Generating music with data: Application of Deep Learning models for symbolic music composition

Pedro Ferreira, Ricardo Limongi
Universidade de São Paulo

Abstract

Language models based on deep learning have shown promising results for artistic generation purposes, including musical generation. However, the evaluation of symbolic musical generation models is mostly based on low-level mathematical metrics (e.g. result of the loss function) due to the inherent difficulty of measuring the musical quality of a given performance, given the subjective nature of music.

This work seeks to measure and evaluate musical excerpts generated by deep learning models from a human perspective, limited to the scope of classical piano music generation.

In this assessment, a population of 117 people performed blind tests with musical excerpts of human composition and musical excerpts generated through artificial intelligence models, including the models PerformanceRNN (Oore et al., 2018), Music Transformer (Cheng-Zhi et al., 2018), MuseNet (Payne, 2019), and a custom model based on GRUs (Lee, 2020).

The experiments demonstrate that musical excerpts generated through models based on the Transformer neural network architecture (Vaswani et al. 2017) obtained the greatest receptivity within the tested population, surpassing results of human compositions. In addition, the experiments also demonstrate that people with greater musical sensitivity and musical experience are more able to identify the compositional origin of the excerpts heard.

Comments from participants with no musical experience, participants with musical experience and professional musicians about some of the tested musical excerpts were also included in this work.

Keywords: music generation; artificial intelligence; NLP;

1. Introduction

Music is one of the universal aspects of all human societies. From tribes in the Amazon to small villages in the Chinese countryside, from ancient Greek society to contemporary digital society, humans use music as a means of expression. It is interesting to note that we can identify concepts in the musical structure that are similar to concepts in the linguistic structure, such as “phrases”, “stresses”, “punctuations”, “sentences”, among others.

Furthermore, as presented by Adorno and Gillespie (1993), music also resembles a language in that both are a temporal succession of articulated sounds that are more than just a single sound, following a certain coherence. Also, in both, we are highly sensitive to the subtleties and irregularities of these sequential data flows (Walder, 2016). The comparison between music and language occurs naturally, considering that both involve complex sequences with meaning. From a neuroscience perspective, there is a growing body of evidence linking the way we perceive music and the way we perceive natural language (Patel, 2010). On the other hand, natural language has been a much-discussed topic within the data world, considering the growing number of articles related to natural language processing (NLP) and models of deep artificial neural networks (Deep Learning). Among these researches, it is worth highlighting the problem of data-driven natural language generation, a topic that is currently the object of rigorous investigation by NLP researchers (Graves, 2013; Mathews et al. 2016).

In addition, it is also worth noting the structural similarities of musical generation models and natural language generation models, based on deep artificial neural networks. The model MuseNet presented by Payne (2019), developed by OpenAI can be taken as an example. This model is based on the GPT-2 language model, also developed by OpenAI and presented by Radford et al. (2019). Taking into account the structural similarities of the sequential nature of language and music, would it be possible for generative neural network architectures, based on natural language models, to express notions of harmony and melody pleasant enough to human ears? Would this musical content artificially generated by neural networks be robust enough to confuse a listener about its compositional origin? These questions served as a basis for the evolution of this work, which sought to explore and validate the result of the application of language models and generative neural network architectures for the purposes of algorithmic music composition.

The main goal of this paper was to measure the musicality of musical excerpts generated by "deep learning" models from a human perspective. To achieve this objective, a population of 117 people, including musicians and non-musicians, was subjected to a blind test to evaluate the musicality of five different musical excerpts, one human composition and the four algorithmic compositions generated by deep learning models. The result of this research contributes with new information about how the musicality of the models differ from each other and how this musicality is perceived in relation to a music of human compositional origin.

Another contribution of this article is to demonstrate that within the population tested, people with greater sensitivity or musical experience are more assertive in defining the compositional origin of the musical excerpts used. The source code of this study will be publicly available, in order to facilitate experimentation and access to the models and data used in this research.

2. Material and methods

Algorithmic music composition presents a challenging technical development. As described by Walder (2016), unlike words that make up the data of a text, multiple musical notes can occur contemporaneously. To address this problem, generative models based on deep neural networks were used for music generation purposes through three types of different approaches, including:

- End-to-end model development (From data to inference)
- Partial model development using transfer of learning through pre-trained model weights
- Human-machine interaction through web interface.

It is worth mentioning that the end-to-end model development approach relies on training techniques that were incorporated with the objective of maximizing the learning of the musical characteristics of the selected musical corpus. After training, the samples were generated in an auto-regressive way from the distributions estimated by the model, following the same modus operandi of the other models used. These distributions were compiled and transformed into musical excerpts, which served as the basis for the blind test, which sought to assess the musical quality of the model from a human perspective. From here, we can highlight the three main pillars for the development of this research, being the **data**, the **models,** and the **blind test**.

2.1. Data

There are multiple perspectives of data collection in relation to a musical artifact, starting from a more elementary approach such as musical analysis of elements in a score to more complex analyzes starting from the raw sound wave of a certain audio file. This work focuses on symbolic musical data, based on the MIDI standard (Loy, 1985) within a musical scope of classical music.

The selection criterion for the dataset choice was based on the academic relevance in the domain of applied machine learning for music information retrieval (MIR). Among the listed sources, 3 main datasets were considered as potential candidates for the scope of this work, namely:

Maestro: Introduced by Hawthorne et al. (2018), Maestro is a music dataset containing over 200 hours of virtuosic piano performance in MIDI format.

JSB Chorales: Introduced by Boulanger-Lewandowski et al. (2012), the JSB choirs are a set of short musical pieces, in four voices, noted for their stylistic homogeneity. The chorales were originally composed by Johann Sebastian Bach in the 18th century. He wrote them by taking pre-existing melodies from contemporary Lutheran hymns and then harmonizing them to create the remaining three-voice parts. The version of the dataset used canonically in representational learning contexts consists of 382 of these corals, with a training/validation/test split of 229, 76, and 77 samples, respectively.

MuseData: Also introduced by Boulanger-Lewandowski et al. (2012), MuseData is an electronic library of classical orchestral and piano music from the Center for Computer-Assisted Research in the Humanities at Stanford University (CCARH). It consists of about 3MB of 783 files.

However, even though these databases are within the same scope (classical music MIDI files), they have different structural natures. Maestro is a piano performance database, JSB Chorales is a 4-part choral database, and MuseData is a miscellaneous compilation of classical orchestral and piano music.

In order to achieve certain consistency within the distribution of information and the modeling process, the choice of a dataset with the minimum variation in its structural nature was chosen. The Maestro database proved to be the most appropriate within these criteria due to its consistency in terms of structural organization of information, since all elements of this database are derived from the same instrument (piano).

2.1.1. Data augmentation

Data augmentation techniques for generating synthetic data were used in order to maximize the assertiveness of the models. These techniques are based on musical transposition and temporal transposition techniques. Musical transposition (or tonal transposition) changes the sequence of music notes to a harmonic field above or below the original harmonic field, meaning that the intervals of the sequences remain the same, but within a different frequency range (e.g. a musical passage with the sequence [do - re] transposed one tone higher becomes [re - mi]). The adopted approach transposes the musical excerpt to all intervals up and down by up to 6 semitones to span a full octave, resulting in 11 new examples plus the original.

On the other hand we have temporal transposition, that focuses on stretching or compressing the notes within a temporal space, that is, spreading the notes in a space larger than the original one (making the performance slower) or compressing them in a space smaller than the original one (leaving the performance faster). The adopted approach stretches each example uniformly in time by ±25% and ±50% , resulting in 4 new examples plus the original. So, for every number n of examples, we will get a number of (n * (12 * 5)) examples after the transformations.

The Maestro dataset has a total of 1276 files already indexed in training (962 files), validation (137 files) and test (177 files). The validation files were added to the training files and the test files were used as validation, thus generating a separation of 1099 training files and 177 validation files. This separation approach aims to maximize the use of available data for model optimization.

Taking into account that the transformations are only performed in the training split, the dataset used in this research adds up to a total of 66,117 performances, of which 65,940 (1099 * (12 * 5)) are used for training and 177 are used for validation.

2.1.2. Data Wrangling

Data wrangling techniques were also applied to refine the data in order to optimize processing and maximize model assertiveness. The MIDI file pre-processing approach for music generation implemented by Oore et al. (2018) and later replicated by Huang et al. (2018) aims to transform MIDI musical elements into a sequence of tokens, which in turn will form the vocabulary of the problem. This vocabulary consists of:

- 128 possible types of **note_on** events representing the start of a note with one of 128 possible MIDI tones.

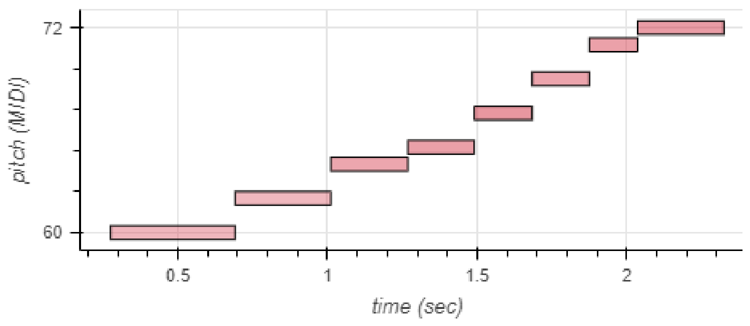
- 128 possible types of **note_off** events that represent the ending of a note with one of 128 possible MIDI tones.

- 32 possible types of intensity events defined as **velocity**. The pitch precedes a **note_on** event, thus defining the pitch dynamics of the future note.

- 100 possible types of time_shift events, representing a temporal shift in sets of 10ms increments, having a temporal possibility space between 1 (10 ms) and 100 (1s).

- **piece_start** token as the starting token of each sequence, symbolizing the beginning of the performance

- **piece_end** token as the final token of each sequence, symbolizing the end of the performance



piece_start time_shift=27 velocity=12
note_on=60 time_shift=42 note_off=60
note_on=62 time_shift=32 note_off=62
velocity=15 note_on=64 time_shift=26
note_off=64 velocity=16 note_on=65
time_shift=22 note_off=65 velocity=18
note_on=67 time_shift=19 note_off=67
velocity=15 note_on=69 time_shift=20
note_off=69 velocity=16 note_on=71
time_shift=16 note_off=71 velocity=17
note_on=72 time_shift=28 note_off=72
piece_end

Figure 1: Plot of a C major scale followed by its tokenized sequence

Note: On the left we can visualize the performance of a scale in C major in a two-dimensional plot of note (pitch) as a function of time (piano roll). On the right we can see the token representation of this performance.

The sum of all elements make up a vocabulary of 390 tokens. It is worth mentioning that the last two transformations (adding **piece_start** and **piece_end** tokens) were inspired by the work of Ens (2020). It is also interesting to detail the approaches adopted in the elements of “velocity” (intensity) and time shift.

2.1.2.1. Intensity quantization

The intensity (“velocity”) of a note is perceived as how loud or soft the note must be articulated. The common notation adopted in classical music has 8 levels of dynamic marking, ranging from ***ppp*** (pianissimo) to ***fff*** (fortissimo). However, the intensity component of the MIDI protocol has 128 levels of variation, ranging from 0 to 127.

The strategy adopted by Oore et al. (2018) and later replicated by Cheng-Zhi et al (2018) was to quantize the possible 128 MIDI intensity levels into 32 “bins”. Taking into account that the original classical music notation has 8 levels of variation, 32 levels would be enough to reduce the sparsity of the model, leading the intensity component to be representative within a smaller space.

2.1.2.2. Quantization of time shift

Timeshift events are calculated as the difference between the temporal distance from the end of a note to the beginning of the next note in the sequence. The time difference calculation in its raw form takes place in a continuous space at the microsecond level. Thus, tokenizing the difference of all temporal possibilities would leave the model calculation unnecessarily sparse, since as noted by Friberg and Sundberg (1993), the perceptible difference in temporal displacement of a single tone in a sequence was not less than approximately 10ms in its majority.

Because of this, to reduce the scope of temporal possibilities, each temporal displacement between one note and another was quantized in a range from 1 (10ms) to 100 (1s) in 10ms increments. As a result, we obtain the space continuum quantization of the temporal displacement in a discrete space of 100 possibilities.

It is also noteworthy that the duration of the note is extended, referring to the fact that the piano pedal is considered in the total duration of the note.

2.1.2.3. Dataset processing and technologies used

The tokenization process takes place in the entire augmented training and validation dataset, with a text file containing the concatenation of sequences of tokens as output.

The developed pre-processing structure is responsible for the acquisition and transformation of the MIDI dataset. The data acquisition part occurs through the download of the MAESTRO dataset (Hawthorne et al. 2018). As mentioned earlier, this dataset is a compilation of ten years of MIDI recordings from the international piano e-Competition. All performances were obtained through a Disklavier, which is a real piano with internal sensors that record MIDI events corresponding to the performer's actions. The MIDI dataset files are organized by the performance competition year. The dataset also has a separation recommendation table for machine learning purposes, where each file is associated with a training, validation, or test label.

The merge of validation data with training data was adopted as an approach, thus using test data for validation. In other words, the division training, validation, and testing becomes training (old training + old validation) and validation (old testing). The choice of this approach comes from the nature of the problem to be solved. Considering that the purpose of this work is to achieve the generation of musical sequences through a causal language model, the test division used in the traditional way does not add much value, considering that its practical application would be extremely similar to the validation division. The adopted approach therefore expands the size of the training dataset, aiming to benefit the model with more data.

The entire process of data acquisition and organization into training and validation takes place in an automated way using bash and Python scripts. With the MIDI files organized, the transformation process can begin.

The transformation process takes place in the following sequence for each MIDI file:

- Applying “sustain” pedal controls, extending the duration of each note according to the pedal dynamics used.
- Synthetic data augmentation using temporal transposition, “stretching” the time by ±25% and ±50%, resulting in 4 new examples plus the original.
- Synthetic data augmentation using pitch transposition in all intervals up and down by up to 6 semitones, resulting in 11 new examples plus the original
- Extracting and transforming MIDI events and text strings.

Each MIDI file (followed by its respective synthetic data) is transformed into a textual sequence. Each textual string is added as a line in the text file relative to its grouping (training or validation). The result of the preprocessing is the transformation of all the MIDI files into two text files, data_train.txt and data_validation.txt. Their sizes are respectively 18GB and 2.1GB after preprocessing all MAESTRO dataset files. The entire data transformation process takes place in an automated manner using the Python programming language, combined with the **note-seq** and **pandas** libraries.

2.2 Models

This work used four different music generation models, two of which are based on recurrent neural network architecture, including Long-Short Term Memory (**LSTM**) models (Yu et al. 2019) and **Gated Recurrent Unit** models (Cho et al. 2014). The other two models were based on the **Transformers** architecture (Vaswani et al. 2017). Among the four models, three use pre-trained weights and one is trained from scratch through an end-to-end execution pipeline (from raw data to training). It is worth mentioning that a GPT2 model (Custom GPT2) was also trained in the same pipeline developed in this work. However, due to hardware limitations, the model did not converge to a significant result to the point of being considered for the blind test.

The reason for choosing this specific set of models came from the idea of comparing musicality between prominent models such as Music Transformers (Cheng-Zhi et al., 2018), MuseNet (Payne, 2019) and Performance RNN (Oore et al., 2018) from a human perspective. This choice also aimed to understand if more complex pre-trained models like Music Transformers and its predecessor Performance RNN would achieve a higher level of musicality than smaller models with similar architecture (Custom GPT2, Custom GRU). In this way, each model would represent a different type of musical quality.

Below, the table of the models that were applied during the research with their respective initial expectations of musicality:

Table 1: List of models used in this work

Model name	Model type	Pre-trained	Musicality expectation
Custom GRU	RNN - GRU	No	Low
Performance RNN	RNN - LSTM	Yes	High
Custom GPT2	Transformer	No	Average
Music Transformers	Transformer	Yes	Very high
MuseNet	Transformer	Yes	Very high

For the pre-trained models, transfer learning was used based on notoriously performant models available in each architecture (Ji et al., 2020). For the Long-Short Term memory architecture, the weights of the **Performance RNN** model presented by Oore et al. (2018) was used. For Transformer-type models, the weights of the **Music Transformers** model, implemented by Cheng-Zhi et al (2018), was used. The **MuseNet** model does not have neither the code nor the model weights open-sourced, but it has a web interface for music generation, which was used in this work. The choice of adopting pre-trained models comes from the hardware limitation to achieve high performance results, given that training models of this nature are inherently complex from a computational perspective.

The entire implementation of the models was carried out using the Python programming language, combined with the **Keras framework** (Chollet, 2015) with **TensorFlow** (Abadi et al. (2015)) and **PyTorch** (Paszke et al. 2019). HuggingFace ecosystem libraries such as **transformers** (Wolf et al. 2019), **datasets** (Lhoest et al. 2021) and **tokenizers** were also used.

2.2.1 Framework for music generation with Performance RNN

The first step of this structure is to obtain the weights of the pre-trained model. As detailed by Simon and Oore (2017), the weights of the models, as well as their source code, are made available by Google (financer of the original research) through its Magenta repository on GitHub. The use of this model is linked to dependencies of other Python libraries, so the creation of a virtual environment with all the necessary dependencies proved to be interesting for reproducibility purposes. The **conda** technology was used to create a virtual environment with all the necessary dependencies.

Custom scripts for using and manipulating the model for MIDI generation were also developed in this work.

Finally, as a result of the sum of the aforementioned elements, we obtain the structure for generating MIDI files with the pre-trained Performance RNN model. The structure can be synthesized in the following steps:

- Obtaining model weights and source code.
- Creation of a virtual environment for dependency resolution and reproducibility.
- Use of custom scripts to manipulate the model for MIDI generation.

2.2.2 Framework for music generation with Music Transformers

Similar to the Performance RNN model structure, the first step of the music generation structure with Music Transformers is to obtain the weights of the pre-trained model. Similar to the Performance RNN, the model's source code is available in Google's Magenta repository (also funding the original research). However, the weights of this model are not as accessible as the Performance RNN. For purposes of exemplifying the use of the model, Huang et al. (2019) provides a jupyter notebook on the Google Colab platform. In this notebook, it can be identified that the model weights are downloaded through the **gsutil** tool, which enables access to Google's internal Cloud storage. The tool is installed by default in the virtual environments created by Colab. Therefore, two ways of accessing the model weights were identified, namely:

- Using **gsutil** on Google Colab followed by manual download. In this approach, the Colab environment is used to download the weights (Colab downloads from Cloud Storage). After downloading the weights in the Colab virtual environment, you can download the weights to the desired environment. (Client withdraws from Colab).

- Use of the **Google Cloud CLI** tool in the desired environment. In this way, the desired environment has direct access to Cloud Storage, without using the Colab platform.

The approach adopted was to use **gsutil** directly in the desired environment without the intermediary of Colab, through the **Google Cloud CLI** tool. To facilitate the automation of the process, the **docker** tool was used together with the official **Google Cloud CLI** image to download the weights of the model. Scripts for automating this process were developed to facilitate reproducibility.

Also similar to the Performance RNN model structure, Music Transformers also comes tied to a set of dependencies from other Python libraries. Therefore, the strategy of creating an adequate virtual environment was also applied in this case, using **conda** technology. However, Music Transformers' **tensor2tensor** dependency was "broken", meaning that a piece of code needed to run the model was not working. After investigations into the reason for the problem, it was identified that the **gym_utils.py** file of the **tensor2tensor** library has an initialization problem in one of the parameters of its functions, where the **kwargs** parameter of the ***register_gym_env*** function was initialized with the value **None**. After removing this behavior, integration with the Music Transformers source code can be successful.

After adapting the dependencies, scripts for using and manipulating the model were developed to generate MIDI files. We can finally summarize the framework for music generation using Music Transformers in the following steps:

- Obtaining model weights through automation scripts, using **docker** and **Google Cloud CLI**.
- Obtaining model source code from the Magenta repository on GitHub.
- Creating a virtual environment for dependency resolution and reproducibility.
- Suitability of dependencies (**tensor2tensor**) for functional integration with the model.
- Use of custom scripts to manipulate the model for MIDI generation.

2.2.3 Framework for music generation for the MuseNet model

The source code and weights of the MuseNet model were not made publicly available by OpenAI, with all information pertinent to the development of the model limited to the article published on the company's blog by Payne (2019).

However, the published article provides an interface for interacting with the model, thus enabling musical generation with the model via the web. It is worth mentioning that the interface has some parameters to guide the musical generation, such as **Style** (composition style), **Intro** (input musical excerpt for conditional generation), and **Instruments**. Aiming to seek homogeneity with the other models used, the **Style** parameter was left with its default value **Chopin** (classical pianist from the romantic period), **Intro** was selected with the **None option (Start from Scratch)** for unconditional generation of tokens, and the parameter **Instruments** has been limited to **Piano**. After adjusting model parameters, tokens were interactively generated until a result lasting 30 seconds was obtained. After the musical generation, the interface offers the download of the created musical excerpt. This whole process can be summarized in the following steps:

- Access to the web interface of the MuseNet model provided by OpenAI.
- Model parameters adjustments.
- Generation of tokens in an iterative way, aiming to complete 30 seconds of music.
- Download the musical excerpt created by the model.

2.2.4 Training framework and music generation for the Custom GRU and Custom GPT2 model

The **Custom GRU** and **Custom GPT2** models were completely developed and trained in this work. These models used the result of data pre-processing as a dataset, detailed in section **2.1.2.3**.

It is worth noting that the **Custom GPT2** model failed to achieve interesting results to be included in the blind test, due to hardware limitations in view of the complexity of the model.

This section seeks to detail the entire "pipeline" used for training music generation models.

2.2.4.1. Custom GRU

The **Custom GRU** model was developed seeking to be an alternative to the **PerformanceRNN** model, bringing an architecture also based on recurrent neural networks. Unlike the **PerformanceRNN** model which is based on layers of Long-Short Term memory (LSTM, (Yu et al. 2019)), the GRU model is based on layers of Gated Recurrent Unit (Gated Recurrent Unit, (Cho et al. 2014)). The development of the model started from the work of Lee (2020), being built using the Python programming language combined with the PyTorch framework (Paszke et al. 2019).

Below is the table of layers used in the model, followed by their respective quantities of parameters:

Table 2: List of layers and parameters of neural networks in the Custom GRU model

Neural Network layer	Number of parameters / Type of activation function
Linear	50.688
Activation	Hyperbolic tangent
Embedding	57.600
Linear	136.192
Activation	Leaky rectified linear unit (Leaky ReLU)
3x Gated Recurrent Unit (GRU)	4.727.808 (3 * 1.575.936)
Linear	368.880
Activation	Softmax

The model has 5,232,880 parameters in its entirety. All linear and Embedding layers are initialized according to the method described by Glorot and Bengio (2010), using a normal distribution. Therefore, the initial weights of the layers mentioned above are the result of the sampled values of N(0, std^2) where:

$$std = gain * \sqrt{\frac{2}{fan_{in} + fan_{out}}} \tag{1}$$

where **gain**: optional scaling factor; fan_{in}: number of layer inputs and fan_{out}: number of layer outputs. On all initializations, **gain** assumes its default value of 1 (resulting in tensors not scaling). The model architecture starts with a linear layer of 32 inputs and 1536 outputs, defined by

$$y = xW^T + b \tag{2}$$

where x: layer input tensor; **W** $^{\mathbf{T}}$: layer weight transposed matrix and **b:** bias matrix.

The first linear layer is followed by an activation function of the hyperbolic tangent type, defined by:

$$tanh = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3}$$

where x: output tensor of the previous layer.

The result of this activation function is then passed to an Embedding layer of dimensionality [240 x 240]. The output of this layer is passed to another Linear layer, followed by a Leaky Rectified Linear Unit (Leaky ReLU) activation function, defined by:

$$LeakyRelu(x) = (0,x) + slope * (0,x) \tag{4}$$

where x**:** output tensor of the previous layer and $slope$**:** negative slope. The negative slope value used in this architecture was 0.1.

Then the model follows with 3 layers of gated recurrent units (Cho et al. 2014), where each layer computes the following function:

$$\begin{aligned} r_t &= \sigma\left(W_{ir} x_t + b_{ir} + W_{hr} h_{(t-1)} + b_{hr}\right) \\ z_t &= \sigma\left(W_{iz} x_t + b_{iz} + W_{hz} h_{(t-1)} + b_{hz}\right) \\ n_t &= tanh\left(W_{in} x_t + b_{in} + r_t * \left(W_{hn} h_{(t-1)} + b_{hn}\right)\right) \\ h_t &= (1 - z_t) * n_t + z_t * h_{(t-1)} \end{aligned} \tag{5}$$

where r_t: reset port; σ: sigmoid function; **W**: matrix of weights; **b**: bias matrix; x_t: entry at time *t* ; $h_{(t-1)}$: hidden state of the layer at time t-1 or the initial hidden state at time 0; z_t**:** update port; n_t: new state port; *: Hadamard product and h_t: hidden state at time *t*;

Each gated recurrent unit layer has a total of 1,575,936 parameters, which are the sum of the layer's hidden weights and biases. Finally, the model ends with a last linear layer followed by a SoftMax activation function to obtain the probability distribution of possible outcomes, defined by the formula:

$$S(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \tag{6}$$

where x: previous layer output tensor.

The loss function is defined with the cross-entropy calculation. The network backpropagation process is calculated with the Adam optimizer (Kingma and Ba, 2014). It is worth noting that gradient clip techniques were applied in the backpropagation process in an attempt to circumvent any explosive gradients. The technique basically consists of normalizing the gradient tensor in a range of -1 and 1 before performing the optimization with Adam.

The model was trained for 18 hours on two 24GB NVIDIA GeForce RTX 3090 GPUs with the following hyper-parameters:

Table 2: List of hyper-parameters for training the Custom GRU model

Hyperparameter	Value
Learning rate	0.001
Batch size	64
Window size	200
Stride	10
Transposition use	False
Control relationship	1
Teacher forcing proportion	1

The model managed to reach the loss function value of 2.23 in the validation data. For token generation, the model output is stochastically sampled using a beam search algorithm while the teacher forcing algorithm is applied to training.

2.2.4.2. Custom GPT2

The **Custom GPT2** model was developed trying to be an alternative to the **Music Transformers** and **MuseNet** models. This model starts from the implementation of a GPT2 causal language model (Radford et al., 2019) offered by the transformers library (Wolf et al. 2019) through the Hugging Face platform.

2.2.4.2.1 Tokenizer

A “tokenizer” was trained to generate the vocabulary of the problem, using the pre-processed dataset of textual sequences as a base. The original tokenization training strategy of the GPT2 model (Radford et al. 2019) is based on the Byte Pair Encoding (BPE) technique, due to the extensive size of the worked vocabulary (50,257 tokens). Taking into account that the data pre-processing strategy adopted in this work results in a relatively small vocabulary (390 tokens), the choice of the Word Level training technique was chosen for the tokenizer, which directly relates an identification number to each vocabulary token, thus making an event-to-token mapping. The **HuggingFace tokenizers** library was chosen to carry out this task due to its ease of manipulation and high performance.

2.2.4.2.2 Inputs and model

The GPT2 model must receive a pre-defined tensor dimensionality for consistency of the matrix calculation in the input layer. The original implementation by Radford et al. (2019) uses sequences of 1024 tokens in “batches” of 512 sequences (input of [1024 x 512]).

Unfortunately, those values could not be replicated due to a hardware limitation. Therefore, to adapt the model to the available hardware resources, sequences of 50 tokens in batches of 64 sequences were chosen (input of [64 x 64]). Sequences having more than 64 tokens were therefore truncated into multiple sequences less than or equal to 64 tokens, e.g. a sequence of 200 tokens [200 x 1] is transformed into four sequences of 64 [64 x 4]. It is worth noting in this example that the last sequence would initially have 8 tokens (200 = 64 + 64 + 64 + 8).

However, as mentioned earlier, the input tensor needs to have consistent dimensionality. Because of this, strings of tokens smaller than 64 will have their difference in "PAD" tokens. "PAD" tokens are ignored by the model. So, in the previous example, the last sequence of 8 tokens is extended with 56 "PAD" tokens, thus adapting the example tensor of [200 x 1] to a tensor of [64 x 4].

2.2.4.2.2 Training

The Custom GPT2 model managed to achieve a loss function value of 2.50 on the validation data, being trained for 116 hours on two 24GB NVIDIA GeForce RTX 3090 GPUs with the following hyper-parameters:

Table 3: List of hyper-parameters for training the Custom GPT2 model

Hyperparameter	Value
Learning rate	0.0005
Batch size	64
Learning rate Scheduler	Cosin
Weight decay	0.1
Gradient accumulation steps	8
Number of attention heads for each attention layer in the Transformer encoder.	8
Number of hidden layers in the Transformer encoder.	6
Embedding and hidden states dimensionality	512

2.2.4.2.3 Detokenizer and Decoder

After training, the model was used to generate musical language. Given one or more tokens in sequence and a maximum of **M** tokens to be generated, the GPT2 model outputs a sequence of dimensionality tokens [1 x M] in an autoregressive manner. The model outputs were stochastically sampled using the Beam Search technique, thus replicating the same strategy used in the Performance RNN, Music Transformers and Custom GRU models. From this point on, the data reverse engineering process can begin. The tokenizer takes place in this step, but this time acting in the mapping of tokens to events, which will consequently be processed by reverse engineering the techniques used in the pre-processing, obtaining a MIDI file as a result. However, the model was not able to converge in results interesting enough for the research, probably due to the complexity of the model in front of the limitation of available hardware.

2.3 Blind test

The blind test was developed in three languages (Portuguese, English and French), being distributed through the SurveyMonkey online platform through three web forms (one for each language). For the blind test, three types of populations were initially selected, namely:

- People who don't listen to classical music very often
- People who listen to classical music often
- Classically trained musicians

The criterion adopted for choosing the aforementioned population was based on the hypothesis that musicians with a classical background and people who listen to classical music frequently have, respectively, a greater ability to distinguish the compositional origin of a piece of music. The purpose of this criterion is to add details for the final analysis of the research.

All subjects had a total of five (number of generative models + 1) musical excerpts to be heard, one of them (+ 1) derived from a composition written by a human, randomly selected within the data used for training. A musical excerpt was collected from each model, without any preconditioning. The musical excerpts were the same for the entire tested population, being initially ordered arbitrarily.

The following five questions were proposed for each musical excerpt:

- On a scale of 0 to 10, how musical do you think the piece of music heard is?
  - Options from 0 to 10

- Do you believe this song was made by a human or a computer?
  - Binary Response (Human or Computer)

- Would you say that the musical excerpt heard resembles classical music?
  - Binary Response (Yes or No)

- (Optional) Add any thoughts you might have about the excerpt heard
  - Free text

After the end of the evaluations of the musical excerpts, the participants were directed to the following set of questions related to their musical experiences:

- How would you rate your musical sensitivity?
  - Options from 0 to 10.

- How often do you listen to classical music?
    - I don't listen to classical music
    - Sometimes
    - Often

- Do you have any practical experience with music?
    - No
    - Yes, I can make music with my voice or a musical instrument
    - Yes, I study/I studied music theory (conservatory, music school, self-taught)
    - Yes, I study/studied at a music university

The test consisted of a total of 23 questions, 20 (5 musical excerpts x 4) questions related to the musical excerpts heard and 3 questions related to the musical experience.

3. Results and discussion

The blind test was performed by a total of 117 participants from different places of the world, including South America (Brazil), North America (USA and Canada) and Europe (France and Germany). The tested population is composed of professional musicians, music students at different levels (from graduation students to technical students) and non-musicians. They were evaluated together through 3 different spectra, namely:

- Classical music consumption frequency
- Musical experience
- Musical sensitivity

The evaluation metric for all populations was based on the success rate of questions 2, 6, 10, 14 and 18, which consists of identifying the compositional origin of the excerpt heard. In other words, the metric was based on the average number of correct answers for the questions “Do you believe this song was made by a human or a computer?”.

3.1 - Classical music consumption frequency

The question 22 of the blind test features the question “How often do you listen to classical music?” with 3 possible answers, including “I don't listen to classical music”, “Occasionally”, “I listen to it often”. Based on the answer to this question, the population was divided into three groups, including:

- Non-classical music listeners (31%)
- Casual classical music listeners (15%)
- Frequent listeners of classical music (54%)

The success rate of each group can be illustrated in the following graph:

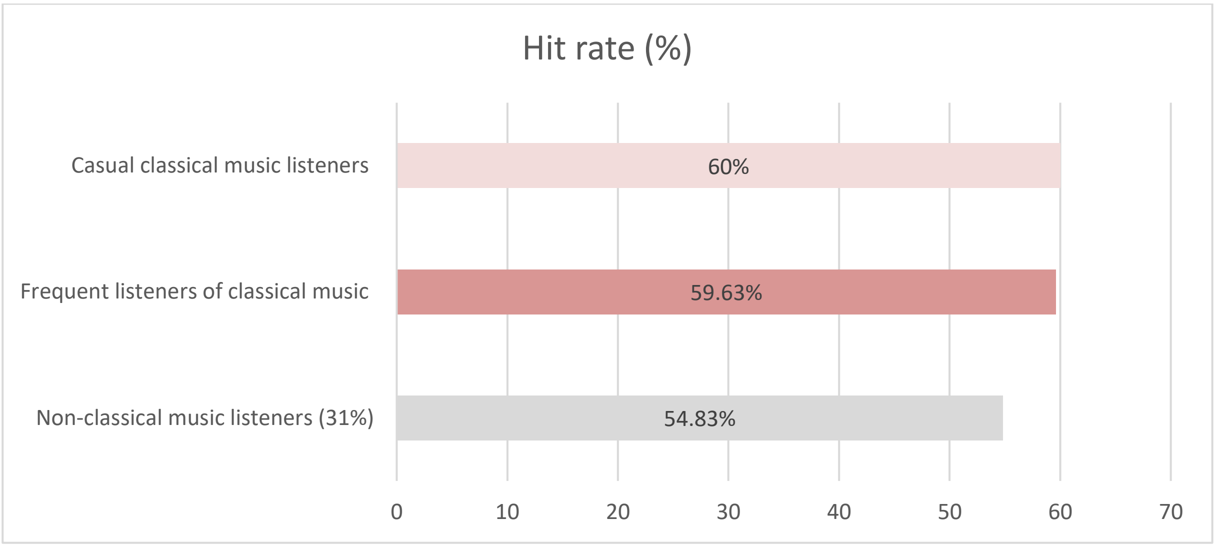


Figure 2. Population hit rate averages across the frequency spectrum of consumption of classical music.

It can be seen from Figure 2 that casual and frequent listeners of classical music were more assertive in identifying the compositional origin of the musical excerpt heard than non-classical music listeners.

3.2 – Music Experience

In this evaluation, the population was grouped by the spectrum of musical experience, based on the answers to question 23 of the blind test “Do you have any practical experience with music?”. The population was divided into the following groups:

- People with no music experience
- Musicians
- Musicians with knowledge of music theory
- Musicians with advanced knowledge of music theory

The average hit rate for each group can be seen in Figure 3:

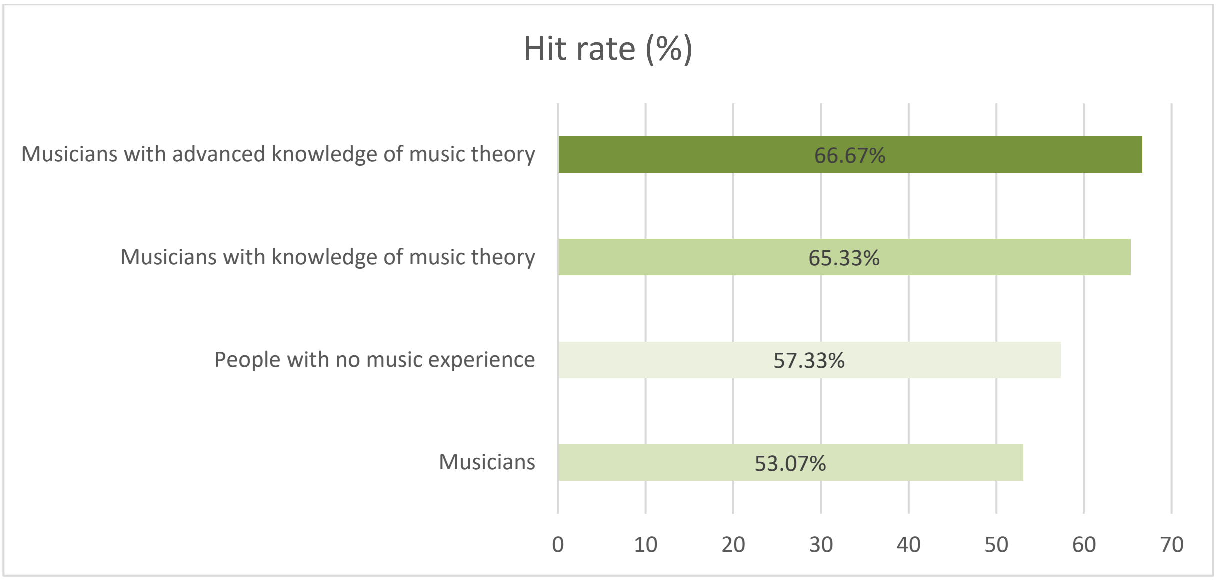


Figure 3. Population hit rate averages across musical experience spectrum.

It can be observed in Figure 3 that musicians with knowledge of music theory were more assertive in identifying the compositional origin of the musical passage heard than people with no experience in music theory.

3.3 – Musical sensitivity

The assessment of musical sensitivity was based on the answers to question 21 of the blind test: "How would you rate your musical sensitivity?" where each test subject makes a self-assessment of their musical sensitivity on a scale from 0 to 10, where 0 was described as "I cannot distinguish musical elements when I listen to music", 5 as "I can distinguish certain musical elements, such as voice and instruments, when listening to music" and 10 as "I can perceive details of harmonic and melodic structure when I listen to music".

The population was grouped by the following filters:

- People with high musical sensitivity (self-assessment > 7)
- People with fair musical sensitivity (3 < self-assessment <= 7)
- People with low musical sensitivity (self-assessment <= 3)

The average hit rate for each group can be seen below in Figure 4:

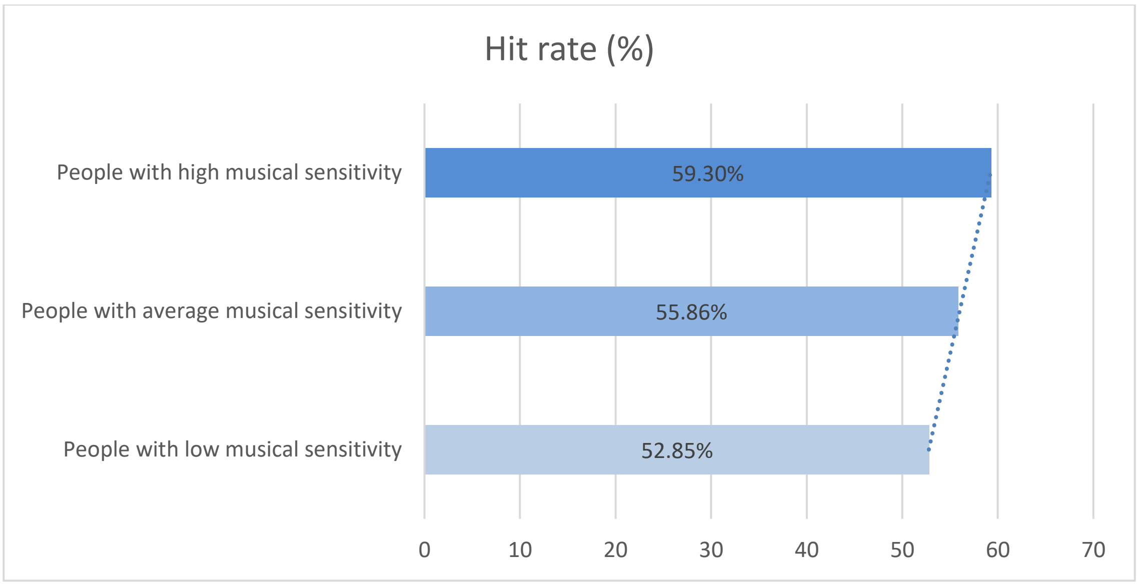


Figure 4. Population hit rate averages across the musical sensitivity spectrum.

As illustrated by Figure 4, there is a linear relationship between musical sensitivity and the hit rate of the tested population, thus demonstrating that within the population tested, people with greater musical sensitivity were more assertive in identifying the excerpts heard.

3.4 – Model Comparison

Data on the approval and evaluation of the musical excerpts heard also was collected during the analysis of each excerpt, which brings interesting insights. One of the perspectives to evaluate the musical quality of each passage is to look at the data by calculating the averages grouped by model, referring to the questions 1, 5, 9, 13, 17 (“On a scale of 0 to 10, how musical do you think this song is?”).

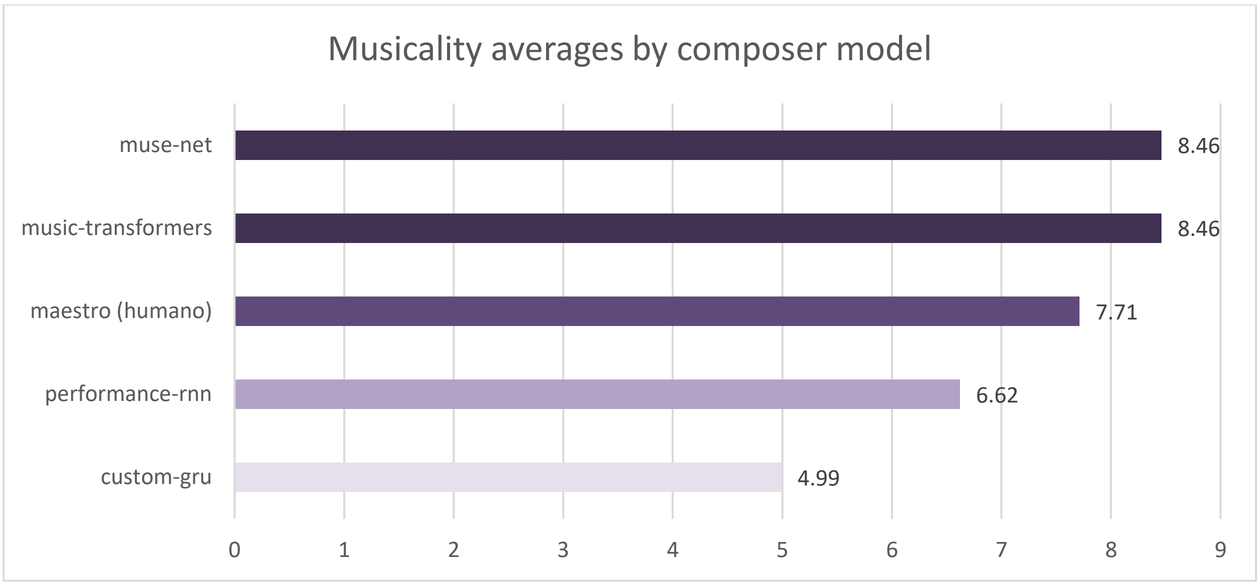


Figure 5. Musicality averages by composer model.

It is interesting to observe in Figure 5 that, within the population tested, that the **MuseNet** and **Music Transformers** models were better evaluated than the musical excerpt of human origin, randomly extracted from the MAESTRO dataset. It is also interesting to note that the models based on RNN (**Performance RNN** and **Custom GRU**) were evaluated as less musical than those of human origin. Another perspective that is worth analyzing starts with the evaluation of the compositional origin by composer model, where we can observe the judgment of the tested population about the origin of the musical excerpt heard.

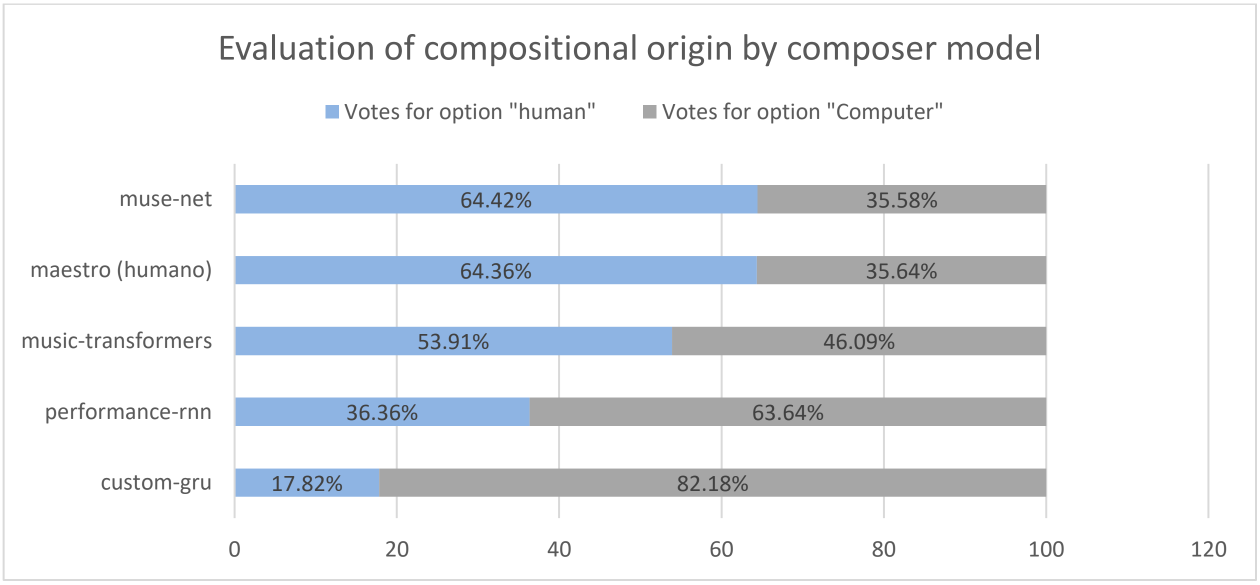


Figure 6. Evaluation of compositional origin by composer model.

As represented in Figure 6, the **MuseNet** model stands out once again, and by a slight margin of difference, the model is interpreted as human more often than the genuinely human one. It is also worth highlighting that the musical excerpts derived from Transformer architectures (**Music Transformer** and **MuseNet**) were able to confuse more than half of the population, which in turn was able to assertively distinguish the compositional origin of models based on RNN (**PerformanceRNN** and **Custom GRU**)

We can approach as a final point of view the approval of musical excerpts as classical music grouped by model, that is, how many people voted “Yes” or “No” for questions like: “Would you say that the musical excerpt heard resembles music of the genre classic?".

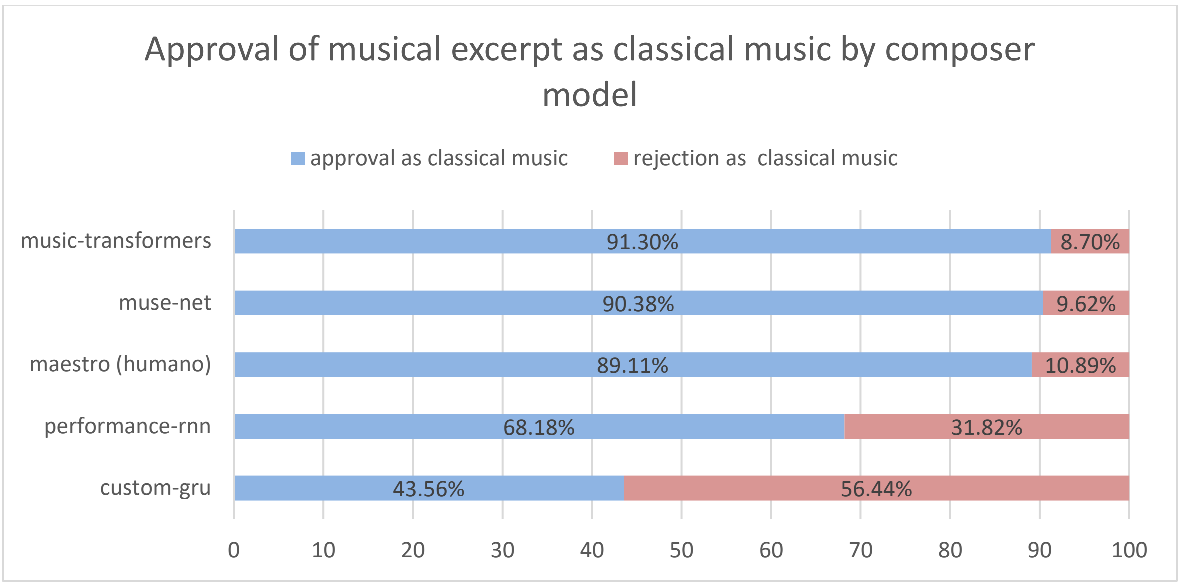


Figure 7. Approval of a musical excerpt as classical music by composer model

The results of the blind test show that the musical quality of the models based on Transformers architectures, for the most part, surpassed results of music of human origin. The results also demonstrate that models based on RNN architectures failed to achieve results that surpassed the results of musicality of human origin. Furthermore, it can be seen that within the tested population, people with more experience in music theory, people who listen to classical music and people with greater musical sensitivity stood out in identifying when the musical excerpts were derived from an AI model and when they were derived from humans, thus corroborating the idea that people with more musical familiarity have a more assertive perception of the intrinsic details that differ a song composed by a machine and a song composed by a human.

4. Conclusion(s) or Final Considerations

This research presented the evaluation of symbolic music generative models from a human perspective, through a blind test available online. This work also details possible pipeline development approaches for music generation through AI, in addition to demonstrating how to use transfer learning to use pre-trained models.

In addition, the results of the blind test brought interesting insights into the musicality of the tested models, demonstrating that the musical excerpts derived from models based on Transformers were the best evaluated, consistently surpassing, from different perspectives, results of human musical excerpts. "Insights" about the population were also presented, corroborating the understanding that within the tested population, people with more experience in music theory, people who listen to classical music and people with greater musical sensitivity were more able to correctly distinguish whether the musical excerpt was composed by a human or an AI.

5. Acknowledgments

I would like to thank everyone who participated, directly or indirectly, in the development of this research work, enriching my learning experience. I express here my deep gratitude for all the support.

References

Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G.S.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S. 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*. Available at: https://www.tensorflow.org/. Accessed on: April 2, 2022.

Adorno, T.W.; Gillespie, S. 1993. Music, language, and composition. *The Musical Quarterly*, *77*(3), pp.401-414.

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

Brunner, G.; Konrad, A.; Wang, Y.; Wattenhofer, R. 2018. MIDI-VAE: Modeling dynamics and instrumentation of music with applications to style transfer. *arXiv preprint arXiv:1809.07600*.

Cho, K.; Van Merriënboer, B.; Bahdanau, D. e Bengio, Y. 2014. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*.

Chollet, F. 2015. Keras. Available at: https://github.com/keras-team/keras. Accessed on: April 2, 2022.

Dong, H.W.; Hsiao, W.Y.; Yang, L.C.; Yang, Y.H. 2018. Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment. In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 32, No. 1).

Ens, J. e Pasquier, P., 2020. Mmm: Exploring conditional multi-track music generation with the transformer. *arXiv preprint arXiv:2008.06048*.

Friberg, A. e Sundberg, J., 1993. Perception of just-noticeable time displacement of a tone presented in a metrical sequence at different tempos. *The Journal of The Acoustical Society of America*, *94*(3), pp.1859-1859.

Gardner, M.W.; Dorling, S.R. 1998. Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. *Atmospheric environment*, *32*(14-15), pp.2627-2636.

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

Glorot, X. e Bengio, Y. 2010. Understanding the difficulty of training deep feedforward neural networks. In Proceedings of the thirteenth international conference on artificial intelligence and statistics (pp. 249-256). JMLR Workshop and Conference Proceedings.

Hawthorne, C.; Stasyuk, A.; Roberts, A.; Simon, I.; Huang, C.Z.A.; Dieleman, S.; Elsen, E.; Engel, J.; Eck, D. 2018. Enabling factorized piano music modeling and generation with the MAESTRO dataset. *arXiv preprint arXiv:1810.12247*.

Huang, C.Z.A.; Vaswani, A.; Uszkoreit, J.; Shazeer, N.; Simon, I.; Hawthorne, C.; Dai, A.M.; Hoffman, M.D.; Dinculescu, M.; Eck, D. 2018. Music transformer. *arXiv preprint arXiv:1809.04281*.

Huang, C.Z.A.; Vaswani, A.; Uszkoreit, J.; Shazeer, N.; Simon, I.; Hawthorne, C.; Dai, A.M.; Hoffman, M.D.; Dinculescu, M.; Eck, D. 2019. Music Transformer: Generating Music with Long-Term Structure. Available at: https://magenta.tensorflow.org/music-transformer. Accessed on: August 12, 2022.

Ji, S., Luo, J. and Yang, X., 2020. A comprehensive survey on deep music generation: Multi-level representations, algorithms, evaluations, and future directions. *arXiv preprint arXiv:2011.06801*.

Kingma, D. P. e Ba, J. 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

Lee, Y.; 2020. PyTorch implementation of Performance RNN, inspired by Ian Simon and Sageev Oore. "Performance RNN: Generating Music with Expressive Timing and Dynamics." Magenta Blog, 2017. Available at: https://github.com/djosix/Performance-RNN-PyTorch. Accessed on: August 12, 2022.

Lhoest, Q.; del Moral, A.; Jernite, Y.; Thakur, A.; von Platen, P.; Patil, S.; Chaumond, J.; Drame, M.; Plu, J.; Tunstall, L.; Davison, J.; Šaško, M.; Chhablani, G.; Malik, B.; Brandeis, S.; Scao, T. , Sanh, V. , Xu, C. , Patry, N. , McMillan-Major, A. , Schmid, P.; Gugger, S. , Delangue, C.; Matussière, T.; Debut, L.; Bekman S.; Cistac, P.; Goehringer, T.; Mustar, V.; Lagunas, F.; Rush, A.; Wolf, T.; Datasets: A Community Library for Natural Language Processing. *arXiv preprint arXiv: 2109.02846.*

Loy, G. 1985. Musicians make a standard: the MIDI phenomenon." Computer Music Journal 9.4 (1985): 8-26.

Mathews, A.; Xie, L.; He, X. 2016. Senticap: Generating image descriptions with sentiments. In *Proceedings of the AAAI conference on artificial intelligence* (Vol. 30, No. 1).

Oore, S., Simon, I., Dieleman, S., Eck, D. and Simonyan, K., 2018. This time with feeling: Learning expressive musical performance. *Neural Computing and Applications*. *arXiv preprint arXiv:1808.03715*

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; Desmaison, A. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, *32*.

Patel, A.D. 2010. *Music, language, and the brain*. Oxford university press.

Payne, C. 2019. Musenet. Available at: https://openai. com/blog/musenet. Accessed on: April 2, 2022.

Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; Sutskever, I. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, *1*(8), p.9.

Simon, I. e Oore, S. 2017. Performance RNN: Generating Music with Expressive Timing and Dynamics. Available at: https://magenta.tensorflow.org/performance-rnn. Accessed on: August 12, 2022.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł. and Polosukhin, I., 2017. Attention is all you need. *Advances in neural information processing systems*, *30*.

Walder, C. 2016. Modeling symbolic music: Beyond the piano roll. *Asian Conference on Machine Learning* (pp. 174-189). PMLR.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M. and Davison, J., 2019. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*.

Yu, Y.; Si, X.; Hu, C.; Zhang, J. 2019. A review of recurrent neural networks: LSTM cells and network architectures. *Neural computation*, *31*(7), pp.1235-1270.

Attachment

Below are some interesting comments collected from the research on the musical excerpts generated by the artificial intelligence models:

“It brings calm to my thoughts, I think of positive things and solutions for something I don't know what it is”
Classical music listener over musical excerpt from the Music Transformers model.

“The rests on lower notes make the melody more harmonious and more sentimental in this case.”
Musician with knowledge in music theory on an excerpt from the Performance RNN model.

“Perfect timing in cross voices, no delay time, so beautiful mechanical polyphony.”
Musician over musical excerpt from the MuseNet model.

“The excerpt is more like a practice of musical scales than a piece of music itself.”
Musician with knowledge in music theory on excerpts from the Custom GRU model.

“It kept the feeling, there was a sequence in the harmony and rhythm, bringing the same feeling to the end.”
Music teacher and professional musician on an excerpt from the Performance RNN model.

“I found it really similar to calm, perfectly sounding classical music.”
Frequent listener of classical music over excerpt from the Music Transformers model.