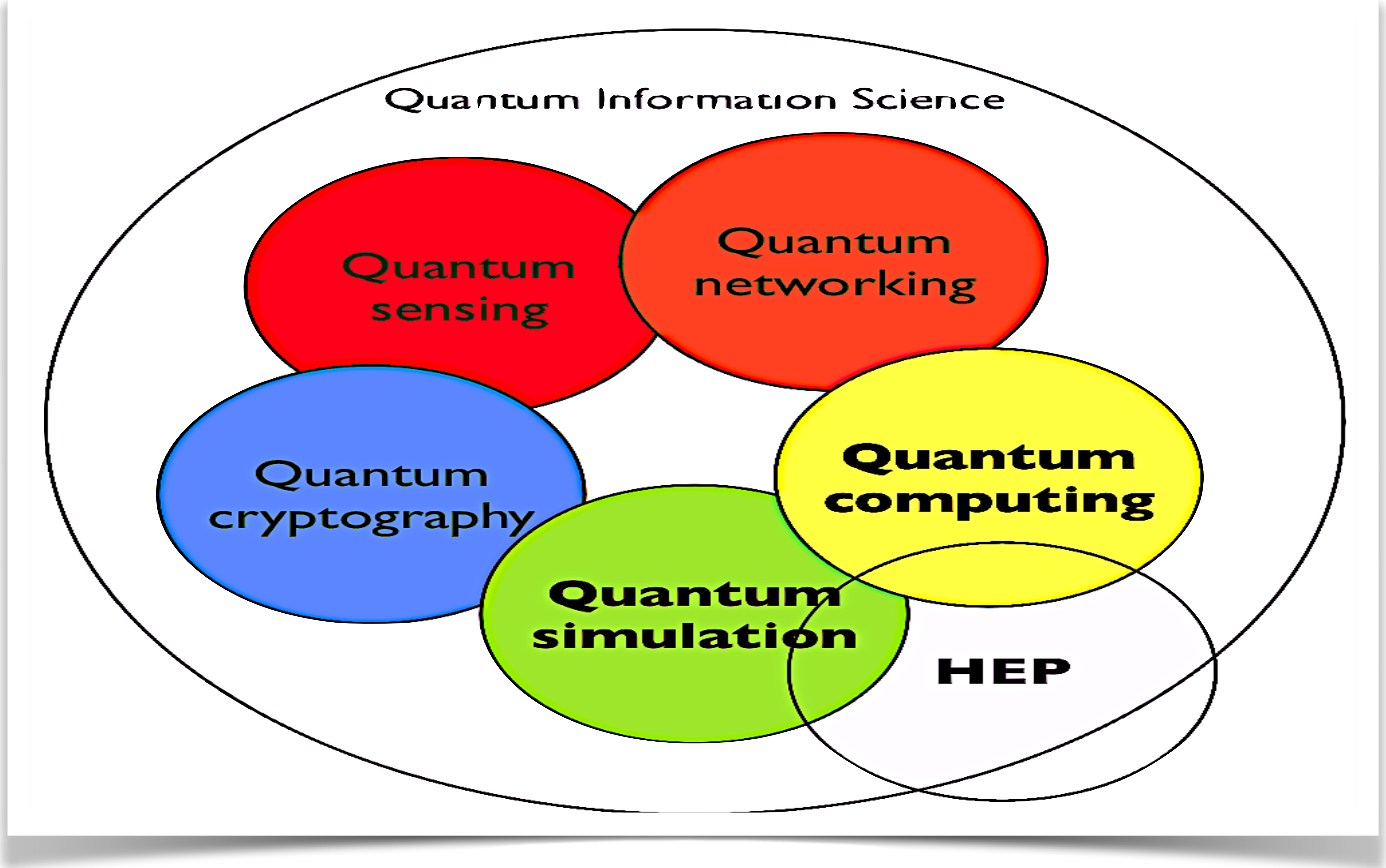
Introduction to quantum machine learning and its future application in High Energy Physics Problems

Abdualazem Fadol Mohammed

February 2, 2024

Introduction

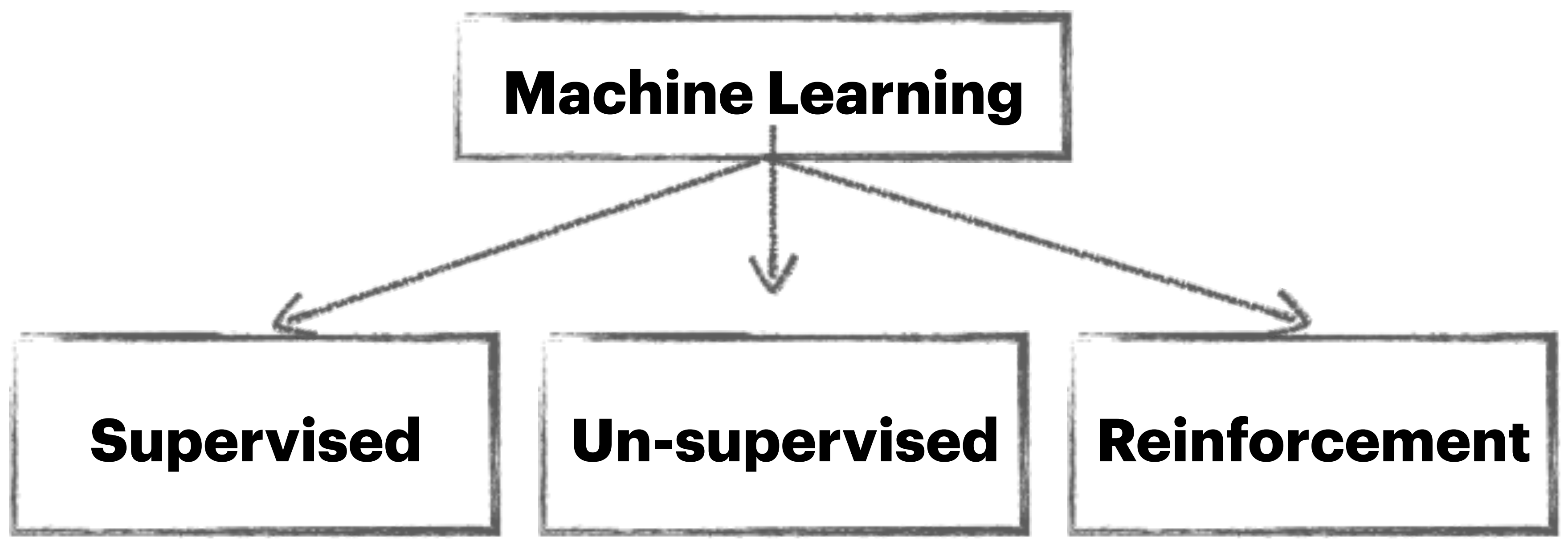
- Machine learning has blossomed in recent decades and has become essential in many fields.
- In High Energy Physics, machine learning plays a significant role in solving or improving problems such as particle identification.
- However, the introduction of quantum computers opened uncharted territories for exploration.
- There is a strong belief that quantum computing will change the computing world as we know it.
- **In several computing aspects:**
  - Improving the computing performance in general
    - Exponential speed up data pattern recognition, etc …
  - Massive data warehouse, exponential, through entanglement.



- Companies such as Google and IBM are committed to accelerating the development of quantum technology.

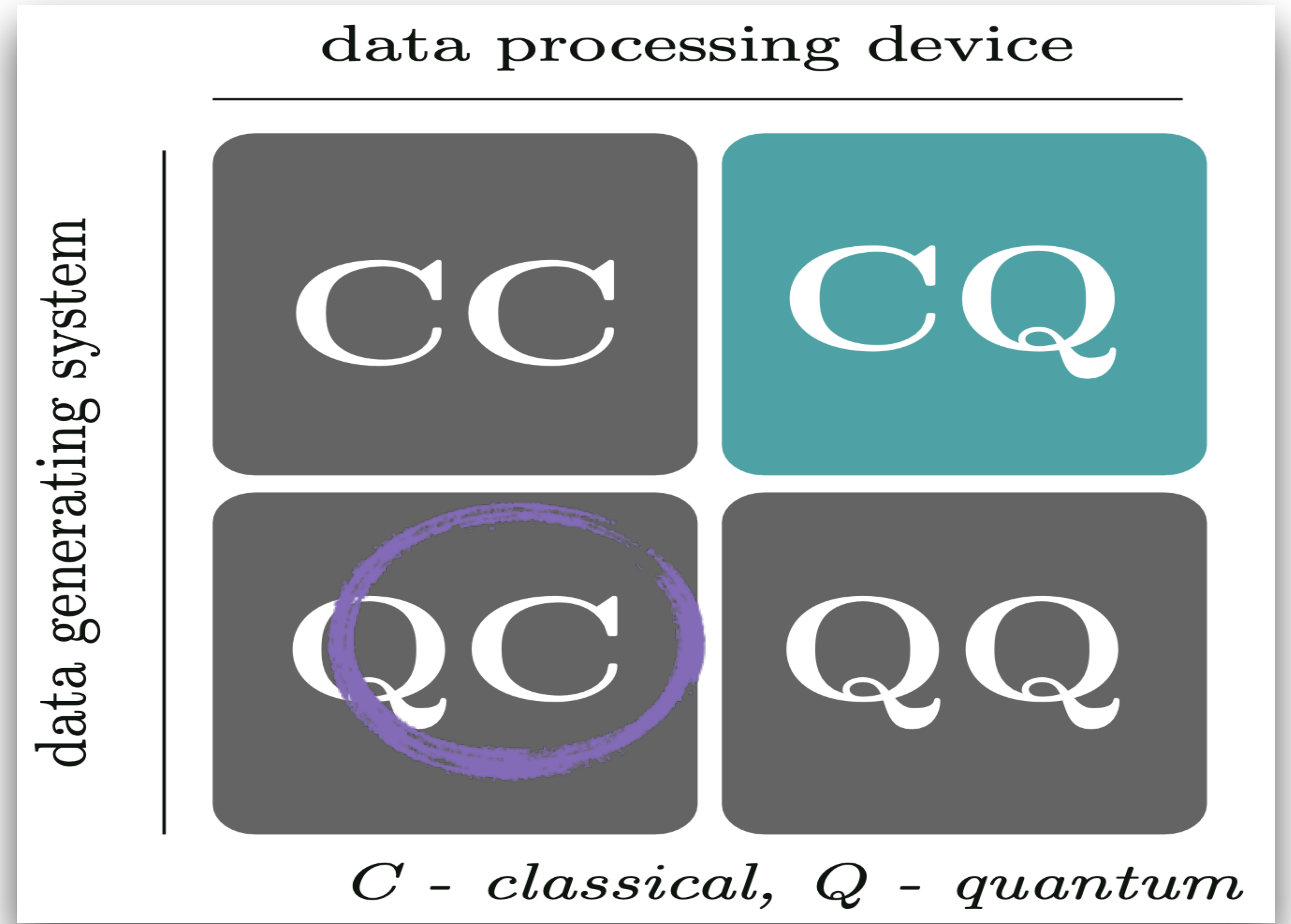
Introduction

- Classical Machine learning seeks to find patterns in data.



- Quantum Machine Learning: Based on the following approaches:
  - Weather data is classical (C) or quantum (Q), and
  - an algorithm runs in a classical (C) or quantum (Q) computer
- Quantum algorithms—an equivalent to classical algorithms:
  - Grover search and amplitude amplification
  - Hybrid Training for Variational Algorithms

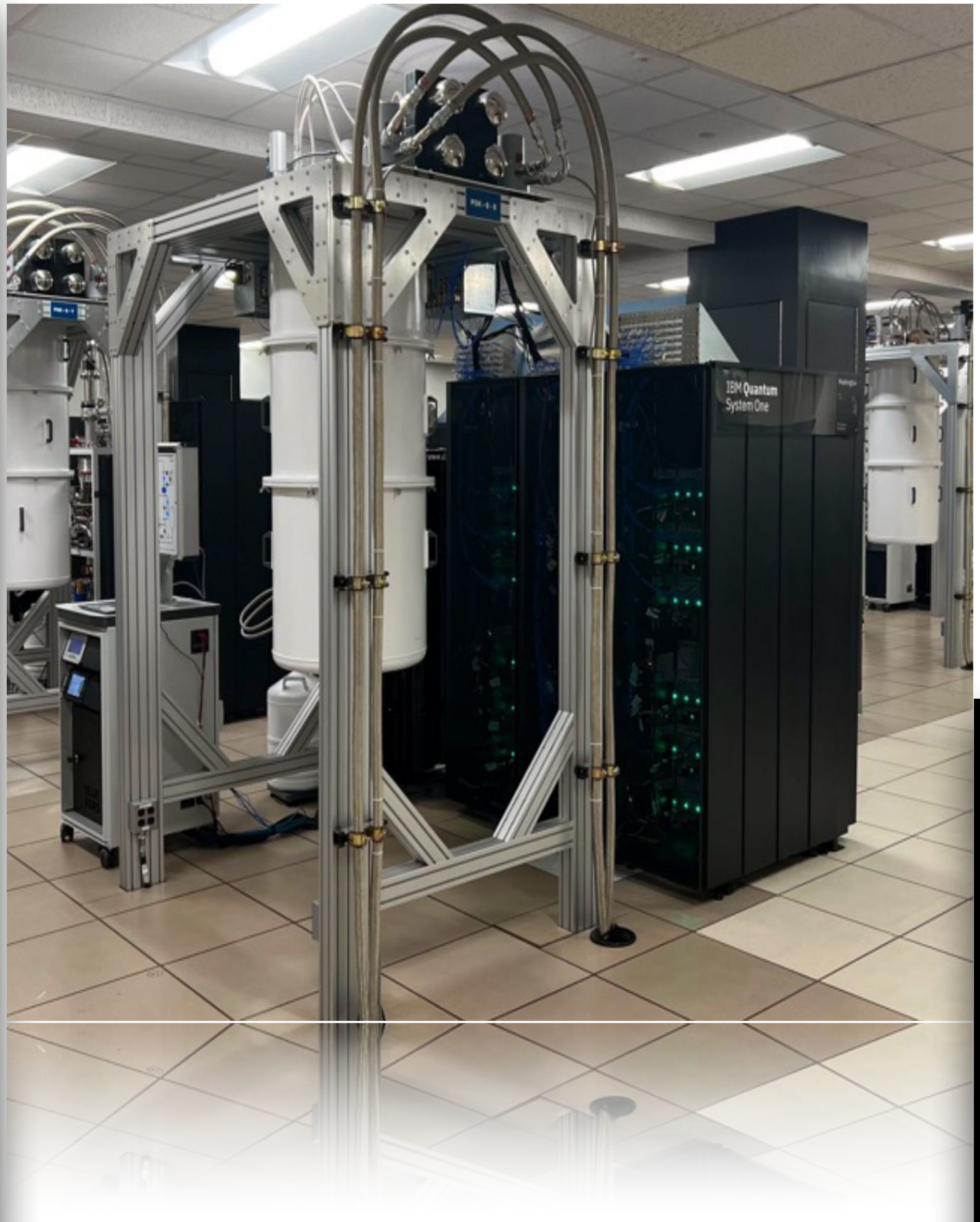
Maria Schuld, and Francesco Petruccione. Vol. 17. Berlin: Springer, 2018



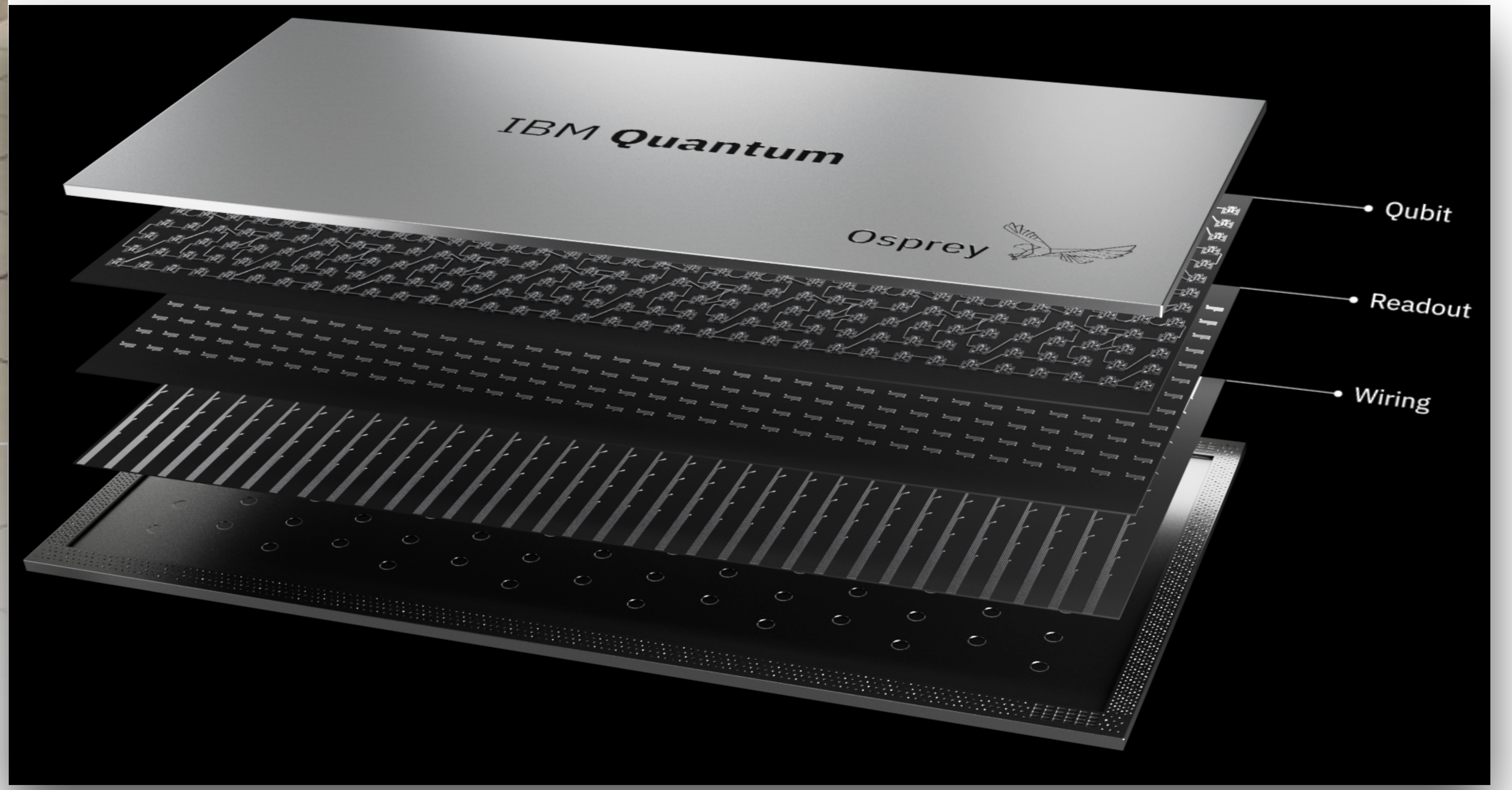
- Examples of the QC (no viable hardware):
  - QPCA
  - QSVM
  - QClustering

IBM quantum computer

[Credited to Thomas Prior for TIME]

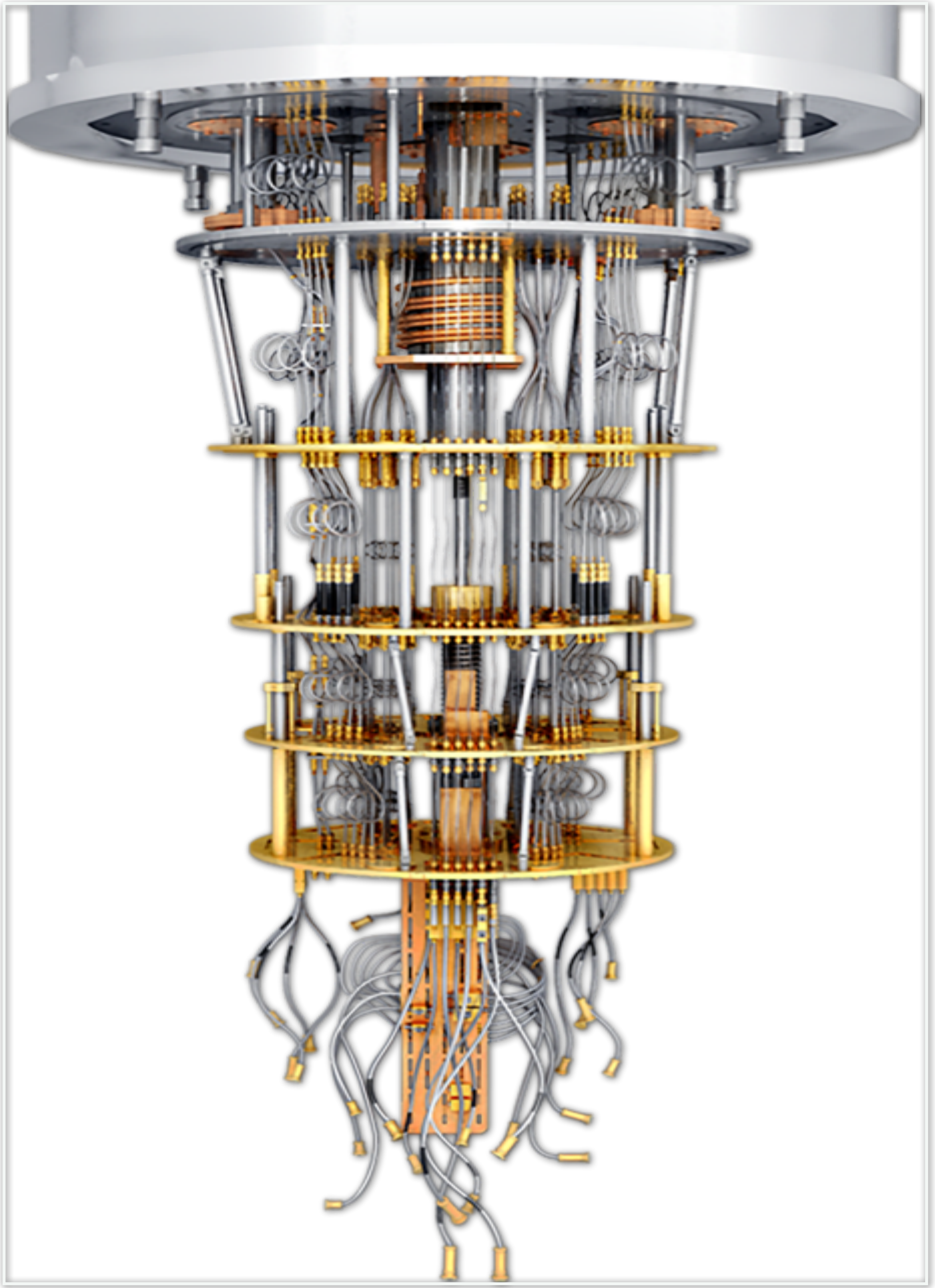


- ☐ Taking quantum computing out of the lab:
  - NY Computing Data Centre.
  - It provides over 20 computers.
  - Scales the processor's availability.
  - It provides over 20 comput



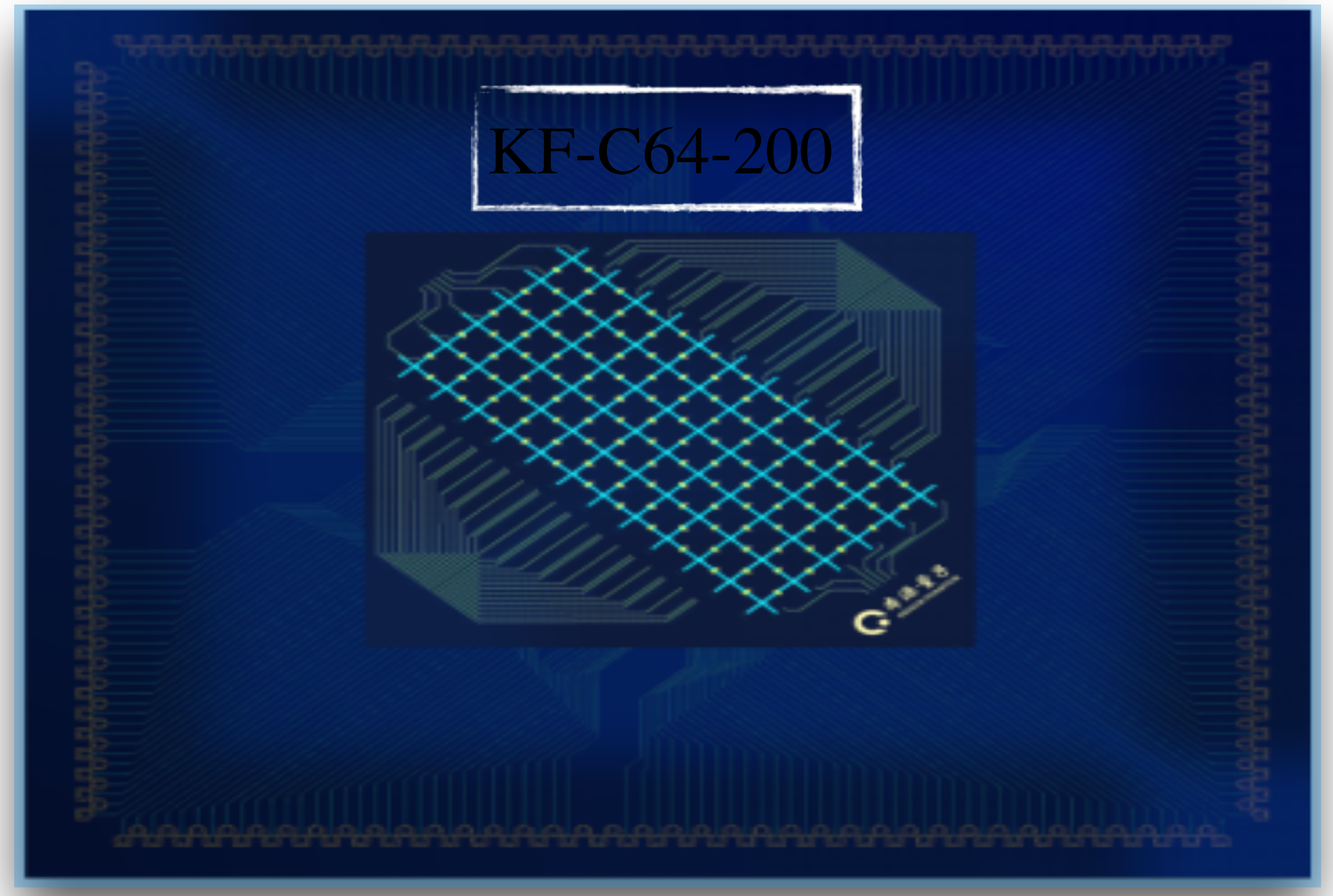
- ☐ [IBM] provides up to 127 qubits for free with an opportunity to apply for a researcher account with more qubits.

Origin quantum computer



Taken from Origin web page

- Origin launched 64-qubit QPU:
  - Single-qubit gate fidelity $> 99.9\%$.
  - Double-qubit gate fidelities $> 98\%$.
  - Readout fidelities $> 96\%$.



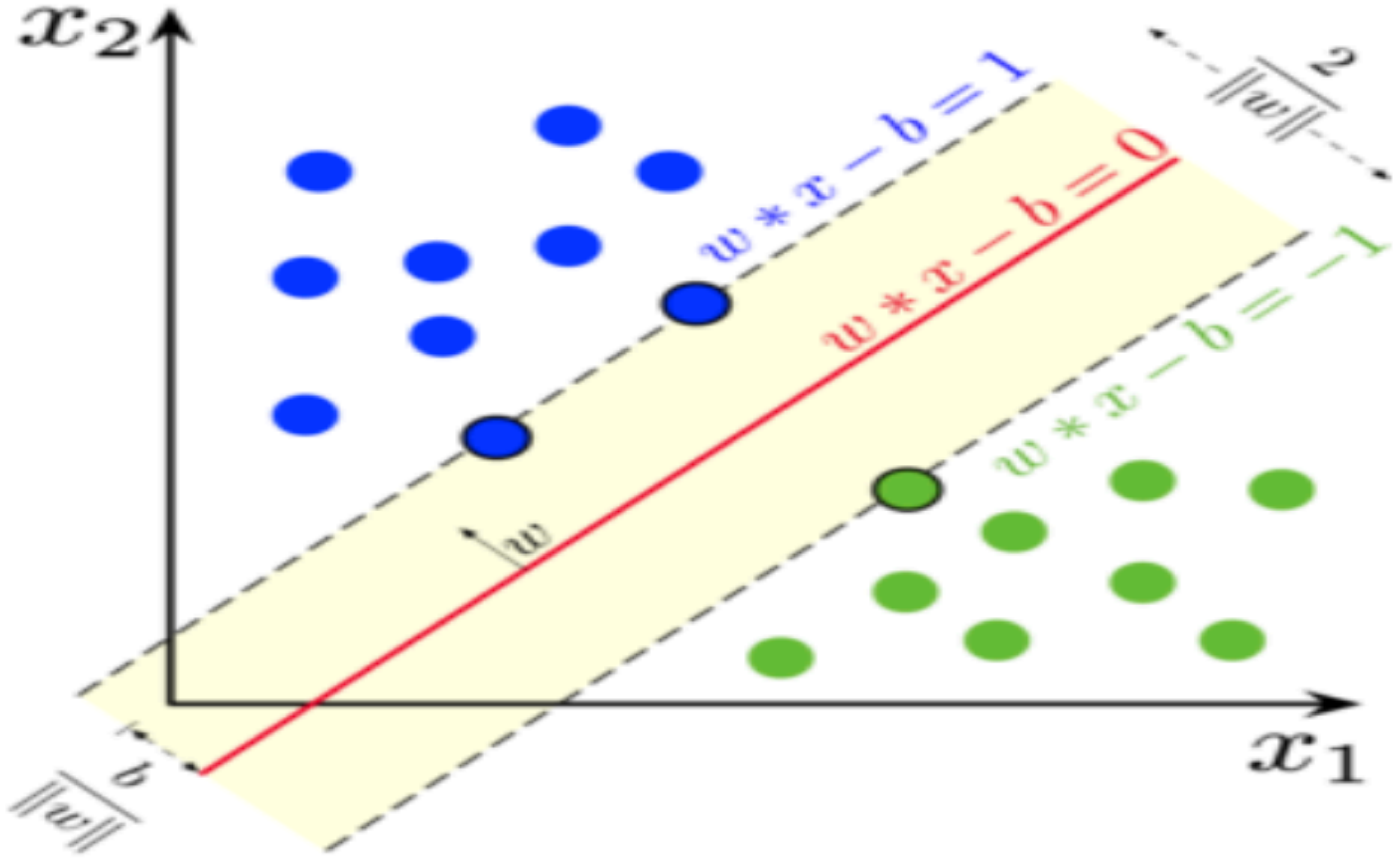
- Origin quantum computer provides up to 6 qubits for free. However, another hardware called Quafu provides up to 136 qubits. The Beijing Academy of Quantum Information Sciences maintains it.

Support vector machines

- SVM is a supervised machine learning algorithm used for classifications.

$$(\vec{x}_i, \vec{y}_i) = (\vec{x}_n, \vec{y}_n)$$

- $\vec{x}_i$ is an n-dimensional vector, and $\vec{y}_i$ is the class label of each data point.



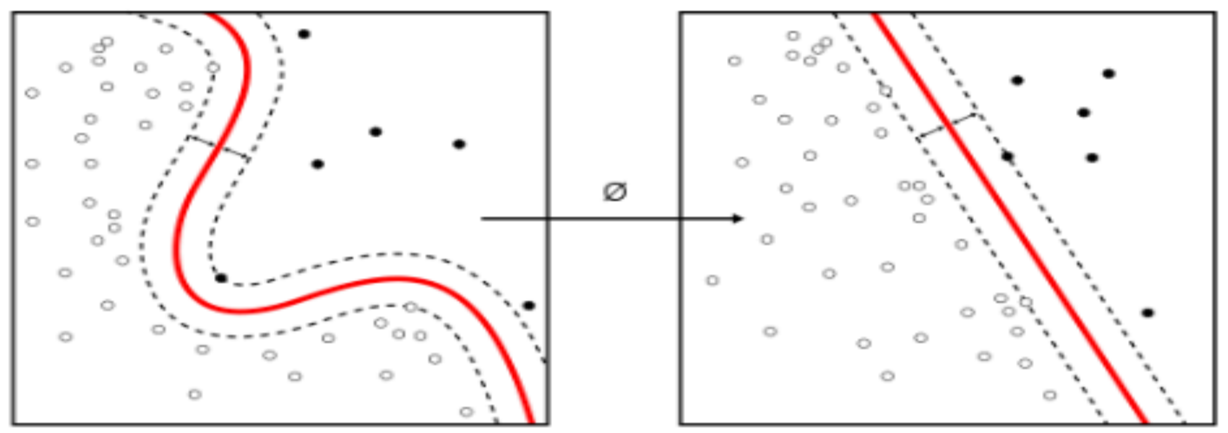
Support vector machines

Kernel trick

- ☐ The dot product of a feature $\vec{x}_i$ and $\vec{x}_j$, after being transferred to a higher dimension via a function f, is called kernel.

$$k_{ij}(\vec{x}_i, \vec{x}_j) = \langle f(\vec{x}_i), f(\vec{x}_j) \rangle$$

- ☐ Non-leaner futures can then mapped to a liner ones.
- ☐ The function $f(\vec{x})$ could be:
  - linear
  - polynomial
  - Radial basis function
  - sigmoid
- ☐ In our case, we'll be using a linear function;
- ☐ and we call the SMV a classical SVM.
- ☐ See examples here, also on our shared directory.



Quantum support vector machines

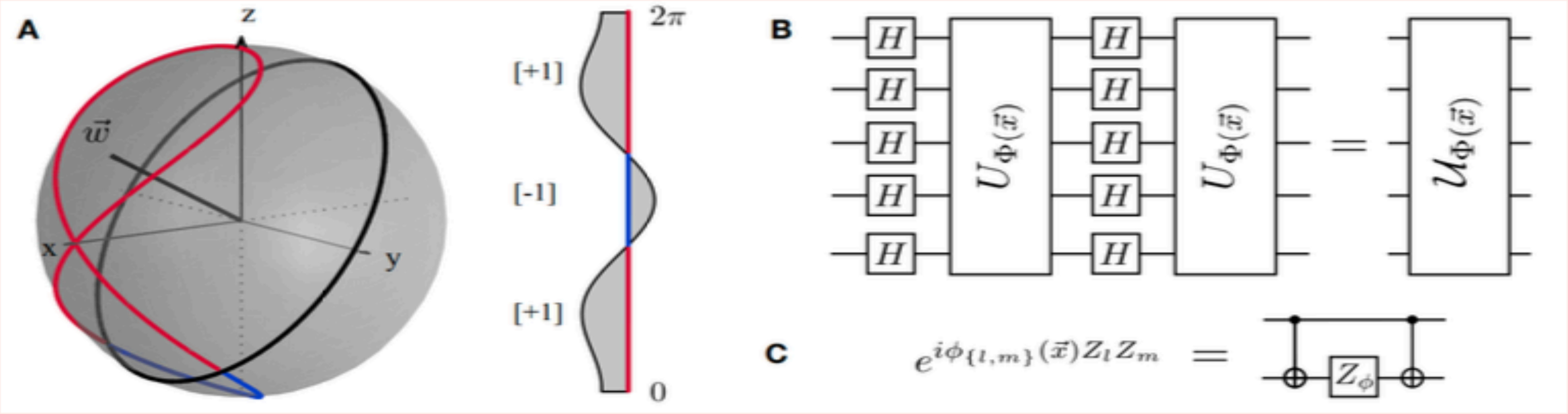
Quantum kernel

- ☐ In a quantum kernel, a classical feature $\vec{x}$ is mapped to higher dimension Hilbert space like $|\phi(\vec{x})\rangle\langle\phi(\vec{x})|$ in such a way that:

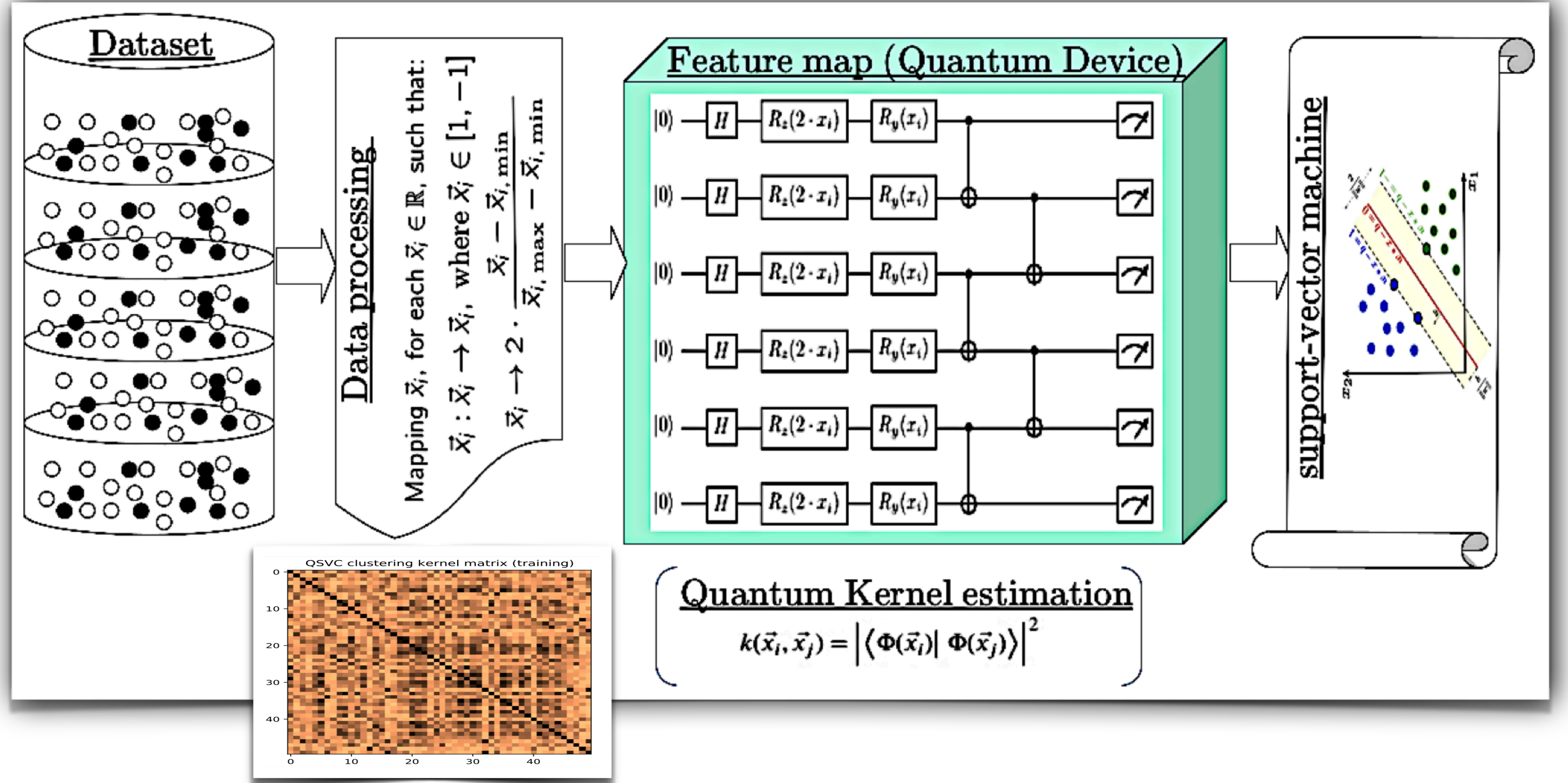
$$k_{ij}(\vec{x}_i, \vec{x}_j) = |\langle\phi(\vec{x}_i)|\phi(\vec{x}_j)\rangle|^2$$

- ☐ Feature map quantum circuits:
  - ZZFeatureMap
  - ZFeatureMap
  - PauliFeatureMap
- ☐ Many more in Qiskit packages.

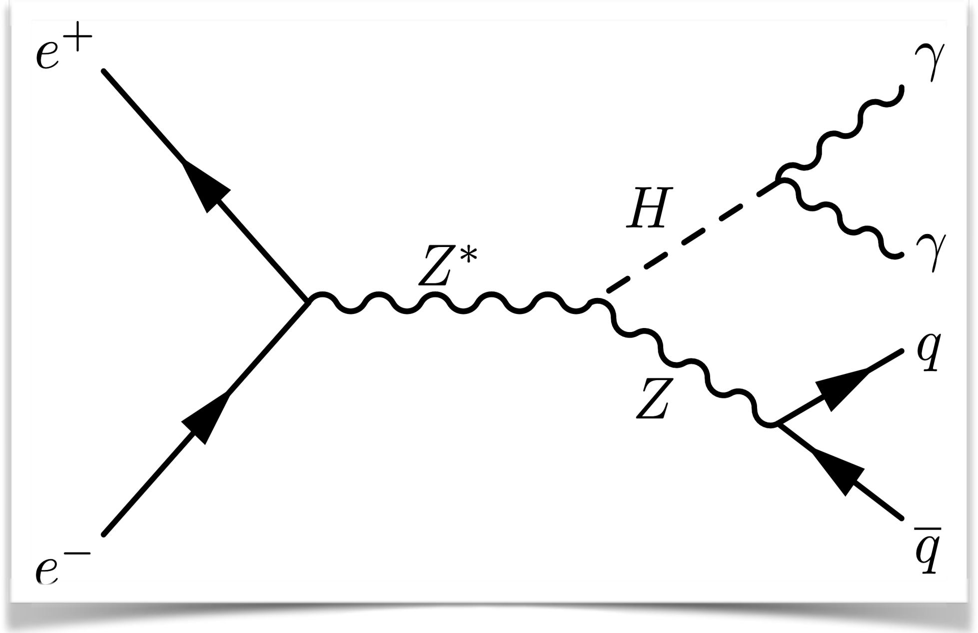
Nature volume 567, pages 209-212 (2019)



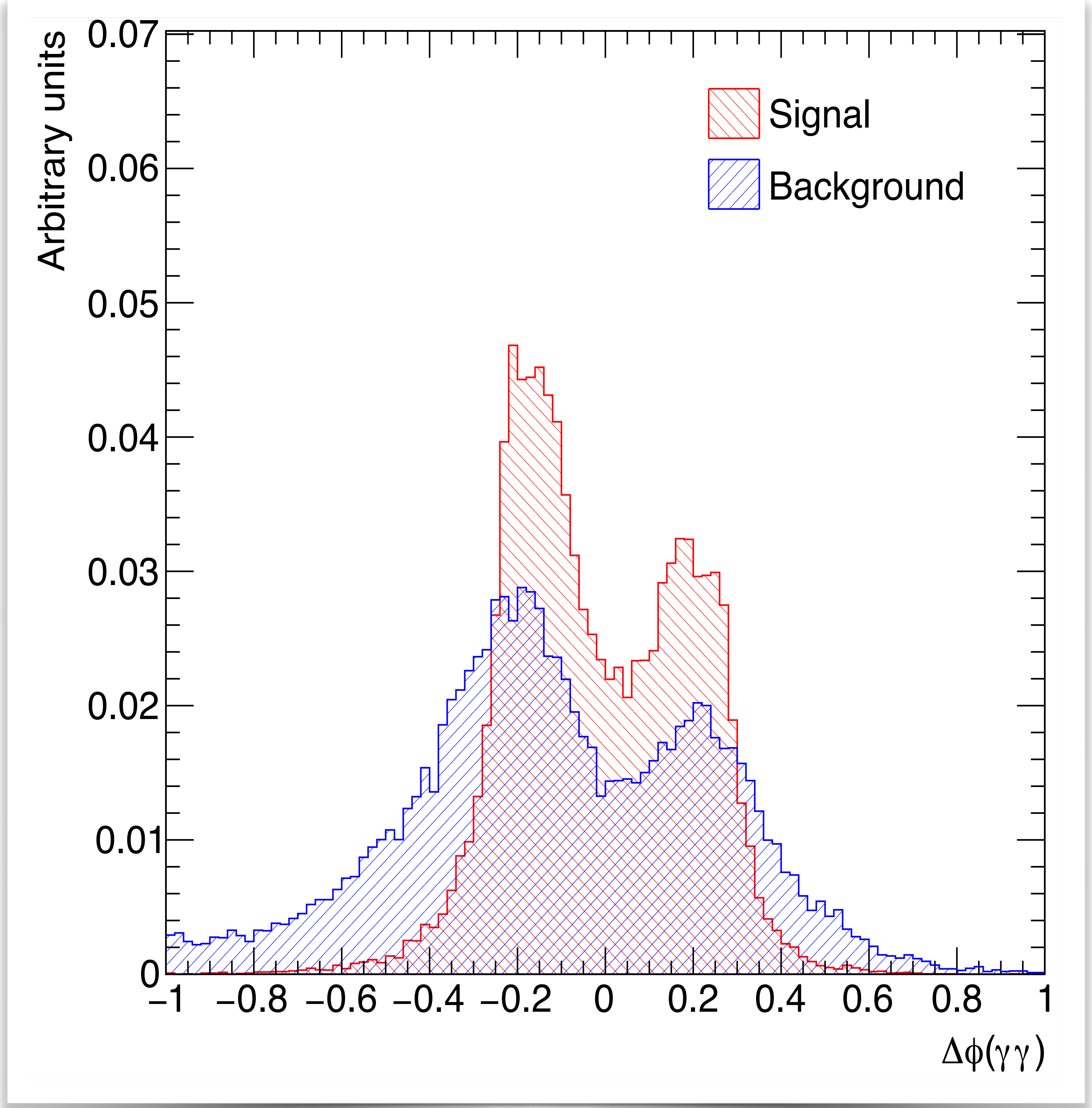
Data encoding and processing



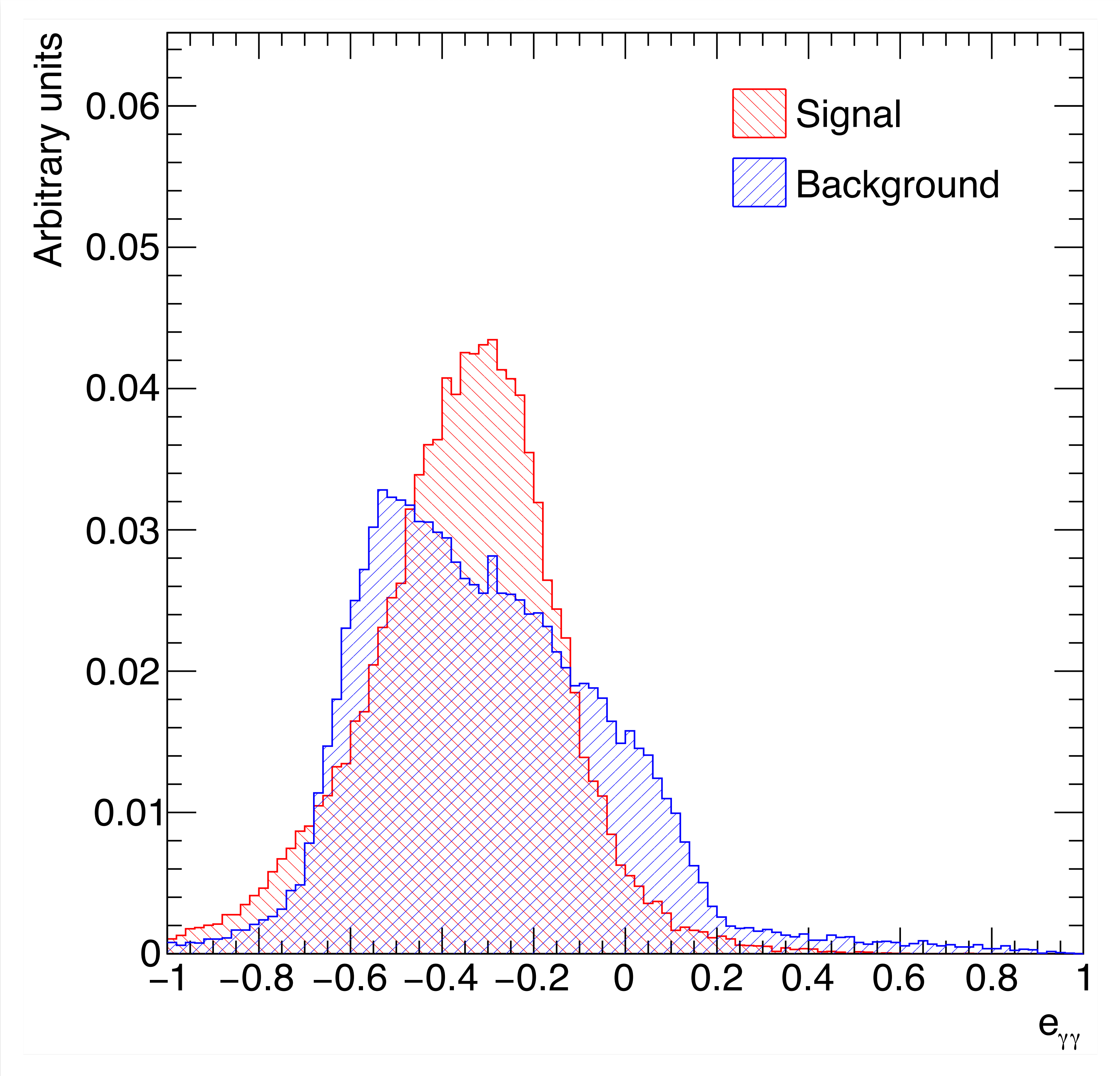
Variables



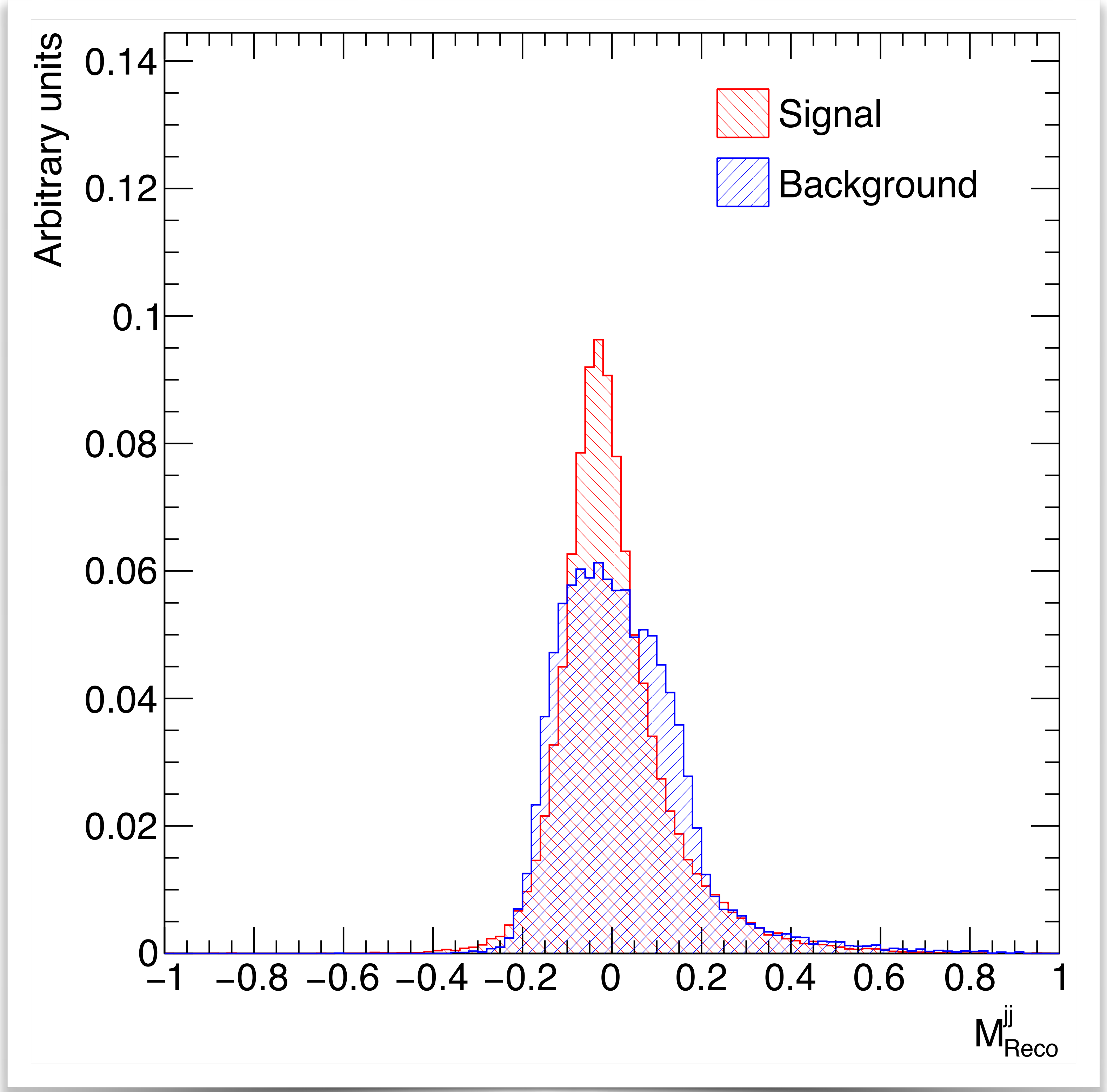
e^+
e^-
Z^*
H
Z
γ
γ
q
$\overline{q}$



Arbitrary units
Signal
Background
$\Delta\phi(\gamma\gamma)$



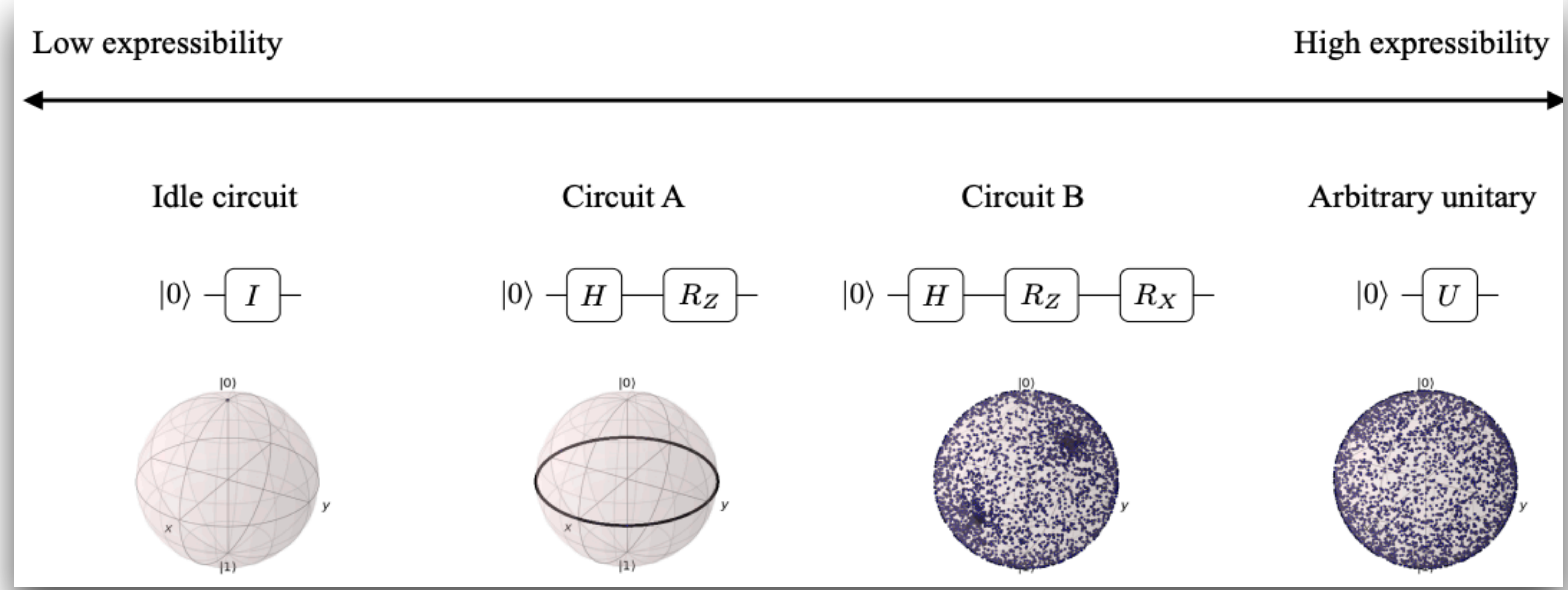
Arbitrary units
Signal
Background
$e_{\gamma\gamma}$

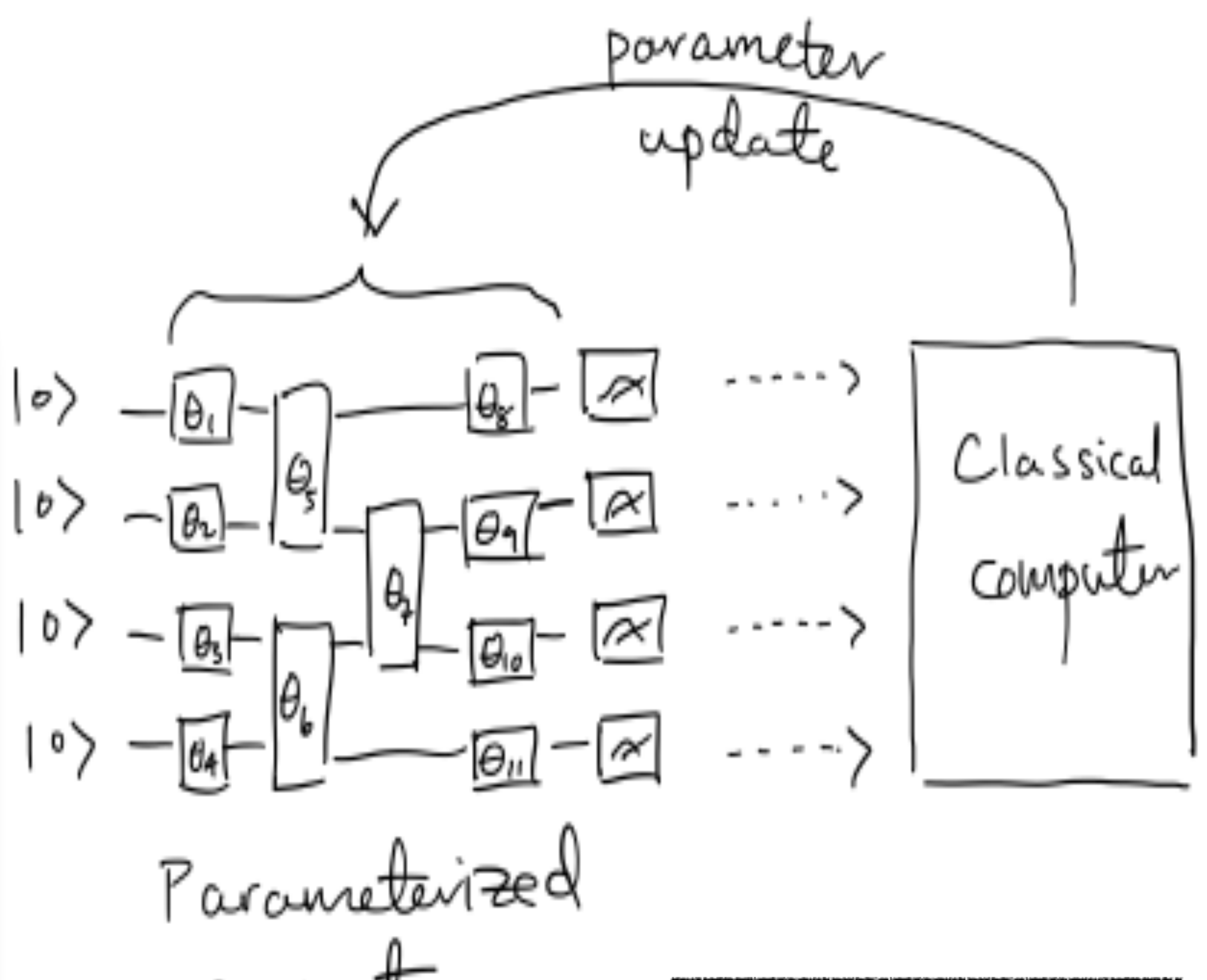


Arbitrary units
Signal
Background
M^{jj}_{Reco}

Quantum feature-map (quantum circut)

- Parametrised Quantum Circuits: useful for short-term quantum devices that can be defined using tunable parameters.
- Keeping in mind certain properties:
  - Expressibility





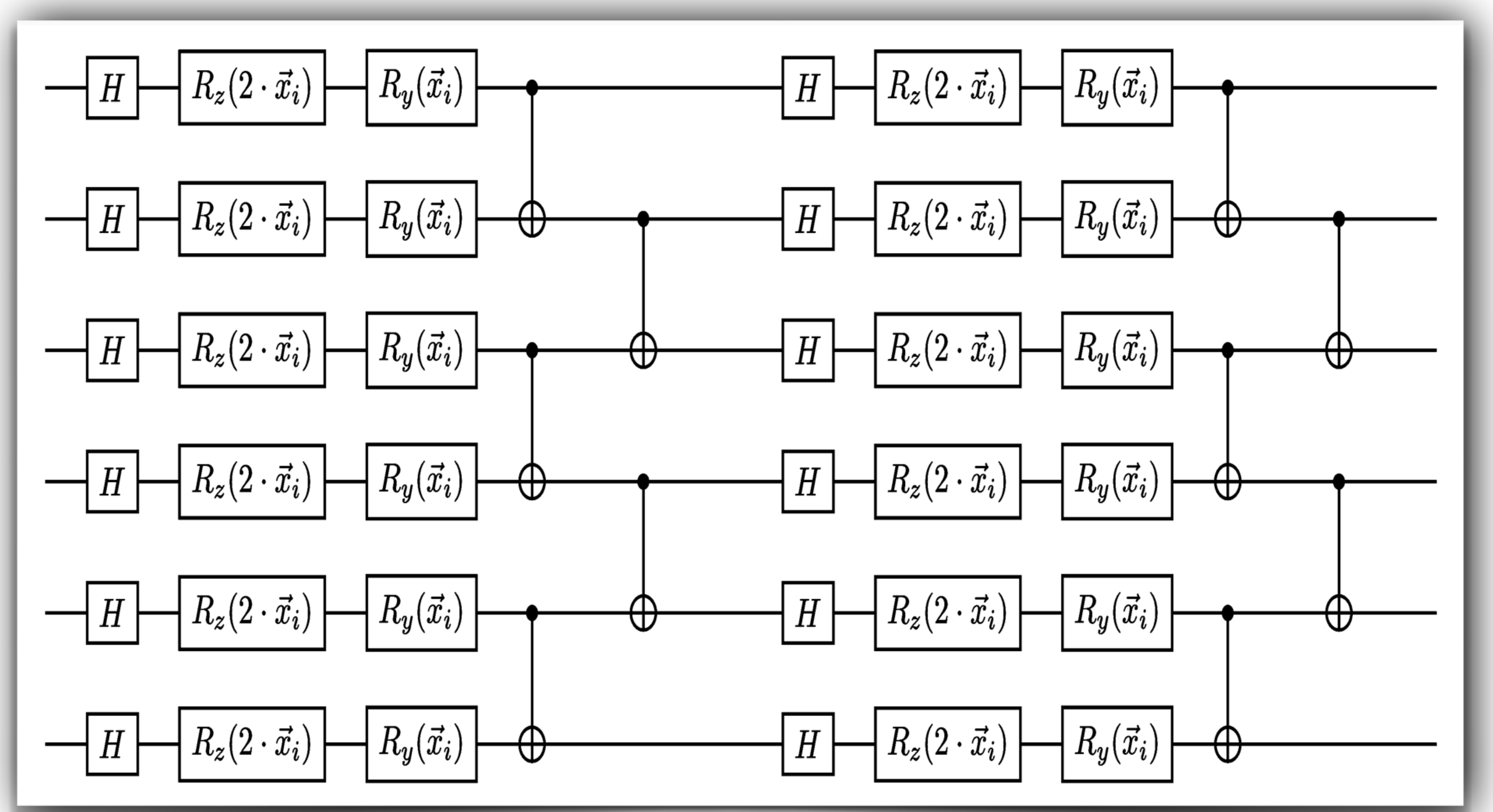
By Hanna Sim

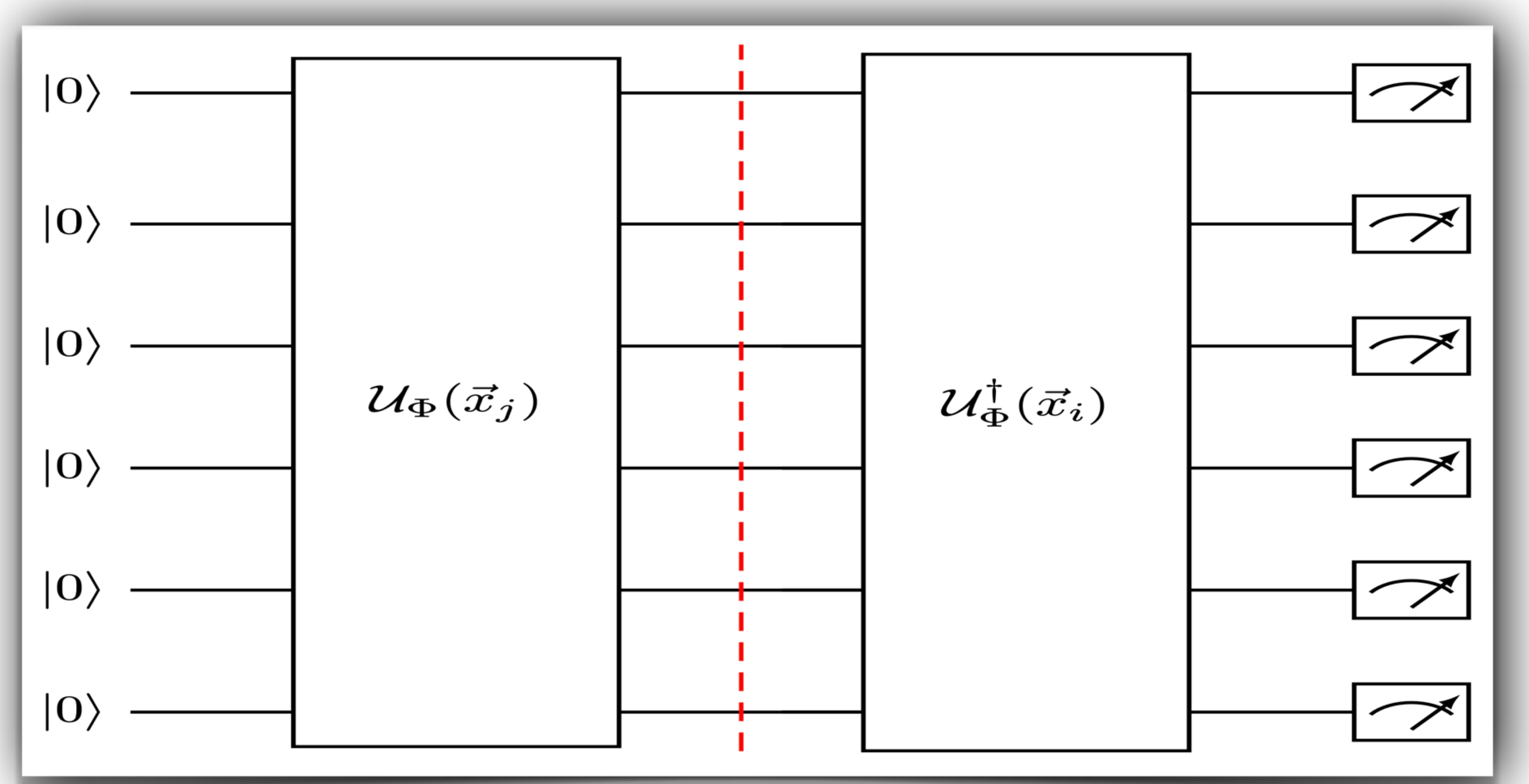
  - Entangling capability: the Meyer-Wallach measure
  - Hardware efficiency: qubits connectivity, fidelities …

Feature map and quantum kernel estimation

ArXiv: 2209.12788

- The quantum feature map dictates the kernel:
  - Single-qubit Hadamard gate
  - Single-qubit rotation gates $R_z(x)$and$R_y(x)$
  - Two-qubit CNOT entangling gates
  - Two identical layers (depth)





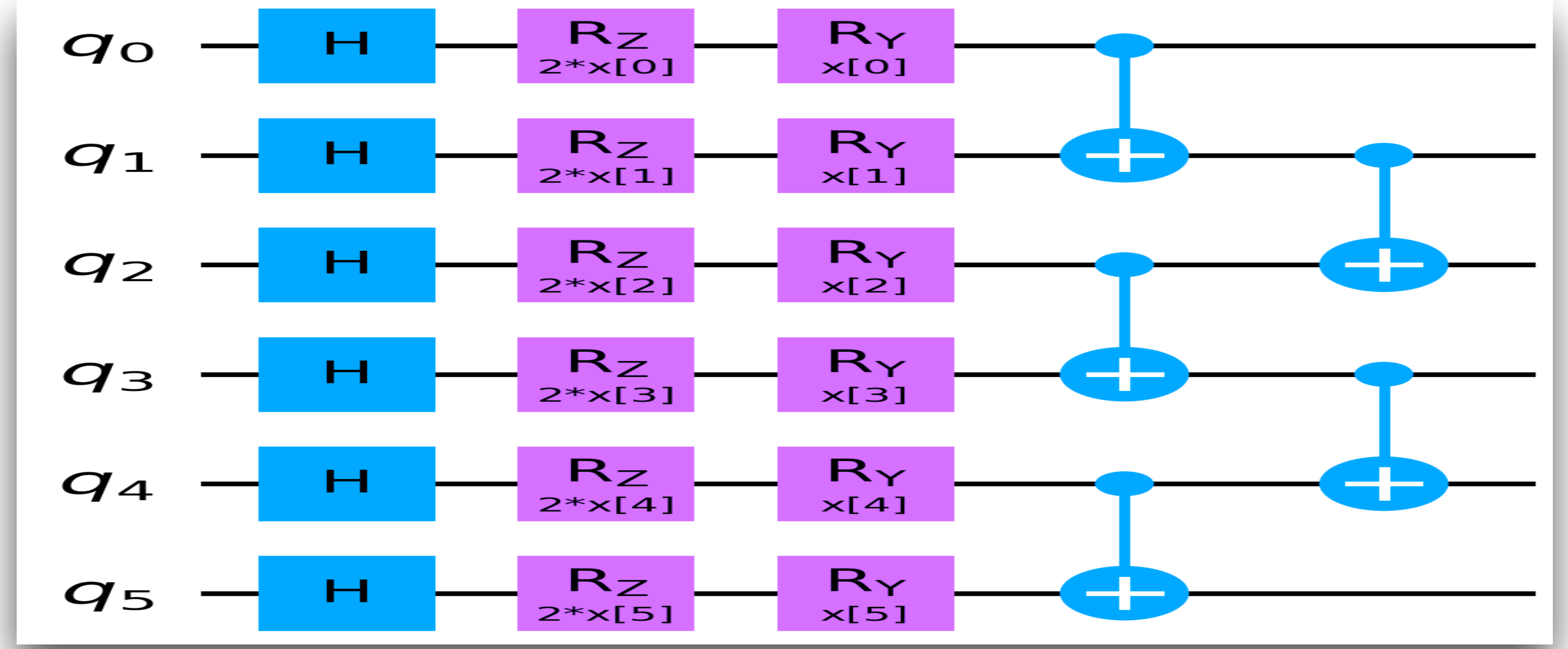
- The quantum support vector kernel estimation:
- $$k(\vec{x}_j, \vec{x}_j) = \left| \left\langle 0^{\otimes N} \right| U^{\dagger}_{\phi(\vec{x}_i)} U_{\phi(\vec{x}_j)} \left| 0^{\otimes N} \right\rangle \right|^2$$
- $N \equiv$ Six qubits are mapped to six variables.
- The expectation of each data point w.r.t the rest.

Feature map optimisation

ArXiv: 2209.12788

- The following feature map form was found to work best for the $e^+e^- \rightarrow ZH \rightarrow \gamma\gamma q\bar{q}$ signal.
- Five thousand events were used with different rotation combinations $R_y(\vec{x})$ and $R_z(\vec{x})$.
- The current entanglements were the best.
- The quantum circuit is repeated twice to achieve better entanglements between qubits.
- The area under the curve (AUC) decides the best rotation and entanglements.

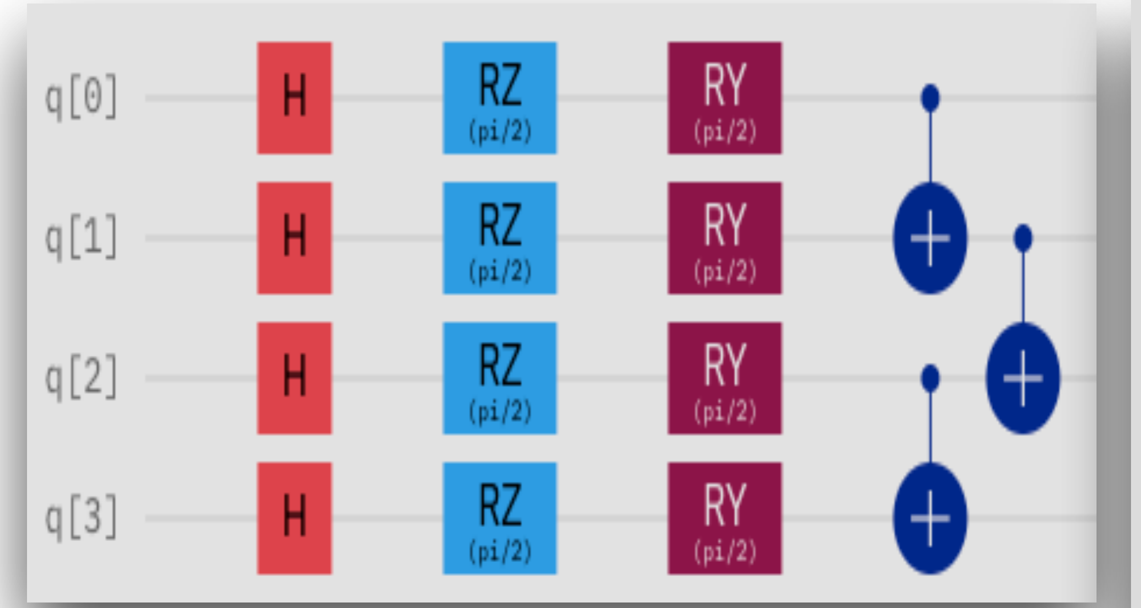
Rotation	Depth	Events	Best AUC	Variation
$R_z(2 \cdot \vec{x}_i) + R_y(\vec{x}_i)$	2	5000	0.935	0.009
$R_z(\vec{x}_i) + R_y(\vec{x}_i)$			0.933	0.015
$R_y(\vec{x}_i) + R_x(\vec{x}_i)$			0.932	0.015
$R_z(\vec{x}_i) + R_z(\vec{x}_i)$			0.932	0.014
$R_y(\vec{x}_i)$			0.928	0.008
$R_z(\vec{x}_i)$			0.928	0.008

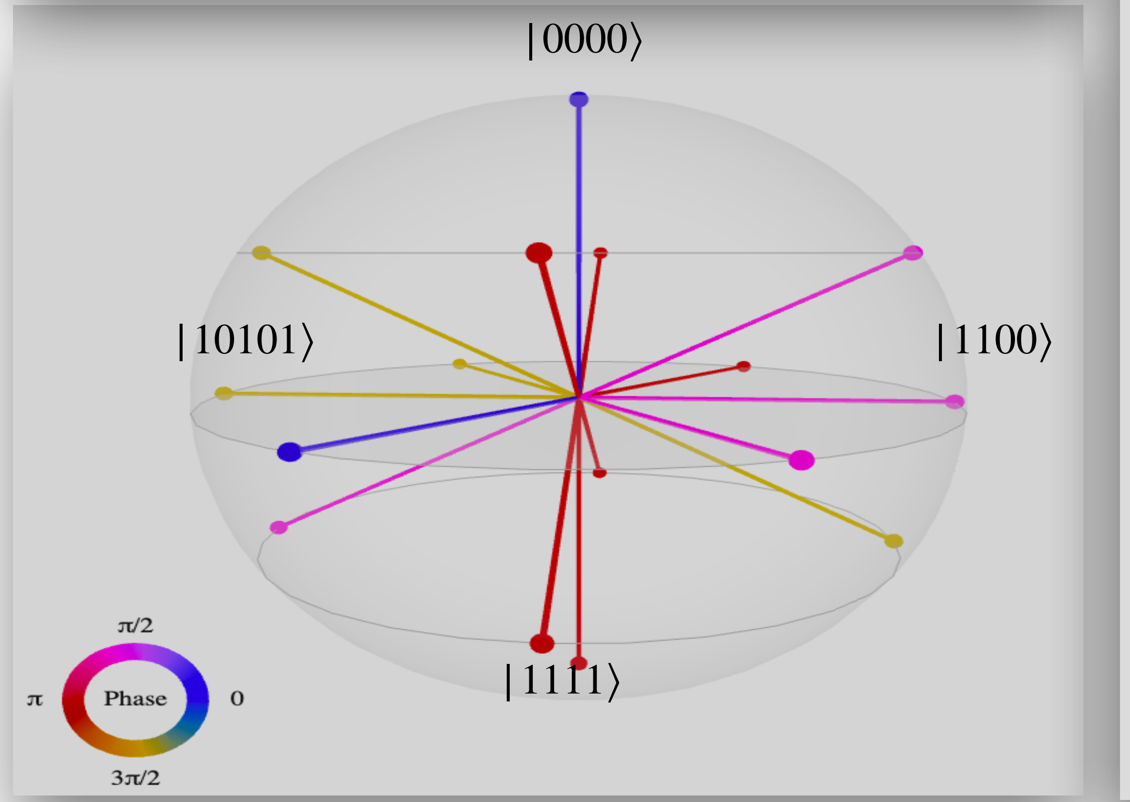


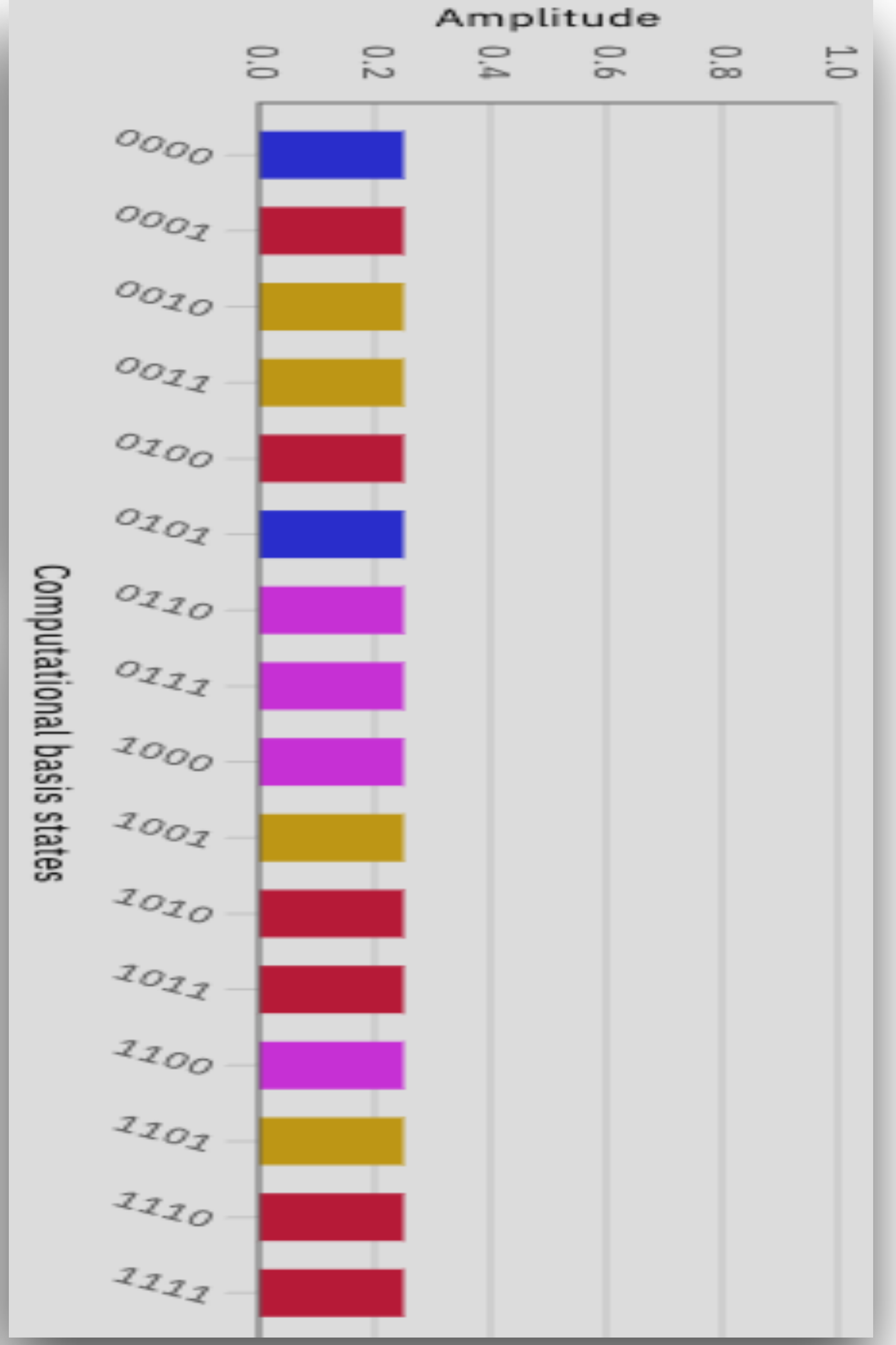
The performance of the quantum simulator

ArXiv: 2209.12788

- The quantum simulator has the following:
  - Statevector Simulator developed by the **Qiskit software package**

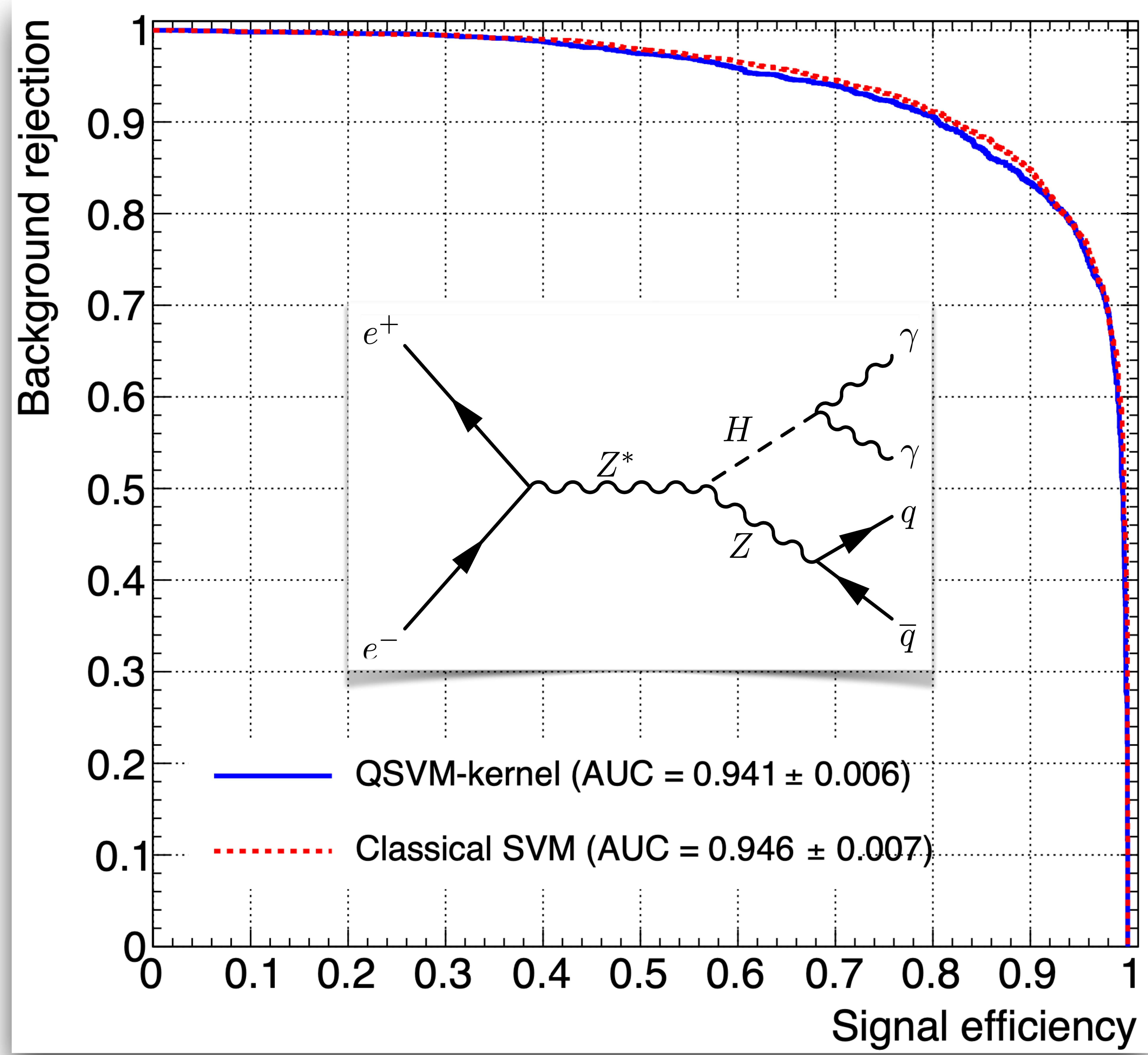






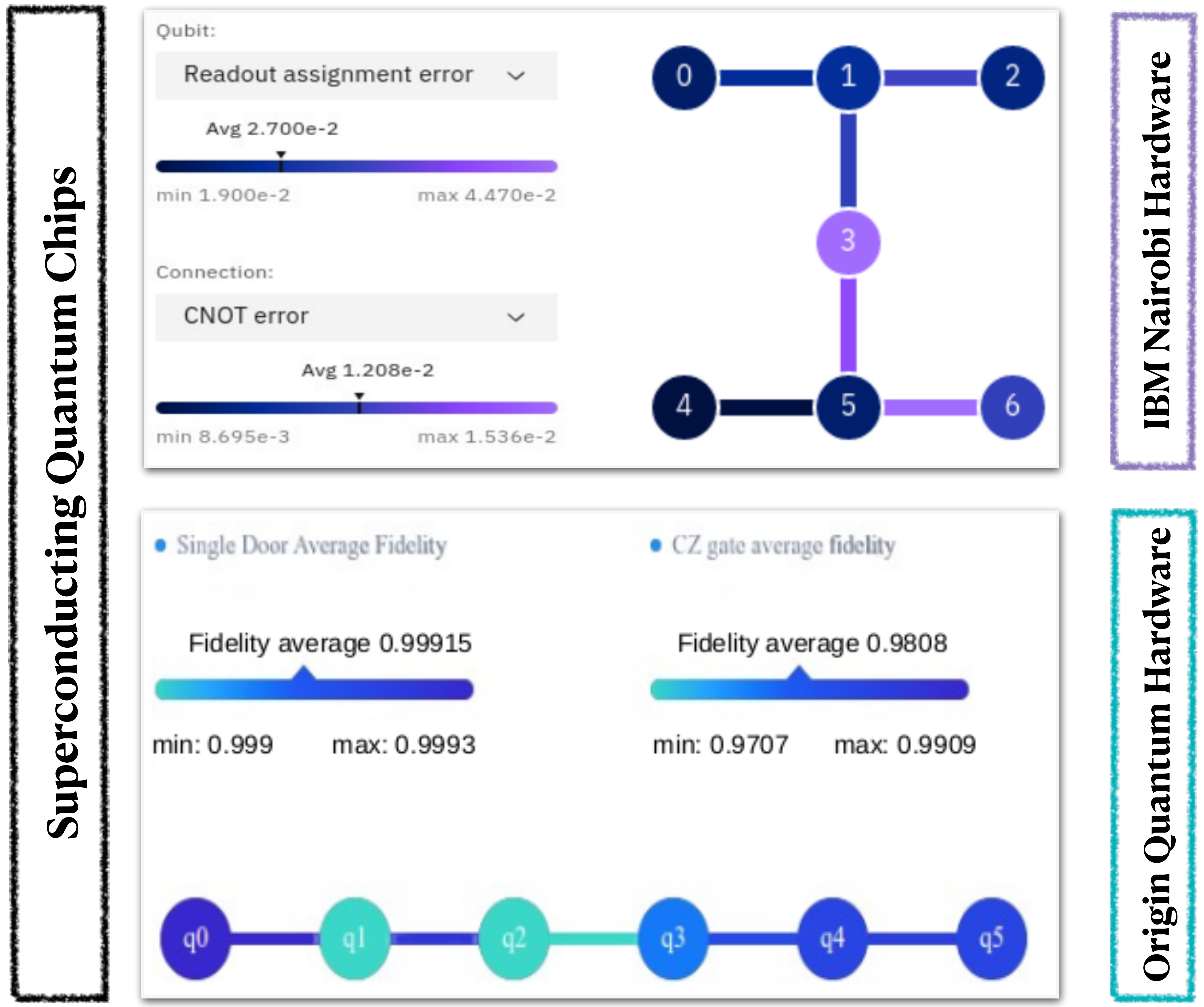
  - Six quantum bits or simply qubits

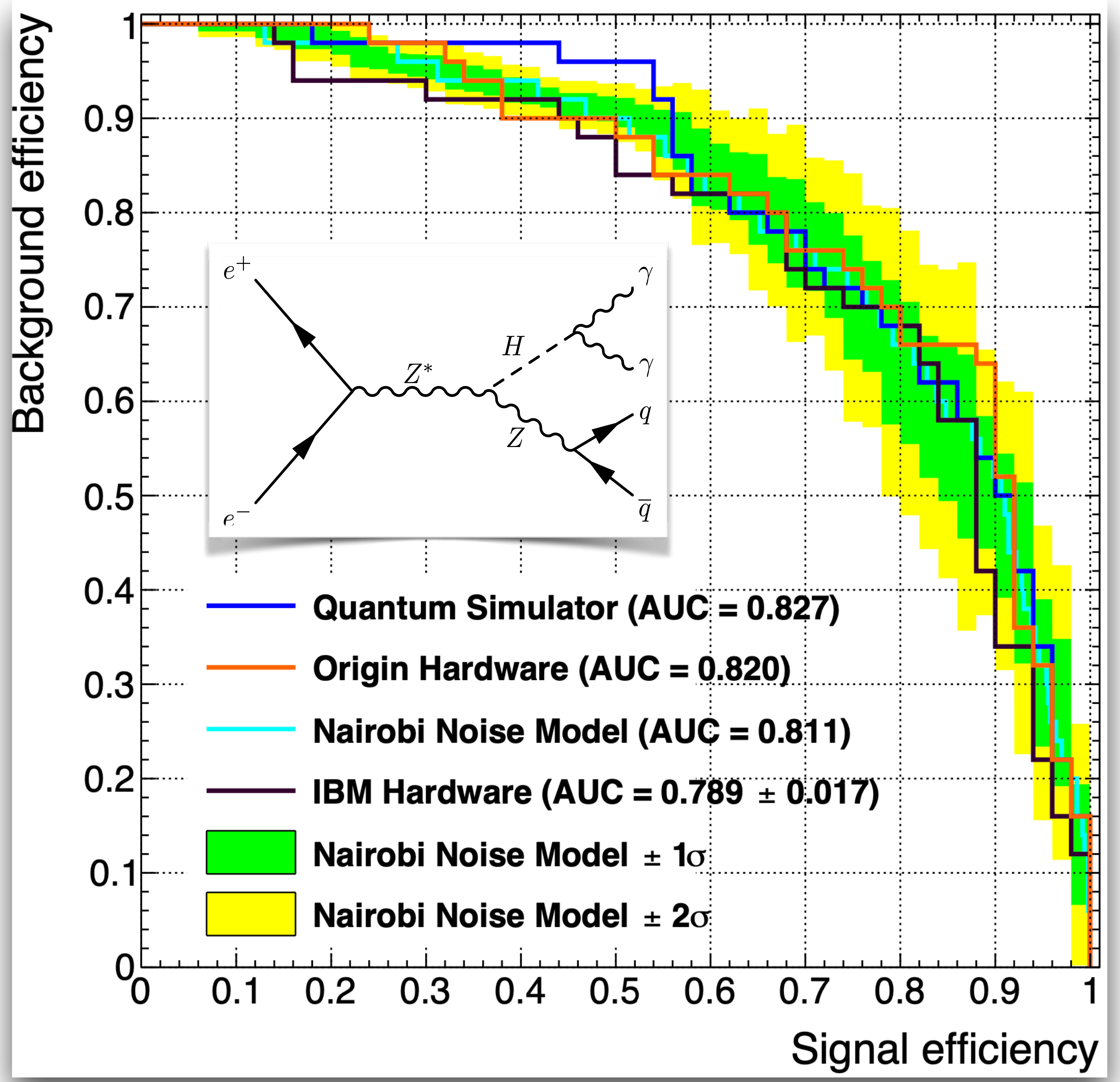
- A total of 12000 events were used.



The performance of quantum computers

[ArXiv: 2209.12788](https://arxiv.org/abs/2209.12788)

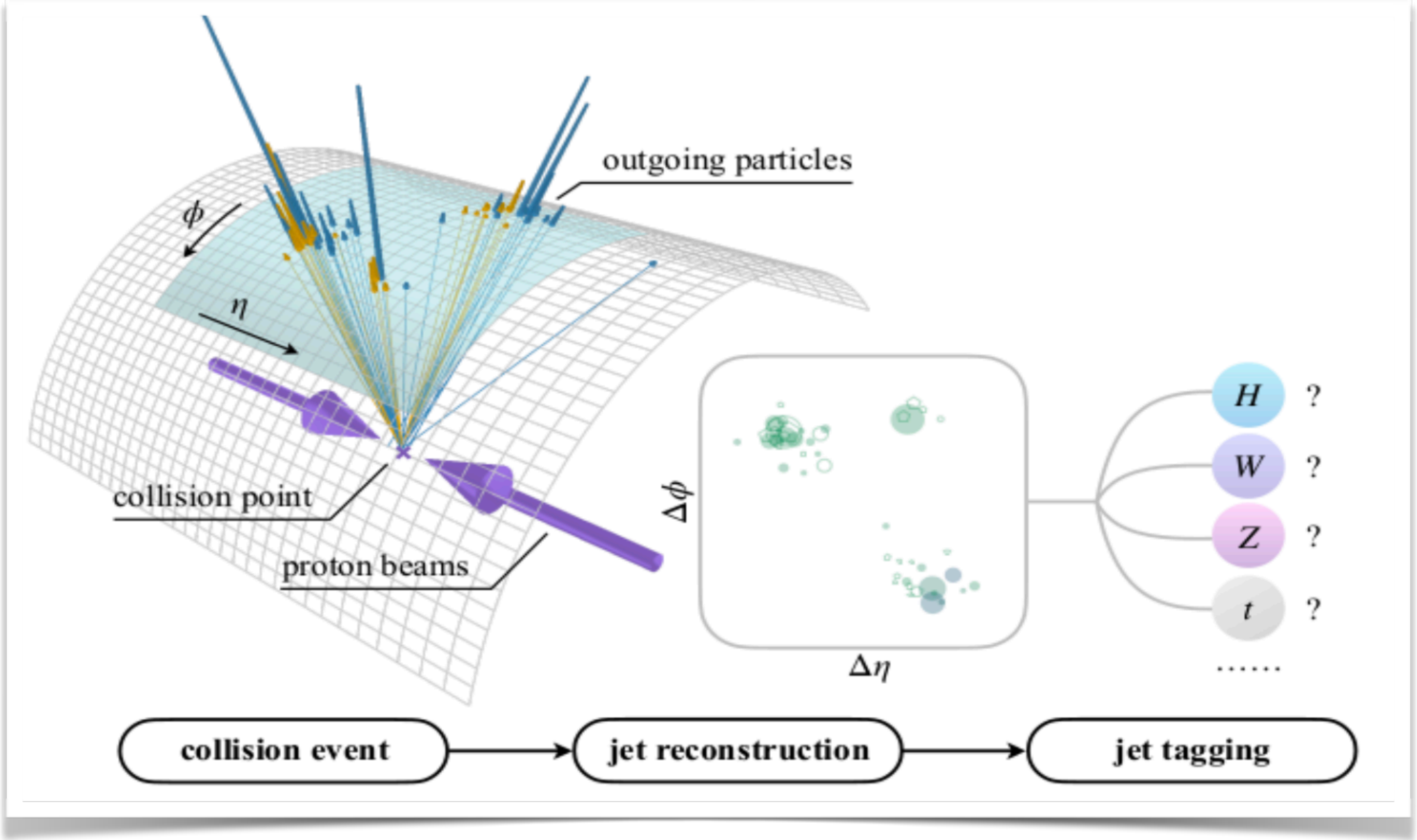




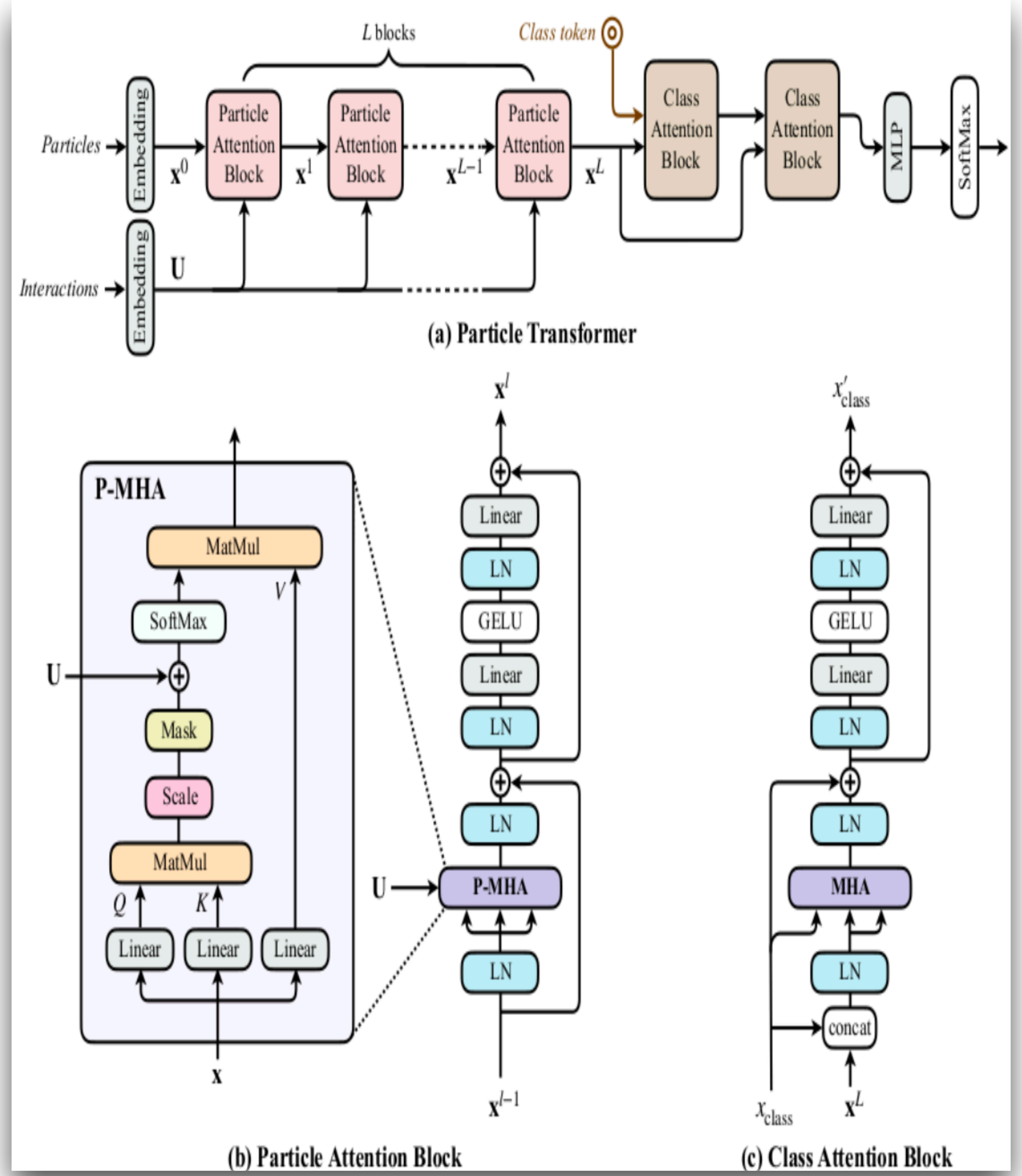
- ❑ Six qubits were used for both quantum hardware.
- ❑ 100 events were used for the training and testing.
- ❑ Comparable performance is observed between IBM's and Origin's quantum hardware.

The Particle Transformer

- Jet tagging classification in particle physics

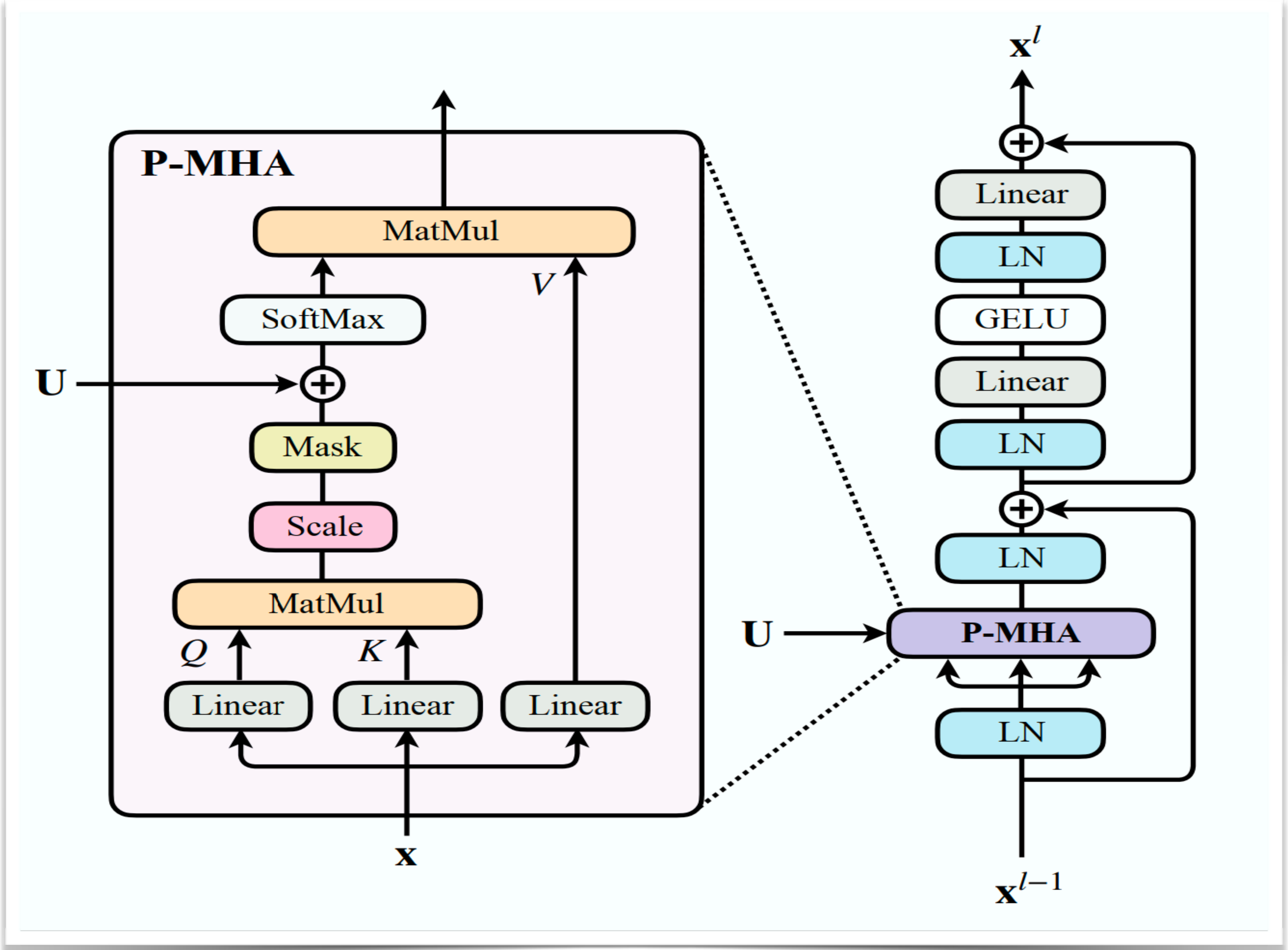


- Particle: a list of features for each particle
- Interactions: features involving a pair of particles
- Passing through a series of “attention” to MLP
- [ArXiv: 2202.03772](https://arxiv.org/abs/2202.03772): Particle Transformer

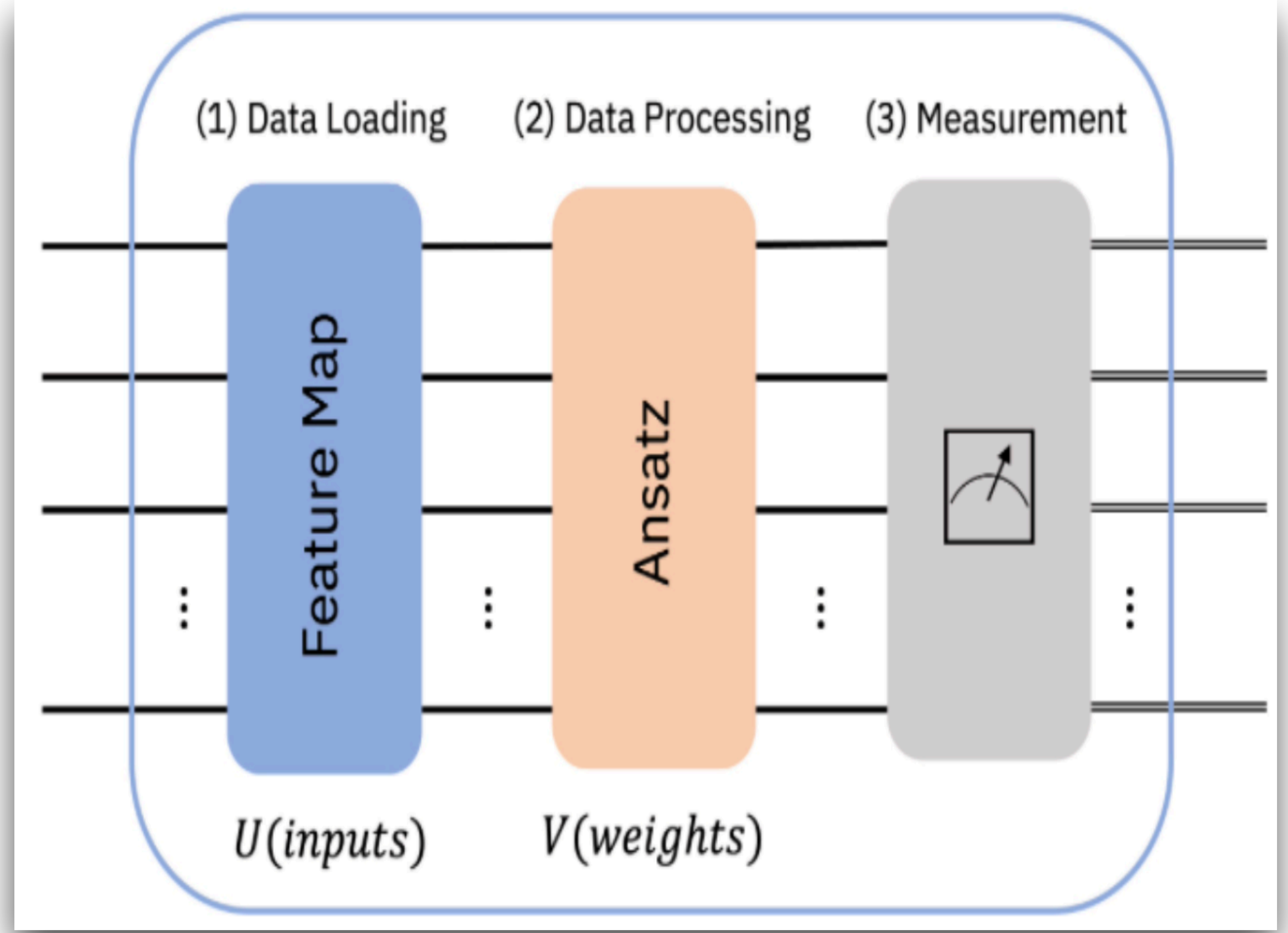


The Quantum Particle Transformer

- Particle transformer
  - Multihead-Attention based on PyTorch
  - The idea is to replace this part with a quantum



Quantum Neural Network



- The trainable parameters are added using the Anzatz with a feature-map that acts as an encoder.

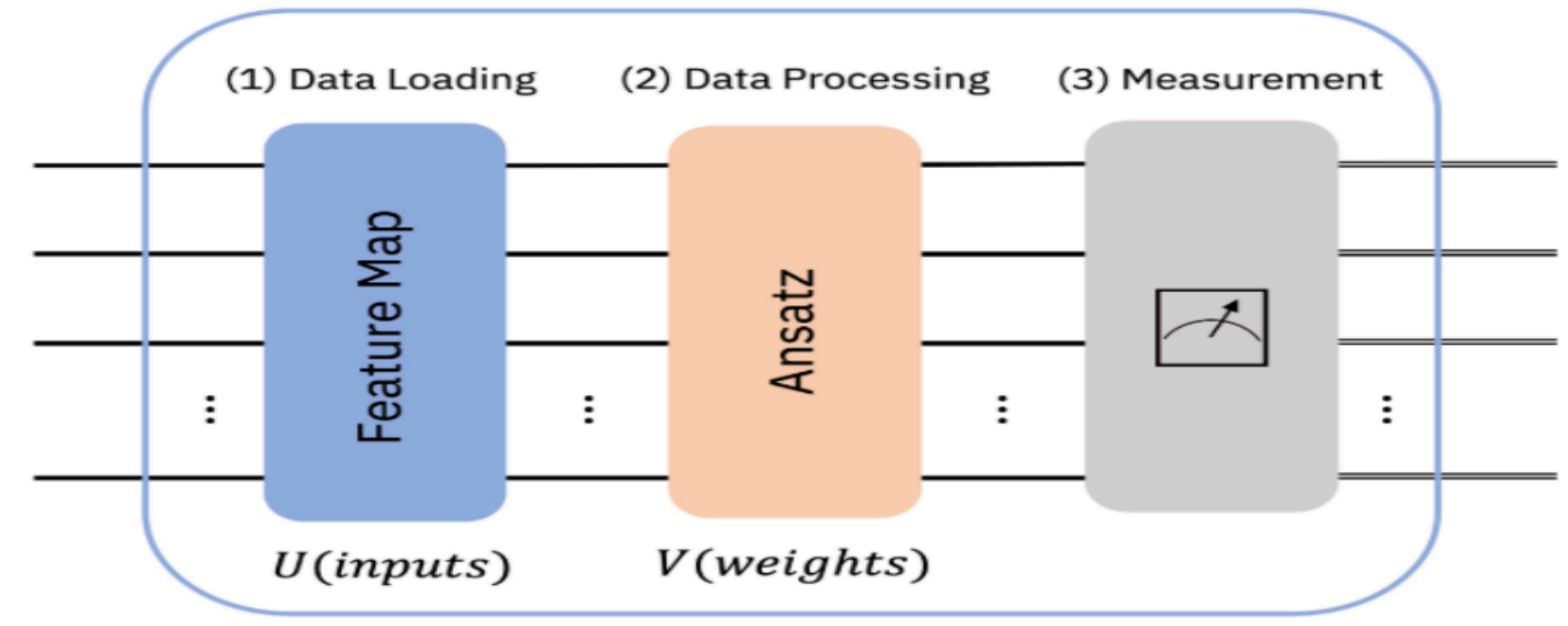
The Quantum Particle Transformer

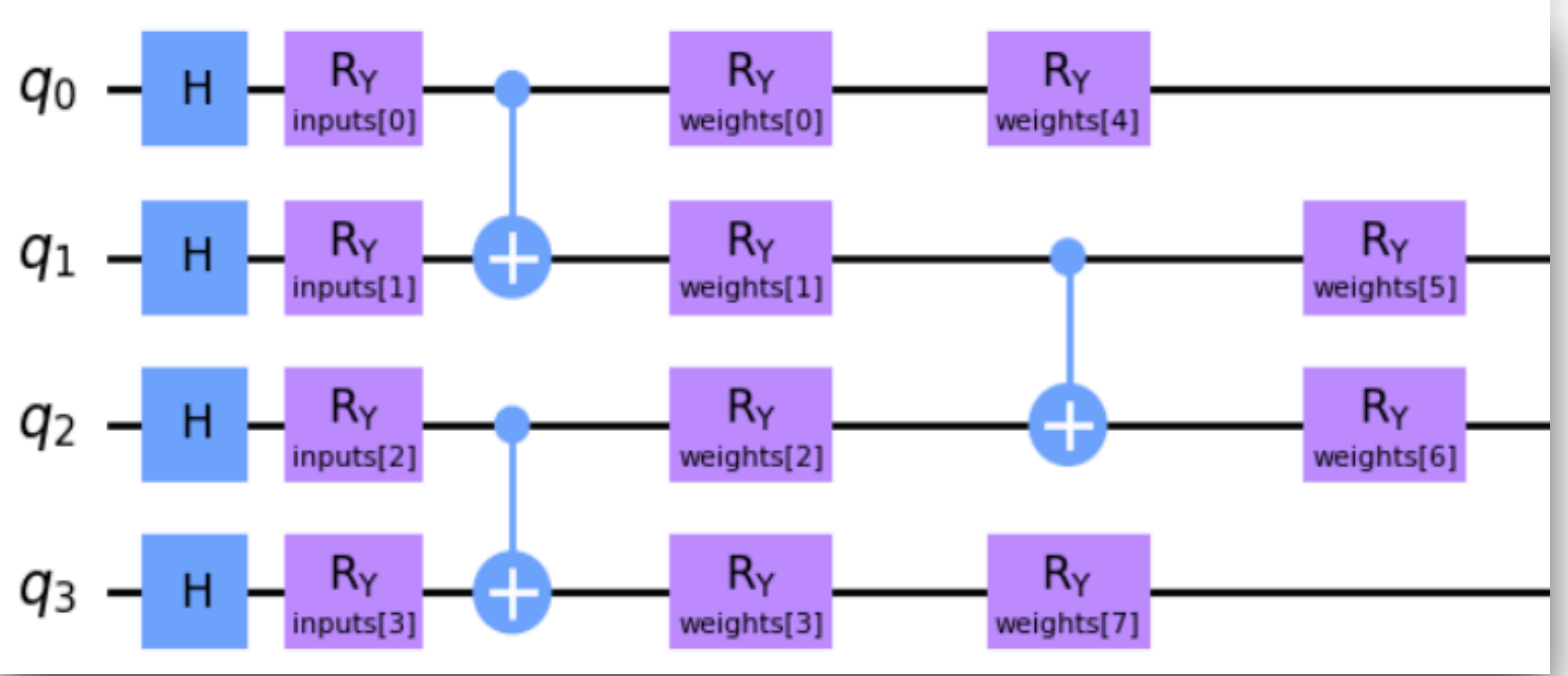
- The implementation of the quantum self-attention

```python
class QuantumSelfAttention(nn.Module):
    def __init__(self, embed_size, heads):
        super(QuantumSelfAttention, self).__init__()
        self.embed_size = embed_size
        self.heads = heads
        self.head_dim = embed_size // heads

        assert (self.head_dim * heads == embed_size), "Embed size needs to be divided by heads"

        self.values = QuantumLinearLayer(self.embed_size, self.head_dim)
        self.keys   = QuantumLinearLayer(self.embed_size, self.head_dim)
        self.queries  = QuantumLinearLayer(self.embed_size, self.head_dim)
        self.fc_out = QuantumLinearLayer(heads * self.head_dim, embed_size)

        #print(values.shape)
        #print(keys.shape)
        #print(queries.shape)

    def forward(self, values, keys, queries, mask):
        N = queries.shape[0]
        value_len, key_len, query_len = values.shape[1], keys.shape[1], queries.shape[1]

        values = self.values(values)
        keys = self.keys(keys)
        queries = self.query(queries)

        values = values.reshape(N, value_len, self.heads, self.head_dim)
        keys = keys.reshape(N, key_len, self.heads, self.head_dim)
        queries = queries.reshape(N, query_len, self.heads, self.head_dim)

        energy = torch.einsum("nqhd,nkhd->nhqk", [queries, keys])

        #if mask is not None:
        #    energy = energy.masked_fill(mask == 0, float("-1e20"))

        attention = torch.softmax(energy / (self.head_dim ** 0.5), dim=3)

        out = torch.einsum("nhql,nlhd->nqhd", [attention, values]).reshape(N, query_len, self.heads * self.head_dim)
        out = self.fc_out(out)

        return out
```

```python
def forward(self, x):
    x /= torch.sqrt(torch.sum(x**2))
    #qnn = QNNetwork(n_qubits)
    qnn_weights = algorithm_globals.random.random(self.qnn.num_weights)
    qnn_forward_batched = self.qnn.forward([x,x], qnn_weights)

    print(f"\nShape: {qnn_forward_batched.shape}")

    return qnn_forward_batched
```

Quantum Neural Network

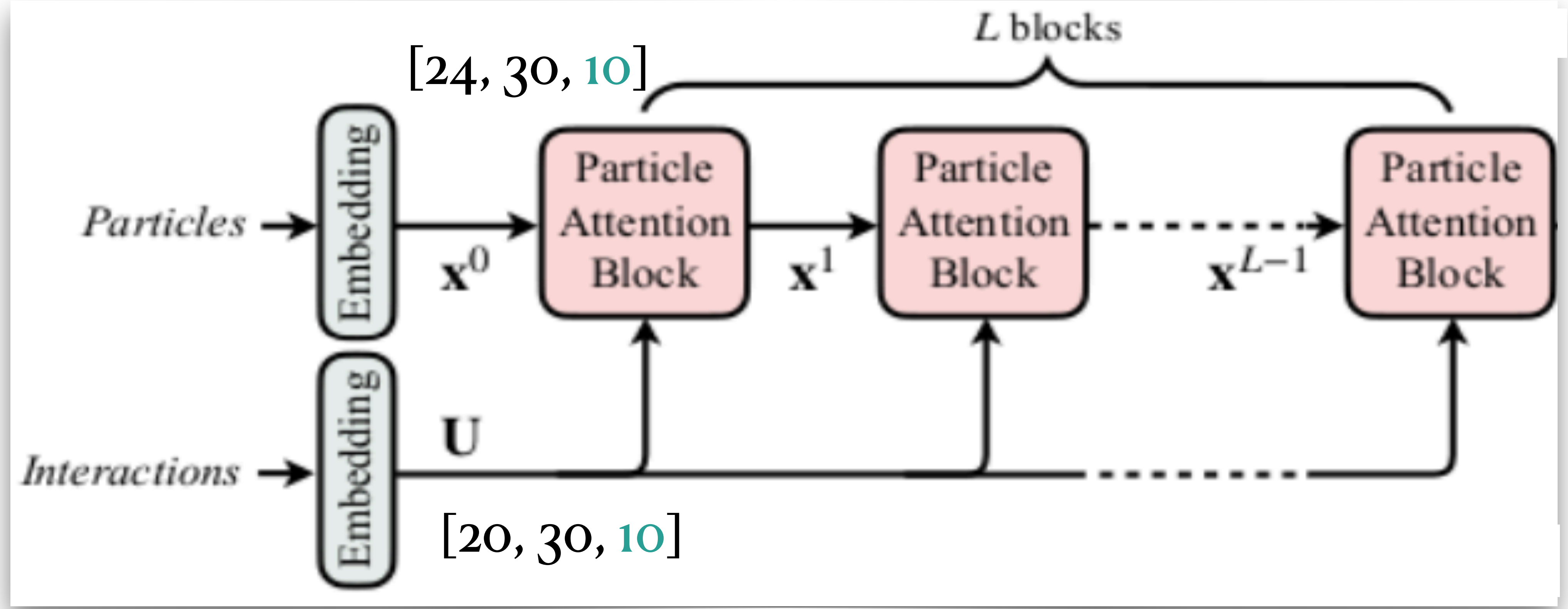




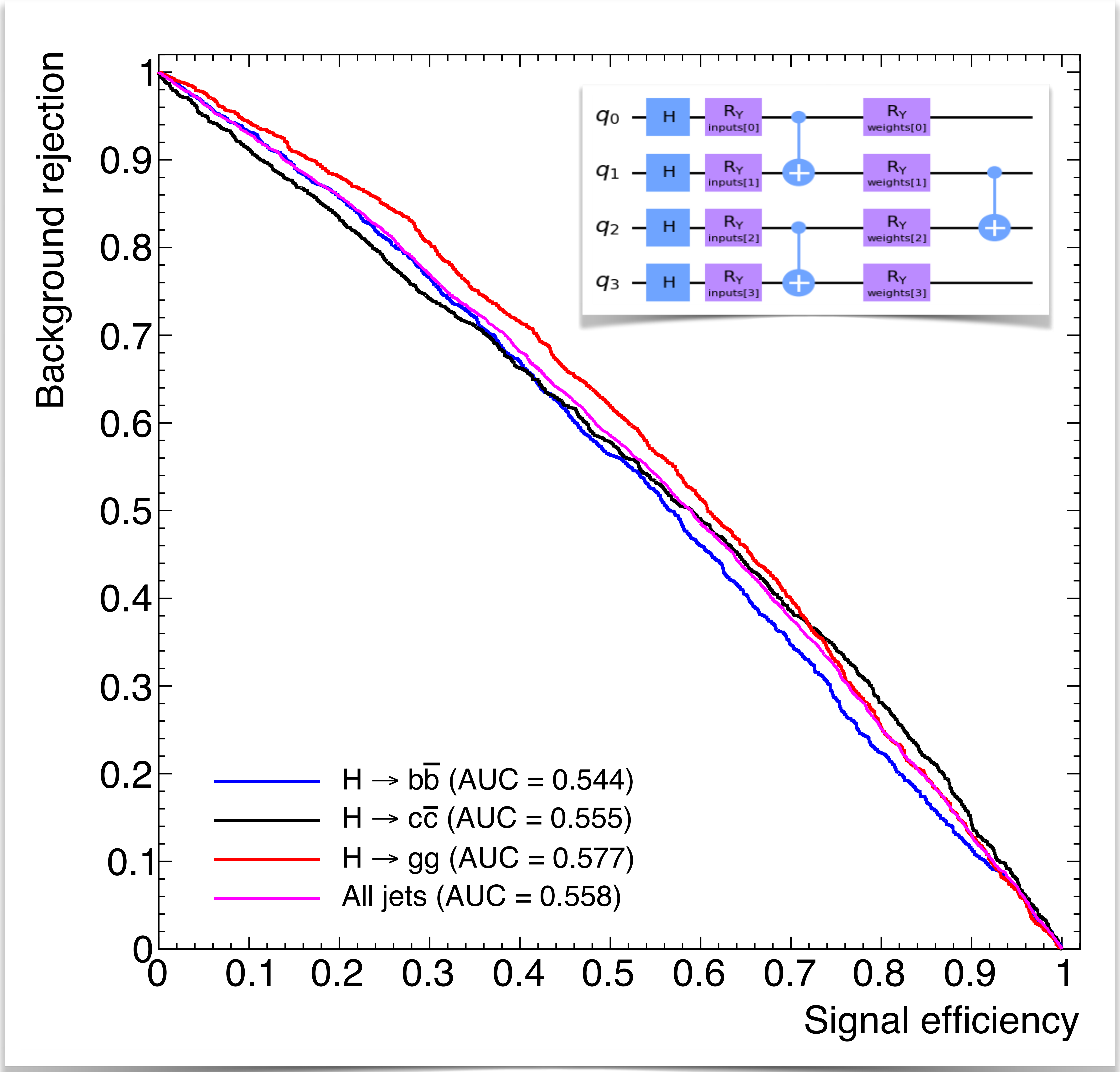
- The trainable parameters are added using the Anzatz with a feature-map that acts as an encoder.

Results: ROC comparing ParT and QParT

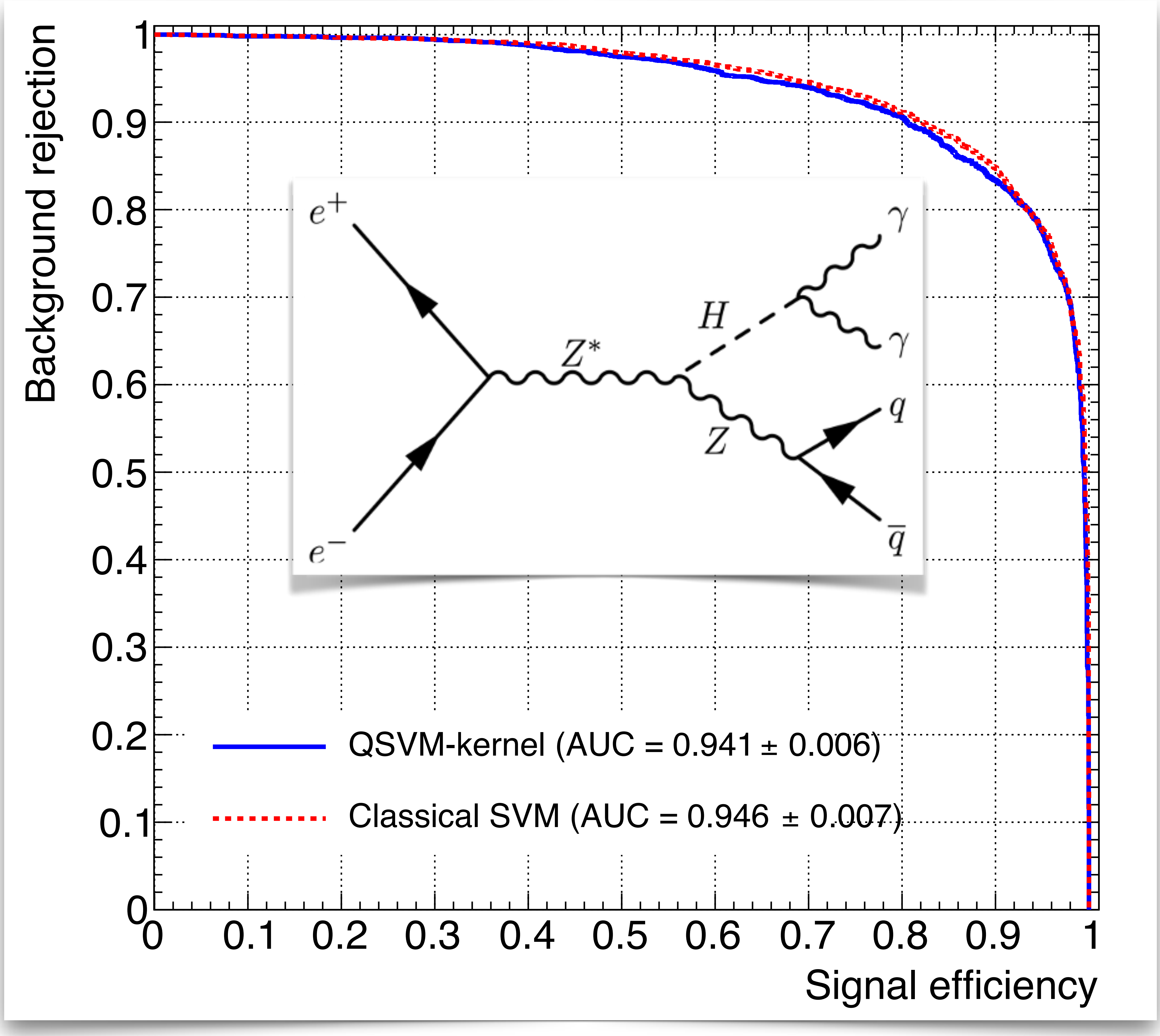
- Three classes were used with 10 particles:
  - $H \to b\bar{b}, H \to c\bar{c}$ and $H \to gg$.
- Network configurations:
  - Pair input dim: 4
  - Two heads and one layer.

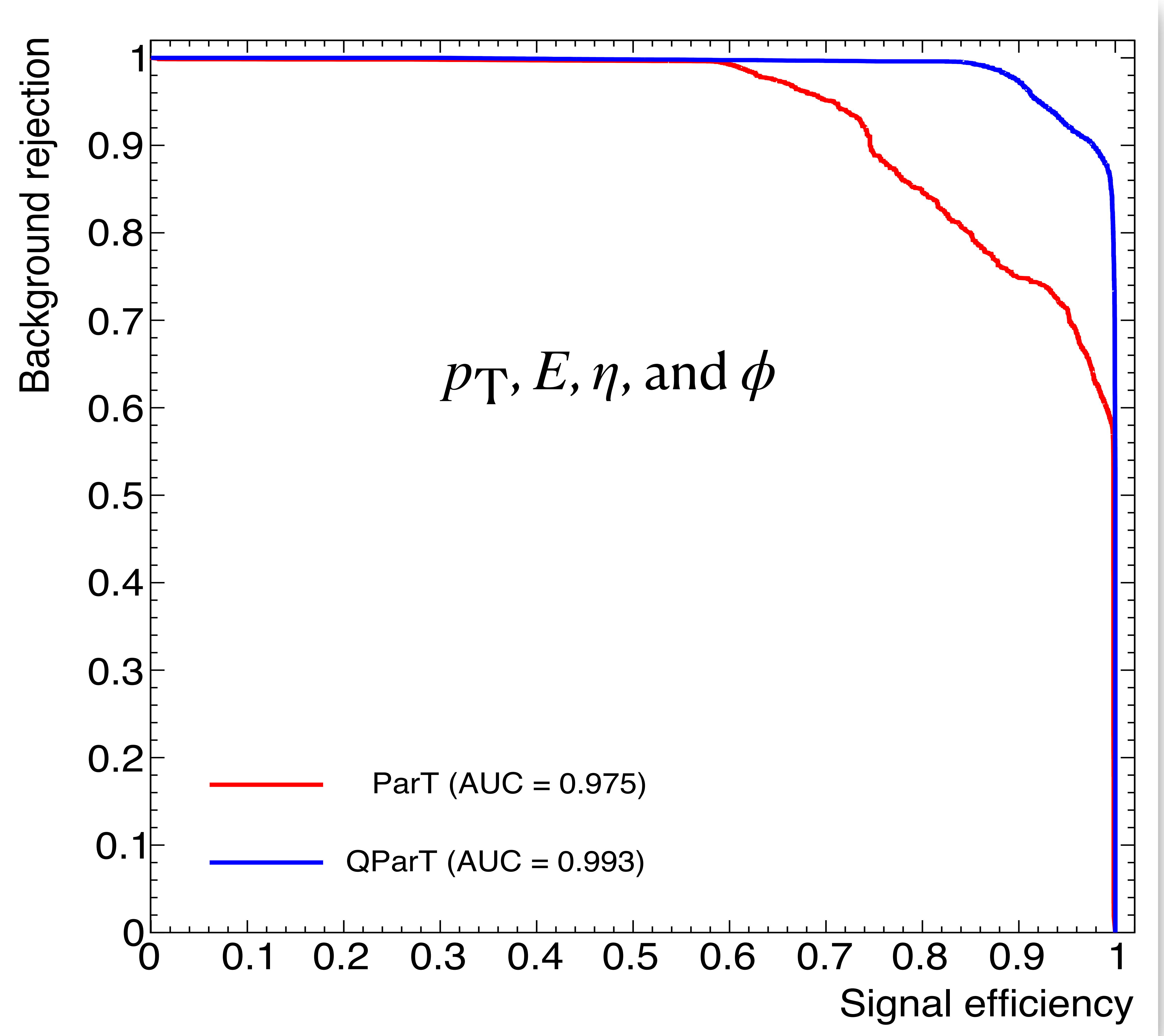


- Training and testing size: 2000 entries for each class.
- Both are not better than the random classifier.



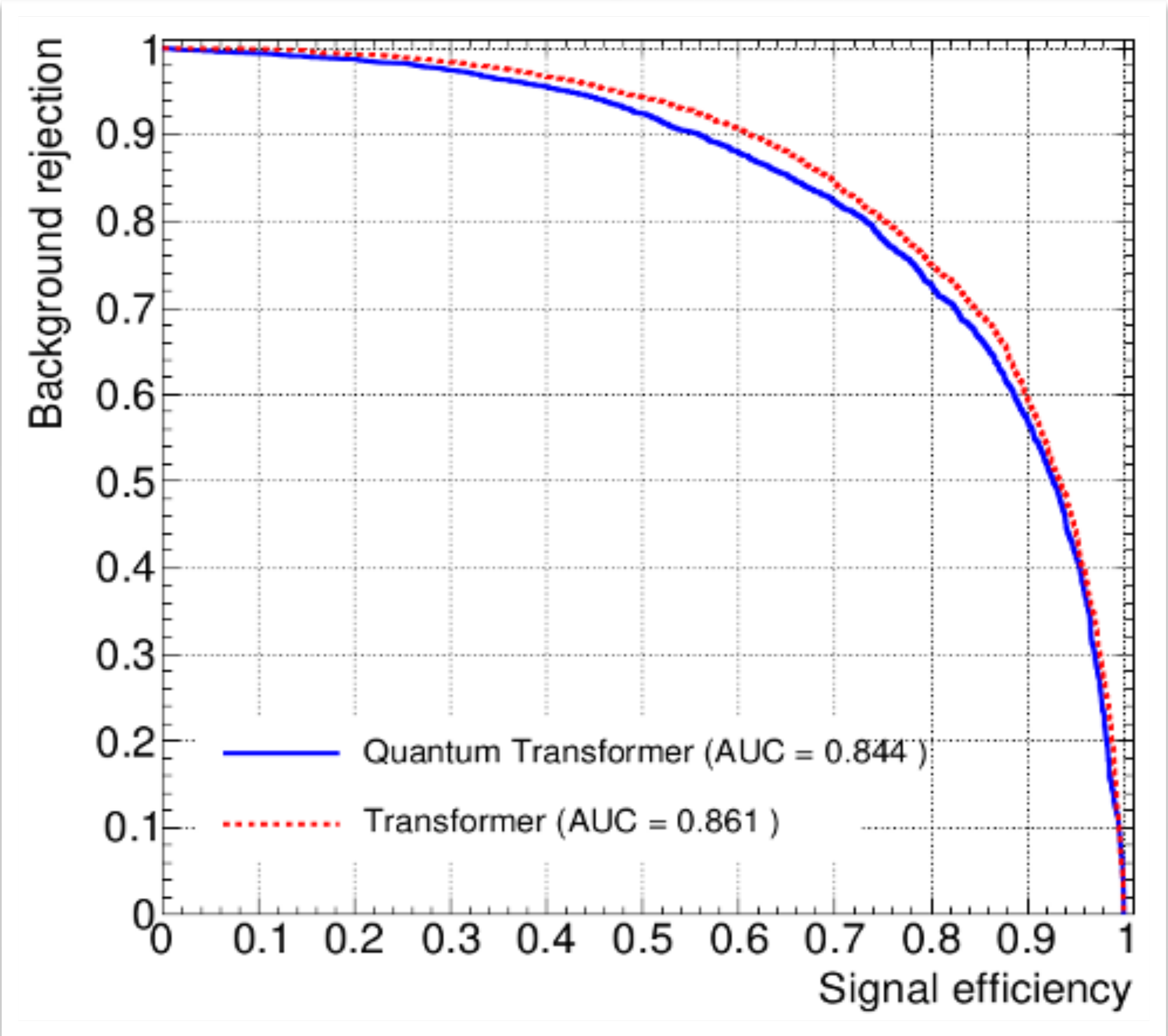
Particle Transformer: signal & background classification

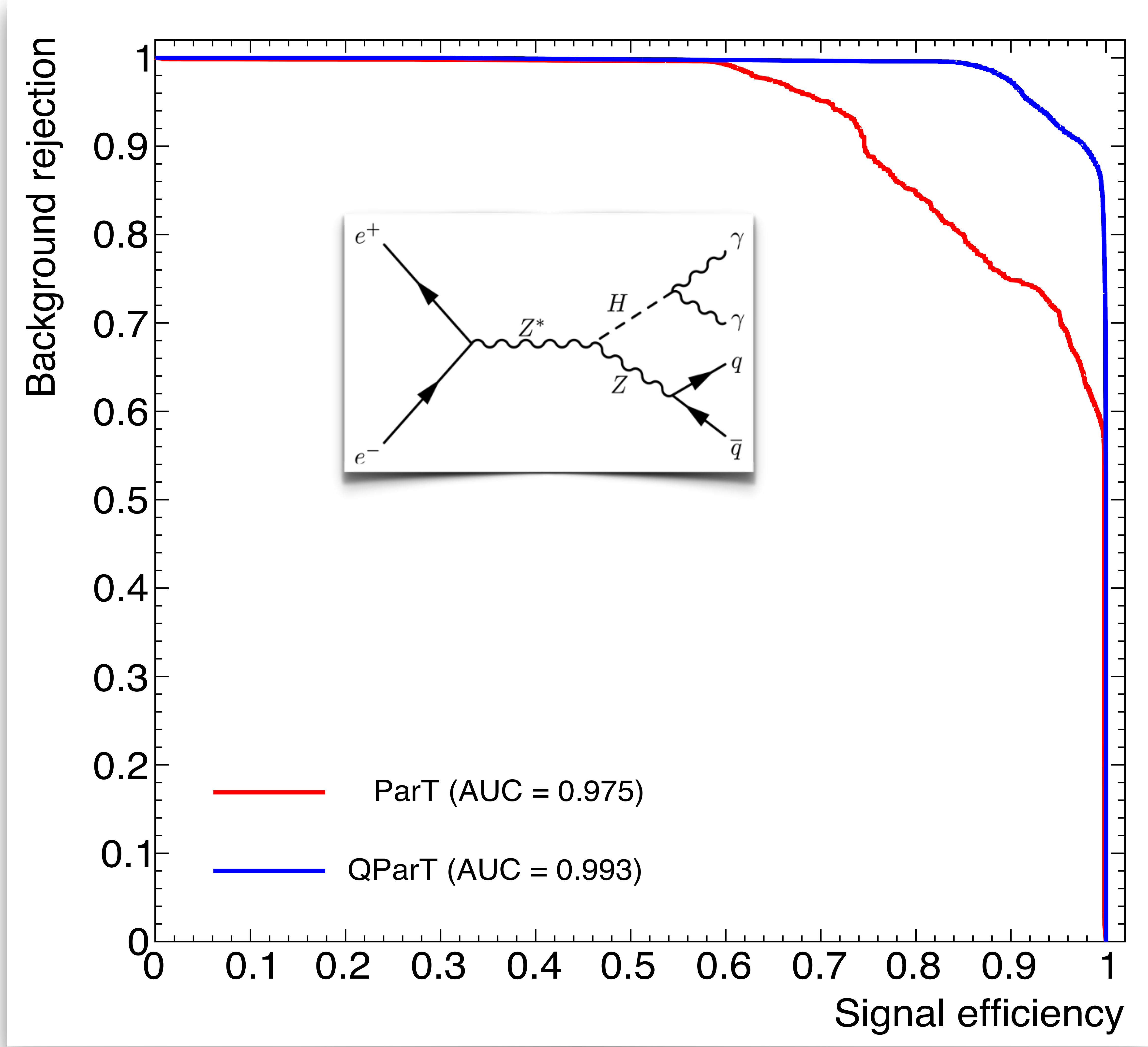




- The SVM result is on the left, and the particle transformer is on the right.

Particle Transformer: signal & background classification





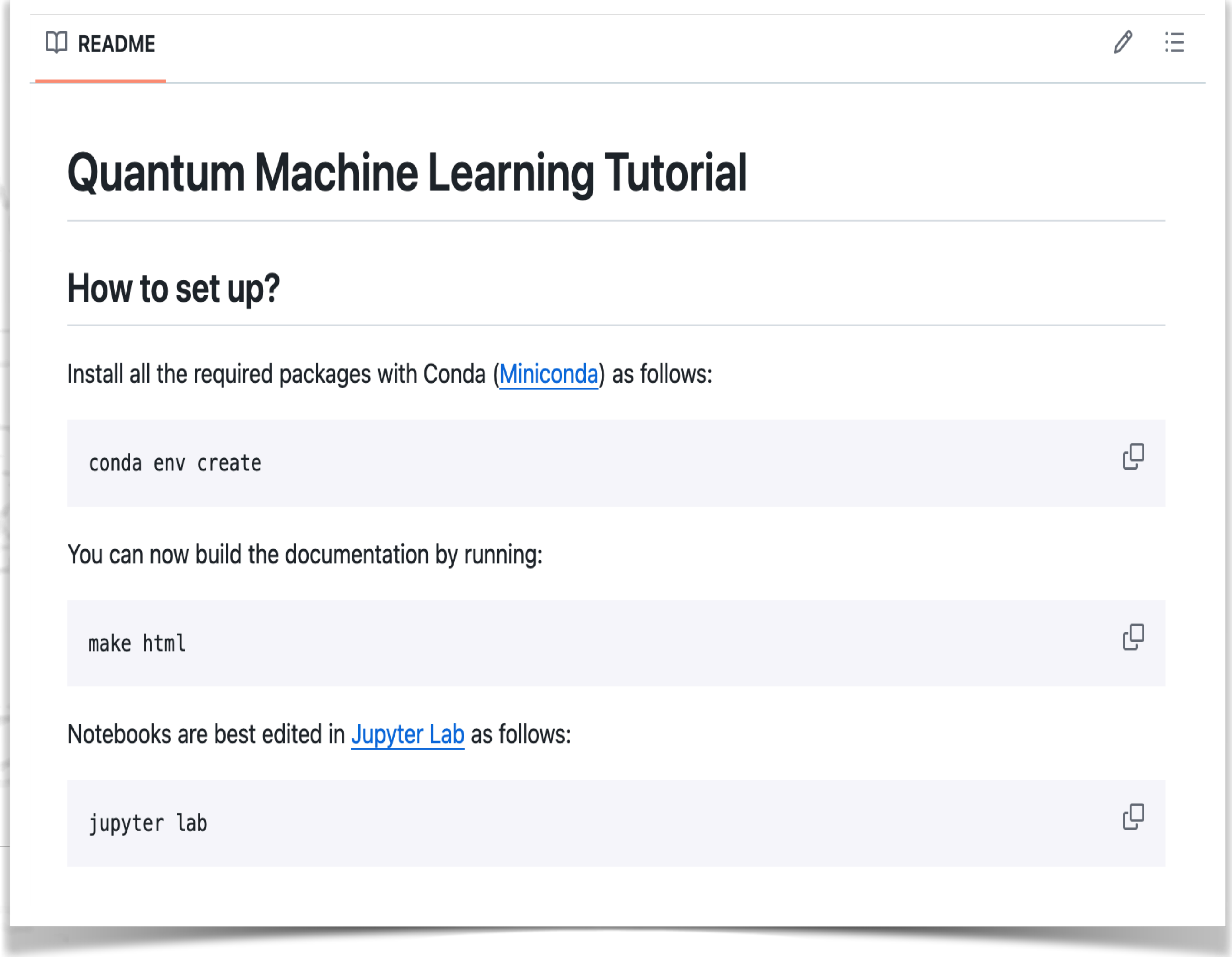
- The PennyLane result is on the left, and the Qiskit is on the right for the transformer implementation.

Summary

- Provided a quick overview of the basic idea behind Quantum Machine Learning and
- showed, as an example, the support-vector machine:
  - A similar performance between classical and quantum was obtained.
  - Study the noise effect with a simplified model.
- Particle transformer is a bit complicated with all the self-attention added to it.
- We constructed a quantum self-attention based on a quantum neural network.

Practical Tutorial

- Check out the GitHub repository [Quantum-machine-learning-in-HEP](). You can follow the instructions there to get the package.
- To run the Quantum Particle Transformer:
  - source /hpcfs/cepc/higgsgpu/amohammed/miniconda3/etc/profile.d/conda.sh
  - conda activate QParT
  - cp -r /hpcfs/cepc/higgsgpu/amohammed/QCWork/QParT .
  - ./train_JetClass.sh QParT kin (make sure you disable the GPU)
  - To run on GPUS: you need to look at this line submit_to_gpus.sh.
  - Then do: source submit_to_gpus.sh



README

Quantum Machine Learning Tutorial

How to set up?

Install all the required packages with Conda (Miniconda) as follows:

```
conda env create
```

You can now build the documentation by running:

```
make html
```

Notebooks are best edited in Jupyter Lab as follows:

```
jupyter lab
```