

# EfficientML.ai Lecture 22

## Quantum Machine Learning

Part I



**Hanrui Wang**

PhD Student, MIT



@hanrui\_w



# Lecture Plan

Today we will:

1. Continue to introduce NISQ devices
2. Introduce Parameterized Quantum Circuit (PQC)
3. Introduce PQC Training
4. Introduce Quantum Classifiers
5. Introduce Noise Aware On-Chip Training (QOC) of PQC
6. Introduce TorchQuantum Library for QML

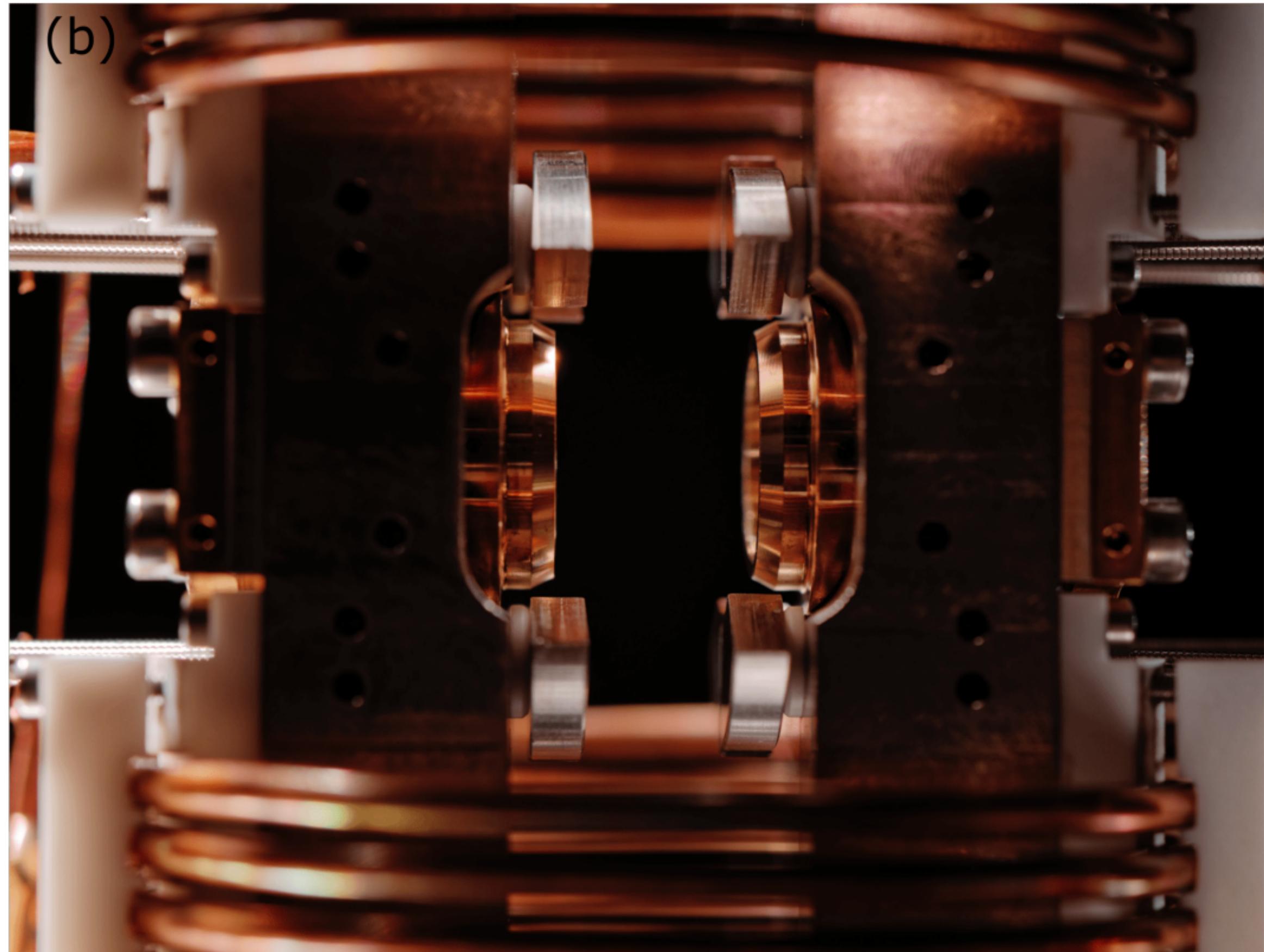
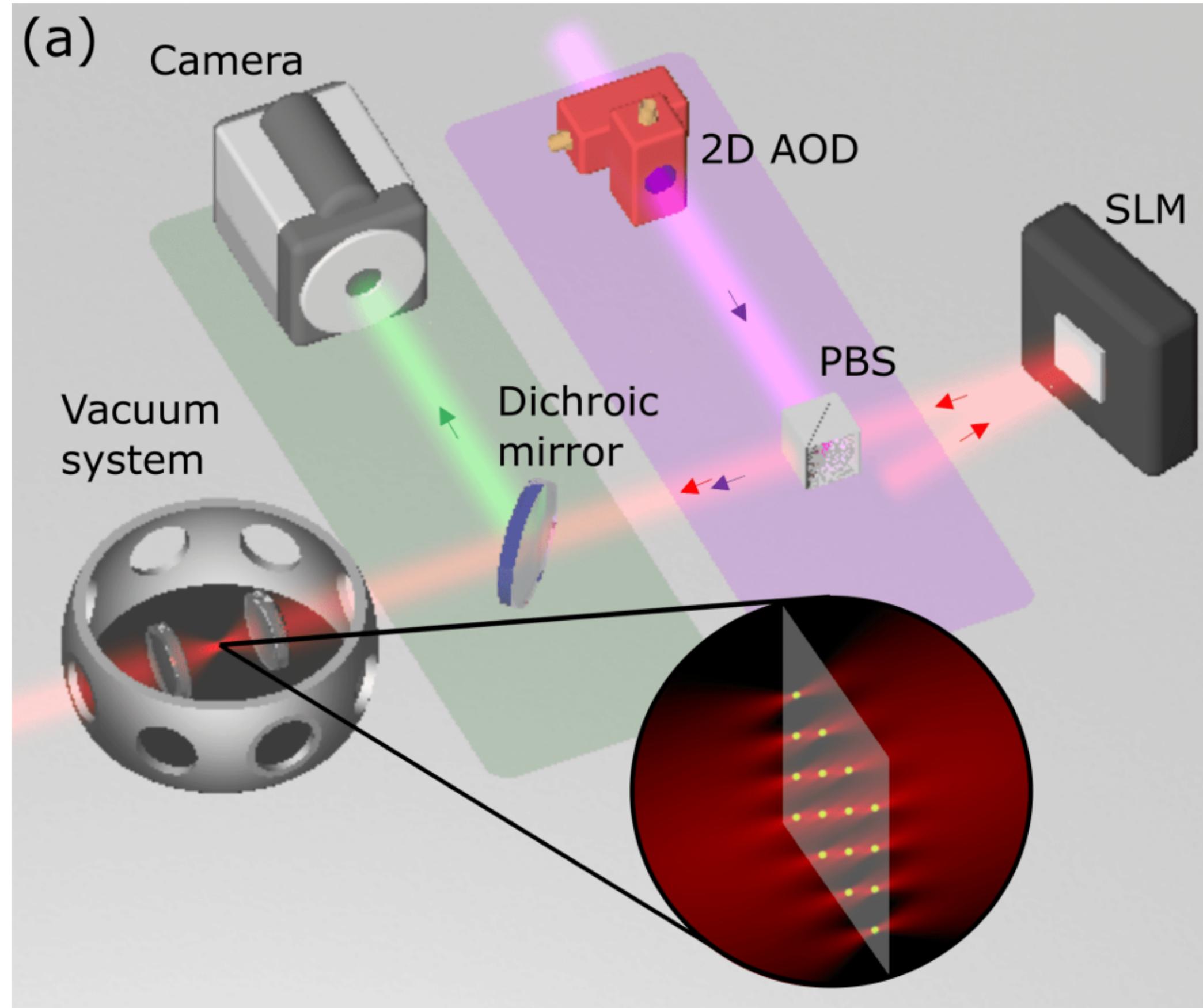
# Section 1

## NISQ Devices (Cont.)

# Neutral Atom Devices

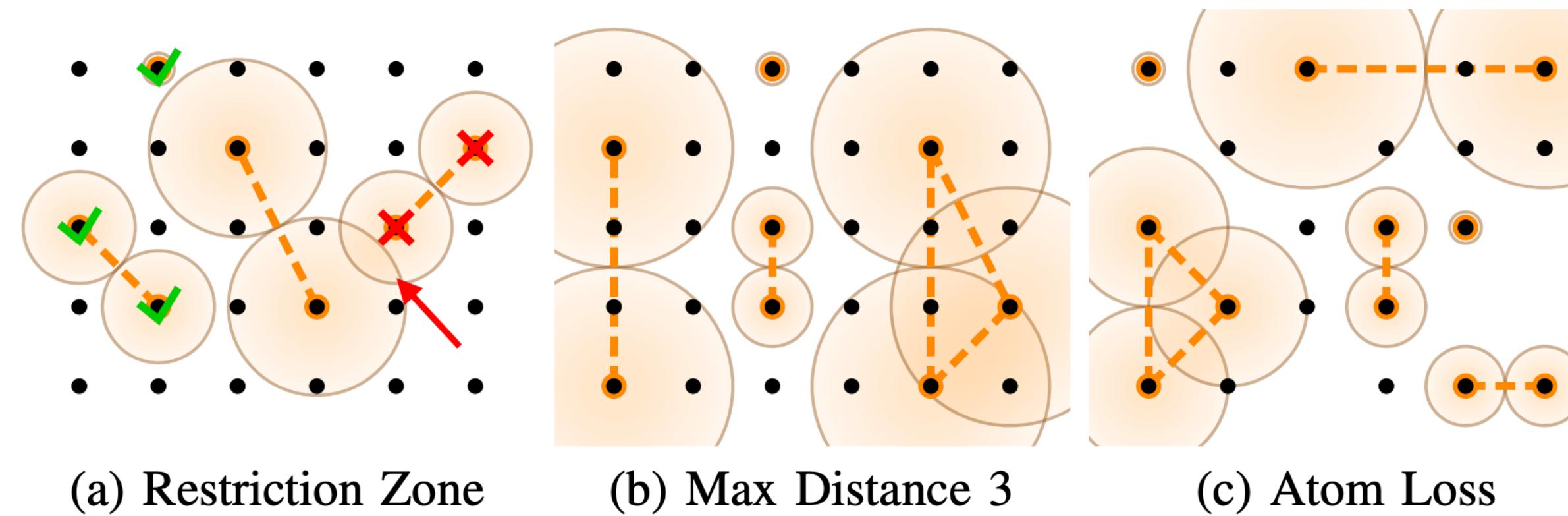
SLM: spatial light modulator

AOD: acousto-optic deflector

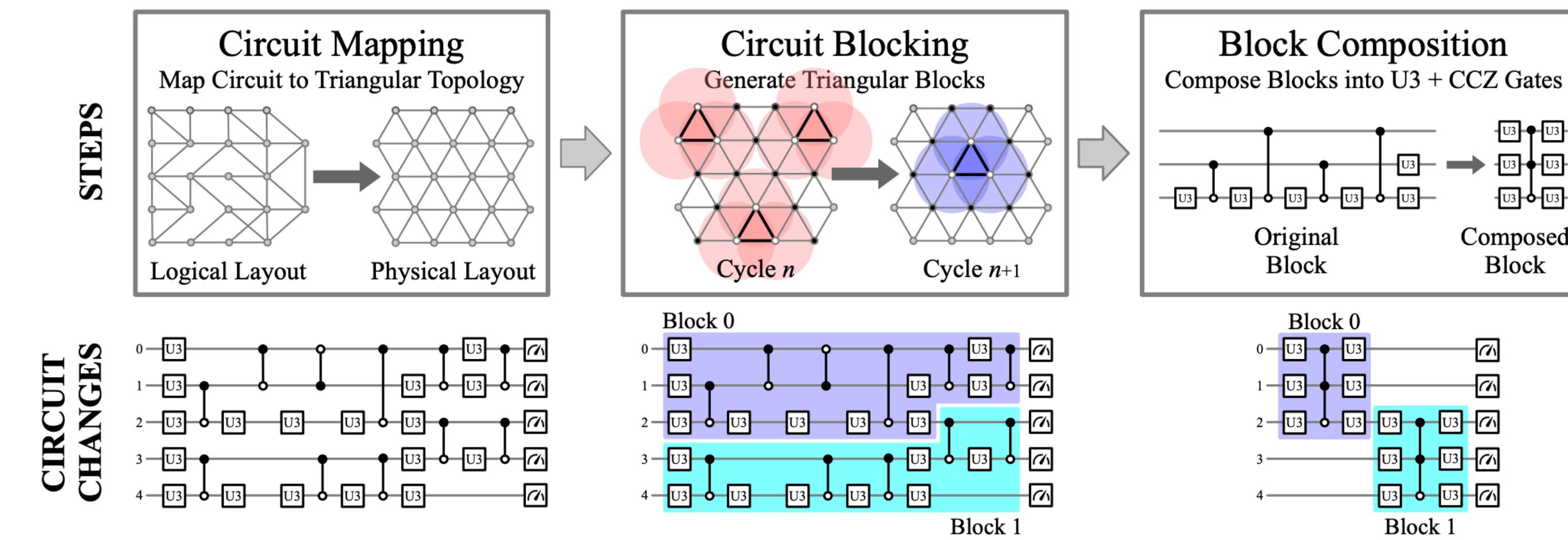


Quantum computing with neutral atoms.

# Fixed Atom Array

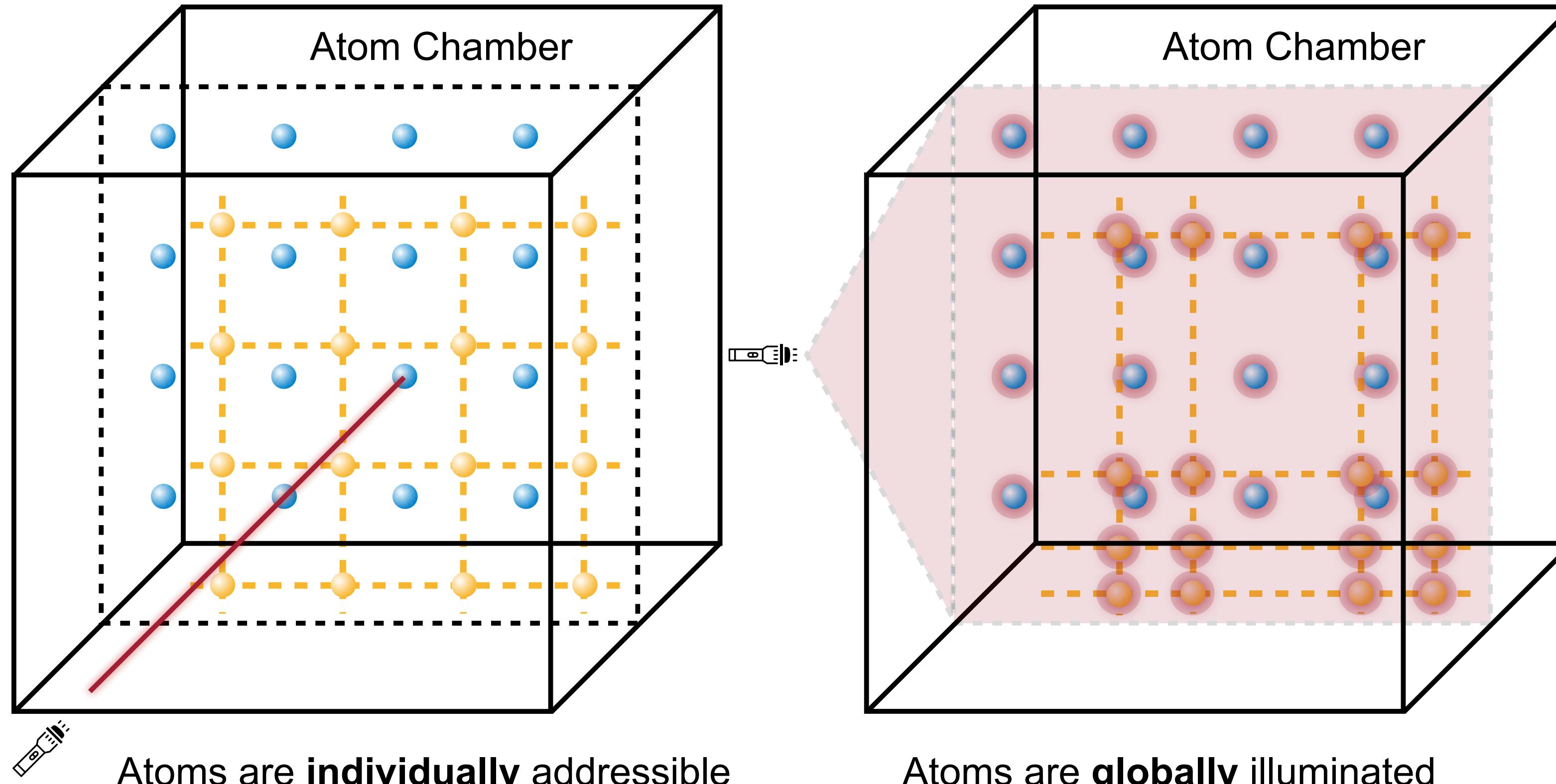


Baker, Jonathan M., et al. "Exploiting long-distance interactions and tolerating atom loss in neutral atom quantum architectures." *ISCA 2021*



Patel, Tirthak, Daniel Silver, and Devesh Tiwari. "Geyser: a compilation framework for quantum computing with neutral atoms." *ISCA 2022*.

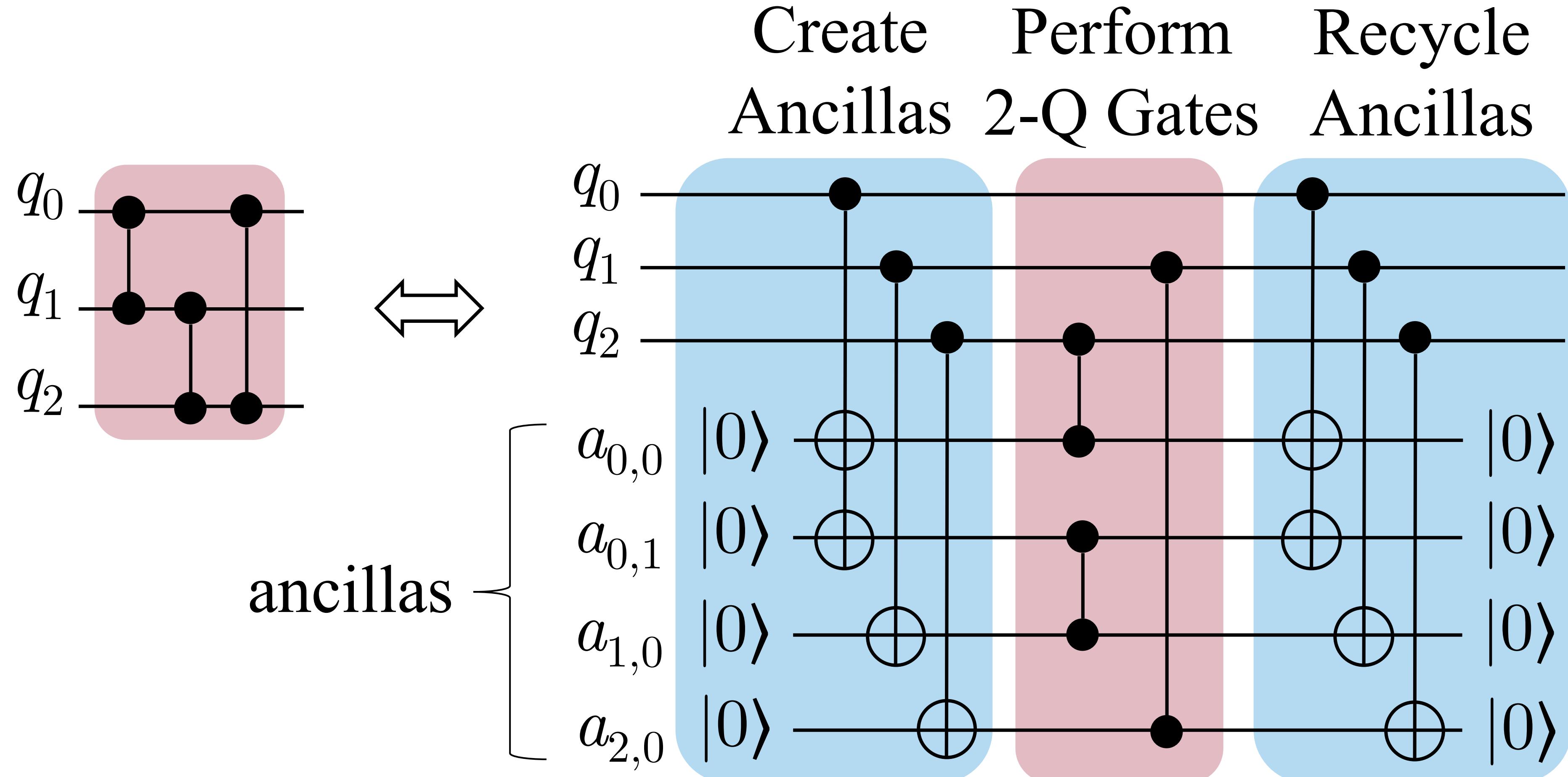
# Movable Atom Array



Atoms are **individually** addressable  
with Raman laser (**perpendicular**  
to the atom plane) to perform  
**single-qubit** gates

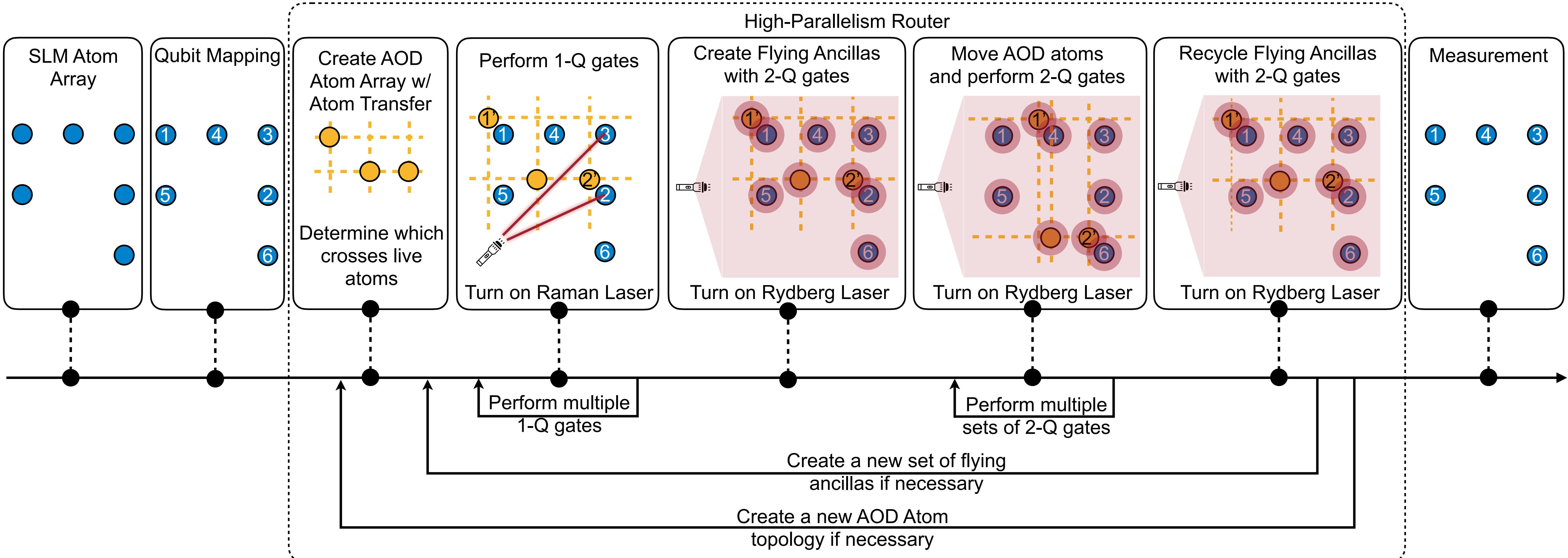
Atoms are **globally** illuminated  
with Rydberg laser (**parallel** to  
the atom plane) to be perform  
**two-qubit** gates

# Flying-Ancilla based Compilation

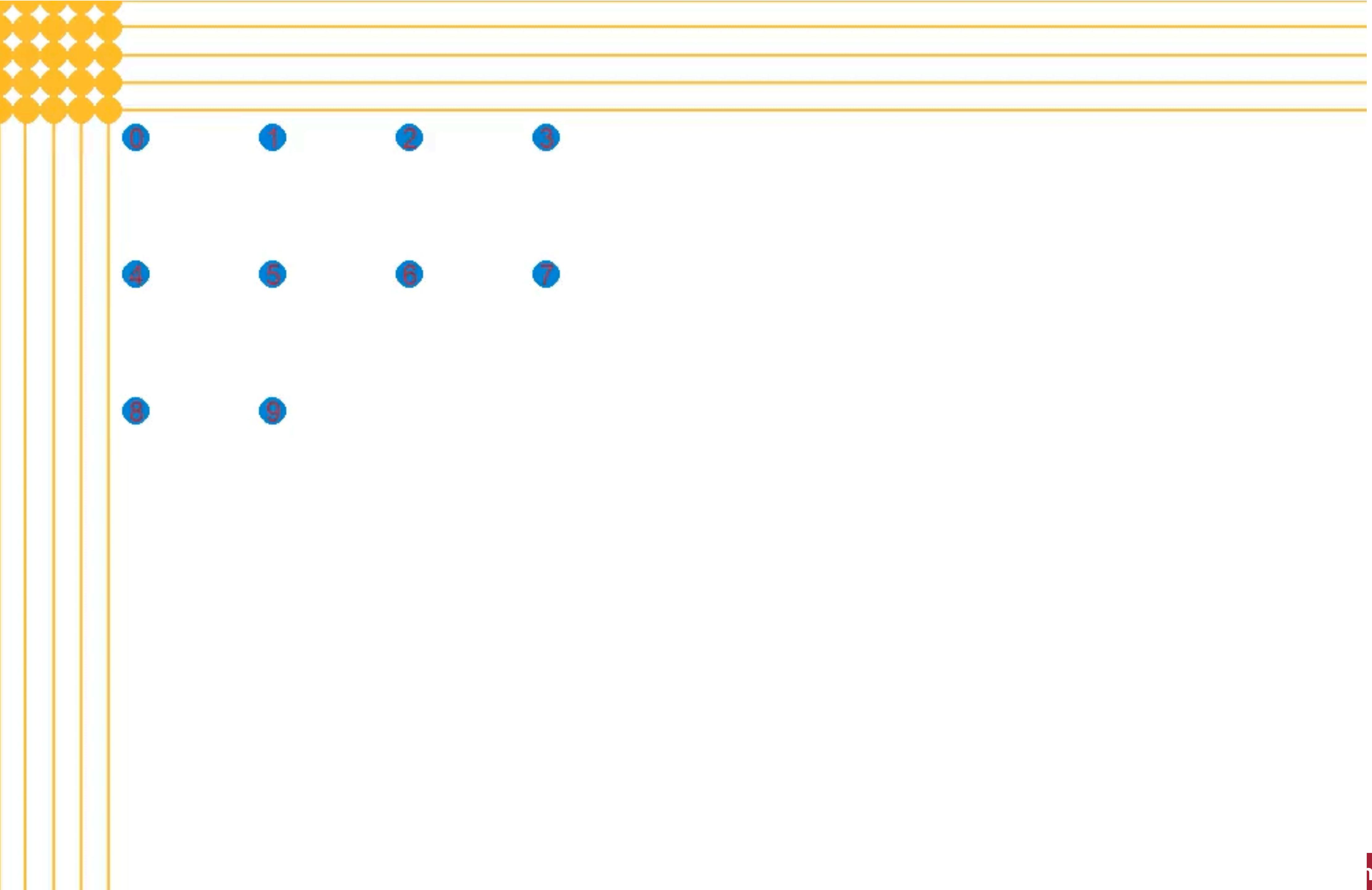


Q-Pilot: Field Programmable Quantum Array Compilation with Flying Ancillas

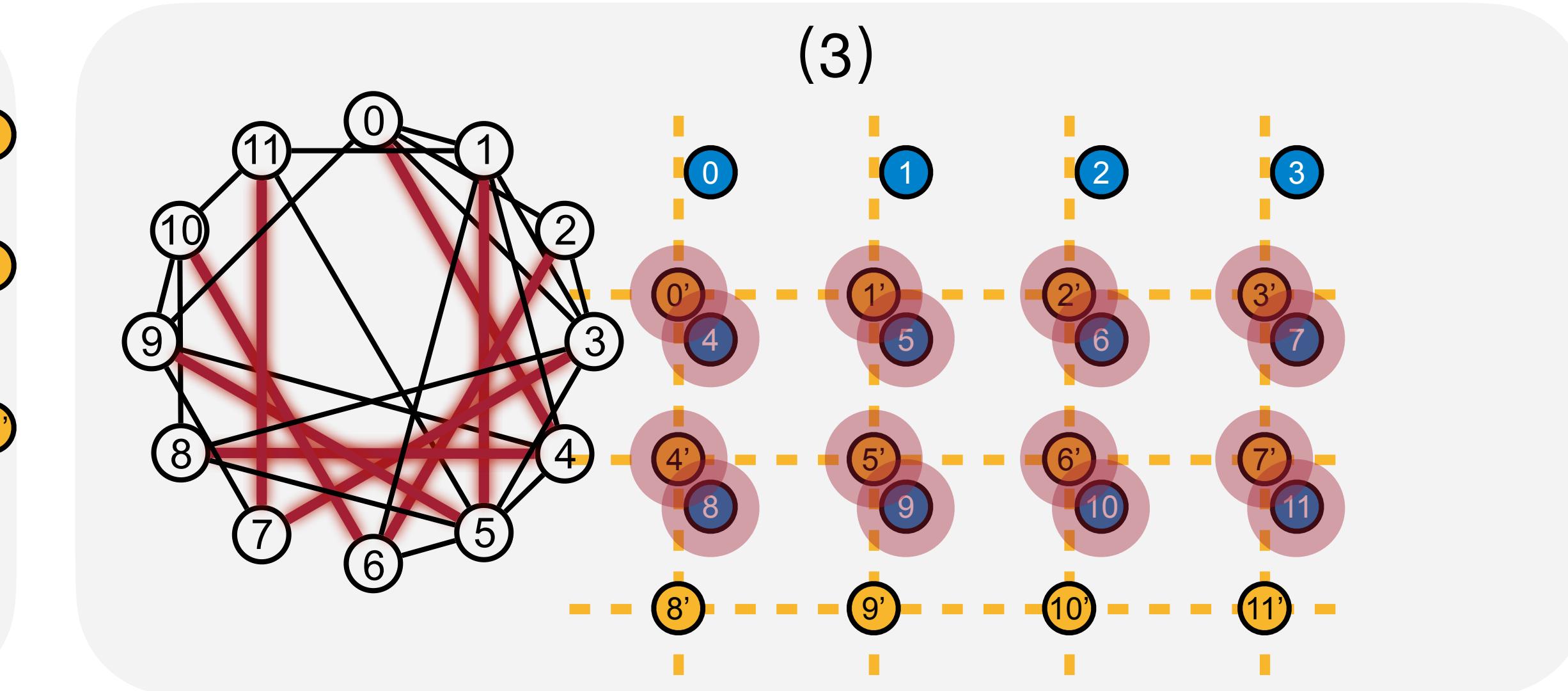
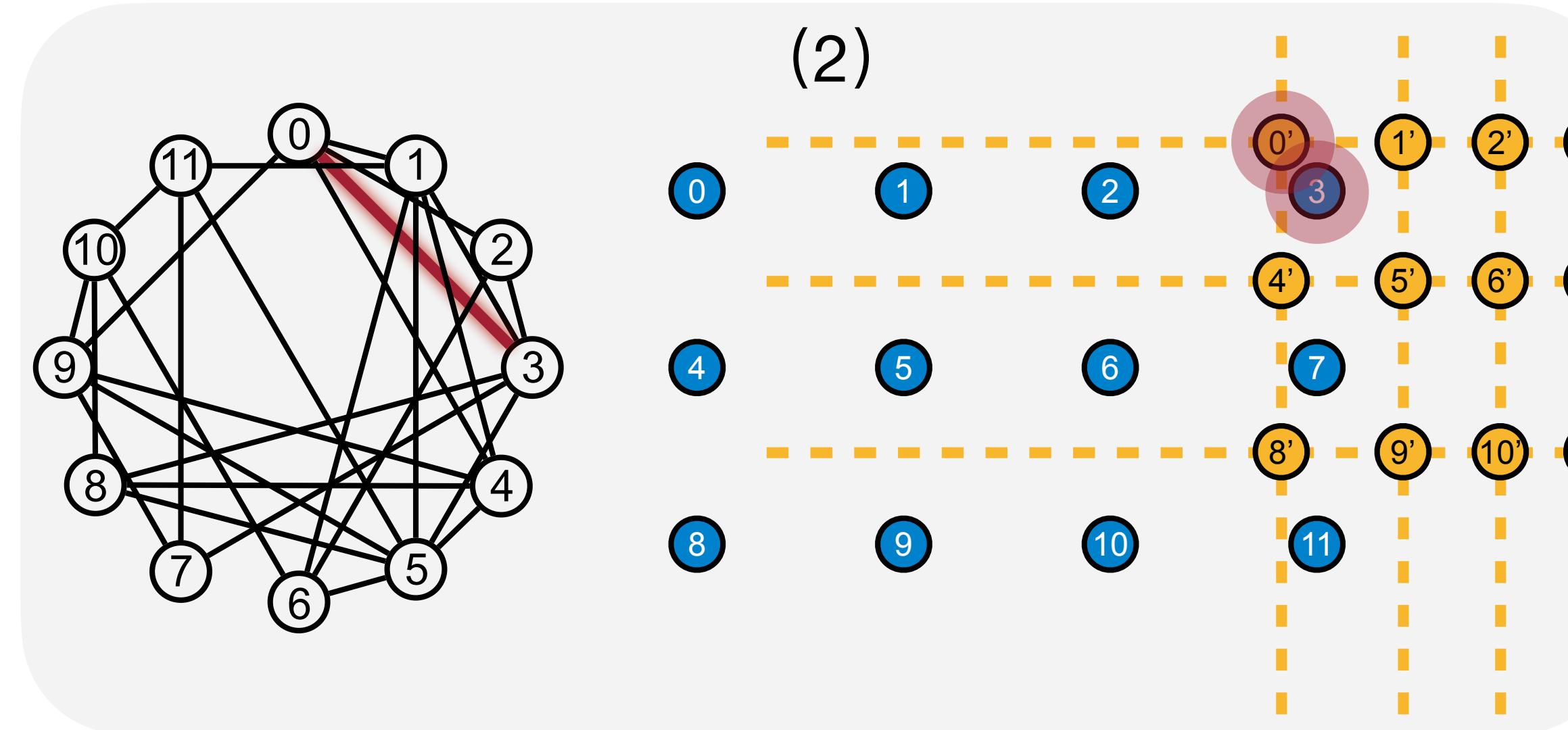
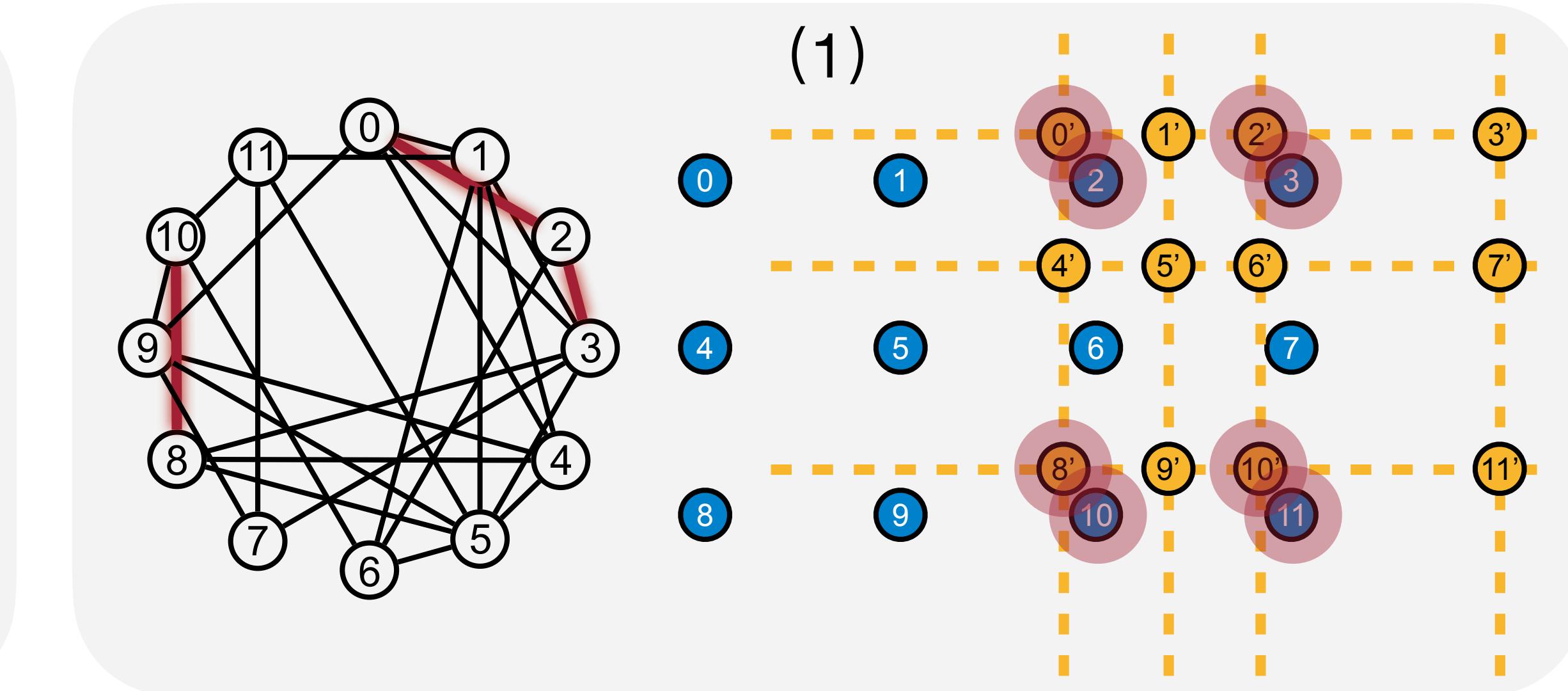
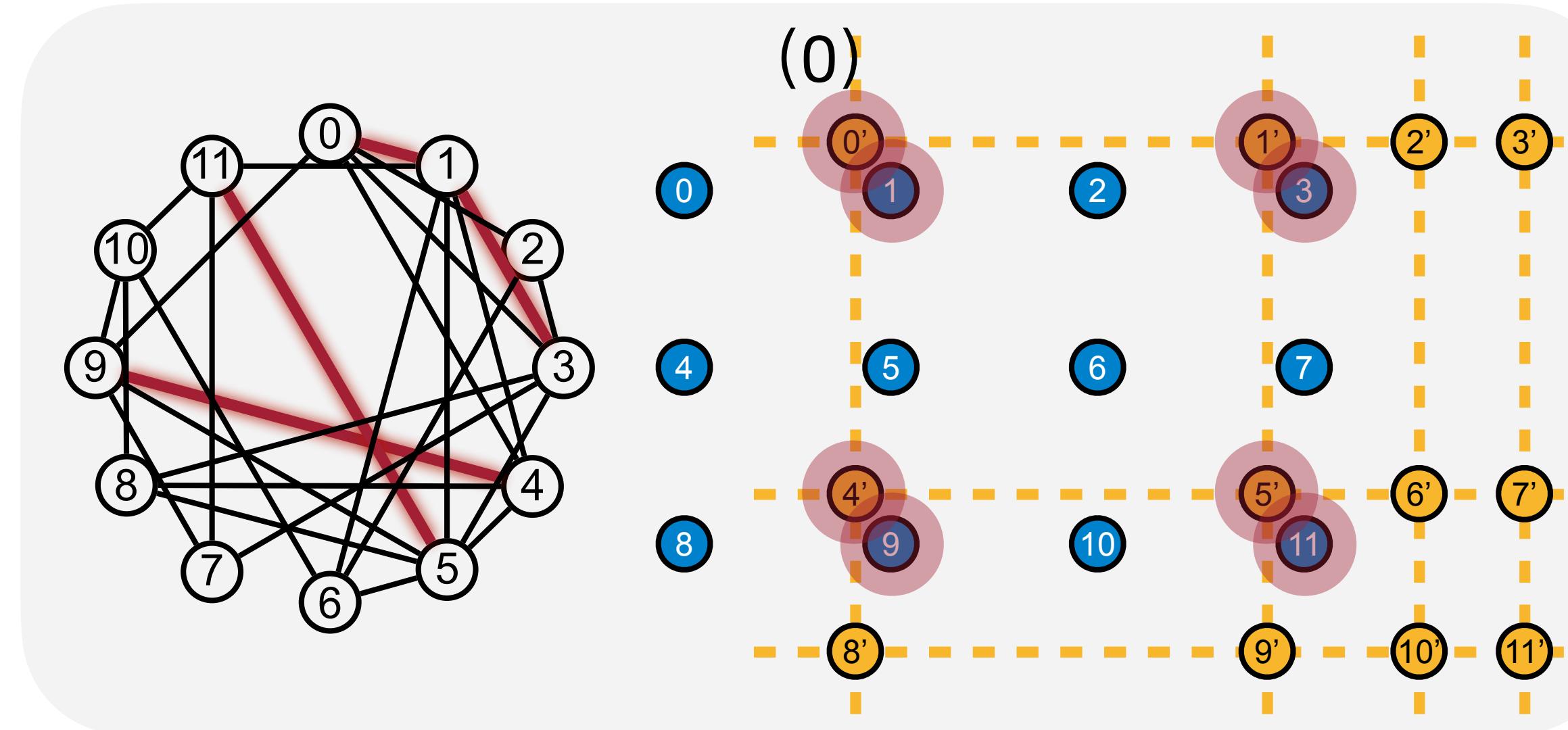
# Flying-Ancilla based Compilation



Q-Pilot: Field Programmable Quantum Array Compilation with Flying Ancillas

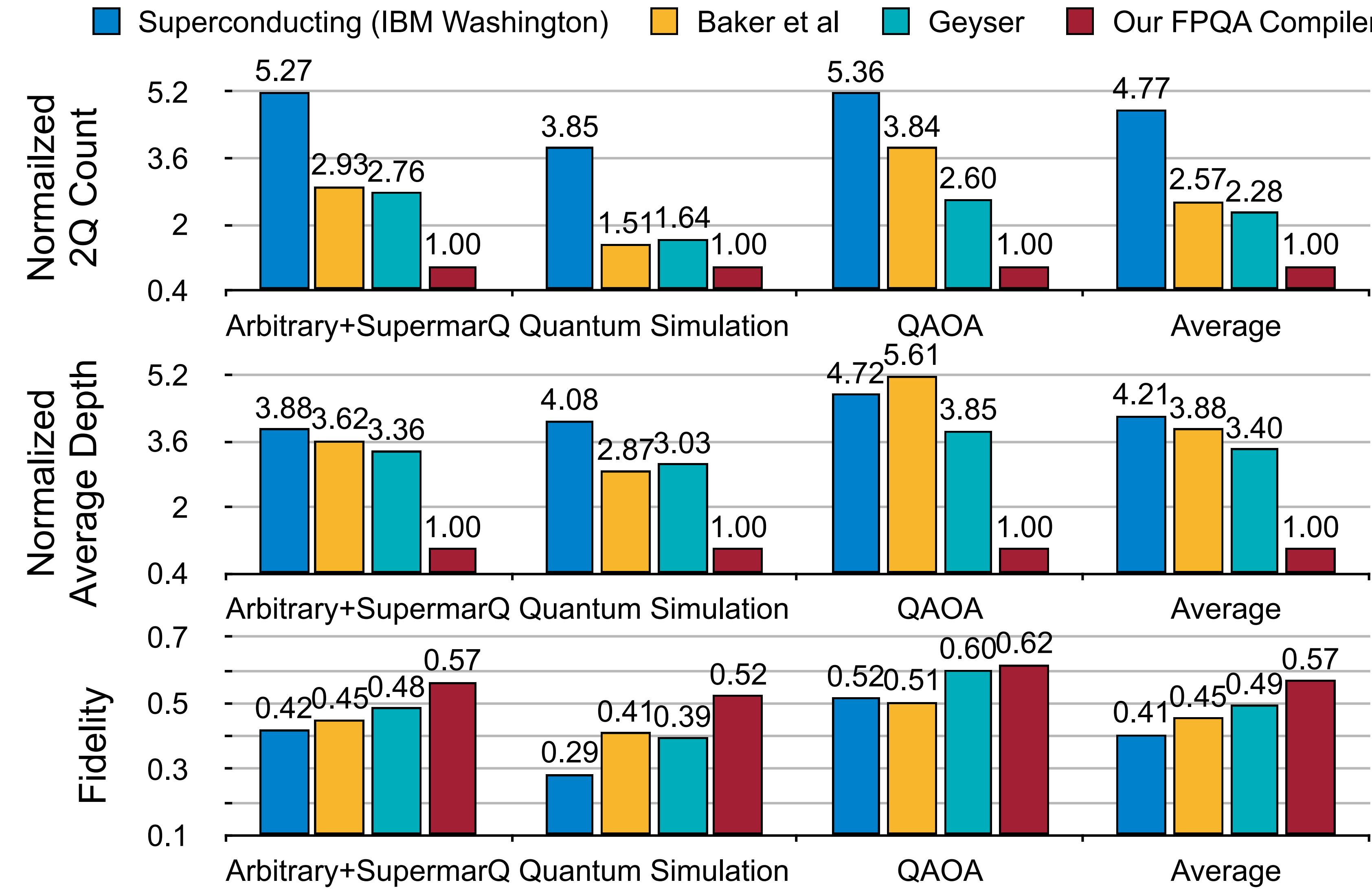


# QAOA Example



Q-Pilot: Field Programmable Quantum Array Compilation with Flying Ancillas

# Compare with Fixed Arrays



Q-Pilot: Field Programmable Quantum Array Compilation with Flying Ancillas

# Section 2

## Parameterized Quantum Circuit

# QML Approaches

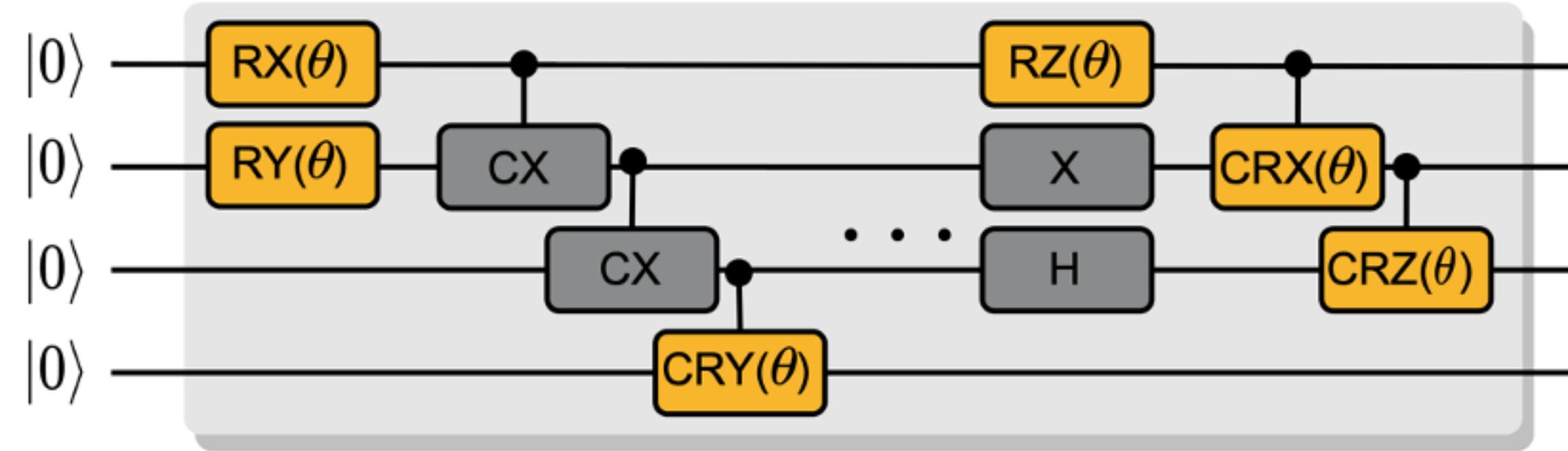
- CC: Classical Data, Classical algorithm: what we have learned from previous lectures
- CQ: Classical Data, Quantum algorithm: what we are going to introduce
- QC: Quantum Data, Classical algorithm: qubit control, calibration, readout...
- QQ: Quantum Data, Quantum algorithm: process quantum info with quantum machine

Type of algorithm	
Type of data	
CC	CQ
QC	QQ

<https://qiskit.org/learn/>

# Parameterized Quantum Circuits

- Circuit that contains fixed gates and parameterized gates



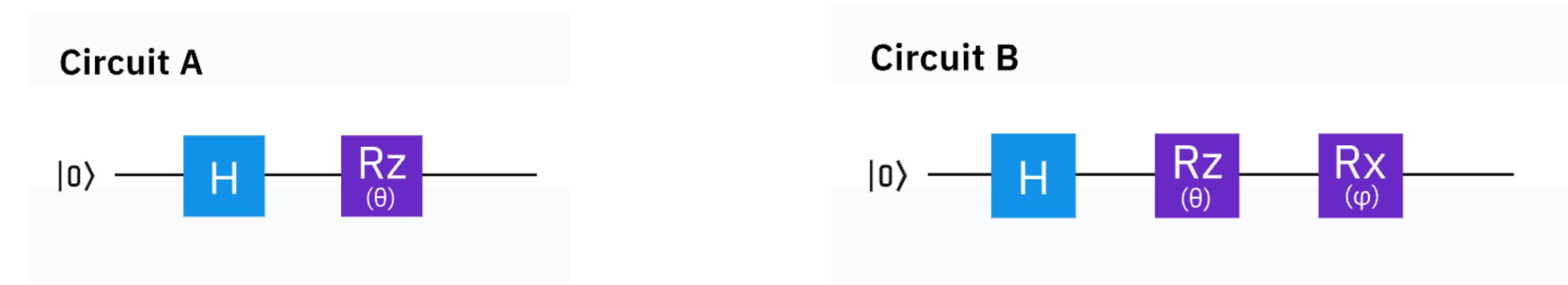
Input	Output			
	00	01	10	11
00	1	0	0	0
01	0	1	0	0
10	0	0	0	1
11	0	0	1	0

$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$

$CRX(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \cos \frac{\theta}{2} & -i \sin \frac{\theta}{2} \\ 0 & 0 & -i \sin \frac{\theta}{2} & \cos \frac{\theta}{2} \end{bmatrix}$

# Expressivity

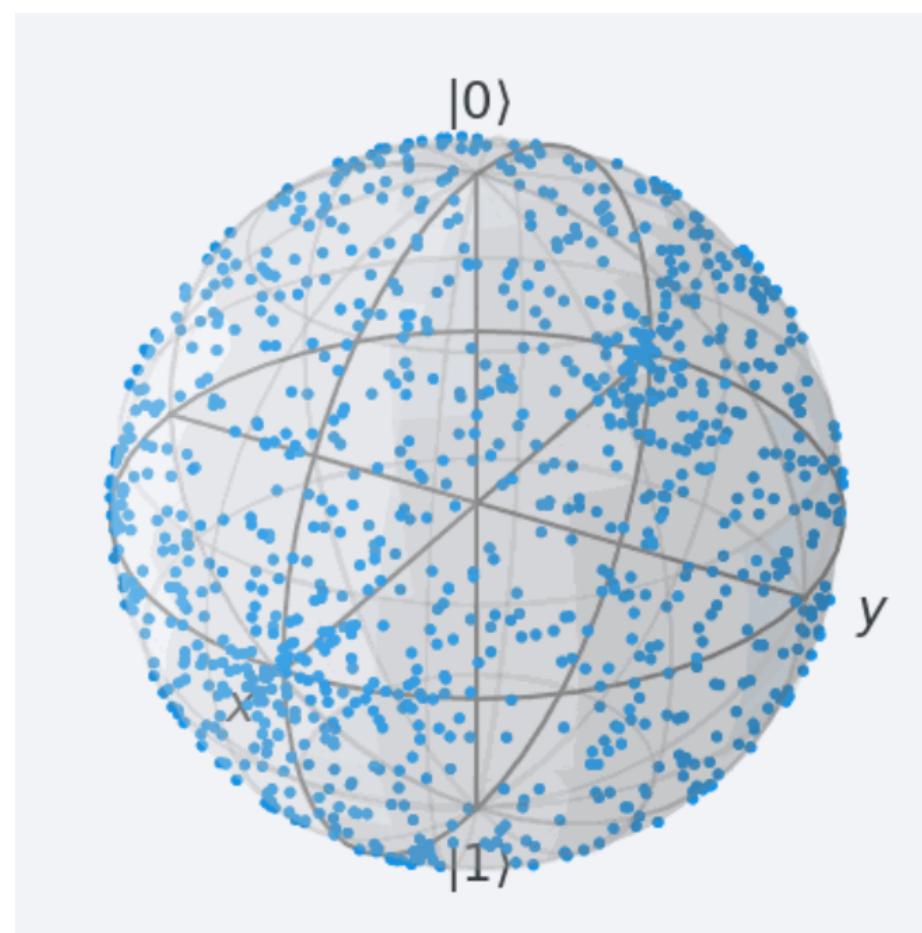
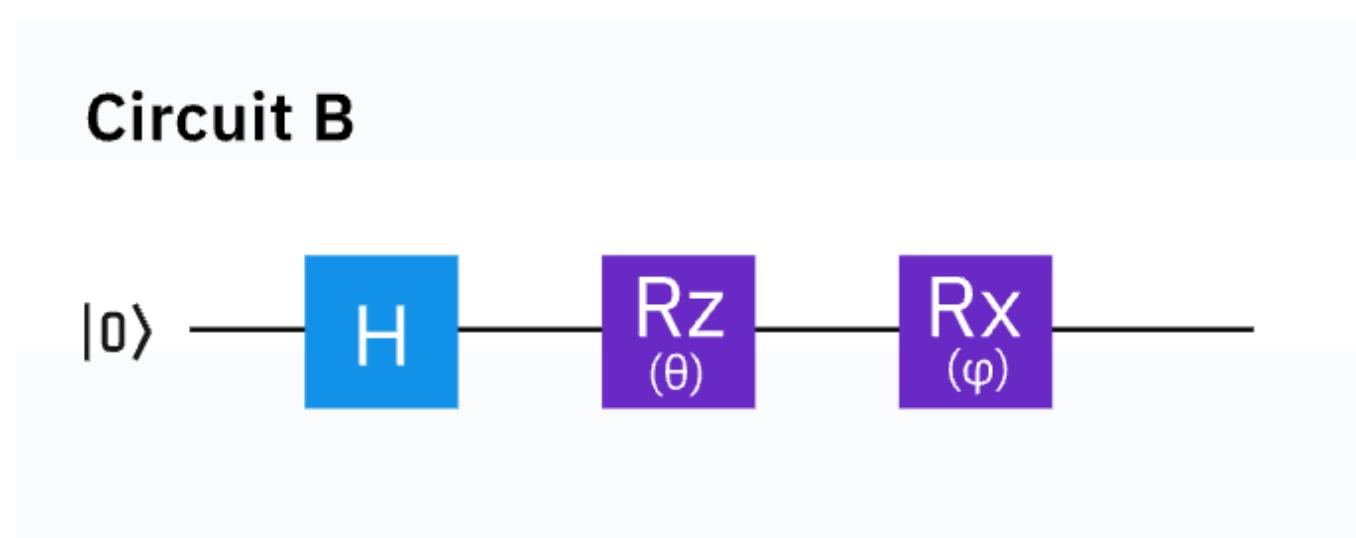
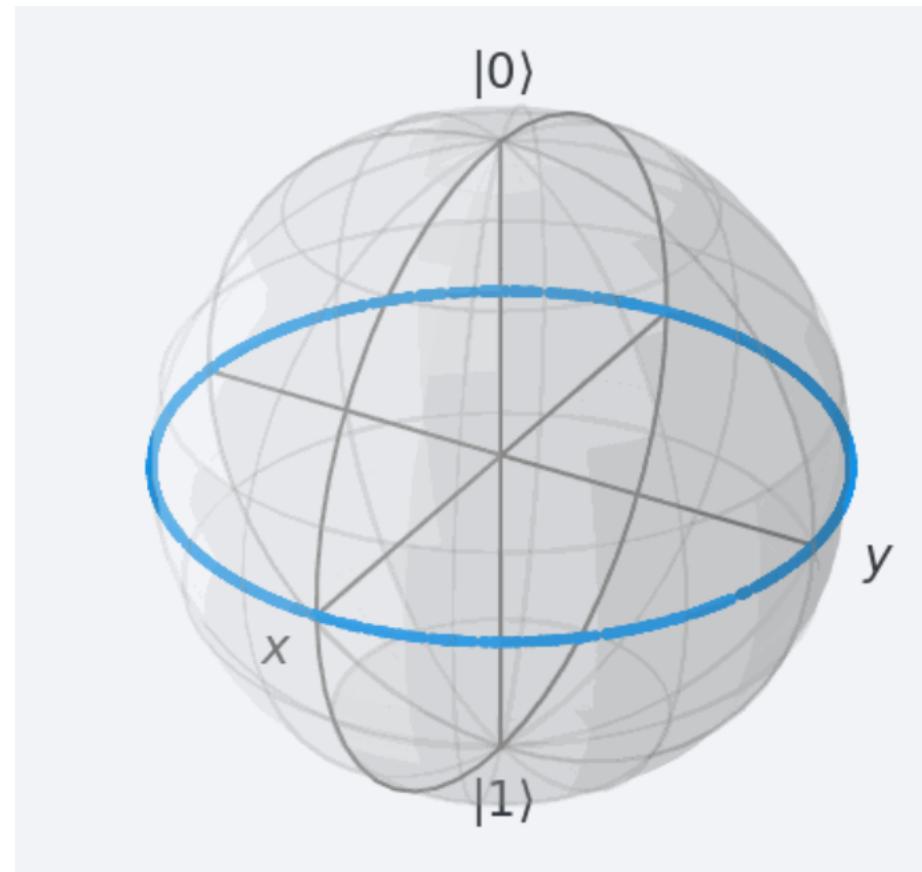
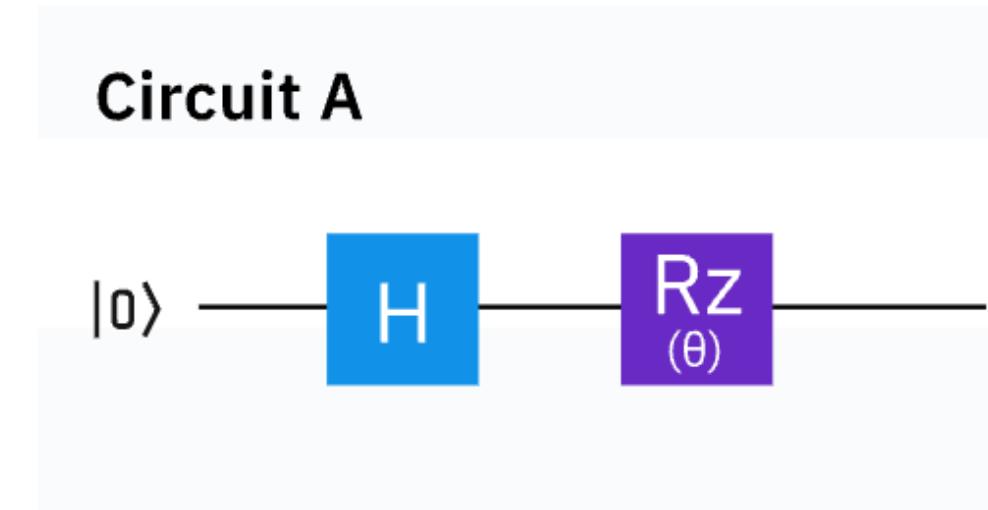
- How the quantum circuit can cover the **Hilbert space**?
- The extent to which the states generated from the circuit deviate from the **uniform** distribution
- Which circuit has **better expressivity** below?



<https://qiskit.org/learn/>

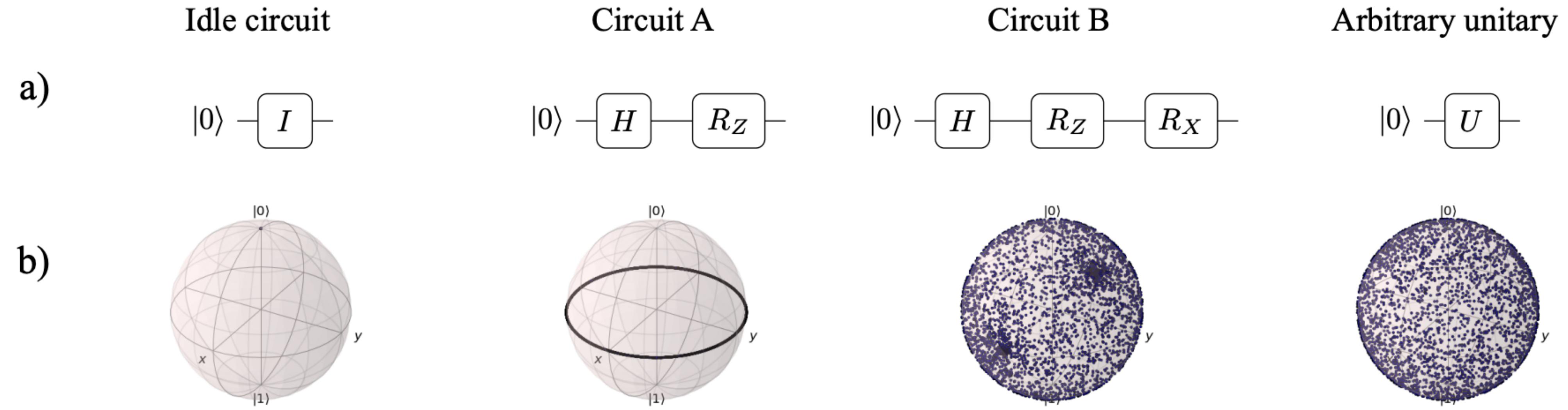
# Expressivity

- Circuit B has better **coverage** of Hilbert Space



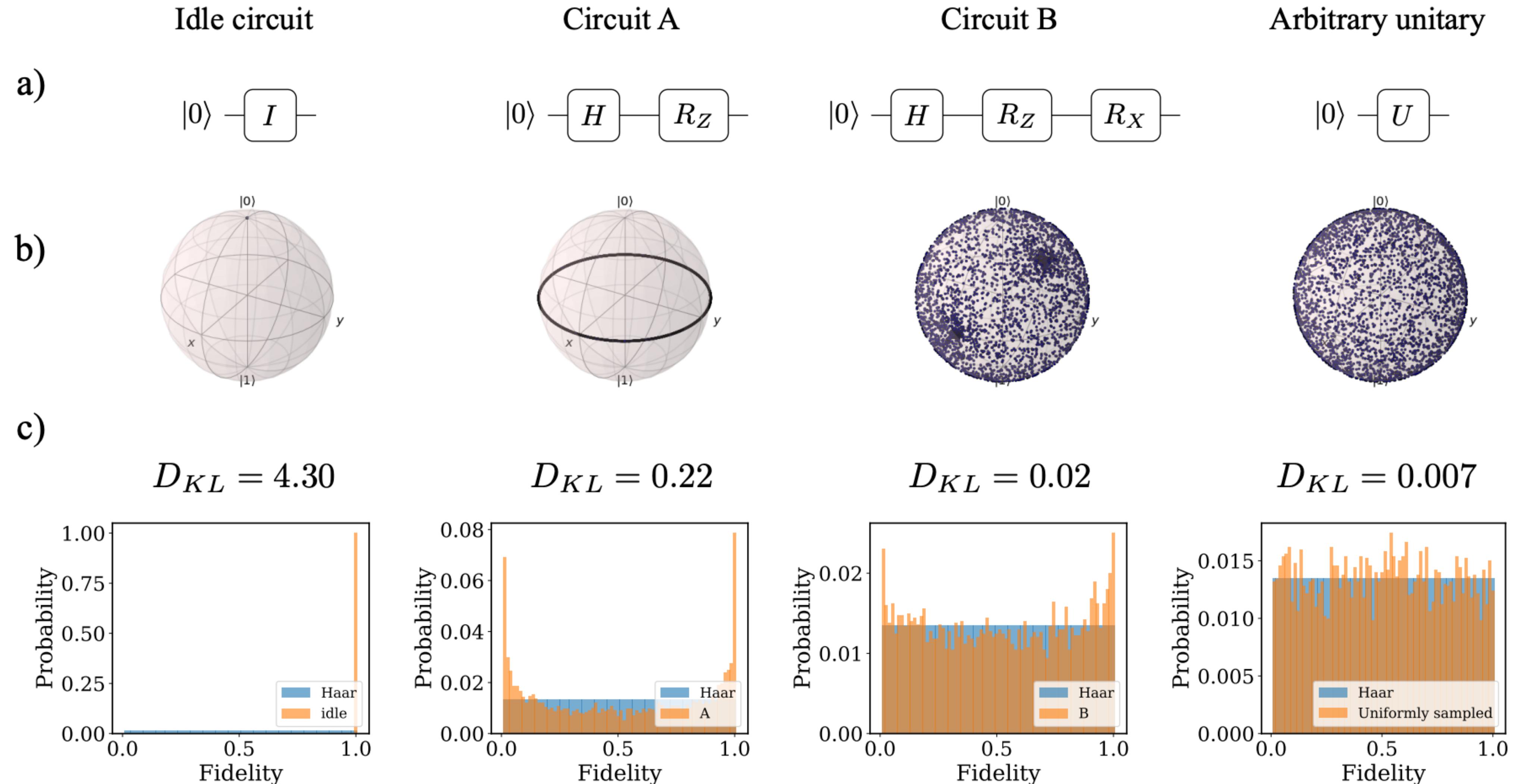
<https://qiskit.org/learn/>

# Expressivity



Expressibility and entangling capability of parameterized quantum circuits for hybrid quantum-classical algorithms

# Expressivity

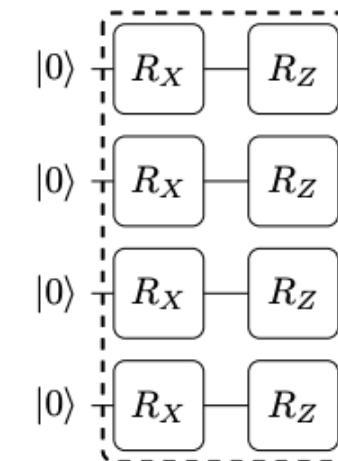


Expressibility and entangling capability of parameterized quantum circuits for hybrid quantum-classical algorithms

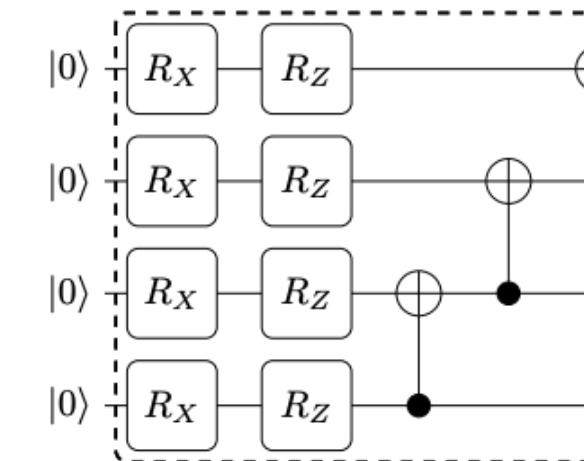
# Entanglement Capability

- The **Meyer-Wallach** measure tells how entangled a given state is. range [0, 1]
  - Unentangled state is 0
  - Full entangled state is 1
- We use **averaged MW** to measure the entanglement capability of circuit

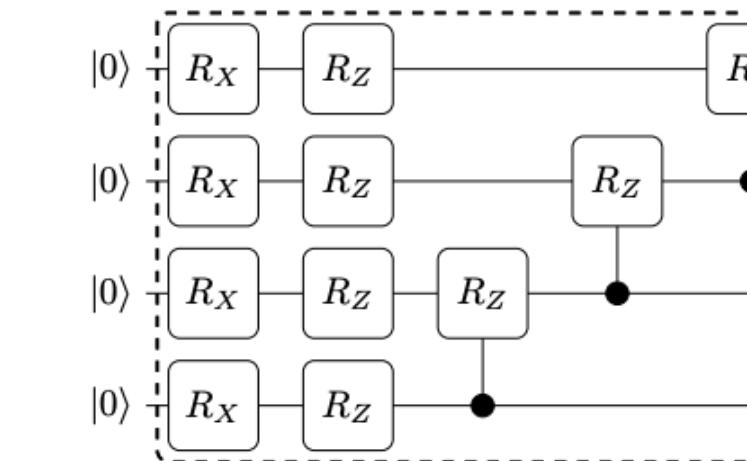
# Entanglement Capability



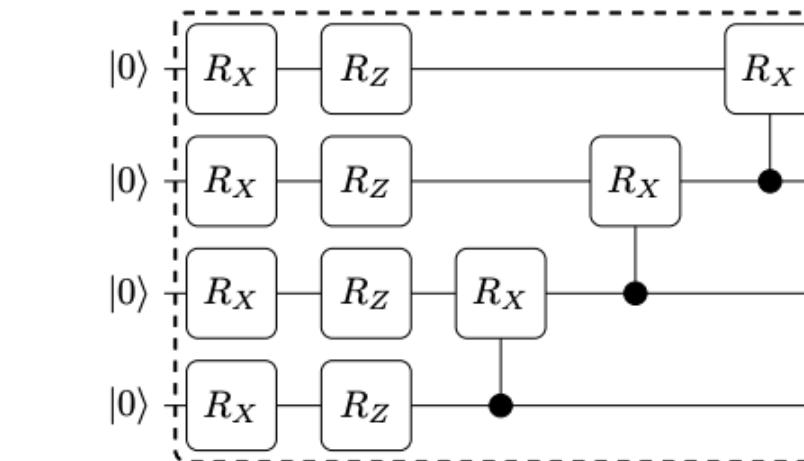
Circuit 1



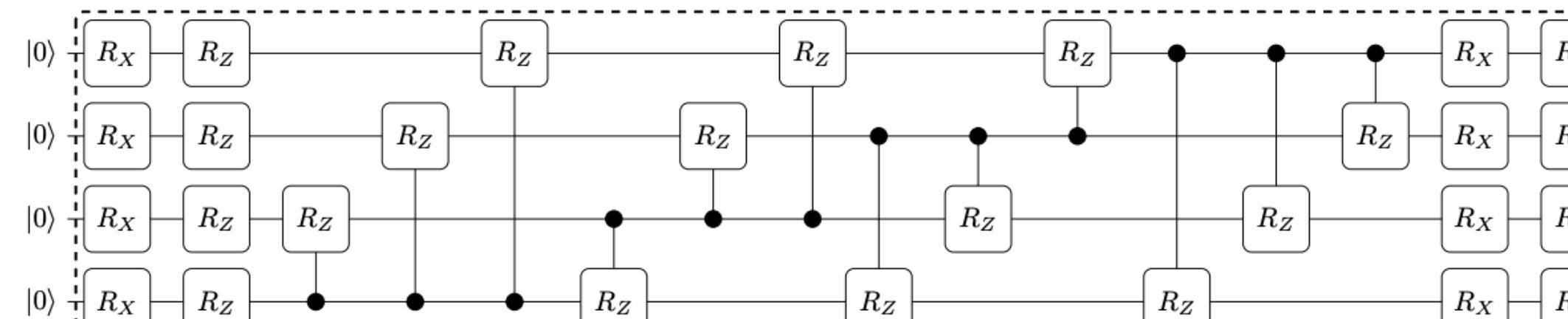
Circuit 2



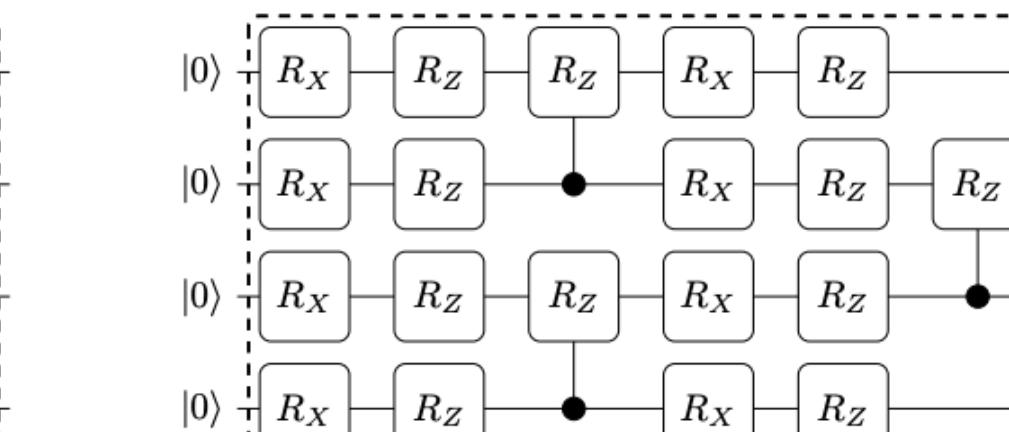
Circuit 3



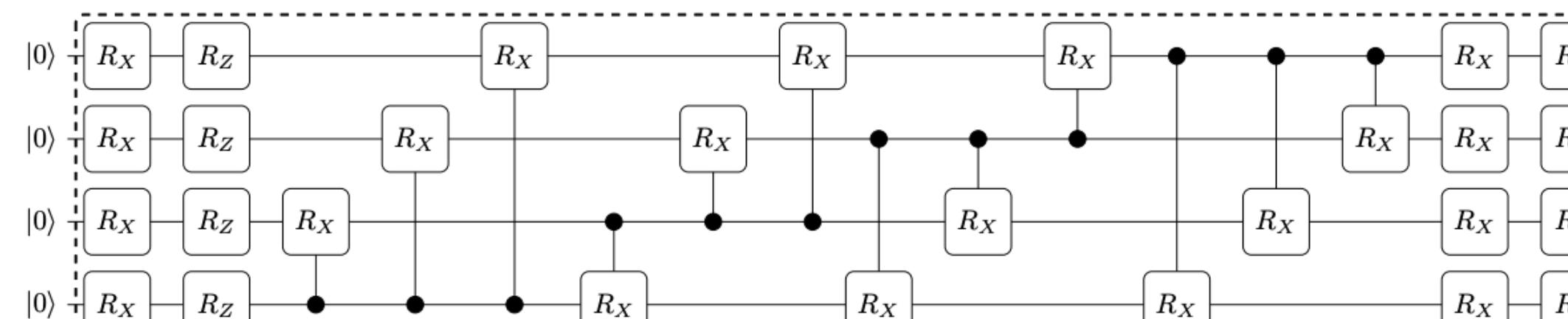
Circuit 4



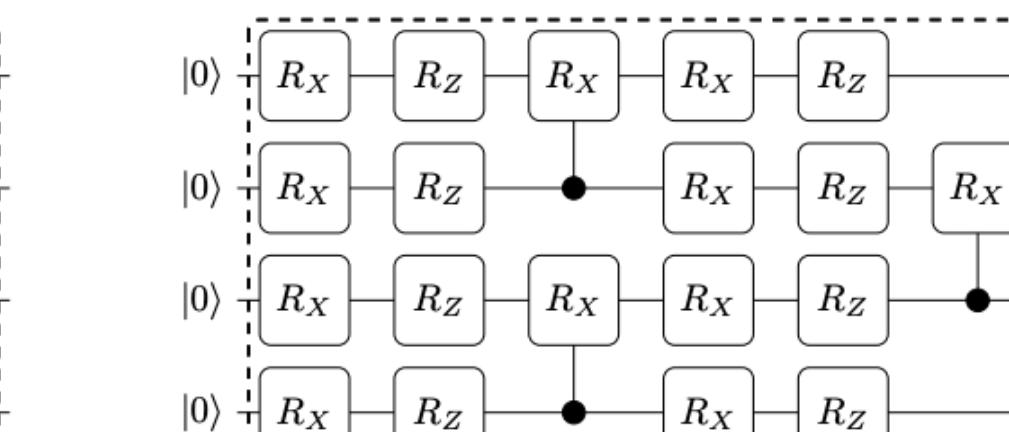
Circuit 5



Circuit 7



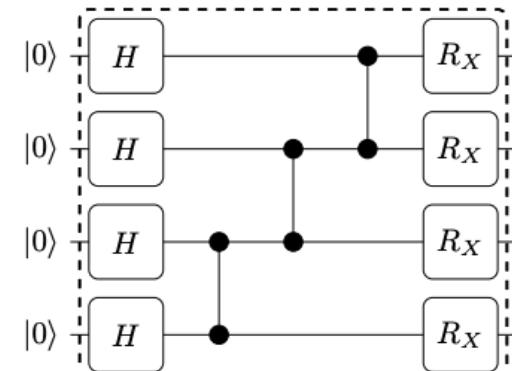
Circuit 6



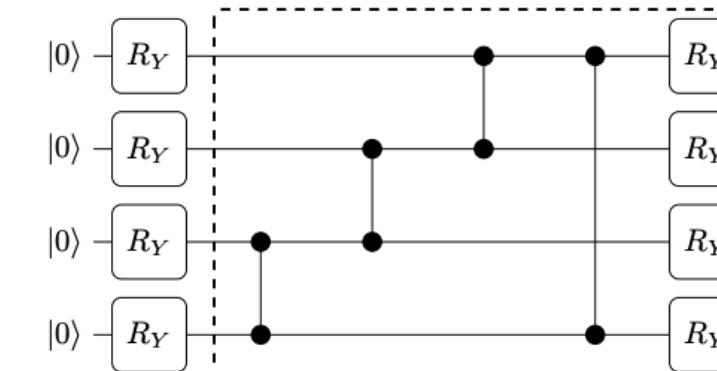
Circuit 8

Expressibility and entangling capability of parameterized quantum circuits for hybrid quantum-classical algorithms

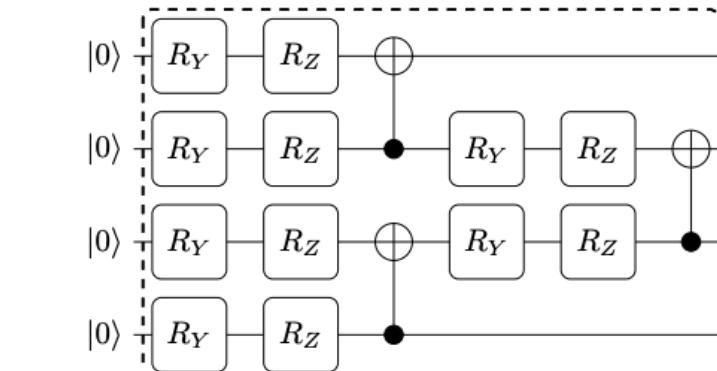
# Entanglement Capability



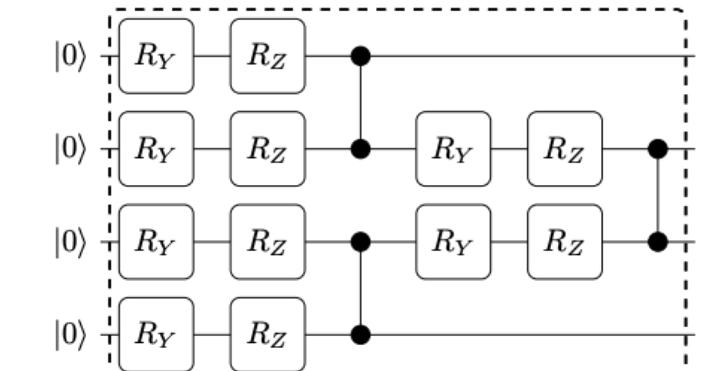
## Circuit 9



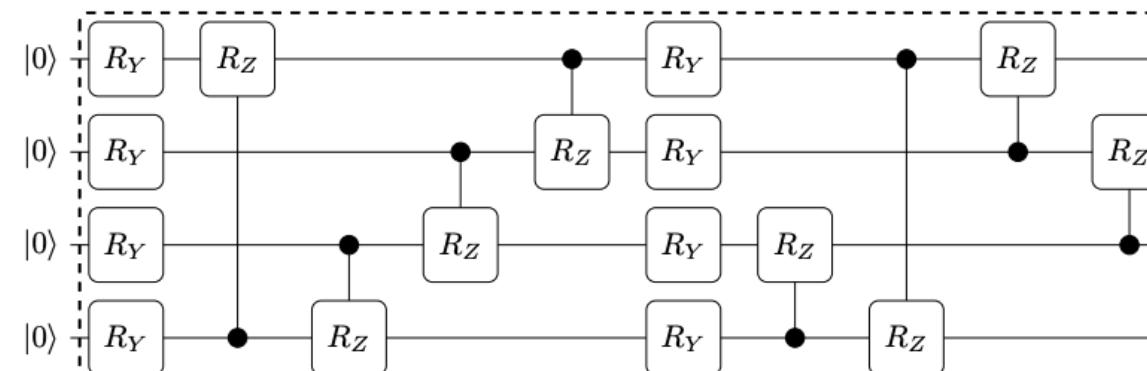
Circuit 10



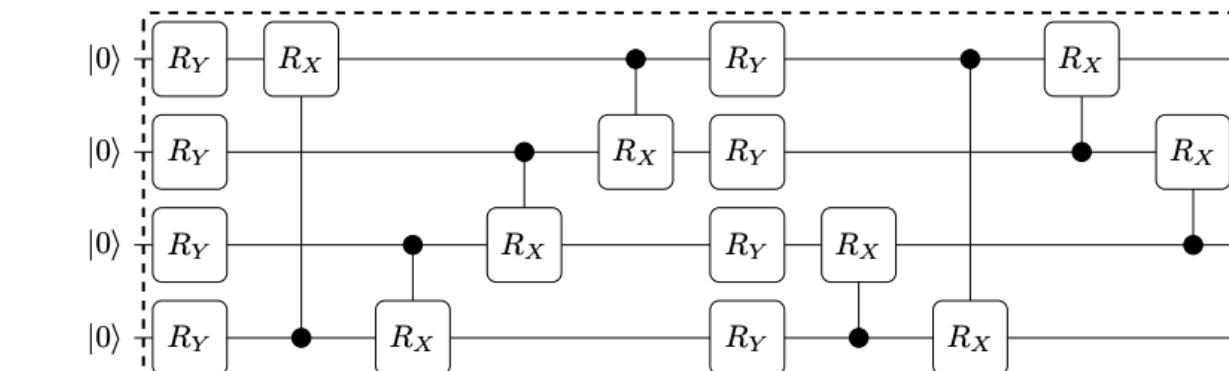
## Circuit 11



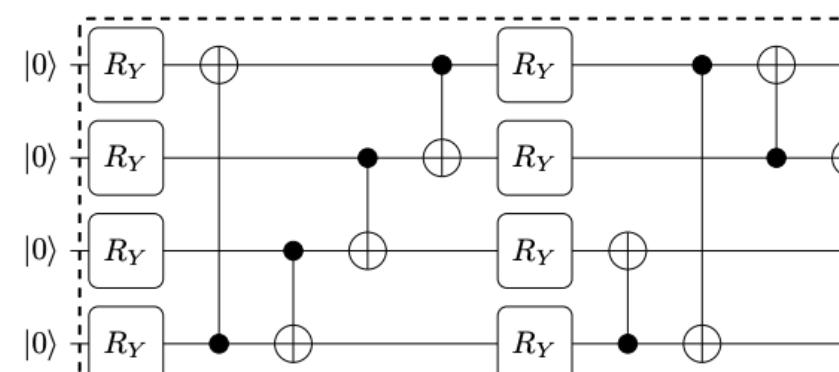
## Circuit 12



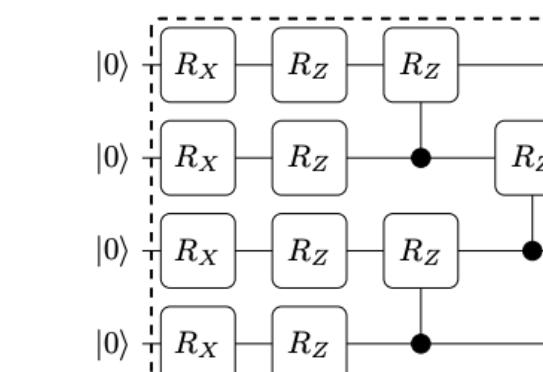
### Circuit 13



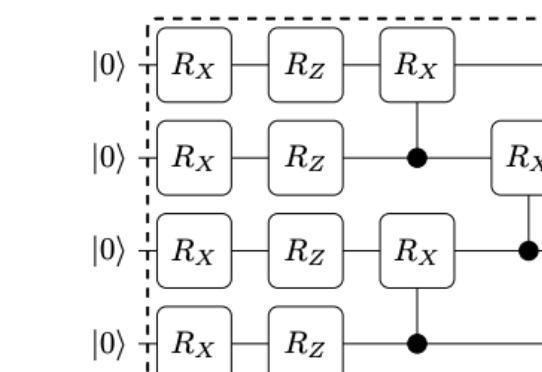
## Circuit 14



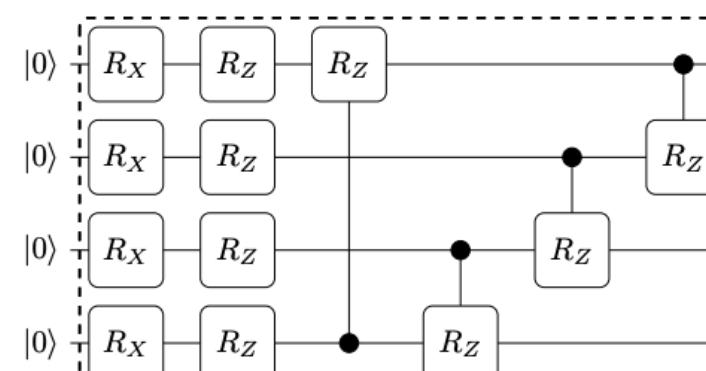
## Circuit 15



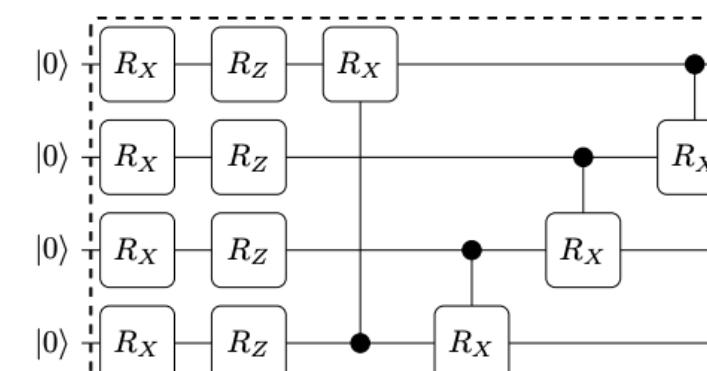
## Circuit 16



## Circuit 17



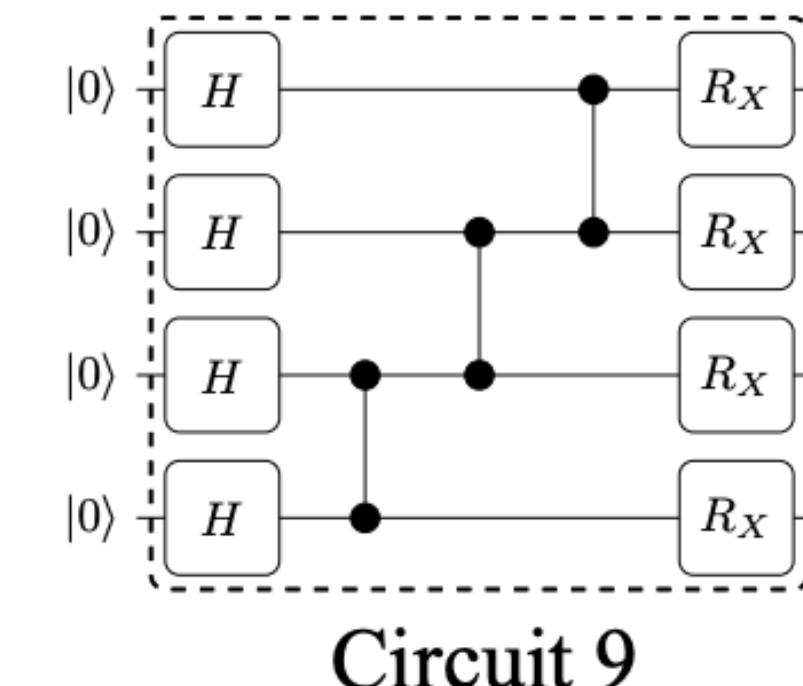
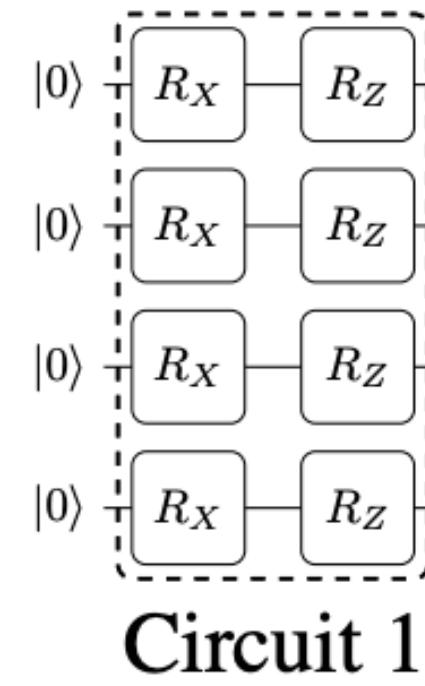
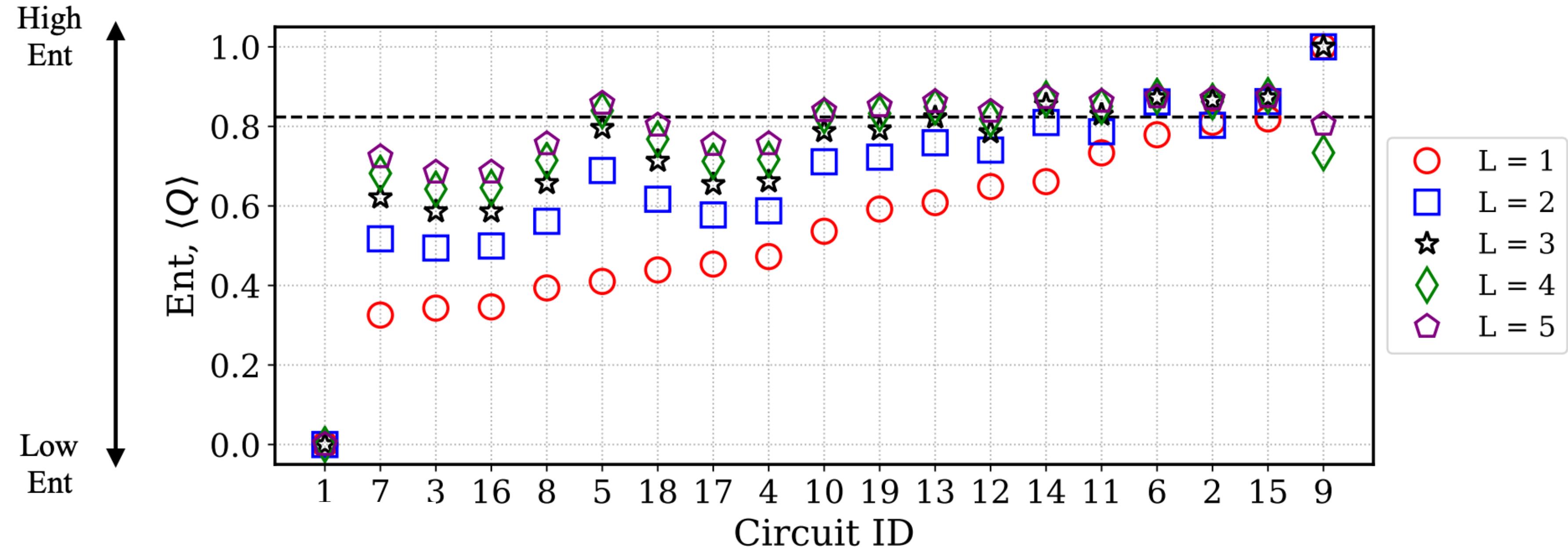
## Circuit 18



## Circuit 19

# Expressibility and entangling capability of parameterized quantum circuits for hybrid quantum-classical algorithms

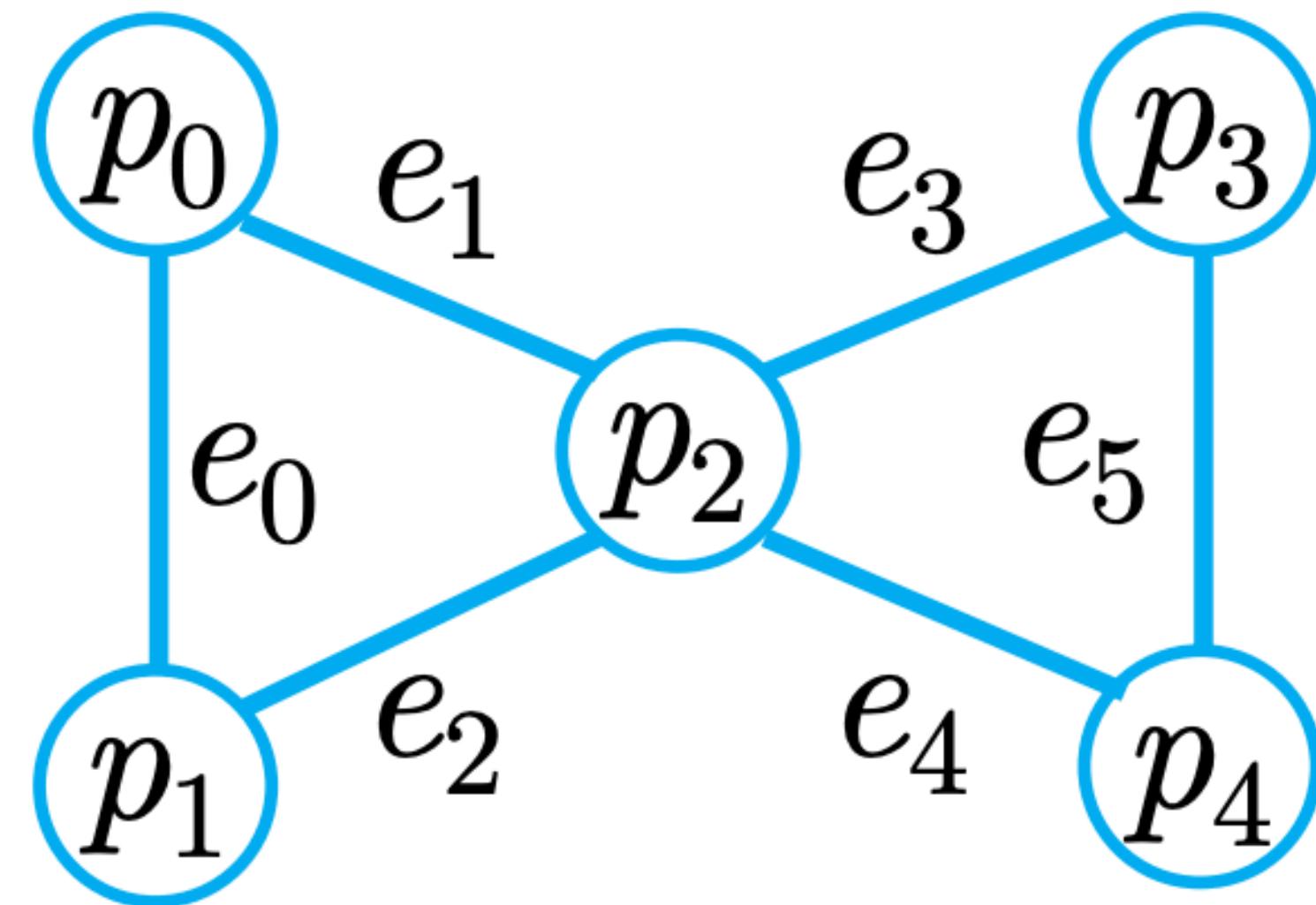
# Entanglement Capability



Expressibility and entangling capability of parameterized quantum circuits for hybrid quantum-classical algorithms

# Hardware Efficiency

- Whether the PQC design considers the qubit **connectivity**?
- whether the gates are **native** gates?



# Data Encoding

- Consider a classical data set  $X$  consisting of  $M$  samples each with  $N$  features

$$\mathcal{X} = \{x^{(1)}, \dots, x^{(m)}, \dots, x^{(M)}\}$$

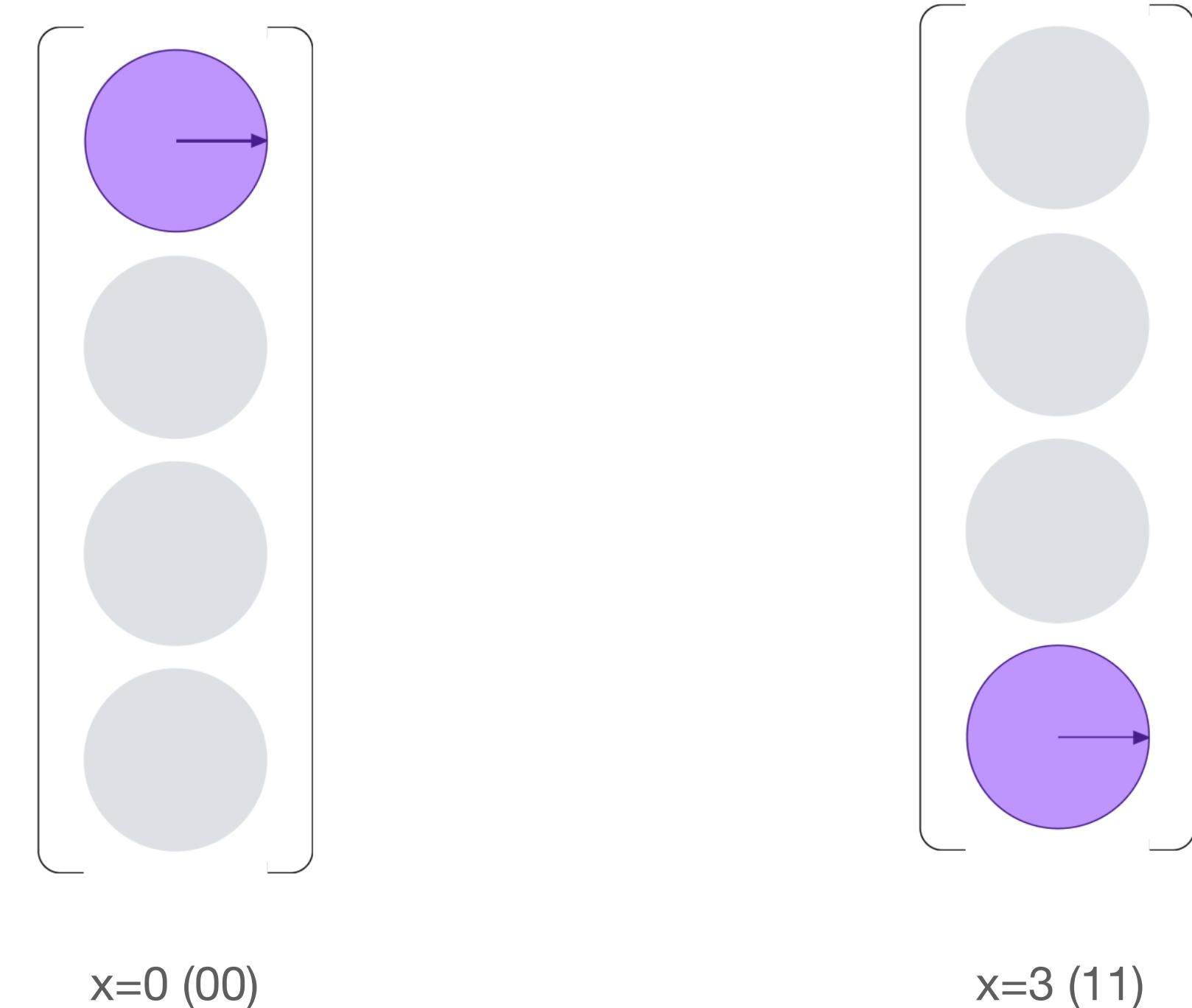
- **Encoding** methods:
  - Basis encoding
  - Amplitude encoding
  - Angle encoding
  - Arbitrary encoding

# Basis Encoding

- Similar to the **binary** representation in the classical machine
- $X = 2 \rightarrow$  binary “10”
- We create a quantum state with all energy on  $|10\rangle$

# Basis Encoding

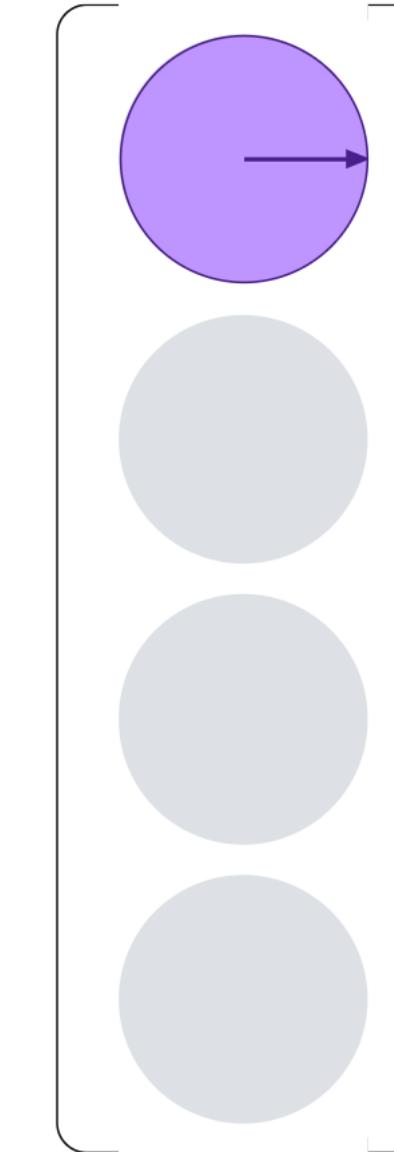
- Similar to the **binary** representation in the classical machine
- $X = 2 \rightarrow$  binary “10”
- Not efficient for one data



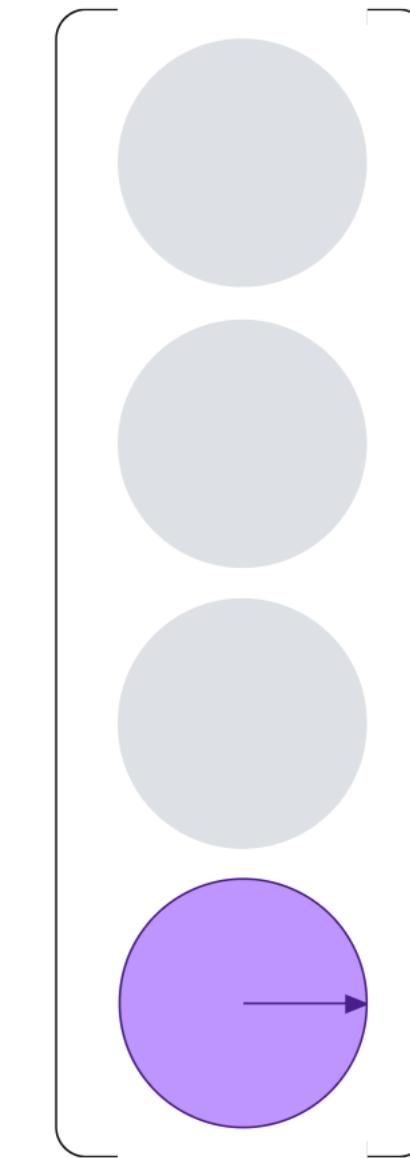
<https://qiskit.org/learn/>

# Basis Encoding

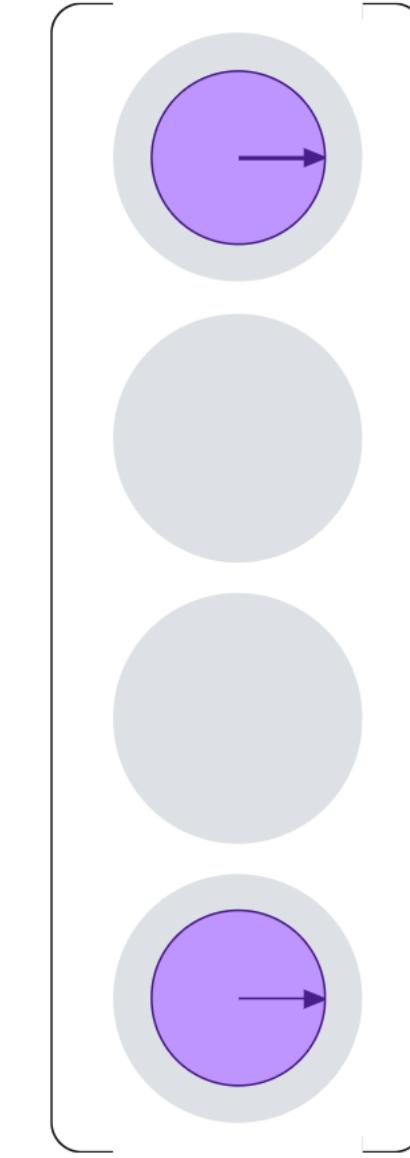
- Similar to the **binary** representation in the classical machine
- $X = 2 \rightarrow 10$
- Not efficient for one data but can be used to represent **multiple data simultaneously**



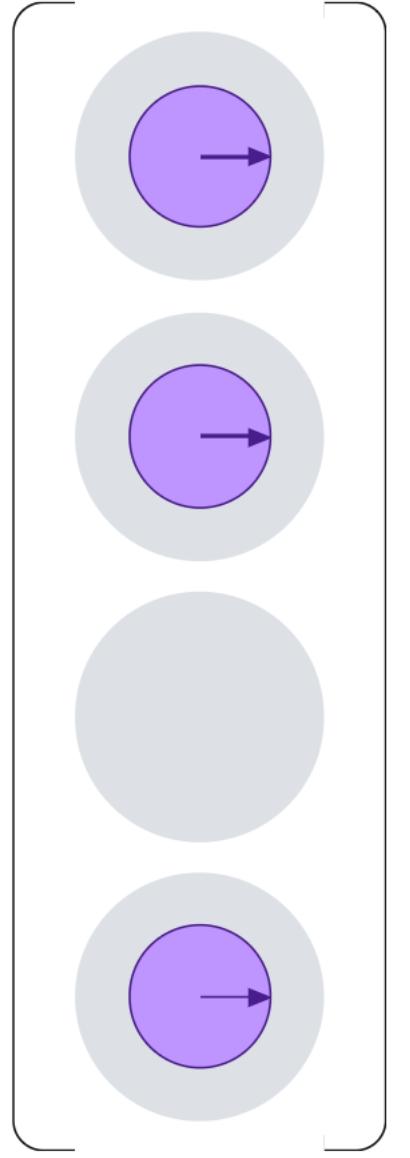
$x=0$  (00)



$x=3$  (11)

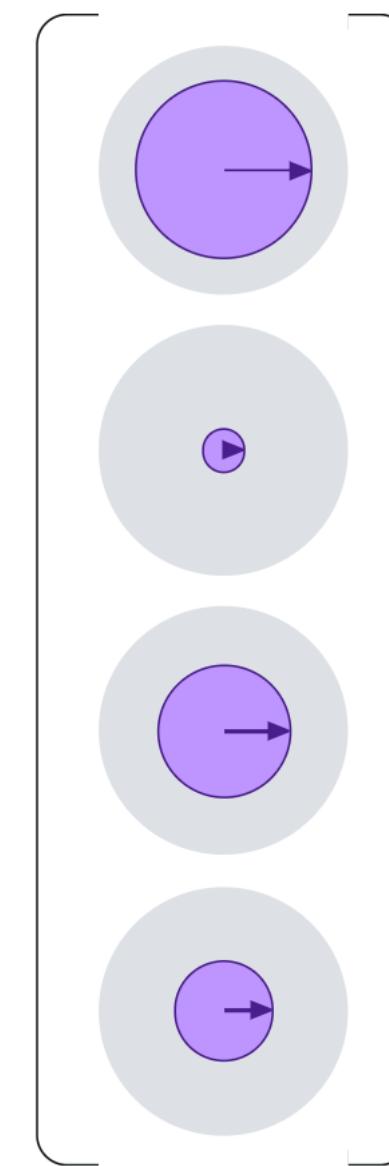
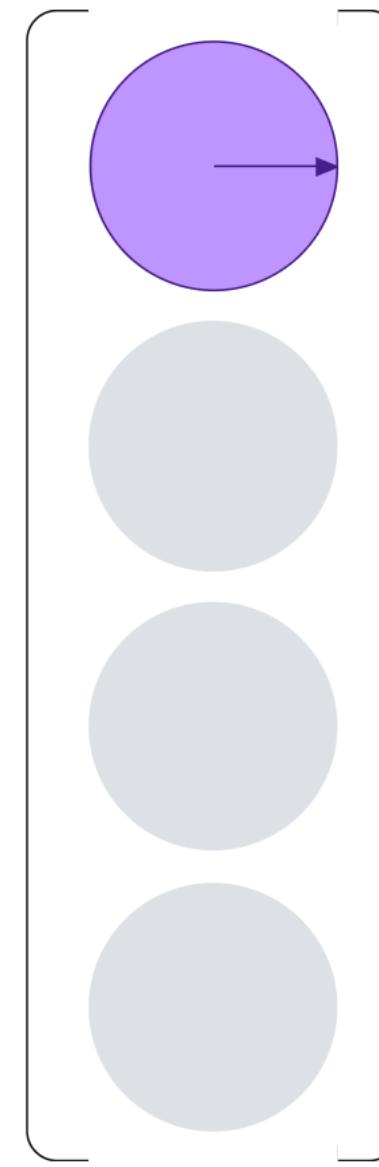


$x=0$  and  $3$  (00+11)



$x=0$  and  $1$  and  $3$  (00+ 01 +11)

# Amplitude Encoding



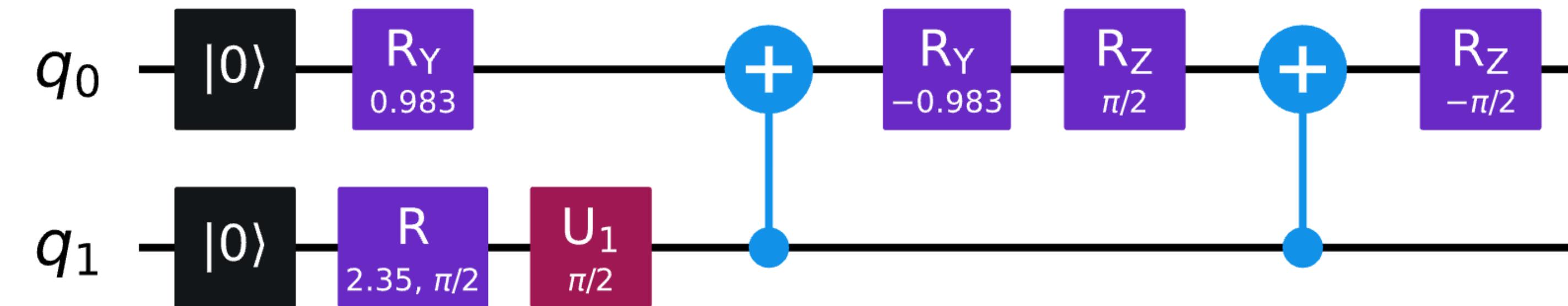
# Amplitude Encoding

- Also able to encode multiple data points together

$$\mathcal{X} = \{x^{(1)} = (1.5, 0), x^{(2)} = (-2, 3)\}$$

$$\alpha = \frac{1}{\sqrt{15.25}}(1.5, 0, -2, 3)$$

$$|\mathcal{X}\rangle = \frac{1}{\sqrt{15.25}}(1.5|00\rangle - 2|10\rangle + 3|11\rangle)$$



<https://qiskit.org/learn/>

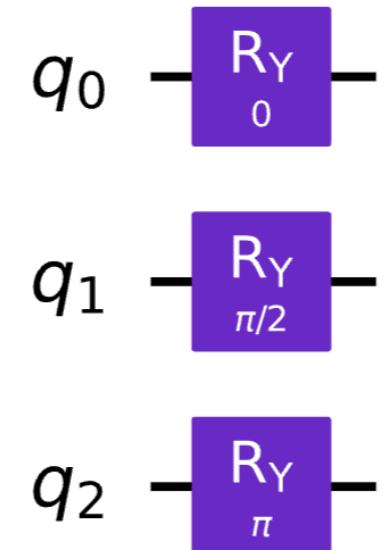
# Angle Encoding

- Encode the data in the **rotation** angles of the qubit gates
- For vector  $[0, \pi/2, \pi]$

$$|x\rangle = \bigotimes_{i=1}^N \cos(x_i)|0\rangle + \sin(x_i)|1\rangle$$

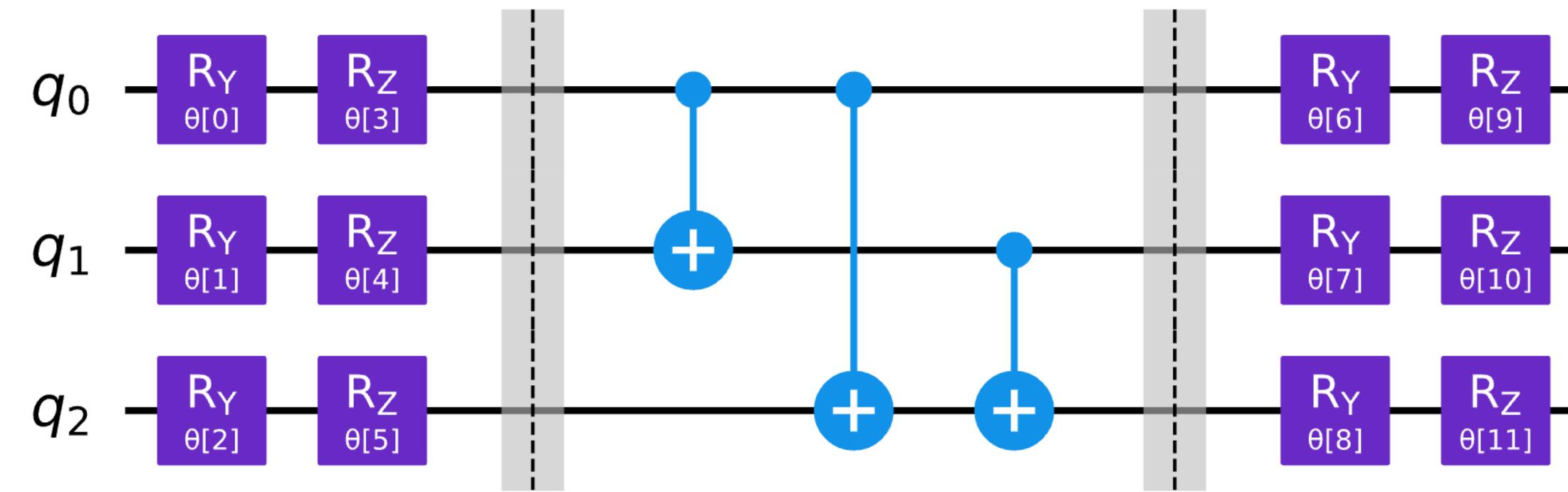
$$S_{x_j} = \bigotimes_{i=1}^N U(x_j^{(i)}) \quad U(x_j^{(i)}) = \begin{bmatrix} \cos(x_j^{(i)}) & -\sin(x_j^{(i)}) \\ \sin(x_j^{(i)}) & \cos(x_j^{(i)}) \end{bmatrix}$$

$$RY(\theta) = \exp(-i\frac{\theta}{2}Y) = \begin{pmatrix} \cos \frac{\theta}{2} & -\sin \frac{\theta}{2} \\ \sin \frac{\theta}{2} & \cos \frac{\theta}{2} \end{pmatrix}$$

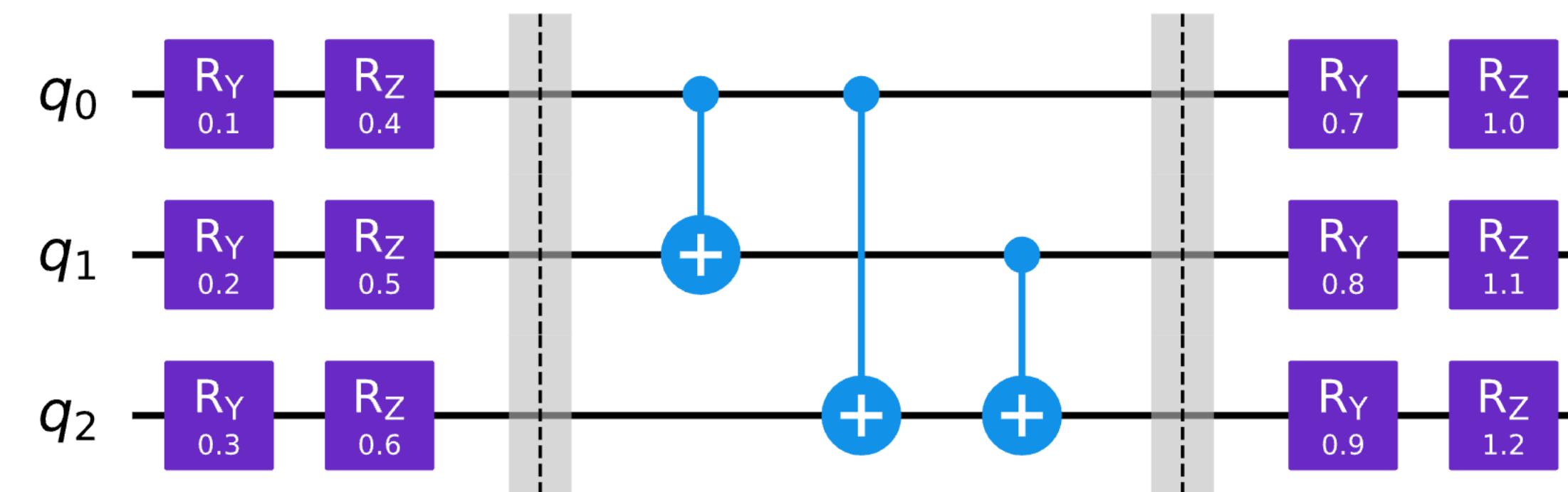


# Arbitrary Encoding

- Design **arbitrary** parameterized quantum circuit and let input data as the rotation angles



$$x = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2]$$

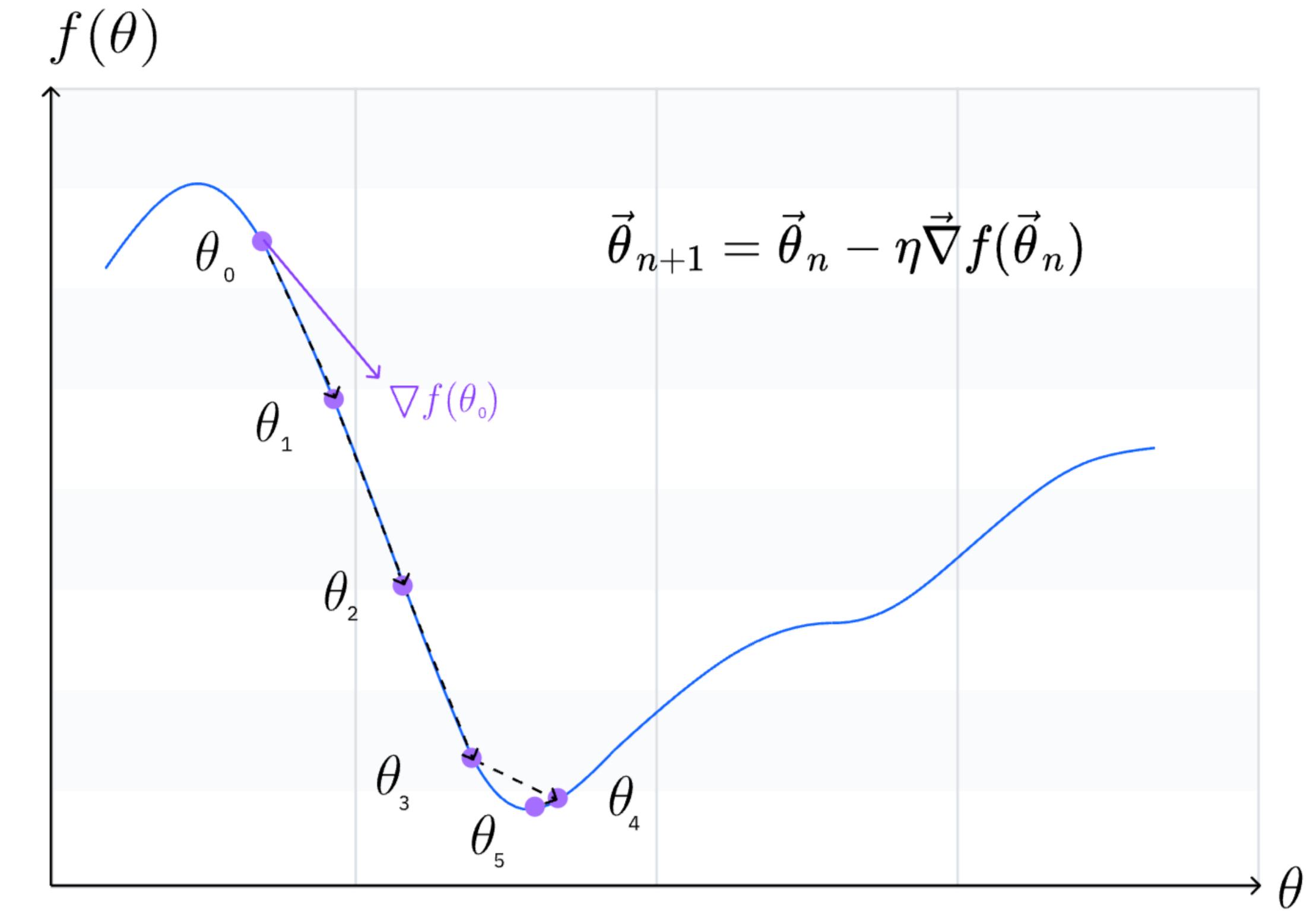
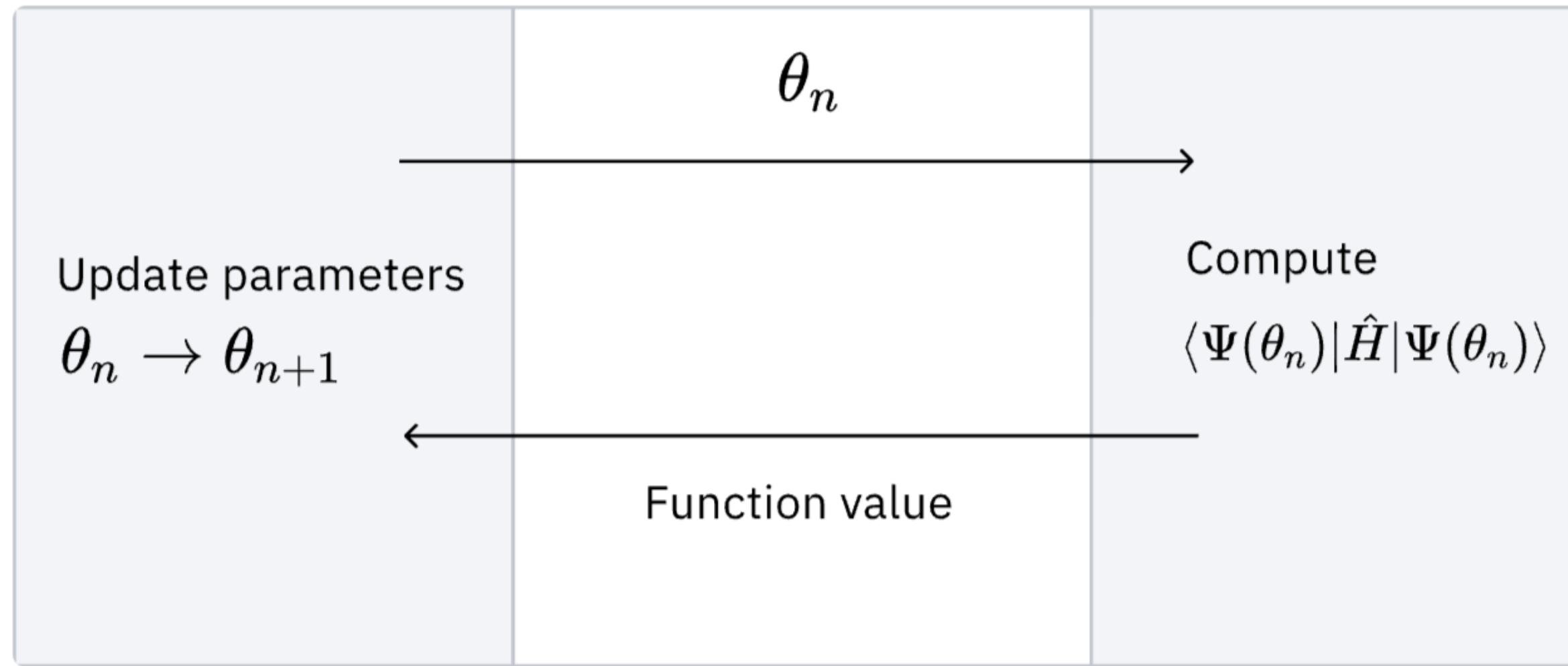


# Section 3

## PQC Training

# PQC Training

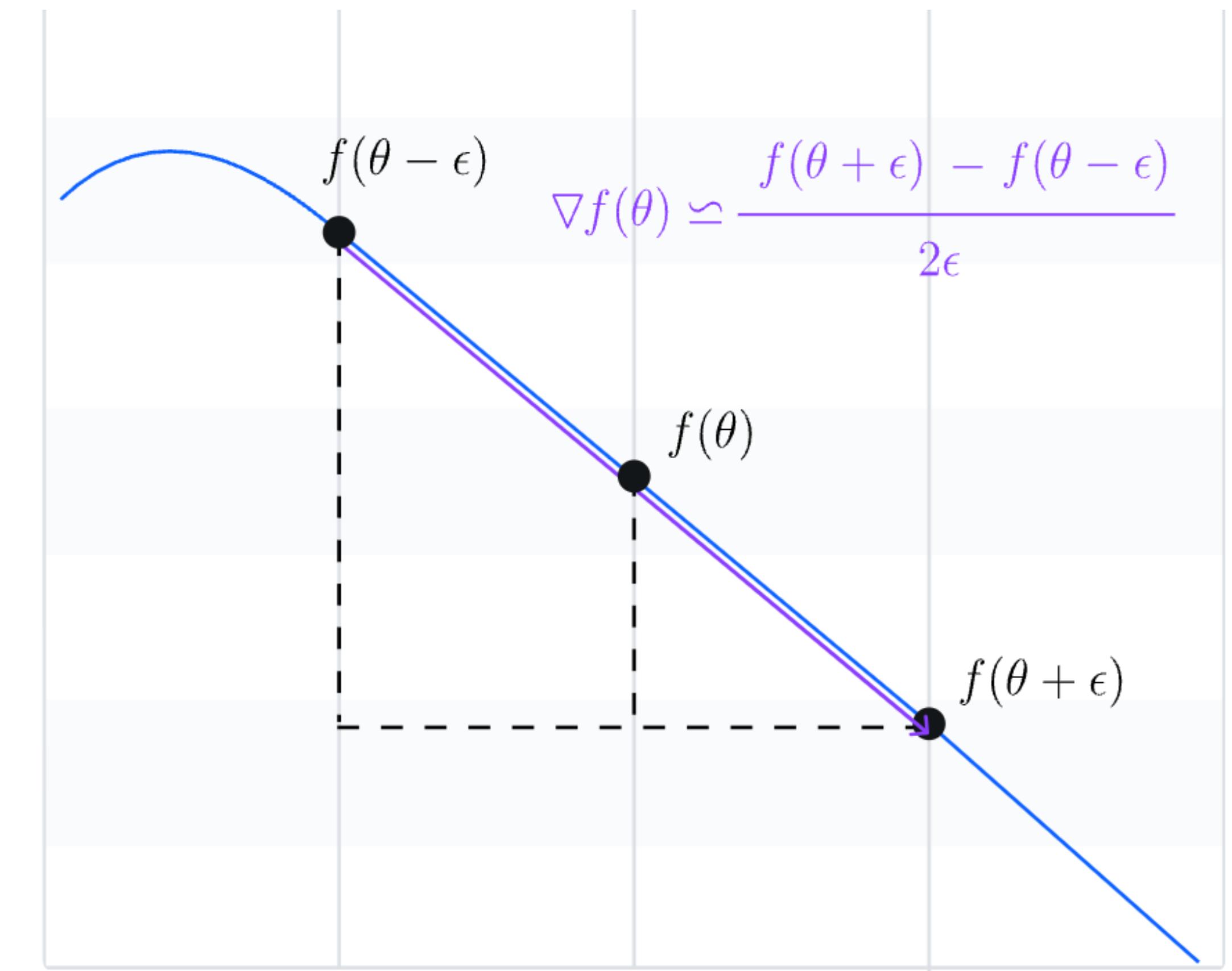
- Train parameters in parameterized quantum circuit to perform **data-driven** tasks



# Finite Difference Gradients

- Works **independently** of function's structure

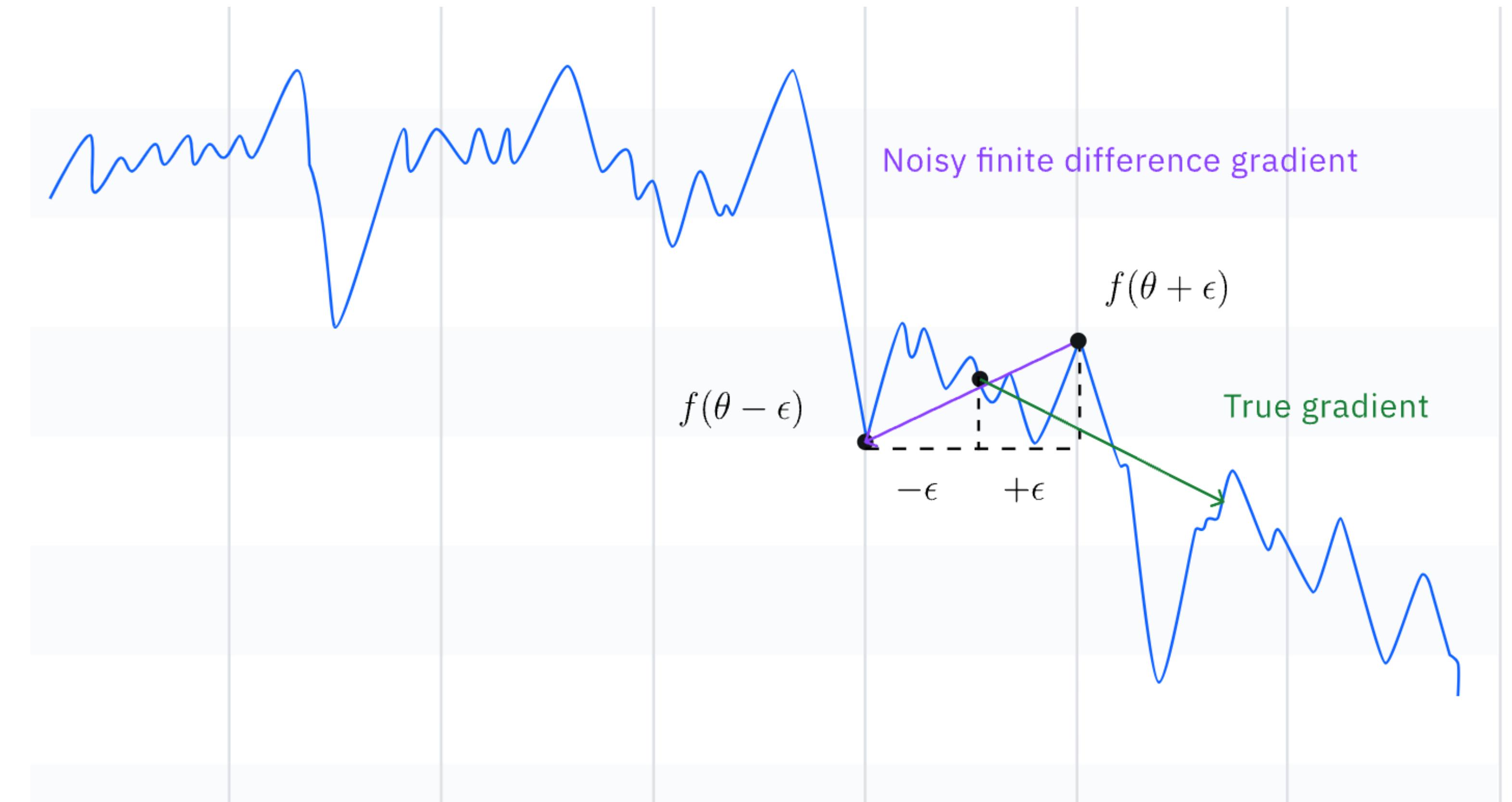
$$\vec{\nabla} f(\vec{\theta}) \approx \frac{1}{2\epsilon} \left( f(\vec{\theta} + \epsilon) - f(\vec{\theta} - \epsilon) \right)$$



<https://qiskit.org/learn/>

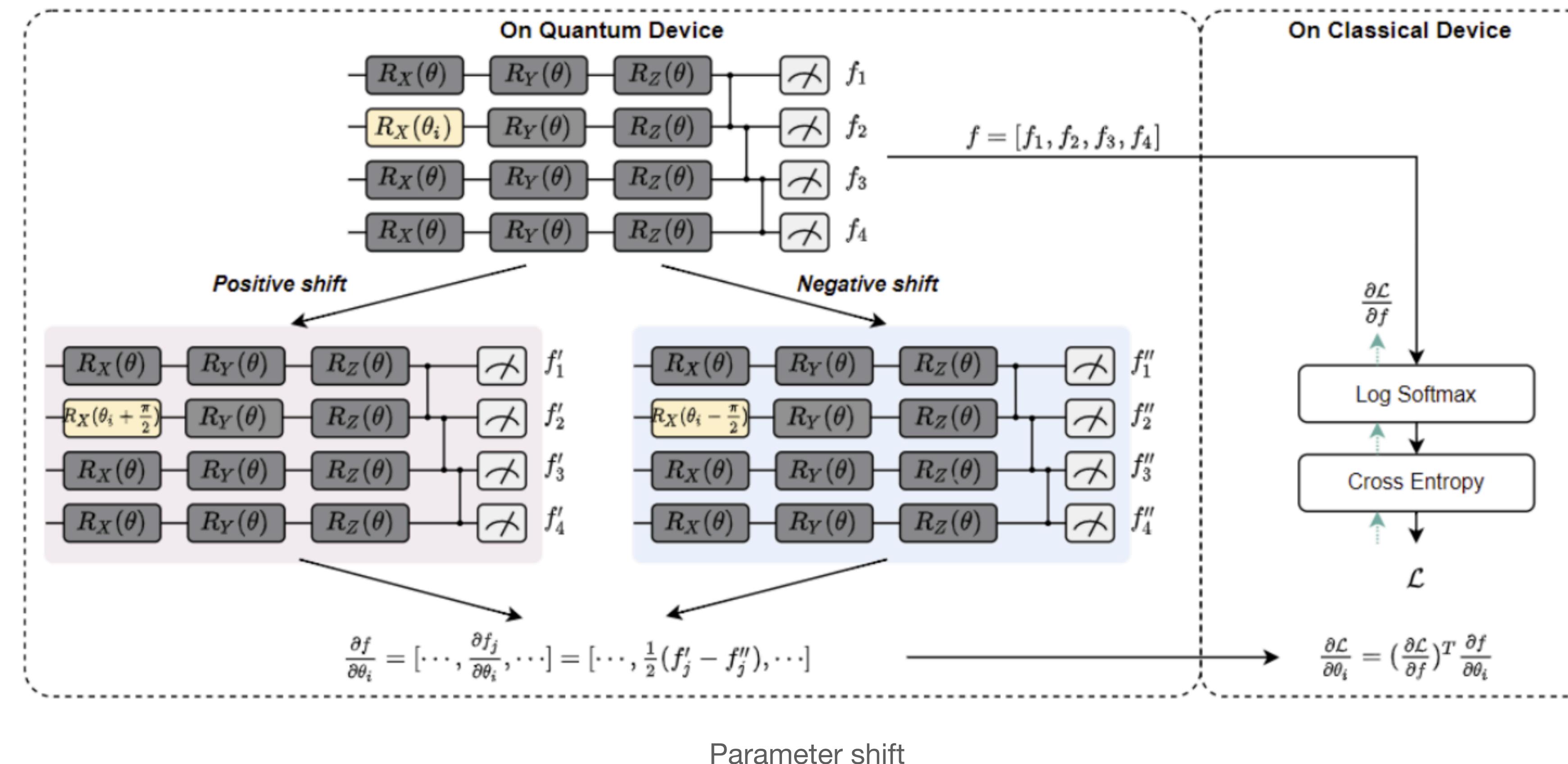
# Finite Difference Gradients

- The accuracy depends on the epsilon



<https://qiskit.org/learn/>

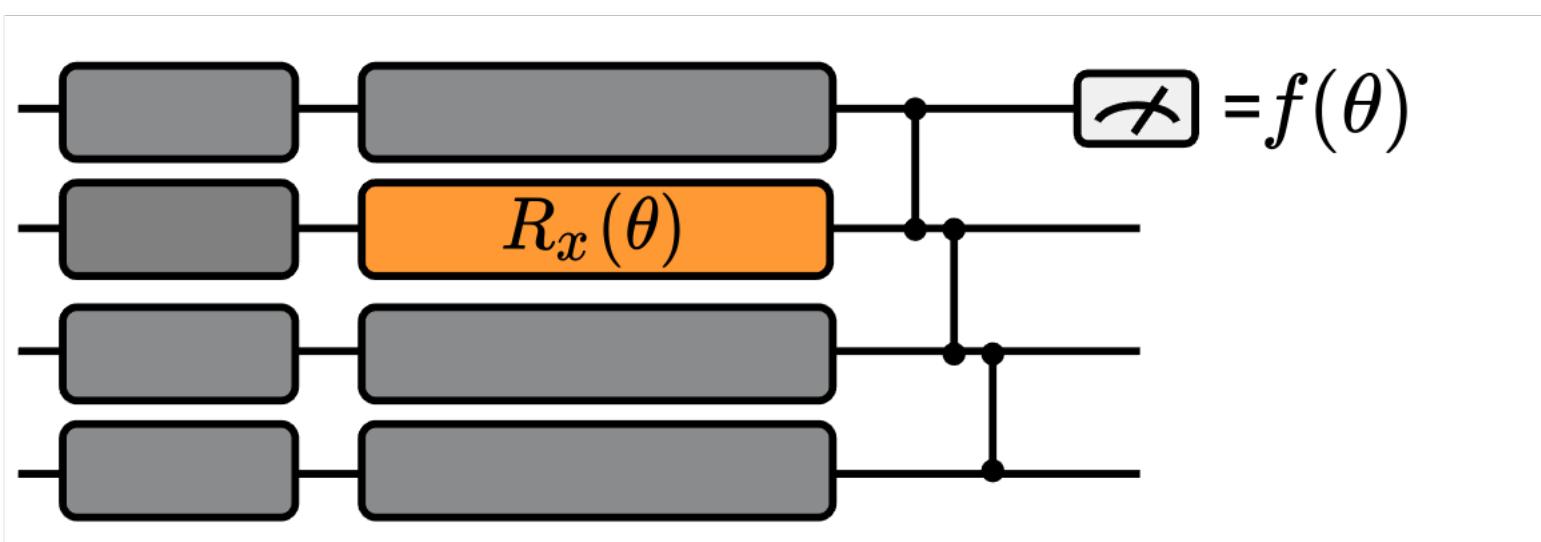
# Parameter-Shift Gradients



<https://qiskit.org/learn/>

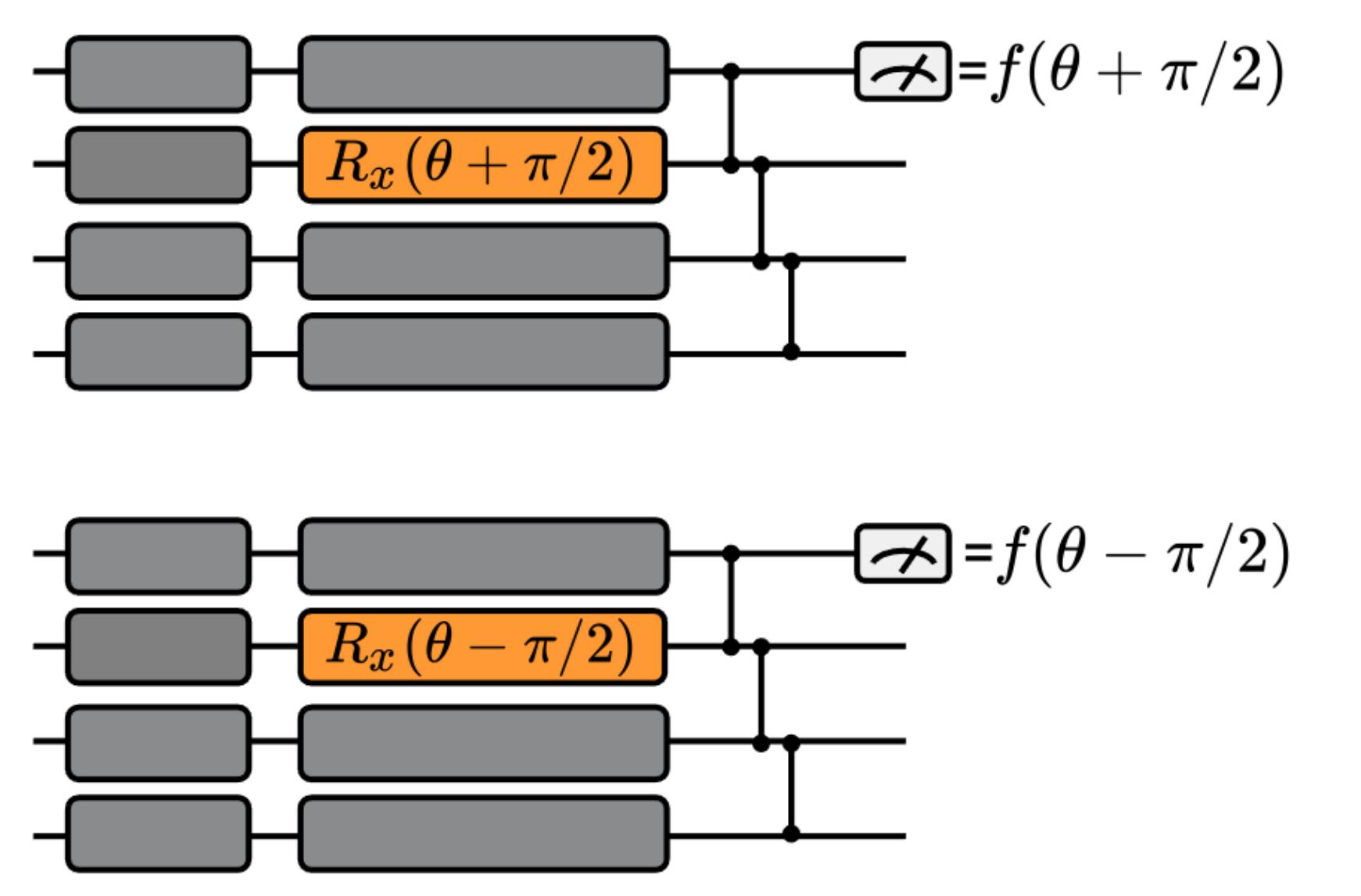
# Parameter-Shift

- Calculate the gradient of theta w.r.t  $f(\theta)$



# Parameter-Shift

- Shift  $\theta$  twice



$$\frac{\partial}{\partial \theta} f(\theta) = \frac{1}{2} \left( f\left(\theta + \frac{\pi}{2}\right) - f\left(\theta - \frac{\pi}{2}\right) \right)$$

# Proof of the parameter shift rule

- Convert to the exponential representation of gates and prove

Assume  $U(\theta_i) = R_X(\theta_i)$ ,  $R_X(\alpha) = e^{-\frac{i}{2}\alpha X}$ , where  $X$  is the Pauli-X matrix.

Firstly, the RX gate is,

$$\begin{aligned} R_X(\alpha) &= e^{-\frac{i}{2}\alpha X} = \sum_{k=0}^{\infty} (-i\alpha/2)^k X^k / k! \\ &= \sum_{k=0}^{\infty} (-i\alpha/2)^{2k} X^{2k} / (2k)! + \sum_{k=0}^{\infty} (-i\alpha/2)^{2k+1} X^{2k+1} / (2k+1)! \\ &= \sum_{k=0}^{\infty} (-1)^k (\alpha/2)^{2k} I / (2k)! - i \sum_{k=0}^{\infty} (-1)^k (\alpha/2)^{2k+1} X / (2k+1)! \\ &= \cos(\alpha/2)I - i \sin(\alpha/2)X. \end{aligned}$$

# Proof of the parameter shift rule

Let  $\alpha = \frac{\pi}{2}$ ,  $R_X(\pm\frac{\pi}{2}) = \frac{1}{\sqrt{2}}(I \mp iX)$ .

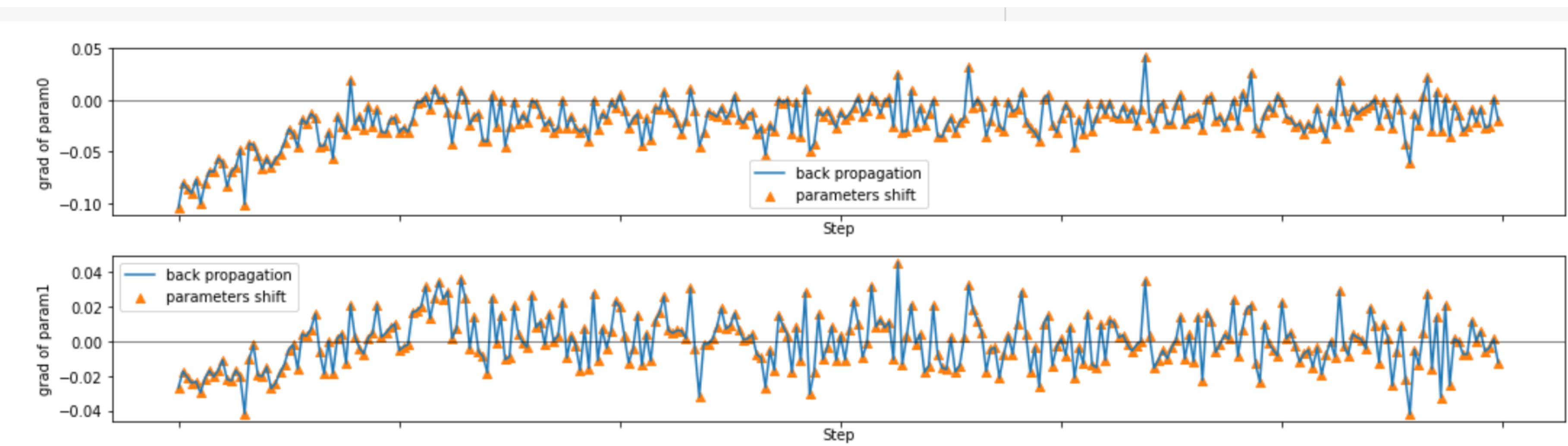
As  $f(\theta) = \langle \psi | R_X(\theta_i)^\dagger \widehat{Q} R_X(\theta_i) | \psi \rangle$ ,  $R_X(\alpha)R_X(\beta) = R_X(\alpha + \beta)$ , and  $\frac{\partial}{\partial \alpha} R_X(\alpha) = -\frac{i}{2} X R_X(\alpha)$ , we have

$$\begin{aligned}\frac{\partial f(\theta)}{\partial \theta_i} &= \langle \psi | R_X(\theta_i)^\dagger \left(-\frac{i}{2} X\right)^\dagger \widehat{Q} R_X(\theta_i) | \psi \rangle + \langle \psi | R_X(\theta_i)^\dagger \widehat{Q} \left(-\frac{i}{2} X\right) R_X(\theta_i) | \psi \rangle \\ &= \frac{1}{4} (\langle \psi | R_X(\theta_i)^\dagger (I - iX)^\dagger \widehat{Q} (I - iX) R_X(\theta_i) | \psi \rangle \\ &\quad - \langle \psi | R_X(\theta_i)^\dagger (I + iX)^\dagger \widehat{Q} (I + iX) R_X(\theta_i) | \psi \rangle) \\ &= \frac{1}{2} (\langle \psi | R_X(\theta_i)^\dagger R_X(\frac{\pi}{2})^\dagger \widehat{Q} R_X(\frac{\pi}{2}) R_X(\theta_i) | \psi \rangle \\ &\quad - \langle \psi | R_X(\theta_i)^\dagger R_X(-\frac{\pi}{2})^\dagger \widehat{Q} R_X(-\frac{\pi}{2}) R_X(\theta_i) | \psi \rangle) \\ &= \frac{1}{2} (f(\theta_+) - f(\theta_-)).\end{aligned}$$

i --

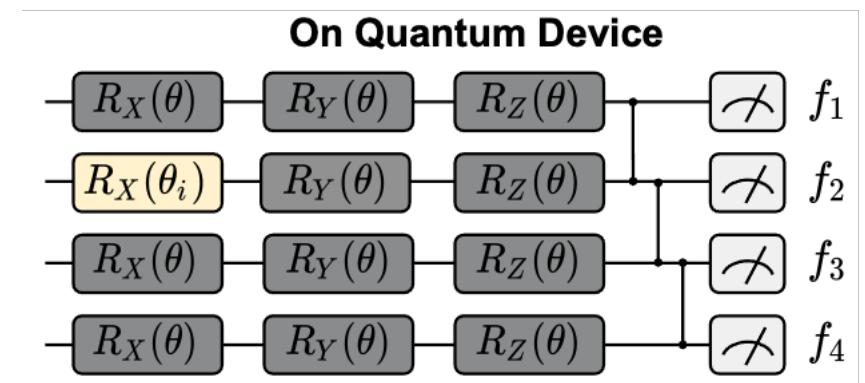
# Back-Propagation

- The gradients can also be calculated with back-propagation
- Can only be used on **classical simulator**
- All computation are essentially differentiable linear algebra



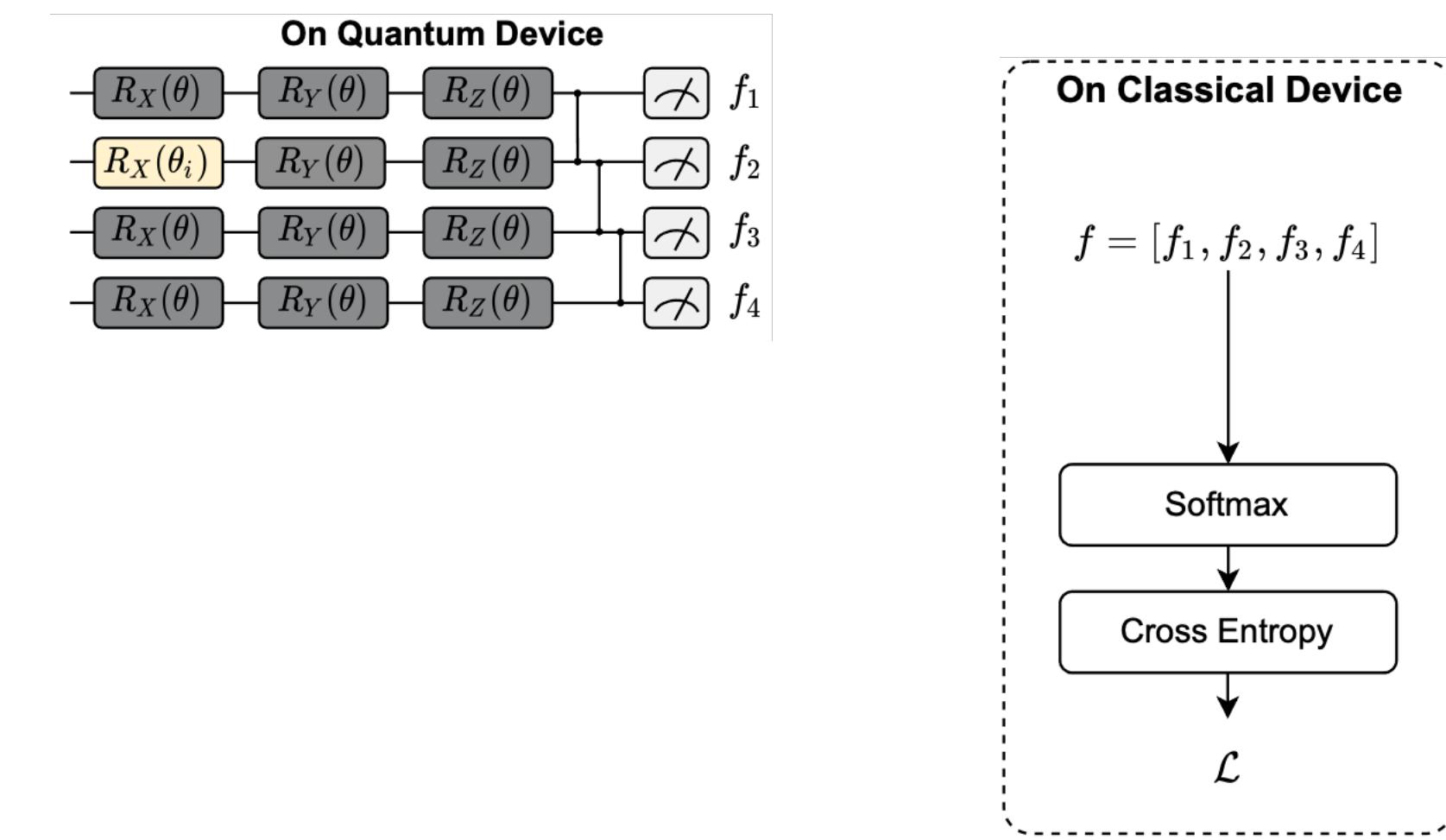
# General Flow to calculate PQC gradients

- Step 1: Run on QC without shift to obtain f



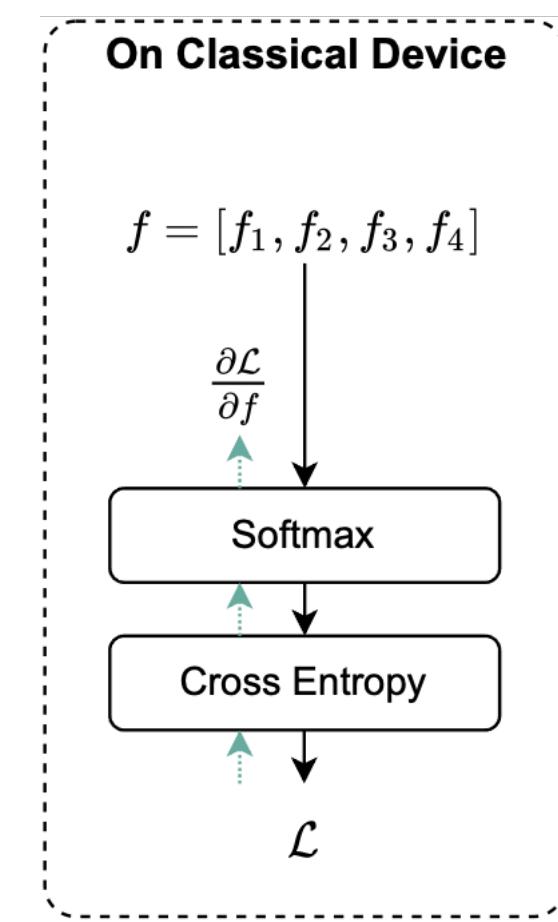
# General Flow to calculate PQC gradients

- Step 2: Further forward to get Loss



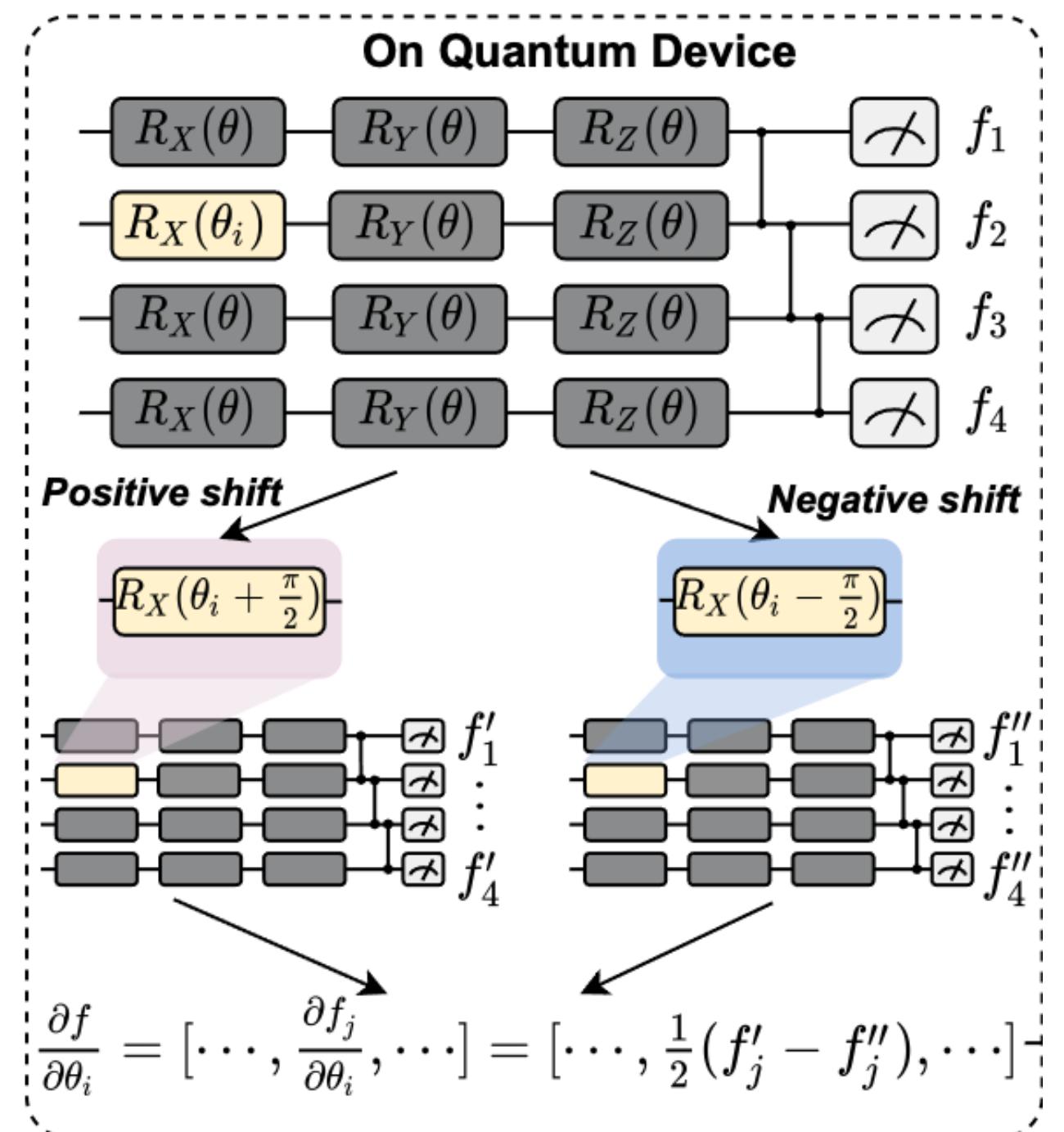
# General Flow to calculate PQC gradients

- Step 3: Backpropagation to calculate  $\partial(\text{Loss})/\partial f(\theta)$



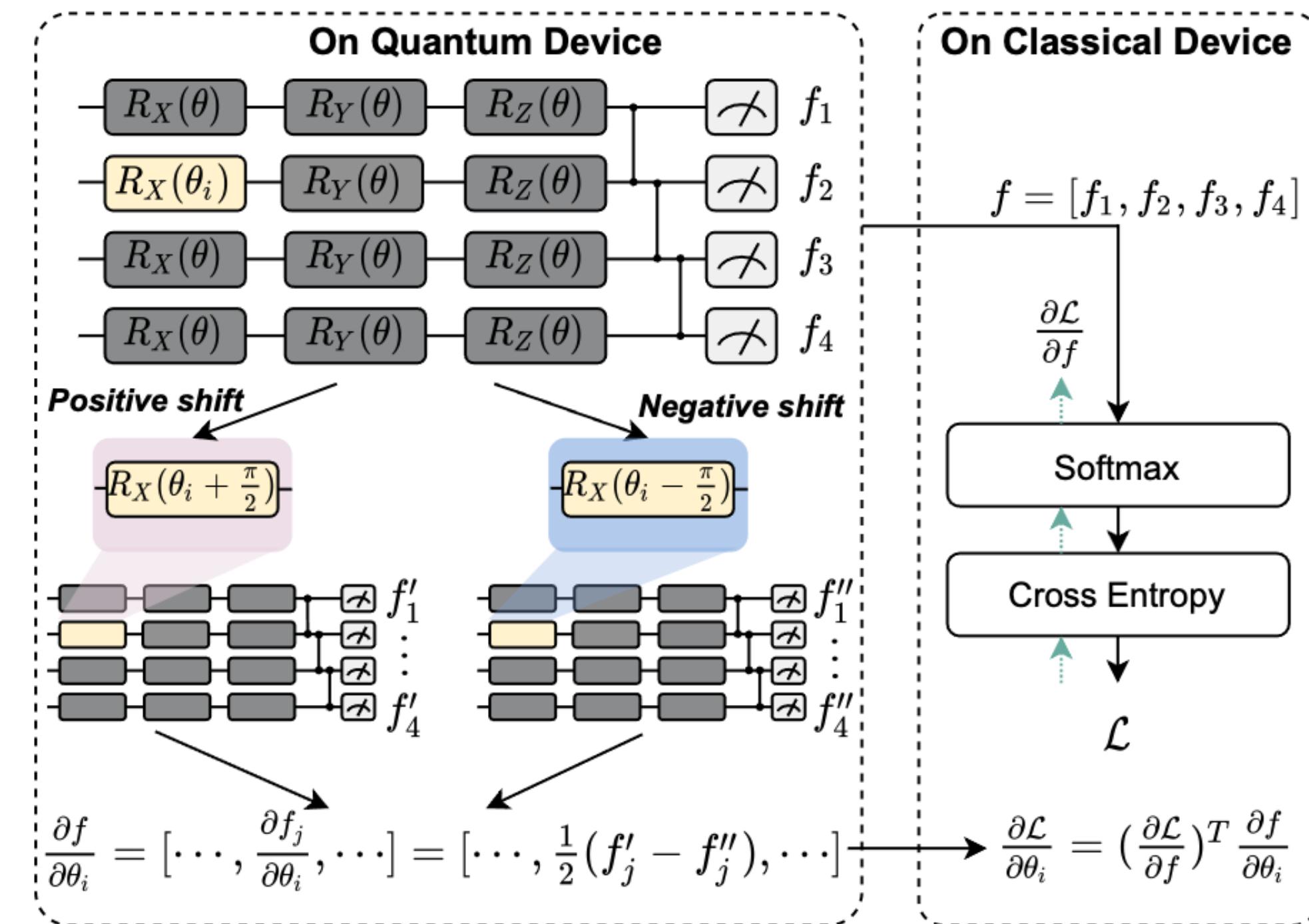
# General Flow to calculate PQC gradients

- Step 4: Shift twice and run on QC to calculate  $(\partial f(\theta)) / (\partial \theta_i)$ 
  - Or use finite differentiate

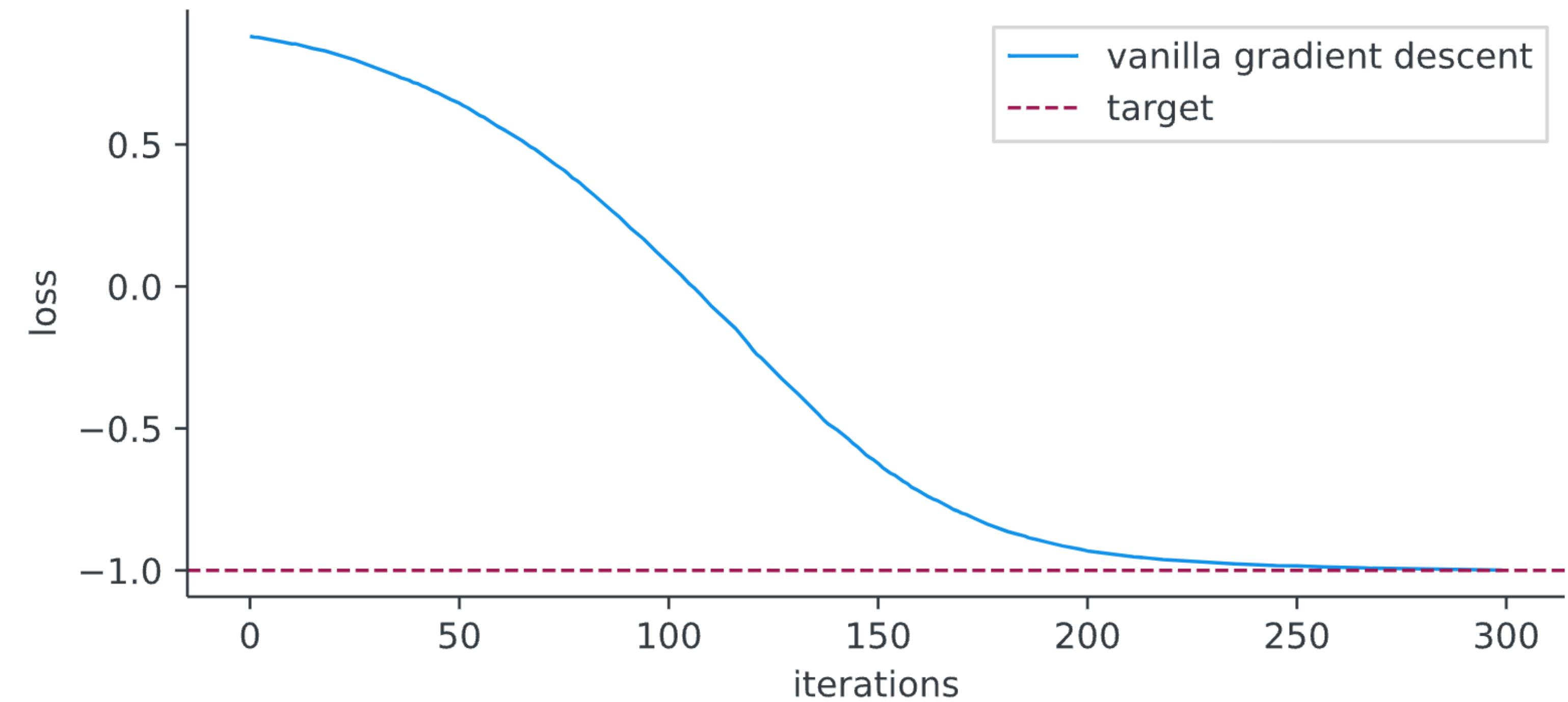


# General Flow to calculate PQC gradients

- Step 5: By Chain Rule, sum over 4 passes (4 qubits)
- Only forward on quantum device



# Training Techniques



# Training Techniques

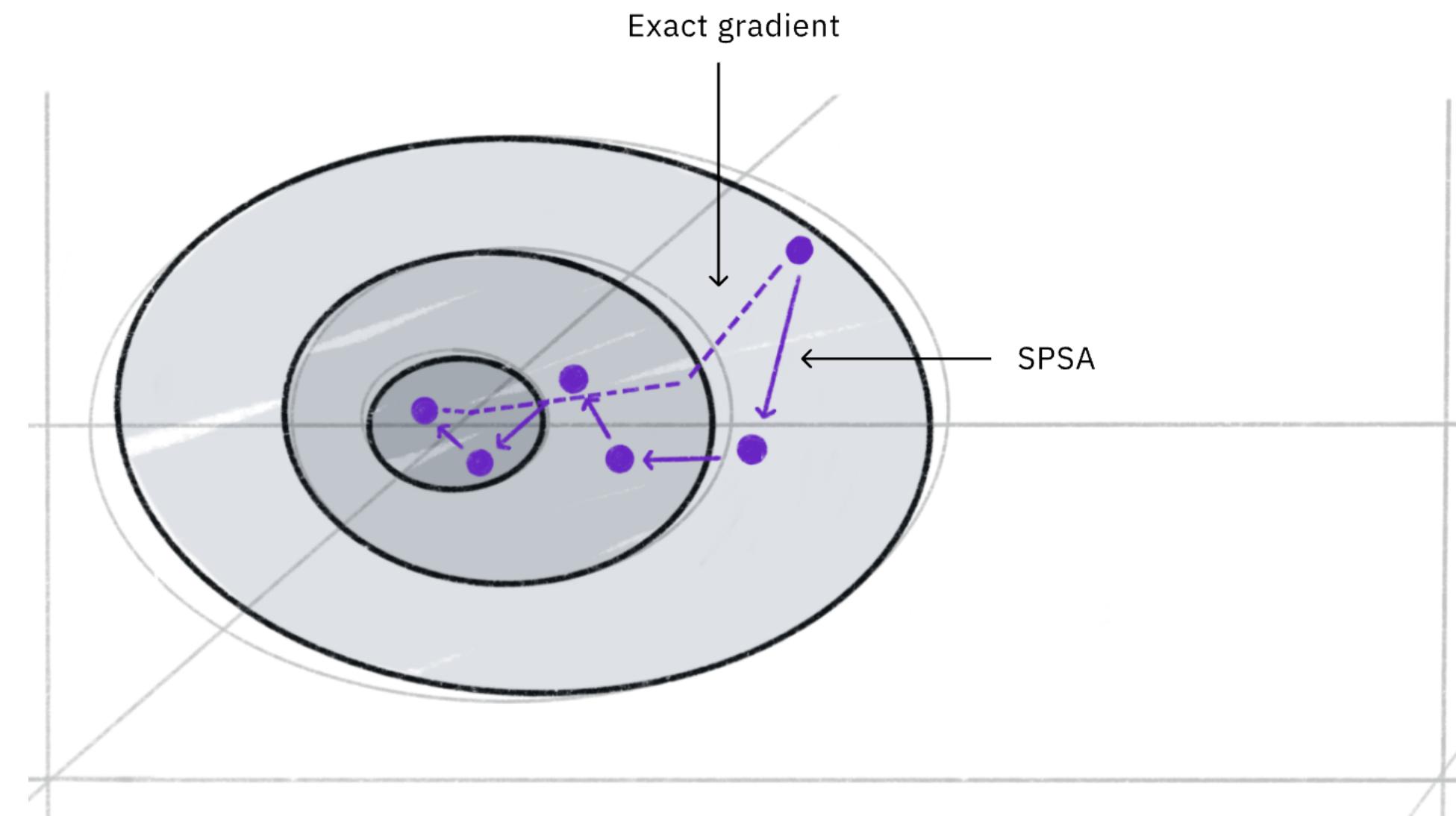
## SPSA

- Simultaneous Perturbation Stochastic Approximation
- How **many** circuit run for the original methods?  $2N$

$$\vec{\nabla} f(\vec{\theta}) = \begin{pmatrix} \frac{\partial f}{\partial \theta_1} \\ \vdots \\ \frac{\partial f}{\partial \theta_n} \end{pmatrix} \simeq \frac{1}{2\epsilon} \underbrace{\begin{pmatrix} f(\vec{\theta} + \epsilon \vec{e}_1) - f(\vec{\theta} - \epsilon \vec{e}_1) \\ \vdots \\ f(\vec{\theta} + \epsilon \vec{e}_n) - f(\vec{\theta} - \epsilon \vec{e}_n) \end{pmatrix}}_{\text{Finite difference:  
Perturb each dim once}} \simeq \frac{f(\vec{\theta} + \epsilon \vec{\Delta}) - f(\vec{\theta} - \epsilon \vec{\Delta})}{2\epsilon} \vec{\Delta}^{-1}$$

Random perturbation  $\vec{\Delta} \in \{+1, -1\}^n$

SPSA:  
simultaneously  
perturb all dims at once!

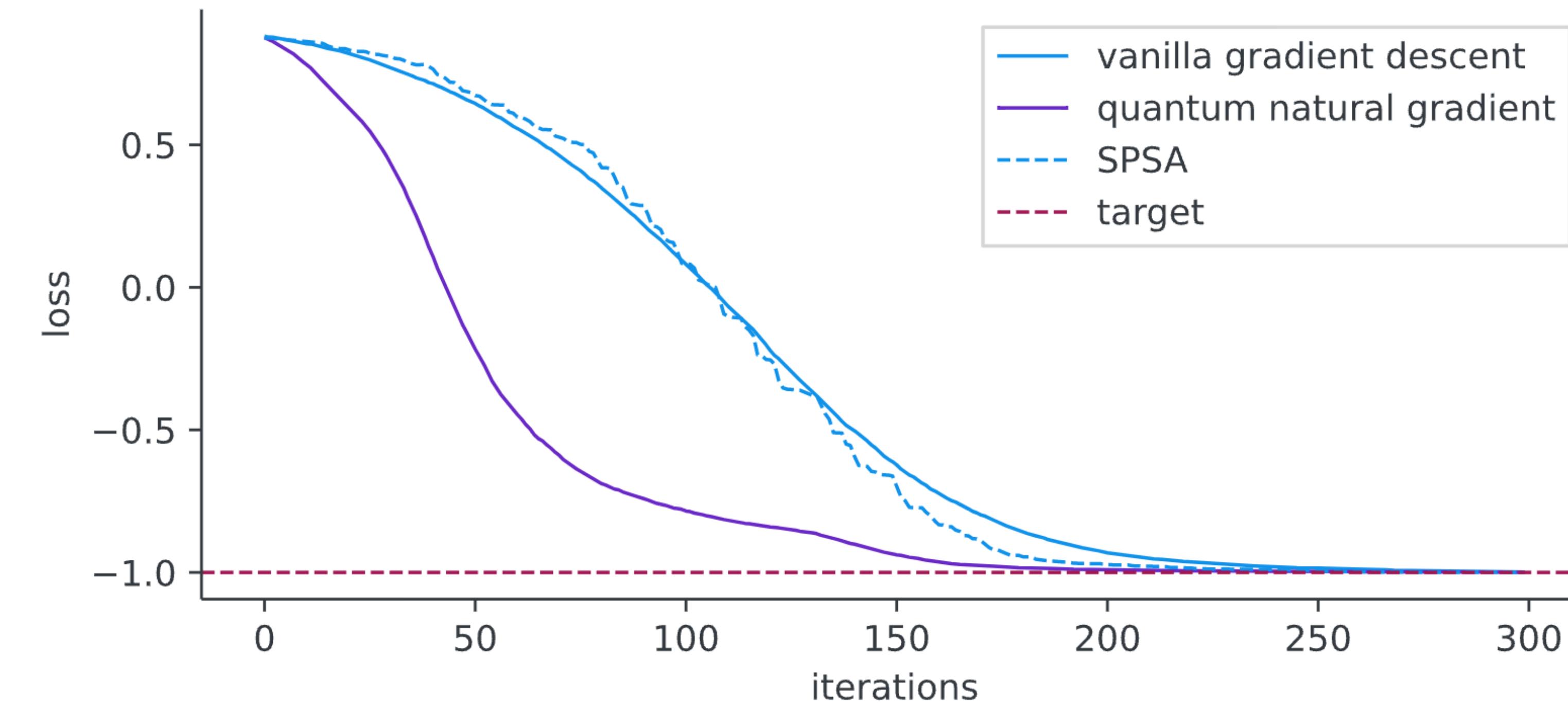


<https://qiskit.org/learn/>

# Training Techniques

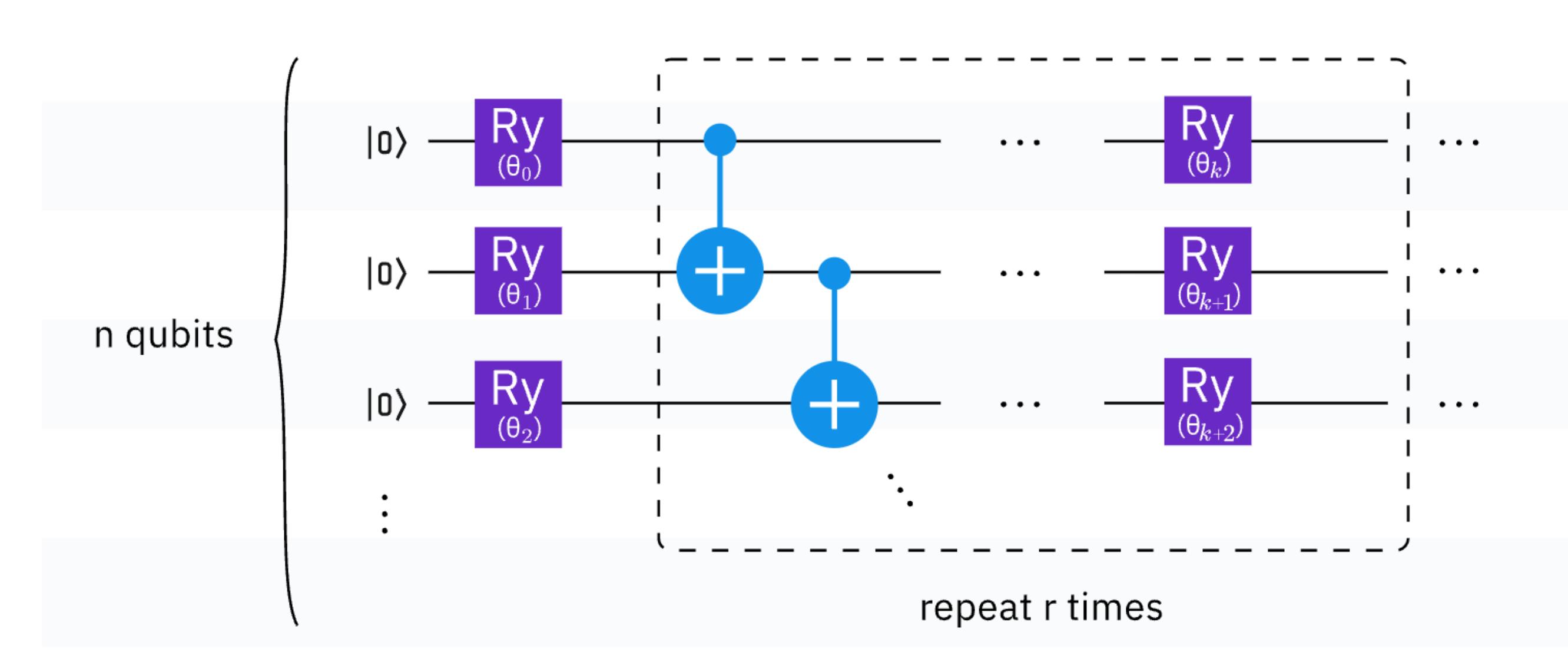
## SPSA

- The SPSA has similar convergence as the gradient descent



# Barren Plateaus

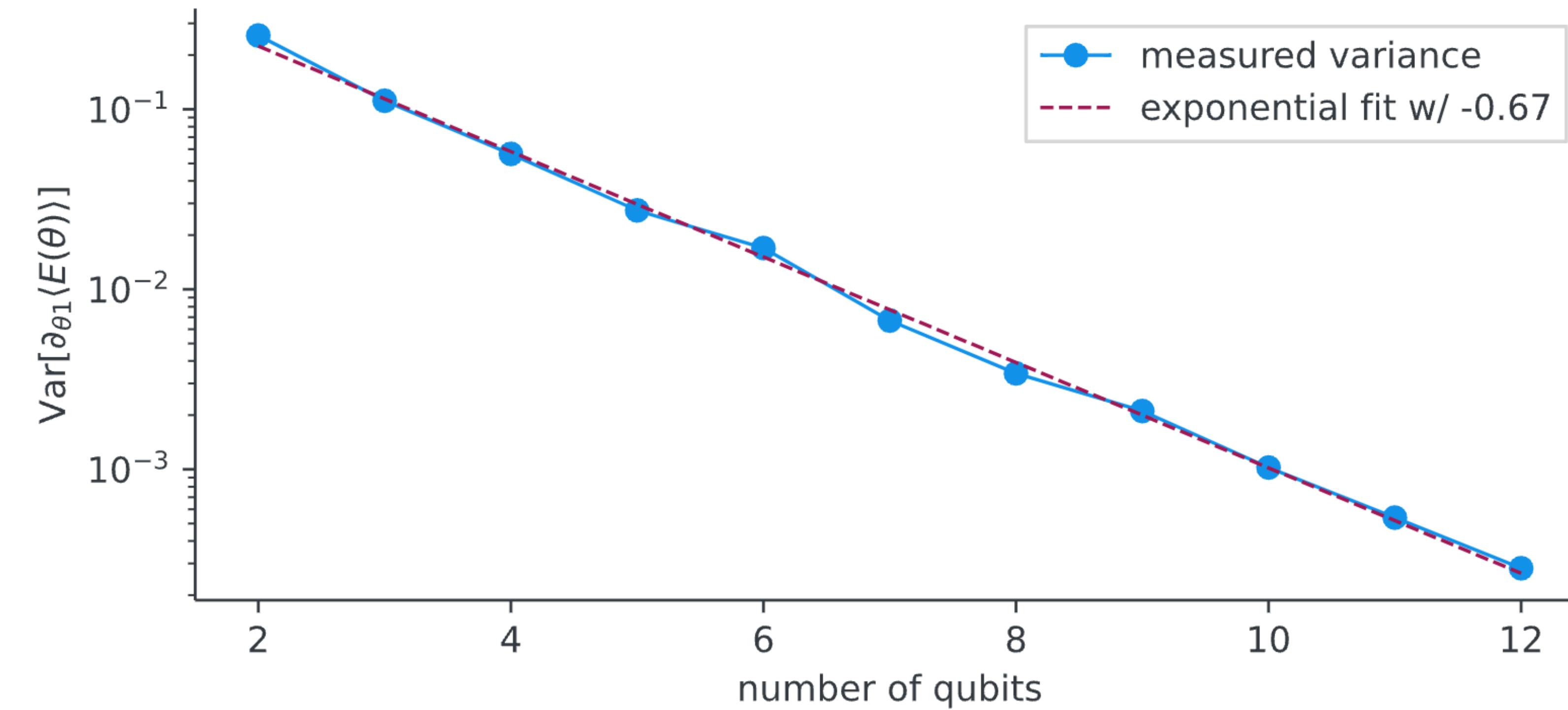
- Vanishing Gradients problem when circuit goes larger



<https://qiskit.org/learn/>

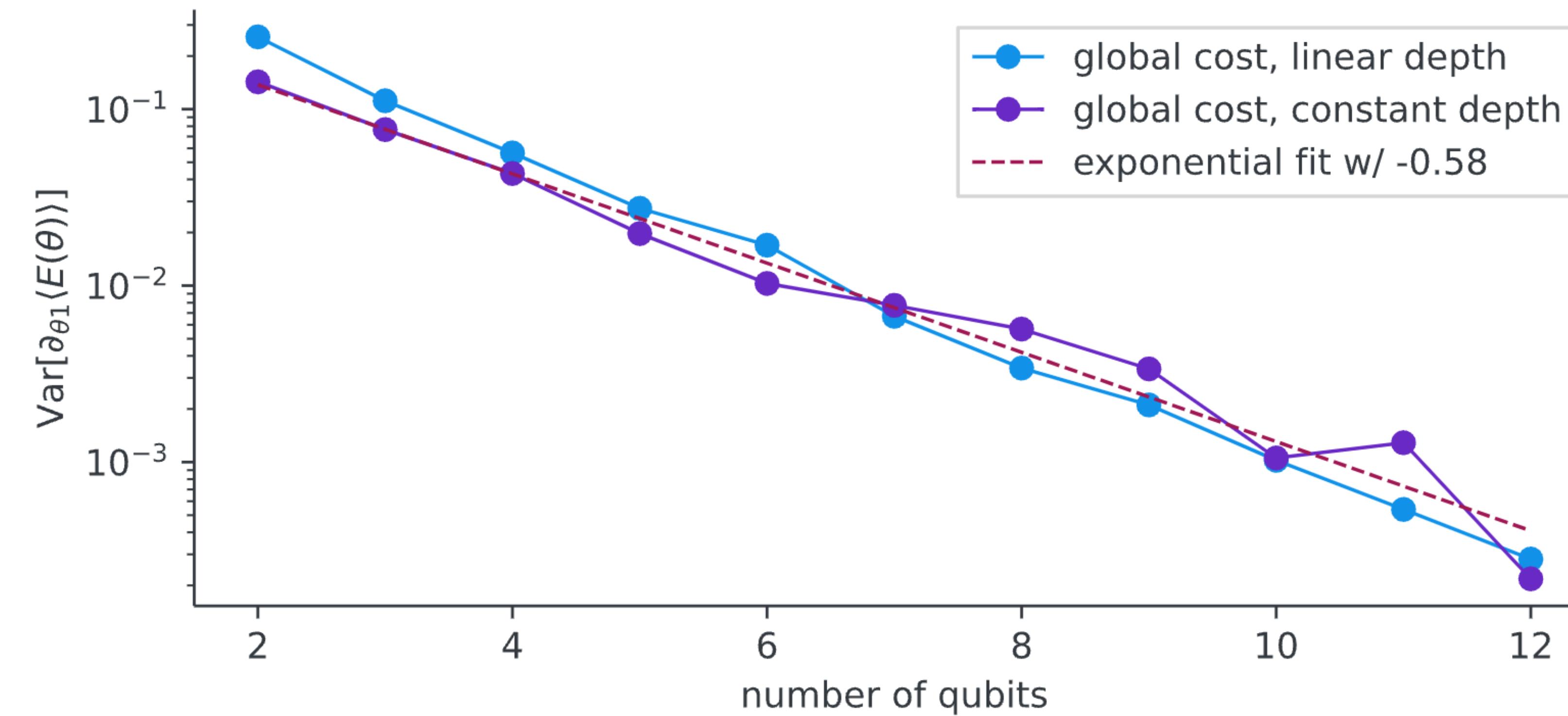
# Barren Plateaus

- Gradient **variance** reduces drastically



# Barren Plateaus

- Gradient **variance** reduces drastically

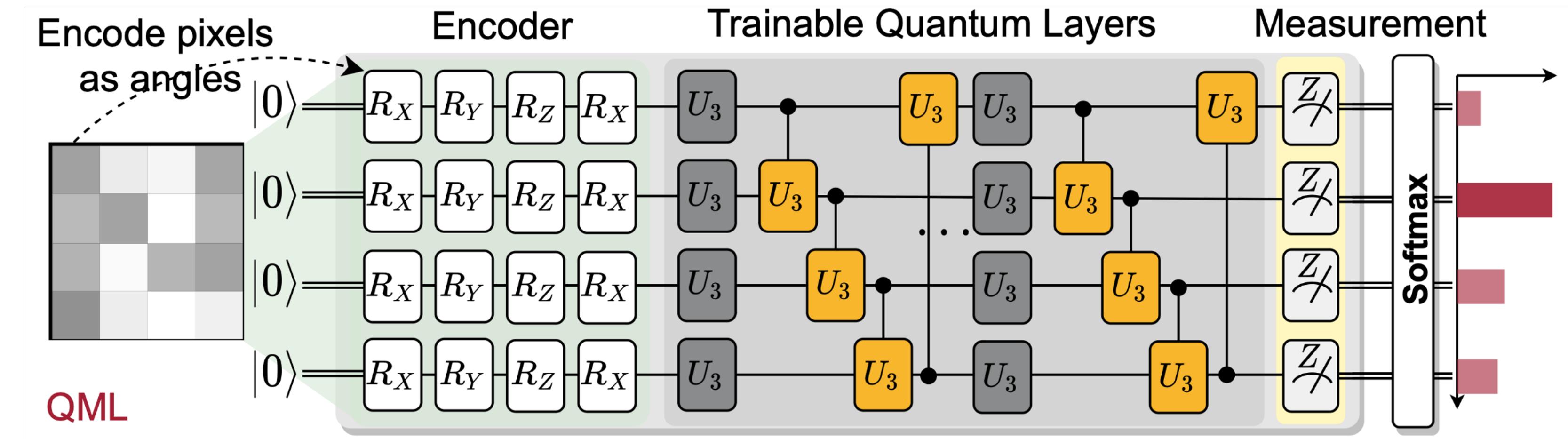


# Section 4

## Quantum Classifiers

# Quantum Classifiers

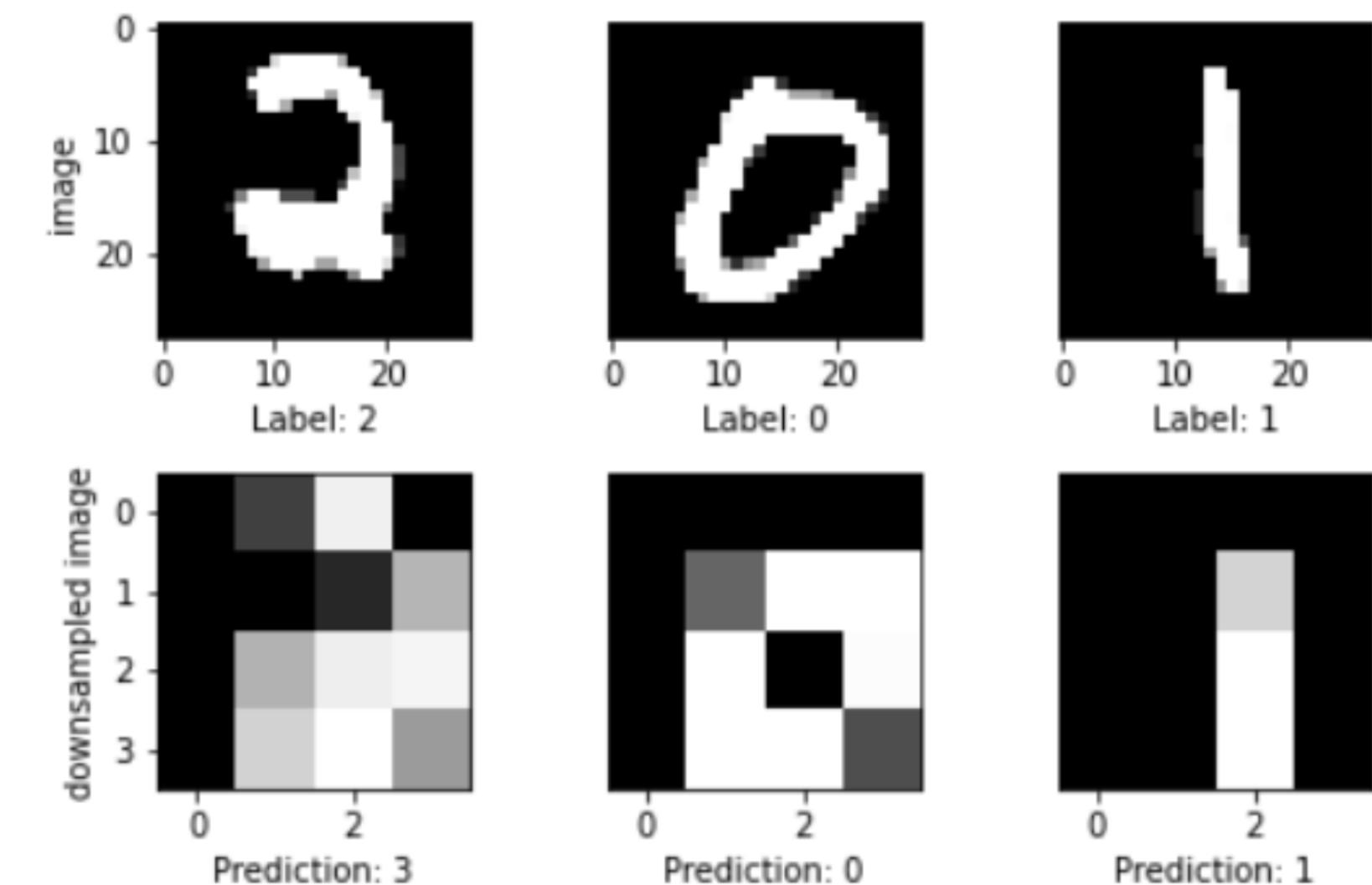
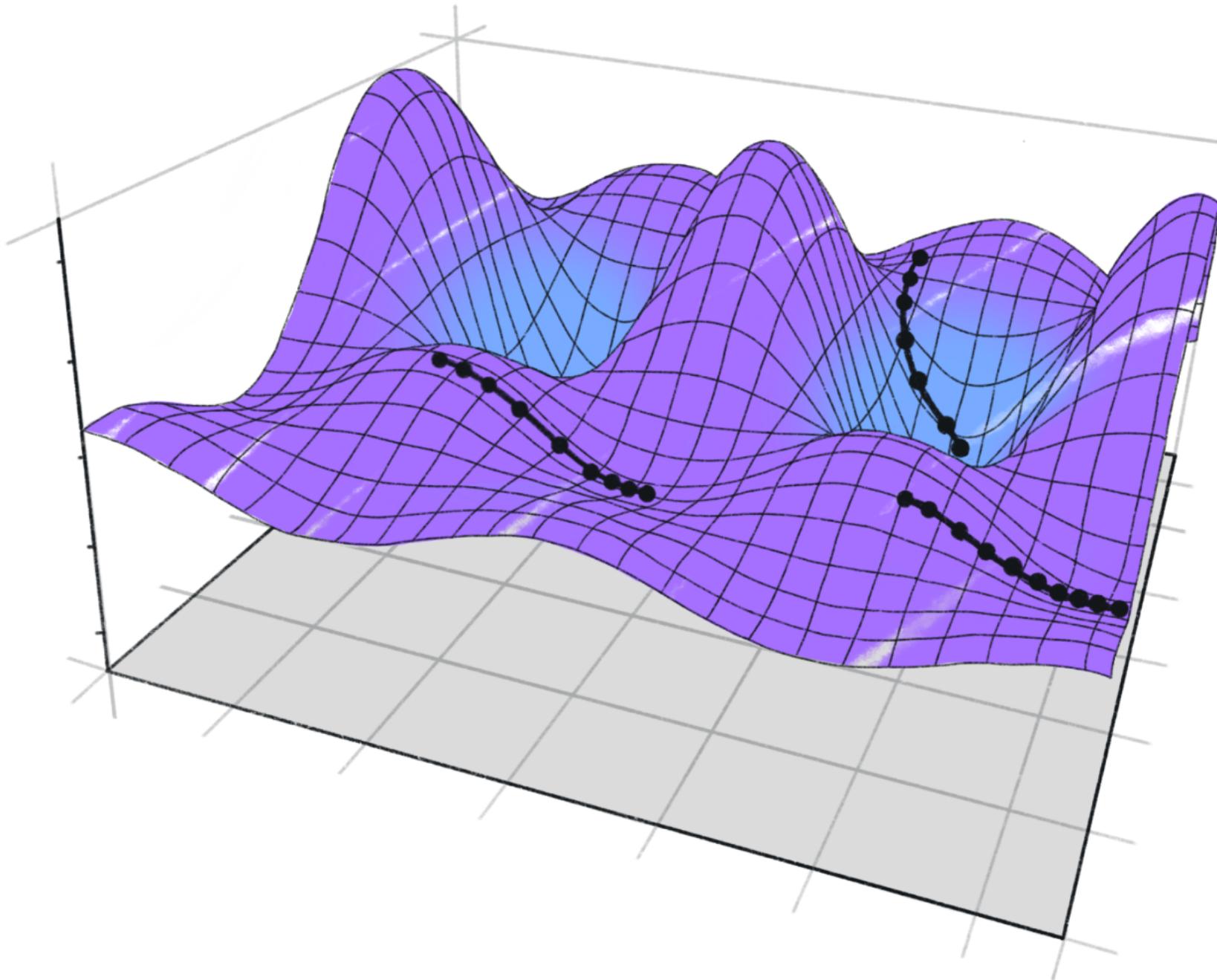
## With Quantum Neural Networks



# Quantum Classifiers

## With Quantum Neural Networks

- Training

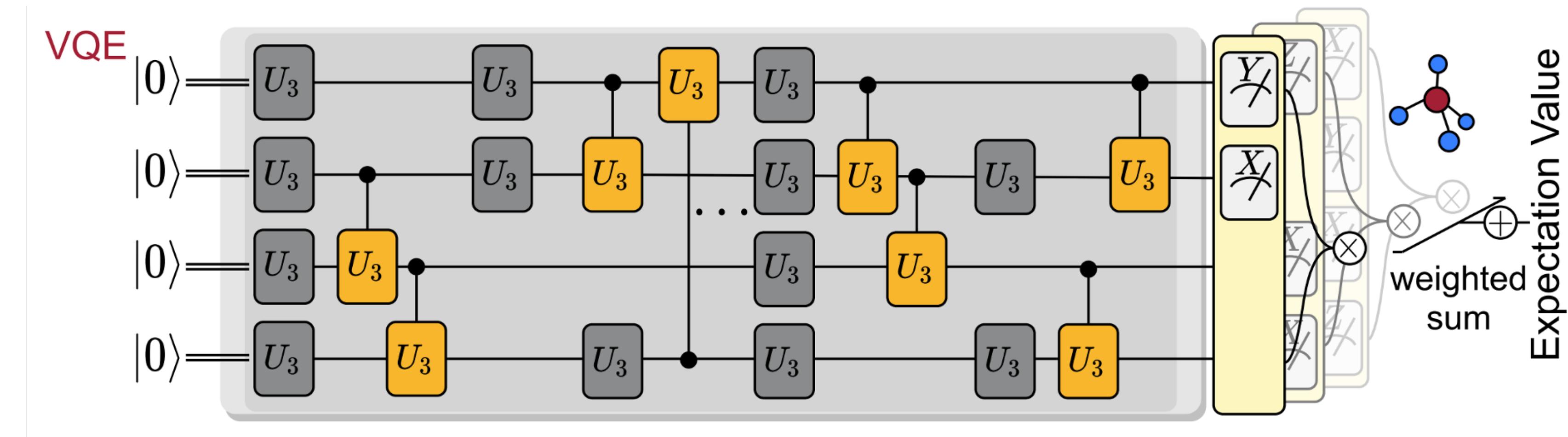


<https://qiskit.org/learn/>

# PQC for Other tasks

## VQE and QAOA

- Variational Quantum eigensolver
- Quantum Approximate Optimization Algorithm (QAOA)

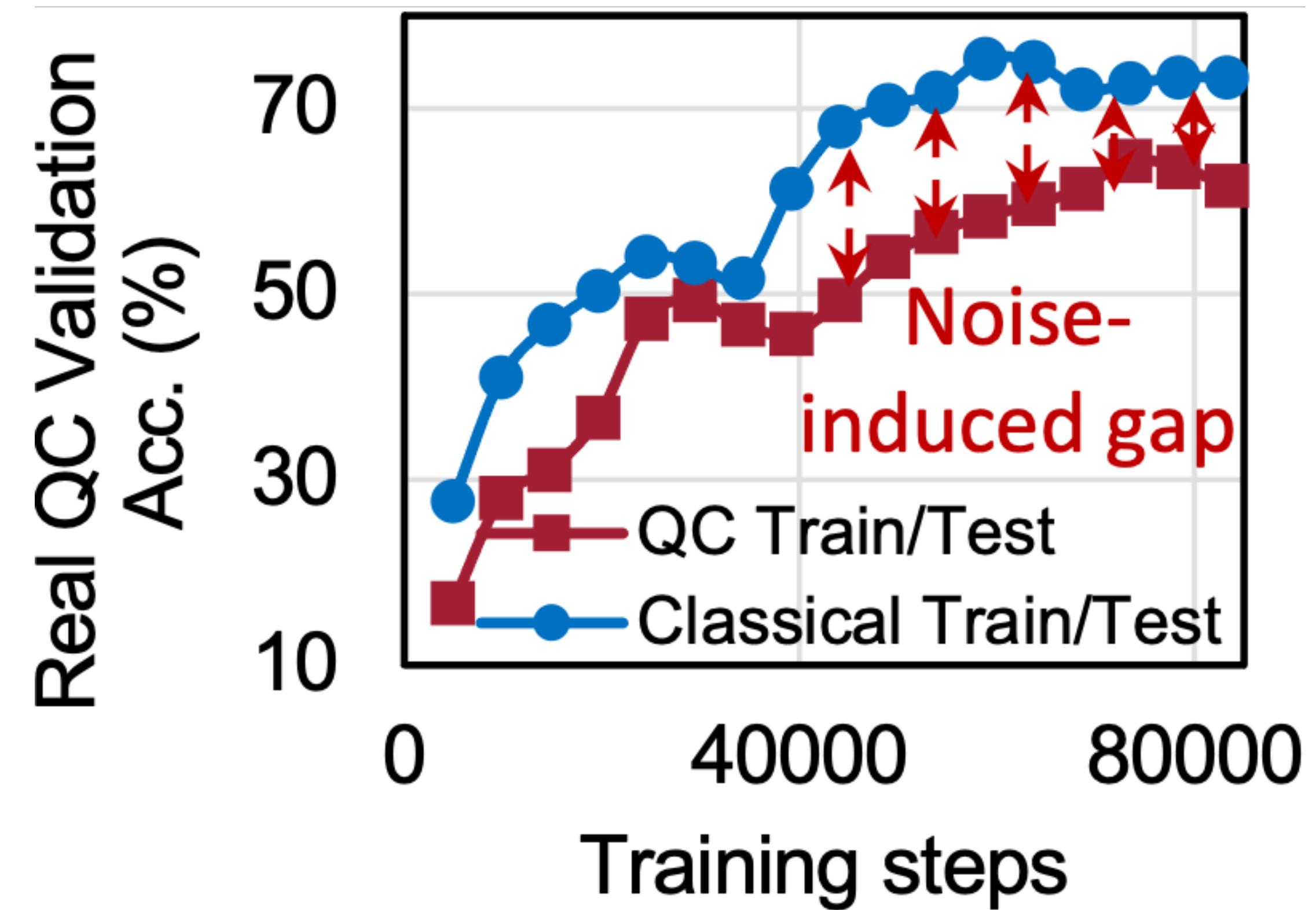


# Section 5

## Noise-Aware On-Chip Training (QOC)

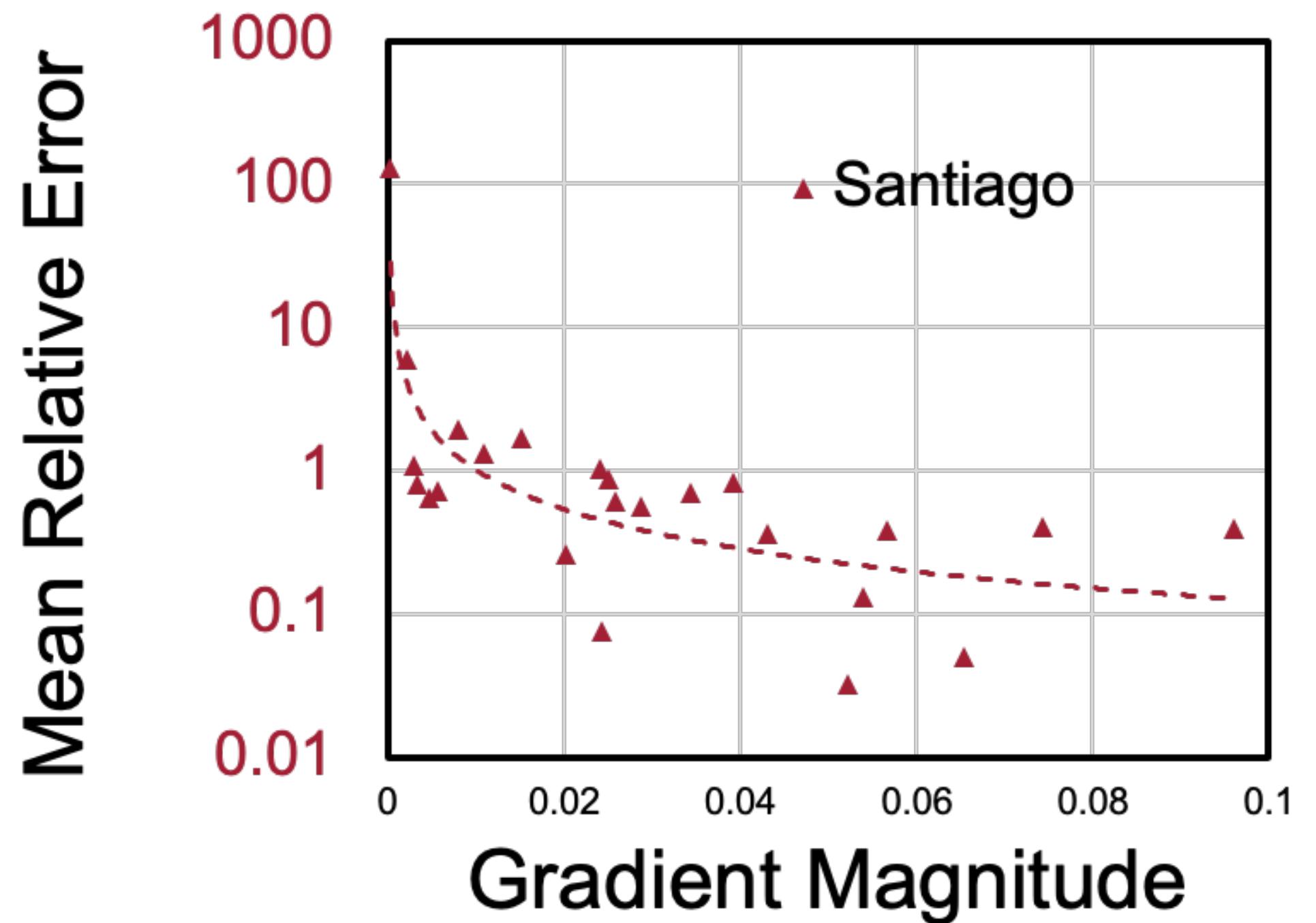
# The impact of noise on gradient computation

- Noise reduces **reliability** of on-chip computed gradients



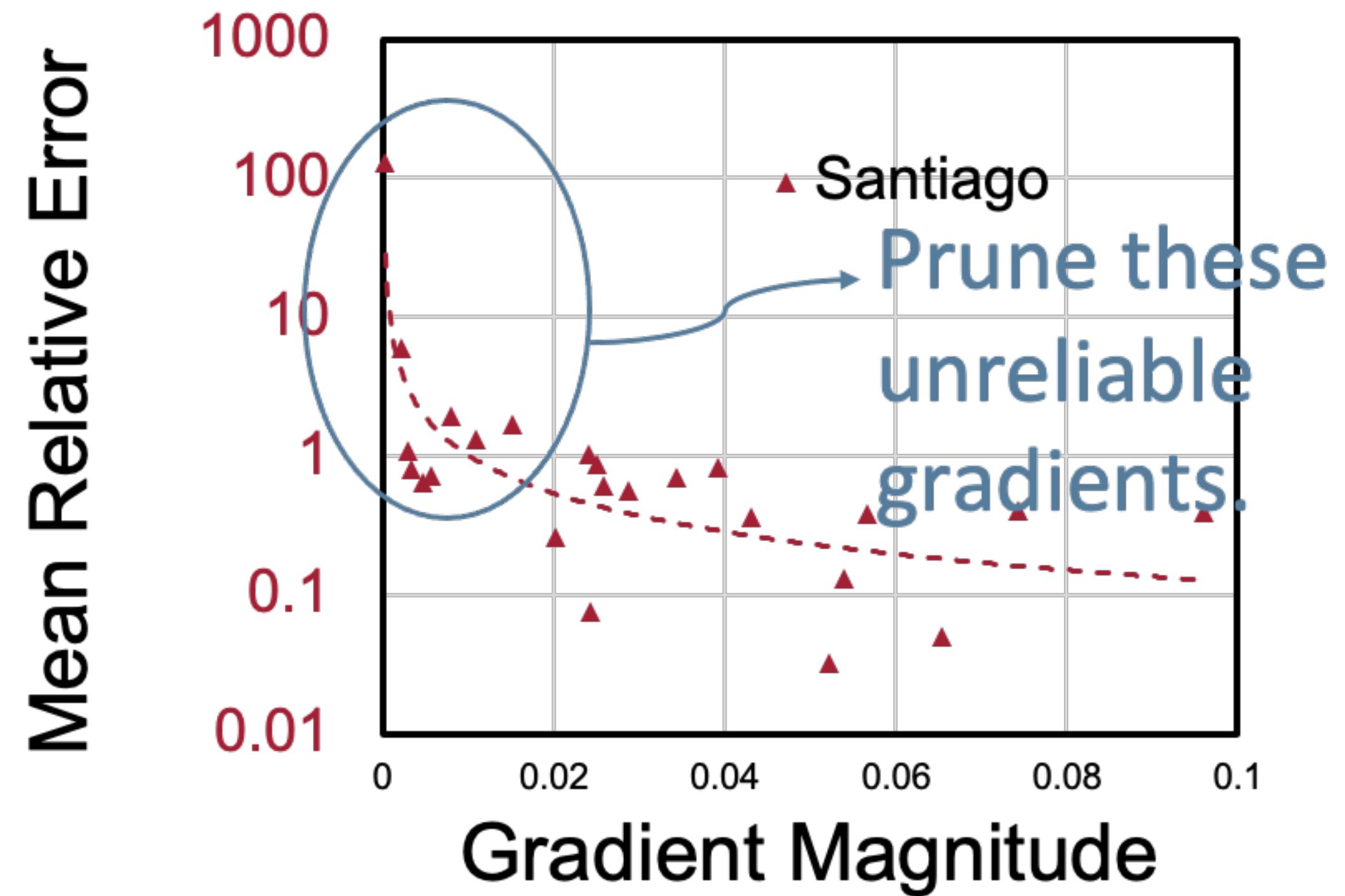
# Noise Impact

- Noise reduces **reliability** of on-chip computed gradients
- **Small magnitude** gradients have large relative errors



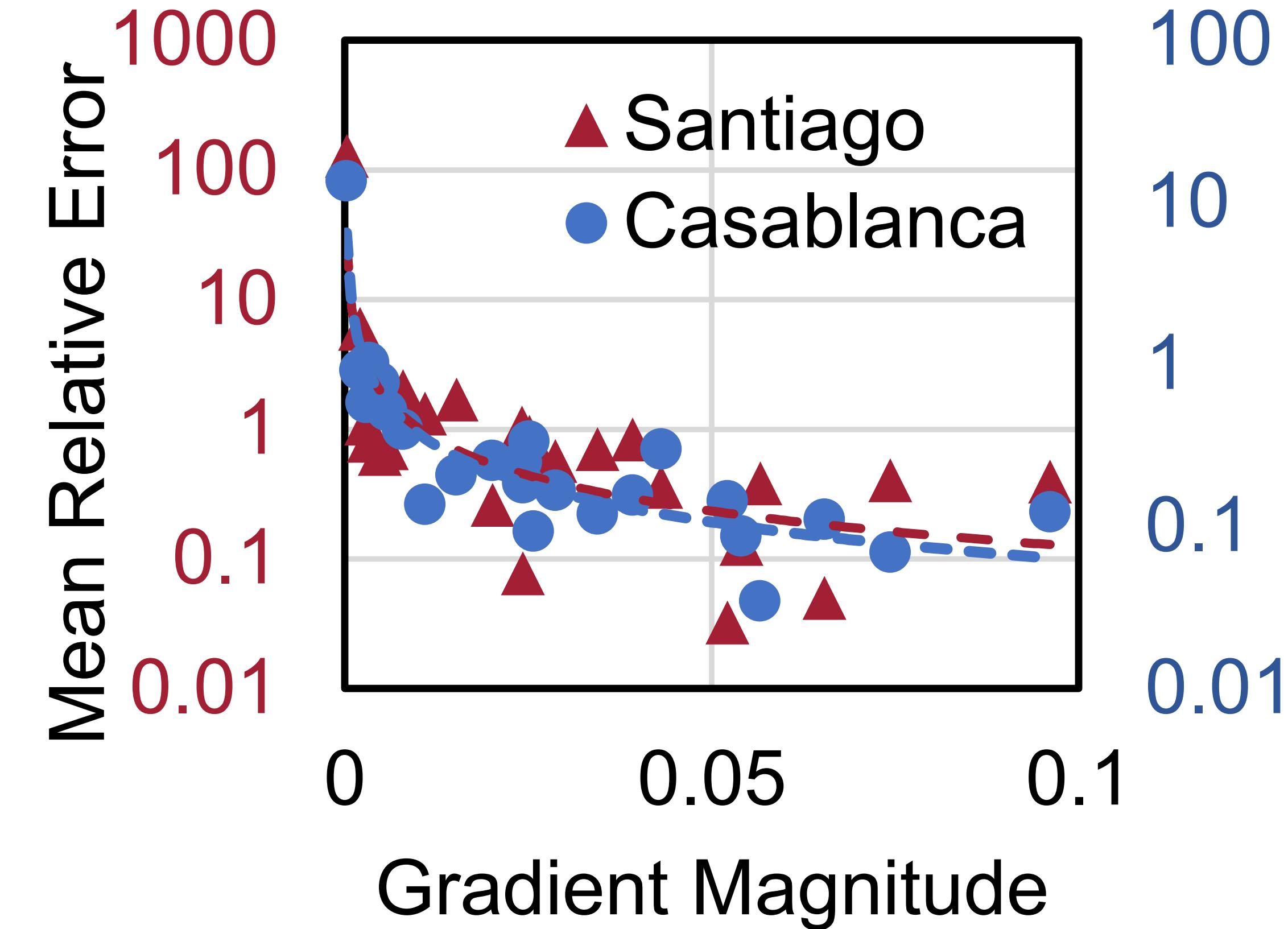
# Noise Impact

- Noise reduces **reliability** of on-chip computed gradients
- **Small magnitude** gradients have large relative errors



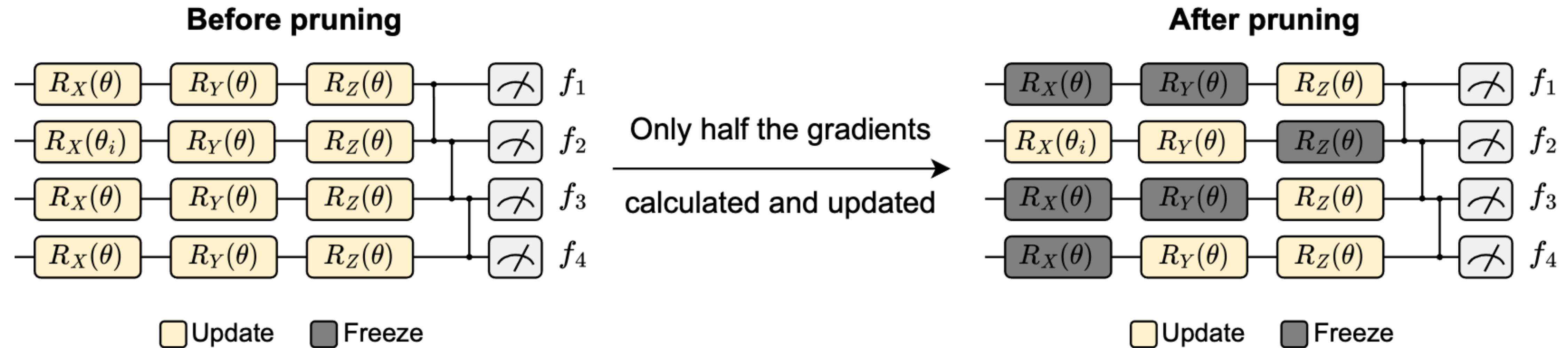
# Probabilistic Gradient Pruning

- Small magnitude gradients have **large relative errors**



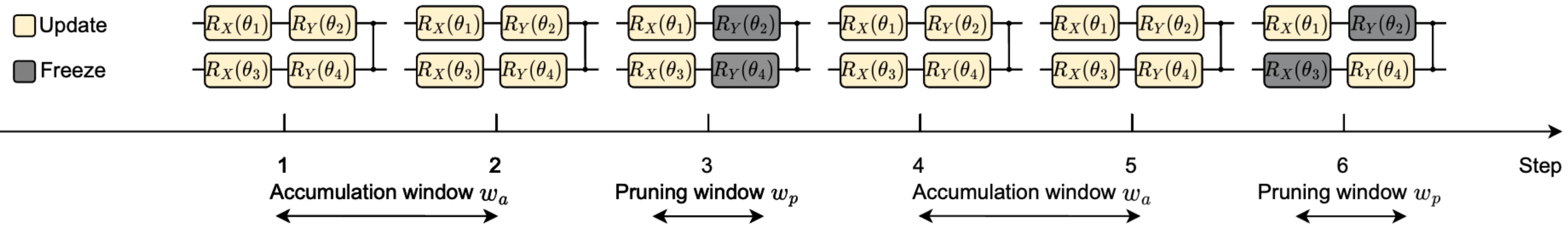
# Probabilistic Gradient Pruning

- Small magnitude gradients have **large relative errors**



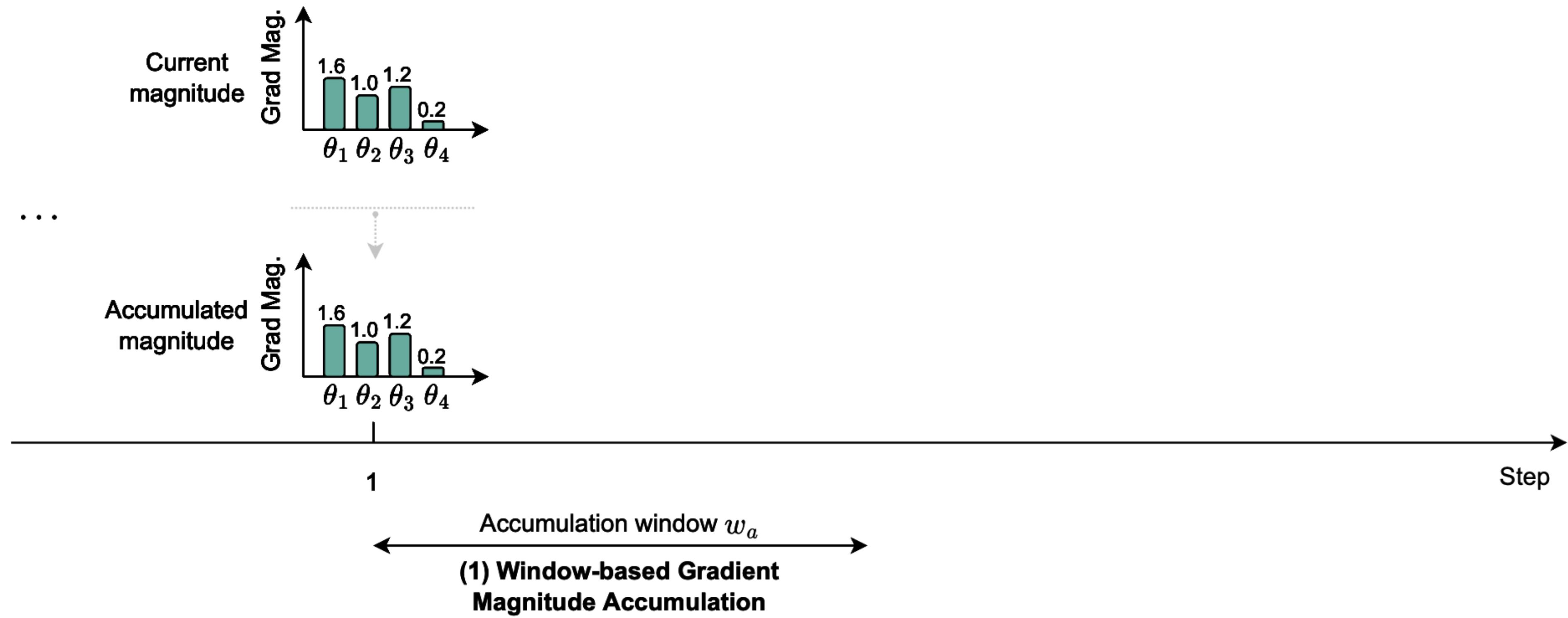
# Probabilistic Gradient Pruning

- **Accumulation Window** followed by **Pruning Window** repeatedly



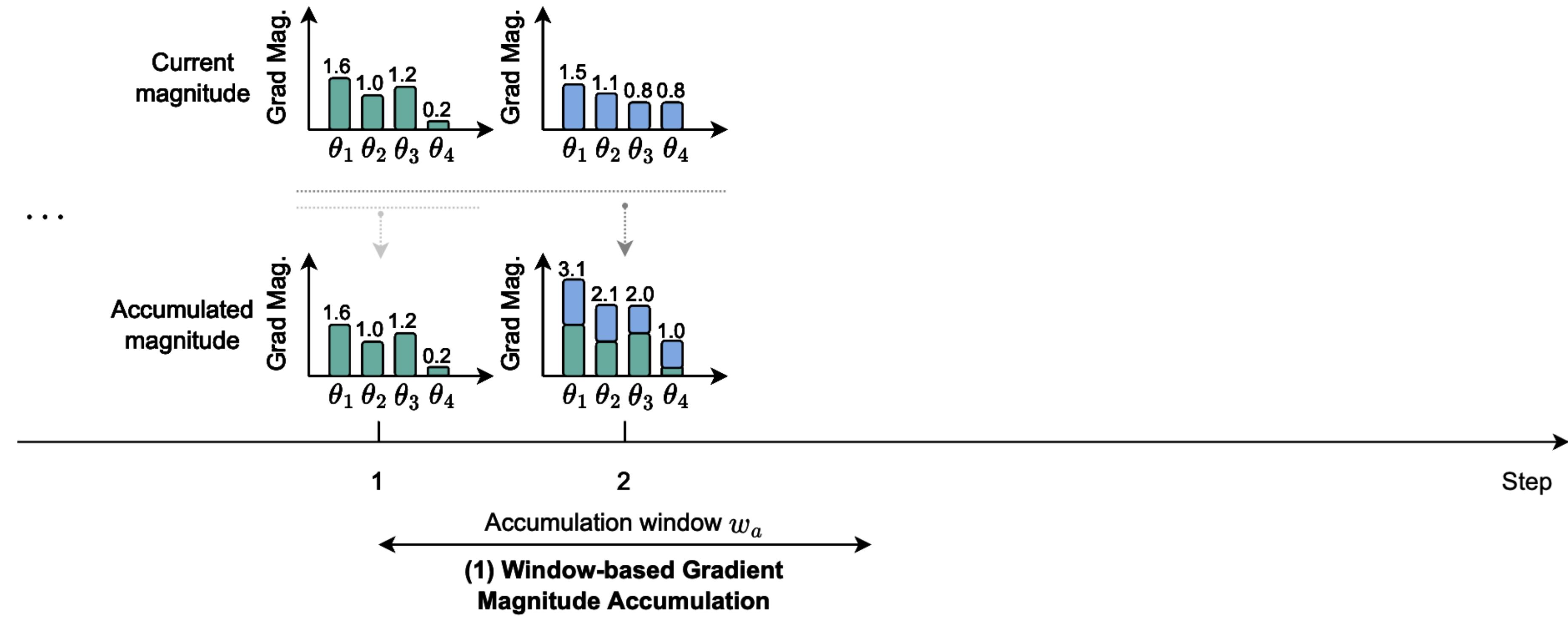
# Accumulation Window

- Keep a **record** of accumulated gradient magnitude



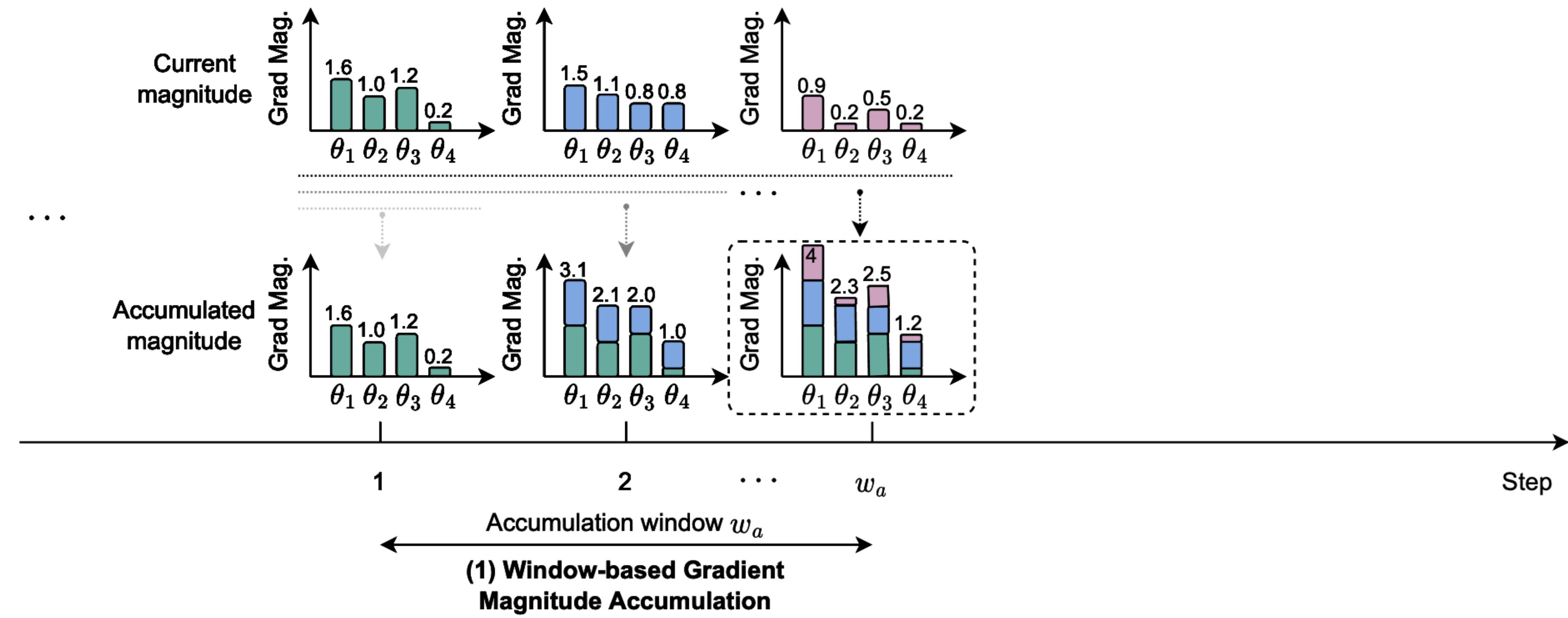
# Accumulation Window

- Keep a **record** of accumulated gradient magnitude.



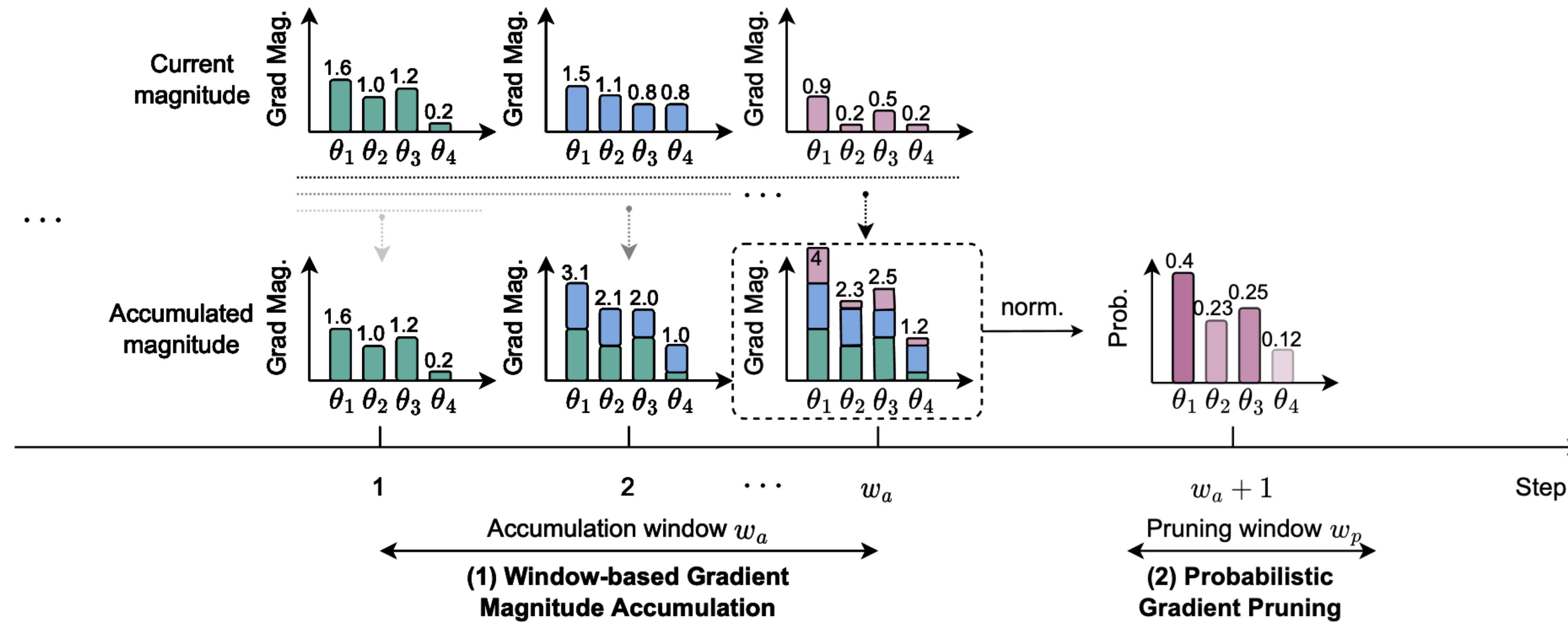
# Accumulation Window

- Keep a **record** of accumulated gradient magnitude



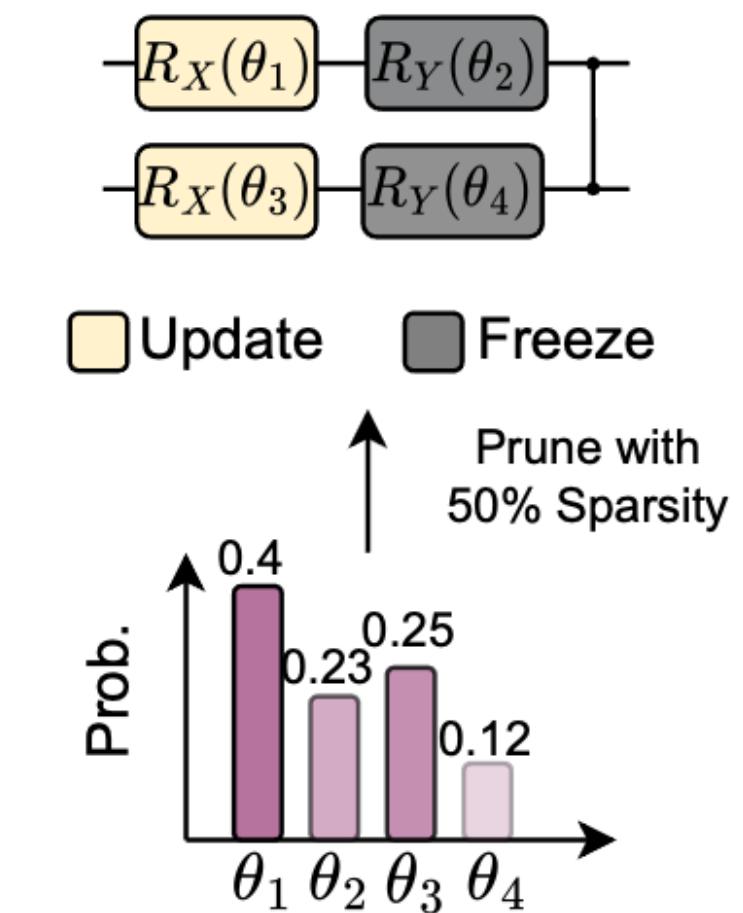
# Pruning Window

- Normalize the accumulated gradient magnitude to a probability distribution



# Pruning Window

- **Prune** the calculation of some gradients according to the probability distribution



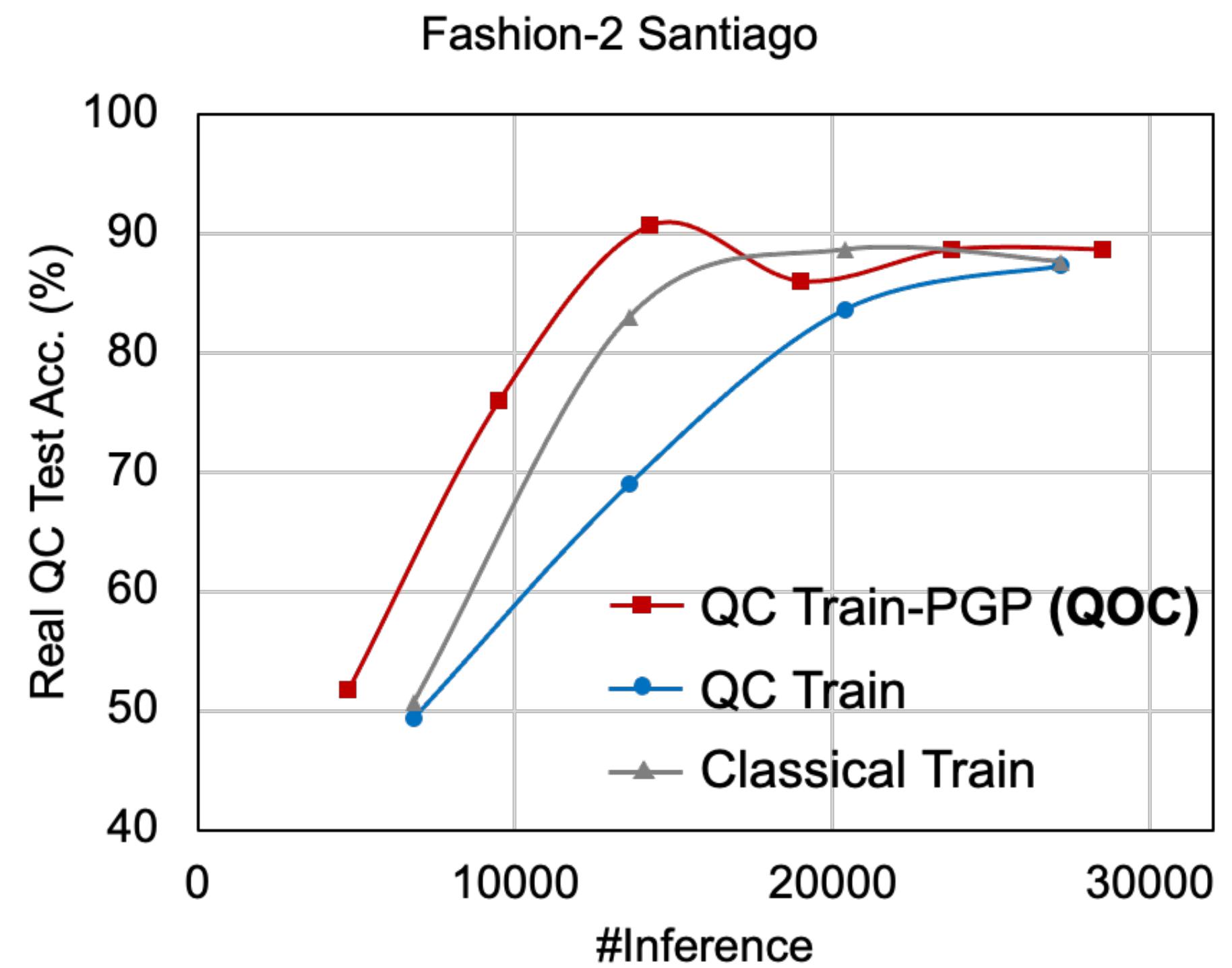
# Classification Results

- Gradient achieves similar results to classical simulation

Method	Acc.	MNIST-4	MNIST-2	Fashion-4	Fashion-2	Vowel-4
		Jarkata	Jarkata	Manila	Santiago	Lima
Classical-Train	Simu.	0.61	0.88	0.73	0.89	0.37
Classical-Train		0.59	0.79	0.54	0.89	0.31
QC-Train	QC	0.59	0.83	0.49	0.84	0.34
<b>QC-Train-PGP</b>		<b>0.64</b>	<b>0.86</b>	<b>0.57</b>	<b>0.91</b>	<b>0.36</b>

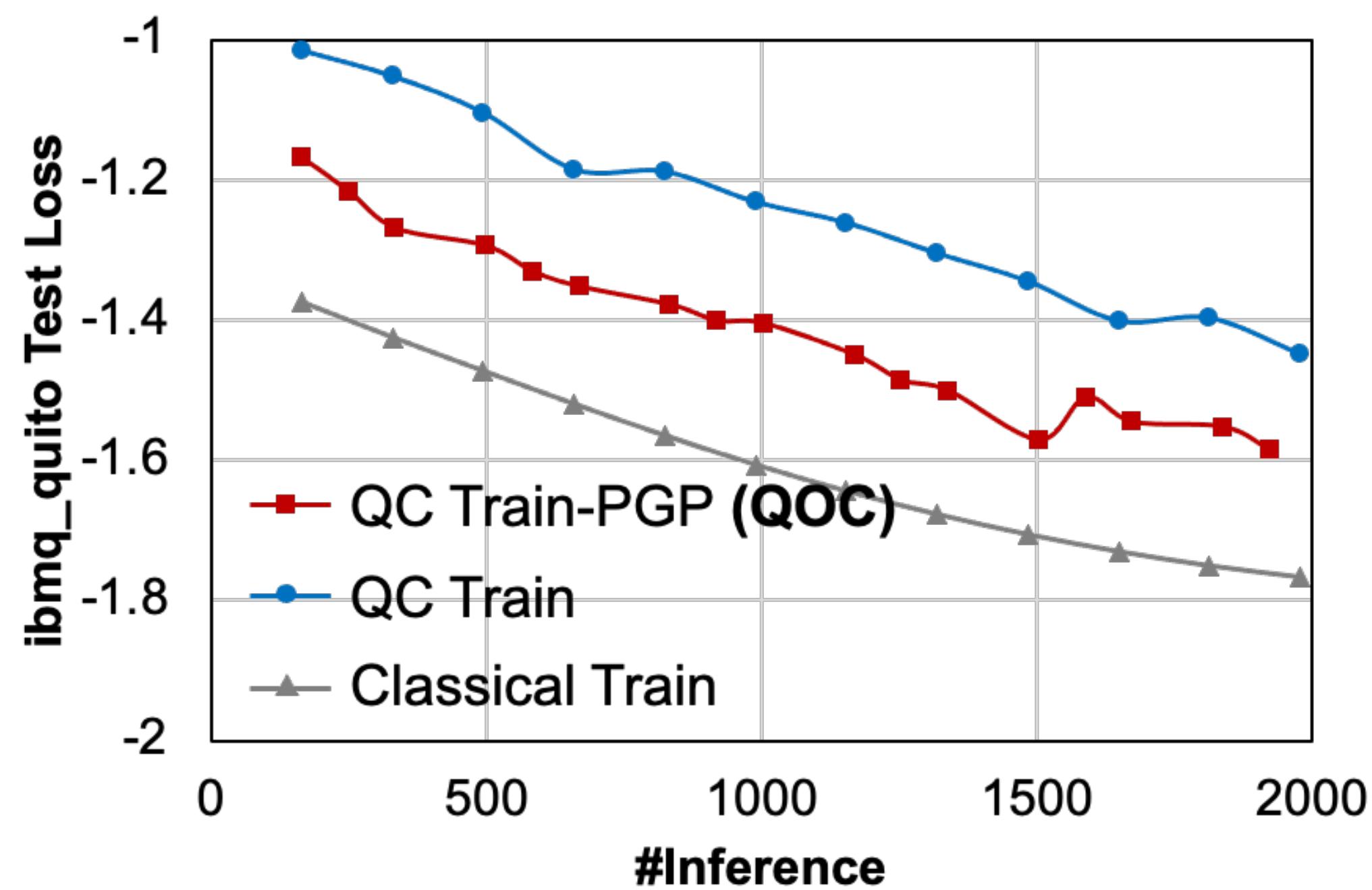
# Classification Results

- Gradient pruning can brings **2%~4% accuracy** improvements
- Pruning accelerates convergence with 2x training time reduction



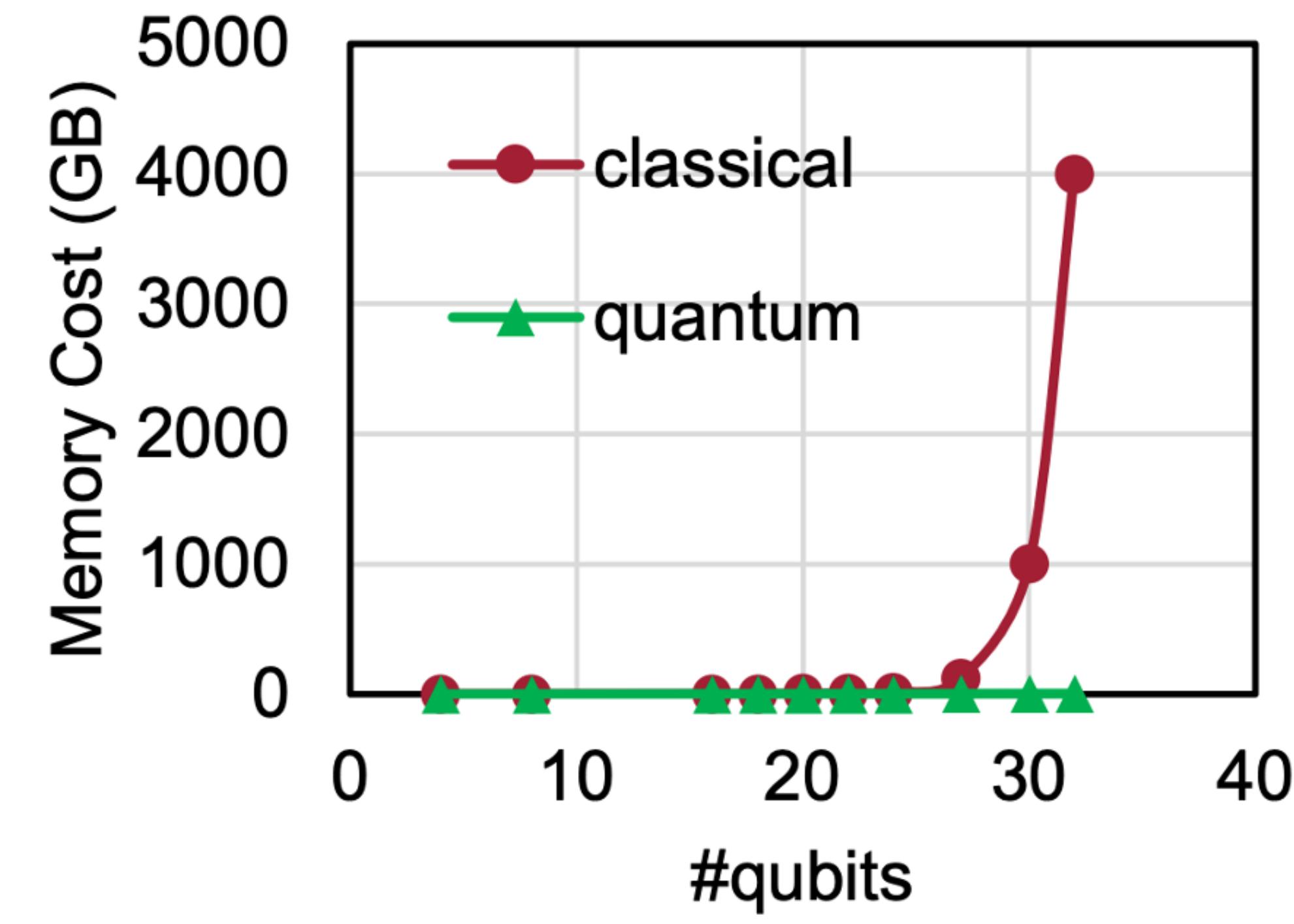
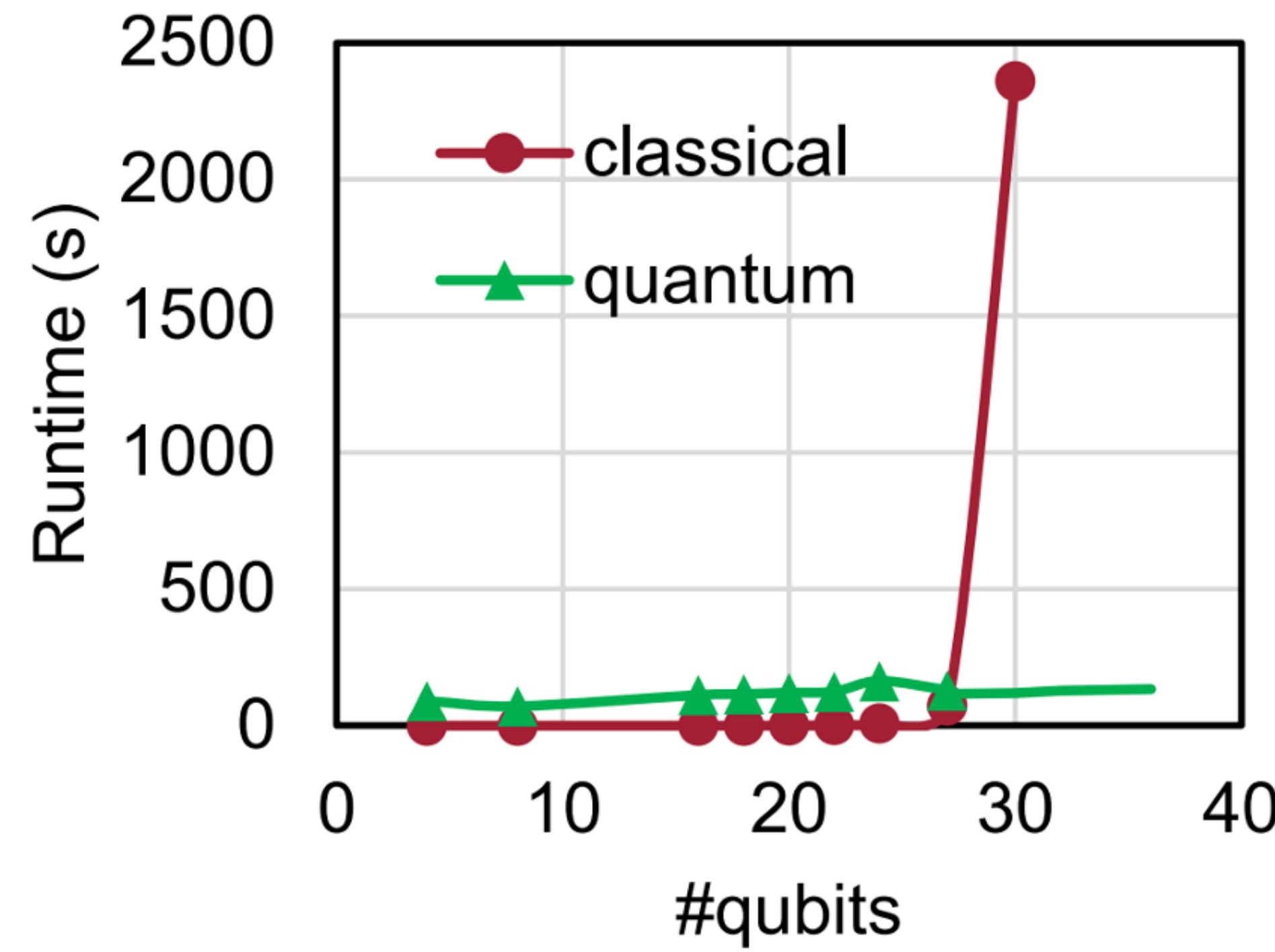
# VQE Results

- Gradient pruning can **reduce the gap** between quantum and classical



# Scalability

- On-chip Training is scalable



# Probabilistic Pruning

- Probabilistic pruning can provide better results

Method	MNIST-4	MNIST-2	Fashion-4	Fashion-2
Deterministic	0.61	0.82	0.72	0.89
Probabilistic	<b>0.62</b>	<b>0.85</b>	<b>0.79</b>	<b>0.90</b>

# TorchQuantum

Good Infrastructure is Critical



Torch  
Quantum

# TorchQuantum

## Good Infrastructure is Critical

- To enable ML-assisted hardware-aware quantum algorithm design
- Need a simulation framework on classical computer
  - **Fast speed**
  - **Convenience:** PyTorch native
  - **Portable** between different frameworks with common Quantum IR
  - Scalable
  - Analyze circuit behavior
  - Study noise impact
  - Develop ML model for quantum optimization

# TorchQuantum

## Good Infrastructure is Critical

- A fast library for classical simulation of quantum circuit in **PyTorch**
  - Automatic gradient computation for training **parameterized quantum circuit**
  - **GPU-accelerated** tensor processing with batch mode support
  - **Density** matrix and state vector simulators
  - **Dynamic** computation graph for easy debugging
  - Easy construction of **hybrid** classical and quantum neural networks
  - **Gate** level and **pulse** level simulation support
  - **Converters** to other frameworks such as IBM Qiskit
  - And so on...

# Construct a QNN with TorchQuantum

Initialize a quantum device

```
import torch.nn as nn
import torch.nn.functional as F
import torchquantum as tq
import torchquantum.functional as tqf

class QFCModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.n_wires = 4
        self.q_device = tq.QuantumDevice(n_wires=self.n_wires)
        self.measure = tq.MeasureAll(tq.PauliZ)
```

Specify encoder gates

```
self.encoder_gates = [tqf.rx] * 4 + [tqf.ry] * 4 + \
                     [tqf.rz] * 4 + [tqf.rx] * 4
```

Specify trainable gates

```
self.rx0 = tq.RX(has_params=True, trainable=True)
self.ry0 = tq.RY(has_params=True, trainable=True)
self.rz0 = tq.RZ(has_params=True, trainable=True)
self.crx0 = tq.CRX(has_params=True, trainable=True)
```

# Construct a QNN with TorchQuantum

Reset statevector

```
def forward(self, x):
    bsz = x.shape[0]
    # down-sample the image
    x = F.avg_pool2d(x, 6).view(bsz, 16)

    # reset qubit states
    self.q_device.reset_states(bsz)
```

Encode classical pixels

```
# encode the classical image to quantum domain
for k, gate in enumerate(self.encoder_gates):
    gate(self.q_device, wires=k % self.n_wires, params=x[:, k])
```

Apply the trainable gates

```
# add some trainable gates (need to instantiate ahead of time)
self.rx0(self.q_device, wires=0)
self.ry0(self.q_device, wires=1)
self.rz0(self.q_device, wires=3)
self.crx0(self.q_device, wires=[0, 2])
```

Apply some non-trainable gates

```
# add some more non-parameterized gates (add on-the-fly)
tqf.hadamard(self.q_device, wires=3)
tqf.sx(self.q_device, wires=2)
tqf.cnot(self.q_device, wires=[3, 0])
tqf.qubitunitary(self.q_device0, wires=[1, 2], params=[[1, 0, 0, 0],
                                                       [0, 1, 0, 0],
                                                       [0, 0, 0, 1j],
                                                       [0, 0, -1j, 0]])
```

Measure to get classical values

```
# perform measurement to get expectations (back to classical domain)
x = self.measure(self.q_device).reshape(bsz, 2, 2)

# classification
x = x.sum(-1).squeeze()
x = F.log_softmax(x, dim=1)

return x
```

# Summary of Today's Lecture

- We learned
  - Neutral Atom NISQ devices
  - Introduce Parameterized Quantum Circuit (PQC)
  - Introduce PQC Training
  - Introduce Quantum Classifiers
  - Introduce Noise Aware On-Chip Training of PQC
  - Introduce TorchQuantum Library for QML

**In next lecture, we will introduce:**

Noise Robust QML

