Developing Quantum Reservoir Computing as Machine Learning of Music

Eduardo Reck Miranda [1] [2] **and Hari Vignesh Shaji** [1]

[1] Interdisciplinary Centre for Computer Music Research (ICCMR) – University of Plymouth
Drake Circus - Plymouth - PL4 8AA - United Kingdom

[2] Quantinuum
Partnership House - London - SW1P 1BX - United Kingdom

eduardo.miranda@plymouth.ac.uk, hari.shaji@plymouth.ac.uk

Abstract. *This paper[1] introduces a Quantum Reservoir Computing approach to learning time-based sequences of events. It focuses on music as a domain application example. Both the learning process of music and the theory of reservoir computing aligning with each other are explored, validating the suitability and potential of quantum reservoirs in excelling at such tasks. Initial experiments were conducted with a musical piece hinting at the exponential increase in efficiency relative to conventional AI models. The paper is organised as follows: It begins with an introduction to the tasks of high-dimensional analysis and temporal learning, which are required for music. Then, it introduces the basics of Reservoir Computing more generally and discusses how it can address those analysis and learning problems. Next, it shows how concepts from Quantum Computing can be leveraged to harness Reservoir Computing, leading to the concept of Quantum Reservoir Computing. Then, it shows the use of a conventional AI model to learn music in comparison with a classical Reservoir Computing approach. Finally, it presents our approach to using Quantum Reservoir Computing to learn and generate music with an example. The paper concludes with final remarks on the results and challenges for future work.*

1. Introduction

Music engages temporal processing intelligence in the brain, resulting in an elegant experience for the listener. Temporal information processing, an important aspect of how the brain learns, also paves the way for its striking generative capabilities.

Current Artificial Intelligence (AI) systems for music - most of which draw inspiration from the human brain, - are able to learn and generate music deemed to be good imitations of music composed by professional human musicians [1, 2, 3]. However, those systems are significantly different from a physical system like the human brain in terms of their efficiency and underlying dynamics.

We are interested in harnessing the natural dynamics of actual physical systems - as opposed to models inspired thereof) to learn and generate music with mechanisms that are closer to the way in which the human brain works, than the mechanisms found in conventional AI systems. To this end, we are exploring alternative paradigms for computation [4, 5] to develop musical AI systems.

Digital computers are ubiquitous nowadays. They are built around the conventional digital theory that is characterised by symbolic, bi-stable dynamics; i.e., logic states 0 and 1, which has enabled the seamless integration between hardware and software. However, alternative accounts of the very notion of 'computation' can widen its common association with the symbolic nature of anything that happens inside a system that transforms input signals into output signals [6]. These do not need to be restricted to the electrical domain of classic digital computers. Rather, they should include physical systems that exhibit complex spatiotemporal dynamics with signals undergoing some sort of transformations all the time across domains such as biological, chemical, mechanical and indeed atomic and subatomic particles. For instance, the mechanical force from wind exerted onto water, which is a fluid mechanical system, transforms that into water waves.

Planet Earth can be considered a physical system in the way it transforms the incoming electromagnetic waves from the sun and emits back filtered electromagnetic waves some of which contain the visible blue and green light radiations. Similarly, one of the most complex physical systems - the human brain - takes in sensory inputs and transforms them into thoughts, actions and behaviours. This is done very efficiently, as the brain only uses an average of 20W of power. This is enough to power a light bulb [7]. For context, training large language models can easily exceed the average annual consumption of multiple households! The computational efficiency of our brains comes from their complex spatial-thermal dynamics and in-memory computing capability, where

[1] Prepared for presentation at ISQCMC 2023 (2nd International Symposium on Quantum Computing and Musical Creativity), 05-06 October 2023, Berlin.

processing and memory happen concurrently. This is rather different from the well-established von Neumann architecture [8] that is currently adopted to build classic digital computers. Classic computers have a sharp distinction between processing and memory units, which are constrained by the speed of serial communication [8].

This paper proposes the use of a relatively new approach for machine-learning, which harnesses quantum mechanics for unconventional computation, referred to as Quantum Reservoir Computing (QRC) [9].

2. High-Dimensional Analysis

A sequence of musical notes represented by a single feature (e.g., pitch) even though is just a one-dimensional stream of data to a computer, as a musical composition to the listener is multi-dimensional information that contributes to its overall complexity. Deep Learning (DL) is a machine-learning method that uses algorithms called 'neural networks' to learn such high-level patterns given enough data for analysis [11]. The relationships between inputs and targets in the data of interest are often nonlinear and complex. Hence, it is not straightforward to program these with conventional programming, like a 2-dimensional grid of pixel data labelling whether the image is of a cat or a dog. The layers in a neural network project a given input data to the high-dimensional feature - or kernel space - where they become more distinguishable or linearly separable, which helps towards finding the relationships. In this regard, neural networks can be classified as universal approximators [12]; as they can approximate almost any function (e.g., classification, regression) given sufficient data.

The commonly used learning mechanism in AI is known as 'supervised learning'. Vast amounts of labelled data are given to a neural network which then adjusts its internal parameters iteratively depending on how much it is close to the labels and creates an approximation that generalises the overall data. This gives it the ability to predict or generate new outputs when unseen data is presented to the system. The fundamental units of neural networks are called 'neurons', which are biologically inspired synapses that can activate or deactivate to produce an output signal based on the inputs.

Feedforward Neural Networks (FNNs), which are basic architectures in DL, consist of layers to process input and output data through 'hidden layers' in the middle. Each layer has several nodes (or 'neurons') and each connection between those nodes is associated with a weight that is adjusted during a learning process referred to as 'backpropagation'. In a nutshell, backpropagation is the application of gradients in the opposite direction.

For a simple learning task given input x_t and target output y_t, a neural network can be used to find an approximation function f that generalizes the relationship as shown in Equation 1:

$$\hat{y}_t = f(x_t) \quad (1)$$

Here, $\hat{y}_t$ is the predicted output. With every iteration of the learning process, loss function $L(y, \hat{y}_t)$ is calculated which is a measure of the difference between predicted output $\hat{y}_t$ and true or target output y_t. The rate of change of this loss function with respect to the model's parameters determines the gradients that update the parameters in each iteration. Through this training process, the goal is to minimise this loss function such that $\hat{y}_t \approx y_t$.

FNNs are not well-suited for learning time-based sequences such as music, because they are primarily designed to map static input-output relationships without considering any temporal dependencies.

3. Temporal Learning

Temporal learning of sequential data requires memory of the past to process the temporal relations within the sequence. Memory can be realised with recurrence. Recurrence is a key concept in both artificial neural network models and biological neural tissue [10]. A Feedback loop in an arbitrary system is a simple example of recurrence as some information about the past is fed back to the input, influencing the next output. An echo, for example, is essentially a short-term memory of a sound wave that is preserved and reproduced after a certain delay due to reflections with the amplitude of the sound wave starts fading over time. This is very similar to the phenomenon of fading memory that is observed in a type of neural network called Recurrent Neural Networks (RNNs) where information is gradually lost over time [13]. Unlike FNNs, the hidden layers in a recurrent neural network architecture contain feedback loops. Any input fed into the network reverberates for a longer period realising short-term memory. The output at a given time, not only depends on current input but also the fading memory of past inputs interacting with it. Hence, the network will now have a sense of order as feeding the same inputs at different orders will result in a different output. This helps RNNs to distinguish and capture the temporal relationships within a sequence as they are not only spread out in the high-dimensional space but also in time.

Most physical systems possess inherent recurrence which makes RC suitable for temporal tasks [10]. Recurrent networks are prominent in biological neural networks like the human nervous system, enabling the brain to perform in-memory computing by storing and processing information simultaneously.

In DL, RNNs are a popular class of architectures used for temporal learning. The recurrence in the hidden layers holds an internal memory state a_t that is updated at each time step. The output of a single unit of RNNs, referred to as a 'recurrent cell', can be described as shown in Equation 2:

$$\hat{y}_t = f(x_t, a_{t-1}) \quad (2)$$

The predicted output $\hat{y}_t$ at a time t is a function of current input x_t and previous memory state a_{t-1}.

One of the general design criteria for RNNs is to handle variable-length sequences [14]. This can make the model be configured for different applications based on the

input x and target output y vector lengths. A many-to-one RNN model can be used for natural language processing (NLP) tasks. For instance, to identify the emotion of a sentence, the model would take in a sequence of words of variable length and output a single emotion. A one-to-many model could be used, for instance, to caption a single static image by generating a sequence of words describing it.

Music sequences are commonly trained with many-to-many model using a continuation of the sequence as the target output [15]. This helps the model keep track of the dependencies of a note to its previous notes as it learns. To generate novel music from a trained model, it can then be re-configured as a one-to-many model. For example, the model could follow a randomly initialised or user-given note and continue generating notes one after another autonomously.

Considering a simple example where music is represented by a one-dimensional stream of pitches, it can be encoded suitably and split into multiple input and target output pairs: x_n and y_n of fixed length with each pair representing a specific sequence in the stream. Figure 1 shows an example sequence consisting of 5 musical notes, which can be split into input y_n and target output y_n of length 4 where the output is delayed by one note.

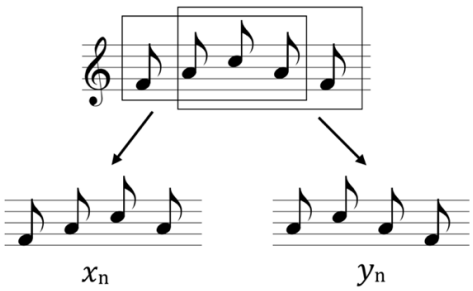

Figure 1: Splitting a sequence into input (left) and output (right) pair.

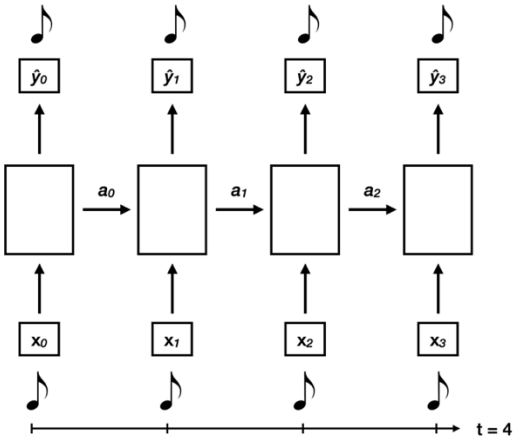

Figure 2: A recurrent cell unrolled in time.

Figure 3 shows a trained model reconfigured as one-to-many model for music generation. Here, given a first note, the first predicted note can be fed back as the input at the second time step and so on to continue the generation.

Multiple such pairs representing different sequences in a music score, can iteratively help the model learn the overall music. Figure 2 shows a recurrent cell operating with input and output pair x_n and y_n, which when unrolled in time shows how it takes input sequentially one element at a time and updates the memory state after each time step.

The fixed sequence length determines its ability to keep track of past dependencies. For instance, at the timestep t = 3, the model learns the probability of producing a note *D* given the memory of past notes *A*, *B* and *C* that appeared before. The longer the sequence length, the more dependencies can be captured with a trade-off in computational power.

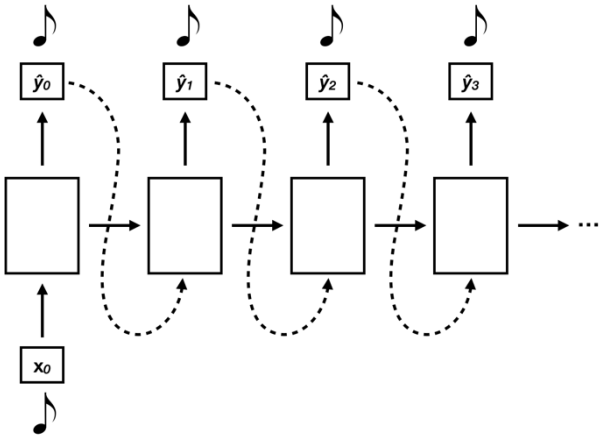

Figure 3: Music generation from a trained model.

Due to their short fading memory, RNNs are not efficient in tracking long-term dependencies that may be required for realistic time-based sequences, such as musical compositions and written texts. The process of backpropagation requires unrolling in time, also called a back propagation through time (BPTT), which causes the gradients to diminish exponentially as they propagate in the opposite direction meaning the parameters in the initial time steps are not effectively updated causing the model to lose context of initial information [13]. BPTT also makes RNNs computationally expensive.

This vanishing gradient problem has been addressed by newer architectures such as gate based RNNs and Transformers. Gate based RNNs allow mechanisms to have control over the flow of information from the past, allowing the model to decide what information to forget and what to retain [16]. A type of such RNN called LSTM (Long Short-Term Memory) networks has been regularly used in the fields for NLP tasks, speech recognition and generative music.

Transformers are alternatives to RNNs, which use a 'self-attention' mechanism [17] to attend to different parts of the input sequence simultaneously allowing it to

effectively capture long-term global dependencies, resulting in faster performance compared to sequential processing of RNNs. Transformers have emerged as a leading architecture for certain NLP tasks and have gained wide popularity after the release of language models such as GPTs (Generative Pre-Trained Transformers).

While those new architectures have surpassed the capabilities of traditional RNNs, they still lack in terms of efficiency which have raised significant concerns regarding their carbon footprint [18].

Reservoir Computing provides a different perspective for temporal tasks, which not only addresses the problems of RNNs but also has the potential for highly efficient learning with good performance.

4. What is Reservoir Computing?

Reservoir Computing (RC) harnesses physical systems to carry out computational tasks. In the general framework of RC, the term 'reservoir' can represent a wide range of systems - either software simulators or physical systems - that satisfy a specific set of constraints, which are detailed in this paper. Some of the physical reservoirs that have been used to experiment for intelligent tasks include a bucket of water, a soft-robotic arm, slime moulds, mechanical wings, and electrochemical systems [10]. As we shall see below, this paper proposes to use atomic and subatomic particles - that is quantum processors - as reservoirs.

RC harnesses the computation capabilities of physical systems for AI application [19]. One of the early implementations of RC is called Echo State Networks (ESN) where the reservoir is implemented with a neural network model [20].

ESNs can be derived from RNNs by introducing an alternate training mechanism where the computationally expensive process of backpropagation through time is essentially bypassed. Instead, in RC the recurrent connections of the network are randomly initialised and fixed such that it simulates a dynamic 'reservoir'. Only the output or readout layer is trained. This is often done by simple linear regression. The same holds for any physical reservoir if it possesses the required dynamics to perform high dimensional, non-linear temporal mapping of inputs. It is important to note that the non-linear dynamics driven by input should be dominant over the internal dynamics of the reservoir, to make the influence of inputs separable in high dimensions and learn them effectively [10]. The high-dimensional reservoir states are the observables that can be directly measured or 'read-out' from the reservoir to train the output layer.

For an input x_t and target output y_t, the reservoir's high dimensional state variables can be defined as a vector $s_t = [s_1, s_2, s_3, \ldots s_N]$ consisting of N observables denoting N dimensions. The state s_t indicates the response of a reservoir at any timestep and can be defined similarly to equation (2) as:

$$s_t = f(x_t, s_{t-1}) \quad (3)$$

where f can represent any non-linear function that maps an input x_t to N-dimensional state s_t with respect to previous state s_{t-1}. A readout function F can then be used to map the states to the output like equation (1) as:

$$\hat{y}_t = F(s_t) \quad (4)$$

where $\hat{y}_t$ and F can be typically obtained by simple linear regression, that minimises the loss function to approximate $\hat{y}_t \approx y_t$.

RC is also capable of multitasking where multiple readout layers can be used with a single reservoir each learning a different target output. This makes it an interesting approach for music modelling as there are multiple features associated with music that need learning simultaneously.

5.1 High-dimensional temporal mapping

In the context of physical reservoirs, the fluid dynamics of waters in a controlled environment can be considered to validate the basic requirements for a reservoir [21]. Consider the input is encoded and fed into the water in the form of mechanical disturbance like dropping stones of variable size at a fixed height. The projections of the water surface at arbitrarily chosen points (or states) are monitored for features such as amplitude, frequency, and phase. Like the feature space of a neural network, the state space of the surface of the water has projected the input encoded with a single feature to multiple features, making it more separable. If multiple encoded inputs are sequentially fed to a bucket of water (i.e., a closed system) at a fixed frequency, the waves that are reflected influence the next input's state, creating a complex state over time with fading memory like RNNs.

In the context of ESN, the input reverberates or spreads through the neural network, influencing the next input similarly. The advantage of ESN over recurrent neural networks is in the reduced design complexity and faster training speeds, as only the output layer is optimised.

5.2 Echo state property and parameters

Along with the discussed properties that are common to both artificial neural networks and reservoirs, another important property to be satisfied by reservoirs, in general, is the echo state property, which states an output at any given time should depend on the current fading memory and not the initial conditions [10]. Essentially the effect of initial conditions should be vanished or 'washed out' gradually, to ensure that the random initialisation does not affect the response of initial time steps. Hence, the RC framework uses a parameter called washout period $T_{washout}$ during which the collected states from the reservoir are not trained. Only the input and output pairs after $t > T_{washout,}$ are considered for training.

A stable reservoir is expected to satisfy the echo state property. This can be validated by measuring the responses of the reservoir with different initial conditions and ensuring convergence. The time it takes for the responses to converge can be a good estimate to define the washout period. The echo state property can be realised by a parameter called spectral radius (ρ) which is a measure of the non-linear transformation of the reservoir [10]. It is

denoted as the largest eigenvalue ρ(W) of the reservoir's internal weight matrix (W). A common condition for stability is to satisfy ρ(W) < 1.

Other considerations used to model an ESN, is the sparse connectivity between reservoir units and leaking rate ε which controls how much of past information to retain or *leak* through each time step.

5. Quantum Reservoir Computing

Quantum Computing gives access to the exponentially expandable state space called Hilbert Space. The large degrees of freedom make it an attractive space for high-dimensional computing tasks like machine learning where high expressive power is desired [22].

Commonly, quantum machine-learning techniques are developed with parameterised circuits, which are comparable to the hidden layers of neural networks that undergo optimization [23]. In these cases, the inputs are generally prepared as initial state and the parameterised circuit is configured with a sequence of gates where the parameters can control the rotation of qubits. As with any supervised learning task, the measurement outcomes for a given input state are compared with the actual output data, to compute a gradient and adjust the parameters of the circuit with each iteration.

In contrast, Quantum Reservoir Computing (QRC) is based on an approach that leaves the quantum system undisturbed through all the iterations and aims to harness the internal evolving effects as a computational resource. Here, a quantum substrate is used as a reservoir whose states in response to a given input are measured and optimized at the output layer. In this context, any naturally occurring noise that near-term devices are prone to can be added to the overall dynamics as an advantage for QRC. In recent years, QRC has been implemented with a few different architectures demonstrating their potential in temporal learning tasks [9, 26, 27, 24]. These architectures are discussed below.

5.1 Quantum states and measurements

The evolution of a pure quantum state ψ from time (*t*-1) to (*t*) is given by unitary time evolution operator U which can be derived from Schrodinger's equation for a closed quantum system as:

$$|\psi_t\rangle = U\,|\psi_{t-1}\rangle \tag{5}$$

where U is generally given as $e^{-iH\tau}$. In Equation 5, H is the Hamiltonian operator (describing the energy of the quantum system) and τ is evolution time.

A more generalized version of Equation (5) with quantum states represented as density matrices ρ rather than wave functions ψ can be derived from the *von Neumann equation*, shown in Equation 6.

$$\rho_t = U\rho_{t-1}U^{\dagger} \tag{6}$$

The density matrix representation ρ applies to realistic environments when a quantum state ψ loses its purity and becomes statistically mixed due to noise or upon measurement. The overall quantum reservoir dynamics subject to time evolution can be generalized with a completely positive trace-preserving (CPTP) map, where T accounts for unitary operations as well as any naturally occurring noise as shown in Equation 7.

$$\rho_t = T\rho_{t-1} \tag{7}$$

The measurement of quantum states after evolution can be performed using Pauli operators [22], which are matrices used to describe both quantum operations and the measurement of states. Thus, for a n-qubit system, 2^n states can be measured on a given computational basis [22].

5.2 Temporal learning review with different QRC architectures

QRC was first introduced in 2017 [9], where researchers simulated the non-linear quantum dynamics of a quantum system for temporal learning tasks and showed that a few qubits exhibit powerful performance comparable to hundreds of nodes in neural network approaches such as ESN and RNN. This is an Analog realisation of the framework where the dynamics arise from interacting spins in a network (or ensemble) of quantum subsystems modelled using a Hamiltonian operator. An example of such Hamiltonian is the widely studied fully connected traverse field Ising model described in Equation 8, as H which essentially encodes the rules for how the quantum subsystems (or qubits), should be intertwined.

$$H = \sum J_{i,j}X_iX_j + h_iZ_i \tag{8}$$

where the tunable coefficients are $J_{i,j}$ representing the inter-qubit interaction strength between qubit pairs *i* and *j*, and *h* denotes the magnetic coupling. *X* and *Z* are Pauli operators acting on qubits *i* and *j*.

In this class of framework, the Hamiltonian parameters are randomly set to model the quantum reservoir, which is then subject to time evolution when injected with an input through one auxiliary qubit. This is analogous to exciting the entire state space of a fluid surface by injecting a single mechanical disturbance onto it. The information from the single auxiliary qubit traverses or spreads through the ensemble of subsystems in the reservoir, evolving it in time.

This class of framework have been recently demonstrated using gate-based IBM Quantum systems as an application for temporal trajectory prediction of mobile wireless networks in comparison with RNN and ESN networks [27]. Here, the input x_t at a time step *t* is encoded as angles of R_Y rotation gates acting on the auxiliary qubit which initializes its state to $\sqrt{1-x_t}|0\rangle + \sqrt{x_t}|1\rangle$. And the simulation of analogue time-evolution described in Equations (5-7) is performed on digital gate-based systems by employing the Suzuki Trotterization method, which effectively discretizes or slices the evolution time into smaller steps by decomposing the unitary operator U into smaller components for approximation [30].

The above-discussed architecture of QRC for temporal learning of a sequence consisting of *k* timesteps can be performed by encoding and feeding the input

through the auxiliary qubit to let the system evolve at every i^{th} timestep $i\tau$ with a total simulation time of $k\tau$. The results extracted from the simulation as reservoir states S_t can then be trained using a readout layer to learn a specific task such as non-linear mapping, prediction, or classification.

A general workflow of such temporal learning architecture equivalent to RNNs is shown in Figure 4.

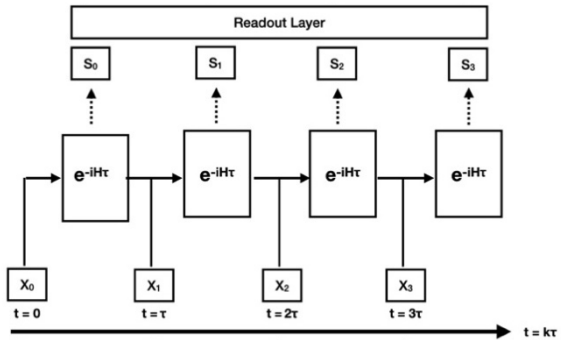

Figure 4: General time evolution architecture.

Due to the increased number of gates required to realise the quantum dynamics in the above class, another class of QRC architecture was developed for digital gate-based implementation on NISQ devices for temporal learning [24]. Here, the reservoir dynamics is modelled digitally using arbitrary parameterized circuits and the input is encoded as probabilities controlling the overall evolution of the quantum system i.e., an input x_t at a time step t is encoded onto a control qubit's state as $(x_t)|0\rangle\langle 0| + (1\text{-}x_t)|1\rangle\langle 1|$. This input-dependent quantum reservoir dynamics can be realised from Equation 7, as follows:

$$\rho_t = T(x_t)\rho_{t-1} \tag{9}$$

Figure 5 shows the quantum circuit implementation of QRC proposed in [24] where $\rho(u)$ and $\rho(\varepsilon)$ are single qubit states, such that the input u controls the unitary evolutions, and leaking rate ε controls the probability of swapping (or resetting) the evolved state with an arbitrary state σ modelling the reservoir's rate of forgetting its initial state.

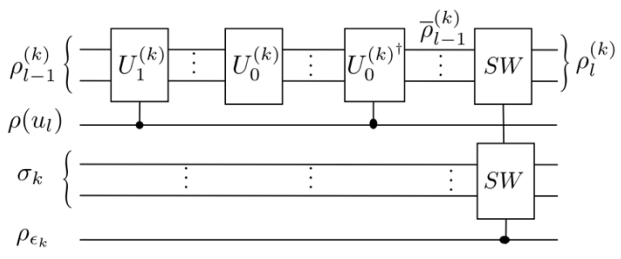

Figure 5: The architecture proposed by Chen *et al.* in [24].

Following the gate-based QRC architecture, [26] has demonstrated an implementation different from the previously discussed ones. Here, the unitary evolutions are not arbitrary but directly parameterized by the input (encoded as angles) to evolve a default initial state of $|0\rangle$. This implementation is specifically designed to study the influence of naturally occurring noise in real quantum hardware. The unitary input-dependent circuit schematics are intentionally made simpler to let the noise (such as decoherence and depolarizing noise) contribute to the reservoir dynamics. Hence, the CPTP map is not modelled like Equation 9 but corresponds to the real quantum device in operation. Figure 6 shows the demonstration of temporal learning tasks performed in [26].

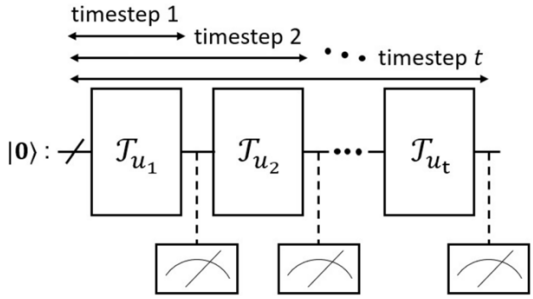

Figure 6: Temporal learning with QRC introduced by Suzuki *et al.* in [26].

5.3 Analysing requirements for sequence modelling

The different implementations of QRC reviewed in Section 5.2 have used 1-D time series to demonstrate proof-of-principle temporal learning tasks such as non-linear mapping and prediction of time series in comparison to classical neural networks [9, 24, 26, 27] and a simple classification task in comparison with linear regression model [26]. There are a few challenges that must be considered when scaling QRC for feasible and efficient learning of complex tasks which are high-dimensional and often data-intensive. The challenges and limitations of implementing music learning are considered in this section which motivates the choice of architecture in Section 5.4 used for the demonstration of music learning in this paper.

5.3.1 Measurement efficiency

With all the implementations of QRC frameworks in NISQ devices discussed above, increasing sequence length extends the circuit horizontally with each time step as clearly indicated in Figure 6. This is because the act of measurement collapses the quantum state (also referred to as 'back action') and the circuit for the next time step must be re-run from the initial time step x_0. This not only increases the circuit complexity but also the time taken to execute and measure many circuits one after another. A typical sequence modelling task requires the following parameters:

- number of iterations for training (N)
- the batch size of each iteration (B)
- sequence length of each batch (L)

With back action in effect, the total circuit measurements required to collect data on reservoir states will be $N \times B \times L$, which will equal several thousands of circuits growing in length.

The restarting measurement protocol can be reinterpreted if the echo state property of a reservoir is considered. By introducing a washout period (W), the measurement restarting time step can now made to be x_W-x_L instead of x_0. This saves the projective measurement of $N \times B \times W$ circuits. Exploiting this property even further, [32] suggests a 'rewinding' protocol where the total execution length can be fixed to W, sliding the restarting time step to $x_{(t-W)} - x_t$ for measurement of a state a time t. Hence, experimentation with different measurement protocols may provide insight into the efficiency of scaling reservoirs in NISQ devices.

5.3.2 Input encoding

The representation of input plays a major role in the performance of many neural network architectures. All the QRC architectures employed for temporal learning (reviewed in Section 5.2) deal with learning of 1-D time series, where the input is encoded through a single auxiliary qubit, control qubit or gate parameter. This will help the network understand the relationship between each input event better (or categorize them better using binary vectors with 'one-hot encoding'). However, this can be considered as a limitation for sequence modelling tasks, as each event at a timestep in the sequence is often encoded as a high-dimensional vector suitable for a neural network can work with.

A powerful technique in NLP called 'word embedding' is an efficient encoding scheme that encodes each event in a vocabulary (i.e., all unique events in the data) into a fixed-length vector of real numbers [33]. This representation can capture the semantic relationship between events. This is very useful for music. A neural network, for instance, can understand the similarity between, say, the words 'cat' and 'dog', by numerical analysis of how close their vector representations are. The same thing can be done with musical notes, for example, with respect to harmonic progressions in a piece of music. Hence, an ideal QRC architecture for advanced tasks such as these should accept a vector representation for each timestep at the input layer for better performance.

The use of vector representation such as word embedding can also pave the way for a technique called 'transfer learning' where the embedding layer can be pre-trained with a different model and re-used, resulting in better efficiency [34]. In NLP, as well as in musical tasks, transfer learning gives a head start. For instance, the words are pre-trained with a larger model consisting of a large vocabulary and can be applied for task-specific models which can be made simpler. Introducing transfer learning to RC can add to the speed and performance of its already efficient learning strategy; this will be shown in Section 6.2.

6. Experiments

6.1 An improved architecture

Based on the requirements and review of existing QRC architecture, we developed a simpler and more efficient one for our experiments. It resembles the architecture of an RNN.

Firstly, we proposed a more general approach to QRC, as shown in Figure 7. This is a more generalized scheme than the ones discussed above. The processing at a single time step has three main elements irrespective of the class of architecture (a) the memory until the previous time step (h_{t-1}), (b) current input (x_t) and (c) current output (y_t).

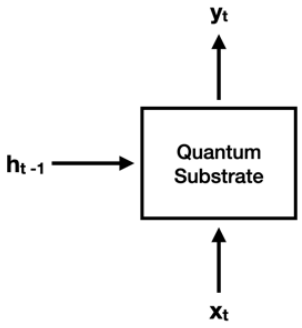

Figure 7: General QRC process at a time step.

Accordingly, we developed an architecture inspired by a recent work on QRC for the study of fluid and thermodynamics [31]. This study was not directly applied to temporal learning, but we found that it is suitable and efficient to implement relative to the architectures in Section 5.2. Here the circuit length is fixed for each timestep but uses a feedback strategy like RNNs, where the memory of previous time steps is fed back to the input externally (Figure 8). The workflow of the architecture has three blocks of unitary evolution. The circuit is initialized with $|0\rangle$ and applied with unitary evolution dependent on previously measured probability amplitudes h_{t-1}, followed by unitary evolution based on current input x_t, followed by arbitrary unitary evolution with a random set of parameters β.

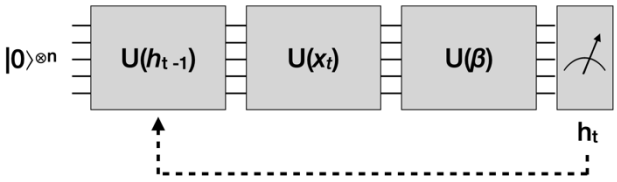

Figure 8: Hybrid classical-quantum RC architecture.

The advantage of our architecture is that each parameter h, x, β can be vectors of different lengths and can be encoded compactly in a fixed number of qubits. Here the length of h is fixed to 2^n states and β can be fixed to n qubits. The input vector at a time step x can be of any length. The circuit schematic proposed in [31] is the same for all these unitary blocks where the encoding is done by

entangling the values together using a circuit schematic of an R_Y gate followed by a CNOT gate as shown in Figure 9. Once the final qubit is reached, the encoding is continued towards top-qubit in zig-zag manner, thus accommodating vectors of any length.

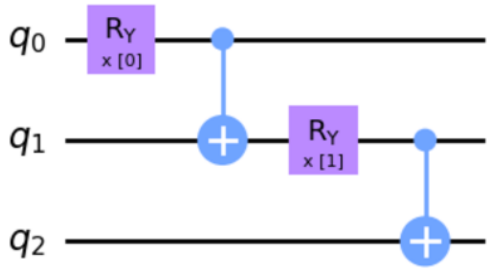

Figure 9: Unitary circuit schematics.

The main advantage with this architecture is that the feedback of measurements bypasses the back action issue and avoids the growing number of gates and circuit complexity of other architectures. Hence, it is efficient to run them for longer sequence lengths, more batch sizes and iterations required for typical sequence modelling tasks.

6.1 Preparation

A monophonic sequence of note from the theme of the soundtrack of the film *Mission Impossible* is used to demonstrate the sequence modelling experiments, firstly with classical neural networks and then with QRC. Figure 10 shows a section of the tune used for training.

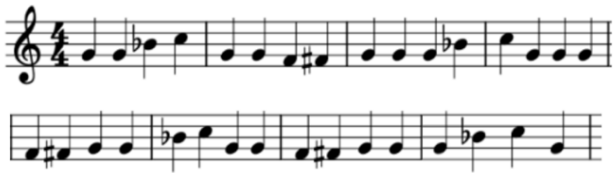

Figure 10: Excerpt from the tune used for training.

The training parameters are set arbitrarily and fixed for all the following models for comparison. No hyper-parameter tuning is performed. A sequence length of 120 is chosen for training the model with a batch size of 16 at each iteration and a total of 150 such batches. A cross-categorical loss function is used that measures the probability distribution of the next note, and learning is done with an Adam optimizer [35] at a rate of 0.005.

6.2 Classical neural network experiment

We demonstrate the music learning capability and the significance of input encoding using classical reservoir computing made of ESN in comparison with an LSTM model using the TensorFlow library [28].

The number of recurrent units in LSTM and ESN are set to 256. The recurrent connections in ESNs are non-trainable and sparse (connectivity set to 0.1) with default spectral radius ρ of 0.9 and leaking rate ε of 1.

First, we demonstrate the learning of ESN to compare the performance with different input embedding dimensions that are randomly initialized. We then fix the embedding dimension to a smaller value of 8 and use pre-trained LSTM input weights to demonstrate the addition of transfer learning. Figure 11 shows the significance of input encoding for a RC model's performance as discussed in Section 5.3.2.

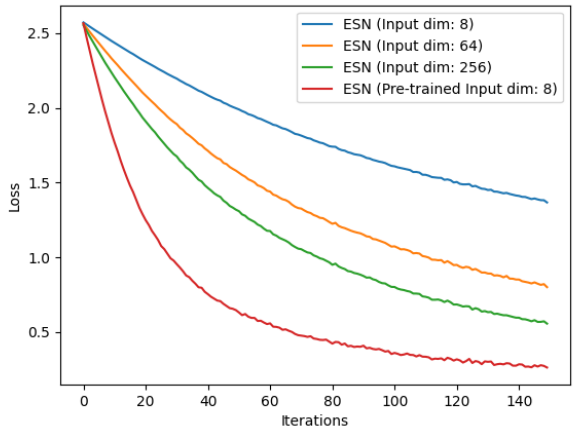

Figure 11: Input encoding with ESN.

It can be observed that increasing the vector size (embedding dimension) accelerates the model's learning to capture the underlying semantics within the input data. Using a pre-learnt embedding, the input sequences represented as vector length of just 8 give the best performance here – which has learnt the music piece well and can generalise new music based on it.

Finally, we compare the performance of ESN and LSTM both with trained input embedding dimension of 64, which makes a total of 3,32,877 trainable parameters for LSTM and 4,173 trainable parameters for ESN corresponding to the output layer. In this setup, Figure 12 shows a stable learning of ESN and a deeper convergence of LSTM. The final loss value of around 0.5 is a good indication of generalization compared to LSTM which is close to 0 and is prone to the problem of over-fitting that may cause the model to memorize the trained music too much making its generalisation capability poor. This can vary according to the dataset.

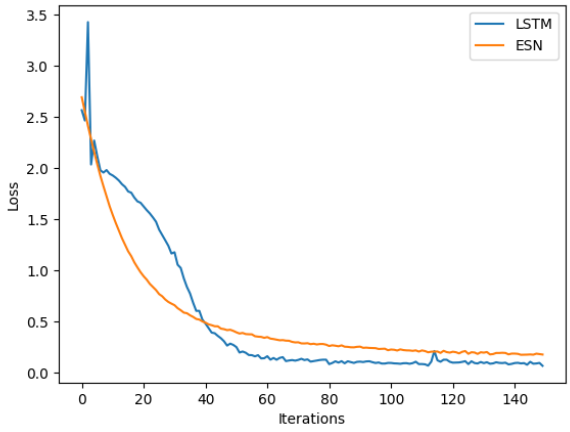

Figure 12: LSTM vs ESN.

Hence, the performance of ESN is on par with LSTM while using only about 1.25% of the trainable parameters of LSTM which makes a big difference in efficiency, especially with the increasing number of recurrent units.

6.3 QRC experiment

We demonstrate a proof-of-principle experiment of learning music with QRC with using Qiskit Aer Simulator with Statevector method as the backend [29]. We choose the simple architecture discussed in Section 5.4 built with 3 qubits and input encoded as the angle of R_Y rotation in the first qubit followed by entanglement. The rest of the specifications are made the same as the previous experiments. In each batch, with a total sequence length of 200, the washout period is set to 80, so the remaining 120 time steps are used for training to match the previous experiments. Training of the collected states was performed with a single linear layer to map the collected states with corresponding target sequences. For a better insight on the nature of the learning curve, the number of training epochs (i.e., a full cycle of training on all batches) is increased as shown in Figure 13.

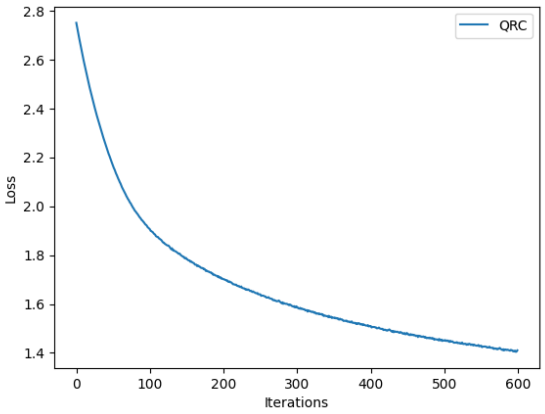

Figure 13: QRC training.

The magnitude of error decrease at the initial stage of training in Figure 13 is a positive sign indicating that the model has learnt some meaningful aspects of the data. It is followed by a slow convergence which reaches a bottleneck indicating room for further improvement to learn the complexities in the data.

Compared to the performance of neural networks in Section 6.2, this is indeed an 'under-fitting' model that generalizes the music with more abstraction than the well-converged classical models. However, the performance can be deemed reasonable for a simulated 3-qubit system with only 117 trainable parameters which is around 2.8% of that of ESN and a mere 0.03% of that of LSTM.

For qualitative analysis, we used the trained readout layer to generate music from the quantum reservoir by feeding in a portion of the original tune long enough to take account of the washout period and getting the 'free-running' or generative response of the reservoir where each predicted note is fed back to the model to predict the next one.

At a glance, it can be noted that the pattern of recurrence of the note *G4* twice or thrice (marked with dotted boxes in Figure 14) is being followed in the generative part, indicating the model's capability to learn simple repetitions from training data. Given that the maximum appeared note in the original training score is *G4*, we repeated the above generative experiment 1000 times (with random initial sequences and generation length same as the original score length) and found that 95% of the times, the note with highest frequency appeared to be G4. This indicates that the model has recognized the significance of a note, which further adds to the evidence of progress in learning. Experimentation with diverse datasets may improve the generalising capability of the model.

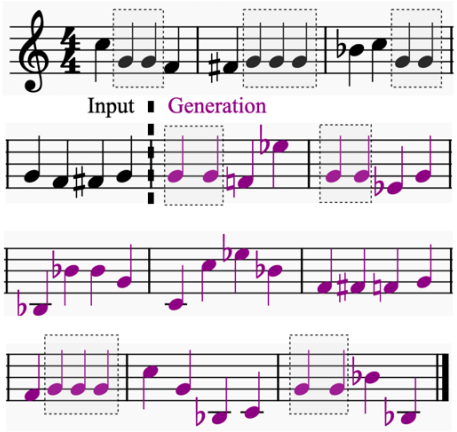

Figure 14: Generated music from QRC. The first 4 bars represent the final portion of the initial sequence.

For a total of 150 batches of size 16, each with a sequence length of 120, a total of 2,88,000 measurements was required, which took under 2 hours on a computer with 8-core CPU and GPU. Due to the efficient architecture and the use of a simulator, the measurement process was feasible in terms of time taken. Thus, paving the way for hyperparameter tuning to find the best configuration of the model. However, it is worth noting that this architecture is not fully quantum as we feed back the measured states externally which is a classical process, making this a hybrid quantum-classical reservoir model [31]. The other fully quantum architectures in Section 5.2 can be experimented efficiently by employing better measurement protocols discussed in Section 5.3.1.

In the experiment conducted above, no input encoding scheme was introduced. As shown in Section 5.3.2, performing similar encoding strategies has the potential to vastly improve the performance of the model.

7. Concluding discussion

We have shown the potential of QRC for learning music more efficiently than conventional AI models. Reducing the scale of trainable parameters while maintaining or improving performance offers several benefits in a resource-intensive generative AI landscape. The initial experiments conducted here suggest the exponential decrease in trainable parameters, going from the LSTM (in a scale of 1,00,000) to ESNs (in the scale of 1,000). This result can be further reduced with the implementation of a physical quantum reservoir, that for a typical music learning task, hints at trainable parameters in the scale of 100 at the output layer. The potential to process high-dimensional musical patterns in the rich Hilbert space

of quantum mechanics opens doors for compelling research and exploration. In future work, we aim to optimize the quantum reservoir model with hyper-parameter tuning and better measurement protocols, to run in real quantum hardware and eventually scale the model to handle complex music tasks and large datasets.

References

1. Cope, D. (1996). *Experiments in Musical Intelligence*. A-R Editions. ISBN-13: 978-0895793379.
2. Miranda, E. R. (Ed.) (2021). *Handbook of Artificial Intelligence for Music*. Springer. ISBN: 978-3030721152.
3. Civit, M., Civit-Masot, J., Cuadrado, F., and Escalona, M. J. (2022). "A systematic review of artificial intelligence-based music generation: Scope, applications, and future trends", *Expert Systems with Applications*. https://doi.org/10.1016/j.eswa.2022.118190
4. Adamatzky, A. (Ed.) (2018). Unconventional Computing. Springer. ISBN 978-1-493968824.
5. Miranda, E. R. (Ed.) (2017). Guide to Unconventional Computing for Music. Springer. ISBN: 978-3319842646.
6. Jaeger, H. (2021). Towards a generalized theory comprising digital, neuromorphic and unconventional computing. *Neuromorphic Computing and Engineering*, 1(1), 012002. https://doi.org/10.1088/2634-4386/abf151
7. Kovác, L. (2010). The 20 W sleep-walkers. EMBO Reports, 11(1), 2. https://doi.org/10.1038/embor.2009.266
8. Von Neumann, J. (1945). First draft of a report on the EDVAC. Moore School of Electrical Engineering, University of Pennsylvania. Retrieved from 10.5479/sil.538961.39088011475779
9. Fujii, K., & Nakajima, K. (2017). Harnessing Disordered-Ensemble Quantum Dynamics for Machine Learning. Physical Review Applied, 8(2). https://doi.org/10.1103/physrevapplied.8.024030
10. Cucchi, M., Abreu, S., Ciccone, G., Brunner, D., & Kleemann, H. (2022). Hands-on reservoir computing: A tutorial for practical implementation. *Neuromorphic Computing and Engineering*, 2(3), 032002. https://doi.org/10.1088/2634-4386/ac7db7
11. Goodfellow, I., Benjio, Y., and Corville,A. (2016). Deep Learning. *The MIT Press*. ISBN: 978-0262035613.
12. Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5), 359–366. https://doi.org/10.1016/0893-6080(89)90020-8
13. https://arxiv.org/abs/1211.5063
14. https://arxiv.org/abs/1409.3215
15. https://arxiv.org/abs/1308.0850
16. Gers, F. A., Schmidhuber, J., & Cummins, F. (2000). Learning to Forget: Continual Prediction with LSTM. *Neural Computation*, 12(10), 2451–2471. https://doi.org/10.1162/089976600300015015
17. https://arxiv.org/abs/1706.03762
18. https://arxiv.org/abs/2211.02001
19. Nakajima, K., & Fischer, I. (Eds.). (2021). Reservoir Computing. *Natural Computing Series*. Singapore: Springer Singapore. https://doi.org/10.1007/978-981-13-1687-6
20. Jaeger, H. (2001). The "echo state" approach to analysing and training recurrent neural networks (GMD Report No. 148). GMD - German National Research Institute for Computer Science.
21. Fernando, C., & Sojakka, S. (2003). Pattern Recognition in a Bucket. In Advances in Artificial Life (Vol. 2801, pp. 588–597). *Lecture Notes in Computer Science*. https://doi.org/10.1007/978-3-540-39432-7_63
22. Nielsen, M. A., & Chuang, I. L. (2010). Quantum Computation and Quantum Information: 10th Anniversary Edition. Cambridge, UK: *Cambridge University Press*. https://doi.org/10.1017/CBO9780511976667
23. Biamonte, J., Wittek, P., Pancotti, N., Rebentrost, P., Wiebe, N., & Lloyd, S. (2017). Quantum machine learning. Nature, 549(7671), 195–202. https://doi.org/10.1038/nature23474
24. Chen, J., Nurdin, H. I., & Yamamoto, N. (2020). Temporal Information Processing on Noisy Quantum Computers. *Physical Review Applied*, 14(2), 024065. https://doi.org/10.1103/PhysRevApplied.14.024065
25. Kutvonen, A., Fujii, K., & Sagawa, T. (2020). Optimizing a quantum reservoir computer for time series prediction. *Scientific Reports*, 10(1), 14687. https://doi.org/10.1038/s41598-020-71673-9
26. Suzuki, Y., Gao, Q., Pradel, K. C., Yasuoka, K., & Yamamoto, N. (2022). Natural quantum reservoir computing for temporal information processing. *Scientific Reports*, 12(1), 1353. https://doi.org/10.1038/s41598-022-05061-w
27. Mlika, Z., Cherkaoui, S., Laprade, J. F., & Corbeil-Letourneau, S. (2023, January 20). User Trajectory Prediction in Mobile Wireless Networks Using Quantum Reservoir Computing. ArXiv preprint arXiv:2301.08796. Retrieved from https://arxiv.org/abs/2301.08796
28. https://arxiv.org/abs/1603.04467
29. IBM Research. (2021). Qiskit: An open-source framework for quantum computing [Computer software]. https://qiskit.org/
30. Berry, D. W., Ahokas, G., Cleve, R., & Sanders, B. C. (2006). Efficient Quantum Algorithms for Simulating Sparse Hamiltonians. *Communications in Mathematical Physics*, 270(2), 359–371. https://doi.org/10.1007/s00220-006-0150-x
31. Pfeffer, P., Heyder, F., & Schumacher, J. (2022). Hybrid quantum-classical reservoir computing of thermal convection flow. *Physical Review Research*, 4(3). https://doi.org/10.1103/physrevresearch.4.033176.
32. Mujal, P., Martínez-Peña, R., Giorgi, G. L., Soriano, M. C., & Zambrini, R. (2023). Time-series quantum reservoir computing with weak and projective measurements. npj Quantum Information, 9(1), 1–10. https://doi.org/10.1038/s41534-023-00682-z.
33. Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013, September 7). Efficient Estimation of Word Representations in Vector Space. ArXiv.org. https://arxiv.org/abs/1301.3781.
34. Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. ArXiv, abs/1810.04805.
35. https://arxiv.org/abs/1412.6980