

L^AT_EX Template for the Conference on AI Music Creativity (AIMC) 2025

Olivier Anoufa*

Department of Music
Ecole Centrale de Lille
Pittsburgh, PA 15213

`olivier.anoufa@centrale.centralelille.fr`

Coauthor

Affiliation
Address
email

Coauthor

Affiliation
Address
email

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 conducted on guitar players, 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 crucial 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, the most popular way to share music among guitarists, 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. Due to the project being the first of its kind and its creative nature, the evaluation of the generated tablatures is mainly done qualitatively.

*Use footnote for providing further information about author (webpage, alternative address)—*not* for acknowledging funding agencies.

1 Introduction

Natural language processing methods for the generation of symbolic music have experienced significant advancements in recent years. The transformer architecture, first introduced for text by Vaswani et al. in 2017², has been later used to generate scores for various instruments, in diverse music genres³. A user study conducted by Bacot et al.⁴ 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 instruments. Indeed, guitarists often resort to writing basic bass lines to accompany their compositions, and an AI tool could help by performing this functional task for them. This need is the starting point for this project, proposed by the Algomus team, part of the CRISAL 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, let us 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)⁵. 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.

Learning score notation can be a daunting task for beginners, especially for string instruments like the guitar. 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, 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 GuitarPro², further extending their use for learning, practice, and composition in genres ranging from classical to popular music⁶. 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 them⁷, 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.

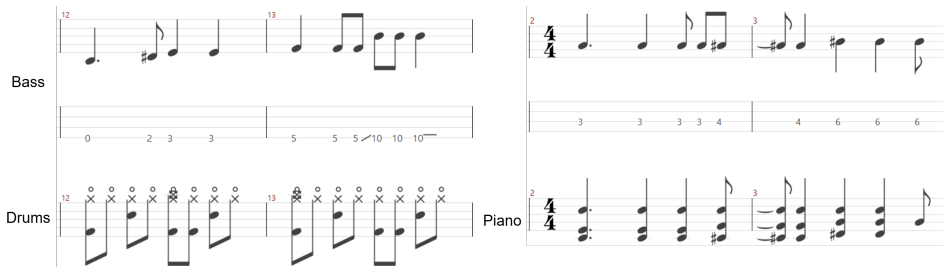


Figure 1: Rhythmic (left) and melodic (right) excerpts in Karma Police (left) and Everything in its Right Place (right) by Radiohead.

²<https://www.guitar-pro.com>

Figure 1 shows a score excerpt of both a rhythmic and a melodic bass³. In the excerpt 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 piano’s lead melody at a lower pitch. In both these excerpts, 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. Before tackling the task, we first need to retrieve 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 [?]. 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 [?] and MAESTRO dataset [?]. 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 [?] for example, focuses on acoustic guitar transcription but does not provide sufficient symbolic information for bass guitar. Similarly, the DadaGP dataset [?] 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 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 repositories 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

³All the work done is publicly accessible, and the excerpts are available as mp4 or mp3 on the GitHub repository: <https://github.com/0litHub/guitar-tab-gen>

dependency on large volumes of task-specific data ?? . For our project, we use the DadaGP dataset. Its very specific tokenization, among others, will be discussed in the next section.

2.2 Tokenization

Tokenization in the context of deep learning music generation has been extensively discussed in several previous works ????? . 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 ?. 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 ?) 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 ??.

The DadaGP tokenization format adopts an event-based approach, similar to other music generation models, by representing musical events as discrete tokens.

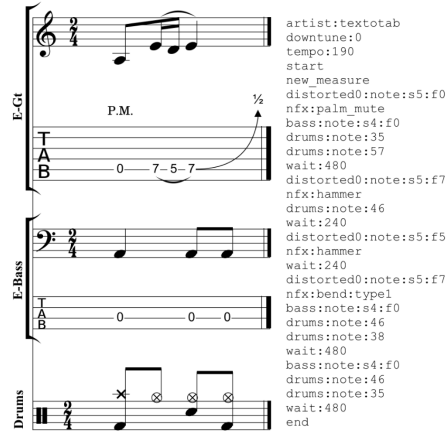


Figure 2: Example measure with a distorted guitar, bass and drums in GuitarPro's graphical user interface (left), and its conversion into token format (right). Figure taken from ?.

It uses a Python script with PyGuitarPro ? to convert GuitarPro files into a tokenized sequence, beginning with metadata tokens which are artist, downtune, tempo, and start. 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. 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 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.

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. The duration of the note is given by the wait token, for instance `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 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. In the example of Figure 2, the bass guitar plays its first note which lasts for 960 ticks. This can be measured as 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. [2]. Their tokenization model introduces a compound word (CP) representation, inspired by previous CP-based approaches [2]. Unlike other 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.

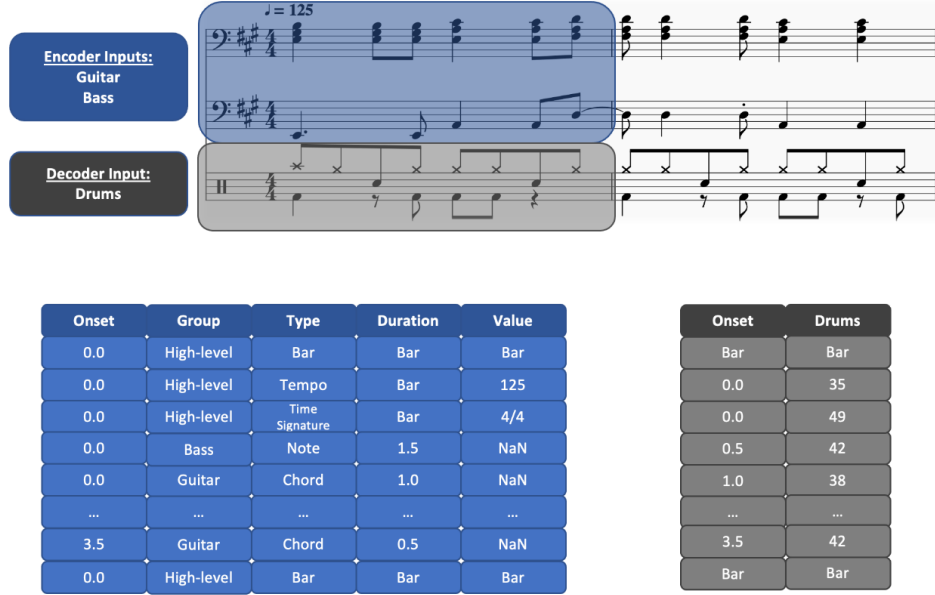


Figure 3: Example of a CP representation used for conditional drums generation. Figure taken from [2].

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 [2]. 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.

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 rhythm 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. ?. GTR CTRL uses a Transformer-XL architecture ?, 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. ?. 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 ?. 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.

3 Data preprocessing

To prepare the data for training, we follow a structured preprocessing pipeline that ensures clean and well-formatted input sequences. The first step involves retrieving and cleaning the DadaGP dataset, extracting tokens specific to selected instruments while preserving relevant note effect tokens and metadata. We then apply an existing method to detect rhythm guitar parts within the dataset, selecting the most rhythmic guitar track for conditioning bass guitar generation. Next, we segment the extracted tokens into overlapping sequences of 16 measures with a stride of 8, applying length constraints to remove excessively long or short sequences. The selected sequences are then formatted into structured dictionaries, where rhythm guitar tokens are used as encoder inputs, and bass guitar tokens serve as decoder targets. Finally, the sequences are converted to token indices, padded to fixed lengths, and split into training, validation, and test sets for model development and evaluation. The following subsections provide a detailed explanation of each of these steps.

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: running 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 pyGuitarPro⁴ is needed for the DadaGP token processing script and is very time-consuming, much more than processing text files. To perform the extraction of instrument-specific tokens, we use the filtering method from the existing token text files. 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

⁴<https://pyguitarpro.readthedocs.io/en/stable/>

wait tokens that were previously separated by notes from instruments that are no longer present. This process took 5min30 on the 26,181 GuitarPro files of DadaGP to generate a copy of the dataset containing token files of bass guitar only.

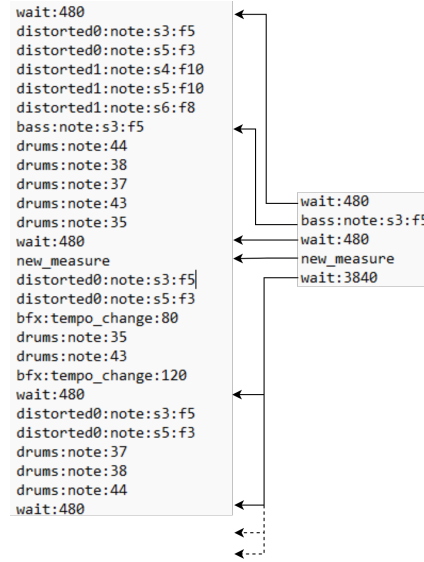


Figure 4: Example of an extraction of bass guitar tokens in The Strokes - Last Nite

An example of token extraction is shown in Figure 4. Note from `distorted0`, `distorted1` and `drums` are removed as well as `bfx` tokens. This figure also shows the concatenation of wait tokens, at the end of the song the bass guitar stops playing before the other instruments. Therefore, we can notice that eight `wait:480` tokens (some are not shown here) are summed up to a single `wait:3840` token.

3.2 Rhythm guitar identification

The algorithm we just presented allows us to extract tokens for any instrument in the DadaGP dataset. In typical rock band instrumentation, rhythm guitars complement bass guitars with their repetitive chord patterns, providing harmonic and rhythmic structure at a higher pitch [?]. This characteristic makes them particularly relevant for conditioning bass generation tasks. Therefore, our first focus is on conditioning bass guitar tablature generation using the rhythm guitar. However, as part of the rhythm section, drums and bass also form a crucial pair in the band. Conditional generation using the drums part was considered, but the absence of pitch and harmonic information in drums made it less relevant for our task.

To identify rhythm guitar parts in the DadaGP dataset, we implemented the method proposed by Régnier et al. [?], 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 bar is to be lead guitar, and the closest to 0, the more likely the bar is to be rhythm guitar. In their paper, they consider that a score below 0.5 correspond to a rhythm guitar parts. As instrumentation can greatly vary between songs and genres, we wanted to select and condition the generation on only one guitar part, the “most” rhythmic among the guitar parts present. 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 rhythm 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 extra ct two more datasets: one containing the rhythm guitar tokens and the other containing both the bass and the rhythm guitar tokens.

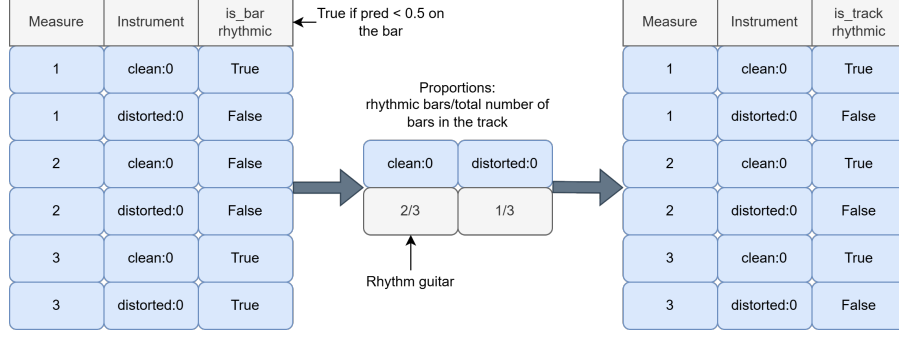


Figure 5: Example of rhythm guitar selection after applying the method of Régnier et al.

Figure 5 shows an example of the selection of the rhythm 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 rhythm guitar and 0 for the other instruments.

3.3 Post processing

To avoid sequences that are too long, 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.

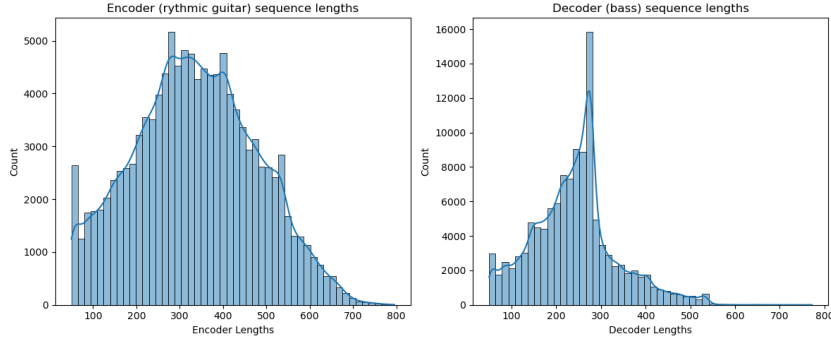


Figure 6: Distribution of sequence lengths in the training dataset

A maximum threshold of 800 tokens, and a minimum threshold of 50 tokens were set to avoid training on sequences that are either too long or that does not contain rhythmic guitar or bass. These are not applied on the complete sequences containing both the tokens of the rhythm guitar and the bass. 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 rhythmic guitar and between 100 and 300 tokens long for the bass. Then, the sequences are fitted to the model’s input requirements using an adapted version of the processing process of Makris et al. ?. The 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 rhythm guitar for the i -th sequence. `dict[All_Events][i]` contains the tokens of bass guitar and rhythm guitar for the i -th sequence.

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 797 tokens for the encoder (Rhythm guitar) and 773 tokens for the decoder (Bass guitar). 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. Those sets are respectively used to train the model, tune the hyperparameters and evaluate the model’s performance.

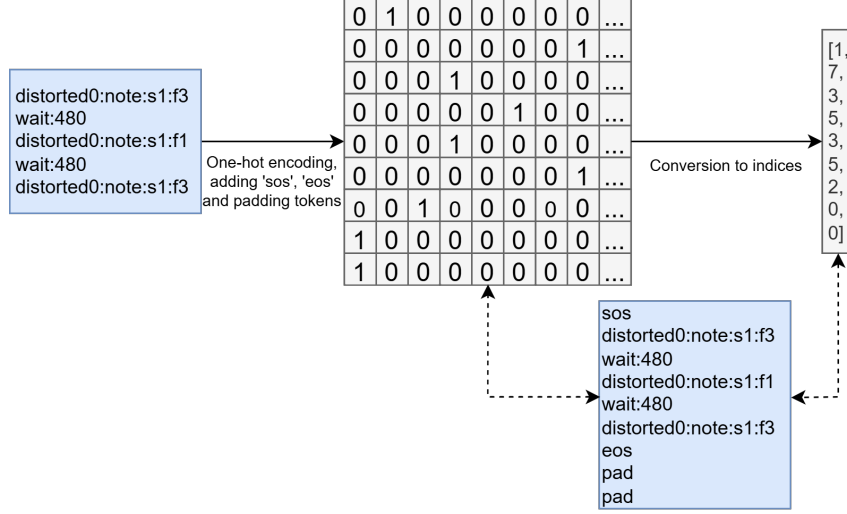


Figure 7: Conversion from a text token sequence to indices

4 Models

4.1 Baseline predictions

We began by establishing a baseline model for bass guitar tablature generation. Initially, we used a pre-trained checkpoint from the DadaGP paper ? without additional training. To generate bass-specific outputs, we experimented with prompts containing single or several bass guitar tokens. Regardless of the number of bass notes given as initial prompt tokens, the model quickly incorporated tokens from other instruments. This is an issue documented in GTR-CTRL ? where they used a metric called the UIP score (Unprompted 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 $-\infty$.



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

An example of a bass tablature generated this way is shown in Figure 8. The generated part repeats the same patterns of two measures over the entire sequence. This part could be used as a simple accompaniment but this model lacks the ability to adapt to the tonality of the song and its rhythm. 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.

4.2 BiLSTM - Transformer model

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

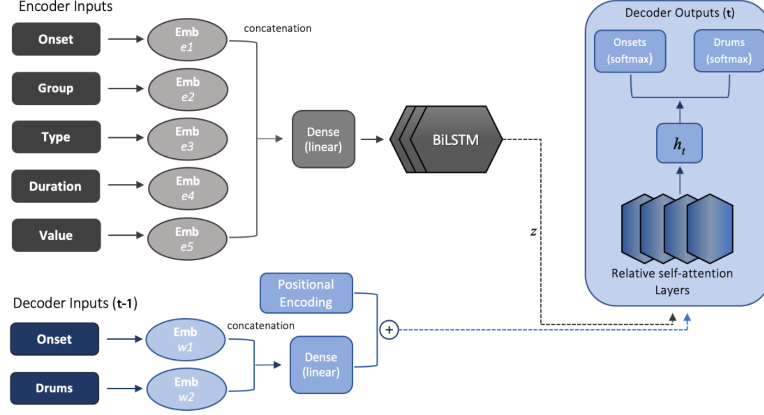


Figure 9: Makris et al. model architecture, taken from their paper ?

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.).

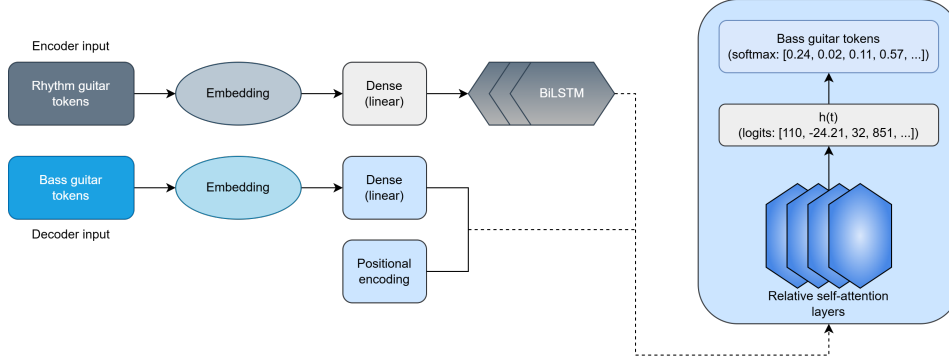


Figure 10: Makris et al. model adapted to our task

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 Encoder processes rhythm guitar tokens using a bidirectional Long Short-Term Memory (BiLSTM) network, allowing the model to capture dependencies both forward and backward in the sequence. The BiLSTM generates a latent variable z , which encodes both rhythmic and structural information from the input sequence. This latent variable serves as a compressed representation of the rhythm guitar part. The Decoder follows an autoregressive approach, predicting each token sequentially. At each time step $t - 1$, the Decoder receives the previously generated token, embedded through a dense layer, along with the latent variable z from the Encoder, which provides contextual information. The Decoder is based on a Transformer architecture with self-attention layers, allowing it to model long-range dependencies within the generated sequence. Unlike standard Transformers that rely solely on absolute positional encodings, we incorporate Relative Global Attention to enhance

the model’s ability to capture hierarchical musical structures. This mechanism enables the model to account for both global position information and relative relationships between musical events, which is crucial for generating coherent bass lines that align with the rhythm guitar.

Following prior work on relative attention, we set the relative attention window size to half of the total sequence length. This balances local context within measures and global context across phrases, improving the overall consistency of the generated bass lines. The final output of the Transformer Decoder consists of relation-aware self-attention activations h_t , which are passed through a Dense layer followed by a softmax activation. Our adapted model consists of approximately 21 million parameters, significantly smaller than architectures such as the Pop Music Transformer, which has around 41 million parameters ?. This makes our approach more efficient while still capturing the necessary musical relationships.

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:

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

where T is the sequence length, K is the vocabulary size, $y_{t,k}$ is the ground-truth token at time step t , and $\hat{y}_{t,k}$ is the predicted probability for token k . The ground-truth token is the token that effectively appears in the original bass guitar sequence at time step t . 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 at most 200 epochs and an early-stopping criterion of 5 epochs without improvement on the validation loss. This means that the training stops if the validation loss does not decrease for 5 consecutive epochs. We used 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.

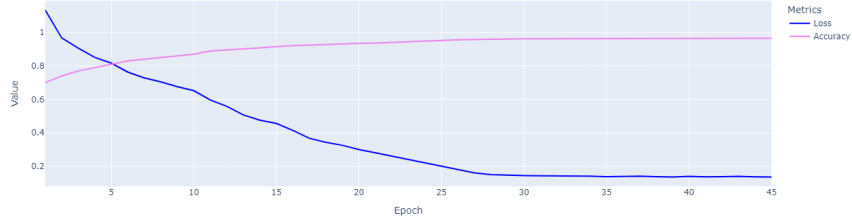


Figure 11: Evolution of validation accuracy and loss during the training

Figure 11 shows the evolution of accuracy and loss during training. The accuracy increases rapidly during the first epochs, then stabilizes around 0.965.

5 Results, discussion and perspectives

5.1 Results

We did not have time to find and implement quantitative evaluation metrics beyond accuracy and loss, so we rely on our expertise to assess the quality of the generated tablatures. In the context of music generation, a creative task, quantitative metrics are rarely enough to evaluate the quality of the generated content. Inference was conducted on the entire test set of 11817 sequences with temperature 0.9. This means that the softmax output of the model is divided by 0.9 in log-space before sampling, which amplifies the differences between the probabilities and thus makes the model

safer. 0.7 and 1.1 were also tested, but 0.9 gave the best results. The model achieved an accuracy of 0.9654 on those sequences, with a corresponding loss of 0.1340. These values are very close to those observed on the validation set.

In this section, we showcase a selection of three results from our experiments⁵. To generate these examples we split the rhythm guitar tracks into 16-measure sequences and generated the bass guitar part for each of them, concatenating the generated sequences to obtain a full song. The examples were carefully chosen to demonstrate the model’s ability to generate tablatures across different musical genres and diverse rhythm guitar patterns. We manually selected rhythm guitar parts and extracted their corresponding tokens. These examples originate from Songsterr⁶ and 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 rhythm guitar parts, so we do not have other models to compare our results to.

The examples chosen are "Ricard Peinard" by Ultra Vomit (metal), "Do I Wanna Know" by Arctic Monkeys (indie rock), and "Saturday Night" by Herbie Hancock (jazz funk). The generated bass tablatures are displayed alongside the rhythm guitar parts used for conditioning and the original bass guitar parts for comparison.

5.2 Examples

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 rhythm guitar part and the original 5-strings bass guitar 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 rhythm guitar part: on the first measure the bass plays several F notes, while the rhythm guitar plays a chord composed of F, C and G notes. Rhythmically, the generated part either copies the rhythm 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 very close to the original bass part on measures 5 to 8.

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 rhythm 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 rhythm 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). We could try to handle this issue by adding a loss term that would penalize the model if the time signature is not respected.

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 generally no rhythm 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 jazz funk is not significantly represented in the dataset, and the model was trained on an agglomerate of genres, it is not able to generate coherent bass lines.

To summarize, the model has great potential in generating bass tablatures of accompaniment that are harmonically and rhythmically coherent with the rhythm 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.

⁵10 experiments can be downloaded from the GitHub repository: <https://github.com/01itHub/guitar-tab-gen>

⁶<https://www.songsterr.com/>

Rhythm guitar part used to condition the generation

Generated bass guitar part

Original bass guitar part

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

5.3 Conclusion

In this work, we explored the conditional generation of bass guitar tablatures using a transformer-based model trained on the DadaGP dataset. We implemented a detailed data preprocessing pipeline, including instrument-specific token extraction, rhythm guitar identification, and structured sequence formatting. Our approach enables the generation of bass lines that are harmonically and rhythmically coherent with a given rhythm guitar part. Our results demonstrate that the model effectively captures the structural role of the bass guitar in various genres, successfully aligning with the rhythmic and harmonic elements of the accompaniment. However, we also identified certain limitations, such as the model's tendency to replicate rhythmic patterns excessively and occasional inconsistencies in time signature adherence. While our study establishes a strong foundation for conditioned bass tablature generation, further refinements in model design, data representation, and evaluation methods could enhance its practical applicability. The next section discusses possible improvements and future research directions.

5.4 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

Rhythm guitar part used to condition the generation

Verse 1

Generated bass guitar part

Original bass guitar part

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

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. We saw in the jazz funk example that the model struggles with genres that are not well-represented in the dataset. Therefore, training the model or fine-tuning it on specific genres could be a good way to both improve the model's performance on that genre and be able to evaluate it more accurately on a more precise task.

We also only used rhythm guitar parts to condition the model, but other instruments could be used as conditioning instruments. We cited drums as a possible conditioning instrument that lacked harmonic information. However, a combination of drums and rhythm guitar could improve the model's performance, although the encoder sequence length would be greatly increased. 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.

"Rhythm" guitar part used to condition the generation

↓

Generated bass guitar part

Original bass guitar part

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