

# A-Muze-Net: Music Generation by Composing the Harmony based on the Generated Melody

Or Goren<sup>1</sup>, Eliya Nachmani<sup>1,2</sup>, and Lior Wolf<sup>1</sup>

<sup>1</sup>The Blavatnik School of Computer Science, Tel-Aviv University

<sup>2</sup>Facebook AI Research

**Abstract.** We present a method for the generation of Midi files of piano music. The method models the right and left hands using two networks, where the left hand is conditioned on the right hand. This way, the melody is generated before the harmony. The Midi is represented in a way that is invariant to the musical scale, and the melody is represented, for the purpose of conditioning the harmony, by the content of each bar, viewed as a chord. Finally, notes are added randomly, based on this chord representation, in order to enrich the generated audio. Our experiments show a significant improvement over the state of the art for training on such datasets, and demonstrate the contribution of each of the novel components.

**Keywords:** Music Generation · Midi processing · Recurrent Neural Networks

## 1 Introduction

We present a new method of symbolic music generation called A-Muze-Net. The method employs relatively low-capacity models, such as LSTM networks, and is trained on a relatively small dataset. In order to generalize well despite the lack of training data, it employs various techniques that are inspired by the common practices of human composers.

First, the harmony is composed after the melody is determined, and is conditioned on the melody. Second, the notes are represented in a way that is scale invariant, by considering the gap in pitch between the notes. Third, instead of separating notes to pitch and length, a single token captures both.

A crucial component of the method is that the melody is encoded by considering the notes at each generated bar and identifying the closest chord to this set of notes. Finally, a heuristic that employs the same chord-view adds random notes in order to make the generated audio more complete.

We demonstrate the advantage of our method over existing methods using a collection of 243 Midi files of Bach music. In addition, an ablation study demonstrates the value of each of the above mentioned contributions.

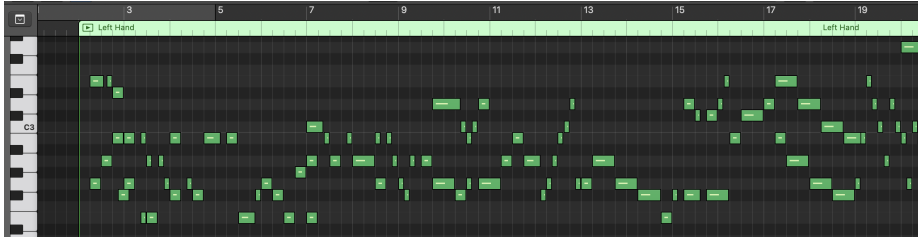
## 2 Related Work

Music generation methods can be divided into a few categories based on the generation domain. Many of the recent works, generate raw audio. **WaveNet** [14] is a convolutional neural network which inputs are raw audio files, and it generates new raw audio files. Since each raw audio timestamp is represented as a 16-bit integer, quantization is applied to reduce the output space [16]. A followup research by Manuzzi et al. [12] employs the same quantization method but employs a biaxial LSTM model [10] which is built as two different LSTM models, one for the time-axis and one for the note-axis. The note-axis LSTM takes two inputs, the previous input as well as the final output from the time-axis LSTM model. **MP3net** [2] generates new mp3 files given mp3 files by using a CNN, their representation is based on the mp3 compression. The most evolved out of these methods is **Jukebox** [4], which compresses the raw audio data to discrete representation and apply high capacity transformer networks [18] to generate songs from many music genres, such as rock and jazz. It is trained on massive amounts of recorded data.

Many of the classical music composition approaches generate music scores [21,15,20], and this line of work has continued into the era of deep learning. **MidiNet** [21] employs a GAN in which both the generator and the discriminator are CNN models, **FlowComposer** [15] which uses Constrained Markov Models [17]. The Part-Invariant model [20] is a single RNN layer model that generates a composition based on an initial part.

Our model generates Midi notes given a prompt that it continues. Recent works that perform the same task include **MuseGan** [6] by Dong et al., which generates novel multi-track Midi files using a GAN model [7] trained on a large scale dataset. A multi-track Midi file contains a separate track for multiple instruments, such as guitar, piano, and drums. It is represented as a **Multi-Track Piano-Roll**. A single Piano-Roll is illustrated in Fig. 1, which is a binary-valued matrix where each row index represent a pitch value and each column index represent a time frame. Dong et al. have later on used a convolutional GAN to generate polyphonic music [5]. Binary neurons are used to generate the binary piano-roll representation, which was found to be more successful than using regular Hard Threshold (HT) or Bernoulli Sampling (BS) as was done in their earlier work.

Boulanger-Lewandowski et al. [1] also used the piano-roll representation but employed the Restricted Boltzmann Machine (RBM) on top of the RNN model in order to generate high-dimensional sequence. The dual-track generator of Lyu et al. [11] generates piano classical music. Similar to our method, it first generates the right-hand part and then the left-hand. In their model, the right-hand is generated by an LSTM, and the left-hand is subsequently generated using a Multi Layer Perceptron. Our left-hand generator is considerably more evolved as it's an LSTM that considers the chord embedding of the right-hand. In addition, while Lyu et al. represent the data as Piano-Rolls, we employ the normalized Midi representation, similar to **BachProp** [3].



**Fig. 1.** Midi representation of prefug3 left-hand track, taken from Complete Bach Midi Index Dataset and opened with Logic Pro software

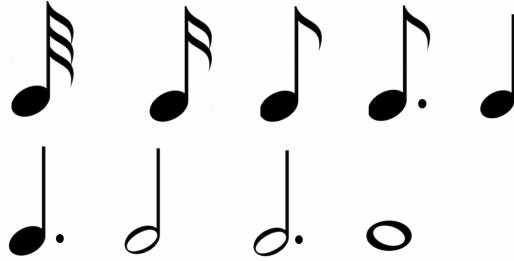
**DeepJ** [13] generates specific music styles: Baroque, Classical and Romantic. Similar to MuseGAN, a piano roll is employed as the underlying Midi representation. They employ a biaxial LSTM model [10], similar to the approach of [12]. Two LSTM models are used, one for the notes pitches and one for the duration of each such note. In order to pass the music genre information an embedding layer is used.

**Music Transformer** [9] is a transformer model with relative attention that generates symbolic music based on multi-track piano-roll representation of Bach’s Chorales dataset. For the Piano-e-Competition dataset they have used the Midi events as their domain. **BachProp** is another LSTM model which is trained on given Midi files and composes new compositions. The normalized Midi representation it employs transforms the Midi into a sequence of notes, each with an associated length. The representation is defined as follows  $note[n] = (T[n], P[n], dT[n])$ , where  $T[n]$  is the duration of the note,  $P[n]$  is the pitch and  $dT[n]$  is the time interval between the current note and the previous one. Their implementation employs three LSTM models, one per each input. **DeepBach** [8] is a deep learning model that uses the Bach’s Chorales dataset, and generates new Chorales like Midi files. They are using only the Chorales, separating these to four different voices, where each voice has a single note at a time. Wu et al. [19] employed a Hierarchical-RNN (HRNN) to generate symbolic music. This was done using a slightly different Midi representation, in which the input domain is the Midi events note-on and note-off, and the time interval since the last event, like [9]. Their HRNN is built from three conditioned RNN models based on the bars, beat and notes.

Our representation is slightly different and employs notes and duration only. Furthermore, we employ a scale-invariant representation, see Sec. 3. In addition, we employ one LSTM for each hand, which are trained subsequently, and do not split the LSTM networks by the type of information of the note tuple.

### 3 Method

We describe the way the midi file is represented, the model we propose, and its training.



**Fig. 2.** All allowed notes lengths, First line from left to right:  $1/32$ ,  $1/16$ ,  $1/8$ ,  $3/16$  and  $1/4$ . Second line from left to right:  $3/8$ ,  $1/2$ ,  $3/4$  and  $1$ . The note images were obtained from <https://www.freepik.com/>.

### 3.1 Midi representation

Given a Midi file, we apply a parser that outputs the note’s pitch and length values for each note in the Midi file for each track. The obtained representation follows closely the music notes representation. The output is a string, in which the alphabet is a sequence of tokens. A sample token is ‘5-X-1/8’, which means note ‘X’ at the fifth octave and length  $1/8$ .

Each token is converted to an integer in the following way. First, we quantize the note’s length into one of the following common values:  $1/32$ ,  $1/16$ ,  $1/8$ ,  $3/16$ ,  $1/4$ ,  $3/8$ ,  $1/2$ ,  $3/4$  and  $1$ , as illustrated in Fig. 2. If, for example, the note’s length is  $11/16$  which is rare, then we assign it to be  $3/4$ .

The note integer representation which the networks employ is the product space of the nine length values and the 128 possible values of a note’s midi-num<sup>1</sup>.

The notes themselves are not taken as absolute notes, such as C (do), D (re), etc. Instead, we represent the Midi data in a way that is invariant to the musical scale used. A Midi file contains the scale information, and we compute each note’s interval in the scale from the first note of the scale.

Two separate sequences are then generated. Specifically, for piano music one sequence is generated for the right-hand and one for the left. In some of the Midi files of the dataset, the separation is not provided. To overcome this, the average pitch for each track is calculated, and the track which has the lowest average pitch is chosen to be the left-hand track, and the maximal average pitch track is chosen to be the right-hand track. This stems from the position of the left hand on the keyboard relatively to the right hand, on the side of the lower notes.

Our representation assumes that there is no more than one note played simultaneously for each track. In case that the input contains multiple simultaneous notes, the right-hand selects the note with the highest pitch, and the left-hand the one with the lowest. Thus, heuristic relies on the observation that in the

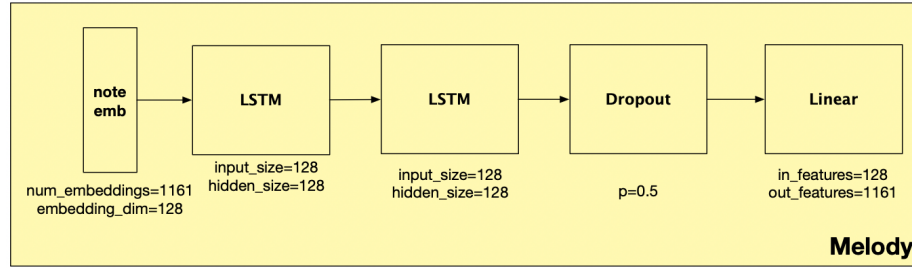
<sup>1</sup> The Midi-num table assigns for each piano keyboard note a number, see <https://computermusicresource.com/midikeys.html>

melody (right-hand) the highest note is more descriptive than the other ones, and vice versa for the harmony (left-hand).

### 3.2 Models

Our method relies on the common practice to compose the melody first and the harmony afterwards<sup>2</sup>. Therefore, the **Right Hand LSTM Model** is applied first. Subsequently, the **Left Hand LSTM Model** is applied conditioned on the output of the **Right Hand LSTM Model**.

The architecture of the **Right Hand LSTM Model** is depicted in Fig. 3. The embedding layer converts each of the possible 1161 integer values of the music representation into an embedding vector in  $\mathbb{R}^{128}$ . This is followed by two LSTM layers with a hidden size of the same dimensionality (128) and there are two layers. This is followed by a dropout layer with a factor of 0.5. Finally, a linear layer projects the LSTM output to a vector of length 1161 that produces the pseudo-probabilities (using softmax) of the next element in the sequence.



**Fig. 3.** Right Hand LSTM Model Architecture

In our method, the harmony generator network is conditioned on a new type of signal we propose, which is the chord analysis of the melody.

The architecture of the **Left Hand LSTM Model** is depicted at Fig. 4. It is similar to the **Right Hand LSTM Model**. The main difference is that a second embedding layer is used. This added embedding, termed the **Chord Embedding Layer** captures the chord that is being played by the right-hand on the current bar. The number of items this embedding encodes is 253 and the embedding dimension is 128. Each of the 253 options encoded a specific combination of notes in the current bar played by the Right Hand Side. Formally, the input of the LSTM after the embedding is:

$$x_{emb} = E_{notes}(x) + E_{chords}(f(R)) \quad (1)$$

<sup>2</sup> See, for example, <https://www.artofcomposing.com/how-to-compose-music-part-3-melody-or-harmony-first>

where  $x_{emb}$  is the LSTM input,  $x$  is the current left-hand note,  $E_{notes}$  is the **Notes Embedding Layer**,  $E_{chords}$  is the **Chords Embedding Layer**,  $R$  is the current right-hand bar's notes and  $f$  is a function that maps a list of notes to their correspondent chord.

For this purpose, we employ a chords hash table that maps the chord's name to its right form, for example the chord **C** is mapped to notes "**C**", "**E**", "**G**". Also we gather groups of 7-chords, 6-chords, 13-chords, 9-chords, diminished chords and augmented chords, by adding/changing the original chord values. For example the chord **Cmaj7** is constructed by "**C**", "**E**", "**G**", "**B**", and **Ddim** is constructed by "**D**", "**F**", "**G**". We apply this method to all possible chords and finally obtain the 253 chords mentioned above.

For recovering the chord associated with a specific bar, we gather all of the notes in that bar, and each chord is scored based on how many notes from a given bar belong to this chord. For example, if the notes are "**C**", "**E**" and "**G**" then chord **Cmaj7** will get a score of 3 and chord **Am** will get a score of 2, However chord **C** will get a score of 13 as these notes are the exact notes within the **C** chord, and it would be picked up. In case of a tie the more common chord would be chosen, i.e., **D** would be picked up rather than **D7**.

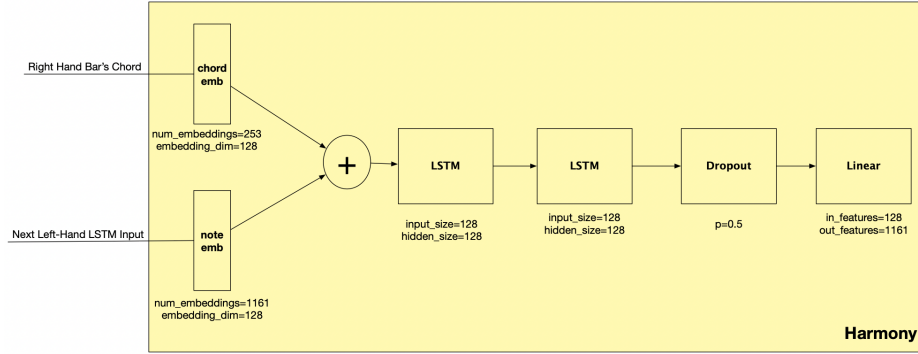


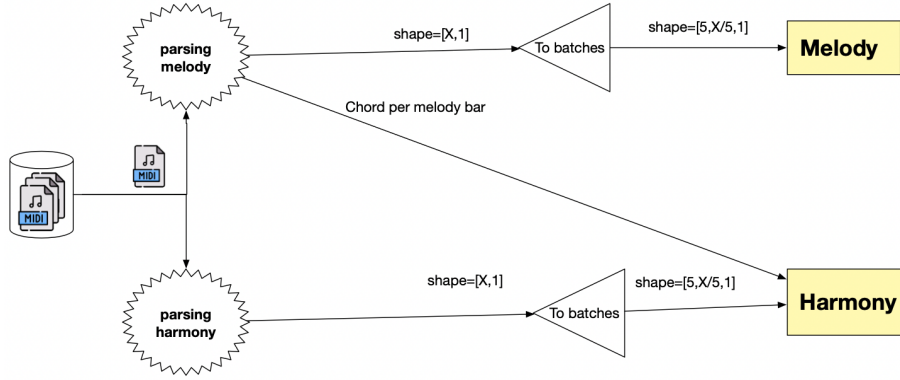
Fig. 4. Left Hand LSTM Model Architecture

### 3.3 A-Muze-Net Model Training

The model training process is depicted in Fig. 5. The given dataset is constructed from multiple Midi files, and we feed one Midi file at a time to the A-Muze-Net Model. The Midi file goes through the parsing methods, and is then divided into batches (the batches do not mix between multiple files).

**Prediction** - The prediction method inputs are:

1. A list of initial Melody notes (right-hand)



**Fig. 5.** A-Muze-Net Model Architecture

2. A list of initial Harmony notes (left-hand)
3. The number of notes to generate

At start the hidden layers of the two LSTM models are initialized and are not being reset throughout the generation part, in order to keep the composition context. The initial melody sequence is fed to the already trained melody LSTM model while using teacher forcing to preserve the specified initial melody. This will set the hidden layer to preserve the initial melody context. Then, the last output from the initial feeding is essentially the first note prediction by the LSTM model. As the output is a probability for each note, only the top-k values are considered and a random value is chosen amongst them, after filtering the "break" notes. Then the chosen note is the next input of the melody LSTM model and again chooses the next predicted note from the next top-k ones. This process is finished when the given **number of notes to generate** is reached.

After this process is finished the chords for each melody bar are calculated and preserved as a sequence for chord per bar.

Then, the Harmony LSTM (Left-hand) is trained the same way as the other one, but the current right-hand bar's chord is passed through to the **Chord Embedding Layer** while the notes are passed through to the **Note Embedding Layer**. The initial sequence is also done in this method with teacher forcing, while the rest of the sequence up until the **number of notes to generate** is done without any teacher forcing.

After both the right-hand track and the left-hand track are generated, we apply a heuristic to add simultaneous notes. For each note in the harmony which is inside the current composition's scale, there is a 50% chance that it'll be accompanied with its perfect-fifth interval note and another 50% chance that it'll be also accompanied by its third interval note which is in the scale.

For example if the current scale is **C** and the current harmony note is **E** then in 50% chance the note **B** (fifth) would be added simultaneously, and another 50% that **G** (third - to form **Em** essentially) would be added.

For out of scale notes the method is different. Instead of adding their fifth or third interval, we add this note again but from a different octave.

For example, if in a composition of scale C the current note is  $4 - B^b$ , for which  $B^b$  is at the fourth octave, then there is a 50% chance that  $5 - B^b$  would be added as well.

For harmony, we apply a similar heuristic. However, instead of a 50% chance of generating simultaneous notes, there is a 10% to generate them since the harmony often generates simultaneous notes to form chords.

## 4 Experiments

For our experiments, we employ the **Complete Bach Midi Index** dataset<sup>3</sup>. It contains approximately 243 Midi files which are divided into several different genre topics, such as chorales, cantatas, fugues and more. The Midi files have different scales and different time-signatures. Most of these Midi files contain two tracks, one for the right-hand (melody) and one for the left-hand (harmony).

As baselines we employ two methods: **MuseGan** [6] and to Lyu et al. research [11]. Since our model is piano-based, we compare our results with the MuseGan piano track.

In addition to comparing with the baselines, we also compare with two ablation variants of our model.

*Ablation 1 - no conditioning of the harmony on the melody* - In this experiment we trained the right-hand (melody) LSTM model and left-hand (harmony) LSTM model separately, meaning that we canceled the **Chords Embedding Layer** from the left-hand LSTM model. This way, both models are independent on one another. After the training was done we generated the Midi files.

*Ablation 2 - note embedding instead of chord embedding* - In this ablation, we maintain the conditioning of the harmony on the melody, but instead of the **Chords Embedding Layer** we simply take the summation of all of the notes' embeddings of the right-hand bar and add them to the current left-hand note's embedding. In other words, we employ the following embedding

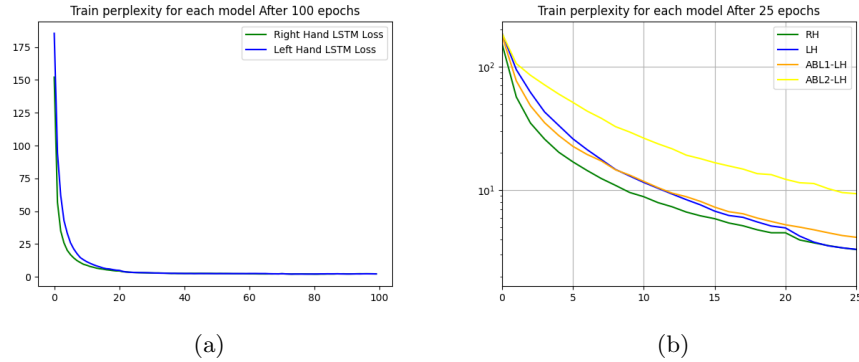
$$x_{emb} = E_{notes}(x) + \sum_{n \in R} E_{notes}(n) \quad (2)$$

Where  $x$  is the current left-track note,  $x_{emb}$  is the embedding output and the input to the LSTM layer,  $E_{notes}$  is the **Notes Embedding Layer**,  $R$  is the list of the correspondent right-hand bar notes.

*Ablation 3 - without the notes addition method* - In this ablation, we remove the part that adds random harmonic notes. In other words, the results are the output of the LSTM models without any further post-processing steps

<sup>3</sup> <http://www.bachcentral.com/midiindexcomplete.html>





**Fig. 6.** (a) Train Perplexity For the Right-Hand LSTM Model and for the Left-Hand LSTM Model. (b) Train Perplexity with Ablation Studies.

The training perplexity of the full method is shown in Fig. 6(a). As can be seen, while the harmony and melody seemingly converged kind of the same way, still the melody converged faster than the harmony.

The training perplexity of the full method compared to the ablation studies training perplexity is shown in Fig. 6(b). One can observe that **ABL1-LH**, which is the Left-Hand LSTM model of ablation study 1 is slightly above our Right-Hand LSTM model and below our Left-Hand LSTM model. This is expected since the **ABL1-LH** is not dependant on the Right-Hand LSTM model, and so its perplexity should be the same as the Right-Hand LSTM model. As for **ABL2-LH**, the obtained loss is considerably above our Left-Hand LSTM model and it takes it longer to converge. This emphasizes the importance of the **Chords Embedding Layer**.

#### 4.1 Results

Following previous work, we consider the following quantitative metrics:

1. **QN (Qualified Note)** - The percentage of notes that were generated with a valid length. For example a note with length lower than  $1/32$  is considered faulty.
2. **UPC (Unique Pitch Class)** - The average amount of different pitches per bar.
3. **TD (Tonal Distance)** - A number that specifies how much the two tracks are aligned chromatically, lower numbers are better.
4. **OOS (Out Of Scale)** - The percentage of generated notes that were out of scale.

Tab. 1 presents the results for our algorithm in comparison to the baselines. As can be seen, our model's **QN** achieves 100%, which means that all notes have a valid length size. This is because we maintain an allowed lengths list that each

generated note has one of these lengths. This way, all of the generated notes are of qualified note lengths, and our method does not get fragments of notes, e.g. notes with length less than  $1/32$ . Also we can see that our model achieves the closest **UPC** value to the True Music UPC value. For the **TD**, where lower values are better, we achieved a lower TD value than **MuseGan**, which means that our tracks are more coherent to each other.

Tab. 1 also depicts the results of the ablation study. Evidently, the UPC values for the ablation methods are lower, which indicates that the tracks are not aligned and do not complete one another. The second ablation study achieved the lowest score which might indicate that the addition of many right-hand notes together with the current left-hand one maybe interfere with the learning method of the model, causing it to generate much fewer notes. We can also observe that the TD values are much higher, and as expected the TD value of the first ablation study is higher than the TD value of the second one, which means that the model with no conditioning at all achieved a worse coherence score.

Our model has a higher OOS percentage in comparison to our ablation studies, which is consistent with the model’s higher UPC value. In a music scale there are only seven notes which are inside the scale, and the five others are considered as out of scale. Since we have UPC value which is higher than seven, we have a high percentage of out of scale notes. Almost all music compositions uses notes from out of scale to generate unique sounds, as is evidenced from the high UPC value of True Music baseline. For example if a composition at scale C uses the **D** chord, it necessarily uses the  $F^\#$  note which is out of scale. Interestingly, the second ablation study achieved 20% OOS although it uses less than seven notes, which reveals a mismatch with the notes being used at the harmony side, pointing to the significance of using the Chord Embedding Layer.

Experiment	QN	UPC	TD	OOS
True Music	98.70%	9.83	-	-
Lyu et al. Pianoroll CNN [11]	91.20%	2.35	-	-
Lyu et al. Embedding Atten-LSTM [11]	90%	7.79	-	-
MuseGan	64%	4.57	0.94	-
A-muse-Net - Ablation1	<b>100%</b>	7.30	0.95	18%
A-muse-Net - Ablation2	<b>100%</b>	6.80	0.90	20%
A-muse-Net - Ablation3	<b>100%</b>	7.70	0.90	18%
A-muse-Net	<b>100%</b>	<b>9.54</b>	<b>0.86</b>	29%

**Table 1.** Quantitative Comparison to outhter methods.

## 4.2 User Study

We asked 17 people to rate the A-Muze-Net generated songs with scores from 1-5 and to state their musical background level, as in **MuseGan** [6] and in Lyu et al. [11]. Each individual listened to ten different clips, which are of length of four-bars, as also was done in the MuseGan research. Tab. 2 presents their average satisfaction out of our generated songs on a scale from one to five.

As can be seen, our method outperforms the baseline methods. Interestingly, while MuseGAN is second by the user rating, it is far lower on the quantitative results.

Experiment	US
True Music	3.80
Lyu et al. Pianoroll CNN [11]	2.40
Lyu et al. Embedding Atten-LSTM [11]	2.70
MuseGan	3.16
A-muse-Net	<b>3.28</b>

**Table 2.** User study results.

## 5 Conclusions

While high-capacity models have now been shown to be able to model music based on very large corpora [4], such models remain computationally inaccessible to most research labs and amassing such data if a copy free way is next to impossible. Furthermore, while such models teach us about AI and large-scale pattern extraction, there is little advancement with regard to the foundations of music making.

In this work, we employ a well established machine learning architecture and try to answer fundamental questions about music representation: (1) How to link the melody and the harmony effectively? (2) How to represent symbolic music in an accessible way? (3) How to capture the transient essence of the melody? (4) How to enrich the generated music?

Our answers to each of these questions have led to an improvement over the options that have been used in the previous work. Collectively, our method provides a sizable improvement in all metrics in comparison to the existing methods.

## Acknowledgements

This project has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant ERC CoG 725974).

## References

1. Boulanger-Lewandowski, N., Bengio, Y., Vincent, P.: Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. arXiv preprint arXiv:1206.6392 (2012)
2. Broek, K.v.d.: Mp3net: coherent, minute-long music generation from raw audio with a simple convolutional gan. arXiv preprint arXiv:2101.04785 (2021)
3. Colombo, F., Gerstner, W.: Bachprop: Learning to compose music in multiple styles. arXiv preprint arXiv:1802.05162 (2018)
4. Dhariwal, P., Jun, H., Payne, C., Kim, J.W., Radford, A., Sutskever, I.: Jukebox: A generative model for music. arXiv preprint arXiv:2005.00341 (2020)
5. Dong, H.W., Yang, Y.H.: Convolutional generative adversarial networks with binary neurons for polyphonic music generation. arXiv preprint 1804.09399 (2018)
6. Dong, H.W., et al.: Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment. In: AAAI (2018)
7. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. NIPS (2014)
8. Hadjeres, G., Pachet, F., Nielsen, F.: Deepbach: a steerable model for bach chorales generation. In: International Conference on Machine Learning (2017)
9. Huang, C.Z.A., Vaswani, A., Uszkoreit, J., Shazeer, N., Simon, I., Hawthorne, C., Dai, A.M., Hoffman, M.D., Dinculescu, M., Eck, D.: Music transformer. arXiv preprint arXiv:1809.04281 (2018)
10. Johnson, D.D.: Generating polyphonic music using tied parallel networks. In: International conference on evolutionary and biologically inspired music and art. pp. 128–143. Springer (2017)
11. Lyu, S., Zhang, A., Song, R.: Dual-track music generation using deep learning. arXiv preprint arXiv:2005.04353 (2020)
12. Manzelli, R., Thakkar, V., Siahkamari, A., Kulis, B.: An end to end model for automatic music generation: Combining deep raw and symbolic audio networks. In: Proceedings of the Musical Metacreation Workshop at 9th International Conference on Computational Creativity, Salamanca, Spain (2018)
13. Mao, H.H., Shin, T., Cottrell, G.: Deepj: Style-specific music generation. In: International Conference on Semantic Computing (ICSC) (2018)
14. van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., Kavukcuoglu, K.: Wavenet: A generative model for raw audio (2016)
15. Papadopoulos, A., Roy, P., Pachet, F.: Assisted lead sheet composition using flow-composer. In: International Conference on Principles and Practice of Constraint Programming (2016)
16. Recommendation, C.: Pulse code modulation of voice frequencies. In: ITU (1988)
17. Roweis, S.T.: Constrained hidden markov models. In: NIPS (1999)
18. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. In: Advances in neural information processing systems. pp. 5998–6008 (2017)
19. Wu, J., Hu, C., Wang, Y., Hu, X., Zhu, J.: A hierarchical recurrent neural network for symbolic melody generation. IEEE Trans. on cybernetics **50**(6) (2019)
20. Yan, Y., Lustig, E., VanderStel, J., Duan, Z.: Part-invariant model for music generation and harmonization. In: ISMIR. pp. 204–210 (2018)
21. Yang, L.C., Chou, S.Y., Yang, Y.H.: Midinet: A convolutional generative adversarial network for symbolic-domain music generation. arXiv:1703.10847 (2017)