Automatic differentiation of photonic circuits

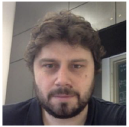
Zoltán Kolarovszki

Eötvös Loránd University
Wigner Research Centre for Physics

May 15, 2023

PIQUASSO

Piquasso team



Zoltán Zimborás

Zoltán Kolarovszki

Péter Rakyta

Tamás Kozsik

Szabolcs Jóczik

Ágoston Kaposi

Gábor Németh

Zsófia Kallus

Tomasz Rybotycki

Michal Oszmaniec

Boldizsár Joó

Henrik Varga

Outline

Photonic Quantum Computing

Photonic circuits as neural networks

Why simulate a photonic quantum computer?

Piquasso simulator framework

Tensorflow integration

Photonic Quantum Computing

Photonic circuits as neural networks

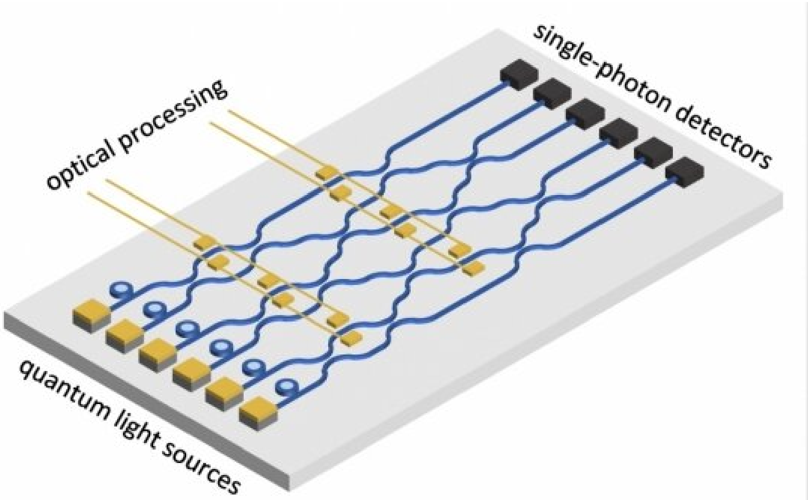
Why simulate a photonic quantum computer?

Piquasso simulator framework

Tensorflow integration

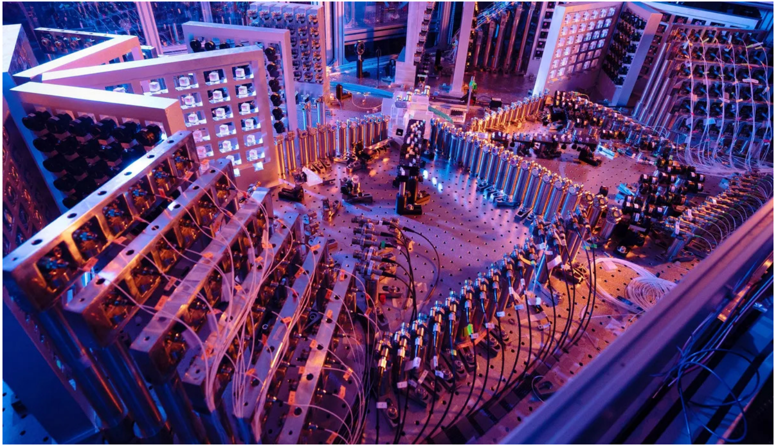
Photonic Quantum Computing

A photonic quantum computer stores information in independent optical modes, called **qumodes**.



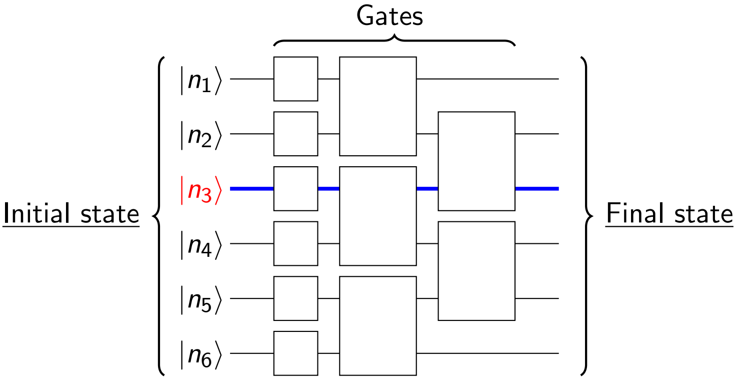
Quantum advantage

The Quantum Information Group of USTC in Hefei (led by Jian-Wei Pan) demonstrated advantage over classical computation in 2020 (with improvements in 2021 and 2023, in the latter mentioning our method as classical benchmark).



PIQUASSO

Modeling an optical circuit



States can be written as:

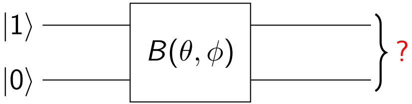
$$|\psi\rangle := \sum_{n_1,\dots,n_d \in \mathbb{Z}^d_{\geq 0}} c_{n_1,\dots,n_d} |n_1 \ \dots \ n_d\rangle \,. \tag{1}$$

Example state: $|\psi\rangle = \frac{1}{\sqrt{2}} |01\rangle + \frac{1}{\sqrt{2}} |10\rangle \,.$

PIQUASSO

Simple example

Circuit with a single beamsplitter gate:



Output state:

$$? = B(\theta, \phi)\,|10\rangle = \cos\theta|10\rangle + e^{i\phi}\sin\theta|01\rangle. \tag{2}$$

Probability distribution: (using Born's rule)

$$\begin{aligned} p(\text{output} = 10) &= \cos^2\theta, \\ p(\text{output} = 01) &= \sin^2\theta. \end{aligned} \tag{3}$$

Idea: Photonic circuits as neural networks?

Qubit-based quantum computing $\implies$ measurement outputs are generally **discrete**.

Photonic quantum computing $\implies$ measurement outputs are **continuous**.

photonic circuit $\sim$ **neural network**

gate parameters $\sim$ **weights**

Circuits are differentiable, e.g.

$$\partial_\theta B(\theta, \phi) |01\rangle = - \sin\theta |01\rangle + e^{i\phi} \cos\theta |10\rangle .$$

PIQUASSO

Neural network model in photonic quantum computing

A classical neural network model is

$$\vec{y} = \mathcal{L}_n \dots \mathcal{L}_1(\vec{x}), \tag{4}$$

- $\vec{x}$ input, $\vec{y}$ output,
- $\mathcal{L}_i$ **neural network layer**,

$$\mathcal{L}_i = \varphi(W\vec{x} + \vec{b}) \tag{5}$$

- $W\vec{x} + \vec{b}$ is **linear** transformation,
- φ **nonlinear** "activation" function.

How can we adapt this contruction to photonic quantum computing?

Photonic Analogue

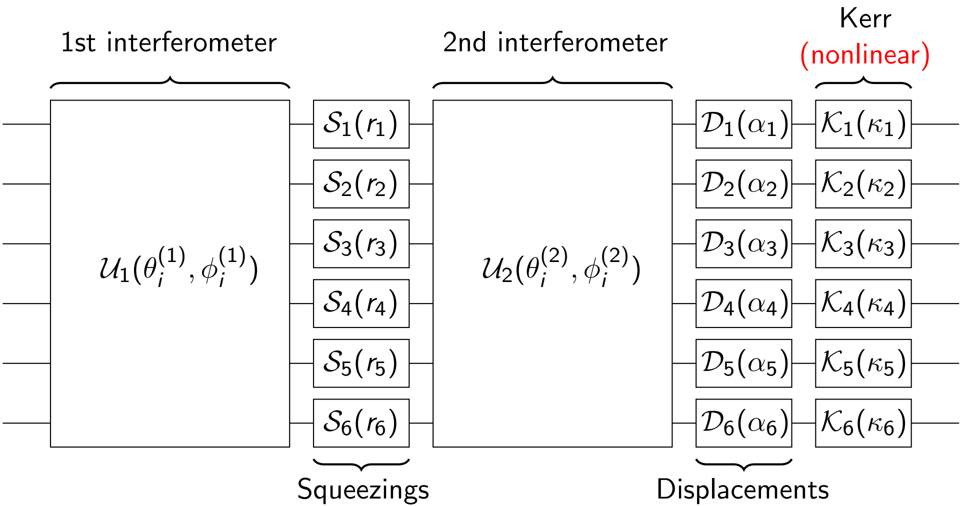
$$\left|\psi'\right\rangle = \mathcal{L}_n \dots \mathcal{L}_1 \left|\psi\right\rangle , \tag{6}$$

- $|\psi\rangle$ input, $|\psi'\rangle$ output state,
- $\mathcal{L}_i := \mathcal{L}_i(\vec{\theta})$: **continuous-variable quantum neural network layers** (CVNN),
- $\vec{\theta}$: **set of parameters** $\iff$ **weights**.

$\mathcal{L}_i$ should be the composition of a **linear** and a **non-linear** transformation.

PIQUASSO

Continuous-Variable Neural Network layer



$$\mathcal{L} := \mathcal{K} \circ \mathcal{D} \circ \mathcal{U}_2 \circ \mathcal{S} \circ \mathcal{U}_1$$

Simulating photonic quantum circuits is needed!

- Photonic quantum computers are still **not widely available**,
- Trying to approximate quantum computing may inspire **better classical algorithms**,
- Can be used to **test hardware**,
- **Aids implementation** of quantum circuits (state learning, gate synthesis).

However: Simulating photonic quantum circuts is classically hard!

PIQUASSO

Example problem: State learning

Given a photonic quantum state $|\psi\rangle$, **how can we prepare it using a photonic quantum computer?**

Solution: CVNN layers!

Cost function:

$$J(|\psi\rangle) = \| |\psi\rangle - |\psi^*\rangle \|_1 \tag{7}$$

$$|\psi^*\rangle = \mathcal{L}_n \circ \cdots \circ \mathcal{L}_1 |0\rangle , \qquad \mathcal{L}_i \textbf{ CVNN layers}. \tag{8}$$

Differentiating CVNN layers $\implies$ **backpropagation**.

PIQUASSO

Photonic Quantum Computing

Photonic circuits as neural networks

Why simulate a photonic quantum computer?

Piquasso simulator framework

Tensorflow integration

Simulating a photonic quantum computer with Piquasso

We are developing a new simulator framework written in Python called **Piquasso**.

We wanted to have a simulator we could experiment with and we could extend and improve by ourselves.

It is also beneficial to have multiple simulators for testing hardware.

Main goals:

- Extensibility (ability to write plugins),
- High performance (via C++ PiquassoBoost plugin),
- Reproducibility,
- Clean code.

Piquasso is open source, available on PyPI:

```
pip install piquasso
```

Simulation of nonlinear gates

Nonlinear gates (the Kerr gate) can only be simulated using the **Fock simulator**, where states are stored with their coefficients in the Fock basis.

We need to differentiate the **Fock simulator**.

Example: Coherent state

$$|\alpha\rangle := e^{-\frac{|\alpha|^2}{2}} \sum_{n=0}^{\infty} \frac{\alpha^n}{\sqrt{n!}} |n\rangle , \tag{9}$$

but we cannot store every coefficients, we need a truncation:

$$|\alpha\rangle := e^{-\frac{|\alpha|^2}{2}} \sum_{n=0}^{c} \frac{\alpha^n}{\sqrt{n!}} |n\rangle . \tag{10}$$

PIQUASSO

2 ways of truncating Fock space

The Fock simulation is an **approximation**.

One has to make a choice which occupation numbers are considered. When imposed, makes the dimension of the space of the system finite.

Strawberry Fields: Local cutoff
Constraint on the particle number **by mode**. State vector size:

$$c^d, \qquad c: \ \textbf{local}\text{ cutoff}, \ d: \ \text{number of modes.} \tag{11}$$

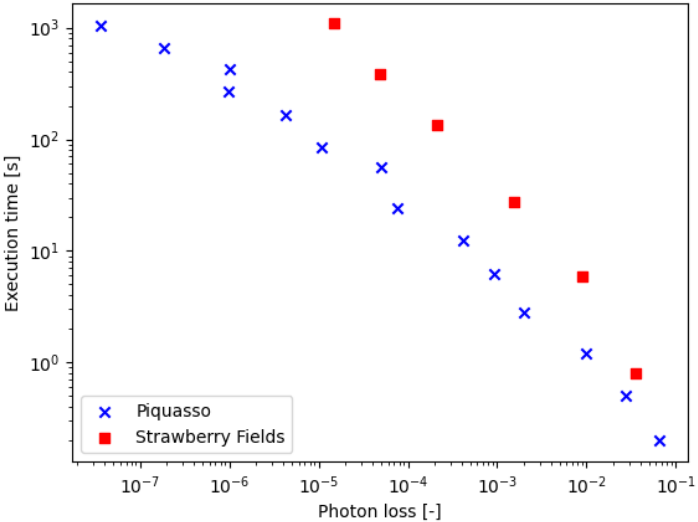
Piquasso: Global cutoff
Constraint on particle number on the whole system. State vector size:

$$\binom{d+c-1}{c-1}, \qquad c: \ \textbf{global}\text{ cutoff}, \ d: \ \text{number of modes.} \tag{12}$$

Cutoff results in photon loss.

PIQUASSO

Photon loss in Piquasso vs. Strawberry Fields



PIQUASSO

Simulation using Tensorflow

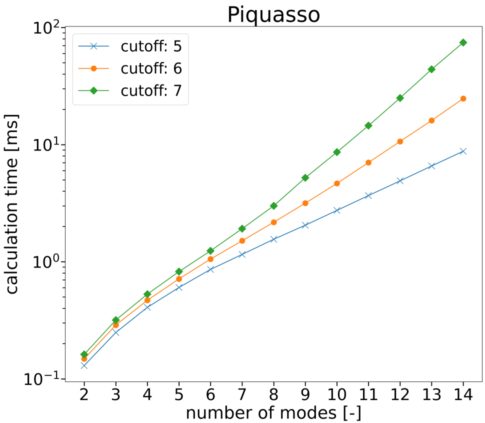
Piquasso now supports the Tensorflow machine learning platform.

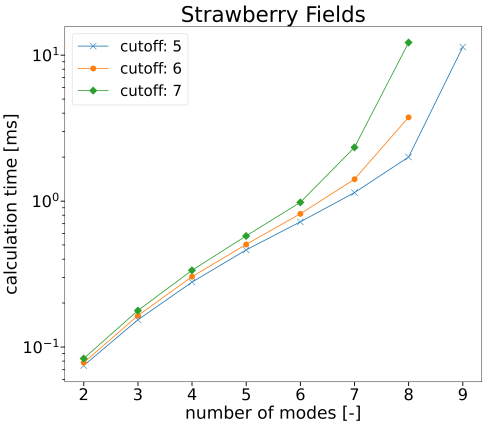
Automatic differentiation of optical circuits

```
config = pq.Config(cutoff=4)
simulator = pq.TensorflowPureFockSimulator(d=3)
theta_1, theta_2 = tf.Variable(1.0), tf.Variable(2.0)
xi = tf.Variable(3.0)

with pq.Program() as program:
    pq.Q() | pq.Vacuum() | pq.Displacement(alpha=[0.1, 0.2, 0.3])
    pq.Q(0, 1) | pq.Beamsplitter(theta=theta_1)
    pq.Q(1, 2) | pq.Beamsplitter(theta=theta_2)
    pq.Q() | pq.Kerr(xi=xi)

with tf.GradientTape() as tape:
    probabilities = simulator.execute(program).state.fock_probabilities

tape.jacobian(probabilities, [theta_1, theta_2, xi])
# [<tf.Tensor: shape=(20,), dtype=float32, numpy=array(
# [ 0.        ,  0.03394834, -0.02936802, -0.00458031,  0.00062407, ...
```

PIQUASSO

Benchmarking single CVNN layers





Thank you for your attention!