COMPOSER'S ASSISTANT: AN INTERACTIVE TRANSFORMER FOR MULTI-TRACK MIDI INFILLING

Martin E. Malandro

Sam Houston State University

`malandro@shsu.edu`

ABSTRACT

We introduce Composer's Assistant, a system for interactive human-computer composition in the REAPER digital audio workstation. We consider the task of multi-track MIDI infilling when arbitrary track-measures have been deleted from a contiguous slice of measures from a MIDI file, and we train a T5-like model to accomplish this task. Composer's Assistant consists of this model together with scripts that enable interaction with the model in REAPER. We conduct objective and subjective tests of our model. We release our complete system, consisting of source code, pretrained models, and REAPER scripts. Our models were trained only on permissively-licensed MIDI files.

1. INTRODUCTION

Many generative models for music exist. For instance, MuseNet [1] and SymphonyNet [2] can generate or continue a piece of music, and Music Transformer [3] can continue a piano performance or harmonize a piano melody. When we tried using these tools as compositional aides, however, we quickly ran into limitations. For instance, while Music Transformer is capable of harmonizing a given melody, it does not offer the ability to keep part of the harmonization and regenerate the other part. MuseNet and SymphonyNet can generate a continuation of a user's prompt, but do not allow the user to regenerate individual instruments or measures within the continuation while keeping the rest of the continuation intact.

DeepBach [4] can perform infilling on Bach-like chorales in a window specified by the user. Motivated by the idea of extending the DeepBach user experience to more styles, arbitrary collections of instruments, and arbitrary infilling target locations, we train a transformer [5, 6] model on the task of multi-track MIDI infilling. Our model allows composers to generate new notes for arbitrary subsets of track-measures in their compositions, conditioned on any contiguous slice of measures containing the subset. (By a *track-measure*, we simply mean a measure within a track—see Figure 1.) We also build a novel system for interacting with our model in the REAPER digital
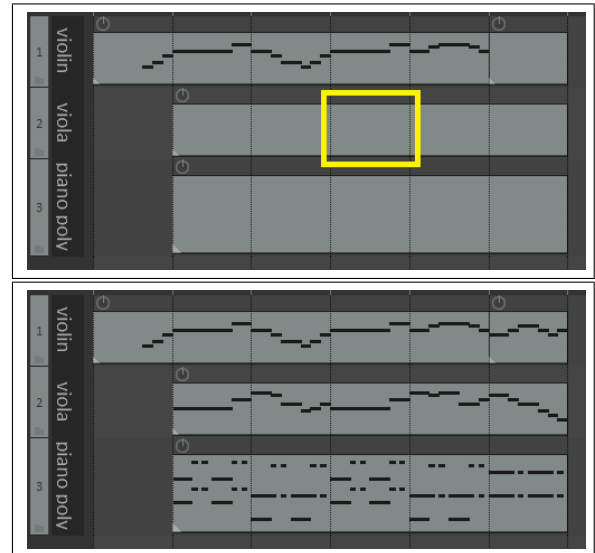
Figure 1. A prompt in REAPER, followed by a model output. Vertical lines separate measures. Users place empty MIDI items in REAPER to tell the model in which measures to write notes, and track names to tell the model what instrument is on each track. A track-measure in the prompt is boxed. Our model writes at least one note into every track-measure in every empty MIDI item in the prompt.

audio workstation (DAW). [1] Our system is cross-platform and easy to install. When a user runs one of our REAPER scripts, a model prompt is built directly from the slice of measures selected in the user's REAPER project, our model evaluates the prompt, and the model output is written back into the user's project—see Figure 1. All of this happens within a few seconds, allowing the user to listen to the output, modify it to create a new prompt, generate an output from that, etc. This allows our model to be used in an interactive manner, where model outputs are refined by the user over the course of several prompts.

We note that our infilling objective includes continuing a piece of music, simply by including empty measures at the end of the piece in the prompt. Additionally, our model has the ability to write variations: One can randomly mask $1/n$ of the track-measures in a measure selection and ask the model to fill in those parts, then feed the result back into the model with another $1/n$ masked, and so on, until

---

[1] Our system and video demo are available at `https://github.com/m-malandro/composers-assistant-REAPER`.

all track-measures have been masked and filled.

Toward the end of this project we discovered MMM [7, 8], which consists of two separate GPT-2–like [9] models trained on the tasks of measure infilling and track infilling. The authors include code to use these models to infill arbitrary subsets of track-measures, as we do. MMM comes in 4-bar and 8-bar variants, which are limited to inputs with 12 and 6 tracks, respectively, and its web demo is limited to inputs having a 4/4 time signature.

The primary contributions of this work are as follows. First, in Section 3 we introduce a novel data filtering and preprocessing approach, applicable to any MIDI dataset used for training models. Our approach helps rectify certain issues we have encountered when using other models. Second, we train and release a new model, capable of infilling arbitrary track-measures in an arbitrary slice of measures in a MIDI file, with no effective restriction (aside from a soft input token limit of 1650) on tempos, number of measures, or number of instrument tracks. Tracks may be polyphonic or monophonic in any combination. The only time signature restriction is that all measures must be eight quarter notes or fewer. Our model is more flexible than MMM and compares favorably to MMM in both objective and subjective tests—see Sections 6–7. Additionally, our model was trained only on permissively-licensed MIDI files, so its outputs should be usable by composers with minimal risk—see Section 5. Finally, we release our complete system, including training code and scripts that enable rapid interaction with our model in REAPER. Our model is the first DAW-integrated model capable of infilling parts for all 128 pitched MIDI instruments (including repeated instruments) and drums, in any combination.

2. RELATED WORK

As mentioned in Section 1, MMM [7, 8] performs multi-track infilling for all MIDI instruments (subject to bar and track limits), and DeepBach [4] performs multi-track infilling for Bach-like chorales. Coconet [10] also performs multi-track infilling for Bach-like chorales. MusIAC [11] incorporates user controls and performs track-based and measure-based infilling, although its inputs and outputs are limited to a maximum of three tracks, 16 measures, and four common time signatures. MusicVAE [12] can interpolate between two given clips of music, which can be viewed as a type of infilling. To our knowledge, all other existing music infilling systems are limited to monophonic infilling [13–17] or single-instrument infilling [18, 19].

Generating or continuing a piece of music can be seen as a special case of infilling. Models which can generate or continue a piece of music include [1–3, 20, 21].

Previous DAW-integrated generative music systems include [4, 18, 22]. NONOTO [23] is a model-agnostic interface that can be linked with a model to perform interactive measure-based infilling. This interface could potentially be altered to allow for the expanded type of infilling our model is capable of. However, we opt to build an interface between our model and REAPER directly, essentially using REAPER as the GUI for our model.

3. DATA FILTERING AND PREPROCESSING

In this section we describe our filtering and preprocessing approach, any portion of which can be applied to any dataset of MIDI files. First, we remove from the dataset any file whose notes seem to have no relation to the underlying grid (Section 3.1). Next, we dedupe files from the dataset using note onset chromagrams (Section 3.2). Finally, we preprocess all remaining files to standardize properties like track order (Section 3.3). This final preprocessing step includes a method for detecting and removing shifted duplicate and near-duplicate tracks within files (Section 3.4).

3.1 Cosine Similarity for On-Grid Note Detection

Every MIDI file has a measure and grid structure defined by tempo and time signature events. However, MIDI file authors are free to ignore this structure, and frequently do when recording free-flowing performances. Other models we have used occasionally write a note in the wrong place—e.g., a 32nd note away from where it clearly should be—and a small experiment we ran suggests that training on MIDI files that don't quantize well to the grid used by the model is a major cause of this. To address this, we remove from our dataset any MIDI file whose note onsets appear to have no relation to the underlying grid. This is not as simple as checking whether all (or most) note onsets occur on the grid, as many MIDI file authors who use the grid include "humanization" of note timings, where many note onsets that occur slightly off the grid nevertheless quantize correctly to the grid. For instance, in a MIDI file with a resolution of 960 ticks per quarter note, a humanized quarter-note performance might have notes occurring in a 40-tick window centered at every 960th tick.

To perform this filtering, given a MIDI file M, we quantize the note onsets in M to a resolution of 12 ticks per quarter note, and we form a length-12 vector v_M whose ith entry ($i \in \{0, \ldots, 11\}$) is the number of note onsets in M occurring i ticks after a grid quarter note. The idea is that if the note onsets in M have nothing to do with the grid, then v_M will point in a similar direction to the uniform vector $v_1 = (1, \ldots, 1) \in \mathbb{R}^{12}$. We therefore compute the cosine of the angle θ_M between v_M and v_1:

$$\cos(\theta_M) = \frac{\langle v_M, v_1 \rangle}{||v_M|| \cdot ||v_1||},$$

and we declare a threshold T such that when $\cos(\theta_M) > T$ we remove the file M from our dataset. Hand exploration indicated that $T = 0.8$ was a reasonable threshold, which we chose for this project. We note that a straight fully-on-grid 8th-note pattern M has $\cos(\theta_M) \approx 0.41$ and a straight fully-on-grid 16th-note pattern M has $\cos(\theta_M) \approx 0.58$.

3.2 Deduping Using Note Onset Chromagrams

We dedupe our dataset to avoid data imbalance during training and to prevent overlap between our training and test sets. Given a MIDI file M, we compute a size-12 set of note onset chromagrams using the following procedure.

First, we remove all drum tracks from M. Then, using a 12-tick-per-quarter-note grid, we quantize the note onsets in M to the nearest 16th note or 8th note triplet. Then, we remove all empty measures at the beginning and end of M, and we replace each set of contiguous empty measures within M with one empty measure. Then, for each tick in M and for each pitch class, we record whether M has at least one note onset of that pitch class at that tick. This information comprises one note onset chromagram for M. The other 11 come from repeating this procedure for each possible transposition of M. We dedupe the dataset by keeping only one file with a given set of note onset chromagrams. Quantization helps us catch files that differ only trivially in grid resolution and/or note onset times, while transposition helps us catch files that differ only in key.

3.3 Preprocessing of Individual MIDI Files

After deduping, we preprocess each MIDI file in our dataset in the following way.

First, we arrange the information in the MIDI file so that each track holds notes for one instrument. We order tracks according to their MIDI instrument number (0–127), taking drums as instrument 128. We also consolidate all drum tracks to a single track, and we apply a drum simplification map (consolidating, e.g., three different bass drum pitches to a single pitch).

Next, we apply pedal information in the file (if present) to extend note lengths, and then delete all continuous controller (cc) data. We do not model cc data in this project.

With the exception of drums, we allow multiple tracks to use the same instrument. However, when this happens, if there is more than one track having a given instrument, we remove all but one of those tracks that are equal to, a shift of, or close to a shift of another track with the same instrument, using the procedure in Section 3.4.

We impose the restriction that all measures must be eight quarter notes or fewer. If any time signature in the file declares longer measures, we alter the time signatures to shorten the measures.

Finally, using a 24-tick-per-quarter-note grid, we quantize the events in the file to the nearest 32nd note or 16th note triplet. This is ultimately the level of quantization we use to train our model. (Earlier experiments involved quantizing to 16th notes or 16th notes + 8th note triplets, which we found insufficient for expressive generation.)

3.4 Removing Shifted Duplicate and Near-Duplicate Tracks

A MIDI file may contain duplicate tracks. Such tracks contain no useful information for modeling, so we remove them. Shifted duplicate tracks are frequently used by MIDI file authors to encode *delay* effects (as the MIDI spec offers no way to encode the use of a delay directly). Choosing to use a delay is a mixing decision, not a compositional decision, and we want our model to focus on making compositional decisions, so we remove shifted duplicate tracks as well. We have also seen tracks that are duplicates or shifted duplicates of other tracks within a file, plus or minus a few notes and/or humanization. We remove such near-duplicate tracks as well.

Given a note n in a track T, let $\text{st}(n)$ and $\text{end}(n)$ indicate the start and end times of the note n, respectively, and let $\text{pitch}(n) \in \{0, \dots, 127\}$ indicate the MIDI pitch of n. We record, for $p \in \{0, \dots, 127\}$, the union of closed intervals

$$I_T(p) = \cup_{n \in T : \text{pitch}(n) = p} \{[\text{st}(n), \text{end}(n)]\} \subseteq \mathbb{R},$$

and we define $|I_T| = \sum_{p=0}^{127} |I_T(p)|$, where $|I_T(p)|$ is the sum of the lengths of the disjoint intervals in $I_T(p)$.

Given tracks T_1 and T_2, we define the *overlap measure* $O(T_1, T_2) \in [0, 1] \subseteq \mathbb{R}$ to be

$$O(T_1, T_2) = \frac{\sum_{p=0}^{127} |I_{T_1}(p) \cap I_{T_2}(p)|}{\max(|I_{T_1}|, |I_{T_2}|)}.$$

The idea is that $O(T_1, T_2)$ measures the percentage of the note intervals in the larger of the two tracks accounted for by the note intervals in the smaller of the two.

We use a threshold of 0.9 for asserting near-overlap between two tracks. As we go through the tracks in a MIDI file in order, a later track T is thrown out if there exists an earlier track T_0 using the same instrument such that some shift T_s of T of no more than a half note has the property that $O(T_0, T_s) \geq 0.9$. In our experience with our trained model, we have found this preprocessing step sufficient to prevent the model from outputting duplicates or shifted duplicates of tracks in its inputs.

4. OUR LANGUAGE

After applying the procedure from Section 3 to a collection of MIDI files, we process the files into an event-based language for modeling. Our language is similar to the standard event-based MIDI language used for piano performance modeling in [3]. However, we use explicit measure tokens to denote the start of each measure. Also, we do not model velocity of individual notes directly. Instead, we assign a dynamics level to each measure based on the average velocity of the notes in the measure. We use eight dynamics levels, with thresholds learned from data.

The tokens used by our language are as follows:

- M:x, $x \in \{0, \dots, 7\}$. Declares a measure of dynamics level x.

- B:x, $x \in \{0, \dots, 7\}$. Declares the tempo (BPM) level at the start of a measure. We use eight tempo levels, with thresholds learned from data.

- L:x, $x \in \{1, \dots, 192\}$. Declares that a measure has length equal to x ticks.

- I:x, $x \in \{0, \dots, 128\}$. Changes the current instrument to MIDI instrument x ($128 = $ drums).

- R:x, $x \in \{1, \dots, 63\}$. Declares that the current instrument is the same MIDI instrument as another instrument in the file/project, but on a different track. Higher x values indicate lower average pitch.
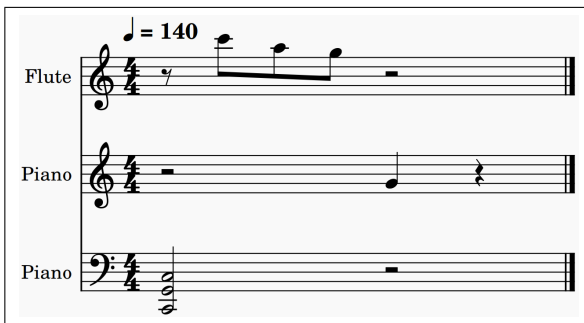
Figure 2. We tokenize this measure as M:5 B:6 L:96 I:0 w:48 d:24 N:67 I:0 R:1 d:48 N:36 N:43 N:48 I:73 w:12 d:12 N:84 w:12 N:81 w:12 N:79. Note that piano and flute are MIDI instruments 0 and 73.

- N:x, $x \in \{0,\dots,127\}$. Note of pitch x. Used by instruments 0–127.

- D:x, $x \in \{0,\dots,127\}$. Drum hit of drum pitch x. Used by instrument 128.

- d:x, $x \in \{0,\dots,192\}$. Sets the duration of each note declared from this point forward to x ticks.

- w:x, $x \in \{1,\dots,191\}$. Advances the current insertion point for new notes in the measure by x ticks.

- `<extra_id_x>`, $x \in \{0,\dots,255\}$. Mask tokens.

- `<mono>`, `<poly>`. Instructs the model to write a monophonic or polyphonic part for a masked part. For our purposes a monophonic part is one where no two notes in the part have the same onset tick.

Tokens are assembled in a standardized manner to represent measures. Each measure begins with M:x, B:x, and L:x tokens. I: commands are included for a measure only when that instrument is present in the measure. We do not intermingle instrument note instructions as we write each measure from left to right (as MuseNet [1] did), as that would make it difficult to mask individual instrument parts within measures. Rather, we write the full part for one instrument within the measure before writing the full part for the next instrument within the measure. Figure 2 contains an example of a tokenized measure.

To form songs, we simply concatenate measures.

5. MODEL, DATA, AND TRAINING PROCEDURE

We use recent recommendations from the language modeling community to design and train our model. Based on the recommendations in [24–26], we choose a T5 (full, relative-attention) encoder-decoder architecture [6]. We opt for a full attention model because such models were found to outperform memory-efficient models in [24] when the full input sequence fits in memory, as we expect to be the case in most real-world applications of our model. Also, we adopt the *DeepNarrow* strategy of [27], opting for a model dimension of 384, 10 encoder layers, and 10 decoder layers. For training, we use the `pytorch` [28]

Hugging Face [29] implementation of T5. For inference, we use nucleus sampling [30] with a threshold of $p = 0.95$.

To train a model that is essentially free of copyright worry, we collect MIDI files from the internet marked as being in the public domain, freely available to use without attribution, or available under a CC BY license. We exclude files marked as having share-alike or non-commercial licenses, since we want composers to be able to use model outputs however they wish. We also collect private donations and files from the internet for which we secure direct permission from the MIDI file authors to use for training. This results in a dataset of approximately 40K MIDI files after filtering. Most of our training files are in Western classical, folk, and hymnal styles, although some modern styles are also present.

We follow the standard approach to the training of large language models of splitting our training procedure into pretraining and finetuning phases. A similar approach was also used in [31]. For pretraining, we use the T5 corrupted-span sequence-to-sequence objective [6]. We begin by pretraining on the 192K training files in the CocoChorales [32] dataset and their piano reductions for three epochs. The CocoChorales are only used for this initial pretraining to teach the model the basics of music theory and our language. We then move on to our dataset of 40K MIDI files. After tokenization and corruption, we greedily chunk each song into inputs of 512 or fewer (short) or 1650 or fewer (long) tokens. Additionally, each song in our dataset is transposed a random amount between -5 and +6 semitones (inclusive) for each epoch. Following the recommendations in [24], we train our model on short examples for 20 epochs and then long examples for 11 epochs. We release the resulting pretrained model, which others may find useful for finetuning on downstream tasks.

For finetuning, we continue to leverage the corrupted-span sequence-to-sequence objective to finetune our model on the task of multi-track MIDI infilling. We create training examples from songs in our training dataset by taking slices of measures from the songs and masking subsets of track-measures from these slices. During finetuning every N:, D:, d:, and w: token for a given track-measure is masked, and corresponds to a single mask token. With probability 0.75, we add a `<poly>` or `<mono>` token corresponding to the nature of the masked tokens for each mask. (We choose not to include these tokens in every training example since users will not always include these instructions in their prompts.) Finetuning examples are limited to inputs with a maximum of 1650 tokens and outputs with a maximum of 1650 tokens.

We generate our finetuning masks by selecting from mask patterns that we consider to be musically relevant and/or useful for training. To help train our model for use on small numbers of measures, we also occasionally (15% of the time) truncate examples to a random smaller number of measures than the number allowed by our token limits. As with pretraining, each example is transposed randomly. We finetune our model for 51 epochs, and we release the resulting finetuned model.

Task	Our Model	Our Model -MP	MMM-8	MMM-4
Note F_1 results. Higher is better.				
8-bar random infill	$\mathbf{0.5414} \pm (0.1887)^a$	$0.5315 \pm (0.1904)^b$	$0.4153 \pm (0.1819)^c$	$0.4025 \pm (0.1652)^d$
16-bar random infill *	$\mathbf{0.5771} \pm (0.1661)^a$	$0.5705 \pm (0.1669)^b$	$0.4133 \pm (0.1534)^c$	$0.4059 \pm (0.1399)^d$
8-bar track infill	$\mathbf{0.179} \pm (0.1902)^a$	$0.1634 \pm (0.18)^b$	$0.1063 \pm (0.1573)^d$	$0.1427 \pm (0.1646)^c$
16-bar track infill	$\mathbf{0.1773} \pm (0.1752)^a$	$0.1609 \pm (0.165)^b$	$0.1107 \pm (0.1383)^d$	$0.1467 \pm (0.1489)^c$
8-bar last-bar fill	$0.5019 \pm (0.2719)^a$	$\mathbf{0.5063} \pm (0.2751)^a$	$0.4329 \pm (0.2445)^b$	$0.3756 \pm (0.2289)^c$
16-bar last-bar fill *	$\mathbf{0.5415} \pm (0.2853)^a$	$0.539 \pm (0.2823)^a$	$0.4338 \pm (0.2468)^b$	$0.3818 \pm (0.2293)^c$
Pitch class histogram entropy difference results. Lower is better.				
8-bar random infill	$\mathbf{0.2845} \pm (0.1627)^a$	$0.2948 \pm (0.1597)^b$	$0.3045 \pm (0.1561)^c$	$0.3049 \pm (0.1497)^c$
16-bar random infill	$\mathbf{0.2691} \pm (0.1325)^a$	$0.2797 \pm (0.1326)^b$	$0.3093 \pm (0.124)^c$	$0.3063 \pm (0.1138)^c$
8-bar track infill	$0.3933 \pm (0.3032)^c$	$0.42 \pm (0.3134)^d$	$\mathbf{0.2864} \pm (0.2966)^a$	$0.3021 \pm (0.2517)^b$
16-bar track infill	$0.3842 \pm (0.2654)^c$	$0.3995 \pm (0.2763)^c$	$\mathbf{0.284} \pm (0.2348)^a$	$0.3036 \pm (0.2072)^b$
8-bar last-bar fill	$\mathbf{0.3018} \pm (0.2661)^a$	$0.3072 \pm (0.2692)^a$	$0.3213 \pm (0.2602)^b$	$0.3439 \pm (0.2777)^c$
16-bar last-bar fill *	$\mathbf{0.2851} \pm (0.2652)^a$	$0.2925 \pm (0.2672)^a$	$0.3209 \pm (0.2619)^b$	$0.3454 \pm (0.2741)^c$
Groove similarity results. Higher is better.				
8-bar random infill	$\mathbf{0.9534} \pm (0.0298)^a$	$0.9519 \pm (0.0306)^b$	$0.9333 \pm (0.0369)^c$	$0.9314 \pm (0.0364)^d$
16-bar random infill *	$\mathbf{0.956} \pm (0.027)^a$	$0.9552 \pm (0.0275)^b$	$0.9323 \pm (0.0337)^c$	$0.9317 \pm (0.032)^c$
8-bar track infill	$\mathbf{0.9115} \pm (0.0592)^a$	$0.9069 \pm (0.0617)^b$	$0.8921 \pm (0.0695)^d$	$0.8987 \pm (0.0626)^c$
16-bar track infill	$\mathbf{0.9113} \pm (0.0547)^a$	$0.9082 \pm (0.0553)^b$	$0.8946 \pm (0.0561)^d$	$0.9011 \pm (0.0536)^c$
8-bar last-bar fill	$0.9517 \pm (0.0414)^a$	$\mathbf{0.9524} \pm (0.0411)^a$	$0.9381 \pm (0.045)^b$	$0.9334 \pm (0.0457)^c$
16-bar last-bar fill *	$\mathbf{0.9544} \pm (0.0481)^a$	$0.9542 \pm (0.0424)^a$	$0.938 \pm (0.051)^b$	$0.9339 \pm (0.0475)^c$

Table 1. Objective infilling summary statistics. All cells are of the form mean $\pm$ (std dev)^s, where s is a letter. Different letters within a row indicate significant location differences ($p < 0.01$) in the samples for that row according to a Wilcoxon signed rank test with Holm-Bonferroni correction. Asterisks ($*$) indicate a significant performance difference ($p < 0.01$) between a 16-bar task and the 8-bar task in the previous row for our model according to a Wilcoxon rank sum test.

6. OBJECTIVE EVALUATION OF OUR MODEL

To form our test set, we select a set of 2500 MIDI files from the Lakh MIDI dataset [33, 34] that is disjoint (according to the procedure in Section 3.2) from our training set, all in 4/4 time and having at least 16 measures. Given a MIDI file in our test set, for each of the three mask patterns below, we select an 8- and a 16-measure slice of the file and mask the selected slice with that mask pattern. We thus generate six test examples from each test file, corresponding to the six different tasks on which we evaluate models. Given a slice of measures, our mask patterns for testing are:

1. Random mask: Each track-measure in the slice is masked with probability 0.5.

2. Track mask: Up to half of the tracks t are selected at random from the slice, and every measure of each such track t is masked.

3. Last-bar mask: Given the last measure m of the slice, measure m of every track is masked. This pattern is used to measure the ability of models to continue songs.

The ground truth for each example consists of the masked notes in the example. In our test data, 99% and 75% of our 8-measure and 16-measure prompts (respectively) encode to 1650 or fewer tokens. When input prompts are longer than 1650 tokens, we chunk the prompts prior to evaluating them with our model.

To compare our model to MMM [7, 8], we modify the MMM Colab worksheet to run our examples through the MMM models in batches. We recreate our test examples, quantizing them from their underlying MIDI files to MMM's 12-tick-per-quarter-note resolution, and then modify them to accommodate the restrictions of the MMM models: Since the 4-bar and 8-bar MMM models are limited to inputs containing a maximum of 12 and 6 tracks, respectively, we chunk each test example into 4-bar and 8-bar chunks, and then we split each chunk into sub-chunks consisting of up to 12 and 6 tracks. The MMM models have a strict input + output token limit of 2048, so when sub-chunking, we only add enough tracks to a sub-chunk to ensure that the input + ground truth has no more than 2048 tokens. This biases the comparison in favor of the MMM models somewhat, as this requires us to look at the length of the ground truth as part of the input chunking procedure. Also, our test set is contained in MMM's training set, but there is no reasonable way to avoid this as the MMM models were trained on the full Lakh MIDI dataset. (We wanted a diverse and well-randomized test set, and the Lakh MIDI dataset is the only publicly-available dataset we are aware of that fits this bill.)

We evaluate models with standard metrics: Note F_1 [35], average pitch class histogram entropy difference [19, 36], and average groove similarity [19, 36]. Note F_1 measures how closely the generated notes match, on a note-for-note basis, the notes in the ground truth. (For our purposes, a generated note matches a note n in the ground

	Real Music	Our Model	MMM
1st place	**66**	32	27
Avg rank	**1.664**	2.032	2.304

p-values	Our Model	MMM
MMM	0.0239	-
Real Music	0.0034	$2.3 \cdot 10^{-5}$

Table 2. Subjective results from our listening test.

truth if and only if its onset tick, measure, pitch, and track match exactly those of n.) The other metrics measure how well certain higher-level statistics of the generated notes match those of real music. For pitch class histogram entropy calculations, drums are ignored. Each metric is computed on a per-example basis, and then for each model, task, and metric, the 2500 results are averaged to give the results in Table 1. For fairness of groove similarity comparison, we use a denominator of 48 for all models. (This is reasonable, as our models and the MMM models both effectively have 48 possible note onset positions per 4/4 measure.) For our model, "-MP" indicates that the examples were evaluated without `<mono>` or `<poly>` tokens present.

For each row of Table 1, we perform a Wilcoxon signed rank test [37] with Holm-Bonferroni correction [38]. We find significant differences between our model and the MMM models in all 18 rows, with our model outperforming the MMM models in 16 out of 18 rows. The MMM models outperform our model only for pitch class histogram entropy difference for full-track infilling.

Additionally, we find a significant difference in our model's performance when `<mono>` and `<poly>` tokens are included in prompts in 11 out of 18 rows. All significant differences favor including these tokens, suggesting that the development of additional user controls (as in [11]) would be a useful line of future work.

Finally, a Wilcoxon rank sum test [37] reveals significant differences ($p < 0.01$) in 8-bar versus 16-bar results for our model in five out of nine comparisons. All significant differences favor the 16-bar results, emphasizing the importance of training on longer measure slices. However, we never observe a significant difference in 8-bar versus 16-bar results for track infilling, suggesting that larger context windows generally provide no additional useful information for completing this particular task.

Additional experiments not reported here indicate that scaling our training approach (training larger models on more data) is a feasible path for improving model performance on the metrics presented here.

7. SUBJECTIVE EVALUATION OF OUR MODEL

While the results in Section 6 are encouraging, the ground truth may not reflect the only reasonable way to fill in missing notes. To help address this, we conducted a small listening test with 25 participants. We prepared nine examples mostly involving melodic generation. Each example consisted of three 8-measure clips, one of which was real multi-track music. The other two clips were created by removing some tracks from the real music and using our model and MMM to fill those tracks. Participants were shown five of the nine examples at random, and for each example were asked to rank the three clips in order of preference. Results are given in Table 2.

A Wilcoxon signed rank test with Holm-Bonferroni correction reveals significant differences in rankings between all three types of music, with p-values given in Table 2. In this test we see a clear preference for real music, and a significant ($p < 0.05$) preference for music generated by our model over music generated by MMM. One expert participant commented that melodies generated by the models were generally more directionless than those in real music, often failing to drive towards a cadence or "payoff." We agree with this assessment, and this is a shortcoming of our model that we hope to address in future work.

8. LIMITATIONS AND RISKS

Our model writes music that is reflective of its training set. Most of our training files are in Western classical, folk, and hymnal styles. While we included in our training set only files marked as being permissively licensed, it is possible that some files were mismarked. It is also theoretically possible for our model to output copyrighted music, even if such music was not present in the training set.

The most common request we have heard from composers to whom we have shown our system is *personalization*. Generally speaking, they do not want systems that write full songs, and they do not want systems that write "generic" music. Rather, they want systems that can suggest ideas in their style. Some small experiments indicate that our finetuned model can be personalized by individuals (by continuing to finetune the model on their own MIDI files) to write in their styles. Low-rank adaptation [39] of our model may also be possible. Personalization is an avenue we would like to explore formally in future work. For now, our code supports training by users, and our model dimensions were chosen carefully to enable this on consumer hardware. A video card with 6 GB of RAM is sufficient to train our released model on examples with input and output lengths of 1024, and 12 GB of RAM is sufficient to train on examples with input and output lengths of 1650. While this can benefit composers who wish to use our system, there is also the risk that our models may be trained by users to impersonate the styles of others.

9. CONCLUSION

We have introduced Composer's Assistant, a system for interactive human-computer composition in the REAPER digital audio workstation. Composer's Assistant performs multi-track MIDI infilling and comes with an easy-to-use interface. We have released our source code, a pretrained model, a finetuned model, and scripts for interacting with our finetuned model in REAPER. Our models were trained only on permissively-licensed MIDI files.

10. ACKNOWLEDGMENT

11. REFERENCES

[1] C. Payne, "MuseNet," openai.com/blog/musenet, 2019.

[2] J. Liu, Y. Dong, Z. Cheng, X. Zhang, X. Li, F. Yu, and M. Sun, "Symphony Generation with Permutation Invariant Language Model," in *Proc. 23rd Int. Society for Music Information Retrieval Conf.*, Bengaluru, India, 2022, pp. 551–558.

[3] 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," in *Int. Conf. Learning Representations*, 2019.

[4] G. Hadjeres, F. Pachet, and F. Nielsen, "DeepBach: a Steerable Model for Bach Chorales Generation," in *Proc. 34th Int. Conf. Machine Learning*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., vol. 70. PMLR, 06–11 Aug 2017, pp. 1362–1371.

[5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is All you Need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017.

[6] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer," *Journal of Machine Learning Research*, vol. 21, no. 140, pp. 1–67, 2020.

[7] J. Ens and P. Pasquier, "MMM : Exploring Conditional Multi-Track Music Generation with the Transformer," *arXiv preprint arXiv: 2008.06048*, 2020.

[8] ——, "Flexible Generation with the Multi-Track Music Machine," in *Extended Abstracts for the Late-Breaking Demo Session of the 21st Int. Society for Music Information Retrieval Conf.*, Montréal, Canada, 2020.

[9] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language Models are Unsupervised Multitask Learners," 2019. [Online]. Available: https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf

[10] C.-Z. A. Huang, T. Cooijmans, A. Roberts, A. Courville, and D. Eck, "Counterpoint by Convolution," in *Proc. 18th Int. Society for Music Information Retrieval Conf.*, Suzhou, China, 2017, pp. 211–218.

[11] R. Guo, I. Simpson, C. Kiefer, T. Magnusson, and D. Herremans, "MusIAC: An Extensible Generative Framework for Music Infilling Applications with Multi-level Control," in *Artificial Intelligence in Music, Sound, Art and Design. EvoMUSART 2022. Lecture Notes in Computer Science*, T. Martins, N. Rodríguez-Fernández, and S. M. Rebelo, Eds., vol. 13221. Springer, Cham, 2022, pp. 341–356.

[12] A. Roberts, J. Engel, C. Raffel, C. Hawthorne, and D. Eck, "A Hierarchical Latent Vector Model for Learning Long-Term Structure in Music," in *Proc. 35th Int. Conf. Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. PMLR, 10–15 Jul 2018, pp. 4364–4373.

[13] G. Mittal, J. Engel, C. Hawthorne, and I. Simon, "Symbolic Music Generation with Diffusion Models," in *Proc. 22nd Int. Society for Music Information Retrieval Conf.*, online, 2021, pp. 468–475.

[14] S. Wei, G. Xia, Y. Zhang, L. Lin, and W. Gao, "Music Phrase Inpainting Using Long-Term Representation and Contrastive Loss," in *IEEE Int. Conf. Acoustics, Speech and Signal Processing*, 2022, pp. 186–190.

[15] A. Pati, A. Lerch, and G. Hadjeres, "Learning to Traverse Latent Spaces for Musical Score Inpainting," in *Proc. 20th Int. Society for Music Information Retrieval Conf.*, Delft, The Netherlands, 2019, pp. 343–351.

[16] K. Chen, C. Wang, T. Berg-Kirkpatrick, and S. Dubnov, "Music SketchNet: Controllable Music Generation via Factorized Representations of Pitch and Rhythm," in *Proc. 21st Int. Society for Music Information Retrieval Conf.*, Montréal, Canada, 2020, pp. 77–84.

[17] C. Benetatos and Z. Duan, "Draw and Listen! A Sketch-Based System for Music Inpainting," *Trans. Int. Society for Music Information Retrieval*, vol. 5, no. 1, pp. 141–155, 2022.

[18] G. Hadjeres and L. Crestel, "The Piano Inpainting Application," *arXiv preprint arXiv: 2107.05944*, 2021.

[19] C.-J. Chang, C.-Y. Lee, and Y.-H. Yang, "Variable-Length Music Score Infilling via XLNet and Musically Specialized Positional Encoding," in *Proc. 22nd Int. Society for Music Information Retrieval Conf.*, online, 2021, pp. 97–104.

[20] Y.-S. Huang and Y.-H. Yang, "Pop Music Transformer: Beat-Based Modeling and Generation of Expressive Pop Piano Compositions," in *Proc. 28th ACM Int. Conf. Multimedia*, ser. MM '20. New York,

---

NY, USA: Association for Computing Machinery, 2020, p. 1180–1188. [Online]. Available: https://doi.org/10.1145/3394171.3413671

[21] W.-Y. Hsiao, J.-Y. Liu, Y.-C. Yeh, and Y.-H. Yang, "Compound Word Transformer: Learning to Compose Full-Song Music over Dynamic Directed Hypergraphs," in *35th AAAI Conf. Artificial Intelligence*, 2021, pp. 178–186.

[22] A. Roberts, C. Kayacik, C. Hawthorne, D. Eck, J. Engel, M. Dinculescu, and S. Nørly, "Magenta Studio: Augmenting Creativity with Deep Learning in Ableton Live," in *Proc. Int. Workshop on Musical Metacreation (MUME)*, 2019.

[23] T. Bazin and G. Hadjeres, "NONOTO: A Model-agnostic Web Interface for Interactive Music Composition by Inpainting," in *Proc. 10th Int. Conf. Computational Creativity*, 2019.

[24] J. Phang, Y. Zhao, and P. J. Liu, "Investigating Efficiently Extending Transformers for Long Input Summarization," *arXiv preprint arXiv: 2208.04347*, 2022.

[25] N. Shazeer, "GLU Variants Improve Transformer," *arXiv preprint arXiv: 2002.05202*, 2020.

[26] Y. Tay, M. Dehghani, S. Abnar, H. W. Chung, W. Fedus, J. Rao, S. Narang, V. Q. Tran, D. Yogatama, and D. Metzler, "Scaling Laws vs Model Architectures: How does Inductive Bias Influence Scaling?" *arXiv preprint arXiv: 2207.10551*, 2022.

[27] Y. Tay, M. Dehghani, J. Rao, W. Fedus, S. Abnar, H. W. Chung, S. Narang, D. Yogatama, A. Vaswani, and D. Metzler, "Scale Efficiently: Insights from Pretraining and Finetuning Transformers," in *Int. Conf. Learning Representations*, 2022.

[28] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2019, vol. 32, pp. 8024–8035.

[29] 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. Le Scao, S. Gugger, M. Drame, Q. Lhoest, and A. Rush, "Transformers: State-of-the-Art Natural Language Processing," in *Proc. 2020 Conf. Empirical Methods in Natural Language Processing: System Demonstrations*. Association for Computational Linguistics, Oct. 2020, pp. 38–45.

[30] A. Holtzman, J. Buys, L. Du, M. Forbes, and Y. Choi, "The Curious Case of Neural Text Degeneration," in *Int. Conf. Learning Representations*, 2020.

[31] 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 *Proc. 20th Int. Society for Music Information Retrieval Conf.*, Delft, The Netherlands, 2019.

[32] Y. Wu, J. Gardner, E. Manilow, I. Simon, C. Hawthorne, and J. Engel, "The Chamber Ensemble Generator: Limitless High-Quality MIR Data via Generative Modeling," *arXiv preprint arXiv:2209.14458*, 2022.

[33] C. Raffel, "The Lakh MIDI Dataset v0.1," https://colinraffel.com/projects/lmd/.

[34] ——, "Learning-Based Methods for Comparing Sequences, with Applications to Audio-to-MIDI Alignment and Matching," Ph.D. dissertation, 2016.

[35] J. Gardner, I. Simon, E. Manilow, C. Hawthorne, and J. Engel, "MT3: Multi-Task Multitrack Music Transcription," in *Int. Conf. Learning Representations*, 2022.

[36] 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 *Proc. 21st Int. Society for Music Information Retrieval Conf.*, Montréal, Canada, 2020, pp. 142–149.

[37] F. Wilcoxon, "Individual Comparisons by Ranking Methods," *Biometrics Bulletin*, vol. 1, no. 6, pp. 80–83, 1945.

[38] S. Holm, "A Simple Sequentially Rejective Multiple Test Procedure," *Scandinavian Journal of Statistics*, vol. 6, no. 2, pp. 65–70, 1979.

[39] E. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "LoRA: Low-Rank Adaptation of Large Language Models," in *Int. Conf. Learning Representations*, 2022.