Research project mid-term report
January 2025

Conditional Generation of Bass Guitar Tablature for Guitar Accompaniment in Western Popular Music

Student:
Olivier ANOUFA
Data Science Master 2

Supervisors:
Alexandre D'HOOGE
Ken DÉGUERNEL
Algomus team, CRIStAL

Abstract — The field of symbolic music generation has seen great advancements with the rise of transformer-based architectures. Addressing a specific need identified through a user study, we focus on developing AI tools to generate bass guitar tablatures conditioned on scores of other instruments in Western popular music. The bass guitar, a vital component of the rhythmic and harmonic sections in music, presents a unique challenge due to its dual role in providing structure and groove. Building upon the tablature notation, which simplifies the interpretation of music for stringed instruments, this work adopts modern encoding schemes to integrate tablature representation into transformer-based models. To this end, the project involves preprocessing a large dataset of music scores, and fine-tuning state-of-the-art transformer architectures. The generated tablatures will then be evaluated both numerically and qualitatively, with feedback from musicians.

Key-words — music information retrieval, conditional generation, guitar accompaniment, bass guitar, tablature;

1 Introduction

Natural language processing methods for the generation of symbolic music have experienced significant advancements in recent years. The transformer architecture, first introduced on text by Vaswani et al. in 2017[18], has been later used to generate scores for various instruments, in diverse styles and genres[12]. A user study conducted by Bacot et al.[3] showed a potential need for accompaniment generation tools for guitarists. The questionnaire was answered by 31 guitarist-composers, and 7 of them followed up with an interview. During the interviews, several guitarists answered that they would like to be able to generate bass guitar lines and drum parts without requiring familiarity with the instruments. Indeed, guitarists often resort to writing basic bass lines to accompany their compositions, and an AI tool could perform this functional task for them.

This need is the starting point for this project, proposed by the Algomus team, part of the CRIStAL laboratory. We focus on the conditional aspect of symbolic music generation, for an instrument that has not been thoroughly studied yet: the bass guitar. Specifically, the goal is to generate bass guitar tablatures given other instruments' scores, in the context of Western popular music. Our objective is to try several combinations of conditioning instruments and to evaluate the quality of the generated tablatures both numerically and with the help of musicians.

To better understand what is at stake in this challenge, we will begin by precisely defining the terms of the subject and the role of the bass guitar in the context of Western popular music. Since the human ear perceives low-frequency pulses more distinctly, the bass guitar is considered part of the rhythmic section of the band (together with the drums)[9]. However, the bass guitar also performs a harmonic — and sometimes melodic — role in the music, sustaining the lead instruments and adding groove to the composition. Its adaptability makes the bass guitar a crucial component across various genres of Western popular music.

Historically, tablatures have been used since the Renaissance period as a simplified notation system for string instruments like the lute. Originating around the 16th century, tablatures were an intuitive alternative to staff notation, allowing musicians to bypass the complexity of interpreting pitches. This system explicitly linked symbols to physical actions on the instrument, such as pressing specific frets or strings. In the 20th century, and with the rise of internet forums and music sharing platforms, tablatures became a popular way to share music scores among amateur musicians. First in ASCII format, tablatures were later digitalized and standardized in formats like GuitarPro (GP), further extending their use for learning, practice, and composition in genres ranging from classical to popular music[16]. Tablatures originally did not contain rhythmic information, which is why they are generally combined with scores. As a majority of guitarists learn many aspects of the songs they play by listening to them[7], tablatures are more of a prescriptive way to teach how to play a song that is already known by the musician. Tablatures are part of what is called symbolic music, distinguished from audio music. This project focuses on symbolic music, which corresponds to the written representation of music. This representation can be in the form of scores or tablatures, but no audio will be involved in the generation process.

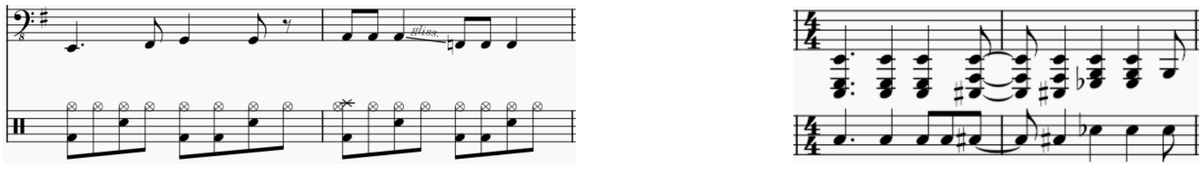


Figure 1: Rhythmic (left) and melodic (right) extracts in Karma Police (left) and Everything in its Right Place (right) by Radiohead.

Figure 1 shows a score extract of both a rhythmic and a melodic bass. In the extract on the left, the bass guitar plays a rhythmic role, accompanying the drums in its pulsating rhythm whereas on the right, the bass guitar plays a melodic role, following the lead guitar's melody at a lower pitch. In both these extracts, the bass guitar is essential to the song's structure. It adds grooves, harmonies and the low frequencies give the impression of a full sound.

Now that we have set the context and defined the terms, we will present the challenges we will face in generating bass guitar tablatures. At a high level, we first need to scrape and preprocess large datasets of music scores. Then, we need to design a computational representation of music that is adapted to the task of generating bass guitar tablatures. That is, a way to encode music scores in a meaningful way for the transformer architecture we will use. Concerning the generation, we will start by leveraging state-of-the-art models but will adapt and tune them to the task at hand.

More precisely, we will use the DadaGP dataset, which contains over 20 000 tablatures of Western popular music in GuitarPro format. This data will be used to train an adapted version of the state of the art model in conditional generation proposed by Makris et al. Originally applied to generate drums parts conditionnally to the rest of the rhythmic section of the band (rhythmic guitar and bass guitar), we will tune this model to generate bass guitar tablatures conditionnally different combinations of the other instruments of the band. We detail the current state of the art in terms of data availability and conditional generation models in the next section.

2 State of the art

Adapting the transformer architecture — originally developed for text processing — to symbolic music presents many challenges. Symbolic music datasets, unlike text corpus, are limited in both size and diversity, posing challenges for training robust models[12]. Tokenization must be tailored to represent pitch, duration, and dynamics, while attention mechanisms require adaptation to capture the hierarchical and temporal structure of music. Finally, data cleaning and preprocessing steps are critical to standardize music scores and ensure the compatibility with sequence models.

2.1 Data availability

Symbolic music datasets are the first necessary resource for training deep learning models, yet their availability and quality significantly vary across domains. In music composition and generation, datasets are often limited in diversity, especially when compared to text or image datasets. For bass guitar tablatures, the challenge is even more pronounced due to the niche nature of the instrument and the focus on other instruments in existing datasets.

The MIDI standard was predominant among symbolic music datasets for decades, allowing the development of resources like the Lakh MIDI dataset[14] and MAESTRO dataset[8]. However, while these datasets offer general-purpose symbolic music, they lack the specificity required for tasks involving tablatures or instrument-specific representation. The GuitarSet dataset[19] for example, focuses on acoustic guitar transcription but does not provide sufficient symbolic information for bass guitar. Similarly, the DadaGP dataset[16] addresses the need for multi-instrument symbolic music data in tablature format, but its emphasis is on rock and metal genres, which limits the diversity of bass guitar styles.

Bass guitar data, especially in tablature format, suffers from a lack of standardization and availability. Tablature files are often stored in private formats like GuitarPro[1] or as non-standardized text files, making it challenging to preprocess them for machine learning tasks. Moreover, rhythm and dynamics, critical elements in bass guitar playing, are frequently absent in publicly available tablatures, complicating their utility in generative tasks.

Efforts like DadaGP illustrate the potential of GuitarPro files repositoriess to create symbolic datasets that include bass guitar parts. Moreover DadaGP also provides a tokenized format inspired by MIDI encodings, offering a foundation for training sequence models. On the other hand, pre-trained models on broader datasets like MAESTRO or Lakh MIDI can be fine-tuned on smaller datasets, reducing the dependency on large volumes of task-specific data[13, 16]. For our project, we use the DadaGP dataset. Its very specific tokenization, among others, will be discussed in the next section.

[1]https://www.guitar-pro.com/fr/

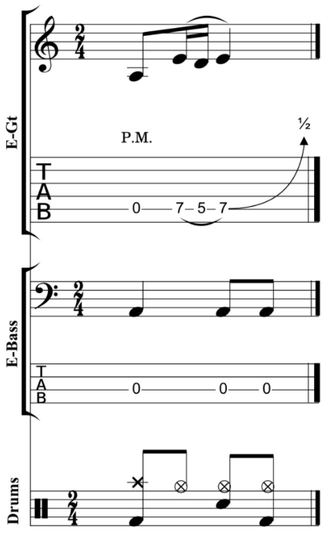
2.2 Tokenization

Tokenization in the context of deep learning music generation has been discussed by several previous works[2, 4, 10, 13, 16].

Much like tokenization in natural language processing (NLP), which breaks down text into words or subwords (e.g., "The quick brown fox" → ["The", "quick", "brown", "fox"]), music tokenization performs a similar process but focuses on musical features such as pitch, duration, and velocity. For example, a melody represented in symbolic form might be tokenized into a sequence like [C4, quarter note, velocity 64, G4, eighth note, velocity 70], making it easier to process for machine learning models.

In symbolic music information retrieval (MIR), tokenization strategies are generally classified into two categories: time-slice-based and event-based[12]. Time-slice-based tokenization divides music into fixed-time intervals, such as 16th notes, which can be represented in formats like multi-hot vectors, capturing simultaneous notes at each time slice. Event-based tokenization, on the other hand, focuses on specific musical events, such as a note being played or a measure starting, often using formats like MIDI, which encode music as a series of events. This approach can involve elementary tokens, which represent individual features like pitch or duration, or composite tokens that aggregate multiple features into one token, providing a more compact representation. Notably, tokenization strategies like REMI (Revamped MIDI-derived events[11]) and MIDI-like tokenization allow for consistent representation of rhythmic and pitch elements while addressing complexities like polyphony and multiple tracks in multi-instrument music[4, 12].

The DadaGP tokenization format adopts an event-based approach, similar to other music generation models, by representing musical events as discrete tokens. It uses a Python script with PyGuitarPro[1] to convert GuitarPro files into a tokenized sequence, beginning with metadata tokens such as artist, downtune, tempo, and start. Pitched instrument notes are encoded with a combination of tokens representing instrument, note, string, and fret, while rests are denoted with a separate token structure. For drumset sounds, a specific tokenization scheme using MIDI percussion maps is employed, where each drum sound is represented by a unique token (e.g., `drums:note:36` for a kick drum). The system also uses wait tokens to separate notes and rests, enabling the model to infer note durations without the need for explicit note-off tokens. This approach ensures that new notes automatically silence previous ones, except in cases where ghost notes or let-ring effects are involved. Additionally, the tokenization format records changes in musical structure, such as new measures, tempo shifts, and note effects, with each change being represented as a distinct token. However, the tokenization does not encode dynamics or articulations. These are supposed to be inferred by the user after generation.



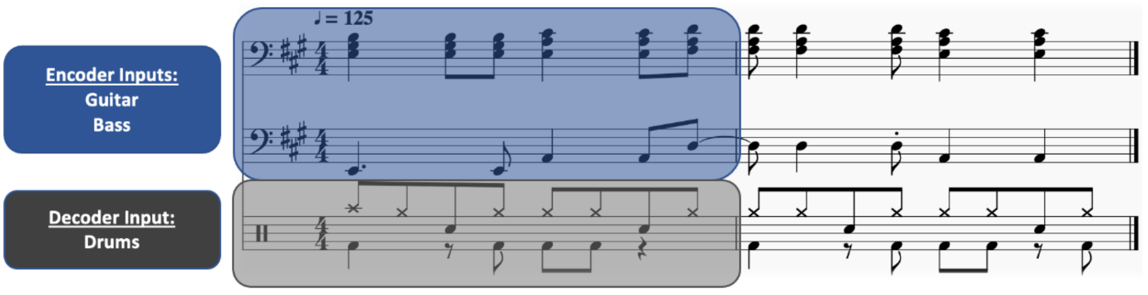
```
artist:textotab
downtune:0
tempo:190
start
new_measure
distorted0:note:s5:f0
nfx:palm_mute
bass:note:s4:f0
drums:note:35
drums:note:57
wait:480
distorted0:note:s5:f7
nfx:hammer
drums:note:46
wait:240
distorted0:note:s5:f5
nfx:hammer
wait:240
distorted0:note:s5:f7
nfx:bend:type1
bass:note:s4:f0
drums:note:46
drums:note:38
wait:480
bass:note:s4:f0
drums:note:46
drums:note:35
wait:480
end
```

Figure 2: Example measure with a distorted guitar, bass and drums in GuitarPro's graphical user interface (left), and its conversion into token format (right). Figure taken from [16]

Figure 2 illustrates the tokenization process for a measure in a GuitarPro file, showing the conversion of a musical event into a tokenized sequence. We notice the three first tokens are the metadata tokens. "start" token marks the beginning of the song. Afterwards, each measure is announced by a "new_measure" token. In between, the instruments' notes are encoded using the tablature notation for the guitars. For instance the first note played here is `distorted0:note:s5:f0` (string 5, fret 0). It is followed by a nfx token "`nfx:palm_mute`", which is a note effect token that applies to the previous note. Finally, the duration of the note is given by the wait token "`wait:480`" (480 ticks in GuitarPro correspond to an eighth note). If no tokens of a specific instrument are present between two wait tokens, it means that the instrument keeps on playing the same note. This is the case of the bass in the example. Its first quarter note lasts for 960 ticks, therefore we have to look after the three first wait tokens (480 + 240 + 240) before seeing a new token for the bass.

Other tokenization strategies have been proposed to address the limitation of input size of transformer-based models and condense all the information about a note in one place. This is the case of the Compound tokenization model developed by Makris et al. Their tokenization model introduces a compound word (CP) representation, inspired by previous CP-based approaches[10]. Unlike traditional tokenization strategies such as MIDI-like or REMI, which encode music as a linear sequence of tokens, CP groups multiple features of a musical event into a single "super token" or "word". This n-dimensional representation significantly reduces sequence length and improves computational efficiency in Transformer-based architectures[13].

In this model, the CP representation is adapted for an Encoder-Decoder architecture to condition drum generation on accompanying rhythm tracks, such as bass and guitar. The Encoder processes a 5-dimensional CP representation that includes rhythm track features and high-level musical information like tempo and time signature, which are known to influence the complexity and density of drum tracks. Both the Encoder and Decoder adopt bar-based segmentation, consistent with existing CP and REMI representations. By combining rhythm section inputs and high-level parameters, this tokenization approach aligns with the fundamental musical principles of contemporary western music's rhythm section[13]. As said previously, our work will use the DadaGP tokenization. However, the nature of the generation task we want to perform is very similar to the one of Makris et al. in the sense that bass guitar and drums are part of the same rhythm section.



Onset	Group	Type	Duration	Value
0.0	High-level	Bar	Bar	Bar
0.0	High-level	Tempo	Bar	125
0.0	High-level	Time Signature	Bar	4/4
0.0	Bass	Note	1.5	NaN
0.0	Guitar	Chord	1.0	NaN
...	...	...	...	...
3.5	Guitar	Chord	0.5	NaN
0.0	High-level	Bar	Bar	Bar

Onset	Drums
Bar	Bar
0.0	35
0.0	49
0.5	42
1.0	38
...	...
3.5	42
Bar	Bar

Figure 3: Example of a CP representation used for training. Figure taken from [13]

Figure 3 shows an example of a CP representation used for training in the model developed by Makris et al. We notice that the rhythmic section, composed of a rhythmic guitar and a bass (top line on the figure), constitutes the conditioning part of the model. Their representation as tokens encode the duration and type of the notes played, but not the pitch of the notes as it is not relevant for the drum generation task. At the bottom line, the drum part's encoding excludes all information about velocities and duration. Drums notes and are only represented by their onset in the bar and the drum component played.

2.3 Conditional generation models

Our first idea was to use the GuitarCTRL model developed by Sarmento et al.[17]. The GTR CTRL model utilizes a Transformer-XL architecture[5], which improves on the vanilla Transformer by introducing recurrence and modified positional encodings, enabling long-term dependency learning. This model includes two key control types: instrumentation (inst-CTRL) and genre (genre-CTRL). For instrumentation, tokens marking the start and end of each instrument are inserted into the header, guiding the model to generate music for specified instruments, such as bass and drums or distorted guitar and drums. Genre control tokens are similarly placed at the beginning of the sequence, with a dataset covering over 200 songs per genre, including Metal, Rock, Punk, Folk, and Classical. The model uses various prompts at inference, ranging from full-prompt (including two measures plus the genre token) to empty-prompt (only the genre token). This model was used to generate a baseline result for our project, conditioning the generation to bass guitar only.

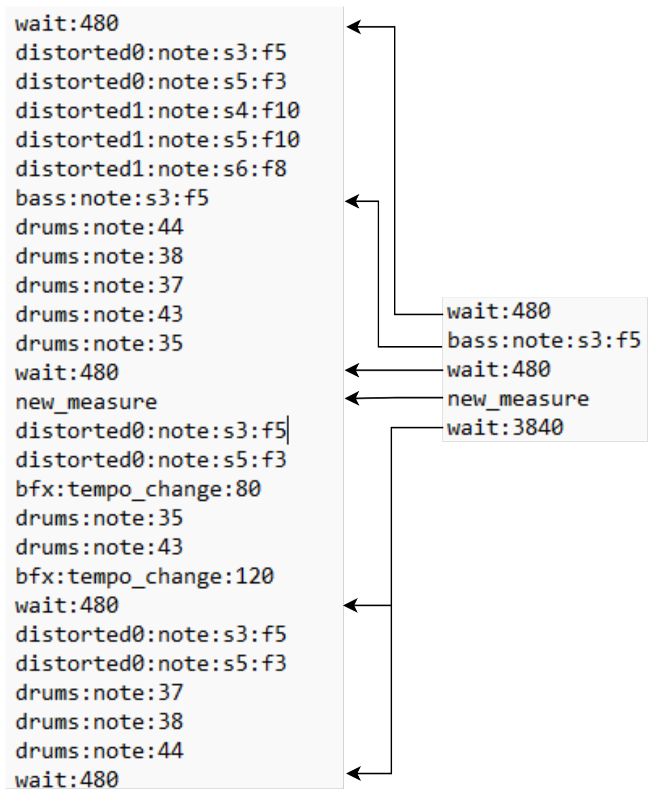
The model that is the current state of the art in conditional drum generation is the one by Makris et al.[13]. It employs an Encoder-Decoder architecture with a BiLSTM Encoder and a Transformer Decoder utilizing Relative Global Attention. The Encoder is made of several BiLSTM layers. It processes Compound Representation (CP) inputs to generate a latent variable z that encodes high-level features of the input and conditional tracks. BiLSTM (Bidirectional Long Short-Term Memory) networks were first introduced by Graves et al. in 2005[6]. It is a type of recurrent neural network (RNN) that processes data in both forward and backward directions. This allows the model to consider both past (previous time steps) and future (upcoming time steps) context in the sequence. The Decoder consists of self-attention layers and feed-forward layers. It combines z with previous timestep embeddings to predict drum onsets and pitches.

The model uses input sizes that are different from the ones we may use with our tokenization. However we are interested in the ability of the model to generate conditionnally throughout the whole sequence. In comparison, GTR CTRL only conditions the generation at the beginning of the sequence. One of the issue we encountered when generating bass guitar was that other instruments ended up being generated further in the sequence. To avoid this we tried setting the probability of predicting tokens from any other instruments to 0, but even then the bass lines generated were not coherent. This model is simply not suited for our task which explains our motivations to adapt the BiLSTM model.

3 Data preprocessing

3.1 DadaGP token preprocessing

The DadaGP dataset provides token files containing information for all instruments in a score. As we want to explore various conditioning combinations for tablature generation, we developed a function to extract tokens specific to selected instruments. We considered several possible ways to perform this extraction: runnning DadaGP's token processing script for each instrument, or filtering the tokens from the complete token text file based on the instrument name. Opening, parsing and looping on the GuitarPro file using pygp is needed for the DadaGP token processing script and is very time-consuming, much more than processing txt files. We recall that DadaGP comprises 26,181 GuitarPro files. Using our algorithm it only took 5min30 to process all the files and generate a copy of DadaGP containing token files of bass guitar only. To perform this operation, we leverage the fact that tokens of a given instrument start with the instrument name followed by a colon, e.g., `bass:` for the bass guitar. However we must be careful not to miss the potential note effect tokens that are not instrument-specific, but come right after the token they are related to. After extracting those tokens and the general tokens (metadata tokens and wait tokens), we sum the potential consecutive wait tokens that were previously separated by notes from instruments that are no longer present.



```
wait:480
distorted0:note:s3:f5
distorted0:note:s5:f3
distorted1:note:s4:f10
distorted1:note:s5:f10
distorted1:note:s6:f8
bass:note:s3:f5
drums:note:44
drums:note:38
drums:note:37
drums:note:43
drums:note:35
wait:480
new_measure
distorted0:note:s3:f5
distorted0:note:s5:f3
bfx:tempo_change:80
drums:note:35
drums:note:43
bfx:tempo_change:120
wait:480
distorted0:note:s3:f5
distorted0:note:s5:f3
drums:note:37
drums:note:38
drums:note:44
wait:480
```

```
wait:480
bass:note:s3:f5
wait:480
new_measure
wait:3840
```

Figure 4: Example of an extraction of bass guitar token in The Strokes - Last Nite

An example of token extraction is shown in Figure 4. This figure also shows the concatenation of wait tokens. At the end of the song, the bass guitar stops playing before the other instruments. We can notice the several `wait:480` tokens that are summed up to a single `wait:3840` token.

3.2 Rhythmic guitar identification

The algorithm we just presented allows us to extract tokens for any instrument in the DadaGP dataset. In typical rock band instrumentation, rhythmic guitars complement bass guitars with their repetitive chord patterns, providing harmonic and rhythmic structure at a higher pitch [15]. This characteristic makes them particularly relevant for conditioning bass generation tasks, therefore, our first focus is on conditioning bass guitar tablature generation using the rhythmic guitar.

To identify rhythmic guitar parts in the DadaGP dataset, we implemented the method proposed by Régnier et al. [15], which uses features describing notes and chords at the bar level, along with corresponding tablatures. Their model outputs a prediction between 0 and 1 for each bar. The closest to 1, the more likely the part is a lead on the bar, and the closest to 0, the more likely it is a rhythmic guitar part on the bar. In their paper, they consider that a score below 0.5 correspond to a rhythmic guitar parts. We wanted to select and condition the generation on only one rhythmic guitar part, the "most" rhythmic. Therefore, we applied the 0.5 threshold to the predictions and selected the part with the highest proportion of rhythmic bar over the course of the song. Implementing this method required adapting their code to the DadaGP dataset. Significant effort was spent mapping the identified rhythmic guitar parts to their respective names in the DadaGP files, as the dataset renames instruments, whereas the identification tool relies on GuitarPro part names. Thanks to this work, we were able to extract two more datasets: one containing the rhythm guitar tokens and the other containing both the bass and the rhythmic guitar tokens.

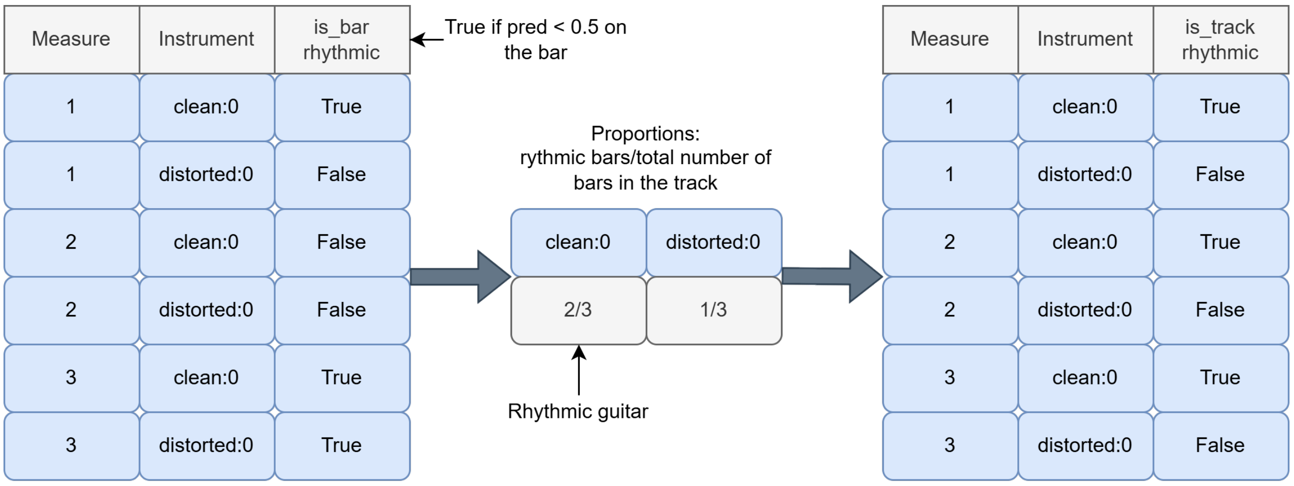


Figure 5: Example of rhythmic guitar selection after applying the method of Régnier et al.

Figure 5 shows an example of the selection of the rhythmic guitar part. After retrieving the instrument with the highest number of rhythmic bars over the song, we generate a column `is_track_rhythmic` that is filled with 1 (or True) for the rhythmic guitar and 0 for the other instruments.

3.3 Post processing

To avoid too long sequences, we chose to extract samples of 16 measures from the songs. We extract sequences with a stride of 8 measures, which allows us to have a good overlap between the sequences. This will help the model to learn the transitions, and understand the various possible contexts around a given sequence. Sequences also have at most 800 tokens and at least 50 tokens. Thresholds were set to avoid training on sequences that are either too long or that does not contain rythmic guitar or bass. These thresholds are not applied on the complete sequences containing both the tokens of the rhythmic guitar and the bass.

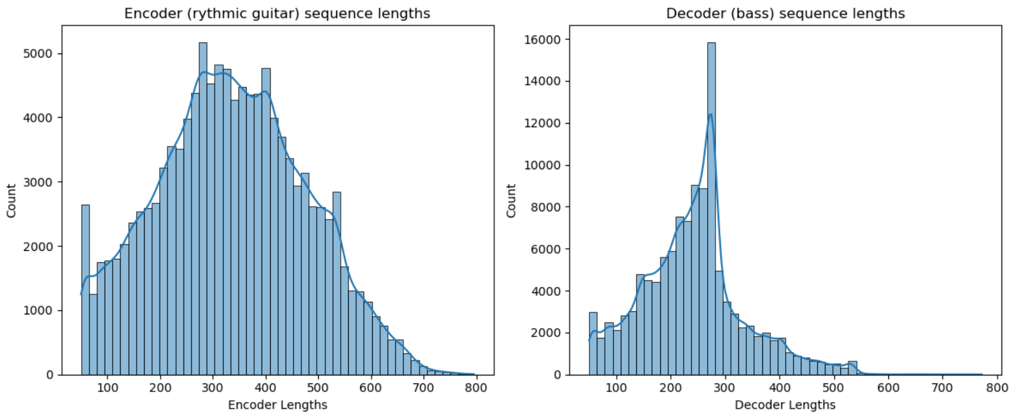


Figure 6: Distribution of sequence lengths in the training dataset

Figure 6 shows the distribution of sequence lengths in the training dataset. The majority of the sequences are between 200 and 400 tokens long for the rythmic guitar and between 100 and 300 tokens long for the bass.

Those sequences fill in a dictionary with keys `Encoder_RG`, `Decoder_Bass` and `All_Events`. For instance `dict['Encoder_RG'][i]` contains the tokens of the rhythmic guitar for the i-th sequence. `dict['All_Ever` contains the tokens of bass guitar and rhythmic guitar for the i-th sequence.

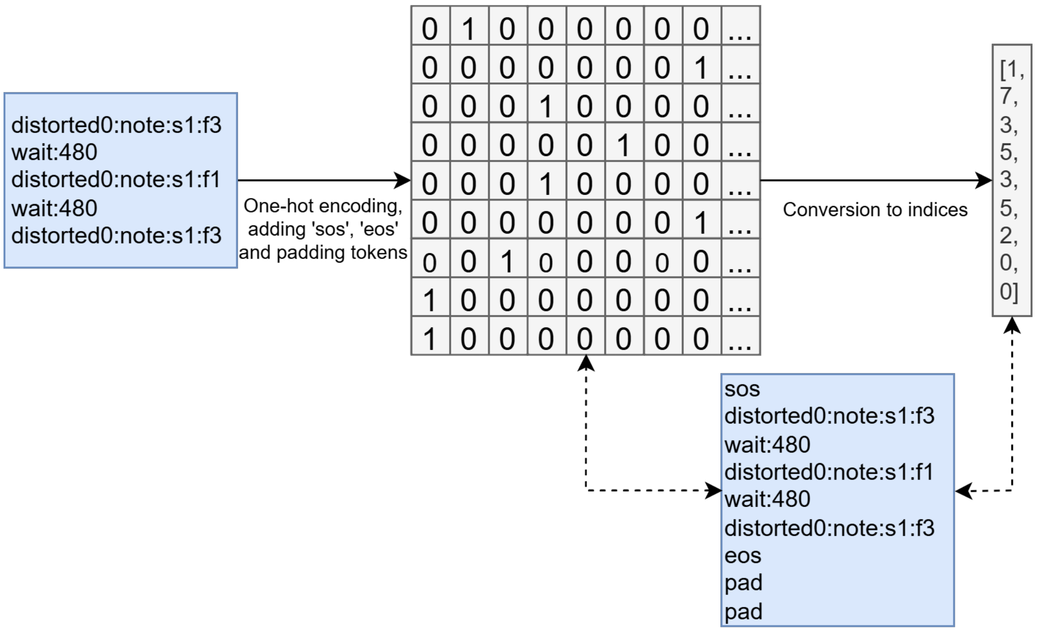


Figure 7: Conversion from a txt token sequence to indices

The sequences are then processed as described in figure 7. They go through a one-hot encoding process, then we add 'start of sequence' and 'end of sequence', and padding tokens after the 'eos' token to reach a fixed length of approximately 800 tokens. The final dictionary of sequences as indices is finally split into training, validation and test sets with proportions 0.8, 0.1 and 0.1 respectively. We generated 118 167 sequences in total, which leads to 94 533 sequences in the training set and 11 817 in both the validation and test sets.

4 Models

4.1 Baseline predictions

We began by establishing a baseline model for bass guitar tablature generation. Initially, we utilized a pre-trained checkpoint from the DadaGP paper [16] without additional training.

To generate bass-specific outputs, we experimented with various prompts. First, we provided a single bass guitar note as the initial token, however the model quickly incorporated tokens from other instruments. This is an issue documented in the GuitarCTRL paper[17] where they used a metric called the UIP score (Unpromped Instrument Presence) to evaluate the model's ability to generate a specific combinations of instruments. To constrain the output to bass tokens, we modified the logits of non-bass instruments by setting them to $-\infty$.

An example of a bass tablature generated this way is shown in Figure 8. While this forced the model to generate only bass tokens, the output quality was poor, featuring repetitive phrases, harmonically incorrect notes, or complete aberrations. This was expected, as the model was not trained explicitly for bass only generation. Unfortunately, we cannot provide a quantitative evaluation of the generated tablatures, as we have not yet defined an evaluation metric.

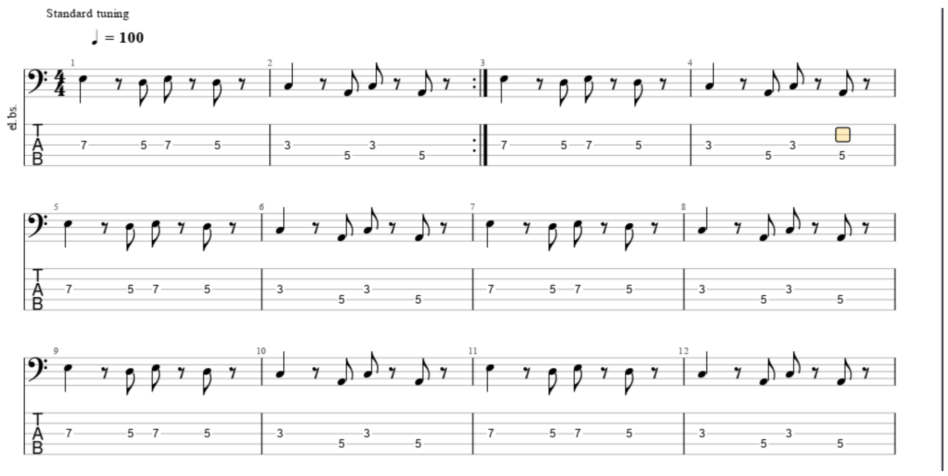


Figure 8: Example of a bass generation from the baseline model

4.2 BiLSTM - Transformer model

This section present the results we get using a model adapted from the work of Makris et al. [13]. Our data was adapted to fit the model's input requirements and the model was also slightly modified to fit our task.

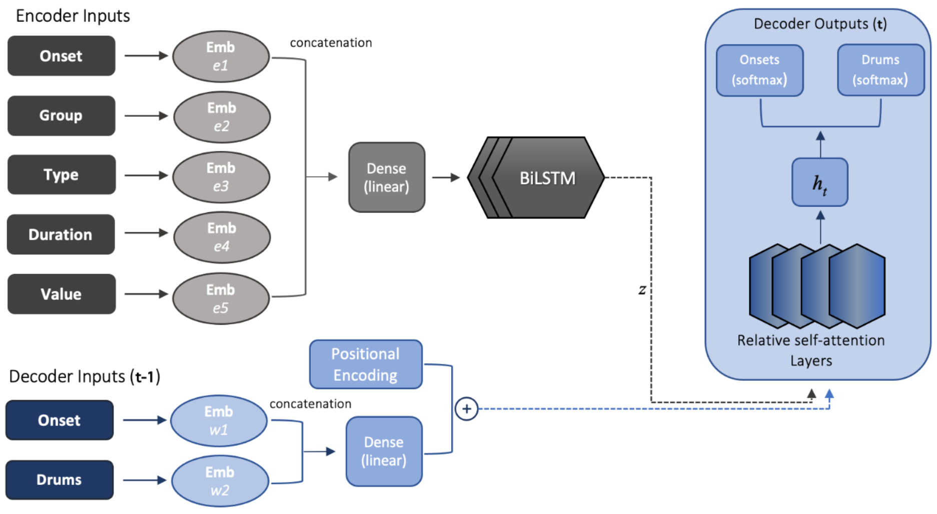


Figure 9: Makris et al. model architecture, taken from their paper [13]

The figure 9 shows the architecture of the model proposed by Makris et al. We have presented the compound representation in the state of the art. This representation concatenates all the parameters necessary to describe a note in a single vector (onset, duration, pitch etc.).

Figure 10 shows the model adapted to our task. In our case, we have chosen to use DadaGP tokenization which is a different representation. On the encoder side, the five embeddings concatenated in Makris et al. are replaced by a single embedding in our case. On the decoder side, the compound word is only made of two inputs because drums notes need much less parameters than other instruments. They are fully described by their onset and a value indicating the drum type. This compound representation is also replaced by a single embedding in our case.

The embeddings are then passed through a dense layer. The encoder input (rhythmic guitar) is passed through a BiLSTM layer. Positional encodings are added to the decoder input and the output of the BiLSTM layer. All these elements are then passed through a transformer layer with self-attention mechanism that outputs

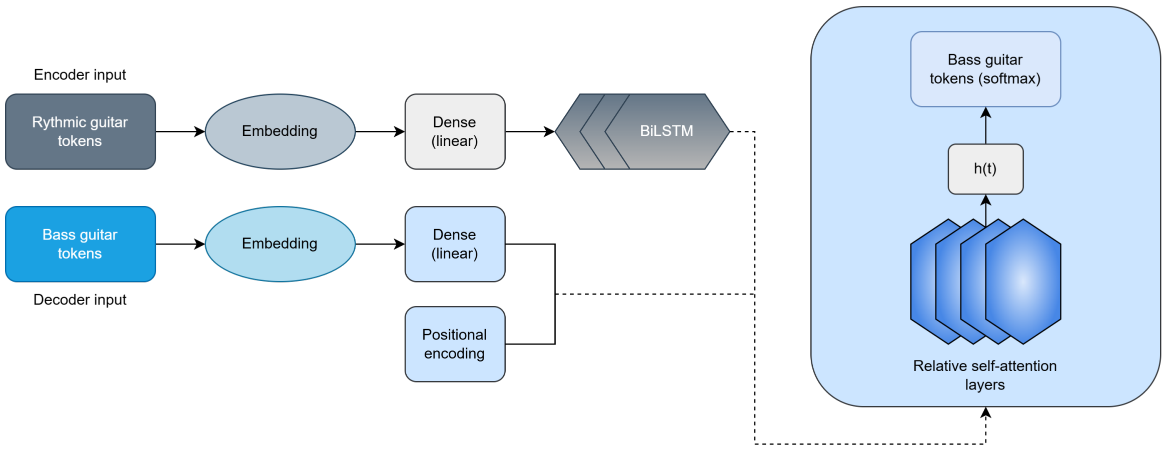


Figure 10: Makris et al. model adapted to our task

an array of vocabulary size with the logits of the next token. These logits are then passed through a softmax layer to get the probabilities of the next token. All these layers lead to a model of 20 881 514 parameters.

To evaluate our model's performance, we use a loss function based on categorical cross-entropy and an accuracy metric designed for sequence modeling. The loss function measures how well the predicted probability distribution aligns with the true tokens and is defined as:

$$\mathcal{L} = -\sum_{t=1}^{T}\sum_{k=1}^{K} y_{t,k} \log \hat{y}_{t,k}$$

where T is the sequence length, K is the vocabulary size, $y_{t,k}$ is a one-hot encoded ground-truth token at time step t, and $\hat{y}_{t,k}$ is the predicted probability for token k.

Our accuracy metric computes the proportion of correctly predicted tokens in a sequence. Both functions incorporate a masking mechanism to handle padding tokens and remove their contribution to the calculations.

4.3 Training

Training was done on GPU with 200 epochs, an early-stopping criterion of 5 epochs without improvement on the validation loss, Adam optimizer with the following hyperparameters: $\alpha = 5e^{-4}$, $\beta_1 = 0.9$, $\beta_2 = 0.98$, $epsilon = 1e^{-9}$, and a batch size of 32.

The training stopped at epoch 37 because the validation loss did not decrease anymore for 5 epochs. We reached 0.97 accuracy on the validation set with 0.14 loss.

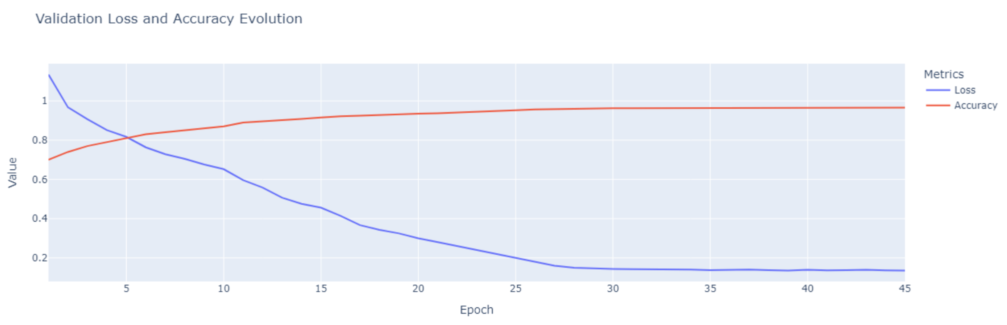


Figure 11: Evolution of accuracy and loss during the training

Figure 11 shows the evolution of accuracy and loss during training. The accuracy increases rapidly during the first epochs, then stabilizes around 0.90.

5 Results, discussion and perspectives

5.1 Results

We did not have time to implement quantitative evaluation metrics beyond accuracy and loss, so we rely on our expertise to assess the quality of the generated tablatures. Inference was conducted on the entire test set of 11,817 sequences. The model achieved an accuracy of [VALUE TO FILL] on those sequences, with a corresponding loss of [VALUE TO FILL]. These values are very close to those observed on the validation set.

In this section, we showcase a selection of results from our experiments. The examples were carefully chosen to demonstrate the model's ability to generate tablatures across different musical genres and diverse rhythmic guitar patterns. We manually selected rhythmic guitar parts and extracted their corresponding tokens. These examples originate from Songsterr [2] but are not part of the training dataset. However, they belong to genres that are well-represented in DadaGP: various types of rock, metal and jazz funk. Our work is the first to generate bass tablatures conditioned on rhythmic guitar parts, so we do not have other models to compare our results to.

5.2 Examples

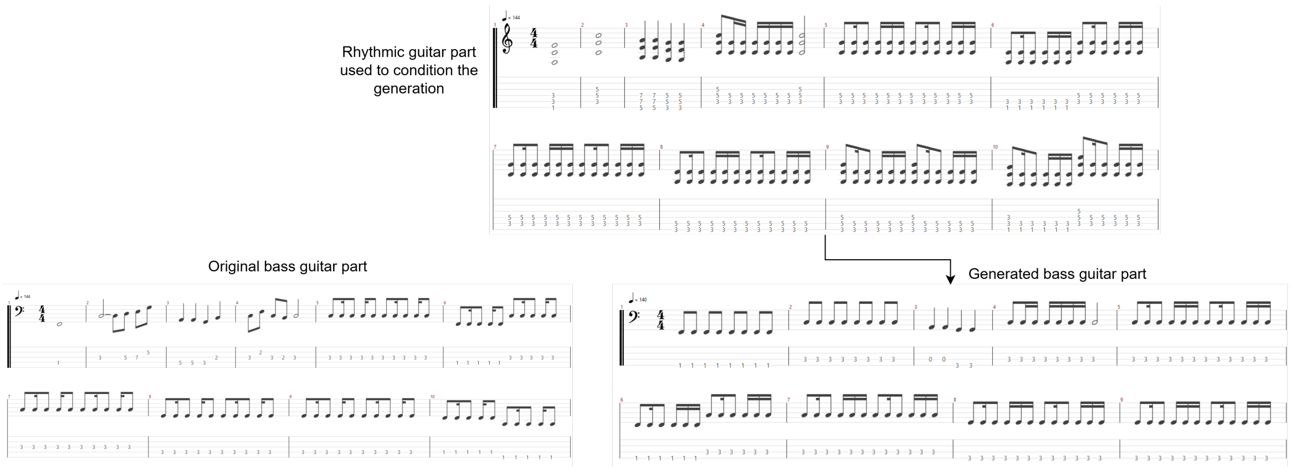
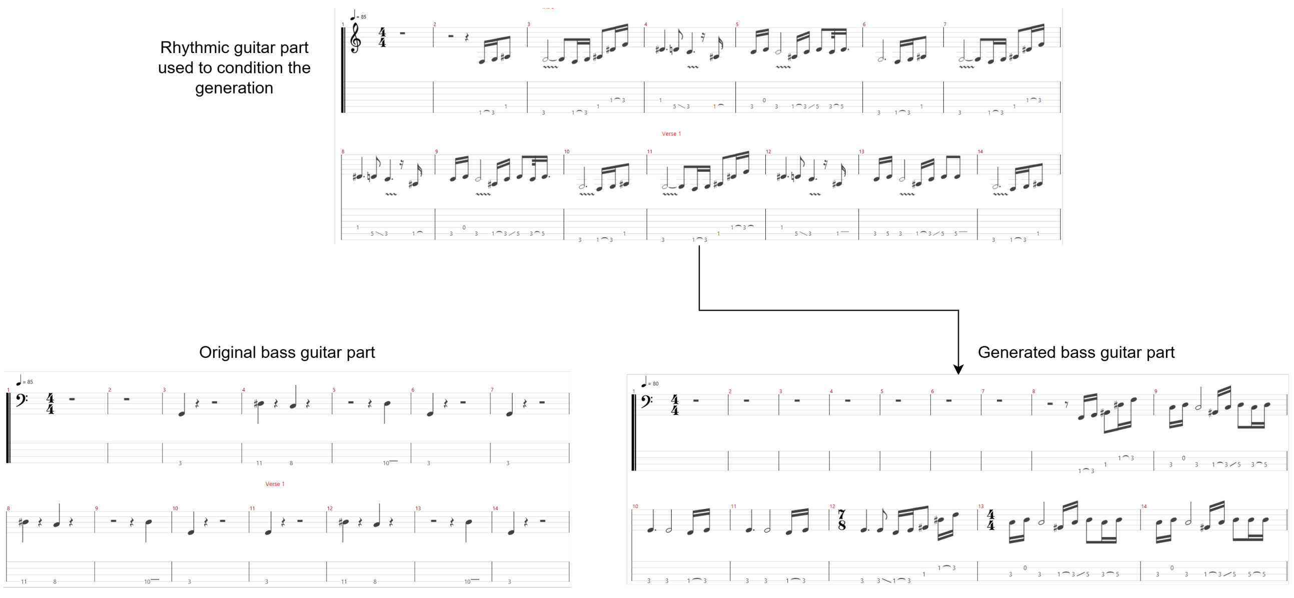

Rhythmic guitar part used to condition the generation
Original bass guitar part
Generated bass guitar part

Figure 12: Example of 9 measures generated by the model based on the rhythmic guitar part of the song "Ricard Peinard" by Ultra Vomit

Figure 12 shows an excerpt of bass tablatures generated by the model in the metal genre. Bass guitar in metal music often plays a supportive role, doubling the rhythm guitar part at a lower pitch. We also display the rhythmic guitar part and the original bass part. Several criterion can be qualitatively used to evaluate the model's performance. First, we can notice that the generated part is harmonically coherent with the rhythmic guitar part: on the first measure the bass plays several F notes, while the rhythmic guitar plays a chord composed of F, C and G notes. Rhythmically, the generated part either copies the rhythmic guitar or plays a simpler pattern such as a succession of eighth notes. This is a correct outcome in the context of metal music, but needs to be further evaluated in other genres. Finally, one may notice that the generated part is identical to the original bass part on measures 5 to 8.

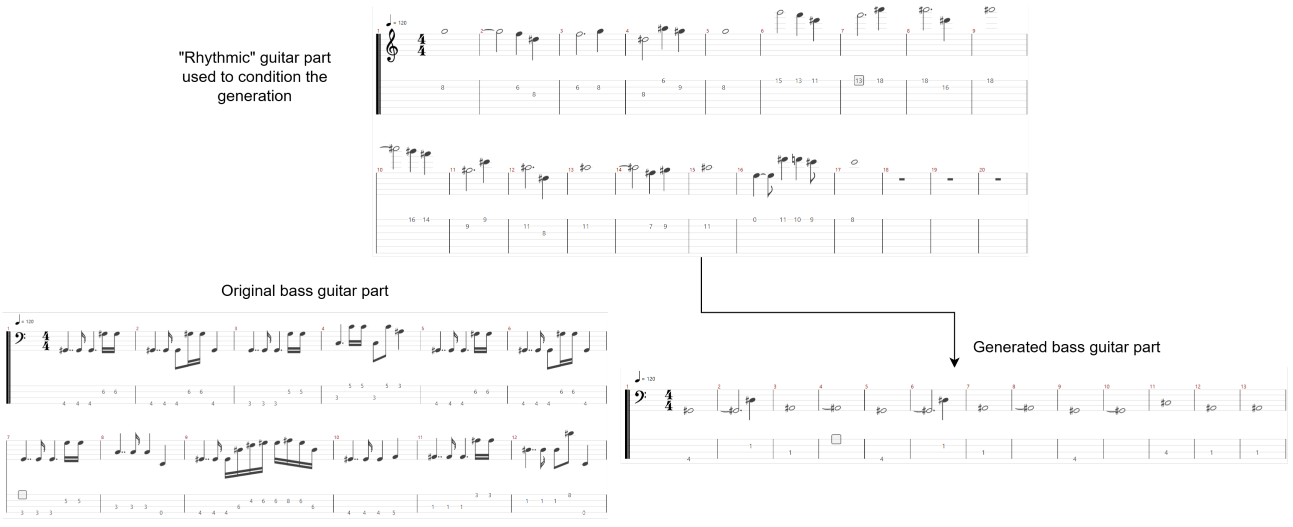
[2]https://www.songsterr.com/



Rhythmic guitar part used to condition the generation
Verse 1
Original bass guitar part
Verse 1
Generated bass guitar part

Figure 13: Example of 9 measures generated by the model based on the rhythmic guitar part of the song "Do I Wanna Know" by Arctic Monkeys

Figure 13 shows an excerpt of bass tablatures generated by the model in the indie rock genre. Bass guitar's role in indie rock music is hard to define due to the genre's diversity. However, this example remains particularly interesting as it demonstrates the model's tendency to copy the rhythmic guitar part even when very simple parts would be more appropriate. Here the original bass part is basic yet essential, providing groove and structure to the song using quarter notes at specific onsets. Nonetheless, the generated part remains harmonically and rhythmically coherent with the rhythmic guitar part. This example also shows a recurrent issue in the model's inability to follow a time signature. Here measure 12 becomes a 7/8 instead of a 4/4 (the model forgot an eighth note in the measure).



"Rhythmic" guitar part used to condition the generation
Original bass guitar part
Generated bass guitar part

Figure 14: Example of 9 measures generated by the model based on the rhythmic guitar part of the song "Saturday Night" by Herbie Hancock

Figure 14 shows an excerpt of bass tablatures generated by the model in the jazz funk genre. Bass guitar in jazz funk music often plays a more melodic role, with complex rhythmic patterns and harmonies. Moreover, there are no rhythmic guitar parts strictly speaking in jazz funk music, so the model generates using a more melodic guitar part. As expected, the generated part is way simpler than the original bass part. It is meant to be an accompaniment to the guitar part, not a melodic part. Moreover, jazz funk music often uses complex harmonies, so the model has a hard time following the harmonic progression. This genre can be considered as a limit case for the model. As it was trained on an agglomerate of genres, it is not able to generate coherent jazz funk bass lines.

To summarize, the model has great potential in generating bass tablatures of accompaniment that are harmonically and rhythmically coherent with the rhythmic guitar part. However, it struggles to adapt to the specificities of each genre and can still make rhythmical mistakes. We tried to build an adaptable model that could generate bass lines for a wide range of styles, but it would be interesting to analyze the model's performance on a specific genre if trained on said genre only.

5.3 Perspectives

With additional time, several improvements could be made to the model architecture and tuning, data processing, and evaluation. One potential enhancement is simplifying the model by removing one of the dense layers in the embedding process, which could reduce complexity without significantly impacting performance. Additionally, experimenting with compound word tokenization as an alternative to the current DADAGP method might yield more meaningful representations.

Beyond architectural changes, a user study could be conducted to assess the quality of the generated bass lines, providing valuable insights into the model's musical coherence. Finally, evaluating the model's environmental impact using Green AI metrics, specifically by measuring the number of floating point operations (FLOPs), would contribute to a better understanding of its computational efficiency.

References

[1] Sviatoslav Abakumov. *PyGuitarPro — PyGuitarPro 0.9.3 documentation*. 2014.

[2] Manvi Agarwal, Changhong Wang, and Gaël Richard. “Structure-informed Positional Encoding for Music Generation”. In: IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). 2024.

[3] Baptiste Bacot, Louis Bigo, and Benoît Navarret. “Tablature software and popular music composition: a user study and perspectives on creative algorithmic tools”. 2025.

[4] Jules Cournut et al. “Encodages de tablatures pour l’analyse de musique pour guitare”. In: *Journées d’Informatique Musicale (JIM 2020)*. 2020.

[5] Zihang Dai et al. “Transformer-XL: Attentive Language Models beyond a Fixed-Length Context”. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. ACL 2019. Ed. by Anna Korhonen, David Traum, and Lluís Màrquez. Association for Computational Linguistics, 2019, pp. 2978–2988.

[6] A. Graves and J. Schmidhuber. “Framewise phoneme classification with bidirectional LSTM networks”. In: *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.* International Joint Conference on Neural Networks 2005. Vol. 4. IEEE, 2005, pp. 2047–2052.

[7] Lucy Green. *How Popular Musicians Learn: A Way Ahead for Music Education*. 2001.

[8] Curtis Hawthorne et al. “Enabling Factorized Piano Music Modeling and Generation with the MAESTRO Dataset”. In: International Conference on Learning Representations. 2018.

[9] Michael J. Hove et al. “Superior time perception for lower musical pitch explains why bass-ranged instruments lay down musical rhythms”. In: *Proceedings of the National Academy of Sciences* 111.28 (2014), pp. 10383–10388.

[10] Wen-Yi Hsiao et al. “Compound Word Transformer: Learning to Compose Full-Song Music over Dynamic Directed Hypergraphs”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 35.1 (2021). Number: 1, pp. 178–186.

[11] Yu-Siang Huang and Yi-Hsuan Yang. “Pop Music Transformer: Beat-based Modeling and Generation of Expressive Pop Piano Compositions”. In: *Proceedings of the 28th ACM International Conference on Multimedia*. MM ’20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 1180–1188.

[12] Dinh-Viet-Toan Le et al. *Natural Language Processing Methods for Symbolic Music Generation and Information Retrieval: a Survey*. 2024.

[13] Dimos Makris et al. “Conditional Drums Generation Using Compound Word Representations”. In: *Artificial Intelligence in Music, Sound, Art and Design*. Ed. by Tiago Martins, Nereida Rodríguez-Fernández, and Sérgio M. Rebelo. Cham: Springer International Publishing, 2022, pp. 179–194.

[14] Colin Raffel. “Learning-Based Methods for Comparing Sequences, with Applications to Audio-to-MIDI Alignment and Matching”. In: (2016).

[15] David Régnier, Nicolas Martin, and Louis Bigo. “Identification of rhythm guitar sections in symbolic tablatures”. In: (2021).

[16] Pedro Sarmento et al. “DadaGP: a Dataset of Tokenized GuitarPro Songs for Sequence Models”. In: *Proceedings of the 22nd International Society for Music Information Retrieval Conference*. 2021. URL: https://archives.ismir.net/ismir2021/paper/000076.pdf.

[17] Pedro Sarmento et al. “GTR-CTRL: Instrument and Genre Conditioning for Guitar-Focused Music Generation with Transformers”. In: *Artificial Intelligence in Music, Sound, Art and Design*. Ed. by Colin Johnson, Nereida Rodríguez-Fernández, and Sérgio M. Rebelo. Springer Nature Switzerland, 2023, pp. 260–275.

[18] Ashish Vaswani et al. “Attention is All you Need”. In: *Advances in Neural Information Processing Systems*. Vol. 30. Curran Associates, Inc., 2017.

[19] Qingyang Xi et al. “Guitarset: A Dataset for Guitar Transcription”. In: (2018).