frontiers | Frontiers in Computer Science

TYPE Original Research
PUBLISHED 30 April 2025
DOI 10.3389/fcomp.2025.1543074

OPEN ACCESS

EDITED BY
Çagri Erdem,
University of Oslo, Norway

REVIEWED BY
Evellin Cardoso,
Universidade Federal de Goiás, Brazil
Vincenzo Madaghiele,
University of Oslo, Norway

*CORRESPONDENCE
Juan S. Vassallo
✉ juan.vassallo@uib.no

RECEIVED 10 December 2024
ACCEPTED 09 April 2025
PUBLISHED 30 April 2025

CITATION
Vassallo JS, Sandred Ö and Vincenot J (2025) NeuralConstraints: integrating a neural generative model with constraint-based composition.
Front. Comput. Sci. 7:1543074.
doi: 10.3389/fcomp.2025.1543074

NeuralConstraints: integrating a neural generative model with constraint-based composition

Juan S. Vassallo[1]*, Örjan Sandred[2] and Julien Vincenot[3]

[1]Faculty for Art, Music and Design, The Grieg Academy, University of Bergen, Bergen, Norway, [2]Marcel A. Desautels Faculty of Music, University of Manitoba, Manitoba, MB, Canada, [3]Department of Music, Harvard University, Cambridge, MA, United States

We present 'NeuralConstraints,' a suite of computer-assisted composition tools that integrates a feedforward neural network as a rule within a constraint-based composition framework. 'NeuralConstraints' combines the predictive generative abilities of neural networks trained on symbolic musical data with an advanced backtracking constraint algorithm. It provides a user-friendly interface for exploring symbolic neural generation, while offering a higher level of creative control compared to conventional neural generative processes, leveraged by the constraint solver. This article outlines the technical implementation of the core functionalities of 'NeuralConstraints' and illustrates their application through specific tests and examples of use.

KEYWORDS

artificial intelligence, machine learning, human-computer interaction, computer-assisted composition, neural networks, deep learning, constraints solver algorithms

Introduction

Music is a field where intellect and emotion converse. Music theorists emphasize the importance of structure as a foundation for creating coherence and meaning within musical works. At the same time, being immersed in a musical culture allows us to *intuitively*[1] appreciate music as an expression beyond our intellect. This duality between the mind's need for structure and its response to emotional depth lies at the heart of musical communication.

Composers use many varied strategies when creating music: some look at structure as a starting point for a new composition, while others lean toward an intuitive approach (Wiggins, 2012; Pohjannoro, 2016; Pohjannoro, 2021). Computer-assisted composition (CAC) has traditionally served to leverage the structural foundations of music composition. These methods potentially assist in the creative process by reducing the cognitive load associated with structural details, freeing composers to concentrate on artistic intent and expression (Xenakis, 1992; Koenig, 1970). The use of symbolic AI and rule-based methods in CAC dates back to some of the earliest experiments in computer-generated music (Brooks et al., 1957; Hiller and Isaacson, 1958). Other approaches include corpus-based methods, such as Markov Chains (Ames, 1989; Pachet and Roy, 2011; Ramanto and Maulidevi, 2017; Vassallo, 2024) and

1 *Intuition*, often described as the ability to understand something without the need for conscious reasoning, is a concept usually echoed in artificial intelligence, with researchers illustrating how these systems can mimic human-like pattern recognition and instinctive decision-making processes. See for example Frantz (2003), Simon and Mellon (1995).

Neural Networks (Todd, 1989; Voisin and Meier, 2009), as well as grammars, evolutionary methods, and more[2].

With the rise of connectionist techniques for symbolic music generation, it is not necessary to formalize rules or algorithms to generate music. Instead, a computational model such as a Neural Network (NN) can be trained with a corpus of example scores to learn. After this training phase, the NN can generate new scores with similar musical properties to the examples it was trained on. This approach closely resembles the intuitive methods composers often use, grounded in experiential processes shaped by the musical cultures in which they are immersed (Pearce and Wiggins, 2007; Wiggins, 2006; Boden, 2004). The strength of this method is also its weakness; since it is based on statistical predictions, we should not expect any output that substantially differs musically from the training set. For a composer who wants to use a generative tool to explore new musical ideas, a NN might not be an ideal method.

Research inquiry

We explored the potential for a generative tool that merges rule-based and inferential approaches, building on suggestions from researchers in the field that emphasize the need to bridge the gap between symbolic and sub-symbolic frameworks for developing innovative creative tools (Briot et al., 2020). For that, we investigated a method for combining NNs with constraint solving techniques to create a tool for compositional experiments, incorporating a NN as a rule within the workflow of the constraint solving algorithm. This approach allows the generation of a musical sequence that follows a rule guided by NN predictions while simultaneously being constrained by additional rules, such as desired or allowed notes at certain time points, allowed interval movements, patterns of repetition or non-repetition, and so on. Consequently, these predictions are adjusted to align with a desired musical behavior, even if that behavior deviates from the patterns present in the training dataset. The constraint solver would provide the logical framework to support these more structural aspects of music, while the NN would offer an inferential method where learned musical examples can influence the generation.

Overview and goal

In this article, we present the CAC library 'NeuralConstraints' as the result of our work toward implementing this approach. First, we discuss constraint solvers in the context of music composition. Next, we discuss neural networks for symbolic music generation. We then detail the technical implementation of 'NeuralConstraints' and provide examples of its use. Finally, we outline its limitations and explore potential future enhancements. Importantly, our goal was not to evaluate the success of these combined techniques in 'NeuralConstraints' based on their ability to replicate existing styles but rather on how effectively two distinct computational paradigms—and, ultimately, compositional approaches—can be integrated into a single unified creative environment that can be deployed on a standard computer without requiring large-scale computational infrastructure.

Constraint solvers and rules

A constraint algorithm, or constraint solver (CS) algorithm, is a computational method that searches for values for a set of variables that satisfy a defined set of constraints. The basic idea of constraint programming is that the user specifies the constraints, and a CS is employed to resolve them, ensuring that all conditions or rules are met. Constraints represent relations, and a constraint satisfaction problem defines the relations that should hold among the given variables (Rossi et al., 2006).

In CAC, a CS algorithm allows the generation of sequences of musical elements such as pitches, durations, dynamics, instrumental techniques, articulations, or any other musical information encoded as symbols (Laurson, 1996). The composer can formalize a set of rules to govern the local organization of these elements in the resulting sequence and how they relate to each other globally (Schilingi, 2009; Sandred, 2009; Sandred, 2010; Sandred, 2021).

The primary components of a musical CS algorithm include:

1. Variables: These are the elements that need values assigned. In its application for music composition, for example, a simple melody can be seen as a sequence of *n* pitch variables, each to be assigned by the CS.
2. Domains: Each variable has a domain, which is the set of candidate values that the variable can take. For the same melody, the domain for each variable could be constrained by the melodic or harmonic context at different points in time.
3. Constraints: These are rules that restrict the values that the variables can take. Again, for a melody, we might allow certain notes to follow others, prohibit others in specific positions within the sequence (such as strong or weak beats), or use only notes that fit the specific range of an instrument, etc.

Constraint rules are typically expressed as logical statements in computer code and fall into two categories: *strict* or *heuristic*. Strict rules (also known as 'True/False' rules) evaluate combinations of elements depending on whether they satisfy the rule and return True or False in each case. These rules are always enforced unless it is logically impossible, and the evaluation of the rule returns False. Consequently, a combination of strict rules can significantly reduce the number of possible combinations of musical elements for a desired sequence. Heuristic rules, by contrast, express a preference toward certain solutions, by assigning them higher weights. Normally, the solution with the highest weight is picked by the engine; however, if another strict rule prevents the choice of the solution with the highest weight, the constraint solver will pick a solution with a lower weight so that the strict rule is met. In music applications, heuristic rules play a distinct role in significantly influencing the musical quality of the solution. Strict rules typically establish the framework for an acceptable solution, while heuristic rules often function as musical directions within the generated music (Ebcioğlu, 1990).

The nature of a musical CS algorithm for creative purposes is that the composer needs to define these rules. The rules can be borrowed from traditional musical theory, or created according to a composer's

2 For comprehensive surveys on research on computer-generated music, see Fernández and Vico (2013) and Papadopoulos and Wiggins (1999).

own theories or experimental practice. In any case, they must find ways to describe the properties and relations of the musical situation they are looking for in the form of a logical statement expressed as computer code, in terms that the CS system can understand and enforce.

While rule formalizations, such as those governing polyphony, counterpoint, and harmony, have been fundamental in music theory long before computers, applying these rules efficiently in computational systems has proven to be a highly complex challenge. The initial Monte Carlo approach—generating random sequences of musical elements and repeatedly trying until an acceptable solution meeting predefined rules was found—proved inefficient, even for moderately complex scores. In current days, CS algorithms typically use some version of *backtracking*, where the software can go back and reconsider earlier decisions without having to try every combination of values. While this dramatically increases the efficiency of the algorithm, it can still be a challenge for a computer to find solutions to more complex musical problems[3].

NNs and predictions

A NN operates fundamentally differently from a CS. After a training phase, the NN can be used to predict the continuation of a musical sequence. Various techniques have been developed for symbolic music generation using NNs that essentially rely on these models learning the structure of existing musical examples and generalizing from these learned structures to compose new pieces. More recently, with the advent of Deep Learning[4], common approaches include:

1. Recurrent Neural Networks (RNNs): RNNs, particularly Long Short-Term Memory (LSTM) networks, are well-suited for modeling longer time dependencies of music (Conner et al., 2022; Kumar Arya et al., 2022).
2. Convolutional Neural Networks (CNNs): CNNs have been applied to symbolic music generation tasks. They are often employed within generative adversarial networks (GANs) (Yang et al., 2017).
3. Variational AutoEncoders (VAEs) in combination with RNN architectures have been used for music generation, employing an encoder-decoder structure with latent probabilistic connections to capture musical structure (Koh et al., 2018).

Despite producing compelling results, these methods often rely on local temporal structures that span only a few bars. They also require complex software implementations and potentially large-scale computing infrastructure for training. Furthermore, these methods primarily aim to replicate the musical characteristics of a dataset rather than support an interactive compositional process, which limits creative agency for composers.

A simple feedforward NN can be implemented without extensive computational infrastructure or advanced Machine Learning knowledge. However, once trained, it remains deterministic; the same input will always yields the same output. This assumes a singular, 'best' solution for constructing a musical sequence, an assumption that rarely aligns with the realities of composition. Additionally, we might want other criteria to influence the behavior of our melody, such as changes in the harmonic, or rhythmic context or to correctly fit in a polyphonic texture.

Combining rules and predictions

To address some of the issues discussed above, we propose a system that involves using a feedforward NN as a heuristic rule within a CS algorithm. In this configuration, the temporal structures of the generated music, both shorter and longer, as well as its local and global scope, can be shaped by constraint rules. This approach affords a higher level of creative agency to the composer compared to other systems without requiring complex software implementation or large computational infrastructure.

At first glance, using a deterministic NN would limit the application of other compositional rules during the generative process, as these NNs typically output the highest confidence prediction without an alternative. However, they can also offer an error metric for predictions, which can serve as a weighting factor for heuristic rules. Predictions with lower error values indicate more favorable solutions, while those with higher error values are considered less favorable but are not entirely excluded. This approach introduces flexibility to the CS process, akin to conventional heuristic methods.

Materials

We use two existing algorithms in Common Lisp as starting points for our research: the music constraint solver 'Cluster-Engine' (CE) and the library 'Simple-Neural-Network' (SNN).

Cluster-Engine

CE is a Lisp-based constraint solver designed to address complex musical problems by integrating rhythm, pitch, and meter across multiple voices within a unified sequential search process (Sandred, 2021). Initially developed as a library for the composition environment PWGL (Laurson et al., 2009), CE has since been ported to Max[5], built on the 'Bach'[6] ecosystem (Agostini and Ghisi, 2012), and is now distributed as part of the external package 'MOZ'Lib'[7]. A key feature of 'MOZ'Lib' is its ability to execute Lisp code directly within the Max environment (Vincenot, 2017), facilitating the use of CE and other Lisp-based functionalities in Max.

3 For an in-depth review on CS techniques in music composition, see Anders (2018).

4 Unlike 'shallow' learning, which involves neural networks with a limited number of layers and often requires manual feature extraction, deep learning utilizes neural networks with multiple layers to learn complex feature representations. Deep learning is also well-suited for large datasets as it captures complex patterns and dependencies through its layered architecture.

5 https://cycling74.com

6 https://bachproject.net

7 https://github.com/JulienVincenot/MOZLib

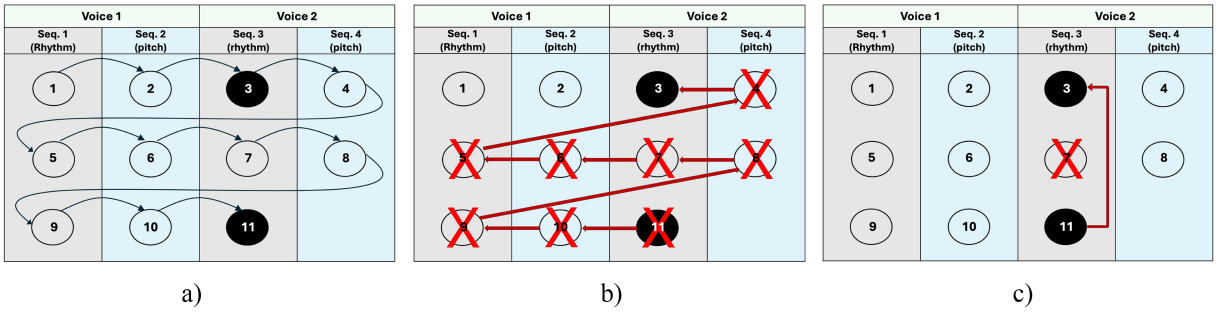

FIGURE 1
(a) Example of CE's normal search process; **(b)** Example of the sequential backtracking method used in other constraint solvers. In the event of a conflict between two variables in a two-voice musical scenario (e.g., variables 3 and 11 in the rhythm sequence for voice 2), all intermediate values are erased; **(c)** CE's backtracking method employs backjumping, which directly targets the source of the conflict. Once variable 3 is reassigned, only variable 7 requires recalculation.

CE enables the codification of musical rules beyond the capacity of monophonic systems, allowing for the reciprocal influence of harmony, rhythm, beat structure, or melodic movement across multiple voices. It supports the use of domains encompassing individual pitches or groups (melodic motifs), intervals or interval groups (intervallic motifs), and individual durations or groups (rhythmic motifs).

CE represents each musical voice using two distinct sequences: one for durations and another for pitches. For a two-voice score, for example, this results in four sequences. Additionally, a global sequence is used to represent metric structure (time signature and subdivision). As a result, solving constraints for two voices would require five engines in total.

CE solves constraints sequentially, selecting candidate values for variables and evaluating them one at a time. Unlike traditional constraint systems, CE splits a problem among various search engines working in parallel, sharing partial solutions. Additionally, they can prompt one another to backtrack during the search process. The advantages of dividing the problem across multiple engines become apparent when logical conflicts arise, and backtracking becomes necessary. CE can identify which variable caused a given conflict, enabling a direct backjump to that position, bypassing intermediate steps. While respecting the variable visitation order, backtracking can occur within any independent sequence, leaving others undisturbed. This significantly improves the process efficiency (Figure 1).

Cluster-Engine rules

A CE rule consists of two parts. The first is the *rule accessor*, which accesses elements in a musical sequence -such as the pitch or duration of a note- or a succession of notes in one or more voices. Accessors vary based on the specific musical information they can access. For instance, one accessor may access pitch or rhythmic information sequentially for a single voice, another may access both pitch and rhythm across multiple voices, while another might be limited to accessing information from different voices only at certain time points in the sequence. The second part is a logical statement that defines the relation between the score elements returned by the rule accessor. This is implemented as a Lisp anonymous (lambda) function.

Rules can be applied in three distinct ways: (1) *Index* rules: These are applied to musical events at specific, fixed positions within the sequence, where each position is represented as a sequentially increasing index number; (2) *Sequence* rules: These rules check the entire sequence of events as it is being built, from the beginning to the current variable being checked; and (3) *Wildcard* rules: Here, the rule is applied stepping through the sequence and accessing events one-by-one or in groups of n adjacent variables (Figure 2).

Simple-Neural-Network[8]

'Simple-Neural-Network' (SNN) is a Common Lisp open-source library that allows for the construction, training, and application of feedforward NNs trained using backpropagation. Users can create NNs for specific prediction or classification tasks by specifying the number of input and output neurons along with the architecture of hidden layers. SNN supports training optimization functions like *batch processing* and *momentum coefficient* and provides functions for retrieving the Mean Absolute Error (MAE) for a single or a set of predictions, or accuracy values for tasks of classification. In addition, it allows for parallel computation support and simple functions for saving and restoring models. The activation function used by the neurons is:

$$A(x) = 1.7159 * \tanh(0.66667 * x)$$

Methods

To establish a bridge between the CS and NN paradigms, we have developed 'NeuralConstraints' (NC) in the Max environment. NC provides a user-friendly visual interface for SNN's training and prediction functionalities using symbolic music datasets. It also works as an add-on to CE, allowing the deployment of trained NNs as rules applicable to the constraint-solving process.

8 Simple-Neural-Network was developed by Guillaume Le Vaillant, and the source code can be found here: https://codeberg.org/glv/simple-neural-network

10.3389/fcomp.2025.1543074

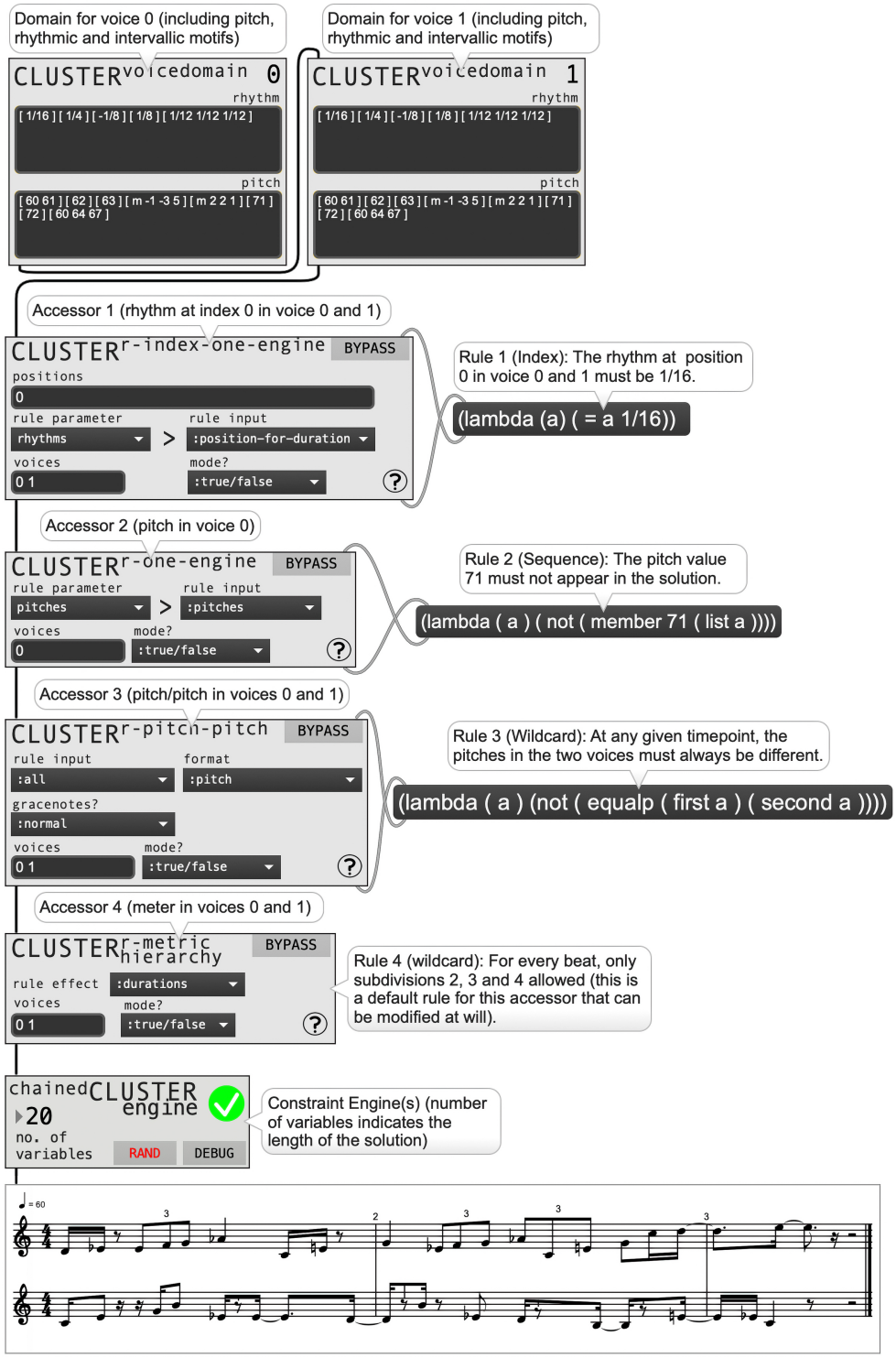

FIGURE 2
The user interface of Cluster-Engine showing examples of four different types of accessors and three different types of rules.

Musical domains

As of now, NC accepts datasets of pitch and rhythmic values. Regarding pitch representation, the NC interface is optimized to receive values in MIDI cents. For the encoding, users can choose between intervals (I), as signed integer values representing the distance in semitones between successive pitches, or pitch class + octave (PC8va) where a pitch is represented by two integers: one for the pitch class (modulo 12) and another for the register, specified by the octave number. For example, the values (0 4) represent a middle (C). The model also supports rhythmic durations (R) expressed as rational numbers (fractions), which are encoded as two integers: the

FIGURE 3
Scheme of the structure of the prediction function.

numerator and the denominator. Rests are indicated by a negative numerator. Each domain can be encoded independently or in combination with another. For example, PC8va + R, or I + R.

Encoding and normalization

In the initial step of encoding a dataset for training a NN, each numeric value in the input and target series is converted into a binary representation, with each bit corresponding to an individual neuron in the NN. In the second step, these binary inputs and targets are then normalized to double-float vectors and finally fed into the NN for training. Conversely, when given an input, the NN predicts values as normalized double-float vectors. To return this result to the original scale of the inputs and targets, an inverse transformation is applied: the prediction must be first denormalized and then converted from binary digits back to the original scale. Below is a scheme of the structure of the prediction function (Figure 3).

Training

Inside the 'NC-train' module, the data is organized into lists[9] of inputs and targets. A key concept is that 'NC-train' arranges the training dataset sequentially to simulate the causality inherent in music. A customizable windowing size determines the number of consecutive input values that lead to a subsequent target, with the target windowing size also customizable. For example, with a windowing size of $n = 4$ inputs and $n = 1$ target, each input list will contain four values, and each corresponding target list will contain one value. This setup enables the neural network to learn—and subsequently predict—values step-by-step, advancing incrementally by one position with each prediction (Figure 4).

The user interface of NC allows for the selection and adjustment of several options for training the NN, such as the windowing size of inputs and targets, the domain (I, R, PC8va, PC8va + R, I + R), the number and size of the hidden layers and the learning rate and number of epochs. In addition, a file name for saving the NN model file on the hard drive must be specified. Other optimization parameters, such as the batch size and momentum coefficient, can also be adjusted. Once all these parameters are set, the training can begin. While it takes place, the ongoing epoch number as well as the corresponding Mean Absolute Error (MAE) of the dataset for that epoch is printed in the Max console. Once the training is complete, the whole sequence of MAE values is displayed as a break-point function. When a NN file is created, it is loaded automatically for testing predictions with original inputs from the model. This facilitates comparing predictions with the real targets, along with the corresponding MAE value (Figure 5).

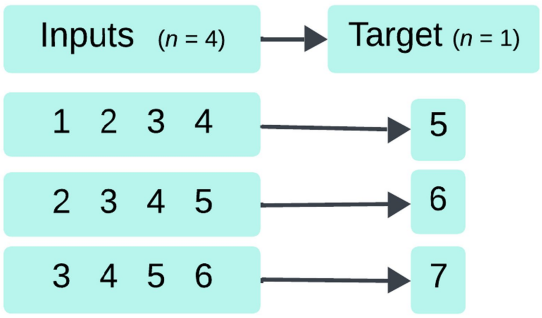

FIGURE 4
Sequential organization of inputs and targets.

Chaining NN and logic rules

In order to apply the NN as a constraint rule, first, it is necessary to load a model file generated by the training module into the CE. The loaded file will be stored in a global variable to be accessed by the engine rule during the search. After this step, it is now possible to add NN rules to the classic chained flow of rules of the CE. NN rules are available as modules that must be connected to a CE accessor pointing to the same musical parameter as the one originally used to train the NN. The example below shows a flow diagram of CE using two rules: a conventional logic rule applied over the rhythmic domain and a NN rule applied over the pitch domain (Figure 6).

MAE rules

SNN provides a function to calculate the Mean Absolute Error (MAE) for a NN over a set of inputs and targets, ranging from the entire dataset to a single data point:

$$MAE = \frac{1}{N \times M} \sum_{i=1}^{N} \sum_{j=1}^{M} | y_{ij} - \hat{y}_{ij} |$$

Where N is the number of input samples, M is the size of the output, y_{ij} is the actual target value and $\hat{y}_{ij}$ is the predicted output value.

9 In Lisp., the list is a fundamental data structure consisting of a sequence of elements: values, symbols, or other lists. This structure is enclosed in parentheses and supports recursive nesting. The quasi-Lispian 'llll' data structure provided by the 'Bach' library provides the integration of Lisp-style nested lists into a Max patch. This structure can theoretically handle nested lists of any length and depth, something not possible by default in Max.

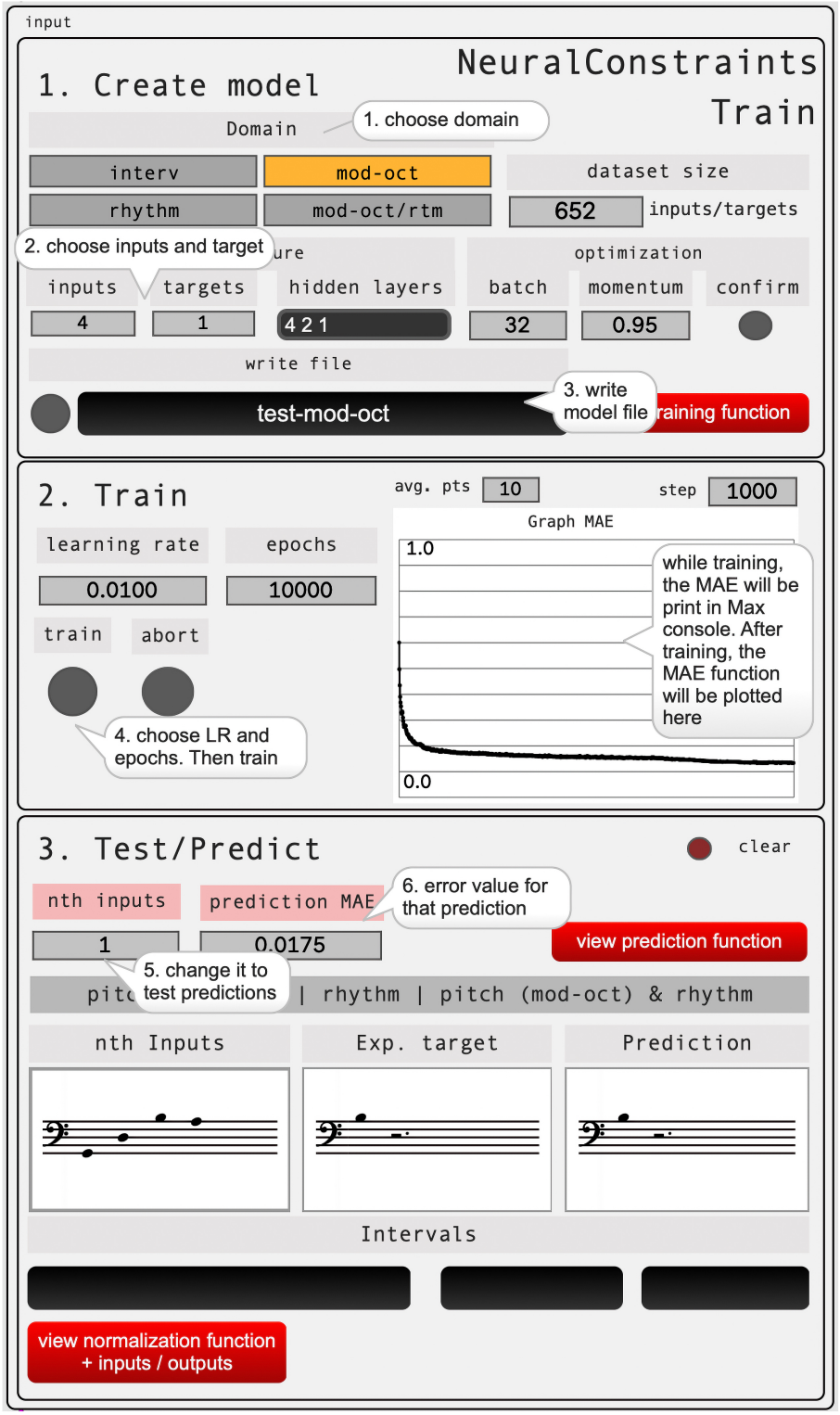

FIGURE 5
The user interface of the module 'NC-Train'.

To obtain the MAE, the function iterates over each input-target pair, where the neural network generates a predicted output for each input, and the corresponding target value (true value) is retrieved. The absolute value of this error is taken and these absolute errors are summed up. Once all inputs are processed, the final MAE is computed as:

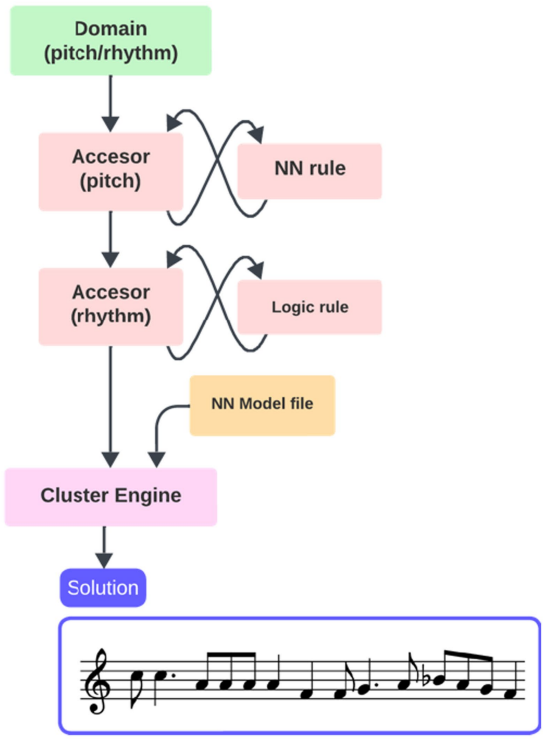

FIGURE 6
Example of a classic chained flow of rules for Cluster-Engine, with two rules at work: a NN rule that accesses pitch information and another logic rule that accesses rhythm.

$$MAE = \frac{total\ absolute\ error}{total\ number\ of\ outputs}$$

When the MAE function is applied to a single input-target pair, the mean calculation simplifies to the absolute error for that specific pair:

$$MAE = |output_i - target_i|$$

Since CE solves constraints sequentially and the SNN is trained to generate outputs in sequence, we experimented with a rule design that uses the negated Mean Absolute Error (MAE $\times -1$) of a single data point as a heuristic weight. At each sequential step, the MAE rule iteratively computes the MAE for each candidate in the CE domain, treating it as the output for the given input list. Since heuristic rules rank solutions based on the highest weight, the MAE must be negated so that values closer to zero correspond to higher weights. As a result, the candidate with the lowest MAE—when negated, yielding the largest weight—is selected as the optimal solution (Figure 7).

In a subsequent refinement of the rule, we replaced the negated MAE value with an optimization function that penalizes higher MAE values and rewards lower ones:

$$weight = \frac{1}{\log(MAE + 1) + 10^{-6}}$$

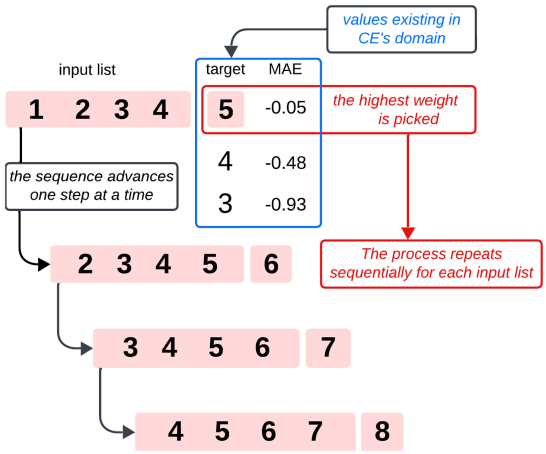

FIGURE 7
Diagram illustrating the sequential and iterative operation of the system using a MAE rule.

TABLE 1 Example of how CE selects the best candidate solution using two versions of the MAE rule in heuristic mode.

Two examples of MAE heuristic rules			
Input list	Candidates from the domain	Weight (negated MAE value)	Weight (MAE after optimization function)
(1 2 3 4)	(3)	−0.93	1.52
(1 2 3 4)	(4)	−0.48	2.55
(1 2 3 4)	(5)	−0.05 (solution picked by CE)	20.49 (solution picked by CE)

The first version uses the negated MAE value as the weight, while the second applies an optimization function to the MAE value that penalizes higher values and rewards lower ones.

The Table 1 provides an example of the process for selecting the best candidate solution using a heuristic rule that calculates the negated MAE, along with another MAE rule that applies the optimization function discussed above to the MAE value.

Prompting

It is recommended to initialize the generation process with a fixed set of domain values corresponding to existing inputs in the dataset. This initialization step helps guide the NN rule toward results resembling the dataset. The module 'NC-Prompt' facilitates this process by allowing the selection of an initial *prompt*[10] from the training dataset. It uses index

10 The word 'prompt' here refers to an initial input that shapes the model's response. While the idea of *prompting* is typically connected to written instructions for large language models (LLMs), it can also encompass other forms of information, such as symbolic representations in musical domains. In the context of 'NeuralConstraints,' a prompt specifically denotes an input that determines the NN's output. Although our prompting mechanism essentially relies on fixing certain variables using index rules—a longstanding approach in CS techniques—we believe the term is particularly fitting, as it aligns with the idea of generating musical continuations, making the concept intuitive for users.

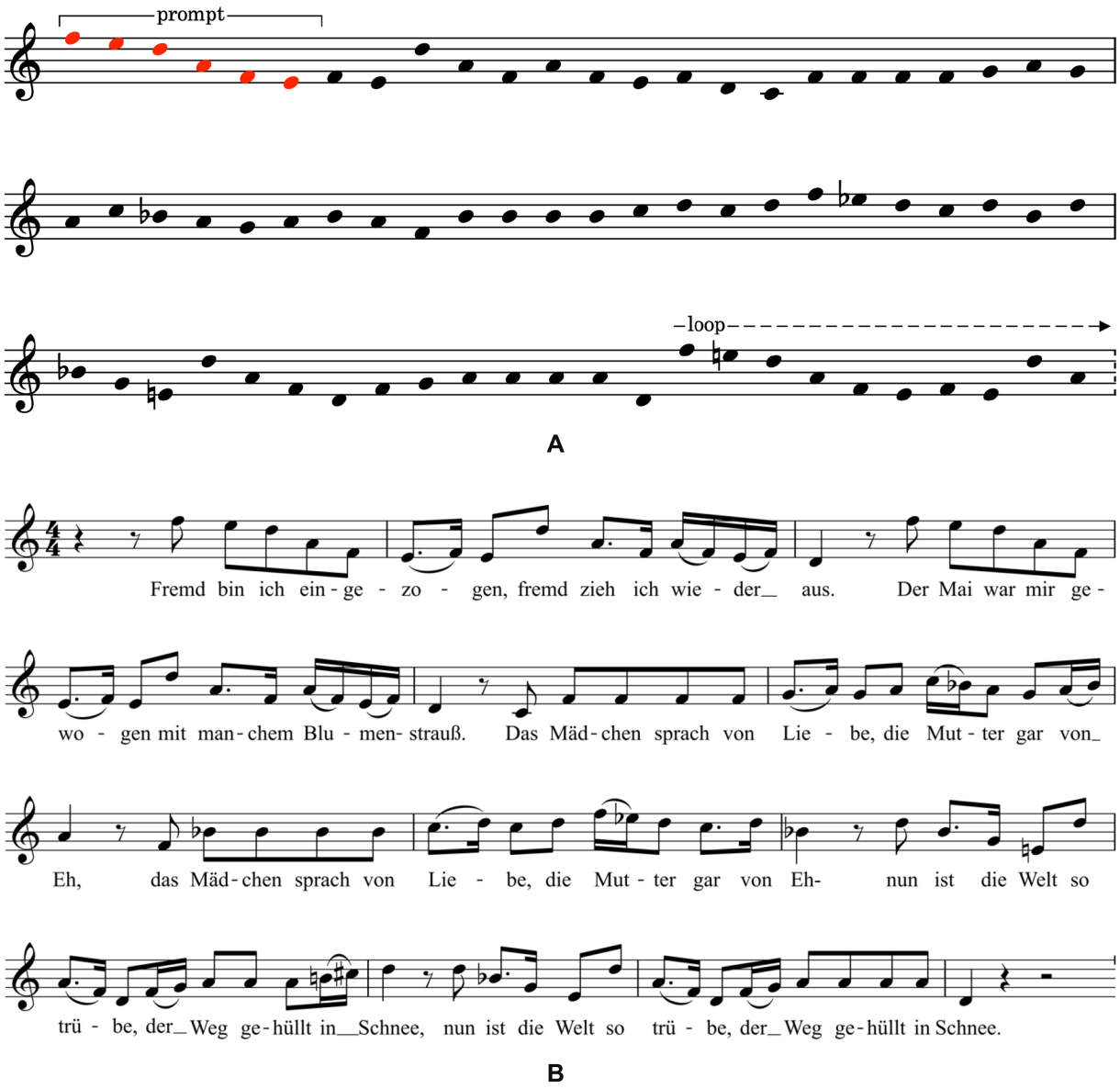

FIGURE 8
(a) Continuation predicted using the first 6 pitch classes from the song 'Gute Nacht'. **(b)** Original vocal melody of the song **Gute Nacht** (first 13 measures). Observe that the neural network accurately predicts the melodic structure of the original piece, with the exception of the repetition in measures 3–4 and the final beat in m. 10/first beat in m. 11.

rules to fix certain variables of the solution (such as the initial ones) to predefined values that the NN takes as input. Additionally, the 'NC-Inputs-Targets' helper module can be used to select a score fragment from the training dataset itself and provide it directly to the 'NC-Prompt' module. Below is a schematic representation of the workflow of NC, including the 'NC-prompt' module, a MAE rule with an optimization function and subsequent evaluation:

Results

For our tests, we trained several NNs using the 'Schubert Winterreise' dataset (Weiß et al., 2021), consisting of the 24 songs of the cycle *Winterreise*, with the music of Franz Schubert (1797–1828) and poetry of Willhelm Müller (1794–1827). In addition, we trained several other NNs using the 'Weimar' dataset of folk melodies (Pfleiderer, 2017).

Using the 'Winterreise' dataset

We trained the model on a corpus of vocal melodies, selected for their adherence to tonal rules[11], focusing solely on the vocal

11 We understand, nevertheless, that a drawback of employing a feedforward NN to predict tonal melodies is that it restricts the analysis to relatively short sequences of inputs, whereas tonality relies on vertical harmonic constructions, as well as possibly longer-span melodic structures.

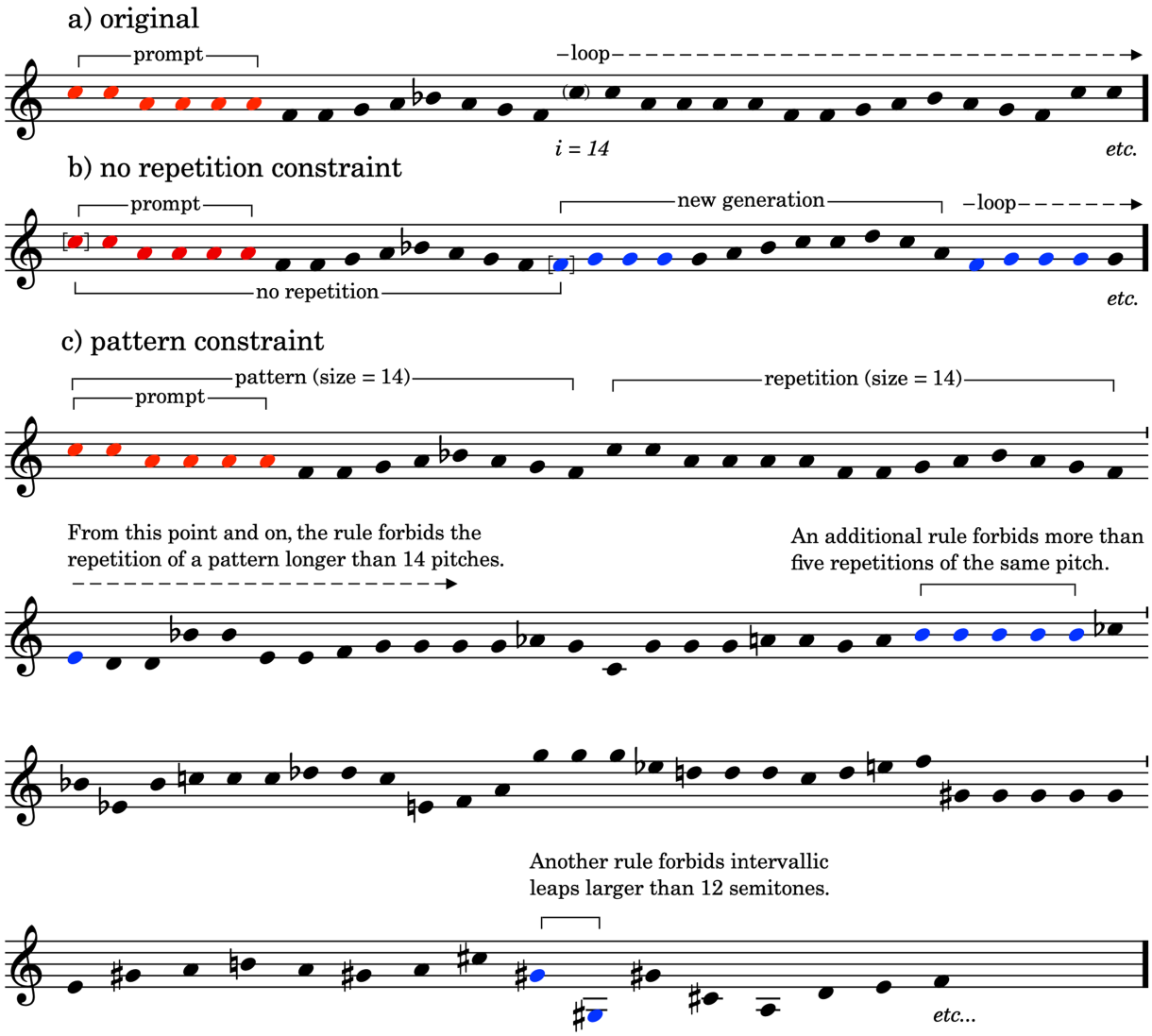

FIGURE 9
(a) shows the original sequence generated using the six first notes of the song *Der Lindembaum;* **(b)** shows the result of applying an index rule preventing pitches 0 and 14 to be the same; **(c)** result of applying a rule to prevent repeating patterns longer than length 14, alongside two additional rules: one limiting interval leaps to 12 semitones and another restricting more than five consecutive repetitions.

melodies and excluding the piano accompaniments. Since we trained the model using the information of pitch class + octave (PC8va), part of the normalization of the dataset involved transposing all the songs to a unique key. This subset of the dataset comprised 4,358 lists of inputs paired with an equal number of targets. The architecture of the network was configured with six inputs (48 neurons), one output (8 neurons), and four hidden layers containing 96, 48, 24, and 16 neurons, respectively. The SNN training function was configured with a batch size of 32, a momentum coefficient of 0.95, and a learning rate of 0.001. The training process ran for 50,000 epochs and the MAE for the dataset converged to approximately 0.07.

It is worth noting that in our pitch-related tests, the *domain* -the pool of pitches that CE can pick from- spans all pitches from MIDI note 53 to 84, a range that covers the full 'Winterreise' dataset. Although the NN was trained using a normalized key across the entire dataset, it would be possible, in theory, to bias it toward more coherent outcomes by constraining the range of possible pitch choices.

We tested the system by applying a MAE heuristic rule and, in place of a prompt, using the first six PC8va from the vocal part of *Gute Nacht*, the opening song of the cycle *Winterreise*. The NN yielded a coherent melodic pattern that spanned up to 62 pitches. The contour of the prediction matched the opening four phrases of the original song, except for inner repetitions. Beyond this point, the melody entered a loop (Figure 8). We observed this as a common occurrence when generating with NN rules. The reason for the looping, as we understand it, comes from the natural behavior of the CE when using a heuristic rule (in this case, a MAE rule), which picks candidates that yield the higher weight, disregarding other potential candidates that, despite yielding lower weights, are also suitable in the musical context.

Combination with logic rules

As an initial step in testing the interaction between the NN and other logic rules, we focused on addressing the issue of the looping occurring

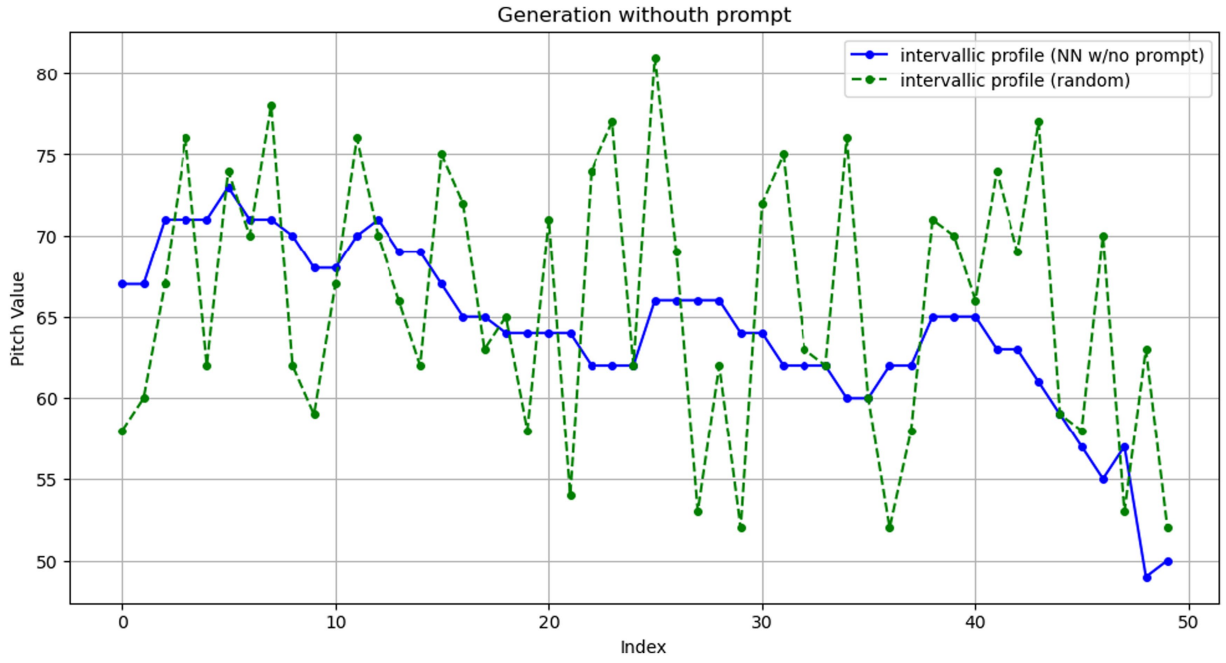

FIGURE 10
Comparison between an intervallic profile generated with a MAE rule without a prompt and a randomly generated profile.

recurrently when using solely NN rules for generation. To tackle this, we explored two approaches. The first, localized in scope, introducing an index rule to ensure that the pitch at the beginning of a loop differs from that picked by the CS, thereby preventing the sequence from restarting:

$$pitch\_at\_index_x \neq pitch\_at\_index_y$$

Using a different prompt—the six initial pitch classes of *Der Lindenbaum* (the fifth song of *Winterreise*)—a loop would begin on pitch 14 (see Figure 9a). By constraining the pitch at index 14 to differ from index 0, the looping at this position was prevented. While this approach avoided looping at position 14 and the new generation continued the original contour of the song, it inevitably led to another loop shortly thereafter (see Figure 9b).

Our second approach introduced a rule based on the function ptrn-find from the CAC library 'OM Morphologie' (Baboni Schillingi et al., 1999)[12], designed to detect repeated patterns within a list of symbols. This rule establishes a maximum threshold for the length of any repeated pattern. For example, setting the maximum length to 14 permits patterns with up to 14 elements but prevents longer repetitions (see Figure 9c). This method offers a global and more adaptive approach to addressing the looping issue. However, its drawback is the increased computational cost when dealing with larger patterns, as the algorithm must identify patterns in every temporary solution for each random pick.

While the first approach was less effective in this particular musical context, it is computationally more efficient than the second. Its flexibility to combine with other rules, such as index-specific constraints applied at various points, still makes it a viable option for experimentation. Ultimately, combining different types of rules yields diverse outcomes, allowing the composer to select the most effective constraint strategies to achieve desired results.

Using the 'Weimar' dataset

We used a subset of the first 100 folk melodies of the 'Weimar' dataset to train a NN on melodic intervals (I). This subset of the dataset comprised only the first 100 melodies. The number of inputs paired with an equal number of targets was 4,391. The network was designed with eight inputs (48 neurons), one output (6 neurons), and four hidden layers containing 96, 48, 24, and 12 neurons, respectively. This time, we allowed the process to run for 100,000 epochs.

Using this model, we tested the system's ability to generate intervallic profiles without an initial prompt. To prevent the CE from randomly selecting values to fill the required number of inputs and targets based on the model's training configuration for MAE calculation—which would result in a randomized profile until the MAE rule begins to influence the generation—we refined the MAE rule so that it incrementally calculates MAE values with a growing list of inputs and a single target (Figure 10).

Two simultaneous NN rules

We explored the application of multiple NN rules concurrently by loading two independent model files into CE. Specifically,

12 https://github.com/openmusic-project/morphologie/releases/tag/v1.1

we combined the two previously discussed NNs—the 'Winterreise' model (PC8va) and the 'Weimar' model (I)—as simultaneous MAE rules, each influencing an independent musical voice. Additionally, we used a CE accessor for pitch information on simultaneous independent voices, incorporating a rule to constrain the harmonic intervals between them. These intervals, expressed as pitch class (modulo 12) differences, were restricted to 3, 4, 7, 8, 9, or 0 at the start of every beat:

$$pitch\_class\_voice_0 - pitch\_class\_voice_1 = \{0,3,4,7,8,9\}$$

Another rule ensured that the 'Winterreise' voice consistently remained above the 'Weimar' voice:

$$pitch\_in\_voice_0 \geq pitch\_in\_voice_1$$

For initialization, the 'Winterreise' model was prompted with the first six pitches of *Gute Nacht* (Figure 11), while the 'Weimar' model was initialized without a prompt. The rhythmic domain for each voice was restricted to quarter-notes ('Winterreise') and sixteen-notes ('Weimar'). The result of this test is shown in Figure 12.

Using NN rules trained on integrated domains

NC facilitates the training of a NN using an encoding that integrates two musical domains, such as pitch class/8va + rhythm (PC8va + R) or intervals + rhythm (I + R). This model can then function as a rule for a CE accessor capable of simultaneously accessing pitch and rhythm information. To test this type of rule, we trained a model using a reduced version of the 'Weimar' dataset, which included only the first ten songs, comprising 505 input-target pairs. The information from PC8va and rhythmic duration (PC8va + R) was integrated and encoded for each value in the input-target lists. The NN was designed with six inputs (84 neurons), one output (14 neurons), and four hidden layers containing 168, 84, 42, and 28 neurons, respectively.

In our tests using this model, we found that the MAE heuristic rule was too weak, either having no observable impact on the solution or resulting in different solutions for each generation. This behavior is peculiar because, even though the NN rule is heuristic, it should consistently favor a preferred solution as the best candidates' weight prevails. To address this, we experimented with setting MAE thresholds to determine acceptable results, effectively turning the MAE heuristic rule into a True/False rule. It's important to note that heuristic rules are not backtracked in the CE; only True/False rules have this capability.

We established a threshold of 0.15 for non-negated MAE metrics without computing the optimization function, forcing the engine to select only predictions below this threshold. In this scenario, the impact of the rule was observable; however, the CE often experienced excessively long iterations or became stuck in backtracking loops. Below are some results using PC8va + R prompts (Figure 13).

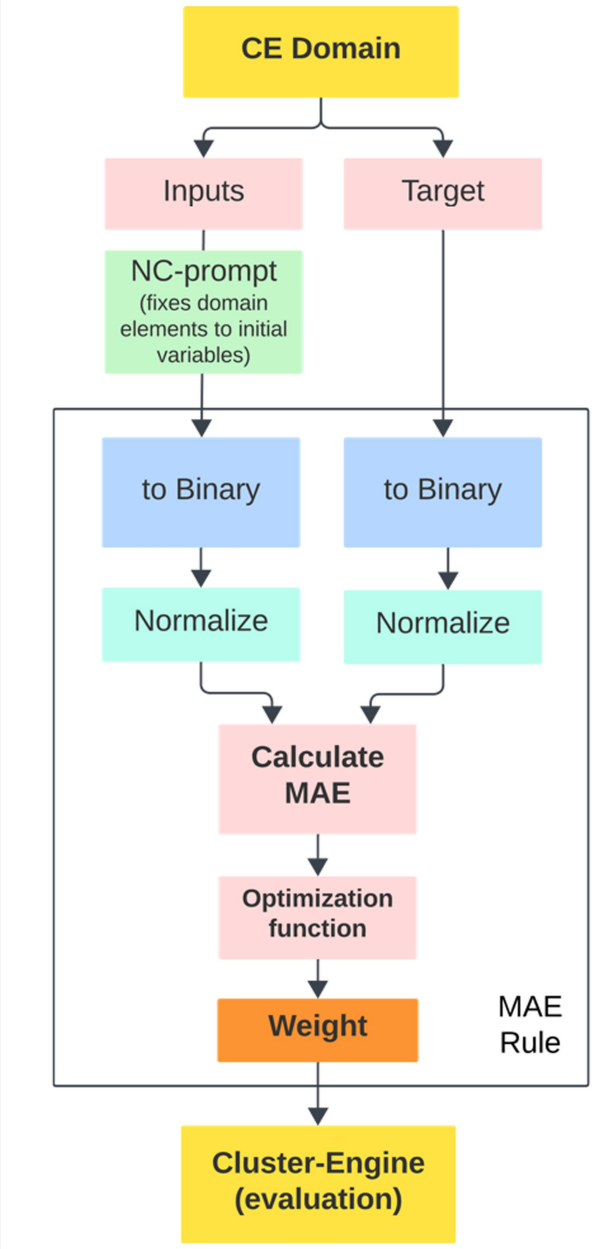

FIGURE 11
Schematic representation of the data flow of inputs with NC-prompt, and the algorithm of a MAE rule.

While MAE metrics for these integrated models dropped significantly during extended training, and the prediction/test results in the 'NC-train' module were generally accurate, handling pitch and rhythm simultaneously with an NN rule in heuristic mode poses challenges for the CE. Although we did not notice issues with weak heuristic rules when using single-domain rules, these problems became evident when using a CE accessor for both pitch and rhythm information simultaneously. Using the MAE rule in strict mode by setting MAE thresholds worked as expected, however, the options for

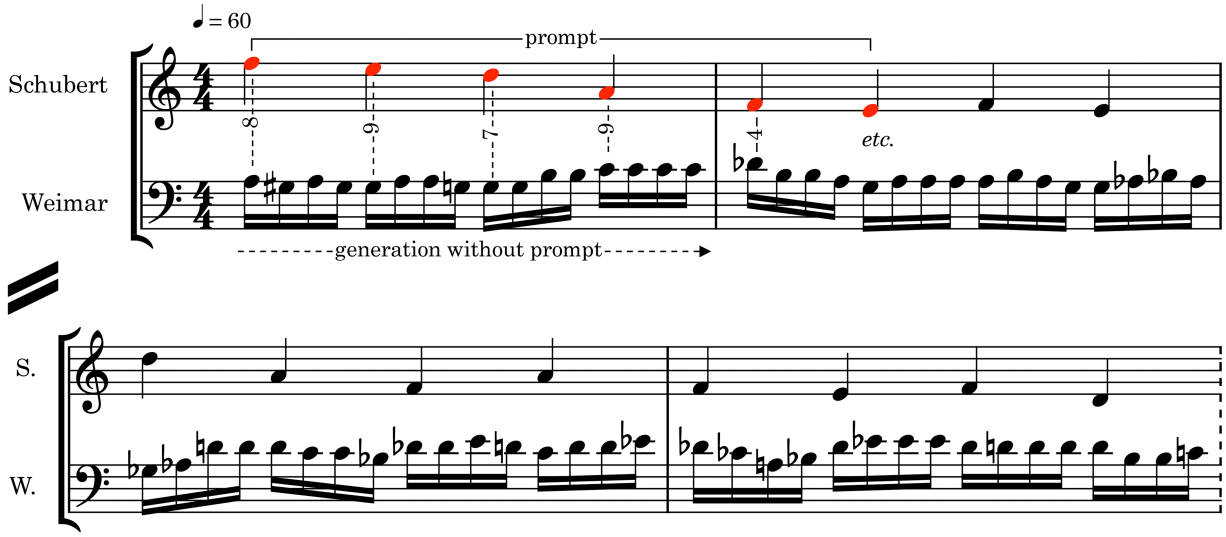

FIGURE 12
Example of using two simultaneous NN rules for independent musical voices, establishing harmonic intervallic constraints between them.

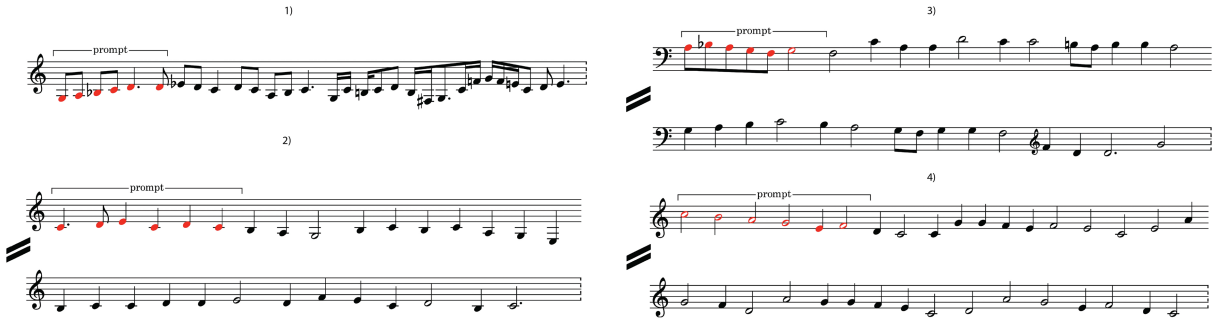

FIGURE 13
Various results using a NN trained on combined domains (PC8va + R) setting a MAE threshold of 0.15.

the CE became significantly restricted, as error values vary largely across the dataset, often forcing it to backtrack and become stuck in backtracking loops.

Discussion

One of the main challenges when generating music sequences is the necessity of modeling coherent both long- and short-span structures. While constraint rules can introduce some control on the longer-term musical behavior, the current architecture of SNN can only learn relatively short-term musical patterns. Therefore, we view our current tests as a starting point, potentially directed toward implementing more advanced NNs[13] as rules that can better account for longer-term musical aspects. However, handling longer-span structures can introduce limitations on the CE side, as these can become computationally expensive, as observed when using the 'OM Morphologie' function.

Currently, the rhythmic domain learns sequences of durations but lacks awareness of their metric structure. Upcoming refinements to the system include to enhance the system's ability to recognize a metric domain—a domain already manageable by the CE—that could enhance the NN's ability to recognize musical patterns based on metric structures.

We considered introducing a parameter to the neural network rule—similar to the 'temperature' parameter in some stochastic networks—to inject randomness and reduce looping behavior. We believe this effect can be achieved on the CE rule side by incorporating an operation that acts directly on the weights, closely aligned with our approach to implementing the optimization function.

13 For example cf. https://github.com/melisgl/mgl

A promising approach could involve using NN rules trained on parameters beyond pitches, durations, and meter, such as dynamics, articulations, or other musical metadata. This is feasible as CE can establish constraint relations between these using an experimental feature currently in development. However, a significant challenge is the increased computational demand of training high-dimensional models, as NC is not built to rely on large-scale computational infrastructure. Additionally, this may require refinements to the CE architecture to enhance its effectiveness and efficiency when exploring integrated musical domains.

In future implementations, we aim to investigate the system in real-time musical contexts such as improvisation or interactive installations. Although the NN must be trained offline, rules can be adjusted in real-time, providing flexibility to the process. CE has already shown robust performance in several live situations[14], indicating its viability for real-time applications.

Conclusion

In this article, we introduced 'NeuralConstraints,' a computer-assisted composition library that combines symbolic neural generation and constraint-based computation. Our primary objective was to explore a tool that could enhance compositional control over a neural generative process. To achieve this, we integrated two Lisp-based algorithms: the music constraint solver 'Cluster-Engine' and the feedforward neural network 'Simple-Neural-Network,' positioning the latter as a heuristic rule within a constraint-based composition workflow. Our tests, using the 'Schubert Winterreise' dataset and the 'Weimar' dataset of folk melodies, involved training models on musical parameters like pitch class + octave, intervals, rhythm, or combinations of these. Our results demonstrated that the NN effectively integrated into CE as a heuristic and strict rule, shaping musical sequences based on the training data. Furthermore, the interaction between NN rules and logical constraint rules successfully produced coherent musical outcomes, as shown in the examples. Although our goal was not primarily to assess their musical quality, the system yielded coherent and potentially interesting results that could inspire further compositional exploration. Still, challenges remain in its present form, such as dealing with longer-term music patterns, including meter and other music parameters for the training of more complex models, and further research is needed around using the system for real-time artistic endeavors.

Data availability statement

The source code, abstractions, example patches and models created for this study can be found in the following link: https://github.com/juansv2k2/NeuralConstraints.git. The 'Schubert Winterreise' datasets used for this research can be found in the link https://zenodo.org/records/3968389. The 'Weimar' dataset of folk melodies used for this research can be found in the link https://jazzomat.hfm-weimar.de/dbformat/dbformat.html.

Author contributions

JVa: Conceptualization, Data curation, Formal analysis, Funding acquisition, Investigation, Methodology, Project administration, Resources, Software, Validation, Visualization, Writing – original draft, Writing – review & editing. ÖS: Conceptualization, Investigation, Project administration, Software, Supervision, Writing – original draft, Writing – review & editing. JVi: Conceptualization, Investigation, Software, Validation, Visualization, Writing – review & editing.

Funding

The author(s) declare that financial support was received for the research and/or publication of this article. The research is funded by the Norwegian Artistic Research Program. However, the publication fee is covered by the University of Bergen.

Acknowledgments

JVa wants to thank the Norwegian Artistic Research Program for funding his Doctoral research and the University of Bergen for providing the infrastructure and administrative resources to carry out this project.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Generative AI statement

The authors declare that Gen AI was used in the creation of this manuscript. To help edit the manuscript. The Generative AI is not listed as an author of the manuscript, the content edited using the Generative AI has been checked for factual accuracy and plagiarism. The normal prompt used is 'Refine. Make only minor editorial adjustments'. Model: ChatGPT 4o, ChatGPT 4.

Publisher's note

14 For example, the sound installation 'Sonic Trails' by Örjan Sandred. The documentation of the work can be accessed here: https://sandred.com/sonictrails

References

Agostini, A., and Ghisi, D. (2012). Bach: an environment for computer-aided composition in max. Icmc 2012: Non-Cochlear Sound - Proceedings of the International Computer Music Conference 2012, 373–378.

Ames, C. (1989). The Markov process as a compositional model: a survey and tutorial. *Leonardo* 22, 175–187. doi: 10.2307/1575226

Anders, T. (2018). Compositions created with constraint programming. New York, NY, United States: The Oxford Handbook of Algorithmic Music.

Baboni Schillingi, J., Voisin, F., and Sarhan, F. (1999). OpenMusic Morphologie: Fonctions d'analyse, de reconnaissance, de classification et de reconstitution de séquences symboliques et numériques. *Ircam documentation*. 16.

Boden, M. A. (2004). The creative mind: Myths and mechanisms. *2nd* Edn. London,New York: Routledge.

Briot, J.-P., Hadjeres, G., and Pachet, F.-D. (2020). Deep learning techniques for music generation. Cham, Switzerland: Springer.

Brooks, F. P., Hopkins, A. L., Neumann, P. G., and Wright, W. V. (1957). An experiment in musical composition. *IRE Trans. Electron. Comput.* Ec-6, 175–182. doi: 10.1109/TEC.1957.5222016

Conner, M., Gral, L., Adams, K., Hunger, D., Strelow, R., and Neuwirth, A. (2022). Music generation using an Lstm. doi: 10.48550/arXiv.2203.12105

Ebcioğlu, K. (1990). An expert system for harmonizing chorales in the style of Js Bach. *J. Log. Program.* 8, 145–185. doi: 10.1016/0743-1066(90)90055-A

Fernández, J. D., and Vico, F. (2013). Ai methods in algorithmic composition: a comprehensive survey. *J. Artif. Intell. Res.* 48, 513–582. doi: 10.1613/jair.3908

Frantz, R. (2003). Herbert Simon. Artificial intelligence as a framework for understanding intuition. *J. Econ. Psychol.* 24, 265–277. doi: 10.1016/S0167-4870(02)00207-6

Hiller, J. L. A., and Isaacson, L. M. (1958). Musical composition with a high-speed digital computer. *J. Audio Eng. Soc.* 6, 154–160.

Koenig, G. M. (1970). The use of computer programmes in creating music. *In Music and Technology (Proceedings of the Stockholm Meeting organized by UNESCO)*, Paris: La Revue Musicale. 93–115.

Koh, E. S., Dubnov, S., and Wright, D. (2018). Rethinking recurrent latent variable model for music composition. doi: 10.48550/arXiv.1810.03226

Kumar Arya, P., Kukreti, P., and Jha, N. (2022). "Music generation using Lstm and its comparison with traditional method" in Advances in transdisciplinary engineering (Ios Press). doi: 10.3233/ATDE220793

Laurson, M. (1996). PatchWork: A visual programming language and some musical applications. Helsinki, Finland: Sibelius Academy.

Laurson, M., Kuuskankare, M., and Norilo, V. (2009). An overview of Pwgl, a visual programming environment for music. *Comput. Music. J.* 33, 19–31. doi: 10.1162/comj.2009.33.1.19

Pachet, F., and Roy, P. (2011). Markov constraints: steerable generation of Markov sequences. *Constraints* 16, 148–172. doi: 10.1007/s10601-010-9101-4

Papadopoulos, G., and Wiggins, G. (1999). Ai methods for algorithmic composition: A survey, a critical view and future prospects. Aisb'99 symposium on musical creativity. Brighton, Uk: Aisb (Society for the Study of Artificial Intelligence and the Simulation of Behaviour).

Pearce, M. T., and Wiggins, G. A. (2007). Evaluating cognitive models of musical composition. Proceedings of the 4th international joint workshop on computational creativity, 73–80.

Pfleiderer, M. (2017). Inside the Jazzomat: New perspectives for jazz research. Mainz, Germany: Schott Campus.

Pohjannoro, U. (2016). Capitalising on intuition and reflection: making sense of a composer's creative process. *Music. Sci.* 20, 207–234. doi: 10.1177/1029864915625727

Pohjannoro, U. (2021). Mind the body: materiality and physicality in a composer's thinking process. *Psychol. Music* 50, 1169–1183. doi: 10.1177/03057356211034916

Ramanto, A. S., and Maulidevi, N. U. (2017). Markov chain based procedural music generator with user chosen mood compatibility. *Int. J. Asia Digit. Art Design Assoc.* 21, 19–24. doi: 10.20668/adada.21.1_19

Rossi, F., Van Beek, P., and Walsh, T. (2006). Handbook of constraint programming. Chantilly, The Netherlands: Elsevier Science & Technology.

Sandred, Ö. (2009). Approaches to using rules as a composition method. *Contemp. Music. Rev.* 28, 149–165. doi: 10.1080/07494460903322430

Sandred, Ö. (2010). Pwmc, a constraint-solving system for generating music scores. Source. *Comput. Music. J.* 34, 8–24. doi: 10.1162/comj.2010.34.2.8

Sandred, Ö. (2021) in Constraint-solving Systems in Music Creation. ed. E. R. Miranda (Cham: Springer International Publishing).

Schilingi, J. B. (2009). Local and global control in computer-aided composition. *Contemp. Music. Rev.* 28, 181–191. doi: 10.1080/07494460903322455

Simon, H. A., and Mellon, C. (1995). Explaining the ineffable: Ai on the topics of intuition, insight and inspiration. *Ijcai* 1, 939–949.

Todd, P. M. (1989). A connectionist approach to algorithmic composition. *Comput. Music. J.* 13, 27–43. doi: 10.2307/3679551

Vassallo, J. S. (2024). Exploring musical procedural rhetoric: computational influence on compositional frameworks and methods in the piece "elevator pitch". *Int. J.Music Sci Technol. Art* 6, 1–16. doi: 10.48293/IJMSTA-114

Vincenot, J. (2017). Lisp in max: exploratory computer-aided composition in real-time. Icmc 2017 proceedings (Shanghai, 2017).

Voisin, F., and Meier, R. (2009). On analytical vs. schizophrenic procedures for computing music. *Contemp. Music. Rev.* 28, 205–219. doi: 10.1080/07494460903322489

Weiß, C., Zalkow, F., Arifi-Müller, V., Müller, M., Koops, H. V., Volk, A., et al. (2021). Schubert Winterreise dataset: A multimodal scenario for music analysis. *J. Comput. Cult. Herit.* 14, 1–8. doi: 10.1145/3429743

Wiggins, G. A. (2006). A preliminary framework for description, analysis and comparison of creative systems. *Knowl.-Based Syst.* 19, 449–458. doi: 10.1016/j.knosys.2006.04.009

Wiggins, G. A. (2012). Defining inspiration? Modelling the non-conscious creative process. The Act of Musical Composition: Studies in the Creative Process. UK: Routledge.

Xenakis, I. (1992). Formalized music: Thought and mathematics in composition. Hillsdale, N.Y: Pendragon Press.

Yang, L.-C., Chou, S.-Y., and Yang, Y.-H. (2017). MidiNet: A convolutional generative adversarial network for symbolic-domain music generation. doi: 10.48550/arXiv.1703.10847