PIQUASSO

A Photonic Quantum Computer Simulation Software Platform

Zoltán Kolarovszki, Tomasz Rybotycki, Péter Rakyta, Ágoston Kaposi, Boldizsár Poór, Szabolcs Jóczik, Dániel T. R. Nagy, Henrik Varga, Kareem H. El-Safty, Gregory Morse, Michał Oszmaniec, Tamás Kozsik, Zoltán Zimborás

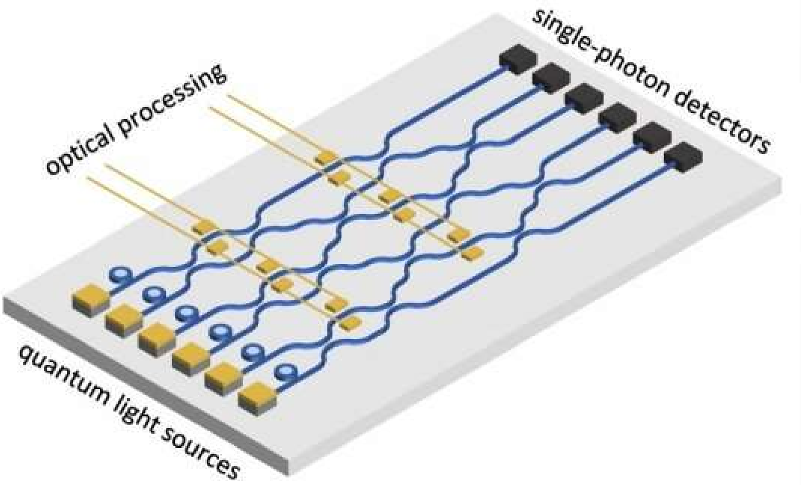
HUN-REN Wigner Research Centre for Physics
Eötvös Loránd University

June 19th, 2024

ReAQCT 2024

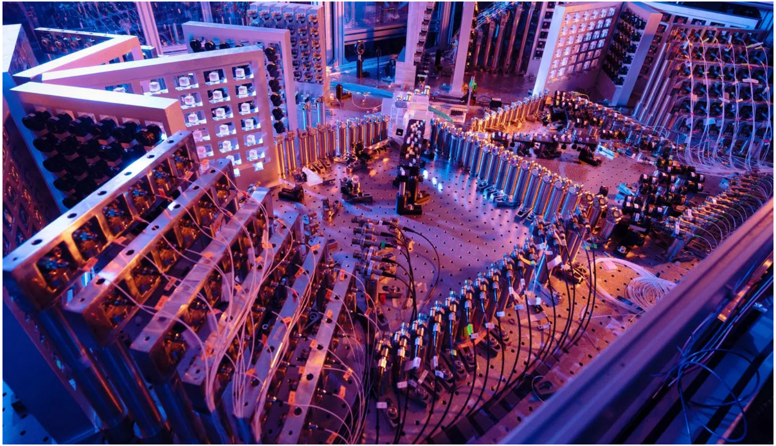
Photonic Quantum Computing

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



Quantum advantage by USTC

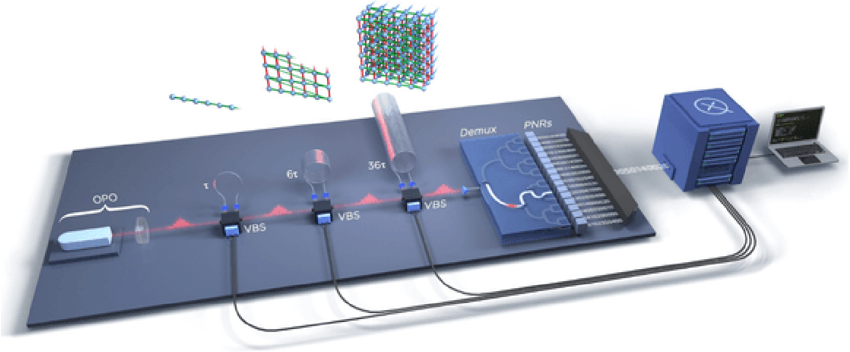
The Quantum Information Group of USTC in Hefei (led by Jian-Wei Pan) demonstrated an advantage over classical computation in 2020 (with improvements in 2021 and 2023, with the latter mentioning our method as a classical benchmark).



PIQUASSO

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

PIQUASSO: PhotonIc QUAntum computer Simulator SOftware

We are developing and maintaining a new simulator framework written in Python called **Piquasso**[1].

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 GitHub[2] and can be installed with:

```
pip install piquasso
```

[1] ZK et al., arXiv:2403.04006

[2] https://github.com/Budapest-Quantum-Computing-Group/piquasso

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.

PIQUASSO

Usage example

```python
simulator = pq.GaussianSimulator(d=5)

with pq.Program() as program:
    pq.Q(all) | pq.Vacuum()

    for i in range(5):
        pq.Q(i) | pq.Squeezing(r=0.1) | pq.Displacement(r=1.0)

    pq.Q(0,1) | pq.Beamsplitter(theta=np.pi / 3)
    pq.Q(2,3) | pq.Beamsplitter(theta=np.pi / 4)
    pq.Q(3,4) | pq.Beamsplitter(theta=np.pi / 5)

    pq.Q(all) | pq.ParticleNumberMeasurement()

result = simulator.execute(program, shots=1000)
print(result.samples)
# [(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 JAX for calculations (with OpenXLA support).

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:

$$c^d, \qquad c: \ \textbf{local} \text{ cutoff}, \ d: \ \text{number of modes.} \tag{1}$$

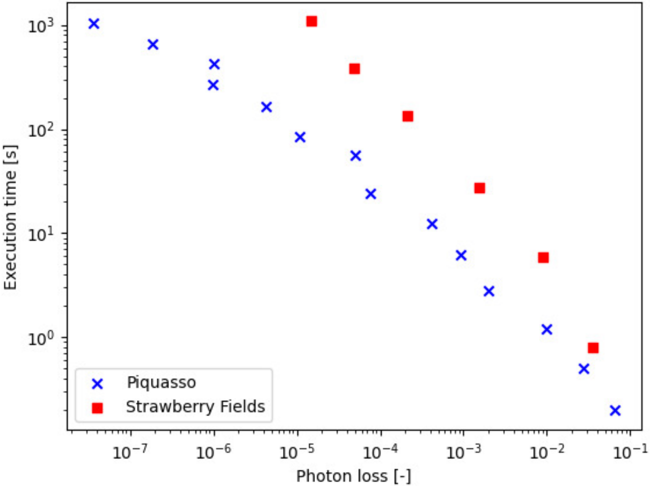
PIQUASSO: **Global cutoff**
Constraint on particle number on the **whole system**. State vector size:

$$\binom{d+c-1}{c-1}, \qquad c: \ \textbf{global} \text{ cutoff}, \ d: \ \text{number of modes.} \tag{2}$$

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

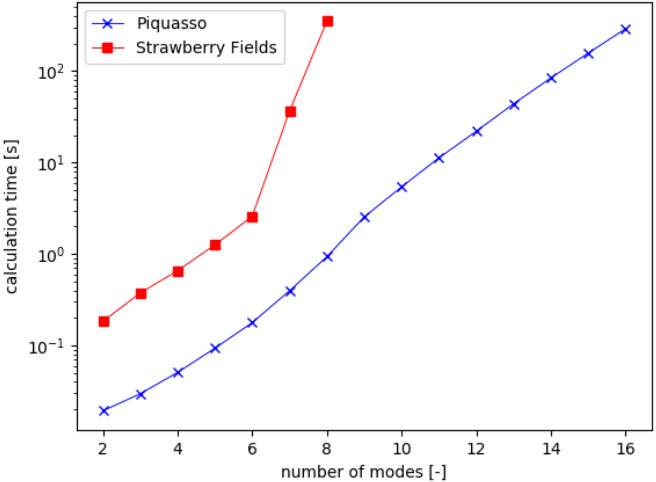
PIQUASSO

Photon losses[3]: PIQUASSO vs. STRAWBERRY FIELDS



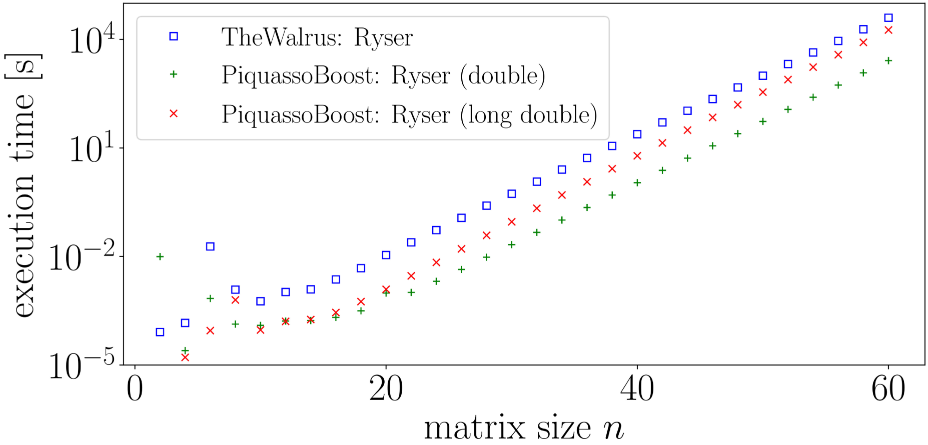
[3] In a circuit using 4 continuous-variable quantum neural network (CVQNN) layers on 8 modes.

Automatic differentiation[4]: PIQUASSO vs. STRAWBERRY FIELDS



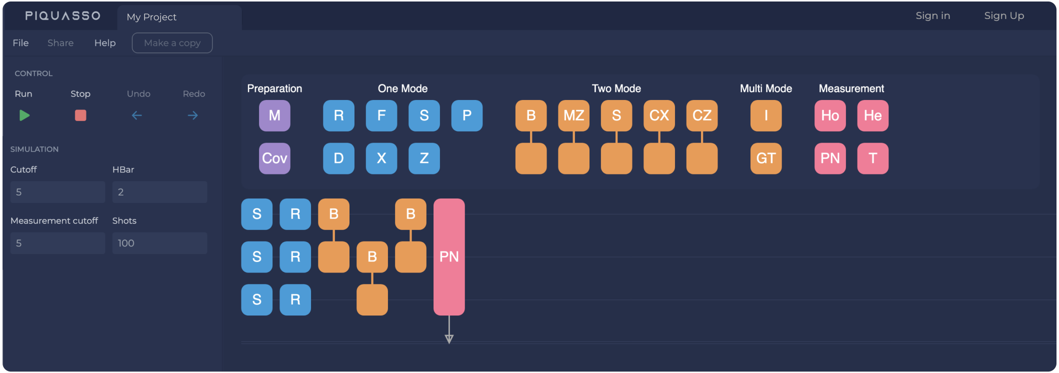
[4] In a circuit using 4 CVQNN layers with cutoff 10.

Hafnian calculation speedup in PiquassoBoost



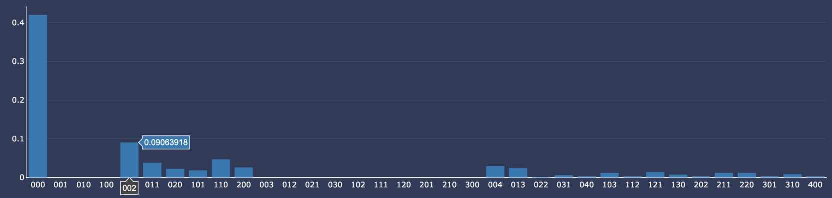
PIQUASSO Composer

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



Particle Number Detection Probabilities

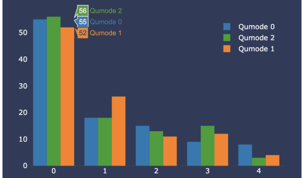
Export Data Export Image



Samples

Export Data Export Image

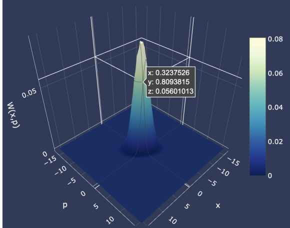
Qumode 0 Qumode 1 Qumode 2



Wigner Function

Export Image

Qumode 0 Qumode 1 Qumode 2



Python Code

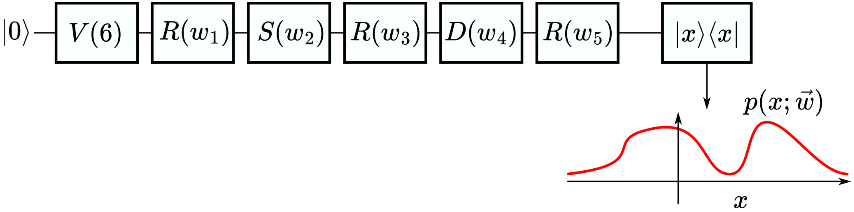
Copy to clipboard

```python
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)[5]

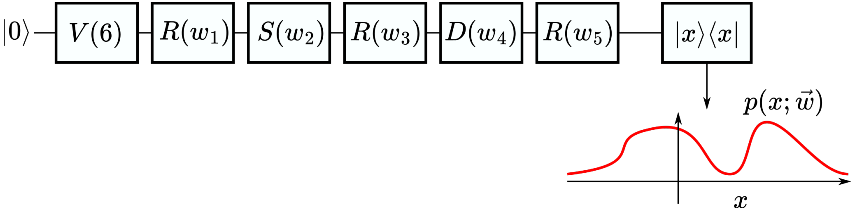
- **Born machine:** Parametrized quantum circuit producing probability distributions via **Born's rule** $\Longrightarrow$ generative quantum machine learning
- **CVBM:** Photonic quantum circuits can generate **position distribution**



[5] ZK, D. T. R. Nagy, Z. Zimborás, 2024, "On the learning abilities of photonic continuous-variable Born machines" [submitted]

I. Training continuous-variable Born machines (CVBMs)[5]

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



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

[5] ZK, D. T. R. Nagy, Z. Zimborás, 2024, "On the learning abilities of photonic continuous-variable Born machines" [submitted]

I. Training continuous-variable Born machines (CVBMs)[5]

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

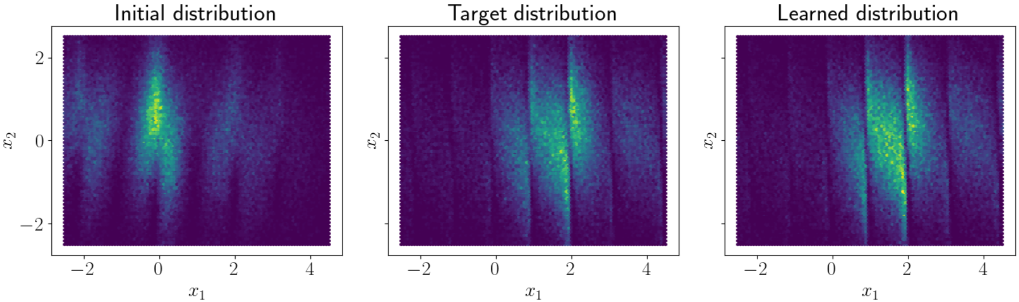
PIQUASSO uses a new classical algorithm for homodyne measurement tailored for **multimode systems**.

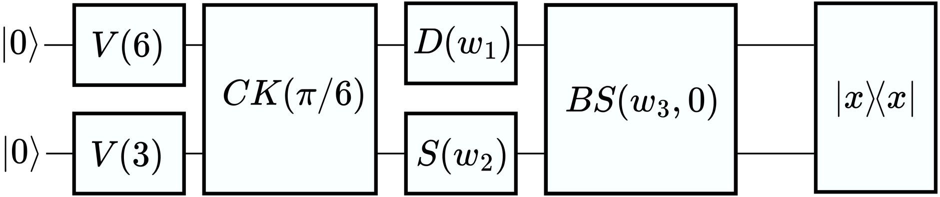
$\sim$ 1000$\times$ speedup over previous solutions!

[5] ZK, D. T. R. Nagy, Z. Zimborás, 2024, "On the learning abilities of photonic continuous-variable Born machines" [submitted]

PIQUASSO

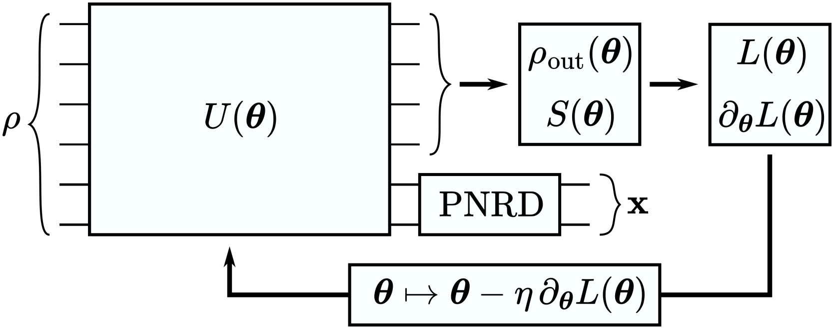
I. Efficient training of multimode CVBMs with PIQUASSO





II. Optimizing non-deterministic gates with imperfect detections[6]

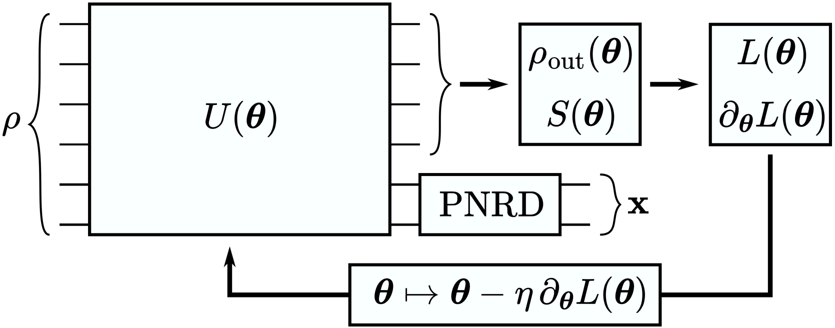
Nondeterministic gates can implement **qubit** gates on a photonic quantum computer.



[6] C. Czabán, ZK, M. Karácsony, Z. Zimborás, 2024, "Suppressing photon detection errors in nondeterministic state preparation", ReAQCT'24

II. Optimizing non-deterministic gates with imperfect detections[6]

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

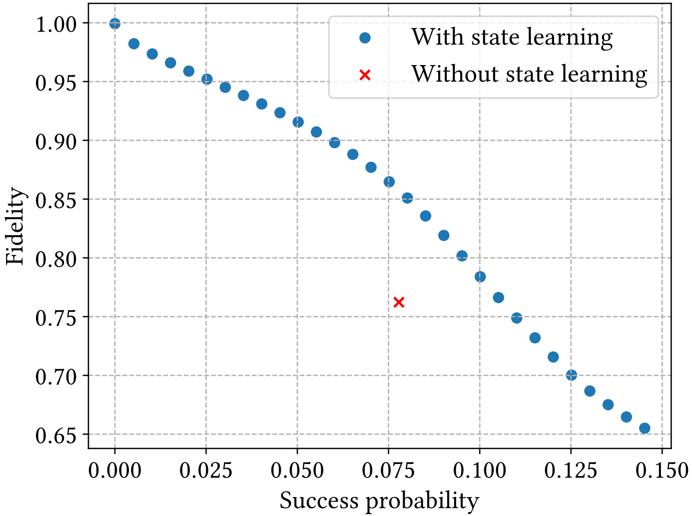


Problem: Photon detectors have biases.

[6] C. Czabán, ZK, M. Karácsony, Z. Zimborás, 2024, "Suppressing photon detection errors in nondeterministic state preparation", ReAQCT'24

PIQUASSO

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



III. Gate synthesis with adaptive circuit compression[8]

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

Existing solution: Gradient-based optimization with universal layers.

Using JAX + OpenXLA $\Longrightarrow$ $\sim 600\times$ speedup![7]

[7]With 25 CVQNN layers and Fock space cutoff 20, averaged from 1000 iterations.
[8]H. Varga, 2024, "Decomposition of unitary matrices based on bosonic Hamiltonian operators into elementary photonic quantum gates" [unpublished]

PIQUASSO

III. Gate synthesis with adaptive circuit compression[8]

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

Existing solution: Gradient-based optimization with universal layers.

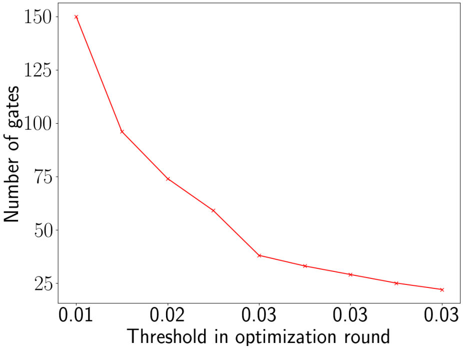
Using + OpenXLA $\Longrightarrow\ \sim 600\times$ speedup![7]

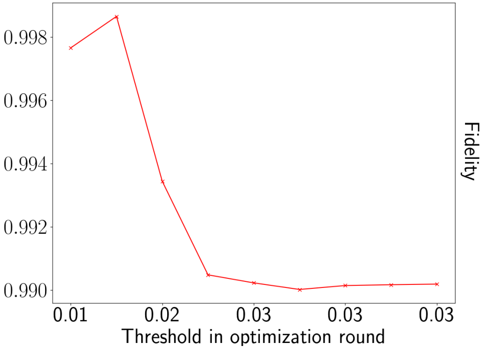
This speedup allows us to **compress** the circuit adaptively, by "throwing" out gates close to identity.

[7]With 25 CVQNN layers and Fock space cutoff 20, averaged from 1000 iterations.

[8]H. Varga, 2024, "Decomposition of unitary matrices based on bosonic Hamiltonian operators into elementary photonic quantum gates" [unpublished]

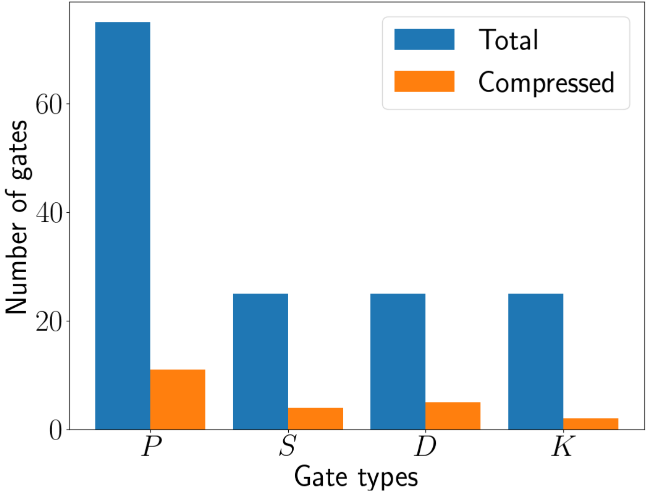
III. Decomposing the cubic phase gate with compression





PIQUASSO

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).