Apprentissage automatique pour la modélisation de musique symbolique

Nathan Fradet

THÈSE DE DOCTORAT

Apprentissage automatique pour la modélisation de musique symbolique

Deep Learning for Symbolic Music Modeling

Auteur:
Nathan Fradet

Thèse préparée au/à
LIP6 - Sorbonne University, CNRS
Aubay

défendue le 14 Mars 2024
pour le obtenir le grade de Docteur de Sorbonne Université

devant le jury composé de :

François Pachet - Spotify	Président du jury
Louis Bigo - LaBRI, Université de Bordeaux	Rapporteur
Philippe Pasquier - Metacreation Lab, Simon Fraser Univ.	Rapporteur
Gaëtan Hadjeres - Sony AI	Examinateur
Amal El Fallah Seghrouchni - LIP6, Sorbonne Université	Directrice de thèse
Jean-Pierre Briot - LIP6, Sorbonne Université	Co-directeur de thèse
Nicolas Gutowski - LERIA, Université d'Angers	Encadrant de thèse
Fabien Chhel - ESEO, ERIS	Encadrant de thèse

SORBONNE UNIVERSITY

Abstract

Deep Learning for Symbolic Music Modeling

by Nathan Fradet

Supervisors:
Amal El Fallah Seghrouchni - LIP6, Sorbonne University - CNRS
Jean-Pierre Briot - LIP6, Sorbonne University - CNRS
Nicolas Gutowski - LERIA, Angers University
Fabien Chhel - ESEO

Keywords: Deep Learning, Music Modeling, Music generation

Symbolic music modeling (SMM) represents the tasks performed by Deep Learning models on the symbolic music modality, among which are music generation or music information retrieval. SMM is often handled with sequential models that process data as sequences of discrete elements called tokens. Such models are widely used for natural language processing (NLP) and leverage the dependencies between the input tokens which represent words or parts of words. The tokenization and detokenization steps are usually performed by one of the several existing software libraries specifically made for NLP. For symbolic music however, there is no extensive dedicated software allowing researchers and engineers to easily use deep learning models with symbolic music. In addition, whereas tokens represent words for natural language, previous work on symbolic music use them to represent note and time attributes. This creates two major concerns: 1) the high length of the token sequences, which is a performance bottleneck; 2) the tokens do not carry much information by themselves other than their absolute value, which does not represent rich and meaningful musical information. In addition, the various ways to serialize musical information into sequences of different types of tokens remain to be analyzed and compared. Lastly, there are only a few publicly shared models, despite music generation being a growing topic. In this thesis, we address these challenges by: 1) developing a complete, flexible and easy to use software library allowing to tokenize symbolic music; 2) analyzing the impact of various tokenization strategies on model performances; 3) increasing the performance and efficiency of models by leveraging large music vocabularies with the use of byte pair encoding; 4) building the first large-scale model for symbolic music generation, which handles multiple instruments and all genres of music, and share it publicly. As we lower the barrier of entry for newcomers as well as experienced researchers and engineers, we aim to encourage more people to develop tools tied to deep learning and symbolic music, such as generative systems for assisted composition.

SORBONNE UNIVERSITÉ

Résumé

Apprentissage automatique pour la modélisation de musique symbolique

par Nathan Fradet

Encadrants:
Amal El Fallah Seghrouchni - LIP6, Sorbonne Université - CNRS
Jean-Pierre Briot - LIP6, Sorbonne Université - CNRS
Nicolas Gutowski - LERIA, Université d'Angers
Fabien Chhel - ESEO

Mots-clés: Apprentissage Automatique, Modelisation de la musique, Génération

La modélisation de la musique symbolique représente les tâches effectuées par les modèles d'apprentissage automatique pour la musique symbolique, parmi lesquelles figurent la génération ou la récupération d'informations musicales. Ces tâches sont généralement effectuées par des modèles séquentiels traitant les données sous forme de séquences d'éléments discrets appelés tokens. Ces modèles sont largement utilisés avec les langues naturelles et exploitent les relations entre les tokens, qui représentent des mots ou des parties de mots. La tokenization et la détokenization sont généralement effectuées par des bibliothèques logicielles existantes spécifiquement conçues pour les langues naturelles. En ce qui concerne la musique symbolique, il n'existe pas de bibliothèque logicielle dédiée permettant de la tokenizer facilement. De plus, dans les travaux antérieurs, les tokens représentent généralement des attributs de notes et de temps, ce qui pose deux problèmes majeurs : 1) la grande longueur de la séquence de tokens, qui représente une limitation de performance ; 2) les tokens ne portent pas beaucoup d'informations par eux-mêmes, hormis leur valeur absolue, ne permettant pas de représenter une information musicale riche. De plus, la modélisation de la musique symbolique reste largement inexplorée à ce jour. Il existe plusieurs façons de sérialiser l'information musicale en séquences de tokens de types différents, qui restent à être analysées. Enfin, il existe très peu de modèles partagés publiquement pour la musique symbolique. Dans cette thèse, nous relevons ces défis en : 1) développant une bibliothèque logicielle complète, flexible et facile à utiliser permettant de tokenizer la musique symbolique ; 2) améliorant les performances et l'efficacité des modèles en exploitant de vastes vocabulaires musicaux construits avec byte pair encoding ; 3) mettant en lumière les principales différences entre les différentes tokenizations de la musique symbolique ; 4) construisant le premier modèle à grande échelle pour la génération de musique symbolique, capable de gérer plusieurs instruments et tous les genres musicaux, et en le partageant publiquement. En abaissant les barrières d'entrée pour les nouveaux venus aussi bien que pour les chercheurs et ingénieurs expérimentés, nous espérons encourager davantage de personnes à développer des outils liant apprentissage profond et musique symbolique, tels que des systèmes de génération pour la composition assistée.

Acknowledgements

I thank Nicolas Gutowski for his long-time support. From the day, we met when I was an undergraduate student, we shared a friendly relationship, driven by our interest in new technologies among others. Nicolas has always been supportive to me, carrier-wise especially. When I had the opportunity to pursue a PhD program, he was to me the most natural supervisor that I wanted to work with and I am grateful that he accepted.

I thank Éric Remilleret for his trust, and having allowed me to pursue this PhD. I met Éric at Aubay during an internship through which he acknowledged the quality of my work and encouraged me to continue to explore the topic of music generation. This thesis and the collaborations that came with it, would not have been possible without him.

I thank Aubay for financing this thesis, which is accountable for the works that we achieved. I extend my sincere appreciation to the visionary management and decision-makers at whose foresight and belief in the value of research and innovation have made this journey possible.

I thank Jean-Pierre Briot and Amal El Fallah for having directed my thesis. Jean-Pierre is a recognized researcher, that was in the way to slow down his responsibilities and work load towards retirement at the beginning ot the thesis. He had no career interests in directing my thesis, other that continuing to contribute scientifically to the field and train a new researcher. He is passionate, as very few researchers are, and his happiness and enthusiasm has always felt like refreshing batches of motivation. I am very grateful for his dedication, as well as for the opportunities and connections he brought. Amal shares the same motivations than Jean-Pierre, with the exception that she is committed to large responsibilities as the head of the Ai Movement of UM6P and member of the COMEST at UNESCO. I deeply respect her dedication towards her home country, and considering her busy schedule, I am thankful for the time and interest she spent for me.

I thank Fabien Chhel for his comprehensive scientific knowledge, that considerably shaped this thesis. Fabien always had a clear and quick understanding of the complex problems that arose in the making of the thesis, and came with new ideas as a source of inspiration.

I thank the members of my thesis comity, Gaëtan Hadjeres and Philippe Esling, for the constructive exchanges we had.

I thank my colleagues from Aubay, LIP6, LERIA and ESEO for the moments we shared. Their company helped me through this journey in a undeniable way.

Lastly, I thank my family and friends for their love and support. I realize that following long studies is a chance, and for most people success do not come only with hard work, but requires good support, financially and emotionally. Having a supportive family allows to better focus and develop ourselves, and is a key factor to success and fulfilment. I am grateful for having received such good care.

Contents

List of Figures

List of Tables

List of Abbreviations

BPE	**B**yte **P**air **E**ncoding
CNN	**C**onvolutional **N**eural **N**etwork
CV	**C**omputer **V**ision
DAW	**D**igital **A**udio **W**orkstation
DL	**D**eep **L**earning
GM*	**G**eneral **M**IDI *1/2
GPU	**G**raphics **P**rocessing **U**nit
LSTM	**L**ong **S**hort-**T**erm **M**emory
MIDI	**M**usic **I**nstrument **D**igital **I**terface
MIR	**M**usic **I**nformation **R**etrieval
ML	**M**achine **L**earning
NLP	**N**atural **L**anguage **P**rocessing
NN	**N**eural **N**etwork
RNN	**R**ecurrent **N**eural **N**etwork
seq2seq	**Seq**uence **to** **Seq**uence
TSE	**T**oken **S**yntax **E**rror
VAE	**V**ariational **A**uto **E**ncoder

List of Symbols

	Typography	
x	italic lowercase letter	$\in \mathbb{R}$, scalar
X	italic uppercase letter	$\in \mathbb{R}$, scalar, usually a constant
$\mathbf{x}$	bold lowercase	$\in \mathbb{R}^d$, vector
$\mathbf{X}$	bold uppercase	$\in \mathbb{R}^{d_1 \times d_2 \cdots}$, matrix or tensor
x_i	subscript index	$\in \mathbb{R} \subset \mathbf{x}$
$x_{i,j}$	coma-separated subscripts	$\in \mathbb{R} \subset \mathbf{X} \subset \mathbf{x}_i$, element (i, j) of $\mathbf{X}$
$X^{(y)}$	superscript in parenthesis	X is related to y (e.g. belongs to)
$\mathcal{X}$	calligraphic letter	set of elements, $\mathcal{X} = \{*_i\}_{i=1}^{X}, \lvert\mathcal{X}\rvert = X$
$[\![x_1, x_1]\!]$	Double square brackets	Interval of integers from x_1 to x_2
	Common notations	
θ	learnable parameters	
p_θ	model with parameters θ	
$\mathbf{w}$	model weights	$\in \mathbb{R}^d$
β	model biases	$\in \mathbb{R}$
ℓ	loss	$\in \mathbb{R}^+$
η	learning rate	$\in \mathbb{R}^+$, usually $\in]0, 1]$
d	dimension of a vector or matrix	$\in \mathbb{N}^\star$
L	sequence length	$\in \mathbb{N}^\star$
t	time step index	$\in \mathbb{N}$
$\mathbf{e}$	embedding	$\in \mathbb{R}^d$
$\mathbf{h}$	hidden state	$\in \mathbb{R}^d$
$\mathcal{V}$	vocabulary	
V	vocabulary size ($\lvert\mathcal{V}\rvert$)	$\in \mathbb{N}$
$\mathcal{O}(\cdot)$	computational complexity	
$\oplus$	concatenation	

List of Publications

ISMIR 2021 LBD	MidiTok: A Python package for MIDI file tokenization [55]
ISMIR 2023	Impact of time and note duration tokenizations on deep learning symbolic music modeling [57]
EMNLP 2023	Byte Pair Encoding for Symbolic Music [56]

Chapter 1

Introduction

Music is historically represented by sequences of symbols describing notes and their attributes. Music composition has been a writing process of notes and annotations, and musicians played, improvised, and wrote music on sheets. Since the last century, the process of music composition has drastically changed with the progress and wide adoption of technology. First with electronic devices, that offered musicians ways to produce unique new sounds from synthesizers, record their music on analog supports, sample them with sequencers, and in the end create new music and even new genres that did not exist. More recently, the main support for music has once again changed with the progress of computers, to become digital. Now, almost every musician composes music with computers or digital devices, to record, process, mix and master songs. The analog devices in studios have been replaced by computers, Digital Audio Workstation (DAW) softwares and Virtual Studio Technology (VST) plugins[1]. Music creation is nowadays mostly computerized.

The rapid advancement of Deep Learning (DL) in since the years 2010s has ushered in a transformative era across various domains, from Natural Language Processing (NLP) to Computer Vision (CV) and beyond. Within this spectrum of innovation, the intersection of Artificial Intelligence (AI) and music, specifically the application of deep learning to symbolic music tasks, has witnessed notable progress. After witnessing the major impact NLP and CV models have had in our daily lives through products that we use daily such as search engines, digital cameras or soon autonomous vehicle, we can easily think that DL will impact the music creation field too. Yet, this domain remains a niche within the broader deep learning landscape, characterized by immense potential and challenges.

Music modeling through deep learning holds great promise, primarily as a tool for assisting musicians in the creative process. However, the utilization of existing models in real-world musical contexts has been hampered by several significant shortcomings. Musicians and composers, despite their enthusiasm for embracing AI-based tools, often find the outputs of current models lacking in the nuanced artistry that characterizes human musical expression. The controllability of these models, essential for shaping music according to artistic intent, is often limited, and the usability of such systems in practical music production workflows remains challenging.

The fundamental objectives of this thesis lie at the heart of addressing some of these critical issues and advancing the state of symbolic music generation with deep learning models.

[1] Digital instruments and effects plugins that can be embedded in DAWs. They can for instance emulate and reproduce the sounds of classic analog synthesizers.

1.1 Objectives of this thesis

Our research endeavors to bridge the gap between the potential of deep learning and the practical expectations of musicians, researchers, and engineers. Our pursuit encompasses two overarching goals:

1. Enhancing model performance and efficiency: We aim to push the boundaries of symbolic music generation by improving the quality, creativity, and expressiveness of the generated compositions. Through innovative approaches in deep learning architecture and training methodologies, we seek to develop models that can produce music compositions more convincing. Furthermore, we strive to enhance the computational efficiency of these models to enable real-time music generation, making them more practical for real-world applications;

2. Lowering the barrier of entry for AI-assisted music creation: In our pursuit of advancing the field, we acknowledge the importance of democratizing access to cutting-edge music generation tools. We aspire to provide open solutions that empower researchers, musicians, and engineers with the means to integrate AI-assisted music creation into their creative processes and products seamlessly. This includes developing user-friendly interfaces, clear documentation, and accessible software implementations that democratize the use of AI in music composition.

As we go at intersection of deep learning and music, our aim is to not only contribute to the academic discourse but also to foster innovation and creativity in the broader music community. By elevating the capabilities of symbolic music generation models and making them more accessible, we aspire to unlock new possibilities for artistic expression and collaboration between humans and machines.

1.2 Organization

The chapters that follow will delve into the intricacies of our research, presenting novel approaches, experimental findings, and practical implementations that collectively serve to advance the field of symbolic music generation with deep learning models.

The second and third chapters (Chapter 2 and Chapter 3) will introduce the background of the thesis, by focusing respectively on the base principles of DL and its existing application to symbolic music. We will detail how DL work, and present the main categories of Neural Network (NN) operations. We will specifically focus on the Transformer architecture, which will be extensively used in the experiments of the thesis. The third chapter will introduce how symbolic music can be used with DL models, and the works that have already been conducted in the field. The fourth chapter (Chapter 4) will introduce their current limitations, as well as the main challenges that researchers and people working in the field are facing.

The Chapter 5 will introduce MidiTok. It is a software contribution, that allows to easily tokenize MIDI files for sequential DL models such as Transformers. To use this model, one must first serialize the input data into a sequence of elements intelligible to the model. This step is called the tokenization, and is performed by a *tokenizer*. Its role is basically to translate the data between its original form to the token form for the model. Before MidiTok, there was no tool allowing to easily tokenize music, while allowing to preprocess the files with great flexibility and extended features. MidiTok will be used for the experiments in this thesis.

Music can actually be tokenized by different ways, as the notes and time information can be serialized by different manners. We will see in the Chapter 6 that the information

representation choices lead to different results. These results may also vary depending on the task at hand, and that the explicit information carried by the tokens is a key factor helping models to learn.

The tokenization of music, as we presented it until the Chapter 7, presents two major drawbacks: 1) the number of token per sequence for a piece of music is very long, leading to computational bottlenecks and inefficient models; 2) the "understanding" capacities of the model are greatly underused due to a mismatch between usual vocabulary sizes and the number of embedding dimensions. We will show in this chapter that Byte Pair Encoding, a compression technique, can address these two problems and allow to increase model performances and efficiency.

Finally, we contribute to open science in the Chapter 8 by publicly sharing large music models. In the last years, we have seen a large number of large models being openly shared on the internet. This sharing stimulates research and innovation, while lowering the barrier of entry into the field as training large models is very expensive. At the time of writing, we are not aware of any performant, general and easy to use model for symbolic music generation. We intend to fill this gap by releasing a first large music model, that can be used for generative purposes and finetuned.

We will finally conclude in the Chapter 9 by summarizing the contributions of the thesis, and give directions that are probably worth to be explored to further improve the deep learning approaches for symbolic music modeling. You will find in the end of the document the appendixes, that bring additional details for some of the previous chapters.

Part I

Overview

Chapter 2

Deep Learning

This chapter aims to give a short overview of the deep learning (DL) field, by introducing the most commonly used techniques. It will not be comprehensive, as the field evolves very quickly, and is large enough to be covered by books [66, 195]. It will rather focus on the fundamental bases of DL, introduce the main families of model architectures and for which purpose they are better used.

2.1 Deep Learning operations and modules

DL is a subfield of machine learning (ML), which is itself a subfield of AI. ML focuses on the development of algorithms and statistical models that solve tasks by learning from data. Hence, these models are not explicitly programmed and automatically learn to perform their task. Within ML, DL more specifically relies on artificial neural networks (NN) to solve complex tasks, such as image recognition, natural language processing, speech recognition, and even autonomous driving.

DL models are particularly suited for tasks that would require too much complexity to be programmed by humans. Image classification, pattern recognition or many computer vision (CV) tasks are concerned. This approach has become increasingly popular in recent years due to the growing availability of large datasets and powerful computational resources. It is a rapidly evolving field that holds tremendous promise for revolutionizing the way we interact with technology and making our lives easier and more efficient.

This section will cover the basics of DL: beginning with how does ML model are trained, following with artificial NN, and finishing with the general categories of DL models.

2.1.1 Machine learning theory

This subsection introduces the elementary theory behind ML - and subsequently DL, and specifically how such model are trained.

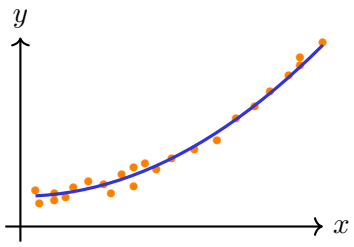


Figure 2.1: The relationship between $\mathcal{X}$ and $\mathcal{Y}$. Here the data $\mathcal{D}$ is represented by the dots, and the blue curve is the relationship the model aims to learn.

The fundamental goal of ML is to model the relationship between an input x and its associated result y. In other word, a model p_θ with trainable parameters[1] will be taught this relationship. The concept is pictured in Figure 2.1. To do this, one must collect a dataset of input and expected output pairs :

$$\mathcal{D} = \{(x_i, y_i)\}_{i=0}^{N}$$

The model learns by having its parameters updated so that $p_\theta(x) \to y$. This is done by computing the discrepancy between the predicted output and the true output, called the loss ℓ: $\ell = f(p_\theta(x), y)$ with f being the loss function. θ is optimized to minimize ℓ, the general training objective can be represented as:

$$\min_{\theta} \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}} [f(p_\theta(x), y)] \tag{2.1}$$

For DL models, optimization is done by gradient backpropagation [163]: the gradients are computed from the loss, operation to operation back to the input, and each parameter $\mathbf{w^t}$ is iteratively updated accordingly and regulated by a learning rate η:

$$\begin{aligned} \nabla\ell(\mathbf{w}) &= \left[\frac{\delta\ell}{\delta w_1}, \frac{\delta\ell}{\delta w_2}, ..., \frac{\delta\ell}{\delta w_n}\right] \\ \mathbf{w^{t+1}} &= \mathbf{w}^t - \eta\nabla\ell(\mathbf{w^t}) \end{aligned} \tag{2.2}$$

The loss function must be adapted to the task at hold. Classification tasks are for instance well coupled with the cross-entropy (negative log-likelihood) loss function.

Parameter updates are performed by an *optimizer*, accordingly to the gradients, in order to reduce the loss ℓ. Common optimizers are stochastic gradient descent [159] and Adam [101].

The learning rate is a hyperparameter that controls how much the model's parameters are updated. The role of the learning rate is crucial, as it can have a significant impact on the performance and convergence of the model. If the learning rate is too low, the model may take a long time to converge, or it may get stuck in a local minimum. On the other hand, if the learning rate is too high, the model may overshoot the optimal solution and diverge or oscillate around it. Therefore, selecting an appropriate learning rate is critical. There are several techniques for choosing the learning rate, such as grid search or random search, though it is most is often determined through a trial-and-error process. Additionally, learning rate scheduling can be used to adjust the learning rate during training, such as reducing it gradually as the model approaches convergence.

When trained, the model can be considered as a parametric approximation of $f : x \to y$. Its number of parameters is limited, and it is expected to perform on examples on which it has not been trained. Therefore it can still predict errors and show some uncertainty.

2.1.2 Artificial Neural Networks

A NN is a type of ML model inspired from the structure and functioning of the brain. It is a set of interconnected processing nodes, called neurons, and organized as successions of layers. Each layer performs a specific function: the first in the input layer, which receives the input data; intermediate (or hidden) layers perform operations on the output of the previous one; and the output layer produces the final prediction, that can take multiple forms, such as a probability distribution, an image, an audio or signal. A neural network is depicted in Figure 2.2.

[1] Often shorten as *weights* for DL models, the distinction is made in Subsection 2.1.2

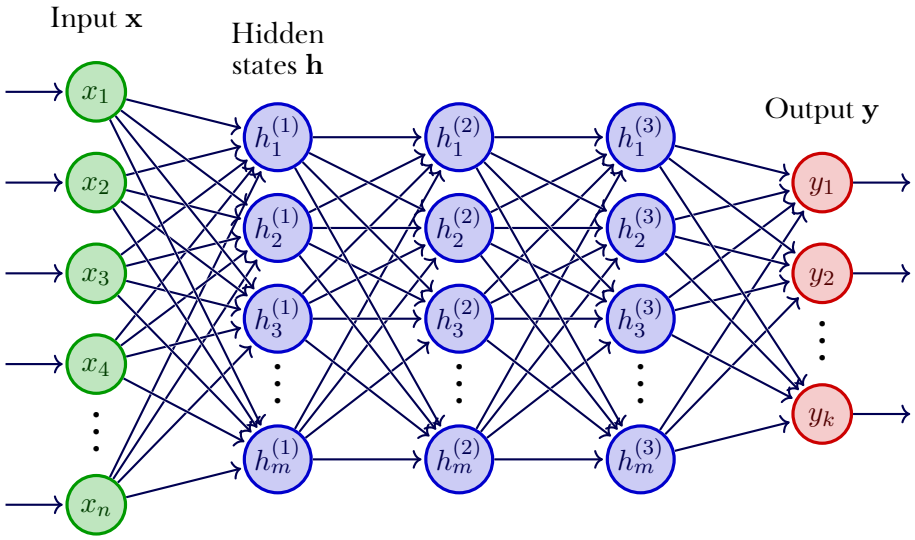


Figure 2.2: Schema of a neural network with four hidden layers. Edges can be seen as weights, biases are not represented.

NNs feature two kind of parameters: weights and biases. Weights, noted $\mathbf{W}$, are factors multiplied with the input feature, while biases are added to the weighted sum of the input features and the weights. For a given layer, weights $\mathbf{W} \in \mathbb{R}^{d_{in}, d_{out}}$ is a matrix and biases $\beta \in \mathbb{R}^{d_{out}}$ a vector, where d_{in} and d_{out} are respectively the number of input and output features. In other words, each neuron receives input from multiple other neurons, performs a weighted sum of those inputs, and adds the bias to it. The operation is described in Equation (2.3), where β and θ are respectively the bias and weights of the hidden layer. The set of all parameters of a model is commonly noted θ.

$$\begin{aligned} h_n &= \beta_n + \sum_{i=1}^{N} w_{n,i} x_i \\ \mathbf{h} &= \mathbf{W} * \mathbf{x} + \beta \end{aligned} \tag{2.3}$$

Weights operations are usually followed by activation functions [7], which introduce non-linearity and helps to stabilize gradient computations.

Neural networks are used in a wide range of applications, including image and speech recognition, natural language processing, recommendation systems, and many others. They are highly flexible and can learn to model complex non-linear relationships in data, making them powerful to automatically solve many tasks that would be too complex to be programmed.

In practice, a NN is made of stacked layers of weights and operations, though which the input is passed successively. The overall combination of layers and operations is called the **architecture**. The NN pictured in Figure 2.2 is a succession of fully-connected layers, and commonly called a Multi-Layer Perceptron (MLP). It is one of the simplest forms of NN. Modern NN will include more complex operations, among which convolution (Subsection 2.1.3), attention (2.2) or normalization [11] operations. It is important to note that for Equation (2.2) to be solvable, each operation of a NN must be differentiable, otherwise the gradients cannot be calculated and therefore the model's parameters cannot be updated.

Training a NN follows the same principles described in Subsection 2.1.1. In practice however, a lot of recommended techniques are used in order to get a good convergence,

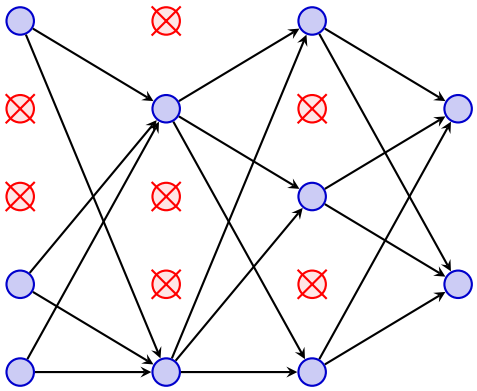


Figure 2.3: Schema of dropout during training: some inputs are omitted in the calculations of the next neuron values. In practice, it is common to use a 10% dropout rate.

get reliable predictions and mostly avoid overfitting. Overfitting happens when a model learns too well on the subset of data it has been trained on, but cannot yield accurate or reliable predictions on example outside of this subset. This is problematic as the end goal of such model is to be used in real conditions, where it is expected to encounter unseen examples. The most common technique to avoid overfitting is dropout. It consists in randomly dropping a proportion of input features at each layers during the training, as depicted in Figure 2.3. The model is then trained to yield predictions with reduced knowledge. This ensures that the remaining connections keep enough information. Other techniques are often used and combined, such as:

- layer normalization;
- batch normalization;
- residual blocks;
- weight decay;
- weight initialization;
- learning rate scheduling;
- data augmentation.

These methods are presented in books from Goodfellow, Bengio, and Courville [66] and Zhang et al. [195].

Multilayer perceptrons are however not suited for all types of data, at least when used alone. For instance, images are composed of multiple channels (for each color component) of pixels and are represented as 3-dimensional tensors: two for the pixel coordinates, one for each channel. One way to use feedforward layers on such data would be to flatten the matrix, but this would result in very large computations that could be intractable and make the model very inefficient. In the next subsections, we will introduce operations that address the most common data specifications: convolution for images and continuous modalities, and recurrent networks for discrete modalities.

2.1.3 Convolution

This subsection introduces the bases of convolution and convolutional neural network (CNN). For more information, we direct you to the comprehensive guide of Dumoulin and Visin [46].

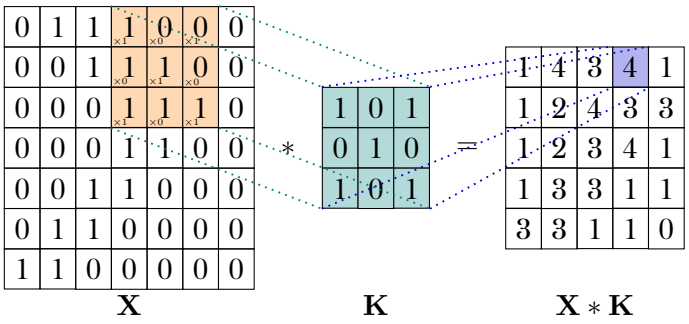


Figure 2.4: Schema of the convolution operation, with an input $\mathbf{X}$, a kernel $\mathbf{K}$ of size 3*3, no padding and a stride of 1. Input values and parameters are represented as integers for readability, in practice they are decimal values.

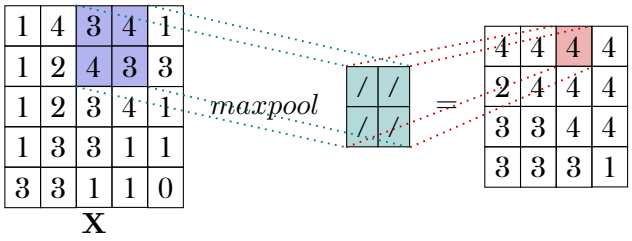


Figure 2.5: Schema of the max-pooling operation over an input $\mathbf{X}$, a kernel of dimension 3*3, no padding and a stride of 1.

A convolutional operation, also known as convolution or filtering, is a fundamental operation in convolutional neural networks (CNNs), a type of neural network commonly used with image, video and audio modalities. It have been first pioneered by Le Cun et al. in the 1990s [111], and grew in popularity to become the backbone operation of most computer vision (CV) DL models. Its big widespread occurred in the early 2010s after the publications of CNN models outperforming the state of art on the ImageNet [36] image classification benchmark. The most impactful work at this time was AlexNet [107], which used the power of GPUs to be scaled-up and yield a 16% error rate, which was 10% lower than the state of the art. Before that, CNN were mostly applied on simpler tasks such as character recognition. From this moment, the research in computer vision boomed and gave birth to bigger and better models.

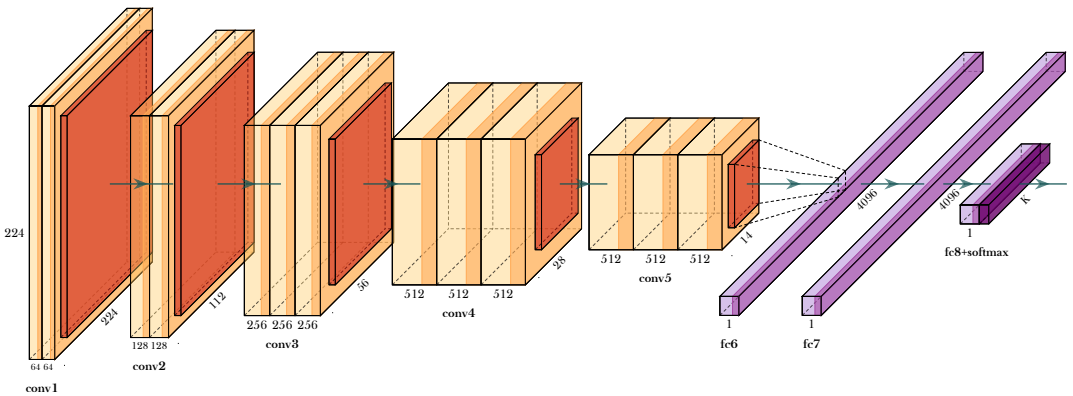


Figure 2.6: Schema of the layers of a CNN: VGG16 [171]. Orange boxes are convolution layers, red ones are pooling layers, and purple ones are fully-connected layers.

Convolution is inspired from the neuron connection patterns of the visual cortex of animals, and operate directly on matrices. The parameters of a convolutional layer is a matrix called kernel, or also filter. The kernel is "slid", or convolved, over the input matrix, the dot product between the filter and the local region of the overlapping matrix is computed for each position. This process results in a new output feature map, which highlights certain visual patterns or features present in the input image. With a kernel $\mathbf{K} \in \mathbb{R}^{m \times m}$, the computation is formulated as:

$$y_{i,j} = \sum_{a=0}^{m-1} \sum_{b=0}^{m-1} k_{a,b} x_{(i+a)(j+b)} \tag{2.4}$$

The operation is pictured in Figure 2.4. On this figure, no padding is performed. In practice padding is commonly performed: the kernel is slid on the edges of the input, and non-existent values are padded with 0, resulting in an output feature map of the same size than the input. The notion of padding is better explained and represented in the guide of Dumoulin and Visin [46].

In practice, an input image is made of several *channels*: one for each color component, green, red and blue. In a convolutional layer, a kernel will in fact have a shape of (w, h, c_{in}, c_{out}), where w and h are respectively its width and height, c_{in} and c_{out} the number of channels in input and output. The number of channels is commonly called the *depth*.

The convolutional layers in a CNN are usually stacked and separated by activation functions and pooling layers. An example of a CNN is pictured in Figure 2.6. Pooling consists in reducing the dimensionality of a feature map. This allows to scale down the feature currently processed, to focus on the essential patterns while reducing the complexity of the computations of the following layers. Pooling is performed in a similar manner than convolution: a window of lower dimension is slid over the feature map, and the output can be either the maximum value or average of the overlapping values. Max-pooling is represented in Figure 2.5.

To this day, CNN remains the preferred solution for computer vision tasks. However, recent research showed how Transformers and attention (introduced in Subsection 2.1.6) can be adapted to images are yield competitive results [45, 192, 40, 39] or build hybrid models [122, 187, 51]. In this thesis, we will not experiment with convolution. Symbolic music is naturally discrete, and other types of models are more suited for this kind of modality. We will present them in the following subsections.

2.1.4 Embeddings

This section introduces the notion of embedding, also commonly called *embedding vector* or *word embedding*. Embeddings are a key element of the experiments of this thesis, and are used in association with the sequential models introduced in the next subsections, that we will also extensively use.

Working with text for NLP tasks imposes to work with a set of finite known words present within the data, called the **vocabulary** noted $\mathcal{V}$. The vocabulary is essentially a look-up table, or dictionary, binding words to unique integer ids. These known words are also called **tokens**. Vocabularies are often made of thousands of words, each of them having a whole variety of meanings and significations. For a DL model to be efficient at processing these words, it must be able to capture their semantic information, and here is the goal of embeddings.

An embedding $\mathbf{e}^d$ is a vector of d dimensions, which represent the semantic information of the associated word. The embeddings of words are automatically learned by a model,

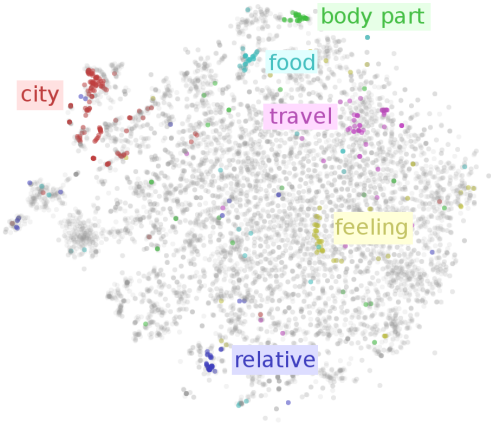


Figure 2.7: 2-dimensional visualization of a learned embedding space. Embeddings are dots, similar embeddings are close within the space.

so that it distributes embeddings with similarities close on specific dimensions, and pushes away those non-related. In other words, the model learns to project the words of $\mathcal{V}$ on a continuous space of d dimensions. Figure 2.7 shows a embedding space, reduced to two dimensions.

Embeddings offer a way for models to capture the semantic of words. Language models (introduced in Subsection 2.1.5 and Subsection 2.1.6) make use of embeddings for their downstream tasks.

A model learns word embeddings by leveraging their relationships from text data. During the training, the model's weights are updated to optimize the embedding vectors for their ability to predict the context in which words appear. Each dimension in the space represents a certain feature or attribute of the words of the vocabulary.

In the 2010s, word2vec [128, 129] and Glove [142] were popular word embedding models. Such models are based on a neural network architecture known as a skip-gram model or a continuous bag-of-words (CBOW) model. In the skip-gram model, the goal is to predict the context words surrounding a given target word within a certain window size. For example, given the sentence "The cat sat on the mat", and a window size of 2, the target word "sat" would be used to predict the context words "cat", "on", and "the".

The CBOW model is similar, but the roles of the target and context words are reversed: it has an input layer, a hidden layer, and an output layer. The input layer represents the target or context words as one-hot vectors. These vectors are then multiplied by a weight matrix to produce embedding vectors, which are fed to the hidden layer. The hidden layer then processes the embeddings to predict the probabilities of the context or target words in the output layer. During training, the model's weights are adjusted to optimize its ability to predict the context words given the target word or vice versa. As a result, the weights of the input layer represent the word embeddings, which capture the semantic meaning of the words in the training data.

Glove [142] on the other hand relies on both local and global statistics. It is based on the co-occurrence matrix of words in a dataset, i.e. the frequency with which words occur together in the same context. It factorizes the co-occurrence matrix into two smaller matrices, one for the rows and one for the columns. These matrices are then used to calculate a weighted least squares regression, which optimizes the embeddings to minimize the difference between their dot product and the log of the co-occurrence count. The intuition behind GloVe is that words that have similar co-occurrence patterns with other words are likely to have similar semantic meanings. For example, the words "cat" and "dog"

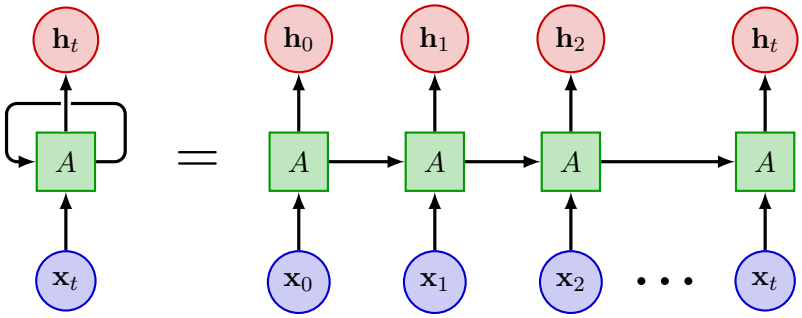


Figure 2.8: Schema of an unrolled RNN layer with input embeddings $\mathbf{x}_i$ and hidden states $\mathbf{h}_i$. The A cell is the same for all operations, that are performed sequentially: h_0 is computed first, then h_1... Only one layer is pictured here. The input can be either word embeddings or the hidden states of the previous layer.

are likely to co-occur with similar words like "pet" or "animal", and thus their embeddings should be similar in the resulting vector space.

At the time of writing, word embeddings are now mostly constructed contextually, i.e. directly conditioned on the whole contexts in which they are being used. The benefit of this method is that it allows to efficiently learn embeddings of word that can have several meanings. It also allows the models to capture more nuanced meaning in the text and perform better NLP tasks.

One of the first model based on this is ELMo [143]. ELMo is a bi-directional LSTM model (Subsection 2.1.5), trained with teacher forcing to predict the next element at each time step.

Another very popular model is BERT [37]. BERT is a Transformer (Subsection 2.1.6) trained to predict the original words from masked inputs. It does not use attention mask, hence all attention scores are computed conditionally to all input words (tokens), i.e. the whole context. This training is actually called "pre-training", as it does not train the model to solve any real task, but is performed before training the model a second time (finetuning) for the downstream tasks this time.

Next, we will introduce the types of sequential models commonly used for NLP tasks, beginning with recurrent neural networks.

2.1.5 Recurrent neural networks

A Recurrent Neural Network (RNN) [163] is a type of neural network that is particularly useful for modeling sequential data, such as time-series data or Natural Language Processing (NLP) tasks.

Unlike traditional feedforward neural networks, which process inputs in a strictly linear fashion, a RNN layer is also commonly called *cell*, and is conceived to process discrete data formatted as sequences of discrete elements. A layer receives information from a time step t, and the previous one $t-1$. The concept is depicted in Figure 2.8, and the operation is described in Equation (2.5).

$$\mathbf{h}_t = \tanh\left(\mathbf{x}_t\mathbf{W}^{(in)} + \mathbf{h}_{t-1}\mathbf{w}^{(past)} + \beta\right) \tag{2.5}$$

This makes RNN capable of processing sequences of inputs of variable length, and compute each hidden stated conditioned on the current and previous elements of the input sequence. A RNN model is usually constituted of stacked RNN layers. A model can can be designed to make classification operations over whole sequences, or to predict the next element in the sequence for each position within it. With the latter objective, a model

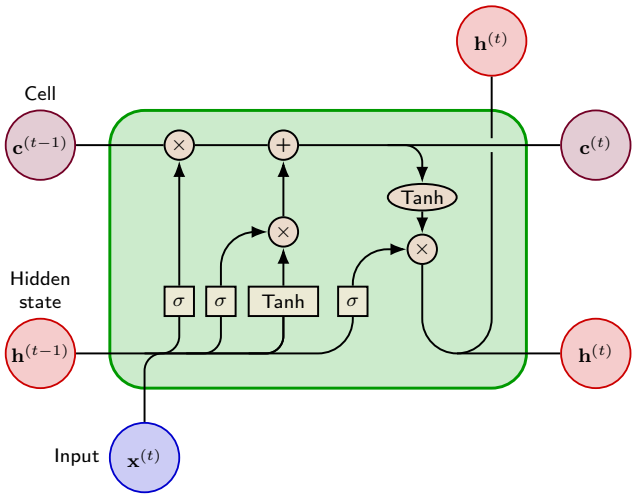


Figure 2.9: Schema of a LSTM cell. σ is the sigmoid function, operators in circles are pointwise: multiplication for $\bigotimes$ and addition for $\bigoplus$. Inner fully-connected layers are omitted.

directly learns to predict what is "coming next". It can then be used to generate content **autoregressively**, that is one element after another.

Training sequential models such as RNN is often performed with a technique called "teacher forcing": the whole ground truth sequence is used to calculate the loss. For example, if we want to train a RNN to generate content, we would need to train it by feeding it a sequence $\mathbf{x}_{0:t-1}$, and computing the loss with the target sequence $\mathbf{x}_{1:t}$. The loss is computed for all $t-1$ time steps independently, then aggregated by its average or maximum value.

RNN come however with a big limitation: vanishing gradients [80]. As gradients are backpropagated through time steps[2], they tend to usually become very small, i.e. vanish, when reaching the first ones, making the weights being updated very slowly. This can decrease the learning capability of a model, and in the worst scenario prevents it to further learn. The vanishing gradient problem is particularly problematic for RNNs. If the gradients in the feedback loop become very small, it can cause the network to have difficulty learning long-term dependencies, as the information from earlier time steps gets "washed out" before it can affect the output.

To address this issue, Hochreiter and Schmidhuber designed the Long Short Term Memory (LSTM) cell [81]. LSTM include an additional gating mechanism that allows the network to selectively "forget" or "remember" information from previous time steps. This extra memory, called the "cell" $\mathbf{c}_t$, is the sum of the weighted cell of the previous time step $\mathbf{c}_{t-1}$ and the weighted new information from the input $\mathbf{x}_t$. The full operation are described in Equation (2.6) where $\mathbf{i}$, $\mathbf{f}$, $\mathbf{g}$ and $\mathbf{o}$ represent respectively the input, forget, cell and output gates. The overall LSTM cell is depicted in Figure 2.9.

[2] Backpropagation through time, often abbreviated "BPTT".

$$
\begin{aligned}
\mathbf{i}_t &= \sigma\left(\mathbf{W}^{(ii)}\mathbf{x}_t + \beta^{(ii)} + \mathbf{W}^{(hi)}\mathbf{h}_{t-1} + \beta^{(hi)}\right) \\
\mathbf{f}_t &= \sigma\left(\mathbf{W}^{(if)}\mathbf{x}_t + \beta^{(if)} + \mathbf{W}^{(hf)}\mathbf{h}_{t-1} + \beta^{(hf)}\right) \\
\mathbf{g}_t &= \tanh\left(\mathbf{W}^{(ig)}\mathbf{x}_t + \beta^{(ig)} + \mathbf{W}^{(hg)}\mathbf{h}_{t-1} + \beta^{(hg)}\right) \\
\mathbf{o}_t &= \sigma\left(\mathbf{W}^{(io)}\mathbf{x}_t + \beta^{(io)} + \mathbf{W}^{(ho)}\mathbf{h}_{t-1} + \beta^{(ho)}\right) \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{g}_t \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c_t})
\end{aligned}
\tag{2.6}
$$

The forget gate controls the importance of the previous memory $\mathbf{c}_{t-1}$ in the current memory $\mathbf{c}_t$, the cell gate controls the importance of the input $\mathbf{x}_t$ in the current memory, and the output gate controls the importance of the current memory in the current hidden state $\mathbf{h}_t$. From this, we can see that the self-loop in LSTM is conditioned on the context $\mathbf{x}_t$ and $\mathbf{h}_{t-1}$ being processed. All gates use the same context, with different weights and biases.

Thanks to their ability to handle longer dependencies, LSTM models became increasingly popular over RNNs. They are however heavier to run, and a lighter gated-cell alternative was invented (which we will not cover here): the Gated Recurrent Unit (GRU) [25].

We also note that all these RNN cells can also be bi-directional: it can receive hidden states from both the previous and next steps in the sequence being processed. Despite this, RNNs still process time steps sequentially, and cannot model the inter-dependencies between all time steps. After being extensively used for many NLP tasks, they became to be overthrown by a new architecture: the Transformer [177].

2.1.6 Attention mechanism and Transformers

Attention mechanism and the Transformer architecture disrupted the whole DL domain. It was first used in the NLP field, outperforming previous methods on NLP tasks, and was quickly adapted to other domains such as computer vision. Transformer progressively replaced RNNs models. The music field also benefited from these techniques and the better performances and results they bring.

Attention

Attention is a DL operation that mimics the cognitive attention of humans and other living beings. When analyzing content, we instinctively pay attention to the relations between its elements. In the sentence *The city of Angers has a very beautiful castle*, the relationship between the words *beautiful* and *castle* is more important than between *Angers* and *beautiful* in this context. Attention in DL aims to measure the importance of these relationships to automatically understand which pairs make sens.

RNNs, which have been the go-to DL architecture before Transformers, "pass" the hidden states from one time-step to another. This limits the computation of distinct relationships between the elements of the input sequence. The LSTM cell, while performing better thanks to its gating and memory mechanism, is however heavier computationally, and still does not model distinct relationships. The attention mechanism was first introduced for this purpose.

Neural machine translation, the task of translating an input text into an other language, has often been tackled with "seq2seq" (Sequence to Sequence) model architectures of

RNNs: an encoder model receives the input sequence and processes it to produce a fixed-size hidden state called the context, that is fed to a decoder than autoregressively generates the translated sequence. This fixed-size context however greatly limits the access of information to the encoder, especially for long input sequences. The attention mechanism was created to alleviate this issue [13].

Bahdanau et al. introduced a way to dynamically update the context [13]. Rather than using a static context vector, the attention will update it by leveraging the relationships between the input and output (target) tokens. These relationships are modeled with trainable weights. The whole process is described in Equation (2.7), where L is the input sequence length, $\mathbf{c}_t$ is the context computed as the sum of the hidden states $\mathbf{H}^{(enc)}$ of the input sequence weighted by alignment scores a, $\mathbf{h}_t^{dec}$ is the decoder (p_θ) output hidden state at position t, and $\mathbf{v}_a$ and $\mathbf{W}_a$ are the attention weights.

$$\begin{aligned} \text{score}(\mathbf{h}_1, \mathbf{h}_2) &= \mathbf{v}_a^\top \tanh(\mathbf{W}_a(\mathbf{h}_1 \oplus \mathbf{h}_2)) \\ a_{t,j} &= \text{softmax}(\{\text{score}(\mathbf{h}_{t-1}^{(dec)}, \mathbf{h}_k^{(enc)})\}_{k=1}^{L}) \\ \mathbf{c}_t &= \sum_{j=1}^{L} \alpha_{t,j} \mathbf{h}_j^{(enc)} \\ \mathbf{h}_t^{(dec)} &= p_\theta(\mathbf{h}_{t-1}^{(dec)}, y_{t-1}, \mathbf{c}_t) \end{aligned} \tag{2.7}$$

score$(\cdot)$ is actually a feedforward neural network trained simultaneously with other parameters, which first concatenates hidden states $h_{t-1}^{(dec)}$ and $h_k^{(enc)}$, then multiplies them with weights before passing the result in a tanh activation for non-linearity. The context is hence computed according to the encoder hidden states, last decoder hidden state, and alignment scores between them. Due to the concatenation operation, this attention scheme could be classified as **additive**.

This first approach of attention allows to computes dynamically the relations between input and output prediction, offering better language understanding and more coherent autoregressive generation. Given this performance improvement, the attention got quickly extended to other variants, and other fields such as computer vision [186].

Luong et al. [120] described two variants of attention: the "global" and the "local" attention. The former is similar to the attention proposed by Bahdanau et al. [13]. The latter first predicts an aligned position p_t for the target word. The score $\mathbf{a}_t = \sum_{i=k_1}^{k_2} a_{t,i}$ is then computed from a window of positions $\mathcal{P} = \{k_1, k_1 + 1, ..., k_2\}$ centered around the position p_t of the source sequence. Figure 2.10 shows a visual representation of these two types of attention. However the most crucial novelty introduced here is the way the attention is calculated. Luong et al. considered three methods referred in Equation (2.8), where $\mathbf{h}_i$ and $\mathbf{h}_j$ are hidden states.

$$\text{score}(\mathbf{h}_i, \mathbf{h}_j) = \begin{cases} \mathbf{h}_i^\top \mathbf{h}_j & \text{Dot} \\ \mathbf{h}_i^\top \mathbf{W}_a \mathbf{h}_j & \text{General} \\ \mathbf{v}_i^\top \tanh(\mathbf{W}_a(\mathbf{h}_i \oplus \mathbf{h}_j)) & \text{Concat} \end{cases} \tag{2.8}$$

The Concat alignment method is the same the one introduced by Bahdanau et al. [13]. The Dot and General methods introduced here yield better results, leading to a faster convergence and lower perplexity[3]. The dot product strategy in particular, is now the base of the modern usages of attention at the time of writing.

[3] Measure of the uncertainty of a prediction. A low perplexity exhibits a certainty in the prediction.

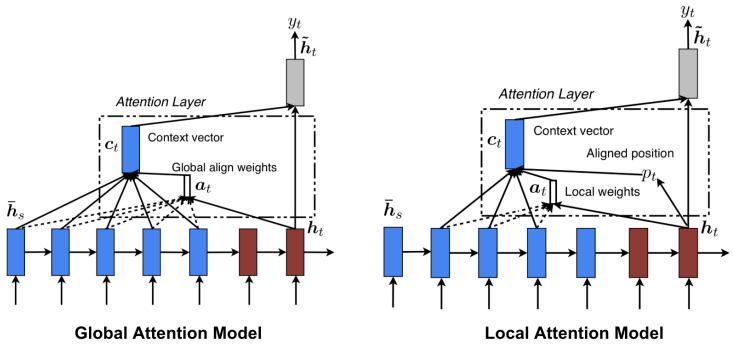


Figure 2.10: The "Global" and "Local" attention mechanisms from [120].

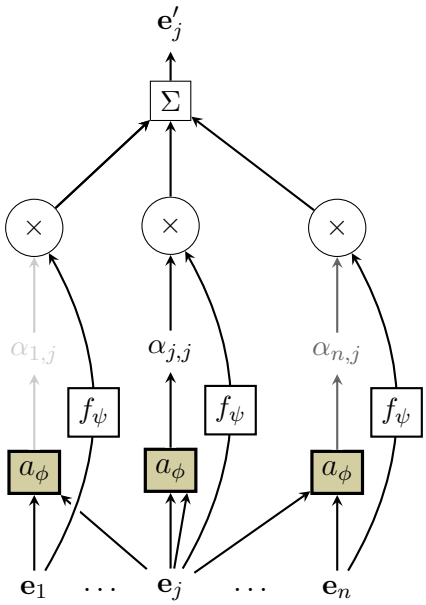


Figure 2.11: Schema of the scaled dot product attention. e'_j is the score for the jth element in the sequence.

Indeed, although these attention techniques brought new ways to efficiently learn representations in NLP tasks, the real disruption happened in 2017 with the introduction of the transformer architecture, and its core: the scaled dot product attention mechanism.

The Transformer architecture

In 2017, Vaswani et al. introduced the Transformer architecture [177] and what they named the scaled dot product attention.

The major novelty resides in the fact that Transformers are only made up of attention and feed-forward layers, whereas attention was used complementary with RNNs in the previous works, hence the title of this now famous paper "Attention is all you need".

The scaled dot product attention, described in Equation (2.9), is similar to the dot product attention, but includes an additional $\frac{1}{\sqrt{d_k}}$ scaling factor, assuring that the dot product does not grow too much in magnitude for large values of d_k, which would push the softmax function where gradients are extremely small. The scaled dot product attention works with *source* and *target* sequences, that are used with the encoder and decoder respectively. Keys

$\mathbf{K}$ and values $\mathbf{V}$ are derived from the source sequence through weight matrices. Queries $\mathbf{Q}$ are derived from the target sequence via the same manner. It is pictured in Figure 2.11.

A distinction is commonly made between what are called **self-attention** and **cross-attention**. The former computes attention between the elements of a sequence themselves (the source only), the latter between the elements of two distinct sequences (source and target).

$$\begin{gathered}\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}})\mathbf{V} \\ \mathbf{Q} \in \mathbb{R}^{T \times d_k},\ \mathbf{K} \in \mathbb{R}^{S \times d_k},\ \mathbf{V} \in \mathbb{R}^{S \times d_v}\end{gathered} \tag{2.9}$$

In most case, a model yields better results when computing attention separately on several chunks of the embedding dimension, as shown in Equation (2.10). Vaswani et al. called this technique **multi-head attention**.

$$\begin{gathered}\text{MultiHeadAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = [head_1; ...; head_h]\mathbf{W}^0 \\ head_i = \text{Attention}(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V) \\ \mathbf{W}_i^Q \in \mathbb{R}^{d_{model} \times d_k},\ \mathbf{W}_i^K \in \mathbb{R}^{d_{model} \times d_k},\ \mathbf{W}_i^V \in \mathbb{R}^{d_{model} \times d_v}\end{gathered} \tag{2.10}$$

The Transformer architecture was originally used for neural machine translation tasks, and so introduced as a seq2seq model. An encoder layer is made of self-attention and a feedforward layer along with a residual block[4] and normalizations. A decoder layer is similar to an encoder one, but includes a cross-attention layer following the self-attention, receiving the output hidden states of the encoder as queries. The overall architecture is showed in Figure 2.12.

Non-seq2seq Transformer are actually the equivalent of the encoder in Figure 2.12, i.e. without cross-attention. Such models, such as BERT [37] are used for language modeling and understanding, and others like the GPT family [145, 146] use a causal attention mask and are used autoregressively as a decoder.

Finally, the scaled dot product attention is a permutation invariant operation. This property can be very useful in some cases where the order of the elements of a sequence is not important, but is problematic when it is such as with text sequences. To alleviate this issue, Vaswani et al. added a positional encoding vector to each embedding before feeding them to the transformer layers. This encoding, described in Equation (2.11) where i is the position within the sequence and δ the dimension, injects in each embedding an information on its absolute position within the sequence, allowing the model to learn and distinguish the order of the given elements. Figure 2.13 shows a 2-dimensional representation of the absolute positional encoding.

$$\text{PE}(i, \delta) = \begin{cases} \sin(\frac{i}{10000^{2\delta'/d}}) & \text{if } \delta = 2\delta' \\ \cos(\frac{i}{10000^{2\delta'/d}}) & \text{if } \delta = 2\delta' + 1 \end{cases} \tag{2.11}$$

It outperformed other models in neural machine translation tasks with fewer training cost, and it did not take long for researchers to adapt the Transformer architecture and scaled dot product attention for other tasks, including music generation. The architecture itself, the attention and positional encoding has as well been subject to improvements and variations increasing the efficiency of models and/or leading to better results. We will specifically focus on these improvements in the next section.

[4] Residual blocks are "skip connection" parts of a model where a hidden state is passed to a series of operations, to which the result is added to this same hidden state as: $\mathbf{h}' = \mathbf{h} + f(\mathbf{h})$. The method was introduced by He et al. [74].

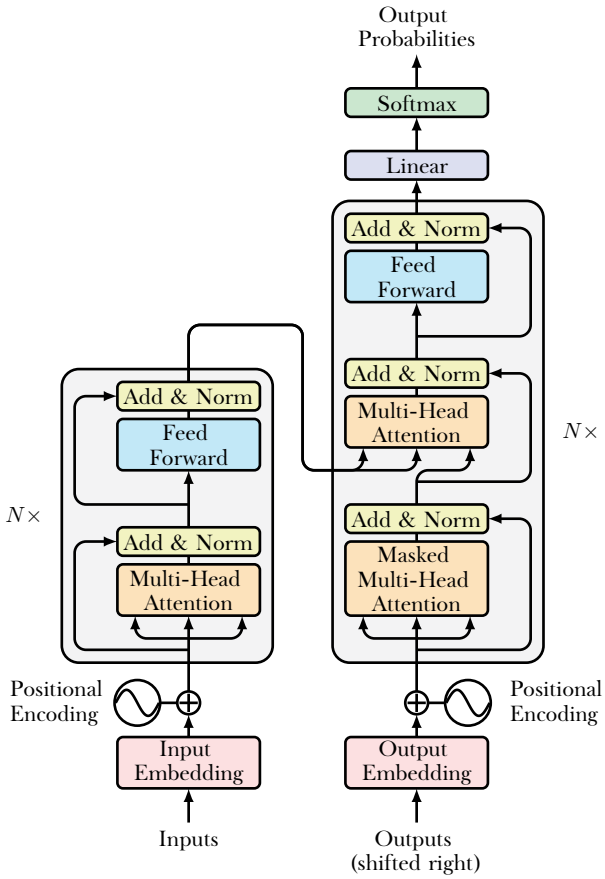


Figure 2.12: The full Transformer architecture, with an encoder (left) and a decoder (right). N refers to the number of layers.

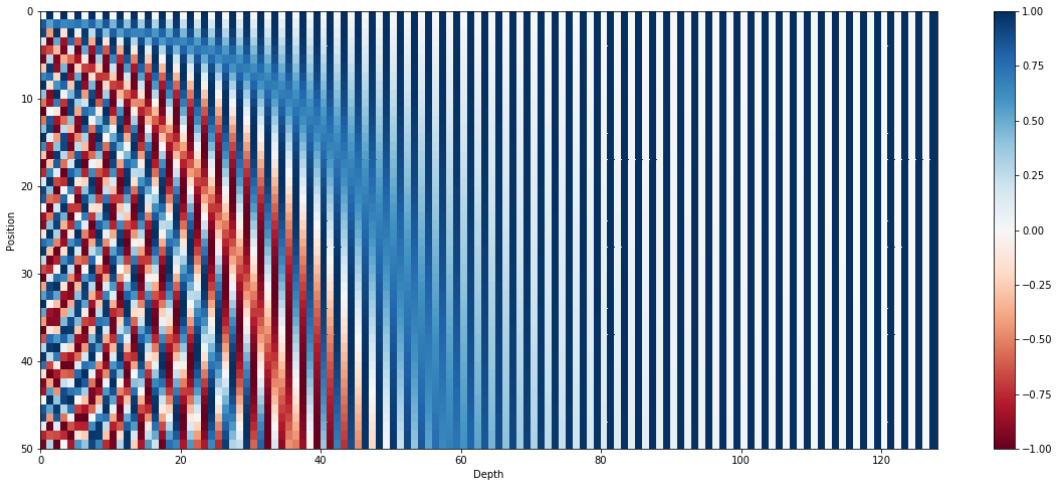


Figure 2.13: 2-dimensional representation of the absolute positional encoding

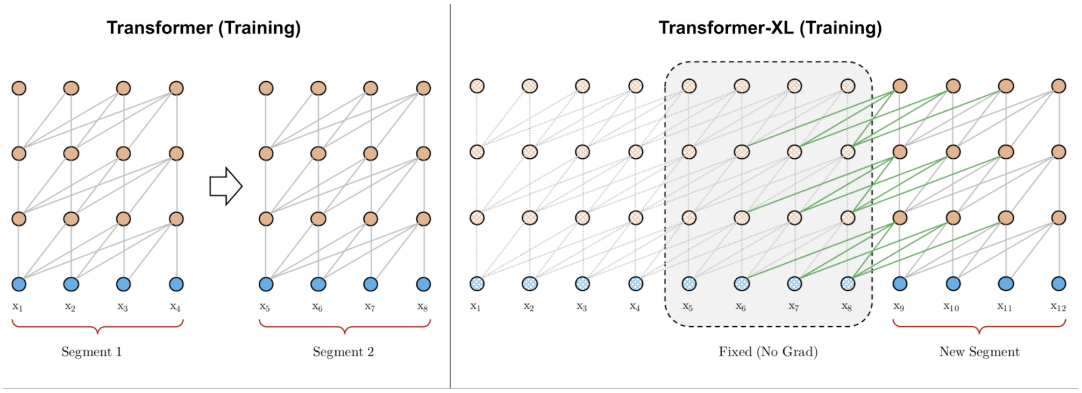


Figure 2.14: Left: a vanilla transformer training scheme where sequences are split into several segments; Right: Transformer-XL's extended attention span during training. Figure from Dai et al. [34].

2.2 Transformers

2.2.1 Improved and efficient transformers

This subsection introduces works extending the capacity of the "vanilla" Transformer architecture.

Transformer-XL

As the scaled dot product complexity grows quadratically with the sequence length, in many cases one must limit the size of the input data during training and inferring. This obviously limits the capacity of a transformer to attend to long sequences, and as for RNNs the model "forgets" what was processed before.

Transformer-XL [34] alleviate this issue with a clever mechanism reusing the previous hidden states. Transformer-XL is a causal transformer, meaning that a hidden state is computed from the previous ones only, intended to be used for content generation. With Transformer-XL, the hidden state $h^n_{\tau+1}$ for the $(\tau+1)$-th segment at the n-th layer is calculated as in Equation (2.12) where SG means the gradients are fixed and $\oplus$ is a concatenation.

$$\begin{aligned} \widetilde{\mathbf{h}}^{n-1}_{\tau+1} &= [\text{SG}(\mathbf{h}^{n-1}_{\tau}) \oplus \mathbf{h}^{n-1}_{\tau+1}] \\ \mathbf{q}^n_{\tau+1}, \mathbf{k}^n_{\tau+1}, \mathbf{v}^n_{\tau+1} &= \mathbf{h}^{n-1}_{\tau+1}\mathbf{W}^\top_q, \widetilde{\mathbf{h}}^{n-1}_{\tau+1}\mathbf{W}^\top_q, \widetilde{\mathbf{h}}^{n-1}_{\tau+1}\mathbf{W}^\top_q \\ \mathbf{h}^n_{\tau} &= \text{layer}_n(\mathbf{q}^n_{\tau+1}, \mathbf{k}^n_{\tau+1}, \mathbf{v}^n_{\tau+1}) \end{aligned} \tag{2.12}$$

This formulation allows to compute a hidden state $\mathbf{h}_t$ based on the previous ones $\mathbf{h}_{<t}$, which are being stored in memory. This not only speeds-up consequently the inferences, but also allows to compute each hidden state based on an extended number of previous hidden states. It allows to compute each token based on a larger past context. Figure 2.14 shows the hidden state reuse of the Transformer-XL.

Additionally, Transformer-XL include a built-in relative and elapsed positional encoding. If using the same positional encoding than the vanilla transformer, i.e. encoding the absolute positions, the previous hidden states would be assigned to different positions when being reused, which does not make sense. To make sure the positional information keeps relevant across segments, Transformer-XL uses the relative distances between elements.

Different from relative positional encoding, [168, 85], it reformulate the attention with absolute encoding as in Equation (2.13), u being the absolute positional information.

$$\begin{aligned} a_{i,j} &= \mathbf{q}_i \mathbf{k}_j^\top = (\mathbf{x}_i + \mathbf{u}_i)\mathbf{W}^q((\mathbf{x}_j + \mathbf{u}_j)\mathbf{W}^k)^\top \\ &= \mathbf{x}_i\mathbf{W}^q\mathbf{W}^{k\top}\mathbf{x}_j^\top + \mathbf{x}_i\mathbf{W}^q\mathbf{W}^{k\top}\mathbf{u}_j^\top + \mathbf{u}_i\mathbf{W}^q\mathbf{W}^{k\top}\mathbf{x}_j^\top + \mathbf{u}_i\mathbf{W}^q\mathbf{W}^{k\top}\mathbf{u}_j^\top \end{aligned} \tag{2.13}$$

From these four terms, Transformer-XL reparametrize the attention calculation as follows in Equation (2.14). The positional information of the key p_j is replaced with a relative positional $r_{i-j} \in \mathbb{R}^d$, the parameter matrix W^k is replaced with two matrices W_C^k and W_R^k for content and location information respectively, and $u_i W^q$ is replaced with two parameters $\mathbf{u}$ and $\mathbf{v}$ for content and location once again.

$$a_{i,j} = \underbrace{\mathbf{x}_i\mathbf{W}^q\,\mathbf{W}_C^{k\top}\,\mathbf{x}_j^\top}_{\text{content-based addressing}} + \underbrace{\mathbf{x}_i\mathbf{W}^q\,\mathbf{W}_R^{k\top}\,\mathbf{r}_{i-j}^\top}_{\text{content-dependent bias}} + \underbrace{\mathbf{u}\,\mathbf{W}_C^{k\top}\,\mathbf{x}_j^\top}_{\text{global content bias}} + \underbrace{\mathbf{v}\,\mathbf{W}_R^{k\top}\,\mathbf{r}_{i-j}^\top}_{\text{global positional bias}} \tag{2.14}$$

The combination of these strategies brought new state of the art results in language modeling on the WikiText-103 [126] and enwik8 datasets. Note that the Hugging Face Transformers [183] library natively implements a similar technique for autoregressive models, allowing to conveniently generate from Transformers very fast.

Transformers for images and localized attention

The attention mechanism was first introduced in the NLP field to process sequential data. But it did not take long before researchers sought to adapt it to images. A challenge though, is that images are 2-dimensional data often composed of a large number of pixels, with several channels (Subsection 2.1.3). Applying the scaled dot product attention over all of these elements would be consequently costly in time and memory, if tractable. A first solution would be to segment the data and apply attention independently on several segments. However the nature of images means that a pixel at position (x, y) should attend its neighboring pixels to make sens for tasks like classification or generation. Attention must then be applied not sequentially, but localized depending on (x, y) like convolution. With this consideration in mind, Image Transformer [138] expressed a way to localize attention around elements for 2-dimensional data, like images.

The image is split into several non-overlapping blocks of queries. Each query within can attend to elements in the memory block which contains those within the query block. This method is showed in Figure 2.15 for 1-dimensional and 2-dimensional applications. For both methods, images are generated autoregressively, one channel at a time.

Image Transformer, is a seq2seq architecture trained on three tasks: 1) Generate the empty parts of uncompleted images; 2) Augment the resolution of low-resolution images (super-resolution); 3) Generate images conditioned on class.

For these three tasks, both 1D and 2D methods produces coherent results, often close to the expected image, proving that attention can efficiently be used with image data, and more generally 2 and 3-dimensional data.

Sparse attention

In order to use attention for images efficiently, Child et al. introduced Sparse Attention [23] a way to factorize the attention patterns across inference steps. This method reduces the attention range of each inference, while making sure that each element attends others sequentially. Figure 2.16 shows a visual representation of these attention schemes. The key idea is to make use of several heads to attend different position sets.

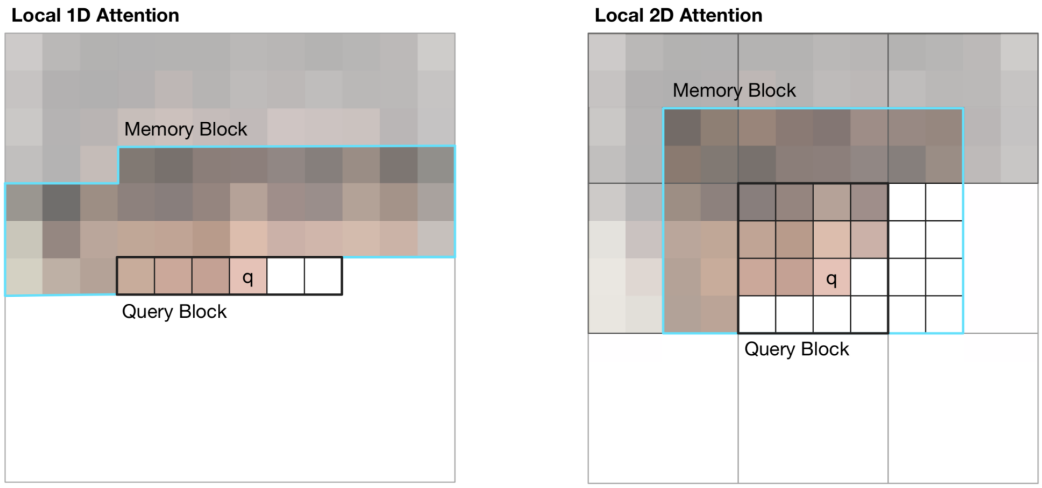
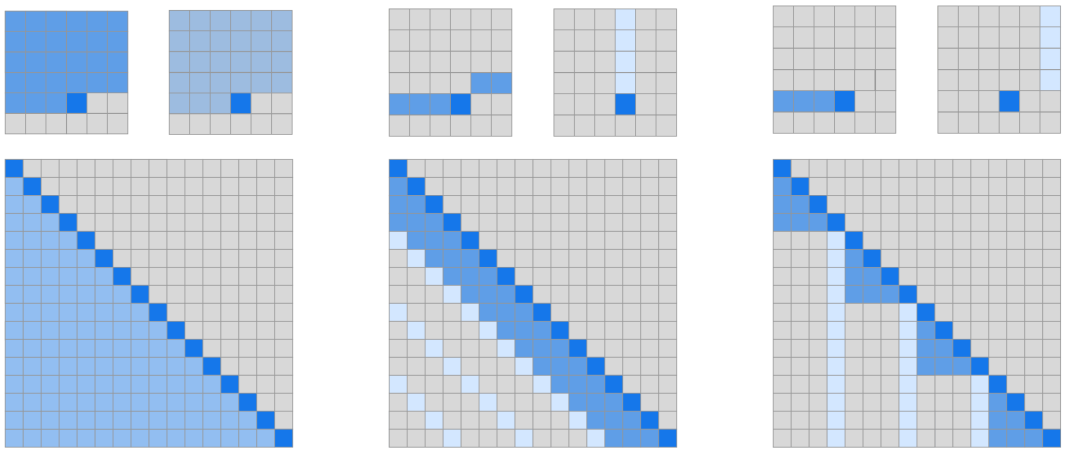


Figure 2.15: The attention range of Image Transformer. The cyan line marks the attention scope of the query q. Left: A 1-dimensional attention range, similar than for sequential data; Right: A 2-dimensional attention range. Figure from [138].



(a) Transformer (b) Sparse Transformer (strided) (c) Sparse Transformer (fixed)

Figure 2.16: The sparse attention schemes. Top rows represents in each cases two heads and their the previous positions they attend to, bottom rows are the full connectivity matrices between inputs (columns) and outputs (rows). a) is a standard Transformer, b) represents the stridden sparse attention, c) the fixed sparse attention. Figure from Child et al. [23].

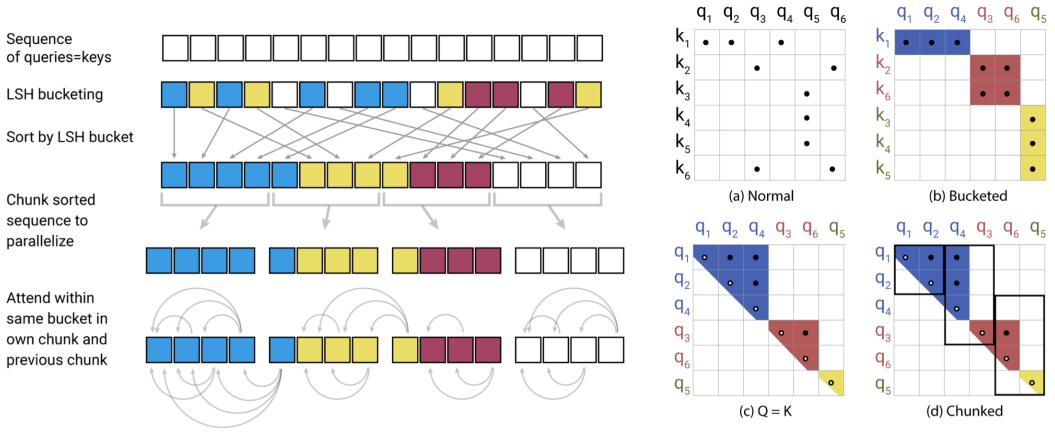


Figure 2.17: The Locality-Sensitive Hashing attention scheme. Queries are gathered into buckets and sorted before attention is applied causally within. Figure from Kitaev, Kaiser, and Levskaya [103].

In the case of stridden sparse attention, one head attends to the l last positions, and another attends to every lth positions. Formally, $S_1 = \{t\}^i_{t=\max(0,i-l)}$ and $S_2 = \{j : (i-j) \bmod l = 0\}$, where l is the stride and set close to $\sqrt{L}$, are the two sets of positions each head attend to. The authors stated that this method is well suited for continuous data like images or audio music. In contrast, it performs poorly for discrete data like text.

The fixed sparse attention is on the other hand more suited for discrete modalities. We have $S_1 = \{j : (\text{floor}(j/l) = \text{floor}(i/l))\}$ and $S_2 = \{j : j \bmod l \in \{t\}^l_{t=l-c}\}$ where c is a hyperparameter, and $c \in \{8, 16, 32\}$ for $l \in \{128, 256\}$ in the paper.

The results of the models tested by the authors on image, audio and text modeling almost reached the one of the vanilla Transformer while being very coherent, but with a much lower computation time.

Reformer and Locality-Sensitive Hashing attention

Motivated once again to increase the efficiency of Transformers, Kitaev, Kaiser, and Levskaya introduced improvements to the Transformer architecture [103], by:

- Approximating the scaled dot product attention with locality-sensitive hashing attention, reducing the complexity from $\mathcal{O}(L^2)$ to $\mathcal{O}(L \log L)$;
- Using reversible layers, enabling to store only activations of all layers only once during training, instead of one per layers required with vanilla Transformers.

Locality-sensitive hashing is an algorithmic method that hashes similar inputs together into a fixed number of buckets. This number is significantly smaller than the number of inputs, allowing to reduce the dimensionality of sets of inputs. In the case of the Reformer, attention is calculated with causality between elements of a same bucket, and the first element of a bucket attends to the whole previous bucket, as shown in Figure 2.17. This allows to decompose attention into a set of smaller attention calculations.

The hashing strategy used by the Reformer is the same as introduced by [5] [5].

The second major improvement brought by the Reformer are the reversible activation layers [64]. The main idea is to allow to recover an activation from the following layer. This way, the model saves memory by recomputing activations during back-propagation rather than storing them. A standard residual layer can be formally expressed as $y = x + \mathcal{F}(x)$,

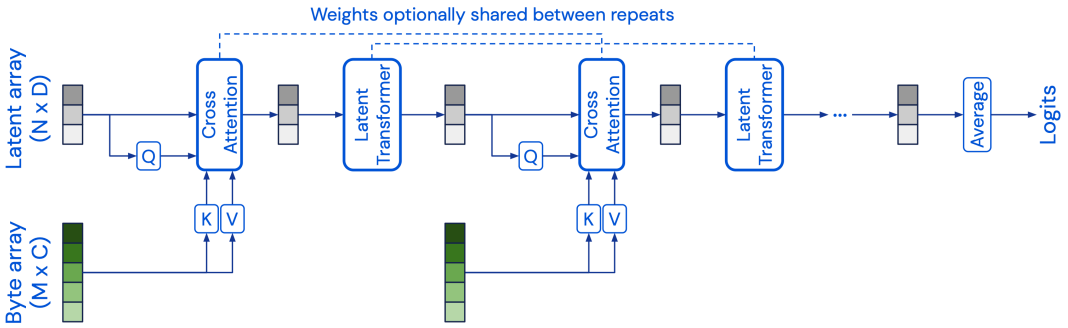


Figure 2.18: Schema of Perceiver and its iterative attention. Figure from Jaegle et al. [89].

whereas the reversible block is decomposed as in Equation (2.15) where the left side is the forward pass and the right the backward, (x_1, x_2) is a pair of inputs.

$$\begin{aligned} y_1 &= x_1 + \mathcal{F}(x_2) & x_1 &= y_1 - \mathcal{F}(x_2) \\ y_2 &= x_2 + \mathcal{G}(y_1) & x_2 &= y_2 - \mathcal{G}(y_1) \end{aligned} \tag{2.15}$$

Reformer applies this principle with $\mathcal{F}$ as the Attention layer and $\mathcal{G}$ as the feed-forward layer, without needing to store the activations. Experiments on the enwik8 and WMT English-German datasets showed that the reversible Transformer performs similarly the vanilla Transformer, but faster.

Iterative Transformer: Perceiver

The Perceiver [89] is one of the first attempts to iteratively apply attention across a set of input sequences. The architecture, shown in Figure 2.18, iteratively projects sequences of embeddings onto a latent array $\mathbf{Z} \in \mathbb{R}^{N \times d}$ via attention[5]. This method reduces drastically the complexity, $\mathcal{O}(NM)$, as $N \ll M$ for any input $\mathbf{X} \in \mathbb{R}^{M \times C}$. And the several iterations can be done using a unique attention and latent transformer (weights are shared between iterations), or several.

The first motivation for this design choices was to build a model capable to handle several modalities. The experiments in the paper modeled video and audio modalities, outperforming state of the art in some tasks or achieving competitive results.

While the Perceiver performs very well on some domains, its output capabilities are limited, as in all cases sampled from the latent bottleneck. Hence DeepMind introduced an improvement of the Perceiver architecture, namely Perceiver IO [90]. The new design, illustrated in fig. 2.19, is in fact a Seq2seq model, conceptually similar to both the original transformer and perceiver.

The encoder part is the original Perceiver model and works the same. The decoder part is a standard attention layer, for which the queries come from the sequence $\mathbf{Y}_t \in \mathbb{R}^{O \times E}$, keys and values from the latent array $\mathbf{Z}$, resulting in the output sequence $\mathbf{Y}_{t+1}$. The decoder could in fact be replaced by a full transformer layer. This design allows to sample and auto-regressively generate sequences from $\mathbf{Z}$.

The authors benchmarked Perceiver IO on several tasks including language understanding, multimodal audio-video-label auto-encoding, and show competitive results against state of the art models.

[5] Called cross-attention in the original paper as the queries comes from the latent and the keys and values from the inputs.

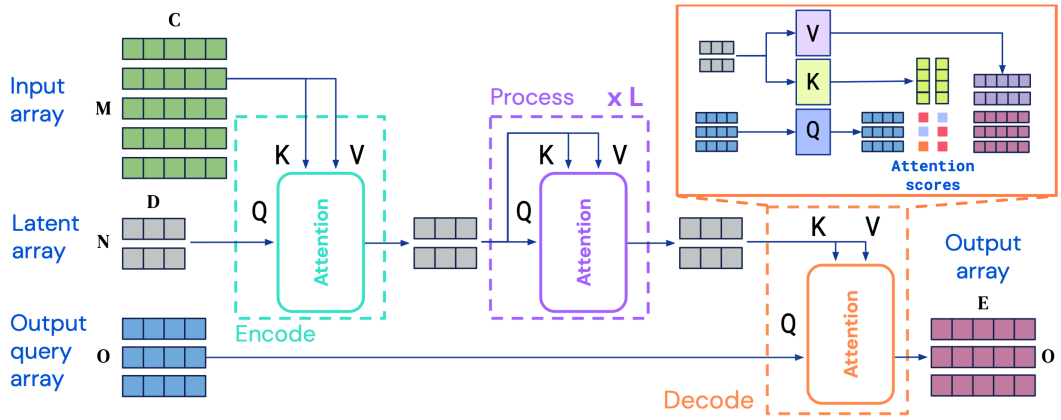


Figure 2.19: Perceiver IO, an improvement of the Perceiver architecture. The additional decoder parts allows to generate auto-regressively flexible output sequences. Figure from Jaegle et al. [90].

2.2.2 Linear transformers

Despite the great results in many applications, and hence its adoption in many domains, the "vanilla" transformer architecture suffers from a bottleneck that prevents its adoption in some cases. As we saw in the last sections, the attention operation first computes the dot product of the query and keys matrices. This imply calculating a matrix $A \in \mathbb{R}^{T \times S}$ where S and T are the target sequence and source sequence lengths respectively, resulting in a $\mathcal{O}(TS)$ time and memory complexity, or $\mathcal{O}(S^2)$ in the case of self-attention in which the complexity grows quadratically with respect to the input sequence length. When dealing with short sequences, or if memory and hardware capacity is not an issue, this can not even be a problem. In many cases however, researchers can easily fall out of memory. When processing long sequences or large data samples such as images, the memory requirement of the attention operation can make it intractable on some GPUs, even with an amount of video memory that would be considered large at time of writing. In order to alleviate this issue, researchers have since looked for ways to reduce this complexity.

The kernel trick to linearize attention

Katharopoulos et al. [97] expressed the idea of approximating the scaled dot product attention matrix using the kernel trick. The kernel trick Equation (2.16) is a method consisting in using a linear classifier to approximate a non-linear operation, dot product especially. Computing the dot product of large vectors can sometimes be intractable on some hardware. The kernel trick allows to reduce the time and memory complexity by linearly approximating the dot product, after projecting the inputs with feature maps $\phi(\cdot)$.

$$\mathrm{K}(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x})\phi(\mathbf{y}) \tag{2.16}$$

The approximation of the attention matrix can be done by computing the product of keys and values first. The overall complexity would grow linearly with the length of the input sequences. Thanks to the associativity of matrix product, and the fact that the dot product of the attention operation is applied row-wise, it can be decomposed as shown in Equation (2.17) where S is the source sequence length.

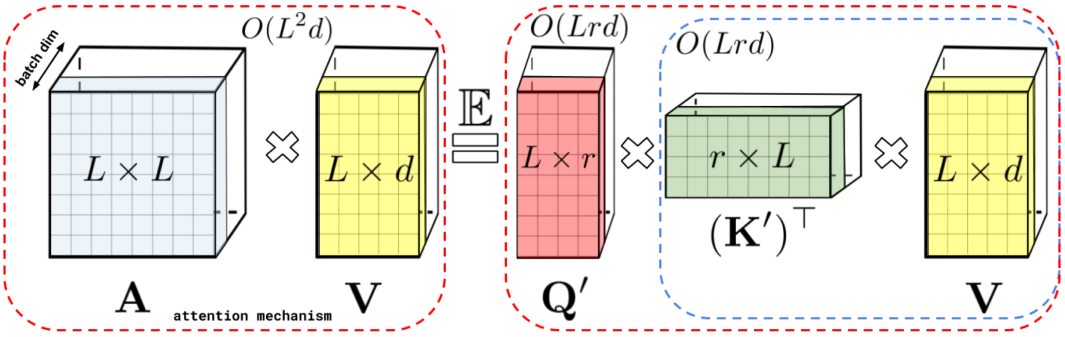


Figure 2.20: Left: the scaled dot product attention; Right: The linear attention using the kernel trick. Figure from [28].

$$\begin{aligned} \text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathbf{V}' &= \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}})\mathbf{V} \\ \mathbf{v}'_i &= \frac{\sum_{j=1}^{S} \text{sim}(\mathbf{q}_i, \mathbf{k}_j)\mathbf{v}_j}{\sum_{j=1}^{S} \text{sim}(\mathbf{q}_i, \mathbf{k}_j)} \\ \text{sim}(\mathbf{q}, \mathbf{k}) &= \exp(\frac{q^\top k}{\sqrt{d^k}}) \end{aligned} \tag{2.17}$$

Given a kernel function $\phi(x)$, we can rewrite Equation (2.17) as follows:

$$\begin{aligned} \mathbf{v}'_i &= \frac{\sum_{j=1}^{S} \phi(\mathbf{q}_i)^\top \phi(\mathbf{k}_j)\mathbf{v}_j}{\sum_{j=1}^{S} \phi(\mathbf{q}_i)^\top \phi(\mathbf{k}_j)} \\ \mathbf{v}'_i &= \frac{\phi(\mathbf{q}_i)^\top \sum_{j=1}^{S} \phi(\mathbf{k}_j)\mathbf{v}_j^\top}{\phi(\mathbf{q}_i)^\top \sum_{j=1}^{S} \phi(\mathbf{k}_j)} \end{aligned} \tag{2.18}$$

The numerator can then be rewritten as $(\phi(\mathbf{Q})\phi(\mathbf{K})^\top)\mathbf{V} = \phi(\mathbf{Q})(\phi(\mathbf{K})^\top\mathbf{V})$, and Figure 2.20 shows a visual comparison of the dot product attention, and its linear approximation with kernel trick. This formulation implies a $\mathcal{O}(TS)$ complexity, T and S being respectively the target sequence and source sequence lengths. Katharopoulos et al. based their experiments with a feature map function $\phi(x) = \text{elu}(x) + 1$.

Compared to a transformer of the same size but with the scaled dot product attention, the results of the linear transformer for a sequence duplication task are a little lower (cross entropy value) and with a slower convergence (in number of gradient steps). But considering the fraction of the memory and time complexity, the linear transformer could be easily scaled up to a lot more parameters and converges more rapidly as an inference takes less time to perform. For a same hardware configuration, a linear transformer could then be much larger than a transformer with scaled dot product attention and possibly give better results. Finally, a linear transformer can of course handle input sequences of much larger lengths, and possibly solve tasks that cannot be run on quadratic transformers.

This linear attention formulation was used in some music generation works [68, 119], showing that this linearization works for such contexts.

Favor+ and Performers

Following this first approach, Choromanski et al. discussed the way to approximate the softmax attention [28] and introduced a new method they called Favor+ (for *Fast Attention Via positive Orthogonal Random features*).

The Favor+ makes use of random, positive and orthogonal features to approximate the attention matrix. Although the use of positive random and orthogonal random features for most kernels are well known methods, Choromanski et al. showed that the use of both works particularly well for the softmax kernel.

The communicated results showed that Performer (transformer with favor+ attention) performed equally, sometimes better, than quadratic transformers on tasks such as protein sequences modeling. More importantly, performers could be trained on large datasets with consequent input lengths while keeping a good quantity of parameters where quadratic transformers were limited, and so was their accuracies.

A similar approach was introduced by Peng et al. [141]. Using random features and normalizing keys and queries, they communicate outperforming results with their approach over Favor+. However their experimental setup, proofs, article and no code shared leaves us sceptic.

Benchmarks of linear attention

A few researchers benchmarked the previous attention linearization strategies on various tasks [166], in order to measure how well they can approximate the scaled dot product in objective environments. The most popular benchmark at time of writing is the Long Range Arena [174].

The Long Range Arena measures the performance, memory footprint and capabilities of efficient transformers. It is built around key ideas such as simplicity, challenge, inputs length capability. The models are benchmarked on several tasks such as byte-level text classification, byte-level document retrieval, image classification or Pathfinder [117].

These benchmarks show interesting analysis. Kernel based transformers seems to not perform very well on hierarchical tasks such as ListOps, but performs really good on text and images. In terms of performances, we see an improvement in both time and memory, which is more noticeable when the sequence length is high. Analyzing the other models also shows us that the Reformer and its locality-sensitive hashing attention is more than two times slower than the vanilla Transformer, while being two times more effective in memory consumption. The Performer [28] seems to be the a.t.o.w. the fastest model, and Linformer [179] the one with the lowest memory consumption.

FlashAttention

FlashAttention [35] is an efficient attention computation process leveraging the GPU's memory specifications to reduce the number of reads/writes between its global memory (VRAM) and cache memory (SRAM). This is done by "tiling" the computation of the attention matrix: blocks of attention computations are loaded into fast SRAM, the results are written into VRAM, resulting in about a 7.6 times speedup of the complete attention computation, and up to 3 times faster training on models such as GPT2 [146] or BERT [37]. The memory usage is also decreased, allowing to use longer context length (token sequence length). FlashAttention is now natively integrated in the most popular DL frameworks and libraries, such as PyTorch [139], Hugging Face Transformers [183] or DeepSpeed [155].

2.3 Generative architectures

So far, we have explored the operations and modules that compose modern DL models, and focused on the Transformer architecture. In this section, we specifically focus on the architectures that allows to generate content. Even though these architectures are mostly applied to generate continuous modalities such as images or audio, we still present them here for the reader's benefit.

These architectures are conceived to output results on a joint distribution, i.e. are more suited for continuous modalities. DL models for text and symbolic music generations are now usually based on the Transformer architecture, to generate tokens autoregressively.

As symbolic music can contains both discrete and continuous features, previous works still successfully implemented some of the following architectures for symbolic music generation, while benefiting from their distinct controllability advantages.

2.3.1 Auto-encoders

An auto-encoder is a model architecture, based on en encoder and a decoder, often being having the same architecture but mirrored, trained to generate content from a low-dimensional space representation. The model is trained by feeding input data to the encoder, that outputs a latent vector. The latent vector is fed to the decoder, and the loss is computed on the ability of the model to reconstruct the original input, often by mean squared error (MSE) or binary cross-entropy. The encoder is then essentially trained to project the input into a lower dimensional latent space that will capture the most important features or patterns in the input data, while the decoder will generate content from the features of this space. By learning to compress and reconstruct the input data, auto-encoders effectively learn a compact representation that captures the most salient features of the data.

Auto-encoders have been used for many applications such as data compression, dimensionality reduction, denoising, anomaly detection, and content generation. For the latter, the latent space can be explored in order to generate content from the decoder. This has however a few shortcomings:

- Limited interpretability: The latent space is often not easily interpretable, it is often difficult to identify the distinct features it represent;
- Overfitting: auto-encoders are prone to overfitting, especially with large models and / or small datasets;
- Dependency on reconstruction error: Auto-encoders rely on the reconstruction error between the input and output data as their training objective. While this can be effective in capturing global patterns, it may not necessarily capture fine-grained details or subtle variations in the data;
- Discrete nature of the latent space: the latent space is expressed as a 1-dimensional vector $\mathbf{z} \in \mathbb{R}^d$, for which each feature is expressed as discrete, thus limiting the range of expressiveness for the decoder.

The last point in particular, gave birth of the Variational Auto-Encoder (VAE) [100]. The key idea behind VAEs is to learn a probability distribution in the latent space rather than just learning a fixed encoding. This is achieved by introducing a probabilistic component into the latent space. Instead of mapping the input directly to a fixed point in the latent space, the encoder network outputs the parameters of a probability distribution,

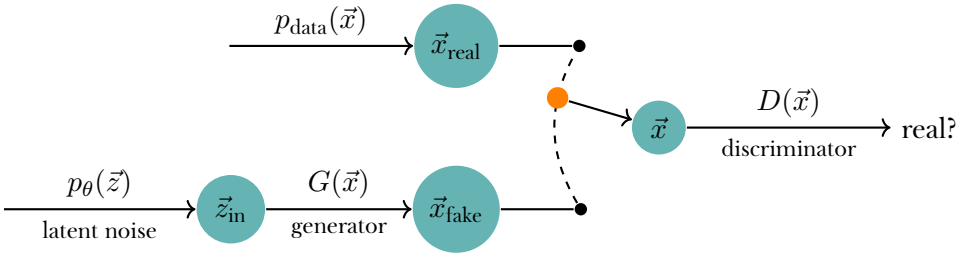


Figure 2.21: Schema of a GAN.

typically a multivariate Gaussian distribution, that represents the uncertainty in the latent representation.

A VAE is trained similarly to any auto-encoder. Additionally, it incorporates a regularization term that encourages the learned distribution in the latent space to be close to a predefined prior distribution, usually a standard Gaussian.

The VAE can generate new samples by sampling from the learned latent space distribution and passing the samples through the decoder network. By sampling different points in the latent space, the VAE can generate diverse and novel outputs that capture the underlying structure of the training data.

Note that as the sampling operation is not differentiable, gradient backpropagation is assured by the reparametrization trick [102].

VAEs learn a continuous latent space representation, where similar points in the latent space correspond to similar data points in the input space. This continuous nature allows for smooth interpolation between different data points and facilitates meaningful exploration and manipulation of the learned representations. The architecture has been used to create many generative models, for image [136] and of course music [160, 197, 95].

2.3.2 GANs

Generative Adversarial Networks are a class of models featuring two parts: a generator g_θ which generate data from random noise, and a discriminator d_δ which receives data and predicts if it generated or from the real data distribution. Both parts are trained adversarially, with the generator trained to fool the discriminator, and the latter to better distinguish what the generator is predicting from the real data. The architecture is depicted in Figure 2.21.

During training, when the discriminator fails, its parameters δ are updated. It is trained to maximize it's result over real data $\max \mathbb{E}\_\mathbf{x} \sim p_x(\mathbf{x})[log\,(d_\delta(\mathbf{x}))]$, while minimizing the prediction of generated samples $g_\theta(\mathbf{z})$, $\min \mathbb{E}_{\mathbf{z}\sim p_z(\mathbf{z})}[log\,(1 - d_\delta(g_\theta(\mathbf{z})))]$.

When the discriminator succeeds, the generator's parameters θ are updated. It is trained to generate samples which maximize their result from the discriminator (fools it), hence minimizing $\mathbb{E}_{\mathbf{z}\sim p_z(\mathbf{z})}[log\,(1 - d_\delta(g_\theta(\mathbf{z})))]$.

This can in turn be written as:

$$\min_\theta \max_\delta \mathcal{L}(g_\theta, d_\delta) = \mathbb{E}\_x \sim p_x(\mathbf{x})[\log d_\delta(x)] + \mathbb{E}_{z\sim p_z(\mathbf{z})}[\log(1 - d_\delta(g_\theta(\mathbf{z})))]$$

The generator will progressively produce content closer to p_d, and the discriminator will distinguish real from fake data with smaller differences. Both models are jointly trained, improving each other.

You see that the generator is directly trained to minimize the distance between its distribution and the data distribution: $\min_\theta \mathrm{KL}(p_\theta||p_d)$. This allows to generate original content which contains the underlying features of the data distribution **implicitly**.

GANs have largely been applied to image generation, yielding impressive results, but also audio, such as StyleGan [96] and GANSynth [47].

Limitations of GANs

GANs are, however, not except of problems and drawbacks. Their training procedure is less stable than those of counterpart models, and they do not provide an easy direct control over the generation.

GANs are known to be hard to train, easily falling into **mode collapse**: during training, as the generator learns to produce samples close to p_x which tricks the discriminator most times, it will likely learn to predict only this specific sample. This is undesirable as we should expect the generator to predict plausible and various results.

GANs can also suffer from vanishing gradients. If the discriminator begins to converge and to be optimal, the gradients computed during the backpropagation will naturally have low values before reaching the generator's parameters, resulting in weak updates. Learning then starts to slow, and can stop. These issues can be alleviated using Wasserstein distance as loss function [8].

Control techniques

Since then, techniques have been developed to control GANs from high level attributes [170, 96], with a feature classifier [113] or even from text captions and contrastive learning with VQGAN + CLIP [6].

Efficient sampling

GAN models were originally presented as sampling from the latent space of the generator to produce content, leaving the discriminator after training. The discriminator however contains useful information on the data distribution that it would be wasteful not to take advantage of, as the common generation practice is based on rejecting unsatisfying samples. Discriminator Driven Latent Sampling (DDLS) [21] proposes to use the discriminator at inference to efficiently sample from the latent space, by defining the generator and discriminator as an energy-based model and running Markov Chain Monte Carlo, yielding results of higher quality and more efficiently.

Ansari et al. proposed a improved and more general framework of sample refinement strategy they called Discriminator Gradient *f*low (DG*f*low) [6]. The main idea is to iteratively refine the latent sample $\mathbf{z}$ by computing the gradient of the f-divergence loss of $d_\delta(g_\theta(\mathbf{z}))$. In other words, we use the generator to generate from $\mathbf{z}$, assess the result with the discriminator, compute gradients from the score and backpropagate them to $\mathbf{z}$ to refine it. This can be iteratively done until the $g_\theta(\mathbf{z})$ is satisfying enough. DG*f*low can be applied to any latent sampling model like GANs, VAE, Normalizing Flows, and its main principle is similar to the how Diffusion models work (Subsection 2.3.3).

Application to discrete data

GANs have essentially been applied to continuous domains so far. The reason is because the output of the generator $\mathbf{y}$ is the input of the discriminator, thus every operation in between must be differentiable. Sequential models however return discrete distributions, from which we would have to sample $\mathbf{y} \sim g_\theta(\mathbf{x})$ in order to estimate the result and pass it to the discriminator. But sampling from a distribution is not a deterministic operation, hence

[6] https://github.com/nerdyrodent/VQGAN-CLIP

not differentiable. This means that during training, the gradients cannot be backpropagated to the generator.

Even though some attempts have been made, and successively created GANs for text with tricks such as Gumbel-Softmax [121, 91], the results are always not competitive with standard autoregressive models [109, 134, 193].

2.3.3 Diffusion models

Diffusion models [77] are a type of generative model designed to model the probability distribution of high-dimensional data. The core idea behind diffusion models is to model the process of gradual diffusion or spreading of information over time. During training, in each diffusion step, the model adds noise to the input data, and is then used backward to denoise the perturbations just added. The loss is computed on its denoising ability [172]. During inference, the model iteratively refines a high dimensional latent noise distribution, until it becomes synthetic data.

In practice during training, a data sample $\mathbf{x}_0$ from the dataset is iteratively added noise in T steps, giving noisy samples $\mathbf{x}_1, \dots, \mathbf{x}_T$. The forward pass of the model p_θ is described in Equation (2.19), with $\beta_t \in [0, 1]$ being a variance schedule over the steps.

$$p_\theta(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}) \quad p_\theta(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}) \tag{2.19}$$

$\mathbf{x}_0$ gradually becomes noise step after step. Reversing this process, i.e. calculating $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ allows us to remove the noise just added.

$$p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) \quad p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \mathbf{\Sigma}_\theta(\mathbf{x}_t, t)) \tag{2.20}$$

The training and inference processes are described in Algorithm 1 and Algorithm 2 (recopied from Ho, Jain, and Abbeel [77]).

Algorithm 1 Diffusion training

1: **repeat**
2: $\mathbf{x} \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, ..., T\})$
4: $\epsilon \sim \mathcal{N}(0, \mathbf{I})$
5: Take gradient descent step on
6: $\nabla_\theta \left\| \epsilon - \epsilon_\theta(\sqrt{\overline{\alpha_t}}\mathbf{x}_0 + \sqrt{1-\overline{\alpha_t}}\epsilon, t) \right\|^2$
7: **until** Convergence

Algorithm 2 Diffusion inference

1: $\mathbf{x}_T \sim \mathcal{N}(0, \mathbf{I})$
2: **for** $t = T, ..., 1$ **do**
3: $z \sim \mathcal{N}(0, \mathbf{I})$ if $t > 1$ else $z = 0$
4: $\mathbf{x}_{t-1} = \frac{\sqrt{\alpha_t}}{1} \left(\mathbf{x}_t - \frac{\sqrt{1-\overline{\alpha_t}}}{1-\alpha_t} \epsilon_\theta(\mathbf{x}_t, t) \right)$
5: **end for**
6: **return** $\mathbf{x}_0$

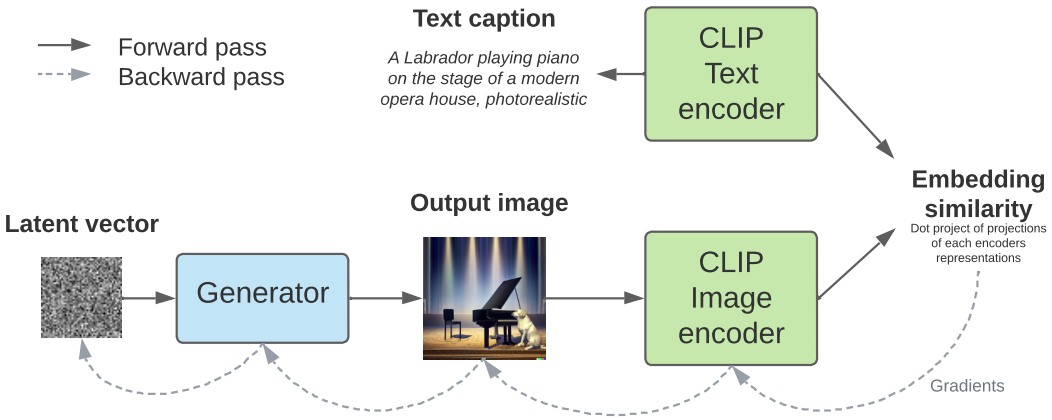


Figure 2.22: Schema of the framework of generation with classifier guidance.

The inference process of diffusion models make them perfectly suited for generation tasks. Contrarily to GANs, VAEs or Transformers, the output result is refined to directly model the data distribution, making it a "generation" only class of model.

To control such type of model, one can compute the loss according to the desired output during inference. This can be done by using a classifier model estimating the alignment between an image $\mathbf{x}$ and a text description $\mathbf{y}$. The idea have been applied to GANs in the very popular notebooks VQGAN+CLIP [32] and BigSleep [132]. They used respectively the VQGAN [51] and BigGAN [19] models as generator, and the CLIP [147] model as guidance model. The overall process is depicted in Figure 2.22.

This classifier-guidance process comes however with a few shortcomings. Most importantly, as diffusion models iteratively denoise inputs, a classifier used to guide the generation needs to be able to cope with high noise levels in order to give a reliable feedback. In practice, the classifier is trained specifically for the guidance purpose. Even with a strong classifier, this approach is limited by the fact that $\mathbf{x}_t$ is not relevant to predict $\mathbf{y}$. Consequently, using the gradient from the classifier might direct into non-relevant directions, and fail to produce the expected result.

These limitations are now erased with a method called classifier-free guidance [79]. As the name implies, no external classifier is used in order to compute the alignment between $\mathbf{x}_t$ and $\mathbf{y}$. Here, the diffusion model is trained with both a conditional $p_\theta(\mathbf{x})$ and a conditional $p_\theta(\mathbf{x}|\mathbf{y})$ objectives. The conditional objective is parametrized with a score estimator $\epsilon_\theta(\mathbf{x}, \mathbf{y})$. During training, a linear combination of both scores is done $(1+w)\epsilon_\theta(\mathbf{z}_\lambda, \mathbf{c}) - w\epsilon_\theta(\mathbf{z}_\lambda)$ where w controls the influence of the guidance. This technique presents the advantage of relying on the diffusion model's knowledge, thus assuring a robust alignment score even with a noisy $\mathbf{x}$. While standard a classifier can ignore most of the feature of $\mathbf{x}$ and still yield competitive results, a conditional generator cannot take such shortcuts. This makes the resulting gradients more reliable, while having to only train a single model.

The classifier-guidance process has been applied to diffusion models with GLIDE [133], DallE-2 [152], Imagen [165], and others [161, 78], to finally become standard.

Diffusion models have gained a lot of attention due to their ability to generate high-quality samples with a wide range of applications. They have been used to create realistic images, complete missing parts of images, and generate new text samples that resemble a given text corpus. Overall, diffusion models provide a powerful framework for generative modeling by capturing complex data distributions.

Chapter 3

Music generation with Deep Learning

Music can be represented in several ways. These representations have been developed to fulfill the needs of musicians to create, compose and analyze music.

We can divide these representations in two major categories: symbolic and audio. Symbolic music, as its name suggests, represents music by its symbolic content i.e. the notes, rests, tempos and all their attributes. It is mainly aimed for musicians to read and write music and relies heavily on music theory. Audio corresponds to the physical representation of music, when being played, i.e. the waveform signal produced by an instrument or a singer, traveling in the air or any material. This can be represented by a waveform, i.e. the amplitude of the sound as a function of time, or a spectrogram.

Audio and symbolic representation, although they can represent the same music, are very different in their format, the information they contain and how to retrieve it. Symbolic music obviously is easier to manipulate in a "creative" context where one arrange notes to form a melody, and voices to create harmony. It lacks however all the information carried by the sound. Music genres such as Electronic, Pop or Rap heavily rely on the use of synthesizers offering infinite possibilities of sound creations. The process to create these sounds has been the theme of many dedicated book [182]. And we could not pass on the evident and essential role of the mixing and mastering steps of any music creation. Even though these are steps which often consist to embellish already made up melodies or arrangements, these can be considered as artistic as they transform sounds with effects, creating new mixtures which considerably modify the way we appreciate music. Lastly, nowadays many musicians create music using samples which are for the most part audio files. DAWs allow to easily manipulate and use both symbolic and audio contents for any creative purpose. Most musicians and producers uses both formats in the music released.

In this chapter, we first introduce the different music format, then proceed with the previous works on music generation with DL. It will not be exhaustive, so for a more comprehensive reading we refer the reader to [18] which present most DL models for music generation, and [93] present does too but with a approach oriented towards DL techniques as we will have here.

3.1 Traditional music representations: score and piano roll

The way music is represented evolved with centuries, towards formats widely spreads and understood allowing musicians to read and write music like one would do with text. The sheet music is probably what comes first to your mind when thinking about music. Several systems were developed by different cultures during history, which were progressively shadowed by the western music notation. A sheet is composed of staves on which are written notes and bars, to be read from left to right and up to down. The position, form and color of a note allows a reader to determine its pitch and duration, see Figure 3.1.

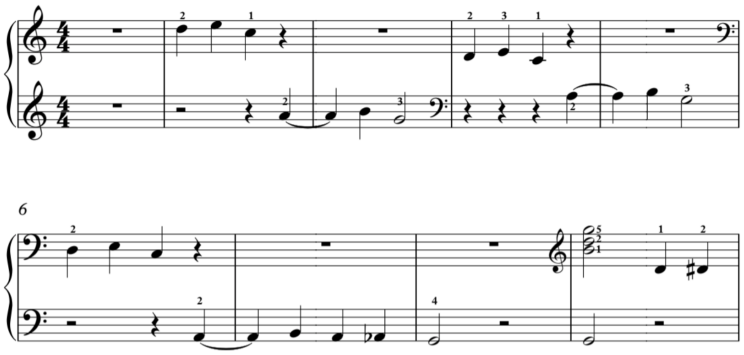


Figure 3.1: A music sheet with two staves of nine bars.

The piano roll was originally a recording medium for operating a mechanical piano, see Figure 3.2. By rotating the roller at a defined speed, the holes punched in it at different positions will automatically play the piano at the desired notes.

Although the mechanical original piano roll format is rarely used nowadays, the piano roll name is still used to describe the visual representation of notes along time that is commonly used in most DAWs as shown in Figure 3.3. The horizontal axis corresponds to the time, the vertical to the pitch of the notes. This offers an appealing and flexible visual representation. Notes can be displayed with visual elements indicating an attribute, like the color representing the velocity[1] in Logic Pro X. Digital supports allow to store and represent music, which we introduce in the next section.

3.2 Symbolic music digital files

3.2.1 MIDI

MIDI, acronym for Musical Instrument Digital Interface, is a technical standard that describes a communication protocol, an electrical connector and a digital file format. It first appeared in the early eighties, when digital instrument manufacturers needed a digital protocol for communication between devices such as synthesizers and computers. It was standardized in 1983 by the first specifications, and is currently maintained by the MIDI Manufacturers Association[2]. Meanwhile new specifications were made, the two major ones and still the norm today being the General MIDI 1 (GM1) and General MIDI 2[3]. These specifications aim to guide the manufacturers to design digital music devices compatible with the ones from other manufacturers.

The MIDI protocol is event based. It consists of a series of events, which can occur in multiple channels. Each event is composed of two key information, 1) the delta time expressed, which is the distance in ticks with the previous event (in the same channel) and so represents its position in time, 2) a message which represents its content.

The tick is the basic time unit of MIDI. It is a clock signal which its frequency is set by the time division[4], which itself can be express in ticks per quarter note or in ticks per second, the most popular choice being the first as it bases itself on musical time units and is tempo independent.

[1] The term velocity comes from the MIDI format (to be introduced in next paragraph). It indicates the force with which a note is played, influencing its volume and dynamism.

[2] MIDI Manufacturers Association website: https://www.midi.org

[3] GM specifications: https://www.midi.org/specifications

[4] Also sometimes called resolution.

Figure 3.2: A mechanical piano playing from a piano roll.

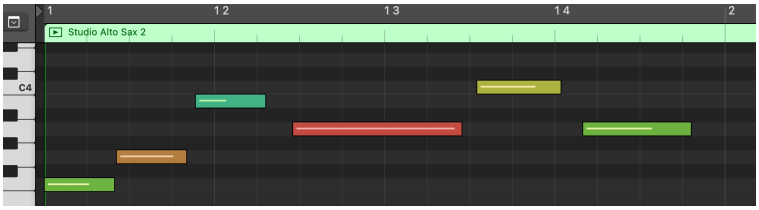


Figure 3.3: A piano roll view in the Logic Pro X DAW.

11000101 - 00111001
Program Change, Channel 5 – Instrument 57 (Trumpet)

10010101 - 00111100 - 00110010
Note On, Channel 5 – Pitch 60 – Velocity 50

10000101 - 00111100 - 00000000
Note Off, Channel 5 – Pitch 60 – Velocity 0

Figure 3.4: MIDI message examples, binary values and their significations.

The message express an event or an information. A message is a series of bytes, the first is the Status byte which specify the type of message and the channel, followed by one or two data bytes which contain the information. Figure 3.4 shows what a sequence of MIDI messages can look like. All the messages and their significations are described in the GM1 and GM2 specifications. For our purposes, the most important and relevant ones which we will parse to extract the musical information are:

- Note On: a note is being played, specifies its pitch and velocity;
- Note Off: a note is released, specifies the note to stop and the velocity;
- Program Change: specify the instrument to play;
- Control Change: a control parameter is modified or applied. The modulation wheel, foot sustain pedal, volume control or bank select are for instance effects transcribed into Control Change messages.

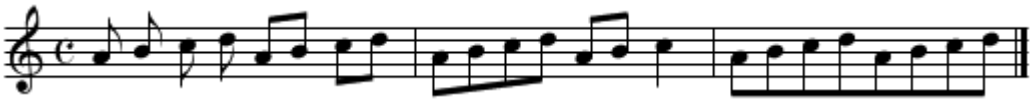
Note that these messages are "voice messages", which means that each of them is applied within a channel that is specified in its status byte. The MIDI protocol handles up to sixteen channels which allows to connect multiple instruments that are playing and communicating all together. The channel 10 is reserved for drums, which is a specific "program" in which the pitch values corresponds to drum sounds like kicks, snares, or hi-hats.

The latest evolution of the MIDI protocol is the MIDI Polyphonic Expression (often shortly called MPE). This new norm allows manufacturers to create MIDI devices on which a specific channel is assigned to each note allowing the user to apply pitch bend and modulation on each key independently. These devices are typically built with touch-sensitive keys. The MIDI Manufacturers Association released the complete specifications on March 2018[5].

3.2.2 ABC

The ABC notation is a notation for symbolic music, and a file format with the extension `.abc`. Its simplicity has made it widely used to write and share traditional and folk tunes from Western Europe. Each tune begins with a few lines indicating its title, time signature, default note length, key and others. Lines following the key represent the notes. A note is indicated by its letter, followed by a `/x` or `x` to respectively divide or multiply its length by $x \in \mathbb{N}^\star$ compared to the default note length. Figure 3.5 show an example of a lead sheet and its ABC notation. An upper case (e.g., A) means a pitch one octave below than a lower case (a).

[5] MPE specifications: https://www.midi.org/midi-articles/midi-polyphonic-expression-mpe



```
X:1
T:Beams
M:C
K:C
A B c d AB cd | ABcd ABc2 | ABcdABcd |
```

Figure 3.5: An example of the ABC notation, taken from abcnotation.com: above is a lead sheet, below its corresponding ABC notation.

This notation is originally designed for writing monophonic melodies, as it cannot represent polyphony outside of specific chord symbols.

3.2.3 MusicXML

MusicXML is an open file format and music notation. Inspired by the XML file format, it is structured with the same item-hierarchy. Figure 3.6 shows an example. The `part-list` references the parts to be written following with the tag `part`. A `measure` is defined with its attributes, followed by notes and their attributes. The common file extensions are `.mxl` and `.musicxml`.

3.3 Audio representations

The two common audio format, used in audio models, that we will present in this sections are waveform and spectrogram.

3.3.1 Waveform

A waveform represents the signal (its amplitudes) of a sound i.e. vibrations propagating in a material as a function of time. Figure 3.7 shows a waveform.

A sound as in Figure 3.7, wether from an instrument, a human voice or music arrangement, is a superposition of many periodic waveforms, defined by their wavelength λ, amplitude α and phase ϕ. The frequency f of a periodic waveform is defined by $f = \frac{c}{\lambda} = \frac{1}{T}$ where c is the celerity and T its period.

Using waveforms with statistical models poses serious challenges. To represent a sound as a waveform with the fewest loss, the Nyquist-Shannon theorem establish that the sample rate, expressed in samples/sec, must be at least twice the highest frequency contained in this sound. As humans can hear frequencies up to 20k Hz, a sample rate for audio should be higher than 40k samples/sec[6]. A model processing raw waveform signals would imply it could be capable to handle $n = L \times r$ continuous values where L is the length of the input in seconds and r the sample rate. For instance a 5 seconds audio input, which can be considered short, built with a sample rate of 44.1k samples/sec, contains 220,500 samples, which can be considered very high for a DL model a.t.o.w. A DL model handling raw audio waveforms is in practice possible [38, 135, 33], but a more common practice consists in extracting features to be used by DL models from audio data. These features are

[6] A very common rate, used for instance on CDs, is 44100 samples/sec.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE score-partwise PUBLIC
    "-//Recordare//DTD MusicXML 3.1 Partwise//EN"
    "http://www.musicxml.org/dtds/partwise.dtd">
<score-partwise version="3.1">
   <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 3.6: A MusicXML code example, representing one measure containing a C4 note.

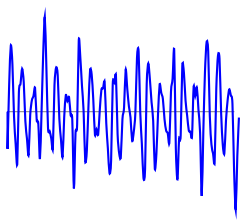


Figure 3.7: The waveform signal of a sound, as the amplitude as a function of the time.

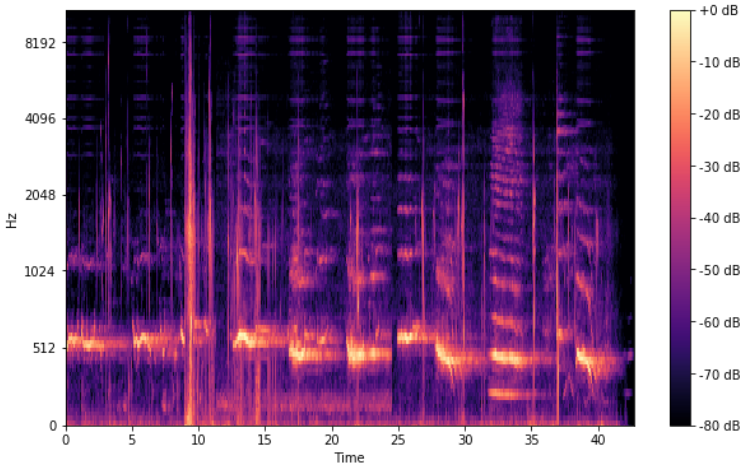


Figure 3.8: The spectrogram of a sound, abscissa is time, ordinate is frequency and the color represents the intensity.

mathematical representations of higher level characteristics of audio, that can be easily used with DL learning models for a wide variety of tasks including speech recognition, source separation and many Music Information Retrieval (MIR) related problems. Depending on the aimed task, these features can have different levels of abstraction and meanings. High levels of abstraction are features understood and manipulated by humans, including rhythm, key, melody and harmony. Mid-level are features we can perceive such as pitch and note onsets. Low-level are statistical features that only make sense to the machine.

Features can be extracted either from trained ML and DL models or statistical and ruled-based models.

3.3.2 Spectrogram

A spectrogram is a visual representation of the frequencies of an audio signal. It depicts the intensity in dB of frequencies through the time. Figure 3.8 shows a spectrogram. It can be calculated with a Short Time Fourier Transform [67], using popular software libraries such as TorchAudio [191] or Librosa [124].

Spectrograms are used in many MIR tasks, and are well suited for CNNs as they are continuous and analogous to monochrome images with foreign patterns [112, 76].

We introduce in the next sections DL models designed to generate music.

3.4 Symbolic music generation with discrete models

Discrete models refer in this thesis to RNNs (Subsection 2.1.5) and Transformers (Subsection 2.1.6) which were first designed for natural language tasks. They process sequence of discrete elements, and output disjoint distributions conditioned on the all or some parts of the input sequence elements.

One of the first RNN-based model for music generation model is FolkRNN [173], a simple RNN trained on folk songs. It is followed by DeepBach [69], a bidirectional LSTM model trained on Bach chorales to perform inpainting with Gibbs sampling. For each time step, four pitch values are represented, one for each voice. This very efficient representation is well suited for Bach chorales and shows pleasing results as each voice plays a note almost all the time.

A more versatile method to tokenize symbolic music is introduced around the same time is the MIDI-Like encoding [137], which simply convert MIDI messages into tokens following the same order. It shows good result and proved that this method works well for expressive music generation from sequential models. MIDI-Like has also been used in Music Transformer [85], and other works on music transcription [72, 62]. It however presents a downside: some `NoteOn` tokens can be predicted without their corresponding `NoteOff` token in the next steps, leading to either notes which does not end (or too long after) or discarded notes, or notes with a default duration depending on the way to deal with this issue.

To address this, the authors of the Pop Music Transformer [87] introduced the REMI encoding, standing for Revamped MIDI. The key difference is a modified time representation. The `NoteOff` and `TimeShift` tokens are replaced with `Duration`, `Bar` and `Position` tokens indicating respectively the duration of a note, when a new bar is beginning and the position within a bar. This representation solves the previous problem with note durations, and the presence of these new time-related tokens marks explicitly the positions of notes, for instance at the beginning of a beat, allowing a model to very easily learn repetitive and common music patterns. REMI also proposed to include tokens to specify ongoing chords (and their quality if known) and tempo tokens.

Hadjeres et al. introduced the Structured tokenization with the Piano Inpainting Application [68] with the main purpose of sorting tokens types with the same recurring pattern: `Pitch`, `Velocity`, `Duration`, `TimeShift`. This way, a model efficiently learns to predict the good token types. A downside would however be that this strategy cannot represent two following notes with a distance higher than the maximum `TimeShift` value of the chosen vocabulary. In other words it cannot represent the time separating two notes if it is longer than the maximum time shift value. Therefore this strategy is mostly suited for single track and continuous performances, in the case of the original paper piano.

MMM [48], standing for Multitrack Music Machine is a multitrack transformer designed for bar-infilling. It represents multiple tracks by concatenating their token sequences, and is trained to generate the missing part of a section marked to be filled in the input.

These previous tokenizations worked well in their conditions, but all faced a major challenge inherently to the sequence lengths: the quadratic complexity of transformer networks, which is discussed in Subsection 2.2.1. To address this challenge, Hsiao et al. [83] used the associative properties of embeddings to group associated tokens together to reduce the sequence length. They called their representation Compound Word, using the same tokens than REMI.

Every token x_t is associated to a type $k \in \mathcal{K}$, for instance `Pitch`. The types are divided into two families: Note which comprise `Pitch`, `Velocity` and `Duration` tokens, and Metric which comprise `Position`, `Chord` and `Tempo` tokens. Tokens are first transformed into embeddings independently, of sizes that can vary depending on the token type[7]. The associated tokens from the same family (such as from the attributes of a distinct note) and special *Ignore* tokens for every other "non-active" types, are concatenated together into a single fixed size vector $\mathbf{c}_t$ which is then multiplied to a learned projection matrix[8] $\mathbf{W}_{in}$ resulting in a final embedding $\mathbf{e}_t$ as described in Equation (3.1).

$$\begin{aligned} \mathbf{p}_{t,k} &= \mathrm{Emb}_{\mathrm{k}}(x_t) \mid k \in [\![1, K]\!] \\ \mathbf{c}_t &= \mathrm{Concat}(\{\mathbf{p}_{t,k}\}_{k=1}^{K}) \\ \mathbf{e}_t &= \mathbf{W}_{in}\mathbf{c}_t + \mathbf{b}_{in},\ \mathbf{b}_{in} \text{ are biases} \end{aligned} \tag{3.1}$$

[7] The authors decided to transform tokens of a larger types, for instance `Pitch` which counts 88 tokens, into embeddings of larger sizes to make them more distinguishable.

[8] No mention is done about biases, but this could be added to the operation like in common linear layers.

At decoding, their model predicts first the family of the next token, then the tokens of types $\mathcal{K}$, as described in Equation (3.2) where H_i refers to the hidden state produced by the i-st layer, l being the number of layers, $\mathbf{w}_f$ and $\{\mathbf{w}_k\}_{k=1}^{K}$ being the parameters of the *K+1* output modules, and f_t the predicted family of the time step t.

$$\begin{aligned} \mathbf{h}_{l,t}^{'} &= \text{Self-attention}(\mathbf{H}_{l-1})_t \\ f_t &= \text{Sample}_\text{f}(\text{Softmax}(\mathbf{w}_f \mathbf{h}_{l,t}^{'}) \\ \mathbf{h}_t^{out} &= \mathbf{w}_{out}[\mathbf{h}_{l,t}^{'} \oplus \text{Emb}_\text{k}(f_t)] \\ y_{t,k} &= \text{Sample}_\text{k}(\text{Softmax}(\mathbf{w}_k \mathbf{h}_t^{out})) \end{aligned} \tag{3.2}$$

During training, the losses of each token type are independently computed from each output layer, then summed to compute the gradients and update the model's parameters. However, this approach presents the downside of sampling from multiple distributions independently from one another during generation, which can add variance and non-relevant results if the perplexity of the distributions are high.

Introduced with MusicBERT [194], Octuple also aggregates tokens together to form merged embeddings, reducing the lengths of the processed sequences. Octuple is first designed to be used in a multitrack context. Each embedding is the aggregation of eight sub token embeddings: *Pitch*, *Velocity*, *Duration*, *Track*, current *Bar*, current *Position*, *Tempo* and *Time Signature*. The pooling strategy is the same than with Compound Word, expect that there is no motion here of the lengths of the embeddings of tokens depending on their types, although it could be easily done. The *Bar* and *Position* embeddings can act as a positional encoding, but the authors still applied a position-wise positional encoding afterward. The decoding strategy is here more simple as they used eight classifiers to predict each token type for a time step. Though they did not share the details of how they built their classifiers, we can assume that there are stacked linear and softmax layers.

Presented with the PopMAG model [158], the MuMIDI encoding is another multitrack MIDI encoding grouping tokens into compound embeddings. Here only note attribute tokens (*Pitch*, *Velocity* and *Duration*) are aggregated. *Bar*, *Position* and *Chord* tokens indicate when these respective events happens in the sequence. A *Track* token indicates the instrument that plays the next predicted notes. And embeddings of the tempo, current bar and current position are associated to every embedding the sequence, creating a position and bar wise positional encoding. In the PopMAG paper, authors distinguishes pitches of drums from pitches of other instruments with dedicated tokens. They used three linear layers to predict the six possible token types at each time step.

The latest models, such as FIGARO [164], MTT [43] or MMM [48] put a special focus on the multitrack capacity of models and their controllability.

3.5 Symbolic music generation with continuous models

Although symbolic music is more intuitively seen as a discrete modality, it has also been used with continuous and non-autoregressive models.

As symbolic music can be represented as a pianoroll, which is essentially a 2-dimensional matrix with a time and pitch axis, it can be used with operations such as CNN (Subsection 2.1.3). In practice though, as convolution works with filters applied on local areas and the active notes are not directly adjacent in the pianoroll, the filter size has to be carefully chosen.

This approach has been used in MuseGAN [44] and Coconet [84]. MuseGan is one of the first research work to generate multi-track symbolic music from a DL model. It is a

multi-level GAN architecture (Subsection 2.3.2) made of CNN layers, which iteratively generate bars of music as pianorolls matrices. Several levels of noises are used for the input: one acting as a global composer and shared for all tracks and bars, one for temporal alignment shared by all tracks at a given time step, and others exclusive to each track acting as their individual conditioning.

However, as a pianoroll is not exactly a continuous representation, its application to continuous operations is not guarantied to yield "efficient" results. It also comes with two major downsides:

- It can in theory represent information other than notes, such as tempo or chord, by allocating a dedicated "row" alongside the pitch axis, but the overall continuous nature of the pianoroll would be further ambiguous and its usage with CNN unlikely to be efficient;
- It cannot distinguish held notes from repeated notes.

For these reasons, researchers put the pianoroll aside and started to work with other continuous techniques, coupled with discrete representations. MusicVAE [160] is VAE (see Subsection 2.3.1) architecture where the decoder is a stack of RNN sub-models, each of them generating sixteen beats. The latent space of the encoder is used as the initial hidden state of the RNNs. This proposition of using multiple RNNs is to alleviate the vanishing gradient problem that very often arises with this type of model, especially with long sequences. The same architecture has been adapted to be used with diffusion models [130] (Subsection 2.3.3): the encoder is now a diffusion model generating the latent embeddings used by the decoders.

Continuous models are however in minority in the symbolic music generation domain. When applied on discrete domains, they tend to suffer from high variances that make them unsuitable to be sampled to generate coherent results. As for text, the vast majority of models are based on discrete sequential models. Some continuous applications can still be used, as done by Choi et al. [27], which uses an autoregressive Transformer to generate music, but uses a seq2seq architecture where the encoder encodes the input into a continuous space, used by the decoder in cross-attention operation to control the generation.

Chapter 4

Challenges in music generation

When used with DL, symbolic music is now mostly handled with sequential models, for their flexibility, greater performances. Using these models requires to first serialize music into sequences of tokens. Previous work already addressed this topic by introducing models, mostly for music generation [68, 87, 85, 83], with specific ways of tokenizing symbolic music. Yet, the topic of how to tokenize music, and its impact on deep learning models is yet to be explored.

4.1 Easy ways to tokenize music

Even though many previous works share the source code of their experiments [83, 87, 194], their implementations are often non trivial to reuse. The code to tokenize music is often hidden, using different package dependencies, and preprocessing MIDIs with various methods. All of this requires time to adapt and fix the code. The barrier to use DL with symbolic music is relatively high compared to other modalities such as text, image or audio which benefit from well established softwares. Moreover, this lack of software make the comparison between music tokenizations unfair, as the preprocessing will be different.

The absence of common or widely used tokenization practice also limits the reutilization pretrained models. In NLP, platforms such as the Hugging Face Hub[1] allows to freely and openly upload and share pretrained models that can be reused by everybody. A tokenizer is always shared with each model, with which the latter has been trained, that can encode and decode the data to be used with the model. There is no such thing for symbolic music. We must state that there are very few pretrained models for symbolic music that are openly shared (as discussed in Section 4.5) anyway. However, we believe the reason is partly due to this lack of tokenization standard.

Creating an easy way to tokenize music will hence lower the barrier of using DL with symbolic music. It would help researchers and engineers and accelerate their productivity, as well as more easily share their works.

4.2 Music representation

Unlike text, symbolic music can be tokenized in different ways, with greater flexibility. Time, note durations, or instruments can be represented in different manners as a sequence of tokens, as such researchers developed various methods to tokenize music [137, 87, 48]. Yet, the previous works do not deeply study the differences between these tokenizations and their impact on model learning. Moreover, these works mostly focus on music generation, leaving out other modeling tasks which are equally important when assessing music tokenization.

[1] https://huggingface.co/models

Yet, the way the music information is represented can easily influence model performances. Generative models are essentially causal, meaning that the computations for a given position within the input sequence will be conditioned only on the previous positions. Hence, generative models have more limited scope and modeling capabilities. On the other hand, the models for other tasks are mostly bidirectional, meaning that the computations are conditioned on both the past and future context (tokens). With `NoteOff` tokens that indicate the note durations implicitly, a generative model could hence be disadvantaged to capture the melody and harmony as some durations cannot be determined at the token positions before the `NoteOff` token, whereas a bidirectional model can access to this information to all the positions.

We can imagine other tokenizations that could combine different representation of distinct musical features such as time, note duration, instrument or tempo information. Some models operate differently, and some information might be more important for different tasks. We can easily identify the possible design choices of symbolic music tokenization, and all the possible combinations, but we do not know exactly how to choose them. For these reasons, an analysis of these choices for different types of models and tasks could bring inside to this field of research, and provide guidance on symbolic music tokenization.

4.3 Token sequence length

By tokenizing music using single tokens for every note attributes (pitch, velocity, duration) and time, we end up with fairly long token sequences, in particular when treating music with several instruments or with high note densities. As a note is serialized into three tokens, its token sequence length would be at least three times its number of notes, to which must be added the number of tokens representing time or other information such as tempo or chords. In all cases, the density of information per token is very low, as each token only represents an absolute value.

This sequence length is problematic when using Transformer models, as their complexity grows quadratically with the input sequence length. Hence a transformer can handle a limited input sequence length that contains very little musical information, or in other words represents few notes or a small number of beats. With larger compute resources, a model could handle longer sequences, but would however be less efficient, and the costly compute requirements would grow quadratically too.

Attempts have been made to reduce this sequence length for symbolic music. The first strategy works by merging the embeddings of tokens that occur simultaneously in time. CPWord [83] merges the embeddings of note attributes, those of tempos with those of positions, and time signature with bar. Octuple [194] goes even further by merging all note attribute embeddings with embeddings representing their absolute time in bar and position. Another technique to reduce sequence length is to use tokens that combine several attributes. For example, LakhNES [41] combines instrument tokens with `NoteOn` and `NoteOff` tokens.

However, these methods show significant drawbacks. Merging embeddings imposes constraints on the software implementation of models, their training and generation. Generating tokens simultaneously from multiple output distributions unconditionally from each other adds variance and unstable results. On the other hand, manually combining tokens yields big vocabularies with a high proportion of tokens not present within the data, so associated with very little probability by the model. Finally, even with these methods the sequence length remains fairly long.

A technique that can efficiently reduce the token sequence length, with few constraints is still to be found and experimented for symbolic music. Moreover, the current techniques do not address the following challenge which concerns the information carried by the tokens.

4.4 Information carried by the tokens

We saw in Subsection 2.1.4 that sequential models contextually learn embedding representations of the tokens of the vocabulary. Having well learned embeddings is an essential feature of these models as they allow to capture the meaning of the data and perform computations solving the tasks they are trained for. To ensure this, the models must be paired with vocabularies containing tokens which represent semantic information themselves. The number of tokens, and in turn embeddings to learn, must be chosen accordingly to the number of dimensions of these embeddings.

Yet, the works addressing symbolic music modeling use small vocabularies, containing between 200 to 500 tokens, with numbers of embedding dimensions ranging from 512 to 1024. In such configuration, i.e. more dimensions than the number of elements to represent in the space, only very few of the embedding space will be used. By comparison, in NLP, the vocabulary sizes can range from 30k to 70k tokens. Moreover, it is important to note that the tokens representing note and time attributes do not carry semantic information other than their absolute values.

In order to increase model performances for symbolic music, it is essential to find better balance between vocabulary size and number of embedding dimensions, while using tokens carrying more information.

4.5 Open source large music generation model

In the recent years, we have witness a growing trend from researchers to openly share (pre)trained models on the internet. The Hugging Face Hub `https://huggingface.co/models` is the most popular platform for this purpose.

Having performant models publicly available brings noticeable benefits for the entire research community and users: it promotes transparency, lower the barrier to entry in the field, drive innovation and make AI fairly accessible to more people. Open models allow people to inspect them and identify risks, biases and limitations to be addressed and in turn improve their safety. They can use them for their own research, by finetuning them or analyzing their behavior, that will in turn profit to other researchers. And as training large models can be very costly, a large number of organizations cannot afford to do it on their own. Making models freely accessible allows a fairer access to smaller actors, and drives innovation and business.

While we can find thousands of models for NLP and CV tasks, there are currently very few publicly available symbolic music generation models, the existing ones are either small, bad, too specific towards a genre or instrument, unusable or obsolete. To this day, such model are not largely used anyway, as they poorly integrate in the workflow of musicians. Yet, we begin to see software allowing to embed DL models as VST plugins into DAWs [70]. We believe this is a big step towards a wider adoption of DL models for music composition, and will lead to more integrations of DL models into DAWs. However, the lack of good and performant models is still a second obstacle to this adoption. It may even be a reason for the slow development of DL model integration solutions. As soon as performant models will be made openly available, these solutions might begin to get more attention and be updated at a faster pace.

Part II

Contribution

Chapter 5

MidiTok

5.1 Introduction

Recent progress in natural language processing (NLP), such as Transformers [177], has been used with symbolic music for several tasks such as generation [85, 87, 164], understanding [194], or transcription [72, 62], with state-of-the-art performances. Most generative deep learning models for symbolic music are nowadays based on LMs. Using LMs for symbolic music requires, as for natural language, to tokenize the music, i.e. serialize it into sequences of distinct tokens. These tokens will represent note attributes such as pitch or duration, and time events.

The tokenization of music, i.e. the conversion of notes into sequences of tokens, is however not a straightforward process. Unlike text, polyphonic music comes with simultaneous notes, each of them having several properties, and the problem becomes even more complex if we consider several tracks or instruments. Many recent research papers introduced different ways to tokenize symbolic music, but few authors share they source code in an easy way to reproduce, or to just use their methods. Furthermore, music files, such as MIDIs, need to be properly preprocessed. This step consist in downsampling its values, such as the onset and offset times of notes, their velocities, duration, or tempo values among others.

With the motivation to offer a friendly and convenient way to tokenize MIDI files, we created MidiTok. It implements the most popular tokenizations, under a unified API. It offers a great flexibility and extended features, so one can easily train and use LMs for symbolic music, and compare the different tokenizations. MidiTok was first introduced in late 2021 [55], and in the meantime received multiple updates until it became established in the community. Today, it is the "go-to" solution for researchers and engineers to use LMs with symbolic music. MidiTok has been built all along the making of this thesis, and has been used for most of the results reported in it. MidiTok is a major living contribution of this thesis, that we believe will continue to evolve.

In the next section, we describe the overall workflow of MidiTok, then the tokenizations and features it implements, and finally some user insights.

5.2 Tokenizing music

In this section, we introduce the data contained in MIDI files, and the previous works which tokenized symbolic music.

5.2.1 The data in MIDI files

When thinking about tokenizing symbolic music, we first need to think about what information present in the MIDI files should be considered. A MIDI file contains several categories of contents. The most important are: tracks of instruments, tempo changes,

time signature changes, key signature changes and lyrics. Tracks themselves contain notes - more precisely their pitch, velocity, onset and offset times - and effects such as sustain pedal, pitch bend and control changes. You can find the complete MIDI specifications on the MIDI Manufacturers Association website[1].

All of these information come as the form of events that occur a certain moments in time. The base time unit of MIDIs is the *tick*, which resolution is called the time division and expressed in ticks per quarter note (or ticks per beat when the time signature is $\frac{*}{4}$). The time division is usually a multiple of 12 and a high value. Common values are 384 and 480. Hence, these events happen each at certain *tick*, and we will have to express them as tokens, along with *time tokens* that accurately represent them at their occurrence time.

Another important aspect is the precision to which represent this information. Values such as Program (the id of an instrument), pitch, velocity range from 0 to 127. They can be considered as "semi-continuous", as 128 is an arguably large number of possible values. But using all these ranges of possible values with a language model is usually not optimal, as the later is a discrete model and so could struggle to efficiently capture the differences between two consecutive values, e.g. a velocity of 100 from a velocity of 101. It is then usually recommended to "discretize" these values, by downsampling the number of possible values. The set of velocities could for example be downsampled to 16 values, equally spaced between 0 to 127. The same downsampling logic should be applied to the time of all the events: instead of aligning the time with a resolution of e.g. 384 ticks per beat, we quantize the times of the events to a resolution of e.g. 8 ticks per beat, which must be sufficiently precise to keep the overall dynamic of the music. An example of time downsampling is depicted in Figure 5.1. Downsampling these values require a careful preprocessing, before representing them as tokens.

Finally, this information can be represented by different manners. In the MIDI protocol, the moments when the notes are played and when their associated key is released come respectively as the form of `NoteOn` and `NoteOff` messages / events. We can then choose to directly tokenize these events, or to explicitly represent the durations of the notes by deducing their values from the difference of time between these messages. The time itself can be represented by using `TimeShift` tokens indicating time movements, or placing `Bar` and `Position` tokens indicating respectively the beginning of the new bar and the current position within the current bar. Representing several tracks from different instruments can also be done by different manners, that we consider a "design" choice. This liberty led researchers to introduce different symbolic music tokenizations, each with their benefits, and us to implement them under a unified API in a flexible and user-friendly library that we called MidiTok.

5.2.2 Previous works

Early works using discrete models for symbolic music, such as DeepBach [69], FolkRNN [173] or BachBot [115], rely on specific tokenizations specifically designed for the data being used, for instance for the four voices of Bach chorales. Non-autoregressive models such as MuseGAN [44] often represent music as pianoroll matrices. Since then, researchers introduced more universal tokenizations for any kind of music. These strategies represent note attributes and time in a general way that can be used with any music. The most commonly used are *Midi-Like* [137] and *REMI* [87]. The former tokenizes music by representing tokens as the same types of events from the MIDI protocol, while the latter represents time with *Bar* and *Position* tokens and note durations with explicit *Duration* tokens. Additionally, *REMI* includes tokens with additional information such as chords and tempo.

[1] https://www.midi.org/specifications

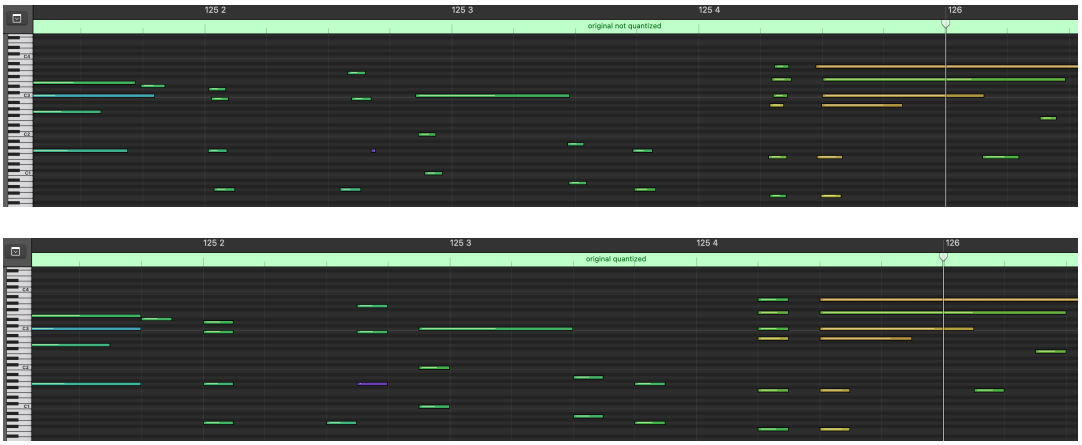


Figure 5.1: Two pianoroll visualizations of a track of piano: top) original MIDI track, as performed by a human; bottom) the same track preprocessed, with onset and offset times aligned to the 8th of beat.

More recently, researchers have focused on improving the efficiency of models with new tokenizations techniques: *Compound Word* [83], *Octuple* [194] and *PopMAG* [158] merge embedding vectors before passing them to the model; 2) LakhNES [41] and [140], SymphonyNet [118] and [56] use tokens combining several values, such as pitches and velocities.

5.3 MidiTok workflow

We introduce in this section the base functioning of MidiTok.

Any MidiTok tokenizer inherits from the `MIDITokenizer` class, which implements all the preprocessing and methods common to all tokenizers. It serves as a global framework, and greatly simplifies the tokenization process.

A tokenizer has to be created from a `TokenizerConfig` object. This configuration holds the parameters defining what type of information will be tokenized, and with which precision. A user can choose whether to tokenize tempos, time signature, rests... or not. He can also decide the resolution of values such velocity, time, or the pitch range to tokenize. From this configuration, the tokenizer will create its vocabulary of tokens. A tokenizer can be saved as a json file, and loaded back as identical without having to provide a configuration.

We consider three categories of tokens: 1) Global MIDI tokens, which represent attributes and events affecting the music globally, such as the tempo or time signature; 2) Track tokens, representing values of distinct tracks such as the notes, chords or effects; 3) Time tokens, which serve to structure and place the previous categories of tokens in time. The categorization of tokens into these three types is important as it will affect the way the tokenizer represents the time.

When tokenizing MIDI tracks, we distinct two modes: a "one token stream" mode which converts all the tracks under a unique sequence of tokens, and a "one stream per track" mode which converts each track independently. In the former mode, the time tokens are created for all the global and track tokens at once, while in the later they are created for each sequence of track token independently.

The tokenization workflow of a MIDI file is as follow:

1. Preprocesses the MIDI object:
   - If in "one token stream" mode, merges tracks of the same program / instrument;

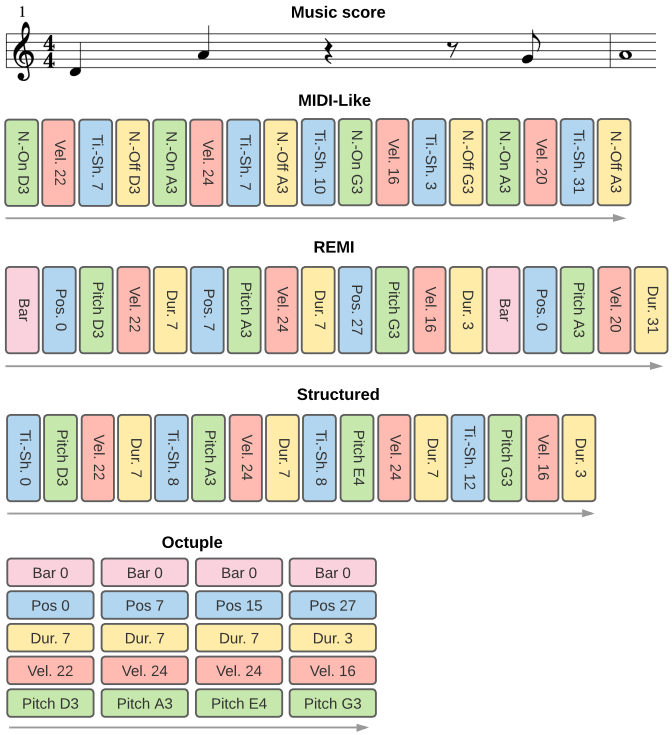


Figure 5.2: A sheet music and several token representations.

- Removes notes with pitches outside the tokenizer's range;
- Resample the MIDI's time division, in other words resample the times of the messages representing the notes, time signature, tempo or other events;
- Deduplicates notes, tempo and time signature changes, and remove empty tracks;

2. Creates global MIDI tokens (tempo...);
3. Creates track tokens (notes, chords);
4. If "one token stream", concatenates all global and track tokens, else concatenates the global tokens to each sequence of track tokens, them sort them by time of occurrence;
5. Deduces the time tokens for all the sequences of tokens (only one if "one token stream");
6. Returns the tokens, as a combination of list of strings and list of integers (token ids).

The first and last step are performed the same way for all tokenizers, while other steps can be performed differently depending on the tokenization. The preprocessing step is essential as it formats the information of a MIDI to fit to the parameters of the tokenizer. The onset and offset times of the track and global tokens are aligned to the time resolution of the tokenizer, as well as their values. This assures us to retrieve the exact same preprocessed MIDI when detokenizing the tokens.

5.4 Music tokenizations

MidiTok implements the most commonly used music tokenizations:

- **MIDILike** [137]: represents MIDI messages as tokens. Notes are represented with `NoteOn` and `NoteOff` tokens, indicating their onset and offset times, and time is represented with `TimeShift` tokens;

- **REMI** [87]: standing for *Revamped MIDI*, it represents note duration with explicit `Duration` tokens in place of the `NoteOff` offset tokens, and time as a combination of `Bar` and `Position` tokens indicating respectively the beginning of a new bar and the position of the time within the current bar;
- **REMI+** [164]: is an extension of *REMI*, allowing to also represent note instruments and time signature;
- **Structured** [68]: similar to *MIDILike*, except that it represents note durations explicitly as *REMI* and always uses the same scheme of token type succession: `Pitch`, `Velocity`, `Duration` and `TimeShift`;
- **TSD** [56]: standing for *TimeShift & Duration*, it is identical to `MIDILike` but with explicit `Duration` tokens;
- **Compound Word** [83]: is similar to REMI, but merges several categories of token embeddings in order to reduce the sequence length for the model. For instance, the `Pitch`, `Velocity` and `Duration` embeddings of a note are first concatenated and projected to get a merged embedding, and several output layers are used to predict these several attributes all at once. We qualify such tokenization as *multi-vocabulary*, as in essence it is based on several distinct vocabularies;
- **Octuple** [194]: also a *multi-vocabulary* tokenization, it works by merged the embeddings of the attributes of each note, along with the `Bar_n` and `Position_p` embeddings representing its position in time, resulting in a token sequence as long as the number of notes tokenized;
- **MuMIDI** [158]: a *multi-vocabulary* tokenization similar to *Compound Word*, but also representing note programs (instruments) along with a built-in positional encoding mechanism based on the number of elapsed bar and position;
- **MMM** [48]: a tokenization for multitrack music generation, including inpaining.

Table 5.1 shows a comparative analysis of these tokenizations, and Figure 5.2 shows different tokenizations applied on an example music sheet melody. In these original works, the authors praise the benefits of using what we call *additional tokens*, which represent information other than the notes and time. Also, multi-vocabulary tokenizations were introduced with the main goal to reduce the length of the sequence of embeddings processed by the model, which can be a bottleneck with Transformer models for which the complexity grows quadratically with. We built MidiTok to offer users the flexibility to choose the types of additional tokens they want to use, and also the possibility to use Byte Pair Encoding for non-multi-vocabulary tokenization to drastically reduce their token sequence lengths. We introduce these features, and more, in the next section.

	Time		Note duration										
Tokenization	`TimeShift`	`Bar + Pos.`	`Duration`	`NoteOff`	Multitrack	One stream	Multi-voc	Chord	Rest	Tempo	Time Sig.	Pedal	Pit. Bend
MIDI-Like [137]	√	-	-	√	†	†	-	‡	‡	‡	‡	‡	‡
REMI [87]	-	√	√	-	†	†	-	‡	‡	‡	‡	‡	‡
Structured [68]	√	-	√	-	†	†	-	-	-	-	-	-	-
TSD [56]	√	-	√	-	†	†	-	‡	‡	‡	‡	‡	‡
CP Word [83]	-	√	√	-	†	†	√	‡	‡	‡	‡	-	-
Octuple [194]	-	√	√	-	√	√	√	-	-	‡	‡	-	-
MuMIDI [158]	-	√	√	-	√	√	√	‡	-	‡	-	-	-
MMM [48]	√	√	-	√	√	√	-	‡	-	‡	‡	-	-

Table 5.1: Comparative table of the tokenizations implemented by MidiTok. †: is true when the tokenizer is configured to represent `Program` tokens; ‡: Is optional. In MidiTok, MMM is implemented using `Duration` tokens.

5.5 Features

5.5.1 Additional tokens

MidiTok allows to choose a set of additional tokens to use. These tokens add more musical information, that can be useful in some cases to model:

- Chord: track tokens describing the chord formed by the following notes. This type of token can help to explicitly model the harmony formed by chords;
- Programs: track token informing of the program, i.e. instrument, of the following notes. This token is natively used in some tokenizations;
- Sustain pedal: track token representing the sustain pedal events;
- Pitch bend: track token representing the pitch bend events;
- Tempo: global token informing of the current tempo, indicating the execution speed;
- Time signature: global token informing of the time signature. The value of the time signature directly impacts the number of beats present in the bars, and the duration of the beats;
- Rest: time token acting as time-shifts, representing rests when no notes is currently being played. The rest is an information by itself that is useful to musicians. Representing it explicitly for a model could likely help it for music modeling tasks, but there is no current research to support this statement.

Huang et al reports that using `Tempo` and `Chord` tokens for a generative Transformer yielded generated results of better quality, that were preferred from human evaluators [87]. In Chapter 6, we will see that tokens representing explicit information usually help the models to learn more efficiently.

5.5.2 Byte Pair Encoding

Byte Pair Encoding (BPE) [59] is a data compression technique. It converts the most recurrent successive bytes in a corpus into newly created ones. BPE is nowadays largely used in the NLP field to build the vocabulary, by automatically creating words and sub-words units from the recurrence of their occurrences within a training corpus [167]. In practice BPE is learned until the vocabulary reaches a target size. We introduce BPE in more details in Chapter 7, and show results of its benefits for symbolic music.

MidiTok allows to use BPE for symbolic music in a simple yet powerful way, to create new tokens that can represent all successive attributes of notes or even successions of notes. Similarly to text, the vocabulary is learned from a corpus of MIDI files. Here however, we use the tokenizer's base vocabulary, that is the set of the basic token representing the information introduced in Section 5.2, as "bytes". MidiTok rely on Hugging Face's tokenizers library [2] for the BPE training and encoding. The library is implemented in Rust and allow very fast computations. To use it for symbolic music, MidiTok associates each base token to a unique byte that the library can recognize.

BPE allows to drastically reduce the sequence length, while taking benefit of the embedding spaces of models such as Transformers, get better results and a faster inference speed [56]. Table 5.2 shows the sequence length reduction offered by BPE. It also shows how BPE increases tokenization and detokenization times. This time increase is however mitigated by the inference speed gains offered by BPE, as the sequence length is drastically reduced.

[2] https://github.com/huggingface/tokenizers

	Voc. size		tokens/beat (↓)		Tok. time (↓)		Detok. time (↓)	
Strategy	TSD	REMI	TSD	REMI	TSD	REMI	TSD	REMI
No BPE	149	162	18.5	19.1	0.174	0.151	0.031	0.039
BPE 1k	1k	1k	9.3 (-49.5%)	10.4 (-45.3%)	0.187	0.163	0.053	0.063
BPE 5k	5k	5k	7.0 (-62.2%)	8.5 (-55.2%)	0.181	0.165	0.053	0.064
BPE 10k	10k	10k	6.3 (-66.0%)	7.7 (-59.7%)	0.183	0.164	0.052	0.065
BPE 20k	20k	20k	5.8 (-68.9%)	6.9 (-63.9%)	0.184	0.163	0.052	0.063
CP Word		188		8.6 (-54.8%)		0.169		0.034
Octuple		241		5.2 (-72.6%)		0.118		0.035

Table 5.2: Vocabulary size, average tokens per beat ratio, and average tokenization and decoding times in seconds on the Maestro dataset [73]. *CP Word* and *Octuple* are grouped with *REMI* as they represent time similarly with `Bar` and `Position` tokens.

The method allows to avoid the drawbacks of multi-vocabulary methods, that are to require to implement multiple input and output modules and use a combination of losses that can lead to unstable learning, slower training, and code adaptations of models and training methods.

5.5.3 Hugging Face Hub integration

The Hugging Face Hub[3] is a model and dataset sharing platform which is widely used in the AI community. It allows to freely upload, share and download models and datasets, directly in the code in a very convenient way. Its interactions rely on an open-source Python package named `huggingface_hub`[4]. As it works seamlessly in the Hugging Face ecosystem, especially the `transformers`[5][183] or Diffusers libraries[6], it stood out and became one of the preferred way to openly share and download models.

Now when downloading a Transformer model, one will need to also download its associated tokenizer to be able to "dialog" with it. Likewise, if one wants to share a models, he will need to share its tokenizer too for people to be able to use it. MidiTok allows to push and download tokenizers in similar way to what is done in the Hugging Face Transformers library.

Internally, MidiTok relies on the `huggingface_hub.ModelHubMixin` component. It implements the same methods commonly used in the Hugging Face ecosystem: `save_pretrained`, `load_pretrained` and `push_to_hub`. Relying on this component allows to easily interact with the hub with as less code and logic as possible, and so a minimal maintenance cost. MidiTok only deals with the two "bridges" between these methods and how tokenizers are saved and loaded.

We still note that at the moment or writing, there are few symbolic models shared on the internet globally, including the Hugging Face hub. Before releasing the interoperability between MidiTok and the hub, users needed to manually download and upload their tokenizers. This is inconvenient as these operations are usually performed within some code pipelines, which are often executed on remote servers. We hence hope that these feature will encourage people to share their models and use those from the community.

5.5.4 Data augmentation

Data augmentation is a technique to artificially increases the size of a dataset by applying various transformations on to the existing data. These transformations consist in altering one or several attributes of the original data. In the context of images, they can include

[3] https://huggingface.co/
[4] https://github.com/huggingface/huggingface_hub
[5] https://github.com/huggingface/transformers
[6] https://github.com/huggingface/diffusers

operations such as rotation, mirroring, cropping or color adjustments. This is more tricky in the case of natural language, where the meaning of the sentences can easily diverge following how the text is modified, but some techniques such as paraphrase generation or back translation can fill this purpose.

The purpose of data augmentation is to introduce variability and diversity into the training data without collecting additional real-world data. Data augmentation can be important and increase a model's learning and generalization, as it exposes it to a wider range of variations and patterns present in the data. In turn it can increases its robustness and decrease overfitting.

MidiTok allows to perform data augmentation, on the MIDI level and token level. Transformations can be made by increasing the values of the velocities and durations of notes, or by shifting their pitches by octaves. Data augmentation is highly recommended to train a model, in order to help a model to learn the global and local harmony of music. In large datasets such as the Lakh [148] or Meta [49] MIDI datasets, MIDI files can have various ranges of velocity, duration values, and pitch. By augmenting the data, thus creating more diversified data samples, a model can effectively learn to focus on the melody, harmony and music features rather than putting too much attention on specific recurrent token successions.

5.5.5 PyTorch data loading

PyTorch is a very popular DL framework, that is widely used in most research and production works. When training a model, the data must be adequately loaded and processed before being fed. For natural language, text data is usually loaded and tokenized on the fly by the CPU, then moved to the GPU on which the model is running. This task is performed by a `Dataset`, a `DataLoader` and a data collator. For symbolic music however, we cannot find a "commonly" used data loading process as the field is not as developed as NLP.

MidiTok offers universal `Dataset` classes to load symbolic music, and data collator to be used with. `DatasetMIDI` loads MIDI files and can either tokenize them on the fly when the dataset is indexed, or pre-tokenize them when creating it and saving the token ids in memory.

When training a model, a user will likely want to limit the possible token sequence length in order to not run out of memory. The dataset classes handle such case and can trim the token sequences. However, it is not uncommon for a single MIDI to be tokenized into sequences that can contain several thousands tokens, depending on its duration and number of notes. In such case, using only the first portion of the token sequence would considerably reduce the amount of data used to train and test a model.

To handle such case, MidiTok provides the `pytorch_data.split_midis_for_training()` method to dynamically split MIDI files into chunks that should be tokenized in approximately a desired number of tokens.

The MidiTok data collator allows to trim token sequences to the desired length and pad them. Padding can be done on the left side, which can be handy when generating autoregressively from a model, and required with some libraries such as Transformers [183].

These methods allows to considerably ease the work of researchers and engineers when training models.

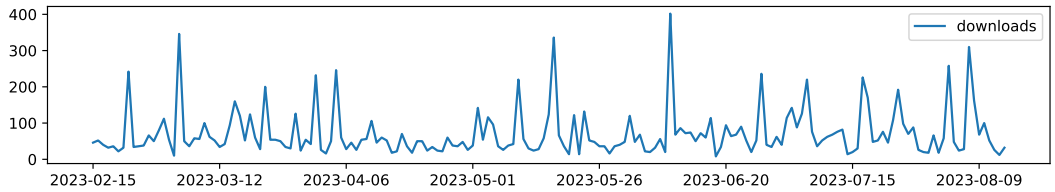


Figure 5.3: Daily downloads of MidiTok on PyPi.

5.5.6 Other useful methods

Finally, MidiTok feature other useful MIDI manipulation and data extraction methods that can be used for other purposes: splitting, concatenating or merging MIDIs, chord extraction, deduplicating overlapping notes, tracks merging by program or category of programs, bar count or note deduplication.

5.6 Broader impact

5.6.1 Usage insights

Since its first version in late 2021, MidiTok gained attention as a simple yet flexible way to tokenize music. It is being used by researchers of the Music Information Retrieval (MIR) community for their works, independent developers, students, and now industrial actors.

As for March 2024, MidiTok gathered more than 550 stars on GitHub, 70 forks, and 20 external contributors. The GitHub repository page counts an average of 100 daily visits, and the package is downloaded on average 900 times per week. Figure 5.3 shows the daily downloads of MidiTok on PyPi, for the february 2023 - august 2023 period. At the time of writing, it counts more than 90k downloads on PyPi since its first release. We cannot reliably estimate the number of monthly or yearly projects using MidiTok, but we can mention that each year a number of research papers published at the recognized ISMIR proceedings share results obtained thanks to MidiTok.

Finally, MidiTok is used as backbone in Qosmo's Neutone plugin[7], to be used in DAWs. Neutone allows to use DL models interactively in any DAW as a VST plugin. This last point may be the most important for the future of MidiTok. As of 2023, the offer of AI assisted music creation tool has been relatively poor, and most musicians do not use them. We recently witnessed the apparition of text-to-music models such as AudioGen [106] or DiffSound [190], capable to generate audio from a text prompt description, but these models are not widely used to generate complete music. While they can produce coherent and high quality results, they face challenges to be used by musicians: their controllability and interactivity are limited, users are not used to create descriptive enough prompts to get the desired results. Neutone tackles the challenge of interactivity by embedding models right in the production tools of musicians, which we believe is a big step towards a larger adoption of DL models in the creation process of musicians.

5.6.2 Involvement with other open-source projects

MidiTok is entirely open-source, and rely on software dependencies that are too. One very important dependency is the one allowing to load and save MIDI files. Until the version 3.0, MidiTok relied on MidiToolkit [8] for these operations. MidiToolkit was created by

[7] https://neutone.space
[8] https://github.com/YatingMusic/miditoolkit/

Wen-Yi Hsiao in 2019 as no Python library allowed to parse MIDI at the "note-level" and with the native tick time unit. Along the years, some bugs began to stack up. We collaborated with Wen-Yi Hsiao in order to keep the library updated, by fixing bugs, optimizing the code and improving its quality and continuous integration.

Although MidiToolkit is a viable, working and established library to parse and save MIDI files, it is written entirely in Python and is not very fast. The loading time of MIDI files has been a bottleneck for the tokenization of MIDI files. In particular, the training of model could be slowed by loading MIDIs, making users first tokenize the whole dataset into Json files, that would then be loaded. In the absence of a fast and efficient Python library, Yikai Liao and Zhongqi Luo created Symusic [9], a library written in C++ allowing to load, save, and perform others operations on MIDIs at a unbeaten speed, about 500 times faster than miditoolkit. Yikai Liao and Zhongqi Luo contacted us in the early development of symusic, allowing us to give advices on the directions to take on its design. We also could progressively test it, in order to improve its usability and making sure it preserves the data integrity of the read and written files.

5.7 Related works

Other similar projects for symbolic music preprocessing (for deep learning models) have been developed. MusPy [42] offers diverse features such as downloading and converting datasets in various formats, analysis and visualization. NoteSeq[10] is also built for music analysis. Both offers methods to tokenize music, but only as *MIDILike* and with limited features.

MidiTok is built with a different vision: to focus solely on tokenization by offering the best features for it, and let the user use other tools better suited for other tasks such as analysis or evaluation.

5.8 Conclusion

Following the growing usage of deep learning models and generative AI, MidiTok stands as an fully-implemented and open source library for symbolic music tokenization. It can easily be used with any other libraries such as Transformers [183], and is built with powerful and flexible features. It gained attention in the MIR community, and we hope to keep benefiting from its feedback and contributions to further improve the library.

We address special thanks to Ilya Borovik and Atsuya Kobayashi for their major contributions, and acknowledge all contributors and people that may have helped in any way for the development of MidiTok. We also address special thanks to Yikai Liao and Zhongqi Luo for their work on the symusic project, which is very valuable gift to the scientific and open-source community and allowed us to make MidiTok significantly faster.

[9] https://github.com/Yikai-Liao/symusic
[10] https://github.com/magenta/note-seq

Chapter 6

Impacts of tokenization designs

6.1 Introduction

Most tasks involving using deep learning with symbolic music [18] are performed with discrete models, such as Transformers [177]. To use these models, the music must be first formatted into sequences tokens.

We saw in Chapter 5 how to tokenize music. Compared to text, tokenizing music provides greater flexibility, as a musical piece can be played by different instruments and composed of multiple simultaneous notes, each having several attributes to represent. As a result, it is necessary to serialize these elements along the time dimension. To achieve this, researchers have developed various methods of tokenizing music [137, 87, 194, 55].

While these works present model performance comparisons between tokenizations, their main differences or similarities are not always clearly stated. Moreover, they mostly focus on music generation, for which evaluations are performed on results obtained autoregressively, which accumulates biases [82] and is arguably difficult to evaluate [189], rather than music modeling more broadly. Yet, Transformer models are trained to learn correlations and make predictions, but not general reasoning. In particular, they struggle at making logical deduction based on information points in the input data [75, 196], but perform tasks better when fed with explicit information and instructions [198]. In the case of symbolic music, it is thus important to study how the ways the music information is represented impact model performances.

In this chapter, we analyze how the tokenization design choices can impact model performances, for several tasks. We focus on three important aspects: the representation of time, note duration and instruments. We believe that they are significant and impactful design choices for any music tokenization approach. Through experiments on composer classification, emotion classification, music generation, and sequence representation, we demonstrate that these design choices produce varying results depending on the task, model type, and inference process.

We present next the related works, followed by an analysis of music tokenization, experimental results, and finally a conclusion. The source code is available for reproducibility[1].

6.2 Related works

We introduced in Chapter 5 most works introducing ways to tokenize music. They mainly compared tokenization strategies by evaluating models with automatic and sometimes subjective (human) metrics, but often do not proceed to comparisons between the ways to represent one of the dimensions we introduced previously. [87] compared results for the generation task, for the use of `Bar` and `Position` tokens versus `TimeShift` in seconds and beats.

[1] https://github.com/Natooz/music-modeling-time-duration

To the best of our knowledge, no comprehensive work and empirical analysis have fairly compared these possible tokenization choices. Conducting such an assessment would require an extensive survey. In this chapter, we focus on the possibles ways to tokenize music, and how they impact models learning.

We specifically show the importance of the explicit information carried by the token types, as they directly impact the performances of models.

In this chapter, we aim to demonstrate the impact of different tokenization choices on models performances and which combinations are suitable for different tasks. We introduce next how music tokenization can be decomposed.

6.3 Decomposing music tokenization

When analyzing the possible designs of music tokenization, we can distinguish seven key dimensions:

- **Time**: Type of token representing time, either *TimeShift* indicating time movements, or *Bar* and *Position* indicating new bars and the positions of the notes within them. We can also consider the unit of *Time-Shift* tokens, either in beats or in seconds.[2]
- **Notes duration**: How notes durations are represented, with either *Duration* or *NoteOff* tokens.
- **Pitch**: Most works use tokens representing absolute pitch values, although recent work shed light on the expressiveness gain of representing as intervals instead [98];
- **Multitrack representation**: The representation of several music tracks in a sequence, i.e., how are the notes linked to their associated track.
- **Additional information**: Any additional information such as chords, tempo, rests, note density. Velocity can also falls in this category;
- **Downsampling**: How "continuous-like" features are downsampled into discrete sets, e.g. the 128 velocity values reduced to 16 values;
- **Sequence compression**: Methods to reduce the sequence lengths, such as merging tokens and embedding vectors.

Some of these dimensions, such as time or pitch, offer few ways to be represented, while others offer more freedom. For instance multitrack can be represented by `Program` tokens[3] preceding notes tokens as in FIGARO [164], distinct tracks sequences separated by `Program` tokens as in MMM [48], combined note and instrument tokens as LakhNes [41] and MuseNet [140], or merging `Program` embeddings with the associated note tokens (MMT [43], MusicBert [194]). One could even infer each sequence separately and lately model their relationships with operations aggregating their hidden states an in ColBERT [99].

The MIDI protocol supports a set of effects and metadata that can also be represented when tokenizing symbolic music, such as tempo, time signature, sustain pedal or control changes. Some works also include explicit `Chord` tokens, detected with rule-based methods. Nevertheless, only a few works experimented with such additional tokens so far ([87, 24]).

[2] In this thesis we only treat of the beat unit. The MIDI protocol represents time in *tick* unit, which value is proportional to the time division (in ticks per beat) and tempo. Hence, working with seconds would require a conversion from ticks.

[3] Following the conventional programs from the MIDI protocol.

Sequence length reduction can be also be handled in very flexible ways. Token sequence length can grow quickly, for dense or multitrack music for instance. However, Transformer models have a time and space complexity that grows quadratically with the input sequence length, requiring either more compute power or to reduce the scope of the data being processed. Reducing the sequence length appears as an appealing solution to increase the performances of such models. This is usually done by either combining tokens within the vocabulary, such as in LakhNES [41] or MuseNet [140] which combine the program and pitch information altogether in a tokens, or by merging the embeddings of tokens to be grouped, such as with CPWord [83] or Octuple [194]. This last solution requires however a modified model architecture, as the model has multiple input and output modules. In Chapter 7, we will focus more in detail on the topic of token sequence length, and show that Byte Pair Encoding (BPE) is actually the method yielding the best results and performances.

Previous works have mainly compared tokenization strategies by evaluating models with automatic and sometimes subjective (human) metrics, but often do not proceed to comparisons between the ways to represent one of the dimensions we introduced previously. [87] compared results for the generation task, for the use of `Bar` and `Position` tokens versus `TimeShift` in seconds and beats.

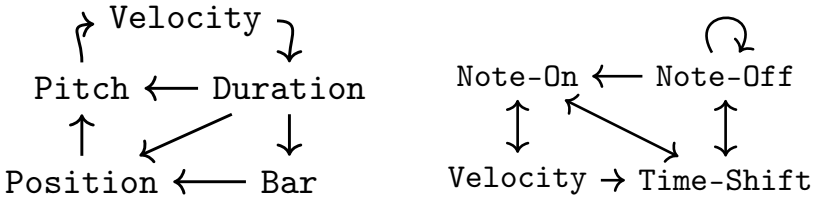
To the best of our knowledge, no comprehensive work and empirical analysis have fairly compared these possible tokenization choices. Conducting such an assessment would require an extensive survey. In this chapter, we will focus on the time, note duration and multitrack dimensions, as they are the among the most important characteristics present in every tokenization.

We want to highlight the importance of the explicit information carried by the token types, as they directly impact the performances of models. `TimeShift` tokens represent explicit time movements, and especially the time distances between successive notes. On the other hand, `Bar` and `Position` tokens bring explicit information on the absolute positions (within bars) of the notes, but not the onset distances between notes. One could assume that the former might help to model melodies, and the latter rhythm and structure. For note duration, `Duration` tokens intuitively express the absolute durations of the notes, while `NoteOff` tokens explicitly indicates the offset times. With `NoteOff`, a model would have to model note durations from the combinations of previous time tokens.

6.4 TSE: Token syntax error metric

To measure the impact of tokenizations on music modeling, we need metrics that can relate to the model's comprehension of the music modality. Previous works often use human preferences metrics, in combinations with feature similarities between original and generated samples. Subjective metrics are best to assess the quality of music, hence a model's capacity to generate pleasant music, while feature similarity allow to measure to a certain degree a model's capability to reproduce similar distributions of features. However, these metrics does not tell much about a model's comprehension of the music modality. Moreover, subjective evaluation are costly and time consuming. In order to tackle the model's comprehension, we designed a metric we called Token Syntax Error (TSE).

Every music tokenization has an underlying syntax of token type and value successions, that can normally happen. For instance, if the last token of an input sequence is of type `Pitch`, some tokenization could impose that the next token to be predicted must be of type `Velocity`. We could also expect a model to not predict more than once the same note at a same time, or to not go back in time.



(a) REMI. (b) MIDI-Like

Figure 6.1: Directed graphs of the token types succession (without additional tokens) for a) REMI [87] and b) MIDI-Like [137].

Successions of incorrect token types can be interpreted as errors of prediction. These errors can help us to measure if a model has efficiently learned the music representation and if it can yield coherent results, or not.

We distinguish five categories of errors:

- **TSE$_{\text{type}}$**: the predicted token is of an invalid type regarding the previous one. For any tokenization, we can draw a directed graph representing the possible token types successions, such as in Figure 6.1;
- **TSE$_{\text{time}}$**: a predicted `Position` value is inferior or equal to the current one, making the time going backward;
- **TSE$_{\text{dupn}}$** (duplicated note): when the model predicts a note that has already been played at the current moment (by the same instrument);
- **TSE$_{\text{nnof}}$** (no NoteOff): when using `NoteOn` and `NoteOff`, and that a `NoteOn` token has been predicted with no `NoteOff` later to end it, or too distant in time;
- **TSE$_{\text{nnon}}$** (no NoteOn): when a `NoteOff` token is predicted but the corresponding note has not been played.

For a given sequence of tokens, TSE measures the ratio, scaled between 0 and 1, of errors for these five categories. A TSE of 0 means that there is no error in the sequence, while a ratio of 1 means only errors were predicted. Our experiments are not concerned by the last two categories as we do not use `NoteOff` tokens.

Finally, we should mention that most of these errors can be avoided by a ruled-based sampling. When predicting a token, one can easily keep track of the time, notes played and token types to automatically exclude invalid predictions. In practice, this can be achieved by setting the invalid indices of the predicted logits to $-\infty$ before softmax.

6.5 Time and note duration

Time and note duration can both be represented in two different ways: `TimeShift` or `Bar / Position` tokens for time, `Duration` or `NoteOff` tokens for note durations. We can then easily classify existing tokenizations based on these criteria, as shown in Table 6.1.

`TimeShift` tokens represent explicit time movements, and especially the time distances between successive notes. On the other hand, `Bar` and `Position` tokens bring explicit information on the absolute positions (within bars) of the notes, but not the onset distances between notes. One could assume that the former might help to model melodies, and the latter rhythm and structure. For note duration, `Duration` tokens intuitively express the absolute durations of the notes, while `NoteOff` tokens explicitly indicates the offset times. With `NoteOff`, a model would have to model note durations from the combinations of previous time tokens.

	Time		Note duration	
Tokenization	`TimeShift`	`Bar` + `Pos.`	`Duration`	`NoteOff`
MIDI-Like [137]	√	-	-	√
REMI [87]	-	√	√	-
Structured [68]	√	-	√	-
TSD [56]	√	-	√	-
Octuple [194]	-	√	√	-

Table 6.1: Time and note duration representations of common tokenizations. `Pos.` stands for Position.

6.5.1 Methodology

For all this chapter, we adopt a common methodology. Unless specified, all experiments will employ the model, model configuration, training procedure and data downsampling described below.

Models and trainings

For all experiments, we use the GPT2 architecture [146], with the same model dimensions: 12 layers, with dimension of 768 units, 12 attention heads and inner feed-forward layers of 3072.

For classification and sequence representation, it is first pretrained on 100k steps and a learning rate of 10^{-4}, then finetuned on 50k steps and a learning rate of 3×10^{-5}, with a batch size of 48 examples. An exception is made for the EMOPIA dataset, for which we set 30k pretraining steps and 15k finetuning steps, as it is fairly small. These models are based on the BERT [37] implementation of the Transformers library [183]. We use the same pretraining than the original BERT: 1) from 15% of the input tokens, 80% is masked with a special `MASK` token, and 20% is randomized; 2) half of the inputs have 50% of their tokens (starting from the end) shuffled and separated with a special `SEP` token, and the model is trained to detect if the second part is the next of the first.

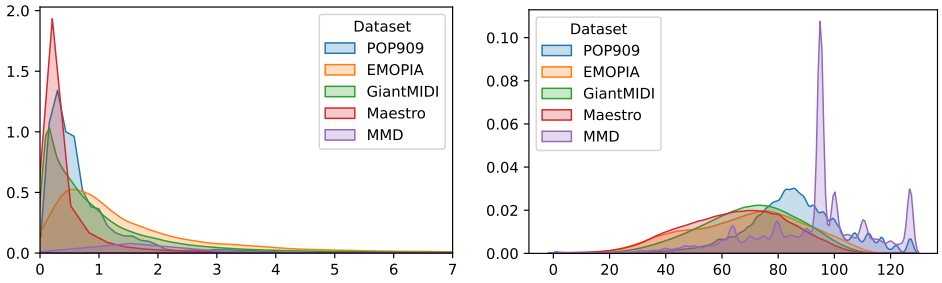
For generation, the model is based on the GPT2 implementation of the Transformers library [183]: it uses a causal attention mask, so that for each element in the sequence, the model can only attend to the current and previous elements. The training is performed with teacher forcing, the cross-entropy loss is defined as: $\ell = -\sum_{t=1}^{n} \log p_\theta(x_t|\mathbf{x}_{\leq n})$.

All trainings are performed on V100 GPUs, using automatic mixed precision [127], the Adam optimizer [101] with $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$, and dropout, weight decay and a gradient clip norm of respectively 10^{-1}, 10^{-2} and 3. Learning rates follow a warm-up schedule: they are initially set to 0, and increase to their default value during the first 30% of training, then slowly decrease back to 0.

10% of the data is used for validation during training, and 15% to test models. Inputs contains 384 to 512 tokens, and begin with a `BOS` (Beginning of Sequence) token and end with a `EOS` (End of Sequence) one.

Tokenizations

We investigate here the four combinations of possible time and note duration representation. In the results, we refer to them as *TS* (`TimeShift`), *Pos* (`Position`), *Dur* (`Duration`) and *NOff* (`NoteOff`). It is worth noting that *TS* + *Dur* is equivalent to *TSD* [56] and *Structured* [68], *TS* + *NOff* is equivalent to *MIDI-Like* [137], and *Pos* + *Dur* is equivalent to *REMI* (without additional tokens for chords and tempo).



(a) Duration in beat (b) Velocity

Figure 6.2: Velocity and duration distributions of the datasets. The duration axis is limited to 7 beats for better visibility.

Data downsampling

When tokenizing symbolic music, continuous characteristics are usually downsampled to discrete sets of values [87, 137, 68]. For instance, velocities can be downsampled from 128 to 32 values. These sets should be sufficiently precise to keep the global information. Downsampling these characteristics helps models to learn more easily, as the values of the reduced sets will be more distinctive, and will help the model to learn more efficiently. Figure 6.2 shows the distributions of velocity and duration values of the notes from the datasets. There is a large proportion of low durations (below two beats). As short notes are more common in these datasets than longer ones, we decide to quantize the duration of notes with different resolutions.

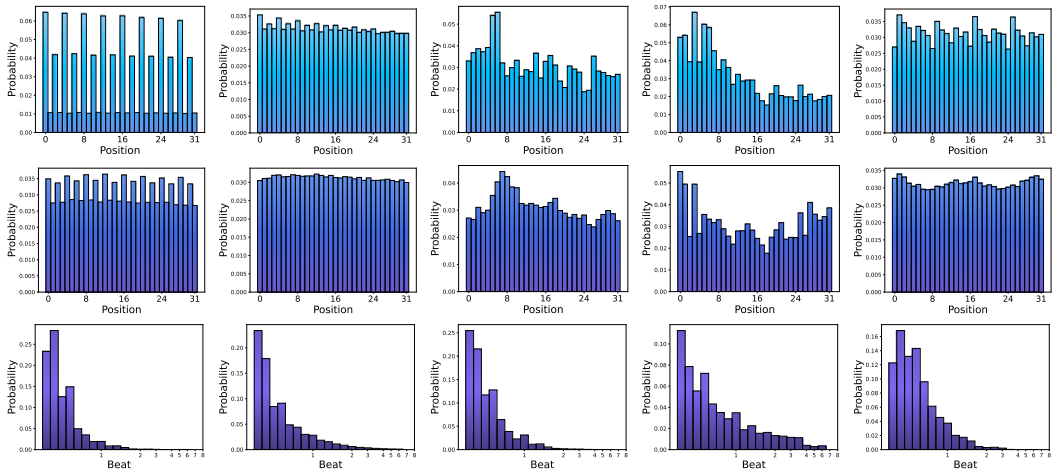
We apply different resolutions for `Duration` and `TimeShift` token values: those up to one beat are downsampled to 8 samples per beat (spb), those from one to two beats to 4 spb, those from two to four beats to 2 spb, and those from four to eight beats to 1 spb. Thus, short notes are represented more precisely than longer ones. `Position` tokens are downsampled to 8 spb, resulting in 32 different tokens as we only consider the 4/* time signature. This allows to represent the 16^{th} note. We only consider pitches within the recommended range for piano (program 0) specified in the General MIDI 2 specifications[4]: 21 to 108. We then deduplicate all duplicated notes. Velocities are downsampled to 8 distinct values. No additional token (e.g., `Chord`, `Tempo`) is used.

We perform data augmentation by creating variations of the original data with pitches increased and decreased by two octaves, and velocity by one value. Finally, following [56], we use Byte Pair Encoding to build the vocabularies up to 2k tokens for generation and 5k for other tasks. All these preprocessing and tokenization steps were performed with MidiTok [55].

6.5.2 Generation

For the generative task, we use the POP909 dataset [181]. The models start with prompt made of between 384 to 512 tokens, then autoregressively generate 512 additional tokens. Evaluation of generated results remains an open issue [189]. Previous work often perform measures of similarity of certain features such as pitch range or class, between prompts and generated results, alongside human evaluations. Feature similarity is however arguably not very insightful: a generated result could have very similar features to its prompts while being of poor quality. Human evaluations, while being more reliable on the quality can also induce biases. Besides, [87] already shows results on an experiment similar to ours.

[4] Available on the MIDI Manufacturers Association website.



(a) *Ts* + *Dur* (b) *Ts* + *NOff* (c) *Pos* + *Dur* (d) *Pos* + *NOff* (e) POP909 data

Figure 6.3: Histograms of the note onset positions within bars (top-row), note offset positions within bars (middle-row) and note durations (bottom-row) of the generated notes. There are 32 possible positions within a bar, numerated from 0 (beginning of bar) to 31 (last 32^{th} note). The durations are expressed in beats, ranging from a 32^{th} note to 8 beats.

Tokenization	$\mathbf{TSE_{type}} \downarrow$	$\mathbf{TSE_{time}} \downarrow$	$\mathbf{TSE_{dupn}} \downarrow$	$\mathbf{TSE_{nnon}} \downarrow$	$\mathbf{TSE_{nnof}} \downarrow$
TS + *Dur*	$< \mathbf{10^{-3}}$	-	0.014	-	-
TS + *NOff*	$< \mathbf{10^{-3}}$	-	**0.001**	0.109	0.040
Pos + *Dur*	0.002	0.113	0.032	-	-
Pos + *NOff*	0.002	0.127	0.005	0.095	0.066

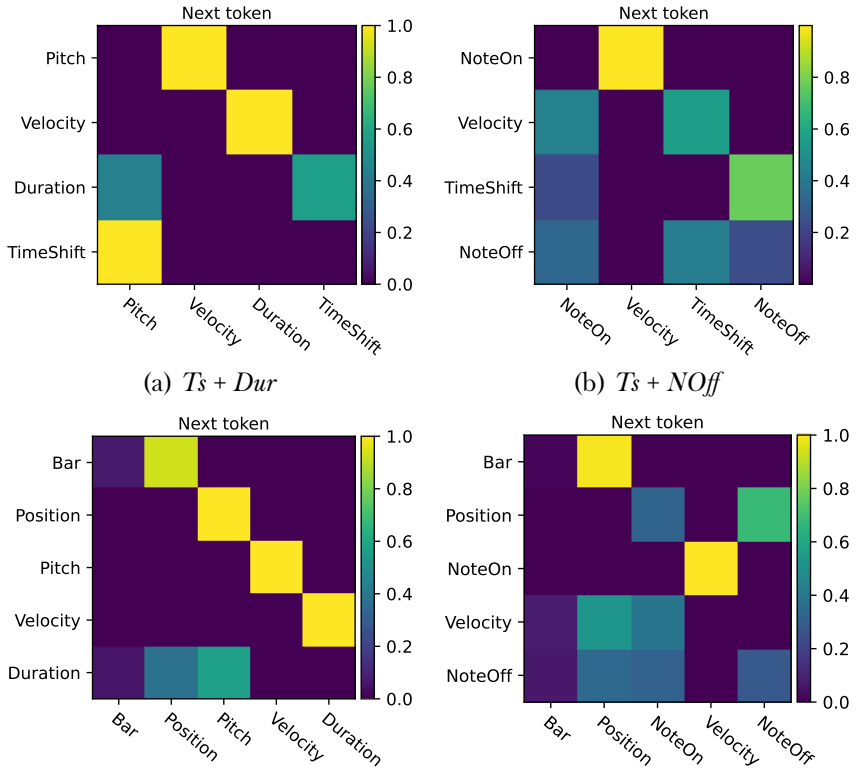
Table 6.2: Prediction error ratios when performing autoregressive generation. - symbol stands for not concerned, and can be interpreted as 0.

Hence we choose to evaluate results on the ratios of prediction errors: Token Syntax Error (TSE) (Section 6.4). This metric is bias-free and directly linked to the design choices of the tokenizations. It allows us to measure how a model achieves to make reliable predictions based on the input context and the knowledge it learned.

The results are reported in Table 6.2. We first observe that the type error ratios are lower than in other categories. This is expected since it is less computationally demanding to model the possible next types depending solely on the last one, rather than on the value of the predicted token, for which the validity depends on a the whole previous context.

`Position` tokens bring almost no type errors, but a noticeable proportion of time errors. When decoding tokens to notes, this means that the time may go backward, and resulting in sections of overlapping notes.

Although `Duration` tokens seem to bring slightly more note duplication errors, the use of `NoteOn` and `NoteOff` tokens results in a considerable proportion of note prediction errors. `NoteOff` tokens predicted while the associated note was not being played ($\mathbf{TSE_{nnon}}$) do not have undesirable consequences when decoding tokens to notes, but it pointlessly extends the sequence, reducing the efficiency of the model, and may mislead the next token predictions. Additionally, `NoteOn` tokens predicted without associated `NoteOff` ($\mathbf{TSE_{nnof}}$) result in notes not properly ended. This error can only be handled by applying a maximum note duration after decoding. Explicit `Duration` tokens allows to specify in advance this information, for both short and long notes. Conversely, with `NoteOff`



(a) *Ts + Dur* (b) *Ts + NOff*

(c) *Pos + Dur* (d) *Pos + NOff*

Figure 6.4: Token type succession heatmaps of the generated results. The vertical axis is a the current token type, the horizontal axis is the next token type following the current one. All rows are normalized.

tokens, the note duration information is implicit and inferred by the combinations of `NoteOn`, `NoteOff` and time tokens. This can be interpreted as an extra effort for the model. Consequently, some uncertainty on the duration accumulates over autoregressive steps during generation. Based on these results, the best tradeoff ensuring good predictions seems to represent time with `TimeShift` tokens and note duration with `Duration` tokens.

In Figure 6.3 we observe the positions within bars and durations of the generated notes. In all cases, onset positions are more distributed at the beginning of the bars. This is especially the case with `Bar` and `Position` tokens, for which we may find unexpected rests at the end of bars, when `Bar` tokens are predicted during the generation before that the current bar is completed. The *TS + Dur* combination places note onsets much more on even positions. The probability mass of *TimeShift* tokens (especially for short values) seems to be much higher. However, this is not the case for the *TS + NOff* combination, as `TimeShift` tokens have to be predicted to move the time on odd positions of note offsets. As shown in Figure 6.4, the model is likely to predict a note (`Pitch`, `Velocity` tokens) after a `TimeShift` token, resulting in evenly distributed onset distribution.

Finally, the use of `NoteOff` tokens tends to produce longer note durations, especially when combined with `Position` tokens. In this last case, we can assume that the model might "forget" the notes currently being played, and that it struggles more to model their durations that have to be implicitly deduced from the past `Bar` and `Position` tokens. These effects are depicted in Figure 6.5 and Figure 6.6. The results of these last figures were obtained with models not fully trained to emphasize the effect of unended notes, as the problem is not so visible with our fully-trained models, as supported by the TSE results of Table 6.2. Such effects can however still be found for smaller models, and should

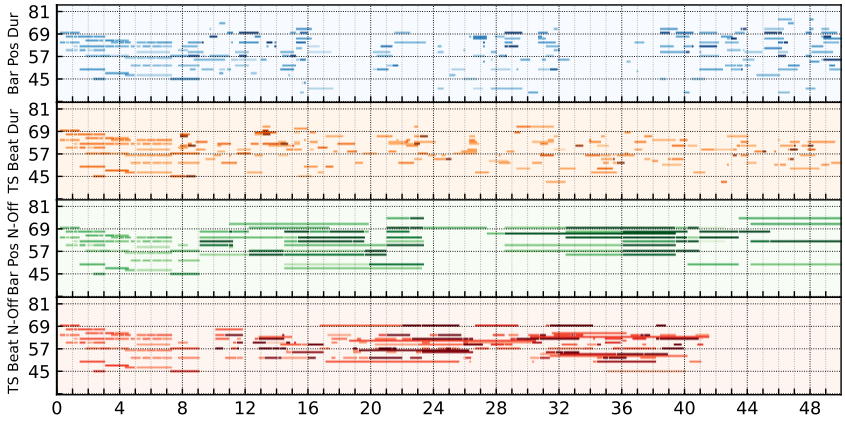


Figure 6.5: Piano roll representation of four continuations of the same 8 beats sample. These results were obtained with models checkpoints at 30% of their total training process.

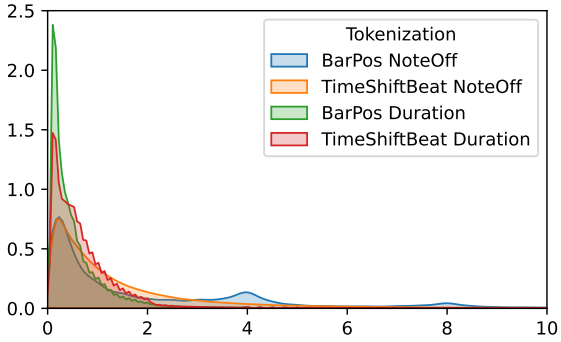


Figure 6.6: Distributions of the durations in beat of generated samples. These results were obtained with models checkpoints at 30% of their total training process.

Tokenization	**Top-20 composers ↑**	**Top-100 composers ↑**	**Emotion ↑**
TS + Dur	**0.973**	**0.941**	**0.983**
TS + NOff	0.962	0.930	0.962
Pos + Dur	0.969	0.927	0.963
Pos + NOff	0.963	0.925	0.956

Table 6.3: Accuracy on classification tasks.

be anticipated.

We hypothesize that the cause of these unended notes results from the higher complexity the use of `NoteOff` tokens implies. Indeed, to generate accurate note durations implicitly implies that the model has to predict `NoteOff` tokens according to the notes currently being played, considering their pitches, starting time and the time passed since. The latter is deduced from the cumulated values of `TimeShift` or `Bar / Position` tokens. This extra complexity confuses the models making them struggling to keep up with their `Duration` counterpart baselines.

6.5.3 Classification

For classification tasks, symbolic music can better suited than audio depending on the data and features to identify. This is particularly true for classical music feature classification, such as composer [104]. Mono-instrument music with complex melodies and harmonies and no particular audio effect benefit from being represented as discrete for classification and modeling tasks. Given this, it felt important to us to conduct experiments on such task.

We choose to experiment with the GiantMIDI [105] dataset for composer classification and the EMOPIA [88] dataset for emotion classification. The results, as shown in Table 6.3, indicate that there is very little difference between the various tokenization methods. However, the combination of `TimeShift` and `Duration` consistently outperforms the others by one point

The classification task involves modeling the patterns from data that are characteristic to composers or emotions. Here, it seems that the time distance between notes, and their explicit duration play a role in these task, more than note offsets or onset positions. This comes with no surprise for the composer classification task, considering that the data is largely composed of complex music with dense melodies and harmonies, featuring mostly short successive notes. Intuitively, patterns of note successions and chords are more easily distinguishable with explicit durations. With implicit note durations, the overall patterns must be deduced by the combinations of `NoteOn` and `NoteOff` tokens while keeping track of the time.

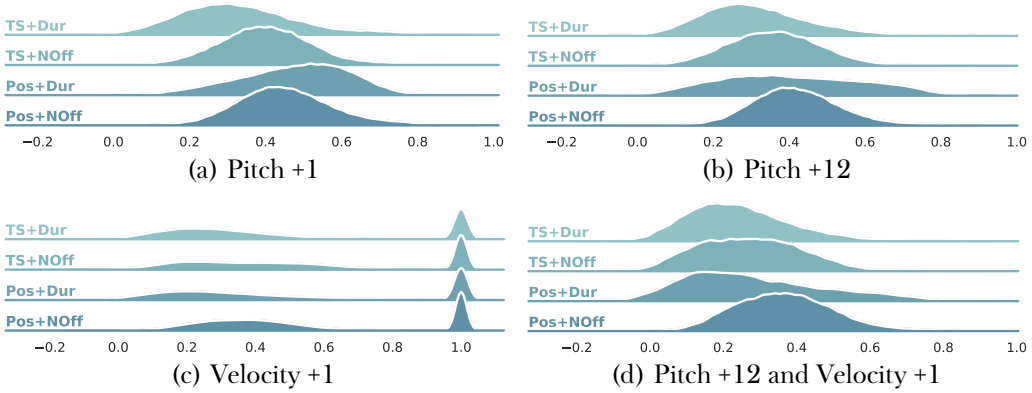
6.5.4 Music transcription

The transcription task is an important application of symbolic music, which we ought to include in our experiments. Its purpose is to transcribe an audio sample into its symbolic equivalent. Recent research built experiment relying on sequence-to-sequence models [72, 62]. We decided to follow the same strategy, for its ease of implementation and good performances: the model is a seq2seq Transformer, the encoder taking mel-spectrograms as inputs, and the decoder autoregressively predicts its symbolic transcription.

Our model is made of 8 layers for both encoder and decoder, an embedding size of 512, 8 attention heads, and inner feedforward layers of size 2048. We used the same audio processing configuration than [72]: a sample rate of 16kHz, FTT window size of 2048 samples, a hop width of 128 samples and 512 mel bins. However, we used `torchaudio`

Tokenization	Ons. + Offs. + Vel. ↑	Onset + Offset ↑	Onset ↑
TS + Dur	0.813	0.822	0.924
TS + NOff	**0.818**	**0.826**	0.925
Pos + Dur	0.808	0.813	**0.928**
Pos + NOff	0.812	0.819	**0.928**

Table 6.4: F1 scores for music transcription. *Onset + Offset* computes the F1 score based on the onset and offset (duration) of the notes, while *Onset* only considers the onset positions.



(a) Pitch +1 (b) Pitch +12

(c) Velocity +1 (d) Pitch +12 and Velocity +1

Figure 6.7: Density plots of cosine similarities between pairs of original and augmented token sequences.

[191] to compute melspectrograms and the T5 implementation of the Hugging Face `transformers` library [183].

We performed the experiment on the Maestro dataset [73], which is made of 1k pairs of audio and MIDI files of classical piano performances. To evaluate the generated transcriptions, we use the `mir_eval` library [149], with the default parameters except an onset tolerance of 62.5 ms, which corresponds to an onset tolerance of a quarter of a beat as the tempo and time division is the same for the whole dataset. `mir_eval` allows to calculate the F1 scores between the expected and generated transcriptions, for individual notes but also for onsets and offsets.

We report the results in Table 6.4. The combinations with the best global performance (onset + offset + velocity) is *TS + NOff*. When looking only at onsets, using *Bar* and *Position* tokens seems to bring better performances. However the combinations using it are underperformed by those using *TimeShift* tokens, meaning that *TimeShift* seems to help the model to predict more accurate note offsets. The scores for the latter are also better with *NoteOff* tokens, showing that explicit offset tokens do help the model to predict offset times. But these offsets time are better predicted when coupled with *TimeShift* tokens than *Bar* and *Position*. This somehow concurs with our previous results in Subsection 6.5.2, where *Pos + NOff* is underperformed by other combinations, as the combination implies an implicit note duration.

6.5.5 Sequence representation

The last task that we wished to explore is sequence representation. It consists in obtaining a fixed size embedding representation of an input sequence of tokens $p_\theta : \mathbb{V}^L \mapsto \mathbb{R}^d$. Here $\mathbb{V} \subset \mathbb{N}$ denotes the token ids of the vocabulary $\mathcal{V}$, L is the variable input sequence length, and d the size of embeddings. In other words, the model learns to project an input token sequence into a embedding space, thus providing a universal representation.

We find this task interesting and well-suited to assess model performances as it directly trains it to model the relationships between tokens within the input sequence and between different representations themselves. While the real-world applications of this task for symbolic music, such as recommending systems, are currently limited, it serves as a useful benchmarking technique for measuring how tokenization impacts the learning of models.

This task has previously been addressed in natural language processing by Sentence-BERT [157] or SimCSE [61]. We adopted the approach of the latter, which uses contrastive learning to train the model to learn sequence representations, for which similar inputs have higher cosine similarities. The sequence embedding is obtained by performing a pooling operation on the output hidden states of the model. We decided to use the last hidden state of the `BOS` token position, as it yielded good results with SimCSE[61][5]. We trained the models with the dropout method: during training, a batch of n sequences $\mathcal{X} = \{\mathbf{x}_i\}_{i=0}^{n}$ is passed twice to the model, but with different dropout masks, resulting in different output sequence embeddings $\mathcal{Z} = \{\mathbf{z}_i\}_{i=0}^{N}$ and $\bar{\mathcal{Z}} = \{\bar{\mathbf{z}}_i\}_{i=0}^{N}$. Although the dropout altered the outputs, most of the input information is still accessible to the model. Hence, we expect pairs of sequence embeddings $(\mathbf{z}_i, \bar{\mathbf{z}}_i)$ to be similar, so having a high cosine similarity. To achieve this objective, we train the model with a loss function defined by the cross-entropy for in-batch pairwise cosine similarities (sim):

$$\ell_i = -\log \frac{e^{\text{sim}(\mathbf{z}_i, \bar{\mathbf{z}}_i)/\tau}}{\sum_{j=1}^{N} e^{\text{sim}(\mathbf{z}_i, \bar{\mathbf{z}}_j)/\tau}} \tag{6.1}$$

As a result, the model will effectively learn to create similar sequence embeddings for similar inputs, while pushing apart those with dissimilarities. We kept a 0.1 dropout value to train the models, and used the GiantMIDI dataset [105].

Evaluation of sequence representation is intuitively performed by measuring the distances and similarities of pairs of similar sequences. We resort to data augmentation by shifting the pitch and velocity of the sequences in order to get pairs of similar music sequences. The augmented data keeps most of the information of the original data. As such, the models are expected to produce similar embeddings for pairs of original-augmented sequence. Ideally, the cosine similarity should be high, yet not to be equal to 1, as this would indicate that the model fails to capture the differences between the two sequences. The results, presented in Figure 6.7, indicate that `Position`-based tokenizations perform slightly better. Therefore, it appears that explicit note onset and offset positions information facilitates models to obtain a universal musical representation.

Unlike classification, the contrastive learning objective models the similarities and dissimilarities between examples in the same batch. In this context, note onset and offset positions appear to be helpful for the models to distinguish music.

We also note the contrasting results when augmenting the velocity. Increasing it by one unit, which would be equivalent to playing just a little bit louder, have arguably a very small impact. As a result, the models mostly produces embeddings that are almost identical for the original and the augmented sequences, but also exhibits uncertainty for a notable proportion of samples.

To complement these results, we estimated the isotropy of sets of sequence embeddings. Isotropy measures the uniformity of the variance of a set points in a space. More intuitively, in an isotropic space, the embeddings are evenly distributed. It has been associated with improved performances in natural language tasks [180, 15, 116], because embeddings are more equally distant proportionally to the density of their area, and are in turn more distinct and distinguishable. We choose to estimate it with the intrinsic dimension of the sets of embeddings. Intrinsic dimension is the number of dimensions required to

[5] SimCSE uses a `CLS` token which is equivalent to `BOS` in our case.

Tokenization	lPCA ↑	MOM ↑	TwoNN ↑	FisherS ↑
TS + Dur	**213**	42.6	34.3	17.5
TS + NOff	161	43.7	32.7	17.5
Pos + Dur	146	39.1	33.1	17.1
Pos + NOff	177	**45.2**	**35.6**	**17.8**

Table 6.5: Intrinsic dimension of sequence embeddings, as an estimation of isotropy.

represent a set of points. It can be estimated through several manners [12]. We choose Principal Component Analysis (PCA) [58], method of moments (MOM) [4], Two Nearest Neighbors (TwoNN) [53] and FisherS [3]. The results, reported in Table 6.5, show that the embeddings created from the *Pos + NOff* combination tends to occupy more space across the dimension of the model, and are potentially better distributed.

6.6 Multitrack music

6.6.1 Representing multiple tracks for sequential models

To this day, there are only a few existing works of DL models for multitrack symbolic music generation. Yet, there are various ways to tokenizer multiple tracks of MIDI instruments. Their goal is to represent a way to associate a note and its attributes to their corresponding MIDI program, or in other words instrument. We describe the five principle, which we will refer as:

- **Program**: a `Program` token is placed before each `Pitch` tokens, indicating the instrument of each note [158, 164];
- **ProgramChange**: a `ProgramChange` token indicates the instrument of the following notes;
- **TrackPitch**: using `TrackPitch` tokens which specify both the pitch and instrument of a given note [41, 140];
- **Merged**: a `Program` embedding is merged with the ones of the notes (`Pitch`, `Velocity`, `Duration`), as with *Octuple* [194];
- **Concat**: the sequences of several tracks are concatenated and separated with *TrackStart*, *Program* and *TrackEnd* tokens, as in MMM [48].

One could even infer each sequence separately and lately model their relationships with operations aggregating their hidden states such as done by ColBERT [99].

Concat is different from the others in the way that it not suited for autoregressive generation with causal models without a specific and dynamically built attention mask. As each track token sequence is concatenated, the attention score at a specific position t within the sequence should be conditioned on the other positions corresponding to events occurring at times (musically speaking, in beats and bars) prior to this of the event associated to token at position t. As such, using a causal mask with such representation would make the attention scores conditioned on all the previous positions, that may include notes and events, within the tracks previously positioned in the sequence, that occur in "the future". Autoregressive generation with *Concat* could be achieved by swapping the order of the track subsequences within the input sequence in order to autoregressively generate the tokens of the last one, until it reaches a defined length, and swap again to generate for another track. Such “chunk-per-chunk” generation would mean that the order of the

	Prog. F1. (↑)	Prog. precision (↑)	Prog. recall (↑)	Note density div. (↓)
Program	**0.93**	0.95	0.92	1.58
ProgramChange	0.92	0.94	0.92	**1.53**
ProgramPitch	**0.93**	**0.96**	0.92	1.68
Merged	0.87	0.83	**0.95**	1.78

Table 6.6: Metrics for the different multitrack strategies. Prog. F1, precision and recall are calculated based on the prompt programs as expected values and generated programs as the results. The note density divergence is the mean difference between the note density in notes per beat between the prompt and the generated continuation, calculated per track.

tracks would influence the final result: for a given section s_t, the first track $s_{t,1}$ would be generated conditioned on the previous section $\mathrm{p}\left(s_{t,1}|s_{<t}\right)$, and the following tracks could then be generated by sampling from $\mathrm{p}\left(s_{t,i}|s_{<t,<i}\right)$. Such method would also impose to use a more complex attention masking strategy in order to let tokens to attend to other tokens located in the past time, from every tracks. This means that the mask would have to be built dynamically while keeping track of the absolute time associated with each token. It is indeed more suited for inpainting tasks, as used in the original work [48] which generate music by sections of bars and complete tracks. Note that this generation procedure could be associated with Gibbs sampling as in [69]. For these reasons, we chose to discard it in this section as it does not fairly compare to the others.

Yet, there is no work presenting analysis and comparing these methods on music modeling. In this section, we focus on the the four first methods, and their impact on the music generation task. For *Merged*, we chose to use the *CPWord* [83] tokenization, with note embeddings being also combined with the embedding of their program token. For the other methods, we chose to represent time with *Bar* and *Position* tokens, similarly to *CPWord* to have a fair comparison, and to *REMI*.

6.6.2 Methodology

We used the same model and training procedure as previously in Subsection 6.5.1, except that we trained the models on 20 epochs and that we used the MMD dataset [49]. It is, to our knowledge, the biggest MIDI dataset publicly available. It features more than 430k MIDI files of all genres of music with multiple tracks. Each piece is matched to Spotify and MusicBrainz ids, allowing to link them with a wide variety of information such as artist or music genre. In order to get a more quality training corpus, we perform a preprocessing step which deduplicates the files of the same music and keeps only the best. It is explained in Appendix A.1, and we impose no requirements on MIDIs before they are added to the graph.

6.6.3 Impact of multitrack tokenization

Our experiment here aims to measure wether a model is able to coherently produce the continuation of a music prompt. More specifically, we measure how a model is able to keep playing the instruments that were present in the prompt. Each prompt is made of 512 tokens, which corresponds to 3 to 10 beats of music in most cases. Considering this short time, we assume that a model should generate notes from the instruments of the prompt, as it is unlikely that a one input tracks contains only a few notes at the beginning.

We compute this measure by measuring the precision, recall and F1 score between the input programs (used as expected) and the generated programs. We report these scores in Table 6.6. It shows that merging the program and pitch tokens (*ProgramPitch*) gives the

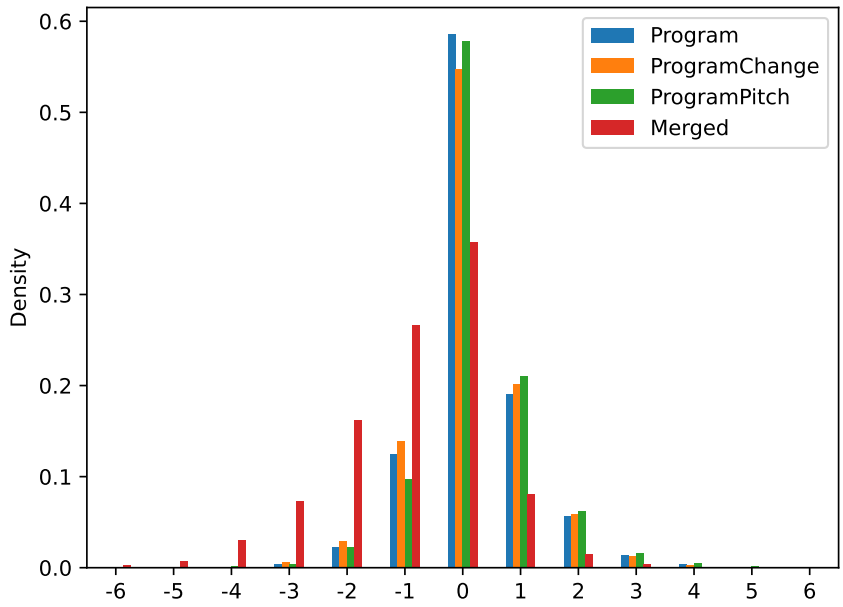


Figure 6.8: Difference of instruments between the prompt conditional input and the results, for the continuation task.

best F1 score and precision. Overall, these methods seem to be better suited to continue the tracks of the input.

We note that *Program* and *ProgramChange* perform almost similarly. It seems that placing a `Program` token before each note does not help the model for the generation. In such case, using `ProgramChange` tokens could be a better option, as it will give shorter sequences of tokens. On the other hand, *Merged* has a good better recall, but has however a lower precision and F1 score. As a result, the model will easily predict unrelated tracks, that were not present in the prompt.

We also measured the divergence between the note density of the tracks from the prompt, and the generated continuations. Although the models perform quite similarly, we still acknowledge that the *ProgramChange* and *Program* methods perform better.

Finally, we plot the difference of number of instruments between the prompt and the continuation for the four methods in Figure 6.8. These numbers are absolute and does not consider if the generated tracks were present in the prompt or not. We first note that *Merged* tend to generate notes from less instruments that what was in the prompt. Considering its good recall score, we can assume that *Merge* focuses on some specific programs, that are statistically more present in the data. Other strategies perform similarly, with *ProgramPitch* tending to predict slightly more new track.

6.7 Conclusion

We have discussed the importance of different aspects of symbolic music tokenization, and focused on three major ones: the time, note duration and multitrack representations. We showed that different tokenization strategies can lead to different model performances due to the explicit information carried by tokens, depending on the task at hand.

Explicitly representing note duration leads to better classification accuracy as it helps the models to capture the melodies and harmonies of a music. Modeling durations, when represented implicitly, adds an extra effort to the model as it must deduce them from the

combinations of time tokens. However, the note offset position information it brings have been found to be more discriminative and effective in our contrastive learning experiment.

For music generation, the time representation plays a significant role, for which the note onset and offsets distributions vary due to the successions of token types. Implicit note durations are less suited for the autoregressive nature of this task, from a prediction error perspective, and sometimes "forgetting" notes being played resulting in higher durations. And when dealing with multitrack music, it appears that using `Program` tokens preceding notes tokens or `ProgramChange` tokens to indicate instrument changes lead to a better respect of the input tracks continuation.

Finally, for music transcription, implicit note durations (note onset and offset) show greater results. The offset positions seems to help the model to learn to efficiently transcript the audio. Furthermore, this representation is better suited for cases where the audio frame contains the end of some notes, but not their onset time. In such case, even an human can hardly estimate the true duration of such note.

We consider this work as a first step into the study of music tokenization for music modeling. We hope that our results can help researchers in their choices of tokenization for symbolic music. We did not experiment with the other music tokenization dimensions such as the downsampling granularity or additional tokens. Furthermore, we believe that more musical reasoning tasks and imposing the model to perform logic deductions to retrieve implicit information from the data, might give more insightful results. Future research will further explore these questions.

Chapter 7

Byte Pair Encoding for symbolic music

7.1 Introduction

When used with deep learning, the symbolic music modality is mostly represented as discrete and used with language models (LM) such as Transformers [177]. These models receive sequences of tokens as input, convert them to learned embedding vectors representing their semantic features in a continuous space, and process these embeddings for the task at hand. A token is a distinct element, known within a finite vocabulary. For natural language, a token can be a word, subword or punctuation mark. For symbolic music, tokens usually represent note attributes or time events, such as pitch or duration. Tokenizing music, i.e., converting raw data into tokens, can be achieved by several ways, as music can be composed of simultaneous tracks, of simultaneous notes with several attributes such as their pitch and duration. Multiple approaches exist to represent these features.

Recently, the token representation of symbolic music has been studied, with the goal to improve 1) the results, e.g. the quality of generated results or the accuracy of Music Information Retrieval (MIR) tasks, and; 2) the efficiency of the models. The former is tackled with more expressive representations [87, 98, 164, 55], and the latter by representations based on either token combinations [140, 41], or embedding pooling [83, 194, 158, 43], which reduce the overall sequence length.

Still, these tokenizations are based on tokens only representing the values of time events and note attributes. This comes with a big limitation: these tokens do not carry much information by themselves. We can assume that their embeddings does not either. By analogy to natural language, these tokens are closer to the character level than word level. Yet, a powerful feature of LMs is their ability to learn (embedding) representations of discrete elements such as tokens, and leverage this information for reasoning and downstream tasks. For natural language, LMs are usually coupled with vocabularies containing up to 50k tokens, represented on a few hundreds dimensions (often between 512 and 2048). Using a vocabulary containing fewer tokens than the number of dimensions used to represent them appears as a suboptimal usage of such models. Moreover, the expressive information carried by music is deduced from the combinations of its notes and their attributes. Considering the infinite possible music arrangements, we can suppose that using solely note attribute embeddings imposes to LMs a heavier modeling effort than embeddings of potential whole note successions that would be more expressive and explicit.

In this chapter, we show that Byte Pair Encoding (BPE, described in Section 7.3) applied to symbolic music allows to address the two goals just mentioned, while outperforming the previous methods and making the model learn better distributed embeddings.

To the best of our knowledge, BPE has yet not been studied for the symbolic music modality, although it can be applied on top of any music tokenization that does not perform embedding pooling. This work aims at closing this gap by shedding light on the results and performance gains of using BPE:

- We experiment on four public datasets [181, 105, 49, 88], with two base tokenizations, on which BPE is learned with several vocabulary sizes, on generation and classification tasks;
- We compare BPE with other sequence reduction techniques introduced in recent research;
- We study the geometry of the learned embeddings, and show that BPE can improve their isotropy and space occupation;
- We show some limits of BPE, such as on the proportion of sampled tokens, and that the vocabulary size has to be carefully chosen.

The source code is provided for reproducibility[1], and we also share a demo website to listen generated results[2].

7.2 Related work

7.2.1 Tokenization of symbolic music

Most deep learning models using symbolic music generation use a specific music tokenization. Early research introduced representations specifically tied to the training data being used, such as DeepBach [69], FolkRNN [173] or BachBot [115]. Non-autoregressive models such as MuseGAN [44] often represent music as pianoroll matrices.

Since, more universal representations have been studied, allowing to convert any sequence of (simultaneous) notes into tokens [137, 87, 68, 55]. Some of them are depicted in Figure 7.1.

7.2.2 Sequence length reduction strategies

More recent works put efforts on the efficiency of the models. As most of them rely on the Transformer architecture [177] and the attention mechanism, their time and space complexity grows quadratically with the input sequence length. This bottleneck led researchers to work on more efficient attention estimations [174], down to linear complexity. In the field of symbolic music specifically, researchers worked on strategies to reduce the sequence length in order to increase 1) the efficiency of the models; 2) the scope of the attention mechanism; 3) the quality of the generated results. These strategies can be split in two categories:

- **embedding pooling** such as *Compound Word* [83] (*CPWord*), *Octuple* [194], PopMag [158], SymphonyNet [118] or MMT [43]. Embeddings of several tokens are merged with a pooling operation. This is often done by concatenating the embeddings and projecting the vector, resulting in an aggregated embedding of fixed size.
- **token combination** such as in MuseNet [140], LakhNES [41] or other recent works [118, 175], which consists of using a vocabulary of tokens representing several attributes, e.g., `Pitch-x_Dur-y` representing both the pitch and velocity information. BPE can be seen as a learned token combination technique.

[1] https://github.com/Natooz/BPE-Symbolic-Music
[2] https://Natooz.github.io/BPE-Symbolic-Music

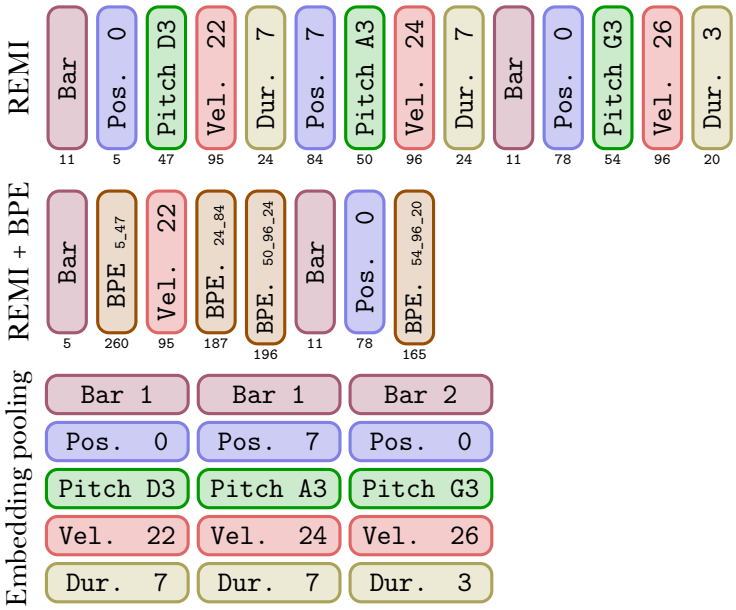


Figure 7.1: Three tokenizations of the same three notes. Tokens are ordered from left to right, the numbers put below are their integer ids. Top row is *REMI* [87], middle correspond to the top row with BPE applied to some tokens, bottom row is a tokenization similar to *Octuple* [194] and *MMT* [43] where the embeddings of the tokens stacked vertically are merged (pooled).

7.2.3 Limitations

One of the main limitation of the previous work is the suboptimal usage of the embedding space of LMs. Most of them use models with embeddings represented from 512 to 1024 dimensions, for vocabularies of less than 500 tokens. As the model contextually learns to gather embeddings along dimensions representing learned features, using a number of dimensions larger than the number of elements to represent causes embeddings to not take advantage of all the space of the embedding dimensions, which will stay unoccupied for a large proportion. For comparison, the same models, when trained on natural language data, use to learn up to 50k embeddings on 512 to 1024 dimensions.

The sequence length reduction strategies just introduced also have big limitations. Embedding pooling: 1) requires specific model input and output modules, which can break compatibility with popular software libraries; 2) requires multiple losses at training, which increases the complexity; 3) for generation, inferring from such model can be seen as sampling from a multivariate distribution, which can be very delicate, as 4) the results can easily degenerate if the pooling does not yield semantically rich embeddings that represent the underlying tokens. On the other hand, token combinations of entire types of tokens can lead to large vocabularies with unused tokens and potentially non-optimized or unbalanced token distributions.

To the best of our knowledge, no work has been conducted on applying BPE (introduced in Section 7.3) to symbolic music generation. Although SymphonyNet [118] introduced a method named MusicBPE, this technique links weakly with BPE and has a limited scope. It adds to the vocabulary new tokens for common chords. These tokens represent pitch value combinations, only for simultaneous notes having the exact same velocity and duration. Unfortunately, it can only be used for limited proportion of notes (and in turn tokens), actually less than a quarter when a strong downsampling is applied (Appendix B.1). As it does not apply on token successions, it cannot capture the contextual

and probability relations between them, including time dependencies. For these reasons, we do not compare it with BPE as it would be irrelevant and unfair.

7.3 Byte Pair Encoding

Byte Pair Encoding (BPE) [59] is a data compression technique. It converts the most recurrent successive bytes in a corpus into newly created ones. For instance, in the character sequence `aabaabaacaa`, the sub-sequence `aa` occurs four times and is the most recurrent one. Learning and applying BPE on this sequence would replace `aa` with a new symbol, e.g., `d`, resulting in a compressed sequence `dbdbdcd`. The latter can be reduced again by replacing the `db` subsequence, giving `eedcd`. In practice BPE is learned on a corpus until the vocabulary reaches a target size. BPE learning is described by the pseudo-code of Algorithm 3.

Algorithm 3 Learning of BPE pseudo-code

Require: Base vocabulary $\mathcal{V}$, target vocabulary size N, dataset $\mathcal{X}$
1: **while** $|\mathcal{V}| < N$ **do**
2: Find $m = \{t_1, t_2\}$, the most recurrent token succession from $\mathcal{X}$
3: $\mathcal{V} \leftarrow \mathcal{V} + [t_{|\mathcal{V}|} : m]$
4: Substitute occurrences of m in $\mathcal{X}$ with $t_{|\mathcal{V}|}$
5: **end while**
6: **return** $\mathcal{V}$

BPE is nowadays largely used in the NLP field as it allows to encode rare words and segmenting unknown or composed words as sequences of sub-word units [167]. Other token aggregation, or vocabulary building techniques exist. The two other most commonly used are Unigram [108] or WordPiece [184], which operations share similarities with BPE.

For natural language, bytes are the distinct characters composing the text. For symbolic music, the distinct note and time attributes can be used as the "bytes" to merge. In this context, BPE can allow to represent a note, or even a succession of notes, that is recurrent in the dataset, as a single token. For instance, a note that would be tokenized as the succession of tokens `Pitch_D3`, `Velocity_60`, `Duration_2.0` could be replaced by a single new one. Rare note (and attributes) can still be tokenized as non-BPE tokens. The same logic applies to time tokens, that can also be associated to note tokens.

7.4 Experimental settings

7.4.1 Models and training

As we specifically focus on LMs, we experiment with the state of the art deep learning architecture for most NLP tasks at the time of writing, the Transformer [177]. We use the GPT2 [146] and BERT [37] implementations of the transformers library [183] for respectively music generation and classification. They are made of 12 layers, embedding sizes of 512, eight attention heads and feed-forward layers of 2048. They count approximately 40M learned parameters. The generators are trained with teacher forcing with the target sequence being the input shifted by one to the left. The classifier are first pretrained to retrieve randomized tokens, and then finetuned to classify the input sequences.

The generator and classifiers are respectively trained and pretrained on 100k steps. For classifiers pretraining, we use the same objective than done with BERT [37]: 15% of each input sequences is masked with a special `MASK` token, 10% of these masked tokens are

randomized, and the loss is computed on the capacity of the model to recover the original tokens. Additionally each sequence is divided into two equal parts separated with a special `SEP` token, and half of the batch sequences have non-related parts. The model predicts if the second part is the next part of the first. The input embedding and output pretraining module weights are tied to improve the performances [144].

The classifiers are then finetuned on 10k steps on the downstream tasks. We feed the output hidden state of the first position (`BOS` token) to an output fully connected layer, to train the model to classify the input sequence.

Trainings are performed on V100 and RTX2080ti GPUs, each time in distributed setups of pairs of the same GPU model, for a total batch size of 128. All trainings are done with automatic mixed-precision [127], the Adam optimizer [101] with $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$, and dropout, weight decay and a gradient clip norm of respectively 10^{-1}, 10^{-2} and 3. We use a one cycle learning rate scheduler: the initial learning rate is close to 0 and gradually grows for the 30% first steps to $1e-4$ for the generators and classifier pretraining and $3e-5$ for the classifier fine-tuning, then slowly decreases down to 0. The model parameters are saved when the validation loss is the lowest ever observed, and after training the last version saved is used for testing.

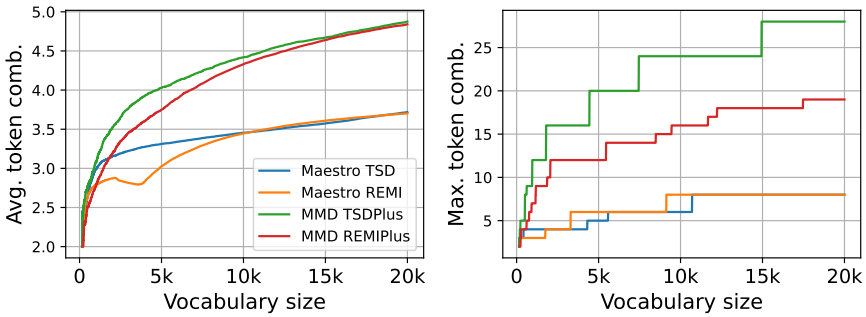
All models receive sequences of 256 to 384 tokens, beginning with special `BOS` (Beginning of Sequence) and ending `EOS` (End of Sequence) tokens. We split datasets in three subsets: one only used for training to update the models, one for validation during training, one used to test the models after training. The last two represent respectively 10% and 15% of the dataset for classification and 2% and 5% for generation.

7.4.2 Tokenization

We experiment with *REMI* [87] and *TSD* (for Time Shift Duration) as base tokenizations, on top of which BPE is applied. Both tokenizations describe notes as a succession of `Pitch`, `Velocity` and `Duration` tokens. *REMI* represents time with `Bar` and `Position` tokens, which respectively indicates when a new bar is beginning and at which position within the time is. *TSD* represents time with `TimeShift` tokens, indicating explicit time movements. For the multitrack MMD dataset, we prepend a `Program` token before the `Pitch` token of each note to represent its instrument.

When tokenizing symbolic music, continuous characteristics are usually downsampled to discrete sets of values [87, 137, 68]. For instance, velocities can be downsampled from 128 to 32 values. These sets should be sufficiently precise to keep the global information. Downsampling these characteristics helps models to learn more easily, as the values of the reduced sets will be more distinctive. Similarly to Subsection 6.5.1, we apply a downsampling on the note attributes and time. We decided to downsample the `Duration` and `TimeShift` tokens with different resolutions: those up to one beat are downsampled to 8 samples per beat (spb), those from one to two beats to 4 spb, those from two to four beats to 2 spb, and those from four to eight beats to 1 spb. This way, short notes are represented more precisely than longer ones, reducing the vocabulary size. For *REMI*, `Position` tokens are downsampled to 8 spb, resulting in 32 different tokens as we only consider the $\frac{4}{*}$ time signature. This allows to represent the 16^{th} note. We only consider pitches within the recommended range for piano (program 0) specified in the General MIDI 2 specifications[3]: 21 to 108. We then deduplicate all duplicated notes. Velocities are downsampled to 8 distinct values. No additional token (e.g., *Chord*, *Tempo*) is used. Finally, as mentioned in Appendix A.1, the set of instruments is downsampled to 13 unique programs.

[3]Available on the MIDI Manufacturers Association website



(a) Avg. nb. of combinations (b) Max. nb. of combinations

Figure 7.2: Average and maximum number of token combinations of tokens learned with BPE in function of the vocabulary size.

	Voc. size		tokens/beat (↓)		Tok. time (↓)		Detok. time (↓)	
Strategy	TSD	REMI	TSD	REMI	TSD	REMI	TSD	REMI
No BPE	149	162	18.5	19.1	0.174	0.151	0.031	0.039
BPE 1k	1k	1k	9.3 (-49.5%)	10.4 (-45.3%)	0.187	0.163	0.053	0.063
BPE 5k	5k	5k	7.0 (-62.2%)	8.5 (-55.2%)	0.181	0.165	0.053	0.064
BPE 10k	10k	10k	6.3 (-66.0%)	7.7 (-59.7%)	0.183	0.164	0.052	0.065
BPE 20k	20k	20k	5.8 (-68.9%)	6.9 (-63.9%)	0.184	0.163	0.052	0.063
PVm	1453	1466	13.4 (-27.8%)	13.8 (-27.4%)	0.134	0.123	0.024	0.026
PVDm	28185	28198	8.2 (-55.5%)	8.6 (-54.8%)	0.119	0.106	0.025	0.030
CP Word		188		8.6 (-54.8%)		0.169		0.034
Octuple		241		5.2 (-72.6%)		0.118		0.035

Table 7.1: Vocabulary size, average tokens per beat ratios, and average tokenization and decoding times in seconds using MidiTok [55] and the Hugging Face tokenizers[4] libraries, on the Maestro dataset.

We choose to experiment with five vocabulary sizes: without BPE, 1k, 5k, 10k and 20k tokens.

Finally, we compare BPE with other sequence reduction strategies introduced in Subsection 7.2.2. We experiment with merging `Pitch` and `Velocity` tokens (*PVm*), and `Pitch`, `Velocity` and `Duration` together (*PVDm*). *PVm* is similar to the strategy used with MuseNet [140]. We also experiment with embedding pooling strategies - *CPWord* [83] and *Octuple* [194] - that we group with *REMI* in our experiments as they represent time similarly. We use the same pooling strategy, and sample independently from the logits of each output modules. All embeddings have the same size than the model dimension.

7.5 BPE learning

BPE allows to significantly reduce the sequence length. As shown in Figure 7.2, the ratio of average number tokens representing a beat can be reduced up to more than 50%. As BPE replaces recurrent pair of bytes in the data, the average number of byte combinations of the vocabulary tends to first quickly increase, then more slowly grow. The inverse tendency can be observed on the tokens per beat ratios shown in Table 7.1, while showing that BPE increase only slightly the tokenization time. The maximum number of byte combinations varies depending on the data. Here, the MMD dataset allows to learn much more combined tokens. This shows that the Maestro dataset contain much diverse token

[4] https://github.com/huggingface/tokenizers

	TSE_{type}(↓)		TSE_{dupn}(↓)		TSE_{time}(↓)		Hum. Fidelity (↑)		Hum. Correctness (↑)		Hum. Diversity (↑)		Hum. Overall (↑)	
Strategy	TSD	REMI	TSD	REMI	TSD	REMI	TSD	REMI	TSD	REMI	TSD	REMI	TSD	REMI
No BPE	1.53	1.34	4.19	5.59	-	**28.93**	4.9%	4.0%	2.0%	2.0%	1.0%	0.0%	4.8%	0.0%
BPE 1k	1.59	0.62	3.60	4.16	-	34.65	13.6%	11.9%	11.8%	14.9%	10.8%	6.8%	8.6%	8.6%
BPE 5k	**0.31**	**0.38**	3.28	4.10	-	39.25	21.4%	**31.7%**	20.6%	21.8%	11.8%	11.7%	20.0%	18.1%
BPE 10k	0.49	1.04	3.83	6.39	-	48.16	23.3%	20.8%	**29.4%**	22.8%	18.6%	20.4%	22.9%	29.5%
BPE 20k	0.38	0.64	4.09	**3.60**	-	52.00	**29.1%**	19.8%	**29.4%**	**24.8%**	**36.3%**	**34.0%**	**30.5%**	**30.5%**
PVm	2.45	2.99	16.90	16.33	-	36.31	2.9%	2.0%	2.9%	0.0%	7.8%	2.9%	4.8%	1.0%
PVDm	0.63	6.32	**2.84**	10.64	-	46.75	4.9%	9.9%	3.9%	11.9%	13.7%	21.4%	8.6%	12.4%
CPWord		6.15		28.55		62.15		0.0%		2.0%		2.9%		0.0%
Octuple		-		244.11		305.43		0.0%		0.0%		0.0%		0.0%

Table 7.2: Metrics of generated results. TSE results are all scaled at e^{-3} for better readability. Hum stand for human, "-" for non-concerned (i.e. 0).

successions, which is not surprising considering that it is made of classical music while MMD contains many genres, among which some with very repetitive patterns. The tokenization time with BPE naturally increases, but stays relatively close to the baselines without BPE.

Appendix B.2 complements this analysis by shedding light on the types of the underlying tokens represented by the newly learned tokens.

7.6 Music generation

Music generation is a popular application of deep learning models [18, 17]. We ought to experiment on this task to demonstrate the benefits of BPE on music modeling. For this task, we choose to use the Maestro dataset [73], which is made of 1k pairs of audio and MIDI files of classical piano performances. Each MIDI file is made of one piano track, with dynamic melodies and complex harmonies. We generate autoregressively the next 512 tokens of input prompts from the test subsets of the Maestro dataset, filtering the logits by keeping the top $p = 0,95$ probability mass (nucleus sampling [82]) and top 15 token probabilities (top-k sampling [54]).

Evaluation of symbolic music is still an open issue [189]. In the absence of automatic metrics measuring the distances between subsets of data, most works evaluate generated results with human surveys along with feature similarity metrics. The latter however cannot capture the quality of music, and is subject to irregularities in case of model over or underfitting. We decide here to replace them with an metric measuring the errors of prediction of the models.

7.6.1 Human evaluations

For both *TSD* and *REMI* tokenizations, we selected about 130 prompts of 4 bars from the test subset, and generated continuations of 512 tokens with all models. We gathered nine participants, among which seven are musicians, to evaluate the results. They were asked to open the MIDI files with Digital Audio Workstation (DAW) softwares such as Logic Pro or Ableton, play each track individually and select the best one on four criteria: 1) fidelity on pitch scale and rhythm regarding the prompt; 2) correctness, i.e. featuring good note succession and harmony; 3) coherent diversity, i.e. featuring diverse correct melodies and harmonies; 4) their overall subjective preference. The advantage of using DAW software is twofold: it allows to conveniently listen the different tracks, and compare them by also visualizing them as pianorolls. You can find more details on the human evaluations in Appendix B.3, and all the generated samples used on the demo website (Section 7.1).

	tok/sec (↑)		beat/sec (↑)		note/sec (↑)		Voc. sampled (↑)	
Strategy	TSD	REMI	TSD	REMI	TSD	REMI	TSD	REMI
No BPE	40.2	43.8	4.5	9.9	10.6	10.9	100%	100%
BPE 1k	78.5	67.0	**13.0**	17.9	20.8	16.8	100%	99.9%
BPE 5k	99.1	83.9	12.8	**30.0**	26.7	20.7	100%	99.8%
BPE 10k	97.5	85.4	12.5	26.0	26.3	21.3	99.9%	99.9%
BPE 20k	**115.6**	**91.7**	12.9	24.9	**31.5**	22.7	99.4%	99.7%
PVm	59.3	58.1	8.2	12.2	15.9	14.9	99.3%	99.0%
PVDm	89.7	87.3	11.4	17.1	24.7	23.4	75.9%	74.3%
CPWord		75.8		15.2		19.0		76.7%
Octuple		-		14.3		**58.5**		57.4%

Table 7.3: Inference speeds on a V100 GPU and a batch size of 64 and ratio of vocabulary sampled during generation. For tok/sec (base tokens per second), the results account for "basic" tokens of note attributes and time. Tok/sec for Octuple is not calculated as the equivalent number of base tokens is not clearly deducible.

7.6.2 Results and analysis

The TSE error ratios, introduced in Section 6.4, and human preferences results are reported in Table 7.2.

As BPE creates new tokens that combine several types and values, it also increases the overall complexity of music modeling when using these tokens. Thus, we initially expected the generative models to predict higher ratios of errors. But surprisingly, it decreases these ratios, except for the time errors with *REMI*. These results show that the models easily capture the information of the tokens, and that the vocabulary can be scaled consequently.

We gathered total of 814 human preferences, with a bit more than 100 for each criteria for *TSD* and *REMI*. There is a clear preference for results with BPE, especially with vocabularies of 10k and 20k tokens. Baselines without BPE still accounts for a few preferences for the fidelity and correctness criteria, but are less preferred overall, especially with *REMI*. We note that the *PVDm* baselines show competitive preferences with BPE baselines, especially for diversity. Octuple and CP Word perform poorly on the other hand, which is not surprising as they are not 100% autoregressive, and the sense of the combinations of tokens sampled unconditionally is likely to degenerate, especially when time and notes are handled all at once. Overall, BPE helps models to generate more natural and pleasant music. The new contextually learned embeddings may represent richer and more explicit information, helping to model the musical information.

Besides results quality, the second big advantage of BPE is the inference speed increase. We reported three inference metrics - tokens, beat and note per second - in Table 7.3, along with the proportion of the vocabulary ever sampled by the models.

We first highlight that models with BPE, up to the maximum vocabulary size tested here, do use most of the tokens of the vocabulary, with a slight decrease as the vocabulary grows. This also supports that the vocabulary can easily be scaled while keeping tokens that are still used by the models.

BPE increases all inference speeds measured by at least two times, even with small vocabularies. We note that the increase of beat/sec does not increase linearly with the vocabulary size, which also indicates that the models predict a higher number of notes per beat. CP Word, despite having low tokens per beat ratios (Table 7.1), yields lower tokens per second generation speeds, due to the additional input and several sampling steps.

	Genre (↑)		Artist (↑)	
Strategy	TSD	REMI	TSD	REMI
No BPE	0.836	0.796	0.907	0.876
BPE 1k	0.882	0.871	0.934	0.920
BPE 5k	0.901	0.875	0.933	0.925
BPE 10k	**0.904**	0.869	**0.937**	0.922
BPE 20k	0.851	0.877	0.909	0.923
PVm	0.853	0.810	0.905	0.886
PVDm	0.875	0.818	0.914	0.893
Octuple	-	**0.923**	-	**0.941**

Table 7.4: Average accuracy of classification models.

	Isoscore (↑)				PCA ID (↑)				FisherS ID (↑)			
	Gen / Maestro		Pt. / MMD		Gen / Maestro		Pt. / MMD		Gen / Maestro		Pt. / MMD	
Strategy	TSD	REMI	TSD	REMI	TSD	REMI	TSD	REMI	TSD	REMI	TSD	REMI
No BPE	0.899	0.883	0.925	0.730	62	66	44	45	5.4	5.2	8.1	7.9
BPE 1k	0.919	0.953	0.981	0.986	100	99	113	102	7.3	6.7	15.5	12.2
BPE 5k	0.965	0.962	0.989	0.989	131	119	145	119	9.0	8.6	16.7	13.7
BPE 10k	0.973	0.973	0.991	0.993	132	118	164	118	9.8	9.6	18.3	15.2
BPE 20k	0.976	0.981	**0.993**	**0.995**	**146**	**122**	**187**	**137**	**10.8**	**10.5**	21.4	16.9
PVm	**0.987**	**0.989**	0.961	0.961	71	67	52	52	7.1	6.8	13.9	14.7
PVDm	0.945	0.942	0.898	0.909	38	39	98	87	4.4	4.4	**24.1**	**22.8**

Table 7.5: Isoscore, and intrinsic dimension (ID) estimations. Gen. corresponds to the causal generative models, Pt. to the pretrained bidirectional models.

7.7 Classification

For our classification task, we experiment with the MMD dataset [49]. Similarly to Section 6.6, we deduplicate the songs of the dataset, but this time by keeping the MIDIs with a $\frac{4}{*}$ time signature and at least three tracks. The process is described in Appendix A.1.

To handle multiple tracks, we placed `Program` tokens before each `Pitch` token of each note to specify its instrument. This strategy is similar to REMIPlus [164].

We perform genre and artist classification, from the 40 and 100 most present genres and artist in the MMD dataset. The results, reported in Table 7.4, show that BPE improves the models performances compared to the baselines without BPE, and outperform the other token combination techniques. The models seem to benefit from larger vocabulary sizes. It however shows limits, as the accuracy does not increase from a vocabulary of 10k to 20k tokens. The Octuple baseline outperforms the others. Here, the model is bidirectional (no attention mask) and we do not sample from multiple distributions. Our assumption is that the reduced sequence length allows to carry more information within a same number of tokens, allowing the models to capture more easily the global melody, harmony and music structure and directly improving their performances.

It concurs with our results, and is explored in the next section by analyzing the geometry of the learned embeddings.

7.8 Learned embedding spaces

We have shown so far that BPE improves the results of music modeling on the generation and classification tasks. Our assumption is that, non-only the reduced sequence length

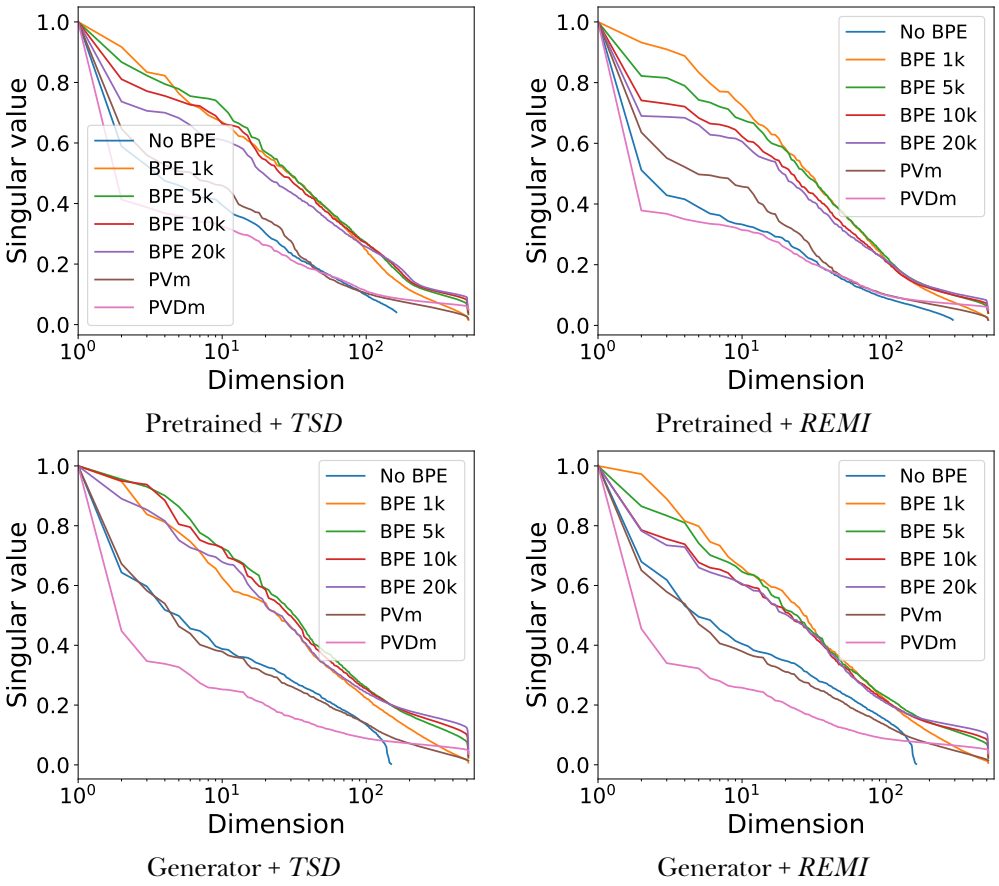


Figure 7.3: Normalized singular values of the embedding matrices. Pretrained refers to the (bidirectional) classification models after pretraining, and generators to the (causal) models for generation after training.

allows to pack more information (longer music piece) within the same number of tokens, but mostly the vocabulary can be scaled while making the model efficiently learn and use the embedding representations of the newly created tokens with BPE.

The embedding space, i.e. the way LMs learn to represent tokens into a continuous space $\mathbb{R}^d$ of d dimensions, has recently been studied [60, 15, 20]. More specifically, it has been shown that most LMs learn anisotropic embedding distributions [52, 156], despite that their isotropy have been linked to improved performances on downstream tasks [65, 178, 15, 116, 150].

Isotropy is a measure of the uniformity of the space occupied by a manifold across all dimensions. A high isotropy is associated with a uniform variance of the distances between the points of the manifold across all dimensions. In our case the manifold is the collection of contextually learned embeddings $X \in \mathbb{R}^{V \times d}$ where V is the vocabulary size and d the model/embedding dimension. An isotropic embedding space can be viewed as a space where the embeddings are uniformly spaced with uniform densities.

Isotropy is often estimated by different ways: singular value decomposition (SVD) [15, 60, 116, 178], intrinsic dimension [20], partition function [10, 131], average cosine similarity [52]. We chose the two firsts, along with IsoScore [162] which alleviates some of their shortcomings, to have results that complement themselves. We did not measure isotropy on models using embedding pooling, as it would be untractable considering the very large number of possible embeddings, and that the low frequency of the majority of them would result in unreliable results.

SVD, for which results are plotted in Figure 7.3, allows to visualize the relative domination of some dimensions. Baselines without BPE and *PVm* and *PVDm* show quicker singular value decays, indicating that fewer dominate, whereas baselines with BPE show more uniformly distributed values.

The intrinsic dimension is an estimation of the minimal number of dimensions n required to represent a manifold in $\mathbb{R}^d, d > n$. It links with isotropy in the sense that an isotropic manifold occupies all the dimensions, hence its intrinsic dimension is close to d. We chose to estimate it with the Principle Component Analysis (PCA) [58] and FisherS [3] methods as they are insensitive to redundancy, focus on common similarities and can scale to large number of points and dimensions. As the embedding matrix is often initialized with a stochastic method around the origin, a simple method can estimate high intrinsic dimensions even though the points coordinates have been very little or not even optimized. This can be the case when a large number of tokens has low frequencies or are absent from the training data, such as with *PVDm*. The intrinsic dimensions results are reported in Section 7.8, along with the IsoScores. In all cases, as the vocabulary grows with BPE, the intrinsic dimension increases, the embeddings occupy more space.

IsoScore is an estimation of isotropy based on the distance of the covariance matrix of a Principle Component Analysis (PCA) and the identity matrix, and is normalized between 0 and 1. As for the intrinsic dimension, the isoscore grows with the vocabulary size, indicating that the embeddings are more uniformly distributed.

We also note that similarly to models trained on natural language [52], our bidirectional models learn more isotropic embeddings than causal (generative) ones. Appendix B.4 depicts UMAP representations of the embedding, showing the narrow cones and clusters they form.

7.9 Conclusion

We showed that BPE can increase the quality of results of Transformer models for symbolic music generation and classification tasks, while significantly improving their efficiency and

inference speed and making better use of their embedding spaces. BPE can be applied on top of any tokenization. The tokenization and decoding times are almost not affected by this extra step, when performed by a well-optimized Rust code. Considering the considerable benefits and low requirements of this technique, we advise anyone using Transformer models with symbolic music to use BPE.

There are still questions that remain uncovered. We showed that 40M parameters models can handle well vocabularies up to 20k tokens with medium-size datasets. We however do not know what are the limits in vocabulary and dataset sizes over which the results might not improve. Moreover, we experimented with BPE, but other common vocabulary building techniques exist, such as Unigram [108] and WordPiece [184]. Recent work on natural language showed that Unigram yielded higher model performances than BPE [16], it might also be the case for symbolic music. Future research will study these questions and hopefully find optimal tokenization guidelines to improve model performances under more various settings.

Chapter 8

Scaling Transformers for autoregressive multitrack symbolic music generation

8.1 Introduction

Multitrack music generation is a challenging task, as it implies a model to grasp complex melodic and harmonic features from multiple instruments, while maintaining structure and rhythm consistency. Moreover, music is multifaceted and is made of hundreds of genres and instruments which are played and arranged by a multitude of ways.

Overcoming this complexity can bring a lot of benefits for music generation tasks. As symbolic music is discrete, this format allows a total control and flexibility for musicians and creators which want to edit and adjust the notes and their attributes. The symbolic representation can then be synthesized into audio, offering to a music producer a high level of control. This represent a significant interest for music creation assisted by Artificial Intelligence (AI). Considering the interest from this public, companies are starting to develop software allowing to use deep learning models directly into Digital Audio Workstations (DAW), both for audio and symbolic content.

Music generation has in the recent years mostly been tackled in the audio domain [106, 1, 30]. Yet, audio samples can only be edited by a fairly limited range of actions, thus reducing their interest for assisted composition. In this context, the user may need to prompt a generative model several times until the result satisfies its expectations, which can be high depending on their level of perfectionism. Additionally, accurate prompts for audio generation can be hard to formulate [1, 86] and this interface of control remains high level and show limitations in the degree of accuracy of specific features that can be expressed with words.

Unfortunately, applications of AI models with symbolic music remains to this day a niche that receives a fraction of the interest and investment of more popular domains such as natural language. As such, there are only a limited number of works on multitrack symbolic music generation. We identify two major gaps in the field, as observed in many of these works:

1. While these works rightfully highlight the advantages of their models, they often do not report the training loss values. As such, it is difficult for researchers to find comparison points to scale their own models, and resort to arbitrary size parameters. Additionally, we cannot know the extent of overfitting in these models;

2. Many do not share their model weights, or in a non user-friendly manner that requires researchers and engineers to spend time and effort to get their hands on ad-hoc code implementations. Consequently, this impedes the reproducibility of

the studies, complicating the assessment of the models performances and their comparison, when feasible.

In order to address these limitations, we introduce SMILE (**S**ymbolic **Mus**I**c** **L**arg**E** music model), a large model for multitrack symbolic music generation. Our contributions are:

- We conduct a study on the scalability of transformer models for autoregressive symbolic music generation, by scaling *SMILE* to 125m, 500m and 1.5b parameters;
- We openly share the models in a user-friendly manner[1], that can be freely finetuned for specific data and usages, along with an interactive demonstration application allowing to easily use the model;
- We analyze the generation performances of the three model sizes, revealing the limitations of our approach as measured by overfitting and consequent biases towards the training data.
- We analyze the distribution of token successions for our music dataset and compare it with those of popular text datasets, highlighting the fundamental diversity difference between these two modalities.

8.2 Related works

Symbolic music generation with deep learning is a field explored from different angles [17]. In addition, most works focus on a single instrument [85, 87]. Multitrack symbolic music generation poses a greater challenge due to the necessity for models to grasp multiple simultaneous notes of different instruments.

MuseGan [44] can be considered one of the first attempts to train a deep learning model to generate multitrack music. The model is a Generative Adversarial Network representing five simultaneous instruments as pianoroll matrices. *LakhNES* [41] is a Transformer model [177] pretrained on the Lakh dataset [148] and finetuned on Nintendo Entertainment System (NES) music, capable of generating four different instruments. Multitrack Music Machine (MMM) [48] is a *GPT-2* based Transformer trained for music inpainting handling all MIDI programs. Multitrack Music Transformer (MMT) [43] is transformer based on a multi-token strategy merging embeddings to shorten the input sequence length. *FIGARO* [164] is also a multitrack model offering controllable features to condition the generation on.

LakhNES counts 41 million parameters, MMM and MMT 20 million parameters, and *FIGARO* counts 44.6 and 43.7 million parameters for its description-to-sequence and VQ-VAE components respectively.

In these previous works, most of the models are relatively small compared to their natural language counterparts, which can reach up to billions of parameters. We can hence wonder how they would scale to larger sizes. We also note that none share the training losses of their models, and only *LakhNES* and *FIGARO* are openly shared[2].

To address these gaps, we conduct experiments by scaling the sizes of transformer models for multitrack music generation and share the models in a user-friendly way.

[1] Code: https://github.com/Natooz/scaling-transformers-music-gen
Models: https://huggingface.co/Natooz
Demo: https://huggingface.co/Aubay/SMILE

[2] The weights of MMT [43] were no longer publicly available when conducting this work in February 2024

Model size	Embedding. size	Num. layers	Num. heads
125m	512	24	8
500m	960	32	12
1b5	1536	40	16

Table 8.1: Differentiating size parameters of SMILE.

8.3 SMILE

We introduce *SMILE*, standing for **S**ymbolic **Mus**I**c** **L**arg**E** music model. We create three variants of *SMILE*. This section presents the model's architecture and the data tokenization.

8.3.1 Model configurations

The backbone of *SMILE* is built on the Mistral implementation of the Hugging Face transformers library[3], which features optimizations allowing to increase its attention span. It is a decoder only transformer, i.e. using causal attention, with two major optimization: grouped query attention [169, 2] and sliding window attention [14]. With grouped query attention, the number of heads for keys and values is reduced so that each pair of keys and values is used for several queries. As the number of calculations is reduced, so is the computation time and memory footprint of the model. Sliding window attention is an attention pattern applied as chunks of fixed window sizes across successive layers. Attention being an operation with a time and space complexity growing quadratically with the sequence length, decomposing it into smaller chunks allows to significantly reduce the memory footprint. The combination of these two optimizations allows to considerably increase the model's efficiency, allowing to use it with higher sequence sizes, hence longer musical structure, which in turn can bring better performances thanks to the additional provided context.

We created three variants of *SMILE*, counting 125 million, 500 million and 1.5 billion parameters. Although these sizes are relatively small compared to popular models for natural language [176, 94], there is at the time of writing no open model nor research work in the field of symbolic music that match comparable sizes. Our strategy is to start with a parameter size that aligns closely with those found in the existing works, and to subsequently scale it by a factor of three, twice. The results show that these sizes are large enough for the models to effectively learn from the data.

Their differentiating parameters are reported in Table 8.1. For each of them, the feedforward size is four times the embedding size, the number of key/value heads is half the number of query (attention) heads, and the sliding window is covers 384 positions[4].

8.3.2 Tokenization

We use the TSD tokenization from MidiTok [55]. This tokenization is identical to the MIDI-Like tokenization [137] used in Music Transformer [85] or MT3 [62], but uses explicit `Duration` tokens instead of `NoteOff` tokens indicating when a specific note ends. We prepend a `Program` tokens before each `Pitch` token to indicate the instrument of each note. We represent the pitch values from 24 to 109, we downsample the velocity values into 24 distinct values equally spaced from 0 to 127 included, and a time downsampling

[3] https://huggingface.co/docs/transformers/model_doc/mistral

[4] The highest attention causality length for the smallest model hence covers 9216 positions, which slightly more than twice the maximum sequence length experimented in this paper (4096).

Data subset	Num. tokens	Avg. num. beats	Avg. num. notes
Pretrained	2.48b	139.3±90.7	1312.1±498.5
Pop	208m	106.3±49.1	1173.7±406.2
Jazz	24m	114.3±64.3	1162.4±290.0
Rock	155m	129.3±69.3	1368.9±467.8

Table 8.2: Total number of tokens in the dataset after data augmentation and encoded with BPE, and average number of beats and notes per training sample.

for `TimeShift` and `Duration` tokens of 8 frames per beat (fpb) for values between 0 to 1 beat, 4 fpb for values from 1 to 2 beat, 2 fpb for values from 2 to 4 beat and 1 fpb for values from 4 to 8 beat. This allows to represent notes durations down to the thirty-second note, however triplets might be represented as non-consecutive note successions separated by rests. `Tempo` tokens are added for each tempo change present in the original files, that we represent with 32 distinct values equally spaced from 50bpm to 200bpm. We add twenty unused tokens are added for potential finetuning purposes. Finally, we follow recent research and trained the tokenizer with Byte Pair Encoding (BPE) [56] to build a vocabulary of 30k tokens that will reduce the sequence lengths.

8.4 Experimental setup

We trained all three model variants for autoregressive symbolic music generation, in order to study the scalability of this type of model. We introduce here the experimental setup.

8.4.1 Data

We trained and evaluate *SMILE* on the MetaMIDI Dataset [49], which is to this day the largest MIDI dataset available for research. It contains 436k MIDI files. Much however are multiple covers of the same music. A music piece can have more than twenty MIDI covers, with some of them sometimes almost identical. Using the whole dataset to train a model would then induce a data imbalance. In order to train *SMILE* with a more evenly distributed dataset (song-wise), we deduplicated each MIDI-audio matches provided by the metadata by computing the matching of the weighted bipartite graph we built from them. We end up with a total of 51.1k unique MIDI files.

In our experiments, we randomly subtract 2% and 5% of these files the validation and test subset respectively, the rest constitutes the training subset. Data augmentation is performed on the training set by increasing and/or decreasing the pitches by one octave, the velocities by one or two values, and duration by two value.

The total number of tokens that represent the data, for the whole dataset and for subsets of specific genres used in Subsection 8.5.3, and the average number of beats and notes per sample are reported in Table 8.2.

8.4.2 Training

Training a large model can be a challenging task depending on the the compute and memory capacities of the hardware used. Even with Graphic Processing Units (GPU) featuring an amount of memory considered large today, e.g. 80GB, many considerations must be taken when training a model on it. The model parameters can take a large proportion of the memory just by themselves. The gradients for each token in the input sequence must be also be stored, along with the optimizer states. To handle such memory usage, we use the ZeRO (Zero Redundancy Optimizer) [151] stage 2 principle: the model

parameters are redundant across all devices, but the data, gradients and optimizer states are partitioned across all devices. We use the DeepSpeed [155] library implementation.

The models are trained with a batch size of 64 samples, sequence lengths of 2048 tokens for 20 epochs. The number of beats and notes per training samples are reported in Table 8.2. With an average training sample of almost 140 beats, *SMILE* is the music model trained with highest context length to our knowledge, allowing it to condition its predictions based on musical content more distant in time.

The AdamW optimizer is used with a $6e-5$ learning rate and a warmup corresponding to 10% of the total training (2 epochs), weight decay (0.01), gradient norm clipping (3) and dropout (0.1). The models are trained on A100 SXM4 (80GB) GPUs: two for the 125m and eight for the 500m and 1b5 models, with bfloat16 mixed-precision.

8.4.3 Music generation

To analyze the performances of the models, we generate samples from the first 2000 files of the test set, and generate the next 2048 tokens. We use a 0.8 temperature and sample from the set of tokens with the top-8 probabilities [54], as these parameters allow to slightly reduce the variance of the results while keeping a degree of diversity.

To benchmark our models against existing ones, we include comparative analysis with outputs from *FIGARO* [164], a recently proposed model for symbolic music generation offering several levels of control, and one of the few openly available[5]. We use the "expert" checkpoint.

8.5 Results

8.5.1 Overfitting dynamics

Our first observation highlights the rapid overfitting of the models, as shown by the losses depicted in Figure 8.1. The cross-entropy loss used here can be seen as the distance between the probability distribution predicted by a model, for the next token in our case, and the expected probabilities as a one-hot distribution centered on the expected next token [123]. A lower loss indicates that the model predicts the next expected token with greater certainty, and thus correlates with accuracy. The absolute value of the loss naturally tends to be greater with the number of categories, i.e. the vocabulary size in our case, which is 30k.

For the 500m and 1b5 models, the validation loss starts to increase after 35k training steps, at approximately 1.7 epochs. The 125m model demonstrates a lower validation loss peaking at 72.5k training steps, which corresponds to 3.45 epochs, and a lower final validation loss.

The training losses for the 500m and 1b5 models approach 0, indicating near-perfect predictions of the ground-truth next tokens in the training data. In other words, the models learned the training data, as supported by their higher losses on the validation subset. Subsequently, we will next assess their capability for data reproduction.

8.5.2 Measuring training data reproduction

Measuring the similarity between two data subsets presents a complex challenge [50]. No universal approach allows to perform this task, but each type of data has to be treated considering its specific features. While some studies propose ranking methods operating at

[5]We initially wanted to include MMT [43] in our study, but the model parameters were not longer publicly available at the time of our experiments.

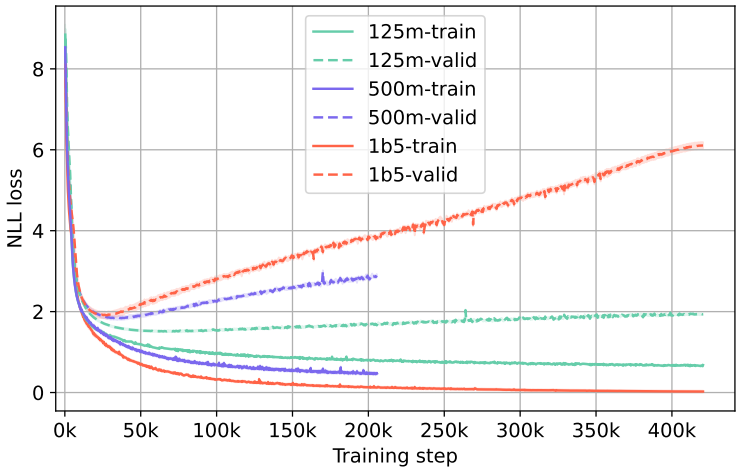


Figure 8.1: Training losses of the *SMILE* models.

	LCStr avg.	LCStr max. (↓)	Lev. avg.	Lev. min. (↑)
SMILE 125m	1.10±0.14	10.09±9.21	0.99±0.00	**0.90±0.07**
SMILE 125m lvl	1.00±0.14	08.21±5.12	0.99±0.00	0.88±0.08
SMILE 500m lvl	0.95±0.17	**8.18±5.41**	0.99±0.00	0.88±0.06
SMILE 1b5	1.19±0.13	106.04±381.22	0.99±0.00	0.86±0.21
SMILE 1b5 lvl	0.98±0.18	11.17±25.29	0.99±0.00	0.88±0.07

Table 8.3: Similarity metrics between 100 results generated by *SMILE* baselines (2048 tokens) and the non-augmented training data. Lev stands for Levenshtein distance.

the MIDI level to measure their similarities [92], such metrics do not directly quantify the similarity between a model's raw predictions and its training data. Consequently, we resort to measure the similarity between the sequences of generated tokens and the sequences of tokens from the training subset. To do so, we use the Longest Common Substring[6] (LCStr) and the Levenshtein distance as an Edit distance. This approach places greater emphasis on successions of symbols without strictly relying on their positions within the compared sequences. We compare the model sizes, and analyze the results generated from their final checkpoints and the checkpoints exhibiting the lowest validation losses (*lvl*).

However, comparing these subsets with the whole training data is highly compute-intensive, even for high-performance processors. To make these computations tractable, we limit our comparison to the non-augmented training samples (47.5k sequences) and 100 generated samples. While imperfect, these results, reported in Table 8.3, still give us insights on the data reproduction.

The average maximum LCStr length measured increases with the model size. We highlight in particular the high standard deviations, as some generated samples share no similarity with any training sample, while others have a LCStr length almost equal to the number of tokens generated (2048), thus nearly replicating a sample from the data. Conversely, the average minimum Levenshtein distance remains relatively consistent accross samples from the final and *lvl* checkpoints. Larger models also exhibit lower minimum Levenshtein distances, however the gap between model sizes is smaller than for maximum LCStr length. However, we note an average LCStr length value for the 1b5 *lvl* and 500m *lvl* checkpoints inferior to 1, which we interpret as one of the effect of their repetitiveness observed in Subsection 8.5.4, as some generated samples might note

[6] Not to be confused with the Common Longest Subsequence.

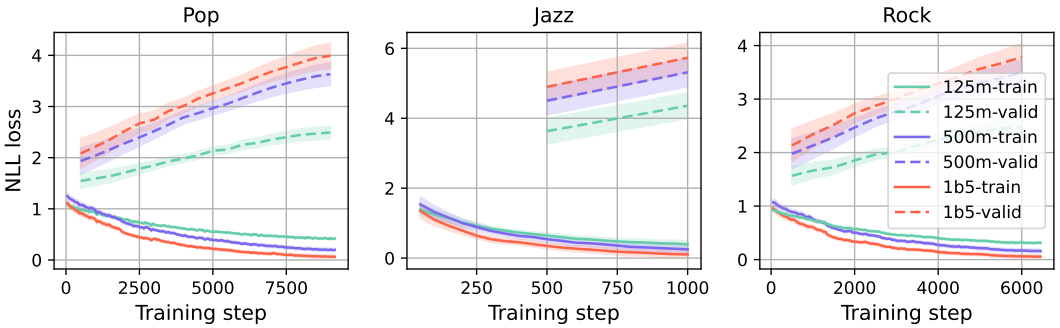
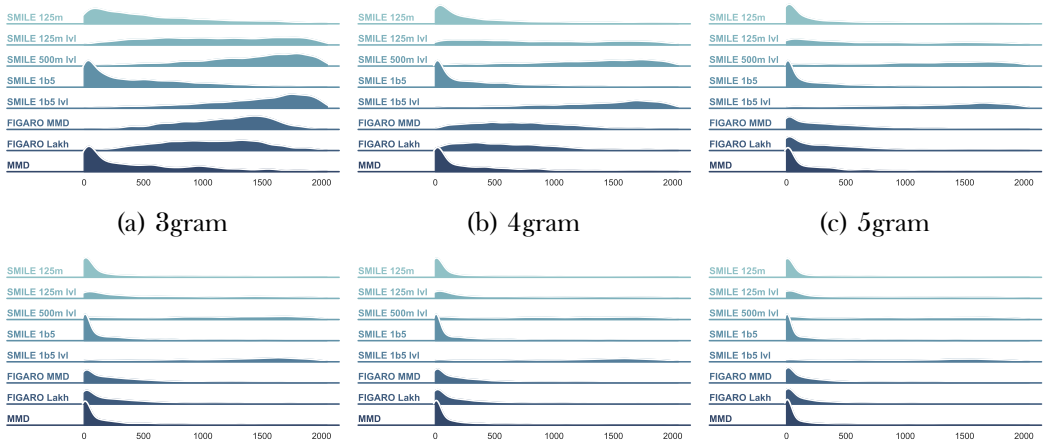


Figure 8.2: Losses when finetuning the *SMILE* models from the *lvl* checkpoints on music genres.



(a) 3gram (b) 4gram (c) 5gram

(d) 6gram (e) 7gram (f) 8gram

Figure 8.3: Distributions of the number of generated tokens per sample (2048 tokens) part of a ngram that is repeated at least four times not consecutively. Additional figures for other ngrams can be found in our code repository[1].

share a single token with most training samples. All the measures are available in our code repository[1].

8.5.3 Effect on finetuning

To measure if the loss momentums previously observed can also be found for smaller subsets of data and shorter trainings, we finetuned the models pretrained at the *lvl* checkpoints on three music genres: pop, jazz and rock. The MIDI files constituting these subsets are selected from the metadata provided from the MetaMIDI dataset. The number of tokens of each subset is reported in Table 8.2. The models are finetuned with the same training parameters than for the pretraining, except for the number of epochs which is 10.

The losses, depicted in Figure 8.2, show similar divergences between the training and validation data. However, it's noteworthy that the final validation losses are lower on these subsets than for the whole dataset. This may be attributed to the higher intra-similarity among the training samples within each genre subset, than among samples across the entire dataset.

	2gram (↓)	3gram (↓)	4gram (↓)	5gram (↓)	6gram (↓)	7gram (↓)	8gram (↓)	9gram (↓)	10gram (↓)	11gram (↓)	12gram (↓)
MMD (reference)	0.2%	0.0%	0.1%	0.0%	0.2%	0.0%	0.0%	0.0%	0.0%	0.0%	0.1%
SMILE 125m	0.8%	0.6%	1.0%	0.4%	1.3%	0.2%	0.9%	0.6%	0.8%	0.2%	1.2%
SMILE 125m lvl	1.1%	0.7%	1.3%	0.6%	1.6%	0.4%	1.5%	0.7%	1.3%	0.4%	1.9%
SMILE 500m	-	-	-	-	-	-	-	-	-	-	-
SMILE 500m lvl	2.3%	2.0%	3.9%	1.6%	4.8%	1.4%	4.6%	2.8%	3.4%	1.1%	6.7%
SMILE 1b5	**0.3%**	**0.2%**	**0.5%**	**0.2%**	**0.4%**	**0.1%**	**0.5%**	**0.1%**	**0.3%**	**0.1%**	**0.4%**
SMILE 1b5 lvl	1.4%	1.1%	2.1%	0.9%	2.6%	0.7%	2.4%	1.4%	2.0%	0.6%	3.6%
FIGARO MMD	4.6%	5.1%	4.6%	4.7%	4.9%	4.4%	4.4%	4.8%	4.4%	4.3%	4.7%
FIGARO Lakh	4.8%	5.4%	4.7%	4.8%	5.2%	4.5%	4.5%	5.1%	4.5%	4.4%	5.0%

Table 8.4: Percentage of generated tokens part of ngrams repeated at least four times successively.

8.5.4 Repetitions in predictions

While listening to the generated results, the first noticeable difference felt between the different baselines is their repetitiveness: non-overfitting models tend to repeatedly predict the same successions of notes. Although repetition is a fundamental aspect of music, there is a reasonable expectation for a degree of variation and evolution, respecting a musical structure. However, in this instance, the non-overfitting models get stuck in repetitive loops, which is in most cases undesirable [82, 185].

In Figure 8.3, we depict the distributions of the number of generated tokens per sample (2048 tokens) that are part of ngrams repeated at least four times non-consecutively. Shorter ngrams, e.g. 3grams or 4grams are naturally more repeated as they represent shorter token combinations. In every cases, overfitted models produce significantly fewer repetitions resulting in more diverse token successions.

In Table 8.4 are reported the percentages of generated tokens that are part of ngrams repeated at least four times consecutively. They represent tiny portion the training data. While the results from the generated tokens remain relatively low, the tokens generated from *lvl* checkpoints show between four to five times more consecutive repetitions compared to those from final checkpoints.

8.5.5 Interactive demo application

Even though, a subjective evaluation is the best way to assess the quality of a generated music, we opted to not do conduct one as it is out of the scope of our study. However, we made available a web-based demonstration application[1] that allows to easily use the model, with an interactive pianoroll interface. We anticipate this tool will assist readers to better assess the generation performances of the models. The source code is open and can freely be reused for other models. Additionally, all the generated results can be found with the models[1].

8.6 Discussion on autoregressive music generation and exposure bias

Symbolic music generation is currently mostly addressed with Transformer models which are used autoregressively. As such, these models are trained with teacher forcing to predict the next token at each position of the input sequence, conditioned on the previous ones. This way of processing information has mostly been borrowed from the research in natural language processing [26], and applied to music. While the method works in practice, it also suffers from a discrepancy between the model's performance on the training data and the results generated from unseen test data. This effect is called the exposure bias [154] and is the subject of many research trying to mitigate it [114, 22, 71, 110], although the

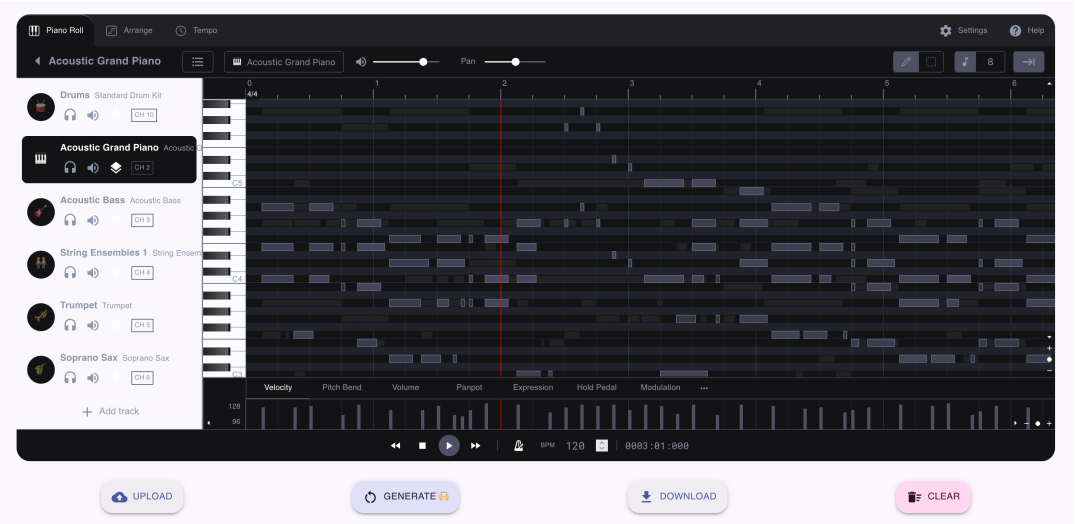


Figure 8.4: Screenshot of the user interface of the SMILE Hugging face Space.

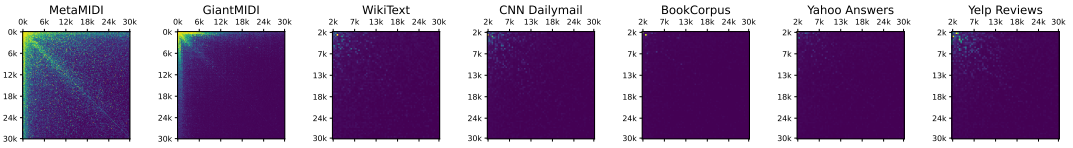


Figure 8.5: Co-occurrence matrices of the next tokens within music (MIDI) and text datasets (others). The matrices are downsampled with maxpooling to 200x200 for MetaMIDI and 70x70 for others, and the first 2k tokens are omitted for text datasets for better readability. The matrices are normalized, no scale is given. The raw values are in our code repository[1].

current tendency to tackle it is to use larger models and training datasets and finetuning methods [29].

While exposure bias is documented for natural language [9, 188], we can wonder how its affects music. Our previous results give some initial insights. Yet, language and music are different modalities that convey information in different ways. Text tokens usually represent words or subwords [167], that represent rich semantics, that the models learn contextually [143, 37, 52]. As such, we can expect for each specific token to be followed by a restricted set of tokens that contains contextually relevant information, thus forming pertinent sentences. For music however, the dynamics explaining the token successions are unclear.

When generating music autoregressively, for either continuation, inpainting or arrangements, a huge number of combinations of notes forming a pleasant music while conditioned on the provided context can be imagined. On the other hand, natural language puts stronger constraints to use specific words because of their semantics, hence resulting in potentially more limited token combinations.

To measure the "intra-compatibility" between tokens, we computed the co-occurrence matrices of the token successions within the MetaMIDI and GiantMIDI datasets along with five popular text datasets for comparison: WikiText, CNN Dailymail, BookCorpus, Yahoo Answers and Yelp Reviews. For each dataset, we trained a tokenizer with byte pair encoding for fair comparison. The matrices are depicted in Figure 8.5. We instantly notice that the music datasets have more sparsely distributed occurrences, whereas the distributions of the text datasets are much more imbalanced with almost all the weight

	Avg. unique next tokens	Avg. entropy	Avg. variance
MetaMIDI	540.99±823.38	4.59±1.16	0.0006±0.0038
GiantMIDI	549.49±908.02	5.54±0.80	0.0000±0.0002
WikiText	291.12±763.94	3.39±1.19	0.0055±0.0207
CNN Dailymail	417.95±907.86	3.57±1.24	0.0052±0.0209
BookCorpus	604.87±1199.10	3.59±1.17	0.0041±0.0185
Yahoo Answers	290.66±731.08	3.65±1.07	0.0031±0.0138
Yelp Reviews	200.01±589.82	3.29±1.06	0.0053±0.0191

Table 8.5: Token succession metrics for music (MetaMIDI and GiantMIDI) and text datasets (others), computed on the distributions next tokens in the co-occurrence matrices. Avg. stands for average and the variance is normalized.

being on the first tokens in the vocabulary, which are likely to be common prepositions. This suggests that the possible token successions in music are much more diverse, and still keep high occurrences for all the tokens in the vocabulary.

This observation is supported by the Table 8.5. The distributions of next tokens show a higher average entropy for music than language, and has a higher number of unique next tokens. Considering the distribution imbalance found for the text datasets, their average numbers of next tokens would have been likely much lower if we omitted the first tokens in the vocabulary. The variance of music distributions are however lower, indicating that their counts of unique next tokens are more consistent than for language.

These results show that the formation of successions of tokens in music is much more diverse than for text. The token of the latter are followed by smaller sets of tokens. Consequently, a music model is likely to predict a higher diversity of combinations of tokens. This diversity might however make the generation task harder for music models, as the tokens of the vocabulary represent notes that can more easily be combined together than tokens from text vocabularies that are less compatible with one another. A greater number of "compatible" next tokens implies that predicting a succession of good ones that form relevant combination of notes is less straightforward, when the model is trained to only predict the next one, thus suffering from exposure bias. It might also explain the tendency of non-overfitting models to fall into loops of repetitive predictions as repetitions are common in the data and thus more susceptible to be predicted [71].

8.7 Conclusion

We investigated the scalability of Transformer models for autoregressive symbolic music generation, up to 1.5 billion parameters. We found that in the simplest training objective of music modeling, these models tend to overfit quickly. Our analysis reveals the biases in data reproduction of overfitting models and highlights the repetitions made by non-overfitting models.

By analyzing the distributions of token successions in the MetaMIDI dataset and popular language datasets, we show that the music tokens exhibit a larger diversity in their successions. This fundamental difference likely contributes to the overfitting and exposure bias we observed. The way music tokens are formed should be considered for music modeling tasks. Therefore, our forthcoming work will focus on experimenting with different training objectives, and generation techniques conditioning generation on future tokens. This approach aims to mitigate this bias with the goal of training generative models with better extrapolation capacities.

Finally, we introduce *SMILE*, the series of models used for our study, which we openly share with the research and engineering communities. We believe that these resources will facilitate further reuse and investigation in the field of symbolic music generation.

Part III

Conclusion

Chapter 9

Conclusion

In this thesis, we explored the realms of symbolic tokenization. Our two main goals were to lower the barriers of entry into the field, and to improve models performances and efficiency. We will conclude this work by giving a summary of our contributions, followed by future research directions that could further contribute to the field.

9.1 Summary of contributions

As symbolic music is a discrete modality, recent researches have mainly tackled symbolic music modeling tasks with discrete models such as Transformers. Such models require to receive data as the form of sequences of distinct tokens. To use them for symbolic music, one must hence serialize, i.e. tokenize, music into tokens, that can then be detokenized back to notes and music. In the previous works, the tokenization process was always performed with code written by the authors. Hence, most tokenizations were done differently, making comparisons difficult, and forcing other people to adapt code from repositories. To address these problems, we developed MidiTok. MidiTok is a Python package providing easy to use, yet highly flexible and configurable, tokenization methods. It comes with a wide variety of features, allowing users to easily experiment and design their own tokenizations. MidiTok gained traction in the MIR community, and benefited from its feedback to improve itself throughout the years. It is now an established tool, which we hope will continue to improved and bring features to the community.

As music tokenization can be performed by different methods, it can be difficult to choose how to do it. In the Chapter 6, we explored different tokenization schemes, especially on the representation of time, note duration and MIDI instrument, and showed that the results can vary following the task at hand. This chapter was also the appropriate place to discuss of what are the tokenization choices, by decomposing music tokenization. We only explored three of seven of them.

Tokenizing music by only using note attributes and time tokens results in relatively long token sequences, and tokens that do not carry meaningful musical information. These are significant problems as the complexity of Transformers models grow quadratically with the input sequence length, and that they deeply rely on the contextually learned information of the tokens to process data and perform the tasks they are trained for. Previous works address the first problem by merging tokens and embedding vectors. We showed that Byte Pair Encoding, a technique commonly used in natural language processing, can address these problems when applied to music. It allows to create vocabularies with new tokens that can represent whole notes and note successions, resulting in much reduced token sequences lengths. The models trained with such vocabularies perform better, especially for music generation, while having a reduced inference time. The tokenization time is however slightly higher due to the BPE encoding and decoding, fortunately this is contained thanks to efficient Rust implementation.

Finally, we scaled Transformer models up to 1.5 billion parameters for symbolic music generation, and release SMILE, the series of associated models. We showed that these models quickly overfit. We measured the data reproduction of these models and found that they can moderately reproduce significant parts of the training data. Yet, this overfitting allows models to predict results of higher quality and far fewer repetitions. We close this chapter with a discussion on the effects of exposure bias on symbolic music generation, highlighting that the token successions formation of symbolic music is more diverse and sparse than text and might explain its the greater discrepancy between train and test results.

9.2 Future directions

We share here some directions that we think could help bring new knowledge to the field, and lead to better model performances.

MidiTok is a very useful tool. In its current state, it could however benefit from additional improvements. One of its biggest bottleneck is the reliance on Python code for operations with high time complexity $\mathcal{O}$. The loading of MIDI files is already performed in C++. Additional operations such as the preprocessing of the MIDI files and parsing of the track and MIDI information could also benefit from being performed in a more efficient programming language. We are also sure that the community will find ways to further improve it in the future.

In the Chapter 6, we only experimented with four of the music tokenizations dimensions we introduced. We consider that Chapter 7 delve into the last dimension. There are still tokenization design choices that remain to be explored. One important in particular is the downsampling. By downsampling, we effectively lose the performance information present in most MIDI dataset. While this step is necessary and helps the models to learn more efficiently, we are not aware of any work experimenting comprehensively with different granularity levels. This could bring more insight and help to find more accurate tradeoff between performance information and model's learning.

In Chapter 6 and Chapter 7, we showed that the explicit information carried by the tokens helps the models to learn. We believe that the expressivity of the musical content is essential in a model's learning performances. Ultimately, including more expressive tokens would be likely to bring better performances. In particular, tokens explicitly describing the texture and structure of music the could bring very useful information to the models. Some works are being conducted on the analysis of texture in symbolic music [31, 63], that could be used for symbolic music modeling.

In Chapter 7, we did not deeply explored the nature of the tokens created by BPE. Yet, as they represent recurrent successive information, analyzing their meanings could give us information on the nature of the data. Furthermore, other token aggregation methods similar to BPE could be applied to symbolic music. Namely, Unigram [108] and WordPiece [184] are commonly used in NLP. Considering they are specifically designed for words, we could think of similar methods more suited for symbolic music.

Part IV

Appendix

Appendix A

Data

A.1 MMD preprocessing

With more than 436k MIDI files, the MMD dataset contains many covers of the same songs, corrupted files and poor quality performances. In order to train our models with a well balanced dataset composed of pieces of good quality, we performed a preprocessing step to deduplicate each song, and keep the best versions.

Each MIDI file has a matching score with audio files linked to Spotify and MusicBrainz ids. Hence, each MIDI file can have high matching scores with several different ids, and an Spotify or MusicBrainz id can have have high matching scores with several different MIDI files.

In order to deduplicate the songs, we represented the matching scores as a weighted bipartite graph, and computed its matching. To build the graph, we first read each MIDI file, add it to the graph if it is not corrupted, and satisfies some other requirements specific to our experiments (e.g. on the number of tracks or time signatures). The opposite nodes are the Spotify ids, and the edges (weights) are the MIDI-audio matching scores. When the graph is complete, we compute its matching in order to have the maximum sum of the weights between pairs of distinct and unique MIDIs and Spotify ids. After matching, we end up with 30k distinct MIDI files.

Finally, among the 128 possible MIDI programs featured in the General MIDI 2 protocol, a large proportion are very similar in the way they are played and sound. For example, there are eight sorts of pianos, that are very similar. A model could struggle to capture the differences between these similar programs, especially considering that the program choices were made by humans and potentially subject to bias or subjective preferences. So in order to reduce the overall complexity and avoid the model to deal with distinguishing these instruments, we merge the tracks of the instruments of the same classes, for the twelfth first categories (programs from 0 to 95) except ensembles (programs 48 to 55), and drums, ending up with twelve distinct programs.

Appendix B

Analysis related to Byte Pair Encoding

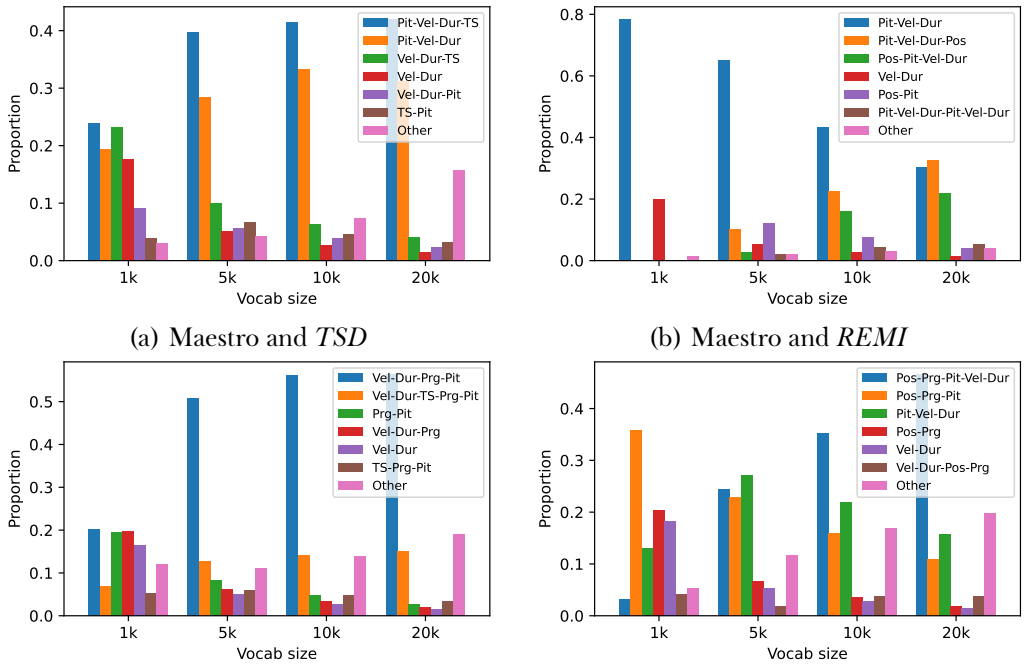
B.1 Proportion of simultaneous notes in common datasets

Table B.1 shows the ratios of notes being played simultaneously (having the same onset and offset times), with the same velocity, for the datasets used in this paper, as well as POP909 [181], GiantMIDI [105] and EMOPIA [88].

The proportion of simultaneous note is low, even with a strong downsampling of their attributes, onset and offset times. Hence, the scope of token aggregation techniques such as in SymphonyNet [118] is arguably limited.

	POP909	**Maestro**	**GiantMIDI**	**MMD**	**EMOPIA**
Ticks	0.014	0.000	0.002	0.143	0.002
Preprocessed (32nd)	0.124	0.129	0.182	0.203	0.124
Preprocessed (16th)	0.175	0.229	0.236	0.222	0.145

Table B.1: Ratio of notes played simultaneously with the same velocity. Preprocessed rows means that the onset and offset times in ticks of the notes have been aligned, to the corresponding portion of bar. For a fair comparison, results for POP909 are for all tracks being merged, and those for MMD are for the unprocessed (vanilla) dataset.



(a) Maestro and *TSD*

(b) Maestro and *REMI*

(c) MMD and *TSD*

(d) MMD and *REMI*

Figure B.1: Normalized distributions of the token types of the BPE tokens, per vocabulary size. Abbreviations in the legend stand for: Pit: Pitch; Vel: Velocity; Dur: Duration; Pos: Position; TS: TimeShift; Prg: Program.

B.2 Types of learned byte pairs

Figure B.1 shows the distribution of token types combinations of the learned BPE tokens. The majority of the learned combinations represents single notes in all cases, except for the case of MMD when tokenized with *TSD*. In this latter case, most BPE tokens begin with *Velocity* base tokens, indicating that the dataset contains a lot of recurrent *Velocity* - *Duration* token successions. With *REMI* however, the *Position* token seems to be more recurrent, showing that the notes have more common onset positions, which is not surprising considering that the MMD dataset features many music of genre known to have repeating patterns. As the vocabulary grows, the combinations tend to be more diverse.

B.3 Human evaluations

We report here the human evaluation instructions given to the participants to assess the generative models:

> Each MIDI file contains several music tracks generated from different Deep Learning models, that are the continuations of the same 4-bars prompt. For each file, you have to choose the best track on several criteria:
>
> - **Fidelity**: the track with the best fidelity (coherent) relative to the prompt, from a tonal and rhythm point of view;
> - **Correctness**: the track with the most correct note successions and harmonies, contrarily to tracks with dissonant notes or unnatural melodies;
> - **Diversity**: the track with the best coherent diversity, i.e. featuring diverse correct melodies, contrarily to a music that would repeat the same

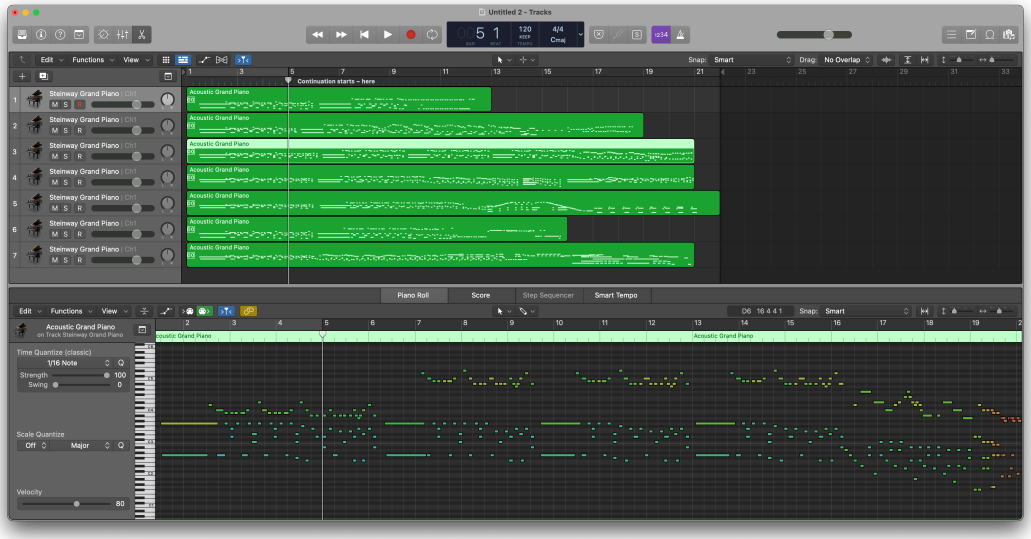


Figure B.2: Example of MIDI file given to participants for human evaluations, opened with the Logic Pro DAW.

note patterns. A "bad" or uncertain music (i.e. non-correct) cannot be considered as diverse;

- **Overall preference**: the track that you overall prefer subjectively;

Do not answer to all for all the files, as the evaluations can be time-consuming. Fix yourself a number of files to evaluate, and randomly pick them from the list.

You will find generated results than can be very similar, even identical sometimes. As such, you might feel uncertain or unable to decide. In such cases, do not answer for all criteria and just skip to the next file. There is no good or wrong answer, you just have to answer subjectively. Trust yourself and trust your musical instinct.

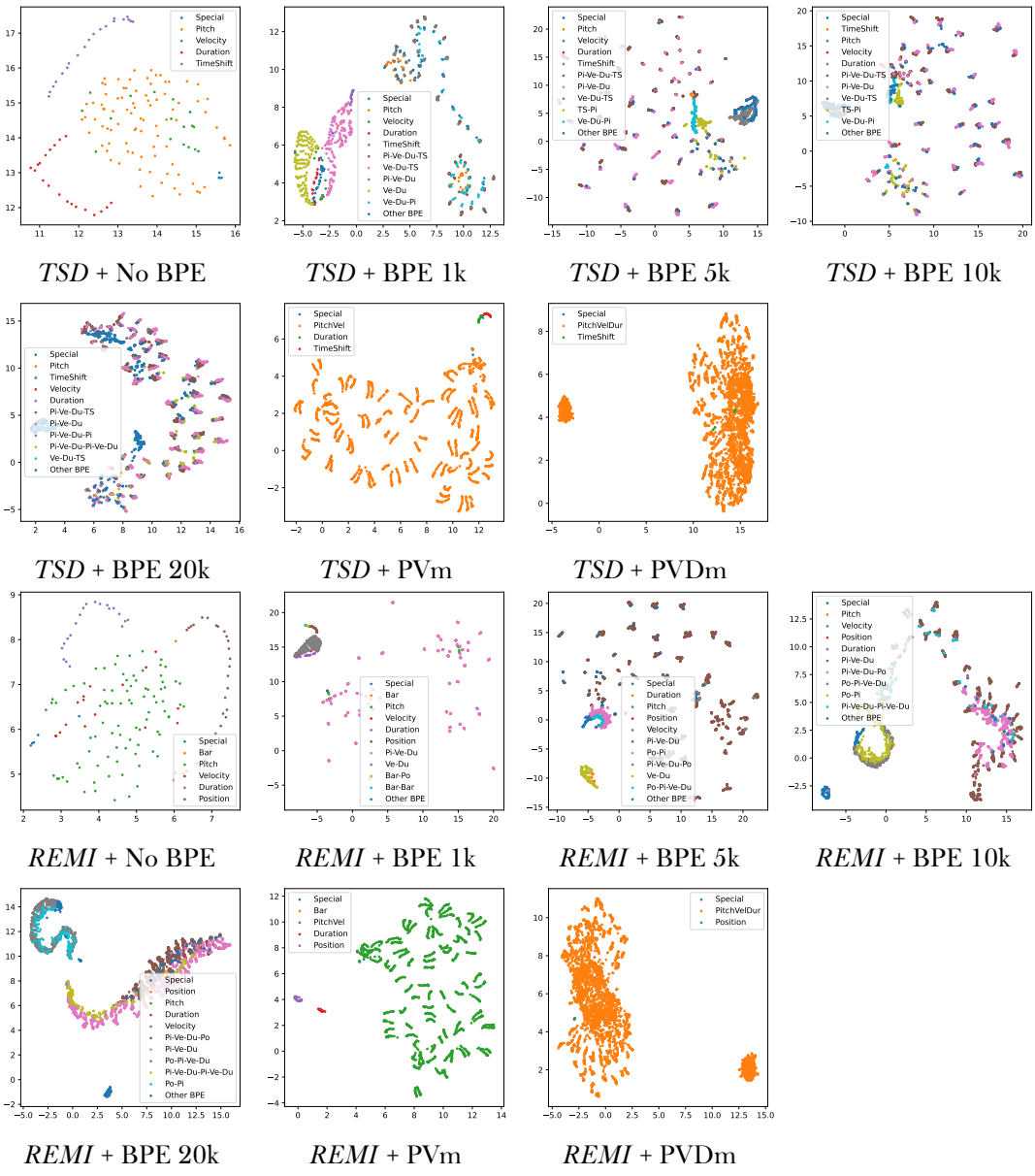
An example of MIDI file open with the Logic Pro DAW is shown in Figure B.2.

B.4 Learned embedding space

UMAP [125] representations shown in Figure B.3, Figure B.4, Figure B.5 and Figure B.6 show the embeddings of the models of the paper, computed with the official UMAP Python package with default parameters. For each figure, only 1k randomly sampled points are represented in order to keep them in vector format without adding too much weight in this file document. We encourage the reader to visualize them on our demo website for a more convenient readability.

The models learn clusters of embeddings of the same type. The embeddings do not occupy the space uniformly, but rather have preferred directions following their type and value. We still note that bi-directional (pretrained) models tends to occupy more space than causal (generative) ones. This especially noticeable for the *PVm* and *PVDm* models. For generative models, the embeddings tends to be oriented towards common dimensions, and slightly spread towards orthogonal one.

Pretrained bi-directional models learn more isotropic embedding representations. The embeddings are spread more uniformly across all directions.



TSD + No BPE | *TSD* + BPE 1k | *TSD* + BPE 5k | *TSD* + BPE 10k

TSD + BPE 20k | *TSD* + PVm | *TSD* + PVDm

REMI + No BPE | *REMI* + BPE 1k | *REMI* + BPE 5k | *REMI* + BPE 10k

REMI + BPE 20k | *REMI* + PVm | *REMI* + PVDm

Figure B.3: UMAP 2d representations of the embeddings of the generators, trained with the Maestro dataset. Abbreviations in legend stand for: Pi: Pitch; Ve: Velocity; Du: Duration; Po: Position; TS: TimeShift.

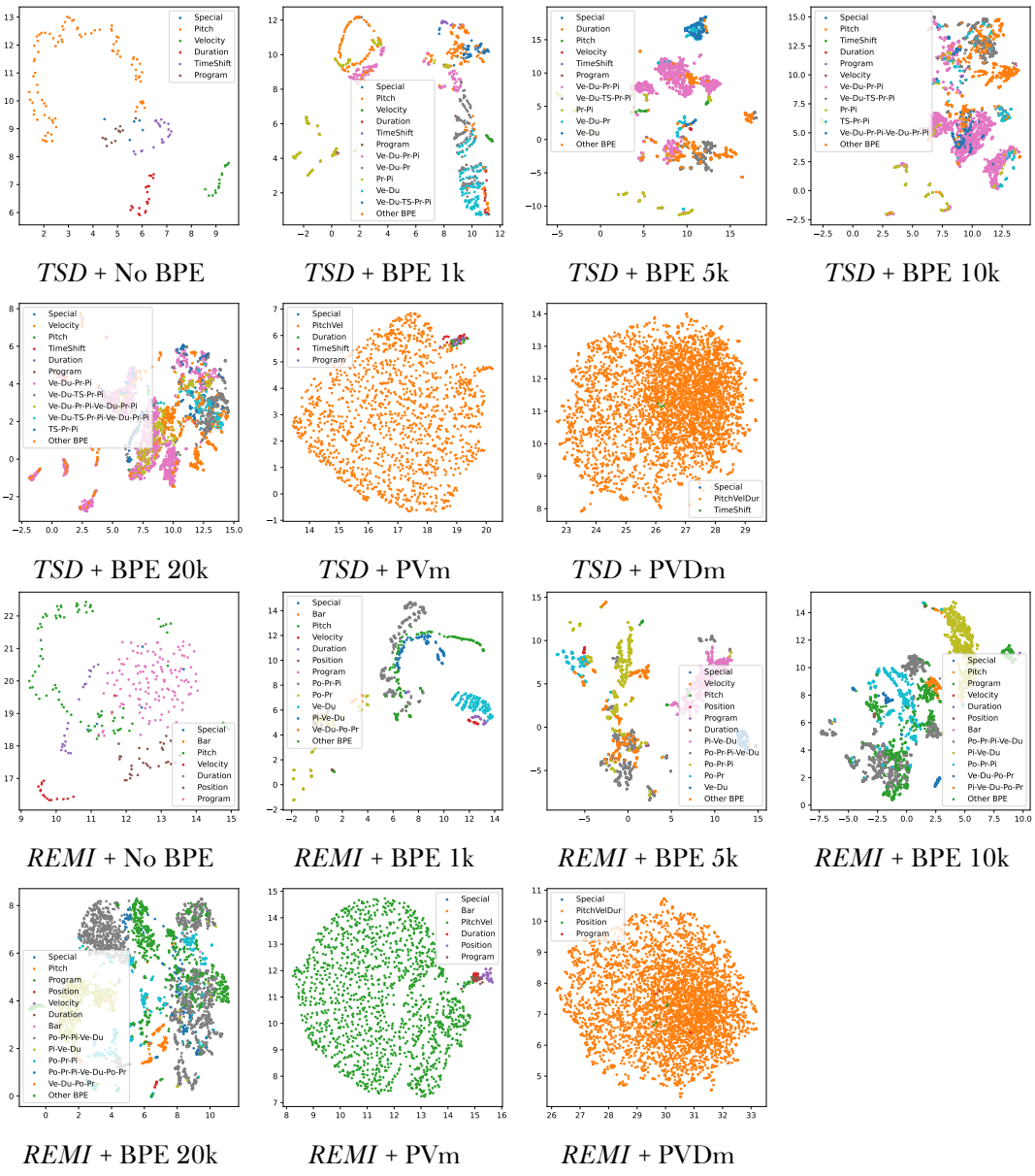
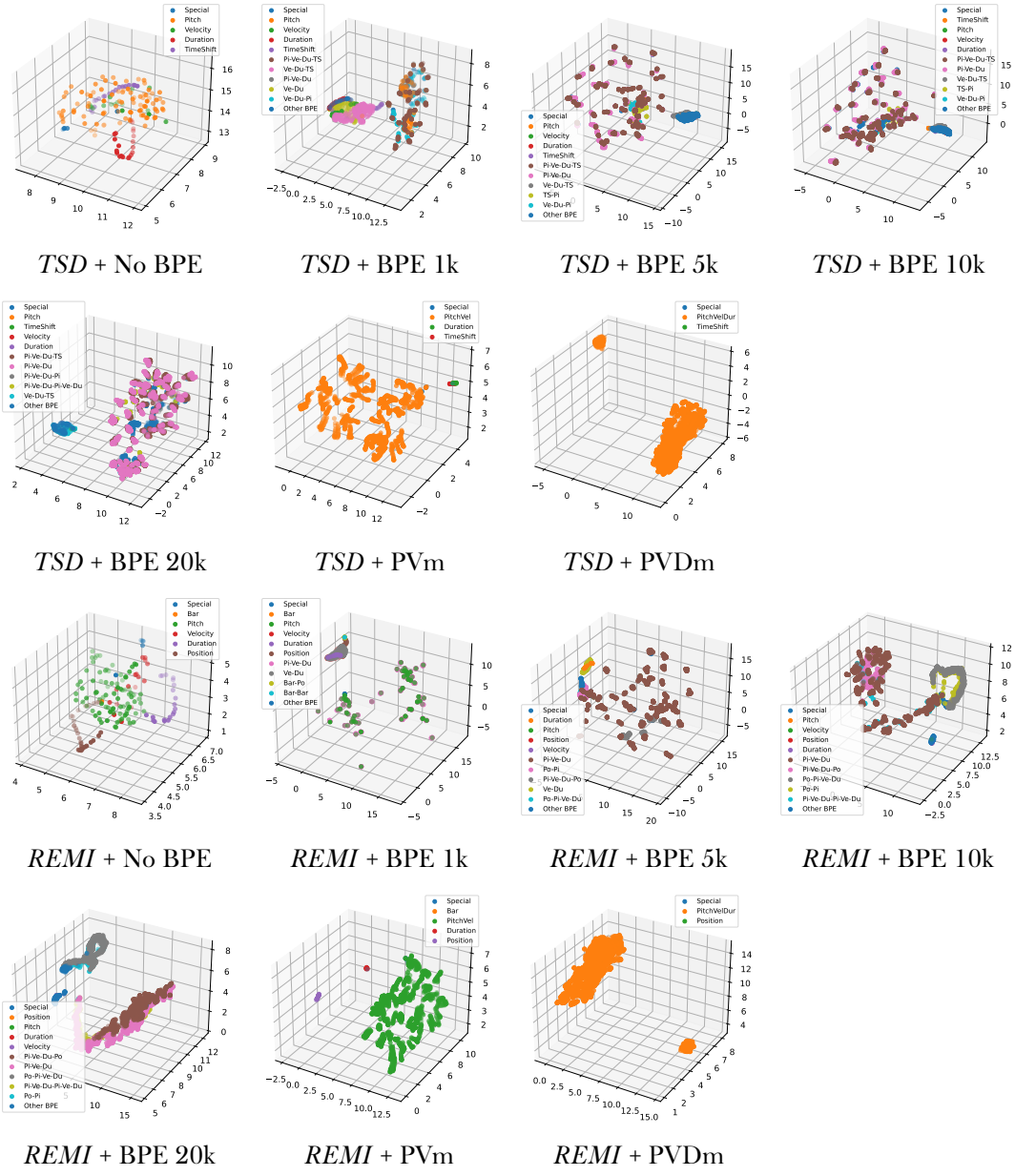


Figure B.4: UMAP 2d representations of the embeddings of the pretrained bidirectional models, trained with the Maestro dataset. Abbreviations in legend stand for: Pit: Pitch; Ve: Velocity; Du: Duration; Po: Position; TS: TimeShift; Pr: Program.



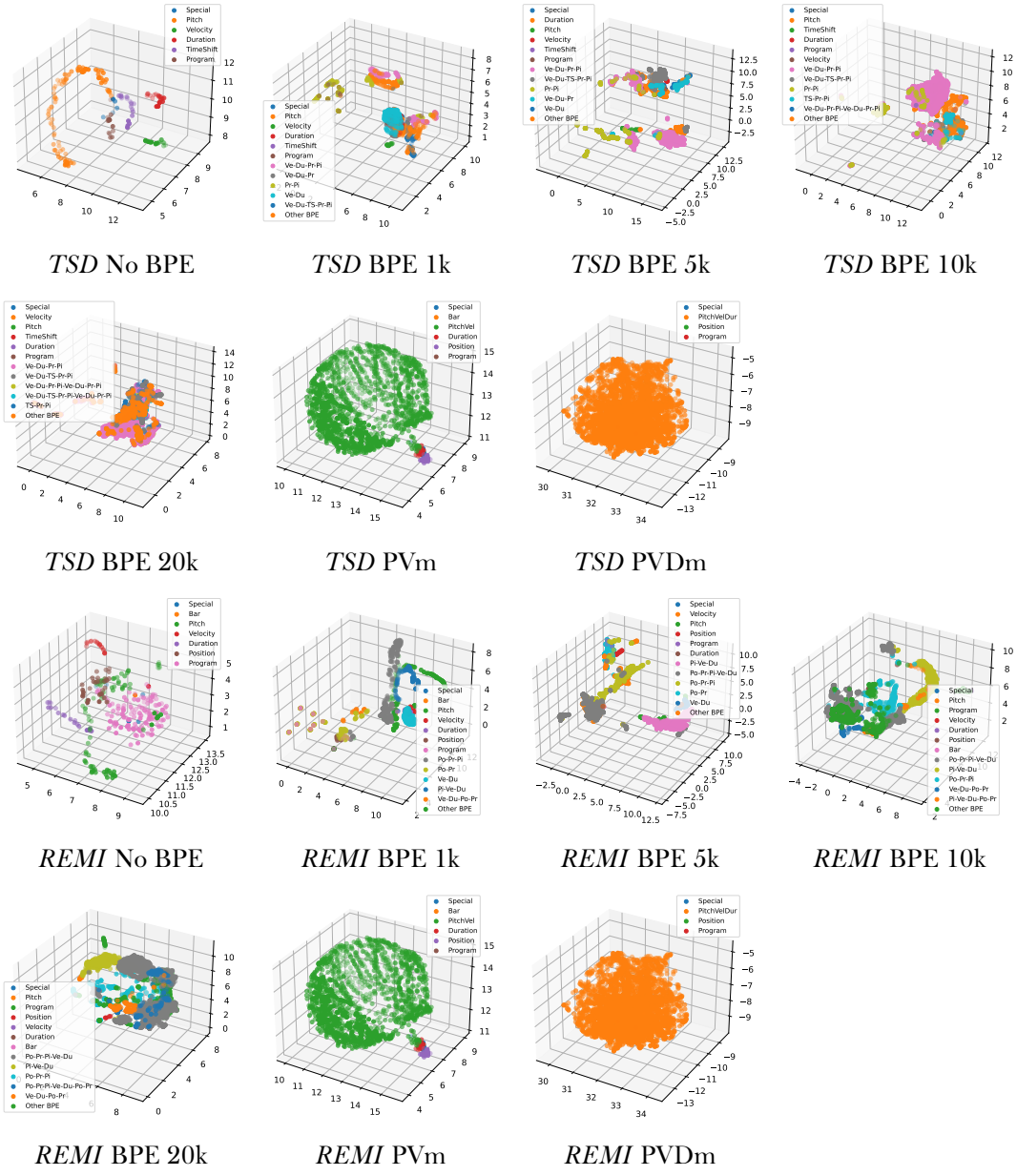
TSD + No BPE | *TSD* + BPE 1k | *TSD* + BPE 5k | *TSD* + BPE 10k

TSD + BPE 20k | *TSD* + PVm | *TSD* + PVDm

REMI + No BPE | *REMI* + BPE 1k | *REMI* + BPE 5k | *REMI* + BPE 10k

REMI + BPE 20k | *REMI* + PVm | *REMI* + PVDm

Figure B.5: UMAP 3d representations of the embeddings of generative models, trained with the Maestro dataset. Abbreviations in legend stand for: Pi: Pitch; Ve: Velocity; Du: Duration; Po: Position; TS: TimeShift.



TSD No BPE | *TSD* BPE 1k | *TSD* BPE 5k | *TSD* BPE 10k

TSD BPE 20k | *TSD* PVm | *TSD* PVDm

REMI No BPE | *REMI* BPE 1k | *REMI* BPE 5k | *REMI* BPE 10k

REMI BPE 20k | *REMI* PVm | *REMI* PVDm

Figure B.6: UMAP 3d representations of the embeddings of pretrained bidirectional models, trained with the MMD dataset. Abbreviations in legend stand for: Pi: Pitch; Ve: Velocity; Du: Duration; Po: Position; TS: TimeShift; Pr: Program.

Bibliography

[1] Andrea Agostinelli et al. *MusicLM: Generating Music From Text*. 2023. arXiv: 2301.11325 [cs.SD].

[2] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. *GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints*. 2023. arXiv: 2305.13245 [cs.CL].

[3] Luca Albergante, Jonathan Bac, and Andrei Zinovyev. "Estimating the effective dimension of large biological datasets using Fisher separability analysis". In: *International Joint Conference on Neural Networks (IJCNN)*. July 2019, pp. 1–8.

[4] Laurent Amsaleg, Oussama Chelly, Teddy Furon, Stéphane Girard, Michael E Houle, Ken-Ichi Kawarabayashi, and Michael Nett. "Extreme-value-theoretic estimation of local intrinsic dimensionality". In: *Data Mining and Knowledge Discovery* 32.6 (Nov. 2018), pp. 1768–1805.

[5] Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya Razenshteyn, and Ludwig Schmidt. "Practical and Optimal LSH for Angular Distance". In: *Advances in Neural Information Processing Systems*. Ed. by C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett. Vol. 28. Curran Associates, Inc., 2015.

[6] Abdul Fatir Ansari, Ming Liang Ang, and Harold Soh. "Refining Deep Generative Models via Discriminator Gradient Flow". In: *International Conference on Learning Representations*. 2021.

[7] Andrea Apicella, Francesco Donnarumma, Francesco Isgrò, and Roberto Prevete. "A survey on modern trainable activation functions". In: *Neural Networks* 138 (2021), pp. 14–32.

[8] Martin Arjovsky, Soumith Chintala, and Léon Bottou. "Wasserstein Generative Adversarial Networks". In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, Aug. 2017, pp. 214–223.

[9] Kushal Arora, Layla El Asri, Hareesh Bahuleyan, and Jackie Cheung. "Why Exposure Bias Matters: An Imitation Learning Perspective of Error Accumulation in Language Generation". In: *Findings of the Association for Computational Linguistics: ACL 2022*. Ed. by Smaranda Muresan, Preslav Nakov, and Aline Villavicencio. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 700–710.

[10] Sanjeev Arora, Yuanzhi Li, Yingyu Liang, Tengyu Ma, and Andrej Risteski. "A Latent Variable Model Approach to PMI-based Word Embeddings". In: *Transactions of the Association for Computational Linguistics* 4 (July 2016), pp. 385–399. eprint: https://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00106/1567396/tacl_a_00106.pdf.

[11] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. "Layer Normalization". In: *Advances in Neural Information Processing Systems*. 2016.

[12] Jonathan Bac, Evgeny M. Mirkes, Alexander N. Gorban, Ivan Tyukin, and Andrei Zinovyev. "Scikit-Dimension: A Python Package for Intrinsic Dimension Estimation". In: *Entropy* 23.10 (2021).

[13] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate". In: *International Conference on Learning Representations*. 2016.

[14] Iz Beltagy, Matthew E. Peters, and Arman Cohan. *Longformer: The Long-Document Transformer*. 2020. arXiv: 2004.05150 [cs.CL].

[15] Daniel Biś, Maksim Podkorytov, and Xiuwen Liu. "Too Much in Common: Shifting of Embeddings in Transformer Language Models and its Implications". In: *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Online: Association for Computational Linguistics, June 2021, pp. 5117–5130.

[16] Kaj Bostrom and Greg Durrett. "Byte Pair Encoding is Suboptimal for Language Model Pretraining". In: *Findings of the Association for Computational Linguistics: EMNLP 2020*. Online: Association for Computational Linguistics, Nov. 2020, pp. 4617–4624.

[17] Jean-Pierre Briot. "From artificial neural networks to deep learning for music generation: history, concepts and trends". In: *Neural Computing and Applications* 33.1 (Jan. 2021), pp. 39–65.

[18] Jean-Pierre Briot, Gaëtan Hadjeres, and François-David Pachet. *Deep Learning Techniques for Music Generation*. Computational Synthesis and Creative Systems. Springer International Publishing, 2020.

[19] Andrew Brock, Jeff Donahue, and Karen Simonyan. "Large Scale GAN Training for High Fidelity Natural Image Synthesis". In: *International Conference on Learning Representations*. 2019.

[20] Xingyu Cai, Jiaji Huang, Yuchen Bian, and Kenneth Church. "Isotropy in the Contextual Embedding Space: Clusters and Manifolds". In: *International Conference on Learning Representations*. 2021.

[21] Tong Che, Ruixiang ZHANG, Jascha Sohl-Dickstein, Hugo Larochelle, Liam Paull, Yuan Cao, and Yoshua Bengio. "Your GAN is Secretly an Energy-based Model and You Should Use Discriminator Driven Latent Sampling". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., 2020, pp. 12275–12287.

[22] Ting-Rui Chiang and Yun-Nung Chen. "Relating Neural Text Degeneration to Exposure Bias". In: *Proceedings of the Fourth BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP*. Ed. by Jasmijn Bastings, Yonatan Belinkov, Emmanuel Dupoux, Mario Giulianelli, Dieuwke Hupkes, Yuval Pinter, and Hassan Sajjad. Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 228–239.

[23] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. *Generating Long Sequences with Sparse Transformers*. 2019. arXiv: 1904.10509 [cs.LG].

[24] Joann Ching and yi-hsuan Yang. "Learning To Generate Piano Music With Sustain Pedals". In: *Extended Abstracts for the Late-Breaking Demo Session of the 22nd International Society for Music Information Retrieval Conference*. 2021.

[25] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1724–1734.

[26] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Ed. by Alessandro Moschitti, Bo Pang, and Walter Daelemans. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1724–1734.

[27] Kristy Choi, Curtis Hawthorne, Ian Simon, Monica Dinculescu, and Jesse Engel. "Encoding Musical Style with Transformer Autoencoders". In: *Proceedings of the 37th International Conference on Machine Learning*. Ed. by Hal Daumé III and Aarti Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, July 2020, pp. 1899–1908.

[28] Krzysztof Marcin Choromanski et al. "Rethinking Attention with Performers". In: *International Conference on Learning Representations*. 2021.

[29] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. "Deep Reinforcement Learning from Human Preferences". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Vol. 30. Curran Associates, Inc., 2017.

[30] Jade Copet, Felix Kreuk, Itai Gat, Tal Remez, David Kant, Gabriel Synnaeve, Yossi Adi, and Alexandre Défossez. "Simple and Controllable Music Generation". In: *Thirty-seventh Conference on Neural Information Processing Systems*. 2023.

[31] Louis Couturier, Louis Bigo, and Florence Levé. "Annotating Symbolic Texture in Piano Music: a Formal Syntax". In: *Sound and Music Computing*. Saint-Etienne, France, June 2022.

[32] Katherine Crowson. *VQGAN+CLIP*. 2022.

[33] Wei Dai, Chia Dai, Shuhui Qu, Juncheng Li, and Samarjit Das. "Very Deep Convolutional Neural Networks for Raw Waveforms". In: *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. New Orleans, LA, USA: IEEE Press, 2017, pp. 421–425.

[34] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. "Transformer-XL: Attentive Language Models beyond a Fixed-Length Context". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, July 2019, pp. 2978–2988.

[35] Tri Dao, Daniel Y Fu, Stefano Ermon, Atri Rudra, and Christopher Re. "FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness". In: *Advances in Neural Information Processing Systems*. Ed. by Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho. 2022.

[36] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. "ImageNet: A large-scale hierarchical image database". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 248–255.

[37] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 4171–4186.

[38] Prafulla Dhariwal, Heewoo Jun, Christine Payne, Jong Wook Kim, Alec Radford, and Ilya Sutskever. *Jukebox: A Generative Model for Music*. 2020. arXiv: 2005.00341 [eess.AS].

[39] Ming Ding, Wendi Zheng, Wenyi Hong, and Jie Tang. *CogView2: Faster and Better Text-to-Image Generation via Hierarchical Transformers*. 2022.

[40] Ming Ding et al. "CogView: Mastering Text-to-Image Generation via Transformers". In: *Advances in Neural Information Processing Systems*. Ed. by M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan. Vol. 34. Curran Associates, Inc., 2021, pp. 19822–19835.

[41] Chris Donahue, Huanru Henry Mao, Yiting Ethan Li, Garrison W. Cottrell, and Julian J. McAuley. "LakhNES: Improving Multi-instrumental Music Generation with Cross-domain Pre-training". In: *Proceedings of the 20th International Society for Music Information Retrieval Conference, ISMIR 2019, Delft, The Netherlands, November 4-8, 2019*. 2019, pp. 685–692.

[42] Hao-Wen Dong, K. Chen, Julian McAuley, and Taylor Berg-Kirkpatrick. "MusPy: A Toolkit for Symbolic Music Generation". In: *International Society for Music Information Retrieval Conference*. 2020.

[43] Hao-Wen Dong, Ke Chen, Shlomo Dubnov, Julian McAuley, and Taylor Berg-Kirkpatrick. "Multitrack Music Transformer". In: *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2023, pp. 1–5.

[44] Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang. "MuseGAN: Multi-track Sequential Generative Adversarial Networks for Symbolic Music Generation and Accompaniment". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 32.1 (Apr. 2018).

[45] Alexey Dosovitskiy et al. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *International Conference on Learning Representations*. 2021.

[46] Vincent Dumoulin and Francesco Visin. *A guide to convolution arithmetic for deep learning*. 2018. arXiv: 1603.07285 [stat.ML].

[47] Jesse Engel, Kumar Krishna Agrawal, Shuo Chen, Ishaan Gulrajani, Chris Donahue, and Adam Roberts. "GANSynth: Adversarial Neural Audio Synthesis". In: *International Conference on Learning Representations*. 2019.

[48] Jeff Ens and Philippe Pasquier. *MMM : Exploring Conditional Multi-Track Music Generation with the Transformer*. 2020. arXiv: 2008.06048 [cs.SD].

[49] Jeffrey Ens and Philippe Pasquier. "Building the metamidi dataset: Linking symbolic and audio musical data". In: *Proceedings of 22st International Conference on Music Information Retrieval, ISMIR*. 2021.

[50] Jeffrey Ens and Philippe Pasquier. "CAEMSI : A Cross-Domain Analytic Evaluation Methodology for Style Imitation". In: *International Conference on Innovative Computing and Cloud Computing*. 2018.

[51] Patrick Esser, Robin Rombach, and Bjorn Ommer. "Taming Transformers for High-Resolution Image Synthesis". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2021, pp. 12873–12883.

[52] Kawin Ethayarajh. "How Contextual are Contextualized Word Representations? Comparing the Geometry of BERT, ELMo, and GPT-2 Embeddings". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 55–65.

[53] Elena Facco, Maria d'Errico, Alex Rodriguez, and Alessandro Laio. "Estimating the intrinsic dimension of datasets by a minimal neighborhood information". In: *Scientific Reports* 7.1 (Sept. 2017), p. 12140.

[54] Angela Fan, Mike Lewis, and Yann Dauphin. "Hierarchical Neural Story Generation". In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics, July 2018, pp. 889–898.

[55] Nathan Fradet, Jean-Pierre Briot, Fabien Chhel, Amal El Fallah Seghrouchni, and Nicolas Gutowski. "MidiTok: A Python package for MIDI file tokenization". In: *Extended Abstracts for the Late-Breaking Demo Session of the 22nd International Society for Music Information Retrieval Conference*. 2021.

[56] Nathan Fradet, Nicolas Gutowski, Fabien Chhel, and Jean-Pierre Briot. "Byte Pair Encoding for Symbolic Music". In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Ed. by Houda Bouamor, Juan Pino, and Kalika Bali. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 2001–2020.

[57] Nathan Fradet, Nicolas Gutowski, Fabien Chhel, and Jean-Pierre Briot. "Impact of time and note duration tokenizations on deep learning symbolic music modeling". In: *Proceedings of the 24th International Society for Music Information Retrieval Conference*. Milano, Italy, Nov. 2023.

[58] K. Fukunaga and D.R. Olsen. "An Algorithm for Finding Intrinsic Dimensionality of Data". In: *IEEE Transactions on Computers* C-20.2 (1971), pp. 176–183.

[59] Philip Gage. "A New Algorithm for Data Compression". In: *C Users J.* 12.2 (Feb. 1994), pp. 23–38.

[60] Jun Gao, Di He, Xu Tan, Tao Qin, Liwei Wang, and Tieyan Liu. "Representation Degeneration Problem in Training Natural Language Generation Models". In: *International Conference on Learning Representations*. 2019.

[61] Tianyu Gao, Xingcheng Yao, and Danqi Chen. "SimCSE: Simple Contrastive Learning of Sentence Embeddings". In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 6894–6910.

[62] Joshua P Gardner, Ian Simon, Ethan Manilow, Curtis Hawthorne, and Jesse Engel. "MT3: Multi-Task Multitrack Music Transcription". In: *International Conference on Learning Representations*. 2022.

[63] Mathieu Giraud, Florence Levé, Florent Mercier, Marc Rigaudière, and Donatien Thorez. "Towards Modeling Texture in Symbolic Data". In: *International Society for Music Information Retrieval Conference - ISMIR*. Taipei, Taiwan, 2014, pp. 59–64.

[64] Aidan N Gomez, Mengye Ren, Raquel Urtasun, and Roger B Grosse. "The Reversible Residual Network: Backpropagation Without Storing Activations". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Vol. 30. Curran Associates, Inc., 2017.

[65] Chengyue Gong, Di He, Xu Tan, Tao Qin, Liwei Wang, and Tie-Yan Liu. "FRAGE: Frequency-Agnostic Word Representation". In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. Vol. 31. Curran Associates, Inc., 2018.

[66] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[67] D. Griffin and Jae Lim. "Signal estimation from modified short-time Fourier transform". In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 32.2 (1984), pp. 236–243.

[68] Gaëtan Hadjeres and Léopold Crestel. *The Piano Inpainting Application*. 2021. arXiv: 2107.05944 [cs.SD].

[69] Gaëtan Hadjeres, François Pachet, and Frank Nielsen. "DeepBach: a Steerable Model for Bach Chorales Generation". In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, Aug. 2017, pp. 1362–1371.

[70] Behzad Haki, Julian Lenz, and Sergi Jorda. "NeuralMidiFx: A Wrapper Template for Deploying Neural Networks as VST3 Plugins". In: *AIMC 2023* (Aug. 2023).

[71] Yongchang Hao, Yuxin Liu, and Lili Mou. "Teacher Forcing Recovers Reward Functions for Text Generation". In: *Advances in Neural Information Processing Systems*. Ed. by S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh. Vol. 35. Curran Associates, Inc., 2022, pp. 12594–12607.

[72] Curtis Hawthorne, Ian Simon, Rigel Swavely, Ethan Manilow, and Jesse H. Engel. "Sequence-to-Sequence Piano Transcription with Transformers". In: *Proceedings of the 22nd International Society for Music Information Retrieval Conference, ISMIR 2021, Online, November 7-12, 2021*. 2021, pp. 246–253.

[73] 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.

[74] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep Residual Learning for Image Recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2016.

[75] Chadi Helwe, Chloé Clavel, and Fabian M. Suchanek. "Reasoning with Transformer-based Models: Deep Learning, but Shallow Reasoning". In: *3rd Conference on Automated Knowledge Base Construction*. 2021.

[76] Romain Hennequin, Anis Khlif, Felix Voituret, and Manuel Moussallam. "Spleeter: a fast and efficient music source separation tool with pre-trained models". In: *Journal of Open Source Software* 5.50 (2020), p. 2154.

[77] Jonathan Ho, Ajay Jain, and Pieter Abbeel. "Denoising Diffusion Probabilistic Models". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., 2020, pp. 6840–6851.

[78] Jonathan Ho, Chitwan Saharia, William Chan, David J. Fleet, Mohammad Norouzi, and Tim Salimans. "Cascaded Diffusion Models for High Fidelity Image Generation". In: *Journal of Machine Learning Research* 23.47 (2022), pp. 1–33.

[79] Jonathan Ho and Tim Salimans. "Classifier-Free Diffusion Guidance". In: *NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications*. 2021.

[80] Sepp Hochreiter. "Recurrent neural net learning and vanishing gradient". In: *International Journal Of Uncertainity, Fuzziness and Knowledge-Based Systems* 6.2 (1998), pp. 107–116.

[81] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780. eprint: `https://direct.mit.edu/neco/article-pdf/9/8/1735/813796/neco.1997.9.8.1735.pdf`.

[82] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. "The Curious Case of Neural Text Degeneration". In: *International Conference on Learning Representations*. 2020.

[83] 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". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 35.1 (May 2021), pp. 178–186.

[84] Cheng-Zhi Anna Huang, Tim Cooijmans, Adam Roberts, Aaron Courville, and Douglas Eck. "Counterpoint by Convolution". In: *Proceedings of 18st International Conference on Music Information Retrieval, ISMIR*. 2017.

[85] Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Ian Simon, Curtis Hawthorne, Noam Shazeer, Andrew M. Dai, Matthew D. Hoffman, Monica Dinculescu, and Douglas Eck. "Music Transformer". In: *International Conference on Learning Representations*. 2019.

[86] Rongjie Huang, Jiawei Huang, Dongchao Yang, Yi Ren, Luping Liu, Mingze Li, Zhenhui Ye, Jinglin Liu, Xiang Yin, and Zhou Zhao. "Make-An-Audio: Text-To-Audio Generation with Prompt-Enhanced Diffusion Models". In: *Proceedings of the 40th International Conference on Machine Learning*. Ed. by Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett. Vol. 202. Proceedings of Machine Learning Research. PMLR, July 2023, pp. 13916–13932.

[87] 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*. MM '20. Seattle, WA, USA: Association for Computing Machinery, 2020, pp. 1180–1188.

[88] Hsiao-Tzu Hung, Joann Ching, Seungheon Doh, Nabin Kim, Juhan Nam, and Yi-Hsuan Yang. "EMOPIA: A Multi-Modal Pop Piano Dataset For Emotion Recognition and Emotion-based Music Generation". In: *Proceedings of the 22nd International Society for Music Information Retrieval Conference, ISMIR 2021, Online, November 7-12, 2021*. Ed. by Jin Ha Lee, Alexander Lerch, Zhiyao Duan, Juhan Nam, Preeti Rao, Peter van Kranenburg, and Ajay Srinivasamurthy. 2021, pp. 318–325.

[89] Andrew Jaegle, Felix Gimeno, Andy Brock, Oriol Vinyals, Andrew Zisserman, and Joao Carreira. "Perceiver: General Perception with Iterative Attention". In: *Proceedings of the 38th International Conference on Machine Learning*. Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, July 2021, pp. 4651–4664.

[90] Andrew Jaegle et al. "Perceiver IO: A General Architecture for Structured Inputs & Outputs". In: *International Conference on Learning Representations*. 2022.

[91] Eric Jang, Shixiang Gu, and Ben Poole. "Categorical Reparameterization with Gumbel-Softmax". In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[92] Ens Jeffrey and Pasquier Philippe. "Quantifying Musical Style: Ranking Symbolic Music based on Similarity to a Style". In: *Proceedings of the 20th International Society for Music Information Retrieval Conference* (Delft, The Netherlands). ISMIR, Nov. 2019, pp. 870–877.

[93] Shulei Ji, Jing Luo, and Xinyu Yang. *A Comprehensive Survey on Deep Music Generation: Multi-level Representations, Algorithms, Evaluations, and Future Directions*. 2020. arXiv: 2011.06801 [cs.SD].

[94] Albert Q. Jiang et al. *Mistral 7B*. 2023. arXiv: 2310.06825 [cs.CL].

[95] Junyan Jiang, Gus G. Xia, Dave B. Carlton, Chris N. Anderson, and Ryan H. Miyakawa. "Transformer VAE: A Hierarchical Model for Structure-Aware and Interpretable Music Representation Learning". In: *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2020, pp. 516–520.

[96] Tero Karras, Samuli Laine, and Timo Aila. "A Style-Based Generator Architecture for Generative Adversarial Networks". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. July 2019.

[97] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. "Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention". In: *Proceedings of the 37th International Conference on Machine Learning*. Ed. by Hal Daumé III and Aarti Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, Aug. 2020, pp. 5156–5165.

[98] Mathieu Kermarec, Louis Bigo, and Mikaela Keller. "Improving tokenization expressiveness with pitch intervals". In: *Extended Abstracts for the Late-Breaking Demo Session of the 23nd International Society for Music Information Retrieval Conference*. 2022.

[99] Omar Khattab and Matei Zaharia. "ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT". In: *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '20. Virtual Event, China: Association for Computing Machinery, 2020, pp. 39–48.

[100] Diederik P Kingma and Max Welling. *Auto-Encoding Variational Bayes*. 2022. arXiv: 1312.6114 [stat.ML].

[101] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. 2015.

[102] Durk P Kingma, Tim Salimans, and Max Welling. "Variational Dropout and the Local Reparameterization Trick". In: *Advances in Neural Information Processing Systems*. Ed. by C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett. Vol. 28. Curran Associates, Inc., 2015.

[103] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. "Reformer: The Efficient Transformer". In: *International Conference on Learning Representations*. 2020.

[104] Qiuqiang Kong, Keunwoo Choi, and Yuxuan Wang. *Large-Scale MIDI-based Composer Classification*. 2020. arXiv: 2010.14805 [cs.SD].

[105] Qiuqiang Kong, Bochen Li, Jitong Chen, and Yuxuan Wang. "GiantMIDI-Piano: A large-scale midi dataset for classical piano music". In: *Transactions of the International Society for Music Information Retrieval*. Vol. 5. 2021, pp. 87–98.

[106] Felix Kreuk, Gabriel Synnaeve, Adam Polyak, Uriel Singer, Alexandre Défossez, Jade Copet, Devi Parikh, Yaniv Taigman, and Yossi Adi. "AudioGen: Textually Guided Audio Generation". In: *The Eleventh International Conference on Learning Representations*. 2023.

[107] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger. Vol. 25. Curran Associates, Inc., 2012.

[108] Taku Kudo. "Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates". In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics, July 2018, pp. 66–75.

[109] Matt J. Kusner and José Miguel Hernández-Lobato. *GANS for Sequences of Discrete Elements with the Gumbel-softmax Distribution*. 2016.

[110] Sylvain Lamprier, Thomas Scialom, Antoine Chaffin, Vincent Claveau, Ewa Kijak, Jacopo Staiano, and Benjamin Piwowarski. "Generative Cooperative Networks for Natural Language Generation". In: *Proceedings of the 39th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato. Vol. 162. Proceedings of Machine Learning Research. PMLR, July 2022, pp. 11891–11905.

[111] Yann LeCun and Yoshua Bengio. "Convolutional Networks for Images, Speech, and Time Series". In: *The Handbook of Brain Theory and Neural Networks*. Cambridge, MA, USA: MIT Press, 1998, pp. 255–258.

[112] Honglak Lee, Peter Pham, Yan Largman, and Andrew Ng. "Unsupervised feature learning for audio classification using convolutional deep belief networks". In: *Advances in Neural Information Processing Systems*. Ed. by Y. Bengio, D. Schuurmans, J. Lafferty, C. Williams, and A. Culotta. Vol. 22. Curran Associates, Inc., 2009.

[113] Minhyeok Lee and Junhee Seok. "Controllable Generative Adversarial Network". In: *IEEE Access* 7 (2019), pp. 28158–28169.

[114] Haoran Li and Wei Lu. "Mixed Cross Entropy Loss for Neural Machine Translation". In: *Proceedings of the 38th International Conference on Machine Learning*. Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, July 2021, pp. 6425–6436.

[115] Feynman T. Liang, Mark Gotham, Matthew Johnson, and Jamie Shotton. "Automatic Stylistic Composition of Bach Chorales with Deep LSTM". In: *Proceedings of the 18th International Society for Music Information Retrieval Conference, ISMIR 2017, Suzhou, China, October 23-27, 2017*. 2017, pp. 449–456.

[116] Yuxin Liang, Rui Cao, Jie Zheng, Jie Ren, and Ling Gao. "Learning to Remove: Towards Isotropic Pre-Trained BERT Embedding". In: *Artificial Neural Networks and Machine Learning – ICANN 2021: 30th International Conference on Artificial Neural Networks, Bratislava, Slovakia, September 14–17, 2021, Proceedings, Part V*. Bratislava, Slovakia: Springer-Verlag, 2021, pp. 448–459.

[117] Drew Linsley, Junkyung Kim, Vijay Veerabadran, Charles Windolf, and Thomas Serre. "Learning long-range spatial dependencies with horizontal gated recurrent units". In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. Vol. 31. Curran Associates, Inc., 2018.

[118] Jiafeng Liu, Yuanliang Dong, Zehua Cheng, Xinran Zhang, Xiaobing Li, Feng Yu, and Maosong Sun. "Symphony Generation with Permutation Invariant Language Model". In: *Proceedings of the 23rd International Society for Music Information Retrieval Conference*. Bengalore, India: ISMIR, Dec. 2022.

[119] Antoine Liutkus, Ondřej Cífka, Shih-Lun Wu, Umut Şimşekli, Yi-Hsuan Yang, and Gaël Richard. "Relative Positional Encoding for Transformers with Linear Complexity". In: *Proceedings of the 38th International Conference on Machine Learning*. Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, July 2021, pp. 7067–7079.

[120] Thang Luong, Hieu Pham, and Christopher D. Manning. "Effective Approaches to Attention-based Neural Machine Translation". In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, Sept. 2015, pp. 1412–1421.

[121] Chris J Maddison, Daniel Tarlow, and Tom Minka. "A* Sampling". In: *Advances in Neural Information Processing Systems*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger. Vol. 27. Curran Associates, Inc., 2014.

[122] Elman Mansimov, Emilio Parisotto, Jimmy Ba, and Ruslan Salakhutdinov. "Generating Images from Captions with Attention". In: *ICLR*. 2016.

[123] Anqi Mao, Mehryar Mohri, and Yutao Zhong. "Cross-Entropy Loss Functions: Theoretical Analysis and Applications". In: *Proceedings of the 40th International Conference on Machine Learning*. Ed. by Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett. Vol. 202. Proceedings of Machine Learning Research. PMLR, July 2023, pp. 23803–23828.

[124] Brian McFee, Colin Raffel, Dawen Liang, Daniel P.W. Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. "librosa: Audio and Music Signal Analysis in Python". In: *Proceedings of the 14th Python in Science Conference*. Ed. by Kathryn Huff and James Bergstra. 2015, pp. 18–24.

[125] Leland McInnes, John Healy, Nathaniel Saul, and Lukas Grossberger. "UMAP: Uniform Manifold Approximation and Projection". In: *The Journal of Open Source Software* 3.29 (2018), p. 861.

[126] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. "Pointer Sentinel Mixture Models". In: *International Conference on Learning Representations*. 2017.

[127] Paulius Micikevicius et al. "Mixed Precision Training". In: *International Conference on Learning Representations*. 2018.

[128] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. "Distributed Representations of Words and Phrases and their Compositionality". In: *Advances in Neural Information Processing Systems*. Ed. by C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger. Vol. 26. Curran Associates, Inc., 2013.

[129] Tomás Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. "Efficient Estimation of Word Representations in Vector Space". In: *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*. 2013.

[130] Gautam Mittal, Jesse H. Engel, Curtis Hawthorne, and Ian Simon. "Symbolic Music Generation with Diffusion Models". In: *Proceedings of the 22nd International Society for Music Information Retrieval Conference, ISMIR 2021, Online, November 7-12, 2021*. 2021, pp. 468–475.

[131] Jiaqi Mu and Pramod Viswanath. "All-but-the-Top: Simple and Effective Post-processing for Word Representations". In: *International Conference on Learning Representations*. 2018.

[132] Ryan Murdock. *BigSleep*. 2022.

[133] Alexander Quinn Nichol, Prafulla Dhariwal, Aditya Ramesh, Pranav Shyam, Pamela Mishkin, Bob Mcgrew, Ilya Sutskever, and Mark Chen. "GLIDE: Towards Photorealistic Image Generation and Editing with Text-Guided Diffusion Models". In: *Proceedings of the 39th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato. Vol. 162. Proceedings of Machine Learning Research. PMLR, July 2022, pp. 16784–16804.

[134] Weili Nie, Nina Narodytska, and Ankit Patel. "RelGAN: Relational Generative Adversarial Networks for Text Generation". In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[135] 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*. 2016. arXiv: 1609.03499 [cs.SD].

[136] Aaron van den Oord, Oriol Vinyals, and koray kavukcuoglu koray. "Neural Discrete Representation Learning". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Vol. 30. Curran Associates, Inc., 2017.

[137] Sageev Oore, Ian Simon, Sander Dieleman, Douglas Eck, and Karen Simonyan. "This Time with Feeling: Learning Expressive Musical Performance". In: *Neural Computing and Applications* 32 (2018), pp. 955–967.

[138] Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. "Image Transformer". In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, July 2018, pp. 4055–4064.

[139] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Vol. 32. Curran Associates, Inc., 2019.

[140] Christine Payne. *MuseNet*. 2019.

[141] Hao Peng, Nikolaos Pappas, Dani Yogatama, Roy Schwartz, Noah Smith, and Lingpeng Kong. "Random Feature Attention". In: *International Conference on Learning Representations*. 2021.

[142] Jeffrey Pennington, Richard Socher, and Christopher Manning. "GloVe: Global Vectors for Word Representation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1532–1543.

[143] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. "Deep Contextualized Word Representations". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, June 2018, pp. 2227–2237.

[144] Ofir Press and Lior Wolf. "Using the Output Embedding to Improve Language Models". In: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*. Valencia, Spain: Association for Computational Linguistics, Apr. 2017, pp. 157–163.

[145] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. *Improving language understanding by generative pre-training*. 2018.

[146] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. *Language Models are Unsupervised Multitask Learners*. 2019.

[147] Alec Radford et al. "Learning Transferable Visual Models From Natural Language Supervision". In: *Proceedings of the 38th International Conference on Machine Learning*. Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, July 2021, pp. 8748–8763.

[148] Colin Raffel. "Learning-Based Methods for Comparing Sequences, with Applications to Audio-to-MIDI Alignment and Matching". PhD thesis. Columbia University, 2016.

[149] Colin Raffel, Brian McFee, Eric J. Humphrey, Justin Salamon, Oriol Nieto, Dawen Liang, and Daniel P. W. Ellis. "MIR_EVAL: A Transparent Implementation of Common MIR Metrics." In: *International Society for Music Information Retrieval Conference*. Ed. by Hsin-Min Wang, Yi-Hsuan Yang, and Jin Ha Lee. 2014, pp. 367–372.

[150] Sara Rajaee and Mohammad Taher Pilehvar. "An Isotropy Analysis in the Multilingual BERT Embedding Space". In: *Findings of the Association for Computational Linguistics: ACL 2022*. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 1309–1316.

[151] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. "ZeRO: Memory Optimizations toward Training Trillion Parameter Models". In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '20. Atlanta, Georgia: IEEE Press, 2020.

[152] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. *Hierarchical Text-Conditional Image Generation with CLIP Latents*. 2022.

[153] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. *Zero-Shot Text-to-Image Generation*. 2021.

[154] Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. "Sequence Level Training with Recurrent Neural Networks". In: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2016.

[155] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. "DeepSpeed: System Optimizations Enable Training Deep Learning Models with Over 100 Billion Parameters". In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '20. Virtual Event, CA, USA: Association for Computing Machinery, 2020, pp. 3505–3506.

[156] Emily Reif, Ann Yuan, Martin Wattenberg, Fernanda B Viegas, Andy Coenen, Adam Pearce, and Been Kim. "Visualizing and Measuring the Geometry of BERT". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Vol. 32. Curran Associates, Inc., 2019.

[157] Nils Reimers and Iryna Gurevych. "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Nov. 2019.

[158] Yi Ren, Jinzheng He, Xu Tan, Tao Qin, Zhou Zhao, and Tie-Yan Liu. "PopMAG: Pop Music Accompaniment Generation". In: *Proceedings of the 28th ACM International Conference on Multimedia*. Seattle, WA, USA: Association for Computing Machinery, 2020, pp. 1198–1206.

[159] Herbert Robbins and Sutton Monro. "A Stochastic Approximation Method". In: *The Annals of Mathematical Statistics* 22.3 (1951), pp. 400–407.

[160] Adam Roberts, Jesse Engel, Colin Raffel, Curtis Hawthorne, and Douglas Eck. "A Hierarchical Latent Vector Model for Learning Long-Term Structure in Music". In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, Oct. 2018, pp. 4364–4373.

[161] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. "High-Resolution Image Synthesis With Latent Diffusion Models". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. May 2022, pp. 10684–10695.

[162] William Rudman, Nate Gillman, Taylor Rayne, and Carsten Eickhoff. "IsoScore: Measuring the Uniformity of Embedding Space Utilization". In: *Findings of the Association for Computational Linguistics: ACL 2022*. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 3325–3339.

[163] David E. Rumelhart and James L. McClelland. "Learning Internal Representations by Error Propagation". In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*. 1987, pp. 318–362.

[164] Dimitri von Rütte, Luca Biggio, Yannic Kilcher, and Thomas Hofmann. "FIGARO: Controllable Music Generation using Learned and Expert Features". In: *The Eleventh International Conference on Learning Representations*. 2023.

[165] Chitwan Saharia et al. "Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding". In: *Advances in Neural Information Processing Systems*. Ed. by S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh. Vol. 35. Curran Associates, Inc., 2022, pp. 36479–36494.

[166] Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. "Linear Transformers Are Secretly Fast Weight Programmers". In: *Proceedings of the 38th International Conference on Machine Learning*. Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, July 2021, pp. 9355–9366.

[167] Rico Sennrich, Barry Haddow, and Alexandra Birch. "Neural Machine Translation of Rare Words with Subword Units". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 1715–1725.

[168] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. "Self-Attention with Relative Position Representations". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, May 2018, pp. 464–468.

[169] Noam Shazeer. *Fast Transformer Decoding: One Write-Head is All You Need*. 2019. arXiv: 1911.02150 [cs.NE].

[170] Alon Shoshan, Nadav Bhonker, Igor Kviatkovsky, and Gérard Medioni. "GAN-Control: Explicitly Controllable GANs". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. Oct. 2021, pp. 14083–14093.

[171] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015.

[172] Yang Song and Stefano Ermon. "Generative Modeling by Estimating Gradients of the Data Distribution". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Vol. 32. Curran Associates, Inc., 2019.

[173] Bob L. Sturm, João Felipe Santos, and Iryna Korshunova. "Folk Music Style Modelling by Recurrent Neural Networks with Long Short-Term Memory Units". In: *Extended abstracts for the Late-Breaking Demo Session of the 16th International Society for Music Information Retrieval Conference*. 2015.

[174] Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. "Long Range Arena : A Benchmark for Efficient Transformers". In: *International Conference on Learning Representations*. 2021.

[175] John Thickstun, David Hall, Chris Donahue, and Percy Liang. *Anticipatory Music Transformer*. 2023. arXiv: 2306.08620 [cs.SD].

[176] Hugo Touvron et al. *Llama 2: Open Foundation and Fine-Tuned Chat Models*. 2023. arXiv: 2307.09288 [cs.CL].

[177] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is All you Need". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Vol. 30. Curran Associates, Inc., 2017.

[178] Lingxiao Wang, Jing Huang, Kevin Huang, Ziniu Hu, Guangtao Wang, and Quanquan Gu. "Improving Neural Language Generation with Spectrum Control". In: *International Conference on Learning Representations*. 2020.

[179] Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. *Linformer: Self-Attention with Linear Complexity*. 2020. arXiv: 2006.04768 [cs.LG].

[180] Tongzhou Wang and Phillip Isola. "Understanding Contrastive Representation Learning through Alignment and Uniformity on the Hypersphere". In: *Proceedings of the 37th International Conference on Machine Learning*. Ed. by Hal Daumé III and Aarti Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, July 2020, pp. 9929–9939.

[181] Ziyu Wang, Ke Chen, Junyan Jiang, Yiyi Zhang, Maoran Xu, Shuqi Dai, Guxian Bin, and Gus Xia. "POP909: A Pop-song Dataset for Music Arrangement Generation". In: *Proceedings of 21st International Conference on Music Information Retrieval, ISMIR*. 2020.

[182] Laurent de Wilde. *Les Fous du Son*. Ed. by Grasset. 2016.

[183] Thomas Wolf et al. "Transformers: State-of-the-Art Natural Language Processing". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45.

[184] Yonghui Wu et al. *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation*. 2016. arXiv: 1609.08144 [cs.CL].

[185] Jin Xu, Xiaojiang Liu, Jianhao Yan, Deng Cai, Huayang Li, and Jian Li. "Learning to Break the Loop: Analyzing and Mitigating Repetitions for Neural Text Generation". In: *Advances in Neural Information Processing Systems*. Ed. by S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh. Vol. 35. Curran Associates, Inc., 2022, pp. 3082–3095.

[186] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention". In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, July 2015, pp. 2048–2057.

[187] Tao Xu, Pengchuan Zhang, Qiuyuan Huang, Han Zhang, Zhe Gan, Xiaolei Huang, and Xiaodong He. "AttnGAN: Fine-Grained Text to Image Generation With Attentional Generative Adversarial Networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. May 2018.

[188] Yifan Xu, Kening Zhang, Haoyu Dong, Yuezhou Sun, Wenlong Zhao, and Zhuowen Tu. *Rethinking Exposure Bias In Language Modeling*. 2020. arXiv: 1910.11235 [cs.CL].

[189] Li-Chia Yang and Alexander Lerch. "On the Evaluation of Generative Models in Music". In: *Neural Comput. Appl.* 32.9 (May 2020), pp. 4773–4784.

[190] Dongchao Yang, Jianwei Yu, Helin Wang, Wen Wang, Chao Weng, Yuexian Zou, and Dong Yu. "Diffsound: Discrete Diffusion Model for Text-to-Sound Generation". In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 31 (2023), pp. 1720–1733.

[191] Yao-Yuan Yang et al. "TorchAudio: Building Blocks for Audio and Speech Processing". In: *arXiv preprint arXiv:2110.15018* (2021).

[192] Jiahui Yu et al. *Scaling Autoregressive Models for Content-Rich Text-to-Image Generation*. 2022.

[193] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. "SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient". In: *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*. AAAI'17. San Francisco, California, USA: AAAI Press, 2017, pp. 2852–2858.

[194] Mingliang Zeng, Xu Tan, Rui Wang, Zeqian Ju, Tao Qin, and Tie-Yan Liu. "MusicBERT: Symbolic Music Understanding with Large-Scale Pre-Training". In: *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*. Online: Association for Computational Linguistics, Aug. 2021, pp. 791–800.

[195] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. *Dive into Deep Learning*. 2023.

[196] Qin Zhang, Shangsi Chen, Dongkuan Xu, Qingqing Cao, Xiaojun Chen, Trevor Cohn, and Meng Fang. "A Survey for Efficient Open Domain Question Answering". In: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Toronto, Canada: Association for Computational Linguistics, July 2023, pp. 14447–14465.

[197] Jingwei Zhao, Gus Xia, and Ye Wang. "Domain Adversarial Training on Conditional Variational Auto-Encoder for Controllable Music Generation". In: *Proceedings of the 23rd International Society for Music Information Retrieval Conference*. 2022.

[198] Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. "Large Language Models are Human-Level Prompt Engineers". In: *The Eleventh International Conference on Learning Representations*. 2023.

Résumé de la thèse

Dans ce résumé, nous introduisons brièvement l'état de l'art de la modélisation de la musique symbolique par apprentissage automatique, et pointons les enjeux actuels que nous abordons dans cette thèse. Nous présenterons les contributions apportées, notamment leurs bénéfices et points d'intérêts, pour enfin introduire des pistes de recherches qui pourraient davantage contribuer au domaine.

Introduction

La musique a historiquement longtemps été représentée sous forme de partition papier. Bien que le format de partition soit toujours largement utilisé de nos jours, il a été remplacé par de nouveaux formats dans certains cas d'usage depuis le siècle dernier. Depuis le siècle dernier, les projets technologiques ont largement influencé et modifié la façon dont nous créons, concevons, utilisons et partageons la musique. L'électronique analogique nous a permis de représenter, stocker et transmettre la musique sous forme d'ondes, sur des supports tels que les vinyles. Dès lors, il nous a été possible de mixer, amplifier, filtrer, altérer le son, voire en produire de nouveaux créés par des oscillateurs et autres composants électronique. Plus récemment, le progrès l'informatique a permis de représenter la musique sous forme numérique, sous forme de séquences de 1 et 0. De nos jours, nous utilisons quotidiennement des appareils numériques, que ce soient des ordinateurs, téléphones ou téléviseurs. Ces appareils sont pour la plupart connectés à internet, par lequel l'information circule de façon numérique. Nos usages courants ont à nouveau considérablement affecté la façon dont nous utilisons la musique, qu'il s'agisse de sa création à son écoute. La musique est aujourd'hui créée à partir de logiciels de création tels que Logic Pro ou Ableton, et est largement diffusée par internet à travers les services de streaming.

Depuis quelques années, nous assistons à un nouveau changement majeur de notre usage des technologies. Les avancées portées au domaine des semi-conducteurs nous permettent de profiter de processeurs de calculs dont les performances ont considérablement augmentées d'année en année. Cette puissance nous a permis de développer des applications logicielles davantage complexifiées, avec une fréquence de calculs par seconde grandissante. Plus spécifiquement, les tâches faisant appel à l'apprentissage automatique en ont grandement bénéficié, jusqu'à faire partie intégrante de notre quotidien à leur tour. L'apprentissage automatique est une technique par laquelle un système est entraîné à réaliser une tâche à partir de données servant d'exemple. On compte parmi ces tâches la reconnaissance d'image ou le traitement automatique du langage naturel. Ces systèmes sont aujourd'hui présents dans de nombreuses applications que nous utilisons tous les jours, tels que les moteurs de recherche, le traitement d'image, et demain la conduite autonome.

L'apprentissage automatique permet également de créer des modèles génératifs, capables de produire du contenu tel que des images, musiques ou vidéos. De grands progrès on notamment été faits pour la génération d'images, donnant naissance à des modèles très performants tels que Dall-E [153, 152] ou Stable Diffusion [161]. La génération de musique pose d'autres challenges et contraintes, mais est toutefois convenablement réalisée sous forme audio et sous certaines conditions par des modèles tels que MusicLM [1], MusicGen [30] and AudioGen [106].

Des méthodes de génération de musique symbolique par apprentissage automatique font également l'objet de vives recherches. L'avantage de ce format par rapport à l'audio est la flexibilité et le contrôle qu'il offre sur le résultat généré. En effet, les notes produites peuvent facilement être modifiées, ainsi que les autres éléments musicaux tels que le tempo. Ce format perd toutefois des informations importantes, faisant le corps de la musique tel que le timbre des instruments, effets et tout élément ne pouvant pas être représentés de façon discrète. La génération de musique symbolique n'a donc pas pour finalité la création de musique sous forme complète, mais vient davantage s'inscrire dans un contexte d'aide à la composition musicale. Compte tenu de la grandissante importance de l'intelligence artificielle dans nos tâches du quotidien, il est raisonnable de penser qu'elle viendra à jouer un rôle important dans la composition de musique.

Apprentissage automatique et musique symbolique

Représentations continues

La musique symbolique est traditionnellement représentée sous forme de partition, comme montrée sur la Figure 3.1. Cette notation, qui existe initialement sur support papier, permet d'inscrire les notes de façon en spécifiant leur hauteur, durée et temps.

Plus récemment, la représentation sous forme de "piano roll", comme montré sur la Figure 3.3, s'est peu à peu insérée dans les outils de production musicale DAW (Digital Audio Workstation). Ce format est pratique pour qu'un utilisateur visualize la musique et l'édite à la souris d'un ordinateur. Il représente la musique sous forme d'une matrice à deux dimensions, l'une pour le temps (en abscisse) et l'autre pour la hauteur des notes (en ordonnée). L'intuitivité de ce format a encouragé les chercheurs à l'utiliser dans des systèmes d'apprentissage automatique, afin d'y accomplir des tâches de MIR (Music Information Retrieval), mais surtout de génération, tels que MuseGAN [44] et Coconet [84]. Comme un pianoroll est une matrice, ces modèles sont basés sur des opérations continues telle que la convolution, et génèrent la musique de façon non-autoregressive.

La musique symbolique peut être néanmoins davantage vue comme discrète que continue, et comporte des informations, telles que le tempo, qui peuvent être difficilement représentées dans une matrice à deux voire trois dimensions. Ces modèles, bien que fonctionnant en pratique avec certains types de données, restent ainsi limités, et ont encouragé les chercheurs à se tourner vers des représentations discrètes.

Représentations discrètes

La musique symbolique peut en effet également être représentée sous forme de séquence d'éléments discrets, à l'instar du texte, qu'il soit de la langue naturelle, code de programmation ou écritures mathématiques. Ces éléments sont communément appelés "*tokens*". Ces séquences de tokens sont ensuite traitées par des modèles discrets, qui vont traiter l'information qu'elles représentent, par la sémantique des tokens et leurs combinaisons, formant par exemple des phrases.

Dans le contexte de la musique symbolique, les notes et leurs attributs, ainsi que le temps doit ainsi être représenté sous forme de tokens. On appelle cette procédure une tokenization. Pour cela, des premiers travaux tels que FolkRNN [173] ou DeepBach [69], ont représenté la hauteur des notes sous forme de tokens, ensuite sérialisés sous forme de séquences traitées par des Réseaux de Neurones Récurrents (RNN). Par la suite, de nouveaux travaux ont proposé des tokenizations plus universelles capables de représenter n'importe quel type de musique. Parmi les plus connues figurent *MIDI-Like* [137] et *REMI* [87]. Pour cette première, les notes sont tokenizées par des tokens de `NoteOn`,

`Velocity` et `NoteOff`, et le temps par des tokens `TimeShift` avec les mêmes structures et valeurs que dans la norme MIDI. *REMI* lui représente explicitement la durée des notes avec des tokens `Duration`, et le temps par des tokens `Bar` et `Position` indiquant le commencement d'une nouvelle mesure et la position absolue du temps de celle actuelle.

Ces représentations discrètes menèrent à la création de modèles génératifs plus performants [85, 87, 48], capable de produire des résultats musicaux plus consistants. Elles permettent également de représenter différentes informations musicales de façon très flexible, telles que le tempo ou les effets. D'autres informations, notamment labels de datasets, peuvent également être inclus. La génération de musique se fait de façon autoregressive, c'est-à-dire en générant le token suivant le dernier de la séquence d'entrée, qui sera réintroduit à plusieurs reprises en entrée du modèle pour générer les tokens l'un après l'autre.

Compte tenu des avantages de la représentation de la musique de façon discrète, les chercheurs ont proposés récemment différentes manières de la tokenizer, avec différents avantages et inconvénients et pour plusieurs cas d'usage.

Enjeux actuels

La modélisation de la musique symbolique présente un certain nombre d'enjeux qui rendent les efforts requis conséquents, et l'efficience des modèles actuels limités. Dans cette thèse, nous adresserons ceux présentés dans cette section.

Implémentation de tokenization de la musique symbolique

Bien que de nombreux travaux de recherche partagent le code source de leurs expériences [83, 87, 194], leurs implémentations ne sont souvent pas triviales à réutiliser. Le code permettant de tokenizer la musique est souvent caché, utilisant différentes dépendances de packages et pré-traitant les fichiers MIDI avec diverses méthodes. Tout cela demande du temps pour adapter et corriger le code. L'obstacle à l'utilisation du DL avec de la musique symbolique est relativement élevé par rapport à d'autres modalités telles que le texte, l'image ou l'audio qui bénéficient de logiciels bien établis. De plus, ce manque de logiciel rend la comparaison entre les tokenizations musicales injuste, car le prétraitement sera différent.

Ce manque limite également le partage et la réutilisation libres de modèles pré-entraînés. En NLP, des plates-formes telles que le hub Hugging Face[1] permettent de télécharger et de partager librement des modèles pré-entraînés. Chaque modèle discret (e.g. Transformer) est nécessairement partagé avec son tokenizer, qui permet de convertir les données en tokens pour son utilisation. Aujourd'hui, très peu de modèles pour la musique symbolique sont partagé (comme discuté dans la Section 4.5). Nous pensons l'absence de standards de tokenizers musicaux et méthode de partage peuvent en partie l'expliquer.

Créer un moyen simple de tokenizer la musique abaissera donc la barrière de l'utilisation de DL avec de la musique symbolique. Cela aiderait les chercheurs et les ingénieurs, en améliorant leur productivité et leur permettant de partager plus facilement leurs travaux.

L'impact des choix de tokenization de la musique symbolique

Contrairement au texte, la musique symbolique peut être sérialisée de différentes manières, et avec une plus grande flexibilité. Il existe plusieurs façons de représenter le temps, la durée des notes ou les instruments sous la forme d'une séquence de tokens. Les chercheurs ont

[1] https://huggingface.co/models

développé diverses méthodes pour tokenizer la musique. Pourtant, ces travaux n'étudient pas en détail les différences entre ces tokenizations et leur impact sur l'apprentissage des modèles. De plus, ils se concentrent principalement sur la génération de musique, laissant de côté d'autres tâches de modélisation tout aussi importantes lors de l'évaluation de la tokenization musicale, telles que la classification ou transcription.

Pourtant, la manière dont les informations musicales sont représentées peut facilement influencer les performances du modèle. Les modèles génératifs sont essentiellement causals, signifiant que les calculs pour une position donnée dans la séquence d'entrée seront conditionnés uniquement par les tokens des positions précédentes. Par conséquent, les modèles génératifs ont une portée et des capacités de modélisation plus limitées. Les modèles entraînés pour d'autres tâches sont pour la plupart bidirectionnels, signifiant que les calculs sont conditionnés à la fois par le contexte passé et celui futur. Avec des tokens `NoteOff` indiquant implicitement les durées des notes, un modèle génératif pourrait donc être désavantagé pour capturer la mélodie et l'harmonie, alors qu'un modèle bidirectionnel pourrait bénéficier des informations de décalage explicites apportées par de tels tokens.

On peut en effet aussi imaginer d'autres différences avec les différentes combinaisons d'informations de temps, de durée de note, d'instrument ou de tempo. Certains modèles fonctionnent différemment et certaines informations peuvent être plus importantes pour différentes tâches. Nous pouvons facilement identifier les choix de conception possibles en matière de tokenization de la musique, ainsi que toutes les combinaisons possibles, mais nous ne savons pas exactement comment les choisir. Pour ces raisons, une analyse de ces choix pour différents types de modèles et de tâches pourrait approfondir ce domaine de recherche et fournir des conseils sur la tokenization symbolique de la musique.

Taille des séquences de tokens

En représentant chaque attribut de note (hauteur, vélocité, durée) et temps sous forme de token distinct, la tokenization de la musique donne des séquences de tokens relativement longues. Étant donné qu'une note est représentée par trois tokens, une musique tokenizée comptera au moins trois fois son nombre de notes en tokens, sans compter ceux représentant le temps et autres informations.

La densité d'informations par token est ainsi faible, et cette longueur est problématique lorsqu'on souhaite utiliser des modèles Transformers, pour lesquels la complexité en temps et mémoire des calculs augmente quadratiquement avec la longueur de la séquence d'entrée. Par conséquent, un Transformer ne peut traiter des séquences avec une certaine taille limite. Compte tenu de la longueur des séquences de tokens de musique, peu d'information, c'est à dire notes, sera alors traitée par le modèle. Avec des capacités de calcul plus importantes, un modèle pourrait gérer des séquences plus longues, mais serait cependant moins efficace, et les exigences de calcul coûteuses augmenteraient également de façon quadratique.

Des tentatives ont été faites pour réduire cette longueur de séquence pour la musique symbolique. La première stratégie consiste à fusionner les vecteurs d'embedding de certains tokens. Par exemple, CPWord [83] fusionne les embeddings des attributs de note, celles des tempos avec celles des positions du temps, et la signature rythmique avec les tokens de nouvelle mesure. Octuple [194] va encore plus loin en fusionnant tous les embeddings d'attributs de notes avec ceux représentant le temps absolu en mesure et position. Une autre technique pour réduire la longueur des séquences consiste à utiliser des tokens combinant plusieurs attributs. Par exemple, LakhNES [41] combine les tokens d'instrument avec les tokens `NoteOn` et `NoteOff`.

Ces méthodes présentent cependant des inconvénients importants. La fusion des embeddings impose des contraintes sur la mise en œuvre logicielle des modèles, leur entraînement et leur génération. La génération simultanée de tokens à partir de plusieurs

distributions de probabilité produites par le modèle, faite indépendamment les unes par rapport aux autres, ajoute de la variance et des résultats instables. La combinaison "manuelle" de tokens produit elle de grands vocabulaires contenant une grande proportion de tokens non-présents dans les données, donc associés à une faible probabilité par les modèles. Enfin, même avec ces méthodes, la longueur des séquences reste toujours relativement longue.

Une technique capable de réduire efficacement la longueur des séquences de tokens, avec des contraintes très strictes, reste à être expérimentée. De plus, les techniques actuelles ne répondent pas à l'autre défi suivant qui concerne l'information portée par les tokens.

Information portée par les tokens

Dans la Subsection 2.1.4, nous introduisons le concept d'embedding et comment les modèles séquentiels apprennent de façon contextuelle ces représentations vectorielles de tokens. La bonne formation de ces vecteurs est un élément essentiel de ces modèles, car ce sont ces embeddings qui permettent aux modèles de capturer la signification des données et d'effectuer des calculs résolvant les tâches pour lesquelles ils sont entraînés. Pour garantir qu'un modèle fasse bon usage de ces fonctionnalités, il doit ainsi être entraînés avec un vocabulaire contenant des tokens représentant des sémantiques riches et diverses. Le nombre de tokens, et donc d'embeddings à apprendre, doit être choisi en fonction du nombre de dimensions formant les embeddings.

Pourtant, les travaux traitant de la modélisation de la musique symbolique utilisent de petits vocabulaires, contenant généralement entre 200 et 500 tokens, avec des embeddings composés de dimensions 512 à 1024 dimensions. Dans une telle configuration, c'est-à-dire plus de dimensions que le nombre d'éléments à représenter dans l'espace, seulement très une petite partie de l'espace sera utilisée. L'espace est ainsi largement sous-utilisé. En comparaison, en NLP, il est courant d'utiliser des vocabulaires contenant de 30 000 à 70 000 tokens. De plus, il est important de noter que les tokens représentant des attributs de notes et valeurs temporelles ne représente pas d'information sémantique autres que leurs valeurs absolues.

Afin d'augmenter les performances des modèles de musique symbolique, il est essentiel de trouver un meilleur équilibre entre la taille du vocabulaire et le nombre de dimensions d'intégration, tout en utilisant des tokens porteurs de plus d'informations.

Étude de la taille de modèles génératifs multitrack

Ces dernières années, de nombreux modèles d'apprentissage automatique (pré)entraînés ont été partagés publiquement, notamment sur le hub Hugging Face.

Le partage ouvert de ces modèles contribue considérablement à l'avancée du domaine de l'IA. Cela favorise notamment la transparence, abaisse les barrières à l'entrée dans le domaine, stimule l'innovation et rend l'IA accessible de façon équitable à toute la communauté. Les modèles ouverts permettent aux chercheurs de les inspecter et d'identifier les risques, les biais et les limites à prendre en compte et d'améliorer ainsi leur sécurité. Ils peuvent les utiliser pour leurs propres recherches, en les peaufinant ou en analysant leur comportement, ce qui profitera à d'autres chercheurs. Et comme l'entraînement de grands modèles peut s'avérer très coûteux, un grand nombre d'organisations ne peuvent pas se permettre de le faire elles-mêmes. Rendre les modèles librement accessibles est davantage équitable pour les petits acteurs, ce qui stimule l'innovation.

Alors qu'il y a des milliers de modèles entraînés pour des tâches de NLP et CV partagés librement sur internet, il n'en existe que très peu pour la génération de musique symbolique. Ceux existants sont soit petits, mauvais, trop spécifiques à un genre ou à un

instrument, inutilisables ou obsolètes. À ce jour, ces modèles ne sont de toute façon pas largement utilisés, car ils s'intègrent mal dans le flux de travail des musiciens. Pourtant, nous commençons à voir des logiciels permettant d'embarquer des modèles DL sous forme de plugins VST dans les DAW [70]. Nous pensons qu'il s'agit d'un grand pas vers une adoption plus large des modèles DL pour la composition musicale et que cela conduira à davantage d'intégrations de ces modèles dans les DAW. Cependant, le manque de modèles performants libres reste un deuxième obstacle à cette adoption. Cela peut même être une des raisons de la lenteur du développement des solutions d'intégration de modèles DL. Dès que des modèles performants seront rendus disponibles, ces solutions pourraient commencer à attirer davantage d'attention et être mises à jour à un rythme plus rapide.

MidiTok

Constatant l'absence d'outil open-source permettant d'aisément tokenizer la musique, nous avons été motivés de créer MidiTok afin de permettre au plus grand nombre d'utiliser les modèles de DL pour cette modalité. MidiTok est un package Python permettant de tokenizer la musique de plusieurs manières, tout en offrant un grand nombre de fonctionnalités et une grande flexibilité. Il implémente les tokenizations les plus utilisées dans le milieu détaillées dans la Section 5.4, et offre la possibilité de contrôler le degré de précision de représentation de valeurs "semi-continues" telles que le temps ou la vélocité. MidiTok est la première contribution en date de cette thèse, et servira pour les suivantes. Comme il s'agit d'une contribution vivante, le package a été régulièrement mis à jour, et continuera de l'être.

Fonctionnement général

Chaque tokenizer de MidiTok hérite de la classe `MIDITokenizer`, qui implémente le prétraitement des fichiers MIDI et les méthodes communes. Cette classe sert de plateforme commune et simplifie considérablement les différentes étapes de tokenization.

Un tokenizer est créé à partir d'un objet `TokenizerConfig`. Cette configuration contient les paramètres définissant quel type d'informations sera tokenizé et avec quelle précision. Un utilisateur peut choisir de tokenizer les tempos, la signature rythmique, les silences... ou non. Il peut également décider de la résolution des valeurs telles que la vélocité des notes, le temps ou la plage de hauteur à tokenizer. À partir de cette configuration, le tokenizer va créer son vocabulaire de tokens. Un tokenizer peut être enregistré sous forme de fichier json et rechargé à l'identique sans avoir à fournir de configuration.

Nous considérons trois catégories de tokens: 1) Les tokens MIDI globaux, qui représentent des attributs et des événements affectant la musique de manière globale, tels que le tempo ou la signature rythmique ; 2) tokens de piste, représentant les valeurs de pistes distinctes telles que les notes, les accords ou les effets ; 3) Les tokens temporels, qui servent à structurer et à placer les catégories de tokens précédentes dans le temps. La catégorisation des tokens en ces trois types est importante, car elle affecte la façon dont le tokenizer représente le temps.

Lors de la tokenization des pistes MIDI, nous distinguons deux modes de fonctionnement : un mode "flux unique" qui convertit toutes les pistes sous une séquence unique de tokens, et un mode "un flux par piste" qui convertit chaque piste indépendamment. Dans le premier mode, les tokens de temps sont créés pour tous les tokens globaux et de piste à la fois, tandis que dans le second, ils sont créés indépendamment pour chaque séquence de tokens de piste et tokens globaux.

Le processus de tokenization d'un MIDI est le suivant :

1. Prétraite le MIDI :

- Si en mode "flux unique", fusionne les pistes du même programme/instrument ;
- Supprime les notes dont la hauteur est en dehors de la plage du tokenizer, sous-échantillonne les vélocités et temps des notes ;
- Sous-échantillonne les valeurs et les temps de tempo ;
- Sous-échantillonne les temps de signature rythmique ;
- Supprime les notes et changements de tempo et signature rythmique dupliqués, ainsi que les pistes vides ;

2. Crée les tokens MIDI globaux (tempo...) ;
3. Crée les tokens de piste (notes, accords) ;
4. Si en "flux unique", concatène tous les tokens globaux et de piste, sinon concatène les événements globaux à chaque séquence d'événements de piste, puis les trie par temps d'apparition ;
5. Déduit les tokens de temps pour toutes les séquences de tokens (une seule si en "flux unique");
6. Retourne les tokens, sous la forme d'une combinaison d'une liste de chaîne de caractères et d'une liste d'entiers (identifiants de token).

La première et la dernière étape sont effectuées de la même manière pour tous les tokenizers, tandis que les autres étapes peuvent être effectuées différemment en fonction de la tokenization. L'étape de prétraitement est essentielle, car elle formate les informations d'un MIDI pour les adapter aux paramètres du tokenizer. Les temps des événements de piste et globaux sont alignées sur la résolution temporelle du tokenizer, ainsi que sur leurs valeurs. Cela nous garantit de récupérer exactement le même MIDI pré-traité lors de la détokenization des tokens.

Autre fonctionnalités

Parmi les fonctionnalités majeures de MidiTok, on note la possibilité d'utiliser une variété de tokens additionnels apportant différentes informations, notamment les accords, le tempo, la signature, les silences, la pédale ou encore modulation. Ces tokens peuvent améliorer les performances des modèles dans certaines conditions [87].

MidiTok offre également la possibilité d'entraîner le tokenizer avec le Byte Pair Encoding afin de créer un vocabulaire plus riche contenant des tokens représentant des combinaisons d'informations. Nous montrons dans le Chapter 7 que cette méthode permet de considérablement améliorer les résultats des modèles tout en augmentant leur efficience. Pour cela, MidiTok s'appuie sur la bibliothèque "tokenizers" développée par Hugging Face [2], qui écrite en langage Rust permet une exécution très rapide. Il s'agit aujourd'hui de la seule implémentation complète et open-source de BPE pour la musique symbolique, et nous espérons pouvoir encourager chercheurs et ingénieurs à l'utiliser.

MidiTok offre également une interface avec le hub Hugging Face[3], permettant aux utilisateurs de partager leur tokenizers MidiTok sur la plateforme. Cette intégration est très importante à nos yeux, comme elle permettra d'encourager les utilisateurs à partager facilement leurs modèles, mais aussi à expérimenter avec ceux disponible sur la plateforme.

Finalement, MidiTok offre des fonctionnalités d'augmentation de donnée essentielle à l'l'entraînement de modèles, ainsi que des interfaces PyTorch permettant de charger efficacement celles-ci pour l'l'entraînement de modèles.

[2] https://github.com/huggingface/tokenizers
[3] https://huggingface.co/

Retour général d'utilisation

Depuis sa première version fin 2021, MidiTok s'est imposé comme une solution simple, efficace et complète pour tokenizer la musique symbolique. Il est utilisé par les chercheurs de la communauté Music Information Retrieval (MIR) pour leurs travaux, par les développeurs indépendants, par les étudiants et désormais par les industriels.

En novembre 2023, MidiTok a rassemblé plus de 470 étoiles sur GitHub, 60 forks et 20 contributeurs externes. La page du référentiel GitHub compte en moyenne 100 visites quotidiennes et le package est téléchargé en moyenne 900 fois par semaine. La Figure 5.3 montre les téléchargements quotidiens de MidiTok sur PyPi, pour la période février 2023 - août 2023. Au moment où nous écrivons, il compte un total de 80 000 téléchargements sur PyPi depuis sa première version. Nous ne pouvons pas estimer de manière fiable le nombre de projets mensuels ou annuels utilisant MidiTok, mais nous pouvons mentionner que chaque année, un certain nombre d'articles de recherche publiés dans le cadre des débats bien reconnus de l'ISMIR utilisent MidiTok comme tokenizer de base. Enfin, MidiTok est utilisé par le plugin Neutone de Qosmo[4], à utiliser dans les DAW. Neutone permet d'utiliser les modèles DL de manière interactive dans n'importe quelle DAW en tant que plugin VST.

MidiTok a gagné du terrain dans la communauté MIR et nous espérons continuer à bénéficier de ses commentaires et contributions pour améliorer encore la bibliothèque.

Analyse de la tokenization de la musique symbolique

Contrairement au texte, la musique symbolique peut être sérialisée de plusieurs manières et avec une plus grande flexibilité. Une musique peut être jouée par différents instruments et composée de plusieurs notes simultanées, chacune ayant plusieurs attributs à représenter. En conséquence, il est nécessaire de sérialiser ces éléments à travers le temps. Pour y parvenir, les chercheurs ont développé diverses méthodes de tokenization de la musique [137, 87, 194, 55].

Bien que ces travaux présentent des comparaisons de performances de modèles entre tokenizations, leurs principales différences ou similitudes ne sont pas toujours clairement indiquées. Pourtant, les modèles Transformers ont du mal à raisonner, c'est-à-dire à faire des déductions logiques basées sur des points d'information dans les données en entrée [75, 196], mais exécutent mieux les tâches lorsqu'on leur fournit des informations et des instructions explicites [198]. Dans le cas de la musique symbolique, il est donc important d'étudier comment la manière dont l'information musicale est représentée impacte les performances du modèle.

Dans le Chapter 6, nous analysons comment les choix de conception de tokenization peuvent impacter les performances de modèles, pour plusieurs tâches différentes. Nous nous concentrons sur trois aspects importants : la représentation du temps, la durée des notes et les instruments.

Décomposition de la tokenization de musique symbolique

En analysant les conceptions possibles de la tokenization musicale, nous pouvons distinguer sept dimensions clé :

- **Temps** : type de token représentant le temps, soit *TimeShift* indiquant les intervalles de temps entre tokens, soit *Bar* et *Position* indiquant les nouvelles mesures et les

[4] https://neutone.space

positions des tokens à l'intérieur de celles-ci. Nous pouvons également considérer l'unité des tokens *TimeShift*, soit en temps musical, soit en secondes[5] ;

- **Durée des notes** : comment les durées des notes sont représentées, avec des tokens *Duration* ou *NoteOff* ;
- **Hauteur des notes** : la plupart des travaux utilisent des tokens représentant des valeurs de hauteur absolues, bien que des travaux récents aient mis en lumière le gain d'expressivité de la représentation sous forme d'intervalles [98] ;
- **Représentation multipiste** : la représentation de plusieurs morceaux de musique dans une séquence, c'est-à-dire comment les notes sont liées à leur piste associée ;
- **Informations supplémentaires** : toute information supplémentaire telle que les accords, le tempo, les silences, la densité des notes. La vélocité des notes peut également entrer dans cette catégorie ;
- **Sous-échantillonnage** : comment les fonctionnalités de type "semi-continu" sont sous-échantillonnées en ensembles discrets, par exemple les 128 valeurs de vélocité réduites à 16 valeurs ;
- **Compression de séquence** : méthodes pour réduire la longueur des séquences, telles que la fusion de tokens et vecteurs embeddings.

Impact de la représentation du temps et de la durée des notes

Le temps et la durée des notes peuvent tous deux être représentés de deux manières différentes : les tokens `TimeShift` ou `Bar / Position` pour le temps, les tokens `Duration` ou `NoteOff` pour les durées des notes. Les tokens `TimeShift` représentent explicitement les intervalles de temps entre tokens, tandis que les tokens `Bar` et `Position` apportent des informations explicites sur les positions absolues (à l'intérieur des mesures) des notes. On pourrait supposer que le premier pourrait aider à modéliser les mélodies, et le second le rythme et la structure. Pour la durée des notes, les tokens `Duration` expriment intuitivement les durées absolues des notes, tandis que les tokens `NoteOff` indiquent explicitement les temps auxquels les notes s'arrêtent. Avec des tokens `NoteOff`, un modèle devrait modéliser les durées de notes à partir des combinaisons de tokens de temps séparant les tokens.

Nous expérimentons avec les quatre combinaisons de représentations de temps et durée de notes, sur trois taches différentes : génération de musique, classification par compositeur, transcription et apprentissage de séquences dans un espace continue à partir d'apprentissage contrastif.

Génération Lors de la génération, on observe des taux d'erreur différents selon la tokenization. Les tokens `Position` n'apportent presque aucune erreur de type, mais une proportion notable d'erreurs de temps. Lors du décodage des tokens en notes, cela signifie que le temps peut revenir en arrière, ce qui peut entraîner des sections de notes qui se chevauchent. Bien que les tokens `Duration` semblent apporter un peu plus d'erreurs de duplication de notes, l'utilisation des tokens `NoteOn` et `NoteOff` entraîne une proportion considérable d'erreurs de prédiction de notes. Les tokens `NoteOff` prédits alors que la note associée n'était pas jouée ($\mathbf{TSE}_{\mathbf{nnon}}$) n'ont pas de conséquences indésirables lors du décodage des tokens, mais cela étend inutilement la séquence, réduisant l'efficacité

[5] Dans cette thèse, nous traitons uniquement de l'unité de temps musical. Le protocole MIDI représente le temps en unité *tick*, dont la valeur est proportionnelle à la division temporelle (en ticks par temps) et au tempo. Par conséquent, travailler avec des secondes nécessiterait une conversion des ticks.

du modèle, et peut induire en erreur les prédictions de tokens suivants. En analysant les caractéristiques des notes générées par les modèles, on observe que l'utilisation de tokens de `Bar` et `Position` mène à davantage de notes jouées en début de mesure. Cela peut s'expliquer par la prédiction de nouvelles mesures alors que celle actuelle n'est pas encore terminée. L'utilisation de tokens `NoteOff` mène également à des notes ayant des durées plus longues. Cela s'explique par le fait que les modèles peuvent facilement oublier les notes ayant été jouées, et donc oublie de prédire un token `NoteOff` pour les terminer. Sur la base de ces résultats, il semble préférable d'utiliser des tokens `Duration` pour la génération.

Classification Nous choisissons d'expérimenter avec l'ensemble de données GiantMIDI [105] pour la classification des compositeurs et l'ensemble de données EMOPIA [88] pour la classification des émotions. Les résultats, comme indiqué dans Table 6.3, indiquent qu'il y a très peu de différence entre les différentes méthodes de tokenization. Cependant, la combinaison de `TimeShift` et `Duration` surpasse systématiquement les autres d'un point.

Transcription La transcription consiste à convertir un contenu musical audio vers sa forme symbolique. Des recherches récentes ont construit des expériences en s'appuyant sur des modèles séquence à séquence [72, 62]. Nous avons décidé de suivre la même stratégie, pour sa facilité de mise en œuvre et ses bonnes performances. Nous avons réalisé l'expérience sur l'ensemble de données Maestro [73], qui est composé de 1 000 paires de fichiers audio et MIDI de performances de piano classique. Pour évaluer les transcriptions générées, nous utilisons la bibliothèque `mir_eval` [149], avec les paramètres par défaut sauf une tolérance d'attaque de 62,5 ms. Nous rapportons les résultats dans Table 6.4. La combinaison avec les meilleures performances globales (début + décalage + vitesse) est *TS + NOff*. En regardant uniquement les débuts, l'utilisation des tokens *Bar* et *Position* semble apporter de meilleures performances. Cependant, les combinaisons qui l'utilisent sont sous-performées par celles utilisant les tokens *TimeShift*, ce qui signifie que *TimeShift* semble aider le modèle à prédire des décalages de notes plus précis. Les scores pour ces derniers sont également meilleurs avec les tokens *NoteOff*, montrant que les tokens de décalage explicites aident le modèle à prédire les temps de fin de notes.

Apprentissage de séquence La dernière tâche que nous avons souhaité explorer est la représentation de séquences. Cela consiste à entraîner un modèle à représenter des séquences de tokens (donc de notes de musique) dans un espace d'embedding continu universel. Cette tâche a déjà été abordée dans le traitement du langage naturel par SentenceBERT [157] ou SimCSE [61]. Nous avons adopté l'approche de cette dernière, qui utilise l'apprentissage contrastif pour entraîner le modèle à apprendre des représentations de séquence, pour lesquelles des entrées similaires ont des similitudes cosinus plus élevées. En conséquence, le modèle apprendra efficacement à créer des vecteurs de séquences similaires pour des entrées similaires, tout en séparant celles présentant des dissimilarités. Nous avons utilisé l'ensemble de données GiantMIDI [105]. L'évaluation a été effectuée en mesurant la distance entre le vecteur d'une séquence de référence, et celui d'une séquence augmentée par rapport à cette référence. Nous avons expérimenté avec plusieurs types d'augmentation : une hauteur de note, un octave, une valeur de vélocité et une combinaison de hauteur et vélocité. Les résultats, présentés dans Figure 6.7, indiquent que les tokenizations utilisant les tokens `Position` donnent des similarités légèrement plus élevées. Par conséquent, il semble que les informations explicites sur les positions d'apparition et de décalage des notes facilitent l'obtention par les modèles d'une représentation musicale universelle. Contrairement à la classification, l'objectif d'apprentissage contrastif modélise

les similitudes et les dissemblances entre les exemples d'un même lot. Dans ce contexte, les positions d'apparition et de décalage des notes semblent utiles aux modèles pour distinguer la musique.

Impact de la représentation multipiste pour la génération

À ce jour, il existe peu de modèles traitant la musique symbolique multipiste. Pourtant, il existe différentes manières de tokenizer plusieurs pistes d'instruments MIDI. Leur but est d'associer une note et ses attributs à leur programme MIDI, ou en d'autres termes à leur instrument. Dans la Section 6.6, nous expérimentons avec quatre tokenizations différentes, visant à mesurer si un modèle est capable de produire de manière cohérente la suite d'une invite musicale. Plus précisément, nous mesurons comment un modèle est capable de continuer à jouer des instruments présents dans l'invite.

Nous calculons cette métrique en mesurant la précision, le rappel et le score F1 entre les instruments en entrée et les instruments générés. Nous rapportons ces scores dans Table 6.6. Cela montre que la fusion des jetons de programme et de pitch (*ProgramPitch*) donne le meilleur score et la meilleure précision F1. Globalement, ces méthodes semblent mieux adaptées pour poursuivre les traces de la saisie. Les tokens *Program* et *ProgramChange* fonctionnent presque de la même manière. Il semble que placer un jeton `Program` avant chaque note n'aide pas le modèle pour la génération. Dans un tel cas, utiliser des jetons `ProgramChange` pourrait être une meilleure option, car cela donnera des séquences de jetons plus courtes. En revanche, *Merged* a un meilleur rappel, mais a cependant une précision et un score F1 inférieurs. En conséquence, le modèle prédira facilement les pistes non liées, qui n'étaient pas présentes dans l'invite.

Nous avons également mesuré la divergence entre la densité des notes des pistes en entrée et celles générées. Nous trouvons des divergences similaires pour toutes les stratégies, qui sont toutefois plus basses pour les modèles utilisant des tokens *ProgramChange* et *Program*.

Byte Pair Encoding pour la musique symbolique

Une des principales contributions de cette thèse est l'étude de l'application du "Byte Pair Encoding" (BPE) à la musique symbolique pour l'apprentissage automatique, menant à de meilleurs résultats et performances.

Problèmes de tokenizations simples

Jusqu'ici, la majorité des travaux tokenizent la musique symbolique en la représentant à partir de tokens représentant uniquement les attributs des notes et du temps. Bien que cela fonctionne, ces méthodes posent deux problèmes majeurs :

1. Les séquences de tokens sont relativement longues, comme chaque note sera représentée par au moins deux tokens. Cela est un problème majeur lorsqu'elles sont utilisées avec des modèles Transformers, pour lesquels la complexité des calculs croit quadratiquement avec la taille de ces séquences ;

2. L'information portée par les tokens est très pauvre. Ceux-ci ne représentent aucune sémantique ou information de plus haut niveau, mais uniquement leur propre valeur absolue. Alors que pour le langage naturel les modèles Transformers fonctionnent principalement en apprenant contextuellement des représentations vectorielles continues des tokens du vocabulaire, ce n'est pour la musique pas le cas. Le modèle se

prive donc de l'apprentissage de représentations plus riches, pourtant essentielles pour les calculs de tâches pour lesquelles ils sont entraînés.

Le premier problème a notamment été adressé, soit en fusionnant les tokens dans le vocabulaire [41], soit en fusionnant les vecteurs d'embeddings des tokens à l'entrée du modèles [83, 194]. Ces méthodes présentent toutefois des contraintes et limites pratiques importantes. Nous proposons ici l'utilisation de BPE afin d'adresser ces deux problèmes.

Byte Pair Encoding

Le BPE [59] est une technique de compression de données. Il remplace les octets successifs les plus récurrents d'un corpus par des octets nouvellement créés. Par exemple, dans la séquence de caractères `aabaabaacaa`, la sous-séquence `aa` apparaît trois fois et est la plus récurrente. Appliquer le BPE sur cette séquence remplacerait `aa` par un nouveau symbole, par exemple `d`, ce qui donnerait une séquence compressée `dbdbdcd`. Cette dernière peut être à nouveau réduite en remplaçant la sous-séquence `db`, donnant `eedcd`. En pratique, le BPE s'apprend sur un corpus jusqu'à ce que le vocabulaire atteigne une taille cible. L'apprentissage du BPE est décrit par le pseudo-code de Algorithm 3.

Le BPE est aujourd'hui largement utilisé en NLP, comme il permet de construire un vocabulaire contenant des mots et parties de mots, comme ce sont les successions d'octets les plus récurrentes dans le texte. Les vocabulaires peuvent ainsi être utilisés pour faire apprendre à des modèles des représentations de mots et parties de mots.

Nouveaux tokens

En l'appliquant à la musique, nous montrons que le BPE permet de créer des nouveaux tokens qui représentent des notes complètes (succession d'attributs), voire des successions de notes. À mesure que le vocabulaire grossit, les nouveaux tokens représentent davantage de combinaisons de tokens de base, comme montré sur la Figure 7.2.

Comme reporté dans la Table 7.1, avec un vocabulaire de 20k tokens appris par BPE, la taille des séquences de token peut être réduit jusqu'à en moyenne 65%. Les temps de tokenization et detokenization sont en conséquence légèrement plus grands, cela dû à l'étape d'encodage-décodage supplémentaire, mais reste toutefois très contenue.

Bénéfices

Nous avons entraînés pour la génération plusieurs modèles avec plusieurs tailles de vocabulaires appris par BPE, allant jusqu'à 20k tokens, ainsi que des modèles implémentant les techniques de réduction de séquence par fusion de tokens et embeddings de travaux précédents. Nous avons ensuite généré des suites de musiques avec chacun de ces modèles, et demandé à des participants humains de choisir leurs préférées selon plusieurs critères : fidélité, diversité, justesse, et globalement. Il en retourne que la plupart des suites générées à partir des modèles entraînés avec BPE étaient préférées par les participants. Nous avons également mesuré les taux d'erreurs de prédiction de ces modèles, et constaté que les modèles avec BPE ne prédisent pas davantage d'erreurs. Ce résultat confirme le fait que les modèles apprennent efficacement ces nouveaux tokens, qui pourtant apportent davantage de complexité. Cette conclusion est aussi appuyée par le fait que presque la totalité de ces tokens sont générés, donc utilisés, par les modèles lors de la génération. Enfin, le dernier gros avantage du BPE est le gain de vitesse de génération, allant jusqu'à 300%, mesurée en tokens, temps et notes par seconde.

Sur une tâche de classification, le BPE permet également d'obtenir de meilleures précisions, mais est toutefois devancé par d'autres méthodes de fusion d'embedding.

Pour finir, lorsqu'on analyse l'espace d'embedding de tous ces modèles, ceux entraînés avec un vocabulaire construit par BPE sont bien mieux utilisés, à la fois en espace total occupé et en isotropie, c'est-à-dire distribution de la variance des positions des embeddings.

Montée en taille de modèles Transformers pour la génération de musique symbolique

La génération de musique multipiste reste encore aujourd'hui une tâche difficile. La musique a de multiples facettes et se compose de centaines de genres et d'instruments joués et arrangés de multiples façons. La musique en général est complexe. Les modèles génératifs de musique multipiste sont jusqu'à présent soit relativement spécialisés, soit souvent limités à des genres ou à des instruments spécifiques [44, 41, 48], et la plupart des modèles plus sofistiqués [164, 43] ne sont pas partagé de façon ouverte ou facilement réutilisables. En conséquence, il est laborieux pour les chercheurs et ingénieur de les utiliser pour leurs propres travaux. De plus, très peu des travaux existants partagent de données sur les fonctions de coût. Ainsi, ces mêmes cherheurs disposent de peu de points de repères afin de dimension leurs modèles pour ces tâches.

Dans cette thèse, nous visons à entrainer une série de modèles de grande taille pour la génération de musique symbolique multipiste et étudier leur mise à l'échelle. Nous nous basons sur trois versions de tailles différentes - 125 millions, 500 millions et 1.5 milliard - et analysons leurs capacités d'apprentissage et de génération.

Procédure d'entraînement

Entraîner de larges modèles est une tâche non-triviale. De nombreux paramètres entrent en jeu pour permettre un entraînement efficace qui puisse être effectué dans une durée raisonnable, souvent simultanément sur plusieurs GPUs. Les plus importantes sont la taille du modèle, le nombre de données d'entraînement, et la configuration matérielle sur laquelle les calculs seront effectués. La gestion de la mémoire est notamment un point d'attention principal. Pour cela, il est aujourd'hui d'usage de procéder à des optimisations visant à réduire la redondance de certaines données à travers les appareils sur lesquels sont entraînés les modèles, notamment les gradients et statuts d'optimizer.

Nous avons bénéficié de l'accès au supercalculateur Jean-Zay, sur lequel sont présents des GPU nvidia A100 et V100. Nous avons utilisé des GPUs pour pré-entraîner les trois versions de nos modèles, ainsi que pour les finetuner sur des genres musicaux spécifiques. Pour ce faire, nous avons fait usage de la bibliothèque DeepSpeed [155], en utilisant le principe ZeRO [151] de stage 2.

Evaluation

Les courbes d'entrainement (Figure 8.1) montrent que les modèles sur-apprennent rapidement, en témoigne les résultats des fonctions de coûts qui augmentent passé un certain point sur les données de validation. Les résultats sur données d'entrainement atteignent des valeurs proches de zero toutefois, montrant que les modèles ont appris à prédire les tokens suivant des données dans une grande parties des cas. Lorsque évalués à leur point de fonction de coût de le plus bas, ces modèles montrent toutefois des prediction très répétitives. Pour finir, nous discutons de l'effet du biais d'exposition sur la modalité de la musique symbolique. Pour ce faire, nous exposons le nombre de combinaisons de successions de tokens que forment les tokens pour deux jeux de données musicaux. Nous les comparons notament à des jeux de données de texte et montrons que les successions de

tokens musicaux sont plus nombreux et variés, et ce pourrait être une des causes principales de cet effet.

Utilisation

Nous avons développé une interface, mise à disposition sur le hub Hugging Face[6], permettant à tout utilisateur d'utiliser le modèle. L'interface se présente sous la forme d'une fenêtre affichant un pianoroll, sur lequel l'utilisateur peut ajouter, modifier et supprimer des notes de plusieurs instruments. Il peut commander la génération de la suite de la musique présente sur le pianoroll, qui apparaîtra dessus une fois terminée.

Conclusion et perspectives

À travers cette thèse, nous avons abaissé les efforts d'entrée au domaine de la modélisation de la musique symbolique, en développant de façon open-source une bibliothèque logicielle permettant de tokenizer la musique de façon simple, flexible et complète.

Considérant la multitude de manières permettant de représenter la musique sous forme de séquences de tokens, nous nous sommes intéressés à celles-ci et avons définis les éléments et choix de design de tokenization autour desquelles elles s'articulent. Nous avons par la suite expérimenté avec les manières de représenter le temps, la durée des notes et les instruments, pour souligner leurs impacts pour différentes tâches.

Jusqu'ici, la modélisation de la musique symbolique souffrait de deux problèmes majeurs : la grande taille des séquences de tokens et la faible information portée par ces tokens. Nous montrons que le BPE, méthode de compression, permet de répondre à ces deux problèmes en créant des vocabulaire de tokens représentant des combinaisons de tokens de base, portant davantage d'information. Les modèles entraînés avec ces vocabulaires sont d'une part bien plus efficients (meilleure vitesse de génération) mais performent également mieux, donnant de meilleurs résultats.

Nous avons finalement entraînés des modèles de génération de musique multipiste de grande taille pour en analyser leurs entraînements et capacités de génération. Nous montrons que ces modèles sur-apprennent rapidement pendant l'entrainement, et sont notament sujet au biais d'exposition, c'est à dire la différence de performances de modèles lorsque évalués sur les données d'entrainement contre en condition réelles de génération.

À la suite de ces travaux, nous identifions plusieurs pistes de recherche qui pourraient permettre de contribuer davantage au domaine. Premièrement, nous n'avons expérimenté qu'avec quatre des choix de design de tokenization, et il reste à étudier les autres. Leurs impacts pourraient permettre de guider plus facilement dans le choix de tokenization de la musique. Deuxièmement, compte tenu de l'importance de l'information portée par les tokens, il nous semble intéressant d'utiliser des tokens décrivant de l'information musicale de plus haut niveau, telle que la texture ou la structure. Pour finir, nous avons expérimenté avec le BPE, mais d'une part nous n'avons que très peu exploré la signification des tokens nouvellement créés qui nous donnent pourtant des informations sur la nature des données, mais d'autre part d'autres algorithmes d'agrégation de tokens existent et pourraient donner lieu à des résultats sur la performance et l'efficience des modèles.

[6] https://huggingface.co/spaces/Aubay/SMILE