Tensor Hero: Generating Playable Guitar Hero Charts from Any Song

Elliott Waissbluth
ewaissbluth
@berkeley.edu

Samantha Carr
samantha.carr
@berkeley.edu

Alex Popescu
alexpo7
@berkeley.edu

Jackie Hu
jackiehu
@berkeley.edu

Abstract

In this paper, we summarize our progress towards creating a system capable of ingesting raw audio files and outputting playable Guitar Hero charts which are matched to the musical structure of that audio. We refer to this task as *abstract transcription*. Drawing on work from music information retrieval (MIR), machine learning, and signal processing, we formulate and review candidate models to accomplish this task, including all data preprocessing and model training steps explored. In addition to abstract transcription, we also detail a system used for *difficulty conversion*, the task of taking a dense and fast Guitar Hero chart and simplifying it to a more sparse and easy format. We detail our model selection and evaluation processes and summarize progress as of the submission of this report.

1 Introduction

Guitar Hero is a rhythm game with a simple guitar-like controller used to play simplified musical scores that simulate the experience of playing songs. It is the goal of the player to follow this score, or *chart*, as closely as possible by using the controller to input each note at the correct time as it scrolls by on the screen. Guitar Hero charts are designed manually in a process which requires slowly parsing a song, second by second, and manually adding and adjusting notes to best capture the motifs, textures, and melodies in the underlying audio.

After Guitar Hero was released in 2005 as a console game, it didn't take long for open source developers to latch onto it and create PC alternatives. The most popular PC Guitar Hero alternative is called *Clone Hero*. These open source alternatives democratized the charting process and were conducive to the formation of a large community with active charters releasing new charts at regular intervals. It is from these community charts, as well as the original Guitar Hero charts, that we have gathered our data.

We aim to create a system that takes on the role of the charter, listening to raw audio and labeling a corresponding Guitar Hero chart. In addition, we take on the task of *difficulty conversion* to translate difficult charts into simpler ones.

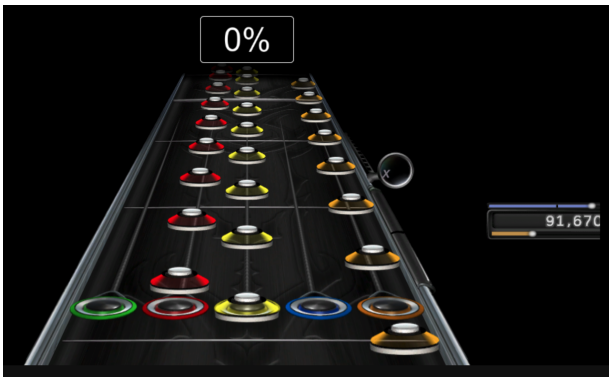

Figure 1: Clone Hero Interface, Display of Chart files

2 Problem Definition

2.1 Abstract Transcription

Broadly speaking, our aim is to create a system that predicts Guitar Hero charts given an audio input. When *charters* score a song, they are limited to the controller's simple five button layout. This forces the charter to simplify the full gamut of notes into a representation that captures its essence within the confines of the controller. This task is best described as *abstract transcription* because the song is transcribed to a simplified score abstracted from its full tonal complexity. Building off of the heuristics defined in prior rhythm game labeling systems (Donahue et al., 2017), our problem can be decomposed into two subtasks: note placement and note selection.

Note placement is concerned with placing notes at correct temporal locations given an audio input.

This task is intrinsically linked to *onset detection*. In MIR, an *onset* is the time marking the beginning of a musical transient (Bello et al., 2005). In other words, an onset is the time at which a unique note or chord begins. Guitar Hero notes are placed primarily at times corresponding to the onsets of the driving melody of the song.

Note selection is the task of selecting the correct note for any given onset. In classical transcription, notes are labeled using the chromatic scale. This provides a nearly objective link between underlying frequency and note label. In the simplified 5-button space of Guitar Hero, this objective link vanishes. A charter could map any underlying frequency to any note because they focus on capturing the patterns present in the melody rather than the exact tones being played. For example, an ascending triplet of chromatic notes, C5 $\rightarrow$ D5 $\rightarrow$ E5, could be mapped to any ascending combination of three Guitar Hero buttons without losing any perceived accuracy by the player.

In order to avoid this labeling ambiguity, we select notes by predicting *contour*, rather than exact buttons, from audio sequences. Contour carries information about the *shape* of a note sequence rather than its precise note selections. To generate a chart, we predict a contour representations then decode it into a note sequence. We describe our contour formulation in more detail in 4.2.2.

2.2 Difficulty Conversion

In every commercial Guitar Hero game, each song can be played at four difficulties: `easy`, `medium`, `hard`, or `expert`. The game designers included these difficulties to make the game accessible and fun for everyone regardless of their skill-level. In the modern charting community, a vast majority of players are at or above the expert level and so most user-created charts are prohibitively difficult for the average player. We detail an automatic system to convert an `expert` chart into one that is `hard`, `medium`, or `easy`, to increase the accessibility to community created charts.

3 Related Work

3.1 Audio Transcription

There are several papers published in the audio transcription research field which we referenced in our own research endeavors this semester. In general, there have been many advancements in AMT (Automatic Music Transcription) field, using various neural architectures to support the goal of ascribing symbolic representational outputs from audio file inputs. For domain specific tasks such as piano transcriptions, high accuracy was achieved using methods involving spectrogram analysis and frequency detection, in combination with onset detection algorithms. (Kwon et al., 2020), for example, successfully used an NLP language model to match note-states to pitch. (Kong et al., 2021) were able to transcribe piano audio through regression on the onset and offset times. For purely audio transcription tasks, these methods have been highly successful.

Figure 2: Guitar Hero Controller

3.2 Transformers

Since our endeavor involves not only the detection of notes and their patterns, but also an abstraction of note patterns needed to create the output chart, we leaned in the direction of a neural network, which is capable of learning these abstract relationships. The most related work informing our research was a paper called Sequence to Sequence Piano Transcription with Transformers (Curtis Hawthorne, 2021), which used a Transformer architecture trained on small sequences of piano audio data to output piano transcriptions. We proceeded with a similar Transformer architecture, as described below, given the success of the piano model for their transcription task.

4 Methods

4.1 Dataset

We primarily collected our training samples from the original Guitar Hero games: *Guitar Hero I, II, III, World Tour, 5*, as well as other expansion packs. We also collected samples from community posted data available to *Clone Hero* Users. Specifically, we used the *Angevil Hero II, Anti Hero, Anti Hero 2, Community Track Pack 6, Digitizer, Facelift Pack*

1, Facelift Pack 2, Focal Point, Guitar Hero X, and *Paradigm* track packs. Each track pack contains multiple songs and each song packages both an audio file, typically `.ogg`, and corresponding chart file, `.chart`, which encodes the stream of notes presented to the user as they play the song.

After failing to converge on a large, diverse dataset, we wanted to narrow our model's training objective towards a specific genre, so we focused on *rock*. First, we filtered all the *Guitar Hero* and *Clone Hero* songs to exclude all genres that were not a subgenre of *rock*, this was accomplished through a combination of manual evaluation and the `genre` tag present in every `.chart` file. We also listened to snippets of each audio file to ensure that the songs primarily featured strong lead guitar melodies that could be clearly differentiated from any other mid-tone melodic instrument. Some `.chart` files cause miscellaneous errors in our preprocessing pipeline. After filtering, our dataset consists of 450 songs (100 samples from *Clone Hero* and 350 samples from *Guitar Hero*).

4.2 Guitar Hero Chart Preprocessing

4.2.1 `.chart` to Notes

A `.chart` file is a UTF-8 encoded text file detailing the notes which comprise a Guitar Hero song. At its core, a `.chart` file is a musical score, as it contains a list of timestamps and corresponding notes. Each `.chart` includes a time signature, T (beats per measure), resolution, R (*ticks*, i.e. note slots, per beat), and tempo, B (beats per minute). Notes can only be placed at these discrete ticks. In practice, the time signature and BPM may change hundreds of times per song to align with the artist's performance.

To simplify this representation, we first modify each `.chart` file to have a constant resolution, tempo, and time signature. We set this at $T = 1$ beats-per-measure, $B = 31.25$ BPM, and $R = 192$. Taken together, these parameters imply a tick resolution of 10ms. Notes are specified to the tick, and conversion often leaves notes labeled at intervals between 10ms ticks. We snap these notes to their closest tick, an imperceptible change for the player.

For our initial investigation, we have elected to simplify charts further by eliminating held notes and replacing each one with a single, non-held note at its onset. This allows us to discard the need for offset detection in addition to onset detection. We have also simplified the notes to be free from *modifiers*, which overrides the game's default note behavior. The modifiers are described in Table 1.

Modifier	Description
Regular	Refers to notes with no modifier
Force	Guitar Hero registers notes placed close together as hammer-ons, which don't require strumming to play but do require the previous sequence of notes to be played correctly. Force notes override this behavior and require strumming explicitly.
Tap	Tap notes are like hammer-ons in that they don't require strumming to play, however, there is no requirement to play the previous sequence of notes correctly.

Table 1: Guitar Hero Note Modifiers

With 5 buttons, there are 30 combinations of notes that may be pressed as well as an *open* note, which is played by strumming the trigger without pressing any buttons. We one hot encode these 31 notes for representation to our systems. Therefore, the notes can be represented as the set $N = \{n \mid n \in [1, 31]\}$

We finally represent entire note sequences as *notes arrays*. A notes array is a 1D array with every index corresponding to a 10ms tick in a song. If there is no note event at a particular tick, we hold the value with a 0. If there is a note, the one hot index of the note is placed at that tick. For example, a 50ms song with a single red note ($n = 2$) at 30ms would be written as `[0, 0, 2, 0, 0]`.

4.2.2 Note Contour Representation

The pivot in our initial objective from an automatic music transcription model, which was a more literal translation, to an abstracted transcription model required a novel note representation. Here we describe the concept of a *note contour*, which encodes the notes in a way that captures the melodic patterns of a song. First, we assigned an ordinality to the set of notes $N = \{n \mid n \in [1, 31]\}$, where they are sorted according to the diagram below. The ordinality is segmented by *note plurality* P, where $|P| = 13$ and describes the number and general shape of held buttons.

Additionally, the concepts of *motion* and *anchor* are critical to note contours. Given a note plurality

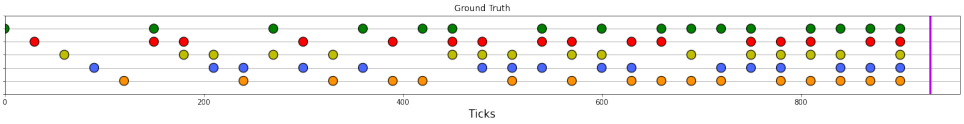

Figure 3: Guitar Hero notes organized by note plurality

Note Plurality	Description	Element
s single	Single notes	`s[0]` = G `s[1]` = R `s[2]` = Y `s[3]` = B `s[4]` = O
$d0$ double 0	Double notes with 0 spaces between them	`d0[0]` = GR `d0[1]` = RY `d0[2]` = YB `d0[3]` = BO
$d1$ double 1	Double notes with 1 space between them	`d1[0]` = GY `d1[1]` = RB `d1[2]` = YO
$d2$ double 2	Double notes with 2 spaces between them	`d2[0]` = GB `d2[1]` = RO
$d3$ double 3	Double notes with 3 spaces between them	`d3[0]` = GO
$t0$ triple 0	Triple notes with 0 spaces between them	`t0[0]` = GRY `t0[1]` = RYB `t0[2]` = YBO
$t1$ triple 1	Triple notes with 1 space between the 2nd and 3rd highest note	`t1[0]` = GRB `t1[1]` = RYO
$t2$ triple 2	Triple notes with 1 space between the 1st and 2nd highest note	`t2[0]` = GYB `t2[1]` = RBO
$t3$ triple 3	Triple notes with 2 spaces between all the notes	`t3[0]` = GRO `t3[1]` = GYO `t3[2]` = GBO
$q0$ quad 0	Quad notes with 0 spaces between them	`q0[0]` = GRYB `q0[1]` = RYBO
$q1$ quad 1	Quad notes with 1 space between all the notes	`q1[0]` = GRYO `q1[1]` = GRBO `q1[2]` = GYBO
p pent	All notes held simultaneously	`p[0]` = GRYBO
o open	Open note	`o[0]` = open

Table 2: Set of Note Pluralities

$p \in P$, the anchor a is the index of the note within the category. For example, if we are looking at a note represented by holding down the green and red buttons (i.e. GR), then that note would be represented by the double 0 note plurality at anchor 0 (i.e. `d0[0] = GR`). Each note prediction is the combination of a note plurality and a motion. The motion, $m \in [-4, 4]$ describes the gap between subsequent notes. If the current note is `d0[0] = GR` and the subsequent note plurality and motion predictions are `d1` and $+2$ respectively, the following note would be `d1[0+2] = YO`. The note plurality For a given note plurality $p \in P$, if $a + m \geq |p|$, then the anchor wraps back around such that the new anchor a is set to 0. Likewise, if $a + m < 0$, then the new anchor a is set to $|p| - 1$.

In what we thought would later be implemented into a custom loss function, we designed a loss weighting function designed to help the model differentiate more successfully between notes and timestep predictions. Since the model's predictions come out as a probabilities array for all timesteps and notes, we formulated a matrix of weights designed to punish incorrect probability values based on their distance from the correct note plurality or timestep. Generally, the largest weight was given to notes predicted as timesteps, and vice versa. From there, a smaller weight was given for incorrect note plurality, and an even smaller plurality for incorrect elements within the correct plurality. For example, a green single note predicted as a double note would be summed to a higher weight than a green single note predicted as a blue single note. The weights were then averaged across the full matrix of output probability arrays to arrive at a single value weight for that output series. However, we failed to design the function in a way to allow gradients to flow through.

4.3 Audio Preprocessing

4.3.1 Source Separation

Before making any predictions about note onsets and sequences, we preprocess the audio to extract the melodic component most likely to map to the Guitar Hero transcription. By extracting this com-

ponent, we free the onset detection and contour prediction algorithms from superfluous information.

We achieve this using musical source separation, the task of splitting a layered musical audio file into its component parts, e.g. drums, vocals, percussion, guitar, etc. For this, we utilized the 4-stem model from the *Demucs* library, which separates audio files into *vocals, drums, bass,* and *other*. Empirically, we found *other* to contain melodies most closely tracked to the Guitar Hero representation, as it tended to be the extracted guitar melody, so we utilized it for our models.

4.3.2 Spectrogram Computation

All audio transformations are completed using the *librosa* library (McFee et al., 2021). In our training data, audio is saved in `.ogg` or `.mp3` format. For consistency, we resample all songs from their original sample rate to 44100 kHz. We compute the log-mel spectrogram of each file with 512 mel bins, an FFT length of 4096 samples, and hop length 441. These parameters produce spectrograms with frequency frames separated at 10ms intervals, which is convenient as it allows us to map notes arrays to frequency frames directly. The spectrogram for each song is normalized in [0,1].

4.4 Model

4.4.1 Onset Detection

In order to determine the note placement, we implemented the onset detection method described in (Mounir et al., 2016). The *onset detection function (ODF)*, referred to as *NINOS*, is a normalized, inverse-sparsity measure that calculates the *discrete Fourier transform (DFT)* of each spectrogram frame and uses the l_2-norm and l_4-norm to generate a joint sparsity and energy per frame. The output of the ODF is then passed through the *peak-picking (PP)* operation described in (Böck et al., 2012). This hyper-parameterized operation selects frames as onsets according to their relative values calculated by the ODF. We optimized our hyperparameters using a grid search tested on six representative 10 second samples of audio.

4.4.2 Architecture

Our transformer model is based on the original transformer proposed in (Vaswani et al., 2017), as this is the default PyTorch implementation. Our model is configured with an embedding size of $d_{model} = 512$, a feed-forward output dimensionality of $d_{ff} = 1024$, a key/value dimensionality of $d_{kv} = 64$, 8-headed attention, and 2 layers in each the encoder and decoder.

The model takes continuous spectrogram frames as input, and therefore it is necessary to replace the embedding layer at the input with a dense layer shared across all windowed frames of each spectrogram input. The dimensionality of the dense layer is $d_{in} = 512$ and is followed by a sigmoid activation function. We add positional embeddings to the output of the dense layer prior to feeding the input to the encoder.

4.4.3 Inputs and Outputs

The data representation for our model is modeled after that proposed in (Curtis Hawthorne, 2021). Pre-computed spectrograms are split into four second chunks, with each chunk being considered a training example. During training, the ground truth onset times are used to extract a corresponding spectrogram frame, as well as the three preceding and three subsequent frames. This gives each onset a 70ms audio representation. Before being fed into the encoder, each of these windowed spectrograms are flattened and fed through a shared dense layer. During inference, spectrograms are split into four second segments and fed to through the transformer individually before a final concatenation.

The model output is a softmax distribution over a discrete vocabulary of note contour events. Each note contour event is encoded as two subsequent predictions, (1) The note plurality, p, (2) The motion, m. Therefore, the transformer learns to output predictions as pairs (note plurality, motion). The possible note pluralities and motions are held in a 25D vector, where 13 spaces are reserved for note pluralities, 9 spaces are reserved for motions, and three spaces are reserved for `<sos>`, `<eos>`, and `<pad>`. The decoder autoregressively samples outputs until an end-of-sequence token is produced. All outputs on this vector are passed through a 512D embedding layer prior to being autoregressively fed into the decoder.

4.4.4 Training

The model is trained end-to-end, without pretraining, using a fixed learning rate `1e-4`, dropout 0.1, and a batch size of 16. Prior to training, we split our dataset into a train-val-test set of 0.9-0.05-0.05. We utilize a NVIDIA V100 GPU hosted on Google Colab to train. The loss function is binary cross-

entropy.

4.5 Difficulty Conversion

The training data set used for generating a difficulty conversion function consists of 124 Guitar Hero songs for which we had charts for all levels of difficulty: `expert`, `hard`, `medium` and `easy`. We then built three probability transition matrices that represent the probability of a given `expert` chart note transition into another difficulty level's note. The final transition matrices are: expert-to-hard, expert-to-medium, expert-to-easy, in descending playing difficulty.

There are 31 possible note states (including open notes, single notes, chords combinations) in a Guitar Hero chart, thus each difficulty transition matrix will be a 31 x 32 matrix, such that entry (I, J) is the probability of transitioning from expert note I to note J, in either the hard, medium, or easy difficulty level. The probability that any particular note is dropped is included in each of these matrices. The probability of transitioning to any particular difficulty state is dependent solely on the current state of expert note, and not on the sequence of state that preceded it. Additionally, the transition matrix must be a stochastic matrix, a matrix whose entries in each row must add up to exactly 1.

The difficulty conversion function takes a desired output difficulty level, a transition matrix corresponding to that desired output, and outputs the new chart in that desired difficulty.

In general, we noticed a 14 percent drop of notes from expert-to-hard, a 24 percent drop of notes from expert-to-medium, and a 32 percent drop of notes from expert-to-easy. We also validated the output difficulty conversion charts with domain knowledge to confirm that there are more single notes in a hard level note chart compared to an expert level note chart for the same song. The orange note value is dropped completely in medium charts, with more limited chord combinations compared to expert level and hard level. Last but not least, the easy level charts only consist of the top three buttons with mostly single notes, and longer intervals in between notes.

We recently found that reducing the number of possible notes per second and mapping applicable notes to these reduced time bins is a reliable way to choose onsets without losing quality in the playing experiments. In the future, we hope to incorporate this into our difficulty conversion model to circumvent the need for a transition-to-empty column in our transition matrices.

5 Evaluation

5.1 Metrics

To compare our predicted chart files to the true chart files there are two tasks we want to optimize: note placement or onsets, which determine the timing of the notes, and note selection, which determine the note type played at the given time. Evaluation based on onsets is a well defined evaluation space in music information retrieval systems. In our project we specifically used the mir_eval library, and followed strategies outlined in (Raffel et al., 2014).

For note selection, it is important to note that notes in Guitar Hero do not correspond to exact tones or frequencies as is the case for the piano. Instead the Guitar Hero notes are a simplified abstraction of five notes - green, red, yellow, blue, orange from head to base of guitar - that don't correspond to precise frequencies we hear, but rather represent relative notes to each other when being played..

Another thing to note about the notes is that they may be played as single, double, triple or very rarely quadruple or quintuple notes. Notes are played in chords (rather than as single notes) for more pronounced strokes or rhythm in songs. Thus, notes being played in these groupings vs. single is a distinct difference and something we expect to be able to detect in songs and represent in the generated charts.

Given these aspects of notes, it becomes clear why getting notes correctly is more ambiguous with a less objective "correct" answer than getting onsets right. It also becomes clear why the differentiation between single vs grouped notes is a relevant high-level metric. With this in mind, the metrics we chose to evaluate our predicted outputs of guitar hero notes were metrics on onset evaluation and note type distributions.

As mentioned above, there are established ways in the field of music information retrieval systems for onset evaluation. mir_eval's library allows for a given margin of error to compare onsets, the time at which a note was struck (ignoring how long it was held), and can evaluate the prediction as "true" for coinciding onsets and "false" for misaligned onsets where coinciding onsets are considered to be any onset within 20ms of the ground truth. Given this,

it calculates the standard metrics F1, precision and recall for each pair of notes arrays.

With respect to note type distributions, we specifically calculated note frequency saturation, note frequency count and note group type frequency count. Note that frequency saturation is a ratio that calculates the number of predicted notes to the number of true notes. Thus, a number larger than 1 shows a bias towards overprediction of notes and vice versa. For the note frequency count, we count the occurrence of each note in N for the true and predicted notes. Again, this shows us potential biases in over- or under-prediction of certain notes. Lastly, different groupings (e.g. single, double, etc.) represent distinct elements in a song, we also counted the occurrence of each grouping for true and predicted notes to see potential over-/under- prediction biases. By teasing apart onset and note type, we can more clearly understand what to improve our model on. Further, we see that note type prediction is a harder task as there is no "universally true" answer for the note chosen.

Lastly, as mentioned above we did attempt a custom loss function in an effort to help the model minimize loss and learn more effectively. Loss was being monitored while training as an indication metric of model performance and learning. After implementation, we observed that the model's loss was not decreasing in any structured or continuous way, indicating that the model was not passing gradients through our loss function. Therefor, we continued using the binary cross-entropy in our final training of the model.

5.2 Transformer Model Evaluation

In our initial formulation, we configured the transformer to learn note placement and note selection simultaneously. After a long process of hyperparameter tuning, dataset adjustments, and note representation experimentation, we found that this formulation of the model was not sufficient for our task. Predicting notes explicitly was an inherently non-objective task, paired alongside the additional task of onset detection and using a relatively small and noisy dataset (28.6 hours) compared to other datasets used for transcription training (400 hours), we found the initial transformer configuration would not work given our resources. Often times, the model would settle on a local minima and validation accuracy diverged before any meaningful output was produced.

In order to increase the objectivity of our training objective, we created the note contour formulation. Although the note contours were more objective than explicit notes, the model still struggled with onset detection. In our dataset, onsets are particularly hard to predict. The non-percussive nature of most electric guitar melodies makes it difficult to computationally extract the onsets of their notes. It was at this point that we decided to explore other options for onset detection, and leave the machine learning model to learn notes.

At this point, since we were only concerned with the model's note prediction at onsets, we decided to change our input to windows of spectrograms centered around ground truth onsets. This allowed us to increase the model's training throughput as it required a smaller architecture. This provided the most promising note generation heuristic so far, outputting note sequences that are fun to play and relatively intuitive.

6 Future Work

In the future, we plan to further iterate on the note contour representation, as it represents well the abstract, relational nature of the notes being charted. There is more nuance that can be added to better capture and represent the patterns. In addition, the model that interprets these notes and makes predictions on novel data has great room for improvement. Specifically, we will use different architectures and vastly expand the dataset to include a far greater amount of training data than we currently have available.

Additionally, although initially the transformer model was handling the onset detection implicitly, ultimately moving the onset detection into its own preprocessing step will improve the accuracy of note placement within the charts. We found that the onset detection inherently learned within the Transformer model was lacking the precision and patterns found in the manual charts, and the transformer architecture didn't allow for honing in on this aspect specifically. Being able to fine-tune the sensitivity and accuracy of the note placement will improve the player experience noticeably. The onset detection algorithm we implemented is the best we could find out of several candidate algorithms, and its diverse hyperparameters could be optimized dynamically in the future, i.e. they will change depending on properties of the input audio.

Future key work will also include the implemen-

tation of held notes and modifiers into the charts. Held notes are notes whose duration is longer than a single "strum", where the offset is just as important as the onset. The modifiers are outlined in Table 1. Lastly, future work will include adapting different models to each genre of music. This will require expanding our dataset to include many more training examples within each genre, and training models specifically on each genre. This will be key work because fundamentally a fast paced, guitar-solo heavy rock song is a different playing experience from a chord-heavy soft rock song. Differentiating and training separately will allow an even more accurate chart representation for each song.

References

J.P. Bello, L. Daudet, S. Abdallah, C. Duxbury, M. Davies, and M.B. Sandler. 2005. A tutorial on onset detection in music signals. *IEEE Transactions on Speech and Audio Processing*, 13(5):1035–1047.

Sebastian Böck, Florian Krebs, and Markus Schedl. 2012. Evaluating the online capabilities of onset detection methods.

Rigel Swavely Ethan Manilow Jesse Engel Curtis Hawthorne, Ian Simon. 2021. Sequence-to-sequence piano transcription with transformers. *CoRR*, arXiv:2107.09142v1. Version 2.

Chris Donahue, Zachary C. Lipton, and Julian McAuley. 2017. Dance dance convolution.

Qiuqiang Kong, Bochen Li, Xuchen Song, Yuan Wan, and Yuxuan Wang. 2021. High-resolution piano transcription with pedals by regressing onsets and offsets times. *ArXiv*, abs/2010.01815.

Taegyun Kwon, Dasaem Jeong, and Juhan Nam. 2020. Polyphonic piano transcription using autoregressive multi-state note model.

Brian McFee, Alexandros Metsai, Matt McVicar, Stefan Balke, Carl Thomé, Colin Raffel, Frank Zalkow, Ayoub Malek, Dana, Kyungyun Lee, Oriol Nieto, Dan Ellis, Jack Mason, Eric Battenberg, Scott Seyfarth, Ryuichi Yamamoto, viktorandreevichmorozov, Keunwoo Choi, Josh Moore, Rachel Bittner, Shunsuke Hidaka, Ziyao Wei, nullmightybofo, Darío Hereñú, Fabian-Robert Stöter, Pius Friesch, Adam Weiss, Matt Vollrath, Taewoon Kim, and Thassilo. 2021. librosa/librosa: 0.8.1rc2.

Mina Mounir, Peter Karsmakers, and Toon van Waterschoot. 2016. Guitar note onset detection based on a spectral sparsity measure. In *2016 24th European Signal Processing Conference (EUSIPCO)*, pages 978–982.

Colin Raffel, Brian Mcfee, Eric J. Humphrey, Justin Salamon, Oriol Nieto, Dawen Liang, Daniel P. W. Ellis, C Colin Raffel, Brian Mcfee, and Eric J. Humphrey. 2014. mir_eval: a transparent implementation of common mir metrics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need.