AN EXPLORATION OF GENERATING SHEET MUSIC IMAGES

Marcos Acosta
Harvey Mudd College
mdacosta@g.hmc.edu

Irmak Bukey
Pomona College
ibab2018@mymail.pomona.edu

TJ Tsai
Harvey Mudd College
ttsai@g.hmc.edu

ABSTRACT

Many previous works in recent years have explored various forms of music generation. These works have focused on generating either raw audio waveforms or symbolic music. In this work, we explore the feasibility of generating sheet music images, which is often the primary form in which musical compositions are notated for other musicians. Using the PrIMuS dataset as a testbed, we explore five different sequence-based approaches for generating lines of sheet music: generating sequences of (a) pixel columns, (b) image patches, (c) visual word tokens, (d) semantic tokens, and (e) XML-based tags. We show sample generated images, discuss the practical challenges and problems with each approach, and give our recommendation on the most promising paths to explore in the future.

1. INTRODUCTION

This work explores the feasibility of generating music in the form of sheet music images. Below, we provide a brief overview of recent work in the area of music generation, motivate the problem of generating sheet music images, and describe five sequence-based approaches that we will explore in this study. Our main goal is to identify the practical challenges and problems with each of the five approaches and to provide a recommendation on the most promising paths to explore in the future.

Music generation has been an active research topic of interest to the MIR community in recent years. Many recent works in this area fall into one or more of the following three research thrusts. The first research thrust is controllable music generation, in which a user can control certain aspects of the music generation process. Some examples of this include providing a template from which variations can be generated [1] [2], providing a video to which a suitable background music track is generated [3], specifying lyrics for the generated song [4], or disentangling different characteristics of music like style & melody [5], pitch & rhythm [6] [7], or structure & content [8]. A second research thrust is generating music with realistic short-term and long-term structure. Most recent works achieve this by representing music as a sequence of discrete tokens and using Transformer-based architectures to learn long-term dependencies. Some prominent examples of this include OpenAI's Jukebox model [4] and Google Magenta's Music Transformer model [9] [10]. Other works focus on encoding music in a way that allows the generated music to exhibit appropriate metrical structure at the beat, bar, and phrase levels [11]. A third research thrust is to generate music in different contexts and musical representations. Some examples include multi-part music [12], multi-track music [13] [14] [15] [16], guitar tabs [17], and jazz lead sheets [18]. Much of this work focuses on how to encode different music representations or different aspects of music (like pitch, duration, velocity, and timing [11]) in a form that is suitable for training with a Transformer model.

This work falls into the last group and explores music generation in a different musical representation: sheet music images. To the best of our knowledge, this is the first systematic study of generating sheet music images.

Why generate music in the form of sheet music images? We present four reasons. First, sheet music is the predominant medium by which compositions are shared in many genres of music. If a piece of music is intended to be a composition (as opposed to a demo for an academic research paper), the expectation is that the composition will be in the form of sheet music. Second, sheet music contains important musical information that is not contained in MIDI files (which is the most common symbolic music representation). For example, musical symbols in the sheet music like bar lines, rests, note ties, phrasing markings, sharps and flats, and key changes may not appear in a generated MIDI file in any explicit way. Third, generating sheet music cleanly decouples the musical composition and expressive performance aspects of music, allowing us to focus exclusively on the composition problem. Fourth, there is an abundance of sheet music data (>650k scores) available to the research community through the International Music Score Library Project (IMSLP) [19]. For the above reasons, we believe that studying the generation of sheet music images is an interesting and relevant problem.

Our goal is to explore the feasibility of generating sheet music. As an initial step, we focus on the specific task of generating monophonic incipit images, which are short snippets of sheet music that are used to identify a musical work. Using the PrIMuS dataset [20] as a testbed, we explore five different sequence-based approaches for generating incipits: (a) generating sequences of pixel columns, (b) generating sequences of image patches, (c) generating sequences of visual word tokens, (d) generating sequences

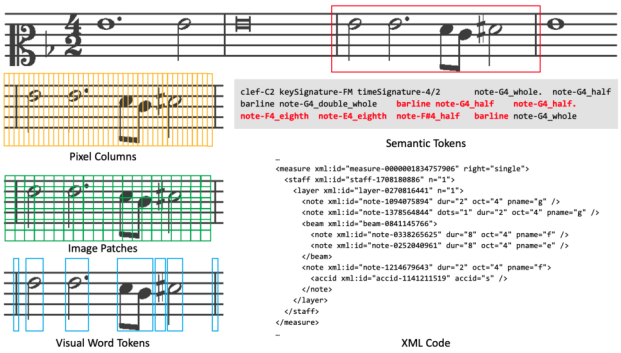

Figure 1. An example incipit from the PrIMuS dataset (top). We explore five different ways to generate sheet music: generating pixel columns, image patches, visual word tokens, semantic tokens, and XML code.

of semantic tokens, and (e) generating MEI code, an XML-based sheet music representation. For each of these five approaches, we train GPT-2 [21] and AWD-LSTM [22] language models, generate example incipits, and characterize the main challenges and problems with each approach.

2. SYSTEM DESCRIPTIONS

In this section we describe five different sequence-based approaches for generating a monophonic incipit image. All five approaches use language models to generate sequences of tokens, but they differ in the way that the tokens are constructed or defined.

2.1 Generating Pixel Columns

The first approach is to generate sequences of pixel columns. This is done in five steps, which are described in the following five paragraphs.

The first step is to standardize the incipits. The raw incipits in the PrIMuS dataset have variable height and width, so it is necessary to standardize the height of the incipit. This is done by selecting a fixed height for all incipits, ensuring that the five staff lines are vertically centered in the resulting image. This ensures that the staff lines appear in the same vertical pixel positions in each image. After the standardization, the images have a fixed height but variable width.

The second step is to convert pixel columns to words. Here, we interpret each standardized image as a sequence of words, where each word corresponds to a single pixel column. We convert pixel columns to words by simply binarizing the (grayscale) pixel values and interpreting each column's binary representation as a word.

The third step is to train a language model on the word sequences. This can be done in a self-supervised manner by predicting the next word in a sequence. We ran experiments with two different language models: (a) the AWD-LSTM model [22], which is a 3-layer LSTM model that incorporates multiple forms of dropout and (b) the GPT-2 small model [21], which is a 6-layer Transformer decoder model. We use the implementation of AWD-LSTM from the fastai [23] library and the implementation of GPT-2 from the huggingface [24] library. We use the default recommended vocabulary size in each respective library (GPT-2 small 30k, AWD-LSTM 60k), and map infrequently occurring words to a special unknown word token <unk>.

The fourth step is to generate new word sequences given a starting seed sequence. We do this by generating a new token at each time step and using the generated token as input to the next time step. For AWD-LSTM, we sample from the softmax distribution with a temperature of 0.8 (default in fastai). For GPT-2, we use top $K = 50$ sampling (default in huggingface).

The fifth step is to render the generated word sequence as an image. Since each word is a binary string of 0's and 1's corresponding to a single pixel column, we simply concatenate the generated words in a matrix and directly render it as a black-and-white image.

2.2 Generating Image Patches

The second approach is to generate sequences of image patches. This is done in five steps, which are described in the following five paragraphs.

The first step is to standardize the incipits. This is done in the same manner as described in Section 2.1 above.

The second step is to convert image patches to words. Figure 1 shows a graphical illustration of this process for a single measure (green squares at left). Instead of using pixel columns as words, we can instead use 15×15 square image patches as words. This allows individual words to retain context in both the vertical and horizontal dimensions, rather than only along the vertical dimension. This approach has been shown to work effectively for image classification in the Vision Transformer model [25]. After breaking the incipit image into square patches, we form a sequence by considering patches from top to bottom and from left to right. Each image patch is binarized and interpreted as a single word.

The third step is to train a language model on the word sequences. This is done in the same manner as before. We tried training both with and without special <col> tokens indicating the end of each column of patches but found that including the <col> tokens did not help.

The fourth step is to generate new word sequences given a starting seed sequence. We generate a sequence of words autoregressively as described previously.

The fifth step is to render the generated word sequence as an image. Each word is a binary string of length $15 \times 15 = 225$ and can be directly reshaped into a binary image patch. The image patches are assembled in the same rasterized order to generate the incipit image.

2.3 Generating Visual Word Tokens

The third approach is to generate sequences of visual word tokens. This is done in five steps, which are described in the following five paragraphs.

The first step is to standardize the incipits. This is done in the same manner as described in Section 2.1.

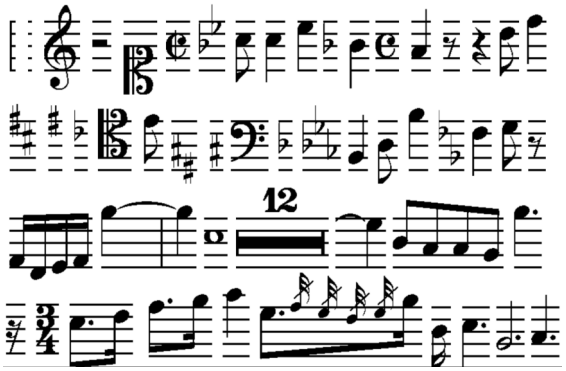

Figure 2. Visual word tokens. The top two rows show some of the most common visual word tokens, and the bottom two rows show a random selection of words in the vocabulary.

The second step is to tokenize each standardized incipit into a sequence of visual word tokens. This can be done by identifying pixel columns that only contain empty staff lines, and then interpreting these pixel columns as a separator (similar to tokenizing text based on whitespace). Figure 1 shows an example of this process for a single measure (bottom left). With the PrIMuS dataset, there are only a few unique "empty" pixel columns, but with other sheet music data one could train a simple model to identify empty pixel columns. Note that this process yields visual word tokens that span the entire height of the incipit and have variable width. The top two rows of Figure 2 show some of the most common visual word tokens, and the bottom two rows show a random selection of other visual word tokens in the vocabulary. One benefit of this approach is that entire symbols (e.g. clef signs) are interpreted as a single word, which allows the model to focus on modeling sequences of symbols rather than on rendering common symbols correctly. We found it helpful to impose a minimum separator width to avoid splitting certain symbols (like single whole notes) into two parts.

The third step is to train a language model on the visual word token sequences. This is done in the same manner as described in Section 2.1. The separator pixel columns are removed from the data and do not appear in training.

The fourth step is to generate new word sequences given a starting seed sequence. We generate a sequence of visual word tokens autoregressively as described in Section 2.1.

The fifth step is to render the generated visual word token sequence as an image. For simplicity, we simply concatenate the visual word tokens side by side and insert a fixed-width separator between each visual word token.

2.4 Generating Semantic Tokens

The fourth approach is to generate a sequence of semantic tokens. This is done in four steps, which are described in the next four paragraphs.

The first step is to represent each incipit as a sequence of semantic tokens. In the original PrIMuS dataset [20], the creators introduce a language for encoding musical symbols in a way that captures semantically meaningful information. Figure 1 shows an example incipit (top) and its corresponding sequence of semantic tokens (gray colored box on right). We tokenize by symbol and by symbol characteristics, so that a string "note-E4_eighth rest-eighth" would be converted into the sequence "note E4 eighth rest eighth". After tokenizing in this manner, there are 279 unique words across the PrIMuS dataset.

The second step is to train a language model on the semantic token sequences. This is done in the same manner as described in Section 2.1.

The third step is to generate new word sequences given a starting seed sequence. We generate semantic word tokens autoregressively as described in Section 2.1.

The fourth step is to render the generated semantic token sequence as an image. We first convert the semantic token sequence into Plaine and Easie [26] code using a relatively simple set of rules and logic. We then render the Plaine and Easie code as an image using Verovio [27].

2.5 Generating XML Code

The fifth approach is to generate MEI [28] code, which is a way of encoding music notation in XML form. This is done in four steps, which are described in the following four paragraphs.

The first step is to preprocess the MEI representation into a sequence of words. Figure 1 shows a portion of the MEI representation (bottom right) that encodes the measure highlighted in red (top right). To simplify the data, we filter out information that is not needed for the engraving process, such as headers, footers, and XML identifiers. We also insert spaces to keep conceptually distinct tokens separate. For example, a tag "<staff n='1'>" would be tokenized into the sequence "<staff", "n=", "'1'", and ">". Tokenizing the MEI representations in this manner, we get a vocabulary of 769 words across the PrIMuS dataset.

The second step is to train a language model on the MEI token sequences. This is done in the same manner as described in Section 2.1.

The third step is to generate new word sequences given a starting seed sequence. We generate a sequence of MEI tokens autoregressively until a closing </music> tag is generated. Note that, unlike the previous four methods, we cannot directly control the width of the generated incipit since the nested XML tags need to be properly closed.

The fourth step is to render the generated MEI representation as an image using Verovio. Note that the generated MEI code may not be a properly formatted XML document, so this rendering process may or may not be successful.

3. EXPERIMENTAL RESULTS

In this section we present sample incipits generated by using all five approaches described in Section 2. We focus on qualitative analysis of the generated images in this section, and in Section 4 we perform several quantitative analyses on the two most promising approaches.

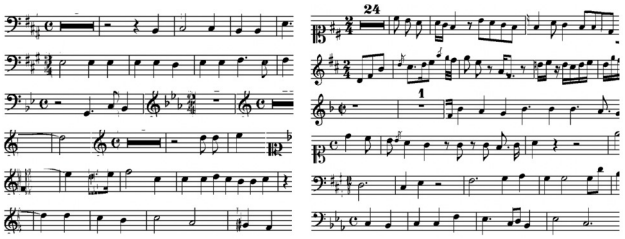

Figure 3. Sample images from generating pixel columns with AWD-LSTM (left) and GPT-2 (right).

All systems described in Section 2 were trained on the PrIMuS dataset [20]. It contains 87,678 real-music incipits, which are sequences of notes (often the first ones) used to identify a melody or musical work. Each incipit in the dataset is available in five different formats: (1) Plaine and Easie code, (2) a rendered image, (3) an MEI representation, (4) a sequence of semantic tokens (as described in Section 2.4), and (5) a sequence of agnostic tokens, which encodes the graphical symbols in the music score with their position in the staff but without any musical meaning. This dataset is ideal for our study because the incipits are short, there are a large number of incipits, and there are multiple representations of the data. We use the rendered image for the first three approaches (pixel columns, image patches, visual word tokens), the semantic tokens for the fourth approach (semantic tokens), and the MEI representation for the fifth approach (XML code).

During language model training, we use a 90-10 train-validation split for AWD-LSTM and a 95-5 train-validation split for GPT-2. Once the language model training has converged, we use the model to generate sample incipits as described in Section 2. Below, we present sample images generated from each of the five approaches and comment on the problems with each approach.

Figure 3 shows sample images from generating pixel columns. The left half of the figure shows images generated from the AWD-LSTM model, and the right half of the figure shows images generated from the GPT-2 model. We can immediately see several issues with the generated images: there are visual artifacts that appear as vertical lines (particularly with the AWD-LSTM model), the measures do not have a consistent number of beats, there are invalid key signatures (top left incipit), and the AWD-LSTM model seems to reset the clef and key/time signatures frequently. On the positive side, this approach is able to model symbols like bar lines, grace notes, clefs, key signatures, time signatures, and multiple bar rests in a single unified framework.

Figure 4 shows sample images from generating image patches. The left half of the figure shows images generated from the AWD-LSTM model, and the right half of the figure shows images generated from the GPT-2 model. We see some visual artifacts that come from incoherently generated symbols (e.g. the time signature in the fourth example on the right, the sixteenth note beams in the last example on the right), the measures do not have a consistent number of beats, and occasionally the model seems to

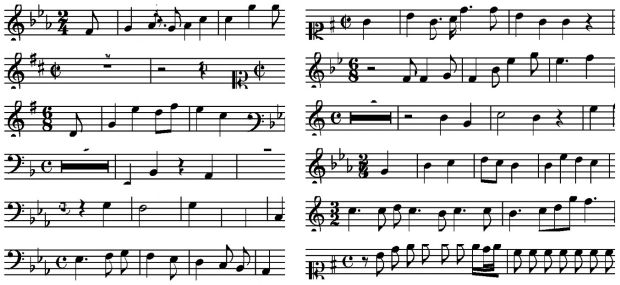

Figure 4. Sample images from generating image patches with AWD-LSTM (left) and GPT-2 (right).

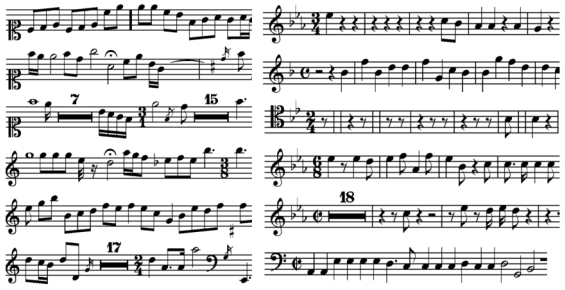

Figure 5. Sample images from generating visual word tokens with AWD-LSTM (left) and GPT-2 (right).

be confused about where each image patch is located relative to the global image (e.g. the incorrectly placed bar lines in the fifth example on the left). Again, the AWD-LSTM model seems to have a problem with resetting the clef and key signature too frequently (e.g. the second and third examples on the left). Compared to the pixel column approach, this approach seems to have somewhat less variety in terms of rhythms and durations (e.g. sixteenth notes, rests).

Figure 5 shows sample images from generating visual word tokens. The left half of the figure shows images generated from the AWD-LSTM model, and the right half of the figure shows images generated from the GPT-2 model. We don't see any visual artifacts in reconstructing symbols since each token is itself an entire symbol (or collection of symbols), but we see a lot of semantically incoherent patterns. For example, there are accidentals that appear without a corresponding note (fifth example on left), ties to notes that don't appear (second example on left), and bar lines sometimes appear too frequently (third example on right) and at other times too infrequently (last three examples on left). Measures still do not have a consistent number of beats, but we do notice that the model is often metrically coherent and correct immediately after a time signature symbol is generated. There are stylistic issues as well: some measures are technically coherent but do not follow typical conventions of sheet music, such as generating three consecutive quarter note rests in a measure rather than a whole measure rest (top example on right).

Figure 6 shows sample images from generating semantic tokens. The left half of the figure shows images generated from the AWD-LSTM model, and the right half of

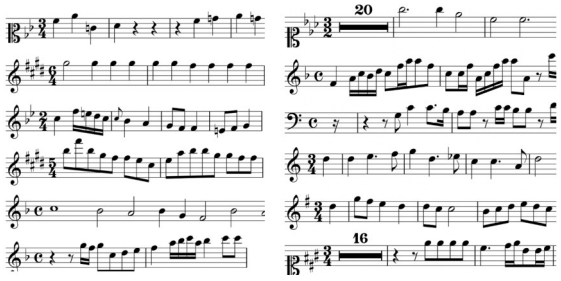

Figure 6. Sample images from generating semantic tokens with AWD-LSTM (left) and GPT-2 (right).

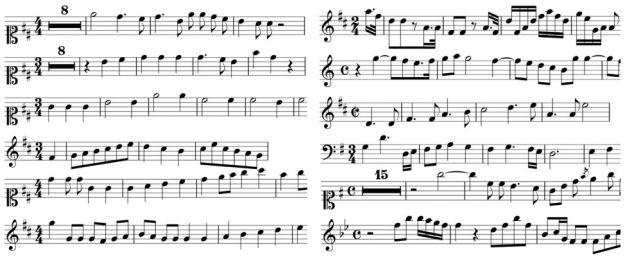

Figure 7. Sample images from generating XML code with AWD-LSTM (left) and GPT-2 (right).

the figure shows images generated from the GPT-2 model. There is a noticeable improvement in semantic coherence compared to the previous three approaches. For example, most generated measures have the correct number of beats, even across different time signatures (e.g. 3/4 in the first example on left, 6/4 in the second example on left, 2/4 in the third example on left, 4/4 in last example on left). However, we still occasionally see measures with an incorrect number of beats (e.g. fifth example on left), and the model particularly seems to struggle with unusual time signatures (e.g. 5/4 in the fourth example on left). The models seem to display a good amount of diversity in rhythms and durations, both with notes and rests. Note that, because the incipit is translated from semantic tokens to Plaine and Easie code and then to a rendered image, accidentals are guaranteed to be rendered correctly with a given key signature (e.g. first and third examples on left).

Figure 7 shows sample images from generating MEI code. The left half of the figure shows images generated from the AWD-LSTM model, and the right half of the figure shows images generated from the GPT-2 model. Note that some generated MEI code was ill-formed and could not be rendered as an image, so the examples below are self-selected from the examples that were valid MEI files. We can see that most measures have the correct number of beats, and this holds true across a range of time signatures (e.g. 4/4, 3/4, 2/4, 2/2 in the examples in Figure 7). Because the MEI representation explicitly encodes symbols like note beams and ties, the generated incipits also seem to obey most stylistic conventions. The AWD-LSTM model has noticeably less diversity than the GPT-2 model, but the latter demonstrates a reasonably good diversity of rhythms and durations in notes and rests.

Summary. The three image-based approaches (pixel columns, image patches, visual word tokens) are not recommended because the generated incipits are not semantically coherent, even when they do not have any obvious visual artifacts. The primary issue with generating semantic tokens and MEI code is metrical coherence – ensuring that the number of beats in a measure is consistent with the time signature. Generating MEI code also has the issue of malformed XML code that cannot be rendered properly. [1]

Our experiments lead us to two main conclusions. First, the two approaches that we believe are most promising to explore in future work are generating sequences of semantic tokens and generating XML code. Second, the main technical issue that must be resolved to generate meaningful sheet music is metrical coherence – exploring ways to utilize powerful and flexible language models while also conforming to the strict metrical rules of music. This emerges as a main challenge for future work.

4. ANALYSIS

In this section we perform additional quantitative analyses on the two recommended approaches: generating semantic tokens and generating MEI code.

We can quantify how metrically coherent the generated semantic token sequences are in the following manner. First, we generate 16,000 incipits in the form of semantic token sequences. When generating these sequences, we specify the clef, key signature, and time signature as part of the seed sequence, and we divide the 16,000 incipits evenly across eight different time signatures. Second, we segment each incipit into distinct measures by detecting bar lines in the generated sequence. If the generated sequence contains a change of time signature, we only consider the portion of the incipit before the time signature change and ignore the rest. This policy allows us to measure and compare metrical coherence across different time signatures. Third, we calculate the fraction of measures that are metrically coherent, which we define as a measure that contains the correct number of beats specified by the time signature. We can determine this based on the time signature and the time duration of each semantic token. Note that some semantic tokens like slurs, bar lines, grace notes, and clef symbols have no duration, while all notes and rests have a non-zero duration.

Table 1 shows the results of this analysis on the generated semantic token sequences. The leftmost column shows the different time signatures that we used as starting seeds. The next two columns show the total number of measures generated by the AWD-LSTM model and the percentage of those generated measures that are metrically coherent. Note that each incipit will have a variable number of measures, so we can only indirectly control the total number of generated measures. The last two columns show the same information for the GPT-2 model. The bottom row shows a total across all time signatures.

[1] One caveat is that our recommendations assume that the amount of training data is the same across different encodings.

Time	AWD-LSTM		GPT-2	
Signature	# meas	% coh	# meas	% coh
2/4	5820	75.2	8794	80.9
3/4	6229	79.5	8191	79.4
4/4	4109	77.6	5936	77.9
5/4	3762	0.3	5276	35.1
6/4	3640	70.4	5553	77.0
7/4	3781	0.05	5947	0.4
3/8	7091	77.9	10069	77.0
6/8	4334	75.8	7303	75.7
Total	45982	61.7	57069	66.0

Table 1. Measuring metrical coherence of generated semantic token sequences. Columns indicate the number of generated measures and the percentage of measures that have the correct number of beats.

Time	LSTM		GPT-2	
Signature	# meas	% coh	# meas	% coh
2/4	4754	35.2	6650	85.4
3/4	4998	69.4	6098	86.9
4/4	4418	74.1	4810	77.8
5/4	5400	3.2	4391	9.0
6/4	4298	13.4	3519	59.5
7/4	5586	2.4	3667	8.3
3/8	4120	22.5	7060	91.5
6/8	4354	81.1	5048	83.0
Total	37928	36.3	41243	68.3

Table 2. Measuring metrical coherence of generated MEI code.

There are two things to notice about Table 1. First, both models have very low metrical coherence for uncommon time signatures like 5/4 and 7/4. For example, less than 1% of measures in 7/4 were metrically coherent in both models, which is likely a reflection of the fact that this time signature has little or no representation in the PrIMuS dataset. Clearly, these language models are not learning a notion of metrical structure and are unable to generalize to unseen or uncommon time signatures. Second, the metrical coherence measures for GPT-2 and AWD-LSTM are very similar for common time signatures like 3/4, 4/4, 3/8, and 6/8, with numbers falling somewhere between 75-80%. For less common time signatures, however, GPT-2 often significantly outperforms AWD-LSTM, suggesting that it is able to generalize slightly better.

We measure the quality of the generated MEI code in two different ways. The first measurement is simply the percentage of generated incipits that are valid XML documents. Note that every MEI document begins with a <music> tag and ends with a corresponding </music> tag. Therefore, when generating MEI code, we autoregressively generate tokens until a closing </music> tag is encountered. The resulting MEI document is then rendered with Verovio into an SVG file. If the rendering process generates an error or warning, we consider the MEI code to be malformed. We generated 16,000 incipits with each model (2,000 incipits for each of the 8 time signatures shown in Table 1) and found that 93.1% of the AWD-LSTM incipits and 95.9% of the GPT-2 incipits were valid.

The second way to measure the quality of generated MEI code is to quantify metrical coherence. This is done in a similar manner as described above: we generate 2,000 incipits for each of the same eight time signatures, segment each incipit into measures by detecting bar lines, and then calculate the percentage of measures that are metrically coherent. Note that some of the incipits will not be valid XML documents, so in our analysis we only consider measures from generated documents that are valid.

Table 2 shows the results of this analysis on the generated MEI incipits. There are two things to notice. First, we see that GPT-2 outperforms AWD-LSTM on metrical coherence for every time signature, sometimes by very large margins. For instance, on incipits with a 3/8 time signature, 91.5% of the measures generated by GPT-2 are metrically coherent, compared to only 22.5% for AWD-LSTM. Second, comparing metrical coherence between semantic tokens (Table 1) and MEI (Table 2), we see that AWD-LSTM performs substantially worse with MEI representations across all time signatures (61.7% vs 36.3% total). GPT-2 seems to perform better with MEI on common time signatures, but has extremely variable performance with uncommon time signatures.

Our quantitative analyses have a clear takeaway: metrical coherence is the primary issue with both approaches. The percentage of metrically coherent measures is sufficiently low as to be prohibitive, particularly with less common time signatures. It is clear that simply having more data will not solve this problem, since having more data will still have an extreme imbalance across different time signatures. Some possible solutions include using rejection sampling or incorporating timing and position information either into the Transformer model as additional embeddings or into the data itself (e.g. REMI [11]). We identify this as a main challenge for future work.

5. CONCLUSION

We explore the feasibility of generating monophonic sheet music images using sequence-based models. We consider five different approaches: generating sequences of pixel columns, image patches, visual word tokens, semantic tokens, and XML-like tags. We train language models for all five approaches on the PrIMuS dataset and generate sample incipits to better understand the pros and cons of each approach. Our main findings are that: (a) the image-based approaches can yield realistic looking sheet music but are often semantically incoherent and are therefore not recommended, (b) generating semantic tokens and MEI code yield the most semantically coherent sheet music, and (c) metrical coherence is the biggest issue that needs to be addressed in future work.

6. ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 2144050.

7. REFERENCES

[1] D. von Rütte, L. Biggio, Y. Kilcher, and T. Hoffman, "FIGARO: Generating symbolic music with fine-grained artistic control," *arXiv preprint arXiv:2201.10936*, 2022.

[2] G. Hadjeres and L. Crestel, "Vector quantized contrastive predictive coding for template-based music generation," *arXiv preprint arXiv:2004.10120*, 2020.

[3] S. Di, Z. Jiang, S. Liu, Z. Wang, L. Zhu, Z. He, H. Liu, and S. Yan, "Video background music generation with controllable music transformer," in *Proceedings of the ACM International Conference on Multimedia*, 2021, pp. 2037–2045.

[4] P. Dhariwal, H. Jun, C. Payne, J. W. Kim, A. Radford, and I. Sutskever, "Jukebox: A generative model for music," *arXiv preprint arXiv:2005.00341*, 2020.

[5] K. Choi, C. Hawthorne, I. Simon, M. Dinculescu, and J. Engel, "Encoding musical style with transformer autoencoders," in *International Conference on Machine Learning*, 2020, pp. 1899–1908.

[6] R. Yang, D. Wang, Z. Wang, T. Chen, J. Jiang, and G. Xia, "Deep music analogy via latent representation disentanglement," in *Proceedings of the International Society for Music Information Retrieval Conference*, 2019, pp. 596–603.

[7] J. Jiang, G. G. Xia, D. B. Carlton, C. N. Anderson, and R. H. Miyakawa, "Transformer VAE: A hierarchical model for structure-aware and interpretable music representation learning," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 516–520.

[8] J. Jiang, G. Xia, and R. Dannenberg, "Representing music structure by variational attention," in *ML4MD Workshop at the International Conference on Machine Learning*, 2019.

[9] C.-Z. A. Huang, A. Vaswani, J. Uszkoreit, I. Simon, C. Hawthorne, N. Shazeer, A. M. Dai, M. D. Hoffman, M. Dinculescu, and D. Eck, "Music transformer: Generating music with long-term structure," in *International Conference on Learning Representations*, 2018.

[10] C. Hawthorne, A. Stasyuk, A. Roberts, I. Simon, C.-Z. A. Huang, S. Dieleman, E. Elsen, J. Engel, and D. Eck, "Enabling factorized piano music modeling and generation with the MAESTRO dataset," in *International Conference on Learning Representations*, 2019.

[11] Y.-S. Huang and Y.-H. Yang, "Pop music transformer: Beat-based modeling and generation of expressive pop piano compositions," in *Proceedings of the ACM International Conference on Multimedia*, 2020, pp. 1180–1188.

[12] F. T. Liang, M. Gotham, M. Johnson, and J. Shotton, "Automatic stylistic composition of Bach chorales with deep LSTM," in *Proceedings of the International Society for Music Information Retrieval Conference*, 2017, pp. 449–456.

[13] J. Ens and P. Pasquier, "MMM: Exploring conditional multi-track music generation with the transformer," *arXiv preprint arXiv:2008.06048*, 2020.

[14] C. Donahue, H. H. Mao, Y. E. Li, G. W. Cottrell, and J. McAuley, "LakhNES: Improving multi-instrumental music generation with cross-domain pre-training," in *Proceedings of the International Society for Music Information Retrieval Conference*, 2019, pp. 685–692.

[15] H.-W. Dong, W.-Y. Hsiao, L.-C. Yang, and Y.-H. Yang, "MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[16] Y. Ren, J. He, X. Tan, T. Qin, Z. Zhao, and T.-Y. Liu, "PopMAG: Pop music accompaniment generation," in *Proceedings of the ACM International Conference on Multimedia*, 2020, pp. 1198–1206.

[17] Y.-H. Chen, Y.-H. Huang, W.-Y. Hsiao, and Y.-H. Yang, "Automatic composition of guitar tabs by transformers and groove modeling," in *Proceedings of the International Society for Music Information Retrieval Conference*, 2020, pp. 756–763.

[18] S.-L. Wu and Y.-H. Yang, "The jazz transformer on the front line: Exploring the shortcomings of AI-composed music through quantitative measures," in *Proceedings of the International Society for Music Information Retrieval Conference*, 2020, pp. 142–149.

[19] "IMSLP Petrucci Music Library," https://imslp.org, accessed: 2022-05-20.

[20] J. Calvo-Zaragoza and D. Rizo, "End-to-end neural optical music recognition of monophonic scores," *Applied Sciences*, vol. 8, no. 4, p. 606, 2018.

[21] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[22] S. Merity, N. S. Keskar, and R. Socher, "Regularizing and optimizing LSTM language models," *arXiv preprint arXiv:1708.02182*, 2017.

[23] J. Howard and S. Gugger, "Fastai: a layered API for deep learning," *Information*, vol. 11, no. 2, p. 108, 2020.

[24] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 2020, pp. 38–45.

[25] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *International Conference on Learning Representations*, 2021.

[26] "Plaine & Easie Code," https://www.iaml.info/plaine-easie-code, accessed: 2022-05-20.

[27] "Verovio: A Music Notation Engraving Library," https://www.verovio.org, accessed: 2022-05-20.

[28] "Music Encoding Initiative," https://music-encoding.org, accessed: 2022-05-20.