and G denotes the generator that converts a random vector $\mathbf{z}$ into a fake sample. Each projector P_l consists of three components: a pre-trained feature network C_l, the CCM modules, and the CSM modules. First, we extract features from L different layers of the feature network C_l, leading to L set of feature maps in different scales. Second, for each scale, the CCM mixes the features across channels by *random* (i.e., not-learned) 1×1 convolutions with an equal number of input and output channels, which can be viewed as the generalization of random permutation. Third, CSM further mixes features across scales by *random* 3×3 convolutions and bilinear upsampling layers, yielding a U-Net architecture that combines the feature maps of different scales. The features fused by CCM and CSM in L different scales would then be fed to each discriminator D_l. The random projections introduced by CCM and CSM have the effect of encouraging each D_l to take into account the entire feature space instead of overfitting to a sub-set of feature space, thereby avoiding mode collapse. Without using gradient penalties and any sophisticated training strategies, Projected GAN updates its loss simply by summing the output of the L discriminators.

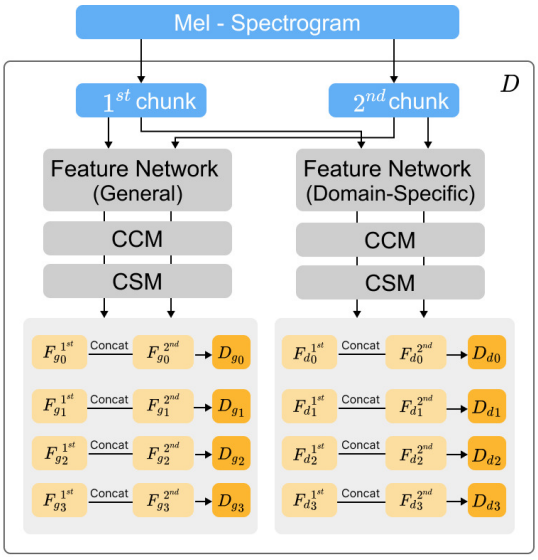

Figure 2. Diagram of the proposed discriminator architecture. A Mel-spectrogram is divided into two chunks and then fed to either a general or a domain-specific pre-trained feature network, or both (for "fusion"). Features computed for both chunks are aggregated at each scale l before feeding to the corresponding discriminator D_l.

3. DATSETS & CLASSIFIERS

Our work is built upon the use of the following datasets.

Youtube-100M [39] is a private dataset of Google, containing 100 million Youtube videos. Each video has on average 5 manually assigned tags, out of 30,871 possible labels. Hershey *et al.* [39] train a VGGish pre-trained network for large-scale audio classification using the dataset. While we are not able to have a copy of the dataset, we can use the pre-trained weights Gemmeke *et al.* [53] share publicly as our general audio classifier.

MagnaTagATune (MTAT) [41] is a dataset commonly-used in research on music auto-tagging. It contains 25,863 music clips, each 29-seconds long. We use MTAT to train one of our domain-specific pre-trained feature networks. We follow the original data split [41] and use only the top 50 tags, including genre and instrumentation labels, as well as decades (e.g., '80s' and '90s') and moods.

Looperman dataset is an in-house collection of loops from `https://www.looperman.com/`, a website hosting free music loops. We get the audio and uploader-provided metadata of 23,983 drum loops and 22,625 synth loops. According to the metadata, the drum loops and synth loops can be categorized to 66 and 58 genres, respectively. We use the Looperman dataset for not only building a domain-specific pre-trained feature network but also for building our audio-domain loop generation model.

Following the preprocessing steps of Hung *et al.* [27], we first apply downbeat tracking via `madmom` [54] to split every loop into single bars and then time-stretch each to 120 BPM (beats-per-minute) by `pyrubberband` [55] to unify their length to be always **2 seconds per loop**. After this preprocessing, we have 128,122 and 42,570 one-bar loops for drum and synth, respectively. We split the data by 80/10/10 for training the loop genre classifier, and use the entire data for music loop generation.

3.1 Pre-trained Feature Networks for Music

General. As aforementioned, we use the VGGish network trained on Youtube-100M [53] as the general pre-trained feature network. Taking the Mel-spectrogram as input, the model uses 2D convolutional blocks and 2D max-pooling layers to compute 128-dimensional features at the output.

Domain specific. Short-Chuck CNN (SCNN) [40] has a simple 2D CNN architecture with 3×3 filters and residual module but it has been shown to be remarkably effective for music auto-tagging. We use in our implementation 6 layers of CNN blocks with a fully connected layer and the residual module. Each CNN block comprises a 2×2 max-pooling layer. We use SCNN to train separate classifiers for MTAT and Looperman from scratch, and then used the trained classifiers as our domain-specific pre-trained feature networks. For MTAT, SCNN achieves 0.909 ROC-AUC and 0.445 PR-AUC. For genre classification of the looperman drum loops, the classification accuracy reaches 0.792. For synth loops, the accuracy attains 0.700.

Fusion. Fusing multiple pre-trained feature networks has also been shown useful for Projected GAN-based image generation [7]. Accordingly, we also experiment with a simple fusion strategy that allows the discriminator to consider both general and domain-specific features at the same time, as depicted in Figure 2.

4. AUDIO GENERATION BY PROJECTED GAN

Following Hung *et al.* [27], we use StyleGAN2 [5] as the backbone of our loop generation model. However, we improve their model in a number of aspects, regarding to not

only the discriminator (i.e., with Projected GAN) but also the generator. We provide the details below.

Improving the generator. The generator G consists of a mapping network G_m and a synthesis network G_s. We implement G_m with only 6 fully-connected layers instead of the 8 in the original StyleGAN2 architecture. Furthermore, echoing the finding in the image domain [56], we find that the length of the vectors $\mathbf{z}$ (i.e., the input of G_m) largely affects the model performance. Setting the length of $\mathbf{z}$ to 512 as done in [27] leads to mode collapse in our preliminary experiments, as shown in Figure 3(a). This may be related to the so-called "intrinsic dimension" of the data [56]; an overly large latent space of $\mathbf{z}$ introduces redundancy that distracts the generator. As smaller $\mathbf{z}$ empirically works better, we set its length to 32 in the experiments reported below. We similarly set the length of the "style code" $\mathbf{w}$ (i.e., the output of G_m) to a small value of 64.

Pipeline The training follows the procedure of the Projected GAN, but we use the following pipeline to match the input shape expected by the pre-trained feature networks. First, we compute the Mel-spectrogram with 512-point window size and 160-point hop size for short-time Fourier Transform (STFT) and 64 Mel channels. Second, we feed a latent $\mathbf{z}$ to the mapping network G_m to generate style codes that modulate the convolutions of the synthesis network G_s, using four upsampling blocks to generate a Mel-spectrogram that corresponds to a synthesized *2-second* loop. We note that our pre-trained feature networks are on *1-second* audio. To address the size mismatch, we split the Mel-spectrogram into **two chunks**, with the first half corresponding to the first second and the other the rest. As illustrated in Figure 2, we feed the two chunks separately to the discriminators, going through the pre-trained feature network, CCM and CSM. Moreover, we concatenate the features with the same scales but in different chunks to aggregate the features from individual chunks, yielding $L = 4$ aggregated features $\{F_l, l = 1...L\}$. We feed each of these features independently to the corresponding discriminators D_l. Additionally, if we consider two pre-trained feature networks simultaneously, referred to as "fusion" in Section 3.1, we have in total $2L$ aggregated features and $2L$ discriminators. Eventually, we sum the loss from these discriminators to update the network.

Similar to Hung *et al.* [27], at inference time, the Mel-spectrogram generated by the generator would go through a MelGAN vocoder [42] to become waveforms.

5. OBJECTIVE EVALUATION

We use the following objective metrics to evaluate the quality and diversity of the generated loops.

Inception Score (IS) [58, 59] measures the quality of the generated loops and detects whether there is a mode collapse by using a pre-trained domain-specific classifier, namely the loop genre classifier for each type of loops (i.e., drum or synth). It penalizes models whose samples cannot be reliably classified into a single class or that only belong to a few from all possible classes.

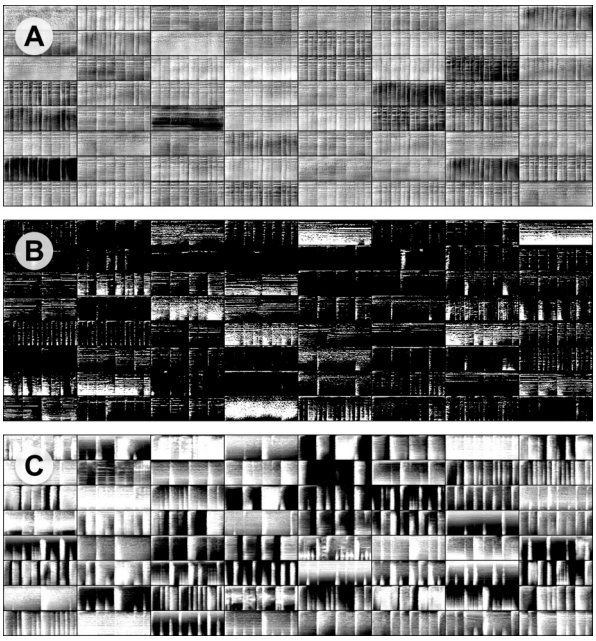

Figure 3. Mel-spectrograms of examples of drum loops generated (a) by a failed case using an overly large latent space for $\mathbf{z}$, leading to mode collapse (see Section 4); (b) Projected GAN with the MTAT domain-specific classifier; (c) Projected GAN with the VGG general classifier.

Fréchet Audio Distance (FAD) [60] reflects both quality and diversity as it measures the distance between continuous multivariate Gaussians fitted to the embeddings of the real and generated loops.

Density & Coverage (D&C) [57] are new metrics that measure respectively the quality and diversity of the generated data. D&C are considered to be more robust against the influence of outliers compared to older metrics such as precision and recall (P&R) [61]. 'D' is the average number of real-sample neighborhood spheres that contain each of the fake samples. It may be greater than 1 depending on the density of reals around the fakes. 'C' is the fraction of real samples whose neighborhoods contain at least one fake sample. In our implementation, we use the hyperparameters suggested by [57]. Moreover, following [57], we calculate D&C with a randomly-initialized VGG16 model that projects samples to VGG16 *fc2* space whose dimension is deliberately set to a small value 64.

We evaluate the following models here:

- **StyleGAN2**: the SOTA for drum loop generation [27].
- **StyleGAN2 (early-stop)**: the StyleGAN2 that stops training at the same point when Projected GAN converges, providing a reference demonstrating the possible advantage of the training efficiency of Projected GAN.
- **Projected StyleGAN2** with different pre-trained feature networks, including 1) the general VGG classifier alone, 2) the domain-specific SCNN classifier trained on MTAT alone (denoted as $\text{SCNN}_{\text{MTAT}}$), 3) the SCNN classifier trained on Looperman alone, using the drum or synth classifier depending on whether it is for generating drum or synth loops (denoted as $\text{SCNN}_{\text{Loop}}$), 4) "fusion" of

Models	Drum loops				Synth loops			
	IS ↑	FAD ↓	D ↑	C ↑	IS ↑	FAD ↓	D ↑	C ↑
A Real data	16.30	0.01	1.00	1.00	14.95	0.01	1.00	1.00
B StyleGAN2 [27]	5.58	2.50	1.01	**0.89**	4.80	3.60	1.19	**0.91**
C StyleGAN2 (early-stop)	5.21	3.40	0.56	0.43	4.55	7.97	1.01	0.75
D Projected StyleGAN2 (VGG)	5.87	3.03	1.06	0.70	4.70	**2.34**	1.31	0.82
E Projected StyleGAN2 ($SCNN_{MTAT}$)	4.75	8.18	0.00	0.00	3.97	7.14	0.001	0.002
F Projected StyleGAN2 ($SCNN_{Loop}$)	4.45	7.21	0.00	0.00	3.08	8.32	0.001	0.002
G Projected StyleGAN2 (VGG+$SCNN_{MTAT}$)	6.22	2.45	**1.11**	0.74	4.73	3.13	1.67	0.85
H Projected StyleGAN2 (VGG+$SCNN_{Loop}$)	**6.31**	**2.34**	1.08	0.73	**4.82**	2.56	**1.70**	0.85

Table 1. Objective evaluation result for the different settings of the Projected GANs trained on the Looperman dataset for loop generation. 'D' and 'C' stand for Density & Coverage [57] (↓/↑: the lower/higher the better; the best in bold).

VGG, $SCNN_{MTAT}$, and 5) "fusion" of VGG, $SCNN_{Loop}$.

Table 1 presents the objective evaluation results of models trained on drum loops and synth loops. Each model generates 10,000 random loops to compute the scores. We also compute the scores using the real data for setting a high anchor of the objective scores.

We see that the best-performing configurations of Projected StyleGAN2 can achieve higher IS and lower FAD than StyleGAN2 (i.e., row B). For drum loops, the IS can be improved from 5.58 to 6.31 (row H) ; for synth loop the FAD can be reduced from 3.60 to 2.34 (row D).

Interestingly and somehow surprisingly, when only a single pre-trained feature network is used (i.e., rows D–F), we find that *only* Projected StyleGAN2 (VGG) performs nicely. The domain-specific pre-trained feature networks actually degrade the performance of loop generation; either Projected StyleGAN2 ($SCNN_{MTAT}$) or Projected StyleGAN2 ($SCNN_{Loop}$) obtains lower IS and higher FAD, and even close-to-zero D&C. This suggests that a general pre-trained feature network works much better than a domain-specific pre-trained feature network in the context of Projected GAN-based unconditional loop generation. This is likely because the discriminator employing a domain-specific pre-trained feature network is too strong compared to the generator, leading to gradient vanishing. Figure 3(b) shows samples generated by Projected StyleGAN2 ($SCNN_{MTAT}$); we see that the model weirdly generates sparse samples most of the time.

Table 1 also shows that the "fusion" of general and domain-specific pre-trained feature networks (rows G & H) performs slightly better in some metrics than using a general pre-trained feature network alone (row D). In particular, we see that Projected StyleGAN2 (VGG+$SCNN_{Loop}$) achieves the highest IS score in both drum and synth loop generation. It also reaches the lowest FAD for drum loops and the second lowest FAD for synth loops. We show examples of its generated synth and drum loops in Figures 1 and 3(c), respectively.

Projected StyleGAN2 appears to have lower D&C compared to StyleGAN2, suggesting that Projected StyleGAN2 sacrifices a little diversity for higher quality of the generated samples. This is presumably because the feature space has been constrained by the pre-trained feature networks throughout the whole training process, making it

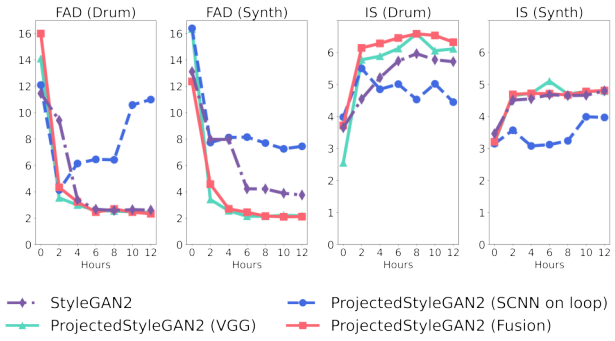

Figure 4. The FAD and IS as a function of training hours.

hard to capture the whole distributions. Future work can be done to remedy this and further improve the diversity of Projected StyleGAN2.

Figure 4 shows how the FAD and IS of four selected models (rows B, D, F, H in Table 1) varies as a function of training time, when all the models are trained separately and independently on a single V100 GPU. For both drum and synth loop generation, Projected GAN leads to lower FAD much faster than the StyleGAN2 baseline. For synth loops, Projected GAN with only 2-hour training reaches lower FAD than the StyleGAN2 baseline with 12-hour training. In general, Projected GAN converges at approximately 2 hours for both drum and synth loops, while StyleGAN2 converges x5 times longer at about 10 hours. Echoing the result in Table 1, using the epoch of StyleGAN2 corresponding to 2-hour training time (i.e., row C) obtains worse scores than the models corresponding to rows B and D in almost all the metrics for both drum and synth loops. Overall, these results nicely demonstrate how Projected GAN speeds up and improves GAN training.

6. SUBJECTIVE EVALUATION

To further assess the loops generated by the models, we conduct a subjective listening test through an anonymous online questionnaire. Each subject is presented with a randomly picked human-made loop from the Looperman dataset (i.e., 'Real') and a randomly-generated loop by each of the following four models: StyleGAN2, StyleGAN2 (early-stop), Projected StyleGAN2 ($SCNN_{Loop}$)

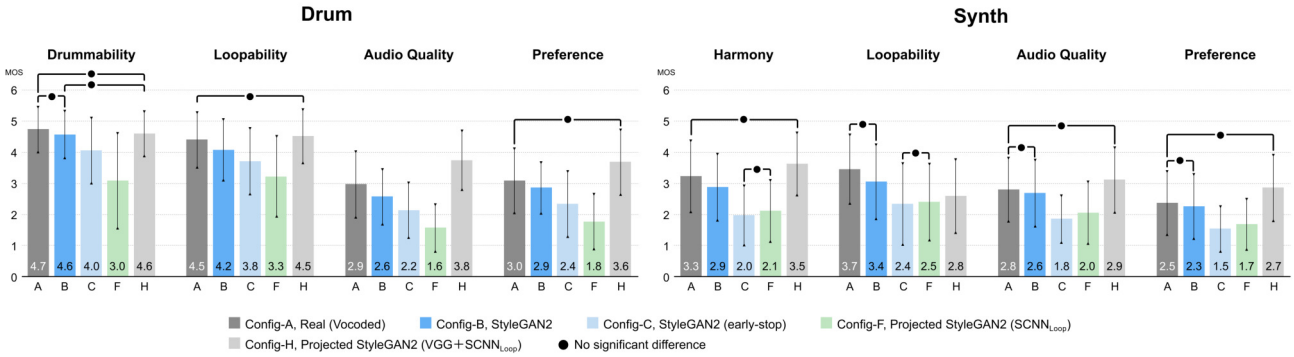

Figure 5. Subjective evaluation results. The performance difference between any pair of models in any metric is statistically significant (p-value< 0.001) under the Wilcoxon signed-rank test, except for the pairs that are explicitly highlighted.

and Projected StyleGAN2 (VGG+SCNN$_{\text{Loop}}$). The subject is then asked to rate each of these loops in terms of the following metrics, all on a five-point Likert scale.

- **Drummability or Harmony**: Drummability (for drum loops) means whether the sample contains percussion sounds, and Harmony (for synth loops) means whether the sample contains harmonic sounds.
- **Loopability**: whether the sample can be played repeatedly in a seamless manner.
- **Audio quality**: whether the sample is free of unpleasant noises or artifacts.
- **Preference**: how much you like it.

To evaluate loopability, we repeat each sample four times in the audio recording. Because the output of the models all goes through the MelGAN vocoder [42] to become waveforms, we do the same for 'Real' to be fair.

Figure 5 shows the averaged results from 35 participants. The responses indicate an acceptable level of reliability, Cronbach's $\alpha = 0.721$. The subjective evaluation result aligns nicely with the objective evaluation result. Projected StyleGAN2 (VGG+SCNN$_{\text{Loop}}$) performs the best and Projected StyleGAN2 (SCNN$_{\text{Loop}}$) performs the worst. Overall, Projected StyleGAN2 (VGG+SCNN$_{\text{Loop}}$) achieves results comparable to StyleGAN2 and even 'Real' for both types of loops. Somehow surprisingly, Projected StyleGAN2 (VGG+SCNN$_{\text{Loop}}$) can even outperform 'Real' in audio quality and preference for both drum and synth loops. We conjecture that this is because the 'Real' here actually stands for the "MelGAN-vocoded" version of the real data, whose quality may have suffered from the artefacts introduced by the vocoder.

Almost all models lead to higher subjective scores for drum loops than for synth loops, suggesting that synth loop generation is more challenging. We note that, for synth loops, Projected StyleGAN2 (VGG+SCNN$_{\text{Loop}}$) actually obtains higher harmony scores than StyleGAN2 and 'Real', but its loopability scores is lower. To further improve its performance for synth loop generation, future work may be done to explicitly consider loopability in model training.

7. LIMITATIONS

We only focus on one-bar loop generation with a specific BPM of 120 in our study, which is "convenient" as it gives us fixed-size Mel-spectrograms to be treated as images by StyleGAN2. However, to be applicable to loop-based music production, future work needs to consider variable BPMs and loops with more bars. As the size of the Mel-spectrogram would not be fixed when we consider variable BPMs, future work may need to adopt generative models other than StyleGAN2 as the backbone. Possible candidates are UNAGAN [23] and VQGAN [62], both of which may also benefit from the ideas of Projected GAN.

To apply Projected GAN, we need to take care of the *shape matching* between the pre-trained networks and the generative model. For example, in our work the pre-trained networks (e.g., the VGGish one) is trained on 1-second audio chunks, but our generator is to generate 2-second loops. The size mismatch can be easily addressed by simply splitting the Mel-spectrogram into two chunks in our work, but this is trickier if we consider BPMs other than 120.

8. CONCLUSION

In this paper, we have demonstrated that Projected GAN can be used to improve the training efficiency and overall performance of GAN-based models for audio generation, specifically the generation of drum loops and synth loops. Moreover, we demonstrated that using a domain-specific pre-trained feature network alone does not work well; we need to use either a general pre-trained feature network, or the fusion of multiple pre-trained feature networks.

This work can be extended in many ways. First, we can expand our work to generate variable-length loops, or to other GAN-based audio-related tasks. Next, we can explore unsupervised or self-supervised approaches (e.g., [63, 64]) to get the pre-trained feature network. Third, we are also interested in using class conditions or attributes for more controllable loop generation. Doing so may help improve the diversity of the generated loops as well, according to related work on image generation [7]. Finally, using diffusion probabilistic models [65, 66] as the generative model for loop generation also worth trying.

9. ACKNOWLEDGEMENT

We are grateful to Tun-Min Hung for helpful discussions during the project. We also thank the anonymous reviewers for their constructive feedbacks. Our research is funded by grant NSTC 109-2628-E-001-002-MY2 from the National Science and Technology Council of Taiwan.

10. REFERENCES

[1] I. Goodfellow *et al.*, "Generative adversarial nets," in *Proc. Advances in neural information processing systems*, 2014, pp. 2672–2680.

[2] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *arXiv:1411.1784*, 2014.

[3] A. Brock, J. Donahue, and K. Simonyan, "Large scale GAN training for high fidelity natural image synthesis," *arXiv preprint arXiv:1809.11096*, 2018.

[4] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," in *Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition*, 2019, pp. 4396–4405.

[5] T. Karras *et al.*, "Analyzing and improving the image quality of StyleGAN," in *Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition*, 2020.

[6] T. Karras, M. Aittala, S. Laine, E. Härkönen, J. Hellsten, J. Lehtinen, and T. Aila, "Alias-free generative adversarial networks," in *Proc. Conf. Neural Information Processing Systems*, 2021.

[7] A. Sauer, K. Schwarz, and A. Geiger, "StyleGAN-XL: Scaling StyleGAN to large diverse datasets," in *Proc. ACM SIGGRAPH Conf.*, 2022.

[8] O. Mogren, "C-RNN-GAN: Continuous recurrent neural networks with adversarial training," in *Proc. Constructive Machine Learning Workshop*, 2016.

[9] L.-C. Yang, S.-Y. Chou, and Y.-H. Yang, "MidiNet: A convolutional generative adversarial network for symbolic-domain music generation," in *Proc. Int. Soc. Music Information Retrieval Conf.*, 2017.

[10] H.-W. Dong *et al.*, "MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment," in *Proc. AAAI Conference on Artificial Intelligence*, 2018.

[11] G. Chen *et al.*, "Musicality-novelty generative adversarial nets for algorithmic composition," in *Proc. ACM Int. Conf. Multimedia*, 2018, p. 1607–1615.

[12] N. Trieu and R. M. Keller, "JazzGAN: Improvising with generative adversarial networks," in *Proc. Int. Workshop on Musical Metacreation*, 2018.

[13] I.-C. Wei, C.-W. Wu, and L. Su, "Generating structured drum pattern using variational autoencoder and self-similarity matrix," in *Proc. Int. Soc. Music Information Retrieval Conf.*, 2019.

[14] H. Jhamtani and T. Berg-Kirkpatrick, "Modeling self-repetition in music generation using generative adversarial networks," in *Proc. Machine Learning for Music Discovery Workshop*, 2019.

[15] N. Zhang, "Learning adversarial Transformer for symbolic music generation," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–10, 2020.

[16] Y. Yu, A. Srivastava, and S. Canales, "Conditional LSTM-GAN for melody generation from lyrics," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 17, no. 1, 2021.

[17] A. Muhamed, L. Li, X. Shi, S. Yaddanapudi, W. Chi, D. Jackson, R. Suresh, Z. C. Lipton, and A. J. Smola, "Symbolic music generation with Transformer-GANs," in *Proc. AAAI Conference on Artificial Intelligence*, 2021.

[18] J. Engel, K. K. Agrawal, S. Chen, I. Gulrajani, C. Donahue, and A. Roberts, "GANSynth: Adversarial neural audio synthesis," in *Proc. Int. Conf. Learning Representations*, 2019.

[19] C. Donahue, J. McAuley, and M. Puckette, "Adversarial audio synthesis," in *Proc. Int. Conf. Learning Representations*, 2019.

[20] P. Chandna, M. Blaauw, J. Bonada, and E. Gómez, "WGANSing: A multi-voice singing voice synthesizer based on the Wasserstein-GAN," in *Proc. European Signal Processing Conf.*, 2019, pp. 1–5.

[21] S. Lattner and M. Grachten, "High-level control of drum track generation using learned patterns of rhythmic interaction," in *Proc. IEEE Work. Applications of Signal Processing to Audio and Acoustics*, 2019.

[22] J. Drysdale, M. Tomczak, and J. Hockman, "Adversarial synthesis of drum sounds," in *Proc. Int. Conf. Digital Audio Effects*, 2020.

[23] J.-Y. Liu, Y.-H. Chen, Y.-C. Yeh, and Y.-H. Yang, "Unconditional audio generation with generative adversarial networks and cycle regularization," in *Proc. INTERSPEECH*, 2020.

[24] J. Nistal, S. Lattner, and G. Richard, "DrumGAN: Synthesis of drum sounds with timbral feature conditioning using generative adversarial networks," in *Proc. Int. Soc. Music Information Retrieval Conf.*, 2020.

[25] J. Nistal, C. Aouameur, S. Lattner, and G. Richard, "VQCPC-GAN: Variable-length adversarial audio synthesis using vector-quantized contrastive predictive coding," in *Proc. IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, 2021.

[26] J. Nistal, S. Lattner, and G. Richard, "DarkGAN: Exploiting knowledge distillation for comprehensible audio synthesis with GANs," in *Proc. Int. Soc. Music Information Retrieval Conf.*, 2021, pp. 484–492.

[27] T. Hung, B. Chen, Y. Yeh, and Y. Yang, "A benchmarking initiative for audio-domain music generation using the Freesound Loop Dataset," in *Proc. Int. Soc. Music Information Retrieval Conf.*, 2021.

[28] B.-Y. Chen, W.-H. Hsu, W.-H. Liao, M. A. M. Ramírez, Y. Mitsufuji, and Y.-H. Yang, "Automatic DJ transitions with differentiable audio effects and generative adversarial networks," in *Proc. IEEE Int. Conf. Acoustics, Speech and Signal Proc.*, 2022.

[29] T. Salimans *et al.*, "Improved techniques for training GANs," in *Proc. Advances in Neural Information Processing Systems*, 2016, pp. 2234–2242.

[30] M. Arjovsky and L. Bottou, "Towards principled methods for training generative adversarial networks," in *Proc. Int. Conf. Learning Representations*, 2017.

[31] N. Kodali *et al.*, "On convergence and stability of GANs," *arXiv preprint arXiv:1705.07215*, 2017.

[32] I. Gulrajani *et al.*, "Improved training of Wasserstein GANs," in *Proc. Advances in Neural Information Processing Systems*, 2017, pp. 5769–5779.

[33] L. Mescheder, A. Geiger, and S. Nowozin, "Which training methods for GANs do actually converge?" in *Proc. Int. Conf. Machine Learning*, 2018.

[34] M. Zhao, Y. Cong, and L. Carin, "On leveraging pretrained GANs for generation with limited data," in *Proc. Int. Conf. Machine Learning*, 2020.

[35] T. Grigoryev, A. Voynov, and Babenko, "When, why, and which pretrained GANs are useful?" in *Proc. Conf. Neural Information Processing Systems*, 2022.

[36] N. Kumari, Z. R., E. Shechtman, and J. Y. Zhu, "Ensembling off-the-shelf models for GAN training," in *Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition*, 2022.

[37] A. Sauer, K. Chitta, J. Müller, and A. Geiger, "Projected GANs converge faster," in *Proc. Advances in Neural Information Processing Systems*, 2021.

[38] K. Choi, G. Fazekas, M. Sandler, and K. Cho, "Transfer learning for music classification and regression tasks," in *Proc. Int. Soc. Music Information Retrieval Conf.*, 2017.

[39] S. Hershey *et al.*, "CNN architectures for large-scale audio classification," in *Proc. Int. Conf. Acoustics, Speech and Signal Processing*, 2017.

[40] M. Won, A. Ferraro, D. Bogdanov, and X. Serra, "Evaluation of CNN-based automatic music tagging models," in *Proc. Sound and Music Computing Conf.*, 2020.

[41] E. Law, K. West, M. Mandel, M. Bay, and J. Downie, "Evaluation of algorithms using games: The case of music tagging," in *Proc. Int. Soc. Music Information Retrieval Conf.*, 2009, pp. 387–392.

[42] K. Kumar *et al.*, "MelGAN: Generative adversarial networks for conditional waveform synthesis," *arXiv preprint arXiv:1910.06711*, 2019.

[43] R. Yamamoto, E. Song, and J.-M. Kim, "Parallel WaveGAN: A fast waveform generation model based on generative adversarial networks with multi-resolution spectrogram," in *IEEE Int. Conf. Acoustics, Speech and Signal Proc.*, 2020, pp. 6199–6203.

[44] M. Bińkowski *et al.*, "High fidelity speech synthesis with adversarial networks," *arXiv preprint arXiv:1909.11646*, 2019.

[45] A. Lavault, A. Roebel, and M. Voiry, "Style-based synthesis of drum sounds with extensive controls using generative adversarial networks," in *Proc. Int. Sound and Music Computing Conf.*, 2022.

[46] A. Ramires *et al.*, "The Freesound Loop Dataset and annotation tool," in *Proc. Int. Soc. Music Information Retrieval Conf.*, 2020, pp. 287–294.

[47] J. Zhao, M. Mathieu, and Y. LeCun, "Energy-based generative adversarial network," in *Proc. Int. Conf. Learning Representations*, 2017.

[48] E. Schonfeld, B. Schiele, and A. Khoreva, "A u-net based discriminator for generative adversarial networks," in *Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition*, 2020, pp. 8207–8216.

[49] Y. Jiang, S. Chang, and Z. Wang, "TransGAN: Two pure Transformers can make one strong GAN, and that can scale up," in *Proc. Conf. Neural Information Processing Systems*, 2021.

[50] J. Yang *et al.*, "VocGAN: A high-fidelity real-time vocoder with a hierarchically-nested adversarial network," in *Proc. INTERSPEECH*, 2020.

[51] J. Kong, J. Kim, and J. Bae, "Hifi-GAN: Generative adversarial networks for efficient and high fidelity speech synthesis," in *Proc. Conf. Neural Information Processing Systems*, 2020.

[52] J. Kim, S. Lee, J. Lee, and S. Lee, "Fre-GAN: Adversarial frequency-consistent audio synthesis," in *Proc. INTERSPEECH*, 2021.

[53] J. F. Gemmeke *et al.*, "Audio Set: An ontology and human-labeled dataset for audio events," in *Proc. IEEE Int. Conf. Acoustics, Speech and Signal Processing*, 2017.

[54] S. Böck *et al.*, "Madmom: A new Python audio and music signal processing library," in *Proc. ACM Multimedia Conf.*, 2016.

[55] "pyrubberband," https://pyrubberband.readthedocs.io/.

[56] P. Pope, C. Zhu, A. Abdelkader, M. Goldblum, and T. Goldstein, "The intrinsic dimension of images and its impact on learning," *arXiv preprint arXiv:2104.08894*, 2021.

[57] M. F. Naeem *et al.*, "Reliable fidelity and diversity metrics for generative models," in *Int. Conf. Machine Learning*, 2020.

[58] T. Salimans *et al.*, "Improved techniques for training GANs," in *Proc. Conf. Neural Information Processing Systems*, 2016, pp. 2226–2234.

[59] S. Barratt and R. Sharma, "A note on the inception score," in *Proc. ICML Works. Theoretical Foundations and Applications of Deep Generative Models*, 2018.

[60] K. Kilgour *et al.*, "Fréchet Audio Distance: A metric for evaluating music enhancement algorithms," *arXiv preprint arXiv: 1812.08466*, 2019.

[61] T. Kynkäänniemi *et al.*, "Improved precision and recall metric for assessing generative models," in *Proc. Conf. Neural Information Processing Systems*, 2019, p. 3929–3938.

[62] P. Esser, R. Rombach, and B. Ommer, "Taming Transformers for high-resolution image synthesis," in *Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition*, 2021.

[63] X. Chen and K. He, "Exploring simple Siamese representation learning," *arXiv preprint arXiv:2011.10566*, 2020.

[64] K. He *et al.*, "Masked autoencoders are scalable vision learners," *arXiv preprint arXiv:2111.06377*, 2021.

[65] S. Liu, D. Su, and D. Yu, "DiffGAN-TTS: High-fidelity and efficient text-to-speech with denoising diffusion GANs," *arXiv preprint arXiv:2201.11972*, 2022.

[66] Z. Wang, H. Zheng, P. He, W. Chen, and M. Zhou, "Diffusion-GAN: Training GANs with diffusion," *arXiv preprint arXiv:2206.02262*, 2022.

MODELING THE RHYTHM FROM LYRICS FOR MELODY GENERATION OF POP SONG

Daiyu Zhang Ju-Chiang Wang Katerina Kosta Jordan B. L. Smith Shicen Zhou
ByteDance
{daiyu.zhang, ju-chiang.wang, katerina.kosta, jordan.smith, zhoushicen}@bytedance.com

ABSTRACT

Creating a pop song melody according to pre-written lyrics is a typical practice for composers. A computational model of how lyrics are set as melodies is important for automatic composition systems, but an end-to-end lyric-to-melody model would require enormous amounts of paired training data. To mitigate the data constraints, we adopt a two-stage approach, dividing the task into lyric-to-rhythm and rhythm-to-melody modules. However, the lyric-to-rhythm task is still challenging due to its multimodality. In this paper, we propose a novel lyric-to-rhythm framework that includes part-of-speech tags to achieve better text-setting, and a Transformer architecture designed to model long-term syllable-to-note associations. For the rhythm-to-melody task, we adapt a proven chord-conditioned melody Transformer, which has achieved state-of-the-art results. Experiments for Chinese lyric-to-melody generation show that the proposed framework is able to model key characteristics of rhythm and pitch distributions in the dataset, and in a subjective evaluation, the melodies generated by our system were rated as similar to or better than those of a state-of-the-art alternative.

1. INTRODUCTION

Setting lyrics to a melody is a common but complex task for a composer. The form, articulation, meter, and symmetry of expression in lyrics can inspire, or set constraints on, the melodic arrangement. Given the importance of melody, it is unsurprising that the decades-long history of Music Metacreation systems includes countless melody-creation systems (see [1] for a review). However, less attention has been paid to the lyric-to-melody generation task (i.e., generating a melody for given input lyrics). The task is challenging for many reasons, including but not limited to: the need to handle the prosody of the text correctly (e.g., one should avoid setting an unstressed word like 'the' on a stressed note in the melody); the need to reflect the structure of the lyrics in the melody; and the need to create a good melody to begin with.

© D. Zhang, J.-C. Wang, K. Kosta, J. B. L. Smith, and S. Zhou. Licensed under a Creative Commons Attribution 4.0 International License (CC BY 4.0). **Attribution:** D. Zhang, J.-C. Wang, K. Kosta, J. B. L. Smith, and S. Zhou, "Modeling the rhythm from lyrics for melody generation of pop song", in *Proc. of the 23rd Int. Society for Music Information Retrieval Conf.,* Bengaluru, India, 2022.

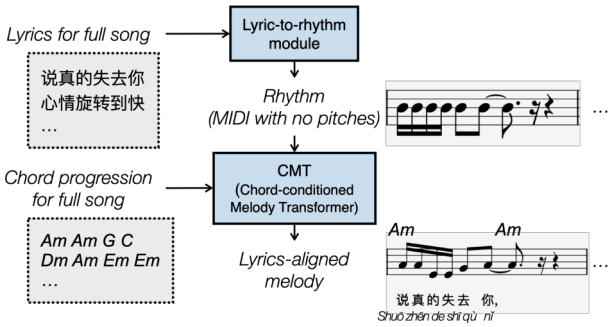

Figure 1: Diagram of the proposed system.

With the rapid growth of deep learning tools, this task has gained more attention, and there are many recent examples of lyric-to-melody creation systems, most using an end-to-end approach [2–5]. Modeling the relationship between lyric syllables and musical notes is a complex, cross-modal task, but it is hoped that we can succeed with a large amount of paired examples (i.e., lyrics aligned to their corresponding melodies). However, acquiring such data is expensive, and using unsupervised learning has shown limited performance gains [3]. All the systems mentioned here are trained on fewer than 200,000 examples of song lyrics; by contrast, the text-to-image system DALL-E has 12 billion parameters and involved hundreds of millions of paired text-image training data [6].

One alternative, suggested in [7], is to pick an intermediate representation and adopt a two-stage approach: one model to convert lyrics to the chosen representation, and a second to convert that to a melody. The motivation is that there is sufficient data to train each model separately, without the paired lyrics-melody data required by the end-to-end approach.

We choose 'rhythm' as the intermediate step because, if we disregard melismas and expressive singing techniques, we can assume there is a one-to-one correspondence between syllables and onsets, and between onsets and melody pitches. Also, there is plenty of data to model each step: first, from karaoke-style scrolling lyrics data, we can obtain an alignment between syllables in lyrics and note onsets in music, and thus note durations and metrical positions, too. Second, there are multiple public datasets from which to learn to assign pitches for each note given their duration. Our goal is then to solve two sub-tasks, namely lyric-to-rhythm and rhythm-to-melody, with an as-

sumption that the rhythm generation process is independent of the pitch generation one [7].

There are many recent melody generation models [8–11], but lyric-to-rhythm modeling is rarely attempted. In this paper, we introduce a novel framework for converting lyrics to rhythms using an encoder-decoder Transformer architecture [12]. The proposed system is outlined in Fig. 1: given an input set of lyrics, a lyric-to-rhythm module assigns onset times and durations for each syllable. This rhythm, along with a user-provided chord progression, is fed into a Chord-conditioned Melody Transformer (CMT) [13], a state-of-the-art melody generation system, to predict the pitch for each note. The details of the lyric-to-rhythm module and the CMT are provided in Sections 3.3 and 3.2, respectively.

2. BACKGROUND

Lyrics and melody are not arbitrarily combined; common sense suggests and prior analysis [14] indicates that patterns in lyrics and melodies are related and can be modeled, in part, with features of the melody (e.g., note duration) and lyrics (e.g., syllable stress). One of the earliest lyric-to-melody systems was designed to handle Japanese prosody [15]: first, the input text was segmented into phrases; next, a set of pre-composed rhythms was searched for one that fit the syllable count and matched the accent pattern of the text; finally, pitches were assigned using dynamic programming to optimise the interval directions with the natural prosody of the words. An earlier lyric-to-rhythm system also leveraged a dataset of pre-composed rhythms that were scored based on their match to the input syllable-stress and word-rarity patterns [16]. Although our system has little in common with these works, we do share the use of rhythm as an intermediate representation.

Algorithms for automatic music generation are a subset of Music Metacreation systems [1], which have been present in Western music in many forms, including being used for the creation of standalone pieces and, either offline or in real-time, as part of the human composition process. With the help of machine learning and deep learning architectures, many such systems have shown to be capable of generating a plausible outcomes that match the musical characteristics of given datasets. Supervised generative models aim to learn a representation of the underlying characteristics of a training set distribution. Depending on the model, this representation can be either explicitly depicted or implicitly used to generate samples from the learned distribution [17].

Some systems aim to generate a part of a musical piece with the aid of another given part (including melody-to-lyrics creation [18], the inverse of the task we consider). Conditioning the choice of parameters in a generative model on data from other modalities, such as a bass line or a structure, can yield controllable generation systems [19][p.82-83]. For the case of using chords to condition melody generation, a recent system adjusting a general adversarial network architecture has been presented in [20] with the option of generating melody lines over a given

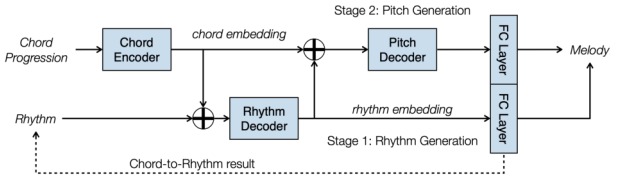

Figure 2: A two-stage structure of CMT, where $\oplus$ represents concatenation. Stage 1: chord-to-rhythm. Stage 2: rhythm+chord-to-pitch based on the result of Stage 1.

accompaniment. The Chord-conditioned Melody Transformer (CMT) [13] is the most recent effort in this area; we adapt much of the design of this system, extending it to accept both lyrics and chords as input. Details of this system, and how we adapt it, follow in Section 3.

3. METHODOLOGY

3.1 System Overview

Our system design is motivated by the Chord-conditioned Melody Transformer (CMT) [13]. The authors of CMT proposed a two-stage system, assuming a hierarchy that the process of generating melodies is two-phase, as depicted in Fig. 2: *Stage 1*, generating the rhythm of notes from chord progressions; *Stage 2*, generating the pitch for each note depending on the chord progressions and generated rhythm. Our proposed system augments CMT by replacing chord-to-rhythm (i.e., Stage 1) with a novel *lyric-to-rhythm module*. As a result, users can input the lyrics and chord progression of a full song in our system (see Fig. 1). Then, the lyric-to-rhythm module generates the MIDI (with empty pitches). Second, CMT processes the MIDI and chord progression to generate the melody. As a result, the rhythm is generated with a global view of the lyrics, while the melody is generated with a causal view of the rhythm and chords.

In the following subsections, we will first review CMT and explain the difficulties of modifying it to handle the lyric-to-melody task in Section 3.2. Then, we will detail our solution in Section 3.3.

3.2 Chord-Conditioned Melody Transformer (CMT)

CMT adopts a pianoroll-like representation [13, 21] that includes chord, rhythm, and pitch (CRP) information. It splits the timeline into semiquaver-length frames (1/4 of a beat), each described by three vectors: a 12-dimensional binary *chord* vector (pitch classes in the chord get a 1); a 3-dimensional one-hot *rhythm* vector (onset, hold state, rest state); and a 22-dimensional one-hot melodic *pitch* vector (for this part we restrict MIDI pitches to between 48 and 67, plus a hold state and a rest state, giving a total dimension of 22). Please refer to [13][Fig. 1] for an illustration.

CMT contains three main modules: *Chord Encoder (CE)*, a bidirectional LSTM [22]; *Rhythm Decoder (RD)*, a stack of self-attention blocks; and *Pitch Decoder (PD)*, another stack of self-attention blocks. In Stage 1, given an input chord progression, the chord embedding encoded