

PIQUASSO

Simulation of photonic quantum computing with automatic differentiation in Piquasso

Zoltán Kolarovszki

HUN-REN Wigner Research Centre for Physics
Eötvös Loránd University

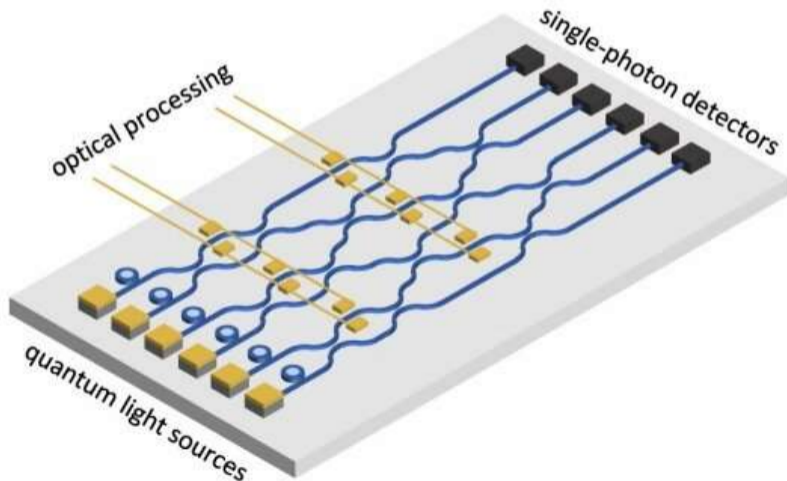
May 30, 2024

GPU day 2024



Photonic Quantum Computing

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



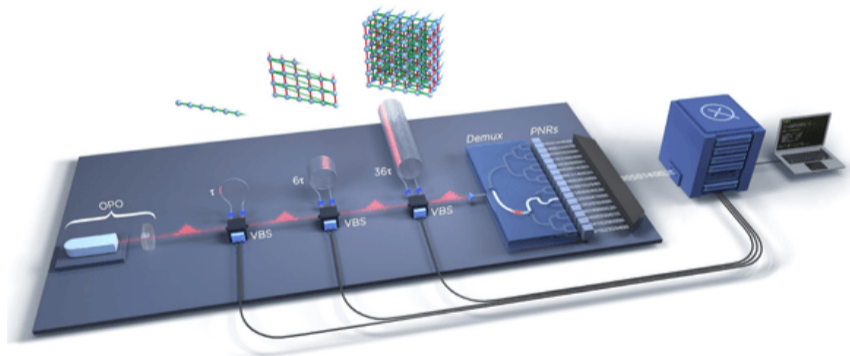
Quantum advantage by USTC

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



Quantum advantage by Xanadu

Xanadu also demonstrated an advantage over classical computation in 2022 on the Borealis chip, which is also publicly available.



Classical simulation of photonic quantum computers

Simulating photonic quantum circuits

We need to **simulate** photonic quantum computers, because:

- ▶ Photonic quantum computers are still **not widely available**, but we want to execute photonic quantum algorithms for research.
- ▶ It can aid **circuit design**.
- ▶ Trying to simulate quantum computing may inspire **better classical algorithms**.
- ▶ It can be used to **certify hardware**.
- ▶ It helps photonic quantum machine learning research via **automatic differentiation**.

Simulating photonic quantum circuits

We need to **simulate** photonic quantum computers, because:

- ▶ Photonic quantum computers are still **not widely available**, but we want to execute photonic quantum algorithms for research.
- ▶ It can aid **circuit design**.
- ▶ Trying to simulate quantum computing may inspire **better classical algorithms**.
- ▶ It can be used to **certify hardware**.
- ▶ It helps photonic quantum machine learning research via **automatic differentiation**.

However: Simulating photonic quantum circuits is classically hard in general!



PIQUASSO: PhotonIc QUAntum computer Simulator SOftware

We are developing and maintaining 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.

Main goals:

- ▶ Extensibility (e.g., TensorFlow, JAX)
- ▶ High performance (e.g., via C++ PiquassoBoost plugin)
- ▶ Reproducibility
- ▶ Clean code

Piquasso is open source, available on GitHub¹ and can be installed with:

```
pip install piquasso
```

¹<https://github.com/Budapest-Quantum-Computing-Group/piquasso>

Available simulators in PIQUASSO

Different state representations are useful depending on the scenario.

In Piquasso, you can choose from the following:

- ▶ `GaussianSimulator`: Simulates **Gaussian** states
- ▶ `SamplingSimulator`: Simulates the **Boson Sampling** scheme
- ▶ `FockSimulator`: Simulates **Fock** states
- ▶ `PureFockSimulator`: Simulates **pure** Fock states

Available simulators in PIQUASSO

Different state representations are useful depending on the scenario.

In Piquasso, you can choose from the following:

- ▶ `GaussianSimulator`: Simulates **Gaussian** states
- ▶ `SamplingSimulator`: Simulates the **Boson Sampling** scheme
- ▶ `FockSimulator`: Simulates **Fock** states
- ▶ `PureFockSimulator`: Simulates **pure** Fock states








Available simulators in **PiquassoBoost**:

- ▶ `BoostedGaussianSimulator`: Same as `GaussianSimulator`, but reimplemented in C++ with improved hafnian and torontonian calculations.
- ▶ `BoostedSamplingSimulator`: Same as `SamplingSimulator`, where sampling algorithm is reimplemented and parallelized in C++ with improved permanent calculation algorithms.

Usage example

```
1 simulator = pq.GaussianSimulator(d=5)
2
3 with pq.Program() as program:
4     pq.Q(all) | pq.Vacuum()
5
6     for i in range(5):
7         pq.Q(i) | pq.Squeezing(r=0.1) | pq.Displacement(r=1.0)
8
9     pq.Q(0,1) | pq.Beamsplitter(theta=np.pi / 3)
10    pq.Q(2,3) | pq.Beamsplitter(theta=np.pi / 4)
11    pq.Q(3,4) | pq.Beamsplitter(theta=np.pi / 5)
12
13    pq.Q(all) | pq.ParticleNumberMeasurement()
14
15 result = simulator.execute(program, shots=1000)
16 print(result.samples)
17 # [(0, 1, 1, 2, 2), (1, 3, 0, 0, 1), (0, 1, 0, 0, 3), ...
```

Choose your framework!

- ▶ **NumpyCalculator**: Default calculator, uses  NumPy,  SciPy and  Numba.
- ▶ **TensorflowCalculator**: Uses  TensorFlow for calculations (with graph compilation and  OpenXLA support).
- ▶ **JaxCalculator**: Uses  for calculations (with  OpenXLA support).



+


TensorFlow



Fock space truncation

The Fock simulation is an **approximation**, due to the ubiquitous cutoff.

STRAWBERRY FIELDS: **Local cutoff**

Constraint on the particle number **by mode**. State vector size:

$$c^d, \quad c : \text{local cutoff, } d : \text{number of modes.} \quad (1)$$

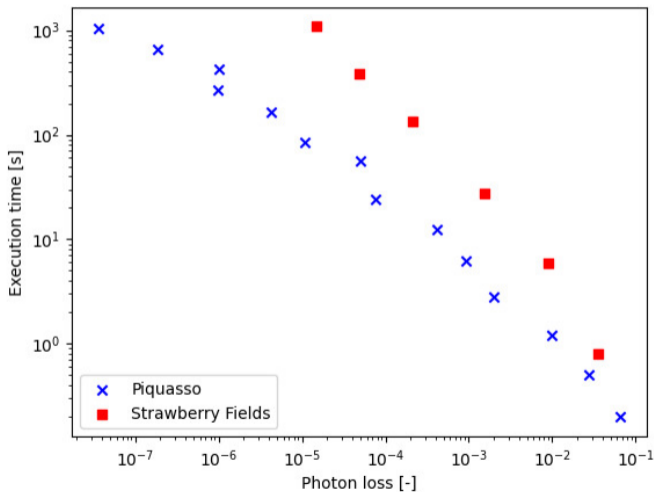
PIQUASSO: **Global cutoff**

Constraint on particle number on the **whole system**. State vector size:

$$\binom{d + c - 1}{c - 1}, \quad c : \text{global cutoff, } d : \text{number of modes.} \quad (2)$$

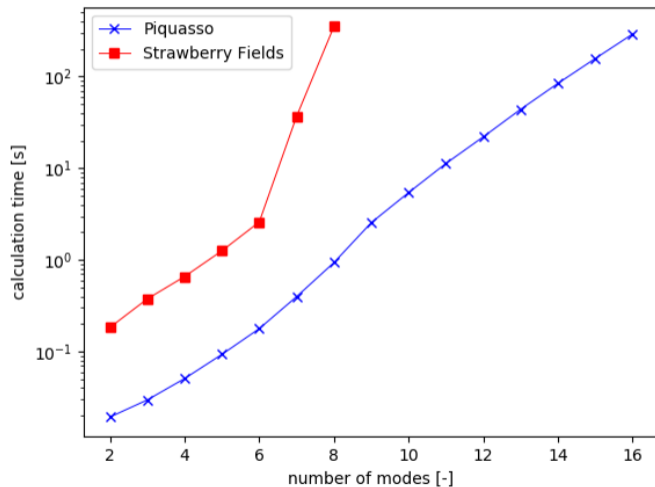
Contributions that are left out by using a global cutoff instead of a local one have small coefficients in most cases.

Photon losses²: PIQUASSO vs. STRAWBERRY FIELDS



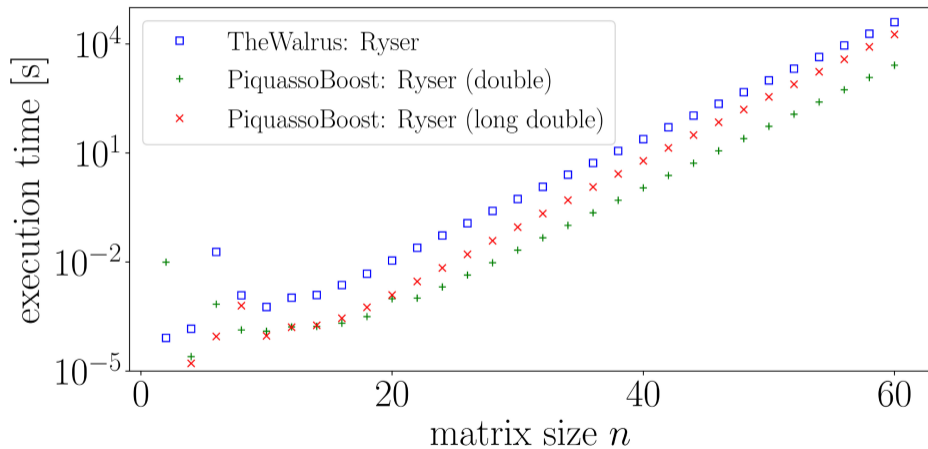
²In a circuit using 4 continuous-variable quantum neural network (CVQNN) layers on 8 modes.

Automatic differentiation³: PIQUASSO vs. STRAWBERRY FIELDS



³In a circuit using 4 CVQNN layers with cutoff 10.

Hafnian calculation speedup in PiquassoBoost



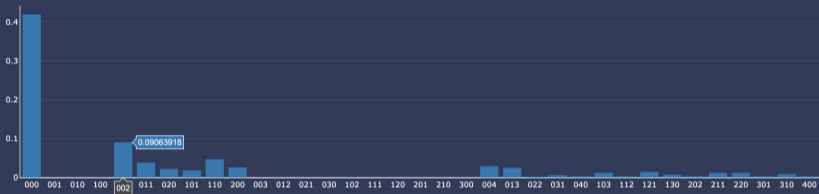
A web-based circuit composer is made available at <https://piquasso.com>.

The screenshot displays the PIQUASSO web-based circuit composer interface. The top navigation bar includes the PIQUASSO logo, a "My Project" tab, and "Sign in" and "Sign Up" links. Below the navigation bar is a menu with "File", "Share", and "Help" options, along with a "Make a copy" button. The main interface is divided into several sections:

- CONTROL:** Features buttons for "Run" (green play icon), "Stop" (red square icon), "Undo" (blue left arrow), and "Redo" (blue right arrow).
- SIMULATION:** Includes input fields for "Cutoff" (set to 5) and "HBar" (set to 2), and "Measurement cutoff" (set to 5) and "Shots" (set to 100).
- Component Palette:** Organized into categories: "Preparation" (M, Cov), "One Mode" (R, D, F, X, S, Z), "Two Mode" (B, MZ, S, CX, CZ), "Multi Mode" (I, GT), and "Measurement" (Ho, He, PN, T).
- Circuit Diagram:** A grid-based workspace showing a quantum circuit. It includes preparation blocks (S, R), single-qubit blocks (B), two-qubit blocks (B), and a measurement block (PN) with a downward-pointing arrow.

Particle Number Detection Probabilities

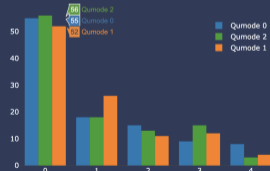
Export Data Export Image



Samples

Export Data Export Image

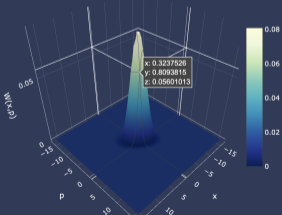
Qumode 0 Qumode 1 Qumode 2



Wigner Function

Export Image

Qumode 0 Qumode 1 Qumode 2



Python Code

Copy to clipboard

```
import numpy as np
import piquasso as pq

with pq.Program() as program:
    pq.Q(0) | pq.Squeezing(r=0, phi=0)
    pq.Q(1) | pq.Squeezing(r=1, phi=0)
    pq.Q(2) | pq.Squeezing(r=1, phi=0)
    pq.Q(0) | pq.Phaseshifter(phi=1.0472)
    pq.Q(1) | pq.Phaseshifter(phi=1.0472)
    pq.Q(2) | pq.Phaseshifter(phi=1.0472)
    pq.Q(0, 1) | pq.Beamsplitter(theta=0.3927, phi=0.7854)
    pq.Q(1, 2) | pq.Beamsplitter(theta=0.3927, phi=0.7854)
    pq.Q(0, 1) | pq.Beamsplitter(theta=0.3927, phi=0.7854)
    pq.Q(0, 1, 2) | pq.ParticleNumberMeasurement()

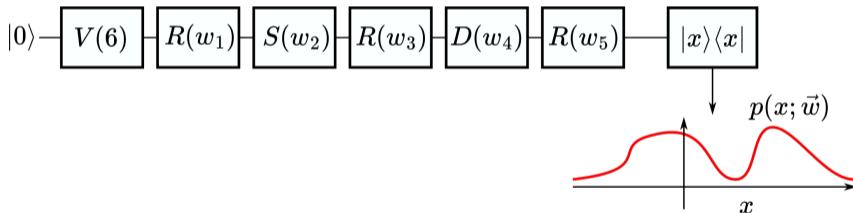
simulator = pq.GaussianSimulator(
    d=3, config=pq.Config(cutoff=5)
)

result = simulator.execute(program, shots=100)
```

Applications

I. Training continuous-variable Born machines (CVBMs)⁴

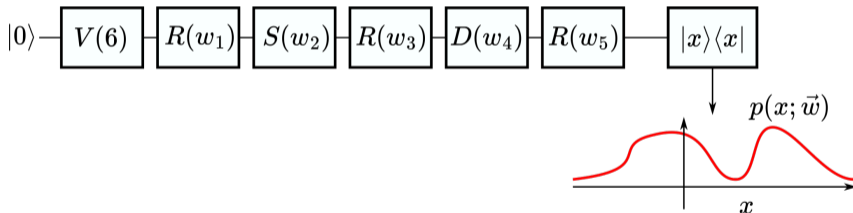
- ▶ **Born machine:** Parametrized quantum circuit producing probability distributions via **Born's rule** \implies generative quantum machine learning
- ▶ **CVBM:** Photonic quantum circuits can generate **position distribution**



⁴Z. Kolarovszki, D. T. R. Nagy, Z. Zimborás, “On the learning abilities of photonic continuous-variable Born machines” [submitted]

I. Training continuous-variable Born machines (CVBMs)⁴

- ▶ **Born machine:** Parametrized quantum circuit producing probability distributions via **Born's rule** \implies generative quantum machine learning
- ▶ **CVBM:** Photonic quantum circuits can generate **position distribution**



However, classically simulation is demanding, **even for a single mode!**

⁴Z. Kolarovszki, D. T. R. Nagy, Z. Zimborás, “On the learning abilities of photonic continuous-variable Born machines” [submitted]

I. Training continuous-variable Born machines (CVBMs)⁴

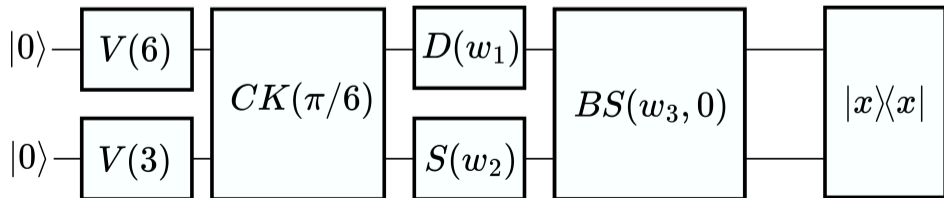
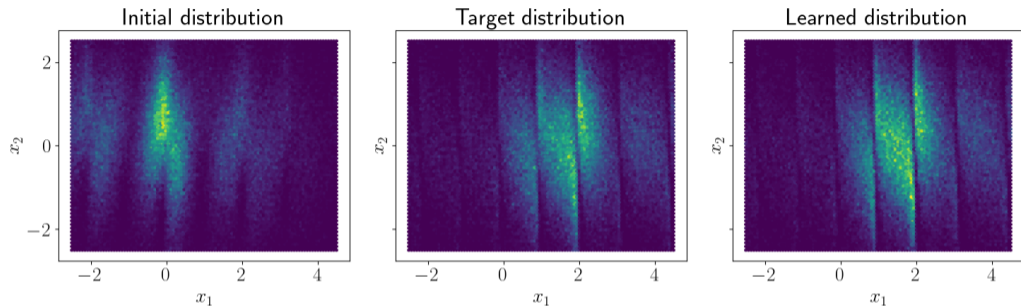
- ▶ **Born machine:** Parametrized quantum circuit producing probability distributions via **Born's rule** \implies generative quantum machine learning
- ▶ **CVBM:** Photonic quantum circuits can generate **position distribution**

PIQUASSO uses a new classical algorithm for homodyne measurement tailored for **multimode systems**.

$\sim 1000\times$ speedup over previous solutions!

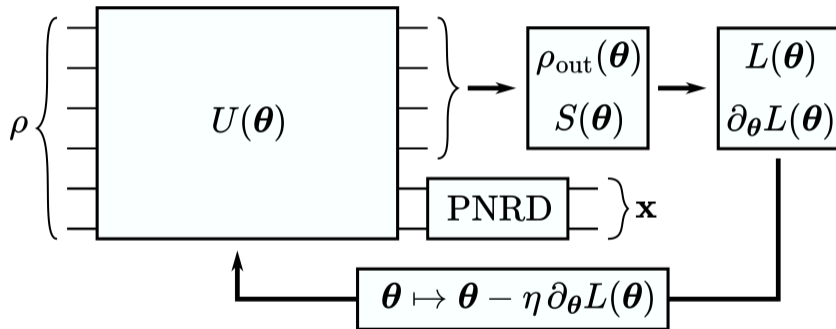
⁴Z. Kolarovszki, D. T. R. Nagy, Z. Zimborás, “On the learning abilities of photonic continuous-variable Born machines” [submitted]

I. Efficient training of multimode CVBMs with PIQUASSO



II. Optimizing non-deterministic gates with imperfect detections⁵

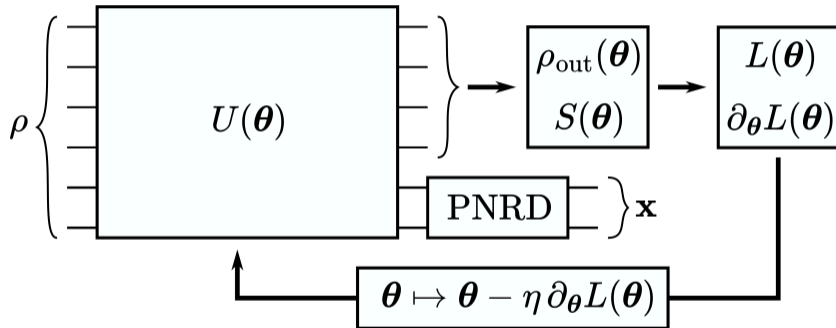
Nondeterministic gates can implement **qubit** gates on a photonic quantum computer.



⁵C. Czabán, Z. Kolarovszki, M. Karácsony, Z. Zimborás, "Suppressing photon detection errors in nondeterministic state preparation" [accepted]

II. Optimizing non-deterministic gates with imperfect detections⁵

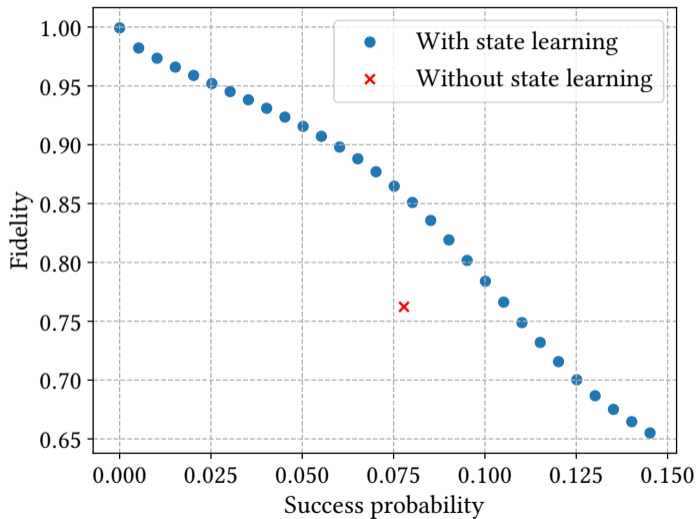
Nondeterministic gates can implement **qubit** gates on a photonic quantum computer.



Problem: Photon detectors have biases.

⁵C. Czabán, Z. Kolarovszki, M. Karácsony, Z. Zimborás, "Suppressing photon detection errors in nondeterministic state preparation" [accepted]

II. Fidelity vs. success rate of conditional sign flip gate



III. Gate synthesis with adaptive circuit compression⁷

Gate synthesis problem: Given a unitary matrix, find a corresponding circuit implementing it!

Existing solution: Gradient-based optimization with universal layers.

Using  +  OpenXLA \implies $\sim 600\times$ speedup!⁶

⁶With 25 CVQNN layers and Fock space cutoff 20, averaged from 1000 iterations.

⁷Henrik Varga, “Decomposition of unitary matrices based on bosonic Hamiltonian operators into elementary photonic quantum gates” [unpublished]

III. Gate synthesis with adaptive circuit compression⁷

Gate synthesis problem: Given a unitary matrix, find a corresponding circuit implementing it!

Existing solution: Gradient-based optimization with universal layers.

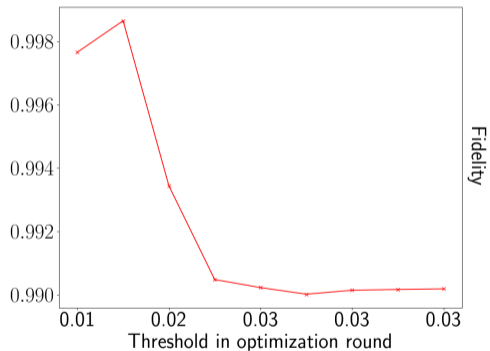
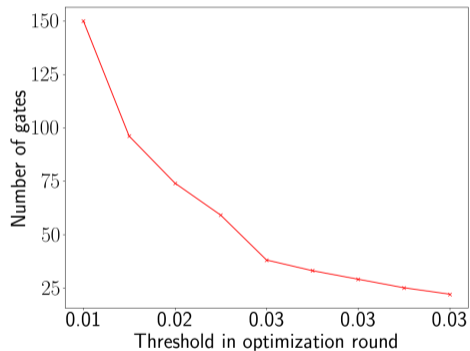
Using  +  OpenXLA \implies $\sim 600\times$ speedup!⁶

This speedup allows us to **compress** the circuit adaptively, by “throwing” out gates close to identity.

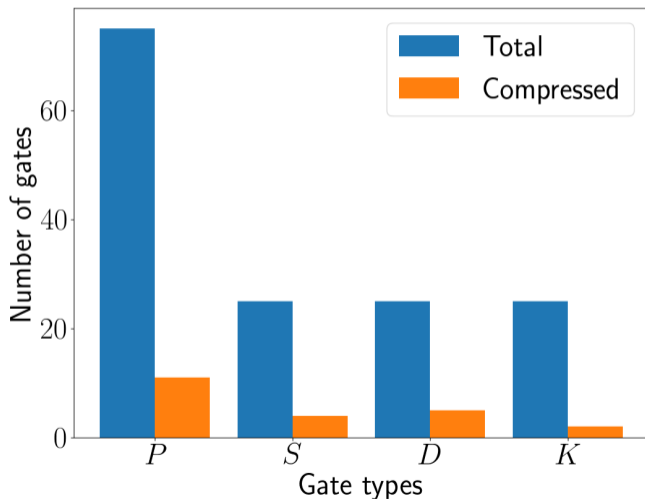
⁶With 25 CVQNN layers and Fock space cutoff 20, averaged from 1000 iterations.

⁷Henrik Varga, “Decomposition of unitary matrices based on bosonic Hamiltonian operators into elementary photonic quantum gates” [unpublished]

III. Decomposing the cubic phase gate with compression



III. Decomposing the cubic phase gate with compression



Thank you for your attention!

This research was supported by the Ministry of Culture and Innovation and the National Research, Development and Innovation Office within the Quantum Information National Laboratory of Hungary (Grant No. 2022-2.1.1-NL-2022-00004).



ELTE
EÖTVÖS LORÁND
UNIVERSITY

HUN
REN



WIGNER



Quantum Information
National Laboratory
HUNGARY



PIQUASSO