Genre-Conditioned Music Generation - A3I Project

Federico Cichetti & Marcello Ceresini

Project Overview

- We introduce a Transformer-based model for the task of **multi-track genre-conditioned song generation.**
- The model uses a **compact** and **expressive new representation** motivated by empirical results and literature works.
- We collected a large **new dataset** of **genre-annotated songs** to train our model.
- We implemented a variety of **metrics** to **evaluate the quality** of generated songs.

Background

There are **3 main categories** which influence representation and modeling:

- **Score generation**, which can be sub-categorized in:
  - **Monophonic** or Melody Generation
  - **Polyphonic** (more than one melody)
  - **Multi-track** (more than one instrument, each with its own dynamics)
- **Performance generation**, which is related to the execution of a score
- **Audio generation**, which is related to the synthesis of an audible song

Also, there are many ways to **represent music**, each with their pros and cons.

- **Event-based**, or **MIDI-like**
  - Contains node **beginning**, **end**, **pitch**, **velocity** and **instrument**. Complete, but times are often expressed in seconds.
- Sequence-based
  - Many sequences, each representing an aspect of the notes. No multi-track music and resolution issues.
- Pianoroll
  - 2D binary matrix (time x pitches). Can be processed as an image, but it’s very inefficient.
- Audio (e.g. waveform or spectrogram)
  - Rich representation, but difficult to use.

Datasets

Many datasets have been used in literature, but most of them are **focused on a single genre** or instrument (e.g. **MAESTRO** on piano/**classical** songs, **NES**-MDB on **videogame** songs, ...)

To create a general model, we instead needed to create a **heterogeneous dataset** with **clear genre annotations**. To this end, we:

- Downloaded a version of **LMD** (Lakh Midi Dataset) containing **115K songs** that had been **matched to entries in MSD** (a dataset of **metadata** for 1M+ songs).
    - Metadata include **title**, **artist**, ... but **NOT THE GENRE**.
- Joined the database of LMD matches together with MSD to **retrieve the artist information for each song**.
- For each song, ran a **query** using the **Spotify API** through the spotipy library to collect the **genre** associated to its **artist**
    - **Spotify** only publicly displays **genre information for artists**, not for songs.
    - The script ran for several days on a Raspberry Pi

Dataset Criticalities and Genre Selection

We make some assumptions:

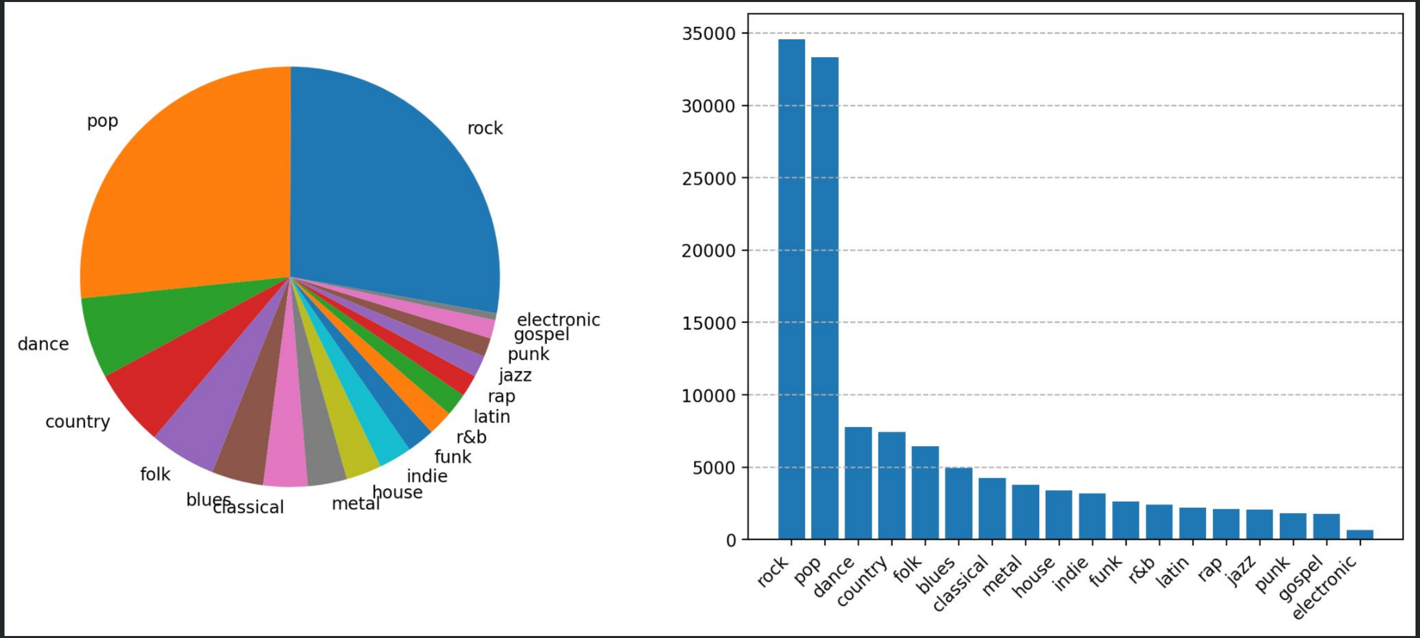
- We **trust the matching between LMD and MSD**
- We **trust Spotify's genre label**
- We **assume** that an **artist** produces **songs of approximately the same genre**

In total **2146 distinct genres** were found, most of them being extremely niche or **variations of the same core genre** (e.g. soft rock, hard rock, album rock, …).

To reduce the number of genres, we **split each label by words** (e.g. all of the above would also be "rock" songs) and hand-selected the **18 most representative genres**.

When creating the **genre labels** (the **vectors** representing the genres for the model) we **did not one-hot encode genres**, as some songs had **multiple genres!** For instance, "pop rock" songs are considered as both "pop" and "rock". Therefore we computed a **"genre distribution"** by normalizing boolean genre vectors (e.g. [pop, rock, …] ➔ [0.5, 0.5, 0… 0]). In this way, we also allow **arbitrary combinations of these genres** in generation.

In total there our dataset has **70,117 genre-labelled songs**, with a noticeable **class imbalance**.

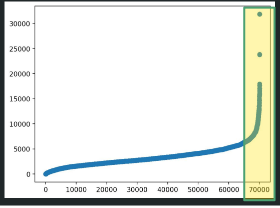
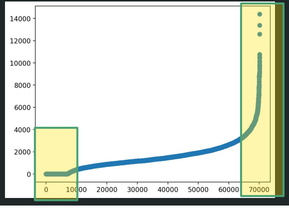
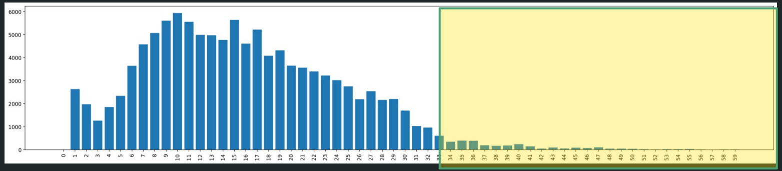
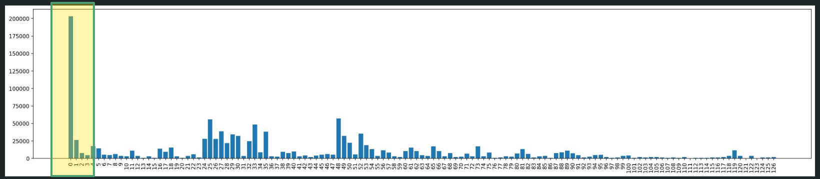
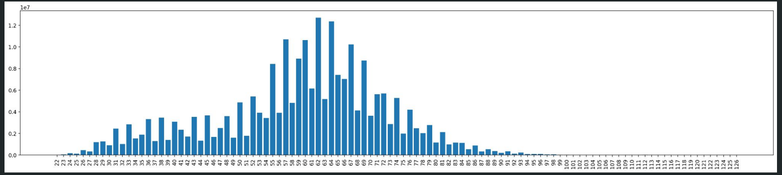


Genres: rock, pop, dance, country, metal, classical, folk, blues, house, indie, latin, jazz, funk, rap, punk, r&b, gospel, electronic.

Dataset Exploration

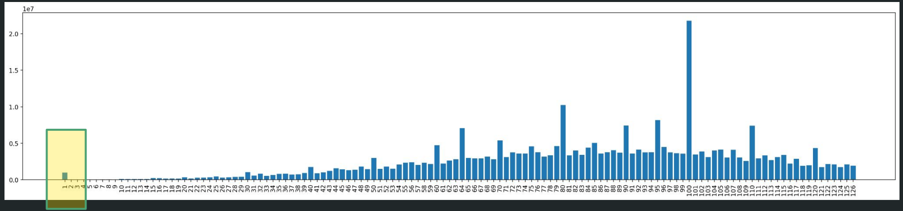
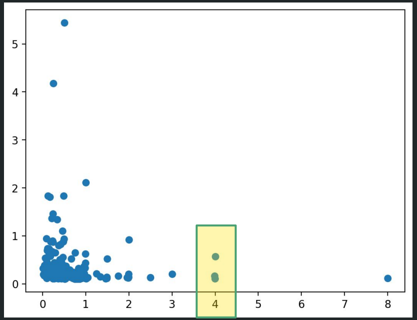
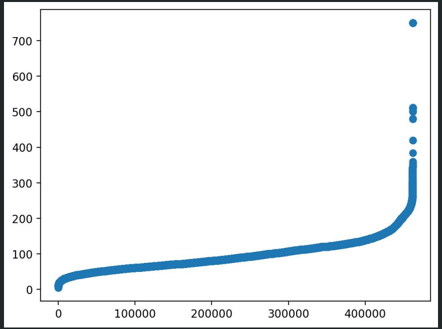
We analysed the dataset extensively: here are some of the most important findings:

- The distribution of the number of notes, of drum notes and of tracks per song all have a long tail
  - Only a small number of songs are too long/too complex.
  - Some have no drums at all
- The instrument frequency is well balanced, with a skew only towards piano
- The pitch distribution seems to follow a normal distribution

Dataset Exploration

- The velocity distribution seems to have an artifact for the lowest value
- Duration of notes contain incoherences possibly due to:
  - Different song resolution
  - Live recording of performance
- The distribution of tempos also has a very long tail:
  - A possible way to cut these is to only keep values that appear in 11% of songs: in this case we have 19 nominators and 8 denominators for the time signature (152 combinations)

Music Representation

Our representation is event-based and takes inspiration by mainly two works:

- MusicBERT, introducing OctupleMIDI as a representation of songs as a sequence of tuples encoding different aspects of notes (pitch, velocity, ...)
- Multitrack Music Transformer, that uses a similar representation with one element of the tuple representing the type of the event.

Our data structure is therefore a sequence of 11-tuples, each with a different possible range of values:

0 (Type)	0-7	4 (Duration)	0-135	8 (Key Sign.)	0-23
1 (Measure)	0-255	5 (Pitch)	0-127	9 (Time Sign.)	0-151
2 (Beat)	0-131	6 (Instr. type)	0-128	10 (Tempo)	0-48
3 (Position)	0-63	7 (Velocity)	0-127		

Music Representation: Types

We define 8 event types:

0. Start of song
    - First token
1. Instrument Definition
    - We define all instruments before starting to create note events to influence the generation space
2. Start of Notes
    - Separates instruments definition from notes definition
3. Note
    - All other parts of the tuple are used to create a note event
4. Change in Key Signature
5. Change in Time Signature
6. Change in Tempo
7. End of Song
    - Also used as padding

} Require that the 8th, 9th or 10th part of the tuple define a new Key, Time or Tempo to be used from that point onwards. We made this event more explicit to ensure that the model treats it as an important element

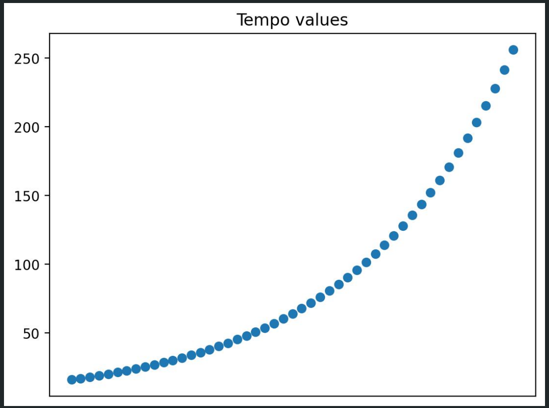
Music Representation: Ranges

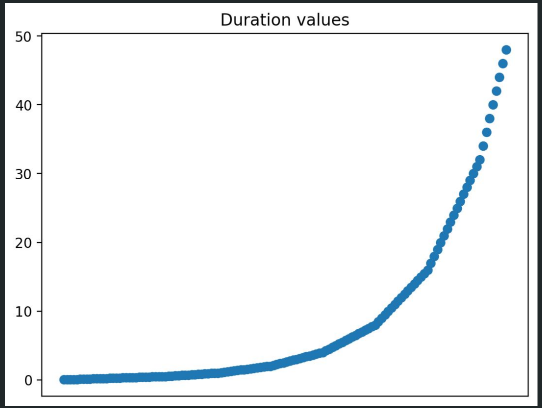
In light of the dataset analysis, we tried to minimise the ranges of values as much as possible while keeping those ranges expressive.

- We allow at most 255 measures (1.28% songs in our dataset are cut)
- Time signature allows numerators [1,..,18,132] (so the maximum beat is 132) and denominators [1,2,4,8,16,32,64,128].
- Position is expressed 1/64 of beats, so we have the range [0,63]
- Pitch and velocity are in the range [0, 127] like in MIDI. The same is true for instrument types, with an additional instrument for drums ([0,...128])

Music Representation: Tempo and Duration Ranges

- For tempo and durations we wanted to allow large values while not sacrificing precision: therefore we use hierarchical structures, where lower durations/tempos are separated by shorter intervals and viceversa.
  - The 136 durations cover a wide range of possibilities from 1/64 of a beat to 48 beats
  - The 49 tempos follow a geometric progression from 16 to 256 qpm
  - Values in the dataset were discretized to the closest allowed tempo/duration with minimal loss.





Song Example

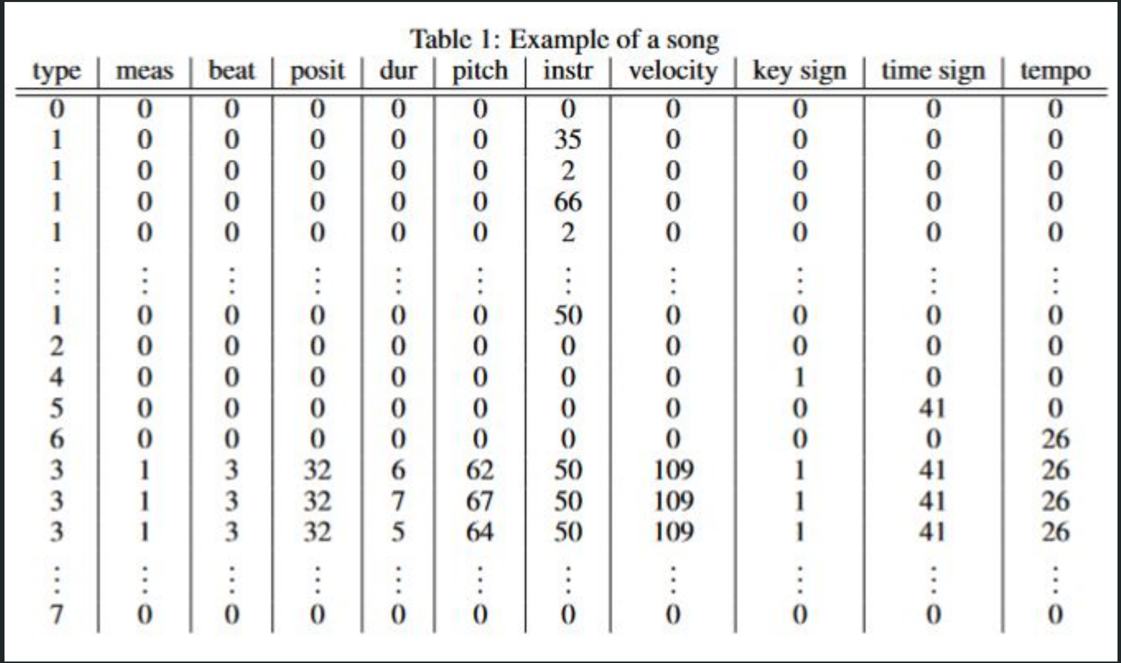


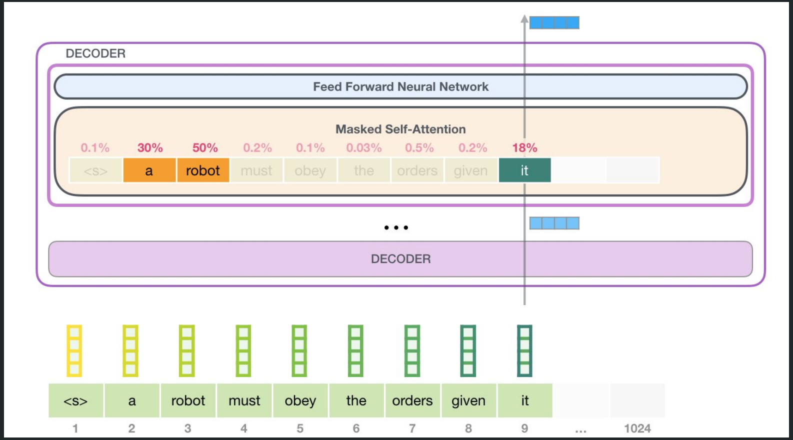
Table 1: Example of a song

type	meas	beat	posit	dur	pitch	instr	velocity	key sign	time sign	tempo
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	35	0	0	0	0
1	0	0	0	0	0	2	0	0	0	0
1	0	0	0	0	0	66	0	0	0	0
1	0	0	0	0	0	2	0	0	0	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
1	0	0	0	0	0	50	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	1	0	0
5	0	0	0	0	0	0	0	0	41	0
6	0	0	0	0	0	0	0	0	0	26
3	1	3	32	6	62	50	109	1	41	26
3	1	3	32	7	67	50	109	1	41	26
3	1	3	32	5	64	50	109	1	41	26
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
7	0	0	0	0	0	0	0	0	0	0

Model Architecture

Motivated by their good results in NLP, we decided to use a **Transformer Decoder** (**GPT-2**) as our core architecture

- The model works **in parallel** on **sequences of tokens**.
- For each token, the model is able to appropriately evaluate the influence of past tokens through the mechanism of **self-attention**.
- The product is a **context embedding**, used to **condition the generation of the next token.**

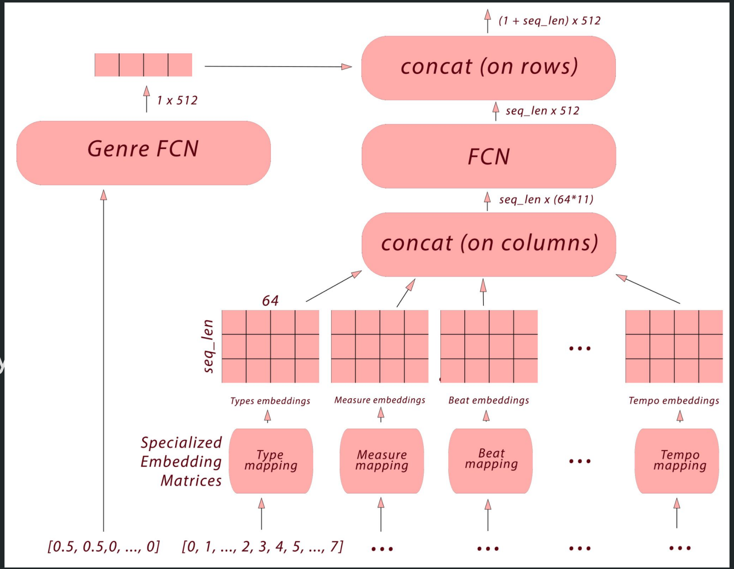


- A sequence of “**heads**” (**FC layers**) can then be used to predict the **probability distributions** of each of the **11 components of the next token** (e.g. using softmax activation) given the context.
- Finally, we can compare the **next GT tuple** with our predicted distribution (e.g. with **categorical cross-entropy**) and generate a feedback signal.

Input Processing & Genre Conditioning

To transform our sequence of tuples into **“input tokens”**, we:

- **One-hot encode each element of the tuple** according to their ranges and **transform them into embeddings** using **11 different embedding matrices** learnt together with the model
  - Output: 11 seq_len x 64 tensors
- **Concatenate** the embeddings **and reduce** the dimensionality to a fixed one with a Dense layer
  - seq_len x 704 ➔ seq_len x 512
- To this we **sum positional embeddings** (sinusoidal embeddings, as proposed in the original paper)



For **Genre Conditioning** we take inspiration from **Ctrl**, a NLP model where **text generation** is **conditioned on a specific style** just by **adding as first token** of the input sequence **a “control token”** (e.g. “horror”, “review”, …). **Self-attention** ensures that the **control token** becomes **part of the context** in a meaningful way.

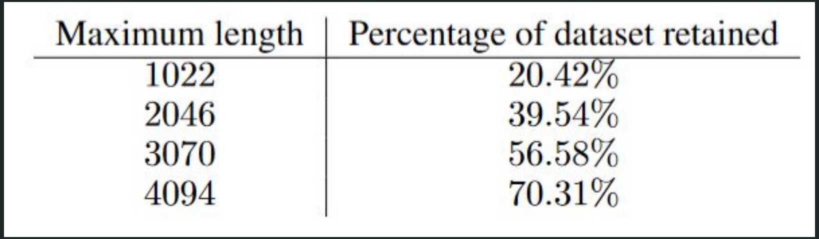
In our case, we add a “**genre token**” obtained by passing the **GT genre vector into a FCN** with the same dimensionality as the other tokens, and **appending it** as first token of the sequence.

Max Length Analysis

Especially when using Transformers, it’s important to study the impact of the maximum sequence length, since the attention mechanism requires quadratic memory wrt sequence length.

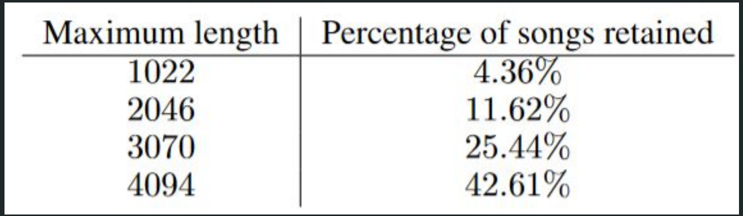
We want our sequences to be as short as possible without throwing away too much information. We explore the effect of discarding long songs or cutting them to a maximum length below.

We eventually chose to cut long songs to 2046 tokens (adding the genre token at the beginning and an end token at the end, our total sequence length becomes 2048).



Maximum length	Percentage of dataset retained
1022	20.42%
2046	39.54%
3070	56.58%
4094	70.31%

Cutting songs



Maximum length	Percentage of songs retained
1022	4.36%
2046	11.62%
3070	25.44%
4094	42.61%

Discarding songs

Model Variants: Regularization

We tried to **guide the model** towards following some of the **syntactical rules of our representation** adding some **additional penalties** when they're broken. The sum of these penalties is added to the loss (x10^{-6})

First output token does not have **type 0**	+4
Second output token does not have **type 1**	+4
Output sequence does not contain **all types at least once**	+(num missing types)^2
From last output type 1 token there should be the following pattern: **[2, 4, 5, 6, 3]**	+1 for each wrong element in the pattern
Mapping output types to [0, 1, 2, 3, 3, 3, 3, 4], the sequence of types must be **monotonic**.	+1 if difference between the type of a token and the following is < 0 or > 1
The sequence of **absolute times of events** (measure * max_beat * max_position + beat * max_position + position) must be **monotonic**	+1 for each token that follows another in the sequence but having lower time measurement
Output notes played by **undefined instruments**	+1 for each undefined instrument in notes
Instruments defined more than once	+1 for each duplicated instrument

Model Variants: Double Head & Masking

A **criticality** of our model is that **all heads are independent from each other given the context.**

Instead, we wanted to allow a **level of communication between the heads** before producing final probability distributions, so that the output is more informed.

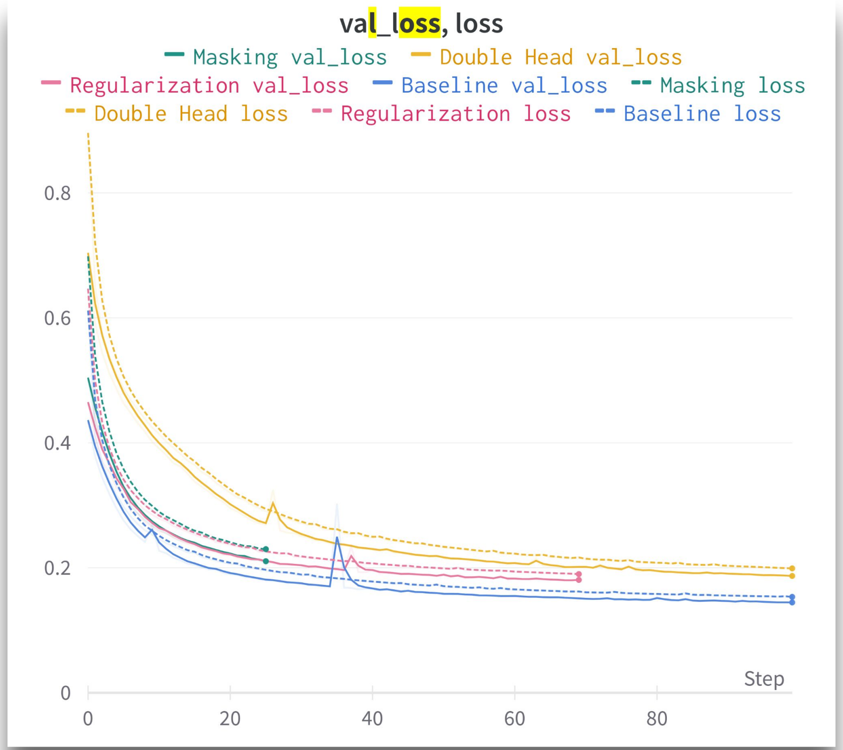
- We **concatenate the output scores** and use a **small bottleneck NN** to **mix them** together
- We apply **another set of heads** on the output embedding and let it produce the final distribution
    - Note: we also could have used a **skip connection.**

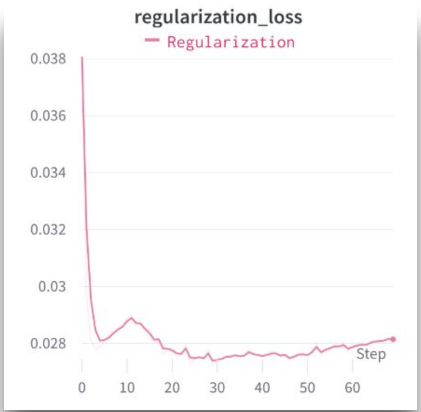
In these experiments we compute a **loss on both head sets**, but on the first set it is weighted 100 times less.

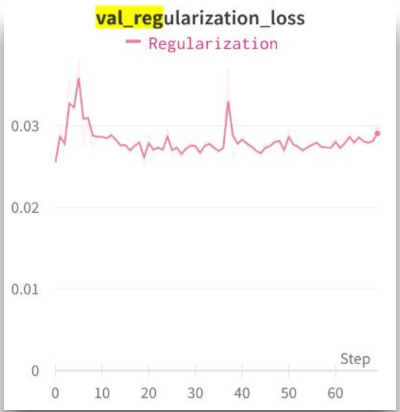
We also tried to implement a **masking mechanism** that injected syntactical knowledge into the model by **blocking some of the possible outputs** during training according to some of the rules described in the regularization slide.

We found this mechanism really **hard to implement** within the framework of Tensorflow, but believe that it could be the source of future work.

Model Variants: Results







Song Generation

We wrote a script to generate full songs given pre-trained weights for the model and a batch of genres. We produce a genre token followed by a 0-type token, feed them to the model and generate the rest of the song autoregressively.

We guide the generation by 0-ing out the probabilities for values that would break our representation semantics and redistributing the remaining mass.

- We keep track of generated types to only allow non-sequence-breaking types
- The type component is sampled first, then we decide which other constraints to pose:
  - Type 2, 7 → We mask every other component to only allow sampling a 0
  - Type 1 → We only allow instruments that were not defined previously and force all other components to 0
  - Type 3 → We mask measure, beat and position distributions to prevent temporal breaking, we mask the instruments that were not defined and we only allow the tempo, time and key signs of the previous event
  - Type 4, 5, 6 → We mask the tempo, key or time sign of the previous event and force all other components to 0

Song Generation - Sampling

We allow several options for sampling.

Firstly, all probability distributions can be modified with temperature T to control the entropy

Then, we implement:

- A “standard sampling”, which simply uses the weights coming from softmax
- “Top-k sampling”, which redistributes the probability mass among the k best-scoring elements of each component
- “Top-p sampling”, which redistributes the probability mass among the best-scoring values up to a cumulative probability sum of p
- “Decreasing top-p sampling”, where p decreases during the sequence generation, so that the beginning of the song is more random while later parts use the established song patterns more.

Evaluation Metrics

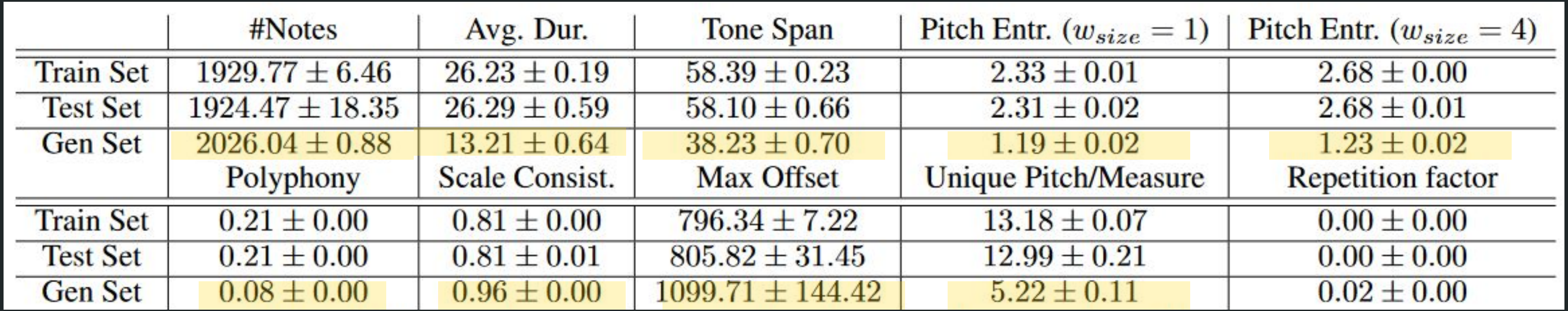
There is **no standard way to evaluate music**, as concepts like “**creativity**” and “**meaning**” are hard to express mathematically. Instead, many works propose metrics assessing **basic technical properties of music** and **compare** the statistics of a **generated set against a set of real music**.

We collect the distributions of:

- **Number of Notes** and **Average Duration**
- **Tone Span** (difference between highest and lowest pitch in a song)
- **Pitch Class Histogram Entropy**: entropy of pitch frequencies within a sliding window of 1 or 4 measures
    - In pieces with a clear tonality, the entropy is expected to be low
- **Polyphony**: frequency of having at least 2 notes played simultaneously
- **Scale Consistency**: likelihood of the most probable scale of the song
    - The most probable scale (major, harmonic or natural minor) is computed by converting the pitches to the 12 base notes and assessing the ratio of notes played on-scale
- **Maximum Offset**: largest offset in time between two events
- **Unique Pitches in Measures**: average number of unique pitches in each measure
- **Repetition Factor**: ratio of measures that are equal to each other in terms of beat, position and pitch of their notes.

Evaluation Metrics: Results

We generate (**Gen Set**) a set of **4002 songs** with top-p (p=0.9) sampling and the **same genre distribution of the training set**.



	#Notes	Avg. Dur.	Tone Span	Pitch Entr. ($w_{size} = 1$)	Pitch Entr. ($w_{size} = 4$)
Train Set	1929.77 ± 6.46	26.23 ± 0.19	58.39 ± 0.23	2.33 ± 0.01	2.68 ± 0.00
Test Set	1924.47 ± 18.35	26.29 ± 0.59	58.10 ± 0.66	2.31 ± 0.02	2.68 ± 0.01
Gen Set	2026.04 ± 0.88	13.21 ± 0.64	38.23 ± 0.70	1.19 ± 0.02	1.23 ± 0.02
	Polyphony	Scale Consist.	Max Offset	Unique Pitch/Measure	Repetition factor
Train Set	0.21 ± 0.00	0.81 ± 0.00	796.34 ± 7.22	13.18 ± 0.07	0.00 ± 0.00
Test Set	0.21 ± 0.00	0.81 ± 0.01	805.82 ± 31.45	12.99 ± 0.21	0.00 ± 0.00
Gen Set	0.08 ± 0.00	0.96 ± 0.00	1099.71 ± 144.42	5.22 ± 0.11	0.02 ± 0.00

The distributions of statistics is quite different, with some highlights:

- The **Number of Notes is high**, meaning that there probably **aren't many instruments or events** apart from notes. Generated songs also have **shorter notes** and **less polyphonic melodies**.
- Tone Span, Pitch Entropy and Unique Pitch/Measure are lower: probably **songs are more predictable**. This can also be seen with the higher Scale Consistency
- Maximum Offset is quite large: there are indeed **large moments of silence** in the songs

Genre Classification

Generated songs **repeat themselves** a lot and have **awkward silences**, but in listening tests we also observe an **affinity** between the **chosen genre** and the **chosen instruments**.

We want to **evaluate** the **capacity of the model** to **take the genre into account**. Our hypothesis is that if the model is really influenced by the genre, **we should expect the** output dense **representation of context** (before the heads) to **partially incorporate genre information**.

Therefore, we propose to build a simple **Genre Classifier on top of the frozen Transformer** and train it for the task of **multi-label classification** using the **genre vector as GT label**.

Since **we don't change the underlying models' weights,** the assumptions is that if we are able to train the model satisfyingly, the genre is indeed affecting the generation.

Classification Metrics

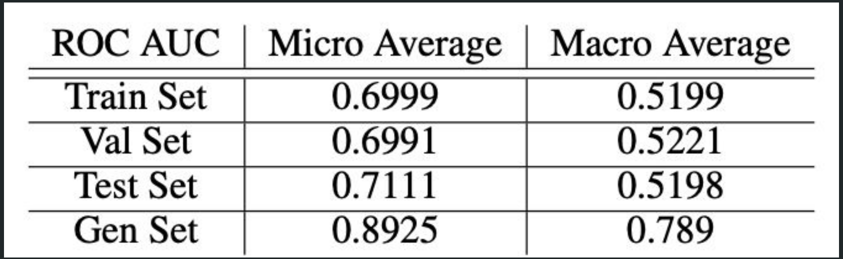
ROC AUC

Problem: since ROC is TPR (true positive rate) / FPR (false positive rate) at different thresholds, the problem must be made binary

- Class by class
- Highest scoring class per sample

The ROC AUC is very low for normal datasets, possible problems:

- Transformer model is not powerful enough to create a good latent representation that encapsulates genre
- Classifier is not powerful enough to extract the information
- Genre classification is not a well-set problem
- Our labels are incorrect so the classification is impossible



ROC AUC	Micro Average	Macro Average
Train Set	0.6999	0.5199
Val Set	0.6991	0.5221
Test Set	0.7111	0.5198
Gen Set	0.8925	0.789

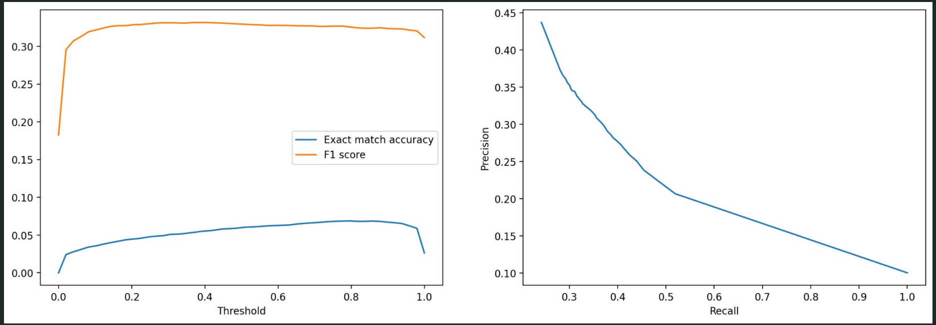
Classification Metrics

Exact Accuracy

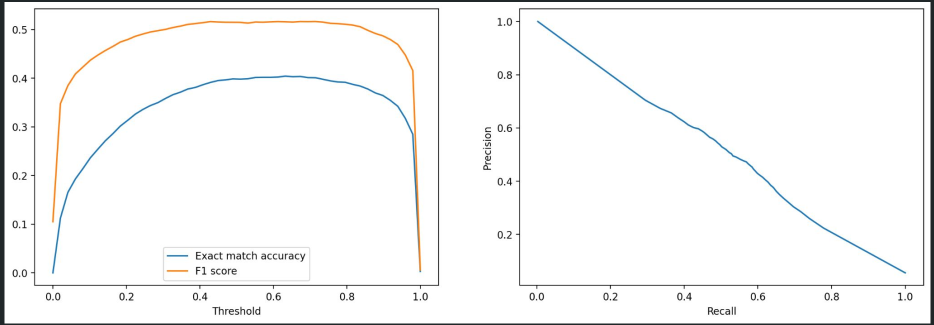
- Only completely correct guess
- Sample wise: we add confusion matrices with varying thresholds and computed accuracy, F1 score and a precision-recall curve

The performance of the gen set is better in both metrics, how?:

- The model only generates from one single genre
- The generation phase propagates genre information until the last layer of latent space
- However, we cannot fully explain such a big improvement



Val set



Gen set

Future Work and Considerations

Generation

- More than one genre during generation
- Clean the dataset more, removing very short notes (probably artifacts)
- Explore the hyperparameters of generation (top-p / top-k)
- Other types of generation (ex. N-beat continuation)

Model and Training

- Randomized cutting of songs (instead of always at the beginning)
- Double Head model with a skip connection
- Masking Model implemented in pytorch

Genre Classifier

- Objective: “how much information about the genre is present in the final embeddings?”
- Avoid influences during training: change the input genre during genre classifier training

Thanks for the Attention!