COMP4801 – Final Year Project

Automatic Music Transcription for Electric Bass Guitar (FYP23026)

Final Report

Yaw Jalik

Supervised by

Prof. Wu Chuan

April 2024

Abstract

Automatic music transcription, the algorithmic conversion of musical audio into symbolic representations, is an intriguing and complex task with numerous challenges. It involves techniques such as signal processing and deep learning to extract valuable information implicitly encoded in audio signals, aiding musicians in education and collaboration. Despite existing research and commercial software, there remains a lack of automatic transcription solutions for the electric bass guitar, a fundamental instrument in contemporary music. Challenges with this instrument largely arise from the limited research focus on notation-level transcriptions that represent high-level musical structures and the scarcity of suitable annotated bass datasets. This project aims to address these challenges with two primary objectives: establishing a proof-of-concept notation-level transcription model for monophonic bass guitar audio and generating appropriate instrument datasets for the model. Drawing inspiration from sequence-to-sequence transcription models and automatic speech recognition, a Transformer-based architecture was adopted, enabling the direct transcription of bass audio into LilyPond format, a popular and concise music notation represented in text. Subsequently, an instrument audio generation pipeline was established using Python scripts, the LilyPond command-line tool, a virtual bass guitar plugin, and the Reaper digital audio workstation. The model was trained and evaluated on three versions of datasets, each comprising 10,000 samples, achieving impressive word and character error rates below 10%. Additionally, a simple web interface was constructed for demonstration purposes.

Acknowledgements

First and foremost, I extend my gratitude to Prof. Wu Chuan for their invaluable supervision and guidance throughout the duration of this project. I also wish to thank Dr. Iran R. Roman from CCRMA at Stanford University for their keen interest and insightful advice on this endeavor. My appreciation goes to Mr. Wang Sheng for their valuable suggestions regarding the model architecture. Additionally, I acknowledge the inspiration derived from AutoBassTab, a tablature-based automatic bass transcription project by Ricky Han, which provided an initial influence for this work. Lastly, I express my heartfelt thanks to Ms. Ooi Xin Ci for their unwavering support throughout this project.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AMT	Automatic Music Transcription
ASR	Automatic Speech Recognition
CNN	Convolutional Neural Network
DAW	Digital Audio Workstation
DSP	Digital Signal Processing
LSTM	Long Short-Term Memory
MIDI	Musical Instrument Digital Interface
MIR	Music Information Retrieval
NLP	Natural Language Processing

1. Introduction

Music Information Retrieval (MIR) is a multidisciplinary field aiming to develop techniques for extracting valuable information from music. One of its tasks is Automatic Music Transcription (AMT), which involves algorithmically converting music audio into symbolic representations like sheet music [1]. This process resembles Automatic Speech Recognition (ASR) but focuses on converting musical audio into a symbolic form rather than converting human speech into natural language.

This introduction begins by offering background information on AMT, the target instrument, and project goals. Section 1.1 delves into important details regarding different scopes of transcription. Sections 1.2 and 1.3 provide motivation for the practical applications of AMT and outline key challenges respectively. Section 1.4 highlights existing research and commercial software solutions, along with their empirical performance. Finally, section 1.5 introduces the instrument of choice, the electric bass guitar, while section 1.6 outlines the project goals centered around the transcription of this instrument.

1.1 Types of Symbolic Representations

The following section will refer to [1] to briefly examine four types of music representations, each exhibiting an increasing level of abstraction and illustrated in Figure 1. The initial type is frame-level transcription, which captures musical events like frequencies within short time frames, typically on the scale of milliseconds. At this level, there are no representations of musical structure; only individual events present within a given time frame are depicted.

Following is note-level transcription, where the concept of a musical note is established. A note is defined by its pitch, onset (starting point), and optional offset (ending point). Detected notes are treated independently of each other.

Advancing further, stream-level transcription aims to categorize notes into groups based on shared characteristics. For instance, notes with similar timbre (sound quality) might be clustered together into their own instrument categories.

Lastly, notation-level transcription attempts to generate human-readable notation. For instance, Western classical notation serves as a widely accepted standard notation system. Readers interested in gaining insight into this format can refer to Appendix A for a brief description, which may provide helpful additional context. Other notation systems include tablature for stringed instruments, lead sheets in Jazz, Chinese jianpu, and many more. Compared to the preceding three scopes, notation-level transcription garners less attention in research, likely due to the substantial challenges involved in handling high-level musical structures (see

section 1.3). Nevertheless, given the practical utility of human-readable notation, this project endeavors to contribute to the advancement of transcription within this scope.

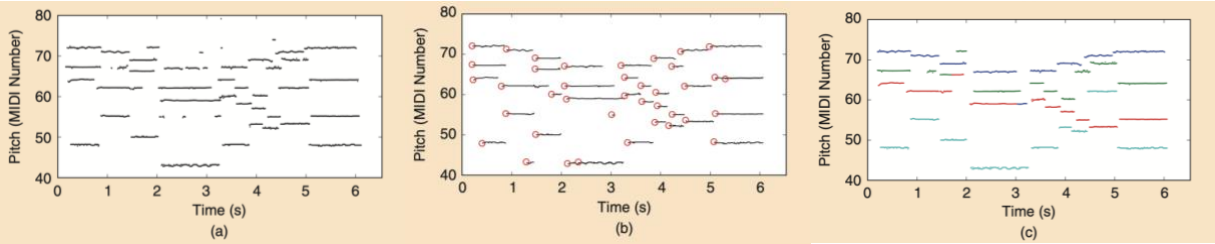


Figure 1. (a) Frame level transcription. (b) Note-level transcription. (c) Stream-level transcription.

1.2 Uses of AMT

Essentially, notation-level transcriptions serve as descriptions of musical pieces and provide instructions on how they can or should be performed. These transcriptions find direct application in music education, where both students and educators stand to benefit from transcription tools for learning and teaching purposes.

Another potentially significant application lies in collaboration between musicians. Notation aids professional musicians in managing large setlists by offering visual cues or allowing them to read music on the fly. Orchestral musicians, for instance, commonly rely on charts while performing on stage. With automatic transcription, artists can also potentially speed up the process of transcribing their unannotated recordings for fellow musicians or create and publish their own sheet music for sale.

As AMT is intricately linked with other MIR tasks, its applications can indirectly benefit various domains. For instance, in copyright enforcement, AMT may contribute to cover song detection, assisting in identifying copyright infringements on media platforms [1].

1.3 Challenges of AMT

Music transcription presents a complex and demanding challenge for both musicians and algorithms alike. For instance, real musical recordings featuring human performances often do not perfectly align or synchronize with transcriptions. This discrepancy may arise from various factors, including creative decisions by musicians to alter the speed (tempo rubato) or simply from the inherent imperfections of human performance.

Furthermore, different instruments, or even variants of the same instrument, yield different sound qualities (timbre) for identical transcriptions, resulting in varying frequency content. Figure 2 shows two spectrograms comparing a melody generated on a virtual piano and a bass guitar. Additional complexities emerge when extracting information from audio containing noise and multiple sound sources, akin to the Cocktail Party problem.

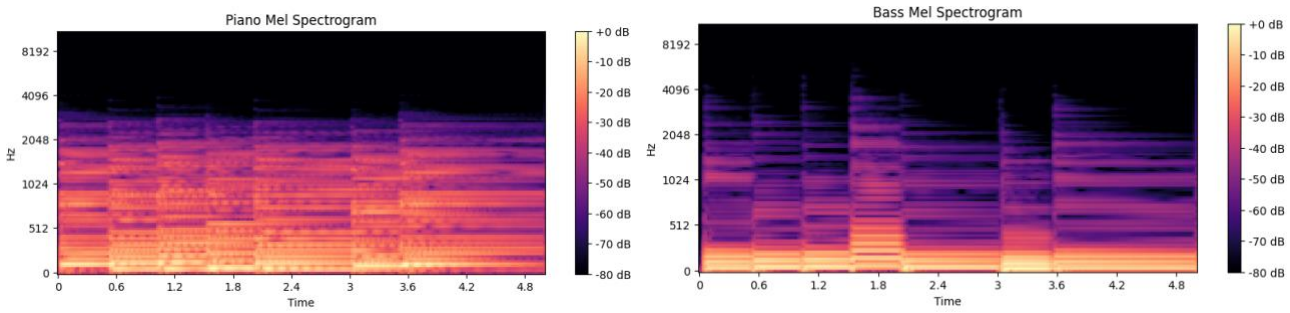


Figure 2. Spectrograms of a melody played on a virtual piano and bass guitar. Different instrument timbre results in different frequency content.

Transcriptions themselves can also be ambiguous. Consider enharmonic equivalence, illustrated in Figure 3, which indicates that two notes sound exactly the same but are notated differently depending on the musical context. For instance, a C sharp (♯) sounds identical to a D flat (♭). This distinction is significant in section 3.2. One such contextual factor is the musical key or a family of pitches. For instance, the key of 'D major' includes a C sharp, whereas the key of 'A flat major' includes a D flat. While this distinction makes sense for musicians familiar with music theory and aware of notational conventions, it can lead to ambiguity without accounting for context.

Consequently, available datasets for music transcription are relatively scarce. According to [2], AMT datasets typically consist of hundreds of hours, whereas ASR datasets extend into the thousands of hours. After all, music transcription is a highly time-consuming and non-trivial task, even for professional musicians, considering the challenges outlined above.

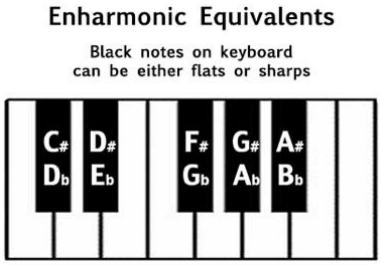


Figure 3. Enharmonic equivalence visualized on the black keys of a piano keyboard. Image obtained from https://www.musictheoryacademy.com/understanding-music/enharmonic-equivalents/

1.4 Existing Solutions

In general, two major approaches have been adopted in AMT: digital signal processing (DSP) and deep learning. DSP techniques are generally faster and more diverse, whereas deep learning achieves better results in narrower domains [1]. Recent research models often combine both approaches, often placing greater emphasis on deep learning models.

A few notable models proposed in research are worth highlighting and have served as major references and inspirations for this project. The first is a paper by Carvalho and Smaragdis that developed an end-to-end model for converting audio into music notation in LilyPond – a digital notation format that we will further discuss in later sections [3]. The model primarily consists of a Convolutional Neural Network (CNN) front-end to transform a one-dimensional audio sequence into a multidimensional representation. This is followed by an encoder-decoder layer using Long Short-Term Memory (LSTM). The paper claimed excellent performance when tested on synthetically generated piano and human chorus audio. Unfortunately, it lacked specific implementation details and source code for replication.

Another notable model is the Multi-task Multitrack Music Transcription (MT3) developed by Magenta; a research project affiliated with Google AI [2]. MT3 was trained on popular available music datasets and utilizes Transformers to transcribe multiple instruments simultaneously into MIDI-like tokens. MIDI, short for Musical Instrument Digital Interface, is a widely used digital instrument communication protocol. While not a formal notation scheme, MIDI can informally represent notes on a piano-roll structure (refer to Appendix B for more details). However, using this model has some drawbacks, such as noticeable confusion in instrument prediction. As claimed by Magenta, predicted instruments within the output may change abruptly [2]. Moreover, MIDI representations do not directly encode the high-level musical structures that we aim to extract. As per their GitHub repository, they also do not appear to support training easily.

There are a few notable options in commercial software that are worth highlighting, namely AnthemScore and ScoreCloud. Like many commercial software products, they are paywalled and closed source, making it difficult to analyze and directly compare their performance. Nevertheless, some users find the results to be generally unsatisfactory for paid software (see Figure 4) [1].

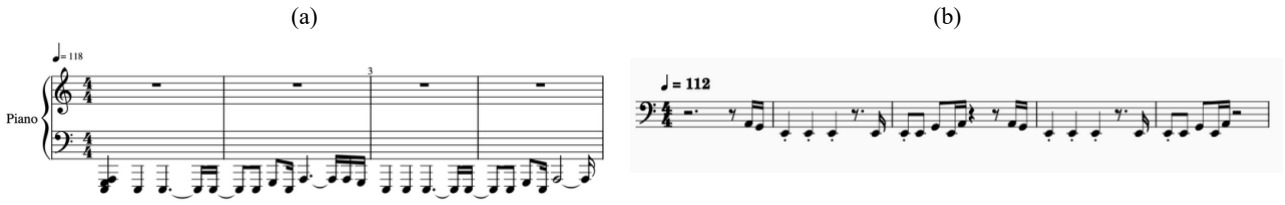


Figure 4. Visual comparison of a short bassline (‘Another One Bites The Dust’ by Queen) predicted by (a) AnthemScore (with free trial license) and (b) original score.

1.5 The Bass Guitar

This project focuses AMT for a specific instrument – the electric bass guitar (see Figure 5). While resembling an electric guitar, the electric bass guitar differs significantly, primarily in producing lower frequencies with the help of thicker strings and a longer neck. Indeed, the term 'bass' refers to low-pitched sounds.

Primarily, this instrument serves a supporting but crucial role in modern music, encompassing rhythm and harmony (see Appendix C for brief descriptions of relevant musical concepts). Consequently, it often receives less attention and glamour compared to other instruments.

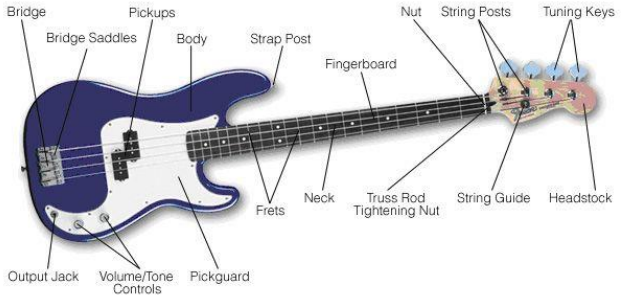


Figure 5. A typical 4-string bass guitar with labelled parts.

The challenges in bass transcription primarily stem from its niche nature. There's a scarcity of high-performing and readily accessible models specifically designed for notating this instrument. Datasets of notated bass audio are rare and often inadequate for the task. While some datasets, such as the MUSDB18 dataset, include bass tracks within multi-instrument arrangements, they are primarily utilized for training audio source separation models. The closest available dataset is the Synthesized Lakh Dataset (Slakh2100), which offers a substantial collection of multi-track audio paired with aligned MIDI. However, as the desired outcome is notation rather than MIDI, this dataset may not directly cater to the goal of notation-level transcription (issues with MIDI further elaborated in section 2.4).

Furthermore, we speculate that the distinctive sonic characteristics of bass, characterized by lower frequencies and overtones, may present challenges for existing pitch estimators and transcription models. These tools are typically trained on a variety of instruments, such as piano and violin, which occupy broader or higher frequency ranges. Figure 6 shows a visualization of the MIDI output produced by Basic Pitch, Spotify’s impressive and lightweight transcription model, which predicted multiple notes despite the audio being monophonic [4].

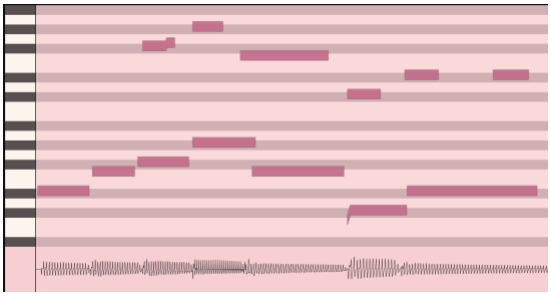


Figure 6. A graphical visualization of the MIDI output generated by Spotify's Basic Pitch model on the same melody in Figure 2, without parameter adjustments *[4]*. The model seems to have predicted polyphonic notes as characterized by upper and lower streams.

1.6 Project Objectives and Scopes

Given the context provided earlier, this project aims to accomplish two main objectives. Firstly, we aim to create a proof-of-concept notation-level transcription model tailored for bass guitar, drawing inspiration from previous research attempts discussed. The inputs to this model consist of monophonic (single note at a time) and isolated (no accompanying instruments) bass guitar audio, while the outputs should comprise of textual representations corresponding to Western classical notation with 12 equal temperament tuning (Appendix D).

Secondly, in response to the scarcity of suitable data, we will devise a pipeline to automatically generate our own annotated synthetic bass guitar audio dataset, drawing from methodologies used in prior research. This dataset will serve as training and evaluation data for our model, ensuring that it has access to sufficient and relevant information during the learning process.

2. Preliminary Attempts

Before delving into the primary methodology, we wish to offer an overview of the initial approach used, inspired by a project called AutoBassTab which pursued a similar goal but using tablature instead of Western notation. This initial model comprised of four components that will be explored in more detail in each section: pitch estimator (section 2.1), onset detector (section 2.2), playing-style classifier (section 2.3), and parser (section 2.4). We will elaborate how the shortcomings encountered with this approach influenced the adoption of our current methodology instead. It is noteworthy that the concept of synthetic data generation was not yet considered at this stage. Thus, the results are empirical, relying on qualitative and auditory confirmation utilizing mostly real bass guitar samples.

2.1 Pitch Estimator

First, a pitch estimator (a frame-level transcriber) was employed to predict the fundamental frequencies in the music audio. The CNN-based pitch estimator CREPE, adopted by AutoBassTab, was selected [5]. This model apparently achieved superior results over other state-of-the-art pitch estimators when evaluated on synthetic

audio, as illustrated in Table 1 [5]. Despite this claim, empirical results demonstrate that its accuracy may not align with its assertions when applied to bass guitar audio. Figure 8 (b) illustrates an obvious octave error when tested on a real bass guitar sample.

Dataset	Threshold	CREPE	pYIN	SWIPE
RWC-synth	50 cents	**0.999±0.002**	0.990±0.006	0.963±0.023
	25 cents	**0.999±0.003**	0.972±0.012	0.949±0.026
	10 cents	**0.995±0.004**	0.908±0.032	0.833±0.055
MDB-stem-synth	50 cents	**0.967±0.091**	0.919±0.129	0.925±0.116
	25 cents	**0.953±0.103**	0.890±0.134	0.897±0.127
	10 cents	**0.909±0.126**	0.826±0.150	0.816±0.165

Table 1. Comparisons of average raw pitch accuracies and standard deviations between CREPE and other state-of-the-art pitch estimators tested on two datasets: RWC-synth and MDB-stem-synth *[5]*.

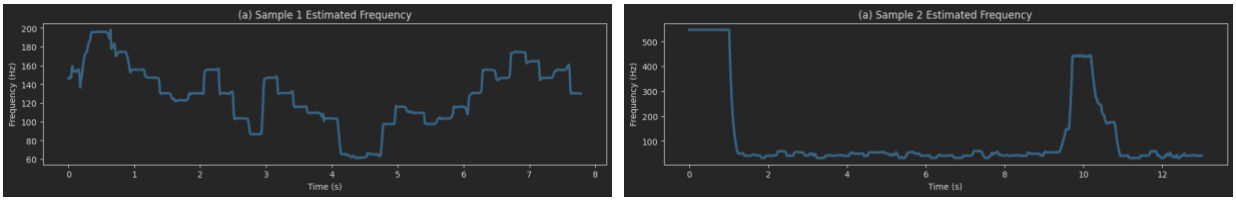


Figure 7. (a) Predicted frequencies of Sample 1, with relatively distinct notes denoted by horizontal line segments. (b) Predicted frequencies of Sample 2, with octave errors at the start and 10-second mark.

2.2 Onset Detection

Subsequently, the model was integrated with an onset detector to estimate the instances when musical notes occur, refining the transcription to note-level. Essentially, onset detection operates by first computing local changes within an audio waveform, referred to as novelty functions or curves; amplitude changes yield an energy novelty curve, while changes in frequency content produce a spectral novelty curve (illustrated in Figure 9) [6]. While a sudden increase in amplitude may frequently correspond to the start of a note, this may not always be the case. For example, a bassist may use a hammer-on technique that immediately alters the pitch without changing the amplitude. Thus, spectral-based novelty functions were used. Subsequently, the peaks of these novelty curves are strategically selected as onset candidates with certain rules and algorithms.

The onset detection function from Librosa, a popular Python digital signal processing library, was utilized. Librosa's onset_detect function first computes a spectral onset strength envelope, essentially a spectral novelty function, and then selects peaks using the heuristics shown in Figure 10 [6].

As an additional step, we programmed additional rules such that potential false positives were naively pruned; if an onset is within 0.1 seconds of its preceding onset, it will be removed as to eliminate closely spaced onsets that may correspond to false positives, visualized as very tightly spaced red vertical lines in Figures 11a and 11c.

However, we observed that parameters that affect the sensitivity of onset detection are not immediately intuitive to adjust. These include hop length, sampling rate, and parameters from peak picking (e.g. pre_max, post_max, delta, etc.). Some sets of parameters seem to work well only for certain samples. To achieve optimal results, this approach may be better suited for those with more experience in signal processing.

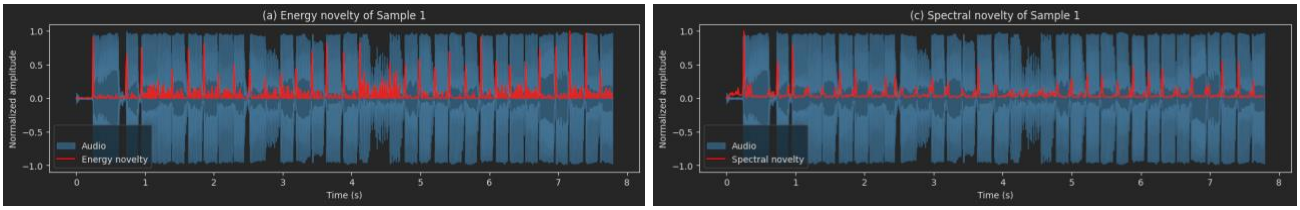


Figure 8. (a) Energy-based novelty function of Sample 1. (b) Spectral-based novelty function of Sample 1

librosa.util.peak_pick(*x, *, pre_max, post_max, pre_avg, post_avg, delta, wait*) [source]

Uses a flexible heuristic to pick peaks in a signal.

A sample n is selected as an peak if the corresponding `x[n]` fulfills the following three conditions:

1. `x[n] == max(x[n - pre_max:n + post_max])`
2. `x[n] >= mean(x[n - pre_avg:n + post_avg]) + delta`
3. `n - previous_n > wait`

where `previous_n` is the last sample picked as a peak (greedily).

Figure 9. Rules and parameters used by Librosa for selecting peaks of signals *[6]*.

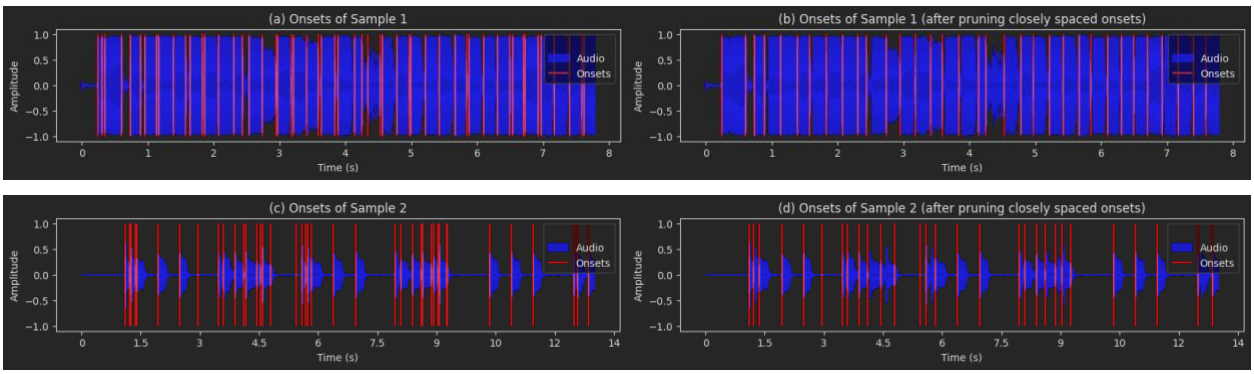


Figure 10. (a) Sample 1 onsets computed by librosa. Each red vertical line corresponds to a predicted onset. (b) Sample 1 onsets after pruning. (c) Sample 2 onsets. (d) Sample 2 onsets after pruning.

2.3 Playing Style Classifier

As an additional challenge, a playing-style classifier was incorporated to predict various playing styles evident in the recording, aiming to enhance transcription accuracy. Playing styles refer to different techniques used by musicians on the same instrument to produce different timbre with the same pitch. For instance, bass techniques like plucking sound thicker and more mellow while slapping sounds percussive and bright.

The deep CNN architectures proposed by [7] were suitable candidates (see Table 2). These accept raw time-domain waveforms as input as opposed to pre-extracted features. The intuition is to enable the model to learn

relevant sonic features autonomously rather than relying on human efforts to identify 'correct' features [7]. Moreover, these architectures omit dense layers, commonly employed after convolutional layers, to maximize learning within the convolutional layers themselves, which are hypothesized to be the primary locus of learning [7]. This promises great performance and quicker development by significantly reducing the time required for feature design.

A slightly modified M5 architecture was implemented in PyTorch, with a higher stride length of 16 in the first convolutional layer for faster processing. The model was trained with the IDMT-SMT-Bass dataset, publicly released by the Fraunhofer Institute for Digital Media Technology [8]. This dataset contains a total of 5323 single bass guitar notes, of which 2347 from the ‘plucking styles’ category was used for training. The plucking styles in the dataset are classified into five distinct classes: finger style (FS), muted (MU), picked (PK), slap-thumb (ST), and slap-pop (SP). Prior to training, these files were downsampled from their original 44.1 kHz sampling rate to 8 kHz before training to reduce input size.

M3 (0.2M)	M5 (0.5M)	M11 (1.8M)	M18 (3.7M)	M34-res (4M)
Input: 32000x1 time-domain waveform				
[80/4, 256]	[80/4, 128]	[80/4, 64]	[80/4, 64]	[80/4, 48]
Maxpool: 4x1 (output: 2000 × n)				
[3, 256]	[3, 128]	[3, 64] × 2	[3, 64] × 4	$\begin{bmatrix} 3, 48 \\ 3, 48 \end{bmatrix} \times 3$
Maxpool: 4x1 (output: 500×n)				
	[3, 256]	[3, 128] × 2	[3, 128] × 4	$\begin{bmatrix} 3, 96 \\ 3, 96 \end{bmatrix} \times 4$
	Maxpool: 4x1 (output: 125 × n)			
	[3, 512]	[3, 256] × 3	[3, 256] × 4	$\begin{bmatrix} 3, 192 \\ 3, 192 \end{bmatrix} \times 6$
	Maxpool: 4x1 (output: 32 × n)			
		[3, 512] × 2	[3, 512] × 4	$\begin{bmatrix} 3, 384 \\ 3, 384 \end{bmatrix} \times 3$
Global average pooling (output: 1 × n)				
Softmax				

Table 2. CNN architectures adopted in *[7]*. M5 (0.5M) represents 5 weight layers and 0.5 million parameters. The notation [80/4, 128] specifies a convolutional layer with receptive field 80, 128 filters, and stride 4. For layers with stride 1 (e.g., [3, 128]), the stride is not explicitly mentioned.

2.4 MusicXML Parser

Lastly, a custom parser combines these data into a notation-level transcription using MusicXML format, a music score representation developed by the World Wide Web Consortium (W3C) employing Extensible Markup Language (XML) tags (see Figure 7). Unfortunately, this parser was not developed. There are no existing parsers specifically designed to directly convert data these data into XML, and creating one is quite a costly endeavor. As a workaround, the pitch estimation and onset data were initially converted to MIDI as an intermediate representation using the following method: each predicted pitch P_t occurring at onset time t was extracted. The durations of each note were determined as the interval between the current onset t_1 and the next onset t_2. Thus, for each P_t, a MIDI note of duration $t_2 - t_1$ was constructed. It is worth noting that this process can be thought as a note-level transcription attempt.

Regrettably, there are still no readily available methods to directly convert MIDI into notation. An approach proposed by [9] seeks to estimate an optimal musical time grid from human-performed MIDI data, which could hold promise for projects that are focused on MIDI to notation conversion. Additionally, MuseScore, a commercial open-source music notation software, possesses an algorithm for converting MIDI to scores (proprietary format), although it remains unpublished and lacks comprehensive documentation. Ambitious developers may consider exploring and reverse engineering their algorithm, which is available in their GitHub repository under the *importmidi.cpp* file.

Considering the inherent challenges in each component of the overall model, we concluded that the model was overly cumbersome, and it was not worthwhile to proceed with its development and formal evaluation.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE score-partwise PUBLIC
    "-//Recordare//DTD MusicXML 4.0 Partwise//EN"
    "http://www.musicxml.org/dtds/partwise.dtd">
<score-partwise version="4.0">
  <part-list>
    <score-part id="P1">
      <part-name>Music</part-name>
    </score-part>
  </part-list>
  <part id="P1">
    <measure number="1">
      <attributes>
        <divisions>1</divisions>
        <key>
          <fifths>0</fifths>
        </key>
        <time>
          <beats>4</beats>
          <beat-type>4</beat-type>
        </time>
        <clef>
          <sign>G</sign>
          <line>2</line>
        </clef>
      </attributes>
      <note>
        <pitch>
          <step>C</step>
          <octave>4</octave>
        </pitch>
        <duration>4</duration>
        <type>whole</type>
      </note>
    </measure>
  </part>
</score-partwise>
```

Figure 11. A simple MusicXML document sample from https://www.w3.org/2021/06/musicxml40/tutorial/hello-world/.

3. Main Methodology

The following describes current methodologies employed for the project. Section 3.1 delves into the present model architecture, while Section 3.2 elaborates on the data generation pipeline. Section 3.3 provides additional details on model training and testing. Finally, section 3.4 describes the web application developed to host and demonstrate the model.

3.1 Current Model Architecture

Drawing inspiration from research and from speech recognition models, we settled on an architecture resembling the one proposed by Carvalho and Smaragdis [3] [10]. We employed a CNN audio encoder but

substituted the LSTM encoder-decoder with a Transformer [11]. The model was implemented in PyTorch. Figure 12 illustrates the diagram of the architecture.

Firstly, a Vocab helper class, frequently utilized in natural language processing (NLP) applications such as machine translation, was devised to manage important metadata associated with the labels. This includes Python dictionaries that bidirectionally map a unique integer to each token, counts the number of occurrences of each token, along with the total vocabulary size. Additionally, it contains auxiliary methods to facilitate the conversion of label strings into tensors of indices and vice versa. The Vocab object is initialized with four special tokens: start (‘{’), end, padding (‘<pad>’), and unknown (‘<unk>’).

We chose a CNN audio encoder for raw audio waveforms instead of spectrograms, a frequently employed feature representation. At this stage, we observed no noticeable disparities in performance, thus kept the CNN approach. The convolutional layers primarily convert a one-dimensional audio waveform into a multi-dimensional representation. We employed 256 filters for this model. Following each convolutional layer, we applied ReLU activation and utilized a max-pooling layer for dimensionality reduction. Subsequently, we incorporated a layer normalization step. Finally, sinusoidal positional encoding was applied to the outputs to embed positional information before handing them over to the Transformer.

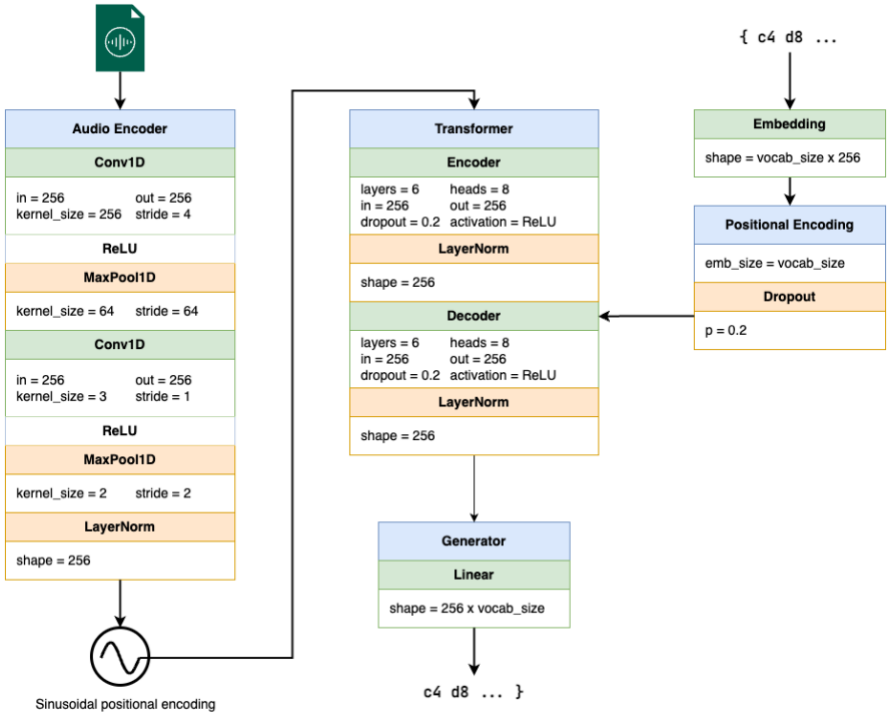


Figure 12. Illustration of the model architecture. The diagram of the Transformer is simplified, with specific details within the encoder and decoder layers omitted. To delve deeper into this, visit the project’s GitHub repository and PyTorch’s Transformer documentation page.

The key advantages of employing a Transformer involve its capability for parallel processing, enabling the utilization of Graphical Processing Units (GPU) during training to speed up the process. In contrast, recurrent units or LSTMs operate sequentially. We adopted PyTorch's multi-head attention Transformer implementation based on the "Attention Is All You Need" paper [11]. In short, this implementation comprises an encoder block designed to capture input context and dependencies, producing what is termed as a memory tensor. Subsequently, the decoder operates on this memory tensor along with an embedded sequence of tokens that correspond to the LilyPond format. We primarily utilized default parameters: 6 layers for both the encoder and decoder, 8 heads for multi-attention, and ReLU activation. The number of features for encoder and decoder inputs are set to 256.

To prepare target labels for input to the decoder, additional steps were necessary. Target masking was generated and applied to prevent the decoder from attending to future positions. PyTorch's *generate_square_subsequent_mask* method produces masks with *-inf* values for these future positions. The target labels undergo transformation into word embeddings via a PyTorch Embedding layer. Sinusoidal positional encoding, supplemented with a dropout layer, was also utilized, following the solution provided in this PyTorch tutorial. At the output, a linear layer was used to convert the decoder output, containing 256 features, into logits with dimensions corresponding to the vocabulary size.

By representing each output token as either a special token (starting and ending token) or a note, we ensure that syntax errors are avoided when compiling the generated scores with the LilyPond command line interface (CLI), as long as the start and end tokens are included. This method differs from outputting individual characters, as seen in the Carvalho and Smaragdis method, which could potentially result in compilation errors [3]. Further details on score syntax are provided in the subsequent section.

3.2 Dataset generation

We devised a three-stage process for producing pairs of synthetic bass guitar audio and corresponding labels in LilyPond format. LilyPond refers to both a notation syntax and an open-source music engraving toolset that is part of the GNU project. It offers a concise method for representing sheet music resembling the LaTeX syntax, in contrast to the more verbose XML representation. Moreover, the toolset incorporates a CLI capable of compiling *.ly* scores into formats such as PDF for practical application.

Typically, notes in LilyPond adhere to the representation outlined in Figure 13. Each note comprises a pitch or a rest, along with optional accidentals (sharps and flats), octave modifiers, and a subdivision. For further clarification on the definitions of common elements in Western notation, refer to Appendix A. It is important to note that these guidelines represent only a subset of LilyPond's interpretative capabilities. For more comprehensive information, visit lilypond.org to explore more.

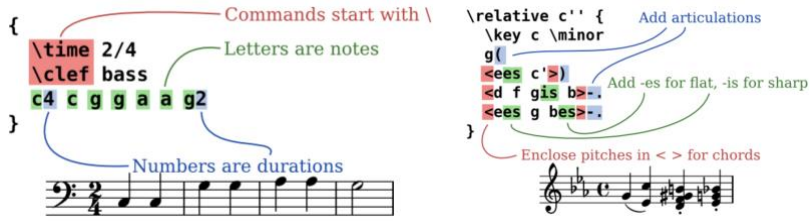


Figure 13. A brief overview of LilyPond's syntax, obtained from https://lilypond.org/text-input.html

First, a Python script generates text labels resembling the shape shown in Figure 14. These labels are structured as text sequences, comprising an opening curly brace, individual notes, and a closing curly brace. These braces signify the start and end tokens of each sequence, commonly denoted as 'SOS' and 'EOS' in NLP tasks. Simultaneously, each label is inserted into a boilerplate score template and stored as a *.ly* file. The guidelines governing label generation are as follows:

- Notes are depicted with absolute pitches in LilyPond's 'absolute mode' and utilize explicit subdivisions instead of 'relative mode'. This approach aims to establish a clearer ground truth and minimize ambiguity. The labels begin with '{' and end with '}'.
- Pitches comprise one of the seven notes from the set *{c, d, e, f, g, a, b}* or a rest denoted by 'r' (indicating a period of silence). Rests are excluded from the initial dataset batch (Version A).
- Only sharps ('is') are employed as the optional accidental to avoid ambiguity stemming from enharmonic equivalence, as explained in the introduction. Thus, the pitches 'b' and 'e' are restricted from having sharps; 'b sharp' is identical to 'c', and 'e sharp' is identical to 'f'. Furthermore, stacking (i.e., double sharp) is not utilized.
- Only one lower octave (',') is employed; the upper octaves are currently not utilized to keep vocabulary size smaller and maintain focus on lower pitches.
- Leading, trailing, and consecutive rests are avoided. Trailing rests are disregarded during the MIDI generation detailed in the subsequent section, while consecutive rests introduce ambiguity (two rests of duration 8 are indistinguishable from one rest of duration 4). Leading rests were also considered for exclusion but should have been retained.
- Notes are randomly generated from the vocabulary list, which comprises all possible combinations of notes. This selection can be musically justified as utilizing the chromatic scale.
- The length of each score varies between 10 to 20 notes, randomly chosen.
- The tempo is fixed at 120 beats per minute (BPM), eliminating the need for tempo prediction.
- The time signature in the output score is set to 4/4; however, no efforts were made to ensure that note subdivisions within a measure adhere to this constraint, as LilyPond does not enforce it. Additionally, there were no attempts to predict the time signature.

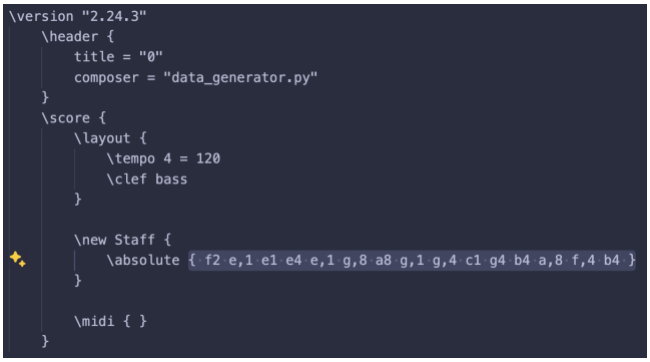


Figure 14. Example score generated by the *data_generator.py* Python script. The highlighted segment corresponds to the label.

Next, the LilyPond CLI was employed to compile the *.ly* scores into MIDI format. Recall that MIDI is a protocol utilized for communication between digital instruments, which includes support for virtual instruments as well.

Finally, these MIDI files were rendered programmatically using a free virtual bass guitar plugin called Ample Bass P Lite II. This workflow was executed in Reaper, a digital audio workstation (DAW) or audio recording software (see Figure 15). Although the Reaper API itself doesn't support automated rendering, a solution called Ultraschall, a podcasting plugin for Reaper, was capable of this. Upon loading the virtual instrument, a custom Lua script was executed to iteratively load MIDI files and invoke the necessary APIs to render audio files.

The audio output specifications are as follows: a sampling rate of 16 kHz, a bit-depth of 16-bit PCM (pulse-code modulation), and a mono channel configuration.

In total, we generated three batches of data, each containing 10,000 samples, with an increasing vocabulary size as illustrated in Table 3. We opted for a sample size of 10,000 as a compromise between dataset size and the time required for audio rendering (see section 4.3 for details on issues). The rationale behind generating different batches was to commence model training with a relatively simpler vocabulary to ensure proficient model performance, gradually progressing to more complex vocabularies. Additionally, while this subset is considerably simplified compared to real music, it serves as a solid foundation for utilization in the proposed model. Consider the number of possible scores for the smallest vocabulary size of 48 and score length 10 (allowing repetitions of all notes): 48^{10} = 64,925,062,108,545,024 possible scores – not a small number.

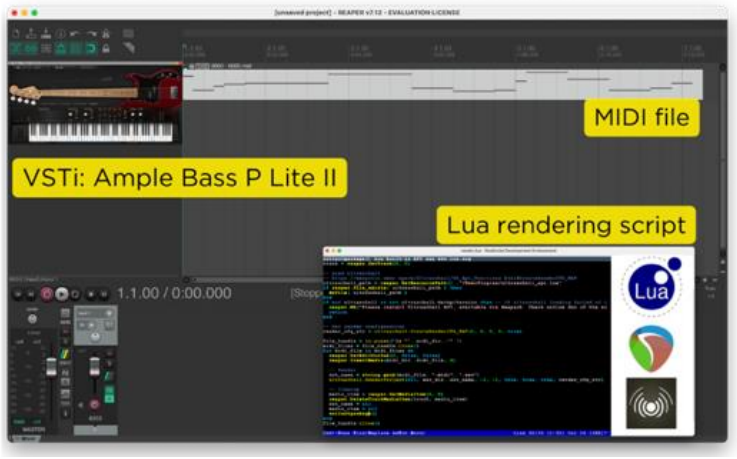


Figure 15. The Reaper DAW with virtual instrument loaded and a text editor for scripting. The Lua script iteratively loads MIDI files onto the project timeline and renders the audio.

Dataset Version	Vocabulary Size
A	48
B	84
C	105

Table 3. Three batches of data with corresponding vocabulary size (number of unique tokens) excluding special tokens (starting, ending, padding, and unknown).

3.3 Training and Testing

For training, teacher forcing was used, where the Transformer decoder receives the correct output labels instead of its own previously generated output. This approach aimed to mitigate error propagation (where small prediction errors accumulate over time) potentially leading to more stable training. The outputs are compared with the true labels without the starting token. Therefore, it is important to offset the input target label such that the ending token is omitted.

The following details for training are as follows. A 90:10 train-test split was employed to allocate more training data to the model. A batch size of 1 was utilized due to the complexities associated with padding and masking audio data using PyTorch; there does not seem to be straightforward methods to handle masking of encoded audio data with paddings. Cross-entropy loss, Adam optimizer, and a learning rate of 1e-5 were utilized. The model was trained for 20 epochs, as the loss appeared to converge around that point.

For testing and inference, the Transformer decoder is configured to predict output tokens autoregressively, utilizing its own previously generated output as target inputs. The target label is initially set as a tensor with a maximum length of 22, comprising the maximum number of notes (20) plus a starting and ending token. It is important to note that the ending token is not utilized for making predictions. The first element represents the start token, while the remainder consists of placeholder unknown tokens '<unk>'. Through masking, the

decoder is effectively prevented from observing the unknown token at the current timestep. During each iteration through the target sequence, the subsequently predicted token replaces the corresponding unknown token. Iterations halt prematurely upon predicting an end token, and the unknown tokens are stripped off.

Torchaudio’s edit distance (also known as Levenshtein distance) is used to calculate how closely the predicted notes align with the labels. This metric captures the total number of substitutions, additions, and deletions required to transform one sequence into another. When comparing strings, it provides a character-level distance, whereas when comparing lists of tokens, it computes a word-level distance. These distances are then used to calculate the character error rate (CER) and word error rate (WER) by normalizing over the number of characters or words present in a particular sample.

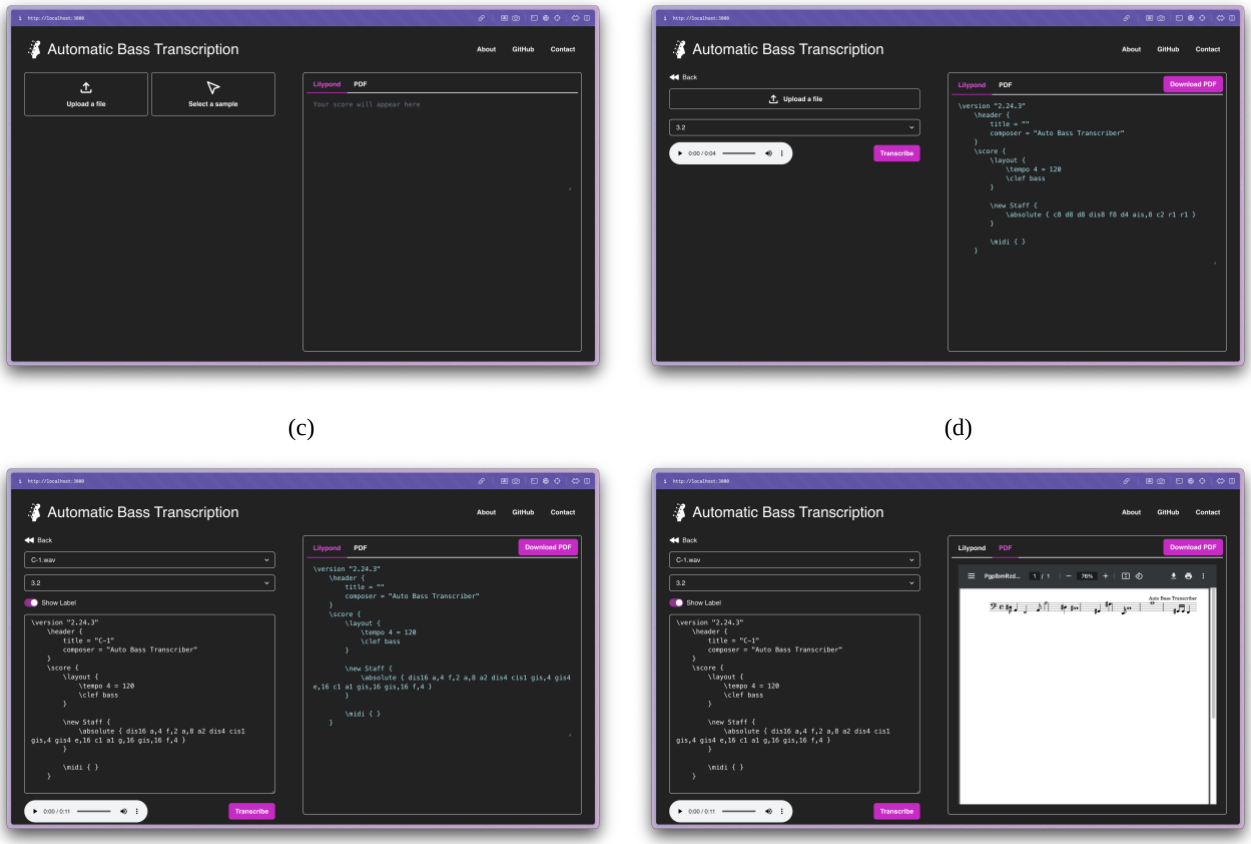
The process was executed on Google Colab, utilizing T4 GPU instances. Training runs typically required an average of 2-3 hours to complete. Following training and testing, the model's state dictionary, along with the Vocab object, were serialized and saved for future use. To utilize the model again, the model object must be recreated with the same parameters, and then the state dictionary and vocabulary are loaded in.

3.4 Web application

A simple web application was created to showcase the model, as shown in Figure 16. The inference API was developed using FastAPI, a Python backend framework. The user interface was built in TypeScript using Next.js, a widely used React framework, and utilizing the Chakra UI library.

Users have the choice to upload their own audio file. Additionally, for demonstration purposes, the client allows users to select sample audio and preview corresponding LilyPond scores retrieved from the server. Currently, users are required to select the desired model for audio transcription; one out of the three models trained on each dataset. Upon clicking the button, a request is sent to the server containing the file name, Base64 encoded audio, and model name. The request handler receives the request, decodes the audio, and stores it in a temporary directory. The audio is loaded and resampled to 16 kHz. The selected model is then loaded and processes the audio input to generate predicted labels, embeds them into a boilerplate score template, and saves the resulting score to a temporary directory. Subsequently, it invokes the LilyPond CLI in a subprocess to compile the score into a PDF file. Finally, a response containing the file name, raw LilyPond score, and Base64 encoded PDF document is sent back to the client. Users can preview the scores on the user interface and download the PDF if desired.

(a) (b)



(c) (d)

Figure 16. (a) Application menu. (b) File upload with LilyPond output. (c) Sample selection with LilyPond output and label preview. (d) PDF preview.

4. Results

Sections 4.1 and 4.2 present the model's training and testing results, respectively. Section 4.3 highlights issues concerning automatic audio rendering within the data generation pipeline.

4.1 Training

Figure 17 shows the training loss curve of the model trained on dataset B across 20 epochs. The loss demonstrates a clear downward trend but exhibits considerable fluctuations. We hypothesize that these fluctuations are likely attributed to the small batch size of 1 sample utilized during training; encountering a particularly challenging sample may result in significant error spikes.

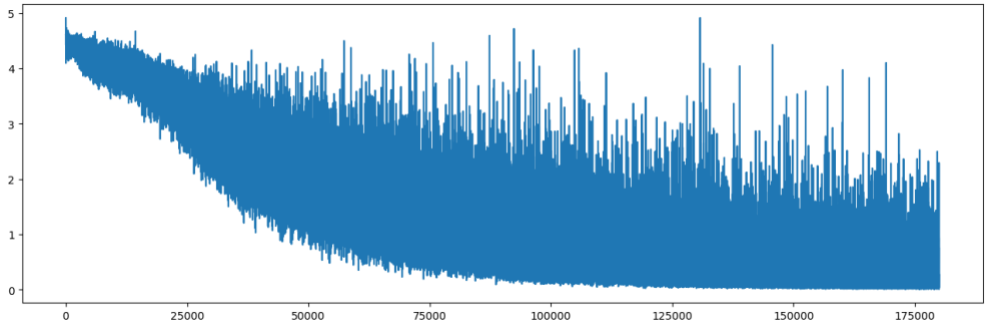


Figure 17. Cross-entropy loss curve. Shows a general decreasing trend but with significant fluctuations.

4.2 Testing

Table 4 displays the mean word error rate (WER) and character error rate (CER) for the identical model architecture trained on each dataset. This is calculated by averaging the total WER and CER over all testing samples. Remarkably, the model achieved both average WER and CER of less than 10% for all three datasets, which is quite satisfactory at this stage. It is expected to observe an increase in error as the vocabulary size expands, as the same amount of data, model parameters, and training parameters were maintained.

Dataset Version	Vocabulary Size	Word Error Rate	Character Error Rate
A	48	2.61%	2.32%
B	84	5.56%	4.56%
C	105	9.32%	7.83%

Table 4. Mean word and character error rates for each dataset.

4.3 Data generation pipeline

The audio generation process utilizing Reaper and the Ultraschall plugin presents significant challenges for scalability and long-term use. We encountered major memory issues involving either the plugin or Reaper itself, necessitating the generation of audio in batches of approximately two to three hundred at a time. Consequently, generating even larger datasets poses a considerable problem at present. Given that automatically rendering audio in this manner is not a conventional task, there is currently a lack of suitable tools for efficiently rendering audio from MIDI and virtual instruments.

5. Future Work

While demonstrating commendable performance, there remains considerable work ahead for the model to effectively handle real-world data. Firstly, addressing the fluctuating loss curve during training is crucial. This could involve increasing the batch size and devising methods to handle padding and masking of audio input. Adjusting the learning rate while augmenting the dataset size could also be beneficial.

To better emulate real music, several approaches can be considered. For instance, expanding the vocabulary size by introducing upper octaves while accounting for the instrument's maximum range can be explored. Currently, the supported range spans from E1 to B2, whereas standard tuned 4-string bass guitars can reach as high as Dsharp4, though this is uncommon. Regarding generation rules, incorporating musical structures via algorithmic music generation could prove to be beneficial, although it may extend a little beyond the current scope of the project. For instance, instead of randomly generating notes using the chromatic scale, different combinations of scales and keys could be introduced. Contextual information may then be incorporated accordingly. To enhance sound quality, various noises, effects (like distortion, fuzz, reverb), or virtual instrument variants can be used. Additionally, modeling human performances could involve introducing timing imperfections by randomly adjusting MIDI data – like the 'humanize' feature available in Reaper. Regarding the previous challenge of modeling playing styles, this could be accomplished by utilizing virtual instruments (usually not free) that contains those samples, along with incorporating additional articulation symbols in LilyPond.

While getting more data is always advantageous, the development of a more efficient audio generation tool is imperative to facilitate the generation of vast amounts of audio data. This might necessitate a separate project altogether.

As the complexity of the data grows, exploring larger models becomes essential. This may involve simply augmenting the number of features, layers, and heads. Given the advancements in large language models that are based on Transformers, this approach to music transcription holds significant potential.

6. Conclusion

In this report, we introduced a proof-of-concept end-to-end transcription model for bass guitar using Transformers, enabling direct conversion of audio into Western notation in Lilypond format. This streamlines the process, eliminating the need for multiple stages that can be cumbersome and prone to errors. Moreover, we tackled the challenge of limited data availability by establishing our own synthetic audio generation pipeline to produce bass guitar audio with corresponding notation. Leveraging insights from previous studies, our model has shown promising results in testing, boasting relatively low word and character error rates on the dataset. However, it remains in its early developmental phase and relies heavily on certain assumptions, particularly regarding the synthetic data generated. Further exploration is essential, encompassing both the refinement of the model architecture and the expansion of the dataset, to ensure robust generalization and practical utility for bassists in future iterations. This includes refining the audio generation methodology, exploring larger model architectures, integrating more realistic musical structures, and potentially modeling various playing styles.

7. References

[1] E. Benetos, S. Dixon, Z. Duan and S. Ewert, "Automatic Music Transcription: An Overview", 2019. doi: 10.1109/MSP.2018.2869928.

[2] J. Gardner, I. Simon, E. Manilow, C. Hawthorne, and J. Engel, “MT3: Multi-Task Multitrack Music Transcription.” arXiv, 2021. doi: 10.48550/ARXIV.2111.03017.

[3] R. G. C. Carvalho and P. Smaragdis, "Towards end-to-end polyphonic music transcription: Transforming music audio directly to a score," 2017, doi: 10.1109/WASPAA.2017.8170013.

[4] Bittner, R. M., José Bosch, J., Rubinstein, D., Meseguer-Brocal, G., and Ewert, S., “A Lightweight Instrument Agnostic Model for Polyphonic Note Transcription and Multipitch Estimation”, 2022. doi:10.48550/arXiv.2203.09893.

[5] J. W. Kim, J. Salamon, P. Li, and J. P. Bello, “CREPE: A Convolutional Representation for Pitch Estimation.” 2018. doi: 10.48550/ARXIV.1802.06182.

[6] B. McFee, C. Raffel, D. Liang, D. P. Ellis, M. McVicar, E. Battenberg, and O. Nieto. “librosa: Audio and music signal analysis in python.” In Proceedings of the 14th python in science conference, pp. 18-25. 2015.

[7] W. Dai, C. Dai, S. Qu, J. Li, and S. Das, “Very Deep Convolutional Neural Networks for Raw Waveforms,” Oct. 2016, doi: https://doi.org/10.48550/arxiv.1610.00087.

[8] J. Abeßer, “IDMT-SMT-Bass”, 2010. https://www.idmt.fraunhofer.de/en/publications/datasets/bass.html

[9] H. G. Michael Clausen, M. Müller, “Estimating Musical Time from Performed MIDI Files”, 2014. doi: 10.5281/zenodo.1417924

[10] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever, “Robust Speech Recognition via Large-Scale Weak Supervision”, 2022. doi:10.48550/arXiv.2212.04356.

[11] A. Vaswani et al., “Attention Is All You Need”, 2017. doi:10.48550/arXiv.1706.03762.

[12] K. Choi, G. Fazekas, K. Cho, and M. Sandler, “A Tutorial on Deep Learning for Music Information Retrieval.”, 2017. doi: 10.48550/ARXIV.1709.04396.

[13] Dolmetsch, “Music theory & history online”. https://www.dolmetsch.com/theoryintro.htm

[14] J. Gibson, “Introduction to MIDI and Computer Music”. https://cecm.indiana.edu/361/midi.html

[15] CCRMA, “Standard MIDI File Structure”. https://ccrma.stanford.edu/~craig/14q/midifile/MidiFileFormat.html

[16] MasterClass, “Music 101: What Is Harmony and How Is It Used in Music” https://www.masterclass.com/articles/music-101-what-is-harmony-and-how-is-it-used-in-music

[17] Britannica, “Equal Temperament”. https://www.britannica.com/art/equal-temperament

8. Appendices

A. Western Classical Notation

This section offers a glimpse into fundamental music theory and key elements within this notation system, aiming to provide context for the project. While it covers a significant amount of information, it represents only a basic and limited subset. Readers can explore https://www.dolmetsch.com/theoryintro.htm, which serves as a comprehensive resource heavily referenced in this section.

Music is represented as a collection of notes, which serve as discrete representations of sound and carry details regarding their pitch (perceived frequency), onset time, and duration. The initial seven letters of the alphabet are employed to signify increasing pitch in a cyclical manner: A B C D E F G A and so forth. These correspond to the white keys found on a piano keyboard. The interval between the two A notes in this instance is referred to as an octave.

Notes corresponding to the black keys are denoted by letters followed by pitch modifiers known as accidentals. A sharp (♯) raises a note's pitch by a half step, while a flat (♭) lowers it by half a step. Consequently, the black key directly above and to the right (a half step higher) of a C is referred to as C sharp. However, it can also be interpreted as a half step lower than D and is thus labeled D flat. This occurrence, where the same pitch is represented by different notations, is called enharmonic equivalence.

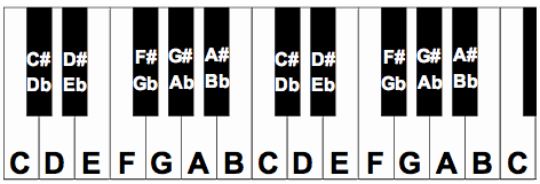


Figure 18. A piano keyboard with names of notes.

Notes are positioned on a grid, usually composed of five horizontal lines known as a staff (or stave), and are read from left to right. A note may be placed directly on a horizontal line or between two lines. The vertical placement conveys the pitch of the note, with a higher position denoting a higher pitch. Ledger lines may be employed to extend the score's range.

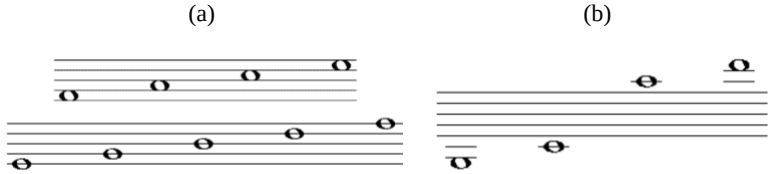


Figure 19. (a) Note placements on staffs. (b) Ledger lines.

A clef symbol, positioned at the start of the staff, establishes a reference pitch for notation. Two commonly used clefs are the Treble clef and the Bass clef. The Treble clef, also called the G clef, is a symbol that loops around the second horizontal line from the bottom, representing the position of the note G4, approximately 391.995 Hz (in Helmholtz pitch notation). Typically, this clef is employed for higher-pitched notes. The Bass clef, known as the F clef, curls around the fourth horizontal line from the bottom, indicating the position of F3, approximately 174.61 Hz. It is utilized for instruments with lower pitches. (a)

(b)

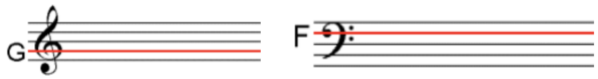


Figure 20. (a) Treble or G clef. (b) Bass or F clef

The duration or subdivision of a note is conveyed through its shape. Table 5a displays a selection of notes with their respective durations and names. Note durations are commonly depicted using fractional units. Rests signify intervals of silence and are represented by a distinct set of shapes, as depicted in Table 5b. Thus, both the horizontal position and the shape together indicate when notes should be played. (a)

(b)

Sign	number equal to 1 semibreve	English	American	Italian	French
𝅜	1/2	breve or brevis	double-whole note	breve (f.)	carrée (f.) or brevis or double-ronde (f.) (meaning square)
𝅝	1	semibreve	whole note	semibreve (f.) intero (m.)	semi-brève or ronde (f.) (meaning round)
𝅗𝅥	2	minim	half note	minima (f.) or metà (f.) or bianca (f.)	blanche (f.) (meaning white)
♩	4	crotchet	quarter note	semiminima (f.) or nera (f.) or quarto (m.)	noire (f.) (meaning black)
♪	8	quaver	eighth note	croma (f.) or ottavo (m.)	croche (f.) (meaning hook)
𝅘𝅥𝅯	16	semiquaver	sixteenth note	semicroma (f.) or sedicesimo (m.)	double croche (f.) (meaning double hook)
𝅘𝅥𝅰	32	demisemiquaver	thirty-second note	biscroma (f.) or trentaduesimo (m.)	triple croche (f.) (meaning triple hook)

Rest	number equal to 1 semibreve	English	American	Italian	French
𝄻	1/2	breve rest	double-whole rest	pausa di breve (f.)	bâton (m.) or pause de brève (f.) or silence de brève (m.)
𝄼	1	semibreve rest	whole rest	pausa di semibreve (f.)	pause (f.)
𝄽	2	minim rest	half rest	pausa di minima (f.)	demi-pause (f.)
𝄾 or 𝄿	4	crotchet rest	quarter rest	pausa di semiminima (f.)	soupir (m.)
𝄿	8	quaver rest	eighth rest	pausa di croma (f.)	demi-soupir (m.)

Table 5. (a) Example of note shapes, durations, and names. (b) Rest shapes, durations, and names.

Notes on a staff are organized into measures or bars, delineated by vertical lines. These measures serve as compact groupings of notes, facilitating musicians in their reading and interpretation. For example, it is far more convenient to instruct musicians to begin at bar 128 rather than specifying the 700th note.

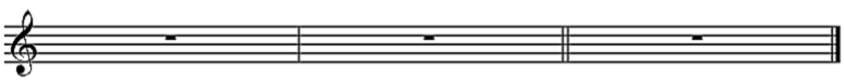


Figure 21. Three empty bars on a staff separated by a single bar line, double bar line (to mark end of sections), and terminal bar line (end of a musical piece).

To determine how many durations worth of notes should be grouped into a bar, a time signatures is used. It is a fractional metric consisting of two numbers: the numerator represents the number of notes to be played in a bar, while the denominator indicates the type of note to be played. A time signature of 3/4 indicates three quarter notes in a bar, while 6/8 indicates six eighth notes per bar. Notice that mathematically, both time signatures occupy the same overall duration. Musicians may argue and debate about their differences in terms of note groupings and accents used.

The speed of a piece of music is commonly measured in beats per minute (BPM), where a beat typically corresponds to a quarter note's duration. In classical music and before the invention of metronomes, tempo markings are used to estimate the speed. Table 6 shows common Italian tempo markings with corresponding estimated BPM values.

Tempo Markings	Definition	Beats per minute (bpm)
Italian		
setting the tempo We thank Tanya Tintner for correcting the spelling of Maelzel		
grave	very slow and solemn	40 bpm or slower (a 1950 metronome suggests 44 bpm)
larghissimo	extremely slow	40 bpm or slower
lentissimo	extremely slow, but not as slow as larghissimo	
adagissimo	extremely slow, but slower than largo	
largo	broad, very slow and dignified	42-66 bpm (a nineteenth-century Mälzel (or Maelzel) metronome suggests 40 bpm) (a 1950 metronome suggests 46 bpm) (a modern electronic metronome suggests 50 bpm)
larghetto	less slow than largo	60-66 bpm (a 1950 metronome suggests 50 bpm) (a modern electronic metronome suggests 60 bpm)
largamente	broadly	
adagio	slow, but not as slow as largo	58-97 bpm (some sources suggest 66-76 bpm while others suggest 48-66 bpm) (a nineteenth-century Mälzel (or Maelzel) metronome suggests 60 bpm) (a 1950 metronome suggests 54 bpm) (a modern electronic metronome suggests 70 bpm)
adagietto	slow, but less slow than adagio	
lento	slow	52-108 bpm (a nineteenth-century Mälzel (or Maelzel) metronome suggests 52 bpm) (a 1950 metronome suggests 52 bpm)
lentamente	slowly	
andantino	a little slower than andante but sometimes a little faster than adagio	(a 1950 metronome suggests 66 bpm)
andante	moving along - walking pace	56-88 bpm (some sources suggest 76-108 bpm) (a nineteenth-century Mälzel (or Maelzel) metronome suggests 69 bpm) (a modern electronic metronome suggests 80-100 bpm)
con moto	with movement, or a certain quickness	
moderato	moderate speed	66-126 bpm (some sources suggest 120-168 bpm) (a nineteenth-century Mälzel (or Maelzel) metronome suggests 84 bpm) (a 1950 metronome suggests 80 bpm) (a modern electronic metronome suggests 110 bpm)
allegretto	pretty lively	(a nineteenth-century Mälzel (or Maelzel) metronome suggests 100 bpm) (a 1950 metronome suggests 100 bpm)

Table 6. Tempo markings in Italian, commonly used in classical music pieces, along with their definitions and estimated BPM values.

B. Musical Instrument Digital Interface (MIDI)

MIDI is communication protocol used by digital instruments. Devices connected to a MIDI bus communicate using MIDI Messages that describe events that occurred, like a key press on a digital piano for instance. Up to 16 channels (independent message paths) can be supported. A MIDI file consists of chunks of bytes, including a header, track, and events. Readers can refer to this page for a detailed specification of the file structure.

C. Examples of High-Level Musical Concepts

This short section briefly highlights three core concepts in music: rhythm, harmony, and melody.

Rhythm refers to temporal placements and patterns of notes. This can be realized using combinations of different note durations, volume, or groupings for instance.

Harmony may be considered as a combination and series of notes. It is an abstract product of a cohesive group of voicings.

Intuitively, melody can be thought of as the part of music that you can sing to. It is also a collection of notes as a single entity.

D. 12-Tone Equal Temperament Tuning

In music, frequencies can be divided in different ways. In modern Western music, the 12 Tone Equal Temperament Tuning system is most frequently used, with the note A4 centered at 440 Hz as a reference point. This involves dividing an octave into 12 equally spaced intervals called semitones on a logarithmic scale. Each subsequent semitone is $\sqrt[12]{2}$ or approximately 1.059 times the previous frequency. For example, the subsequent note of A4 (440 Hz) is Asharp4, which is $440 * \sqrt[12]{2} = 466.16$ Hz.