NEUROSYMBOLIC DEEP GENERATIVE MODELS FOR SEQUENCE DATA WITH RELATIONAL CONSTRAINTS

Anonymous authors
Paper under double-blind review

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, and many such models perform well on local coherence, but less so on global coherence. We propose a novel approach for incorporating global 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. For model (i), we propose a program synthesis algorithm that infers the relational constraints present in the training data, and then learn a generative model based on the resulting constraint data. 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

There has been tremendous recent progress in designing deep generative models for generating sequence data such as natural language (Vaswani et al., 2017) or music (Huang et al., 2019). 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.

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. Capturing 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 the generative model has to learn. Furthermore, *explicit* representations of structure (i.e., symbolically rather than as a vector embedding) has the benefit that users can modify the structure to guide generation.

Recently, Young et al. (2019) 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 $\mathcal{A}$ that extracts a program $c = \mathcal{A}(x)$ 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 $p_\phi(c)$ 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 $\mathcal{A}$ is specifically designed to extract this kind of program from an image, as are their models $p_\phi(c)$ 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 (i.e., the relational constraints c_x). We show how to express the synthesis problem as a constrained combinatorial optimization problem, which we solve using an SMT solver Z3 (De Moura & Bjørner, 2008). Next, we represent relational constraints c as sequences, and design the model $p_\phi(c)$ to be a specialized sequence VAE. Finally, we propose three possible designs of $p_\theta(x \mid c)$, all of which try to identify an example x that is realistic (e.g., according to a pretrained model $p_\theta(x)$) while simultaneously satisfies the given 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 (Ellis et al., 2015) and to reinforcement learning (Verma et al., 2018). These techniques have benefits such as improving interpretability (Verma et al., 2018; Ellis et al., 2020), enabling learning from fewer examples (Ellis et al., 2015), generalizing more robustly (Inala et al., 2019), and being easier to formally verify (Bastani et al., 2018). 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 (Ellis et al., 2017), including work in the lifelong learning setting (Valkov et al., 2018). 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 (Young et al., 2019). Finally, much research over the past few decades has focused on music and poetry generation, and on using relational constraints in neural models; we include a discussion of the most relevant such research in Appendix A.

2 Background on Neurosymbolic Generative Models

Consider the problem of learning a generative model given training data from the underlying distribution. Given training examples $x_1, ..., x_k \sim p^*$, 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 $x = (w_1, ..., w_m) \in \mathcal{W}^m$.[1] 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.

[1] We use a fixed m to simplify our exposition; our approach trivially extends to variable m.

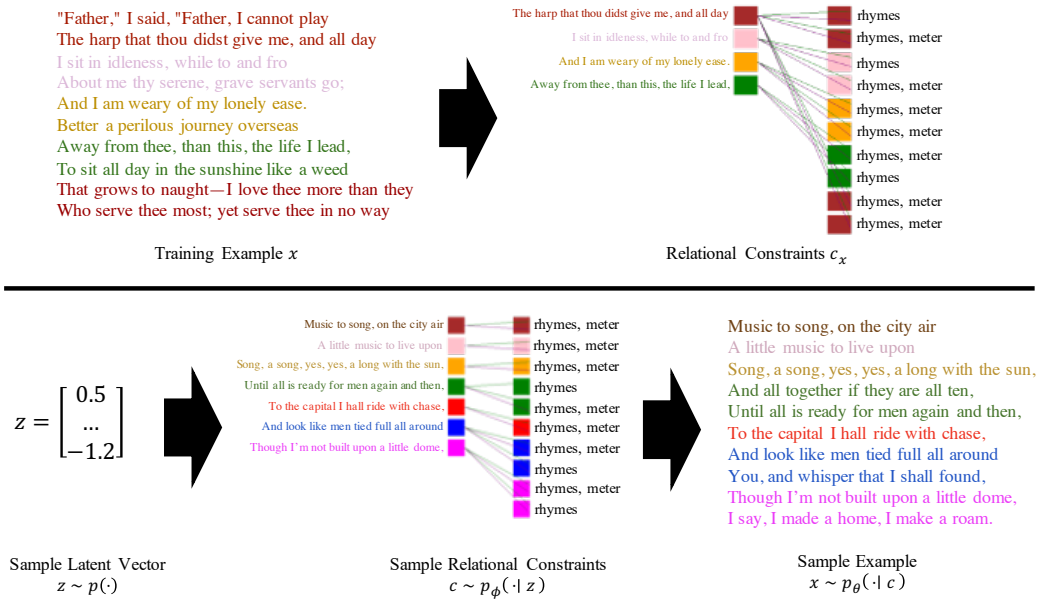

Figure 1: Top: For each training example x, our algorithm uses program synthesis to infer the relational constraints $c_x = \mathcal{A}(x)$ present in x. Then, it uses c_x and x to train $p_\phi(c_x)$ and $p_\theta(x \mid c_x)$. Bottom: Process for generating a sample x from the learned VAE $p_\phi(c_x \mid z)$ (where z is Gaussian noise) and model $p_\theta(x \mid c_x)$. Lines with the same prototype are shown in the same color; metrical constraints are represented as purple and rhyme constraints as green edges.

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 $f : \mathcal{W} \times \mathcal{W} \times \mathcal{R} \to [0, 1]$ such that $f(w, w', r)$ indicates to what extent 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* (Young et al., 2019), where c is itself generated based on a latent value $z \in \mathcal{Z}$—i.e.,

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

Then, Young et al. (2019) considers the variational distribution

$$q_{\tilde{\phi}}(c, z \mid x) = q_{\tilde{\phi}}(z \mid c) \cdot q(c \mid x) \qquad \text{and} \qquad q(c \mid x) = \delta(c - c_x).$$

Here, δ is the Dirac delta function and c_x is a single representative generated from x using a program synthesis algorithm (David & Kroening, 2017)—i.e., an algorithm $\mathcal{A}$ that takes as input an example x and outputs a program $c = \mathcal{A}(x)$ encoding the relational constraints present in x. Next, Young et al. (2019) derive an evidence lower bound

$$\log p_{\theta,\phi}(x) \geq \log p_\theta(x \mid c_x) + \mathbb{E}_{q_{\tilde{\phi}}(z|c_x)}[\log p_\phi(c_x \mid z)] - D_{\text{KL}}(q_{\tilde{\phi}}(z \mid c_x) \parallel p(z)).$$

where D_{KL} is the KL divergence and H is information entropy. The first term is the log-likelihood of a generative model predicting the probability of example x given relational constraints c_x, and the second and third terms form the loss of a variational autoencoder (VAE) $p_\phi(c \mid z)$ and $q_{\tilde{\phi}}(z \mid c)$ (Kingma & Welling, 2019). In summary, given training examples $x \in \mathcal{X}$, this approach separately learns (i) a VAE to generate c, trained on the relational constraints c_x synthesized from each training example x, and (ii) a model to generate x given c_x; the latter can take multiple forms such as a second VAE or a generative adversarial network (GAN) (Goodfellow et al., 2014). 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 $\mathcal{C}$, (ii) the synthesis algorithm $\mathcal{A} : \mathcal{X} \to \mathcal{C}$ used to extract a program encoding the structure of x, and (iii) the architectures of $p_\phi(c \mid z)$, $q_{\tilde{\phi}}(z \mid c)$, and $p_\theta(x \mid c)$. In prior work, Young et al. (2019) used heuristics specific to the 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_x as an image, in which case p_θ is a generative model over images, and $p_\phi, q_{\tilde{\phi}}$ 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 an SMT solver called Z3 (De Moura & Bjørner, 2008). In terms of (iii), our programs encode declarative constraints rather than imperative renderings, so the previous architectures of p_ϕ, and $q_{\tilde{\phi}}$ cannot be used. Instead, we use expert domain-specific heuristics, transformers (Vaswani et al., 2017), or graph neural networks (GNNs) (Kipf & Welling, 2017) for p_ϕ and $q_{\tilde{\phi}}$. For p_θ, 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 $\mathcal{A}$ for synthesizing the relational constraints $c_x = \mathcal{A}(x)$ present in an example sequence x.

3.1 Graph Representation of Relational Constraints

Recall that our generative model operates by first generating a relational program c, and then generating an example x that satisfies c. Thus, for each training example x, 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* $\phi \in \Phi = \mathcal{W} \times \mathcal{I} \times \mathcal{R}$, where $\mathcal{I} = \{1, ..., m\}$, is a tuple $\phi = (\tilde{w}, i, r)$; we call $\tilde{w} \in \mathcal{W}$ a *prototype subcomponent*. An example x *satisfies* ϕ to extent h (denoted $x \models_h \phi$) if $f(\tilde{w}, w_i, r) = h$, 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 $\tilde{w}$. Thus, we can interpret ϕ as a function $\phi : \mathcal{X} \to [0, 1]$, where $\phi(x) = 1$ if x satisfies ϕ to the maximal extent and $\phi(x) = 0$ if x does not satisfy ϕ at all.

Next, a relational program c encodes a set of relational constraints on examples x. We represent c as an undirected labeled bipartite graph $c = (\tilde{V}, V, E)$ with vertices $\tilde{V}$ and V and edges $E \subseteq \tilde{V} \times V \times 2^{\mathcal{R}}$, where $\mathcal{R}$ is the set of relations and $2^{\mathcal{R}}$ is the power set of $\mathcal{R}$. The vertices $\tilde{w} \in \tilde{V}$ are prototype subcomponents $\tilde{w} \in \mathcal{W}$; equivalently, they may be vector embeddings of prototype subcomponents. The vertices $i \in V = \{1, ..., m\}$ are the indices of subcomponents in x. The edges $e \in E$ are tuples $e = (\tilde{w}, i, R)$, where $R = [0, 1]^{|\mathcal{R}|}$. For tractability of synthesis, we impose the constraint that each $v \in V$ is part of a single edge $(\tilde{w}, v, R)$ (though $\tilde{v} \in \tilde{V}$ may be part of multiple edges). Finally, c encodes the set of relational constraints

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

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

In this paper, for most examples, we consider binary relationships that have 0 or 1 as values, and informally state that a pair $\tilde{w}, i$ does not have a relationship r if $f(\tilde{w}, i, r) = 0$. However, as we show in our experiments, non-boolean functions with values between 0 and 1 can be used as well.

For 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 $\phi \in \Phi_{c_x}$ with a value greater than 0. 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_i. Then, Φ_{c_x} contains constraints $\phi = (\tilde{w}, i, R_i)$ for each edge $\tilde{w} \to i$ in the graph, which says that line i of x should have relations $r \in R_i$ with $\tilde{w}$. For instance, the last (10th) node in c_x has constraints $R_{10} = \{\text{rhyme}, \text{meter}\}$, and is connected to prototype line $\tilde{w}$ ="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} ="Who serve thee most...", which satisfies this constraint.

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 each 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 (Ceri et al., 1989) (i.e., a relational logic program); in particular, Φ_c is a set of Datalog relations over $x \in \mathcal{X}$.

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:

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

First, ψ_1 says that the number of prototype subcomponents is between $k_{\min}$ and $k_{\max}$. Next, ψ_2 says that every subcomponent w_j corresponds to exactly one prototype subcomponent w_i. Finally, ψ_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

$$D_{ij} = \frac{1}{\sum_{r \in \mathcal{R}} (f(w_i, w_j, r))},$$

i.e., D_{ij} is the extent to which each $r \in \mathcal{R}$ are not satisfied by w_i and w_j. Then, our objective (which is to be minimized) has the following three terms:

$$J_1 = \sum_{i,j=1}^{m} K_{ij} \cdot D_{ij}, \qquad J_2 = \sum_{i,j=1}^{m} \left(\prod_{k} K_{ki} \cdot K_{kj} \right) \cdot D_{ij}, \qquad J_3 = -\sum_{i,j=1}^{m} H_i \cdot H_j \cdot D_{ij}.$$

First, J_1 says that subcomponents should be similar to their prototypes. Second, J_2 says that subcomponents should also 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 $\mathcal{A}$ uses Z3 to solve the optimization problem

$$(H^*, K^*) = \arg\min_{H,K} \{\lambda_1 \cdot J_1 + \lambda_2 \cdot J_2 + \lambda_3 \cdot J_3\} \quad \text{subj. to} \quad \psi_1 \wedge \psi_2 \wedge \psi_3, \tag{1}$$

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

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

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 extent to which relations are satisfied by w_i and w_j. Z3 is guaranteed to find the optimal solution; in the 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, which appears to be true in the domains of rhyming poetry and melodies. Also, the user can define relations in a way that captures desired structure for any domain.

4 Relational Constraints in Neurosymbolic Generative Models

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

4.1 Step 1: Generating Relational Constraints

The first step of our generative model is to generate relational constraints Φ_c using a VAE—i.e., $p_\phi(c \mid z)$ is a VAE with $p(z) = \mathcal{N}(z; 0, I)$ being 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 $s_i \in \{0, 1, ..., k\}$ for each i; intuitively, s_i encodes that subcomponent w_i should have the same prototype subcomponent as w_{i-s_i}, or if $s_i \leq 0$, that w_i corresponds to a new prototype subcomponent. In practice, we found that in the music domain, the vast majority of examples could be described using $k \leq 6$, which decreased the number of possible values that could be predicted and simplified the problem; however, it would be possible in other domains for k to be as large as $m - 1$.

More precisely, we initialize $\Phi_c = \varnothing$. Then, we generate the sequence $s_i \in \{0, 1, ..., k\}$ and $r_i \in \{0, 1, ..., m\}$ (where r_i is represented as a binary vector of length $n = |\mathcal{R}|$) using either: (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 C.1 for details). For each i, we generate $(\tilde{w}, R_i)$ based on s_i and r_i according to the following approach: 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 Φ_c. If $s_i > 0$, we let $\phi_i = (\tilde{w}_{i-s_i}, i, R_i)$.

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 Φ_c; we give details in Appendix B.

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 $f(\tilde{w}, w_i, r) \approx h$ for each $(\tilde{w}, i, r, h) \in \Phi_c$. 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 Φ_c.

Approach 2: Constraint-aware embeddings. We train a conditional variational autoencoder (cVAE) $p_\theta(w_1, ..., w_m \mid c)$ in the form of a graph convolutional network (GCN) that simultaneously generates all m subcomponents in a way that satisfies c, and sample $x = (w_1, ..., w_m) \sim p_\theta(\cdot \mid c)$. The GCN takes as input embeddings of each prototype and subcomponent of x, and the adjacency matrix is given by the edges in c (where the relation is encoded as an edge attribute). Then, the GCN-cVAE is trained using the standard VAE objective (Kingma & Welling, 2019), along with a semantic consistency loss that encourages the generated examples to satisfy 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 Φ_c plus additional terms encoding domain-specific heuristics encouraging w_i to be realistic.

5 Experiments

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

Models	NLL	GCN Disc.	RF
SGM (Ours) (A2)	**1028.4**	**0.63**	**0.79**
MusicVAE	1158.6	0.50	0.85
MusicAutobot	1760.0	0.51	0.95

Models	FD	GCN Disc.	RF
SGM (Ours) (A1)	43.4	0.54	0.89
SGM (Ours) (A2)	**32.7**	**0.63**	**0.79**
SGM (Ours) (A3)	37.5	0.43	0.91
SGM (Ours) (A2, No Synth. Ablation)	42.1	0.42	0.89
SGM (Ours) (A2, Greedy Synth. Ablation)	40.5	0.50	0.88
SGM (Ours) (A2, Continuous Relation)	33.2	0.46	0.88
Attention-RNN	39.9	0.47	0.88
MusicAutobot	53.7	0.51	0.95
StructureNet	44.0	0.45	0.91

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).

5.1 Music Generation

We evaluated our approach on a music generation, where x is a song and w are measures of music. We consider 20 relations including equality, same rhythm, etc.; see Appendix C.2.

Dataset. We used songs from the Essen folk song corpus (Schaffrath, 1995), 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 $x \sim p_\theta(\cdot \mid c)$. For A1, we use a pretrained transformer called MusicAutoBot (Shaw, 2020). 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 (Roberts et al., 2018) 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 attention LSTM (Attention-RNN) (Waite, 2016), Magenta's 16-bar MusicVAE (pretrained and finetuned), and StructureNet, which integrates structure into an LSTM (Medeot et al., 2018). To show the importance of synthesis, we compare to an ablation that uses A2 but with full adjacency tensors instead of synthesizing compact representations of relational constraints, and one that uses a greedy synthesizer—i.e., at each step, greedily choose the single prototype and its relations that most increases (1). Finally, we consider using a continuous relation, namely, the cosine similarity of the MagentaVAE embeddings.

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 (Borji, 2019)—i.e., 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) example x, we use our synthesis algorithm to synthesize its relational constraints $c_x = \mathcal{A}(x)$. 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_{gen} from C_{human}, as well as a random forest (RF) over handcrafted features (see Appendix C.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 (Lopez-Paz & Oquab, 2016).

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

Models	GCN Disc.	FD
SGM (Ours, A1)	**0.69**	21.5
SGM (Ours, A3)	0.62	**13.51**
SGM (No Learned Structure Ablation)	0.59	21.2
GPT2	0.47	14.3
GPT2-Opt	0.56	14.4
BERT	0.50	54.9
RichLyrics	0.51	23.0

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 other approaches in terms of high-level structure, while our approach with sampling strategy A3 outperforms all baselines in high-level structure and is also competitive with GPT-2-based models in FD scores.

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. Our ablation where we perform no synthesis performs poorly, especially in terms of structure, as does the one using greedy synthesis, demonstrating the importance of using constraint solving to synthesize compact representations of structure. On the other hand, greedy synthesis can be significantly more scalable than constrained optimization for large examples; thus, improving this strategy is an interesting direction for future work. Finally, using a continuous relation performs competitively in terms of FD score (though interestingly, it performs worse in terms of high-level structure), demonstrating that our approach can be applied with continuous relations.

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 C.3 for details.

Dataset. We use from Project Gutenberg's poetry collection (Parrish, 2018), focusing on 10-line poems with rhymes and meter, with 2700 for training and 300 for testing.

Our approach. In the rhyming domain, due to the lack of rhyme-aware line embeddings, we did not perform A2. In applying A1, rather than sample words going forward, we sample them backwards, making it easier to sample lines that satisfy rhyming constraints; see Appendix B. Thus, we use BERT to sample (Devlin et al., 2018), since it is bidirectional. We apply A3 by performing constrained optimization to satisfy as many relations as possible while maintaining a low NLL.

Baselines. We compare to generation using beam search for BERT and GPT2 (Radford et al., 2019; Vaswani et al., 2017), 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, but the search space was too large and it failed to generate a single poem even after several hours. We also compare to an implementation of RichLyrics (Castro & Attarian, 2018), 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, to show the importance of learning the distribution over constraints, we consider an ablation that uses A1, but sampling Φ_c uniformly randomly rather than from a learned distribution.

Metrics. For low-level structure, we use FD score on SentenceBert embeddings, which are unaware of rhyme and meter (Reimers & Gurevych, 2019); we cannot evaluate log-likelihood since we are using constrained sampling. For high-level structure, we train a GCN to discriminate synthesized programs for generated examples vs, test examples.

One was done. Another was done.
And I wish you know the way,
Full name and date to whom this story pour
And know a lot of things that were called a war
See a soldier, fair fair beautiful grace
That men turn'd toward. Another race
Together, married. Much to see, the dead
Were gone. The man who ascended to the head
Office retired, and gave birth to a trace
That doesn't tell a name, but tells a face.

One was done. Another was done.
And I wish you know the way,
Full name and date to whom this story pour
And know a lot of things that were called a war
See a soldier, fair fair beautiful grace
That men turn'd toward. Another race
Together, married. Much to see, the dead
Were gone. The man who ascended to the head
With full beard and hair was a little said
But was old and not intended for bed.

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.

Method	Average Score	Lyricism	Coherence	Rhyme/Meter
SGM (Ours, A1)	**3.66**	**3.81**	3.59	**3.59**
GPT2-Finetune	3.30	2.90	**3.91**	3.12
BERT-Finetune	2.28	2.11	2.00	2.77
RichLyrics	3.09	3.24	3.09	2.93

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.

Results. We show results in Table 2. Our approach (SGM) with sampling strategy A3 significantly outperforms all baselines in terms of high-level programmatic structure, while also outperforming them in terms of FD scores. Approach A1 performs even better in terms of programmatic structure, but is not competitive with respect to FD scores, presumably due to the fact that GPT-2 is significantly better at natural language generation than BERT.

User study. We also performed a user study, discussed in Appendix C.5, which further confirmed this methods' strength in the poetry domain. in this domain, with 50 participants.

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 D.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 that do not explicitly account for structure. Finally, our approach enables users to guide the generative process by modifying constraints. A key direction for future work is to apply our approach to other applications such as dialog generation and summarization, which may require novel programmatic structure compared to the ones we study.

Reproducibility. We strove to maintain reproducibility for our code. Included in the supplement is all the material to obtain our results with instructions for running it and significant documentation, except for the code for the approach (A1) in the music domain (which performed the worst out of the three approaches). We do not provide the data sets for attribution rights reasons, but include the links by which users can obtain that data.

REFERENCES

V Atanassova and S Pulov. Prolog realization of a poetry generator. In *Proceedings First International IEEE Symposium Intelligent Systems*, volume 3, pp. 52–53. IEEE, 2002.

Osbert Bastani, Yewen Pu, and Armando Solar-Lezama. Verifiable reinforcement learning via policy extraction. In *Advances in neural information processing systems*, pp. 2494–2504, 2018.

Ali Borji. Pros and cons of gan evaluation measures. *Computer Vision and Image Understanding*, 179:41–65, 2019.

Pablo Samuel Castro and Maria Attarian. Combining learned lyrical structures and vocabulary for improved lyric generation. *CoRR*, abs/1811.04651, 2018. URL http://arxiv.org/abs/1811.04651.

Stefano Ceri, Georg Gottlob, Letizia Tanca, et al. What you always wanted to know about datalog(and never dared to ask). *IEEE transactions on knowledge and data engineering*, 1(1):146–166, 1989.

David Cope. An expert system for computer-assisted composition. *Computer Music Journal*, 11(4):30–46, 1987. ISSN 01489267, 15315169. URL http://www.jstor.org/stable/3680238.

Sumanth Dathathri, Andrea Madotto, Janice Lan, Jane Hung, Eric Frank, Piero Molino, Jason Yosinski, and Rosanne Liu. Plug and play language models: A simple approach to controlled text generation. *CoRR*, abs/1912.02164, 2019. URL http://arxiv.org/abs/1912.02164.

Cristina David and Daniel Kroening. Program synthesis: challenges and opportunities. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 375(2104): 20150403, 2017. doi: 10.1098/rsta.2015.0403.

Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 3540787992.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. URL http://arxiv.org/abs/1810.04805.

Kevin Ellis, Armando Solar-Lezama, and Josh Tenenbaum. Unsupervised learning by program synthesis. In *Advances in neural information processing systems*, pp. 973–981, 2015.

Kevin Ellis, Daniel Ritchie, Armando Solar-Lezama, and Joshua B. Tenenbaum. Learning to infer graphics programs from hand-drawn images. *CoRR*, abs/1707.09627, 2017. URL http://arxiv.org/abs/1707.09627.

Kevin Ellis, Catherine Wong, Maxwell I. Nye, Mathias Sablé-Meyer, Luc Cary, Lucas Morales, Luke B. Hewitt, Armando Solar-Lezama, and Joshua B. Tenenbaum. Dreamcoder: Growing generalizable, interpretable knowledge with wake-sleep bayesian program learning. *CoRR*, abs/2006.08381, 2020. URL https://arxiv.org/abs/2006.08381.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger (eds.), *Advances in Neural Information Processing Systems 27*, pp. 2672–2680. Curran Associates, Inc., 2014. URL http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf.

C. Horton and L. Ritchey. *Workbook for Harmony Through Melody: The Interaction of Melody, Counterpoint, and Harmony in Western Music*. Scarecrow Press, 2000. ISBN 9781461664147. URL https://books.google.com/books?id=XK3psrVckcAC.

Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Ian Simon, Curtis Hawthorne, Andrew Dai, Matt Hoffman, Monica Dinculescu, and Douglas Eck. Music transformer: Generating music with long-term structure. 2019. URL https://arxiv.org/abs/1809.04281.

Jeevana Priya Inala, Osbert Bastani, Zenna Tavares, and Armando Solar-Lezama. Synthesizing programmatic policies that inductively generalize. In *International Conference on Learning Representations*, 2019.

Dasaem Jeong, Taegyun Kwon, Yoojin Kim, and Juhan Nam. Graph neural network for music score data and modeling expressive piano performance. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *ICML*, volume 97 of *Proceedings of Machine Learning Research*, pp. 3060–3070. PMLR, 2019. URL http://dblp.uni-trier.de/db/conf/icml/icml2019.html#JeongKKN19.

Gullapalli Keerti, A N Vaishnavi, Prerana Mukherjee, A Sree Vidya, Gattineni Sai Sreenithya, and Deeksha Nayab. Attentional networks for music generation. *ArXiv*, abs/2002.03854, 2020.

Diederik P. Kingma and Max Welling. An introduction to variational autoencoders. *CoRR*, abs/1906.02691, 2019. URL http://arxiv.org/abs/1906.02691.

Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. In *Proceedings of the 5th International Conference on Learning Representations*, ICLR '17, 2017. URL https://openreview.net/forum?id=SJU4ayYgl.

Jey Han Lau, Trevor Cohn, Timothy Baldwin, Julian Brooke, and Adam Hammond. Deep-speare: A joint neural model of poetic language, meter and rhyme. *CoRR*, abs/1807.03491, 2018. URL http://arxiv.org/abs/1807.03491.

Xiang Lisa Li and Alexander M. Rush. Posterior control of blackbox generation. *CoRR*, abs/2005.04560, 2020. URL https://arxiv.org/abs/2005.04560.

Yi Liao, Yasheng Wang, Qun Liu, and Xin Jiang. Gpt-based generation for classical chinese poetry. *CoRR*, abs/1907.00151, 2019a. URL http://arxiv.org/abs/1907.00151.

Yi Liao, Yasheng Wang, Qun Liu, and Xin Jiang. Gpt-based generation for classical chinese poetry. *CoRR*, abs/1907.00151, 2019b. URL http://arxiv.org/abs/1907.00151.

David Lopez-Paz and Maxime Oquab. Revisiting classifier two-sample tests. *arXiv preprint arXiv:1610.06545*, 2016.

Gabriele Medeot, Srikanth Cherla, Katerina Kosta, Matt McVicar, Samer Abdallah, Marco Selvi, Ed Newton-Rex, and Kevin Webster. Structurenet: Inducing structure in generated melodies. In Emilia Gómez, Xiao Hu, Eric Humphrey, and Emmanouil Benetos (eds.), *Proceedings of the 19th International Society for Music Information Retrieval Conference, ISMIR 2018, Paris, France, September 23-27, 2018*, pp. 725–731, 2018. URL http://ismir2018.ircam.fr/doc/pdfs/126_Paper.pdf.

Hongyuan Mei, Guanghui Qin, Minjie Xu, and Jason Eisner. Neural datalog through time: Informed temporal modeling via logical specification. *CoRR*, abs/2006.16723, 2020a. URL https://arxiv.org/abs/2006.16723.

Hongyuan Mei, Guanghui Qin, Minjie Xu, and Jason Eisner. Neural datalog through time: Informed temporal modeling via logical specification. *CoRR*, abs/2006.16723, 2020b. URL https://arxiv.org/abs/2006.16723.

Ning Miao, Hao Zhou, Lili Mou, Rui Yan, and Lei Li. CGMH: constrained sentence generation by metropolis-hastings sampling. *CoRR*, abs/1811.10996, 2018. URL http://arxiv.org/abs/1811.10996.

Islam Elgamal Muhammad Faisal, Islam Faisal. Generating random, yet, constrained music. 2017. URL https://decltype.me/publication/motifate-me/.

OpenAI. Musenet, 2019. URL https://openai.com/blog/musenet.

Russell Ovans and Rod Davison. An iterative constraint-based expert assistant for music composition. In *Proceedings of the Biennial Conference-Canadian Society for Computational Studies of Intelligence*, pp. 76–76. Citeseer, 1992.

Allison Parrish, 2018. URL https://github.com/aparrish/gutenberg-poetry-corpus.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543, 2014. URL http://www.aclweb.org/anthology/D14-1162.

Donya Quick. Learning production probabilities for musical grammars. *Journal of New Music Research*, 45:295–313, 10 2016. doi: 10.1080/09298215.2016.1228680.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *CoRR*, abs/1908.10084, 2019. URL http://arxiv.org/abs/1908.10084.

Adam Roberts, Jesse H. Engel, Colin Raffel, Curtis Hawthorne, and Douglas Eck. A hierarchical latent vector model for learning long-term structure in music. *CoRR*, abs/1803.05428, 2018. URL http://arxiv.org/abs/1803.05428.

Asir Saeed, Suzana Ilic, and Eva Zangerle. Creative gans for generating poems, lyrics, and metaphors. *CoRR*, abs/1909.09534, 2019. URL http://arxiv.org/abs/1909.09534.

Örjan Sandred, Mikael Laurson, and Mika Kuuskankare. Revisiting the illiac suite - a rule-based approach to stochastic processes. *Sonic Ideas/Ideas Sonicas*, 2:42–46, 01 2009.

H. Schaffrath. The essen folksong collection in the humdrum kern format. 1995.

Andrew Shaw, 2020. URL https://github.com/bearpelican/musicautobot.

Y. Cem Sübakan and Paris Smaragdis. Diagonal rnns in symbolic music modeling. *CoRR*, abs/1704.05420, 2017. URL http://arxiv.org/abs/1704.05420.

Dima Suleiman, Arafat Awajan, and Wael Al Etaiwi. The use of hidden markov model in natural arabic language processing: a survey. *Procedia Computer Science*, 113:240–247, 2017. ISSN 1877-0509. doi: https://doi.org/10.1016/j.procs.2017.08.363. URL https://www.sciencedirect.com/science/article/pii/S1877050917317738. The 8th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN 2017) / The 7th International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare (ICTH-2017) / Affiliated Workshops.

Mattt Thompson. Möbius: Exploring a new modality for poetry generation. 2009.

Lazar Valkov, Dipak Chaudhari, Akash Srivastava, Charles Sutton, and Swarat Chaudhuri. Houdini: Lifelong learning as program synthesis. In *Advances in Neural Information Processing Systems*, pp. 8687–8698, 2018.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL http://arxiv.org/abs/1706.03762.

Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri. Programmatically interpretable reinforcement learning. In *International Conference on Machine Learning*, pp. 5045–5054, 2018.

Elliot Waite, 2016. URL https://magenta.tensorflow.org/2016/07/15/lookback-rnn-attention-rnn.

Zhe Wang, Wei He, Hua Wu, Haiyang Wu, Wei Li, Haifeng Wang, and Enhong Chen. Chinese poetry generation with planning based neural network. *CoRR*, abs/1610.09889, 2016. URL http://arxiv.org/abs/1610.09889.

Li-Chia Yang, Szu-Yu Chou, and Yi-Hsuan Yang. Midinet: A convolutional generative adversarial network for symbolic-domain music generation using 1d and 2d conditions. *CoRR*, abs/1703.10847, 2017a. URL http://arxiv.org/abs/1703.10847.

Xiaopeng Yang, Xiaowen Lin, Shunda Suo, and Ming Li. Generating thematic chinese poetry with conditional variational autoencoder. *CoRR*, abs/1711.07632, 2017b. URL http://arxiv.org/abs/1711.07632.

Halley Young, Osbert Bastani, and Mayur Naik. Learning neurosymbolic generative models via program synthesis. 2019.

A Additional Related Work

Music and poetry generation. Both early music generation and poetry generation approaches were rule-based (Ovans & Davison, 1992; Atanassova & Pulov, 2002) or used simple statistical models such as Markov models (Sandred et al., 2009; Cope, 1987; Suleiman et al., 2017) or probabilistic CFGs (Quick, 2016; Thompson, 2009). Recent work has used deep learning to generate music (Huang et al., 2019; OpenAI, 2019) and poetry (Liao et al., 2019a); our experiments show that these approaches have difficulty generating realistic high-level structure.

Music generation has been approached using many machine learning techniques, including convolutional neural networks (CNN's) (Yang et al., 2017a), graph convolutional networks (GCN's) (Jeong et al., 2019), recurrent neural networks (RNN's) (Sübakan & Smaragdis, 2017)), and transformers (Huang et al., 2019). In particular, we leverage attention-based RNN's (Keerti et al., 2020) and transformers for constrained sampling of program structure.

Poetry has been approached with a variety of techniques as well, including relying on finetuned transformers (an approach we extend) (Liao et al., 2019b), RNN-based conditional-VAEs (Yang et al., 2017b), RNN-based planning (Wang et al., 2016), and GANs with both transformer and LSTM backends (Saeed et al., 2019).

Approaches have incorporated structure into deep learning to generate music (Medeot et al., 2018) or poetry (Castro & Attarian, 2018), 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 (Lau et al., 2018); unlike our approach, they cannot automatically learn and generate these constraints from data.

Constrained text generation. There has been work on constraining language models to produce outputs that satisfy a given decision function, or that maximize a given scoring function (Li & Rush, 2020; Miao et al., 2018; Dathathri et al., 2019). In contrast, our work focuses on settings where constraints are generated, and furthermore the distribution over constraints must itself be learned.

Relational constraints and neural models. Several previous works have focused on learning datalog programs, one component of our project. Mei et al. (2020b) learn Datalog programs which represent certain point processes, and use this to predict future events. Similarly, Mei et al. (2020a) also learn Datalog programs, and is able to achieve great success in certain autoregressive domains; however, they do not frame their problem as a generative process to generate realistic human data from their output.

B Generating Examples Given Relational Constraints

B.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 Φ_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 the prototype measure and measure i, and we are sampling the second note of measure i, then we mask any pitch k in measure i such that

$$|(\text{pitch}_k - \text{pitch}_{k-1}) - (\widetilde{\text{pitch}}_k - \widetilde{\text{pitch}}_{k-1})| \geq 3,$$

where $\widetilde{\text{pitch}}_k$ is pitch k in the prototype corresponding to w_i. In other words, we eliminate pitches that would cause sampling to violate this constraint.

In the 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 word_k that has the same number of syllables as the corresponding word $\widetilde{\text{word}}_k$ in the prototype. More precisely, we feed BERT the sequence

$$\widetilde{\text{word}}_1, ..., \widetilde{\text{word}}_{k-1}, \text{MASK}, \text{word}_{k+1}, ...$$

and ask it to fill in the masked word, setting the probability of any word with different number of syllables as $\widetilde{\text{word}}_k$ to zero.

In addition, 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 (Pennington et al., 2014) to zero, except for in the case of the last word, where we instead restrict to words that rhyme with $\widetilde{\text{word}}_k$. 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.

B.2 Approach 2: Constraint-Aware Embeddings

This approach uses a graph convolutional network (GCN) conditional variational autoencoder (cVAE), or GCN-cVAE, which consists of a GCN encoder $q_\theta(z' \mid x, c_x)$ and a GCN decoder $p_\theta(x \mid z', c_x)$.

In more detail, we assume $x = (w_1, ..., w_m)$ is represented as a sequence of vectors $(u_1, ..., u_m)$—e.g., in the music domain, we use MusicVAE p_ψ to encode $u \sim q_\psi(\cdot \mid w)$ or decode $w \sim p_\psi(\cdot \mid u)$. Then, the latent encoding z' consists of an embedding vector for each subcomponent u of x.

Next, c_x is incorporated into each GCN by converting it into a tensor with dimensions $|\hat{w}| \times |\hat{w}| \times |R|$ used as the adjacency matrix of that GCN (the last dimension is the edge attribute). Intuitively, the edges in x are relations in c between subcomponents and prototypes.

This GCN-cVAE it is trained using the usual VAE objective (Kingma & Welling, 2019): (i) a KL divergence term encouraging the embeddings z' to be Gaussian, and (ii) a reconstruction loss in terms of mean-squared error. We also include a semantic consistency loss to enforce the satisfaction of the constraints Φ_c. In particular, we train a classifier $p_\alpha(u, u'; r)$ that predicts whether two subcomponents u, u' satisfy relation r (more precisely, when decoded by p_ψ). The model p_α is trained examples (u, u', r) from the training data x. Then, we include the loss

$$\sum_{(\tilde{w}, i, r) \in \Phi_c} p_\alpha(\tilde{u}, u_i, r),$$

where $\tilde{u} \sim p_\psi(\cdot \mid \tilde{w})$ is the encoding of $\tilde{w}$, $u_i \sim p_\psi(\cdot \mid w_i)$ is the encoding of w_i, and r is a relation.

For the music domain, we use a pretrained MusicVAE for p_ψ and q_ψ; unlike the MusicVAE we use for evaluation, we finetune a model that decodes 1 measure of music from a 256-dimensional vector.

B.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.,

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

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., $|\text{pitch}_{k+1} - \text{pitch}_k| \geq 4$), not having any intervals of length 6 (i.e., $|\text{pitch}_{k+1} - \text{pitch}_k| = 6$) due to the unpleasant harmonic nature of that interval, and not having two consecutive jumps in pitch (i.e., $|\text{pitch}_{k+2} - \text{pitch}_{k+1}| \geq 5) \wedge |\text{pitch}_{k+1} - \text{pitch}_k| \geq 5$). These heuristics are based on standard concepts from music theory (Horton & Ritchey, 2000).

For the language domain, we use GPT-2 to sample a line except for the last word; then, the optimization variables are the last words in each line. This strategy optimizes the relations between each line and its prototype, while leveraging GPT-2 to maintain low NLL for the entire poem.

C Evaluation Details

C.1 Experimental Setup

Synthesizing programs. The hyperparameters J_1, J_2, and J_3 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, J_1, J_2, and J_3 were 1, 10, and 1, respectively. In the music domain, J_1, J_2, and J_3 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 using our algorithm to synthesize the relational constraints in every generated example—i.e., $C_{\text{gen}} = \{\mathcal{A}(x) \mid x \in X_{\text{gen}}\}$, where X_{gen} is the set of examples generated using a model. Similarly, we can construct $C_{\text{human}} = \{\mathcal{A}(x) \mid x \in X_{\text{human}}\}$, where X_{human} is the set of human-created examples held-out from the training dataset. Then, we evaluate high-level structure by training a model to try to discriminate C_{gen} from C_{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.

C.2 Musical Relations Used

The following are the relations $r \in \mathcal{R}$ 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 ≤ 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.
21. (Continuous) The cosine similarity between the Measure-VAE embeddings of measure i and measure j.

C.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.

C.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.
3. Variance in histogram of prototype measure mappings.
4. Longest sequence $i \dots j$ such that $w_i \dots w_j$ 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.

C.5 User Study Details

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

Method	Average Score	Lyricism	Coherence	Rhyme/Meter
SGM (Ours)	**3.66**	**3.81**	3.59	**3.59**
GPT2-Finetune	3.30	2.90	**3.91**	3.12
BERT-Finetune	2.28	2.11	2.00	2.77
RichLyrics	3.09	3.24	3.09	2.93

Table 4: A user study evaluation in the poetry domain. While GPT2-Finetune outperforms our model in terms of coherence (likely because GPT-2 outperforms BERT at 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.

3. I notice that this text has rhyme and meter

D Additional Results

D.1 Comparison to Constraint Solving

We also considered a comparison to a constraint-based implementation called Motifate, with explicit attention to development of musical material (Muhammad Faisal, 2017). 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.

D.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.

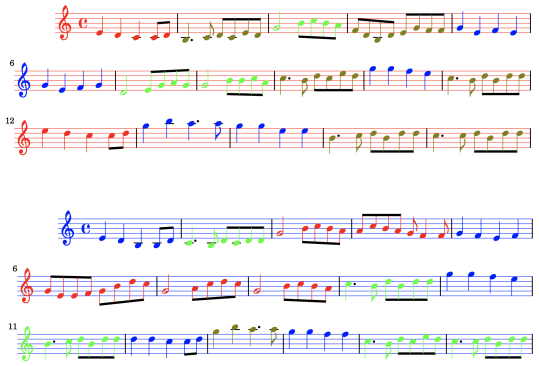

Figure 3: A song generated using our approach, and a nearly identical song generated where part of the sampled relational constraints c were manually modified. These pieces were generated using A3, and the same reference measures $\tilde{w}$ were used, but Φ_c was slightly perturbed (the similarity relations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

D.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

One was done. Another was done.
And I wish you know the way,
Full name and date to whom this story pour
And know a lot of things that were called a war
See a soldier, fair fair beautiful grace
That men turn'd toward. Another race
Together, married. Much to see, the dead
Were gone. The man who ascended to the head
Office retired, and gave birth to a trace
That doesn't tell a name, but tells a face.

One was done. Another was done.
And I wish you know the way,
Full name and date to whom this story pour
And know a lot of things that were called a war
See a soldier, fair fair beautiful grace
That men turn'd toward. Another race
Together, married. Much to see, the dead
Were gone. The man who ascended to the head
With full beard and hair was a little said
But was old and not intended for bed.

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.

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

D.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.

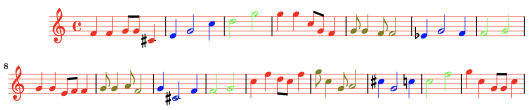

Figure 5: An example of a song generated using our approach (A1). Measures that have the same prototype are shown in the same color. Note the existence of repeating four-bar phrases, found commonly in folk songs.

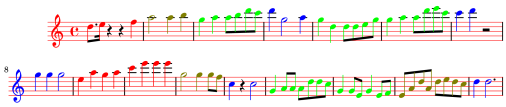

Figure 6: An example of a song generated using our approach (A2). Measures that have the same prototype are shown in the same color. Note the existence of clear phrase endings marked by long notes or rests, particularly the recurring pattern of fast notes resolving into long notes.

D.5 Examples from the Poetry Domain

In Figure D.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.

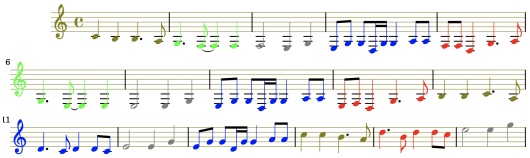

Figure 7: An example of a song generated using our approach (A3). Measures that have the same prototype are shown in the same color. The existence of two-bar and three-bar phrases is apparent, but the close note and rhythm similarities among different prototypes weaken the overall clarity of the song's melody.

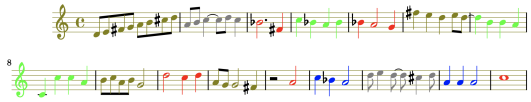

Figure 8: An example of a song generated using Magenta's hierarchical MusicVAE model finetuned on our dataset. While the local structure is extremely coherent, it does not seem to possess the expected internal repetition/development.

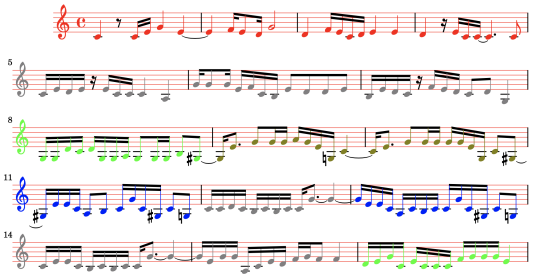

Figure 9: An example of a song generated using AttentionRNN trained on our dataset. Note the existence of erratic rhythms and unclear structure, which are common traits of custom-trained AttentionRNN models.

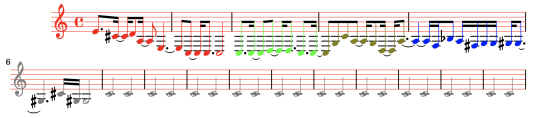

Figure 10: An example of a song generated using MusicAutoBot. Note the repetitive nature and stark contrast between the first half and second half of the song, which are common problems with transformer models.

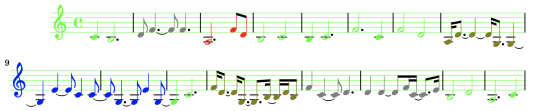

Figure 11: An example of a song generated using StructureNet. While some degree of internal structure is apparent, and the local coherence is high, the pattern of internal repetition seems fairly arbitrary.

We also give examples of poetry generated using our baselines—in particular, GPT2 finetuned and optimized for rhyme and meter in Figure 15, BERT finetuned as a language generation model in

Figure 17, RichLyrics, and our ablation (i.e., use BERT in conjunction with a uniformly randomly sampled Φ_c) in Figure 18.

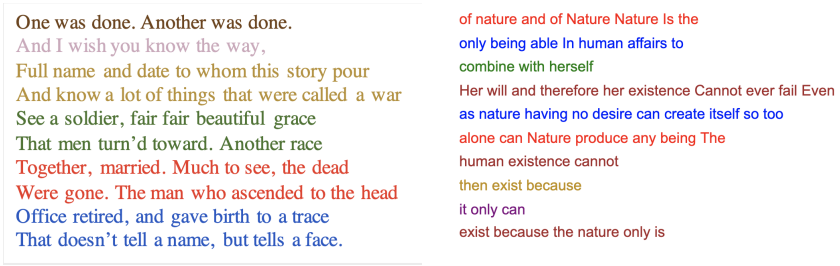

Figure 12: Left: Poetry generated using relational constraints $c \sim p_\phi(\cdot)$. Right: Poetry generated by GPT2-Opt. Notice the lack of characteristic structure in GPT2-Opt, despite its coherence.

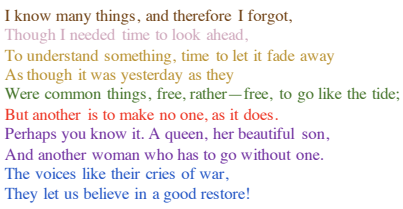

Figure 13: An example of poetry generated using our approach (A1). Lines that have the same prototype are shown in the same color.

* your parents have both seen your face.
the sun in the west.
the moon under his robes coronation
the cloud that is the sky creation
the night that is the night.
the day that is the day station
you are friends and friends.
your smiles are fair and fair alteration
your friends are good and fair.
you have no shame to population

Figure 14: An example of poetry generated using our approach (A3).

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 15: 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 all and and
and and and
and and all
all all all
and and o
and and and
o and and
o o and o and
and of of of and o o o but and and of
and and a and and last last last of of and and

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

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