DOI: 10.58895/ksp/1000162754-14 erschienen in:
Proceedings. 33. Workshop Computational Intelligence - Berlin, 23. – 24. November 2023
DOI: 10.5445/KSP/1000162754 | https://www.ksp.kit.edu/site/books/e/10.5445/KSP/1000162754/

Adaptive video game music as a multi-objective benchmark for conditional autoregressive models

Fabian Ostermann

Technische Universität Dortmund
Otto-Hahn-Str. 14, 44227 Dortmund, DE
E-Mail: fabian.ostermann@tu-dortmund.de

1 Introduction

The aim of this paper is to investigate and discuss the use of generative neural networks to reconstruct handcrafted adaptive music. We choose the dynamic compositions from the 1991 video game *Monkey Island 2* [1], which has consistently been recognized as a role model and masterpiece in this field [2, 3]. The music in this game is generated in real-time during gameplay, leading us to define our task as the successive prediction of note events using autoregressive models. Furthermore, the game's music adapts to player actions, so the generative task additionally is conditional.

In Section 2, we explain that music data contains very different types of information and show how similarly constructed statistical errors can have very different effects according to cognitive music perception, which brings additional complexity to this problem domain. In Section 3, we explain how the dataset was obtained and what attributes it has. The dataset generator itself is *openly available*[1] for further academic or educational use. We argue that, on the one hand, music generation is a *fun* problem domain without ethical issues, which has the potential to motivate people to engage with AI technology. On

[1] `https://github.com/fabianostermann/WoodtickWalkingSimulator`

the other hand, it really makes a challenging benchmark for complex multi-task learning concepts and multi-objective optimization strategies. Moreover, coping with the adaptive real-time aspect of video game music is exceptionally difficult and, to our knowledge, poses a unique challenge. For our practical investigation, Section 4, we define seven problem dimensions specific to this task. For each dimension, we conduct experiments and discuss the unique challenges associated with processing music data. In the process, we propose areas for further investigation, which will be summarized in Section 5.

2 Music as a problem domain

Music is a complex matter. Its description needs a lot of dimensions [4], which makes it vulnerable to combinatorial explosion. All attempts of symbolic representation, like modern western notation, need some form of severe simplification.[2] The example of MIDI is used to explain which minimum set of dimensions is required.

MIDI [5] is a widely used music data communication protocol that was also used for the music of *Monkey Island 2*. MIDI was designed as a standard to connect all kinds of electronic musical instruments. It defines atomic musical instructions called *MIDI events* that carry byte-sized information about pitch, timing and loudness[3]. The beginning and the end of a note are divided into two different events. The desired instrument sound is implicitly set. The MIDI stream consists of 16 different channels. A so-called *program change* event is sent to a channel to request an instrument change. A special case is channel 10, which is used for drums, and where the pitch information is remapped to specify which drum instrument to trigger (e.g., snare, bass drum, hihat).

MIDI and its cluttered specification appears a bit dated by today's standards. Its main advantage is its wide popularity. A lot of MIDI-encoded music exists and can be directly used to build machine learning datasets [6]. However, it is

[2] An exception may be the approach to process music at the audio signal level [7, 8].

[3] Be aware that the loudness of a note in the context of MIDI usually is referred to as *velocity*, because the velocity of pressing down a key on a piano keyboard determines its loudness. We stick to the term *loudness* here since velocity can easily be misinterpreted as a temporal property.

usually not practical to use MIDI events directly as input data for classification or generation tasks.[4] Therefore, it is necessary to convert a stream of MIDI events into a sequence of semantic features.

We choose to define every note as a 5-tuple

$$note_i = (\Delta\text{time},\ \text{duration},\ \text{loudness},\ \text{pitch},\ \text{instrument}) \tag{1}$$

and a music sequence of length n as $seq = \{note_1, note_2, \ldots, note_n\}$. The time difference Δtime is calculated as the distance in seconds to the previous note event. The duration is the distance of beginning and end of the note in seconds. The loudness is the MIDI velocity value normalized to [0,1]. The pitch is one of $\{0, 1, ..., 127\}$ which linearly maps to the keys of a piano, where 60 denotes the middle C. The instrument is an integer label determined by searching in the MIDI stream for the last *program change* event that occurred on the same MIDI channel.[5]

Note that this scheme is one possibility of many and that the concrete encoding is usually important [9, 10]. Also note that the different types of information are not equally important to the cognitive perception of the music, since, e.g., Δtimecan be allowed to vary by a few milliseconds without the perception of rhythmical flaws while missing a pitch by only a semitone[6] instantly results in severe harmonic dissonances. Changes in duration, loudness or instrument are rather perceived as variations than mistakes. Among all, Δtimeand pitch are (by far) the most important features to re-recognizing music.

That said, music makes a complex multi-objective learning problem that mixes regression (Δtime, duration, loudness) and classification (pitch, instrument) tasks. Therefore, it can be approached by multi-task learning, which was already applied to music with the most success in music transcription [11, 12].

[4] For example, we tested direct prediction of byte events on our training data. It resulted in accuracy of 90%+, which, however, is not at all useful for actual generation since the smallest mistake leads to fatal syntax corruption.

[5] The actual code implementation makes use of the python package *prettymidi* [13]

[6] A semitone is the smallest possible distance between two different tones in western tonal music.

3 Dataset of adaptive video game music

The 5-tuple encoding above can be applied to all music of information depth equal to simple MIDI. For the scope of our study, we decided to learn from data that consist of the adaptive game music that was originally composed and programmed by Michael Land and Peter McConnell for the all-time adventure game classic *Monkey Island 2* [1]. Despite the game's release in 1991, the significant effort invested in creating the adaptive compositions, including the manual crafting of musical transitions between numerous points in the music variations, remains extraordinary. It continues to serve as a role model in the eyes of experts and critics until today [2, 3].[7]

We selected a specific piece of music from the game: In the town known as Woodtick, which the main character visits at the beginning of the game, the music changes as the character enters different locations. But the music is not just replaced or blended over. Instead, variations of the town music smoothly appear in various manually-prepared dynamic transitions. We have developed a random walking simulator for the game, which is *openly available*[8]. It automatically walks the main character through Woodtick and records the stream of MIDI events using the *ScummVM* emulator [14]. We also track the location change events that trigger the musical transitions in the MIDI stream.

Since a note from a music sequence in close scope depends linearly on its previous notes, the process of creating the sequence can be modeled as an autoregressive task. However, in a wider scope, changing the location leads to distinct musical continuations. Therefore, the event of changing the location must be considered as a conditional input c_t for the autoregressive model

$$note_t = f(note_{t-1}, note_{t-2}, ..., note_{t-p}, c_t), \tag{2}$$

[7] The reason is interesting: The simple MIDI protocol was dropped when waveform sample playback became possible on home computers. This leap in acoustic complexity made the possibility of handcrafting complex adaptive music compositions impossible [2]. The game industry opted for the cinematic feel at the expense of musical immersion. And this trend continues to this day, with each increase in computing power primarily used for better and better game graphics. However, this circumstance is likely to change soon, as the power of home computers is increasing faster and faster due to current improvements in AI technology [15], which could reach a limit of necessary realism (as it was for 4k screen resolution).

[8] `https://github.com/fabianostermann/WoodtickWalkingSimulator`

where p is the context length, i.e. the number of considered previous events. We will use a neural network to map the function f.

The big difference to models usually applied to music generation is that we are not trying to create a general framework for diverse musical outcomes [16, 8]. Instead, we want to approximate a single composition. However, since this composition is adaptive, it differs every time it is played. But the number of possible variations is limited for a finite period of playback. That means, although the model is autoregressive as described, it does not need to generalize for all possible inputs.

4 Problem dimensions

In this section, we present seven different problem dimensions of our task: neural network architecture and type, context length, data input representation, multi-task learning, complexity reduction, multi-objective optimization and rare events. This list of seven is not exhaustive, as there are additional parameters, e.g., batch size, learning rate or the choice of the optimizer itself.[9] However, we chose to focus on discussing these seven dimensions because we found them to be less universal and, in some aspects, unique to the context of music data. We will present some practical experiments on how to optimize parameters for each of the dimensions. Given their interdependency, where changes in one parameter can consistently affect results in other problem dimensions, an all-at-once optimization approach would be ideal. However, since this is beyond the scope of complexity, we decided to propose a chain of optimizations knowing that changing their order may influence final results. Notice that it was also not possible to exhaustively optimize each dimension. We will present exemplary results, discuss how each problem dimension should be addressed, name applicable strategies and name aspects for future investigations.

To optimize the multi-task models, we used an averaging loss function as the overall mean of *cross entropy* loss for pitch and instrument and *mean*

[9] We choose a batch size of 4096, $l_r = 0.001$ and the Adam optimizer [17].

Table 1: Loss and accuracy score comparison of different neural architectures and types

recurrent type	layers	units	dense	loss	↑accuracy
LSTM	2	256	–	**1.081±0.01**	**0.807±0.03**
LSTM	1	256	128	1.105±0.01	0.712±0.01
LSTM	1	256	–	1.107±0.01	0.706±0.02
GRU	2	256	–	1.147±0.00	0.693±0.02
GRU	1	256	128	1.117±0.01	0.657±0.01
GRU	1	256	–	1.178±0.01	0.565±0.02

absolute error loss for the regression tasks of Δtime, duration and loudness. We used the latter instead of *mean squared error* to mitigate averaging issues associated with large magnitude differences. For evaluation, we will mainly provide accuracy score. For Δtime, duration and loudness, we simply defined a deviation of less than 0.05 as sufficiently accurate.[10] Due to random weight initialization, we conducted 5 statistical repetitions on 5 different simulator runs, each consisting of 2 hours of music. Every loss and accuracy score below is reported as mean ± standard deviation of these 5 runs.

4.1 Architectures and neural network types

The first problem when coping with neural networks is always to choose a network type and to determine its structure and size. There are a few general rules to follow [18], but most parameters must be reconsidered with every new problem. Since we have time series data, the natural choice is to use recurrent networks. We chose the popular *long-short term memory* (LSTM) unit [19] and compare with its less complex but more efficient variant *gated reccurent unit* (GRU) [20].

Table 1 shows that an LSTM with 2 layers is performing best for accuracy and loss. A concatenated dense layer helped in case of only one layer. The same applies to GRU, which in comparison performed worse. Note that using only

[10] For Δtime and duration 0.05 equals to 50 ms.

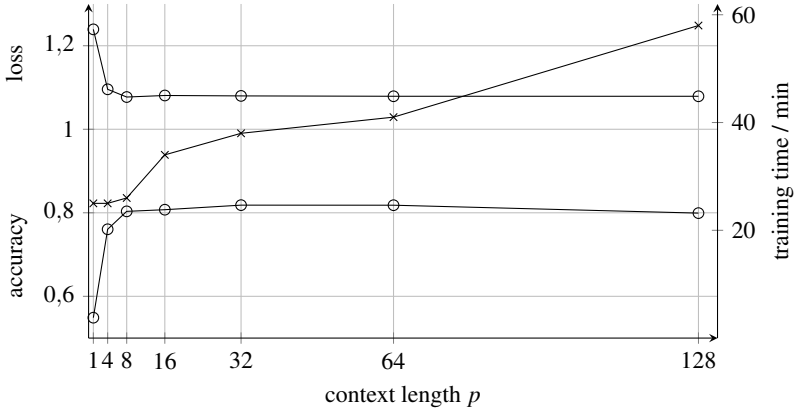

Figure 1: Accuracy score, loss and training time in relation to context length. Circle markers correspond to the left y-axis, cross markers to the right.

the last hidden state of the LSTM gave similar results to GRU, but using hidden state and last cell state improved the results significantly.

We can clearly see that the choice of model type and architecture is, as expected, of critical importance. Another approach could be to use 1D convolutional filters. State-of-the-art autoregressive modeling likely involves the use of transformer models [21, 9], the application of which remains future work.

4.2 Context length

The number of considered previous events is a crucial parameter for autoregressive tasks. It determines, how much context information the model has to predict the next event. For this problem dimension, we used the 2 layered LSTM, which performed best in the previous section. Figure 1 shows, that for already 8 events the accuracy reaches a plateau. After that, it only improves slightly. However, the time needed for calculation increases further since the number of weight parameters inside the LSTM increases non-linearly. With $p = 16$, the calculation needs about one third more time than with $p = 8$.[11] In

[11] 34 min to 26 min on a Nvidia A100 GPU.

order to have more capacity for diverse experiments on the following dimensions, we decided to use $p = 8$ from here on. Please note that a sequence length of $p = 8$ in music holds much information (cf. Eq.1). Just imagine, if someone sings to you 8 notes of a well-known melody, you will probably be able to recognize it.

4.3 Input representation

The internal representation of data can have great influence on the model performance [22]. The categorical information about pitch and instrument before was one-hot encoded with $n_c = 60$ unique classes for pitch and $n_c = 13$ different instruments. With a sequence length of $p = 8$ and the 3 other tasks there were already $8 \cdot (60 + 13 + 3) = 608$ input values. As the model complexity is relative to the input size, we aimed to decrease the number of inputs to the LSTM by adding an input embedding layer [18]. The embedding size was determined as $\lfloor \frac{n_c}{5} \rfloor + 1$. This definition can be considered another hyperparameter to be optimized.

By applying input embedding, we increased the accuracy from 0.807 ± 0.03 to 0.867 ± 0.02 with a decreasing loss from 1.081 ± 0.01 to 0.867 ± 0.02. A further approach to directly use the integer label as input[12], which reduces the input size to 1, performed worse with an accuracy of just 0.716 ± 0.02.

4.4 Multi-task learning vs. ensemble learning

To learn all objectives separately and predict them with an ensemble of models improves evaluation results [23, 18] but comes at a higher computational cost. If not calculated in parallel, it is far slower in inference (and training, cf. Table 2), which might cause severe problems when used for real-time conditioned generation or on low performance hardware.[13]

[12] For pitches, this may be called an *ordinal encoding*, since it provides a natural order.

[13] Regarding the present context, both requirements are met for video games. In addition, computational resources are oftentimes reserved for rendering high-resolution 3D graphics.

Table 2: Loss and accuracy scores for different ensemble and multi-task models. Values belonging to one individual model share a frame.

	accuracy		
Δtime	0.999±0.00	0.955±0.01	0.945±0.01
duration	0.980±0.00	0.876±0.01	0.862±0.02
loudness	0.975±0.00	0.977±0.00	0.880±0.03
pitch	0.907±0.01	0.670±0.02	0.696±0.03
instr.	0.950±0.00	0.953±0.01	0.954±0.00
mean	0.962±0.00	0.886±0.00	0.867±0.02
∑ training time	106 min	43 min	29 min

Table 2 shows that learning single tasks (left column with numbers) is much easier than learning multiple tasks at once (right column). Learning the regression tasks (Δtime, duration, loudness) or the categorical task separately (middle column) only improves scores slightly (except for loudness). However, handling this interdependency is a major topic. E.g., the choice of the next pitch is highly dependent on which instrument is chosen. Before, they were predicted in parallel, but the accuracy may be improved by using an hierarchical learning approach to predict the objectives one after another.

4.5 Complexity reduction

This dimension is to be understood as the output equivalent of the input representation from Section 4.3. As Table 2 shows, predicting pitch is the most difficult objective. That probably is because it has the most classes and the largest dependencies to the previous notes. In addition, only a near 100% accurate prediction is satisfactory from a psychological point of view. When the timing is slightly off or the melody is played by the wrong instrument, the mistake is not perceived as serious as that of choosing the wrong pitch.[14] In this context, leveraging the semantic nature of music data can be used to further reduce complexity. The pitch information can be divided into two bits

[14] An exception is the drums, because when playing back a melody on the MIDI drum channel, it becomes completely unrecognizable. Hence, another approach to complexity reduction could involve segregating the prediction of drum instruments from melody and harmony instruments.

Table 3: Accuracy scores for predicting pitch divided into pitch class and octave.

pitch		Δtime	duration	loudness	instrument	mean
0.696±0.03		0.945±0.01	0.862±0.02	0.880±0.03	0.954±0.00	0.867±0.02
0.802±0.01	0.908±0.01	0.900±0.02	0.795±0.04	0.817±0.04	0.953±0.00	0.862±0.02
pitch class	octave					

of information: octave and pitch class. This introduces an additional task to the multi-task model but has the advantage that predicting one of 12 pitch classes can be learned with a higher accuracy. In addition, playing in the wrong octaves (of 5) is not perceived as nearly as disharmonic as any smaller semitone error.[15] Table 3 shows a boost in overall pitch accuracy by splitting up the task. But the multi-task difficulty increases and thus all the single regression tasks drop in accuracy. However, the average accuracy remains unchanged.

Another approach of complexity reduction is to transform each linear regression task into a classification task by binning or clustering. This might be advantageous since continuous regression is at times more difficult, especially when combined with a classification task in a multi-task learning problem (cf. Section 4.4).

A more complex approach may be to encode segments of the music with a (variational) autoencoder that is then controlled by another agent that is rewarded to recreate the composition in a reinforcement learning setting. Moreover, this approach would be able to come up with some novel variations of the music.

4.6 Multi-objective optimization

Since the present task is inherently multi-objective, we can also consider to improve the loss function itself. Up to here, the loss value was calculated as the equally-weighted mean of all single loss values of all the objectives. Since pitch was identified to be the hardest but most important objective, we try to

[15] The auditory phenomenon of *tonal fusion* [24] explains that two notes played in the interval of one octave are most likely to be perceived as a single tone among all possible intervals.

Table 4: Accuracy scores of different prioritization of pitch under weight w_p.

w_p	pitch	Δtime	duration	loudness	instrument	mean
0.0	0.017±0.00	0.859±0.03	0.735±0.04	0.778±0.04	0.950±0.00	0.668±0.02
0.2 (mean)	**0.681±0.02**	**0.949±0.01**	**0.869±0.01**	**0.885±0.01**	**0.955±0.00**	**0.868±0.01**
0.5	0.613±0.02	0.757±0.04	0.614±0.02	0.670±0.03	0.950±0.00	0.721±0.02
0.8	0.616±0.03	0.579±0.01	0.451±0.03	0.436±0.04	0.948±0.00	0.606±0.02
1.0	0.629±0.02	0.053±0.01	0.051±0.01	0.038±0.02	0.951±0.01	0.344±0.00
random	0.605±0.02	0.802±0.02	0.679±0.05	0.603±0.05	**0.955±0.00**	0.729±0.01

determine if it is possible to boost its accuracy by prioritizing it over the other objectives. The following formular will be used for prioritizing objective p with weight w_p over all the other n objectives:

$$loss = w_p \cdot loss_p + \sum_{i=1}^{n} (\frac{1 - w_p}{n}) \cdot loss_i \tag{3}$$

For $w_p = 0.2$, this formula corresponds to the average weighting used before.

Table 4 shows that the applied weighting procedure is not able to boost the accuracy of the pitch prediction. However, a w_p of 0.0 prevents any learning success for pitch, but surprisingly the other objectives do not benefit from it. When increasing w_p to 1.0, the other objective drop to an accuracy of nearly 0 as expected (except instrument). The pitch objective, however, does not benefit from this either, since its accuracy decreases in comparison to the equal weighting ($w_p = 0.2$). The baseline comparison to a random weighting, that randomly changes w_p on each call, also shows no improvement.

In total, weighting for prioritization was not successful, since equally weighting did indead perform better. This result may be explained by destabilizing the gradient descent. If so, parameters like batch size and learning rate must also be reconsidered here. In any case, this topic is complex and worth investigating in future work, e.g. by applying Chebyshev loss or other strategies from the domain of multi-objective optimization.

4.7 Rare events

Music compositions typically exhibit numerous redundancies, such as repetitions, reprises, and motivic variations. As a result, music data often is imbalanced. In the simulation, we observe that the main character spends roughly half of the time outside in the town scene (due to random walking), leading to a significantly higher presence of the corresponding music. Furthermore, location changes account for only a maximum of 5% of the data, resulting in a severe underrepresentation of individual musical transitions, which are about 5 to 10 per possible location transition. This may explain why all the training sessions could never reach 100% accuracy and why, e.g., instrument performs well even if zero weighted (cf. Table 4 where $w_p = 1.0$) To meet this circumstance, heavy dataset balancing [25] is needed. One approach without removing samples from the dataset is to use the loss values as an information of success and to prioritize samples of higher loss during training, either by loss weighting or by dynamic adjustment of selection probabilities. This topic also remains work for future investigations.

5 Conclusions

We have seen that music as a problem domain provides a rich ground for diverse research questions. Datasets with adaptive music from the video game *Monkey Island 2* can be easily generated using our openly available generator. We have also provided experimental setups and analyses covering the problem dimensions of neural network architecture and type, context length, data input representation, multi-task learning, complexity reduction, multi-objective optimization and rare events. These aspects are critically important and, in some aspects, unique to the task of adaptive music generation. We have presented numerous ideas for future work. Implementing more sophisticated multi-objective loss strategies and to cope with rare events by adaptive dataset balancing are both worth further investigations.

The concept of autoregressive music generation also holds the potential to be applied in the creation of original compositions for video games featuring

adaptive music. Surprisingly, this powerful component [26] is still underutilized in the industry. The notion of using variational autoencoders controlled by a reinforcement learning agent is intriguing. A solution to this challenge surely is of interest not only to the video game industry but also to broader contexts, as the future trend is clearly heading towards "non-linear media" [27].

References

[1] R. Gilbert, T. Schafer, and D. Grossman. *Monkey Island 2 – LeChuck's Revenge*. LucasArts/Lucasfilm Games, 1991. `https://archive.org/details/msdos_Monkey_Island_2_-_LeChucks_Revenge_1991`. Accessed 24 Sep 2023.

[2] M. Sweet. *Writing Interactive Music for Video Games: A Composer's Guide (Game Design)*. Addison-Wesley Professional, Boston, USA, 2014.

[3] T. Summers. *Unterstanding Video Game Music*. Cambridge University Press, Cambridge, UK, 2016.

[4] M. Lepper. *de Linguis Musicam Notare – Beiträge zur Bestimmung von Semantik und Stilistik moderner Musiknotation durch mathematische Remodellierung* (german). In R. Großmann and H. Kinzler, editors, *Beiträge zur Medienästhetik der Musik*. epOs-Verlag, Osnabrück, Germany, 2021.

[5] The MIDI Association (TMA). *Musical Instrument Digital Interface – Official Specifications*. 1983. `https://midi.org/specifications`. Accessed 24 Sep 2023.

[6] C. Raffel. *Learning-Based Methods for Comparing Sequences, with Applications to Audio-to-MIDI Alignment and Matching*. PhD thesis, Columbia University, New York, USA, 2016.

[7] S. Mehri, K. Kumar, I. Gulrajani, R. Kumar, S. Jain, J. Sotelo, A. Courville, and Y. Bengio. SampleRNN: An unconditional end-to-end neural audio generation model. In *International Conference on Learning Representations*, 2017.

[8] A. Agostinelli, T. I. Denk, Z. Borsos, J. Engel, M. Verzetti, A. Caillon, Q. Huang, A. Jansen, A. Roberts, M. Tagliasacchi, M. Sharifi, N. Zeghidour, and C. Frank. MusicLM: Generating music from text. *arXiv preprint*, 2023.

[9] Y.-S. Huang and Y.-H. Yang. Pop music transformer: Beat-based modeling and generation of expressive pop piano compositions. MM '20, page 1180–1188, New York, USA, 2020. Association for Computing Machinery.

[10] D. P. Pazel. *Music Representation and Transformation in Software - Structure and Algorithms in Python*. Springer Cham, 2022.

[11] R. Kelz, S. Böck, and C. Widnaer. Multitask learning for polyphonic piano transcription, a case study. In *2019 International Workshop on Multilayer Music Representation and Processing (MMRP)*, pages 85–91, 2019.

[12] J. P. Gardner, I. Simon, E. Manilow, C. Hawthorne, and J. Engel. MT3: Multi-task multitrack music transcription. In *International Conference on Learning Representations*, 2022.

[13] C. Raffel and D. P. W. Ellis. Intuitive analysis, creation and manipulation of midi data with prettymidi. In *15th International Conference on Music Information Retrieval Late Breaking and Demo Papers*, 2014.

[14] P. Kołodziejski, E. Sandulenko, E. J. T. Sømåen, and L. S. Mari. *ScummVM: Script creation utility for maniac mansion – Virtual Machine*. ScummVM Team, 2020–2023. `https://docs.scummvm.org/en/v2.7.0`. Accessed 24 Sep 2023.

[15] H. C. Lin and A. Burnes. Nvidia DLSS 3: AI-powered performance multiplier boosts frame rates by up to 4x. *Nvidia GeForce News*, 2022. `https://www.nvidia.com/en-us/geforce/news/dlss3-ai-powered-neural-graphics-innovations/`. Accessed 24 Sep 2023.

[16] J.-P. Briot, G. Hadjeres, and F.-D. Pachet. *Deep Learning Techniques for Music Generation*. Springer, Berlin, Heidelberg, Germany, 2020.

[17] D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2015.

[18] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, Cambridge, Massachusetts, USA, 2016.

[19] S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997.

[20] K. Cho, B. van Merrienboer, Çaglar Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Conference on Empirical Methods in Natural Language Processing*, 2014.

[21] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, page 6000–6010, 2017.

[22] O. Neumann, N. Ludwig, M. Turowski, B. Heidrich, V. Hagenmeyer, and R. Mikut. Smart data representations: Impact on the accuracy of deep neural networks. In *Proceedings 31st Workshop Computational Intelligence*, pages 113–130, Berlin, Germany, 2021. KIT Scientific Publishing.

[23] P. Sollich and A. Krogh. Learning with ensembles: How overfitting can be useful. In *Proceedings of the 8th International Conference on Neural Information Processing Systems*, volume 8, page 190–196, 1995.

[24] M. Ebeling. Musical structures and their perception. In C. Weihs, D. Jannach, I. Vatolkin, and G. Rudolph, editors, *Music Data Analysis: Foundations and Applications*, Chapman & Hall/CRC Computer Science & Data Analysis. Taylor & Francis Group, New York, USA, 2019.

[25] V. Werner de Vargas, J. A. Schneider Aranda, R. dos Santos Costa, P. R. da Silva Pereira, and J. L. Victória Barbosa. Imbalanced data preprocessing techniques for machine learning: a systematic mapping study. *Knowledge and Information Systems*, 65(1):31–57, 2023.

[26] R. Stevens, D. Raybould, and D. McDermott. Extreme ninjas use windows, not doors: Addressing video game fidelity through ludo-narrative music in the stealth genre. In *Proceedings 56th International AES Conference: Audio for Games*, 2015.

[27] K. Tatar and P. Pasquier. Musical agents: A typology and state of the art towards musical metacreation. *Journal of New Music Research*, 48(1):56–105, 2019.