Tutorial on QuantumFlow: A Co-Design Framework of Neural Network and Quantum Circuit towards Quantum Advantage

Session 4: Future Works: QF-Mixer, QF-RobustNN, and Others

Weiwen Jiang, Ph.D.

Assistant Professor

Electrical and Computer Engineering

George Mason University

wjiang8@gmu.edu

https://jqub.ece.gmu.edu

Agenda – Session 4: Extensions

Zhepeng Wang
Electrical and Computer Engineering
George Mason University
zwang48@gmu.edu

- **QF-Mixer: Exploring Quantum Neural Architecture**
  - Motivation: Existing Quantum Neuron Designs Can Be Complementary
  - Design Principle: Mixing Designs is Harder Than Your Thoughts!
  - Results
- **QF-RobustNN: Learning Noise in Quantum Neural Networks**
- **Open Questions and Future Work**

Speaker Information

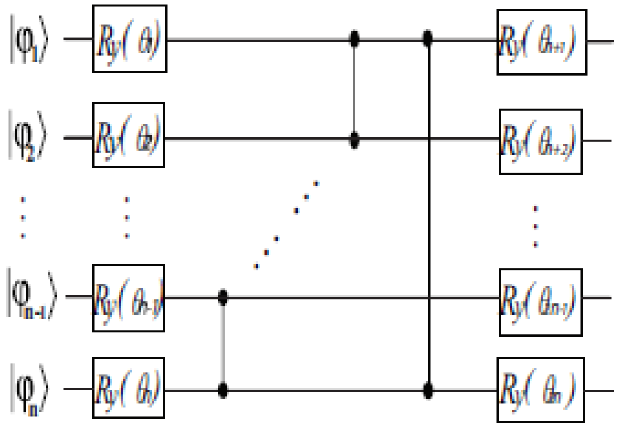
Dr Jiang's Quantum-Classical Computer-Aided Design Lab (JQub)

- **Zhepeng Wang**
  - Ph.D. student, ECE, George Mason University
  - Graduate research assistant of JQub
  - M.S., ECE, University of Pittsburgh
  - B.S., CS, Harbin Institute of Technology
- **Advisor: Dr. Weiwen Jiang**
  - Assistant Professor, ECE, George Mason University
  - Founder and director of JQub
- **Research Interest**
  - Quantum machine learning
  - On-Device AI

Background and Motivation

Existing Quantum Neuron Designs

▪ **Variational quantum circuit (VQC)-based neuron**



(b) V-NEU

V-Neuron (V-NEU)

- A widely used quantum neuron
- Reuse the input qubits as output qubits

Advantage

- **Real-valued weights**

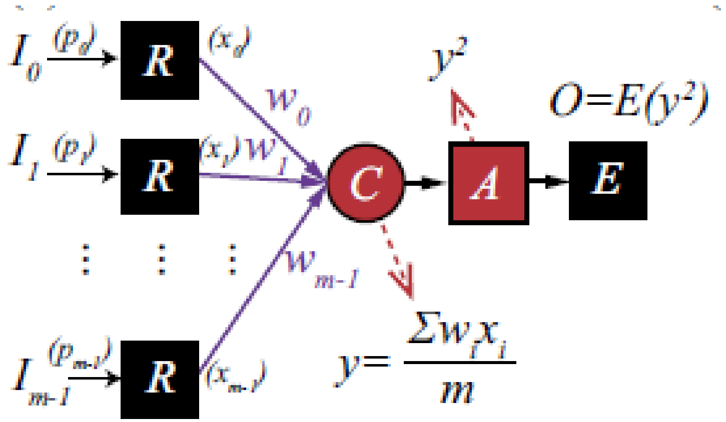
Disadvantage

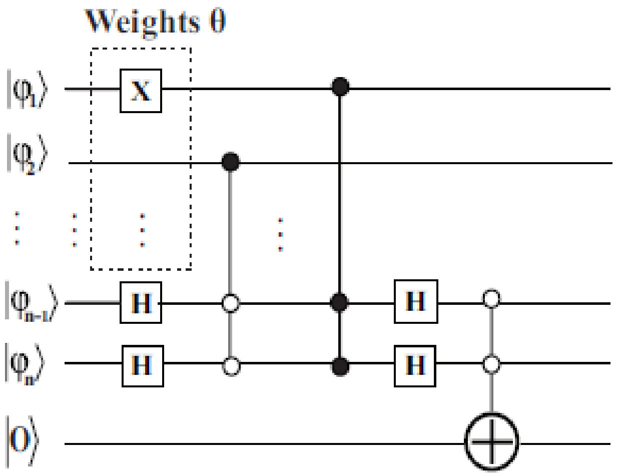
- **Linear classifier**
- **Cannot be extended to multiple nonlinear layers with low cost**

- Make use of the entanglement from quantum computing to increase the model complexity

Existing Quantum Neuron Designs

- **Customized neurons of QuantumFlow**





(d) P-NEU

P-Neuron (P-NEU)

- Input encoding: *Probability encoding (Angle encoding)*
- Output encoding: *Probability encoding*

Advantage

- **Easy to be stacked to form multiple nonlinear layers**

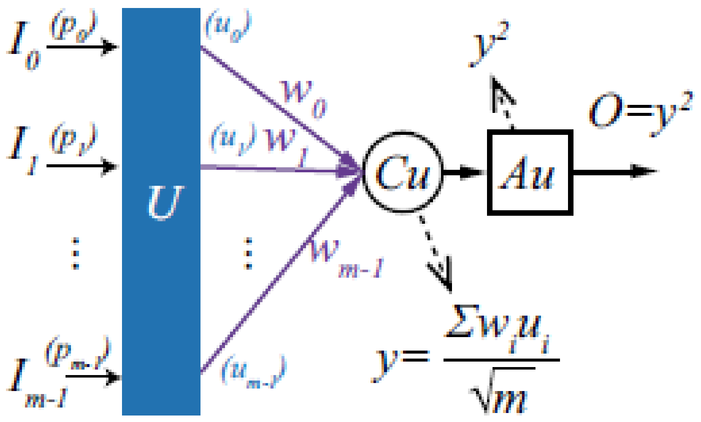
Disadvantage

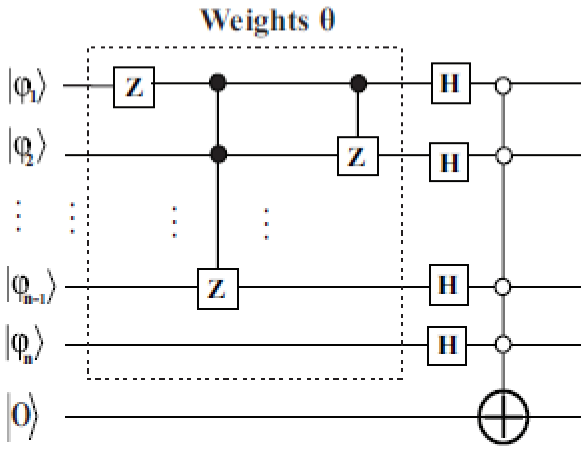
- **Binary weights**

[1] W. Jiang, et al. [A Co-Design Framework of Neural Networks and Quantum Circuits Towards Quantum Advantage](), Nature Communications

Existing Quantum Neuron Designs

- **Customized neurons of QuantumFlow**





(e) U-NEU

U-Neuron (U-NEU)

- Input encoding: *Amplitude encoding*
- Output encoding: *Probability encoding*

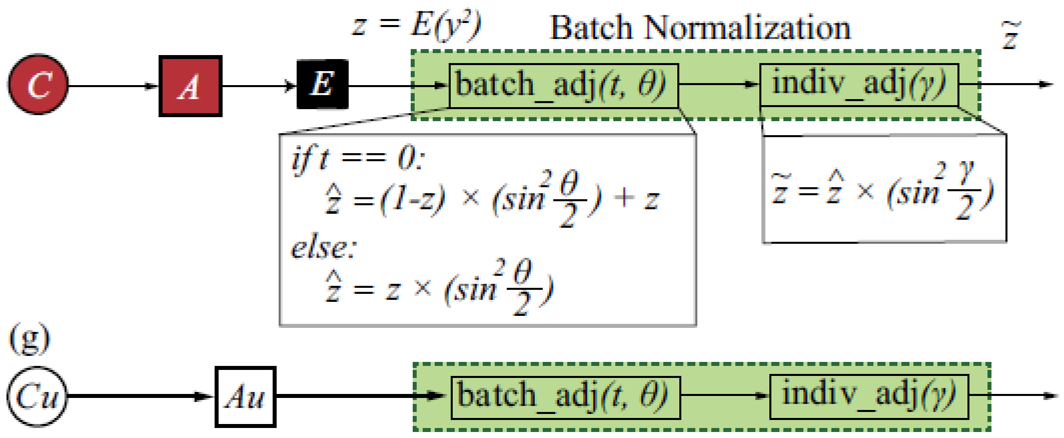
Advantage
• **It could be connected to P-Neuron seamlessly** • **It achieves quantum advantage**

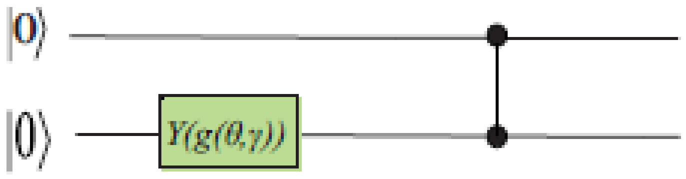
Disadvantage
• **Binary weights**

[1] W. Jiang, et al. A Co-Design Framework of Neural Networks and Quantum Circuits Towards Quantum Advantage, Nature Communications

Existing Quantum Neuron Designs

- **Customized neurons of QuantumFlow**





(c) N-NEU

N-Neuron (N-NEU)

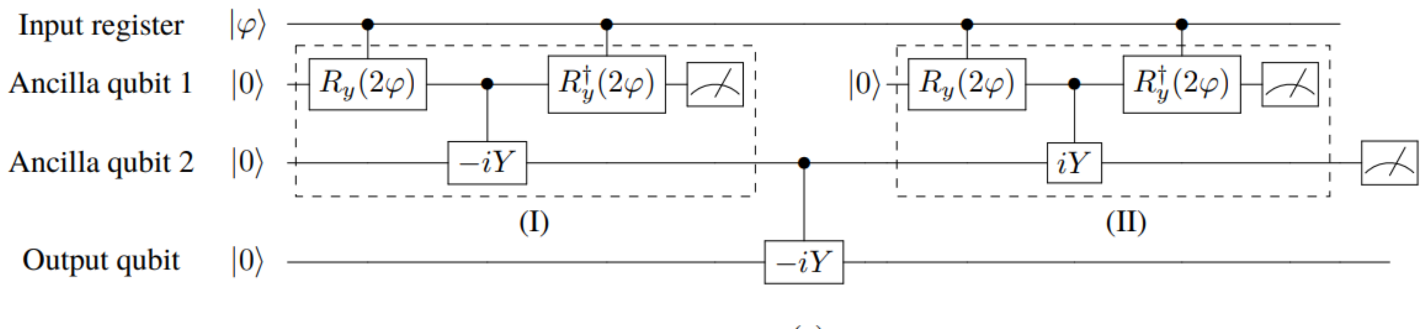
- Input encoding: *Probability encoding*
- Output encoding: *Probability encoding*

- **Optional**
- **Batch normalization**

[1] W. Jiang, et al. A Co-Design Framework of Neural Networks and Quantum Circuits Towards Quantum Advantage, Nature Communications

Existing Quantum Neuron Designs

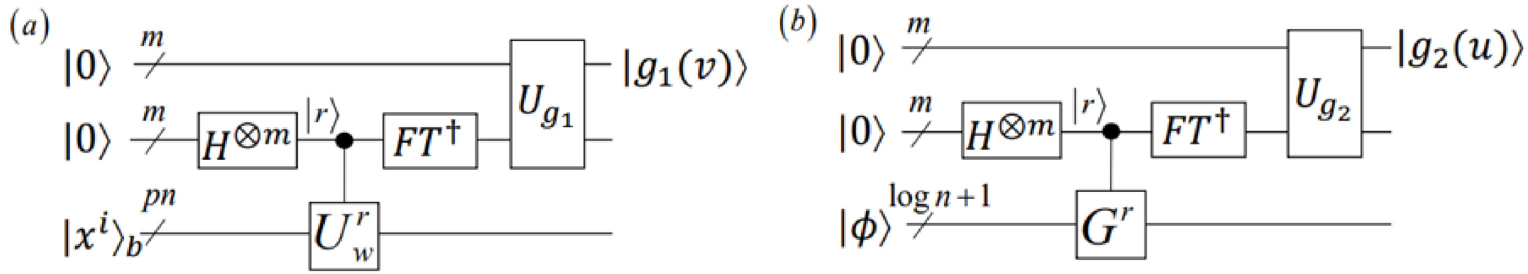
- **Q-Neuron**



Disadvantage

- **Data encoding: one-to-one mapping (almost impossible to achieve quantum advantage)**
- **Repeat-until-success to build non-linear function (Inefficient)**

- **Q-Non-Linear Neuron**



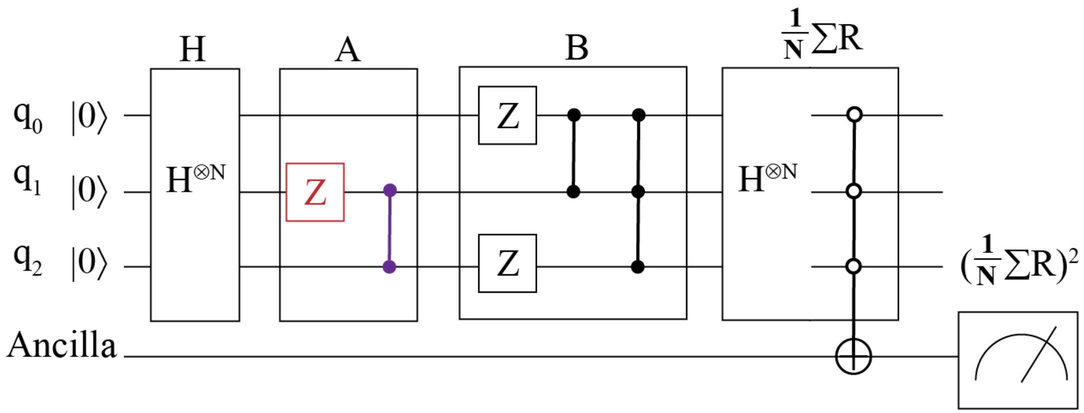
Apply boolean function to realize any non-linear function

Disadvantage

- **Quantum advantage cannot be achieved**

Existing Quantum Neuron Designs

- **Q-Artificial Neuron**



Implementing binary perceptron in quantum computer

Disadvantage
• **Both inputs and weights are binary**

Z. wang, et al. Exploration of Quantum Neural Architecture by Mixing Quantum Neuron Designs

Motivation

- Mixing different neurons could improve the performance of NN running on classical computers
- V-Neuron and neurons of QuantumFlow are complementary

V-NEU

Advantage
• Real-valued weights

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

Complementary

Neurons from QuantumFlow

Disadvantage
• Binary weights

Advantage
• Easy to be extended to multiple nonlinear layers w/t measurement

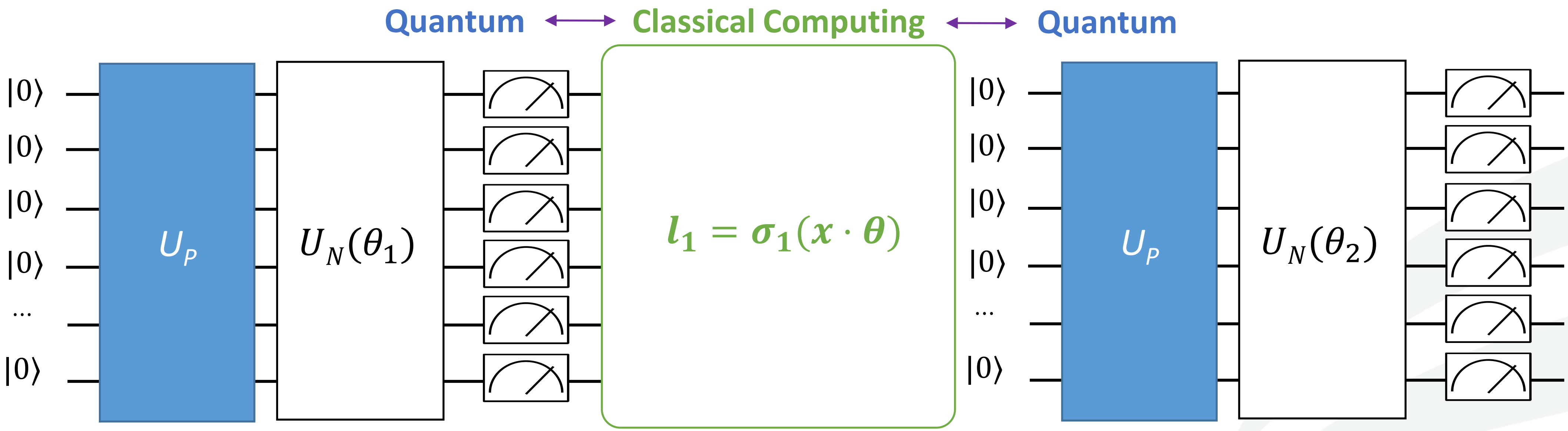
Motivation

- Challenges for mixing neurons
  - Quantum-classical communication overhead
  - Inconsistent design of same type of neurons
  - Inefficient training on classical computers

QF-Mixer

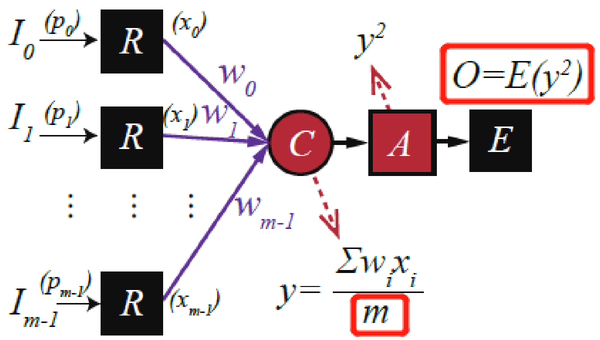
Design Goals of Mixing Neurons

- **Execution on quantum devices only**
  - W/t Measurement
  - No expensive quantum-classical communication overhead

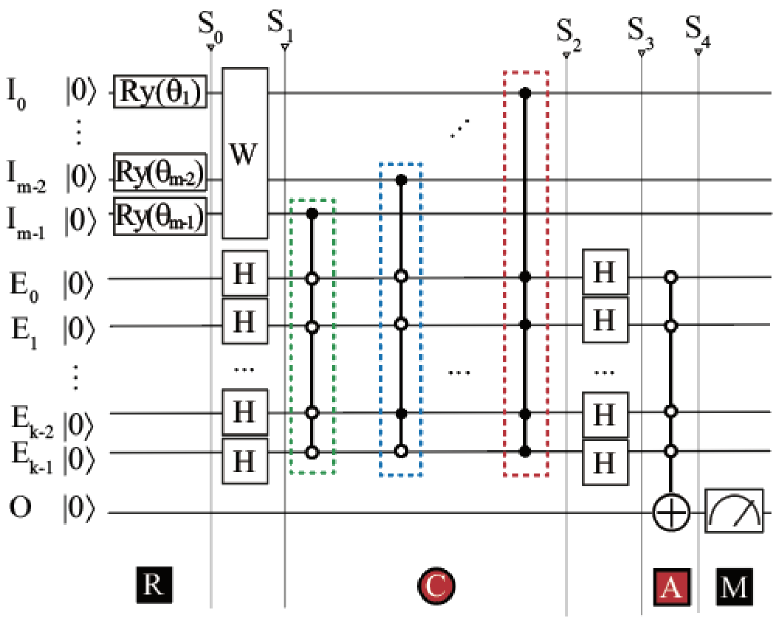


Design Goals of Mixing Neurons

- **Consistency on the encoding method of neurons regardless of the placement**
  - Consistent neuron computation
  - Consistent circuit design



P-NEU Neural Computation

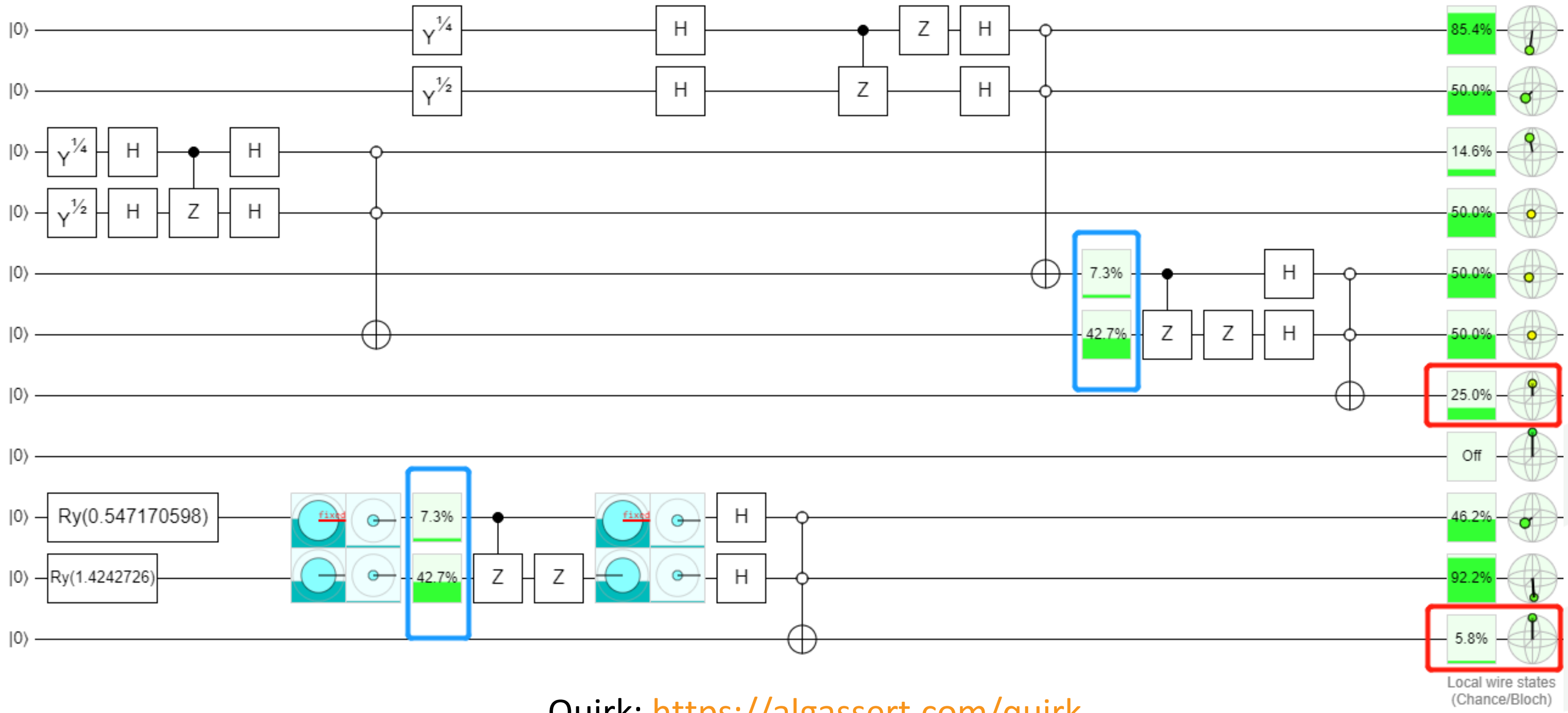


P-NEU Circuit implementation

[1] W. Jiang, et al. A Co-Design Framework of Neural Networks and Quantum Circuits Towards Quantum Advantage, Nature Communications

Design Goals of Mixing Neurons

- **Training efficiently on classical computers**
  - Training the whole QNN directly on classical computing is costly
  - Decoupling the neurons of different layers is important



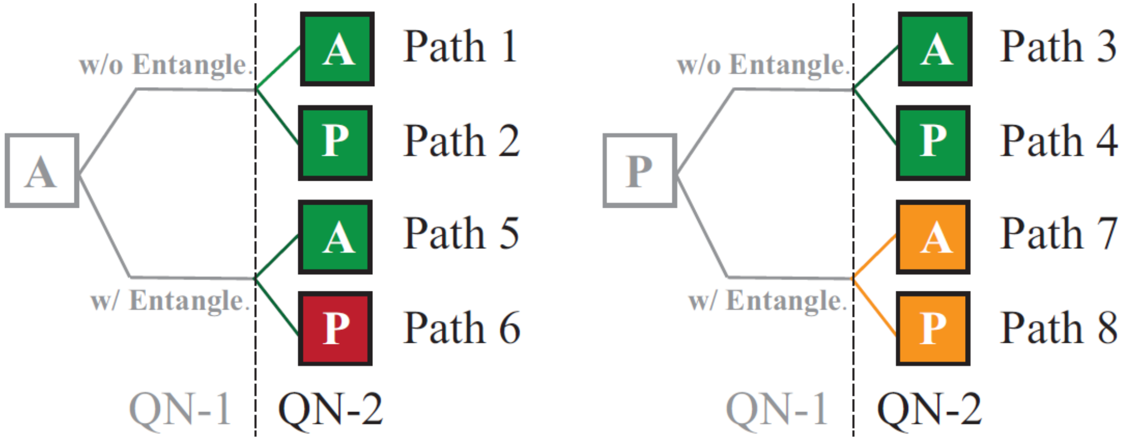
Quirk: https://algassert.com/quirk

Design Goals of Mixing Neurons

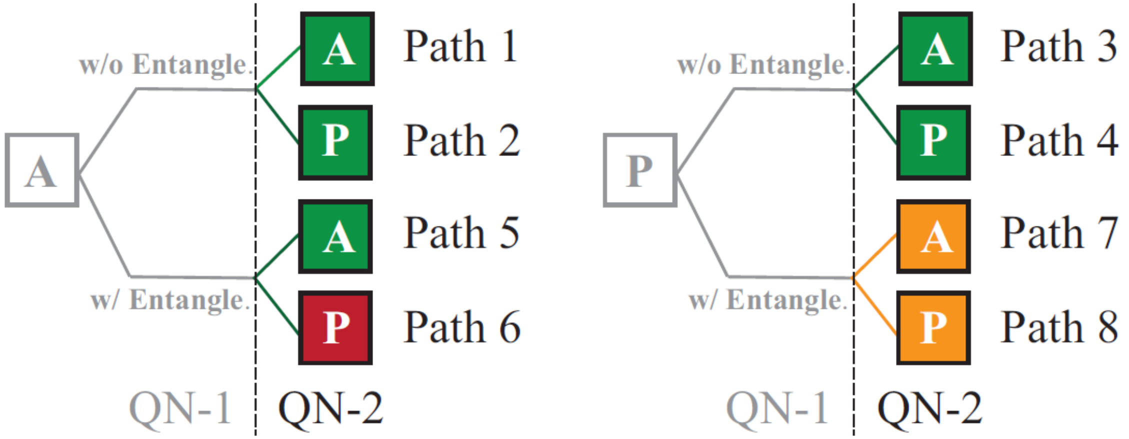
- **Execution on quantum devices only**
  - W/t Measurement
  - No expensive quantum-classical communication overhead
- **Consistency on the encoding method of neurons regardless of the placement**
  - Consistent neuron computation
  - Consistent circuit design

- **Training efficiently on classical computers**
  - Training the whole QNN directly on classical computing is costly
  - Decoupling the neurons of different layers is important

Design Principles



Design Principles



- **P: Probability encoding**
- **A: Amplitude encoding**

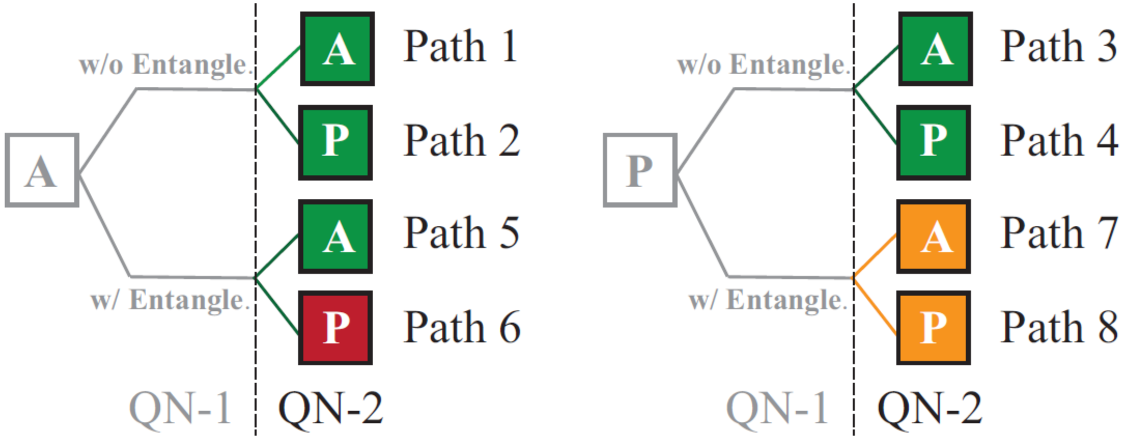
Output qubits of QN-1 are not entangled

- **Principle 1 (*Path 1-4*)**
  - The output qubits from QN-1 are decoupled with the output qubits of its previous layers.
  - Conclusion: Feasible

Output qubits of QN-1 are entangled

- **Principle 2 (*Path 5*)**
  - W/o probability encoding involved, there is no requirement on the decoupling
  - Conclusion: Feasible

Design Principles

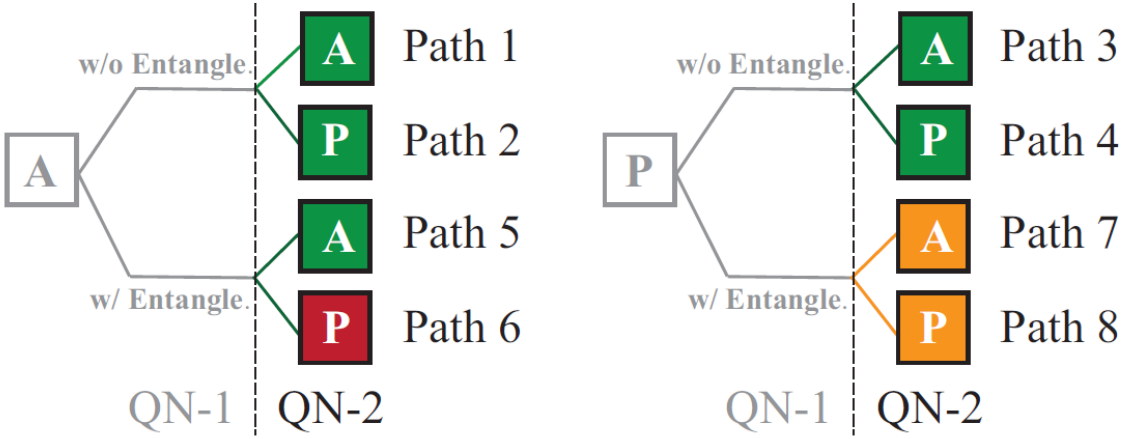


- **P: Probability encoding**
- **A: Amplitude encoding**

Output qubits of QN-1 are entangled

- **Principle 3 (*Path 6*)**
  - When QN-2 is a neuron in the first layer of a QNN and uses probability encoding, the input qubits are required to be independent.
  - Based on the goal of consistency, when QN-1 is the neuron in other layers, independence requirement should also hold.
  - Conclusion: Infeasible

Design Principles



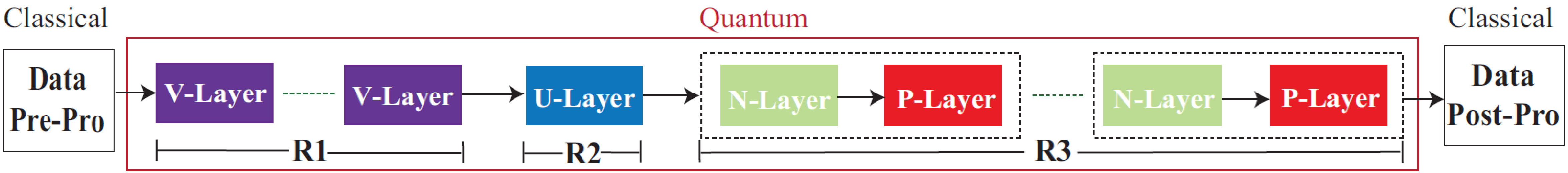
- **P: Probability encoding**
- **A: Amplitude encoding**

Output qubits of QN-1 are entangled

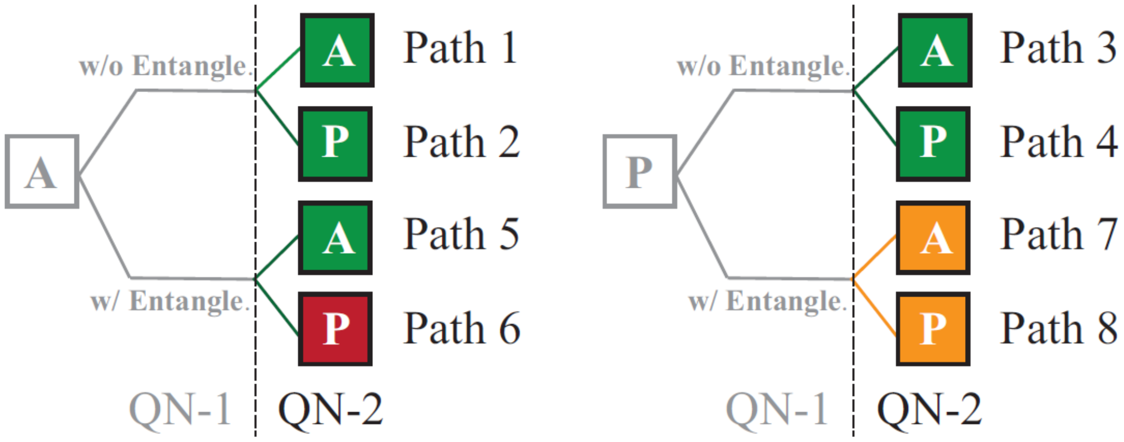
- **Principle 4 (*Path 7*)**
  - Conclusion: Conditional
  - Condition: The inputs qubits of QN-1 are reused by the output qubits, such as V-Layer.
- **Principle 5 (*Path 8*)**
  - Conclusion: Conditional
  - Condition:
    - Output qubits of QN-1 are used as control end without phase kickback
    - The operations on the output qubits of QN-1 only rotates them around X-axis

QF-MixNN

- **Pure quantum architecture**
  - The neural computation is conducted purely on quantum devices
  - Data pre-processing and post-processing are on classical devices
- **V-Layer should be the first**
  - Applying amplitude encoding to the input data
  - The extreme case is V-Layers only
  - Larger *R1* provides more real-valued weights
- **Multi-layer QNN can be formed**
  - U-Layer provides the non-linearity to the V-Layers, which will be added if *R2 = 1*
  - Larger *R3* corresponds to more non-linear layers



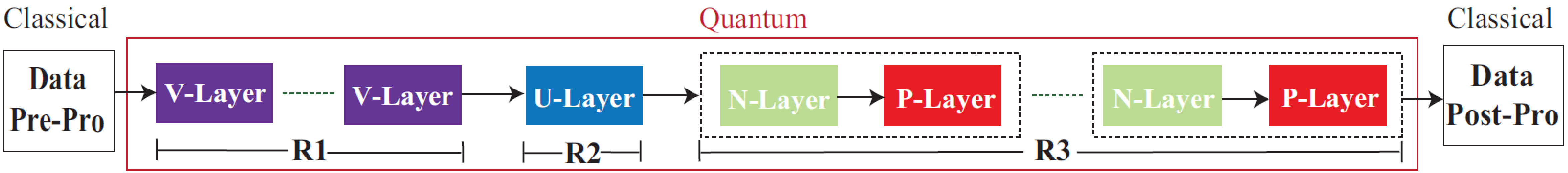
The Design of QF-MixNN Follows the Principles



Neuron Type	Input Encoding Method	Output Encoding Method
U-Neuron	Amplitude	Probability
V-Neuron	Amplitude	Amplitude/Probability
P-Neuron	Probability	Probability
N-Neuron	Probability	Probability

- V-NEU to V-NEU: Path 5
- V-NEU to U-NEU: Path 5
- U-NEU to N-NEU: Path 8
- N-NEU to P-NEU: Path 8
- V-NEU to P-NEU: Path 8

Feasible!



Experimental Results

QF-MixNN Achieves the Best Accuracy on MNIST

TABLE I
EVALUATION OF QNNS WITH DIFFERENT NEURAL ARCHITECTURE

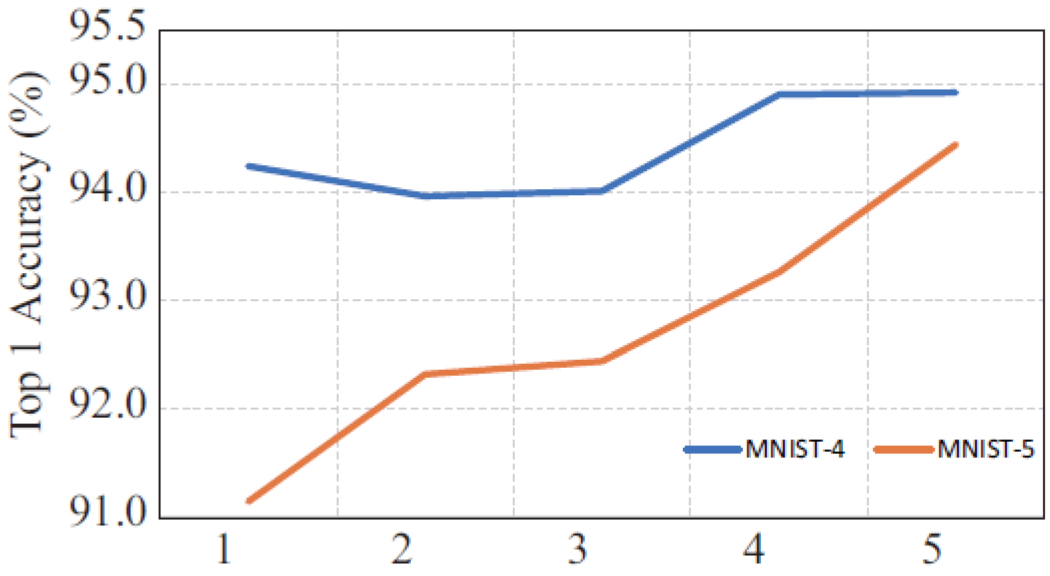
Architecture		MNIST-2†	MNIST-3†	MNIST-4‡	MNIST-5‡	MNIST§
VQC (V×R1)		**97.91%**	90.09%	93.45%	91.35%	52.77%
QuantumFlow		95.63%	91.42%	94.26%	89.53%	69.92%
QF-MixNN	V+U	97.36%	**92.77%**	**94.41%**	**93.85%**	88.46%
	V+U+P	87.45%	82.9%	92.44%	91.56%	**90.62%**
	V+P	91.72%	76.93%	88.43%	85.02%	49.57%

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

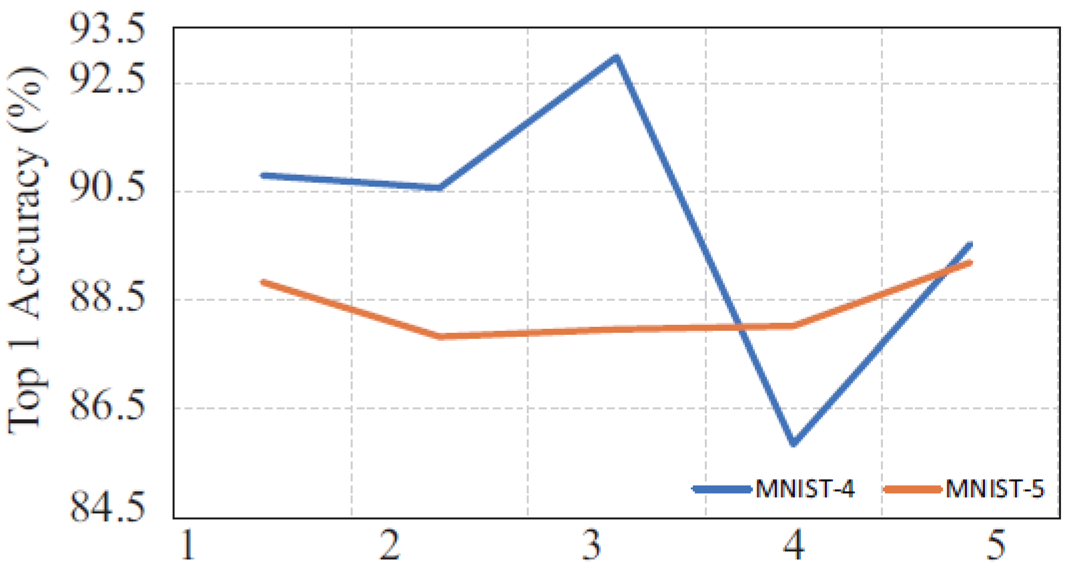
- **Non-linearity is important**. A linear decision boundary is not sufficient for complicated tasks.
- **Real-valued weight is helpful**. It increases the representation capability of QNN significantly.

OF-MixNN takes the advantage of both VQC-based QNN and QF-Net from Quantumflow.

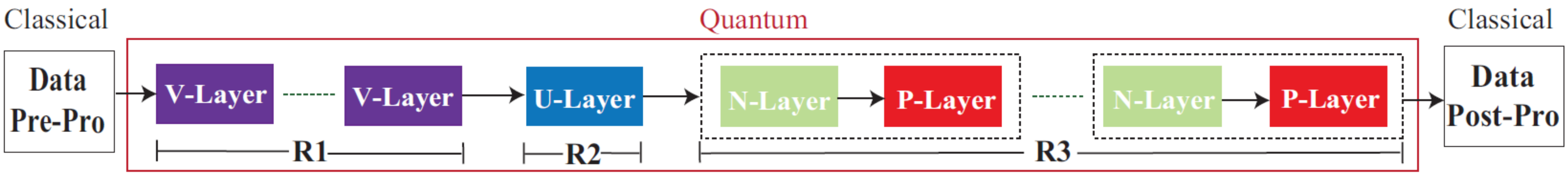
Increasing the Number of V-Layers Could Improve the Accuracy



(a) V X R1 + U



(b) V X R1 + U + P



API Demostration

QF-MixNN: c_qf_mixer.Net

Sub module of `qfnn.qf_fb`

- **Initialization:** (1) image_size: $\mathcal{M}$ for an image with size of $(\mathcal{M}$ X $\mathcal{M})$;

  (2) layer: An array describes the architecture of QF-MixNN;

  current support neuron: V-NEU (‘v5’ and ‘v10’), U-NEU (‘u’),

  P-NEU (‘p’), N-NEU (‘n’);

  (3) training: True if you want to train the QNN;

  (4) binary: *True* if the input image is binary;

  (5) given_ang: available only when N-NEU is used;

  (6) train_ang: available only when N-NEU is used
- **Forward:** (1) Given batch of encoded input images with size of $(\mathcal{M}$ X $\mathcal{M})$;

  (2) output the batch of prediction results;

QF-MixNN: c_qf_mixer.Net

Example: V + U

```
layers = [['v', 16], ['u', 2]] # Specify the architecture of QF-MixNN
img_size = 4 # Input image size is 4x4

# Initialize QF-MixNN
from qfnn.qf_fb.c_qf_mixer import Net
model = Net(img_size, layers, False, False) # Training and binary are False
model.load_state_dict(checkpoint["state_dict"]) # Load the pretrained parameters

# Encode the input image
to_quantum_data = ToQuantumData(img_size)
output_data = to_quantum_data(data)

# Make prediction using QF-MixNN
output = model(output_data, training=False) # It will call forward function
#show your circuit
print("inference result:", output)
```

QF-MixNN: c_qf_mixer.Net

Example: U + V

```
layers = [['u', 4], ['p2a', 16], ['v', 2]] # Specify the architecture of QF-MixNN
img_size = 4 # Input image size is 4x4

# Initialize QF-MixNN
from qfnn.qf_fb.c_qf_mixer import Net
model = Net(img_size, layers, False, False) # Training and binary are False
model.load_state_dict(checkpoint["state_dict"]) # Load the pretrained parameters

# Encode the input image
to_quantum_data = ToQuantumData(img_size)
output_data = to_quantum_data(data)

# Make prediction using QF-MixNN
output = model(output_data, training=False) # It will call forward function
#show your circuit
print("inference result:", output)
```

qfnn API Example (5)

QF-Mixer

V + U Example

U + V Example

V + U Example

U + V Example

QF-MixNN: c_qf_mixer.Net

More Examples

The Key point is to build the parameter *'layers'* !

```
layers = [['v', 16],['u', 2]] # V + U

layers = [['v', 4],['p', 2]] # V + P

layers = [['v', 16],['u', 4],['p', 2]] # V + U + P

layers = [['v', 16],['v', 16],['u', 4],['p', 2]] # V + V + U + P

layers = [['v', 16],['v', 16],['u', 4],['p', 4],['p', 2]] # V + V + U + P + P


img_size = 4 # Input image size is 4x4

# Initialize QF-MixNN
from qfnn.qf_fb.c_qf_mixer import Net
model = Net(img_size, layers, False, False) # Training and binary are False
model.load_state_dict(checkpoint["state_dict"]) # Load the pretrained parameters

# Encode the input image
to_quantum_data = ToQuantumData(img_size)
output_data = to_quantum_data(data)

# Make prediction using QF-MixNN
output = model(output_data, training=False) # It will call forward function
#show your circuit
print("inference result:", output)
```

The Other part remains unchanged.

Agenda – Session 4: Extensions

Zhiding Liang
Computer Science and Engineering
University of Notre Dame
zliang5@nd.edu

- QF-Mixer: Exploring Quantum Neural Architecture
- **QF-RobustNN: Learning Noise in Quantum Neural Networks**
  - Introduction to Noise in Quantum Computing
  - Motivation: Error Can Corrupt Quantum NN and Compiling Leads to Lengthy Learning
  - Application-Specific Compiler is Needed
  - Results
- **Open Questions and Future Work**

Speaker Biography

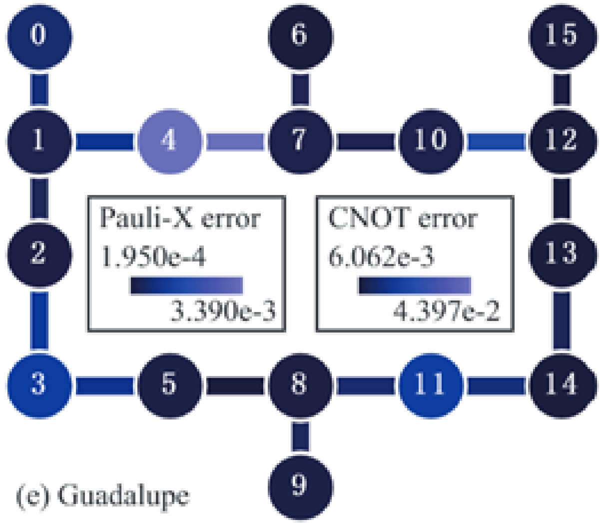
Email: zliang5@nd.edu

Zhiding Liang
Graduate Assistant in Sustainable Computing Laboratory (SCL)

- Education:
  - PhD student, Computer Science and Engineering, University of Notre Dame (2021-)
  - BS, Electrical Engineering, University of Wisconsin, Madison (2020)
- Supervisor:
  - Prof. Yiyu Shi (University of Notre Dame)
  - Prof. Weiwen Jiang (George Mason University)
- Research Interests:
  - Quantum Machine Learning
  - Quantum Compiler

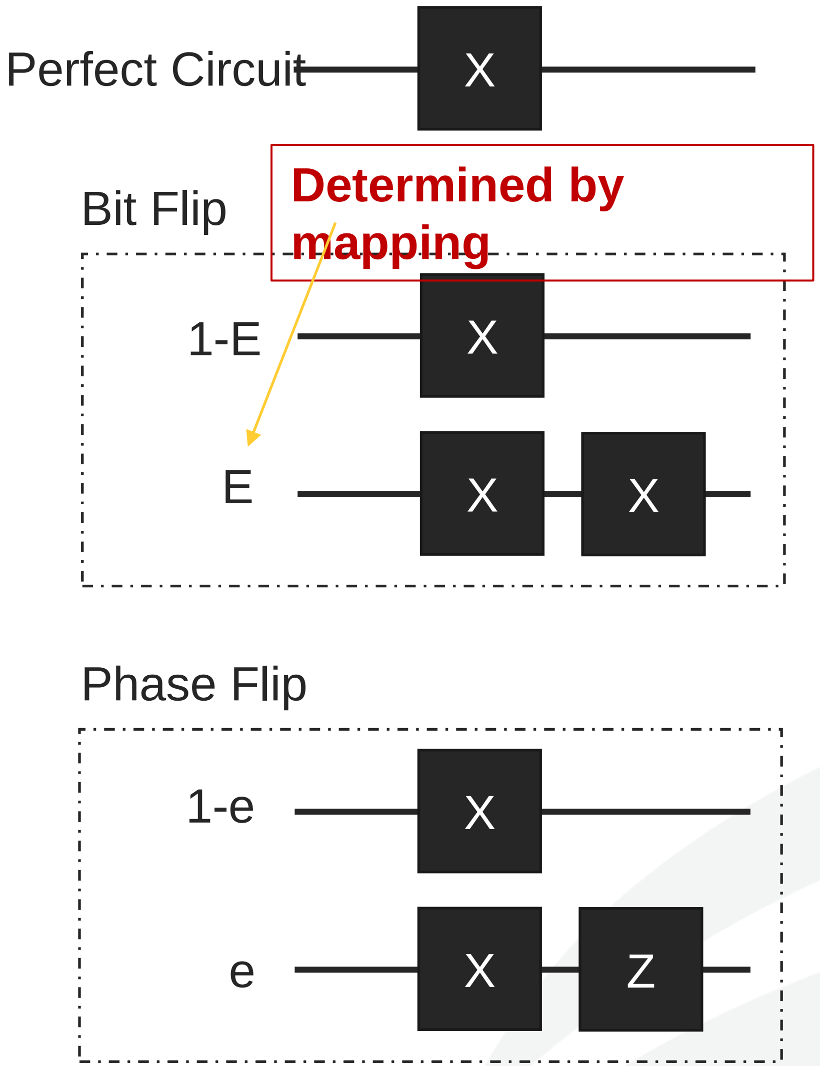
Quantum Errors

Quantum devices have high error rate



(e) Guadalupe

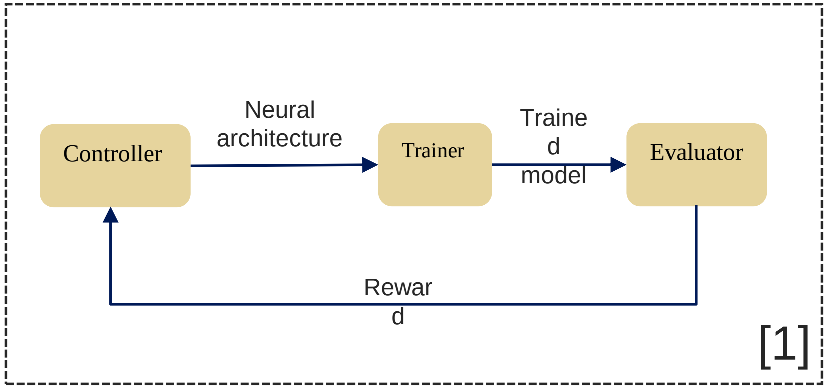
Error rate on a bit in CMOS Device error rate is about 10^{-15}
But error rate on a quantum bit reaches 10^{-4} to 10^{-2}



Why Error Need to Be Learned in Quantum Neural Network?

Noise Model	Affected Gates	Error Rate	Accuracy	Simulation Time （per image）
No Error	0	-	98.04%	5.00s
Bit Flip Error	X, CX, CCX	0.1	53.30%	568.50s
Bit Flip Error	X, CX, CCX	0.01	88.24%	540.00s
Phase Flip Error	Z, CZ	0.1	64.29%	545.00s
Phase Flip Error	Z, CZ	0.01	91.67%	511.14s
Bit+Phase	X,CX,CCX,Z,CZ	0.1	45.10%	628.00s
Bit+Phase	X,CX,CCX,Z,CZ	0.01	77.78%	532.04s

- Noise can significantly affect the performance of Quantum Neural Network.

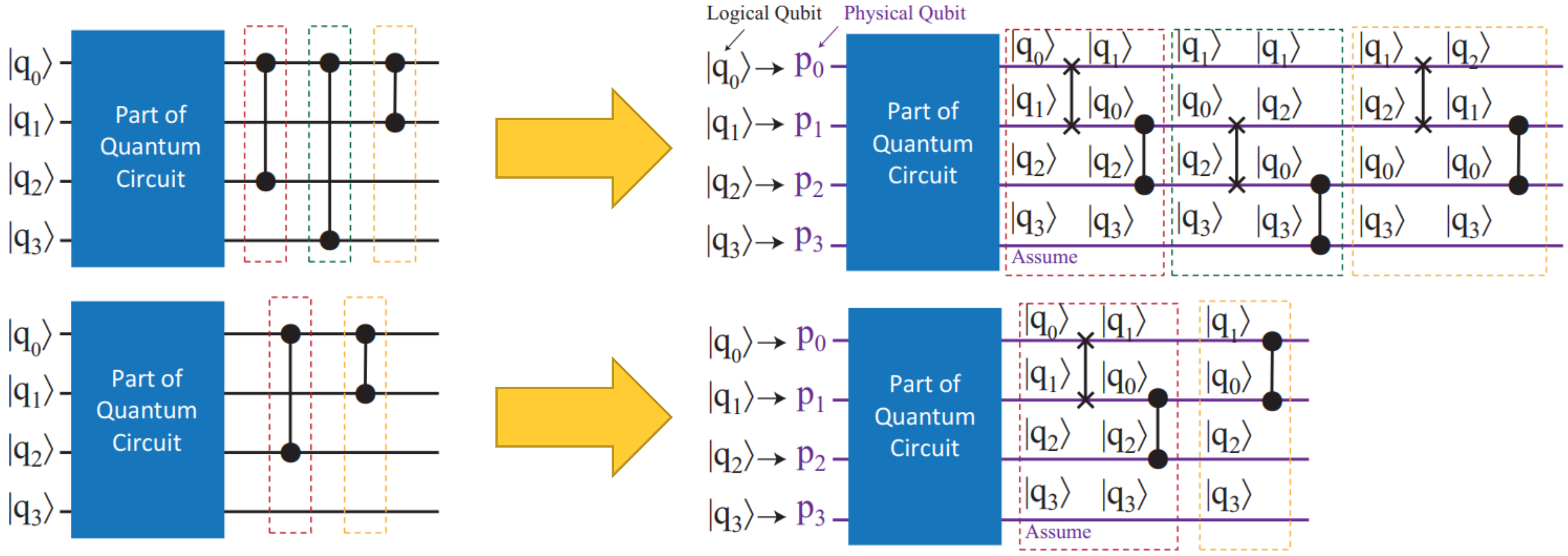


- Error can be learned into Classical Neural Network.

[1] Zheyu Yan, Da-Cheng Juan, X. Sharon Hu and Yiyu Shi, "Uncertainty Modeling of Emerging Device based Computing-in-Memory Neural Accelerators with Application to Neural Architecture Search," *in Proc. of the Asia and South Pacific Design Automation Conference* (ASP-DAC), 2021

Challenges in learning error rate into quantum neural network

Challenges to Learn Error in Quantum Neural Network?



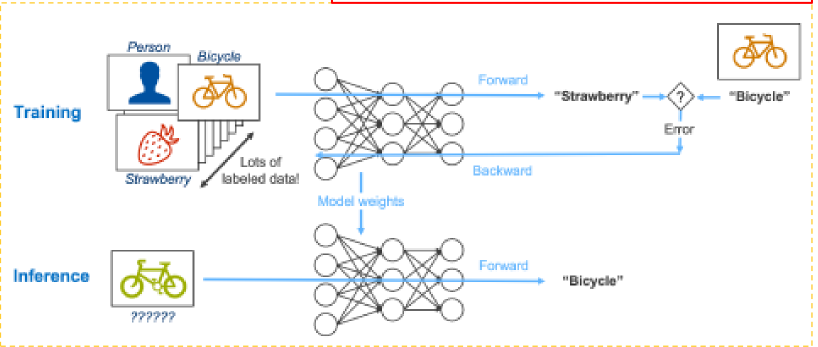
Challenges:

1. Error unpredictable on the quantum circuit. We need to fix the logical-physical qubits mapping.

Challenges to Learn Error in Quantum Neural Network?

$$Time = \quad Training\ time \quad + \quad Compiling\ Time$$

Training Neural Network



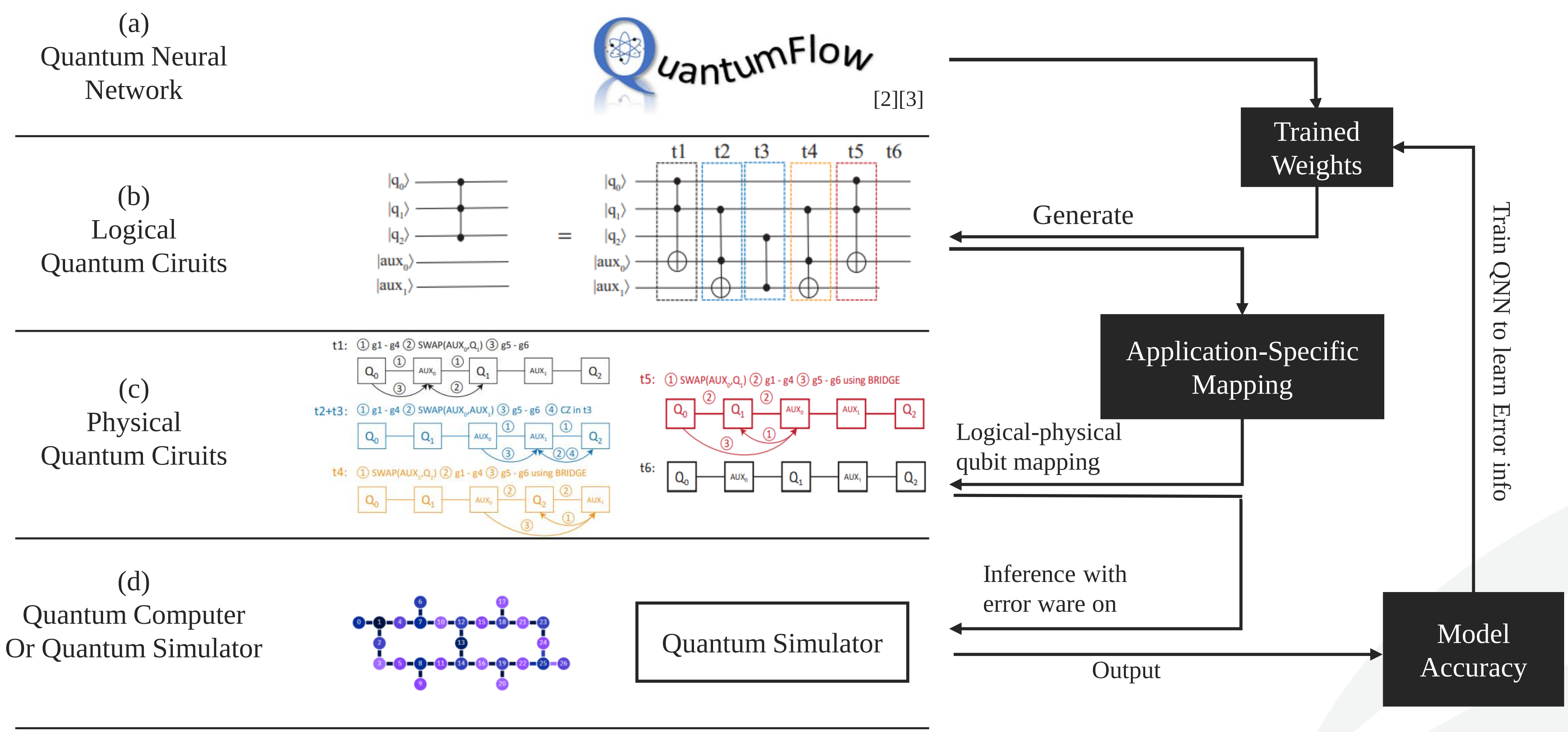
Time Consuming!

Existing Quantum Compiler

Challenges:

1. Error unpredictable on the quantum circuit. We need to fix the logical-physical qubits mapping.
2. Existing compiler may not for training, because they are always time-consuming. We need to build a compiler with faster compiling speed.

Quantum-error-aware Training Framework



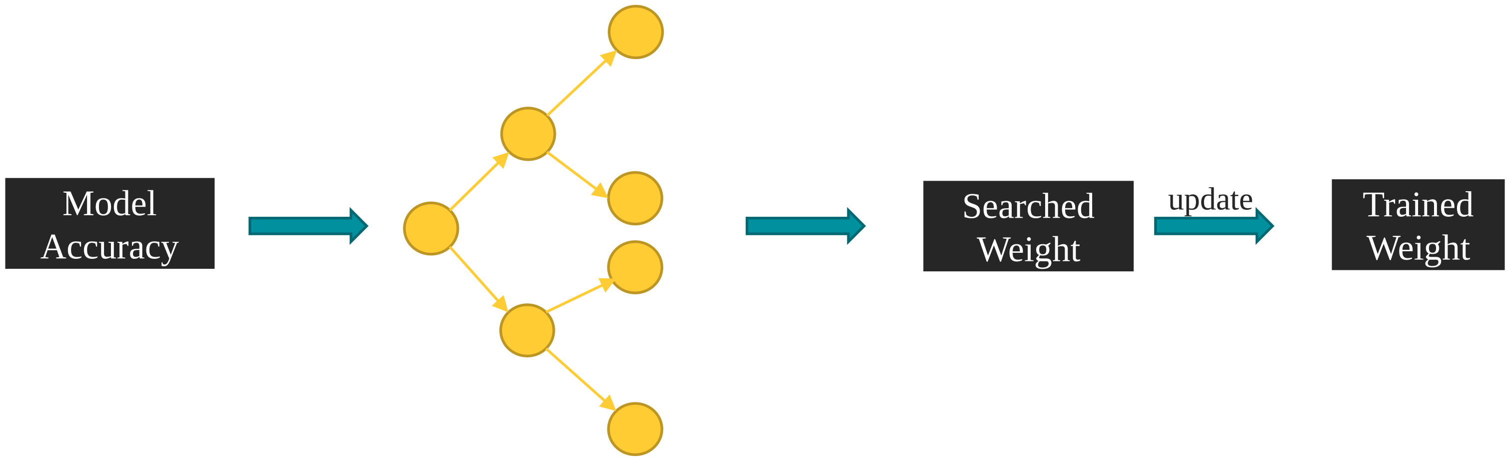
[2] Weiwen Jiang, Jinjun Xiong, and Yiyu Shi. "A codesign framework of neural networks and quantum circuits towards quantum advantage". In: *Nature communications* 12.1 (2021), pp. 1–13.
[3] Weiwen Jiang, Jinjun Xiong, and Yiyu Shi. "When Machine Learning Meets Quantum Computers: A Case Study". In: *2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE. 2021, pp. 593–598.

Train QNN to learn error info

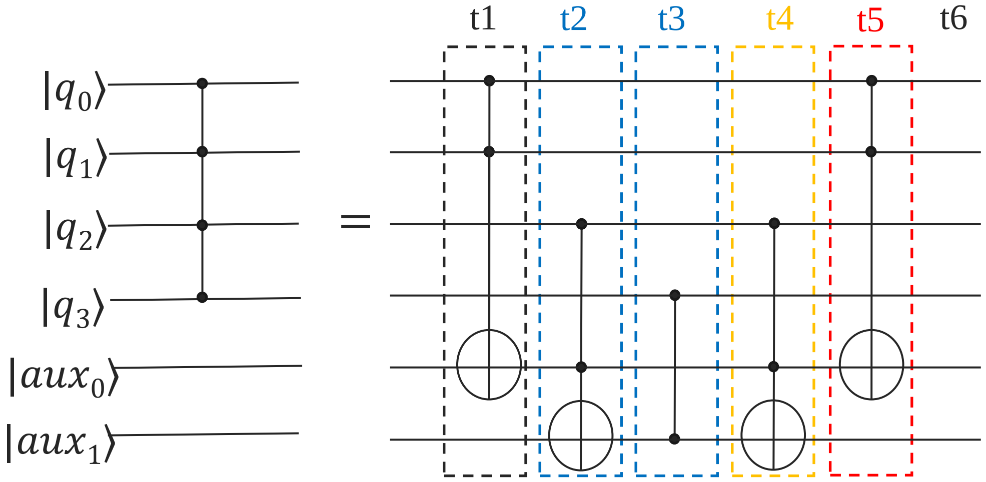
The quantum-error-aware training framework can be better to demonstrate as follow equation:

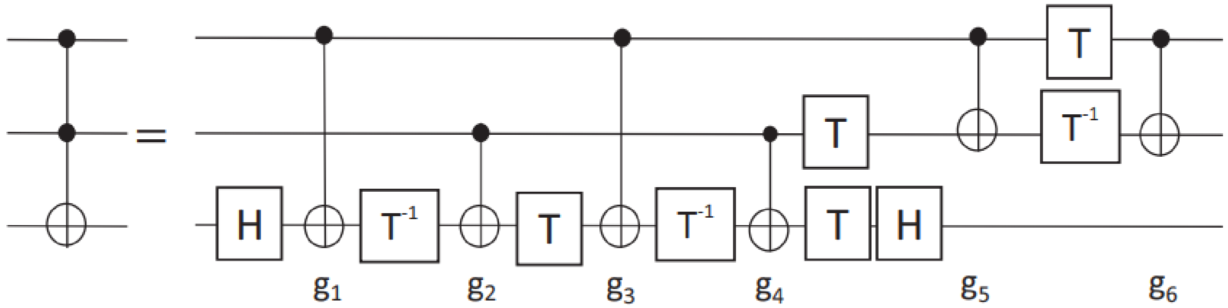
$$c_i = circ(W_i)$$
$$m_i = Map(c_i, PhyQ)$$
$$e_i = Error(m\_i)$$
$$a_i = Inference(m_i, e_i)$$

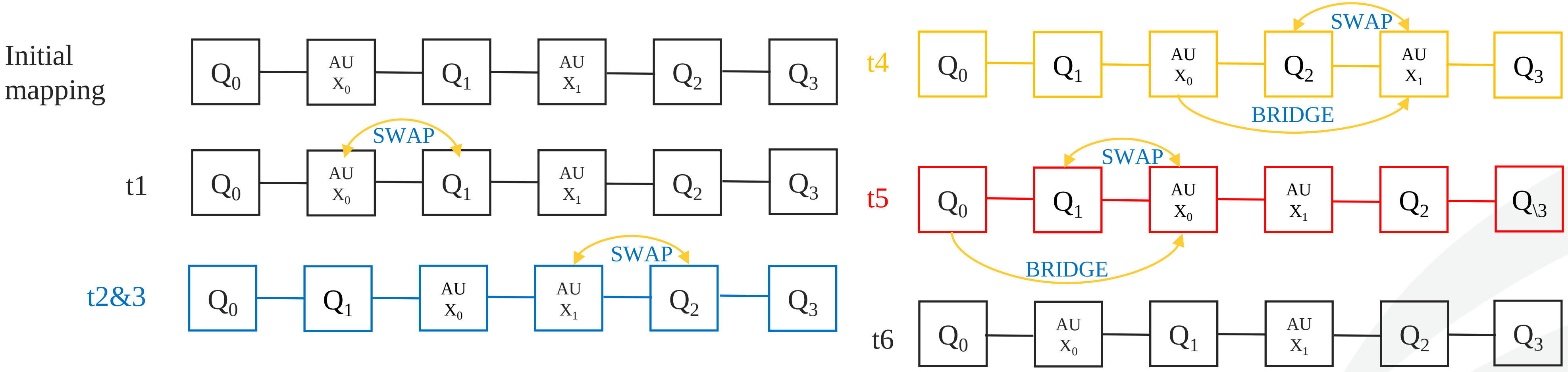
- c_i is the logical quantum circuit generated by one identified weight W_i in the i-th iteration.
- m_i is represented the c_i mapping to physical qubits.
- e_i is the error model based on m_i.
- a_i is accuracy of the QNN with W_i ,which is executing by the physical quantum circuit m_i with error model e_i .



Application-Specific Mapping







QF-RobustNN Result

QF-RobustNN Results on IBM Quantum Simulator

Error Rate	Baseline	QF-RobustNN	
	Acc.	Acc.	Weight
0 (Perfect)	94%	-	[-1,-1,1,1,1,1,1,1], [1,1,1,1,1,1,1,1]
0.0001	84%	84%	[-1,-1,1,1,1,1,1,1], [1,1,1,1,1,1,1,1]
0.0005	75%	76%	[1,1,1,-1,-1,-1,-1,-1], [-1,-1,-1,1,1,-1,-1,-1]
0.001	74%	76%	[1,1,-1,1,1,-1,-1,1], [-1,1,1,1,-1,1,1,1]
0.01	75%	80%	[-1,-1,1,-1,1,-1,1,1], [1,1,1,-1,-1,1,-1,1]
0.05	49%	75%	[1,1,-1,-1,1,-1,1,1], [-1,1,-1,-1,-1,-1,-1,1]
0.1	47%	75%	[1,1,-1,-1,1,-1,1,-1], [1,1,-1,-1,1,-1,1,-1]

- Baseline model has the initial weight [-1,-1,1,1,1,1,1,1], [1,1,1,1,1,1,1,1] , which has the best performance on perfect environment.
- Here is an observation that the QF-RobustNN push the improvement on accuracy as the error rate become larger.

Comparison of Compilers Elapsed Time Results on IBM Q Montreal

Compiler Name	Circuit Complexity Level		
	Simple	Middle	Complex
QUEKO [4]	484.755s	5332.903s	Over 10 hours
HA [5]	4.765s	4.696s	4.201s
OURS	0.008s	0.015s	0.020s
Imp. (vs QUEKO)	60594.38×	355526.87×	>1800000×
Imp. (vs HA)	595.63×	313.07×	210.05×

- Apply two existing compilers to be the baseline for efficiency comparison.
  - (1) Quantum Mapping Examples with Known Optimal (QUEKO) [5]
  - (2) Heuristic-based Hardware-Aware mapping algorithm (HA) [6]
- Results demonstrate our application-specific compiler can be efficiently integrated into the training framework.

[4] Bochen Tan and Jason Cong. "Optimal layout synthesis for quantum computing". In: *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE. 2020, pp. 1–9.
[5] Siyuan Niu et al. "A Hardware-Aware Heuristic for the Qubit Mapping Problem in the NISQ Era". In: *IEEE Transactions on Quantum Engineering 1 (2020),* pp. 1– 14.

Comparison of Compilers Results on IBM Q Montreal

Compiler Name	Model	Accuracy	Extra SWAP gate	Elapsed Time
OURS	baseline	74%	15	0.020s
HA	QF-RobustNN	75%	43	3.955s
OURS	QF-RobustNN	80%	12	0.014s

- The comparison of compilers results demonstrate the efficiency advantage that QF-RobustNN has.
  - Higher accuracy
  - Less Extra gate cost
  - Shorter elapsed time.

Thank you!