# QuantumNAS TorchQuantum interview

Hanrui Wang, Song Han
MIT HAN Lab

Torch Quantum

2022/03/04 MIT News

MIT HAN LAB

# Outline

- Background
- QuantumNAS
- RobustQNN and On-chip QNN
- TorchQuantum
- Example: TorchQuantum for MNIST

# Quantum Bit

- Quantum Bit (Qubit)
  - Statevector: contains $2^n$ complex numbers for n qubit system
  - The square sum of magnitude of $2^n$ numbers are 1
  - 1 qubit:

$$\begin{bmatrix} a_0 \\ a_1 \end{bmatrix} \quad a_0, a_1 \in \mathbb{C}$$
$$|a_0|^2 + |a_1|^2 = 1$$

  - 2 qubits:

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} \quad a_0, a_1, a_2, a_3 \in \mathbb{C}$$
$$|a_0|^2 + |a_1|^2 + |a_2|^2 + |a_3|^2 = 1$$

# Quantum Bit

- Classical bits represented in statevector
  - Classical 0: $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$
  - Classical 1: $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$
  - An arbitrary quantum states: $\begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = a_0 \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix} + a_1 \cdot \begin{bmatrix} 0 \\ 1 \end{bmatrix}$

# Quantum Gates

- Qubit gates: operations on one qubit or multiple qubits
- The qubit gates can be represented with matrix format with dimension $2^n \times 2^n$
- All gate matrices are unitary matrices: the conjugate transpose is the same as its inverse
- Single qubit gates:
  - Not (X) gate:

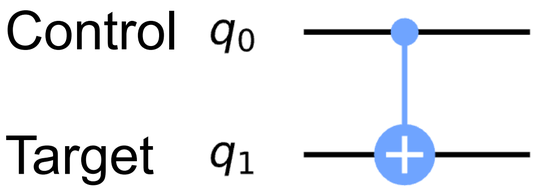
$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

  - Parameterized gate: Rotation X (RX) with parameter theta

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

# Quantum Gates

- 2-qubit gates:
  - Controlled Not (CNOT) gate:




$$CNOT = \begin{array}{c} \\ 00 \\ 01 \\ 10 \\ 11 \end{array} \begin{array}{c} \begin{array}{cccc} \text{Output} & & & \\ 00 & 01 & 10 & 11 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

Input

  - Controlled Rotation X (CRX) gate

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

# Quantum Gates

- Applying a gate to qubits is performing matrix-vector multiplication between the gate matrix and statevector
  - Apply an X gate to classical state 0, we get 1

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

  - Apply an CNOT gate to state 10, we get 11

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

# Source of Quantum Advantage

- One qubit carries more information then one classic bit
- The statevector length is exponentially to the number of qubits

# QuantumNAS: Noise-Adaptive Search for Robust Quantum Circuits [HPCA 2022]

Hanrui Wang, Yongshan Ding, Jiaqi Gu, Zirui Li, Yujun Lin, David Z. Pan, Frederic T. Chong, Song Han

Torch Quantum

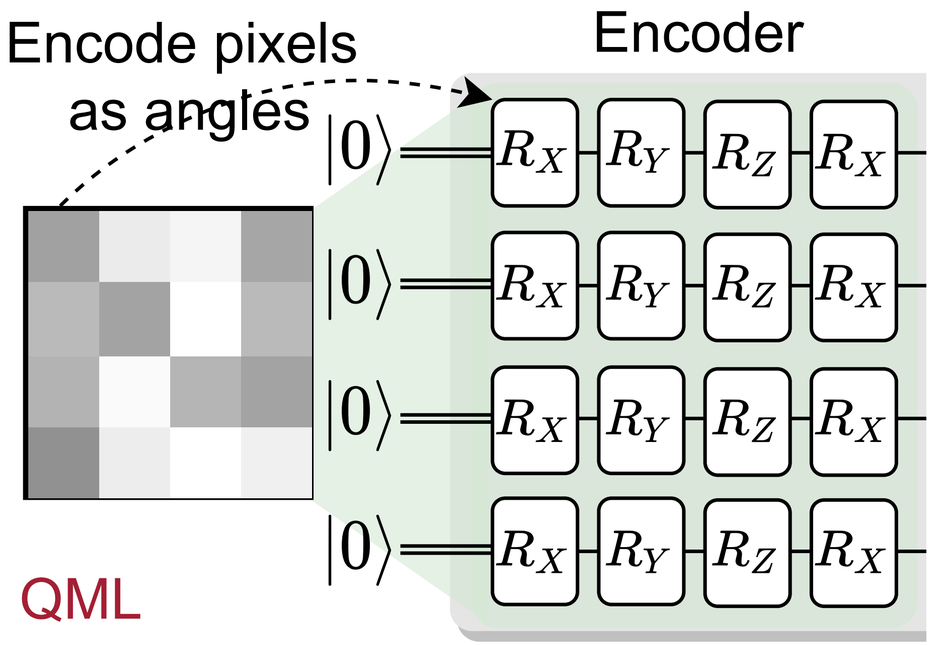
MIT HAN LAB

# QuantumNAS

- Quantum computing has potentials to bring exponential advantages
- Now, machines with tens or hundreds of qubits are available
- Currently: NISQ era
  - Insufficient qubits for powerful quantum algorithms such as Shor's algorithm
  - Imperfect qubits: quantum noise forms the bottleneck
- Hybrid quantum classical computing with parameterized quantum circuit is a promising way to achieve quantum advantage
  - VQE
  - QAOA
  - QNN
- PQC: quantum circuits with fixed gates and parameterized gates:
  - Pipeline: design the circuit architecture (Ansatz); then train the parameters to solve certain problem

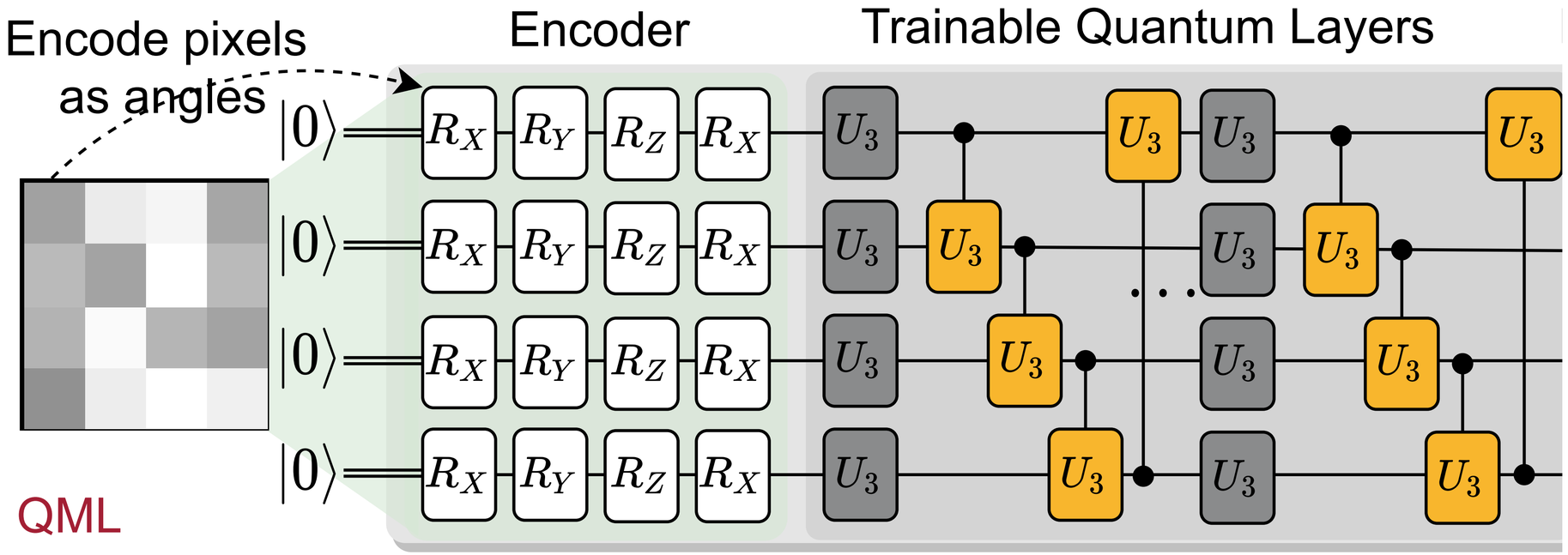
# QuantumNAS

- Parameterized quantum circuits
  - Quantum Neural Networks



# QuantumNAS

- Parameterized quantum circuits
  - Quantum Neural Networks

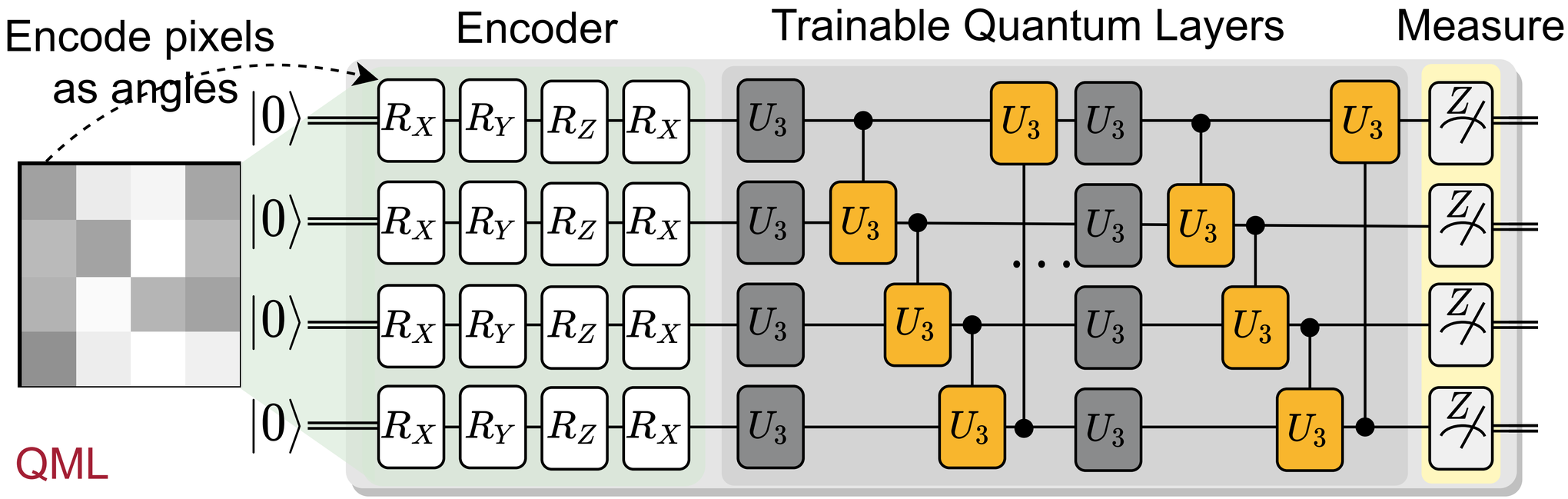



Torch Quantum

MIT HAN LAB

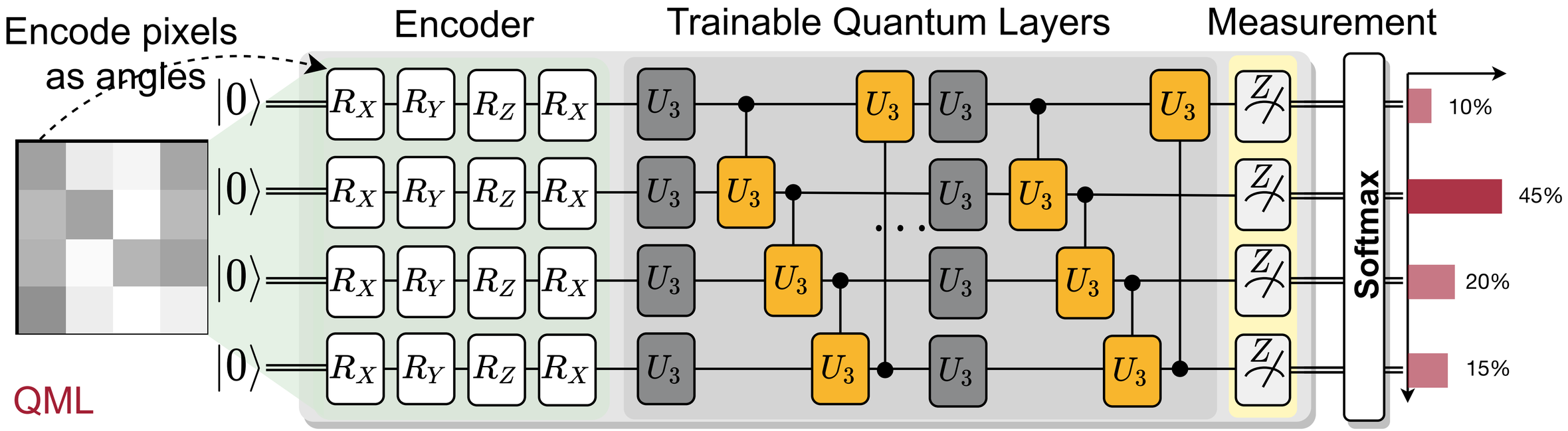
# QuantumNAS

- Parameterized quantum circuits
  - Quantum Neural Networks



# QuantumNAS

- Parameterized quantum circuits
  - Quantum Neural Networks




- Softmax $\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$

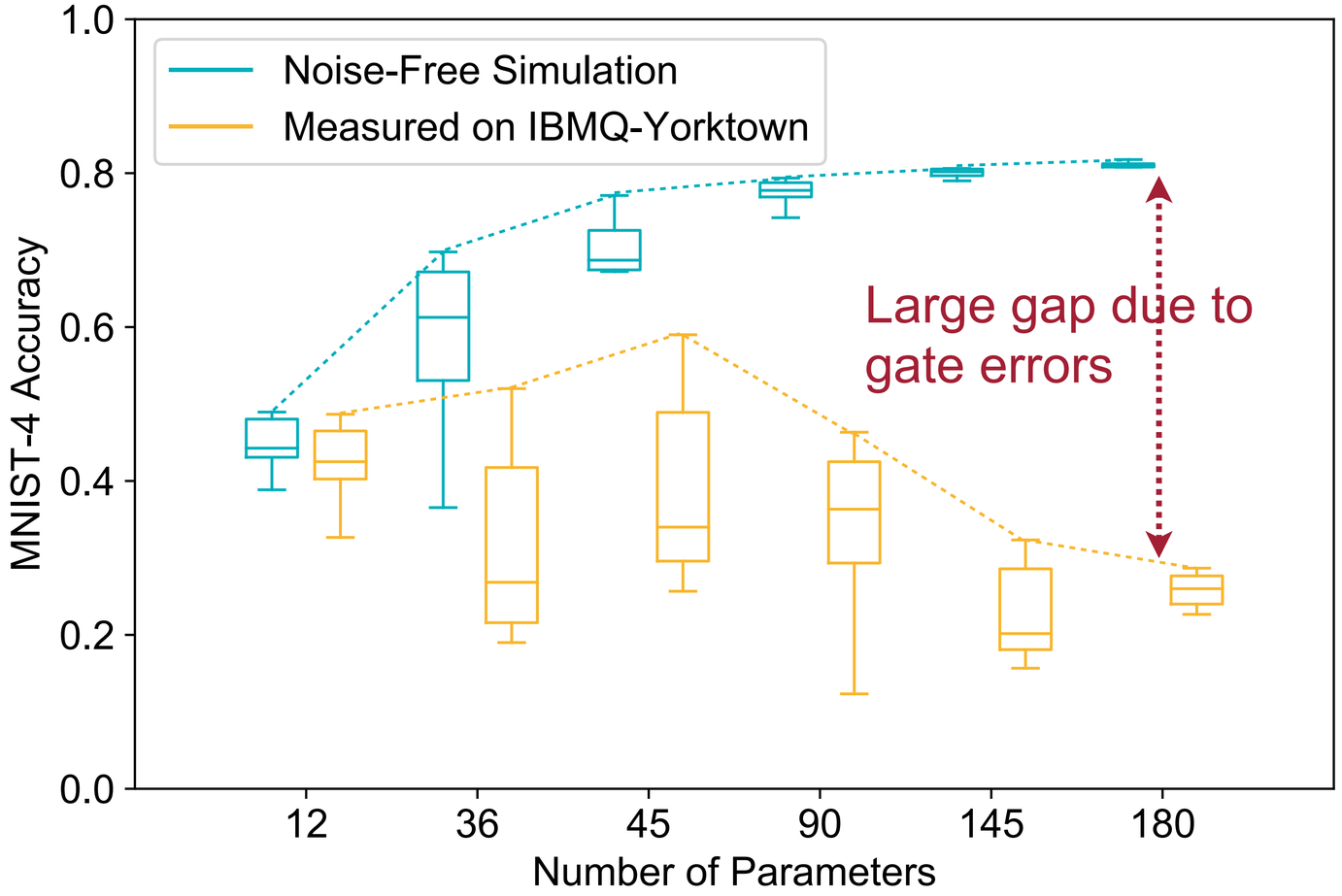
Torch Quantum

MIT HAN LAB

# QuantumNAS

- PQC challenge:
  - How to design the circuit architecture in the huge design space?
    - Naive way: in one iteration
      - Sample circuit architecture candidates
      - Train parameters
      - Select good ones
    - Very expensive
  - How to find robust circuit architecture?

# QuantumNAS




- More parameters increase the noise-free accuracy but degrade measured accuracy
- Under same #parameters, measured accuracy of different circuit ansatzes varies a lot

# QuantumNAS: Decouple the Training and Search

Naive Search

**For** q_devices:
  **For** search episodes: // meta controller
    **For** circuit training iterations:
      update_parameters(); **Expensive**
    **If** good_circuit: **break;**

# QuantumNAS: Decouple the Training and Search

Naive Search

**For** q_devices:
  **For** search episodes: // meta controller
    **For** circuit training iterations:
      update_parameters(); **Expensive**
    **If** good_circuit: **break;**

**=>**

QuantumNAS

**For *SuperCircuit*** training iterations: **Expensive**
  update_parameters(); **training**

**decouple**

**For** q_devices:
  **For** search episodes: **search**
    sample from ***SuperCircuit***; **Light-Weight**
    **If** good_circuit: **break;**
    //no training

Torch Quantum

MIT HAN LAB

# QuantumNAS

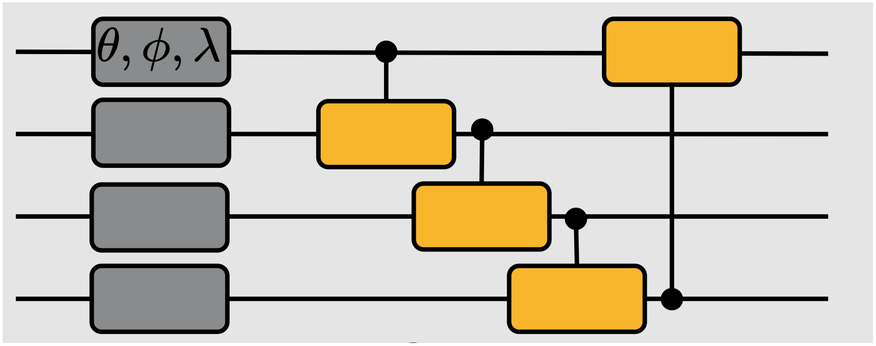
- SuperCircuit Construction and Training
- Noise-Adaptive Evolutionary Co-Search of SubCircuit and Qubit Mapping
- Searched SubCircuit Training
- Iterative Pruning

# QuantumNAS

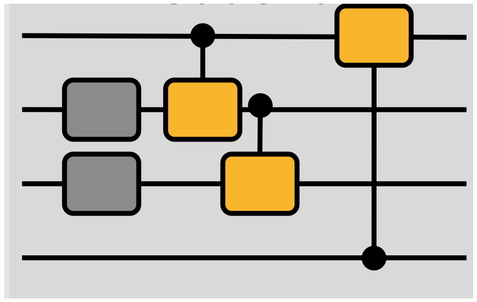
- SuperCircuit Construction and Training
- Noise-Adaptive Evolutionary Co-Search of SubCircuit and Qubit Mapping
- Searched SubCircuit Training
- Iterative Pruning

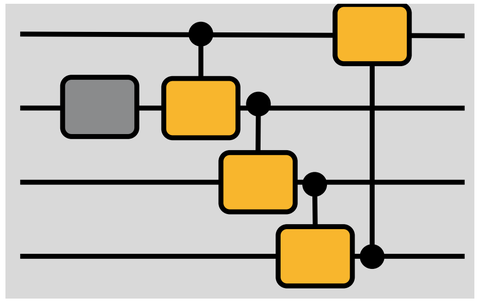
# SuperCircuit Construction

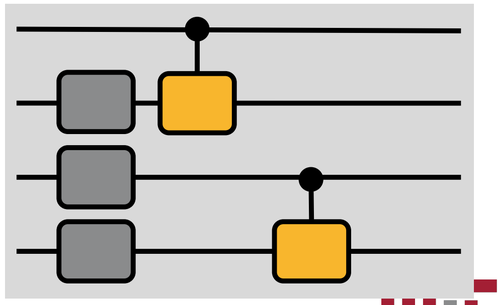
- Why construct a SuperCircuit?
  - Use it for efficient search of circuit candidates with no need to train them to final
- SuperCircuit: the circuit with the largest number of gates in the design space
  - Example: SuperCircuit in U3+CU3 space




- Each candidate circuit in the design space (called SubCircuit) is a subset of the SuperCircuit





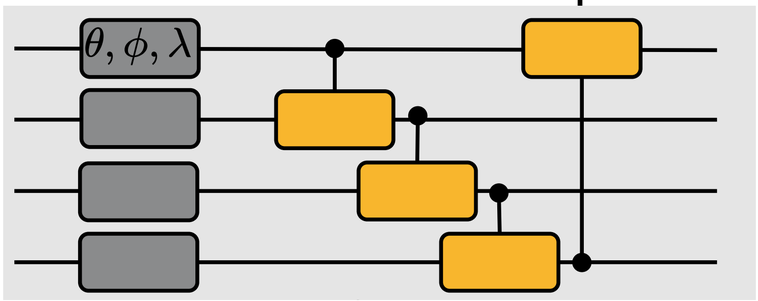


Torch Quantum

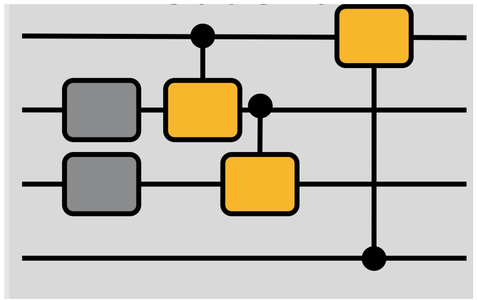
MIT HAN LAB

# SuperCircuit Training

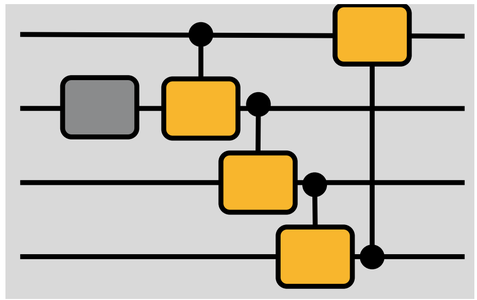
- In one SuperCircuit Training step:
  - Sample a gate subset of SuperCircuit (a SubCircuit)
  - Only use the subset to perform task and updates the parameters in the subset
  - Parameter updates are cumulative across steps
- Supercircuit:



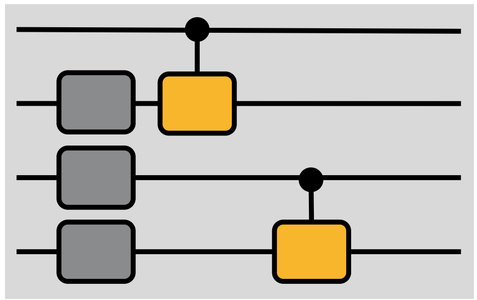

- Sampled subsets (SubCircuits of different steps)



Step1



Step2



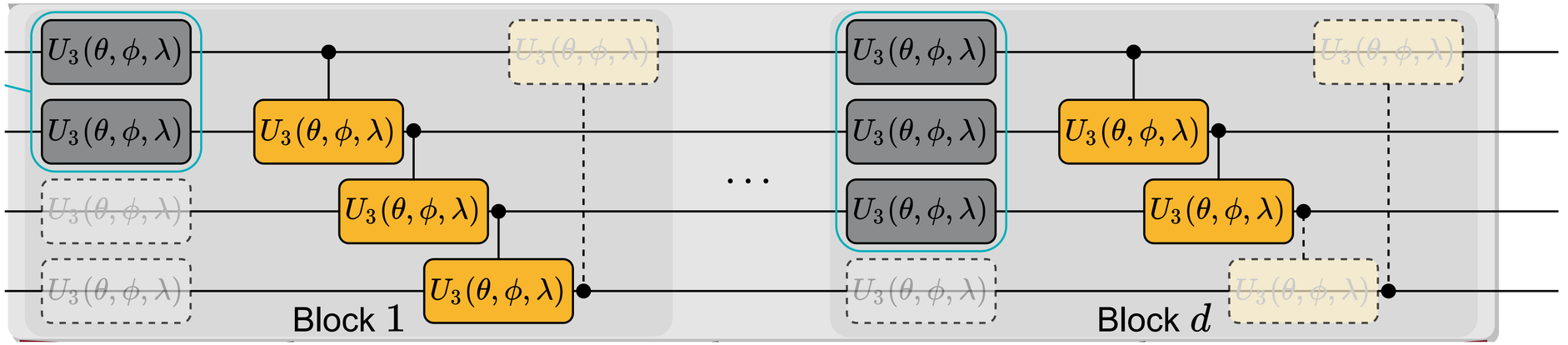
Step3

Torch Quantum

MIT HAN LAB

# More Details on Construction and Sampling

- In reality, we can stack multiple blocks of quantum layers to have enough parameters and a large design space

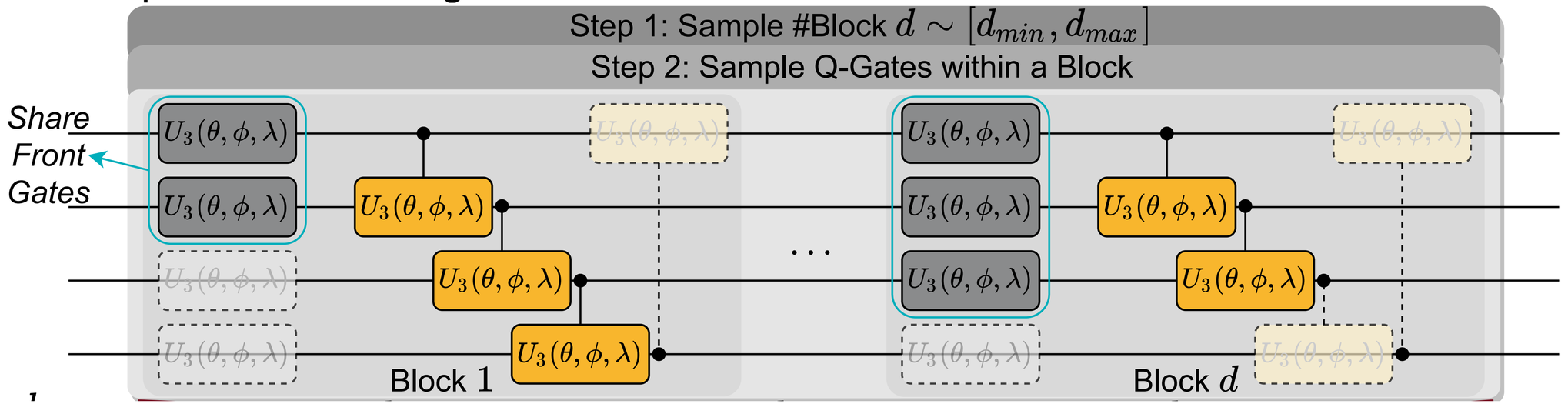



Torch Quantum

MIT HAN LAB

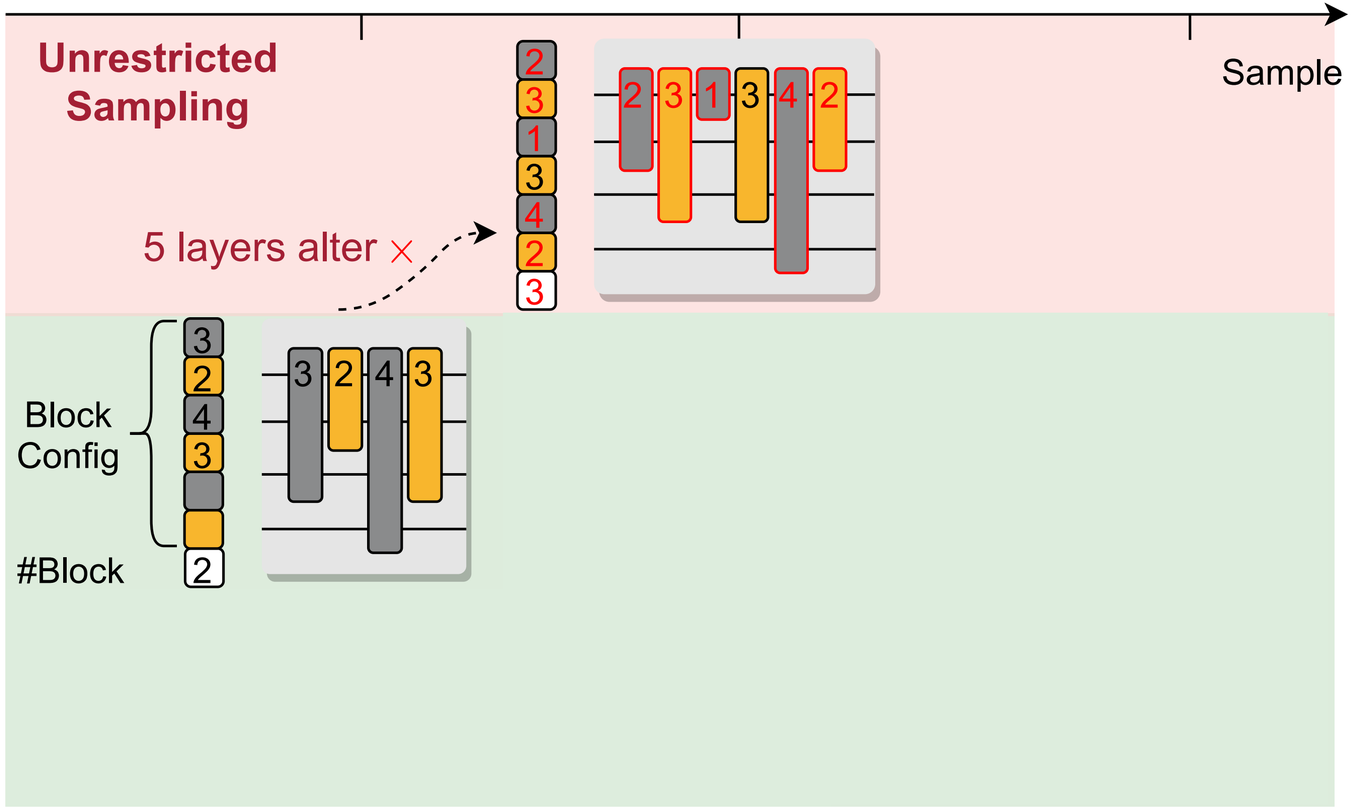
# Front Sampling

- In reality, we can stack multiple blocks of quantum layers to have enough parameters and a large design space
- During sampling, we first sample total number of blocks, then sample gates within each block
  - Front sampling: Only the **front several** blocks and front several gates can be sampled to make SuperCircuit training more stable



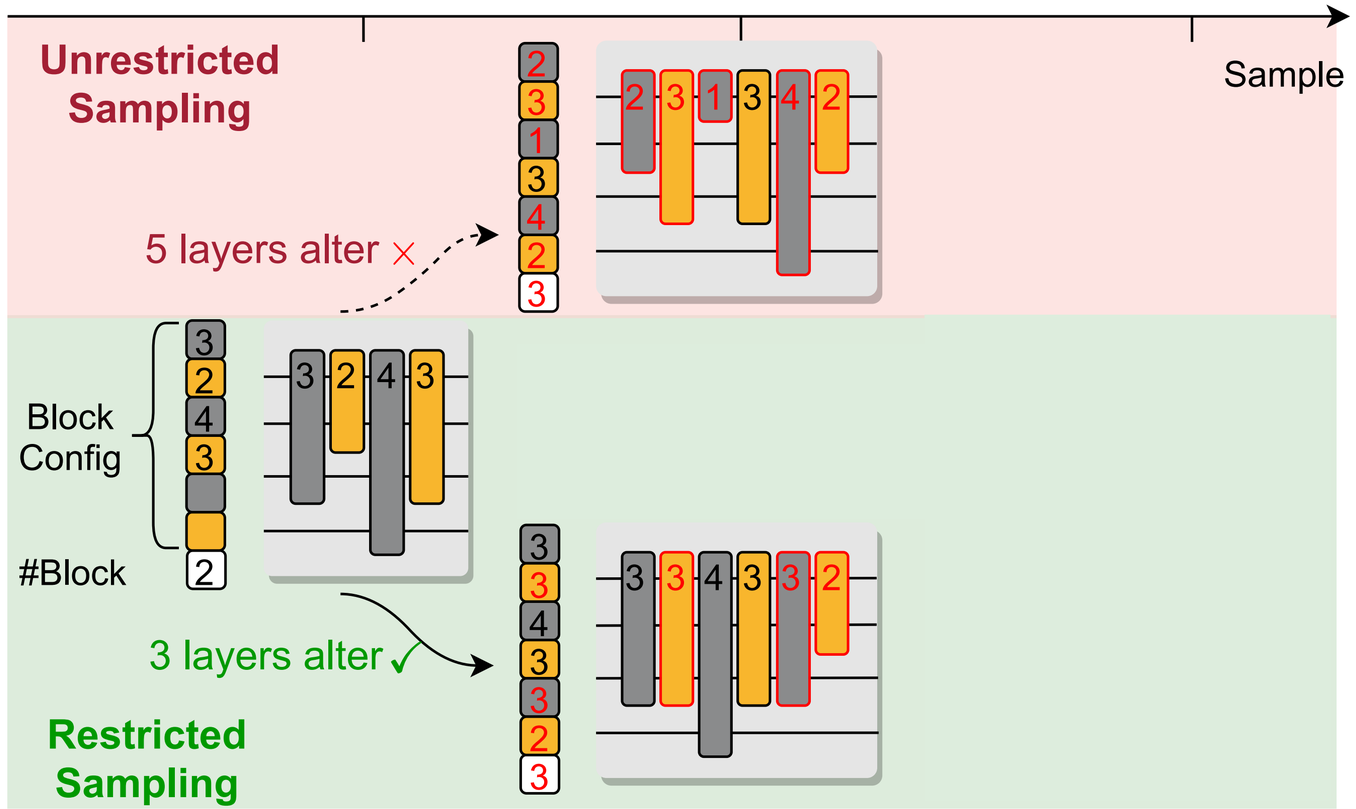
# Restricted Sampling

- Restricted Sampling:
  - Restrict the difference between two consecutively sampled SubCircuits



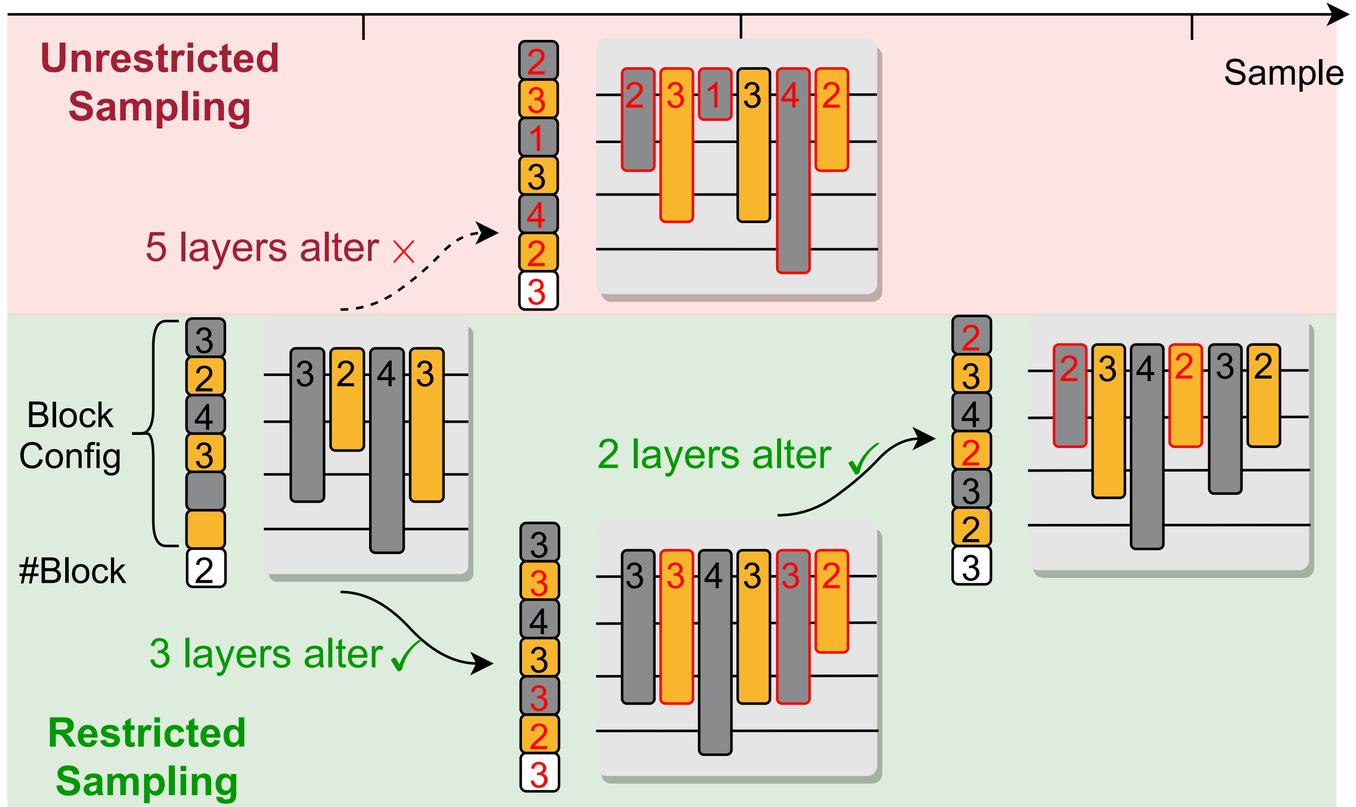
# Restricted Sampling

- Restricted Sampling:
  - Restrict the difference between two consecutively sampled SubCircuits



# Restricted Sampling

- Restricted Sampling:
  - Restrict the difference between two consecutively sampled SubCircuits

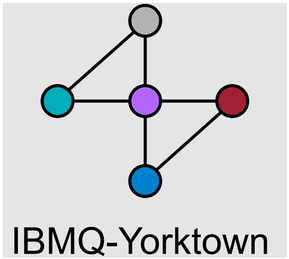


# QuantumNAS

- SuperCircuit Construction and Training
- Noise-Adaptive Evolutionary Co-Search of SubCircuit and Qubit Mapping
- Searched SubCircuit Training
- Iterative Pruning

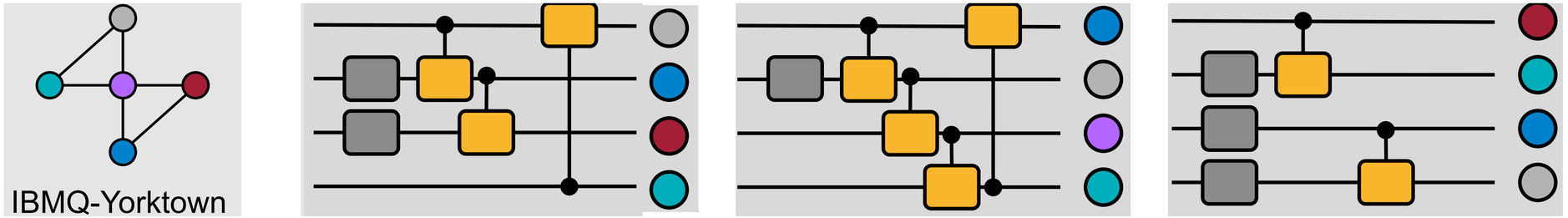
# Noise-Adaptive Evolutionary Co-Search

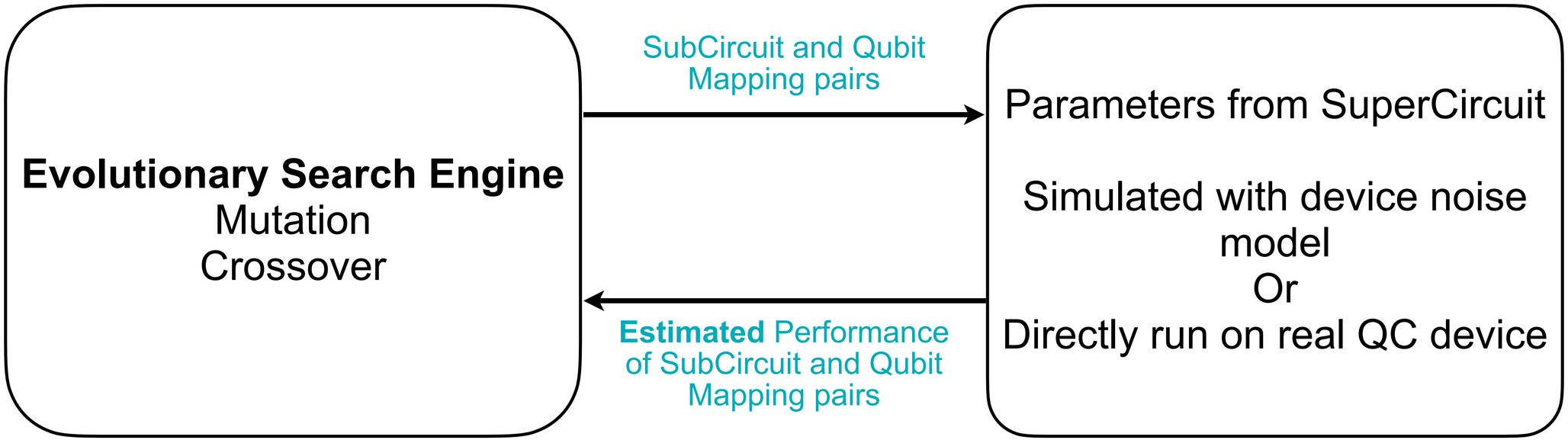
- Search the best SubCircuit and its qubit mapping on target device



# Noise-Adaptive Evolutionary Co-Search

- Search the best SubCircuit and its qubit mapping on target device



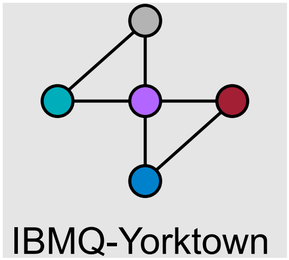



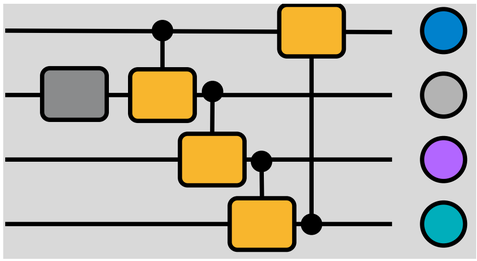
# QuantumNAS

- SuperCircuit Construction and Training
- Noise-Adaptive Evolutionary Co-Search of SubCircuit and Qubit Mapping
- Searched SubCircuit Training
- Iterative Pruning

# Searched SubCircuit Training

- Train the SubCircuit parameters from scratch and get final parameters



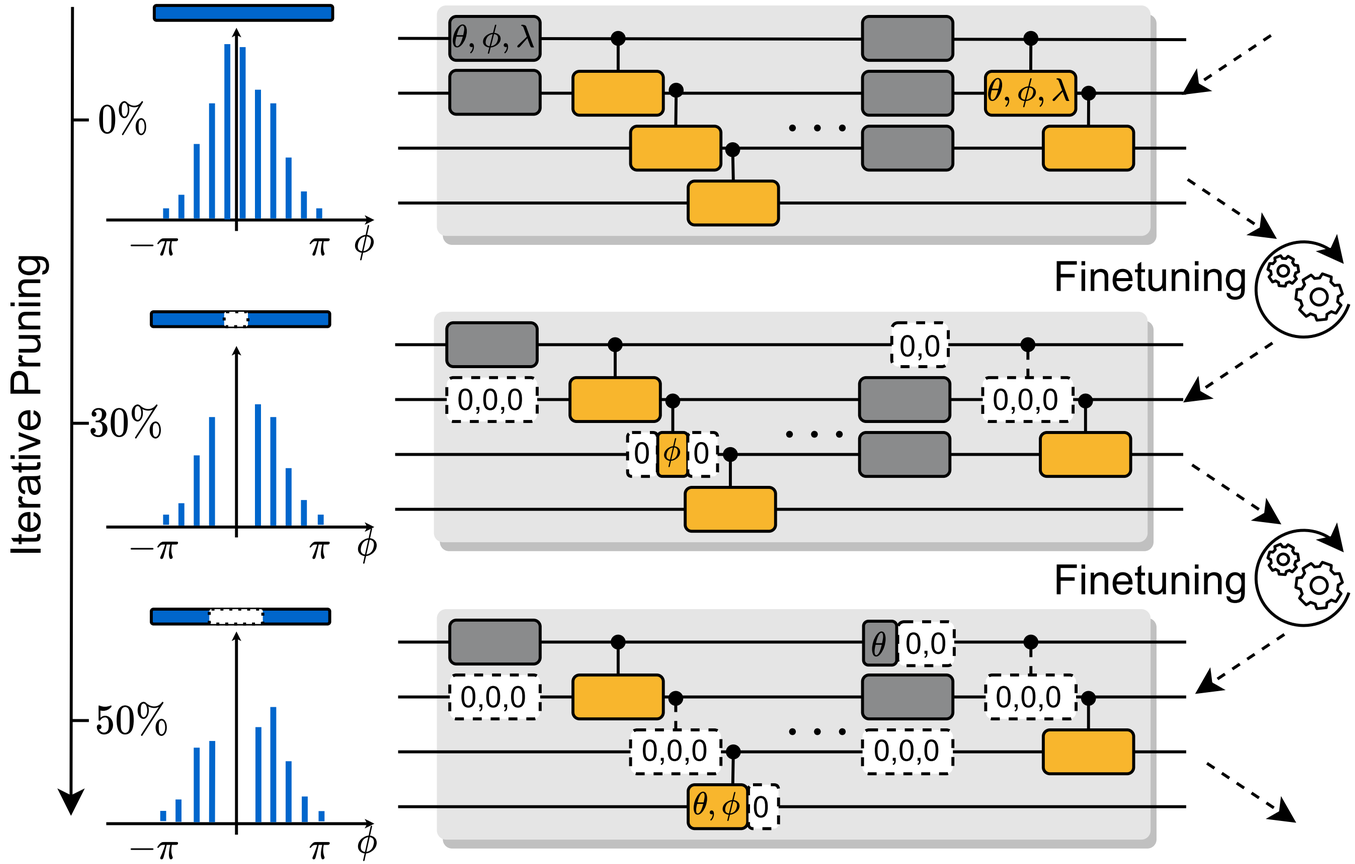



# QuantumNAS

- SuperCircuit Construction and Training
- Noise-Adaptive Evolutionary Co-Search of SubCircuit and Qubit Mapping
- Searched SubCircuit Training
- Iterative Pruning

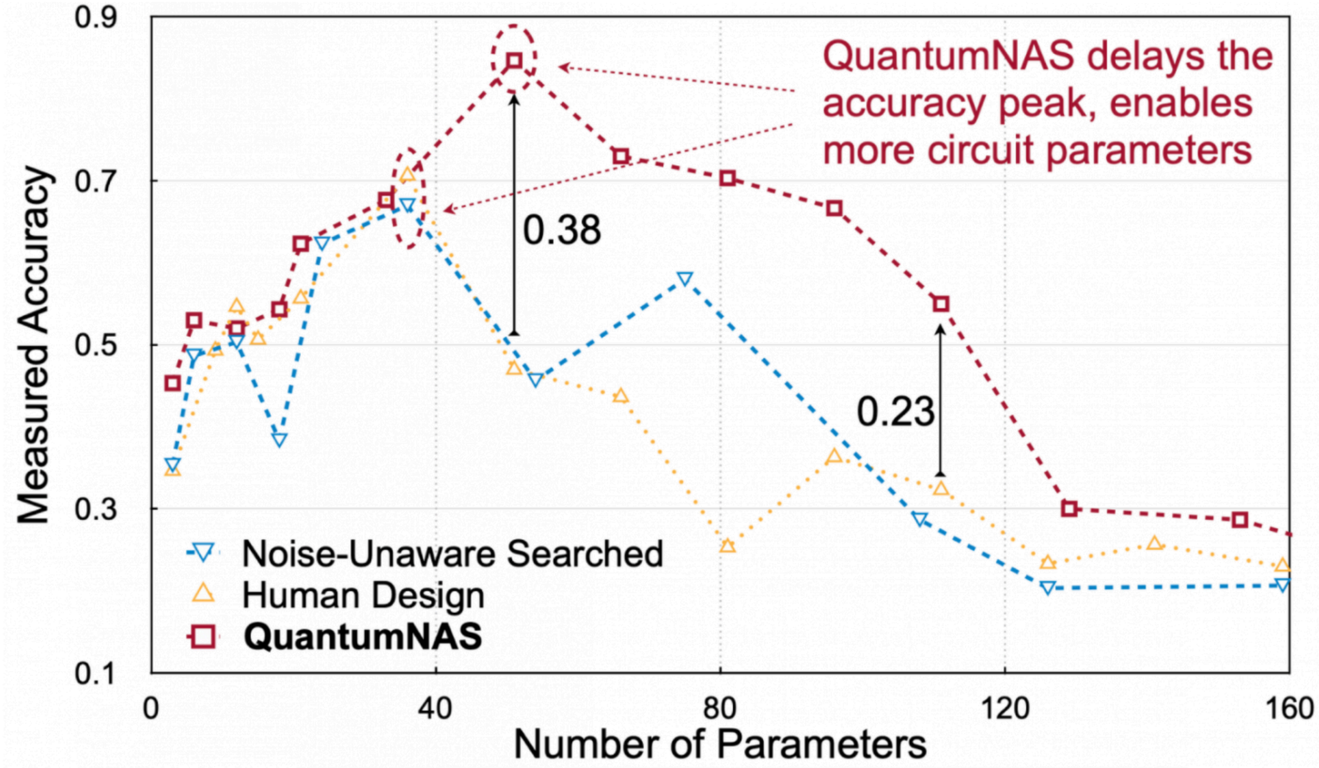
# Iterative Pruning

- Some gates has parameters close to 0
  - Rotation gate with angle close to 0 has small impact on the state
- Iteratively remove small-magnitude gates and fine-tune the remaining parameters



# QML Results

- 4-classification: MNIST-4 U3+CU3 on IBMQ-Yorktown

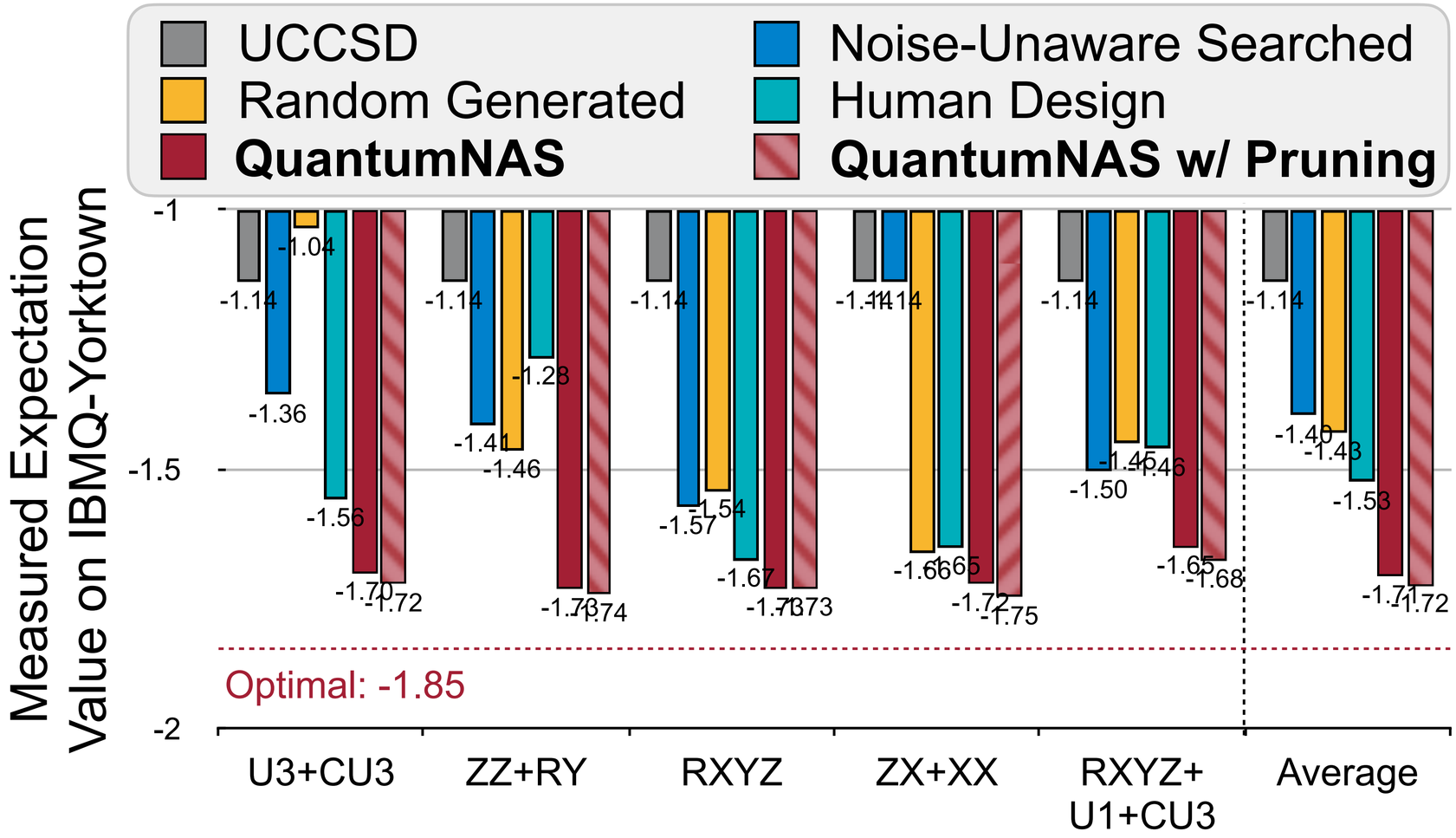



Torch Quantum

MIT HAN LAB

# H2 VQE Results

- H2 in different design spaces on IBMQ-Yorktown




Torch Quantum

MIT HAN LAB

## RobustQNN: Noise-Aware Training for Robust Quantum Neural Networks [DAC 2022]

Hanrui Wang, Yongshan Ding, Jiaqi Gu, Zirui Li, Yujun Lin, David Z. Pan, Frederic T. Chong, Song Han

## On-chip QNN: Towards Efficient On-Chip Training of Quantum Neural Networks [DAC 2022]

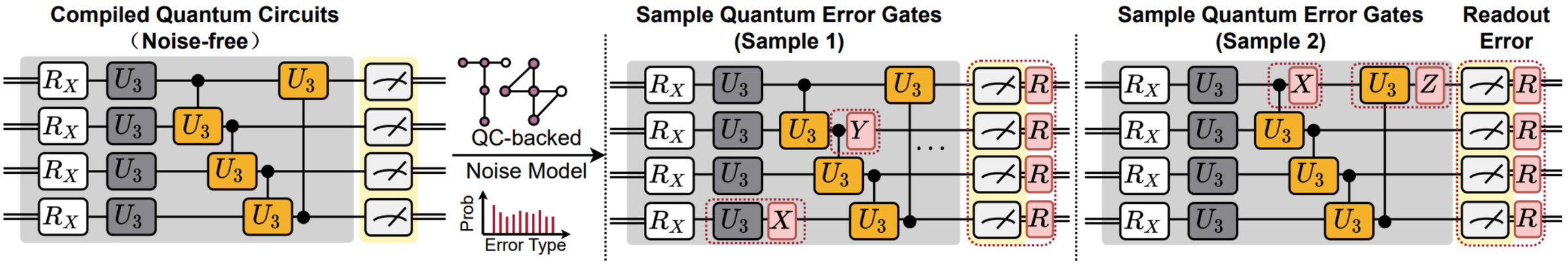
Hanrui Wang*, Zirui Li*, Jiaqi Gu, Yongshan Ding, David Z. Pan, Song Han

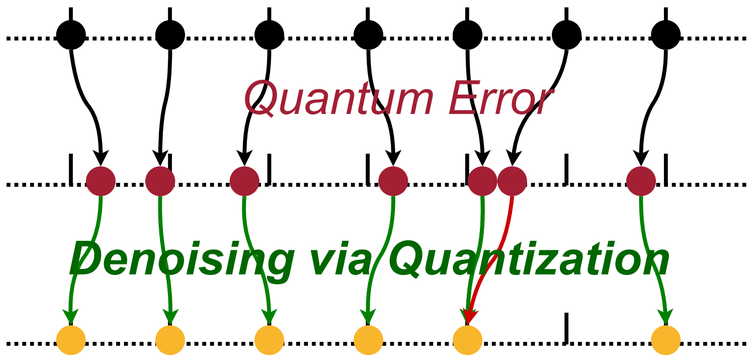
Torch Quantum

MIT HAN LAB

# RobustQNN

- QuantumNAS: find the **circuit architecture** robust to noise
- RobustQNN: further make the **parameter** robust to noise



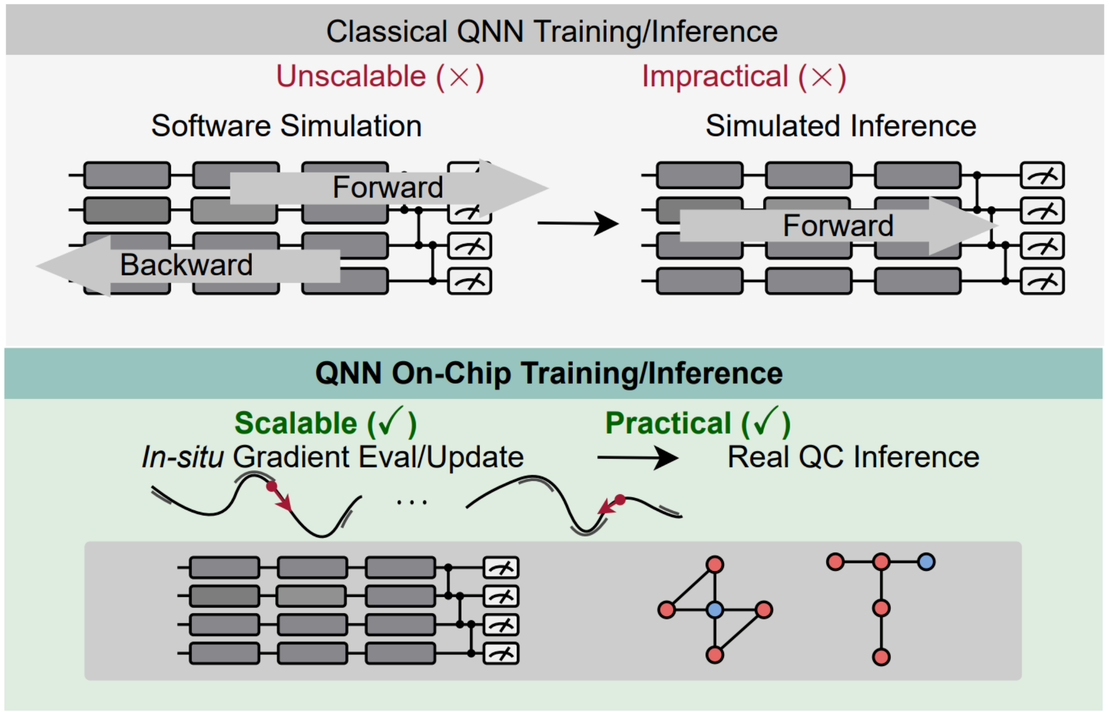



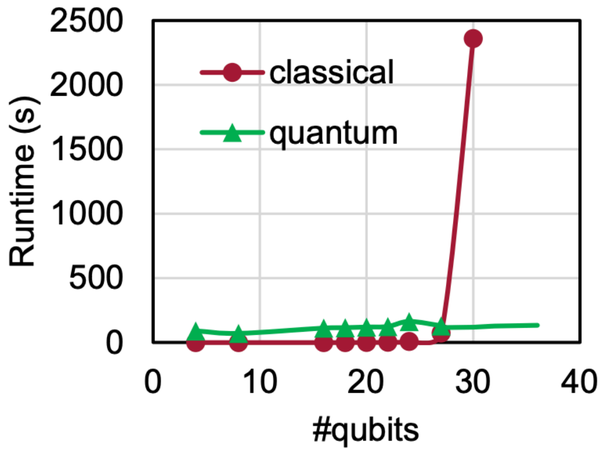

Torch Quantum

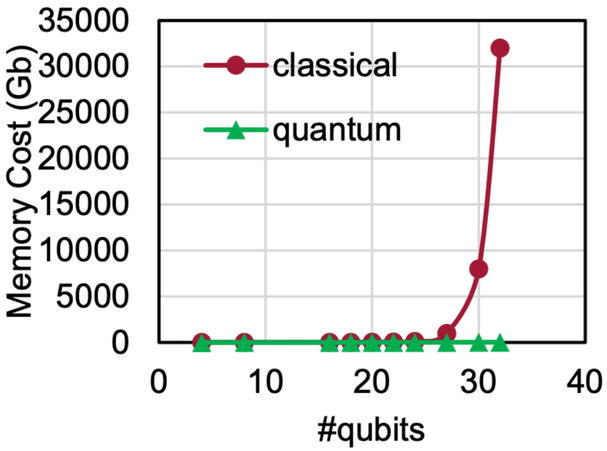
MIT HAN LAB

# On-chip QNN

- How to make the training process scalable, in order to achieve potential quantum advantage?



# Torch Quantum

# TorchQuantum

- Features
  - Easy construction of **parameterized quantum circuits** such as Quantum Neural Networks in PyTorch
  - Support **batch mode inference and training** on GPU/CPU, supports highly-parallelized training
  - Support **easy deployment** on real quantum devices such as IBMQ
  - Provide tutorials, videos and example projects of QML and using ML to optimize quantum computer system problems

MIT HAN LAB

# PyTorch Implementations

- Statevector

```
_state = torch.zeros(2 ** self.n_wires, dtype=C_DTYPE)
_state[0] = 1 + 0j
```

- Quantum Gates

```
'cnot': torch.tensor([[1, 0, 0, 0],
                      [0, 1, 0, 0],
                      [0, 0, 0, 1],
                      [0, 0, 1, 0]], dtype=C_DTYPE),
```

MIT HAN LAB

# PyTorch Implementations

- Quantum Gates

```
def crx_matrix(params):
    theta = params.type(C_DTYPE)
    co = torch.cos(theta / 2)
    jsi = 1j * torch.sin(-theta / 2)

    matrix = torch.tensor([[1, 0, 0, 0],
                           [0, 1, 0, 0],
                           [0, 0, 0, 0],
                           [0, 0, 0, 0]], dtype=C_DTYPE, device=params.device
                          ).unsqueeze(0).repeat(co.shape[0], 1, 1)
    matrix[:, 2, 2] = co[:, 0]
    matrix[:, 2, 3] = jsi[:, 0]
    matrix[:, 3, 2] = jsi[:, 0]
    matrix[:, 3, 3] = co[:, 0]

    return matrix.squeeze(0)
```

- Matrix-vector multiplication: torch.einsum and torch.bmm

MIT HAN LAB

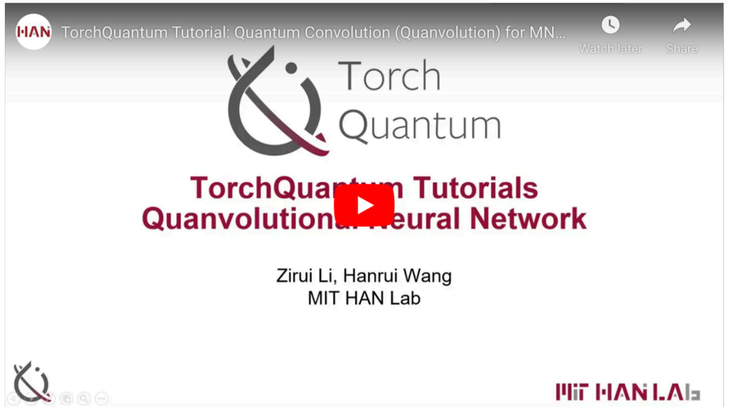
# Examples and tutorials

## Quantum Convolution (Quanvolution) for MNIST image classification

Authors: Zirui Li, Hanrui Wang

Use Colab to run this example: Open in Colab

See this tutorial video for detailed explanations:

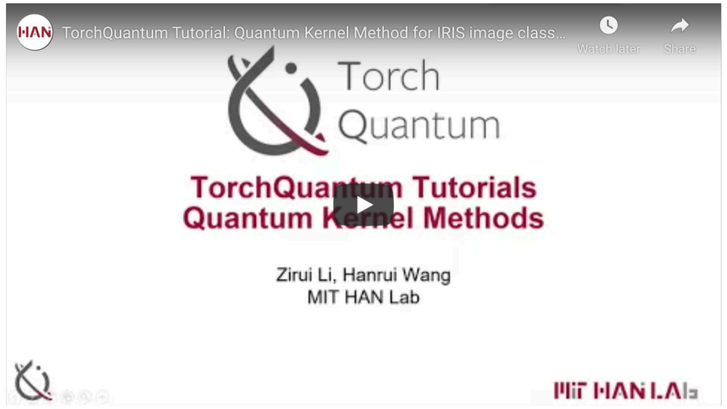



## Quantum Kernel Method

Authors: Zirui Li, Hanrui Wang

Use Colab to run this example: Open in Colab

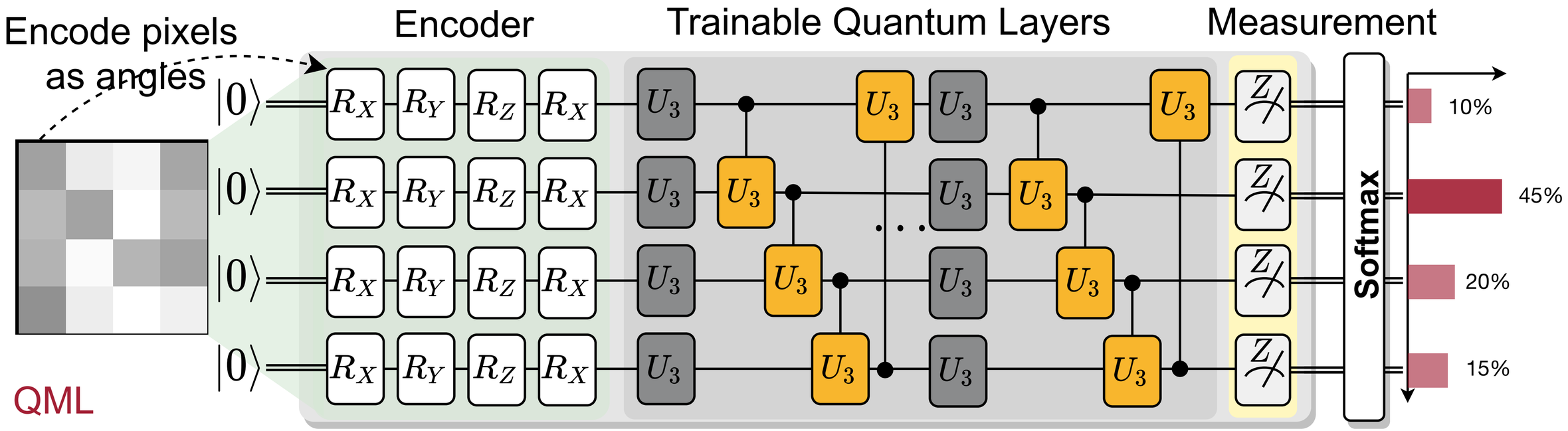
See this tutorial video for detailed explanations:



# MNIST Example

# MNIST Example

- Parameterized quantum circuits
  - Quantum Neural Networks




- Softmax $\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$

Torch Quantum

MIT HAN LAB

# MNIST Example

Initialize a quantum device

Specify encoder gates

Specify trainable gates

```python
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)

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

Torch Quantum

MIT HAN LAB

# MNIST Example

Reset statevector

Encode classical pixels

Apply the trainable gates

Apply some non-trainable gates

Measure to get classical values

```
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 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])

  # 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])

  # 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]])

  # 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
```

Torch Quantum

# MNIST Example

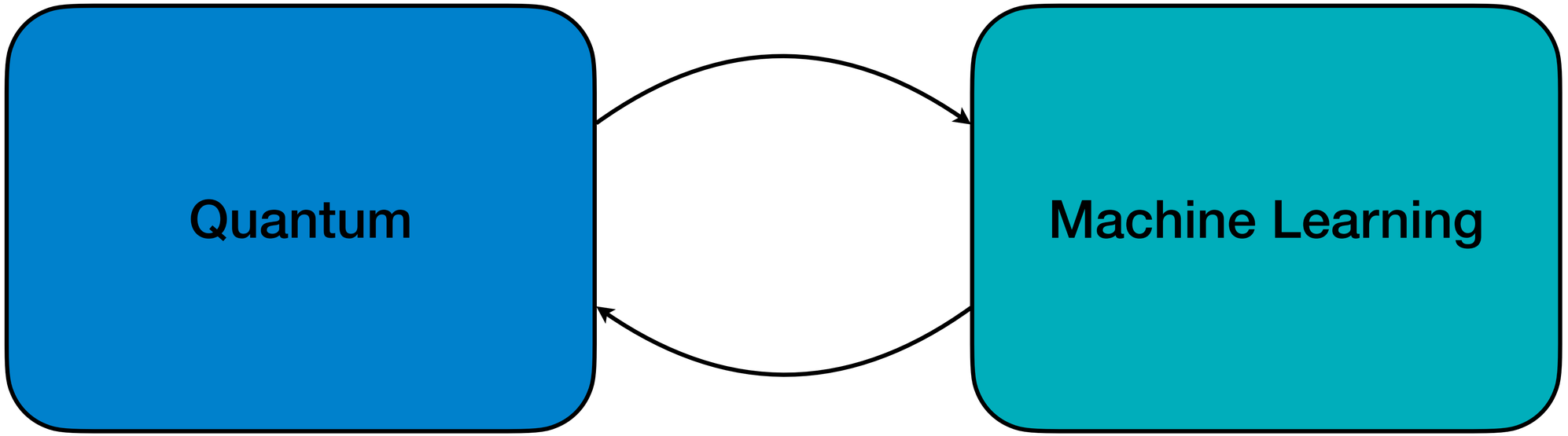
Initialize a QiskitProcessor

The ‘forward’ will be run on real QC

```
# then try to run on REAL QC
backend_name = 'ibmqx2'
print(f"\nTest on Real Quantum Computer {backend_name}")
processor_real_qc = QiskitProcessor(use_real_qc=True,
                                    backend_name=backend_name)
model.set_qiskit_processor(processor_real_qc)
valid_test(dataflow, 'test', model, device, qiskit=True)
```

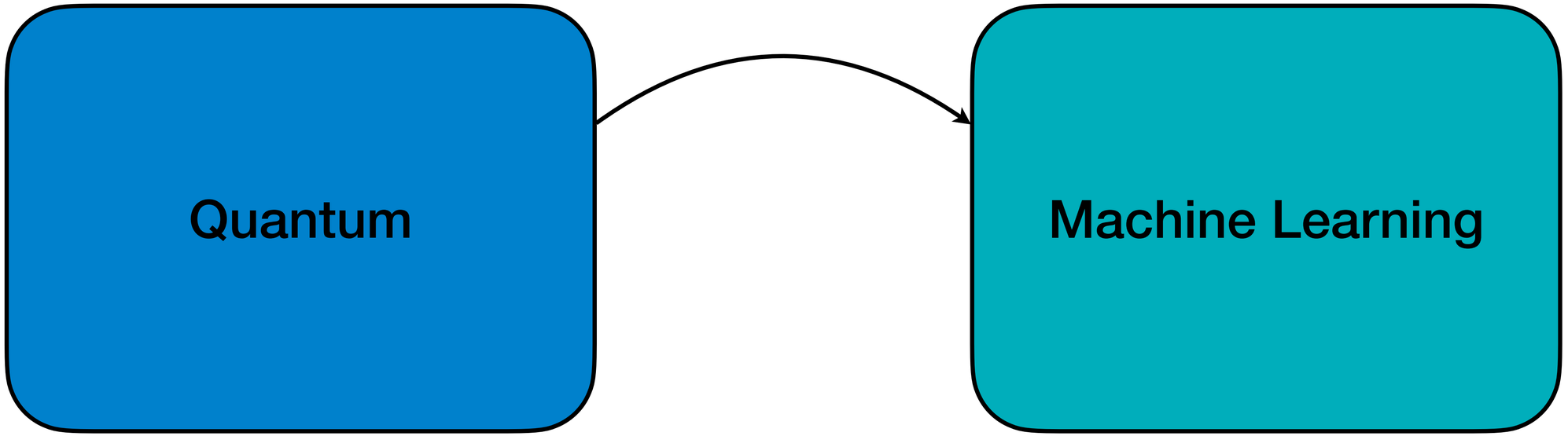
# TorchQuantum

- TorchQuantum — A library for interdisciplinary research of quantum computing and machine learning




# TorchQuantum

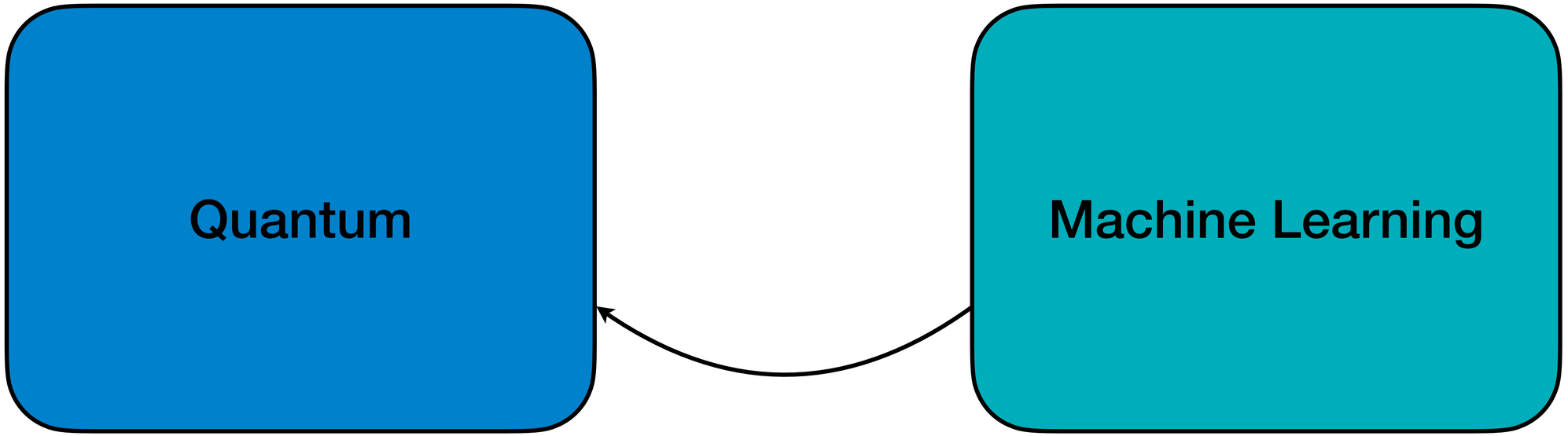
- TorchQuantum — A library for interdisciplinary research of quantum computing and machine learning




- Quantum for Machine learning
  - Quantum neural networks
  - Quantum kernel methods
  - …

# TorchQuantum

- TorchQuantum — A library for interdisciplinary research of quantum computing and machine learning




- Machine Learning for Quantum
  - ML for quantum compilation (qubit mapping, unitary synthesis)
  - ML for optimal control
  - …

# Take home messages

- Parameterized quantum circuit are useful on NISQ devices (VQE, QNN, QAOA)
  - QuantumNAS for circuit **architecture** search
  - RobustQNN for robust **parameter** training
  - On-chip QNN for **scalable** training of large PQC, on real machine
- TorchQuantum: **library** for Quantum ML and ML for Quantum

# Thank you!

# Q&A

Torch Quantum

https://github.com/mit-han-lab/torchquantum

qmlsys.mit.edu

MIT HAN LAB