MusiComb: a Sample-based Approach to Music Generation Through Constraints

MusiComb: a Sample-based Approach to Music Generation Through Constraints

Luca Giuliani
University of Bologna
luca.giuliani13@unibo.it

Francesco Ballerini
University of Bologna
francesco.ballerini4@unibo.it

Allegra De Filippo
University of Bologna
allegra.defilippo@unibo.it

Andrea Borghesi
University of Bologna
andrea.borghesi3@unibo.it

***Abstract*—Recent developments in the field of deep learning have steered research on music generation systems towards a massive use of large end-to-end neural architectures. The capability of these systems to produce convincing outputs has been extensively proven. Nonetheless, they usually come with several drawbacks, such as a low degree of user control, a lack of global structure, and the inherent impossibility of online generation due to high computational costs. Our contribution is two-fold: first, we identify these limitations and show how they have been discussed and partially addressed in the existing literature; then, we propose a novel music generation approach aimed at overcoming such limitations, by properly combining a set of samples under user-defined constraints. We model our task as a job-shop problem, and we show that interesting results can be obtained at very low computational costs. Our framework is genre-independent as it deals with samples metadata rather then individual notes, even though additional genre-specific constraint could be introduced by users to meet their stylistic requirements.**

***Index Terms*—Machine Learning, Constraint Optimization, Music Generation, Human-in-the-Loop**

I. Introduction

The interplay between Artificial Intelligence (AI), creativity and arts has been the subject of many research efforts in recent years [1]. Several directions have been investigated, from the spread of synthetic image generation tools to the ever more pervasive large language models used to create natural language texts. These advances have a big impact on culture and society, as creativity has always been deemed to belong to humans only; hence, a hot debate is currently ongoing. While AI-generated art is very unlikely to supplant human artistic endeavours, such an investigation can provide insights to better understand the human creative processes as well [2]. Recent works demonstrated how a less threatening approach can be considered as well, in particular human creativity can be *assisted* by AI techniques rather than superseded [3].

Another area of artistic creation that has been the subject of intense research from the AI community is the generation of music [4]. Most of the recent works focus on sub-symbolic approaches, exploiting huge amounts of data and very large Deep Learning (DL) models. Conversely, before the advent of DL models, the majority of synthetic music generators were based on symbolic approaches such as Constraint Programming (CP) and Optimization, exploiting the fact that music composition is typically based on a set of precise rules [5].

DL models usually come with several drawbacks, such as a low degree of user control, a lack of global structure, an the impossibility of online generation due to their high computational costs and specific hardware requirements [6]. In this perspective, our contribution is two-fold: first, we identify these limitations and show how they have been discussed and addressed in the existing literature; and second, we propose a novel music generation approach aimed at overcoming such limitations by properly arranging a set of samples under certain constraints. We show that promising results can be obtained at a very low computational cost. We argue that our framework is genre-independent, as it deals with samples metadata rather then individual notes, and it potentially addresses one of the main open challenges of music generation systems, that of online generation. Since we model the task as a job-shop problem, the time required to solve it is low, meaning that it can be potentially used in a live setting. Finally, by integrating these high-level rules with the sub-symbolic composition of each single chunk, we demonstrate that it is possible to achieve a higher-level of diversity at the cost of a smaller time and energy consumption, while also allowing practitioners to intervene in the final composition once it is generated.

II. State of the Art

Computer-aided music generation has a long tradition whose roots trace back to almost two centuries ago. In a series of sketches on the Analytical Engine [7], Ada Lovelace suggests that it would be possible to "compose elaborate and scientific pieces of music of any degree of complexity or extent" provided that the fundamental relations between sounds are correctly encoded. While this strict logical-mathematical approach may seem too restrictive for modern songs, it easily suits the mindset of composers with classical training [8]. For this reason, it is not surprising that many pioneering works on algorithmic composition leveraged rule-based symbolic frameworks. Among all, we mention the famous 1957 composition Illiac Suite [9], which used a generate-and-test algorithm to guarantee constraint satisfaction in the output score for a string quartet, and CHORAL [10], an expert system by Ebcioğlu which adopts more than 350 logical rules to generate multi-voicing harmonization in the style of Bach chorales.

This work has been supported by ICT-48-2020 Project TAILOR funded by European Union's Horizon 2020 programme (GA No. 952215).

After the introduction of Transformer architectures, new systems such as Music Transformer [11] were able to outperform the state of the art thanks to their ability to handle multiple genres while maintaining a coherent structure throughout the generation. Likewise, works like MusicLM [12], Jukebox [13], and Noise2Music [14] managed to generate musical artifacts directly in the audio domain rather then in musical notation using similar architectures. More than 70% of papers published on the topic in the last five years are based on DL [4], while the majority of the remaining ones present evolutionary-based solutions. Nevertheless, large end-to-end neural models for music generation seem to be yet far from achieving the results obtained by their textual and visual counterparts, especially for those systems that aim at generating raw audio, which are still prone to introduce recognizable artifacts such as abrupt timbre changes, choppy tracks and, in general, a massive presence of noisy and dirty signals which prevent from their further use in a professional production.

On a final note, we acknowledge that some recent works have tried to make explicit use of rules and constraints within a larger artificial intelligence framework. For example, [15] proposes a system that can generate polyphonic music with recurring patterns and tension profile by solving an optimization problem based on a model for tonal tension known as the spiral array. Similarly, [16] develops an architecture that combines a Markov model with a finite state automaton to enforce an arbitrary global structure on the generated output by means of appropriate sampling strategies. Finally, [17] defines a novel framework for computer-aided music generation based on two consecutive steps: in the first step, a sub-symbolic model is asked to create short samples with appropriate musical metadata about genre, time signature, chord progression, etc.; then, in the second step, these samples are arranged together to create the final track. Since the authors' contribution is limited to the creation of the dataset and the subsymbolic model, *we build on that by proposing a simple but effective implementation for the combinatorial task.*

III. Motivation

Before introducing our methodology, we will discuss some limitations of the systems presented in the previous section, such as: (a) the difference between classical composition and modern music production, (b) the lack of perceived ownership experienced by practitioners in creative domains when facing AI-based tools, and (c) the inherent speed and robustness issues of large end-to-end neural models.

a) Modern Music Production: Avdeef [18] marks a clear distinction between generative systems for classical music and for pop songs. While composers with classical training are used to reason with rigorous melodic, harmonic, and rhythmic patterns which mainly apply on a local scale, modern music production is strongly influenced by the massive presence of pre-recorded samples, which are selected and assembled together in order to create the final track. As pointed out by Zils and Pachet [19], this style of composition is much more present in electronic music – i.e. Techno, Dance, Hip-Hop, EDM, etc. – but it rapidly became a common practice both in pop and other music genres thanks to the development of new software such as ProTools, Cubase, and other Digital Audio Workstations (DAWs). To the best of our knowledge, there have been very few works that addressed algorithmic composition in this ways; still, we believe that such an approach to algorithmic composition would bring three main advantages. Firstly, since the resulting track is obtained by arranging blocks – i.e. samples – in a two-dimensional grid, it would be possible to modify their position or content, hence allowing to post-process the generated output in a simple way. Secondly, contrarily to the vast majority of research works in algorithmic composition which are tied to a specific style [4], sample-based systems are virtually able to work with all the existing music genres provided that a matching database is given. Thirdly, this methodology mimics the pipeline adopted by commercial software for music production, thus making it easier to take advantage of already established environments.

b) Perceived Ownership: Despite scarce, research works on perceived ownership and user satisfaction during the interaction with generative AI systems show that artists feel a higher sense of ownership when they contribute with a more active role in the creative process. For example, [20] reports that a set of interviewed writers reclaimed less ownership on the generated material whenever paired to an AI assistant which was doing too much work; on the contrary, whenever the assistant was delegated to a more mechanical role, something like a "word calculator", the perceived ownership increased. Additionally, authors also mention that the workload experienced by writers was way smaller when the assistant was used as a "cognitive offloading tool" – i.e. as an augmenting tool to suggest new options in case of creative blocks – rather than when they had to proofread the output it generated from scratch. To the best of our knowledge, [21] was the only one within the music field who partially took this effect into consideration, although in a very brief and qualitative way. Still, this is particularly relatable for those systems that generate raw audio rather than symbolic musical notation. Indeed, these outputs are usually subject to recognizable artifacts and other noisy and dirty signals. This, along with the fact that a single stereo track with all the instruments is produced, makes it almost impossible to post-process it, thus relegating human operators to a filtering rather than a creative role where their main occupation is to throw out all the bad-sounding material until a good generation happens. By switching to a sample-based co-creative approach, we aim to achieve the double benefit of increased sense of ownership and reduced workload.

c) Reliable Generation: Two of the main open issues in the field of computer-based music generation are the high computational requirements and the lack of compliance to global structure. Only few works directly addressed these issues with ad hoc design choices [22]–[24]. All these systems share a constraint-based backbone which grants performers the possibility to steer the generative process at their convenience, with guarantees on the produced output. For example, [22]

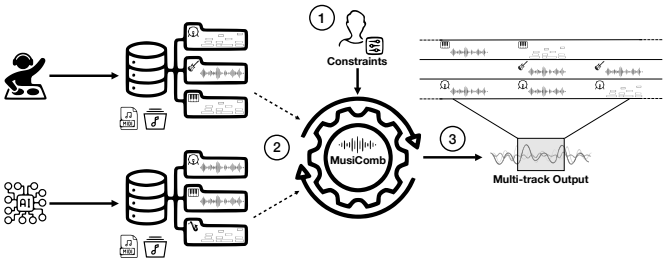

Fig. 1. MusiComb generation pipeline.

allows to impose constraints on song sections in order to obtain a certain global structure, while [24] builds an interactive-by-design system where global continuity constraints can be imposed to create a novel track from pre-existing audio samples. This inspired our CP-based design, as we acknowledge that the adoption of explicit constraints guarantees both a faster and more robust inference, which could potentially allow our system to be used in a live environment.

IV. Method

Our problem formulation is mainly inspired and extends the framework proposed by Hyun et al. [17], who introduced a dataset – dubbed *ComMU* – for the task of *combinatorial music generation*. The task, as proposed by the authors, consists of two consecutive stages, where at first a machine learning model is trained on ComMU and used to generate new samples given the desired metadata, and then the generated samples are combined together into a complete piece of music. In their work, the authors only focused on the first stage. Specifically, they trained a Transformer-XL [25] model on ComMU, and showed the quality of the generated samples. With *MusiComb*, we precisely address the second stage of the task, and extend it with some practical considerations about the nature and origin of the samples. Figure 1 illustrates our proposed pipeline, which is based on the following steps:

1) Users choose the shared metadata, i.e. genre, time, progression of the musical composition, etc...
2) A subset of matching samples is either queried from a database or generated via an artificial intelligence model – in our experiments, we use ComMU as source database and Hyun et al.'s Transformer as generative model.
3) The retrieved samples are arranged together using a Constraint Programming approach.

We stress that the set of matching samples queried in step 2 can be either in MIDI or raw audio format, and either human- or machine-generated, as this differentiates our approach from all the other works that we previously mentioned.

A. CP Model

In order to solve point 3, we model it as a *job-shop problem* where tasks represent samples, and machines represent *track roles*: main melody, riff, sub melody, accompaniment ("acc."), bass, and pad. These roles were chosen as they are part of the metadata available in the ComMU dataset, but different ones can be potentially adopted depending on the metadata available in the samples pool.

We solve the problem via Constraint Programming (CP) to obtain the multi-track output. As displayed in Figure 2, we have a series of jobs that need to be executed, each of which corresponds to a music sample in the queried subset. Samples have their own *start* and *end* times, which are used to enforce the global constraints, namely: (i) each of them must belong to a certain track role, (ii) no overlapping is allowed between samples having the same track role, and (iii) each sample must reach its end before a new one is played. We model this requirements leveraging the $\textsc{Overlap}(\cdot,\cdot)$ constraint, and given the subset S of matching samples we eventually impose:

$$\mathrm{role}(a) = \mathrm{role}(b) \implies \neg\textsc{Overlap}(a,b), \quad \forall a,b \in S$$

namely the overlap between two consecutive samples is forbidden *only* for pairs of samples (a,b) that have the same track role. Samples with different track roles can overlap, but to force a beat-alignment between superimposed samples we force them to start at the same time, i.e.:

$$\textsc{Overlap}(a,b) \implies \mathrm{start}(a) = \mathrm{start}(b), \quad \forall a,b \in S$$

We use a Cumulative constraint to model the importance of each track role and limit the number of tracks that are allowed to play together. This global constraint requires the set S of all the queried samples, a demand value which represents the "cost" of each sample, and an overall $\mathrm{capacity}$ which can be sustained by the final arrangement. For this purpose, we define the demand of each sample $a \in S$ as:

$$\mathrm{demand}(a) = \begin{cases} \texttt{\#main} & \text{if role}(a) \in \{\text{main melody, riff}\} \\ \texttt{\#side} & \text{if role}(a) \in \{\text{sub melody, acc.}\} \\ \texttt{\#back} & \text{if role}(a) \in \{\text{bass, pad}\} \end{cases}$$

In the model we are introducing in this paper, we adopt the following values: $\texttt{\#main} = 3$, $\texttt{\#side} = 2$, and $\texttt{\#back} = 1$. Likewise, we use a total $\mathrm{capacity}$ of 6 to reflect the number of roles that the samples can assume in the original dataset. These values were selected after a preliminary evaluation, and follow the known musical rationale according to which more prominent track roles – main melodies and riffs – should be paired with more background-like ones – basses and pads. Notice that such values force the solver to discard degenerate solutions obtained by arranging all the samples in parallel or in a row, i.e. by playing them all at the same time or one after the other, respectively. The definition of such values is a custom design choice, which can steer the model towards one of the two extreme cases, hence resulting in different outcomes.

The solution of the job-shop problem is eventually obtained by minimizing $\max_{a \in S}\{\mathrm{end}(a)\}$. Indeed, although we are not actually interested in the track being as short as possible, we aim to minimize the latest end time – commonly referred to as *makespan* – so to discard degenerate solutions consisting in samples arranged sequentially.

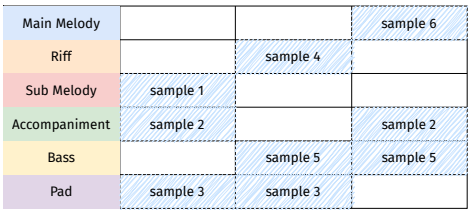

Fig. 2. We arrange music samples in a bidimensional grid where the horizontal axis represents time while the vertical one represents their track role.

Test	BPM	Progression	Genre	Key	Measures	Execution Time ComMU	Execution Time Generated
1	130	Am-F-C-G-Am-F-C-G	New Age	Am	8	3s	14s
2	80	Am-Gmaj7-Fmaj7-G-Cmaj7-Dm7-Am-A#maj7-E+-Am	Cinematic	Am	8	3s	15s
3	120	C-F-Am-G	Cinematic	Cmaj	8	3s	16s
4	100	F-G-Em-Am-F-G-Em-Am	New Age	Cmaj	4	4s	13s

TABLE I
METADATA AND EXECUTION TIME FOR EACH GENERATION TEST.

B. Sample selection

The constraints mentioned in the previous subsection describe *how* samples in the queried subset S are arranged together. However, we have yet to specify *which* samples are included in S, and consequently used as building blocks of the final track. Recalling the pipeline illustrated in Figure 1, in the first step of our framework users are required to specify the input metadata. Then, the subset S of samples sharing those metadata can be obtained either by querying a database of existing samples or by leveraging a conditional generative model able to produce short pieces of music having the required properties – or even both things together.

As we extend the approach of Hyun et al. [17], we focus our analysis to their framework, hence using samples that are either extracted from the ComMU dataset, or generated by their proposed Transformer model. In the former case, the ComMU dataset is queried for one sample for each track role; if there is no sample with that specific combination of track role and metadata, a random sample with the same metadata and different role is picked instead. This process is repeated until 6 samples have been selected – i.e. as many as there are track roles. Accordingly, if samples are produced by the Transformer deep neural network, one sample for each track role is generated. In both cases, 3 out of the 6 samples are repeated throughout the composition: this is achieved by defining a boolean variable for each sample, whose value is set by the solver as part of the optimization process.

V. Empirical Evaluation

To empirically validate our approach, we perform some experiments aimed at assessing the quality of the generated output as well as the computational time required for the generation. In this section, we provide a description of the experimental setting with pointers to musical compositions generated with MusiComb as well as the wall-clock times needed to produce the corresponding program outputs, along with a final discussion and a qualitative analysis. The code and scripts needed to reproduce the results are publicly available at the following link: https://github.com/frallebini/musicomb.

All the experiments were performed on an Intel Xeon Gold 6226R CPU and a NVIDIA Tesla V100S GPU. The code of MusiComb is written in `Python 3.8.5`, leveraging the `mido` library to handle MIDI files. The job-shop problem is implemented with the CP-SAT interface of Google OR-Tools through the APIs offered by the `ortools` package, while for the sample generation process we leverage the `torch` model implemented in Hyun et al. [17], with no changes neither in the architecture nor in the weights. For samples directly queried from ComMU, the execution ran entirely on the CPU; on the contrary, tests providing for the sample generation step ran both on the GPU and CPU, respectively for the neural inference and the constraint solver instance.

Table I reports the metadata used for each test as well as the execution times. All the outputs generated by MusiComb during these experiments are in symbolic musical notation, as both the dataset and the transformer contain and handle MIDI files only. For this reason, the reported times do not consider the overhead introduced by the additional MIDI-to-audio conversion step, which is manually performed leveraging the GarageBand software[1]. The metadata are chosen by the user and shared by the samples which constitute the building blocks of the final composition. We restricted our analysis to "New Age" and "Cinematic" compositions, being them the only two available genres in ComMU, while the other metadata are a subset of the larger set characterizing the dataset. Each test consists in the creation of a single piece of music that matches the given metadata, which can be found and listened to at the following link: https://soundcloud.com/musicomb.

The obtained results demonstrate that the computational cost required by MusiComb makes it a valid candidate for real-time music generation. MusiComb can generate music much faster than other end-to-end neural counterparts, thanks to the efficient CP model. We reckon that the realization that a job-shop problem – a basic and well-studied CP model – can be used to generate realistic musical composition is a non-trivial contribution of this paper. Moreover, the generated outputs effectively mimic the style of the original samples as well as all the other metadata, hence respecting user requirements. This is a precise design choice, as we purposely modelled the task to obtain a novel rearrangement of the initial samples. Moreover, as the original samples all have similar duration and we decided to have 6 samples per generated composition, the final pieces all have a roughly similar duration.

Furthermore, a comparison of the times reported in Table I for the two settings demonstrates MusiComb's capability to allow for the choice of a desired trade-off between output variability and computational requirements. In fact, "ComMU" tests have an average speedup of 350% – 3.25 against 14.5

[1] https://www.apple.com/mac/garageband/, version 10.4.8

seconds – respectively to "Generated" ones, but the latter better exploit the extrapolation abilities of neural generative models to create more varied tracks. Again, faster inference and more reliable outputs might be preferred in live performance environments, while more exploratory behaviours could be favored during the offline compositional work.

On a final note, it is worth mentioning that we were forced to perform the audio synthesis step *after* the score generation for the "Generated" tests, as Hyun et al.'s Transformer is designed to produce symbolic music notation only. On the contrary, for "ComMU" tests we could have achieved the same result – with the exact same execution times and audio output – even if the two phases were swapped, i.e. if we first converted *all* the samples in the dataset using GarageBand and then concatenated them together directly in raw audio format. Nonetheless, since the two approaches are equivalent from a computational viewpoint, we decided to post-process the generated scores to limit the manual workload.

VI. Conclusion

We presented *MusiComb*, a sample-based approach for music generation through constraints. We framed the music composition task as a CP problem, with the goal of arranging multiple samples into a larger musical composition; the samples can be retrieved either from an existing database or created ex novo through DL generation techniques. We showed the feasibility of the proposed approach by generating different musical pieces and demonstrated its low computational requirements and potential of deployment in real-time applications, given its ability to generate new tracks much faster than they are reproduced. Still, we reiterate that this is a preliminary study, hence multiple extensions are possible.

In future works we plan to explore several research directions. The natural follow-up is to perform analysis to subjectively evaluate the generated music. This can be done by recruiting participants willing to assess the quality of the musical compositions, ideally including both artists and laymen. We will then apply MusiComb to (1) different sample datasets and (2) different sample generator. This would allow for a broader evaluation of the system. Similarly, our approach could be used to combine even more track as output, if the original samples include them; we will investigate this as well, with a focus on scalability, as more tracks correspond to a more complex CP model.

As we noticed, MusiComb is quick enough to be used in real time. Altough we tested it using a relatively high-end graphical card (NVIDIA Tesla V100S GPU), the GPU is needed only in the sample generation phase due to the computational requirements of the Transformer network. This process is not mandatory in our approach, as the samples can theoretically be obtained by querying already existing pools. Moreover, in the case of synthetic generation using complex deep neural networks, this phase can be performed offline, with no real-time requirement, thus without the need of top-performing computational resources. For this reason, we aim at running additional experiments using consumer-grade hardware for the search of the solution to the CP model, as this is the only phase that is expected to be run online.

References

[1] S. Shahriar, "Gan computers generate arts? a survey on visual arts, music, and literary text generation using generative adversarial network," *Displays*, p. 102237, 2022.

[2] N. Anantrasirichai and D. Bull, "Artificial intelligence in the creative industries: a review," *Artificial intelligence review*, pp. 1–68, 2022.

[3] A. D. Filippo, L. Giuliani, E. Mancini, A. Borghesi, P. Mello, and M. Milano, "Towards symbiotic creativity: A methodological approach to compare human and AI robotic dance creations," in *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*. International Joint Conferences on Artificial Intelligence Organization, Aug. 2023.

[4] M. Civit, J. Civit-Masot, and et al., "A systematic review of artificial intelligence-based music generation: Scope, applications, and future trends," *Expert Systems with Applications*, vol. 209, p. 118190, dec 2022.

[5] O. Laske, "Composition theory: An enrichment of music theory," *Journal of New Music Research*, vol. 18, no. 1-2, pp. 45–59, 1989.

[6] S. Dadman, B. A. Bremdal, B. Bang, and R. Dalmo, "Toward interactive music generation: A position paper," *IEEE Access*, vol. 10, pp. 125 679–125 695, 2022.

[7] L. F. Menabrea and A. K. C. of Lovelace, *Sketch of the Analytical Engine Invented by Charles Babbage, Esq*. Richard and John E. Taylor, 1843.

[8] T. Anders, *Compositions Created with Constraint Programming*, R. T. Dean and A. McLean, Eds. Oxford University Press, feb 2018.

[9] L. A. Hiller Jr and L. M. Isaacson, "Musical composition with a high speed digital computer," 1957.

[10] K. Ebcioğlu, "An expert system for harmonizing four-part chorales," *Computer Music Journal*, vol. 12, no. 3, p. 43, 1988.

[11] C.-Z. A. Huang, A. Vaswani, J. Uszkoreit, N. Shazeer, C. Hawthorne, A. M. Dai, M. D. Hoffman, and D. Eck, "Music transformer: Generating music with long-term structure," *arXiv preprint arXiv:1809.04281*, 2018.

[12] A. Agostinelli, T. Denk, and et al., "Musiclm: Generating music from text," 2023. [Online]. Available: https://arxiv.org/abs/2301.11325

[13] P. Dhariwal, H. Jun, C. Payne, J. W. Kim, A. Radford, and I. Sutskever, "Jukebox: A generative model for music," 2020.

[14] Q. Huang, D. S. Park, and et al., "Noise2music: Text-conditioned music generation with diffusion models," *preprint arXiv:2302.03917*, 2023.

[15] D. Herremans and E. Chew, "MorpheuS: Generating structured music with constrained patterns and tension," *IEEE Transactions on Affective Computing*, vol. 10, no. 4, pp. 510–523, oct 2019.

[16] F. Pachet, A. Papadopoulos, and P. Roy, "Sampling variations of sequences for structured music generation." in *ISMIR*, 2017.

[17] L. Hyun, T. Kim, H. Kang, M. Ki, H. Hwang, K. Park, S. Han, and S. J. Kim, "ComMU: Dataset for combinatorial music generation," in *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022.

[18] M. Avdeeff, "Artificial intelligence & popular music: SKYGGE, flow machines, and the audio uncanny valley," *Arts*, vol. 8, no. 4, p. 130, oct 2019. [Online]. Available: https://doi.org/10.3390%2Farts8040130

[19] A. Zils and F. Pachet, "Musical mosaicing," in *Digital Audio Effects (DAFx)*, vol. 2, 2001, p. 135.

[20] K. I. Gero and L. B. Chilton, "Metaphoria," in *Proc. of the 2019 Conference on Human Factors in Computing Systems*. ACM, 2019.

[21] R. B. Tchemeube, J. Ens, C. Plut, P. Pasquier, M. Safi, Y. Grabit, and J.-B. Rolland, "Evaluating human-AI interaction via usability, user experience and acceptance measures for MMM-c: A creative AI system for music composition," in *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*. International Joint Conferences on Artificial Intelligence Organization, Aug. 2023.

[22] M. Marchini, F. Pachet, and B. Carré, "Rethinking reflexive looper for structured pop music." in *NIME*, 2017, pp. 139–144.

[23] F. Pachet, P. Roy, and B. Carré, "Assisted music creation with flow machines: towards new categories of new," *Handbook of Artificial Intelligence for Music: Foundations, Advanced Approaches, and Developments for Creativity*, pp. 485–520, 2021.

[24] J.-J. Aucouturier and F. Pachet, "Jamming with plunderphonics: Interactive concatenative synthesis of music," *Journal of New Music Research*, vol. 35, no. 1, pp. 35–50, mar 2006.

[25] Z. Dai, Z. Yang, and et al., "Transformer-XL: Attentive language models beyond a fixed-length context," 2019.