MUSPY: A TOOLKIT FOR SYMBOLIC MUSIC GENERATION

Hao-Wen Dong **Ke Chen** **Julian McAuley** **Taylor Berg-Kirkpatrick**
University of California San Diego
{hwdong, knutchen, jmcauley, tberg}@ucsd.edu

ABSTRACT

In this paper, we present MusPy, an open source Python library for symbolic music generation. MusPy provides easy-to-use tools for essential components in a music generation system, including dataset management, data I/O, data preprocessing and model evaluation. In order to showcase its potential, we present statistical analysis of the eleven datasets currently supported by MusPy. Moreover, we conduct a cross-dataset generalizability experiment by training an autoregressive model on each dataset and measuring held-out likelihood on the others—a process which is made easier by MusPy's dataset management system. The results provide a map of domain overlap between various commonly used datasets and show that some datasets contain more representative cross-genre samples than others. Along with the dataset analysis, these results might serve as a guide for choosing datasets in future research. Source code and documentation are available at https://github.com/salu133445/muspy.

1. INTRODUCTION

Recent years have seen progress on music generation, thanks largely to advances in machine learning [1]. A music generation pipeline usually consists of several steps—data collection, data preprocessing, model creation, model training and model evaluation, as illustrated in Figure 1. While some components need to be customized for each model, others can be shared across systems. For symbolic music generation in particular, a number of datasets, representations and metrics have been proposed in the literature [1]. As a result, an easy-to-use toolkit that implements standard versions of such routines could save a great deal of time and effort and might lead to increased reproducibility. However, such tools are challenging to develop for a variety of reasons.

First, though there are a number of publicly-available symbolic music datasets, the diverse organization of these collections and the various formats used to store them presents a challenge. These formats are usually designed for different purposes. Some focus on playback capability

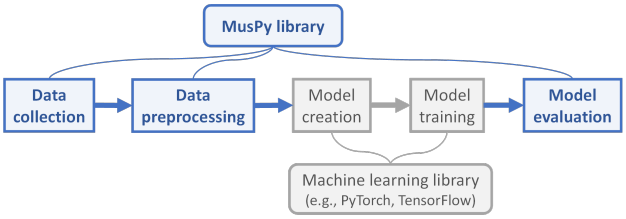


Figure 1. An example of a learning-based music generation system. MusPy provides basic routines specific to music as well as interfaces to machine learning frameworks.

(e.g., MIDI), some are developed for music notation softwares (e.g., MusicXML [2] and LilyPond [3]), some are designed for organizing musical documents (e.g., Music Encoding Initiative (MEI) [4]), and others are research-oriented formats that aim for simplicity and readability (e.g., MuseData [5] and Humdrum [6]. Oftentimes researchers have to implement their own preprocessing code for each different format. Moreover, while researchers can implement their own procedures to access and process the data, issues of reproducibility due to the inconsistency of source data have been raised in [7] for audio datasets.

Second, music has hierarchy and structure, and thus different levels of abstraction can lead to different representations [8]. Moreover, a number of music representations designed specially for generative modeling of music have also been proposed in prior art, for example, as a sequence of pitches [9–12], events [13–16], notes [17] or a time-pitch matrix (i.e., a piano roll) [18, 19].

Finally, efforts have been made toward more robust objective evaluation metrics for music generation systems [20] as these metrics provide not only an objective way for comparing different models but also indicators for monitoring training progress in machine learning-based systems. Given the success of mir_eval [21] in evaluating common MIR tasks, a library providing implementations of commonly used evaluation metrics for music generation systems could help improve reproducibility.

To manage the above challenges, we find a toolkit dedicated for music generation a timely contribution to the MIR community. Hence, we present in this paper a new Python library, MusPy, for symbolic music generation. It provides essential tools for developing a music generation system, including dataset management, data I/O, data preprocessing and model evaluation.

With MusPy, we provide a statistical analysis on the eleven datasets currently supported by MusPy, with an eye

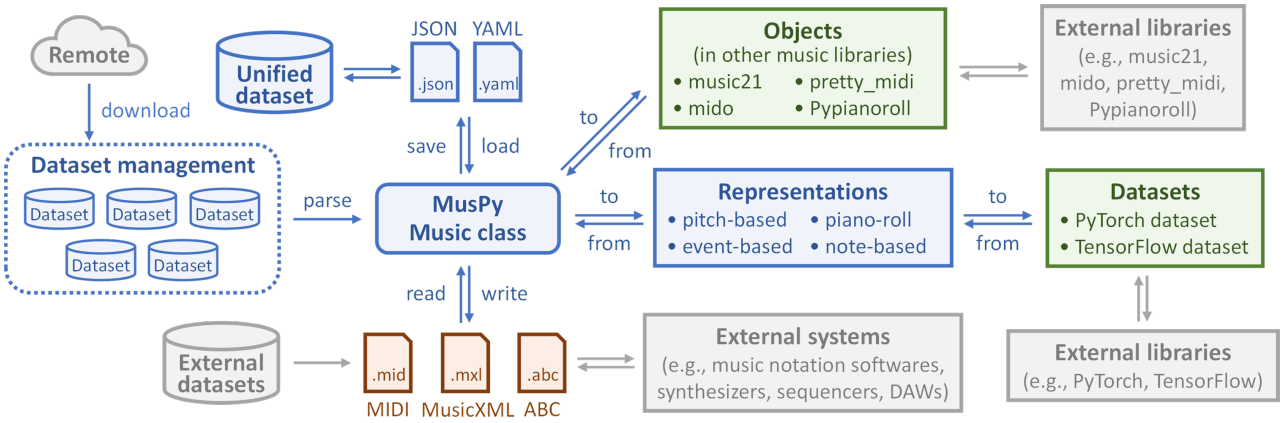


Figure 2. System diagram of MusPy. The MusPy Music object at the center is the core element of MusPy.

to unveiling statistical differences between them. Moreover, we conduct three experiments to analyze their relative diversities and cross-dataset domain compatibility of the various datasets. These results, along with the statistical analysis, together provide a guide for choosing proper datasets for future research. Finally, we also show that combining multiple heterogeneous datasets could help improve generalizability of a music generation system.

2. RELATED WORK

Few attempts, to the best of our knowledge, have been made to develop a dedicated library for music generation. The Magenta project [22] represents the most notable example. While MusPy aims to provide fundamental routines in data collection, preprocessing and analysis, Magenta comes with a number of model instances, but is tightly bound with TensorFlow [23]. In MusPy, we leave the model creation and training to dedicated machine learning libraries, and design MusPy to be flexible in working with different machine learning frameworks.

There are several libraries for working with symbolic music. music21 [24] is one of the most representative toolkits and targets studies in computational musicology. While music21 comes with its own corpus, MusPy does not host any dataset. Instead, MusPy provides functions to download datasets from the web, along with tools for managing different collections, which makes it easy to extend support for new datasets in the future. jSymbolic [25] focuses on extracting statistical information from symbolic music data. While jSymbolic can serve as a powerful feature extractor for training supervised classification models, MusPy focuses on generative modeling of music and supports different commonly used representations in music generation. In addition, MusPy provides several objective metrics for evaluating music generation systems.

Related cross-dataset generalizability experiments [15] show that pretraining on a cross-domain data can improve music generation results both qualitatively and quantitatively. MusPy's dataset management system makes it easier for us to thoroughly verify this hypothesis by examining pairwise generalizabilities between various datasets.

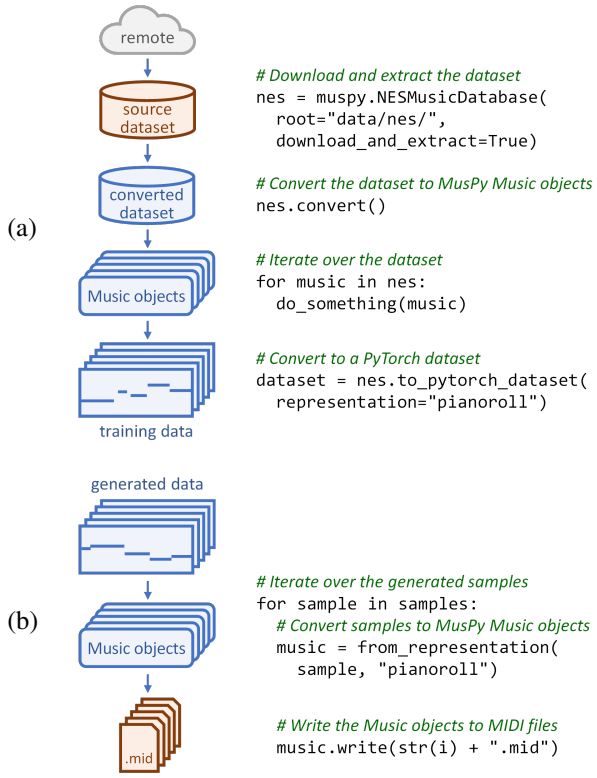


Figure 3. Examples of (a) training data preparation and (b) result writing pipelines using MusPy.

3. MUSPY

MusPy is an open source Python library dedicated for symbolic music generation. Figure 2 presents the system diagram of MusPy. It provides a core class, MusPy Music class, as a universal container for symbolic music. Dataset management system, I/O interfaces and model evaluation tools are then built upon this core container. We provide in Figure 3 examples of data preparation and result writing pipelines using MusPy.

3.1 MusPy Music class and I/O interfaces

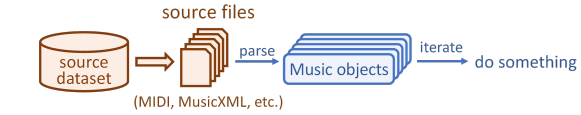
We aim at finding a middle ground among existing formats for symbolic music and design a unified format dedicated

Dataset	Format	Hours	Songs	Genre	Melody	Chords	Multitrack
Lakh MIDI Dataset (LMD) [26]	MIDI	>9000	174,533	misc	△	△	△
MAESTRO Dataset [27]	MIDI	201.21	1,282	classical			
Wikifonia Lead Sheet Dataset [28]	MusicXML	198.40	6,405	misc	✓	✓	
Essen Folk Song Database [29]	ABC	56.62	9,034	folk	✓	✓	
NES Music Database [30]	MIDI	46.11	5,278	game	✓		✓
Hymnal Tune Dataset [31]	MIDI	18.74	1,756	hymn	✓		
Hymnal Dataset [31]	MIDI	17.50	1,723	hymn			
music21 Corpus [24]	misc	16.86	613	misc	△		△
Nottingham Database (NMD) [32]	ABC	10.54	1,036	folk	✓	✓	
music21 JSBach Corpus [24]	MusicXML	3.46	410	classical			✓
JSBach Chorale Dataset [11]	MIDI	3.21	382	classical			✓

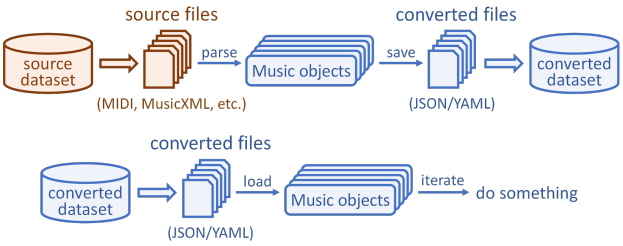
Table 1. Comparisons of datasets currently supported by MusPy. Triangle marks indicate partial support. Note that, in this version, only MusicXML and MIDI files are included for the music21 Corpus.

	MIDI	MusicXML	MusPy
Sequential timing	✓		✓
Playback velocities	✓	△	✓
Program information	✓	△	✓
Layout information		✓	
Note beams and slurs		✓	
Song/source meta data	△	✓	✓
Track/part information	△	✓	✓
Dynamic/tempo markings		✓	✓
Concept of notes		✓	✓
Measure boundaries		✓	✓
Human readability		△	✓

Table 2. Comparisons of MIDI, MusicXML and the proposed MusPy formats. Triangle marks indicate optional or limited support.



(a) on-the-fly mode



(b) preconverted mode

Figure 4. Two internal processing modes for iterating over a MusPy Dataset object.

for music generation. MIDI, as a communication protocol between musical devices, uses velocities to indicate dynamics, beats per minute (bpm) for tempo markings, and control messages for articulation, but it lacks the concepts of notes, measures and symbolic musical markings. In contrast, MusicXML, as a sheet music exchanging format, has the concepts of notes, measures and symbolic musical markings and contains visual layout information, but it falls short on playback-related data. For a music generation system, however, both symbolic and playback-specific data are important. Hence, we follow MIDI's standard for playback-related data and MusicXML's standard for symbolic musical markings.

In fact, the MusPy Music class naturally defines a universal format for symbolic music, which we will refer to as the MusPy format, and can be serialized into a human-readable JSON/YAML file. Table 2 summarizes the key differences among MIDI, MusicXML and the proposed MusPy formats. Using the proposed MusPy Music class as the internal representation for music data, we then provide I/O interfaces for common formats (e.g., MIDI, MusicXML and ABC) and interfaces to other symbolic music libraries (e.g., music21 [24], mido [33], pretty_midi [34] and Pypianoroll [35]). Figure 3(b) provides an example of result writing pipeline using MusPy.

3.2 Dataset management

MusPy provides an easy-to-use dataset management system similar to torchvision datasets [36] and TensorFlow Dataset [37]. Table 1 presents the list of datasets currently supported by MusPy and their comparisons. Each supported dataset comes with a class inherited from the base MusPy Dataset class. The modularized and flexible design of the dataset management system makes it easy to handle local data collections or extend support for new datasets in the future. Figure 4 illustrates the two internal processing modes when iterating over a MusPy Dataset object. In addition, MusPy provides interfaces to PyTorch [38] and TensorFlow [23] for creating input pipelines for machine learning (see Figure 3(a) for an example).

3.3 Representations

Music has multiple levels of abstraction, and thus can be expressed in various representations. For music generation in particular, several representations designed for

Representation	Shape	Values	Default configurations
Pitch-based	$T \times 1$	$\{0, 1, \ldots, 129\}$	128 note-ons, 1 hold, 1 rest (*support only monophonic music*)
Event-based	$T \times 1$	$\{0, 1, \ldots, 387\}$	128 note-ons, 128 note-offs, 100 time shifts, 32 velocities
Piano-roll	$T \times 128$	$\{0, 1\}$ or $\mathbb{R}^+$	$\{0, 1\}$ for binary piano rolls; $\mathbb{R}^+$ for piano rolls with velocities
Note-based	$N \times 4$	$\mathbb{N}$ or $\mathbb{R}^+$	List of $(time, pitch, duration, velocity)$ tuples

Table 3. Comparisons of representations supported by MusPy. T and N denote the numbers of time steps and notes, respectively. Note that the configurations can be modified to meet specific requirements and use cases.

generative modeling of symbolic music have been proposed and used in the literature [1]. These representations can be broadly categorized into four types—the pitch-based [9–12], the event-based [13–16], the note-based [17] and the piano-roll [18,19] representations. Table 3 presents a comparison of them. We provide in MusPy implementations of these representations and integration to the dataset management system. Figure 3(a) provides an example of preparing training data in the piano-roll representation from the NES Music Database using MusPy.

3.4 Model evaluation tools

Model evaluation is another critical component in developing music generation systems. Hence, we also integrate into MusPy tools for audio rendering as well as score and piano-roll visualizations. These tools could also be useful for monitoring the training progress or demonstrating the final results. Moreover, MusPy provides implementations of several objective metrics proposed in the literature [17, 19, 39]. These objective metrics, as listed below, could be used to evaluate a music generation system by comparing the statistical difference between the training data and the generated samples, as discussed in [20].

- *Pitch-related metrics*—polyphony, polyphony rate, pitch-in-scale rate, scale consistency, pitch entropy and pitch class entropy.
- *Rhythm-related metrics*—empty-beat rate, drum-in-pattern rate, drum pattern consistency and groove consistency.

3.5 Summary

To summarize, MusPy features the following:

- Dataset management system for commonly used datasets with interfaces to PyTorch and TensorFlow.
- Data I/O for common symbolic music formats (e.g., MIDI, MusicXML and ABC) and interfaces to other symbolic music libraries (e.g., music21, mido, pretty_midi and Pypianoroll).
- Implementations of common music representations for music generation, including the pitch-based, the event-based, the piano-roll and the note-based representations.
- Model evaluation tools for music generation systems, including audio rendering, score and piano-roll visualizations and objective metrics.

All source code and documentation can be found at `https://github.com/salu133445/muspy`.

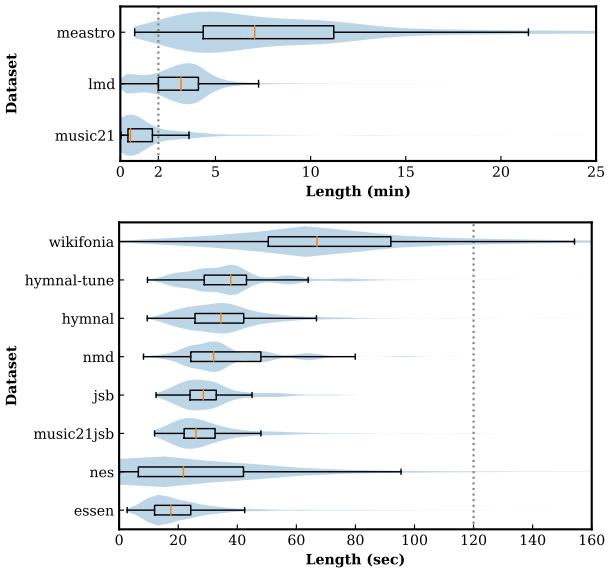


Figure 5. Length distributions for different datasets.

4. DATASET ANALYSIS

Analyzing datasets is critical in developing music generation systems. With MusPy's dataset management system, we can easily work with different music datasets. Below we compute the statistics of three key elements of a song—length, tempo and key using MusPy, with an eye to unveiling statistical differences among these datasets. First, Figure 5 shows the distributions of song lengths for different datasets. We can see that they differ greatly in their ranges, medians and variances.

Second, we present in Figure 6 the distributions of initial tempo for datasets that come with tempo information. We can see that all of them are generally bell-shaped but with different ranges and variances. We also note that there are two peaks, 100 and 120 quarter notes per minute (qpm), in Lakh MIDI Dataset (LMD), which is possibly because these two values are often set as the default tempo values in music notation programs and MIDI editors/sequencers. Moreover, in Hymnal Tune Dataset, only around ten percent of songs have an initial tempo other than 100 qpm.

Finally, Figure 7 shows the histograms of keys for different datasets. We can see that the key distributions are rather imbalanced. Moreover, only less than 3% of songs are in minor keys for most datasets except the music21 Corpus. In particular, LMD has the most imbalanced key distributions, which might be due to the fact that C major is

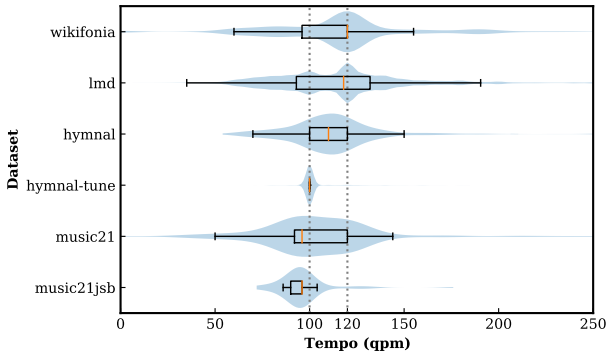


Figure 6. Initial-tempo distributions for different datasets (those without tempo information are not presented).

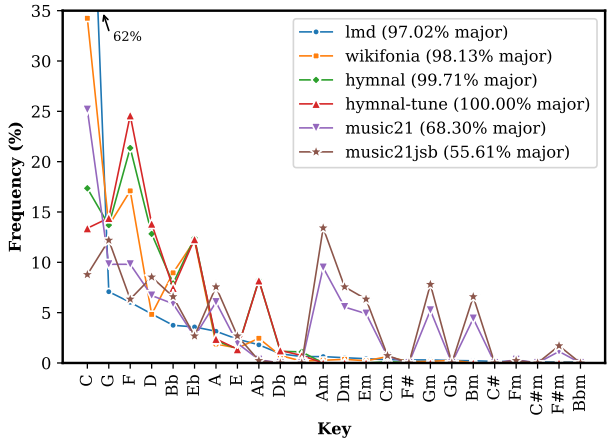


Figure 7. Key distributions for different datasets. The keys are sorted w.r.t. their frequencies in Lakh MIDI Dataset.

often set as the default key in music notation programs and MIDI editors/sequencers. [1] These statistics could provide a guide for choosing proper datasets in future research.

5. EXPERIMENTS AND RESULTS

In this section, we conduct three experiments to analyze the relative complexities and the cross-dataset generalizabilities of the eleven datasets currently supported by MusPy (see Table 1). We implement four autoregressive models—a recurrent neural network (RNN), a long short-term memory (LSTM) network [40], a gated recurrent unit (GRU) network [41] and a Transformer network [42].

5.1 Experiment settings

For the data, we use the event representation as specified in Table 3 and discard velocity events as some datasets have no velocity information (e.g., datasets using ABC format). Moreover, we also include an end-of-sequence event, leading to in total 357 possible events. For simplicity, we downsample each song into four time steps per quarter note and fix the sequence length to 64, which is equivalent to four measures in $4/4$ time. In addition, we discard repeat information in MusicXML data and use only melodies in Wikifonia dataset. We split each dataset into train–test–validation sets with a ratio of $8:1:1$. For the training, the models are trained to predict the next event given the previous events. We use the cross entropy loss and the Adam optimizer [43]. For evaluation, we randomly sample 1000 sequences of length 64 from the test split, and compute the perplexity of these sequences. We implement the models in Python using PyTorch. For reproducibility, source code and hyperparmeters are available at `https://github.com/salu133445/muspy-exp`.

[1] Note that key information is considered as a meta message in a MIDI file. It does not affect the playback and thus can be unreliable sometimes.

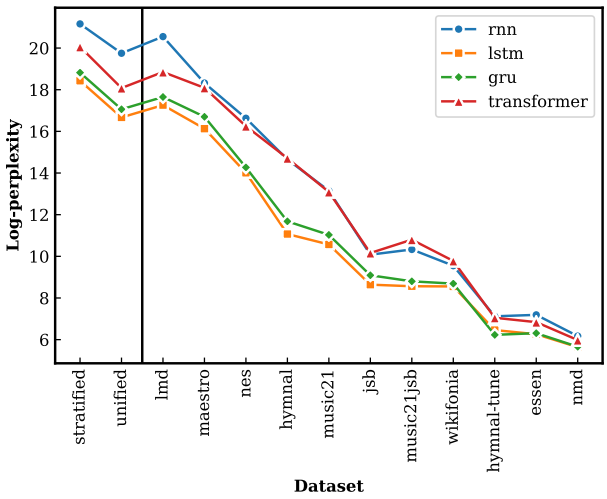


Figure 8. Log-perplexities for different models on different datasets, sorted by the values for the LSTM model.

5.2 Autoregressive models on different datasets

In this experiment, we train the model on some dataset $\mathcal{D}$ and test it on the same dataset $\mathcal{D}$. We present in Figure 8 the perplexities for different models on different datasets. We can see that all models have similar tendencies. In general, they achieve smaller perplexities for smaller, homogeneous datasets, but result in larger perplexities for larger, more diverse datasets. That is, the test perplexity could serve as an indicator for the diversity of a dataset. Moreover, Figure 9 shows perplexities versus dataset sizes (in hours). By categorizing datasets into multi-pitch (i.e., accepting any number of concurrent notes) and monophonic datasets, we can see that the perplexity is positively correlated to the dataset size within each group.

5.3 Cross-dataset generalizability

In this experiment, we train a model on some dataset $\mathcal{D}$, while in addition to testing it on the same dataset $\mathcal{D}$, we also test it on each other dataset $\mathcal{D}'$. We present in Figure 10 the perplexities for each train–test dataset pair. Here are some observations:

- Cross dataset generalizability is not symmetric in general. For example, a model trained on LMD generalizes well to all other datasets, while not all models trained on

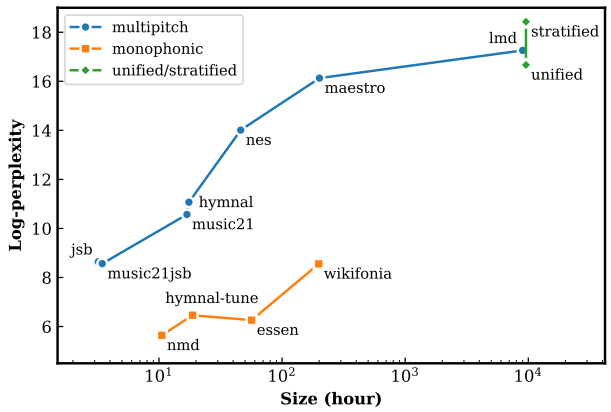


Figure 9. Log-perplexities for the LSTM model versus dataset size in hours. Each point corresponds to a dataset.

other datasets generalize to LMD, which is possibly due to the fact that LMD is a large, cross-genre dataset.

- Models trained on multi-pitch datasets generalize well to monophonic datasets, while models trained on monophonic datasets do not generalize to multi-pitch datasets (see the red block in Figure 10).
- The model trained on JSBach Chorale Dataset does not generalize to any of the other datasets (see the orange block in Figure 10). This is possibly because its samples are downsampled to a resolution of quarter note, which leads to a distinct note duration distribution.
- Most datasets generalize worse to NES Music Database compared to other datasets (see the green block in Figure 10). This is possibly due to the fact that NES Music Database contains only game soundtracks.

5.4 Effects of combining heterogeneous datasets

From Figure 10 we can see that LMD has the best generalizability, possibly because it is large, diverse and cross-genre. However, a model trained on LMD does not generalize well to NES Music Database (see the brown block in the close-up of Figure 10). We are thus interested in whether combing multiple heterogeneous datasets could help improve generalizability.

We combine all eleven datasets listed in Table 1 into one large *unified* dataset. Since these datasets differ greatly in their sizes, simply concatenating the datasets might lead to severe imbalance problem and bias toward the largest dataset. Hence, we also consider a version that adopts stratified sampling during training. Specifically, to acquire a data sample in the *stratified* dataset, we uniformly choose one dataset out of the eleven datasets, and then randomly pick one sample from that dataset. Note that stratified sampling is disabled at test time.

We also include in Figures 8, 9 and 10 the results for these two datasets. We can see from Figure 10 that combining datasets from different sources improves the generalizability of the model. This is consistent with the finding in [15] that models trained on certain cross-domain

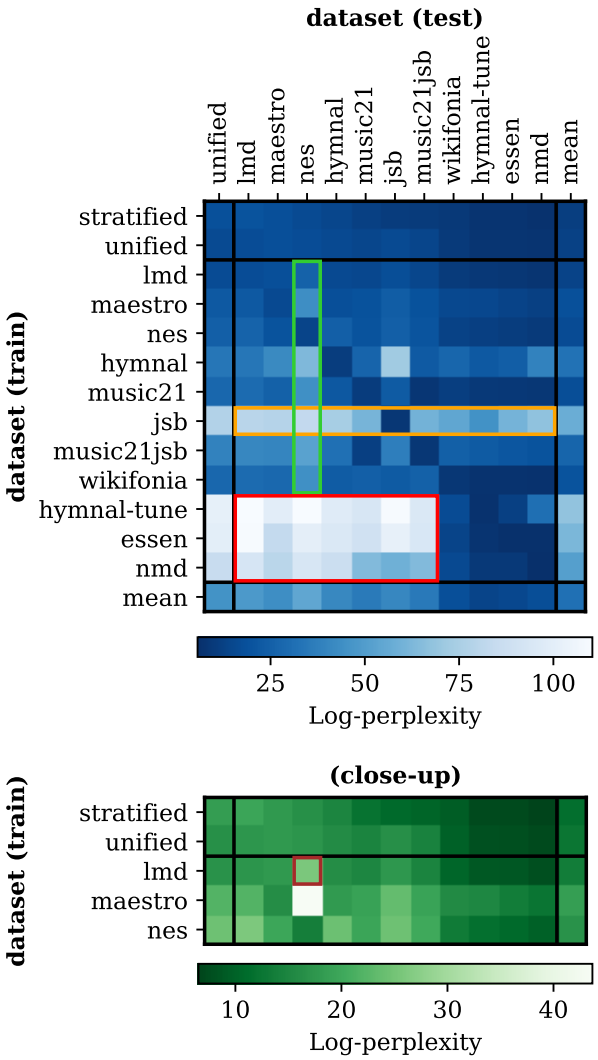


Figure 10. Cross-dataset generalizability results. The values and colors represent the log-perplexities of a LSTM model trained on a specific dataset (row) and tested on another dataset (column). The datasets are sorted by the diagonal values, i.e., trained and tested on the same dataset.

datasets generalize better to other unseen datasets. Moreover, stratified sampling alleviates the source imbalance problem by reducing perplexities in most datasets with a sacrifice of an increased perplexity on LMD.

6. CONCLUSION

We have presented MusPy, a new toolkit that provides essential tools for developing music generation systems. We discussed the designs and features of the library, along with data pipeline examples. With MusPy's dataset management system, we conducted a statistical analysis and experiments on the eleven currently supported datasets to analyze their relative diversities and cross-dataset generalizabilities. These results could help researchers choose appropriate datasets in future research. Finally, we showed that combining heterogeneous datasets could help improve generalizability of a machine learning model.

7. REFERENCES

[1] J.-P. Briot, G. Hadjeres, and F. Pachet, "Deep learning techniques for music generation: A survey," *arXiv preprint arXiv:1709.01620*, 2017.

[2] M. Good, "Musicxml for notation and analysis," in *The Virtual Score: Representation, Retrieval, Restoration*, W. B. Hewlett and E. Selfridge-Field, Eds. Cambridge, Massachusetts: MIT Press, 2001, ch. 8, pp. 113–124.

[3] "Lilypond," https://lilypond.org/.

[4] A. Hankinson, P. Roland, and I. Fujinaga, "The music encoding initiative as a document-encoding framework," in *Proc. of the 12th International Society for Music Information Retrieval Conference (ISMIR)*, 2011.

[5] W. B. Hewlett, "MuseData: Multipurpose representation," in *Beyond MIDI: The Handbook of Musical Codes*, E. Selfridge-Field, Ed. Cambridge, Massachusetts: MIT Press, 1997, ch. 27, pp. 402–447.

[6] D. Huron, "Humdrum and Kern: Selective feature encoding," in *Beyond MIDI: The Handbook of Musical Codes*, E. Selfridge-Field, Ed. Cambridge, Massachusetts: MIT Press, 1997, ch. 27, pp. 375–401.

[7] R. M. Bittner, M. Fuentes, D. Rubinstein, A. Jansson, K. Choi, and T. Kell, "mirdata: Software for reproducible usage of datasets," in *Proc. of the 20th International Society for Music Information Retrieval Conference (ISMIR)*, 2019.

[8] R. B. Dannenberg, "A brief survey of music representation issues, techniques, and systems," *Computer Music Journal*, vol. 17, no. 3, pp. 20–30, 1993.

[9] M. Mozer, "Neural network music composition by prediction: Exploring the benefits of psychoacoustic constraints and multi-scale processing," *Connection Science*, vol. 6, pp. 247–280, 1994.

[10] D. Eck and J. Schmidhuber, "Finding temporal structure in music: Blues improvisation with LSTM recurrent networks," in *Proc. of the IEEE Workshop on Neural Networks for Signal Processing*, 2002, pp. 747–756.

[11] N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent, "Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription," in *Proc. of the 29th International Conference on Machine Learning (ICML)*, 2012.

[12] A. Roberts, J. Engel, C. Raffel, C. Hawthorne, and D. Eck, "A hierarchical latent vector model for learning long-term structure in music," in *Proc. of the 35th International Conference on Machine Learning (ICML)*, 2018.

[13] S. Oore, I. Simon, S. Dieleman, D. Eck, and K. Simonyan, "This time with feeling: Learning expressive musical performance," *Neural Computing and Applications*, vol. 32, 2018.

[14] C.-Z. A. Huang, A. Vaswani, J. Uszkoreit, I. Simon, C. Hawthorne, N. Shazeer, A. M. Dai, M. D. Hoffman, M. Dinculescu, and D. Eck, "Music transformer: Generating music with long-term structure," in *Proc. of the 7th International Conference for Learning Representations (ICLR)*, 2019.

[15] C. Donahue, H. H. Mao, Y. E. Li, G. W. Cottrell, and J. McAuley, "Lakhnes: Improving multi-instrumental music generation with cross-domain pre-training," in *Proc. of the 20th International Society for Music Information Retrieval Conference (ISMIR)*, 2019.

[16] Y.-S. Huang and Y.-H. Yang, "Pop music transformer: Generating music with rhythm and harmony," *arXiv preprint arXiv:2002.00212*, 2020.

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

[18] L.-C. Yang, S.-Y. Chou, and Y.-H. Yang, "Midinet: A convolutional generative adversarial network for symbolic-domain music generation," in *Proc. of the 18th International Society for Music Information Retrieval Conference (ISMIR)*, 2017.

[19] H.-W. Dong, W.-Y. Hsiao, L.-C. Yang, and Y.-H. Yang, "MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment," in *Proc. of the 32nd AAAI Conference on Artificial Intelligence (AAAI)*, 2018.

[20] L.-C. Yang and A. Lerch, "On the evaluation of generative models in music," *Neural Computing and Applications*, vol. 32, pp. 4773–4784, 2018.

[21] C. Raffel, B. McFee, E. J. Humphrey, J. Salamon, O. Nieto, D. Liang, and D. P. W. Ellis, "mir_eval: A transparent implementation of common MIR metrics," in *Proc. of the 15th International Society for Music Information Retrieval Conference (ISMIR)*, 2014.

[22] "Magenta," https://magenta.tensorflow.org/.

[23] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: A system for large-scale machine learning," in *Proc. of the 12th USENIX Symp. on Operating Systems Design and Implementation (OSDI)*, 2016.

[24] M. S. Cuthbert and C. Ariza, "Music21: A toolkit for computer-aided musicology and symbolic music data," in *Proc. of the 11th International Society for Music Information Retrieval Conference (ISMIR)*, 2010.

[25] C. Mckay and I. Fujinaga, "JSymbolic: A feature extractor for MIDI files," in *Proc. of the 2006 International Computer Music Conference (ICMC)*, 2006.

[26] C. Raffel, "Learning-based methods for comparing sequences, with applications to audio-to-MIDI alignment and matching," Ph.D. dissertation, Columbia University, 2016.

[27] 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 *Proc. of the 7th International Conference on Learning Representations (ICLR)*, 2019.

[28] "Wikifonia," http://www.wikifonia.org/.

[29] "Essen folk song database," https://ifdo.ca/~seymour/runabc/esac/esacdatabase.html.

[30] C. Donahue, H. H. Mao, and J. McAuley, "The NES music database: A multi-instrumental dataset with expressive performance attributes," in *Proc. of the 19th International Society for Music Information Retrieval Conference (ISMIR)*, 2018.

[31] "Hymnal," https://www.hymnal.net/.

[32] "Nottingham database," https://ifdo.ca/~seymour/nottingham/nottingham.html.

[33] "Mido: Midi objects for python," https://github.com/mido/mido.

[34] C. Raffel and D. P. W. Ellis, "Intuitive analysis, creation and manipulation of MIDI data with pretty_midi," in *Late-Breaking Demos of the 15th International Society for Music Information Retrieval Conference (ISMIR)*, 2014.

[35] H.-W. Dong, W.-Y. Hsiao, and Y.-H. Yang, "Pypianoroll: Open source Python package for handling multitrack pianorolls," in *Late-Breaking Demos of the 19th International Society for Music Information Retrieval Conference (ISMIR)*, 2018.

[36] S. Marcel and Y. Rodriguez, "Torchvision the machine-vision package of torch," in *Proc. of the 18th ACM International Conference on Multimedia*, 2010.

[37] "Tensorflow datasets," https://www.tensorflow.org/datasets.

[38] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32 (NeurIPS)*, 2019, pp. 8024–8035.

[39] S.-L. Wu and Y.-H. Yang, "The jazz transformer on the front line: Exploring the shortcomings of ai-composed music through quantitative measures," in *Proc. of the 21st International Society for Music Information Retrieval Conference (ISMIR)*, 2020.

[40] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[41] K. Cho, B. van Merrienboer, C. Gulcehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," in *Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[42] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems 30 (NeurIPS)*, 2017.

[43] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. of the 3rd International Conference for Learning Representations (ICLR)*, 2014.