qmlsys.mit.edu

QuantumNAS: Noise-Aware Search for Robust Quantum Circuits and the TorchQuantum Library

Hanrui Wang
MIT HAN Lab

[HPCA'22] QuantumNAS: Noise-adaptive search for robust quantum circuits

[DAC'22] QuantumNAT: Quantum Noise-Aware Training with Noise Injection, Quantization and Normalization

[DAC'22] QOC: Quantum On-Chip Training with Parameter Shift and Gradient Pruning

Torch Quantum

2022/06/28 EQuS

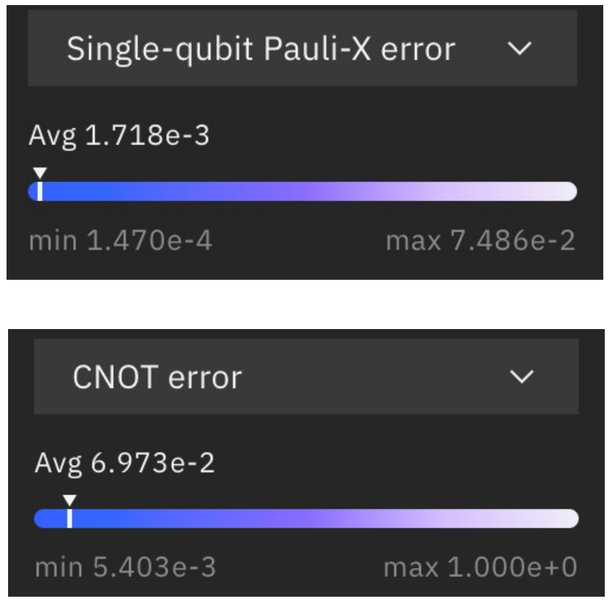
MIT HAN LAB

Outline

- Background
- QuantumNAS
- TorchQuantum Library
- Conclusion

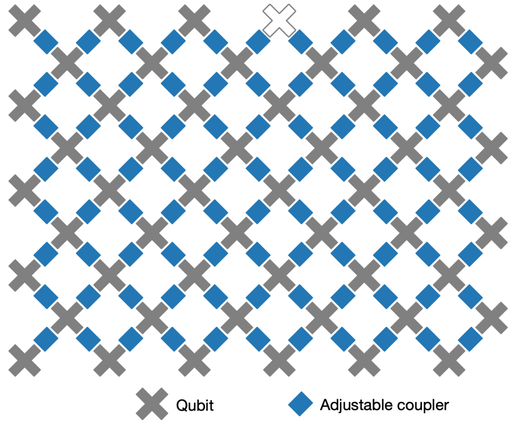
NISQ Era

- Noisy Intermediate-Scale Quantum (NISQ)
  - **Noisy**: qubits are sensitive to environment; quantum gates are unreliable
  - **Limited number** of qubits: tens to hundreds of qubits



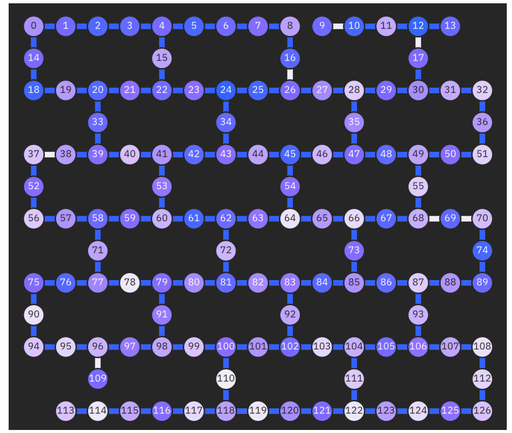
Gate Error Rate

https://quantum-computing.ibm.com/



Google Sycamore

https://www.nature.com/articles/s41586-019-1666-5

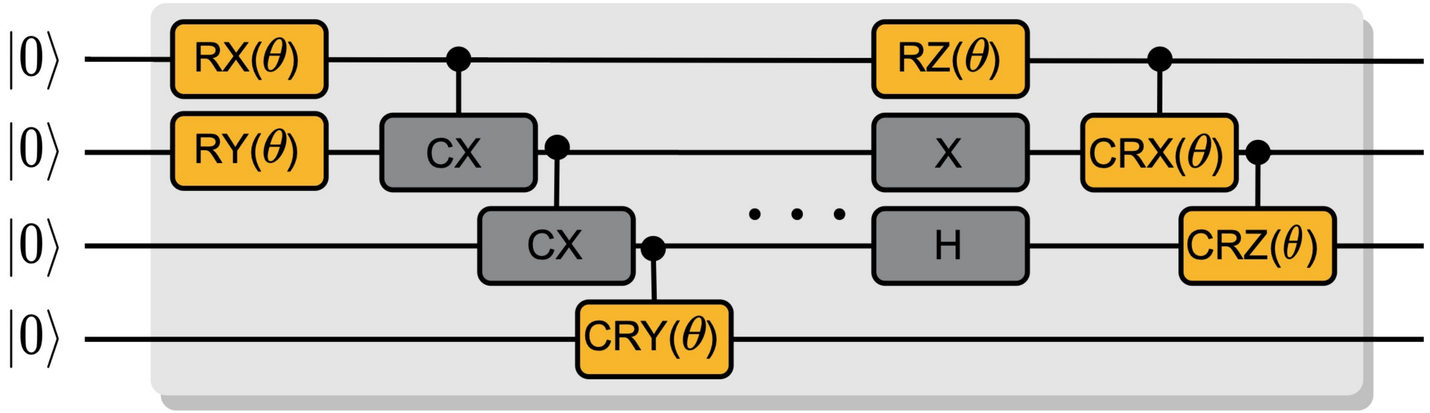


IBM Washington

https://quantum-computing.ibm.com/

Parameterized Quantum Circuits

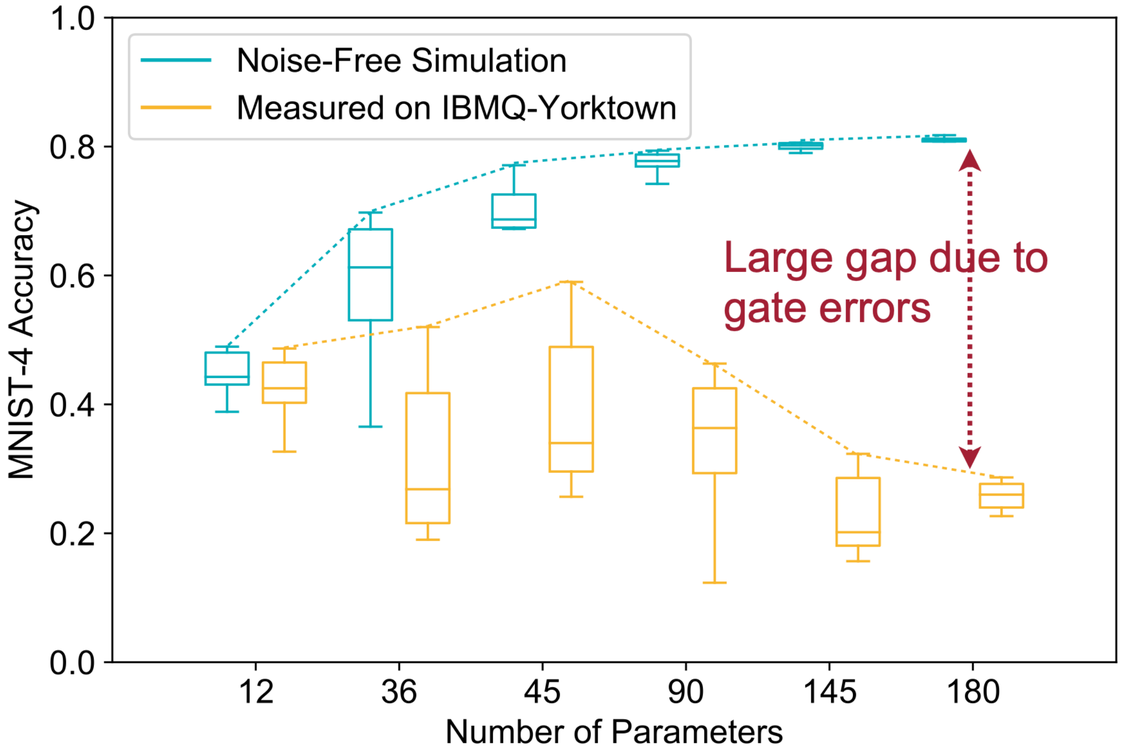
- Parameterized Quantum Circuits (PQC)
- Quantum circuit with fixed gates and parameterized gates



- PQCs are commonly used in **hybrid classical-quantum models** and show promises to achieve quantum advantage
  - Variational Quantum Eigensolver (VQE)
  - Quantum Neural Networks (QNN)
  - Quantum Approximate Optimization Algorithm (QAOA)

Challenges of PQC — Noise

- Noise **degrades** PQC reliability
- More parameters increase the noise-free accuracy but degrade the measured accuracy
- Therefore, circuit architecture is critical



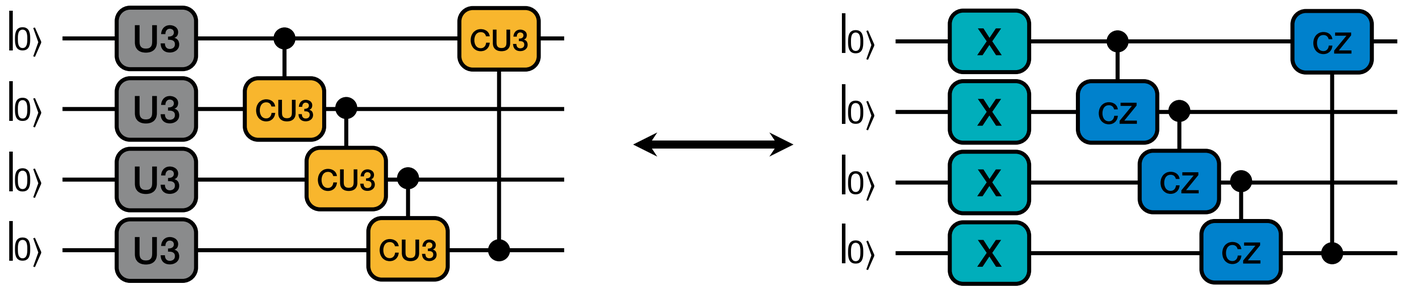
Torch Quantum

qmlsys.mit.edu

MIT HAN LAB

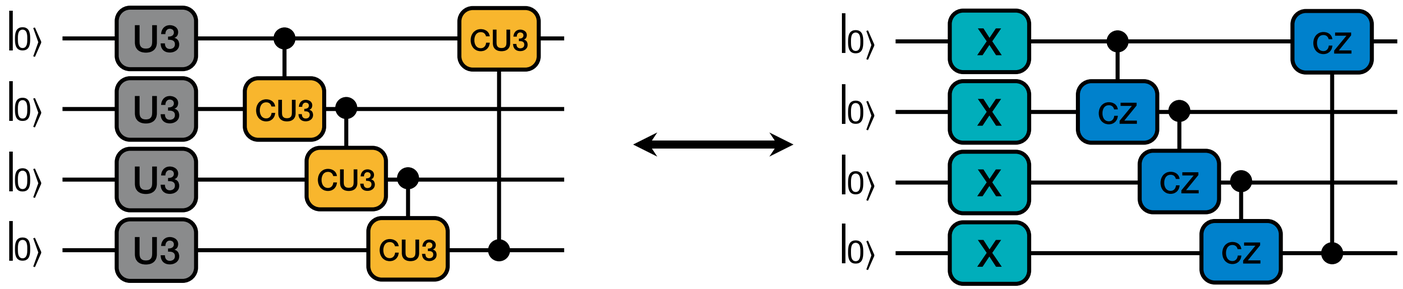
Challenges of PQC — Large Design Space

- Large design space for circuit architecture
  - Type of gates

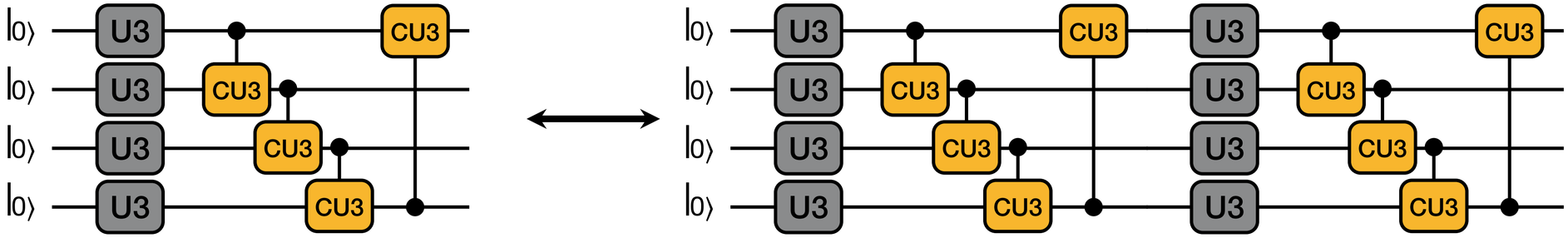


Challenges of PQC — Large Design Space

- Large design space for circuit architecture
  - Type of gates

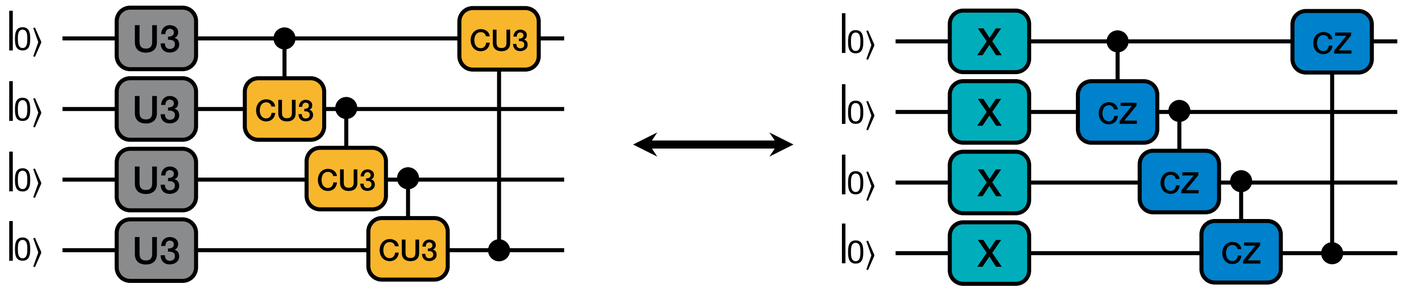


  - Number of gates

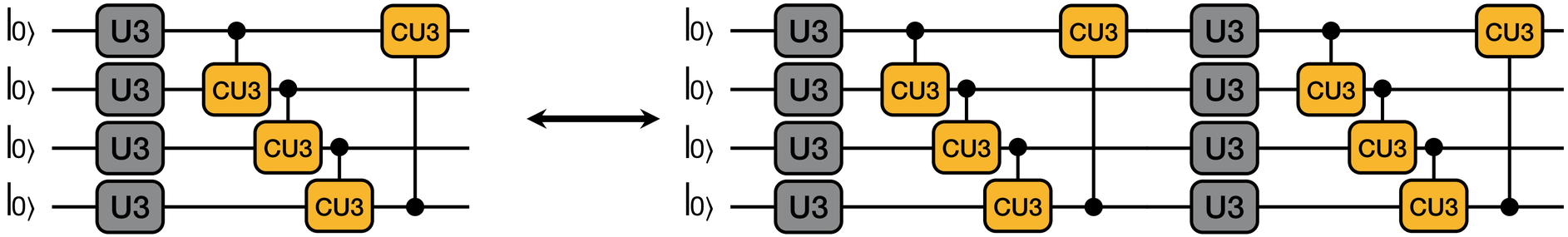


Challenges of PQC — Large Design Space

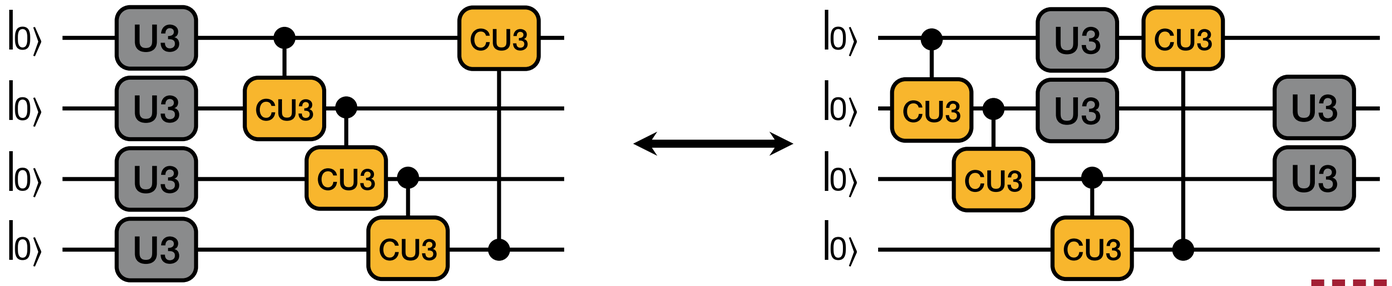
- Large design space for circuit architecture
  - Type of gates



  - Number of gates



  - Position of gates



Goal of QuantumNAS

Automatically & efficiently search for noise-robust quantum circuit

Train one “SuperCircuit”, providing parameters to many “SubCircuits”

Solve the challenge of large design space

(1) Quantum noise feedback in the search loop
(2) Co-search the circuit architecture and qubit mapping

Solve the challenge of large quantum noise

QuantumNAS

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

QuantumNAS

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

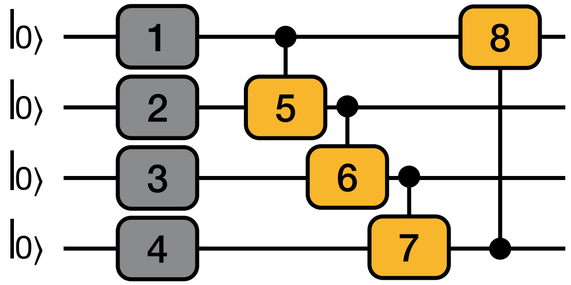
SuperCircuit & SubCircuit

- Firstly construct a design space. For example, a design space of maximum 4 U3 in the first layer and 4 CU3 gates in the second layer

SuperCircuit & SubCircuit

- Firstly construct a design space. For example, a design space of maximum 4 U3 in the first layer and 4 CU3 gates in the second layer

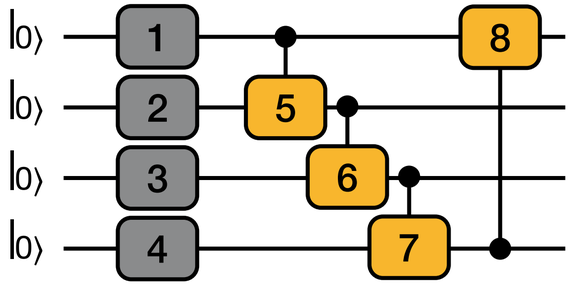
- SuperCircuit: the circuit with the **largest** number of gates in the design space
  - Example: SuperCircuit in U3+CU3 space



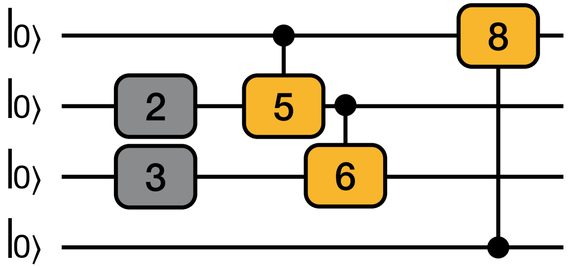
SuperCircuit & SubCircuit

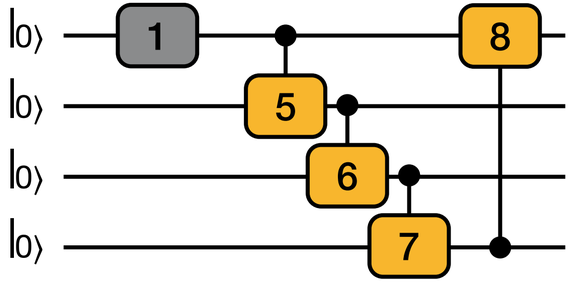
- Firstly construct a design space. For example, a design space of maximum 4 U3 in the first layer and 4 CU3 gates in the second layer

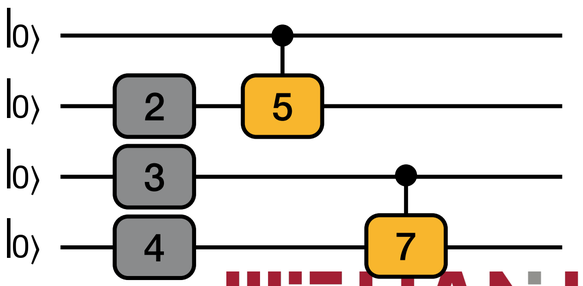
- SuperCircuit: the circuit with the **largest** number of gates in the design space
  - Example: SuperCircuit in U3+CU3 space



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







Torch Quantum

qmlsys.mit.edu

MIT HAN LAB

SuperCircuit Construction

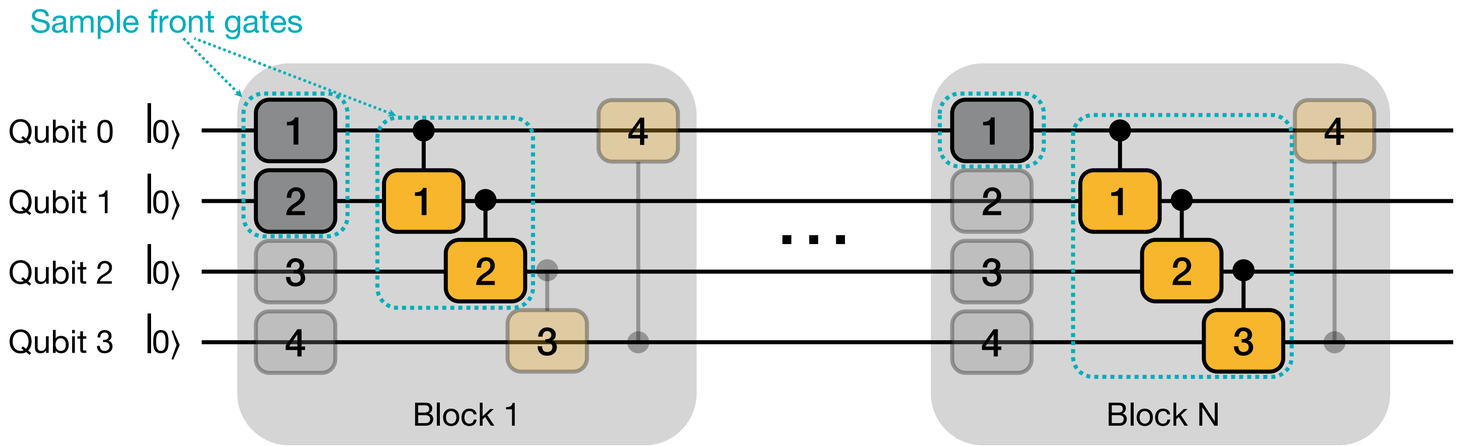
- Why use a SuperCircuit?
  - Enables **efficient** search of architecture candidates without training each
  - SubCircuit inherits parameters from SuperCircuit
  - With **inherited** parameters, we find some good SubCircuits, we find that they are **also good SubCircuits** with parameters **trained from-scratch** individually

SuperCircuit Training

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

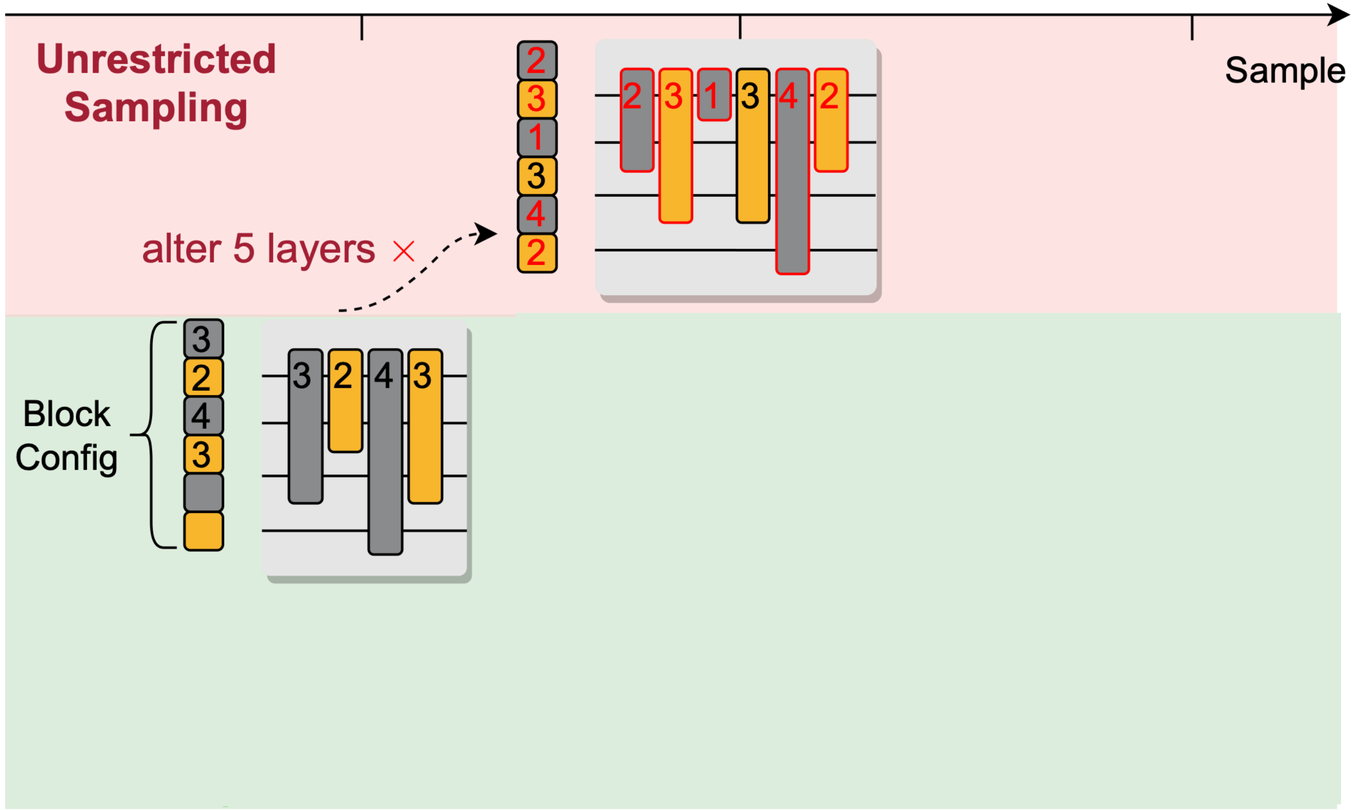
Front Sampling

- 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



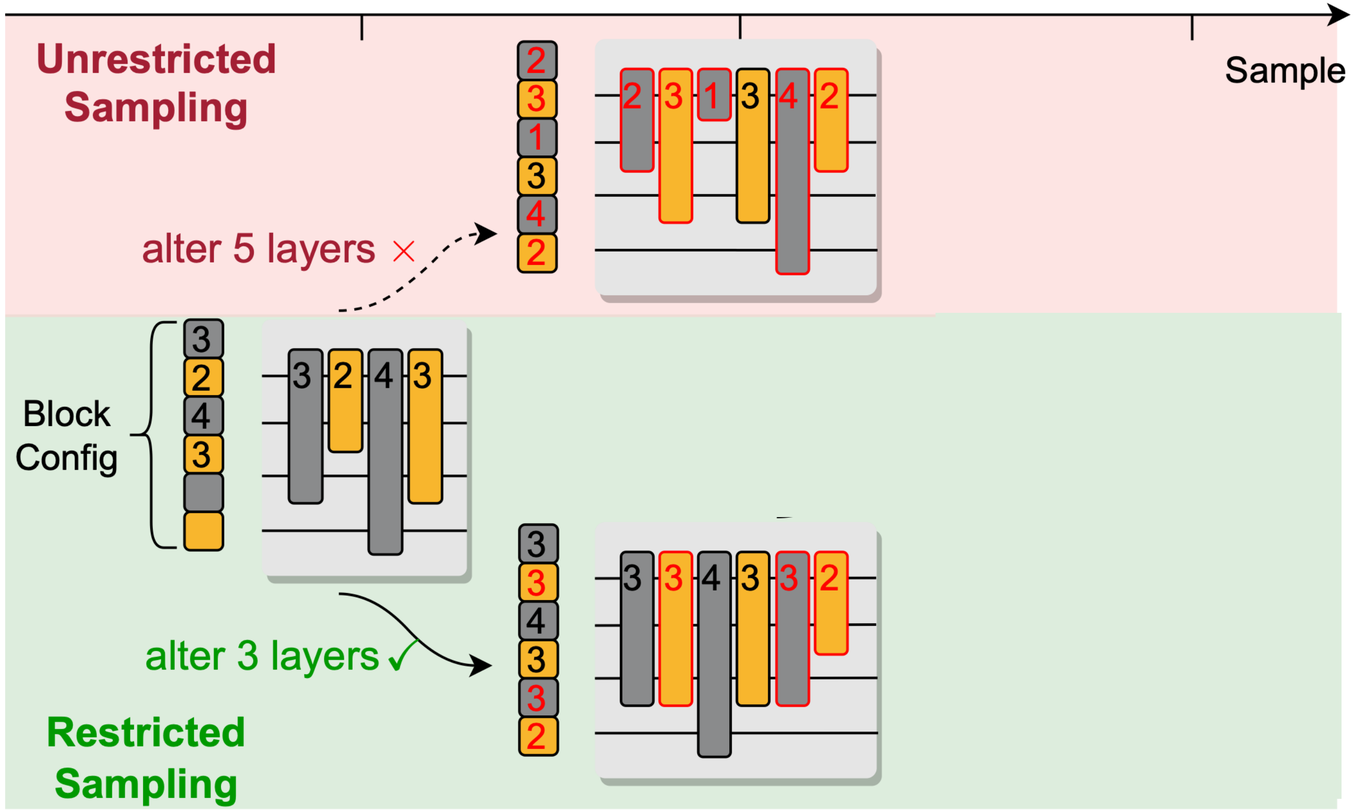
Restricted Sampling

- Restricted Sampling:
  - Restrict the difference between SubCircuits of two consecutive steps
  - For example: restrict to at most 4 different layers



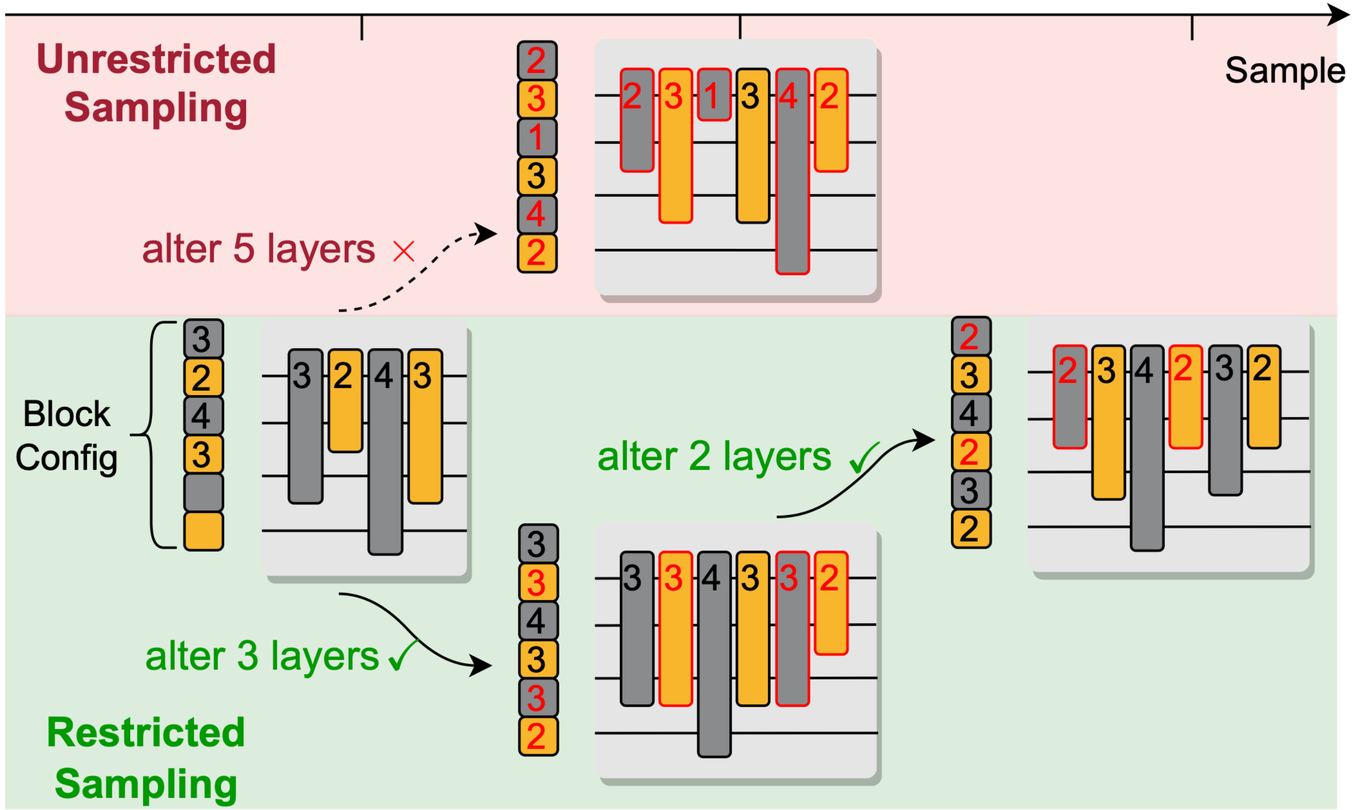
Restricted Sampling

- Restricted Sampling:
  - Restrict the difference between SubCircuits of two consecutive steps
  - For example: restrict to at most 4 different layers



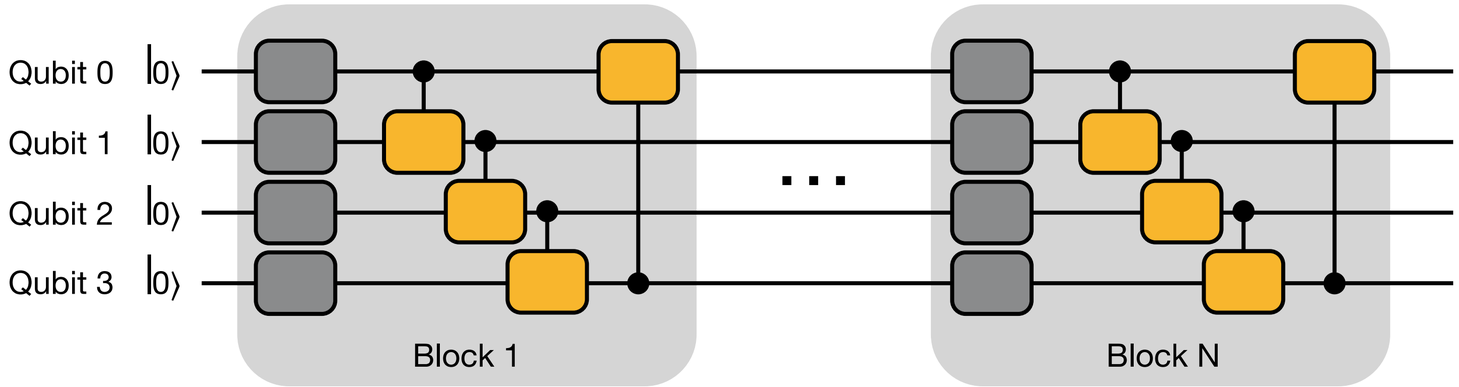
Restricted Sampling

- Restricted Sampling:
  - Restrict the difference between SubCircuits of two consecutive steps
  - For example: restrict to at most 4 different layers



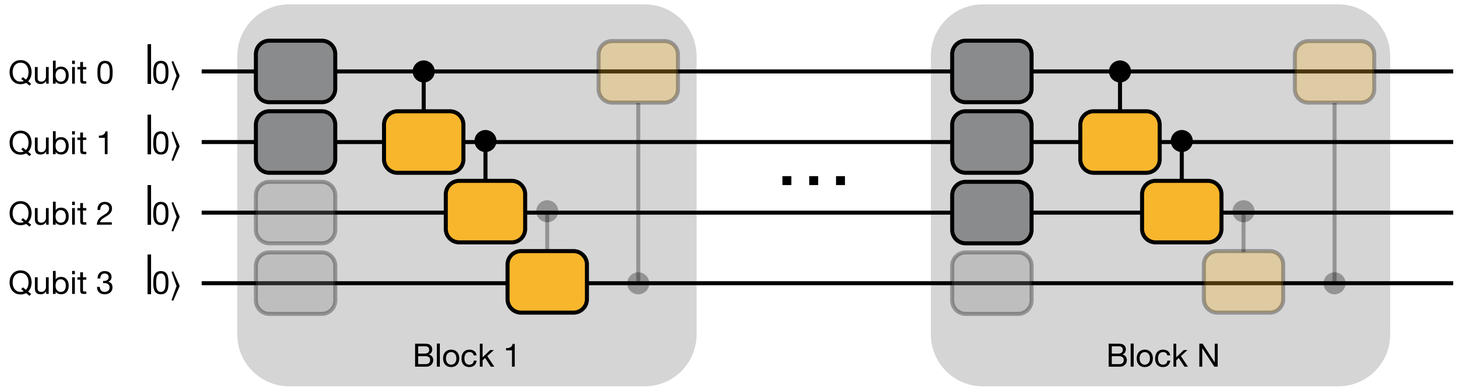
Train SuperCircuit for Multiple Steps

- In one SuperCircuit Training step: Sample and Train



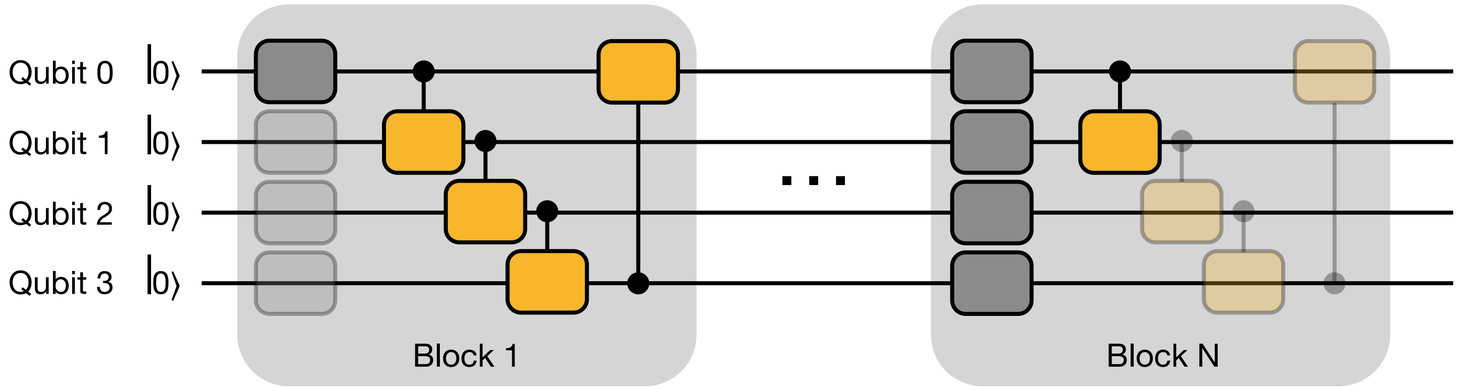
Train SuperCircuit for Multiple Steps

- In one SuperCircuit Training step: Sample and Train



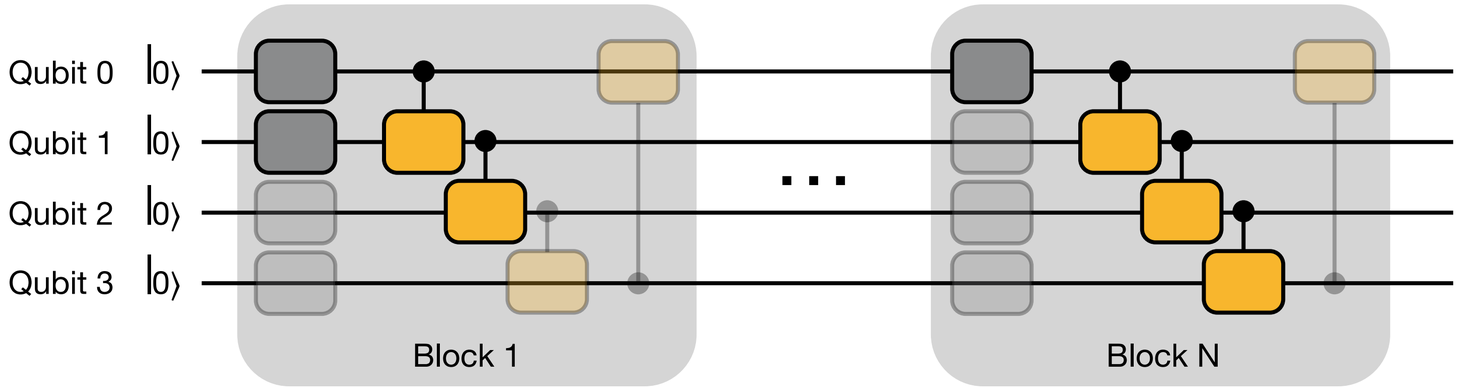
Train SuperCircuit for Multiple Steps

- In one SuperCircuit Training step: Sample and Train



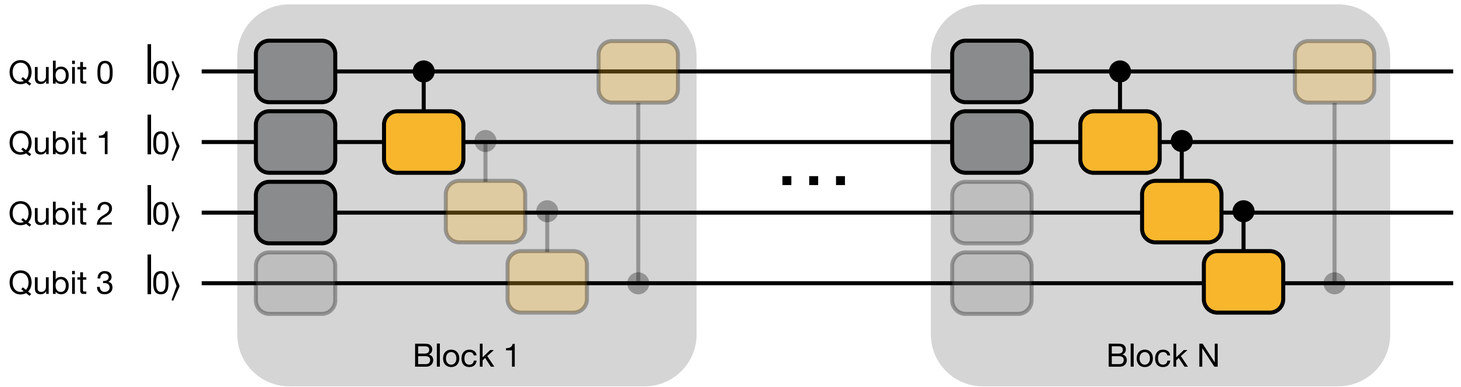
Train SuperCircuit for Multiple Steps

- In one SuperCircuit Training step: Sample and Train



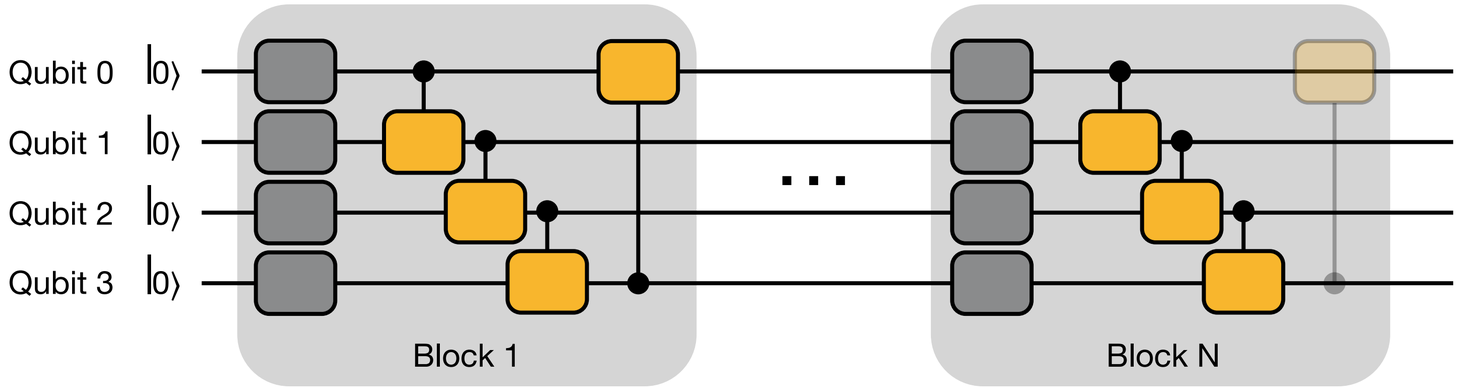
Train SuperCircuit for Multiple Steps

- In one SuperCircuit Training step: Sample and Train



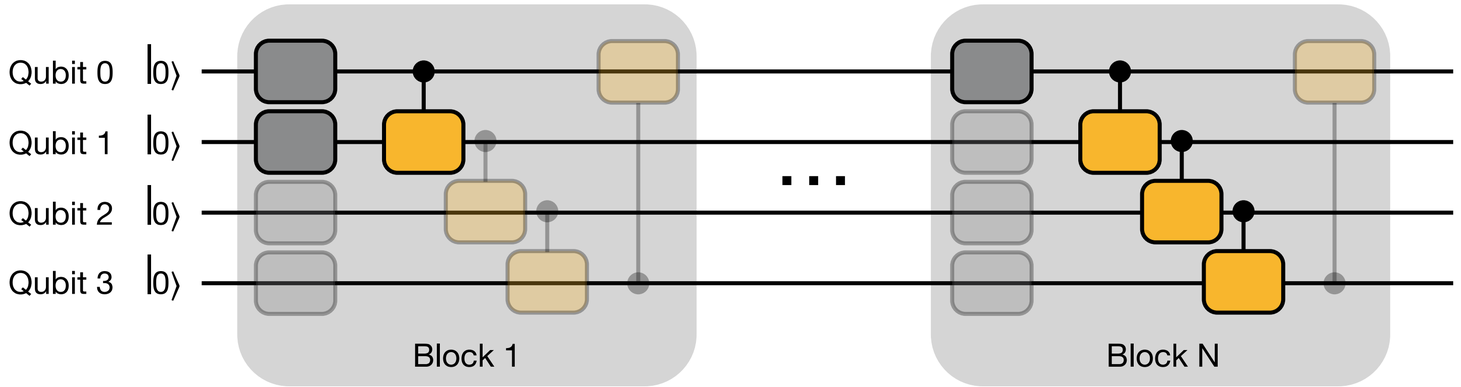
Train SuperCircuit for Multiple Steps

- In one SuperCircuit Training step: Sample and Train



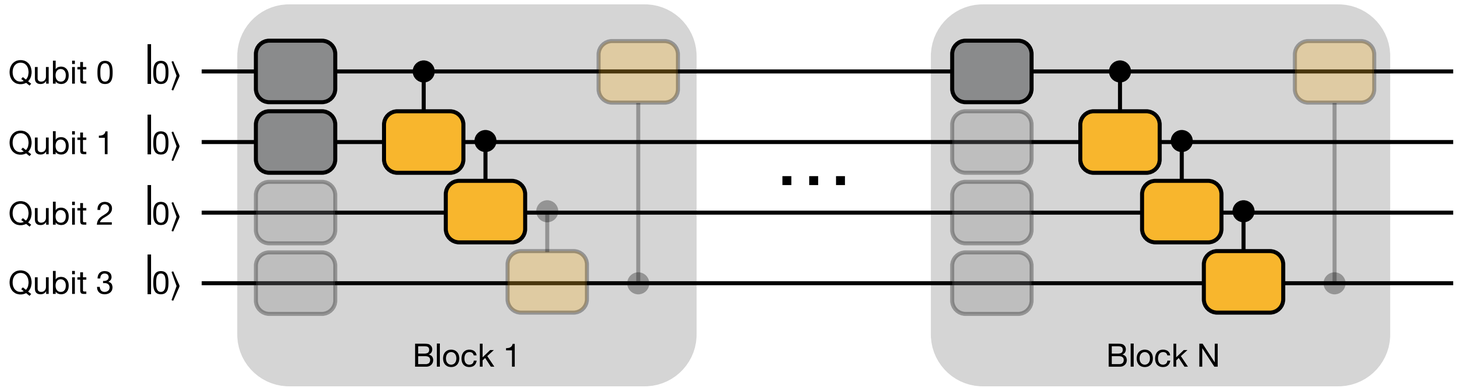
Train SuperCircuit for Multiple Steps

- In one SuperCircuit Training step: Sample and Train



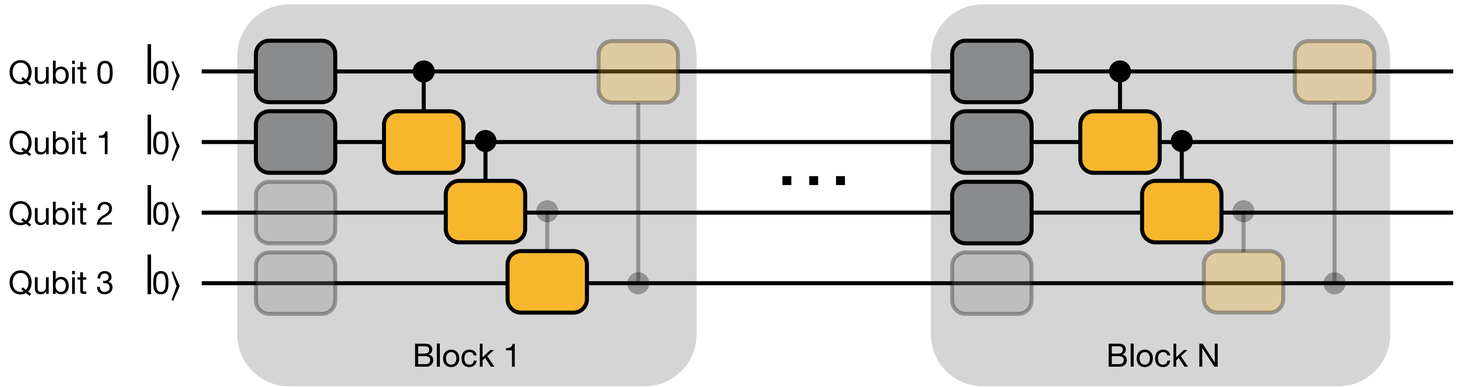
Train SuperCircuit for Multiple Steps

- In one SuperCircuit Training step: Sample and Train



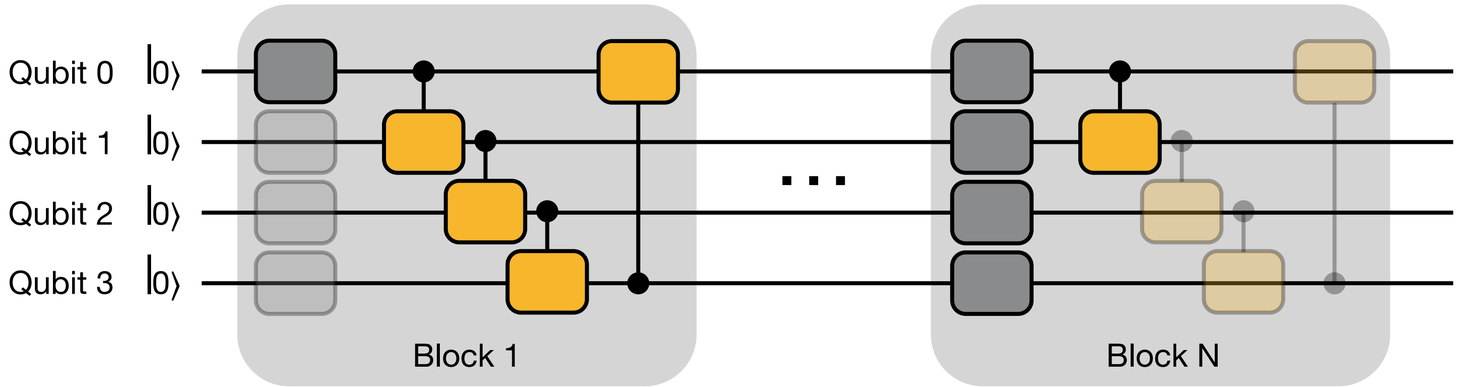
Train SuperCircuit for Multiple Steps

- In one SuperCircuit Training step: Sample and Train



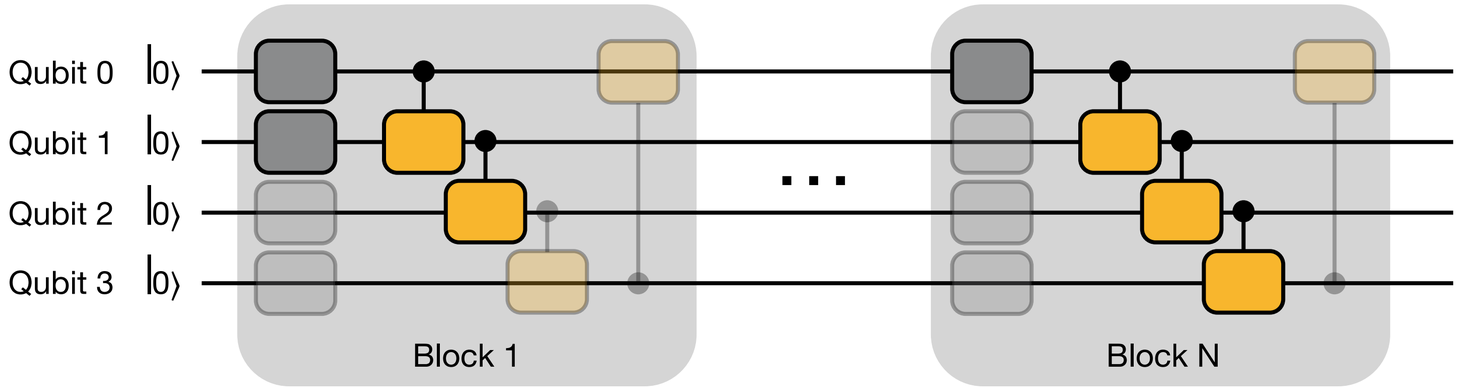
Train SuperCircuit for Multiple Steps

- In one SuperCircuit Training step: Sample and Train



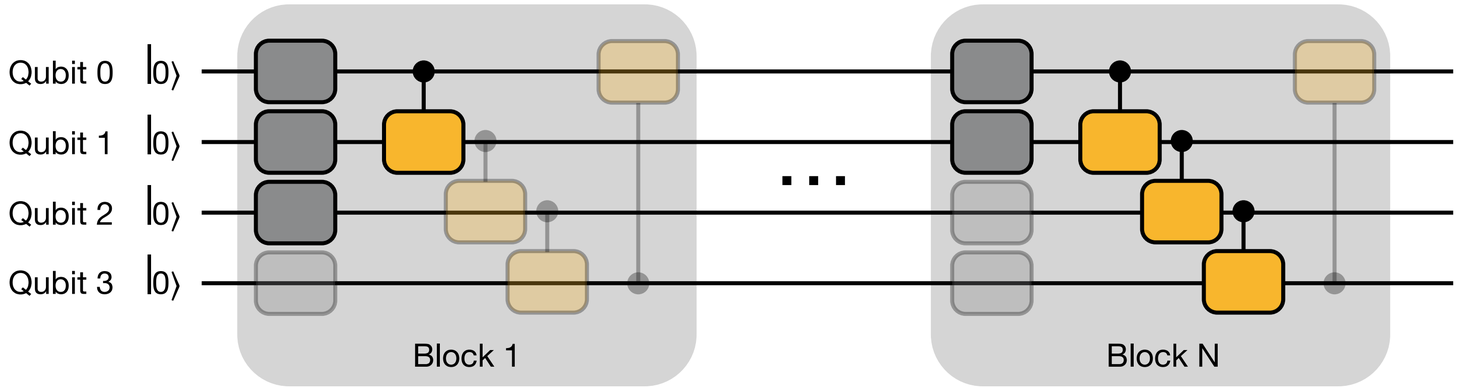
Train SuperCircuit for Multiple Steps

- In one SuperCircuit Training step: Sample and Train



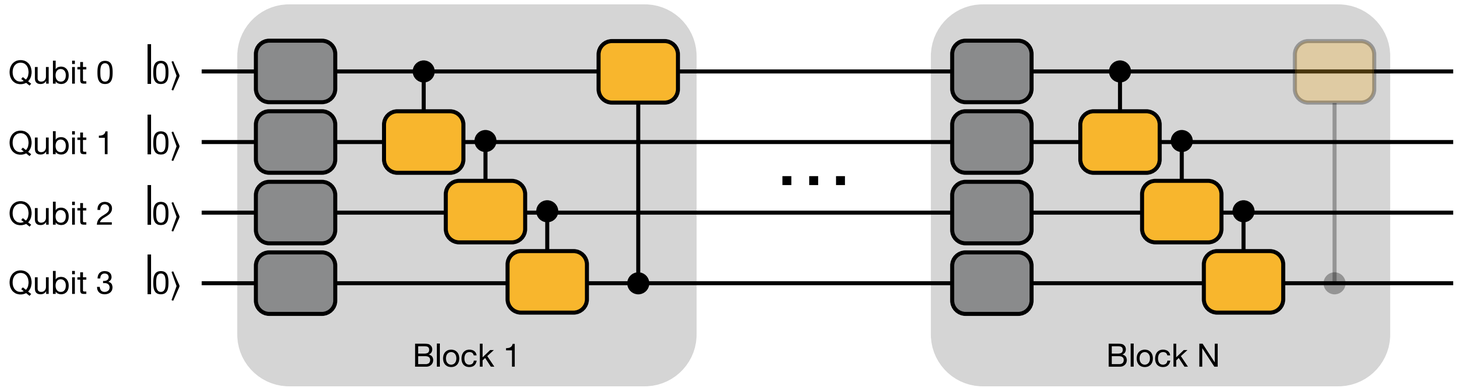
Train SuperCircuit for Multiple Steps

- In one SuperCircuit Training step: Sample and Train



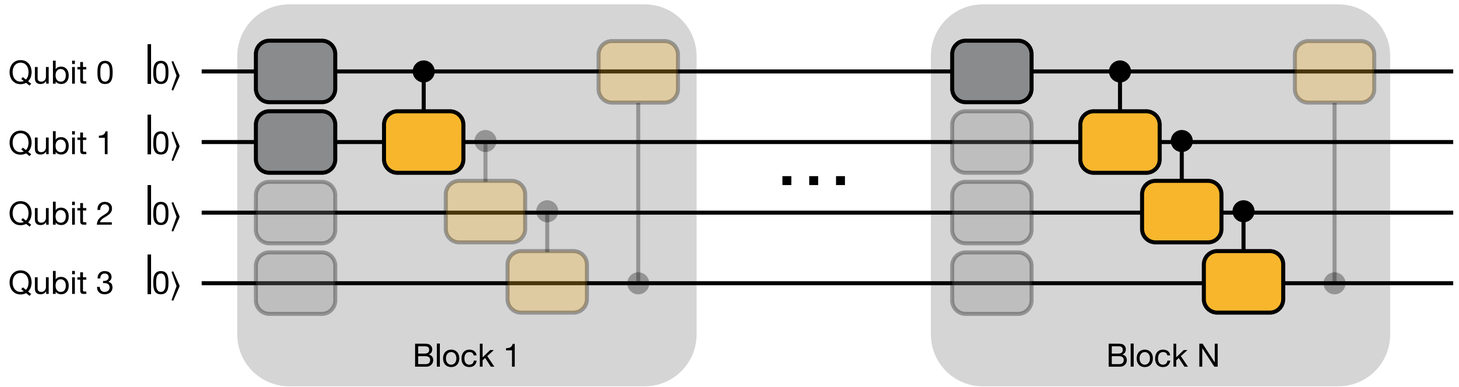
Train SuperCircuit for Multiple Steps

- In one SuperCircuit Training step: Sample and Train



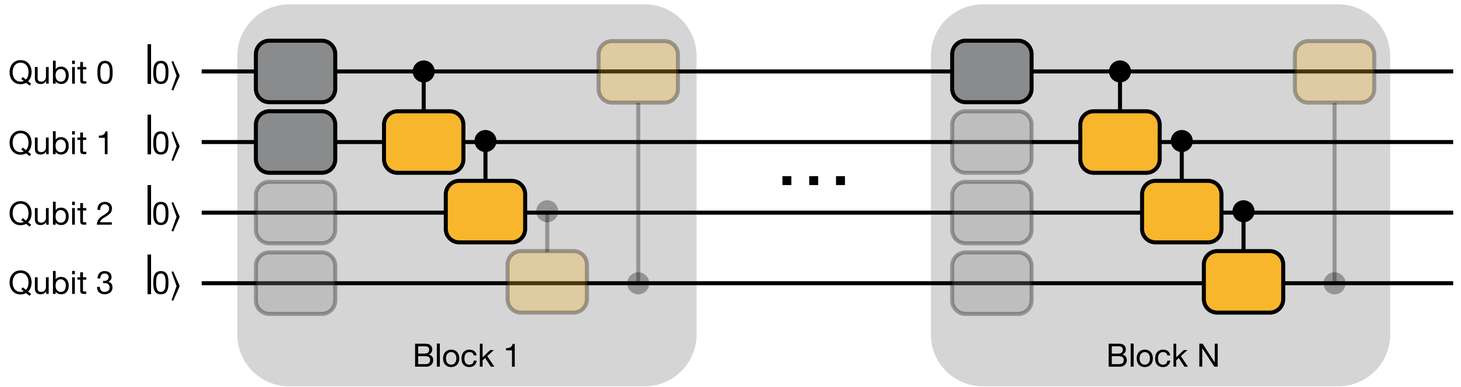
Train SuperCircuit for Multiple Steps

- In one SuperCircuit Training step: Sample and Train

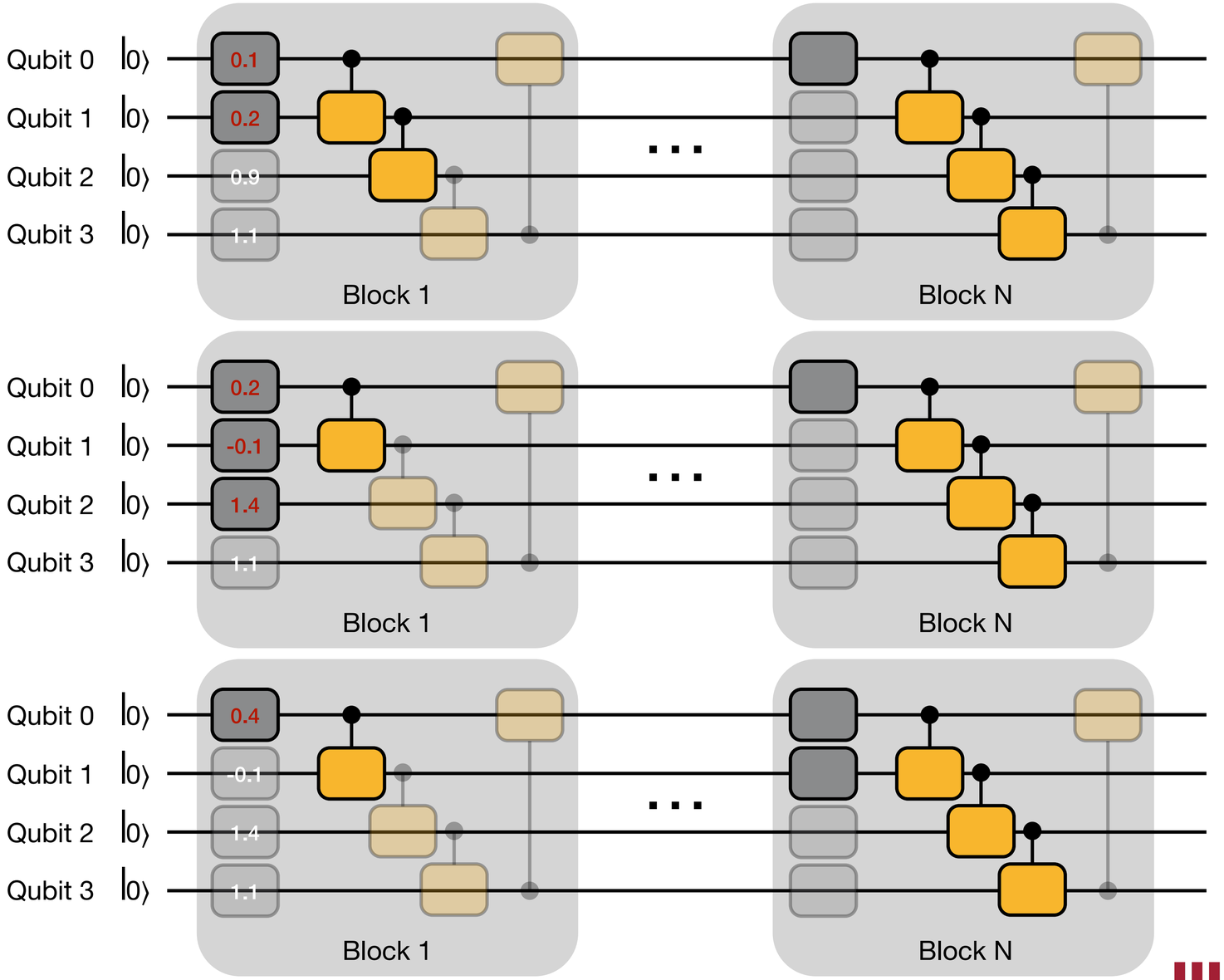


Train SuperCircuit for Multiple Steps

- In one SuperCircuit Training step: Sample and Train



Train SuperCircuit for Multiple Steps



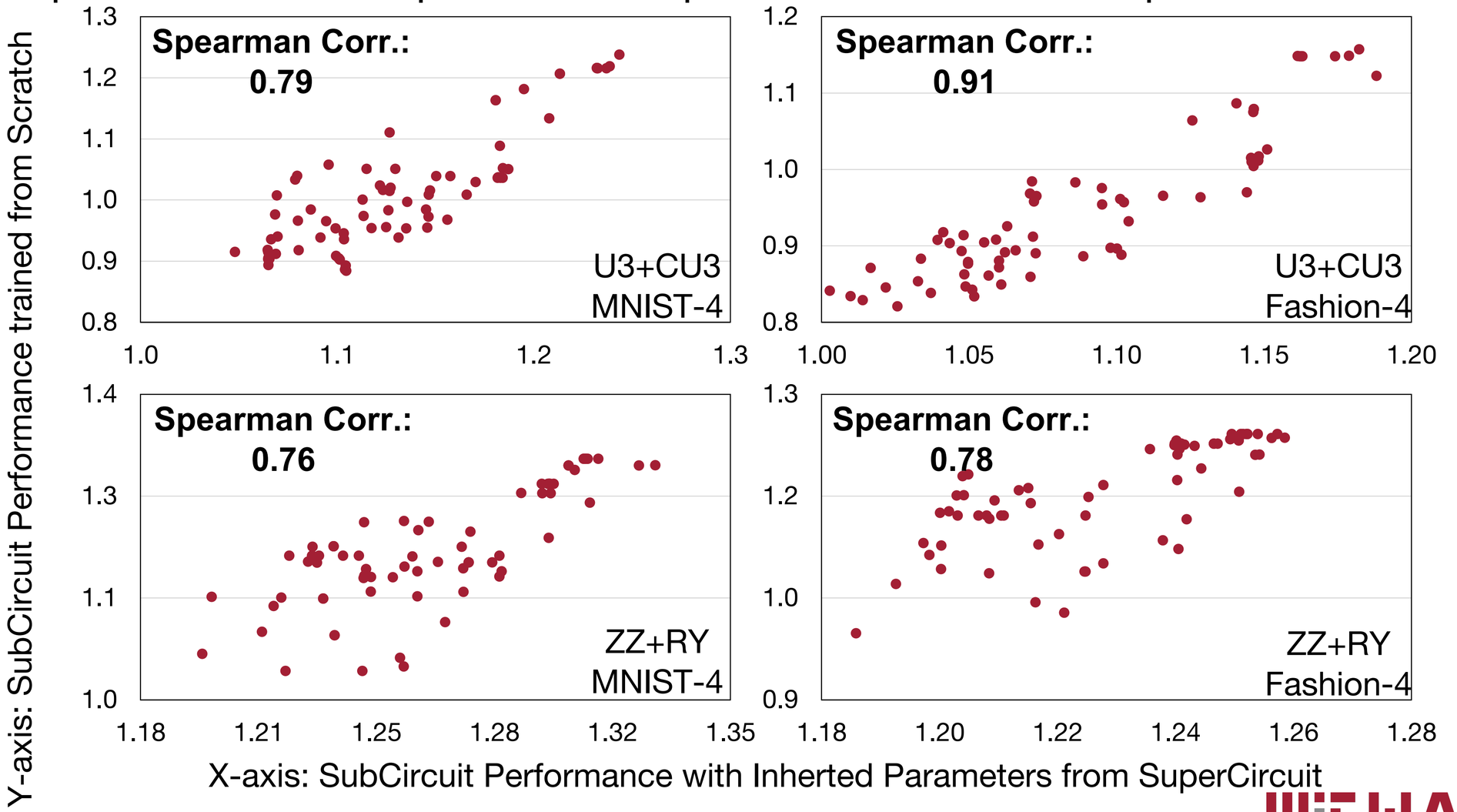
Torch Quantum

qmlsys.mit.edu

MIT HAN LAB

How Reliable is the SuperCircuit?

- Inherited parameters from SuperCircuit can provide accurate relative performance



Torch Quantum

qmlsys.mit.edu

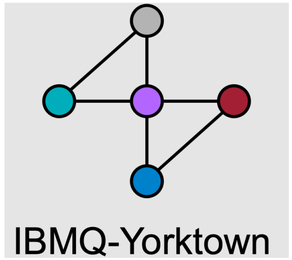
MIT HAN LAB

QuantumNAS

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

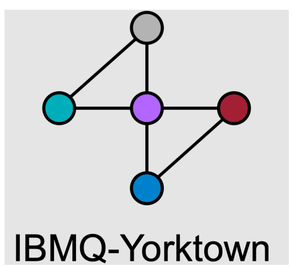
Noise-Adaptive Evolutionary Co-Search

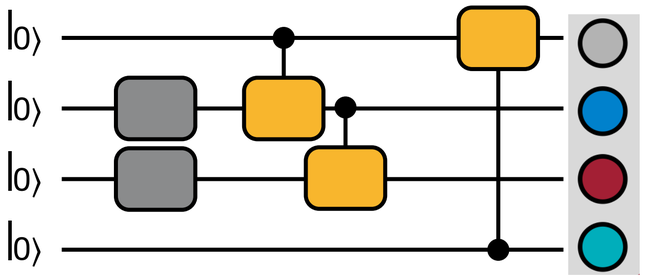
- Search the best SubCircuit and its qubit mapping on target device

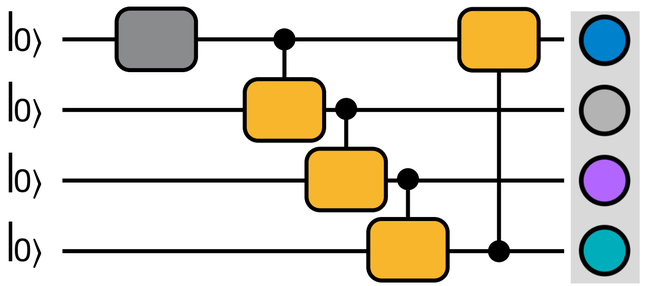


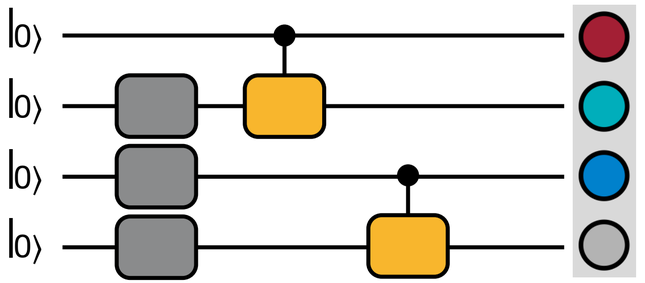
Noise-Adaptive Evolutionary Co-Search

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









Evolutionary Search Engine
Mutation
Crossover

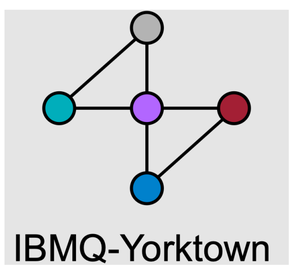
SubCircuit and Qubit Mapping pairs

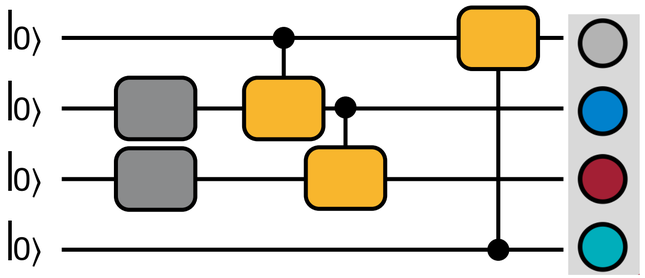
Parameters from SuperCircuit

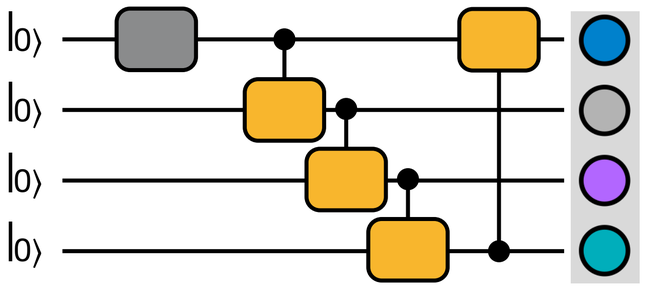
Simulate with noise model
Or
Directly run on real QC device

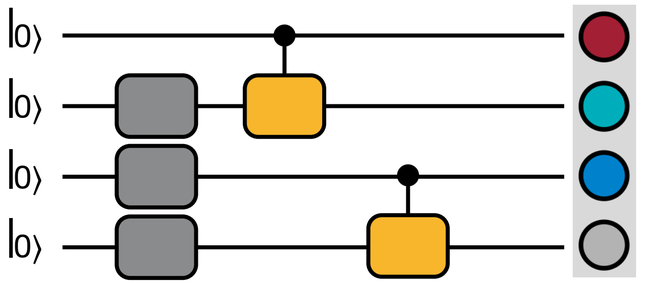
Noise-Adaptive Evolutionary Co-Search

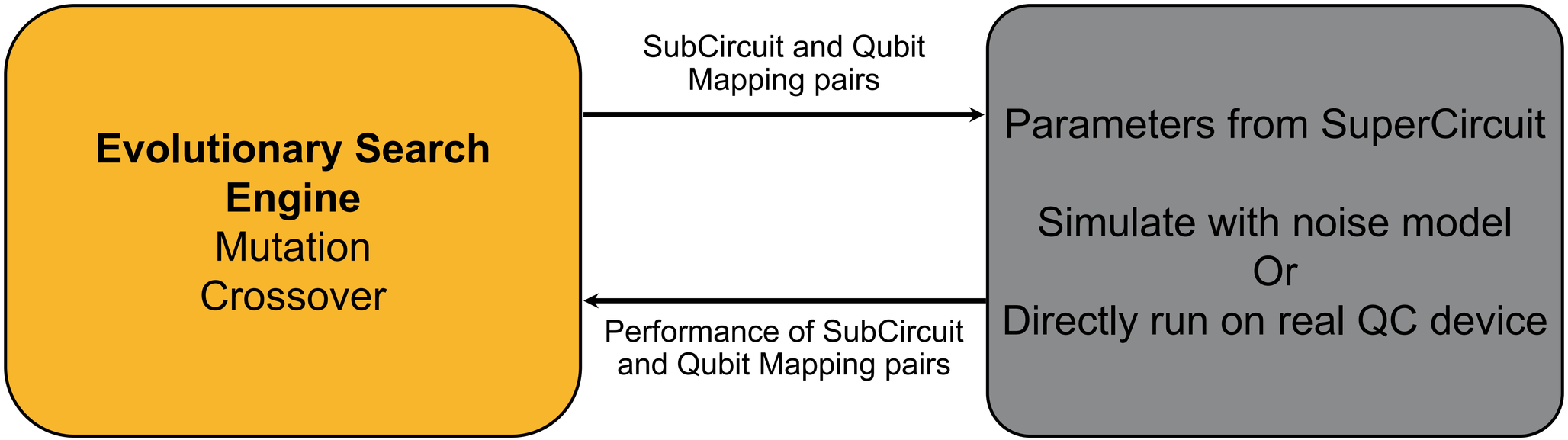
- Search the best SubCircuit and its qubit mapping on target device





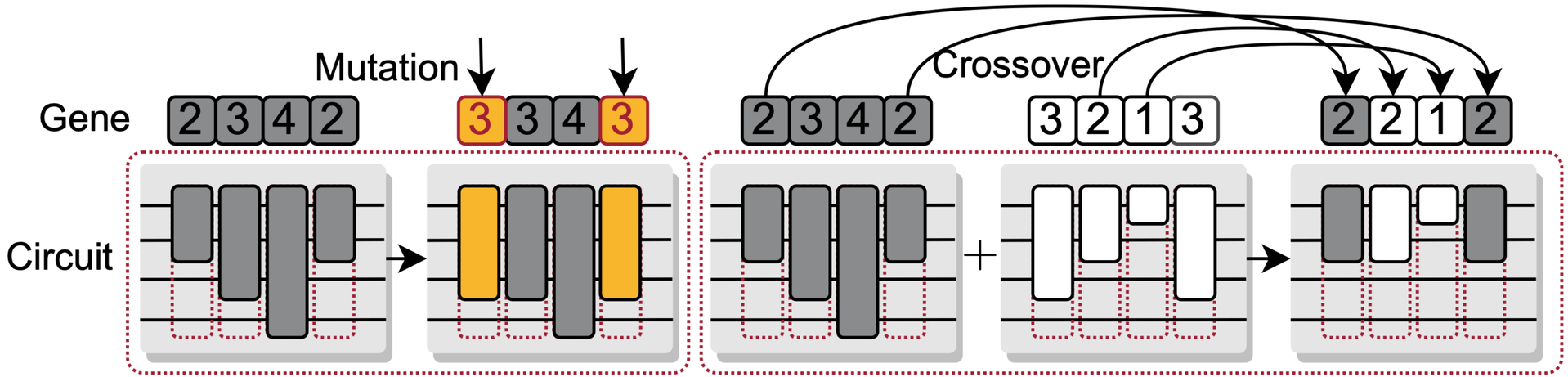






Mutation and Crossover

- Mutation and crossover create new SubCircuit candidates



QuantumNAS

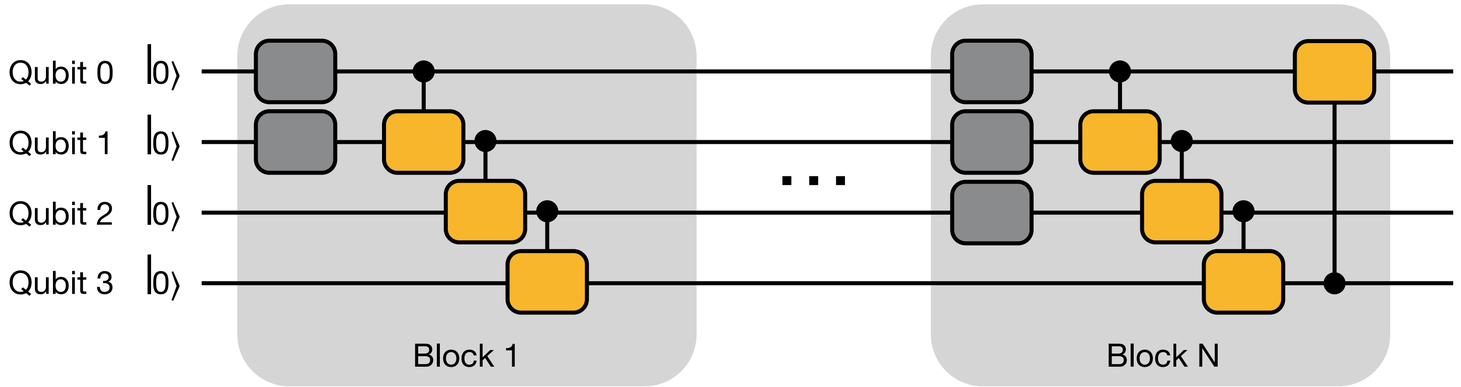
- SuperCircuit Construction and Training
- Noise-Adaptive Evolutionary Co-Search of SubCircuit and Qubit Mapping
- Train the Searched SubCircuit
- Iterative Quantum Gate Pruning

QuantumNAS

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

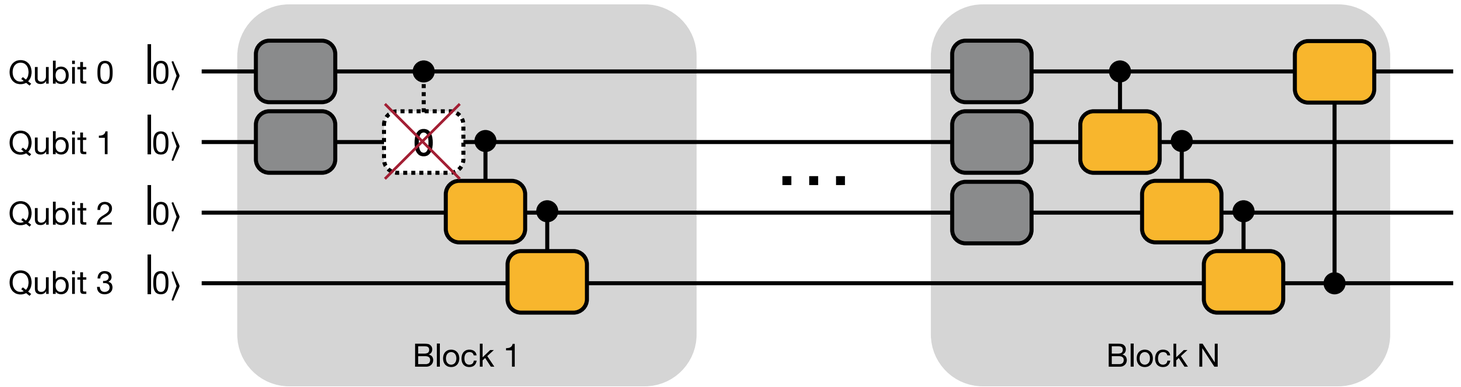
Iterative Pruning

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



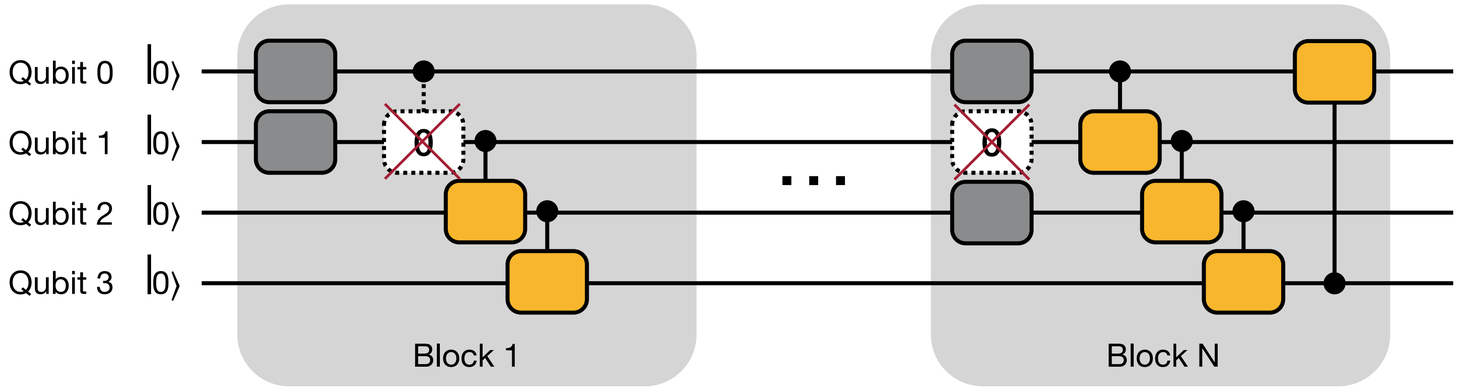
Iterative Pruning

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



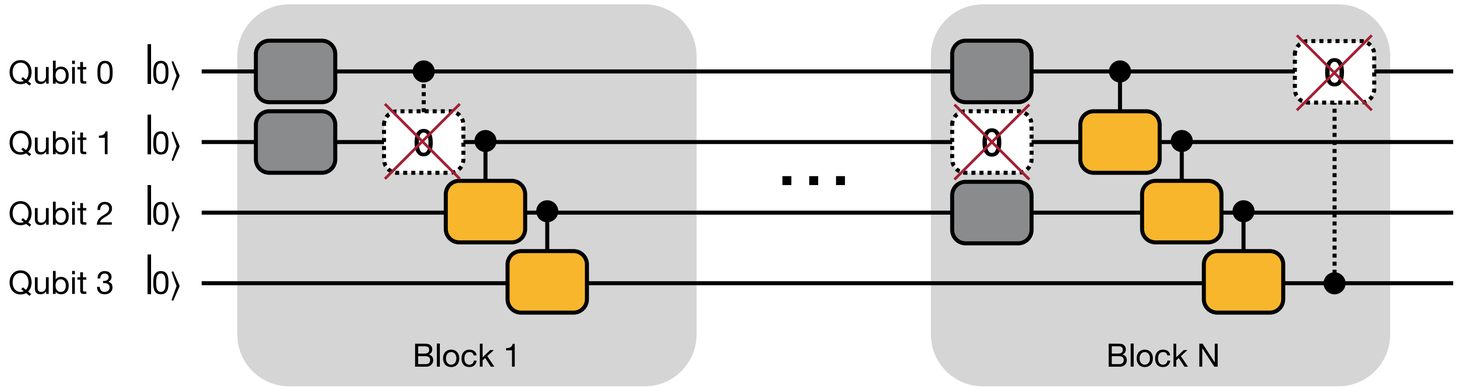
Iterative Pruning

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



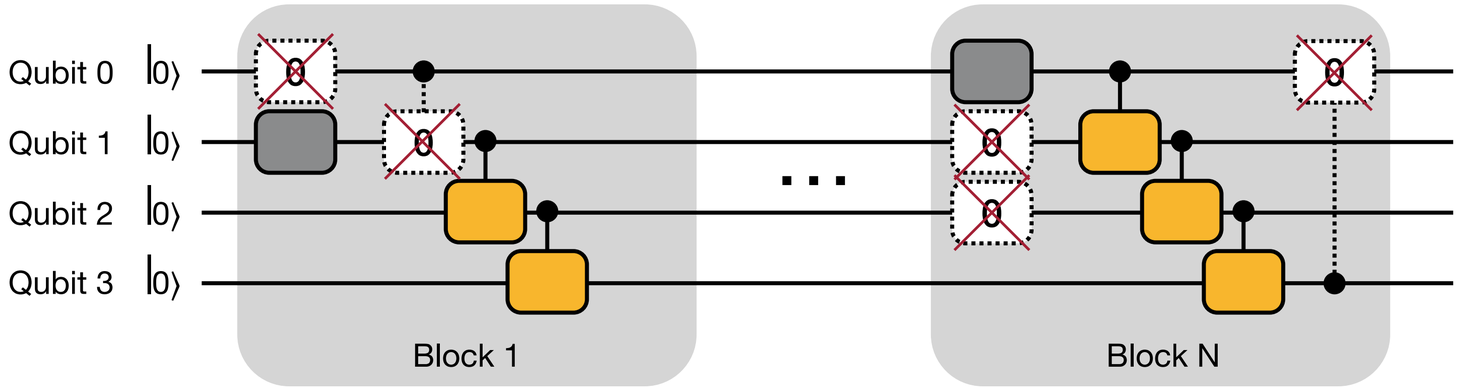
Iterative Pruning

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



Iterative Pruning

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

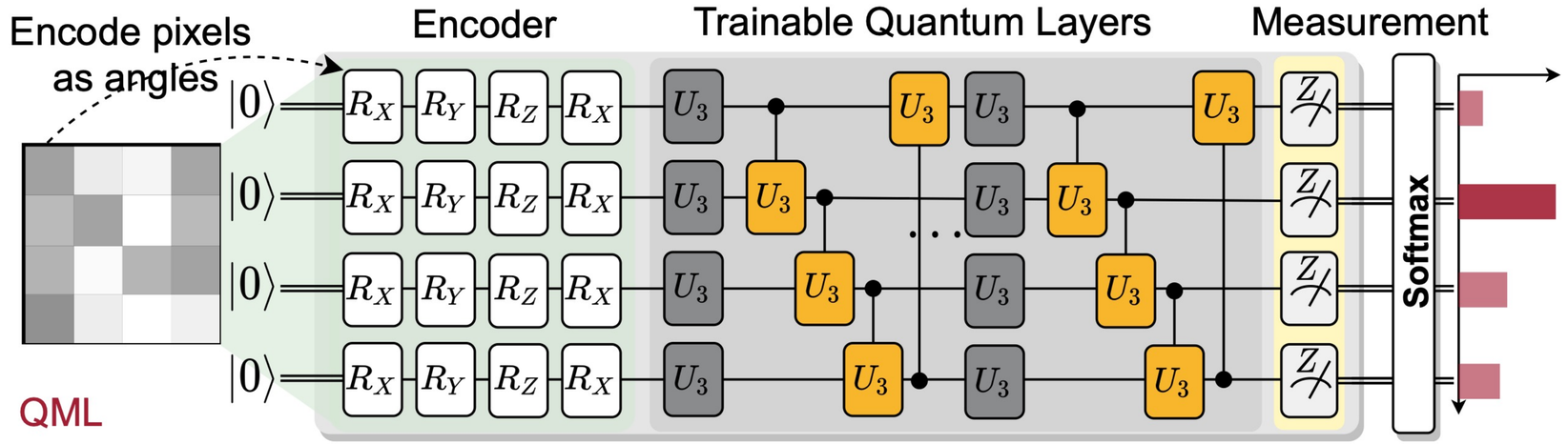


Evaluation Setups: Benchmarks and Devices

- Benchmarks
  - QML classification tasks: MNIST 10-class, 4-class, 2-class, Fashion 4-class, 2-class, Vowel 4-class
  - VQE task molecules: H2, H2O, LiH, CH4, BeH2
- Quantum Devices
  - IBMQ
  - #Qubits: 5 to 65
  - Quantum Volume: 8 to 128

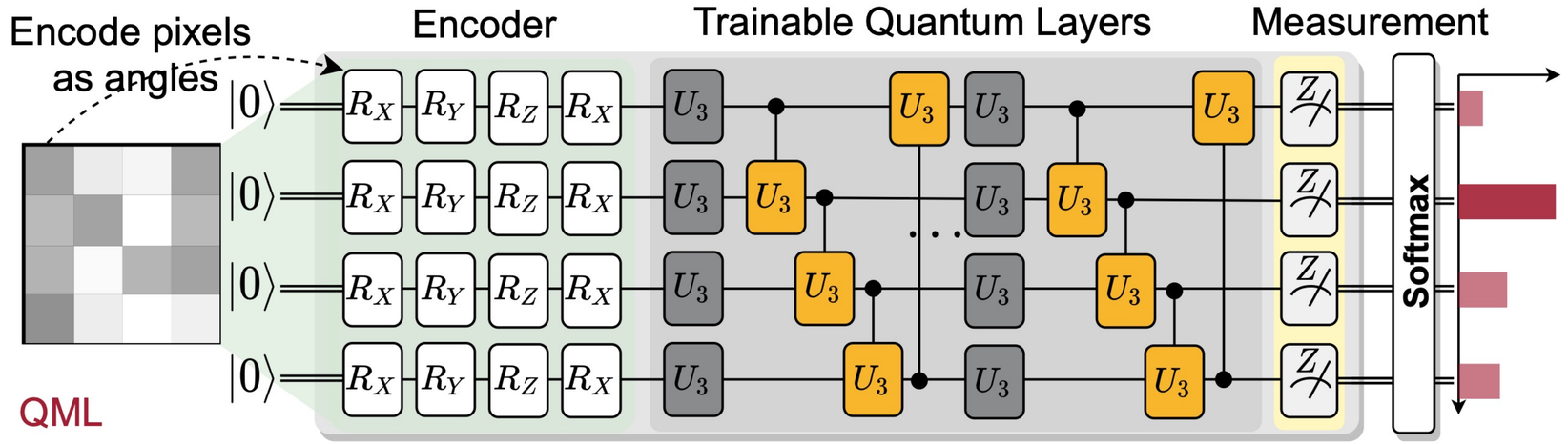
Benchmarks: QNN and VQE

- Quantum Neural Networks: classification

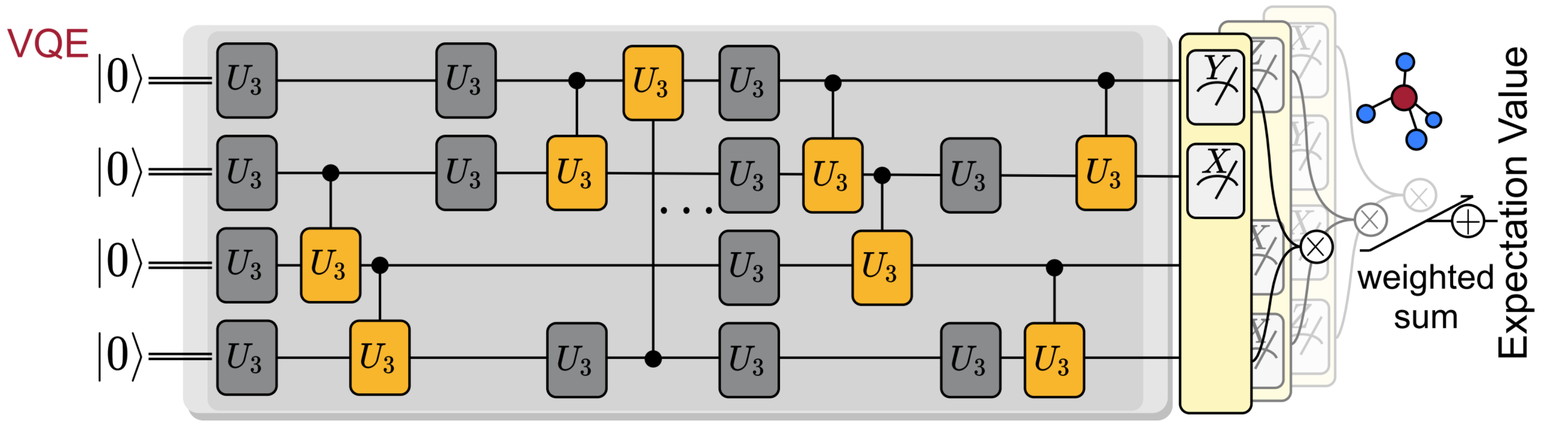


Benchmarks: QNN and VQE

- Quantum Neural Networks: classification

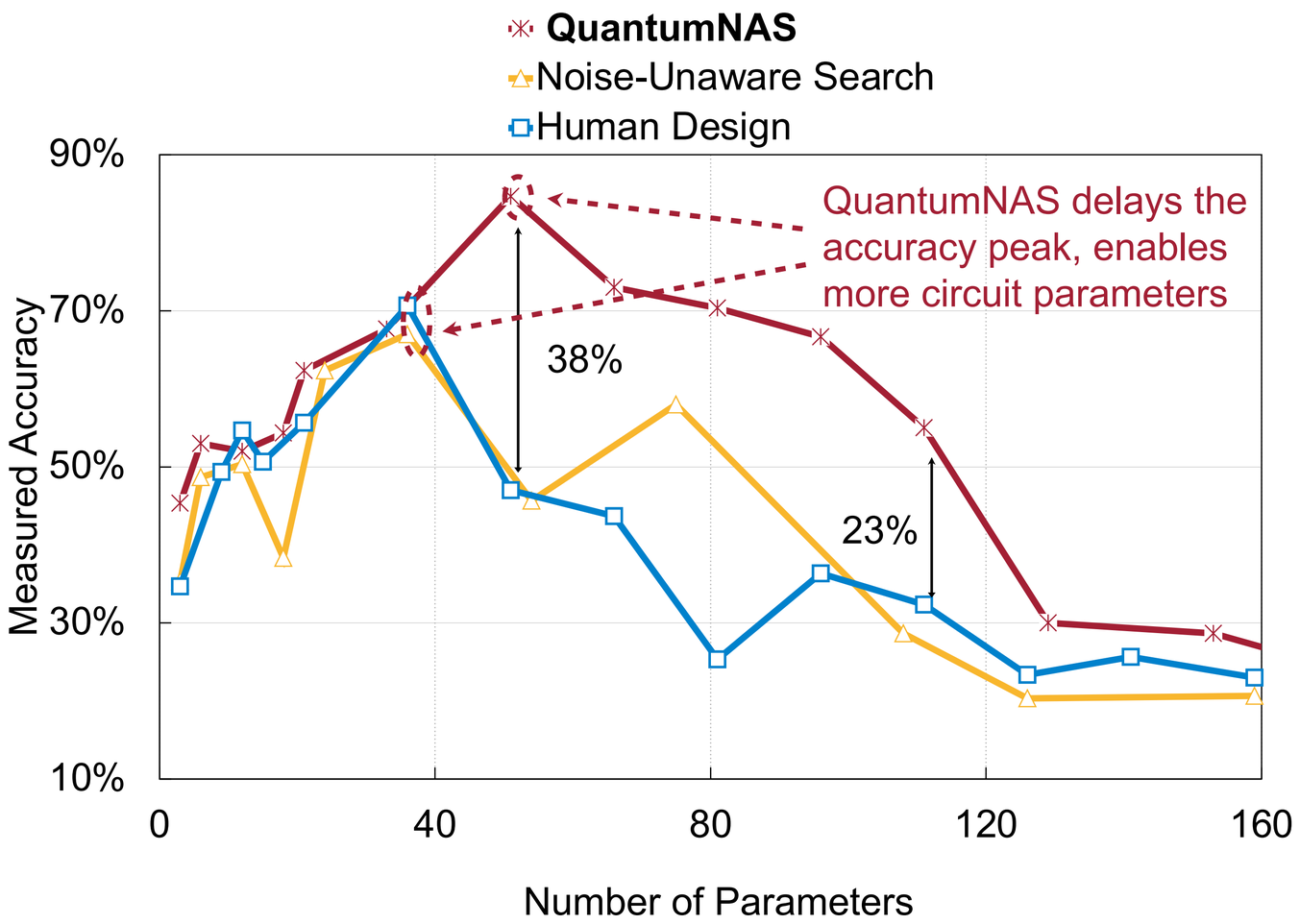


- Variational Quantum Eigensolver: finds the ground state energy of molecule Hamiltonian



QML Results

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



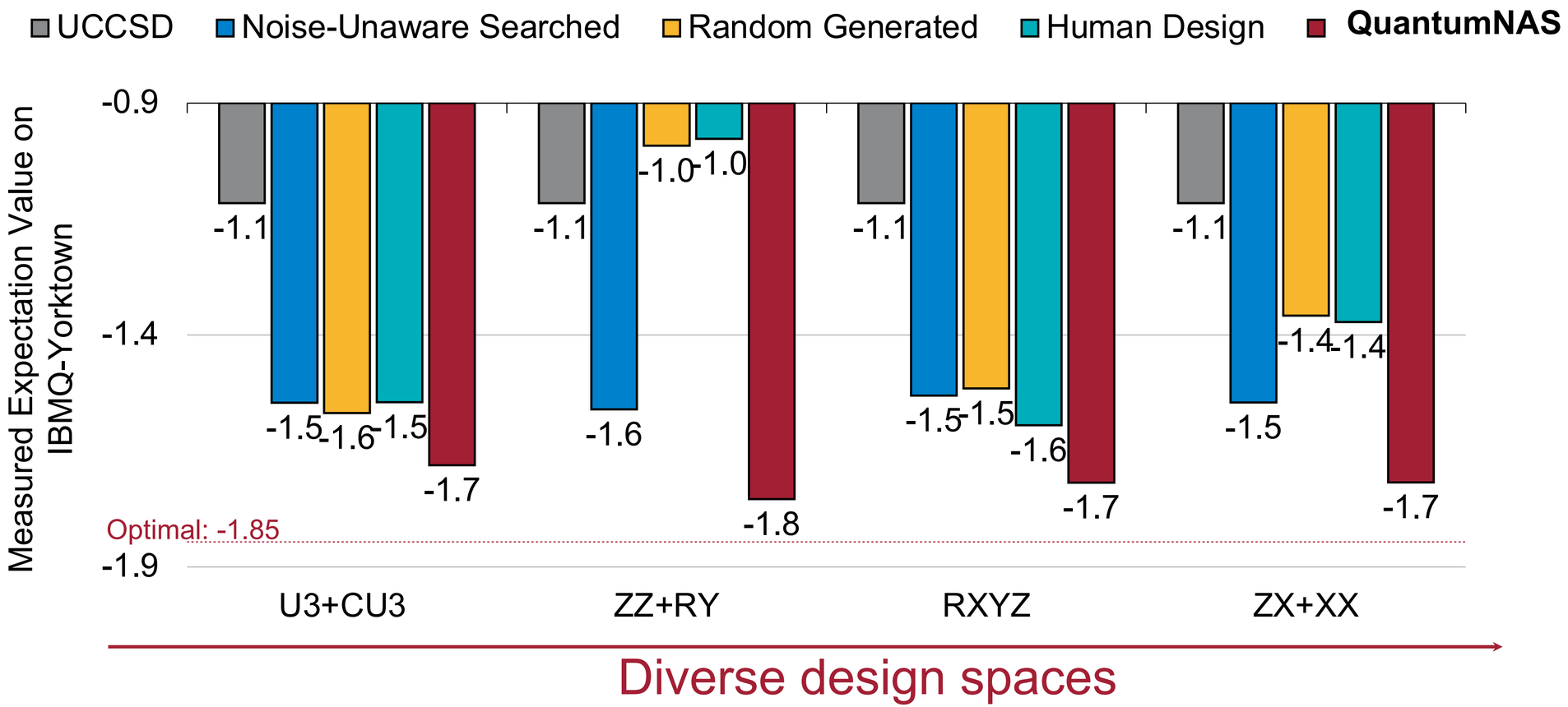
Torch Quantum

qmlsys.mit.edu

MIT HAN LAB

Consistent Improvements on Diverse Design Spaces

- H2 in different design spaces on IBMQ-Yorktown



Scalable to Large #Qubits

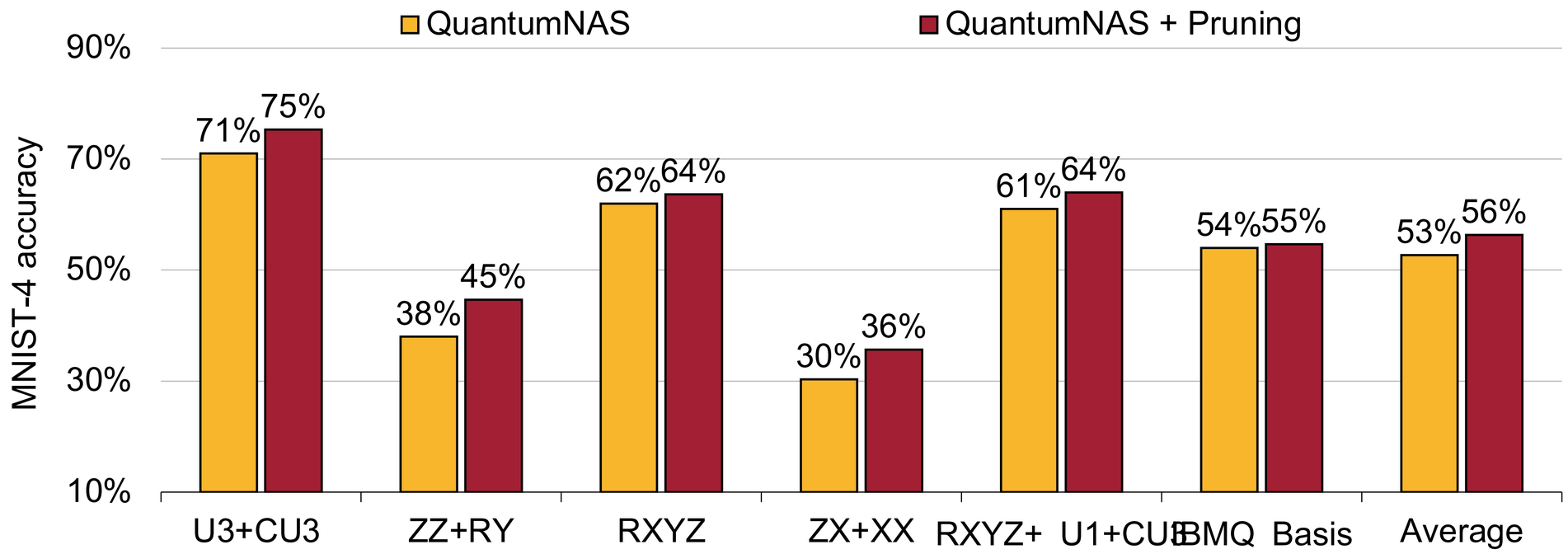
- On large devices
- MNIST-10 accuracy

More Qubits

Method	Noise-Unaware Searched	Random	Human	**QuantumNAS**
Melbourne (15Q, 8QV, use 15Q)	11%	10%	15%	**32%**
Guadalupe (16Q, 32QV, use 16Q)	14%	12%	10%	**15%**
Montreal (27Q, 128QV, use 21Q)	13%	7%	14%	**16%**
Manhattan (65Q, 32QV, use 21Q)	11%	11%	15%	**18%**

Effective of Quantum Gate Pruning

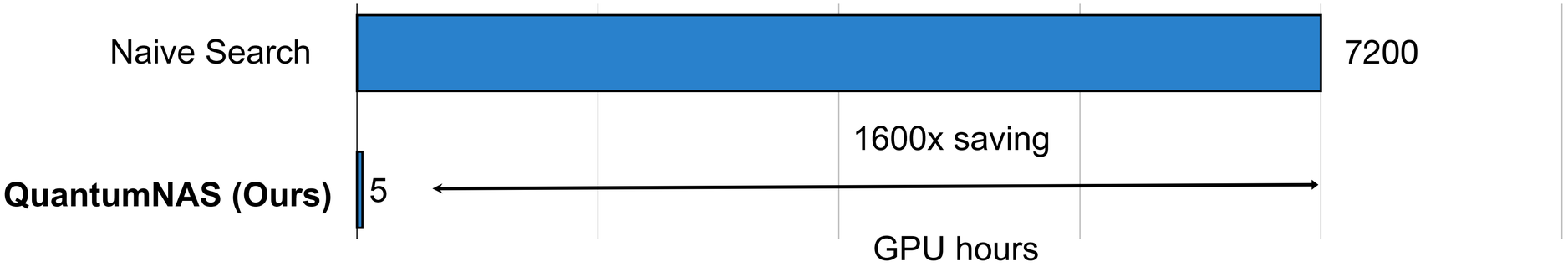
- For MNIST-4, Quantum gate pruning improves accuracy by 3% on average



Time Cost

- On 1 Nvidia Titan RTX 2080 ti GPU

#qubits \ Step	SuperCircuit Training	Noise-Adaptive Co-search	SubCircuit Training	Deployment on Real QC
4 Qubits	0.5h	3h	0.5h	0.5h
15 Qubits	5h	5h	5h	1h
21 Qubits	20h	10h	15h	1h



Torch Quantum

Open-source: TorchQuantum

- TorchQuantum — An open-source library for differentiable quantum simulation

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

TorchQuantum

- Features
  - Easy construction and simulation of quantum circuits in **PyTorch**
  - **Dynamic** computation graph for easy debugging
  - **Gradient** support via autograd
  - **Batch** mode inference and training on CPU/GPU.
  - Easy **deployment** on real quantum devices such as IBMQ
  - Easy **hybrid** classical-quantum model construction
  - (coming soon) **pulse-level** simulation
  - Tutorials, videos and example projects of QML and using ML to optimize quantum computer system problems
- Will be on **QCE** (Quantum Computing Engineering) tutorials

Open-source: TorchQuantum

- 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),
```

Torch Quantum

qmlsys.mit.edu

MIT HAN LAB

Open-source: TorchQuantum

- 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)
```

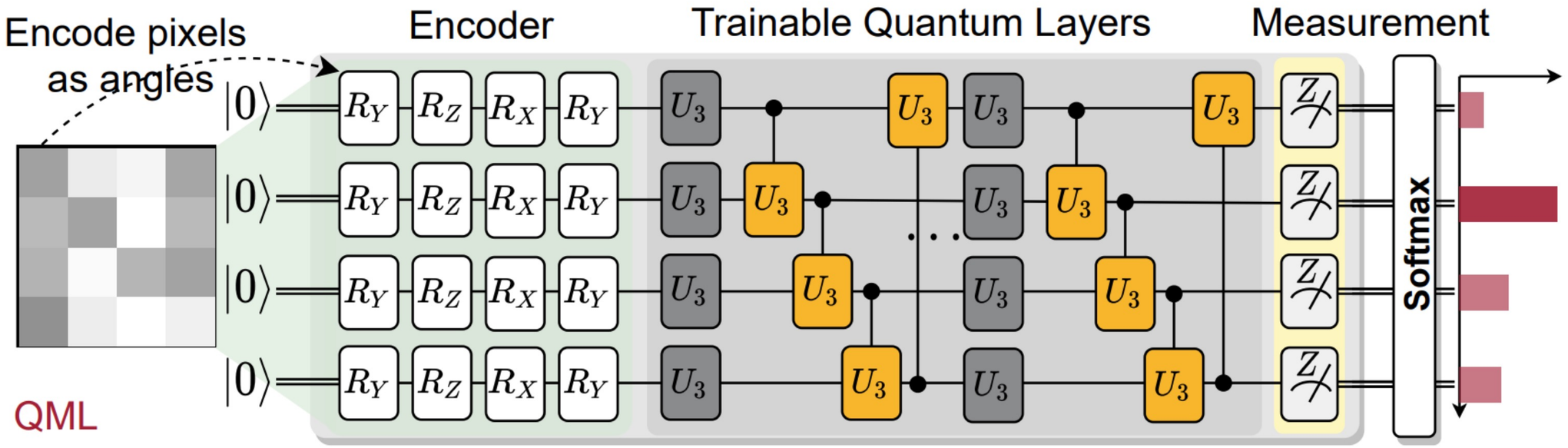
- Matrix-vector multiplication: torch.einsum and torch.bmm

Torch Quantum

qmlsys.mit.edu

MIT HAN LAB

MNIST Example with QNN



MNIST Example

Initialize a quantum device

Specify encoder gates

Specify trainable gates

```
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

qmlsys.mit.edu

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

qmlsys.mit.edu

MIT HAN LAB

MNIST Example

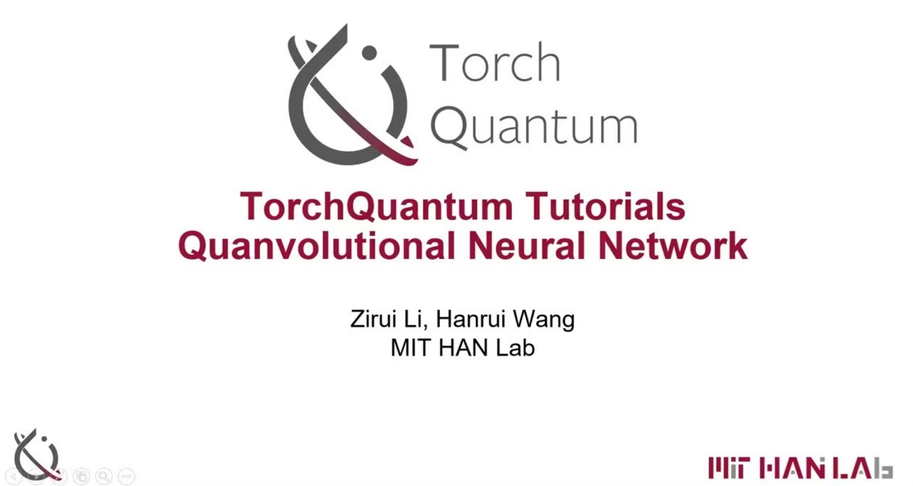
Initialize a QiskitProcessor

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)
```

Examples and tutorials

- Tutorial Colab and videos



Thank you for listening!

- Take home
  - **TorchQuantum:** fast open-source **library** for quantum simulation
  - **QuantumNAS:** framework to search for **noise-robust** circuit architecture

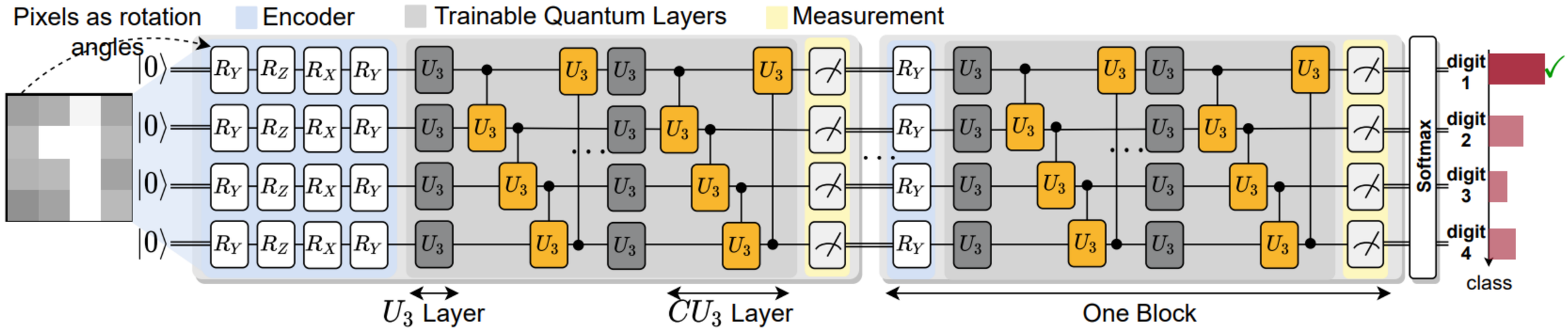
Torch Quantum

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

qmlsys.mit.edu

QuantumNAT

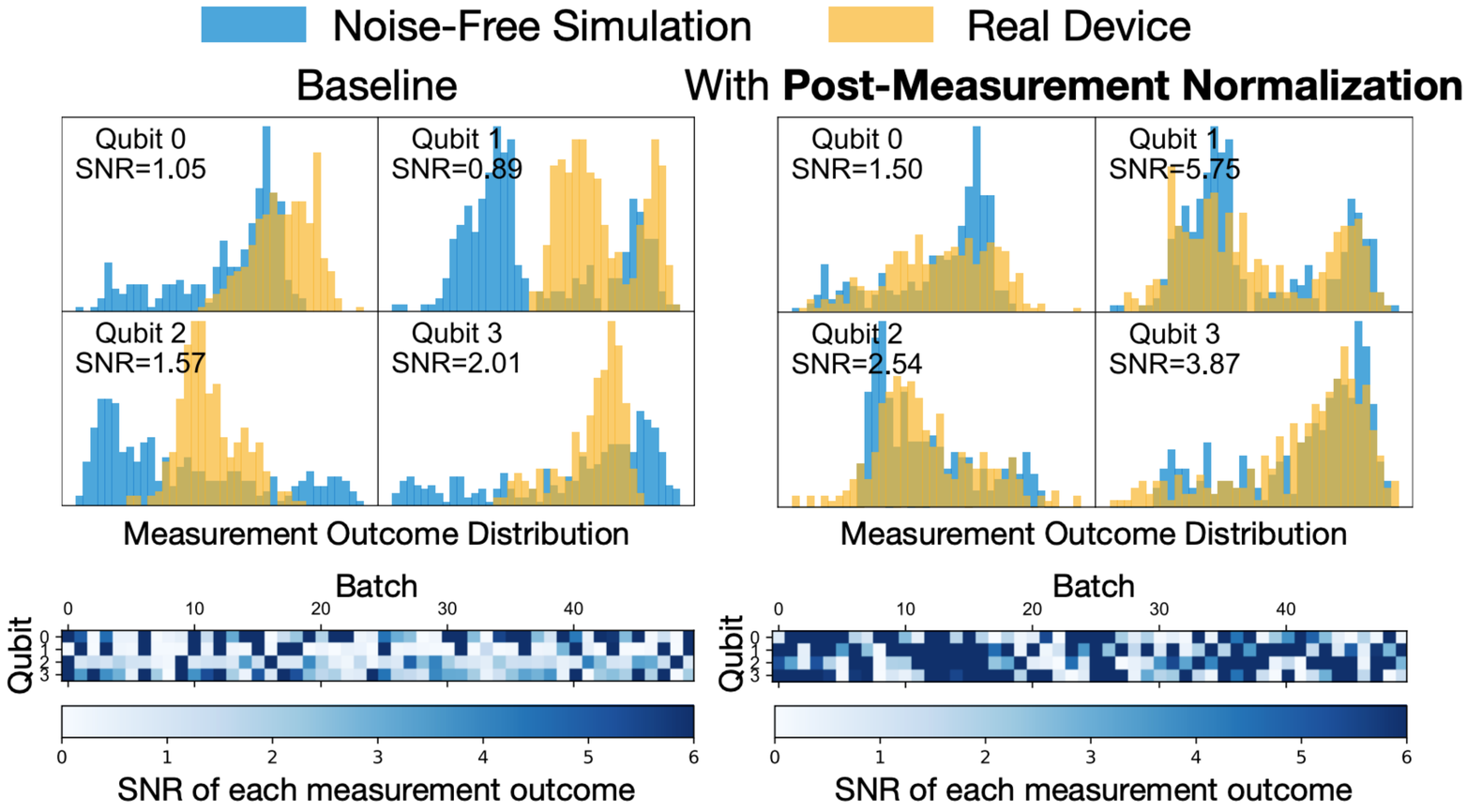
- **QuantumNAS: find the circuit architecture robust to noise**
- **QuantumNAT: further make the parameter robust to noise**



[DAC'22] Wang et, at, QuantumNAT: Quantum Noise-Aware Training with Noise Injection, Quantization and Normalization

Post-Measurement Normalization

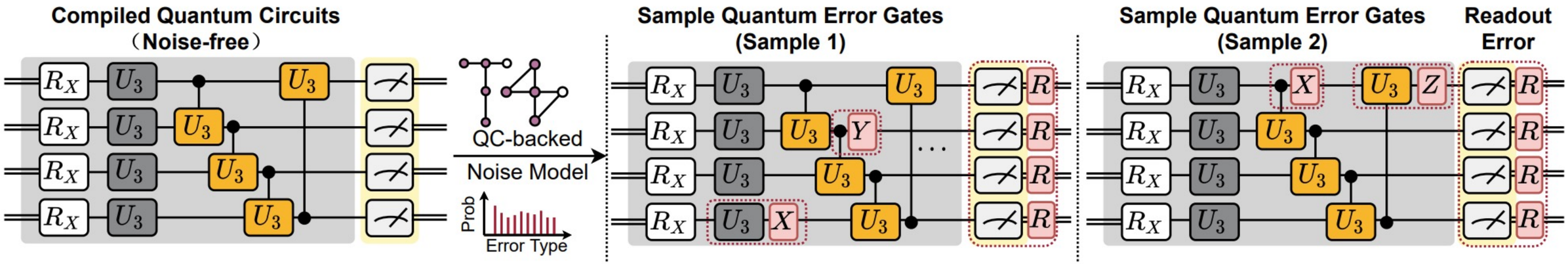
- **Normalize the measurement outcomes on the batch dimension**



2nd block output of a 3-block model, IBMQ-Quito

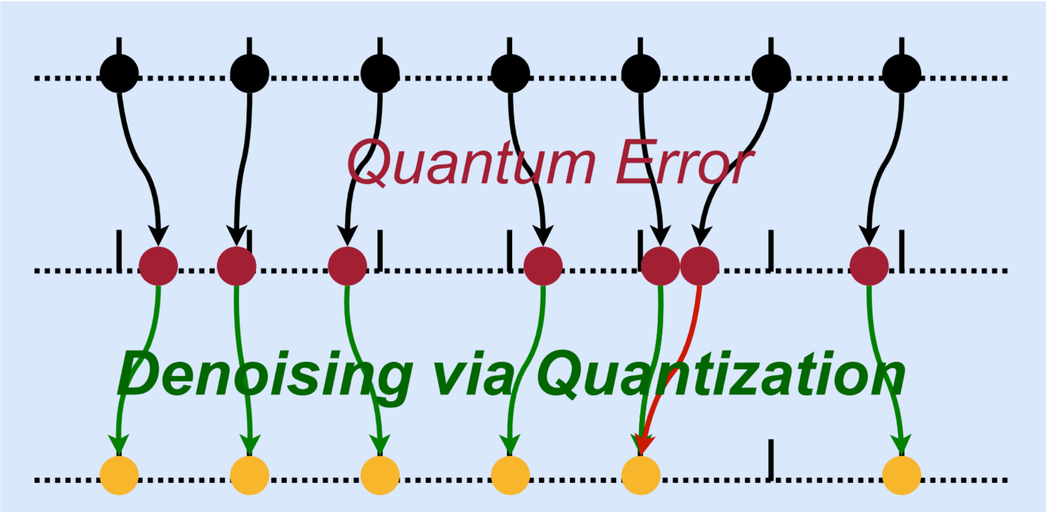
Noise Injection

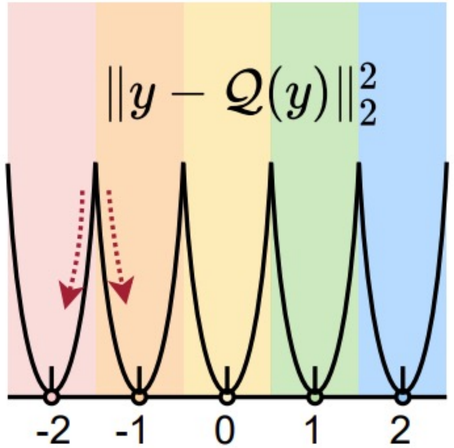
- **Insert noise gate during training, according to the noise model**
- **For each step, sample new positions for noise gates**



Post-Measurement Quantization

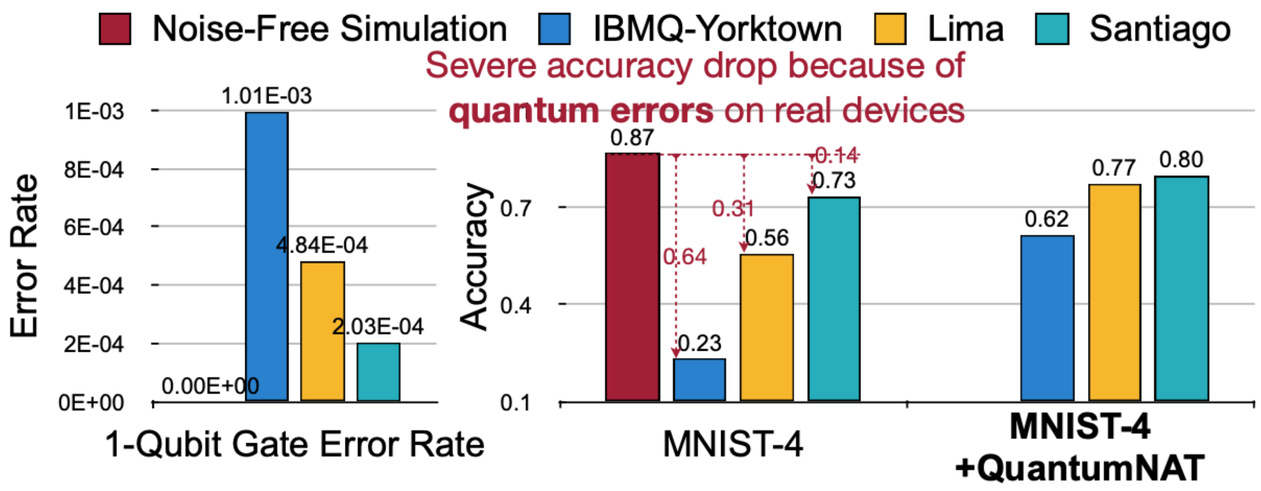
- **Quantization provides denoising effects**
- **Quadratic penalty loss to encourage measurement outcomes close to quantization centroids**

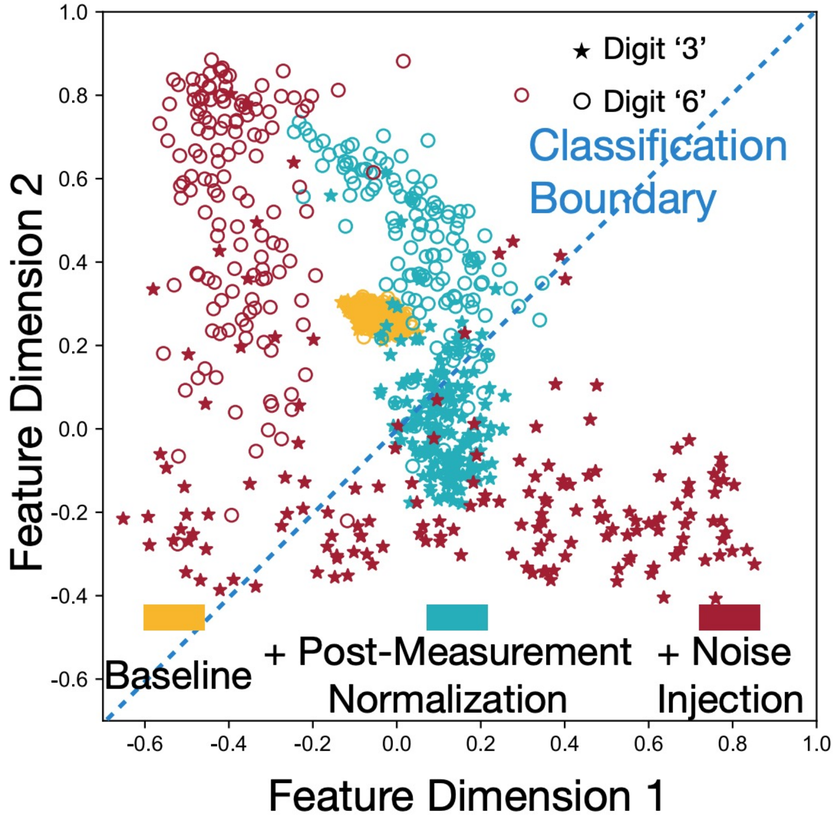




Evaluation

- **On classification tasks with QNN**

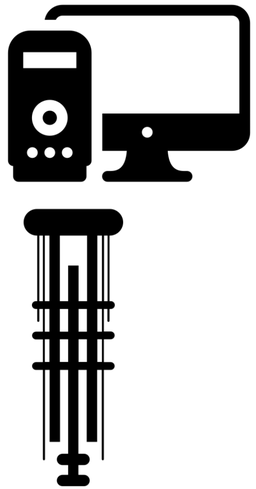




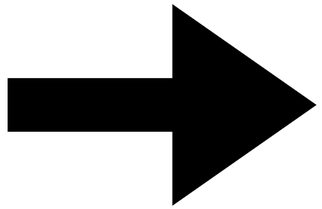
QOC (Quantum On-Chip Training)

Train on classical

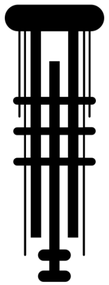
Deploy on quantum



Scalability ↑



Train & Deploy on quantum



[DAC'22] Wang et, at, QOC: Quantum On-Chip Training with Parameter Shift and Gradient Pruning