Compositional Steering of Music Transformers

Halley Young[1,2], Vincent Dumoulin[3], Pablo S. Castro[3], Jesse Engel[2] and Cheng-Zhi Anna Huang[3]

[1]University of Pennsylvania, USA
[2]Google Brain, USA
[3]Google Brain, Canada

Abstract

Musical composition is a combinatorial art where composers extend sequences by choosing from a vast set of possible *feature combinations* that yield the compositions their distinctive qualities. Increasingly, composers are using generative models, such as music transformers, for crafting their pieces. Unfortunately, for composers to "steer" these models to satisfy their qualitative features typically requires retraining (which can be prohibitively expensive); further, existing models are unable to deal with arbitrary combinations of features at scale. In this paper we build on lightweight fine-tuning methods, such as prefix tuning and bias tuning, to propose a novel contrastive loss that enables us to steer music transformers over arbitrary combinations of logical features, with a relatively small number of extra parameters. We provide both quantitative and qualitative evaluations of our method which demonstrate its efficacy with respect to existing methods, as well as a case-study where our method was used to compose long-form musical pieces. Musical examples are available for listening online. [1]

Keywords

creativity, co-creativity, human-AI co-creation, music generation, controllable generative models, compositionality

1. Introduction

Musicians and novices are increasingly experimenting with generative models in music making [44, 19, 18]. Yet generative models are often not trained to support human objectives (such as controllability) but to maximize proxy metrics that are easy to automate (such as likelihood of data according to a model), making it difficult for users to steer these models towards expressing users' musical intentions [44, 19, 32]. In the language domain, recent research in making generative models (such as large language models) more controllable and useful in real-world applications have focused on ways of adapting unconditioned language models to perform well on conditioned tasks they were not initially trained on. Methods such as fine-tuning, side-tuning, bias-tuning, and prompt or prefix tuning have emerged as lead candidates for such tasks such as steering the sentiment of or words mentioned in a sentence [2, 28, 27, 8]. However, such tasks are normally presented in a non- or minimally-compositional approach (possibly controlling for two orthogonal variables, such as sentiment and topic [25], but rarely more than that).

In contrast, in the domain of music generation, sequence continuation is inherently a highly compositional problem: the user likely has many aspects of the output they would like to control, such as speed, dynamics, harmony, or texture, each of which can be decomposed into multiple sub-features. Furthermore, the relationship between these features and the output is more diffuse than in the NLP settings: while in examples such as [25], individual words indicate the different conditions being satisfied, in music the entire sequence of tokens are used in evaluating a single feature (for instance, average pitch over a span of time depends on all tokens representing that span). This makes compositionality even more challenging than in the text generation setting. On the other hand, music is a sequential domain where there are clear logical features which can be examined, such as average dynamics or number of notes per second. This further motivates using music as a test-bed for highly compositional tasks involving simultaneously steering an autoregressive model towards several particular desired attributes according to different classifiers.

As a more immediate motivation, consider the following scenario: a composer wants to sample from the pretrained Music Transformer [20] to complete a musical phrase. In addition to the overall musical quality of the continuation, they want it to stay in key *and* switch to using block chords (i.e. a few notes played together at once). Repeatedly sampling continuations from the model and cherry-picking a good sample (*rejection sampling*) would be very labor-intensive, but assuming they can compute binary features for "stays in key" and "uses block chords", the composer could sample a large number of continuations and cherry-pick from the smaller set of continuations which exhibit all features (thereby delegating

Joint Proceedings of the ACM IUI Workshops 2022, March 2022, Helsinki, Finland
EMAIL: halleyy@seas.upenn.edu (H. Young); annahuang@google.com (C. A. Huang)

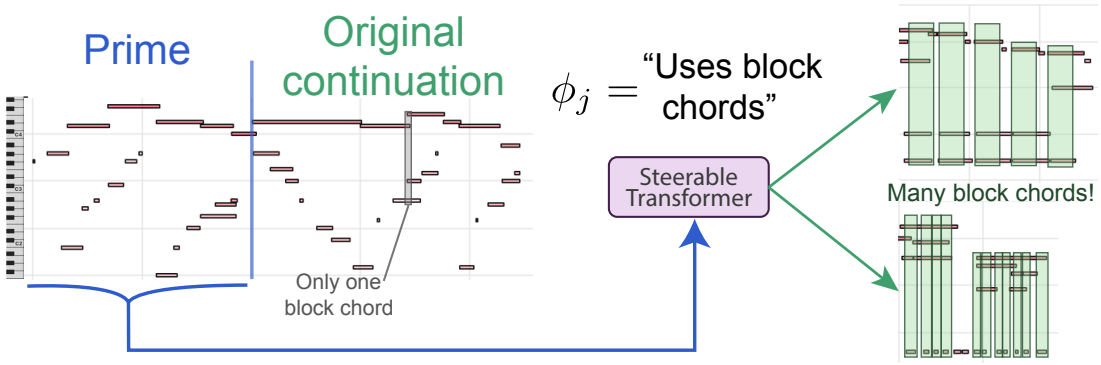

Figure 1: Showcasing a prime from the test set steered in two different ways. The original continuation has arpeggiation (the steep rising and falling lines) throughout. The right piano rolls were steered with a logical feature that checks for many simultaneous notes (block chords). In particular, the bottom right piano roll shows that the model learned to "exploit" the logical feature by repeating a low note, while still sounding musical. Hear this example online.

part of the accept/reject step). While significantly less labor-intensive, this solution could potentially require sampling enormous amounts of continuations if, for instance, the requested features significantly differ from those of the priming sequence (such that it and the desired continuation form an unlikely sequence according to the generative model). In other words, the pre-trained model could be a poor proposal distribution for some applications.

On the ML side of this work, we are interested in developing better proposal distributions through an adaptation approach which can steer the generative model towards continuations which i) are significantly more likely to exhibit the requested feature, and ii) exhibit a satisfactory musical quality. The approach should be able to accommodate a large number of features without adding significant memory or computation overhead. We achieve this by making features composable, making it possible to steer features independently and also multiple features at once. Figure 1 shows that when using a pre-trained transformer model augmented with a relatively small number of additional parameters, we are able to steer towards arbitrary logical music features and achieve realistic music generation simply by sampling directly from the model. In contrast, the same unconditional transformer model fails to produce any examples satisfying those features even when generating a hundred samples.

Our approach can be used to support human-AI co-creation, where musicians can compose on the level of musical "features" as opposed to notes. Musicians can prototype the high-level "shape" of the music by specifying how various features change throughout the piece, and in turn steer and curate music transformers to creatively fill in the details. With our method, a composer can control both the short-chunk features, and the long-form structure, by chaining together chunks steered in different directions (using different combinations of features), while maintaining long-term coherence (by leveraging the transformers' full self-attention receptive field). Listen to examples on this page [1] for longer steered examples, and compositions semi-automatically generated using our feature tuning approach in a musician-directed way.

2. Problem formulation

Music Transformer is an autoregressive language model which decomposes the joint probability of a sequence of tokens $x_1, \ldots, x_N$ (where $x_n \in \mathcal{K}$, and $\mathcal{K}$ is a set of categorical tokens) into

$$p(\mathbf{x} = x_1, \ldots, x_N) = p(x_1) \prod_{n=2}^{N} p(x_n \mid x_1, \ldots, x_{n-1}). \tag{1}$$

It leverages a common modeling approach which represents the conditional probabilities $p(x_n \mid x_1, \ldots, x_{n-1})$ using a neural network [3]. As its name implies, Music Transformer uses a Transformer network architecture [48]. Each token x_n is first mapped to a real-valued embedding $\mathbf{e}_n$ (for instance using a lookup table), then the network maps each sequence $\mathbf{e}_1, \ldots, \mathbf{e}_{n-1}$ to a probability distribution for the value of x_n over the elements of $\mathcal{K}$.

Sequence continuation in an autoregressive language model works by repeatedly sampling from its distribution over the next token given the previous tokens. Start-

[1] Listen to musical examples at https://storage.googleapis.com/composing-features/index.html

ing from some priming sequence $\mathbf{x}_p = (x_0, \ldots, x_M)$, we first sample $y_{M+1} \sim p(\cdot \mid x_0, \ldots, x_M)$, then $y_{M+2} \sim p(\cdot \mid x_0, \ldots, x_M, y_{M+1})$, and so on, until the end of the continued sequence $\mathbf{x}_c = (y_{M+1}, \ldots, y_N)$. [2] Many downstream tasks can be cast as sequence continuation problems, including the steerable music generation problem investigated in this work.

We are given a set of features $\Phi = \{\phi_j\}_{j=1}^{J}, \phi_j \in \mathcal{K}^N \to \{0, 1\}$. Each ϕ_j takes the value 1 if a prime-continuation pair exhibits that feature (which we note as $(\mathbf{x}_p, \mathbf{x}_c) \models \phi_j$), and 0 otherwise. Note that the features must take both prime and continuation sequences as input, since some continuation features may be relative to the priming sequence (e.g. "significantly higher pitch"). Our true objective with respect to feature ϕ_j is to steer the model towards a distribution which maximizes

$$\mathbb{E}_{\mathbf{x}_c \mid \mathbf{x}_p}\left[(\mathbf{x}_p, \mathbf{x}_c) \models \phi_j\right] \tag{2}$$

while maintaining musicality. This objective is non-differentiable because $(\mathbf{x}_p, \mathbf{x}_c) \models \phi_j$ is a non-differentiable satisfiability criterion.

In addition to the single-feature problem, we also consider the problem of *composed* features $\hat{\Phi}$, i.e.

$$(\mathbf{x}_p, \mathbf{x}_c) \models \hat{\Phi} \quad \equiv \quad \bigwedge_{\phi_j \in \hat{\Phi} \subseteq \Phi} (\mathbf{x}_p, \mathbf{x}_c) \models \phi_j, \tag{3}$$

to account for scenarios where a user is interested in steering the model towards multiple features (such as in the "stays in key" and "uses block chords" scenario discussed in the introduction).

3. Proposed approach

We start by describing our proposed approach in the single-feature case and later on explain how we adapt it to the compositional case.

3.1. Likelihood-based training

While approaches using reinforcement learning—such as KL-regularized deep Q-learning [21]—could be used to overcome the non-differentiability problem, in this work we consider a proxy loss in the form of the negative log-likelihood

$$l = -\log p_\theta(\mathbf{x}_c \mid \mathbf{x}_p), \tag{4}$$

which we use in two ways:

1. **Positively**: given a prime–continuation pair $(\mathbf{x}_p, \mathbf{x}_c) \models \phi_j$, we find an adaptation θ_j of the model's parameters θ that minimizes $l_{\checkmark}$ (we use the symbol $\checkmark$ to denote the fact that l is computed using the correct parameters θ_j). By using prime–continuation examples that sound musical, we ensure that the steered model stays musically grounded.
2. **Negatively**: we can also take advantage of other features ϕ_i for which $(\mathbf{x}_p, \mathbf{x}_c) \not\models \phi_i$, by maximizing $l_{\times}$ (we use the symbol $\times$ to denote the fact that l is computed using the incorrect parameters θ_i).

The positive case corresponds to maximum-likelihood training. Additionally, we can exploit the intuition that the adapted parameters θ_j should "explain" the prime–continuation pair $(\mathbf{x}_p, \mathbf{x}_c) \models \phi_j$ better than θ_i (for some feature ϕ_i for which $(\mathbf{x}_p, \mathbf{x}_c) \not\models \phi_i$) or θ (the non-adapted model parameters, with a corresponding loss $l_\emptyset$). In other words, we can maximize the probability of choosing θ_j over θ_i and θ by minimizing a contrastive loss of the form

$$-\log\left(\frac{p_{\theta_j}(\mathbf{x}_c \mid \mathbf{x}_p)}{p_{\theta_j}(\mathbf{x}_c \mid \mathbf{x}_p) + p_{\theta_i}(\mathbf{x}_c \mid \mathbf{x}_p) + p_\theta(\mathbf{x}_c \mid \mathbf{x}_p)}\right) = -\log\left(\frac{e^{-l_{\checkmark}}}{e^{-l_{\checkmark}} + e^{-l_{\times}} + e^{-l_\emptyset}}\right) \tag{5}$$

We propose a loss that interpolates between Equations 4 and 5 using an α coefficient (which is treated as a hyperparameter):

$$\mathcal{L}_j = \alpha \cdot l_{\checkmark} - (1-\alpha) \cdot \log\left(\frac{e^{-l_{\checkmark}}}{e^{-l_{\checkmark}} + e^{-l_{\times}} + e^{-l_\emptyset}}\right) \tag{6}$$

Intuitively, the maximum-likelihood setting should suffice to achieve our adaptation goals, but in practice we find that the approach benefits from the inclusion of negative cases through a contrastive loss term. We tried different α values and found $\alpha = 0.8$ to work well in practice. See Figure 2 for an illustration of the training setup.

3.2. Feature-conditional adaptation

Fine-tuning all model parameters can be prohibitive if the number J of features is large (let alone combinatorially large in the compositional case); however, recent work provides effective and lightweight alternatives:

1. **Prefix tuning** [28] works by prepending learnable task embeddings $\mathbf{e}_{-K}, \ldots, \mathbf{e}_{-1}$ to the priming sequence embeddings $\mathbf{e}_1, \ldots, \mathbf{e}_M$. The loss

[2] To simplify the discussion, we assume a fixed sequence length N, but the explanation applies to sequences of varying lengths as well.

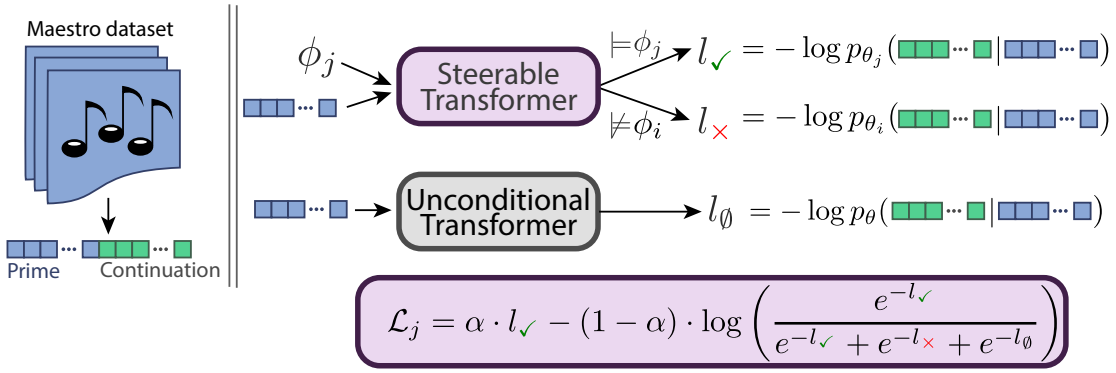

Figure 2: Overall training setup, where the (positive) feature being learned is ϕ_j (with parameters θ_j), and the negative feature is ϕ_i (with parameters θ_i). The parameters θ, used for the unconditional negative log-likelihood, are not learned.

gradient is then backpropagated through the language model and into the task embeddings.

2. **Bias-tuning** [2] works by adapting a small subset $\theta_b \subset \theta$ of the transformer's parameters, namely the biases of its affine transformations. Since these biases amount to a small fraction of the model's parameters, in the case where the number of tasks is relatively small, tuning separate θ_b for each task becomes feasible. We present an extension to bias-tuning where the number of tasks is exponential in the number of total classification functions, using an approach which nevertheless only requires a number of tuned parameters linear in the number of total classification functions.

In practice, while prefix tuning showed promise in the single-feature setting, we were unable to make it work in the compositional setting. We therefore focus our investigation on bias-tuning for the compositional domain.

In the compositional setting, a naive approach requires learning $2^{|\Phi|} - 1$ model adaptations. Instead, we propose to express the adaptation for a composed feature $\hat{\Phi}$ as the combination of the θ_j of its underlying features $\phi_j \in \hat{\Phi}$. More specifically, for bias-tuning we average the adapted biases as

$$\theta_b = \frac{1}{|\hat{\Phi}|} \sum_{\phi_j \in \hat{\Phi}} \theta_{b,j}. \tag{7}$$

Note that we do not simply use the above heuristic to compose feature adaptations *post-hoc*; we *train* the model in a compositional setting (by sampling prime–continuation pairs $(\mathbf{x}_p, \mathbf{x}_c) \models \hat{\Phi}$ for various composed features) so that the single-feature adaptations can learn to work well in conjunction with each other. See Figure 3 for an illustration of our prefix-tuning and bias-tuning setup.

4. Experimental setup

In this section we describe our experimental setup: the musical features we want to enable users to control, the procedure for setting up the training data for adapting the feature-conditional parameters, and the details of the prefix tuning and bias tuning setup.

4.1. Musical and dataset features

Musical features To support users in controlling different aspects of music, such as harmony, texture, speed, and dynamics, we implemented eighteen features (see Appendix A for a complete list).

We included both *absolute* and *relative* features. Absolute features apply only to the continuation, while relative features describe the relationship between the prime and the continuation, such as the *average pitch* in the continuation being *significantly higher* than that in the prime or the *rhythmic density* in the continuation being *significantly lower* than that in the prime. Note that this type of model should work with any feature function which takes in sequences and returns a Boolean. In this paper, we chose examples that have clear musical meanings and are easy to implement.

Dataset features Before adapting model parameters to specific binary musical features, we may first need to "fine-tune" a model to better fit the general musical style of a given dataset (in order to support the scenario where a user brings their own dataset). We refer to these dataset-level features as $\phi_{\mathcal{D}}$.

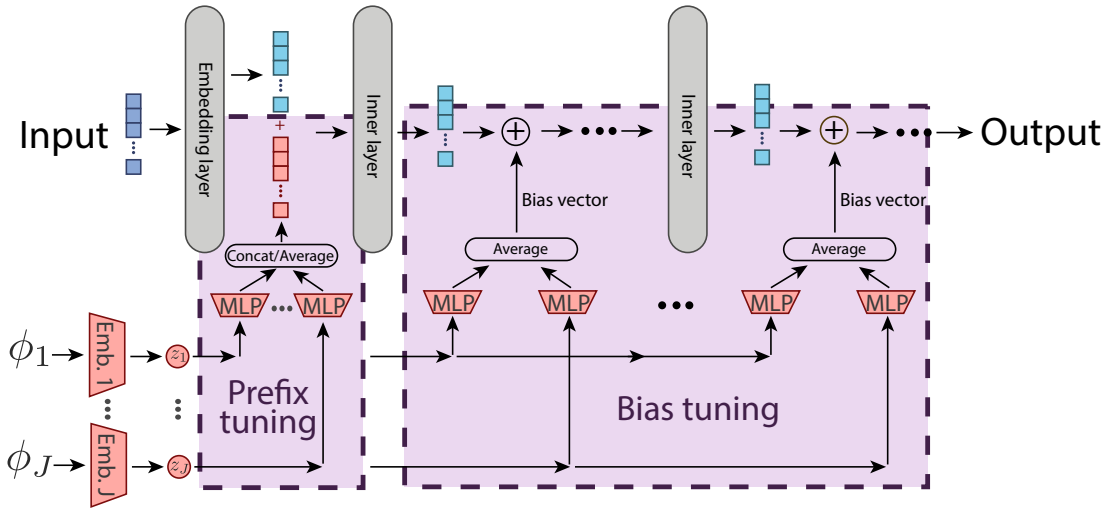

Figure 3: Schematic of how prefix tuning and bias tuning augments the Transformer architecture. Prefix tuning works by prepending learnable embeddings to the the priming sequence, while bias tuning works by adapting the biases in the feed-forward layers of the Transformer. To support steering multiple features at once, these embeddings and biases are averaged before they are added to the original Transformer.

4.2. Training data and setup

The unconditioned model we adopt as a base for adaptation is a pretrained music transformer [3] trained on transcribed YouTube piano performances (where the music was typically more melodic). To mimic the user bringing their own dataset with a distinctive style, we use the Maestro dataset [14], an open-source collection of virtuoso performances of classical piano music.

To prepare the prime–continuation pairs needed for likelihood based training, we serialize the Maestro piano performances into event-based encoding [33] (the same representation that the pretrained music transformer was trained on) and then take random crops of length 200 tokens (which lasts approximately 4 to 20 seconds long). We then split each crop in half, resulting in a prime $\mathbf{x}_p$ and continuation $\mathbf{x}_c$ pair that is each 100 token long.

To prepare training sets for feature-specific adaptations, for each feature $\phi_j \in \Phi$ we take the subset (with replacement) of all the pairs that exhibit the feature (i.e. $(\mathbf{x}_p, \mathbf{x}_c) \models \phi_j$). Note that since all prime–continuation pairs are drawn from the Maestro dataset, we implicitly assume that $(\mathbf{x}_p, \mathbf{x}_c) \models \phi_{\mathcal{D}}$ always holds.

Single-feature setting For the non-contrastive setting, for each feature ϕ_j we minimize the loss introduced in Equation 4 to adapt its parameters θ_j to better fit the set of prime–continuation pairs where $(\mathbf{x}_p, \mathbf{x}_c) \models \phi_j$.

[3] See blog post "Generating Piano Music with Transformer" https://magenta.tensorflow.org/piano-transformer.

In the contrastive setting, we minimize the contrastive loss introduced in Equation 6. For each prime–continuation pair $(\mathbf{x}_p, \mathbf{x}_c) \models \phi_j$ used to compute $l_{\checkmark}$, we also need to select a "negative" case for computing $l_{\times}$. We achieve this by selecting another feature ϕ_i at random, and a prime–continuation pair where both $(\mathbf{x}_p, \mathbf{x}_c) \models \phi_i$ and $(\mathbf{x}_p, \mathbf{x}_c) \not\models \phi_j$ holds.

Compositional setting Training with a contrastive loss yielded more effective models in the single-feature setting (see section 5 for details), hence we adopt the contrastive loss for all of the subsequent experiments in the compositional setting.

To prepare training examples for steering any number of features at once, we first extend Φ so that each feature has a corresponding "negated" feature (e.g. "*has both loud and soft pitches*" vs "*no contrast in dynamics*"), bringing the total number of features from $|\Phi| = 18$ to $2 \cdot |\Phi| = 36$. By definition, each prime–continuation pair exhibits exactly 18 features. When sampling a prime–continuation pair $(\mathbf{x}_p, \mathbf{x}_c)$, we compute $l_{\checkmark}$ with respect to a random subset of those 18 features (with size drawn at random from $U[1, 12]$). We compute $l_{\times}$ by negating some of the sampled features.

4.3. Evaluation metrics

During training time, we minimize a likelihood-based proxy loss to fit the prime–continuations pairs as well as

possible, either non-contrastively or contrastively. However, there is no guarantee that a model trained this way with a lower loss would be more effective in being steered to produce requested musical features. Hence, we propose to evaluate our "downstream" true objective using the following two axes:

- **Sampling efficacy** When steered to generate a particular feature, sampling efficacy is measured as the proportion of generated samples exhibiting the requested feature. As all of our current musical features are implemented as logical functions that return a Boolean value, we can algorithmically compute this.
- **Musical quality** Not only do we want the generated samples to exhibit a requested feature, we also want the musical quality of that sample to be high. To evaluate this, we carried out a listening test with musicians to compare the musical quality of samples generated from the unconditioned model and adapted models (see subsection 6.2 for details).

In the section below, we define how we evaluate sampling efficacy and explain how we break down reporting results according to the "inherent" difficulty in steering.

4.3.1. Sampling efficacy

At test time, what we care about is when a user requests a feature ϕ_j, regardless of what the prime $\mathbf{x}_p$ is, what is the probability that a tuned model can achieve it (i.e. generate continuations $\hat{\mathbf{x}}_c$ that exhibit feature ϕ_j).

Hence, we propose the following *sampling efficacy* (i.e. probability of achievement) as our true evaluation metric to quantify *on average* using rejection sampling, what proportion of a model's generated continuations $\hat{\mathbf{x}}_c$ exhibit feature ϕ_j

$$\mathbb{E}_{\mathbf{x}_p}\mathbb{E}_{\hat{\mathbf{x}}_c|\mathbf{x}_p}\left[(\mathbf{x}_p, \hat{\mathbf{x}}_c) \models \phi_j\right] \tag{8}$$

where the first expectation is under the data distribution (approximated by sampling primes $\mathbf{x}_p$ from a given dataset), and the second is under the model being evaluated (approximated by conditioning on a prime $\mathbf{x}_p$, and then using the model to generate continuations $\hat{\mathbf{x}}_c$).

Steering difficulty Intuitively, given a prime $\mathbf{x}_p$, certain musical features would follow more musically than others. That is, the prime can impact how difficult it is to steer a model to generate continuations $\hat{\mathbf{x}}_c$ that exhibit a certain feature ϕ_j, hence affecting the *sampling efficacy* of a model on that feature.

To address this "confounding factor", we propose to breakdown our analysis of *sampling efficacy* based on if a feature musically follows a prime or not. To approximate this, we check in the dataset (that a model was trained on) if the continuation of a prime carries a feature or not. We group primes in the following two categories

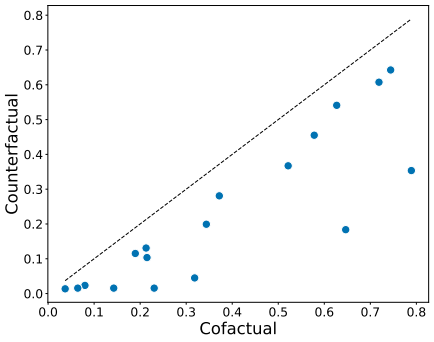

Figure 4: Using the unconditioned model, the probability of achieving a feature is higher when a feature is a cofactual to a prime (x-axis) versus when it is counterfactual to a prime (y-axis). Each blue dot represents a feature (there are 18 features total). All the dots are below the diagonal line, demonstrating that this phenomenon holds for all features.

- **Cofactual prime** A prime $\mathbf{x}_p$ is considered *cofactual* to a feature ϕ_j when the prime has a continuation (in the dataset) that exhibits that feature, that is $(\mathbf{x}_p, \mathbf{x}_c) \models \phi_j$.
- **Counterfactual prime** A prime $\mathbf{x}_p$ is considered *counterfactual* to a feature ϕ_j where $(\mathbf{x}_p, \mathbf{x}_c) \not\models \phi_j$.

The unconditioned music transformer model allows us to establish a *baseline* for "inherently" how difficult or easy it is to achieve a certain feature by priming the model and measuring how often its generated continuation carries that feature.

Figure 4 shows for each feature (blue dot) the probability of achieving it when the unconditioned music transformer was primed with cofactual primes (x-axis) versus counterfactual primes (y-axis). Intuitively, we expect it to be more difficult for a prime that is counterfactual to a feature to produce continuations with that feature, indeed this is what we observe, i.e. in the figure all the blue dots are below the diagonal line. Given this consistency, whenever we report results on *sampling efficacy* (probability of achievement) in the paper, we breakdown our results to compare how methods fare on the hard (counterfactual) versus the easier (cofactual) cases.

4.4. Implementation

Single-feature setting We use a standard prefix tuning model as per [28], associating every feature ϕ_j with a prefix v_j. As in their paper, we used the finding that a relatively low-dimensional embedding space of feature prefixes (200 dimensions) projected to a higher-

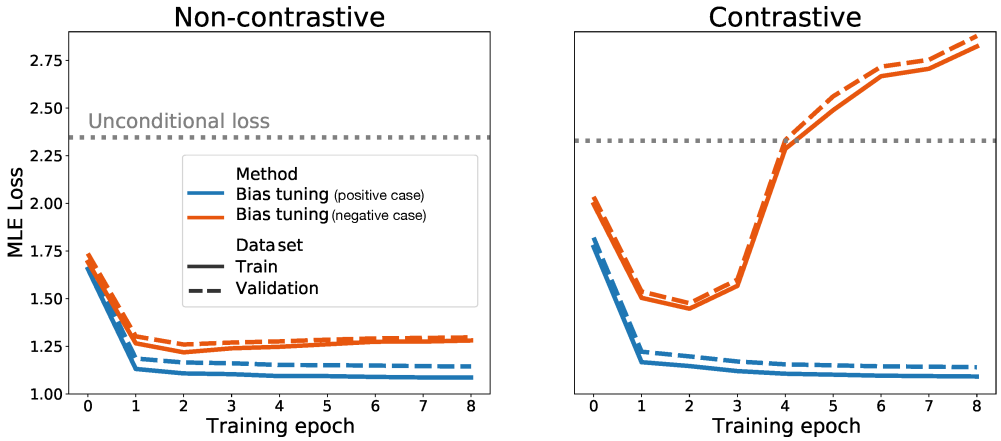

Figure 5: Comparing training losses for bias-tuning (in the single-feature setting), the visually "diverging" curves in the contrastive setting (right) show that this setting is more effective in learning parameters that discriminate between positive (blue curve, $l_{\checkmark}$) and negative cases (orange curve, $l_{\times}$).

dimensional embedding (2048 dimensions) using a shared MLP across feature prefixes worked well. The pre-trained transformer model had no parameters modified, but was changed to accept these 2048-dimensional vectors as four 512-dimensional vectors prepended before the 512-dimensional vectors representing the input tokens (as the transformer's latent embedding was 512 dimensions). The resulting sequence of vectors were masked using causal attention in order to predict $\mathbf{x}_c$.

Compositional setting The loss used in the compositional setting is almost identical to Equation 6, the main difference being in the contrastive loss implementation. The features used to compute $l_{\times}$ are obtained by negating some of the features used to compute $l_{\checkmark}$, and the fraction β of negated features is used to modulate the interpolation coefficient as $\alpha' = \alpha - \min(\alpha, \beta)$. The intuition is that the differences in steering should be more noticeable when comparing feature sets with less overlap.

5. The single-feature setting

We first examine the single-feature setting, before going into the compositional setting. For the experiments in this section, we focus on investigating if our proposed contrastive loss (introduced in Equation 6) is beneficial. We find that indeed it is, that training with a contrastive loss not only enables learning of parameters that better "explain" musical features in a more "discriminate" fashion, but also results in a more steerable model, i.e. when conditioned on a musical feature, more likely to generate samples that exhibit that feature.

5.1. Likelihood-based training dynamics

Intuitively, the maximum-likelihood setting should suffice to achieve our adaptation goals, but in practice we find that the approach benefits from the inclusion of negative cases through a contrastive loss term (introduced in Equation 6), as demonstrated in Figure 5. The positive case (blue line) corresponds to the maximum-likelihood term, while the negative case (orange line) corresponds to the contrastive loss term. The visually "diverging" training curves in the contrastive setting show that it has learned to "explain" features using very different parameters. That is, when evaluating a prime–continuation pair $(\mathbf{x}_p, \mathbf{x}_c)$ with parameters θ_i adapted for feature ϕ_i that the pair does not exhibit $(\mathbf{x}_p, \mathbf{x}_c) \not\models \phi_i$, the likelihood loss becomes very high.

5.2. Sampling efficacy

In this section, we compare how different methods perform "downstream" on the objective we care about, that is when the user brings a prime and a desired feature, how likely is a model able to generate a continuation that exhibits that feature. The *sampling efficacy* metric (defined in in subsection 6.1) gives on average a method's probability in achieving requested features.

Procedure In the following experiments, to compute a method's *sampling efficacy* on a feature, we take 512 random primes from the feature's validation set, and for each prime generate 10 continuations, and then check what proportion of the continuations satisfy the requested feature. We then average among all features to obtain the average sampling efficacy of a method. This is the cofactual case. While for the counterfactual case, instead

	Prefix tuning		
	Contrastive	Non-contrastive	Unconditioned
Cofactual	**0.58 ± 0.01**	0.54 ± 0.01	0.38 ± 0.01
Counterfactual	**0.39 ± 0.01**	0.33 ± 0.01	0.23 ± 0.01

Table 1
Comparing *sampling efficacy* (i.e. probability of achievement, higher is better), prefixing tuning (especially with a contrastive loss) significantly increases the probability of achieving a requested feature both in the easier cofactual cases and the harder counterfactual cases. The ± gives the 95% confidence intervals.

of sampling primes from a feature's own validation set, we sample the primes at random from another feature's validation set.

Results In Table 1, we see that prefixing tuning (especially with a contrastive loss) significantly increases sampling efficacy over the unconditioned model, both in the easier cofactual cases, the harder counterfactual cases. All of the improvements are of a large margin, for example in the hard counterfactual cases, prefix tuning with a contrastive loss increases the probabiliy of achieving a feature by 70%, and within prefix tuning, the gain from switching from a non-contrastive loss to contrastive loss is 18%.

As sampling efficacy can vary with prime and feature requested, in Figure 6 we enumerate how different methods perform for each of the musical features (18 total). We see prefix tuning (especially with a contrastive loss) yields a higher sampling efficacy (y-axis) for most features. As contrastive loss yielded more effective models in the single-feature setting, we adopt the contrastive loss for all of the subsequent experiments in the compositional setting.

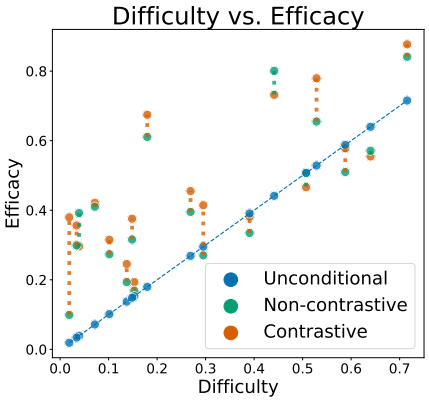

Figure 6: Prefix tuning a model (especially with a contrastive loss) increases its *sampling efficacy* (y-axis) for most of the 18 single features. The colored dots that line up vertically correspond to the same feature. Each feature's *steering difficulty* (x-axis) is quantified as the *sampling efficacy* of the unconditioned model, which gives a baseline of how difficult or easy it is to sample that given feature. Most of the green (non-contrastive) and orange (contrastive) dots are above the diagonal line, meaning prefix tuning increases the probability of achieving them. The vertical dotted orange lines show the improvement in the contrastive setting over the non-contrastive setting.

6. The compositional setting

In the following experiments, we first illustrate how difficult compositional steering is under an unconditioned model. Then, we show quantitatively how our bias-tuning method can enable compositional steering with much higher sampling efficacy, and also qualitatively how it achieves this while also improving on the musical quality of the steered examples (compared to if we were generating from the "tail" of the unconditioned model).

6.1. Sampling efficacy

Unconditioned model Intuitively, we expect it would be ineffective to rely on rejection sampling (on an unconditioned model) to give us samples that include a specific large number of features. Under a naive assumption of independence among features, one would expect that the efficacy of rejection sampling in achieving a total of n features would roughly equal p^n, where p is the probability of a feature being satisfied.

While the above assumption is naive, we do find that in the unconditioned setting, the sampling efficacy of the unconditioned model drops substantially as the number of requested features increases. For example, when the number of requested *cofactual* features is 6, only 0.2% of the time were all features achieved after rejection sampling (see the unconditional line in Figure 7).

Bias tuning With bias tuning, sampling efficacy is more effective overall and less brittle with respect to the number of features. We see in Figure 7 [4] that even though sampling efficacy decreases for all methods when more

[4]Note that the single-feature steering efficacy presented in the compositional setting (Figure 7) is much higher than that in Table 1

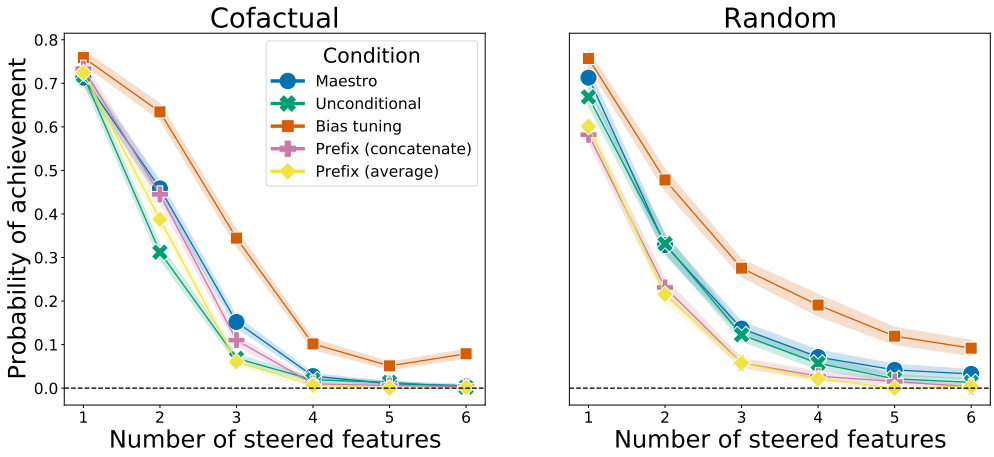

Figure 7: Bias tuning is more effective overall and less brittle with respect to the number of steered features. While we expect sampling efficacy to decrease with more steered features, bias tuning always outperformed all other methods (in both the cofactual and the random case) and still retained a viable probability of achievement even when 6-feature combinations are requested (at 7.9%, meaning 1 in 13 samples exhibits the requested features).

features are requested as expected, bias-tuning remained a viable steering approach even in the 6-feature setting, with a 7.9% probability of achievement versus only 0.2% for the unconditioned model. This meant for the user, using bias-tuning on average 13 samples is needed to arrive at one that exhibits their requested 6 features, while for the unconditioned model it would take 500 samples, making the latter an unfeasible approach.

Comparing the other methods, bias tuning to the Maestro dataset (without tuning for specific features) improves over the unconditioned model slightly (with larger gains in the 2 to 3 feature case in the cofactual setting), and as expected underperforms bias tuning which is tuned to also condition on specific features. Bias tuning outperforms prefix tuning consistently in the compositional setting.

6.2. Musical quality

The above experiments evaluate the effectiveness of methods in achieving the requested features. To evaluate the musical quality of these "successful" generations, we conducted a listening study with musicians to compare the steering of three features between our overall best adaptation approach (i.e. bias tuning) and the baseline of rejection sampling on the unconditional model.

Listening study setup The listening test consists of questions posed as pairwise comparisons of musical examples, where both examples started with the same prime as the first half of the example, while one's continuation was generated by the unconditional model, while another was generated with our bias-tuned model. The ordering within the pairs were randomized. Listeners were asked to rate which one they thought was more musical. To prepare the samples, we randomly picked 120 primes from the test set. For steering each prime, we randomly chose a three-feature set that was present in the dataset as the conditioning features.

Results We asked eight musicians to each rate fifteen pairs. The results show that our bias-tuning approach was much preferred over the unconditional, and the results were statistically significant ($p < 0.0003$). Bias tuning won 63 of the pairwise comparisons, tied for 26 pairs, and lost for 31 pairs. This shows that our bias-tuning approach is not only more effective in steering features, but also produces musically more compelling results.

Discussion It may seem surprising that bias-tuning was able to produce samples that were perceived as more musical than the original expressive unconditional model. We hypothesize that this is because we are essentially sampling from the "tail" of the unconditioned model's distribution when using rejection sampling (i.e. only accepting samples that exhibit the requested feature). Since the unconditional model is not trained to generate specific features, we may have to disregard for example

for the single-feature setting. This is due to difference in the procedure we used in the two settings. In the compositional setting, we sampled features according to how they were distributed in the datasets, while in the single-feature setting we sampled features uniformly at random. Hence, in the compositional setting, features that appear often in the dataset (perhaps typical of the dataset's musical style) is sampled more often. The sampling efficacy is much higher possibly because these feature are also easier to achieve.

99.8% of the samples generated before finding a satisfactory sample (as seen in the 6-feature setting described in subsection 6.1). This resulting distribution is very different from the distribution of the unconditional model without rejection sampling.

We further hypothesize that, via the proxy loss of increasing the likelihood of a set of continuations with a given feature, a bias-tuned model learns more likely ways to satisfy that given feature. Hence, its continuations are not only more likely to be effective at satisfying that feature, but also to do so in a musically likely manner, which is often rated by listeners as more musical too.

7. Case study: Human-AI co-creation

In the previous sections, we evaluated our approach algorithmically, and found that it is quite effective in steering music transformers to generate continuations with multiple requested features present. In this section, we wanted to understand how a generative model with such steerability can be useful in a human-AI co-creation setting. As a preliminary study, one asked one of our co-authors, who was also a composer, to put our model to test by carrying out computer-assisted composition.

User background and creative strategies Our composer had a background in both purely human and algorithmic composition. She experimented with using the tool in several settings. In some settings, her goal was to co-create music of a specific high-level *"vibe"*, which she accomplished through the inclusion of specific features and manually curating the generated samples.

- To achieve *"pleasant but low energy"*, she chose features such as *diatonicism and harmonic stasis, block chords, lack of extreme dynamic change, and a mostly low rhythmic density*. Listen to this example and other *"vibes"* online. [5]
- Inspired by a given prime that had a *"roller-coaster"*-like quality, she wanted to write a piece with a cyclic shape which oscillated between melodic and textural extremes. She accomplished this through the coupling and cycling of features such as *relative and absolute rhythmic density, pitch height, and existence of block chords*.

Seeing that it is possible to steer music transformer to generate specific high-level *"vibes"* through low-level features, our composer was inspired to compose a *theme and variation* where each variation would continue the same prime but with a different *"vibe"* by invoking a different set of low-level features. [6] This allowed her to combine her knowledge of non-automated algorithmic composition to compose a "quasi-algorithmic" suite of variations. Algorithmic composition involves using computational logic to choose notes; instead, in this piece, she used computational logic to choose features.

Composer's reflection The composer's findings were that the tool enabled her on every end of the human-machine spectrum: as a composer who occasionally wants to avoid using any generation tools in their final output, as a composer who wants to co-create in order to minimize both manual coding and manual composing, and as an algorithmic composer. As a composer trying to improve her "manual" composition and analysis skills within specific scenarios (e.g., fast-paced and tense music with chromaticism but also a degree of stasis), and who is used to having to manually search for examples of repertoire having such properties in order to learn from them, the ability to generate a piece of music tailored to a specific scenario forms a surprisingly powerful pedagogical tool which she found that she can leverage when composing by hand.

The workflow of co-creating music made her feel like she was the author (rather than the computer), but allowed her to create piano music of a complexity and virtuosity her piano skills do not currently afford. The composer felt that this tool was even more powerful in the context of algorithmic composition. Her typical workflow in algorithmic composition involves using tools such as SuperCollider [51] to adapt an existing algorithm to generate low-level notes; here she could still use algorithmic thinking, but at a higher level of abstraction. For instance, to compose the suite of variations presented earlier, she wrote a python program to orchestrate which set of musical features are used for each variation. In contrast, while in the past she would invoke a Markov chain to flesh out each variation, here she could invoke a powerful generative model.

In this case study, as the composer was also the creator of this tool, she was in a unique position to explore the full potential of this tool. To make the tool accessible to a broader audience, future work could include "Hello AI"-like [5] exercises to introduce how to compose algorithmically with transformers, akin to the ones found in textbooks on SuperCollider and other algorithmic composition environments.

[5] Listen to different co-created *"vibes"* at https://storage.googleapis.com/composing-features/index.html#vibes.

[6] Listen to a co-created theme and variation at https://storage.googleapis.com/composing-features/index.html.

8. Related Literature

Our work builds on language modeling, further exploring ways to "tune" these models for user control, through "fine-tuning" approaches such as prefix and bias tuning. In particular, we leverage contrastive learning and compositionality to derive a lightweight augmentation for steering large transformer-based language models. Our approach enables human-AI co-creation, enabling users to compose at a higher level of abstraction by specifying features while music transformers fills in the notes. In the following, we provide a brief overview for each of the aforementioned related research areas.

Language models as generative models This work would be impossible without the wealth of research on transformer models for sequence generation tasks. Starting with Vaswani et al. [48], researchers realized that this paradigm enables far more coherent and diverse generation than the recurrent neural networks typically used before [40]. Another milestone was the development of GPT-3, which showed that with sufficient size/training data such models could potentially perform few shot learning [4]. Several subsequent papers, however, demonstrated the inadequacy of few-shot learning for many tasks [37]. A few alternatives to online few-shot learning have since emerged.

"Finetuning" language models Prompt tuning was initially explored ad-hoc in the context of finding ways to produce interesting output by GPT-3, and was formalized by [27]. While originally prompts were designed as tokens from the transformer's vocabulary, subsequent studies generalized them to any embeddings prepended to the input [28]. We extend research into prefix tuning by considering aggregation methods among compositional prefixes. Feature-wise transformations, such as elementwise scaling and/or biasing of features in a network based on side-information (such as labels for a conditioned task), have been applied in a wide variety of problem settings—see [11] for an overview. We draw direct inspiration from BitFit's bias-tuning approach [2] and cast it as a feature-wise transformation approach. By factoring the additive perturbations in Figure 3 into their preceding layers, our bias-tuning implementation can be described as a multi-task variant of BitFit where the parameterizations are tied across features. A key difference is that our feature-specific adaptations are designed to be composable, which to our best knowledge has not yet been explored in the context of large language models—although compositional adaptations using feature-wise multiplicative interactions have been studied in the context of zero-shot image classification [38]. Similarly, side-tuning, or summing task-specific features with general language-model features, has shown significant enhancements in few-shot learning, but is typically not performed compositionally [53].

Contrastive learning Contrastive learning is used in representation learning to train a network which maps "similar" (positive) inputs to nearby representations and "dissimilar" (negative) inputs far away from the positive inputs. See [26] for a theoretical framework and overview. In generative modeling, contrastive divergence [15] was proposed to train Restricted Boltzmann Machines [43], image-to-image translation models [1, 35, 30], and conditional [23] and unconditional [22] generative adverarial networks [12]. Our contrastive formulation differs from previous work in two ways. First, rather than selecting positive and negative examples related to the conditioning signal (musical features in our case) and using the contrastive loss to predict which example "agrees" with the signal, we select positive and negative *conditioning signals* (i.e. different musical features) and use the contrastive loss to predict which conditioning signal explains the prime–continuation pair best. Second, we also treat the absence of a conditioning signal (i.e. the original generative model) as a negative conditioning signal, meaning that we want the model conditioned on the "correct" musical features to explain the prime–continuation pair better than the unconditional model. Other work leveraging language models for multiple tasks include CTRL [25] and Plug-and-Play models [8]. However, CTRL has the disadvantage that it requires knowing the tasks during the training of the large language model, while Plug-and-Play requires multiple passes through the language model, which can be expensive for sufficiently large models.

Controllable generative models for music Advances in sequence modeling [46, 49, 47, 7] has enabled long-form music generation in both the symbolic domain [33, 20, 36, 31, 16] and the audio domain [46, 14, 10, 9]. Similar to language, researchers in music generation have been adapting these language models towards controllable generation, such as by conditioning on one part of a musical piece to complete the rest, such as melody harmonization [41, 29, 6], or more generally arbitrary partial score completion [17, 13]. Representational learning approaches such as autoencoders (AEs) and variational autoencoders (VAEs) have also been used for steering interpolations or transformations along learned latent dimensions, a low-level disentangled attribute-based dimension such as note density [39, 24], or a high-level learned dimension such as energy level in mood that is then realized through its mapping to low-level features such as note-density and rhythm [45], controlling chord progressions and texture independently [42, 50], or rear-

ranging a piece to have increased polyphony or rhythmic density [52].

Human-AI co-creation in music Controllable generative models typically does not offer users full control (i.e. only allows users to specify a small number of low-level or high-level controls, or through an example), while relying on its learned stylistic distribution and/or features encoded from the user-specified template piece to fill in the rest of the musical details [44, 32, 19]. In contrast, traditional constraint satisfaction based music generation systems do not have prior knowledge of the desired stylistic distribution, instead rely on users to specify a large number of musical constraints to guide its search (see [34] for a survey). When using the former systems, users may still feel a lack of agency, while the latter can impose a laborious process. Our approach explores the space in between, allowing users to compose multiple features along different musical dimensions for short chunks (similar to constraint specification), while leveraging pretrained transformers‘ expressiveness to aid users in maintaining coherence in virtuosic long-form composition.

9. Conclusion

We have shown that music transformers can be directed towards a specific generative “task" using some of the same methods as natural language transformers. In addition, we have studied compositionality in this domain. Compositionality (including relatively high levels of compositionality) is critical in the music domain (and other domains) if the user wants control over the output. We establish that compositionality is a hard problem, and propose adaptations of several solutions from the literature (bias-tuning and prefix-tuning) to address these challenges. We find success with bias-tuning, but not with prefix-tuning. While our results are promising, there is clearly significant room for improving on the efficacy of steering a transformer compositionally.

We have provided a preliminary demonstration of how our approach, compositional steering, enables human-AI co-creation, where musicians can compose on the level of musical “features” as opposed to notes. Musicians can prototype the high-level “shape” of the music by specifying how various features change throughout the piece, and in turn steer and curate music transformers to creatively fill in the details. We envision future work to enable end-user machine learning where users can define their own features or provide their own musical examples, and leverage our lightweight compositional bias-tuning approach to learn new controls to steer expressive music transformers compositionally.

References

[1] Kyungjune Baek, Yunjey Choi, Youngjung Uh, Jaejun Yoo, and Hyunjung Shim. 2021. Rethinking the truly unsupervised image-to-image translation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 14154–14163.

[2] Elad Ben-Zaken, Shauli Ravfogel, and Yoav Goldberg. 2021. BitFit: Simple Parameter-efficient Fine-tuning for Transformer-based Masked Language-models. *ArXiv* abs/2106.10199 (2021).

[3] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A neural probabilistic language model. *The journal of machine learning research* 3 (2003), 1137–1155.

[4] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. *Advances in Neural Information Processing Systems* (2020).

[5] Carrie J Cai, Samantha Winter, David Steiner, Lauren Wilcox, and Michael Terry. 2019. "Hello AI": Uncovering the onboarding needs of medical practitioners for human-AI collaborative decision-making. *Proceedings of the ACM on Human-computer Interaction* 3, CSCW (2019), 1–24.

[6] Kristy Choi, Curtis Hawthorne, Ian Simon, Monica Dinculescu, and Jesse Engel. 2020. Encoding musical style with transformer autoencoders. In *International Conference on Machine Learning*. PMLR, 1899–1908.

[7] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. 2020. Rethinking attention with performers. *ICLR* (2020).

[8] Sumanth Dathathri, Andrea Madotto, Janice Lan, Jane Hung, Eric Frank, Piero Molino, J. Yosinski, and Rosanne Liu. 2020. Plug and Play Language Models: A Simple Approach to Controlled Text Generation. *ArXiv* abs/1912.02164 (2020).

[9] Prafulla Dhariwal, Heewoo Jun, Christine Payne, Jong Wook Kim, Alec Radford, and Ilya Sutskever. 2020. Jukebox: A Generative Model for Music. *arXiv preprint arXiv:2005.00341* (2020).

[10] Sander Dieleman, Aaron van den Oord, and Karen Simonyan. 2018. The challenge of realistic music generation: modelling raw audio at scale. In *Ad-*

vances in Neural Information Processing Systems.

[11] Vincent Dumoulin, Ethan Perez, Nathan Schucher, Florian Strub, Harm de Vries, Aaron Courville, and Yoshua Bengio. 2018. Feature-wise transformations. *Distill* (2018). https://doi.org/10.23915/distill.00011 https://distill.pub/2018/feature-wise-transformations.

[12] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. *Advances in neural information processing systems* 27 (2014).

[13] Gaëtan Hadjeres, François Pachet, and Frank Nielsen. 2017. DeepBach: a Steerable Model for Bach Chorales Generation. In *International Conference on Machine Learning*. 1362–1371.

[14] Curtis Hawthorne, Andriy Stasyuk, Adam Roberts, Ian Simon, Cheng-Zhi Anna Huang, Sander Dieleman, Erich Elsen, Jesse Engel, and Douglas Eck. 2019. Enabling Factorized Piano Music Modeling and Generation with the MAESTRO Dataset. In *International Conference on Learning Representations*. https://openreview.net/forum?id=r1lYRjC9F7

[15] Geoffrey E Hinton. 2002. Training products of experts by minimizing contrastive divergence. *Neural computation* 14, 8 (2002), 1771–1800.

[16] Wen-Yi Hsiao, Jen-Yu Liu, Yin-Cheng Yeh, and Yi-Hsuan Yang. 2021. Compound Word Transformer: Learning to Compose Full-Song Music over Dynamic Directed Hypergraphs. *AAAI* (2021).

[17] Cheng-Zhi Anna Huang, Tim Cooijmans, Adam Roberts, Aaron Courville, and Doug Eck. 2017. Counterpoint by Convolution. In *Proceedings of the International Conference on Music Information Retrieval*.

[18] Cheng-Zhi Anna Huang, Curtis Hawthorne, Adam Roberts, Monica Dinculescu, James Wexler, Leon Hong, and Jacob Howcroft. 2019. The Bach Doodle: Approachable music composition with machine learning at scale. *ISMIR* (2019).

[19] Cheng-Zhi Anna Huang, Hendrik Vincent Koops, Ed Newton-Rex, Monica Dinculescu, and Carrie J Cai. 2020. AI Song Contest: Human-AI co-creation in songwriting. *ISMIR* (2020).

[20] 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. 2019. Music Transformer. In *International Conference on Learning Representations*.

[21] Natasha Jaques, Shixiang Gu, Richard E. Turner, and Douglas Eck. 2016. Tuning Recurrent Neural Networks with Reinforcement Learning. *CoRR* abs/1611.02796 (2016). arXiv:1611.02796 http://arxiv.org/abs/1611.02796

[22] Alexia Jolicoeur-Martineau. 2018. The relativistic discriminator: a key element missing from standard GAN. *arXiv preprint arXiv:1807.00734* (2018).

[23] Minguk Kang and Jaesik Park. 2020. Contragan: Contrastive learning for conditional image generation. *arXiv preprint arXiv:2006.12681* (2020).

[24] Lisa Kawai, Philippe Esling, and Tatsuya Harada. 2020. Attributes-aware deep music transformation. In *Proceedings of the 21st international society for music information retrieval conference, ismir*.

[25] Nitish Shirish Keskar, Bryan McCann, Lav R. Varshney, Caiming Xiong, and Richard Socher. 2019. CTRL: A Conditional Transformer Language Model for Controllable Generation. *CoRR* abs/1909.05858 (2019). arXiv:1909.05858 http://arxiv.org/abs/1909.05858

[26] Phuc H Le-Khac, Graham Healy, and Alan F Smeaton. 2020. Contrastive representation learning: A framework and review. *IEEE Access* (2020).

[27] Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The Power of Scale for Parameter-Efficient Prompt Tuning. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

[28] Xiang Lisa Li and Percy Liang. 2021. Prefix-Tuning: Optimizing Continuous Prompts for Generation. *CoRR* abs/2101.00190 (2021). arXiv:2101.00190 https://arxiv.org/abs/2101.00190

[29] Feynman Liang. 2016. BachBot: Automatic composition in the style of Bach chorales. *Masters thesis, University of Cambridge* (2016).

[30] Rui Liu, Yixiao Ge, Ching Lam Choi, Xiaogang Wang, and Hongsheng Li. 2021. Divco: Diverse conditional image synthesis via contrastive generative adversarial network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 16377–16386.

[31] Antoine Liutkus, Ondřej Cıfka, Shih-Lun Wu, Umut Simsekli, Yi-Hsuan Yang, and Gael Richard. 2021. Relative positional encoding for transformers with linear complexity. In *International Conference on Machine Learning*. PMLR, 7067–7079.

[32] Ryan Louie, Andy Coenen, Cheng Zhi Huang, Michael Terry, and Carrie J. Cai. 2020. Novice-AI Music Co-Creation via AI-Steering Tools for Deep Generative Models. *Conference on Human Factors in Computing Systems (CHI)* (2020).

[33] Sageev Oore, Ian Simon, Sander Dieleman, Douglas Eck, and Karen Simonyan. 2020. This time with feeling: Learning expressive musical performance. *Neural Computing and Applications* 32, 4 (2020), 955–967.

[34] François Pachet and Pierre Roy. 2001. Musical harmonization with constraints: A survey. *Constraints* 6, 1 (2001), 7–19.

[35] Taesung Park, Alexei A Efros, Richard Zhang, and Jun-Yan Zhu. 2020. Contrastive learning for unpaired image-to-image translation. In *European Conference on Computer Vision*. Springer, 319–345.

[36] Christine Payne. 2019. MuseNet. https://openai.com/blog/musenet. Accessed: 2020-05-04.

[37] Ethan Perez, Douwe Kiela, and Kyunghyun Cho. 2021. True Few-Shot Learning with Language Models. *CoRR* abs/2105.11447 (2021). arXiv:2105.11447 https://arxiv.org/abs/2105.11447

[38] Senthil Purushwalkam, Maximilian Nickel, Abhinav Gupta, and Marc'Aurelio Ranzato. 2019. Task-driven modular networks for zero-shot compositional learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 3593–3602.

[39] Adam Roberts, Jesse Engel, Colin Raffel, Curtis Hawthorne, and Douglas Eck. 2018. A hierarchical latent vector model for learning long-term structure in music. *ICML* (2018).

[40] Alex Sherstinsky. 2018. Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network. *CoRR* abs/1808.03314 (2018). arXiv:1808.03314 http://arxiv.org/abs/1808.03314

[41] Ian Simon, Dan Morris, and Sumit Basu. 2008. MySong: automatic accompaniment generation for vocal melodies. In *Proceedings of the SIGCHI conference on human factors in computing systems*. ACM.

[42] Ian Simon, Adam Roberts, Colin Raffel, Jesse Engel, Curtis Hawthorne, and Douglas Eck. 2018. Learning a latent space of multitrack measures. *arXiv preprint arXiv:1806.00195* (2018).

[43] P. Smolensky. 1986. *Information Processing in Dynamical Systems: Foundations of Harmony Theory*. MIT Press, Cambridge, MA, USA, 194–281.

[44] Bob L Sturm, Oded Ben-Tal, Una Monaghan, Nick Collins, Dorien Herremans, Elaine Chew, Gaëtan Hadjeres, Emmanuel Deruty, and François Pachet. 2019. Machine learning research that matters for music creation: A case study. *Journal of New Music Research* 48, 1 (2019).

[45] Hao Hao Tan and Dorien Herremans. 2020. Music fadernets: Controllable music generation based on high-level features via low-level feature modelling. *ISMIR* (2020).

[46] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. 2016. WaveNet: A Generative Model for Raw Audio. *arXiv preprint arXiv:1609.03499* (2016).

[47] Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. 2017. Neural discrete representation learning. *NeurIPS* (2017).

[48] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. *CoRR* (2017).

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

[50] Ziyu Wang, Dingsu Wang, Yixiao Zhang, and Gus Xia. 2020. Learning interpretable representation for controllable polyphonic music generation. *ISMIR* (2020).

[51] Scott Wilson, David Cottle, and Nick Collins. 2011. *The SuperCollider Book*. The MIT Press.

[52] Shih-Lun Wu and Yi-Hsuan Yang. 2021. MuseMorphose: Full-Song and Fine-Grained Music Style Transfer with Just One Transformer VAE. *arXiv preprint arXiv:2105.04090* (2021).

[53] Jeffrey O. Zhang, Alexander Sax, Amir Roshan Zamir, Leonidas J. Guibas, and Jitendra Malik. 2019. Side-Tuning: Network Adaptation via Additive Side Networks. *CoRR* abs/1912.13503 (2019). arXiv:1912.13503 http://arxiv.org/abs/1912.13503

A. Musical Features

Note that while some features may appear to be opposites (e.g., "loud" vs "soft"), and while it is true that they are mutually exclusive, in fact it is possible for a sequence to satisfy neither (e.g., if it's in a middle dynamic level).

Absolute features:

1. "Loud" - minimum velocity is greater than 60
2. "Soft" - maximum velocity is less than 60
3. "Has Dynamic Contrast" - Extreme Dynamic Contrast" - The sequence has two notes whose velocities differ by more than 30
4. "Extreme Dynamic Contrast" - The sequence has two notes whose velocities differ by more than 70
5. "All consonances" - the sequence has no dissonances (simultaneous notes with an absolute difference modulo 12 of 1, 2, 10, 11, or 6)
6. "Long sharp dissonance" - the sequence has a sharp dissonance (simultaneous notes being played with a difference of 1 or 11) that lasts for a significant amount of time
7. "Only melody" - only a single note is playing at a time
8. "Few onsets" - when grouping notes according to attack time, there are few groups per second
9. "Many onsets" - when grouping notes according to attack time, there are many groups per second

10. “Blocks of two" - there are groups of two notes being played simultaneously
11. “Larger blocks" - there are blocks of three or more notes being played simultaneously
12. “Within single key" - all notes fit within a single major scale

Relative features:

1. “Significantly lower average pitch than prime"
2. “Significantly higher average pitch than prime"
3. “Significantly higher number of grouped attacks than prime"
4. “Significantly lower number of grouped attacks than prime"
5. “Significantly more notes per second than prime"
6. “Significantly fewer notes per second than prime"
7. “Could fit in the same key as the prime"
8. “Has 2 or more pitch classes (pitch mod 12) which the prime doesn’t have"