UNIVERSITATEA DIN BUCUREȘTI

FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ

SPECIALIZAREA INFORMATICĂ

Lucrare de licență

GENERATING TRADITIONAL ROMANI MUSIC WITH TRANSFORMERS

Absolvent
Budacă Răzvan-Gabriel

Coordonator științific
Lect.dr. Sergiu Nisioi

București, iunie 2023

Rezumat

În ultimii ani Transformerele au fost folosite din ce în ce mai mult pentru a genera muzică în domeniu simbolic. În același timp, muzica lăutărească este o formă de arta transmisă într-o bulă închisă care dispare încet. Am hotărât să duc mai departe acest Large Language Model și să văd dacă poate învăța subtilitățile și laitmotivele unui gen de muzică mai complex care se învață doar după ureche. În acest sens am colectat un nou dataset care să înglobeze lăutărie din mai multe țări, am antrenat un Transformer și am analizat melodiile generate folosind tehnici de NLP adaptate pe notație muzicală, dar și metode de muzicologie computațională. Astfel am obținut un tool ce duce mai departe muzica lăutărească, poate fi folosit de începători pentru analiza ei si poate fi folosit pentru a genera melodii originale in acest stil.

Abstract

Transformers have been used a lot in the last years when it comes to music generation in symbolic domain. At the same time, traditional Romani music is a dying art form which gets passed on to the newer generation in a small bubble. To combat this, I took this Large Language Model and put it to the test to see if it can learn the subtleties and leitmotifs of a more complex music genre which is learned only by ear. In order to do this, I collected a new dataset which contains traditional Roma music from different countries, I trained a Transformer and analyzed the generated songs using NLP methods adapted for musical notation, but also methods used in computational musicology. As a result I got a tool which keeps the genre alive, can be used for study and generates novel melodies.

Contents

Chapter 1

Introduction

Traditional Romani music is a rich and vibrant cultural heritage that has been passed down through generations. It encompasses a diverse range of musical styles, expressive melodies, and intricate rhythms, reflecting the history, emotions, and experiences of the Romani people. Preserving and promoting this musical tradition is of utmost importance, not only for the Romani community but also for the broader appreciation and understanding of world music.

In recent years, there has been a growing interest in employing artificial intelligence techniques for music generation[9][28][27]. One prominent approach is the use of Transformer models, which have demonstrated remarkable success in various natural language processing tasks. The Transformer model's ability to capture long-range dependencies and generate coherent sequences has sparked curiosity about its potential application in the realm of music composition.

This bachelor's thesis aims to explore the possibilities of using Museformer, a Transformer based model, to generate traditional Romani music. By leveraging the power of AI and deep learning, we seek to contribute to the preservation, innovation, and exploration of Romani musical heritage. This research not only holds potential for generating new melodies but also provides a platform for experimentation and creative collaboration within the music production community.

To achieve this, we will employ a dataset comprising a collection of authentic Romani melodies. These melodies will serve as the foundation for training the Transformer model, allowing it to learn the stylistic and structural characteristics inherent in Romani music.

In the following chapters I will present the necessary theory about Transformers and the specifics of Museformer, the representation of music notes in symbolic domain, the Jaccard Index and Locality Sensitivity Hashing used for evaluating the model, important music theory concepts like keys, intervals and scales and a classic key detection algorithm.

Lastly, I will be talking about my personal contribution in collecting the dataset, training the model and evaluating it's performance with significant metrics like the Jaccard similarity or metrics which involve music theory, like the model's preference for notes

which come from traditional Roma scales, or it's preference for ornamentation in this style by using trills. In the end I will present the results of the listening test, where people listened to the best melodies generated by the model and voted on the best sounding one and the one which best fits this beautiful style.

This thesis was not written with ill intent towards the Roma people and the model was not trained in order to appropriate their culture and tradition. This is a work of passion and appreciation towards their culture.

Chapter 2

Preliminaries

2.1 Transformers

2.1.1 The usual transformer

The transformer is a Large Language Model with an encoder-decoder structure which uses a so-called "self-attention" mechanism which basically creates a new token by taking into account previously created tokens.[23]

The encoder receives a series of symbol representations $(x_1, x_2, ..., x_n)$ and maps them to a sequence $y = (y_1, y_2, ..., y_n)$. Given a sequence y, the decoder will generate a new output $(z_1, z_2, ..., z_n)$. Every time a new token is generated, previous tokens are consumed in order to understand the "context".[23]

An attention function encompasses the mapping of a query and a set of key-value pairs to generate an output. In this mapping procedure, vectors represent the query, keys, values, and output. The output is derived by computing a weighted sum of the values, where the weight assigned to each value is determined by a compatibility function that considers the relationship between the query and its corresponding key.[23]

2.1.2 Museformer

Museformer, a Transformer-based model introduced by Yu et al. (2022) [27], follows the original architecture but introduces a novel attention scheme known as Fine- and Coarse-Grained Attention. The Fine-Grained Attention focuses on the preceding bar of the melody to maintain a coherent structure, while the Coarse-Grained Attention considers the remaining bars [27].

Let's assume we have a melody that has been transformed into a sequence of tokens, denoted as $X = (X_1, X_2, ..., X_b)$. Here, b represents the total number of bars in the melody. Each bar sequence can be represented as $X_i = (x_{i,1}, x_{i,2}, ..., x_{i,|X_i|})$, where $|X_i|$ signifies the length of the bar in tokens. To summarize each bar, a summary token

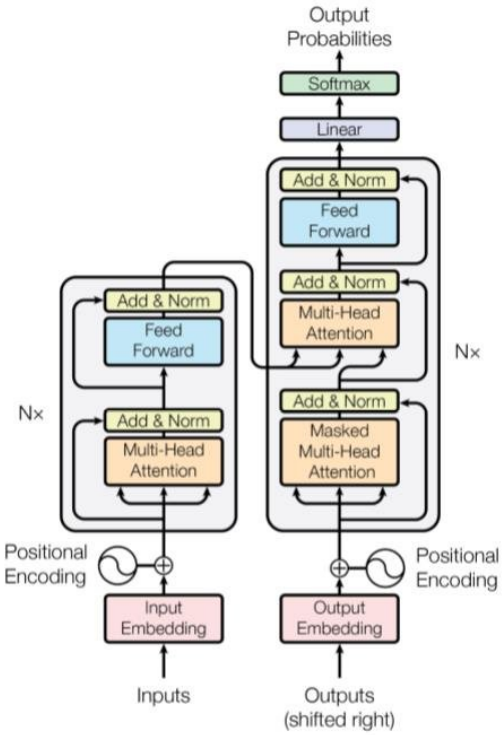

Figure 2.1: Architecture of a Transformer[23]

denoted as s_i is inserted at the end of the respective bar, which captures the essential information of the i-th bar. Consequently, the token sequence is transformed into $X = (X_1, s_1, X_2, s_2, ..., X_b, s_b)$ [27].

The core concept of the Fine- and Coarse-Grained Attention lies in the selective processing of tokens. When processing the i-th bar, the model applies Fine-Grained Attention to the (i-1)-th bar, whereas for the other bars, it utilizes the summary token for Coarse-Grained Attention [27].

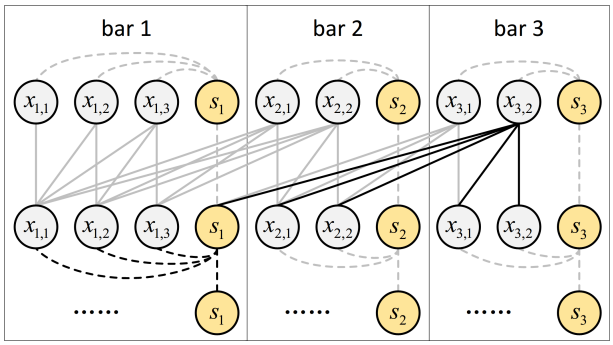

Figure 2.2: Fine and coarse grained attention for a melody of 3 bars. The 3rd bar takes into account all tokens from the 2nd bar, but only the summary token for the first bar.[27]

2.2 Symbolic Domain Representation

Museformer and Transformers in general work with token sequences, not MIDI notes. In order to bridge the gap between MIDI and a string notation, Museformer uses a representation method similar to REMI[9], where each MIDI note is mapped to a sequence of tokens.[27]

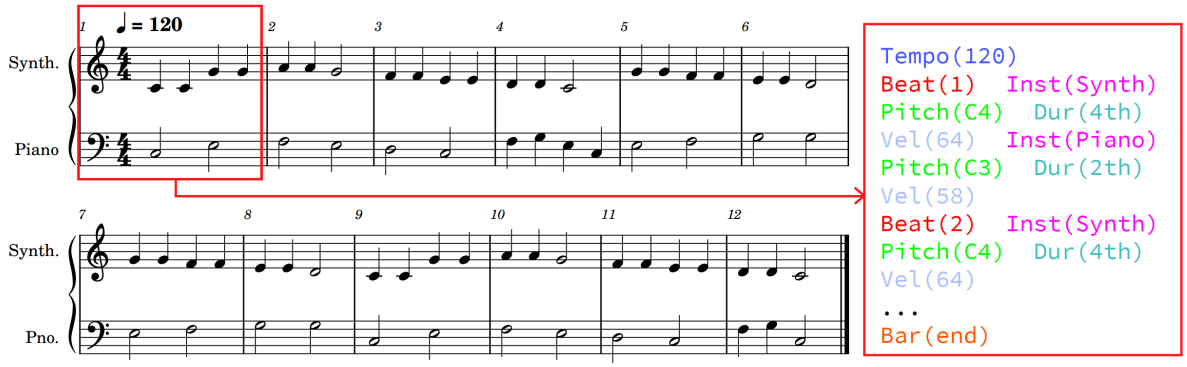

Figure 2.3: Music score for *Twinkle Twinkle Little Star* and it's token representation for the first bar.[27]

As seen in Figure 2.3, all important data from a song is mapped to a string of tokens: tempo, each beat, notes for each beat, velocity of the note, duration of the note, the instrument which plays the note and separators for the bars. Museformer also takes into consideration the time signature of a melody and it's changes.[27]

2.3 Locality Sensitivity Hashing and Jaccard Index

Let's say you have a huge collection of sets and you want to check if one of your sets is similar to one from the collection.

When it comes to similarity, the Jaccard Index[10] helps to quantize how similar 2 sets are. Given two sets A and B, the Jaccard Index can be calculated as follows:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{2.1}$$

However, when it comes to multiple sets, computational costs increase. Here's where Locality Sensitivity Hashing (LSH) comes into place. Using LSH paired with MinHashing can reduce query times to sub-linear cost.[16]

Locality-Sensitive Hashing (LSH) is commonly implemented using multiple hash functions to efficiently "hash" items. This technique increases the chances of similar items being assigned to the same bucket, while dissimilar items are less likely to collide. To identify potential matches, we consider pairs of items that hash to the same bucket across

any of the hash functions. By focusing on these candidate pairs, we can reduce the number of comparisons needed for similarity checking.

The underlying assumption is that dissimilar pairs will rarely hash to the same bucket, minimizing the need for further analysis. However, in cases where dissimilar pairs do collide in the same bucket, they are considered false positives. These instances are expected to be a relatively small fraction of all pairs. On the other hand, we expect that most truly similar pairs will hash to the same bucket under at least one hash function. In the event that some truly similar pairs do not share the same bucket, they are regarded as false negatives. Nonetheless, we aim to keep the occurrence of false negatives to a minimum.[22]

2.4 Music Theory

2.4.1 Key

When listening to music, one can say a song is in a certain key. A key is the main group of notes which form the harmonic foundation of a song. If all the notes used in a song are part of the C Major scale (Figure 2.4), then we can say that the song is in C Major.[4]

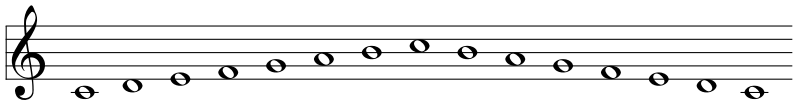

Figure 2.4: C Major scale

In this key, we would call the note C the Tonic of the key, because it is the first note of the scale and it feels the most consonant and relaxed note.

There are major keys for each of the notes from A to G, minor keys, and modes for each of the keys.

Sometimes, when trying to choose a key for a song while listening, the Tonic of the key might not match with the Tonic of the actual song. This is a sign that the song might be in a mode of the key.[7]

There are 7 modes:

2.4.2 Intervals

In music theory, intervals refer to the distance or relationship between two pitches. They are a fundamental concept used to describe the relative pitch of musical notes. Intervals can be classified based on their size, which is determined by counting the number of letter names and their respective accidentals between the two pitches. For example,

Mode	Tonic relative to major scale	Degrees
Ionian	I	1-2-3-4-5-6-7-8
Dorian	ii	1-2-♭3-4-5-6-♭7-8
Phrygian	iii	1-♭2-♭3-4-5-♭6-♭7-8
Lydian	IV	1-2-3-♯4-5-6-7-8
Mixolydian	V	1-2-3-4-5-6-♭7-8
Aeolian	vi	1-2-♭3-4-5-♭6-♭7-8
Locrian	vii^{ø}	1-♭2-♭3-4-♭5-♭6-♭7-8

Table 2.1: Modes of the Major scale.[7]

the interval between C and E is called a major third because there are two letter names (C and D) and two accidentals (C♯ and D♯) in between. Intervals can also be categorized as consonant or dissonant, depending on their perceived level of harmonic stability and tension. Consonant intervals, such as perfect fifths and major thirds, are considered stable and harmonically pleasing, while dissonant intervals, like minor seconds and tritones, create a sense of tension and require resolution. Intervals play a crucial role in music theory, composition, and harmony, providing a framework for understanding and analyzing the relationships between pitches and forming the building blocks of melodies, chords, and harmonies.[6]

Number of Semitones	Interval Name
0	Perfect unison
1	Minor second
2	Major second
3	Minor third
4	Major third
5	Perfect fourth
6	Tritone
7	Perfect fifth
8	Minor sixth
9	Major sixth
10	Minor seventh
11	Major seventh
12	Perfect octave

Table 2.2: The usual name of each interval, along the number of semitones.

2.4.3 Scales

In music theory, a scale is a group of notes ordered by pitch. Different scales can have different musical "flavors", or can be specific to musical styles.

The most important scales when talking about traditional Romani Music are the Minor scale, the Major scale (as they are essential in Western music), the Double harmonic

minor/ Hungarian minor/ Gypsy minor scale[19], the Double harmonic major/ Gypsy Major scale[2], the Harmonic minor scale[2] and the Romanian minor/Ukrainian Dorian/Altered Dorian scale[13]. These scales can be known by multiple names but they refer to the same musical concept.

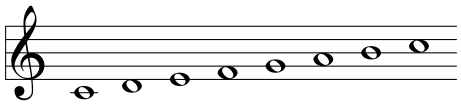

Figure 2.5: C Major scale

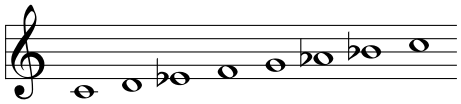

Figure 2.6: C Minor scale. The 3rd, 6th and 7th degrees are flattened.

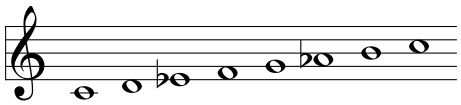

Figure 2.7: C Harmonic Minor scale. The 3rd and 6th degrees are flattened.

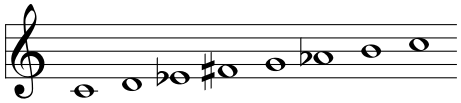

Figure 2.8: C Double Harmonic Minor scale. The 3rd and 6th degrees are flattened and the 4th degree is sharpened.

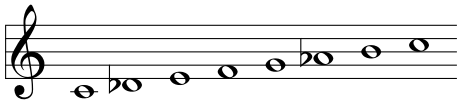

Figure 2.9: C Double Harmonic Major scale. The 2nd and 6th degrees are flattened. It can be reffered to as a gypsy scale because of the diminished step between the first and second degree.

2.5 Key Detection Algorithm

The Krumhansl-Schmuckler Key-Finding Algorithm utilizes ”key profiles” to determine the key of a musical piece[21]. These profiles consist of vectors with 12 values,

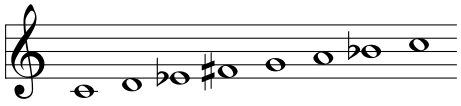

Figure 2.10: C Ukrainian Dorian scale. The 3rd and 7th degrees are flattened, while the 4th degree is sharpened. The 3 semitone jump from the third degree to the fourth degree give this scale a Gypsy feel.

which represent the relative stability of the 12 pitch classes when compared to a specific key. The creation of these key profiles was based on experiments conducted by Krumhansl and Kessler. In these experiments, participants were asked to evaluate how well each pitch class "fit" within a given key context, such as a cadence or scale. A higher value in a key profile indicates that the corresponding pitch class was considered a good match for that particular key. As a result, each of the 24 major and minor keys has its own unique key profile[21].

Pitch Class	**Number**
C	6.35
C#	2.23
D	3.48
D#	2.33
E	4.38
F	4.09
F#	2.52
G	5.19
G#	2.39
A	3.66
A#	2.29
B	2.88
c	6.33
c#	2.68
d	3.52
d#	5.38
e	2.60
f	3.53
f#	2.54
g	4.75
g#	3.98
a	2.69
a#	3.34
b	3.17

Table 2.3: Krumhansl-Kessler probe-tone profiles.[11] An uppercase letter indicates a major key, while a lowercase letter indicates a minor key.

After Krumhansl and Kessler came up with their weights, other people proposed new values: Aarden-Essen, Bellman-Budge, Temperley-Kostka/Payne and Craig Sapp.

To determine the most probable key of a given song, we generate a pitch histogram of all the notes, choose one of the weights arrays proposed, normalize everything and apply a Pearson correlation. The highest coefficient indicates the most probable key of the given song.[11]

$$key_k = \arg\max_k \left(\frac{1}{N} \sum_{i=1}^{N} z_{x,i} \cdot z_{y_k,i} \right) \tag{2.2}$$

where:

- k is an index over all the keys
- N is the total number of pitch-clases
- x is the pitch-class histogram
- y_k is a particular scale-degree weighting set for a particular key
- z indicates the z-score version of the sequence

2.6 Computational Musicology Tools

Computational musicology is an interdisciplinary field that combines music theory, computer science, and data analysis techniques to study and understand music from a computational perspective. It leverages computational methods to analyze large-scale music datasets, extract meaningful patterns and structures, and gain insights into various aspects of music, including composition, performance, and perception. By employing algorithms and statistical models, computational musicologists can explore music in new ways and uncover hidden relationships, trends, and cultural influences.[24]

In computational musicology, researchers use computational tools and techniques to analyze music notation, audio recordings, and metadata associated with music. These tools enable tasks such as music transcription, melody extraction, chord analysis, genre classification, style modeling, and sentiment analysis. These analyses provide valuable insights into music history, evolution, and cultural context. Computational musicology also facilitates the development of computer-assisted composition systems, music recommendation engines, and interactive music systems that enhance the creative process for composers and producers. Ultimately, computational musicology contributes to our understanding of music as a complex and multifaceted art form, bridging the gap between music theory, technology, and empirical research.[24]

Pretty-midi[18] is a Python library designed to simplify the manipulation, analysis, and synthesis of MIDI files. Developed by Colin Raffel and Daniel P. W. Ellis, it provides

Note	Aarden-Essen	Temperley-Kostka/Payne	Bellman-budge	Craig-Sapp
C	17.7661	0.748	16.80	2.0
C#	0.145624	0.060	0.86	0.0
D	14.9265	0.488	12.95	1.0
D#	0.160186	0.082	1.41	0.0
E	19.8049	0.670	13.49	1.0
F	11.3587	0.460	11.93	1.0
F#	0.291248	0.096	1.25	0.0
G	22.062	0.715	20.28	2.0
G#	0.145624	0.104	1.80	0.0
A	8.15494	0.366	8.04	1.0
A#	0.232998	0.057	0.62	0.0
B	4.95122	0.400	10.57	1.0
c	18.2648	0.712	18.16	2.0
c#	0.737619	0.084	0.69	1.0
d	14.0499	0.474	12.99	0.0
d#	16.8599	0.618	13.34	1.0
e	0.702494	0.049	1.07	0.0
f	14.4362	0.460	11.15	1.0
f#	0.702494	0.105	1.38	0.0
g	18.6161	0.747	21.07	2.0
g#	4.56621	0.404	7.49	1.0
a	1.93186	0.067	1.53	0.0
a#	7.37619	0.133	0.92	1.0
b	1.75623	0.330	10.21	0.0

Table 2.4: Musical Profiles Table.[11] An uppercase letter indicates a major key, while a lowercase letter indicates a minor key.

a user-friendly interface for reading, modifying, and generating MIDI data. With Pretty-midi, users can easily access and modify elements of MIDI files such as notes, chords, tempo, and control change events. The library also offers MIDI-to-audio conversion capabilities, allowing users to render MIDI compositions into audio formats. Additionally, Pretty-midi provides visualization tools for piano rolls, event plots, and control change graphs, enhancing the understanding and analysis of MIDI data. It is a versatile tool used in music production, algorithmic composition, music analysis, and machine learning applications.[18]

Music21[14] is a Python library designed for music analysis, manipulation, and composition. It provides a comprehensive set of tools and functionalities for working with musical notation and symbolic music data. With Music21, users can easily read, write, and analyze music in various formats, including MIDI, MusicXML, and more.The library offers a wide range of features, including music notation rendering, pitch and interval calculations, chord and harmony analysis, key and mode detection, and melodic pattern matching. It also supports music generation and composition through algorithmic methods such as Markov chains and species counterpoint rules.[14]

Chapter 3

Personal Contribution

Everything mentioned here (dataset, the training jupyter notebook, generated melodies in audio, midi and token form, analysis scripts and tables/graphs) can be found on GitHub[1]

3.1 Dataset

The collected dataset contains a total of around 725 songs which amount to around 60h of content. The songs were handpicked from the Romanian folklore, but it also contains traditional songs from Bulgaria, Turkey, Greece and the Arabian Peninsula. As a nomadic group, the Roma people gathered music influences from all around the world, so even if the dataset contains a mix of cultures, most of the songs have similar traits.[13]

All these songs were downloaded from midi websites such as Clape.ro[5], Midis101[15] and PyianistSet[17], but also from YouTube channels specialized in song transcribing and tutorials: MitzaBV[3] and Dan Bacau[1].

When it comes to the YouTube channels, to extract a midi file from the videos I used youtube-dl-gui[26], a third party app which downloads multiple videos from YouTube, and video2midi[20], an open source tool which extracts MIDI information from a recorded Synthesia[12] video.

However, most MIDI files contain both the melody and the harmony aspects of a song. To train more efficiently, I chose to extract only the lead tracks of a song, tracks which basically mimic the right playing hand of a piano player. To extract the melody and counter-melody parts of a song, I used pretty-midi[18], an open-source collection of tools for handling MIDI data. Using these tools I removed the instruments which are clearly marked as drums or bass; to further remove harmony based channels, I cut any instrument which has notes below a certain pitch (most lead tracks have notes in the upper ranges of pitches). Moreover, harmony channels contain mostly progressions of chords. To remove

[1] https://github.com/BudiGabe/Generating-Traditional-Romani-Music-with-Transformers

these, I removed any channels which have multiple notes played at the same time above a certain threshold.

In the end I was left with a dataset composed of lead tracks in the style of traditional Romani music from around the world.

To confirm the validity of this dataset, I compared it with POP909[25], a dataset used for training Large Language Models in the past, which has 909 songs of around 60 hours of content.

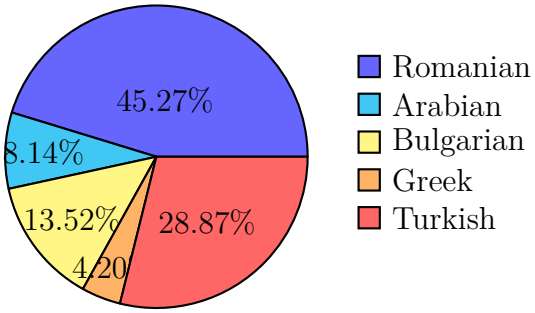

Figure 3.1: Distribution of songs in the dataset

3.2 Model Training

The first training was done on the basic dataset of around 725 songs for 430 epochs using Google Colab's V100 GPU. Before training, I used a 8/1/1 split for the dataset and all songs were converted into REMIGEN tokens, a variant of REMI presented in Figure 2.3. All songs were normalized to C Major or A minor to facilitate learning.

As seen in Figures 3.2 and 3.3, after the first training the model does not seem to be able to produce novel music, but rather generate extremely similar melodies to those in the dataset, judging by the validation loss.

To combat this, I decided to augment the dataset by creating two new tracks from each song, by pitch shifting them 3 semitones up and 3 semitones down.

The second training was done on the augmented dataset of around 2100 songs for 400 epochs using Google Colab's V100 GPU. Before training, the same split and token conversion were used. However, song normalization was disabled, it would have negated the positives of augmentation.

As seen in figures 3.2 and 3.3, the model performed better. I had to stop training at 400 epochs due to missing funds and Colab's prices for GPUs, but the validation loss could still go lower than this.

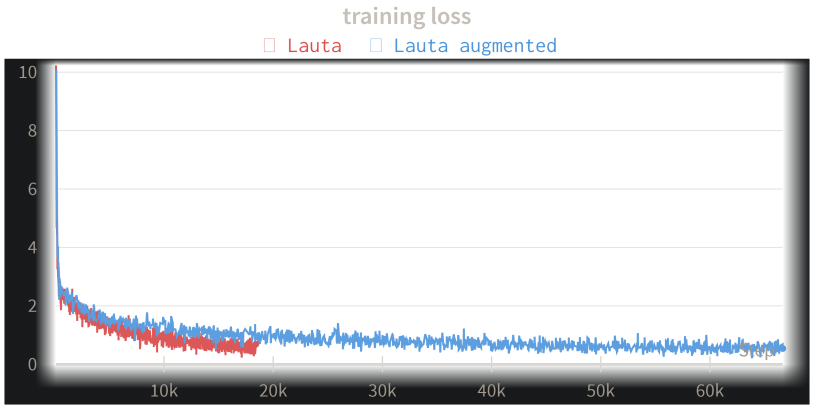

Figure 3.2: Training loss graph for the Museformer model. Lauta marks the non augmented dataset, while Lauta augmented marks the augmented dataset.

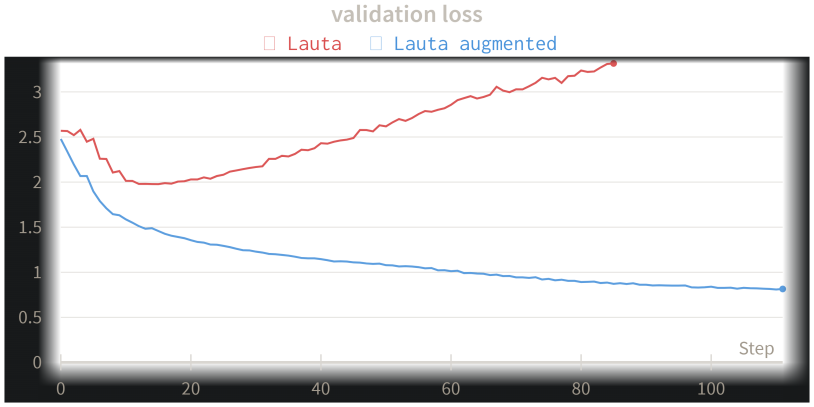

Figure 3.3: Validation loss graph for the Museformer model. Lauta marks the non augmented dataset, while Lauta augmented marks the augmented dataset. You can clearly see that the model started to overfit early on the first training, while it could have trained for a longer time on the second training.

3.3 Evaluation

To evaluate how the model performs, I opted to use traditional NLP algorithms like the Jaccard Similarity, Locality Sensitive Hashing and ngrams, but applied on music specific tokens like measures and intervals. Besides these methods, I came up with metrics which evaluate how close to the traditional Romani style the generated songs are, by analyzing the model's preference for certain notes above others and also a metric which measures how often a Romani performance trick/ornament is used in songs: the trill.

In the following section we will explore how the model performed in 4 checkpoints: 2 from the first training (best checkpoint when it comes to validation loss and the last, overfit checkpoint) and 2 from the second training (the last checkpoint which is also the best when it comes to validation loss, and a checkpoint from the middle, where the validation loss is still decent).

For the evaluation I've chosen 10 generated melodies from each of the 4 checkpoints. For each melody I did my best to choose an accurate Tonic, which enables the study of modes.

In figure 3.4 we can see the score for a random generated melody.

3.3.1 Jaccard similarity

In order to obtain the Jaccard Index for different generated songs, I used the token form of the songs. Moreover, to accurately detect similar melodies I removed any tokens that didn't mark the pitch of a note or the start of a new bar. When it comes to comparing melodies, the velocity of a note, the time signature of the song or the instrument playing don't really matter. If the same notes are playing, a melody can easily be recognized. Moreover, a song can be played in a different key, pitched up or down, and still be recognized. To combat this, instead of comparing pitch tokens I compared the intervals present in each bar, or the relation between notes instead of the absolute values of the notes.

When it comes to min hashing and locality sensitive hashing when calculating the Jaccard index, I decided to hash the individual measures of each song, just like you would hash the words of a sentence in NLP. However, there is a posibility that a musical phrase could start in the middle of a bar and continue up to the middle of the next bar. To combat this, I decided to also compare ngrams, where n is the average length in notes of the measures in the dataset.

The Jaccard threshold, the index which says how similar to the dataset a generated melody should be, was calculated based on the average note count of the songs. In the worst case scenario, the intersection would contain would contain all the notes from the generated melody, and the reunion would contain all the notes from the dataset melody. Since the dataset melodies contain way more notes than the generated ones, a low Jaccard threshold was obtained.

Comparing measures

Below are the results when comparing songs measure by measure.

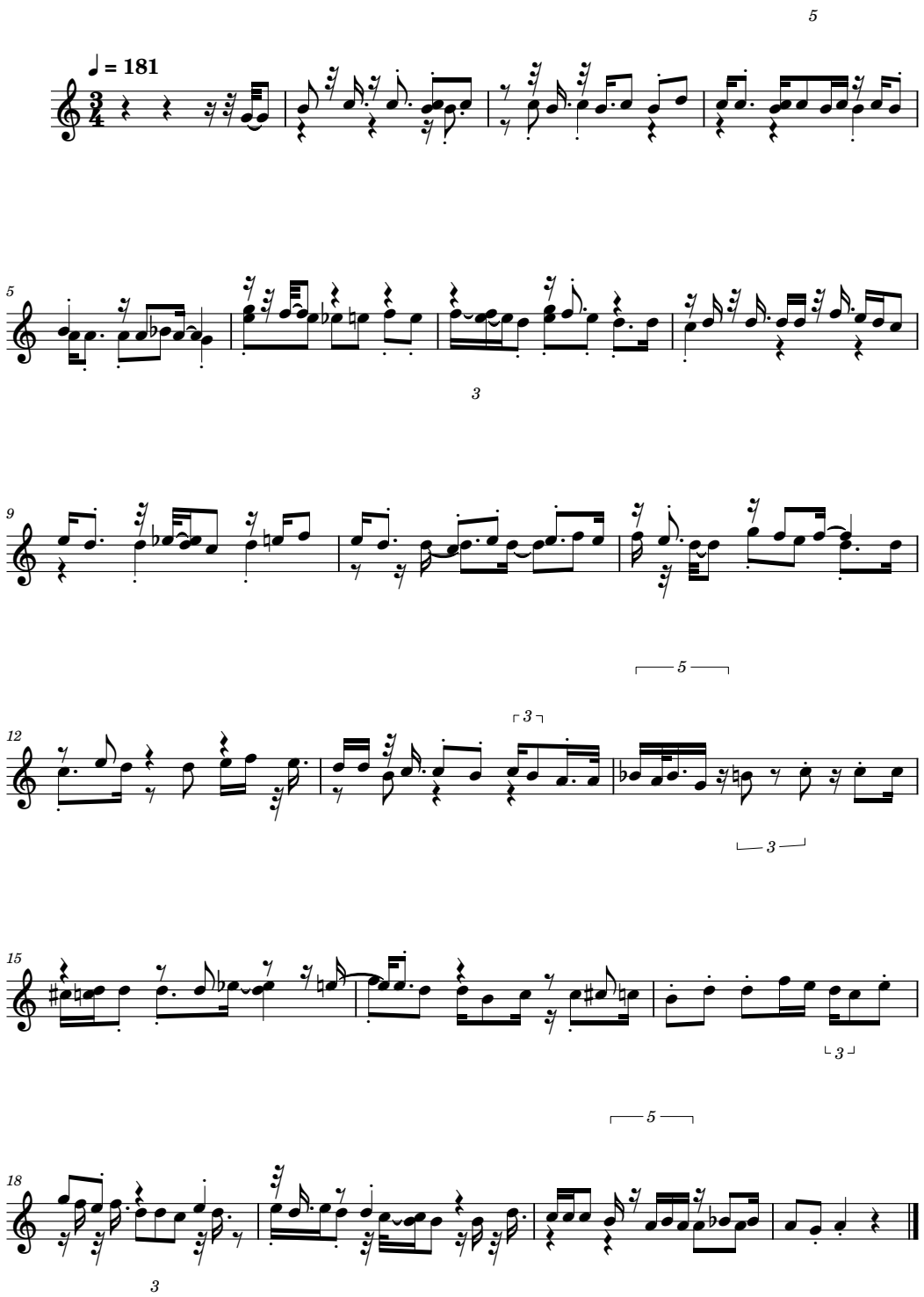

Figure 3.4: One of the generated melodies after training. We can clearly see trills, syncopation, and the repeated use of motifs.

Song	Approximate Neighbours
random-13-A	3
random-15-A-E	0
random-16-G	0
random-18-A	0
random-19-A	0
random-23-A	0
random-3-C	3
random-32-G	0
random-41-A	21
random-5-A	0

Table 3.1: Number of approximate neighbours for melodies generated by the **last** checkpoint of the model trained on the **augmented** dataset. Since each melody is tripled as part of the augmentation, 3 approximate neighbours indicate only one similarity.

Song	Approximate Neighbours
random-1-A	0
random-11-A	0
random-12-A	0
random-13-A	0
random-14-C	0
random-19-A	0
random-21-C	0
random-3-C	0
random-4-A	0
random-7-A	0

Table 3.2: Number of approximate neighbours for melodies generated by the **middle** checkpoint of the model trained on the **augmented** dataset.

Song	Approximate Neighbours
1random-2-A	2
1random-4-D	0
random-10-B	0
random-11-E	0
random-12-G	0
random-17-G	0
random-19-D	0
random-20-A	0
random-7-A	0
random-9-E	2

Table 3.3: Number of approximate neighbours for melodies generated by the **best** checkpoint of the model trained on the **basic** dataset.

Song	Approximate Neighbours
new-A	0
new-D	0
random-1-A	1
random-12-A	0
random-13-A	2
random-15-A	0
random-17-A	1
random-4-A	0
random-5-D	0
random-9-E	0

Table 3.4: Number of approximate neighbours for melodies generated by the **last** checkpoint of the model trained on the **basic** dataset.

Comparing ngrams

Below are the results when comparing songs by ngrams, where n is the average number of notes per measure.

Song	Approximate Neighbours
random-13-A	0
random-15-A-E	0
random-16-G	0
random-18-A	0
random-19-A	0
random-23-A	0
random-3-C	3
random-32-G	0
random-41-A	0
random-5-A	0

Table 3.5: Number of approximate neighbours for melodies generated by the **last** checkpoint of the model trained on the **augmented** dataset.

Song	Approximate Neighbours
random-1-A	0
random-11-A	0
random-12-A	0
random-13-A	0
random-14-C	0
random-19-A	0
random-21-C	0
random-3-C	0
random-4-A	0
random-7-A	0

Table 3.6: Number of approximate neighbours for melodies generated by the **middle** checkpoint of the model trained on the **augmented** dataset.

Song	Approximate Neighbours
1random-2-A	0
1random-4-D	0
random-10-B	0
random-11-E	0
random-12-G	0
random-17-G	0
random-19-D	3
random-20-A	5
random-7-A	1
random-9-E	0

Table 3.7: Number of approximate neighbours for melodies generated by the **best** checkpoint of the model trained on the **basic** dataset.

Song	Approximate Neighbours
new-A	2
new-D	1
random-1-A	1
random-12-A	5
random-13-A	2
random-15-A	0
random-17-A	3
random-4-A	0
random-5-D	1
random-9-E	0

Table 3.8: Number of approximate neighbours for melodies generated by the **last** checkpoint of the model trained on the **basic** dataset.

Looking at tables 3.5 and 3.1, it looks like most of the melodies were original. Song random-41-A seems to have 7 similar neighbours (each melody was tripled in the augmented dataset), but it's entirely possible that the melody uses traditional leitmotifs, well known in folklore, which cannot really be blamed.

The middle checkpoint of the model trained on the augmented dataset performed the best, looking at tables 3.6 and 3.2. No neighbours were found.

The checkpoints obtained from the basic dataset seem to have performed a bit more poorly, especially the checkpoint with the best validation loss, as seen in tables 3.8 3.4 3.7 3.3.

3.3.2 Key detection extended

To implement the Key for Key algorithm[21] I used the pretty-midi library to gain access to MIDI note information. To properly create an accurate pitch histogram, instead of counting note apparitions I counted notes in sixteenths in to keep track of note duration, not just pitch.

When it comes to the weights, I chose to implement options for the classic Krumhansl-Kessler weights, Aarden-Essen and Craig-Sapp.[11] After implementing them, I ended up using Craig-Sapp Simple Weights because they out-perform the other weights despite their simplicity.[8] The weights are made up of 2 for the tonic and dominant, 1 for other diatonic notes and 0 for non-diatonic notes.[8]

Before calculating the coefficients, an important step is to normalize the pitch histogram and the chosen weights, so I used the zscores of these arrays. After normalizing, we just apply the Pearson correlation. In the end, the highest coefficient indicates the most probable key for the song.

After finding the basic key of a melody, it's time to dig deeper. First of all, I counted how many notes are considered out of key, as these notes are a sign of improvisation and jazzyness specific to the Roma musicians: modulation of keys, augmented intervals, Gypsy scales, grace notes, chromatic and diatonic passages.[2] After this high-level overview of the melody, I went deeper and analyzed the model's preference for notes and intervals specific to the Roma people, notes which create musical phrases indicating chromatic passages, Gypsy scales, grace notes and augmented intervals.

Before running the analysis scripts, I listened to every melody and noted the Tonic of each song in the midi filename. The Tonic helps to identify if a song is in a certain mode of a key.

This analysis was possible using pretty-midi[18], for the classic key detection algorithm, and music21[14], which allowed to generate specific scales and transpose notes.

The following tables present the model's preference for notes from traditional scales considered specific for Roma music over basic notes from the simple minor and major keys.

Song	Key	Out of key	Harmonic Minor		Romanian Minor				Double major				Gypsy Minor						Phrygian (dominant)					
			7	♯7	4	♯4	6	♯6	2	♭2	6	♭6	3	♭3	4	♯4	6	♭6	2	♭2	6	♭6	7	♭7
random-13-A	A minor	23	2	11	-	-	-	-	37	4	17	23	-	-	-	-	-	-	-	-	-	-	-	-
random-15-E	A minor	36	26	12	-	-	-	-	16	13	16	27	-	-	-	-	-	-	16	13	16	27	26	4
random-16-D	A minor	47	35	12	35	12	1	12	12	1	16	22	19	16	35	12	1	6	-	-	-	-	-	-
random-18-A	A minor	12	15	7	-	-	-	-	16	2	18	32	-	-	-	-	-	-	-	-	-	-	-	-
random-19-D	A minor	35	1	5	1	5	7	27	27	7	6	18	1	6	1	5	7	14	-	-	-	-	-	-
random-23-A	A minor	29	1	17	-	-	-	-	24	0	13	47	-	-	-	-	-	-	-	-	-	-	-	-
random-3-C.	A minor	40	6	0	-	-	-	-	14	1	7	23	-	-	-	-	-	-	-	-	-	-	-	-
random-32-G	A minor	40	16	2	-	-	-	-	32	3	6	15	-	-	-	-	-	-	-	-	-	-	-	-
random-41-A	A minor	10	3	7	-	-	-	-	51	0	16	21	-	-	-	-	-	-	-	-	-	-	-	-
random-5-A.	A minor	38	5	9	-	-	-	-	28	1	7	16	-	-	-	-	-	-	-	-	-	-	-	-
			110	82	36	17	8	39	257	32	122	244	20	22	36	17	8	20	16	13	16	27	26	4

Table 3.9: Note preference for melodies generated by the **last** checkpoint of the model trained on the **augmented** dataset.

Song	Key	Out of key	Harmonic Minor		Romanian Minor				Double major				Gypsy Minor						Phrygian (dominant)					
			7	♯7	4	♯4	6	♯6	2	♭2	6	♭6	3	♭3	4	♯4	6	♭6	2	♭2	6	♭6	7	♭7
random-1-A	C major	10	-	-	-	-	-	-	40	2	14	0	-	-	-	-	-	-	-	-	-	-	-	-
random-11-A	E minor	9	26	0	26	0	4	0	0	4	30	46	2	30	26	0	4	26	-	-	-	-	-	-
random-12-A	E minor	19	31	4	31	4	8	5	5	8	26	22	0	26	31	4	8	26	-	-	-	-	-	-
random-13-A	E minor	38	11	0	11	0	31	5	5	31	36	36	4	36	11	0	31	14	-	-	-	-	-	-
random-14-C	A minor	10	18	4	-	-	-	-	0	1	20	40	-	-	-	-	-	-	-	-	-	-	-	-
random-19-A	C major	23	-	-	-	-	-	-	23	19	9	0	-	-	-	-	-	-	-	-	-	-	-	-
random-21-C	A minor	16	10	10	-	-	-	-	21	3	8	30	-	-	-	-	-	-	-	-	-	-	-	-
random-3-C	A minor	8	30	4	-	-	-	-	17	0	24	15	-	-	-	-	-	-	-	-	-	-	-	-
random-4-A	A minor	13	2	0	-	-	-	-	22	4	9	37	-	-	-	-	-	-	-	-	-	-	-	-
random-7-A	A minor	5	12	0	-	-	-	-	49	5	0	7	-	-	-	-	-	-	-	-	-	-	-	-
			140	22	68	4	43	10	182	77	176	233	6	92	68	4	43	66	0	0	0	0	0	0

Table 3.10: Note preference for melodies generated by the **middle** checkpoint of the model trained on the **augmented** dataset.

Song	Key	Out of key	Harmonic Minor		Romanian Minor				Double major				Gypsy Minor						Phrygian (dominant)					
			7	♯7	4	♯4	6	♯6	2	♭2	6	♭6	3	♭3	4	♯4	6	♭6	2	♭2	6	♭6	7	♭7
1random-2-A	A minor	16	7	0	-	-	-	-	45	3	1	24	-	-	-	-	-	-	-	-	-	-	-	-
1random-4-D	A minor	16	0	8	0	8	0	24	24	0	24	48	0	24	0	8	0	23	-	-	-	-	-	-
random-10-B	E minor	0	26	0	-	-	-	-	0	0	27	83	-	-	-	-	-	-	0	0	27	83	26	0
random-11-E	A minor	0	71	0	-	-	-	-	0	0	37	136	-	-	-	-	-	-	0	0	37	136	71	0
random-12-G	A minor	66	89	0	-	-	-	-	0	45	2	5	-	-	-	-	-	-	-	-	-	-	-	-
random-17-G	A minor	77	88	0	-	-	-	-	18	47	0	6	-	-	-	-	-	-	-	-	-	-	-	-
random-19-D	A minor	90	12	0	12	0	24	6	6	24	0	12	20	0	12	0	24	44	-	-	-	-	-	-
random-20-A	A minor	52	3	24	-	-	-	-	32	12	11	18	-	-	-	-	-	-	-	-	-	-	-	-
random-7-A	A minor	1	11	0	-	-	-	-	47	0	0	8	-	-	-	-	-	-	-	-	-	-	-	-
random-9-E	A minor	15	0	15	-	-	-	-	23	0	31	41	-	-	-	-	-	-	23	0	31	41	0	0
			307	47	12	8	24	30	195	131	133	381	20	24	12	8	24	67	23	0	95	260	97	0

Table 3.11: Note preference for melodies generated by the **best** checkpoint of the model trained on the **basic** dataset.

Song	Key	Out of key	Harmonic Minor		Romanian Minor				Double major				Gypsy Minor						Phrygian (dominant)					
			7	♯7	4	♯4	6	♯6	2	♭2	6	♭6	3	♭3	4	♯4	6	♭6	2	♭2	6	♭6	7	♭7
new-A	A minor	0	41	0	-	-	-	-	51	0	6	6	-	-	-	-	-	-	-	-	-	-	-	-
new-D	A minor	14	12	0	-	-	-	-	16	1	5	25	-	-	-	-	-	-	-	-	-	-	-	-
random-1-A	A minor	23	16	0	-	-	-	-	22	8	1	13	-	-	-	-	-	-	-	-	-	-	-	-
random-12-A	A minor	5	20	3	-	-	-	-	63	2	19	8	-	-	-	-	-	-	-	-	-	-	-	-
random-13-A	A minor	15	13	9	-	-	-	-	22	2	18	47	-	-	-	-	-	-	-	-	-	-	-	-
random-15-A	A minor	39	0	22	-	-	-	-	34	8	26	5	-	-	-	-	-	-	-	-	-	-	-	-
random-17-A	A minor	11	13	0	-	-	-	-	20	0	23	38	-	-	-	-	-	-	-	-	-	-	-	-
random-4-A	A minor	30	14	7	-	-	-	-	27	1	10	13	-	-	-	-	-	-	-	-	-	-	-	-
random-5-D	A minor	34	2	31	2	31	0	9	9	0	25	33	3	25	2	31	0	35	-	-	-	-	-	-
random-9-E	A minor	24	15	24	-	-	-	-	25	0	23	19	-	-	-	-	-	-	25	0	23	19	15	0
			146	96	2	31	0	9	289	22	156	207	3	25	2	31	0	35	25	0	23	19	15	0

Table 3.12: Note preference for melodies generated by the **last** checkpoint of the model trained on the **basic** dataset.

We can see that all checkpoints have some preference for the Romani style. However, due to the undertrained nature of the best validation loss checkpoint obtained from the basic dataset, I doubt it was intentional for this checkpoint.

More songs have inclinations towards Gypsy Minor in the augmented checkpoints, but more songs have inclinations towards the mode Phrygian (Dominant) in the non-augmented checkpoints. However, more songs have Romanian Minor elements in the augmented checkpoints.

When it comes to Harmonic Minor, both parties have some inclinations.

3.3.3 Trill detection

Excessive ornamentation is an essential trait for the traditional music of the Roma people.[2][19] A well known ornament is called a Trill, a quick alternation between 2 notes which are a semitone or a tone apart.

To evaluate the model further, I chose to detect how many notes are part of trills in each melody, and even see what percentage of the melody is made up of trills.

To detect a trill, I selected groups of 3 consecutive short notes which are apart by a maximum of 2 semitones and come in a quick succession.

The following tables show the percentage of notes part of a trill in the songs.

Song	Trills	Percentage
random-13-A	29	0.52
random-15-E	0	0.0
random-16-D	29	0.51
random-18-A	24	0.43
random-19-D	22	0.39
random-23-A	18	0.25
random-3-C	17	0.31
random-32-G	25	0.45
random-41-A	13	0.2
random-5-A	25	0.44

Table 3.13: Trill count and percentage of notes part of a trill for melodies generated by the **last** checkpoint of the model trained on the **augmented** dataset.

Song	Trills	Percentage
random-1-G	30	0.53
random-11-A	27	0.48
random-12-A	23	0.4
random-13-A	22	0.4
random-14-C	28	0.5
random-19-G	27	0.47
random-21-C	21	0.38
random-3-C	29	0.52
random-4-A	31	0.56
random-7-A	24	0.42

Table 3.14: Trill count and percentage of notes part of a trill for melodies generated by the **middle** checkpoint of the model trained on the **augmented** dataset.

Song	Trills	Percentage
1random-2-A	13	0.24
1random-4-D	0	0.0
random-10-B	15	0.26
random-11-E	0	0.0
random-12-G	6	0.09
random-17-G	0	0.0
random-19-D	8	0.11
random-20-A	6	0.09
random-7-A	19	0.34
random-9-E	18	0.31

Table 3.15: Trill count and percentage of notes part of a trill for melodies generated by the **best** checkpoint of the model trained on the **basic** dataset.

Song	Trills	Percentage
new-A	18	0.33
new-D	26	0.47
random-1-A	27	0.49
random-12-A	22	0.42
random-13-A	24	0.44
random-15-A	16	0.27
random-17-A	28	0.5
random-4-A	22	0.39
random-5-D	30	0.53
random-9-E	16	0.28

Table 3.16: Trill count and percentage of notes part of a trill for melodies generated by the **last** checkpoint of the model trained on the **basic** dataset.

It looks like the middle checkpoint from the augmented dataset has the most trills on average, followed by the overfit checkpoint from the non-augmented dataset. Coming right up is the last augmented checkpoint which still stands strong.

3.3.4 Listening test

The comparison could not be complete without a listening test. I created a Google Form where I asked people if they listen to traditional Roma music frequently and made them choose which melody sounds better and which melody is closer to the Roma style. The melodies were handpicked from the previously shown sets, 1 from each set.

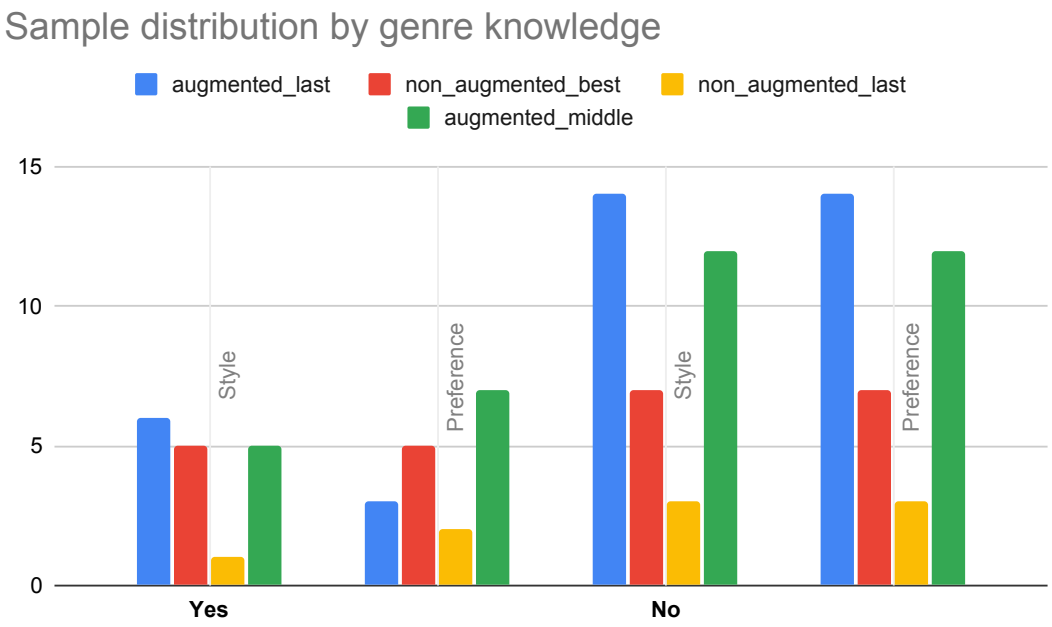

Figure 3.5: Sample distribution by genre knowledge

It looks like the people who actively search for and listen to this style have preffered the middle augmented checkpoint, but propose that the last augmented checkpoint is more fitting for the traditional Roma style. At the same time, the people who do not listen to this genre regularly have considered the last augmented checkpoint to be fitting for both categories. Looking at table 3.9, we can clearly see that random-23-A, the song used in the form, indeed has augmented intervals like ♯7 and ♭6, and looking at table 3.14, a quarter of the song is made up of trills.

Overall, the best checkpoint in terms of personal prefference was the middle augmented checkpoint and the checkpoint which best reproduced the Roma style

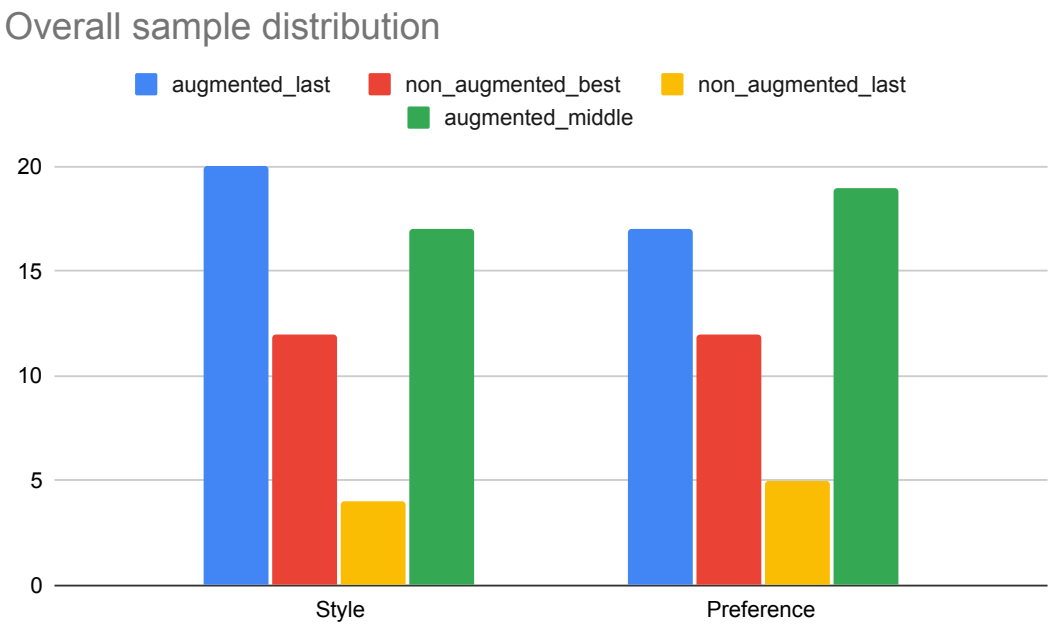

Figure 3.6: Overall sample distribution

was the last augmented checkpoint.

Chapter 4

Conclusions

In addition to its cultural and artistic significance, the system developed in this thesis holds immense potential as a valuable tool for both music enthusiasts looking to add this style to their repertoire and contemporary music producers. As a tool for Romani music study, the generated melodies serve as a rich resource for analyzing and understanding the intricate melodic patterns, scales, ornamentation, and improvisational elements that characterize traditional Romani music.

Moreover, this system can serve as a powerful tool for contemporary music producers seeking to incorporate elements of traditional Romani music into their compositions. By providing an AI-generated repertoire of melodies rooted in Romani musical traditions, producers can seamlessly infuse the vibrant spirit of Romani music into their own creative works. Whether it is for producing original compositions inspired by Romani melodies or incorporating Romani musical motifs into diverse genres and styles, this system opens up exciting possibilities for cross-cultural musical collaborations and innovative music production.

The model seems to be able to produce novel melodies which incorporate well-known leitmotifs from the Romani culture and have the ”oriental” feeling specific to the Romani style.

In the future, more songs could be transcribed by ear and added to the dataset for better training. There are plenty of well-known songs which are passed down through generations by ear, not by music notation, and these might be lost to time as it passes.

Bibliography

[1] Dan Bacau. *Dan Bacau YouTube Channel*. URL: https://www.youtube.com/@0075871(visited on 06/06/2023).

[2] Max Peter Baumann. "The Reflection of the Roma in European Art Music." In: *The World of Music*38.1(1996), pp. 95–138. ISSN: 00438774. URL: http://www.jstor.org/stable/41699074(visited on 06/05/2023).

[3] Mitza BV. *Mitza BV YouTube Channel*. URL: https://www.youtube.com/@mitzabv(visited on 06/06/2023).

[4] Samuel Chase. *What Is A Key In Music? A Complete Guide*. URL: https://hellomusictheory.com/learn/keys/(visited on 06/05/2023).

[5] clape.ro. *Clape.ro - Romanian Keyboardists Community*. URL: http://clape.ro/(visited on 06/06/2023).

[6] Dan Farrant. *A Guide To Music Intervals: The Gaps Between The Notes*. URL: https://hellomusictheory.com/learn/intervals/(visited on 06/07/2023).

[7] Dan Farrant. *The Musical Modes: What Are They?* URL: https://hellomusictheory.com/learn/modes/(visited on 06/05/2023).

[8] Fede Camara Halacand Daniel Shanahan. *Course on Computational Musicology: Key Finding*. URL: https://fdch.ar/computational_musicology-8824/colabs/unit_1-symbolic_data/Week_04-Key_Finding.html#Krumhansl-Schmuckler/Krumansl-Kessler(visited on 06/07/2023).

[9] Yu-Siang Huangand Yi-Hsuan Yang. *Pop Music Transformer: Beat-based Modeling and Generation of Expressive Pop Piano Compositions*. 2020. arXiv: 2002.00212[cs.SD].

[10] Fatih Karabiber. *Jaccard Similarity*. URL: https://www.learndatasci.com/glossary/jaccard-similarity/(visited on 06/05/2023).

[11] *Keycor manpage*. URL: https://web.archive.org/web/20220812060114/http://extra.humdrum.org/man/keycor/(visited on 06/06/2023).

[12] Synthesia LLC. *Synthesia*. URL: https://synthesiagame.com/(visited on 06/06/2023).

[13] Peter Manuel. "Modal Harmony in Andalusian, Eastern European, and Turkish Syncretic Musics." In: *Yearbook for Traditional Music*21(1989), pp. 70–94. ISSN: 07401558. URL: http://www.jstor.org/stable/767769(visited on 06/05/2023).

[14] Michael Cuthbert and Christopher Ariza and contributors. *music21 - A Toolkit for Computer-Aided Musical Analysis and Computational Musicology*. URL: %7Bhttp://web.mit.edu/music21/%7D.

[15] midis101.com. *Midis101*. URL: https://www.midis101.com/(visited on 06/06/2023).

[16] *MinHash LSH*. URL: https://ekzhu.com/datasketch/lsh.html(visited on 06/05/2023).

[17] Piyanist Set. *Tüm Klavyeler İçin Dev MIDI Arşivi 2022 by M. Taşdemir - Bedava İndir (Free Download)*. URL: https://www.piyanistset.com/tum-klavyeler-icin-dev-midi-arsivi-2022-by-m-tasdemir-bedava-indir-free-download/1519/(visited on 06/06/2023).

[18] Colin Raffeland Daniel P. W. Ellis. *Intuitive Analysis, Creation, and Manipulation of MIDI Data with pretty-midi*. 2014. URL: https://colinraffel.com/publications/ismir2014intuitive.pdf.

[19] A. T. Sinclair. "Gypsy and Oriental Music." In: *The Journal of American Folklore*20.76(1907), pp. 16–32. ISSN: 00218715, 15351882. URL: http://www.jstor.org/stable/534723(visited on 06/05/2023).

[20] svsdval. *video2midi - Convert Video to MIDI files*. URL: https://github.com/svsdval/video2midi(visited on 06/06/2023).

[21] David Temperley. “What’s Key for Key? The Krumhansl-Schmuckler Key-Finding Algorithm Reconsidered.” In: (1999). URL: http://davidtemperley.com/wp-content/uploads/2015/11/temperley-mp99.pdf.

[22] Jeff Ullman. *Mining of Massive Datasets*. Chapter 3: LSH for Minhash Signatures. 2014. URL: http://infolab.stanford.edu/~ullman/mmds/ch3.pdf(visited on 06/05/2023).

[23] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser,and Illia Polosukhin. *Attention Is All You Need*. 2017. arXiv: 1706.03762[cs.CL].

[24] Anja Volk, F. Wiering,and Peter Van Kranenburg. “Unfolding the potential of computational musicology.” In: (Jan. 2011).

[25] Ziyu Wang, Ke Chen, Junyan Jiang, Yiyi Zhang, Maoran Xu, Shuqi Dai, Xianbin Gu,and Gus Xia. *POP909: A Pop-song Dataset for Music Arrangement Generation*. 2020. arXiv: 2008.07142[cs.SD].

[26] *youtube-dl-gui*. URL: https://github.com/oleksis/youtube-dl-gui(visited on 06/06/2023).

[27] Botao Yu, Peiling Lu, Rui Wang, Wei Hu, Xu Tan, Wei Ye, Shikun Zhang, Tao Qin,and Tie-Yan Liu. *Museformer: Transformer with Fine- and Coarse-Grained Attention for Music Generation*. 2022. arXiv: 2210.10349[cs.SD].

[28] Xueyao Zhang, Jinchao Zhang, Yao Qiu, Li Wang,and Jie Zhou. “Structure-Enhanced Pop Music Generation via Harmony-Aware Learning.” In: *Proceedings of the 30th ACM International Conference on Multimedia*. ACM, 2022. DOI: 10.1145/3503161.3548084. URL: https://arxiv.org/pdf/2109.06441.pdf.