QuantumNAS: Noise-Adaptive Search for Robust Quantum Circuits

Hanrui Wang[1], Yongshan Ding[2], Jiaqi Gu[3], Yujun Lin[1], David Pan[3], Fred Chong[2], Song Han[1]
[1]MIT, [2]University of Chicago, [3]University of Texas at Austin

Outline

- Introduction
- Background
- QuantumNAS
  - SuperCircuit Training
  - Noise-Adaptive Co-Search
  - Searched SubCircuit Training
  - Gate Pruning
- Evaluation
- Conclusion

Outline

- Introduction
- Background
- QuantumNAS
  - SuperCircuit Training
  - Noise-Adaptive Co-Search
  - Searched SubCircuit Training
  - Gate Pruning
- Evaluation
- Conclusion

Introduction

- Quantum computing has potentials to bring exponential advantages
- Currently: NISQ era
  - Insufficient qubits for error correction
  - Imperfect qubits: quantum noise forms the bottleneck

Introduction

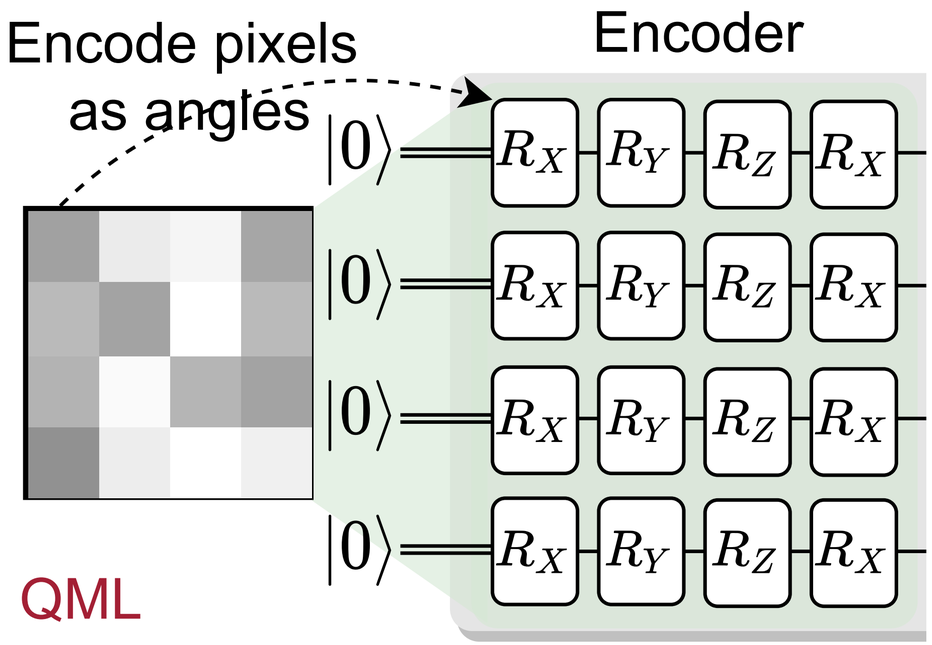
- Existing works focus on noise-adaptive quantum compilation:
  - Noise adaptive qubit mapping
  - Noise adaptive instruction scheduling
  - Noise adaptive crosstalk mitigation
  - …
- They are optimizing the compilation process with a **fixed** quantum circuit
- QuantumNAS:
  - Target at **parameterized** quantum circuits
  - **Noise-adaptive co-optimization** of compilation (qubit mapping) and circuit (Ansatz)

Outline

- Introduction
- Background
- QuantumNAS
  - SuperCircuit Training
  - Noise-Adaptive Co-Search
  - Searched SubCircuit Training
  - Gate Pruning
- Evaluation
- Conclusion

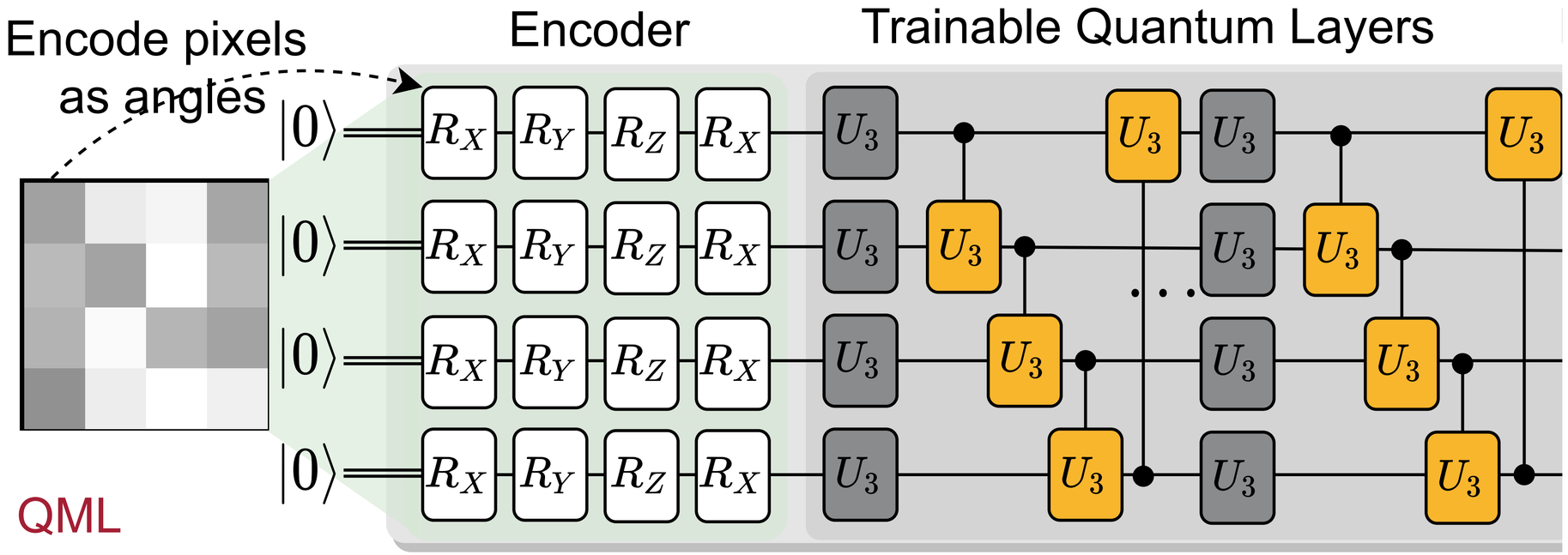
Background

- Parameterized quantum circuits
  - Quantum Neural Networks



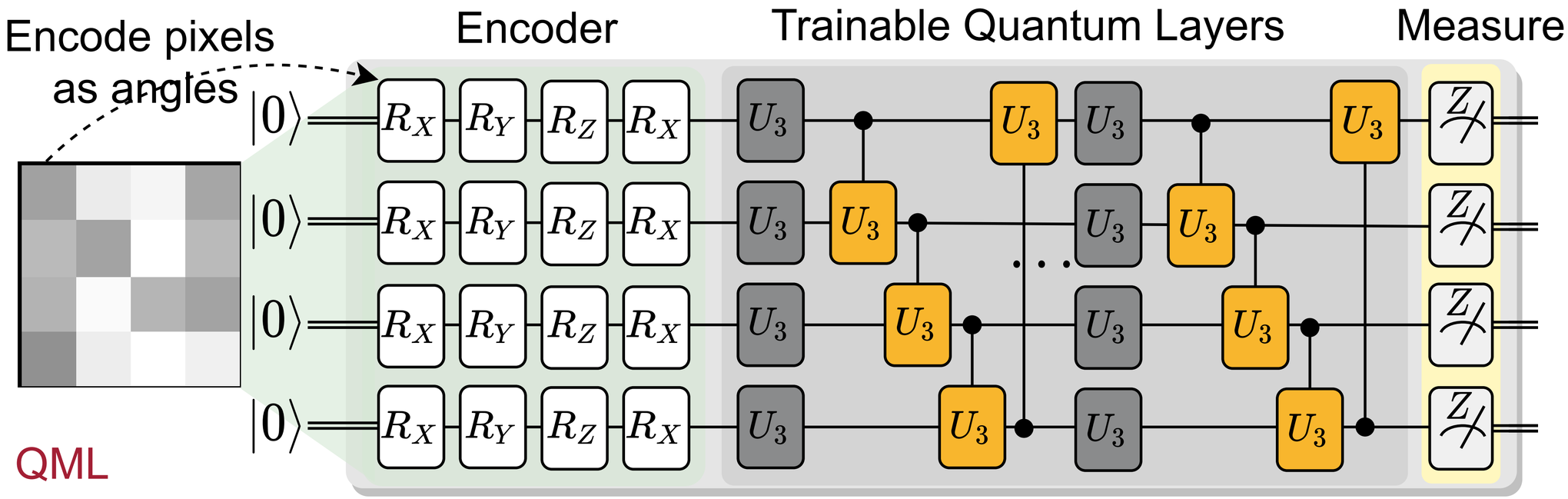
Background

- Parameterized quantum circuits
  - Quantum Neural Networks



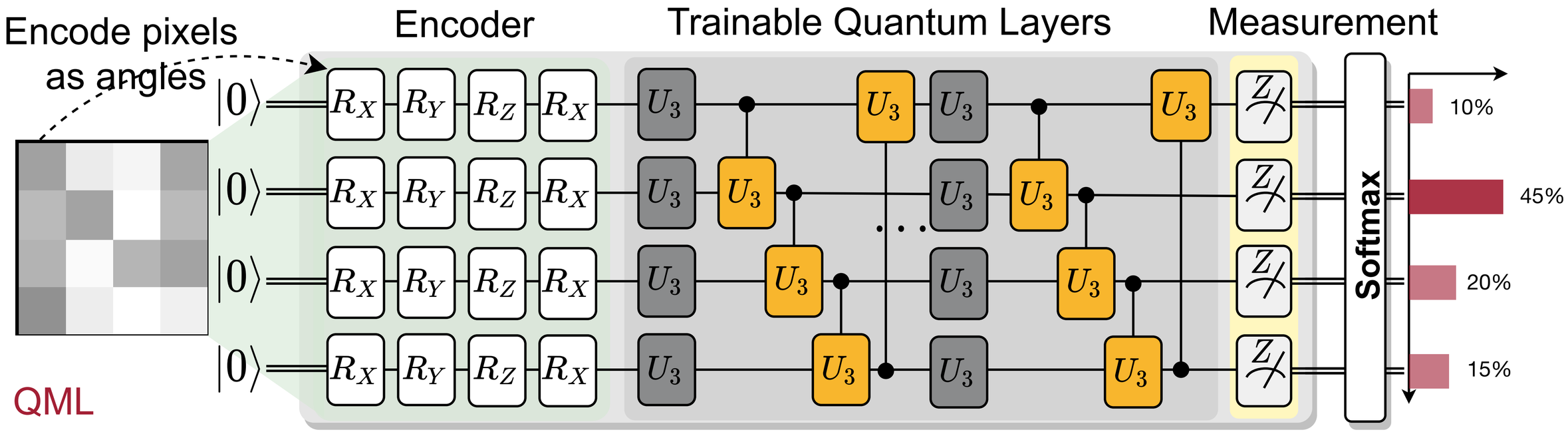
Background

- Parameterized quantum circuits
  - Quantum Neural Networks



Background

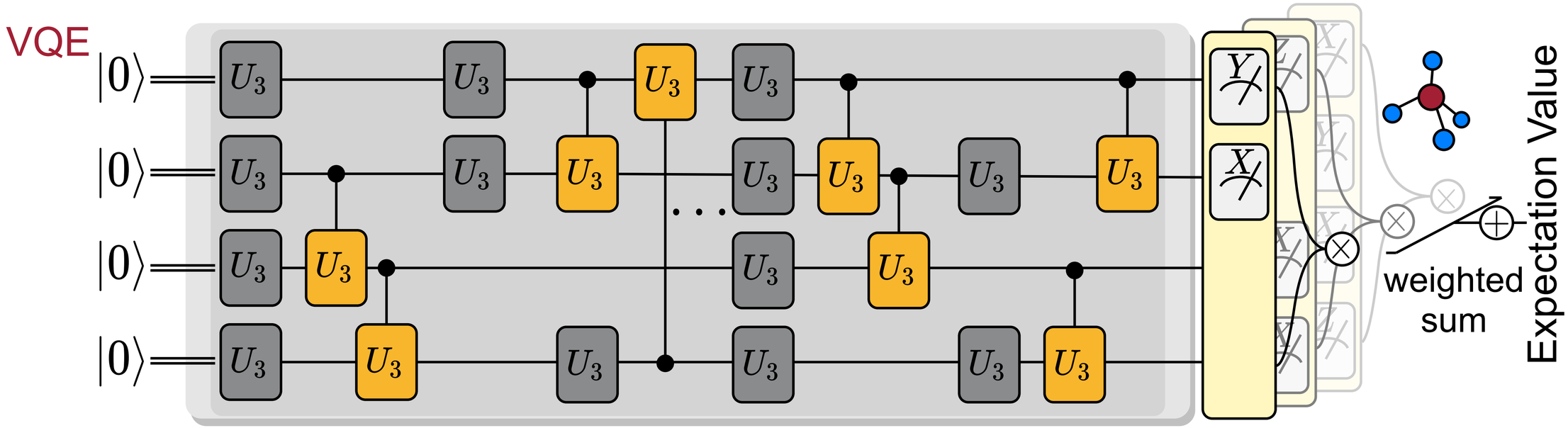
- Parameterized quantum circuits
  - Quantum Neural Networks



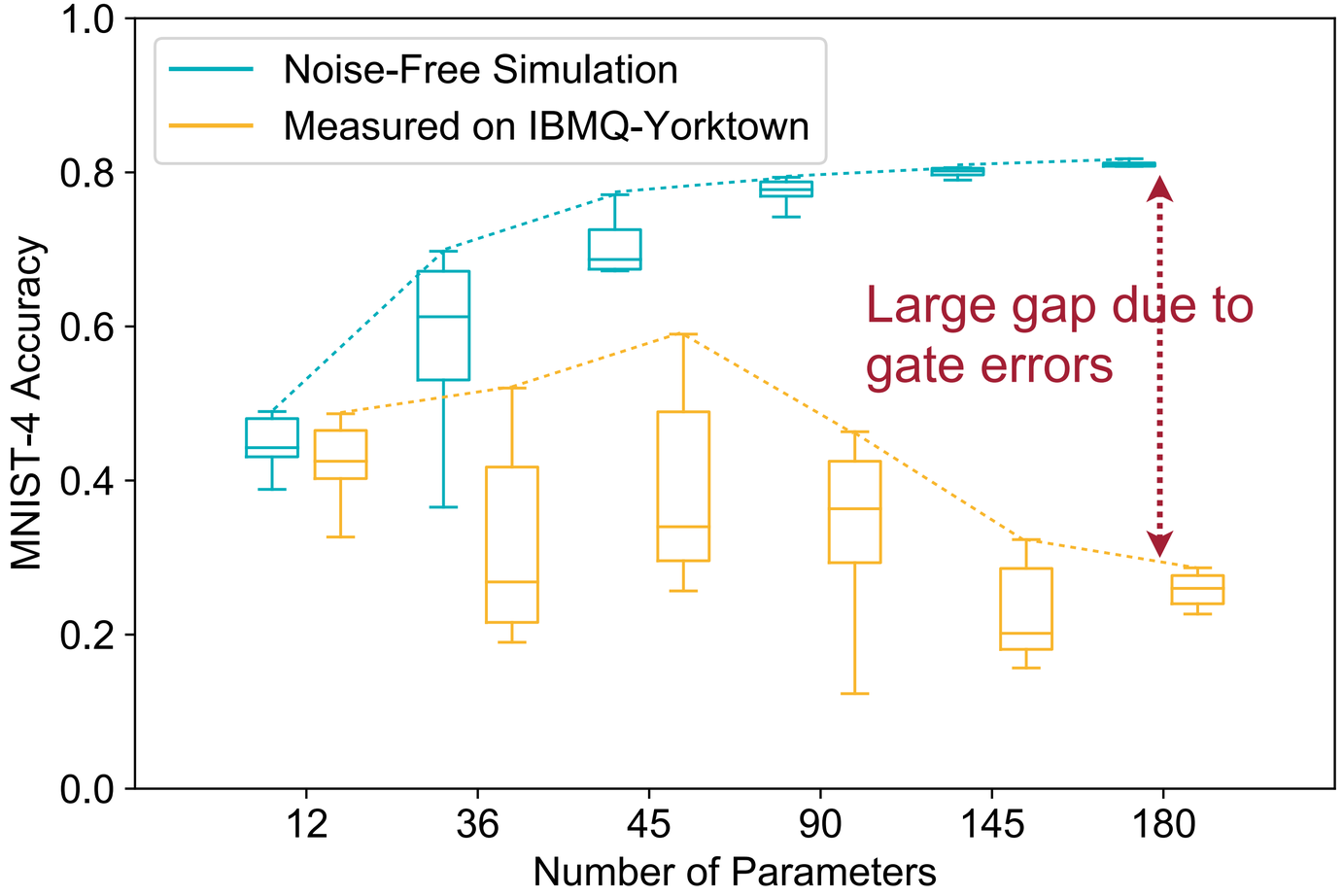
- Softmax $\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$

Background

- Parameterized quantum circuits
  - Variational Quantum Eigensolver



Background



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

Background

- How to find the most robust circuit?
- Naive search for robust circuit
  - In one search step: sample many circuit ansatzes, train each to get **final** parameters, compare performance
  - Too costly
- QuantumNAS aims to find robust circuit in a more **scalable** way

QuantumNAS: Decouple the Training and Search

Naive Search

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

QuantumNAS: Decouple the Training and Search

Naive Search

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

=>

QuantumNAS

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

decouple

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

search

QuantumNAS

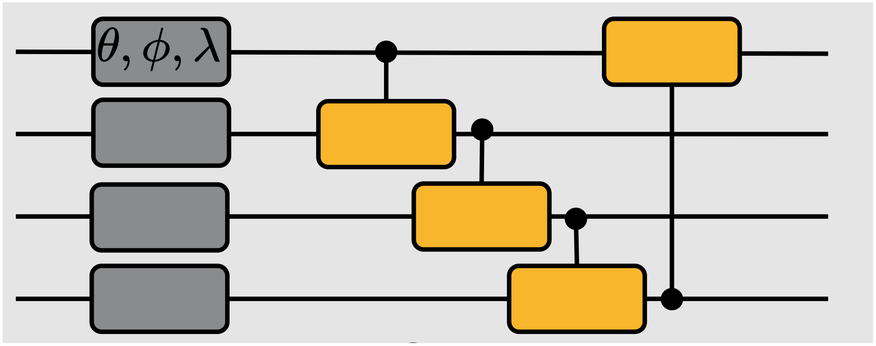
- SuperCircuit Construction and Training
- Noise-Adaptive Evolutionary Co-Search of SubCircuit and Qubit Mapping
- Searched SubCircuit Training
- Iterative Pruning

QuantumNAS

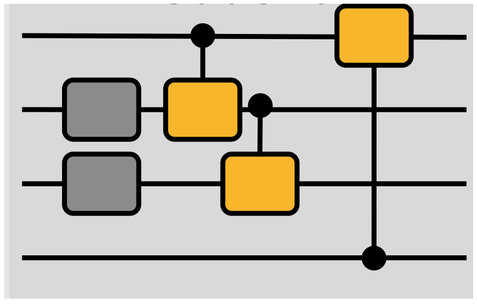
- SuperCircuit Construction and Training
- Noise-Adaptive Evolutionary Co-Search of SubCircuit and Qubit Mapping
- Searched SubCircuit Training
- Iterative Pruning

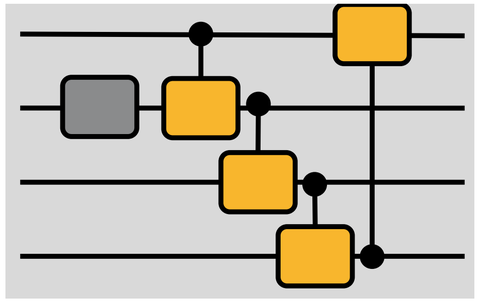
SuperCircuit Construction

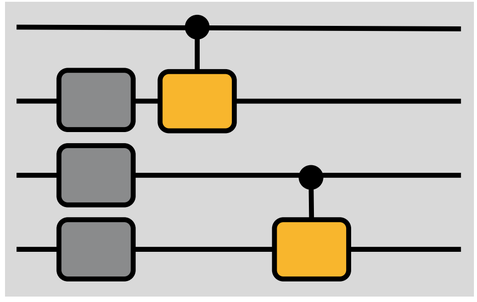
- Why construct a SuperCircuit?
  - Use it for efficient search of circuit candidates with no need to train them to final
- SuperCircuit: the circuit with the largest number of gates in the design space
  - Example: SuperCircuit in U3+CU3 space



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

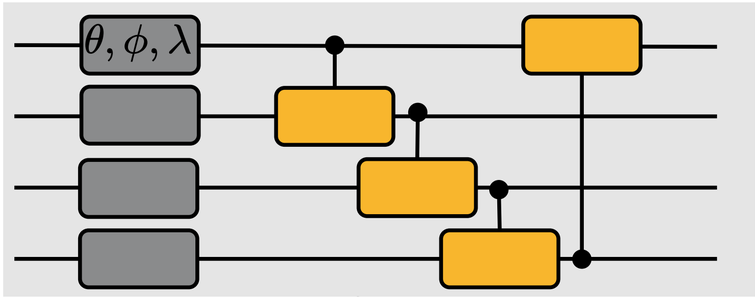




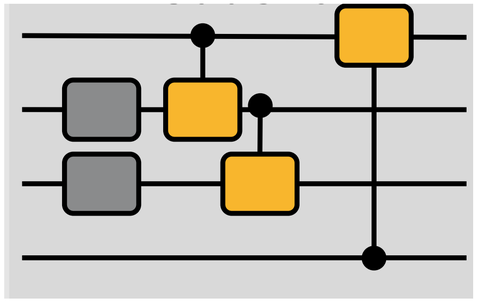


SuperCircuit Training

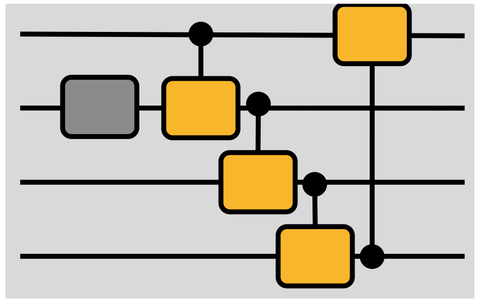
- In one SuperCircuit Training step:
  - Sample a gate subset of SuperCircuit (a SubCircuit)
    - Front Sampling, Progressive Shrinking and Restricted Sampling
  - Only use the subset to perform task and updates the parameters in the subset
  - Parameter updates are cumulative across steps
- Supercircuit:



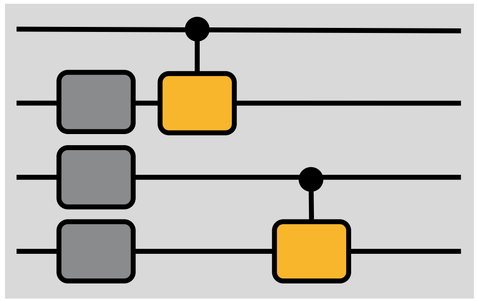
- Sampled subsets (SubCircuits of different steps)



Step1



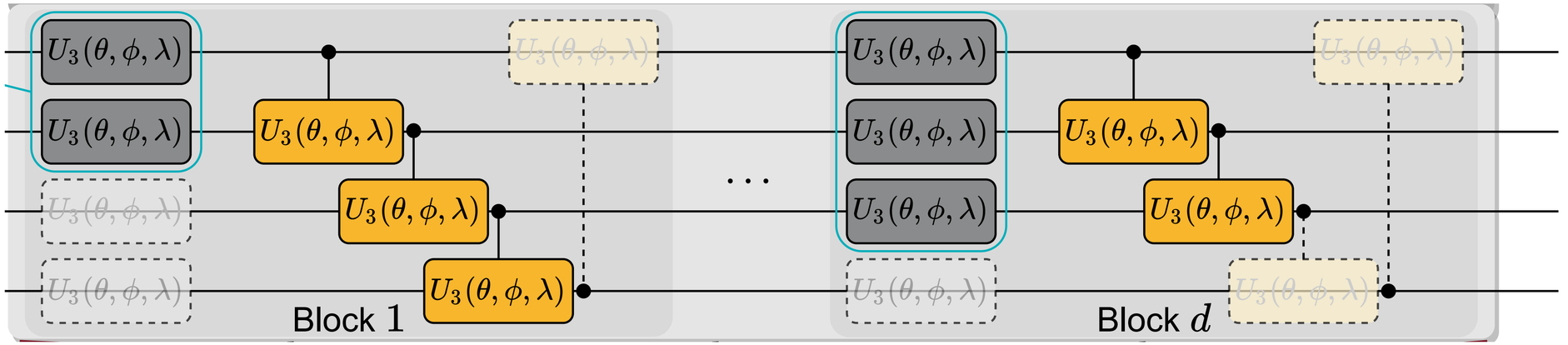
Step2



Step3

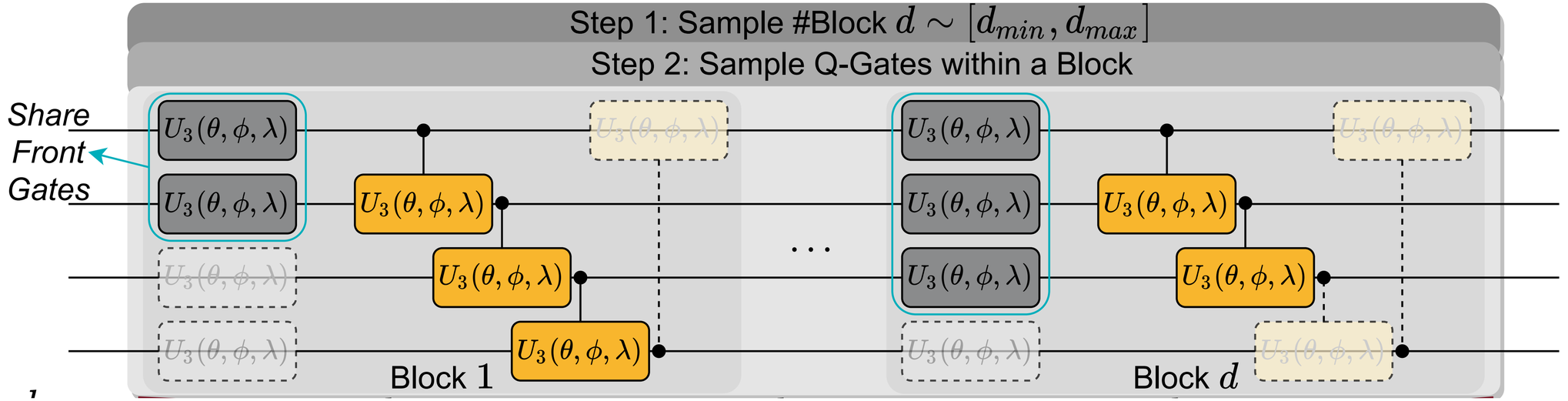
More Details on Construction and Sampling

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



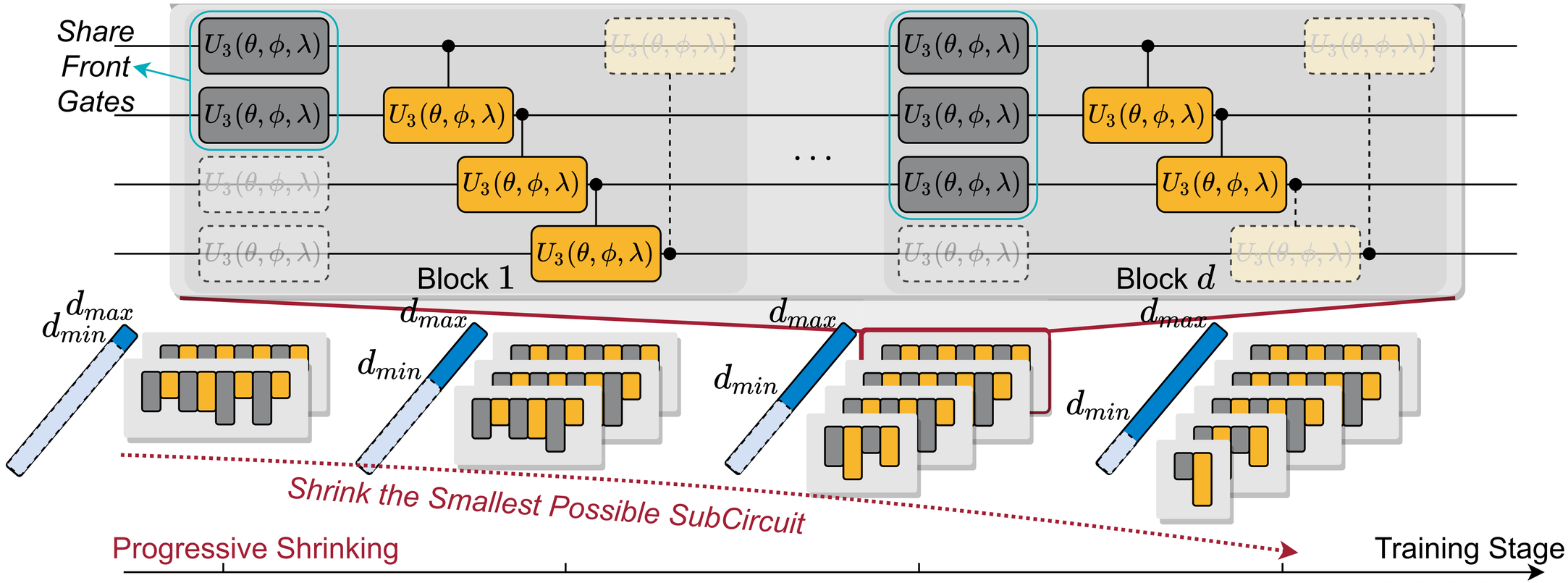
Front Sampling

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



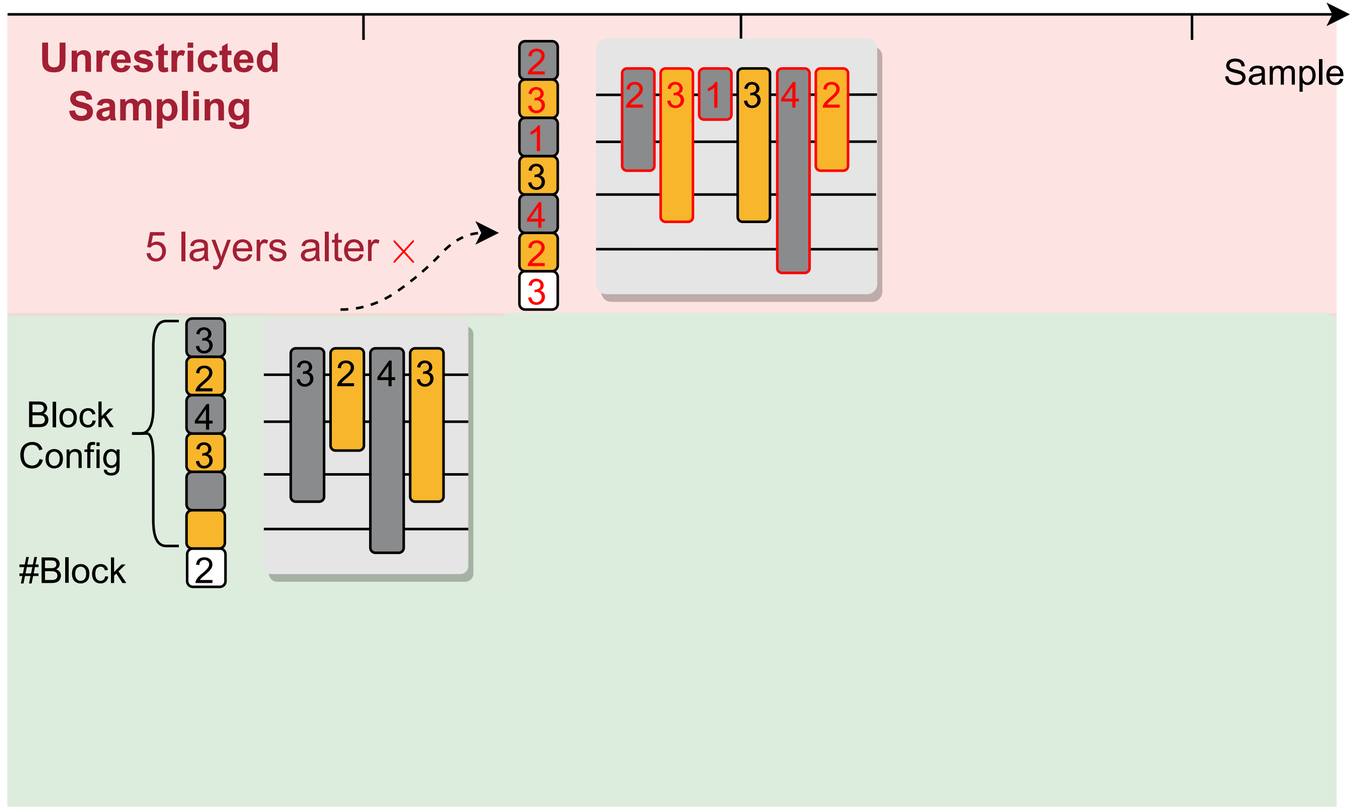
Progressive Shrinking

- Progressive Shrinking:
  - Make parameters in large index block well-trained
  - Progressively reduce the lower bound of possible block number



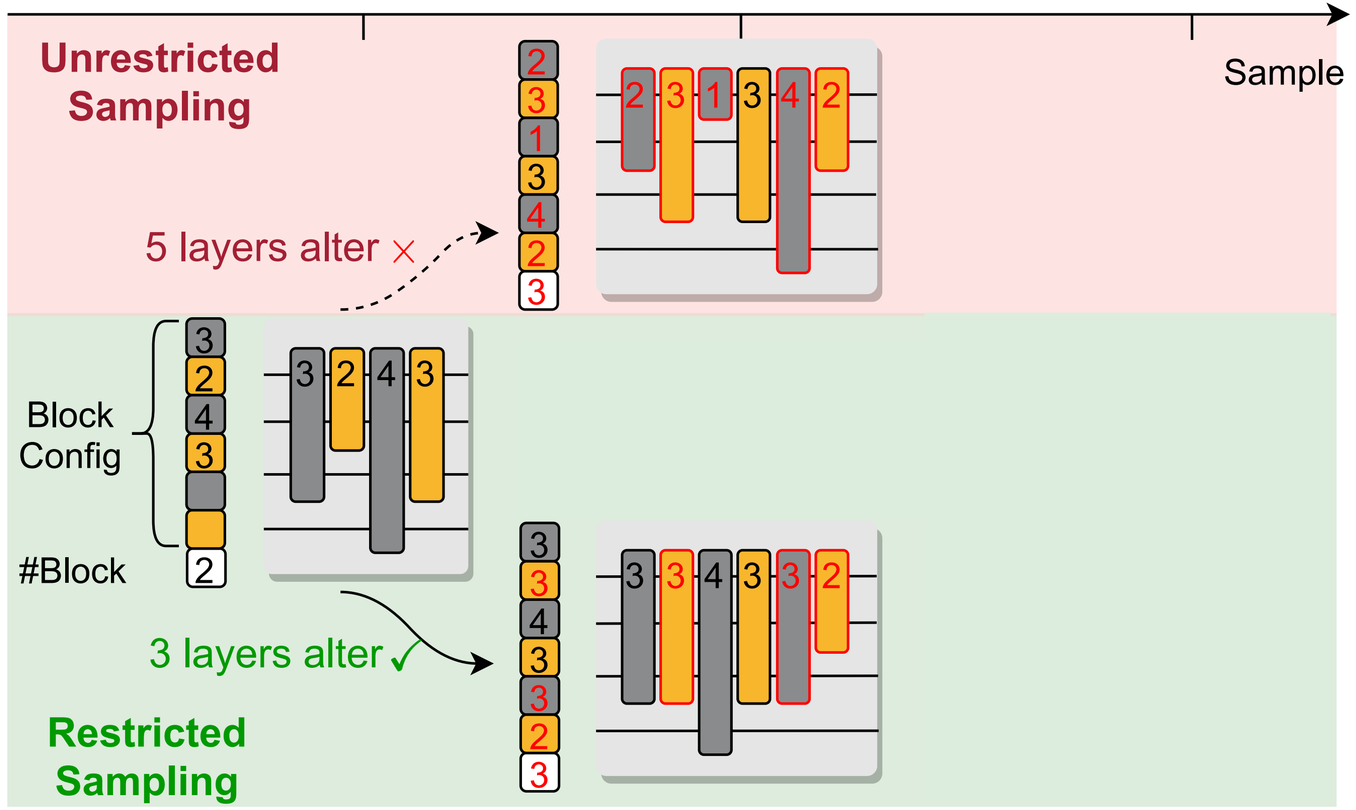
Restricted Sampling

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



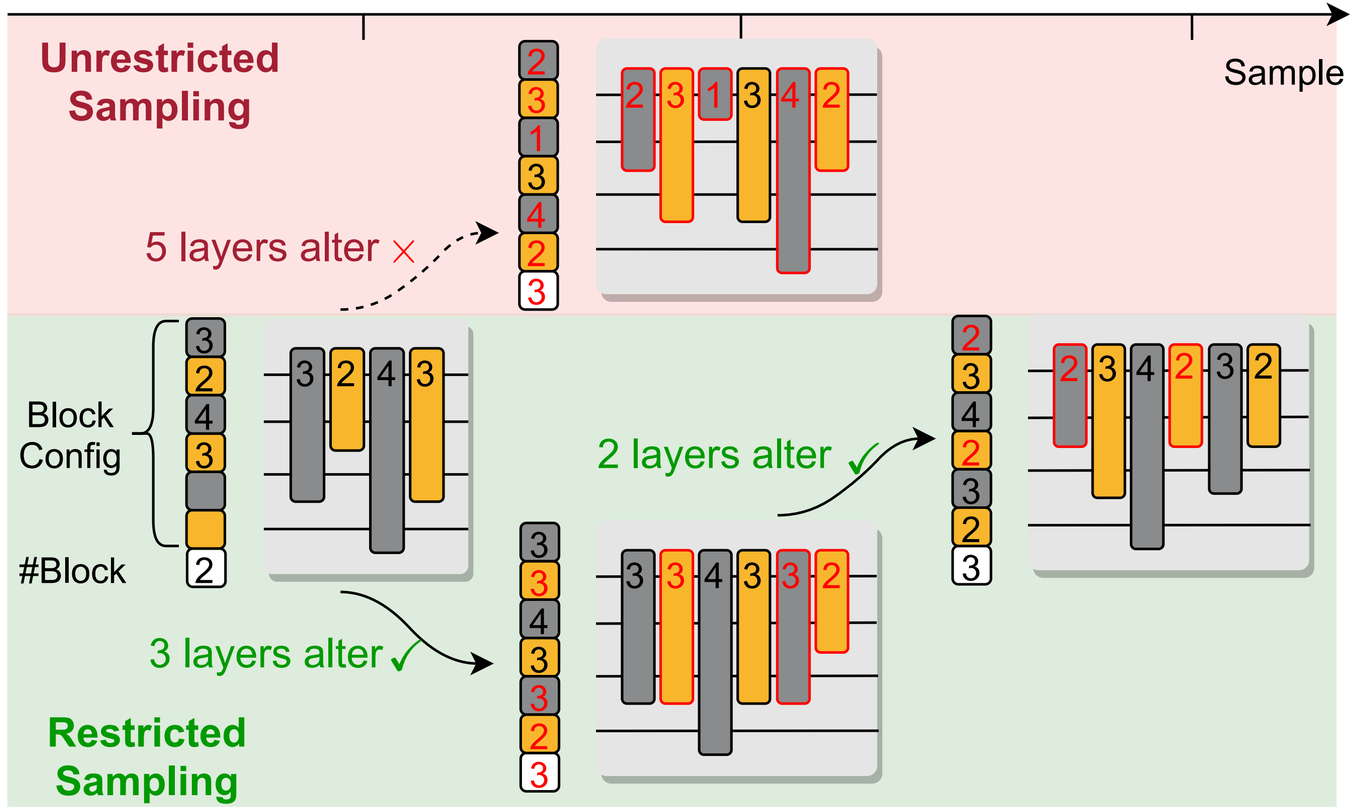
Restricted Sampling

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



Restricted Sampling

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

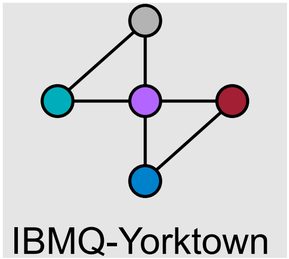


QuantumNAS

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

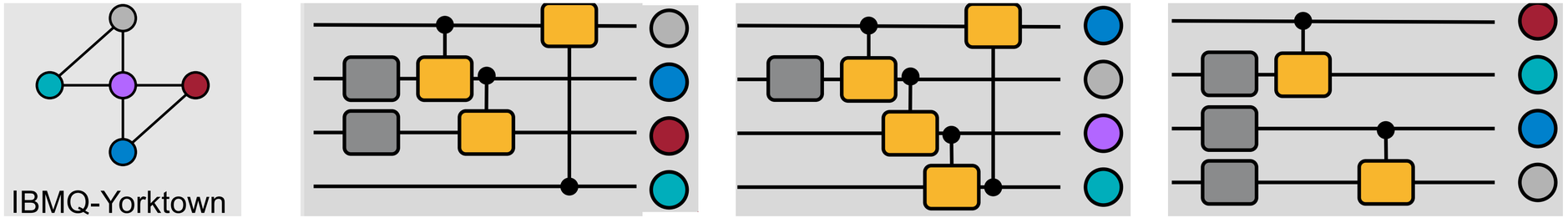
Noise-Adaptive Evolutionary Co-Search

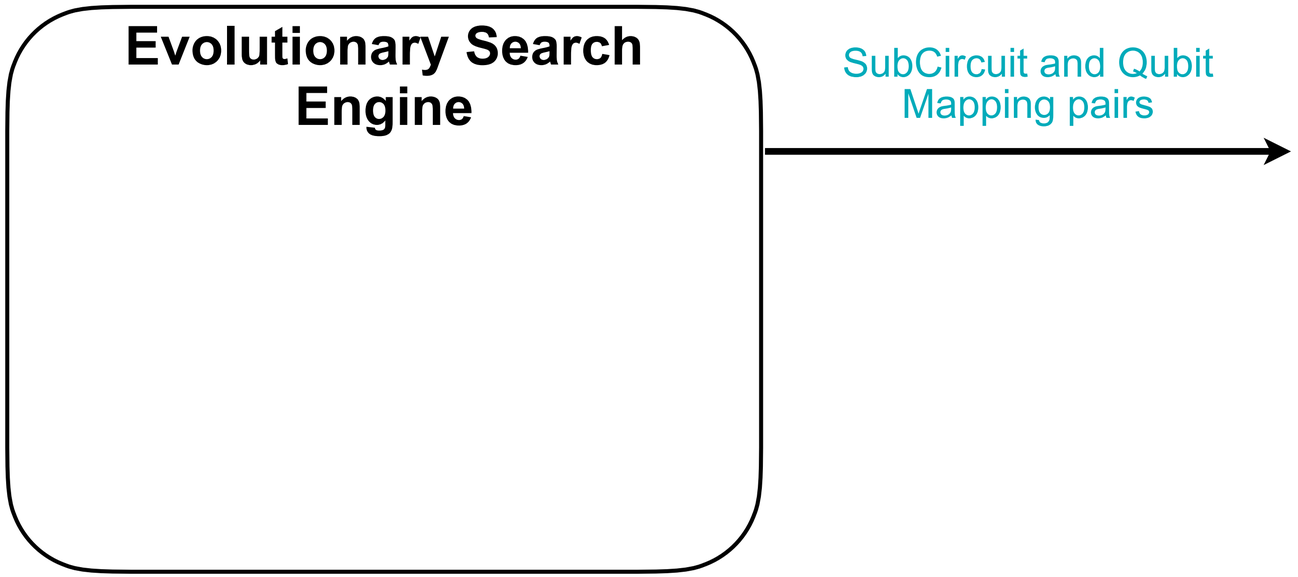
- Search the best SubCircuit and its qubit mapping on target device



Noise-Adaptive Evolutionary Co-Search

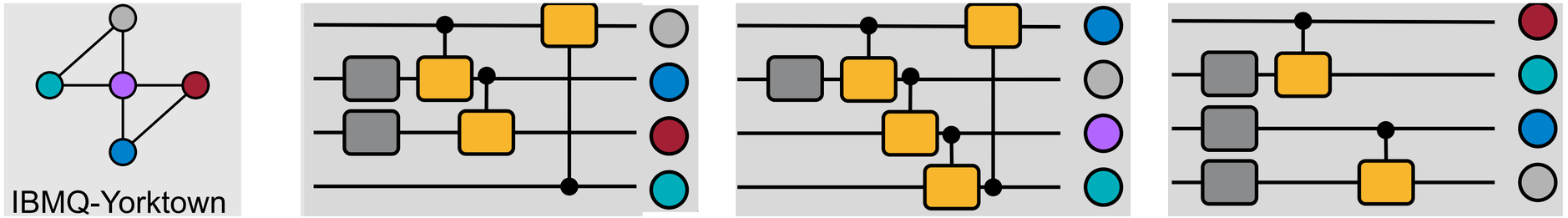
- Search the best SubCircuit and its qubit mapping





Noise-Adaptive Evolutionary Co-Search

- Search the best SubCircuit and its qubit mapping



Evolutionary Search Engine

SubCircuit and Qubit Mapping pairs

Parameters from SuperCircuit

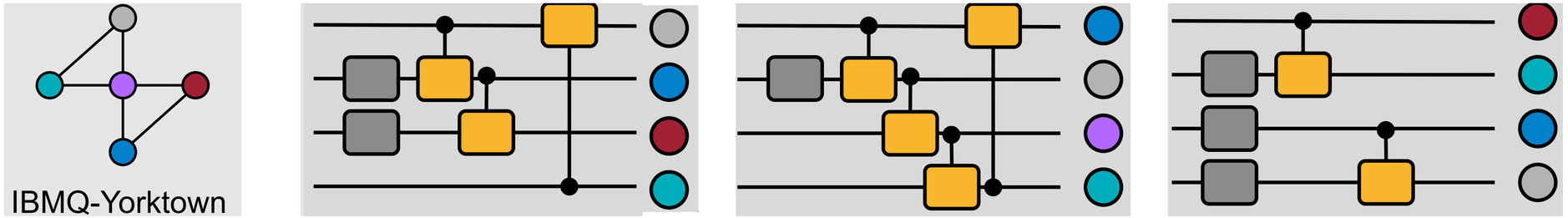
Simulated with device noise model

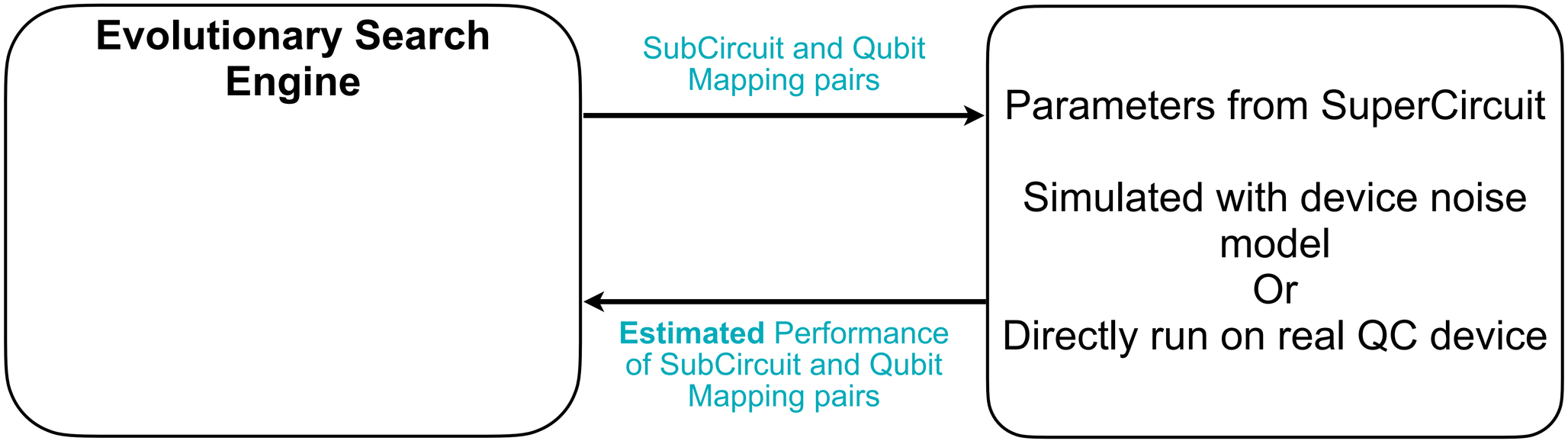
Or

Directly run on real QC device

Noise-Adaptive Evolutionary Co-Search

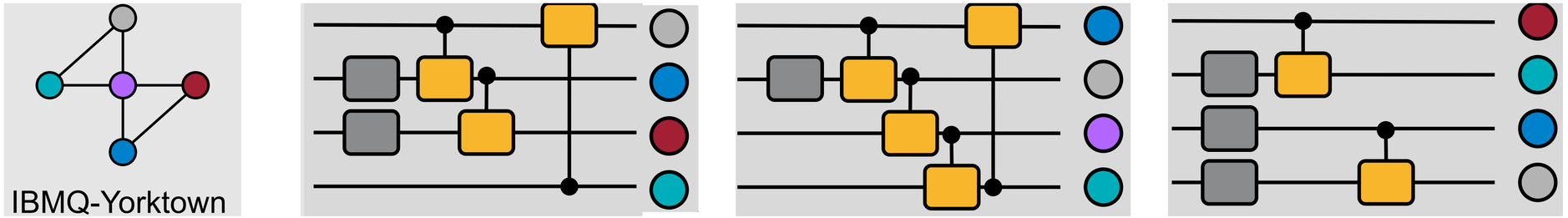
- Search the best SubCircuit and its qubit mapping





Noise-Adaptive Evolutionary Co-Search

- Search the best SubCircuit and its qubit mapping



Evolutionary Search Engine

Mutation: randomly mutate the SubCircuit and qubit mapping

Crossover: generate a new sample from two parent samples

SubCircuit and Qubit Mapping pairs

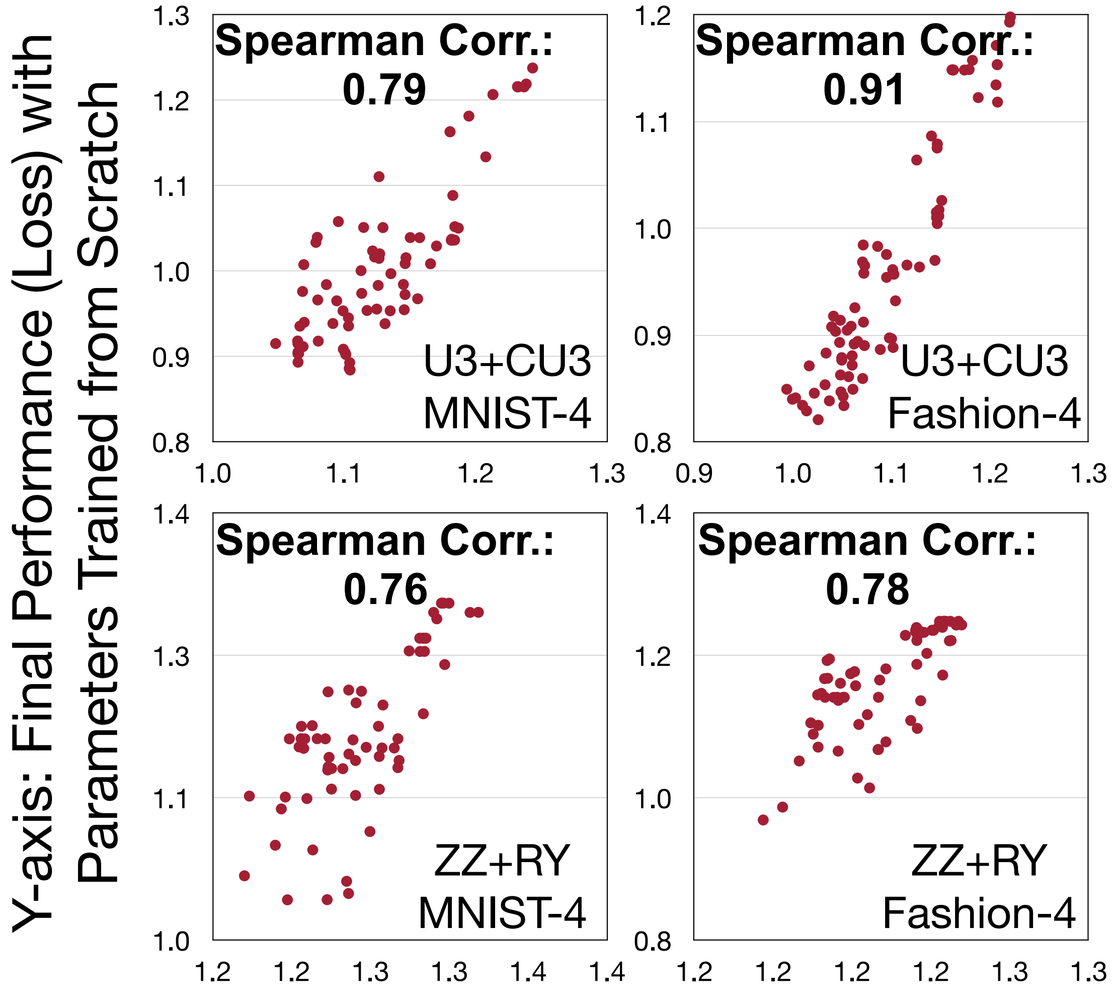
Estimated Performance of SubCircuit and Qubit Mapping pairs

Parameters from SuperCircuit

Simulated with device noise model
Or
Directly run on real QC device

How Reliable is the SuperCircuit?

- Inherited parameters from SuperCircuit can provide accurate relative performance

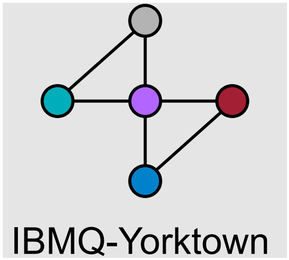


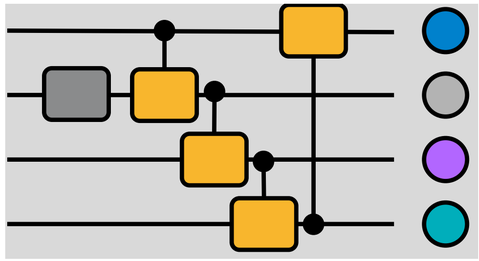
QuantumNAS

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

Searched SubCircuit Training

- Train the SubCircuit parameters from scratch and get final parameters



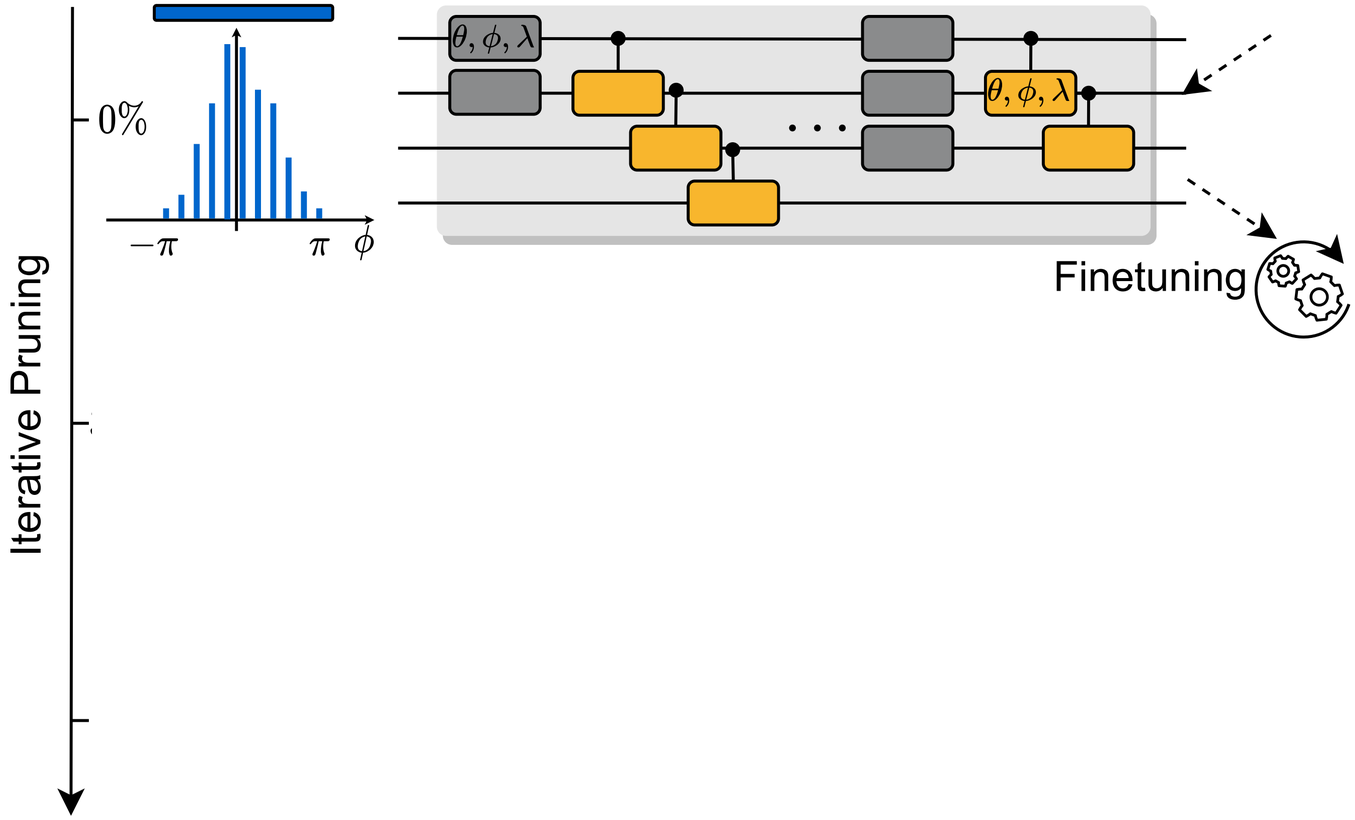


QuantumNAS

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

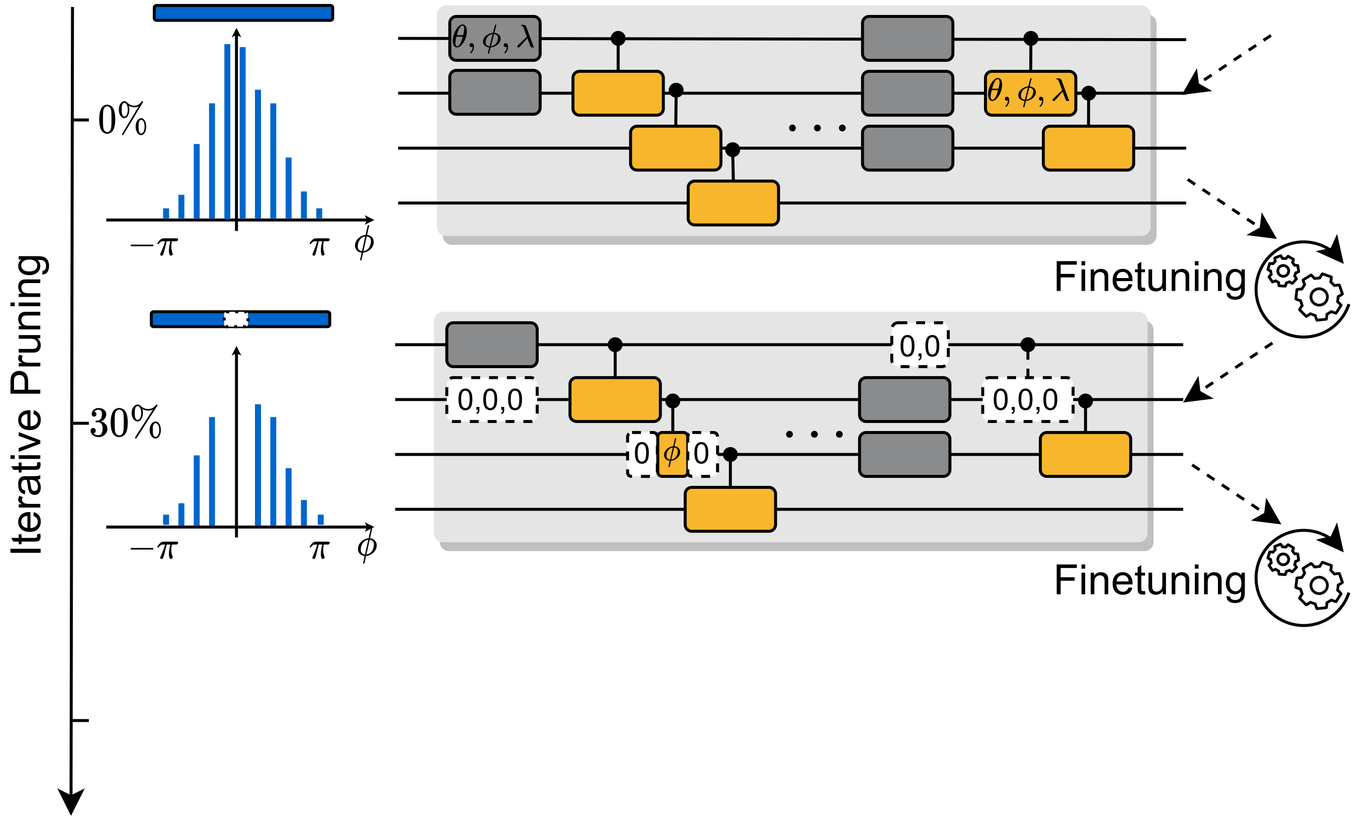
Iterative Pruning

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



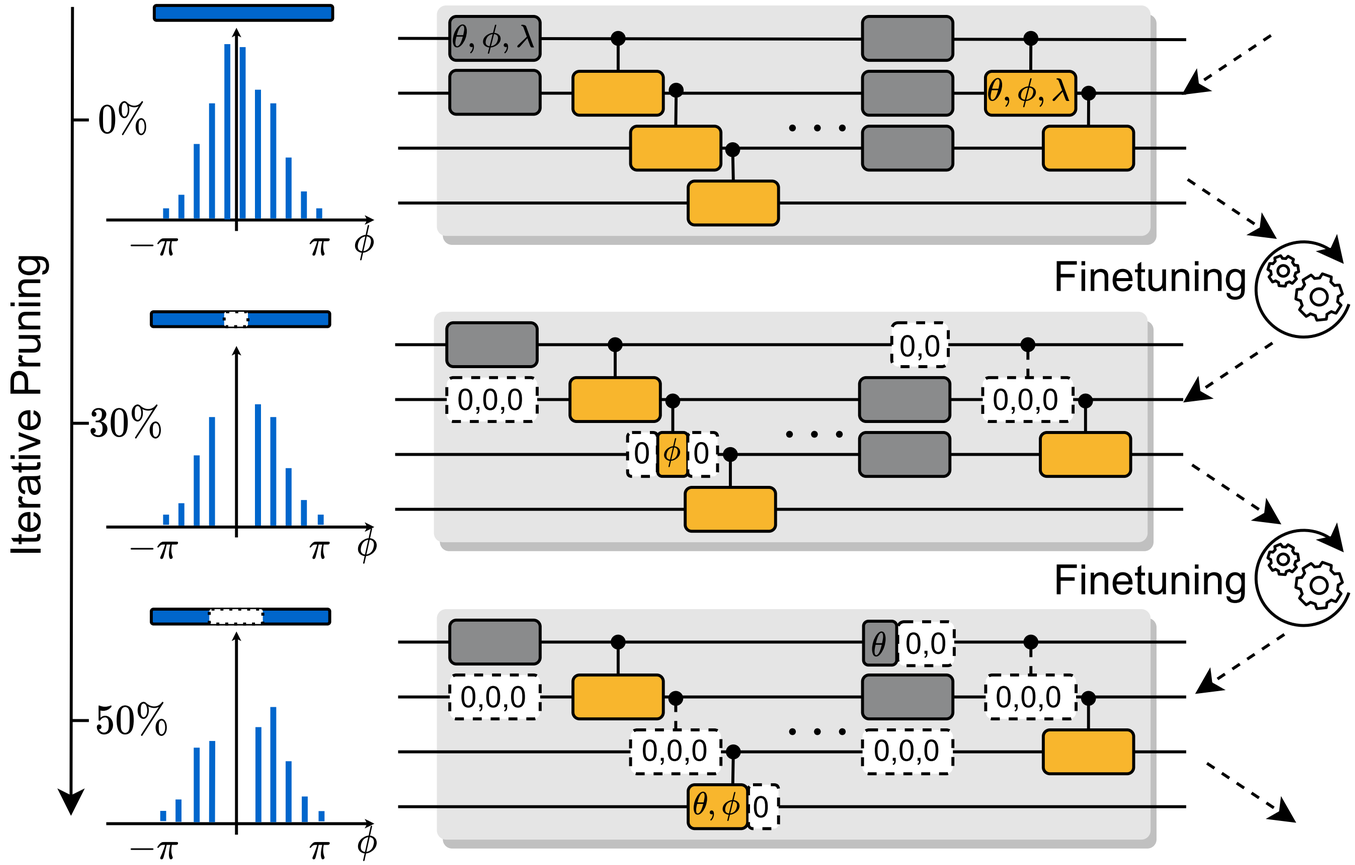
Iterative Pruning

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



Iterative Pruning

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



Parameterized QC Library — QuantumEngine

- Easy construction of parameterized quantum circuits in PyTorch.
- Support **batch** mode inference and training on CPU/GPU.
- Support both **static and dynamic** computation graph for easy debugging.
- Support **easy deployment** on real quantum devices such as IBMQ.

Parameterized QC Library — QuantumEngine

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

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

    self.encoder_gates = [tqf.rx] * 4 + [tqf.ry] * 4 + \
                         [tqf.rz] * 4 + [tqf.rx] * 4
    self.rx0 = tq.RX(has_params=True, trainable=True)
    self.ry0 = tq.RY(has_params=True, trainable=True)
    self.rz0 = tq.RZ(has_params=True, trainable=True)
    self.crx0 = tq.CRX(has_params=True, trainable=True)
```

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

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

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

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

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

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

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

  return x
```

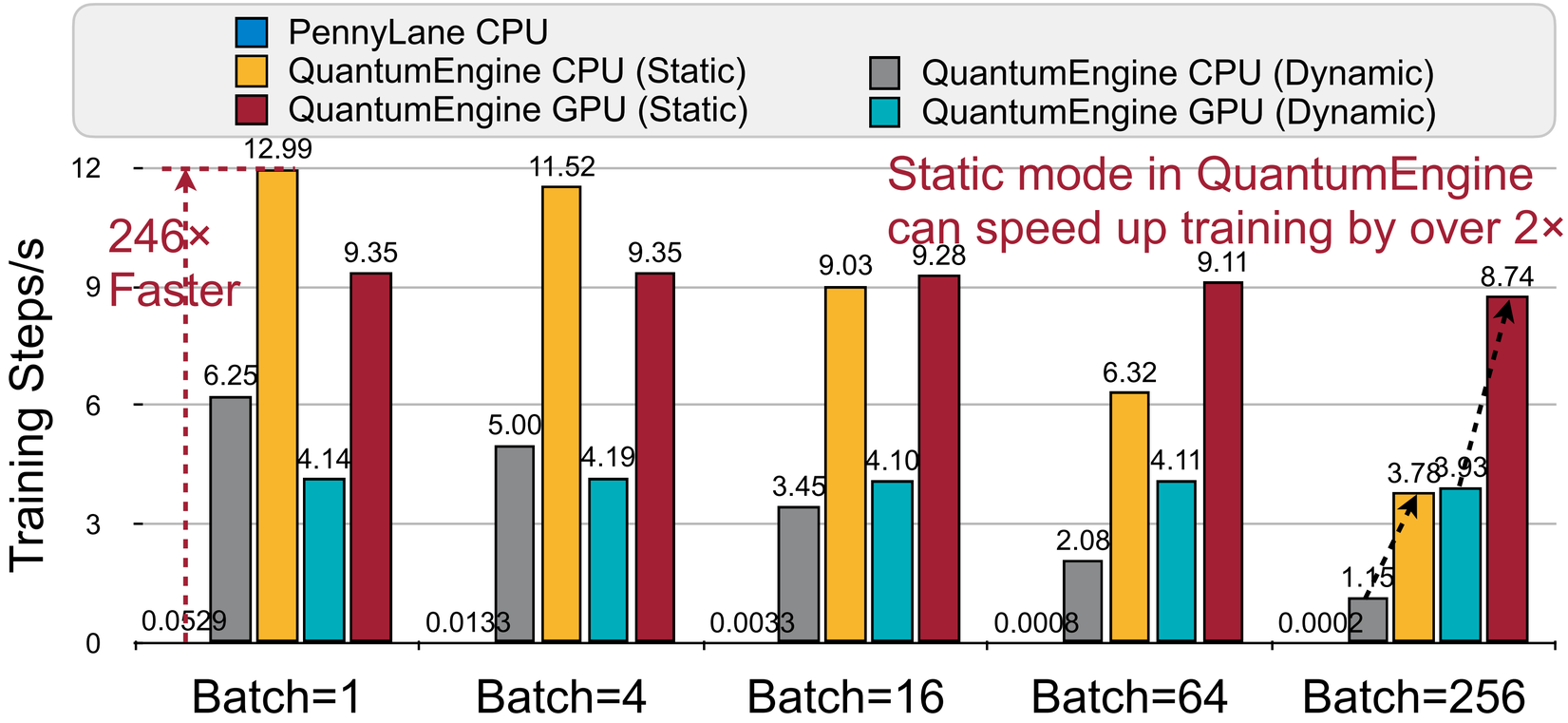
Parameterized QC Library — QuantumEngine

- Easy deployment:

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

Parameterized QC Library — QuantumEngine

- Fast training speed:



Outline

- Introduction
- Background
- QuantumNAS
  - SuperCircuit Training
  - Noise-Adaptive Co-Search
  - Searched SubCircuit Training
  - Gate Pruning
- Evaluation
- Conclusion

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

Evaluation Setups: Design Spaces

- Circuit Design Spaces
  - ‘U3+CU3’: one block has one U3 layer with a U3 gate on each qubit, and one CU3 layer with ring connections, 8 blocks
  - ‘ZZ+RY’: one block has one ZZ layer, and one RY layer, 8 blocks
  - ‘RXYZ’: one block has one RX, RY, RZ and CZ layer, 8 blocks
  - ‘ZZ+XX’: one block has one ZZ layer, and one XX layer, 8 blocks
  - ‘RXYZ+U1+CU3’: one block has 11 layers: RX, S, CNOT, RY, T, SWAP, RZ, H, √SWAP, U1, and CU3, 4 blocks
  - ‘IBMQ Basis’: use IBMQ basis gate set: one block has 6 layers: RZ, X, RZ, SX, RZ, CNOT, 20 blocks

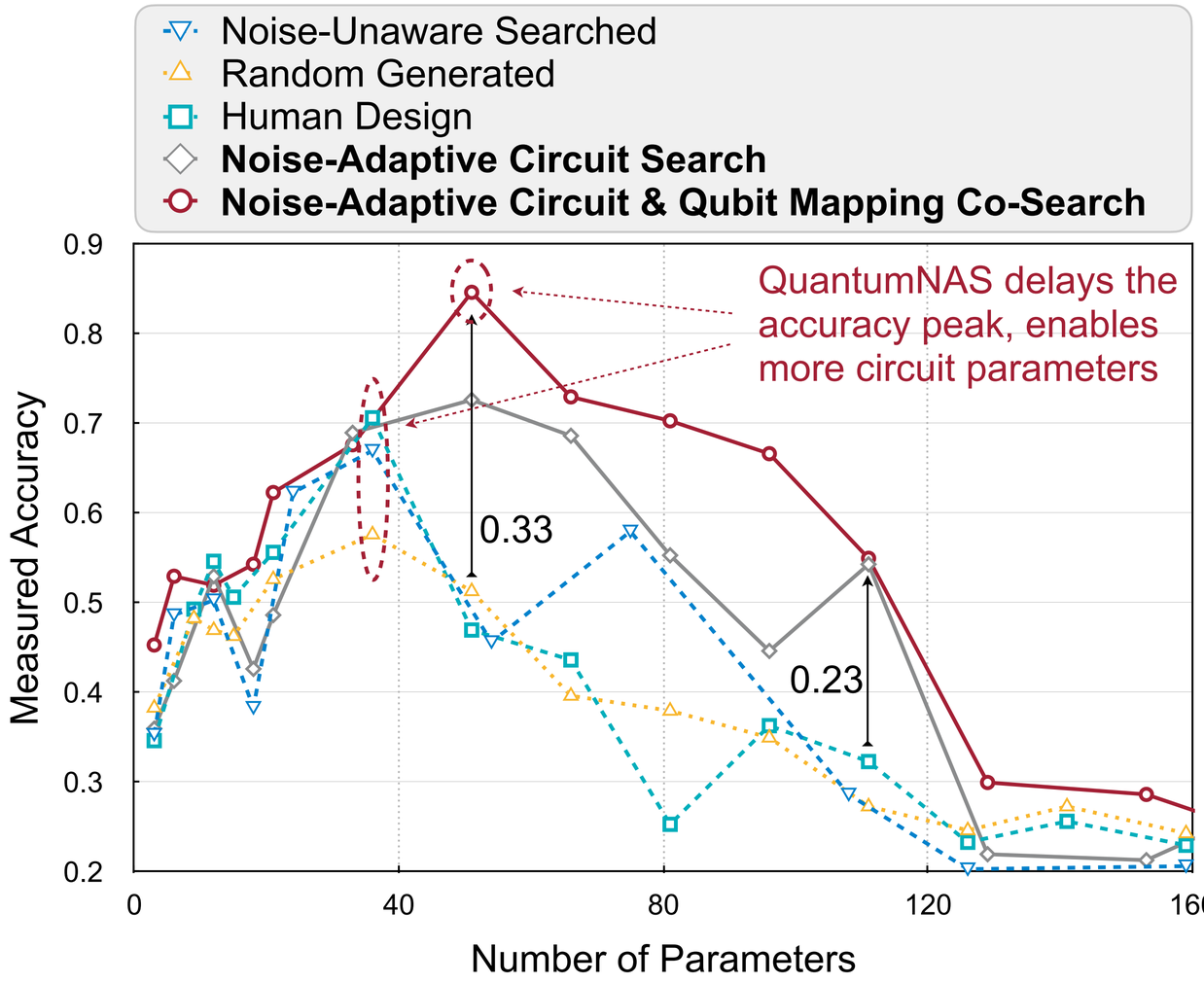
Evaluation Setups: Baselines

- 6 Baselines for QML/QVE
  - *Noise-Unaware Search*: no noise information is involved during search
  - *Random generation*: same gate set, same #parameters with QuantumNAS
  - *Human Design*: same #parameters with QuantumNAS, full width in the front blocks
  - *Human+noise-adaptive mapping*: human design with noise-adaptive qubit mapping[1]
  - *Human+Sabre mapping*: human design with Sabre qubit mapping[2]
  - *Human(1/2 #params)+Sabre mapping:* 1/2 #parameters with QuantumNAS with Sabre qubit mapping
- UCCSD for VQE

[1]Murali, Prakash, et al. "Noise-adaptive compiler mappings for noisy intermediate-scale quantum computers." ASPLOS. 2019.
[2]Li, Gushu, Yufei Ding, and Yuan Xie. "Tackling the qubit mapping problem for NISQ-era quantum devices." ASPLOS. 2019.

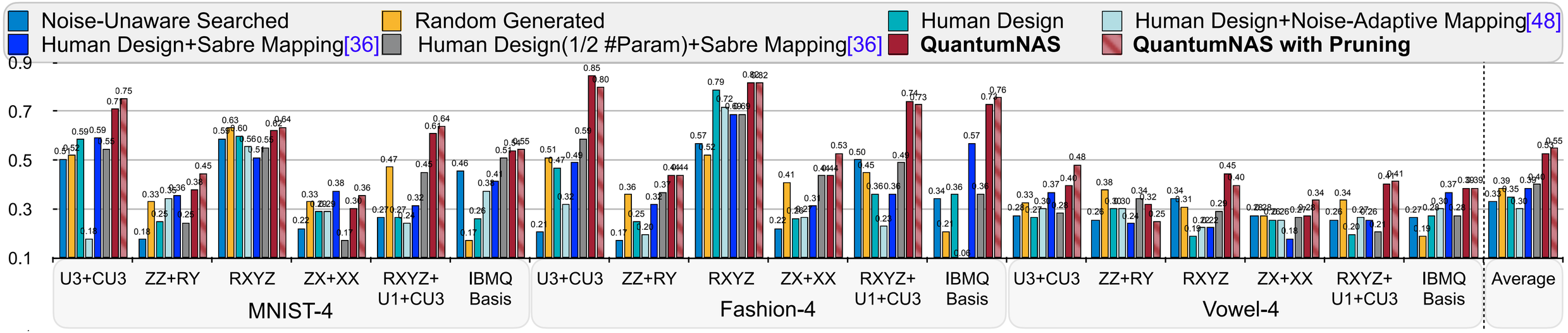
QML Results

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



QML Results

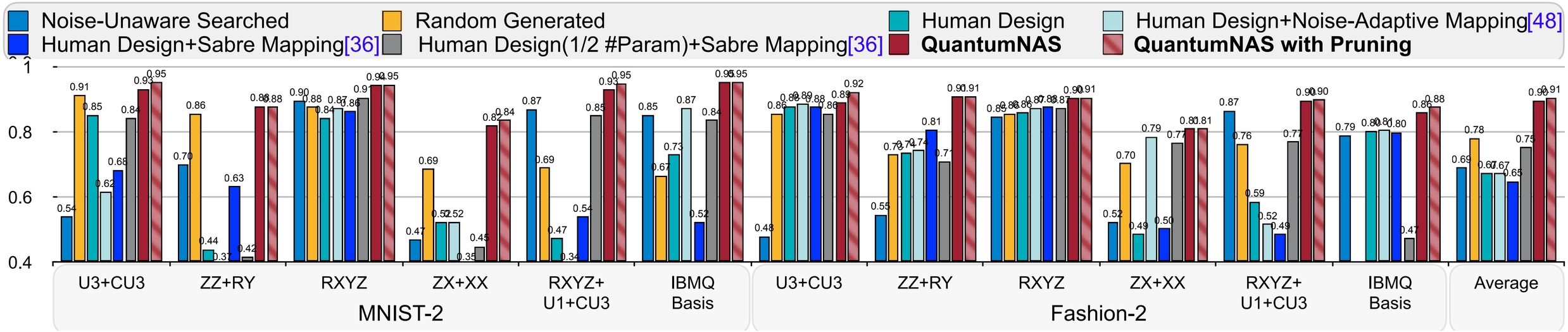
- 4-classifications: MNIST-4, Fashion-4, Vowel-4 in 6 design spaces on IBMQ-Yorktown



Method	Noise-Unaware Searched	Random	Human	Human+Noise -adaptive mapping	Human+ Sabre mapping	Human(1/2params)+Sabre mapping	**QuantumNAS**	**QuantumNAS w/ Pruning**
Average Accuracy	0.33	0.39	0.35	0.30	0.39	0.40	**0.53**	**0.55**

QML Results

- 2-classifications: MNIST-2 and Fashion-2 in 6 design spaces on IBMQ-Yorktown



Method	Noise-Unaware Searched	Random	Human	Human+Noise -adaptive mapping	Human+ Sabre mapping	Human(1/2params)+Sabre mapping	**QuantumNAS**	**QuantumNAS w/ Pruning**
Average Accuracy	0.69	0.78	0.67	0.67	0.65	0.75	**0.90**	**0.91**

Results on Different Devices

- On different 5-Qubit devices
- MNIST-4, Fashion-4, Vowel-4, MNIST-2, Fashion-2 averaged

Method	Noise-Unaware Searched	Random	Human	**QuantumNAS**
Belem (5Q, 16QV)	0.47	0.50	0.67	**0.76**
Quito (5Q, 16QV)	0.73	0.68	0.74	**0.79**
Athens (5Q, 32QV)	0.50	0.63	0.68	**0.77**
Santiago (5Q, 32QV)	0.74	0.73	0.75	**0.80**

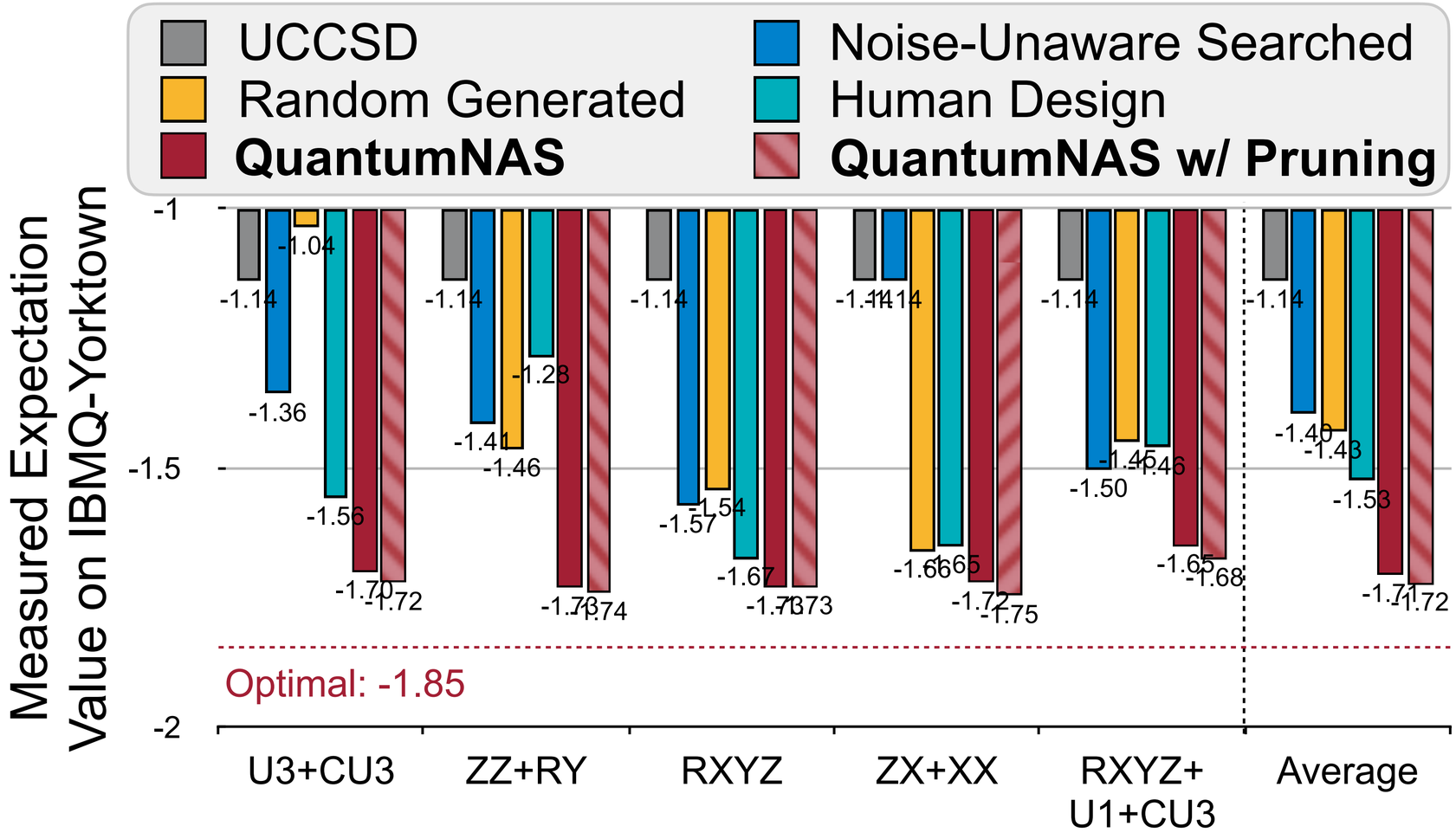
On Large Devices

- On large devices
- MNIST-4, Fashion-4, Vowel-4, MNIST-2, Fashion-2, MNIST-10 averaged

Method	Noise-Unaware Searched	Random	Human	**QuantumNAS**
Melbourne (15Q, 8QV, use 15Q)	0.31	0.42	0.53	**0.60**
Guadalupe (16Q, 32QV, use 16Q)	0.39	0.42	0.58	**0.62**
Montreal (27Q, 128QV, use 21Q)	0.57	0.48	0.61	**0.65**
Manhattan (65Q, 32QV, use 21Q)	0.52	0.53	0.62	**0.66**

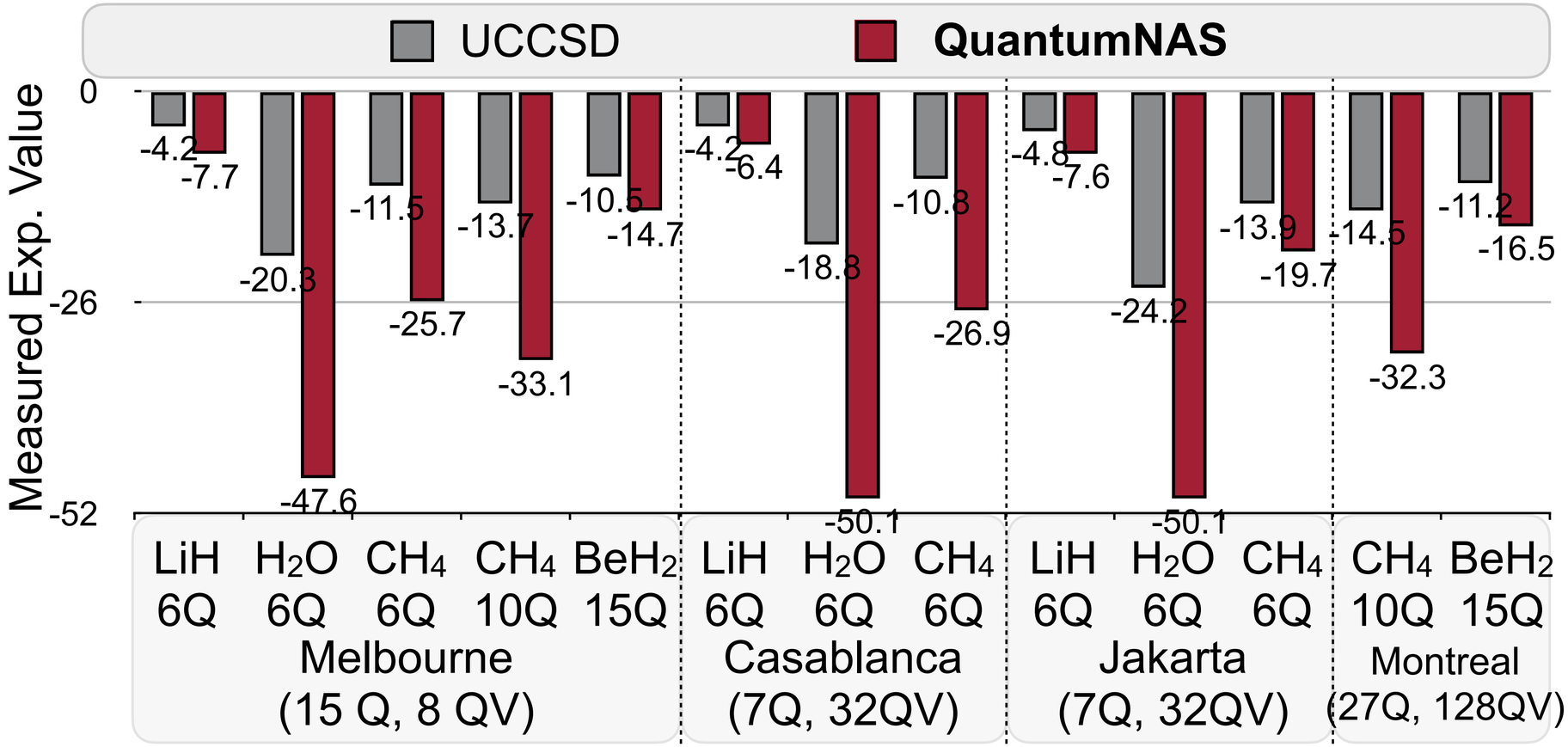
H2 VQE Results

- H2 in different design spaces on IBMQ-Yorktown



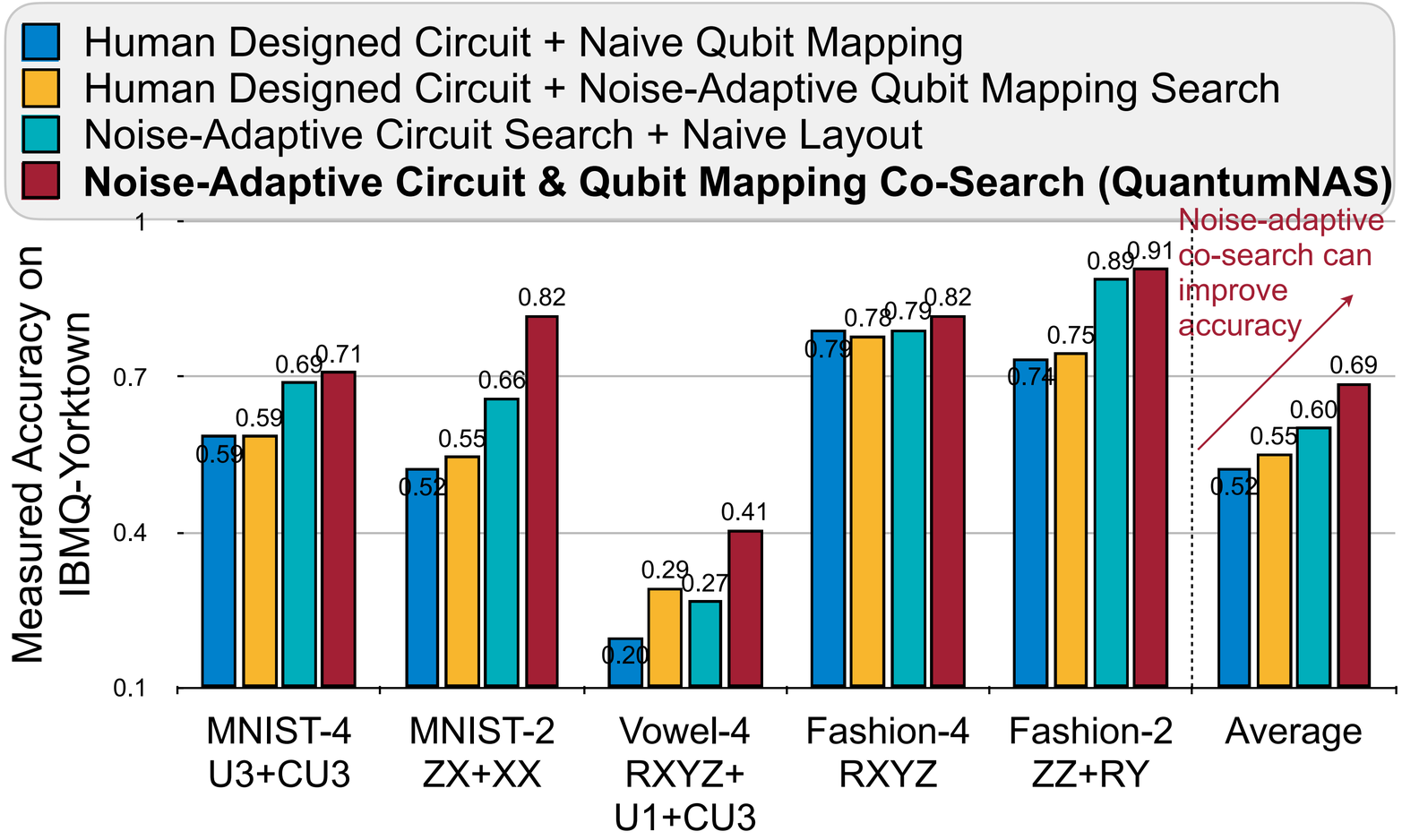
VQE Results of Various Molecules

- VQE on different devices



Effect of Circuit & Qubit Mapping Co-Search

- Co-search is better than only search ansatz/mapping



Scalability

- The whole process can be done on quantum machines
  - SuperCircuit, SubCircuit training with parameter shift on QC device (no harder than training a normal variational quantum circuit)
  - Search with evaluation results on QC device
- Even with whole pipeline on Quantum, QuantumNAS still reduces search cost, because only one SuperCircuit is trained instead of many circuit candidates.
- Different devices can reuse the same SuperCircuit

Time Cost

- On 1 Nvidia Titan RTX 2080 ti GPU

Stage	SuperCircuit Training	Noise-Adaptive Co-search	SubCircuit Training	Deployment on Real QC
4Q	0.5h	3h	0.5h	0.5h
15Q	5h	5h	5h	1h
21Q	20h	10h	15h	1h

Outline

- Introduction
- Background
- QuantumNAS
  - SuperCircuit Training
  - Noise-Adaptive Co-Search
  - Searched SubCircuit Training
  - Gate Pruning
- Evaluation
- Conclusion

Conclusion


opensource

- SuperCircuit based co-search for most robust circuit Ansatz and qubit mapping
- Iterative pruning to further remove redundant gates
- Achieved over 95% 2-class, 85% 4-class and 32% 10-class classification accuracy
- QC library open-sourced: https://github.com/Hanrui-Wang/pytorch-quantum
- Outlook:
  - A general method applicable to all parameterized QC program
  - Search for Ansatz to alleviate Barren Plateau
  - Search for best embedding/readout

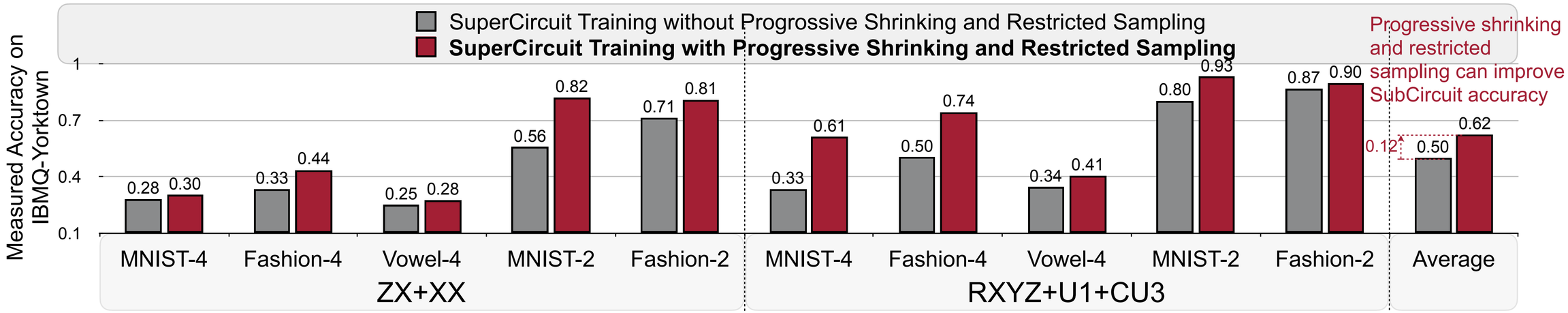
Thank you!

Backup Slides

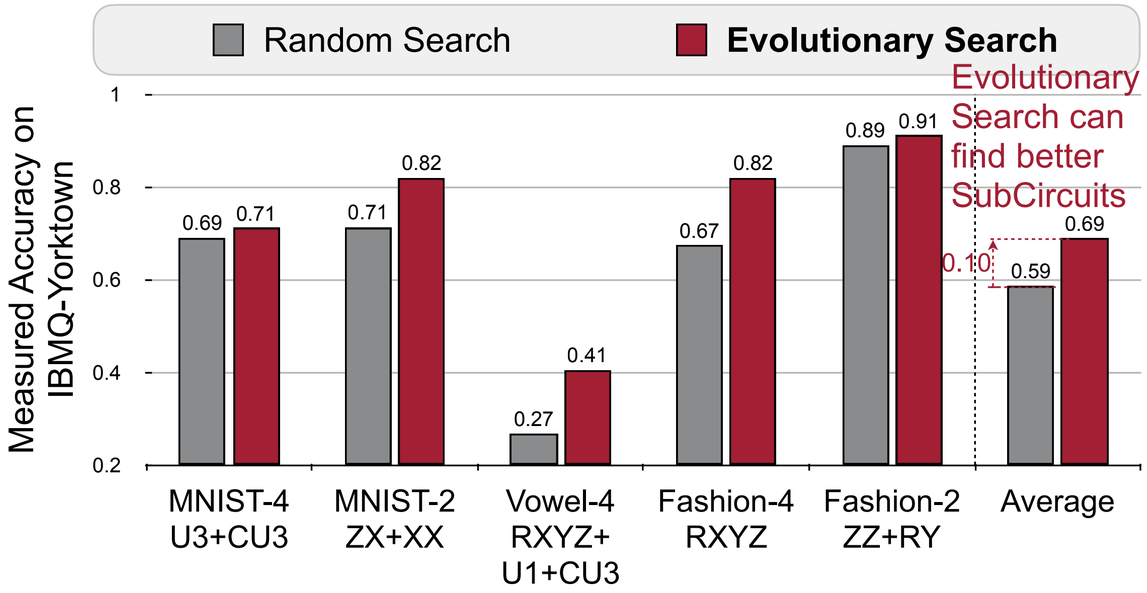
Iterative pruning

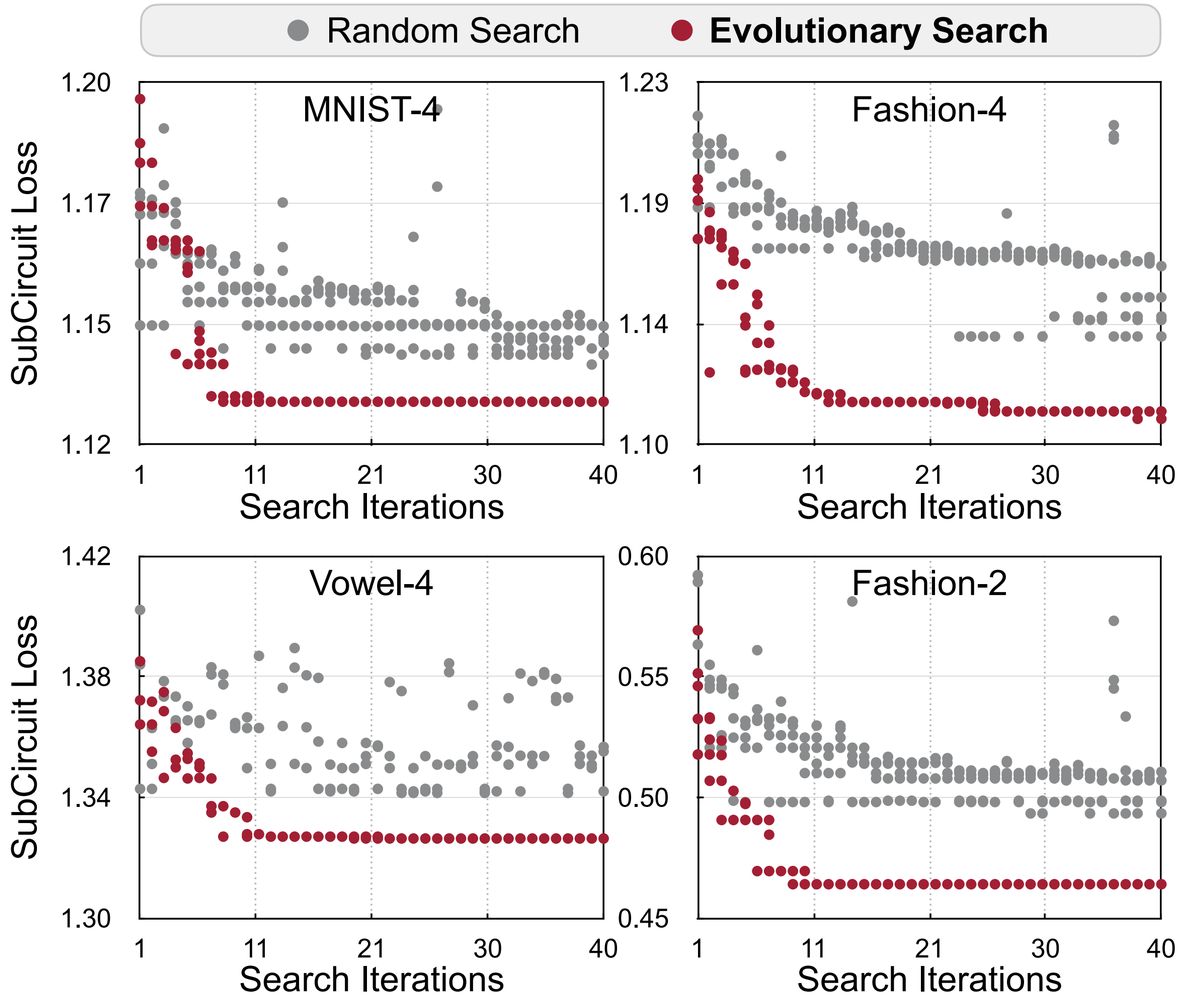
U3	$\lvert(\theta,\phi,\lambda)$	$\lvert(\mathbf{0},\phi,\lambda)$	$\lvert(\theta,\phi,\mathbf{0})$	$\lvert(\theta,\mathbf{0},\mathbf{0})$	$\lvert(\mathbf{0},\phi,\mathbf{0})$	$\lvert(\mathbf{0},\mathbf{0},\lambda)\rvert$
#Compiled Gates	5	1	4	4	1	1

Progressive shrinking

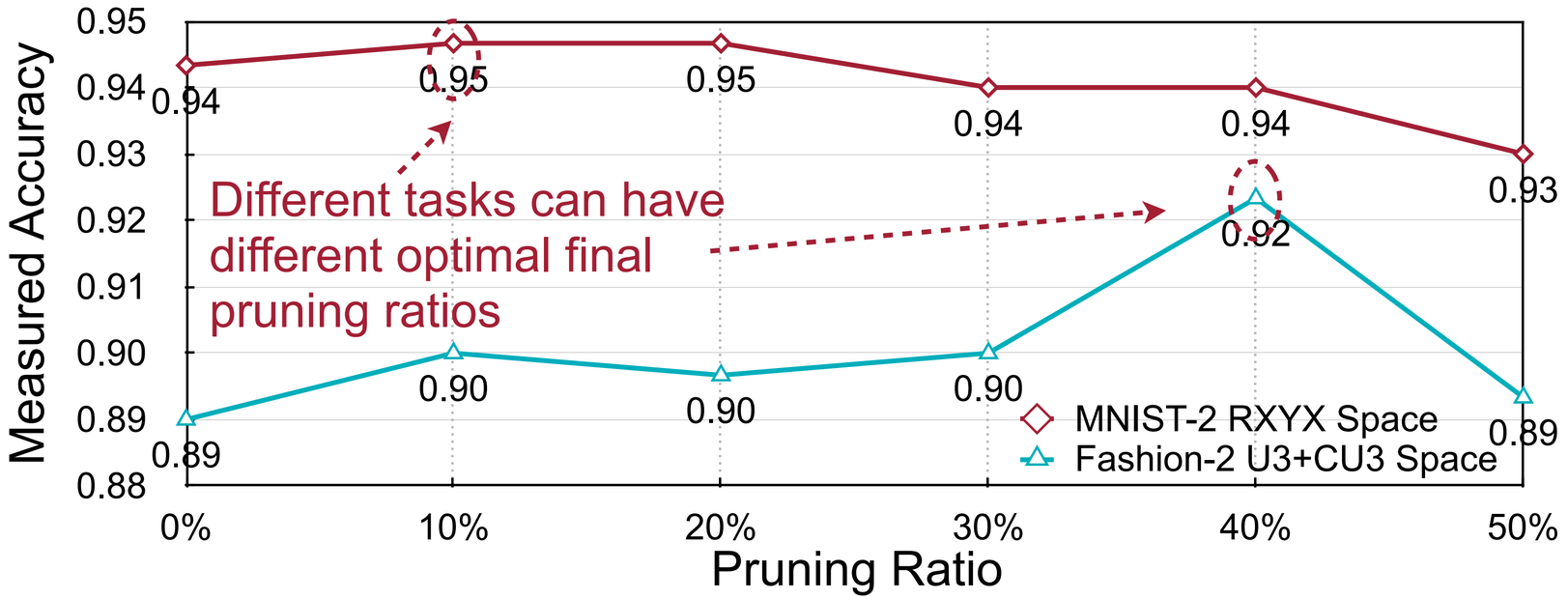


Random search vs evolutionary search

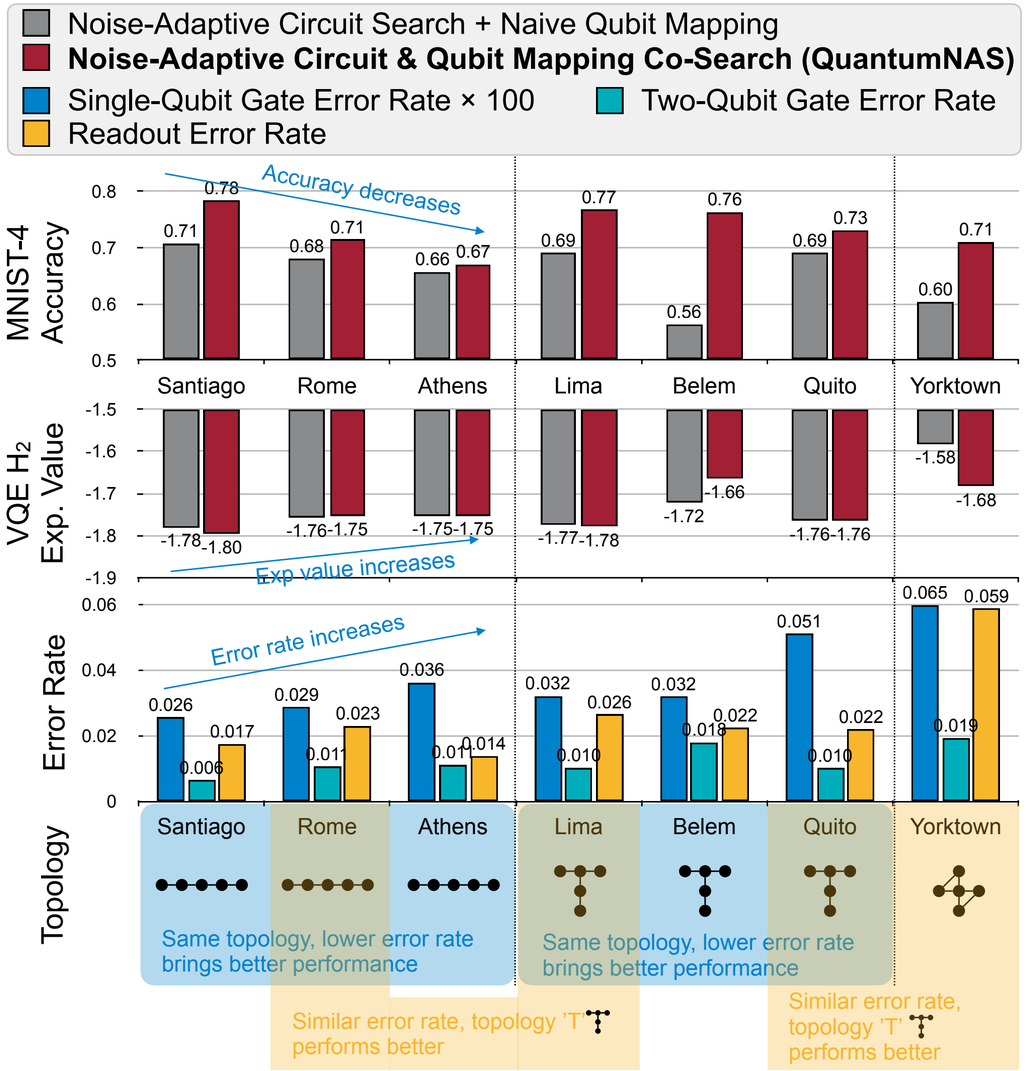




Pruning


Measured Accuracy
0.95
0.94
0.93
0.92
0.91
0.90
0.89
0.88
0.94
0.95
0.95
0.94
0.94
0.93
0.92
0.89
0.90
0.90
0.90
0.89
Different tasks can have different optimal final pruning ratios
MNIST-2 RXYX Space
Fashion-2 U3+CU3 Space
0%
10%
20%
30%
40%
50%
Pruning Ratio

Topology/Error rate



Device-specific

Run on ↓ Searched for →	Yorktown	Belem	Santiago
Yorktown	**0.85**	0.60	0.54
Belem	0.67	**0.77**	0.43
Santiago	0.82	0.81	**0.85**