A Minimal Template for Interactive Web-based Demonstrations of Musical Machine Learning

Vibert Thio, Hao-Min Liu, Yin-Cheng Yeh, and Yi-Hsuan Yang
Research Center for Information Technology Innovation, Academia Sinica, Taipei, Taiwan
{vibertthio, paul115236, ycyeh, yang}@citi.sinica.edu.tw

ABSTRACT
New machine learning algorithms are being developed to solve problems in different areas, including music. Intuitive, accessible, and understandable demonstrations of the newly built models could help attract the attention of people from different disciplines and evoke discussions. However, we notice that it has not been a common practice for researchers working on musical machine learning to demonstrate their models in an interactive way. To address this issue, we present in this paper an template that is specifically designed to demonstrate symbolic musical machine learning models on the web. The template comes with a small codebase, is open source, and is meant to be easy to use by any practitioners to implement their own demonstrations. Moreover, its modular design facilitates the reuse of the musical components and accelerates the implementation. We use the template to build interactive demonstrations of four exemplary music generation models. We show that the built-in interactivity and real-time audio rendering of the browser make the demonstration easier to understand and to play with. It also helps researchers to gain insights into different models and to A/B test them.

ACM Classification Keywords
D.2.2 Design Tools and Techniques: Modules and interfaces; H.5.2 User Interfaces: Prototyping; H.5.5 Sound and Music Computing: Systems

Author Keywords
Musical interface; web; latent space; deep learning

ACM Reference Format
Vibert Thio, Hao-Min Liu, Yin-Cheng Yeh, and Yi-Hsuan Yang. 2019. A Minimal Template for Interactive Web-based Demonstrations of Musical Machine Learning. *In Joint Proceedings of the ACM IUI 2019 Workshops*, Los Angeles, USA, March 20, 2019, 6 pages.

INTRODUCTION
Recent years have witnessed great progress in applying machine learning (ML) to music related problems, such as thumbnailing [10], music generation [5, 7, 21], and style transfer [15]. To demonstrate the result of such musical machine learning models, researchers usually put the audio output as the result on the accompanying project websites. This method works well in the early days. However, as the ML models themselves are getting more complicated, some concepts of the algorithms may not be clearly expressed with only static sounds.

In the neighboring field of computer vision, many interactive demonstrations of ML models have been developed recently. Famous examples include DeepDream [18], image/video style transfer [8, 23], and DCGAN [19]. These interactive demos provoke active discussions and positive anticipation about the technology. Nevertheless, the demonstration of musical machine learning models is not as easy in the case of computer vision, due to the fact that it involves audio rendering (i.e., we cannot simply use images for demonstration). Web Audio API, a high-level JavaScript API for processing and synthesizing audio in web applications, was published only in 2011, which is not far from now compared to WebGL and other features of the browser. Furthermore, interactivity is needed to improve understandability and create engaging experiences.

Musical machine learning is gaining increasing attention. We believe that if more people from other fields, such as art and music, start to appreciate the new models of musical machine learning, it is easier to create an active community and to stimulate new ideas to improve the technology.

The goal of this paper is to fulfill this need by building and sharing with the community a template that is designed to demonstrate ML models for symbolic-domain music processing and generation, in an interactive way. Therefore, The template is also open-source on GitHub (`https://github.com/vibertthio/musical-ml-web-demo-minimal-template`).

RELATED WORKS

Audio Rendering in Python
When it comes to testing or interacting with the musical machine learning models, the output of the models must be rendered as audio files or streams to be listened to by humans. Most researchers in the field nowadays use Python as the programming language for model implementation because of the powerful ML and statistical packages built around it. For example, `librosa` [17] is a Python package often used for audio and signal processing. It includes functions for spectral analysis, display, tempo detection, structural analysis, and output. Many interactive demonstrations are built with librosa on the Jupyter Notebook. However, a major drawback of this approach is that the audio files have to be sent over the Internet

for demonstration, which can be slow sometimes depending on the network connection bandwidth.

Another widely-used Python package is `pretty_midi` [20], which is designed for manipulation of symbolic-domain data such as Musical Instrument Digital Interface (MIDI) data. It could be used as a tool to render the symbolic output of a musical machine learning model, such as a melody generation model [24]. The problem is that after getting the result as a MIDI file, the user still has to put it into a digital audio workstation (DAW) to synthesize the audio waveform from the MIDI. For better listening experience, the researcher still has to synthesize the audio files offline and then send the audio files over the Internet for demonstration.

Different from prior works, we propose to use Tone.js, a JavaScript framework for rendering MIDI files into audio directly in the browser on the client side. This turns out to be a much more efficient way to demonstrate a symbolic musical ML model. It also helps build an interactive demo.

Interactive Musical Machine Learning

Similar to our work, Vogl *et al.* [7] introduced an interactive App for drum pattern generation based on ML. They used generative adversarial networks (GAN) [9] as the generative model, which is trained on a MIDI dataset. The user interface consists of some classic sequencer with an x/y pad that controls the complexity and the loudness of the drum pattern generated. Additionally, controls for genre and swing are said to be provided. However, both the demo and its source code cannot be found online currently. It is not clear whether the App is built on iOS, Android, or the Web.

Closely related to our project is the MusicVAE model [21] presented by Magenta, Google Brain Team. MusicVAE is a generative recurrent variational autoencoder (VAE) model that can generate melodies and drum beats. Importantly, the authors also released a JavaScript package called `Magenta.js` (`https://github.com/tensorflow/magenta-js/`) [22], to make their models more accessible. They also provide some pre-trained models of MusicVAE along with other ones. There are several interactive demos using the package, as can be found on their website [16]. Most of them are well designed, user-friendly, and extremely helpful for understanding the models. Yet, the major drawback is that the codebase of the project is a *monolithic* one [11] and is therefore quite big.[1] Users may not easily modify the code for customization. For example, because Magenta uses Tensorflow as the backbone deep learning framework, it is hard for PyTorch users to use `Magenta.js`.

TEMPLATE DESIGN

In this paper, we present a minimal template, which is simple, flexible, and designed for interactive demonstration of symbolic musical machine learning models on the web.

[1]A monolithic repo is defined in [11] as: "a model of source code organization where engineers have broad access to source code, a shared set of tooling, and a single set of common dependencies. This standardization and level of access are enabled by having a single, shared repo that stores the source code for all the projects in an organization." This is the case of the Google Magenta project.

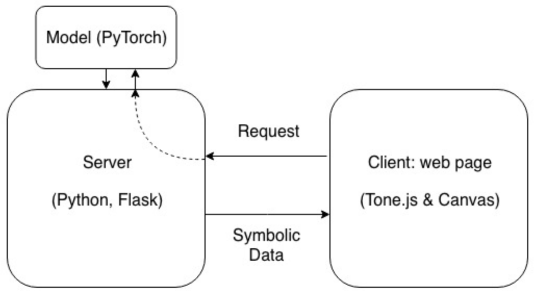

Figure 1. The schematic diagram of the proposed template.

We choose the web as the platform for the demonstrations for several reasons. First, it is convenient as the user only has to open a browser or click the hyperlink to play with the models. Second, it is inherently interactive. The system can utilize a plenty of forms of interaction available in the browser to create the specific user experience.

Requirements

In the design process, we have prioritized some crucial qualities. First, we made the structure of the design as simple as possible. In most cases, the demo is for a proof-of-concept rather than to showcase a ready-to-sell product. Hence, we desire that a person with basic knowledge of Python and JavaScript could understand our template within a short period of time so that the template can serve as a minimal starting point.

Second, the codebase should be small, so that transplanting a new model into this template is easier. Moreover, a small codebase also makes it easier to debug.

Third, the audio rendering must be interactive and real-time. The demonstrations must be responsive to some inputs from the user so that the user could understand the model by knowing how it works in several different ways. As for researchers, if the result could be rendered instantly, it would be easier to A/B test different designs of models or parameters.

Finally, we want the components of the template to be modular, so that they can be reused and recombined easily. Such components may include, e.g., `chord progression`, `pianoroll`, `drum pattern`, and `sliders`. Practitioners can build their own demonstrations based on these components.

System Architecture

As shown in Figure 1, the system consists of three parts: a musical machine learning model, a server, and a client. When a user opens the URL of the demonstration site, it will load the client program into the browser and render the basic interface. The client program will send a request to the server to fetch the data. The server program will parse the request and use the function based on the model to make the corresponding output and send it back to the client to render the audio effect.

Server

We used `Flask` (`http://flask.pocoo.org/`), a lightweight web application framework, to build the server. Flask only han-

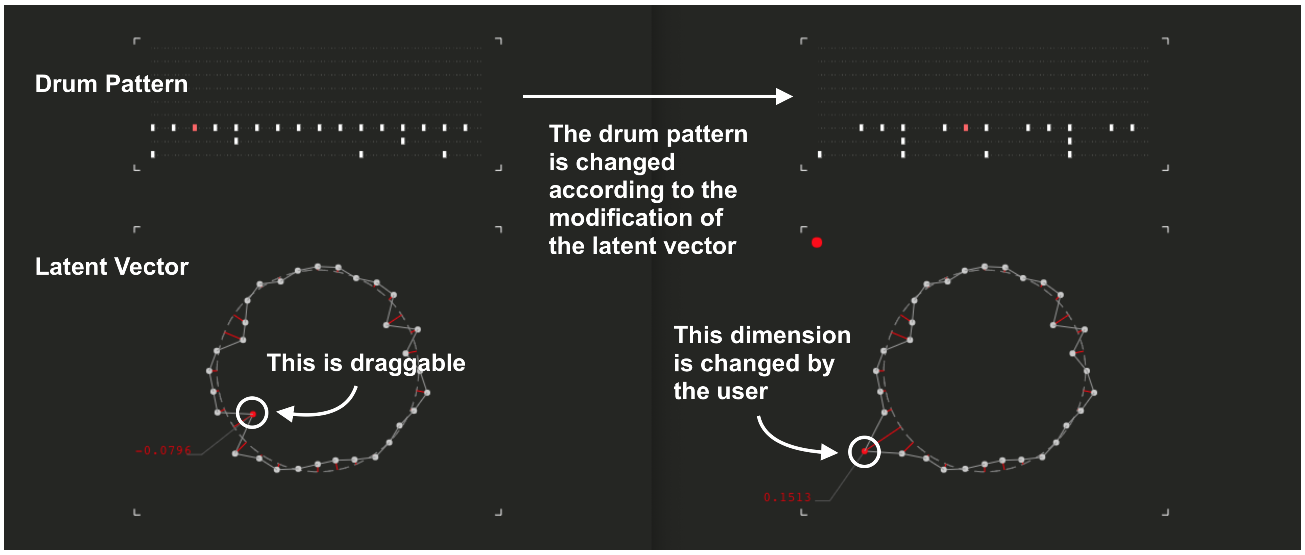
Drum Pattern
Latent Vector
The drum pattern is changed according to the modification of the latent vector
This is draggable
This dimension is changed by the user
-0.0796
0.1513

Figure 2. The Latent Inspector. Left: a snapshot of the demonstration. The upper half is an editable drum pattern display, whereas the bottom half is a graph showing the N-dimensional latent vector (here $N = 32$) of the VAE model. Each vertex of the graph can be adjusted independently, to change the latent vector and accordingly the drum pattern. This is exemplified by the snapshot shown on the right. Although not shown in the figure, we can also click on the drum pattern display to modify the drum pattern directly, which would also change the latent vector accordingly.

dles essential core functions to build a web server, such as representational state transfer (REST) handling the request from the client. Therefore, we can build the server without any redundant elements but focus on the function of the model. As a result, the server template code has only about 150 lines, excluding the model implementation part.

Client

Several technologies have been built to render the real-time audio output since the Web Audio API was released in 2010 [2]. `Tone.js` (https://tonejs.github.io/) is such a framework for creating interactive music in the browser. It provides simple workflows for synthesizing audio from oscillators, organizing the audio samples, musician-friendly API, and timeline schedule. It makes the development of real-time rendering from the output of the ML model much easier. Added in the new standard HTML5, the HTML canvas element can be used to draw dynamic graphics via writing JavaScript [1]. Thus, we use JavaScript canvas with `Tone.js` to create audio and visual experience coherently.

The modularization is taken care of in the design of the interface (see Figures 2–5). The layout of the user interfaces is implemented as a grid system. This speeds up the design process because it simplifies the choices for the positions of the elements and the margins between them. The recurring elements, such as `pianoroll` and `drum pattern` display, are implemented based on object-oriented principles, thus they can be re-used easily. See Table 1 for a summary.

DEMONSTRATIONS DESIGN

We built four different demonstrations based on the proposed template. We call them 'Latent Inspector,' 'Song Mixer,' 'Comp It,' and 'Tuning Turing.' Each of them was designed to serve one exact purpose and demonstrate a single idea based on the musical machine learning model. The classes of the models are not limited to certain ones. For instance, the first two demonstrate the musical machine learning models based on VAE [13]. In contrast, the last two are mainly based on a recurrent neural network (RNN). Also, the type of instruments could be different. For example, the first one is about percussion and the other three are about melody. This is designed deliberately to show the general purpose of this template. We aim to make them more understandable and interesting by adding interactivity, interface design, and visual effects.

Latent Inspector with DrumVAE

DrumVAE is an original work. It uses VAE for generating one-bar drum patterns. Drum patterns are represented using the `pianoroll` format [6] with 96 time-step per bar. It compresses (or encodes) the drum patterns into a latent space via a bidirectional gated recurrent unit (BGRU) neural network [4]. The outputs from BGRUs are used as mean and variance of a Gaussian distribution. A latent vector is sampled from the Gaussian distribution. We apply the similar but reverse structure of the encoder in the decoder and pass the latent vector into it to reconstruct the drum patterns. It is trained on one-bar drum patterns collected from the Lakh Pianoroll Dataset [5], considering the following nine drums: kick drum, snare drum, closed/open hi-hat, low/mid/high toms, crash cymbal, and ride cymbal.

The Latent Inspector, shown in Figure 2, lets the user modify the latent vector of a drum pattern displayed in the browser to find out how the drum pattern will alter correspondingly. On the other hand, the user can also modify the drum pattern to observe the changes in the latent vectors.

X/Y pads are used in other works to explore the latent space. Yet, the dimension of the latent vectors used in practice is

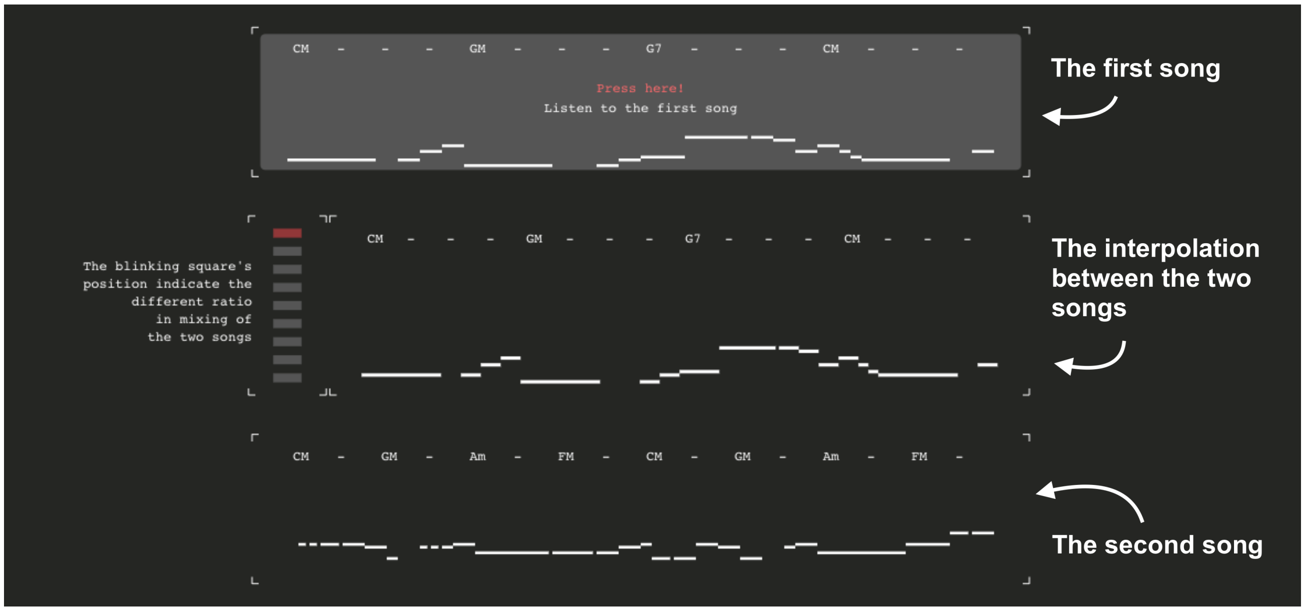
CM - - - GM - - - G7 - - - CM - - -
Press here!
Listen to the first song
The first song
CM - - - GM - - - G7 - - - CM - - -
The blinking square's position indicate the different ratio in mixing of the two songs
The interpolation between the two songs
CM - GM - Am - FM - CM - GM - Am - FM -
The second song

Figure 3. The Song Mixer. The top and bottom panels display the melody and the chords of the first and the second song. In the middle is the interpolation between the two songs. We add visual aids to guide the user to interact with the App. For example, the top panel is highlighted in this figure to invite the user to listen to the first song, before the second song and then the interpolations.

usually larger than two. As a result, we designed a circular diagram which can represent high dimensional data. As shown in Figure 2, the latent vector or our DrumVAE model has dimension $N = 32$. Since the effect of every dimension should be symmetrical in the latent vector of DrumVAE, using circular diagram can eliminate the terminal point of the line chart.

It is possible to further improve the UI by adding conditional functionalities, to give each vertex some musical or semantic meaning. While this can be a future direction, we argue that the current is also interesting— for musicians, it is sometimes more interesting to have a bunch of knob of unknown functionalities to play with.

The demo website of Latent Inspector can be found at http://vibertthio.com/drum-vae-client/public/.

Song Mixer with LeadSheetVAE

LeadSheetVAE is another model we recently developed [14]. It is also based on a VAE, but it is designed to deal with lead sheets instead of drum patterns. A *lead sheet* is composed of a melody line and a sequence of chord labels [14]. We consider four-bar lead sheets here. Melody lines and chord sequences are represented using one-hot vectors and chroma vectors, respectively. It resembles the structure of DrumVAE, but the main difference is that by the end of the encoder the output of the two BGRUs (one for melody and one for chords) are concatenated and passed through few dense layers for calculating the mean and variance for the Gaussian distribution. In the decoder, we apply two unidirectional GRUs to reconstruct the melody lines and chord sequences. The model is trained on the TheoryTab dataset [14] with 31,229 four-bar segments of lead sheets featuring different genres. LeadSheetVAE can generate new lead sheets from scratch, but we use it for generating interpolations here.

The Song Mixer, shown in Figure 3, takes two existing lead sheets as input and shows the interpolations of them generated by LeadSheetVAE. Similarly, a user can modify the melody or chords using the upper panel, or choose other lead sheets from our dataset, to see how it affects the interpolation.

The aim of this demo is to make the interpolation understandable. Therefore, we build interactive guidance with visual cues through the process to make sure the user grasp the idea of lead sheet interpolation. The demo website of Song Mixer can be found at http://vibertthio.com/leadsheet-vae-client/.

Evaluating the quality of interpolations generated by general VAE models (not limited to music-related ones), and many other generative models, has been known to be difficult. A core reason is that there is no ground truth for such interpolations. Song Mixer makes it easy to assess the result of musical interpolations. Moreover, with the proposed template, it is easy to extend Song Mixer to show the interpolation produced by two different models side-by-side and in-sync in the middle of the UI. This facilitates A/B testing the two different models with a user study.

Comp It & Tuning Turing with MTRNNHarmonizer

Finally, MTRNNHarmonizer is another new model that we recently developed.[2] It is an RNN-based model for adding chords to harmonize a given melody. In other words, given a melody line, the model produces a chord sequence to make it a lead sheet. The model is special in that it takes a multi-task learning framework to predict not only the chord label but

[2]More details of the model will be provided in a forthcoming paper.

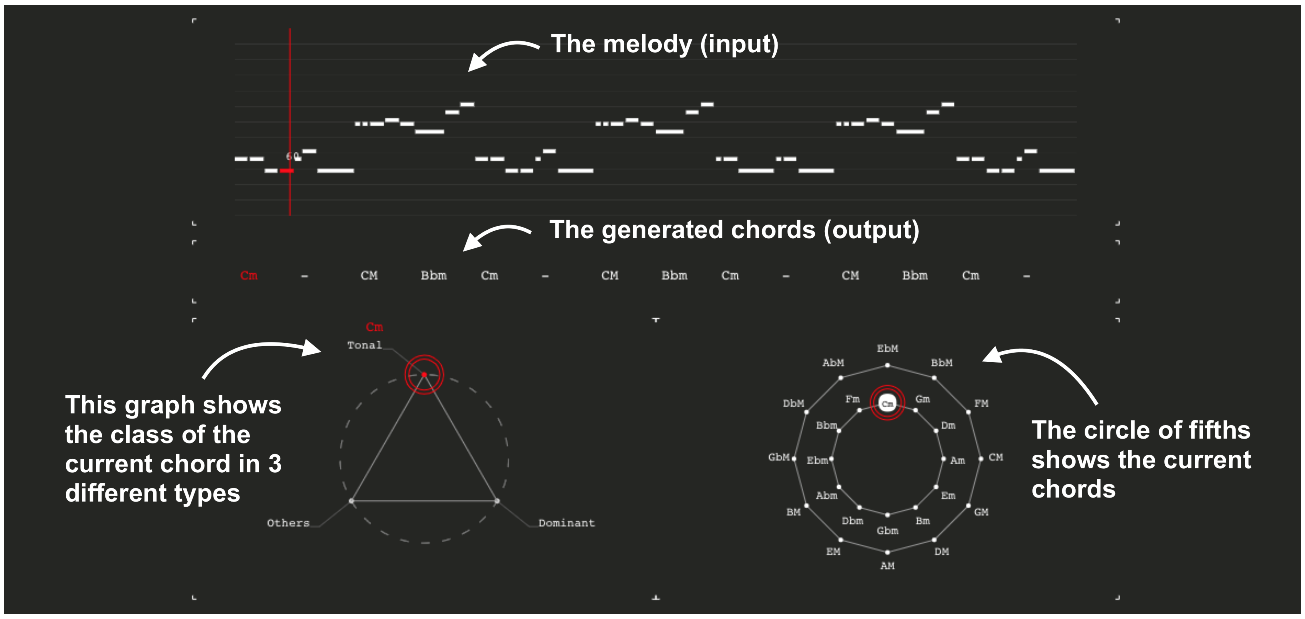
The melody (input)
60
The generated chords (output)
Cm - CM Bbm Cm - CM Bbm Cm - CM Bbm Cm -
This graph shows the class of the current chord in 3 different types
Cm
Tonal
Others
Dominant
The circle of fifths shows the current chords
EbM
AbM
BbM
DbM
FM
GbM
CM
BM
GM
EM
DM
AM
Fm
Cm
Gm
Bbm
Dm
Ebm
Am
Abm
Em
Dbm
Bm
Gbm

Figure 4. Comp It. The upper half is the editable melody and the chords predicted by the underlying melody harmonization model. In the lower left is a graph showing which class the current chord belongs to. In the lower right is a graph showing the position of the current chord on the so-called circle of fifths. The current melody note and chord being played are marked in red.

also the chord's functional harmony, for a given segment of melody (half-bar in our implementation). Taking the functional harmony into account makes the model less sensitive to the imbalance of different chords in the training data. Furthermore, the chord progression can have the phrasing that better matches the given melody line.

Similar to the two aforementioned demos, Comp It allows a user to modify the melodies displayed in the browser to find out how this will alter the chord progression correspondingly. Furthermore, as shown in Figure 4, we add a triangular graph and an animated *circle of fifths* [12] graph to visualize the changing between different chord classes. The triangular graph displays the chord class of the chord being played, covering tonal, dominant, and sub-dominant. The circle of fifths graph, on the other hand, organizes the chords in a way that reflects the "harmonic distance" between chords [3]. These two graphs make it easier to study the chord progression generated by the melody harmonization model, which is MTRNNHarmonizer here but can be other models in other implementations.

Furthermore, we made a simple *Turing game* for the model, called "Tuning Turing." As shown in Figures 5, the player has to pick out the harmonization generated by the model from two music clips. There are both "practice mode" and "challenge mode." The former has 6 fixed levels. In the "challenge mode," the player can keep playing until three wrong answers.

The demo website of Comp It and Tuning Turing can be found at `http://vibertthio.com/m2c-client/` and `http://vibertthio.com/tuning-turing/` respectively.

AVAILABILITY

Supplementary resources including open source code will be available at the GitHub repos (`https://github.com/vibertthio`), including the template (`https://github.com/vibertthio/musical-ml-web-demo-minimal-template`), the interfaces, and the ML models.

CONCLUSION

This paper presents an open-source template for creating an interactive demonstration of musical machine learning on the web along with four exemplary demonstrations. The architecture of the template is meant to be simple and the codebase is small so that other practitioners can implement their models with it within a short time. The modular design makes the musical component reusable. The interactivity and real-time audio rendering of the browser make the demonstration easier to understand and to play with. However, we try to elaborate the quantitative aspects of the project without quantitative analysis. For future work, we will run user studies to validate the effectiveness of these projects. With more intuitive, accessible, and understandable demonstrations of the new models, we hope new people might be brought together to form a larger community to stimulate new ideas.

REFERENCES

1. 2006. Canvas API. MDN Web docs. (2006). `https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API`.
2. 2011. Web Audio API. W3C. (2011). `https://www.w3.org/TR/2011/WD-webaudio-20111215/`.
3. Juan Bello and Jeremy Pickens. 2005. A robust mid-level representation for harmonic content in music signals. In *Proc. Int. Soc. Music Information Retrieval Conf.*
4. Kyunghyun Cho and others. 2014. Learning phrase representations using RNN encoder-decoder for statistical

Demonstration	Modules
Latent Inspector	audio rendering (sample)
	editable pianoroll (drum)
	editable latent vector (circular)
	**radio panel (genre selection)
Song Mixer	audio rendering (synthesize)
	editable pianoroll (melody) × 3
	chord visualization (text)
	radio panel (interpolations selection)
Comp It	audio rendering (synthesize)
	editable pianoroll (melody)
	chord visualization (text, function, circle of fifths)
Tuning Turing	audio rendering (sample)
	waveform visualization

Table 1. Some of the modules are reused by more than one demonstration. For example, three of them uses "editable pianoroll". In our implementation, We reuse the modules to reduce the development effort. Therefore, it is useful and convenient to develop new demos with these modules. ** indicates that a feature that has not been implemented yet.

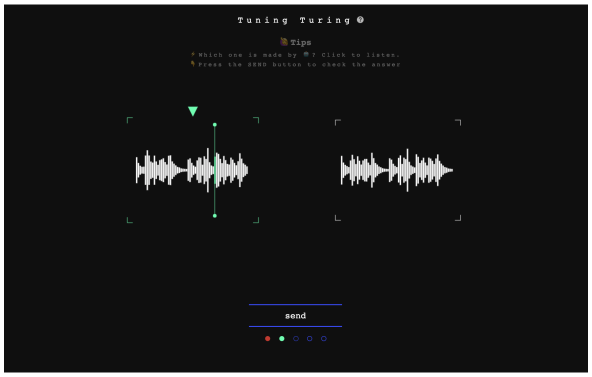
Tuning Turing
Tips
Which one is made by ? Click to listen.
Press the SEND button to check the answer
send

Figure 5. Tuning Turing. Two different kinds of harmonization for a single melody are rendered on the page. The player has to pick out the one generated by the algorithm and send the result after choosing with the mouse.

machine translation. *CoRR* abs/1406.1078 (2014). `http://arxiv.org/abs/1406.1078`

5. Hao-Wen Dong and others. 2018a. MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment. Proc. AAAI Conf. Artificial Intelligence.
6. Hao-Wen Dong, Wen-Yi Hsiao, and Yi-Hsuan Yang. 2018b. Pypianoroll: Open source Python package for handling multitrack pianoroll. In *Proc. Int. Soc. Music Information Retrieval Conf.* Late-breaking paper; `https://github.com/salu133445/pypianoroll`.
7. Hamid Eghbal-zadeh and others. 2018. A GAN based drum pattern generation UI prototype. ISMIR Late Breaking and Demo Papers.
8. Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. 2015. A neural algorithm of artistic style. (2015). `https://arxiv.org/abs/1508.06576`.
9. Ian J. Goodfellow and others. 2014. Generative adversarial nets. In *Proc. Advances in Neural Information Processing Systems*. 2672–2680.
10. Yu-Siang Huang, Szu-Yu Chou, and Yi-Hsuan Yang. 2018. Pop music highlighter: Marking the emotion keypoints. *Transactions of the International Society for Music Information Retrieval* 1, 1 (2018), 68–78.
11. Ciera Jaspan and others. 2018. Advantages and disadvantages of a monolithic codebase. In *Proc. Int. Conf. Software Engineering*.
12. Claudia R. Jensen. 1992. A theoretical work of late seventeenth-century muscovy: Nikolai Diletskii's "Grammatika" and the earliest circle of fifths. *J. American Musicological Society* 45, 2 (1992), 305–331.
13. Diederik P. Kingma and Max Welling. 2014. Auto-encoding variational Bayes. In *Proc. Int. Conf. Learning Representations*.
14. Hao-Min Liu, Meng-Hsuan Wu, and Yi-Hsuan Yang. 2018. Lead sheet generation and arrangement via a hybrid generative model. In *Proc. Int. Soc. Music Information Retrieval Conf., Late Breaking and Demo Papers*.
15. Chien-Yu Lu and others. 2019. Play as You Like: Timbre-enhanced multi-modal music style transfer. Proc. AAAI Conf. Artificial Intelligence.
16. Google Brain Magenta. 2018. Demos. Magenta Blog. (2018). `https://magenta.tensorflow.org/demos`.
17. Brian McFee and others. 2015. librosa: Audio and music signal analysis in python. Proc. 14th Python in Science Conf., 18–25.
18. Alexander Mordvintsev, Christopher Olah, and Mike Tyka. 2015. Inceptionism: Going deeper into neural networks. Google AI Blog. (2015). `https://ai.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html`.
19. Alec Radford, Luke Metz, and Soumith Chintala. 2015. Unsupervised representation learning with deep convolutional generative adversarial networks. (2015). `https://arxiv.org/abs/1511.06434`.
20. Colin Raffel and Daniel P. W. Ellis. 2014. Intuitive analysis, creation and manipulation of MIDI data with pretty_midi. ISMIR Late Breaking and Demo Papers.
21. Adam Roberts and others. 2018a. A hierarchical latent vector model for learning long-term structure in music. (2018). `https://arxiv.org/abs/1803.05428`.
22. Adam Roberts, Curtis Hawthorne, and Ian Simon. 2018b. Magenta.js: A JavaScript API for Augmenting Creativity with Deep Learning. (2018). `https://ai.google/research/pubs/pub47115`.
23. Manuel Ruder, Alexey Dosovitskiy, and Thomas Brox. 2018. Artistic style transfer for videos and spherical images. *Int. J. Computer Vision* (2018). `http://lmb.informatik.uni-freiburg.de/Publications/2018/RDB18`
24. Ian Simon and Sageev Oore. 2017. Performance RNN: Generating music with expressive timing and dynamics. (2017). `https://magenta.tensorflow.org/performance-rnn`.