TISMIR

Draw and Listen! A Sketch-Based System for Music Inpainting

RESEARCH

CHRISTODOULOS BENETATOS
ZHIYAO DUAN
*Author affiliations can be found in the back matter of this article

ubiquity press

ABSTRACT

In this work we implement an interactive system for filling in missing measures in a monophonic music piece. This system takes a user's hand-drawn curves as input and generates a melody whose rhythm and pitch contour match with the curves. Contrary to previous interactive music inpainting work, users of the proposed system do not need to understand the music notation; they just need a rough idea of the shape of the melody and draw it out. This system is implemented under the variational auto-encoder framework and is enabled by a proposed melody disentanglement scheme to disentangle relative pitch, relative rhythm and musical context. We also create a web-based graphical user interface to facilitate the user interaction. We evaluate the system on a commonly used Irish folk song dataset. Objective and subjective evaluations show that this novel interaction is intuitive and effective for melody inpainting, and the proposed neural approach outperforms two baselines we developed based on previous work, in terms of musicality and fidelity to the user's input.

CORRESPONDING AUTHOR:
Christodoulos Benetatos
Department of Electrical and Computer Engineering, University of Rochester, NY, US
c.benetatos@rochester.edu

KEYWORDS:
music inpainting; human computer interaction; automatic music generation; variational autoencoder

TO CITE THIS ARTICLE:
Benetatos, C., and Duan, Z. (2022). Draw and Listen! A Sketch-Based System for Music Inpainting. *Transactions of the International Society for Music Information Retrieval,* 5(1), 141–155. DOI: https://doi.org/10.5334/tismir.128

1. INTRODUCTION

In the area of automatic music generation, one of the research goals is to design interactive systems that engage users in the creative process. The level of user engagement varies. Some systems allow users to provide a seed input (Mao et al., 2018; Hadjeres and Nielsen, 2020; Huang et al., 2019) or simple controls (Wuerkaixi et al., 2021) to guide the music generation process. Other systems require a constant input stream from the users for real-time accompaniment generation (Dannenberg and Raphael, 2006) or collaborative music improvisation (Lewis, 2000; Benetatos et al., 2020). Another dimension of user engagement is the level of music knowledge that the user is assumed to have. Some systems only assume users to understand high-level semantic meanings such as genre or mood (Mao et al., 2018), while others require users to know basic music theory (Chen et al., 2020) or even keyboard and improvisation skills (Benetatos et al., 2020).

In this work, we focus on the task of *music inpainting*, namely the task of filling in missing measures given a known musical context. Music inpainting can find many applications in computer assisted music composition and human-computer interaction systems for creative use cases. Previous work on music inpainting either does not support user interaction (Pati et al., 2019), or assumes users to know some music theory in order to achieve musically meaningful results (Chen et al., 2020). Our goal is to design a system that engages users into the music inpainting process without assuming a musical background. Specifically, our system[1] allows users to draw curves, and the system will turn them into musically meaningful content matching with the given musical context (see Figure 1).

We achieve this by factorizing a to-be-generated melody into three basic aspects: a) the relative pitch, b) the relative rhythm, and c) music-theory-related information (e.g., key, pitch distribution). Here a) and b) are provided by users by drawing a pitch curve and additional note density curves, while c) is inferred by the system from surrounding measures to complement the intuitive inputs from the user. Previous research has shown that ordinary users, even without any musical background, understand the concepts of *high* and *low* of pitch and *dense* and *sparse* of note events (Dowling et al., 1999). In addition, most people are able to "hum" the (exact or approximate) pitch and rhythm of a melody they have in mind, even without any musical training. Therefore, drawing a relative pitch curve with additional curves to indicate the relative note density, in our view, is a feasible task for a wide range of users. Our proposed system can thus help ordinary users to engage in music inpainting and other collaborative music composition practices.

The contributions of this paper are threefold. First, we propose a novel and intuitive way for users without musical training to control important musical elements (pitch and rhythm) in an interactive melody inpainting process. Second, we design a neural approach that disentangles relative pitch, relative rhythm, and other music-theory-related information (e.g., key, note and onset distributions) for this interaction. Third, objective and subjective evaluations show that the proposed neural approach achieves significantly better results than a rule-based and a genetic algorithm baseline, in terms of musicality and fidelity to the user's input.

2. RELATED WORK

2.1 CONTOUR BASED MUSIC GENERATION

The idea of designing music interaction systems based on human ability of understanding and processing melodic contours has been investigated for decades. The "UPIC" (Xenakis, 1977) is a pioneering work by Iannis Xenakis, which enables users to directly control the characteristics of a waveform, including pitch and dynamics, using freehand drawings on a board. Users can draw many lines which are mapped to oscillators with predefined timbre characteristics. The *highs* and *lows* of the drawn lines control the *highs* and *lows* of each oscillator's pitch using a straightforward mapping. This work has been the inspiration for the development of other works that convert a graphical score to a synthesized audio waveform (or events sent to an external synthesizer), such as "IanniX" (Coduys and Ferry, 2004), "Metasynthesis" (U&I-Software, 1997), and "Music Sketcher" (Thiebaut et al., 2008).

A similar idea was exploited by Berg et al. (2012), where the body movements of a user are used to generate music by mapping the 3D coordinates of all body joints to the parameters of a frequency modulation synthesizer. Even though no drawing is involved here, the movement trajectory of the body parts can be seen as melodic contour drawing.

Additionally, there are systems that focus on symbolic music generation. A famous system is "Hyperscore" (Farbood et al., 2004), which aims to provide opportunities to novices, especially children, to compose music using drawings and other graphical tools. "Hyperscore" provides a melody-pattern-based composition. First, users create their own melodic patterns (each with a different color), and then they compose a piece by drawing curves of different colors on a canvas. The color of a curve indicates the melodic pattern to be played, while its shape indicates transformations on the original melodic pattern.

In Piano Genie (Donahue et al., 2019), users can play melodies on a full 88-key piano by just controlling a small 8-key keyboard, where the 8 keys are lined up from low to high in pitch. The direction of pitch movement of the generated melody follows that on the small keyboard, and the note onsets match the key strokes.

Another more relevant system to ours is included in the notation software named "Pizzicato" (Arpege-Music, 2013), where users can fill in a measure by drawing a pitch curve. However, this is a rule-based system, and it requires users to provide the exact rhythm as well as chord labels for every beat. Furthermore, the final notes are generated by selecting the note contained in the chord that is closest to the pitch curve at each onset position, instead of being dependent on the surrounding measures. As a consequence, to generate music that fits the context, a user needs to understand music notation, rhythm and even chords. Therefore, it is not suitable for ordinary users without a music background.

Finally, "JamSketch" (Kitahara et al., 2018) is a system that generates melodies based on pitch curves drawn by the user, in a real-time improvisation task. A genetic algorithm is designed to determine the notes of the melody, however, the chord progression is given and the rhythmic pattern is selected from a predefined set of rhythms. While this rhythm limitation is addressed in follow-up work (Yasuhara et al., 2019), where another genetic algorithm is used to generate an appropriate rhythmic pattern based on the user's input, this system cannot be used in tasks where a chord progression is not available.

2.2 MUSIC INPAINTING

Music inpainting has been applied to both the audio and symbolic domains. In the audio domain, the inpainting methods try to recover missing data in the waveform, which can occur due to various reasons such as distortions and transmission errors (Adler et al., 2012; Marafioti et al., 2020). The same term has also been used to describe the bandwidth extension problem, where the missing high frequency content has to be estimated (inpainted) from the low frequencies (Greshler et al., 2021).

Different from the audio domain, applications for symbolic music inpainting are not motivated by data recovery problems but by the need for creating new interactive tools for music creation. We will describe two recent neural network based works. In "InpaintNet" (Pati et al., 2019), given the past and future content of a missing part of music, it predicts a latent vector representation of the missing part which is later decoded to the symbolic score format using the decoder of a Variational Autoencoder (VAE). In "InpaintNet", a user cannot guide the generated result. "Music SketchNet" (Chen et al., 2020) builds on "InpaintNet" and tries to solve this interactivity problem by allowing users to specify some music ideas to guide the final result. A user can specify a sequence of note names, or the rhythmic pattern of a to-be-generated measure. Even though this interactivity is very useful, the user has to make decisions based on music theory to achieve harmonically coherent results, which can be difficult for non-musician users.

3. PROPOSED METHOD

3.1 KEY IDEA

The key idea behind our proposed system is the following natural way of modeling a melody. We can think of any melody as an integration of pitch *p*, rhythm *r*, and other music-theory-related information. Pitch can be decomposed into the relative pitch *rp* and the pitch offset (or average pitch level) *po*. Similarly, rhythm *r* can be seen as the sum of relative rhythm *rr* and the rhythm offset (or maximum note duration) *ro*. Music-theory-related information can be inferred from surrounding measures, or the musical context. For example, a melody A consisting of a quarter note followed by two eighth notes and a melody B consisting of an eighth note followed by two sixteenth notes, have the same *rr*, but different *ro* (quarter for A, eighth for B). Finally, the musical *context*, or *c*, describes shared patterns with surrounding measures. Using non-rigorous math language, a melody can be represented as:

$$\begin{aligned} \textit{Melody} = \; & \textit{relative pitch} \\ & \oplus \textit{ pitch offset} \\ & \oplus \textit{ relative rhythm} \\ & \oplus \textit{ rhythm offset} \\ & \oplus \textit{ context}. \end{aligned} \tag{1}$$

We make use of the circled addition symbol to note that this equation is conceptual and does not represent the actual addition of vectors.

When the first four factors of Equation (1) are "precise", the context factor *c* is not needed to infer the melody. In case that the information stored in the first 4 factors is vague, the missing information can be complemented by the context *c*. For example, if the relative pitch factor *rp* only stores the pitch trend but not the exact intervals, the missing pitch information can be approximately inferred from the context *c* (e.g., the key and pitch distribution).

Another extreme case is when the first 4 factors are completely missing. In this case, all the information about the melody has to be inferred from the context *c*. A fully automatic inpainting system such as InpaintNet (Pati et al., 2019) is such an example, where the missing measure is completely estimated from the surrounding measures.

3.2 TASK DESIGN

As shown in Figure 1, the proposed interactive music inpainting task is to fill the missing middle measure of a three-measure excerpt of a monophonic melody by drawing curves.

The user uses the rectangular shaped canvas (Figure 1 middle) to draw a pitch curve (green) and optional note density curves (blue) to guide the melody's relative pitch and relative rhythm. The horizontal axis of the canvas is time covering one bar, while the vertical axis is the pitch axis. The user can also use the two sliders to control the pitch offset and rhythm offset of the generated melody.

From these user inputs, the system generates music for the middle measure and displays it in the score (Figure 1 bottom).

The reason that we allow users to draw optional note density curves is to give them better control of the rhythm of the generated melody. A pitch curve itself may or may not contain important rhythm information. For example, a pitch curve with many prominent peaks and valleys, such as that in Figure 4, should correspond to a melody that has at least as many notes as the peaks and valleys, with onsets around them. On the other hand, a monotonically increasing pitch curve, such as the second curve in Figure 3, does not provide enough information to decide on the number and the location of notes. The optional note density curves (i.e., blue lines in Figure 1) are stacked above or below the pitch curve. The more note density curves are stacked in an area, the higher note density the area has, following a similar concept of beams in music notation.

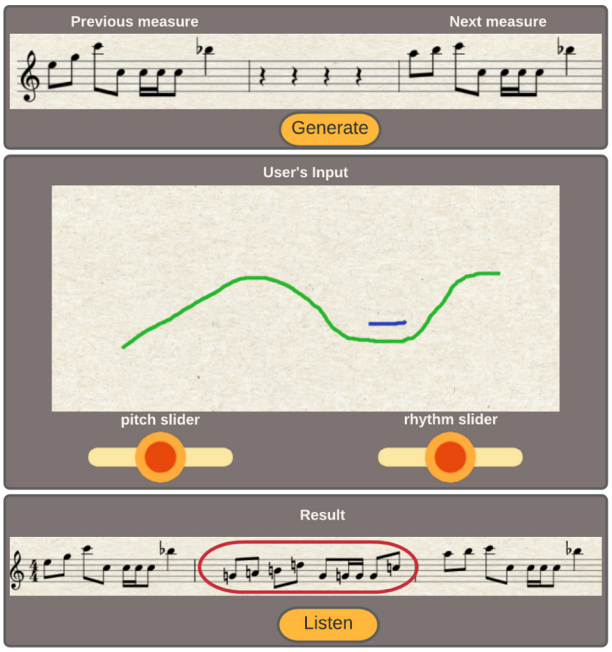

Figure 1: Top: A triplet of music measures. The first and third are the context measures, and the middle one is to be "inpainted". Middle: The user input area consisting of a canvas for drawing the curves and the sliders to control the pitch offset and note density for the whole measure. Bottom: The generated result.

These four types of user input have a direct correspondence to the first four factors of Equation (1). The fifth factor, context, is estimated by the system from the surrounding measures. As explained in Section 3.1, when the first four factors provide less complete (or more vague) information, the generation of the missing melody is more dependent on the context *c*, i.e., the surrounding measures. One way to control the "vagueness" of the first four factors is to apply different levels of quantization to pitch and rhythm representations computed from the drawn curves.

3.3 MODEL ARCHITECTURE

The model we propose is a variational auto-encoder (VAE) (Kingma and Welling, 2014; Higgins et al., 2016) based architecture. As shown on the left side of Figure 2, we have two encoders, Q_{curve} and $Q_{context}$, to process the user's input and the context measures, respectively. The encoders' output forms the VAE's latent vector $Z = [Z_c; Z_{rp}; Z_{rr}]$, i.e., a concatenation of three vectors, where Z_c stores the context information, Z_{rp} stores the relative pitch information, and Z_{rr} stores information about the relative rhythm. On the right side, we use three decoders, P_{rp}, P_r, and P_{midi}, with different intermediate losses to achieve the desired disentanglement of the three latent variables.

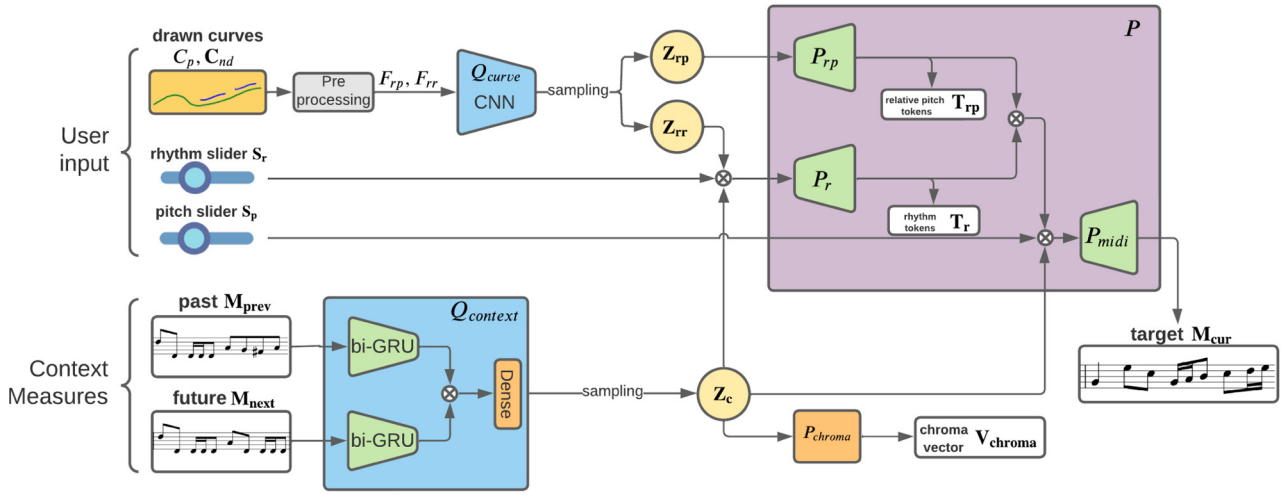

Figure 2: Overall structure of the proposed VAE-based model. Multiple encoders are used to encode user's input and context measures, and multiple decoders are used to achieve the desired disentanglement of latent variables and to generate the missing measure.

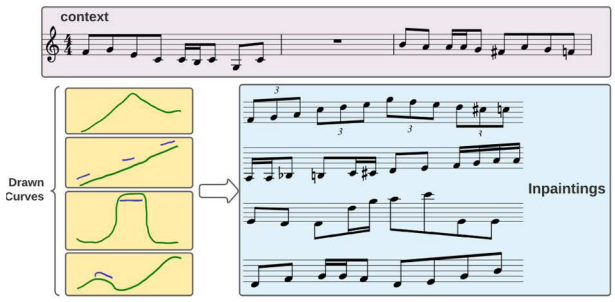

Figure 3: An example of 4 different input curves using the same context measures. The rhythm slider was set to 2 ('high') for the first, and 1 ('medium') for the last three. Note how the pitch contour and rhythm of the generated melodies follow the hand-drawn pitch curves (green) and note density curves (blue).

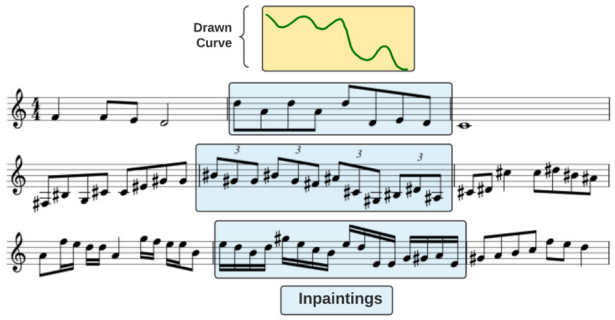

Figure 4: An example of one hand-drawn pitch curve working with three different context measures to generate different but context matching melodies. The rhythm slider was set to 0 ('low') for the first, and 2 ('high') for the last two.

3.3.1 Encoders

The VAE encoder Q consists of a CNN encoder Q_{curve} for the user drawn curves, and a GRU encoder $Q_{context}$ for the surrounding context measures. The conditional probability for the overall encoder is factorized as:

$$\begin{aligned} Q(Z \mid F_{rp}, F_{rr}, M_{prev}, M_{next}) = \\ Q_{curve}(Z_{rp}, Z_{rr} \mid F_{rp}, F_{rr}) \times \\ Q_{context}(Z_c \mid M_{prev}, M_{next}), \end{aligned} \tag{2}$$

where F_{rp} and F_{rr} denote the relative pitch and relative rhythm functions, respectively, which are derived from the users' input curves (see Section 3.4.1). Note that Q_{curve} only encodes user drawn curves but not the slider inputs. Instead, slider inputs go to the decoders directly. This pushes the latent vectors Z_{rp} and Z_{rr} to learn relative pitch and rhythm representations.

The pitch and rhythm functions are each a 512-D vector and are stacked together into a 2×512 matrix. The curve encoder Q_{curve} consists of 7 layers of residual blocks with kernel size 3, batch normalization, no pooling operations, and leaky ReLU as an activation function. The output of Q_{curve} is the concatenation of latent vectors Z_{rp} and Z_{rr}, each with dimension of 85. $Q_{context}$ consists of two GRU encoders, one for the previous measure and one for the next measure. We use an embedding layer of size 128 to encode the input of each GRU encoder, and each of them has 2 bidirectional layers with 2048 units. Their outputs are concatenated and then put through a dense layer to get the latent vector Z_c, with a dimension of 85.

3.3.2 Decoders

We design three VAE decoders based on Yang et al. (2019). Two of them are intermediate decoders, where P_{rp} generates the relative pitch output T_{rp}, and P_r generates the rhythm output T_r. The third decoder, P_{midi}, takes the previous intermediate outputs as input and generates the MIDI output (i.e., final output of the model) for the middle measure M_{cur} (see Section 3.4). The conditional probability is factorized as:

$$\begin{aligned} P(M_{cur} \mid Z, S_p, S_r) = P_{midi}(M_{cur} \mid Z_c, T_{rp}, T_r, S_p) \times \\ P_{rp}(T_{rp} \mid Z_{rcp}) \times \\ P_r(T_r \mid Z_c, Z_{rr}, S_r), \end{aligned} \tag{3}$$

where S_p and S_r denote the pitch and rhythm slider values, respectively.

We can see that Z_c is an input not only to the final decoder P_{midi}, but also to the rhythm encoder P_r. The reason is that we expect Z_c to contain some rhythmic information about the context measures, which may be useful for generating the rhythm T_r of the missing measure. This is further explained with an example in Section 3.5.2. The three VAE decoders each consist of 2 layers of unidirectional GRUs with 2048 units, and we use cross-entropy losses to train them.

Finally, we add an additional MLP decoder, $P_{chroma}(V_{chroma}|Z_c)$, to generate the chroma vector V_{chroma} which is the 12-bin binary chroma vector of measure M_{cur}, with active pitch classes taking the value of 1. We use binary cross-entropy to train this branch. This multi-task scheme helps to reduce out-of-key notes in the generated melody.

3.4 DATA REPRESENTATION

In this section, we describe the system input and output representations during inference.

3.4.1 User Drawings

As mentioned in Section 3.2, the user draws a pitch curve C_p and one or more optional note density curves $\mathbf{C}_{nd}$ (Figure 1) on a canvas. The user's 2D input is first split into connected components using the depth-first search (DFS) algorithm. The longest component is considered the pitch curve C_p, and is converted to a 512-point 1D signal recording the Y coordinates of the curve. It is then filtered with a Butterworth low-pass filter with a normalized cutoff frequency $\Omega_{3dB} = 1/50$, to give us the relative pitch function F_{rp}. The rest of the components of the drawing, if any, are treated as the note density curves, from which the relative rhythm function F_{rr} is calculated. Each note density curve is projected to the X axis to obtain a binary 512-point 1D signal, whose value is 1 within the horizontal range of the curve and

0 outside the range. All of the 1D signals of all projected note density curves are summed together to create the relative rhythm function F_{rr}, which describes note density over time. Note that since the system requires $0 \leq F_{rr} \leq 2$, values of F_{rr} exceeding 2 will be clipped.

3.4.2 Pitch and Rhythm Slider Inputs

During inference, we simply use 3-value sliders, which directly map to 0, 1 and 2 from low to high.

3.4.3 MIDI Output

Like Yang et al. (2019), for the MIDI output vector T_{midi}, we use MIDI numbers from 0 to 127 for note onsets, 128 for the holding state and 129 for rests. Instead of a regular 16th note quantization grid, we use the method from Pati et al. (2019), an uneven grid that is able to encode both 8th-note triplets and 16th notes using 24 grid positions per measure.

3.4.4 Rhythm and Relative Pitch Outputs

We need to define the two intermediate output vectors for rhythm T_r and relative pitch T_{rp}, to train the intermediate decoders P_r and P_{rp} respectively. Both relative pitch and rhythm output vectors have the same length as the final MIDI output vector, which is 24. For T_r, we follow Yang et al. (2019), and use three tokens: 0 for onsets, 1 for the holding state and 2 for rests.

The process for obtaining the T_{rp} is the following. For every pair of consecutive onsets in positions i and $j = i + n$ where $n > 0$, we calculate the MIDI interval $\Delta = T_{midi}[j] - T_{midi}[i]$ and then perform a linear interpolation between $i + 1$ and j,

$$T_{rp}[k] = (k-i)\frac{\Delta}{n},\ k \in [i+1, j]. \tag{4}$$

In other words, $T_{rp}[k]$ describes the hypothetical pitch interval between the current position k (not necessarily an onset) and the previous note onset, as if the note changed its pitch linearly. The reason for this treatment is to simulate hand-drawn pitch contours, which typically show a continuous transition instead of a step from one note to another.

As explained in Section 3.2, the more precise the T_{rp} values (pitch intervals) are, the less information is needed from the musical context c in the melody generation, and the more closely the generated melody follows the input pitch curve. However, our goal is to view the input curve as providing a prime cue for generating the melody instead of creating a precise 1–1 mapping like what "Pizzicato" (Arpege-Music, 2013) does. Therefore, we quantize the T_{rp} values to the closest quantization levels to make their representations more vague in the latent vector Z_{rp}. By controlling the level of quantization, we can control the relative influences of the input curves and the musical context to the final generation. After some experiments, we empirically choose to use 7 levels with central bins at {–9, –5, –1, 0, 1, 5, 9} in semitone units.

3.5 TRAINING

The proposed system takes user drawings and slider values as the input and generates MIDI notes as the output. To train the system, one way is to collect such input-output pairs from experienced users. This, however, is very time consuming and requires music expertise of the users. In this paper, we propose to use MIDI melodies and simulate user drawings and slider values. Specifically, we derive a relative pitch function, a relative rhythm function (Section 3.5.2) and pitch and rhythm slider values (Section 3.5.3), from the middle measure of each three-measure music excerpt. In this way, we are able to create many input-output pairs to train our system.

3.5.1 Datasets

We use the Irish Folk Music dataset (Sturm et al., 2016) to train our system. We only keep the MIDI files in 4/4 time signature that have more than 3 measures, totaling 24,065 songs and we randomly split them into 8:1:1 for training, validation, and test sets. After that, we extract all possible combinations of three consecutive measures to create the final training dataset, totalling 710,002 three-bar excerpts.

The excerpt melodies extracted from the Irish Folk Music dataset, though diverse, have a limited note range, rarely exceeding an octave. This could be a problem for the Q_{curve} encoder when users draw curves with significant fluctuations. To fix that, we also create a dataset of randomly generated three-bar melodies, within a maximum range of 2 octaves. This dataset contains much more diverse pitch fluctuations and helps Q_{curve} to be prepared for a wide range of drawn curves from users.

3.5.2 Deriving Relative Pitch and Rhythm Functions

As mentioned in the beginning of Section 3.5, we need to derive the relative pitch and relative rhythm functions for the middle bar of each training excerpt to simulate the user input. In order to extract these functions from the MIDI melodies, we use a regular time quantization grid of 48 positions per measure to represent the melodies. Note that this melody representation is different from the one we used in Section 3.4.3. For the relative pitch function F_{rp}, we first obtain a piece-wise linear curve by connecting note onsets. The Y-axis is measured in MIDI number. We then evenly sample this curve into a 512-point sequence, and low-pass filter it using a Butterworth low-pass filter with a normalized cutoff frequency $\Omega_{3dB} = 1/50$. Finally, since we find that a melody rarely (for this dataset) exceeds the range of 2 octaves within a measure, we normalize it by subtracting the middle value of its range and divide it by 12. In the rare cases (0.015% of all

measures) that the value range exceeds 2 octaves, we divide it by the half of the range instead to ensure the normalized values are in [–1,1].

To derive the relative rhythm (i.e., relative note density) function F_{rr}, it is necessary to derive two intermediate quantities. First, we define a vector *rc* of length 48 to represent the note density computed from note durations. At each note onset position *i*, it takes the value $rc_i = \log_2(1/dur_i)$, where dur_i is the duration of the current note divided by the total duration of the measure.

In our data, the smallest note duration is a 16th note, so $0 = \log_2(1/1) \leq rc_i \leq \log_2(1/(1/16)) = 4$. For positions that are not onsets, the value of the previous onset position is used. After that, we also evenly sample this vector into a 512-point sequence, to match the length of the relative pitch function F_{rp}. Second, we define a vector *qrc* which is the result of a 3-level quantization of *rc*:

$$qrc_i = \begin{cases} 0 & 0 < rc_i \leq 1.25 \\ 1 & 1.25 < rc_i \leq 3.29. \\ 2 & rc_i > 3.29 \end{cases} \quad (5)$$

Finally, we set $F_{rr} = qrc - \min(qrc)$. Subtracting the minimum value of *qrc* makes F_{rr} only contain information about the relative rhythm.

The quantization step is to make the rhythm representation more vague, as explained in Section 3.1. The discarded rhythmic information due to quantization is expected to be retrieved from the context measures through Z_c. An example of rhythmic information that Z_c can encode is whether the context measures have many triplet notes, or 16th notes. A relative rhythm function F_{rr} may indicate high or low note density, but the actual note durations that will be used depend on the musical context. In fact, this is exactly what we see happening in some of our model outputs in Figure 4, where for the same note density controls, a pitch curve produces 8th-note triplets (line 2) and 16th patterns (line 3), depending on the surrounding measures.

3.5.3 Deriving Slider Values

For the pitch slider S_p, we calculate the average pitch value of the melody, and we apply a 3-level quantization, with central bins at {65, 71, 77}, in the unit of MIDI numbers. These quantization values correspond to the *low*, *middle* and *high* areas of a treble stave. As for the rhythm slider S_r, we set it to $S_r = \min(qrc)$, where *qrc* is defined in Section 3.5.2.

3.5.4 Training Configuration

A value of 0.1 was used to weight the KL loss, as proposed in β-VAE (Higgins et al., 2016). For the stochastic gradient descent, the Adam algorithm was used (Kingma and Ba, 2015), with a linear learning rate schedule from 0.0003 to 0.00001. We trained for 200 epochs with batch size 128, and used teacher forcing to train the RNN decoders P_{midi}, P_{rp} and P_r, with probability 1.0, which was gradually reduced until 0.5.

As for training using the dataset of randomly generated melodies, since the notes are random, there is neither correlation among the three bars nor useful musical information within each measure. Therefore, for such training data, we only use the intermediate relative pitch cross-entropy loss but not the other musically meaningful losses.

During training, we sample these randomly generated excerpts with probability 0.1 and the Irish excerpts with probability 0.9. We also apply a random shift with probability 0.1 to each note in the melodies along both time and pitch. This is to compensate for the variability of hand drawn curves of the users during inference. The shifts along the time axis were no more than an eighth note, and along the note axis were no more than 2 semitones. We should mention that we apply this random shift only when deriving the relative pitch and relative rhythm functions, without applying any shifting on the training targets.

4. EXPERIMENTS

4.1 BASELINES

In this section, we conduct objective and subjective evaluations to assess the feasibility and usability of the proposed system for drawing-based melody inpainting. We compare it with two closely related methods as baselines under the same user interface. The comparisons focus on algorithm design instead of the interface design, as the former is the main contribution of this work.

4.1.1 Rule-Based Baseline (RB)

We develop a rule-based system as a baseline, following the gist of "Pizzicato" (Arpege-Music, 2013). Many necessary modifications have to be introduced, since in the task described in "Pizzicato" (see Section 2), the generation of the music measure does not depend on the surrounding measures, and the user is required to provide the exact rhythm and chord labels for every position.

The RB baseline generates the rhythm of the melody in the following way. Note onsets are placed at local peaks and valleys of the relative pitch function F_{rp}. If the function does not have any peaks and valleys, then we fill the measure with notes whose duration depends on the value of the rhythm slider S_r (quarter notes for S_r = 0, eighth notes for S_r = 1 and sixteenth notes for S_r = 2). If the user draws extra note density curves $\mathbf{C}_{nd}$, we add extra onsets in the area under the curves fitting in the 16th note grid. Given the variability and the inaccuracy of a user's input, we quantize the position of the note density curves and the indices of the peaks of the pitch curve to the nearest sixteenth note.

To decide the pitch at each onset position, we first find the note n that corresponds to the value of the pitch curve in the onset position. Then, we replace n with the closest note (in terms of MIDI numbers) from an appropriate set of notes.

This set consists of all the notes that belong to the key implied by the two surrounding measures. We estimate the key using the Krumhansl key detection algorithm (Krumhansl and Kessler, 1982) implemented in the music21 python library (Cuthbert and Ariza, 2010). We consider this a strong baseline, because the generated melody always matches the input curves, and notes always belong to the estimated key.

4.1.2 Genetic Algorithm Baseline (GA)

We also develop a baseline based on the work of the "JamSketch" system (Kitahara et al., 2018; Yasuhara et al., 2019). Again, we introduce necessary modifications to make it comparable with our system. The original "JamSketch" does not support 16th notes, and the note selection does not depend on the previous measures but on the predetermined chord progression for this measure. Finally, the inputs of our system include additional note density curves and extra pitch and rhythm sliders, which do not exist in "JamSketch". We implement these modifications by adding new terms in their genetic algorithms' fitness functions.

Following their work, we first use a genetic algorithm to generate the rhythm in a form of a binary vector (1 for onsets and 0 for holds), and then use another genetic algorithm to generate the pitches for the onset positions indicated by the rhythm vector.

For the following analysis, we will use the letters *u,c,s* as superscripts to indicate vectors related to the user input, a chromosome and the solution of a genetic algorithm.

Rhythm Generation

As mentioned earlier, the rhythm chromosomes r^c are binary vectors of size $L = 24$, where L is the number of grid positions in a bar as described in Section 3.4.3. We define F_{rr}^u as the relative rhythm function extracted from the user's input and F_{rr}^c as the one extracted from a chromosome. Additionally, we define $n^c = \text{sum}(r^c)$ as the number of notes/onsets in a chromosome and d^c as a vector of size n^c containing the duration of each note of the rhythm chromosome.

For generating rhythm, we use a combination of global and local features to design the fitness function

$$f(R) = w_0 \times sim(R) + w_1 \times lik(R) + w_2 \times seq1(R) + w_3 \times slid(R) \quad (6)$$

where $w_0 = 6$, $w_1 = 1$, $w_2 = 1$ and $w_3 = 2$ are weights for the four terms, and their values are empirically chosen with dozens of trials to optimize the performance:

- $sim(R)$: The similarity of a chromosome's rhythm with the desired rhythm of the user

$$sim(R) = -\sum_{i=0}^{L-1} (F_{rr}^u[i] - F_{rr}^c[i])^2. \quad (7)$$

- $lik(R)$: The likelihood of a chromosome, which is the same as that of Yasuhara et al. (2019).

$$lik(R) = -\sum_{i=0}^{L-1} P(r_i^c \mid i), \quad (8)$$

where $P(r_i^c|i)$ is the probability of having an onset at position i, and it is calculated from the dataset.

- $seq1(R)$: The duration bi-grams. This term is not present in Yasuhara et al. (2019) and it significantly improves the generated rhythm.

$$seq1(R) = -\sum_{i=1}^{n^c-1} P(d_i^c \mid d_{i-1}^c), \quad (9)$$

where $P(d_i^c \mid d_{i-1}^c)$ is the conditional probability of a note with duration d_{i-1}^c being followed by one with d_i^c, and it is calculated from the dataset.

- $slid(R)$: The global note density. This terms takes into account the rhythm slider input.

$$slid(R) = -|\, den^c - S_r \,|, \quad (10)$$

where den^c measures the note density of the current chromosome

$$den^c = \begin{cases} 0, & \text{for } 0 < n^c < 6 \\ 1, & \text{for } 6 \le n^c < 12, \\ 2, & \text{for } n^c \ge 12 \end{cases} \quad (11)$$

and S_r is the slider input taking three values 0,1,2.

After this genetic algorithm runs, we pick the rhythm solution r^s that has the largest fitness score as the chromosome r^c in the final generation.

Pitch Generation

Given the rhythm solution r^s from the previous rhythm generation step, we use another genetic algorithm to generate the pitch solution vector p^s, which has size n^s. The pitch chromosome p^c consists of pitch genes that can take MIDI values from 48 to 84. From p^c we can derive the chromatic pitch class vector of the chromosome using $cpc^c = p^c \bmod 12$. We also define F_{rp}^u as the relative pitch function extracted from the user's input and F_{rp}^c as the one extracted from a pitch chromosome. Finally, we define t^s as a vector of size n^s containing the position of each note of the rhythm solution.

The fitness function for pitch generation is

$$f(P) = w_0 \times sim(P) + w_1 \times seq1(P) + w_2 \times seq2(P) + w_3 \times harm(P) + w_4 \times slid(P), \quad (12)$$

where the weights $w_0 = 3$ and $w_1 = w_2 = w_3 = w_4 = 1$ were manually set for the five terms:

- $sim(P)$: The similarity of the current chromosome's pitch curve with the desired pitch curve of the user

$$sim(P) = -\sum_{i=0}^{n^s-1}(F_{rp}^u[t_i^s] - F_{rp}^c[t_i^s])^2. \quad (13)$$

- $seq1(P)$: The chromatic pitch class bi-grams

$$seq1(P) = -\sum_{i=1}^{n^s-1} P(cpc_i^c \mid cpc_{i-1}^c), \quad (14)$$

where $P(cpc_i^c \mid cpc_{i-1}^c)$ is the conditional probability of a note with chromatic pitch class cpc_{i-1} being followed by one with cpc_i.

- $seq2(P)$: The chromatic pitch class delta bi-grams

$$seq2(P) = -\sum_{i=1}^{n^s-1} P(cpc_i^c - cpc_{i-1}^c \mid cpc_{i-1}^c - cpc_{i-2}^c). \quad (15)$$

- $harm(P)$: The fitness of the chromosome's notes with the given context

$$harm(P) = -(1 - \frac{\sum_{x \in N} \mathbf{1}_K(x)}{|N|}), \quad (16)$$

where K is the set of the chromatic pitch classes that belong to the key implied by the surrounding measures, and N is the multiset that contains the chromatic pitch classes of the current chromosome, and

$$\mathbf{1}_K(x) := \begin{cases} 1 & \text{if } x \in K, \\ 0 & \text{if } x \notin K. \end{cases} \quad (17)$$

- $slid(P)$: The global pitch offset

$$slid(P) = -|po^c - S_p|, \quad (18)$$

where po^s measures the pitch offset of the current solution

$$po^c = \begin{cases} 0, & \text{for } n^s < 65 \\ 1, & \text{for } 65 \le n^s < 71, \\ 2, & \text{for } n^s \ge 71 \end{cases} \quad (19)$$

and S_p is the pitch slider input taking three possible values 0,1,2.

In the original "JamSketch" implementation, they added an additional *pitch entropy* factor to discourage the generation of melodies with many repetitive notes. Another trick is that they created a prefix tree from the dataset melodies, and they used it to randomly generate the initial pitch chromosomes for the genetic algorithm. We omit both of those tricks since they did not seem to affect the result.

4.2 OBJECTIVE EVALUATION ON SIMULATED INPUTS

We attempt an objective evaluation of the model's performance using simulated data. That is, we simulate users' inputs to the inpainting task using relative pitch and rhythm functions extracted from a randomly selected measure from the test dataset. Specifically, the simulation process goes as this: first, a random 3-measure excerpt [M_{prev}, M_{cur}, M_{next}] is selected from the Irish test dataset. The first and third measures serve as the context for the inpainting task while the middle measure is discarded.

Next, we randomly choose another measure $M_{cu}^{'}$, and extract the relative pitch and rhythm functions $F_{rp}^{'}$, $F_{rr}^{'}$ and slider values $S_p^{'}$ and $S_r^{'}$ as described in Section 3.5, as if they were derived from a user's input. Feeding $F_{rp}^{'}$, $F_{rr}^{'}$, $S_p^{'}$, $S_r^{'}$ and the context measures to each inpainting model, we generate a new measure $M_{cur}^{''}$, from which we calculate the functions $F_{rp}^{''}$ and $F_{rr}^{''}$ and slider values $S_p^{''}$ and $S_r^{''}$.

Ideally, the generated measure should fit to the context, and its extracted curves and slider values should be similar to those of the simulated input. We therefore design three tasks to measure the pitch contour similarity, rhythm similarity and context fitness, respectively. For each task we run the above simulation process 1000 times. The results are summarized in Table 1. Descriptions of these three tasks follow.

METRIC	PROPOSED	RULE-BASED	GA	M_{CUR} ("MISSING" MEASURE)	M_{CUR}' (FROM WHICH USER INPUT IS DERIVED)
Pitch curve DTW cost	6.1 ± 5.2	**3.6 ± 2.4**	13.3 ± 10.6	54.6 ± 38.3	–
Pitch slider match rate	98%	**100%**	93%	–	–
Rhythm curve DTW cost	**8.4 ± 13.1**	23.1 ± 41.1	41.0 ± 52.5	53.8 ± 61.3	–
Rhythm slider match rate	**97%**	92%	90%	–	–
Context match	86%	100%	80%	87%	45%

Table 1: Objective evaluation of measures generated by the three comparison methods (proposed, rule-based, and GA) taking a simulated user input that is derived from a random measure. As controls, evaluation of the original "missing" measure M_{cur} and the measure from which the user input is derived $M_{cur}^{'}$ is also provided. Please refer to the main text for explanations of the metrics.

4.2.1 Pitch Contour Match

For this task, we make the observation that the rhythm of the generated melody actually affects the melody pitch contour. Taking the example of the GA method, if the genetic algorithm responsible for rhythm generation fails to generate enough note onsets, then the melody pitch contour will not match the user's input pitch curve. To isolate the rhythm effects, we manually interrupt the normal flow of generation to enforce the ground-truth rhythm. We use a dense vector that consists of 16 sixteenth note positions to represent the ground-truth rhythm.

A dense rhythm vector, compared to a sparse vector that contains fewer note positions, allows the final note generation stage to realize the pitch curve as faithfully as possible. It is noted that some of the 16 note positions will be onsets while the others will be holds, depending on the ground-truth rhythm. Enforcing a specific rhythm is easy since for all three methods the generation process is sequential; first the rhythm and then the pitch. For the baseline methods RB and GA, we omit the rhythm generation part and replace the binary rhythm vector with the desired one, while for our proposed model, we replace the output logits of the rhythm decoder P_r with those that correspond to our desired rhythm.

We then use dynamic time warping (DTW) to globally align the relative pitch function extracted from the generated measure $F_{rp}^{''}$ and the one extracted from the randomly chosen measure $F_{rp}^{'}$, and use the alignment cost to evaluate the match between the simulated input's and the generated melody's relative pitch function for each method. The mean and standard deviation of the alignment cost over the 1000 runs is 6.1±5.2 for the proposed model, 3.6±2.4 for RB, and 13.3±10.6 for GA. For reference, we also extract the ground-truth relative pitch function F_{rp} from the discarded middle measure of the 3-measure excerpt where the context measures come from, and calculate the average alignment cost between F_{rp} and $F_{rp}^{''}$. This reference can be viewed as the average alignment cost between two random pitch curves in the dataset. As they are unrelated, the alignment cost is as high as 54.6±38.3. Furthermore we measure the percentage of having $S_p^{'} = S_r^{''}$ (i.e., matching pitch offsets between the randomly chosen and the generated measure), which is 98% for the proposed model, 100% for RB and 93% for GA. As expected, the RB method achieves the best results, as it is designed to directly maximize the relative pitch contour match. Additionally, both our method and the GA baseline significantly improve the relative pitch contour matching compared to the reference.

4.2.2 Rhythm Match

For this task, we let the systems generate the rhythm uninterrupted, and the same as before, we use DTW to globally align the relative rhythm function extracted from the generated measure $F_{rr}^{''}$ and the one extracted from the randomly chosen measure $F_{rr}^{'}$, and use the alignment cost to evaluate the rhythm match. The mean and standard deviation of the alignment cost over the 1000 runs is 8.4±13.1 for the proposed model, 23.1±41.1 for RB, and 41.0±52.5 for GA. Again for reference, we extract the ground-truth relative rhythm function F_{rr} from the discarded middle measure of the 3-measure excerpt where the context measures come from, and we calculate the average alignment cost between $F_{rr}^{''}$ and F_{rr}, and the value is 53.8±61.3. Furthermore, we measure the percentage of having $S_r^{'} = S_r^{''}$ (i.e., matching rhythm offsets between the randomly chosen and the generated measure), which is 97% for the proposed model, 92% for RB and 90% for GA.

4.2.3 Context Match

In an attempt to evaluate the match of the generated melody with its surrounding measures, we calculate the percentage of notes (chromatic pitch classes) in the generated measure $M_{cur}^{''}$ that also belong to the key (using the Krumhansl algorithm) of the original measures M_{prev}, M_{cur}, M_{next}. The average percentage was 86% for the proposed model, 80% for the GA baseline, and 100%, which is by design, for the RB baseline. It is noted that this metric is not an accurate descriptor of the musicality of the result but just a rough indicator, as 1) the key can be ambiguous when only looking at the three measures, and 2) even if the key is clear, passing notes not belonging to the key can still produce musically pleasant results. Nevertheless, we still think that this metric provides a basic safeguard of ensuring the basic musicality. For reference, the value of this metric for the 3-measure excerpts in the dataset is 87%, while if we replace the middle measure with a random one from the dataset, it goes to 45%. These results suggest that, even though our proposed method achieves the closest score to the one calculated from the original excerpts, for all methods, the generated melody fits well to the the musical context, at least in terms of pitch selection.

4.3 USER STUDIES

We perform subjective evaluation of the proposed system and the baseline systems, to answer two research questions: 1) Is drawing curves and setting sliders an intuitive way to generate melodies for users with diverse musical backgrounds? 2) How is the quality of the music inpainting results of the proposed system compared to that of the baselines? This study is approved by the University of Rochester's Institutional Review Board.

4.3.1 User Recruitment

We recruited 23 participants (11 female, 12 male) with various musical backgrounds. Their participation was voluntary (with oral consent) and without money incentives. We asked them to self-assess their musical

background with a 4-way choice question. Specifically, 4 were professional musicians; 4 had been playing an instrument for more than two years, but not professionally; 6 were not familiar with Western music notation, but had some general understanding of music fundamentals, and the last 9 did not belong to any of the above categories. We regard the latter two groups as not having received formal musical training, or more generally, not having a musical background.

Although our system was primarily designed for non-musicians, we included amateur and professional musicians as we believe their opinions are important for assessing the musicality of the results.

4.3.2 User Interface and System Configuration

For this experiment, we developed a simple web-based interface, shown in Figure 5, according to the task described in Section 3.2. It consists of a canvas area for users to draw the curves with their mouse, and three identical display areas for showing the context measures and the generated results for each method. The assignment of methods to the display areas is randomized for every trial to reduce biases toward a specific method. There are buttons that affect all three methods: the "Get new context" button randomly chooses context measures from the test dataset and displays them in all display areas. The "Clear Curve" button clears the canvas area. Additionally, each display area has its own pitch and rhythm sliders to adjust the relative levels, a "Generate" button to generate the melody, and a "Play" button to play the result. The right side of each display area is the evaluation sub-area, where users are asked to rate the pitch contour match, note density match and the overall musicality of the result. At the bottom of the page is the "Submit" button, which submits all the input, output, and evaluation data after each trial for later analysis. Finally, instructions of this task are shown at the bottom left area of the interface throughout the experiment.

One problem that occurred in the design of this study was the uneven execution time of these methods. For RB it was about 2 seconds and for the proposed method it was about 2.5 seconds, both measured on an Intel i7 CPU. For GA, the nature of this method allows us to determine the number of generations and limit the execution time to our will. We set the execution time limit to 3 seconds, which seemed sufficient for GA to reach a reasonable performance without incurring too much time cost on users. Even though we did not conduct any rigorous evaluation, letting the GA method run for more than 3 seconds did not seem to improve the results. We then added a 1 second delay and a 0.5 second delay to RB and the proposed methods respectively to prevent users from identifying these methods by their execution time.

4.3.3 Task

To start a user's task, we dedicated 10-15 minutes to explain what they had to do, answer their questions, and run one or two training trials. We found this step necessary especially for participants with no musical background. After that, each user had to use the interface for at least 3 trials. Each trial starts by fetching new context measures from the test dataset, drawing the curves, adjusting the

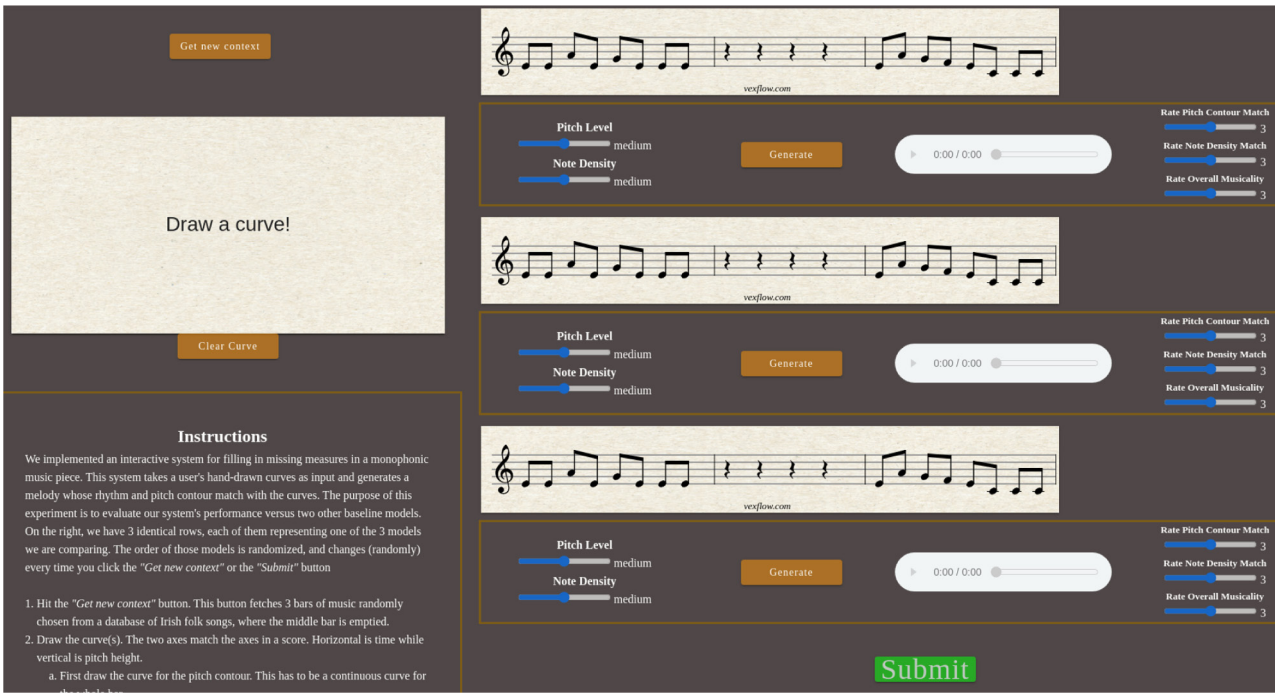

Figure 5: The web application developed for user studies. Detailed instructions are shown on the bottom left of the page throughout the duration of the experiment. On the top left is the canvas where users draw the curves. On the right side, three identical rows of context measures, generated results and control units are displayed corresponding to the three comparison methods (with a random order for each trial).

sliders and generating a new melody for each method. After listening to each result, they had to rate the pitch contour match, the note density match and the overall musicality of the inpainting result for each method using the three evaluation sliders mentioned above. Each slider uses a five-point scale from 1 to 5, with 5 indicating the "best". We refer to these questions with labels Q1-Q3. The participants were also encouraged to think-aloud while performing the task, in order to help us get some more insights.

In addition to the three evaluation questions for each method in each trial, after finishing all trials, they were asked three questions to rate their overall user experience throughout all trials and all three methods using a scale from 1 to 5, with 5 being "Very Agree". We refer to these questions with labels Q4-Q6.

Q1 Rate the similarity between the hand-drawn pitch curve and the generated pitch contour.
Q2 Rate the effect of the note density inputs (rhythm slider and note density curves) to the melody's rhythm.
Q3 Rate the overall musicality of the final result.
Q4 Do you think this system provided an intuitive way to generate a melody ?
Q5 Do you think drawing a pitch curve is an intuitive way to control a melody's contour?
Q6 Do you think drawing additional note density curves is an intuitive way to control a melody's rhythm?

4.3.4 Results

Figure 6 shows the evaluation results for the six questions. For Q1-Q3, there are 112 data points each, as each user completed 4.8 trials on average. For questions Q4-Q6, 23 data points are available, as there was one response from each user.

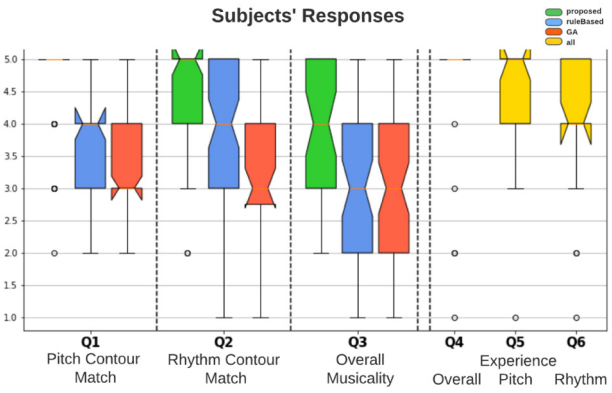

Figure 6: Boxplots of the answers (ratings on a 1–5 scale) to the six subjective evaluation questions from all of the 23 participants (the higher the better). Each box for Q1–Q3 contains 112 points, while for Q4–Q6 contains 23 points. The notch in each plot represents the 95% confidence interval around the median. Outliers are shown as circles.

For each of the questions Q1-Q3, we run the Kruskal-Wallis H-test (a non-parametric version of ANOVA), to determine whether there is any statistical difference in the performance of the methods. The p-values obtained for all three questions are less than 10^{-5}, suggesting the existence of statistical differences. We then run pairwise Mann Whitney U tests for each question.

The results of Q1 (pitch contour match) support the objective evaluation findings in Section 4.2.1, that is, the melodic contour of the generated melody is closely related to the input pitch curve for all the methods, with median values above or equal to 3. In addition, our proposed method achieves significantly better ratings compared to the baselines ($p < 10^{-19}$ vs RB and $p = 10^{-21}$ vs GA). One difference from the objective evaluation in Section 4.2.1 is that the RB baseline achieves a lower score than our proposed method. This happens because in the objective evaluation we used the predefined "dense" rhythm vectors (see Section 4.2.1), while on this experiment the rhythm vector is generated according to the user's input. A "sparse" rhythm vector does not contain enough notes to realize the user's pitch curve. The RB outperforms the GA in terms of pitch contour matching ($p < 0.005$).

For Q2 (note density match), the proposed method also receives significantly better ($p < 10^{-5}$ vs RB and $p < 10^{-19}$ vs GA) ratings, showing its superior ability of following the note density input controls (note density curves and rhythm slider). Comparing the two baselines, RB achieves better results ($p < 10^{-5}$). These results are in agreement with the ones obtained from our objective evaluation (third and fourth line of Table 1).

For Q3, we can see that the proposed method generates melodies that are significantly more musical ($p = 10^{-14}$ vs both). Additionally, the users thought that GA results were similar in musicality to those of RB ($p > 0.4$). One reason for the significantly lower musicality score of the baselines, we believe, is due to the fact that the baselines are prone to generating uncommon rhythmic patterns (both of them) and out of context pitches (GA), as also shown in Section 4.2. For RB, some of the note positions are determined by the peak and valley positions of the pitch curve; this sometimes results in off-beat notes. In contrast, for the proposed method, the shift invariance of its CNN encoder Q_{curve} and the data augmentation tricks introduced in Section 3.5.1 make it invariant to this type of misalignment. As for the GA, this method cannot sufficiently explore its large search spaces given the time constraints (i.e., 3 seconds in our experiments); compared to the task of Yasuhara et al. (2019), our search space is much larger.

In Figure 7 we provide two representative examples obtained during the user studies. On the left example, the user gave a much higher musicality rating to the VAE-generated example compared with the baseline ones, even though the output of all 3 methods received high

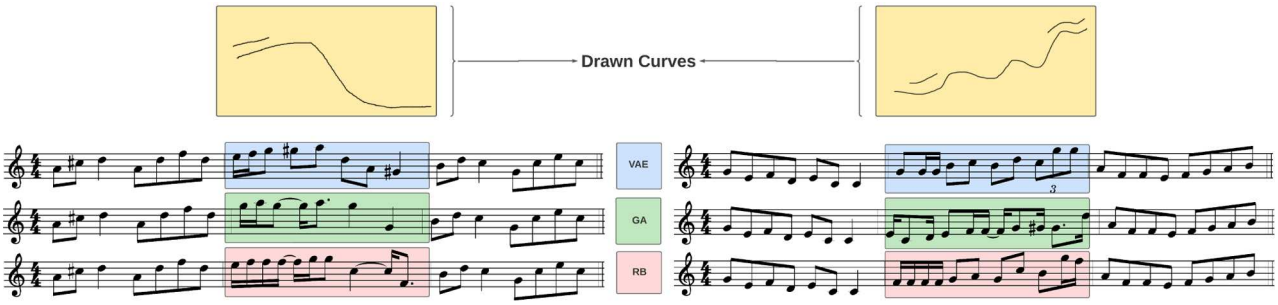

Figure 7: Two examples obtained during the user studies. The first line is the output of the proposed VAE model and the next two contain the baselines' ouputs.

pitch and rhythm match scores. The VAE output used more common rhythmic patterns than the baselines and generated a chromatic passing note that leads to a more surprising and musical melody.

The results for Q4–Q6 clearly show that this "Draw and Listen" interaction was very intuitive for most of the users. We also present some insights we gathered due to the think-aloud policy, and some discussions we had with the users. Out of the three users who rated questions Q4–Q6 less than 3, all of them were musicians (one professional and two amateurs). The professional and one of the amateurs expressed their preferences for the traditional way of melody composition, but they also believed that this new type of interaction can be useful for non-musicians. Additionally, some of the not-so-advanced users who had some basic experience with music production tools, had many ideas on how to incorporate a system like this in their preferred digital audio workstations (DAWs) to edit and generate MIDI music.

5. LIMITATIONS

While the proposed model outperforms the two baselines on all the questions, there is also much room for improvement. For example, there may be a tradeoff between faithfully following the user's input and musicality; it would be useful to have users control this tradeoff through a parameter.

Another limitation is that the model only supports inpainting of a single measure given its immediate neighbors. Supporting a variable number of context measures is not the main focus of this work, but it can be realized by extending our current architecture or combining it with models that already support this feature (e.g., Chen et al., 2020).

Furthermore, in areas of high note density (indicated by the user using note density curves or the slider), the proposed model tends to generate mostly 8th-triplet notes instead of 16th notes (first line in Figure 3 or second line in Figure 4). It also never generates rests. These limitations are due to the data bias in the Irish dataset used for training, which contains 3 times more 8th-note triplets than 16th notes and less than 0.1% of the tokens are rests.

Finally, regarding the user interface, one feature that most of the users asked, is the ability to click and drag parts of the curve to make minor adjustments. Some users also expressed the desire to be able to explicitly indicate areas for the model to generate rests.

6. APPLICATIONS

As a music inpainting tool, the proposed system could be used as a composition plugin for music notation editors where users could use the "pencil" tool to generate melodies on the spot, conditioned on surrounding measures. Additionally, since our system uses MIDI to represent notes, it can work in applications that are not based on Western music notation to lower the user barrier. For example, it can be implemented as a plugin for digital audio workstations (DAWs), which typically use MIDI instead of music notation to represent the music content. Additionally, the melody factorization we just described can be used to support a number of diverse applications. We can replace the hand-drawn curves with anything that resembles a curve, such as a stock price curve or a human body movement curve, to support creative use cases.

In addition, we can use it to measure melodic similarity and query a melody database by converting melodies in the latent space and calculating their latent vector distances based on any of the disentangled factors (relative pitch and rhythm).

7. CONCLUSIONS

In this work we proposed a new melody disentanglement scheme that decomposes a melody to three basic aspects: a) the relative pitch, b) the relative rhythm, and c) music theory-related information (e.g., key, pitch distribution). Based on this factorization, we developed a VAE-based system for interactive musical inpainting. This system allows users to draw pitch and optional note density curves to guide the pitch contour and rhythm of

the generated melodies. We ran objective experiments on simulated data and designed a web application to conduct subjective experiments with 23 participants. Results showed that this novel interaction is intuitive and effective, and that our model outperforms two baselines based on previous work. In the future we plan to address some of the limitations of the current system and also explore more possible applications of the proposed melody disentanglement in symbolic music generation and analysis.

NOTE

1 https://github.com/xribene/DrawAndListen.

ACKNOWLEDGEMENTS

This work is supported by the National Science Foundation grant No. 1846184.

COMPETING INTERESTS

The authors have no competing interests to declare.

AUTHOR AFFILIATIONS

Christodoulos Benetatos orcid.org/0000-0002-4008-3133
Department of Electrical and Computer Engineering, University of Rochester, NY, US

Zhiyao Duan orcid.org/0000-0002-8334-9974
Department of Electrical and Computer Engineering, University of Rochester, NY, US

REFERENCES

Adler, A., Emiya, V., Jafari, M. G., Elad, M., Gribonval, R., and **Plumbley, M. D.** (2012). Audio inpainting. *IEEE Transactions on Audio, Speech and Language Processing*, 20(3):922–932. DOI: https://doi.org/10.1109/TASL.2011.2168211

Arpege-Music (2013). Pizzicato notation software. http://www.arpegemusic.com/manual36/EN855.htm. Online; accessed 9 December 2021.

Benetatos, C., VanderStel, J., and **Duan, Z.** (2020). BachDuet: A deep learning system for humanmachine counterpoint improvisation. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 635–640.

Berg, T., Chattopadhyay, D., Schedel, M., and **Vallier, T.** (2012). Interactive music: Human motion initiated music generation using skeletal tracking by Kinect. In *Proceedings of the Conference of the Society for Electro-Acoustic Music in the United States*.

Chen, K., Wang, C.-i., Berg-Kirkpatrick, T., and **Dubnov, S.** (2020). Music sketchnet: Controllable music generation via factorized representations of pitch and rhythm. In *Proceedings of the 21st International Society for Music Information Retrieval Conference*, pages 77–84. ISMIR.

Coduys, T. and **Ferry, G.** (2004). Iannix aesthetical/symbolic visualisations for hypermedia composition. In *Journees d'informatique musicale*.

Cuthbert, M. S. and **Ariza, C.** (2010). Music21: A toolkit for computer-aided musicology and symbolic music data. In Downie, J. S. and Veltkamp, R. C., editors, *Proceedings of the International Society for Music Information Retrieval Conference*, pages 637–642.

Dannenberg, R. B. and **Raphael, C.** (2006). Music score alignment and computer accompaniment. *Communications of the ACM*, 49(8):38–43. DOI: https://doi.org/10.1145/1145287.1145311

Donahue, C., Simon, I., and **Dieleman, S.** (2019). Piano Genie. In *Proceedings of the 24th International Conference on Intelligent User Interfaces*, pages 160–164, New York, NY, USA. Association for Computing Machinery. DOI: https://doi.org/10.1145/3301275.3302288

Dowling, W. J., Barbey, A., and **Adams, L.** (1999). Melodic and rhythmic contour in perception and memory. In Yi, S., editor, *Music, Mind, and Science*, pages 166–188. Seoul National University Press.

Farbood, M. M., Pasztor, E., and **Jennings, K.** (2004). Hyperscore: A graphical sketchpad for novice composers. *IEEE Computer Graphics and Applications*, 24(1):50–54. DOI: https://doi.org/10.1109/MCG.2004.1255809

Greshler, G., Shaham, T. R., and **Michaeli, T.** (2021). Catch-A-Waveform: Learning to generate audio from a single short example. *arXiv preprint arXiv:2106.06426*.

Hadjeres, G. and **Nielsen, F.** (2020). Anticipation-RNN: Enforcing unary constraints in sequence generation, with application to interactive music generation. *Neural Computing and Applications*, 32(4):995–1005. DOI: https://doi.org/10.1007/s00521-018-3868-4

Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S., and **Lerchner, A.** (2016). beta-VAE: Learning basic visual concepts with a constrained variational framework. In *5th International Conference on Learning Representations*.

Huang, A., Hawthorne, C., Roberts, A., Dinculescu, M., Wexler, J., Hong, L., and **Howcroft, J.** (2019). Bach Doodle: Approachable music composition with machine learning at scale. In *Proceedings of the 20th International Society for Music Information Retrieval Conference (ISMIR)*.

Kingma, D. and **Ba, J.** (2015). Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Kingma, D. P. and **Welling, M.** (2014). Auto-encoding variational Bayes. In *Proceedings of the 2nd International Conference on Learning Representations*.

Kitahara, T., Giraldo, S., and **Ramirez, R.** (2018). JamSketch: Improvisation support system with GA-based melody

creation from user's drawing. In Aramaki, M., Davies, M. E. P., Kronland-Martinet, R., and Ystad, S., editors, *Music Technology with Swing*, pages 509–521. Springer International Publishing. DOI: https://doi.org/10.1007/978-3-030-01692-0_34

Krumhansl, C. L. and **Kessler, E. J.** (1982). Tracing the dynamic changes in perceived tonal organization in a spatial representation of musical keys. *Psychological Review*, 89(4):334. DOI: https://doi.org/10.1037/0033-295X.89.4.334

Lewis, G. E. (2000). Too many notes: Computers, complexity and culture in Voyager. *Leonardo Music Journal*, pages 33–39. DOI: https://doi.org/10.1162/096112100570585

Mao, H. H., Shin, T., and **Cottrell, G.** (2018). DeepJ: Style-specific music generation. In *2018 IEEE 12th International Conference on Semantic Computing (ICSC)*, pages 377–382. IEEE. DOI: https://doi.org/10.1109/ICSC.2018.00077

Marafioti, A., Majdak, P., Holighaus, N., and **Perraudin, N.** (2020). GACELA: A generative adversarial context encoder for long audio inpainting of music. *IEEE Journal of Selected Topics in Signal Processing*, 15(1):120–131. DOI: https://doi.org/10.1109/JSTSP.2020.3037506

Pati, A., Lerch, A., and **Hadjeres, G.** (2019). Learning to traverse latent spaces for musical score inpainting. In *Proceedings of the 20th International Society for Music Information Retrieval Conference*, pages 343–351. ISMIR.

Sturm, B. L., Santos, J. F., Ben-Tal, O., and **Korshunova, I.** (2016). Music transcription modelling and composition using deep learning. *Conference on Computer Simulation of Musical Creativity*.

Thiebaut, J.-B., Healey, P. G., and **Bryan-Kinns, N.** (2008). Drawing electroacoustic music. In *International Computer Music Conference*.

U&I-Software (1997). Metasynth + Xx. https://uisoftware.com/metasynth/. Online; accessed 9 December 2021.

Wuerkaixi, A., Benetatos, C., Duan, Z., and **Zhang, C.** (2021). Collagenet: Fusing arbitrary melody and accompaniment into a coherent song. In *Proceedings of the 22nd International Society for Music Information Retrieval Conference*.

Xenakis, I. (1977). Upic. https://en.wikipedia.org/wiki/UPIC. Online; accessed 9 December 2021.

Yang, R., Wang, D., Wang, Z., Chen, T., Jiang, J., and **Xia, G.** (2019). Deep music analogy via latent representation disentanglement. In *Proceedings of the 20th International Society for Music Information Retrieval Conference*, pages 596–603. ISMIR.

Yasuhara, A., Fujii, J., and **Kitahara, T.** (2019). Extending JamSketch: An improvisation support system. In *16th Sound and Music Computing Conference*, pages 289–290.

TO CITE THIS ARTICLE:
Benetatos, C., and Duan, Z. (2022). Draw and Listen! A Sketch-Based System for Music Inpainting. *Transactions of the International Society for Music Information Retrieval*, 5(1), 141–155. DOI: https://doi.org/10.5334/tismir.128

Submitted: 22 December 2022 **Accepted:** 04 August 2022 **Published:** 02 November 2022