UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

MUSIB: MUSIC INPAINTING BENCHMARK

TESIS PARA OPTAR AL GRADO DE
MAGÍSTER EN CIENCIAS, MENCIÓN COMPUTACIÓN

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN COMPUTACIÓN

MAURICIO JESÚS ARANEDA HERNÁNDEZ

PROFESOR GUÍA:
FELIPE BRAVO MARQUEZ

PROFESOR CO-GUÍA:
DENIS PARRA SANTANDER

MIEMBROS DE LA COMISIÓN:
NELSON BALOIAN TATARYAN
EDUARDO GRAELLS GARRIDO
ELIANA SCHEIHING GARCIA

SANTIAGO DE CHILE
2023

MUSIB: Referencia de evaluacion para autocompletado musical contextual

La Generación Automática de Música, y en particular, el Inpainting (o autocompletación contextual) de Partituras Musicales, es un tema fascinante en la investigación del Aprendizaje Automático, sin embargo, actualmente no es posible comparar los enfoques que resuelven esta tarea: las métricas, los conjuntos de datos y las representaciones de los datos difieren de un artículo a otro, lo que dificulta conocer el progreso de la comunidad. En este contexto, el desarrollo de un benchmark estandarizado para evaluar adecuadamente estos modelos sería de gran ayuda a efectos de análisis y reproducibilidad.

En esta tesis discutimos cómo los diferentes modelos generan música y cómo podemos resolver este problema de evaluación con MUSIB: the Musical Score Inpainting Benchmark, nuestra propuesta de nuevo benchmark con condiciones estandarizadas. Para evaluar adecuadamente estos modelos proponemos dos conjuntos de métricas extendidas desde la literatura: *Métricas de notas* y *Métrica de divergencia*. La primera se basa en la comparación individual de los atributos de las notas, mientras que la segunda se basa en la comparación de las distribuciones entre el conjunto de datos original y los datos generados artificialmente.

Nuestros experimentos sugieren hallazgos interesantes en relación con el estado del arte del inpainting de partituras musicales. 1) La reproducibilidad de los modelos no es del todo precisa, lo que demuestra que todavía se puede mejorar para que los modelos sean más reproducibles. 2) El rendimiento de los modelos depende en gran medida de la cantidad de datos de entrenamiento. 3) El rendimiento de todos los modelos varía según el conjunto de datos. 4) Todos los modelos aprenden algunos aspectos de la música mejor que otros (el ritmo por sobre el tono, por ejemplo).

Este trabajo constituye la primera aproximación a la evaluacion estandarizada del inpainting musical, siendo un importante soporte para el estudio de la Generación Musical Automatizada.

MUSIB: Music Inpainting Benchmark

Automatic Music Generation, and in particular, Musical Score Inpainting, is a fascinating topic in Machine Learning research, however, it is currently not possible to compare approaches that solve this task: metrics, datasets, and data representations differ from paper to paper, which hinders the progress the community has made. In this context, the development of a standardized benchmark to properly evaluate these models would be of great help for analytical and reproducibility purposes.

In this thesis, we discuss how different models generate music and how we can solve this evaluation problem with MUSIB: the Musical Score Inpainting Benchmark, our proposed new benchmark with standardized conditions. In order to properly evaluate these models we propose two sets of metrics extended from the literature: *Note Metrics* and *Divergence Metrics*. The first relies on a one-on-one comparison of notes attributes while the second relies on the comparison of distributions between the original dataset and the artificially generated data.

Our experiments suggest interesting findings regarding the start of the art for musical score inpainting. 1) The replicability of models is not quite accurate, showing that there is still room for improvement in making models more reproducible. 2) The performance of models is highly dependent on the amount of training data. 3) The performance of all models varies as the dataset varies. 4) All models learn some aspects of music better than others (rhythm over the pitch, for example).

This work constitutes the first approach to benchmark the musical inpainting task, being an important support for the study of Automated Music Generation.

A Erick por enseñarme a amar la música.
A Daniel por enseñarme a amar la programación.

Agradecimientos

Primero que todo, le agradezco a mis padres, Marisol y Alan, por siempre inculcarme el espiritu de aprender más y superarme, gracias a ellos puedo decir que estoy donde estoy. Agradezco a mis hermanos Erick y Daniel por compartirme sus pasiones en las areas que disfrutan, como son la musica y la programacion. De ellos viene gran parte de la inspiracion para esta tesis y ademas para quien soy hoy.

En segundo lugar, agradezco profundamente a Valeria por ser mi compañera de vida estos años y por apoyarme en las alegrías, penas y frustraciones que me ha traído el proceso de escribir una tesis.

En tercer lugar, agradezco a mis profesores guias, Felipe y Denis, por su apoyo a lo largo de esta tesis. A Felipe por sus esfuerzos en enseñarme a aplicar buenas metodologias para mi investigacion y sobre todo su motivacion por querer aprender nuevas cosas junto a mi en esta area no tan explorada como es la generacion de musica automatica, y a Denis por recibirme y compartir conmigo sus conocimientos del mundo de la creatividad e inteligencia artificial junto al grupo de CreativAI.

Le agradezco tambien a mis amigos, Benjamin, Nicolas y Jose por compartir conmigo el amor por la musica por años desde los tiempos del colegio. De la misma forma agradezco al Chavo, al Santos, a Rodrigo, y a Matias por ser grandes amigos que formé en la universidad, con quienes ademas en algun momento pude discutir ideas sobre esta tesis.

Finalmente, agradezco el financiamiento rebicido por parte del Instituto Milenio Fundamento de los Datos (IMFD), junto con la infraestructura de computo provista por ReLeLa.

Table of Content

List of Tables

List of Figures

Capítulo 1

Introduction

Automatic music generation is a research topic in machine learning that is especially useful for assisting musicians in the process of composing new music. Its goal is to develop computational systems that provide recommendations to enhance human creativity.

Multiple models have been proposed to assist in this human-creative process. In recent years, deep learning techniques have emerged as the tool of choice for model design in this field, mainly due to their ability to learn complex implicit rules and temporal dependencies in data [6].

However, most approaches to music generation assume that new music is conditioned by the music that precedes it, which forces sequential generation. This introduces limitations on the degree of interaction between musicians and computers, as musicians' composition is highly non-sequential: different segments of a piece are made, then expanded, combined, and refined in several stages to completion, rather than a single sequential stage from beginning to end [4].

For this reason, we focus on the *Musical Score Inpainting* (or *Infilling*) task, which aims to fill in incomplete pieces of music to better mimic the non-sequential musical composition process. In this configuration, musicians can enter incomplete ideas they are working on and receive recommendations on how to progress that are specifically tailored to their context, making interaction easier and more intuitive.

Although several methods have been proposed to address this task, there is no comparable configuration among them: training and evaluation are performed in different datasets using different metrics as well. This brings a problem when comparing approaches, hindering the progress development in the state of the art and the understanding of what is useful or not.

To address these issues, we propose an evaluation procedure to serve as a suitable standardization framework for all music inpainting methods. First, we compiled the four most recent models up to 2021 and adapted them from their source code so that they could all be run in a single environment. These models share the same base framework (PyTorch), which made the adaptation appropriate to the time span of the project. To validate our correct reproduction of these methods, we performed experiments to compare the metrics stated in each work with the results we obtained. We then defined our evaluation procedure and applied it to these four models to analyze the results. In the next section, we describe in detail the technical problem and standardization challenges in the evaluation of our selected models.

1.1. Problem Statement

The evaluation of models for Musical Score Inpainting presents significant challenges in terms of defining a standardized framework to adequately measure performance:

- First, the metrics previously proposed in the literature cannot be used as-is: the results for these metrics differ when applied to different representations of the same data [23]. This is undesirable as proper evaluation requires comparing multiple methods with different representations.
- Second, the sets of metrics utilized for evaluation on each method proposal differ from paper to paper. This implies that different evaluations measure different properties that they do not share with each other, making them not comparable.
- Third, methods are applied to different datasets with different characteristics such as data format, number of samples, average song length, and notes/rhythms distributions, among others. Consequently, the performance of a model in one dataset is not necessarily the same in another.
- Finally, the output of each model is generated by a random process to encourage variability in the recommendations. This makes the evaluation process non-trivial, as there may be several samples generated that sound good in context even though they are not an exact reproduction of the true data.

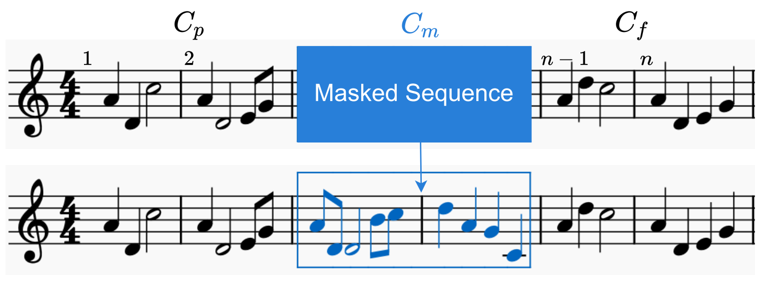


Figure 1.1: Musical Score Inpainting setup. The measures in the middle of a score are unknown at first, then generated and conditioned to the surrounding context.

That said, it is not clear whether it is possible to formulate an evaluation procedure that brings together all previous evaluations into a single framework that can be applied to general music inpainting models.

1.2. Hypothesis

In this thesis, we hypothesize that it is possible to find a unifying pattern across several models of musical score inpainting that enables a direct comparison of performances for multiple AI methods. Furthermore, we argue that it is possible to extend current evaluation procedures to include measures on the expected variability of model's output.

1.3. Objectives

1.3.1. General Objective

The main objective of our research is to develop an evaluation framework that allows to properly compare different approaches for musical score inpainting, thus providing solid evidence to define the current progress of this task and its state of the art. The idea is to establish which model works best under defined circumstances and metrics, replicating, integrating and testing previously proposed methods for comparison with different datasets.

1.3.2. Specific Objectives

1. Propose and develop an evaluation method that is agnostic to the data representations chosen by the different models of music inpainting.
2. Introduce a formalization of Note Metrics and Divergence Metrics: two types of evaluation derived from our methodology.
3. Benchmark and analyze the results of each music inpainting model under standardized conditions.
4. Integrate models for musical score inpainting in an open-source library for easy reproducibility of the experiments conducted.

1.4. Methodology

This section presents the methodology proposed to fulfill the specific objectives of our research. Concisely, our work mainly consists of the following steps:

1. Literature review of music generation and musical score inpainting and selection of models for the task based on two criteria: source code availability and recency of publication.
2. Reproduce the implementation of an existing model for musical score inpainting. The idea is to determine how to generalize the data flow for several models from a study

case. In particular Data Processing, Feature Extraction, Model Architecture, Training, and Evaluation.

3. Design a module to serve as a unified pipeline to transform raw data from MIDI/MXL format to vector representations needed for each model and their consequent training/evaluation. This requires the abstraction of the logic of all model pipelines.

4. Study the replicability of experiments for each model when compared to their former published datasets, taking as a comparison the Loss value declared on each publication.

5. Design and implementation of our proposed evaluation pipeline, which includes Note Metrics and Divergence Metrics. We formalize their properties for standardization and generalization with respect to different data representations.

1.5. Thesis Structure

The remainder of this thesis is organized as follows:

In Chapter 2 we discuss the theoretical background necessary to understand the concepts presented in our research. Chapter 3 presents the models, datasets, and metrics that conform to our proposed benchmark MUSIB. Chapter 4 describes code design and implementation details. Finally, the last chapter summarizes the conclusions of this work and discusses possible extensions of the project for future work.

Capítulo 2

Background and Related Work

This chapter presents the scientific disciplines involved in our research and the related bibliography. First, an overview of the area of knowledge in which this thesis is developed and its methods is given. Then, the overview of Musical Generation is explained: preliminary musical concepts, digital representation of music, vector data encodings and the formalization of our studied sub-task Musical Score Inpainting. Next, in related work, an overview of benchmarks and previous evaluations performed by other research communities for different tasks in the context of machine learning is presented to show the need for stronger replicability standards. Finally, it presents current approaches to music inpainting and with what metrics these approaches have been measured.

2.1. Scientific Disciplines

2.1.1. Artificial Intelligence

Artificial intelligence (AI) is a branch of computer science that theorizes and develops computer systems capable of performing tasks that normally require human intelligence, such as visual perception, speech recognition, decision-making, language translation, and even tasks to enhance human creativity, such as automated music generation or text-to-image systems such as Dall-e 2 [43].

To be considered artificially intelligent, a system would need at least these capabilities (as described in [34]):

- Knowledge representation: able to hold knowledge and store it somewhere.
- Automated reasoning: able to reason based on stored knowledge.
- Machine learning: able to learn from its environment (data).

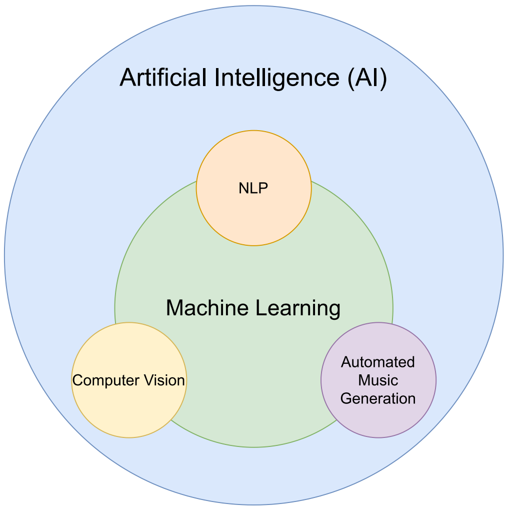


Figure 2.1: Diagram showing the relation of Artificial Intelligence, Machine Learning, and some disciplines such as Computer Vision, Natural Language Processing (NLP), and Automated Music Generation

2.1.2. Machine Learning

Machine learning consists of the use and development of computer systems that are capable of learning and adapting without following explicit instructions, by using algorithms and statistical models to analyze and draw conclusions from patterns in data.

Machine Learning algorithms are often classified into unsupervised and supervised models, depending on the degree of human interaction required to use each approach.

Supervised machine learning learns the relationship between input data and labeled output data. The vast majority of available data is unlabeled raw data. Human interaction is usually necessary to accurately label data ready for supervised learning. This makes this kind of algorithm dependent on human intervention, hence the name *supervised*. Once a model has been trained to predict labels, it is often used to classify new and unseen datasets or to predict outcomes given previous data.

Unsupervised machine learning, on the other hand, identifies patterns and trends in raw datasets, by clustering similar data into a specific number of groups. When working with raw data, there is no need to label input and outputs, so the only human intervention is to define, for example, the number of clusters expected to be obtained from the data. Unsupervised learning is a powerful tool for gaining insight from data, and is often used as a data exploration method to better understand data sets.

Classical Machine Learning

This approach consists of developing models in which the data is intended to be represented by its most important features, which are those that best improve the quality of a model's output. This process requires complex feature engineering in which human experts define what characteristics of the data are best suited to a given problem. The main drawback is that engineering this solution is time-consuming and does not guarantee that the selected features are the optimal combination for a problem.

Neural Networks

Neural networks, also known as artificial neural networks (ANNs), are a subset of machine learning algorithms whose name and structure are inspired by the way that biological neurons signal to one another.

Neural networks are comprised of multiple layers of nodes, containing an input layer, one or more hidden layers, and an output layer. Each node, which mimics a neuron, connects to another and has an associated weight at each connection. The output of any individual node is calculated as a non-linear transformation of the values present on each previous neuron, and then this data is sent to the next layer of the network.

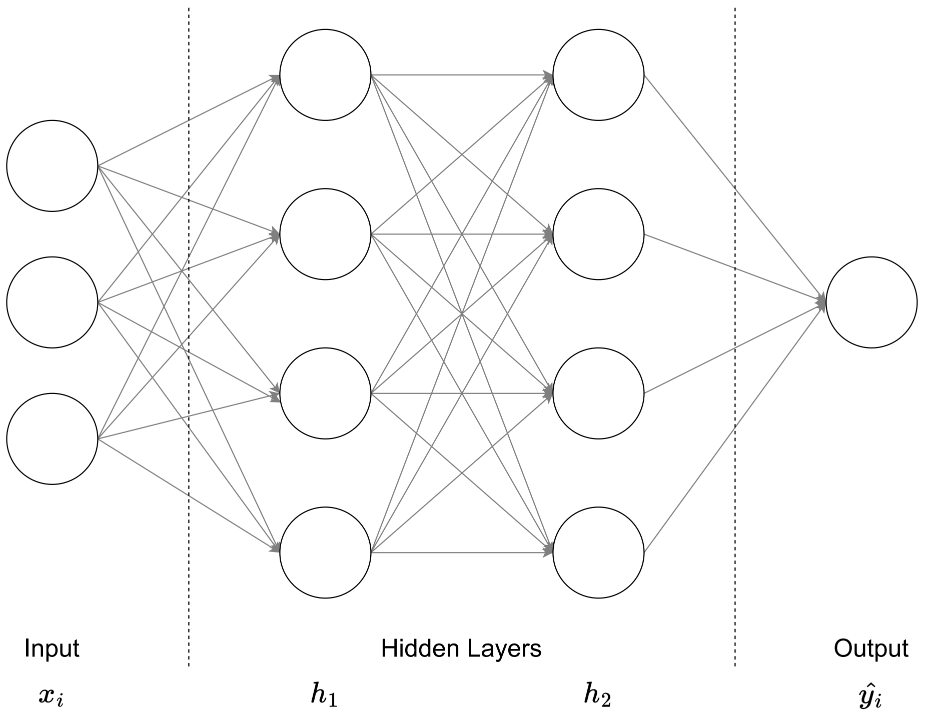


Figure 2.2: Diagram of a Feedforward Neural Network.

Recently, approaches based on Neural Networks have been preferred to Classical Machine Learning mainly because of these features:

- Feature engineering is omitted: Neural Network's layer-based architecture extracts and abstracts key information from raw data to focus on the important parts of the data. In other words, the network highlights and use the relevant data information automatically, so there is no need to extract features manually, which is a time-consuming activity.

- Can approximate any function: In 1989 Hornik et. Al [27] proved the *Universal Approximation Theorem* for ANNs. This implies that neural networks can represent a wide variety of interesting functions when given appropriate weights. This shows the potential that this approach can achieve. Note, however, that this theorem states that such a construction for an ANN is possible, but it does not provide the construction for the weights.

Formally, Artificial Neural Networks are a family of functions that given an input x, and a set of parameters θ learned from the observation of previous data will approximate its output $\hat{y}$ as close as possible to the true output value y.

$$f(x;\theta) = \hat{y} \backsim y$$

The choice of the definition of the function f will determine the class type of the network. The simplest class of ANN is called Feedforward Neural Network and has been successfully used for multiple purposes. However, multiple variations of this architecture have been developed through the years to better suit different problem domains and obtain better results. In particular, we will discuss the common Neural Network approaches used for the music generation methods that will be presented in this document.

Feedfoward Neural Network

The feedforward neural network was the first and simplest type of artificial neural network devised. In this network, the information moves in only one direction forward from the input nodes, through the hidden nodes, to the output nodes. There are no cycles or loops in the network. The output nodes are used to classify the input into a class, thus the softmax will output the corresponding class.

$$\begin{aligned} h^{(0)} &= \sigma(W_\theta^{(0)} x + b_\theta^{(0)}) \\ h^{(i)} &= \sigma(W_\theta^{(i)} h^{(i-1)} + b_\theta^{(i)}) \\ f(x;\theta) &= softmax(\sigma(W_\theta^{(n)} h^{(n)} + b_\theta^{(n)})) \end{aligned}$$

Where σ is a non-linear function: commonly Sigmoid, tanh or ReLu.

Recurrent Neural Networks (RNN)

A recurrent neural network (RNN) [21] is a class of artificial neural networks where recurrent connections between nodes allow them to exhibit temporal dynamic behavior. Derived from feedforward neural networks, RNNs have an internal state mechanism that allows sequences of variable-length inputs to be processed.

Recurrent networks can have additional store states, and the storage can be under direct control by the neural network. Such controlled states are referred to as gated states or gated memory and are part of Long Short-Term Memory Networks (LSTM) [25] and Gated Recurrent Units (GRU) [12].

The function f for this class of neural networks is defined as:

$$h^{(0)} = random\_sample()$$
$$h^{(i)} = tanh(x^{(i)}U + h^{(i-1)}V + b)$$

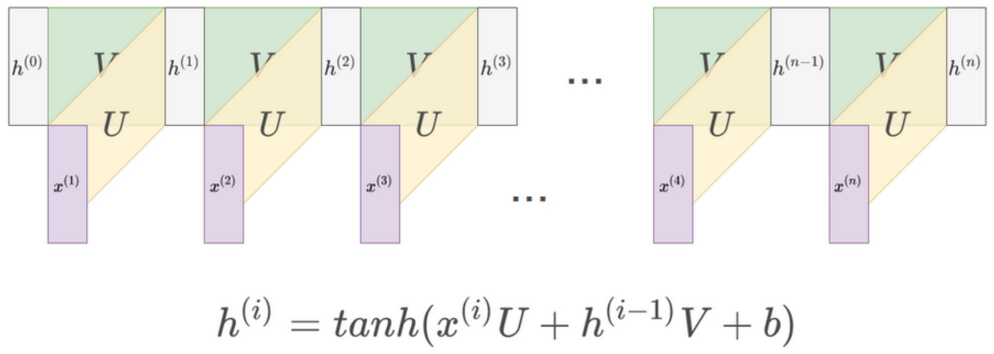


Figure 2.3: Diagram of a Recurrent Neural Network shown as a multiplication of matrices and vectors.

The last vector $h^{(n)}$ in the RNN condenses the information of the whole sequence x. Note that there is no softmax function in the formulation of RNNs. This is due to its design not being necessarily made for classification. The vector $h^{(n)}$ can be used for classifying, but also used to generate a new sequence y through sampling conditioned on h and the output of each step.

Variational Auto Encoder (VAE)

A Variational Autoencoder [33] is an architecture composed of both an encoder and a decoder network that is trained to minimize the reconstruction error between the encoded-decoded data and the initial data. In order to introduce regularisation to the latent space instead of encoding an input as a single point, it encodes it as a distribution over the latent space.

This regularisation induces variance in order to ensure points close to each other will share properties of the original encoded input, making the transition through different groups of points smoother.

The variational encoder architecture is formally described as:

$$\mu_\theta = g_\theta(x)$$

$$\sigma_\theta = h_\theta(x)$$
$$\xi = N(0, I)$$
$$z = \sigma_\theta \xi + \mu_\theta$$
$$\hat{x}_\theta = f_\theta(z)$$

Where the optimization goal is to minimize the following expression:

$$\mathcal{L} = ||x - \hat{x}_\theta||^2 + KL[N(\mu_\theta, \sigma_\theta), N(0, I)]$$

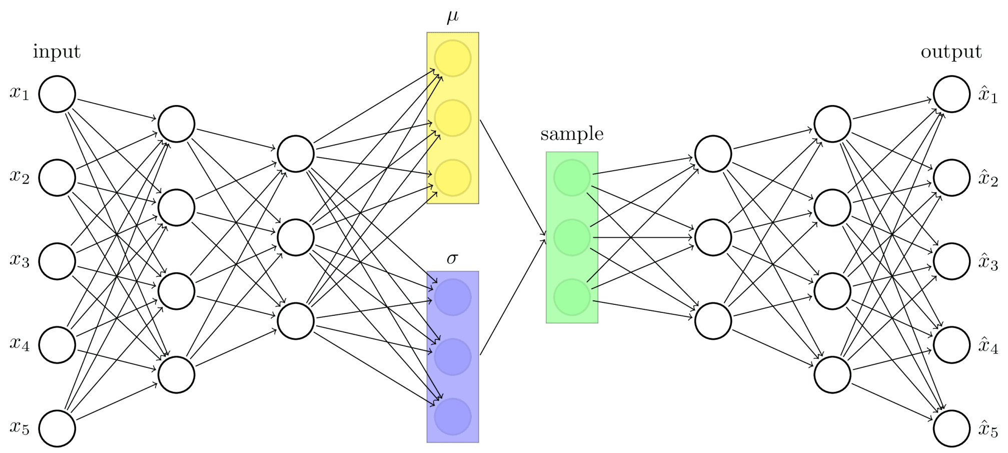


Figure 2.4: Diagram of a Variational Autoencoder Network. The bottleneck encodes the input as the distribution parameters μ and σ. The input is then reconstructed through sampling from the distribution defined by these parameters.

Transformers

A transformer is a Neural Network architecture that adopts a mechanism called self-attention which highlights the relevant context for any position in an input sequence when being processed.

Like recurrent neural networks (RNNs), transformers are designed to process sequential input data. However, transformers process the entire input all at once to allow higher parallelization than RNNs and therefore reduce training times.

Transformers have been documented to outperform several approaches when exposed to massive amounts of data [19].

2.1.3. Automated Music Generation

Composing music automatically with computers is challenging since it requires mimicking musical rules, which is complex to formally describe in the context of a musical language as it is very ambiguous depending on the context or style of a song, and as it is constantly evolving.

Several computational methods have been proposed to address this human creative process, which can be divided into three main approaches: Rule-based systems, Classical Machine Learning, and Deep Learning.

Rule-based systems

This approach consists of defining hand-crafted rules to introduce knowledge into generative systems by finding musical patterns or regularities. The major drawback of this approach is that it is time-consuming and not extensible to new data due to the difficulty in the maintenance of a large number of rules.

An interesting case of this approach is *Generative Grammars* [20, 53, 9]. They describe different music styles through knowledge-based grammar which creates chord progressions or melodies on the basis of what is observed in musical corpora and rules of a particular style.

Classical Machine Learning-based systems

This approach utilizes algorithms to statistically analyze a dataset to apply a pre-defined algorithm over relevant features engineered by an expert. Some of these approaches include:

- Hidden Markov Models [1, 47, 42]: In this approach, the music is represented as a sequence of states where the transition probabilities between a note to another are calculated by frequency in the dataset.
- Genetic Algorithms [26, 37, 44]: In this approach, a probability distribution over relevant features is set to randomly sample and create music. This sampling is further refined by setting constraints over pre-defined musical rules.

Deep Learning-based systems

Deep Learning-based approaches [31, 50, 17, 39] are currently the best performers in music generation tasks. The main advantage over previous approaches comes from the fact that neural networks capture relevant information through time and infer the inner rules of the sequences, which means that expert knowledge is not required to encode several rules by hand as before. Under this approach, the most commonly used models are recurrent neural networks, encoder-decoder architectures, and lately transformer-based models.

This approach has been mostly stated as a Supervised Learning task consisting of a classification problem, where each note represents a class. The goal is that given a sequence of musical tokens a model could predict the next note to belong to the correct class. This prediction is done iteratively through *Auto-regression*, where each note prediction will condition the next prediction step recursively.

2.2. Music Generation

2.2.1. Preliminary Concepts

In this section we present some musical concepts required to follow the document:

- Note: Representation of pitch and duration of a sound in musical notation.
- Pitch: Highness or lowness of a sound in terms of frequency. Often discretized as part of pitch classes. For example: C, D, E, F, etc.
- Rhythm: The placement of sounds in time. This includes both the onset of a note but also the duration of the note.
- Melody: Sequence of notes for a single instrument or vocal, with at most one note playing at the same time.
- Harmony: Group of notes that sound at the same time.
- Monophonic music: Sequence of notes exclusively formed by melodies for one instrument/voice.
- Polyphonic music: Sequence of notes for one or many instruments where more than one note can be played at the same time.
- Musical Key: A main group of pitches, or notes, that form the harmonic foundation of a piece of music.
- Beat: Basic unit of time that will mark regularly repeating events.
- Measure: Segment of time corresponding to a specific number of beats.
- Time signature: Specifies how many beats are contained in each measure (bar)
- Dynamics: Variation in loudness between notes. It gives expressiveness to music performances.

2.2.2. Data Representation

Symbolic music vs Audio Signal

Music Generation approaches often treat musical data as symbolic representations. This means that data is encoded as a sequence of events regarding attributes of notes such as pitches, onsets, and dynamics instead of raw audio signals which are harder to process.

Symbolic music has multiple advantages compared to audio representations:

- Representations are closer to an actual musical score than audio. This is due to the use of metadata that explicit attributes such as tempo, pitch, rhythm, onsets, velocity, time signature, etc.

- Extracting information and applying common operations such as adding/removing notes, key changes, changes in tempo, etc is easier.
- Symbolic data uses less disk space than audio signals due to encoding just the events of interest as opposed to sampling each millisecond in audio representations.
- Transformations of symbolic data to another source of musical data are direct: symbolic-to-score and even symbolic-to-audio.

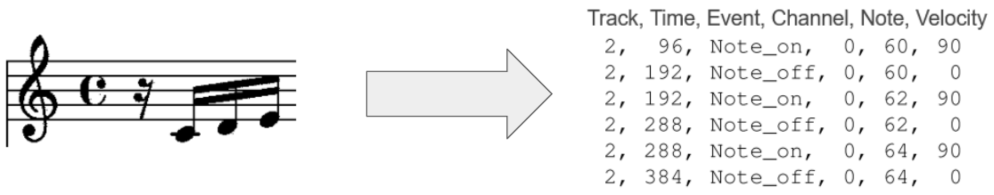
However, symbolic music also has disadvantages when compared to audio data:

- The expressiveness of symbolic data is very limited due to an oversimplification of audio signals into discretized events.
- The conversion from symbolic-to-audio formats makes the music feel unrealistic when is listened to due to the utilization of digitalized instruments.

Data Format

Symbolic music is often formatted as either MIDI or MusicXML files.

Musical Instrument Digital Interface (MIDI) is a protocol originally designed to communicate computers with electronic musical instruments. MIDI store event messages about musical notes: *Track* for specifying the current instrument, clock signals to set the *Tempo* of specific events in absolute time, *Channel* to define where the output of an event will be played, parameters such as Velocity to define the volume of a note, among several other metadata. An example of MIDI encoding is shown in Figure 2.5.

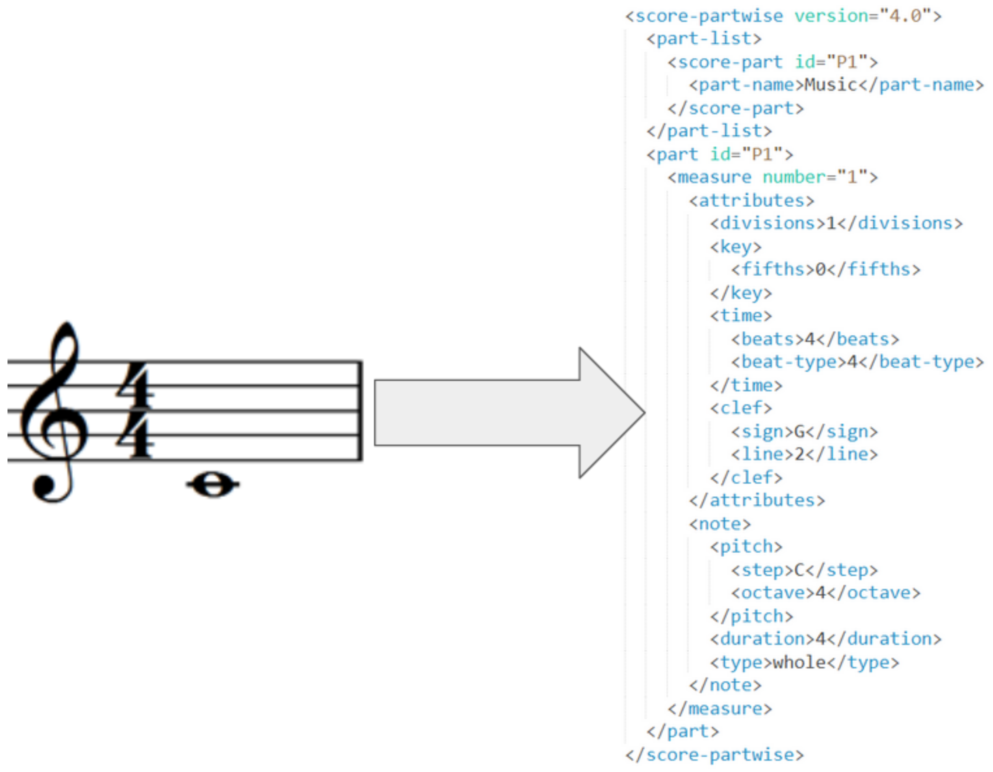


```
Track, Time, Event, Channel, Note, Velocity
 2,   96, Note_on,  0, 60, 90
 2, 192, Note_off, 0, 60,  0
 2, 192, Note_on,  0, 62, 90
 2, 288, Note_off, 0, 62,  0
 2, 288, Note_on,  0, 64, 90
 2, 384, Note_off, 0, 64,  0
```

Figure 2.5: MIDI representation of music score.

MusicXML, on the other hand, is a digital sheet music interchange and distribution format based on XML syntax. Its goal is to create a universal format for common Western music notation. In contrast to MIDI, it is more musical-oriented than event-oriented. It stores musical information based on relative positions (musical scores) instead of absolute time. It also contains musical information as *Key* and the names of the notes contained in a file. An example of MusicXML format is shown in Figure 2.6.

Temporal Discretization



```xml
<score-partwise version="4.0">
  <part-list>
    <score-part id="P1">
      <part-name>Music</part-name>
    </score-part>
  </part-list>
  <part id="P1">
    <measure number="1">
      <attributes>
        <divisions>1</divisions>
        <key>
          <fifths>0</fifths>
        </key>
        <time>
          <beats>4</beats>
          <beat-type>4</beat-type>
        </time>
        <clef>
          <sign>G</sign>
          <line>2</line>
        </clef>
      </attributes>
      <note>
        <pitch>
          <step>C</step>
          <octave>4</octave>
        </pitch>
        <duration>4</duration>
        <type>whole</type>
      </note>
    </measure>
  </part>
</score-partwise>
```

Figure 2.6: MusicXML representation of music score.

Since time works over a continuous space, it is necessary to initially decide how to discretize each note event over time to match its musical representation and thus determine if it is a quarter note, eighth note, etc.

The choice of how to discretize temporal data first considers how fast the notes are played measured in beats per minute (bpm), which will define the tempo of the piece. Once the tempo is set, the minimal step of a time grid will consider an arbitrary but fixed number of subdivisions for each beat.

Common choices seen in automated music generation consider 2, 4, 6, and 8 subdivisions for each beat [23]. In musical terms, this translates as the minimal note length for each discretization as an eighth note (quaver), sixteenth note (semi-quaver), sixteenth triple note, and thirty-second note (demisemiquaver), respectively. The time discretization is also often referred to over 4/4 musical measures, in which case they will contain 8, 16, 24, and 32 subdivisions per measure, respectively.

Temporal Scope

When describing the temporal grid for our data, there are two main approaches to considering the temporal scope of the musical piece. Figure 2.7 shows an example of both approaches listed below.

- Time Based Discretization: The representation of the musical piece is done through an equally spaced grid of time steps, usually set to the shortest note duration. Each time step corresponds to a specific temporal moment where an event could or not be present. Since may be time steps without notes it may induce sparsity to the representation.

- Note Based Discretization: In this approach, there is no fixed time step. The granularity of the representation is defined depending on the number of notes and how long they last. Note that, by considering one note as a single processing step, the number of processing steps and its memory usage is greatly reduced.

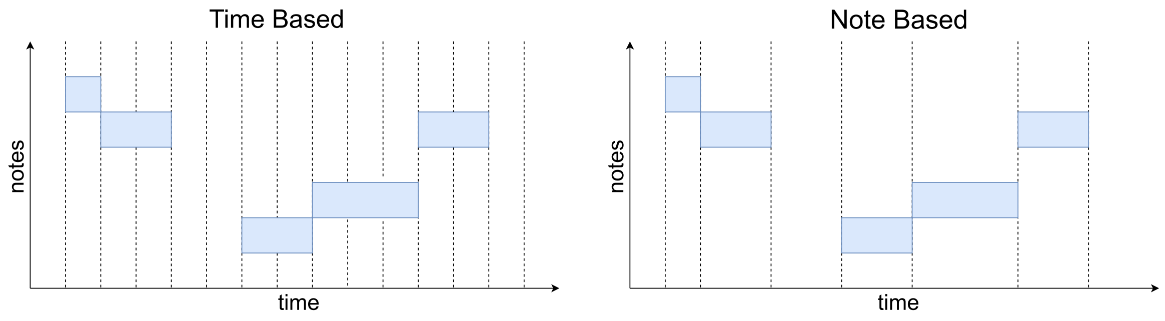


Figure 2.7: Temporal scope comparison for a piano roll-like representation. Dotted lines represent the granularity of each representation.

2.2.3. Vector Encoding

Piano Roll

Piano Roll is a common vector representation for music. It encodes music as a sequence of time steps where each time is a multi-hot vector that indicates whether a note is being played or not. This representation has a limitation for distinguishing the end of a note and the start of another note. Formally,

$$x \in \{0,1\}^{I \times T \times N}$$

where $\{0,1\}^N$ represents a multi-hot vector that indexes notes currently being played with 1 and 0 otherwise, N represents notes' vocabulary size, I the number of instruments, and T the number of time-steps.

Note Sequence

Note Sequence is a particular case of piano roll that is often used for monophonic setups. Since the vector in the note's dimension will be a one-hot vector, its representation can be compressed as a single integer value indexing the position where the vector is one. By doing this the data is compactly stored. The encodings of silences and holds are added as two extra tokens. Then, the values in this encoding can be seen as:

$$x \in \{0, 129\}^{I \times T}$$

where {0,129} represents the token space (0 to 127 represent pitches, 128 hold, 129 silence), I the instrument dimension, and T the time-step dimension. An example of this encoding is shown in Figure 2.8 (b).

Factorization

Chen et. al [11] introduced the concept of factorization over Note Sequence encoding. It consists of representing the music in two different dimensions: pitch and rhythm. An example of this encoding is shown in Figure 2.8(c).

REMI

REMI [32] is an encoding scheme inspired by MIDI that follows the Note-Based temporal scope strategy. It encodes each note as a tuple for:

- Tempo: Beats per Minute (BPM) for the measure where the note is present.
- Bar-Start: Boolean value that represents whether the note is the start of a music measure or not.
- Relative Position: Integer value indexing the position of the note in the subdivision of a measure.
- Pitch: Integer value ranging between 0-127 that represents which note is being played.
- Velocity: Integer value representing the loudness of the note being played.
- Duration: Integer representing how many minimal time-steps does the note last.

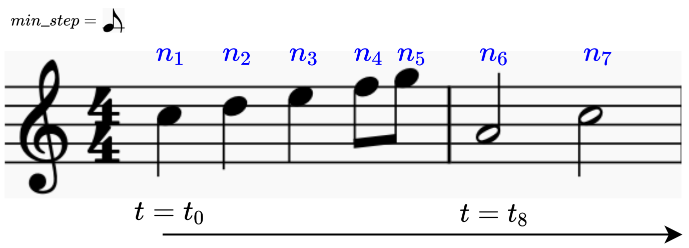
Formally defined as:

$$x \in \{(tempo, beat, pos, px, vel, dur)\}^{I \times M \times N(I,M)}$$

where I represents the instrument's cardinality, M the number of measures in the song, $N(I, M)$ represents the number of notes, which depends on the instrument index and the current musical measure index.

An example of this encoding is shown in Figure 2.8 (d).

(a)



(b) $x = [C_4, \_, D_4, \_, E_4, \_, F_4, G_4, A_3, \_, \_, \_, C_4, \_, \_, \_]$

(c)
$$\begin{aligned} x &= (x_{pitch}, x_{rhythm}) \\ x_{pitch} &= [C_4, D_4, E_4, F_4, G_4, A_3, C_4] \\ x_{rhythm} &= [1,0,1,0,1,0,1,1,1,0,0,0,1,0,0,0] \end{aligned}$$

(d) $x = [\,[n_1, n_2, n_3, n_4, n_5], [n_6, n_7]\,]$

n	Tempo	Bar Start	Position	Pitch	Velocity	Duration
n_1	120	1	0 | 8	C_4	90	♩
n_2	120	0	2 | 8	D_4	90	♩
n_3	120	0	4 | 8	E_4	90	♩
n_4	120	0	6 | 8	F_4	90	♪
n_5	120	0	7 | 8	G_4	90	♪
n_6	120	1	0 | 8	A_3	90	𝅗𝅥
n_7	120	0	4 | 8	C_4	90	𝅗𝅥

Figure 2.8: Vector encoding for each method discussed in this work. (a) Musical input data. (b) The encoding used in ARNN [22] and InpaintNet [40]. The token ”_represents the state of holding a note. (c) The factorized encoding used in Music SketchNet[11]. (d) The REMI-based encoding used in VLI [10].

2.3. Music Inpainting Task

Task Formalization

In most Machine Learning tasks, a formal definition is usually introduced in order to better understand a problem. This process exhibits the input and output variables of the task of interest. In our context, we introduce the formal definition of *Musical Score Inpainting* as stated in [40].

Given a past musical context C^p, a future musical context C^f, the modeling task is to generate an inpainted sequence C^m, which can connect C^p and C^f in a musically meaningful manner.

Note that the phrase "musically meaningful manner" does not represent a formal definition itself since musicality is open to interpretation and subject to change depending on how it is measured. Although several musical metrics have been proposed for this purpose, designing metrics that numerically show the quality of a musical piece is still an open problem. One of the most limiting factors is that given a musical piece, multiple variations of a piece could be considered as musically correct as the others.

2.4. Related Work

2.4.1. Similar Benchmarks

In the latest years, several research communities have highlighted the need for stronger standards of replicability. Without it, research communities might have a false sense of what is the state of the art in different tasks.

For instance, Dacrema et al. [15] benchmarked several deep learning methods against traditional approaches for the task of collaborative filtering recommendation, finding that there was not a clear and consistent improvement of deep learning techniques against classical machine learning methods. Furthermore, Arango et al. [2] found that the task of hate speech detection had made less progress than reported in the literature after benchmarking several methods under the same datasets with equal training and testing conditions. These examples support the need for strong, fair, and replicable benchmarks to claim progress in a task. These works inspire us to develop MUSIB for music inpainting.

2.4.2. Music Inpainting Models

This section describes the four models evaluated in our work. We emphasize that the decision to implement these models is based on the feasibility of replicating their code in a single environment. We mention, however, all inpainting models published to date, to the best

Model	Architecture	Year	Music Type	Base Framework
CocoNet	CNN	2017	Polyphony	TensorFlow
DeepBach	RNN	2017	Polyphony	Pytorch
InpaintNet	VAE + RNN	2019	Monophony	Pytorch
SketchNet	VAE + RNN	2020	Monophony	Pytorch
AnticipationRNN	RNN	2020	Monophony	Pytorch
VLI	XL-Net	2021	Polyphony	Pytorch
DiffModel	Diffusion models	2021	Monophony	Flax
MusIAC	Transformer	2022	Polyphony	Pytorch

Table 2.1: Existing models for music inpainting

of our knowledge, in Table 2.1 for reference. We highlight the general strategies/architecture utilized for each model alongside with their source implementation framework.

Anticipation RNN

Hadjeres et Al [22] proposed Anticipation RNN. This model represents input as a Note Sequence. The main idea consists in capturing two temporal sequences with two RNNs: one encodes unary-constraints embeddings while the other auto-regressively generates tokens conditioned on these constraints. Unary constraints allow the model to include pre-defined notes at arbitrary timesteps. Constraining C_p and C_f before the generation process recreates the musical score inpainting setup. See Figure 2.11 for reference.

Music InpaintNet

Proposed by Pati et Al [40], it uses VAEs [33] to encode isolated monophonic measures into a latent space vector (called MeasureVAE). The model uses this normalized space to operate the past z_p and future z_f latent context with a called LatentRNN. This last representation is then hierarchically decoded into beats and ticks for reconstructing the inpainted sequence. See Figure 2.9 for reference.

Music SketchNet

Proposed by Chen et Al. [11], it utilizes a similar strategy to [40] by applying VAEs for encoding music measures. However, prior to encoding into latent space, it modifies the input data representation to naturally separate pitch and rhythm into two separate dimensions. Their VAE then encodes these two separate channels and decodes them hierarchically. A final training phase is done to condition the final output with users' input over general constraints on pitch and rhythms. See Figure 2.10 for reference.

Variable Length Piano Infilling (VLI)

Proposed by [10], this model is designed to create polyphonic music inpainting based on XLNet [51]. Since monophonic music is a particular case of polyphonic music, it is possible to evaluate this model over monophonic music. The method encodes each note event as a word token and feeds it to a pre-trained language model over Wikipedia. They incorporate a musically specialized positional encoding called relative bar encoding to keep track of the relative position of each note within its context.

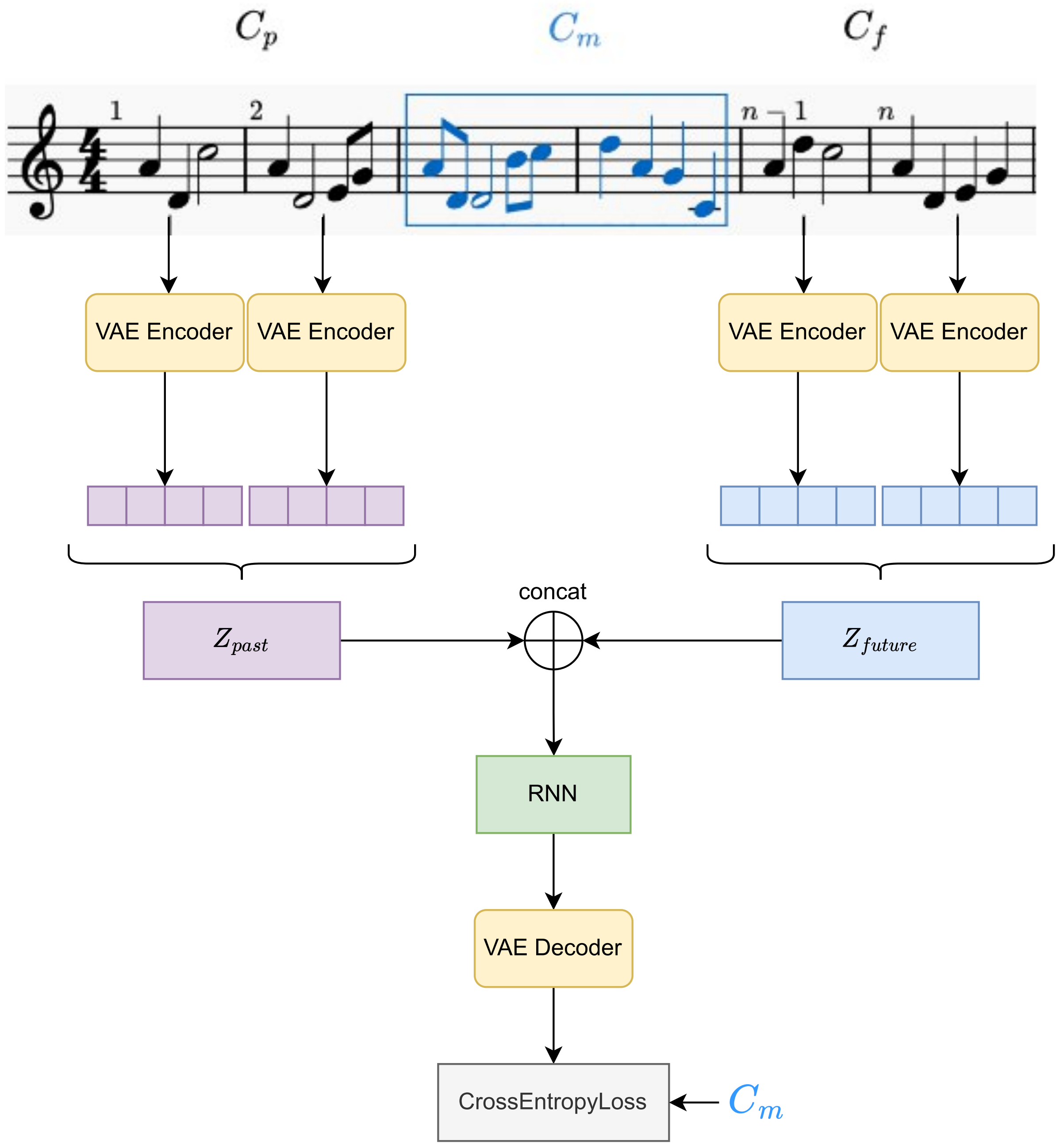


Figure 2.9: Diagram of the Music Inpaintnet architecture.

2.4.3. Metrics

Negative Log-Likelihood (NLL) has been widely used both for training and evaluating models on music inpainting. This value represents a statistical distance between two given distributions, where the lower the value, the closer these distributions are. The main drawback of this function is that the value itself does not represent any musical concept nor captures the domain's semantics and thus cannot be analyzed intuitively.

Two simple yet intuitive metrics were proposed by Chen et Al [11] called *pAcc* and *rAcc*. They represent whether the model generates the correct pitch/rhythm token in the correct time-step position. However, pAcc has limitations in distinguishing, for example, whether a note is misplaced or is different from the expected pitch, while rAcc can change its values depending on chosen time step discretization.

Pitch Class Histogram Entropy and *Grooving Pattern Similarity* [49] were used in [10] to compare the similarity of musical attributes between measures in the infilled part with those present in the context. These metrics were used assuming that to generate fluent music, the metrics calculated on C_m should be close to the ones calculated for C_p and C_f. They state that the lower the differences between the middle part metrics and its context, the better the model is. However, this is not necessarily true since values too close to zero indicate that the model is just repeating its context instead of articulating past and future musical ideas.

Finally, some other metrics capture more general musical attributes that are not directly comparable between two sequences but can be useful in understanding the output of a model. For example, *Number of Silence* [8] calculates the percentage of empty time steps in a sequence to check if a model is generating too much silence or not.

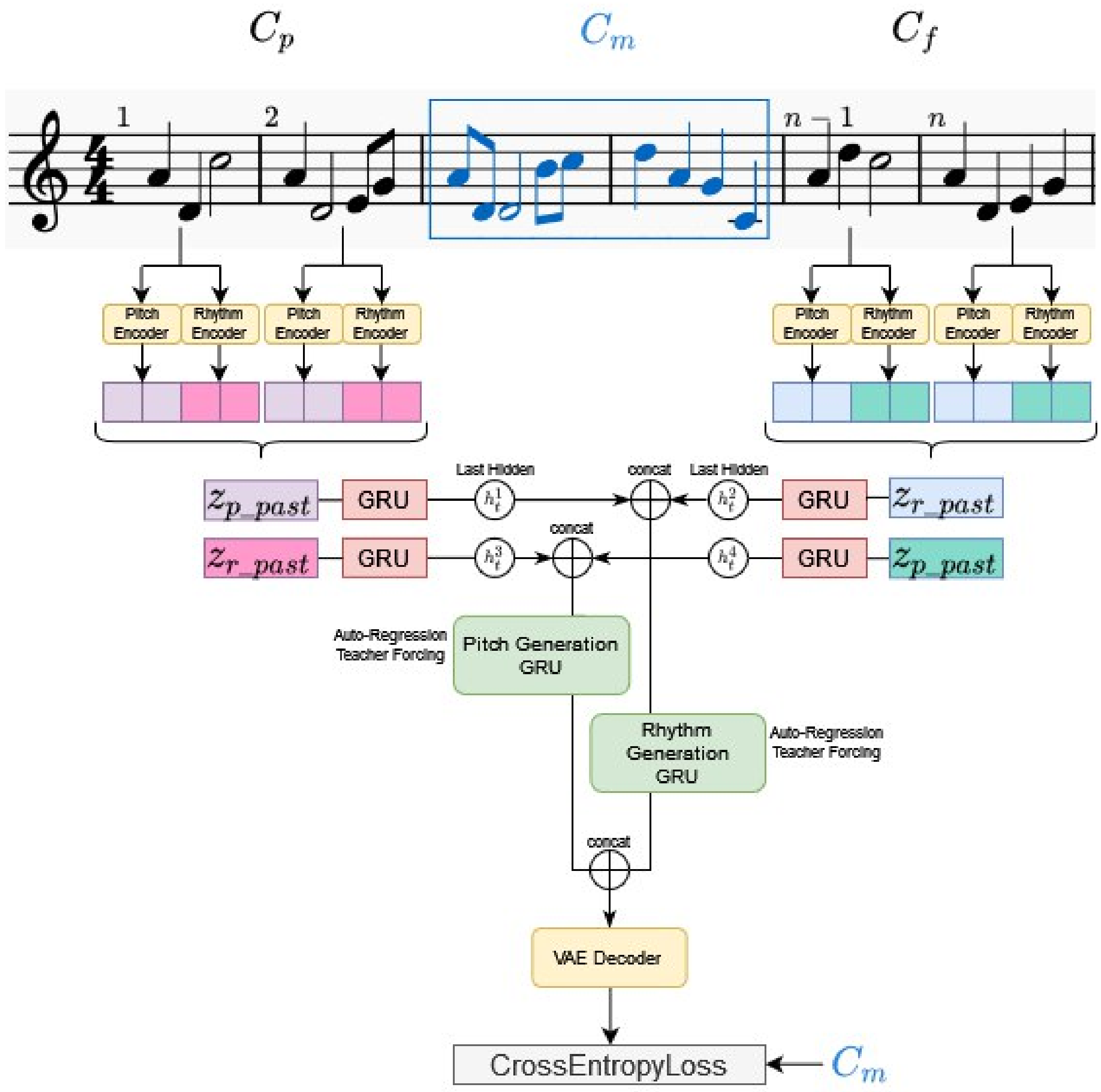


Figure 2.10: Diagram of the Music SketchNet architecture.

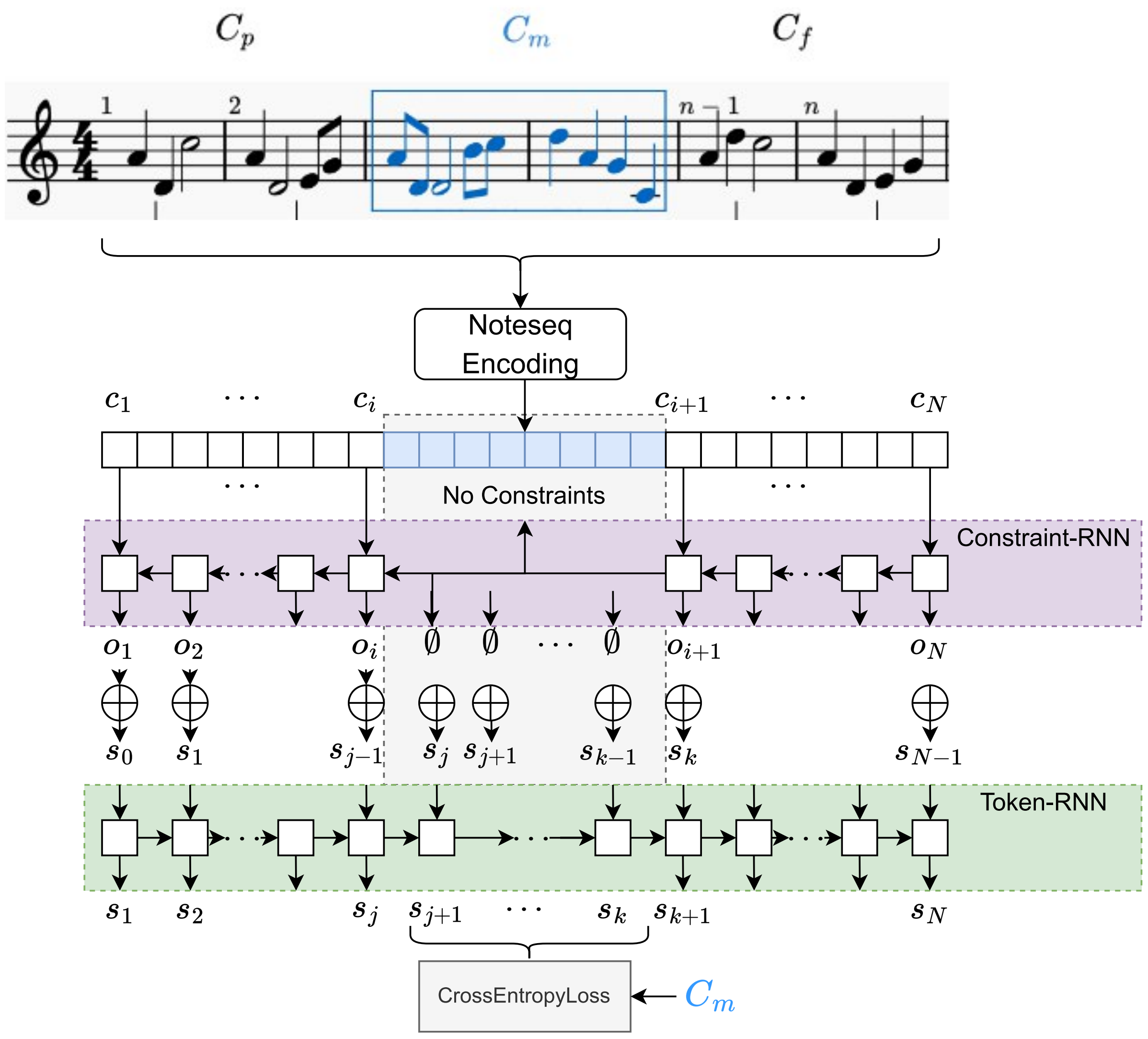


Figure 2.11: Diagram of the Anticipation RNN.

Capítulo 3

MUSIB: Music Inpainting Benchmark

This chapter aims to present and formally describe the algorithms and the math underlying our theoretical proposal for the evaluation methodology in MUSIB. For a detailed description of the implementation of MUSIB, see Chapter 4.

3.1. Motivation

Most evaluations of musical inpainting models do not share data representations, metrics, or datasets. This hinders finding out what progress has been made in the task, what current limitations need to be addressed, and which ideas have proven to be successful and could be further exploited.

Our proposed benchmark considers four models out of the eight listed on table 2.1. We selected the most recently published models (by September of 2021), limited to four models based on time restrictions. This decision was suitable since all these models shared the same framework.

To illustrate the difference in evaluation among these four models we show in Table 3.1 the evaluation setups for Anticipation-RNN (ARNN), Music Inpaintnet, Music SketchNet, and Variable Length Piano Infilling (VLI).

Anticipation-RNN validated its method by comparing its model against a modified version of this same proposed architecture over JSB Chorales. Then, InpaintNet verified that their models' results accomplished better validation loss than Anticipation-RNN over Irish-FolkSong, however, there were no comparisons over JSB Chorales, the original dataset for ARNN. Later, SketchNet evaluated their model against InpaintNet using NLL, pAcc, and rAcc over IrishFolkSong, introducing two new metrics which made the comparison harder with previous approaches outside InpaintNet. Finally, VLI tested their model against two baselines also firstly introduced in their paper which were based on language models: ILM, and FELIX over AILabs1k7.

On top of that, the results for each metric are dependent on the data representation chosen

Model	Representation	Dataset	Metrics
VLI	REMI-16	AILabs1k7	H1, H4, GS
SketchNet	NoteSeq-24	IrishFolk	NLL, pAcc, rAcc
InpaintNet	NoteSeq-24	IrishFolk	NLL
A-RNN	NoteSeq-16	JSBChorales	Accuracy, JS Div

Table 3.1: Original evaluation conditions for music inpainting models, showing how difficult is to compare them.

on each model. This implies that, for instance, the results for the NLL metric shown in the SketchNet [11] publication can not be compared since they may have different magnitudes than the one presented in InpaintNet [40]. This phenomenon is empirically shown in [29] for the NLL metric over the same data when represented as quarters, eighths, and sixteenths time-step resolution per measure. In this research, we formally show that in a similar way, Pitch Accuracy and Rhythm Accuracy [11] also change their value when changing the data representation, and a way of standardizing the metric values even when data representations change.

MUSIB: the music inpainting benchmark is proposed as a new evaluation methodology to standardize conditions to directly evaluate different inpainting models. It comprises the study of four models, over two datasets, with seven different metrics.

3.2. Datasets

We selected JSB Chorales and IrishFolkSong datasets to implement our evaluation. We prioritized these datasets since they have been used to train several musical inpainting models. They also represent different musical styles which provide meaningful differences in their musical content. Finally, they have an important difference in size, making generalizing results a challenging task.

JSB Chorales [13] dataset contains 408 samples of 4-voices chorales pieces. Each sample corresponds to the harmonization of a hymn. Its source format is MXL; however, we transform the data to MIDI format to have a single pipeline to process all data.

IrishFolkSong [45] dataset contains 45,849 pieces of monophonic folk tunes in midi format.

An overview of the pitch distribution for both dataset is shown in Figure 3.1 and 3.2. It can be observed from this that: 1) Both datasets do not share support sets. 2) The pitch distribution for both datasets tend to concentrate most values in the middle of the support set. 3) The pitch classes are unbalanced in both datasets.

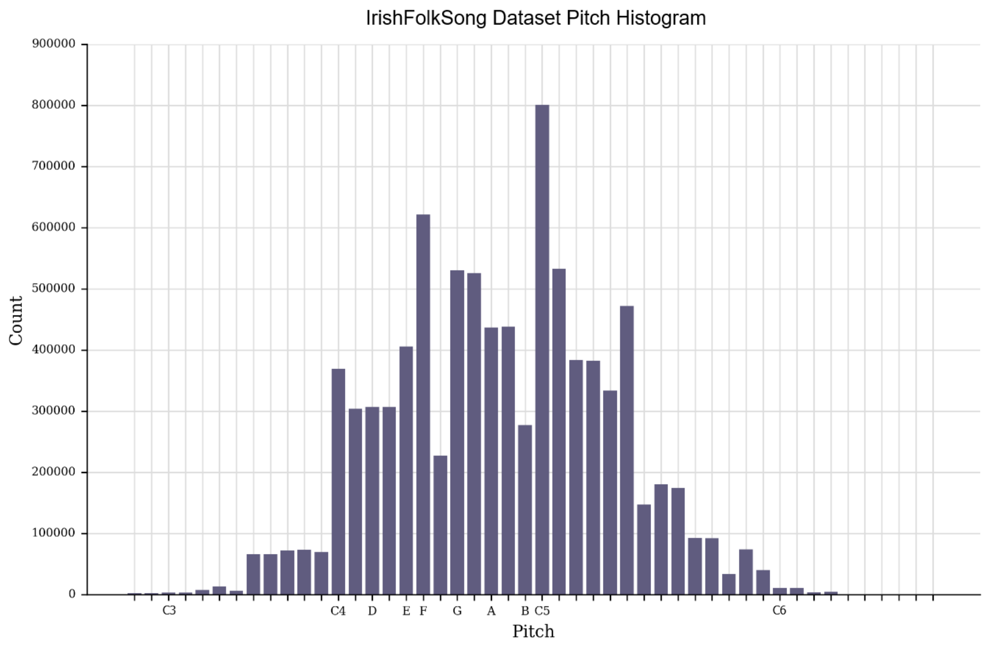


Figure 3.1: Pitch distribution for the IrishFolkSong dataset.

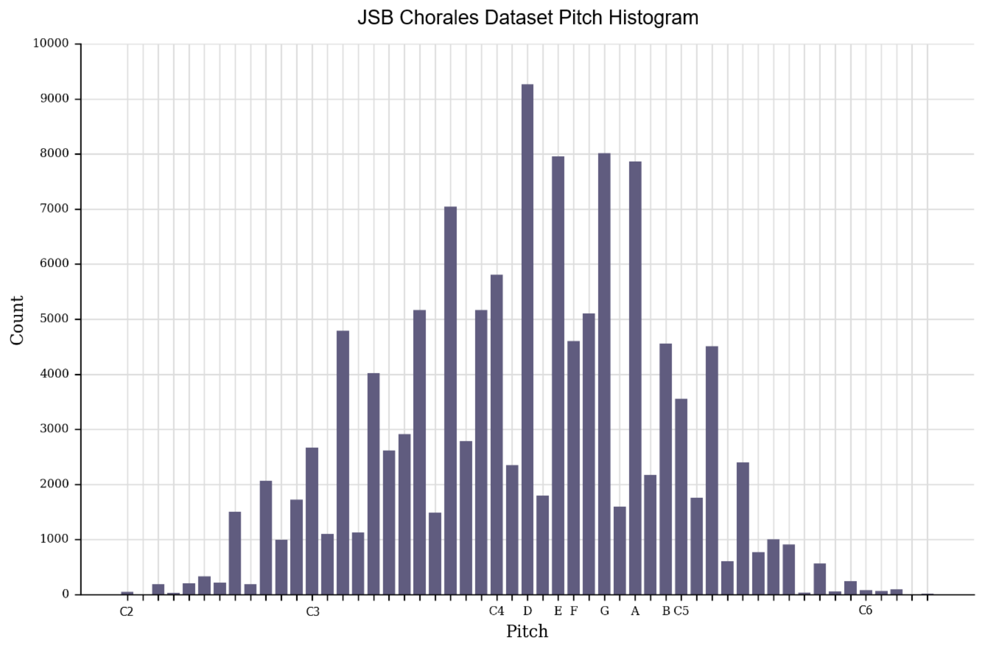


Figure 3.2: Pitch distribution for the JSB Chorales dataset.

For both datasets, we filtered invalid files (i.e., no instruments or zero-length), repeated files (files with the same hash), and files shorter than 16-measures long. We also filtered the data in IrishFolkSong to only have pieces on 4/4 time signatures. This last step is done to reproduce conditions described in papers that originally used this dataset.

The final JSB Chorales dataset contains 171 songs, totaling 13,304 measures that were grouped into 2,360 contexts. IrishFolkSong dataset ended up with 17,538 songs, 605,164 measures, and 324,556 grouped contexts.

3.3. Evaluation

We classify MUSIB metrics into two groups: *Note Metrics* and *Divergence Metrics*.

3.3.1. Note Metrics

Note metrics directly compare note attributes in predicted data vs true data, one note at a time. We argue that for measuring the quality of notes predicted, we need to compare at least three dimensions: Position, Pitch, and Rhythm. Position indicates the time when a note starts, Pitch is the pitch of a note (C_4, G_2, $D_3^{\#}$, ...), and Rhythm is the combination of duration and position of a note. We represent all notes in Y_{true} and Y_{pred} as triplets for these three dimensions when evaluating note metrics (see Figure 3.5 for reference).

Position Score

Position Score is a metric proposed in this work that measures the similarity of two musical sequences in terms of the position of their notes.

We argue that to correctly measure notes' position similarity, a metric needs to be able to meet the following requirement:

1. Be equipped with a strategy to align the notes' positions within gold and predicted sequences independently of the order in which they appear.
2. Handle sequences with potentially different number of notes.
3. Reward sequences that share the same positions for their notes.
4. Penalize sequences that do not share the same positions for their notes.
5. Penalize generated sequences with different number of notes than expected.

Delving deeper into requirement (1), we should point out that the i-th note of the gold sequence may be present as the j-th note of the predicted sequence. Therefore, to check that a given position has been correctly predicted, it is important that our metric can align the positions between the two sequences to perform a proper evaluation.

Taking the above into account, we construct our metric as an F1 score calculated from gold and predicted note's positions whose internal variables (i.e., True Positives, False Positives, False Negatives) are computed as follows:

- True Positives (TP): A note's position is present in both sequences.
- False Positives (FP): A note's position is present in the generated sequence when it was not present in the gold sequence.
- False Negatives (FN): A note's position is missing in the generated sequence when it was present in the gold sequence.

Note that True Negatives are not part of the F1 score function and thus its definition is not stated here.

Next, we discuss how each of the the aforementioned requirements are satisfied by our F1 metric:

1. By defining the process of alignment based on checking the presence of a note within a given sequence we resolve the ordering problem between non-matching sequences.
2. Building the internal variables of the F1 Score based on the alignment of positions allows us to compare sequences with different number of notes since the match of positions for the i-th and j-th note may occur at arbitrary indexes in arbitrary long sequences.
3. Both values *precision* and *recall* will increase as the number of True Positives increases, increasing F1-Score performance, and thus rewarding sequences that share positions.
4. Both values *precision* and *recall* will decay as FP and FN increase. Note that metric functions such as *Accuracy* would not be able to penalize missing notes (FN). Additionally, there is no difference in cost for different types of mis-classifications in this task. Either adding or removing notes to the generated sequence with respect to the gold sequence would have the same impact in musicality. Due to this, both the recall and precision do not need particular weights when being evaluated, discarding alternatives such as F_β functions.
5. If the generated sequence contains more notes than the true sequence, the number of false positives will increase. Similarly, if the number of notes is smaller than the true sequence, the number of false negatives will increase. Both cases imply that F1-Score will decrease in performance, either by a worse Recall or Precision. This implies that *Position Score* penalizes sequences with a different number of notes than expected.

We formally define *Position Score* as:

$$pos_{F1}(y, \hat{y}) = F_{1_{(tp,fp,fn)}}(y, \hat{y})$$

$$tp(y, \hat{y}) = \sum_{i=0}^{n} \mathbb{1}_{y_i \in \hat{y}}$$

$$fp(y, \hat{y}) = \sum_{j=0}^{m} \mathbb{1}_{\hat{y}_i \notin y}$$

$$fn(y, \hat{y}) = \sum_{i=0}^{n} \mathbb{1}_{y_j \notin \hat{y}}$$

where y is the list of positions in the gold sequence, $\hat{y}$ is the list of positions in the predicted sequence, tp is the function that computes true positives, fp is the function that computes false positives, fn is the function that computes false negatives, n is the number of notes present in the gold sequence, m is the number of notes present in the predicted sequence, and $F_{1_{(tp,fp,fn)}}(y, \hat{y})$ is the f1 score computed from the result of the fp, tp, fn functions applied over y and $\hat{y}$.

Pitch Accuracy

Firstly defined by Chen et Al. [11], is the percent of pitches correctly predicted over the total of pitches in a sequence. The metric is thought as a comparison of two musical sequences, where if a pitch is present at a given time index, the metric function checks the equality of this pitch in the same index for the other sequence.

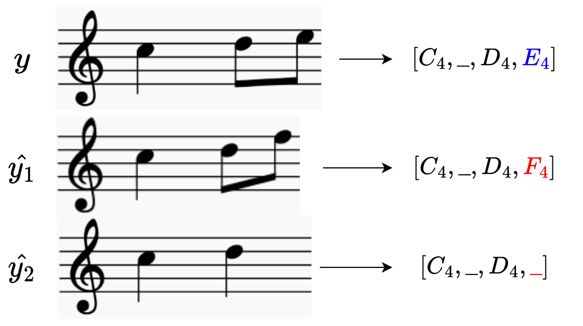
For our evaluation procedure we slightly modified the application of the metric. We argue that the result of this metric may be misleading in explaining two fundamentally different musical phenomenons. In particular, with this metric as is, a mismatch of pitch might represent either:

1. The first note and the note to be compared (both at time index i) do not share the same pitch (e.g. one note is F3 and the other one is D4), or
2. There is a note at time index i for the first sequence, but there is no matching note at the same time index in the sequence to compare because there is a silence or hold token.

The second case is a case of misplacing of notes instead of an error of pitches. In Figure 3.3 (a) we show an example where the two different phenomenon lead to the same result.

To address this ambiguity, we restrict the instances to which this metric is applied to only pairs of notes with matching positions within their corresponding sequences; otherwise, the comparison is omitted. The intuition is that notes that share position but not pitch will be measured by *Pitch Accuracy*, while the misplaced notes will be measured by *Positional Score*. We argue that this modification gives a clearer understanding of the output of the *Pitch Accuracy* metric, addressing potential concerns when comparing against *Positional Score*. We show an example in Figure 3.3 (b).

Formally, *Pitch Accuracy* is defined as:



(a) $\Rightarrow$ $pAcc(y, \hat{y}_1) = 2/3$

$pAcc(y, \hat{y}_2) = 2/3$

(b) $\Rightarrow$ $pAcc(y, \hat{y}_1) = 2/3 \quad pos_{f1}(y, \hat{y}_1) = 3/3$

$pAcc(y, \hat{y}_2) = 2/2 \quad pos_{f1}(y, \hat{y}_2) = 2/3$

Figure 3.3: Example of evaluation between an expected sequence y and two generated sequences $\hat{y}_1$ and $\hat{y}_2$. (a) Shows the results when applying *Pitch Accuracy* over time indexes as proposed by Chen et. al [11]. (b) Shows the results when applying our proposed modification to *Pitch Accuracy* in conjuction with our proposed *Position Score*.

$$pAcc(y, \hat{y}) = \frac{1}{N} \sum_{i=0}^{N} \sum_{j=0}^{N} \mathbb{1}_{pitch(y_i)=pitch(\hat{y}_j)} \mathbb{1}_{pos(y_i)=pos(\hat{y}_j)}$$

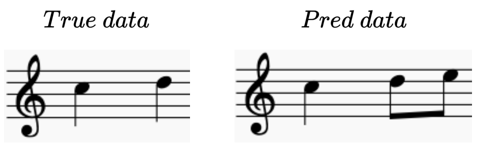
where N is the number of notes that share positions in both sequences, y is the gold sequence of notes, $\hat{y}$ is the predicted sequence of notes, $pitch(y_i)$ retrieves the pitch value of the note at index i, $pos(y_i)$ retrieves the position of the note at index i. Note that the metric definition sums the number of notes that match both pitch and position and then normalizes it by the number of notes present in both sequences.

Rhythm Accuracy

Firstly defined by Chen et Al. [11], is the percent of notes' duration correctly predicted over the total of notes.

For our evaluation procedure we slightly modified the application of the metric. We argue that this metric as is does not correctly measure the performance of the models due to differences in the results when it is applied to the same data with different notes' resolutions. We show an example in Figure 3.4.

Note that the issue comes from the fact that the duration of a note is stored as multiple tokens, one per time-step. Changing the resolution of the sequence affects the representation of hold/silence classes while keeping intact the number pitch classes. This unbalances the overall distribution and raises errors where rhythm tokens are confused with pitch tokens.



$$min\_step = ♪ \longrightarrow [C_4, \_, D_4, \_] \quad [C_4, \_, D_4, E_4] \longrightarrow pAcc(y, \hat{y}) = \frac{1}{1+1} = \frac{1}{2}$$

$$min\_step = 𝅘𝅥𝅯 \longrightarrow [C_4, \_, \_, \_, D_4, \_, \_, \_] \quad [C_4, \_, \_, \_, D_4, \_, E_4, \_] \longrightarrow pAcc(y, \hat{y}) = \frac{5}{5+1} = \frac{5}{6}$$

Figure 3.4: Example of Rhythm Accuracy giving different results when applied to the same data with different resolution.

In order to fix this behaviour we need to transform the input data before applying the metric such that the rhythm is a single value attached to a note instead of multiple values distributed among multiple time steps. This can be done by representing each note as Note-based discretization (see Temporal Scope in section 2.2.2) including the number of time-steps that a note is held as the rhythm value. The comparison then is applied similarly to *Pitch Accuracy*, where if two notes match in position, then the rhythm values of both notes are compared else the comparison is skipped and falls under *Position Score* evaluation.

Formally, *Rhythm Accuracy* is defined as:

$$rAcc(y, \hat{y}) = \frac{1}{N} \sum_{i=0}^{N} \sum_{j=0}^{N} \mathbb{1}_{duration(y_i)=duration(\hat{y}_j)} \mathbb{1}_{pos(y_i)=pos(\hat{y}_j)}$$

where N is the number of notes that share positions in both sequences, y is the gold sequence of notes, $\hat{y}$ is the predicted sequence of notes, $duration(y_i)$ retrieves the pitch value of the note at index i, $pos(y_i)$ retrieves the position of the note at index i. Note that the metric definition sums the number of notes that match both duration and position and then normalizes it by the number of notes present in both sequences.

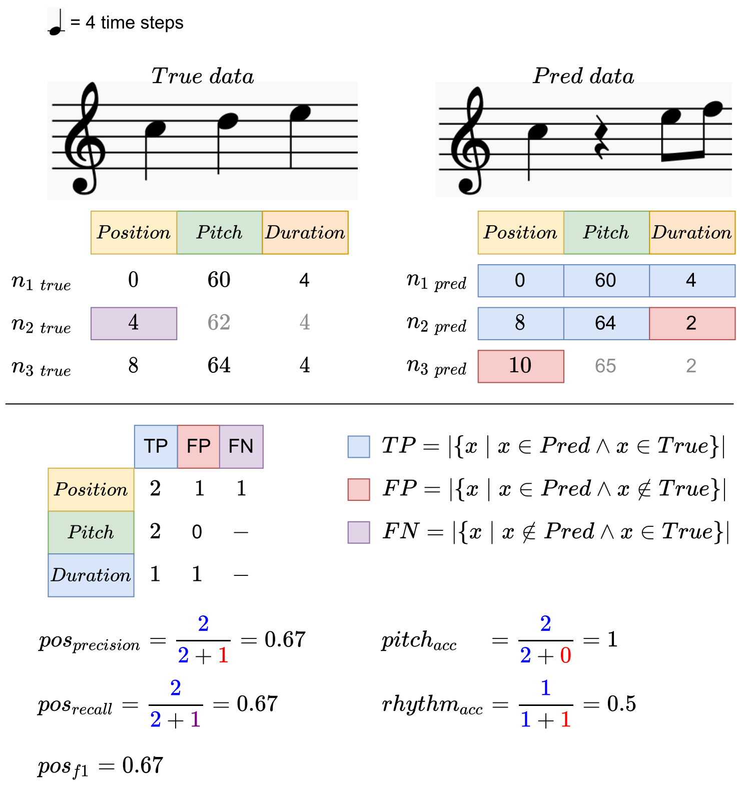
An example of these three metrics, including the data representation proposed for their application is shown in Figure 3.5.

3.3.2. Divergence Metrics

Although note metrics are useful for one-on-one comparison, there are cases in music generation where the attributes can not be directly compared since there are multiple correct options.

This variability in music is common and even desirable. However, there is a lack of methods to measure the correct variability of these attributes in generated data.

How do we verify that a given musical attribute in a set of predicted songs is within the correct range of variability? We argue that we need to look at the distribution of this attribute in true data and measure how close it is to the one in generated data. By measuring



	Position	*Pitch*	*Duration*
$n_{1\ true}$	0	60	4
$n_{2\ true}$	4	62	4
$n_{3\ true}$	8	64	4

	Position	*Pitch*	*Duration*
$n_{1\ pred}$	0	60	4
$n_{2\ pred}$	8	64	2
$n_{3\ pred}$	10	65	2

	TP	FP	FN
Position	2	1	1
Pitch	2	0	–
Duration	1	1	–

🟦 $TP = \lvert\{x \mid x \in Pred \land x \in True\}\rvert$

🟥 $FP = \lvert\{x \mid x \in Pred \land x \notin True\}\rvert$

🟪 $FN = \lvert\{x \mid x \notin Pred \land x \in True\}\rvert$

$$pos_{precision} = \frac{2}{2+1} = 0.67 \qquad pitch_{acc} = \frac{2}{2+0} = 1$$

$$pos_{recall} = \frac{2}{2+1} = 0.67 \qquad rhythm_{acc} = \frac{1}{1+1} = 0.5$$

$$pos_{f1} = 0.67$$

Figure 3.5: Note metrics evaluation pipeline. We represent each note in true and predicted data as triplets (Position, Pitch, Duration). We compute true positives, false positives, and false negatives for predicted positions. Then we calculate the position-F1 score, pitch accuracy, and rhythm accuracy. Since we can only compare notes present on both sets, we filter false positives and false negatives when calculating pitch and rhythm accuracy.

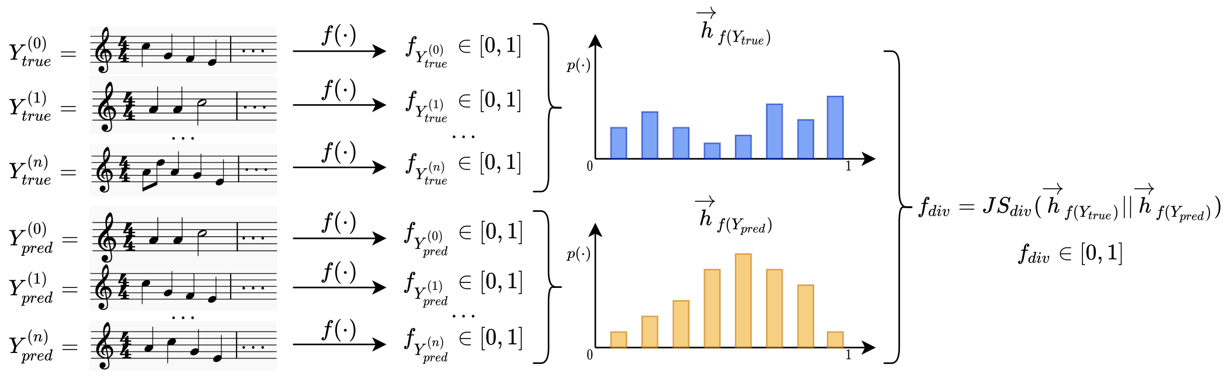

$Y_{true}^{(0)} =$
$Y_{true}^{(1)} =$
$Y_{true}^{(n)} =$
$Y_{pred}^{(0)} =$
$Y_{pred}^{(1)} =$
$Y_{pred}^{(n)} =$
$f(\cdot)$
$f_{Y_{true}^{(0)}} \in [0,1]$
$f_{Y_{true}^{(1)}} \in [0,1]$
$f_{Y_{true}^{(n)}} \in [0,1]$
$f_{Y_{pred}^{(0)}} \in [0,1]$
$f_{Y_{pred}^{(1)}} \in [0,1]$
$f_{Y_{pred}^{(n)}} \in [0,1]$
$\overrightarrow{h}_{f(Y_{true})}$
$\overrightarrow{h}_{f(Y_{pred})}$
$p(\cdot)$
0
1
$f_{div} = JS_{div}(\overrightarrow{h}_{f(Y_{true})} || \overrightarrow{h}_{f(Y_{pred})})$
$f_{div} \in [0,1]$

Figure 3.6: Divergence Metric of an arbitrary function f. Each sequence in Y_{true} and Y_{pred} is mapped to a single value in $[0, 1]$. Then, the distribution of these values for each set is compared using Jensen-Shannon Divergence.

this closeness between distributions we relax the condition of correctness to accept multiple valid answers.

We introduce *Divergence Metrics* to implement this measuring procedure by defining the divergence of a metric function $f : (x_0, x_1, \cdots, x_n) \to [0, 1]$ between true data Y_{true} and predicted data Y_{pred} as:

$$f_{div}(Y_{true}||Y_{pred}) = JS_{div}(\overrightarrow{h}_{f(Y_{true})}||\overrightarrow{h}_{f(Y_{pred})})$$

where $\overrightarrow{h}_{f(A)}$ is the histogram of the function values when applied to all sequences in a set and JS_{div} is the Jensen-Shannon Divergence [36] [16]. In our evaluation, we set $\overrightarrow{h}_{f(A)}$ to have 100 bins. This concept is illustrated in Figure 3.6.

Note that the metric function f maps a whole sequence to a single value. Since the value of the metric f changes from piece to piece, $f(A)$ will compute a distribution of values given a set A. Therefore, to compare the distribution of a metric in generated data against the one in true data we use a divergence. We choose JS_{div} since it is bounded, symmetric and do not require matching supports [38].

Silence Density Divergence quantifies if the predicted data contains the right amount of silence when compared to the distribution of silence in true data. It is formally defined as:

$$S_{div}(Y_{true}||Y_{pred}) = JS_{div}(\overrightarrow{h}_{S(Y_{true})}||\overrightarrow{h}_{S(Y_{pred})})$$

$$S(x) = \frac{1}{T}\sum_{t=0}^{T} \mathbb{1}_{n\_notes(x_t)=0}$$

where T is the total time steps in the sequence, and $n\_notes(x_t)$ is the function that counts the number of notes played at a given time step x_t.

Pitch Class Divergence quantifies how similar is the pitch entropy in C_m and $\{C_p \cup C_f\}$. This comparison is done by computing the *Pitch Class Histogram Entropy* according to the notes pitch classes (i.e. C, C#, ..., A#, B) as defined in [49] for each isolated measure. Then we calculate the mean difference between pitch entropy in C_p and pitch entropy in $\{C_p \cup C_f\}$. We formally define H_{div} as:

$$H_{div}(Y_{true}||Y_{pred}) = JS_{div}(\overrightarrow{h}_{H(Y_{true})}||\overrightarrow{h}_{H(Y_{pred})})$$

$$H(x) = \frac{1}{n_1 n_2}\sum_{i=0}^{n_1}\sum_{j=0}^{n_2}|\mathcal{H}_{m_i} - \mathcal{H}_{m_j}|$$

$$\mathcal{H}_{m_i} = \mathcal{H}(\overrightarrow{h}_{pitch(m_i)}) = -\sum_{i=0}^{11} h_i \log_2(h_i)$$

where n_1 is the number of measures in C_m, n_2 is the number of measures in $\{C_p \cup C_f\}$, m_i are the measures in C_m, m_j are the measures in $\{C_p \cup C_f\}$, and $\overrightarrow{h}_{pitch(m_i)}$ is the pitch class histogram of a measure m_i.

Groove Similarity Divergence measures how similar are the groove patterns between C_m and $\{C_p \cup C_f\}$. The comparison is made by representing each measure as a sequence of time steps, where 1 corresponds to the start of a note and 0 is either a hold or a rest. Then the two sequences are compared by penalizing each unmatched value with an XOR function. Similar to *Pitch Class Divergence* we calculate the mean difference between groove similarities in C_p and $\{C_p \cup C_f\}$.

$$GS_{div}(Y_{true}||Y_{pred}) = JS_{div}(\overrightarrow{h}_{GS(Y_{true})}||\overrightarrow{h}_{GS(Y_{pred})})$$

$$GS(x) = \frac{1}{n_1 n_2}\sum_{i=0}^{n_1}\sum_{j=0}^{n_2}\mathcal{GS}(\overrightarrow{g}^{m_i}, \overrightarrow{g}^{m_j})$$

$$\mathcal{GS}(\overrightarrow{g}^{a}, \overrightarrow{g}^{b}) = 1 - \frac{1}{T}\sum_{t=0}^{T-1} XOR(g_t^a, g_t^b)$$

where n_1 is the number of measures in C_m, n_2 is the number of measures in $\{C_p \cup C_f\}$, m_i are measures in C_m, and m_j are measures in $\{C_p \cup C_f\}$, and $\overrightarrow{g}^{m_i}$ is the encoding of a measure as a rhythm sequence where 1 is an onset, while 0 represents hold or silence.

IrishFolk Dataset (≈ 300K samples)

Model	$NLL\downarrow$	$pos_{F1}\uparrow$	$pAcc\uparrow$	$rAcc\uparrow$	$S_{div}\downarrow$	$H_{div}\downarrow$	$GS_{div}\downarrow$
Anticipation-RNN	0.453 (*0.662)	0.930	0.657	0.860	0.017	0.060	0.007
InpaintNet	0.487 (*0.662)	0.860	0.517	0.750	0.013	0.174	0.024
SketchNet	0.539 (*0.516)	0.914	0.560	0.868	**0.005**	0.134	0.009
VLI	0.059	**0.968**	**0.911**	**0.965**	0.015	**0.010**	**0.006**

Table 3.2: MUSIB evaluation on IrishFolk Dataset.

JSB Chorales Dataset (≈ 2.4K samples)

Model	$NLL\downarrow$	$pos_{F1}\uparrow$	$pAcc\uparrow$	$rAcc\uparrow$	$S_{div}\downarrow$	$H_{div}\downarrow$	$GS_{div}\downarrow$
Anticipation-RNN	0.459	0.832	0.243	0.682	0.240	0.525	0.232
InpaintNet	0.327	**0.852**	**0.505**	**0.788**	**0.059**	0.411	**0.153**
SketchNet	0.605	0.833	0.272	0.708	0.079	0.529	0.228
VLI	1.053	0.827	0.283	0.747	0.087	**0.286**	0.306

Table 3.3: MUSIB evaluation on JSB Chorales Dataset.

3.4. Results and Discussion

In Table 3.2 and 3.3 we present the results of all four models evaluated on the IrishFolk and JSB Chorales Dataset, respectively. Arrows indicate if higher/lower values represent better performance. Values in parentheses are evaluation results declared in the literature. Note that VLI's NLL value is not comparable to the rest since the class prediction setup is encoded differently.

From Table 3.2 we can observe in the NLL metric that the results declared in the literature are not exactly reproduced in our experiments. In particular, the result for Anticipation-RNN, InpaintNet, and SketchNet, shows a percentual difference of +32 %, +26 %, and -4 %, respectively. We explain this behavior by two variables: hyperparameters and split sets. The reproduction of the models was performed by utilizing the hyperparameters defined on the publication of each model although the hyperparameters observed on the official projects' source code were different, causing potential inconsistencies. Additionally, since the split sets were not publicly available we defined our own sets for training, validation, and testing.

We observe that VLI is the best performer for IrishFolk Dataset in all metrics except for S_{div} while InpaintNet is the best performer on JSB Chorales Dataset for all metrics except for H_{div}. VLI, which is based on the Transformer architecture, performs better on a larger dataset. This is supported by the literature, where it has been documented that transformers models require larger datasets to generalize properly [19].

As seen in Figure 3.7 and 3.8, all models have a significant drop in performance from one dataset to another, being the only exception the InpaintNet model for the rAcc metric. In particular, the model with the biggest drop in performance is VLI. We argue that this tendency is explained by the difference in the size of each dataset.

Although InpaintNet achieves the best performance in JSB Chorales, the results roughly

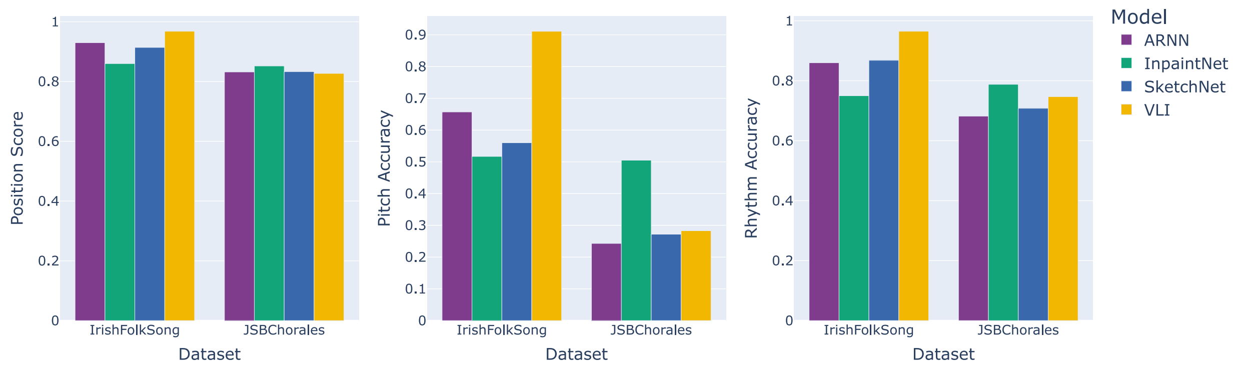


Figure 3.7: Comparison of Note Metrics for different datasets.

surpass 50 % pitch accuracy. This exhibits a significant gap in performance for this dataset in contrast to IrishFolkSong which is yet to be solved. Interestingly, from Figure 3.7 and 3.8, we note that InpaintNet has the most stable performance across datasets, having the lowest variation in results of all methods when testing them over IrishFolkSong and JSBChorales.This property may be helpful to improve the stability of other models when less data is available, thus improving the task in general.

SketchNet is good at reproducing silence distributions, as seen in S_{div} metric on both datasets, surpassing VLI in IrishFolkSong and seconding InpaintNet in JSB Chorales. This may be explained by SketchNet design. Its representation of data explicitly separates rhythm from the pitch, which may help the model focus more on rhythmic patterns and, consequently, on tokens of silence.

VLI is the best model for resembling the distribution of pitch classes between infilled data and its context, as seen in H_{div} metric. This may be explained by its XLNet-based architecture, which would make the model learn the correct pitch distribution first before the correct sequential order of the notes.

Looking at pos_{f1}, $pAcc$, and $rAcc$ values across all models we can see that models consistently perform the best in pos_{f1} metric compared to other Note Metrics. Similarly, all models perform better in $rAcc$ than in $pAcc$ independent of the dataset. We theorize that music inpainting models first learn to predict the correct onsets of notes, then learn the rhythm, and finally the pitch. However, further experimentation needs to be done in order to verify this hypothesis.

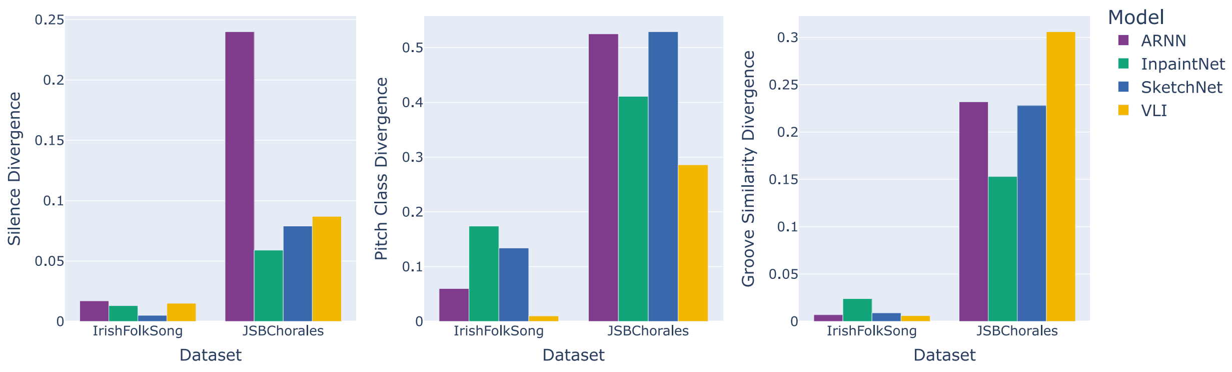


Figure 3.8: Comparison of Divergence Metrics for different datasets.

Capítulo 4

MUSIB Implementation Details

In this section, we show the implementation details for our framework MUSIB. The key components of the code are grouped as *Data Processing*, *Feature Extraction*, *Model Architecture*, *Training*, and *Evaluation*. All the code is written in Python, alongside multiple data science libraries such as Pandas or Numpy, and all the models are written with the Pytorch framework. The source code for MUSIB is publicly available and can be found in [1].

4.1. Data Processing

The data processing pipeline is in charge of collecting the raw data and filtering valid data according to the conditions presented in Chapter 3. Three modules separate the codes:

1. download_from_source.py
2. make_frames.py
3. clean_data.py

4.1.1. Download from source

This module is intended to collect, download, and format automatically all the required data files for the models present in MUSIB, assembling all the different data sources in one place.

As discussed in Chapter 3, we used two datasets for evaluation purposes: JSBChorales and IrishFolkSong, however, a third dataset called AILabs1k7 is also downloaded for checking the reproducibility of the re-implementation of the VLI model over our environment.

[1]`https://github.com/maranedah/music_inpainting_benchmark`

Dataset	Format	Source
JSBChorales	MusicXML	Github
IrishFolkSong	MIDI	Github
AILabs1k7	MIDI	Google Drive

Table 4.1: Comparison of datasets formats and sources

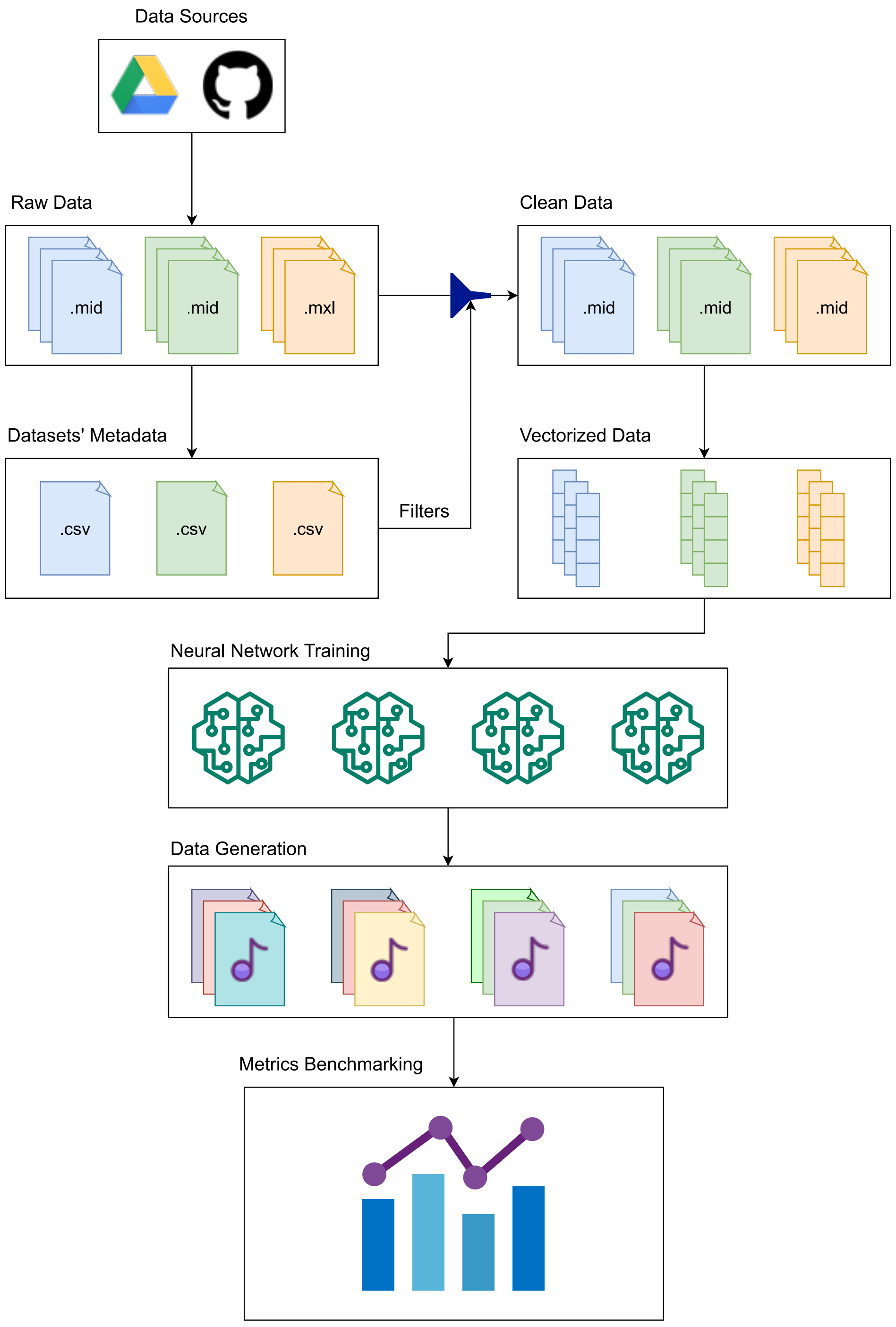


Figure 4.1: Diagram of the overall data pipeline in MUSIB.

To simplify further processing of the data, JSBChorales is transformed to MIDI format using the music21 library so all datasets are in the same format. The transformation from MusicXML to MIDI implies a loss of information, however, it is not relevant to our case since it affects metadata outside of pitch and rhythm.

4.1.2. Make data frames

This module is in charge of generating a dataframe that contains all the relevant metadata for each file, in particular for musical data. This metadata will be used for further filtering useful data.

Each file entry in the data frames will contain:

- filename: Name of the data file.
- n_instruments: Number of instruments in the file.
- instrument_names: Tuple containing the name id for the instruments present in the file.
- n_notes: Tuple containing the number of notes for each instrument in the file.
- is_monophony: Tuple containing boolean values for each instrument in the file representing whether each instrument is monophonic or polyphonic.
- max_poly: Tuple containing the max number of notes played at the same time for each instrument.
- polys_percent: Tuple containing the percent of time steps where simultaneous notes are played vs the total of time steps where notes are played for each instrument.
- tempo_changes: List of tuples representing all tempo changes for a piece (in bpm) alongside the time when the given tempo starts applying.
- tsc_length: Number of Time Signature Changes (TSC) in the musical piece.
- first_tsc: The first signature observed in a song. Can be 4/4, 3/4, 9/8, etc.
- is_4_4: A boolean value representing whether a song is in 4/4 time signature during all the piece.
- tsc: List of time signature changes present in the song.
- duration: Max duration of a song, considering all instruments present in the file.
- n_measures: Number of music measures in the song. It is calculated from the duration column.
- is_empty: Boolean value representing if a file has no instruments or zero notes.
- hash_val: String representing the file bytecode to identify possible repetition of songs.

4.1.3. Clean data

This module is in charge of filtering raw data to get the final train/val/test sets. We define five filters for our data:

- Non-empty: the file contains at least one instrument, and at least one note event.
- Non-repeated: the file does not share hash with other files, in which case the first is preserved and the copy or copies is discarded.
- 4/4 time signature: The file starts with a 4/4 time signature, and that time signature never changes in the span of the musical piece.
- Monophony: The file contains only one instrument, and such instrument never plays more than one note per time-step.
- Min Length: The file is at least sixteen measures long.

Table 4.2 shows the filters applied for each dataset. For the case of IrishFolkSong, the dataset is filtered with 4/4 time signature just to replicate the experiments in [11] and [40], although is not necessary. JSBChorales is not filtered as monophony since it contains multiple instruments even though all instruments are monophonic when separated. This processing step is done further in the pipeline. For more details see Section 4.2.2.

Filter	IrishFolk	JSBChorales	AILabs1k7
Non-empty	✓	✓	✓
Non-repeated	✓	✓	✓
4/4 time signature	✓		
Monophony	✓		
Min Length (16 measures)	✓	✓	

Table 4.2: Filters applied to each dataset.

4.2. Feature Extraction

The feature extraction is in charge of encoding the filtered data files into vectors to be fed into the model, as discussed in Section 2.2.3. The vector encoding can be Noteseq or REMI depending on the model. The module is separated into the following files:

- encoded_midi_noteseq.py
- encoded_midi_remi.py
- midi_dataset.py

4.2.1. Encoding MIDI as vectors

Encoding MIDI events into vectors is a challenging task. The absolute time defined on each MIDI event needs to be matched to a musical representation of rhythm: quarter notes, eighth notes, etc, which depends on the current tempo associated with each note.

Although the models replicated in this work contained logic for encoding the vectors from MIDI files, the algorithms were meant to work only in their source dataset and failed when applied to new datasets. This is due to hard-coding features from the dataset into the algorithm, making assumptions from the data such as constant tempo, same temporal resolution on the oncoming files, fixed ranges for pitch, velocity, etc. All these limitations held the algorithms from making generalizations.

Based on the implementation for vector encoding for VLI [10], we designed a new algorithm for vector encoding that could replicate the outputs of the original algorithms on their source code and datasets, but also could generalize to new datasets with different properties.

Our algorithm is separated into two parts: The generation of a time grid, and the approximation of note events to the grid.

Generation of the time grid

The time grid will contain all musically valid time steps found in a MIDI file given the number of subdivisions per measure for the model and the tempo specifications for the piece. See tempo_changes in Section 4.1.2 for details on tempo data.

Algorithm 1 Compute time grid

```
1: **Input:** Time-step subdivision $n$, list of tempo tuples containing bpm value and tempo start $\mathcal{T}$
2: Create empty $time\_grid$
3: **for** $i$ from 0 to $N-2$ **do**
4:     $bpm \leftarrow \mathcal{T}[i][bpm]$
5:     $start \leftarrow \mathcal{T}[i][start]$
6:     $end \leftarrow \mathcal{T}[i+1][start]$
7:     $time\_per\_beat \leftarrow bpm/60$
8:     $min\_time\_step \leftarrow time\_per\_beat/n$
9:     **while** $curr\_time < end$ **do**
10:        Append $curr\_time$ to $time\_grid$
11:        $curr\_time \leftarrow curr\_time + min\_time\_step$
12:    **end while**
13: **end for**
```

Approximation of note events

Although note events could have slight variations on absolute time when being played, the rhythmic intention is often pretty clear. To determine the rhythm of notes (i.e. start and end of notes) we approximate the absolute time of the events to the closest time present on the time grid with the following expression:

$$t_{quantized} = \underset{t \in \mathcal{T}}{\operatorname{argmin}}(|t - t_{event}|)$$

where $\mathcal{T} \in \mathbb{R}^n$ is the time grid vector and $t_{event} \in \mathbb{R}$ is the time for a given note event.

4.2.2. MIDI Dataset

File in charge of generating the batch logic for each model. Each file is loaded according to the split for train/val/test, then sent to the corresponding encoder depending on the model, where given a time resolution and a context size it will sample a vector representing a valid musical context from the file. For the case of JSBChorales, since the instruments are monophonic, each instrument is separately sampled from the file to construct monophonic contexts. These sampling procedures are random for training and deterministic for validation/testing. The retrieved vectors are then post-processed with zero or more of the following operations:

- Padding: Adds empty notes to match the number of tokens required for the model language utilized in VLI (512 tokens).
- Factorization: Separates notes into pitch and rhythm as described in Section 2.2.3. Applied for SketchNet and its internal VAE.
- FixEmpty: Adds an artificial note token for measures with zero notes played as in the implementation of [11]. This special encoding covers the cases of empty measures not being handled by some architectures. Utilized in SketchNet, InpaintNet, and their internal VAE models.

Model	Time Resolution	Context Size	Representation	FixEmpty	Factorize
VLI	16	16	REMI		
SketchNet	24	16	Noteseq	✓	✓
InpaintNet	24	16	Noteseq	✓	
ARNN	24	16	Noteseq		
VAE SketchNet	24	1	Noteseq	✓	✓
VAE InpaintNet	24	1	Noteseq	✓	

Table 4.3: Models encoding defined in MIDI Dataset

4.3. Model Architecture and Training

Module in charge of defining the model architectures for each method studied in this work. Each model code is included in model.py and train_model.py files:

- model.py: The class definition for each model. It exhibits relevant operations of the model, such as layer definitions and the forward function definition.
- train_model.py: Encapsulates the training and evaluation logic of the models. Its purpose is to condense the source code of all models into a file structure that would allow invoking the methods of a model through the same interface: $init\_model()$, $epoch\_step()$, $generate()$.

To be able to run the training procedure for each model we defined a general trainer function that iterates the training epochs by applying the epoch step function. This trainer is also in charge of validating the model, applying early stopping if required, saving the model checkpoints, and monitoring the training metrics in the console.

Composing all models under this module structure was a challenging task since each model had a different training logic in its code. For instance, the VLI model had its training algorithm inside the class definition for the architecture instead of having a separate class or script for that functionality. The same occurs with ARNN where additionally the class definition for the model includes the evaluation procedure.

4.4. Evaluation

Finally, the evaluation module is in charge of the evaluation of the output of each model. It comprises the logic for:

- Generating data: Standardize the output of all models to be readable by each of our proposed metrics.
- Evaluating data: Defines the metrics functions proposed in our benchmark and applies them to report the results.
- Transform vector data into MIDI: Allows to play the output of the models as standard MIDI format, which can be reproduced in most audio programs.

Capítulo 5

Conclusions and Future Work

5.1. Conclusions

In this thesis, we have proposed MUSIB, a new benchmark for musical score inpainting evaluation. We publicly released our experiments, models, data, and code for easier reproducibility[1]. We compiled, extended and proposed metrics to measure meaningful musical attributes in generated data. In particular, we defined two approaches for measuring musical score inpainting performances: Note Metrics and Divergence Metrics. The first relies on a one-on-one comparison of note attributes while the second relies on the comparison of distributions between the original dataset and the artificially generated data.

Our experiments allowed us to verify our hypothesis. We show that it is possible to find a unifying pattern across several models of musical score inpainting to directly compare their performance, validating our hypothesis. More over, we show that is possible to measure the variability of the models with respect to similar songs, being able to quantify the degree of closeness of musical terms by applying statistical strategies over key musical properties. We consider this last part to be particularly relevant for people interested in working with generative models since the evaluation of a generated piece with respect to what is correct is not trivial. This sets the paradigm of "what is correct"from "something is good if it achieves the rules for correctness arbitrarily defined"to "something is good if it is similar to other examples we know are good (because they are the training data and follow a particular musical style)".

We additionally summarize the key advantages of our evaluation proposal and its current limitations.

Advantages:

- We introduce a new evaluation procedure that compiles and standardizes metrics previously not compatible for different musical inpainting experiments.
- We introduce Position Score, a new metric that quantifies the alignment of matching

[1]`https://github.com/maranedah/music_inpainting_benchmark`

pairs of notes which works in pairs of sequences with different number of notes.

- We propose a new paradigm for evaluation (Divergence Metrics): metrics that respond to statistical information rather than a singular expected value.

Limitations:

- The correlation of human evaluation and our evaluation metrics is yet to be clarified.
- The calculation of divergence metrics is resource intensive because the number of samples needed for the evaluation has to be similar in size to the input data set.

5.1.1. Main Findings

The results obtained from our benchmark suggest interesting findings regarding the state-of-the-art of musical inpainting task:

1. It was not possible to exactly reproduce the results declared on the literature, having differences ranging from 4 % to 32 %. This suggests that there is still room for improvement in making the musical inpainting models more reproducible.
2. The performance of existing models is highly dependent on the amount of training data: while VLI, a transformer-based model, achieves the best results when more data is available, InpaintNet, a VAE-based model, excels when less data is available.
3. The performance of all models varies consistently and significantly as the dataset varies.
4. When comparing performance on the note metrics (onset, pitch, and rhythm), all models rank these metrics consistently, achieving the highest results for position score, then rhythm accuracy, and finally pitch accuracy. This gives clear signals regarding which aspects of the notes are harder to capture for any model.

These results have a number of implicates:

1) Replicating models may unexpectedly show that the results of a model can be better than the results declared in their original paper. In our case, we showed that Anticipation-RNN and InpaintNet got even better results than those described in prior work. In this case, we argue that the difference comes from a mismatch of the hyperparameters explained in their first publication and the hyperparameters set on the released code. This arises an interesting question: how many models are underperforming due to an incorrect setting of hyperparameters instead of a bad architecture design? This question is yet to be solved and opens up a path to check if newer models are intrinsically better than older models or if it is just that older models require a better tuning of hyperparameters.

2) Results show a trend that models trained on the bigger dataset perform better. Considering that, and the fact that models based on Deep Learning are expected to be trained with high amounts of data, we consider that an extension of musical inpainting models to

be compatible with datasets with more data samples such as LAKH would greatly benefit the state of the art on the task. However, one considerable limitation is that bigger datasets are more likely to be less standardized and thus new assumptions to process data will need to be done, such as the standardization of time events to fall into a singular resolution that fits for all samples. Additionally, existing datasets that are bigger than the ones we studied are less likely to contain monophonic pieces, which may need to define strategies to extract monophonic instruments such as voice, wind instruments, etc.

3) Study of performances of these models requires to consider several datasets since the differences in musical attributes greatly impact the ability to be reproduced by a given model.

We believe that our metrics can explain properties not shown in previous work that could be relevant for the music generation topic. These metrics of course can be further improved, and a correct evaluation is not necesarily limited to only these metrics. However, in order to make replicable experiments and to be able to set an argument on the state of the art, each new research should at least evaluate their proposal with the same set of metrics utilized in past works.

We expect our note metrics to show simpler musical behaviour of the predicted sequences even though it is not correlated to human evaluations. These type of metrics are to have a direct and interpretable meaning, making the output of the models easier to understand. In the other hand, we expect our divergence metrics to be more correlated to human evaluation since the point of comparison for humans can be thought as a distribution of properties from multiple songs heard. In case this metrics perform poorly we argue that the issue might be the metric function chosen rather than the algorithm for comparison. This metrics can be further extended by adding musical rules.

We expect a degree of correlation between our metrics. In particular since Rhythm and Silence are somewhat related, Rhythm Accuracy and Silence Divergence could be related. Similarly Pitch Accuracy and Pitch Histogram might have a degree of correlation.

Metrics can be further improved by adding different weights to different pitch errors. Usually changing a note for its third, fifth or octave won't have a great impact on the musicality of a piece. However, smaller distances of notes such as semitone would probably affect the sound of a piece.

5.1.2. Future Work

Considering our findings and our proposed benchmark, there are several possible extensions to this work. For instance, including more datasets that share similar sizes would clarify whether the difference in results we have shown depends only on the volume of data or if there are other variables such as the musical style that affect the models' performances.

In terms of evaluation, additional subjective listening experiments may be helpful to exhibit correlations between our proposed metrics and human perception of the generated data.

The evaluation of the models could also include data augmentation strategies. Different methods such as transposing a sequence to all keys or randomly changing a note with its third or fifth may have a different impact on the models' performance.

ur Note Metrics could be further extended. One promising idea is to add different weights to different pitch errors. This would follow the musical intuition that some notes are more dissonant than others. Changing a note for its third, fifth or octave should not have a great impact on the musicality of a piece since they share an implicit sense of harmony. On the other hand, changing a note for a non-harmonic note such as its second or tritone interval would likely make the piece to feel less enjoyable.

Similarly, new Divergence Metrics that evaluate other musical attributes could be of interest when evaluating music generation. This may include the amount of self-replication for the sequences on a given dataset, the number of harmonies or intervals, and the amount of polyphony, among others.

As a final remark, we hope that the proposed benchmark will benefit the community by paving the way to facilitate the reproducibility and evaluation of music inpainting models, as well as providing standards for the development of new ones.

Bibliografía

[1] Teodor Lucian Anton and Stefan Trausan-Matu. Byzantine music composition using markov models. In Adrian Sabou and Philippe A. Palanque, editors, *15th International Conference on Human Computer Interaction, RoCHI 2018, Cluj-Napoca, Romania, September 3-4, 2018*, pages 71–75. Matrix Rom, 2018.

[2] Aymé Arango, Jorge Pérez, and Barbara Poblete. Hate speech detection is not as easy as you may think: A closer look at model validation. In *Proceedings of the 42nd international acm sigir conference on research and development in information retrieval*, pages 45–54, 2019.

[3] J.S. Bach. *389 Chorales: for SATB Voices; Choral Score*. Kalmus Classic Edition. Alfred Publishing Company, 1985.

[4] Blanka Bogunović. Creative cognition in composing music. *New Sound International Journal of Music*, 53(1):89–117, 2019.

[5] Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. *arXiv preprint arXiv:1206.6392*, 2012.

[6] Jean-Pierre Briot, Gaëtan Hadjeres, and François-David Pachet. *Deep Learning Techniques for Music Generation*. Springer, 2020.

[7] Filippo Carnovalini and Antonio Rodà. Computational creativity and music generation systems: An introduction to the state of the art. *Frontiers in Artificial Intelligence*, 3:14, 2020.

[8] M. Cartagena. Exploring symbolic music generation techniques using conditional generative adversarial networks. Master's thesis, Escuela de Ingeniería. Pontificie Universidad Católica de chile, 2021.

[9] Edgar Castillo-Barrera and O. Vital-Ochoa. Using l-system grammars for music automatic generation. In Hamid R. Arabnia and Youngsong Mun, editors, *Proceedings of the 2008 International Conference on Artificial Intelligence, ICAI 2008, July 14-17, 2008, Las Vegas, Nevada, USA, 2 Volumes (includes the 2008 International Conference on Machine Learning; Models, Technologies and Applications)*, pages 318–322. CSREA Press, 2008.

[10] Chin-Jui Chang, Chun-Yi Lee, and Yi-Hsuan Yang. Variable-length music score infilling via xlnet and musically specialized positional encoding. In *Proc. of the 22th Int. Society for Music Information Retrieval Conf.*, pages 97–104, 2021.

[11] Ke Chen, Cheng-i Wang, Taylor Berg-Kirkpatrick, and Shlomo Dubnov. Music sketchnet: Controllable music generation via factorized representations of pitch and rhythm. In *Proc. of the 21th Int. Society for Music Information Retrieval Conf.*, pages 77–84, 2020.

[12] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[13] Michael Scott Cuthbert and Christopher Ariza. music21: A toolkit for computer-aided musicology and symbolic music data. 2010.

[14] Michael Scott Cuthbert and Christopher Ariza. Music21: A toolkit for computer-aided musicology and symbolic music data. In J. Stephen Downie and Remco C. Veltkamp, editors, *Proceedings of the 11th International Society for Music Information Retrieval Conference, ISMIR 2010, Utrecht, Netherlands, August 9-13, 2010*, pages 637–642. International Society for Music Information Retrieval, 2010.

[15] Maurizio Ferrari Dacrema, Paolo Cremonesi, and Dietmar Jannach. Are we really making much progress? a worrying analysis of recent neural recommendation approaches. In *Proceedings of the 13th ACM conference on recommender systems*, pages 101–109, 2019.

[16] Michel Marie Deza and Elena Deza. Encyclopedia of distances. In *Encyclopedia of distances*, pages 1–583. Springer, 2009.

[17] Chris Donahue, Huanru Henry Mao, Yiting Ethan Li, Garrison W Cottrell, and Julian McAuley. Lakhnes: Improving multi-instrumental music generation with cross-domain pre-training. *arXiv preprint arXiv:1907.04868*, 2019.

[18] Chris Donahue, Huanru Henry Mao, and Julian McAuley. The nes music database: A multi-instrumental dataset with expressive performance attributes. In *ISMIR*, 2018.

[19] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[20] Lukas Eibensteiner, Martin Ilcík, and Michael Wimmer. Temporal-scope grammars for polyphonic music generation. In Daniel Winograd-Cort and Jean-Louis Giavitto, editors, *FARM 2021: Proceedings of the 9th ACM SIGPLAN International Workshop on Functional Art, Music, Modelling, and Design, Virtual Event, Korea, 27 August 2021*, pages 23–34. ACM, 2021.

[21] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

[22] Gaëtan Hadjeres and Frank Nielsen. Anticipation-rnn: Enforcing unary constraints in sequence generation, with application to interactive music generation. *Neural Computing and Applications*, 32(4):995–1005, 2020.

[23] Gaëtan Hadjeres, François Pachet, and Frank Nielsen. Deepbach: a steerable model for bach chorales generation. In *International Conference on Machine Learning*, pages 1362–1371. PMLR, 2017.

[24] Curtis Hawthorne, Andriy Stasyuk, Adam Roberts, Ian Simon, Cheng-Zhi Anna Huang, Sander Dieleman, Erich Elsen, Jesse Engel, and Douglas Eck. Enabling factorized piano music modeling and generation with the MAESTRO dataset. In *International Conference on Learning Representations*, 2019.

[25] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[26] David M. Hofmann. A genetic programming approach to generating musical compositions. In Colin G. Johnson, Adrián Carballal, and João Correia, editors, *Evolutionary and Biologically Inspired Music, Sound, Art and Design - 4th International Conference, EvoMUSART 2015, Copenhagen, Denmark, April 8-10, 2015, Proceedings*, volume 9027 of *Lecture Notes in Computer Science*, pages 89–100. Springer, 2015.

[27] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[28] Wen-Yi Hsiao, Jen-Yu Liu, Yin-Cheng Yeh, and Yi-Hsuan Yang. Compound word transformer: Learning to compose full-song music over dynamic directed hypergraphs. *arXiv preprint arXiv:2101.02402*, 2021.

[29] Cheng-Zhi Anna Huang, Tim Cooijmans, Adam Roberts, Aaron Courville, and Douglas Eck. Counterpoint by convolution. In *Proc. of the 18th Int. Society for Music Information Retrieval Conf.*, pages 211–218, 2017.

[30] Cheng-Zhi Anna Huang, Tim Cooijmans, Adam Roberts, Aaron Courville, and Douglas Eck. Counterpoint by convolution. In *Proceedings of ISMIR 2017*, 2017.

[31] Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Ian Simon, Curtis Hawthorne, Andrew M Dai, Matthew D Hoffman, Monica Dinculescu, and Douglas Eck. Music transformer. *arXiv preprint arXiv:1809.04281*, 2018.

[32] Yu-Siang Huang and Yi-Hsuan Yang. Pop music transformer: Beat-based modeling and generation of expressive pop piano compositions. In *Proceedings of the 28th ACM International Conference on Multimedia*, pages 1180–1188, 2020.

[33] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In Yoshua Bengio and Yann LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[34] Joost N Kok, Egbert J Boers, Walter A Kosters, Peter Van der Putten, and Mannes Poel. Artificial intelligence: definition, trends, techniques, and cases. *Artificial intelligence*, 1:270–299, 2009.

[35] Qiuqiang Kong, Bochen Li, Jitong Chen, and Yuxuan Wang. Giantmidi-piano: A large-scale midi dataset for classical piano music. *arXiv preprint arXiv:2010.07061*, 2020.

[36] Jianhua Lin. Divergence measures based on the shannon entropy. *IEEE Trans. Inf. Theory*, 37(1):145–151, 1991.

[37] Yoichiro Maeda and Yusuke Kajihara. Rhythm generation method for automatic musical composition using genetic algorithm. In *FUZZ-IEEE 2010, IEEE International Conference on Fuzzy Systems, Barcelona, Spain, 18-23 July, 2010, Proceedings*, pages 1–7. IEEE, 2010.

[38] Frank Nielsen. On a generalization of the jensen–shannon divergence and the jensen–shannon centroid. *Entropy*, 22(2):221, 2020.

[39] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[40] Ashis Pati, Alexander Lerch, and Gaëtan Hadjeres. Learning to traverse latent spaces for musical score inpainting. In *ISMIR*, 2019.

[41] Colin Raffel. *Learning-based methods for comparing sequences, with applications to audio-to-midi alignment and matching.* Columbia University, 2016.

[42] Adhika Sigit Ramanto and Nur Ulfa Maulidevi. Markov chain based procedural music generator with user chosen mood compatibility. *International Journal of Asia Digital Art and Design Association*, 21(1):19–24, 2017.

[43] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 2022.

[44] Shipra Shukla and Haider Banka. Monophonic music composition using genetic algorithm and bresenham's line algorithm. *Multim. Tools Appl.*, 81(18):26483–26503, 2022.

[45] Bob L. Sturm, João Felipe Santos, Oded Ben-Tal, and Iryna Korshunova. Music transcription modelling and composition using deep learning. *CoRR*, abs/1604.08723, 2016.

[46] Benigno Uria, Iain Murray, and Hugo Larochelle. A deep and tractable density estimator. In *International Conference on Machine Learning*, pages 467–475. PMLR, 2014.

[47] Karsten A. Verbeurgt, Michael Dinolfo, and Mikhail Fayer. Extracting patterns in music for composition via markov chains. In Robert Orchard, Chunsheng Yang, and Moonis Ali, editors, *Innovations in Applied Artificial Intelligence, 17th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, IEA/AIE 2004, Ottawa, Canada, May 17-20, 2004. Proceedings*, volume 3029 of *Lecture Notes in Computer Science*, pages 1123–1132. Springer, 2004.

[48] Ziyu Wang, Ke Chen, Junyan Jiang, Yiyi Zhang, Maoran Xu, Shuqi Dai, Xianbin Gu, and Gus Xia. Pop909: A pop-song dataset for music arrangement generation. *arXiv preprint arXiv:2008.07142*, 2020.

[49] Shih-Lun Wu and Yi-Hsuan Yang. The jazz transformer on the front line: Exploring the shortcomings of ai-composed music through quantitative measures. In Julie Cumming, Jin Ha Lee, Brian McFee, Markus Schedl, Johanna Devaney, Cory McKay, Eva Zangerle, and Timothy de Reuse, editors, *Proceedings of the 21th International Society for Music Information Retrieval Conference, ISMIR 2020, Montreal, Canada, October 11-16, 2020*, pages 142–149, 2020.

[50] Li-Chia Yang, Szu-Yu Chou, and Yi-Hsuan Yang. Midinet: A convolutional generative adversarial network for symbolic-domain music generation. *arXiv preprint arXiv:1703.10847*, 2017.

[51] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, 32, 2019.

[52] Li Yao, Sherjil Ozair, Kyunghyun Cho, and Yoshua Bengio. On the equivalence between deep nade and generative stochastic networks. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 322–336. Springer, 2014.

[53] Halley Young. A categorial grammar for music and its use in automatic melody generation. In Michael Sperber and Jean Bresson, editors, *Proceedings of the 5th ACM SIGPLAN International Workshop on Functional Art, Music, Modeling, and Design, FARM@ICFP 2018, Oxford, UK, September 9, 2017*, pages 1–9. ACM, 2017.