A Creative Tool for the Musician Combining LSTM and Markov Chains in Max/MSP

Nicola Privato(✉), Omar Rampado, and Alberto Novello

Conservatorio C. Pollini of Padua, Padua, Italy
nicola.privato@gmail.com, omar@ognibit.it,
alberto.novello@conservatoriopollini.it

Abstract. Scramble is a standalone MIDI tool developed in Max/MSP for the real-time generation of polyphonic music, that combines Markov chains and LSTM neural networks. It offers to the performing musician and composer a simplified user interface for the analysis of MIDI files, the creation of models including expressive parameters beside pitch and rhythm, and the interactive control of the generated output.

In this paper we describe and motivate the strategies we implemented for the analysis and representation of the data, and the encoding techniques we resorted to in order to facilitate the detection of low-level relationships whilst saving computational resources. We also describe a representation of pitch and time domains suitable for both the Markov chain and the LSTM modules, and detail the tool's architecture both from a functional standpoint and from the perspective of the user. We conclude by presenting the testing results, by discussing the main limitations of the system and how we intend to address them in future iterations.

Keywords: LSTM neural network · Markov chain · MIDI composer · Max/MSP neural network

1 Introduction

The possible approaches to algorithmic composition by the means of computer-based applications include knowledge-based systems, evolutionary algorithms, Markov chains (MC) and Artificial Neural Networks (ANNs) [11]. The last two methods, at the core of this work, tend to optimally respond to different needs and use cases [2].

The most common ANNs architectures for the generation of music are feedforward networks [8], autoencoders [20], Restricted Boltzmann Machines (RBM) [1] and Recurrent Neural Networks (RNN) [2]. Among the latter category, Long Short-Term Memory (LSTM) neural networks are a subgroup that solves the vanishing gradient problem that characterises Vanilla RNNs, and are thus capable of recognizing underlying relationships in long time series [6,7,9].

T. Martins et al. (Eds.): EvoMUSART 2022, LNCS 13221, pp. 228–242, 2022.
https://doi.org/10.1007/978-3-031-03789-4_15

ANNs can be consistent in their predictions and perform well on generalization tasks. On the other hand, the required computational effort and memory usage are usually high [10]. Furthermore, systems based on ANNs require long training time and large datasets.

On the contrary, thanks to their relative simplicity, MC can provide the user with a greater degree of real-time control over the produced output [3]. This advantage comes with some limitations, most notably an inherent inability to generalize and the risk of plagiarism as the order of the pitch-transitions is raised [15]. Furthermore, by incorporating multiple musical dimensions (e.g. pitch and velocity) inside a single chain, the number of states is increased and the possible transitions are reduced.

In order to take advantage of the specificities offered by both systems we combined one MC controlling pitch transitions with one LSTM for all other musical variables. On playback, a single MC random walk's step is forwarded to the LSTM, which pairs it with the remaining data and triggers the output of the next step. The parallel processing of the MIDI source by the two modules allows to combine different input models, thus producing a wide variety of musical outcomes. At the same time, the serial connection in the output stage grants the musical coherence of the material (see 4).

We implemented this system with *Scramble*[1], a MIDI tool for the real-time generation of polyphonic music that allows the user to access the capabilities of machine learning without the requirement of coding skills. The implementation of Scramble arises out of the need to explore the interaction between artists and autonomous systems [16]. Our aim is to offer a flexible and easy-to-use tool applicable to interactive performance, composition or autonomous music generation. The output can be easily controlled by the user even in a performative environment and includes, beside pitch and rhythm generation, often overlooked elements such as BPM and dynamic variability. At its core, Scramble combines one MC model for pitch state transitioning with a LSTM neural network for time-related and dynamic variables. It also offers a simplified user interface for the analysis of MIDI files, the generation of editable models, and for the real-time playback and adjustment of the MIDI output.

We chose to develop Scramble in Max/MSP[2] since it is a very diffused platform among electroacoustic musicians, in order to explore how more evolved machine learning techniques can be integrated into the already established artistic praxis. Indeed, even though the Max/MSP offers a few basic machine learning libraries ([19], `ml-lib`[3]), the application of more complex algorithms such as LSTM neural network remains for the most part unexplored.

[1] https://www.dropbox.com/sh/v869gify6pm6ta7/AAAcEjlWj-kIJ2VpyNfCWHN7a?dl=0

[2] https://cycling74.com/products/max

[3] https://github.com/irllabs/ml-lib.

Scramble is based on two external modules: an obsolete LSTM developed by Wesley Jackson (2011)[4] and a MC external from the `ml.star`[5] library by Benjamin Day Smith [19]. The LSTM module initially presented some major bugs and design limitations that have been fixed or worked around during Scramble's development. Bug fixes are listed in the provided changelog file[6] and include compatibility with Max 8.0, import and export methods, separate threading implementation, introduction of batch load for all training data.

2 Data Representation

2.1 Pitch

In the case of polyphonic music, pitch is often represented to the learning algorithm in the form of a piano roll where each bit is the active or inactive state of a note in a many-hot encoding vector [12]. In order to cover the whole range of a piano, 88 inputs need to be allocated for pitch. This alone would have exceeded the capabilities of the LSTM module, which is limited by design to a maximum of 70 input and output neurons. Furthermore, such representation is very sparse and produces a class imbalance for most pitches [11]. We instead opted for a multi-voice approach similar to Hadjeres [8]. We considered a maximum polyphony of ten voices, and allocated the notes from the first to the tenth voice starting from the highest one. This approach greatly reduced the number of necessary inputs for the LSTM module, and left enough room to precisely encode the remaining variables.

2.2 Time

Briot et al. [2] distinguish the possible approaches to the temporal representation for LSTM networks into three main categories:

- *Global representation*, usually based on feedforward or autoencoder architectures, where a fixed, pre-established length for the whole output is given (Minibach [2], DeepHear [20]).
- *Time-step*, where the shortest time-step is chosen as the minimum fixed granularity of the temporal subdivision in any other step. This approach, whilst allowing a straightforward sequencing of the temporal dimension, results in the generation of a large amount of data [18].
- *Note-step*, where no fixed time-step is present and granularity adapts to the length of each note [13]. This approach allows for a remarkable reduction in the dataset size.

[4] Starting from J. Franklin's source code (2004), based on F. Gers, N. Schraudolph and J. Schmidhuber pseudocode (2002).
[5] https://www.benjamindaysmith.com/ml-machine-learning-toolkit-in-max/.
[6] https://www.dropbox.com/s/hy3lxel1awzb82f/CHANGELOG.txt?dl=0.

We adopted a note-step approach because global representation is usually performed through feedforward networks instead of RNNs and because its fixed length would have limited the possible uses of the tool. Furthermore, a small number of data points reduces the average training time, which is critical for a tool conceived as a hands-on instrument. Finally, in a time-step approach, slicing any note or chord whose granularity is higher than the minimum fixed would generate two identical states. This has to be avoided in order to preserve the MC variability, since the probabilities of each state transitioning into itself would grow exponentially.

A note-step approach has a major limitation when dealing with pure polyphony: a single time-step value for all of the notes inside a temporal slice allows for a correct time representation only as long as all the notes are equally long. We therefore introduced a note-length value for each individual pitch inside the cluster. Even though this increases the number of dimensions for the vector representing each state, it simplifies the time representation in-between the states by implicitly incorporating ties and rests. For instance, a step of 1 beat at 60 BPM might contain one pitch whose individual note-length is 500 ms, resulting in a rest of a half beat. On the contrary, if one of the notes is 2000 ms long, it is sustained for another 1000 ms even after the next state is triggered.

2.3 MIDI Analyzer

The module responsible for the aforementioned representation of pitch and time domains is the *MIDI Analyzer*. It generates the MIDI pre-processed datasets (MDS) by extracting information from the real-time MIDI playback instead of directly analyzing the files. This approach, at the expense of analysis speed, allows Scramble to potentially connect with live MIDI instruments (see 7).

The real-time stream from the incorporated player is split into indexed states containing all the note-on events occurring inside a 2 to 10 ms span.[7] In parallel, a timer measures the milliseconds between note-on and note-off events. Even if the note-off event happens in the following states, the value is assigned to the state where the measurement started and paired with the pertaining pitch and velocity.

A second timer is responsible for calculating the *step-length*, defined as the time between any two subsequent clusters of note-on events. The BPM rate at the beginning of the state is finally appended. By combining the BPM and step-length, the relative rhythmic value of each state is calculated with (1).

$$R_i = \frac{(t_{i+1} - t_i) \times bpm_i}{60000} \tag{1}$$

where i is the index of the state, t_i is the time of the ith state.

The MDS, editable from the GUI, is used to generate the input data for the MC and the LSTM modules and displays each state as Fig. 1.

[7] The milliseconds span varies depending on the selected analysis speed.

Fig. 1. State, Note_1, Velocity_1, Note_1 Length, Note_2, Velocity_2, Note_2 Length, R(s), BPM

3 LSTM Data Encoding

The combination of the 2011 LSTM module by Wesley Jackson with the Max/MSP environment does not offer high computational performances. We therefore chose to adopt a series of deterministic encoding techniques instead of assigning to the neural network the task of detecting all the low-level relationships in the datasets. In contrast with recent solutions described in Kumar [11], Liang [12], Sun [21], Colombo [4], Yang [23], we adapted to our needs some of the techniques proposed by Mozer [13] and Franklin [5], based on Shepard's [17] studies on psychoacoustics.

3.1 Pitch

Notes are encoded by incorporating their spatial representation on the circle of fifths (Fig. 2). Each note is paired with a combination of six binary units, and one unit is flipped at every next position on the circle. The similarity of two pitches is therefore exponentially related to their distance in the representational space [14].

In order to represent octave heights, piano roll positions are simply normalized from 0 to 1. We chose to implement octave representation through a single real value because more precise information on pitch within the octaves is already conveyed through the circle of fifths [13].

3.2 Rhythm

For the representation of rhythm, we used multiple binary units to encode the rhythmic relationships between states, and single real values to represent less critical information such as individual note lengths and BPM. We partially adopted the approach implemented by Mozer for his CONCERT system, consisting of a five-dimensional space divided into three components, that provides a parallel of Shepard's psychoacoustically-motivated approach to pitch encoding [13,17].

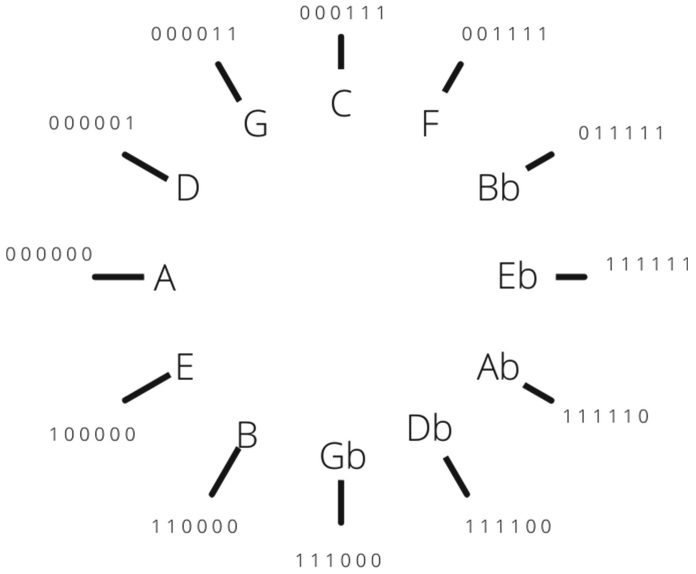

Fig. 2. Spatial representation of pitch encoding.

The duration of each state is conveyed by a total of sixteen binary values, divided into (A) one group of four, (B) one group of three and (C) one group of nine. A and B combined identify the smallest subdivision of each state up to $\frac{1}{12}$ of a single beat. C provides the number of beats to be added. For instance, a total duration of $\frac{18}{12}$ is represented as $\frac{12}{12}$ by C, and $\frac{6}{12}$ by A and B combined. C implements a one-hot nine-dimensional vector to encode up to a maximum musical value of 8 beats. A 4 modulo operation (A) and a 3 modulo operation (B) are applied to the numerator of any value less than or equal to 12. The remainder identifies the value to flip on a one-hot four-dimensional vector for A and a one-hot three-dimensional vector for B. The two vectors combined unequivocally designate a rhythmical value. As Mozer notes [13], this provides similar representations for related durations, since eight-notes (2 on A, 0 on B) and quarter-notes (0 on A, 0 on B) share the same B remainder; eight-note triplets (0 on A, 1 on B) and quarter-note triplets (0 on A, 2 on B) share the same remainder on A; quarter-notes and half-notes share the same remainders on both circles (0 on A, 0 on B) (Table 1).

Table 1. Rhythm encoding. A is modulo 4, B is modulo 3 of Numerator. Musical Value is arbitrarily defined within the minimum available temporal scope.

Numerator	A	B	Musical value
0	0	0	Quarter note
1	1	1	Thirty-two note
2	2	2	Sixteen note triplet
3	3	0	Sixteen note
4	0	1	Eight note triplet
5	1	2	Sixteen note + sixteen note triplet
6	2	0	Eight note
7	3	1	Eight note + thirty-two note
8	0	2	Eight note + sixteen note triplet or quarter-note triplet
9	1	0	Eight note + sixteen note
10	2	1	Eight note + eight note triplet
11	3	2	Eight note + sixteen note + sixteen note triplet
12	0	0	Quarter note

Since the aforementioned system establishes a detailed representation of the rhythmic structure between states, the remaining two time-related dimensions (BPM and individual note-lengths) can be encoded via single non-binary values without compromising the rhythmic structure of the output. Also, since each state contains only one step-length value but may contain up to ten pitches, each one with a given duration, encoding each note-length with sixteen binary values would cause the number of inputs to exceed the system capacity. The MIDI velocities ranging from 0 to 127 (integers) are simply normalized from 0 to 1 (real).

3.3 Machine Learning Datasets

The MDS provided by the MIDI Analyzer is split into two different training sets. The first one (*Pitch Set*) encodes the pitch values as a sequence of tuples structured as described at (2).

$$i, [p(n_{i,1}), o(n_{i,1})], [p(n_{i,j}), o(n_{i,j})] \dots [p(n_{i,10}), o(n_{i,10})] \tag{2}$$

where i is the index of state, p is the pitch encoding function, $n_{i,j}$ is the jth note in the ith state, o is the normalization function of the pitch octave.

The second set (*TD Set*) encodes all time and dynamics related data as a sequence of tuples structured as described at (3).

$$i, [v(n_{i,1}), l(n_{i,1})], [v(n_{i,j}), l(n_{i,j})] \dots [v(n_{i,10}), l(n_{i,10})], R_i, bpm_i \tag{3}$$

where i is the index of state, v is the velocity normalization function, $n_{i,j}$ is the jth note in the ith state, l is the note length normalization function, R_i is the relative rhythmic value of ith state (1), bpm_i is the beat per minute in the ith state.

4 Scramble

Scramble is structured into two main functional parts: a *Training Module* (Fig. 3) and a *Processing Module* (Fig. 4). The former deals with the analysis of MIDI files and the generation of MDSs for the training of the MC and LSTM module, the latter deals with the interaction between the two and with the real-time generation and manual adjustment of the MIDI output.

The MIDI pre-processed dataset generated by the analyzer can be edited, stored and listened to through a secondary player (*MDS Check*), and is used to build the Markov chain and LSTM datasets. The two sets respectively generate or extend pre-existing Markov pitch transitions, and train the LSTM neural network.

Since our aim with Scramble is to provide musicians with a creative tool that allows a high degree of experimentation and because no correspondence to an expected result is sought, we decided to suggest a timer in order to pause the LSTM training beforehand. The user may then check the outcome and if necessary resume training with the same dataset or with a new one. Alternatively, the user can wait until the desired mean square error is reached (see Sect. 5). The model can then be stored and recalled, and may be used as the basis for subsequent training with new datasets.

The Processing Module is where the MC and the LSTM interact. Once activated, the MC routes a single random state of up to ten pitches to the LSTM module, which outputs velocities, note-lengths, step-length and tempo. All the data relative to individual pitches is decoded and forwarded to the MIDI device selected by the user. Step-length and tempo are instead used to determine the time before the next MC output is triggered.

The user can influence the LSTM output by changing the MC pitch state transitions in real time through the MC module. For instance, by using a MC set in a particular musical range it is possible to recall a specific LSTM behavior. We chose pitch as the main control element because it is more straightforward for the user to enter a precise pitch value than velocity or time-related ones.

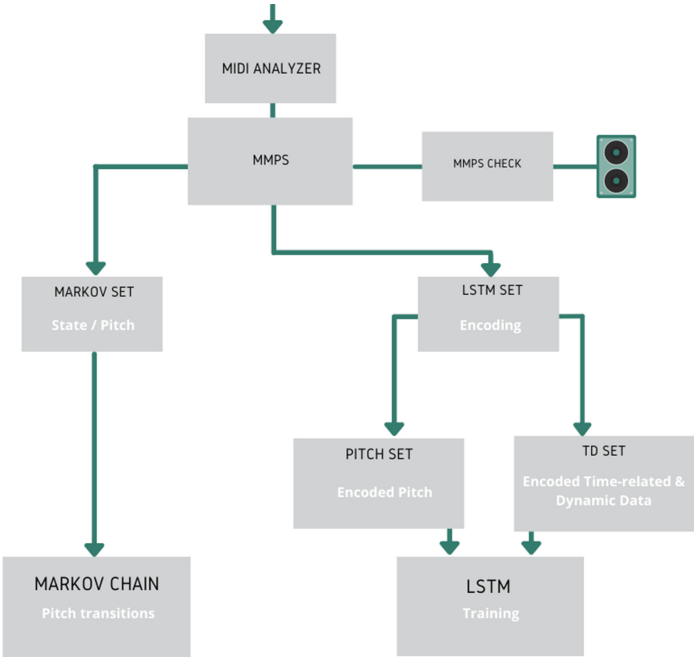

Fig. 3. Scramble training module block diagram.

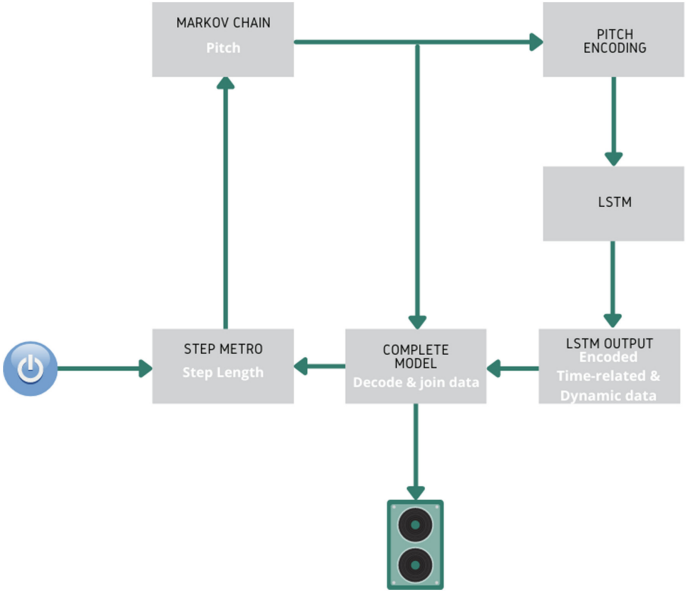

Fig. 4. Scramble processing module block diagram.

5 User Interface

Scramble user interface consists of three horizontal sections from top to bottom. The topmost section (Fig. 5) is dedicated to the analysis of MIDI files. From the central panel, it also provides feedback and instructions on the use of Scramble: by approaching the cursor over the question mark on each sub-menu, a short explanation of the function is visualized on the screen. The user may open any MIDI file and start the analysis at a desired speed. If the original file does not contain BPM information, a default value is automatically assigned to each state of the generated dataset. The value can be manually changed from the *Manual BPM* box. Once the input file is analyzed, the MDS is built in memory. It can be visualized as text, saved, edited, and aurally checked starting from any selected state number.

The central section (Fig. 6) comprises two menus: one for interacting with the MC and dedicated to the generation of pitch transitions, the other for the LSTM model training selection. From the *Markov-Pitch* tab the user may build, edit or save a set, or open an existing one. By pressing the *Build* button, a new pitch-transition table is generated, and by pressing *Grow* any number of MIDI files can be stacked in order to generate complex chains. From the *Order* box, the user may select the number of steps to consider for pitch transitions. All operations on this box can be executed in real time with no interruptions while the system is performing. The *LSTM* menu allows the generation and editing of the training sets (Pitch and TD set) out of the MDS. A timer controls the length of the training, which can be suspended and stored, opened and resumed at will, even with different datasets.

The last section (Fig. 7) is dedicated to the real-time control of the performance and to the user's customization of the LSTM network. At the centre, the *Player* sub-menu allows to select the MIDI device the data will be routed to, and to activate the system. The *Offsets* tab offers the possibility to customize the output by manually entering *BPM*, *Velocity* and *Note Lengths*. All the offsets can be dynamically applied during the performance. The *Settings* menu is dedicated to the customization of the LSTM architecture. The user may experiment with up to 70 blocks and 70 neurons per block. From this menu, it is also possible to change the stop error value. *Alpha* boxes allow the tweaking of the learning rate parameters in input, hidden and output layers. Finally, the *Max Step Length* box restricts the maximum number of beats per each state before the network training section.

6 Experimental Results

The focus of the architecture is to generate creative outcomes including recognizable melodic, rhythmic and expressive patterns learnt from one or more

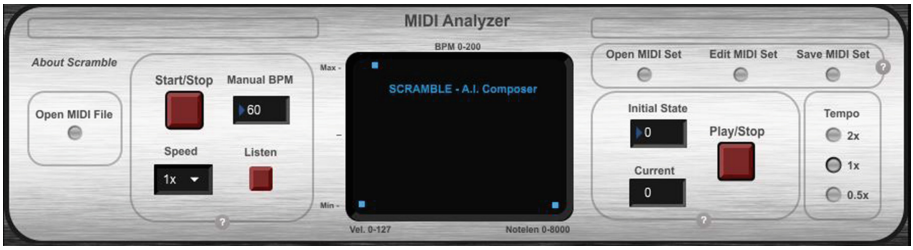

Fig. 5. Scramble user interface, topmost section.

Fig. 6. Scramble user interface, central section.

Fig. 7. Scramble user interface, bottom section.

MIDI files. For this reason the overall results cannot be estimated with a statistical approach based exclusively on the correspondence of the generated data with a defined expectation. To evaluate the overall quality we engaged 6 professional musicians and submitted them a form to fill after using the tool. We informed the participants on how to use the software and asked them to connect the tool's output to a polyphonic MIDI instrument. We provided them with a link[8] to the software (both Windows OS and Mac OS), an introductory video, a short help file and a manual. We also provided 3 pre-built MC

[8] https://www.dropbox.com/sh/v869gify6pm6ta7/AAAcEjlWjkIJ2VpyNfCWHN7a?dl=0.

sets (Arnold Schönberg, Drei Klavierstücke; Vincent Youmans, Tea For Two theme with piano accompaniment; Keith Jarrett, Köln Concert part 1) and 3 LSTM models pre-trained on individual songs (Modest Mussorgsky, Pictures at an Exhibition; Claude Debussy, Clair de Lune; Charlie Parker, Anthropology theme and solo). We chose not to adhere to a specific music style in order to increase the output variability, and to anonymize and unmatch the files in order to avoid expectation biases. The users were free to interact in real time with playback and MC parameters.

We provided three scenarios:

1. *Accompanied interaction*: given three predefined combinations of MC sets and LSTM models among the nine available, the user may freely tweak the parameters.
2. *Unaccompanied interaction*: the user is free to experiment with any combination of the provided sets and models.
3. *Free use (Optional)*: the user is free to train the system with any MIDI file that includes velocity and tempo variations.

The same questions were asked for each scenario. "*How coherent is the generated music (phrasing)?*", "*How coherent is the generated music (form)?*", "*How interesting is the generated music?*", "*How expressive is the generated music (dynamics)?*", "*How varied is the generated music (rhythm)?*", "*How easy is Scramble to use?*", "*How much could you customize the musical output?*". All the answers are in a five-point Likert scale (1 is low and 5 is high).

One additional question was asked to the subjects: "*In which context would you imagine using Scramble?*". The possible answers were "*Artistic Installation*", "*Live Performance*", "*Assisted Composition*". Only one answer accepted.

The results of both the accompanied and unaccompanied surveys displayed in Fig. 8 suggest the overall efficacy of the system in generating coherent and interesting results. Expressivity and rhythm are acceptable but may be improved with a new, less limited LSTM module. The user interface is perceived as easy to use, but the experience may be enhanced by improving feedback and adding graphic details.

The results of the survey on the applicative context of the tool are equally distributed (33.3% each).

Since the optional scenario (free use) is more demanding in terms of time and hardware usage, only three of the subjects replied. The results are: phrasing 4.3, form 3.3, interesting 4.3, dynamics 3.0, rhythm 4.6, easy 3.6, customizable 4.6. The number of responses is low but coherent with those of the mandatory tests.

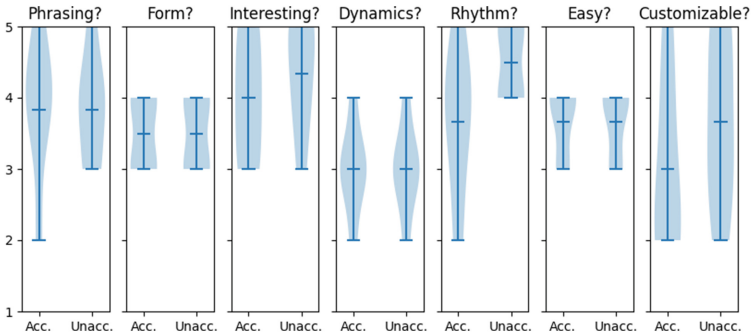

Fig. 8. Distribution of answers for mandatory scenarios. The middle bars indicate the mean values.

7 Conclusions and Future Work

We described Scramble, a generative MIDI composer that combines the advantages offered by Markov chains in terms of live control of the generated output with the ability to generalize offered by LSTM neural networks. Its graphic user interface can be easily and intuitively operated by the musician, allowing for flexibility and dynamic musical experimentation. Scramble combines the pitch transitions generated by a Markov Chain with the velocity, rhythm and BPM information provided by an LSTM neural network trained on the desired musical style or author. We also described the structure of the system and the encoding techniques applied to the input and output data.

As suggested by the survey's results, the tool may prove useful in scenarios such as live audio installations, performance or composition. On the other hand, each specific task may require a more dedicated version. We therefore foresee the development of the present Scramble's iteration in two different directions: one optimized for the interaction required by live performance, one dedicated to autonomous music generation. Scramble's live version is currently under development, it will adapt the MIDI Analyzer input to live polyphonic MIDI sources, and offer the possibility to temporarily store the pitch data into five different buffers optionally controlled by footswitches. The user will therefore be capable of dynamically changing and combining selected transition tables in real time. If on the one hand the present architecture offers the possibility to actively control the MIDI output by changing the transition models, on the other hand the performances offered by the MC on the macro-formal level are quite limited. Scramble's autonomous version will therefore substitute the Markov chain with a second LSTM dedicated to pitch generation [5]. It will also offer a faster MIDI Analyzer extracting MDSs directly from file. In order to allow for some degree of control by the user, options such as plans [22] and reinforcement learning will be also explored [11].

The MC module by Benjamin Day Smith offers a limited number of controls over the pitch state transitions, namely order selection and stacking of multiple

chains. In order to improve the overall performances whilst maintaining the real-time control features that characterize this iteration of Scramble, we foresee the development of a new MC external allowing to incorporate the basic compositional techniques of selectable music styles.

The LSTM module as designed by Wesley Jackson presents some design limitations that could not be fixed: it is constrained to a single instantiation, with fixed size limits on input and output and a limited number of layers and neurons per layer. It also provides poor error evaluation and the lack of performance optimization options results in a lengthened training time. We are therefore currently working on the development of a new LSTM external for Max/MSP, that would solve the design limitations we experienced with the present version, offer more input and output capabilities and overall higher performances. The new LSTM external will substitute the current one in all of Scramble's iterations, and become the core of a future Max/MSP RNN package.

References

1. Boulanger-Lewandowski, N., Bengio, Y., Vincent, P.: Modeling temporal dependencies in high-dimensional sequences: application to polyphonic music generation and transcription. In: Proceedings of the 29th International Conference on Machine Learning (ICML 2012) (2012)
2. Briot, J.P., Hadjeres, G., Pachet, F.D.: Deep Learning Techniques for Music Generation, vol. 1. Springer, Cham (2020). https://doi.org/10.1007/978-3-319-70163-9
3. Cleeremans, A., Servan-Schreiber, D., McClelland, J.L.: Finite state automata and simple recurrent networks. Neural Comput. **1**(3), 372–381 (1989). https://doi.org/10.1162/neco.1989.1.3.372
4. Colombo, F., Muscinelli, S.P., Seeholzer, A., Brea, J., Gerstner, W.: Algorithmic composition of melodies with deep recurrent neural networks (2016)
5. Franklin, J.A.: Jazz melody generation from recurrent network learning of several human melodies. In: FLAIRS Conference, pp. 57–62 (2005)
6. Gers, F.A., Schmidhuber, J., Cummins, F.: Learning to forget: continual prediction with LSTM. Neural Comput. **12**(10), 2451–2471 (2000)
7. Greff, K., Srivastava, R.K., Koutník, J., Steunebrink, B.R., Schmidhuber, J.: LSTM: a search space Odyssey. IEEE Trans. Neural Netw. Learn. Syst. **28**(10), 2222–2232 (2016)
8. Hadjeres, G., Pachet, F., Nielsen, F.: DeepBach: a steerable model for Bach chorales generation. In: International Conference on Machine Learning, pp. 1362–1371. PMLR (2017)
9. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
10. Justus, D., Brennan, J., Bonner, S., McGough, A.S.: Predicting the computational cost of deep learning models. In: 2018 IEEE International Conference on Big Data (Big Data), pp. 3873–3882. IEEE (2018)
11. Kumar, H., Ravindran, B.: Polyphonic music composition with LSTM neural networks and reinforcement learning. arXiv preprint arXiv:1902.01973 (2019)
12. Liang, F.: BachBot: automatic composition in the style of Bach chorales. Univ. Cambridge **8**, 19–48 (2016)

13. Mozer, M.C.: Neural network music composition by prediction: exploring the benefits of psychoacoustic constraints and multi-scale processing. Connect. Sci. **6**(2–3), 247–280 (1994)
14. Mozer, M.C., Soukup, T.: Connectionist music composition based on melodic and stylistic constraints. In: Advances in Neural Information Processing Systems, pp. 789–796. Citeseer (1991)
15. Papadopoulos, A., Roy, P., Pachet, F.: Avoiding plagiarism in Markov sequence generation. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 28 (2014)
16. Privato, N., Novello, A.: Generative scores and data mining: W.E.I.R.D. enters the stage. In: Carvalhais, M., Verdicchio, M., Ribas, L., Rangel, A. (eds.) Proceedings of the 9th Conference on Computation, Communication, Aesthetics X, pp. 126–139. i2ADS (2021). https://2021.xcoax.org/xCoAx2021.pdf
17. Shepard, R.N.: Geometrical approximations to the structure of musical pitch. Psychol. Rev. **89**(4), 305 (1982)
18. Simon, I., Roberts, A., Raffel, C., Engel, J., Hawthorne, C., Eck, D.: Learning a latent space of multitrack measures. arXiv preprint arXiv:1806.00195v1 (2018)
19. Smith, B.D., Garnett, G.E.: Unsupervised play: machine learning toolkit for max. In: NIME (2012)
20. Sun, F.: DeepHear - composing and harmonizing music with neural networks, September 2015. https://fephsun.github.io/2015/09/01/neural-music.html. Accessed 1 Mar 2021
21. Sun, Z., et al.: Composing music with grammar argumented neural networks and note-level encoding. In: 2018 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC), pp. 1864–1867. IEEE (2018)
22. Todd, P.M.: A connectionist approach to algorithmic composition. Comput. Music J. **13**(4), 27–43 (1989)
23. Yang, L.C., Chou, S.Y., Yang, Y.H.: MidiNet: a convolutional generative adversarial network for symbolic-domain music generation. In: Proceedings of the 18th International Society for Music Information Retrieval Conference, pp. 324–331. ISMIR (2017)