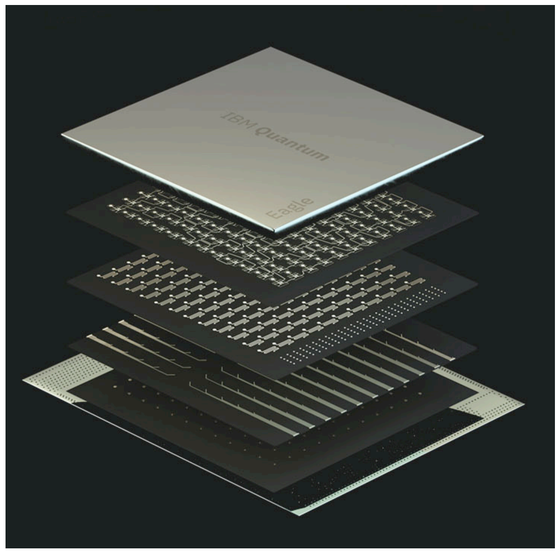
TorchQuantum: A Fast Library for Quantum Computing in PyTorch

Organizers: Hanrui Wang[1], Zhiding Liang[2], Jinglei Cheng[3], Dantong Li[4], Weiwen Jiang[5], Yongshan Ding[4], Fred Chong[6], Song Han[1]

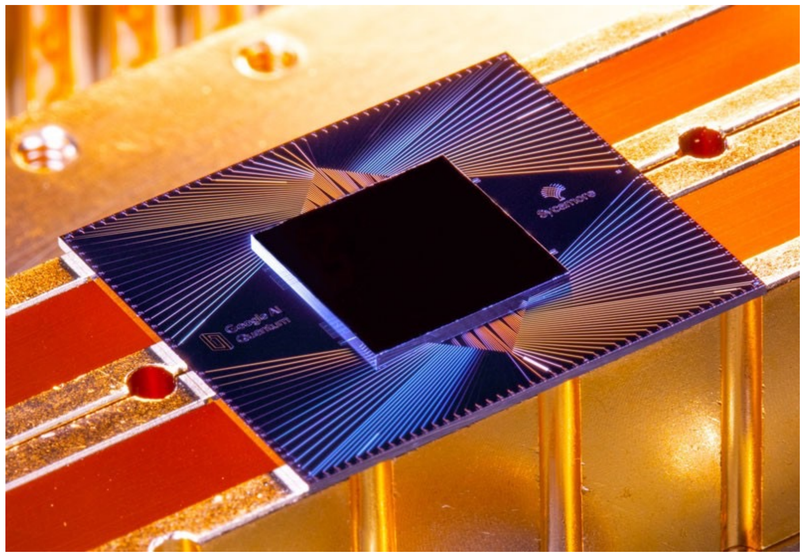
[1]MIT, [2]Notre Dame, [3]Purdue, [4]Yale, [5]GMU, [6]UChicago

Quantum Computing

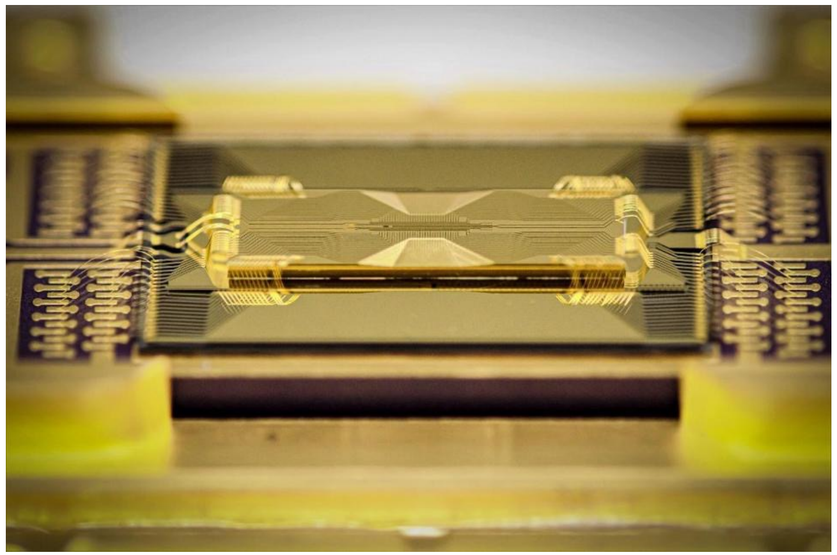
- Fast progress of quantum devices
- Different technologies
  - Superconducting, trapped ion, neutral atom, photonics, etc.



IBM
433 Qubit



Google
53 Qubits



IonQ
32+ Qubits

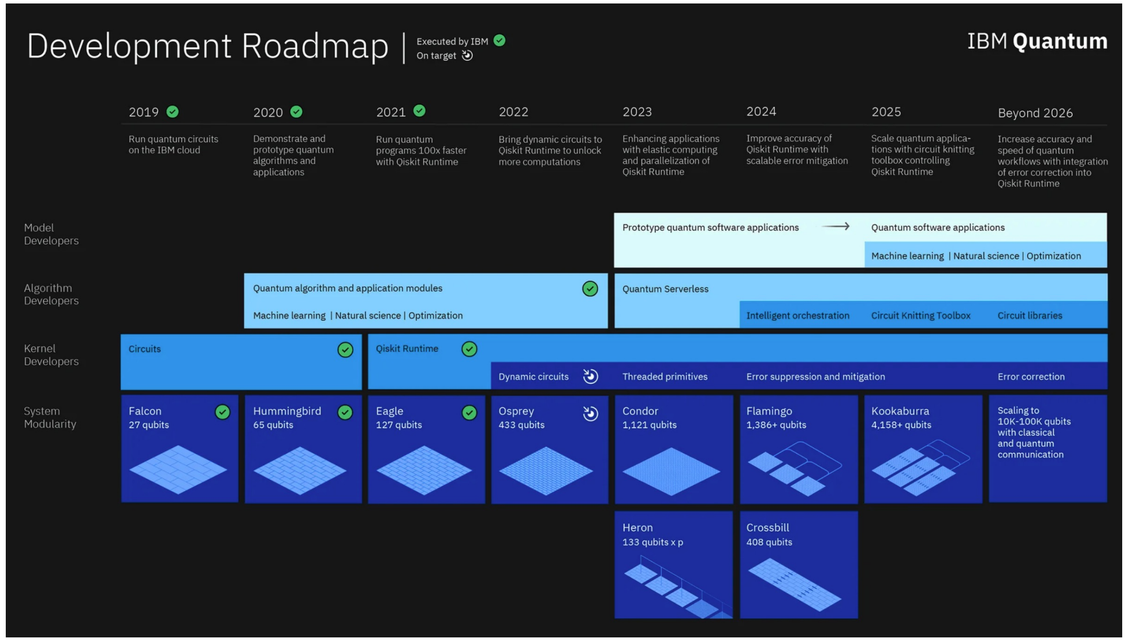
Torch Quantum

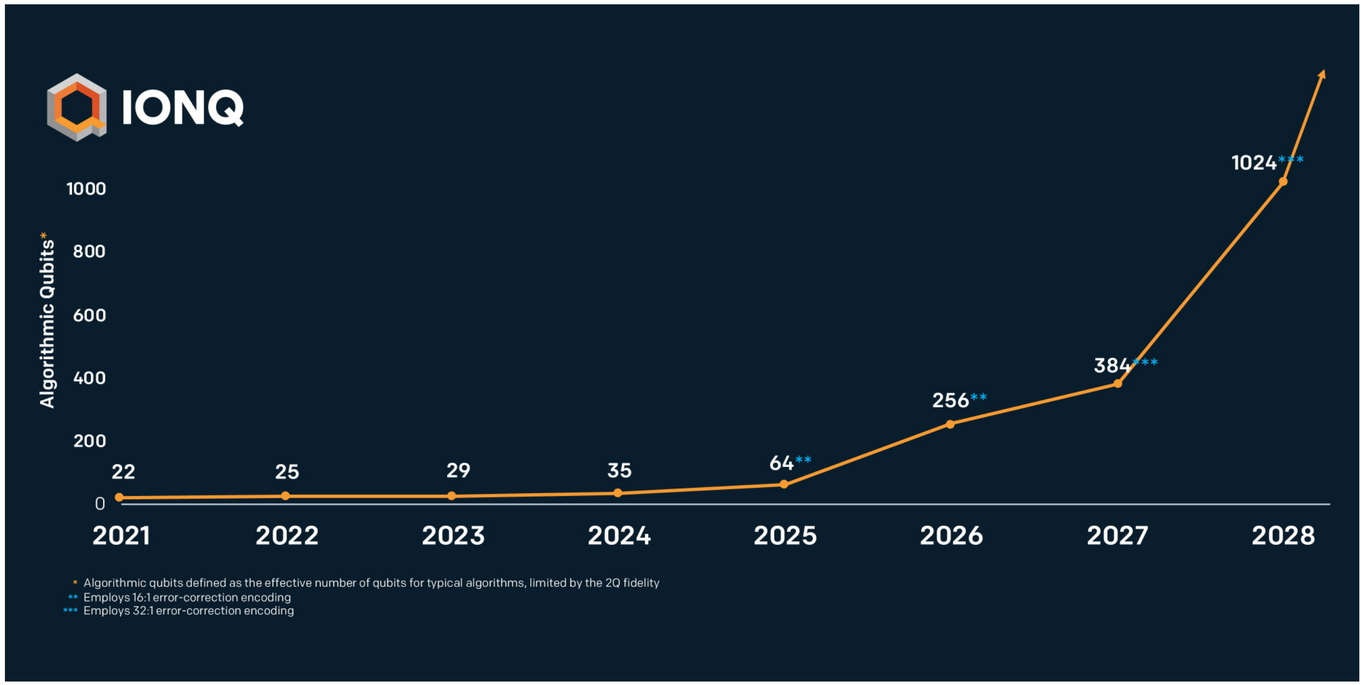
Materials at: https://torchquantum.org

MIT HAN LAB

Quantum Computing

- The number of qubits increases exponentially over time
- The computing power increases exponentially with the number of qubits
- => “doubly exponential” rate





IBM Roadmap

IonQ Roadmap

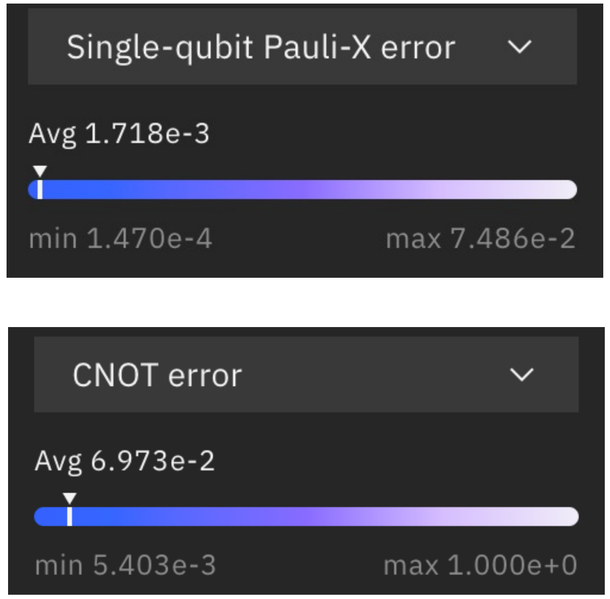
Torch Quantum

Materials at: https://torchquantum.org

MIT HAN LAB

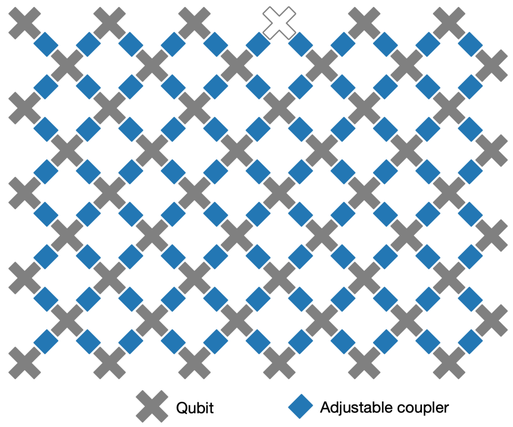
Quantum Computing in NISQ Era

- Noisy Intermediate-Scale Quantum (NISQ)
  - **Noisy**: qubits are sensitive to environment; quantum gates are unreliable
  - **Limited number of qubits**: tens to hundreds of qubits
  - **Limited connectivity:** no all-to-all connections



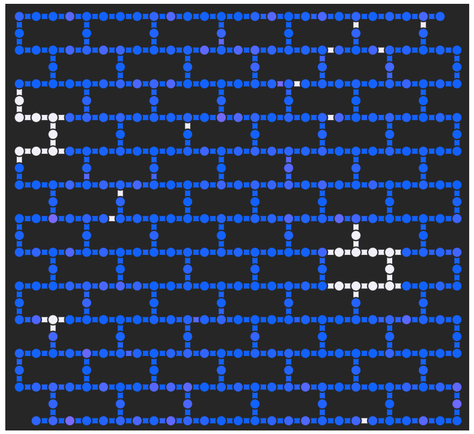
IBMQ Gate Error Rate

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



Google Sycamore 53Q

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

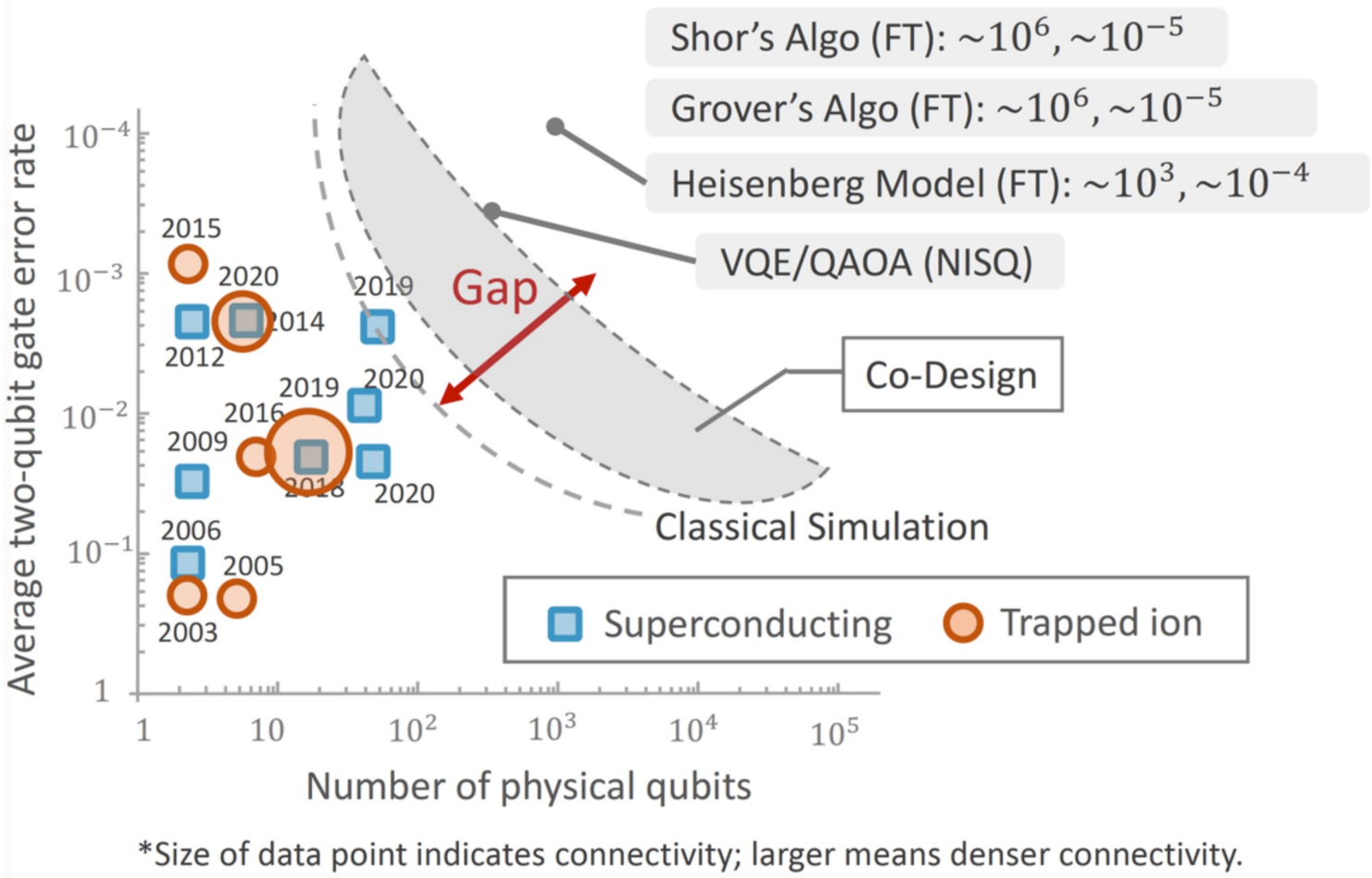


IBM Washington 433Q

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

A Large Gap between Powerful Quantum Algorithms and Current Devices

- Close the gap with **machine learning** and **hardware-aware algorithm design**



*Size of data point indicates connectivity; larger means denser connectivity.

Fred Chong, https://indico.fnal.gov/event/47147/attachments/138920/174186/Chong-JTFI-2021.pdf18.pdf

Materials at: https://torchquantum.org

Torch Quantum

MIT HAN LAB

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 Library

- 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, OpenQASM
  - And so on…

Torch Quantum

Materials at: https://torchquantum.org

MIT HAN LAB

TorchQuantum Tutorial Outline

Section 1

Quantum Basics and TorchQuantum Usage

1.1 Introduction to Quantum Computing (Prof. Yongshan Ding)

1.2 TorchQuantum API and examples (Hanrui Wang)

Section 2

Use TorchQuantum on Gate Level Optimization

2.1 QuantumNAS Ansatz Search and On-Chip Training (Hanrui Wang)

2.2 Variational Quantum Circuit Compression (Prof. Weiwen Jiang)

2.3 Various VQA Gradient Estimation Methods (Dantong Li)

Section 3

Use TorchQuantum on Pulse Level Optimization

3.1 Intro to Pulse Level Quantum Control (Zhiding Liang)

3.2 Variational Pulse Learning & Optimal Control (Jinglei Cheng)

Torch Quantum

Materials at: https://torchquantum.org

MIT HAN LAB

TorchQuantum Tutorial Outline

Section 1

Quantum Basics and TorchQuantum Usage

1.1 Introduction to Quantum Computing (Prof. Yongshan Ding)

1.2 TorchQuantum API and examples (Hanrui Wang)

Section 2

Use TorchQuantum on Gate Level Optimization

2.1 QuantumNAS Ansatz Search and On-Chip Training (Hanrui Wang)

2.2 Variational Quantum Circuit Compression (Prof. Weiwen Jiang)

2.3 Various VQA Gradient Estimation Methods (Dantong Li)

Section 3

Use TorchQuantum on Pulse Level Optimization

3.1 Intro to Pulse Level Quantum Control (Zhiding Liang)

3.2 Variational Pulse Learning & Optimal Control (Jinglei Cheng)

Torch Quantum

Materials at: https://torchquantum.org

MIT HAN LAB

TorchQuantum Tutorial Outline

Section 1

Quantum Basics and TorchQuantum Usage

1.1 Introduction to Quantum Computing (Prof. Yongshan Ding)

1.2 TorchQuantum API and examples (Hanrui Wang)

Section 2

Use TorchQuantum on Gate Level Optimization

2.1 QuantumNAS Ansatz Search and On-Chip Training (Hanrui Wang)

2.2 Variational Quantum Circuit Compression (Prof. Weiwen Jiang)

2.3 Various VQA Gradient Estimation Methods (Dantong Li)

Section 3

Use TorchQuantum on Pulse Level Optimization

3.1 Intro to Pulse Level Quantum Control (Zhiding Liang)

3.2 Variational Pulse Learning & Optimal Control (Jinglei Cheng)

Torch Quantum

Materials at: https://torchquantum.org

MIT HAN LAB

TQ for Statevector simulation

- Performing matrix-vector multiplication between the gate matrix and statevector
  - The tq.QuantumDevice stores the statevectors
    - `q_dev = tq.QuantumDevice(n_wires=5)`
  - Three ways of applying quantum gates: method 1:
    - `import torchquantum.functional as tqf`
    - `tqf.h(q_dev, wires=1)`

TQ for Statevector simulation

- Performing matrix-vector multiplication between the gate matrix and statevector
  - The tq.QuantumDevice stores the statevectors
    - `q_dev = tq.QuantumDevice(n_wires=5)`
  - Three ways of applying quantum gates: method 2:
    - `h_gate = tq.H()`
    - `h_gate(q_dev, wires=3)`

TQ for Statevector simulation

- Performing matrix-vector multiplication between the gate matrix and statevector
  - The tq.QuantumDevice stores the statevectors
    - `q_dev = tq.QuantumDevice(n_wires=5)`
  - Three ways of applying quantum gates: method 3:
    - `q_dev.h()`
    - `q_dev.rx(wires=1, params=0.2 * np.pi)`

TQ for Statevector simulation

- Performing matrix-vector multiplication between the gate matrix and statevector
- The implementation of statevector and quantum gates are using the native data structure in PyTorch

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

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

TQ for Density Matrix simulation

- Performing matrix-vector multiplication between the gate matrix and statevector
  - The tq.DensityMatrix class stores the denstiy matrix
    - `q_mat = tq.DesityMatrix(n_wires=5)`
  - Applying quantum gates
    - `q_mat.h()`
    - `q_mat.rx(wires=1, params=0.2 * np.pi)`

```python
class DensityMatrix(nn.Module):

    def __init__(self,n_wires: int,
                 bsz: int = 1):
        """Init function for DensityMatrix class(Density Operator)
        Args:
            n_wires (int): how many qubits for the densityMatrix.
        """
        super().__init__()

        self.n_wires=n_wires
        """
        For example, when n_wires=3
        matrix[001110] denotes the index of |001><110|=|index1><index2|
        Set Initial value the density matrix of the pure state |00...00>
        """
        _matrix = torch.zeros(2 ** (2*self.n_wires), dtype=C_DTYPE)
        _matrix[0] = 1 + 0j
        _matrix = torch.reshape(_matrix, [2]*(2*self.n_wires))
        self.register_buffer('matrix', _matrix)


        repeat_times = [bsz] + [1] * len(self.matrix.shape)
        self._matrix = self.matrix.repeat(*repeat_times)
        self.register_buffer('matrix', self._matrix)

        """
        Whether or not calculate by states
        """
        self._calc_by_states=True
```

Torch Quantum

Materials at: https://torchquantum.org

MIT HAN LAB

Dynamic computation graph

- Performing matrix-vector multiplication between the gate matrix and statevector
  - The tq.QuantumDevice class stores the statevectors
    - `q_dev = tq.QuantumDevice(n_wires=5)`
    - `q_dev.h(wires=1)`
    - `print(q_dev)`
    - `q_dev.sx(wires=3)`
    - `print(q_dev)`

Batch Mode Tensorized Processing

- Performing matrix-vector multiplication between the gate matrix and statevector
  - The tq.QuantumDevice class stores the statevectors
    - `q_dev = tq.QuantumDevice(n_wires=5, bsz=64)`

GPU Support

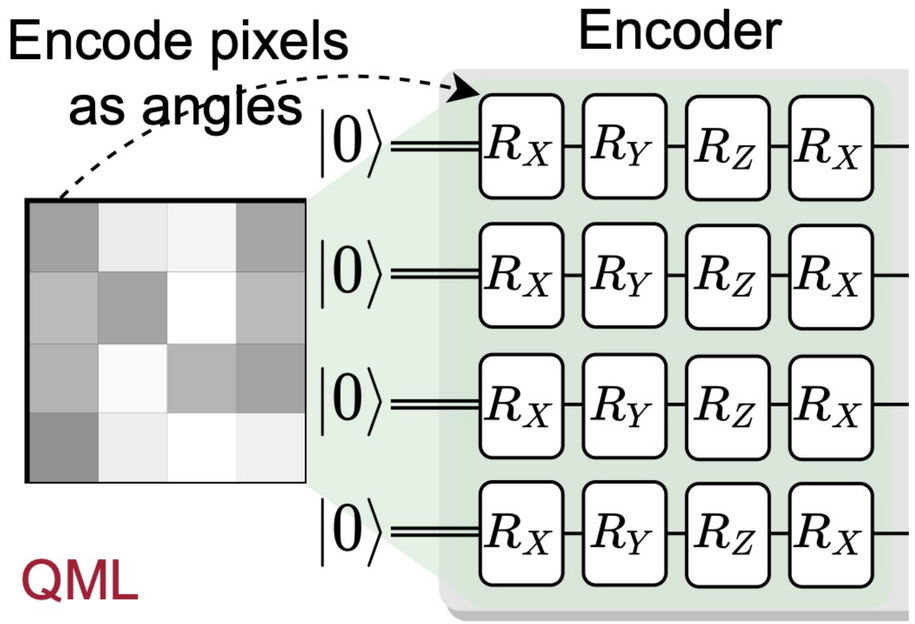
- Performing matrix-vector multiplication between the gate matrix and statevector
  - The tq.QuantumDevice class stores the statevectors
    - `q_dev = tq.QuantumDevice(n_wires=5, bsz=64)`
    - `q_dev.to(torch.device('cuda'))`
  - Leverage the fast GPU support

Automatic Gradient Computation

- Performing matrix-vector multiplication between the gate matrix and statevector
  - The tq.QuantumDevice class stores the statevectors
    - `q_dev = tq.QuantumDevice(n_wires=2)`
    - `target_quantum_state = torch.tensor([0, 0, 0, 1])`
    - `loss = 1 - (q_state.get_states_1d()[0] @ target_quantum_state).abs()`
    - `loss.backward()`

Encoding Classical Data to Quantum various encoder support

- `tq.AmplitudeEncoder()` encodes the classical values to the amplitude of quantum statevector
- `tq.PhaseEncoder()` encodes the classical values using the rotation gates



Measurement

- Measure the state on the default Z basis
  - `tq.measure(qdev, n_shots=1024)`

- Obtain the expval on a observable by stochastic sampling (doable on simulator and real quantum hardware)
  - `expval_sampling = tq.expval_joint_sampling(qdev, 'ZX', n_shots=1024)`

- Obtain the expval on a observable by analytical computation (only doable on classical simulator)
  - `expval = tq.expval_joint_analytical(qdev, 'ZX')`

Construct a Circuit Class

- Construct a class for circuit model
- `tq.QuantumModule` class
- In the __init__ function, create all the gates that will be used
- In the forward function, specify how the gates will be used in the circuit

```
Class q_model(tq.QuantumModule)
  def __init__():
     self.n_wires = 2
     self.rx_0 = tq.RX(has_params=True, trainable=True)
     self.ry_0 = tq.RY(has_params=True, trainable=True)
  def forward(q_dev):
     self.rx_0(q_dev)
     self.ry_0(q_dev)
```

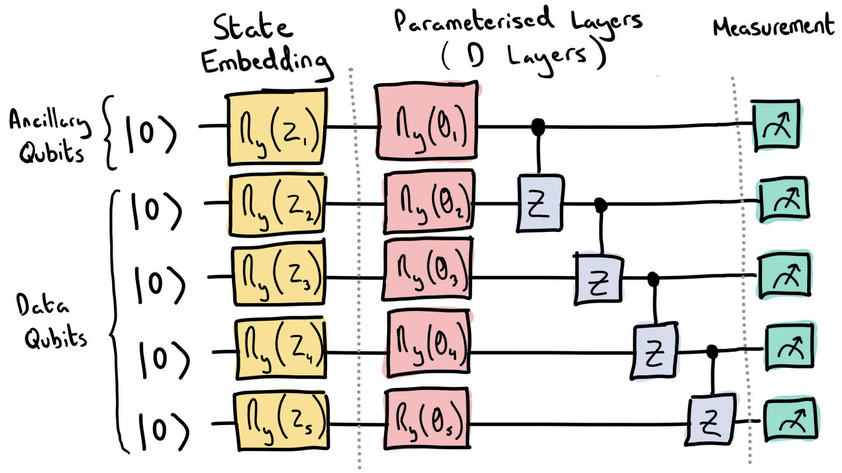
Conversion tq model to other frameworks

- Convert the tq.QuantumModule to other frameworks such as Qiskit
  - `from torchquantum.plugins.qiskit_plugin import tq2qiskit`
  - `circ = tq2qiskit(q_dev, q_model)`
  - `circ.draw('mpl')`
- Convert to OpenQASM
  - `qdev = tq.QuantumDevice(n_wires=2, bsz=5, device="cpu", record_op=True)`
  - `from torchquantum.plugins import op_history2qasm`
  - `print(op_history2qasm(qdev.n_wires, qdev.op_history))`

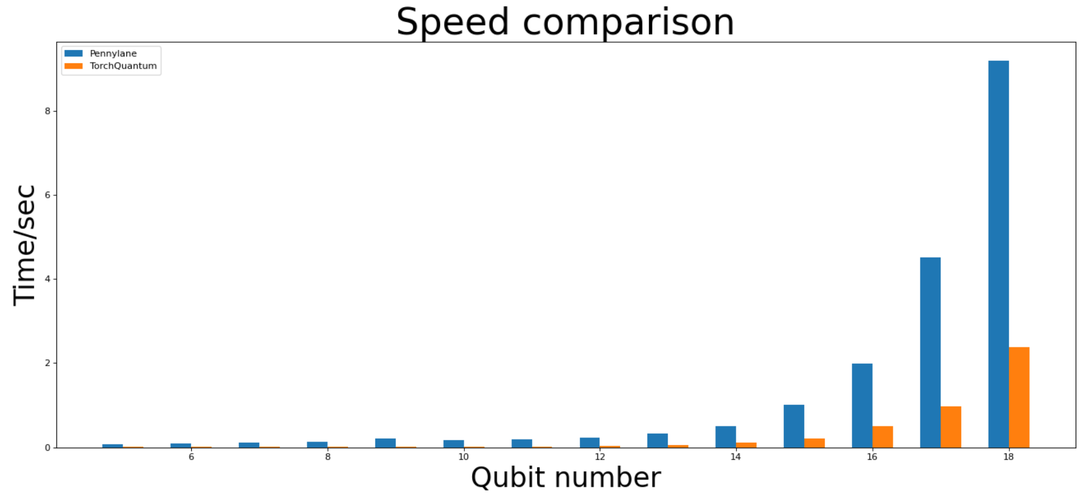
Easy Deployment on Real Quantum Machines

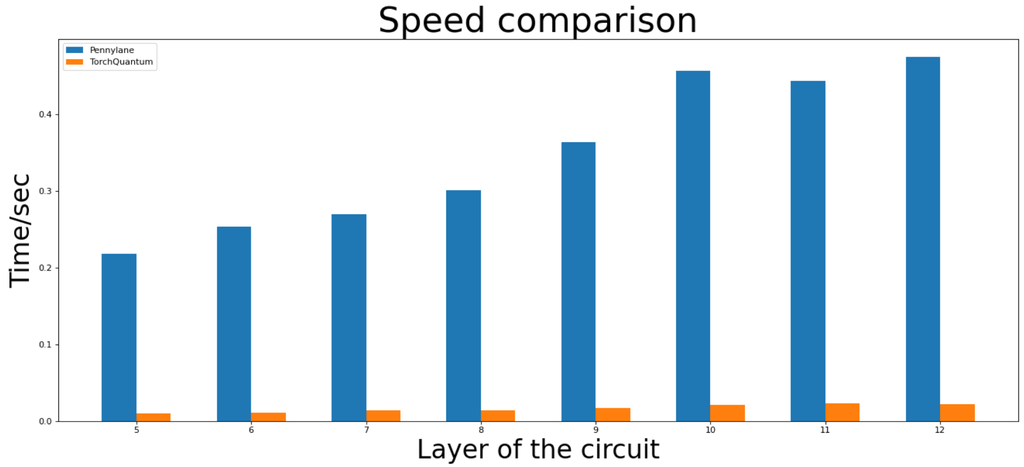
- Convert the tq.QuantumModule to other frameworks such as Qiskit
- `from torchquantum.plugins.qiskit_plugin import tq2qiskit`
- `from torchquantum.plugins.qiskit_processor import QiskitProcessor`
- `processor = QiskitProcessor(use_real_qc=False, max_jobs=1)`
- `circ = tq2qiskit(q_dev, model)`
- `circ.measure_all()`
- `res = processor.process_ready_circs(q_dev, [circ])`

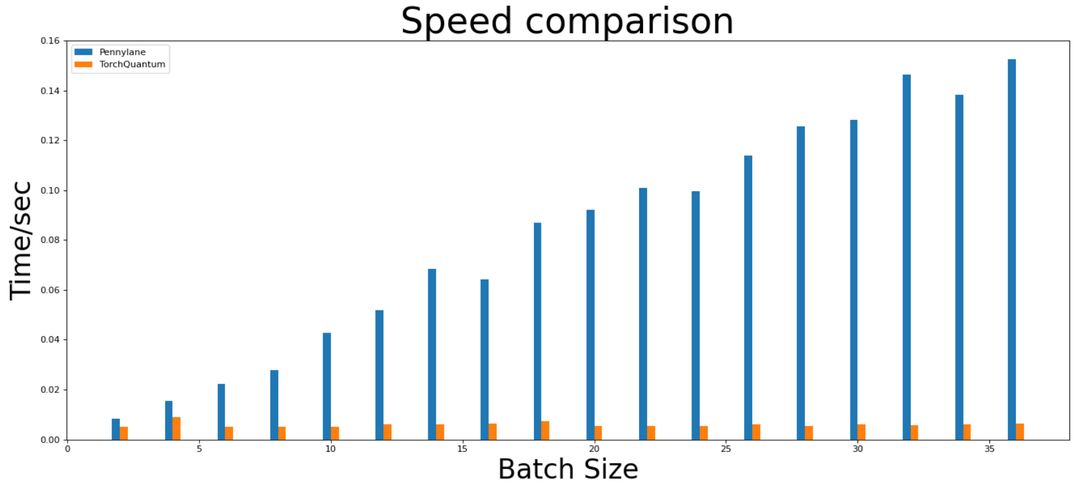
Compare the Speed of TQ and Pennylane



https://pennylane.ai/qml/_images/qcircuit.jpeg







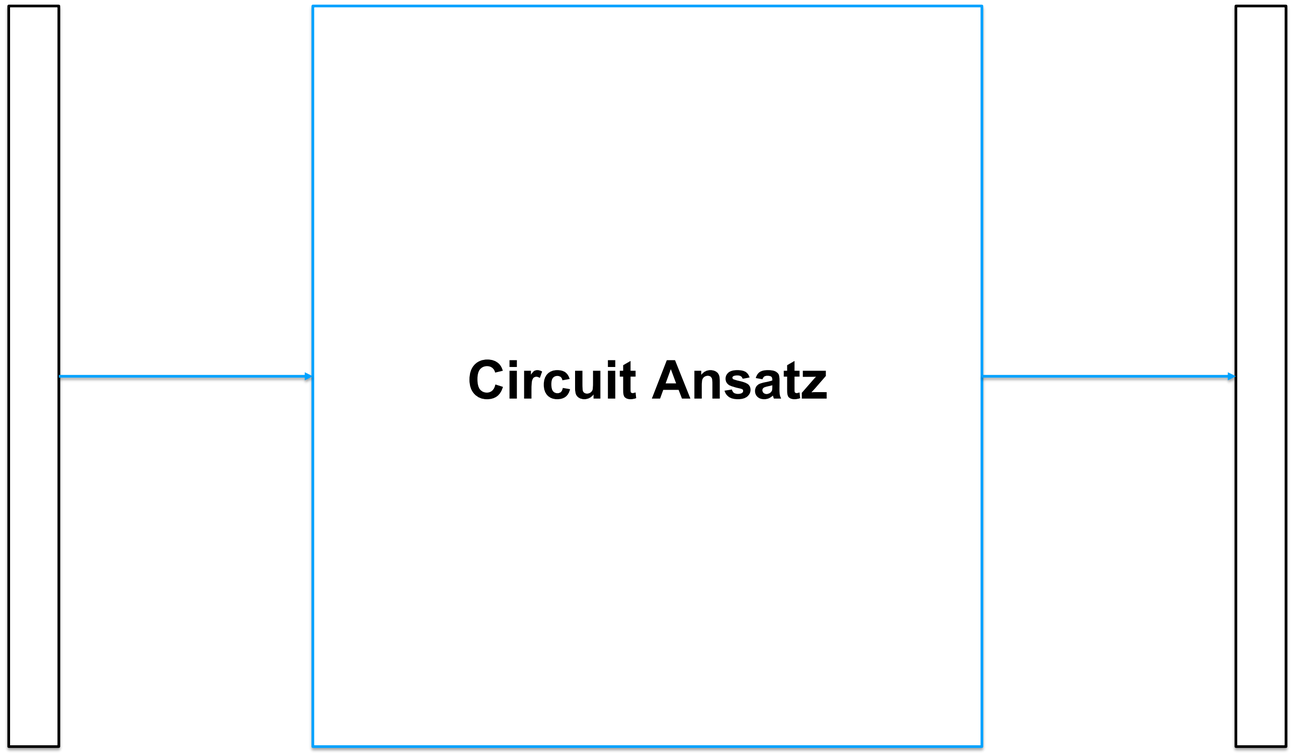
Torch Quantum

Materials at: https://torchquantum.org

MIT HAN LAB

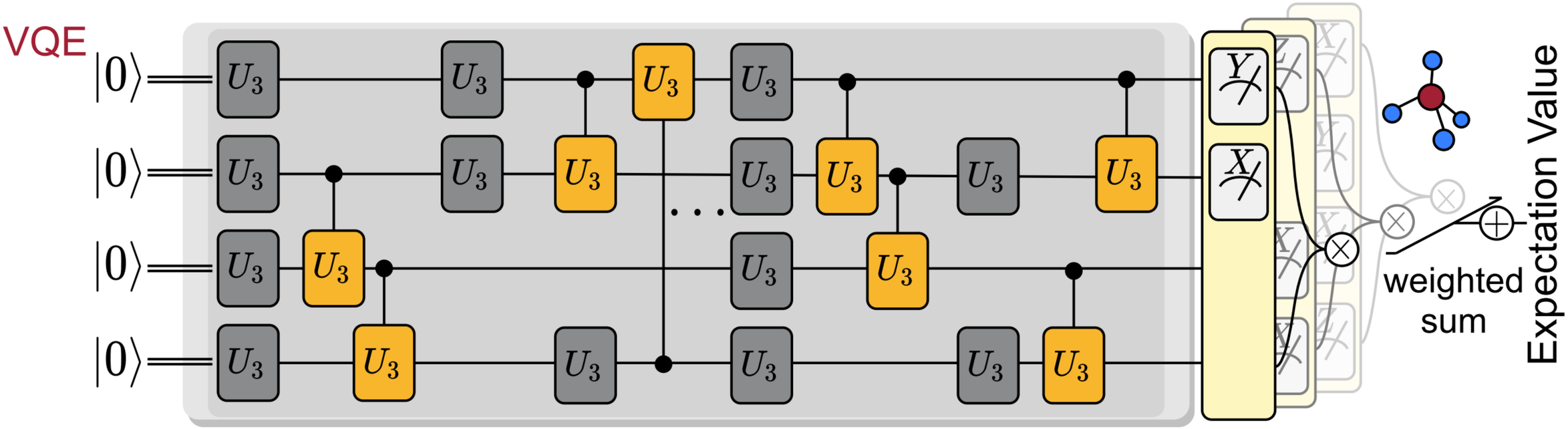
State Preparation with Parameterized Circuit

- Use parameterized circuit to prepare initial quantum states



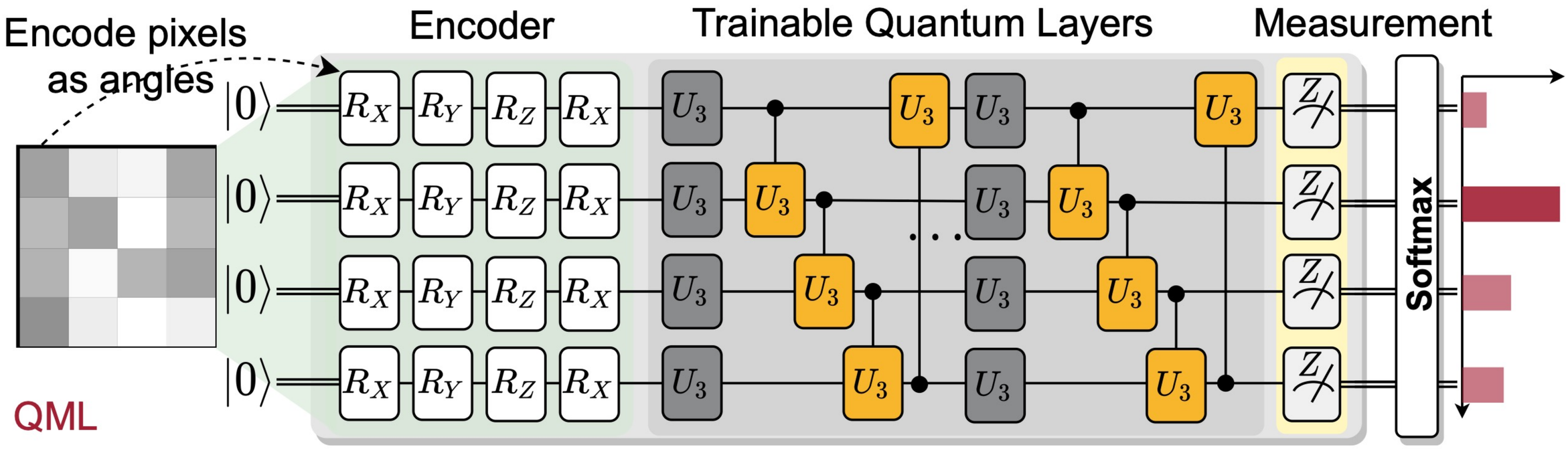
Variational Quantum Eigensolver

- VQE: Finds the ground state energy of molecule Hamiltonian

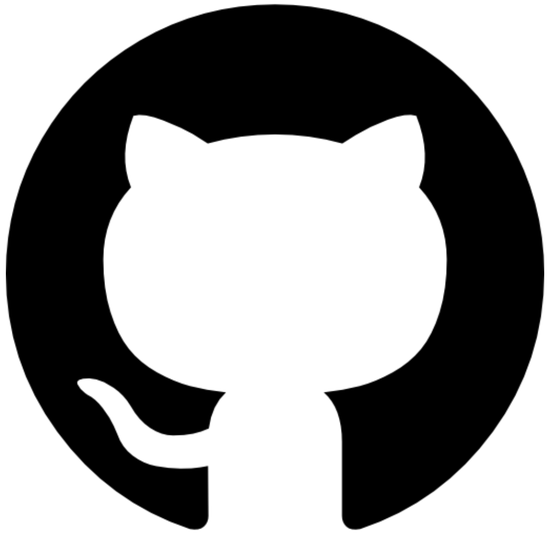


Quantum Neural Networks for MNIST Classification

- Encode the classical values using phase encoding
- Then trainable quantum layers
- Finally measurement layers



Hands-On Section
2.1 TQ Basic Operations and Examples



Torch Quantum

Materials at: https

TorchQuantum Tutorial Outline

Section 1

Quantum Basics and TorchQuantum Usage

1.1 Introduction to Quantum Computing (Prof. Yongshan Ding)

1.2 TorchQuantum API and examples (Hanrui Wang)

Section 2

Use TorchQuantum on Gate Level Optimization

2.1 QuantumNAS Ansatz Search and On-Chip Training (Hanrui Wang)

2.2 Variational Quantum Circuit Compression (Prof. Weiwen Jiang)

2.3 Various VQA Gradient Estimation Methods (Dantong Li)

Section 3

Use TorchQuantum on Pulse Level Optimization

3.1 Intro to Pulse Level Quantum Control (Zhiding Liang)

3.2 Variational Pulse Learning & Optimal Control (Jinglei Cheng)

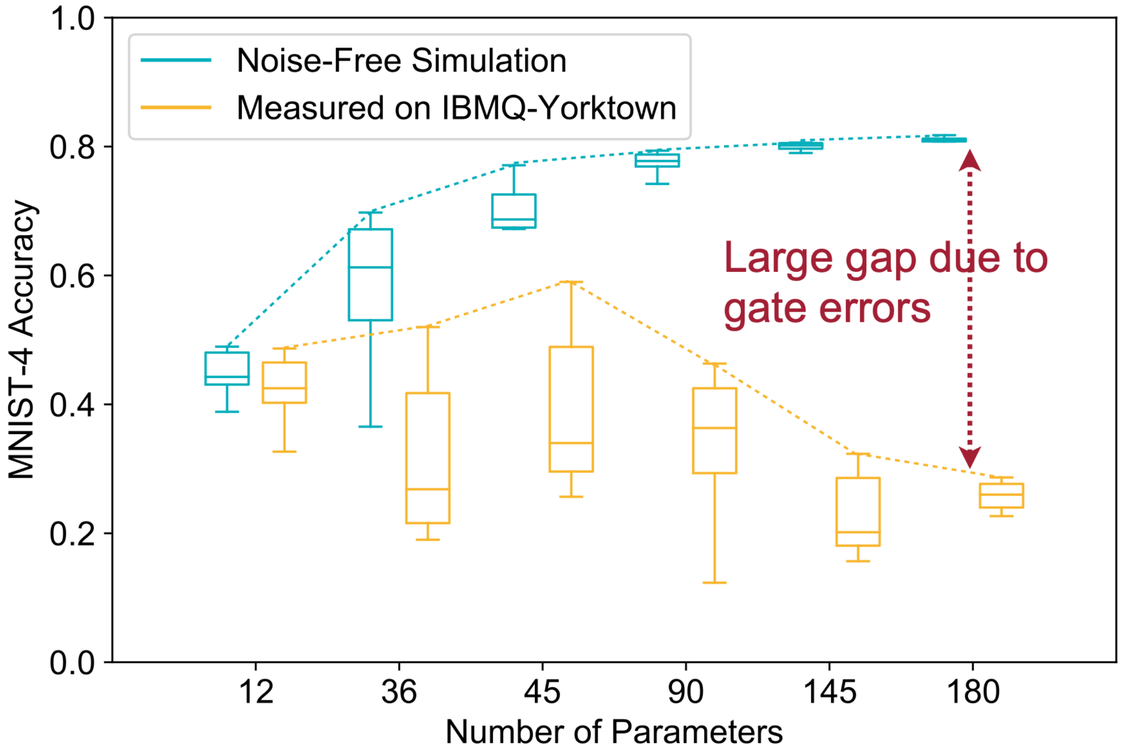
Torch Quantum

Materials at: https://torchquantum.org

MIT HAN LAB

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



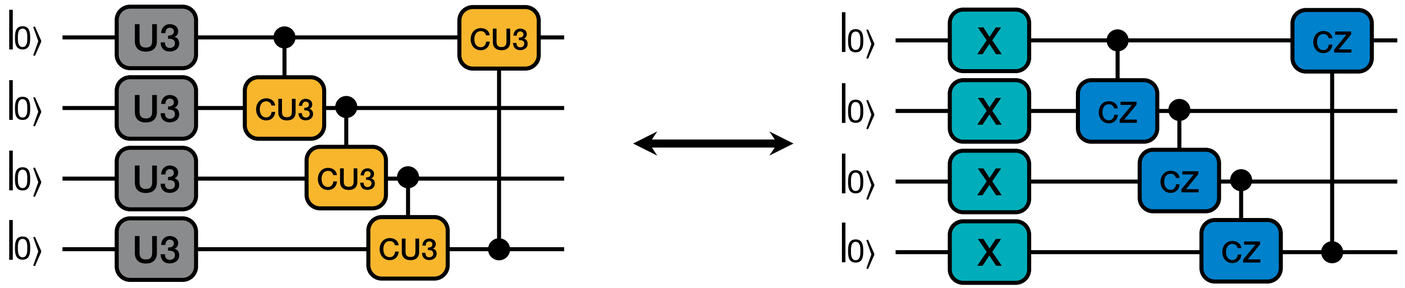
Torch Quantum

Materials at: https://torchquantum.org

MIT HAN LAB

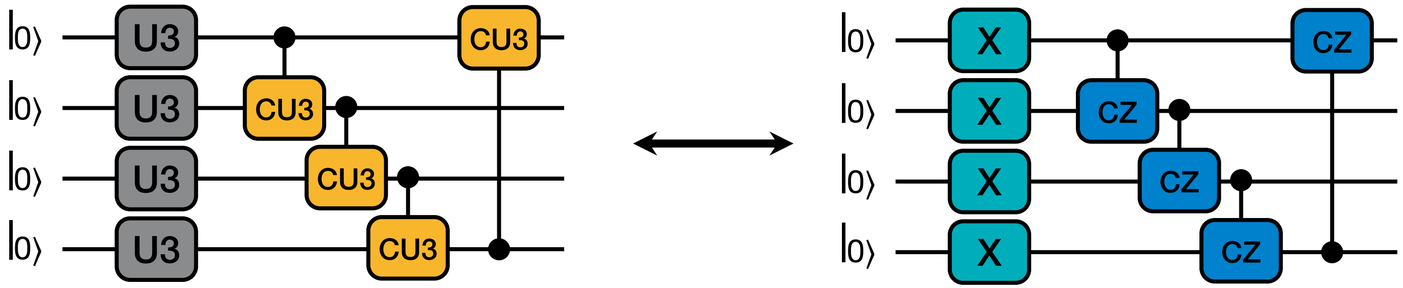
Challenges of PQC — Large Design Space

- Large design space for circuit architecture
  - Type of gates

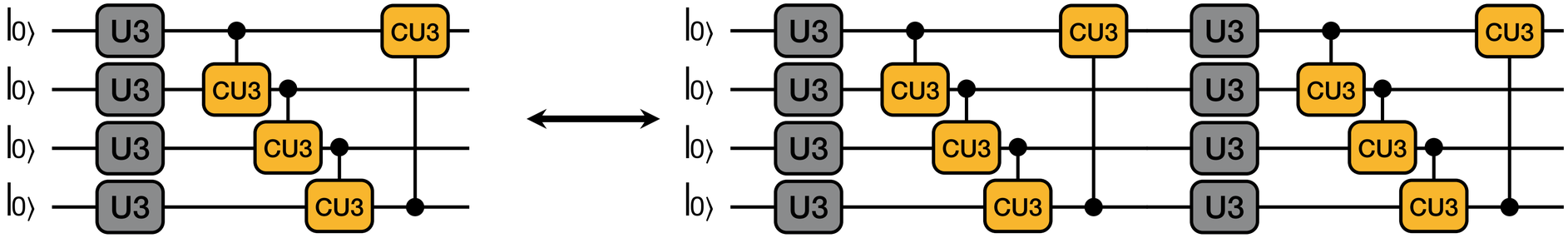


Challenges of PQC — Large Design Space

- Large design space for circuit architecture
  - Type of gates

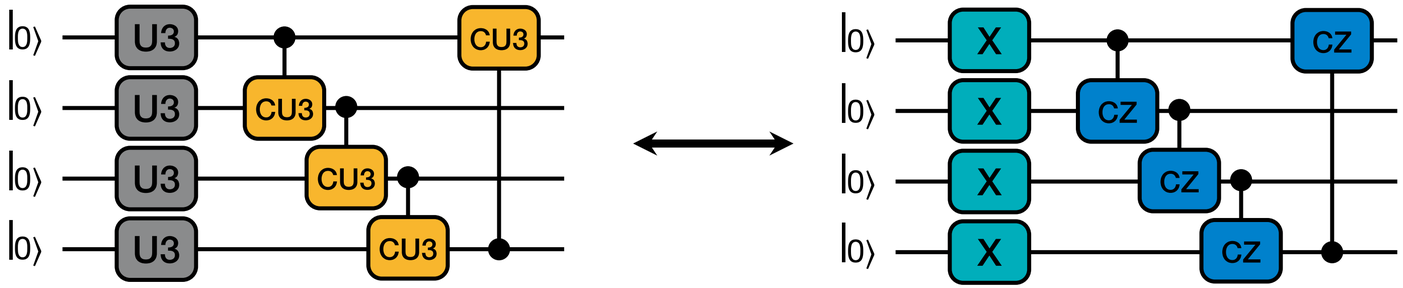


  - Number of gates

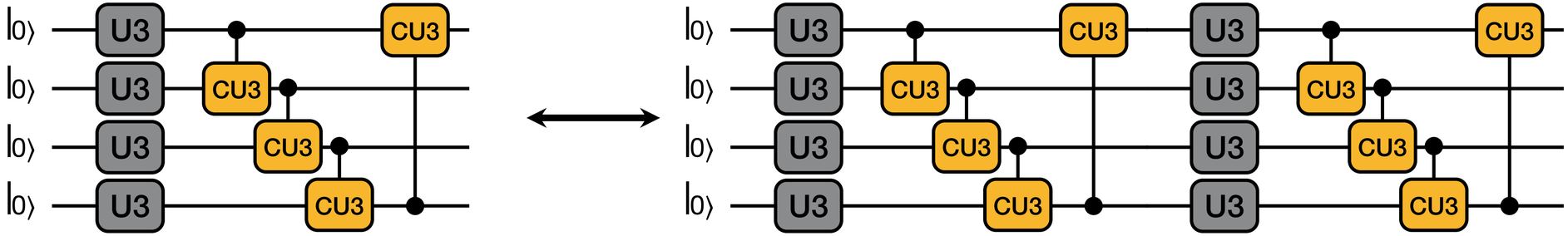


Challenges of PQC — Large Design Space

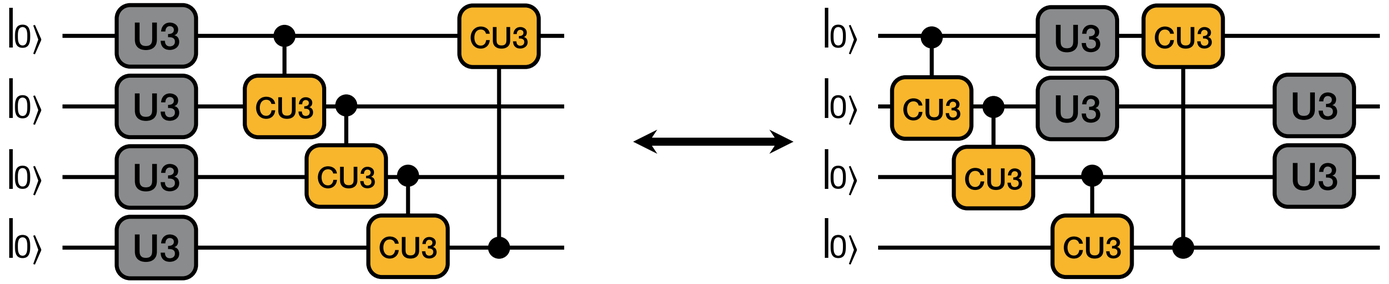
- Large design space for circuit architecture
  - Type of gates



  - Number of gates



  - Position of gates



Torch Quantum

Materials at: https://torchquantum.org

MIT HAN LAB

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

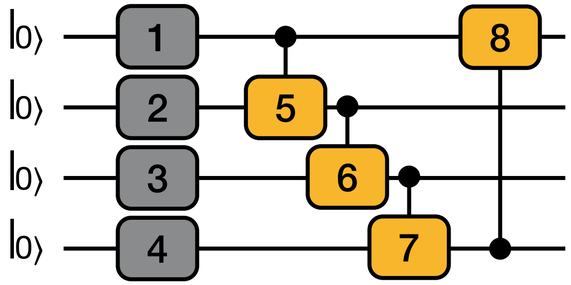
SuperCircuit & SubCircuit

- 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

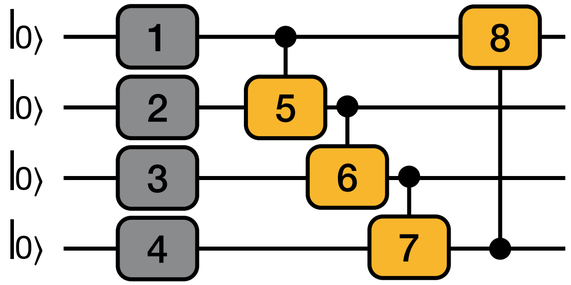
- SuperCircuit: the circuit with the **largest** number of gates in the design space
  - Example: SuperCircuit in U3+CU3 space



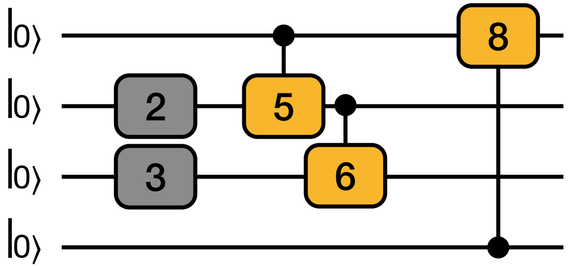
SuperCircuit & SubCircuit

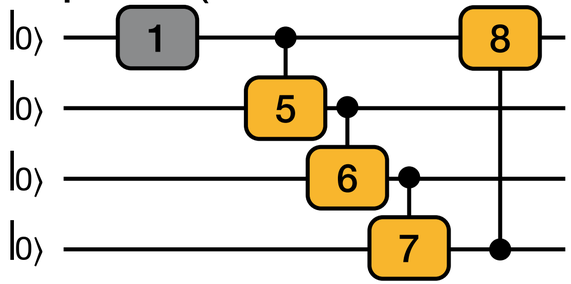
- Firstly construct a design space. For example, a design space of maximum 4 U3 in the first layer and 4 CU3 gates in the second layer

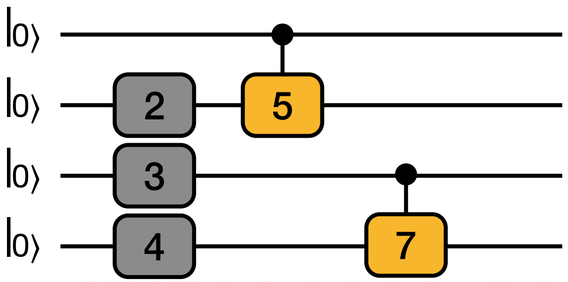
- SuperCircuit: the circuit with the **largest** number of gates in the design space
  - Example: SuperCircuit in U3+CU3 space



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







Torch Quantum

Materials at: https://torchquantum.org

MIT HAN LAB

SuperCircuit Construction

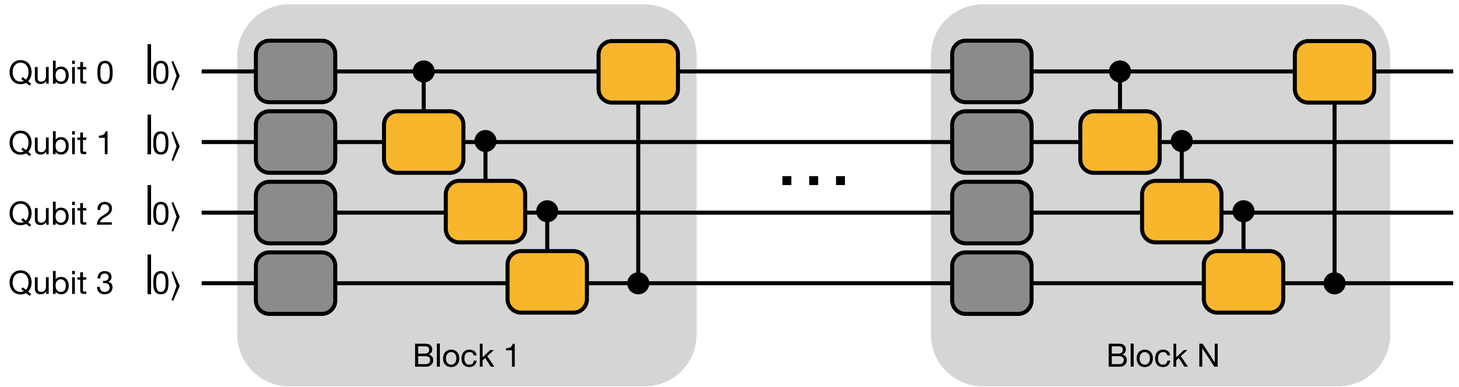
- Why use a SuperCircuit?
  - Enables **efficient** search of architecture candidates without training each
  - SubCircuit inherits parameters from SuperCircuit
  - With **inherited** parameters, we find some good SubCircuits, we find that they are **also good SubCircuits** with parameters **trained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

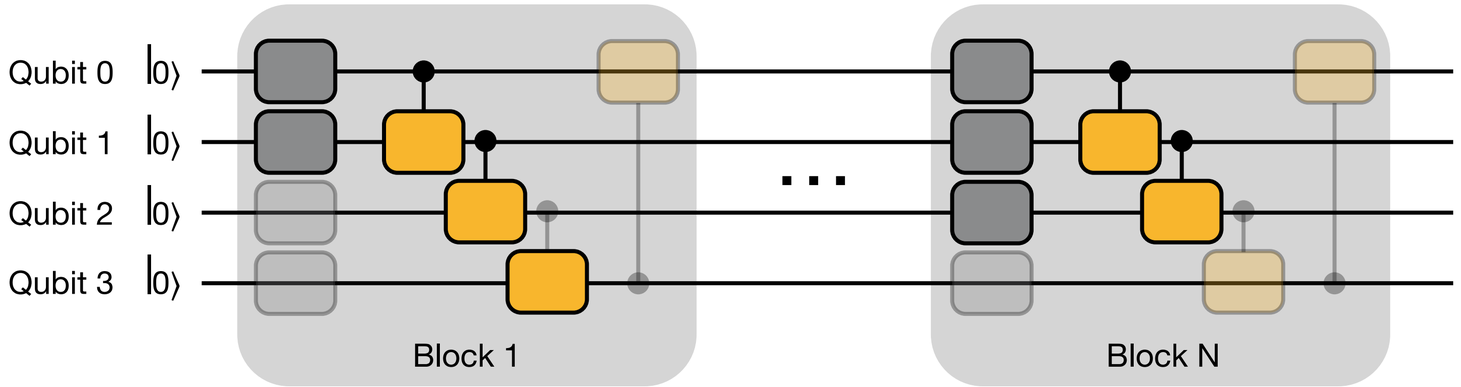
Train SuperCircuit for Multiple Steps

- In one SuperCircuit Training step: Sample and Train



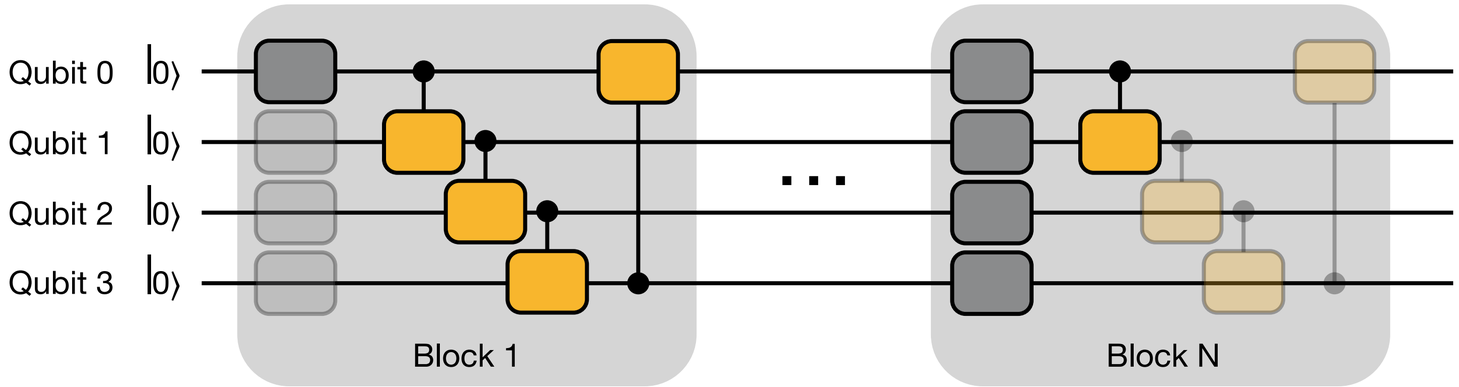
Train SuperCircuit for Multiple Steps

- In one SuperCircuit Training step: Sample and Train



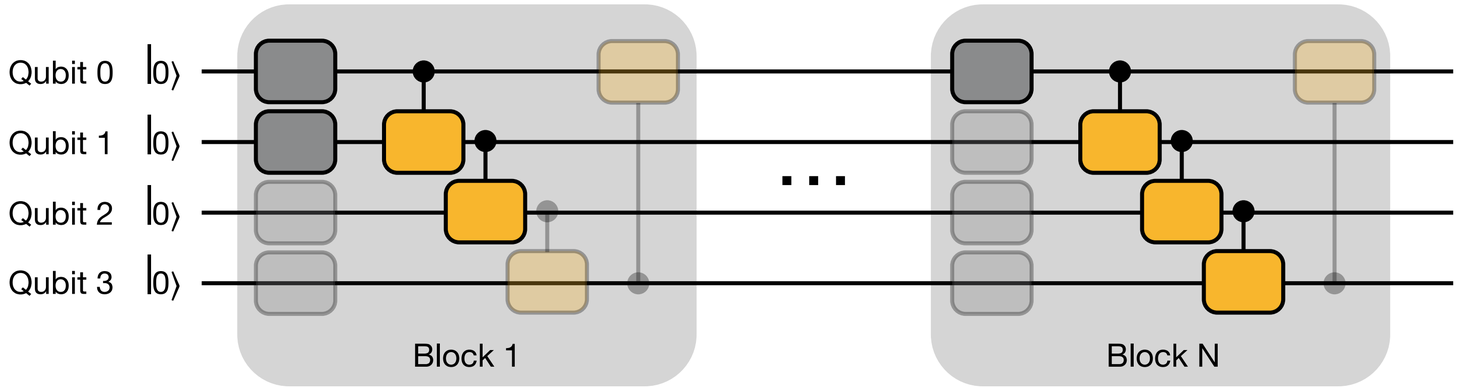
Train SuperCircuit for Multiple Steps

- In one SuperCircuit Training step: Sample and Train



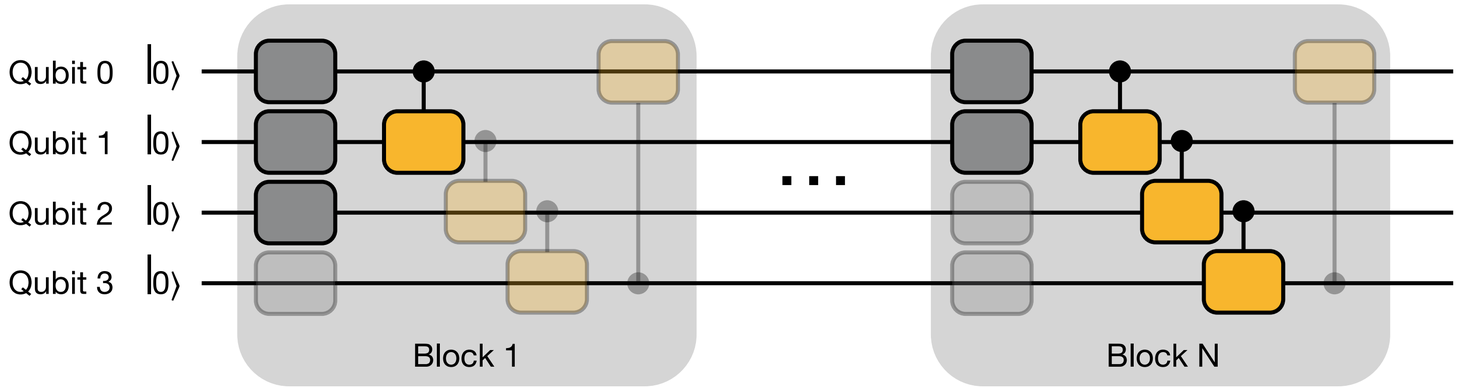
Train SuperCircuit for Multiple Steps

- In one SuperCircuit Training step: Sample and Train



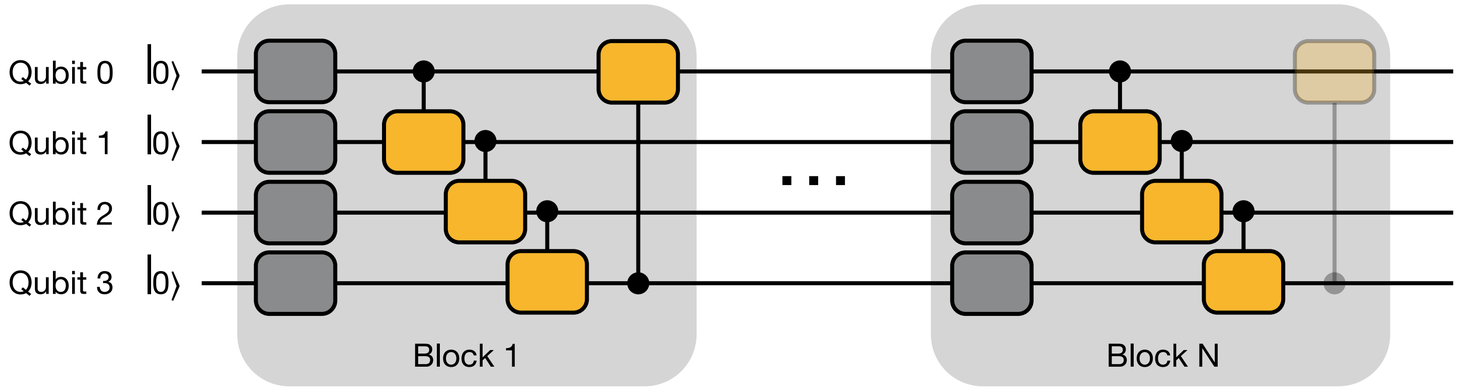
Train SuperCircuit for Multiple Steps

- In one SuperCircuit Training step: Sample and Train



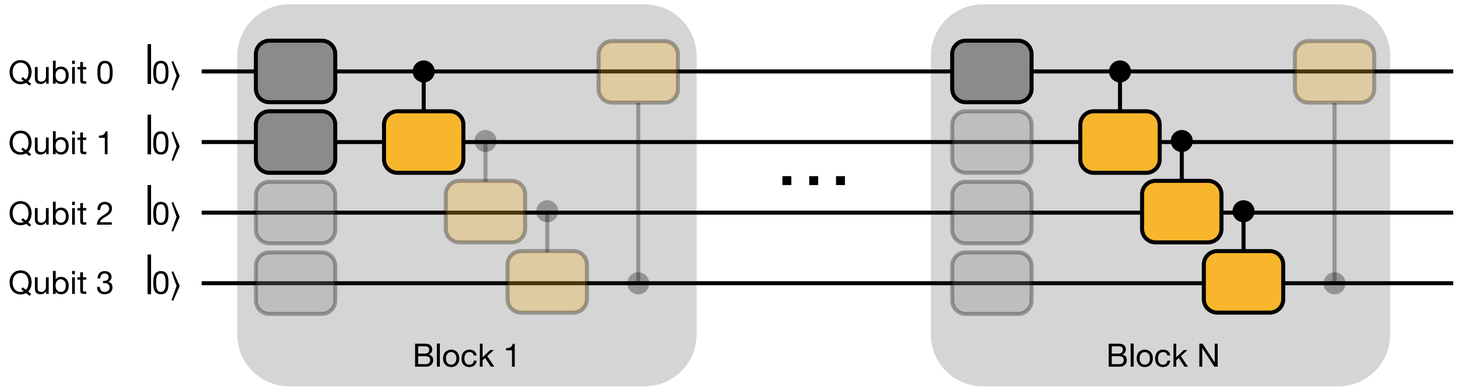
Train SuperCircuit for Multiple Steps

- In one SuperCircuit Training step: Sample and Train



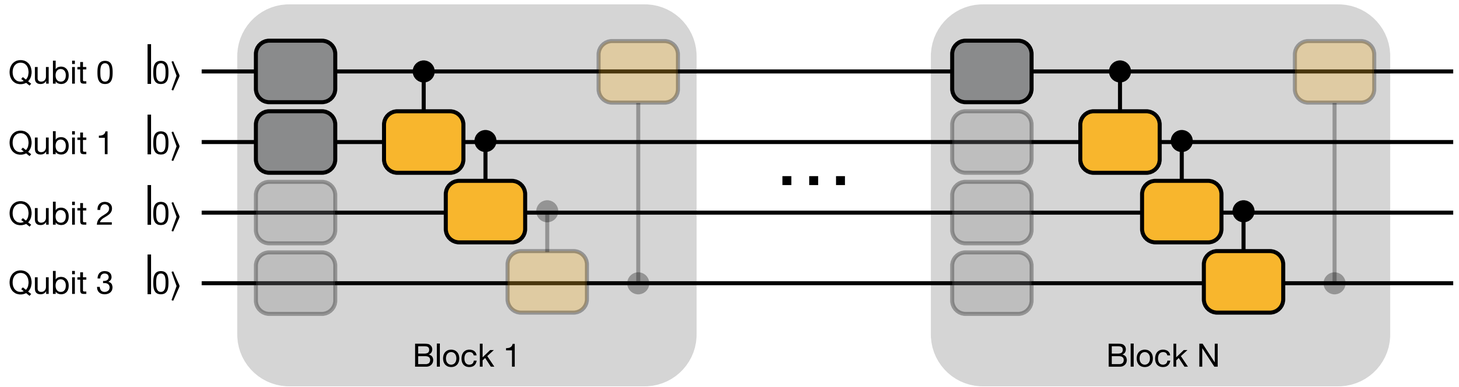
Train SuperCircuit for Multiple Steps

- In one SuperCircuit Training step: Sample and Train



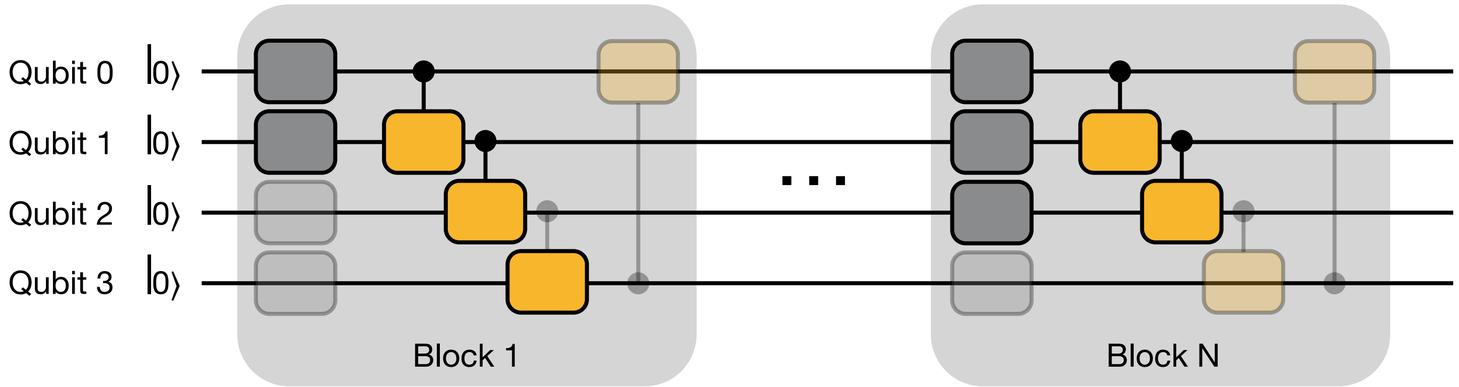
Train SuperCircuit for Multiple Steps

- In one SuperCircuit Training step: Sample and Train



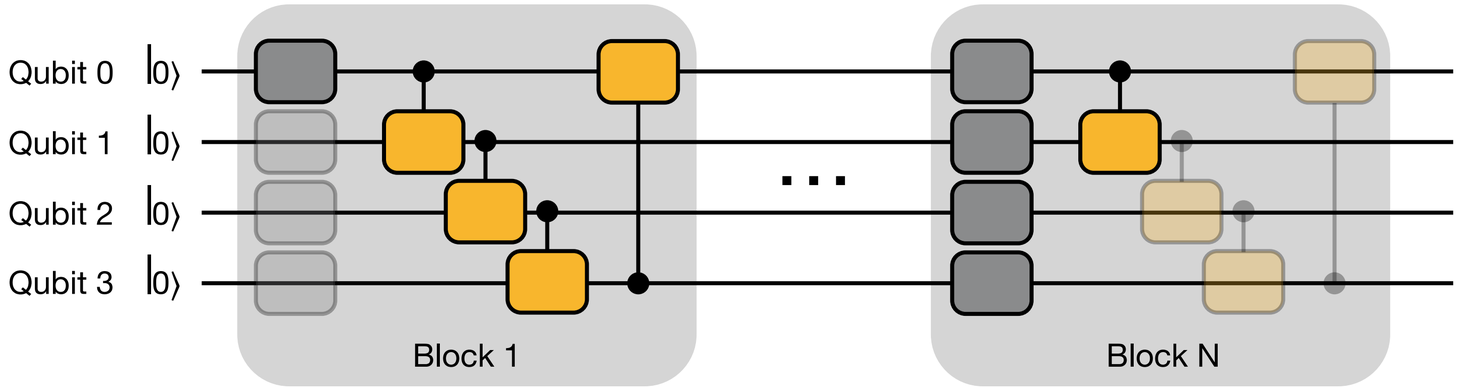
Train SuperCircuit for Multiple Steps

- In one SuperCircuit Training step: Sample and Train



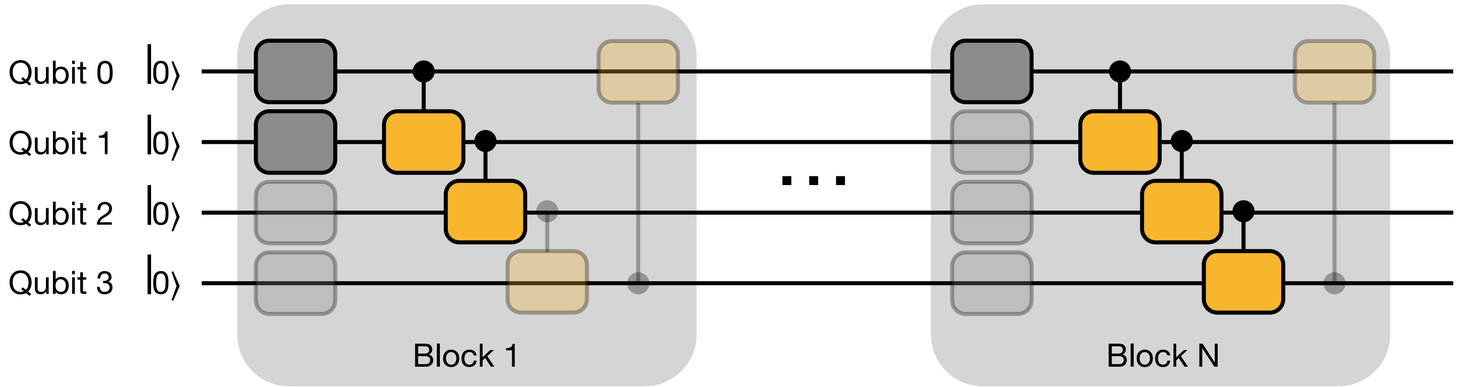
Train SuperCircuit for Multiple Steps

- In one SuperCircuit Training step: Sample and Train



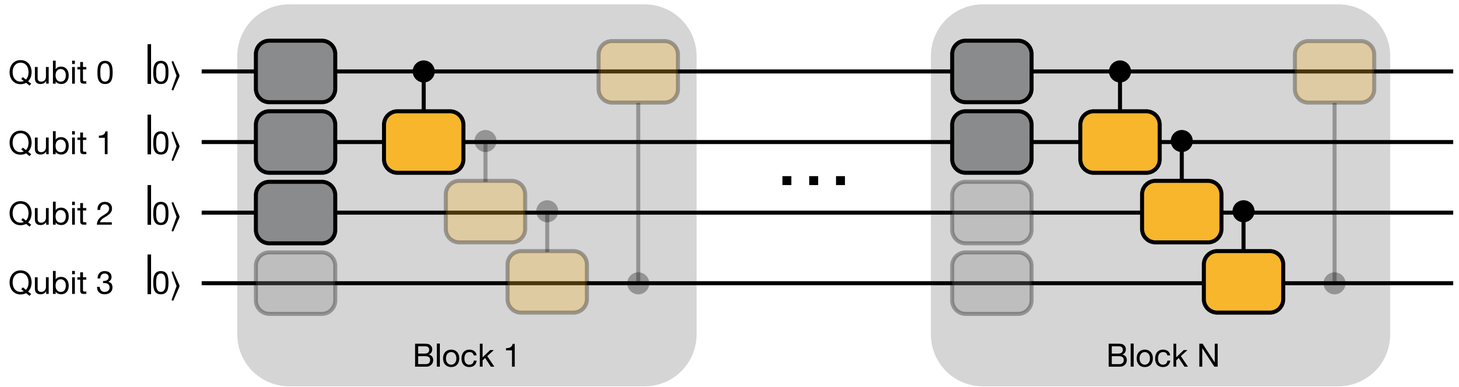
Train SuperCircuit for Multiple Steps

- In one SuperCircuit Training step: Sample and Train



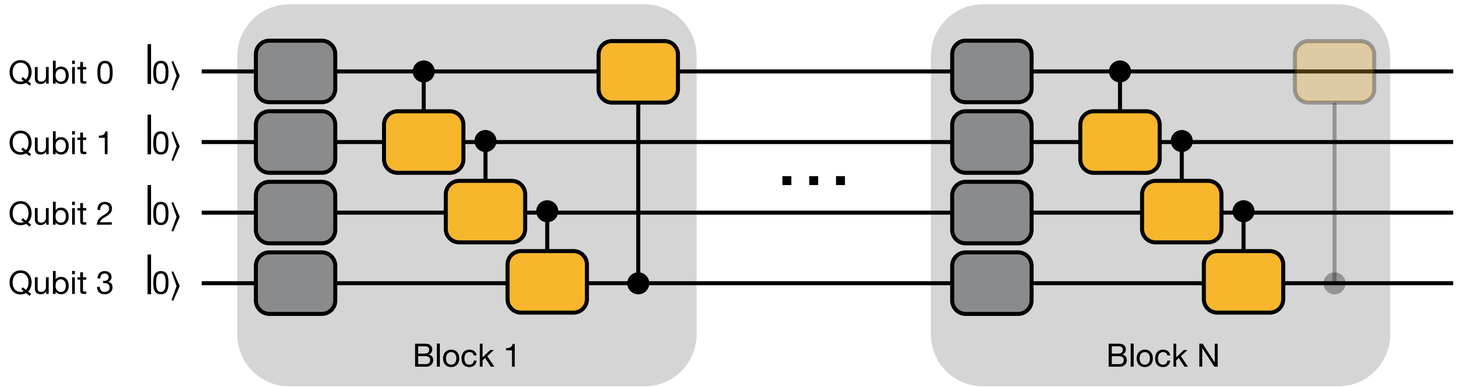
Train SuperCircuit for Multiple Steps

- In one SuperCircuit Training step: Sample and Train



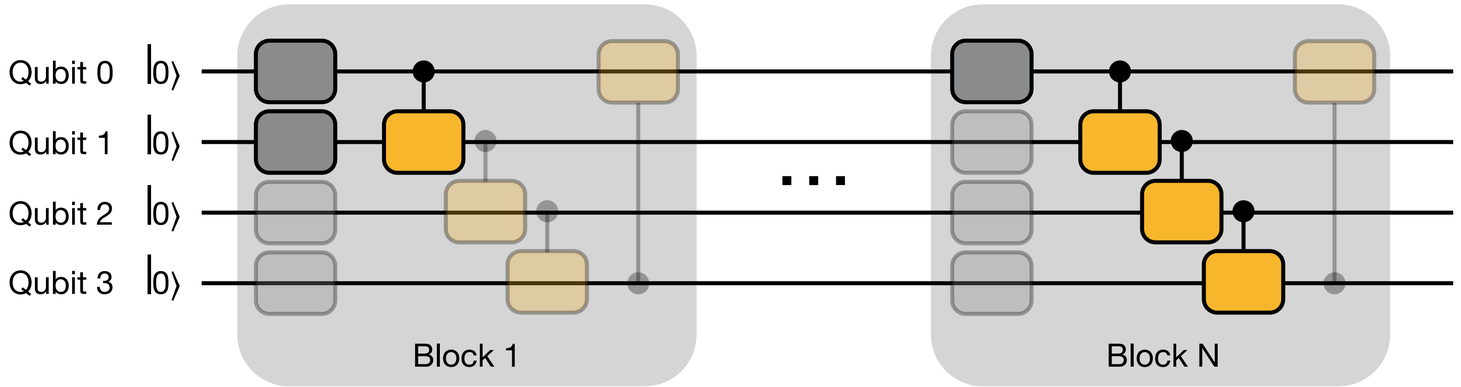
Train SuperCircuit for Multiple Steps

- In one SuperCircuit Training step: Sample and Train



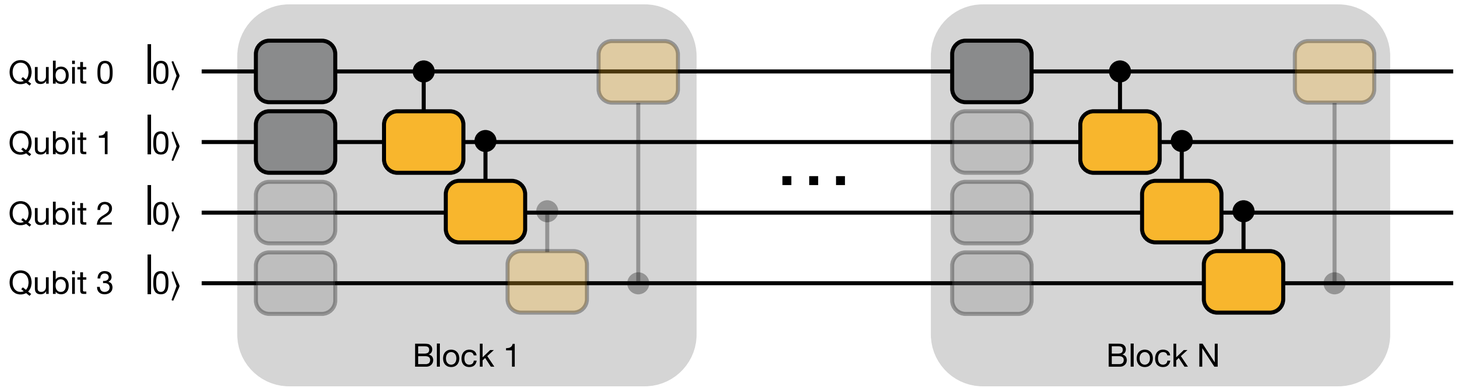
Train SuperCircuit for Multiple Steps

- In one SuperCircuit Training step: Sample and Train

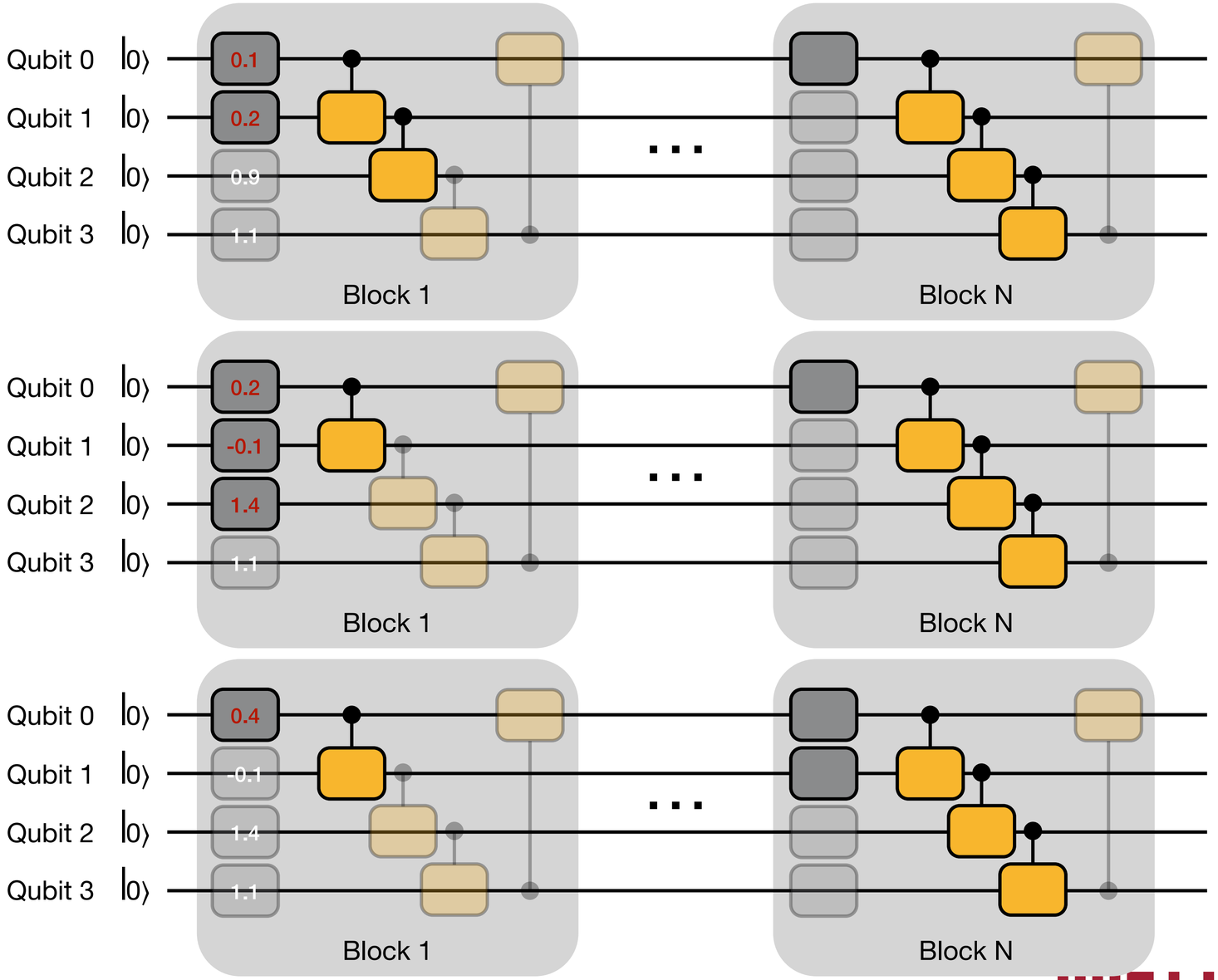


Train SuperCircuit for Multiple Steps

- In one SuperCircuit Training step: Sample and Train



Train SuperCircuit for Multiple Steps



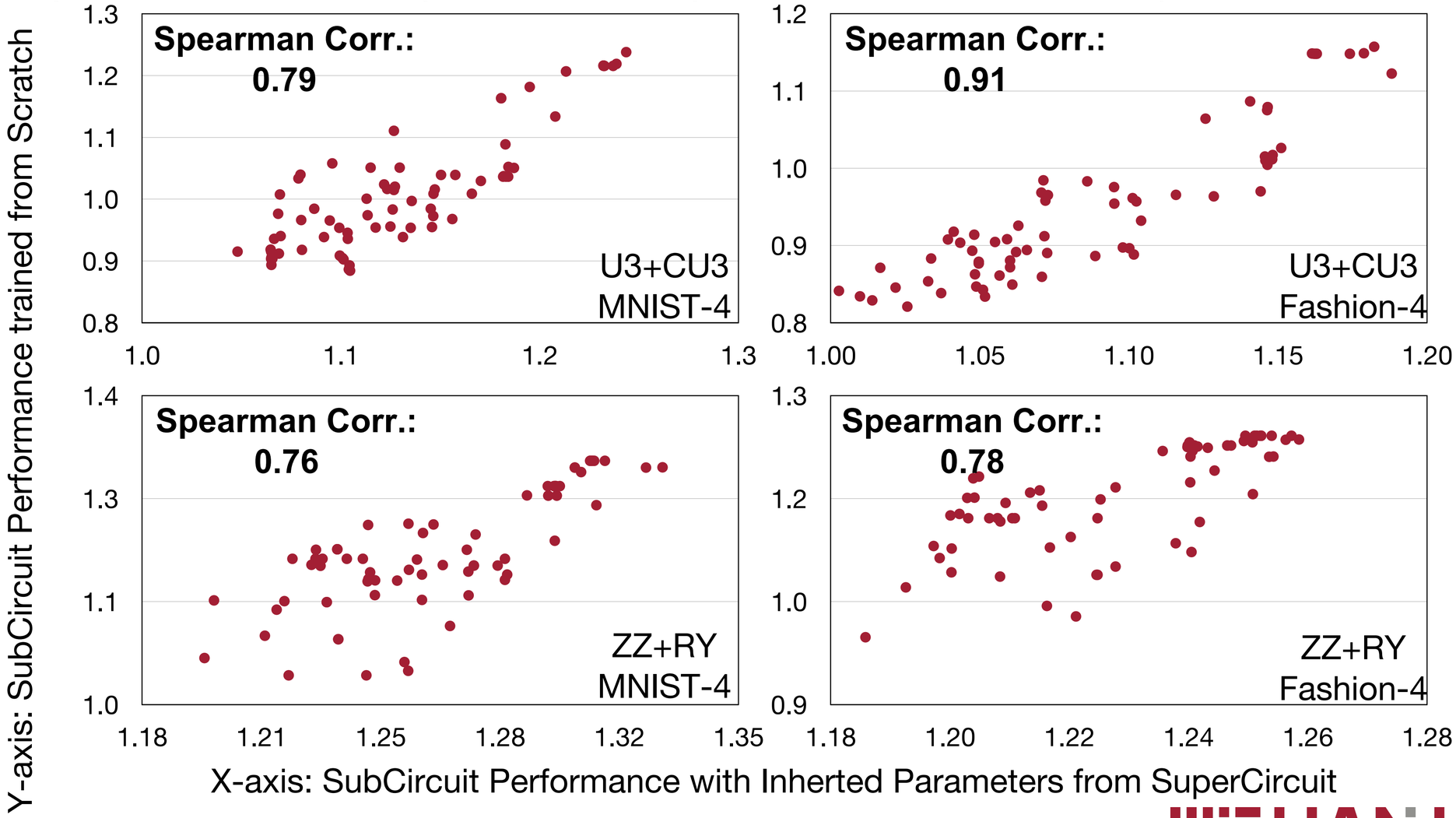
Torch Quantum

Materials at: https://torchquantum.org

MIT HAN LAB

How Reliable is the SuperCircuit?

- Inherited parameters from SuperCircuit can provide accurate relative performance



Torch Quantum

Materials at: https://torchquantum.org

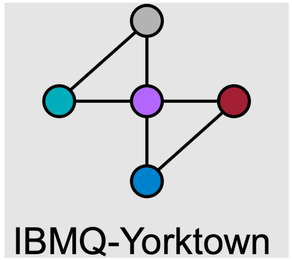
MIT HAN LAB

QuantumNAS

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

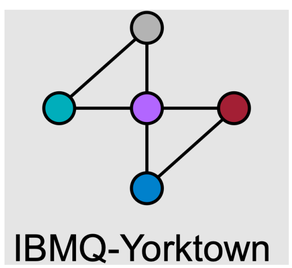
Noise-Adaptive Evolutionary Co-Search

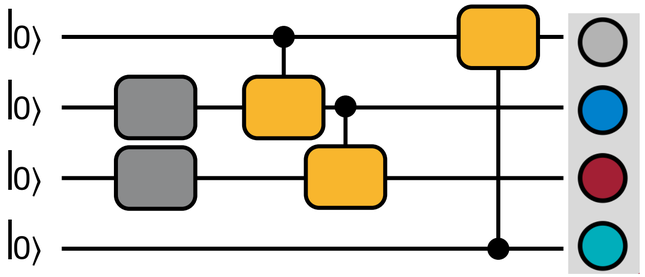
- Search the best SubCircuit and its qubit mapping on target device

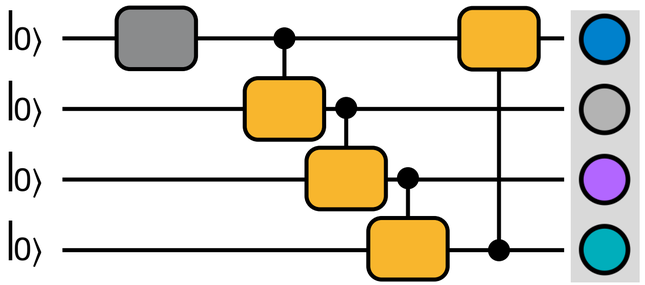


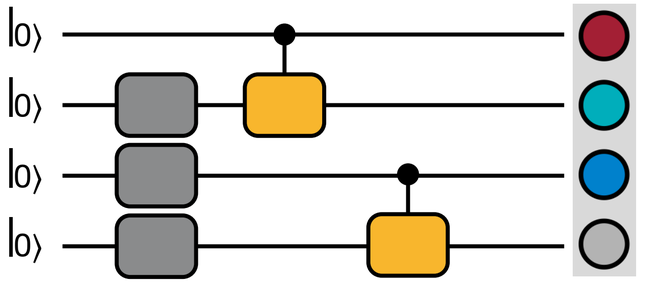
Noise-Adaptive Evolutionary Co-Search

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









Evolutionary Search Engine
Mutation
Crossover

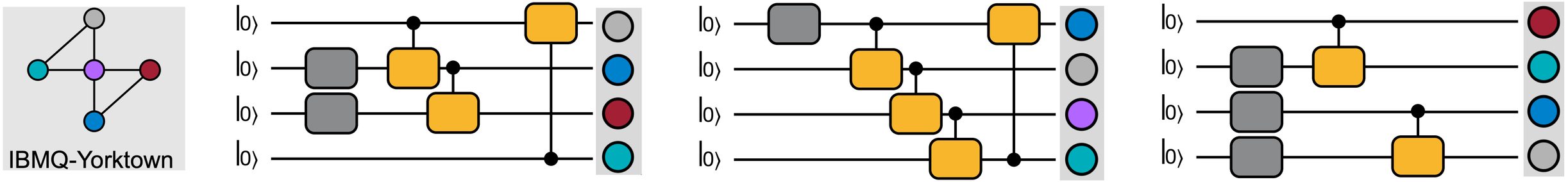
SubCircuit and Qubit Mapping pairs

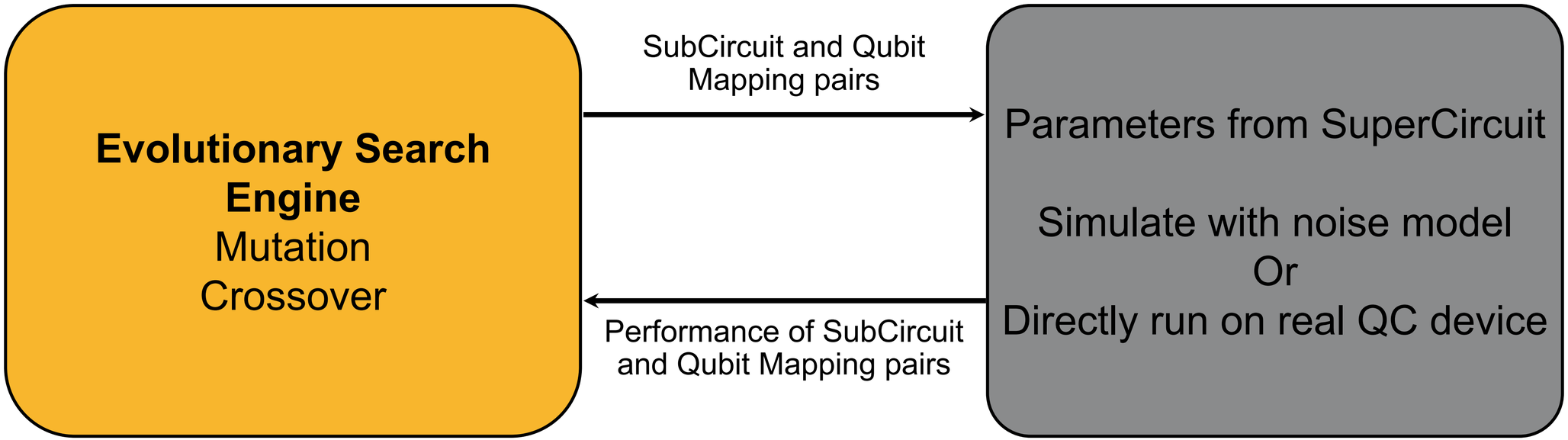
Parameters from SuperCircuit

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





QuantumNAS

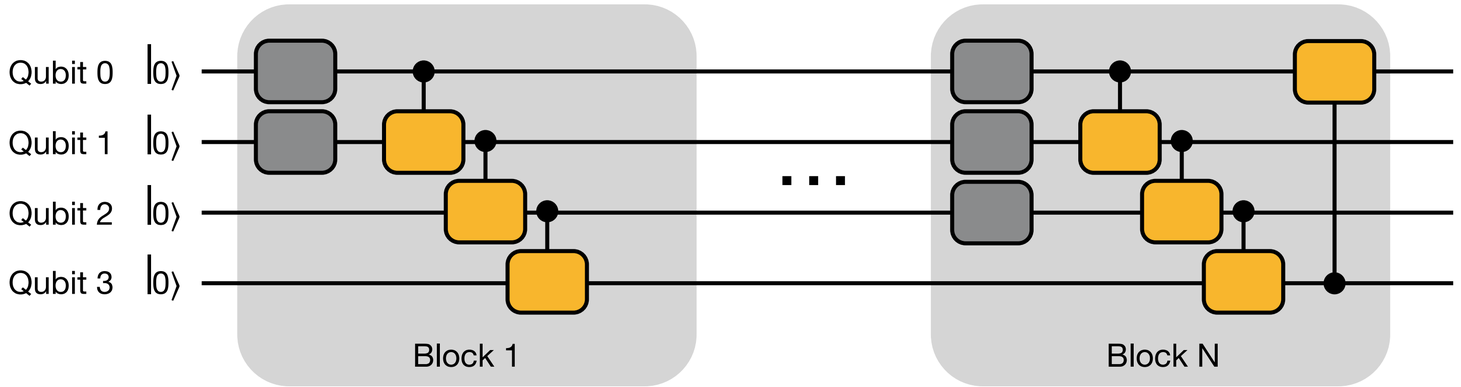
- SuperCircuit Construction and Training
- Noise-Adaptive Evolutionary Co-Search of SubCircuit and Qubit Mapping
- Train the Searched SubCircuit
- Iterative Quantum Gate Pruning

QuantumNAS

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

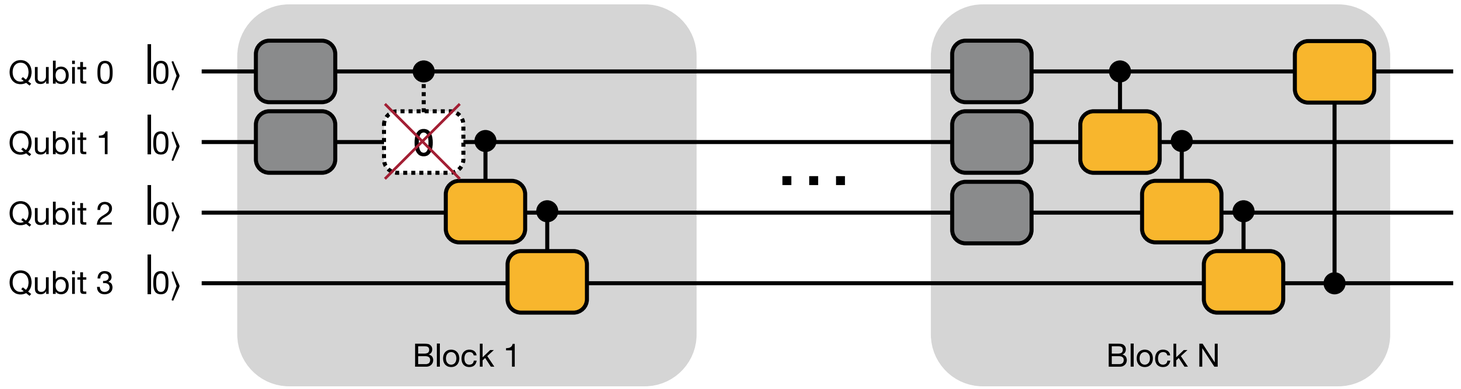
Iterative Pruning

- 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



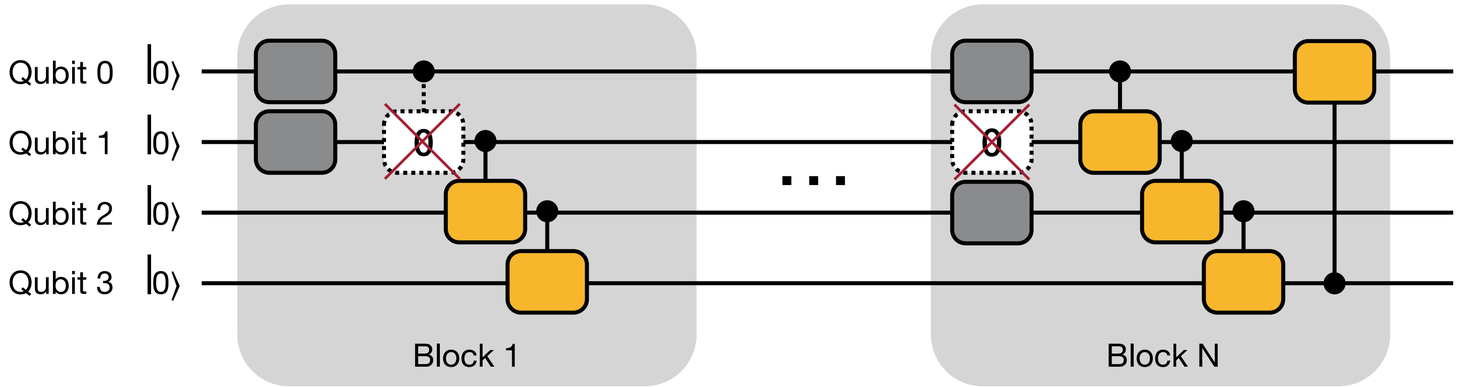
Iterative Pruning

- 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



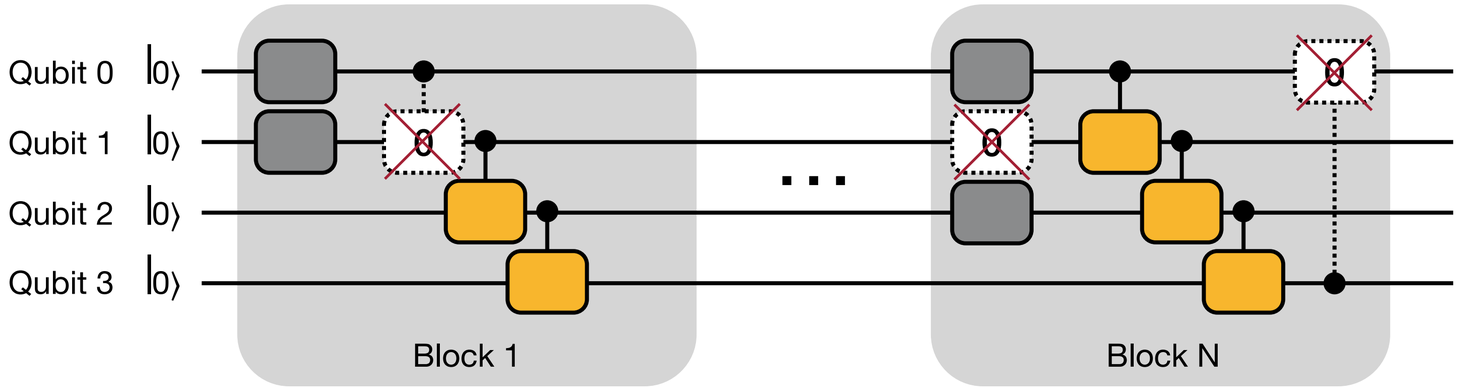
Iterative Pruning

- 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



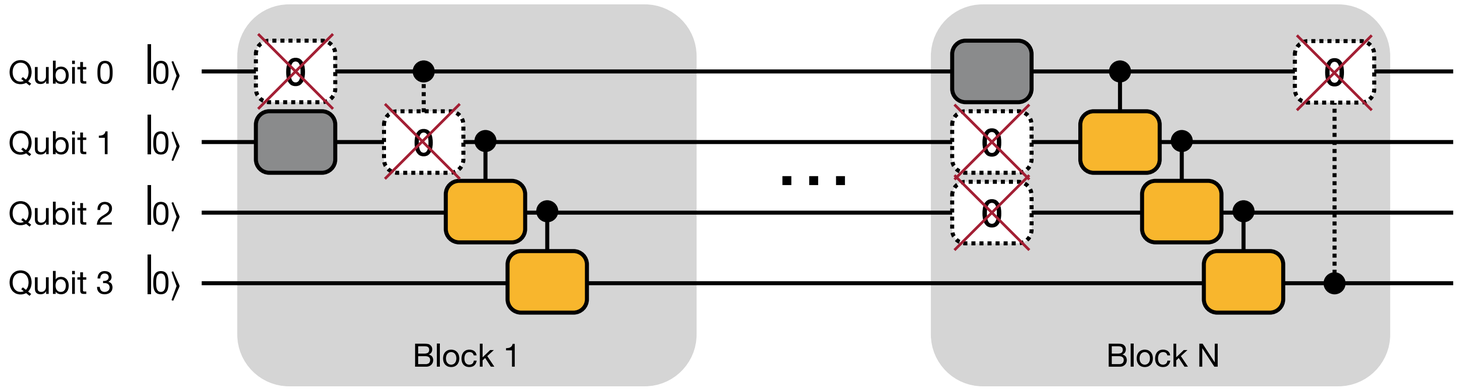
Iterative Pruning

- 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

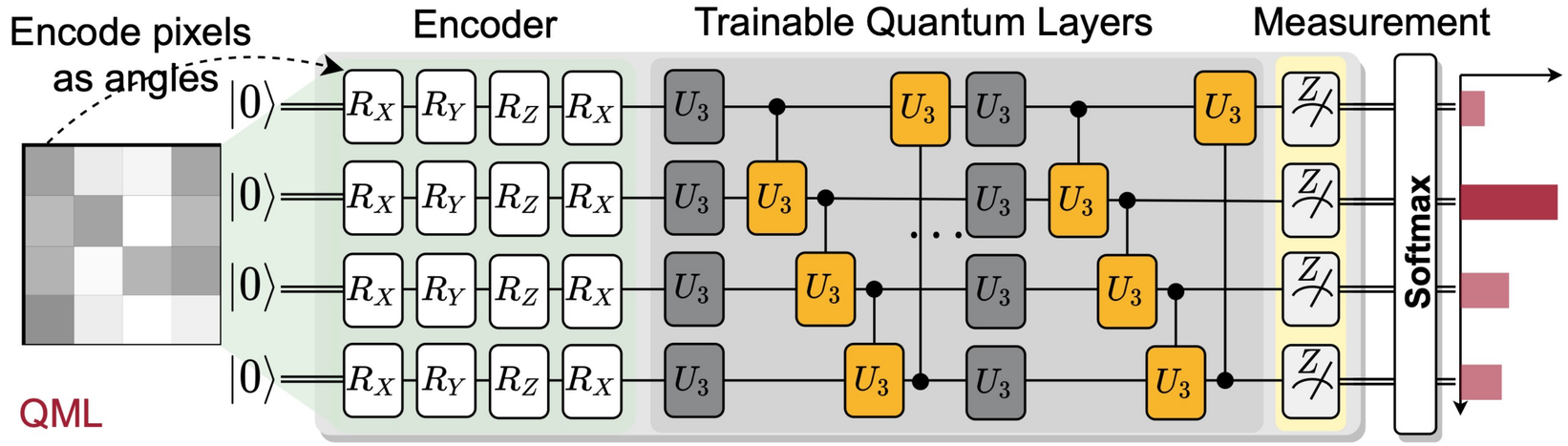


Evaluation Setups: Benchmarks and Devices

- 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

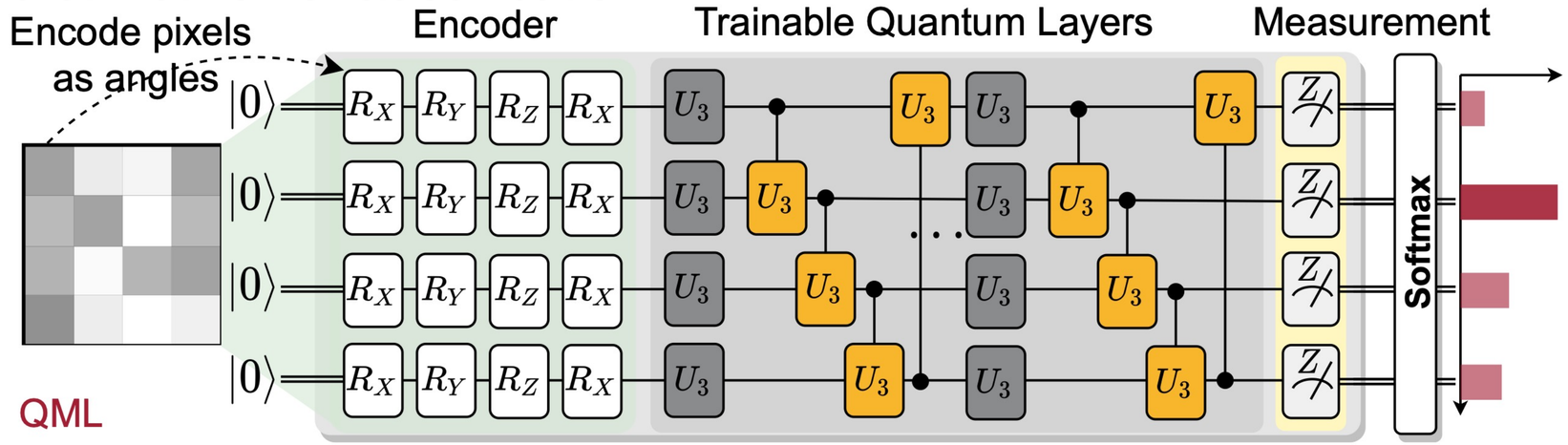
Benchmarks: QNN and VQE

- Quantum Neural Networks: classification

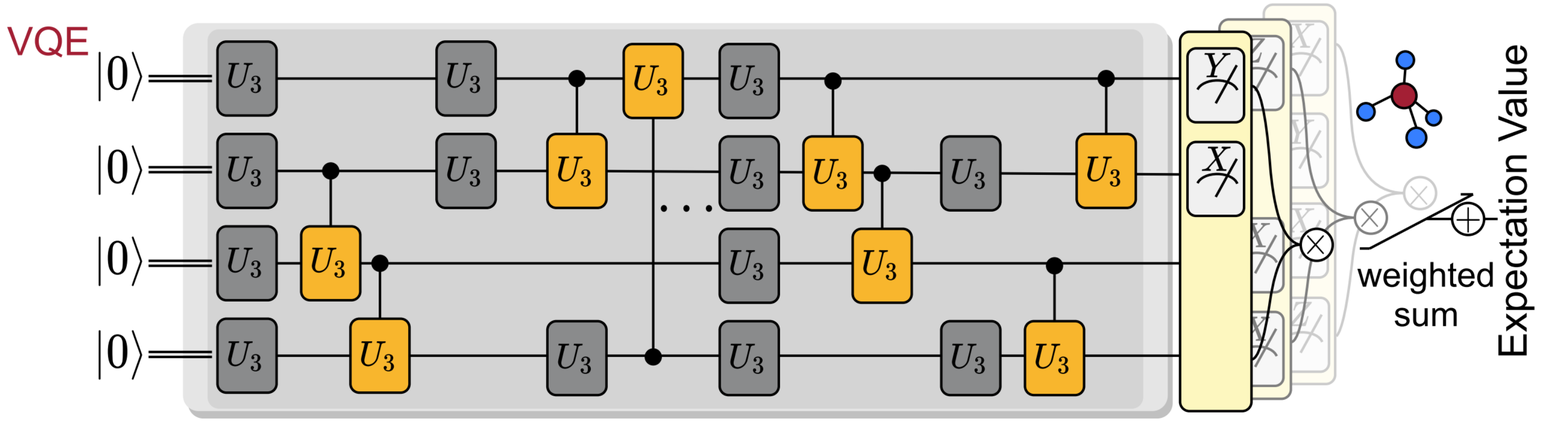


Benchmarks: QNN and VQE

- Quantum Neural Networks: classification



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

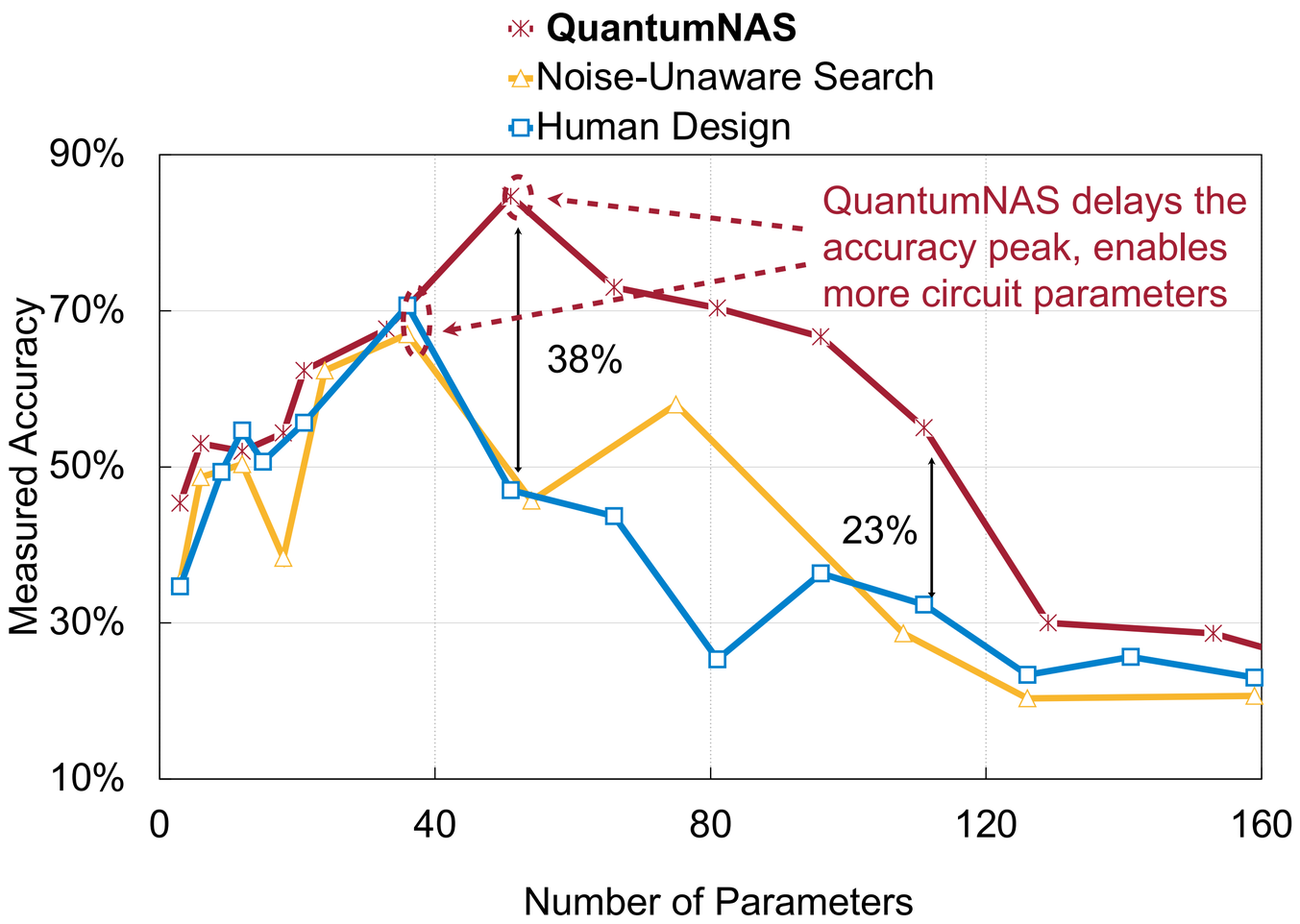


Torch Quantum

Materials at: https://torchquantum.org

QML Results

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



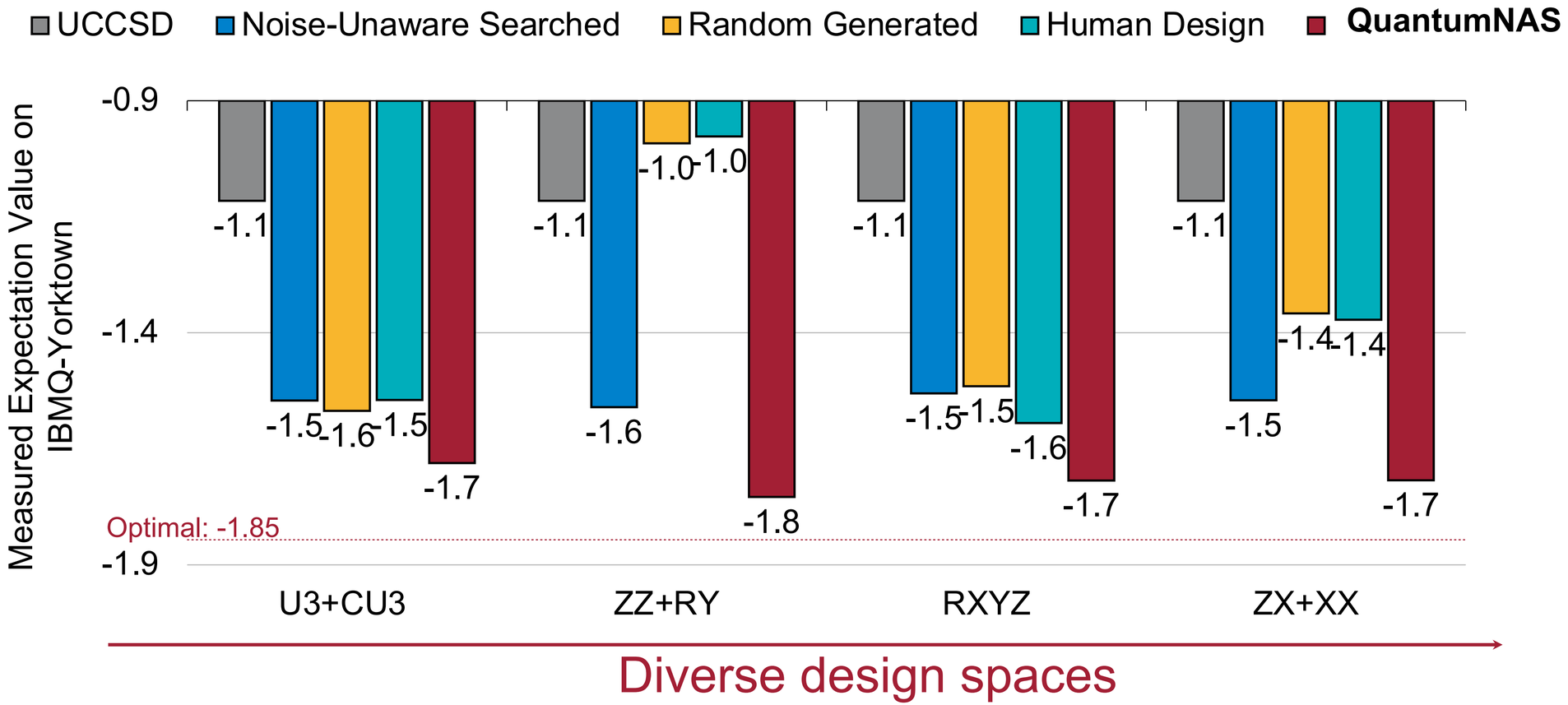
Torch Quantum

Materials at: https://torchquantum.org

MIT HAN LAB

Consistent Improvements on Diverse Design Spaces

- H2 in different design spaces on IBMQ-Yorktown



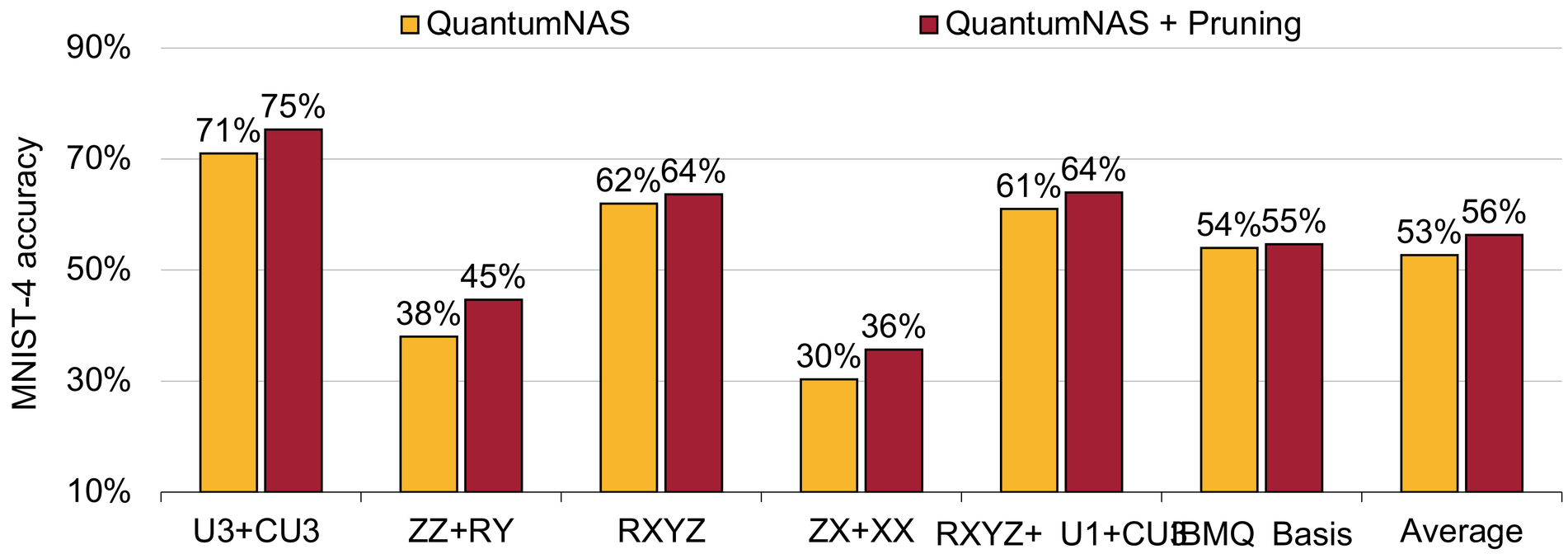
Torch Quantum

Materials at: https://torchquantum.org

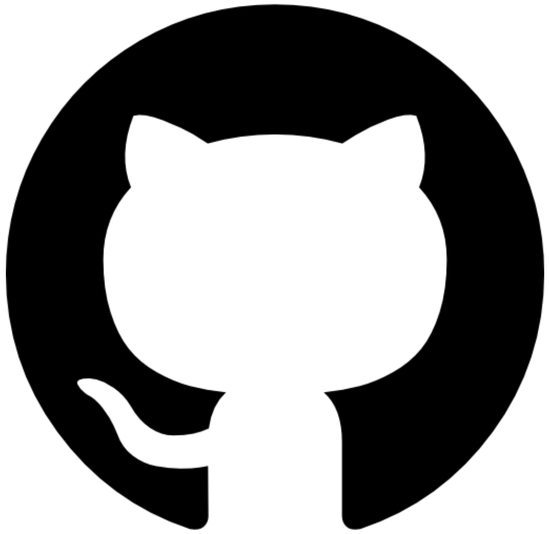
MIT HAN LAB

Effective of Quantum Gate Pruning

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



Hands-On Section
3.1 QuantumNAS



Torch Quantum

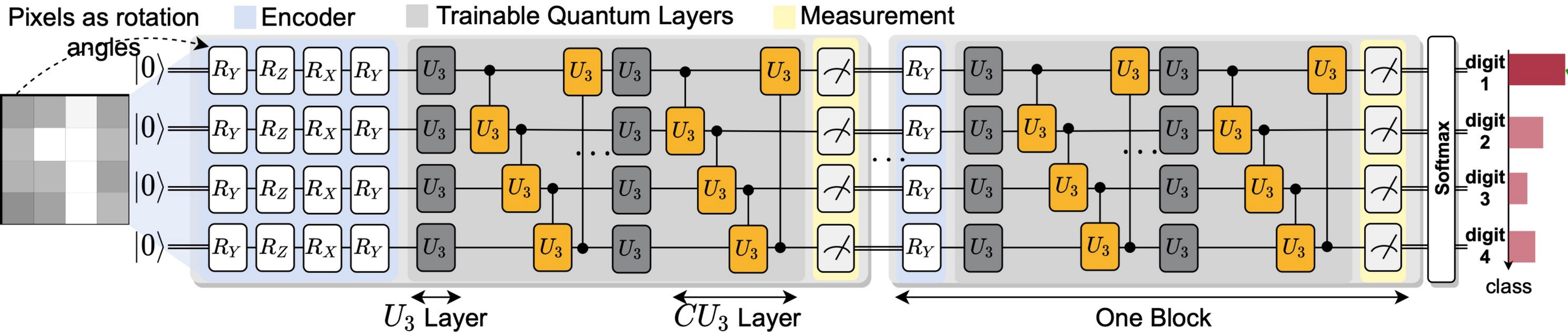
Materials at: https

QuantumNAS vs. QuantumNAT

- **QuantumNAS** finds noise robust circuit **architecture**
- **QuantumNAT** finds noise robust circuit **parameters**

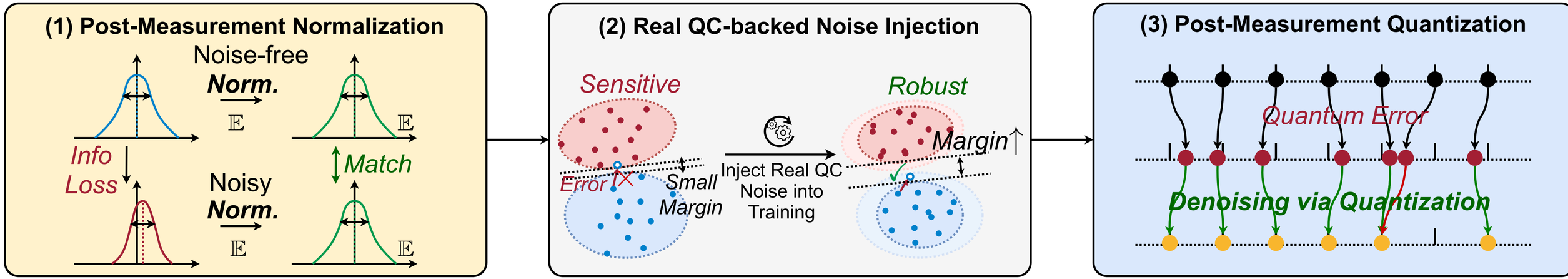
PQC Circuit in QuantumNAT

- QNN with multiple nodes
  - Encoder
  - Trainable Quantum Layers
  - Measurements

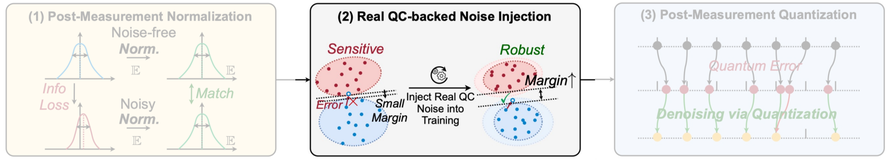


Three Techniques in QuantumNAT

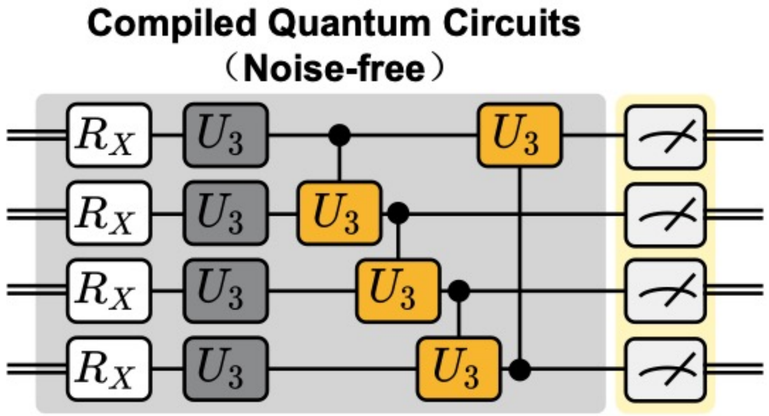
- QuantumNAT:
  - Normalization: mitigate noise impact
  - Noise injection: make the parameters aware of noise
  - Quantization: mitigate noise impact



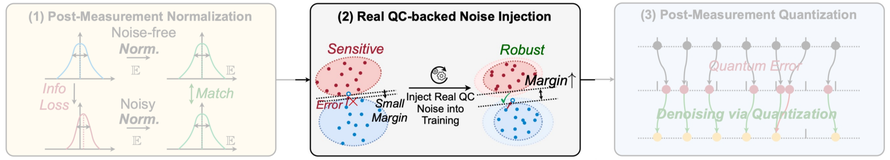
Noise Injection



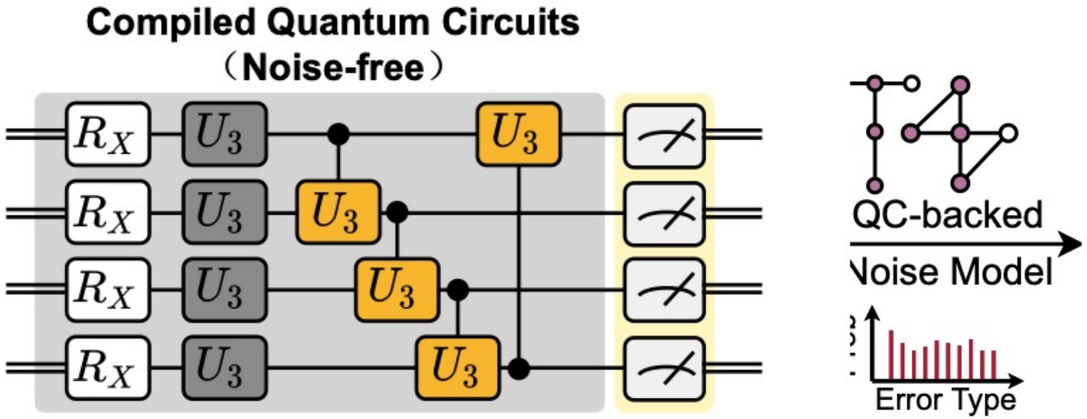
- Inject noise during training on classical simulator
  - Pauli error
  - Readout error



Noise Injection



- Inject noise during training on classical simulator
  - Pauli error
  - Readout error



Pauli error: SX gate: {X: 0.00096, Y: 0.00096, Z: 0.00096, None: 0.99712}

Readout Error Matrix:

0.984, 0.016

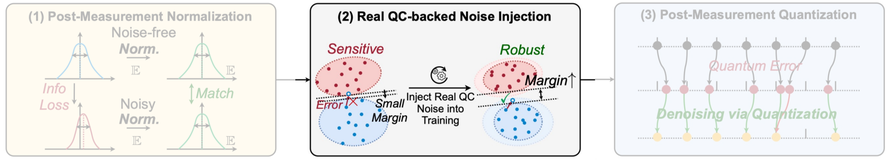
0.022, 0.978

Materials at: https://torchquantum.org

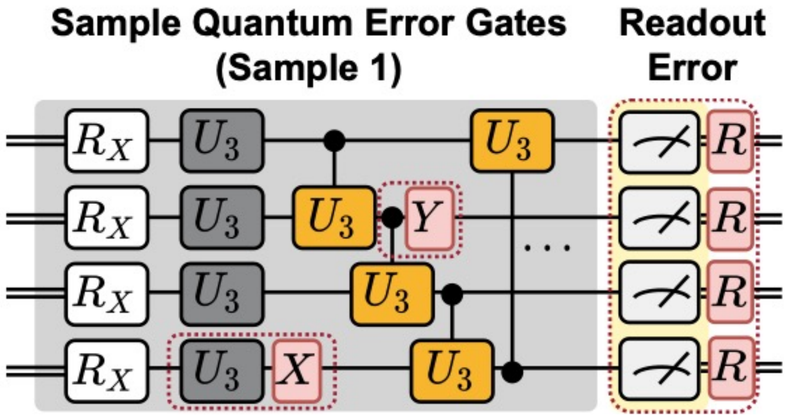
Torch Quantum

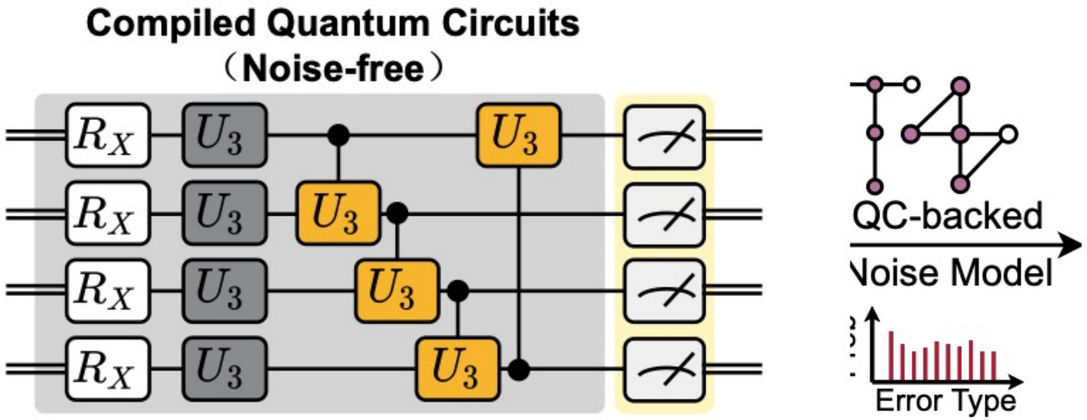
MIT HAN LAB

Noise Injection



- Inject noise during training on classical simulator
  - Pauli error
  - Readout error





Pauli error: SX gate: {X: 0.00096, Y: 0.00096, Z: 0.00096, None: 0.99712}

Readout Error Matrix:

0.984, 0.016

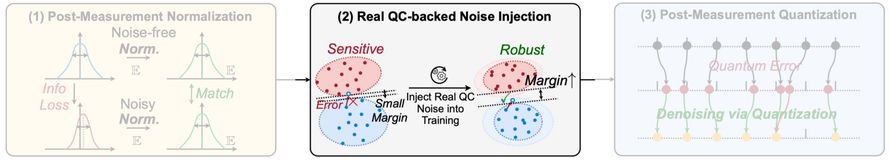
0.022, 0.978

Materials at: https://torchquantum.org

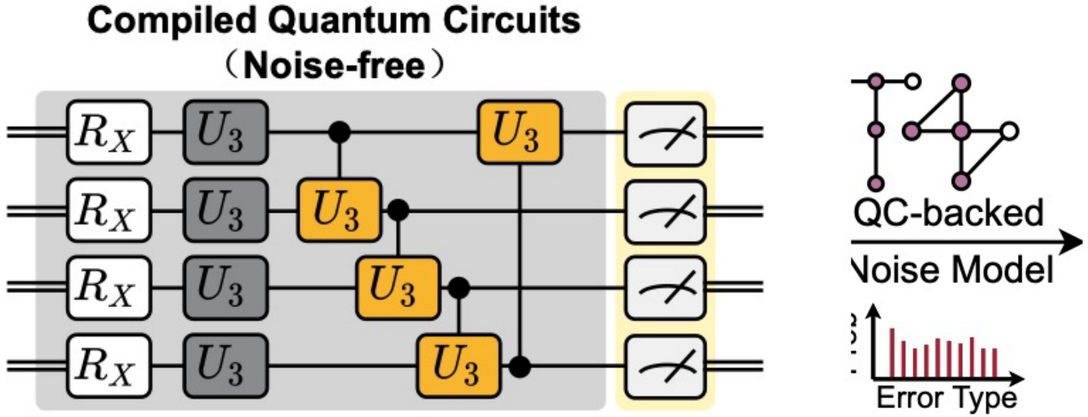
Torch Quantum

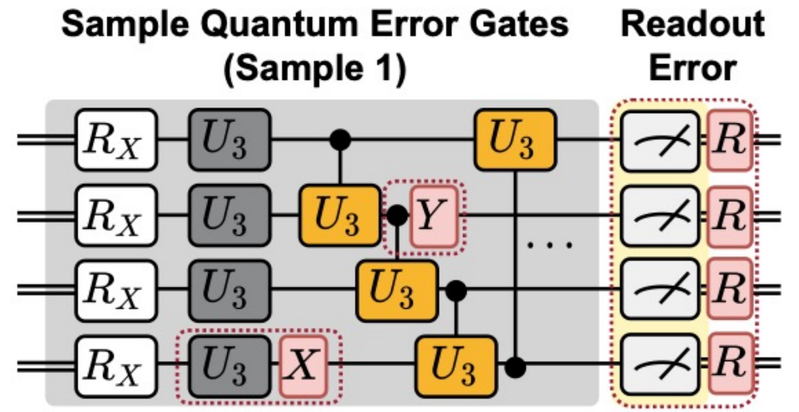
MIT HAN LAB

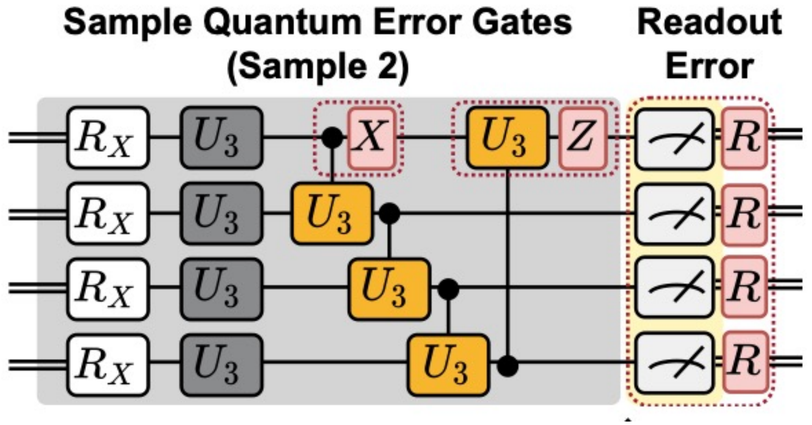
Noise Injection



- Inject noise during training on classical simulator
  - Pauli error
  - Readout error







Pauli error: SX gate: {X: 0.00096, Y: 0.00096, Z: 0.00096, None: 0.99712}

Readout Error Matrix: 0.984, 0.016
0.022, 0.978

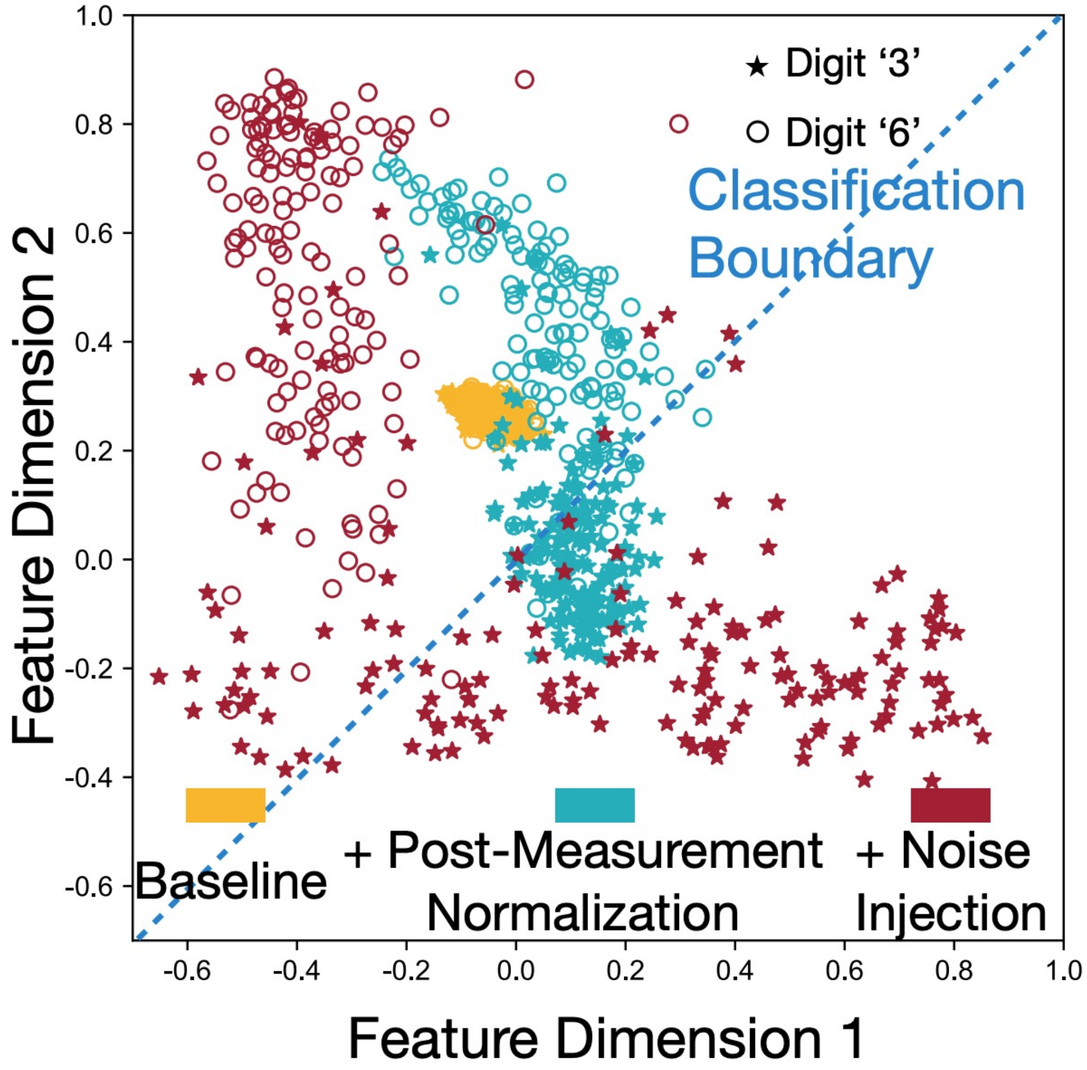
Materials at: https://torchquantum.org

Torch Quantum

MIT HAN LAB

Visualization

- QuantumNAT stretches the distribution of features
  - MNIST-2 classification task

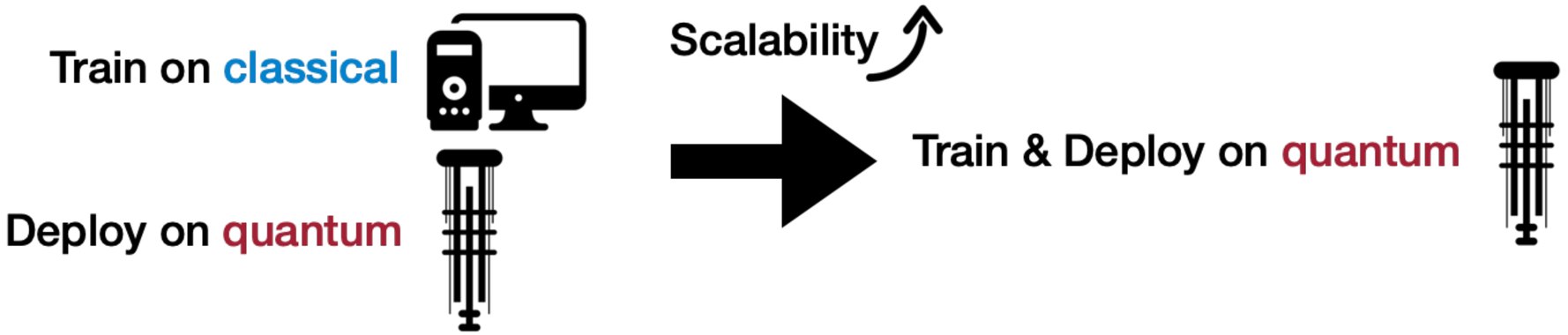


Noise Model in TQ

```
noise_model_tq = tq.NoiseModelTQ(
        noise_model_name='ibmq_quito’,
        factor=10,
        add_thermal=True
    )
```

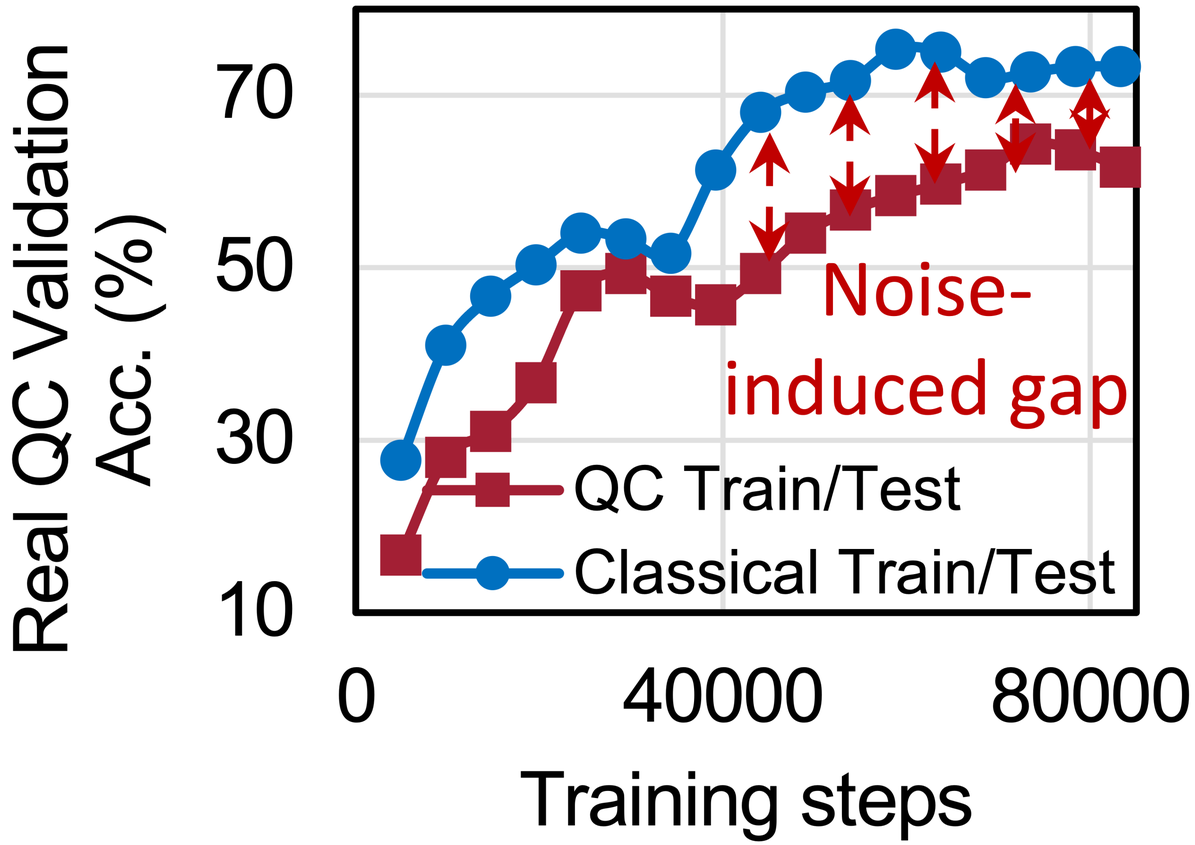
QOC for High Scalability

- How to further improve the scalability of PQC training?
- Train the parameters directly on real quantum machine



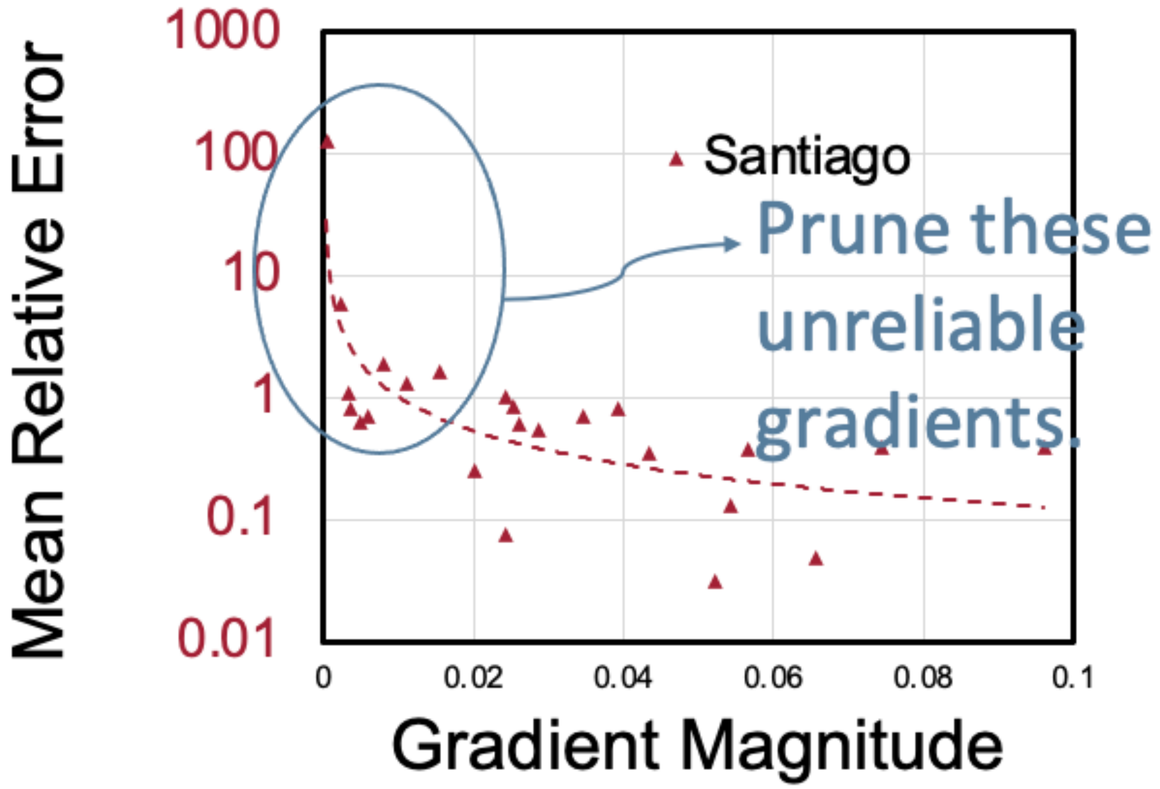
Challenge of On-chip Training: noise

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



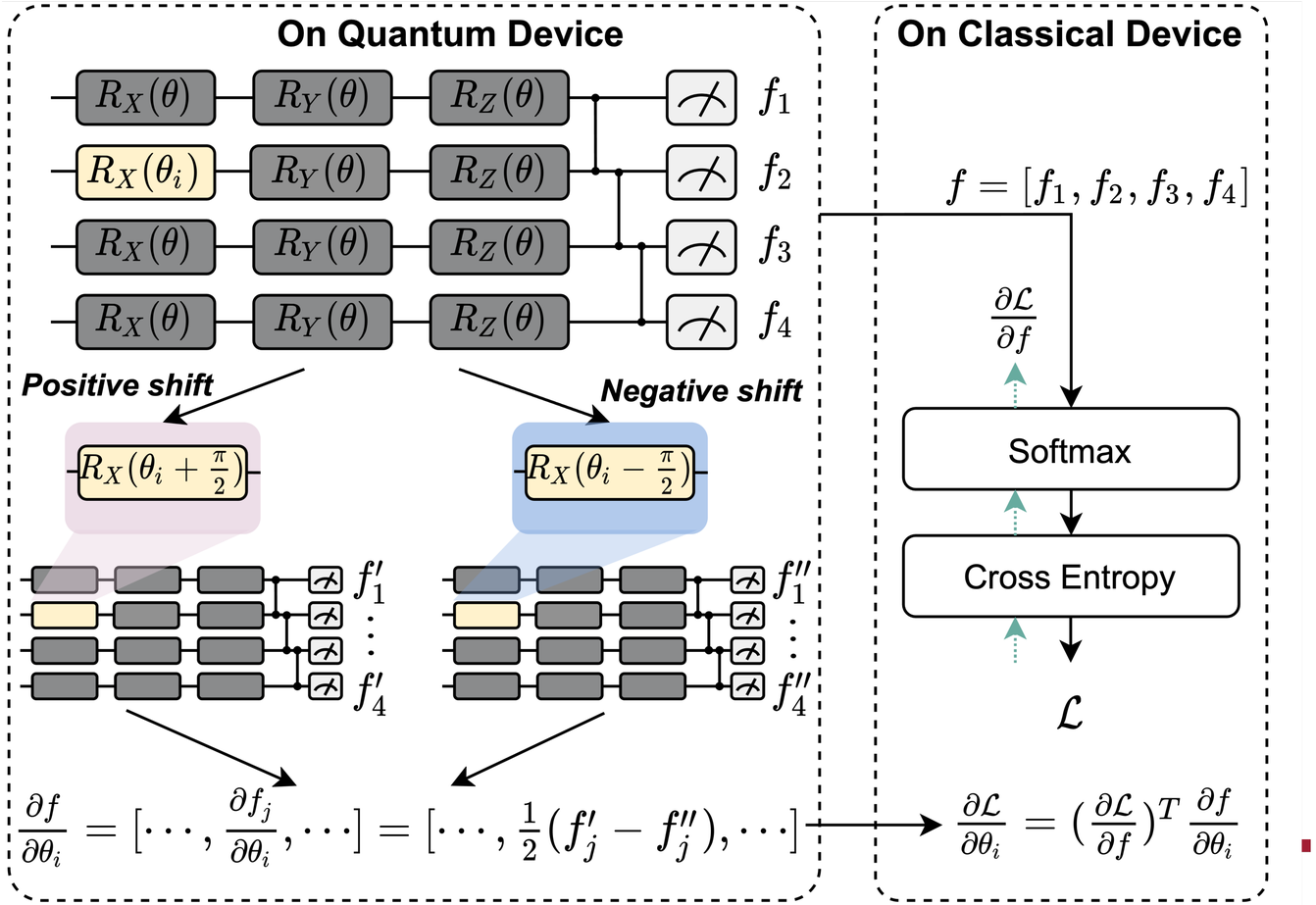
Challenge of On-chip Training: noise

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



Calculate Gradients of PQC

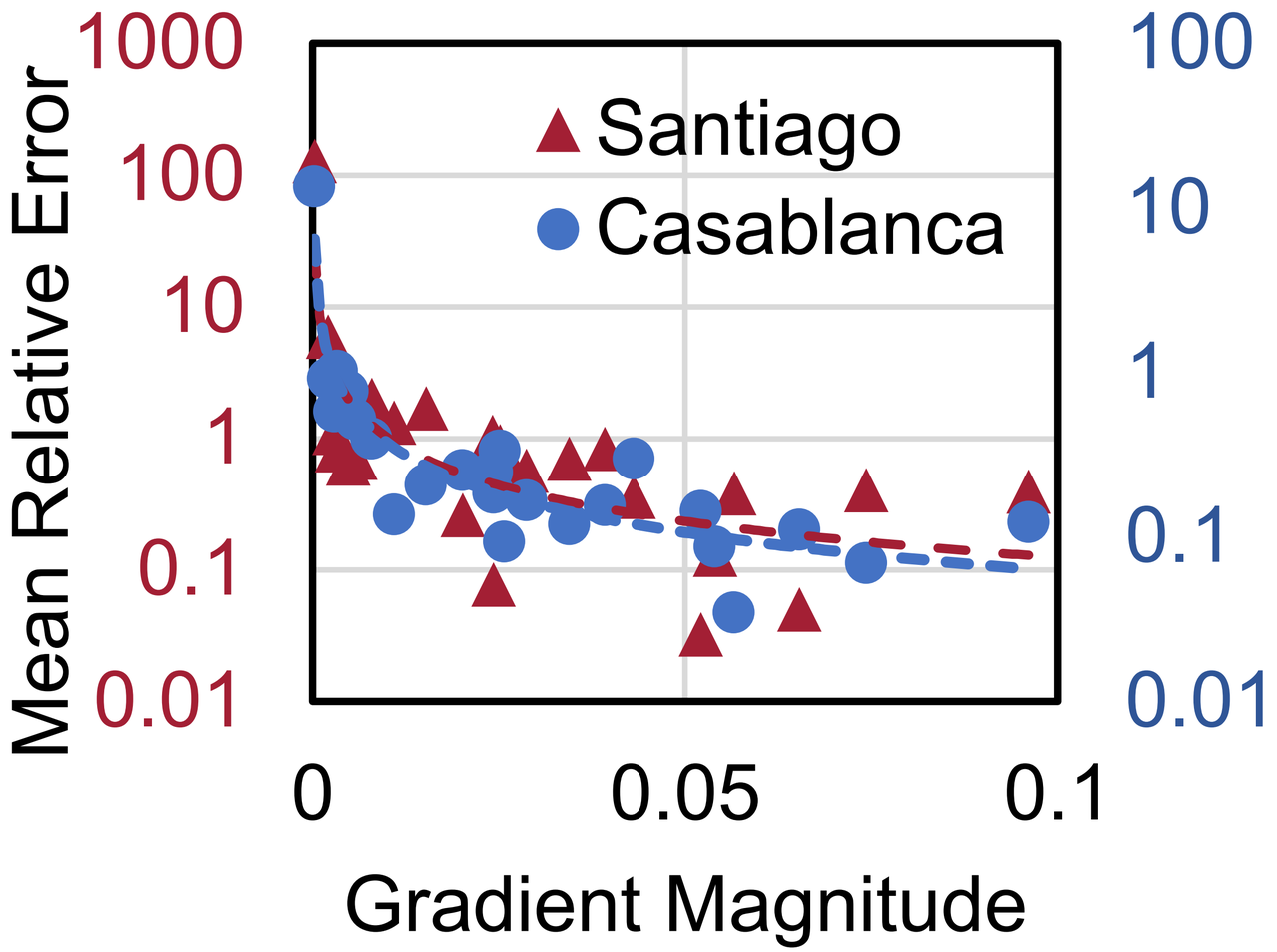
- Only **forward** on quantum device



Torch Quantum

Probabilistic Gradient Pruning

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

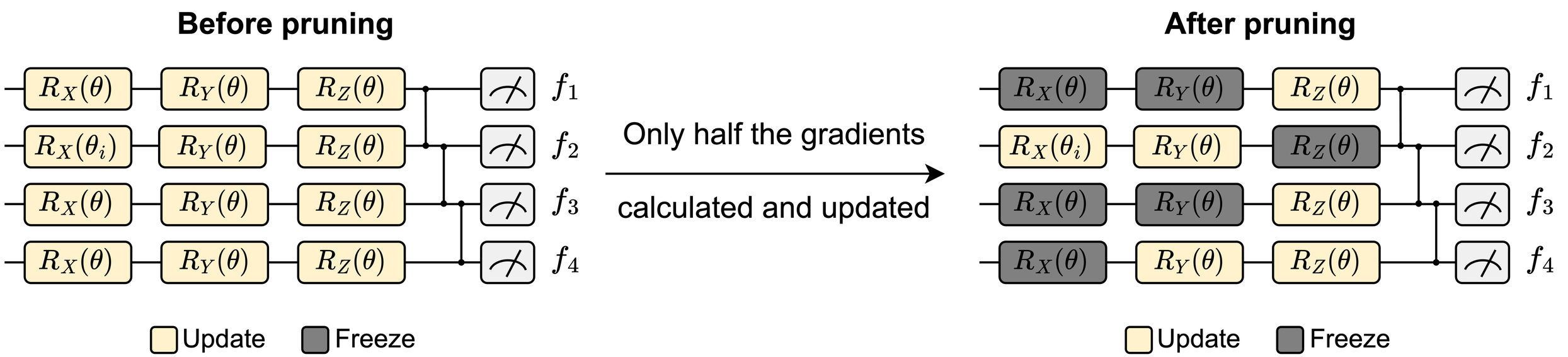


Torch Quantum

Materials at: https://torchquantum.org

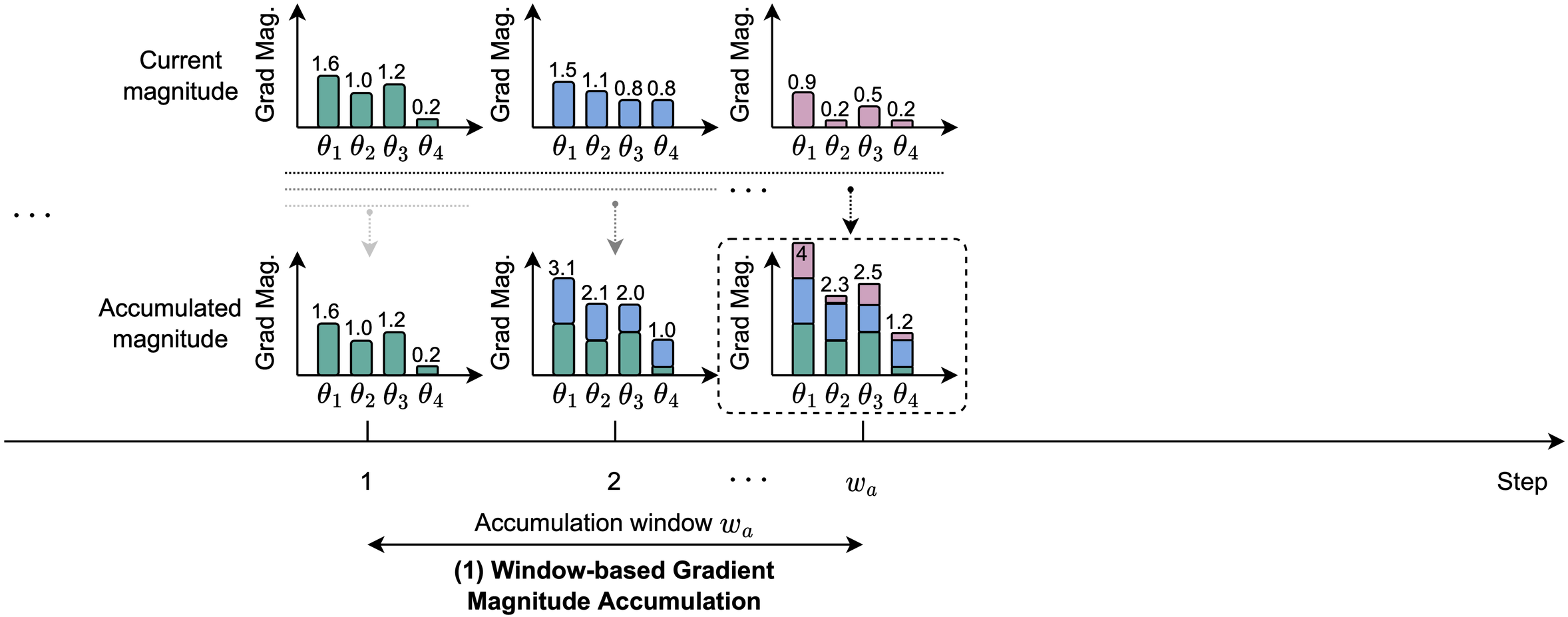
MIT HAN LAB

Probabilistic Gradient Pruning



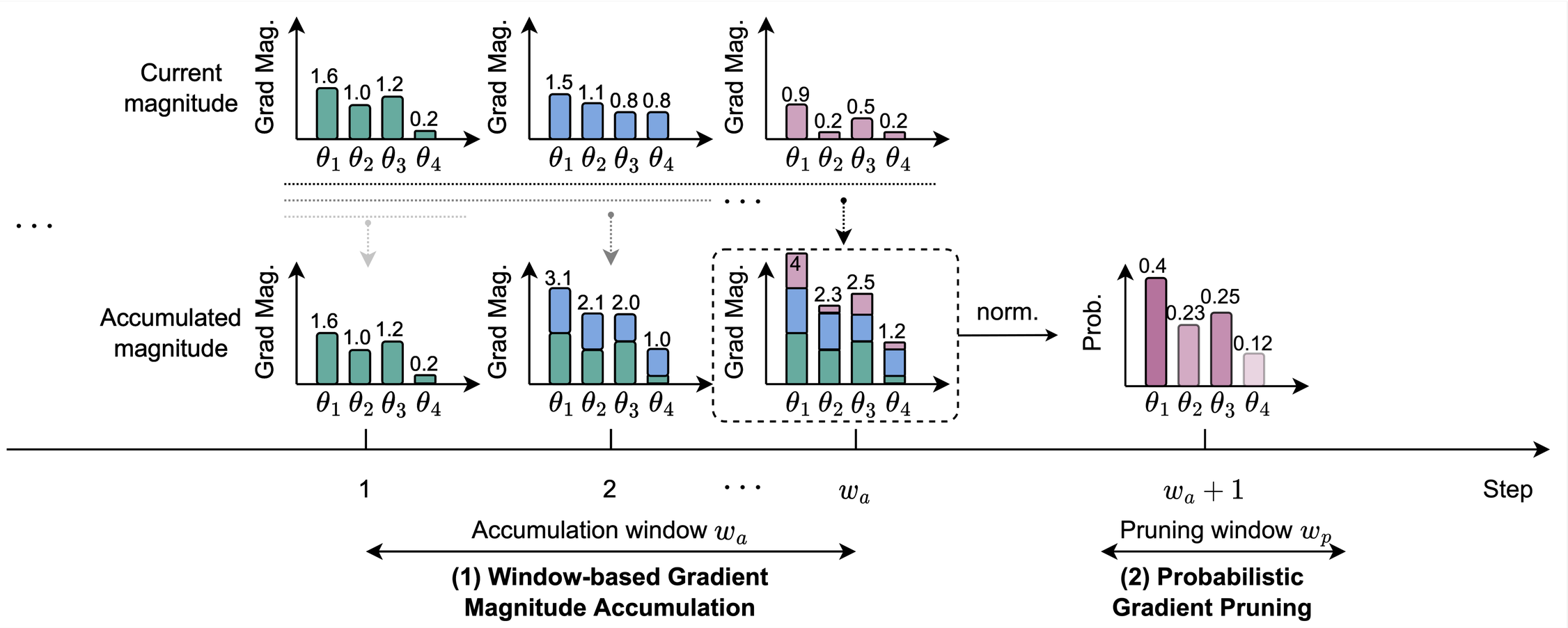
Accumulation Window

- Keep a record of accumulated gradient magnitude



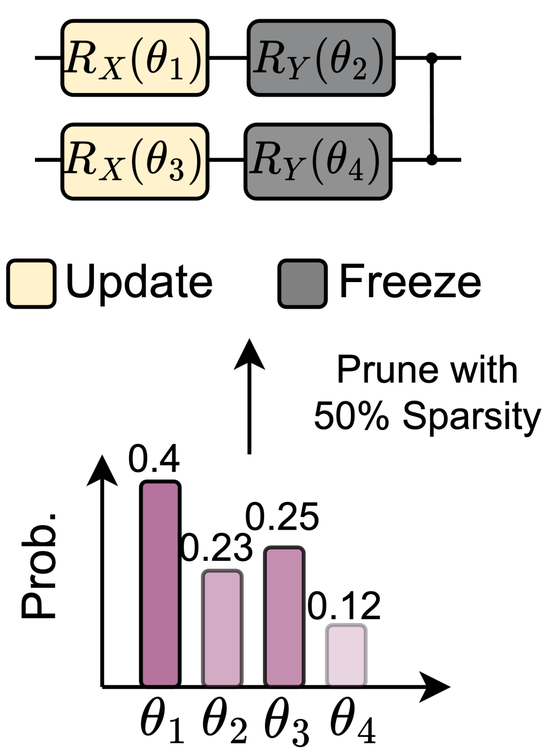
Accumulation Window

- Normalize the accumulated gradient magnitude to a probability distribution



Accumulation Window

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



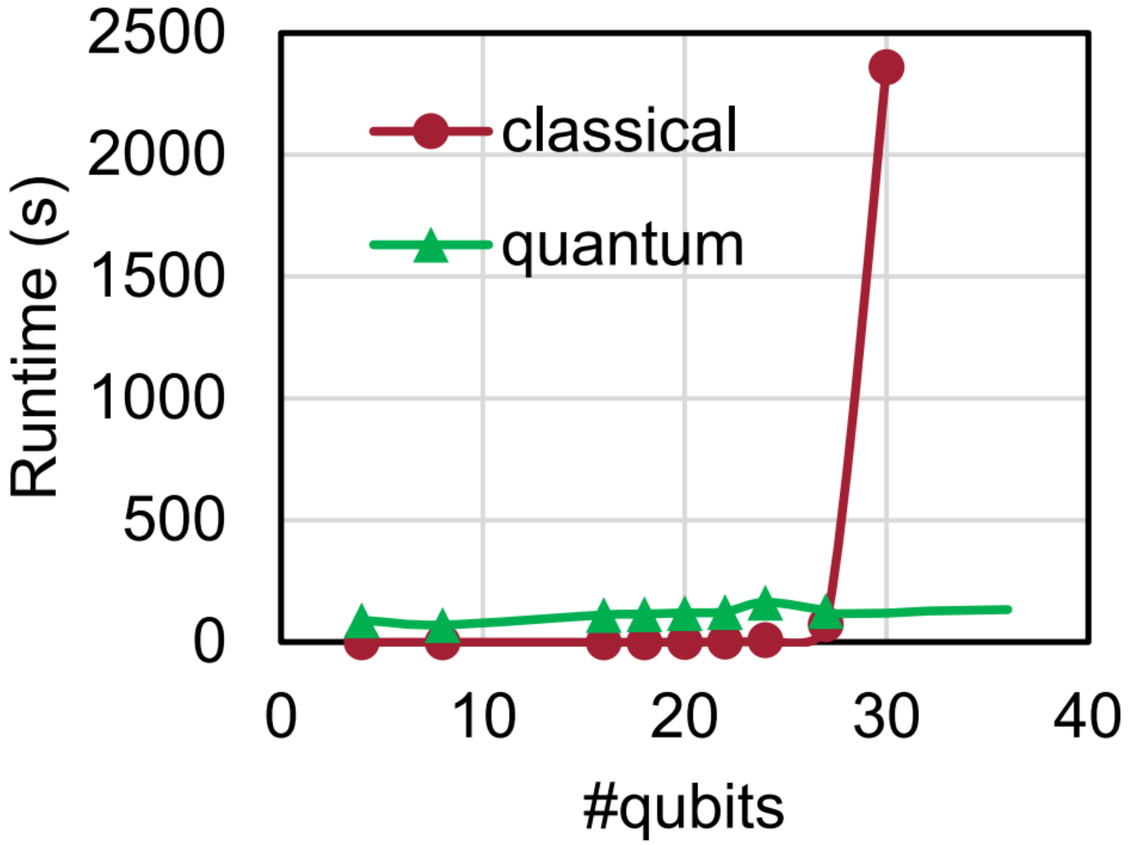
QNN results

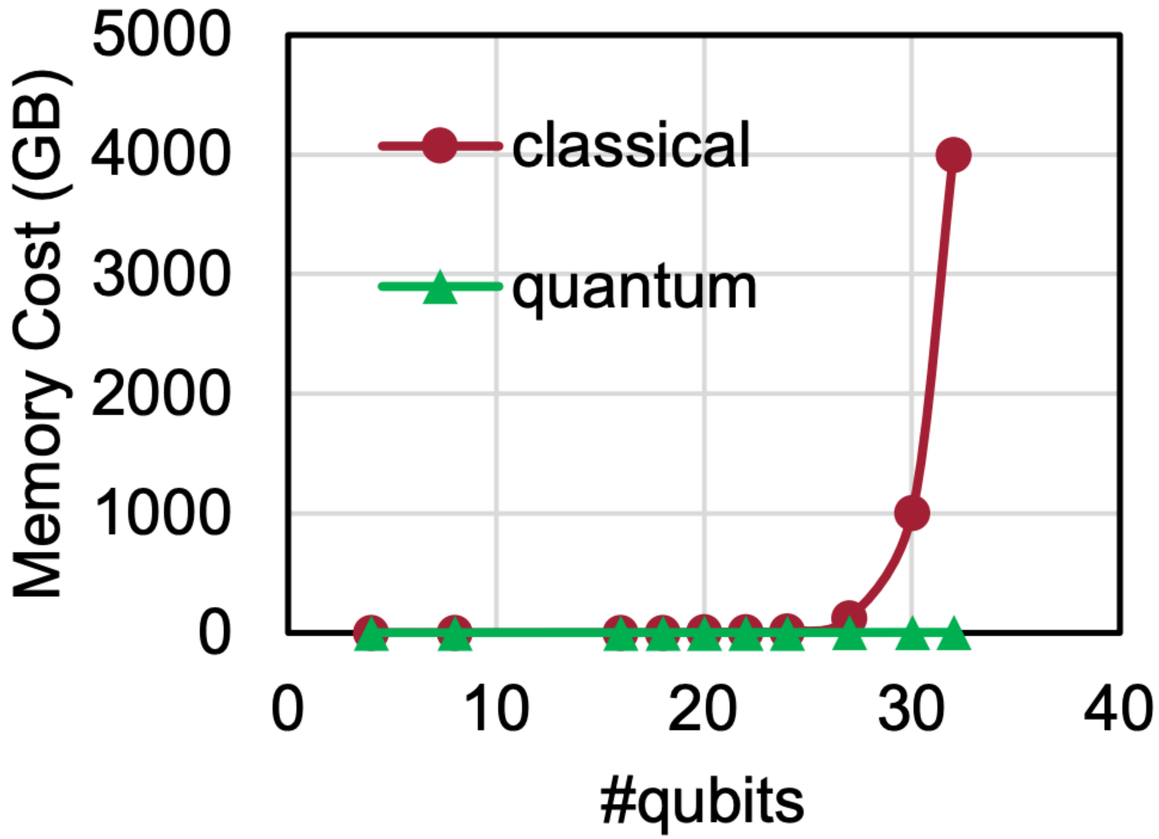
- QOC achieved similar results to classical simulation

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

Scalability Improvements

- On-chip Training is scalable





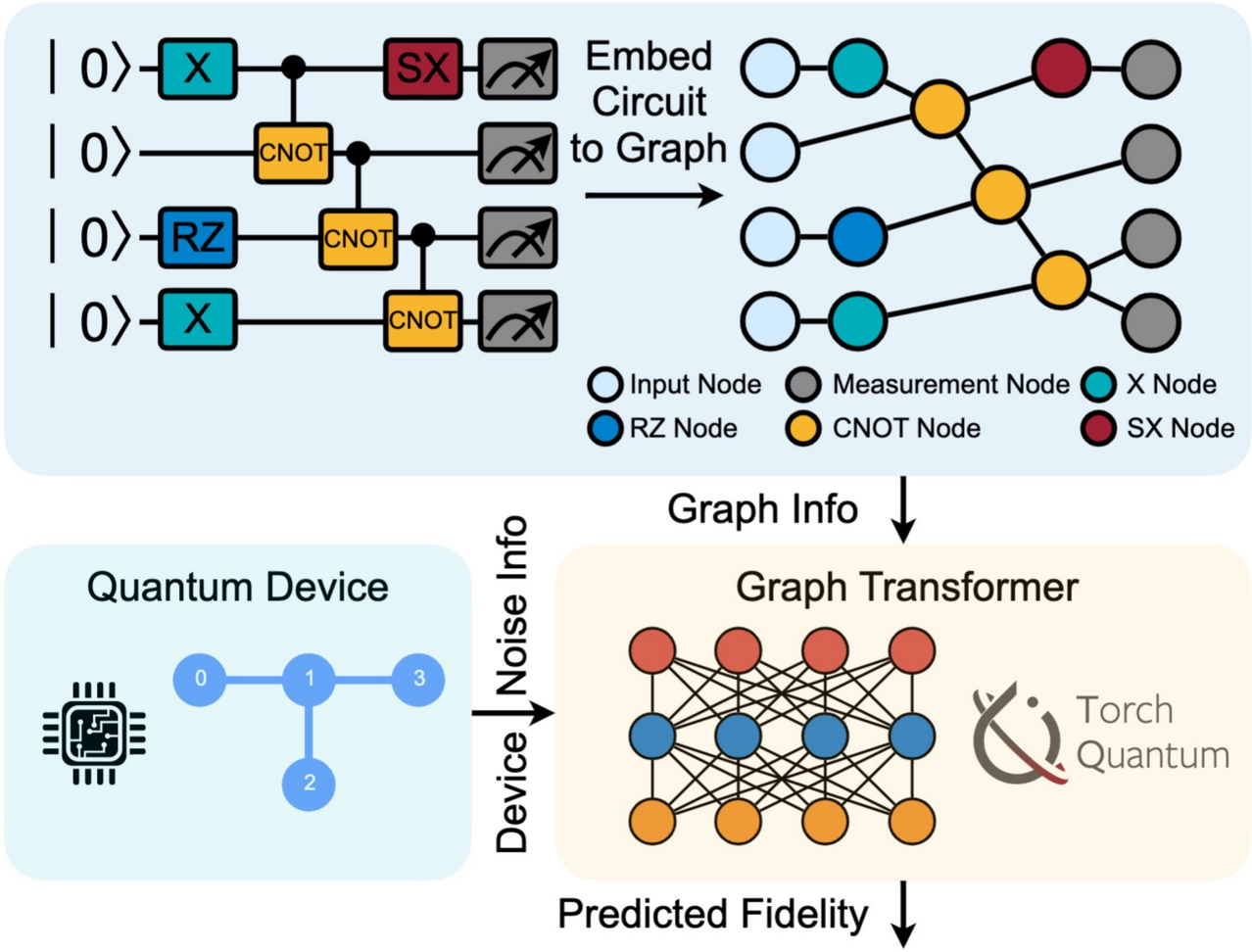
Torch Quantum

Materials at: https://torchquantum.org

MIT HAN LAB

Transformer for Quantum Circuit Reliability Prediction

- Use the circuit graph information



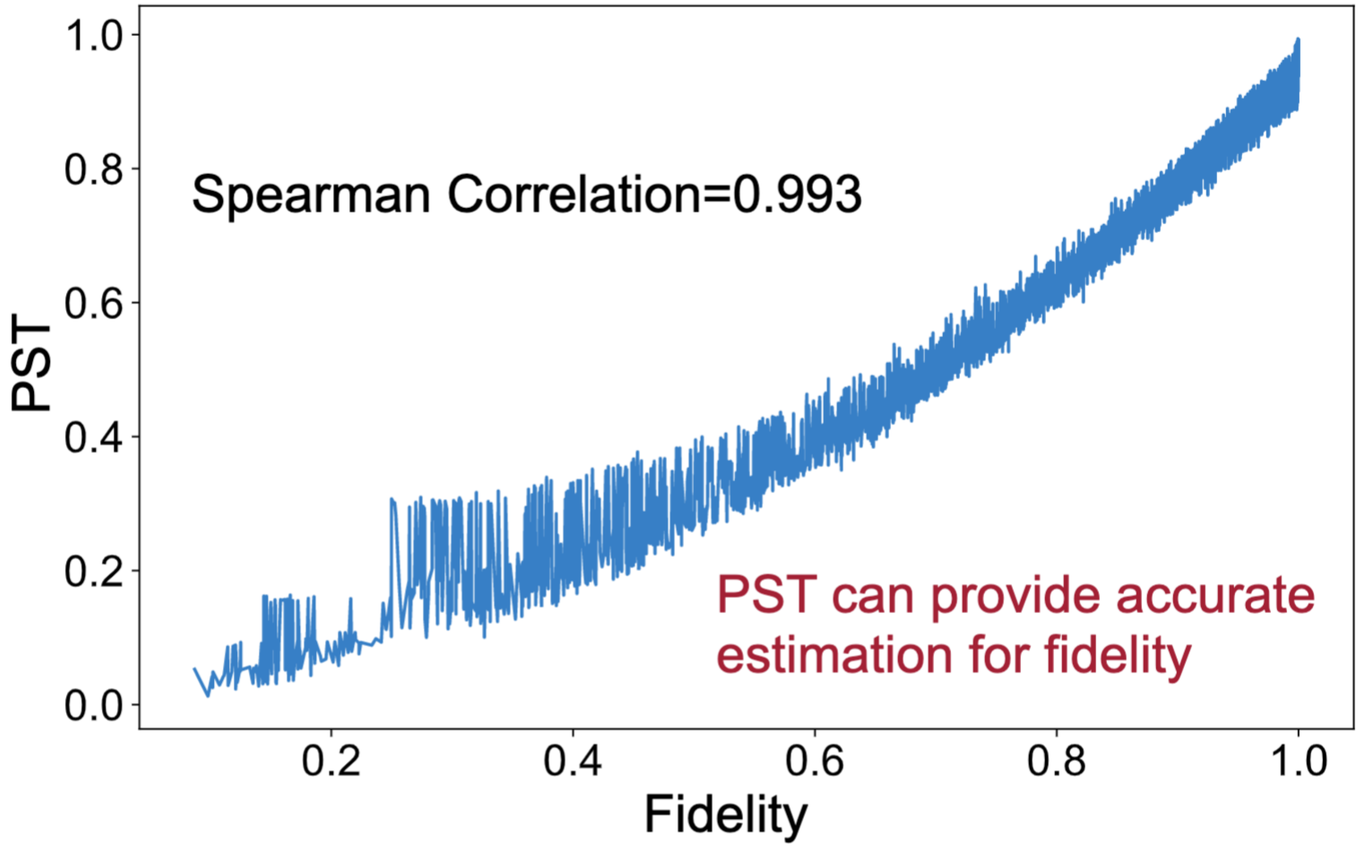
Torch Quantum

Materials at: https://torchquantum.org

Transformer for Quantum Circuit Reliability Prediction

- Use PST as the metrics

$$PST = \frac{\#Trials\ with\ output\ same\ as\ initial\ state}{\#Total\ trials}$$



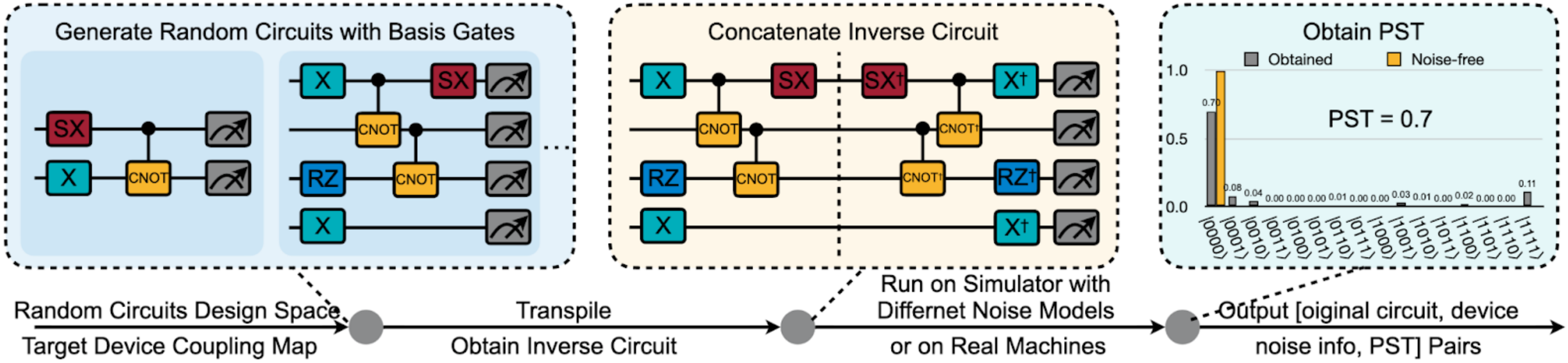
Torch Quantum

Materials at: https://torchquantum.org

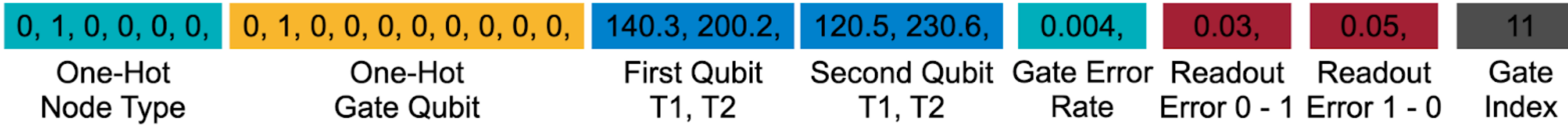
MIT HAN LAB

Dataset collection

- Collect dataset on real machine / noisy simulator

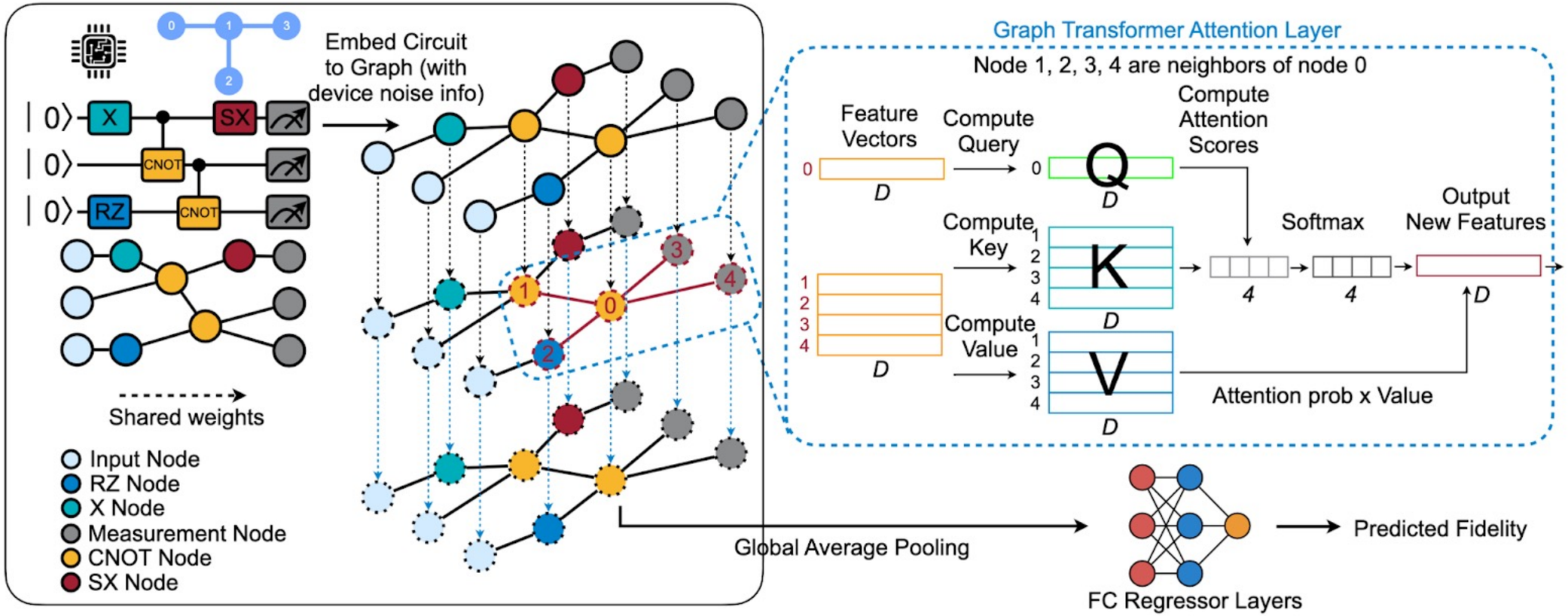


Features on each node



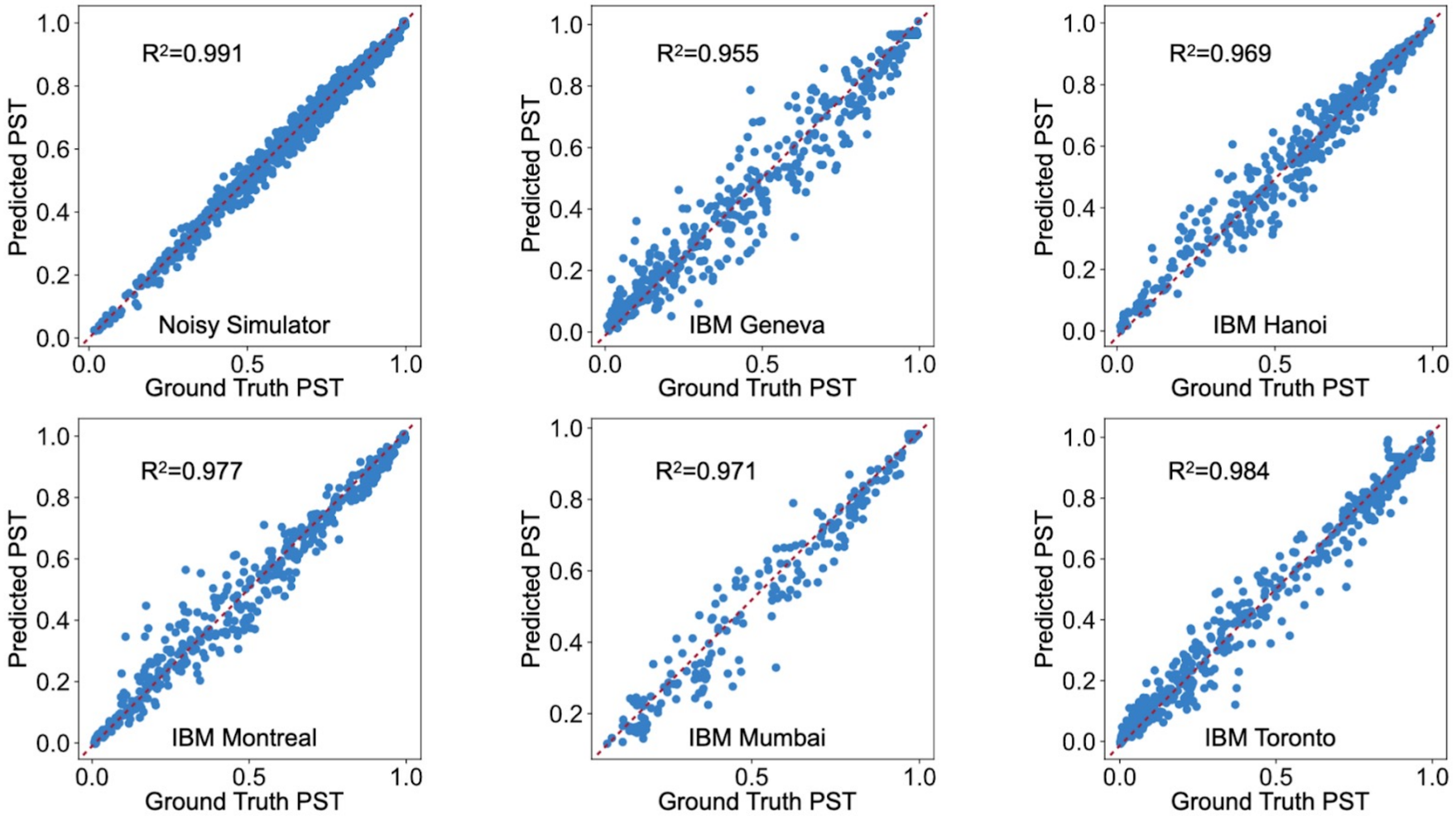
Graph Transformer

- Graph Transformer



Evaluation

- On random generated circuit



Torch Quantum

Materials at: https://torchquantum.org

TorchQuantum Tutorial Outline

Section 1

Quantum Basics and TorchQuantum Usage

1.1 Introduction to Quantum Computing (Prof. Yongshan Ding)

1.2 TorchQuantum API and examples (Hanrui Wang)

Section 2

Use TorchQuantum on Gate Level Optimization

2.1 QuantumNAS Ansatz Search and On-Chip Training (Hanrui Wang)

2.2 Variational Quantum Circuit Compression (Prof. Weiwen Jiang)

2.3 Various VQA Gradient Estimation Methods (Dantong Li)

Section 3

Use TorchQuantum on Pulse Level Optimization

3.1 Intro to Pulse Level Quantum Control (Zhiding Liang)

3.2 Variational Pulse Learning & Optimal Control (Jinglei Cheng)

Torch Quantum

Materials at: https://torchquantum.org

MIT HAN LAB

Thank you for listening!

Torch Quantum

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

qmlsys.mit.edu