# Anticipatory Music Transformer

**Anonymous authors**
**Paper under double-blind review**

## Abstract

We introduce *anticipation*: a method for constructing a controllable generative model of a temporal point process (the event process) conditioned asynchronously on realizations of a second, correlated process (the control process). We achieve this by interleaving sequences of events and controls, such that controls appear following stopping times in the event sequence. This work is motivated by problems arising in the control of symbolic music generation. We focus on infilling control tasks, whereby the controls are a subset of the events themselves, and conditional generation completes a sequence of events given the fixed control events. We train anticipatory infilling models using the large and diverse Lakh MIDI music dataset. These models match the performance of autoregressive models for prompted generation, with the additional capability to perform infilling control tasks, including accompaniment. Human evaluators report that an anticipatory model produces accompaniments with similar musicality to even music composed by humans over a 20-second clip.

## 1 Introduction

Imagine you are given a melody, and asked to compose a harmonizing accompaniment. This melody is an example of a *temporal point process*: a sequence of (musical) events that arrive stochastically at points in time. An accompaniment to this melody is also a temporal point process. The events in the accompaniment are tightly correlated—but often asynchronous—with the events in the melody. Generating an accompaniment to a given melody is an example of a control task: we seek the ability to generate an accompaniment (the events) conditioned on a given melody (the controls). Models that generate symbolic music (i.e., compose) subject to user-specified controls are of broad interest as tools for music co-creation (Louie et al., 2020).

Motivated by this example, we are interested in constructing generative models of a temporal point process (the *event* process) that can be conditioned on realizations of a second, correlated point process (the *control* process). Generating an accompaniment to a melody is an instance of a more general *infilling* task, whereby we generate a complete realization of a temporal point process given partial observation of a subset of its events. Infilling is a powerful control mechanism for music generation: previous work on musical infilling (Huang et al., 2017) powered the J.S. Bach Google Doodle (Huang et al., 2019), an interactive music experience with broad popular appeal.

The dynamics of a temporal point process can be captured by a neural autoregressive model trained to predict the next event in a time-ordered sequence (Du et al., 2016). A natural extension of this paradigm to conditional distributions is sequence-to-sequence modeling (Seq2Seq) (Sutskever et al., 2014), whereby the control sequence is prepended to the sequence of events. For long sequences, Seq2Seq places time-localized controls far from the events they describe. While there is substantial recent work on long-context modeling (Child et al., 2019; Dao et al., 2022; Gu et al., 2022; Hawthorne et al., 2022), rather than brute-force the learning of artificial long-range dependencies, we propose to structure conditional generation so that a control on time $t$ is located close to events near time $t$. Our premise is that the most relevant context for predicting the next event is the recent event history (unidirectional context) and both recent and near-future controls (bidirectional context).

Standard practice to efficiently train an autoregressive model relies on the observation that context for prediction at one index in the sequence is a prefix of the context for predictions at future indices. This allows us to update the model based on $M - 1$ predictions for each sequence of length $M$. Conditioning on

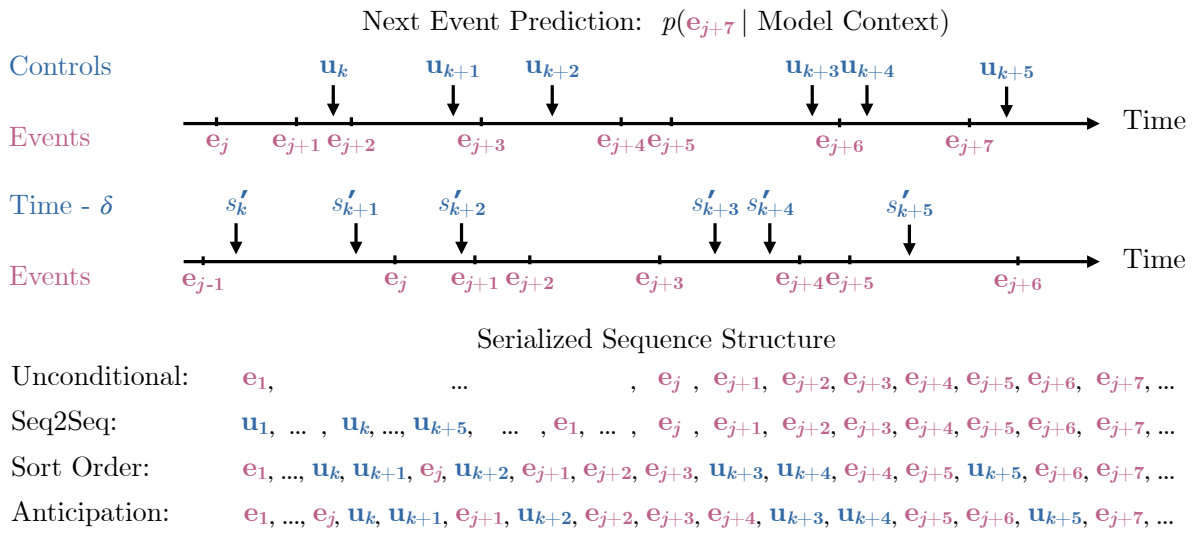


Figure 1: We construct conditional generative models for sequences of events $\mathbf{e}_{1:N}$, conditioned on controls $\mathbf{u}_{1:K}$. We serialize these paired sequences to define an autoregressive factorization of the joint distribution over events and controls. Anticipation interleaves event and control sequences so that a control $\mathbf{u}_k$ on time $s_k$ appears in the recent history when predicting events near time $s_k$. An anticipated control $\mathbf{u}_k$ on time $s_k$ appears as if it were at approximately time $s'_k = s_k - \delta$. For example, when predicting $\mathbf{e}_{j+7}$ above, the recent context of the anticipation sequence contains contains past events and controls, as well as the future control $\mathbf{u}_{k+5}$; we say that a model predicting $\mathbf{e}_{j+7}$ given this context *anticipates* the control $\mathbf{u}_{k+5}$, approximately $\delta$ seconds in advance. Crucially, to be able to condition on controls, the index that immediately preceeds each control in the serialized sequence must be a stopping time, a property that naively interleaving events and controls using the sort order of times $s'_k$ does not satisfy.

asynchronous controls by constructing an ad-hoc context (e.g., the $M/2$ previous events and $M/2$ nearest controls) to predict each event would be computationally wasteful: we want to define a single, coherent interleaved sequence of events and controls. This is straightforward if the events and controls are synchronous: to condition on bidirectional control context $[t - \delta, t + \delta]$ at time $t$, simply shift the control sequence by a time offset $\delta$ and construct a joint sequence by interleaving events and controls at alternating sequence positions. Or alternatively, construct an encoder-decoder model that ingests the paired control tokens through a separate encoder (Raffel et al., 2020).

When events and controls are asynchronous, simple approaches to interleaving these sequences make sampling from the ensuing joint model intractable. This includes the natural *sort order*, whereby we interleave a control on time $t$ as if it were at time $t - \delta$. For tractable sampling, we will see in Section 3 that the index in the interleaved sequence that immediately precedes a control must be a *stopping time* (Billingsley, 1995). We propose a method for interleaving asynchronous events and controls such that a control on time $t$ appears in the interleaved sequence at a stopping time close to events near time $t - \delta$. We call this method *anticipation*. The interval $\delta > 0$ is a hyperparameter chosen to be long enough to give the model time to account for (i.e. anticipate) upcoming controls, but short enough to maintain proximity of events and controls (if $\delta = \infty$, we recover Seq2Seq modeling). The interleaved structure of anticipation is visualized in Figure 1.

**Contributions.** We define an arrival-time encoding of temporal point processes that is amenable to expressive autoregressive sequence modeling and facilitates anticipation (Section 2). We describe the interleaved structure of an anticipatory autoregressive model, and how to train and sample from this model (Section 3). We apply anticipation to construct anticipatory infilling models for music, trained on the Lakh MIDI music dataset Raffel (2016). These models unlock new control capabilities for music generation without sacrificing the performance of unconditional generation (Section 4). We will release all code for reproducing these models, along with pre-trained model weights. For audio samples, see https://dblblnd.github.io/tmlr2023/.

## 2 Music as a Temporal Point Process

A marked temporal point process consists of sparse events situated at points in continuous time (Daley & Vere-Jones, 2007).

**Definition 2.1.** A *marked temporal point process* consists of time-ordered events $\mathbf{e}_i = (t_i, \mathbf{m}_i)$, where $t_i \in \mathbb{R}_+$ ($t_i \leq t_j$ if $i < j$) is a point in time and $\mathbf{m}_i \in \mathcal{V}$ is a mark from a finite vocabulary $\mathcal{V}$.

Given controls $\mathbf{u}_{1:K}$ provided by a user, we say that we can *control* generation of the event process $\mathbf{e}_{1:N}$ with respect to $\mathbf{u}_{1:K}$ if we can sample from $p(\mathbf{e}_{1:N}|\mathbf{u}_{1:K})$. We focus on *infilling* control, whereby the controls $\mathbf{u}_{1:K}$ share a vocabulary with the events $\mathbf{e}_{1:N}$. Given a user-specified set of $K$ events $\mathbf{u}_{1:K}$, we would like to generate a complete realization of the process $\mathbf{e}_{1:N}$ such that $\mathbf{u}_{1:K} \subseteq \mathbf{e}_{1:N}$. This generalizes the *span-infilling* task—which ask us to generate a missing contiguous span of events—previously studied in the music literature (Ippolito et al., 2018; Chang et al., 2021; Tan et al., 2021; 2022).

Music can be described as a marked temporal point process, where the vocabulary of marks consists of musical notes and other musical events. In this work, we restrict the vocabulary of musical events to notes consisting of a *pitch* $\mathbf{p}$, *instrument* class $\mathbf{k}$, and *duration* $\mathbf{d}$. Following the MIDI standard (International MIDI Association, 1988), we quantize pitch $\mathbf{p} \in \{0, \ldots, 127\}$ according to the 12-tone Western scale ($p = 60$ denotes "middle C", or 261Hz); we represent instrument classes $\mathbf{k} \in \{0, \ldots, 128\}$ using standard MIDI instrument codes, including drums (code 128); we represent duration in units of seconds, quantized to 10ms intervals with a maximum duration of 10 seconds. We represent combined pitch and instrument information using a single value $\mathbf{n} = 128\mathbf{k} + \mathbf{p}$. A mark is thus a note $\mathbf{m}_i = (\mathbf{d}_i, \mathbf{n}_i) \in \mathcal{V}$ from a vocabulary of 17,512 marks, and music consists of these notes situated at points in time.

**Modeling Arrival Times.** Stochastic arrival times $t_i$ are the defining characteristic of a temporal point process. In Section 2.1 we propose an *arrival-time tokenization* that represents a realization of a marked point process as a sequence of (discretized) arrival times and marks. In Section 3 we exploit an invariance of this sequence representation (context-free subsequences) to create anticipation. We also consider an *interarrival-time tokenization* that is comparable to encodings used in recent work on music generation (Huang et al., 2018; Donahue et al., 2019; Payne, 2019; Oore et al., 2020). Both of these encodings reduce point process modeling to sequence density estimation. This allows us to directly apply the full modern machinery of causal autoregressive transformers and large language models to modeling point process data. However, only the proposed arrival-time encoding facilitates the construction of anticipation.

**Related Work on Modeling Point Processes.** In the temporal point processes literature, substantial work focuses on modeling a conditional intensity function (Du et al., 2016; Omi et al., 2019; Zuo et al., 2020), e.g., the Hawkes process (Mei & Eisner, 2017). In contrast, we directly model the point process itself (density estimation). In this regard, our work is most comparable to Shchur et al. (2020); but whereas that work models continuous densities, we model discrete distributions over quantized time values. Older work on music rasterizes time with a uniformly sampled grid (Boulanger-Lewandowski et al., 2012) (i.e., a piano-roll). In the piano-roll setting, there are strong solutions to accompaniment and infilling tasks that avoid the complexity of asynchronicity that we address in this paper (Hadjeres et al., 2017; Huang et al., 2017; Pati et al., 2019; Chen et al., 2020). However, for diverse and rhythmically intricate music data (including Lakh MIDI) piano-roll rasterization comes at a cost: if the rasterization is coarse then rhythmic detail is lost, and if the rasterization is fine then the dimensionality of these piano-rolls becomes unwieldy.

### 2.1 Encoding Music as Sequences

We represent arrival times $t_i$ using a finite vocabulary of 10,000 values, quantizing time in 10ms intervals (the quantization proposed by Music Transformer Huang et al. (2018)) with a maximum time of 100 seconds. While many musical performances are longer than 100 seconds, we relativize arrival times to the beginning of the model context. The models described in this paper have context length $M = 1024$. Differences between $M$ successive arrival times that exceed 100 seconds appear in less than 0.1% of Lakh MIDI training examples; we discard these examples during preprocessing. By quantizing time we reduce temporal point processes to

discrete sequences composed of successive pairs of times $\mathbf{t}_i \equiv \texttt{quantize}(t_i)$ and marks $\mathbf{m}_i$ or, in the music application, triplets of time, duration, and note $(\mathbf{t}_i, \mathbf{d}_i, \mathbf{n}_i)$.

**Definition 2.2.** The *arrival-time tokenization* of events $\mathbf{e}_{1:N}$ is a sequence $\mathbf{x}_{1:3N}$ defined by

$$\mathbf{x}_{3i-2} = \mathbf{t}_i, \qquad \mathbf{x}_{3i-1} = \mathbf{d}_i, \qquad \mathbf{x}_{3i} = \mathbf{n}_i. \tag{1}$$

The vocabulary has size $|\mathcal{V}| = 27{,}512$: 10,000 quantized times, 1000 quantized durations, and 16,512 instrument-pitch pairs.

Crucially, the triplets $(\mathbf{x}_{3i-2}, \mathbf{x}_{3i-1}, \mathbf{x}_{3i})$ of these sequences are *context-free*: if we re-order the triplets, the semantics of the sequence do not change. We can recover the original ordering by sorting these triplets according to their arrival times $\mathbf{x}_{3i-2}$. We exploit the re-orderability of arrival time tokenized sequences to construct anticipatory autoregressive infilling models in Section 3. A similar encoding has been used by Gardner et al. (2021), but their work does not exploit this re-orderability.

The more common practice in symbolic music modeling encodes music as a sequence of *onset* and *offset* events, separated by interarrival times $\Delta \in \mathbb{R}_+$. For each event $\mathbf{e} = (\mathbf{t}, \mathbf{m})$, we define onset $\mathbf{on} = (\mathbf{t}, \mathbf{n})$ and offset $\mathbf{off} = (\mathbf{t} + \mathbf{d}, \mathbf{n})$.

**Definition 2.3.** Given events $\mathbf{e}_{1:N}$, let $\mathbf{x}'_{1:2N}$ denote an interleaved sequence of onsets $\mathbf{on}_{1:N}$ and offsets $\mathbf{off}_{1:N}$, ordered by time, where values $\mathbf{x}'_i = (\mathbf{t}'_i, \mathbf{n}'_i)$ have interarrival times $\Delta_i = \mathbf{t}'_{i+1} - \mathbf{t}'_i$. The *interarrival-time tokenization* of events $\mathbf{e}_{1:N}$ is a sequence $\mathbf{x}_{1:4N}$ defined by

$$\mathbf{x}_{2i-1} = \mathbf{x}'_i, \qquad \mathbf{x}_{2i} = \Delta_i. \tag{2}$$

Following standard practice, we omit interarrival tokens when $\Delta_i = 0$. The vocabulary has size $|\mathcal{V}| = 34{,}024$: 16,512 note onsets, 16,512 note offsets, and 1000 quantized interarrival times.

In contrast to Definition 2.2, the interarrival-time tokenization described by Definition 2.3—and other common music encodings including including REMI (Huang & Yang, 2020) and OctupleMidi (Zeng et al., 2021)—are *context sensitive*: the timings of tokens are determined contextually by their positions in the sequence. Interarrival-time tokenization may appear less compact than arrival-time tokenization: each event is encoded as an onset and an offset—each with an associated interarrival time—totaling 4 tokens, versus a triplet of arrival time, duration, and note totaling 3 tokens. However, omitting zero-duration interrarival times results in sequences of comparable length under either encoding for music datasets including Lakh MIDI.

## 3 Anticipation

Given an *anticipation interval* $\delta > 0$, we construct a sequence $\mathbf{a}_{1:N+K} = \texttt{interleave}_\delta(\mathbf{e}_{1:N}, \mathbf{u}_{1:K})$ that interleaves events $\mathbf{e}_{1:N}$ and controls $\mathbf{u}_{1:K}$ such that a control $\mathbf{u}_k$ on time $s_k$ appears close to events near time $s_k - \delta$; i.e., we *anticipate* $\mathbf{u}_k$, $\delta$ seconds in advance. We can then construct a standard autoregressive sequence model over interleaved sequences $\mathbf{a}_{1:N+K}$. Model predictions under the anticipatory ordering $\mathbf{a}_{1:N+K}$ combine a filtering (i.e., causal) estimate based on the local history of events with a smoothing (i.e., bidirectional) estimate based on local controls (Wiener et al., 1949). The value of $\delta$ is a hyperparameter that controls the degree of smoothing. If $\delta$ is too short, then the model will be blind to upcoming control context. If $\delta$ is too long, then time-locality of the sequence $\mathbf{a}_{1:N+K}$ is broken.

The natural *sort order* that interleaves events and controls by merging a control on time $s_k$ between events at times $t_j, t_{j+1}$ such that $t_j \leq s_k - \delta < t_{j+1}$ makes inference intractable. During inference, we only have access to prefixes of $\mathbf{a}_{1:N+K}$, and so the criterion that determines whether to condition on $\mathbf{u}_k$ at index $i$ must be a function of the history $\mathbf{a}_{1:i-1}$. However, the sort order placement of a control $\mathbf{u}_k$ depends on both the event that precedes it and the event that *follows* it in the merged sequence, which requires information that is unavailable at inference time. We formalize this problem using the concept of a *stopping time* (Definition 3.3). Definition 3.1 describes a version of anticipation that interleaves controls $\mathbf{u}_k$ at indices $\tau_{\mathbf{u}_k} \in \{1, \ldots, N+K\}$ after a stopping time $\tau_{\mathbf{u}_k} - 1$.

**Definition 3.1.** (Anticipation) Let $\mathbf{e}_{1:N}$ be events with vocabulary $\mathcal{V}$ and let $\mathbf{u}_{1:K}$ be controls with vocabulary $\mathcal{U}$. Given $\delta > 0$, we define a combined sequence $\mathbf{a}_{1:N+K} \equiv \texttt{interleave}_\delta(\mathbf{e}_{1:N}, \mathbf{u}_{1:K})$ with vocabulary $\mathcal{U} \cup \mathcal{V}$ that interleaves these two sequences. We write $t_j, s_k$ to indicate the arrival times of event $\mathbf{e}_j$ and control $\mathbf{u}_k$ respectively. To simplify notation we define $t_0 = s_0 = -\infty$. In the combined sequence $\mathbf{a}_{1:N+K}$, event $\mathbf{e}_j$ and control $\mathbf{u}_k$ appear (respectively) at indices

$$\tau_{\mathbf{e}_j} = j + \arg\max_{0 \le k \le K} \{t_{j-1} \ge s_k - \delta\}, \tag{3}$$

$$\tau_{\mathbf{u}_k} = k + \arg\min_{0 \le j \le N} \{t_j \ge s_k - \delta\}. \tag{4}$$

Unpacking this definition, Equation (3) says that event $\mathbf{e}_j$ appears in sequence $\mathbf{a}_{1:N+K}$ after the first $j-1$ events (term $j$ in the sum) and after any controls that appear earlier in sequence $\mathbf{a}_{1:N+K}$ (the arg max term, mirroring Equation (4)). Equation (4) says that control $\mathbf{u}_k$ appears after the first $k-1$ controls and *after* the first event $\mathbf{e}_j$ exceeding time $s_k - \delta$. The indices $\tau$ of events and controls in the sequence $\mathbf{a}_{1:N+K}$ are random variables, determined by stochastic realizations of $\mathbf{e}_{1:N}$ and $\mathbf{u}_{1:K}$.

An *anticipatory autoregressive model* is an autoregressive model defined over sequences of events and controls $\mathbf{a}_{1:N+K}$ interleaved according to Equations (3) and (4):

$$p(\mathbf{a}_{1:N+K}) = \prod_{i=1}^{N+K} p(\mathbf{a}_i | \mathbf{a}_{1:i-1}). \tag{5}$$

**Example 3.2.** Suppose $t_1 = 1$, $t_2 = 3$, $t_3 = 5$, $s_1 = 7$, and $\delta = 5$. Then $s_1 - \delta = 2$ and

$$\tau_{\mathbf{e}_1} = 1 + \arg\max_k \{-\infty \ge s_k - \delta\} = 1, \tag{6}$$

$$\tau_{\mathbf{e}_2} = 2 + \arg\max_k \{1 \ge s_k - \delta\} = 2, \tag{7}$$

$$\tau_{\mathbf{e}_3} = 3 + \arg\max_k \{3 \ge s_k - \delta\} = 4, \tag{8}$$

$$\tau_{\mathbf{u}_1} = 1 + \arg\min_j \{t_j \ge 2\} = 3. \tag{9}$$

Therefore $\mathbf{a}_{1:4} = (\mathbf{e}_1, \mathbf{e}_2, \mathbf{u}_1, \mathbf{e}_3)$. Contrast this order with the more natural sort order that interleaves $\mathbf{u}_1$ as if it were an event at time $s_1 - \delta = 2$: $(\mathbf{e}_1, \mathbf{u}_1, \mathbf{e}_2, \mathbf{e}_3)$. Under the sort order, $\mathbf{u}_1$ appears *before* the first event $\mathbf{e}_j$ exceeding time $s_1 - \delta$. This position can only be determined retroactively (after observing the position of $\mathbf{e}_2$) which precludes autoregressive inference. Formally, the problem here is that $\tau_{\mathbf{u}_1} - 1$ is not a stopping time.

### 3.1 Stopping Times

Informally, a stopping time of a stochastic process $\mathbf{x}$ is a random index $\tau$ for which the occurrence of an event $\{\tau = i\}$ can be determined based only on the information observed in the prefix $\mathbf{x}_{1:i}$ (modeled by a sigma algebra $\mathcal{F}_i$). A classic example of a stopping time is a *first hit time*: the first index at which a stochastic process $\mathbf{x}$ attains a value $v$. This is a stopping time because the condition $\{\tau = i\}$ can be determined at time $i$ simply by inspecting whether $\mathbf{x}_i = v$. An example of a random time that is not a stopping time is a *last exit time*: the last index $i$ at which $\mathbf{x}_i = v$. In contrast to the first hit time, the last exit time can only be determined after observing the entire process $\mathbf{x}$.

**Definition 3.3.** (Stopping Times) Let $I$ be an ordered index set, let $(\Omega, \mathcal{F}, (\mathcal{F}_i)_{i \in I})$ be a filtered measurable space, and let $\tau : \Omega \to I$ be a random index defined on this space. We say that $\tau$ is a *stopping time* if $\{\tau = i\} \in \mathcal{F}_i$ for all $i \in I$.

In our case, $I = \{1, \ldots, N+K\}$, and $\tau$ is a random index into the sequence $\mathbf{a}_{1:N+K}$. The filtration $(\mathcal{F}_i)_{i \in I}$ consists of the sigma algebras $\mathcal{F}_i$ generated by the prefix sequences $\mathbf{a}_{1:i}$. While we adopt the conventional terminology of stopping *times*, in this case it might be better to think of $\tau$ as a stopping *index* of $\mathbf{a}_{1:N+K}$, not to be confused with a (continuous) random time in the underlying point process.

To condition on a control $\mathbf{u}_k$, it is essential that $\tau_{\mathbf{u}_k} - 1$ is a stopping time: inference relies on a criterion computed at each index $i-1$ to determine whether to (temporarily) stop sampling and insert a control $\mathbf{a}_i = \mathbf{u}_k$, or to continue sampling events $\mathbf{a}_i \sim p(\mathbf{e}_i|\mathbf{a}_{1:i-1})$ (see Section 3.4). During inference, we only have access to the prefix $\mathbf{a}_{1:i-1}$ and therefore the criterion that determines whether to condition on $\mathbf{u}_k$ at index $i$ must be a function of this history. Whereas $\tau_{\mathbf{u}_k} - 1$ is a stopping time, examples of random indices that do *not* appear after stopping times include:

1. $\sigma_{\mathbf{u}_k} = k + \arg\min_j \{t_{j+1} \geq s_k - \delta\}$. This is where $\mathbf{u}_k$ would appear in sort order, as if $\mathbf{u}_k$ were at time $s_k - \delta$ (naively anticipating $\mathbf{u}_k$, $\delta$ seconds in advance).
2. $\sigma_{\mathbf{u}_k} = k + \arg\min_j \{t_{j+10} \geq s_k\}$. This is 10 indices before where $\mathbf{u}_k$ would appear in sort order, as if $\mathbf{u}_k$ were an event at time $s_k$ (naively anticipating $\mathbf{u}_k$, 10 indices in advance).

In both cases, $\{\sigma_{\mathbf{u}_k} = i\}$ cannot be determined based on observation of the prefix $\mathbf{a}_{1:i-1}$. Each random index depends on unobserved future events and therefore $\sigma_{\mathbf{u}_k} - 1$ is not a stopping time.

### 3.2 Sparse Sequences

The interleaving rule proposed in Definition 3.1 provides no guarantee that a control will appear some number of seconds (or number of indices) in advance of the time that it controls. For very sparse sequences (relative to $\delta$) a control can even appear *after* the time that it controls.

**Example 3.4.** Suppose $t_1 = 1$, $t_2 = 2$, $t_3 = 5$, $s_1 = 4.5$, and $\delta = 2$. Then $s_1 - \delta = 2.5$ and

$$\tau_{\mathbf{e}_1} = 1 + \arg\max_k \{-\infty \geq s_k - \delta\} = 1, \tag{10}$$

$$\tau_{\mathbf{e}_2} = 2 + \arg\max_k \{1 \geq s_k - \delta\} = 2, \tag{11}$$

$$\tau_{\mathbf{e}_3} = 3 + \arg\max_k \{3 \geq s_k - \delta\} = 3, \tag{12}$$

$$\tau_{\mathbf{u}_1} = 1 + \arg\min_j \{t_j \geq 2.5\} = 4. \tag{13}$$

Therefore $\mathbf{a}_{1:4} = (\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3, \mathbf{u}_1)$. And in particular, the control $\mathbf{u}_1$ on time $s_1 = 4.5$ appears after event $\mathbf{e}_3$, which occurs at time $t_3 = 5$.

**Definition 3.5.** Given a sequence of events $\mathbf{e}_{1:N}$ that occur at times $\mathbf{t}_{1:N}$, let $\Delta_{\max}$ denote the maximum distance between adjacent events, i.e.

$$\Delta_{\max} = \max\{\mathbf{t}_{i+1} - \mathbf{t}_i : 1 \leq i < N\}. \tag{14}$$

We say that the sequence $\mathbf{e}_{1:N}$ is $\Delta_{\max}$-*dense*.

For a $\Delta_{\max}$-dense sequence $\mathbf{e}_{1:N}$, we can guarantee that a control on time $\mathbf{t}$ appears at or before time $\mathbf{t} - \delta + \Delta_{\max}$ using the sequence order given by Definition 3.1. If $\delta - \Delta_{\max} > 0$ is small, the model may have little time to plan for anticipated controls; if $\Delta_{\max} > \delta$ then we risk anticipating some controls after the times that they are supposed to control, as in Example 3.4. To ensure dense sequences, we insert special `REST` events into inter-event intervals that exceed a target density $\Delta^*$: if $n\Delta^* < \mathbf{t}_{i+1} - \mathbf{t}_i \leq (n+1)\Delta^*$ then we insert $n$ `REST` events at times $\mathbf{t}_i + \Delta^*, \mathbf{t}_i + 2\Delta^*, \ldots, \mathbf{t}_i + n\Delta^*$. This is analogous to how a musician counts out rests in a musical score.

**Example 3.6.** Continuing Example 3.4, we see that $\Delta_{\max} = 3 > 2 = \delta$. Inserting `REST` events to ensure $\Delta^*$-density for $\Delta^* = 1$, the event sequence becomes $\mathbf{e}'_{1:5}$, adding `REST` tokens at times 3 and 4 such that

$$\mathbf{e}'_1 = \mathbf{e}_1, \qquad \mathbf{e}'_2 = \mathbf{e}_2, \qquad \mathbf{e}'_3 = (3, \texttt{REST}), \qquad \mathbf{e}'_4 = (4, \texttt{REST}), \qquad \mathbf{e}'_5 = \mathbf{e}_3. \tag{15}$$

In this case, the anticipatory interleaving of $\mathbf{u}_1$ with $\mathbf{e}'_{1:5}$ is $\mathbf{a}' = (\mathbf{e}'_1, \mathbf{e}'_2, \mathbf{e}'_3, \mathbf{u}_1, \mathbf{e}'_4, \mathbf{e}'_5)$.

And in this case, we anticipate the control $\mathbf{u}_1$ on time $s_1 = 4.5$ between the event $\mathbf{e}'_3$ at time 3 and the event $\mathbf{e}'_4$ at time 4. All the anticipatory models in this paper are trained with $\Delta^* = 1$ second.

### 3.3 Training Anticipatory Models

We train anticipatory autoregressive models using standard maximum likelihood estimation of the sequences $\mathbf{a}_{1:N+K}$ (Definition 3.1). We tokenize these sequences according to the encoding described in Definition 2.2: $\mathbf{x}_{1:3(N+K)} \equiv \texttt{tokenize}(\mathbf{a}_{1:N+K})$. We follow a standard sequence packing procedure (see, e.g, Appendix B of Brown et al. (2020)) to construct training examples of fixed length $M$ (the model context) from variable-length sequences $\mathbf{x}_{1:3(N+K)}$, using a special event `SEP` as the sequence separator. We prepend each training example with a single global control code $\mathbf{z} \in \{0, 1\}$ that indicates whether the example contains local controls $\mathbf{u}_k$; setting $\mathbf{z} = 0$ (no controls) facilitates comparisons between anticipatory and autoregressive models. If the training example spans multiple sequences, then $\mathbf{z}$ describes the sequence preceding the first `SEP` event in the example. We randomly shuffle and mini-batch training examples for stochastic gradient training.

We can choose to either predict all of $\mathbf{a}_{1:N+K}$, learning a *joint* generative model over events and controls, or alternatively just predict the events $\mathbf{e}_{1:N}$ (by zeroing out the training losses at indices corresponding to controls) and learn a *conditional* generative model over the events, given controls. In this paper, we predict controls in addition to the events: this maximizes the number of predictions for each example ($M-1$ predictions for a training example of length $M$). For the infilling application (Section 3.5) predicting events and predicting control are similar enough tasks that improvements in the two tasks ought to reinforce each other; this reinforcement has been observed empirically in the language modeling domain (see Donahue et al. (2020), Appendix C).

Dividing up a dataset into training examples of length $M$ introduces boundary effects. For general autoregressive models, this procedure results in an "early token curse," whereby predictions early in a training example must be made with limited context (Press et al., 2021). For anticipatory autoregressive models, more subtle boundary effects arise. For an anticipation interval $\delta$, controls on first $\delta$ seconds of the training example do not appear in the context: these controls appear at the end of the previous training example. If $\delta$ is large relative to $M$ then the boundary effects become severe: many event predictions will be made without the relevant contextual controls, and vice-versa. This tempers the value of making predictions for all $M-1$ sequence indices, imposing a drag on training efficiency for large values $\delta$ relative to $M$. Like the early token curse, this effect is mitigated with larger model contexts $M$ relative to $\delta$; in practice, the maximum practical anticipation interval $\delta$ is thus coupled with the context size of the model that we plan to train.

### 3.4 Anticipatory Inference

We draw conditional samples from an anticipatory autoregressive model $p$ according to the procedure described in Algorithm 1. If there are controls ($K \neq 0$) we set $\mathbf{z} = 1$ (anticipatory sampling mode). In the outer loop, we sample $\mathbf{a}_i \sim p(\cdot|\mathbf{a}_{1:i-1})$. We impose three constraints upon the sampling distribution, all implemented by logit masking: (1) each arrival time must equal or exceed the previous arrival time (monotonicity) (2) a time token must follow a note token, a duration token must follow a time token, and a note token must follow a duration token (proper ordering) and (3) the model must not generate a control token (controls are pre-specified by the user).

Each time we draw a sample $\mathbf{a}_i$ from the model, we note its time $t_i = \texttt{time}(\mathbf{a}_i)$ and check for controls satisfying the condition given by Equation (4) (leveraging the fact that this condition is a stopping time). We anticipate these controls by appending them to the generated output following $\mathbf{a}_i$. The sequence $\mathbf{a}_{1:N+K}$ returned by Algorithm 1 contains $N$ events interleaved with $K$ labels. We can postprocess this sequence by stripping out the controls to recover the sequence of generated events $\mathbf{e}_{1:N}$.

### 3.5 Anticipatory Infilling Models

We apply the anticipatory modeling framework to infilling control, whereby the labels $\mathbf{u}_{1:K}$ consist of a subset of the events $\mathbf{e}_{1:N}$. We duplicate the event vocabulary to distinguish between regular events $\mathbf{e}'_j \in \mathcal{V}$ and control events $\mathbf{u}_k \in \mathcal{U}$; let $\varphi : \mathcal{V} \to \mathcal{U}$ denote the (bijective) map between the event and control vocabularies. In this case, the combined sequence $\mathbf{a}_{1:(N-K)+K} = \mathbf{a}_{1:N}$ (Definition 3.1) is a re-ordering of

**Algorithm 1:** Anticipatory Autoregressive Sampling

**Parameters:** Anticipatory autoregressive model $p$ with context length $M$
Anticipation interval $\delta > 0$
**Input** : Time-localized controls $\mathbf{u}_{1:K}$ (monotone increasing in time)
Non-localized controls $\mathbf{z}$ (global control codes)
**Output** : A generated sequence $\mathbf{a}_{1:N+K}$

```
a_0 ← SEP                                   // A special sequence separator event
i ← 1                                       // Index i tracks position in the generated sequence
k ← 0                                       // Index k tracks position in the control sequence
do
    t ← time(a_{i-1})                       // Get the time t of the previous event a_{i-1}
    while time(u_k) ≤ t + δ                 // While there are controls before time t + δ
    do
        a_i ← u_k                           // Anticipate control u_k at index i
        i ← i + 1                           // Advance to index i + 1
        k ← k + 1                           // Consume control u_k
    end
    Sample a_i ~ p(·|z, a_{i-M-Length(z):i-1})   // Sample an event from the model
    i ← i + 1                               // Advance to index i + 1
while a_i ≠ SEP
return a_{1:i-1}                            // The value N+K = i-1
```

the event sequence $\mathbf{e}_{1:N}$, consisting of events $\mathbf{e}'_{1:N-K} \subseteq \mathbf{e}_{1:N}$ and controls $\mathbf{u}_{1:K}$, with $\varphi^{-1}(\mathbf{u}_{1:K}) \subseteq \mathbf{e}_{1:N}$ and $\mathbf{e}_{1:N-K} \cap \varphi^{-1}(\mathbf{u}_{1:K}) = \varnothing$. Using an arrival-time tokenization of events (Definition 2.2) allows us to re-order $\mathbf{e}_{1:N}$ while preserving the semantics of the sequence. We can recover the original sequence by converting the control events back to the event vocabulary and sorting all the events according to their arrival times:

$$\mathbf{e}'_{1:N-K}, \mathbf{u}_{1:K} = \texttt{split}(\mathbf{a}_{1:N}), \tag{16}$$

$$\mathbf{e}_{1:N} = \texttt{sort}\left(\mathbf{e}'_{1:N-K} \cup \varphi^{-1}(\mathbf{u}_{1:K})\right). \tag{17}$$

We describe the precise encoding of $\mathcal{U} \cup \mathcal{V}$ that we use for music infilling models in Appendix C.

Previously we have assumed that the controls $\mathbf{u}_{1:K}$ are given to us, separate from the the events $\mathbf{e}_{1:N}$. For the infilling task, there is no distinction between events and controls in the training data; a user could specify an arbitrary set of events to condition on at inference time. To train an infilling model, we must impose a prior over the distribution over subsets of control events $\mathbf{u}_{1:K} \subseteq \mathbf{e}_{1:N}$. This prior should simulate common infilling patterns, and generalize to accommodate patterns that we did not foresee during training. To that end, we propose a distribution over control events consisting of a mixture of random spans of time, random subgroups of instrument, and uniformly random events. We describe specific details of this prior distribution in Appendix D.

We caution that the conditionals learned by an anticipatory infilling model are not consistent with a unique joint distribution over sequences $\mathbf{e}_{1:N}$. Given controls $\mathbf{u}_{1:K}$, the map between the sequences $\mathbf{e}_{1:N}$ and $\mathbf{a}_{1:N}$ defined by Equation (17) is a bijection. The probability distribution over sequences $\mathbf{a}_{1:N}$ therefore also defines a distribution over sequences $\mathbf{e}_{1:N}$. Every subset $\varphi^{-1}(\mathbf{u}_{1:K}) \subseteq \mathbf{e}_{1:N}$ can be used as infilling controls, each resulting in a distinct anticipatory sequence $\mathbf{a}_{1:N}$. But the probability distributions $p(\mathbf{e}_{1:N})$ implied by sequences $\mathbf{a}_{1:N}$ will not be the same. We rely on the model's ability to well-approximate the underlying distribution over sequences $\mathbf{e}_{1:N}$ to ensure some degree of compatibility between these families of learned distributions.

## 4 Anticipatory Infilling Models of Music

We train and release anticipatory infilling models on the Lakh MIDI dataset (Raffel, 2016). See Appendix B for licensing information and Section A for a discussion of copyright considerations regarding models trained

Table 1: Evaluation Loss. All losses are reported on the Lakh MIDI test set sequences $\mathbf{e}_{1:N}$ (without anticipation). For the arrival-time models, we define a per-event loss $L_{\mathbf{e}}$ summed across event triples (Definition 2.2) and report the per-event perplexity $\text{ppl}(\mathbf{e}) = \exp(L_{\mathbf{e}})$. This loss decomposes into $L_{\mathbf{e}} = L_{\mathbf{t}} + L_{\mathbf{d}} + L_{\mathbf{n}}$ with corresponding perplexities for onsets $\mathbf{t}$, durations $\mathbf{d}$ and notes $\mathbf{n}$. The time-normalized bits per second metric (bps) (Thickstun et al., 2019) is defined in Appendix F. Parameter counts differ between arrival and interarrival models, due to the difference in vocabulary size. Anticipatory models are indicated by the 'AM' flag. For anticipatory models, we evaluate with $\mathbf{z} = 0$ (see Section 3.3).

| # | Config | Params | Steps | Encoding | AM | bps | ppl($\mathbf{e}$) | ppl($\mathbf{t}$) | ppl($\mathbf{d}$) | ppl($\mathbf{n}$) |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Small | 112M | 100k | interarrival | ✗ | 85.9 | - | - | - | - |
| 2 | Small | 128M | 100k | arrival | ✗ | 80.4 | 14.9 | 1.59 | 3.90 | 2.40 |
| 3 | Small | 128M | 100k | arrival | ✓ | 80.7 | 15.0 | 1.58 | 3.98 | 2.40 |
| 4 | Small | 128M | 800k | arrival | ✗ | 75.7 | 12.7 | 1.53 | 3.65 | 2.27 |
| 5 | Small | 128M | 800k | arrival | ✓ | 75.0 | 12.4 | 1.52 | 3.64 | 2.24 |
| 6 | Medium | 360M | 100k | arrival | ✓ | 74.4 | 12.1 | 1.54 | 3.55 | 2.22 |
| 7 | Medium | 360M | 200k | arrival | ✓ | 71.5 | 11.1 | 1.51 | 3.39 | 2.16 |
| 8 | Medium | 360M | 800k | arrival | ✓ | 69.7 | 10.4 | 1.49 | 3.29 | 2.12 |
| 9 | Large | 780M | 100k | arrival | ✓ | 73.2 | 11.7 | 1.52 | 3.44 | 2.23 |

on Lakh MIDI. The Lakh MIDI dataset consists of 178,561 MIDI files (event sequences) that we preprocess into 663,555,310 events (1,990,665,930 tokens using arrival-time encoding) encompassing 8943 hours of music. This dataset is orders of magnitude larger than other common music datasets (Dong et al., 2020), but orders of magnitude smaller than the datasets used to train large models in other domains, e.g. language (Gao et al., 2020) and vision (Schuhmann et al., 2022). For reference, the OpenWebText corpus that reproduces the training set used to train GPT-2 contains approximately 10 billion tokens (Gokaslan & Cohen, 2019). We describe training splits and additional information about Lakh MIDI and preprocessing in Appendix E.

We parameterize anticipatory infilling models using causal masked transformers (Vaswani et al., 2017) (decoder-only models) with anticipation interval $\delta = 5$ seconds. This allows the models to look ahead (i.e. anticipate) controls up to 5 seconds before their arrival. Due to the high cost of training, we defer systematic ablations of this hyper-parameter to future work. We train models at three scales, following GPT-2 naming conventions (Radford et al., 2019): Small (128M parameters), Medium (360M parameters), and Large (780M parameters). Because anticipatory models are trained like sequence models on the augmented sequences $\mathbf{a}_{1:N+K}$ (see Section 3.3) we are able to use standard libraries for training anticipatory music transformers; in this work, we use the Levanter library for training.[1] For additional details of models and training procedures, see Appendix G.

### 4.1 Automatic Metrics

See Table 1 for a summary of configurations and performance metrics derived from the log-loss for anticipatory infilling models trained on Lakh MIDI. Arrival-time encoding appears more effective for training autoregressive transformers than interarrival-time encoding (rows 1 and 2). Anticipatory training incurs a small tax on performance compared to a baseline autoregressive model for short training schedules (compare rows 2 and 3) however this gap disappears with a longer training schedule (compare rows 4 and 5). In this sense, anticipation unlocks infilling control capabilities in autoregressive music models "for free." Larger models achieve significantly better perplexity on test (compare rows 3, 6, and 9) as do longer training schedules (compare rows 2 and 4; 3 and 5; 6, 7, and 8). Therefore, despite the relatively small size of the Lakh MIDI dataset compared to language modeling datasets, we do not appear to have saturated this dataset or overfit: better results are likely achievable with larger-scale models, trained for longer.

[1] https://github.com/stanford-crfm/levanter

### 4.2 Human Evaluation

We solicited human evaluation of generated music to ground the performance of these models in human assessments of quality, to compare these models to other music generation systems, and to evaluate the anticipatory capabilities of these models that are not captured by the predictive log-loss. We evaluate generated outputs for two tasks—described below—following a similar procedure to Huang et al. (2018), whereby we ask workers to identify which of two 20-second synthesized audio clips is more *conventionally musical*. We recruited crowd workers on the Amazon Mechanical Turk platform to perform these tasks. We paid workers \$0.75 US dollars for each pairwise evaluation. Assuming that workers listen to each clip twice—and spend an additional 40 seconds to make their decision and overhead time between tasks—this amounts to two minutes of time per task, or a \$22.50 hourly rate. We pre-qualified workers for by asking them to distinguish between five pairs of human compositions versus melodies accompanied by the random retrieval baseline (described below for the accompaniment task).

All samples from our models used for human evaluation are generated using nucleus sampling (Holtzman et al., 2020) with top-p probability $p = 0.95$. We chose this threshold by manually inspecting music generated from the 800k-step Medium model (Row 8 in Table 1) conditioned on prompts from the validation set and using values of $p \in \{0.9, 0.95, 0.98, 1.0\}$. We chose the value $p$ that most consistently produced high quality music (in the authors' judgement). Sampling with $p < 1.0$ generates less diverse music; anecdotally we observe that music generated at $p = 0.95$ is more conservative than both music generated at $p = 1.0$ and music composed by humans (e.g., sampling at $p = 0.95$ is less likely to introduce a new instrumental part). It is possible that our instruction for workers to identify the most conventionally musical clip favors this more conservative generated music.

**Prompt Continuation.** Workers evaluated 50 clips, each consisting of a three-bar prompt—randomly selected from the beginning of tracks in the Lakh MIDI test set—followed by a continuation generated by a model or from the original human composition. See Table 2 for human evaluation of music generated from prompts, compared to human compositions; for full pairwise comparison results, see Appendix J. We use this task to compare our model in the standard autoregressive setting to a baseline Music Transformer (Huang et al., 2018) as implemented by von Rütte et al. (2023) and trained on the Lakh MIDI dataset; we refer to this as the FIGARO baseline. While continuing a prompt does not require anticipation, evaluators find that music generated by anticipatory models given a prompt is considerably more musical than music generated by the baseline FIGARO model (this should not be surprising; the FIGARO Music Transformer is a smaller, 30M parameter model). Human evaluators also prefer the Small arrival-time model (Row 3; Table 1) over the Small interarrival-time model (Row 1; Table 1) which is consistent with the superior log-loss of the arrival-time model.

**Accompaniment.** Workers evaluated 20 clips, each consisting of a five-second prompt (randomly selected from tracks in the Lakh MIDI test set) followed by a 15-second *accompaniment* (i.e. infilling) conditioned on the prompt and the full (20 seconds) melodic line. See Table 3 for human evaluation of generated music accompaniments, compared to human compositions; for full pairwise comparison results, see Appendix J. For this task, we crudely define *melodic line* to be the instrumental part with the highest pitch. This task allows us to probe the infilling capabilities of our anticipatory autoregressive models. We compare anticipation to two baseline accompaniment algorithms: random retrieval and autoregressive accompaniment. Random retrieval is a simple baseline whereby we accompany the melody with a random 15-second clip retrieved from elsewhere in the track. Autoregressive accompaniment is a best effort algorithmic attempt to use autoregressive generation (without anticipation) to solve the accompaniment task; see Appendix I for details. Both anticipatory and autoregressive accompaniments are generated using a Medium anticipatory model (Row 8; Table 1). Evaluators express a mild preference for anticipatory accompaniments over the human composition skyline. While this result is not statistically significant, it points to the effectiveness of anticipation, especially in the more constrained accompaniment setting.

**Related Work on Musical Accompaniment.** In contrast to harmonization tasks, which seek to accompany a melody with simple chords (Simon et al., 2008; Yeh et al., 2021; Chen et al., 2021; Wu et al., 2021; Rhyu et al., 2022), here we seek to generate complete, asynchronous musical accompaniments to a

Table 2: Human evaluation of generated continuations of three-bar musical prompts versus human compositions. P-values are reported using a Wilcoxon signed rank test. Row numbers reference Table 1.

| Model | Wins | Ties | Losses | p-value |
|---|---|---|---|---|
| Medium (Row 8) | 44 | 29 | 77 | 0.0027 |
| Music Transformer (von Rütte et al., 2023) | 13 | 18 | 119 | $2.806 \times 10^{-20}$ |
| Small (Row 3) | 43 | 23 | 84 | 0.0002 |
| Small (Row 1) | 31 | 16 | 103 | $4.976 \times 10^{-10}$ |

Table 3: Human evaluation of 15-second accompaniments versus human compositions. P-values are reported using a Wilcoxon signed rank test.

| Algorithm B | Wins | Ties | Losses | p-value |
|---|---|---|---|---|
| Anticipatory | 18 | 31 | 11 | 0.194 |
| Autoregressive | 5 | 10 | 45 | $1.542 \times 10^{-08}$ |
| Retrieval | 2 | 6 | 52 | $1.017 \times 10^{-11}$ |

given melody. Dong et al. (2018) previously considered an accompaniment task using piano-roll encodings of the Lakh MIDI dataset: see Section 2 for a discussion of the limitations of piano-roll encodings. Both Zhu et al. (2018) and Ren et al. (2020) propose encoder-decoder architectures for conditional generation of accompaniments given a melody: we discuss obstructions to efficiently training encoder-decoder models of music at scale in Section 1. Shih et al. (2022) consider a more abstract form of conditioning, using a melody as thematic material to generate an arrangement—rather than a literal accompaniment—of the given melody. Our accompaniment task is also loosely analogous to recent work on vocal accompaniment in the audio domain (Donahue et al., 2023).

## 5 Related Work on Controllable Generative Modeling

Markov-chain Monte Carlo (MCMC) samplers are a natural candidate for conditional sampling tasks, and in particular infilling. Diffusion models have proven highly effective as generative models of images with infilling capabilities (Ho et al., 2020; Lugmayr et al., 2022). Masked language models can also be repurposed as MCMC samplers for infilling (Goyal et al., 2021; Mireshghallah et al., 2022; Wang & Xia, 2021). These models are not easily applied to conditional point process generation, which requires the generation of variable numbers of events within a specified region of space or time. MCMC models of symbolic music compromise by either discretizing time with piano-rolls (Hadjeres et al., 2017; Huang et al., 2017) or modeling coarse, fixed-rate latent variables, without the ability to control granular details of the variable-length sequence (Mittal et al., 2021).

Anticipatory infilling models are motivated in part by control capabilities of recent text infilling models (Zhu et al., 2019; Du et al., 2022; Aghajanyan et al., 2022) and the growing empirical evidence that autoregressive models can be augmented with infilling objectives without sacrificing unconditional modeling performance (Donahue et al., 2020; Bavarian et al., 2022). An analogous approach to span-infilling for music using Seq2Seq conditioning was proposed by Ippolito et al. (2018). Our discussion of anticipation draws inspiration from the asynchronous control setting considered by Hassibi et al. (1999). Anticipatory infilling models also draw inspiration from orderless NADE (Uria et al., 2014) and XLNet (Yang et al., 2019), which learn ensembles of models over different autoregressive factorizations. Anticipatory infilling models differ from these methods by (1) applying *local* rather than global permutations of the factorization order and (2) achieving this by permuting the sequence itself (facilitated by context-free arrival-time encoding; Section 2.1) rather than masking it.

## 6 Conclusion

Provided that locality continues to be an important inductive bias for learning, we believe that anticipation will be useful for modeling conditional temporal point processes. Locality is a perennial theme in the machine learning literature. Popular model architectures including ConvNets (Forsyth et al., 1999) and LSTM (Hochreiter & Schmidhuber, 1997) exploit locality as an inductive bias. Many long context Transformers adopt local sparsity as an approximation to dense attention (Child et al., 2019; Beltagy et al., 2020; Zaheer et al., 2020). Locality can be exploited to improve training efficiency via staged training (Press et al., 2021; 2022). The local structure of anticipatory sequences is conducive to the broad class of methods that exploit or depend upon locality. Because anticipation only intervenes to modify the data, it can be seamlessly combined with other modeling innovations, e.g., in the music space: the relative transformer (Huang et al., 2018), the compound word transformer (Hsiao et al., 2021), and RIPO attention (Guo et al., 2022).

Generative music models have not yet reached a broad audience alongside generative models of language (Brown et al., 2020) or images (Ramesh et al., 2022). Slow adoption of music models within creative communities is partly attributable to the difficulty of controlling these models (Briot & Pachet, 2020). Users value agency in human-AI collaborations, preferring to take an active role over more automated solutions (Oh et al., 2018; Roy et al., 2019). We note the proliferation of recent work on music generation (primarily in the audio domain) controlled by text (Forsgren & Martiros, 2022; Kreuk et al., 2022; Agostinelli et al., 2023; Schneider et al., 2023; Huang et al., 2023b;a); we are intrigued by the possibility of applying anticipation to generate symbolic music conditioned on localized text labels (e.g., lyrics). More broadly, we view symbolic music generation and audio music synthesis as complementary, analogous to text generation and speech synthesis. We hope that the Anticipatory Music Transformer may fill a role that is currently underdeveloped in the construction of controllable music generation systems that support the human creative process.

## References

2d Circuit. Authors Guild, inc. v. Google, inc. 721 F.3d 132, 2015. A.2

Sajjad Abdoli. Iranian traditional music dastgah classification. In *International Society for Music Information Retrieval*, 2011. A.3

Daron Acemoglu and Pascual Restrepo. Artificial intelligence, automation, and work. In *The economics of artificial intelligence: An agenda*. University of Chicago Press, 2018. A.1

Armen Aghajanyan, Bernie Huang, Candace Ross, Vladimir Karpukhin, Hu Xu, Naman Goyal, Dmytro Okhonko, Mandar Joshi, Gargi Ghosh, Mike Lewis, et al. Cm3: A causal masked multimodal model of the internet. *arXiv preprint arXiv:2201.07520*, 2022. 5

Andrea Agostinelli, Timo I Denk, Zalán Borsos, Jesse Engel, Mauro Verzetti, Antoine Caillon, Qingqing Huang, Aren Jansen, Adam Roberts, Marco Tagliasacchi, et al. Musiclm: Generating music from text. *arXiv preprint arXiv:2301.11325*, 2023. 6

Mohammad Bavarian, Heewoo Jun, Nikolas Tezak, John Schulman, Christine McLeavey, Jerry Tworek, and Mark Chen. Efficient training of language models to fill in the middle. *arXiv preprint arXiv:2207.14255*, 2022. 5

Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020. 6

Emmanouil Benetos and Andre Holzapfel. Automatic transcription of turkish makam music. In *International Society for Music Information Retrieval*, 2013. A.3

Patrick Billingsley. *Probability and Measure (3rd Edition)*. John Wiley & Sons, 1995. 1

Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. Modeling temporal dependencies in high-dimensional sequences: application to polyphonic music generation and transcription. In *International Conference on Machine Learning*, 2012. 2

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax. G

Jean-Pierre Briot and François Pachet. Deep learning for music generation: challenges and directions. *Neural Computing and Applications*, 2020. 6

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 2020. 3.3, 6

Nicholas Carlini, Daphne Ippolito, Matthew Jagielski, Katherine Lee, Florian Tramer, and Chiyuan Zhang. Quantifying memorization across neural language models. *arXiv preprint arXiv:2202.07646*, 2022. A.2

Chin-Jui Chang, Chun-Yi Lee, and Yi-Hsuan Yang. Variable-length music score infilling via xlnet and musically specialized positional encoding. In *International Society for Music Information Retrieval*, 2021. 2

Ke Chen, Cheng-i Wang, Taylor Berg-Kirkpatrick, and Shlomo Dubnov. Music sketchnet: Controllable music generation via factorized representations of pitch and rhythm. In *International Society for Music Information Retrieval*, 2020. 2

Yi-Wei Chen, Hung-Shin Lee, Yen-Hsing Chen, and Hsin-Min Wang. Surprisenet: Melody harmonization conditioning on user-controlled surprise contours. In *International Society for Music Information Retrieval*, 2021. 4.2

CHESScom. Chess is booming! and our servers are struggling. https://www.chess.com/blog/CHESScom/chess-is-booming-and-our-servers-are-struggling, 2023. Accessed on 22 May 2023. A.4

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019. 1, 6

Daryl J Daley and David Vere-Jones. *An introduction to the theory of point processes: volume II: general theory and structure*. Springer Science & Business Media, 2007. 2

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. *Advances in Neural Information Processing Systems*, 2022. 1

Chris Donahue, Huanru Henry Mao, Yiting Ethan Li, Garrison W Cottrell, and Julian McAuley. Lakhnes: Improving multi-instrumental music generation with cross-domain pre-training. In *International Society for Music Information Retrieval*, 2019. 2

Chris Donahue, Mina Lee, and Percy Liang. Enabling language models to fill in the blanks. In *Annual Meeting of the Association for Computational Linguistics*, 2020. 3.3, 5

Chris Donahue, Antoine Caillon, Adam Roberts, Ethan Manilow, Philippe Esling, Andrea Agostinelli, Mauro Verzetti, Ian Simon, Olivier Pietquin, Neil Zeghidour, et al. Singsong: Generating musical accompaniments from singing. *arXiv preprint arXiv:2301.12662*, 2023. 4.2

Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang. Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018. 4.2

Hao-Wen Dong, Ke Chen, Julian McAuley, and Taylor Berg-Kirkpatrick. Muspy: A toolkit for symbolic music generation. In *International Society for Music Information Retrieval*, 2020. 4

Nan Du, Hanjun Dai, Rakshit Trivedi, Utkarsh Upadhyay, Manuel Gomez-Rodriguez, and Le Song. Recurrent marked temporal point processes: Embedding event history to vector. In *SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016. 1, 2

Zhengxiao Du, Yujie Qian, Xiao Liu, Ming Ding, Jiezhong Qiu, Zhilin Yang, and Jie Tang. Glm: General language model pretraining with autoregressive blank infilling. In *Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2022. 5

Seth Forsgren and Hayk Martiros. Riffusion - Stable diffusion for real-time music generation, 2022. URL https://riffusion.com/about. 6

David A Forsyth, Joseph L Mundy, Vito di Gesú, Roberto Cipolla, Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. Object recognition with gradient-based learning. *Shape, Contour and Grouping in Computer Vision*, pp. 319–345, 1999. 6

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020. 4

Joshua P Gardner, Ian Simon, Ethan Manilow, Curtis Hawthorne, and Jesse Engel. Mt3: Multi-task multitrack music transcription. In *International Conference on Learning Representations*, 2021. 2.1

Aaron Gokaslan and Vanya Cohen. Openwebtext corpus, 2019. URL http://Skylion007.github.io/OpenWebTextCorpus. 4

Kartik Goyal, Chris Dyer, and Taylor Berg-Kirkpatrick. Exposing the implicit energy networks behind masked language models via metropolis–hastings. In *International Conference on Learning Representations*, 2021. 5

Albert Gu, Karan Goel, and Christopher Re. Efficiently modeling long sequences with structured state spaces. In *International Conference on Learning Representations*, 2022. 1

Zixun Guo, Jaeyong Kang, and Dorien Herremans. A domain-knowledge-inspired music embedding space and a novel attention mechanism for symbolic music modeling. In *AAAI Conference on Artificial Intelligence*, 2022. 6

Gaëtan Hadjeres, François Pachet, and Frank Nielsen. Deepbach: a steerable model for bach chorales generation. In *International Conference on Machine Learning*, 2017. 2, 5

Arash Hassibi, Stephen P Boyd, and Jonathan P How. Control of asynchronous dynamical systems with rate constraints on events. In *Conference on Decision and Control*. IEEE, 1999. 5

Curtis Hawthorne, Andrew Jaegle, Cătălina Cangea, Sebastian Borgeaud, Charlie Nash, Mateusz Malinowski, Sander Dieleman, Oriol Vinyals, Matthew Botvinick, Ian Simon, et al. General-purpose, long-context autoregressive modeling with perceiver ar. In *International Conference on Machine Learning*, 2022. 1

Michael A Hedderich, Lukas Lange, Heike Adel, Jannik Strötgen, and Dietrich Klakow. A survey on recent approaches for natural language processing in low-resource scenarios. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2021. A.3

Peter Henderson, Xuechen Li, Dan Jurafsky, Tatsunori Hashimoto, Mark A. Lemley, and Percy Liang. Foundation models and fair use. *arXiv preprint arXiv:2303.15715*, 2023. A.2

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units. *arXiv preprint arXiv:1606.08415*, 2016. G

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 2020. 5

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 1997. 6

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022. G

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. In *International Conference on Learning Representations*, 2020. 4.2

Wen-Yi Hsiao, Jen-Yu Liu, Yin-Cheng Yeh, and Yi-Hsuan Yang. Compound word transformer: Learning to compose full-song music over dynamic directed hypergraphs. In *AAAI Conference on Artificial Intelligence*, 2021. 6

Cheng-Zhi Anna Huang, David Duvenaud, and Krzysztof Z Gajos. Chordripple: Recommending chords to help novice composers go beyond the ordinary. In *Proceedings of the 21st international conference on intelligent user interfaces*, 2016. A.4

Cheng-Zhi Anna Huang, Tim Cooijmans, Adam Roberts, Aaron Courville, and Douglas Eck. Counterpoint by convolution. In *International Society for Music Information Retrieval*, 2017. 1, 2, 5

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*, 2018. 2, 2.1, 4.2, 4.2, 6, H, 8

Cheng-Zhi Anna Huang, Curtis Hawthorne, Adam Roberts, Monica Dinculescu, James Wexler, Leon Hong, and Jacob Howcroft. The Bach Doodle: Approachable music composition with machine learning at scale. In *International Society for Music Information Retrieval*, 2019. 1

Qingqing Huang, Daniel S Park, Tao Wang, Timo I Denk, Andy Ly, Nanxin Chen, Zhengdong Zhang, Zhishuai Zhang, Jiahui Yu, Christian Frank, et al. Noise2music: Text-conditioned music generation with diffusion models. *arXiv preprint arXiv:2302.03917*, 2023a. 6

Rongjie Huang, Jiawei Huang, Dongchao Yang, Yi Ren, Luping Liu, Mingze Li, Zhenhui Ye, Jinglin Liu, Xiang Yin, and Zhou Zhao. Make-an-audio: Text-to-audio generation with prompt-enhanced diffusion models. *arXiv preprint arXiv:2301.12661*, 2023b. 6

Yu-Siang Huang and Yi-Hsuan Yang. Pop music transformer: Beat-based modeling and generation of expressive pop piano compositions. In *ACM International Conference on Multimedia*, 2020. 2.1, H

International MIDI Association. Standard midi files, 1988. 2

Daphne Ippolito, Anna Huang, Curtis Hawthorne, and Douglas Eck. Infilling piano performances. In *NIPS Workshop on Machine Learning for Creativity and Design*, 2018. 2, 5

Felix Kreuk, Gabriel Synnaeve, Adam Polyak, Uriel Singer, Alexandre Défossez, Jade Copet, Devi Parikh, Yaniv Taigman, and Yossi Adi. Audiogen: Textually guided audio generation. *arXiv preprint arXiv:2209.15352*, 2022. 6

Mina Lee, Percy Liang, and Qian Yang. Coauthor: Designing a human-ai collaborative writing dataset for exploring language model capabilities. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, 2022. A.5

Jordan Lenchitz. Reconsidering quantization in MIR. In *International Society for Music Information Retrieval*, 2021. A.3

Robert Levinson. The case for computer chess engines in the classroom. *International Journal of Technology, Knowledge and Society*, 2011. A.4

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2017. 6, G

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2018. 6, G

Ryan Louie, Andy Coenen, Cheng Zhi Anna Huang, Michael Terry, and Carrie J Cai. Novice-ai music co-creation via ai-steering tools for deep generative models. In *CHI Conference on Human Factors in Computing Systems*, 2020. 1

Andreas Lugmayr, Martin Danelljan, Andres Romero, Fisher Yu, Radu Timofte, and Luc Van Gool. Repaint: Inpainting using denoising diffusion probabilistic models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022. 5

Sancho McCann. Copyright throughout a creative ai pipeline. *Canadian Journal of Law & Technology*, 2021. A.2

Hongyuan Mei and Jason M Eisner. The neural hawkes process: A neurally self-modulating multivariate point process. *Advances in Neural Information Processing Systems*, 2017. 2

Fatemehsadat Mireshghallah, Kartik Goyal, and Taylor Berg-Kirkpatrick. Mix and match: Learning-free controllable text generationusing energy language models. In *Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2022. 5

Gautam Mittal, Jesse Engel, Curtis Hawthorne, and Ian Simon. Symbolic music generation with diffusion models. In *International Society for Music Information Retrieval*, 2021. 5

Changhoon Oh, Jungwoo Song, Jinhan Choi, Seonghyeon Kim, Sungwoo Lee, and Bongwon Suh. I lead, you help but only with enough details: Understanding user experience of co-creation with artificial intelligence. In *CHI Conference on Human Factors in Computing Systems*, 2018. 6

Takahiro Omi, Kazuyuki Aihara, et al. Fully neural network based model for general temporal point processes. *Advances in Neural Information Processing Systems*, 2019. 2

Sageev Oore, Ian Simon, Sander Dieleman, Douglas Eck, and Karen Simonyan. This time with feeling: Learning expressive musical performance. *Neural Computing and Applications*, 2020. 2

Ashis Pati, Alexander Lerch, and Gaëtan Hadjeres. Learning to traverse latent spaces for musical score inpainting. In *International Society for Music Information Retrieval*, 2019. 2

Christine Payne. Musenet, 2019. URL openai.com/blog/musenet. 2

Ofir Press, Noah A Smith, and Mike Lewis. Shortformer: Better language modeling using shorter inputs. In *Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 2021. 3.3, 6

Ofir Press, Noah Smith, and Mike Lewis. Train short, test long: Attention with linear biases enables input length extrapolation. In *International Conference on Learning Representations*, 2022. 6

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 2019. 4

Jack W Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, et al. Scaling language models: Methods, analysis & insights from training gopher. *arXiv preprint arXiv:2112.11446*, 2021. 6

Colin Raffel. *Learning-based methods for comparing sequences, with applications to audio-to-midi alignment and matching*. Columbia University, 2016. 1, 4

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 2020. 1

Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 2022. 6

Yi Ren, Jinzheng He, Xu Tan, Tao Qin, Zhou Zhao, and Tie-Yan Liu. Popmag: Pop music accompaniment generation. In *ACM International Conference on Multimedia*, 2020. 4.2

Seungyeon Rhyu, Hyeonseok Choi, Sarah Kim, and Kyogu Lee. Translating melody to chord: Structured and flexible harmonization of melody with transformer. *IEEE Access*, 2022. 4.2

Quentin Roy, Futian Zhang, and Daniel Vogel. Automation accuracy is good, but high controllability may be better. In *CHI Conference on Human Factors in Computing Systems*, 2019. 6

Tim Salimans, Andrej Karpathy, Xi Chen, and Diederik P Kingma. Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications. In *International Conference on Learning Representations*, 2017. F

Flavio Schneider, Zhijing Jin, and Bernhard Schölkopf. Moûsai: Text-to-music generation with long-context latent diffusion. *arXiv preprint arXiv:2301.11757*, 2023. 6

Christoph Schuhmann, Romain Beaumont, Richard Vencu, Cade Gordon, Ross Wightman, Mehdi Cherti, Theo Coombes, Aarush Katta, Clayton Mullis, Mitchell Wortsman, et al. Laion-5b: An open large-scale dataset for training next generation image-text models. In *Neural Information Processing Systems Track on Datasets and Benchmarks*, 2022. 4

Oleksandr Shchur, Marin Biloš, and Stephan Günnemann. Intensity-free learning of temporal point processes. In *International Conference on Learning Representations*, 2020. 2

Yi-Jen Shih, Shih-Lun Wu, Frank Zalkow, Meinard Muller, and Yi-Hsuan Yang. Theme transformer: Symbolic music generation with theme-conditioned transformer. *IEEE Transactions on Multimedia*, 2022. 4.2

Ian Simon, Dan Morris, and Sumit Basu. Mysong: automatic accompaniment generation for vocal melodies. In *CHI Conference on Human Factors in Computing Systems*, 2008. 4.2

Benjamin LW Sobel. Artificial intelligence's fair use crisis. *Columbia Journal of Law & the Arts*, 2017. A.2

Gowthami Somepalli, Vasu Singla, Micah Goldblum, Jonas Geiping, and Tom Goldstein. Diffusion art or digital forgery? investigating data replication in diffusion models. *arXiv preprint arXiv:2212.03860*, 2022. A.2

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems*, 2014. 1

Chih-Pin Tan, Chin-Jui Chang, Alvin WY Su, and Yi-Hsuan Yang. Music score expansion with variable-length infilling. *arXiv preprint arXiv:2111.06046*, 2021. 2

Chih-Pin Tan, Alvin WY Su, and Yi-Hsuan Yang. Melody infilling with user-provided structural context. *arXiv preprint arXiv:2210.02829*, 2022. 2

John Thickstun, Zaid Harchaoui, Dean P Foster, and Sham M Kakade. Coupled recurrent models for polyphonic music composition. In *International Society for Music Information Retrieval*, 2019. 1, F

Benigno Uria, Iain Murray, and Hugo Larochelle. A deep and tractable density estimator. In *International Conference on Machine Learning*, 2014. 5

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017. 4

Venkata Subramanian Viraraghavan, Rangarajan Aravind, and Hema A Murthy. Precision of sung notes in carnatic music. In *International Society for Music Information Retrieval*, 2018. A.3

Dimitri von Rütte, Luca Biggio, Yannic Kilcher, and Thomas Hoffman. FIGARO: Generating symbolic music with fine-grained artistic control. In *International Conference on Learning Representations*, 2023. 4.2, 2, H, 8

Nikhil Vyas, Sham Kakade, and Boaz Barak. Provable copyright protection for generative models. *arXiv preprint arXiv:2302.10870*, 2023. A.2

Ziyu Wang and Gus Xia. Musebert: Pre-training music representation for music understanding and controllable generation. In *International Society for Music Information Retrieval*, 2021. 5

Norbert Wiener, Norbert Wiener, Cyberneticist Mathematician, Norbert Wiener, Norbert Wiener, and Cybernéticien Mathématicien. *Extrapolation, interpolation, and smoothing of stationary time series: with engineering applications*. MIT press Cambridge, MA, 1949. 3

Shangda Wu, Yue Yang, Zhaowen Wang, Xiaobing Li, and Maosong Sun. Melody harmonization with controllable harmonic rhythm. *arXiv preprint arXiv:2112.11122*, 2021. 4.2

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in Neural Information Processing Systems*, 2019. 5

Yin-Cheng Yeh, Wen-Yi Hsiao, Satoru Fukayama, Tetsuro Kitahara, Benjamin Genchel, Hao-Min Liu, Hao-Wen Dong, Yian Chen, Terence Leong, and Yi-Hsuan Yang. Automatic melody harmonization with triad chords: A comparative study. *Journal of New Music Research*, 2021. 4.2

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big bird: Transformers for longer sequences. *Advances in Neural Information Processing Systems*, 2020. 6

Mingliang Zeng, Xu Tan, Rui Wang, Zeqian Ju, Tao Qin, and Tie-Yan Liu. Musicbert: Symbolic music understanding with large-scale pre-training. In *Findings of the Association for Computational Linguistics*, 2021. 2.1

Hongyuan Zhu, Qi Liu, Nicholas Jing Yuan, Chuan Qin, Jiawei Li, Kun Zhang, Guang Zhou, Furu Wei, Yuanchun Xu, and Enhong Chen. Xiaoice band: A melody and arrangement generation framework for pop music. In *SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018. 4.2

Wanrong Zhu, Zhiting Hu, and Eric Xing. Text infilling. *arXiv preprint arXiv:1901.00158*, 2019. 5

Simiao Zuo, Haoming Jiang, Zichong Li, Tuo Zhao, and Hongyuan Zha. Transformer hawkes process. In *International Conference on Machine Learning*, 2020. 2

# A Ethical Considerations for Generative Models of Music

We find it likely that generative models of music will become widely deployed and used within the next several years (if not sooner). We are particularly concerned about the economic implications of this technology: how will these models affect labor markets for creative work (Section A.1)? We are uncertain about the legal status of these models: how will music created using models be treated by intellectual property and copyright law (Section A.2)? We are also curious how the advent of these models will change our perceptions and experience of art and creativity. Will widespread deployment of models trained on Western music reinforce existing Western hegemony in music culture (Section A.3)? Could the use of these models lead to creative stagnation, or loss of interest in music as an art form (Section A.4)? Analogous questions and concerns have been raised in the broader context of research on generative models, but the focus of this discussion is often grounded in generation of language or images. These discussions are worth revisiting in the specific context of music generation.

## A.1 The Creative Economy and Labor Markets

Music composition has economic value. What happens to the music economy—and to the people who rely on it for their livelihoods—if its creative product can be automatically generated by a machine? Economic research suggests that new technologies exert two countervailing effects on the labor market: a *displacement effect* whereby automation reduces demand for labor, and a *productivity effect* whereby greater productivity increases demand for labor (Acemoglu & Restrepo, 2018). It is unclear to us what the net effect of these two forces will be on the size of the labor market for music composition. If latent demand for new music is high, then decreasing the cost of music production could lead to more employment opportunities for composers. That said, this reconfiguration of the music economy might be disruptive, and success in the new music economy could require a new skillset that is difficult (or unappealing) for current workers to acquire. We suspect that generative music technology will affect the current music economy, and may displace some workers.

One response to concerns about labor displacement is that researchers should focus on building productivity-enhancing systems that augment human capabilities, rather than automating them. This is an appealing perspective for researchers, because it suggests that we can actively steer the development of generative technologies towards pro-social outcomes. Indeed, the methods developed in this paper facilitate the creation of controllable generative models that support the human creative process, rather than automating it. That said, augmentative and productivity-enhancing technologies can still cause labor displacement. A system that augments human capabilities can be used to replace a high-paid, high-skill worker with a low-paid, low-skill worker; or to replace a team of workers with a single highly-productive worker. Augmentation can also change the nature of creative work, potentially devaluing or automating aspects of the job that bring a worker joy.

## A.2 Copyright

While the use of copyrighted data to train models is often defensible under fair use doctrine (2d Circuit, 2015), the use of such data for *generative* modeling is an open legal question (Sobel, 2017; Henderson et al., 2023). To evaluate the copyright status of generative music models, it may be helpful to distinguish between the model itself (i.e. the parameters) and outputs generated by the model (McCann, 2021). It is unclear how the law will ultimately view (the parameters of) a generative model. Is the model a transformative fair use of the training data? Is the model itself a creative construct subject to copyright?

Regarding the outputs of a generative music model: generative models have been shown to plagiarize training data in both the language (Carlini et al., 2022) and vision (Somepalli et al., 2022) domains. Based on this evidence, we are certain that generative models of music are similarly capable of plagiarism. Technical mitigation strategies to prevent this behavior based on differential privacy or near access-freeness are nascent (Vyas et al., 2023), as are socio-technical strategies, such as providing copyright holders an opt-out mechanism to exclude their content from training datasets (Henderson et al., 2023). The models trained in

this paper provide no technical guarantees against plagiarism. Therefore, any outputs of this model should be presumed to risk infringing on copyrighted material.

The Lakh MIDI dataset used to train models for this paper is licensed under the Creative Commons CC-BY 4.0 terms, which ostensibly allows free redistribution, reuse, remixing, and transformation of its content for any purpose. However, the contents of this dataset consists of MIDI files aggregated from a variety of sources and subject to their own (heterogeneous, mostly undocumented) copyright terms. In many cases, these MIDI files are derivative work, transcribed from source material (e.g., pop music) which is itself subject to copyright. Therefore, we presume that the copyright status of Lakh MIDI is more restrictive than its license would suggest.

### A.3 Western Bias

While the methods developed in this paper are broadly applicable to modeling diverse musical traditions, the MIDI data format and the Lakh MIDI dataset used to train the anticipatory music transformer both impose a strong bias towards the Western music tradition. In Section 2 we adopted a musical vocabulary that describes musical pitch according to the 12-tone chromatic scale, which we inherit from the MIDI format of our training data. Models trained using this vocabulary are incapable of expressing music outside the 12-tone scale. This excludes, for example, the Koron and Sori quarter tones of Persian Dastgah music (Abdoli, 2011), the 53 Koma of Turkish makam (Benetos & Holzapfel, 2013), and the continuous-pitch gamaka appearing in Carnatic music (Viraraghavan et al., 2018). For further discussion of the consequences of modeling using a 12-tone discrete scale, see Lenchitz (2021).

The Lakh MIDI dataset used to train anticipatory music transformers in Section 4 consists almost exclusively of Western music. This inevitably biases the model's predictions towards infilling completions that follow Western rules of composition. We encourage work towards building models for other music traditions. We believe that the primary obstruction to this work will be the limited availability of data. In this sense, extending the work presented in this paper to other music traditions may be analogous to work in the natural language processing community on low-resource language modeling (Hedderich et al., 2021).

### A.4 Models and Art

If generated music becomes too good, might people lose interest in creating music altogether? We find this unlikely. Our appreciation and emotional attachment to art is closely connected to its provenance, the story behind its creation, and the story of its creator. Machine-generated music is unlikely to supplant high art, although we expect that many artists may incorporate generative tools into their creative processes. Economic pressures aside, artists are free to use generative tools or not. We expect many artists to embrace these tools, but surely others will eschew them in favor of more traditional creative processes.

From a humanist perspective, we are optimistic for a future with generative models. Due to the high skill required to compose music, most people are currently unable to create music at all. Lowering the barrier to music creation is more likely to increase interest in music than to decrease it. A strong generative model of music can be repurposed as a learning tool, providing feedback on compositions and ideas for how to improve a composition (Huang et al., 2016). We draw an analogy to modern chess engines, which play the game of chess at a far higher level than even the best human players. Despite this, public interest in the game of chess has never been higher (CHESScom, 2023): we celebrate human performance, and it is irrelevant that the computer on our phone could beat us every time. Chess engines offer universal access to a form of chess education: whereas chess development previously required tutoring by a strong chess player (requiring money and access) young players today can learn to play well by inspecting how the engine plays (Levinson, 2011). We anticipate similar opportunities for generative models to increase access to music education.

### A.5 Releasing the Anticipatory Music Transformer

We see many opportunities for generative models to support human creativity and expression. At the same time, we are concerned for the economic prospects of current participants in the music labor market, whose work may disrupted or displaced by the deployment of generative music technologies. In our work, we strive

to develop methods that maximize the humanistic potential for generative models of music. Specifically, we focus on controllable generation, which places the expressive power of these models under the control of their users. We focus on generation of intermediate *symbolic* music representations, which could be integrated as an assistive tool—analogous to a writing assistant (Lee et al., 2022)—into the music sequencing and synthesis workflow of a modern digital audio workstation. In contrast to audio models, symbolic music generation could provide a less disruptive integration of generative music technologies into existing workflows and thus support current participants in the music labor market. Nevertheless, we find it impossible to weigh the opportunities presented by these models against the challenges this technology might pose for workers in the music economy. Criticisms of capitalist economic structures—that accrue the windfalls of automation and productivity-enhancing technologies neither to their inventors nor to workers whose labor they displace—are beyond the scope of this paper. We welcome feedback on our decision to pursue this line of research; we hope to foster a discussion of how we can steer future research in this area towards methods that serve and support composers and musicians.

## B Licensing

We release the code for constructing anticipatory infilling models and weights for the models discussed in this paper under the Apache License, Version 2.0.

## C Encoding Details for Anticipatory Infilling Models.

The tokenization of training examples (Section 3.3) for anticipatory infilling models using an arrival-time encoding of events (Definition 2.2) is described by Definition C.1. As discussed in Section 3.5, we double the base vocabulary to distinguish between anticipated events and non-anticipated events. We also include a sequence separator token and global control codes $\mathbf{z} \in \{\texttt{AR}, \texttt{AAR}\}$ (Section 3.3) as well as the `REST` token (Section 3.2).

**Definition C.1.** (Arrival-Time Training Example) Let $\mathbf{a}_{1:(M-1)/3}$ be a training example, possibly re-ordered according to Definition 3.1 for infilling. An *arrival-time tokenized training example* is a sequence $\mathbf{x}_{1:M}$, defined as follows. We define two special tokens $\texttt{REST} = 27512$ and $\texttt{SEP} = 55025$, and two control tokens $\texttt{AR} = 55026$ and $\texttt{AAR} = 55027$. Separation between two sequences is indicated by a triple of three `SEP` tokens. The first token $\mathbf{x}_1$ in every training example encodes the global control code $\mathbf{z}$:

$$\mathbf{x}_1 \in \{\texttt{AR}, \texttt{AAR}\}, \qquad \text{(the anticipation control code).} \tag{18}$$

If $\mathbf{a}_i$ is a non-anticipated event then

$$\mathbf{x}_{3i-2} \in \{0, \ldots, 10000\} \cup \{\texttt{SEP}\}, \qquad (0 - 100\text{s, 10ms quantized}) \tag{19}$$
$$\mathbf{x}_{3i-1} \in \{10000, 10000 + 1000\} \cup \{\texttt{SEP}\}, \qquad (0 - 10\text{s, 10ms quantized}) \tag{20}$$
$$\mathbf{x}_{3i} \in \{11000, \ldots, 11000 + 16512\} \cup \{\texttt{SEP}\} \cup \{\texttt{REST}\}, \qquad (\text{instruments} \times \text{pitches}). \tag{21}$$

Otherwise (if $\mathbf{a}_i$ is an anticipated event)

$$\mathbf{x}_{3i-2} \in \{27513, \ldots, 27513 + 10000\} \cup \{\texttt{SEP}\}, \qquad (0 - 100\text{s, 10ms quantized}) \tag{22}$$
$$\mathbf{x}_{3i-1} \in \{37513, 37513 + 1000\} \cup \{\texttt{SEP}\}, \qquad (0 - 10\text{s, 10ms quantized}) \tag{23}$$
$$\mathbf{x}_{3i} \in \{38513, \ldots, 38513 + 16512\} \cup \{\texttt{SEP}\} \cup \{\texttt{REST}\}, \qquad (\text{instruments} \times \text{pitches}). \tag{24}$$

The total vocabulary size is 55028.

**Example C.2.** The arrival-time tokenization (processed for autoregressive training) of the first for bars of the lullaby "Twinkle, Twinkle, Little Star," played on a piano at tempo quarter=120:

$\mathbf{x}_{0:46}$ = [55026, 55025, 55025, 55025, 0, 10048, 11060, 50, 10048, 11060, 100, 10048, 11067, 150, 10048, 11067, 200, 10048, 11069, 250, 10048, 11069, 300, 10095, 11067, 400, 10048, 11065, 450, 10048, 11065, 500, 10048, 11064, 550, 10048, 11064, 600, 10048, 11062, 650, 10048, 11062, 700, 10095, 11060].

See Table 4 for a structured description of this sequence.

The tokenization of training sequences for autoregressive models using an interarrival-time encoding (Defintion 2.3) is described by Definition C.3. We double the base vocabulary of notes and instruments to distinguish between onsets and offsets. We truncate interarrival times longer than 10 seconds (to 10 seconds).

**Definition C.3.** (Interarrival-Time Training Example) Let $\mathbf{e}_{1:(M/4)}$ be a training example of events (Definition 2.1). An *interarrival-time tokenized training example* is a sequence $\mathbf{x}_{1:M}$, defined as follows. We define a single special token `SEP` = 34024. Separation between two sequences is indicated by a single `SEP` token. Using the notation $\mathbf{x}'_{1:(M/2)}$ from Definition 2.3, if $\mathbf{x}'_i$ is an onset,

$$\mathbf{x}_{2i} \in \{1000, \ldots, 1000 + 16512\}, \qquad \text{(instruments} \times \text{pitches)}. \tag{25}$$

If $\mathbf{x}'_i$ is an offset,

$$\mathbf{x}_{2i} \in \{17512, \ldots, 17512 + 16512\}, \qquad \text{(instruments} \times \text{pitches)}. \tag{26}$$

And regardless,

$$\mathbf{x}_{2i+1} \in \{0, 1000\}, \qquad (0 - 10\text{s, 10ms quantized}). \tag{27}$$

As described in Definition 2.3, we omit tokens $\mathbf{x}_{2i+1} = 0$. The total vocabulary size is 34025.

**Example C.4.** The interarrival-time tokenization of the first for bars of the lullaby "Twinkle, Twinkle, Little Star," played on a piano at tempo quarter=120:

$\mathbf{x}_{0:56}$ = [34024, 1060, 48, 17572, 2, 1060, 48, 17572, 2, 1067, 48, 17579, 2, 1067, 48, 17579, 2, 1069, 48, 17581, 2, 1069, 48, 17581, 2, 1067, 95, 17579, 5, 1065, 48, 17577, 2, 1065, 48, 17577, 2, 1064, 48, 17576, 2, 1064, 48, 17576, 2, 1062, 48, 17574, 2, 1062, 48, 17574, 2, 1060, 95, 17572].

In this case there are no interarrival times of length zero and this tokenization is slightly less compact than arrival-time tokenization. But if we were to extend the length of each note to a full beat, the interarrival-time tokenization becomes:

$\mathbf{x}_{0:43}$ = [34024, 1060, 50, 17572, 1060, 50, 17572, 1067, 50, 17579, 1067, 50, 17579, 1069, 50, 17581, 1069, 50, 17581, 1067, 100, 17579, 1065, 50, 17577, 1065, 50, 17577, 1064, 50, 17576, 1064, 50, 17576, 1062, 50, 17574, 1062, 50, 17574, 1060, 100, 17572]

And in this case, interarrival-time tokenization is slightly more compact.

## D A Prior over Music Infilling Controls

We propose three types of anticipation, i.e., distributions over events to condition on as controls. First we propose *span anticipation*, whereby we anticipate all tokens in a given span in order to explicitly promote the model's ability to fill-in-the-middle. Second, we propose *instrument anticipation*, whereby we anticipate all tokens except for a specified instrumental part, supporting a workflow whereby supplemental instrumental parts are generated to complement a pre-existing ensemble. And third, we propose *random anticipation*, whereby we randomly anticipate events at some fixed rate to accommodate a broader possible range of user-specified anticipation patterns.

We apply apply these patterns of anticipation to training data according to the following distributions:

- **Span anticipation**. We randomly anticipate consecutive subsequences of events spanning $\delta$ seconds, at an exponential rate $\lambda$. We fix $\lambda = .05$, and at interarrival times $i \sim \text{Exp}(\lambda)$ we anticipate the events $\mathbf{e}_{i:j}$ where $j = \min\{t_j : t_j > t_i + \delta\}$.

Table 4: Arrival-time tokenization of "Twinkle, Twinkle, Little Star," played on a piano at tempo quarter=120. For clarity, we group the sequence of tokens tokens into triplets (one event per row).

| Token Values | | | Event Description | | | |
|---|---|---|---|---|---|---|
| Arrival Time | Duration | Note | **t** | **d** | **p** | **k** |
| 55,025 | 55,025 | 55,025 | Sequence Separator Event | | | |
| 0 | 10,048 | 11,060 | 0s | 480ms | C4 | piano |
| 50 | 10,048 | 11,060 | 0.5s | 480ms | C4 | piano |
| 100 | 10,048 | 11,067 | 1s | 480ms | G4 | piano |
| 150 | 10,048 | 11,067 | 1.5s | 480ms | G4 | piano |
| 200 | 10,048 | 11,069 | 2s | 480ms | A4 | piano |
| 250 | 10,048 | 11,069 | 2.5s | 480ms | A4 | piano |
| 300 | 10,095 | 11,067 | 3.5s | 950ms | G4 | piano |
| 400 | 10,048 | 11,065 | 4s | 480ms | F4 | piano |
| 450 | 10,048 | 11,065 | 4.5s | 480ms | F4 | piano |
| 500 | 10,048 | 11,064 | 5s | 480ms | E4 | piano |
| 550 | 10,048 | 11,064 | 5.5s | 480ms | E4 | piano |
| 600 | 10,048 | 11,062 | 6s | 480ms | D4 | piano |
| 650 | 10,048 | 11,062 | 6.5s | 480ms | D4 | piano |
| 700 | 10,095 | 11,060 | 7s | 950ms | C4 | piano |

- **Instrument anticipation**. For an event sequence with $J$ unique instrument parts, we uniformly sample $j \in \{1, \ldots, J-1\}$ and randomly sample $j$ instrumental parts without replacement. We anticipate all events in the sequence associated with these $j$ parts.
- **Random anticipation**. We uniformly sample a rate $r \in \{0.1, \ldots, 0.9\}$ for each event sequence. We randomly anticipate an $r$ fraction of events in the sequence.

We balance the overall training distribution using the following mix: 10% without anticipation (standard autoregressive training), 10% with span anticipation, 40% with instrument anticipation, and 40% with random anticipation.

These anticipation patterns facilitate interaction with an anticipatory music transformer via user-specified control sequences. For example, the accompaniment task evaluated in Section 4 is modeled by instrument anticipation when $j = 1$. Note that the prior over infilling controls facilitates much more general interaction patterns than the prompt continuation and melodic accompaniment tasks studied in Section 4. We defer further study of the capabilities of anticipatory infilling models to future work.

To train an anticipatory infilling model, we augment the training dataset using the distribution of anticipation patterns specified above. We perform these augmentations during preprocessing, resulting in an augmented dataset derived from the original data. In particular, augment the Lakh MIDI dataset by a factor of 30. Using the prior distribution described above, this augmented dataset contains (i) 3 copies of the original dataset, verbatim; (ii) 3 augmented copies of the dataset with different random anticipated spans; (iii) 12 augmented copies of the dataset with different randomly anticipated instrument subsets for each event sequence; and (iv) 12 augmented copies of the dataset with different randomly anticipated events at different rates for each event sequence.

Augmenting by a factor of 30 results in an arrival-time encoded Lakh MIDI train set of approximately 52B tokens. For training examples of length 1024 and batch size 512, this results in approximately one epoch per 100,000 optimization steps. For models trained with larger step counts, we take multiple passes over the augmented dataset. Given the relatively small size of the Lakh MIDI dataset, anticipation may be of interest as a regularization technique (see Appendix K for some evidence of possible regularizing effects attributable to anticipation). In this case, it could be fruitful to (i) increase the augmentation factor further during training, and (ii) place a smaller weight on training without anticipation, to minimize the amount of duplication of the original (unaugmented) dataset within the augmented dataset.

## E Details of the Lakh MIDI Dataset

The Lakh MIDI dataset includes no metadata, so we have little fine-grained or quantitative insight into the origins and contents of this dataset. Based on a randomly sampling of the dataset's contents, we observe that it contains many arrangements of modern pop music, transcripts of classical western compositions, and original compositions (with varying degrees of quality).

**Preprocessing** Out of the 178,561 sequences in the Lakh MIDI dataset, we were able to successfully parse 174,046 sequences.[2] We discard 7032 sequences that are shorter than 100 events or 10 seconds in length. We also discard 40 event sequences that are longer than one hour in length (inspection reveals that many of these sequences are corrupt). Finally, we discard 2227 sequences that contain more than 16 unique instrument parts: representing music with more than 16 parts as MIDI (for synthesizing outputs) requires multiplexing multiple instruments onto MIDI channels; we avoid this complexity by simply excluding very large ensembles. The resulting dataset consists of 164,747 event sequences. We split this dataset in to train, validation, and test splits according to the leading hexadecimal digit of each file's MD5 hash:

- Train: hashes `0`–`d`, 144,202 event sequences, 7827 hours of music.
- Validation: hash `e`, 10,212 event sequences, 555 hours of music.
- Test: hash `f`, 10,333 event sequences, 561 hours of music.

We tokenize the dataset using the arrival-time and interarrival-time encodings described in Section 2.1. As seen in Table 5, the interarrival-time tokenization is slightly more compact. To illustrate the long, variable-rate nature of symbolic music event sequences, we plot the distributions of sequence length (Figure 2) and event rate (Figure 3) for the Lakh MIDI validation split.

Table 5: The number of tokens in the Lakh MIDI dataset, using tokenizations described in Section 2.1.

| Encoding | Train | Validation | Test | Overall |
|---|---|---|---|---|
| arrival | 1,741,830,387 | 123,785,046 | 125,050,497 | 1,990,665,930 |
| interarrival | 1,612,129,280 | 114,519,040 | 115,409,920 | 1,842,058,240 |

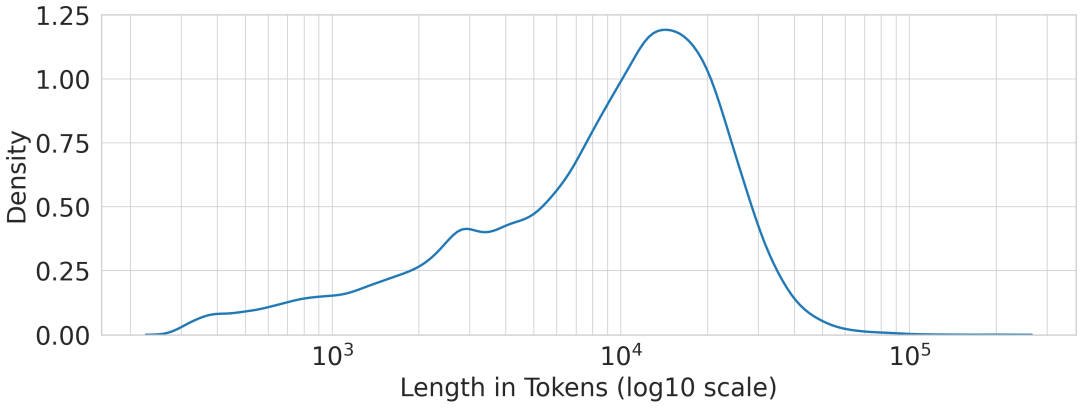


Figure 2: The distribution of sequence lengths calculated for the arrival-time tokenized Lakh MIDI validation split. Mean sequence length is 12,071 tokens, with a standard deviation of 9711 tokens.

[2]We parse Midi files using the Mido library: https://github.com/mido/mido.

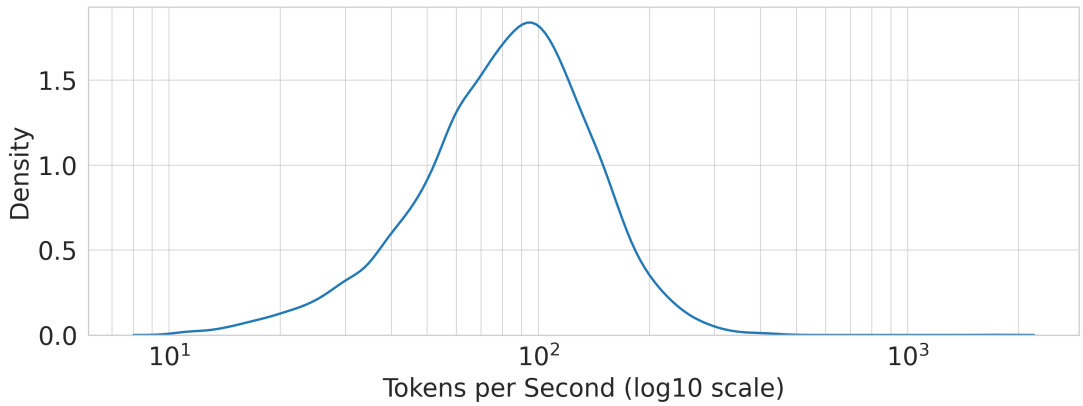


Figure 3: The distribution of instantaneous tokens/second calculated for the arrival-time tokenized Lakh MIDI validation split. Mean instantaneous tokens/second for the Lakh MIDI dataset is 68 with a standard deviation of 51 events/second.

## F Measuring Cross-Entropy in Bits per Second

Because the arrival-time tokenization (Definition 2.2) and the interarrival-time tokenization (Defintion 2.3) describe nearly equivalent information, the log-loss of models trained using either encoding can be meaningfully compared via a unit conversion. To make comparison agnostic to encoding, we report losses in units of bits per second: this is the total log-loss of the test set, divided by the number of seconds of music in the test set. Concretely, given a per-token loss $L$ reported in nats per token, conversions for interrarival time and arrival time encodings of the test set defined in Section E are:

$$\text{bps}_{\text{arrival}}(L) = L \times \frac{1}{\log(2)} \times (\texttt{test_tokens}_{\text{arrival}}/(\texttt{test_hours} \times 3600)), \tag{28}$$

$$\text{bps}_{\texttt{interarrival}}(L) = L \times \frac{1}{\log(2)} \times (\texttt{test_tokens}_{\text{interarrival}}/(\texttt{test_hours} \times 3600)). \tag{29}$$

Concretely, for the Lakh MIDI dataset, from Table 5 we have $\texttt{test_tokens}_{\text{arrival}} = 125{,}050{,}497$, $\texttt{text_tokens}_{\text{interarrival}} = 115{,}409{,}920$, and the length of the test set (independent of encoding) is $\texttt{test_hours} = 560.98$ hours. The remaining factors convert units from nats to bits (the log(2) factor) and from hours to seconds (the 3600 factor). This normalized form of the log-loss is analogous to the bits per dimension loss commonly reported for image generation (Salimans et al., 2017) and the bits per beat loss for musical scores (Thickstun et al., 2019).

Truncations during tokenization (see Section 2.1) result in slightly different information content between an arrival-time tokenized sequence and the corresponding interarrival-time tokenized sequence. In particular, 0.05% of note durations in the Lakh MIDI dataset exceed 10 seconds, and are truncated (to 10 seconds) under arrival-time tokenization. Likewise, a total of 3947 interarrival times (0.00%) in the Lakh MIDI dataset exceed 10 seconds, and are truncated (to 10 seconds) under interarrival-time tokenization. Finally, 0.09% of training examples under arrival-time tokenization exceed 100 seconds in length, and are discarded from the arrival-time tokenized dataset. These instances are sufficiently rare that they exert a negligible effect on the information content (and therefore on bits per second comparisons) between the arrival-time and interarrival-time tokenized sequences.

## G Hyperparameters and Resources

All models are parameterized by standard, decoder-only causally masked transformers with GeLU nonlinearities (Hendrycks & Gimpel, 2016). The models are implemented in Jax (Bradbury et al., 2018) and trained on Google TPU v3 hardware. We observe that large-scale pseudorandom number generation in Jax

Table 6: Model Configurations.

| | Small | Medium | Large |
|---|---|---|---|
| **Architecture Hyperparameters** | | | |
| Layers | 12 | 24 | 36 |
| Attention Heads | 12 | 16 | 20 |
| Hidden Dimensions | 768 | 1024 | 1280 |
| Sequence Length | | 1024 tokens | |
| Residual Dropout | | 0.1 | |
| Embedding Dropout | | 0.1 | |
| Attention Dropout | | 0.0 | |
| Weight Decay | | 0.1 | |
| **Optimizer Hyperparameters** | | | |
| Max Learning Rate | 0.0006 | 0.0003 | 0.0002 |
| Optimizer | | AdamW (Loshchilov & Hutter, 2018) | |
| $(\beta_1, \beta_2, \epsilon)$ | | $(0.9, 0.999, 1 \times 10^{-8})$ | |
| Batch Size | | 512 sequences (= 524,288 Tokens) | |
| Warmup | | 1000 steps ($\approx$ 50M tokens) | |
| Learning Rate Schedule | | Cosine decay (no restarts) (Loshchilov & Hutter, 2017) | |
| Gradient Clipping | | Clipping above $\|\nabla\| = 1$ (Rae et al., 2021) | |
| **Training Resources** | | | |
| Throughput (tokens/second) | 690,000 | 260,000 | 140,000 |
| Throughput (seconds/iteration) | 0.76 | 2.02 | 3.74 |
| Hardware | | Google TPU v3-32 pod slice | |

is slow, and therefore eschew the standard attention dropout regularization. The models are optimized using AdamW (Loshchilov & Hutter, 2018). The learning rate schedule consists of a 1000 step linear warmup to a maximum learning rate, followed by a single cycle of cosine decay (Loshchilov & Hutter, 2017) over the remaining steps to a final learning rate of zero. Following Chinchilla compute-optimality recommendations, we train each model for a number of steps that is approximately proportional to the model's size (Hoffmann et al., 2022). Configuration details of the models and optimization are presented in Table 6.

Most of these models were trained on TPU v3-32 pod slices, which in practice are approximately equivalent to a GPU machine with 8 NVIDIA A100's. Training throughput for each model configuration using a v3-32 is reported in Table 6. Conversions of these throughput statistics to wall-clock estimates of training time for the models featured in Section 4 are shown in Table 7. Thus, the total training time on v3-32's for the models featured in this paper was approximately

$$1121 \text{ hours} = 3 \times 21 \text{ hours} + 2 \times 169 \text{ hours} + 56 \text{ hours} + 112 \text{ hours} + 448 \text{ hours} + 104 \text{ hours}. \tag{30}$$

In addition to training the models featured in this paper, substantial additional TPU hours were consumed during the development phase of this research. We crudely estimate that total TPU-hours consumed for this work were approximately 3-5 times the hours reported in Equation (30).

## H Details of the FIGARO Music Transformer Baseline

We compare our models to the implementation of Music Transformer (Huang et al., 2018) described by von Rütte et al. (2023). We use the official public implementation for training and sampling from this model.[3] While both our models and this baseline are based on the Transformer architecture and trained on Lakh MIDI, there are at several notable distinctions. The FIGARO Music Transformer architecture implements the relative attention mechanism proposed by (Huang et al., 2018). The FIGARO Music Transformer inputs

[3]https://github.com/dvruette/figaro

Table 7: Estimated wall-clock training time for the Small, Medium, and Large model configurations described in Table 6, using a Google TPU v3-32 pod slice.

| Config | Training Steps | Hours of Training |
|---|---|---|
| Small | 100k | 21 |
| Small | 800k | 169 |
| Medium | 100k | 56 |
| Medium | 200k | 112 |
| Medium | 800k | 448 |
| Large | 100k | 104 |

are tokenized using a REMI encoding that accounts for metrical structure (Huang & Yang, 2020). The FIGARO Music Transformer is also a smaller-scale model than the other models considered in this paper: a six-layer transformer (approximately 30M parameters) trained for 100k iterations on sequences of length 256. Some of these factors (e.g., model scale, sequence length) clearly favor the models presented in this paper; others (e.g., relative attention) may favor the FIGARO model. With so many uncontrolled variables, we caution against drawing conclusions about individual engineering choices in the design of the FIGARO Music Transformer versus the models proposed in this paper.

While pre-trained checkpoints of the FIGARO models are available, the training and evaluation splits used for the pre-trained checkpoints are incompatible with the splits defined in Section E. Therefore, we re-train our own version of the model using hashes `0`–`d` as the training split. A comparison of our re-trained model and the reference model checkpoint are presented in Figure 8. For definitions and discussion of the evaluation metrics compared here, see von Rütte et al. (2023). Our version of the FIGARO model matches or slightly outperforms the reference model on these metrics.

Neither our arrival-time tokenization (Definition 2.2) nor FIGARO's REMI encode all the nuances of music described in, e.g., a MIDI file. Because FIGARO explicitly models the metrical structure of music, we prompt using a fixed number of measures (three bars) rather than a fixed amount of time: this is six seconds of prompt material for music in 4/4 time when quarter=120. We select three-bar prompts for the study in the range of 4-6 seconds, thus excluding music with a very fast or slow tempo. To create fair comparisons between music composed by humans, FIGARO, and the Anticipatory Music Transformer, we apply the following procedure for constructing prompts:

- Human compositions: we encode human compositions (initially expressed as MIDI) using the FIGARO tokenizer, and then re-encode these samples using our own tokenizer.
- FIGARO samples: we encode prompts using the FIGARO tokenizer, and re-encode completions of these prompts generated by FIGARO using our own tokenizer.
- Anticipatory Music Transformer: we encode prompts using the FIGARO tokenizer, re-encode the prompts using our own tokenizer, and generate completions from our own models using these re-encoded prompts.

All music is thus restricted to the musical vocabulary of our arrival-time tokenization.

Table 8: Quantitative evaluation metrics for the FIGARO implementation (von Rütte et al., 2023) of Music Transformer (Huang et al., 2018) trained on the FIGARO training data split, vs. the split defined in Section E.

| Train Split | I ↑ | C ↑ | TS ↑ | ND ↓ | P ↑ | V ↑ | D ↑ | $s_c$ ↑ | $s_g$ ↑ |
|---|---|---|---|---|---|---|---|---|---|
| (FIGARO split) | 0.191 | 0.048 | 0.751 | 2.192 | 0.563 | 0.153 | 0.312 | 0.306 | 0.510 |
| Hashes `0`—`d` | 0.207 | 0.050 | 0.770 | 1.523 | 0.564 | 0.158 | 0.289 | 0.305 | 0.517 |

## I A Baseline Autoregressive Infilling Algorithm

Algorithm 2 describes the baseline autoregressive infilling algorithm evaluated in Table 3. Without the ability to anticipate future controls, this algorithm proceeds by sampling from the model until the time of the sampled event time exceeds the time of the next control. At this point, we insert this control (and any other controls prior to the sampled event) into the sequence prior to the sampled event. We then proceed to continue sampling from the model.

This algorithm is naive because the model cannot condition on upcoming controls until after the time at which they occur. In the most optimistic scenario, because the model is a good predictor of events, we hope that it will assign high probability to the upcoming control events and therefore that the model will generate events that are consistent with (reasonable, unsurprising) controls. But ultimately there is entropy in the music process, and the model will necessarily sometimes generate events that are inconsistent with upcoming controls. To make matters worse, these inconsistencies are then written into the history and the model makes subsequent predictions conditioned on these mistakes; this has a tendency to compound the errors over time, creating highly dissonant music.

A more subtle instance of this failure of compounding errors is the algorithm's tendency to double the control events. It is relatively common for the model will exactly predict an upcoming control event. In this case, two copies of the event are written into the history. When subsequent controls are inserted into the history, the model exhibits a tendency to perpetuate this doubling, generating new events that copy every control that we insert into the history, like a vexing child copying a sibling's every word.

**Algorithm 2:** Autoregressive Sampling (Baseline)

**Parameters:** `Autoregressive model` $p$ `with context length` $M$
**Input** : `Time-localized controls` $\mathbf{u}_{1:K}$ `(monotone increasing in time)`
**Output** : `A generated sequence` $\mathbf{a}_{1:N+K}$

```
a_0 ← SEP ;                                   // A special sequence separator event
i ← 1 ;                                       // Index i tracks position in the generated sequence
k ← 0 ;                                       // Index k tracks position in the control sequence
do
    Sample e ~ p(·|a_{i−M:i−1}) ;             // Sample an event from the model
    t ← Time(e) ;                             // Get the time t of the event e
    while Time(u_k) ≤ t ;                     // While there are controls before time t
     do
        a_i ← u_k ;                           // Anticipate control u_k at index i
        i ← i + 1 ;                           // Advance to index i + 1
        k ← k + 1 ;                           // Consume control u_k
    end
    a_i ← e ;                                 // Append the newly sampled event
    i ← i + 1 ;                               // Advance to index i + 1
while a_i ≠ SEP;
return a_{1:i−1} ;                            // Index i − 1 is N + K
```

## J Details of Human Evaluation

Workers chosen to evaluate generated music were selected from a pool of crowd workers on the Amazon Mechanical Turk platform, according to the qualification procedure described below. Evaluators were provided with the interface shown in Figure 4 and instructed to judge the relative musicality of two 20-second music clips. We presented the music as audio, synthesized using Apple's DLSMusicDevice sound system. Each pair of clips begins with the same prompt: three bars for the prompted completion task and five seconds for the accompaniment task. Evaluators were instructed to judge which clip is more musical. Based on feedback from a pilot study, we clarified in the detailed instructions that we are interested in musicality in

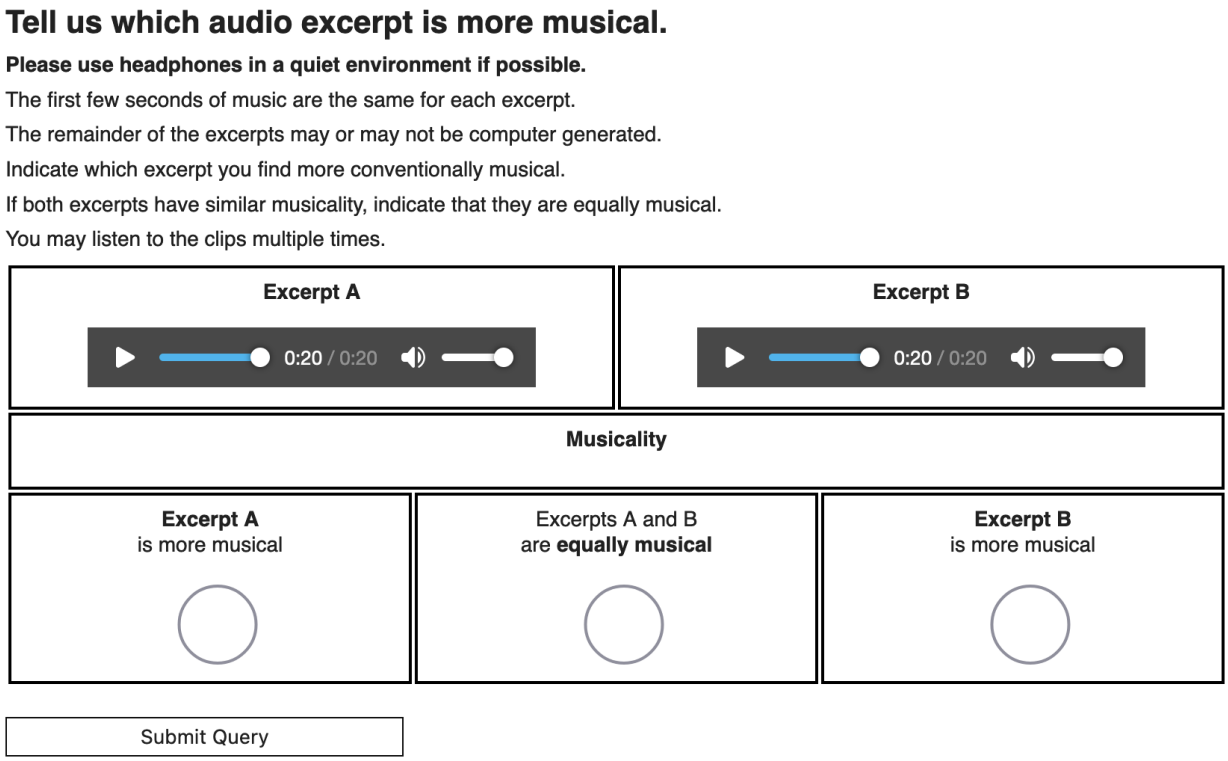


Figure 4: The interface used by evaluators to assess the relative musicality of paired music clips.

a *conventional* sense. We allowed evaluators to indicate that the two clips are equally musical, avoiding a forced choice between the two clips.

All prompts and melodies used for evaluation are sampled randomly from the test set. A notable confounder in this evaluation is that the Lakh MIDI contains many popular, recognizable songs. Study participants remarked that they recognized the origin of certain prompts. In these cases, we instructed the participants to "Please try to rate the clips based on their musicality rather than recognition." However, it may be difficult to set aside knowledge of the canonical completion in these cases.

Details of qualification, prompt continuation, and accompaniment evaluations are described below. For an illustration of the prompt continuation and accompaniment tasks, see Figure 5. For complete pairwise evaluations of models in the prompt continuation task, see Table 9. For complete pairwise evaluations of algorithms in the accompaniment tasks, see Table 10. For each task, we collected evaluations from three unique workers; we defer an analysis of worker agreement to future work.

**Qualification** Workers who correctly identified music composed by humans as more musical than random retrieval in at least 4/5 comparisons were deemed qualified to participate in subsequent model evaluations: 15 out of 20 workers who participated in qualification advanced to the main prompt continuation and accompaniment evaluations. Workers performed a total of 100 comparisons.

**Prompt Continuation** We compare continuations of four models: (i) the Small interarrival-time model (Row 1, Table 1) (ii) the Small anticipatory model (Row 3, Table 1) (iii) the large anticipatory model (Row 5, Table 1) and (iv) the FIGARO Music Transformer. We also compare to (v) human compositions (skyline). For each of 50 prompts, we create $50 \times \binom{5}{2} = 500$ pairwise comparisons between continuations (as well as the baseline and skyline) we asked three human evaluators to indicate which clip is more musical, or that the clips are equally musical. Workers performed a total of 1500 comparisons.

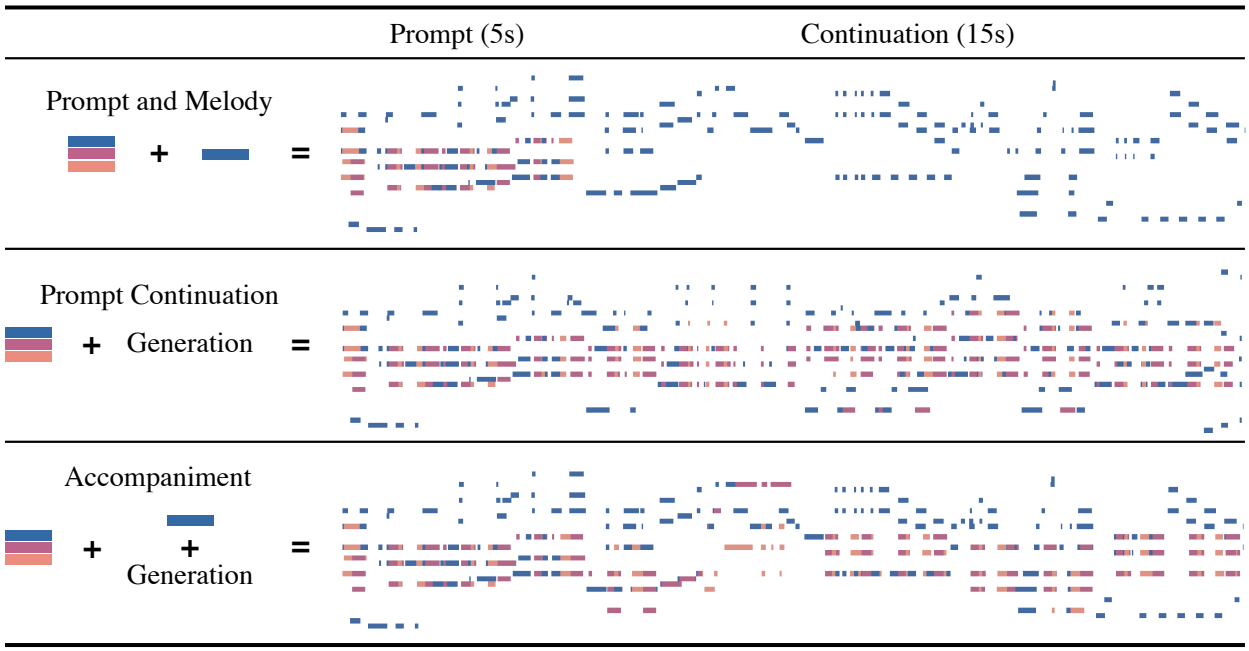


Figure 5: Visualizations of 20-second music clips. Each rectangle indicates a musical event with an onset time, duration (width), and pitch (height). Colors indicate distinct instrumental parts. For the accompaniment task, events in the blue instrumental part are used as control events. Top: a five-second prompt followed by the original continuation of only the melodic instrumental line (highest; blue). Middle: the five-second prompt followed by a generated autoregressive continuation, ignoring the original melodic line. Bottom: the prompt followed by a generated anticipatory accompaniment of the original melodic instrumental line.

**Accompaniment** We compare accompaniments using (i) anticipatory autoregressive sampling (Algorithm 1) versus (ii) baseline autoregressive sampling. We also compare to (iii) baseline completions randomly sampled from the test set and (iv) skyline human compositions. When sampling without anticipation, we insert events from the melody into the conditional history of the model once generation passes them; we describe this modified sampling procedure formally in Appendix I. For each of 20 three-bar prompts and single-part continuations, we generated accompaniments from each model. For each of the $20 \times \binom{4}{2} = 120$ pairwise comparisons between continuations (as well as the baseline and skyline) we asked three human evaluators to indicate which clip is more musical, or that the clips are equally musical. Workers performed a total of 360 comparisons.

## K Training Optimization Logs

Figure 6 and Figure 7 plot estimates of the train set and test-set losses over the course of optimization for the arrival-time models considered in Section 4. Losses are computed every 10,000 steps from logged model checkpoints; the intermediate checkpoints for all 8 of these models (as well as the small interarrival-time model) are available on request.

Training loss is significantly lower than test for all models, evidence of some amount of overfitting during many epochs of optimization on the Lakh MIDI dataset. Nevertheless, we observe better test set performance for larger models, trained longer, indicating that we have not completely saturated performance on the Lakh MIDI dataset using this scale of models and computational resources. That said, the relative test set loss improvements vs train set improvements when we increase the model size from Small to Medium are much larger than the relative gains of increasing the model size from Medium to Large: compare train vs. test loss of the Small (Row 3) Medium (Row 6) and Large (Row 9) models at 100k steps. This might suggest that we are approaching the point of diminishing returns for scaling the compute (steps) and size of models trained on Lakh MIDI, and that more data would be needed to effectively train substantially larger models.

Table 9: Human evaluation of paired completions of 3-bar musical prompts generated by different algorithms, and human compositions. P-values are reported using a Wilcoxon signed rank test. Row numbers reference Table 1.

| Model A | Model B | Wins (A) | Ties | Wins (B) | p-value |
|---|---|---|---|---|---|
| Human Composition | Medium (Row 8) | 77 | 29 | 44 | 0.0027 |
| | Music Transformer | 119 | 18 | 13 | $2.806 \times 10^{-20}$ |
| | Small (Row 3) | 84 | 23 | 43 | 0.0002 |
| | Small (Row 1) | 103 | 16 | 31 | $4.976 \times 10^{-10}$ |
| Medium (Row 8) | Music Transformer | 95 | 24 | 31 | $1.187 \times 10^{-08}$ |
| | Small (Row 3) | 65 | 27 | 58 | 0.528 |
| | Small (Row 1) | 96 | 17 | 37 | $3.122 \times 10^{-7}$ |
| Music Transformer | Small (Row 3) | 36 | 16 | 98 | $8.509 \times 10^{-08}$ |
| | Small (Row 1) | 46 | 18 | 86 | 0.0005 |
| Small (Row 3) | Small (Row 1) | 82 | 17 | 51 | 0.0071 |

Table 10: Human evaluation of paired 15-second accompaniments generated by different models, and human-composed accompaniments. P-values are reported using a Wilcoxon signed rank test.

| Algorithm A | Algorithm B | Wins (A) | Ties | Wins (B) | p-value |
|---|---|---|---|---|---|
| Human Composition | Anticipatory | 11 | 31 | 18 | 0.194 |
| | Autoregressive | 45 | 10 | 5 | $1.542 \times 10^{-08}$ |
| | Retrieval | 52 | 6 | 2 | $1.017 \times 10^{-11}$ |
| Anticipatory | Autoregressive | 47 | 6 | 7 | $5.230 \times 10^{-8}$ |
| | Retrieval | 45 | 11 | 4 | $4.709 \times 10^{-9}$ |
| Autoregressive | Retrieval | 33 | 12 | 15 | 0.009 |

For the Small models, we observe that autoregressive training results in slightly better test-set performance at 100,000 optimization steps. But at 800,000 steps the situation reverses, and the anticipatory model performs slightly better. We suspect that this is evidence of anticipatory training having a regularizing effect on the optimization. The Small, 800k step anticipatory model also exhibits some training instability in these plots. We observed similar instabilities during training the other anticipatory and autoregressive models, but they do not appear at the 10,000-step granularity pictured here. We undertook no measures to adjust for these instabilities, simply letting the optimizations run to completion without intervention.

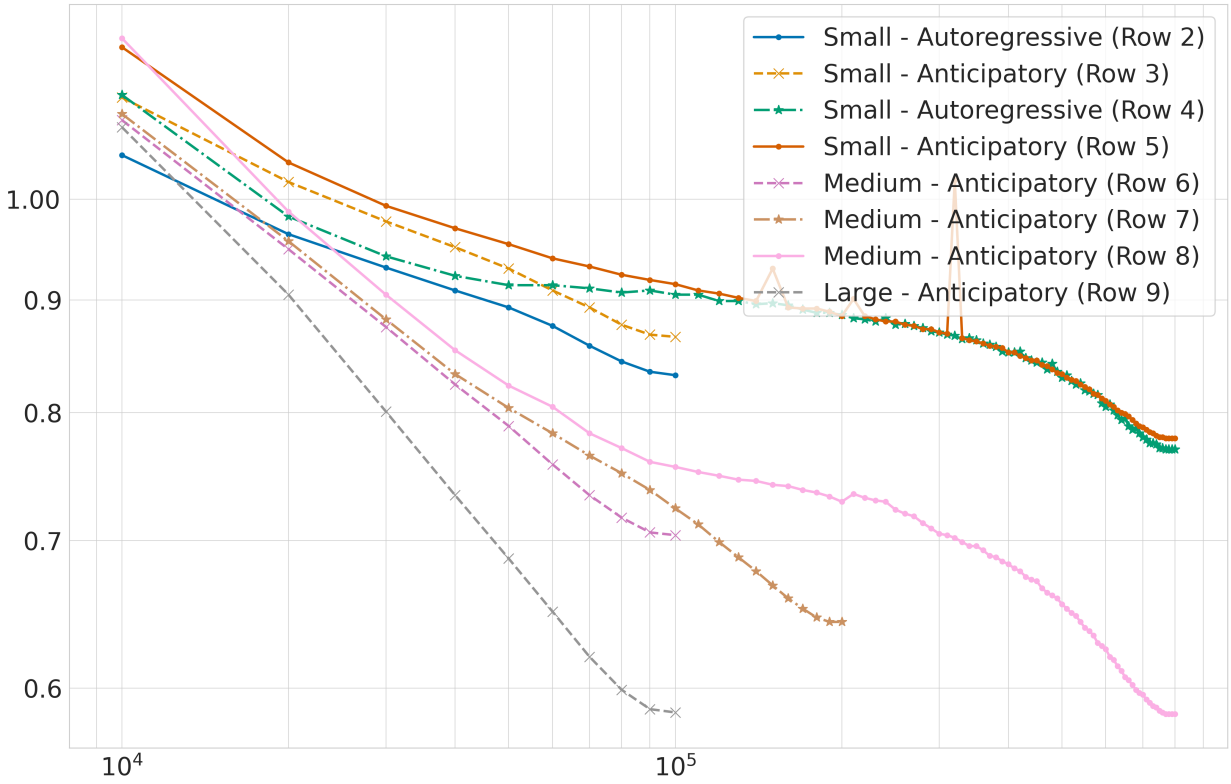


Figure 6: Mean per-token train set log-loss $L_{\text{token}}$ (in nats) of models, estimated every 10,000 steps over the course of training. The estimates are computed using a 1/100 subset of the train set. The per-token loss $L_{\text{token}}$ is related the event loss reported in Table 1 by the relationship $L_{\text{token}} = \log(L_{\mathbf{e}})/3$.

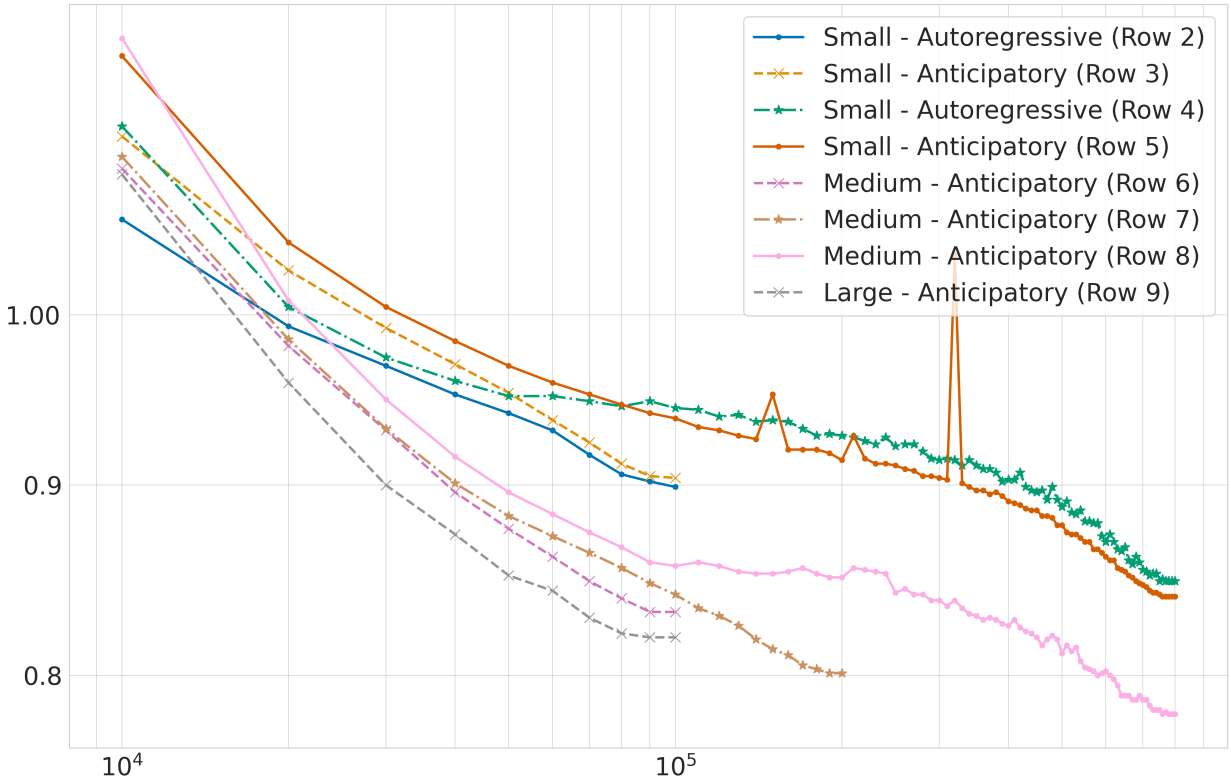


Figure 7: Mean per-token test set log-loss $L_{\text{token}}$ (in nats) of models, estimated every 10,000 steps over the course of training. The estimates are computed using a 1/10 subset of the test set. The per-token loss $L_{\text{token}}$ is related the event loss reported in Table 1 by the relationship $L_{\text{token}} = \log(L_{\mathbf{e}})/3$.