# Real-Time Drum Accompaniment Using Transformer Architecture

**Behzad Haki**[*]
Music Technology Group
Universitat Pompeu Fabra
Barcelona, Spain
behzad.haki@upf.edu

**Marina Nieto**[*]
Music Technology Group
Universitat Pompeu Fabra
Barcelona, Spain
marinanietogimenez@gmail.com

**Teresa Pelinski**
Centre for Digital Music
Queen Mary University of London
London, UK
t.pelinskiramos@qmul.ac.uk

**Sergi Jordà**
Music Technology Group
Universitat Pompeu Fabra
Barcelona, Spain
sergi.jorda@upf.edu

## Abstract

This paper presents a real-time drum generation system capable of accompanying a human instrumentalist. The drum generation model is a transformer encoder trained to predict a short drum pattern given a reduced rhythmic representation. We demonstrate that with certain design considerations, the short drum pattern generator can be used as a real-time accompaniment in musical sessions lasting much longer than the duration of the training samples. A discussion on the potentials, limitations and possible future continuations of this work is provided.

## 1 Introduction

Deep generative models in music have been exceptionally successful in (1) learning the underlying musical characteristics of a corpora, and (2) generating stylistically consistent, yet somewhat novel, musical content. Despite their success, in many music generation tasks, these models are noticeably large and rely on large training corpora, therefore, they are financially and computationally costly to train and also to deploy. As a result, the majority of these works mostly focus on offline (non-real-time) content generation. We believe that there is a need to explore whether and how these models can be modified to be deployed in real-time applications.

We are currently working on a line of research specifically aimed at performance oriented real-time drum generation. For this line of work, we have decided to take a bottom-up approach to the design and development of the project. That is, we start with a limited context, and subsequently, if successful, increase the scope of the study. To this end, as an initial attempt, we focused on developing a drum generation system that continuously accompanies an instrumentalist performance in real-time under a number of well-defined conditions: (1) the system should work with any arbitrary instrument, and (2) the system should be designed such that any trained model can be re-utilized without re-training in as many future iterations of the work as possible. Figure 1 illustrates the real-time system envisioned for this work.

In order to meet these conditions, we made a number of decisions. Firstly, to be able to use the drum generator as an accompaniment to an arbitrary instrument, inspired by [1], we decided to base the generations only on a rhythmic interpretation of the input pattern that disregards the pitch information and only takes into account the velocity and timing of more prominent events. Secondly, instead of focusing on arbitrarily long generations, we limited the generations to short patterns with fixed duration; in this manner, we would be able to better study the limitations of the system, and if

[*] Equal Contribution

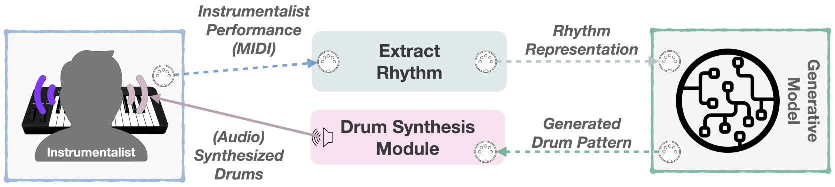


Figure 1: Real-time context envisioned for this work
An instrumentalist performs on a MIDI-enabled instrument of choice. A rhythmic pattern representing the performance is extracted, using which the generative engine suggests an accompanying drum pattern

successful, we would be able to incrementally increase the scope of the generations, possibly with partially re-using the already trained network dealing with shorter generations. As a consequence, we needed to select a model that would work not only for short sequences but also work for much longer ones. Given the recent success of transformer architecture in generating arbitrarily long sequences, we decided to focus on this architecture.

## 2 Related Work

The most relevant, and one of the main inspirations behind this work, is the GrooVAE Tap2Drum model developed by Gillick et. al [1]. This model is a sequence-to-sequence variational auto-encoder network that converts a "tapped" sequence of onsets and their associated timings into a multi-voice human-like performance on a drum kit. Moreover, in [2], McCormack et. al. present an AI drummer that, in real-time, responds to a human instrumentalist improvisation. In this work, the authors investigate the importance and the impact of non-musical communication between an AI accompaniment and an instrumentalist in a real-time improvisational setting. Additionally, in [3], Gómez-Marín et. al. present a system that generates drum patterns by navigating a similarity space constructed by incorporating multiple rhythm similarity measures. Similarly, in [4], Vogl et. al. demonstrate a generative drum machine with a GAN-based generation engine user-controllable by a number of parameters such as genre, complexity, and loudness.

Many variations of the transformer architectures [5-8] have been used for music generation. Some of these works have focused on single instrument score and/or performance generation for a variety of applications including but not limited to guided/unguided generation from scratch, continuation, humanization and score infilling [9-14]. The first work to focus on multi-track generation is probably MuseNet [15], in which a GPT-2 [7] transformer has been trained on multi-track scores from various artists and styles. In another work, Donahue et. al. [16] explore the benefits of pre-training on general datasets for tasks invloving transformers for multi-instrument music generation. Subsequently, in [17], Ens and Pasquier presented Multi-track Music Machine (MMM), a controllable multi-track generative system. Finally, In [18], Nuttall et. al. propose a transformer-XL architecture trained on a small vocabulary of tokens capable of generating long consistent drum performances either from scratch or given a priming pattern.

## 3 Generative Transformer Engine

As mentioned in Section 1, we decided to base the generations only on a rhythmic representation of a pattern played by an instrumentalist. To this end, we start by disregarding pitch information so as to obtain a representation that only consists of a sequence of onsets with their corresponding timings and velocities. To disregard the pitch information, we flatten a performed pattern (which could be melodic, polyphonic or even multi-voice) into a single voice by maximum pooling the velocities of voices/pitches at each time step and then using the velocity and timing information associated with the pooled events - we call the resulting sequence *Monotonic Groove*.

An ideal drum generator should be able to accompany an instrumentalist by both reinforcing and/or complementing the performed instrumental rhythm. This rhythmic function of the drums can vary depending on what type of instrument is to be accompanied. For this work, however, we focus our initial studies on reinforcement of instrumental patterns. The rationale behind this decision was that, as proposed by Gillick et. al. [1], this task can be accomplished simply using input rhythmic patterns derived directly from the drum patterns themselves, removing the need for collecting instrument specific training samples. One additional benefit of this approach is that, the model trained in this setting can be reused within larger models aimed at instrument specific drum generation - i.e. a model trained to predict a monotonic drum groove from a monotonic groove extracted from the accompanied instrument can be paired with the model trained for this work so as to generate drum patterns that have higher awareness of the rhythmic function of the accompanied instrument (see Figure 2).

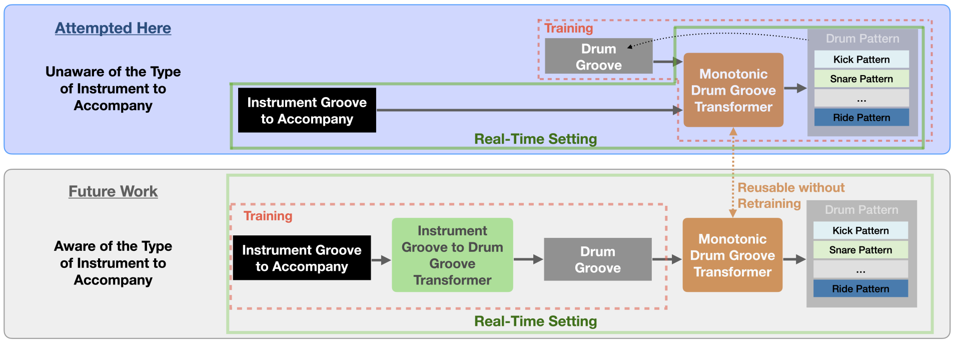

Attempted Here
Unaware of the Type of Instrument to Accompany
Training
Drum Groove
Instrument Groove to Accompany
Monotonic Drum Groove Transformer
Drum Pattern
Kick Pattern
Snare Pattern
...
Ride Pattern
Real-Time Setting
Future Work
Aware of the Type of Instrument to Accompany
Reusable without Retraining
Training
Instrument Groove to Accompany
Instrument Groove to Drum Groove Transformer
Drum Groove
Monotonic Drum Groove Transformer
Drum Pattern
Kick Pattern
Snare Pattern
...
Ride Pattern
Real-Time Setting

Figure 2: Instrument-unaware (top) vs instrument-aware (bottom) systems
In this paper (top figure), the drum generator is trained on a rhythmic pattern derived from a given target drum pattern. In this context, the drum generator learns to *reinforce* an incoming groove. In the future (bottom figure), a second model can be trained to learn a mapping between a specific instrument's rhythmic pattern and the rhythmic pattern of the accompanying drums. By considering the rhythmic function of a specific instrument, the system will be able to both *reinforce* and/or *complement* the instrumental rhythm

The design of the generative transformer model was based on the approach discussed above. In sections 3.1 and 3.2, we discuss the dataset and the methodology used for developing the generative engine. Moreover, in section 3.3, we evaluate a number of developed versions of the transformer model, prior to deploying them in the real-time system.

### 3.1 Dataset and Data Representation

The models were trained on the Groove MIDI Dataset (GMD) [1] (see Table 1). To reduce the dimensionality of the recorded performances, similar to [1], the 22-voice vocabulary of the recordings were mapped onto to a smaller 9-voice set.[2] These patterns were used as target outputs during the training process. Moreover, the input monotonic drum grooves were obtained from these target drum patterns. Lastly, as detailed in Table 2, similar to [1], the input/output data are represented using three stacked matrices (called *HVO*) containing hit, velocity and offset (micro-timing) information.

Table 1: Dataset Overview (2-Bar Beats in 4/4)

| | | Style | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Split[3] | 2-Bar Loops | Rock | Latin | Funk | Jazz | Afrobeat | Hiphop | Other |
| Train (80%) | 16195 | | | | | | | |
| Test (10%) | 2054 | | | | | | | |
| Validation (10%) | 2021 | | | | | | | |
| Total | 20270 | 32% | 17% | 13% | 9% | 5% | 5% | 19% |

Table 2: Input/output sequence representation (2-bar beats in 4/4, 9 drum voices, $16^{th}$ note resolution)

| Tag | Notation | Definition |
|---|---|---|
| Hits | $H_{32\times 9}$ | $h_{ij} \in \{0, 1\}$: drum hit occurrence at time step $i$ for $j^{th}$ voice |
| Velocities | $V_{32\times 9}$ | $v_{ij} \in [0, 1]$: velocity value at time step $i$ for $j^{th}$ voice |
| Offsets | $O_{32\times 9}$ | $v_{ij} \in [-0.5, 0.5)$: onset time deviation from $i^{th}$ $16^{th}$ note gridline for $j^{th}$ voice |
| $HVO$ | $M_{32\times 27}$ | $m_{ij}, m_{i(j+9)}, m_{i(j+18)} = h_{ij}, v_{ij}, o_{ij}$ |

[2] The 9-voice set consists of Kick, Snare, Low/Mid/Hi Toms, Closed/Open Hats, Ride and Crash categories
[3] For this work, the experiments are conducted using the same splits as provided in GMD

### 3.2 Architecture and Training

The transformer architecture used in this work was based on the encoder section of the original transformer [5] (see Figure 3). The network was implemented without any modification, with the exception that we avoided the tokenization of input/output events. As such, we directly used the $HVO$ tensors as input/outputs to the system.

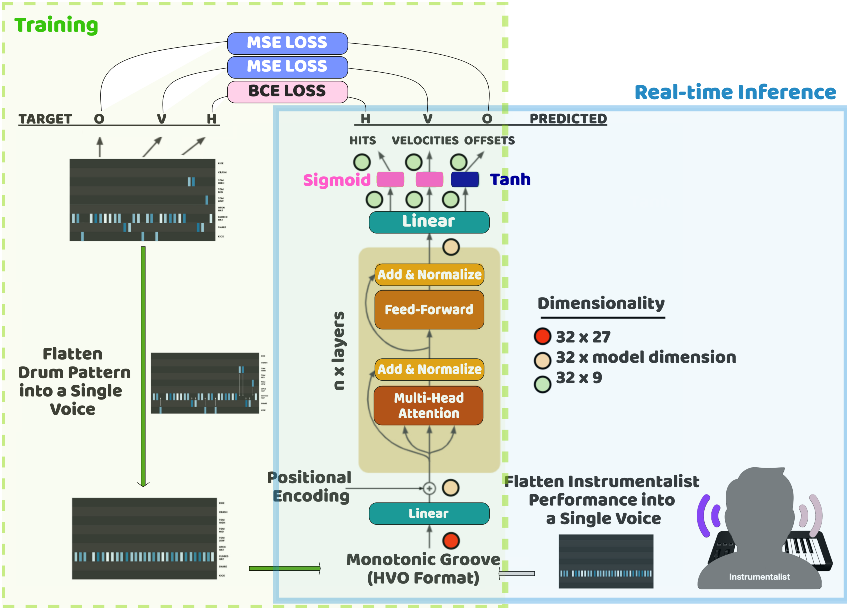


Figure 3: Overview of the model architecture
The transformer is trained to generate a drum pattern only using the flattened version of the same pattern. During inference, a rhythmic representation is derived in a similar manner from an instrumentalist's performance; subsequently, this representation is fed to the generator, using which a drum pattern is generated to obtain a drum performance

Similar to [1], a total loss value is calculated using a binary cross-entropy loss (BCE) for the predicted hits, and Mean Square Error losses (MSE) for predicted velocities and micro-timings. To account for the over-representation of silent non-events in the training set, we scale the loss values calculated at non-hit locations. A stochastic gradient decent (SGD) optimizer with constant learning rate was used to update the network parameters during the training process. The results of our training, our evaluations as well as all our training conditions are fully documented and publicly available to ensure the accuracy and reproducibility of the results presented in this paper.[4]

Over 150 versions of the model with randomly selected hyper-parameter values were trained so as to select a number of candidate models for further evaluation and implementation in the real-time environment. Four models resulting in highest hit prediction accuracy and lowest offset loss were selected as the final candidates. Table 3 summarizes these models.

Table 3: Hyper-parameter Configurations for the Final Selected Models

| | Model 1 | Model 2 | Model 3 | Model 4 |
|---|---|---|---|---|
| model dimension | 512 | 512 | 128 | 128 |
| feed-forward dimension | 16 | 64 | 16 | 128 |
| number of attention heads | 4 | 4 | 1 | 4 |
| number of encoder layers | 6 | 8 | 11 | 11 |

### 3.3 Evaluation

One major challenge in developing generative models for music is the validation and comparison of trained models. While conducting qualitative analysis of the generations using human listeners can

[4] A detailed review of training and source code: `https://github.com/behzadhaki/MonotonicGrooveTransformer`

be illuminating, during the development process, this type of analysis is very costly and impractical as the validation process needs to be conducted on many candidate models.

In this section, we provide a number of objective evaluations conducted on the generations obtained from a select number of models trained. The aim of these evaluations is two-fold: (1) to establish whether the generations are statistically similar to the ground truth data, and (2) how these generations statistically compare against GrooVAE Tap2Drum model.[5] We base these evaluations on a number of rhythm-related features summarized in Table 4 [3, 19- 23].

Table 4: List of features used for evaluation

| Feature | Description |
|---|---|
| NoI | Total number of instruments in sample [3] |
| Total Step Density | Ratio of steps with at least one hit over the total number of steps [3] |
| Average Voice Density | $\frac{1}{32}\sum_{i=1}^{32}\frac{NoI_{t=i}}{9}$ |
| (low/mid/hi)ness | Step density of low (Kick), mid (Snare and Toms), or high (Hats, Ride and Crash) parts divided by total step density [3] |
| Vel Similarity Score | Difference between velocities of the second bar minus the first one |
| Weak to Strong Ratio | $\frac{\text{number of onsets not on downbeats}}{\text{number of onsets on downbeats}}$ [19] |
| Combined Sync | Sum of per voice syncopation values [19] |
| Polyphonic Sync | Polyphonic syncopation measure proposed by Witek et. al. [19,20] |
| (Low/Mid/Hi)sync | Monophonic syncopation of low/mid/high voice groups [3] |
| (Low/Mid/Hi)syness | Ratio of Syncopation to total onsets in each part [3] |
| Complexity | Complexity measure based on mean of density and syncopation [19, 21] |
| AC Skewness, Max, Centroid and Harmonacity | Autocorrelation curve attributes of velocity profiles [19,22,23] |
| Swingness | Measure of the amount of swing in a pattern [19] |
| Laidbackness | Measure of style by incorporating pushed or laid-back onsets [19] |
| Timing Accuracy | Mean of micro-timings of all onsets [19] |

We start the evaluations by analyzing the distribution of hits, velocities and offsets in the generated patterns obtained from our models as well as GrooVAE Tap2Drum. Subsequently, we will provide a relative feature-based comparison between these models. The analysis in this section was done using the validation set of GMD. This portion of data was neither used during the training process nor during the hyper-parameter tuning process.[6]

### 3.3.1 Quality of Predicted Hits, Velocities and Offsets

For any 2-bar pattern, 288 ($= 32 \times 9$) events need to be predicted, with non-hits (silences) significantly out-numbering the hits (onsets). As a result, it is quite important for the models to (1) preserve the proportions of the hits to non-hits and to (2) correctly predict the locations of hits/non-hits.

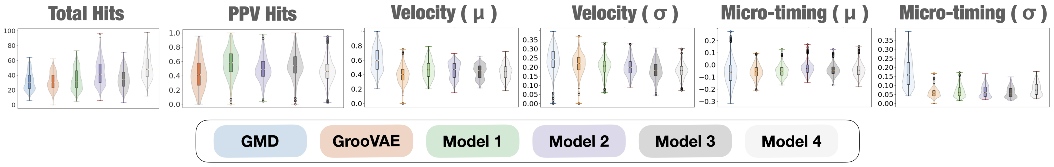


Figure 4: Distributions of hits, velocities and micro-timings (offsets) per sample

As shown in Figure 4, the distributions of total number of hits per each sample in the validation set and the generated sets show that, with the exception of Models 2 and 4, all of the models generate patterns with number of hits close to that of the target patterns. However, looking at the Predictive Positive Value (PPV) of generations (i.e. the ratio of correctly predicted hits to all hit predictions), with the exception of Model 1, in more than half of the generations, more than 50% of the hits are predicted at a location where silence was expected. Moreover, the distributions of mean and standard deviation of the velocities and micro-timings show that all models generate velocities that are lower than that of the ground truth. Lastly, the generated micro-timings seem to be more compressed while

[5] GrooVAE evaluation done using the checkpoint available at `http://goo.gl/magenta/groovae-colab`

[6] Validation set generations: `https://wandb.ai/anonmmi/AIMC2022/reports/sm--VmlldzoxNDc3ODM4`

having a significantly lower standard deviation compared to the target data. These two observations show that our proposed models as well as the GrooVAE model struggle to confidently generalize the velocities and micro-timings of events.

### 3.3.2 Relative Comparison

In [24], Yang et. al. propose an objective evaluation method for generative models in music. In this work, a set of features are extracted from a dataset as well as samples obtained from one or more generative models. Afterwards, the distributions of these features are measured either (1) on their own (absolute analysis)[7], or (2) relative to other samples within the same set or relative to samples within another set. We use the second proposed methodology to study/compare our trained models against each other and also against the GrooVAE Tap2Drum model [1]. For this evaluation, we used 400 random samples from the evaluation set of GMD. Subsequently, using each of the models, we generated 400 samples by passing the monotonic grooves of the target drum patterns through the generative models. The features summarized in Table 4 were then extracted from the target samples as well as the generations. These feature values were then used to construct two sets of distance arrays: (1) Distances of each feature value relative to every other value within the same set (Intra-set), and (2) Distances of each feature value in generated set from all values in target set (Inter-set).

Similar to [24], for each of these distance arrays, a probability density function (pdf) was estimated using a Gaussian kernel and Scott's bandwidth selection method [25, 26]. Finally, for each feature, the distance of the inter-set pdf to the intra-set pdf was calculated using Kullback–Leibler (KL) divergence distance measure. Additionally, the overlapping area (OA) of these two features was also calculated.[7] As show in Figure 5, plotting KL and OA for two sets of generations allows for a visual inspection of the performance of two generative models against one another. Note that in these plots a lower KL and a higher OA value (i.e. closer to the upper left corner) implies better generalization of statistical characteristics of the training set.

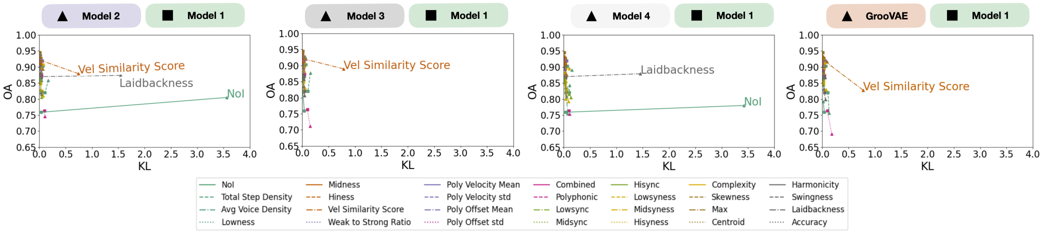


Figure 5: Comparison of KL/OA for inter-set distance PDFs and GMD intra-set distance PDFs

Figure 5 illustrates that, across all features, Model 1 generations are closer to the training set. As a result, this model was selected for deployment in the final real-time system. Moreover, compared to GrooVAE, the generations from Model 1 perform better in terms of Velocity Similarity Score, i.e. the second to first bar symmetry of the generations for Model 1 are closer to the expected target set.

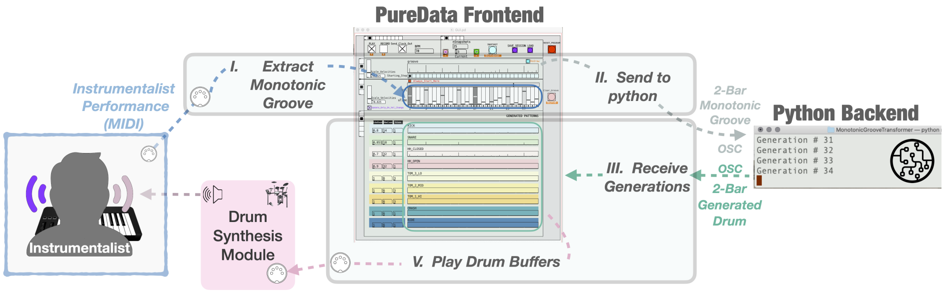
## 4 Real-Time System

For this work, we focused on developing a real-time drum generator that continuously accompanies an instrumentalist. In this context, the generator is required to generate content that "supports" the instrumental performance. As a result, the generator needs to be continuously aware of the state of the performance, meaning that it needs to be not only aware of the performance near the time of generation, but also needs to be aware of the progression of the performance leading up to the time of generation. The transformer model developed for this work has an attention span of 2-bars. This means that the generative model in its current state is inherently incapable of taking into account the events received or generated prior to a given 2-bar segment used for generation. This limited attention span could potentially lead to fragmented and inconsistent generations and hence significantly limit the experience of the user with the system.

[7] Detailed analysis results: `https://wandb.ai/mmil_upf/AIMC2022/reports/Analysis--VmlldzoyMzIyNjEx`

During the design stage of the work, we were aware of this problem, however, as previously mentioned, the intention of this work was to investigate the effectiveness of a "bare minimum" transformer in a real-time accompaniment setting similar to the accompaniment setting shown in Figure 1. As such, we strictly decided to base our initial experiments on a short-term fixed span transformer, and if proven successful, in the future iterations incrementally expand the attention span and the memory of the network. To improve this issue, however, instead of modifying the network to have long-term memory, we decided to provide the model with a summary of past events memorized in the input buffer. Perhaps the simplest, yet highly effective, way of achieving this approach is through "over-dubbing" the previously played performed grooves, rather than providing the model with only two bars of performed groove leading up to the time of generation. With this approach, we can provide the model with a "sense" of the overall "feel" of the performance unfolding over a longer period. To this end, we implemented an internal buffer for registering the extracted monotonic grooves in an overdub mode, meaning that once the system generates a 2-bar pattern using a 2-bar monotonic groove, the content of the groove is preserved in the buffer. In this manner, any given location within the buffer can be overwritten only when a new note corresponding to the same location is received.

The developed real-time system[4] consists of a python backend and a pure-data [27] frontend. The python backend is used for running the generative engine, while the pure-data frontend is used as an interface to allow the user to interact with the system. Additionally, the frontend is in charge of processing incoming *MIDI* performances and preparing the generations for play-back. During a performance, the back-and-forth communication between the user and the drum generator is strictly done through *MIDI*, hence, the user's performance and the generated drums are synthesized using external synthesizers chosen by the user. Moreover, a fixed tempo internal clock within the frontend is used to register incoming notes as well as facilitate the playback of the generated notes. This tempo is adjustable by the user, meaning that the system does not infer the tempo from the user's performance, rather, the user is required to synchronize with the internal clock of the system. To facilitate the synchronization of the user and the system, a click sound is used to allow the user to be aware of the metrical location of the internal clock.



```
Generation # 31
Generation # 32
Generation # 33
Generation # 34
```

Figure 6: Overview of the developed real-time system

As shown in Figure 6, during the performance, the pure-data frontend patch receives a *MIDI* stream of notes from the device on which the user performs. The frontend immediately registers the onset time of a received note so as to identify the closest grid-line. The location of the identified grid-line is then used to store the velocity and the micro-timing (offset) information in separate dedicated buffers of length 32 (corresponding to 2 bars of $16^{th}$-note grid-lines). Simultaneously, using an *OSC* connection [28], the grid-line location, the offset and the velocity information of the received note are transmitted to the python backend. The backend also consists of two dedicated buffers of length 32 for velocities and micro-timings; these buffers (called groove buffers) always reflect the state of their corresponding counterparts in the frontend. If multiple notes are received near the vicinity of a grid-line (i.e. a polyphonic and/or multi-voice input), only the note with largest velocity gets registered.

Whenever a new note performed by the user is received and placed in the internal buffers, the python backend passes the entire content of the groove buffers into the trained transformer model. This forward pass through the model results in a new 2-bar drum pattern which gets immediately sent back to the frontend. The frontend has a dedicated playback buffer for each of the 9 generated drum voices. Locked to the internal clock, the frontend continuously loops through these buffers to play back the generations stored in the buffers.

## 5 Discussion

At this point, we have not yet conducted a qualitative user study involving the system. However, we have done extensive sessions with the system and we believe it is valuable to discuss our observations and provide suggestions on how to continue this line of research in future. A number of these sessions have been recorded and are publicly available.[8]

To test the usability of the system in practical settings, our system was deployed and tested on a consumer laptop[9] and all computations were carried out on CPU. The generation time on our local machines was typically less than 4 milliseconds, resulting in a highly responsive system capable of generating immediate responses to changes in the incoming performance. However, in certain situations in which large number of changes to the incoming pattern was recorded, the computations overloaded the processor, and consequently, interrupted the progression of the master clock and the playback of the sequences; in these situations, we had to actively re-synchronize to the shifted internal clock. To overcome this issue, we allow for manually specifying the minimum duration between two consecutive generations.

Given that the models were not explicitly trained on instrumental patterns, the system tends to generate patterns that feel more of a reinforcement of the input pattern rather than a complement. However, with over-dubbing the input rhythms, as the session progresses, the models can base their generations on a more "complete" (or rather a bigger picture) rhythmic pattern than that of the most recent 2-bar input. As a result, at the very beginning of the sessions, sometimes the generations seemed less confident and more "parallel" to the instrumental performance. As the session progressed, with more variations over-dubbed in the groove buffers, the generations started to form some level of consistency, and at times, even felt not only responsive but also complementary to the performance. On the flip side, as the sessions went on for longer periods (i.e. the groove buffer was overloaded with previously played onsets), the model was less responsive to to the immediate input unless we manually "manipulated" the content of the buffer by playing overly active yet quite patterns.

Another observation was that the system was noticeably responsive to unintended timing errors. If the user performed an event with an unintended timing error, the system would also replicate the same error. This is understandable as the model has no "awareness" about the level of expertise of the performer. Perhaps a user adjustable quantization factor can be beneficial to "correct" unintended errors in these cases. Alternatively, the training set could be enhanced to include randomly corrupted versions of the training samples as well.

Finally, to allow for some level of controllability, we have implemented a manual gain control to boost/lower the velocity values of the events in the groove buffer. Moreover, we allow the user to manually adjust the sampling thresholds of each instrument as well as the maximum allowed density for each drum part. In the next iterations of the work, we will explore the controllability of the generations using a number of high level parameters such as genre, overall loudness and overall density of the generation.

## 6 Conclusion

In creative applications focused on real-time interaction, large architectures are either computationally costly to design and deploy and/or are impractical as they require large number of training data. As a result, for these types of applications, there needs to be an active effort on exploring the potentials of state-of-the-art architectures in limited environments. To this end, in this paper, we presented a real-time drum accompaniment system using a relatively light-weight transformer model. We demonstrate that with certain conscious design choices, a limited model can be affordably deployed in a real-time setting. In our future works, we intend to incrementally expand on the existing approach so as to explore and better understand the potentials and limitations of these models under controlled conditions with strict requirements.

[8] Links to audio/video demos can be found here: `https://github.com/behzadhaki/MonotonicGrooveTransformer`

[9] A 2015 Intel quad-core Macbook Pro with 16GB of ram

# References

[1] J. Gillick, A. Roberts, J. H. Engel, D. Eck, and D. Bamman, "Learning to groove with inverse sequence transformations," in *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, vol. 97 of *Proceedings of Machine Learning Research*, pp. 2269–2279, PMLR, 2019.

[2] J. McCormack, T. Gifford, P. Hutchings, M. T. L. Rodriguez, M. Yee-King, and M. d'Inverno, "In a silent way: Communication between AI and improvising musicians beyond sound," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, CHI 2019, Glasgow, Scotland, UK, May 04-09, 2019*, p. 38, ACM, 2019.

[3] D. Gómez-Marín, S. Jordà, and P. Herrera, "Drum rhythm spaces: From polyphonic similarity to generative maps," *Journal of New Music Research*, vol. 49, no. 5, pp. 438–456, 2020.

[4] R. Vogl, H. Eghbal-Zadeh, and P. Knees, "An automatic drum machine with touch UI based on a generative neural network," in *Proceedings of the 24th International Conference on Intelligent User Interfaces: Companion, Marina del Ray, CA, USA, March 16-20, 2019*, pp. 91–92, ACM, 2019.

[5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 5998–6008, 2017.

[6] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pp. 4171–4186, Association for Computational Linguistics, 2019.

[7] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[8] Z. Dai, Z. Yang, Y. Yang, J. G. Carbonell, Q. V. Le, and R. Salakhutdinov, "Transformer-xl: Attentive language models beyond a fixed-length context," in *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pp. 2978–2988, Association for Computational Linguistics, 2019.

[9] C. A. Huang, A. Vaswani, J. Uszkoreit, I. Simon, C. Hawthorne, N. Shazeer, A. M. Dai, M. D. Hoffman, M. Dinculescu, and D. Eck, "Music transformer: Generating music with long-term structure," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, OpenReview.net, 2019.

[10] K. Choi, C. Hawthorne, I. Simon, M. Dinculescu, and J. H. Engel, "Encoding musical style with transformer autoencoders," in *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, vol. 119 of *Proceedings of Machine Learning Research*, pp. 1899–1908, PMLR, 2020.

[11] Y. Huang and Y. Yang, "Pop music transformer: Beat-based modeling and generation of expressive pop piano compositions," in *MM '20: The 28th ACM International Conference on Multimedia, Virtual Event / Seattle, WA, USA, October 12-16, 2020*, pp. 1180–1188, ACM, 2020.

[12] Y. Chen, Y. Huang, W. Hsiao, and Y. Yang, "Automatic composition of guitar tabs by transformers and groove modeling," in *Proceedings of the 21th International Society for Music Information Retrieval Conference, ISMIR 2020, Montreal, Canada, October 11-16, 2020*, pp. 756–763, 2020.

[13] S. Wu and Y. Yang, "The jazz transformer on the front line: Exploring the shortcomings of ai-composed music through quantitative measures," in *Proceedings of the 21th International Society for Music Information Retrieval Conference, ISMIR 2020, Montreal, Canada, October 11-16, 2020*, pp. 142–149, 2020.

[14] W. Hsiao, J. Liu, Y. Yeh, and Y. Yang, "Compound word transformer: Learning to compose full-song music over dynamic directed hypergraphs," in *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pp. 178–186, AAAI Press, 2021.

[15] C. Payne, "Musenet," *OpenAI Blog*, vol. 3, 2019.

[16] C. Donahue, H. H. Mao, Y. E. Li, G. W. Cottrell, and J. J. McAuley, "Lakhnes: Improving multi-instrumental music generation with cross-domain pre-training," in *Proceedings of the 20th International Society for Music Information Retrieval Conference, ISMIR 2019, Delft, The Netherlands, November 4-8, 2019*, pp. 685–692, 2019.

[17] J. Ens and P. Pasquier, "MMM : Exploring conditional multi-track music generation with the transformer," *CoRR*, vol. abs/2008.06048, 2020.

[18] T. Nuttall, B. Haki, and S. Jorda, "Transformer neural networks for automated rhythm generation," in *Proceedings of the International Conference on New Interfaces for Musical Expression*, (Shanghai, China), June 2021.

[19] F. Bruford, O. Lartillot, S. McDonald, and M. B. Sandler, "Multidimensional similarity modelling of complex drum loops using the groovetoolbox," in *Proceedings of the 21th International Society for Music Information Retrieval Conference, ISMIR 2020, Montreal, Canada, October 11-16, 2020*, pp. 263–270, 2020.

[20] M. A. Witek, E. F. Clarke, M. Wallentin, M. L. Kringelbach, and P. Vuust, "Syncopation, body-movement and pleasure in groove music," *PloS one*, vol. 9, no. 4, p. e94446, 2014.

[21] G. Sioros and C. Guedes, "Complexity driven recombination of MIDI loops," in *Proceedings of the 12th International Society for Music Information Retrieval Conference, ISMIR 2011, Miami, Florida, USA, October 24-28, 2011*, pp. 381–386, University of Miami, 2011.

[22] O. Lartillot, T. Eerola, P. Toiviainen, and J. Fornari, "Multi-feature modeling of pulse clarity: Design, validation and optimization," in *Proceedings of the 9th International Conference on Music Information Retrieval, Drexel University, Philadelphia, PA, USA, September 14-18, 2008*, pp. 521–526, 2008.

[23] M. Panteli, B. Rocha, N. Bogaards, and A. Honingh, "A model for rhythm and timbre similarity in electronic dance music," *Musicae Scientiae*, vol. 21, no. 3, pp. 338–361, 2017.

[24] L. Yang and A. Lerch, "On the evaluation of generative models in music," *Neural Computing and Applications*, vol. 32, no. 9, pp. 4773–4784, 2020.

[25] D. W. Scott and S. R. Sain, "Multidimensional density estimation," *Handbook of statistics*, vol. 24, pp. 229–261, 2005.

[26] B. A. Turlach, "Bandwidth selection in kernel density estimation: A review," in *CORE and Institut de Statistique*, Citeseer, 1993.

[27] M. S. Puckette, "Pure data," in *Proceedings of the 1997 International Computer Music Conference, ICMC 1997, Thessaloniki, Greece, September 25-30, 1997*, Michigan Publishing, 1997.

[28] M. Wright, A. Freed, and A. Momeni, "Opensound control: State of the art 2003," in *New Interfaces for Musical Expression, NIME-03, Proceedings, Montréal, Canada, May 22-24, 2003* (F. Thibault, ed.), pp. 153–159, Faculty of Music, McGill University, 2003.