Neurosymbolic Deep Generative Models for Sequence Data with Relational Constraints

Halley Young  
Department of Computer Science  
University of Pennsylvania  
halleyy@seas.upenn.edu

Maxwell Du  
Department of Computer Science  
University of Pennsylvania  
mdu@seas.upenn.edu

Osbert Bastani  
Department of Computer Science  
University of Pennsylvania  
obastani@seas.upenn.edu

Abstract

There has been significant recent progress designing deep generative models that generate realistic sequence data such as text or music. Nevertheless, it remains difficult to incorporate high-level structure to guide the generative process. We propose a novel approach for incorporating structure in the form of relational constraints between different subcomponents of an example (e.g., lines of a poem or measures of music). Our generative model has two parts: (i) one model to generate a realistic set of relational constraints, and (ii) a second model to generate realistic data satisfying these constraints. To train model (i), we propose a novel program synthesis algorithm that infers the relational constraints present in the training data, and then train the models based on the resulting relational constraints. In our experiments, we show that our approach significantly improves over state-of-the-art in terms of capturing high-level structure in the data, while performing comparably or better in terms of low-level structure.

1 Introduction

Over the past few years, there has been tremendous progress in designing deep generative models for generating sequence data such as natural language [1] or music [2]. These approaches leverage the vast quantities of data available in conjunction with unsupervised and self-supervised learning to learn probabilistic models of the data; then, new examples can be generated by sampling from these models, with the possibility of conditioning on initial elements of the sequence.

Despite this progress, a key challenge facing deep generative models is the difficulty incorporating high-level structure into the generated examples—e.g., rhyming and meter across lines of a poem, or repetition across measures of a piece of music. The ability to capture high-level structure is important for improving the quality of the generated data, especially in low-data regimes where only small numbers of examples are available—intuitively, knowledge of the structure compresses the amount of information that the generative model has to learn. Furthermore, explicit representations of structure—i.e., in a symbolic way rather than implicitly in a vector embedding—can have the added benefit that users can modify the structure to guide generation.

Recently, [3] proposed neurosymbolic generative models for incorporating high-level structure into image generation, focusing on simple 2D repeating patterns in images of building facades (e.g., repeating windows). The basic idea is to leverage program synthesis to extract structure from data—in particular, given an example image $x$, they devise an algorithm $A$ that extracts a program $c = A(x)$.

Preprint. Under review.
that represents the set of 2D repeating patterns present in training examples \( x \). Then, using the pairs \((x, c)\), they train two generative models: (i) a model \( p_\theta(c) \) that generates a program, and (ii) a model \( p_\theta(x \mid c) \) that generates an image that contains the structure represented by \( c \).

However, their approach is heavily tailored to images in two ways. First, their representation of structure is geared towards simple patterns occurring in images of building facades. Second, their algorithm \( A \) is specifically designed to extract this kind of program from an input image, as are their models \( p_\theta(c) \) for generating programs \( c \) and \( p_\theta(x \mid c) \) for generating images \( x \) conditioned on \( c \).

We represent the relational constraints \( c_x \) present in an example \( x \) by relating each subcomponent \( w \) of a given example \( x \) with a prototype \( \tilde{w} \), which can be thought of as the “original” subcomponent from which \( w \) is constructed. In particular, the relationship between \( w \) and \( \tilde{w} \) is labeled with a set of relations \( R \), which encodes the constraint that \( w \) and \( \tilde{w} \) should satisfy relation \( r \) for each \( r \in R \). Importantly, while each subcomponent is associated with a single prototype, each prototype may be associated with multiple subcomponents. As a consequence, different subcomponents associated with the same prototype are related in some way. This representation is compact, only requiring linearly many constraints in the number of subcomponents in \( x \) (assuming the number of prototypes is constant). Compactness ensures the representation both generalizes well and is easy to generate.

Then, we design a synthesis algorithm that can extract an optimal representation of the structure present in a training example \( x \). We show how to express the synthesis problem as a constrained combinatorial optimization problem, which we solve using an SMT solver Z3 \[4\]. Next, we represent \( c \) as a sequence, and design \( p_\theta(c) \) to be inferred through a specialized sequence VAE. Finally, we propose three possible designs of \( p_\theta(x \mid c) \) based on trying to identify an example \( x \) that is realistic (e.g., according to a pretrained model \( p_\theta(x) \)) while simultaneously satisfies the constraints \( c \).

We evaluate our approach on two tasks: poetry generation, where the relational constraints include rhyming lines or lines with shared meter, and music generation, where the relational constraints include equality in terms of pitch or rhythm, that one measure is a transposition of another (i.e., pitches shifted up or down by a constant amount), etc. We show that our approaches outperform or perform similarly to state-of-the-art models in terms of low-level structure, while significantly outperforming them in terms of high-level structure. We also perform a user study in the poetry domain to determine user-perceived quality of the generated poetry along three dimensions (structure, lyricism, and coherence), and found that on average, our approach outperformed state-of-the-art baselines including GPT-2. Finally, we demonstrate how our approach allows users to guide the generation process without sacrificing overall realism by specifying values of constraints.

**Example.** Figure 1 illustrates how our approach is applied to generate poetry. During training, our approach uses program synthesis to infer relational constraints \( c_x \) present in the examples \( x \), and uses both \( x \) and \( c_x \) to train the generative models. Here, \( c_x \) is a bipartite graph, where the LHS vertices are prototypes, and the RHS vertices correspond to lines of \( x \). Each vertex on the right is connected to exactly one prototype, and is labeled with constraints on how it should relate to its prototype. To generate new examples, it first samples relational constraints \( c \), and then samples an example \( x \) that satisfies \( c \) — i.e., we need to choose a line to fill each RHS node in a way that the line satisfies the relations with its prototype. Furthermore, a user can modify the sampled constraint \( c \) to guide the generative process. Thus, our approach enables users to flexibly incorporate domain knowledge on the high-level structure of the data into the generative process, both in terms of the relational constraints included and by allowing them to modify the generated relational constraints.

**Related work.** There has been recent work using program synthesis to improve machine learning. For instance, it has been applied to unsupervised learning of latent structure in drawings \[5\] and to reinforcement learning \[6\]. These techniques have benefits such as improving interpretability \[6, 7\], enabling learning from fewer examples \[5\], generalizing more robustly \[8\], and being easier to formally verify \[9\]. More recently, there has been work leveraging program synthesis in conjunction with deep learning, where the DNN handles perception and program synthesis handles high-level structure \[10\], including work in the lifelong learning setting \[11\]. In contrast to these approaches, our focus is on generative models. In particular, we extend recent work leveraging these ideas for image generation to incorporating high-level relational structure into sequence generation tasks \[3\].

Early music generation approaches were rule-based \[12\] or used simple statistical models such as Markov models \[13, 14\] or probabilistic CFGs \[15\]. Recent work has used deep learning to generate music \[2, 16\] and poetry \[17\]; our experiments show that these approaches have difficulty...
"Father," I said. "Father, I cannot play
The harp that thou didst give me, and all day
I sit in idleness, while to and fro
About me thy serene, grave servants go;
And I am weary of my lonely ease.
Better a perilous journey overseas
Away from thee, than this, the life I lead,
To sit all day in the sunshine like a weed
That grows to naught—I love thee more than they
Who serve thee most; yet serve thee in no way.

We are interested in domains where likely examples satisfy latent relational constraints.

Consider the problem of learning a generative model given training data from the underlying distribution. Given training examples \( x_1, \ldots, x_k \sim p^* \), our goal is to learn a generative model \( p_\theta \approx p^* \) from which we can draw additional samples \( x \sim p_\theta \). We consider sequence data—i.e., an example \( x \in \mathcal{X} \) is a sequence \( x = (w_1, \ldots, w_m) \in \mathcal{W}^m \). For example, each subcomponent \( w \) may be a line of a poem or a measure of music, and \( x \) may be a poem or song.

We are interested in domains where likely examples satisfy latent relational constraints \( c \in \mathcal{C} \) over the subcomponents. For instance, \( c \) may say that two measures \( w_i \) and \( w_j \) of \( x \) start with the same series of pitches, or that two lines \( w_i \) and \( w_j \) of \( x \) rhyme. We assume given a set of relations \( \mathcal{R} \) (e.g., \( r \in \mathcal{R} \) might be "rhyme" or "equal"), and a function \( f : \mathcal{W} \times \mathcal{W} \times \mathcal{R} \rightarrow \mathbb{B} \) (where \( \mathbb{B} = \{0, 1\} \)) such that \( f(w, w', r) \) indicates whether \( w \) and \( w' \) satisfy relation \( r \). Then, \( c \) is a compact representation of the relations present in an input \( x \). We describe the structure of \( c \) in detail in Section 3.1; for now, the approach we describe works for any choice of \( c \). In particular, we build on neurosymbolic generative models \([4]\), where \( c \) is itself generated based on a latent value \( z \in \mathcal{Z} \)—i.e.,

\[
p_{\theta, \phi}(x) = \int \sum_{c \in \mathcal{C}} p_\theta(x \mid c) \cdot p_\phi(c \mid z) \cdot p(z) dz.
\]

Figure 1: Top: Process for training. For each training example \( x \), our algorithm uses program synthesis to infer the relational constraints \( c_x = \mathcal{A}(x) \) present in \( x \). Then, (i) uses \( c_x \) to train \( p_\theta(c) = \mathbb{E}_{z \sim p(z)}[p_\phi(c \mid z) \cdot p(z)] \), and (ii) uses \( (c_x, x) \) to train \( p_\theta(x \mid c) \). Bottom: Process for generating a sample \( x \) from the learned models \( p_\phi(c \mid z) \) and \( p_\theta(x \mid c) \). Lines with the same prototype are shown in the same color; metrical constraints are represented as purple and rhyme constraints as green edges.

generating realistic high-level structure. Approaches have incorporated structure into deep learning to generate music \([18]\) or poetry \([19]\), but they are domain specific; we find they do not perform at a human level on capturing global (and sometimes local) structure. Some approaches incorporate expert-provided constraints such as rhyme and meter to generate poetry \([20]\); unlike our approach, they cannot automatically learn and generate these constraints from data.

2 Background on Neurosymbolic Generative Models

We use a fixed \( m \) to simplify our exposition; our approach trivially extends to variable \( m \).
We describe how we represent relational constraints. We design a synthesis algorithm that expresses the synthesis problem as a constrained combinatorial equation is the log-likelihood of a generative model predicting the probability of example x given relational structure x, and the second and third terms form the loss of a variational autoencoder (VAE) pφ(c | z) and qφ(z | c) [22]. In summary, this approach separately learns (i) a VAE to generate c given z, and (ii) a generative model to generate x given c; the latter can take multiple forms such as a second VAE or a generative adversarial network (GAN) [23]. This approach is called synthesis-guided generative models (SGM) since it uses program synthesis to guide training.

To leverage this framework, we have to instantiate (i) the space of relational constraints C, (ii) the synthesis algorithm A : X → C used to extract a program encoding the structure of x, and (iii) the architectures of pφ(c | z), qφ(z | c), and pφ(x | c). In previous work, [3] used heuristics specific to the image domain to achieve these goals—in particular, they used (i) simple equality constraints on sub-regions of the image designed to capture 2D repeating patterns, (ii) a custom synthesis algorithm that greedily adds constraints in the data to the program, and (iii) a representation of c as an image, in which case pφ is a generative model over images, and pφ, qφ based on an encoding of c as a fixed-length vector.

We design a synthesis algorithm that expresses the synthesis problem as a constrained combinatorial optimization problem, which it solves using Z3 [4]. In terms of (iii), our programs encode declarative constraints rather than imperative renderings, so the previous architectures of pφ, and qφ cannot be used. Instead, we use expert domain-specific heuristics, transformers [1], or graph neural networks (GNNs) [24] for pφ and qφ. For pφ, we propose several methods for imposing the constraints encoded by c when generating an example x.

3 Relational Constraints for Sequence Data

We describe how we represent relational constraints r, as well as our algorithm A for synthesizing the relational constraints c_x = A(x) present in an example sequence x.

3.1 Graph Representation of Relational Constraints

Recall that our generative models operate by first generating a relational program c, and then generating an example x that satisfies c. Thus, for each datapoint x in our dataset, we need to design a relational program c that encode constraints on the structure of x. A program c encodes a set of relational constraints, each of which imposes a constraint that subcomponents of x should have certain kinds of relations. We begin by describing the structure of a single relational constraint, and then describe how c encodes a set of relational constraints.

A relational constraint φ ∈ Φ = W × I × R, where I = {1, ..., m}, is a tuple φ = (w_i, r); we call w_i a prototype subcomponent. An example x satisfies φ (denoted x |= φ) if f(w_i, w_i, r) = 1, where w_i is the ith subcomponent of x. That is, φ says the ith subcomponent w_i of x should have relation r with prototype subcomponent w. Thus, we can interpret φ as a function φ : X → B, where φ(x) = 1 if x satisfies φ and φ(x) = 0 otherwise.

Next, a relational program c encodes a set of relational constraints on examples x. We represent c as an undirected labeled bipartite graph c = (V, E) with vertices V and E edges E ⊆ V × V × 2^R, where R is the set of relations and 2^R is the power set of R. The vertices w ∈ V are prototype subcomponents w ∈ W; equivalently, they may be vector embeddings of prototype subcomponents. The vertices i ∈ V = {1, ..., m} are the indices of subcomponents in x. The edges e ∈ E are tuples e = (w, i, R), where R ⊆ 2^R. For tractability of synthesis, we impose the constraint that each v ∈ V...
is part of a single edge \((\tilde{w}, v, R)\) (though \(\tilde{w} \in \tilde{V}\) may be part of multiple edges). Finally, \(c\) encodes the set of relational constraints

\[
\Phi_c = \{(\tilde{w}, i, r) \mid (\tilde{w}, i, R) \in E \land r \in R\}.
\]

In other words, \(c\) includes the relational constraint that each subcomponent \(w_i\) of \(x\) should have all relations \(r \in R\) with prototype \(\tilde{w}\), where \(r\) is connected to \(\tilde{w}\).

As an example, in Figure 1, the graph shown on the top right encodes a relational constraint \(c_x\), and the top right shows an example \(x\) that satisfies all the constraints \(\phi \in \Phi_c\). The nodes on the left-hand side of \(c_x\) are prototype subcomponents \(\tilde{w} \in \mathcal{W}\), each of which is a line of poetry. The nodes on the right-hand side correspond to indices \(i\) (from \(i = 1\) on top to \(i = m = 10\) on the bottom); each one is labeled with a set of relations \(R_{ij}\). Then, \(\Phi_c\) contains constraints \(\phi = (\tilde{w}, i, R_{ij})\) for each edge \(\tilde{w} \rightarrow i\) in the graph, which says that line \(i\) of \(x\) should have relations \(r \in R_{ij}\) with \(\tilde{w}\). For instance, the last (10th) node in \(c_x\) has constraints \(R_{10} = \{\text{rhyme, meter}\}\), and is connected to prototype line \(\tilde{w} = \text{“The harp that thou...”}\). Thus, this edge encodes a constraint \(\phi = (\tilde{w}, 10, R_{10})\) saying that the last line of \(x\) should rhyme and have the same meter as \(\tilde{w}\). Indeed, the last line of \(x\) is \(w_{10} = \text{“Who serve thee most...”}\), which rhymes and has the same meter as “The harp that thou...”.

**Remark 3.1.** We use prototypes rather than direct relationships between components to ensure the size of the graph is tractable—with this choice, the graph is linear in the size of the input (assuming the number of prototypes is constant) rather than quadratic. A compact graph is both easy to synthesize (for training) and train a model to generate (for generation). In our experiments, we show that our approach significantly outperforms attempting to generate full graphs (i.e., adjacency tensors).

**Remark 3.2.** We refer to \(c\) as a program since it can be interpreted as a Datalog program \([25]\) (i.e., a relational logic program). At a high level, \(\Phi_c\) is a set of Datalog relations over examples \(x \in \mathcal{X}\). Thus, \(c\) can be interpreted as a program \(c : \mathcal{X} \rightarrow \mathbb{B}\) such that \(c(x) = 1\) if \(\phi(x) = 1\) for all \(\phi \in \Phi_c\) and \(c(x) = 0\) otherwise.

### 3.2 Synthesizing Relational Constraints

Recall that when training our generative model, we need to design a program synthesis algorithm \(\mathcal{A}\) that synthesizes a relational program \(c_x = \mathcal{A}(x)\) that best encodes the latent relational constraints present in each training example \(x\). A key question is where the prototypes come from. We simply choose the prototypes \(\tilde{w}\) to be actual subcomponents in \(x\). Thus, \(c_x\) encodes that subcomponents of \(x\) are each related to one of a small number of distinguished subcomponents of \(x\). As described below, we formulate the problem of synthesizing \(c_x\) as a constrained optimization problem.

**Optimization variables.** The variables are a binary vector \(H \in \mathbb{B}^m\) and a binary matrix \(K \in \mathbb{B}^{m \times m}\). Intuitively, \(H_i\) indicates whether subcomponent \(w_i\) of \(x\) is a prototype subcomponent in \(c\), and \(K_{ij}\) indicates whether \(w_i\) is the prototype for subcomponent \(w_j\).

**Constraints.** Our optimization problem has the following three constraints:

\[
\psi_1 \equiv k_{\min} \leq \sum_{i=1}^{m} H_i \leq k_{\max}, \quad \psi_2 \equiv \bigwedge_{j=1}^{m} \bigwedge_{i=1}^{m} K_{ij} = 1, \quad \psi_3 \equiv \bigwedge_{i=1}^{m} \bigwedge_{j=1}^{m} K_{ij} \leq m \cdot H_i.
\]

First, \(\psi_1\) says that the number of prototype subcomponents is between \(k_{\min}\) and \(k_{\max}\). Next, \(\psi_2\) says that every subcomponent \(w_j\) corresponds to exactly one prototype subcomponent \(w_i\). Finally, \(\psi_3\) says that for every \(i\), if \(w_i\) is the prototype subcomponent of \(w_j\) according to \(K\), then it must be a prototype subcomponent according to \(H\) as well.

**Objective.** The objective of our optimization problem is expressed in terms of a precomputed distance matrix \(D \in \mathbb{R}^{m \times m}\), where \(D_{ij}\) measures the similarity between components \(w_i\) and \(w_j\); smaller values indicate a greater degree of similarity. In particular, we define

\[
D_{ij} = \frac{1}{|\mathcal{R}|} \sum_{r \in \mathcal{R}} \mathbb{I}(f(w_i, w_j, r) = 0),
\]

i.e., \(D_{ij}\) is the fraction of relations that are not satisfied by \(w_i\) and \(w_j\). Then, our objective (which is to be minimized) has the following three terms:

\[
J_1 = \sum_{i,j=1}^{m} K_{ij} \cdot D_{ij}, \quad J_2 = \sum_{i,j=1}^{m} \left( \prod_{k} K_{ki} \cdot K_{kj} \right) \cdot D_{ij}, \quad J_3 = - \sum_{i,j=1}^{m} M_i \cdot M_j \cdot D_{ij}.
\]
We describe our model for generating examples \( x \). We sample values \( c \) using a domain-specific generative model \( \phi \) we let \( \theta \) correspond to a new prototype subcomponent. More precisely, we initialize \( w_i \) that encodes that subcomponent \( w_i \) should be similar to its prototype. Second, \( J_2 \) says that subcomponents should be similar to other subcomponents that share the same prototype. Third, \( J_3 \) says that different prototype subcomponents should be dissimilar.

**Optimization problem.** Our algorithm \( A \) uses Z3 to solve the optimization problem

\[
(H^*, K^*) = \arg \min_{H,K} \{ \lambda_1 \cdot J_1 + \lambda_2 \cdot J_2 + \lambda_3 \cdot J_3 \} \quad \text{subj. to} \quad \psi_1 \land \psi_2 \land \psi_3,
\]

where \( \lambda_1, \lambda_2, \lambda_3 \in \mathbb{R}_{\geq 0} \) are hyperparameters. Finally, to construct \( c_x \), \( A \) chooses

\[
\tilde{V} = \{ w_i \mid H_i^* = 1 \}, \quad V = \{ 1, ..., m \}, \quad E = \{ (w_i,j,R_{ij}) \mid K_{ij}^* = 1 \},
\]

where \( R_{ij} = \{ r \in \mathcal{R} \mid f(w_i,w_j,r) = 1 \} \)—i.e., \( \tilde{V} \) are the prototype subcomponents according to \( H^* \), \( E \) are the edges according to \( K^* \), and \( R_{ij} \) are the relations satisfied by \( w_i \) and \( w_j \). Z3 is guaranteed to find the optimal solution; in the unlikely event that multiple such solutions exist, it chooses one nondeterministically. Intuitively, our approach should perform well when a handful of prototypes are sufficient to approximately capture the relational structure in the data. Also, the user has the ability to define their relations in a way that captures desired structure.

4 Neurosymbolic Generative Models with Relational Constraints

We describe our model for generating examples \( x \). Recall that our approach proceeds in two steps: (i) generate \( c \), and (ii) generate \( x \) given \( \Phi_c \). We describe each of these steps in detail below.

4.1 Step 1: Generating Relational Constraints

The first step of our generative model is to generate relational constraints \( \Phi_c \) using a VAE—i.e., \( z \sim p(z) \) and \( c \sim p_{\theta}(\cdot \mid z) \), where \( p_{\theta}(c \mid z) \) is a VAE and \( p(z) = N(z; 0, I) \) is a Gaussian distribution. The main choice is the architecture to use for the VAE. In particular, we consider a representation of \( c \) as a sequence \((s_1, ..., s_m)\), where \( s_i \in \{0, 1, ..., m\} \) for each \( i \); intuitively, \( s_i \) encodes that subcomponent \( w_i \) should have the same prototype subcomponent as \( w_{i-s_i} \), or if \( s_i = 0 \), that \( w_i \) corresponds to a new prototype subcomponent.

More precisely, we initialize \( \Phi_c = \emptyset \). Then, we generate the sequence \( s_i \in \{0, 1, ..., m\} \) and \( r_i \in \{0, 1, ..., m\} \) (where \( r_i \) is represented as a binary vector of length \( n = |\mathcal{R}| \) using one of two approaches: (i) an LSTM-VAE, or (ii) a feedforward network whose output is iteratively sampled from as a categorical distribution and then used as input in the next step (see Appendix B.1 for details). For each \( i \), we generate \((\tilde{w}_i, R_i)\) based on \( s_i \) and \( r_i \). If \( s_i = 0 \), we generate a new prototype subcomponent \( \tilde{w} \) using a domain-specific generative model, and add \( \phi_i = (\tilde{w}_i, i, R_i) \) to \( \Phi_c \). If \( s_i > 0 \), we let \( \phi_i = (w_{i-s_i}, i, R_i) \).

4.2 Step 2: Generating Examples Given Relational Constraints

Next, we describe how we implement the second step \( p_{\theta}(x \mid c) \) of our generative model. We propose three approaches for generating \( x \) given \( \Phi_c \); we give details in Appendix A.

**Approach 1:** Constrained sampling. We sample values \( x \sim p_{\theta}(\cdot) \) by sequentially sampling \( w_i \sim p_{\theta}(\cdot) \) from a pretrained generative model \( p_{\theta}(w) \). We do so using rejection sampling at each step—i.e., we sample \( w_i \sim p_{\theta}(\cdot) \) until we find \( w_i \) satisfying \( f(\tilde{w}, w_i, r) = 1 \) for each \((\tilde{w}, i, r) \in \Phi_c \). In addition, to speed up sampling, at each step of sampling \( w_i \) (e.g., a word in a line or a pitch in a measure), we eliminate choices that violate \( \Phi_c \).

**Approach 2:** Constraint-aware embeddings. We train a conditional generative model \( p_{\theta}(w_1, ..., w_m \mid c) \) (in the form of a graph convolutional network) that simultaneously generates all \( m \) subcomponents in a way that satisfies \( c \), and sample \( x = (w_1, ..., w_m) \sim p_{\theta}(\cdot \mid c) \).

**Approach 3:** Combinatorial optimization. We sample \( x \sim p_{\theta}(\cdot) \) by sequentially generating \( w_i \) by solving an optimization problem whose objective is to maximize adherence to \( \Phi_c \) plus additional terms encoding domain-specific heuristics encouraging \( w_i \) to be realistic.
Table 1: Results for the music domain. Left: We show negative log-likelihood (“NLL”, lower is better) on the held-out human test set (i.e., by estimating the ELBo using sampling). Right: We show Fréchet distance on MusicVAE embeddings (“FD”, lower is better). Both: We show the cross-entropy loss of the graph discriminator trained to distinguish synthesized programs of generated examples vs. held-out test set examples (“GCN Disc.”, higher is better), and the accuracy of a random forest trained to do the same thing on a handcrafted featurization of the programs (“RF”, lower is better). The highest score in each column is bolded. As can be seen, our approach with sampling strategy A2 outperforms the baselines on all metrics, also outperforming the ablation using the same strategy but without program synthesis (i.e., using the full adjacency tensor).

5 Experiments

We evaluate our approach on two domains: music and poetry generation. We provide details on experimental design and additional results in Appendix B.

5.1 Music Generation

We evaluated our approach on a music generation task. In this setting, \( x \) is a song, and \( w \) are measures of music. We consider 20 relations including equality, same rhythm, same pitch progression, etc.; a full list is given in Appendix B.2.

**Dataset.** We used songs from the Essen folk song corpus [26], using 2223 for training and 555 for testing (after removing examples with less than 16 measures or that were not in the standard 4/4 meter). For this dataset, we used each of the three approaches A1, A2, and A3 described in Section 4 to sample \( x \sim p_\theta(\cdot | c) \). For A1, we use a pretrained transformer called MusicAutoBot [27]. For A2, we require a generative model that constructs vector embeddings of measures; we use the pretrained version of Magenta’s MusicVAE which embeds pairs of measures [28] and adapted it to produce single-measure embeddings. We finetune all models on our training examples.

**Baselines.** We compare to MusicAutoBot, a pretrained and finetuned LSTM with attention (AttentionRNN) [29], Magenta’s 16-bar MusicVAE (pretrained and finetuned), and StructureNet, an approach that integrates structure into an LSTM [18]. In addition, we compare to an ablation where we use sampling approach A2, but with full adjacency tensors instead of synthesizing compact program representations; this ablation demonstrates the importance of our compact representation.

**Metrics.** We compare performance in terms of both high-level and low-level structure. For low-level structure, we use the negative log likelihood (NLL) on a held-out test set for MusicVAE, MusicAutoBot, and our approach with strategy A2. The remaining approaches are not probabilistic (or estimating probabilities is intractable). For these approaches, we use a variant of the standard Fréchet distance (FD) score used to evaluate GANs [30]—i.e., we compute the Fréchet distance between the MusicVAE (16-bar) embeddings of the generated music and the held-out test set.

For high-level structure, given a generated (or human held-out) example \( x \), we use our synthesis algorithm to synthesize its relational structure \( c_x = A(x) \). Then, given a collection \( C_{\text{gen}} = \{c_x \mid x \in X_{\text{gen}}\} \) of synthesized structure for generated examples, along with a collection \( C_{\text{human}} = \{c_x \mid x \in X_{\text{human}}\} \) of synthesized structure for the held-out human examples, we train a graph convolutional neural network (GCN) to try and discriminate \( C_{\text{gen}} \) from \( C_{\text{human}} \), as well as a random forest (RF) over handcrafted features (see Appendix B.4). Intuitively, higher discriminative power should indicate less realistic structure. In both cases, we use a balanced dataset (i.e., 50% human held-out and 50% generated) so random predictions have accuracy 0.5. Recent work has shown that such discriminator-based metrics are valid for evaluating quality of generated examples [31].
Table 2: Results for the poetry domain. We show Fréchet distance on SentenceBERT embeddings ("FD", lower is better), along with the cross-entropy loss of the graph discriminator trained to distinguish synthesized programs of generated examples vs. held-out test set examples ("GCN Disc.", higher is better). The best score in each column is bolded. As can be seen, our approach (SGM) with sampling strategy A1 outperforms all baselines in terms of high-level structure, though GPT2-based models and our ablation outperform it in terms of low-level structure.

Results. In Table 1 we show results for models for which we can compute the test set NLL (left) and results for the remaining models (right). As can be seen, our approach (SGM) with sampling strategy A2 outperforms all other models in both tables, in terms of both high-level structure and low-level structure. In Table 1 (left), the closest alternative is MusicVAE, for which the NLL is not too much larger; however, it performs significantly worse than our approach in terms of high-level structure. In Table 1 (right), we find that our other approaches also perform well (though not as well as A2). In particular, A1 performs well in terms of low-level structure, but is more mixed in terms of high-level structure. In contrast, A3 performs well in terms of high-level structure, but is mixed in terms of low-level structure, most likely since it does not use a learning-based model to generate low-level structure. Finally, note that our ablation performs poorly, especially in terms of structure, demonstrating the importance of synthesizing compact representations of structure.

5.2 Poetry Generation

Next, we apply our approach (SGM) to poetry generation; in this case, $x$ is a poem, and $w$ is a line. We consider two relations, rhyming and equal meter; see Appendix B.3 for details.

Dataset. We use Project Gutenberg’s poetry collection [32], filtered to focus on examples that contain rhymes and meter. We use 2700 10-line poems for training and 300 for testing.

Our approach. We were unable to apply A3 due to the large size of the vocabulary, making constrained optimization infeasible. We were also unable to apply A2 because state-of-the-art generative models such as BERT and GPT2 were unable to capture rhyming and meter, since they operate at the word level where this information is unavailable. In A1, rather than sample words going forward, we sample them backwards, making it easier to sample lines that satisfy rhyming constraints; see Appendix A. Thus, we use BERT to sample [33], since it is bidirectional.

Baselines. We compare to Huggingface’s implementation of generation using beam search for BERT and GPT2 [34] [1], both finetuned on our dataset. We also consider a variant GPT2-Opt of GPT2 where we use beam search to choose line breaks in a way that maximizes occurrences of rhyme and meter. We also tried a variant of GPT2 that used constrained sampling to try and find poems that fit a given rhyme and meter scheme, but the search space was too large and it was unable to generate a single poem even after several hours. We also compare to an implementation of RichLyrics [19], where the consecutive parts of speech for each line given the previous line and the ability to fill in the correct word for the given part of speech were both learned separately from the corpus.

Finally, we consider an ablation where we perform constrained sampling according to our approach (A1), but with a uniformly random $\Phi_c$ rather than sampling it from a learned distribution. This ablation demonstrates the importance of learning the distribution over constraints.

Metrics. As before, we compare both high-level structure and low-level structure. For low-level structure, we measure the FD score using SentenceBert embeddings, which are unaware of rhyme, meter, or sentence ordering [35]. In this case, because our approach uses constrained sampling from a pretrained generative model, we could not evaluate negative log-likelihood according to our approach. For high-level structure, we train a GCN discriminator to discriminate between synthesized programs for the generated examples vs. the test set examples.
<table>
<thead>
<tr>
<th>Method</th>
<th>Average Score</th>
<th>Lyricism</th>
<th>Coherence</th>
<th>Rhyme/Meter</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGM (Ours)</td>
<td>3.66</td>
<td>3.81</td>
<td>3.59</td>
<td>3.59</td>
</tr>
<tr>
<td>GPT2-Finetune</td>
<td>3.30</td>
<td>2.90</td>
<td>3.91</td>
<td>3.12</td>
</tr>
<tr>
<td>BERT-Finetune</td>
<td>2.28</td>
<td>2.11</td>
<td>2.00</td>
<td>2.77</td>
</tr>
<tr>
<td>RichLyrics</td>
<td>3.09</td>
<td>3.24</td>
<td>3.09</td>
<td>2.93</td>
</tr>
</tbody>
</table>

Table 3: A user study evaluation in the poetry domain. While GPT2-Finetune outperforms our model in terms of coherence (presumably due to the well-known superiority of GPT-2 over BERT for generation), our method outperforms in terms of overall lyricism (i.e., whether the poem reads like poetry or prose), prominence of rhythmic/metrical structure, and average score.

Figure 2: Left: Poetry generated using relational constraints $c \sim p_\phi(\cdot)$. Right: user modified variant of $c$ where the last two lines share a prototype with the two lines before them.

**Results.** We show results in Table 2. Our approach (SGM) with sampling strategy A1 significantly outperforms all baselines in terms of high-level programmatic structure. Furthermore, it outperforms all baselines except GPT2-based models and our ablation on non-structure metrics. In particular, our ablation performs slightly better than our model in terms of low-level structure, most likely since the FD scores are based on embeddings unaware of rhyme and meter, or even sentence order. As expected, our approach significantly outperforms the ablation in terms of high-level structure.

GPT2 and GPT2-Opt produce more human-like output according to the FD score, likely since GPT2 is significantly better at natural language generation than BERT. If we could instead build on GPT2, then our approach would likely achieve better performance; however, we cannot do so since we rely on backwards sampling, which GPT2 does not support. We leave this direction to future work.

**User study.** We also performed a user study in this domain, with 50 participants. We asked each user to score 12 (randomly shuffled) poems (from 1 to 5, with 5 being the highest score) generated by the four techniques along 3 dimensions (for a total of 36 questions): lyricism (i.e., whether the text reads more like prose or poetry), coherence (i.e., whether the text appears cohesive in nature), and rhyme/meter (i.e., whether the poem exhibits rhyming and meter). The results are shown in Table 3. They support our empirical findings that, while GPT-2 outperforms our BERT-based approach in coherence, our approach is more poetic and has a stronger rhythmic and metrical structure than the baselines. We provide additional details in Appendix E.1.

**User modifications.** A key benefit of our approach is that the user can modify the relational constraints $c$ (or construct their own from scratch) for use in the second step $p_\theta(x \mid c)$, giving the user a way to guide the generative process. An example in the poetry domain is shown in Figure 5.2 and musical examples are shown in Appendix E.2.

6 Conclusion

We have presented a novel approach for representing and synthesizing relational constraints on sequence data, and for generating examples whose relational structure resembles that of the training data. Our experiments demonstrate that we outperform existing approaches in terms of achieving human-like structure, while performing comparably or better on both a user study and widely-used quantitative metrics which do not explicitly account for structure. Equally importantly, our approach gives the user a way to guide the generative process by modifying the relational constraints.
Limitations. Our work assumes that the relational primitives are known; a key direction for future work is automatically discovering these primitives. In addition, we anticipate that better strategies for integrating our synthesized structure with deep language models would improve performance. We do not foresee any significant societal impacts or ethical issues with our work beyond generic issues with designing more powerful text generation models (e.g., ability to generate harmful content).

References


**Checklist**

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper’s contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [Yes]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]

2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [No]

3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [No]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [No]
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]
   (d) Did you discuss whether and how consent was obtained from people whose data you’re using/curating? [No]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [Yes]

If you used crowdsourcing or conducted research with human subjects...

1. Did you include the full text of instructions given to participants and screenshots, if applicable? [Yes]
2. Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A] Our user study does not require IRB review according to the guidelines provided by our institution.

12
3. Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [Yes]
A Generating Examples Given Relational Constraints

A.1 Approach 1: Constrained Sampling

In the music domain, we choose the pretrained generative model \( p_\theta(w) \) to be a pretrained version of MusicAutoBot. To generate \( x \), we sequentially sample each measure \( w_i \) conditioned on all prior measures \( w_1, ..., w_{i-1} \). Each measure is sampled by sequentially sampling a sequence of pitch-duration pairs until the total duration is 16 beats (i.e., the length of a measure). During sampling, we mask pitch-duration pairs that cannot satisfy \( \Phi_c \) (i.e., we set their sampling probability to zero and rescale the remaining probabilities). For instance, if the “has similar interval” relation is supposed to hold between the \( i \)th prototype measure and measure \( i \), and we are sampling the second note of measure \( i \), then we mask any pitch \( j \) in measure \( i \) such that

\[
|\text{pitch}_j - \text{pitch}_{j-1} - \text{pitch}_{j-1} - \text{pitch}_{j-2}| \geq 3,
\]

where pitch \( j \) is pitch \( j \) in the prototype corresponding to \( w_i \). In other words, we eliminate pitches that would cause sampling to violate this constraint.

In the poetry domain, we finetune a pretrained BERT model on our dataset, by taking the pretrained models weights and then training the model on our dataset with a strong gradient weight decay. BERT has the ability to complete masked words in a sentence. We leverage this ability to sample lines that rhyme and have the same meter, which is a challenging task since such lines are a tiny fraction of the search space. We describe how we simultaneously handle rhyming and equal meter; the cases where only one of these two constraints has to hold are similar. Given a prototype \( \tilde{w} \), we work backwards—on each step \( j \), we sample from BERT a word \( \tilde{\text{word}}_j \) that has the same number of syllables as the corresponding word \( \text{word}_j \) in the prototype. More precisely, we feed BERT the sequence

\[
\tilde{\text{word}}_1, ..., \tilde{\text{word}}_{j-1}, \text{MASK}, \text{word}_{j+1}, ...
\]

and ask it to fill in the masked word, setting the probability of any word with different number of syllables as \( \text{word}_j \) to zero. In addition, in order to avoid producing a line which is similar to the original line, we also set the probability of any word too similar to the original word in terms of GloVe cosine similarity [36] to zero, except for in the case of the last word. For the last word (which we sample first), we additionally restrict to words that rhyme with \( \text{word}_j \). To increase diversity, we sample the remaining words twice—(i) backwards-to-forwards from word \( k - 1 \) to word 1, where \( k \) is the number of words, and (ii) we resample each of the \( k - 1 \) words (i.e., except the last word) in a random order. We discard any lines which, according to BERT, after being sampled are determined to be too unlikely when preceded by the previously generated lines.

A.2 Approach 2: Constraint-Aware Embeddings

In this approach, we start with a pretrained generative model \( p_\theta(w) \) that ignores \( c \); in particular, we assume that

\[
p_\theta(w) = \int p_\theta(w | u) \cdot p(u)du,
\]

where \( p_\theta(w | u) \) is the decoder network of a VAE over \( w \), and \( p(u) = \mathcal{N}(0, I) \). Now, rather than sample \( w_i \sim p_\theta(\cdot) \), we train another generative model

\[
p_\psi(u_1, ..., u_m | c) = \mathcal{N}(\mu_\psi(c), \Sigma_\psi(c))
\]

to generate latent vectors \( u_i \in \mathcal{U} \) such that \( w_i \sim p_\theta(\cdot | u_i) \) are likely to satisfy \( \Phi_c \). More precisely, \( \mu_\psi \) and \( \Sigma_\psi \) are the intermediate outputs of a graph convolutional network (GCN) [24] that takes as input the graph \( c \) (where edge attributes between nodes encode \( \Phi_c \)) and ultimately outputs a sequence \( (u_1, ..., u_m) \).

Our approach can be considered to be a graph autoencoder in the sense that the objective function used during training rewards the reconstruction of the exact embeddings of the nodes and (implicitly, through the relationship consistency loss) their edge attributes. Our graph encoder/decoder produces one latent vector per node, which are rewarded for being close to i.i.d. Gaussian random variables.
with mean zero and variance one. Note that in this approach no explicit constraints are enforced; however, they are enforced implicitly through the relationship consistency and reconstruction loss. Note also that it is necessary to enforce the i.i.d. property with respect to the joint distribution of all nodes in order to obtain realistic results when sampling new datapoints assuming an i.i.d. normal distribution over the latent states.

To train \( p_\psi \), we construct a training example \((c, (u_1, \ldots, u_m))\) for each training example \(x = (w_1, \ldots, w_m)\), where \(u_1, \ldots, u_m\) are obtained by the encoder network \(q_\theta(w \mid u)\)—i.e., \(u_i \sim q_\theta(c \mid w_i)\). Then, we train \( p_\psi \) using the objective

\[
J(\psi) = \sum_{(c, u)} D_{KL}(\mathcal{N}(\mu_\psi(c), \Sigma_\psi(c)) \parallel \mathcal{N}(0, I)) + \sum_{i=1}^m \|u_i - \mu_\psi(c)_i\|_2^2 + J_{rel}(\mu_\psi(c); c).
\]

The first term enforces that the distribution of the latent vectors \(u\) should be Gaussian, and the second term enforces that each latent vector \(u\) should be close to its original value according to the VAE encoder \(q_\theta(u \mid w)\). The third term is designed to enforce the satisfaction of the constraints \(\Phi_c\). In particular, we train a kind of “semantic discriminator” \(p_\alpha(u, u' \mid r)\), that predicts whether \(w \sim \theta_0(c \mid u)\) and \(w' \sim \theta_0(c \mid u')\) satisfies relation \(r\)—i.e., \(f(w, w', r) = 1\). The network \(p_\alpha\) is trained on data generated from the given training examples \(x\). Then, given \(p_\alpha\), we want \((u_1, \ldots, u_m) = \mu_\psi(c)\) to satisfy

\[
p_\alpha(u_i, \tilde{u}, r) \approx \begin{cases} 1 & \text{if } (\tilde{w}, i, R) \in \Phi_c \land r \in R \\ 0 & \text{otherwise}, \end{cases}
\]

where \(\tilde{u} \sim q_\theta(c \mid \tilde{w})\). In other words, we want to generate an example \(x\) that satisfies the relations in \(c\) according to \(p_\alpha\). In particular, we use the loss

\[
J_{rel}(\tilde{u}; c) = \sum_{i=1}^m \sum_{r \in R} CE\left(p_\alpha(u_i, \tilde{u}, r), 1 \left( (\tilde{w}, i, R) \in \Phi_c \land r \in R \right) \right),
\]

where \(\tilde{u} \sim q_\theta(c \mid \tilde{w})\) and where CE denotes the cross-entropy loss. Once we have trained \(p_\psi\), we generate sequences by sampling \(\tilde{u} \sim p_\psi(\cdot \mid c)\) and \(u_i \sim \theta_0(\cdot \mid u_i)\), and constructing \(x = (w_1, \ldots, w_m)\).

For the music domain, we use embeddings from a pretrained Magenta MusicVAE; unlike the MusicVAE used for evaluation, we use and finetune a model that decodes only 1 measure of music from a 256-dimensional vector. Then, we use the decoder portion of this model to convert the embeddings \(w_1, \ldots, w_m \sim p_\psi(u_1, \ldots, u_m \mid c)\) sampled from the GCN-VAE \(p_\psi\) into measures. The graphs in the training set vary in size depending on the number of prototype measures.

### A.3 Approach 3: Combinatorial Optimization

Given sampled program \(c\), this approach attempts to generate values \(x = (w_0, \ldots, w_m)\) such that \(x \models \Phi_c\) by solving a system of constraints. However, when generated using a neural network, relational constraints \(\phi \in \Phi_c\) are not always consistent with one another, so we convert the constraint \(x \models \Phi_c\) into an objective—i.e.,

\[
x = \arg\max_{x \in \mathcal{X}} \sum_{i=1}^m \sum_{r \in R} \mathbb{1}(R(\tilde{w}, w_i, n) \Leftrightarrow (\tilde{w}, i, n) \in \Phi_c).
\]

The ability to encode this optimization problem as one that Z3 can solve depends on the domain and relations. For this approach to work, we may need to include additional, handcrafted terms in the objective that encourage the generated example \(x\) is realistic.

For the music domain, the optimization variables are the optimal sequence of pitches and their durations. The objective function is a linear combination of the degree to which \(x\) satisfies \(c\), along with domain-specific heuristics—e.g., minimizing large jumps in pitch values (i.e., \(|\text{pitch}_{i+1} - \text{pitch}_i| \leq \delta\)...
pitch \_j \geq 4\), not having any intervals of length 6 (i.e., |pitch\_i+1 − pitch\_i| = 6) due to the unpleasant harmonic nature of that interval, and not having two consecutive jumps in pitch (i.e., |pitch\_i+2 − pitch\_i+1| ≥ 5) ∧ |pitch\_i+1 − pitch\_i| ≥ 5). These heuristics are based on standard concepts from music theory [37].

B Evaluation Details

B.1 Experimental Setup

Synthesizing programs. The hyperparameters J1, J2, and J3 in the program synthesis task, as described in the main section of this paper, regulate the degree to which the optimization favors solutions which have high similarity between prototype and sequence measures, have high similarity between elements sharing a prototype, and have high difference between prototypes, respectively. Their values were different with respect to the two different domains. In the poetry domain, J1, J2, and J3 were 1, 10, and 1, respectively. In the music domain, J1, J2, and J3 were 1, 5, and 1, respectively. These values were arrived at through attempting to arrive at results which closely matched a set of human (author) annotated programs.

Generating c. To generate c in the poetry domain, we use an LSTM-VAE with 6 LSTM layers and a latent size of 50. This model is trained to reproduce a given sequence of \((s_i, r_i)\) pairs which are given as input, with an additional requirement that the distribution of their encodings should be roughly equivalent to a Gaussian normal distribution. In the music domain, while we experimented with using an LSTM-VAE, empirically we had more success using a feedforward 3-layer network which took the previous n (usually n = 6) \((s_i, r_i)\) pairs, and outputted a distribution over the following pair.

Each \((s_i, r_i)\) pair is represented as a \((S + |R|)\)-dimensional vector, where \(S\) is the maximum distance between objects with the same prototype and \(R\) is the set of relations.

High-level structure. We evaluate high-level structure by training a model to try to discriminate \(C_{\text{gen}}\) from \(C_{\text{human}}\); if the model achieves lower performance, then the quality of high-level structure is higher. A general approach is to train a graph neural network (e.g., a graph convolutional network) to do so; this model takes as input the graph structure of relational constraints \(c\), along with vector embeddings of the prototype subcomponents, and outputs whether \(c \in C_{\text{gen}}\) or \(c \in C_{\text{human}}\). We balance the data so it consists of 50% human data and 50% generated data. We report the cross-entropy (CE) loss; higher values correspond to better generative models. In the music domain, we additionally used a random forest (RF) trained on a manual featurization of \(c\). We report the accuracy of the RF; lower values (i.e., closer to 50%) correspond to better generative models.

B.2 Musical Relations Used

The following are the relations \(r \in R\) used in the music domain:

1. Measures \(i\) and \(j\) have the same pitch classes.
2. Measures \(i\) and \(j\) have the same pitch class prefix.
3. Measures \(i\) and \(j\) have the same pitch class suffix.
4. Measures \(i\) and \(j\)'s pitches have an edit distance of 1.
5. Measures \(i\) and \(j\) have approximately the same interval structure.
6. Measures \(i\) and \(j\) have the same interval prefix.
7. Measures \(i\) and \(j\) have the same interval suffix.
8. Measures \(i\) and \(j\) have the same note (pitch + duration) prefix.
9. Measures \(i\) and \(j\) have the same note (pitch + duration) suffix.
10. Measures \(i\) and \(j\) have the same rhythm.
11. Measures $i$ and $j$’s rhythm has an edit distance of $\leq 2$.
12. Either measure $i$’s onsets are a subset of measure $j$’s onsets, or measure $j$’s onsets are a subset of measure $i$’s onsets.
13. Measures $i$ and $j$ have the same rhythmic and melodic contour.
14. Measures $i$ and $j$ have the same rhythmic and melodic contour prefix.
15. Measures $i$ and $j$ have the same rhythmic and melodic contour suffix.
16. Either the first or second half of measures $i$ and $j$ are identical.
17. Either both or neither of measures $i$ and $j$ have leaps.
18. Measures $i$ and $j$ fit within the same diatonic scale.
19. Either both or neither of measures $i$ and $j$ have syncopation.
20. Either both or neither of measures $i$ and $j$ have consecutive notes shorter than an eighth note.

**B.3 Poetry Relations Used**

The following are the relations $r \in \mathcal{R}$ used in the poetry domain:
1. Lines $i$ and $j$ have the same end rhyme.
2. Lines $i$ and $j$ have the same meter.

**B.4 Random Forest Features**

The following are the manually constructed features used in the random forest discriminator for the music domain:
1. Mean number of relations between prototype and sequence measures.
2. Variance of number of relations between prototype and sequence measures.
4. Longest sequence $i \ldots j$ such that $w_i \ldots w_j$ all have the same prototype measure.
5. Number of pairs $(i, j)$ such that $\tilde{w}_i = \tilde{w}_j$ and $\tilde{w}_{i+1} = \tilde{w}_{j+1}$.
6. Mean distance between two measures with the same prototype.
7. Variance in distance between two measures with the same prototype.

**B.5 Comparison to Constraint Solving**

We also considered a comparison to a constraint-based implementation called Motifate, with explicit attention to development of musical material [38]. This approach was designed with heuristics for 3-beat measures, while our evaluation models anticipated 4-beat measures, so we could not obtain FD scores. Nevertheless, we found that even the structure was insufficient—its RF discriminator had accuracy 0.91, and its GCN discriminator had cross entropy loss 0.43, both of which are significantly worse than the other approaches.

**C Societal Impacts & Ethical Considerations**

While we do not foresee significant societal impacts and ethical issues, there are some possible considerations to be aware of. First, we draw in our music synthesis research on a variety of folk songs, including those collected from indigenous peoples. We must consider the ethical implications of using this material in such a fashion. Second, in the poetry domain, we found that we could not open-source the output, as a large percentage of poems contained disturbing, offensive, or inappropriate material. Future research should focus on how to bias a language model away from such language.
D Assets used

Synthesis was performed using the Z3 library [4]. We employed the torch_geometric library in our implementation of approach A2 [39], and relied on the pre-existing GPT and BERT models from huggingface in the poetry domain [40]. We used lilypond to generate visual music scores, and music21 [41] to generate audio files and perform data processing in the music domain. We built on an existing system, musicautobot [27], in our approach A1.

E Qualitative Results

E.1 Details of the user study

50 participants took place in the study on Mechanical Turk. Each was paid $5 to complete a survey with 12 questions (3 poems each from four sources, Ours, GPT2-Finetune, BERT-Finetune, and RichLyrics). All poems were chosen automatically by taking the top 3 examples from the generated datasets according to GPT2-log-likelihood. The participants were asked to rank the following 3 statements from "strongly disagree" to "strongly agree" (1-5) as follows:

1. It is obvious that this is a poem
2. This text is coherent
3. I notice that this text has rhyme and meter

E.2 Conditioning on User-Provided Structures

Here we show how user modifications can occur in the music and poetry settings. By explicitly modifying $c$, we are able to generate two pieces of poetry or two tunes with similar internal patterns but with different structural characteristics.

Figure 3: A song generated using our approach, and a nearly identical song generated where part of the sampled relational constraints $c$ were manually modified. These pieces were generated using A3, and the same reference measures $\tilde{w}$ were used, but $\Phi_c$ was slightly perturbed (the similarity relations were changed).

E.3 Qualitative Observations on the Music Domain

In addition to quantitative measurements, we evaluated the strengths and weaknesses of our approach using A2 (which was the best according to quantitative metrics). According to our observations, the strengths of A2 include clearer phrases with obvious resolutions, likely and plausibly repetitive rhythms, intervals between notes which seemed plausible but not overly repetitive, and less variance in quality. However, the results were not very rhythmically diverse, and certain idiomatic patterns of resolutions of intervals between notes and at the end of phrases were not followed. Furthermore, AttentionRNN does better in terms of creating realistic chord progressions (we did not explicitly consider chord progressions in our model; doing so is a promising direction for future work). Finally,
Figure 4: Left: Poetry generated using relational constraints $c \sim p_{\phi}(\cdot)$. Right: user modified variant of $c$ where the last two lines share a prototype with the two lines before them.

while global structure is much better than the baselines, examples still relatively infrequently had the full four-bar repetitions characteristic of much folk music.

E.4 Examples from the Music Domain

We show an example of generated songs using our approach with each A1, A2, and A3 in Figure 5, Figure 6, and Figure 7, respectively, and show an example generated using each of the baselines MusicVAE16, AttentionRNN, MusicAutoBot, and StructureNet in Figures 8, 9, 10, & 11 respectively. Qualitatively, the generated music and poetry appears plausible, exhibiting realistic high-level structure without sacrificing low-level structure.

Figure 5: An example of a song generated using our approach (A1). Measures that have the same prototype are shown in the same color. Note the existence of repeating four-bar phrases, found commonly in folk songs.

Figure 6: An example of a song generated using our approach (A2). Measures that have the same prototype are shown in the same color. Note the existence of clear phrase endings marked by long notes or rests, particularly the recurring pattern of fast notes resolving into long notes.

Figure 7: An example of a song generated using our approach (A3). Measures that have the same prototype are shown in the same color. The existence of two-bar and three-bar phrases is apparent, but the close note and rhythm similarities among different prototypes weaken the overall clarity of the song’s melody.
Figure 8: An example of a song generated using Magenta’s hierarchical MusicVAE model finetuned on our dataset. While the local structure is extremely coherent, it does not seem to possess the expected internal repetition/development.

Figure 9: An example of a song generated using AttentionRNN trained on our dataset. Note the existence of erratic rhythms and unclear structure, which are common traits of custom-trained AttentionRNN models.

Figure 10: An example of a song generated using MusicAutoBot. Note the repetitive nature and stark contrast between the first half and second half of the song, which are common problems with transformer models.

Figure 11: An example of a song generated using StructureNet. While some degree of internal structure is apparent, and the local coherence is high, the pattern of internal repetition seems fairly arbitrary.

E.5 Examples from the Poetry Domain

In Figure E.5 we show an example poem generated using our approach (top) along with one generated using GPT2-Opt (bottom). As can be seen, the GPT2-Opt poem does not capture structure in the same way human poems do—e.g., adjacent lines are unrelated, lines have very unequal length, and the only rhymes are the word “the” in the brown lines and the words “to” and “too” in the green lines. There is even less structure in poems generated using vanilla GPT2. Thus, GPT2 is completely unable to capture high-level structure in the real poetry provided as training data. In contrast, our poem captures structure very similar to the human poem shown in Figure 1 such as rhyming adjacent lines.

We also give examples of poetry generated using our baselines—in particular, GPT2 finetuned and optimized for rhyme and meter in Figure 14, BERT finetuned as a language generation model in Figure 16, RichLyrics, and our ablation (i.e., use BERT in conjunction with a uniformly randomly sampled $\Phi_c$) in Figure 17.
Figure 12: Left: Poetry generated using relational constraints \( c \sim p_\phi(\cdot) \). Right: Poetry generated by GPT2-Opt. Notice the lack of characteristic structure in GPT2-Opt, despite its coherence.

I know many things, and therefore I forgot,
Though I needed time to look ahead,
To understand something, time to let it fade away
As though it was yesterday as they
Wore common things, free, rather—free, to go like the tide;
But another is to make no one, as it does.
Perhaps you know it. A queen, her beautiful son,
And another woman who has to go without one.
The voices like their cries of war,
They let us believe in a good restore!

Figure 13: An example of poetry generated using our approach. Lines that have the same prototype are shown in the same color.

Through the air and through the sky
And through all the world
I saw the sun the moon a star shine
In the midst of the stars
The stars were shining in my eyes my heart
Was throbbing with joy I felt
My heart was beating with love
I was
A little child in a little town
Where the little boys play

Figure 14: A poem generated using GPT2-Opt. It is more plausible than BERT in terms of global structure, which may be due to the fact that GPT2 is a better text generation tool than BERT, but it is still somewhat repetitive and its structure is not very human-like.

all and and
and and
and and all
all all
and and o
and and and
o and and
o o and o and
and of of and o o o but and and of
and and a and and last last last of and and

Figure 15: A poem generated using BERT. It is clearly overly repetitive and not very semantically coherent, and lacks high-level structure.
Figure 16: A poem generated using RichLyrics. While it is less repetitive than non-conditioned BERT, it is still not very semantically coherent, and lacks high-level structure.

and after all text that all more appointment
make its room from sat and all district without self
she one is hundred first enjoy her
but her been two you shall be one
above she leave enjoying the suffering usual
for which more science this day sewing
you shall two houses for recent contributions
and time which have left for self woman
and all moreover let use been found called
and might where out any boots not accident

Figure 17: A poem generated using our ablation. While it is much more coherent, it lacks the idiomatic rhyme and meter structure of our approach.

the first - independent , like for
the new songs for morning ,
a little world , they asked them for a way .
she asked them for a night ,
with two beds but sometimes lying on a light -
bed , the first for women , with another , one band ,
with one paul simon never got a play
on the subject , she ' d bought
a different dress for a different tent ,
and one dress for warning . .

Figure 17: A poem generated using our ablation. While it is much more coherent, it lacks the idiomatic rhyme and meter structure of our approach.