the balance between the two options is not maintained, then the output either loses the long-term consistency or generates a repetitive sequence. Either way, if an idea needs to be injected to the sequence, it can be either replaced with the existing sequence or be averaged with it. Here, the averaging option is chosen because it has less impact on short-term dependencies.

Implementation

Python is the main language used to implement this work. Music Embedding, Pypianoroll, and pretty_midi packages were used for data-handling tasks while Keras and TensorFlow were used to define and train the Deep Learning models.

A considerable number of models had to be trained and evaluated for this work. Therefore, it was crucial to exploit hardware accelerators to be able to deliver results in a timely manner. For this reason, all hyper-parameters were set to powers of two if it was possible. Google Colab was chosen as the platform due to its availability and ease of use. Google Colab offers virtual machines (runtime) on Google Cloud Platform (GCP). Each runtime runs on Ubuntu OS and uses Jupyter Notebook as an interface. The free version of Google Colab offers 13 GB of RAM along with access to K80 GPUs. Some models of this work have millions of parameters and the dataset contains hundreds of thousands tensors. Therefore, purchasing the Pro version of Google Colab was necessary. The Pro version of Google Colab provides access to P100 GPUs and 26 GB of RAM. Still, at some models, there was memory limitations on the number of parameters. Also, for some data operations, software-level techniques had to be implemented to mitigate the limited memory problem.

Post-Processing Results

CompoNet generates symbolic music, i.e., it is not an end-to-end approach and the output requires post-processing before being ready to be heard. While different tasks can be implemented in the post-processing, the following were implemented in the current work:

Trimming

OrchNet generates 64 time steps per each sample, the trailing time steps which are the result of zeros padded in the input need to be trimmed. These trailing time steps share the same probability value. This characteristic was utilized by traversing the generated sequence backwards and monitoring the generated probabilities. The sequence was trimmed at the point where the first change in the probabilities was observed.

Sampling

The output branches of OrchNet use softmax activation function. In a classification problem, this function generates the probability of each class being selected. Here, instead of choosing the option with the highest probability, all options are sampled with respect to their probabilities.

Key and Scale Matching

The dataset was generated from MIDI files. While MIDI standard is capable of storing key signature and scale information, it is not mandatory. Since many of the MIDI files in the dataset did not contain this information and it is not possible to accurately determine the scale and key signature of a piece automatically, OrchNet and CompoNet are not designed to contain and learn these information. Therefore, the generated music needs to be matched to a scale and key. This was done by checking all 12 major scales and counting the number of notes that do not follow the scale pattern (alteration). The scale with the minimum number of alterations was chosen and alterations were matched to the closest neutral note (Tonic, Subdominant, Dominant) in the scale. The sequence was trimmed at a note (or chord) matching the scale's Tonic to create a cadence sensation. The minor scales were not considered because each minor scale is relative to a major scale and they share the same key signature; subsequently, they share the same notes. Although a major scale is different from its relative minor, for the purpose of scale matching they can be used interchangeably.

Note Quantification

Since 72 different possibilities are considered for note duration, the output does not automatically snap to standard note duration such as whole or black notes. It is possible to have a note which is only one pixel short of a standard note duration. Similarly, it is possible to have a note with only one pixel. In these cases it is reasonable to apply note quantification to enhance both rhythm and readability of the generated score. It is worth mentioning that the 72 possibilities were considered for the input data because different pieces have different rhythmic patterns and there is no one-fits-all approach to mitigate this.

Assigning Instrument

One advantage of generating music in symbolic format rather than audio format is that each track can be assigned to different instruments and ensembles to find the most suitable timbre for the generated sequence. This was done considering the available options and aesthetic aspects.

Octave and Range Matching

OrchNet has 4 bits to encode the octave of an interval. Due to the employed sampling mechanism, it is possible to have intervals spanning over three octaves. However, the assigned instrument might have a limited range and not be able to perform the generated note. In such scenario, the out-of-range notes were mapped to identical note which is in the feasible range.

Audiolization

The generated symbolic music needs to be converted to audio format to be heard. This was done using *Sibelius Music Notation Software 6* and its *Essential Sound Set*[31].

Turing Test

Evaluation of results is one the major challenges in AMG[32]. This is because it is not possible to develop metrics to assess the quality of generated results in an objective manner. To mitigate this, Turing test can be utilized to evaluate the results if human music listeners can tell the difference between the pieces generated by our system and human music composers (shown in Table 3).

Table 3. List of pieces used in the Turing test

Composer	Piece Name
Albeniz	Torre Bermeja Serenata from Piezas caracteristicas
Alkan	Etude Op 35 No 5
Bacewitz	Sonata No 4 for Piano Violin Mov 4
Coleridge-Taylor	La Tarantelle Fretillante
Faure	Dolly suite
Liszt	Valse Oubliee No 1
Meyerbeer	Grand March
Satie	Gnossienne No.3
Shostakovich	Ten short piano pieces No 1 Recitative
Wagner	Traume

Test Design

To perform the Turing test, 10 pieces were generated by CompoNet to be compared against 10 randomly-selected pieces from the dataset. To keep the test fair and reasonable, the following steps were taken:

- To ensure respondents are judging the pieces rather than answering the question from their memories, famous pieces were removed from consideration.
- To ensure the sequence of pieces is not impacting the results, they were put in a random sequence.
- To ensure the quality of audio is not impacting the results, all 20 pieces were audiolized with the same engine.
- To increase the likelihood of respondent completely listing to the piece before making a decision, all pieces were kept under one minute.

For each piece, respondents were asked *"How do you evaluate this piece of music?"* and they were asked to choose one the following options:

- A. I am confident it is composed by a human.
- B. I think it might be composed by a human.
- C. I am not sure.
- D. I think it might be composed by AI Software.
- E. I am confident it is composed by AI Software.

Conducting the Test

Questionpro website was used as the platform of conducting the test. The link to the test was published via author's social media, asking for volunteers to take the test. The link was visited 260 times, the test was started 169 times, and 88 respondents completed the test. This study was approved by the institutional review board of North Carolina A&T State University. No personal information was collected from the study participants. Informed consent was obtained from each participant prior to their participation in the study. The study was fully compliant to the IRB guidelines.

Demography of Respondents

Before beginning the test, respondents were asked to self-identify their age and gender groups voluntarily. Fig. 7 and Fig. 8 illustrate the age and gender distributions of respondents respectively. Additionally, *Questionpro* automatically provides geographical distribution of respondents which is provided in Table 4 and Fig. 9. Based on these information, the respondents of this test include a diverse range of age, gender, and geographical location, which reduces potential biases in the results.

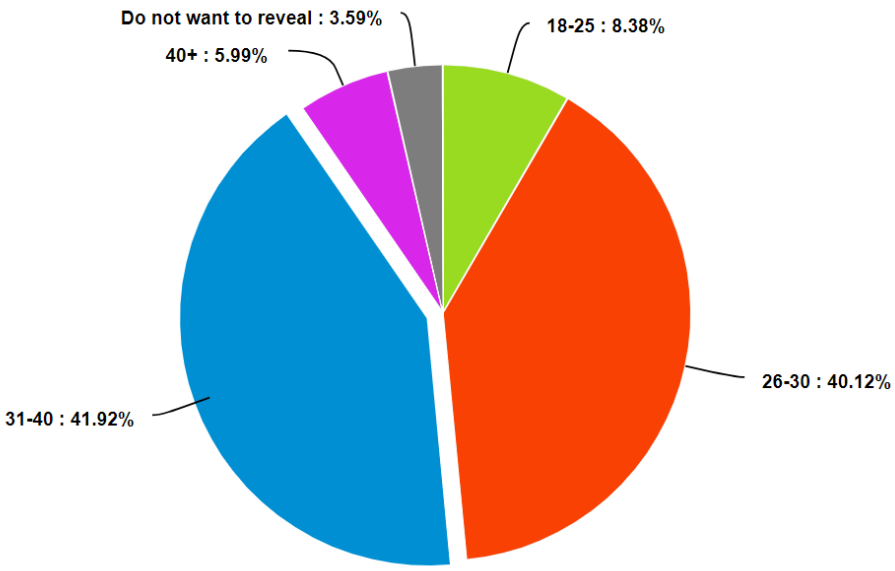


Figure 7. Age distribution of all respondents

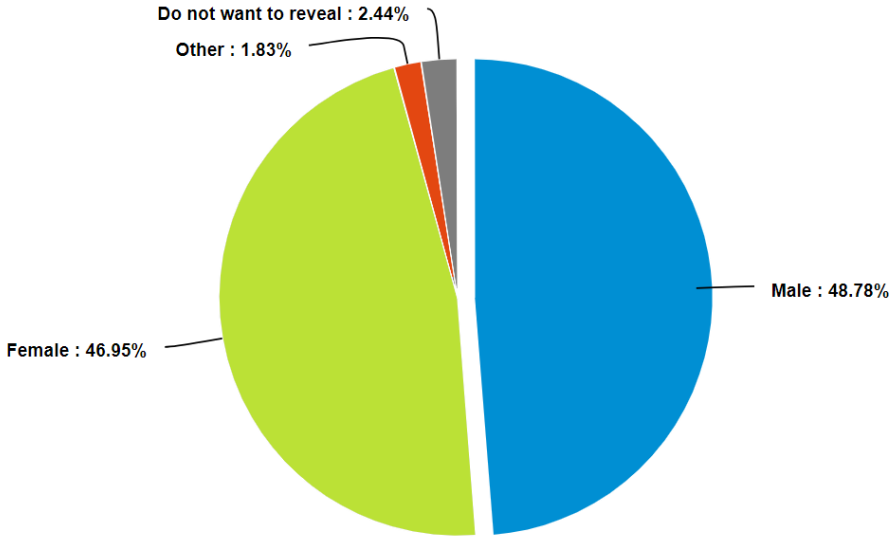


Figure 8. Gender distribution of all respondents

Figure 9. Geographical distribution of all respondents

Table 4. Geographical distribution of all respondents

Country	Responses
United States	65.19%
Iran	23.76%
France	2.76%
Germany	1.66%
Canada	1.10%
United Kingdom	1.10%
Italy	1.10%
Netherlands	1.10%
Romania	0.55%
Switzerland	0.55%
Austria	0.55%
Australia	0.55%
Total	100.00%

Table 5. Mapping responses to numerical values

Response	Value
I am confident it is composed by a human.	1
I think it might be composed by a human.	0.5
I am not sure	0
I think it might be composed by AI Software.	-0.5
I am confident it is composed by AI Software.	-1

Test Results

A scoring mechanism was used to compare the pieces of music in the test. The score of each piece was calculated as:

$$s = \frac{\sum_{i=1}^{n} v_i}{n} \tag{1}$$

where s is the score of the piece, i is the number of respondent, n is the total number of respondents, and v_i is the value of the i_{th} respondent's response for the piece based on Table 5. The calculated score is in the range of [-1, 1], where -1 means all respondents have confidently identified the piece synthesized by CompoNet, while 1 shows that all respondents were confident the piece was composed by human.

Based on Table 6, which the contains the score of each piece calculated from the result of respondents who completed the test, both the highest and lowest scores belong to CompoNet outputs. Both human-composed pieces and CompoNet outputs are almost equally present in the upper and lower halves of Table 6. The statistical characteristics of CompoNet and pieces composed by human are provided in Table 7. Since the obtained scores do not follow a normal distribution and the number of samples in each group is 10, the non-parametric Mann–Whitney U test was used to check the validity of the null hypothesis:

H_0; There is a difference between the ranks of the pieces generated by CompoNet and composed by human.

versus the alternative hypothesis:

H_1; There is no difference between the ranks of the pieces generated by CompoNet and composed by human.

The result of Mann-Whitney test is as follows: $U(N_{CompoNet} = 10, N_{Human} = 10) = 33.5, z - score = -1.209941, p - value = 0.226301 (p(x \leq Z) = 0.113151)$. The test statistic Z is in the 95% critical value accepted range: [-1.9600 : 1.9600]. The $U - value$ satisfies the critical value of U at $p < 0.05$ which is 23. Therefore, the result is not significant at $p < 0.05$. The $p - value$ equals 0.226301, $(p(x \leq Z) = 0.113151)$. This means that if we would accept H_0, the chance of type II error (accepting a false null hypothesis) would be too high: 0.2263 (22.63%). In other words, the difference between the randomly selected output of CompoNet and pieces composed by human is not high enough to be statistically significant. This indicates that CompoNet output is indistinguishable from pieces which are composed by human and CompoNet can mimic classic composers whose works are present in the dataset.

References

1. Hiller, L. A. & Isaacson, L. M. *Experimental Music; Composition with an Electronic Computer* (Greenwood Publishing Group Inc., USA, 1979).
2. Xenakis, I. *Formalized Music: Thought and Mathematics in Composition*. Harmonologia series (Pendragon Press, 1992).
3. Blackwell, T. Swarm music: improvised music with multi-swarms. *Artif. Intell. Simul. Behav. Univ. Wales* **10**, 142–158 (2003).
4. Eck, D. & Schmidhuber, J. A first look at music composition using lstm recurrent neural networks. *Istituto Dalle Molle Di Studi Sull Intelligenza Artif.* **103**, 48 (2002).
5. Van Der Merwe, A. & Schulze, W. Music generation with markov models. *IEEE MultiMedia* **18**, 78–85, 10.1109/MMUL.2010.44 (2011).
6. Kosta, K., Marchini, M. & Purwins, H. Unsupervised chord-sequence generation from an audio example. In *ISMIR*, 481–486 (2012).
7. Colombo, F., Muscinelli, S. P., Seeholzer, A., Brea, J. & Gerstner, W. Algorithmic composition of melodies with deep recurrent neural networks. *arXiv preprint arXiv:1606.07251* (2016).

Table 6. Score of each piece based on the Turing test

Piece	Composer	Score
8	CompoNet	0.3920
La Tarantelle Fretillante	Coleridge-Taylor	0.3391
Gnossienne No.3	Satie	0.3352
9	CompoNet	0.1591
4	CompoNet	0.1080
Sonata No 4 for Piano Violin Mov 4	Bacewitz	0.0852
Valse Oubliee No 1	Liszt	0.0398
Dolly suite	Faure	0.0341
Etude Op 35 No 5	Alkan	0.0000
10	CompoNet	0.0000
Grand March	Meyerbeer	-0.0568
Traume	Wagner	-0.0739
1	CompoNet	-0.0795
Torre Bermeja Serenata from Piezas caracteristicas	Albeniz	-0.1092
6	CompoNet	-0.1477
Ten short piano pieces No 1 Recitative	Shostakovich	-0.2102
5	CompoNet	-0.2386
3	CompoNet	-0.2644
7	CompoNet	-0.2670
2	CompoNet	-0.2784

Table 7. Statistical characteristics of the performed Turing test

	Median	Mean	Standard Deviation
CompoNet	-0.1136	-0.0616	0.2136
Human	0.0170	0.0383	0.1695

8. Sturm, B. L., Santos, J. F., Ben-Tal, O. & Korshunova, I. Music transcription modelling and composition using deep learning. *arXiv preprint arXiv:1604.08723* (2016).

9. Jaques, N. *et al.* Tuning recurrent neural networks with reinforcement learning (2017).

10. Dieleman, S., van den Oord, A. & Simonyan, K. The challenge of realistic music generation: modelling raw audio at scale. In *Advances in Neural Information Processing Systems*, 7989–7999 (2018).

11. Van den Oord, A. *et al.* Wavenet: A generative model for raw audio. arxiv 2016. *arXiv preprint arXiv:1609.03499* .

12. Huang, C. A. *et al.* An improved relative self-attention mechanism for transformer with application to music generation. *CoRR* **abs/1809.04281** (2018). 1809.04281.

13. Hadjeres, G., Nielsen, F. & Pachet, F. Glsr-vae: Geodesic latent space regularization for variational autoencoder architectures. 1–7, 10.1109/SSCI.2017.8280895 (2017).

14. Goodfellow, I. *et al.* Generative adversarial nets. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D. & Weinberger, K. Q. (eds.) *Advances in Neural Information Processing Systems 27*, 2672–2680 (Curran Associates, Inc., 2014).

15. Yang, L.-C., Chou, S.-Y. & Yang, Y.-H. Midinet: A convolutional generative adversarial network for symbolic-domain music generation. *arXiv preprint arXiv:1703.10847* (2017).

16. Mogren, O. C-rnn-gan: Continuous recurrent neural networks with adversarial training. *arXiv preprint arXiv:1611.09904* (2016).

17. Yu, L., Zhang, W., Wang, J. & Yu, Y. Seqgan: Sequence generative adversarial nets with policy gradient. In *Thirty-first AAAI conference on artificial intelligence* (2017).

18. Guimaraes, G. L., Sanchez-Lengeling, B., Outeiral, C., Farias, P. L. C. & Aspuru-Guzik, A. Objective-reinforced generative adversarial networks (organ) for sequence generation models. *arXiv preprint arXiv:1705.10843* (2017).

19. Dong, H.-W., Hsiao, W.-Y., Yang, L.-C. & Yang, Y.-H. Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment. In *Thirty-Second AAAI Conference on Artificial Intelligence* (2018).

20. Dong, H.-W. & Yang, Y.-H. Convolutional generative adversarial networks with binary neurons for polyphonic music generation. *arXiv preprint arXiv:1804.09399* (2018).

21. Guan, F., Yu, C. & Yang, S. A gan model with self-attention mechanism to generate multi-instruments symbolic music. In *2019 International Joint Conference on Neural Networks (IJCNN)*, 1–6 (2019).

22. Raffel, C. *Learning-based methods for comparing sequences, with applications to audio-to-midi alignment and matching*. Ph.D. thesis, Columbia University (2016).

23. Schmidhuber, J. Composing music with lstm recurrent networks - blues improvisation, http://people.idsia.ch/ juergen/blues (2002).

24. The session data https://github.com/adactio/thesession-data (2020).

25. Nottingham dataset, http://www-labs.iro.umontreal.ca/ lisa/deep/data (2012).

26. Theory tab, https://www.hooktheory.com/theorytab.

27. HekmatiAthar, S. & Anwar, M. Music embedding: A tool for incorporating music theory into computational music applications (2021). 2104.11880.

28. Fedorak, B. Classical music of various composers in midi format, https://www.kaggle.com/blanderbuss/midi-classic-music/ (2018).

29. Hao-Wen Dong, Y.-H. Y., Wen-Yi Hsiao. Pypianoroll: Open source python package for handling multitrack pianorolls (2018).

30. Colin Raffel, D. P. W. E. Intuitive analysis, creation and manipulation of midi data with pretty_midi (2014).

31. Avid. Sibelius essential sound set (2021).

32. Liebman, E. & Stone, P. Artificial musical intelligence: A survey. *arXiv preprint arXiv:2006.10553* (2020).

Author contributions statement

S.H. and M.A. designed the study. S.H. implemented the algorithm and conducted the study. S.H. and M.A. analyzed the results. S.H. wrote the first draft. M.A. supervised the research and contributed to the writing of the manuscript.

Competing interests

The authors declare no competing interests.