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

Session 3: Build Quantum Circuit for NN Acceleration using QFNN

Weiwen Jiang, Ph.D.

Assistant Professor

Electrical and Computer Engineering

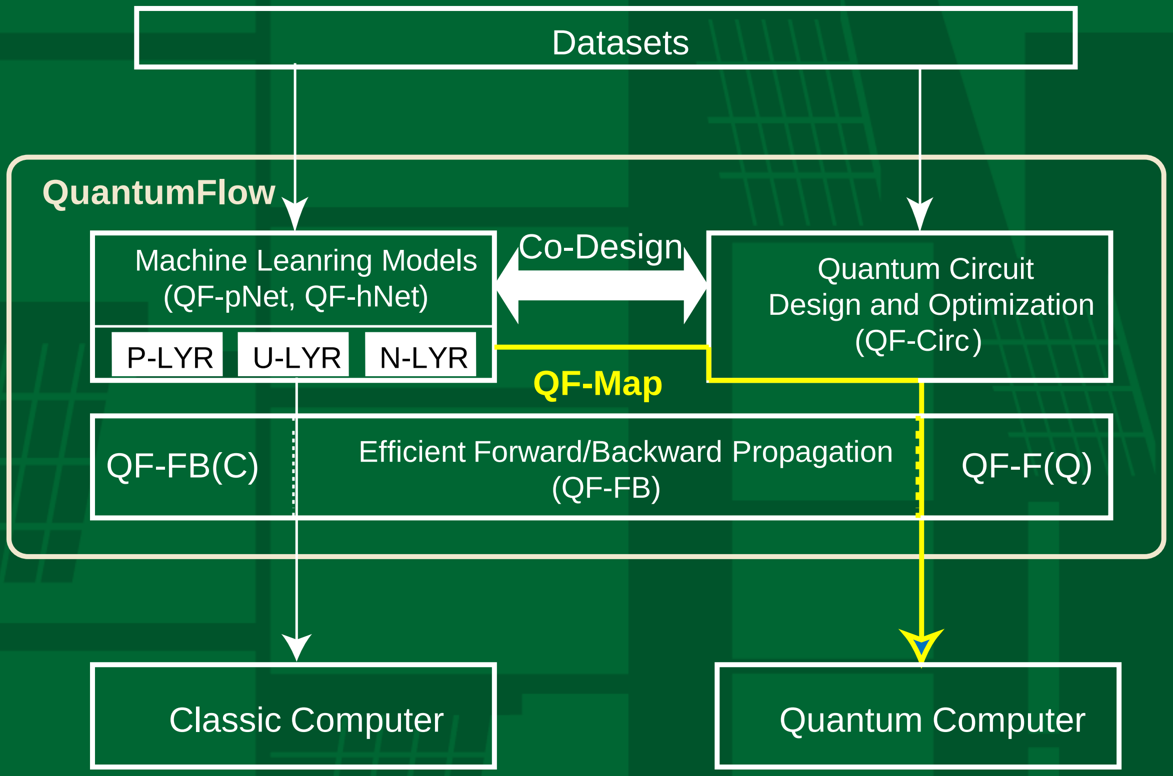
George Mason University

wjiang8@gmu.edu

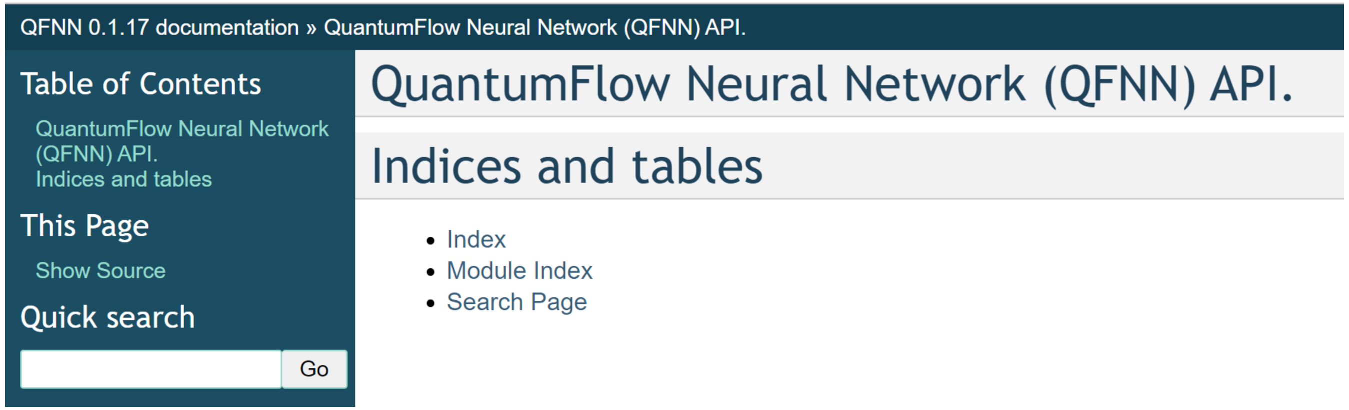
https://jqub.ece.gmu.edu

API: QuantumFlow Neural Network (qfnn)

import qfnn



Documentation and Project repo



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

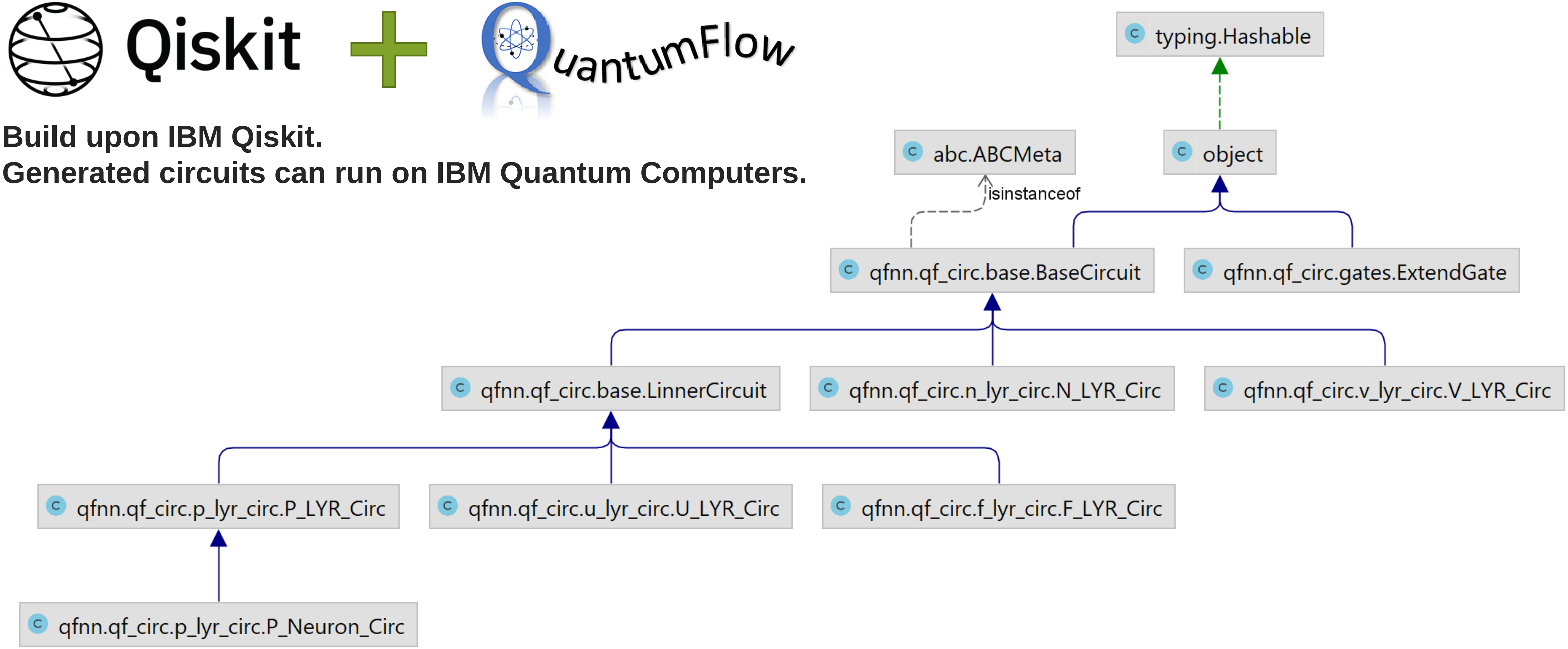
https://github.com/jqub/qfnn

Agenda – Session 3: QFNN API

- **Introduction to QFNN**
  - qf_circ
  - qf_net
  - qf_fb
  - qf_map
- **Building QuantumFlow using QFNN**
- **Beyond QuantumFlow with QFNN**

QF-Circ

Build upon IBM Qiskit.
Generated circuits can run on IBM Quantum Computers.



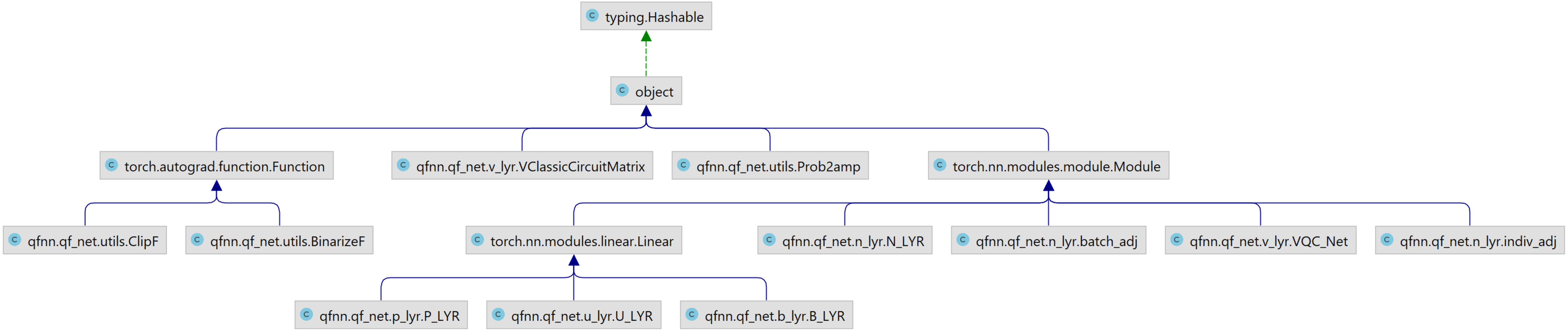
Agenda – Session 3: QFNN API

- **Introduction to QFNN**
  - qf_circ
  - qf_net
  - qf_fb
  - qf_map
- **Building QuantumFlow using QFNN**
- **Beyond QuantumFlow with QFNN**

QF-Net

PyTorch + QuantumFlow

Build upon PyTorch.



Agenda – Session 3: QFNN API

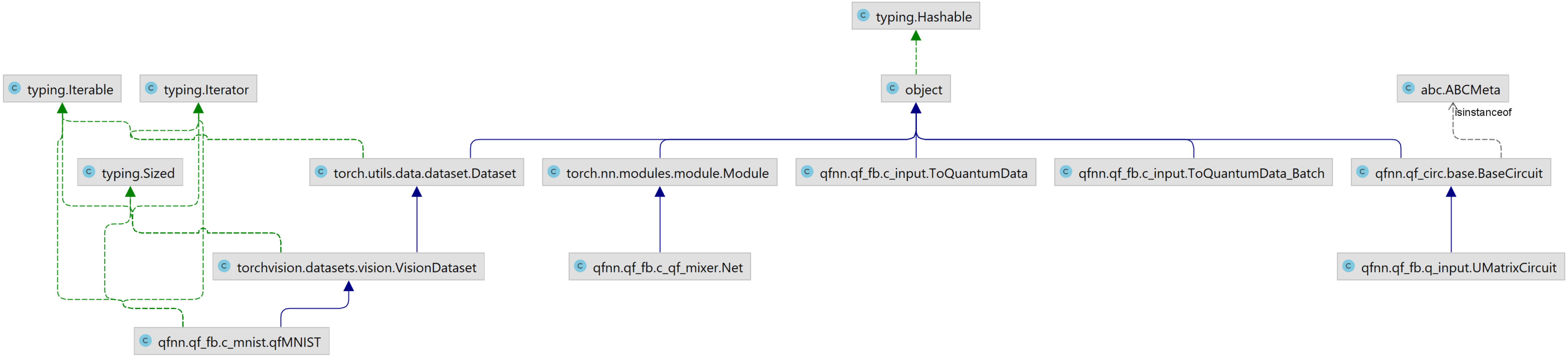
- **Introduction to QFNN**
  - qf_circ
  - qf_net
  - qf_fb
  - qf_map
- **Building QuantumFlow using QFNN**
- **Beyond QuantumFlow with QFNN**

QF-FB

Qiskit + PyTorch + QuantumFlow

Build upon Qiskit and PyTorch.

- Generate network and the generated network can be trained/tested on the PyTorch framework.
- Prepare unitary matrix to translate data from classical to quantum.



Agenda – Session 3: QFNN API

- **Introduction to QFNN**
  - qf_circ
  - qf_net
  - qf_fb
  - qf_map
- **Building QuantumFlow using QFNN**
- **Beyond QuantumFlow with QFNN**

QF-MAP

qfnn.qf_map.u_lyr_map module

```
qfnn.qf_map.u_lyr_map.Mapping_U_LYR(sign, target_num, digits)

qfnn.qf_map.u_lyr_map.change_sign(sign, bin)

qfnn.qf_map.u_lyr_map.find_start(affect_count_table, target_num)

qfnn.qf_map.u_lyr_map.print_info()

qfnn.qf_map.u_lyr_map.recursive_change(direction, start_point, target_num, sign,
affect_count_table, quantum_gates)
```

This module will be further developed to include **Quantum Compiling techniques** for quantum neural networks.

e.g., **QF-RobustNN**

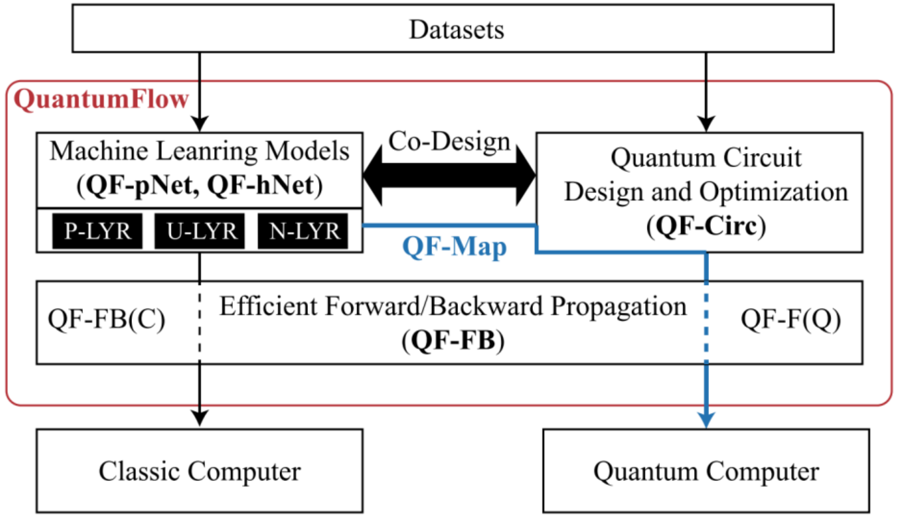
Algorithm 4: QF-Map: weight mapping algorithm

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
}

Agenda – Session 3: QFNN API

- **Introduction to QFNN**
  - Structure: qf_circ, qf_net, qf_fb, qf_map
- **Building QuantumFlow using QFNN**
  - QF-pNet
  - QF-hNet
  - QF-FB
- **Beyond QuantumFlow with QFNN**
  - FFNN
  - VQC
  - QF-Mixer



QF-pNet --- P-LYR based Quantum Neuron: *P_Neuron_Circ*

Sub module of `qfnn.qf_circ`

- **Given:** (1) Number of input neuron $\mathcal{N}$; (2) input $\mathcal{I}$; (3) weights $\mathcal{W}$; (4) an empty quantum circuit $\mathcal{C}$
- **Do:** (1) Create input qubits **Q1**; (2) create auxiliary qubits **Q2**; (3) create output qubits **Q3**; **(4)** create the circuit
- **Output:** (1) Quantum circuit $\mathcal{C}$ with encoded inputs $\mathcal{I}$ and embedded weights $\mathcal{W}$ on $\mathcal{N}$ qubits; (2) sets of qubits (**Q1-3**)

$\mathcal{C}$

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

$\mathcal{N}$

```
#init circuit
p_layer_example = P_Neuron_Circ(4)
```

Q1, **Q2**, **Q3**

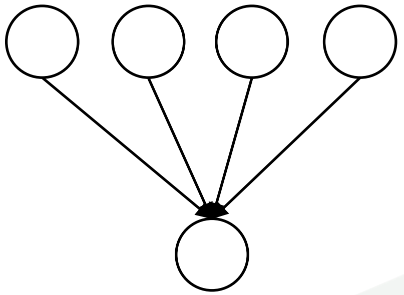
```
#create qubits to be invovled and store them
inps = p_layer_example.add_input_qubits(circuit_demo,'p_input')
aux =p_layer_example.add_aux(circuit_demo,'aux_qubit')
output = p_layer_example.add_out_qubits(circuit_demo,'p_out_qubit')
```

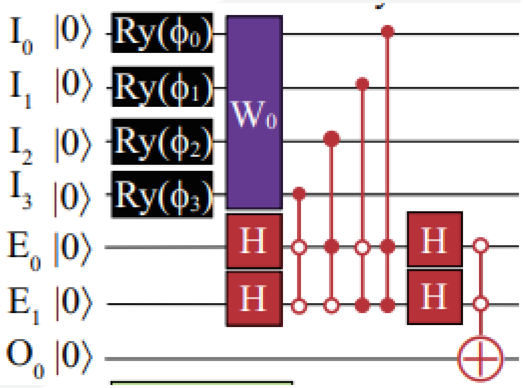
(4) $\mathcal{W}$ $\mathcal{I}$

```
#add p-neuron to the circuit
p_layer_example.forward(circuit_demo,[weight_1[0]],inps[0],output,aux, input)
```

$\mathcal{C}$

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





QF-pNet --- P-LYR as the last layer (sharing inputs): *P_LYR_Circ*

Sub module of `qfnn.qf_circ`

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

  (3) a quantum circuit $\mathcal{C}$ with previous layers; (4) set of output qubits **Q3.**
- **Do:** (1) create output qubits **OutQ**; **(2)** create the circuit;

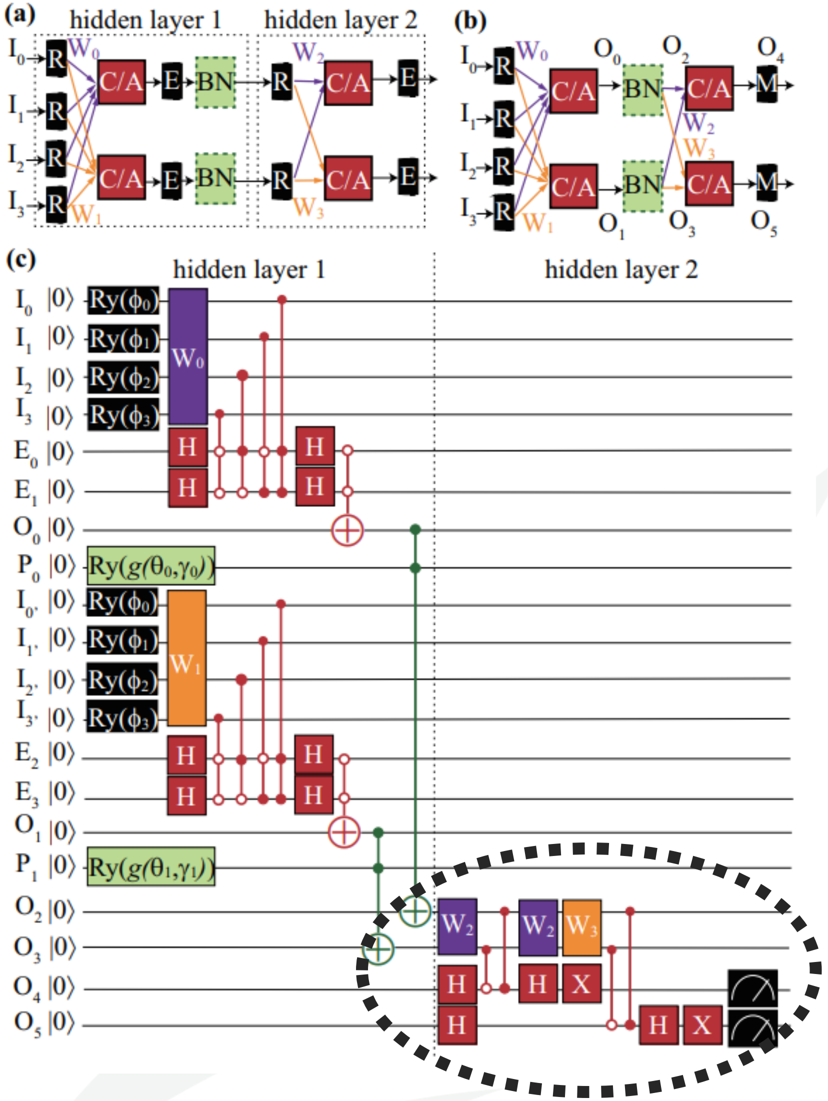
  (3) add measurement to extract results.
- **Output:** (1) Quantum circuit $\mathcal{C}$ with multiple layers; (2) output qubits **OutQ.**

```
p_layer = P_LYR_Circ(2,2)

# Create output qubits
p_layer_output = p_layer.add_out_qubits(circuit)

# Build the second layer
p_layer.forward(circuit,weight_2,output_list,p_layer_output)

# Extract the results at the end of the quantum circuit
add_measure(circuit,p_layer_output,'reg')
print("Output layer created!")

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

OutQ (labels the add_out_qubits line)

(2) (labels the forward line)

(3) (labels the add_measure line)

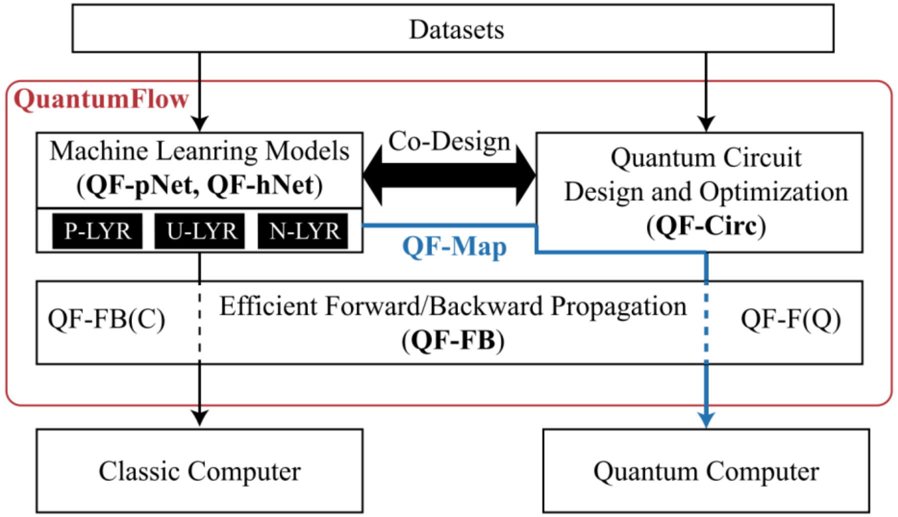


qfnn API Example (1)

QF-pNet

Agenda – Session 3: QFNN API

- Introduction to QFNN
  - Structure: qf_circ, qf_net, qf_fb, qf_map
- Building QuantumFlow using QFNN
  - QF-pNet
  - QF-hNet
  - QF-FB
- Beyond QuantumFlow with QFNN
  - FFNN
  - VQC
  - QF-Mixer



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 (2)

QF-hNet

Agenda – Session 3: QFNN API

- **Introduction to QFNN**
  - Structure: qf_circ, qf_net, qf_fb, qf_map
- **Building QuantumFlow using QFNN**
  - QF-pNet
  - QF-hNet
  - QF-FB
- **Beyond QuantumFlow with QFNN**
  - FFNN
  - VQC
  - QF-Mixer

qfnn API Example (3)

QF-FB

Agenda – Session 3: QFNN API

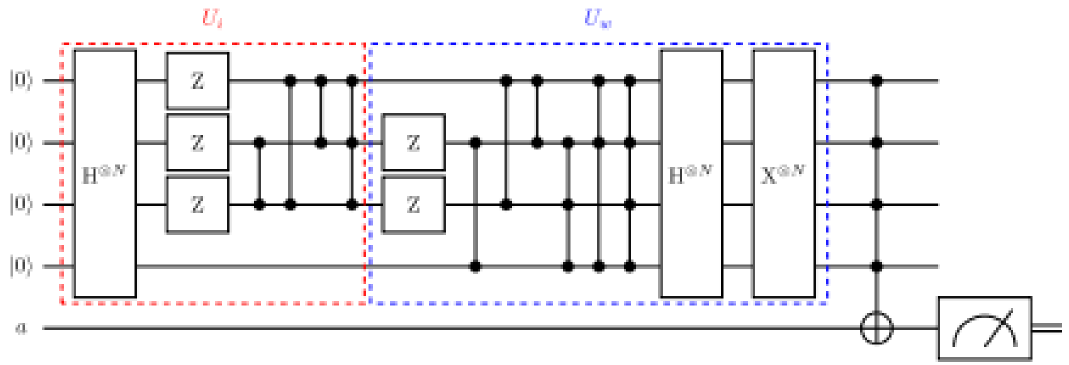
- **Introduction to QFNN**
  - Structure: qf_circ, qf_net, qf_fb, qf_map
- **Building QuantumFlow using QFNN**
  - QF-pNet
  - QF-hNet
  - QF-FB
- **Beyond QuantumFlow with QFNN**
  - FFNN
  - VQC
  - QF-Mixer

FFNN: 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)
```

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

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

$\mathcal{C}$

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

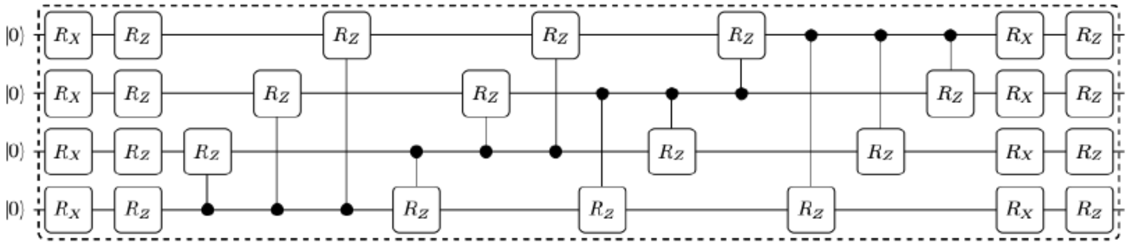
qfnn API Example (4)

FFNN

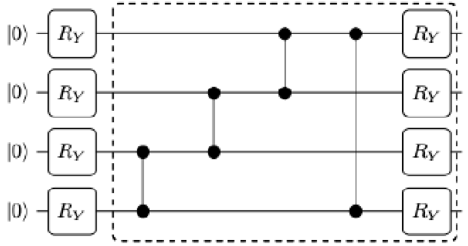
VQC: Variational Quantum Circuits

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

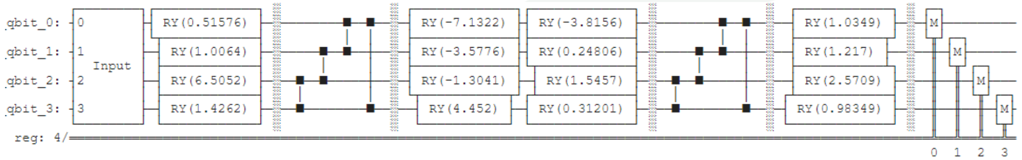


Circuit 5



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)
```



qfnn API Example (5)

VQC

Agenda – Session 3: QFNN API

- **Introduction to QFNN**
  - Structure: qf_circ, qf_net, qf_fb, qf_map
- **Building QuantumFlow using QFNN**
  - QF-pNet
  - QF-hNet
  - QF-FB
- **Beyond QuantumFlow with QFNN**
  - FFNN
  - VQC
  - QF-Mixer → Next Session after the introduction of QF-Mixer

wjiang8@gmu.edu

George Mason University

4400 University Drive
Fairfax, Virginia 22030

Tel: (703)993-1000