Let the Music Flow Where the Modal Branches Lead

Lorenzo Balboni[1], Federico Manzella[1,*] and Guido Sciavicco[1]

[1]*University of Ferrara, Italy*

Abstract

This paper presents a novel approach to symbolic music generation using Modal Decision Trees (MDTs), a class of interpretable, rule-based models that integrate modal logic into the decision-tree algorithm. We construct a dataset from multitrack MIDI files, representing each track as a binary pianoroll matrix, and frame the generation task as a binary classification problem: predicting whether a pitch should be played at each time step based on a local window of context. The performed experiments demonstrate that MDTs, despite not leveraging explicit pitch identities, achieve strong performance, with an F1 score of 0.926 and balanced accuracy of 0.943. The generated music closely follows the input melodies, with some modifications that reflect the model's learned patterns. While MDTs do not yet match the performance of deep learning models for temporal sequence modeling, our results highlight their potential as interpretable tools for symbolic music generation. We discuss future directions, including modal decision forests, multi-track analysis, and controlled randomness to enhance generative diversity.

Keywords

MIDI, Symbolic Music, Algorithmic Composing, Music Generation, Symbolic Machine Learning, Interpretable

1. Introduction

AI-generated music relies on two main forms of representation: audio and symbolic. Audio representation directly reproduces sound, rich in timbral and performative details. Generation in this format focuses on synthesizing the sound signal itself. Symbolic representation, by contrast, abstracts music into notes, rhythms, and structures, similar to traditional notation, enabling more efficient and interpretable processing.

This distinction defines two primary approaches [1]: music generation (audio-based) [2] and algorithmic composition (symbolic-based) [3]. The former emphasizes sonic rendering, while the latter focuses on musical structure. Each has distinct computational requirements and objectives.

Historically, algorithmic composition dates back centuries, from Pythagoras to Mozart's Musikalisches Würfelspiel [4], and later to the Illiac Suite (1957). Audio generation, on the other hand, is a more recent development, closely tied to advances in deep learning. Successful deep learning approaches include RNNs, LSTMs, GANs, and VAEs [5, 6, 7], which are capable of capturing complex patterns in data, though they often lack interpretability.

Choosing between audio and symbolic formats involves a fundamental trade-off: expressiveness and sonic fidelity versus control, efficiency, and structural clarity. This balance continues to guide current research into hybrid methods.

Algorithmic composition has its roots in symbolic AI [8], well before the emergence of deep learning. Grounded in explicit, transparent rules, this tradition formalizes musical knowledge using logic and structured representations.

Symbolic music generation has historically employed strategies such as rule-based systems, controlled randomness, and logical structures. Over time, various paradigms have emerged, including stochastic models like Markov chains [9, 10], musical grammars [11, 12, 13], expert systems (e.g., Ebcioğlu's

7th International Workshop on Artificial Intelligence and Formal Verification, Logic, Automata, and Synthesis (OVERLAY 2025), October 26, 2025, Bologna, Italy

*Corresponding author.

lorenzo02.balboni@edu.unife.it (L. Balboni); federic.manzella@edu.unife.it (F. Manzella); guido.sciavicco@unife.it (G. Sciavicco)

https://ferdiu.it (F. Manzella); https://sites.google.com/unife.it/guido/home (G. Sciavicco)

0009-0005-1917-7660 (L. Balboni); 0000-0002-4944-2163 (F. Manzella); 0000-0002-9221-879X (G. Sciavicco)

CHORAL [14]), and case-based reasoning [15, 16], eventually leading to hybrid solutions that integrate multiple techniques [17]. A notable example is David Cope's EMI [18], which can emulate classical styles using symbolic rules. Unlike neural models, these systems are fully interpretable, offering direct control and allowing insight into the creative processes they employ.

2. Symbolic music generation using Modal Decision Trees

Modal logic generalizes propositional logic by introducing modal operators for necessity ($\Box$) and possibility ($\Diamond$). Its semantics rely on Kripke models (W, R, V), where worlds W relate via accessibility R, and valuation V assigns truth to propositions in each world. Modal logic enables reasoning about complex relational structures, underpinning formalisms for temporal, spatial, epistemic, and deontic reasoning.

Modal Decision Trees (MDTs) [19, 20] integrate modal logic into decision-tree frameworks, allowing symbolic learning directly from structured data modeled as finite Kripke structures. Unlike classical decision trees, MDTs handle relational, temporal, and spatial information explicitly within modal logic formulas at decision nodes. MDTs thus maintain interpretability and provide logical explanations, making them suitable for structured domains requiring transparent reasoning and robust predictions.

In this work we choose to plug into the learning algorithm the *interval temporal modal logic*, specifically the logic of *Halpern and Shoham* (HS) [21]. This approach allows one to represent the musical data as sequences of intervals rather than points in time. Each interval can relate to others through one of the Allen's interval relations [22] being *adjacent to*, *later than*, *begins*, *ends*, *during* and *overlaps*, $\{A, L, B, E, D, O\}$, all of their inverse relations $\{\overline{A}, \overline{L}, \overline{B}, \overline{E}, \overline{D}, \overline{O}\}$, and the *same interval* (the identity I). These relations are treated as modalities, allowing the model to reason about temporal patterns via existential or universal operators over these relations. This logic is used to make decisions about sequences of notes by evaluating dynamic features over intervals, rather than over single points. Each split in the tree involves checking whether there exists a related interval that satisfies a given property, such as whether a particular melodic or harmonic pattern occurs *during* or *overlaps with* the current interval. This interval-based reasoning is crucial in music, where structure often emerges from temporal relationships between note groups, not just from isolated events. By learning decision rules over these modal and temporal features, MDTs can capture recurring musical patterns and generate structurally coherent sequences.

To leverage the expressive capacity of this approach, we trained the model to generate symbolic music using a dataset of time-pitch matrices encoding note activations. The model predicts whether a pitch should be played or not at each time step, based on a window of previously played pitches. Generation is achieved by sliding this window both vertically and horizontally across the input representation of musical notes, allowing the MDT to determine pitch activation. This window can be seen as the *context* that the algorithm can use to build intervals and search relations in. The model operates without explicit knowledge of pitch labels. That is, it does not know which specific pitch (e.g. C4) it is evaluating, nor the identities of pitches in the input window. Instead, it relies purely on the structural relationships within the local pitch-time context to make predictions by dynamically evaluating features over intervals: 1. whether a note is never present in an interval (evaluating the minimum value), 2. whether a note is present at least once in an interval (evaluating the maximum value), and 3. how long the note was played in the given interval (evaluating the average value).

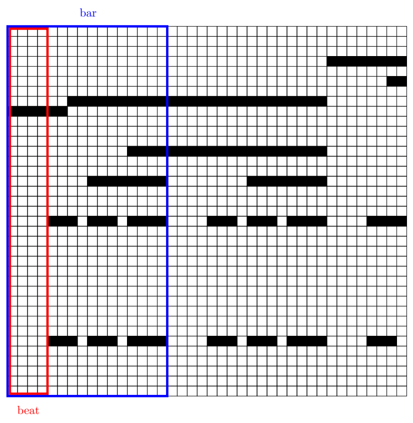
Before delving into the dataset, let's introduce the concept of a *pianoroll matrix*, which can be easily derived from symbolic music formats like MIDI. It is defined as a matrix $P \in \mathbb{N}^{n \times m}$, where:

- n is the number of supported pitches (128 in MIDI notation, from C-1 to G9).
- m is the number of time steps.
- $p_{i,j}$ is a natural number indicating the velocity of the pitch i at time j.

Due to the nature of MIDI files, velocity values range from 0 to 127, where 0 indicates that a note is not played at all, and 127 represents the maximum volume.

$$\begin{array}{cc} & \xrightarrow{\hspace{6em} time \hspace{6em}} \\ P = & \begin{bmatrix} p_{1,1} & \cdots & p_{1,m} \\ \vdots & \ddots & \vdots \\ p_{128,1} & \cdots & p_{128,m} \end{bmatrix} \Big\uparrow \, pitches \end{array}$$

(a)



(b)

Figure 1: (a) A matricial representation of a pianoroll matrix P, m time steps long and (b) a portion of binary pianoroll matrix P derived from a MIDI file. The red rectangle represents a segment of the matrix representing a single beat, which is divided into 4 timesteps since the resolution is 4. The blue rectangle represents a bar, which is made up of 4 beats.

Since predicting the exact velocity of a note is a more complex task, we simplify the problem into binary classification by predicting whether a note is played or not, rather than its velocity. This is equivalent to assigning class 0 to velocity values of 0, and class 1 to all non-zero values. So, the pianoroll matrices we are dealing with are binary matrices, where each element $p_{i,j}$ is either 0 or 1.

In Figure 1b, we present an example of a segment from a binary pianoroll matrix P derived from a MIDI file. Each beat has a *resolution* of 4 timesteps, meaning that each beat is represented by 4 consecutive columns in the matrix. As mentioned earlier, the matrix is structured such that each row corresponds to a pitch, and each column corresponds to a time step.

Given the definition of the pianoroll matrix, we formally define the dataset as $\mathcal{D} = \{(\mathbf{X}^{(i)}, y^{(i)})\}_{i=1}^{N}$, where N is the number of instances and each instance $\mathbf{X}^{(i)} \in \{0,1\}^{vision \times lookback}$ is a binary matrix representing a window in a pianoroll matrix, where:

- $vision$ is the number of pitches of an instance calculated as seen in Equation 1.
- $lookback$ is the number of timesteps of an instance.

$$vision = 2 \cdot octave\ vision + 1 \tag{1}$$

To control the size of the instances in the dataset, and thus determine how much context in terms of pitches and time steps should be provided to the model for predicting whether a pitch is present, we define two parameters:

- *octave vision* determines how many octaves above and below a central pitch to include in the instance.
  - If the octave vision is set to 0, each instance will have 1 feature, which corresponds to a single pitch, the one being evaluated by the model (Figure 2a).
  - If the octave vision is set to 1, each instance will have 25 features, which corresponds to the pitch being evaluated and the 12 pitches in the octave above and below it (Figure 2b).
- *lookback beats* determines how many beats to look back.
  - By default, each beat has a resolution of 4 timesteps, so a *lookback beats* of 1 corresponds to 4 timesteps, a *lookback beats* of 2 corresponds to 8 timesteps, and so on.
  - This parameters indirectly controls the number of instances (N), indeed given a pianoroll matrix $P \in \mathbb{N}^{n \times m}$, the number of instance generate from it is $\mid N_P \mid = (n - lookback) \times m$.

Each instance $\mathbf{X}^{(i)}$ represents a window of size $vision \times lookback$ in the pianoroll matrix P, outlined as the blue rectangle (Figure 2b), which , and the label $y^{(i)}$ is defined as the green rectangle, which represents the pitch being evaluated by the model at the current time step.

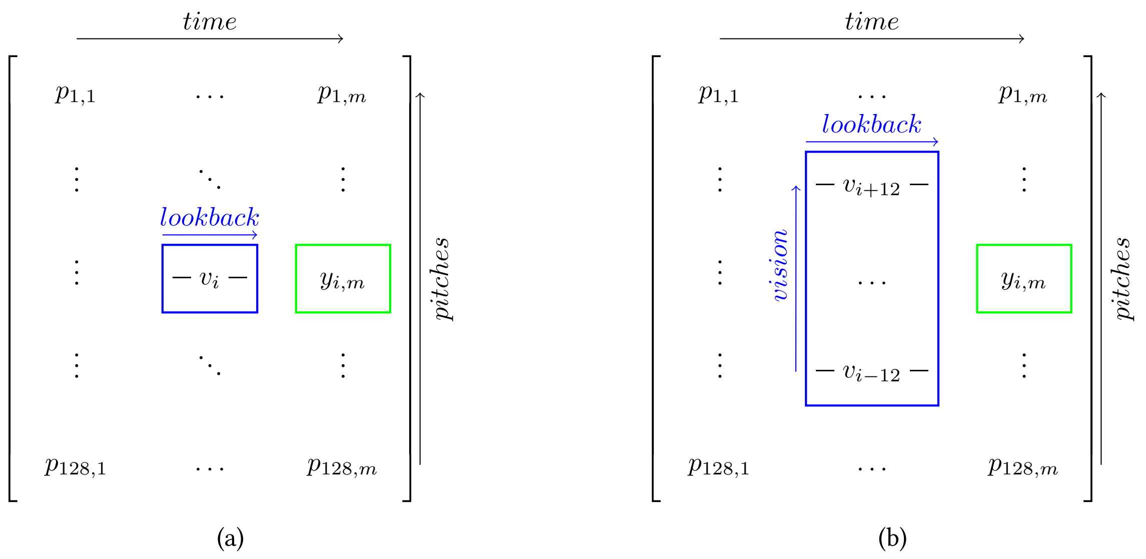


Figure 2: Extraction of an instance from a pianoroll matrix. The green rectangle represents the label $y^{(i)}$, while the blue rectangle represents a pianoroll window $\mathbf{X}^{(i)}$ with $vision$ of 0 (a) and with $vision$ of 1 (b).

By sliding a window of size $vision \times lookback$ across P and extracting the corresponding label, we can generate the dataset.

Furthermore, because instances with label 0 significantly outnumber those with label 1, we reduce the dataset size by cropping each pianoroll matrix P to its played pitch range. This range is defined as the interval between the lowest and highest pitches that are played at least once. All pitches outside this range are removed. To further reduce the imbalance of the dataset, which is still dominated by instances with label 0, we transposed all pianorolls to the key of C major. Consequently, we removed all pitches corresponding to sharp and flat notes, as they do not belong to the C major scale.

This simplification makes the task of music composition more manageable without imposing significant limitations. Previous studies have employed datasets where all songs are transposed to the same scale, such as C major, to standardize the data and facilitate the learning process [23]. Since the model predicts whether a specific pitch is played based solely on a local window of the pianoroll, it does not require knowledge of the absolute pitch identities within that window—only the relative patterns and activity matter, making the exact pitch range irrelevant for learning. Moreover, a composition written in one scale can easily be transposed to another by shifting all notes by a fixed interval. This is a straightforward operation in MIDI, which avoids the complications that typically arise when transposing audio. This assumption holds true for compositions in the major and natural minor scales.

Finally, the dataset is derived from a collection of multitrack MIDI files representing modern pop compositions in 4/4 time, where each track corresponds to a single instrument or, more generally, a distinct musical layer (e.g., bass, arpeggio, lead, melody, or rhythm). The Python library `pypianoroll` [24] was extensively used, as it enables the representation of MIDI files as multitrack objects, in which each track of the original MIDI file is represented by its own pianoroll matrix.

Another version of the dataset has been created to let Decision Trees (DTs) learn from it. In this case, the instance is defined as a vector of size $vision \times lookback$, where each element corresponds to a pitch at a specific time step.

Several experiments have been conducted by varying the dataset configuration parameters as well as the model type, comparing standard *Decision Trees* (DT) and *Modal Decision Trees* (MDT). Among these, the best results were obtained using a dataset of 3624 instances, each with a size of 25×64, corresponding to an *octave vision* of 1 and a *lookback beats* of 16. For this specific dataset configuration, both models were trained and evaluated under identical conditions to ensure a fair comparison.

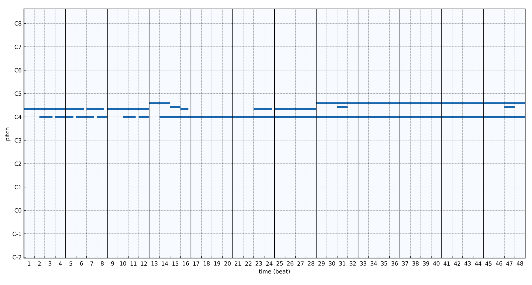
Overall, the Modal Decision Tree showed improved performance over the standard Decision Tree across major evaluation metrics. It achieved a higher F1 score (0.926), indicating a better balance between precision and recall, as well as greater overall accuracy (93.8%) and balanced accuracy (94.3%), reflecting robustness to class imbalance. The Kappa coefficient for the MDT was also notably higher (0.852), suggesting stronger agreement between predictions and ground truth beyond chance.

Metric	Modal Decision Tree	Decision Tree
FScore	0.926	0.904
Accuracy	0.938	0.920
BalancedAccuracy	0.943	0.920
Kappa	0.852	0.808
MisclassificationRate	0.062	0.080
FalseDiscoveryRate	0.088	0.108
FalseNegativeRate	0.057	0.079
NegativePredictiveValue	0.912	0.891
TrueNegativeRate	0.943	0.920

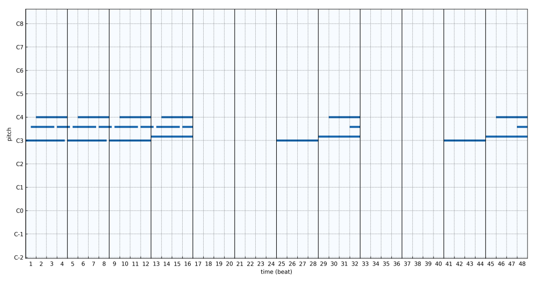
Table 1: Comparison of evaluation metrics between the best performing Modal Decision Tree and standard Decision Tree trained on the same dataset configuration.

Generations were performed on test MIDI files using both models. While the Modal Decision Tree often produced sequences that loosely echoed the input melody, with simplified or slightly altered patterns, the standard Decision Tree typically generated outputs consisting of sparse, disconnected notes, sometimes repeating the same pitch excessively or lacking any coherent structure. However, it is important to note that the difference in generative quality, though present, is not dramatic: most generations from both models lack true musicality or structure. That said, the Modal Decision Tree occasionally succeeds in producing musically plausible phrases. Informal evaluation suggests that, out of ten generations, the MDT typically yields two somewhat coherent results, whereas the standard DT consistently fails to produce anything musically meaningful. In essence, while both models struggle to generate convincing music, the Modal Decision Tree demonstrates a slight but meaningful edge in its capacity to occasionally generate outputs that resemble simplified musical phrases.

These findings are summarized in Table 1 and visually illustrated in Figure 3.



(a) Generated music example 1



(b) Generated music example 2

Figure 3: Examples of generated music, where the input to the model ends at beat 16 and generation starts.

3. Conclusions

The symbolic music generation obtained from the experiments presented in this work produced results that, while not outstanding, are promising and open to improvement. This approach, based on Modal Decision Trees, has not been previously applied to this task. Its performance is naturally lower compared to more established models such as RNN and LSTM networks, which are better suited for handling temporal sequences. However, this work introduces a novel interpretable and rule-based method that complements existing approaches. Future work will explore improvements such as constructing modal decision forests, analyzing multiple tracks simultaneously, and introducing controlled randomness to allow the same input to produce different outputs at each inference.

Declaration on Generative AI

The authors declare that no Generative Artificial Intelligence (GenAI) tools or services were used in the preparation, writing, editing, or revision of this paper. All content was created solely by the authors.

References

[1] L. Wang, Z. Zhao, H. Liu, J. Pang, Y. Qin, Q. Wu, A review of intelligent music generation systems, 2023. URL: https://arxiv.org/abs/2211.09124. arXiv:2211.09124.

[2] Y. Chen, L. Huang, T. Gou, Applications and advances of artificial intelligence in music generation:a review, 2024. URL: https://arxiv.org/abs/2409.03715. arXiv:2409.03715.

[3] S. Tian, C. Zhang, W. Yuan, W. Tan, W. Zhu, Xmusic: Towards a generalized and controllable symbolic music generation framework, 2025. URL: https://arxiv.org/abs/2501.08809. arXiv:2501.08809.

[4] D. Williams, A. Kirke, E. Miranda, E. Roesch, I. Daly, S. Nasuto, Investigating affect in algorithmic composition systems, Psychology of Music 43 (2014). doi:10.1177/0305735614543282.

[5] H.-W. Dong, W.-Y. Hsiao, L.-C. Yang, Y.-H. Yang, Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment, 2017. URL: https://arxiv.org/abs/1709.06298. arXiv:1709.06298.

[6] L.-C. Yang, S.-Y. Chou, Y.-H. Yang, Midinet: A convolutional generative adversarial network for symbolic-domain music generation, 2017. URL: https://arxiv.org/abs/1703.10847. arXiv:1703.10847.

[7] A. Roberts, J. Engel, C. Raffel, C. Hawthorne, D. Eck, A hierarchical latent vector model for learning long-term structure in music, 2019. URL: https://arxiv.org/abs/1803.05428. arXiv:1803.05428.

[8] J. Fernandez, F. Vico, Ai methods in algorithmic composition: A comprehensive survey, Journal of Artificial Intelligence Research 48 (2013) 513–582. URL: http://dx.doi.org/10.1613/jair.3908. doi:10.1613/jair.3908.

[9] Volume information, Computer Music Journal 10 (1986) 17–39. URL: http://www.jstor.org/stable/3680287.

[10] D. Ponsford, G. Wiggins, C. Mellish, Statistical learning of harmonic movement, Journal of New Music Research 28 (1999) 150–177.

[11] J. A. Moorer, Music and computer composition, Commun. ACM 15 (1972) 104–113. URL: https://doi.org/10.1145/361254.361265. doi:10.1145/361254.361265.

[12] D. Lidov, J. Gabura, A melody writing algorithm using a formal language model, Computer Studies Humanities Verbal Behaviour 4 (1973).

[13] G. M. Rader, A method for composing simple traditional music by computer, Commun. ACM 17 (1974) 631–638. URL: https://doi.org/10.1145/361179.361200. doi:10.1145/361179.361200.

[14] K. Ebcioğlu, An expert system for harmonizing four-part chorales, Computer Music Journal 12 (1988) 43–51. URL: http://www.jstor.org/stable/3680335.

[15] F. Pereira, C. Grilo, L. Macedo, A. Cardoso, Composing Music using Case-Based Reasoning, 1997.

[16] P. Ribeiro, F. Pereira, M. Pérez Ferra, A. Cardoso, Case-based melody generation with muzacazuza (2002).

[17] G. Ramalho, J.-G. Ganascia, Simulating creativity in jazz performance (1999).

[18] D. Cope, Experiments in musical intelligence (emi): Non-linear linguistic-based composition, Interface 18 (1989) 117–139. doi:10.1080/09298218908570541.

[19] D. Della Monica, G. Pagliarini, G. Sciavicco, I. E. Stan, Decision trees with a modal flavor, in:

A. Dovier, A. Montanari, A. Orlandini (Eds.), AIxIA 2022 – Advances in Artificial Intelligence, Springer International Publishing, Cham, 2023, pp. 47–59.

[20] F. Manzella, G. Pagliarini, G. Sciavicco, I. E. Stan, Efficient modal decision trees, in: R. Basili, D. Lembo, C. Limongelli, A. Orlandini (Eds.), AIxIA 2023 – Advances in Artificial Intelligence, Springer Nature Switzerland, Cham, 2023, pp. 381–395.

[21] J. Halpern, Y. Shoham, A propositional modal logic of time intervals, Journal of the ACM 38 (1991) 935–962.

[22] J. F. Allen, Maintaining knowledge about temporal intervals, Communications of the ACM 26 (1983) 832–843.

[23] H.-W. Dong, W.-Y. Hsiao, L.-C. Yang, Y.-H. Yang, Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment, in: Proceedings of the 32nd AAAI Conference on Artificial Intelligence, 2018. URL: https://arxiv.org/abs/1709.06298.

[24] Y.-S. Huang, Y.-H. Yang, pypianoroll: Open source python library for handling multitrack pianorolls, Late-Breaking/Demo ISMIR 2018 (2018). URL: https://github.com/salu133445/pypianoroll.