Tutorial on QuantumFlow+VACSEN: A Visualization System for Quantum Neural Networks on Noisy Quantum Devices

Session 3: Quantum Neural Network Compression

Zhepeng Wang
Ph.D. Student
Electrical and Computer Engineering
George Mason University
zwang48@gmu.edu
https://jqub.ece.gmu.edu

How to Compress a Quantum Neural Network?

Quantum Neural Network Compression

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

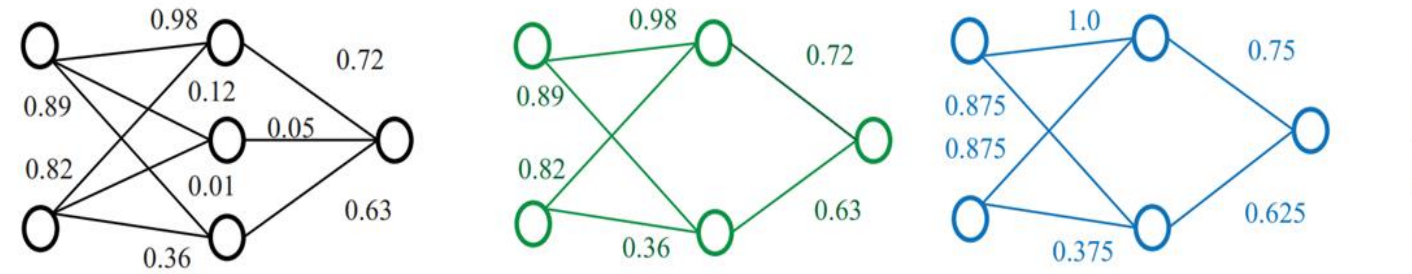
Zhirui Hu, Zhepeng Wang (Presenter), Dr. Weiwen Jiang

Department of Electrical and Computer Engineering

JQub @ George Mason University

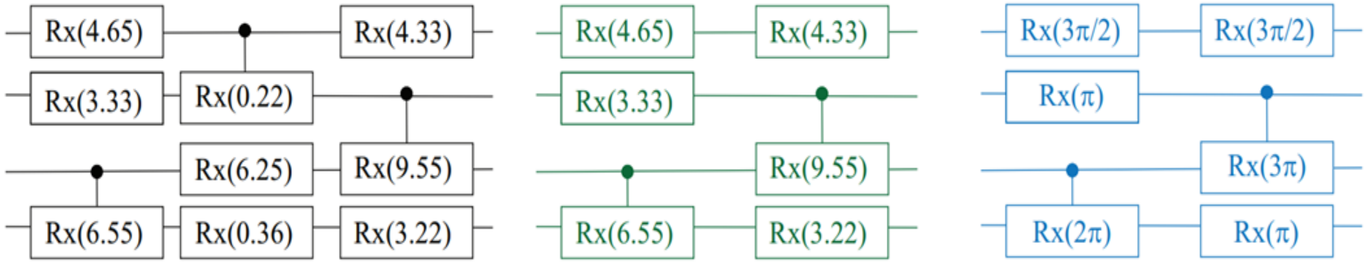
Motivation and Background

- Pruning and Quantization in Classical ML



(a) Non-Compression Classical NN (b) Classical NN with Pruning (c) Pruned NN with Quantization

- Pruning and Quantization in Quantum ML



(e) Non-Compression QNN (f) QNN with Pruning (g) Pruned QNN with Quantization

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

Motivation and Background

- Quantum Neural Network Compression Should be Compilation Aware

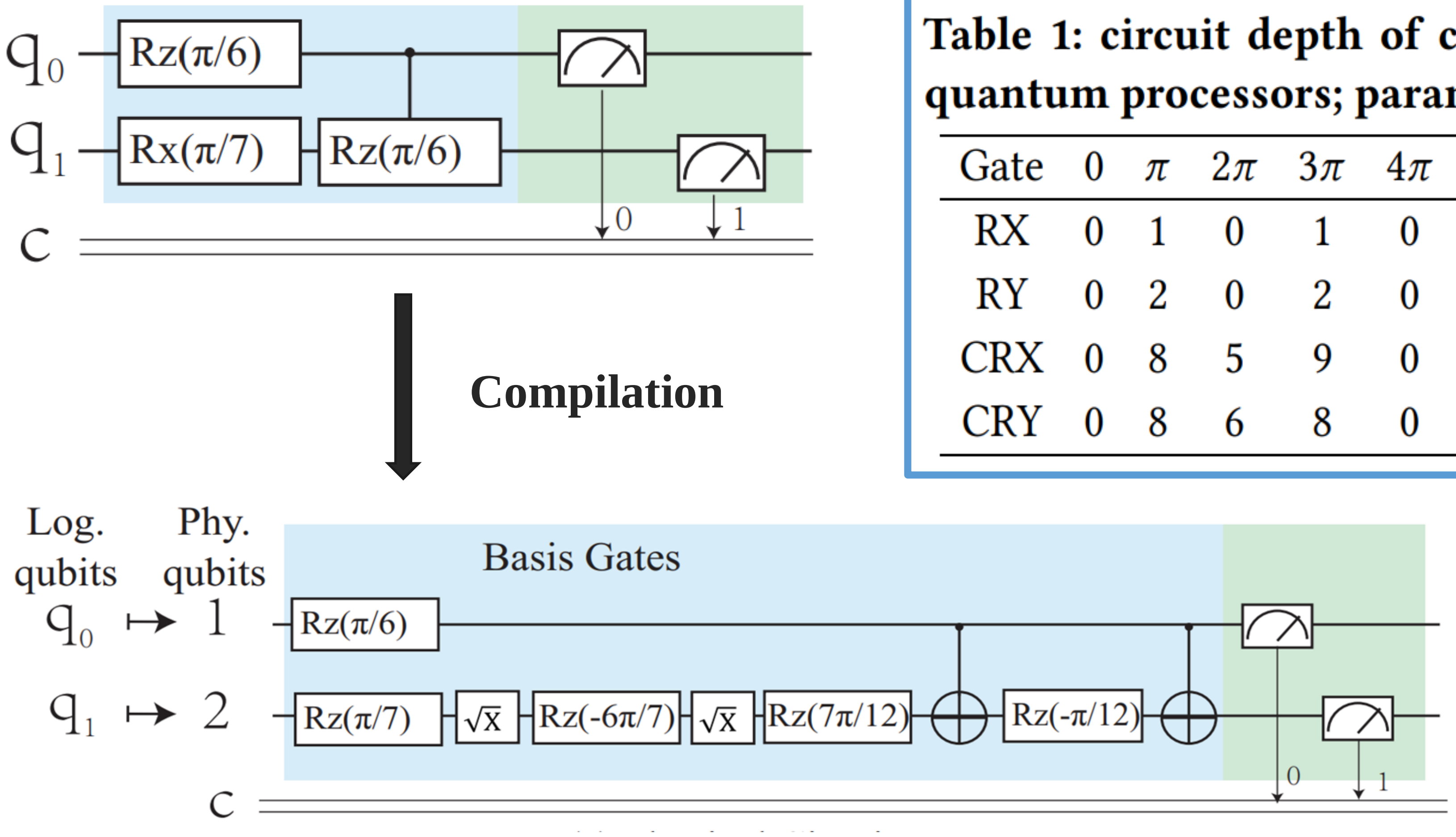


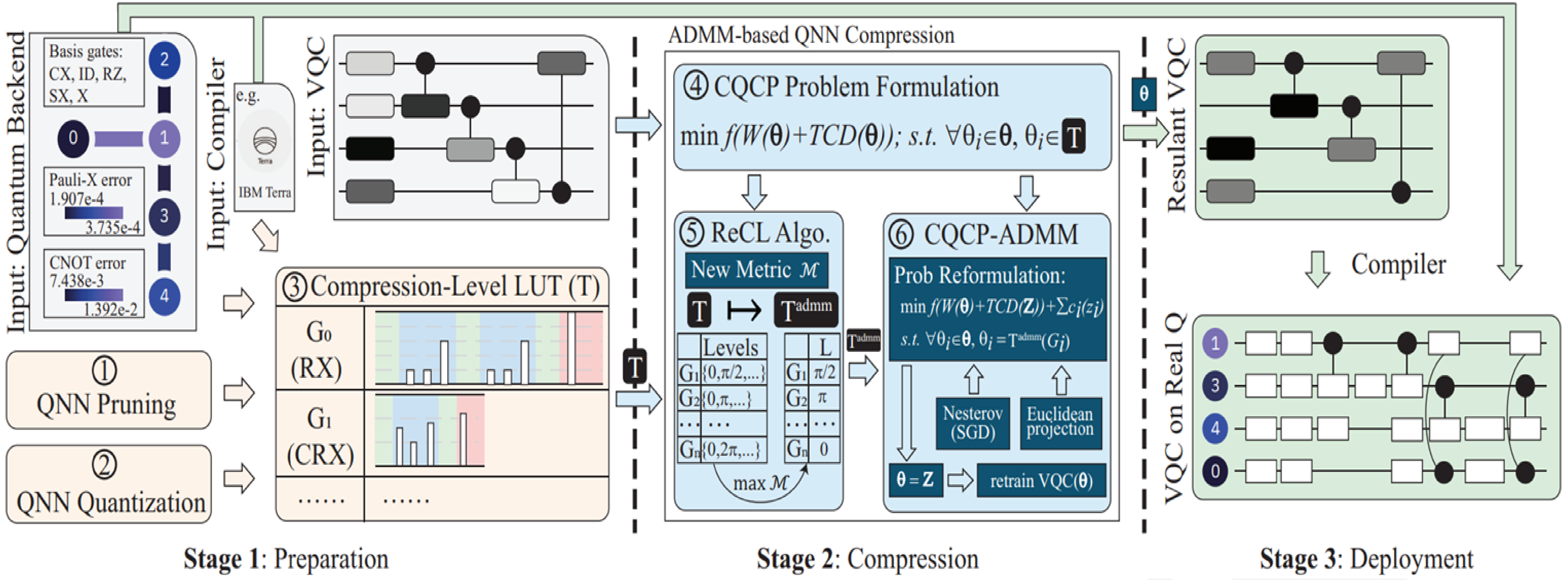
Table 1: circuit depth of compiled quantum gates on IBM quantum processors; parameters are in the range of $[0, 4\pi]$

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

- General Overview

Three stages: 1. Preparation; 2. Compression; 3. Deployment

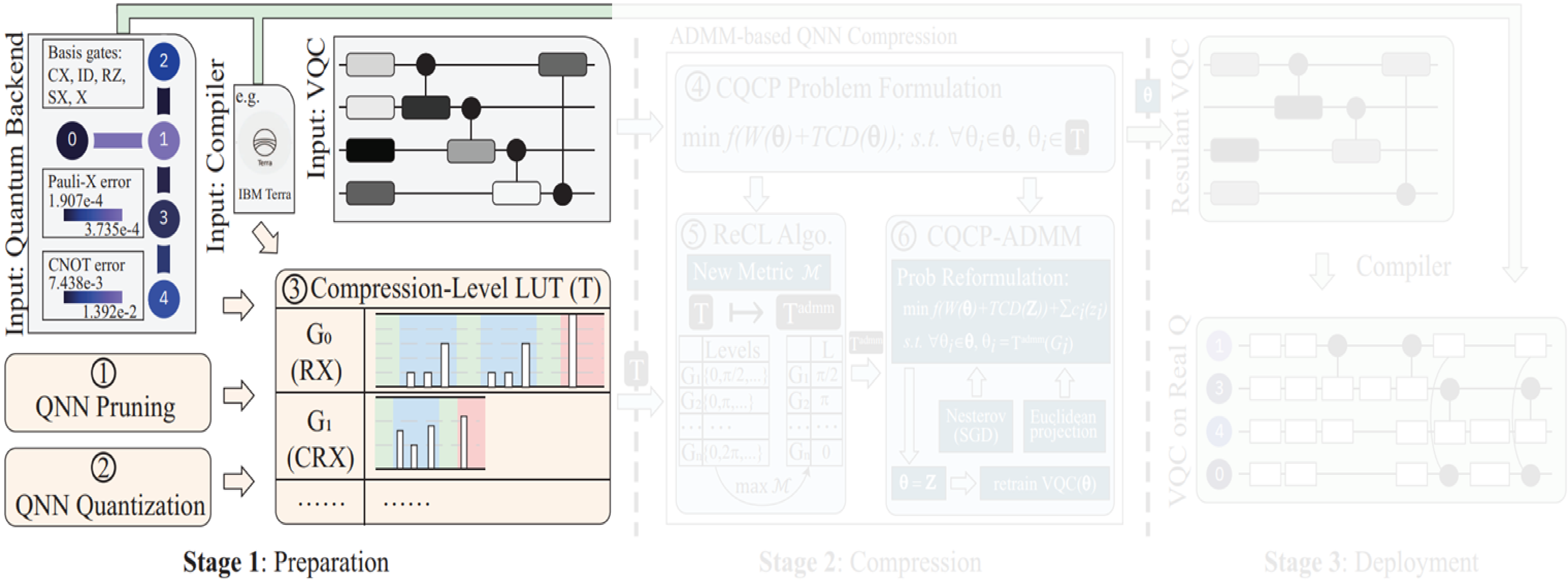


CompVQC

- LUT Construction and Training a Quantum Model
- Reconstruct LUT for ADMM
- Compression based on ADMM
- Deployment

CompVQC

- LUT Construction and Training a Quantum Model



CompVQC

- LUT Construction and Training a Quantum Model

❑ Compression-Level Lookup Table (LUT)

A combination of pruning/quantization level called as “compression level”.

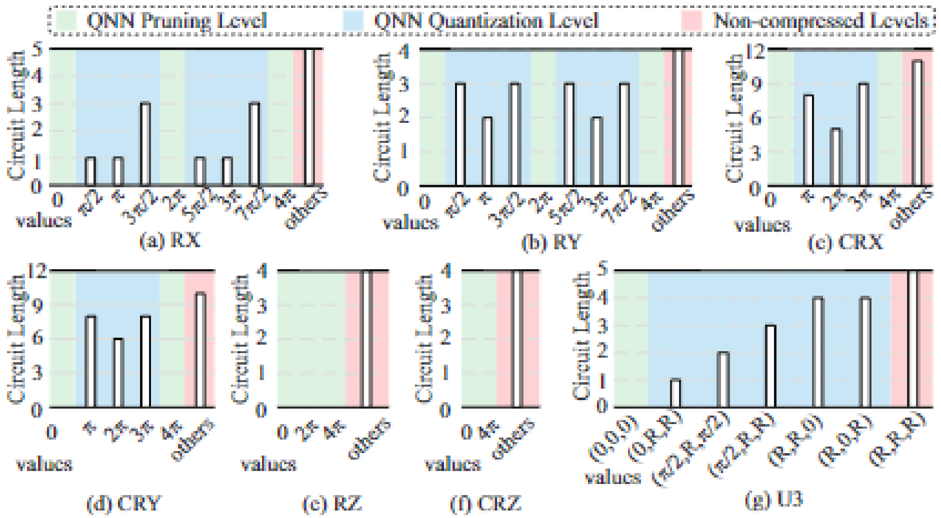


Table 1: circuit depth of compiled quantum gates on IBM quantum processors; parameters are in the range of $[0, 4\pi]$

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

❑ VQC Pre-Training

A VQC model is pre-trained for compression and the training process is implemented with **Torch Quantum**.

Hands-On Tutorial (1) : LUT Construction

Input

- Fixing points list
- Logical Gates List to be used
- Quantum Backend

```
#Input
test_fixing_points = [math.pi*4,math.pi*2,math.pi,math.pi*3,math.pi/2,
                      math.pi/2*5,math.pi/2*7,math.pi/2*3,math.pi/6]
logical_gates = ['rx','ry','rz','crx','cry','crz']
backend = FakeValencia()

#api
df = LUT_construction(test_fixing_points,logical_gates,backend)
```

Do

- Get the compiler for the backend
- Get the compiled circuit length of each logical gate at each special fixing points

fixing_points	rx	ry	rz	crx	cry	crz
12.57	0	0	0	0	0	0
6.28	0	0	0	5	6	4
3.14	1	2	1	8	8	4
9.42	1	2	1	9	8	4
1.57	1	3	1	11	10	4
7.85	1	3	1	11	10	4
11.00	3	3	1	11	10	4
4.71	3	3	1	11	10	4
0.52	5	4	1	11	10	4

Output

- Get the compiler for the backend

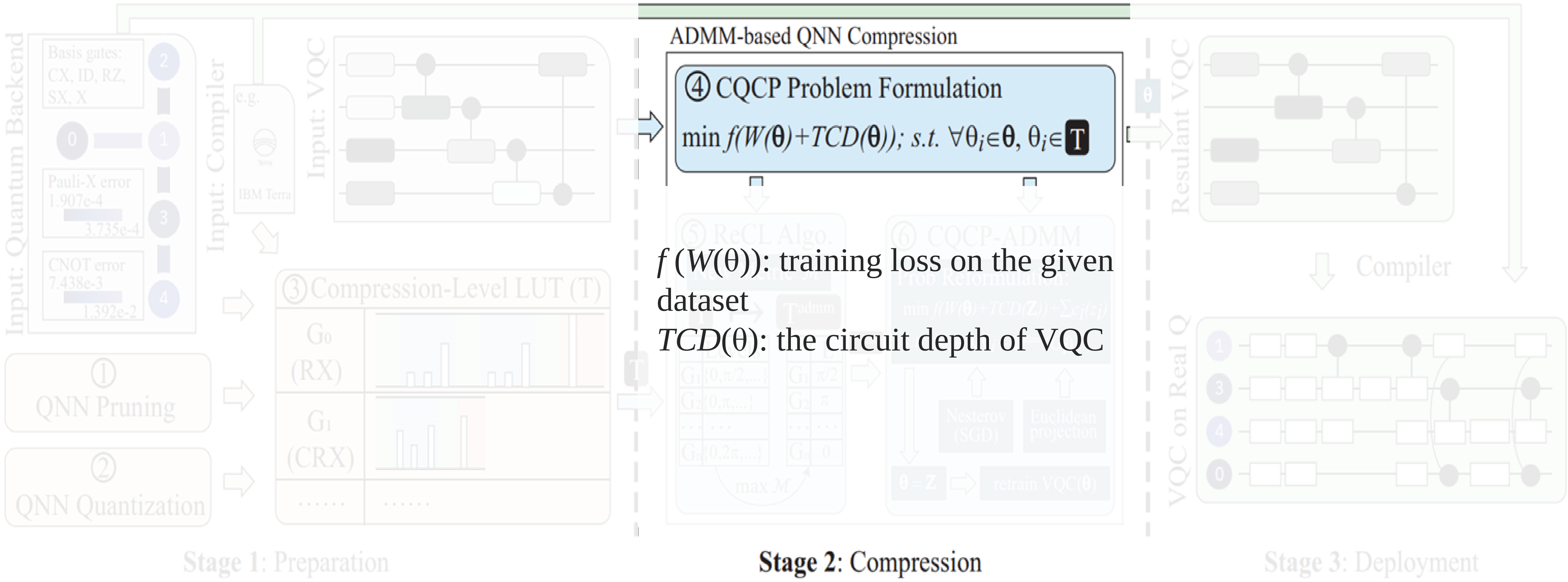
CompVQC

- LUT Construction and Training a Quantum Model
- Reconstruct LUT for ADMM
- Compression based on ADMM
- Deployment

CompVQC

- Problem Definition

Given VQC **W(θ)**, LUT ***T***, quantum compiler **C**, the problem is to determine trainable parameters **θ**, such that:



$f(W(\theta))$: training loss on the given dataset

$TCD(\theta)$: the circuit depth of VQC

CompVQC

- Reconstruction LUT for ADMM

Process is conducted by traversing all quantum gates in VQC and select the compression target with highest metric.

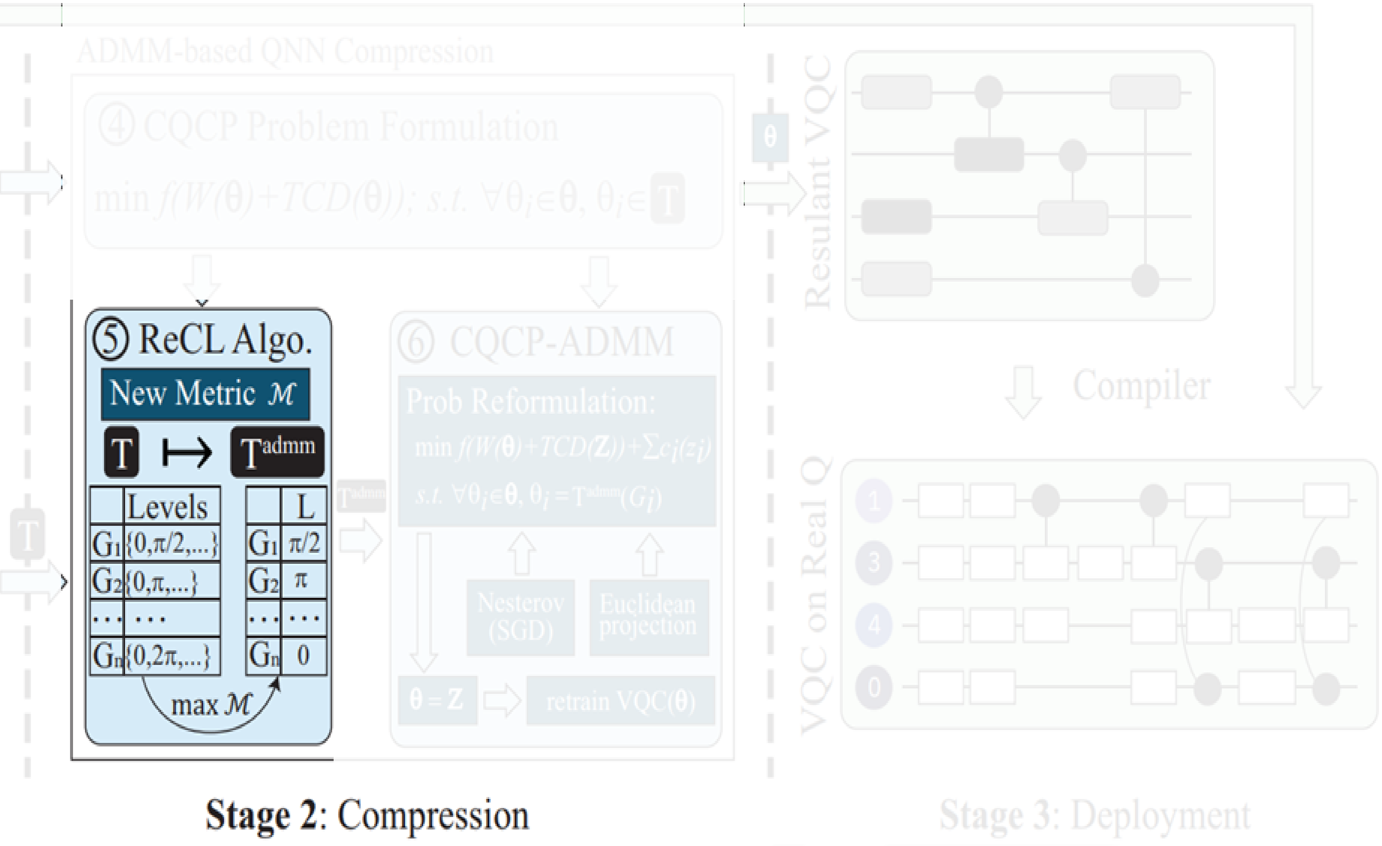
A heuristic metric for the choice

$$\mathcal{M}(\theta, G_i(\gamma_{i,k})) = acc(W(\theta^{i,k})) \cdot \tau(\theta^{i,k}, \theta)$$

$$\tau(\theta^{i,k}, \theta) = \frac{TCD(\theta)}{TCD(\theta^{i,k})}$$

$acc(W(\theta^{i,k}))$: the accuracy of the VQC under new parameters

$TCD(\theta)$: the inverse of the compression ratio by changing parameters from θ to $\theta^{i,k}$



Hands-On Tutorial (2) : Reconstruct LUT for ADMM

Input

- trained model
- Original LUT
- The metrics function of accuracy and length

```
#input
model = torch.load('model.pth')
lut = pd.read_csv('lut.csv')
def metrics_func(acc,depth):
    return acc+1.0/depth
backend = FakeValencia()
```

For each parameter, Do

- Replace it with points at compression level in original LUT while fixing other parameters
- Calculate the metrics of each new model
- Select the point with the highest metric as the compression level for ADMM

```
#api
new_lut = LUT_reconstrution(model,lut,backend,metrics_func)
```

Output

```
[ 7.85  1.57 11.    0.    3.14  1.57  3.14  3.14  0.    0.    0.    0.
  0.    0.    0.    0.    0.    0.  ]
```

- A new LUT for ADMM

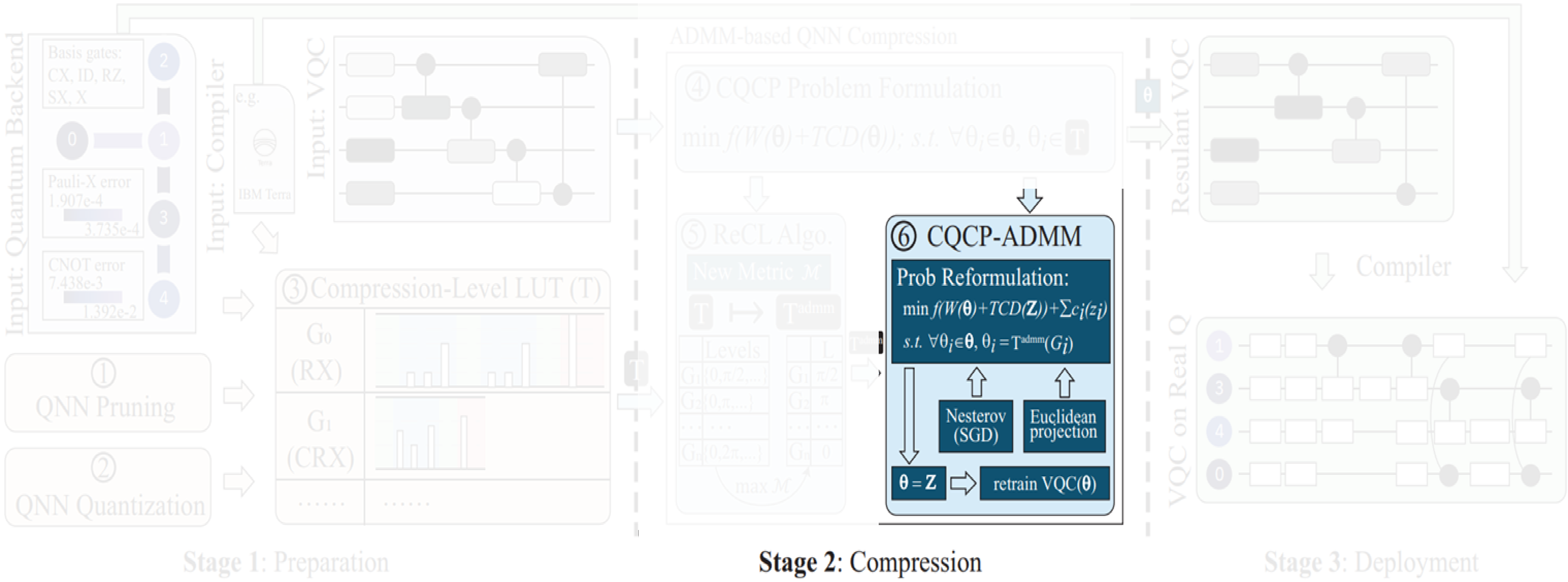
CompVQC

- LUT Construction and Training a Quantum Model
- Reconstruct LUT for ADMM
- Compression based on ADMM
- Deployment

CompVQC

- Compression based on ADMM

Each parameter can either be compressed to the target value in T^{admm} or not compressed.



CompVQC

- Compression based on ADMM

Given reconstructed compression-level LUT T^{admm}, the CQCP is formulated as:

$$\min_{\{\theta_i\}} \quad f(W(\theta)) + TCD(Z) + \sum_{\forall z_i \in Z} c_i(Z_i),$$
$$s.t. \qquad \forall \theta_i \in \theta, \quad \theta_i = T^{admm}(G_i).$$

Z: a set of auxiliary variables for subproblem decomposition and $z_i \in Z$ is corresponding to $\theta_i \in \theta$

$f(W(\theta)) + TCD(Z)$: the objective function in the original CQCP problem(previously seen).

$$c_i(Z_i) = \begin{cases} 0 & \text{if } \theta_i \in T^{s,r}(G_i), T^{s,r} = T^{admm} \odot mask^r \\ +\infty & \text{if } otherwise. \end{cases}$$

$c_i(Z_i)$: An indicator function to serve as a penalty term

$mask^r$: variable to indicate whether the parameters will be compressed at iteration *r.*

Hands-On Tutorial (3) : Compression based on ADMM

Input

- A trained model
- A new LUT for ADMM

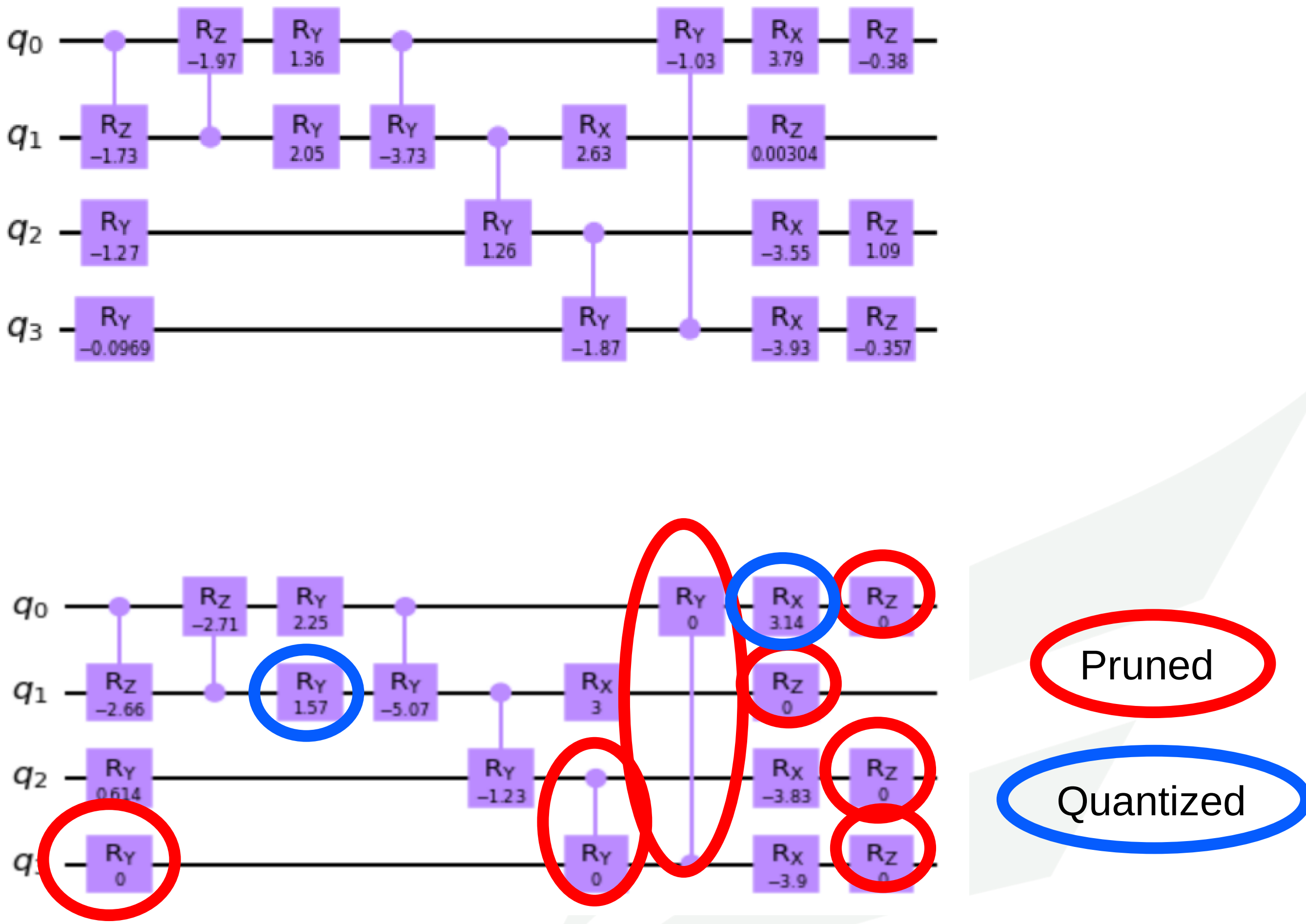
Do

- Compress a model with ADMM
- Fine-tune the compressed model

Output

- A compressed model

	Circuit Length	Accuracy
Original model	51	94.2%
Compressed	35	97.10%

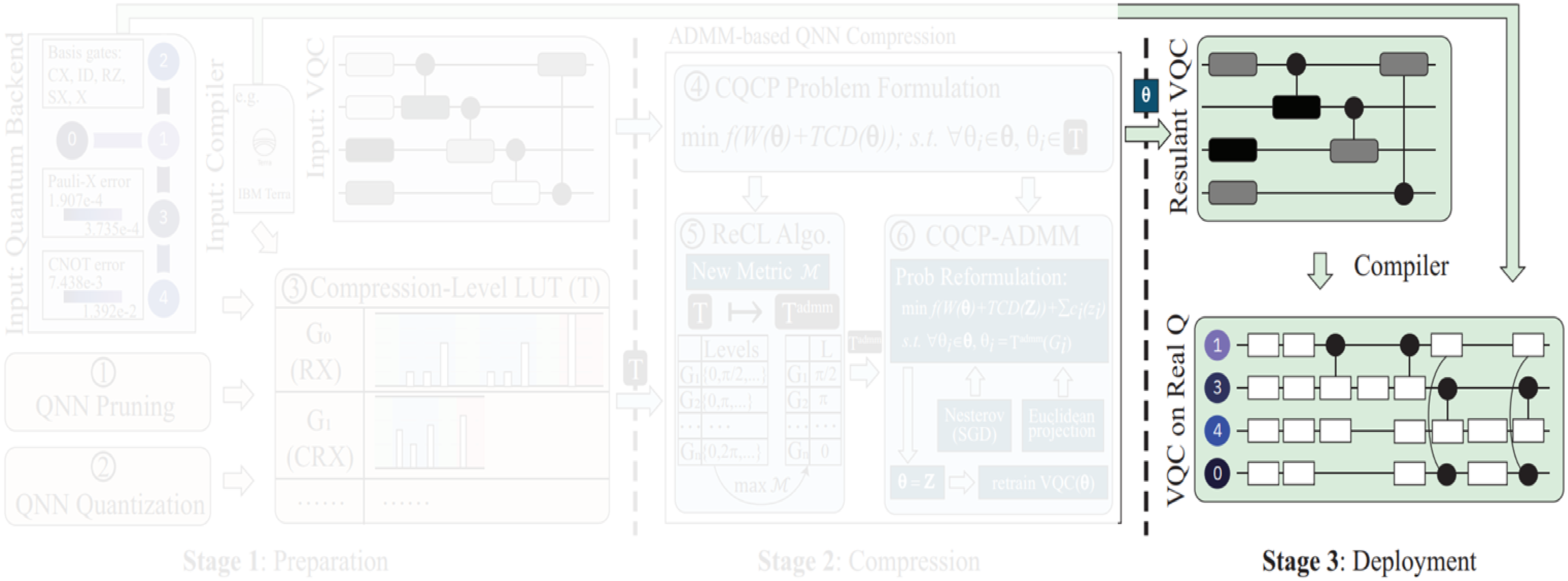


CompVQC

- LUT Construction and Training a Quantum Model
- Reconstruct LUT for ADMM
- Compression based on ADMM
- Deployment

CompVQC

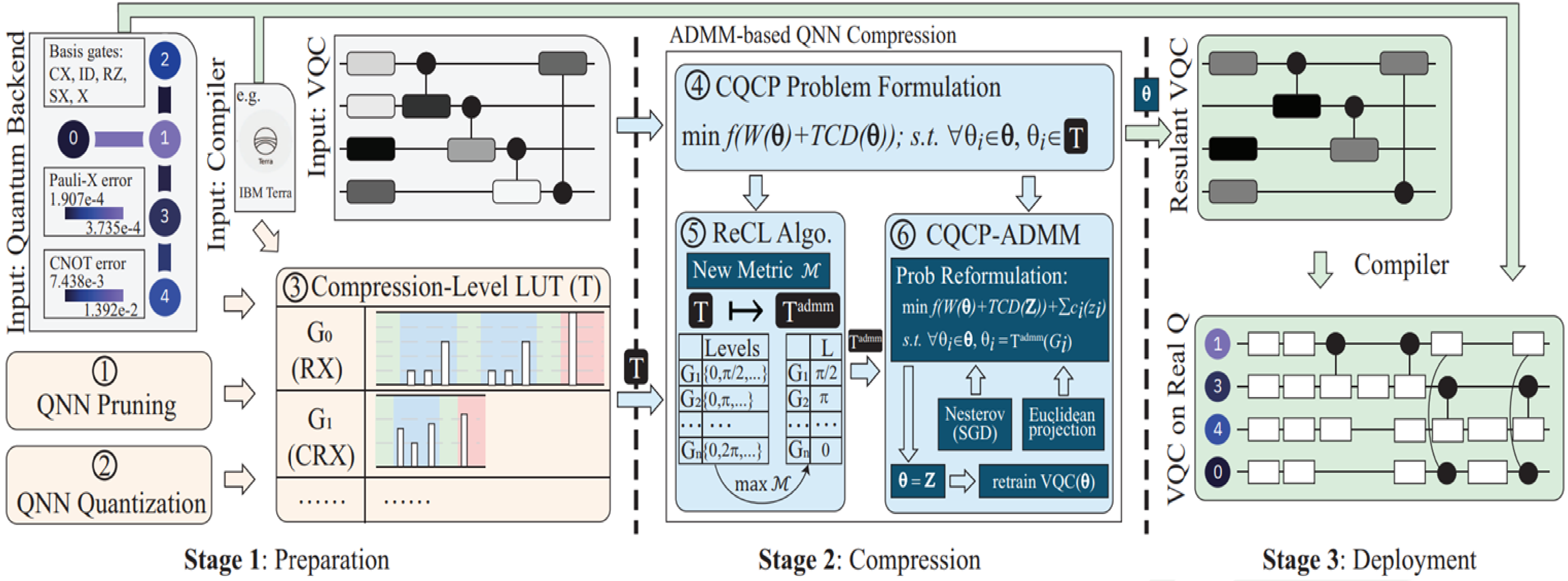
- Deployment



CompVQC

- General Overview

Three stages: 1. Preparation; 2. Compression; 3. Deployment



Hands-On Tutorial (1)
LUT Construction

https://jqub.ece.gmu.edu/categories/QFV/

Hands-On Tutorial (2) Reconstruct LUT for ADMM

https://jqub.ece.gmu.edu/categories/QFV/

Hands-On Tutorial (3) Compression based on ADMM

https://jqub.ece.gmu.edu/categories/QFV/

Experimental Results

- Simulation Results on ML Dataset

CompVQC can maintain high accuracy with <1% accuracy loss. And the reduction of circuit length is up to 2.5X.

Table 2: Comparison among different methods on the accuracy performance and the TCD of the VQC

Compression Method	MNIST-2 Acc. (vs. Baseline)	MNIST-2 TCD (Speedup)	Fashion-MNIST-2 Acc. (vs. Baseline)	Fashion-MNIST-2 TCD (Speedup)
Vanilla VQC	82.74%(0)	121(0)	87.58%(0)	92(0)
Zero-Only-Pruning	80.58%(-2.16%)	70(1.73×)	86.92%(-0.67%)	63(1.46×)
CompVQC-Pruning	81.83%(-0.91%)	74(1.64 ×)	87.41%(-0.17%)	47(1.96×)
CompVQC-Quant	80.99%(-1.75%)	108(1.10×)	86.25%(-1.33%)	74(1.24×)
CompVQC	**81.83%(-0.91%)**	**47(2.57×)**	**87.58%(-0.00%)**	**47(1.96×)**

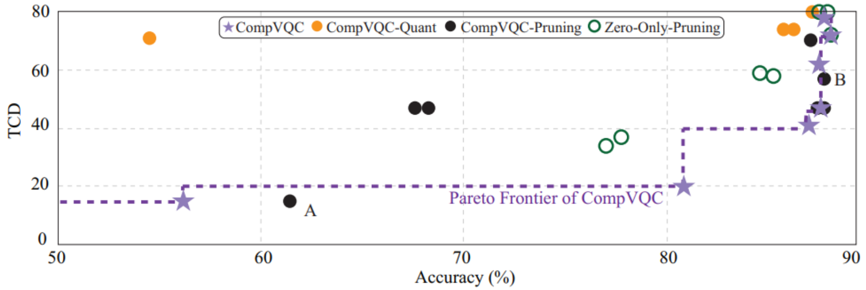


Figure 5: Main results: The Accuracy-Circuit Depth Tradeoff on Fashion-MNIST2

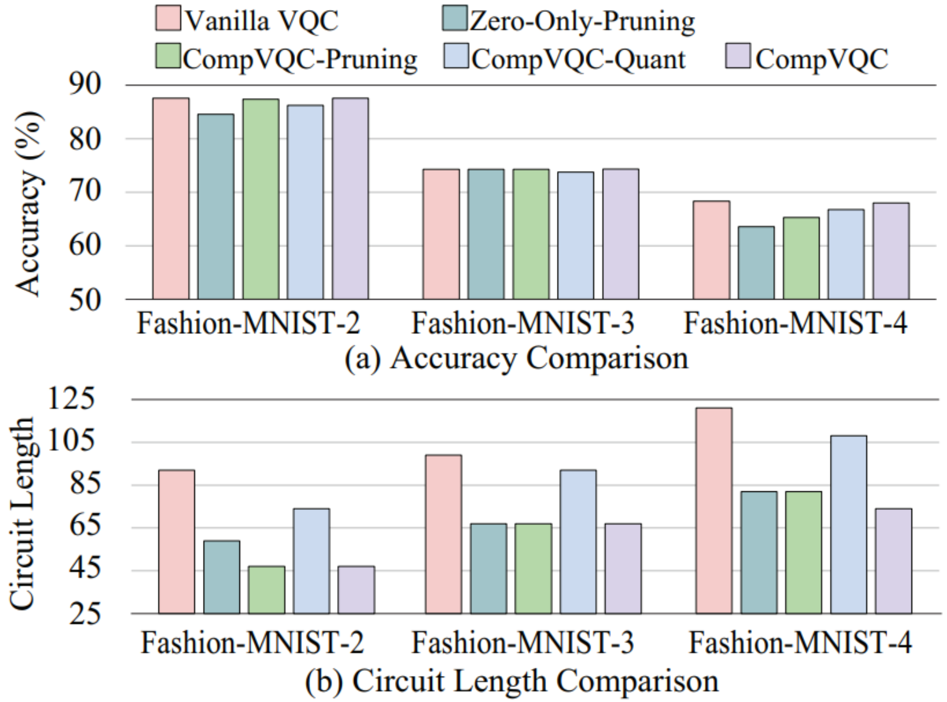


Figure 6: Main Results: CompVQC Scalability on Fashion-MNIST with 2-4 class

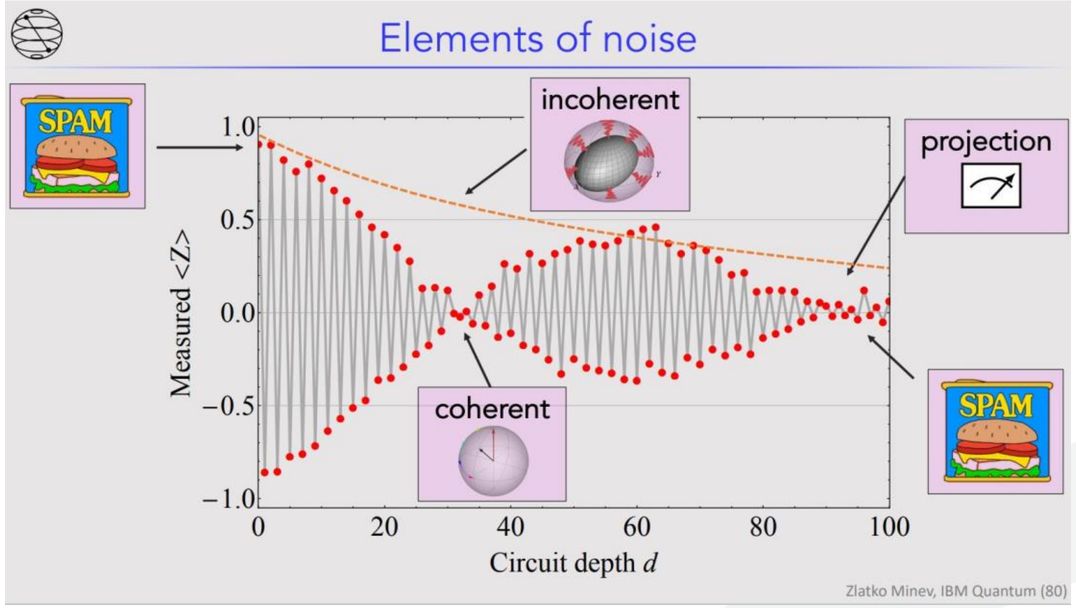
Experimental Results

- Results on Multiple IBM Quantum Computers

CompVQC can reduce circuit length by 2x while the accuracy is also higher in a noisy environment.

Datasets		Syn-Dataset-4		Syn-Dataset-16	
Compression Method		Acc. (vs. Baseline)	TCD (Speedup)	Acc. (vs. Baseline)	TCD (Speedup)
Qiskit Aer	Vanilla VQC	94%(0)	23(0)	96%(0)	51(0)
	Comp-VQC	99%(5%)	11(2.09×)	98%(2%)	23(2.22×)
IBM Q	Vanilla VQC	79%(-15%)	23(1.00×)	86%(-10%)	51(1.00×)
	Comp-VQC	99%(5%)	11(2.09×)	98%(2%)	23(2.22×)

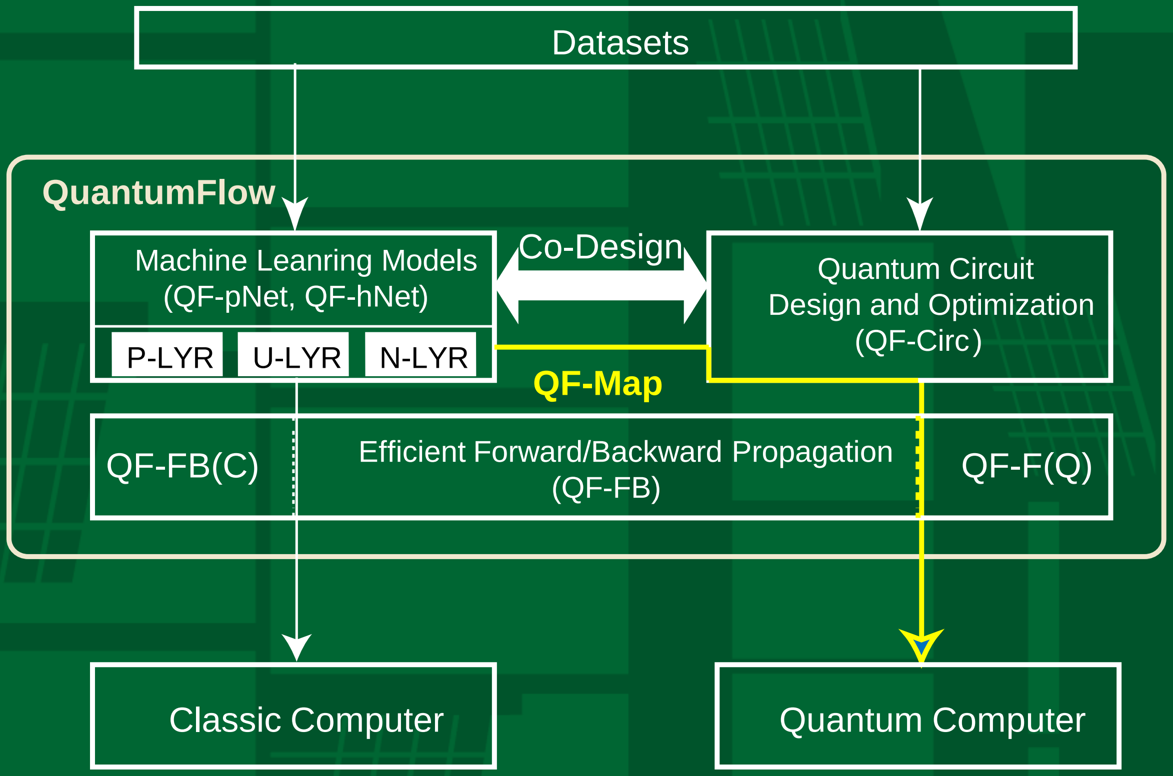
Acc.(vs. Baseline)	ibm_lagos	ibm_perth	ibm_jakarta
Vanilla VQC(TCD=23)	79%(0)	86%(0)	92%(0)
CompVQC(TCD=11)	99%(20%)	98%(12%)	100%(8%)



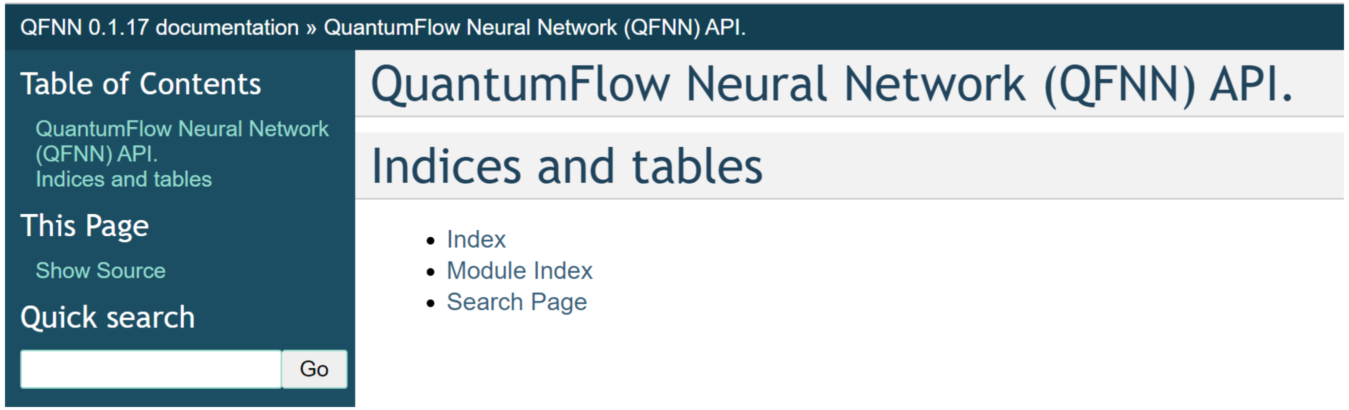
Circuit compression can make the QNN model more robust to the noise

API: QuantumFlow Neural Network (qfnn)

import qfnn



Documentation and Project repo



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

https://github.com/jqub/qfnn

QF-hNet: U-LYR

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

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

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

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

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

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

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

qfnn API Example

QF-hNet

https://jqub.ece.gmu.edu/categories/QFV/

zwang48@gmu.edu

George Mason University

4400 University Drive
Fairfax, Virginia 22030

Tel: (703)993-1000