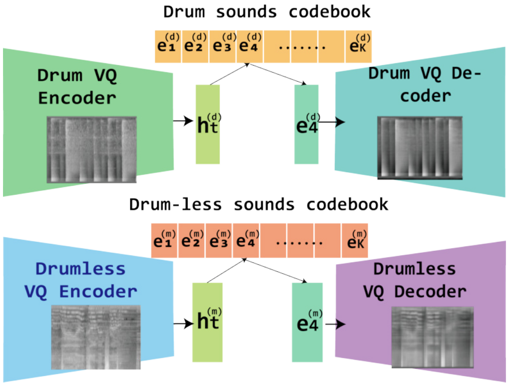

Figure 2: We use separate VQ-VAEs for the drumless and drum tracks, both operating on the Mel-spectrograms.

Specifically, VQ-VAE consists of an encoder and a decoder, referred to as the VQ-VAE encoder $\mathcal{E}$ and VQ-VAE decoder $\mathcal{D}$ below, respectively. Given an audio waveform $\mathbf{x} \in \mathbb{R}^{1\times N}$, where N denotes the number of audio samples (e.g., $N = 1{,}058{,}400$ for a 24-second audio clip sampled at 44.1 kHz), the VQ-VAE encoder would convert $\mathbf{x}$ into a sequence of latent vectors $\{\mathbf{h}_t \in \mathbb{R}^D, t = 1, ..., T\}$, where the sequence length T is proportional to N, and the VQ-VAE decoder would have to reconstruct $\mathbf{x}$ from the "vector-quantized" version of the latent vectors, denoting as $\{\mathbf{h}'_t \in \mathbb{R}^D, t = 1, ..., T\}$, that is obtained by finding the nearest prototype vector $\mathbf{h}'_t = \mathbf{e}_z$ of each $\mathbf{h}_t$ in a codebook of prototype vectors $\{\mathbf{e}_k \in \mathbb{R}^D, k = 1, ..., K\}$, where K is the size of the codebook. Each prototype vector can be regarded as a cluster centroid in the latent space as a result of K-means clustering. The VQVAE encoder/decoder and the codebook are jointed learned by minimizing the reconstruction loss $\|\mathbf{x}_t - \mathcal{D}(\mathbf{h}'_t)\|_2^2$, and the commitment loss of the clustering $\|\mathcal{E}(\mathbf{x}_t) - sg(\mathbf{e}_z)\|_2^2$, where $\mathbf{x}_t$ denotes a slice of $\mathbf{x}$, $sg(.)$ the stop-gradient operation, and $\mathbf{h}_t = \mathcal{E}(\mathbf{x}_t)$.

Once the VQ-VAE is trained, the $\mathbf{x}$ can be viewed as a sequence of "IDs" $\{z_t \in \mathbb{Z}_{1:K}, t = 1, ..., T\}$, each corresponding to the index of the element of the codebook that is used to represent each slice of $\mathbf{x}$. The Transformer can then be trained on such sequences of IDs to learn the underlying language, or "composition rules," of the audio codes. Once trained, the transformer can be used to generate a novel sequence of codes, which can then be converted into an audio waveform by the VQVAE decoder.

As the waveforms are extremely long, Jukebox actually uses a "multi-scale" VQ-VAE that converts a waveform into three levels of codes, and accordingly three Transformers for building the LM at each level [7]. KaraSinger works on Mel-spectrograms but also uses multi-scale VQ-VAE [8]. We simplify their architecture by working on Mel-spectrograms with only one level of codes instead of multiple levels, as introduced below.

3. PROPOSED METHODS

In our task, we are given pairs of audio waveforms, namely a drumless stem $\mathbf{x}^{\mathrm{m}}$ and a drum stem $\mathbf{x}^{\mathrm{d}}$. Our goal is to train a model that can generate $\mathbf{x}^{\mathrm{d}}_*$ given an unseen $\mathbf{x}^{\mathrm{m}}_*$ from the test set. To achieve so, we propose several extensions of the Jukebox model. While we focus on drum accompaniment generation only, the same methodology may apply equally well to other conditional generation tasks.

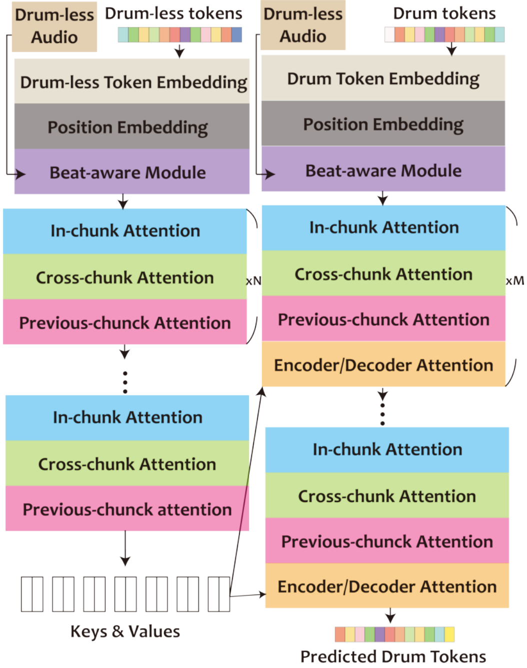

Figure 3: Details of our Transformer encoder/decoder.

Two VQ-VAEs. As illustrated in Figure 2, we build separate VQ-VAEs for $\mathbf{x}^{\mathrm{m}}$ and $\mathbf{x}^{\mathrm{d}}$ using drumless stems and drum stems respectively. Once trained, the drumless VQ-VAE encoder and the drum VQ-VAE encoder would convert $\mathbf{x}^{\mathrm{m}}$ and $\mathbf{x}^{\mathrm{d}}$ into $\{z^{\mathrm{m}}_t \in \mathbb{Z}_{1:K^{\mathrm{m}}}, t = 1, ..., T\}$ and $\{z^{\mathrm{d}}_t \in \mathbb{Z}_{1:K^{\mathrm{d}}}, t = 1, ..., T\}$ separately. Assuming that the drumless tracks are more diverse, we suggest use a larger codebook size for the drumless sounds codebook than the drum sounds codebook, namely $K^{\mathrm{m}} \geq K^{\mathrm{d}}$.

Mel VQ-VAE & Vocoder. To reduce computational cost, we build VQ-VAEs that take Mel-spectrograms as the input and target output. Specifically, we use the convolutional blocks of UNAGAN [5] to build a pair of VQ-VAE encoder and VQ-VAE decoder that has symmetric architectures (one downsampling and the other upsampling). We omit the details of UNAGAN due to space limit. Our Transformer accordingly learns the LM for codes of the Mel-spectrograms. Once a novel sequence of (drum) codes is generated by the Transformer, we use our (drum) VQ-VAE decoder to convert the codes into a Mel-spectrogram, and then use a neural *vocoder* [42] to convert the Mel-spectrogram into the corresponding waveform. While the latent vector $\mathbf{h}_t$ (and accordingly z_t) corresponds to a slice of waveform in the original Jukebox, here the latent vectors $\mathbf{h}^{\mathrm{m}}_t$ and $\mathbf{h}^{\mathrm{d}}_t$ (and z^{m}_t, z^{d}_t) both correspond to a fixed number of L frames of the short-time Fourier Transform (STFT) while computing the Mel-spectrograms.

Seq2seq Transformer. While the Jukebox model uses a Transformer decoder to model the sequences $\{z_t, t = 1, ..., T\}$ corresponding to mixtures of sounds, in our case we need a dedicated Transformer encoder to take the se-

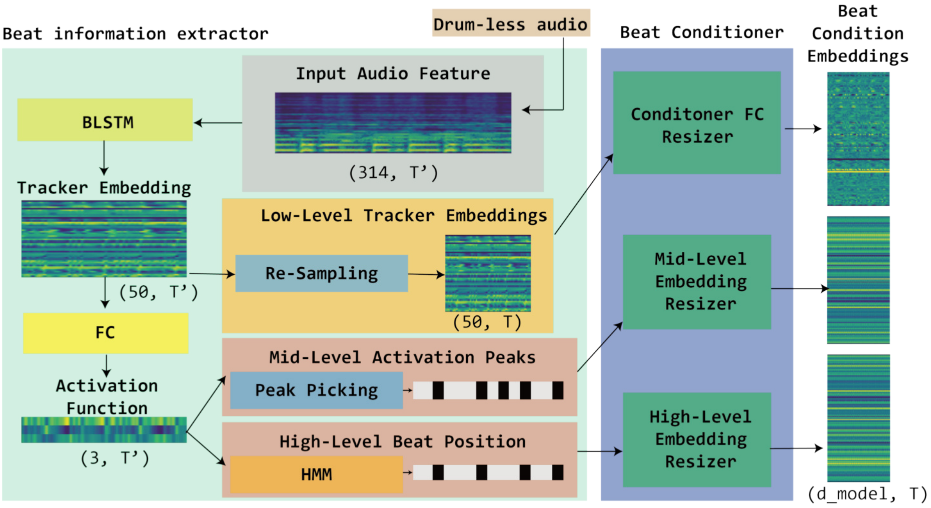

Figure 4: Details of the proposed beat-aware module that extracts beat condition embeddings from the drumless audio.

quence $\{z_t^{\mathrm{m}}, t = 1, ..., T\}$ corresponding to a drumless audio as the input, and a Transformer decoder to generate the sequence $\{z_t^{\mathrm{d}}, t = 1, ..., T\}$ corresponding to the accompanying drum audio as the output, leading to a sequence-to-sequence (seq2seq) architecture. Following Jukebox, we use the attention mechanism of the Scalable Transformer [15] in our Transformer encoder/decoder. As depicted in Figure 3, to determine its output z_i^{d} at each i, our Transformer decoder uses factorized "in-chunk," "cross-chunk" and "previous-chunk" attention layers to attend to the drum codes it generates previously, namely $z_{<i}^{\mathrm{d}} = \{z_t^{\mathrm{d}}, t = 1, ..., i-1\}$, to maintain the coherence of its output. A "chunk" here is a slice of the corresponding sequence. Moreover, the Transformer decoder uses "encoder/decoder attention" to attend to the final layer of the Transformer encoder to get contextual information from the drumless code sequence $\{z_t^{\mathrm{m}}, t = 1, ..., T\}$. To better synchronize the input and output, position embeddings are used. We refer readers to [7, 15] for details.

Beat-Aware Module. Figure 3 also shows that we use a novel "beat-aware module" (after the position embedding) in both Transformer encoder and decoder. We note that the code z_t^{m} is trained to provide essential information needed to reconstruct the drumless audio $\mathbf{x}_t^{\mathrm{m}}$, so the information carried by z_t might be *mixed*, covering different musical aspects including timbre, style, rhythm, etc. To supply the Transformer with *clear* rhythm-related information of $\mathbf{x}^{\mathrm{m}}$, we might need such a dedicated beat-aware module.

As depicted in Figure 4, the beat-aware module consists two sub-modules, the "beat information extractor" and the "beat conditioner." The former extracts beat-related information per frame from $\mathbf{x}^{\mathrm{m}}$ using an existing beat/downbeat tracker [19], while the latter incorporates that beat-related information for each t as a beat condition embedding $\mathbf{c}_t^{\mathrm{m}} \in \mathbb{R}^{d_{\text{model}}}$ that has the same length d_{model} as the token embedding of the Transformers, and adds together the beat condition embedding and token embedding per t to serve as the input to the subsequent attention layers.

Despite that deep learning-based beat/downbeat trackers such as those proposed by Böck *et al.* [43–45] have achieved excellent performance for mainstream pop, rock or dance music, their performance and behaviors are influenced by the sound source composition (i.e., drum/non-drum sounds) of their training data [19, 46]. Considering that our input music is drumless, which is quite different from the music that existing common beat/downbeat trackers [44, 45] are trained with and trained for, we use in our beat information extractor the "non-drum tracker" developed by Chiu *et al.* [19] exclusively for drumless stems. In Section 4, we describe three ways to extract different levels of beat information feature from the non-drum tracker.

4. BEAT CONDITION EMBEDDING

As shown on the left of Figure 4, following the common design [44], the beat/downbeat tracker [19] uses BLSTM layers to get 50-dimensional "tracker embeddings" from $\mathbf{x}^{\mathrm{m}}$, and then uses fully-connected (FC) layers to compute 3-dimensional "activation functions" indicating the likelihood of observing a beat, downbeat, or non-beat at each frame. Finally, either a simple peak-picking or a hidden Markov model (HMM)-based algorithm [44] can be used to finalize the beat and downbeat time positions from the activation functions. We accordingly investigate extracting features from different stages of this processing pipeline.

Low-level (tracker) embeddings. We simply use the high-dimensional tracker embeddings, resampling it temporally, pooling them every L frames, and converting each of them to the desired length d_{model} with an FC.

High-level beat/downbeat positions. From the beat and downbeat time positions estimated by HMM, a frame can be labeled as either a beat, downbeat, or non-beat. We learn three d_{model}-dimensional embedding vectors corresponding to each case, and represent every L frames with one of the three vectors according to the frame labels.

Mid-level activation peaks. As the beat/downbeat positions are sparse over time, we consider a "denser" version by simply picking the peak positions of the activation functions by `scipy.signal.find_peaks` [47] and similarly learning embedding vectors that are assigned to the frames according to whether a frame corresponds to a peak or not. Such a peak might represent a musical onset.

5. EXPERIMENT SETUP

Multi-track datasets for research on musical source separation [17] usually host recordings that each consists of an isolated drum stem along with stems corresponding to other instruments. We can simply take the drum stem as the target drum track, and the summation of the remaining stems as the input drumless track. In our implementation, we used the multi-track recordings from three datasets. MUSDB18 [17] contains 150 recordings, each of which has four stems corresponding to vocal, drums, bass, and "others." It is commonly used in source separation. MedleyDB [48] contains 196 multi-track recordings, each of which includes a drum stem along with many other stems. MixingSecret [49] has 257 multi-track recordings with various instruments, all including drums. We removed the 46 duplicate recordings between MUSDB18 and MedleyDB and the 100 duplicate recordings between MUSDB18 and MixingSerect, leading to 457 recordings to be used in our work. We randomly split the recordings into 80%, 10%, 10% as the training set, validation set (for parameter tuning), and testing set (for objective and subjective evaluation) at the "recording-level," ensuring that a recording does not appear in different splits.

All the recordings are sampled at 44.1 kHz and the stems are all monaural. Following Jukebox [7], we used 24-second audio clips in our work. We sliced each recording (and accordingly the stems) to 23.8-second audio clips with 50% temporal overlaps (as $2^{20}/44100 = 23.8$), and discarded the clips that do not contain any drum sounds. We then computed the Mel-spectrograms of each clip with PyTorch v1.7.1 with a Hann window of 1,024 samples for STFT, a hop size of 256 samples and 80 Mel-filter banks.

To evaluate the performance of the proposed JukeDrummer and validate the effectiveness of model components, we adopted the following variants in our experiments. [2]

- `seq2seq+beat (low)`: given a drumless clip $\mathbf{x}^{\mathrm{m}}$, we computed the drumless codes $\{z_t^{\mathrm{m}}\}$ and the low-level beat/downbeat tracker embeddings $\{\mathbf{c}_t^{\mathrm{m}}\}$ as the model input, to predict the drum codes $\{z_t^{\mathrm{d}}\}$ that eventually lead to the generated drum clip $\mathbf{x}^{\mathrm{d}}$, using the proposed seq2seq Transformer.
- `seq2seq+beat (mid)`: using the beat condition embeddings computed from the mid-level activation peaks (see Section 4) for $\{\mathbf{c}_t^{\mathrm{m}}\}$ instead.
- `seq2seq+beat (high)`: using the beat condition embeddings from the high-level beat/downbeat positions as $\{\mathbf{c}_t^{\mathrm{m}}\}$ instead.
- `seq2seq w/o beat`: to study the usefulness of the beat conditioning, we predicted $\{z_t^{\mathrm{d}}\}$ from $\{z_t^{\mathrm{m}}\}$, not using any beat-related conditions at all.
- `decoder+beat (low)`: to study the usefulness of the drumless codes $\{z_t^{\mathrm{m}}\}$, we used only the low-level beat condition embeddings $\{\mathbf{c}_t^{\mathrm{m}}\}$ as input to predict $\{z_t^{\mathrm{d}}\}$, via a Transformer decoder-only architecture (i.e., not seq2seq Transformer).
- `decoder w/o beat`: this is the baseline drummer that "plays its own," generating $\{z_t^{\mathrm{d}}\}$ autoregressively via a Transformer decoder without taking any information (neither $\{z_t^{\mathrm{m}}\}$ nor $\{\mathbf{c}_t^{\mathrm{m}}\}$) from the drumless clip $\mathbf{x}^{\mathrm{m}}$ it is supposed to play along to.

While the Mel-spectrogram for a clip in our case has 4,096 frames, the VQ-VAE encoder downsamples it to a sequence of T = 1,024 latent vectors, namely each corresponding to $L = 4$ frames. For VQ-VAE, we set the codebook size of drumless sounds K^{m} = 1,024 and that of drums $K^{\mathrm{d}} = 32$ (so $K^{\mathrm{m}} \gg K^{\mathrm{d}}$), and the dimension of the latent vectors $D = 64$. The VQ-VAE encoders/decoders for both drumless and drums all have two layers. For those models employing a seq2seq LM, the Transformer encoder has 9 layers and the Transformer decoder has 20 layers. [3]

At inference time, we used the drum VQ decoder to convert the drum codes $\{z_t^{\mathrm{d}}\}$ to a Mel-spectrogram, which is then turned into the waveform of the drum clip $\mathbf{x}^{\mathrm{d}}$ by a HiFi-GAN V1 vocoder [42]. We trained the vocoder from scratch with audio of drum sounds from our dataset for 2.5 days, and then, inspired by [50,51], fine-tuned it on the reconstructed Mel-spectrograms of the Drum VQ decoder.

6. OBJECTIVE EVALUATION

As audio-domain drum accompaniment generation is new, we propose customized metrics. Specifically, we use the "drum tracker" trained exclusively for drum stems by Chiu *et al.* [19], which follows exactly the same pipeline as the non-drum tracker (cf. Figure 4), to extract rhythmic features at three different levels for the ground-truth and generated drum sounds for the test split. We then compare the rhythmic features of the ground-truth and the generated in such three levels, using the mean square error for the low-level tracker embeddings (**TrackEmb-MSE**), cross entropy for the mid-level activation functions (**Act-Entropy**), and the F-measure for beat/downbeat estimation (**B/DB-F1**). The last one, computed by `mir_eval` [52], uses the beat positions of the ground-truth drums as the reference and those of the generated drums as the estimated beats. These metrics evaluates only the *rhythmic consistency* between the generated drums and the drumless audio (using its human-made drum track as a proxy), not other aspects such as stylistic consistency and audio quality, which will be evaluated subjectively in Section 7.

[2] A reviewer suggested that we should have compared our model with other accompaniment generation models that operate in the symbolic domain such as [20] and [21], saying that we can use existing audio samples or drum synthesis models to render their output to audio. However, the problem is that such a symbolic-domain accompaniment generation model also requires its input to be a MIDI file rather than audio.

[3] We used Adam as our optimizer with a learning rate of 0.0003 for both VQ-VAE and Transformer LM. We trained our LM with a batch size of 16, input feature dimension $d_{\text{model}} = 512$, two heads for multi-head attention, and the chunk size for factorized attention being 16. For those employing a decoder-only architecture, the number of layers of the Transformer decoder reduces to 15 because 5 of the layers corresponding to the "encoder/decoder attention" are removed.

Model	Objective metrics			Subjective MOS ($\in [1,5]$)				
	TrackEmd-MSE↓	Act-Entropy↓	B/DB-F1↑	$\mathbf{R}^{\mathtt{m/d}}$	$\mathbf{S}^{\mathtt{m/d}}$	$\mathbf{Q}^{\mathtt{d}}$	$\mathbf{R}^{\mathtt{d}}$	$\mathbf{O}^{\mathtt{d}}$
`seq2seq+beat (low)`	**.068**±.023	**.928**±.189	**.340**	**3.27**	**3.44**	**3.39**	**3.37**	**3.20**
`seq2seq+beat (mid)`	.077±.022	.963±.200	.212	—	—	—	—	—
`seq2seq+beat (high)`	.080±.024	.938±.160	.132	—	—	—	—	—
`seq2seq w/o beat`	.081±.022	.968±.233	.110	1.78	2.34	2.95	2.58	2.05
`decoder+beat (low)`	**.068**±.023	**.931**±.200	**.339**	**3.05**	**3.34**	**3.33**	**3.07**	**3.17**
`decoder w/o beat`	.087±.025	.987±.240	.114	1.59	1.83	2.56	1.88	1.73
Real data (not vocoded)	—	—	—	4.39	3.95	3.85	4.61	4.17

Table 1: Results of objective and subjective evaluation of variants of the proposed JukeDrummer model. The metrics are (from left to right): beat/downbeat tracking embedding MSE, beat/downbeat activation entropy, beat/downbeat F1, $\mathbf{R}^{\mathtt{m/d}}$ (rhythmic consistency), $\mathbf{S}^{\mathtt{m/d}}$ (stylistic consistency), $\mathbf{Q}^{\mathtt{d}}$ (audio quality), $\mathbf{R}^{\mathtt{d}}$ (rhythmic stability), $\mathbf{O}^{\mathtt{d}}$ (overall). ↓/↑: the lower/higher the better; best two results (among the six model variants) per column highlighted in bold.

Result shown in the middle of Table 1 clearly shows that the models with low-level beat information achieve lower MSE, cross entropy, and higher F1, suggesting the usefulness of the low-level beat information for rhythmic consistency. The mid-level and high-level ones seem less effective (possibly because they are relatively monotonic), so we did not evaluate them further in Section 7. The objective scores also suggest that the Transformer encoder does not contribute much to rhythmic consistency.

7. SUBJECTIVE EVALUATION

With an online study, we solicited 22 anonymous volunteers to rate the result for 3 out of 15 random drumless tracks (each 23.8 seconds) from the test split. Each time, a volunteer listened to a drumless tracks ($\mathbf{x}^{\mathrm{m}}_{*}$) first, and then (in random orders) the mixture (i.e., $\mathbf{x}^{\mathrm{m}}_{*} + \mathbf{x}^{\mathrm{d}}_{*}$) containing drum samples generated by four different models, plus the real human-made one (to set a high anchor). The volunteer then rated them in the following aspects on a 5-point Likert scale: **rhythmic consistency** between the drumless input and generated drums; **stylistic consistency** concerning the timbre and arrangement of the drumless input and generated drums; **audio quality** and **rhythmic stability** (whether the drummer follows a steady tempo) of the generated drum; and **overall** perceptual impression.

The mean opinion scores (MOS) in Table 1 show that `seq2seq+beat (low)` consistently outperforms the others, validating the effectiveness of using both the drumless codes and beat conditions. `decoder+beat (low)` performs consistently the second best, outperforming the two models without beat information significantly in three aspects according to paired t-test (p-value$<$ 0.05), validating again the importance of the beat-aware module. Complementing Section 6, the MOS result suggests that the beat conditions seem more important than the drumless codes, though the best result is obtained with both.

Figure 5 further demonstrates that, given the same input, our model can generate multiple accompaniments with diversity in both beat and timbre. Diversity is an interesting aspect that is hard to evaluate, but it is desirable as there is no single golden drum accompaniment for a song. This may also explain why the F1 scores in Table 1 seem low.

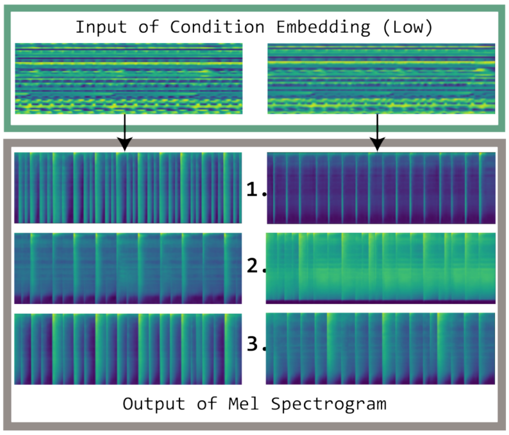

Figure 5: Two sets of three different generated samples by the same model given the same beat condition embedding.

Verbal feedbacks from the subjects confirm that our best model generates drum accompaniment that is rhythmically and stylistically consistent with the input, especially for band music or music with heavy use of bass. However, the model still has limits. At times the model generates total silence, though it can be avoided by sampling the LM again. The model may struggle to change its tempo going through different sections of a song. Moreover, the generation might be out-of-sync with the input in the beginning few seconds, until the model gets sufficient context. Please visit the demo page for various examples.

8. CONCLUSION

We have presented JukeDrummer, a novel audio-to-audio extension of OpenAI's JukeBox model capable of adding the drum part of a drumfree recording in the audio domain. To our knowledge, this represents the first attempt to audio-domain generation conditioned on drumless mixed audio. With objective and subjective evaluations, we validated the effectiveness of the customized VQ-VAE plus the seq2seq Transformer design, and the proposed beat-aware module. Among the beat conditions, we found that the low-level embeddings work the best. Future work can be done to further improve the language model (LM), and to extend our work to other audio-to-audio generation tasks.

9. ACKNOWLEDGEMENT

We are grateful to the anonymous reviewers for their valuable comments. This research is funded by grant NSTC 109-2628-E-001-002-MY2 from the National Science and Technology Council of Taiwan.

10. REFERENCES

[1] J. Engel, C. Resnick, A. Roberts, S. Dieleman, D. Eck, K. Simonyan, and M. Norouzi, "Neural audio synthesis of musical notes with WaveNet autoencoders," in *Proc. Int. Conf. Machine Learning*, 2017.

[2] 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.

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

[4] 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.

[5] 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.

[6] J. Engel, L. Hantrakul, C. Gu, and A. Roberts, "DDSP: Differentiable digital signal processing," in *Int. Conf. Learning Representations*, 2021.

[7] P. Dhariwal, H. Jun, C. Payne, J. W. Kim, A. Radford, and I. Sutskever, "Jukebox: A generative model for music," *arXiv preprint arXiv:2005.00341*, 2020.

[8] C.-F. Liao, J.-Y. Liu, and Y.-H. Yang, "KaraSinger: Score-free singing voice synthesis with VQ-VAE using Mel-spectrograms," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, 2022, pp. 956–960.

[9] C. M. Payne, "MuseNet," *OpenAI Blog*, 2019.

[10] C. Donahue *et al.*, "LakhNES: Improving multi-instrumental music generation with cross-domain pre-training," in *Proc. Int. Soc. Music Information Retrieval Conf.*, 2019.

[11] Y.-H. Chen, Y.-H. Huang, W.-Y. Hsiao, and Y.-H. Yang, "Automatic composition of guitar tabs by Transformers and groove modeling," in *Proc. Int. Soc. Music Inf. Retr. Conf.*, 2020.

[12] Z. Wang, Y. Zhang, Y. Zhang, J. Jiang, R. Yang, J. Zhao, and G. Xia, "PianoTree VAE: Structured representation learning for polyphonic music," in *Proc. Int. Soc. Music Inf. Retr. Conf.*, 2020.

[13] M. Suzuki, "Score Transformer: Transcribing quantized MIDI into comprehensive musical score," in *Proc. Int. Soc. Music Inf. Retr. Conf., Late-breaking demo paper*, 2021.

[14] A. Vaswani *et al.*, "Attention is all you need," in *Proc. Advances in Neural Information Processing Systems*, 2017, pp. 5998–6008.

[15] R. Child, S. Gray, A. Radford, and I. Sutskever, "Generating long sequences with sparse Transformers," *arXiv preprint arXiv:1904.10509*, 2019.

[16] C. A. Huang, H. V. Koops, E. Newton-Rex, M. Dinculescu, and C. J. Cai, "AI song contest: Human-AI co-creation in songwriting," in *Proc. Int. Soc. Music Inf. Retr. Conf.*, 2020, p. 708–716.

[17] Z. Rafii *et al.*, "The MUSDB18 corpus for music separation," 2017. [Online]. Available: https://doi.org/10.5281/zenodo.1117372

[18] E. Deruty, M. Grachten, S. Lattner, J. Nistal, and C. Aouameur, "On the development and practice of AI technology for contemporary popular music production," *Transactions of the International Society for Music Information Retrieval*, vol. 5, no. 1, pp. 35–49, 2022.

[19] C.-Y. Chiu, W.-Y. Su, and Y.-H. Yang, "Drum-aware ensemble architecture for improved joint musical beat and downbeat tracking," *IEEE Signal Processing Letters*, vol. 28, pp. 1100–1104, 2021.

[20] R. Dahale, V. Talwadker, P. Verma, and P. Rao, "Neural drum accompaniment generation," in *Proc. Int. Soc. Music Inf. Retr. Conf., Late-breaking demo paper*, 2021.

[21] D. Makris, G. Zixun, M. Kaliakatsos-Papakostas, and D. Herremans, "Conditional drums generation using compound word representations," in *Proc. Artificial Intelligence in Music, Sound, Art and Design*, 2022, p. 179–194.

[22] J. Gillick, A. Roberts, J. Engel, D. Eck, and D. Bamman, "Learning to groove with inverse sequence transformations," in *Proc. Int. Conf. Machine Learning*, 2019.

[23] V. Thio, H.-M. Liu, Y.-C. Yeh, and Y.-H. Yang, "A minimal template for interactive web-based demonstrations of musical machine learning," in *Proc. Workshop on Intelligent Music Interfaces for Listening and Creation*, 2019.

[24] G. Alain, M. Chevalier-Boisvert, F. Osterrath, and R. Piche-Taillefer, "DeepDrummer: Generating drum loops using deep learning and a human in the loop," *arXiv preprint arXiv:2008.04391*, 2020.

[25] N. Tokui, "Can GAN originate new electronic dance music genres? – Generating novel rhythm patterns using GAN with genre ambiguity loss," *arXiv preprin: 2011.13062*, 2020.

[26] H.-W. Dong, W.-Y. Hsiao, L.-C. Yang, and Y.-H. Yang, "MuseGAN: Symbolic-domain music generation and accompaniment with multi-track sequential generative adversarial networks," in *Proc. AAAI Conf. Artificial Intelligence*, 2018.

[27] I. Simon, A. Roberts, C. Raffel, J. Engel, C. Hawthorne, and D. Eck, "Learning a latent space of multitrack measures," *arXiv preprint arXiv:1806.00195*, 2018.

[28] H.-M. Liu and Y.-H. Yang, "Lead sheet generation and arrangement by conditional generative adversarial network," in *Proc. IEEE. Conf. Machine Learning and Applications*, 2018, pp. 722–727.

[29] Y. Ren, J. He, X. Tan, T. Qin, Z. Zhao, and T.-Y. Liu, "PopMAG: Pop music accompaniment generation," in *Proc. ACM Multimedia Conf.*, 2020.

[30] 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.

[31] A. Ramires, P. Chandna, X. Favory, E. Gómez, and X. Serra, "Neural percussive synthesis parameterised by high-level timbral features," in *Proc. IEEE Int. Conf. Acoustics, Speech and Signal Processing*, 2020.

[32] C. Aouameur, P. Esling, and G. Hadjeres, "Neural drum machine: An interactive system for real-time synthesis of drum sounds," *arXiv preprint arXiv:1907.02637*, 2019.

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

[34] ——, "Style-based drum synthesis with GAN inversion," in *Proc. Int. Soc. Music Inf. Retr. Conf., Late-breaking demo paper*, 2021.

[35] M. Tomczak, M. Goto, and J. Hockman, "Drum synthesis and rhythmic transformation with adversarial autoencoders," in *Proc. ACM Int. Conf. Multimedia*, 2020, p. 2427–2435.

[36] 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.

[37] A. v. d. Oord, O. Vinyals, and K. Kavukcuoglu, "Neural discrete representation learning," in *Proc. Advances in Neural Information Processing Systems*, 2017.

[38] L. Kaiser *et al.*, "Fast decoding in sequence models using discrete latent variables," in *Proc. Int. Conf. Machine Learning*, 2018, pp. 2390–2399.

[39] A. Razavi *et al.*, "Generating diverse high-fidelity images with VQ-VAE-2," in *Proc. Advances in Neural Information Processing Systems*, 2019.

[40] A. Polyak *et al.*, "Speech resynthesis from discrete disentangled self-supervised representations," in *Proc. Interspeech*, 2021, pp. 3615–3619.

[41] J. Walker, A. Razavi, and A. v. d. Oord, "Predicting video with VQVAE," *arXiv preprint arXiv:2103.01950*, 2021.

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

[43] S. Böck, F. Korzeniowski, J. Schlüter, F. Krebs, and G. Widmer, "madmom: A new Python audio and music signal processing library," *Proc. ACM Multimed. Conf.*, pp. 1174–1178, 2016.

[44] S. Böck *et al.*, "Joint beat and downbeat tracking with recurrent neural networks," in *Proc. Int. Soc. Music Information Retrieval Conf.*, 2016.

[45] S. Böck and M. E. P. Davies, "Deconstruct, analyse, reconstruct: How to improve tempo, beat, and downbeat estimation," in *Proc. Int. Soc. Music Inf. Retr. Conf.*, 2020, p. 574–582.

[46] C.-Y. Chiu, J. Ching, W.-Y. Hsiao, Y.-H. Chen, W.-Y. Su, and Y.-H. Yang, "Source separation-based data augmentation for improved joint beat and downbeat tracking," in *Proc. Eur. Signal Process. Conf.*, 2021.

[47] P. Virtanen *et al.*, "SciPy 1.0: Fundamental algorithms for scientific computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[48] R. Bittner *et al.*, "MedleyDB: A multitrack dataset for annotation-intensive MIR research," in *Proc. Int. Soc. Music Inf. Retr. Conf.*, 2014, [Online] http://medleydb.weebly.com/.

[49] "The 'Mixing Secrets' free multitrack download library," [Online] https://www.cambridge-mt.com/ms/mtk/.

[50] K.-W. Kim, S.-W. Park, J. Lee, and M.-C. Joe, "ASSEM-VC: Realistic voice conversion by assembling modern speech synthesis techniques," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, 2022.

[51] X. Tan *et al.*, "NaturalSpeech: End-to-end text to speech synthesis with human-level quality," *arXiv preprint arXiv:2205.04421*, 2022.

[52] C. Raffel *et al.*, "mir_eval: A transparent implementation of common MIR metrics," in *Proc. Int. Soc. Music Inf. Retr. Conf.*, 2014, pp. 367–372.

LEARNING HIERARCHICAL METRICAL STRUCTURE BEYOND MEASURES

Junyan Jiang[1,2] **Daniel Chin**[1,2] **Yixiao Zhang**[1,3] **Gus Xia**[1,2]
[1] Music X Lab, NYU Shanghai [2] MBZUAI [3] Centre for Digital Music, QMUL
jj2731@nyu.edu, daniel.chin@nyu.edu, yixiao.zhang@qmul.ac.uk, gxia@nyu.edu

ABSTRACT

Music contains hierarchical structures beyond beats and measures. While hierarchical structure annotations are helpful for music information retrieval and computer musicology, such annotations are scarce in current digital music databases. In this paper, we explore a data-driven approach to automatically extract hierarchical metrical structures from scores. We propose a new model with a Temporal Convolutional Network-Conditional Random Field (TCN-CRF) architecture. Given a symbolic music score, our model takes in an arbitrary number of voices in a beat-quantized form, and predicts a 4-level hierarchical metrical structure from downbeat-level to section-level. We also annotate a dataset using RWC-POP MIDI files to facilitate training and evaluation. We show by experiments that the proposed method performs better than the rule-based approach under different orchestration settings. We also perform some simple musicological analysis on the model predictions. All demos, datasets and pre-trained models are publicly available on Github [1] .

1. INTRODUCTION

Music contains rich structures at different levels, and the structure annotations play an important role in music understanding [1, 2] and generation [3, 4]. Progress has been made in music structure analysis on certain levels, like beat tracking [5–7], downbeat detection [8–11] and part segmentation [12–15]. These levels of structures are often inter-connected with each other, and can be described in a hierarchical way. For example, a part may contain several sections, each containing several measures. Measures can be further decomposed into beats.

Several views of hierarchical music structures are formally discussed in the Generative Theory of Tonal Music (GTTM) [16], including the grouping structure, the metrical structure, the time-span tree, and the prolongational tree, each focusing on different music properties (i.e., the grouping structure focuses more on melodic grouping while the metrical structure focuses on rhythmic patterns). Among these structures, we choose the metrical structure as the topic for two reasons: (1) For polyphonic music, the metrical structures of different voices are usually compatible, building up a common song-level metrical structure. This property makes data annotation easier and provides opportunities for self-supervision; (2) Some low-level metrical structures (e.g., beats and downbeats) are already annotated in music scores and most MIDI datasets, which can be a helpful source of supervision. In this paper, we will focus our analysis on pop songs, which often contain a well-defined hierarchy of metrical structures [17].

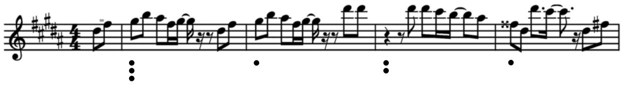

Figure 1. The hierarchical metrical structure of the beginning of RWC-POP No. 001. Only the main melody is shown. Despite the pick-up bar, these 4 measures are grouped into 2 hypermeasures of length 2, which are further grouped into 1 hypermeasure of length 4.

Our main goal is to infer the high-level metrical structures given low-level ones like beats and downbeats. The hierarchy of the metrical structure is created by recursively grouping lower metrical units into upper ones (see Figure 1 for an example). We call the grouping of measures (or other larger metrical units) a *hypermeasure*, and the number of units that form the group as its *hypermeter* [18, 19]. While some properties of beats and measures can be generalized to upper-level metrical structures, there are still many differences to take into account. Similar to the meter, the hypermeter of each layer tends to stay the same to maintain a regular rhythmic pulse, but it is not uncommon to see hypermeter changes in a piece, as shown in Figure 4. Such changes occur more often than low-level meter changes since listeners are less sensitive to long-term rhythmic regularity. Another major difference is that the decision of upper-level metrical structures requires a longer context in the time domain compared to downbeat or beat tracking.

To resolve these issues, we design a new model that contains a Temporal Convolutional Network (TCN) front-end for metrical level prediction and a Conditional Random Field (CRF) decoder for joint metrical structure decoding. We design the transition of CRF hidden states to allow hypermeter changes with some penalties. To han-

[1] https://github.com/music-x-lab/Hierarchical-Metrical-Structure

Attribution: J. Jiang, D. Chin, Y. Zhang and G. Xia, “Learning Hierarchical Metrical Structure Beyond Measures”, in *Proc. of the 23rd Int. Society for Music Information Retrieval Conf.,* Bengaluru, India, 2022.

dle polyphony, the model takes an arbitrary number of voices/tracks as input, and predicts a confidence score for each track. The final prediction is a weighted average of the results from all tracks. We annotated 70 songs from the RWC-POP dataset and used them for model training and evaluation. We conduct experiments under different orchestration setups with results shown in section 4.4.

2. RELATED WORK

2.1 Downbeat and Meter Tracking

Downbeat tracking for raw audio is a well-studied task with many promising results. Krebs et al. [8] use particle filters to find the downbeats, yielding a 2-level metrical structure. Durand et al. [9] use an ensemble of convolutional networks to locate downbeats robustly. Fuentes et al. [10] use skip-chain CRF and deep learning to track downbeats, noting that longer-term musical contexts can better inform downbeat tracking. The reader is referred to Gouyon and Dixon's review [11] for non-symbolic low-level metrical analyzers before 2005.

Notably, locating the beats and downbeats for symbolic music is not a trivial task. Kostek et al. [20] apply neural networks and rough sets to a polyphonic symbolic piece and classify whether each note is accented or not, yielding a 2-level metrical structure (beat and downbeat). Chuang and Su [21] use various RNNs to classify each timestep in a piano roll into non-beat, beat, or downbeat.

Another related task is time signature detection. Benoit [22] uses symbolic-level auto-correlation to obtain a 4-level metrical structure, and ultimately extracts the meter. More auto-correlative methods [23, 24] share a similar logic, since note onsets usually display periodicity in every measure. More recently, inner metric analysis was also used to infer the time signature by Haas et al. [25].

2.2 Music Segmentation

The task of music segmentation is to infer musically meaningful section or part boundaries from the music content. Audio-based music segmentation is usually achieved by detecting similarity or repetition of the audio spectral features. McFee and Ellis [13] evaluate inter-time frame similarity and use spectral clustering to obtain a multi-level segmentation of music. Salamon et al. [14] and McCallum [15] replace traditional features with pre-trained deep embeddings to estimate timbre and harmonic similarity. Tralie and McFee [26] fuse multiple similarity metrics for better prediction results. Ullrich et al. [27] train fully supervised CNN on a segment-annotated dataset to detect segment boundaries. See Dannenberg and Goto's review [28] and the 2010 SOTA report by Paulus et al. [29] to learn more about audio-based music segmentations.

Segmentation of symbolic music relies more on domain knowledge of music composition. Van der Werf and Hendriks [30] restate GTTM grouping rules in terms of Optimality Theory (OT) and design a Prolog program to find an optimal parse for short monophonic pieces. Dai et al. [31] identify phrases as units of melodic repetition by minimizing the Structural Description Length (SDL) for the entire piece, and then extract a 2-layer hierarchy.

2.3 Hierarchical Structure Analysis

The Generative Theory of Tonal Music (GTTM) [16] discusses several views of the hierarchical music structures, but GTTM does not describe how to realize an analyzer computationally. Various efforts have been made to mechanize GTTM. For example, Jones et al. [32] use a rule-based expert system to obtain a 6-level metrical structure for monophonic pieces, satisfying all GTTM's well-formedness rules while following Povel's grid theory [33]. Rosenthal's Machine Rhythm [34] ventures into the polyphonic domain and uses rule-based methods to extract a 3-level rhythmic annotation for MIDI input. Temperley and Sleator [35] use a preference-rule approach and dynamic programming to obtain the optimal parse. Hamanaka et al. [36] propose the Automatic Time-span Tree Analyzer (ATTA) for structural analysis of 8-bar monophonic scores. Temperley [37] use Bayesian reasoning to jointly analyze metrical, harmonic, and stream structures. Wojcik and Kostek [18] use rule-based methods to retrieve hypermetric rhythm from only the melody.

Machine learning models are also used for hierarchical structure analysis. Hamanaka et al. propose Deep-GTTM [38, 39] which is a fully automatic analyzer that uses a neural network to predict the applicability of GTTM rules for each note, yielding a 5-level metrical structure for 8-bar monophonic scores.

There are some issues when applying previous works to large polyphonic MIDI databases. Most systems work only for monophonic music or music with limited polyphony. Also, they often work in a short context, usually up to 8 bars.

3. METHODOLOGY

3.1 Problem Setting

We first formally define the metrical structure prediction task for this paper. Assume we have a music score with T voices (or tracks, in the sense of MIDI files) $\mathbf{m}_1, ..., \mathbf{m}_T$. We also have a list of pre-annotated downbeats $d_1, ..., d_N$ for the music. The aim is to assign hierarchical metrical labels $l_i \in \{0, 1, ..., L\}$ to each downbeat d_i where L is the total number of layers we want to build beyond measures. Each label serves as the level of the metrical boundary at d_i. $l_i = l$ means d_i serves as a metrical boundary of all levels for the first l levels beyond measures. Specially, $l_i = 0$ means d_i serves only as a measure boundary but not any metrical boundary beyond measures.

If we use GTTM's metrical structure notation, we can use (l_i+1) dots to represent a metrical boundary level of l_i. For the example in figure 1, the first 4 measures (excluding the pickup measure) would have labels $l_1 = 2, l_2 = 0, l_3 = 1, l_4 = 0$.

We can also introduce the following notations.

Hypermeasures: A hypermeasure of level l is an interval