ENCODING MUSICAL STYLE WITH TRANSFORMER AUTOENCODERS

Anonymous authors
Paper under double-blind review

ABSTRACT

We consider the problem of learning high-level controls over the global structure of sequence generation, particularly in the context of symbolic music generation with complex language models. In this work, we present the Transformer autoencoder, which aggregates encodings of the input data across time to obtain a global representation of style from a given performance. We show it is possible to combine this global embedding with other temporally distributed embeddings, enabling improved control over the separate aspects of performance style and and melody. Empirically, we demonstrate the effectiveness of our method on a variety of music generation tasks on the MAESTRO dataset and an internal, 10,000+ hour dataset of piano performances, where we achieve improvements in terms of log-likelihood and mean listening scores as compared to relevant baselines.

1 INTRODUCTION

There has been significant progress in generative modeling, particularly with respect to creative applications such as art and music (Oord et al., 2016; Engel et al., 2017b; Ha & Eck, 2017; Huang et al., 2019a; Payne, 2019). As the number of generative applications increase, it becomes increasingly important to consider how users can interact with such systems, particularly when the generative model functions as a tool in their creative process (Engel et al., 2017a; Gillick et al., 2019) To this end, we consider how one can learn high-level controls over the global structure of a generated sample. We focus on symbolic music generation, where Music Transformer (Huang et al., 2019b) is the current state-of-the-art in generating high-quality samples that span over a minute in length.

The challenge in controllable sequence generation is the fact that Transformers (Vaswani et al., 2017) and their variants excel as language models or in sequence-to-sequence tasks such as translation, but it is less clear as to how they can: (1) *learn* and (2) *incorporate* global conditioning information at inference time. This contrasts with traditional generative models for images such as the variational autoencoder (VAE) (Kingma & Welling, 2013) or generative adversarial network (GAN) (Goodfellow et al., 2014) which typically incorprate global conditioning as part of their training procedure (Sohn et al., 2015; Sønderby et al., 2016; Isola et al., 2017; Van den Oord et al., 2016).

In this work, we introduce the Transformer autoencoder, where we aggregate encodings across time to obtain a holistic representation of the performance style. We show that this learned global representation can be incorporated with other forms of structural conditioning in two ways. First, we show that given a performance, our model can generate performances that are similar in style to the provided input. Then, we explore different methods to combine melody and performance representations to harmonize a melody in the style of the given performance. In both cases, we show that combining both global and fine-scale encodings of the musical performance allows us to gain better control of generation, separately manipulating both the style and melody of the resulting sample.

Empirically, we evaluate our model on two datasets: the publicly-available MAESTRO (Hawthorne et al., 2019) dataset, and an internal dataset of piano performances transcribed from 10,000+ hours of audio (Anonymous for review). We find that the Transformer autoencoder is able to generate not only performances that sound similar to the input, but also accompaniments of melodies that follow a given style, as shown through both quantitative and qualitative experiments as well as a user listening study. In particular, we demonstrate that our model is capable of adapting to a particular musical style *even in the case* where we have one single input performance.

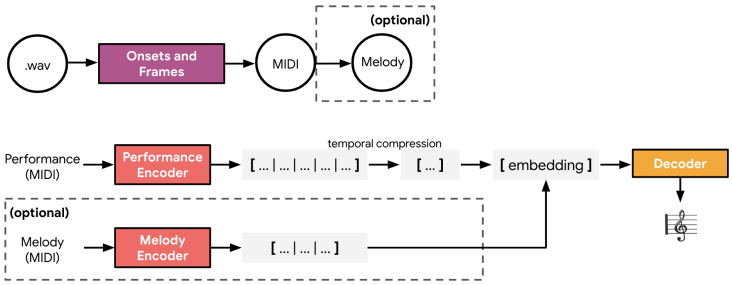


Figure 1: A flowchart of the Transformer autoencoder. We first transcribe the `.wav` data files into MIDI using the Onsets and Frames framework, then encode them into performance representations to use as input. The output of the performance encoder is aggregated across time and (optionally) combined with a melody embedding to produce a representation of the entire performance, which is then used by the Transformer decoder at inference time.

2 Preliminaries

2.1 Data Representation for Music Generation

The MAESTRO (Hawthorne et al., 2019) dataset consists of over 1,100 classical piano performances, where each piece is represented as a MIDI file. The internal performance dataset consists of over 10,000 hours of piano performances transcribed from audio (Anonymous for review). In both cases, we represent music as a sequence of discrete tokens, effectively formulating the generation task as a language modeling problem. The performances are encoded using the vocabulary as described in (Oore et al., 2018), which captures expressive dynamics and timing. This performance encoding vocabulary consists of 128 `note_on` events, 128 `note_off` events, 100 `time_shift` events representing time shifts in 10ms increments from 10ms to 1s, and 32 quantized `velocity` bins representing the velocity at which the 128 `note_on` events were played.

2.2 Music Transformer

We build our Transformer autoencoder from Music Transformer, a state-of-the-art generative model that is capable of generating music with long-term coherence (Huang et al., 2019b). While the original Transformer uses a self-attention mechanism that operates over absolute positional encodings of each token in a given sequence (Vaswani et al., 2017), Music Transformer replaces this with *relative* attention (Shaw et al., 2018), which allows the model to keep better track of regularity based on event orderings and periodicity in the performance. Huang et al. (2019b) propose a novel algorithm for implementing relative self-attention that is significantly more memory-efficient, enabling the model to generate musical sequences over a minute in length. For more details regarding the self-attention mechanism and Transformers, we refer the reader to (Vaswani et al., 2017; Parmar et al., 2018).

3 Conditional Generation with the Transformer Autoencoder

3.1 Model Architecture

We leverage the standard encoder and decoder stacks of the Transformer as a foundation for our model, with minor modifications that we outline below.

Transformer Encoder: For both the performance and melody encoder networks, we use the Transformer's stack of 6 layers which are each comprised of a: (1) multi-head relative attention mechanism; and a (2) position-wise fully-connected feed-forward network. The performance encoder takes as input the event-based performance encoding of an input performance, while the melody encoder learns an encoding of the melody which has been extracted from the input performance. Depending on the music generation task, which we elaborate upon in Section 3.2, the encoder output(s) are fed into the Transformer decoder. Figure 1 describes the way in which the encoder and decoder networks are composed together.

Transformer Decoder: The decoder shares the same structure as the encoder network, but with an additional multi-head attention layer over the encoder outputs. At each step of generation, the decoder takes in the output of the encoder, as well as each new token that was previously generated.

The model is trained end-to-end with maximum likelihood. That is, for a given sequence x of length n, we maximize $\log p_\theta(x) = \sum_{i=1}^{n} \log p_\theta(x_i|x_{<i})$ with respect to the model parameters θ.

3.2 Conditioning Mechanism

Performance Conditioning and Bottleneck For this task, we aim to generate samples that sound "similar" to a conditioning input performance. We incorporate a bottleneck in the output of the Transformer encoder in order to prevent the model from simply memorizing the input (Baldi, 2012). Thus, as shown in Figure 1, we mean-aggregate the performance embedding across the time dimension in order to learn a global representation of style. This mean-performance embedding is then fed into the autoregressive decoder, where the decoder attends to this global representation in order to predict the appropriate target. Although this bottleneck may be undesirable in sequence transduction tasks where the input and output sequences differ (e.g. translation), we find that it works well in our setting where we require the generated samples to be *similar in style* to the input sequence.

Melody & Performance Conditioning Next, we synthesize any given melody in the style of a *different* performance. Although the setup shares similarities to that of the melody conditioning problem in (Huang et al., 2019b), we note that we also provide a conditioning performance signal, which makes the generation task more challenging. During training, we follow an internal procedure to extract melodies from performances in the training set, quantize the melody to a 100ms grid, and encode it as a sequence of tokens that uses a different vocabulary than the performance representation. We then use two distinct Transformer encoders (each with the same architecture) as in Section 3.1 to separately encode the melody and performance inputs. The melody and performance embeddings are combined to use as input to the decoder.

We explore various ways of combining the intermediate representations: (1) `sum`, where we add the performance and melody embeddings together; (2) `concatenate`, where we concatenate the two embeddings separated with a `stop` token; and (3) `tile`, where we tile the performance embedding across every dimension of time in the melody encoding. In all three cases, we work with the mean-aggregated representation of the input performance. We find that different approaches work better than others on some dataets, a point which we elaborate upon in Section 5.

Input Perturbation In order to encourage the encoded performance representations to generalize across various melodies, keys, and tempos, we draw inspiration from the denoising autoencoder (Vincent et al., 2008) as a means to regularize the model. For every target performance from which we extract the input melody, we provide the model with a *perturbed* version of the input performance as the conditioning signal. We allow this "noisy" performance to vary across two axes of variation: (1) `pitch`, where we artificially shift the overall pitch either down or up by 6 semitones; and (2) `time`, where we stretch the timing of the performance by at most 5%. In our experiments, we find that this augmentation procedure leads to samples that sound more pleasing (Oore et al., 2018). We provide further details on the augmentation procedure in Appendix A.

4 Kernel Evaluation Metric

Although a variety of different metrics have been proposed to quantify both the quality (Engel et al., 2019) and similarity of musical performances relative to another (Yang & Lerch, 2018; Hung et al., 2019), we note that the development of a proper metric to measure such characteristics in music generation is an open question. We build upon the ideas of (Hung et al., 2019; Jitkrittum et al., 2019) to capture the stylistic features of a given performance.

4.1 Performance Features

First, we construct a representational feature of each performance by aggregating summary statistics derived from the performance encoding: note density, average pitch, average velocity, average

duration, variation of pitch, variation of velocity, and variation of duration.The note density refers to the average number of notes per second in a given performance: a higher note density often indicates a fast-moving piece, while a lower note density correlates with softer, slower pieces. The pitch denotes whether the performance is played in a higher or lower octave, velocity denotes how hard a key is pressed, and the duration describes for how long each note is pressed in a performance.

4.2 Evaluation Metric

To compare the similarities of our constructed feature vectors, we use the Inverse Multiquadric (IMQ) kernel to project the features into an infinite-dimensional space (Tolstikhin et al., 2017; Jitkrittum et al., 2019):

$$k(x, y) = \frac{1}{\sqrt{||x - y||^2 + C^2}}$$

where C is a tunable hyperparameter. We found empirically that the value of C did not have a large effect, and set $C = 10$ for our experiments.

We project all input and generated performances into the same feature space using the IMQ kernel, and use a Maximum Mean Discrepancy (MMD)-like approach to compute the similarity between two performances. Specifically, given a single input performance x and a set of conditionally-generated samples $\mathcal{Y} = \{y\}_{i=1}^{n}$:

$$-\frac{1}{n}\mathbb{E}_{x\sim p(x), y\sim q(y)}[k(x, y)] + \frac{1}{n^2}\mathbb{E}_{y, y'\sim q(y)}[k(y, y')]$$

Then, we pick the most similar performances out of the generated outputs according to this metric. We omit the usual first term in the MMD loss that quantifies the similarity between the input sets when we only have one conditioning input performance. In the case that we have a one-to-one mapping between an input performance and an output sample $x \rightarrow y$, we work with the kernel similarity directly: $k(x, y)$. We demonstrate empirically that this kernel metric identifies the relevant characteristics of interest in our generated performances in Section 5.

5 Experiments

Datasets: We used both MAESTRO (Hawthorne et al., 2019) and our internal dataset (Anonymous for review) for the experimental setup. We used the standard 80/10/10 train/validation/test split from MAESTRO v1.0.0, and augmented the dataset by 10x using pitch shifts of no more than a minor third and time stretches of at most 5%. We note that this augmentation is distinct from the noise-injection procedure referenced in Section 3: the data augmentation merely increases the size of the initial dataset, while the perturbation procedure operates *only* on the input performance signal. The internal dataset did not require any additional augmentation.

Experimental Setup: We implemented the model in the Tensor2Tensor framework (Vaswani et al., 2017), and used the default hyperparameters for training: 0.2 learning rate with 8000 warmup steps, `rsqrt_decay`, 0.2 dropout, and early stopping for GPU training. For TPU training, we use AdaFactor with the `rsqrt_decay` and learning rate warmup steps to be 10K. We adopt many of the hyperparameter configurations from (Huang et al., 2019b), where we reduce the query and key hidden size to half the hidden size, use 8 hidden layers, use 384 hidden units, and set the maximum relative distance to consider to half the training sequence length for relative global attention. We set the maximum sequence length to be 2048 tokens, and a filter size of 1024. We provide additional details on the model architectures and hyperparameter configurations in Appendix C.

5.1 Log-Likelihood Evaluation

As expected, we find that the Transformer autoencoder framework with the encoder output bottleneck outperforms other baselines. In Tables 1 and 2, we see that all conditional model variants outperform their unconditional counterparts. Interestingly, we find that for the melody & performance model, different methods of combining the embeddings work better for different datasets. For example, `concatenate` led to the lowest NLL for the internal dataset, while `sum` outperformed all other variants for MAESTRO. We report NLL values for both MAESTRO and the internal dataset for the perturbed-input model variants in Appendix B.

Model variation	MAESTRO	Internal
Unconditional model with rel. attention (Huang et al., 2019b)	1.840	1.49
Performance autoencoder with rel. attention (ours)	**1.799**	**1.384**

Table 1: Note-wise test NLL on the MAESTRO and internal datasets, with event-based representations of lengths $L = 2048$. We exclude the performance autoencoder baseline (no aggregation) as it memorized the data (NLL $= 0$). Conditional models outperformed their unconditional counterparts.

Model variation	MAESTRO	Internal
Melody-only Transformer with rel. attention (Huang et al., 2019b)	1.786	1.302
Melody & performance autoencoder with rel. attention, sum (ours)	**1.706**	1.275
Melody & performance autoencoder with rel. attention, concat (ours)	1.713	**1.237**
Melody & performance autoencoder with rel. attention, tile (ours)	1.709	1.248

Table 2: Note-wise test NLL on the MAESTRO and internal datasets with melody conditioning, with event-based representations of lengths $L = 2048$. We note that `sum` worked best for MAESTRO, while `concatenate` outperformed all other baselines for the internal dataset.

5.2 Similarity Evaluation

We use our similarity metric to evaluate whether using a conditioning signal in both the (a) performance autoencoder and (b) melody & performance autoencoder produces samples that are more similar in style to the conditioning inputs from the evaluation set relative to other baselines.

First, we sample 500 examples from the evaluation set and use them as conditioning signals to generate one sample for each input. Then, we compare each conditioning signal to: (1) the generated sample and (2) an unconditional sample. We compute the similarity metric as defined in Section 4 pairwise and take the average over 500 examples. As shown in Table 3, we find that the performance autoencoder generates samples that have 36% higher similarity to the conditioning input as compared to the unconditional baseline.

Similarity Metric	MAESTRO	Internal
Performance autoencoder (ours)	**0.0840**	**0.0944**
Unconditional baseline (Huang et al., 2019b)	0.0455	0.0694
Melody & performance autoencoder (ours)	0.147	0.157
Melody-only Transformer (Huang et al., 2019b)	0.110	0.145
Performance-only autoencoder (ours)	**0.172**	**0.161**
Unconditional baseline (Huang et al., 2019b)	0.005	0.138

Table 3: Average similarities for both the performance and melody & performance autoencoders.

For the melody & performance autoencoder, we sample 717*2 distinct performances – we reserve one set of 717 for conditioning performance styles, and the other set of 717 we use to extract melodies in order to synthesize in the style of a *different* performance. We compare the melody & performance autoencoder to 3 different baselines: (1) one that is conditioned only on the melody (Melody-only); (2) conditioned only on performance (Performance-only); and (3) an unconditional language model. Interestingly, we find that the kernel evaluation metric is more sensitive to the performance style than melodic similarity. Table 3 demonstrates that the Melody-only autoencoder suffers without the performance conditioning, while the Performance-only model performs best. The Melody & performance autoencoder performs comparably to the best model.

5.3 INTERPOLATIONS

5.3.1 PERFORMANCE AUTOENCODER

In this experiment, we test whether the performance autoencoder can successfully interpolate between different performances. First, we sample 1000 performances from the internal test set (100 for MAESTRO, due to its smaller size), and split this dataset in half. The first half we reserve for the original starting performance, which we call "performance A", and the other half we reserve for the end performance, denoted as "performance B." Then we use the performance encoder to encode performance A into its compressed representation z_A, and do the same for performance B to obtain z_B. For a range $\alpha \in [0, 0.125, \ldots, 0.875, 1.0]$, we sample a new performance perf_{new} that results from decoding $\alpha \cdot z_A + (1 - \alpha) \cdot z_B$. We observe how the similarity metric defined in Section 4 changes between this newly interpolated performance perf_{new} and performances $\{\text{A, B}\}$.

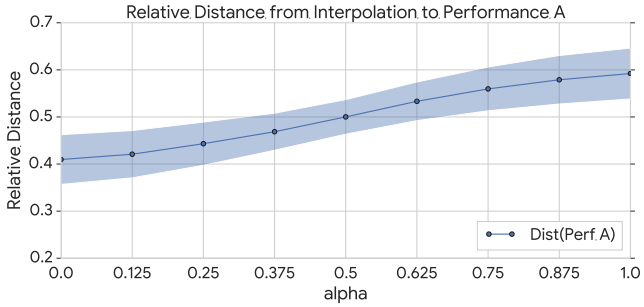
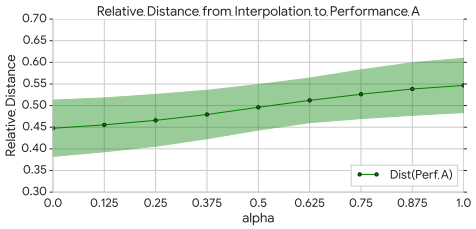
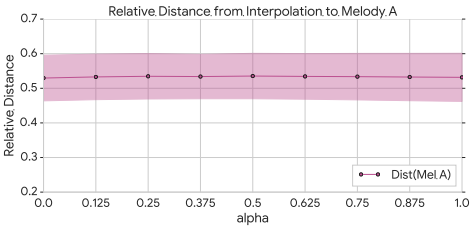


Figure 2: For the internal dataset, the relative distance from performance A ($\alpha = 1$) to the interpolated sample increases as α is slowly increased to 1.0.

Specifically, we compute the kernel similarity using the IMQ kernel between each input performance A and interpolated sample perf_{new} for all 500 samples, and compute the same pairwise similarity for each performance B. We then compute the normalized distance between each interpolated sample and the corresponding performance A or B, which we denote as: $\texttt{rel_distance(perf A)} = 1 - \frac{\texttt{sim_A}}{\texttt{sim_A + sim_B}}$, where $\texttt{sim_A}$ is defined by our kernel similarity metric. We average this distance across all elements in the set and find in Figure 2 that the relative distance between performance A slowly increases as we increase α from 0 to 1, as expected. We note that it is not possible to conduct this interpolation study with non-aggregated baselines, as we cannot interpolate across variable-length embeddings. We find that a similar trend holds for MAESTRO as in Figure 3(a).



(a) Relative distance to performance A



(b) Relative distance to melody A

Figure 3: For the internal dataset, relative distance from performance A ($\alpha = 1$) as α is slowly increased to 1.0 while the conditioned melody remains fixed. As in (b), we note that the relative distance to the fixed conditioning melody with respect to a random performance remains fixed while the interpolation is conducted between performances A and B, which shows that we can control for elements of style and melody separately.

5.3.2 MELODY & PERFORMANCE AUTOENCODER

We conduct a similar study as above, now with the melody & performance autoencoder. For this study, we hold out 716 unique melody-performance pairs (where the melody is not derived from the same performance) from the internal evaluation dataset and 50 examples from MAESTRO. We then

interpolate across the different performances, while keeping the conditioning melody input the same across the interpolations.

As shown in Figure 3(a), we find that a similar trend holds as in the performance autoencoder: the newly-interpolated samples show that the relative distance between performance A increases as we increase the corresponding value of α. We note that the interpolation effect is slightly lower than that of the previous section, particularly because the interpolated sample is also dependent on the melody that it is conditioned on. Interestingly, in Figure 3(b), we note that the relative distance between the input performance from which we derived the original melody remains fairly constant across the interpolation procedure. This suggests that we are able to factorize out the two sources of variation and that varying the axis of the input performance keeps the variation in melody constant.

5.4 Human Evaluation

We also conducted listening studies to evaluate the perceived effect of performance and melody conditioning on the generated output. Using models trained on the internal dataset, we conducted two studies: one for performance conditioning, and one for melody and performance conditioning.

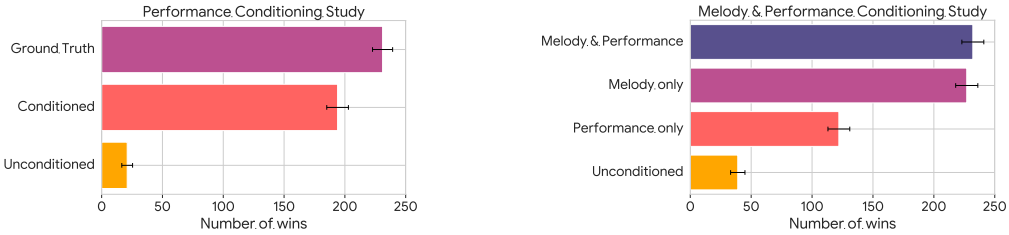


Figure 4: Results of our listening studies, showing the number of times each source won in a pairwise comparison. Black error bars indicate estimated standard deviation of means.

For performance conditioning, we presented participants with a 20s performance clip from the evaluation dataset that we used as a conditioning signal. We then asked them to listen to two additional 20s performance clips and use a Likert scale to rate which one sounded most similar in style to the conditioning signal. The sources the participants rated included "Ground Truth" (a different snippet of the same sample used for the conditioning signal), "Conditioned" (output of the Performance Autoencoder), and "Unconditioned" (output of unconditional model). For this study, 492 ratings were collected, with each source involved in 328 pair-wise comparisons.

For melody and performance conditioning, we presented participants with a 20s performance clip from the evaluation dataset and a 20s melody from a different piece in the evaluation dataset that we used as our conditioning signals. We then asked them to listen to two additional 20s performance clips and use a Likert scale to rate which sounded most like the conditioning melody played in the style of the conditioning performance. The sources the participants rated included "Melody & Performance" (output of the Melody-Performance Autoencoder), "Melody only" (output of a model conditioned only on the melody signal), "Performance only" (output of a model conditioned only on the performance signal), and "Unconditioned" (output of an unconditional model). For this study, 714 ratings were collected, with each source involved in 357 pair-wise comparisons.

Figure 4 shows the number of comparisons in which each source was selected as being most similar in style to the conditioning signal. A Kruskal-Wallis H test of the ratings showed that there is at least one statistically significant difference between the models: $\chi^2(2) = 332.09,\ p < 0.05\ (7.72\text{e}{-73})$ for melody conditioning and $\chi^2(2) = 277.74,\ p < 0.05\ (6.53\text{e}{-60})$ for melody and performance conditioning. A post-hoc analysis using the Wilcoxon signed-rank test with Bonferroni correction showed that there were statistically significant differences between all pairs of the performance study with $p < 0.05/3$ and all pairs of the performance and melody study with $p < 0.05/6$ except between the "Melody only" and "Melody & Performance" models ($p = 0.0894$).

These results demonstrate that the performance conditioning signal has a clear effect on the generated output. In fact, the effect was sufficiently robust that in the 164 comparisons between "Ground Truth" and "Conditioned", participants said they had a preference for "Conditioned" 58 times.

Even though the results between "Melody-only" and "Melody & Performance" are close, this study demonstrates that conditioning with both melody and performance results in outputs that are better at achieving the goal than conditioning on performance alone, and they are competitive with melody-only conditioning, despite the model having to deal with the complexity of incorporating both conditioning signals. In fact, we find quantitative evidence that human evaluation is more sensitive to melodic similarity, as the "Performance-only" model performs worst – a stark contrast to the results from the kernel evaluation metric in Section 5.2.

We provide several audio examples demonstrating the effectiveness of these conditioning signals in the online supplement at `http://bit.ly/2l14pYg`. Our qualitative findings from the audio examples and interpolations, coupled with the quantitative results from the similarity metric and the listening test which capture different aspects of the synthesized performance, support the finding that the Melody & Performance autoencoder offers significant control over the generated samples.

6 Related Work

Sequential autoencoders: Building on the wealth of autoencoding literature (Hinton & Salakhutdinov, 2006; Salakhutdinov & Hinton, 2009; Vincent et al., 2010), our work bridges the gap between the traditional sequence-to-sequence framework (Sutskever et al., 2014), their recent advances with various attention mechanisms (Vaswani et al., 2017; Shaw et al., 2018; Huang et al., 2019b), and sequential autoencoders. Though (Wang & Wan, 2019) propose a Transformer-based conditional VAE for story generation, the self-attention mechanism is shared between the encoder and decoder. Most similar to our work is that of (Kaiser & Bengio, 2018), which uses a Transformer decoder and a discrete autoencoding function to map an input sequence into a discretized, compressed representation. We note that this approach is complementary to ours, where a similar idea of discretization may be applied to the output of our Transformer encoder. The MusicVAE (Roberts et al., 2018) is a sequential VAE with a hierarchical recurrent decoder, which learns an interpretable latent code for musical sequences that can be used during generation time. This work builds upon (Bowman et al., 2015) that uses recurrence and an autoregressive decoder for text generation. Our Transformer autoencoder can be seen as a deterministic variant of the MusicVAE, with a complex self-attention mechanism based on relative positioning in both the encoder and decoder to capture more expressive features of the data at both the local and global scale.

Controllable generations using representation learning: There has also been a wide range of recent work on controllable generations, where we focus on the music domain. (Engel et al., 2017a) proposes to constrain the latent space of unconditional generative models to sample with respect to some defined attributes, whereas we explicitly define our conditioning signal in the data space and learn a global representation of its style during training. The Universal Music Translation network aims to translate music across various styles, but we note that it is not directly comparable to our approach as they work with raw audio waveforms (Mor et al., 2018). Both (Meade et al., 2019) and the recent MuseNet (Payne, 2019) generate music based on user preferences, but adopt a slightly different approach: the models are specifically trained with labeled tokens (e.g., composer and instrumentation) as conditioning information to the input, while our Transformer autoencoder's global style representation is learned in an unsupervised way.

7 Conclusion

We proposed our Transformer autoencoder for conditional music generation, a sequential autoencoder model which utilizes an autoregressive Transformer encoder and decoder for improved modeling of musical sequences with long-term structure. We show that this model allows users to easily adapt the outputs of their generative model using even a single input performance. Through experiments on MAESTRO and an internal dataset, we demonstrate both quantitatively and qualitatively that our model generates samples that sound similar in style to a variety of conditioning signals relative to baselines. For future work, it would be interesting to explore other training procedures such as variational techniques or few-shot learning approaches to account for situations in which the input signals are from slightly different data distributions than the training set.

REFERENCES

Pierre Baldi. Autoencoders, unsupervised learning, and deep architectures. In *Proceedings of ICML workshop on unsupervised and transfer learning*, pp. 37–49, 2012.

Samuel R Bowman, Luke Vilnis, Oriol Vinyals, Andrew M Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. *arXiv preprint arXiv:1511.06349*, 2015.

Jesse Engel, Matthew Hoffman, and Adam Roberts. Latent constraints: Learning to generate conditionally from unconditional generative models. *arXiv preprint arXiv:1711.05772*, 2017a.

Jesse Engel, Cinjon Resnick, Adam Roberts, Sander Dieleman, Mohammad Norouzi, Douglas Eck, and Karen Simonyan. Neural audio synthesis of musical notes with wavenet autoencoders. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1068–1077. JMLR. org, 2017b.

Jesse Engel, Kumar Krishna Agrawal, Shuo Chen, Ishaan Gulrajani, Chris Donahue, and Adam Roberts. Gansynth: Adversarial neural audio synthesis. *arXiv preprint arXiv:1902.08710*, 2019.

Jon Gillick, Adam Roberts, Jesse Engel, Douglas Eck, and David Bamman. Learning to groove with inverse sequence transformations. *arXiv preprint arXiv:1905.06118*, 2019.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pp. 2672–2680, 2014.

David Ha and Douglas Eck. A neural representation of sketch drawings. *arXiv preprint arXiv:1704.03477*, 2017.

Curtis Hawthorne, Andriy Stasyuk, Adam Roberts, Ian Simon, Cheng-Zhi Anna Huang, Sander Dieleman, Erich Elsen, Jesse Engel, and Douglas Eck. Enabling factorized piano music modeling and generation with the MAESTRO dataset. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=r1lYRjC9F7`.

Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

Cheng-Zhi Anna Huang, Tim Cooijmans, Adam Roberts, Aaron Courville, and Douglas Eck. Counterpoint by convolution. *arXiv preprint arXiv:1903.07227*, 2019a.

Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Ian Simon, Curtis Hawthorne, Noam Shazeer, Andrew M. Dai, Matthew D. Hoffman, Monica Dinculescu, and Douglas Eck. Music transformer: Generating music with long-term structure. In *International Conference on Learning Representations*, 2019b. URL `https://openreview.net/forum?id=rJe4ShAcF7`.

Hsiao-Tzu Hung, Chung-Yang Wang, Yi-Hsuan Yang, and Hsin-Min Wang. Improving automatic jazz melody generation by transfer learning techniques. *arXiv preprint arXiv:1908.09484*, 2019.

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1125–1134, 2017.

Wittawat Jitkrittum, Patsorn Sangkloy, Muhammad Waleed Gondal, Amit Raj, James Hays, and Bernhard Schölkopf. Kernel mean matching for content addressability of gans. *arXiv preprint arXiv:1905.05882*, 2019.

Łukasz Kaiser and Samy Bengio. Discrete autoencoders for sequence models. *arXiv preprint arXiv:1801.09797*, 2018.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Nicholas Meade, Nicholas Barreyre, Scott C Lowe, and Sageev Oore. Exploring conditioning for generative music systems with human-interpretable controls. *arXiv preprint arXiv:1907.04352*, 2019.

Noam Mor, Lior Wolf, Adam Polyak, and Yaniv Taigman. A universal music translation network. *arXiv preprint arXiv:1805.07848*, 2018.

Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

Sageev Oore, Ian Simon, Sander Dieleman, Douglas Eck, and Karen Simonyan. This time with feeling: learning expressive musical performance. *Neural Computing and Applications*, pp. 1–13, 2018.

Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Łukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. *arXiv preprint arXiv:1802.05751*, 2018.

Christine Payne. Musenet, 2019. URL `https://openai.com/blog/musenet/`.

Adam Roberts, Jesse Engel, Colin Raffel, Curtis Hawthorne, and Douglas Eck. A hierarchical latent vector model for learning long-term structure in music. *arXiv preprint arXiv:1803.05428*, 2018.

Ruslan Salakhutdinov and Geoffrey Hinton. Deep boltzmann machines. In *Artificial intelligence and statistics*, pp. 448–455, 2009.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. *arXiv preprint arXiv:1803.02155*, 2018.

Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. In *Advances in neural information processing systems*, pp. 3483–3491, 2015.

Casper Kaae Sønderby, Tapani Raiko, Lars Maaløe, Søren Kaae Sønderby, and Ole Winther. Ladder variational autoencoders. In *Advances in neural information processing systems*, pp. 3738–3746, 2016.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pp. 3104–3112, 2014.

Ilya Tolstikhin, Olivier Bousquet, Sylvain Gelly, and Bernhard Schoelkopf. Wasserstein auto-encoders. *arXiv preprint arXiv:1711.01558*, 2017.

Aaron Van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. Conditional image generation with pixelcnn decoders. In *Advances in neural information processing systems*, pp. 4790–4798, 2016.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pp. 1096–1103. ACM, 2008.

Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(Dec):3371–3408, 2010.

Tianming Wang and Xiaojun Wan. T-cvae: Transformer-based conditioned variational autoencoder for story completion. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pp. 5233–5239. International Joint Conferences on Artificial Intelligence Organization, 7 2019. doi: 10.24963/ijcai.2019/727. URL `https://doi.org/10.24963/ijcai.2019/727`.

Li-Chia Yang and Alexander Lerch. On the evaluation of generative models in music. *Neural Computing and Applications*, pp. 1–12, 2018.

APPENDIX

A FURTHER DETAILS ON INPUT PERTURBATION PROCEDURE

We elaborate upon the perturbation procedure as follows, which *only* applies to the melody & performance conditioning music generation task. First, we take a target performance that we would like our model to predict and extract the corresponding melody from this performance. We use this "clean" melody as part of our conditioning signal. Then, we modify the conditioning performance by either shifting the pitch up or down 6 semitones and stretching the timing by $\pm$ 5%. Then for each new data point during training, a single noise injection procedure is randomly sampled from the cross product of all possible combinations of 12 pitch shift values and 4 time stretch values (evaluated in intervals of 2.5%). At test time, the data points are left unperturbed.

B NLL EVALUATION FOR "NOISY" MODEL

Below, we provide the note-wise test NLL on the MAESTRO and internal dataset with melody conditioning, where the conditioning performance is perturbed by the procedure outlined in Section 3.

Model variation	MAESTRO	Internal Dataset
Noisy Melody TF autoencoder with relative attention, sum	1.721	1.248
Noisy Melody TF autoencoder with relative attention, concat	1.719	1.249
Noisy Melody TF autoencoder with relative attention, tile	1.728	1.253

Table 4: Note-wise test NLL on the MAESTRO and Internal Dataset piano performance datasets with melody conditioning, with event-based representations of lengths $L = 2048$.

C MODEL ARCHITECTURE AND HYPERPARAMETER CONFIGURATIONS

We mostly use the default Transformer architecture as provided in the Tensor2Tensor framework, such as 8 self-attention heads as listed in the main text, and list the slight adjustments we made for each dataset below:

C.1 MAESTRO

For the MAESTRO dataset, we follow the hyperparameter setup of (Huang et al., 2019b):

1. num hidden layers = 6
2. hidden units = 384
3. filter size = 1024
4. maximum sequence length = 2048
5. maximum relative distance = half the hidden size
6. dropout = 0.1

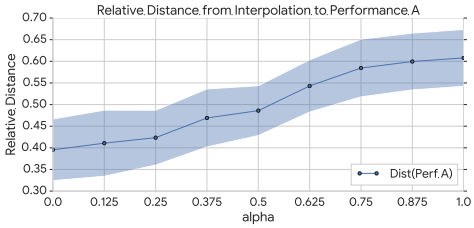
C.2 INTERNAL DATASET

For the internal dataset, we modify the number of hidden layers to 8 and slightly increase the level of dropout.

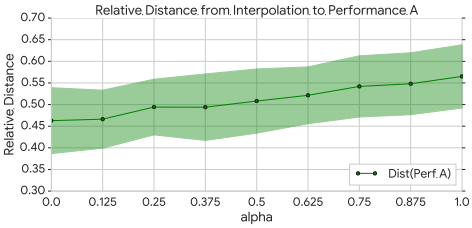
1. num hidden layers = 8
2. hidden units = 384
3. filter size = 1024
4. maximum sequence length = 2048
5. maximum relative distance = half the hidden size
6. dropout = 0.15

D INTERPOLATIONS DETAILS

Interpolation relative distance results for the (a) performance and (b) melody & performance Transformer autoencoders for the MAESTRO dataset.



(a) Relative distance from interpolated sample to the original starting performance.

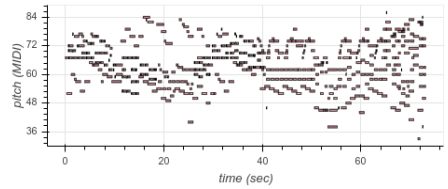


(b) Relative distance from the interpolated sample to the original melody, which is kept fixed.

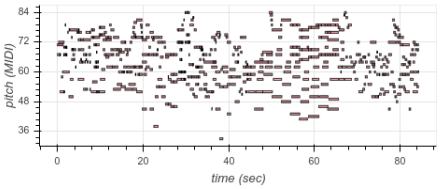
Figure 5: The distance to the original performance increases as the value of α increases in (a), as expected. In (b), we see that there is a very slight increase in the relative distance to the original melody during the interpolation procedure.

E INTERNAL DATASET PERFORMANCE INTERPOLATIONS

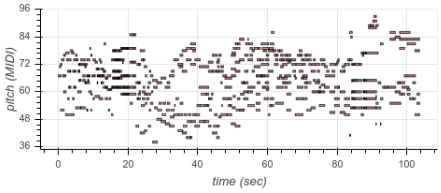
Here, we provide piano rolls demonstrating the effects of latent-space interpolation for the internal dataset, for both the (a) performance and (b) melody & performance Transformer autoencoder. For similar results in MAESTRO as well as additional listening samples, we refer the reader to the online supplement: `http://bit.ly/2l14pYg`.



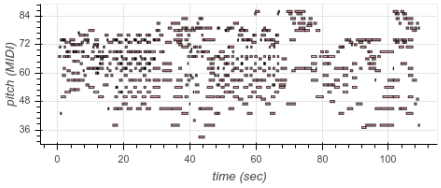
(a) Original starting performance



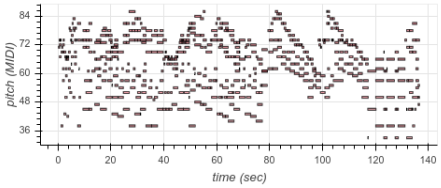
(b) $\alpha = 0.125$



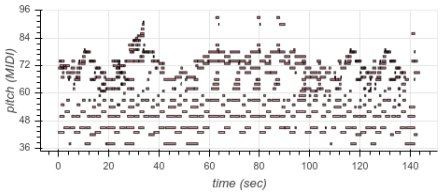
(c) $\alpha = 0.375$



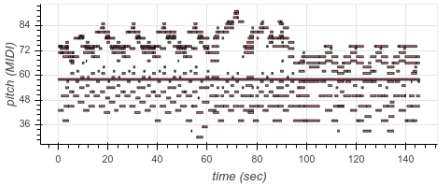
(d) $\alpha = 0.5$



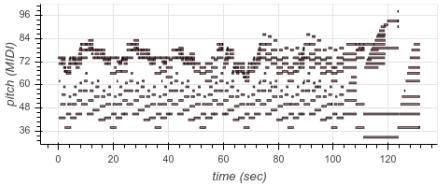
(e) $\alpha = 0.625$



(f) $\alpha = 0.875$



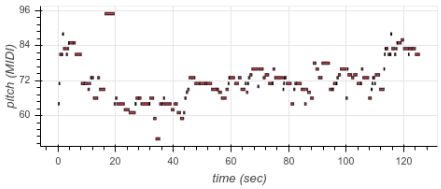
(g) $\alpha = 1.0$ (reconstruction)



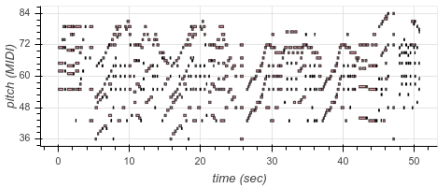
(h) Original final performance

Figure 6: Interpolation of a starting performance (a) from the internal dataset to a final performance (h), with the coefficient α controlling the level of interpolation between the latent encodings between the two performances.

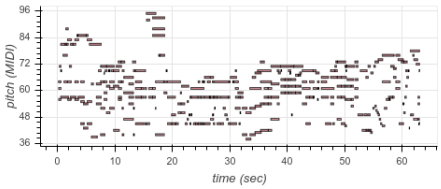
F INTERNAL DATASET MELODY INTERPOLATION



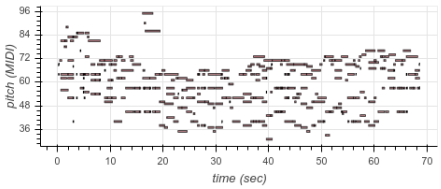
(a) Original melody



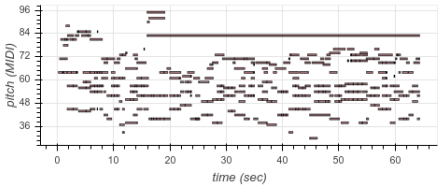
(b) Original starting performance



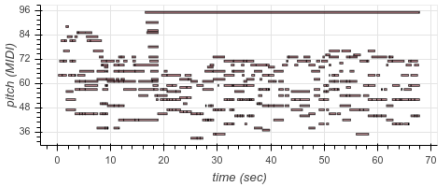
(c) $\alpha = 0$ (reconstruction)



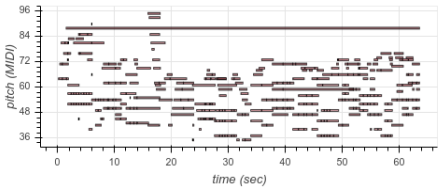
(d) $\alpha = 0.125$



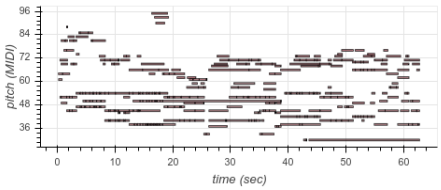
(e) $\alpha = 0.375$



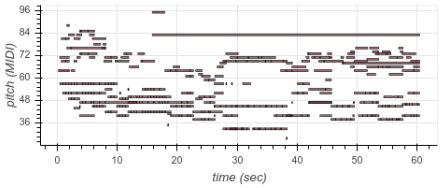
(f) $\alpha = 0.5$



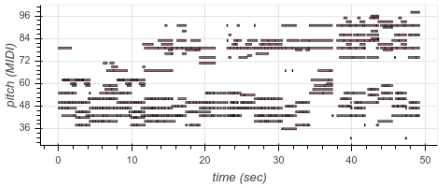
(g) $\alpha = 0.625$



(h) $\alpha = 0.875$



(i) $\alpha = 1.0$ (reconstruction)



(j) Original final performance

Figure 7: Interpolation of a starting performance (b) from the internal dataset to a final performance (j), with the coefficient α controlling the level of interpolation between the latent encodings between the two performances. The original conditioning melody (a) is kept fixed throughout the interpolation.