the occurrence of the preceding subdominant and dominant harmony), which in [4] are identified with specialized heuristics. In total, we store 135 features per node.

4. PROBLEM SETTING & CORPORA

In this work, we are interested in cadences of the Baroque and Classical periods. The main focus will be on detecting Perfect Authentic Cadences (PAC); where our annotated datasets permit, we will also consider root position Imperfect Authentic Cadences (rIAC) and Half Cadences (HC). The manual annotations in these datasets mark a cadence as occurring on the beat where the final I (i) arrives. Our precise task thus is to predict, for every note of the score, whether this note is contained in a cadence's arrival beat.

To benchmark our method, we used two datasets also used by Bigo et al. [4], and a third one annotated by Allegraud and al. [18]. The first set contains the 24 fugues from Bach's Well-tempered Clavier, Book I. The cadence annotations were presented in [11]. The second dataset contains 45 movement expositions from Haydn string quartets; the cadence annotations were produced by Sears and colleagues [19]. The last dataset contains 31 movements of Mozart string quartets with cadence annotations included. All the scores were retrieved from `http://kern.ccarh.org` and were parsed in python using the partitura package [14]. [2]

Cadences occur with low frequency in music. In particular, for the corpora we cover in this paper, cadences of all types combined account for less than 2% of the total notes in the score. Our produced score graphs range from approximately 25k nodes for the Bach fugues all the way to 70k nodes for the Mozart string quartets with more than 750k edges. Table 1 gives detailed dataset statistics.

5. MODEL

5.1 Graph Convolutional Network

The authors of [4] underline the importance of non-fixed positions for the cadence anchor points. We address this by employing a graph convolutional network. Graph Convolution Networks (GCNs) are based on the same principle as CNNs, but in the context of graphs we encounter the message passing concept, meaning convolution occurs only among nodes connected by edges. This theoretically allows local features to connect with distant features of their k-hop neighbors. Therefore, graph representation can learn, using local node information, higher lever information by sampling information from neighbors. Figure 2 illustrates the neighbor sampling concept.

For our model, we propose *Stochastic GraphSMOTE*, a Graph Convolutional Network with a built-in graph Auto-Encoder and Synthetic Minority Over-sampling for imbalanced node classification. The model consists of 4 parts, the encoder, a SMOTE layer in the encoder's latent space, the decoder, and the classifier. The structure of the model

[2] For reproducibility, we provide the generated graphs that were used for training on `https://github.com/manoskary/tonnetzcad`

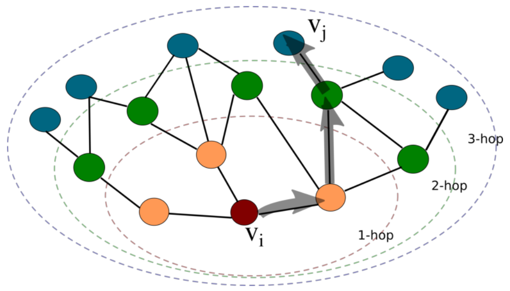

Figure 2. Multi-hop Neighborhood sampling. v_j is 3-hop neighbor of v_i. Color cues mark the k-hop neighborhoods occurring within the ellipses. The arrows demonstrate a random walk starting from v_i and ending at v_j.

follows GraphSMOTE [5] but with some major differences, mainly to adapt for stochastic training, which is needed because of the large size of our score graphs.

The encoder applied to a node i is defined as a standard GraphSAGE [20] stack given by:

$$\begin{aligned}
\mathbf{h}_{\mathcal{N}(i)}^{(l+1)} &= \mathrm{mean}\left(\{\mathbf{W}_{pool}^{(l+1)} \cdot \mathbf{h}_j^l, \forall j \in \mathcal{N}(i)\}\right)\\
\mathbf{h}_i^{(l+1)} &= \sigma\left(\mathbf{W}_{enc}^{(l+1)} \cdot \mathrm{concat}(\mathbf{h}_i^l, \mathbf{h}_{\mathcal{N}(i)}^{l+1})\right)\\
\mathbf{h}_i^{(l+1)} &= \mathrm{norm}(\mathbf{h}_i^l)
\end{aligned}$$

where $h_i^{(l)}$ is the hidden representation of node i on layer l, σ is an activation function, norm is a normalization function, $\mathbf{W}$ are learnable weights, and $\mathcal{N}(i) = \{j \mid (i,j) \in E\}$ are the neighbors of node i. Let $B \subseteq V$ a subset of nodes denoting a batch sample. Then, given L the total number of hidden layers, $\mathbf{H}_B^{(enc)} = \{\mathbf{h}_u^{(L+1)} \mid u \in B\}$.

5.2 Dealing with Extreme Class Imbalance: Stochastic GraphSMOTE

Since cadences are very sparse, we need to introduce a balancing technique in order to avoid gradient convergence that will result in predicting only the majority class, i.e., absence of cadence. To counter this effect, we introduce a SMOTE layer that is applied in the *latent space* of the encoder. SMOTE generates synthetic samples with the same label as the minority class (see [21] for details). The main novelty of our model is that the SMOTE is performed for each batch separately.

In each batch, we count the occurrence, μ_i, for each of the classes $i \in I$. In the binary setting, let μ_M be the number of samples with the same label as the majority class and μ_m be the number of samples with the same labels as the minority class. By generating $(\mu_M - \mu_m)$ samples with the same label as the minority class, we force a $1:1$ binary class distribution. To generate these samples, in each batch, we randomly select a sample instance of the minority class as an anchor point and gather the k nearest neighbor samples of the same class within the batch. Finally, μ samples are generated as random linear interpolations between a randomly selected neighbor out of the k, and the selected anchor point in the euclidean space. Performing

Dataset	Pieces	Nodes	Edges	PAC	rIAC	HC
Bach Fugues	24	24,567	229,107	237	78	15
Haydn String Quartets	45	38,661	441,491	434	24	340
Mozart String Quartets	31	68,190	762,796	1,089	-	1,930

Table 1. Cadence nodes constitute less than 2% of all nodes.

SMOTE in the latent space assumes that a more appropriate representation for the generation of the synthetic minority samples is learned.

If $\mathbf{H}_B^{(enc)}$ is the hidden representation of the batch sampled nodes after the encoder layer, then $\mathbf{H}_B^{(smote)}$ is the SMOTE upsampling algorithm applied on $\mathbf{H}_B^{(enc)}$. Our Decoder layer is responsible for generating edges within the original nodes of the graph and the synthetic ones, created by SMOTE. The decoder output is described by the following equation:

$$\mathbf{A}_B^{(dec)} = \sigma\left(\mathbf{H}_B^{(smote)} \cdot \mathbf{W}^{(dec)} \cdot \text{transpose}(\mathbf{H}_B^{(smote)})\right)$$
$$\mathbf{A}_B^{(thr)} = \text{hardshrink}\left(\mathbf{A}_B^{(dec)}, \tau\right)$$

where $W^{(dec)}$ are the decoder's learnable weights, σ is a sigmoid activation function, and hardshrink is the hard shrinkage function with threshold τ. $\mathbf{A}_B^{(dec)}$ is the generated adjacency from the decoder and $\mathbf{A}_B^{(thr)}$ is a thresholded adjacency by a factor τ.

We define a regularization loss that aims at constraining the generated adjacency close to the original, defined by:

$$\mathcal{L}_B^{(dec)} = \text{BCE}\left(\mathbf{A}_B^{(dec)}, \mathbf{A}_B\right)$$

where BCE is the binary cross entropy loss, $\mathbf{A}_B^{(dec)}$ is the generated adjacency of the decoder for batch sample B and A_B is the adjacency matrix for batch sample B. Since we learn an edge generator which is good at reconstructing the adjacency matrix using the encoder's latent representations, it should also give adequate edge predictions for synthetic nodes.

The GNN classifier is composed of a GraphSAGE layer [20] with a linear layer on top. By adding a graph convolution layer such as GraphSAGE in the classifier, we can benefit from learning information from the generated adjacency and the neighbors of nodes. The GraphSAGE layer of the classifier is slightly different from the encoder because it performs directly on the generated thresholded adjacency of each batch sample:

$$\mathbf{h}_{\mathcal{N}(i)}^{(clf)} = \text{mean}\left(\mathbf{W}^{(pool)} \cdot \mathbf{A}_B^{(thr)}[i,:] \cdot \mathbf{H}_B^{(enc)}\right)$$
$$\mathbf{h}_i^{(clf)} = \text{norm}\left(\sigma\left(\mathbf{W}^{(clf)} \cdot \text{concat}\left(\mathbf{h}_i^{(enc)}, \mathbf{h}_{\mathcal{N}(i)}^{(clf)}\right)\right)\right)$$
$$\mathbf{h}_i^{(clf)} = \text{softmax}(\mathbf{W}^{(proj)} \cdot \mathbf{h}_i^{(clf)})$$

where $\mathbf{h}_i^{(clf)}$ are the predicted class probabilities of node i, $\mathbf{W}$ are learnable weights, $\mathbf{A}_B^{(thr)}$ is the generated thresholded adjacency from the decoder, $\mathbf{H}_B^{(enc)}$ are the batch encodings of the encoder and $\mathbf{h}_i^{(enc)}$ is the encoder's output for node i. During training, we use $\mathbf{H}_B^{(smote)}$ and $\mathbf{h}_i^{(smote)}$ respectively instead of $\mathbf{H}_B^{(enc)}$ and $\mathbf{h}_i^{(enc)}$. We define the *total loss* of our model for batch samples B:

$$\mathcal{L}_B^{(tot)} = \mathcal{L}_B^{(CE)} + \gamma * \mathcal{L}_B^{(dec)}$$

where $\mathcal{L}_{CE}$ signifies the cross entropy loss and γ is a hyper parameter.

Our model is trained stochastically, meaning that to create each batch a subset B of nodes are sampled. From these sampled nodes, given a pre-defined depth k, we retrieve the immediate neighbors of every $v \in B$ up to their k-hop neighbors in the graph G. We use neighbor sampling to reduce the cost of retrieving all up to k-hop neighbors of v by defining a maximum number ϕ_l of neighbors per depth layer l.

6. EXPERIMENTS

We conduct three main experiments. The first compares our model to the state of the art results in [4], using the same data and train/test setup. The second experiment focuses on multi-class learning of the particular type of cadence using different sets of features, in order to investigate how the model generalizes to a more complex setting and inspect the relevance of different feature sets. The third experiment investigates how neighbor convolution contributes to the model's performance. [3]

We fix our model with a hidden dimension of 256, with $L = 2$ hidden layers with $\phi_1 = 10$ and $\phi_2 = 25$ sampled neighbors for hidden layers 1 and 2 of the encoder, respectively, and one hidden layer of the same dimension for the classifier. The learning rate is set at 0.007, the weight-decay at 0.007, with a batch size of 1024, $k = 3$ for SMOTE, the decoder regularization loss multiplier $\gamma = 0.5$, and adjacency threshold value $\tau = 0.5$.

6.1 Quantitative Results

Table 2 summarizes the results of the first experiment, comparing our model's performance to the state of the art. [4] The reference model [4] can only classify at the beat level; our representation and classification model are more flexible in this regard, as they have access to, and describe, individual notes. In particular, our model can provide predictions at three different levels, note-wise, onset-wise and beat-wise predictions (the latter two simply by aggregation). In Table 2 we present the results of these predictions at all levels,

[3] All results, experiments, and the trained models are available on `https://wandb.ai/melkisedeath/CadenceDetection`

[4] In accordance with [4], we ignore the HC in Bach and rIAC in Haydn, because of their low numbers.

Dataset	Model	F1 Note	F1 Onset	F1 Beat	Prec. Beat	Recall Beat
	Bigo et al. model	-	-	**0.80**	**0.89**	0.72
Bach Fugues (PAC)	SGSMOTE	0.85	0.75	0.73	0.70	0.77
(12 fugues)	Pretrained SGSMOTE	**0.90**	**0.83**	**0.80**	0.74	**0.89**
	Bigo et al. model	-	-	0.68	0.71	0.65
Bach Fugues (rIAC)	SGSMOTE	**0.87**	**0.75**	**0.73**	**0.75**	0.72
(12 fugues)	Pretrained SGSMOTE	0.87	0.73	0.71	0.62	**0.82**
	Bigo et al. model	-	-	**0.69**	**0.60**	**0.82**
Haydn String Quartets (PAC)	SGSMOTE	0.77	0.56	0.59	0.47	0.78
(21 pieces)	Pretrained SGSMOTE	**0.81**	**0.63**	0.64	0.54	0.78
	Bigo et al. model	-	-	0.29	0.19	**0.56**
Haydn String Quartets (HC)	SGSMOTE	0.65	0.32	0.30	0.33	0.27
(21 pieces)	Pretrained SGSMOTE	**0.69**	**0.44**	**0.41**	**0.41**	0.41

Table 2. Results using half of the dataset for training, half for testing. Bach: fugues no.1-12 were used for training, no.13-24 for testing; Haydn: random 21:21 split. The pretrained network was trained on the other dataset, i.e. *Pretrained SGSMOTE* for Bach Fugues was pre-trained on string quartets, etc. Classification is binary, the presented $F1$ scores are for the positive class, i.e., the cadence (PAC: Perfect Authentic Cadence; rIAC: root position Imperfect AC; HC: Half Cadence).

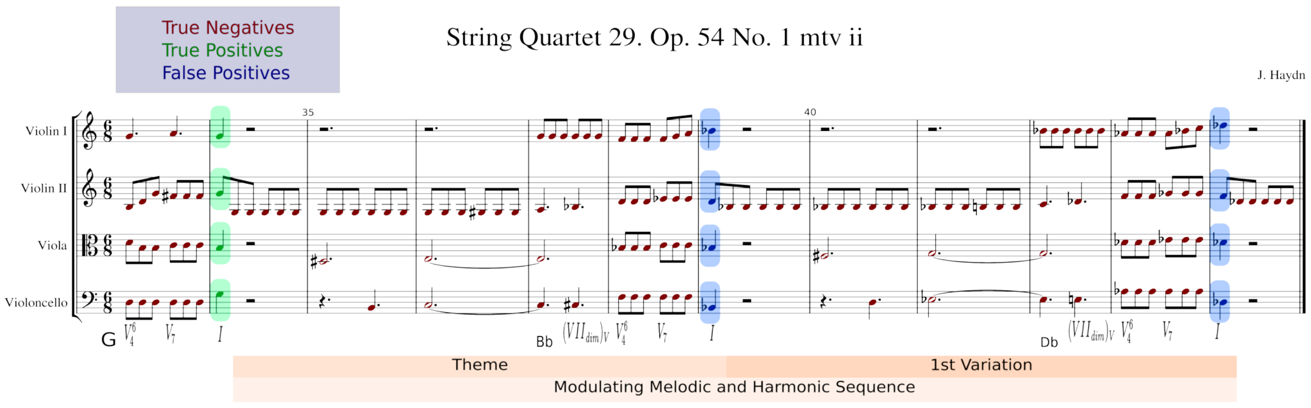

Figure 3. Haydn's String Quartet 29. Op.54 No.1 Mvt. II, mm. 33-45. Showing the output of the Stochastic GraphSMOTE Network for PAC prediction. True negatives are marked with red, true positives with green, false positives with blue. A partial analysis shows the chords towards the end of cadences and highlights a modulating sequence where every sequence ends with a cadential pattern, which counts as false positive predictions by the network.

in terms of F1 score. Only beat-wise scores are given for the reference model (taken from [4]). The last two columns of table 2 give the recall and precision for the beat-wise prediction. All metrics are presented for the positive, i.e. minority/cadence, class.

Our model matches or slightly surpasses the state of the art in rIAC detection in Bach fugues and on HCs in Haydn string quartets but does not reach the reference model's F1 results in PAC detection. We additionally present a pre-trained version of Stochastic GraphSMOTE, where the network was first trained on additional data and fine-tuned for the task. Specifically, the network for PAC prediction in Bach was pre-trained on the string quartets and vice versa. Pre-training, and thus the need for additional data, is the price we pay for the generality of the graph representation and the consequent size (number of parameters) of the deep network. Pre-training helps to (markedly) improve the results on HC, catch up with the reference on PAC in Bach, and narrow the gap on PAC in Haydn.

Generally, our results agree with [4] in implying that half cadences (HC) seem significantly harder to identify than authentic cadences, both perfect and imperfect. Another, more specific, observation concerns different ways in which the compared models achieve their overall F1 scores. In the PAC detection tasks, in particular, we observe comparable or higher recall of our model compared to the reference, but lower precision. This observation motivated us to check some of our model's false positive predictions; Section 6.2 below will show several instructive examples of 'almost correct' identifications.

The *second experiment* we conducted (Table 3) focuses on comparing the relevance of feature groups. For compactness we present here a multi-class classification scenario where we account not only for the existence of a cadence but also for the type of cadence present; that is, we have tree-class problems: no cadence, PAC, or rIAC (Bach) / HC (Haydn, Mozart). We compare two configurations: using all available features (as in the first experiment, feature set *all* in the table), or only feature sets 1 and 2, excluding the cadence-specific features (category 3 in Section 3.1; marked *general* in the table). Given this 3-class setting, we chose to report the macro averaged F1 score over all three classes.

Dataset	Features	F1 Note	F1 Beat
Bach Fugues	general	0.602	0.667
(PAC & rIAC)	all	0.653	0.702
Haydn String Quart.	general	0.542	0.610
(PAC & HC)	all	0.648	0.663
Mozart String Quart.	general	0.584	0.569
(PAC & HC)	all	0.588	0.606

Table 3. Three-class cadence classification with two different feature sets. Results were obtained by 5 fold cross validation (70% of pieces for training, 10% validation, 20% testing); no pre-training. Feature set *all* contains all features from Section 3.1; *general* excludes Category 3 cadence-specific engineered features.

(Macro averaging was chosen to counter the overwhelming effect of the majority class *no cadence*). The results (see Table 3) support the relevance of carefully devised cadence-related features à la [4]. However, also the general-purpose category 1 and 2 features alone support non-trivial cadence recognition and discrimination performance, which implies that the relational graph representation in combination with a convolutional approach manages to enrich highly local features with relevant non-local score context.

To investigate this latter aspect in more detail, we run a *third experiment*, to look at the effect of neighbor convolution depth on the obtainable classification score, again at three prediction granularity levels (note, onset, beat). Convolution depth refers to the number l of hidden layers of the encoder and the subsequent neighbor sampling up to l-hop neighbors. Our results (see Table 4) suggest that neighbor convolution clearly contributes to learning non-local features. Best results are achieved when using a convolution depth of 2. Increasing the receptive field beyond that level, we observed some instabilities emerging in the learning model, which could be attributed to the common vanishing gradient problem in deep GCNs [22].

Depth	F1 Note	F1 Onset	F1 Beat
None	0.833	0.671	0.667
1-hop	0.854	0.707	0.701
2-hop	**0.869**	**0.737**	**0.732**
3-hop	0.836	0.706	0.659

Table 4. Effect of neighbor convolution depth on PAC prediction in Bach fugues. The F1 Note/Onset/Beat scores presented are binary, i.e., for the PAC class. Depth refers to neighbor convolution depth. *None* means no graph convolution.

6.2 A Qualitative Look

Motivated by the fact that our model, while higher on recall, seems to be lower on precision than the model in [4], we take a closer look at some of the false positives in individual examples. Our findings suggest that many false positive predictions resemble cadences, in terms of tonal structure or implications, and could be considered and annotated as such, but lack some main components.

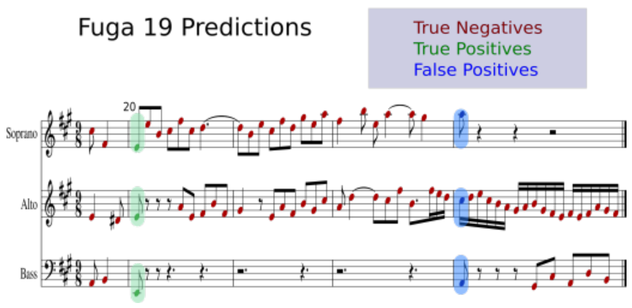

Figure 4. Predictions of Stochastic GraphSMOTE for fugue No.19, J.S.Bach, Well-tempered Clavier.

Figure 4 shows an example. The cadence prediction by our model on the downbeat of bar 23 is a false positive, according to the ground truth annotation. However, one could argue that the passage clearly has a cadence-like role, marking the end of the 2nd fugal episode and the return to the original tonality of A major [23].

Another example is the passage discussed in Fig.4 of [4], where a pattern occurs that has all the technical ingredients of a PAC, but was not annotated as such for (debatable) higher-level musicological considerations. Again, our model's PAC prediction there counts as a false positive.

As a final example, consider mm. 33-45 of Haydn's Op.54 No.1, 2nd mvt (Figure 3). We observe two false positive beat-wise predictions (8 if we count note-wise) in bars 39 and 44, respectively, following a true PAC on the beginning of bar 34. A harmonic analysis of these bars indicates a proper PAC preparation with text-book voice leading on the cadence arrival point in every occasion. These two false positive PACs form part of a modulating melodic and harmonic sequence; whether to classify them as cadences is a matter of higher-level musicological considerations.

We cite these few qualitative examples in an attempt to show that our prediction model can identify many more cadential patterns than the raw experimental figures suggest, but by design cannot consider high-level musical considerations such as, e.g., whether PAC-like patterns that occur in sequence should count as PACs or not.

7. CONCLUSION

We have presented a graph approach to effectively target the cadence detection task on symbolic classical scores. We demonstrated that our Graph Convolutional Network, Stochastic GraphSMOTE, can learn using only local note features, without the need for any musical assumptions about cadence anchor points. Furthermore, our network can produce fine-grained predictions at the level of individual notes.

Future work will address the performance of the model on different tasks, using the same graph representation. We hope to be able to show that this simple but general and natural representation of scores in terms of graphs can support a broad variety of symbolic music analysis and classification tasks.

8. ACKNOWLEDGEMENTS

This work is supported by the European Research Council (ERC) under the EU's Horizon 2020 research & innovation programme, grant agreement No. 101019375 ("Whither Music?"), and the Federal State of Upper Austria (LIT AI Lab). The authors would like to thank Dr. Hamid Eghbal-zadeh for helpful discussions on Graph Neural Networks.

9. REFERENCES

[1] F. Korzeniowski, S. Oramas, and F. Gouyon, "Artist similarity with graph neural networks," in *Proceedings of the 22nd International Society for Music Information Retrieval Conference*, 2021.

[2] C.-Z. A. Huang, C. Hawthorne, A. Roberts, M. Dinculescu, J. Wexler, L. Hong, and J. Howcroft, "The bach doodle: Approachable music composition with machine learning at scale," in *Proceedings of the 18th International Society for Music Information Retrieval Conference*, 2019.

[3] C. Hawthorne, A. Stasyuk, A. Roberts, I. Simon, C.-Z. A. Huang, S. Dieleman, E. Elsen, J. Engel, and D. Eck, "Enabling factorized piano music modeling and generation with the maestro dataset," in *Proceedings of 7th International Conference on Learning Representations*, 2019.

[4] L. Bigo, L. Feisthauer, M. Giraud, and F. Levé, "Relevance of musical features for cadence detection," in *Proceedings of the 19th International Society for Music Information Retrieval Conference (ISMIR 2018)*, 2018.

[5] T. Zhao, X. Zhang, and S. Wang, "Graphsmote: Imbalanced node classification on graphs with graph neural networks," in *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, 2021.

[6] A. Popoff, M. Andreatta, and A. Ehresmann, "Relational poly-klumpenhouwer networks for transformational and voice-leading analysis," *Journal of Mathematics and Music*, vol. 12, no. 1, 2018.

[7] E. Karystinaios, C. Guichaoua, M. Andreatta, L. Bigo, and I. Bloch, "Music genre descriptor for classification based on tonnetz trajectories," in *Proceedings of Journées Informatiques Musicales*, 2021.

[8] C. Shi, Y. Li, J. Zhang, Y. Sun, and S. Y. Philip, "A survey of heterogeneous information network analysis," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 1, pp. 17–37, 2016.

[9] D. Jeong, T. Kwon, Y. Kim, and J. Nam, "Graph neural network for music score data and modeling expressive piano performance," in *International Conference on Machine Learning*, 2019.

[10] D. Jeong, T. Kwon, Y. Kim, K. Lee, and J. Nam, "Virtuosonet: A hierarchical rnn-based system for modeling expressive piano performance." in *Proceedings of the 20th International Society of Music Information Retrieval Conference*, 2019.

[11] M. Giraud, R. Groult, E. Leguy, and F. Levé, "Computational fugue analysis," *Computer Music Journal*, vol. 39, no. 2, 2015.

[12] P. R. Illescas, D. Rizo, and J. M. I. Quereda, "Harmonic, melodic, and functional automatic analysis," in *Proceedings of the International Computer Music Conference*, 2007.

[13] D. R. Sears and G. Widmer, "Beneath (or beyond) the surface: Discovering voice-leading patterns with skip-grams," *Journal of Mathematics and Music*, vol. 15, no. 3, 2021.

[14] C. E. C. Chacón, P. Silvan, E. Karystinaios, F. Foscarin, M. Grachten, and G. Widmer, "Partitura: A python package for symbolic music processing," in *Proceedings of the Music Encoding Conference*, 2022.

[15] C. E. C. Chacón, "Computational modeling of expressive music performance with linear and non-linear basis function models," Ph.D. dissertation, Johannes Kepler University, Austria, 2018.

[16] M. Schuijer, *Analyzing atonal music: Pitch-class set theory and its contexts.* University Rochester Press, 2008.

[17] V. P. Dwivedi, C. K. Joshi, T. Laurent, Y. Bengio, and X. Bresson, "Benchmarking graph neural networks," *arXiv preprint arXiv:2003.00982*, 2020.

[18] P. Allegraud, L. Bigo, L. Feisthauer, M. Giraud, R. Groult, E. Leguy, and F. Levé, "Learning sonata form structure on mozart's string quartets," *Transactions of the International Society for Music Information Retrieval (TISMIR)*, vol. 2, no. 1, 2019.

[19] D. R. Sears, M. T. Pearce, W. E. Caplin, and S. McAdams, "Simulating melodic and harmonic expectations for tonal cadences using probabilistic models," *Journal of New Music Research*, vol. 47, no. 1, pp. 29–52, 2018.

[20] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in neural information processing systems*, vol. 30, 2017.

[21] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[22] G. Li, M. Muller, A. Thabet, and B. Ghanem, "Deepgcns: Can gcns go as deep as cnns?" in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019.

[23] "Bach: Prelude and fugue no.19 in a major, bwv 864 analysis," May 2018. [Online]. Available: https://tonic-chord.com/bach-prelude-and-fugue-no-19-in-a-major-bwv-864-analysis/

DOMAIN ADVERSARIAL TRAINING ON CONDITIONAL VARIATIONAL AUTO-ENCODER FOR CONTROLLABLE MUSIC GENERATION

Jingwei Zhao[2,4] **Gus Xia**[3,5] **Ye Wang**[1,2,4]
[1] School of Computing, NUS [2] Institute of Data Science, NUS [3] Music X Lab, NYU Shanghai
[4] Integrative Sciences and Engineering Programme, NUS Graduate School [5] MBZUAI
jzhao@u.nus.edu, gxia@nyu.edu, wangye@comp.nus.edu.sg

ABSTRACT

The variational auto-encoder has become a leading framework for symbolic music generation, and a popular research direction is to study how to effectively *control* the generation process. A straightforward way is to control a model using different conditions during inference. However, in music practice, conditions are usually sequential (rather than simple categorical labels), involving rich information that overlaps with the learned representation. Consequently, the decoder gets confused about whether to "listen to" the latent representation or the condition, and sometimes just ignores the condition. To solve this problem, we leverage *domain adversarial training* to *disentangle* the representation from condition cues for better control. Specifically, we propose a condition corruption objective that uses the representation to denoise a corrupted condition. Minimized by a discriminator and maximized by the VAE encoder, this objective adversarially induces a condition-invariant representation. In this paper, we focus on the task of melody harmonization[1] to illustrate our idea, while our methodology can be generalized to other controllable generative tasks. Demos and experiments show that our methodology facilitates not only condition-invariant representation learning but also higher-quality controllability compared to baselines.

1. INTRODUCTION

In deep music generation, improving *controllability* has been a major challenge that gains increasing research attention [1–6]. In practice, controllability is typically implemented under a conditional architecture, where the generation process is biased by external condition inputs. For example, EC^2-VAE [7] learns a representation z_x of 8-beat melody x while the underlying chords are given as condition c. The system is controllable if the generated melody can adapt to variable chords properly. For such representation-learning architectures, however, the decoder tends to find a shortcut from z_x to x without attending to c, leading to "condition collapse". The reason for this, as we argue, is that z_x is inevitably intertwined with condition c in the representation space, as c is often an innate property of x. In the case of EC^2-VAE, the condition of chords is very much implied by the melody.

To address this problem, the representation z_x must be disentangled from condition c. A popular way to achieve this goal is to use an adversarial objective that predicts c from z_x, as shown in Figure 1. On the one hand, this objective is optimized by a discriminator; on the other hand, the encoder is trained to "fool" the discriminator by detaching c-related cues out of z_x. In this way, the decoder cannot find a shortcut in z_x but is forced to seek c to reconstruct x. Such a technique stems from *domain adversarial training* (DAT) [8], where the "domain" is interpreted as "condition" that controls the generation.

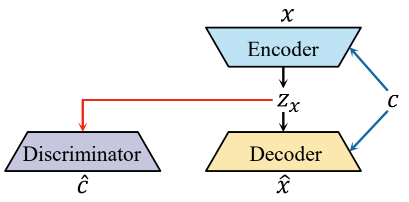

Figure 1: An illustration of domain adversarial training over a conditional generation architecture.

Apparently, DAT can be a powerful tool for controllable music generation. Previous studies [9, 10] have discussed simple scenarios where the condition is a global label (*e.g.*, note density). In music practice, however, local and sequential conditions [11] are more common. In such cases, c may not be fully implied by x, so the objective that simply predicts c from z_x does not necessarily hold.

In this paper, we focus on sequential conditions and develop a generalized form of DAT for controllable music generation. We illustrate our methodology with the task of *chord representation learning conditioned on melody*, where x stands for the chord progression, and c is the melody condition. In general, a chord progression can match many melodies, so we cannot directly predict c (melody) from z_x (chord) for the DAT objective. Instead, we leverage z_x to reconstruct c from a corrupted condition c^*. We rely on c^* to provide the melody context that cannot be hinted by chord x; on the other hand, the corrupted

[1] Demos and codes via https://zhaojw1998.github.io/DAT_CVAE.

information reveals c's harmonic dependency on x, which we enforce the discriminator to learn. With proper corruption design, our DAT objective can be generalized to more scenarios with sequential conditions.

A well-trained model with good controllability can help us harmonize a new melody using the representation (style) of an existing chord progression. Experiments show that our model performs an excellent disentanglement of data representation from the condition, and the controllability outperforms the baselines. In summary, our contributions in this paper are as follows:

- **A general approach to controllability**: Based on a novel adversarial objective with condition corruption, we generalize domain adversarial training to music generation with sequential conditions;
- **A novel harmonization methodology**: We present a representation learning-based method for melody harmonization. Our current model harmonizes pop and folk melodies with the triad and seventh chords.

2. RELATED WORKS

2.1 Domain Adversarial Training

Domain adversarial training (DAT) is a representation learning approach initially proposed for domain adaptation tasks [12–14]. Through an adversarial process as described in Section 1, DAT enforces *domain invariance* to data representation so that it can be adapted to different domains flexibly. Such adaptability to new domains is analogous to controllability with new conditions. For generation tasks, DAT has been utilized to learn a condition-invariant data representation. Such invariance enforces the decoder to use condition information for reconstruction [15]. During inference, the decoder "listens to" new conditions as well and generates new data in a controllable way.

The first attempts that incorporate DAT with generation dealt with facial image generation conditioned on binary attributes (*e.g.*, male or female) [15, 16]. Such conditions cannot be explicitly supervised because we cannot find any pair of images that represents the same person both male and female. Fortunately, DAT enforces attribute invariance at encoding and learns attribute dependency at decoding, thus circumventing this problem. Recently, DAT has been extended to symbolic music generation conditioned on various attributes. Kawai *et al.* adopts DAT to a variational auto-encoder (VAE) for melody generation conditioned on statistical attributes (*e.g.*, note density) [9]. Later, Matsuoka *et al.* generalizes this methodology to generating polyphonic music with similar conditions [10].

For previous works, the conditions are particularly a global statistical label, which only represents a limited scenario of controllable generation. In our paper, we generalize the usage of DAT to sequential conditions. Conditioned on an 8-bar melody, we aim to learn a pitch-invariant representation of an 8-bar chord progression, which can later be adapted to varied melody conditions and to harmonize them. Our main novelty lies in a special design of the adversarial objective, which is to denoise corruption rather than make full prediction. This technique greatly helps us in dealing with the nuance of sequential conditions.

2.2 Controllable Music Generation

Controllable music generation takes various forms in terms of controlling technique and music representation [17]. For controlling technique, controllability can be achieved by sampling, interpolation, conditioning, and more ways [11]. For music representation, controls can be performed over statistical music properties (pitch variability, note density, etc.) [9, 10], compositional factors (chord progression, texture and rhythmic patterns, etc.) [7, 18–20], high-level semantics (emotion, cultural style, etc.) [21], and so on. With the development of representation learning, such properties can be abstracted and disentangled for flexible control.

In this paper, we are interested in chord representation learning conditioned on melodies, which falls into the category of controlling compositional factors via conditioning. Various conditional architectures, such as conditional VAE (C-VAE) [22], have been applied for similar purposes [7, 18–21]. However, as the condition is often easily implied by the representation, the decoder tends to skip the condition, and simply reconstruct the data for whatever conditions. To eradicate this problem, we introduce domain adversarial training and generalize it to sequential conditions (in our case, an 8-bar lead melody). Our model learns a pitch-invariant chord representation so that we can generate chord progressions harmoniously conditioned on varied melodies. Such control over compositional factors is common to broader music generation scenarios, and our methodology is generally applicable as well.

3. METHODOLOGY

In this section, we introduce our methodology with domain adversarial training on learning chord representation conditioned on the melody. An overview of our model is illustrated in Figure 2. We first describe our data representation and structure in Section 3.1. Then, we introduce our proposed model in Section 3.2. Finally, we elaborate on our novel design of condition corruption in Section 3.3.

3.1 Date Representation and Structure

3.1.1 Chord Representation

Our model generates an 8-bar chord progression conditioned on the melody. We quantize the chord progression at 1-beat unit and derive $T = 32$ timesteps. The maximum note count P for each chord is 4, which means we can flexibly represent any type of triad and seventh chords. Specifically, we treat chord progression as a piece of polyphony and follow [18] to represent it in both a surface structure (as model input) and a deep structure (for encoding).

The surface structure is a nested array of pitch attributes, denoted by $\{x_p^t | 1 \leq t \leq T, 1 \leq p \leq P\}$. Concretely, x_p^t is the p^{th} lowest pitch onset at time step t. We