DESIGN AUTOMATION CONFERENCE

59

JULY 10 - 14, 2022

MOSCONE WEST CENTER

SAN FRANCISCO, CA, USA

Towards Quantum Learning Democratization

—— Start from Building a Quantum Neural Network Design Stack

Zhepeng Wang, Zhiding Liang, Yiyu Shi, Weiwen Jiang

JQub @ Mason | SCL @ Notre Dame

07/11/2022

What is Classical AI Democratization & What is the Challenge?

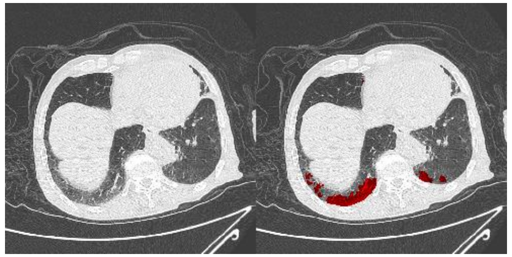
"It's here to collaborate, to augment, to enhance human lives and productivity and make everybody's life better. And related to that, is to **democratize A.I.** in a way that everybody gets benefit. Not just a few, or a selected group." **Fei-Fei Li, 2017**

Medical AI Scenario

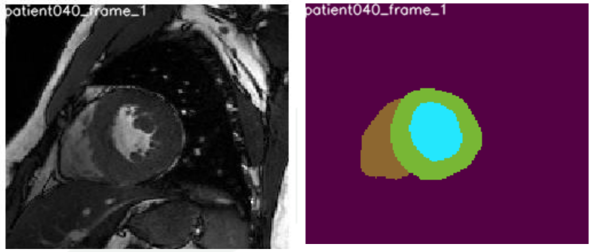
AR/VR in Surgery

Medical Diagnosis

AI Can Perform Medical Tasks



COVID CT Segmentation



Real-Time MRI Segmentation

Let Doctors Design Neural Networks?

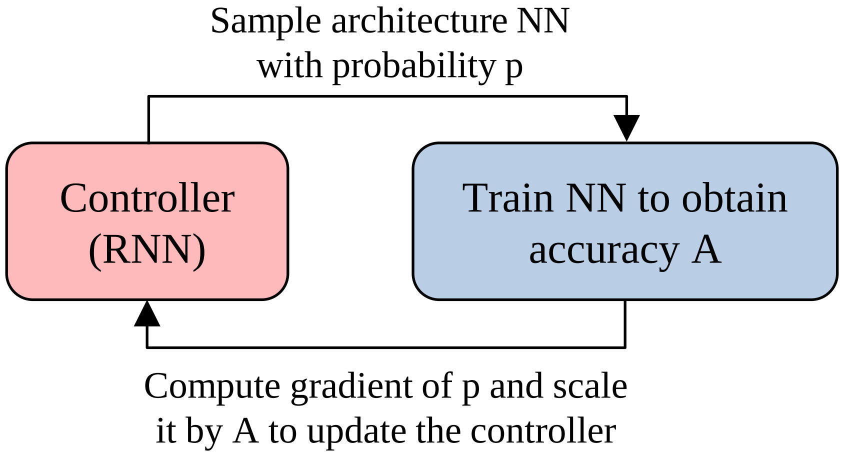
Progress of Classical AI Democratization

Google's Initial Contributions
(Neural Architecture Search)

Given: Dataset

Objective:
- Automated search for NN **(w/o human)**
- Maximize accuracy on the given dataset

Output: A neural network architecture



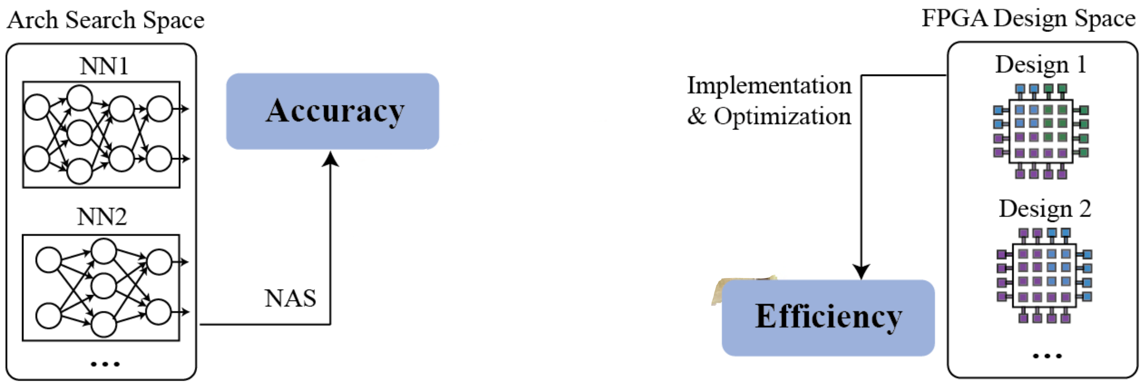
[ref] Zoph, Barret, and Quoc V. Le. "Neural architecture search with reinforcement learning." *ICLR 2017*

Our Contributions
(Network-Accelerator Co-Design)

Given: (1) Dataset; (2) Target hardware, e.g., FPGA.

Objective:
- Automated search for NN and HW design
- Maximize accuracy on the given dataset
- Maximize hardware efficiency

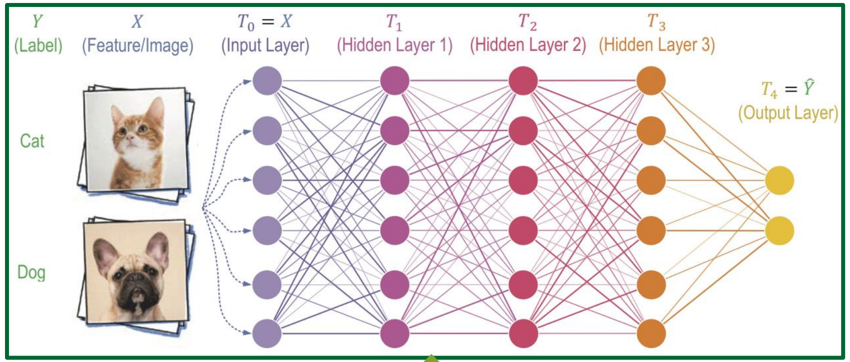
Output: A pair of neural network and hardware design

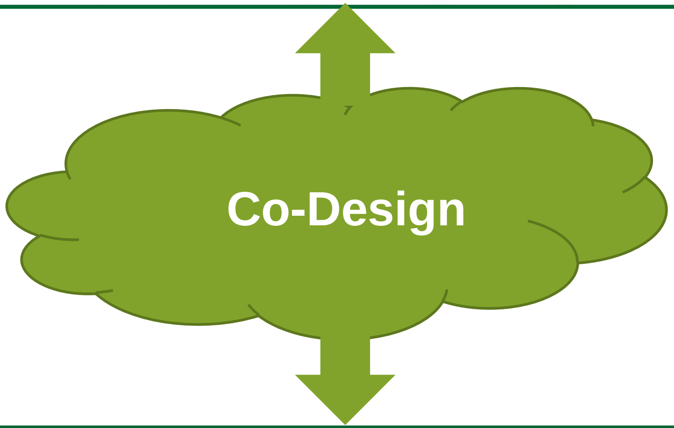


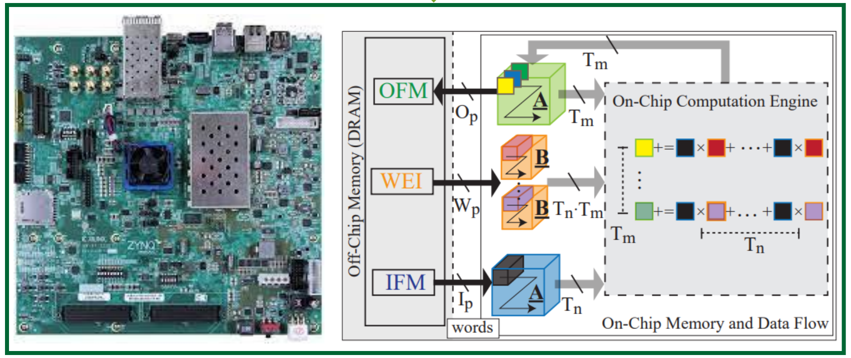
[ref] Jiang, Weiwen, et al. "Accuracy vs. efficiency: Achieving both through fpga-implementation aware neural architecture search." *DAC 2019*. (BEST PAPER NOMINATION)

[ref] Jiang, Weiwen, et al. "Hardware/software co-exploration of neural architectures", TCAD 2020 (BEST PAPER AWARD)

Co-Design Stack of Neural "Architectures"







- What is the best **Neural Network Architecture** for FPGAs
- Model optimization (pruning and quantization)?
- Library

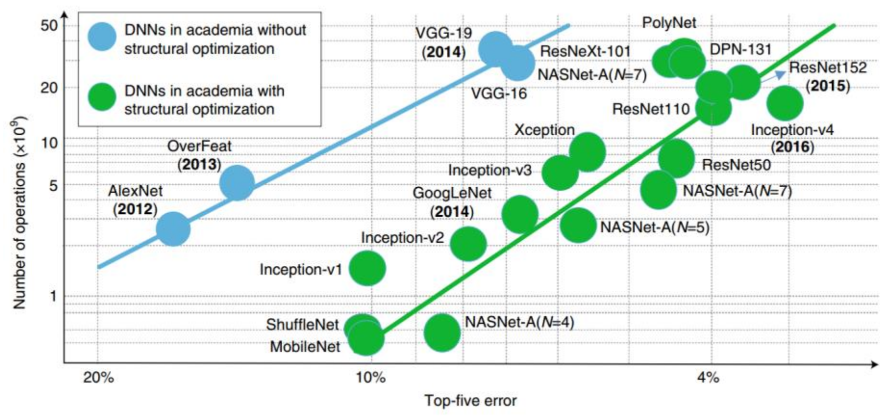
Co-Design Framework (e.g., Our FNAS)		
	Network exploration	NAS (Google)
	Network compression	Deep Comp (Stanford)
	Programming library	DNNBuilder (UIUC)
	Hardware accelerator	DNN on FPGA (UCLA)

- Mapping and scheduling?
- What is the best **FPGA Architecture** for neural networks

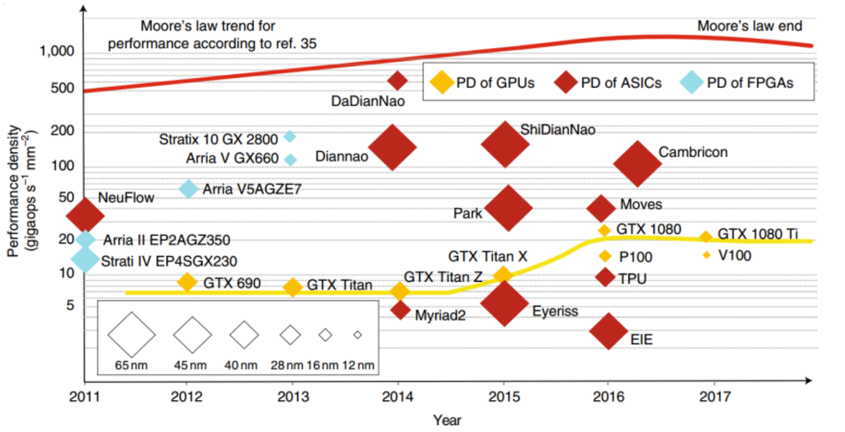
Outline

- Background
- Perspective: Co-Design --- from Classical to Quantum
- Built Design Stack from JQub
  - Quantum Neuron with Quantum Advantage: QuantumFlow
  - Quantum Neural Network Exploration: QF-Mixer
  - Quantum Pluse: VQP
  - Quantum Neural Network Compression: CompVQC
  - Quantum NN Library: QFNN
- Conclusion

Bottlenecks in Classical Computing



Deep neural network grows exponentially



Perf. of classical computing stops increasing

Medical AI Scenario: (Input size exponentially grows from Radiology to Pathology Imaging)

Radiology Imaging

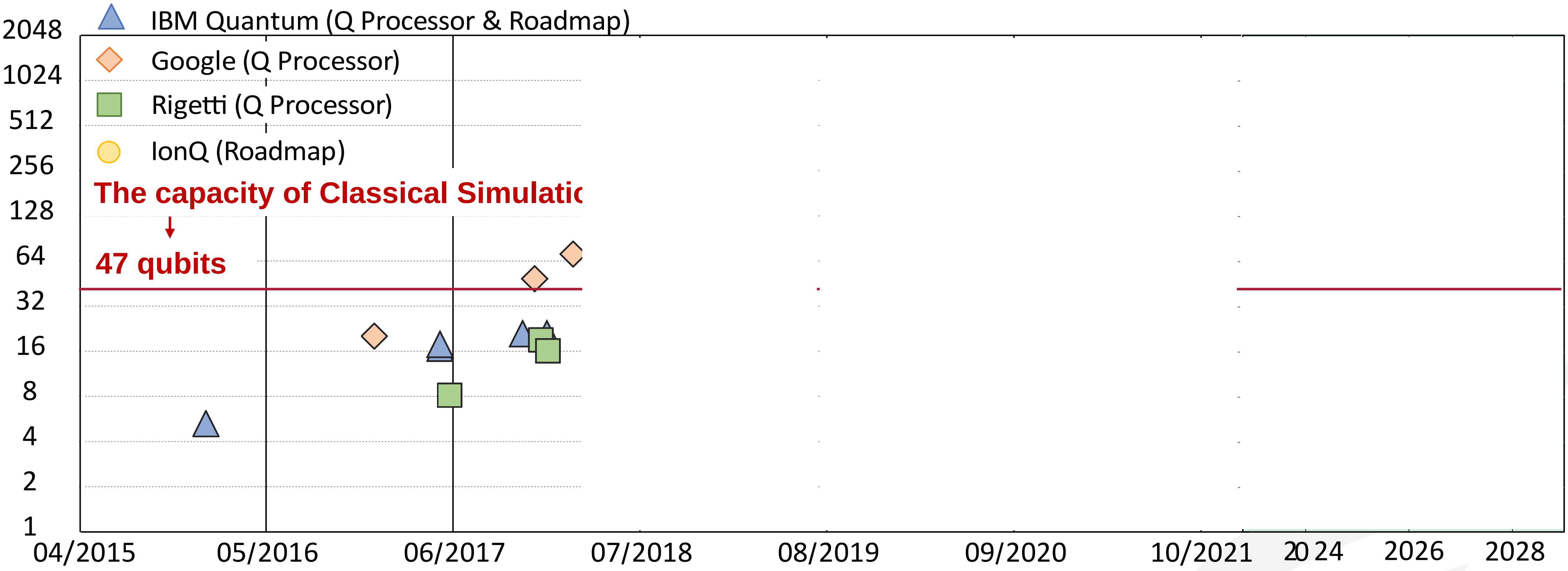
Radiology Modality	Avg. Size (MB)
CT Scan	153.4
MRI	98.6
X-ray angiography	157.5
Ultrasound	69.2
Breast imaging	38.8

Pathology Imaging

Biopsy Type	Compressed Size(MB)/Study	Original Size (GB)
Dermatopathology	1,392 (20x compression)	**27**
Head and neck	1,965 (20x compression)	**38**
Hematopathology	40,300 (40x compression)	**1574**
Neuropathology	1,872 (20x compression)	**37**
Thoracic pathology	3,240 (20x compression)	**63**

[ref] Lauro, Gonzalo Romero, et al. "Digital pathology consultations—a new era in digital imaging, challenges and practical applications." *Journal of digital imaging* 26.4 (2013).

Impossible in Classical But Possible in Quantum Computing

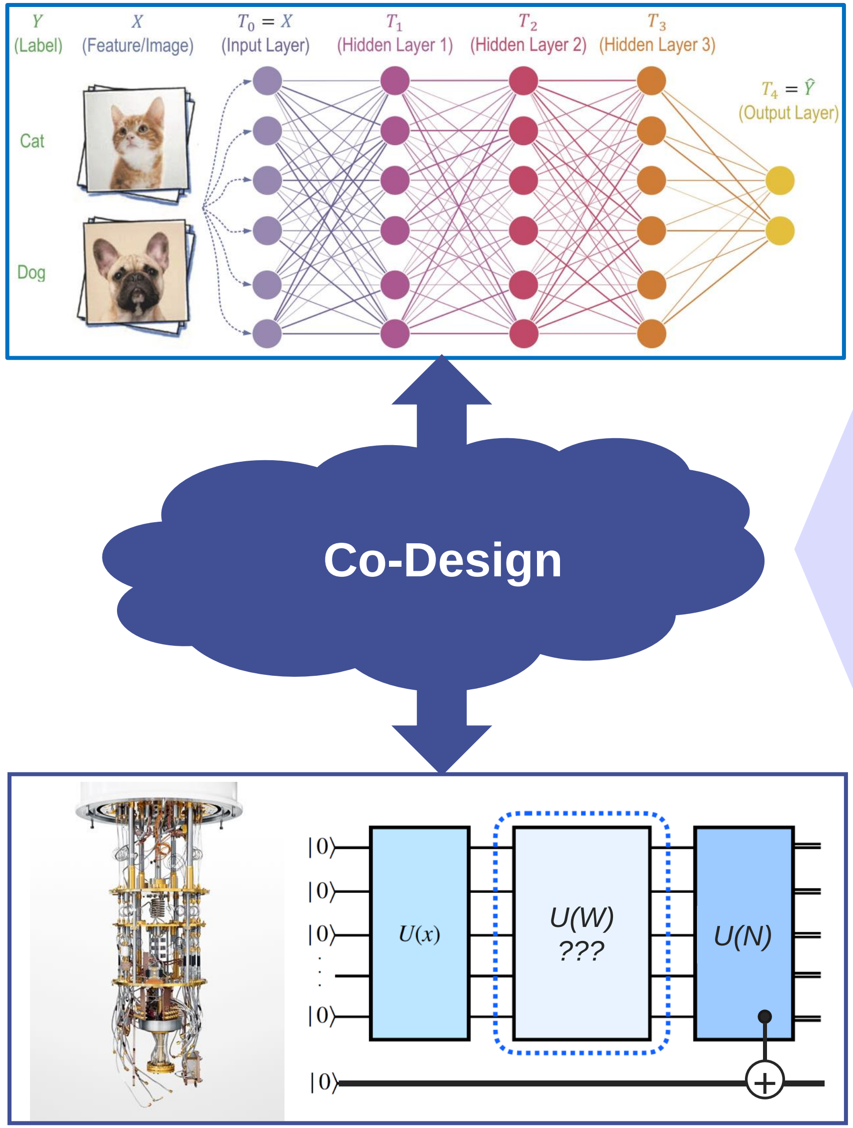


The maximum qubits that supercomputers can simulate for arbitrary circuits is less than 47 qubits.

(1) Summit w/ 2.8 PB memory for **47 qubits**; (2) Sierra w/ 1.38 PB memory for **46 qubits**;
(3) Sunway TaihuLight w/ 1.31 PB memory for **46 qubits**; (4) Theta w/ 0.8 PB memory for **45 qubits**.

[ref] Wu, Xin-Chuan, et al. "Full-state quantum circuit simulation by using data compression." Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. 2019.

Our Works: Co-Design of Neural Networks and Quantum Circuit

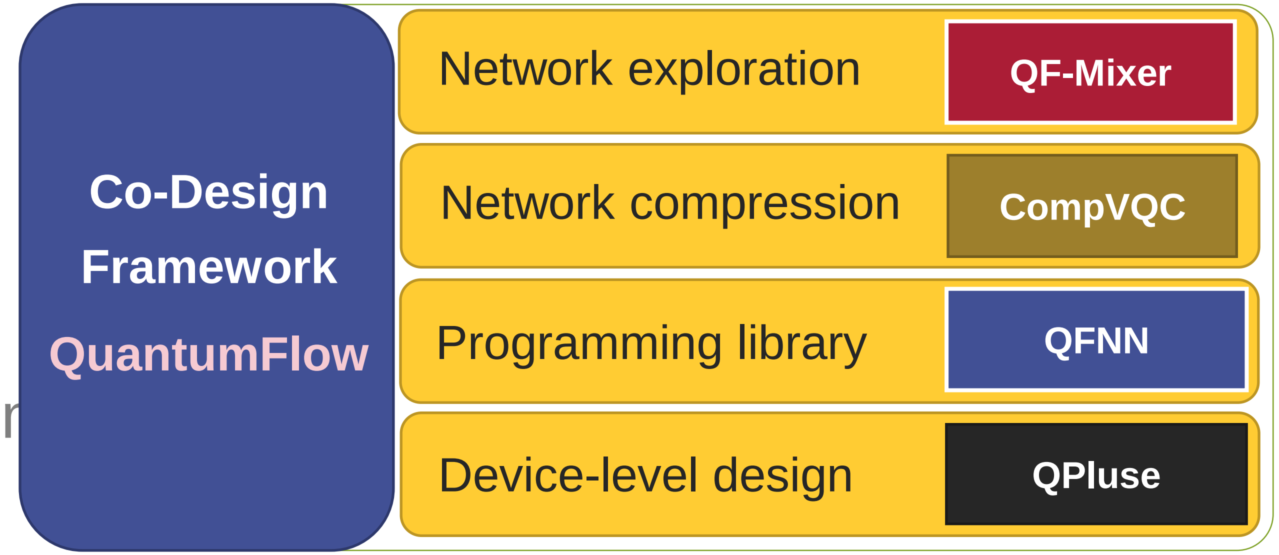


- What is the best **Neural Network Architecture** for QC?
- Can we **compress** the quantum neural network?
- Library

Co-Design Framework QuantumFlow		
	Network exploration	QF-Mixer
	Network compression	CompVQC
	Programming library	QFNN
	Device-level design	QPluse

- ......
- What is the best **QC design** for neural networks?

Outline



- Background
- Perspective: Co-Design --- from
- Built Design Stack from JQub
    - Quantum Neuron with Quantum Advantage: QuantumFlow
    - Quantum Neural Network Exploration: QF-Mixer
    - Quantum Pluse: VQP
    - Quantum Neural Network Compression: CompVQC
    - Quantum NN Library: QFNN
- Conclusion

Quantum Neuron: QuantumFlow

A Co-Design Framework of Neural Networks and Quantum Circuits Towards Quantum Advantage

Published at Nature Communications 2021

nature COMMUNICATIONS

Presenter: Weiwen Jiang

Co-Design Framework QuantumFlow

Network exploration	**QF-Mixer**
Network compression	**CompVQC**
Programming library	**QFNN**
Device-level design	**QPluse**

Invited & Tutorial Talks:

IBM **Quantum** Summit September 15-17, 2020

IEEE QUANTUM WEEK

IEEE International Conference on Quantum Computing and Engineering — QCE21

59 DESIGN AUTOMATION CONFERENCE

EMBEDDED SYSTEMS WEEK

IEEE/ACM 2021 INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN ICCAD 40th Edition

VILLANOVA UNIVERSITY

…

Classical Bit vs. Quantum Bit

Classical Bit

- 2 basic states — **0**, **1** (OFF or ON)
- Mutually exclusive

$$X = 0 \ \boldsymbol{or} \ 1$$

Classical Bit

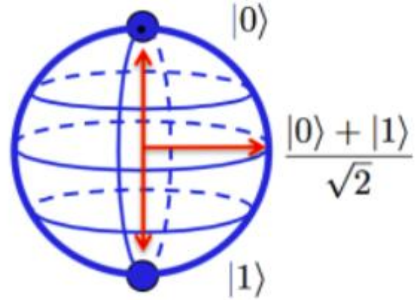
Quantum Bit (Qubit)

- 2 basic states — $|0\rangle$, $|1\rangle$ (ket 0, ket 1)
- Uses **superposition** of both states with “quantum" effect store information.
- Thus, it represents both $|0\rangle$ and $|1\rangle$ at the same time.

$$|\psi\rangle = |0\rangle \ \boldsymbol{and} \ |1\rangle$$

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \quad \rightarrow \begin{pmatrix} \alpha \\ \beta \end{pmatrix}$$

Bloch Sphere



Qubit

Multiple-Qubits System

2 Classical Bits

00 **or** 01 **or** 10 **or** 11

n bits for 1 value
$\boldsymbol{x \in [0, 2^n - 1]}$

2 Qubits

$c_{00}|00\rangle$ **and** $c_{01}|01\rangle$ **and** $c_{10}|10\rangle$ **and** $c_{11}|11\rangle$

n bits for 2^n values
$\boldsymbol{a_0, a_1, a_2, \cdots\ a_n}$

Qubits: $\boldsymbol{q_0, q_1}$

$$|q_0\rangle = a_0|0\rangle + a_1|1\rangle$$

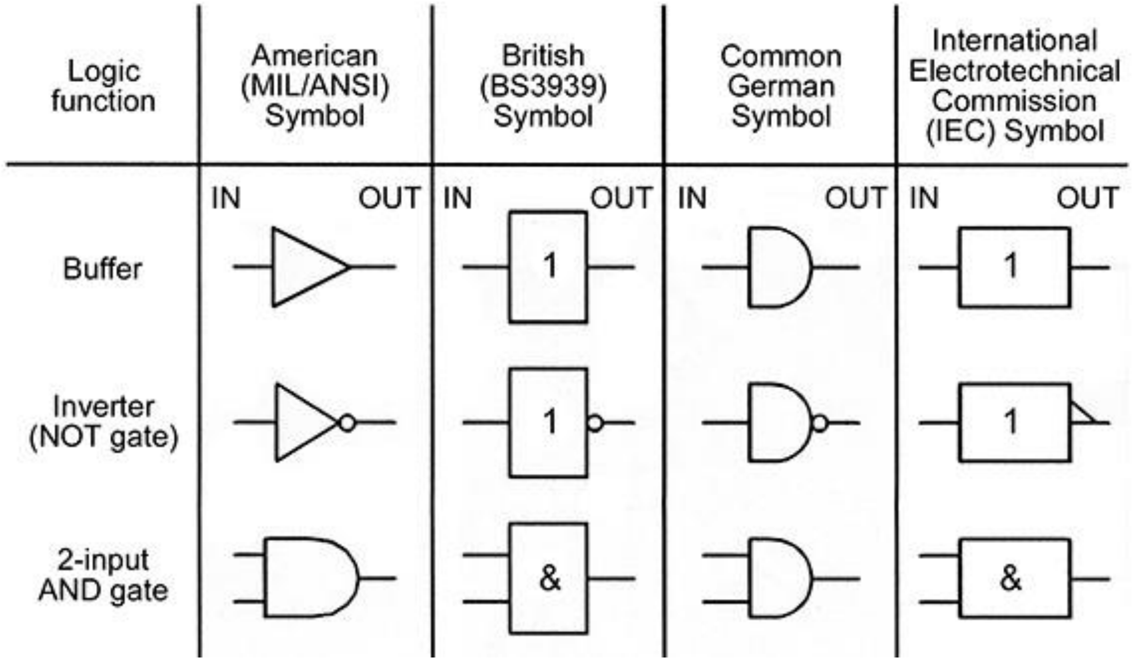
$$|q_1\rangle = b_0|0\rangle + b_1|1\rangle$$

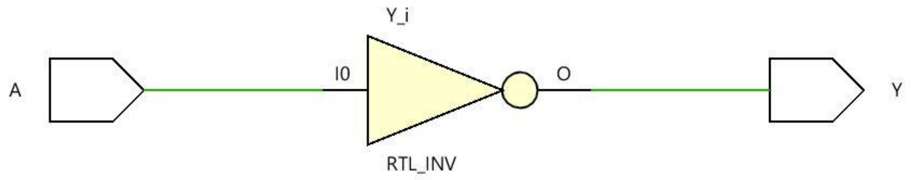
$$\boldsymbol{|q_0, q_1\rangle = |q_0\rangle \otimes |q_1\rangle}$$
$$\boldsymbol{= c_{00}|00\rangle + c_{01}|01\rangle + c_{10}|10\rangle + c_{11}|11\rangle}$$

$$\boldsymbol{|q_0, q_1\rangle = |q_0\rangle \otimes |q_1\rangle = \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} \otimes \begin{pmatrix} b_0 \\ b_1 \end{pmatrix}}$$

$$\boldsymbol{= \begin{pmatrix} a_0 \times \begin{pmatrix} b_0 \\ b_1 \end{pmatrix} \\ a_1 \times \begin{pmatrix} b_0 \\ b_1 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} a_0 b_0 \\ a_0 b_1 \\ a_1 b_0 \\ a_1 b_1 \end{pmatrix} = \begin{pmatrix} c_{00} \\ c_{01} \\ c_{10} \\ c_{11} \end{pmatrix}}$$

Computation: Logic Gates vs. Quantum Logic Gates

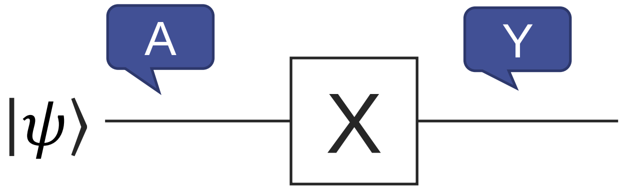




$$Y = \bar{A}$$

A	Y
0	1
1	0

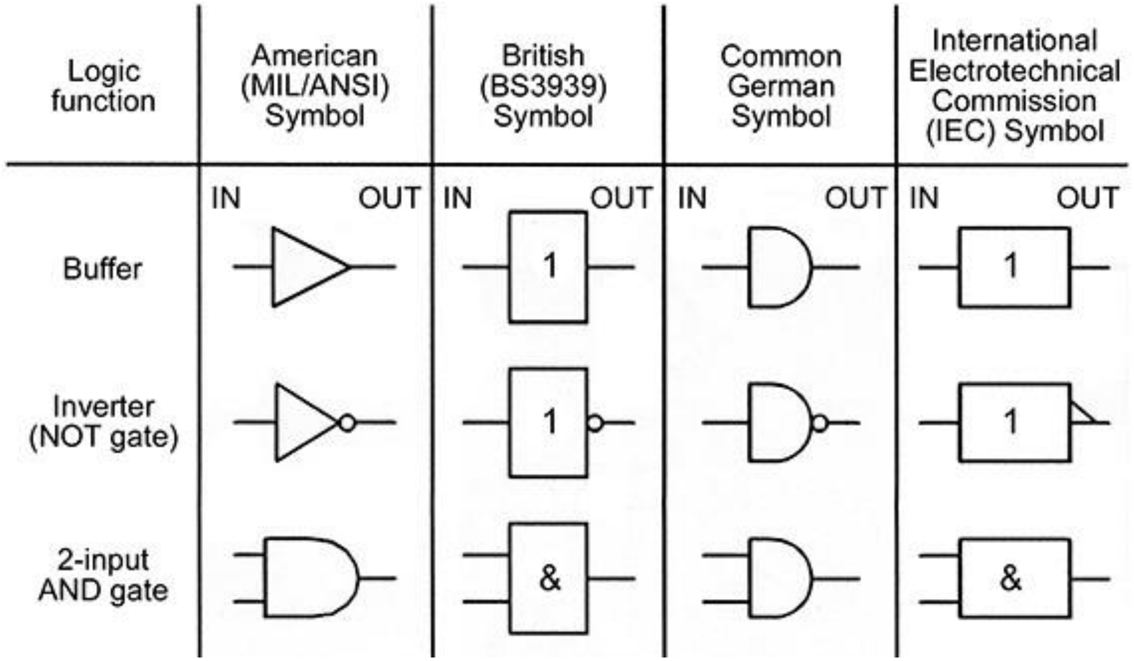
Operator	Gate(s)	Matrix
Pauli-X (X)	$\boxed{X}$ $\oplus$	$\begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}$
Pauli-Y (Y)	$\boxed{Y}$	$\begin{bmatrix}0 & -i\\i & 0\end{bmatrix}$
Pauli-Z (Z)	$\boxed{Z}$	$\begin{bmatrix}1 & 0\\0 & -1\end{bmatrix}$
Hadamard (H)	$\boxed{H}$	$\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\1 & -1\end{bmatrix}$
Phase (S, P)	$\boxed{S}$	$\begin{bmatrix}1 & 0\\0 & i\end{bmatrix}$

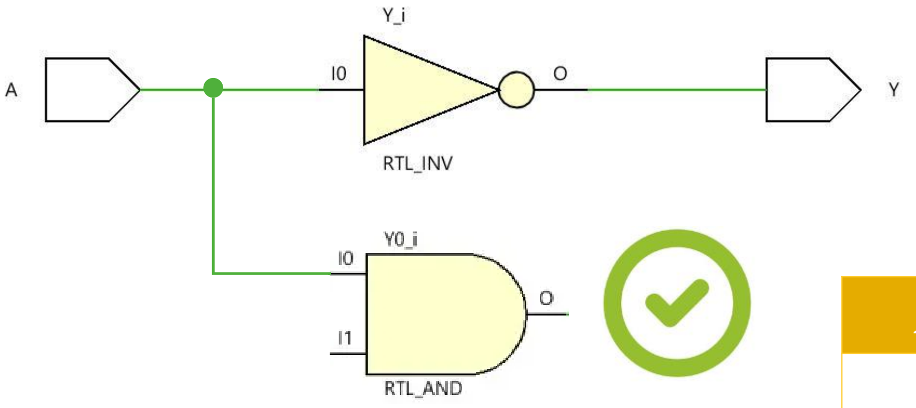


$$|Y\rangle = X \times |A\rangle$$

$$Y \begin{pmatrix} y_0 \\ y_1 \end{pmatrix} = \boxed{X} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \times A \begin{pmatrix} a_0 \\ a_1 \end{pmatrix}$$

Computation: Logic Gates vs. Quantum Logic Gates

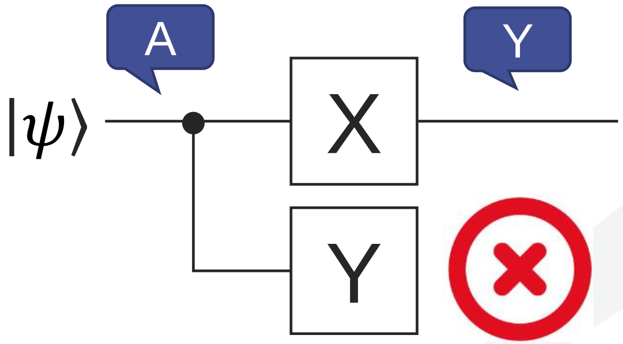




$$Y = \bar{A}$$

A	Y
0	1
1	0

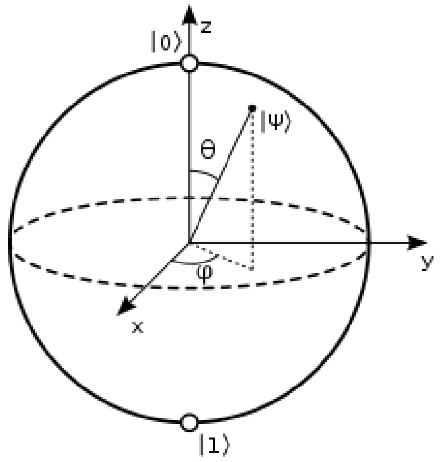
Operator	Gate(s)		Matrix
Pauli-X (X)	$\boxed{X}$	$\oplus$	$\begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}$
Pauli-Y (Y)	$\boxed{Y}$		$\begin{bmatrix}0 & -i\\i & 0\end{bmatrix}$
Pauli-Z (Z)	$\boxed{Z}$		$\begin{bmatrix}1 & 0\\0 & -1\end{bmatrix}$
Hadamard (H)	$\boxed{H}$		$\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\1 & -1\end{bmatrix}$
Phase (S, P)	$\boxed{S}$		$\begin{bmatrix}1 & 0\\0 & i\end{bmatrix}$



$$|Y\rangle = X \times |A\rangle$$

$$\text{Y}\begin{pmatrix}y_0\\y_1\end{pmatrix} = \boxed{X}\begin{pmatrix}0 & 1\\1 & 0\end{pmatrix} \times \text{A}\begin{pmatrix}a_0\\a_1\end{pmatrix}$$

General U Gates



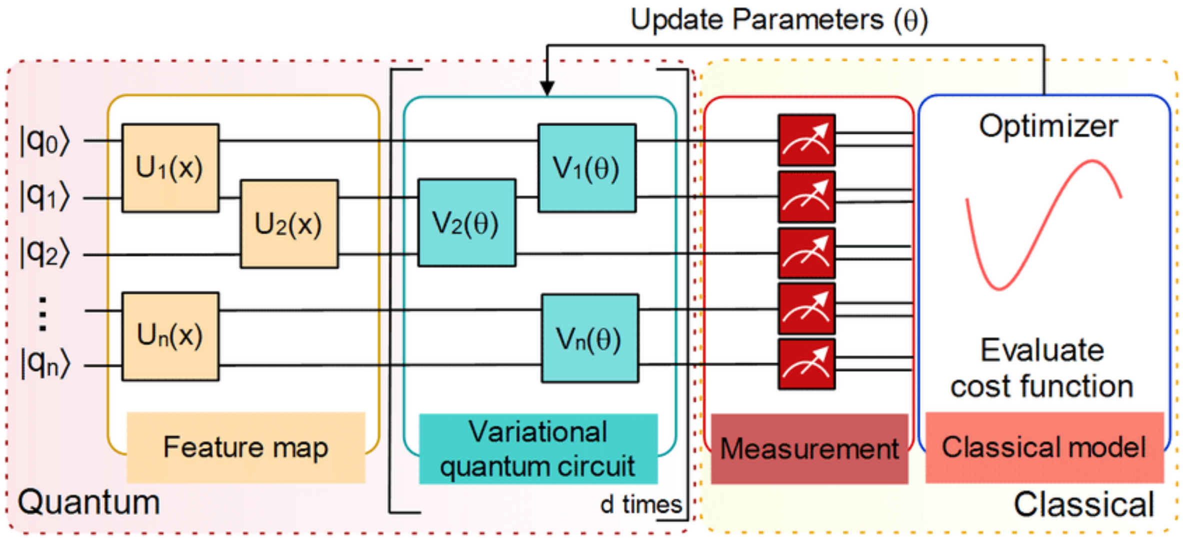
Single-Qubit Gates

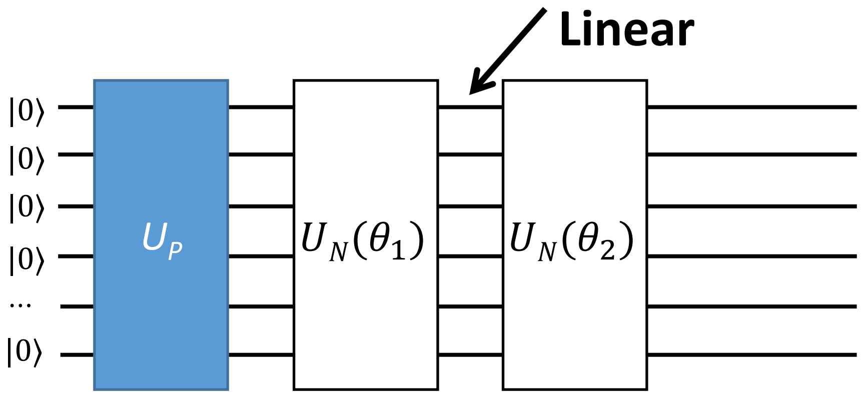
- Pauli operators: X, Y, Z Gates
- Hadamard gate: H Gate
- **General gate: U Gate**

$$\mathbf{U_3}(\theta,\phi,\lambda) = \begin{pmatrix} \cos(\theta/2) & -e^{i\lambda}\sin(\theta/2) \\ e^{i\phi}\sin(\theta/2) & e^{i(\lambda+\phi)}cos(\theta/2) \end{pmatrix}$$

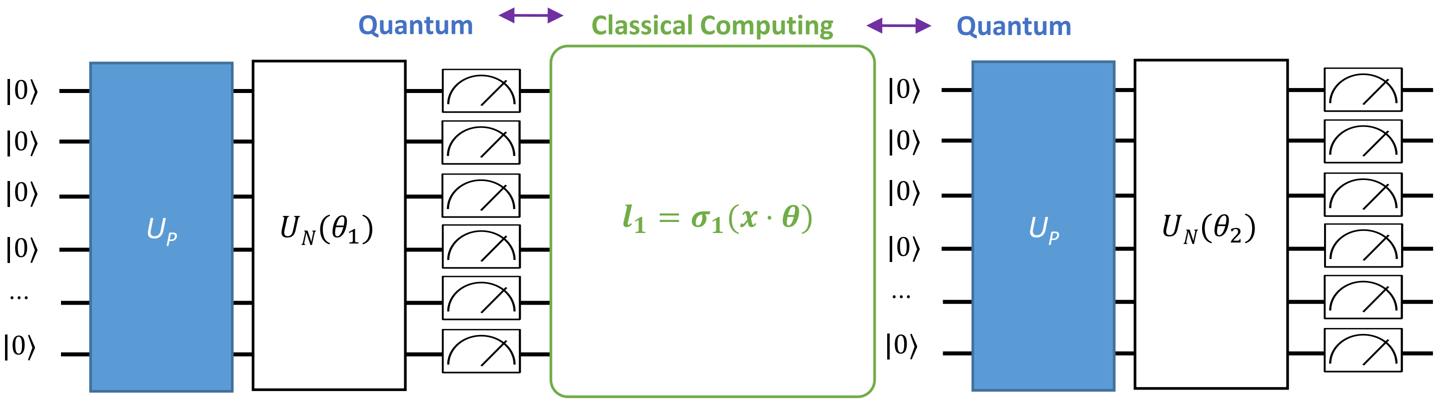
$$\mathbf{U_3}\,|\mathbf{0}\rangle = \begin{pmatrix} \cos(\theta/2) \\ e^{i\phi}\sin(\theta/2) \end{pmatrix} = \cos(\theta/\mathbf{2})\,|\mathbf{0}\rangle \;+\; \mathbf{e}^{\mathbf{i}\phi}\sin(\theta/\mathbf{2})\,|\mathbf{1}\rangle$$

Two Paths of Quantum Machine Learning: Path 1 --- VQC





[ref] Sen, P., Bhatia, A.S., Bhangu, K.S. and Elbeltagi, A., 2022. Variational quantum classifiers through the lens of the Hessian. Plos one, 17(1), p.e0262346.



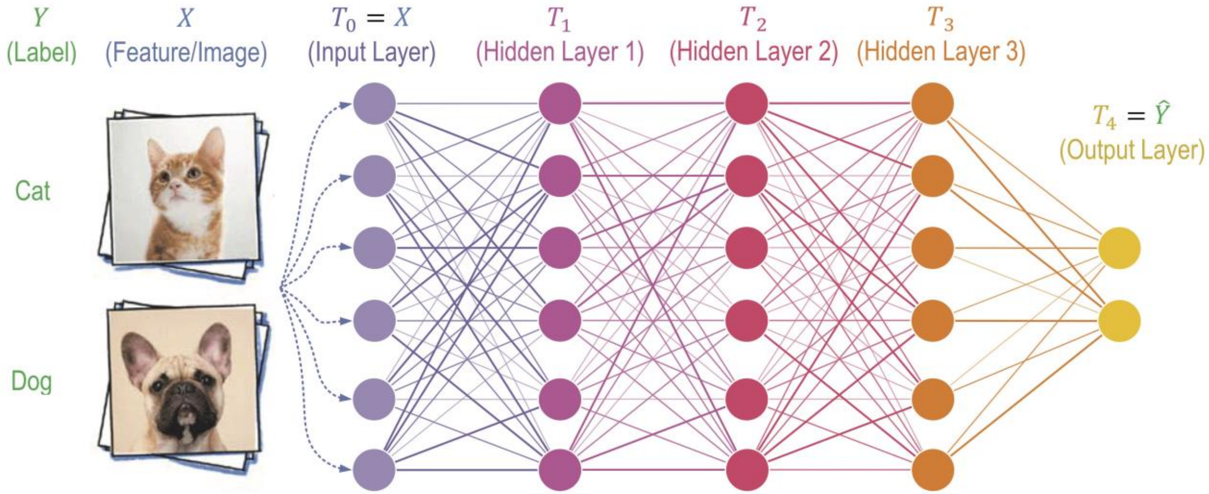
Pros:

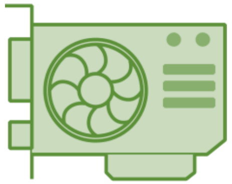
- Easy to **implement**

Cons:

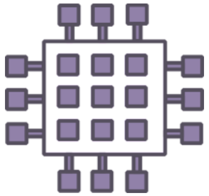
- On intermediate-scale quantum devices, no works show that **performance** of QML can beat classical ML, *so far*.
- Have no **non-linear** in the network
- Incur **heavy overhead** for **non-linearity**

Two Paths of Quantum Machine Learning: Path 2 --- Q Accelerator

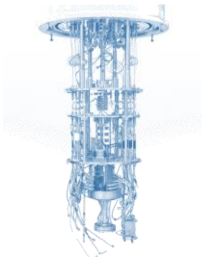




CPU/GPU Accelerator



FPGA/ASIC Accelerator



Quantum Accelerator

Pros:

- Same Performance as Classical ML

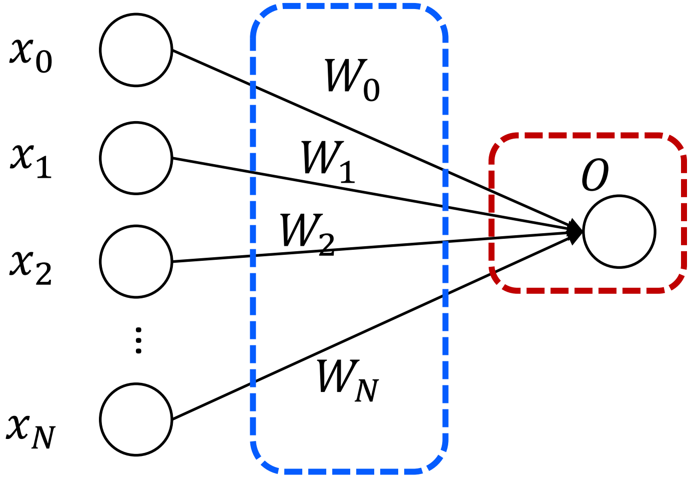
Questions:

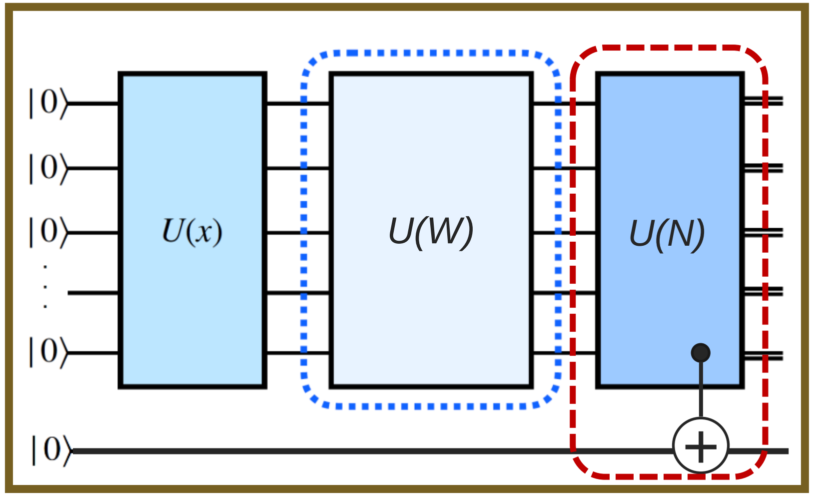
- How to design?
- Advantage?

QuantumFlow Answered Two Fundamental Questions

Fundamental questions:

- Can we use quantum gates to **correctly implement** neural functions?





- How to design quantum circuit to achieve **quantum advantage**?

$$O = \delta\left(\sum_{i\in[0,N)} x_i \times W_i\right)$$

where δ is a quadratic function

Classical Computing:

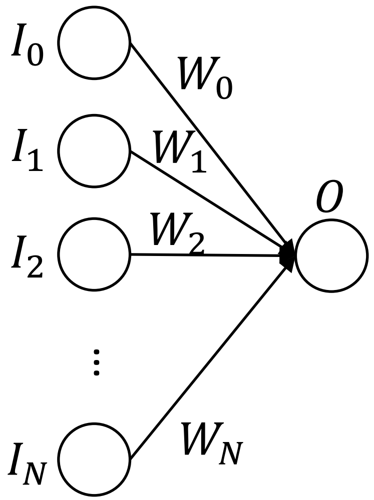
Complexity of $\boldsymbol{O(N)}$

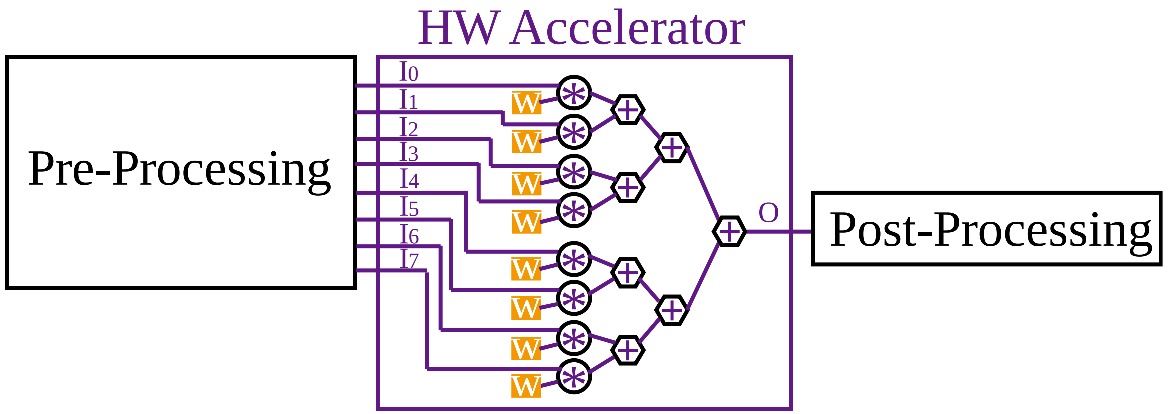
Quantum Computing:

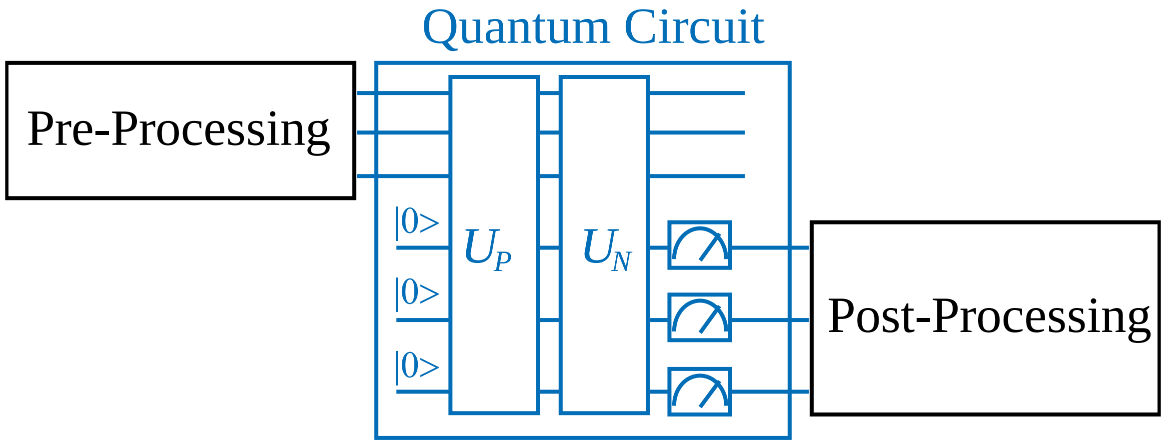
Can we reduce complexity to

$\boldsymbol{O(ploylogN)}$, say $\boldsymbol{O(log^2N)}$?

Neural Network Accelerator Design from Classical to Quantum Computing







(1) Data Pre-Processing (*PreP*)

(2) HW Acceleration

(3) Data Post-Processing (*PostP*)

(1) Data Pre-Processing (*PreP*)

(2) HW/Quantum Acceleration

(2.1) U_p Quantum-State-Preparation

(2.2) U_N Quantum Neural Computation

(2.3) *M* Measurement

(3) Data Post-Processing (PostP)

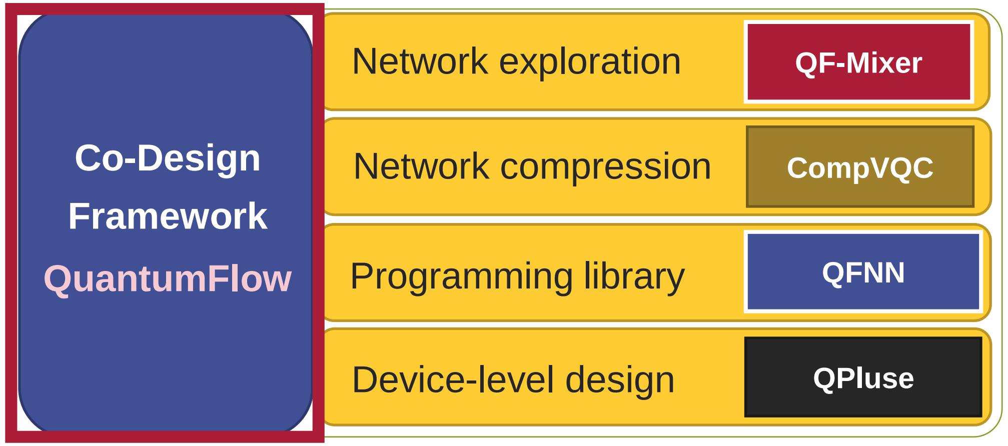
$$PreP + U_P + U_N + M + PostP$$

Hands-On: QuantumFlow

A Co-Design Framework of Neural Networks and Quantum Circuits Towards Quantum Advantage

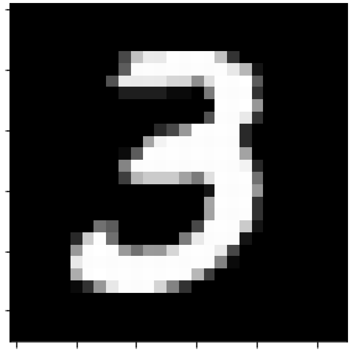
Published at Nature Communications 2021

Presenter: Zhepeng Wang



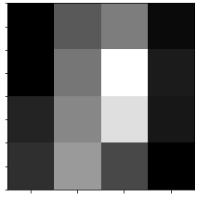
PreP + U_P + U_N + *M* + *PostP* : Data Pre-Processing

- **Given:** (1) 28×28 image, (2) the number of qubits to encode data (say Q=4 qubits in the example)
- **Do:** (1) downsampling from 28×28 to $2^Q = 16 = 4 \times 4$; (2) converting data to be the state vector in a unitary matrix
- **Output:** A unitary matrix, $M_{16\times16}$



Step 1: Downsampling

From 28×28 to 4×4



$$\begin{bmatrix} 0.0039 & 0.2118 & 0.2941 & 0.0275 \\ 0.0039 & 0.2784 & 0.5961 & 0.0667 \\ 0.0863 & 0.3176 & 0.5216 & 0.0588 \\ 0.1137 & 0.3608 & 0.1725 & 0.0039 \end{bmatrix}$$

$$\begin{bmatrix} 0.0039 & 0.2118 & 0.2941 & 0.0275 \\ 0.0039 & 0.2784 & 0.5961 & 0.0667 \\ 0.0863 & 0.3176 & 0.5216 & 0.0588 \\ 0.1137 & 0.3608 & 0.1725 & 0.0039 \end{bmatrix}$$

Step 2: Formulate Unitary Matrix

Applying SVD method

(See Listing 1 in ASP-DAC SS Paper)

Unitary matrix: $M_{16\times16}$

[SS] W. Jiang, et al. When Machine Learning Meets Quantum Computers: A Case Study, ASP-DAC'21

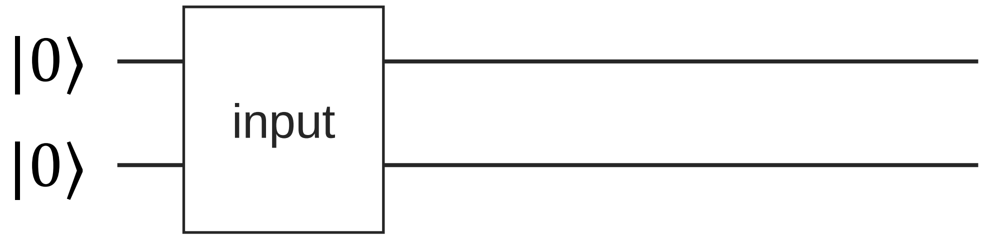
PreP + ***U_P*** + *U_N* + *M* + *PostP* --- Data Encoding / Quantum State Preparation

- **Given:** The unitary matrix provided by *PreP*, $M_{16\times16}$
- **Do:** Quantum-State-Preparation, encoding data to qubits
- **Verification:** Check the amplitude of states are consistent with the data in the unitary matrix, $M_{16\times16}$

Let's use a 2-qubit system as an example to encode a matrix $M_{4\times4}$

data_matrix

$$\begin{bmatrix} 0.3 & 0.5 \\ 0.7 & 0.9 \end{bmatrix} \xrightarrow{PreP} \begin{bmatrix} \mathbf{0.2343} & X & X & X \\ \mathbf{0.3904} & X & X & X \\ \mathbf{0.5466} & X & X & X \\ \mathbf{0.7028} & X & X & X \end{bmatrix} \xrightarrow{U_P}$$



State Transition:

data_matrix $|00\rangle$

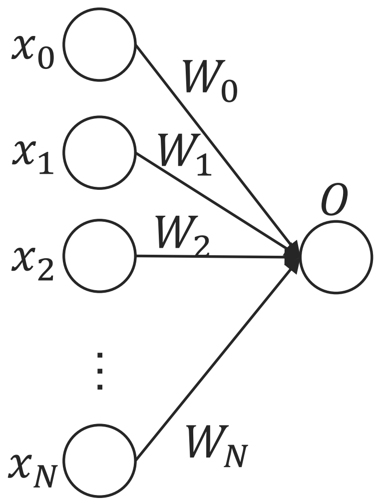
IBM Qiskit Implementation:

```
inp = QuantumRegister(4, ”in_qubit”)
circ = QuantumCircuit(inp)
iniG = UnitaryGate(data_matrix, label=”input”)
circ.append(iniG, inp[0:4])
```

Hands-On Tutorial (1)

PreP + U_P

PreP + *U_P* + ***U_N*** + *M* + *PostP* --- Neural Computation



- **Given:** (1) A circuit with encoded input data $\boldsymbol{x}$; (2) the trained binary weights $\boldsymbol{w}$ for one neural computation, which will be associated to each data.
- **Do:** Place quantum gates on the qubits, such that it performs $\frac{(\boldsymbol{x}*\boldsymbol{w})^2}{\|\boldsymbol{x}\|}$.
- **Verification:** Whether the output data of quantum circuit and the output computed using torch on classical computer are the same.

Target: $O = \left[\frac{\sum_i (x_i \times w_i)}{\sqrt{\|x\|}}\right]^2$

- **Assumption 1:** Parameters/weights (W_0 --- W_N) are binary weight, either +1 or -1
- **Assumption 2:** The weight $W_0 = +1$, otherwise we can use $-\boldsymbol{w}$ (quadratic func.)

Step 1: $m_i = x_i \times w_i$

Step 2: $n = \left[\frac{\sum_i (m_i)}{\sqrt{\|x\|}}\right]$

Step 3: $O = n^2$

Quantum Neuron Design: Step 1

Step 1: $m_i = x_i \times w_i$

EX: 4 input data on 2 qubits

$$\boldsymbol{x} = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} \qquad \boldsymbol{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \end{bmatrix} \qquad \begin{aligned} w_0 &= 1 \\ w_1 &= 1 \\ w_2 &= 1 \\ w_3 &= -1 \end{aligned}$$

$$\Rightarrow \boxed{m_3 = -1 \times a_3 = -a_3}$$

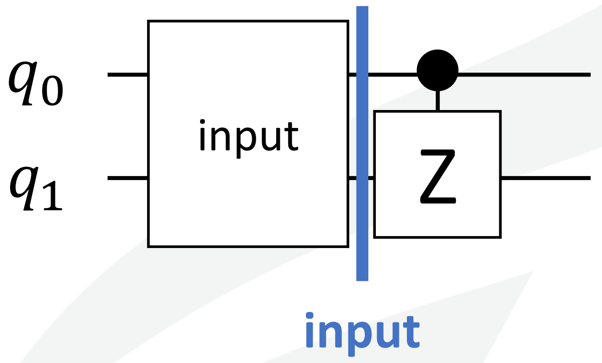
Output = **U** × **Input**

a_0	$\|00\rangle$
a_1	$\|01\rangle$
a_2	$\|10\rangle$
$m_3 = -a_3$	$\|11\rangle$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} \times$$

a_0	$\|00\rangle$
a_1	$\|01\rangle$
a_2	$\|10\rangle$
a_3	$\|11\rangle$

Quantum Circuit



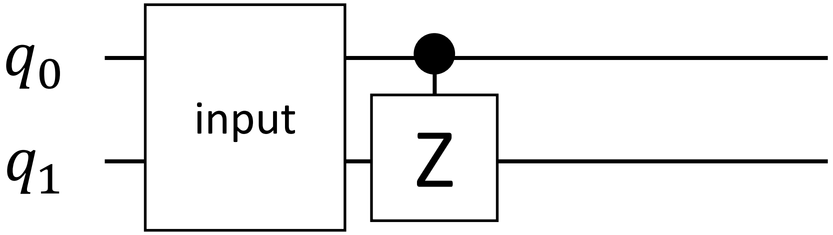
Quantum Neuron Design: Step 1

Step 1: $m_i = x_i \times w_i$

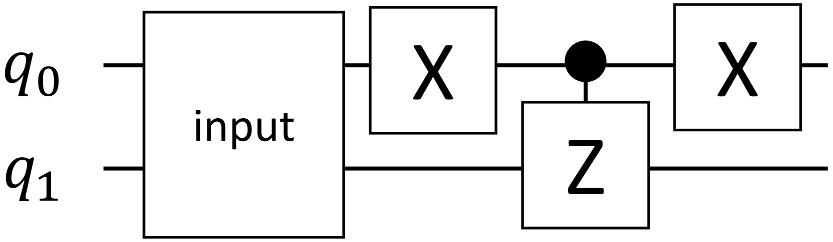
EX: 4 input data on 2 qubits

$$\boldsymbol{w} = \begin{bmatrix} +1 \\ +1 \\ +1 \\ -1 \end{bmatrix} \text{or} \begin{bmatrix} +1 \\ +1 \\ -1 \\ -1 \end{bmatrix} \text{or} \begin{bmatrix} +1 \\ -1 \\ -1 \\ -1 \end{bmatrix} \text{or} \begin{bmatrix} +1 \\ +1 \\ -1 \\ +1 \end{bmatrix} \text{or} \begin{bmatrix} +1 \\ -1 \\ -1 \\ +1 \end{bmatrix} \text{or} \begin{bmatrix} +1 \\ -1 \\ +1 \\ +1 \end{bmatrix}$$

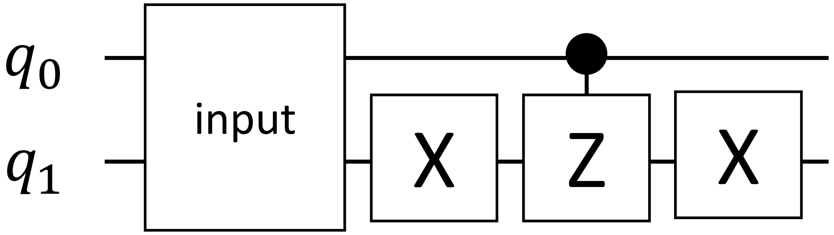
$$\boldsymbol{w} = \begin{bmatrix} +1 \\ +1 \\ +1 \\ -1 \end{bmatrix}$$

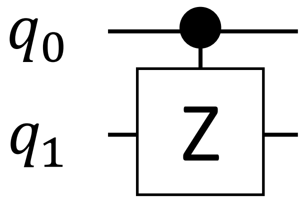


$$\boldsymbol{w} = \begin{bmatrix} +1 \\ -1 \\ +1 \\ +1 \end{bmatrix}$$

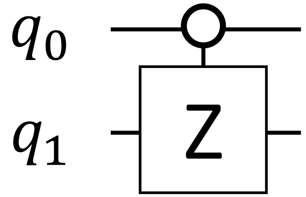


$$\boldsymbol{w} = \begin{bmatrix} +1 \\ +1 \\ -1 \\ +1 \end{bmatrix}$$

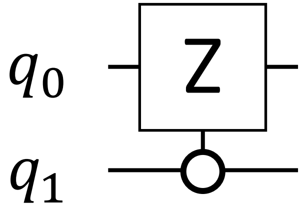




Flip the sign of $|11\rangle$



Flip the sign of $|01\rangle$



Flip the sign of $|10\rangle$

Quantum Neuron Design: Step 2

Step 2: $n = \left\lceil \frac{\sum_i (m_i)}{\sqrt{\|x\|}} \right\rceil$

EX: 4 input data on 2 qubits

Output $=$ **U** $\times$ **Input**

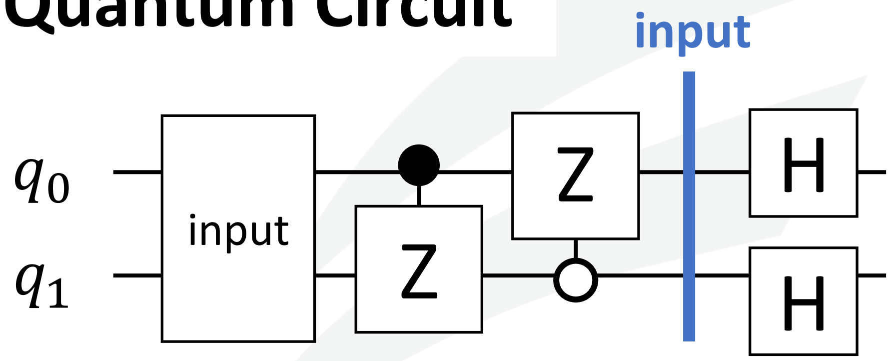
$\sum_i (m_i) / \sqrt{\|x\|}$	$\|00\rangle$
Do not care 1	$\|01\rangle$
Do not care 2	$\|10\rangle$
Do not care 3	$\|11\rangle$

$$= \frac{1}{\sqrt{\|x\|}} \begin{bmatrix} 1 & 1 & 1 & 1 \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{bmatrix} \times$$

m_0	$\|00\rangle$
m_1	$\|01\rangle$
m_2	$\|10\rangle$
m_3	$\|11\rangle$

note: $\|x\| = 2^N$

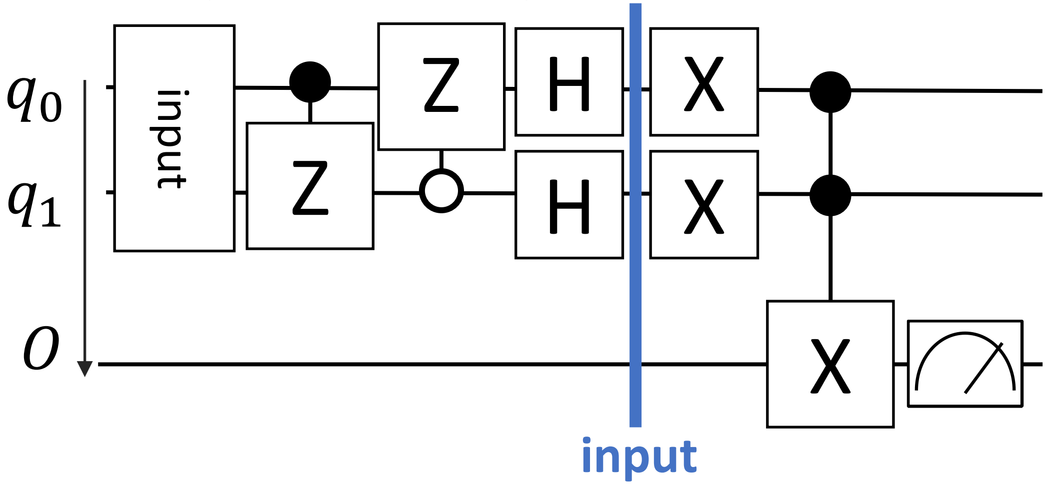
Quantum Circuit



Quantum Neuron Design: Step 3

Step 3: $O = n^2$

EX: 4 input data on 2 qubits



Input

$\sum_i (m_i) / \sqrt{\|x\|}$	$\|000\rangle$
0	$\|001\rangle$
Do not care 1	$\|010\rangle$
0	$\|011\rangle$
Do not care 2	$\|100\rangle$
0	$\|101\rangle$
Do not care 3	$\|110\rangle$
0	$\|111\rangle$

$X^{\otimes 2}$

Do not care 3	$\|000\rangle$
0	$\|001\rangle$
Do not care 2	$\|010\rangle$
0	$\|011\rangle$
Do not care 1	$\|100\rangle$
0	$\|101\rangle$
$\sum_i (m_i) / \sqrt{\|x\|}$	$\|110\rangle$
0	$\|111\rangle$

CCX

Do not care	$\|000\rangle$
0	$\|001\rangle$
Do not care	$\|010\rangle$
0	$\|011\rangle$
Do not care	$\|100\rangle$
0	$\|101\rangle$
0	$\|110\rangle$
$\sum_i (m_i) / \sqrt{\|x\|}$	$\|111\rangle$

Output

$$P\{O = |1\rangle\} = P\{|001\rangle\} + P\{|011\rangle\} + P\{|101\rangle\} + P\{|111\rangle\} = \left[\frac{\sum_i (m_i)}{\sqrt{\|x\|}}\right]^2$$

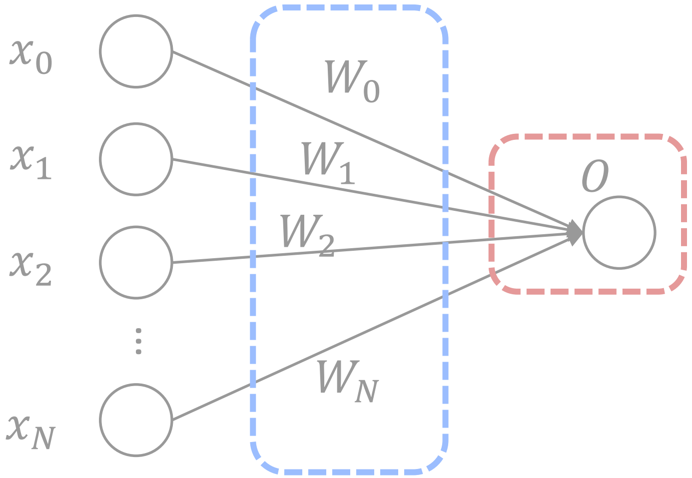
Hands-On Tutorial (2)

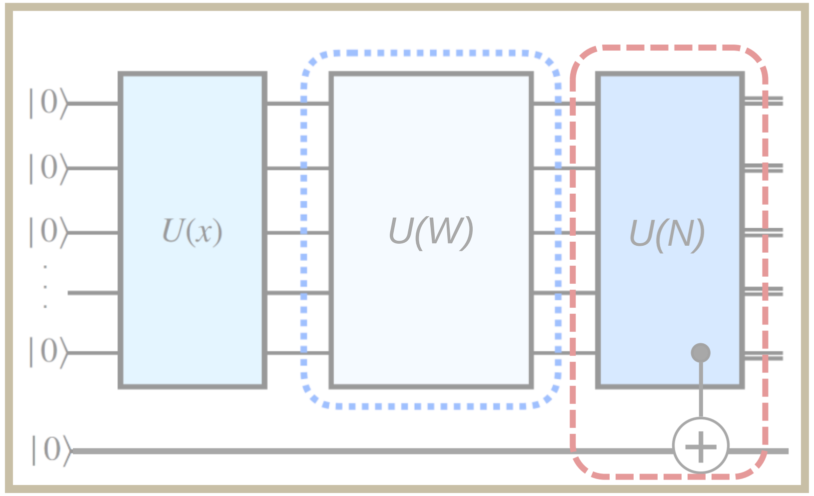
$PreP + U_P + U_N$

QuantumFlow Answered Two Fundamental Questions

Fundamental questions:

- Can we use quantum gates to **correctly implement** neural functions?





- How to design quantum circuit to achieve **quantum advantage**?

$$O = \delta\left(\sum_{i\in[0,N)} x_i \times W_i\right)$$

where δ is a quadratic function

Classical Computing:

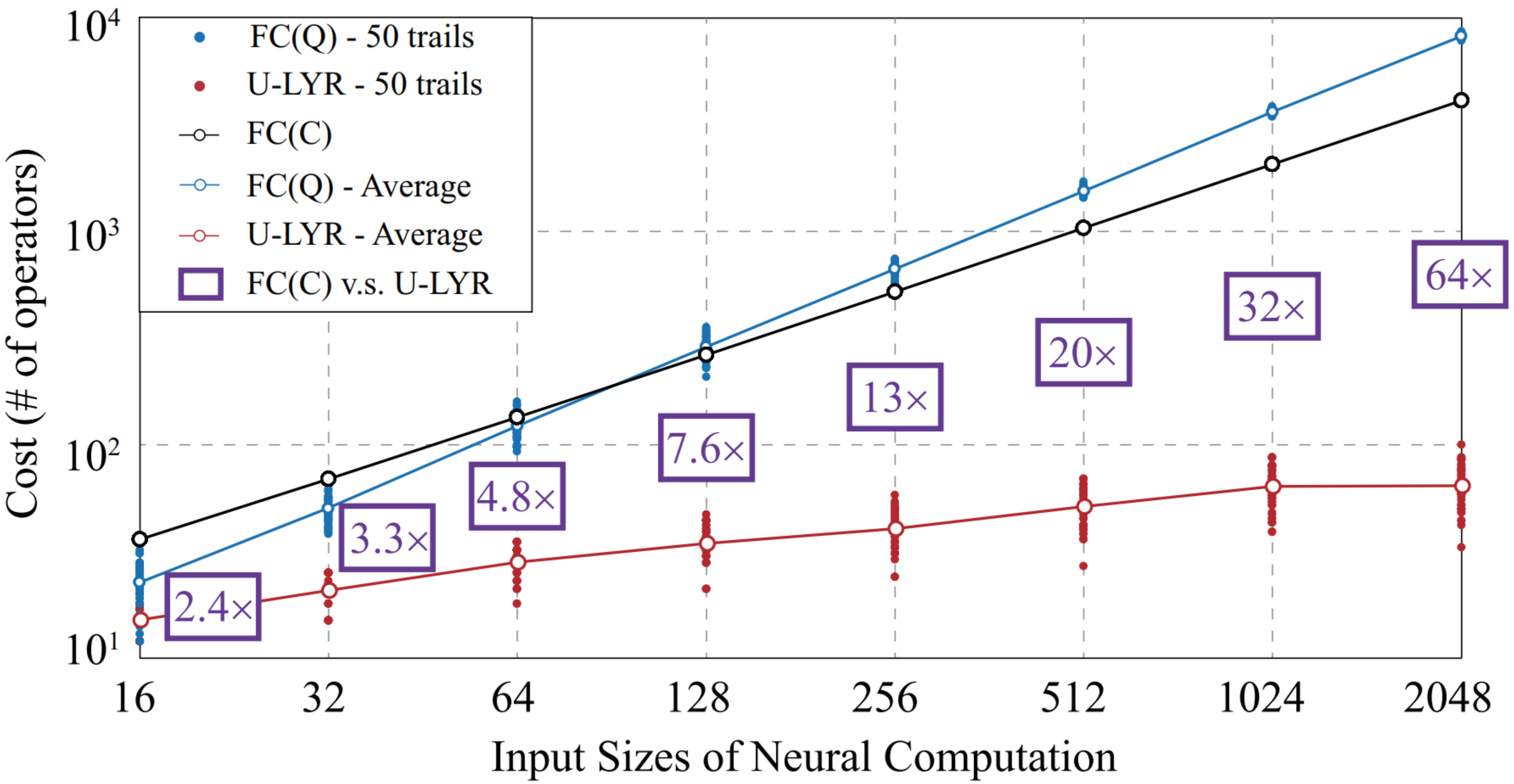
Complexity of $\boldsymbol{O(N)}$

Quantum Computing:

Can we reduce complexity to

$\boldsymbol{O(ploylogN)}$, say $\boldsymbol{O(log^2N)}$?

QuantumFlow Results



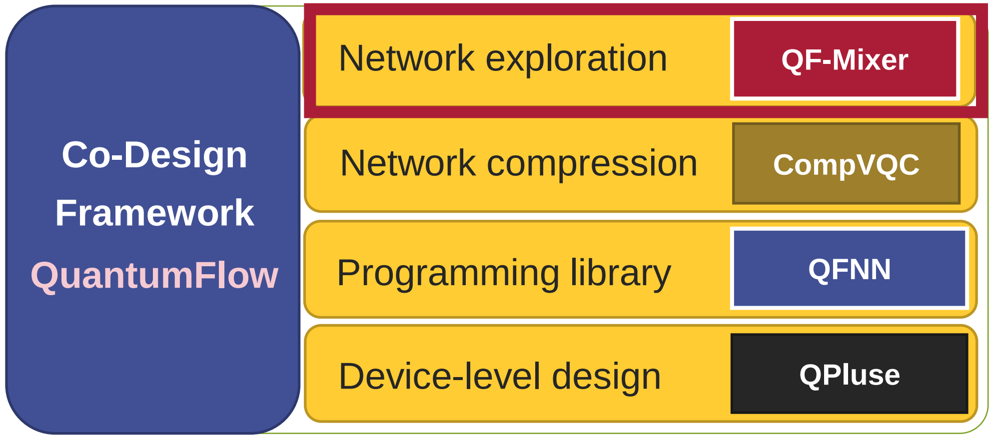
[ref] Tacchino, F., et al., 2019. An artificial neuron implemented on an actual quantum processor. *npj Quantum Information*, *5*(1), pp.1-8.

Quantum Neural Network: QF-Mixer

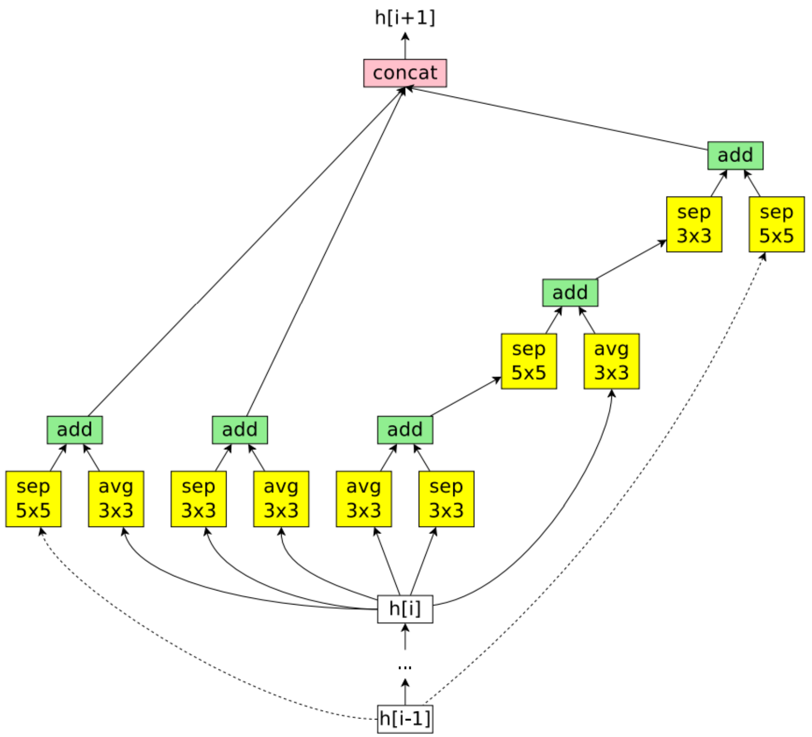
Exploration of Quantum Neural Architecture by Mixing Quantum Neuron Designs

Published at IEEE/ACM International Conference on Computer-Aided Design 2021

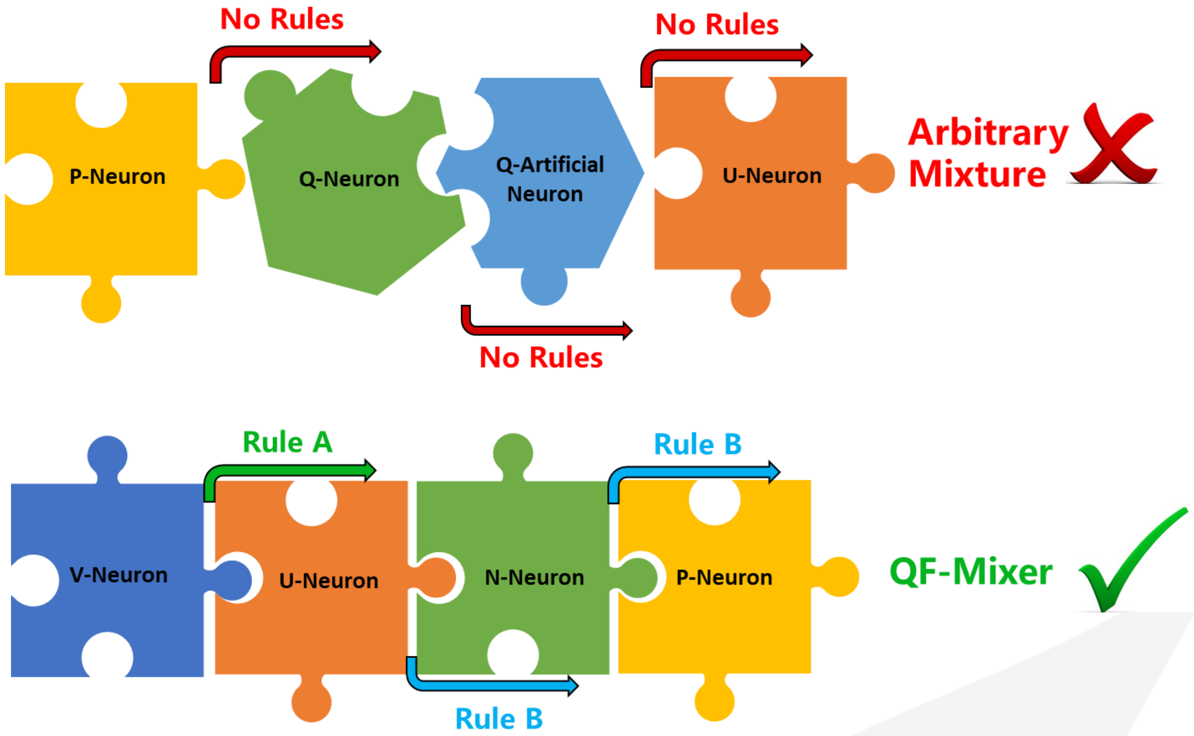
Presenter: Zhepeng Wang



Challenges



Reduction Cell



Different operators/neurons in classical computing can be connected seamlessly.

Connect different quantum neurons may incur high overhead; will not be seamless.

QF-MixNN

- VQC and QuantumFlow are complementary to each other and can be mixed.

VQC

Advantage

- Real-valued weights

Disadvantage

- Linear classifier
- Cannot be extended to multiple nonlinear layers with low cost

Mixing

1+1 > 2

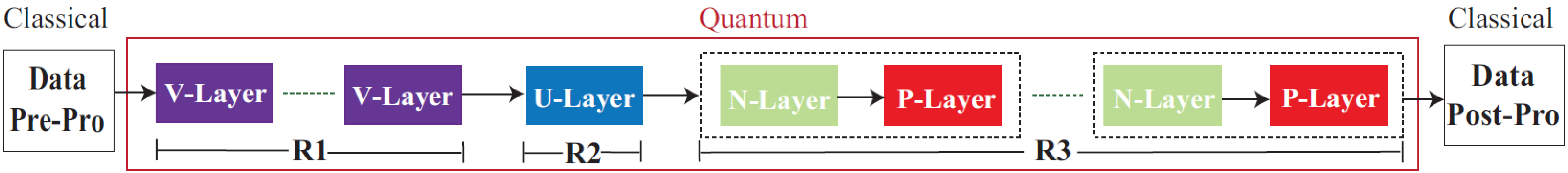
QuantumFlow

Disadvantage

- Binary weights

Advantage

- Easy to be extended to multiple nonlinear layers w/t measurement



QF-MixNN Achieves the Best Accuracy on MNIST

TABLE I
EVALUATION OF QNNS WITH DIFFERENT NEURAL ARCHITECTURE

Architecture		MNIST-2†	MNIST-3†	MNIST-4‡	MNIST-5‡	MNIST§
VQC (V×R1)		**97.91%**	90.09%	93.45%	91.35%	52.77%
QuantumFlow		95.63%	91.42%	94.26%	89.53%	69.92%
	V+U	97.36%	**92.77%**	**94.41%**	**93.85%**	88.46%
QF-MixNN	V+U+P	87.45%	82.9%	92.44%	91.56%	**90.62%**
	V+P	91.72%	76.93%	88.43%	85.02%	49.57%

Input resolutions: † 4×4; ‡ 8×8; § 16×16;

- **Non-linearity is important**. A linear decision boundary is not sufficient for complicated tasks.
- **Real-valued weight is helpful**. It increases the representation capability of QNN significantly.

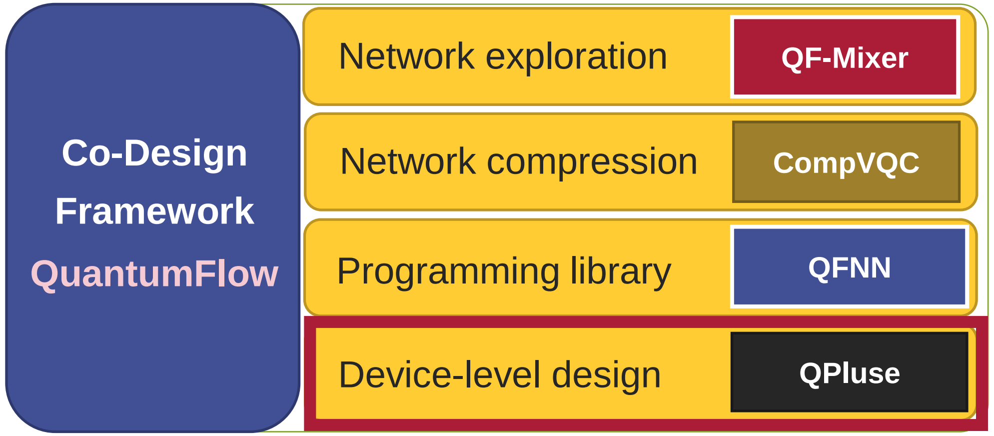
- Achieve **highest accuracy** for full set of MNIST dataset
- **QF-MixNN** takes the advantage of both **VQC-based QNN** and **QF-Net** from Quantumflow.

Quantum Pulse: VQP

Variational Quantum Pulse Learning

Published at IEEE Quantum Week 2022

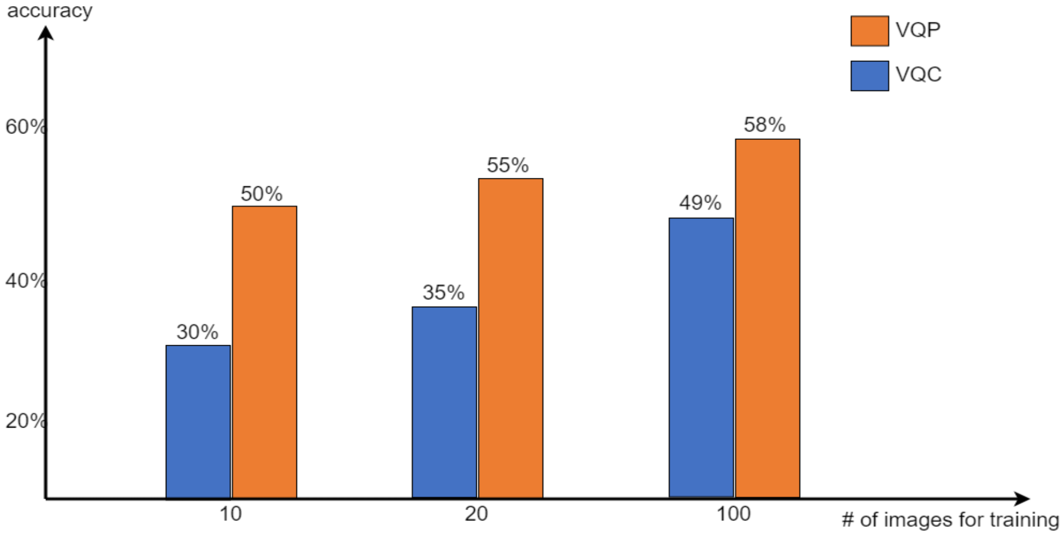
Presenter: Zhiding Liang



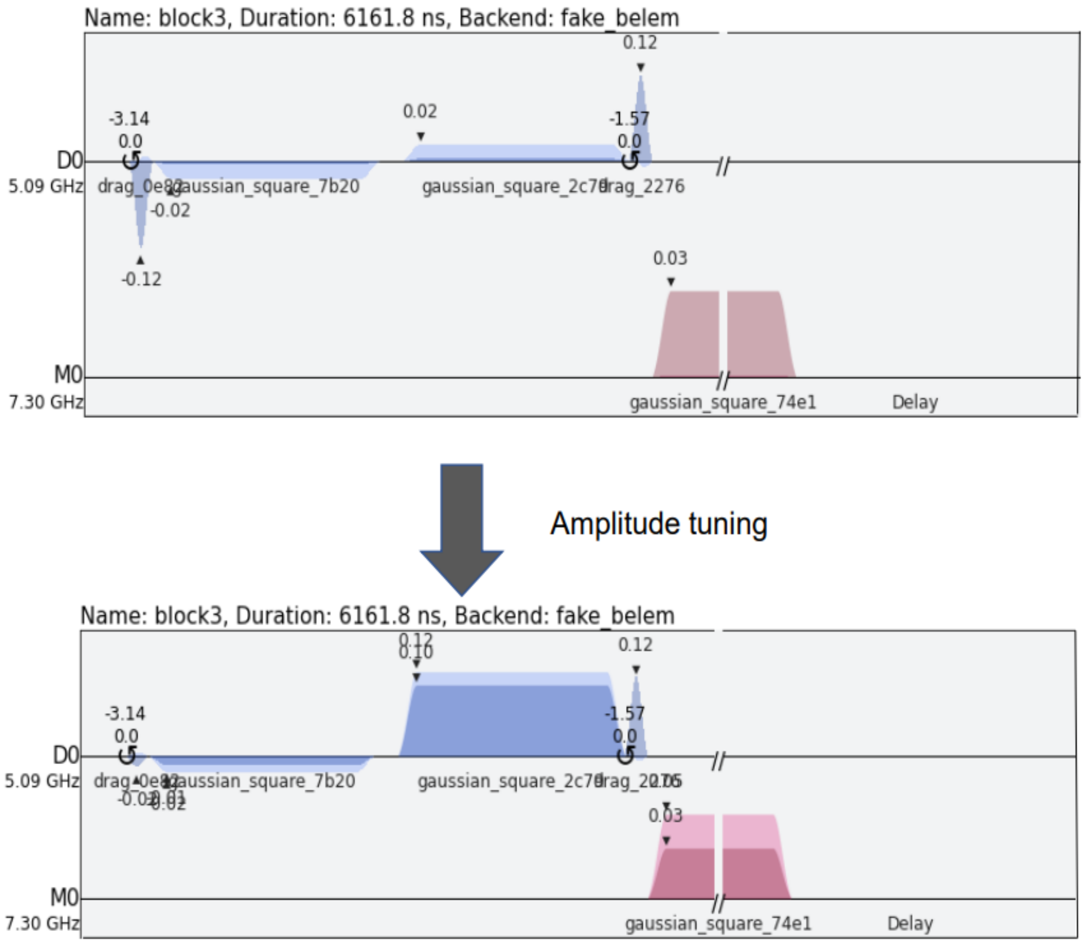
Why pulse learning?

UNIVERSITY OF NOTRE DAME

- Variational quantum circuit (VQC) shows the potential on ML tasks on explore search space due to the property.
- Compared to the VQC, VQP has more parameters that learnable.
- Compared to the VQC, VQP avoid partial of noise from decoherence error.
- Compared to the VQC, VQP directly change the physical parameters on physical pulses. Thus, gain the flexible on the control.



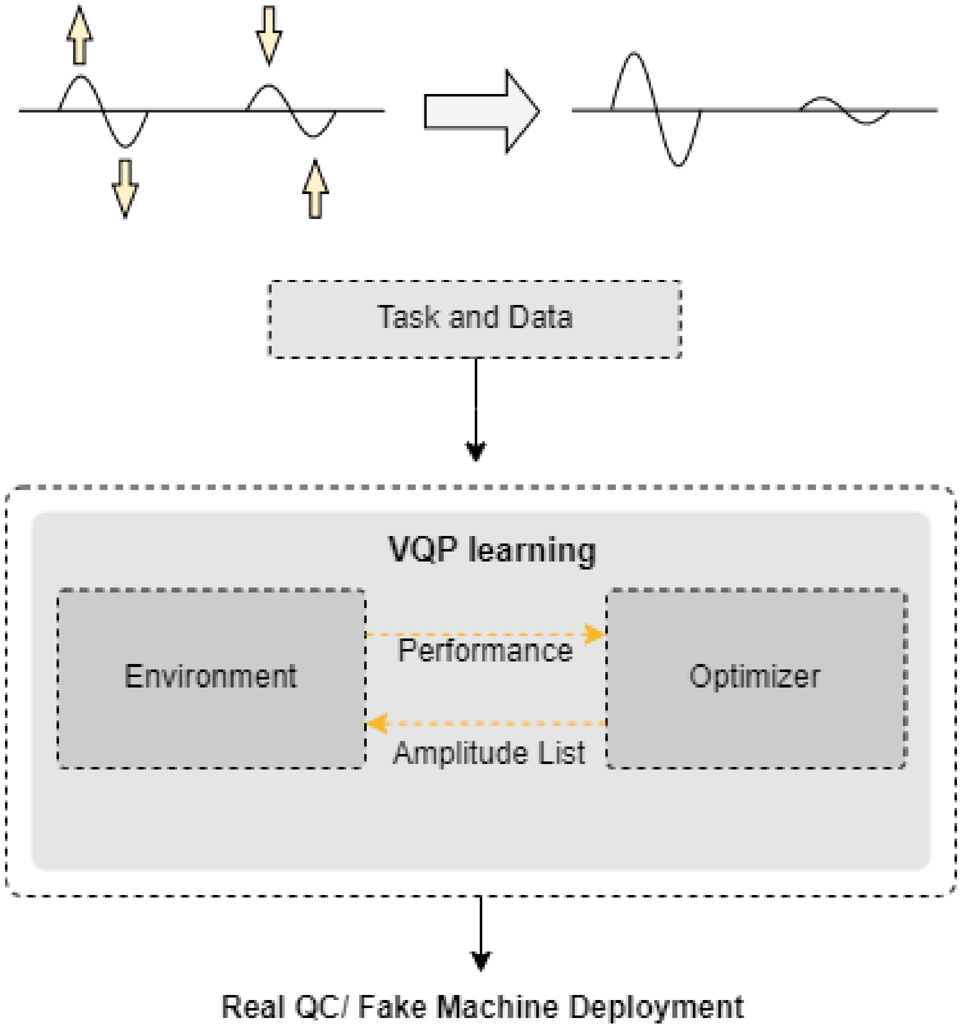
Why pulse learning?

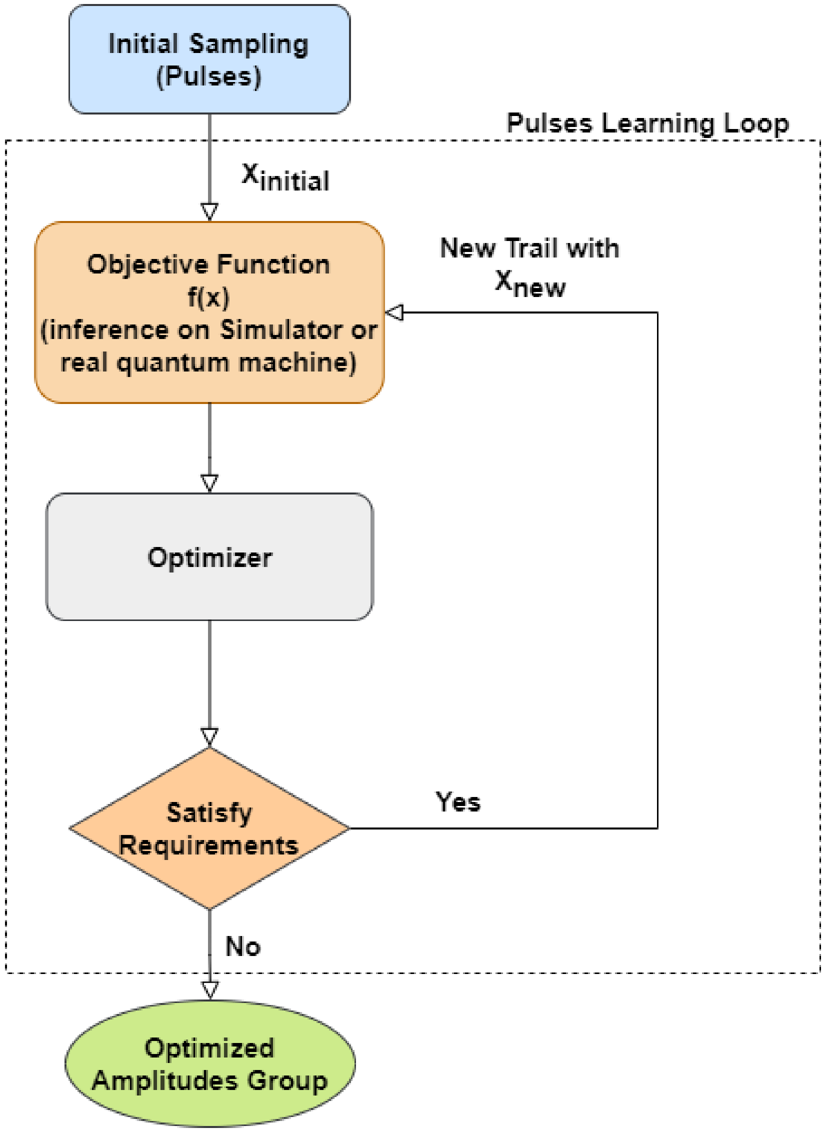


$$
\begin{aligned}
H = &\sum_{i=0}^{1}(U_i(t)+D_i(t))\sigma_i^X + \sum_{i=0}^{1} 2\pi\nu_i(1-\sigma_i^Z)/2 \\
&+\omega_B a_B a_B^\dagger + \sum_{i=0}^{1} g_i\sigma_i^X(a_B+a_B^\dagger)
\end{aligned}
\qquad (1)
$$

$$
\begin{aligned}
D_i(t) &= Re(d_i(t)e^{iw_{d_i}t}) \\
U_i(t) &= Re[u_i(t)e^{i(w_{d_i}-w_{d_j})t}]
\end{aligned}
\qquad (2)
$$

Optimization Framework?





Why pulse learning?

Form of CX gate	Time Duration Noise simulator (Quito)	Noise simulator (Belem)	Noise simulator (Jakarta)
CRX(π) gate	26832.0dt	32016.0dt	26832.0dt
CX gate	25136.0dt	27728.0dt	25136.0dt

Advantage in specific gate

Model	# of Gate	Time Duration ibmq_jakarta	Noise simulator (Belem)
VQP	**9**	**40816.0dt**	**45168.0dt**
VQC*	12	58896.0dt	58768.0dt
VQP_transpiled	**11**	**32368.0dt**	**32816.0dt**
VQC*_transpiled	17	53008.0dt	46192.0dt

Advantage in general circuit

Experiment Result

UNIVERSITY OF NOTRE DAME

Model	Accuracy Noise simulator (Belem)	ibmq_jakarta
VQC learning 20	0.57	0.58
VQP learning 20	**0.6**	**0.69**
VQC learning 100	0.61	0.59
VQP learning 100	**0.63**	**0.64**
VQC learning MNIST 20	0.6	0.56
VQP learning MNIST 20	**0.66**	**0.62**
VQC learning MNIST 100	0.57	0.62
VQP learning MNIST 100	**0.61**	**0.71**

Achieves higher accuracy under same condition

Model	# of Gates	Accuracy
VQC_base	9	0.62
VQP	**9**	**0.71**
VQC*	12	0.68

VQC with more gates has similar performance in terms of accuracy

Challenge for Pulse Learning

1. Non-gradient-based optimizer has randomness when parameter in high dimensional space.

2. Qiskit pulse simulator is not efficient, e.g., need around 3 mins to finish a 9-gate circuit.

Model	# of Gates	Accuracy
VQC_base	9	0.62
VQP	**9**	**0.71**
VQC*	12	0.68

Model	# of Gates	Accuracy
VQP	**9**	**0.71**
VQC with gradient	9	0.73
VQC* with gradient	12	0.77

This table shows the VQC with gradient-based algorithm and VQP with Bayesian optimization framework, both for same machine learning task.

Gradient based algorithm shows advantages than BO based.

Solution and future task:

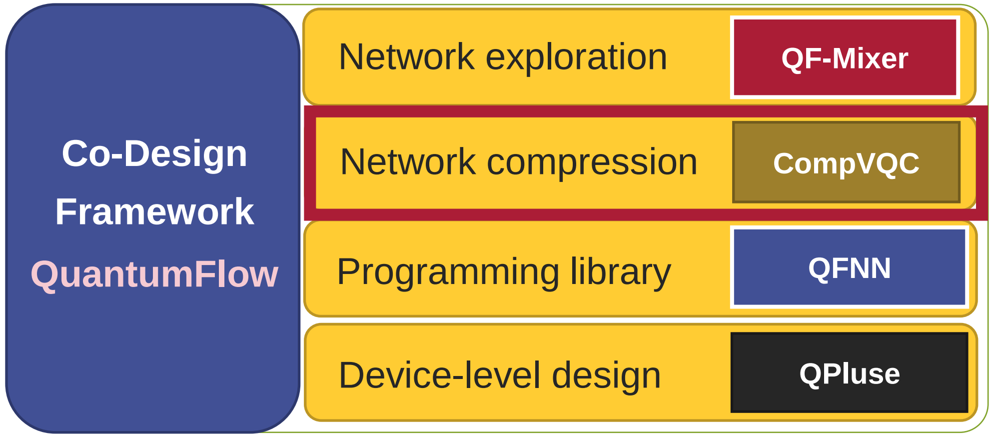
1. Improvement on optimization process.
2. Developing an efficient and differentiable pulse simulator.

CompVQC

Quantum Neural Network Compression

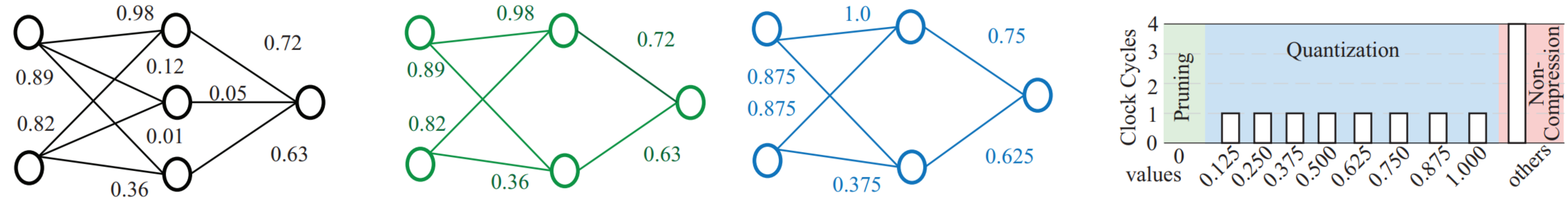
https://arxiv.org/pdf/2207.01578.pdf

Presenter: Weiwen Jiang



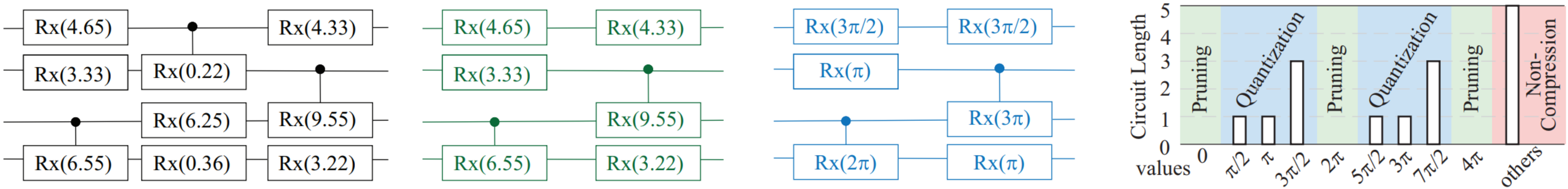
Compression: From Classical To Quantum

- Pruning and Quantization in Classical ML



(a) Non-Compression Classical NN (b) Classical NN with Pruning (c) Pruned NN with Quantization (d) Cost of Different Levels in Classical NN

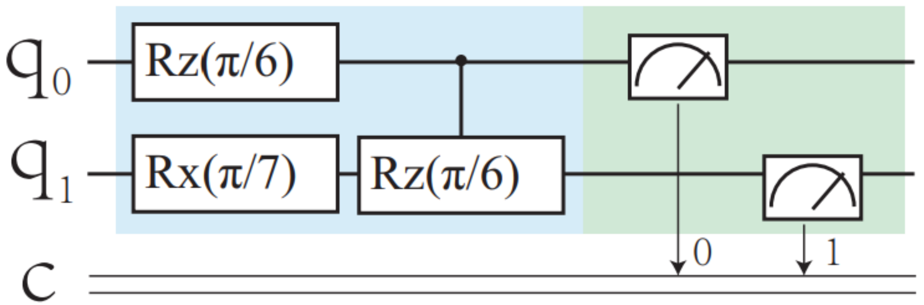
- Pruning and Quantization in Quantum ML



(e) Non-Compression QNN (f) QNN with Pruning (g) Pruned QNN with Quantization (h) Cost of Different Levels in RX Gate in QNN

- **Pruning:** Not only 0 can be pruned, but also 2pi, 4pi, etc.
- **Quantization:** Different quantization level may have different cost

Quantum Compression is Compilation Aware



Compilation

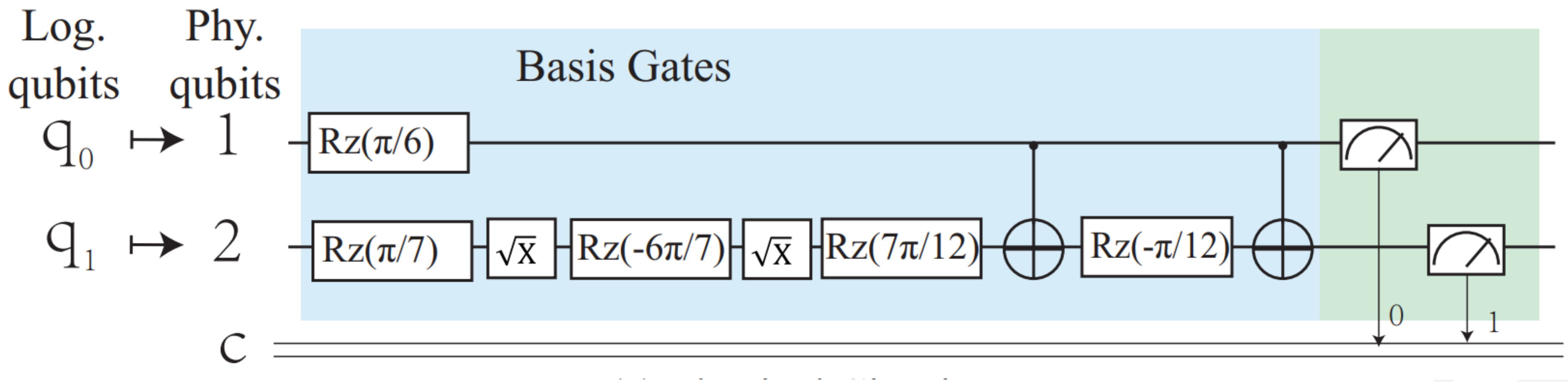
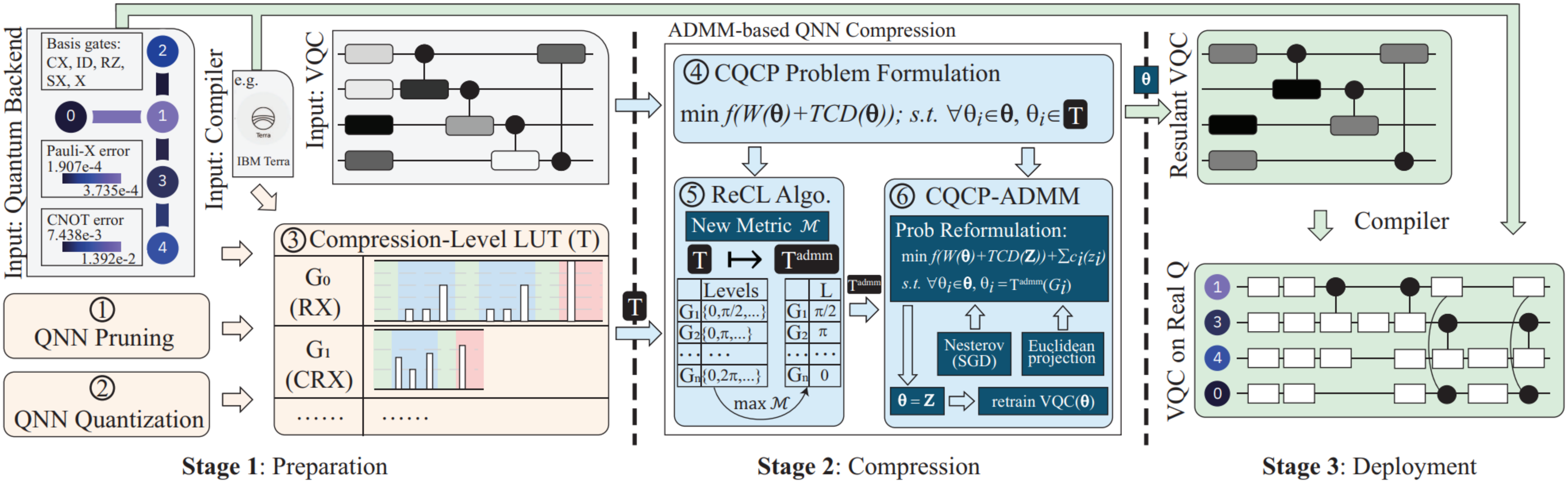


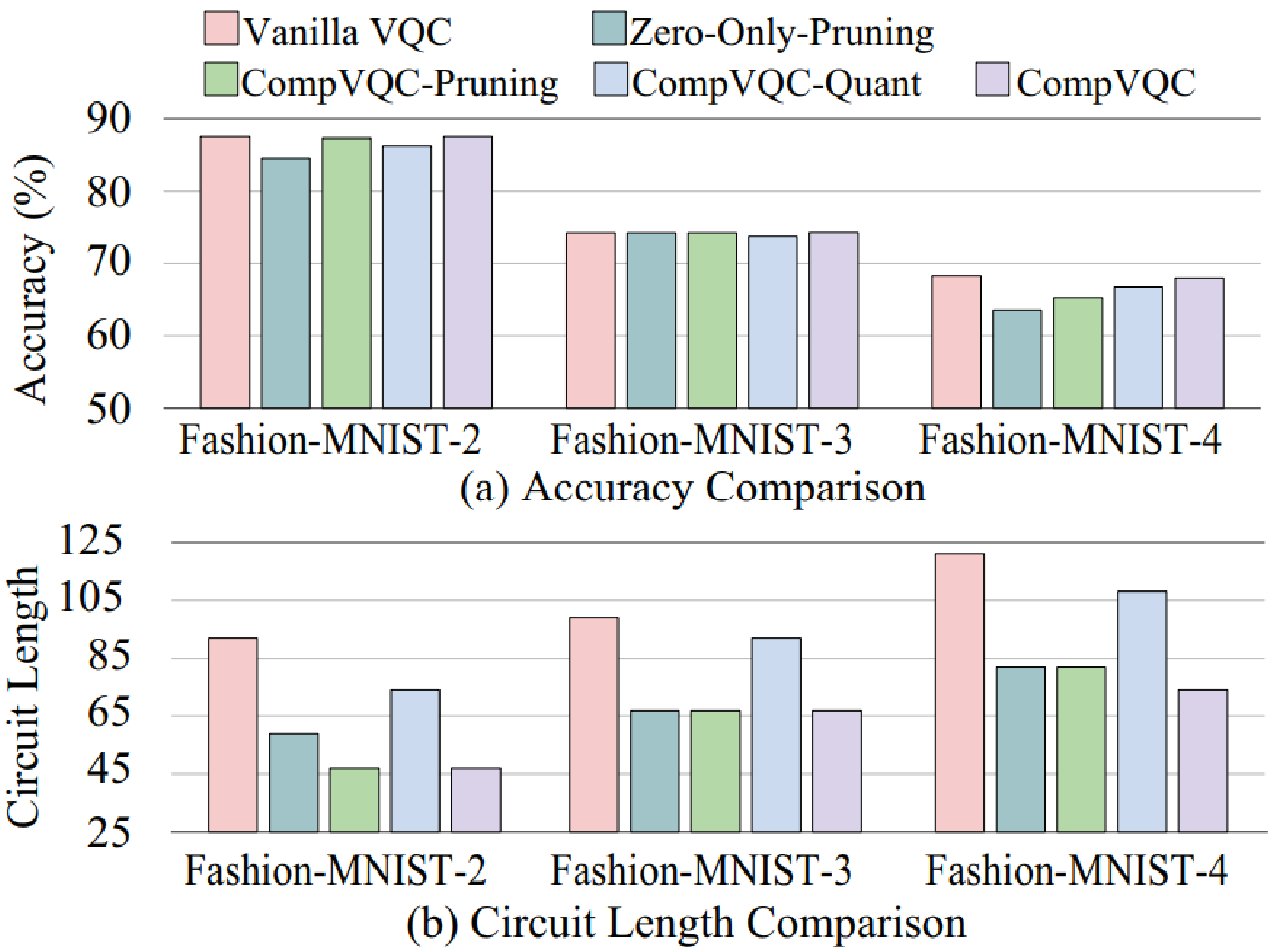
Table 1: circuit depth of compiled quantum gates on IBM quantum processors; parameters are in the range of $[0, 4\pi]$

Gate	0	π	2π	3π	4π	$\pi/2$	$3\pi/2$	$5\pi/2$	$7\pi/2$	others
RX	0	1	0	1	0	1	3	1	3	5
RY	0	2	0	2	0	3	3	3	3	4
CRX	0	8	5	9	0	11	11	11	11	11
CRY	0	8	6	8	0	10	10	10	10	10

CompVQC Framework



Results



- CompVQC can maintain high accuracy with **<1% accuracy loss**
- CompVQC can reduce circuit length by **2.5X**

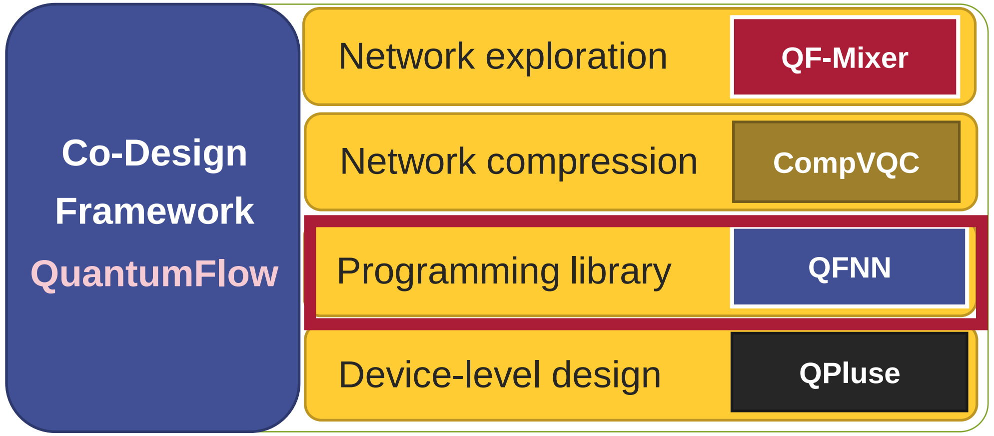
Quantum NN Library: QFNN

QuantumFlow Neural Network (QFNN) API

IEEE QUANTUM WEEK
IEEE International Conference on Quantum Computing and Engineering — QCE21

Released at IEEE International Conference on Quantum Computing and Engineering

Presenter: Weiwen Jiang



https://github.com/JQub/QuantumFlow_Tutorial
(Source Code of All Hands-On in Tutorial)

https://github.com/JQub/qfnn
(Source Code of QFNN API & Place to post Issues)

https://pypi.org/project/qfnn/
(Package of QFNN on PYPI)

https://libraries.io/pypi/qfnn/
(QFNN on Libraries.io)

Open-Source Quantum NN Library: QFNN

Qiskit + PyTorch + QuantumFlow

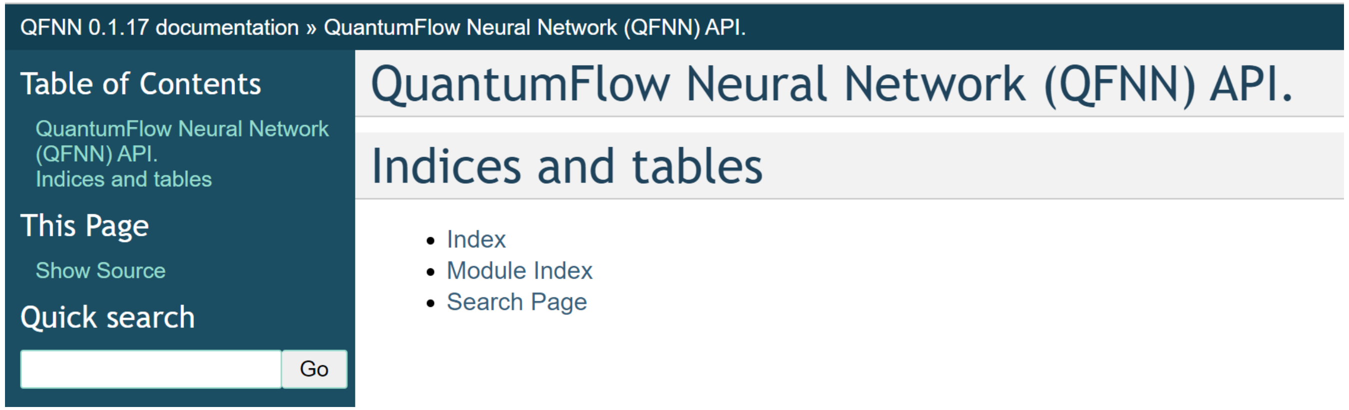

QFNN 0.1.17 documentation » QuantumFlow Neural Network (QFNN) API.

Table of Contents

QuantumFlow Neural Network (QFNN) API.
Indices and tables

This Page

Show Source

Quick search

Go

QuantumFlow Neural Network (QFNN) API.

Indices and tables

- Index
- Module Index
- Search Page

https://jqub.ece.gmu.edu/categories/QF/qfnn/index.html

https://github.com/jqub/qfnn

Example 1: QuantumFlow

Sub module of `qfnn.qf_circ`

- **Given:** (1) Number of input neural$2^{\mathcal{N}}$; (2) number of output neuron $\mathcal{M}$;

  (3) input $\mathcal{I}$; (4) weights $\mathcal{W}$; (5) an empty quantum circuit $\mathcal{C}$
- **Do:** (1) Encode inputs to the circuit; (2) embed weights to the circuit; (3) do accumulation and quadratic function
- **Output:** (1) Quantum circuit $\mathcal{C}$ with $\mathcal{M}$ output qubits

$\mathcal{C}$

```
#create circuit
circuit = QuantumCircuit()
```

$\mathcal{N}$ for $2^{\mathcal{N}}$ data $\mathcal{M}$

```
#init circuit, which is corresponding to a neuron with 4 qubits and 2 outputs
u_layer = U_LYR_Circ(4,2)

#create qubits to be invovled
inps = u_layer.add_input_qubits(circuit)
aux =u_layer.add_aux(circuit)
u_layer_out_qubits = u_layer.add_out_qubits(circuit)
```

$\mathcal{W}$ $\mathcal{I}$

```
#add u-layer to your circuit
u_layer.forward(circuit,binarize(weight_1),inps,u_layer_out_qubits,quantum_matrix,aux)
```

$\mathcal{C}$

```
#show your circuit
circuit.draw('text',fold=300)
```

Algorithm 4: QF-Map: weight mapping algorithm

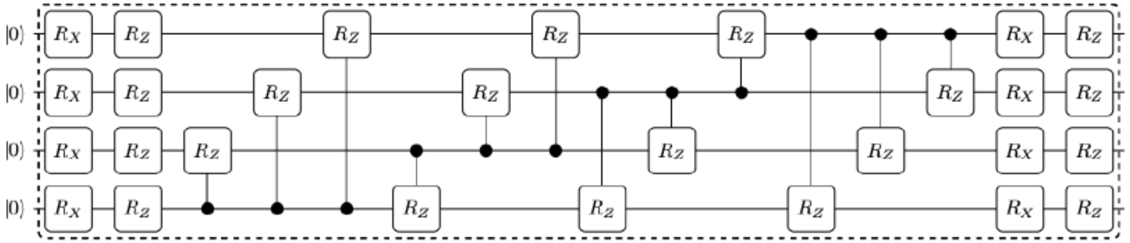
Input: (1) An integer $R \in (0, 2^{k-1}]$; (2) number of qbits k;
Output: A set of applied gate G

```
void recursive(G,R,k){
    if (R < 2^{k-2}){
        recursive(G,R,k − 1); // Case 1 in the third step
    }
    else if (R == 2^{k-1}){
        G.append(PG_{2^{k-1}}); // Case 2 in the third step
        return;
    }else{
        G.append(PG_{2^{k-1}});
        recursive(G,2^{k-1} − R,k − 1); // Case 3 in the third step
    }
}
// Entry of weight mapping algorithm
set main(R,k){
    Initialize empty set G;
    recursive(G,R,k);
    return G
}
```

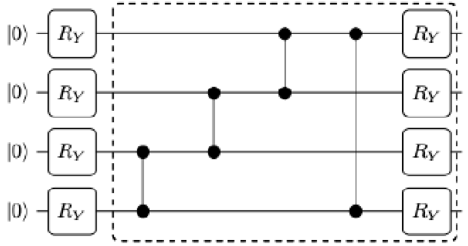
Example 2: Variational Quantum Circuits

Sub module of `qfnn.qf_circ`

- **Given:** (1) Number of input qubits $\mathcal{N}$; (2) weights $\mathcal{W}$; (3) a quantum circuit $\mathcal{C}$ with input data having been encoded
- **Do:** (1) embed weights $\mathcal{W}$ to the circuit;
- **Output:** (1) Quantum circuit $\mathcal{C}$ with measurements



Circuit 5

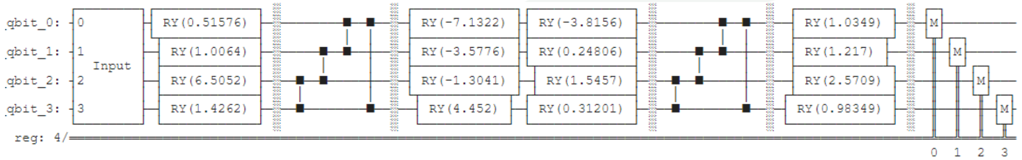


Circuit 10

```
#define your input qubits
vqc = V_LYR_Circ(4)
#add the first v-layer  to your circuit; We currently provide V10 and V5 only
vqc.forward(circuit,inputs,'v10',np.array(theta1,dtype=np.double))
#add the second v-layer  to your circuit
vqc.forward(circuit,inputs,'v10',np.array(theta2,dtype=np.double))

circuit.barrier()
#add measurement to your circuit if needed
add_measure(circuit,[inputs[0][0],inputs[0][1],inputs[0][2],inputs[0][3]],'reg')

circuit.draw('text',fold=300)
```

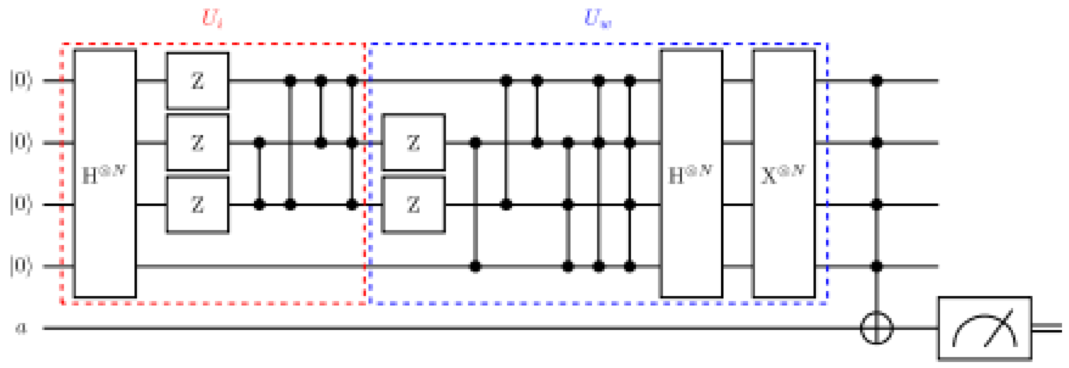


Example 3: An artificial neuron implemented on an actual quantum processor

Sub module of `qfnn.qf_circ`

- **Given:** (1) Number of input qubits $\mathcal{N}$; (2) number of output neuron $\mathcal{M}$;

  (3) a quantum circuit $\mathcal{C}$ with input data having been encoded
- **Do:** (1) embed weights to the circuit; (2) do accumulation and quadratic function
- **Output:** (1) Quantum circuit $\mathcal{C}$ with $\mathcal{M}$ output qubits



$\mathcal{N}$ $\mathcal{M}$

```
#define your input and repeat number
f_layer = F_LYR_Circ(4,2)

#add qubits to your circuit if needed
aux = f_layer.add_aux(circuit)
f_layer_out_qubits = f_layer.add_out_qubits(circuit)

#add f-layer to your circuit
f_layer.forward(circuit,binarize(weight_1),inputs,f_layer_out_qubits,None,aux)

circuit.barrier()
circuit.draw('text',fold=300)
```

$\mathcal{C}$ $\mathcal{W}$

$\mathcal{C}$

Outline

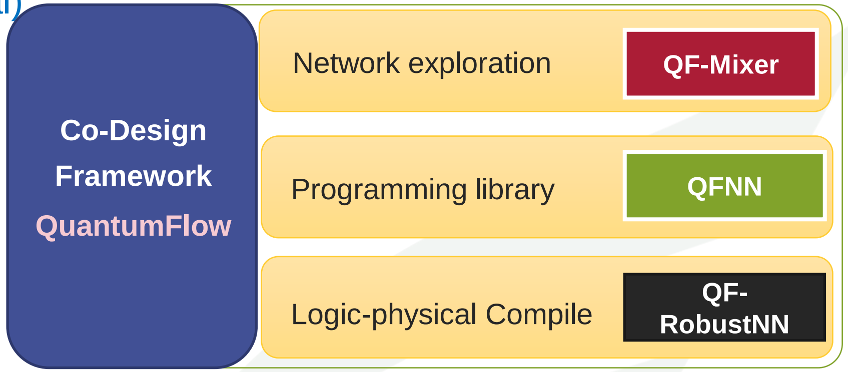
- Background
- Perspective: Co-Design --- from Classical to Quantum
- Built Design Stack
  - Quantum Neuron with Quantum Advantage: QuantumFlow
  - Quantum NN Library: QFNN
  - Quantum Neural Network: QF-Mixer
  - Quantum Pluse: VQP
- **Conclusion**

Conclusion & Resources

- How to build up quantum circuit for **neural networks** from scratch
- **Co-design stack** can build a better *quantum neural network accelerator*
- Along with the development of quantum computers and quantum neural networks, we will see **real-world applications** in the NISQ Era

https://github.com/JQub/QuantumFlow_Tutorial (Source Code of All Hands-On in Tutorial)

https://github.com/JQub/qfnn (Source Code of QFNN API & Place to post Issues)



https://pypi.org/project/qfnn/ (Package of QFNN on PYPI)

https://libraries.io/pypi/qfnn/ (QFNN on Libraries.io)

https://jqub.ece.gmu.edu (JQub Website)

https://jqub.ece.gmu.edu/categories/QF (News and **slides**)

https://jqub.ece.gmu.edu/categories/QF/qfnn/ (QFNN Documents)

https://www.nature.com/articles/s41467-020-20729-5

https://arxiv.org/pdf/2012.10360.pdf

https://arxiv.org/pdf/2109.03806.pdf

https://arxiv.org/pdf/2109.03430.pdf

Thank you!

Goal: From AI Democratization to Quantum AI Democratization

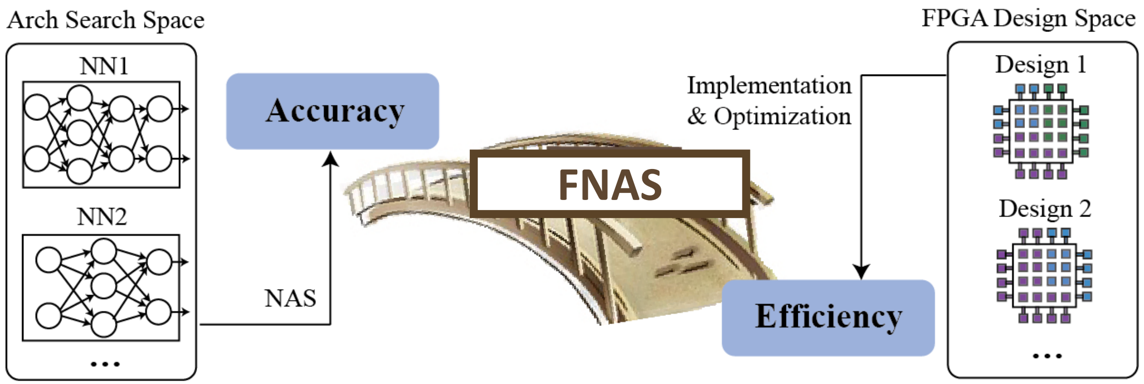
Our Previous Contributions

(Network-Accelerator Co-Design)

Given: (1) Dataset; (2) Target Hardware, e.g., FPGA.

Objective:
- Automated search for NN and HW design
- Maximize accuracy on the given dataset
- Maximize hardware efficiency

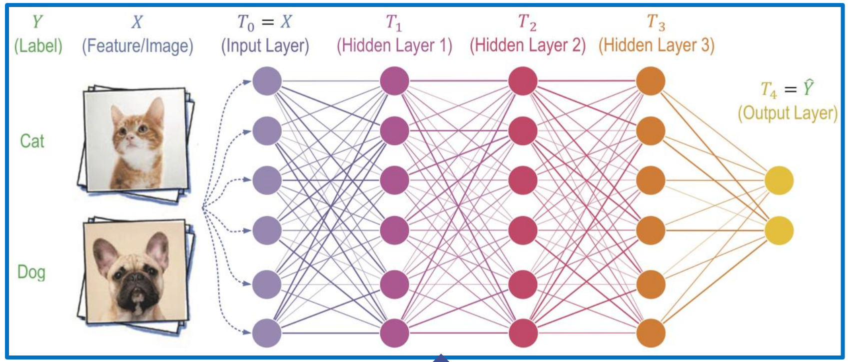
Output: A pair of neural network and hardware design



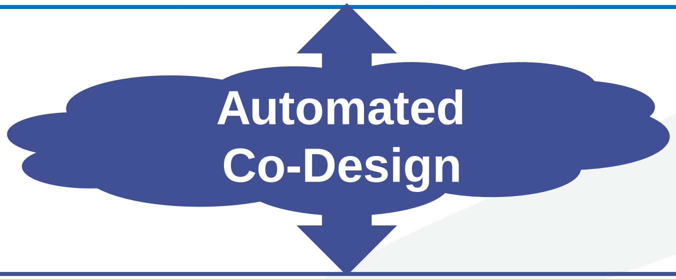
[ref] Jiang, Weiwen, et al. "Accuracy vs. efficiency: Achieving both through fpga-implementation aware neural architecture search." *DAC 2019*.

Quantum AI Democratization

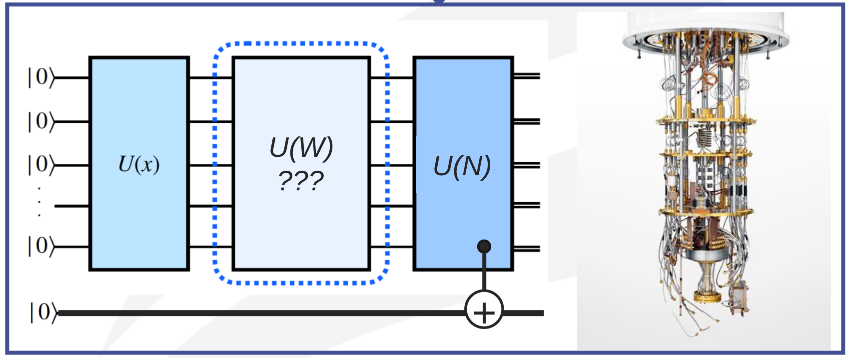
Given:
- Dataset
- Q Processor



Objective:
- Accuracy ↑
- # of Qubits ↓
- Circuit Len. ↓



Output:
- Q NN
- Q Circuit
- Q Implement



Progress of Classical AI Democratization

Google's Initial Contributions
(Neural Architecture Search)

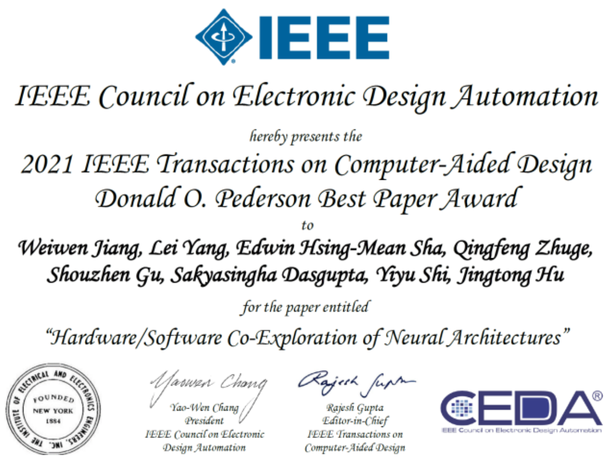
- NAS with RL (ICLR 2017)
- NAS with Para. Sharing (ICML 2018)
- NASNet (CVPR 2018)
- MNasNet (CVPR 2019)

…

Our Contributions
(Network-Accelerator Co-Design)

- FNAS (DAC 2019, Best Paper Nomination, BPN)
- FPGA & Network (CODES+ISSS 2019, BPN)
- NANDS for NoC (ASP-DAC 2020, BPN)
- FNAS+ (IEEE TCAD 2021 Best Paper Award)
- First place of 31st ACM SIGDA UBooth@DAC'21

…



PI: "Software Defined FPGA Hardware and Co-Exploration for Real-Time Applications", NSF IUCRC ASIC Center, **100K**, (Co-PI: Yiran Chen @ Duke)

Co-PI: "RAPID: Collaborative Research: Independent Component Analysis Inspired Statistical Neural Networks for 3D CT Scan Based Edge Screening of COVID-19", NSF IIS, **98K**, (PI: Prof. Yiyu Shi)

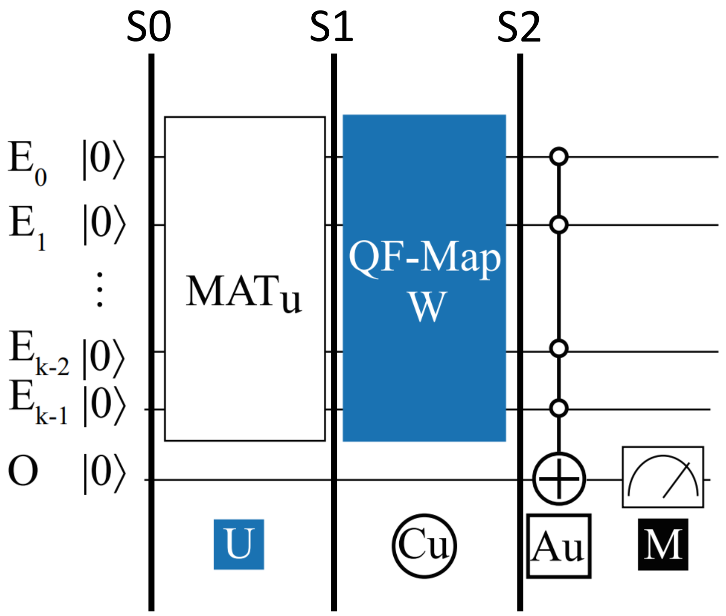
Co-PI: "Hardware/Software Co-Exploration of Multi-Modal Neural Architectures Targeting AR/VR Glasses", Facebook Research Funding, **75K**, (PI: Prof. Yiyu Shi)

$[0, 0.9, 0, 0, 0, 0.1, 0, 0, 1.0, 0.5, 0.5, 0, 0, 0, 0]^T$

U ↓

$[0, 0.59, 0, 0, 0, 0.07, 0, 0, 0.66, 0.33, 0.33, 0, 0, 0, 0]^T$

QuantumFlow: Taking NN Property to Design QC



S0 -> S1:

$$(v_o; v_{x1}; v_{x2}; \dots; v_{xn}) \times \begin{pmatrix} 1 \\ 0 \\ \dots \\ 0 \end{pmatrix} = (v_0)$$

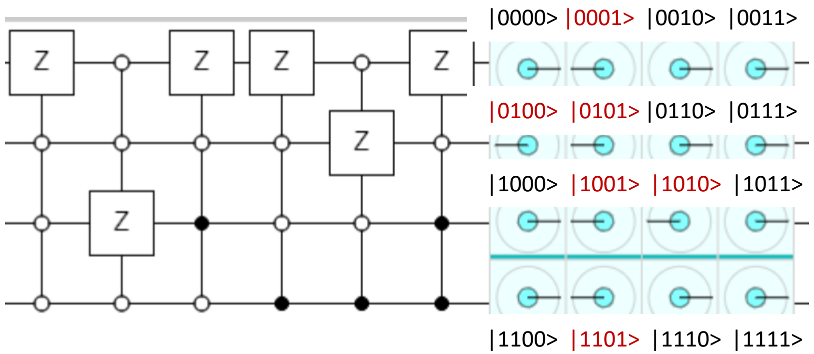
$$S1 = [0, 0.59, 0, 0, 0, 0.07, 0, 0, 0.66, 0.33, 0.33, 0, 0, 0, 0]^T$$

S1 -> S2:

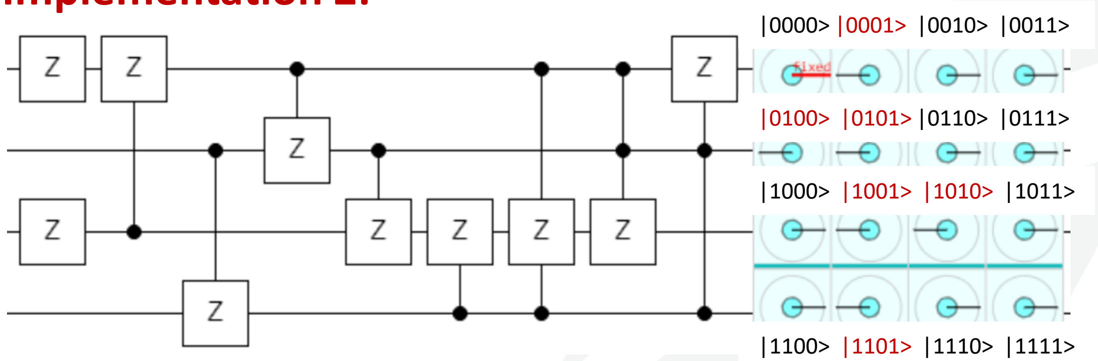
$$\begin{array}{l} \mathrm{W} = [+1, -1, +1, +1, -1, -1, +1, +1, +1, -1, -1, +1, +1, -1, +1, +1]^T \\ |0000\rangle\ |0001\rangle\ |0010\rangle\ |0011\rangle\ |0100\rangle\ |0101\rangle\ |0110\rangle\ |0111\rangle\ |1000\rangle\ |1001\rangle\ |1010\rangle\ |1011\rangle\ |1100\rangle\ |1101\rangle\ |1110\rangle\ |1111\rangle \end{array}$$

$$S2 = [0, -0.59, 0, 0, -0, -0.07, 0, 0, 0, -0.66, -0.33, 0.33, 0, -0, 0, 0]^T$$

Implementation 1 (example in Quirk):

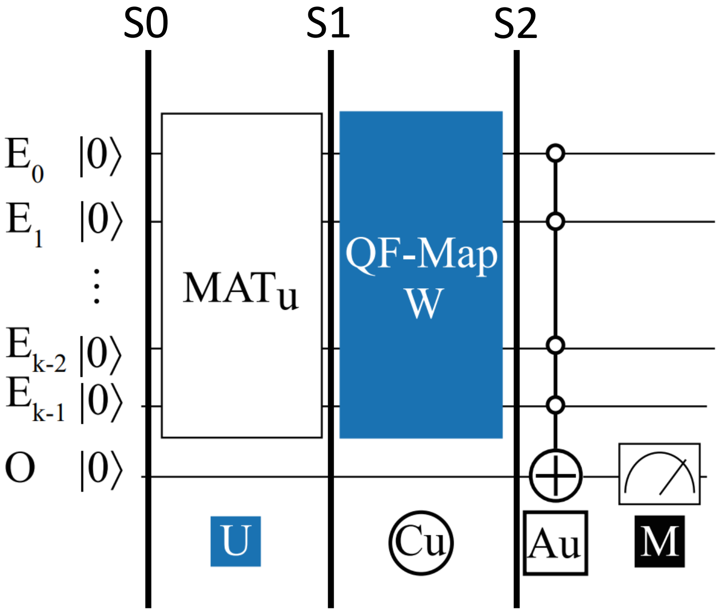


Implementation 2:



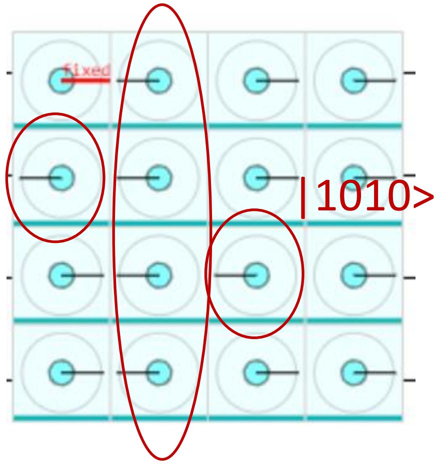
[ref] Tacchino, F., et al., 2019. An artificial neuron implemented on an actual quantum processor. *npj Quantum Information*, 5(1), pp.1-8.

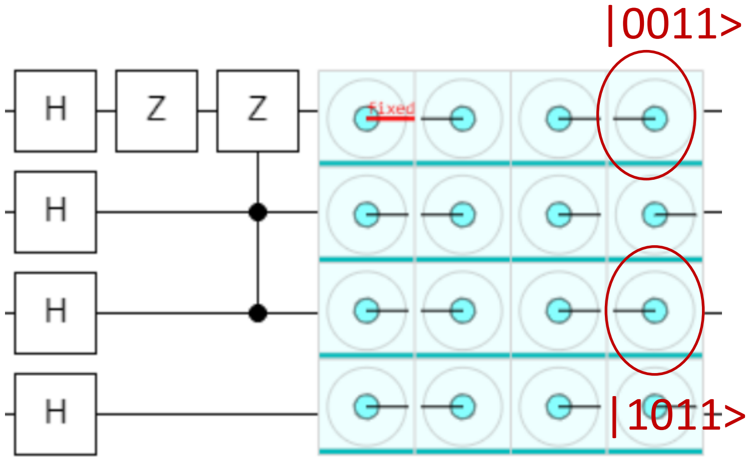
QuantumFlow: Quantum Neuron Optimization



Property from NN

- The **weight order** is not necessary to be fixed, which can be adjusted if the order of inputs are adjusted accordingly
- **Benefit:** No need to require the positions of sign flip are exactly the same with the weights; instead, only need the number of signs are the same.





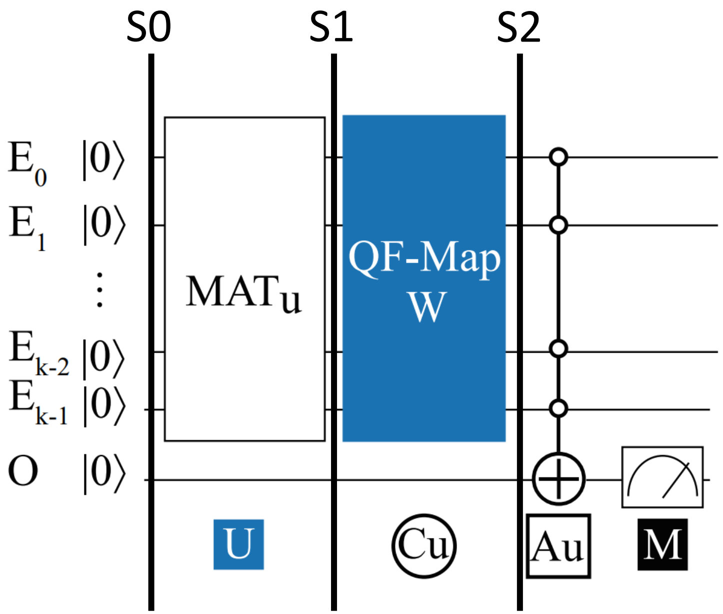
$$S1 = [0, 0.59, 0, \mathbf{0}, \mathbf{0}, 0.07, 0, 0, 0.66, \mathbf{0.33}, \mathbf{0.33}, 0, 0, 0, 0]^T$$

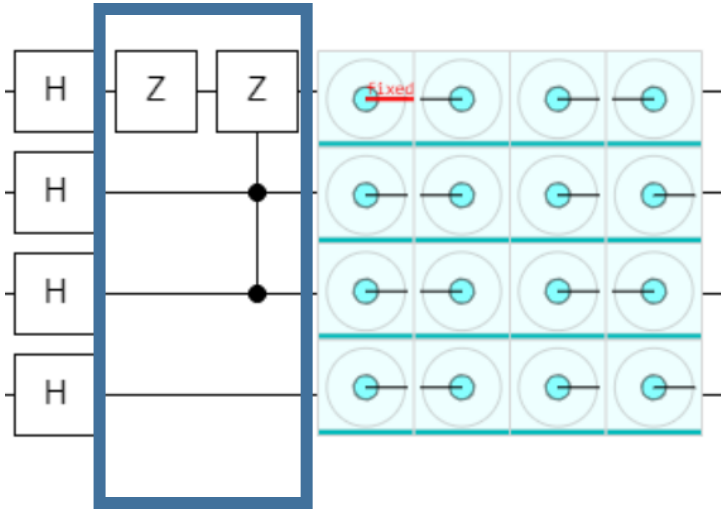
ori + - - +

fin - + + -

$$S1' = [0, 0.59, 0, \mathbf{0.33}, \mathbf{0.33}, 0.07, 0, 0, 0.66, \mathbf{0}, \mathbf{0}, 0, 0, 0, 0]^T$$

QuantumFlow: Quantum Neuron Optimization





Algorithm 4: QF-Map: weight mapping algorithm

Input: (1) An integer $R \in (0, 2^{k-1}]$; (2) number of qbits k;
Output: A set of applied gate G

void recursive(G,R,k){
    if ($R < 2^{k-2}$){
        recursive(G,R,$k-1$); // Case 1 in the third step
    }
    else if ($R == 2^{k-1}$){
        $G.append(PG_{2^{k-1}})$; // Case 2 in the third step
        return;
    }else{
        $G.append(PG_{2^{k-1}})$;
        recursive(G,$2^{k-1}-R$,$k-1$); // Case 3 in the third step
    }
}
// Entry of weight mapping algorithm
set main(R,k){
    Initialize empty *set* G;
    recursive(G,R,k);
    return G
}

Used gates and Costs

Gates	Cost
Z	1
CZ	1
C^2Z	3
C^3Z	5
C^4Z	6
…	…
C^kZ	2k-1

Worst case: all gates

$O(k^2)$