Qiskit-symb: a Qiskit Ecosystem package for symbolic Quantum Computation

International Symposium on Grids and Clouds 2025

Simone Gasperini
PhD student in Data Science and Computation
University of Bologna and INFN Bologna, Italy

20 March 2025
Academia Sinica
Taipei, Taiwan

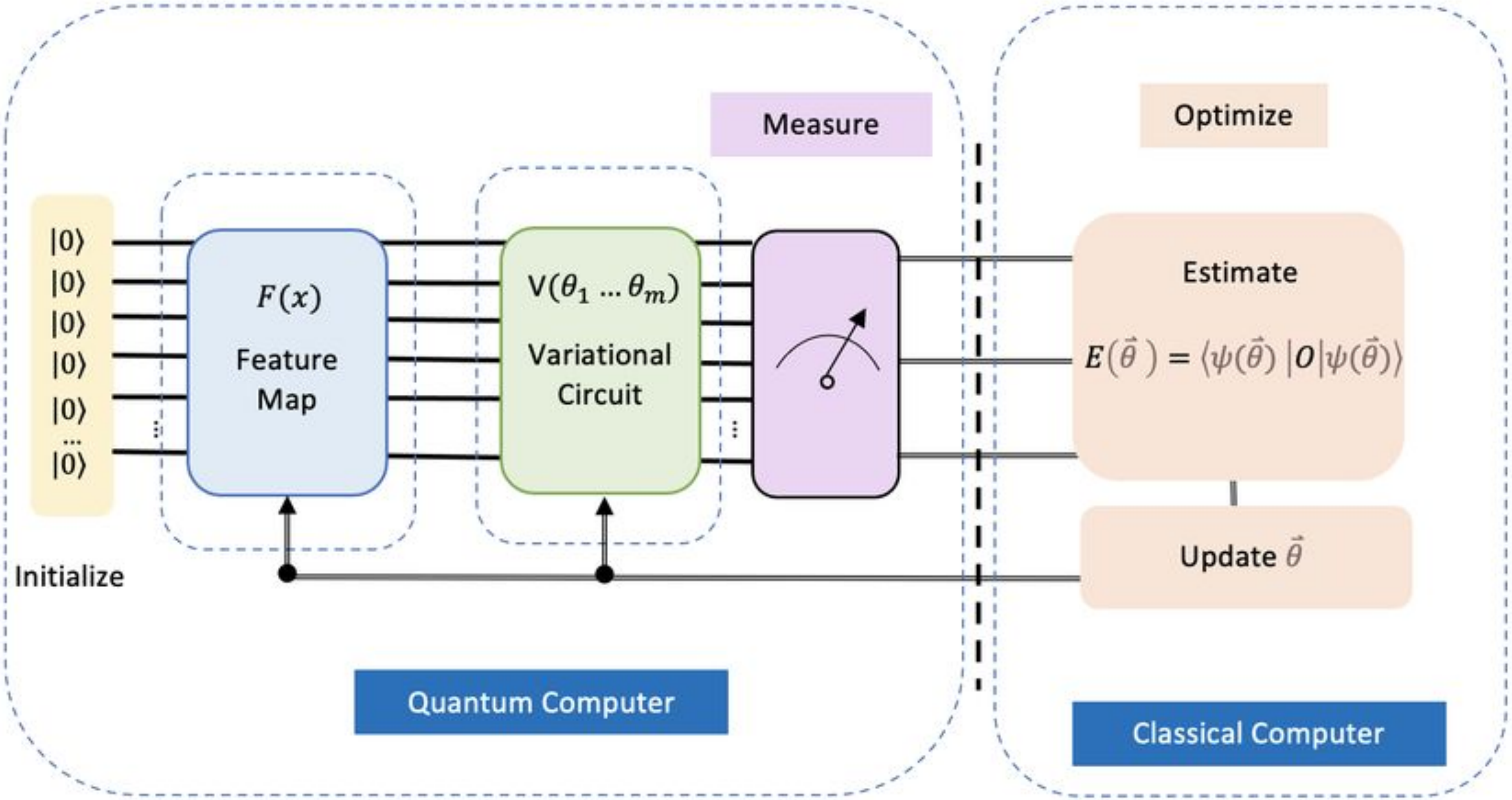
Outline

- Overview of state-of-the-art frameworks for Quantum Machine Learning (QML):
    - Quantum Neural Network
    - Quantum Kernel methods

- Introduction to the **Qiskit Ecosystem** and Qiskit SDK

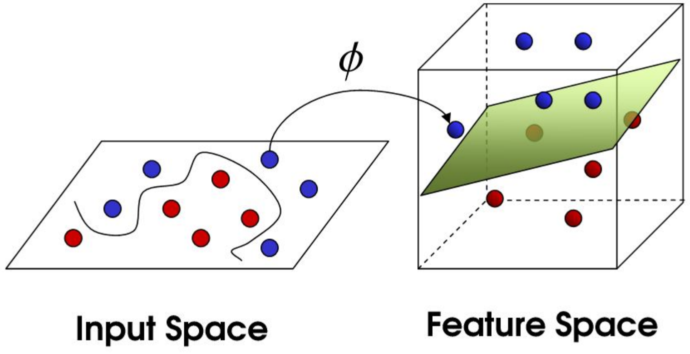
- **Qiskit-symb** for symbolic Quantum Computation in Qiskit
- **Qiskit-symb** for Parameterized Quantum Circuits (PQCs) simulation in QML

Quantum Neural Network



Kernel methods

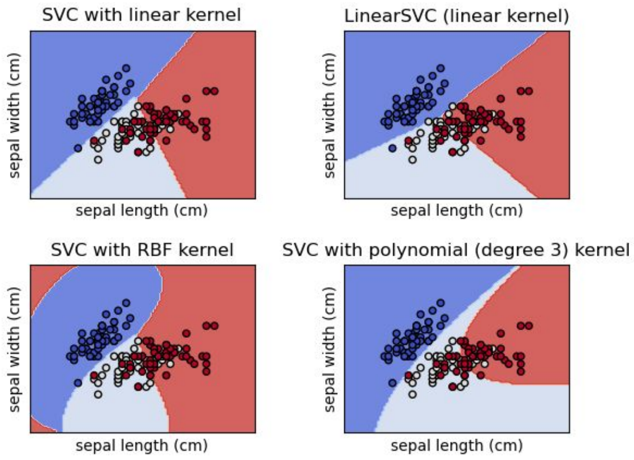
Consider a supervised classification task with complex decision boundaries in the **input space**. A function $\boldsymbol{\phi}$ can be used to map each data point in a higher-dimensional **feature space** with simple decision boundaries.



$$K(x_i, x_j) = \phi^T(x_i) \cdot \phi(x_j)$$

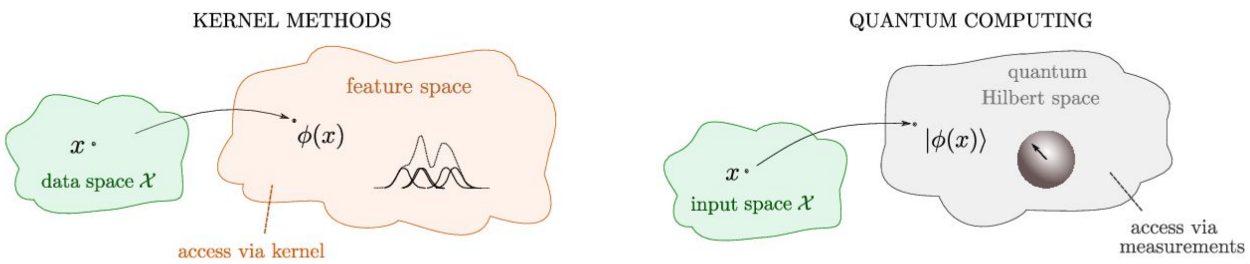
Kernel trick: it's possible to compute the kernel without having to calculate or even know anything about $\boldsymbol{\phi}$

Supervised classification using **SVM model**



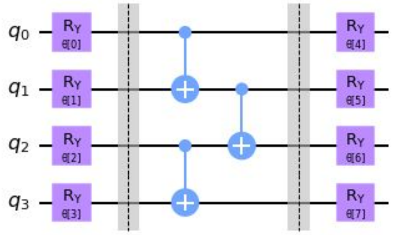
⚠ Computational complexity $\mathrm{O}(\mathrm{N}^2)$, with $\mathrm{N} =$ number of data points

Quantum Kernel



$$|\phi(x)\rangle = U_{\phi(x)}|0^{\otimes n}\rangle$$

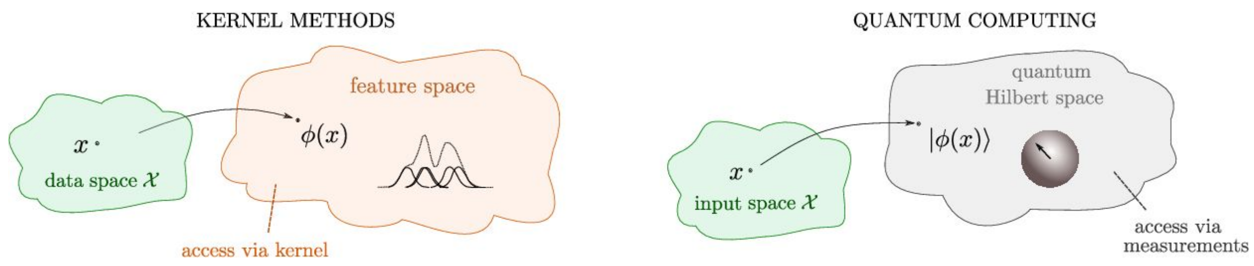
Quantum **feature map** as a PQC



$$K(x_i, x_j) = |\langle \phi^\dagger(x_i)|\phi(x_j)\rangle|^2$$
$$= |\langle 0^{\otimes n}|U^\dagger_{\phi(x_i)} U_{\phi(x_j)}|0^{\otimes n}\rangle|^2$$

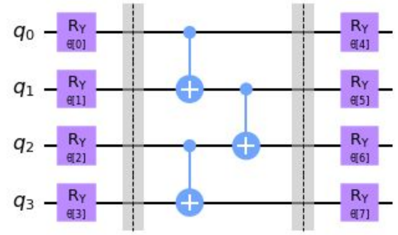
The kernel function value is computed as the probability of measuring the state $|0^{\otimes n}\rangle$ at the end of the circuit

Quantum Kernel



$$|\phi(x)\rangle = U_{\phi(x)}|0^{\otimes n}\rangle$$

Quantum **feature map** as a PQC



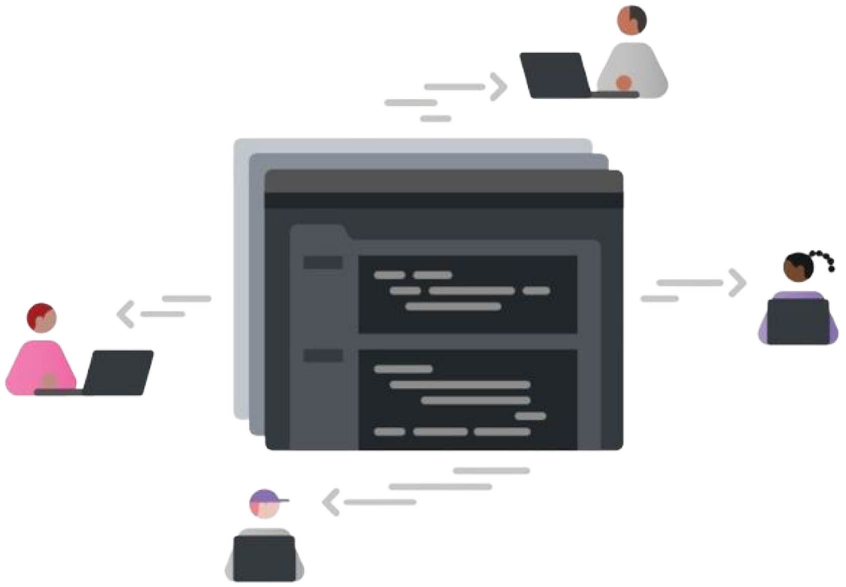
$$K(x_i, x_j) = |\langle \phi^\dagger(x_i)|\phi(x_j)\rangle|^2$$
$$= |\langle 0^{\otimes n}|U^\dagger_{\phi(x_i)} U_{\phi(x_j)}|0^{\otimes n}\rangle|^2$$

The kernel function value is computed as the probability of measuring the state $|0^{\otimes n}\rangle$ at the end of the circuit

$O(N^2)$ executions of the same PQC with different parameters values!

Qiskit ecosystem

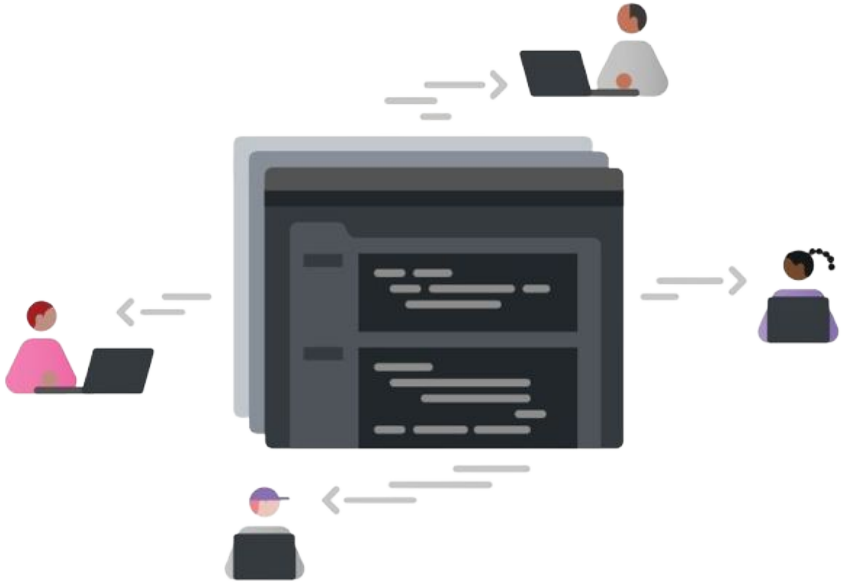
The Qiskit ecosystem is a collection of tools created by researchers and developers who use Qiskit every day



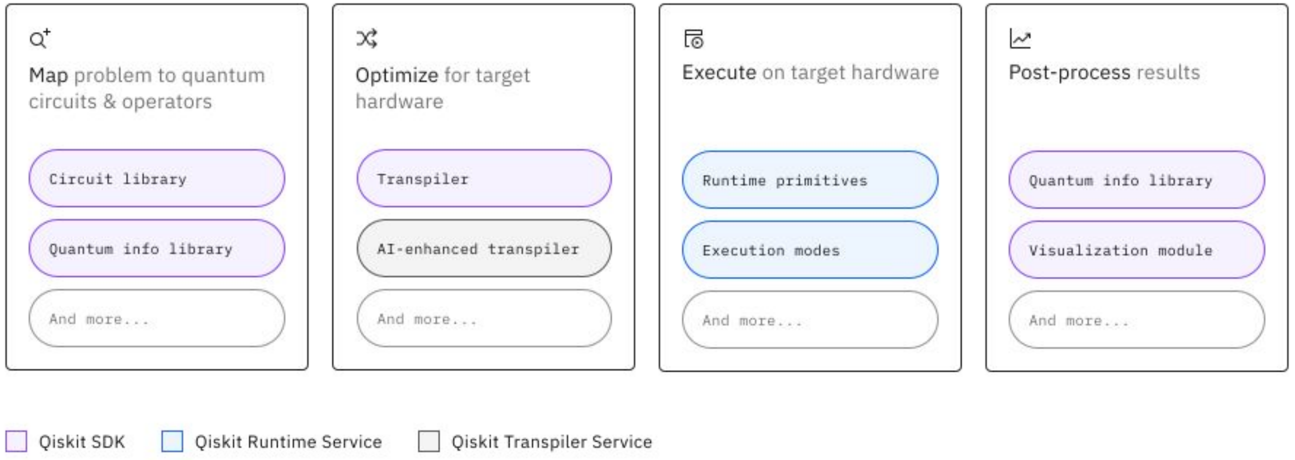
https://www.ibm.com/quantum/ecosystem

Qiskit ecosystem

The Qiskit ecosystem is a collection of tools created by researchers and developers who use Qiskit every day

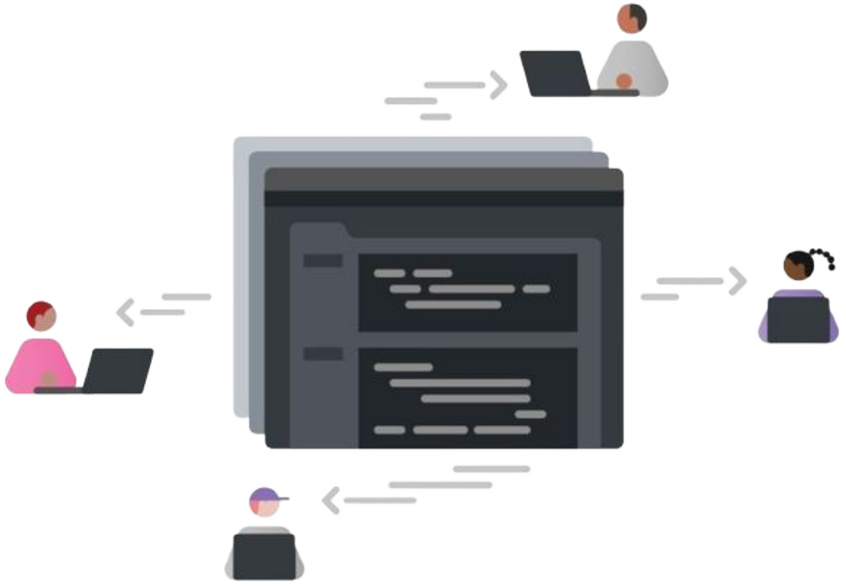


https://www.ibm.com/quantum/ecosystem

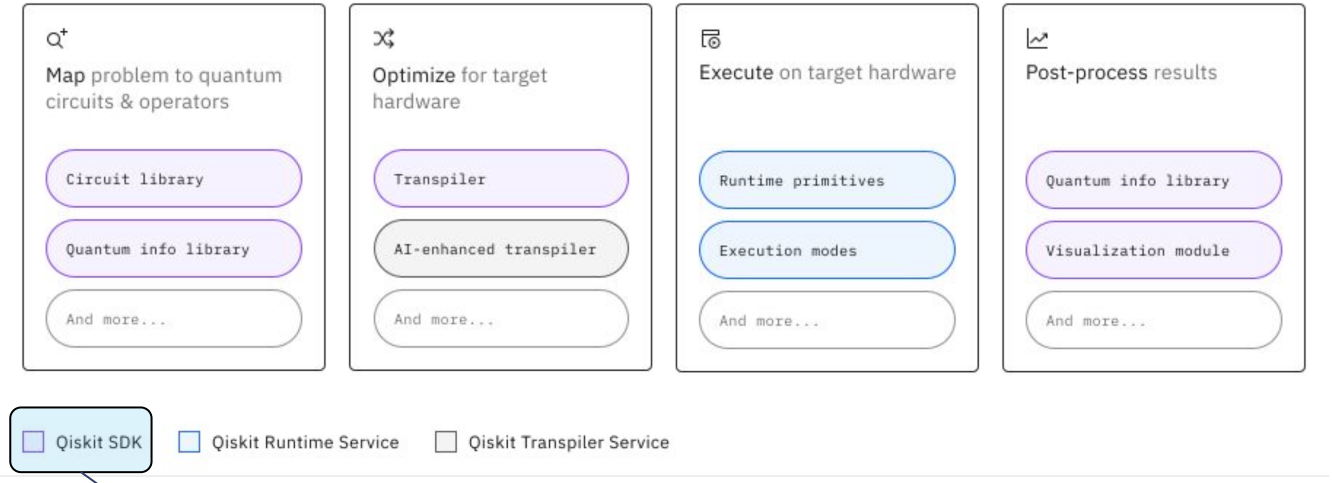


Qiskit ecosystem

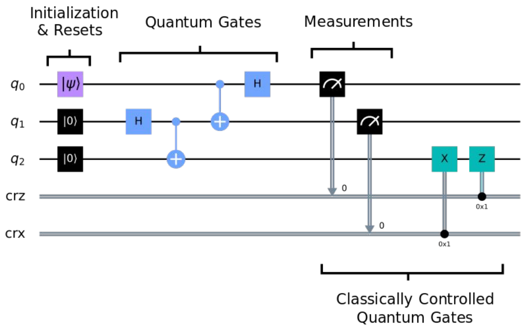
The Qiskit ecosystem is a collection of tools created by researchers and developers who use Qiskit every day

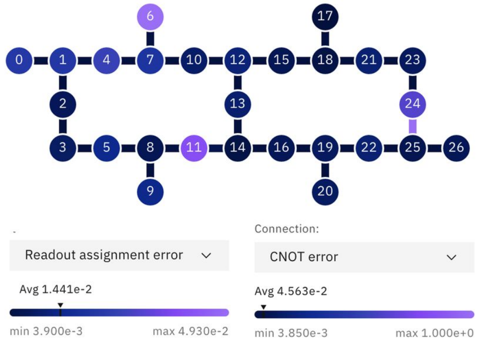


https://www.ibm.com/quantum/ecosystem



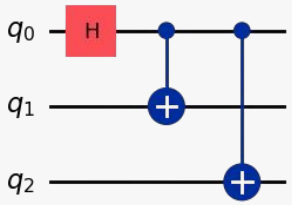
"The Qiskit SDK is an open-source SDK for working with quantum computers at the level of extended (static, dynamic, and scheduled) quantum circuits, operators, and primitives"





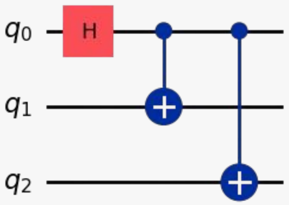
Quantum info library

```python
from qiskit import QuantumCircuit

qc = QuantumCircuit(3)
qc.h(0)
qc.cx(0, 1)
qc.cx(0, 2)

qc.draw('mpl')
```



Quantum info library

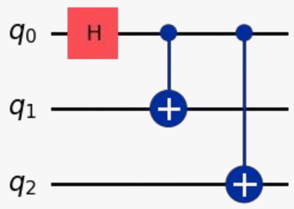
```python
from qiskit import QuantumCircuit

qc = QuantumCircuit(3)
qc.h(0)
qc.cx(0, 1)
qc.cx(0, 2)

qc.draw('mpl')
```




```python
from qiskit.quantum_info import Statevector

psi = Statevector(qc)
```

Quantum info library

```python
from qiskit import QuantumCircuit

qc = QuantumCircuit(3)
qc.h(0)
qc.cx(0, 1)
qc.cx(0, 2)

qc.draw('mpl')
```

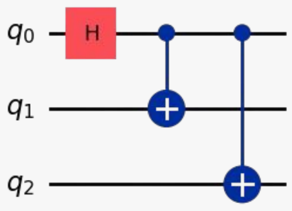



```python
from qiskit.quantum_info import Statevector

psi = Statevector(qc)
```

$$
\begin{aligned}
|\psi\rangle &= \frac{1}{\sqrt{2}}(|000\rangle + |111\rangle) \\
&= \begin{bmatrix} \frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} \end{bmatrix}
\end{aligned}
$$

Quantum info library

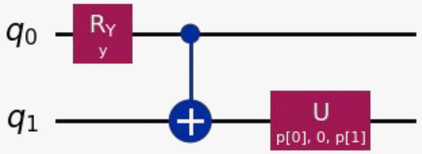
```python
from qiskit import QuantumCircuit

qc = QuantumCircuit(3)
qc.h(0)
qc.cx(0, 1)
qc.cx(0, 2)

qc.draw('mpl')
```




```python
from qiskit.quantum_info import Statevector

psi = Statevector(qc)
```

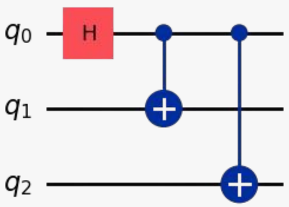
$$|\psi\rangle = \frac{1}{\sqrt{2}}(|000\rangle + |111\rangle)$$
$$= \begin{bmatrix} \frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} \end{bmatrix}$$

```python
from qiskit import QuantumCircuit
from qiskit.circuit import Parameter, ParameterVector

y = Parameter('y')
p = ParameterVector('p', length=2)

pqc = QuantumCircuit(2)
pqc.ry(y, 0)
pqc.cx(0, 1)
pqc.u(p[0], 0, p[1], 1)

pqc.draw('mpl')
```



Quantum info library

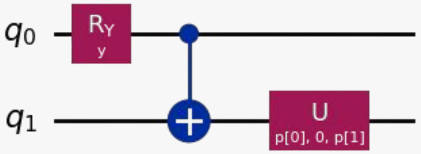
```python
from qiskit import QuantumCircuit

qc = QuantumCircuit(3)
qc.h(0)
qc.cx(0, 1)
qc.cx(0, 2)

qc.draw('mpl')
```




```python
from qiskit.quantum_info import Statevector

psi = Statevector(qc)
```

$$
\begin{aligned}
|\psi\rangle &= \frac{1}{\sqrt{2}}(|000\rangle + |111\rangle) \\
&= \begin{bmatrix} \frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} \end{bmatrix}
\end{aligned}
$$

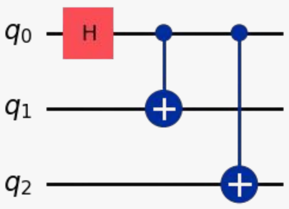
```python
from qiskit import QuantumCircuit
from qiskit.circuit import Parameter, ParameterVector

y = Parameter('y')
p = ParameterVector('p', length=2)

pqc = QuantumCircuit(2)
pqc.ry(y, 0)
pqc.cx(0, 1)
pqc.u(p[0], 0, p[1], 1)

pqc.draw('mpl')
```




```python
from qiskit.quantum_info import Statevector

psi = Statevector(pqc)
```

Quantum info library

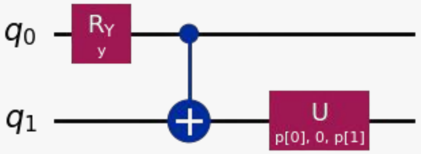
```python
from qiskit import QuantumCircuit

qc = QuantumCircuit(3)
qc.h(0)
qc.cx(0, 1)
qc.cx(0, 2)

qc.draw('mpl')
```




```python
from qiskit.quantum_info import Statevector

psi = Statevector(qc)
```

$$|\psi\rangle = \frac{1}{\sqrt{2}}(|000\rangle + |111\rangle)$$
$$= \begin{bmatrix} \frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} \end{bmatrix}$$

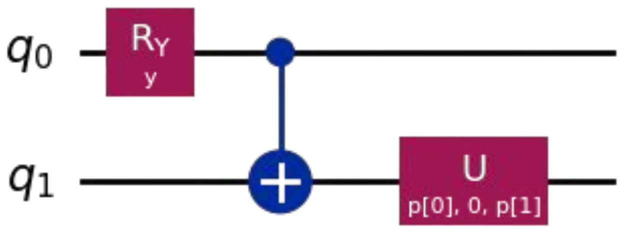
```python
from qiskit import QuantumCircuit
from qiskit.circuit import Parameter, ParameterVector

y = Parameter('y')
p = ParameterVector('p', length=2)

pqc = QuantumCircuit(2)
pqc.ry(y, 0)
pqc.cx(0, 1)
pqc.u(p[0], 0, p[1], 1)

pqc.draw('mpl')
```




```python
from qiskit.quantum_info import Statevector

psi = Statevector(pqc)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-7-032f9a8c4db1> in <cell line: 0>()
      1 from qiskit.quantum_info import Statevector
      2
----> 3 psi = Statevector(pqc)

7 frames
/usr/local/lib/python3.11/dist-packages/qiskit/circuit/parameterexpression.py in __float__(self)
    549         except (TypeError, RuntimeError) as exc:
    550             if self.parameters:
--> 551                 raise TypeError(
    552                     f"ParameterExpression with unbound parameters ({self.parameters}) "
    553                     "cannot be cast to a float."

TypeError: ParameterExpression with unbound parameters (dict_keys([Parameter(y)])) cannot be cast to a float.
```

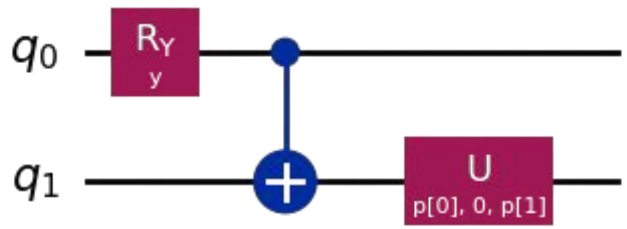
Qiskit-symb solution




```python
from qiskit_symb.quantum_info import Statevector

psi = Statevector(pqc)
psi.to_sympy()
```

$$\left[\cos\left(\frac{p[0]}{2}\right)\cos\left(\frac{y}{2}\right) \quad -e^{1.0ip[1]}\sin\left(\frac{p[0]}{2}\right)\sin\left(\frac{y}{2}\right) \quad \sin\left(\frac{p[0]}{2}\right)\cos\left(\frac{y}{2}\right) \quad e^{1.0ip[1]}\sin\left(\frac{y}{2}\right)\cos\left(\frac{p[0]}{2}\right)\right]$$

Qiskit-symb solution



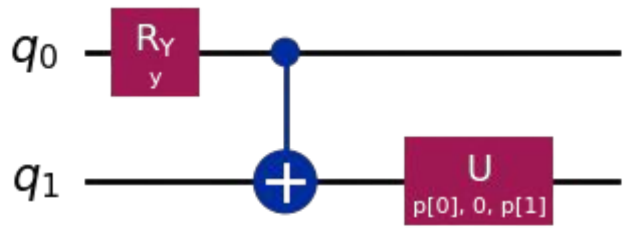

```python
from qiskit_symb.quantum_info import Statevector

psi = Statevector(pqc)
psi.to_sympy()
```

$$\left[\cos\left(\frac{p[0]}{2}\right)\cos\left(\frac{y}{2}\right) \quad -e^{1.0ip[1]}\sin\left(\frac{p[0]}{2}\right)\sin\left(\frac{y}{2}\right) \quad \sin\left(\frac{p[0]}{2}\right)\cos\left(\frac{y}{2}\right) \quad e^{1.0ip[1]}\sin\left(\frac{y}{2}\right)\cos\left(\frac{p[0]}{2}\right)\right]$$

Consider a fixed PQC you need to execute many times with a different set of parameters values at each execution. You can use **qiskit-symb** to perform the (symbolic) linear algebra evaluation just once!

```python
from qiskit_symb.quantum_info import Statevector

psi = Statevector(pqc)
sim = psi.to_lambda()
```

Qiskit-symb solution



Qiskit symbolic

```python
from qiskit_symb.quantum_info import Statevector

psi = Statevector(pqc)
psi.to_sympy()
```

$$\left[\cos\left(\frac{p[0]}{2}\right)\cos\left(\frac{y}{2}\right) \quad -e^{1.0ip[1]}\sin\left(\frac{p[0]}{2}\right)\sin\left(\frac{y}{2}\right) \quad \sin\left(\frac{p[0]}{2}\right)\cos\left(\frac{y}{2}\right) \quad e^{1.0ip[1]}\sin\left(\frac{y}{2}\right)\cos\left(\frac{p[0]}{2}\right)\right]$$

But… There is much more!

Consider a fixed PQC you need to execute many times with a different set of parameters values at each execution. You can use **qiskit-symb** to perform the (symbolic) linear algebra evaluation just once!

```python
from qiskit_symb.quantum_info import Statevector

psi = Statevector(pqc)
sim = psi.to_lambda()
```

```python
import numpy as np

sim(np.pi/2, 0, 0)
```

```
array([ 0.70710678+0.j, -0.        +0.j,  0.70710678+0.j,  0.        +0.j])
```

Call Python lambda function to perform full-statevector simulation

Qiskit-symb performance

qiskit-symb time = **sympy eval** + **sympy lambdify** + **python run**

```
psi = Statevector(pqc)
```

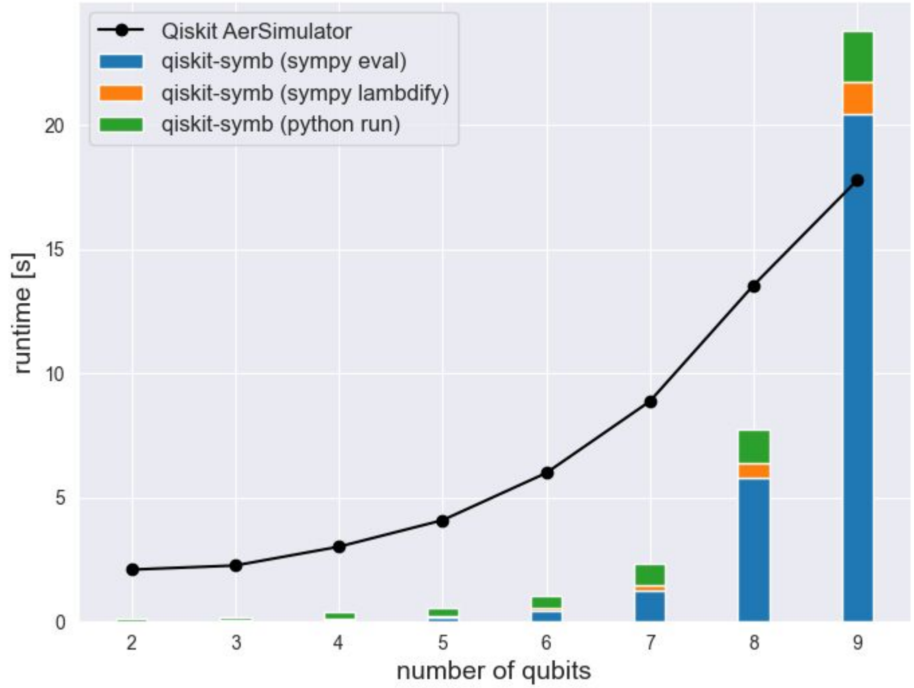
```
sim = psi.to_lambda()
```

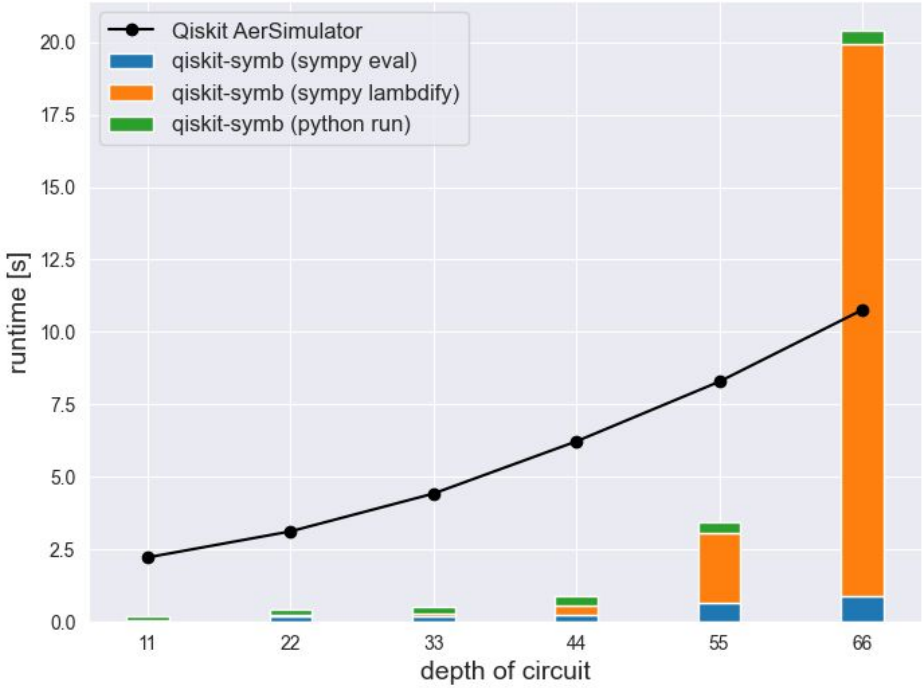
```
sim(*params)
```

Qiskit-symb performance

qiskit-symb time = **sympy eval** + **sympy lambdify** + **python run**

`psi = Statevector(pqc)` `sim = psi.to_lambda()` `sim(*params)`





The PQC used for the performance benchmark study is the `ZZFeatureMap` *provided by Qiskit. Each runtime refers to 10k executions of the same PQC with different random values assigned to the parameters at each execution.*

```
qiskit==1.2.4
qiskit-aer==0.15.1
qiskit-symb==0.4.0
```

Thank you!

Qiskit
symbolic

qiskit-symb Public

Symbolic evaluation of parameterized quantum circuits in Qiskit

GitHub

quantum-computing sympy symbolic-computation qiskit

Python 33 2 Apache License 2.0 2 issues need help Updated 2 weeks ago

bitly