MIT HAN LAB

EfficientML.ai Lecture 23 Noise Robust Quantum Machine Learning

Part II

Hanrui Wang

PhD Student, MIT

@hanrui_w

Lecture Plan

Today we will:

1. Continue TorchQuantum examples
2. Introduce robust state preparation
3. Introduce robust PQC architecture search
4. Introduce robust PQC parameter training

Section 1

TorchQuantum (continued)

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)`
- Two 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)`
- Two ways of applying quantum gates: method 1:
  - `import torchquantum.functional as tqf`
  - `tqf.h(q_dev, wires=1)`
- Two 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.QuantumState class can also store the statevectors
  - `q_state = tq.QuantumState(n_wires=5)`
- Three ways of applying quantum gates
  - `import torchquantum.functional as tqf`
  - `tqf.h(q_state, wires=1)`
  - `h_gate = tq.H()`
  - `h_gate(q_state)`
  - `q_state.h()`
  - `q_state.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),
```

Dynamic Computation Graph

- Performing matrix-vector multiplication between the gate matrix and statevector
- The `tq.QuantumState` class can also store the statevectors
  - `q_state = tq.QuantumState(n_wires=5)`
  - `q_state.h(wires=1)`
  - `print(q_state)`
  - `q_state.sx(wires=3)`
  - `print(q_state)`

Batch Mode Tensorized Processing

- Performing matrix-vector multiplication between the gate matrix and statevector
- The `tq.QuantumState` class can also store the statevectors
  - `q_state = tq.QuantumState(n_wires=5, bsz=64)`

GPU Support

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

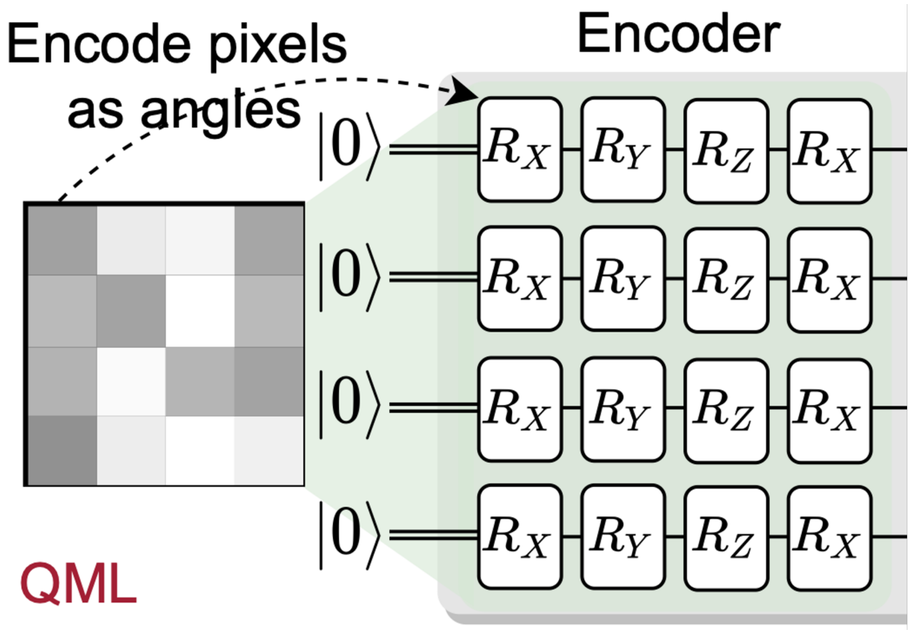
Automatic Gradient Computation

- Performing matrix-vector multiplication between the gate matrix and statevector
- The `tq.QuantumState` class can also store the statevectors
  - `q_state = tq.QuantumState(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

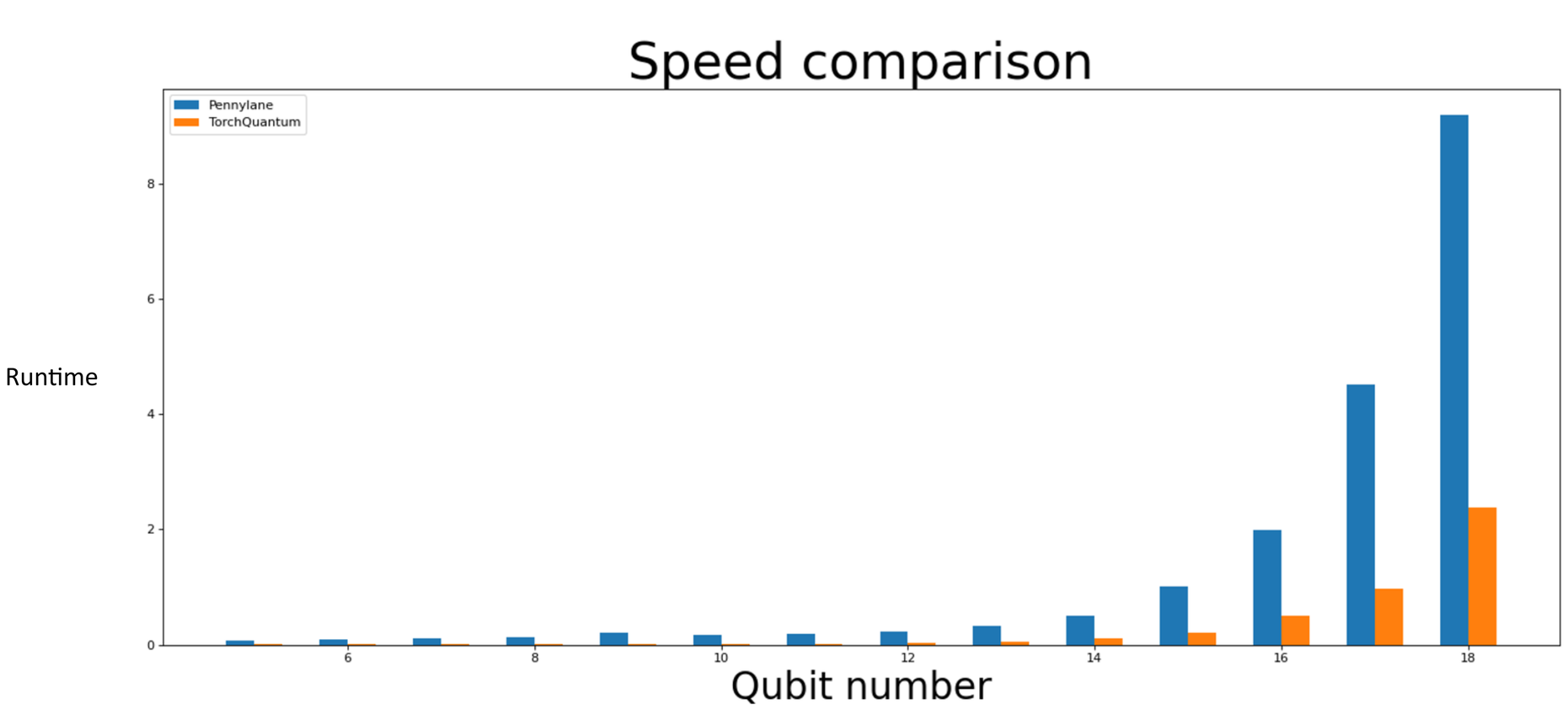


Convert tq Model to Other Frameworks

- 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')`

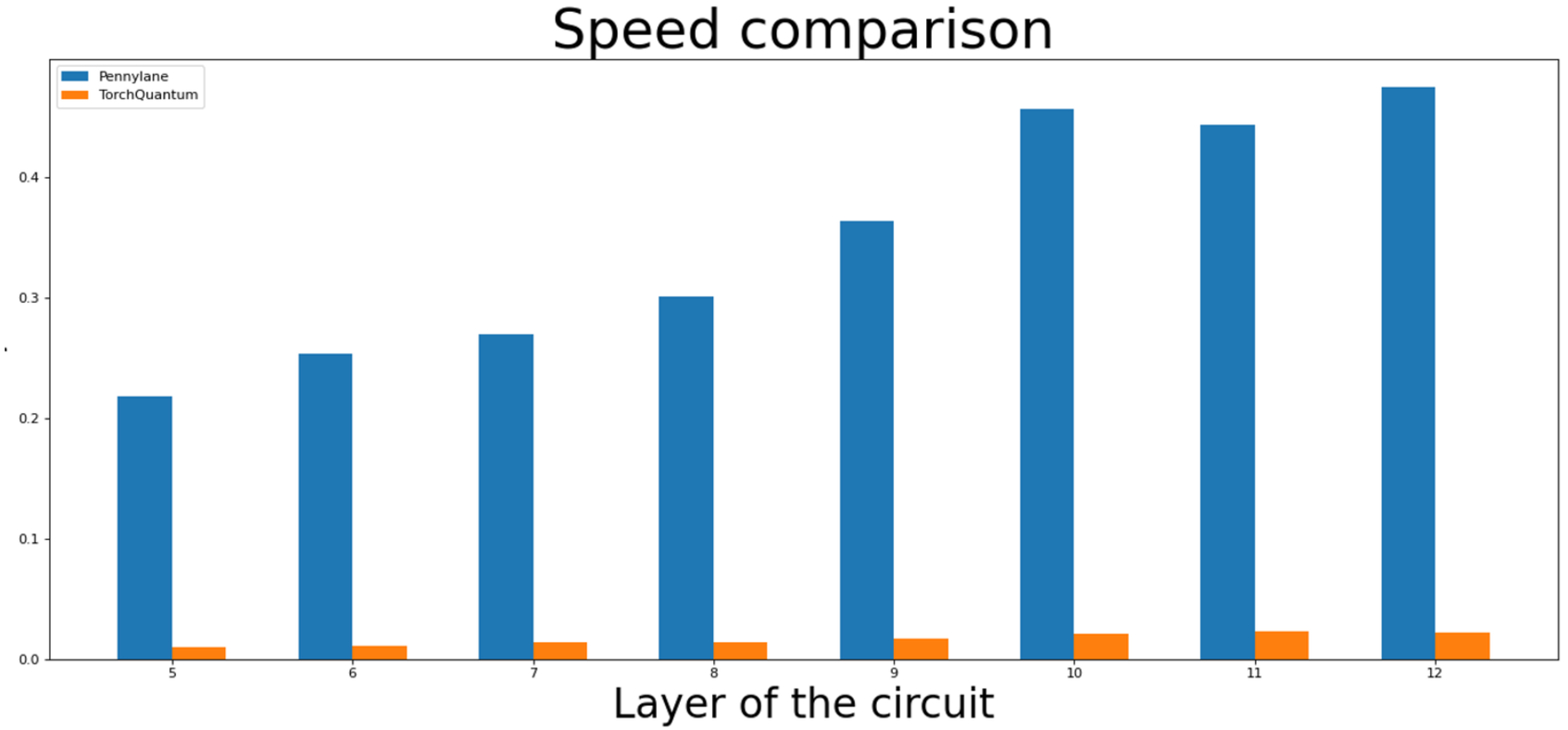
Speedup over Other Frameworks

Comparison to Pennylane



Speedup over Other Frameworks

Comparison to Pennylane

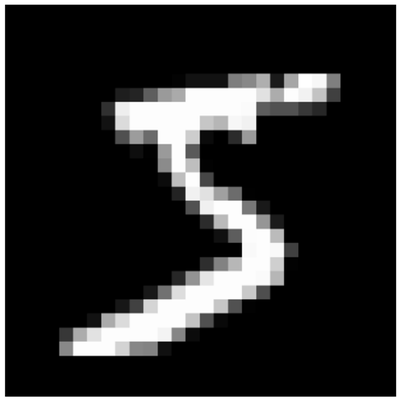


Section 2

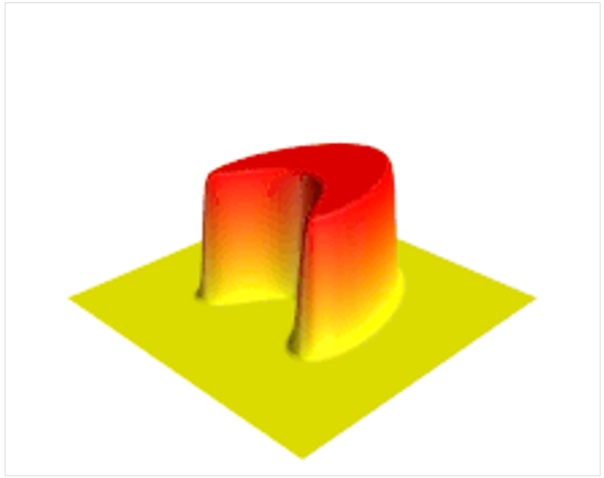
Robust Quantum State Preparation

Quantum State Preparation

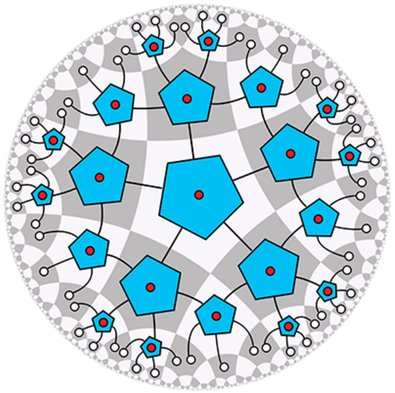
- Prepare the initial state of the quantum device



Amplitude Encoding in QML



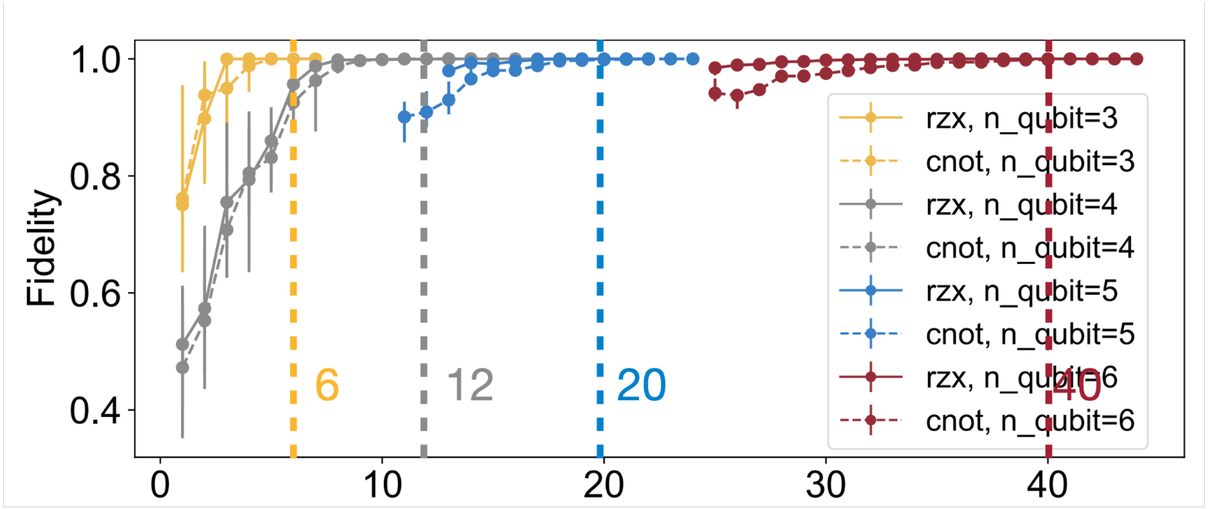
Initial States in PDE

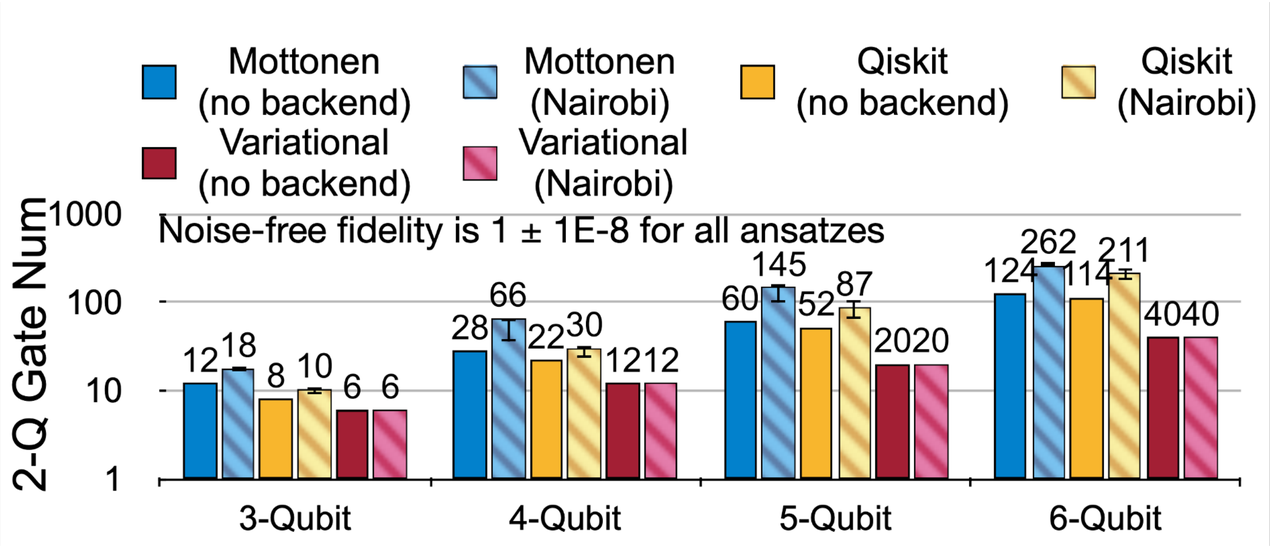


Initial States in Quantum Error Correction

Cost of Variational State Preparation

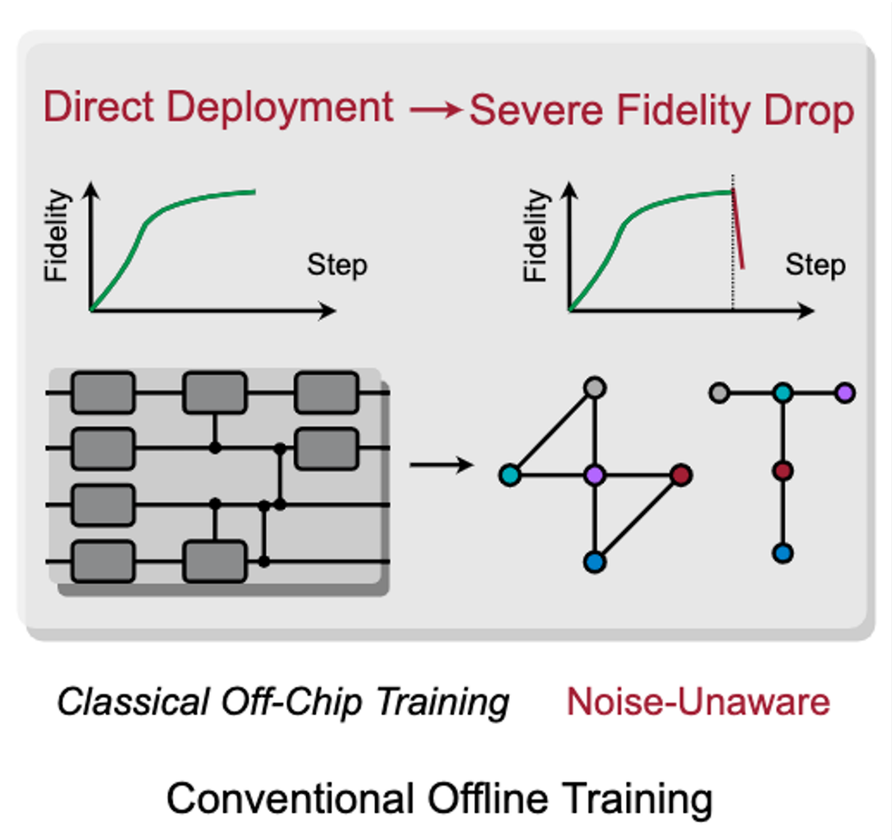
- Number of 2Q gate required is O(2^N)
- Variational State preparation requires fewer number of gates





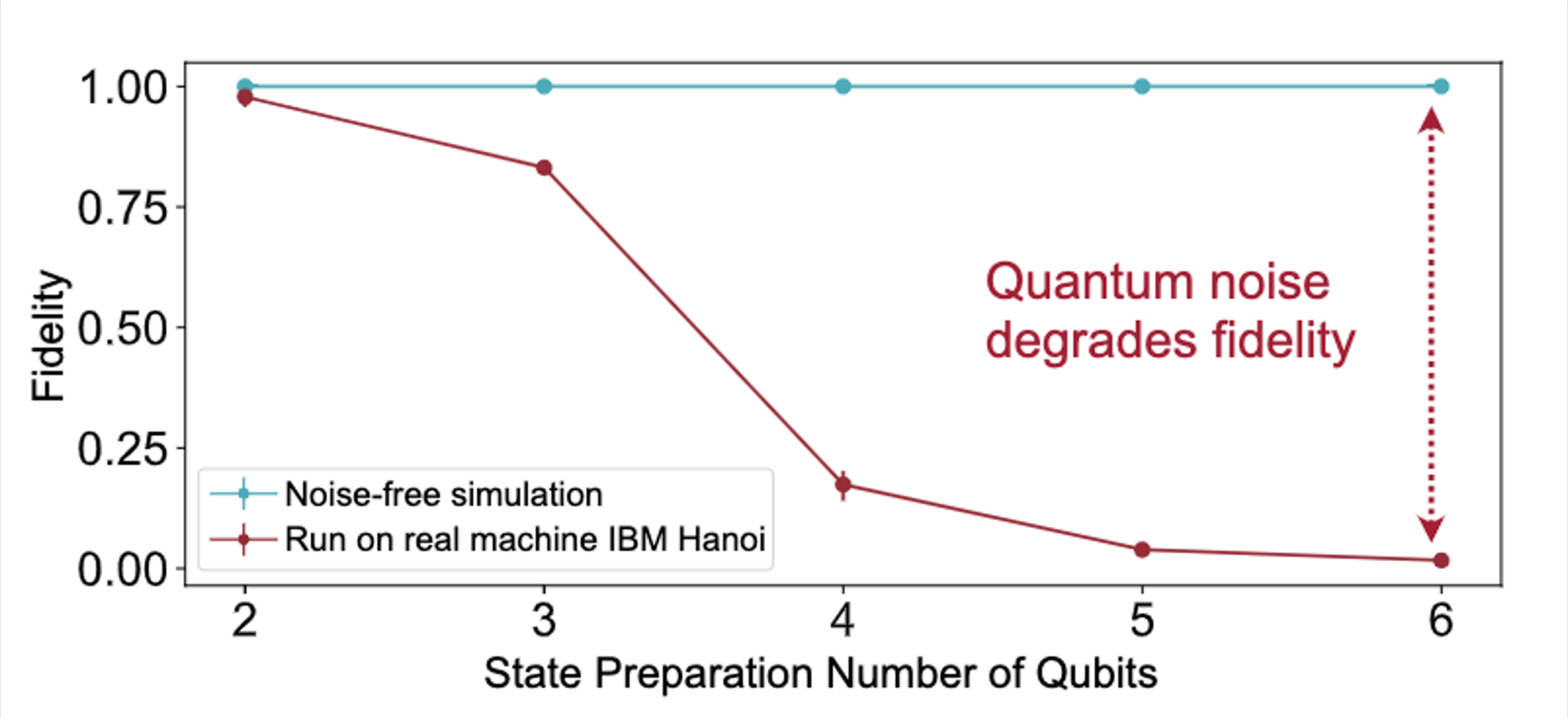
Quantum Noise Impact

- Noise degrades state prep fidelity

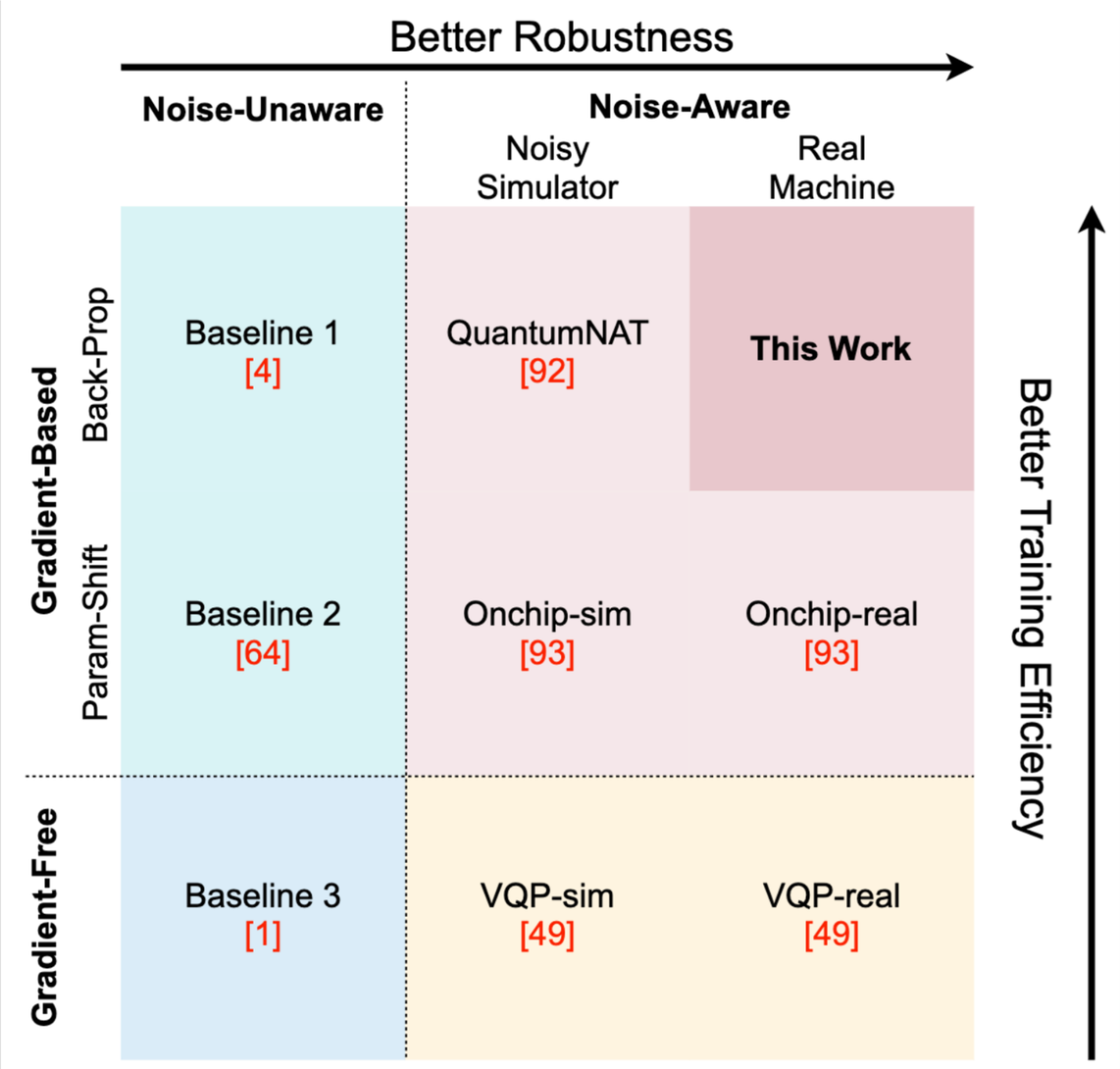


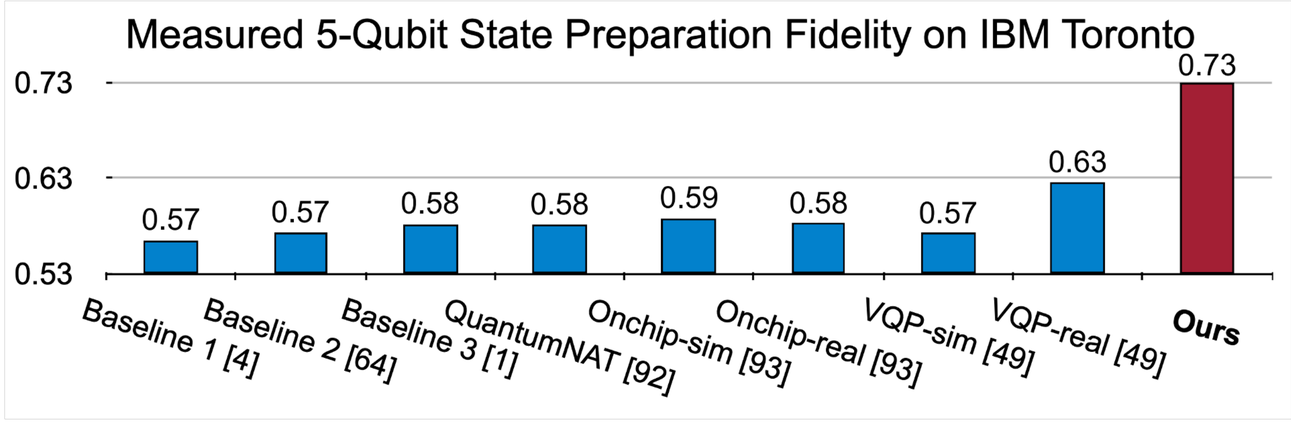
Quantum Noise Impact

- Noise degrades state prep fidelity



Prior Work for Robust Variatonal Circuit





	Parameter-Shift	Gradient-Free	RobustState
Scaling w.r.t. #Params	$\mathcal{O}(n)$	Unscalable	$\mathcal{O}(1)$
Gradient Guidance	✓	✗	✓

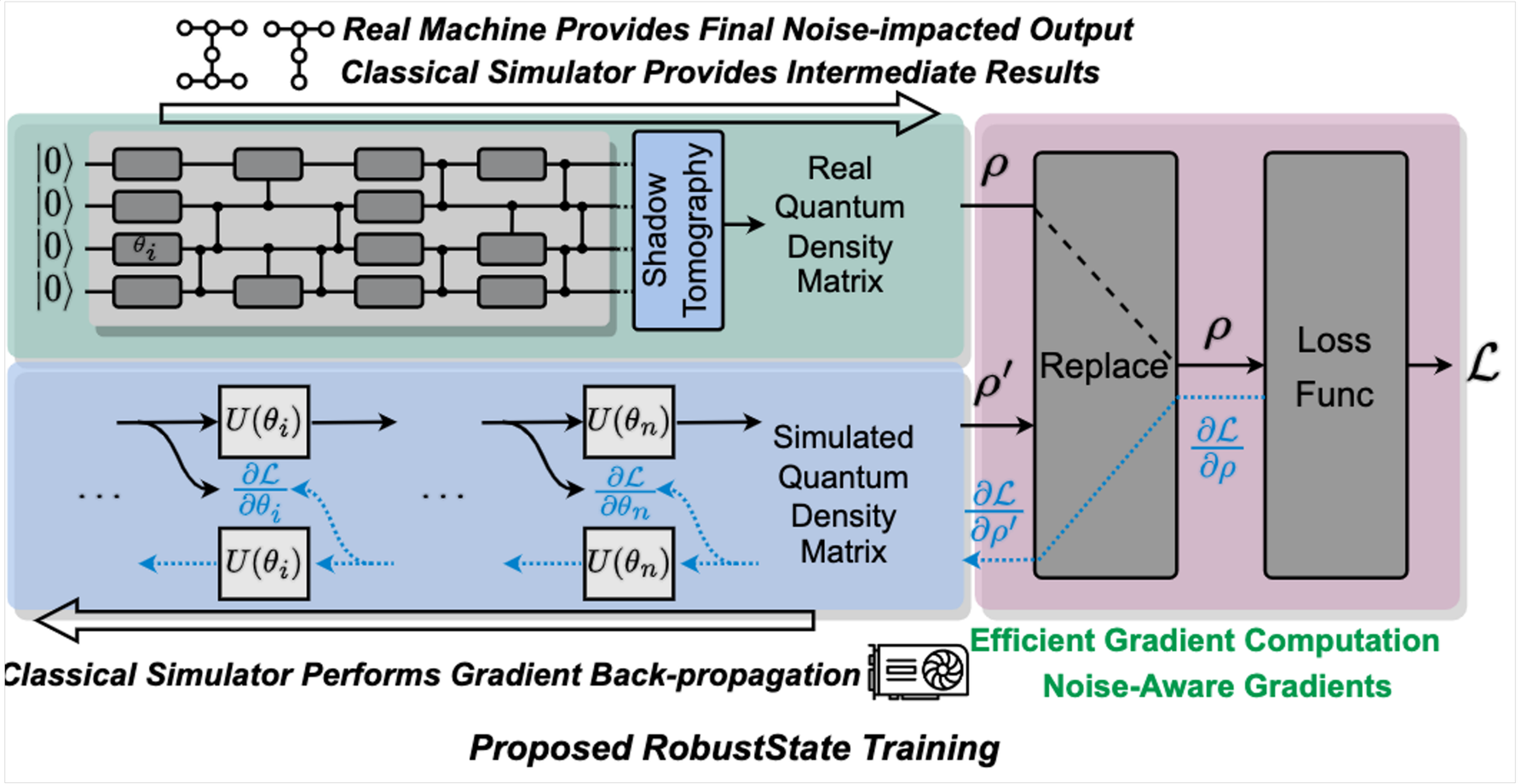
Robust State Prep

- Gradient proxy
- Native Pulse
- Hardware Efficient Ansatz

Gradient Proxy

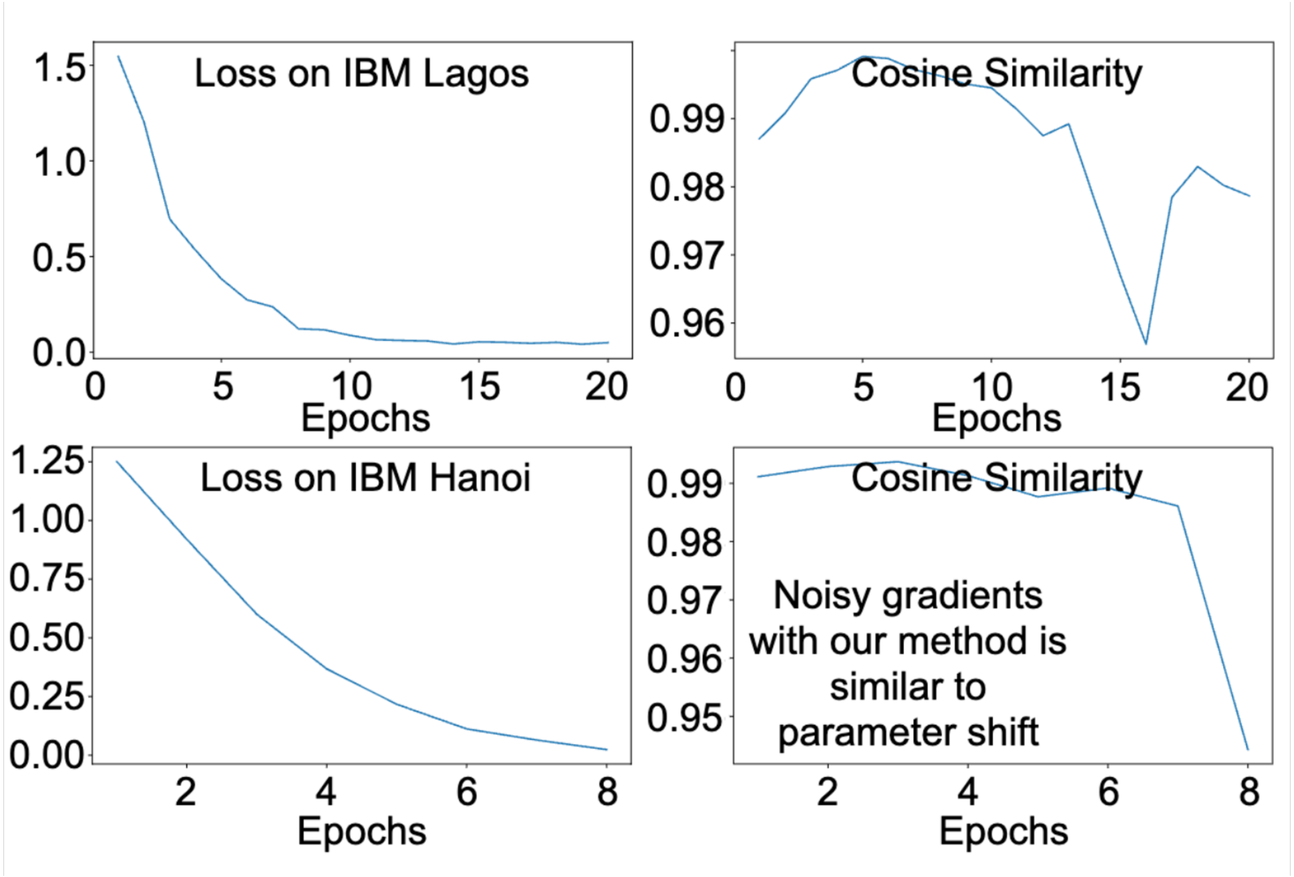
Forward on real device; backward on simulator

- Make the parameters aware of the real noise



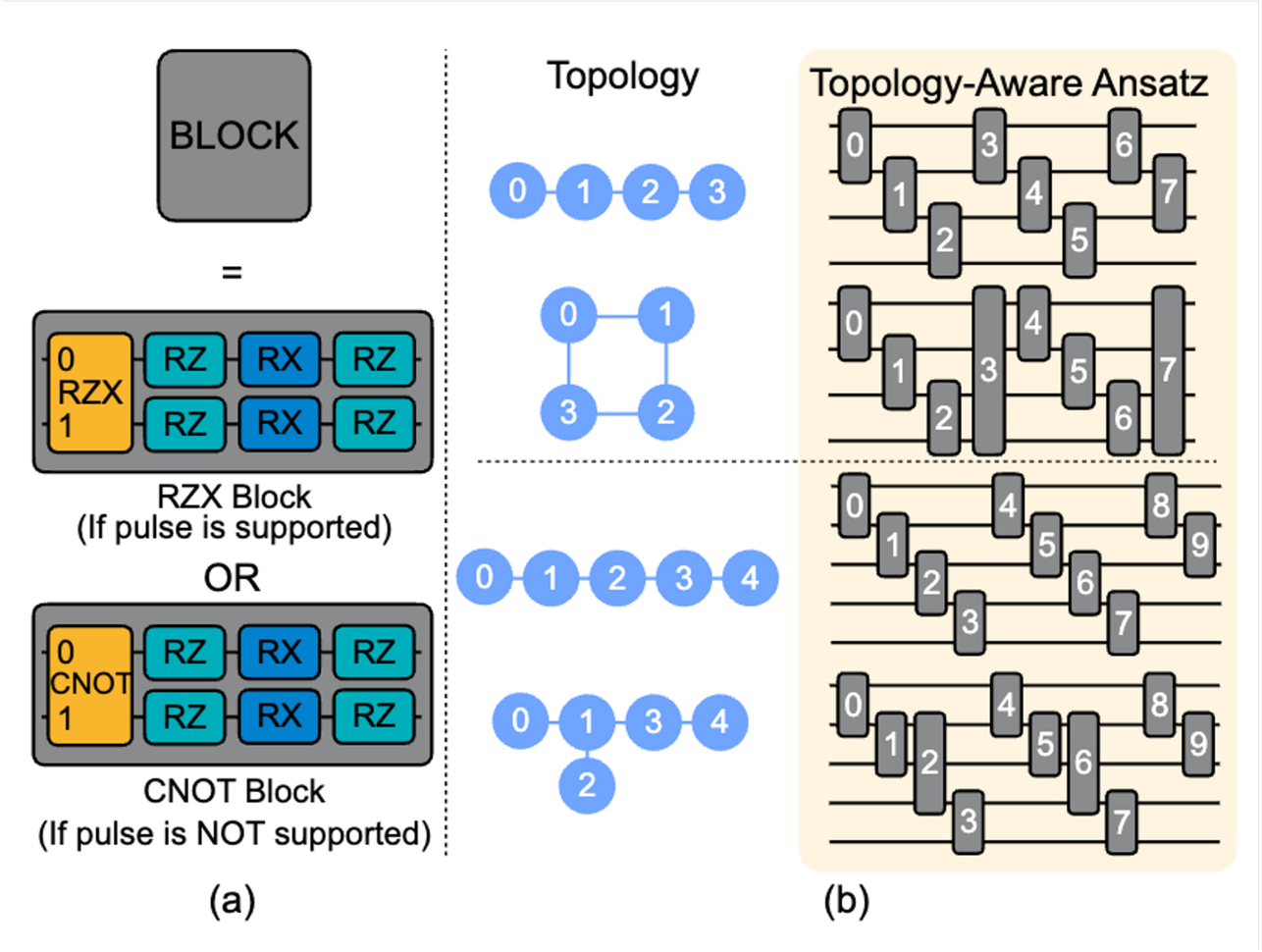
Is the estimated gradient accurate?

- Noise-aware gradients approximated with RobustState are close to the accurate ones computed with the parameter shift rule



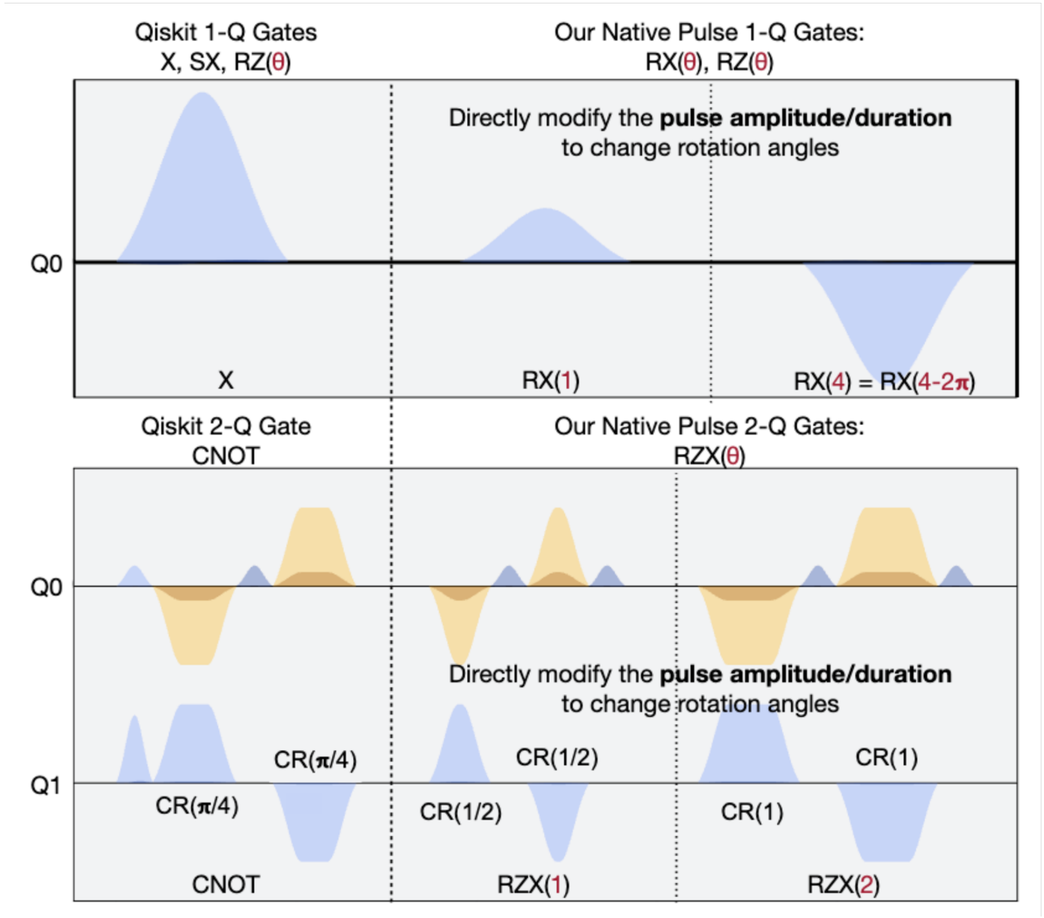
Hardware Efficient Ansatz

- Adapt to the hardware topology
-



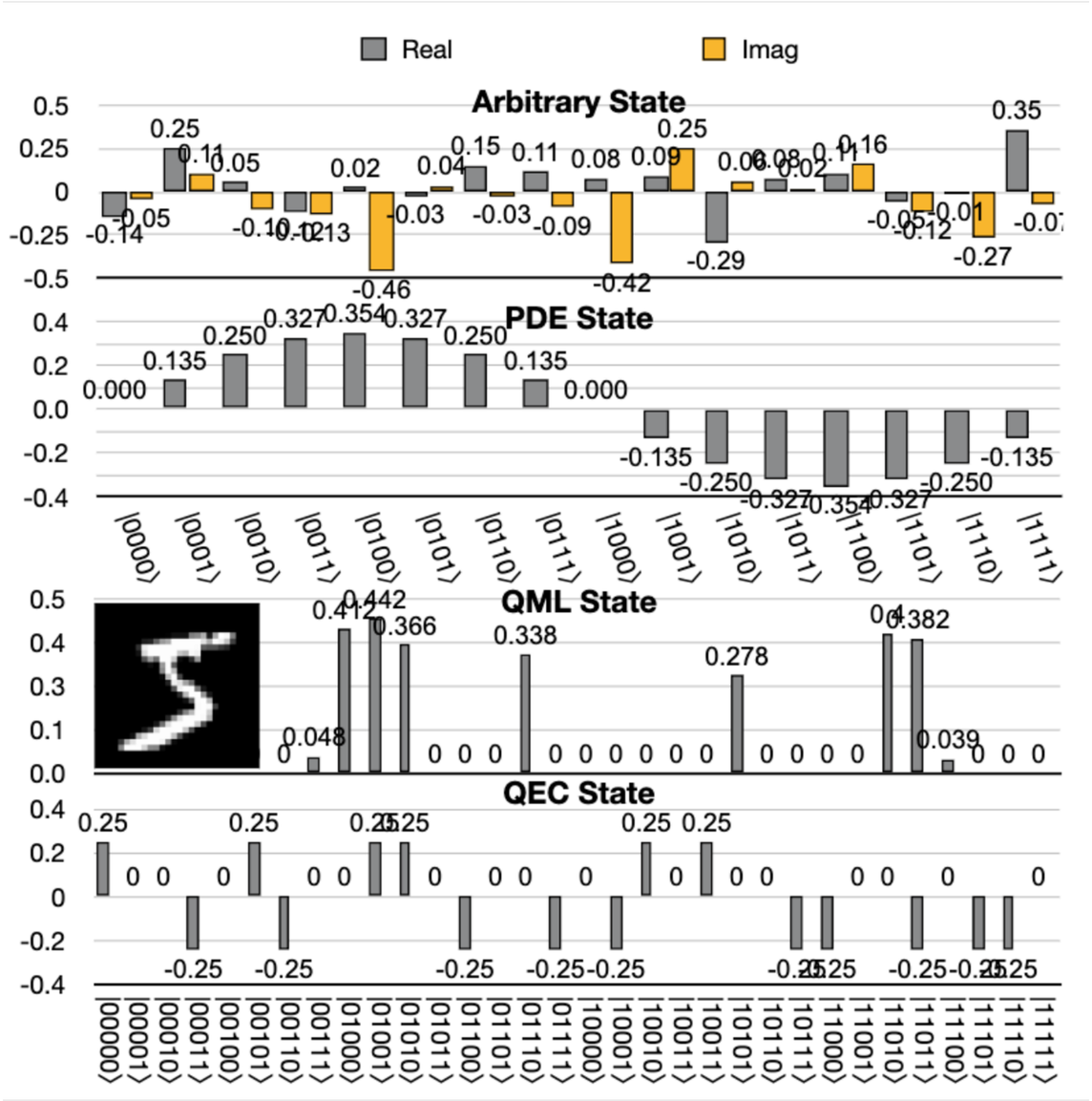
Optimize on the Pulse level

- Scale the pulse magnitude according to the parameter.

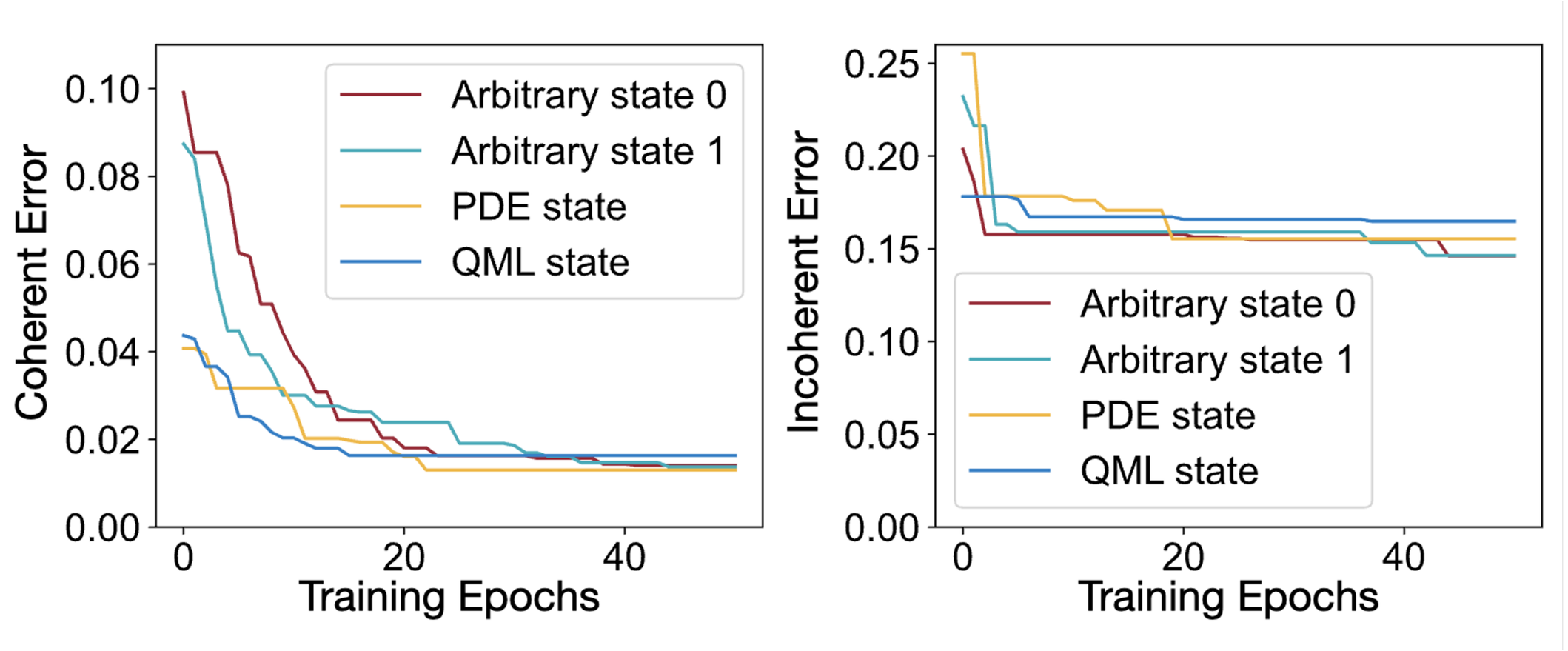


Evaluation

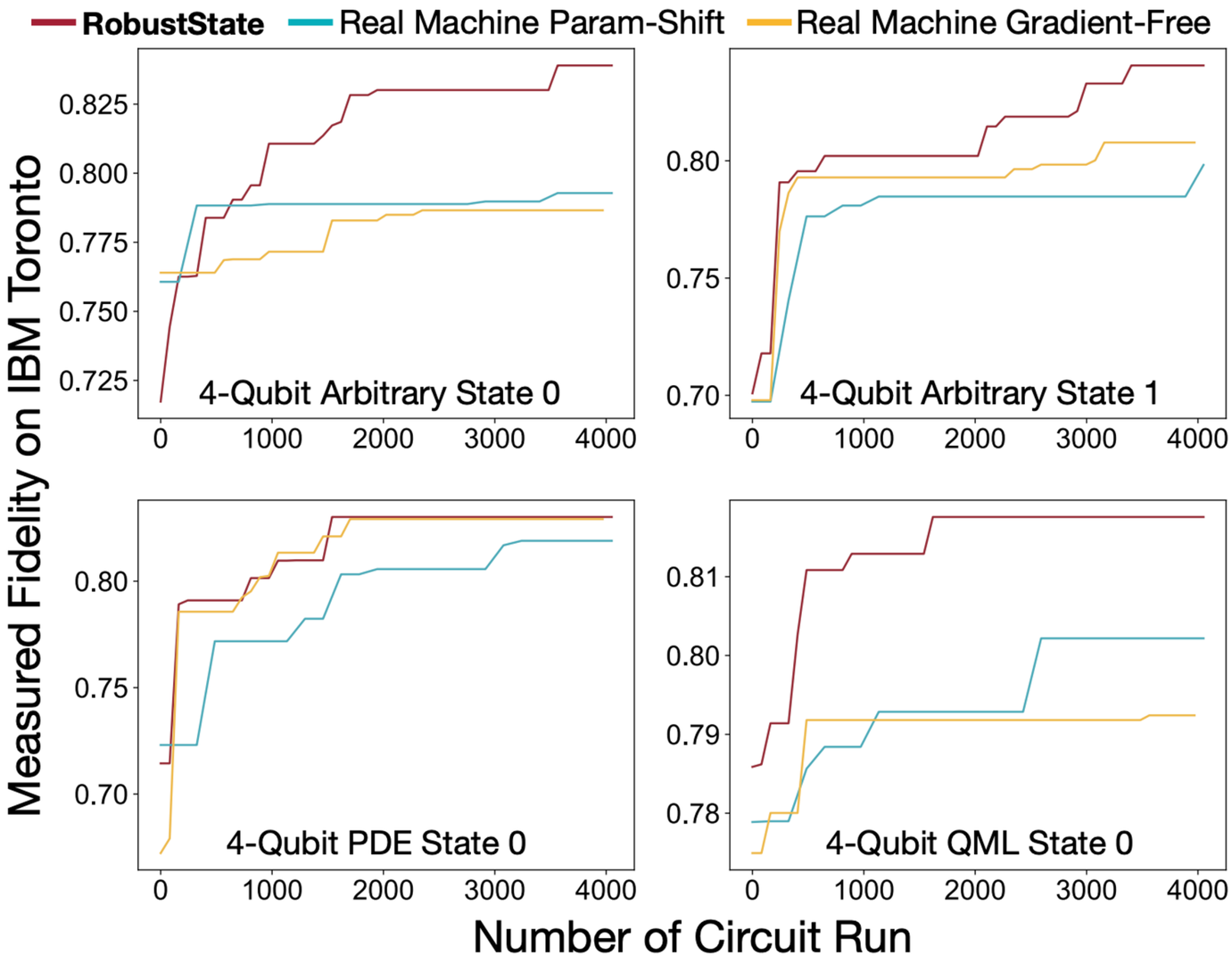
Benchmarks



Reduction of Coherent Errors



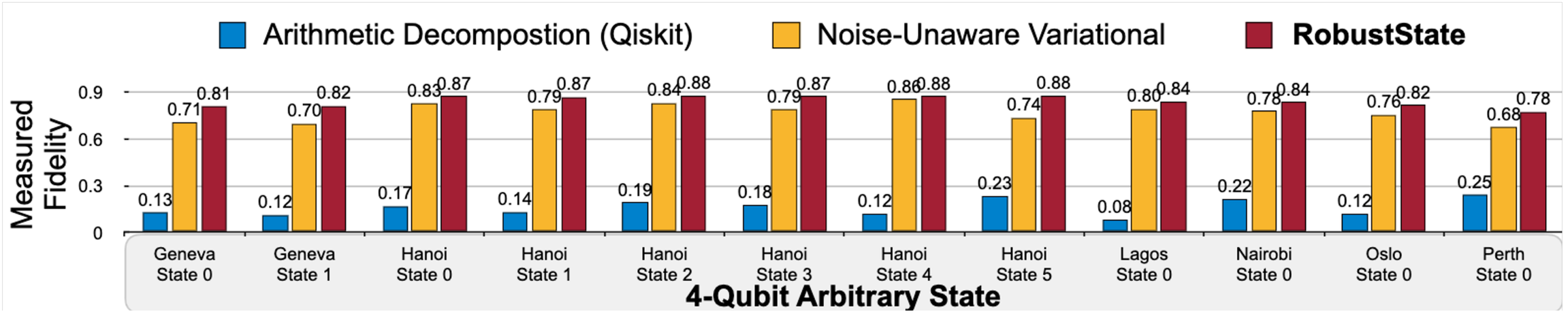
Efficiency over parameter shift



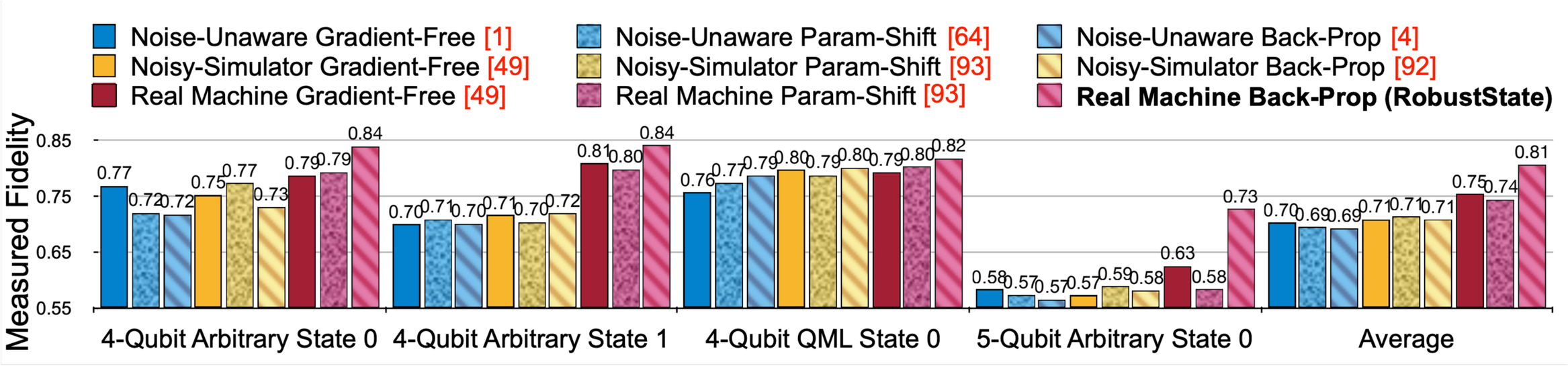
Comparison with arithmetic decomposition

Fidelity	Arbitrary	PDE	QML	Avg.
Mottonen [4], [66]	0.156	0.175	0.269	0.200
Mottonen+SABRE [4], [45], [66]	0.099	0.401	0.299	0.266
Qiskit [36]	0.176	0.277	0.481	0.311
Qiskit + SABRE [45]	0.262	0.266	0.626	0.385
Ours	**0.777**	**0.713**	**0.718**	**0.736**

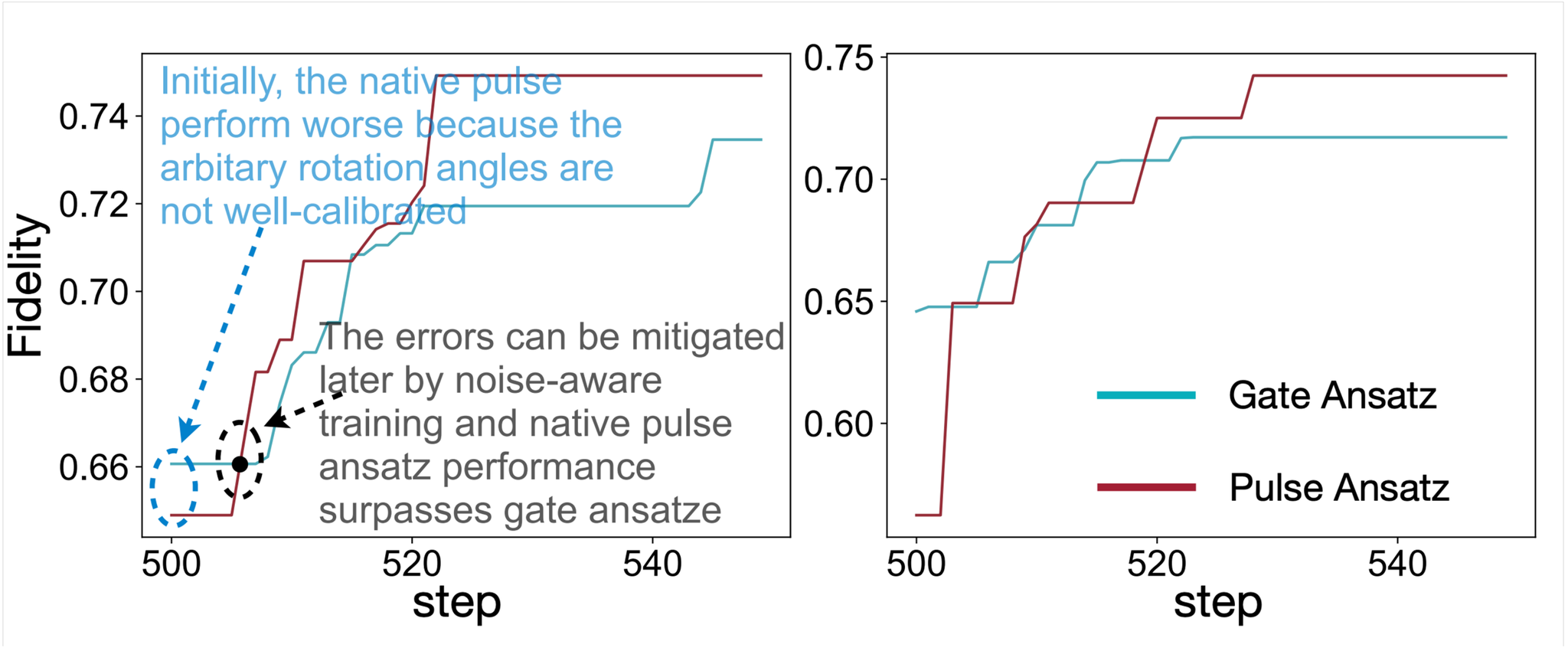
Result on real quantum hardware



Comparison to other robust VQC training methods



Pulse ansatz vs gate ansatz

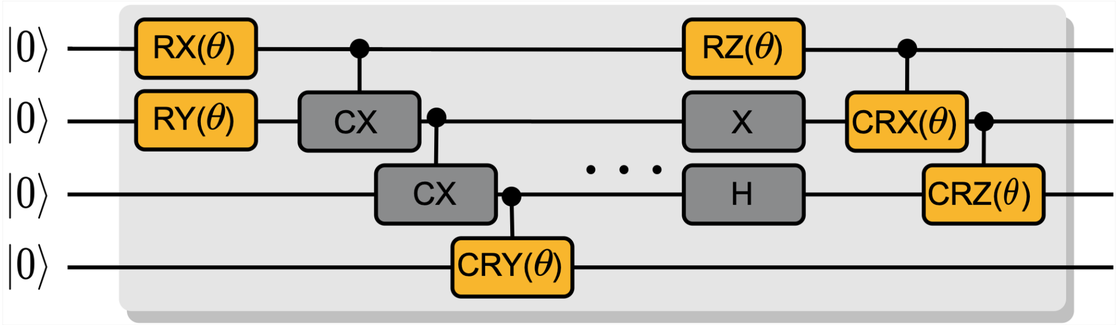


Section 3

Robust Quantum Circuit Architecture Search

Parameterized Quantum Circuits

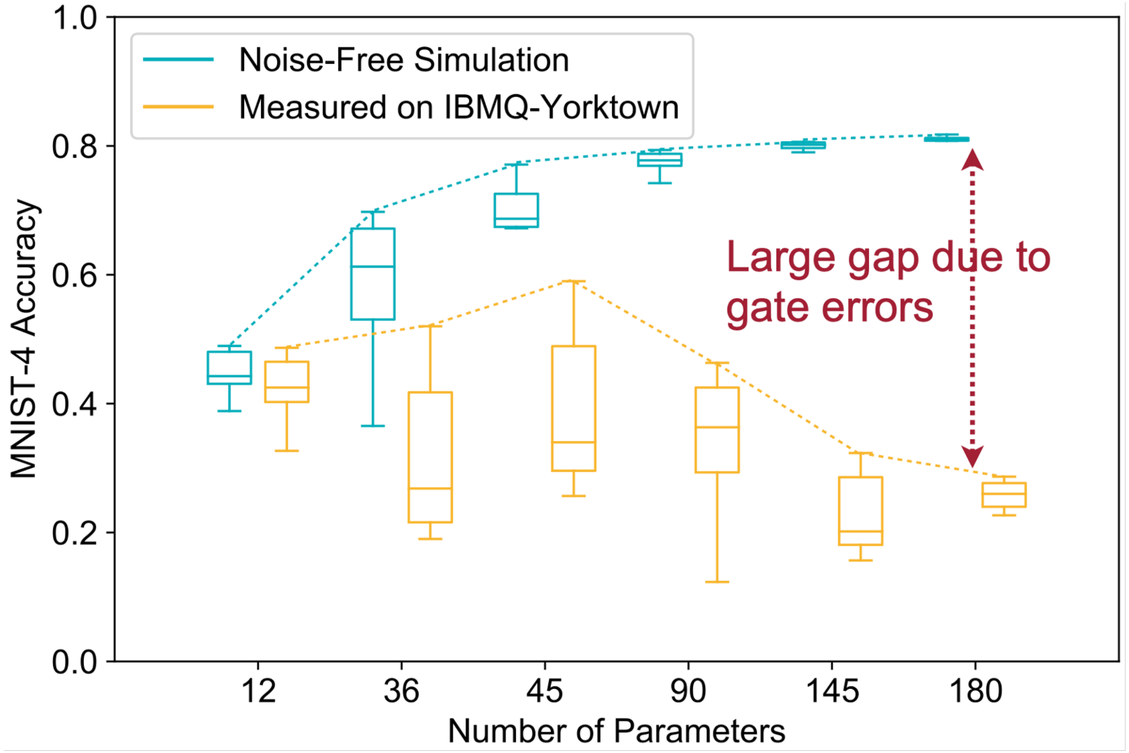
- Parameterized Quantum Circuits (PQC)
- Quantum circuit with fixed gates and parameterized gates



- 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)

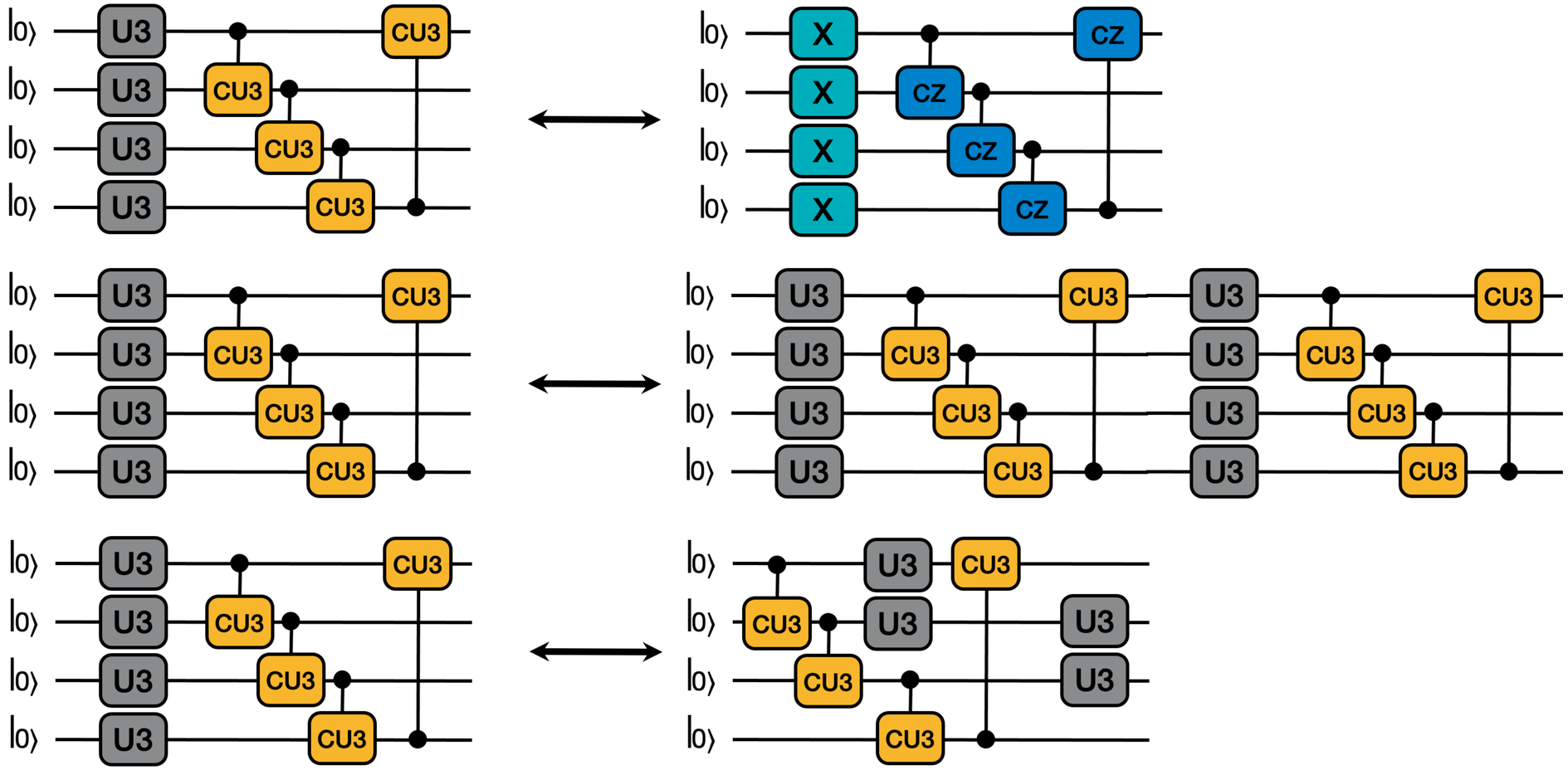
Challenges of PQC — Noise

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



Challenges of PQC — Large Design Space

- Large design space for circuit architecture
  - Type of gates
  - Number of gates
  - Position of gates



QuantumNAS Framework

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

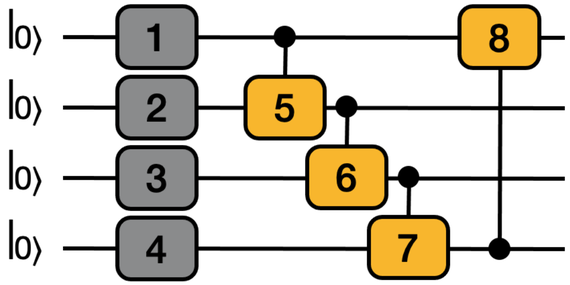
QuantumNAS Framework

Four Steps

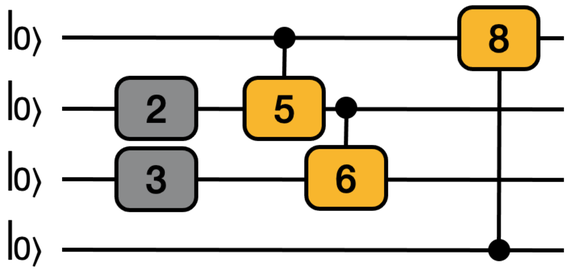
- SuperCircuit Construction and Training
- Noise-Adaptive Evolutionary Co-Search of SubCircuit and Qubit Mapping
- Train the Searched SubCircuit
- Iterative Quantum Gate Pruning

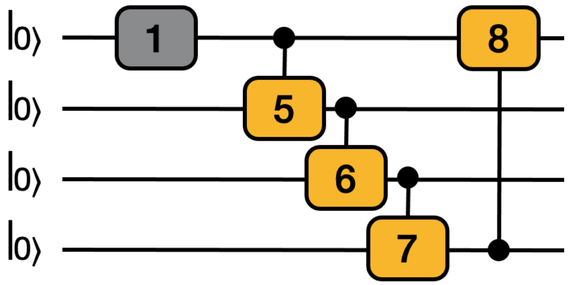
SuperCircuit & SubCircuit

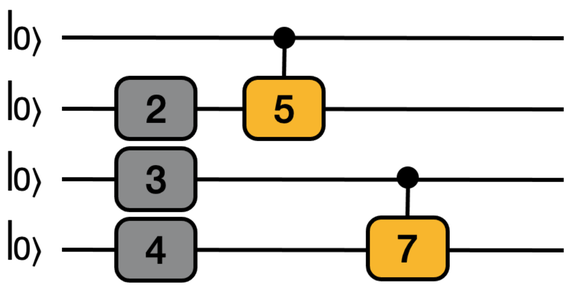
- Firstly construct a design space. For example, a design space of maximum 4 U3 in the first layer and 4 CU3 gates in the second layer
- SuperCircuit: the circuit with the **largest** number of gates in the design space
  - Example: SuperCircuit in U3+CU3 space



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







SuperCircuit Construction

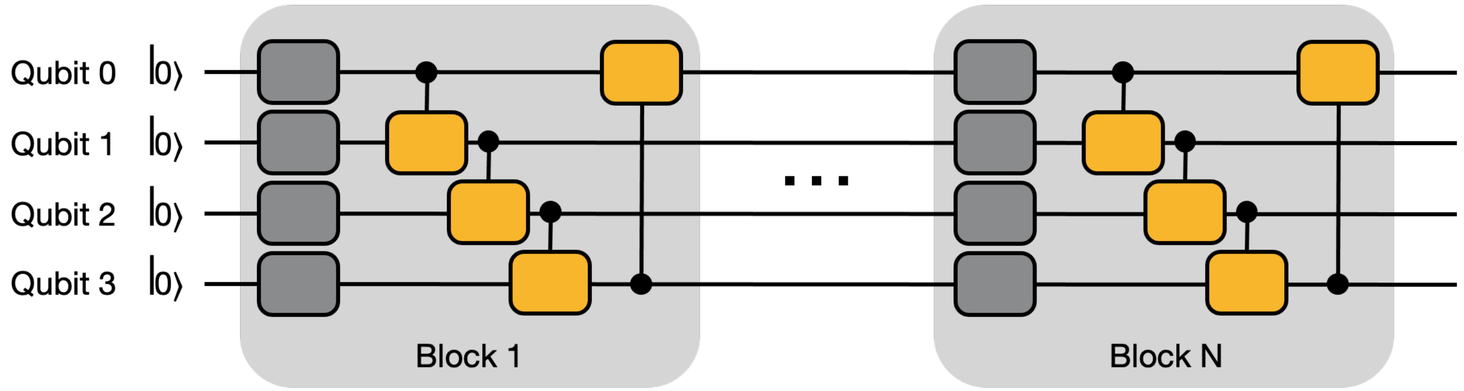
- Why use a SuperCircuit?
  - Enables **efficient** search of architecture candidates without training each
  - SubCircuit inherits parameters from SuperCircuit
  - With **inherited** parameters, we find some good SubCircuits, we find that they **are also good SubCircuits** with parameters **trained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

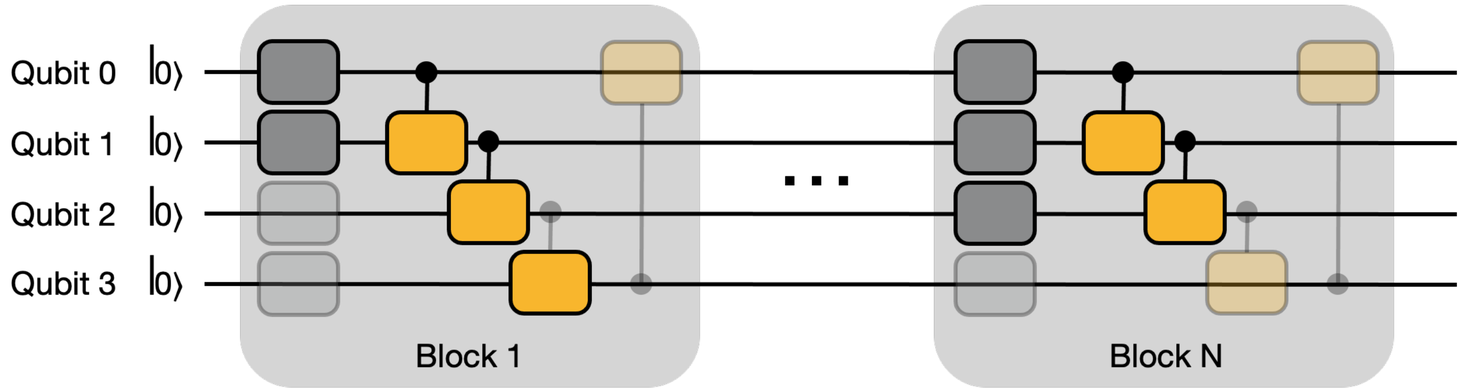
Train SuperCircuit for Multiple Steps

- In one SuperCircuit Training step: Sample and Train



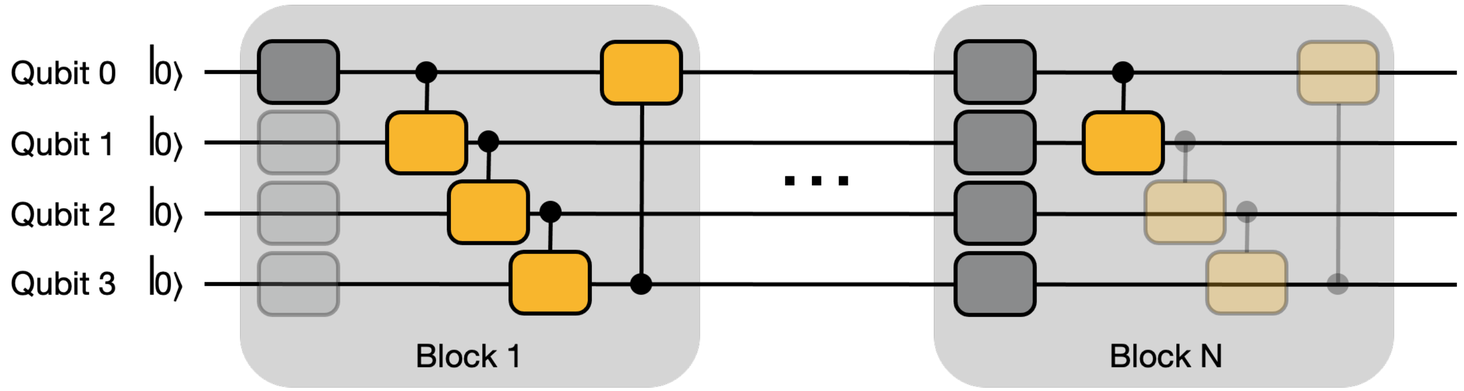
Train SuperCircuit for Multiple Steps

- In one SuperCircuit Training step: Sample and Train



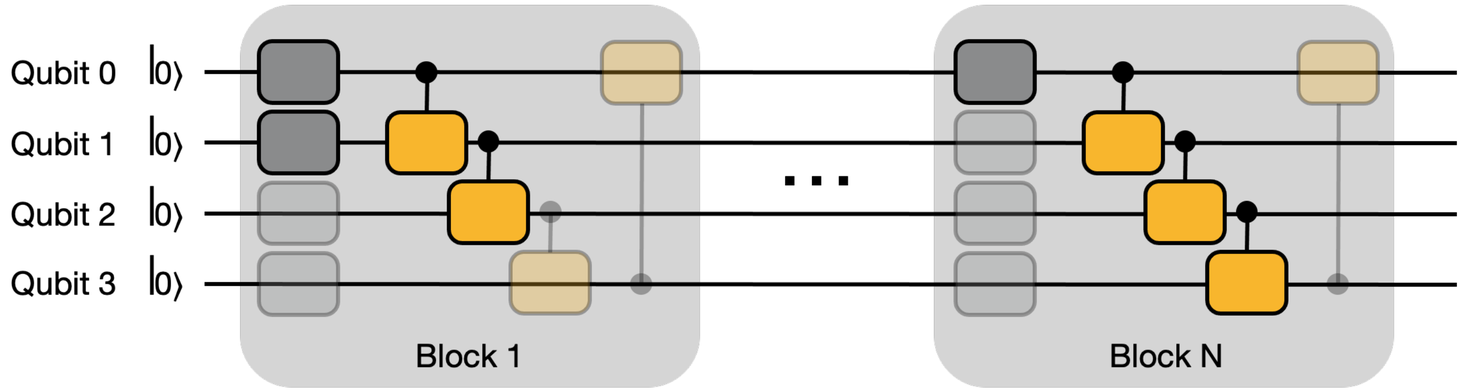
Train SuperCircuit for Multiple Steps

- In one SuperCircuit Training step: Sample and Train



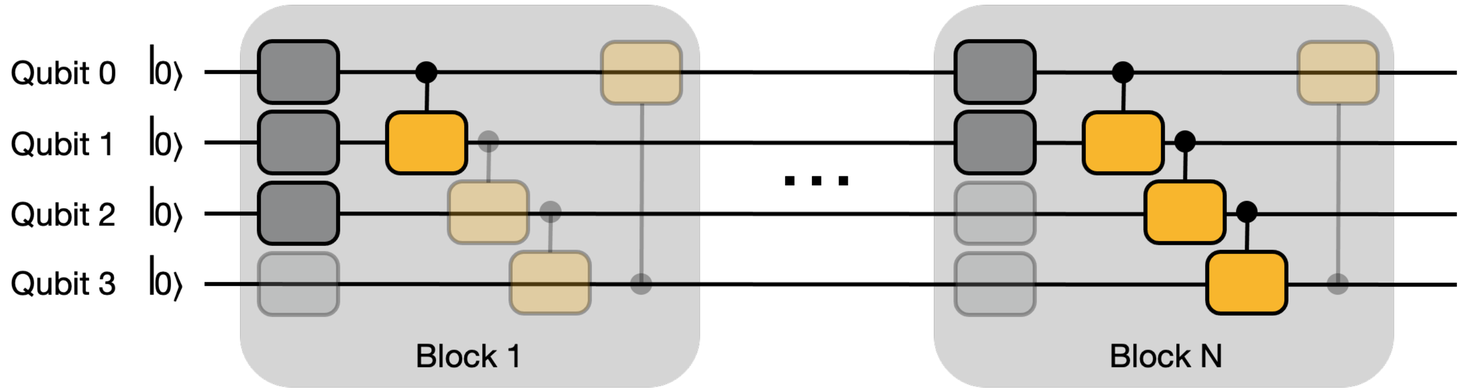
Train SuperCircuit for Multiple Steps

- In one SuperCircuit Training step: Sample and Train



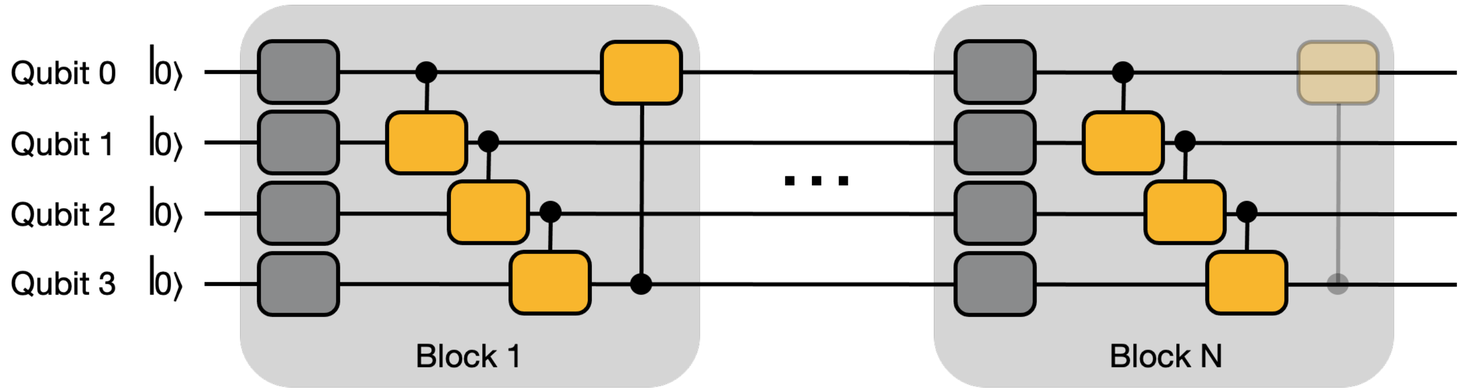
Train SuperCircuit for Multiple Steps

- In one SuperCircuit Training step: Sample and Train



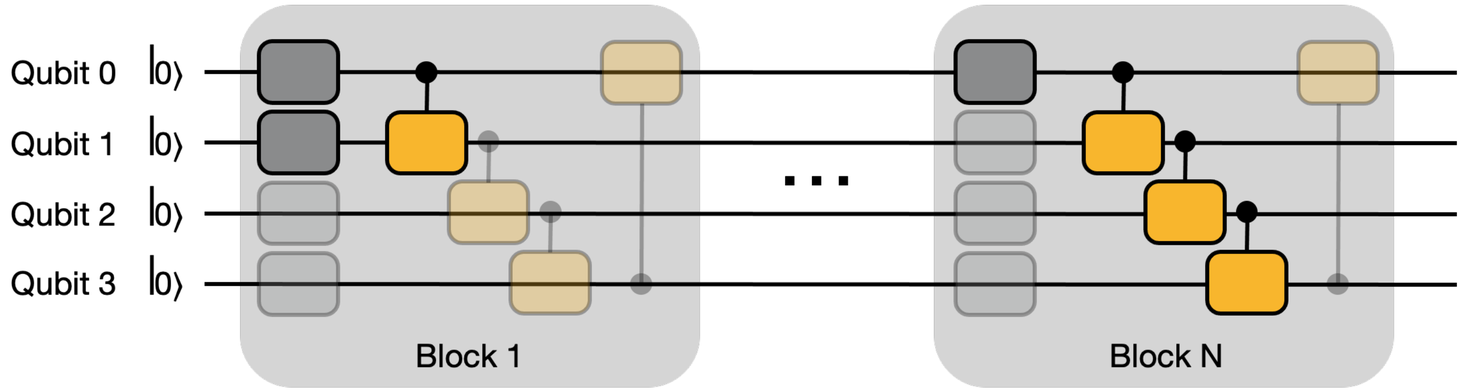
Train SuperCircuit for Multiple Steps

- In one SuperCircuit Training step: Sample and Train



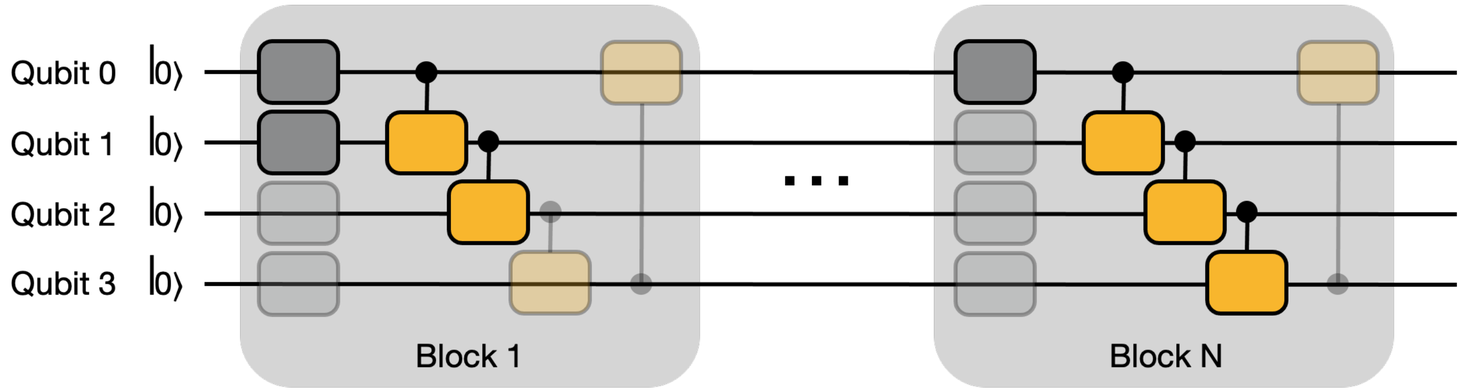
Train SuperCircuit for Multiple Steps

- In one SuperCircuit Training step: Sample and Train



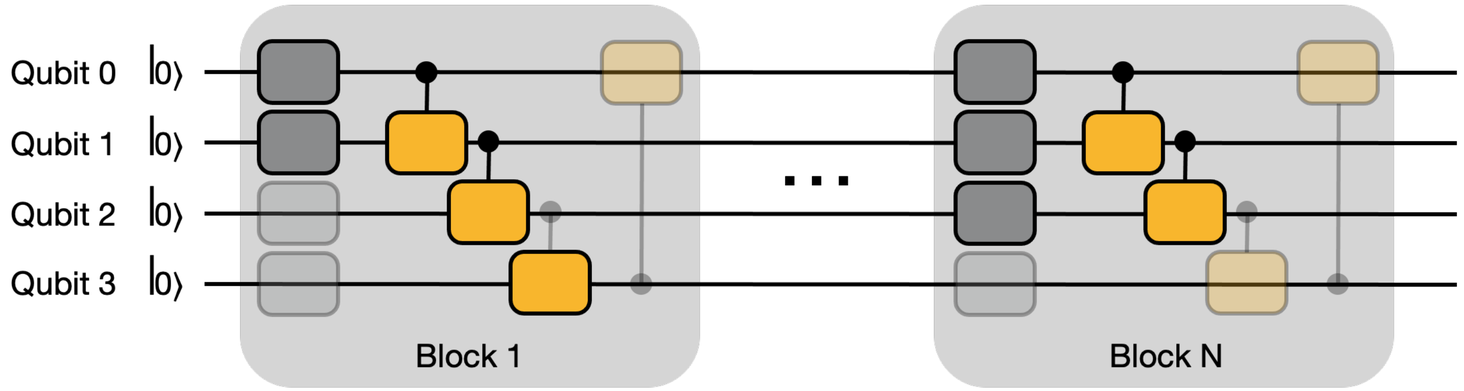
Train SuperCircuit for Multiple Steps

- In one SuperCircuit Training step: Sample and Train



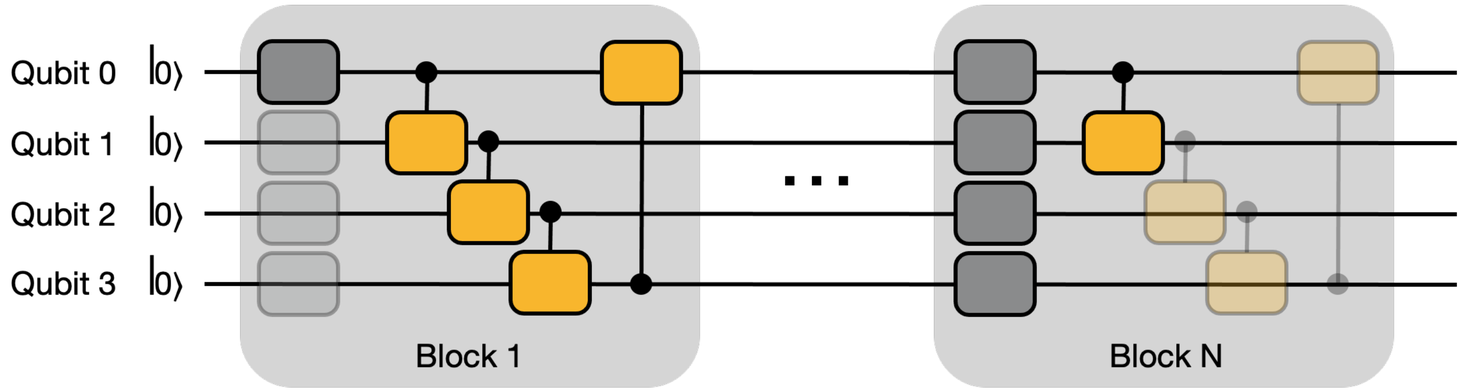
Train SuperCircuit for Multiple Steps

- In one SuperCircuit Training step: Sample and Train



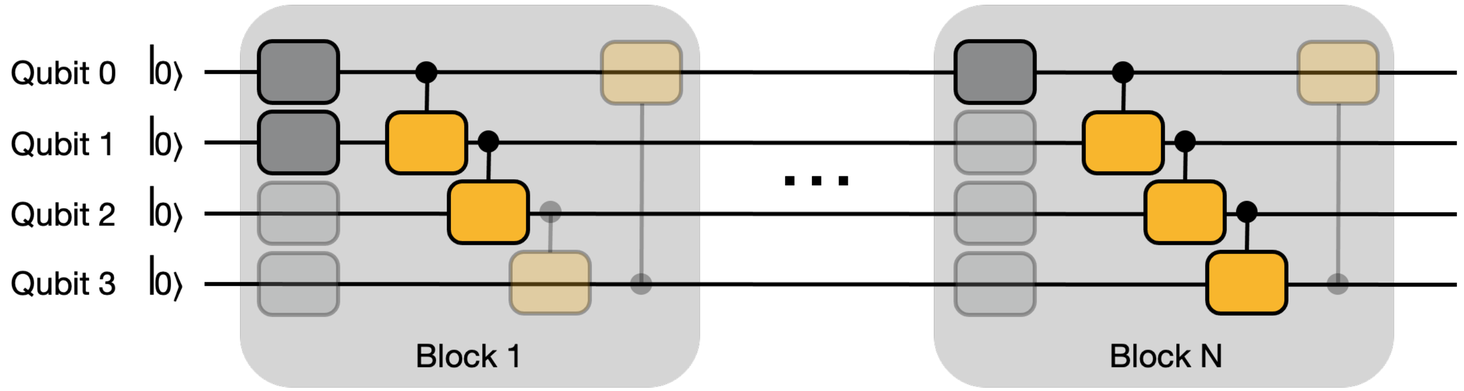
Train SuperCircuit for Multiple Steps

- In one SuperCircuit Training step: Sample and Train



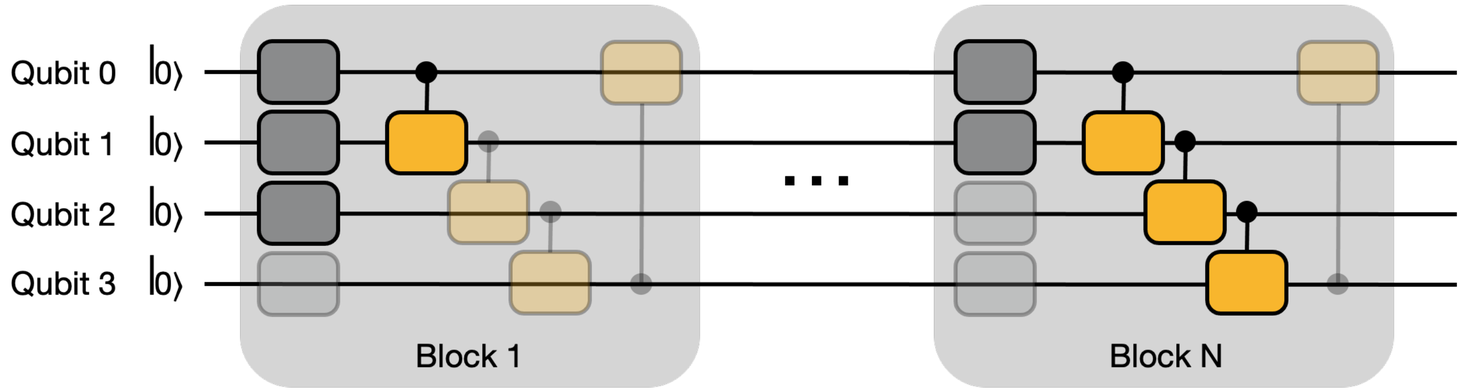
Train SuperCircuit for Multiple Steps

- In one SuperCircuit Training step: Sample and Train



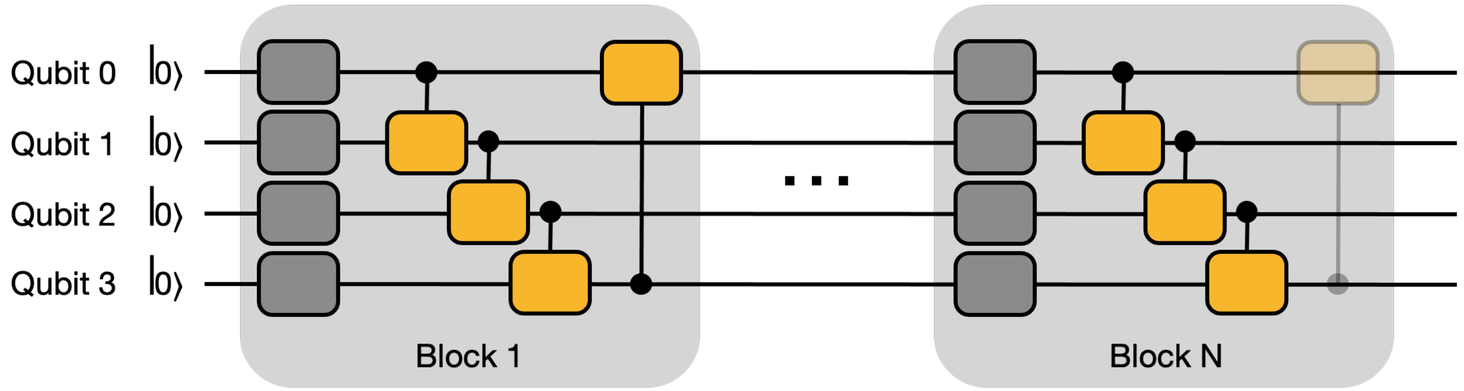
Train SuperCircuit for Multiple Steps

- In one SuperCircuit Training step: Sample and Train



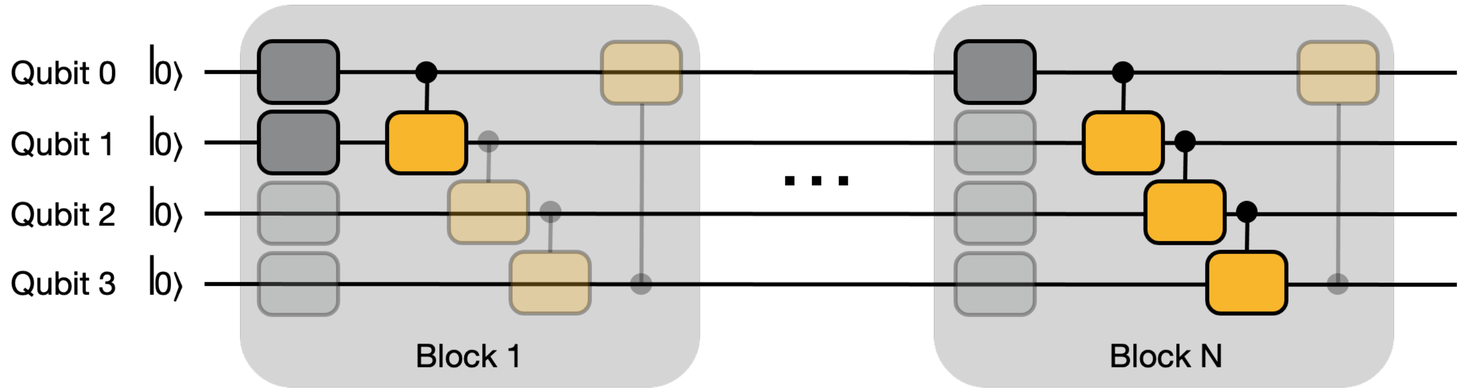
Train SuperCircuit for Multiple Steps

- In one SuperCircuit Training step: Sample and Train



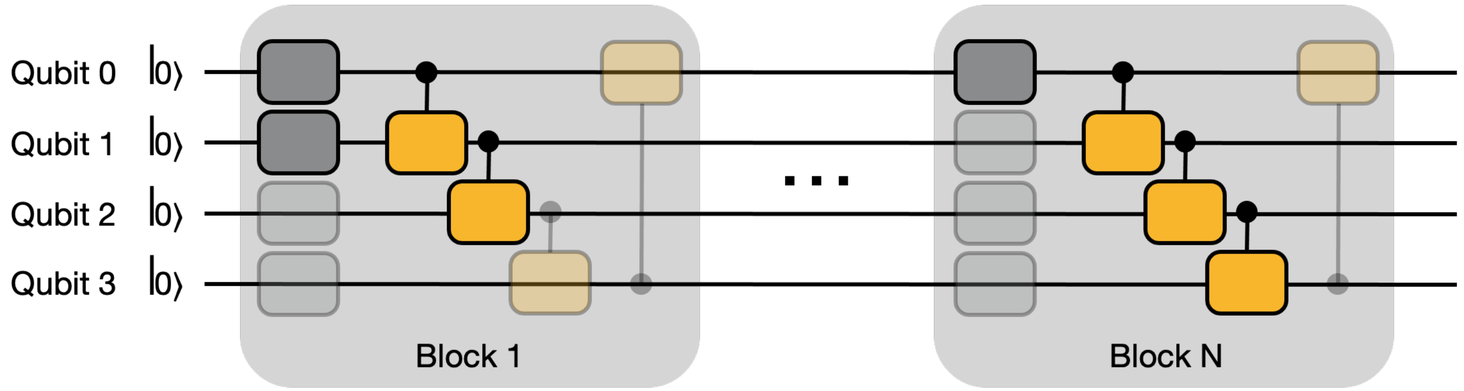
Train SuperCircuit for Multiple Steps

- In one SuperCircuit Training step: Sample and Train



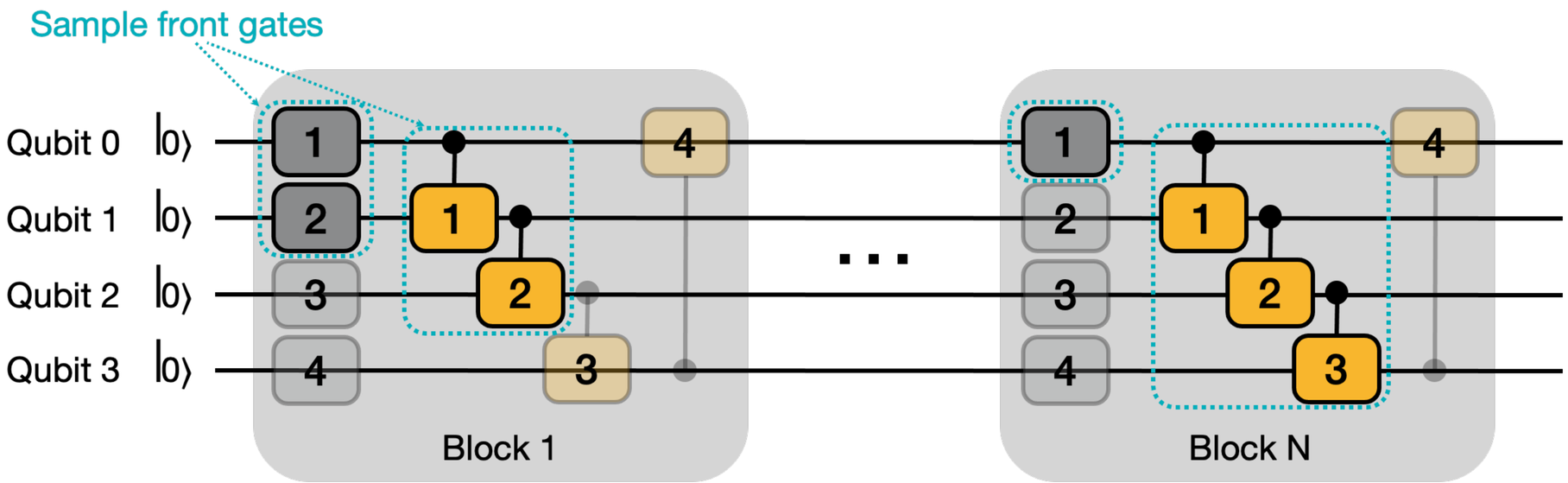
Train SuperCircuit for Multiple Steps

- In one SuperCircuit Training step: Sample and Train



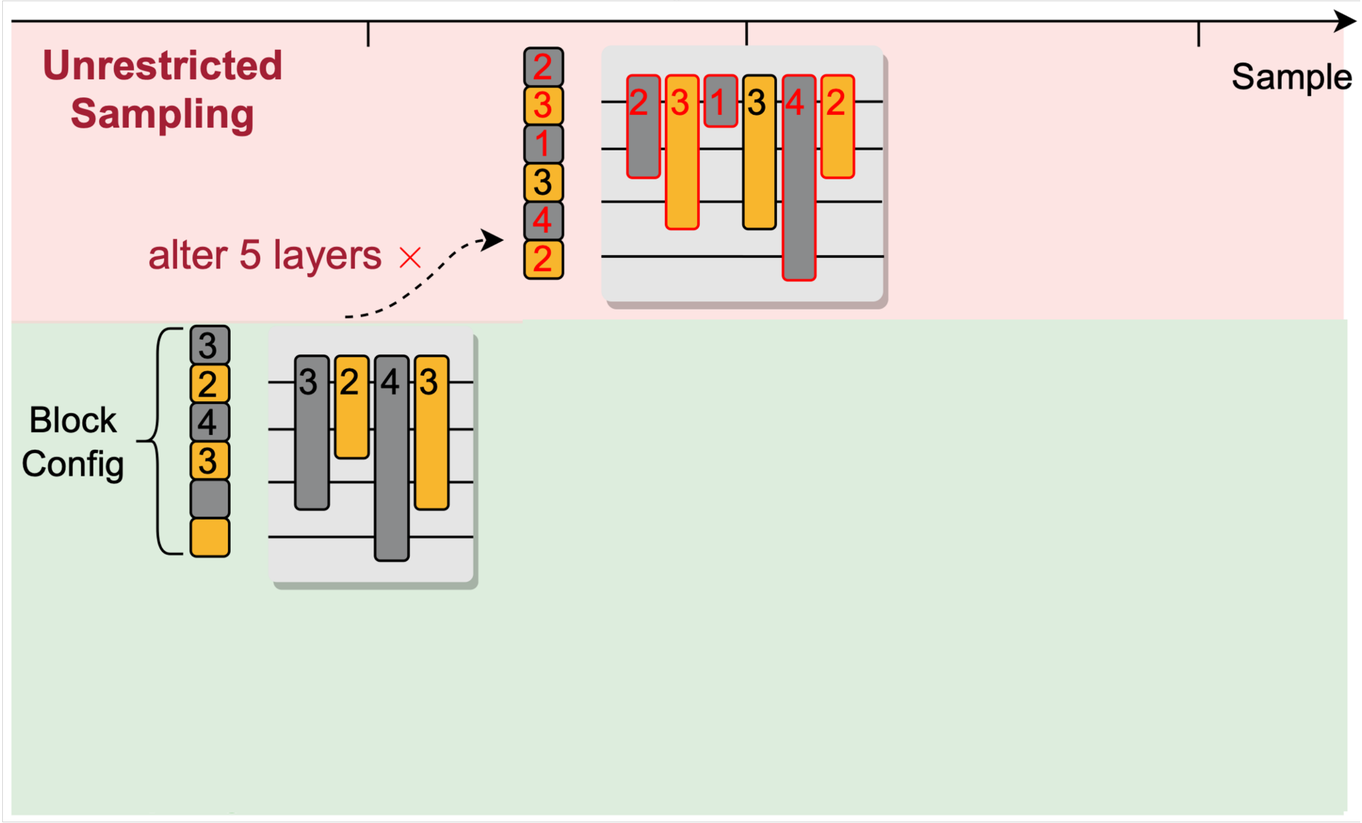
Front Sampling

- 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
-



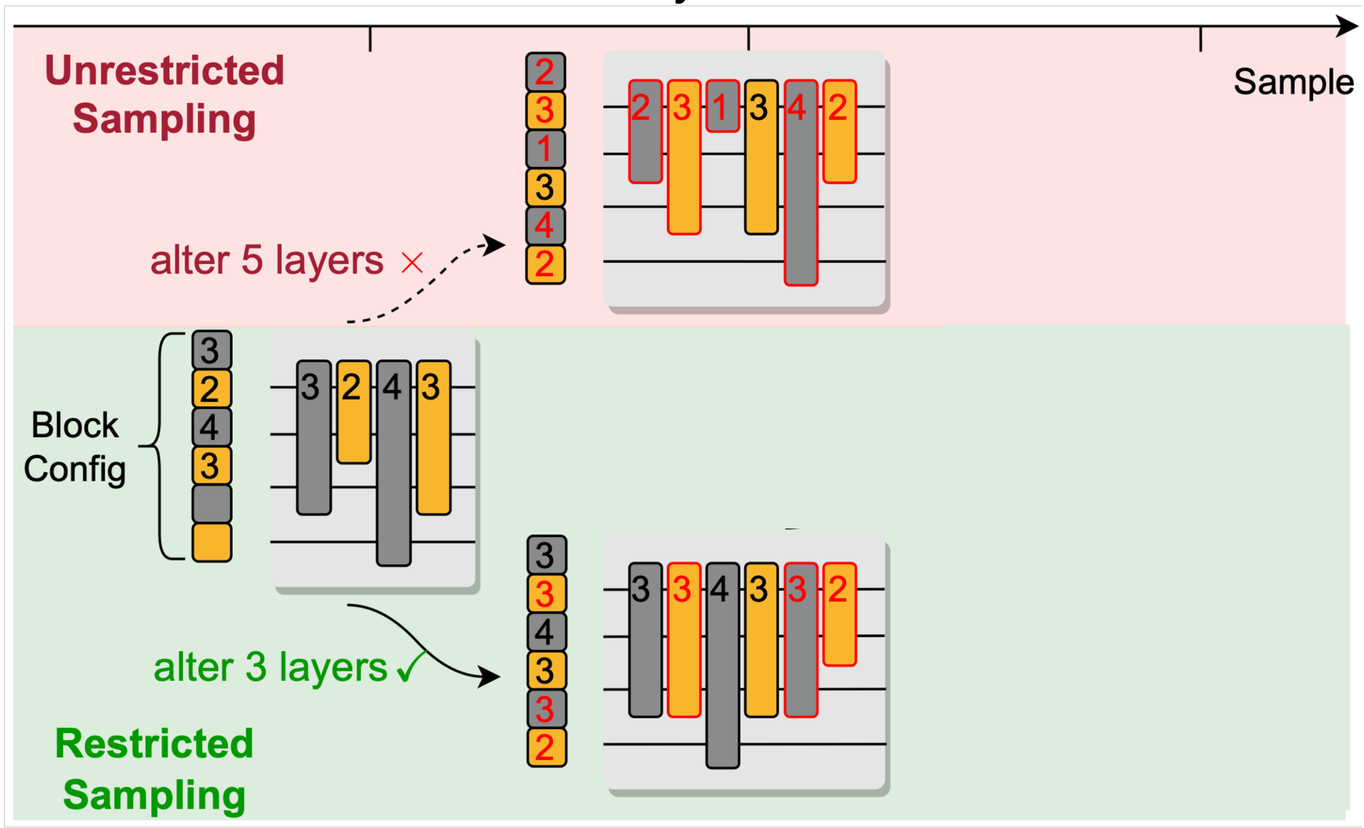
Restricted Sampling

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



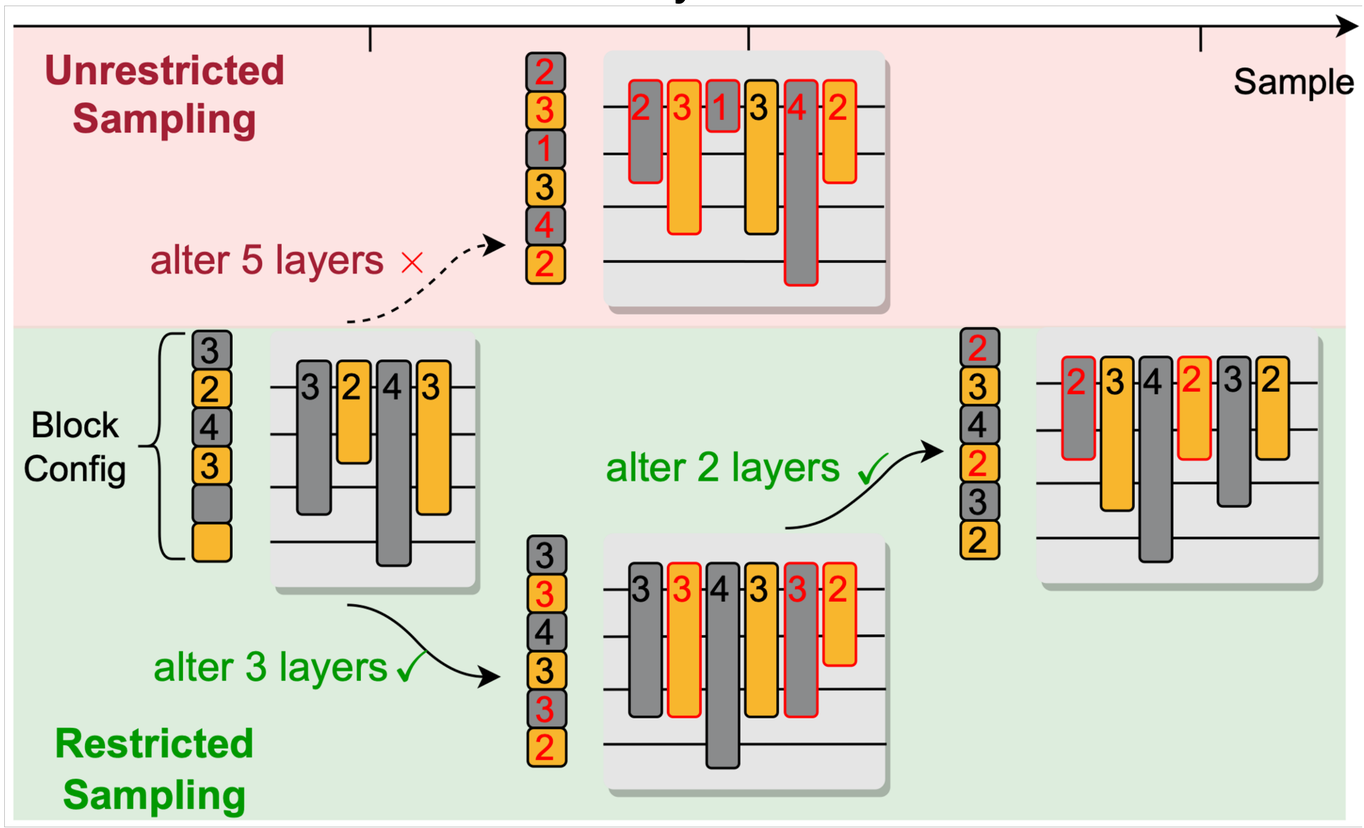
Restricted Sampling

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

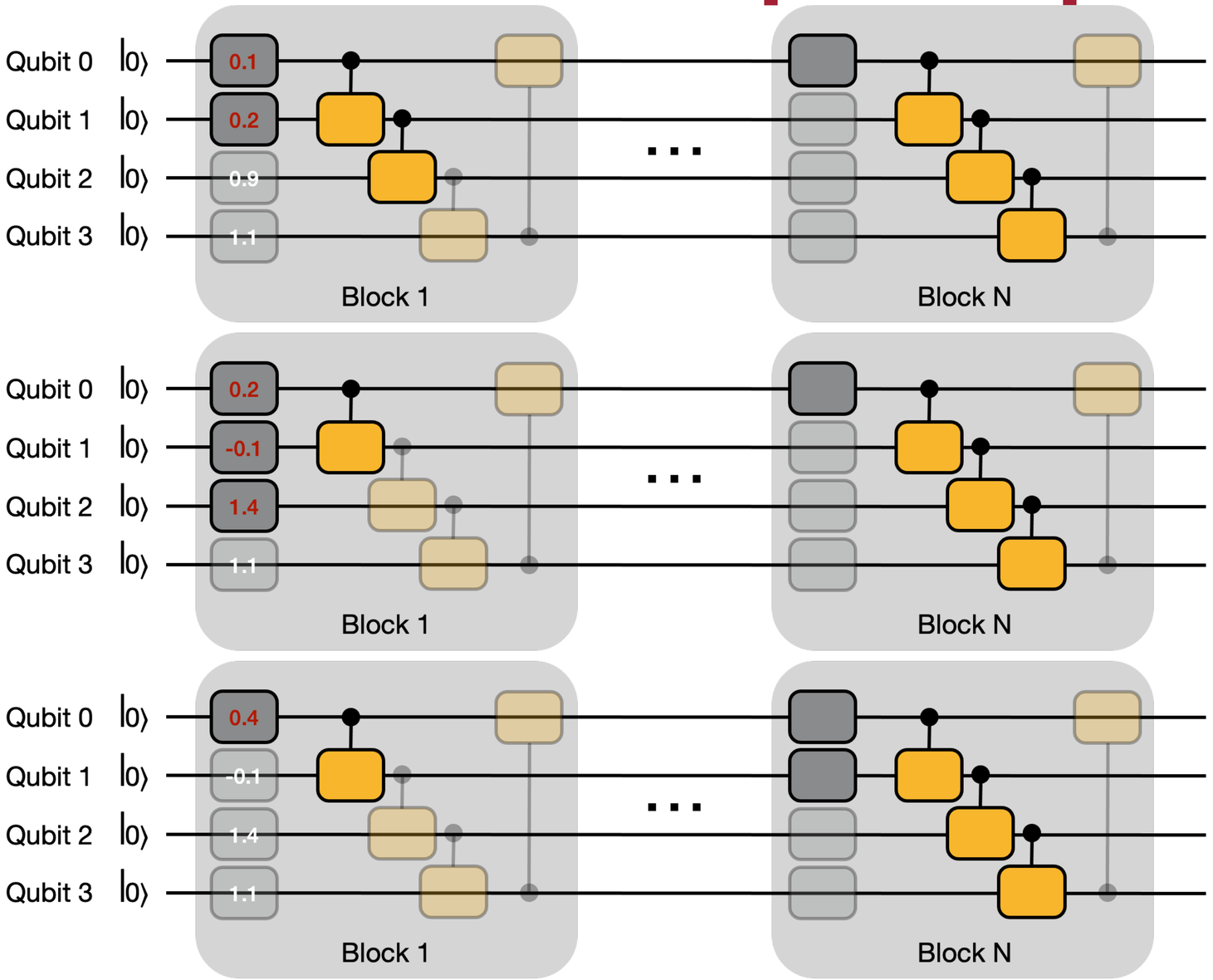


Restricted Sampling

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

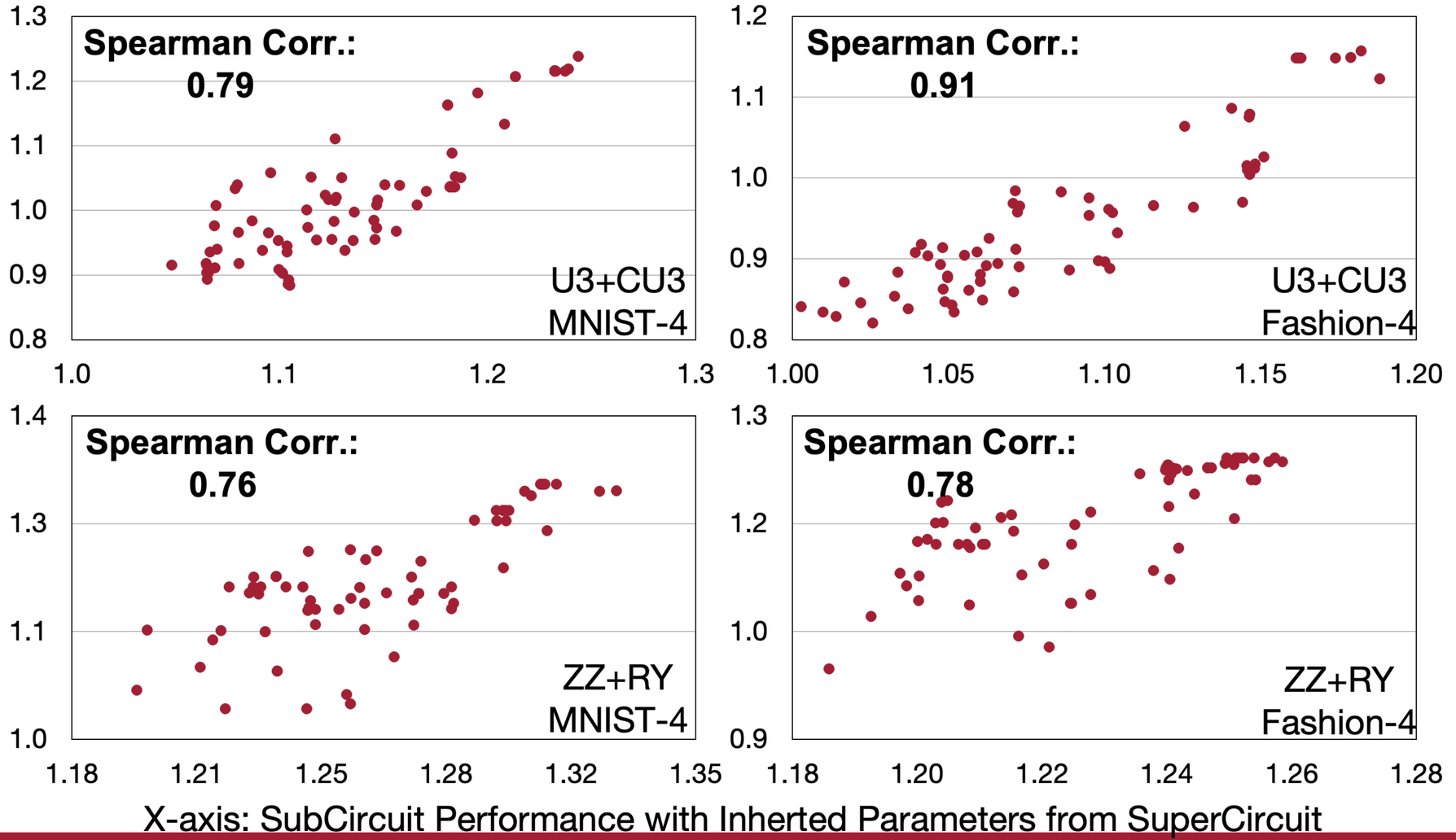


Train SuperCircuit for Multiple Steps



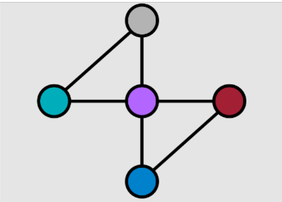
How Reliable is the SuperCircuit?

- Inherited parameters from SuperCircuit can provide accurate **relative performance**



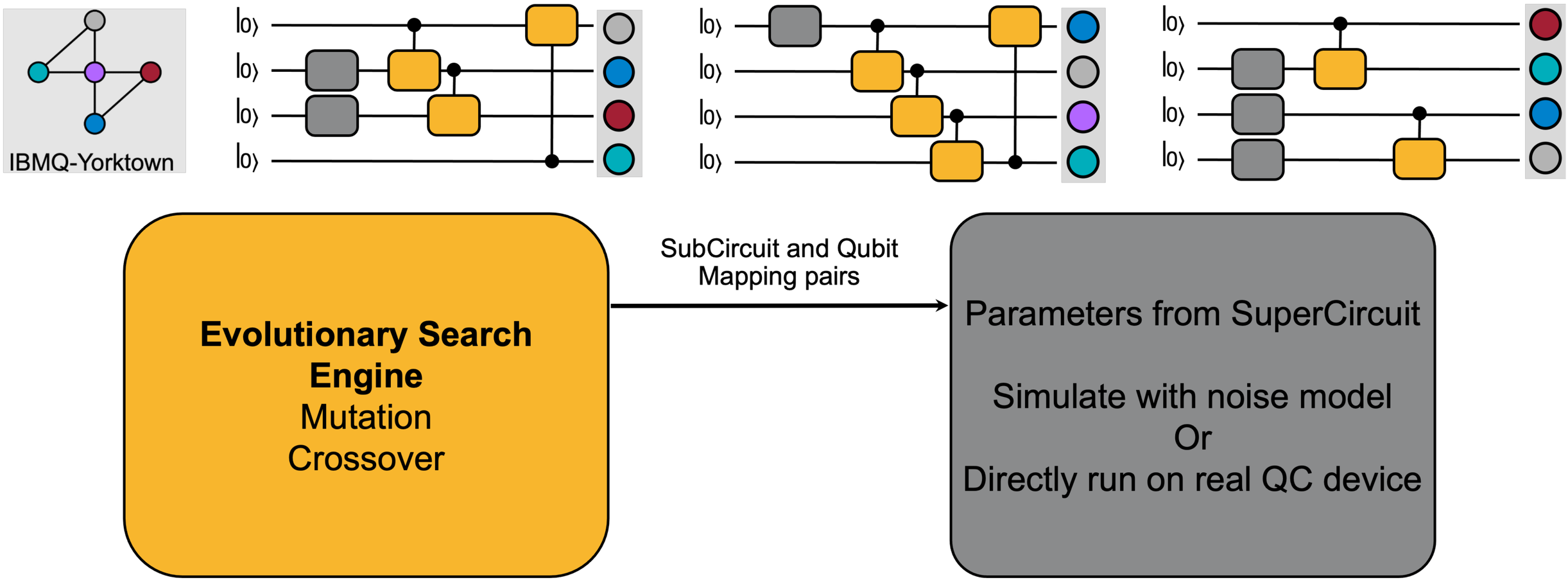
Noise-Adaptive Evolutionary Search

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



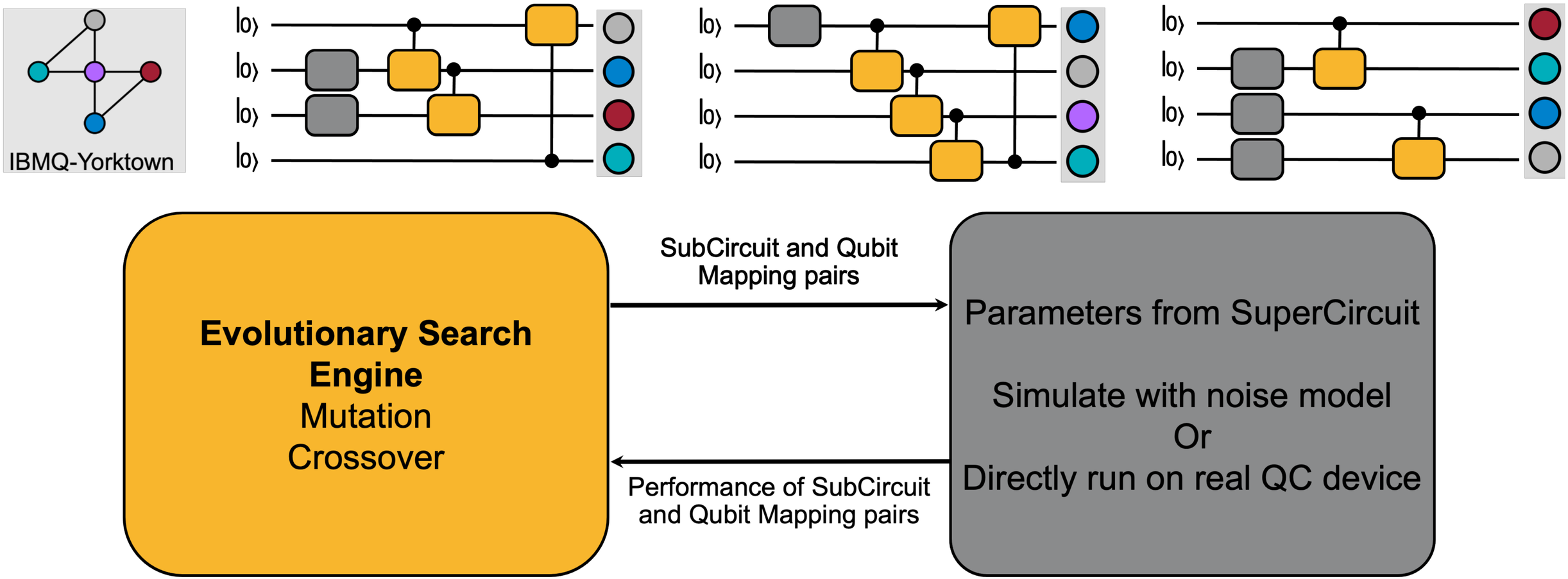
Noise-Adaptive Evolutionary Search

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



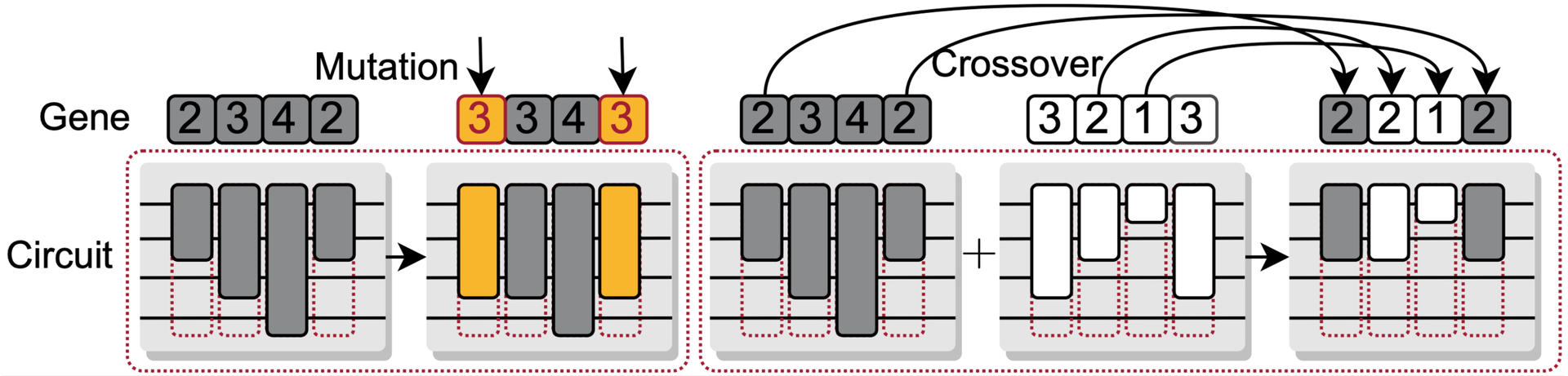
Noise-Adaptive Evolutionary Search

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



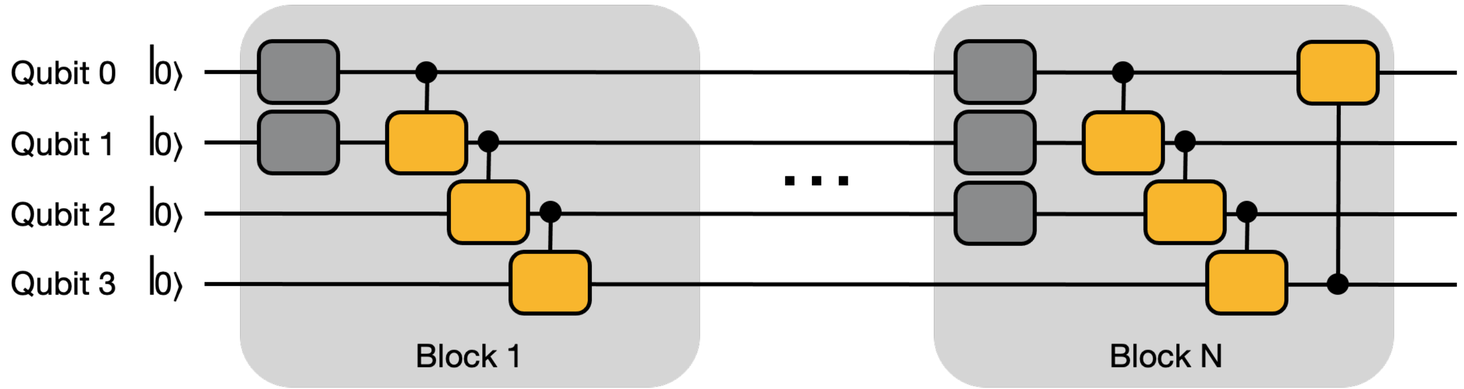
Mutation and Crossover

- Mutation and crossover create new SubCircuit candidates



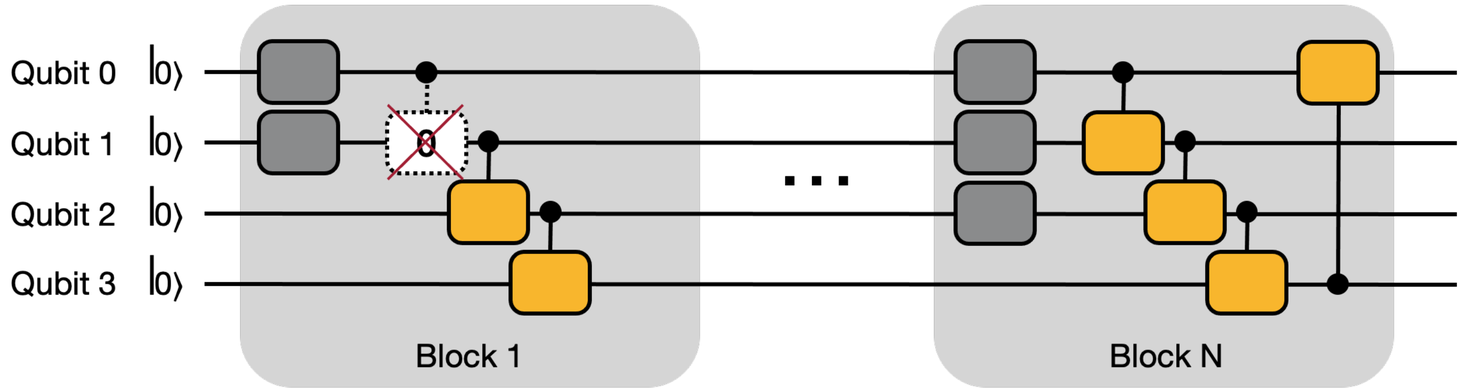
Iterative Quantum Gate Pruning

- 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



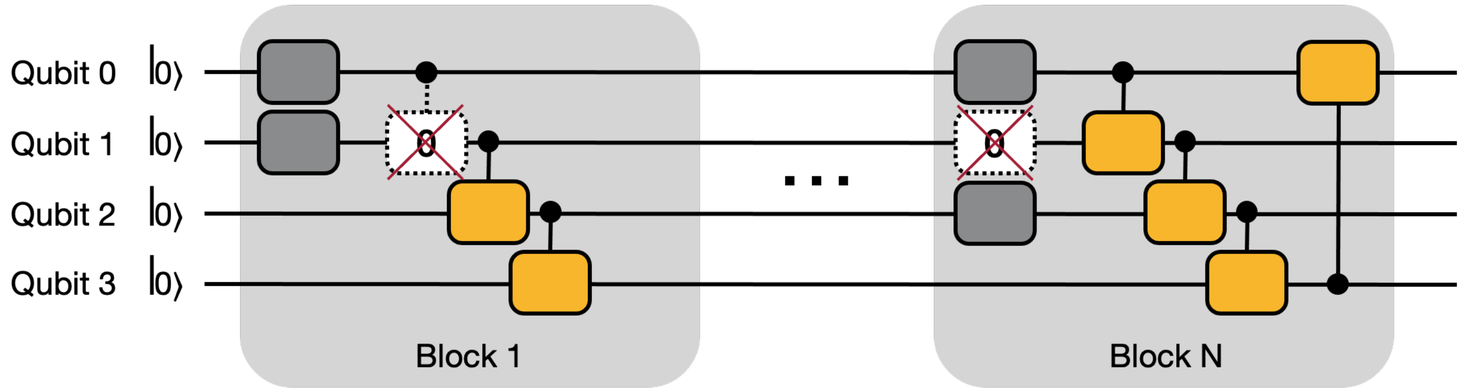
Iterative Quantum Gate Pruning

- 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



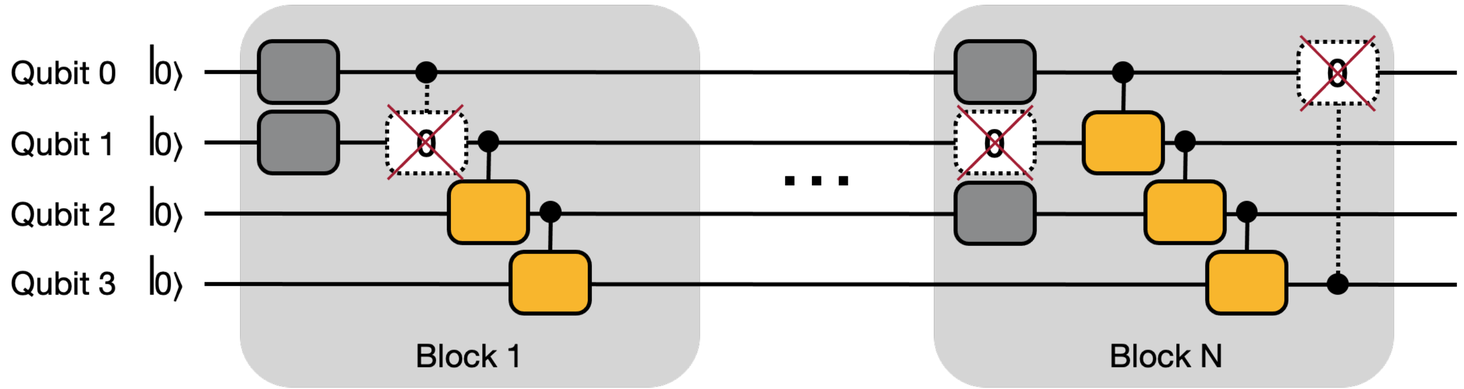
Iterative Quantum Gate Pruning

- 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



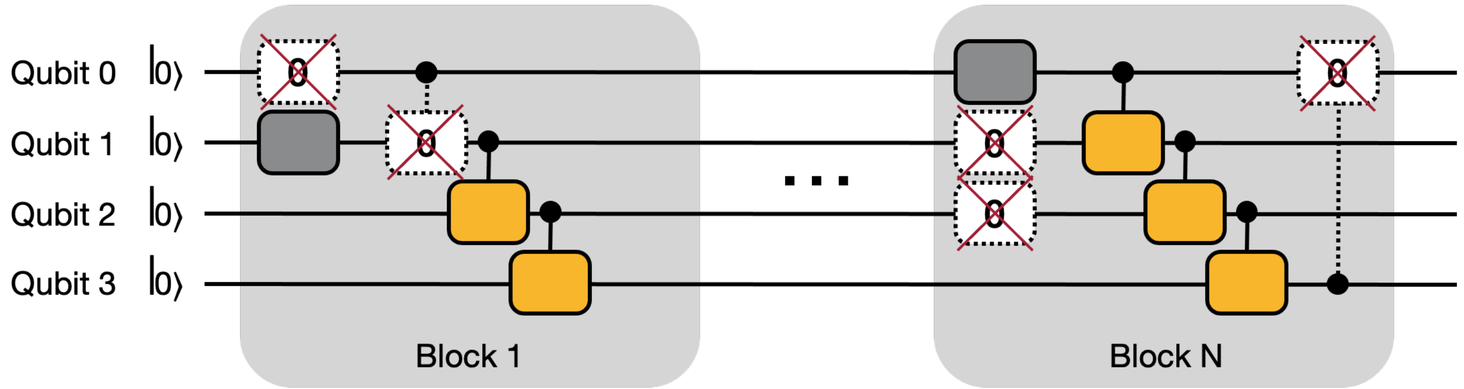
Iterative Quantum Gate Pruning

- 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 Quantum Gate 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

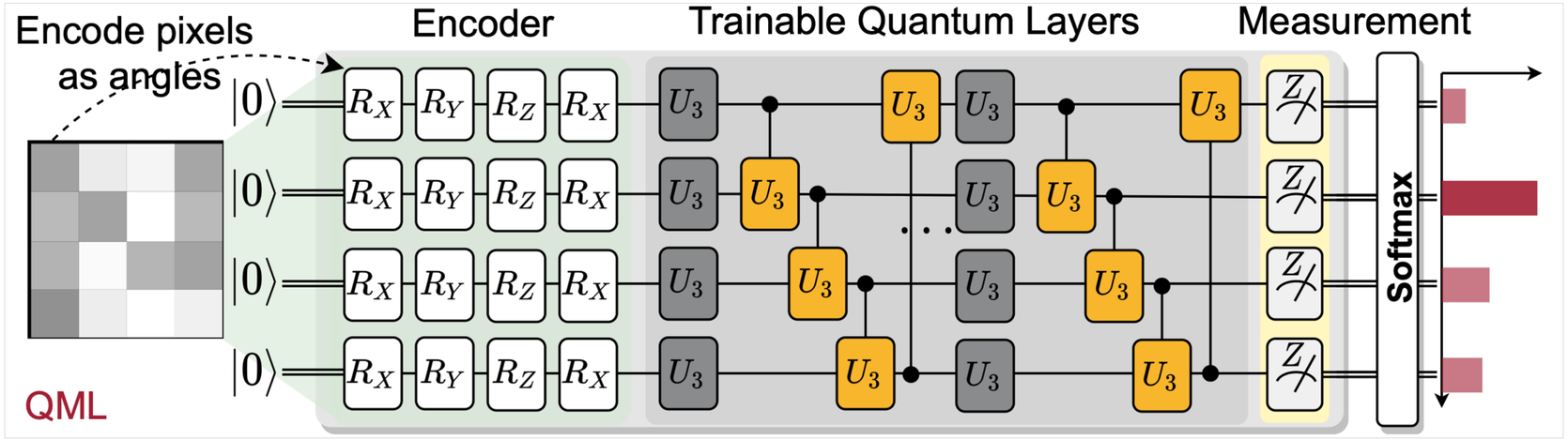


Evaluation

- 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

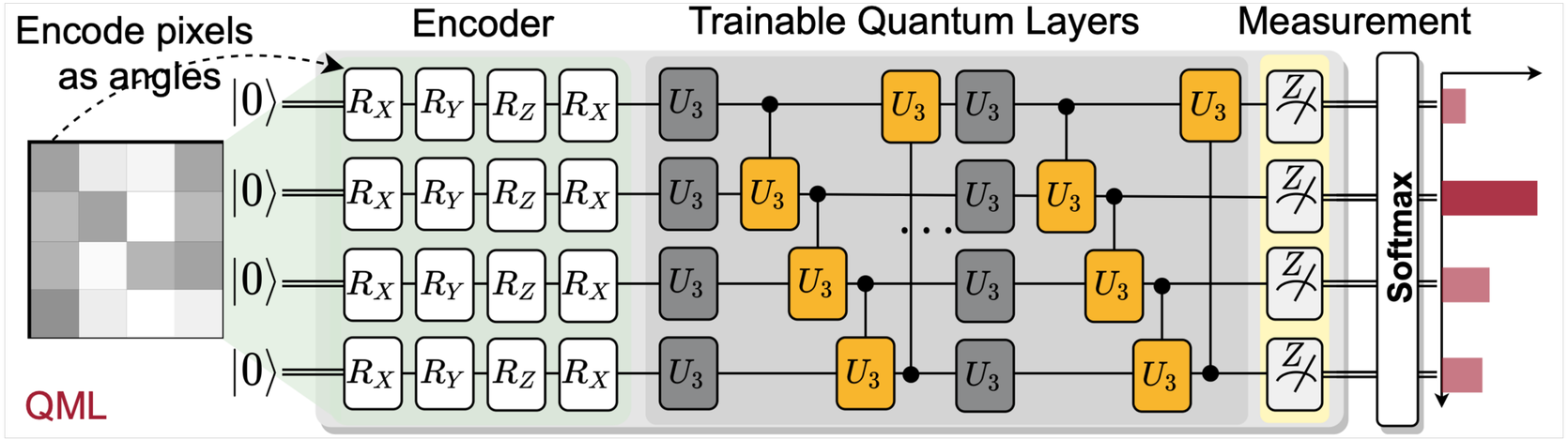
Benchmarks: QNN and VQE

- Quantum Neural Networks: classification
-

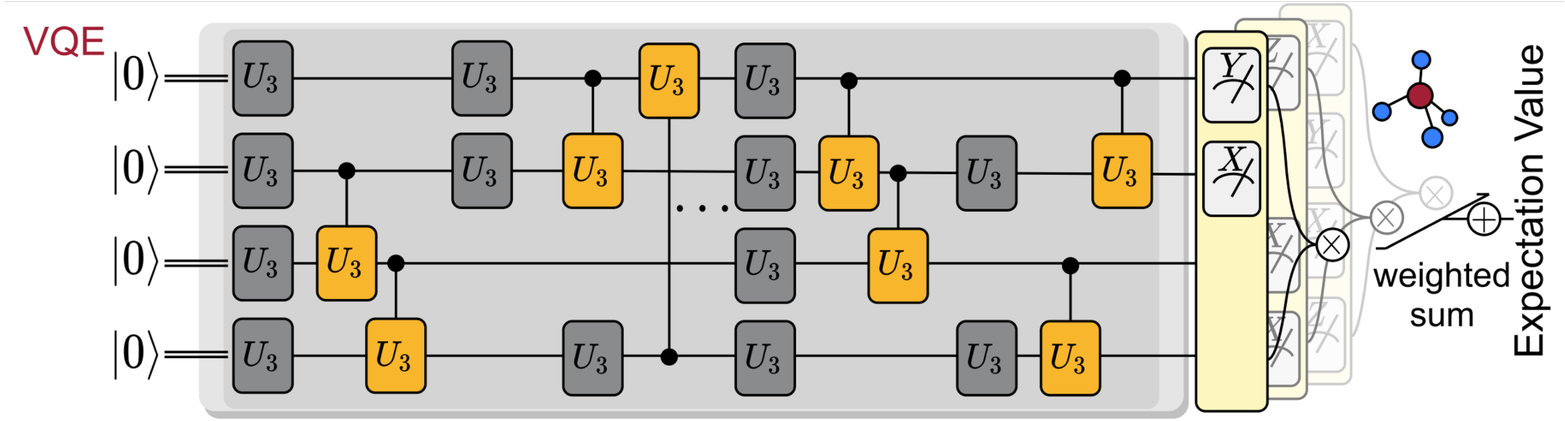


Benchmarks: QNN and VQE

- Quantum Neural Networks: classification

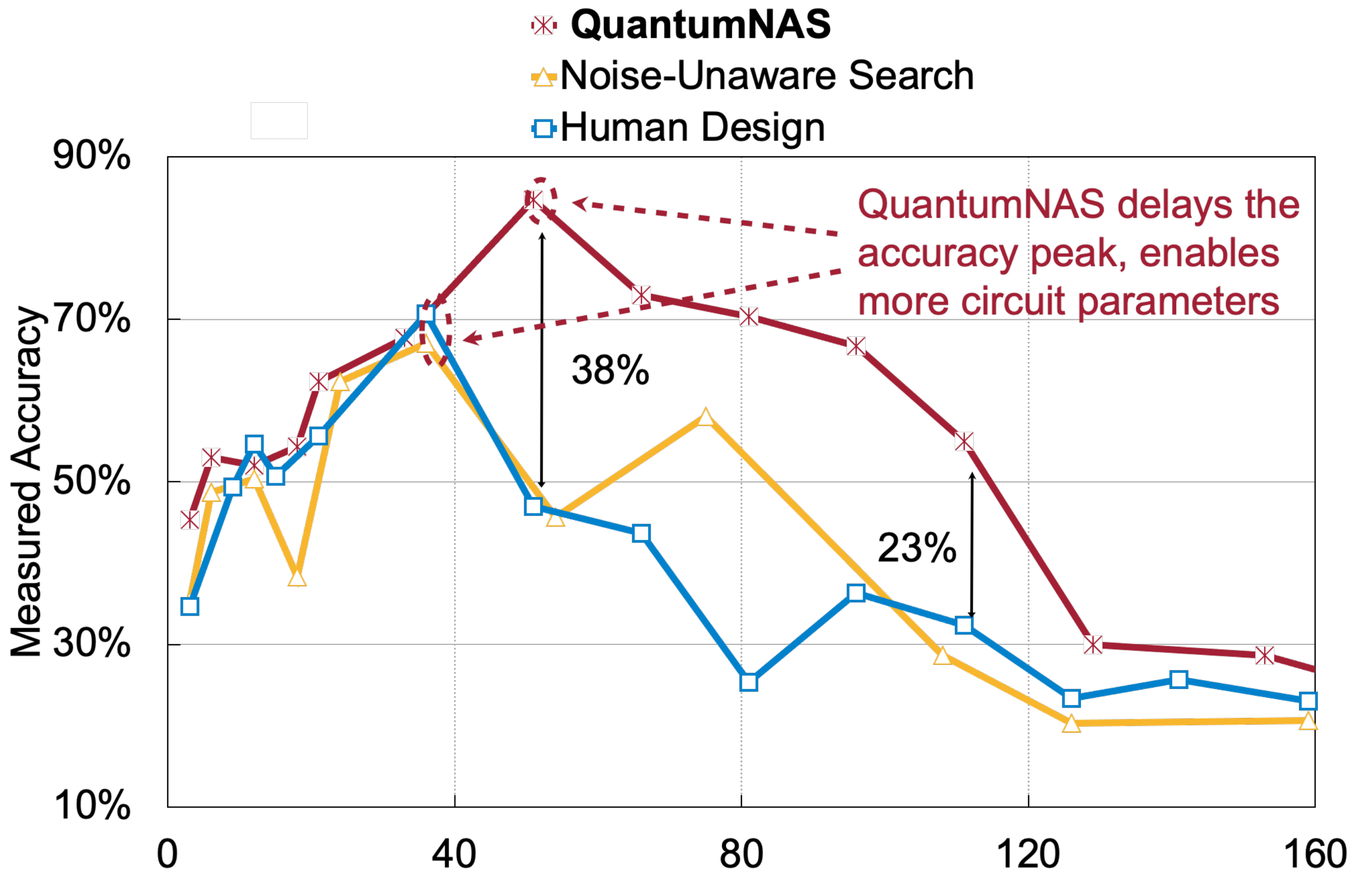


- Variational Quantum Eigensolver: finds the ground state energy of molecule Hamiltonian



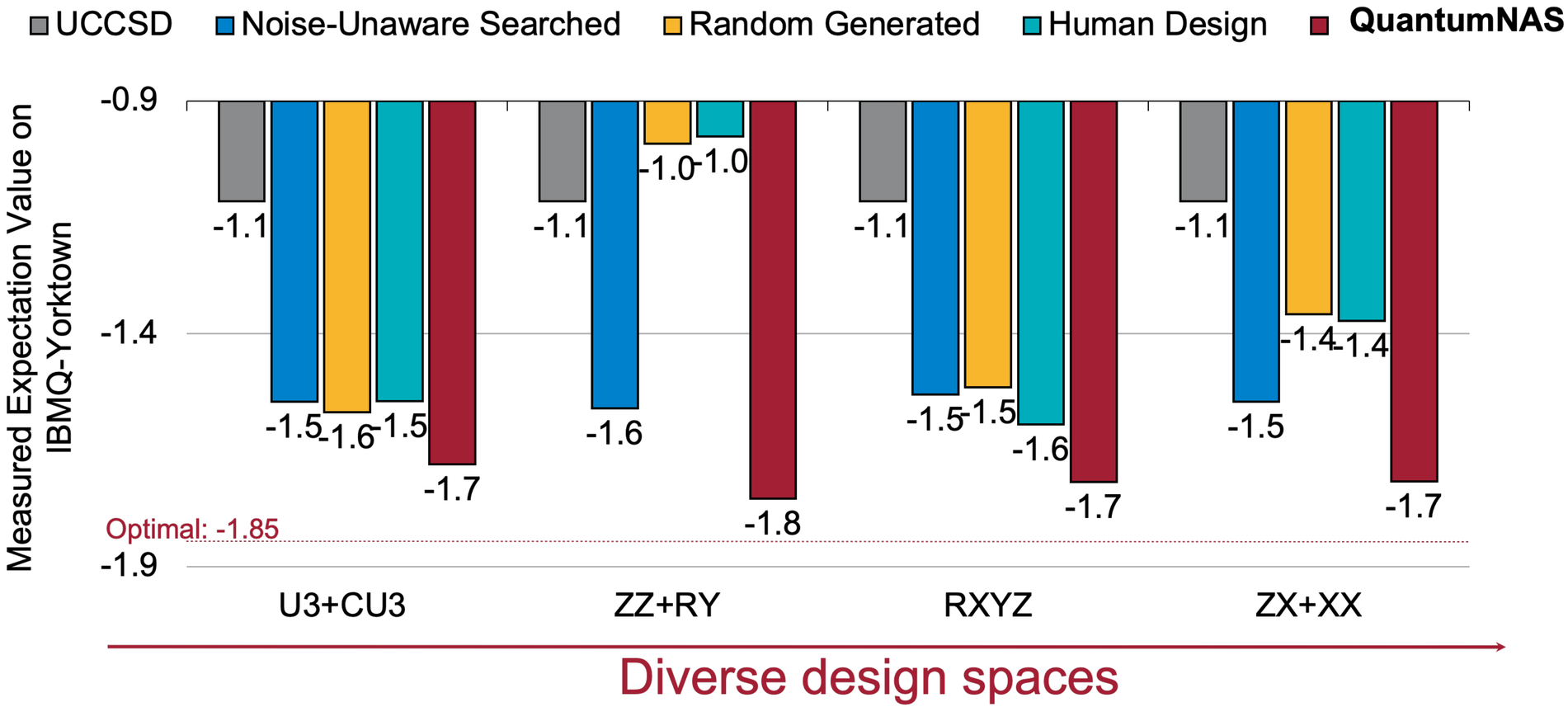
QML Results

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



Consistent Improvements on Diverse Design Spaces

- H2 in different design spaces on IBMQ-Yorktown



Scalable to Large number of Qubit

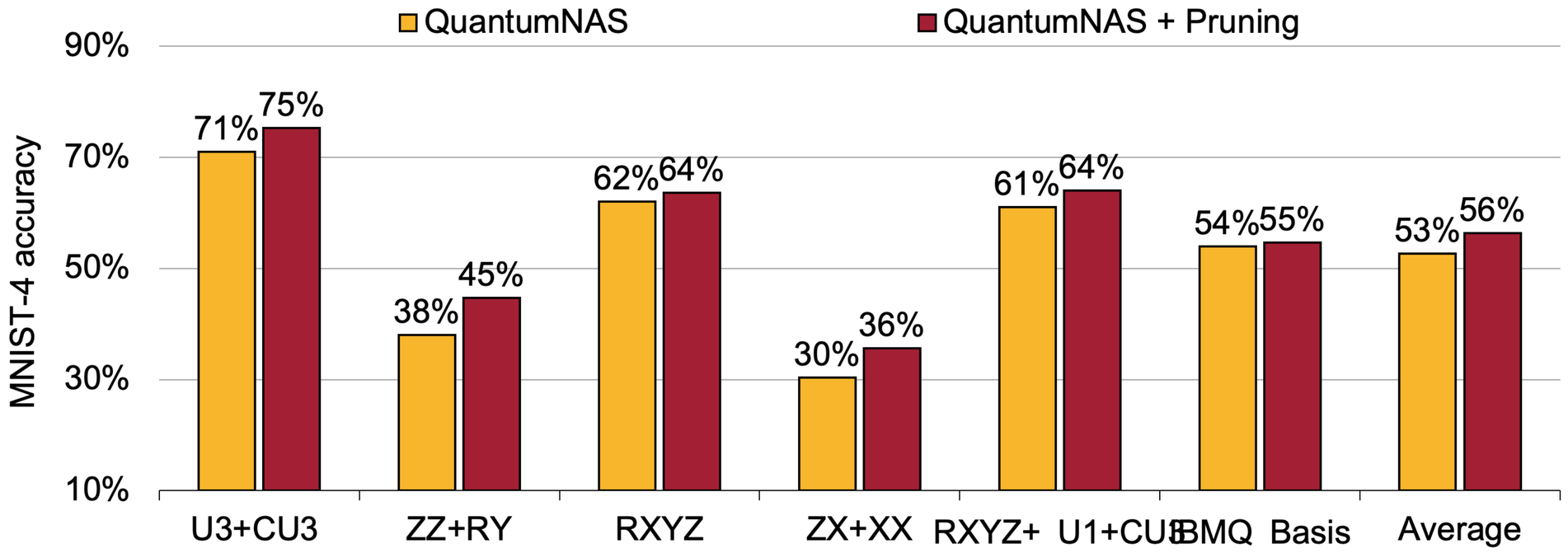
- On large devices
- MNIST-10 accuracy

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%**

Effectiveness of Quantum Gate Pruning

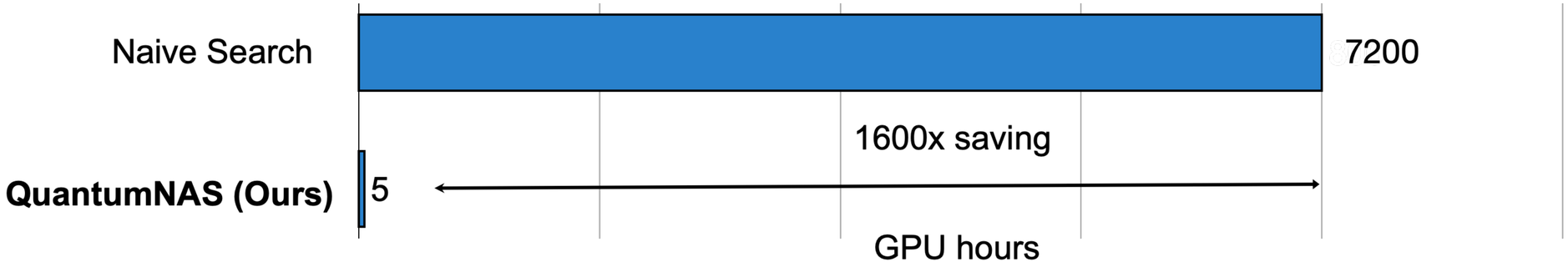
- For MNIST-4, Quantum gate pruning improves accuracy by 3% on average



Time Cost

- On 1 Nvidia Titan RTX 2080 ti GPU

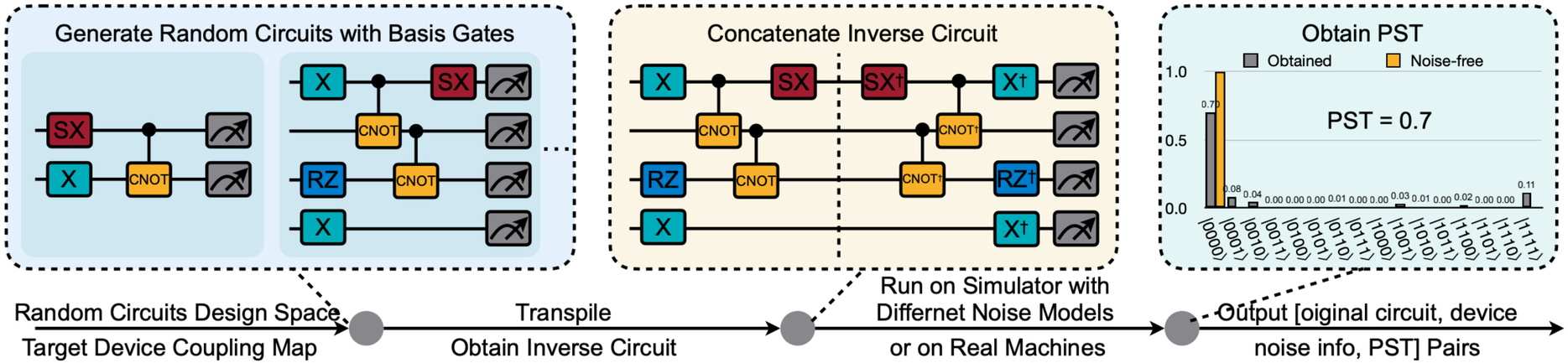
Step / #qubits	SuperCircuit Training	Noise-Adaptive Co-search	SubCircuit Training	Deployment on Real QC
4 Qubits	0.5h	3h	0.5h	0.5h
15 Qubits	5h	5h	5h	1h
21 Qubits	20h	10h	15h	1h



Other Search Frameworks

Faster method to evaluate real device performance

- Graph transformer for circuit fidelity estimation

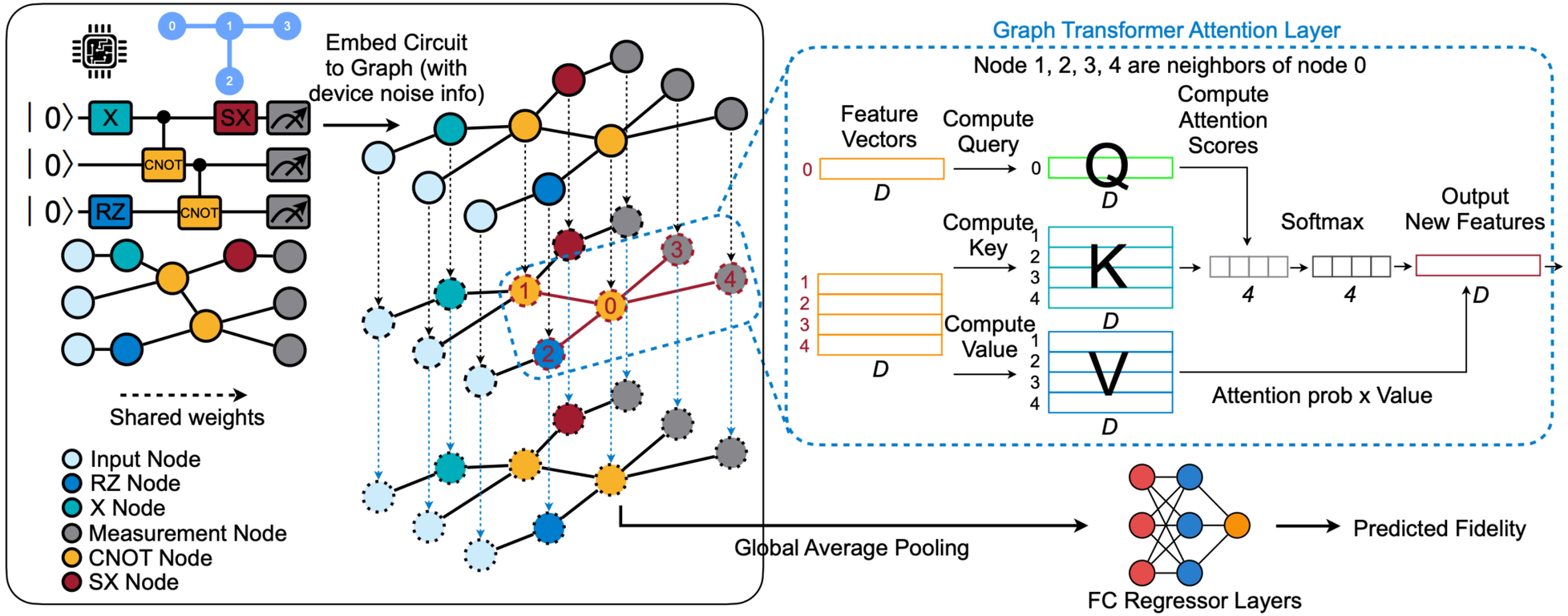


QuEst: Graph Transformer for Quantum Circuit Reliability Estimation

Other Search Frameworks

Faster method to evaluate real device performance

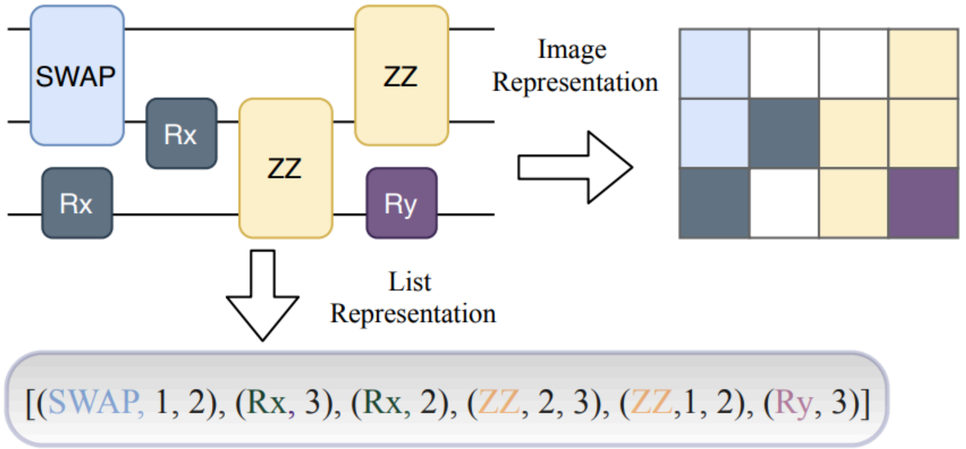
- Graph transformer for circuit fidelity estimation



QuEst: Graph Transformer for Quantum Circuit Reliability Estimation

Other Search Frameworks

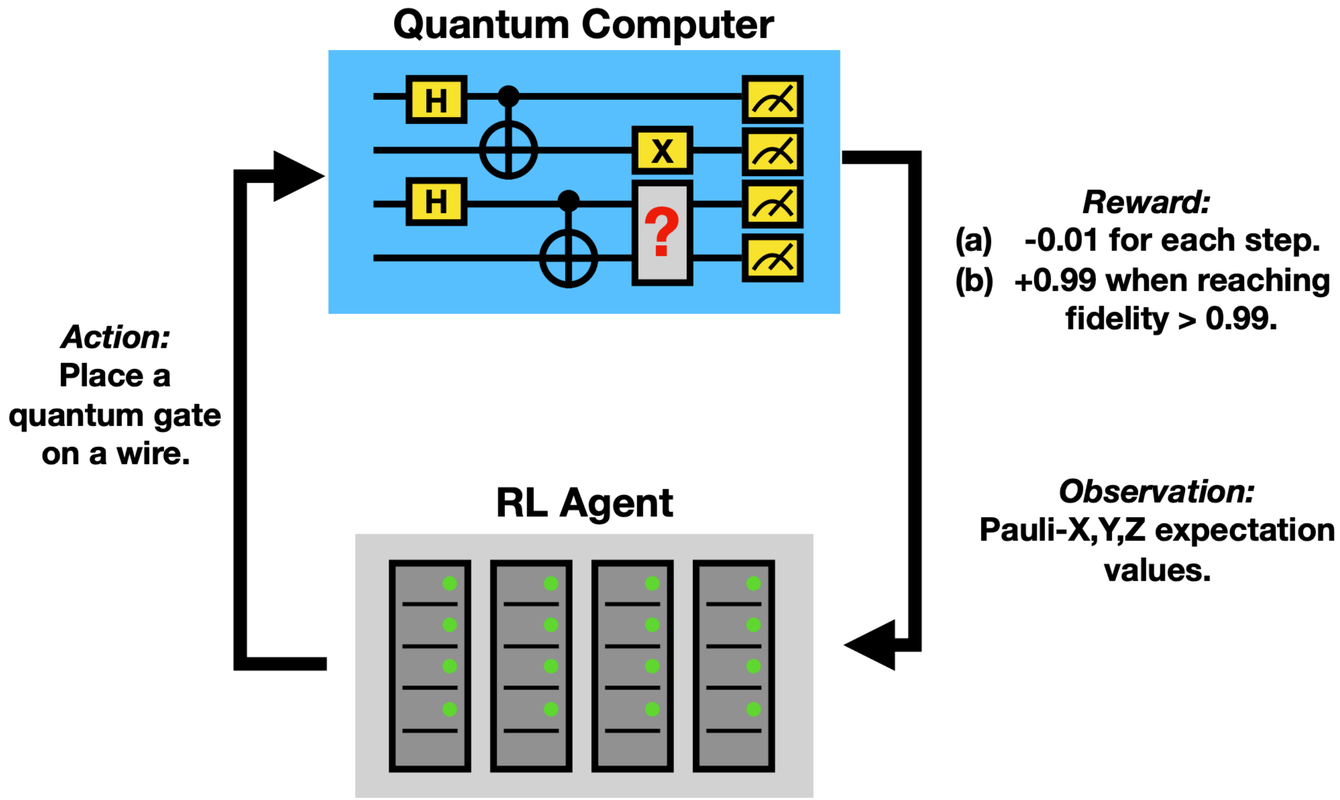
Faster method to evaluate real device performance



Neural Predictor based Quantum Architecture Search

Other Search Frameworks

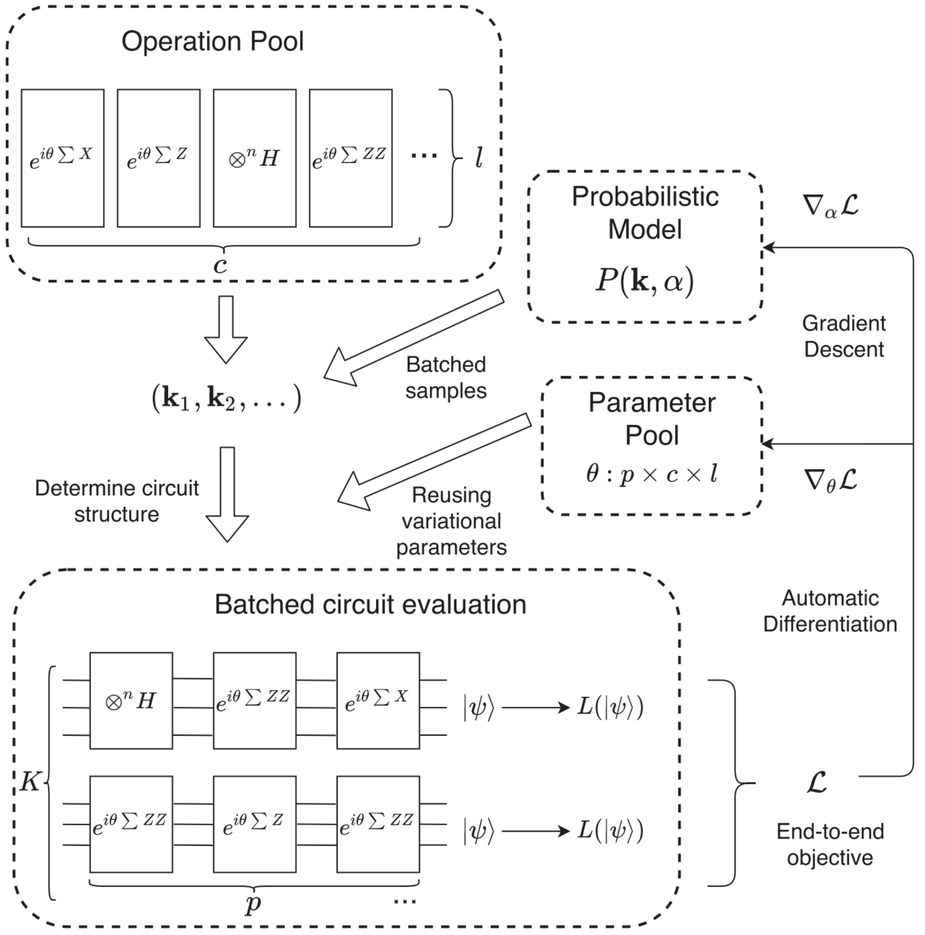
RL for Architecture Search



Quantum Architecture Search via Deep Reinforcement Learning

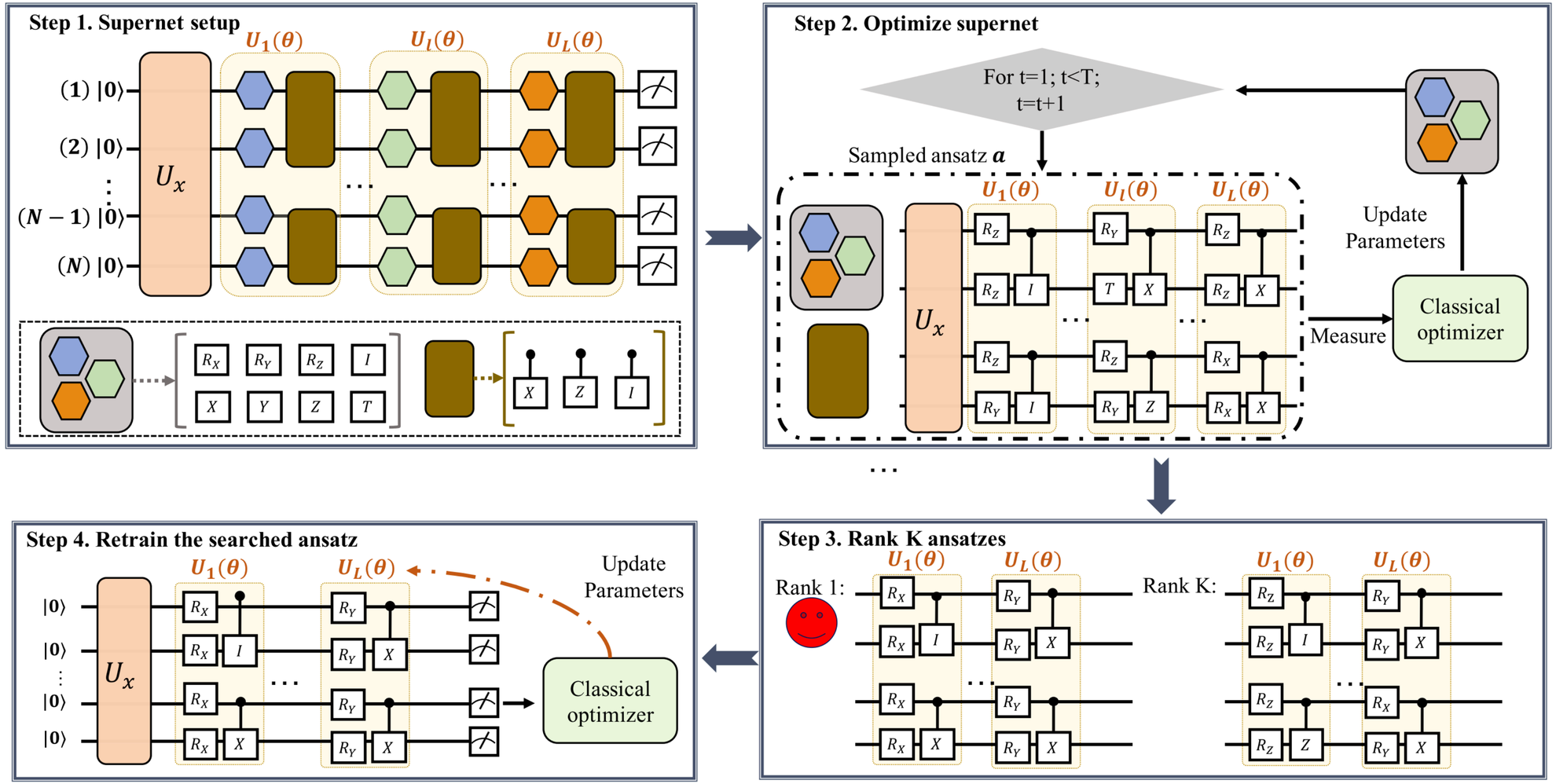
Other Search Frameworks

Differentiable Search



Differentiable quantum architecture search

Other Search Frameworks



Quantum circuit architecture search for variational quantum algorithms

Section 4

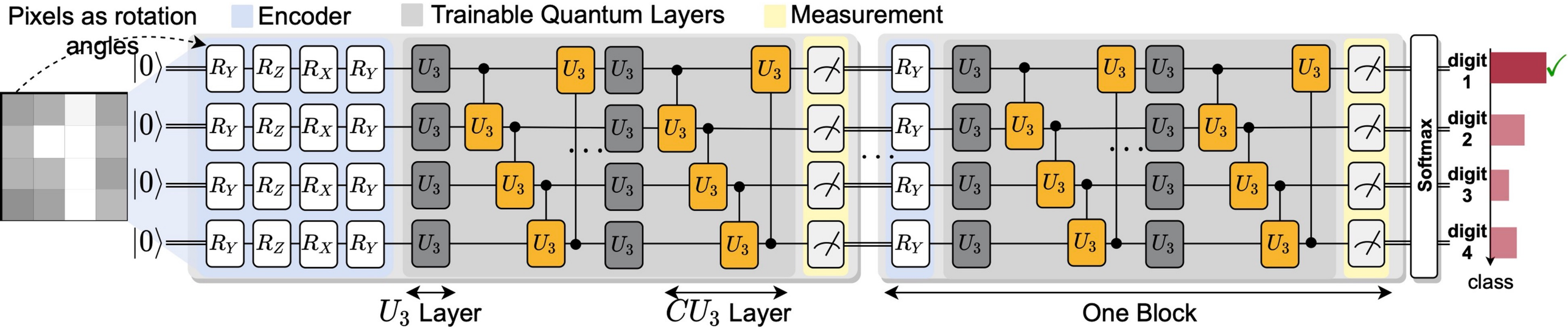
Robust PQC Parameter Training

Robustness of PQC parameters

- Previous several works focus on PQC architecture robustness
- What about PQC parameter robustness?

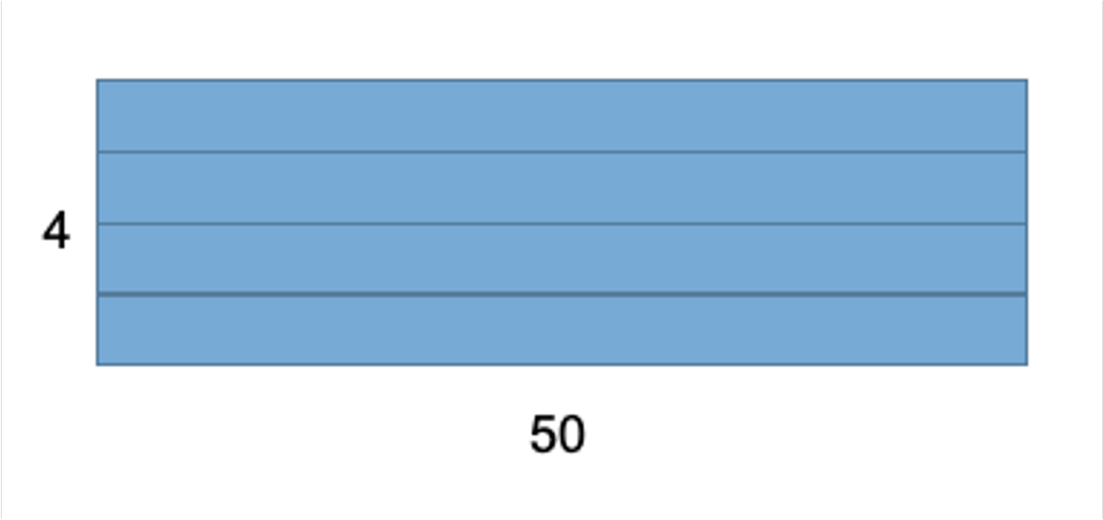
Multi-Node QNN

- Each node contains encoder, trainable layers and measurement



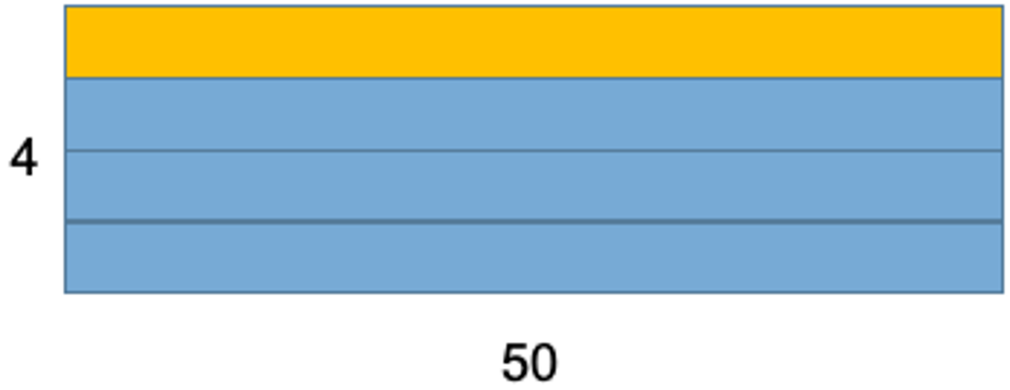
Post-Measurement Normalization

- Normalize the measurement outcome
  - Along the batch dimension
  - For example, we train the 4-qubit PQC with batch=50 then we have results as 50 * 4 values



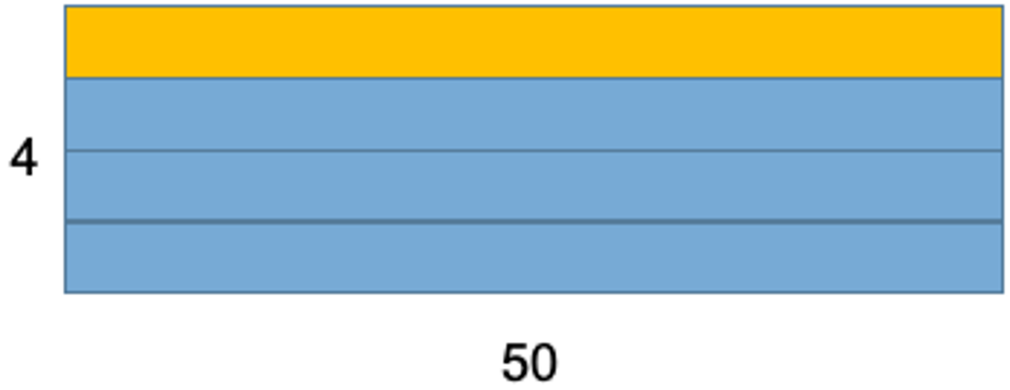
Post-Measurement Normalization

- Normalize the measurement outcome
  - Along the batch dimension
  - For example, we train the 4-qubit PQC with batch=50 then we have results as 50 * 4 values
    - Compute the mean and std on of the measurement outcome on each qubit across batch dim



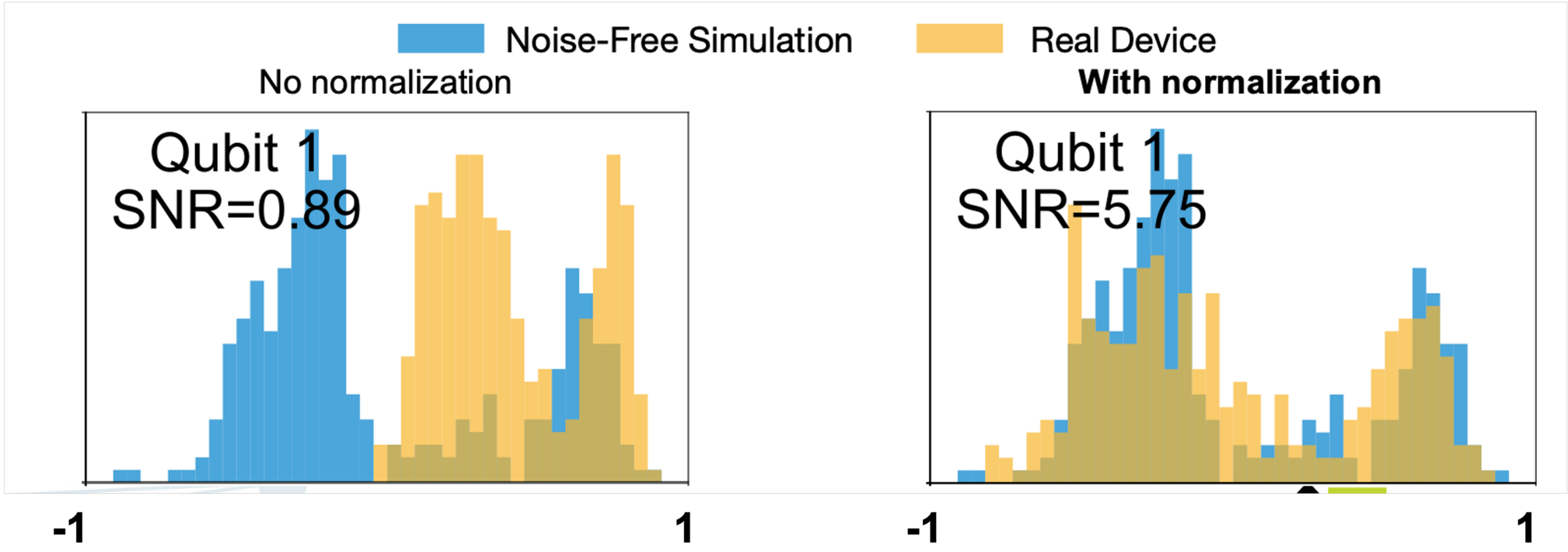
Post-Measurement Normalization

- Normalize the measurement outcome
  - Along the batch dimension
  - For example, we train the 4-qubit PQC with batch=50 then we have results as 50 * 4 values
    - Compute the mean and std on of the measurement outcome on each qubit across batch dim
    - Normalize the measurement outcome with the computed **mean and std**



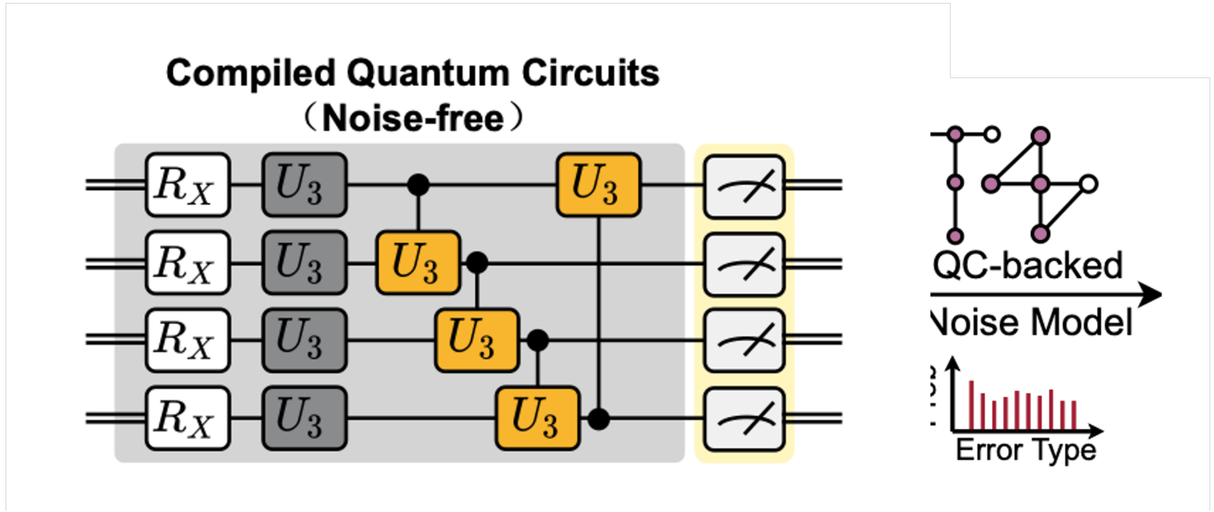
Post-Measurement Normalization

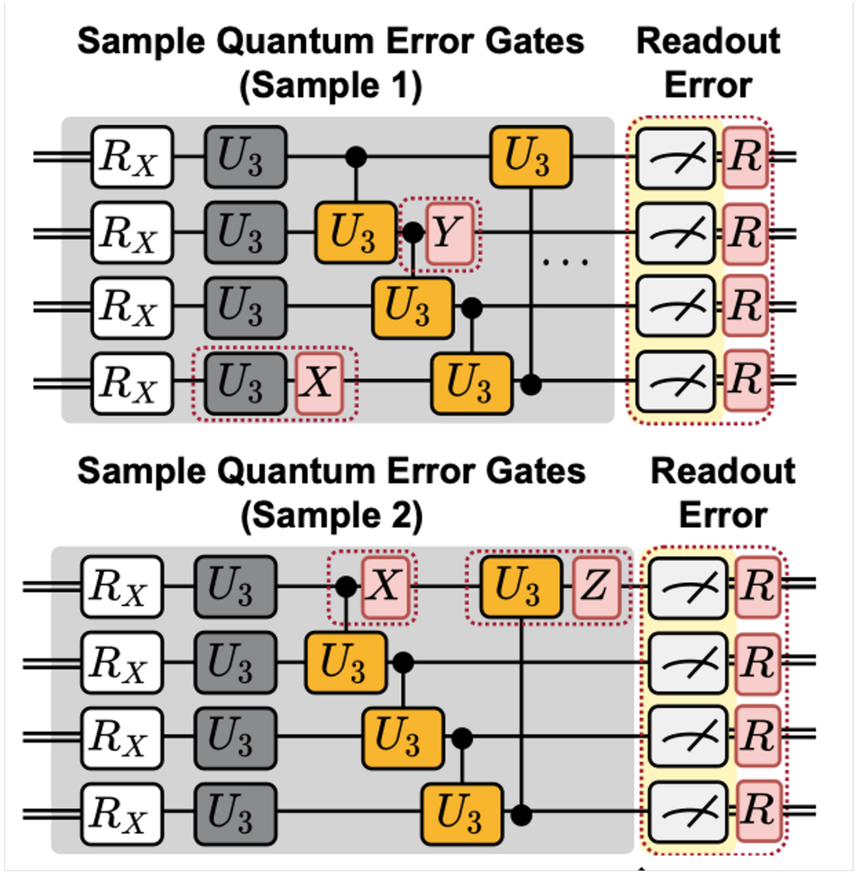
- Normalize the measurement outcome
  - Along the batch dimension
- Measurement outcome distribution of 50 quantum circuits:



Noise Injection

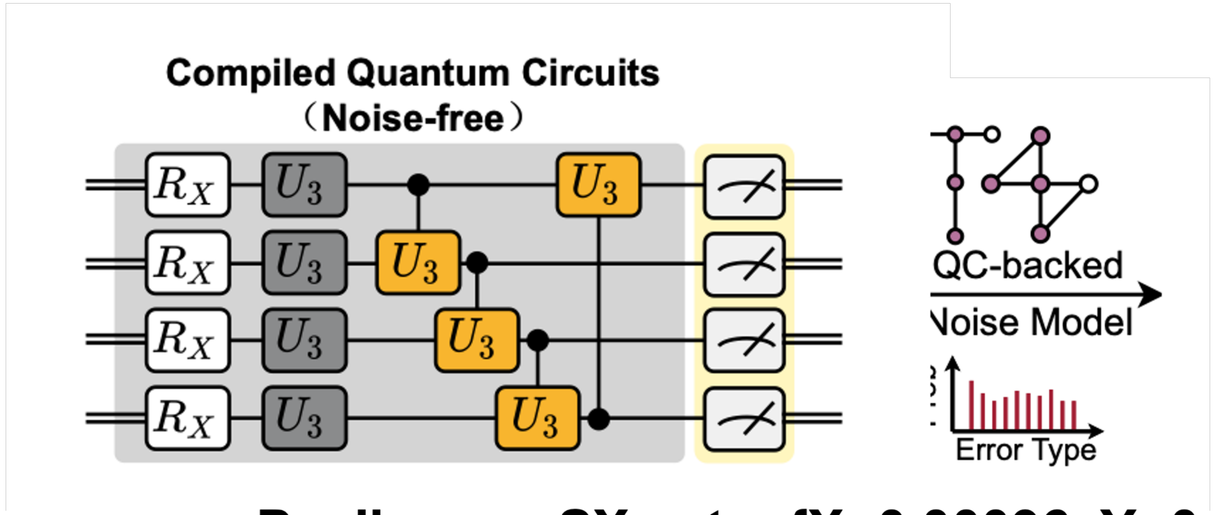
- Inject noise during training on classical simulator
  - Pauli error
  - Readout error

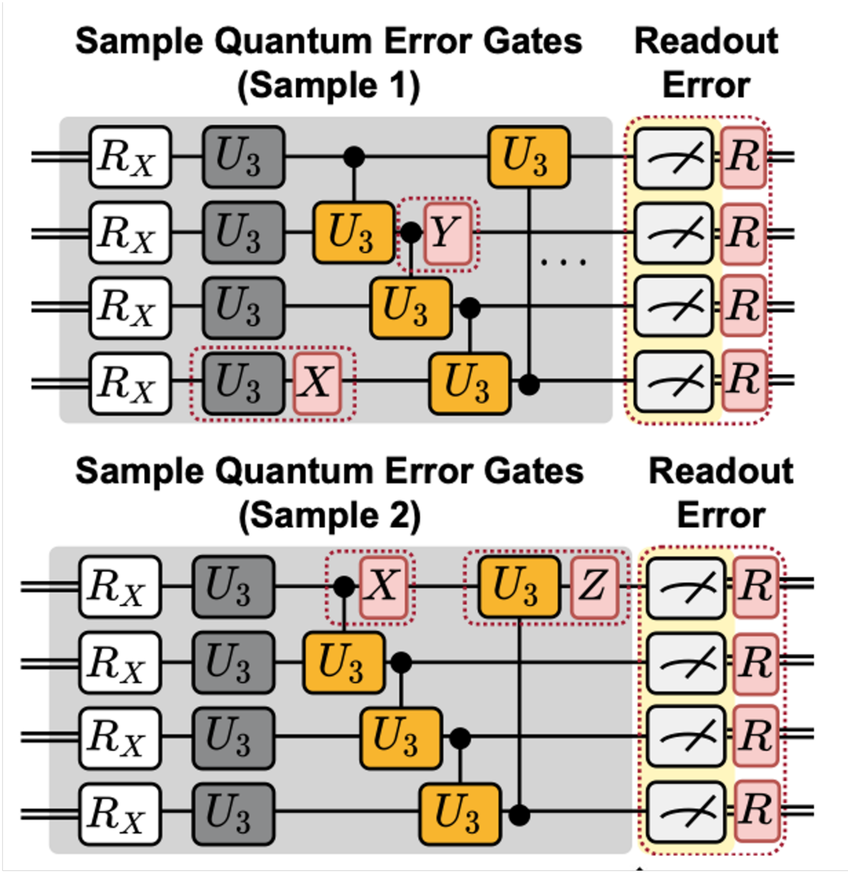




Noise Injection

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



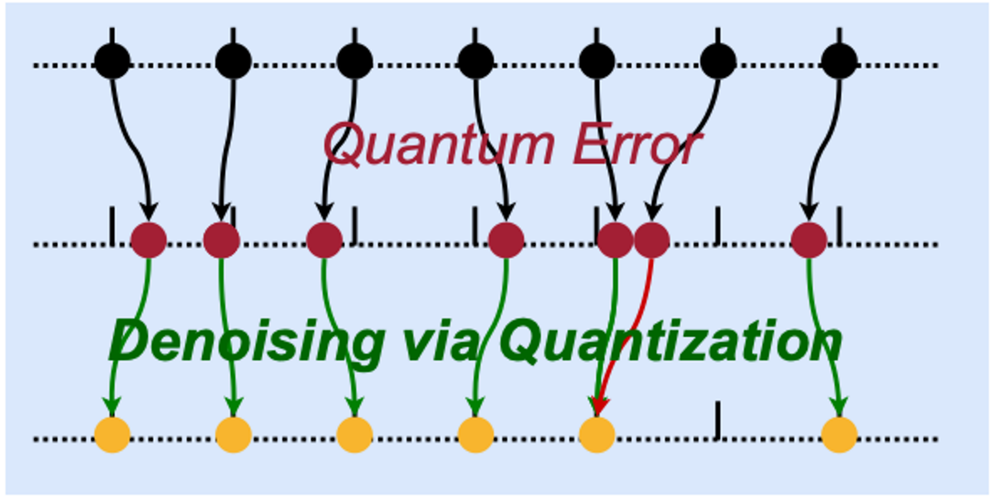


Pauli error: SX gate: {X: 0.00096, Y: 0.00096, Z: 0.00096, None: 0.99712}

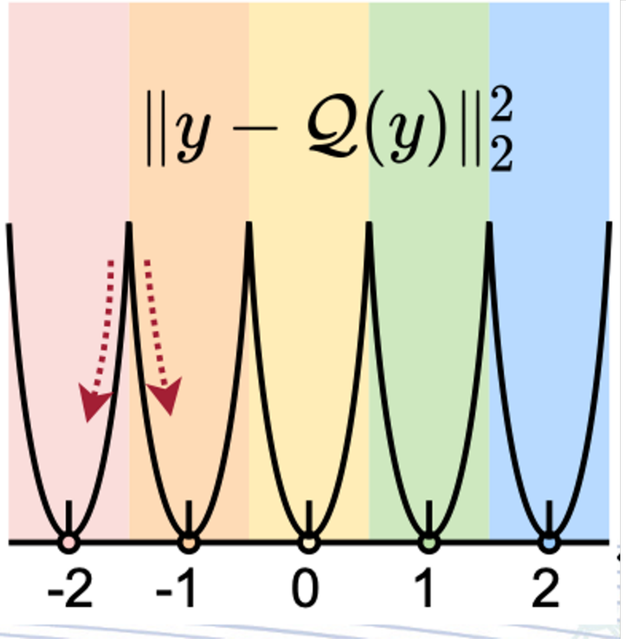
Readout Error Matrix: **0.984, 0.016**
0.022, 0.978

Post-Measurement Quantization

- Quantize measurement outcomes
  - Denoising effect
  - Small errors will be mitigated

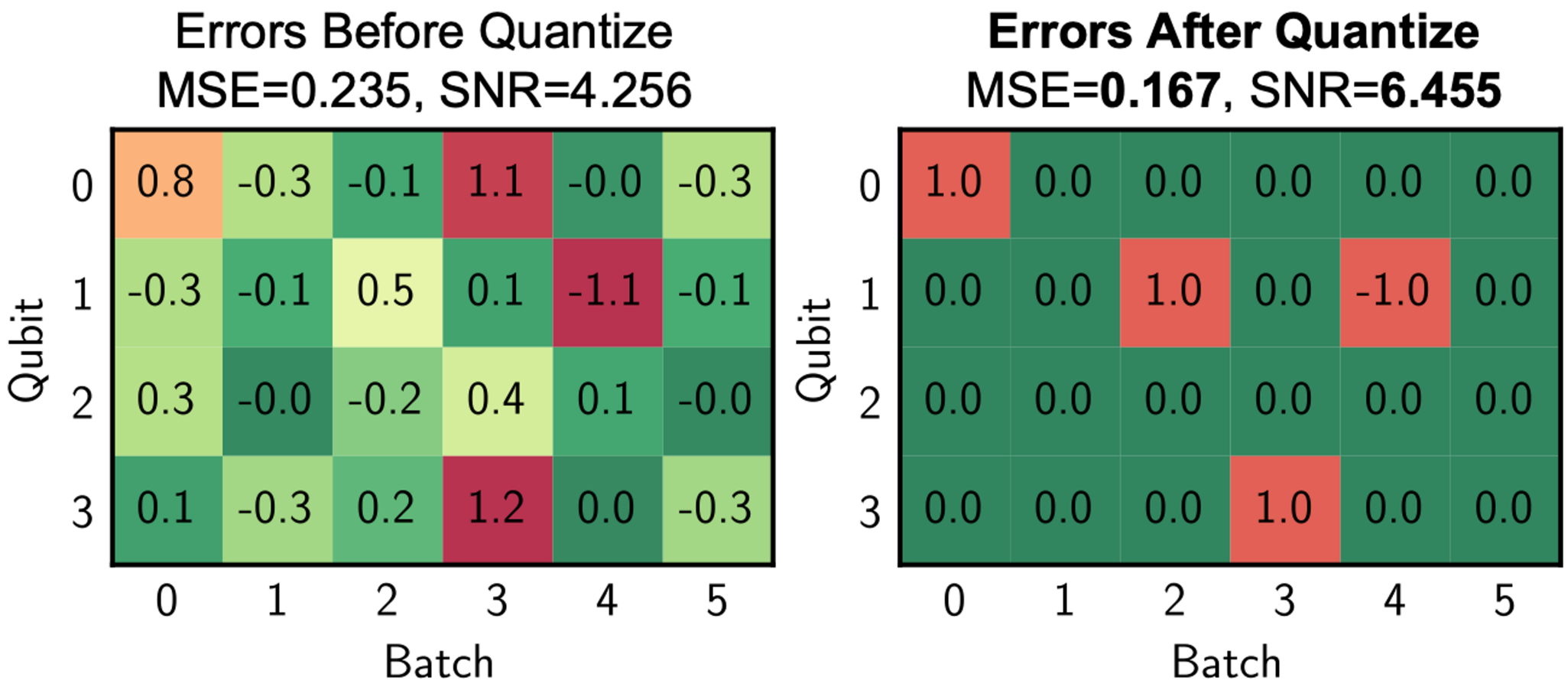


- **Loss** term to encourage measurement outcomes to be close to **centroids**



Post-Measurement Quantization

- Quantization reduces errors and improves SNR



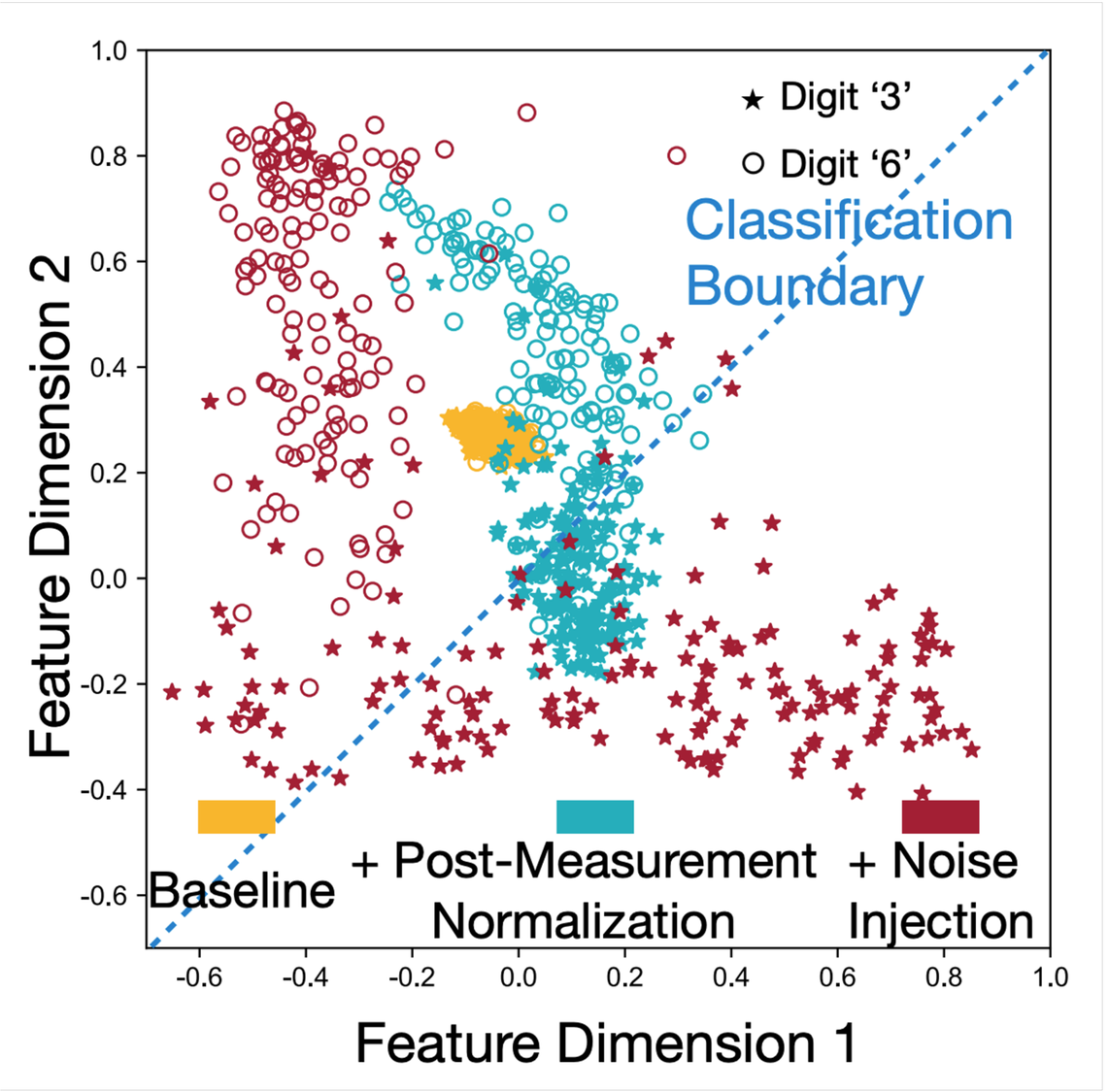
Consistent Improvement on Various Benchmarks

- IBMQ Santiago

Method	MNIST-4	FMNIST-4	Vowel-4	MNIST-2	FMNIST-2	Cifar-2
Baseline	0.30	0.32	0.28	0.84	0.78	0.51
+ Normalization	0.41	0.61	0.29	0.87	0.68	0.56
+Noise Injection	0.61	0.70	0.44	0.93	0.86	0.57
+ Quantization	**0.68**	**0.75**	**0.48**	**0.94**	**0.88**	**0.59**

Visualization of Feature Space

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



Summary of Today's Lecture

- **Today we learned**
  - TorchQuantum Usage (Continued)
  - Robust state preparation
  - Robust PQC architecture search
  - Robust PQC parameter training
- **Next Class**
  - Final project presentations