Quantum reservoir to reconstruct polynomial functions

Leopoldo Villoni

June 2025

1 Introduction to Reservoir Computing Framework

Nowadays, machine learning has become increasingly important in the study of complex systems. However, this progress has also led to the development of increasingly complex neural network architectures, which require high computational resources and may suffer from difficulties in training efficiency, stability, and scalability. One of the main challenges is to deal with the non-linear components of these models. **Reservoir Computing (RC)** offers a potential solution to simplify the construction and training of these models. The core idea behind RC is to decouple the architecture into two main components: the **reservoir** and the **readout**. In this framework, input signals are projected into a high-dimensional computational space through the dynamics of a fixed, non-linear system called reservoir. This reservoir acts as a complex dynamical system and is treated as a black box: input data is injected, the system evolves naturally carrying information about the initial state of the reservoir and the input signal. Part of the reservoir output is used, together with the input signal, to update the reservoir state. This process, known as the *feedback mechanism*, is a key feature of reservoir computing, as it allows the system to retain memory of previous states. This is an essential property when dealing with time-dependent or sequential data, where information from past inputs must influence the current state. Only the readout layer is trained. Its role is to map the reservoir states to the target output, typically using simple linear regression. The main advantage of this paradigm is that the non-linear processing, which is normally the most computationally demanding and difficult to optimize, relies completely on the internal dynamics of the reservoir, which remains untrained. Training is restricted to the linear readout, making the learning task much simpler and more efficient. This overview of the reservoir computing framework is inspired by the exposition provided in Stepney's tutorial [4].

2 Echo-State Network (ESN)

Reservoir computing can be implemented both using standard digital networks and physical dynamical systems. One way to implement it digitally is through an **Echo-State Network (ESN)**. In this approach, the output is generated by an untrained, randomly connected network and extracted by a linear readout layer (see Figure 1). The internal state is updated according to the following rule:

$$\begin{aligned}&x_{\text{out}}(i) = F\left[W_{\text{in}} \cdot x(i) + W \cdot y\right] \cdot L + (1 - L) \cdot y,\\ &y = x_{\text{out}}(i-1).\end{aligned}$$

Here, the parameter L, called **leaky rate**, determines how much "memory" of the previous state is retained. The matrices W_{in} and W are randomly initialized with dimensions Dxn and DxD,respectively, where n is the input dimension and D is the reservoir size. F is the activation function, typically a non-linear function such as `tanh` or ReLu.

For the reservoir to function properly, it must be *stable.* This means that the effect of a given input should fade over time, preventing the internal dynamics from becoming chaotic or dominated by past inputs. This requirement is known as the *echo-state property.* In other words, past inputs should influence the internal state to provide memory, but this influence should decay over time so that new inputs can still affect the system meaningfully. The **spectral radius (sr)** is introduced to guarantee stability by rescaling the weight matrix W as follow

$$W \to W \cdot \frac{\text{sr}}{\lambda_{\text{max}},}$$

where λ_{max} is the largest absolute eigenvalue of W.

All these hyperparameters must be tuned in order to improve the model's performance on the target task. Finally, the reservoir state is passed to the linear readout in order to reconstruct the final output.

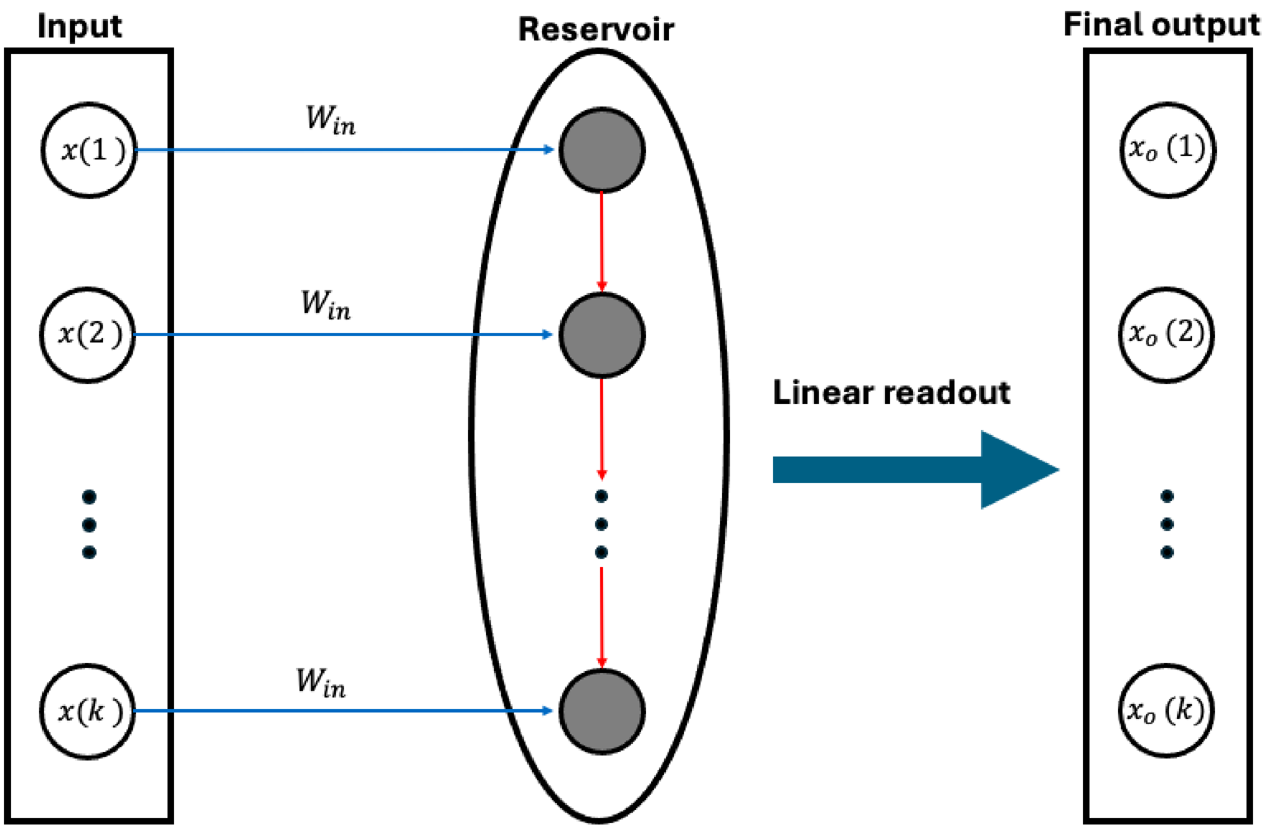


Figure 1: The input sequence $\{x(i)\}$ is fed into a fixed, randomly connected reservoir via input weights W_{in}. The reservoir processes the input through its internal dynamics, where recurrent connections (indicated by red arrows) implement a feedback mechanism that allows the system to retain memory of previous states. A trained linear readout maps the reservoir states to the final output $\{x_o(i)\}$.

3 Physical Implementation of Reservoir Computing

Let us now explore how the physical implementation of reservoir computing works, and how it can offer advantages over its digital counterpart. In reservoir computing, the role of the reservoir is to introduce non-linearity into the system—something that, in standard neural network architectures, is achieved using complex activation functions that often require significant computational resources. Digital implementations such as Echo State Networks (ESNs) rely on random weights to extract relevant features from the data. However, for highly complex tasks, this approach can be not enough unless the reservoir size is significantly increased- this behavior will be discussed in more detail later- potentially leading to instability or increased computational costs. The idea behind physical reservoirs is to exploit the intrinsic non-linearity of a selected physical system to perform feature extraction. The input data is encoded by mapping it into the initial state of the system. This state then evolves naturally according to the system's physical dynamics. The resulting final state is used as the output of the reservoir and serves as the input to the readout layer-for example, a simple linear regression, which can be implemented digitally with a small computational effort. The goal is therefore to identify a suitable physical system that is capable of:

- encoding the input data by appropriately setting parameters of the device that contains the system;
- providing measurable output signals to feed into the readout layer;
- introducing sufficient non-linearity to solve the task effectively.

In this work, we explore a quantum implementation of reservoir computing using an **integrated photonic circuit**

(IPC) injected with two-photon states. In this case, the dynamical system is represented by the propagation of photon pairs injected through the input ports of the circuit. The IPC is highly reconfigurable, allowing the input to be encoded by tuning specific physical parameters—such as internal phases—within the device. Since photon propagation depends on these physical parameters, the resulting output probabilities at the circuit's exit ports are influenced by the input and can be used as the reservoir output. Moreover, part of the output is used to dynamically update internal parameters of the circuit, thus implementing a feedback mechanism.
Furthermore, the system offers an additional configuration option: the photon pairs injected into the circuit can be either distinguishable or indistinguishable, depending on their relative optical path length. The significance of this feature will be discussed in detail later.

4 Structure and Goals of This Experiment

The aim of this study is to investigated whether a physical quantum reservoir, implemented using an **integrated photonic circuit (IPC)**, can perform the reconstruction of polynomial functions. In particular, we aim to understand whether a purely quantum effect such as photon **indistinguishability** can influence the performance of a classical learning task by evaluating its effect on the reservoir's ability to perform such reconstructions. Before addressing the quantum reservoir implementation, we first built and trained a digital **ESN** model to study how its dimensionality affects the reconstruction of monomial functions of different orders. In fact, the reservoir size D defines the dimensionality of the weight matrices and, consequently, the dimension of the abstract space in which the input is projected. A higher-dimensional reservoir enables the extraction of more features from the input. Therefore, we expect that more complex tasks—such as the reconstruction of highly non-linear polynomial functions—require a larger reservoir size. Following the study of the ESN, the next step was to investigate the behavior of a quantum reservoir, which was first simulated digitally to analyze its capability over the target task and to optimize the measurement parameters. Finally, we conducted the experiment using real data to reconstruct a polynomial function with the photonic reservoir. We compare the results obtained in the two-photons scenarios, highlighting the role of photon indistinguishability in the computational process.

4.1 Dataset Preparation and Performance Evaluation

The dataset is prepared in the same way throughout all parts of the experiment. We first generate a set $\{x\}$ of $k = 150$ values, equally spaced in the interval $[0, 1]$. The corresponding target set is constructed using polynomial target functions, which vary between different sections of the experiment. The initial 80% of the values is used as the training set, while the remaining 20% forms the test set. The input set $\{x\}$ is fed into the reservoir, and the output is processed by a *Ridge regression model* to reconstruct the target function. This model is implemented using the `scikit-learn` Python library. It solves a regularized linear least-squares problem, minimizing the following loss function:

$$\|y - Xw\|_2^2 + \alpha\|w\|_2^2,$$

where X is the matrix of input data, y is the target vector, and w is the vector of model weights to be learned. The term $\|w\|_2^2$ penalizes large weights, encouraging them to remain small. This reduces the variance of the model and improves generalization. The hyperparameter $\alpha > 0$ controls the strength of the regularization. Regularization is essential to prevent overfitting.
The model's performance is evaluated using the **Mean Squared Error (MSE)** metric computed over the test set:

$$\text{MSE} = \frac{1}{k}\sum_i (y_i - \hat{y}_i)^2,$$

where y_i are the target values and $\hat{y}_i$ are the predicted values returned by the Ridge model and $k = 150$.

5 Experimental Set-up

In this section, we describe the experimental set-up used to implement the quantum reservoir. We start with a laser operating at a wavelength of $\lambda_0 = 785$nm. A second harmonic generation (SHG) crystal is then used to halve the wavelength, producing light at $\lambda_{\text{UV}} = 392.5\,\text{nm}$. This crystal is made of a non-linear optical material which, when pumped with a laser, generates light at twice the frequency of the incoming beam.

5.1 Spontaneous Parametric Down-Conversion

5.1.1 Theory of SPDC

Subsequently, the ultraviolet beam is used to generate **Spontaneous Parametric Down-Conversion (SPDC)**. A comprehensive derivation and discussion of this process can be found in Gerry and Knight [1]. Let us regard a non-linear crystal with a second-order susceptibility $\chi^{(2)}$, pumped with a laser beam. The interaction Hamiltonian for the process takes the form:

$$\hat{H}_I \sim \chi^{(2)} a_p a_s^\dagger a_i^\dagger + \text{c.c.},$$

where a_p is the annihilation operator for the pump beam, $a_s^\dagger$ and $a_i^\dagger$ are the creation operators for the "signal" and "idler" beams, respectively (these labels are conventional and somewhat arbitrary.). Starting from a single pump beam photon-typically in the ultraviolet-, and vacuum states for both signal and idler, the interaction yields this state:

$$|1\rangle_p |0\rangle_s |0\rangle_i \to a_p a_s^\dagger a_i^\dagger |1\rangle_p |0\rangle_s |0\rangle_i = |0\rangle_p |1\rangle_s |1\rangle_i ,$$

where the photons produced in the signal and idler modes are generated simultaneously. Conservation of energy and momentum requires:

$$\hbar\omega_p = \hbar\omega_s + \hbar\omega_i,$$
$$\hbar\boldsymbol{k_p} = \hbar\boldsymbol{k_s} + \hbar\boldsymbol{k_i}.$$

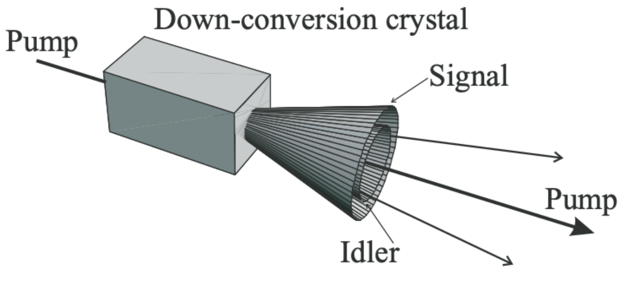
where $\omega_{p,s,i}$ and $\boldsymbol{k}_{p,s,i}$ are the frequencies and wave vectors of the pump, signal, and idler beams, respectively. These equations are known as *phase-matching* conditions. There are two kind of SPDC processes. In *type I* (see Figure 2a), both the signal and the idler photons have the same polarization which is orthogonal to that of the pump. The interaction Hamiltonian for this process is the following:

$$\hat{H}_I = \hbar\eta\ a_p a_s^\dagger a_i^\dagger + \text{c.c.},$$

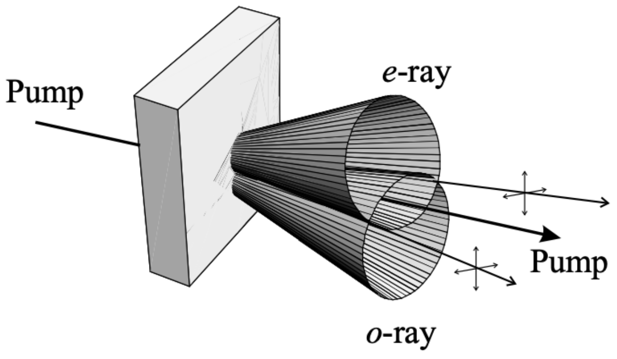
where we defined $\eta \propto \chi^{(2)}\mathcal{E}_p$, with $\mathcal{E}_p$ being the pump field amplitude. Momentum conservation in type I SPDC leads to photons emission along opposite sides of concentric cones centered on the pump beam direction. Hence, photons can be emitted along infinitely many directions on the cones, with varying the wavelengths depending on the phase-matching conditions.

In *type II SPDC* (see Figure 2b), the signal and idler photons have orthogonal polarizations. The two photons are emitted along different cones,still symmetric with respect to the pump direction in order to preserve the momentum. One photon belongs to the ordinary wave (perpendicular to the optical axis), the other to the extraordinary wave (parallel to the optic axis). This type of SPDC allows the generation of entangled states. One method to extract entangled photon pairs is to select only those emitted along the intersection of the two cones, using pinholes. We denote vertical and horizontal polarization states as $|V\rangle$ and $|H\rangle$, respectively. The Hamiltonian for this process, after momentum post-selection, becomes:

$$\hat{H}_I = \hbar\eta \left(a_{Hs}^\dagger a_{Vi}^\dagger + a_{Vs}^\dagger a_{Hi}^\dagger \right) + \text{c.c.}.$$



(a) SPDC Type I — The UV pump beam generates signal and idler photons with identical polarization (orthogonal to the pump), emitted along concentric cones. Different emission angles correspond to different wavelengths, ranging from orange to deep red.



(b) SPDC Type II — Signal and idler photons have orthogonal polarizations and are emitted along two intersecting cones due to birefringence, corresponding to the ordinary and extraordinary rays.

Figure 2: Comparison between Type I and Type II SPDC processes. Images taken from Gerry and Knight [1].

5.1.2 SPDC in the experiment

In our experiment, photon pairs are generated via type II SPDC. This process is implemented by pumping a β-Barium Borate (BBO) crystal with an ultraviolet laser beam at $\lambda_{\mathrm{UV}} = 392.5\,\mathrm{nm}$. Each photon in the pair has orthogonal polarization.

To make the two photons indistinguishable, one of them passes through a half-wave plate (HWP) oriented at 45° with respect to the optical axis, while the other passes through a HWP aligned at 0°. The former rotates the linear polarization by 90°, while the latter leaves the linear polarization unchanged. A bandpass filter (BPF) with a $3nm$ bandwidth is placed after the HWP to ensure spectral indistinguishability. Then we use a polarized beam splitter to transmit only the selected polarization, eliminating background noise.

5.2 Intergrated Photon Circuit

The photon pairs are then collected using optical couplers inside single mode fiber (SMF). One photon is sent directly to an input port of an **integrated photonic circuit (IPC)**, while the other is routed through a piezoelectric delay line that allows precise control of its optical path length. This enables us to control the distinguishability between the two photons by measuring the **visibility** V. The distinguishability is tuned by exploiting the Hong–Ou–Mandel (HOM) effect: by changing the relative delay between the two photons and fitting the resulting coincidence curve, we identify the position corresponding to the indistinguishability condition. This procedure is detailed in a later section. The photons are injected into input ports 1 and 4 of the photonic circuit. They propagate through the waveguides and can exit from any of the available output ports. Inside the circuit, the waveguides get closer to each other at eight different points, known as directional couplers; in these regions, quantum tunneling allows them to behave as a symmetric beam splitter (50:50 BS) enabling quantum interference between photons entering different paths. The circuit includes thermo-optic phase shifters-denoted as R_x in the figure 3-, which allow to control the photons phases by applying an external voltage due to a thermo-optic effect. For this experiment we adjust only the phases ϕ_B,ϕ_4 and ϕ_D, the other phases are kept fixed.

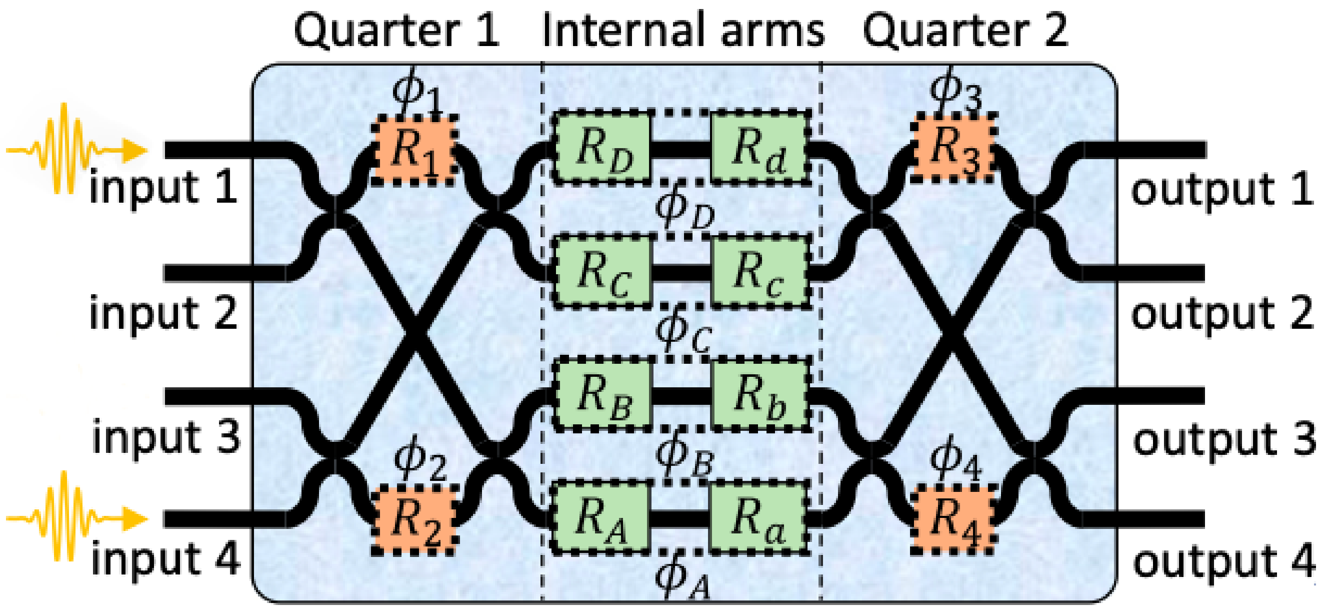


Figure 3: Photonic circuit used to implement the quantum reservoir.

After propagating through the circuit, the photons may exit from any of the four output ports, either both from the same port or from different ports. This results in a total of 10 possible two-photon output configurations. Since the output probabilities depend on the internal phases, data can be encoded by assigning a specific phase value to each element of the input set. In this experiment, the phase ϕ_B is used for input encoding. For each input value, we collect statistics over the output events. The resulting frequencies represent the reservoir output; part of this output is then used to set the other two phases, ϕ_D and ϕ_4, thereby enabling the feedback mechanism. Each output port of the photonic circuit is connected to a symmetric (50:50) fiber beam splitter (FBS), which allow to discriminate between one-photon and two-photon events. Finally, each output channel of the four beam splitters is connected to an avalanche photodiode detector (APD) for single-photon counting.

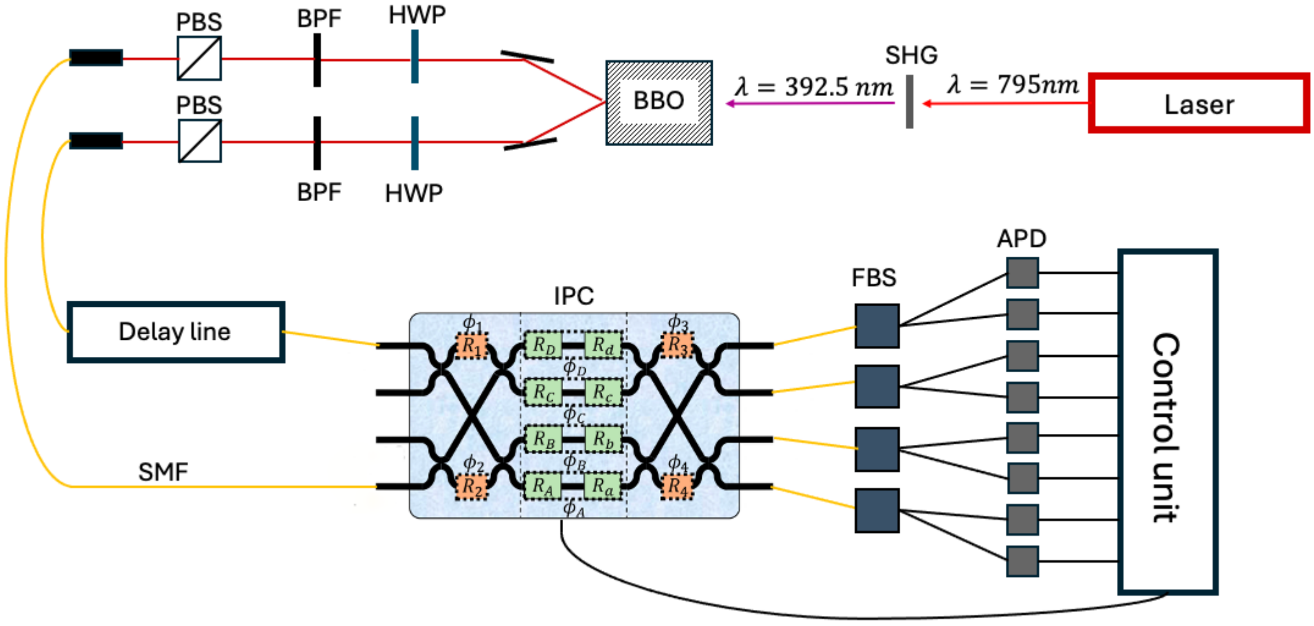


Figure 4: Schematic of the experimental setup. (i) Red light from the laser passes through the second harmonic generation (SHG) crystal. (ii) Photon pairs are generated via spontaneous parametric down-conversion (SPDC) in the BBO crystal. Their polarizations are then matched using half-wave plates (HWPs). A bandpass filter (BPF) selects a narrow frequency range, and a polarizing beam splitter (PBS) removes noise. (iii) The photons are coupled into single-mode fibers (SMFs): one is sent through a piezoelectric delay line, the other directly into the integrated photonic circuit (IPC). (iv) The IPC output ports are connected to fiber beam splitters (FBSs) and then to avalanche photodiodes (APDs). A control unit collects the photon counts and manages the experimental apparatus.

5.3 Visibility control and HOM effect

One of the goals of the experiment is to evaluate the effects of photon indistinguishability on a classical learning task. We define **visibility (V)** as the quantity that quantifies how indistinguishable the two photons are. The **Hong-Ou-Mandel (HOM)** effect is used to experimentally measure this quantity. Classically, sending two single photons into input ports 1 and 2 of a symmetric beam splitter (50:50 BS), leads to three possible outcomes: both photons exit from port 3, both from port 4, or one from each port. However, quantum mechanically, when the two photons are indistinguishable, two-photon interference suppresses the outcome where the photons exit from different ports. As a result, only the bunching events (both photons exiting the same port) are observed. This phenomenon can be described mathematically representing the BS with a transformation matrix, which relates the creation operators $(\hat{a}_1^\dagger, \hat{a}_2^\dagger)$ at the input ports to the creation operators $(\hat{a}_2^\dagger, \hat{a}_4^\dagger)$ at the output ports. For a symmetric 50:50 beam splitter, the transformation is:

$$\hat{a}_1^\dagger = \frac{1}{\sqrt{2}}(i\hat{a}_3^\dagger + \hat{a}_4^\dagger)$$
$$\hat{a}_2^\dagger = \frac{1}{\sqrt{2}}(\hat{a}_3^\dagger + i\hat{a}_4^\dagger)$$

We inject two identical photons into input ports 1 and 2. The initial state is

$$|11\rangle_{in} = \hat{a}_1^\dagger \hat{a}_2^\dagger |00\rangle_{in}\,.$$

Our goal is to map this input state to the corresponding output state, which belongs to a different Hilbert space. Since injecting no photons into the beam splitter leads to no photons at the output, we assume the vacuum state at the input maps to the vacuum at the output: $|00\rangle_{in} = |00\rangle_{out}$ Applying the beam splitter transformation to the input creation operators, we obtain:

$$\hat{a}_1^\dagger \hat{a}_2^\dagger |00\rangle_{in} \to \frac{1}{2}(i\hat{a}_3^\dagger + \hat{a}_4^\dagger)(\hat{a}_3^\dagger + i\hat{a}_4^\dagger)|00\rangle_{out} = \frac{i}{\sqrt{2}}(|20\rangle_{out} + |02\rangle_{out}).$$

Where observe that the term $|11\rangle_{out}$, corresponding to one photon in each output port, is absent. Thus, when changing the relative optical path between the two photons and measuring coincidences at the two outputs, we expect a dip in the coincidence rate when the photons become indistinguishable. In practice, however, in real experiments we deal with photon wave packets rather than ideal Fock states. As a result, this transition is smooth. When plotting the coincidence count as a function of the relative delay, we observe a dip—known as the **HOM dip**—at the point of maximum indistinguishability. The example above describes a simple two-mode interferometer with a single beam splitter. For more detail on the Hong-Ou-Mandel effect and the theoretical treatment of beam splitters, see for example Gerry and Knight [2]. In our case, the situation is more complex: the interferometer is implemented by the entire integrated photonic circuit. However, the underlying principle remains the same. The circuit contains two **quarters**—components functionally analogous to beam splitters with four spatial modes—and a network of internal arms. We can assign transformation matrices to each part of the circuit: **quarter 1**, the **internal arms**, and **quarter 2**.

$$U_{Q1} = \frac{1}{2}\begin{pmatrix} e^{i\phi_2} & ie^{i\phi_2} & i & -1 \\ ie^{i\phi_2} & -e^{i\phi_2} & 1 & i \\ i & 1 & -e^{i\phi_1} & ie^{i\phi_1} \\ -1 & i & ie^{i\phi_1} & e^{i\phi_1} \end{pmatrix} \quad U = \begin{pmatrix} e^{i\phi_D} & 0 & 0 & 0 \\ 0 & e^{i\phi_C} & 0 & 0 \\ 0 & 0 & e^{i\phi_B} & 0 \\ 0 & 0 & 0 & e^{i\phi_A} \end{pmatrix} \quad U_{Q2} = \frac{1}{2}\begin{pmatrix} e^{i\phi_4} & ie^{i\phi_4} & i & -1 \\ ie^{i\phi_4} & -e^{i\phi_4} & 1 & i \\ i & 1 & -e^{i\phi_3} & ie^{i\phi_3} \\ -1 & i & ie^{i\phi_3} & e^{i\phi_3} \end{pmatrix}$$

The matrices shown above are reported as given in [3]. The overall evolution of the photonic circuit is described by the unitary matrix $M = U_{Q2} \cdot U \cdot U_{Q1}$. Given an initial state, such as $|1001\rangle$(one photon in port 1 and one in port 4)-, the final state can be evaluated using this matrix. Although the final state depends on the specific values of the phases, quantum interference is expected to suppress certain output configurations regardless of those values. That is, while the exact set of suppressed configurations may vary with the phases, the presence of destructive interference remains a general feature of the indistinguishable case.

To observe this effect, we measure coincidence counts between the detectors at output ports of the external beam splitters-located after the circuit-, while continuously varying the relative optical path using a piezoelectric delay stage. During this measurement, all phases are kept fixed. When the two photons become indistinguishable, the interference leads to a dip in the coincidence counts between certain output ports.
The **visibility** is defined as:

$$V = \frac{C_M - C_m}{C_M},$$

where C_M and C_m represent the maximum and minimum coincidence counts, respectively—i.e., the counts far from and at the center of the dip. This visibility quantifies the degree of photon indistinguishability, ranging from $V = 0$ (fully distinguishable) to $V = 1$ (perfectly indistinguishable).
Now we have to find the position of the stage to obtain the indistinguishability. Chosen a plot showing the dip, we fit the experimental coincidence data assuming Gaussian temporal profiles for the photon wave packets. This allows us to extract the position of maximum indistinguishability, corresponding to the minimum of the HOM dip.
To determine the optical path corresponding to maximum indistinguishability, we fit the experimental coincidence data using a Gaussian dip model, assuming the photon wave packets have Gaussian temporal profiles. The data are fitted as a function of the position of the piezoelectric stage. The fitting function is:

$$f(x) = A\left(1 - Ve^{-\frac{(x-x_0)^2}{2\sigma^2}}\right),$$

where

- x_0 is the position corresponding to maximum indistinguishability (minimum of the dip);
- σ is the width of the dip;
- $A = f(\pm\infty)$ represents the coincidence count outside the dip C_M;
- V is the visibility, extracted directly from the fit.

Note that the visibility parameter V in the fit matches its experimental definition:

$$V = \frac{f(\pm\infty) - f(x_0)}{f(\pm\infty)} = \frac{C_M - C_m}{C_M}.$$

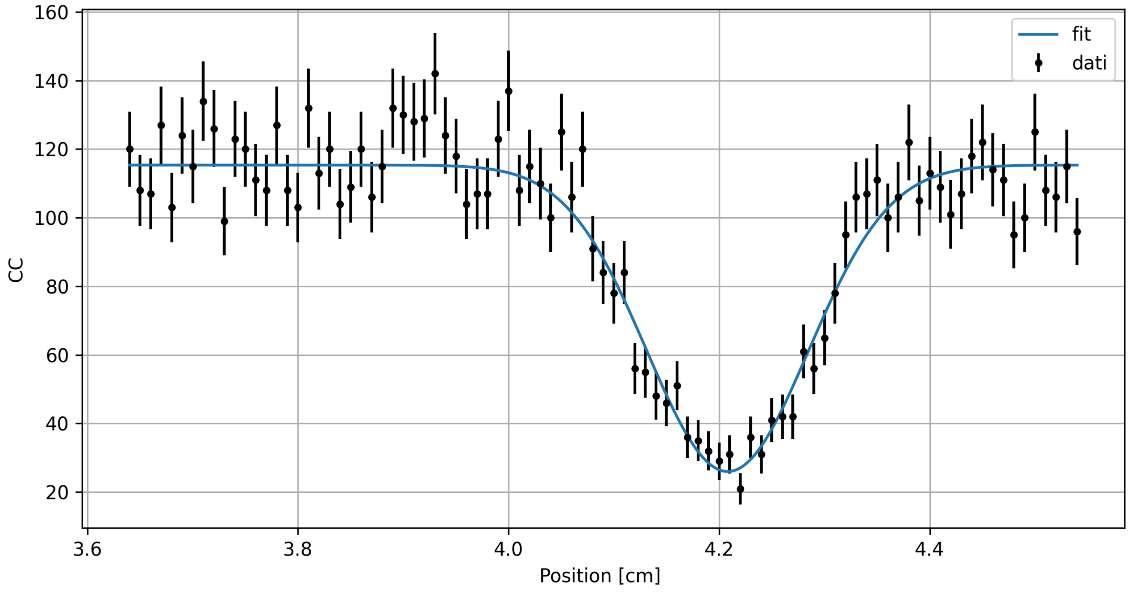


Figure 5: Fit of the HOM dip used to estimate the optimal position for maximum indistinguishability.

Parameter	**Value**
A	115.4 (1.5)
x_0 [cm]	4.208 (0.003)
σ [cm]	0.077 (0.003)
V	0.77 (0.02)

Table 1: Fitted parameters for the HOM interference curve.

6 Digital Reservoir Implementation

We implemented a digital reservoir using an **Echo State Network (ESN)** model to study how its dimensionality affects the reconstruction of monomial functions of different orders. We first generated the input set $\{x\}$ following the procedure described in Section 4.1. The corresponding target set was built using monomial functions of the form $f(x) = x^N$ for several values of N in the interval $[3, 20]$. The input set $\{x\}$ was fed into the reservoir, whose output is read by a *Ridge regression model* to reconstruct the target function. For each combination of N and the reservoir size D we performed an optimization on the following hyperparameters: spectral radius, leaky rate, activation function, and the regularization parameter (α) of the Ridge regression. Optimization was carried out using the *Optuna* library, which begins with a random sampling of the hyperparameter space and iteratively improves the choice based on performance. To reduce the risk of suboptimal convergence due to unlucky choice of initial guesses-potentially causing the research of parameters to get stuck in local minima—each optimization process was repeated $M = 20$ times. The random seed for initializing the ESN weight matrices was kept fixed, so that the only stochastic element in each run was the initial hyperparameter configuration. Model performance was evaluated using the **Mean Squared Error (MSE)** metric. As the value of N increases, the target function becomes more non-linear, and therefore more difficult to reconstruct. So we expect that higher-order polynomials require a more complex model—specifically, a reservoir with larger dimensionality. This trend is confirmed by Figure 6, which show that higher values of D yield better reconstruction performance. However, increasing the reservoir size D comes at a cost: it leads to a higher computational effort.

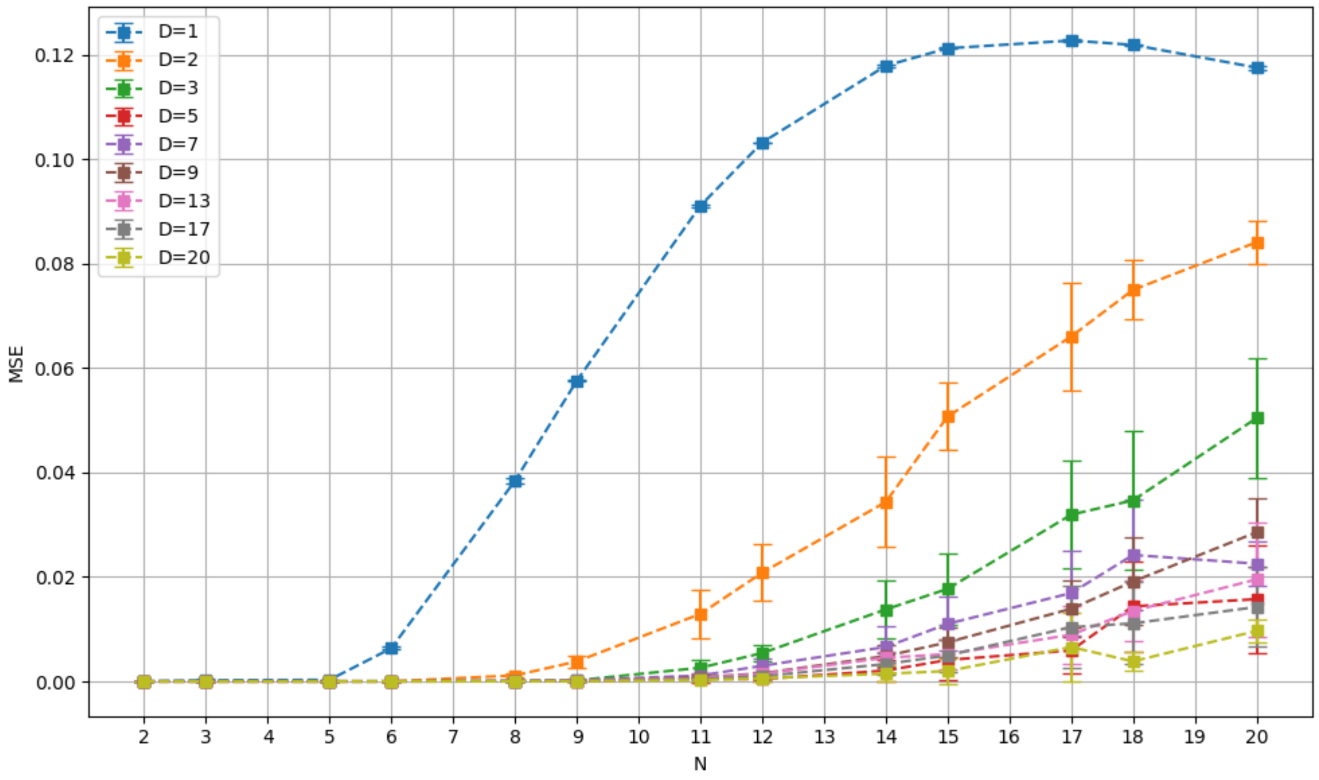


Figure 6: MSE vs N, for different of reservoir size D.

7 Quantum Reservoir Implementation

We implemented the quantum reservoir using the **integrated photonic circuit (IPC)** shown in Figure 3. The goal is twofold: first, to reconstruct functions of the form $f(x) = x^N$ for various values of N; and second, to reconstruct a more complex polynomial function. The experiment is carried out in two different scenarios: **distinguishable photons** ($V = 0$) and **indistinguishable photons** ($V = 1$). These two configurations are obtained by adjusting the piezoelectric stage: the indistinguishable scenario is reached by positioning the stage at the location of the HOM dip found in Section 5.3, while the distinguishable case is obtained by moving the stage sufficiently far from that point. The aim is to investigate whether a purely quantum feature as the photon indistinguishability can unfluence the performance of a classical learning task.
The input data is encoded into the IPC through phase modulation; specifically, we use the phase ϕ_B to encode the input values. A scaling factor is required to map each input value x_i to a corresponding phase $\phi_B(x_i)$. Photon pairs are injected in the input ports 1 and 4 of the IPC. Since the output probabilities depend on the internal phases, each input value corresponds to a different set of output probabilities. These probabilities are experimentally estimated from the relative frequencies of the observed two-photon outcomes, obtained by counting coincidences at the output ports. Coincidence detection is important as it allows to eliminate background noise. In fact, every time a photon count is recorded in one channel, we expect a simultaneous coincidence in another channel due to the other photon of the pair. A coincidence is defined as two detections occurring within a specific time window. If the window is too short, genuine coincidences may be missed; if it is too long, accidental coincidences (noise) may be erroneously counted.

The resulting outcome probabilities are then passed to a *Ridge regression model*, which reconstructs the target function. Since the task involves reconstructing the polynomial functions over a sequence of equally spaced values in the interval $[0, 1]$, and these values are fed into the system sequentially, we must incorporate a feedback mechanism—as done in the digital ESN—to retain memory of past inputs. This is achieved by dynamically setting the phases ϕ_4 and ϕ_D based on two selected output probabilities from the previous step. Two leaky rate parameters are introduced to control how much memory of the previous state is retained in the system.

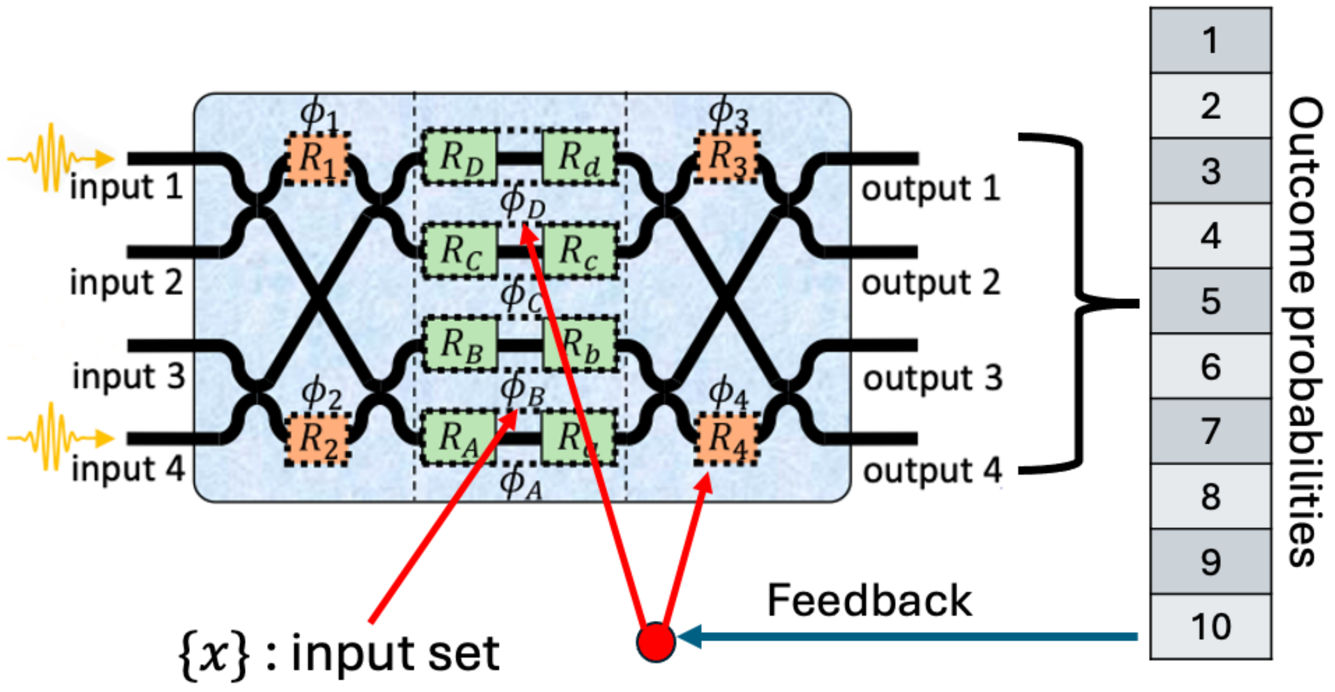
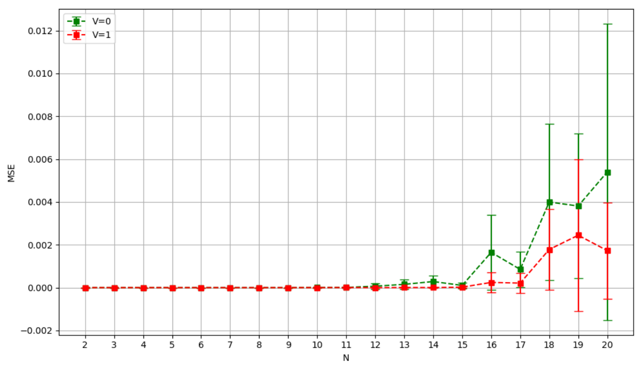


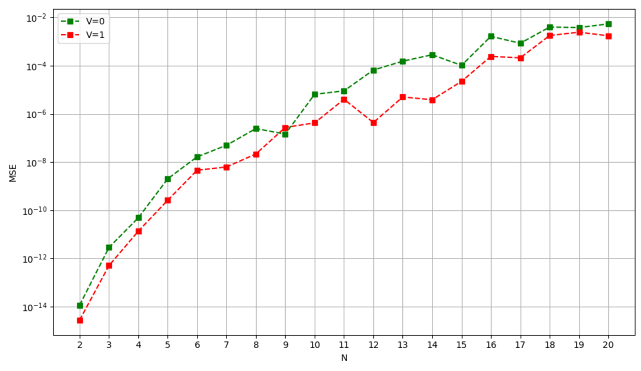
Figure 7: Diagram of the experimental encoding and feedback scheme. Input data are encoded in phase ϕ_B. Two of the outcome probabilities are used to set the phases ϕ_D and ϕ_4. This feedback mechanism allows to retain memory of the previous state.

7.1 Quantum Reservoir Simulation

Before performing real measurements,we simulated the system for both the **distinguishable** ($V = 0$) and **indistinguishable** ($V = 1$) photon scenarios. To perform the simulation we first built the data set as in the digital reservoir case. Then we encoded the input data by setting the phases. The output probabilities were calculated using the analytical model for two-photon interference using the actual physical parameters of the photonic chip; the analytical function is different for the two photon scenarios $V = 0, 1$. We performed a numerical optimization to find the best set of hyperparameters: the scaling factor for mapping input values to phase ϕ_B, the outcomes used for feedback (memory), their associated leaky rate, and the regularization parameter for the *Ridge regression*. As in the digital reservoir case, the optimization is performed using *Optuna*; the optimization process is repeated $M = 10$ times for each value N-the order of the monomial functions- and of the visibility V. Finally, using the optimized parameters, we evaluated the model's performance in both the cases using the mean squared error (MSE).



(a) MSE in linear scale.



(b) MSE in logarithmic scale.

Figure 8: Comparison of mean squared error (MSE) for distinguishable ($V = 0$) and indistinguishable ($V = 1$) photons, in linear and logarithmic scales.

Results show that the $V = 1$ model (indistinguishable photons) slightly outperforms the $V = 0$ case in this task. This advantage becomes more evident as the order N of the polynomial functions increases.
This result suggests that, similarly to how increasing the dimensionality in classical reservoirs enhances performance, quantum features such as photon indistinguishability may provide a performance benefit over the distinguishable photons scenario.

7.2 From Ideal to Practical Implementation

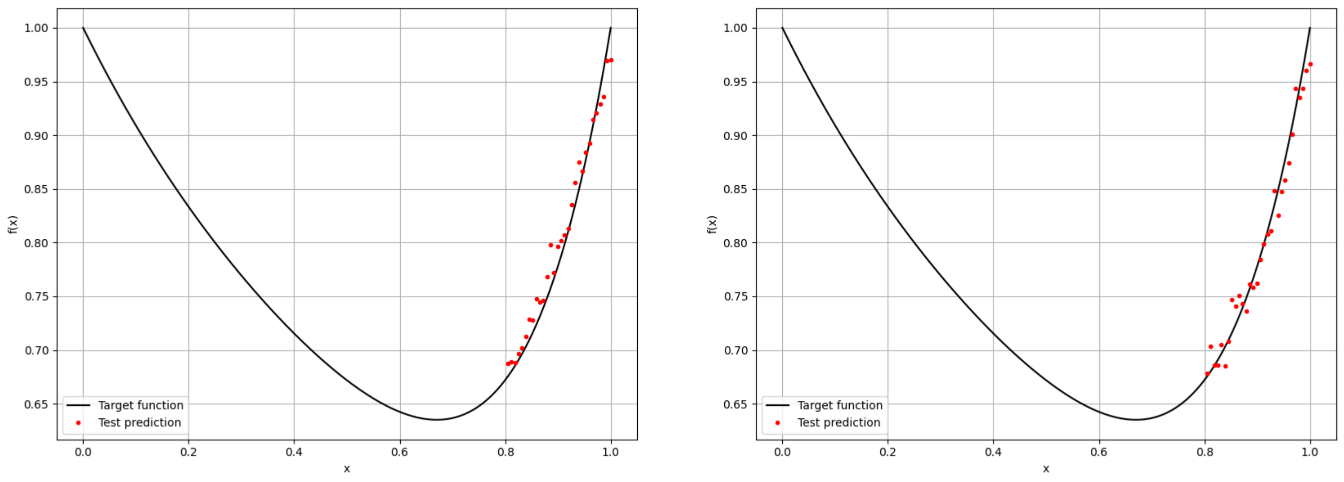
While the ideal simulation assumes full control over continuous phase values and perfect measurement conditions, real-world implementations inevitably introduce constraints. In practice, we are limited to a discrete set of available phase values. Therefore, the next step is to perform an optimization that constrains the model to use only this finite set of phases. Moreover, instead of relying on theoretical probabilities obtained from the analytical model, we simulated experimental data by generating photon detection events using a binomial distribution. The success probabilities were set to the theoretical values, while the number of trials n_γ corresponds to the number of detected photons. We used a realist value $n_\gamma = 20000$, which is reproducible in the actual experiment.
The target function to be reconstructed is

$$f(x) = \sum_{n=0}^{6} (-x)^n,$$

We repeated this procedure for both scenarios $V = 0$ and $V = 1$. The optimization was performed evaluating the average of the mean squared error (MSE) on the test set over $M = 5$ attempts for each trial set of parameters in order to avoid an influence by the photons statistic. Once obtained the optimal parameters, we evaluated the performance by repeating the reconstruction multiple times and computing the average mean squared error (MSE). Figure 9 shows an example of the reconstructed function for both models.

Parameter	$V = 1$	$V = 0$
ϕ_D (fd)	1.316	-2.659
ϕ_4 (f4)	1.729	5.352
ϕ_{in} (a_in)	-3.142	-3.142
Output for memory (outd)	2	6
Output for memory (out4)	7	1
Ridge regularizer α	10^{-13}	10^{-5}

Table 2: Optimized parameters for the cases $V = 1$ and $V = 0$.



(a) Reconstruction using the $V = 0$ model (distinguishable photons).

(b) Reconstruction using the $V = 1$ model (indistinguishable photons).

Figure 9: Example of target function reconstruction using optimized simulated quantum reservoirs. Comparison between distinguishable ($V = 0$) and indistinguishable ($V = 1$) photon scenarios.

	V=0	**V=1**
MSE	$(4.28 \pm 1.38) \times 10^{-4}$	$(6.73 \pm 3.72) \times 10^{-4}$

Table 3: Average mean squared error (MSE) obtained from the simulated reconstruction of the target function using the optimized parameters for both scenarios: distinguishable photons ($V = 0$) and indistinguishable photons ($V = 1$).

Contrary to the previous results, the new analysis shows that the $V = 0$ model (distinguishable photons) slightly outperforms the $V = 1$ model. This is not entirely surprising. Indeed, in the previous case, a significant difference between the two models started to emerge only for polynomial orders $N \gtrsim 16$ (see Figure 8). Therefore, under the constraints imposed by the limited set of available phases, the $V = 0$ model appears to perform better on this specific task.

7.3 From Simulation to Experiment

We performed the real experiment using the optimized parameters (see Table 2).
First, we adjusted a piezoelectric stage to set the configuration to either $V = 1$ or $V = 0$. The measured visibility in the indistinguishable case is $V_{\text{exp}} = 0.77$, which differs from the ideal value $V = 1$. This discrepancy is likely due to the fact that the beam splitters are not perfectly symmetric and the photon spectra are not exactly matched, so the photons are not perfectly indistinguishable. This resulted in a degradation of the visibility during the actual experiment. From now, we refer to the V=1 model as the V=0.77 model, using the experimental value of the visibility.
Then we defined the detection settings:

- **Exposure time**: the time interval during which the detector is active;
- **Number of acquisitions**: instead of a single long exposure (which could increase noise), we performed multiple acquisitions with shorter exposure times;
- **Coincidence Delay**: the maximum time difference between two detection events to be considered a coincidence;

Parameter	**Value**
Exposure time [s]	5
Number of acquisitions	19

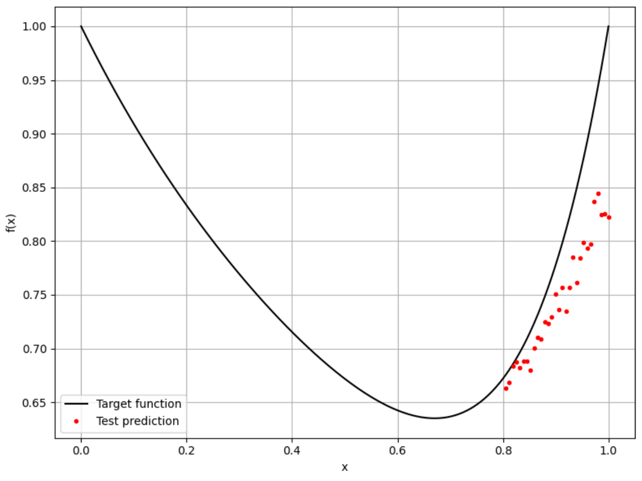
Table 4: Summary of the experimental parameters used in the data acquisition process.

We generated the input dataset $\{x\}$ taking $k = 150$ equally spaced values in the interval $[0, 1]$ and computed the target set $\{y\} = f(\{x\})$ where f is the function $f(x) = \sum_{n=1}^{6}(-x)^n$.
For each input x_i, currents were automatically regulated by the control system to set the corresponding phases. The acquisition process was then executed, and the output event frequencies were collected. This procedure is repeated for both the scenarios: $V = 0.77$ and $V = 0$.
The entire process was fully automated via a Python script.
The output event frequencies were then fed into the *Ridge regression model* to perform a further optimization on the Ridge regularization parameter α and, finally, to reconstruct the target function.

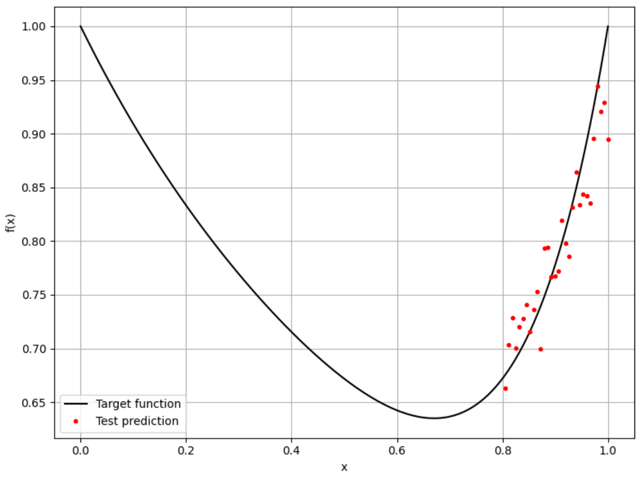
	$\mathbf{V_{\text{th}} = 0}$	$\mathbf{V_{\text{th}} = 1}$
	$\mathbf{V_{\text{exp}} = 0}$	$\mathbf{V_{\text{exp}} = 0.77}$
MSE	5.31×10^{-3}	1.30×10^{-3}

Table 5: MSE from the experimental reconstruction using optimized parameters. The measured visibility in the $V_{\text{th}} = 1$ case was $V_{\text{exp}} = 0.77$.

Results show that, contrary to the simulation, the $V = 1$ model (indistinguishable photons) slightly outperforms the $V = 0$ model (distinguishable photons). As previously discussed, this outcome is consistent with the observation that, for polynomial functions of this order, the performance gap between the two models is less pronounced than at higher orders. Examining the function reconstruction (Figure 10), the $V = 0.77$ model appears to reproduce the behavior of the test set more accurately, while the performance of the $V = 0$ model shows a greater degradation. This



(a) Reconstruction of the target function using the $V = 0$ model (distinguishable photons).



(b) Reconstruction using the $V = 0.77$ model (almost indistinguishable photons).

Figure 10: Reconstruction of the target function with optimized experimental quantum reservoirs. Comparison between distinguishable ($V = 0$) and almost indistinguishable ($V = 0.77$) photon scenarios.

suggests that the indistinguishable case may be less sensitive to experimental fluctuations—such as variations in the average number of detected photons. However, to draw a definitive conclusion, the experiment should be repeated multiple times, as was done in the simulations. Moreover, the performance of the $V = 0.77$ model could be improved by adjusting the visibility closer to the ideal value $V = 1$.

8 Conclusions

We have shown that a quantum reservoir implemented with an integrated photonic circuit can successfully reconstruct polynomial functions. Simulations performed on high-order polynomials—corresponding to an increased complexity of the system the model is dealing with—suggest that the use of indistinguishable photons allows the model to achieve better performance respect to the distinguishable case. This result shows how a purely quantum effect, such as photon indistinguishability, can influence a classical learning task.

The experimental reconstruction of a nonlinear function appears to support this conclusion. However, to confirm this result more definitively, further experiments on more complex target functions should be conducted. This would help clarify whether indistinguishability offers a consistent advantage in quantum reservoir computing architectures. If this is the case, while improving a standard digital model typically requires increasing the reservoir size -as shown previously- a quantum reservoir unlocks an additional "degree of freedom": using the same experimental apparatus, performance can be enhanced by exploiting photon indistinguishability.

A Appendix

In this section codes used to perform the simulations are provided.

A.1 Echo-State Network (ESN)

Below is the implementation of the class representing the Echo State Network (ESN). This class includes both the reservoir and the linear readout, which is implemented using Ridge regression. We first define all the parameters that characterize the ESN. Then, we implement the functions to update the reservoir state, perform predictions of the target function, and train the ESN model.

```
class ESN:
def __init__(self, input_dim, reservoir_dim, output_dim, sr=1.0, leaky_rate=1,
             input_scaling=1, alpha=1e-15, activation='tanh',seed=None):
    if seed is not None:
        np.random.seed(seed)
    # Input dimension (number of features in the input)
    self.input_dim = input_dim
    # Reservoir dimension (number of internal recurrent units)
    self.reservoir_dim = reservoir_dim
    # Output dimension (number of target outputs)
    self.output_dim = output_dim

    #self.bias = np.random.uniform(-0.1, 0.1, (self.reservoir_dim, 1))
    self.bias = np.random.uniform(-input_scaling, input_scaling, (self.reservoir_dim, 1))
    # Spectral radius: controls the echo state property and dynamics stability
    self.sr = sr
    # Leaky rate: defines the memory decay; lower values lead to longer memory
    self.leaky_rate = leaky_rate
    # Input scaling: controls the influence of the input signal on the reservoir
    self.input_scaling = input_scaling
    # Ridge regression regularization parameter (to prevent overfitting)
    self.alpha = alpha
    # Activation function applied in reservoir units (non-linearity)
    self.activation = activation

    # Input weight matrix Win: maps input to reservoir, scaled by input_scaling
    self.Win = np.random.uniform(-self.input_scaling, self.input_scaling,
                                (reservoir_dim, input_dim))
    # Reservoir weight matrix W: internal recurrent weights
    self.W = np.random.uniform(-1, 1, (reservoir_dim, reservoir_dim))

    # Normalize W to have the desired spectral radius (largest absolute eigenvalue)
    radius = max(abs(np.linalg.eigvals(self.W)))
    if radius < 1e-10:
        print("Warning: Spectral radius too small, adjusted to 1e-10.")
        radius = 1e-10
    self.W *= (sr / radius)

    # Initial reservoir state (typically zero)
    self.state = np.zeros((reservoir_dim, 1))
    # Linear readout using Ridge regression
    self.ridge = Ridge(alpha=self.alpha)

def _apply_activation(self, x):
    """Apply the selected activation function element-wise."""
    if self.activation == 'tanh':
        return np.tanh(x)
    elif self.activation == 'relu':
        return np.maximum(0, x)
    elif self.activation == 'sigmoid':
        return 1 / (1 + np.exp(-x))
    elif self.activation == 'identity':
        return x
    else:
        raise ValueError(f"Unknown activation function: {self.activation}")

def update(self, x):
    """Update the reservoir state for a single time step."""
    pre_activation = np.dot(self.Win, x) + np.dot(self.W, self.state) + self.bias
    activated = self._apply_activation(pre_activation)
    # Leaky integration: mix previous state with current activated state
    self.state = (1 - self.leaky_rate) * self.state + self.leaky_rate * activated
    return self.state

def train(self, X_train, y_train,):
    """Train the readout layer (Ridge regression) on reservoir states."""
    reservoir_states = []
    self.state = np.zeros((self.reservoir_dim, 1))
    for x in X_train:
        state = self.update(x)
        reservoir_states.append(state.flatten())
    reservoir_states = np.array(reservoir_states)
    self.ridge.fit(reservoir_states , y_train)

def predict(self, X_test):
    """Predict output values for input sequences using the trained readout."""
    test_states = []
    for x in X_test:
        state = self.update(x)
        test_states.append(state.flatten())
    test_states = np.array(test_states)
    return self.ridge.predict(test_states)
```

The optimization is carried out using the *Optuna* library. This library requires the definition of an objective function, which must return the quantity to be minimized—in this case, the mean squared error (MSE). The procedure begins by generating the input values and the corresponding target function. Both sets are then split into a training set and a test set: the former is used to train the model, while the latter is used to evaluate its performance via the MSE. Next, we define the search space for the hyperparameters to be optimized. The objective function performs the training process and returns the MSE computed on the test set. As described previously, this optimization is repeated $M = 20$ for each combination of N (polynomial order) and D (reservoir size), in order to reduce the impact of random initialization.

```
def objective(trial,N,D):
def func(x,N): return x**N
# Generate data
X_train = x_k[:cut, :]
X_test = x_k[cut:, :]
y_train = func(X_train, N)
y_test = func(X_test, N)

# Define the hyperparameters to optimize
parameters = {
    'input_dim': 1,
    'reservoir_dim': D,
    'output_dim': 1,
    'sr': trial.suggest_float('sr', 1e-13, 0.7, log=True),
    'leaky_rate': trial.suggest_float('leaky_rate', 0.5, 1.),
    'input_scaling': 1,
    'alpha': trial.suggest_float('alpha', 1e-15, 1e-2, log=True),
    'activation': trial.suggest_categorical('activation', ['identity','tanh', 'relu', 'sigmoid']),
    'seed': 12345
}
# Create the ESN model with the suggested hyperparameters
esn = ESN(**parameters)
# Train the ESN model
esn.train(X_train, y_train)
# Make predictions on the test set
y_pred = esn.predict(X_test)
test_mse = np.mean((y_pred - y_test)**2)
```

```
return test_mse
```

A.2 Ideal Quantum Reservoir

Below is the objective function used to perform the *Optuna* optimization in the ideal quantum reservoir case. The structure of this function is similar to that used for the ESN model, but it differs in the set of hyperparameters and in the implementation of the reservoir. The hyperparameters include: the input scaling factor (`a_in`) used to encode the data into the phase ϕ_B; two integers (`outcome`, `outcome1`) specifying which output probabilities to use as feedback; the corresponding leaky rate values (`f`, `f1`) that determine the memory contribution of these feedback signals; and the regularization parameter for the Ridge regression model(`alpha`). All internal phases are initially set to zero. The output probabilities are then evaluated using the analytical function `lHd_2photon_ideal()`.

```
def objective(trial,V,N):
fnl_k = x_k**N
s_k = fnl_k

#  parameters
a_in = trial.suggest_float("a_in", -np.pi, np.pi)
f = trial.suggest_float("f", -2*np.pi, 2*np.pi) # 'leaky rate' per outecome
f1 = trial.suggest_float("f1", -2*np.pi, 2*np.pi) # 'leaky rate' per outecome1
outcome = trial.suggest_int("outcome", 0, 9)
outcome1 = trial.suggest_int("outcome1", 0, 9)
alpha = trial.suggest_float("alpha", 1e-16, 1e-3, log=True)

phases = {"phi1": np.zeros(k), "phi2": np.zeros(k),
  "phiA": np.zeros(k), "phiB": np.zeros(k),
  "phiC": np.zeros(k), "phiD": np.zeros(k),
  "phia": np.zeros(k), "phib": np.zeros(k),
  "phic": np.zeros(k), "phid": np.zeros(k),
  "phi3": np.zeros(k), "phi4": np.zeros(k)}

phases["phiB"] = a_in * x_k

P = np.zeros((10, k))

for i in range(k):
    for j in range(10):
        P[j, i] = lHd_2photon_ideal(
            j, [0]*5,
            np.array([phases['phiA'][i], phases['phiB'][i],
                      phases['phid'][i], phases['phi3'][i], phases['phi4'][i]]), V=V)

    if i < k - 1:
        phases["phid"][i+1] = P[outcome, i] * f
        phases["phi4"][i+1] = P[outcome1, i] * f1

# Train/test split
cut = int(0.8 * k)
X_train, y_train = P.T[:cut], s_k[:cut]
X_test, y_test = P.T[cut:], s_k[cut:]
# Ridge regression
model = Ridge(alpha=alpha)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
test_mse = np.mean((y_pred - y_test)**2)

return test_mse
```

A.3 Non-Ideal Quantum Reservoir

Objective function used for the optimization of the non-ideal quantum reservoir in the $V = 1$ scenario. with the only difference being the function used to evaluate the output probabilities: `1Hd_2photon_fit_VX()`, where $X = 0, 1$ indicates the visibility setting. The structure of the function is similar to the one used in the ideal case, with a few key differences. In particular, we introduce the function `setta_fasi_theo_kdtree_exclude()` to determine the experimental phases corresponding to the ideal ones. This function exploits the `KDTree` algorithm from the Python *scipy* library to efficiently find the closest available phase values. Indeed, the set of experimentally accessible phases is stored as a three-dimensional tensor with dimensions on the order of 10^6x10^6x10^6. Running over all the indices to find the optimal match would be computationally prohibitive.

so running on all the indices of this tensor to find the best values is an extremely computational expensive task. The KDTree algorithm provides an efficient solution to this problem. Once the theoretical probabilities are computed, we simulate the experimental detection process using a binomial distribution. The resulting frequencies serve as the reservoir output used for training the model.

```
def make_objective(fasi_c, tree,N):
    def objective(trial):
        k = 150
        sts = 20000

        s_k = np.linspace(0, 1, k)
        train_size = int(0.8 * k)
        #train_s_k, test_s_k = s_k[:train_size], s_k[train_size:]
        mse = 0

        fd = trial.suggest_float("fd", -10.0, 10.0)
        f4 = trial.suggest_float("f4", -10.0, 10.0)
        a_in = trial.suggest_categorical("a_in", [-np.pi, -0.5*np.pi, 0.5*np.pi, np.pi])
        outd = trial.suggest_int("outd", 0, 9)
        out4 = trial.suggest_int("out4", 0, 9)
        exp = trial.suggest_int("exp", -15, 0)
        alpha = 10 ** exp

        M=5
        for _ in range(M):
            # Phases initialization
            fasi = {
                "phiA": np.zeros(k),
                "phiB": np.zeros(k),
                "phid": np.zeros(k),
                "phi3": np.zeros(k),
                "phi4": np.zeros(k)
            }
            fasi["phiB"] = a_in * s_k
            P = np.zeros((10, k))
            Prec = np.zeros((10, k))
            data = np.zeros((k, 3))
            used_indices = set()

            for i in range(k):

                c = np.angle(np.exp(1j * np.array([fasi["phiB"][i],
                             fasi["phid"][i], fasi["phi4"][i]])))

                cres0, idx = setta_fasi_theo_kdtree_exclude(c, fasi_temp, tree,
                                                            used_indices, max_k=k)
                cres = np.zeros(5)
                cres[1], cres[2], cres[4] = cres0[0], cres0[1], cres0[2]

                data[i, :] = cres[1], cres[2], cres[4]

                for d in range(10):
                    # Sostituisci con la tua funzione
                    P[d, i] = lHd_2photon_fit_V1(d, fasi_offset(0), cres)

                counts = np.random.multinomial(sts, P[:, i])
                Prec[:, i] = counts / sts

                if i < k - 1:
                    fasi["phid"][i+1] = Prec[outd, i] * fd
                    fasi["phi4"][i+1] = Prec[out4, i] * f4

            X_train, y_train = Prec.T[:train_size], fnl_k(s_k[:train_size],N) #s_k[:train_size]**N
            X_test, y_test = Prec.T[train_size:], fnl_k(s_k[train_size:],N)  #s_k[train_size:]**N

            X_train = X_train.reshape(X_train.shape[0], -1)
            X_test = X_test.reshape(X_test.shape[0], -1)

            model = Ridge(alpha=alpha)
            model.fit(X_train, y_train)
            y_pred = model.predict(X_test)

            mse += np.mean((y_pred - y_test)**2)
        return mse/M  # Average MSE over all sts values

    return objective
```

References

[1] Gerry C, Knight P.
Optical test of quantum mechanics.
In: *Introductory Quantum Optics.*
Cambridge University Press; 2004:213–237.

[2] Gerry C, Knight P.
Beam splitters and interferometers.
In: *Introductory Quantum Optics.*
Cambridge University Press; 2004:135–149.

[3] M. Valeri, V. Cimini, S. Piacentini, F. Ceccarelli, E. Polino, F. Hoch, G. Bizzarri, G. Corrielli, N. Spagnolo, R. Osellame, and F. Sciarrino,
Experimental multiparameter quantum metrology in adaptive regime,
Phys. Rev. Res. **5**, 013138 (2023).
`https://link.aps.org/doi/10.1103/PhysRevResearch.5.013138`

[4] Susan Stepney.
Physical reservoir computing: a tutorial.
Natural Computing, 23:665–685, 2024.
doi: 10.1007/s11047-024-09997-y.