# ASMD: AN AUTOMATIC FRAMEWORK FOR COMPILING MULTIMODAL DATASETS WITH AUDIO AND SCORES

**Federico Simonetta** **Stavros Ntalampiras** **Federico Avanzini**
University of Milan
Department of Computer Science
LIM - Music Informatics Laboratory
[name].[surname]@unimi.it

## ABSTRACT

This paper describes an open-source Python framework for handling datasets for music processing tasks, built with the aim of improving the reproducibility of research projects in music computing and assessing the generalization abilities of machine learning models. The framework enables the automatic download and installation of several commonly used datasets for multimodal music processing. Specifically, we provide a Python API to access the datasets through Boolean set operations based on particular attributes, such as intersections and unions of composers, instruments, and so on. The framework is designed to ease the inclusion of new datasets and the respective ground-truth annotations so that one can build, convert, and extend one's own collection as well as distribute it by means of a compliant format to take advantage of the API. All code and ground-truth are released under suitable open licenses.

## 1. INTRODUCTION

A recent trend in computer science is the adoption of multimodal strategies for increasing the effectiveness of algorithmic solutions in several domains [1–5]. This comes as a natural consequence of the a) ever-increasing availability of computational resources, which are now able to deal with big data, and b) popularity of machine learning algorithms, the performance of which is boosted as more data (including multimodal) becomes available. As a result, machine learning technologies are now employed in novel and unexplored ways.

In the context of music information processing, several tasks still pose unsolved challenges to the research community, and multimodal approaches could provide a promising path. The fields of *multimodal music processing* and *multimodal music representation* have already been investigated in previous works [6, 7].

Two issues that are more and more debated in several research fields are the ability to *reproduce* published results [8, 9] and to *generalize* the resulting models [10].



Reproducibility is associated with the differences occurring in various implementations of the same method. As an example, one issue is related to the different data formats used in music and in the available datasets, which might cause troubles in the translation between representation formats and, consequently, in the reproducibility of research.

The generalization problem instead is due, among other factors, to the need of large and well-annotated datasets for training effective models. In particular, the whole field of music information processing has only a limited number of large datasets which could be much more useful if they could be merged together. Music itself, moreover, is particularly affected by the difficulty of creating accurate annotations to evaluate and train models, often hindering the collection of large datasets and causing a low generalization ability.

With these three keywords in mind (*multimodal*, *reproducibility* and *generalization*), we have built ASMD to help researchers in the standardization of music ground-truth annotations and in the distribution of datasets. ASMD is the acronym for Audio-Score Meta-Dataset and provides a framework for describing, converting, and accessing a single dataset which includes various datasets – hence the expression *Meta-Dataset*; it was born as a side-project of a research about audio-to-score alignment and, consequently, it contains audio recordings and music scores for all the data included in the official release – hence the *Audio-Score* part. However, we have endeavoured to make ASMD able to include any contribute from anyone. ASMD is available under free licenses. [1]

A similar effort is held by `mirdata` [11], a Python package for downloading and using common MIR datasets. However, our work is more focused on multimodality and tries to keep the entire framework easily extensible and modular.

In the following sections, we describe a) the design principles, b) the implementation details, c) a few use cases and, d) possible future works.

## 2. DESIGN PRINCIPLES AND SPECIFICATIONS

In this section we present the principles which guided the design of the framework. Throughout this paper, we are

[1] Code is available at `https://framagit.org/sapo/asmd/`, documentation is available at `https://asmd.readthedocs.io/`

going to use the word *annotation* to refer to any music-related data complementing an audio recording. For instance, common types of annotations are music notes, f0 for each audio frame, beat position, separated audio tracks, etc.

### 2.1 Generalization

With *generalization*, we mean the ability of including different datasets which are distributed with various formats and annotation types in the model generation process. This is an important issue especially during the conversion procedure: since we aimed at distributing a conversion script to recreate annotations from scratch for the sake of *reproducibility*, we need to be able to handle all various storage systems – e.g. file name patterns, directory structures, etc. – and file formats – e.g. midi, csv, musicxml, ad-hoc formats, etc.

Also, our ground-truth format should be generic enough to represent all the information contained in the available ground-truths and, at the same time, it should permit to handle datasets with different ground-truth types – i.e. one dataset could provide *aligned notes* and *f0*, while another one could provide *aligned notes* and *beat-tracking*, and they should be completely accessible.

### 2.2 Modularity

*Modularity* refers to the re-use of parts of the framework in different contexts. Modularity is important during both addition of new datasets and usage of the API. To ease the conversion between ground-truth file formats, the user should be able to re-use existing utilities to include additional datasets. Moreover, the user should be allowed to use only some parts of the datasets and the corresponding annotations.

### 2.3 Extensibility

The purpose of the framework is to create a tool to help the standardization of music information processing research. Consequently, we aimed for a framework that is completely open to new additions: it should be easy for the user to add new datasets without editing sources from the framework itself. Also, it should be easy to convert from existing formats in order to take advantage of the API and to be able to merge existing datasets. Finally, the framework should provide a usable format to add new annotations so that new datasets can be natively created with the incorporated tools.

### 2.4 Set operability

Since the framework aims at merging multiple datasets, we wanted to add the ability to perform set operations over datasets. As an example, within the context of *automatic music transcription* research, several large datasets exist consisting of piano music [12–14], but only few and considerably smaller are available for other instruments [15–19]. Consequently, a useful feature of the framework would be the ability to select only some songs from multiple datasets based on particular attributes, such as the instrument involved, the number of instruments, the composer or the type of ground-truth available for that song.

### 2.5 Copyrights

A common issue with distributing music recordings and annotations are copyrights. Today, most of the datasets typically used for music information processing are released under Creative Commons Licenses, but there are many exceptions of datasets released under closed terms [16, 20] or not released at all because of copyright restrictions [21]. To overcome this problem, we wanted all datasets to be downloadable from their official sources, in order to avoid any form of redistribution. Nonetheless, all the annotations that we produced were redistributable under Creative Commons License.

### 2.6 Audio-score oriented

Besides the effort to produce a general framework for music processing experiments, this project was born as a utility during conducting research addressing the audio-to-score problem. The underlying idea is that we have various scores and large amounts of audio available to end-users, thus trained models could easily take advantage of such multimodality (i.e. the ability of the model to exploit both scores and audio). The main problem is the availability of data for training the models: there is abundance of aligned data, but without the corresponding scores; on the other hand, when scores are available, aligned performances are almost invariably missing. Thus, the choice of the datasets that are included at now has mainly been focused on datasets providing audio, symbolic scores and alignment annotations. However, since datasets fitting all these requirements are quite rare, we wanted to augment the data available to increase the alignment data usable in our research.

## 3. IMPLEMENTATION DETAILS

This section details the implementation satisfying the design principles outlined in section 2. Figure 1 depicts the structure of the overall framework and the interactions between its modules.

### 3.1 The datasets.json file

The entire framework is based on a small-sized but fundamental JSON file loaded by the API and the installation script to get the path where files are installed. Moreover, the user can optionally set a custom directory where to decompress downloaded files if the hard-disk space is a critical issue. Once the installation path is found, the script looks for the existing directories in that path to discover which datasets are already installed and skips them. The API, instead, uses the information of the installation directory to decouple the definition of each single dataset from the directory structure of the user: a user can have the same dataset installed in multiple directories, or use the same dataset from different *datasets.json* without interfering with the API.

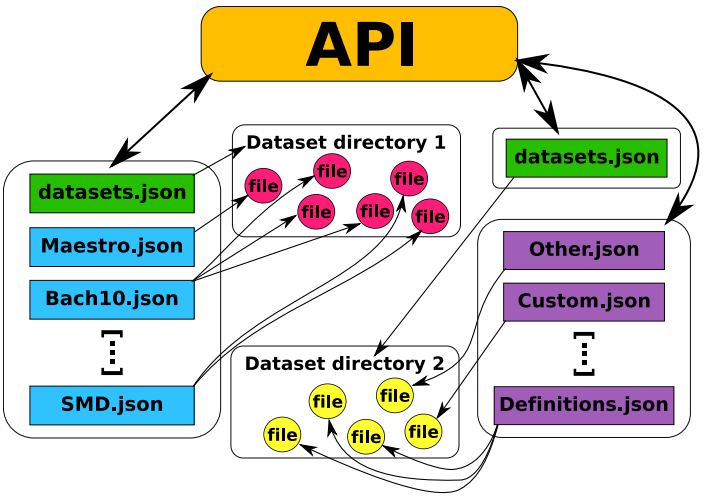


Figure 1. Block diagram of the proposed framework: API interacts with definitions and `datasets.json`; the former contain references to the actual sound recording files and annotations, while the latter contains references to the dataset root path.

### 3.2 Definitions

In the context of this framework, a *dataset definition* is essentially a JSON [2] file which contains generic description of a dataset. *Definitions* are built by using a pre-defined schema allowing the definition of various information useful for the installation of the dataset and for the usage of the API – e.g. for filtering the dataset. If any of the information is not available for a dataset, the value `unknown` is offered as well.

Examples of information contained in *definitions* are:

- `ensemble`: if the dataset contains solo instrument music pieces or ensemble;
- `instruments`: a list of instruments that are used in the dataset;
- `sources`: if source-separated tracks are available, their format can be added here;
- `recording`: the format of audio recordings;
- `install`: field containing all information for installing the dataset: URL for downloading, shell commands for post-processing data, and so on;
- `ground-truth`: field associated to each type of ground-truth supported by the framework indicating whether the specific annotation type is available or not – see Sec 3.3;
- `songs`: a list of songs with meta-data such as the composer name and instruments used in these songs and with the list of paths to the audio recordings and to the annotations.

Once a dataset has been described in this schema, its definition can be used *out-of-the-box* by simply specifying to the API the path of its folder, possibly containing other dataset definitions. All the paths specified in a *definition* must be relative to the installation directory as described in Sec. 3.1.

For the sake of generalization, we had to deal with a wide heterogeneity in path management among datasets. For instance, Bach10 [18] provides one different annotation file per each instrument in a song; in such a case we list all the annotation files for each song and leave to the API the task of reassembling them. PHENICX [16], instead, only provides source-separated tracks and thus we list all of them to reference the mixed track; again, we leave to the API the task of mixing them. In general, we have kept the following principle: if a list of paths is provided where one would logically expect a single path – such as in mixed tracks or annotation files – it is intended that the files in the list should be "merged" whatever this means for that specific file-type. For instance, if multiple audio recordings are provided instead of only one, it is assumed that the *mixed* track is derivable by adding (and normalizing) all listed tracks; if multiple annotation files are provided, it is assumed that each annotation file refers to a different instrument.

### 3.3 Annotations

Annotations are added in a custom JSON compressed format stored in the same directory of the audio track that they refer to. In fact, annotation files can be stored anywhere and their path must be provided in the dataset definition relatively to the installation path defined in *datasets.json*. Moreover, one annotation file must be provided for each instrument of the track; if multiple instruments should refer to the same annotations – e.g. first and second violins – the annotation file can be only one, but in the dataset definition file, its path should be repeated once for each instrument referring to it.

Multiple types of annotations are available, but not all of them are provided for all the datasets in the official collection. In the dataset definition, the type of annotations available should be explained. In our implementation, we used 3 different levels to describe ground truth availability

[2] `https://www.json.org/json-en.html`

```
import audioscoredataset as asd

d = asd.Dataset()
d.filter(instruments=['piano'], ensemble=False, composer='Mozart',
↪  ground_truth=['precise_alignment'])

# get audio and all the annotations
audio_array, sources_array, ground_truth_array = d.get_item(1)

# get only the annotations you want
audio_array = d.get_mix(2)
source_array = d.get_source(2)
ground_truth_list = d.get_gts(2)

# get a MIDI Toolbox-like numpy array
mat = d.get_score(2, score_type=['precise_alignment'])

# get a pianoroll numpy array
mat = d.get_pianoroll(2, score_type=['non_aligned'])

# or to process songs in parallel using joblib:
def processing(i, dataset, **args, *kwargs):
    mat = d.get_score(2, score_type=['precise_alignment'])
    # other stuffs here
    pass

d.filter(instruments=['violin']).parallel(processing, n_jobs=-1)
```

Listing 1: Example of usage for official *definitions*

and reliability:

**0:** annotation type not available

**1:** annotation type available and manually or mechanically annotated: this type of annotation has been added by a domain expert or some mechanical transducer – e.g. Disklavier.

**2:** annotation type available and algorithmically annotated: this type of annotation has been added by exploiting a state-of-art algorithm.

The types of annotations currently supported are:

1. *precise alignment*: onsets and offsets times in seconds, pitches, velocities and note names for each note played in the recording, taking into account asynchronies inside chords;
2. *broad alignment*: same as *precise alignment* but the alignment does not consider asynchronies inside chords;
3. *non aligned notes*: same as *precise alignment* but not aligned (see 3.4 for more information);
4. *f0*: the f0 of this instrument for each audio frame in the corresponding track;
5. *beats non aligned*: time instances of beats in the non-aligned data;
6. *instrument*: General Midi program number associated with this instrument, starting from 0, while value 128 indicates a drums kit.

### 3.4 Alignment

As described in section 2.6, this project originated for music alignment research. One problem is the lack of large datasets containing audio recordings, aligned notes and symbolic non aligned scores.

The approach that we used to overcome this problem is to statistically analyze the available manual annotations and to augment the data by approximating them through the statistical model. To prevent biases, we also replaced the manual annotations with the approximated ones.

For now, the statistical analysis is simple: we compute the mean and the standard deviation of offsets and onsets for each piece. Then, we store the histogram of the standardized offsets and onsets of each note; we also store histograms of the mean and standard deviation values of each piece. To create new misaligned data, we chose a standardized value for each note accompanied by a mean and a standard deviation for each piece, using the corresponding histograms; with these data, we can compute a non-standardized value for each note. Note that initially the histograms are normalized so that they satisfy certain given constraints. In the distributed code, the standardized values are normalized to 1 (that is, the maximum value is 1 second), while standard deviations are normalized to 0.2.

An additional problem is due to the fact that the time units in the aligned data are seconds, while those in the scores are note lengths – e.g. breve, semibreve and so on. Usually, one translates a note length to seconds by using BPM; however, in some scores the BPM annotation is unavailable or is not reliable. Hence, during the statistical analysis, we always consider the tempo as 20 BPM, which is a non-usual BPM, in the attempt of minimizing its overall influence. If we used a usual BPM, such as 60 or 120, songs with BPM near to that value would have biased the analysis. Moreover, models trained using the produced alignment annotations are ensured to be BPM-independent. Note that one

```
import audioscoredataset as asd

d = asd.Dataset(['path/to/directory/containing/custom/definitions',
↪  'path/to/the/official/definitions/'])
d.filter(instruments=['piano'], ensemble=False, composer='Mozart',
↪  ground_truth=['precise_alignment'])
```

Listing 2: Example of usage for custom *definitions*

can still try to derive BPM information by making a BPM estimation over the audio [22, 23], a process which highly depends on the algorithm's precision.

### 3.5 API

The framework is complemented with a Python API written in Cython. [3] It allows in particular to load various dataset definitions aside of the official ones. The API provides methods to retrieve audio and annotations in various structures, such as a matrix list of notes similar to the one used by *Matlab MIDI Toolbox* [24] or pianorolls. Thanks to the API, one can also filter the loaded datasets' songs based on the original dataset, active instrument, ensemble or solo instrumentation, composer, available annotation types, etc.

Moreover, since the API basically consists in a class representing a large dataset, it is very easy to extend it in order to use it in conjunction of PyTorch or TensorFlow frameworks for training neural network models. In Sec. 4 we provide an example of the specific functionality.

### 3.6 Conversion

To give the user the ability to write his/her own definitions without having to edit the framework code, we designed a conversion procedure as follows:

1. the creator can use already developed conversion tools for the most common file formats (MIDI, sonic visualizer, etc.);
2. the creator can still write an ad-hoc function which converts a file from the original format to the ASMD one; in this case the creator has to decorate the conversion function with a special decorator provided by ASMD;
3. the creator adds the needed conversion function in the `install` section in the dataset definition;
4. the user can run the conversion script for only a specific dataset or for all other datasets.

All the technical details are available in the official documentation. [4]

## 4. USE CASES

This section demonstrates the efficacy of the ASMD framework through four different use cases.

[3] `https://cython.org/`
[4] `https://asmd.readthedocs.io/`

### 4.1 Using API with the official dataset collection

To use the API, the user should carry out the following steps:

- import `audioscoredataset`;
- create a `audioscoredataset.Dataset` object, giving the path of the `datasets.json` file as an argument to the constructor;
- use the `filter` method on the object to filter data according to his/her needs (conveniently, it is also possible to re-filter them at a later stage, without reloading the `datasets.json` file);
- retrieve elements by calling the `get_item` method or similar ones.

After the execution of the `filter` method, the `Dataset` instance will contain a field `paths` representing the list of correct paths to the files requested by the user. Listing 1 shows an example of such an operation.

### 4.2 Using API with definitions for a customized dataset

Whenever the user wishes to apply customized definitions, he/she need simply to provide the list of directories to the `Dataset` constructor, as shown in listing 2.

### 4.3 Using ASMD with PyTorch

Integrate *ASMD* with *PyTorch* is straightforward. The user has to inherit from both PyTorch and ASMD `Dataset` classes and to implement the `__getitem__` method. Listing 3 shows such an example.

### 4.4 Writing a conversion function and a custom dataset definition

Towards adding new definitions enabling users to download datasets, a user should also provide a conversion function. Listing 4 is an example of one can write its own conversion function. However, conversion functions for the most common file types – such as Midi and Sonic Visualizer – are already provided by the framework.

## 5. CONCLUSIONS

Future works will focus on the enhancement of conversion and installation procedures, as well as on the definition of standards for music annotations. In addition, multimodal music processing often requires processing of annotation types not included in this version of the framework, but

```python
import torch
import audioscoredataset as asd
from torch.utils.data import Dataset as TorchDataset

class MyDataset(asd.Dataset, TorchDataset):
    def __init__(self, *args, **kargs):
        super().__init__(['path/to/definitions']).filter(instruments=['piano'])

    def __getitem__(self, i):
        # for instance, return the MIDI Toolbox-like score
        return torch.tensor(self.get_score(i))

    def another_awsome_method(self, *args, **kargs):
        print("Hello, world!")

for i, mat in enumerate(MyDataset()):
    # train your nn model here
```

Listing 3: Example for using ASMD inside PyTorch

```python
from audioscoredataset.convert_from_file import convert, prototype_gt
from copy import deepcopy

# use @convert
@convert(['.myext'])
def function_which_converts(filename, *args, **kargs):
    # prepare empty output
    out = deepcopy(prototype_gt)

    # open file
    data = csv.reader(open(filename), delimiter=',')

    # fill output dictionary
    for row in data:
        out[alignment]["onsets"].append(float(row[0]))
        out[alignment]["offsets"].append(float(row[0]) + float(row[2]))
        out[alignment]["pitches"].append(int(row[1])

    return out
```

Listing 4: Example for writing a custom conversion function

could instead be handled in a future release. Some annotation types could be stored in standalone formats and users should be able to distribute annotations focusing only on a specific ground truth kind, thus enhancing the distributed infrastructure of ASMD.

Studying the user experience of the framework should also be a priority: for instance, users could be able to choose datasets also based on the estimation of the download time since for some datasets that is a relevant issue. Labels used in the annotation format are also relevant to ease the usage of the framework by new users, especially in a multidisciplinary field such as sound and music computing.

This paper presented a new framework for multimodal music processing. We hope that our efforts in easing the development of multimodal machine learning approaches for music information processing will be useful to the sound and music computing community. We are completely aware that for a truly general and usable framework, the participation of various and different perspectives is needed and we are therefore open to any contribution towards the creation of utilities that allow training and testing multimodal models, ensuring reasonable generalization ability and reliable reproducibility of scientific results.

### Acknowledgments

We would like to thank all people that worked on the datasets used in the ASMD framework: Bach10 [18], Maestro [12], MusicNet [15], PHENICX Anechoic [16], SMD [13], Traditional Flute Dataset [17], TRIOS [19] and Vienna 4x22 Piano Corpus [14].

## 6. REFERENCES

[1] T. Baltrusaitis, C. Ahuja, and L.-P. Morency, "Multimodal machine learning: A survey and taxonomy,"

*IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 41, no. 2, pp. 423–443, 2018.

[2] S. Ntalampiras, F. Avanzini, and L. A. Ludovico, "Fusing acoustic and electroencephalographic modalities for user-independent emotion prediction," in *2nd IEEE Int. Conf. on Cognitive Computing (ICCC)*, 2019, pp. 36–41.

[3] S. Ntalampiras, D. Arsić, M. Hofmann, M. Andersson, and T. Ganchev, "PROMETHEUS: heterogeneous sensor database in support of research on human behavioral patterns in unrestricted environments," *Signal, Image and Video Processing*, vol. 8, no. 7, pp. 1211–1231, 2012.

[4] P. K. Atrey, M. A. Hossain, A. E. Saddik, and M. S. Kankanhalli, "Multimodal fusion for multimedia analysis: A survey," *J. of Multimedia Systems*, vol. 16, no. 6, pp. 345–379, 2010.

[5] M. Minsky, "Logical versus analogical or symbolic versus connectionist or neat versus scruffy," *AI Magazine*, vol. 12, no. 2, pp. 34–51, 1991.

[6] F. Simonetta, S. Ntalampiras, and F. Avanzini, "Multimodal Music Information Processing and Retrieval: Survey and Future Challenges," in *1st Int. Work. on Multilayer Music Representation and Processing (MMRP)*, 2019, pp. 10–18.

[7] L. A. Ludovico, A. Baratè, F. Simonetta, and D. A. Mauro, "On the adoption of standard encoding formats to ensure interoperability of music digital archives: The ieee 1599 format." in *6th Int. Conf. on Digital Libraries for Musicology (DLfM)*, 2019, pp. 20–24.

[8] M. Baker, "1,500 scientists lift the lid on reproducibility," *Nature News*, vol. 533, no. 7604, p. 452, 2016.

[9] M. Hutson, "Artificial intelligence faces reproducibility crisis," *Science*, vol. 359, no. 6377, pp. 725–726, 2018.

[10] Y. S. Abu-Mostafa, M. Magdon-Ismail, and H.-T. Lin, *Learning From Data*. AMLBook, 2012.

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

[12] C. Hawthorne, A. Stasyuk, A. Roberts, I. Simon, C.-Z. A. Huang, S. Dieleman, E. Elsen, J. H. Engel, and D. Eck, "Enabling factorized piano music modeling and generation with the maestro dataset." in *7th Int. Conf. on Learning Representations (ICLR)*, 2019.

[13] M. Müller, V. Konz, W. Bogler, and V. Arifi-Müller, "Saarland music data (SMD)," in *12th Int. Conf. on Music Information Retrieval (ISMIR)*, 2011.

[14] W. Goebl. (1999) The vienna 4x22 piano corpus. [Online]. Available: http://dx.doi.org/10.21939/4X22

[15] J. Thickstun, Z. Harchaoui, D. P. Foster, and S. M. Kakade, "Invariances and data augmentation for supervised music transcription." in *43rd Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)*, 2018, pp. 2241–2245.

[16] M. Miron, J. J. Carabias-Orti, J. J. Bosch, E. Gómez, and J. Janer, "Score-informed source separation for multichannel orchestral recordings." *J. Electrical and Computer Engineering*, vol. 2016, pp. 1–19, 2016.

[17] J. P. Brum. (2018) Traditional flute dataset for score alignment". [Online]. Available: https://www.kaggle.com/jbraga/traditional-flute-dataset

[18] Z. Duan and B. Pardo, "Soundprism: An online system for score-informed source separation of music audio." *J. Sel. Topics Signal Processing*, vol. 5, no. 6, pp. 1205–1215, 2011.

[19] J. Fritsch. (2012) The trios score-aligned multitrack recordings dataset. [Online]. Available: https://c4dm.eecs.qmul.ac.uk/rdr/handle/123456789/27

[20] F. Simonetta, F. Carnovalini, N. Orio, and A. Rodà, "Symbolic music similarity through a graph-based representation," in *13th Audio Mostly Int. Conf. (AM'18)*, 2018.

[21] F. Simonetta, C. Cancino-Chacón, S. Ntalampiras, and G. Widmer, "A convolutional approach to melody line identification in symbolic scores," in *20th Int. Conf. on Music Information Retrieval Conference (ISMIR)*, 2019.

[22] H. Schreiber and M. Müller, "A single-step approach to musical tempo estimation using a convolutional neural network." in *19th Int. Conf. on Music Information Retrieval (ISMIR)*, 2018, pp. 98–105.

[23] S. Böck, F. Krebs, and G. Widmer, "Accurate tempo estimation based on recurrent neural networks and resonating comb filters." in *16th Int. Conf. on Music Information Retrieval (ISMIR)*, 2015, pp. 625–631.

[24] T. Eerola and P. Toiviainen, "Mir in matlab: The midi toolbox." in *5th Int. Conf. on Music Information Retrieval (ISMIR)*, 2004.