

# A Quantum Natural Language Processing Approach to Musical Intelligence

Eduardo Reck Miranda<sup>1,2</sup>, Richie Yeung<sup>2</sup>, Anna Pearson<sup>2</sup>, Konstantinos Meichanetzidis<sup>2</sup>, and Bob Coecke<sup>2</sup>

<sup>1</sup>*ICCMR, University of Plymouth, Plymouth, UK*

<sup>2</sup>*Cambridge Quantum, Oxford, UK*

<sup>1</sup>*eduardo.miranda@plymouth.ac.uk*

<sup>2</sup>*{richie.yeung, anna.pearson, k.mei, bob.coecke}@cambridgequantum.com*

December 10, 2021

## Abstract

There has been tremendous progress in Artificial Intelligence (AI) for music, in particular for musical composition and access to large databases for commercialisation through the Internet. We are interested in further advancing this field, focusing on composition. In contrast to current ‘black-box’ AI methods, we are championing an *interpretable compositional* outlook on generative music systems. In particular, we are importing methods from the Distributional Compositional Categorical (DisCoCat) modelling framework for Natural Language Processing (NLP), motivated by musical grammars. Quantum computing is a nascent technology, which is very likely to impact the music industry in time to come. Thus, we are pioneering a Quantum Natural Language Processing (QNLP) approach to develop a new generation of intelligent musical systems. This work follows from previous experimental implementations of DisCoCat linguistic models on quantum hardware. In this chapter, we present *Quanthoven*, the first proof-of-concept ever built, which (a) demonstrates that it is possible to program a quantum computer to learn to classify music that conveys different meanings and (b) illustrates how such a capability might be leveraged to develop a system to compose meaningful pieces of music. After a discussion about our current understanding of music as a communication medium and its relationship to natural language, the chapter focuses on the techniques developed to (a) encode musical compositions as quantum circuits, and (b) design a quantum classifier. The chapter ends with demonstrations of compositions created with the system.

## 1 Introduction

When people say that John Coltrane’s *Alabama* is awesome or that Flow Composer’s<sup>1</sup> *Daddy’s car* is good, what do they mean by ‘awesome music’ or ‘good music’? This is debatable. People have different tastes and opinions. And this is true whether the music is made by a human or a machine.

The caveat here is not so much to do with the terms ‘awesome music’ or ‘good music’, or whether it is made by humans or machines. The caveat is with the word ‘mean’.

---

<sup>1</sup>Flow Composer is an AI lead sheet generator developed at Sony CSL Paris [Pachet et al., 2021]. A lead sheet is a form of musical notation that specifies the essential elements (melody, lyrics, and harmony) of a song.

In the last 20 years or so, there has been tremendous progress in Artificial Intelligence (AI) for music. But computers still cannot satisfactorily handle meaning in music in controlled ways that generalises between contexts. There is AI technology today to set up computers to compose a decent pastiche of, say, a Beethoven minuet; e.g., there are connectionist (a.k.a., ‘neural networks’) systems for composition that have been trained on certain genres or styles. (For a comprehensive account of the state of the art of AI for music, please refer to [Miranda, 2021a].) However, it is very hard to program a computer to compose a piece of music from a request to, say, ‘generate a piece for Alice’s tea party’. How would it know how tea party music should sound like, or who Alice is? And how would it relate such concepts with algorithms to compose?

A challenging task is to design generative systems with enough knowledge of musical structure, and ability to manipulate said structure, so that given requests for mood, purpose, style, and so on, are appropriately met. Systems that currently attempt to perform such task, specially for music recommendation systems, work in terms of finding and exploiting correlations in large amounts of human-annotated data; e.g., [Adiyansjah et al., 2019]. These for example would fill a preference matrix, encoding information such as ‘likes’ for songs, which are categorised already in certain genres by listeners.

Following an alternative route, which comes under the umbrella of *Compositional Intelligence* [Coecke, 2021], we are taking the first steps in addressing the aforementioned challenge from a Natural Language Processing (NLP) perspective, which adopts a structure-informed approach.

Providing domain-specific structure to intelligent systems in a controlled and scalable way is a nontrivial challenge: we would need to ensure that the system does not significantly lose flexibility to adapt on different datasets. Nevertheless, having structure present in intelligent systems has potential benefits, such as increased interpretability. Another potential benefit is the need for fewer training parameters; in general, a learning system would need a set of free parameters so that it learns structure from scratch. But in our case, a structural approach is motivated by an analogy between grammar in language, and structure in musical compositions. Specifically to the present work, we are pioneering a Quantum Natural Language Processing (QNLP) approach to develop a new generation of intelligent musical systems. This in turn is motivated by our specific choice of mathematical grammar and a modelling framework for NLP, referred to as Distributional Compositional (DisCo). DisCo models are amenable to implementation on quantum computers due to a shared mathematical structure between grammar and quantum theory.

Our aim in this work is to train parameterised quantum circuits to learn to classify music conveying different meanings. And then, use the classifier to compose music that conveys such meanings via rejection sampling. Here, the notion of ‘meaning’ is defined as *perceived properties of musical compositions, which might holistically characterise different types of music*. This is the first step towards the ambition of being able to ask an autonomous intelligent system to generate bespoke pieces of music with particular meanings, required purposes, and so on.

Our classification system is trained on human-annotated examples. It scans given examples for musical elements, which we refer to as *musical snippets*. A snippet can be as small as a single musical note. But it can also be a melody, a bar of music, or a group of bars. A main assumption made by the system - known also as the *distributional hypothesis* - is that snippets that occur in similar contexts convey similar meanings. This is a hypothesis that regards meanings of words in text: words that appear in similar contexts convey similar meanings. Then, a combination of the meanings of all snippets determines the meaning of entire compositions, or sections thereof<sup>2</sup>. These combinations follow *grammatical* rules. Our aim is to input these rules into the machine along with meaning-encodings of the snippets, to allow it to identify different types of music. In the near future we hope it

---

<sup>2</sup>For large pieces of music, it is assumed that different sections may convey different meanings, in the same way that sentences or paragraphs convey localised meanings within the overall narrative of a large text.

will be able to use this knowledge to compose pieces, or sections thereof, conveying specific meanings. Towards the end the chapter we provide a glimpse of how this might be achieved.

Technically, the system assigns types to snippets, in analogy with a parser assigning grammatical part-of-speech tags to words, such as noun, adjective, verb, and so on. Types follow specific compositional rules, and in the specific grammar model that we employ - that is, *pregroup grammar* - types obey algebraic relations.

The rules induce a dependency structure onto a musical piece that is composed by the snippets. Then, *meaning* enters the picture by sending types to vector spaces of specific dimensions. Snippet-meaning is encoded inside these vector spaces in terms of multilinear maps, i.e., *tensors*. The compositional rules represented by the dependency structure induce a pattern of tensor-contractions between the tensors.

The choice to embed meaning in vector spaces is motivated by a similar mathematical structure between the algebra of the types and the composition of tensors. Importantly, the tensors form a network, which reflects the dependencies as *inherited* by the grammatical structure of a piece of music. However, as we are encoding meaning in tensors, we realise that the computational cost of manipulating and composing tensors is prohibitive for large numbers of snippets, even for mainframe supercomputers, let alone your average desktop. These would require high dimensionality of vector spaces along with a highly connected dependency network.

Enter the quantum computer! Quantum processes constitute a powerful choice for a playground in which to encode meaning. After all, quantum theory can be formulated entirely in terms of complex-valued tensor networks. Specifically, quantum theory can be fully considered around one fundamental principle: the way quantum systems compose to form larger ones, and the mathematical operation that jointly describes multiple quantum systems is the tensor product.

Emerging quantum computing technology promises formidable processing power for some tasks. As it turns out, and we will see this below, the mathematical tools that are being developed in QNLP research are quantum-native, by yet another analogy between mathematical structures. That is, they enable direct correspondences between language and quantum mechanics, and consequently music. In this case, (a) the meaning of words can be encoded as quantum states and (b) grammars would then correspond to quantum circuits [Coecke and Kissinger, 2017].

Despite the rapid growth in size and quality of quantum hardware in the last decade, quantum processors are still limited in terms of operating speed and noise-tolerance. However, they are adequate to implement proof-of-concept experiments of our envisaged system.

In this chapter we present *Quanthoven*, the first proof-of-concept ever built, which (a) demonstrates that it is possible to program a quantum computer to learn to classify music that conveys different meanings and (b) illustrates how such a capability might be leveraged to develop Artificial Intelligence systems able to compose meaningful pieces of music.

The remainder of this chapter is structured as follows:

- Section 2 explores our current understanding of music as a communication medium and its relationship to natural language, which motivates our research.
- Section 3 contextualises our work in relation to approaches to computational modelling of music and algorithmic composition.
- Section 4 introduces the very basics of quantum computing, just enough to follow the technical aspects of the quantum classifier (i.e., machine learning algorithm) discussed in this chapter.
- Section 5 introduces Distributional Compositional Categorical (DisCoCat) modelling, which uses Category Theory to unify natural language and quantum mechanics. It also shows how DisCoCat can model music.

- Section 6 describes the DisCoCat music model that we developed for this project, including the process by which musical pieces are generated using a bespoke grammar and converted into a quantum circuit. It delineates the machine learning algorithm and training methodology.
- Section 7 illustrates how the results from the classification can be leveraged to generate two classes of musical compositions.

The chapter ends with final remarks. A number of appendices provide complementary information.

## 2 Music and Meaning

It is widely accepted that instrumental music<sup>3</sup> is a non-verbal language. But what does music communicate? What is meaning in music?

Some advocate that music communicates nothing meaningful because it cannot express ideas or tell stories in the same way that verbal languages can. Music has no ‘words’ to express things like ‘Bob’, ‘mussels and fries’, ‘beer’, ‘Bob loves mussels and fries with a beer’, and so on. But this is a rather defeatist view.

Conversely, others advocate that music can communicate messages of some sort. For example, the imitation of cuckoo birds in Ludwig van Beethoven’s Pastoral Symphony is often thought to signify ‘springtime has arrived’ [Monelle, 1992]. But this is speculation. We do not know if Beethoven really wanted to communicate this.

There have been studies, however, suggesting that different types of chords, rhythms, or melodic shapes may convey specific emotions. For instance, the notion that a minor chord conveys sadness and a major one happiness has been demonstrated experimentally [Bakker and Martin, 2014]. And brain imaging studies have identified correlations between perceived musical signs and specific patterns of brain activity associated with feelings. These studies support the notion that music conveys, if anything, affective states [Koelsch, 2014] [Daly et al., 2015]. But still, there is no treatise to date on how to communicate a specific feeling in a composition. There are no universal rules for doing this.

Yet, it would be perfectly possible to have musical languages matching the expressive power of verbal languages. Creators of Esperanto-like verbal languages, such as Dothraki, made for the TV series *The Game of Thrones*, have developed fully-fledged musical languages [Peterson, 2015] [Moore, 2019]. It is just that *Homo sapiens* have not naturally evolved one as such. This is because we evolved musicality for purposes other than verbal communication. Thus, meaning in music is not quite the same as meaning in verbal languages. But what is it? To tackle this question, we ought to examine how the brain makes sense of language and music.

### 2.1 Brain Resources Overlap

Music and verbal languages do have a lot in common. There is scientific evidence that the brain resources that humans deploy to process music and verbal languages overlap to a great degree [Jäncke, 2012]. Stefan Koelsch proposed an ambitious neurocognitive model of music perception supported by extensive research into the neural substrates shared by music and language [Koelsch, 2011]. Of course, there are differences too. For instance, whereas, the brain engages specialised areas for processing language (e.g., Broca’s and Wernicke’s areas), music processing tends to be distributed over a number of areas, which are not necessarily specialised for music.

Let us consider, for instance, the notion of narrative, in text and music. We seem to deploy cognitive strategies to make sense of music, which are similar to those employed to read a text. This is because

---

<sup>3</sup>That is, music without singing. No lyrics. Only musical instruments.

verbal languages and music share similar intrinsic structures [Patel and Morgan, 2016]. Or at least our brain thinks that this is the case. This is questionable. But it is a useful assumption, which is somewhat supported by experiments testing if musical cognitive functions can influence linguistic ones, and vice-versa.

Notable experiments highlighted the benefits of musical proficiency for the acquisition of linguistic skills, in particular learning a second language [Milovanov and Tervaniemi, 2011] [Patel, 2011]. And Dawson et al. provided experimental evidence that native speakers of distinct languages process music differently [Dawson et al., 2017]. This seems to be determined by the structural and phonological properties of their respective languages. For example, speakers of languages that has words whose meaning is determined by the duration of their vocalization tend to deal with rhythm more accurately than speakers of languages whose durations do not interfere with meaning.

## 2.2 Meaning is Context

In a similar fashion to narrative in text, musical narrative delineates relationships and changes of state of affairs from event to event [Jordan and Kafalenos, 1994]. Auditory events in a piece of music acquire significance within a structure that our mind's ear imposes on them. Different types of music define systems of relations among these elements, which induce us to assign them structural impressions; or create categories. The more we listen to the music of certain styles, the more familiar the categories that define such styles become. And the higher the chances that those assigned impressions might be elicited when we listen to such kinds of pieces of music again and again.

Thus, we parse and interpret auditory streams according to our own made up mental schemes. We might as well refer to such schemes as *musical grammars*. And the labels we give to impressions as *meanings*.

A piece of music may as well make one feel sad or happy, or remember Bob, or think of mussels and fries. And one might even hear Beethoven shouting from his grave that springtime has arrived. Thus, for instance, music with fast repetitive patterns might be categorized as 'rhythmic', 'energetic', or 'exciting'. In contrast, music composed of irregular successions of notes forming melodies and exquisite harmonies might be categorised as 'melodic', 'relaxing' or 'impressionistic'. But this is not as simple as it appears to be. What happens when a piece is prominently rhythmic but is also melodic; e.g., ballroom dance music? We could think of a new category. But what if yet another piece is only slightly more melodic than rhythmic? The boundaries of such categories are not trivial to establish.

Anyway, one thing is certain: the more culturally related a group of listeners are, the higher the chances that they might develop shared repertoires of mental schemes and impressions.

Incidentally, we learn verbal languages in a similar way to what we just described. We make up the meaning of words and learn how to assemble them in phrases from the context in which they are used. We do this as we grow up and continue doing so throughout our lifetime [Eliot, 1999]. Perhaps the main difference is that meaning in verbal languages needs to be more precise than meaning in music. That is, meaning in music tends to be more fluid than in language. As stated above, however, one can construct musical languages if one attempts to.

## 3 Computational Modelling and Algorithmic Composition

Scholars from both camps (i.e., linguistics and musicology) have developed comparable models to study linguistic and musical structures, and respective mental schemes [Chomsky, 2006] [Lerdhal and Jackendoff, 1996] [Baroni, 1999] [Frank et al., 2012].

Not surprisingly, computational models of linguistic processes have been successfully used to model musical processes and vice-versa. David Cope harnessed transition networks grammars [Cope,

1991], which have been used for the analysis and synthesis of natural language [Woods, 1970], to develop systems to analyse and synthesise music. A system to generate music using stochastic context-free grammars was proposed by [Perchy and Sarria, 2016]. Indeed, Noam Chomsky’s context-free grammars [Hopcroft and Ullman, 1979] proved to be relevant for music in many ways. In particular transformative grammars [Chomsky, 1975], which suit well the widely adopted method of composing music referred to as variations on a theme. And [Miranda, 2008] created a system whereby a group of interactive autonomous software agents evolved a common repertoire of intonations (or prosodies) to verbalise words. The agents evolved this by singing to each other. Moreover, evolutionary models informed by neo-Darwinism [Gouyon et al., 2002] have been developed to study the origins of language and music almost interchangeably [Miranda et al., 2010] [Boer, 1999].

Generally, there have been two approaches to designing computer systems to generate music, which we refer to as the Artificial Intelligence (or AI) and the algorithmic approaches, respectively [Miranda, 2014] [Miranda, 2001].

The AI approach is concerned with embedding the system with musical knowledge to guide the generative process. For instance, computers have been programmed with rules of common practice for counterpoint and voicing in order to generate polyphonic music. And machine-learning technology has enabled computers to learn musical rules automatically from given scores, which are subsequently used to generate music [Cope, 2000]. The linguistic-informed modelling mentioned above falls in this category.

Conversely, the algorithmic approach is concerned with translating data generated from seemingly non-musical phenomena onto music. Examples of this approach abound. Computers have been programmed to generate music with fractals [Hsü and Hsü, 1990] and chaotic systems [Harley, 1995]. And also with data from physical phenomena, such as particle collision [Cherston et al., 2016], and DNA sequences [Miranda, 2020].

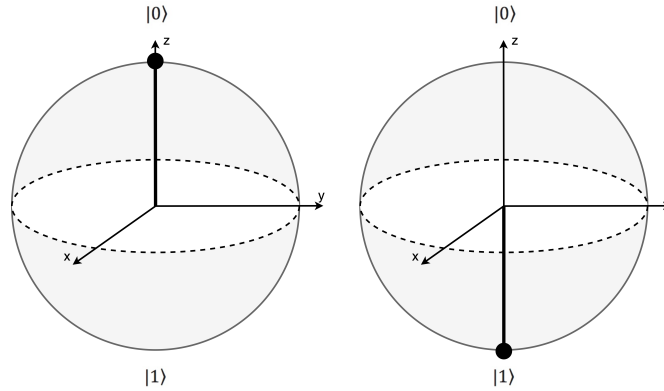
Aesthetically, the algorithmic approach tends to generate highly novel and rather unusual music; some may not even consider them as music. The AI approach tends to generate imitations of existing music; that is, imitations in the style of the music that was used to teach the AI system. Neither approach can, however, satisfactorily respond to sophisticated requests such as ‘generate something for Alice’s tea party’ or ‘compose a tune to make Bob feel energetic’. The QNLP outlook that we are developing is aimed at improving this scenario. In a way, it falls into ‘AI approach’ category, but we are aiming beyond parodying existing styles.

## 4 Brief Introduction to Quantum Computing

A detailed introduction to quantum computing is beyond the scope of this chapter. This can be found in [Nielsen and Chuang, 2011], [Coecke and Kissinger, 2017] and [Sutor, 2019]. Nevertheless, in this section we briefly introduce a few basic concepts deemed necessary to follow the discussions in the following sections.

Our starting point is a quantum bit, known as a *qubit*, which is the basic unit of information carrier in a quantum computer. Physically, it is associated with a property of a physical system; e.g., the spin of an electron up or down along some axis. A qubit has a state  $|\psi\rangle$ , which lives in a 2-dimensional complex vector space, referred to as Hilbert space. The orthonormal basis vectors  $|0\rangle$  and  $|1\rangle$ , related to measurement outcomes 0 and 1, respectively, allow us to write the most general state of a qubit as a linear combination of the basis vectors known as *superposition*:  $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$  with  $\alpha, \beta \in \mathbb{C}$  and  $|\alpha|^2 + |\beta|^2 = 1$ .

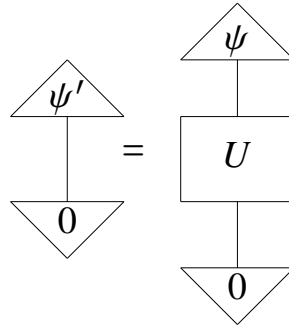
An important aspect of quantum computers is that they are fundamentally probabilistic. This leads to the situation where even if one knows that a qubit is in a state  $|\psi\rangle$ , one can only obtain measurement outcomes  $i = \{0, 1\}$  with probability given by the Born rule  $P(i) = |\langle i|\psi\rangle|^2$ , which gives the square of



**Figure 1:** The Pauli  $\mathbf{X}$  gate rotates the state vector (pointing upwards on the figure on the left side) by 180 degrees around the  $x$ -axis (pointing downwards on the figure on the right).

the norm of the so-called amplitude  $\langle i|\psi\rangle \in \mathbb{C}$ . So,  $\langle i|$  is a quantum effect, also known as a ‘bra’ in Dirac notation, and is the dual vector of the state, or ‘ket’,  $|i\rangle$ . For the general single-qubit superposition in the paragraph above,  $P(0) = |\alpha|^2$  and  $P(1) = |\beta|^2$ .

In order to picture a single qubit, imagine a transparent sphere with opposite poles. From its centre, a vector whose length is equal to the radius of the sphere can point to anywhere on the surface. This sphere is called the *Bloch sphere* and the vector is referred to as a *state vector* (Fig. 1). Before measurement, the evolution of a single qubit (that is, a qubit that is not interacting with any other qubits) is described by the transformation of its state vector with a unitary linear map  $U$ , so that  $|\psi'\rangle = U|\psi\rangle$  (Fig. 2). However if a qubit interacts with other qubits, then things become a little more complicated; more about this below.



**Figure 2:** Diagram showing the evolution of an isolated qubit in initial state  $|\psi\rangle$  with unitary map  $U$  composed with effect  $\langle 0|$ . (Taken from [Lorenz et al., 2021] and used with permission.)

In simple terms, quantum computers are programmed by applying sequences of unitary linear maps to qubits. Programming languages for quantum computing provide a number of such linear maps, referred to as *gates*, which act on qubits. For instance, the ‘not gate’, rotates the state vector by 180 degrees around the  $x$ -axis of the Bloch sphere (Fig. 1); that is, if the qubit vector is pointing to  $|0\rangle$ , then this gate flips it to  $|1\rangle$ , or vice-versa. This gate is often referred to as the ‘Pauli  $\mathbf{X}$  gate’. A more generic rotation  $\mathbf{R}_x(\theta)$  gate is typically available for quantum programming, where the angle for the rotation around the  $x$ -axis is specified. Obviously,  $\mathbf{R}_x(\pi)$  applied to  $|0\rangle$  or  $|1\rangle$  is equivalent to applying  $\mathbf{X}$  to  $|0\rangle$  or  $|1\rangle$ . Similarly, there are  $\mathbf{R}_z(\varphi)$  and  $\mathbf{R}_y(\theta)$  gates for rotations on the  $z$ -axis and  $y$ -axis of the Bloch sphere, respectively. An even more generic gate is typically available, which is a unitary rotation gate, with 3 Euler angles:  $\mathbf{U}(\theta, \varphi, \lambda)$ . Essentially, all single-qubit quantum gates perform rotations,

which change the amplitude distribution of the system. And in fact, any qubit rotation can be specified in terms of  $\mathbf{U}(\theta, \varphi, \lambda)$ ; for instance  $\mathbf{R}_x(\theta) = \mathbf{U}(\theta, -\frac{\pi}{2}, \frac{\pi}{2})$ .

Quantum computation gets really interesting with gates that operate on multiple qubits, such as the controlled  $\mathbf{X}$  gate, or  $\mathbf{CX}$  gate; commonly referred to as the  $\mathbf{CNOT}$  gate. The  $\mathbf{CNOT}$  gate puts two qubits in *entanglement*.

Entanglement establishes a curious correlation between qubits. When considering its action on the computational states, the  $\mathbf{CNOT}$  gate applies an  $\mathbf{X}$  gate on a qubit only if the state of another qubit is  $|1\rangle$ . Thus, the  $\mathbf{CNOT}$  gate establishes a dependency of the state of one qubit with the value of another. In practice, any quantum gate can be made conditional and entanglement can take place between more than two qubits.

An important gate for quantum computing is the Hadamard gate (referred to as the ' $\mathbf{H}$  gate'). It puts the qubit into a balanced superposition state consisting of an equal-weighted combination of two opposing states:  $|\alpha|^2 = 0.5$  and  $|\beta|^2 = 0.5$ .

A combination of  $\mathbf{H}$  and  $\mathbf{CNOT}$  gates enables the implementation of the so-called Bell states; a form of maximally entangled qubits, which is explored later on in this chapter to represent grammatical structures.

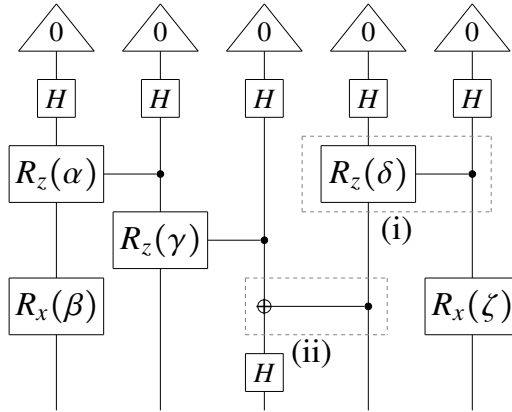
A quantum program is often depicted as a circuit diagram of quantum gates, showing sequences of gate operations on the qubits (Figure 3). So, if one has  $n$  qubits, then their joint state space is given by the tensor product of their individual state spaces and has dimension  $2^n$ . The evolution of many qubits interacting with each other is given by a (global) exponentially large unitary map acting on the entire joint exponentially large state space.

There are two ways of thinking of a quantum circuit. On an abstract level, it can be viewed as the application of linear map to a vector, which computes the entire state of the system at a later time. As stated above, these complex-valued matrices and vectors are exponentially large in terms of the amount of qubits required to encode them. Therefore, simulations of quantum systems are believed to be a hard task for classical computers. And on a physical, or operational, level, a quantum circuit is a set of instructions for a quantum computer to execute.

In the circuit model of quantum computation, qubits typically start at easily preparable states,  $|0\rangle$ , and then a sequence of gates are applied. Next, the qubits are read and the results are stored in standard digital memory, which are accessible for further handling. In practice, a quantum computer works alongside a classical computer, which in effect acts as the interface between the user and the quantum machine. The classical computer enables the user to handle the measurements for practical applications. As a quantum computer is probabilistic, a circuit must be run several times to give statistics to estimate outcome probabilities. The design of a circuit must be such that it encodes the problem in a way that the outcome probabilities obtained give the answer to what one wishes to solve.

Building and running a quantum computer is an engineering task on a completely different scale to that of building and running a classical computer. Not only must qubits be shielded from random errors picked up from the environment, one must also avoid unwanted interactions between multiple qubits within the same device. In order to avoid such problems, the number of successive operations on qubits (i.e., 'circuit depth') are limited on current quantum hardware. It is expected that to give an advantage for large scale problems one must have many fault-tolerant qubits, which are obtained by error correction techniques. These involve encoding the state of a 'logical' qubit in the state of many 'physical' qubits. As this is a difficult engineering task, quantum computers with fault-tolerant qubits are not available at the time of writing. Current machines are known as Noisy Intermediate-Scale Quantum (NISQ) Computers. At the time of writing, they have circa 100 physical qubits. But this number is increasing fast.





**Figure 3:** Example of a quantum circuit of the kind used in our system: the Hadamard gate  $\mathbf{H}$ , the X-rotation gate  $\mathbf{R}_x(\beta)$  by angle  $\beta$ , the controlled Z-rotation gate (i), part of which is a Z-rotation gate  $\mathbf{R}_z(\delta)$  by angle  $\delta$ , and finally the **CNOT** gate. (Taken from [Lorenz et al., 2021] and used with permission.)

## 5 DisCoCat Modelling

Development in the growing new field of QNLP is greatly facilitated by the Distributional Compositional Categorical (DisCoCat) modelling of natural language semantics. In DisCoCat, grammar dictates the composition of word-meanings to derive the meaning of a whole sentence [Coecke et al., 2010].

DisCoCat is a natural framework to develop natural language-like generative music systems with quantum computing. At the core of DisCoCat, as the model was originally formulated, is Joachim Lambek’s, algebraic model of pregroup grammar; the curious reader may refer to [Coecke et al., 2013] for a more rigorous mathematical treatment.

In DisCoCat, the compositional structure of language is captured by (symmetric) monoidal categories, a.k.a. *process theories*. This mathematical formalism comes equipped with a formal graphical notation in the form of *string diagrams*. Meaning is encoded by a mapping, which endows semantics to the diagram representing the grammatical structure of a text. This is in fact how DisCoCat was introduced in the first instance, by sending words to vectors, or in general, higher-order tensors, which are connected according to grammatical dependencies to form a tensor network. Contracting the tensors along the connections results in the derivation of the meaning of a text.

Categorical quantum mechanics (CQM) is a framework that reformulates quantum theory in terms of process theories. In this context, string diagrams describe quantum protocols involving quantum states, transformations, and measurements [Abramsky and Coecke, 2008], [Coecke and Kissinger, 2017]. Low-level and fine-grained diagrammatic calculi, which build on the philosophy of CQM, such as ZX-calculus [Coecke and Duncan, 2011], are promising complementary — or even alternative — approaches to reasoning about quantum systems and processes. We see then that string diagrams provide a common formalism, via which we can associate language with quantum theory. After all, Hilbert spaces, which is where quantum states are encoded, are vector spaces. By analogy, many-body quantum states encode word-meanings. And grammatical reductions correspond to processes such as quantum maps, quantum effects, and measurements. To take advantage of such analogy, we employ DisCoPy [Felice et al., 2020] to perform grammar-based musical experiments on quantum processors. DisCoPy is an open-source toolbox for manipulating string diagrams and implementing mappings to underlying semantics of our choosing.

Among its features, DisCoPy includes tools for defining Context-Free Grammars (CFGs), Lambek’s

pregroup grammars [Lambek, 1999], tensor networks [Orus, 2014], ZX-calculus, and other diagram-based reasoning processes, under the same compositional paradigm. In essence, and particular to this work, DisCoPy provides the means for mapping representations of musical systems into quantum circuits, encoding quantum processes, to be implemented on a quantum computer.

Prior to the work presented here, [Meichanetzidis et al., 2020] and [Meichanetzidis et al., 2021] used DisCoPy and Cambridge Quantum’s compiler t|ket) [Sivarajah et al., 2020] to develop and deploy a pioneering QNLP experiment on quantum hardware. Here the sentences were represented as parameterised quantum circuits and the parameters were trained to perform a sentence-classification task. A larger-scale experiment of the same nature was reported by [Lorenz et al., 2021]. The models built for those experiments have been further developed into a Python library for QNLP known as lambeq [Kartsaklis et al., 2021], which is named in homage to Joachim Lambek. Indeed, part of the work reported in this chapter was conducted in tandem with the development of lambeq.

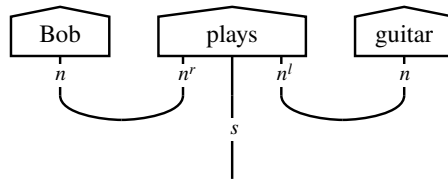
## 5.1 A Musical DisCoCat Model

In pregroup grammars, the syntax of a sentence is captured by a finite product of words of different pregroup types. In our case, musical snippets will play the role of words, and musical compositions the role of sentences.

Let us define a musical composition  $\sigma$  as a finite product of musical snippets  $w$  as follows:  $\sigma = \prod w_i$ , where the product operation captures the sequential nature of the string representing a musical sequence, in analogy with words being placed side by side in a sentence. Each snippet is assigned a pregroup type  $t_w = \prod b_i^{k_i}$  comprising a product of basic types  $b_i$  from a set of types  $B$ . Basic types also have adjoints:  $k_i \in \{\dots, ll, l, \_, r, rr, \dots\}$ . Then, the type of a musical composition  $\sigma$  is simply the product of the types of its snippets.

A composition is deemed valid, that is, grammatically correct — or musical — if and only if its type reduces to a special type  $s \in B$ . These reductions take place by means of an algebraic process of pair-wise annihilations of basic types and their adjoints according to the following rules:  $b^l \cdot b \rightarrow 1$ ,  $b \cdot b^r \rightarrow 1$ ,  $b^{ll} \cdot b^l \rightarrow 1$ ,  $b^r \cdot b^{rr} \rightarrow 1$ ,  $\dots$ . To visualise this, let’s go back to natural language and consider a sentence with a transitive verb: *Bob plays guitar*. Here, the transitive verb is a process that expects two input nouns of, say, type  $n$ , on each side, in order to output a sentence type  $s$ . The type of the transitive verb is therefore denoted as  $n^r \cdot s \cdot n^l$ . There exists a grammatical reduction following the algebraic rules of pregroup types, as shown in Eq. 1.

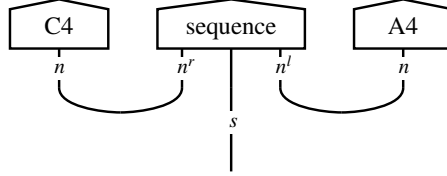
$$n \cdot (n^r \cdot s \cdot n^l) \cdot n \rightarrow (n \cdot n^r) \cdot s \cdot (n^l \cdot n) \rightarrow 1 \cdot s \cdot 1 \rightarrow s \quad (1)$$



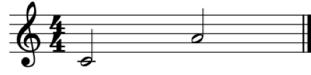
**Figure 4:** Pregroup diagram for the transitive sentence ‘Bob plays guitar’.

The DisCoCat string diagram for Eq. 1 is shown in Fig. 4. The wires carry a type. They bent in a U-shape, or cups, represent the reductions. Complex networks of processes and relationships can be designed by connecting boxes with input and output wires, and observing that the types are respected and are reducible as required.

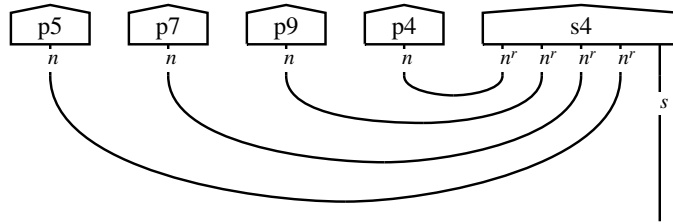
Such pregroup modelling is directly applicable to music. For instance, consider the example shown in Fig. 5. Here we defined a relationship  $n^r \cdot s \cdot n^l$  between two musical notes (type  $n$ ): C4 and A4. The relationship states that note A4 follows note C4 (Fig. 6). In this particular example, *sequence* is not a verb but an actual action to be carried out by some hypothetical generative music system.



**Figure 5:** Pregroup diagram for musical form shown in Fig. 6.



**Figure 6:** A sequence of two notes: C4 and A4.



**Figure 7:** Pregroup diagram for a musical composition form.

Fig. 7 depicts an example of a pregroup diagram representing a short musical composition, which is shown in Fig. 8. In this case there are five musical snippets (labelled as p4, p9, p7 and p5), each of which comprising several musical notes forming two entire musical bars each.

The semantic map is a functor  $\mathcal{F}$  that sends basic types  $(n, s)$  to vector spaces  $(N, S)$ . Take the example in Fig. 4, ‘Bob plays guitar’. The DisCoCat diagram can be decomposed as  $\text{diagram} = (\text{Bob} \otimes \text{plays} \otimes \text{guitar}) \circ \text{cups}$ . The second word ‘plays’ is a transitive verb and has the pregroup type  $n^r \cdot s \cdot n^l$ , whilst ‘Bob’ and ‘guitar’ have pregroup type  $n$ . The functor  $\mathcal{F}$  sends the diagram to vector space semantics by dropping the adjoints  $(l, r)$  and sending the pregroup type to the corresponding vector spaces. Based on their pregroup types, the words of the sentence will be sent to tensors of the corresponding vector space (Eq. 2).

$$\begin{aligned} \mathcal{F}(\text{Bob}), \mathcal{F}(\text{guitar}) &\in \mathcal{F}(n) = N \\ \mathcal{F}(\text{plays}) &\in \mathcal{F}(n^r \cdot s \cdot n^l) = N \otimes S \otimes N \end{aligned} \quad (2)$$

Moreover,  $\mathcal{F}$  translates all reductions cups to tensor contractions. Conceptually, the functor can be thought of as something that performs tensor contraction between the words and the cups to return a vector for the whole sentence (Eq. 3). In section 6.4, we will relay the details of the corresponding mapping to quantum processes.

$$\mathcal{F}(\text{diagram}) = (\mathcal{F}(\text{Bob}) \otimes \mathcal{F}(\text{plays}) \otimes \mathcal{F}(\text{guitar})) \circ \mathcal{F}(\text{cups}) \in S \quad (3)$$

Exactly the same principles are applicable to music. For instance, assuming the example in Fig. 5, and a set of musical notes  $N$ , the relationship *sequence* acquires the meaning  $s = C4 \cdot R \cdot A4$  with  $\{C4, A4\} \in N$  and  $R \in (N \otimes S \otimes N)$ .

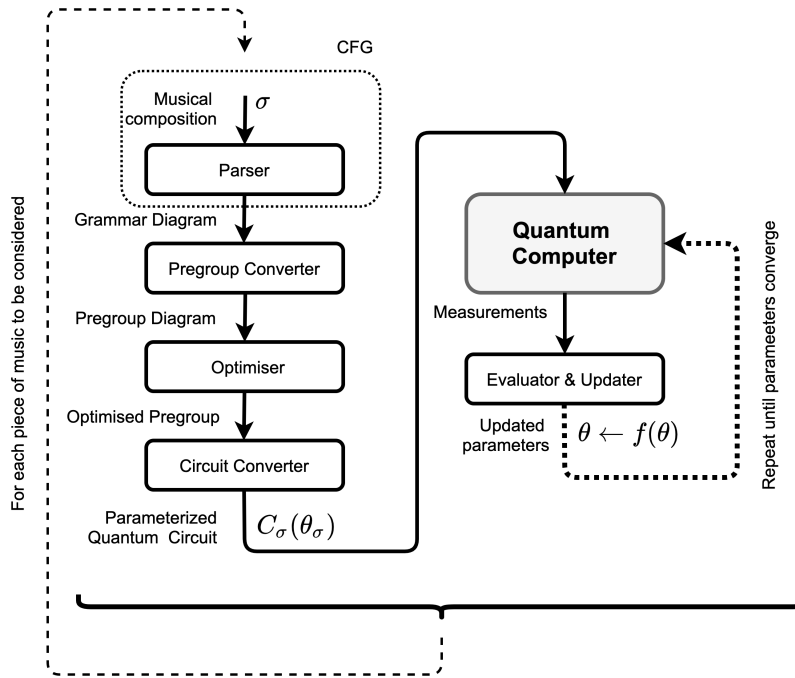


**Figure 8:** A short musical composition corresponding to the pregroup diagram in Fig.7.

## 6 Machine Learning of Music

In this section we introduce our DisCoCat music model and the quantum machine learning algorithm that we built to differentiate between two categories of music. Example code for the machine learning process can be found at [ <https://github.com/CQCL/Quanthoven> ]. A schematic overview of the algorithm is shown in Fig. 9.

In a nutshell, the algorithm learns from corpora generated by a bespoke context-free grammar (CFG). However, the quantum machine learning algorithm needs to see the structure of the compositions in the form of parameterised quantum circuits. Thus, the system needs to transform their underlying CFG structures into quantum circuits. This is done by converting CFGs into pregroups diagrams, which are then optimised before they are translated into quantum circuits. This optimisation is highly advisable because cups tend to produce more qubits than necessary when converting pregroup diagrams into a quantum circuit. The system then generates instructions for an appropriate quantum computer to run the circuit.



**Figure 9:** Schematic overview of our quantum machine learning algorithm.

## 6.1 Generating a Training Corpus with a Context-Free Grammar

Here we define a Context-Free Grammar (CFG) to generate musical compositions for piano, which will be used to teach the machine learning algorithm. And *Quanthoven* will use this CFG to generate new pieces after it has learnt to classify them.

The lexicon of our CFG contains four types of musical snippets. As it was mentioned already, a snippet is the equivalent of a word in natural language. Whereas a word is formed by letters from an alphabet, a snippet is formed by notes from a pitch framework<sup>4</sup>. Likewise, whilst combinations of words form sentences, combinations of snippets form musical sequences; that is, musical compositions.

As the second level of the Chomsky hierarchy, CFGs are expressive enough to model a vast portion of natural language. An informal, diagrammatic representation of a CFG is to draw words in the lexicon as boxes with one output wire. And production rules can be represented as boxes with multiple input wires and one output wire. Then, sequences can be built by freely composing these boxes. A more detailed explanation about combining CFGs with monoidal category theory can be found in Appendix A.1.

Verbal languages have different *types* of words, such as nouns, verbs, adjectives, and so on. Grammatical rules then dictate how they are combined to form sentences. Similarly, our composition

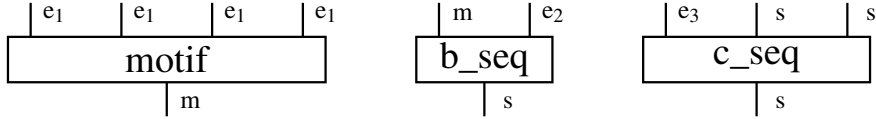
<sup>4</sup>In the context of this work, a musical pitch framework defines the notes that characterise a specific musical style or culture. An octave is defined by two notes, one having twice the pitch of another. Different pitch frameworks have their own way of subdividing the octave to create distinct scales of musical notes. Conventionally, there are 1,200 cents to the octave. Western European music normally subdivides the octave into 12 notes, equally-spaced at 100 cents. This forms the so-called chromatic scale. However, some cultures subdivide the octave into notes with different spacings between one another. For example, Slendro Indonesian music subdivides the octave into five notes, spaced by 240 cents from each other. And most Middle Eastern music divides the octave into six notes spaced by 200 cents. Furthermore, no matter how an octave is divided, the notes can be played at different registers, or heights, depending on the instrument at hand. For instance, most pianos can produce 88 notes, accommodating seven octaves. That is, one can play seven different notes ‘E’ on the piano (or ‘Mi’ in traditional nomenclature). The snippets discussed in this article are made from the piano’s 88 note-set.

system contains four *types* of snippets: ground (g), primary (p), secondary (s) and tertiary (t) snippets. Some examples are shown in Fig. 12. These will be represented below as  $s$ ,  $e_1$ ,  $e_2$  and  $e_3$ , respectively. We annotate the wires of the boxes with these types to enforce how the lexicon and production rules can be combined in a grammatical way to form musical compositions (Fig. 10).



**Figure 10:** Diagrammatic representation of snippets.

Languages have different kinds of sentences; e.g., simple, compound, complex, and compound-complex sentence structures. By the same token, here we have three types of musical sequences: motif, basic (b\_seq) and composite (c\_seq) sequences, respectively (Fig. 11).



**Figure 11:** Diagrammatic representation of musical sequences.

In a nutshell, a motif is composed of four primary snippets, and a basic sequence is composed of a motif followed by a secondary snippet. A composite sequence will always start with a tertiary snippet, followed by two elements. Each of them can be one of three options: a tertiary snippet, a basic sequence, or another composite one.

Unlike natural languages, the productions rules of this musical grammar are defined explicitly. Therefore it is possible to implement a parser to recover the derivation from the sequence. In fact, our CFG is parsable by a LL(1) parser, and hence is a LL(1) grammar [Rosenkrantz and Stearns, 1970]. In contrast, parsing natural language requires contextual understanding of the words, so a statistical CCG parser [Yoshikawa et al., 2017] combined with a conversion from CCG to DisCoCat [Yeung and Kartsaklis, 2021] is necessary. By explicitly implementing a parser, we show that the grammatical structure is recoverable from a sentence and need not be explicitly provided to the DisCoCat model: other models are welcome to take advantage of it.

Firstly, we generated a corpus of 100 compositions for piano. Then, we annotated the corpus according their meaning: *rhythmic* or *melodic*. As explained earlier, the overarching meaning of a composition describes an overall perceived property. Thus, compositions perceived as having prominent fast and loud rhythmic patterns were labelled as RIT (for rhythmic). Otherwise, those as having above all successions of notes forming melodies and harmonies were labelled as MEL (for melodic).

The annotation of the corpus was carried out manually. The labellings were agreed upon by the authors after they independently listened to the 100 compositions and reached consensus. Eq. 4 shows how the corpus  $\Sigma$  of annotated compositions  $\sigma$  look like. The symbols  $t_3$ ,  $g_1$ ,  $p_3$ , and so on, represent the snippets that form a respective composition.

$$\begin{aligned} \Sigma = \{ & (1, \text{MEL}, [t_3, g_1, p_3, p_1, p_3, p_3, s_3]), \\ & (2, \text{RIT}, [p_4, p_9, p_7, p_5, s_1]), \\ & (3, \text{RIT}, [t_3, g_2, g_2]), \\ & \dots \} \end{aligned} \quad (4)$$



**Figure 12:** Examples of primary, secondary and tertiary snippets, respectively.

The complete lexicon of musical snippets are provided in Appendix A.2. Although some snippets are more rhythmic than others, and vice-versa, there is considerable overlap of snippets between the two categories. This ensures that the machine learning task is realistic. After annotation, the corpus is split into training, development, and test sets with a 50 + 25 + 25 split, respectively. The compositions last for different durations, ranging for a few seconds to various minutes. A symbolic representation of the dataset is available in Appendix A.3<sup>5</sup>

## 6.2 Pregroup Converter: from Context-Free Grammars to Pregroup Grammars

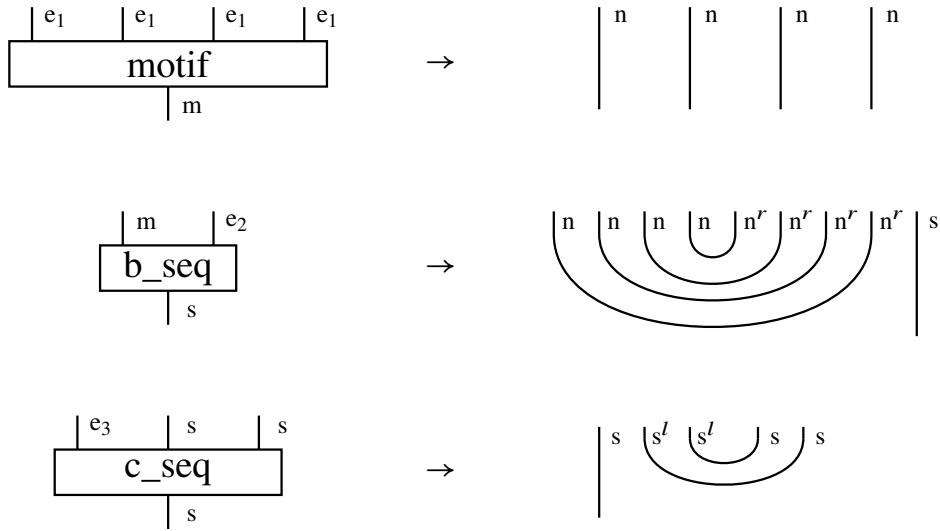
In this section we give a functorial mapping from musical CFGs to pregroups. The DisCoCat model is more naturally suitable for pregroup grammars as they are lexicalised grammars. In DisCoCat, most of the grammatical information is stored in the lexicon instead of the grammatical production rules. This is not the case for CFGs. Hence the need to convert to pregroups.

In natural language, a transitive verb is a word that takes a noun on the left and a noun on the right to give a sentence, and so has the pregroup type  $F(VP_{TV}) = n^r \cdot s \cdot n^l$ . An adjective is a word that takes a noun on the right to give another noun, and so has the pregroup type  $F(Adj) = n \cdot n^l$ .

Let us reason with analogies here: first we fix the types ground and primary elements,  $s$  and  $e_1$ , to be the atomic types for our pregroup grammar. This is done by setting  $F(s) = s$  and  $F(e_1) = n$ . Since a motif is made of four primary elements, we give it the pregroup type  $F(m) = n \cdot n \cdot n \cdot n$ . A secondary element takes a motif on the left to give a sentence, it has the pregroup type  $F(e_2) = F(m^r \cdot s) = F(e_1 \cdot e_1 \cdot e_1 \cdot e_1)^r \cdot s = n^r \cdot n^r \cdot n^r \cdot n^r \cdot s$ . Finally, a tertiary element takes two sentences on the right to give another sentence, and so it has the pregroup type  $F(e_3) = s \cdot s^l \cdot s^l$ .

Once we have used our functor to convert the CFG types into the appropriate pregroup types, the CFG production rules become pregroup contractions, which are represented as ‘cups’ (Fig. 13). An example of this conversion on an actual musical grammar can be found in Fig. 14.

<sup>5</sup>Audio files are available at [ <https://github.com/CQCL/Quanthoven/> ]. Note, these were synthesised rather than played on



**Figure 13:** Diagrammatic representation of pregroup contractions.

### 6.3 Optimiser: Diagrammatic Rewriting

Once a CFG diagram is converted into a pregroup grammar diagram, a circuit functor can be applied to transfer the pregroup diagram into a quantum circuit; this will be explained below, in section 6.4. However, due to the length of the compositions produced by our generation procedure, the resulting quantum circuit, after the functorial conversion has been applied, has too many qubits to be efficiently simulated before being executed on quantum hardware.

Furthermore, to perform the highly entangled qubit measurements — or Bell measurements — required by pregroup grammars, we need to use postselections. To perform a postselection on a qubit, one measures the qubit and then discards the circuits that do not have the desired outcome on the respective qubit. As a rough estimate, postselection halves the number of usable shots<sup>6</sup>. By performing  $n$  postselections, we reduce the number of usable shots by a factor of  $2^n$ . For example, if a 12-qubit circuit requires 11 postselections, then we would expect approximately only 4 out of 8192 shots to be usable after postselection. This makes our results even more sensitive to the noise of a quantum computer.

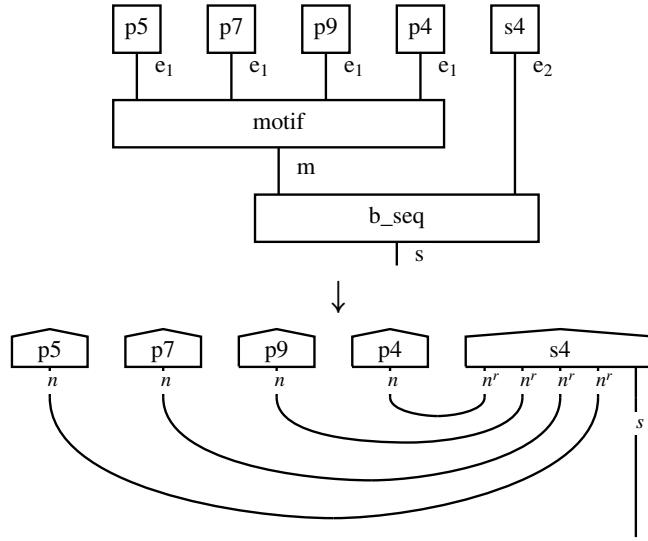
One way to ameliorate both problems is to perform some diagrammatic rewriting. By rotating certain words in the pregroup diagram, we can eliminate the cups that correspond to Bell measurements using the ‘snake equation’. In terms of linear algebra, the rotation operation corresponds to taking the transpose. This rewriting technique, first used in [Lorenz et al., 2021], can also be applied to our musical pregroup diagrams. In Fig. 15, the number of postselections performed is halved from 8 to 4, so the number of useful shots has increased by a factor of 16. By applying this rewriting technique to our diagrams, we reduce the worst-case circuit size in our training set from 25 qubits to 13 qubits. This is a reduction of 12 postselections. Thus, the number of shots required is reduced by a factor of 4096.

---

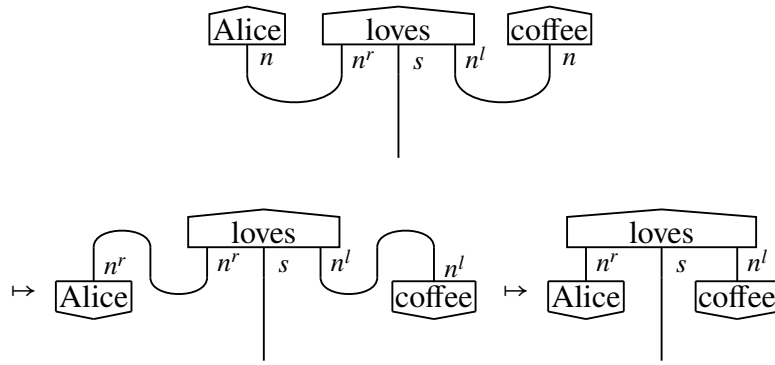
a real instrument by a musician.

<sup>6</sup>‘Shots’ is a term used to refer to repeated runs of a circuit.





**Figure 14:** Converting a CFG diagram into pregroup grammar.



**Figure 15:** Optimisation through rewriting.

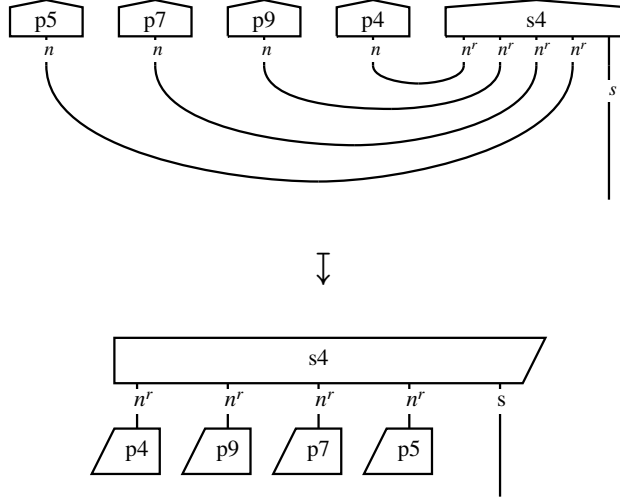
## 6.4 Circuit Converter: Translating Musical Compositions into Quantum Circuits

With DisCoCat, a string diagram for a musical composition  $\sigma$  (which is generated by a CFG and thus has a corresponding pregroup diagram) can be translated onto a parameterised quantum circuit  $C_\sigma(\theta_\sigma)$  over a parameter set  $\theta_\sigma$ . The translation assigns a number of qubits  $q_b$  to each wire carrying a type  $b$ . Therefore, Hilbert spaces are defined on the wires (Fig. 16).

The snippet-state of the  $\eta$ -th snippet in a string diagram is reinterpreted as a pure quantum state prepared from a given reference product-state by a circuit  $C_\eta$  (Eq. 5).

$$C_\eta(\theta_\eta) |0\rangle^{\otimes q_\eta}, \quad q_\eta = \sum_{i=1}^{|\eta|} q_{b_i} \quad (5)$$

The width of a snippet-circuit depends on the number of qubits assigned to each of the  $|\eta|$ -many basic types  $b \in B$  assigned to the snippet. Given a composition  $\sigma$ , the system firstly concatenates the snippet-states of the snippets as they appear in the composition. Concatenation corresponds to taking their tensor product (Eq. 6).



**Figure 16:** The pregroup diagram in Fig. 14 rewritten by removing the caps.

$$C_{\sigma}(\theta_{\sigma})|0\rangle^{\otimes q_{\sigma}} = \bigotimes_{\eta_j} C_{\eta_j}(\theta_{\eta_j})|0\rangle^{\otimes q_{\eta_j}} \quad (6)$$

The circuit in Fig. 17 prepares the state  $|\sigma(\theta_{\sigma})\rangle$  from the all-zeroes basis states. A musical composition is parameterised by the concatenation of the parameters of its snippets:  $\theta_{\sigma} = \cup_{\eta \in \sigma} \theta_{\eta}$ , where  $\theta_{\eta}$  defines the snippet-embedding  $|\eta(\theta_{\eta})\rangle$ .

Then, Bell effects are applied, as determined by the cups representing pregroup reductions. Cups are interpreted as entangling circuits that implement a Bell effect by postselecting on a Bell state. This ‘effect of grammar’, denoted  $g$ , acts on the state of the composition and returns the state  $g(|\sigma(\theta_{\sigma})\rangle)$ .

## 6.5 Training the System to Classify

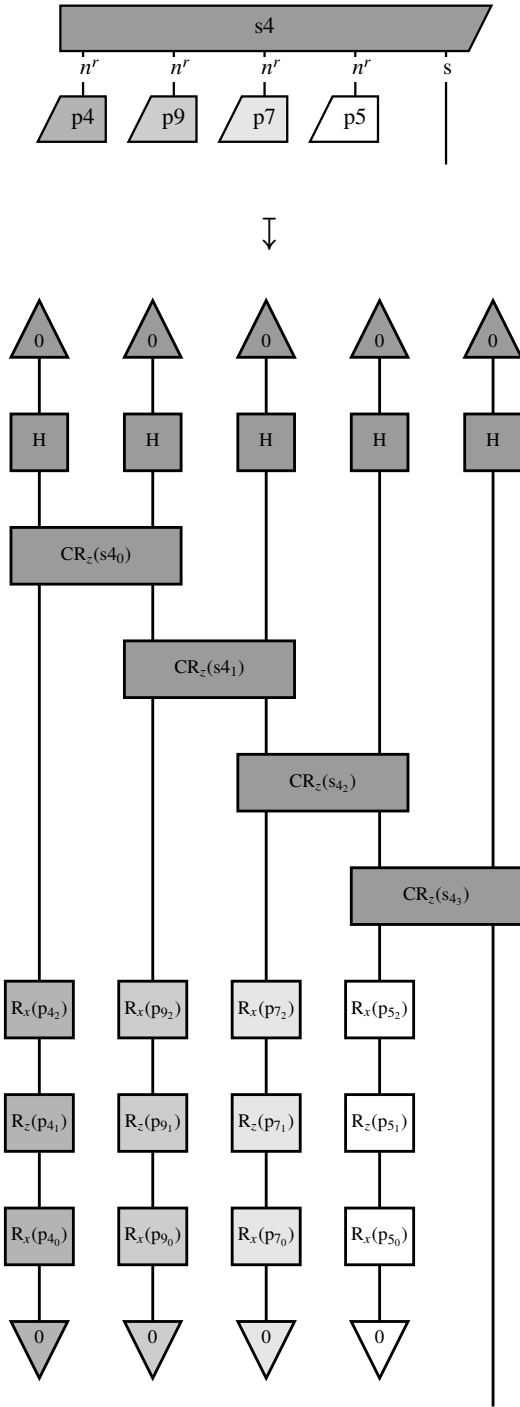
Here we describe how we trained the system with the aforementioned musical dataset to distinguish between two categories of music: rhythmic or melodic.

As we are performing binary classification, we choose to represent the output of the system in one-hot encoding. This means that our two class labels are  $[0, 1]$  and  $[1, 0]$  corresponding to ‘melodic’ and ‘rhythmic’ music respectively. In the DisCoCat model, this can be conveniently achieved by setting the sentence dimension space (recall the whole circuit is represented by an open sentence wire) of our DisCoCat model to one qubit. The circuit  $C_{\sigma}$  for each composition  $\sigma$  is evaluated for the current set of parameters  $\theta_{\sigma}$  on the quantum computer giving output state  $|C(\theta_{\sigma})\rangle$ . The expected prediction  $L_{\text{pred}}^i(\sigma, \theta)$  is given by the Born rule in Eq. 7, where  $i \in \{0, 1\}$  and  $\theta = \cup_{\sigma \in \Sigma} \theta_{\sigma}$ .

$$L_{\text{pred}}^i(\sigma, \theta) := |\langle i | C_{\sigma}(\theta_{\sigma}) \rangle|^2 \quad (7)$$

By running and measuring the outcome (either  $[0, 1]$  or  $[1, 0]$ ) of the circuit many times, the average outcome will converge towards this value. Then  $L_{\text{pred}}^i(\sigma, \theta)$  is normalised to obtain a smooth probability distribution (Eq. 8).

$$l_{\text{pred}}^i(\sigma, \theta) := \frac{L_{\text{pred}}^i(\sigma, \theta) + \epsilon}{\sum_j (L_{\text{pred}}^j(\sigma, \theta) + \epsilon)} \quad (8)$$



**Figure 17:** The pregroup diagram in Fig. 16 converted to a quantum circuit, according to the IQP and Euler decomposition ansätze. Regions of the diagram and circuit have been coloured to illustrate the transformation of each part of the diagram into a quantum circuit.

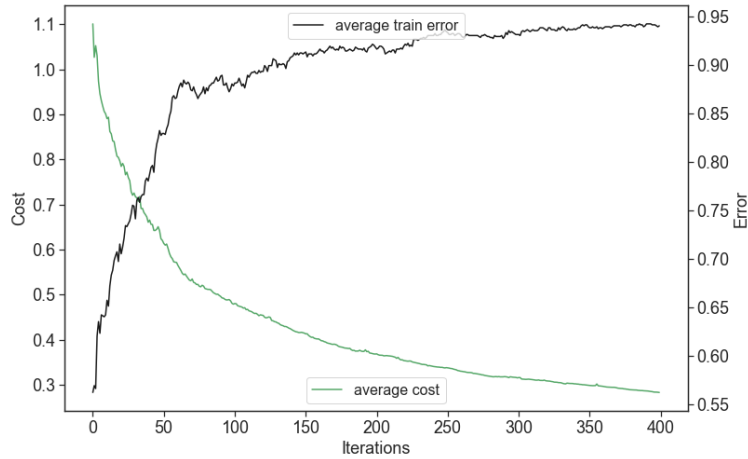
The smoothing term is chosen to be  $\epsilon = 10^{-9}$ . The predicted label is obtained from the probability distribution by setting a decision threshold  $t$  on  $l_{\text{pred}}^0(\sigma, \theta)$ : if  $t < l_{\text{pred}}^0(\sigma, \theta)$  then the predicted label is  $[0, 1]$ . And if  $t \geq l_{\text{pred}}^0(\sigma, \theta)$  then the predicted label is  $[1, 0]$ . This threshold can be adjusted depending on the desired sensitivity and specificity of the system. For this chapter, the threshold  $t = 0.5$  is selected.

To predict the label for a given snippet well, we need to use optimisation methods to find optimal parameters for our model. We begin by comparing the predicted label with the training label using a loss function. Since we are performing a classification task, the binary cross-entropy loss function is chosen (Eq. 9).

$$\text{BCE}(\theta) = \sum_{\sigma} \sum_{i \in \{0,1\}} l_{\text{label}}^i(\sigma, \theta) \log(l_{\text{pred}}^i(\sigma, \theta)) \quad (9)$$

A non-gradient based optimisation algorithm known as SPSA (Simultaneous Perturbation Stochastic Approximation) is used to minimise our loss function [Spall, 1998]. Alternatively, one could use gradient-based optimisation algorithms by differentiating the DisCoCat diagram [Toumi et al., 2021]. However, SPSA was found to be sufficient for our purposes.

In order to minimise the loss function the system learns to classify the compositions by adjusting the parameters of the musical snippets. Philosophically, this is in contrast to typical connectionist machine learning models - that is, neural networks - where the weights of the network are adjusted rather than the weights of the word embeddings. This is because pregroup grammars are lexical. And in lexical grammars, whilst words have parameters that can be changed, the grammatical structure cannot be changed.



**Figure 18:** Average over 20 different runs of a classical simulation of the training set showing the cost (green, left hand side axis) and the training error (black, right hand side axis).

Unfortunately, the training process cannot be performed entirely on a real quantum device at the time of writing. Access to quantum hardware still is limited, due to the demand for and the limited number of available quantum processors. This can be problematic for running variational tasks where circuits must be run for many iterations in order to get results.

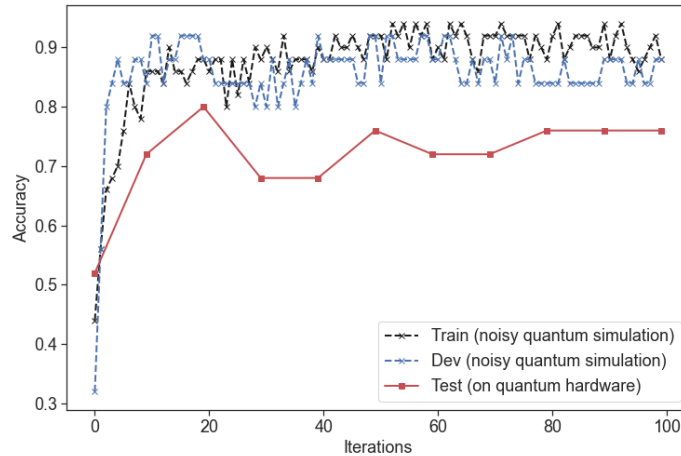
Therefore, we firstly pre-train the model parameters using exact tensor contraction of the quantum circuits on a classical computer. We use Python’s JAX library<sup>7</sup> for just-in-time compilation, which

<sup>7</sup><https://jax.readthedocs.io/en/latest/>

speeds up the pre-training. This speed up allows us to efficiently perform grid search and find the best hyper-parameter settings for the SPSA optimiser. The results for such a classical simulation are shown in Fig. 18 for the final settings chosen for the subsequent training.

Then, we simulate the quantum process on a classical machine. In order to do this, we need to take into account the statistical nature of our experiments, which would involve running the same circuit thousands of times to obtain measurement statistics. And we also need to bear in mind the noise that is present in a real device. The parameters learnt from the exact pre-training is transferred to a model that closely resembles the real quantum device, which includes noise simulation. This will ultimately improve testing performance on the real quantum device. So, the same circuit is simulated 8,192 times with a noise model specific to the quantum device that will be used for testing. This process is carried out for each composition in the training set, for a chosen number of iterations. The train and development set results from the quantum simulations are shown in Fig. 19.

Once the pre-training phase is complete, and we are happy with the simulated results on the development set, then we take the parameters from the noisy quantum simulation and evaluate the test set on a real quantum computer. We used IBM Quantum’s device `ibmq_guadalupe`, which is a 16-qubit superconducting processor with a quantum volume of 32. Despite the expected differences between simulation and a quantum hardware processing, we can see that the model learns to classify compositions in the test set with an impressive final accuracy of 76%, as shown in Fig. 19.



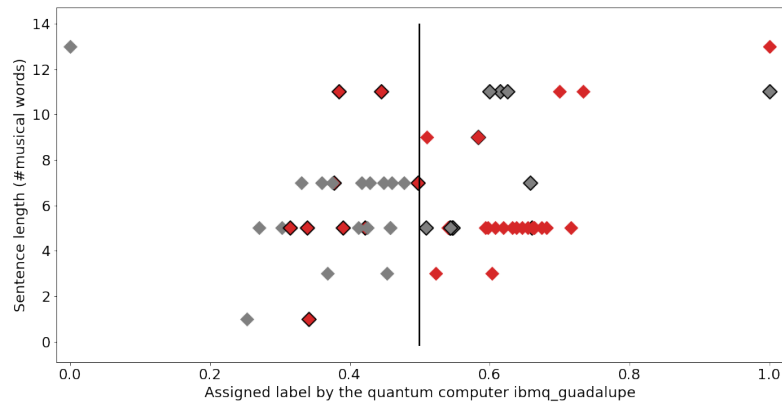
**Figure 19:** Noisy quantum simulation to train the model (black), noisy quantum simulation of the development set (blue) and the test set run on on quantum hardware (red). The noise model used for the noisy simulation was that of the `ibmq_guadalupe` device, which is the hardware that we subsequently used for running the test set.

### 6.5.1 Further Classification Tests

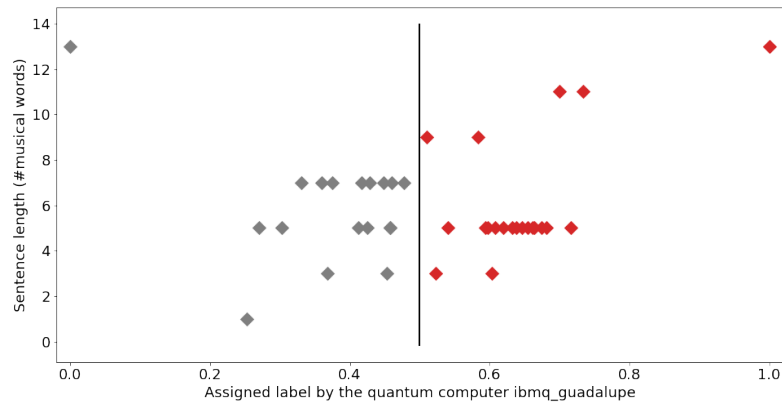
The results from the testing with quantum hardware (Fig. 19) exceeded our expectations. We conducted additional tests to probe the system further, which confirmed the system’s positive performance.

For the new tests we ran our CFG to produce two additional datasets: one containing 60 compositions and another containing 90. As with the previous dataset, we auditioned and annotated the pieces manually. Again, both sets contain compositions of varying lengths. Then, we submitted the sets to the quantum classifier. Below we display the results from a different perspective than in Fig. 19.

Fig. 20 shows that the system correctly classified 39 compositions from the set of 60. Melodic pieces corresponds to a value plotted on the first element in the tuple of  $< 0.5$  (grey), whereas the Rhythmic ones corresponds to a value  $\geq 0.5$  (red). Misclassified compositions have a black outline around them, as well as being the wrong colour for the side of the graph that they appear on. For clarity, Fig. 21 plots only those compositions that were classified correctly.



**Figure 20:** Quantum computing classification of dataset with 60 compositions.



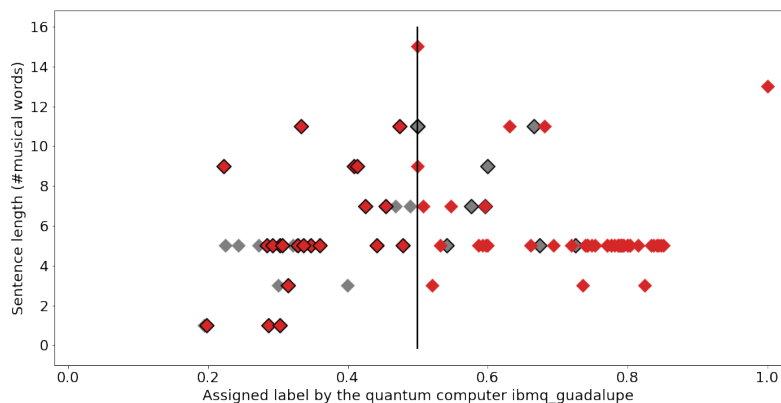
**Figure 21:** A version of Fig. 20 plotting only those pieces that were classified correctly.

Next, Fig. 22 shows that the system correctly classified 59 compositions from the set of 90. Again, melodic pieces corresponds to a value plotted on the first element in the tuple of  $< 0.5$  (grey), whereas the Rhythmic ones corresponds to a value  $\geq 0.5$  (red). And Fig. 23 shows the same results, but plots only those compositions that were classified correctly.

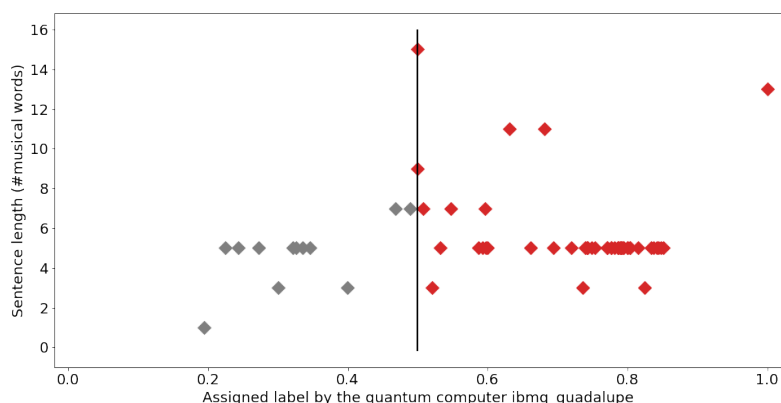
## 7 *Quanthoven*: Leveraging the Quantum Classifier to Compose

Composer Lejaren Hiller, who also was a chemistry teacher, allegedly is a pioneer of programming computers to generate music. In 1957, he teamed up with Leonard Isaacson, also a chemist and

composer, to program the ILLIAC I<sup>8</sup> (Illinois Automatic Computer) at the University of Illinois, USA, to compose music. The computer produced materials for their string quartet, entitled *Illiatic Suite*<sup>9</sup>.



**Figure 22:** Quantum computing classification of dataset with 90 compositions.



**Figure 23:** A version of Fig. 22 plotting only those pieces that were classified correctly.

In their initial experiments, they programmed the computer to do two things: (a) generate musical data<sup>10</sup> (pseudo-randomly), and (b) test if the data satisfied musical rules. Musical data that did not satisfy the rules were discarded. Conversely, those that satisfied the rules were kept for the ongoing composition. For instance, for the first movement of *Illiatic Suite* they set the machine with rules from the famous treatise *Gradus ad Parnassum*, written by Joseph Fux in the 17<sup>th</sup> century [Mann, 1965].

Hiller’s approach became an archetype for algorithmic music composition with computers, which is often referred to as the *generate-and-test* approach (Fig. 24). Various variants have been implemented, with increasingly sophisticated methods for generating musical data and probing them [Edwards, 2011] [Miranda, 2001]. Such systems might generate notes that are checked one at a time, or phrases, larger

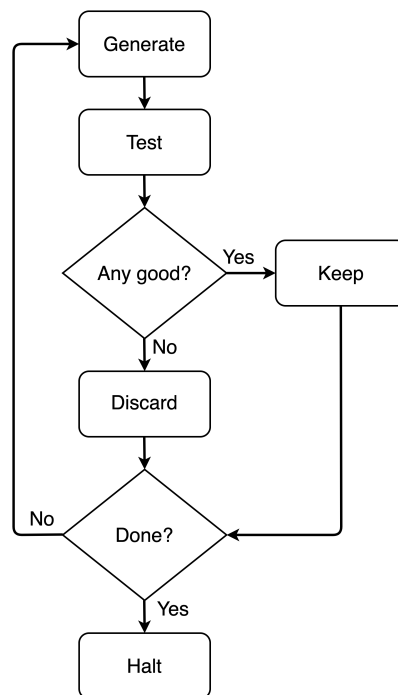
<sup>8</sup>This was the first von Neumann architecture computer built and owned by a university in the USA. It was put into service on September 22, 1952.

<sup>9</sup>Later retitled as “String Quartet No. 4”.

<sup>10</sup>That is, sequences of symbols representing notes, rhythms and expressions such as ‘pizzicato and ‘arco’, and so on.

sequences or even entire pieces. Human approval often takes place as well, whereby composers may further discard or amend computer-validated materials.

The generate-and-test approach also informed the development of Artificial Intelligence (AI) techniques for musical composition [Miranda, 2021a] [Fernandez and Vico, 2013], including highly praised constraint-satisfaction methods [Anders and Miranda, 2009]. Moreover, the generate-and-test method epitomises Evolutionary Computing techniques for music composition; e.g., using genetic algorithms [Miranda and Biles, 2007]. Such systems generate new sequences using biologically inspired processes (e.g., genetic mating, reproduction and mutation) and then employ ‘fitness measurements’ to evaluate them [Miranda, 2011].



**Figure 24:** Archetypal generate-and-test approach to algorithmic music composition.

The quantum machine learning paradigm discussed above fits the generate-and-test approach to algorithmic music composition very nicely. To probe this, we developed *Quanthoven*, a system for composition with the classifier presented above acting as the evaluator. In a nutshell, the CFG (section 6.1) generates compositions and the trained classifier (section 6.5) evaluates them (Fig. 24).

At the introduction, we mentioned that our ambition is to develop systems to aid musicians to compose ‘music for Alice’s tea party’, or a ‘tune to make Bob feel energetic’. In order to do this, the machine is required to handle the meaning of music; that is, perceived properties of musical compositions, which holistically characterise their style. Our proof-of-concept gives a good hint about how this can be achieved.

Effectively, our machine learning algorithm learned what we mean by ‘melodic’ and ‘rhythmic’ music. The CFG can generate pieces with snippets combined in ways that may render a piece more melodic than rhythmic, or vice-versa. Some might be neither to our ears, others might be both. But the quantum classifier is able to suggest whether a CFG-generated piece is one or the other.

As a demonstration, we set ourselves the task of composing 4 pieces of music with *Quanthoven*: two for relaxing at a tea party and two aimed at inducing excitement. We set *Quanthoven* to generate two dozen pieces and save four, two for each category. The others were discarded on the spot; we do not even know how they sounded like.



The saved pieces were encoded as MIDI files. Then, we uploaded these files into a music editor, formatted them, added titles and other details, and printed the scores. We hired a professional pianist to record them. The two pieces for relaxing at a tea party were entitled as *Bob's Chamomile Slowdown* and *Alice's Mushroom Trip*. The ones aimed at making one feel excited were entitled *Bob's Cigar Buzz* and *Alice's Caffeine Rush*.

The score for *Alice's Caffeine Rush* is shown in Figs. 25 and 26. The eyes of a trained musician would immediately see patterns that characterise this piece as rhythmic, in particular the section starting at bar 15. The scores for the other three pieces are provided in Appendix A.4.

Alice's Caffeine Rush 1

Quanthoven

$\text{♩} = 96$

-- CQ & ICCMR, Aug 2021 --

**Figure 25:** First page of the composition *Alice's Caffeine Rush*.

## 8 Final Remarks

Recordings of all four pieces are released in [SoundClick](#) [Miranda, 2021b]. You are invited to listen to the compositions and make your own mind whether you agree with *Quanthoven* or not. We are rather satisfied with the results. But of course, they are debatable.

This chapter started by saying that people have different tastes and opinions about music. And it also noted that people perceive and react to music differently from each other. However, the five authors agreed of the ‘meanings’ here. And the aim was to teach the machine to classify the pieces according to our opinions. Objectively, the system does what it says on the tin.

The image shows a musical score for the second page of the composition 'Alice's Caffeine Rush'. The score is written for piano and consists of five systems of music. The first system starts at measure 15 with a tempo of 100 and a forte (f) dynamic. The second system continues to measure 20. The third system continues to measure 25. The fourth system starts at measure 25 with a tempo of 96. The fifth system concludes the piece with a double bar line and a fortissimo (ff) dynamic marking.

**Figure 26:** Second page of the composition *Alice's Caffeine Rush*.

We made the conscious decision of conducting this experiment with music generated from a CFG designed from the ground up, completely from scratch. Of course, we could have written a piece of software to extract a CFG from given corpora of musical scores; e.g., [Bod, 2001]. Ultimately, we would extract this information directly from audio recordings. However, we wanted to probe our approach with something other than existing styles of music. Moreover, we wanted to ensure that, at this stage of our research, the two categories in question share the same grammatical structure. Effective annotation methods for a much larger database should also be developed; for instance, based on information extracted from social media.

As quantum computing hardware and error-correction technologies evolve, we will certainly continue pushing the boundaries in tandem, with compositions of increased sophistication, more instruments, longer durations, more categories (or ‘meanings’), and so on. These will probably require

generative mechanisms other than CFG, something along the lines of transformational-generative grammars. Also, an important next step would be to design an actual quantum generative music engine drawing directly from the classifier, as opposed to using the classifier as a filter.

## 9 Acknowledgements

This project was developed during E. R. Miranda’s research residency at Cambridge Quantum in 2021, which was partially funded by the QuTune Project<sup>11</sup>. The authors thank pianist, Lauryna Sableveciute, and recording engineers, John Lowndes and Manoli Moriaty, at Liverpool Hope University, for the recordings of *Bob’s Chamomile Slowdown*, *Alice’s Mushroom Trip*, *Bob’s Cigar Buzz*, and *Alice’s Caffeine Rush*.

## References

- S. Abramsky and B. Coecke. Categorical quantum mechanics. In K. Engesser et al., editor, *Handbook of Quantum Logic and Quantum Structures*. Elsevier, Amsterdam, 2008. URL <https://arxiv.org/abs/0808.1023>.
- A. Adiyansjah, A. S. Gunawan, and D. Suhartono. Music Recommender System Based on Genre using Convolutional Recurrent Neural Networks. *Procedia Computer Science*, 157:99–109, 2019. doi: 10.1016/j.procs.2019.08.146.
- T. Anders and E. R. Miranda. Interfacing Manual and Machine Composition. *Contemporary Music Review*, 28(2):133–147, 2009. doi: 10.1080/07494460903322422.
- D. Bakker and F. H. Martin. Musical chords and emotion: Major and minor triads are processed for emotion. *Cognitive Affective and Behavioral Neuroscience*, 15(1):15–31, 2014. doi: 10.3758/s13415-014-0309-4.
- M. Baroni. Musical Grammar and the Study of Cognitive Processes of Composition. *Musicae Scientiae*, 3(1):3–21, 1999.
- J. C. Bod. Probabilistic Grammars for Music. 2001. URL [https://www.researchgate.net/publication/2402113\\_Probabilistic\\_Grammars\\_for\\_Music](https://www.researchgate.net/publication/2402113_Probabilistic_Grammars_for_Music).
- B. de Boer. Evolution and self-organisation in vowel systems. *Evolution of Communication*, 3(1):79–103, 1999.
- J. Cherston, E. Hill, S. Goldfarb, and J. A. Paradiso. Musician and Mega-Machine: Composition Driven by Real-Time Particle Collision Data from the ATLAS Detector. In *Proceedings of NIME 2016*, pages 11–15, 2016.
- N. Chomsky. *The Logical Structure of Linguistic Theory*. Springer, New York, NY, 1975.
- N. Chomsky. *Language and Mind*. Cambridge University Press, Cambridge, 2006.
- B. Coecke. Compositionality as we see it, everywhere around us. 2021. URL <https://arxiv.org/abs/2110.05327>.

---

<sup>11</sup>QuTune kick-started in the Spring of 2021 thanks to funding kindly provided by the UK National Quantum Technologies Programme’s QCS Hub: [ <https://iccmr-quantum.github.io/> ]

- B. Coecke and R. Duncan. Interacting quantum observables: categorical algebra and diagrammatics. *New Journal of Physics*, 13(4):04301, 2011. URL <https://arxiv.org/abs/0906.4725>.
- B. Coecke and A. Kissinger. *Picturing Quantum Processes: A First Course in Quantum Theory and Diagrammatic Reasoning*. Cambridge University Press, Cambridge, 2017.
- B. Coecke, S. Mehrnoosh, and S. Clark. Mathematical foundations for a compositional distributional model of meaning. *Linguistic Analysis*, 36(1-4):345–384, 2010. URL <https://arxiv.org/abs/1003.4394>.
- B. Coecke, E. Grefenstette, and M. Sadrzadeh. Lambek vs. Lambek: Functorial vector space semantics and strings diagrams for Lambek calculus. *Annals of Pure and Applied Logic*, 164:1079–1100, 2013. doi: 10.1016/j.apal.2013.05.009.
- D. Cope. Recombinant Music: Using the Computer to Explore Musical Style. *Computer*, 24(7):22–28, 1991. doi: 10.1109/2.84830.
- D. Cope. *The Algorithmic Composer*. A-R Editions, Madison, WI, 2000.
- I. Daly, D. Williams, J. Hallowell, F. Hwang, A. Kirke, A. Malik, J. Weaver, E. R. Miranda, and S. J. Nasuto. Music-induced emotions can be predicted from a combination of brain activity and acoustic features. *Brain and Cognition*, 101:1–11, 2015. doi: 10.1016/j.bandc.2015.08.003.
- C. Dawson, D. Aalto, J. Šimko, M. Vainio, and M. Tervaniemi. Musical Sophistication and the Effect of Complexity on Auditory Discrimination in Finnish Speakers. *Frontiers in Neuroscience*, 11:213, 2017. doi: 10.3389/fnins.2017.00213.
- M. Edwards. Algorithmic Composition: Computational Thinking in Music. *Communications of the ACM*, 54(7):58–67, 2011.
- L. Eliot. *Early Intelligence*. Penguin Books, London, 1999.
- G. de Felice, A. Tuomi, and B. Coecke. DisCoPy: Monoidal Categories in Python. Presented at Applied Category Theory 2020 conference, 2020. URL <https://arxiv.org/abs/2005.02975>.
- J. D. Fernandez and F. Vico. AI Methods in Algorithmic Composition: A Comprehensive Survey. *Journal of Artificial Intelligence Research*, (48):513–582, 2013.
- S. L. Frank, R. Bod, and M. H. Christiansen. How hierarchical is language use? *Proceedings of the Royal Society B*, 279(1747):4522–4531, 2012. doi: 10.1098/rspb.2012.1741.
- P.-Y. Gouyon, J.-P. Henry, and J. Arnould. *Gene Avatars: The Neo-Darwinian Theory of Evolution*. Kluwer Academic/Plenum Publishers, New York, NY, 2002.
- J. Harley. Generative Processes in Algorithmic Composition: Chaos and Music. *Leonardo*, 28(3): 221–224, 1995.
- J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Boston, MA, 1979.
- K. J. Hsü and A. J. Hsü. Fractal geometry of music. *Proceedings of the National Academy of Sciences, Physics*, 87:938–941, 1990.
- L. Jäncke. The relationship between music and language. *Frontiers in Psychology*, 3:123, 2012. doi: 10.3389/fpsyg.2012.001.

- R. Jordan and E. Kafalenos. Listening to Music: Semiotic and Narratological Models. *Musikometrik*, 6:87–115, 1994.
- D. Kartsaklis, I. Fan, R. Yeung, A. Pearson, R. Lorenz, A. Toumi, G. de Felice, K. Meichanetzidis, S. Clark, and B. Coecke. lambeq: An Efficient High-Level Python Library for Quantum NLP. *arXiv preprint arXiv:2110.04236*, 2021.
- S. Koelsch. Toward a neural basis of music perception - a review and updated model. *Frontiers in Psychology*, 2:110, 2011. doi: 10.3389/fpsyg.2011.00110.
- S. Koelsch. Brain correlates of music-evoked emotions. *Nature Reviews Neuroscience*, 15:170–180, 2014. doi: 10.1038/nrn3666.
- J. Lambek. *Type Grammar Revisited*, volume 582 of *Lecture Notes in Computer Science*, pages 1–27. Springer, Heidelberg, 1999.
- F. Lerdhal and R. Jackendoff. *A Generative Theory of Tonal Music*. The MIT Press, Cambridge, MA, 1996.
- R. Lorenz, A. Pearson, K. Meichanetzidis, D. Kartsaklis, and B. Coecke. QNLP in Practice: Running Compositional Models of Meaning on a Quantum Computer, 2021. URL <https://arxiv.org/abs/2102.12846>.
- A. Mann. *The Study of Counterpoint: From Johann Joseph Fux's Gradus Ad Parnassum*. W. W. Norton and Company, New York, 1965.
- K. Meichanetzidis, A. Toumi, G. de Felice, and B. Coecke. Grammar-Aware Question-Answering on Quantum Computers, 2020. URL <https://arxiv.org/abs/2012.03756>.
- K. Meichanetzidis, S. Gogioso, G. de Felice, A. Chiappori, N. and Toumi, and B. Coecke. Quantum Natural Language Processing on Near-Term Quantum Computers. *Electronic Proceedings in Theoretical Computer Science*, 340:213–229, Sep 2021. ISSN 2075-2180. doi: 10.4204/eptcs.340.11. URL <http://dx.doi.org/10.4204/EPTCS.340.11>.
- R. Milovanov and M. Tervaniemi. The interplay between musical and linguistic aptitudes: a review. *Frontiers in Psychology*, 2:321, 2011. doi: 10.3389/fpsyg.2011.00321.
- E. R. Miranda. *Composing Music with Computers*. Focal Press, Oxford, 2001.
- E. R. Miranda. Emergent songs by social robots. *Journal of Experimental and Theoretical Artificial Intelligence*, 20(4):319–334, 2008. doi: 10.1080/09528130701664640.
- E. R. Miranda. *A-Life for Music: Music and Computer Models of Living Systems*. A-R Editions, Middleton, WI, 2011. ISBN 978-0-89579-673-8.
- E. R. Miranda. *Thinking Music: The inner workings of a composer's mind*. University of Plymouth Press, Plymouth, 2014.
- E. R. Miranda. Genetic Music System with Synthetic Biology. *Artificial Life*, 26(3):1–27, 2020. doi: [http://dx.doi.org/10.1162/artl\\_a\\_00325](http://dx.doi.org/10.1162/artl_a_00325).
- E. R. Miranda, editor. *Handbook of Artificial Intelligence for Music Foundations, Advanced Approaches, and Developments for Creativity*. Springer International Publishing, Cham, Switzerland, 2021a. doi: 10.1007/978-3-030-72116-9.

- E. R. Miranda. *Quantum Computer Music Album*. SoundClick, 2021b. URL [www.soundclick.com/LudovicoQuanthoven](http://www.soundclick.com/LudovicoQuanthoven).
- E. R. Miranda and J. A. Biles. *Evolutionary Computer Music*. Springer, London, 2007.
- E. R. Miranda, S. Kirby, and P. Todd. On Computational Models of the Evolution of Music: From the Origins of Musical Taste to the Emergence of Grammars. *Contemporary Music Review*, 22(3): 91–111, 2010. doi: 10.1080/0749446032000150915.
- R. Monelle. *Linguistics and Semiotics in Music*. Harwood Academic Publishers, Chur, Switzerland, 1992.
- J. Moore. The Language of Moss. Online publication, 2019. URL <https://web.archive.org/web/20090813110752/http://www.thelanguageofmoss.com/>. Accessed on 02 April 2021.
- M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, New York, NY, USA, 10th edition, 2011. ISBN 1107002176, 9781107002173.
- R. Orus. A practical introduction to tensor networks: Matrix product states and projected entangled pair states. *Annals of Physics*, 349:117–158, 2014. URL <https://arxiv.org/abs/1306.2164>.
- F. Pachet, P. Roy, and B. Carre. Assistive Music Creation with Flow Machines: Towards New Categories of New. In E. R. Miranda, editor, *Handbook of Artificial Intelligence for Music*, pages 485–520. Springer, London, 2021. ISBN 978-3-030-72115-2.
- A. Patel and E. Morgan. Exploring Cognitive Relations Between Prediction in Language and Music. *Cognitive Science: A Multidisciplinary Journal*, 41(S2):303–320, 2016. doi: 10.1111/cogs.12411.
- A. D. Patel. Why would musical training benefit the neural encoding of speech? The OPERA hypothesis. *Frontiers in Psychology*, 2:142, 2011. doi: 10.3389/fpsyg.2011.00142.
- S. Perchy and G. Sarria. Musical Composition with Stochastic Context-Free Grammars. In *Proceedings of 8th Mexican International Conference on Artificial Intelligence*, 2016. URL <https://hal.inria.fr/hal-01257155>. Accessed on 05 April 2021.
- D. Peterson. *The Art of Language Invention*. Penguin Random House, New York, N, 2015.
- D.J. Rosenkrantz and R.E. Stearns. Properties of deterministic top-down grammars. *Information and Control*, 17(3):226–256, 1970. ISSN 0019-9958. doi: [https://doi.org/10.1016/S0019-9958\(70\)90446-8](https://doi.org/10.1016/S0019-9958(70)90446-8). URL <https://www.sciencedirect.com/science/article/pii/S0019995870904468>.
- S. Sivarajah, S. Dilkes, A. Cowtan, W. Simmons, A. Edgington, and R. Duncan.  $\text{t|ket}\rangle$ : a retargetable compiler for NISQ devices. *Quantum Science and Technology*, 6(1):014003, Nov 2020. ISSN 2058-9565. doi: 10.1088/2058-9565/ab8e92. URL <http://dx.doi.org/10.1088/2058-9565/ab8e92>.
- J.C. Spall. Implementation of the simultaneous perturbation algorithm for stochastic optimization. *IEEE Transactions on Aerospace and Electronic Systems*, 34(3):817–823, 1998. doi: 10.1109/7.705889.
- R. S. Sutor. *Dancing with Qubits: How quantum computing works and how it can change the world*. Packt, Birmingham, UK, 2019. ISBN 978-1-838-82736-6.

- A. Toumi, R. Yeung, and G. de Felice. Diagrammatic Differentiation for Quantum Machine Learning. *Electronic Proceedings in Theoretical Computer Science*, 343:132–144, Sep 2021. ISSN 2075-2180. doi: 10.4204/eptcs.343.7. URL <http://dx.doi.org/10.4204/EPTCS.343.7>.
- W. A. Woods. Transition networks grammars for natural language analysis. *Communications of the ACM*, 13(10), 1970. doi: 10.1145/355598.362773.
- R. Yeung and D. Kartsaklis. A CCG-Based Version of the DisCoCat Framework. In *Proceedings of the 2021 Workshop on Semantic Spaces at the Intersection of NLP, Physics, and Cognitive Science (SemSpace)*, pages 20–31, Groningen, The Netherlands, June 2021. Association for Computational Linguistics. URL <https://aclanthology.org/2021.semSPACE-1.3>.
- M. Yoshikawa, H. Noji, and Y. Matsumoto. A CCG Parsing with a Supertag and Dependency Factored Model. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 277–287. Association for Computational Linguistics, 2017. doi: 10.18653/v1/P17-1026. URL <http://aclweb.org/anthology/P17-1026>.

# A Appendices

## A.1 Context-Free Grammars and Monoidal Category Theory

A context-free grammar (CFG) is typically defined with a set of terminal words  $T$ , non-terminal symbols  $N$ , and a set of production rules of the form  $A \rightarrow \alpha$ , where  $A \in N$  is a non-terminal symbol and  $\alpha \in (N \cup T)^*$  is a string of terminal words and non-terminal characters. Any sentence that can be produced by freely applying the production rules on the starting terminal symbol  $S$  is considered to be a grammatical sentence.

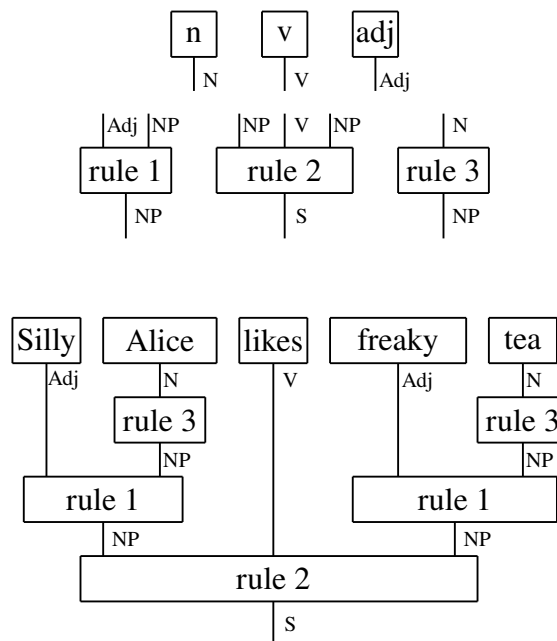
For example, below is a CFG which defines a collection of sentences:

$$\begin{aligned} NP &\rightarrow \text{Adj NP}, S \rightarrow NP V NP, NP \rightarrow N \\ N &\rightarrow n \in \{Alice, tea\} \\ V &\rightarrow v \in \{drinks, likes, hates\} \\ \text{Adj} &\rightarrow adj \in \{funny, silly, freaky\} \end{aligned}$$

The general framework of CFGs can be thought of as a freely generated monoidal category. In diagrammatic notation, a monoidal category consists of boxes with input and output wires. The ‘types’ of the wires are the generating objects of the monoidal category. Wires of matching type can be connected by extending the wire vertically.

In a free monoidal category, any diagram produced by freely composing the boxes is a valid diagram in the category. This resembles how CFGs allow free applications of the production rules.

Production rules in a CFG of the form  $A \rightarrow \alpha$  are represented as a box with multiple input wires representing the string  $\alpha$  and output wire representing the non-terminal  $A$ . Input wires representing terminal words in  $\alpha$  are closed with boxes of type  $I \rightarrow T$ , where  $I$  is an auxiliary type for terminal types. For example, the CFG above can be described by freely composing the boxes in Fig. 27.



**Figure 27:** A diagrammatic representation of the CFG example (top) and a sentence built using it (bottom).



## A.2 Lexicon of Musical Snippets

The lexicon for the music composition system presented in this chapter contains four types of snippets: ground (g), primary (p) secondary (s) and tertiary (t) snippets. Here, snippets are for music what words are for natural languages; languages have different types of words, such as nouns, verbs, and adjectives.

Figure 28 shows two ground snippets, g1 and g2. Snippet g1 is in 3/4 time, marked *mp*, and consists of two staves. The right staff has a melodic line with a slur over the first two measures and a fermata over the last measure. The left staff has a bass line with a slur over the first two measures. Snippet g2 is in 4/4 time, marked *f*, and also consists of two staves. The right staff has a melodic line with a slur over the first two measures and a fermata over the last measure. The left staff has a bass line with a slur over the first two measures.

Figure 28: Ground snippets g1 and g2.

Figure 29 shows nine primary snippets, p1 through p9. Snippets p1, p2, p3, p6, p7, p8, and p9 are in 3/4 time, while p4, p5, and p9 are in 4/4 time. Snippets p1, p2, p3, p6, p7, p8, and p9 are marked *p*, *mf*, and *mp* respectively. Snippets p4, p5, and p9 are marked *f*. Each snippet consists of two staves (treble and bass clef) with various melodic and harmonic patterns.

Figure 29: Primary snippets p1, p2, p3, p4, p5, p6, p7, p8, and p9.

Figure 30 shows four secondary snippets, labeled s1, s2, s3, and s4, arranged in a 2x2 grid. Each snippet is a two-staff piano piece in 4/4 time. Snippet s1 (top-left) begins with a piano (*mp*) dynamic and a triplet of eighth notes in the right hand, followed by a crescendo to a forte (*f*) dynamic. Snippet s2 (top-right) starts with *mp* and ends with *f*. Snippet s3 (bottom-left) starts with *mp*. Snippet s4 (bottom-right) starts with *mp* and ends with *f*. The right hand of each snippet features a melodic line with slurs and ties, while the left hand provides a harmonic accompaniment.

Figure 30: Secondary snippets s1, s2, s3, and s4.

Figure 31 shows three tertiary snippets, labeled t1, t2, and t3. Snippets t1 and t2 are two-staff piano pieces in 4/4 time, both starting with a mezzo-forte (*mf*) dynamic. Snippet t1 (top-left) features a melodic line in the right hand with slurs and ties, and a bass line in the left hand. Snippet t2 (top-right) has a similar structure. Snippet t3 (bottom) is also a two-staff piano piece in 4/4 time, starting with *mf*, and features a more complex rhythmic pattern in the right hand with slurs and ties, and a bass line in the left hand.

Figure 31: Tertiary snippets t1, t2 and t3.

## A.3 Musical Datasets

### A.3.1 Training Data

$\Sigma_\alpha = \{(1, \text{MEL}, [t3, g1, g1]),$   
(2, MEL, [t3, p8, p1, p8, p1, s4, g1]),  
(3, RIT, [t3, p9, p9, p5, p9, s4, g2]),  
(4, RIT, [p9, p4, p4, p4, s3]),  
(5, RIT, [p4, p9, p9, p9, s4]),  
(6, RIT, [p5, p9, p4, p5, s1]),  
(7, RIT, [t3, p9, p5, p7, p7, s1, g2]),  
(8, MEL, [p1, p8, p2, p3, s4]),  
(9, MEL, [p6, p1, p8, p3, s4]),  
(10, MEL, [t2, g1, t1, g1, g1]),  
(11, RIT, [p4, p5, p5, p7, s4]),  
(12, MEL, [t2, t3, t2, g1, g1, g1, g1]),  
(13, MEL, [p8, p1, p6, p2, s4]),  
(14, RIT, [p5, p7, p7, p9, s2]),  
(15, RIT, [p5, p7, p5, p9, s4]),  
(16, RIT, [t3, g2, p9, p9, p7, p7, s1]),  
(17, MEL, [p6, p2, p3, p2, s1]),  
(18, MEL, [t1, g1, p3, p1, p1, p1, s2]),  
(19, RIT, [p7, p7, p5, p7, s1]),  
(20, RIT, [t3, p5, p9, p5, p7, s2, g2]),  
(21, RIT, [p7, p4, p4, p9, s2]),  
(22, RIT, [p5, p9, p5, p4, s1]),  
(23, RIT, [p9, p9, p7, p9, s1]),  
(24, RIT, [p7, p7, p9, p7, s1]),  
(25, RIT, [p9, p4, p5, p7, s4]),  
(26, MEL, [t3, p8, p6, p1, p2, s2, g1]),  
(27, MEL, [t2, p6, p8, p8, p8, s1, g1]),  
(28, MEL, [p6, p3, p6, p6, s2]),  
(29, MEL, [p6, p8, p3, p1, s2]),  
(30, MEL, [p1, p2, p6, p6, s4]),  
(31, MEL, [p6, p1, p2, p8, s3]),  
(32, MEL, [p1, p1, p6, p3, s2]),  
(33, RIT, [p4, p9, p7, p5, s1]),  
(34, RIT, [t2, g2, p5, p4, p5, p4, s1]),  
(35, RIT, [p9, p9, p9, p5, s3]),  
(36, RIT, [p9, p9, p5, p5, s4]),  
(37, MEL, [p6, p2, p6, p2, s1]),  
(38, RIT, [p5, p5, p5, p5, s4]),  
(39, MEL, [p6, p1, p1, p1, s1]),  
(40, RIT, [p5, p7, p5, p4, s2]),  
(41, MEL, [t2, g1, p1, p6, p2, p6, s2]),  
(42, MEL, [p1, p6, p3, p8, s2]),  
(43, RIT, [p4, p5, p7, p7, s1]),  
(44, MEL, [p1, p6, p8, p1, s2]),  
(45, MEL, [p6, p1, p2, p3, s1]),  
(46, RIT, [t2, g2, t1, g2, p9, p5, p4, p9, s2]),  
(47, MEL, [t3, t1, g1, g1, p8, p6, p6, p8, s4]),  
(48, MEL, [p2, p3, p8, p6, s2]),

(49, RIT, [t1, g2, t1, p7, p4, p7, p7, s4, g2]),  
 (50, RIT, [t1, t2, g2, p4, p9, p4, p7, s4, g2])}

### A.3.2 Development Data

$\Sigma_{\beta} = \{(51, \text{MEL}, [t1, g1, p8, p6, p8, p2, s4]),$   
 (52, MEL, [t1, p6, p8, p2, p8, s4, g1]),  
 (53, RIT, [p4, p5, p7, p4, s2]),  
 (54, MEL, [t2, p2, p8, p1, p6, s2, g1]),  
 (55, MEL, [p8, p2, p3, p1, s2]),  
 (56, MEL, [t3, p6, p2, p8, p2, s3, g1]),  
 (57, RIT, [p4, p9, p4, p9, s1]),  
 (58, RIT, [p7, p9, p5, p4, s4]),  
 (59, RIT, [t2, p7, p7, p4, p9, s1, g2]),  
 (60, RIT, [p9, p9, p7, p5, s4]),  
 (61, RIT, [p5, p9, p5, p7, s1]),  
 (62, RIT, [p4, p5, p4, p9, s2]),  
 (63, RIT, [p9, p4, p9, p9, s4]),  
 (64, RIT, [p4, p7, p5, p5, s3]),  
 (65, MEL, [p2, p1, p2, p2, s3]),  
 (66, RIT, [p7, p7, p9, p7, s4]),  
 (67, MEL, [p6, p3, p6, p6, s4]),  
 (68, RIT, [p9, p7, p9, p4, s3]),  
 (69, RIT, [p9, p9, p4, p5, s2]),  
 (70, MEL, [p8, p3, p2, p6, s2]),  
 (71, RIT, [p9, p4, p5, p4, s4]),  
 (72, MEL, [p6, p6, p2, p3, s4]),  
 (73, RIT, [p7, p7, p4, p5, s2]),  
 (74, MEL, [t3, g1, t3, g1, p1, p2, p2, p3, s2]),  
 (75, MEL, [t1, g1, g1])}

### A.3.3 Testing Data

$\Sigma_{\gamma} = \{(76, \text{RIT}, [p9, p9, p9, p9, s1]),$   
 (77, MEL, [p3, p6, p8, p2, s4]),  
 (78, RIT, [p4, p7, p5, p9, s1]),  
 (79, RIT, [p9, p9, p5, p5, s3]),  
 (80, RIT, [p7, p9, p9, p7, s2]),  
 (81, RIT, [t3, g2, p4, p4, p9, p9, s4]),  
 (82, RIT, [p4, p4, p7, p9, s2]),  
 (83, RIT, [p4, p5, p9, p4, s4]),  
 (84, MEL, [t3, p6, p6, p8, p3, s4, g1]),  
 (85, MEL, [p2, p6, p1, p8, s3]),  
 (86, MEL, [p6, p2, p8, p3, s4]),  
 (87, MEL, [p8, p3, p1, p2, s2]),  
 (88, MEL, [t1, g1, t3, g1, g1]),  
 (89, RIT, [p9, p7, p9, p9, s2]),  
 (90, RIT, [p9, p7, p7, p9, s3]),  
 (91, MEL, [t3, g1, p3, p1, p3, p3, s3]),  
 (92, MEL, [p3, p3, p1, p8, s3]),  
 (93, MEL, [t2, p2, p1, p3, p8, s3, g1]),

(94, MEL, [t3, g1, p3, p8, p3, p3, s2]),  
(95, MEL, [p2, p1, p6, p6, s3]),  
(96, RIT, [t1, p5, p9, p4, p9, s3, g2]),  
(97, RIT, [t2, p9, p7, p5, p5, s4, g2]),  
(98, MEL, [t2, p2, p1, p6, p1, s4, g1]),  
(99, RIT, [t3, p9, p5, p9, p9, s1, t3, g2, g2]),  
(100, MEL, [t2, g1, g1])}

# A.4 Compositions

## Bob's Chamomile Slowdown

1

Quanthoven

♩ = 60

*mf* *p* *rall.*

♩ = 74

*mp* *p* *mf*

5 10 15 20 25

3

-- CQ & ICCMR, Aug 2021 --

Figure 32: The composition *Bob's Chamomile Slowdown*.

# Alice's Mushroom Trip

1

Quanthoven

♩ = 60  
*mp*

♩ = 72  
*p*

♩ = 60  
*mf*

♩ = 72  
*p* *rall.* *pp*

-- CQ & ICCMR, Aug 2021 --

Figure 33: The composition *Alice's Mushroom Trip*.

# Bob's Cigar Buzz

1

Quanthoven

♩ = 88

♩ = 100

*p*

*f*

5

*ff*

*f*

10

*ff*

-- CQ & ICCMR, Aug 2021 --

**Figure 34:** First page of the composition *Bob's Cigar Rush*.



2

15  $\text{♩} = 120$   $\text{♩} = 200$

20  $\text{♩} = 100$

25  $\text{♩} = 160$

30

35  $\text{♩} = 96$  *accel.*  $ff$

Figure 35: Second page of the composition *Bob's Cigar Rush*.