International Conference - Innovation-2021-Innovation-2021
International Journal of Scientific Research in Computer Science, Engineering and Information Technology ISSN : 2456-3307 (www.ijsrcseit.com)
Volume 8, Issue 3, May-June-2021

Development of Symbolic Music Generation Technique Based on Deep Learning and AI

Vincy Kaushik[1], Pravin Kumar Mishra[2]
[1]Bharat Institute of Technology, Meerut, Uttar Pradesh, India
[2]Assistant Professor, Bharat Institute of Technology, Meerut, Uttar Pradesh, India

ABSTRACT

In this work we propose MusPy, a Python open source toolkit for the creation of symbolic music. MusPy provides easy to use tools for key music generating components like dataset administration, data I/O, data preparation, and model assessment. We offer the statistical analysis of the eleven presently supported MusPy datasets to demonstrate their potential. Moreover, by training an autoregressive model on each dataset, we undertake a cross-data generalisation experience and measure the likelihood of the rest — a process made easy by a MusPy dataset management system. The results reveal a domain map that overlaps different frequently used data sets with more cross-gender examples in some data sets than in other. These results might serve as a reference for selecting data sets in future study, alongside the examination of data sets.

Keywords : Symbolic Music Generation, AI, Deep Learning, MIDI

I. INTRODUCTION

As shown in Fig. 1, a pipeline for music creation typically consists of the following steps: data collection, data preprocessing, model building, model training, and model evaluation. While certain components must be customised for each model, others may be shared across many systems. Numerous data sets, representations, and metrics have been given in the literature, most notably for the creation of symbolic music[1]. This may save much time and effort and may result in increased repeatability when a basic toolkit implementing standard versions of such processes is used. However, such tools are challenging to develop for a number of reasons.

To begin, although a large number of symbolic music datasets are publicly available, organising these collections and preserving the many formats used to capture them is difficult. These formats are often used for a variety of reasons. Some are developed for replay capabilities (e.g., MIDI), while others are developed to support the Music Encoding Initiative (MEI)[4]. Others are developed to support research-oriented formats aimed at simplicity and readability (e.g., MuseData [5] and Humdrum [6]), such as Music XML[2] and LilyPond[3], as well as the Music Encoding Initiative. Often, researchers must write their own code for each preprocessing format. Frequently, researchers must write their own preparation code for each format. While researchers may develop their own data access and processing methods, issues of repeatability have been highlighted in [7] for audio datasets because to a lack of coherence in the raw data.

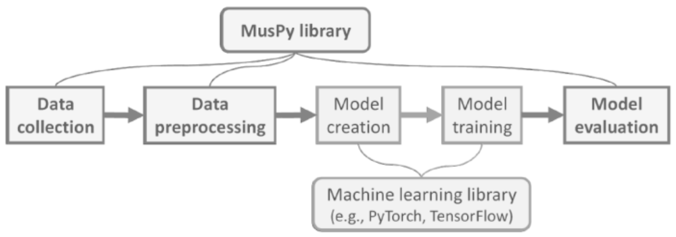


Fig. 1. An example of a method for learning music

Second, the structure and hierarchy of music can lead to diverse depictions of abstraction[8]. Further, in the context of a previous art, certain music performances were also offered as a sequence of pitch [9-12], events [13-16], notes [17] or a matrix of time pitch (i.e. a piano rollen).

Finally, attempts were undertaken to make objective assessment measurements for music production systems more robust[17], since these measurements give an objective approach to compare various models but also to monitor development of training on machine-based learning systems. Due to mireval effectiveness in evaluating typical MIR tasks, a library that implements frequently used assessment measurement methods for systems generating music might assist enhance productiveness.

We find a toolset for generating music that contributes to the MIR community in due course to meet these problems. So in this paper we offer a new Python library, MusPy, for the creation of symbolic music. It offers fundamental instruments for building a system for generating music, including data set management, I/O data, preprocessing and model assessment.

We do a statistical analysis of the 11 presently supported data sets with MusPy, in order to detect statistical discrepancies. Furthermore, we perform three experiments to assess their relative diversity and the compatibility of different data sets with each other. Together with the statistical analysis, these results give a guidance for selecting correct data sets for future study. Finally, we also show that combining multiple heterogeneous datasets could help improve generalizability of a music generation system.

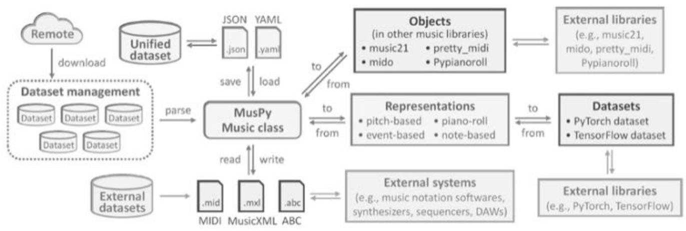


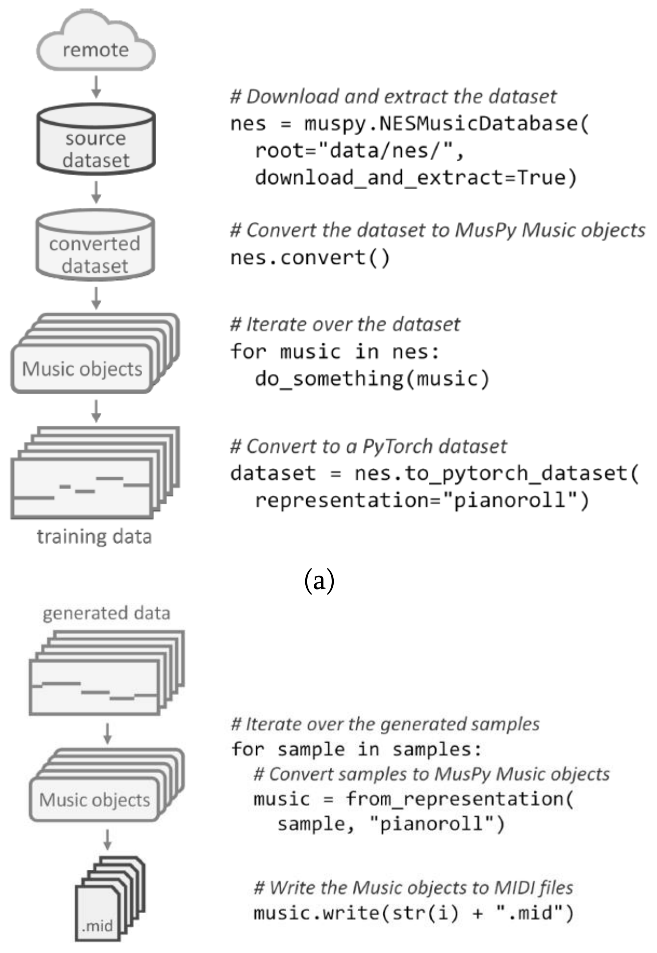
Fig. 2. MusPy Music object at the center is the core element of MusPy.

II. RELATED WORK

A specialised library for music generation has been developed with little effort, to the best of our knowledge. The most remarkable example is the Magenta project[5]. While MusPy seeks to offer core data collecting, pre-processing, and analysis procedures, Magenta has several model instances, but it is very closely linked with TensorFlow[23]. We at MusPy allow specialised machine learning libraries to design the model generation and training and to make MusPy adaptable in the work with multiple machine learning frameworks.

Many libraries are available to deal with symbolic music. Music21[4] is one of the most representative instruments of computer musicology research and objectives. Music21 has its own corpus, while MusPy has no dataset. Instead, MusPy provides online downloading data sets with tools to manage distinct collections, so that new datasets may be extended in future easily. jSymbolic [5] is focusing on extracting symbolic music statistical information. While jSymbolic may serve as a powerful feature extractor for the training of classification models supervised, MusPy concentrates on generative music modelling and promotes many common representations in the production of music.

Furthermore, MusPy offers numerous objective measurements for assessing systems for music production. Connected generalisation studies using cross-datasets[5] indicate that cross-domain pretraining can both qualitatively and quantitatively improve results of music creation. MusPy's data set management mechanism facilitates us in analysing the generalizability of different datasets in pairs to properly check this hypothesis.



```
# Download and extract the dataset
nes = muspy.NESMusicDatabase(
  root="data/nes/",
  download_and_extract=True)

# Convert the dataset to MusPy Music objects
nes.convert()

# Iterate over the dataset
for music in nes:
  do_something(music)

# Convert to a PyTorch dataset
dataset = nes.to_pytorch_dataset(
  representation="pianoroll")
```

```
# Iterate over the generated samples
for sample in samples:
  # Convert samples to MusPy Music objects
  music = from_representation(
    sample, "pianoroll")

  # Write the Music objects to MIDI files
  music.write(str(i) + ".mid")
```

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

III. ABOUT THE LIBRARY MUSPY

MusPy is a Python package open source for the creation of symbolic music. The system diagram of MusPy is shown in Fig. 2. It offers a core class, MusPy Music class, as a universal symbolic container. This core container is then incorporated into the data set management system, I/O interfaces and model assessment tools. In Fig. 3, we offer examples of data preparation and pipelines with MusPy.

A. *Muspy Music Class And I/O Interfaces*

Our goal is to establish a medium ground between existing symbolic music forms, and to build a unified music generating format. MIDI employs speeds for the transfer of dynamics, bpm for tempo markings and control messages for articulation as part of the protocol for communication between musical instruments. The notions of notes, measurements, and symbolic musical markings are missing, however. MusicXML instead incorporates the notion of notes, measures, and symbolic musical markers as a sheet music exchange format and provides layout information, but it does not include the reproduction data. It also has visual layout information. But symbolic and playback data are crucial for a music generating system. Thus, we follow the playback data standard of MIDI and MusicXML's symbolic music marking standard.

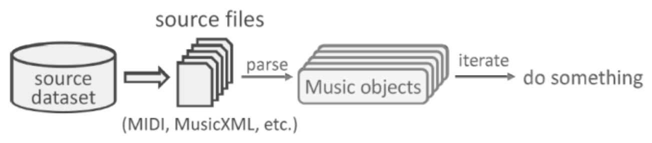
Table 1. Currently supported comparisons of MusPy datasets. The markings of triangle show partial backing. Note that just MusicXML and MIDI files for music21 Corpus have been provided in this version.

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			✓

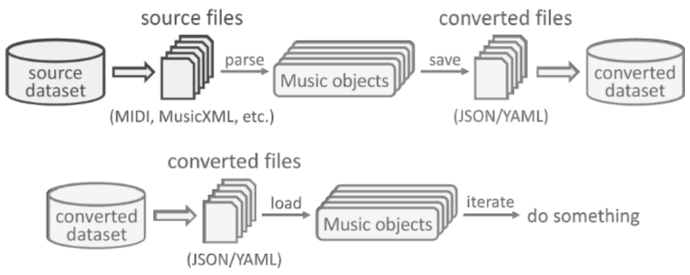
Table 2. Comparisons of MIDI, MusicXML and the proposed MusPy formats. Triangle marks indicate optional or limited support.

	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		△	✓

In reality, a universal format for symbolic music that we call MusPy is automatically defined in the MusPy Music Class and can be transformed into a JSON/YAML human-reading file. Table 2 outlines the significant variations between MIDI, MusicXML and the MusPy formats suggested. We will offer an interface to existing symbolic music libraries, e.g. music21(24), mido[23], pretty midi[33] and Pypianoroll[35], using the proposed MusPy Music Class as the internal representation for music data. The result pipeline utilising MusPy is shown in Fig. 3(b).



(a) on-the-fly mode



(b) preconverted mode

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

B. Dataset Management

Similar to Torchvision datasets[36] and TensorFlow data set[37], MusPy provides an easy-to-use framework for data set management. Table 1 lists and compares of the datasets that MusPy presently supports. An inherited class from the MusPy Dataset base is provided with each supported dataset. The modular and flexible architecture of the dataset management system simplifies the maintenance of local data collections or the future extension of support for additional datasets. In the iteration via MusPy data object, Fig. 4 shows two internal processing modes. MusPy also has PyTorch[38] and TensorFlow[23] interface APIs for the creation of machine-learning input pipelines.

C. Representations

Music has several abstract levels, and may thus be stated in several forms. Several representations have been suggested and utilised in literature for the production of Music in particular, intended for the generative modelling of symbolic music[1]. These portrays may be roughly classified into four kinds — pitch-based representations [9-12], event-based representations [13-016], note-based representations [17] and piano-rolls [18,19].

Table 3. T and N indicate number of timescales and notes, respectively. Comparisons supporting MusPy. Note that you can modify settings to comply with unique needs and use scenarios.

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

A comparison of them is provided in Table 3. We develop these representations in MusPy and integrate them into the dataset management system. Fig. 3(a) shows an example of how training data are prepared in the NES Music Database piano-roll format using MusPy.

D. Model Evaluation Tools

Another important component of creating music generating systems is model assessment. As a result, we incorporate MusPy's audio rendering and score and piano-roll visualisation capabilities. Additionally, these tools may be beneficial for evaluating training progress and presenting final outcomes. Additionally, MusPy implements a number of objective measures described in the literature [17]. As described in [14], these objective metrics may be used to assess a music generating system by analysing the statistical difference between the training and produced samples. Polyphony, polyphony rate, pitch-in-scale rate, scale consistency, pitch entropy, and pitch class entropy are all pitch-related measures. Empty-beat rate, drum inpattern rate, drum pattern consistency, and groove consistency are all rhythm-related measures.

E. Summary

To summarize, MusPy features the following:

Using PyTorch and TensorFlow interfaces, this data management solution manages commonly used datasets. Interfaces to a variety of symbolic music libraries (for example, music21, mido, beautiful midi, and Pypianoroll) as well as data I/O for popular symbolic music formats (e.g., MIDI, MusicXML, and ABC). Implementations of commonly used musical notation systems, including pitch-based, event-based, piano-roll, and note-based notation. Instruments for assessing music generation models, including audio rendering, score and piano-roll visualisations, and objective metrics..

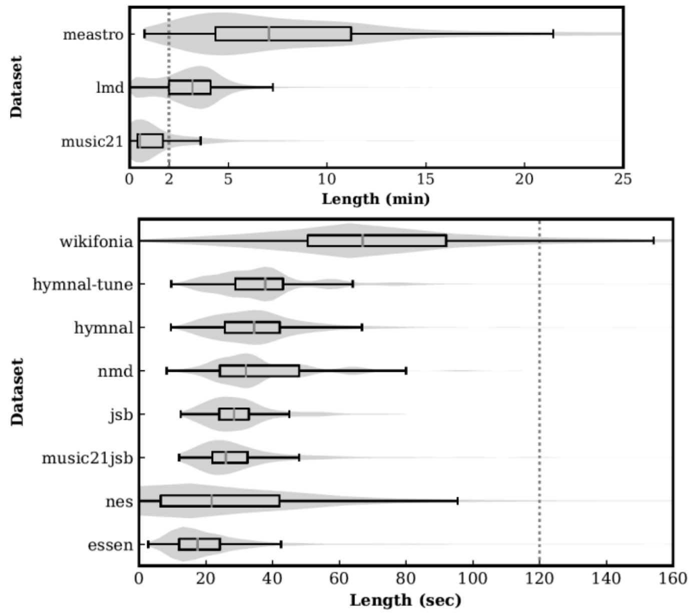


Fig. 5. Length distributions for different datasets.

IV. DATASET ANALYSIS

Analyzing datasets is important for creating systems for musical composition. We can simply deal with various music datasets using MusPy's dataset management framework. We use MusPy to calculate statistics for three critical aspects of a song—length, tempo, and key—with the goal of revealing statistical disparities across these datasets. To begin, Fig. 5 illustrates the distributions of song durations for various datasets. As we can see, their ranges, medians, and variances are very different.

Second, we show in Fig. 6 the initial tempo distributions for datasets that have tempo information. As can be seen, they are all essentially bell-shaped, although with varying ranges and variations. Additionally, we see two peaks in the Lakh MIDI Dataset (LMD), at 100 and 120 quarter notes per minute (qpm), which may be due to the fact that these two numbers are often selected as the default tempo settings in music notation applications and MIDI editors/sequencers. Additionally, only about ten percent of songs in the Hymnal Tune Dataset begin with a tempo other than 100 qpm.

Finally, Fig. 7 depicts the important histograms for various datasets. As may be seen, the key distributions are rather asymmetric. Furthermore, with the exception of the music21 Corpus, fewer than 3% of songs are in minor keys. LMD has the most skewed key distributions, which may be because C major is often selected as the default key in music notation applications and MIDI editors/sequencers. These statistics may serve as a reference for future researchers in terms of selecting appropriate datasets.

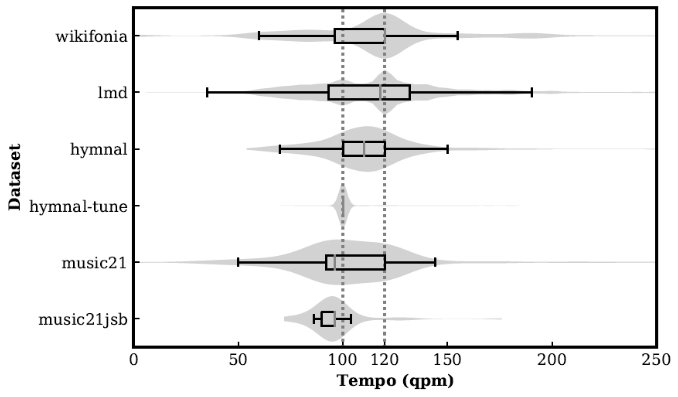


Fig. 6. Initial-tempo distributions for different datasets (those without tempo information are not presented).

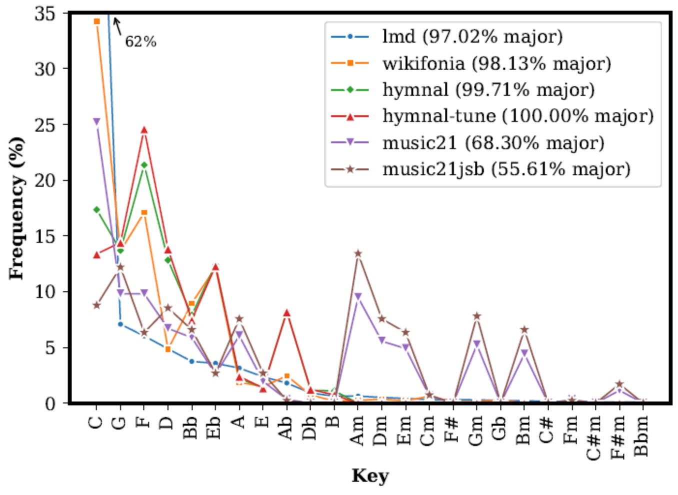


Fig. 7. Key distributions for different datasets. The keys are sorted w.r.t. their frequencies in Lakh MIDI Dataset.

V. 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 shortterm memory (LSTM) network a gated recurrent unit (GRU) network [4] and a Transformer network.

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 [3]. 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.

A. Autoregressive Models On Different Datasets

We train the model on a dataset and then test it on the same dataset in this experiment. Fig. 8 illustrates the perplexities associated with various models on various datasets. As can be seen, all models exhibit similar characteristics. They attain lower perplexities when dealing with small, homogenous datasets, but result in higher perplexities when dealing with bigger, more varied datasets. That is, the test perplexity may serve as a proxy for the dataset's variety. Additionally, Fig. 9 illustrates perplexities as a function of dataset size (in hours). By classifying datasets as multi-pitch (i.e., capable of taking any number of concurrent notes) or monophonic, we can observe that perplexity is positively linked with dataset size within each category.

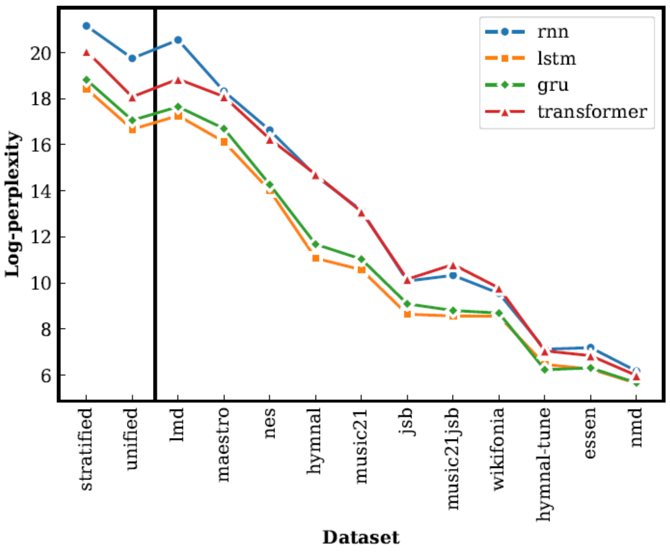


Fig. 8. Log-perplexities for different models on different datasets, sorted by the values for the LSTM model.

In this experiment, we train a model on a dataset and then test it on many other datasets '. The perplexities for each train–test dataset pair are shown in Fig. 10. The following are some observations:

In general, generalizability across datasets is not symmetric. For instance, although a model trained on LMD generalises well to all other datasets, not all models trained on other datasets generalise to LMD, which may be owing to LMD's size and cross-genre nature.

Multi-pitch models generalise effectively to monophonic datasets, while monophonic models do not transfer to multi-pitch datasets (see the red block in Fig. 10).

The model developed using the JSBach Chorale Dataset is not generalizable to any of the other datasets (see the orange block in Fig. 10). This may be because its samples are downsampled to a quarter note resolution, resulting in an unique note duration distribution.

In comparison to other datasets, the majority of datasets generalise poorly to the NES Music Database (see the green block in Fig. 10). This may be because the NES Music Database only includes game soundtracks.

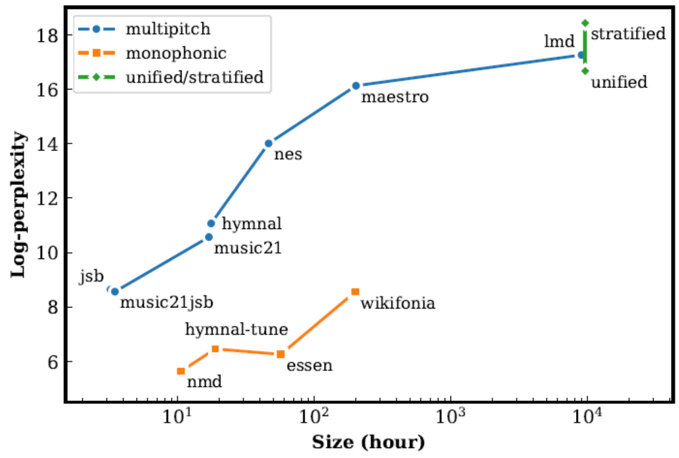


Fig. 9. Log-perplexities for the LSTM model versus dataset size in hours. Each point corresponds to a dataset.

B. Effects Of Combining Heterogeneous Datasets

As shown in Fig. 10, LMD has the greatest generalizability, perhaps due to its size, diversity, and crossgenre nature. A model trained on LMD, on the other hand, does not generalise well to the NES Music Database (see the brown block in the close-up of Fig. 10). Thus, we are interested in determining if combining several heterogeneous datasets may aid in increasing generalizability.

All eleven datasets mentioned in Table 1 are combined into a single big unified dataset. Given the size disparity between both datasets, merely concatenating them may result in a significant imbalance and bias toward the larger dataset. As a result, we examine a variant that employs stratified sampling throughout the training process. To get a data sample from the stratified dataset, we uniformly choose one of the eleven datasets and then randomly select one sample from it. Take note that during test time, stratified sampling is deactivated. Additionally, we present the findings for these two datasets in Figures 8, 9 and 10. As shown in Fig. 10, integrating datasets from several sources enhances the model's generalizability. This is consistent with the result in [15] that models trained on specific cross-domain datasets generalise more well to previously unknown datasets. Additionally, stratified sampling mitigates the source imbalance issue by decreasing perplexities in the majority of datasets at the expense of higher perplexity on LMD.

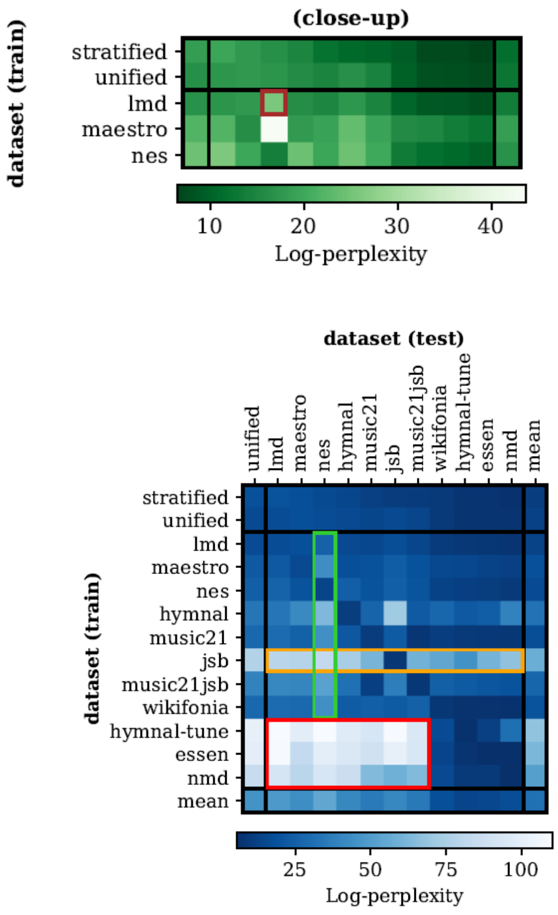


Fig. 10. Results on cross-dataset generalizability. The numbers and colours indicate the log-perplexities of an LSTM model trained on one dataset (row) and tested on another (column). The datasets are ordered according to their diagonal values, which means that they may be trained and tested on the same dataset.

VI. CONCLUSION

We've introduced MusPy, a new toolset for building music generating systems. We covered the library's architecture and functionality, as well as examples of data pipelines. We performed statistical research and experimentation on the eleven presently supported datasets using MusPy's dataset management system to determine their relative diversity and cross-dataset generalizability. These findings may aid researchers in selecting suitable datasets for future study. Finally, we demonstrated how integrating diverse datasets may assist enhance a machine learning model's generalizability.

VII. 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]. 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.

[3]. 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.

[4]. A. Roberts, J. Engel, C. Raffel, C. Hawthorne, and

[5]. 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.

[6]. 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.

[7]. 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.

[8]. 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.

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

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

[11].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.

[12].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.

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

[14].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.

[15].C. Mckay and I. Fujinaga, "JSymbolic: A feature extractor for MIDI files, C. Raffel, "Learning-based methods for comparing sequences, with applications to audio-to-MIDI alignment and

matching," Ph.D. dissertation, Columbia University, 2016.

[16]. 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.

[17]. 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.