\*\*Instruction for Quantum Neural Network Implementation\*\*

### **Using Cirq and the Data Encoding Process: A Beginner-Friendly Explanation**

#### **What is Cirq?**

* Cirq is a **quantum computing library developed by Google**. It is used to design and simulate **quantum circuits**, which are the foundation of computations on quantum computers.
* Quantum circuits operate on **qubits**, the basic unit of quantum information. A qubit can exist in a superposition of both 0 and 1 simultaneously, unlike a classical bit that is strictly 0 or 1.

### **How to Use Cirq for Quantum Circuit Design**

1. **Create Qubits**:

Qubits are the fundamental building blocks of quantum circuits. In Cirq, you can define qubits as follows:  
 import cirq

qubits = [cirq.GridQubit(i, 0) for i in range(4)]

* + - GridQubit(i, 0) creates a qubit at position (i, 0) on a 2D grid.
    - The code above generates 4 qubits: q0,q1,q2,q3q\_0, q\_1, q\_2, q\_3.

1. **Create a Circuit**:

Use the Circuit object to start building your quantum circuit:  
 circuit = cirq.Circuit()

* + - This initializes an empty circuit where quantum gates can be added.

1. **Encode Data into Qubits**:

To represent data in qubits, you use rotation gates. For example:  
 import sympy

theta = sympy.Symbol('theta')

circuit.append(cirq.rx(theta).on(qubits[0]))

* + - cirq.rx(theta) applies a rotation around the X-axis by an angle θ\theta to the qubit.
    - .on(qubits[0]) applies this operation to the first qubit (q0q\_0).
  + By converting data values into angles (θ\theta), you can encode them into the quantum state of the qubit.

1. **Add Entanglement**:

Entanglement is introduced to connect qubits and allow them to share information. In Cirq, you can do this using gates like XX and YY:  
 circuit.append(cirq.XX(qubits[0], qubits[1]))

circuit.append(cirq.YY(qubits[1], qubits[2]))

* + - These operations create interactions between qubits, enabling the circuit to learn complex data relationships.

### **How to Encode Data into Qubit States**

1. **Normalize the Data**:

To encode data into a quantum circuit, first normalize the values to fit within a valid range for quantum gates, such as [0,π][0, \pi]:  
 normalized\_value = (data\_value / max\_value) \* np.pi

* + - This scales the data to an appropriate angle for rotation.

1. **Map to Rotation Angles**:

Use the normalized data to rotate each qubit. For instance:  
 for i, value in enumerate(sample): # sample is one row of data

circuit.append(cirq.rx(value).on(qubits[i]))

* + - Each data point determines the rotation angle for a corresponding qubit.

1. **Create a Circuit for a Data Sample**:

Combine the above steps into a function that takes a data sample and outputs a quantum circuit:  
 def encode\_data\_to\_circuit(data, qubits):

circuit = cirq.Circuit()

for i, value in enumerate(data):

circuit.append(cirq.rx(value).on(qubits[i]))

return circuit

### **Example: Encoding Simple Data**

# Create qubits

qubits = [cirq.GridQubit(i, 0) for i in range(3)] # 3 qubits

# Sample data (e.g., [0.5, 1.0, 0.2])

data\_sample = [0.5, 1.0, 0.2]

normalized\_data = [(value / max(data\_sample)) \* np.pi for value in data\_sample]

# Build a quantum circuit

circuit = cirq.Circuit()

for i, value in enumerate(normalized\_data):

circuit.append(cirq.rx(value).on(qubits[i]))

# Print the resulting circuit

print(circuit)

Output:

(0, 0): ───Rx(1.57)───

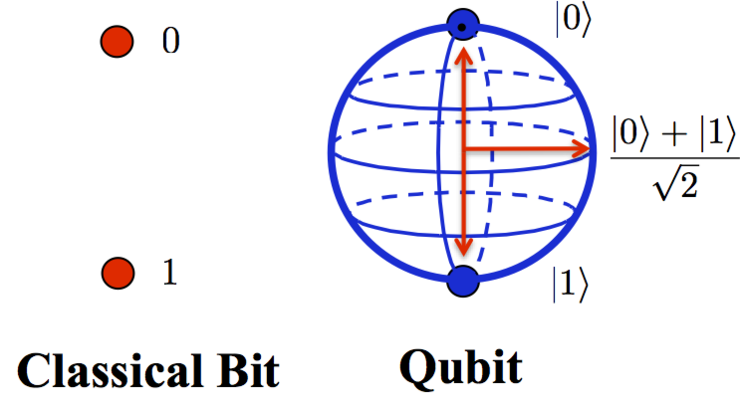
(1, 0): ───Rx(3.14)───

(2, 0): ───Rx(0.63)───

This circuit represents the data [0.5,1.0,0.2][0.5, 1.0, 0.2] as rotations on three qubits. Each qubit's state is rotated based on the normalized value of the corresponding data point.

### **Why Map Data to Qubit Rotation Angles?**

* **Representing Data in Quantum Computers**:  
  + Quantum computers process information using **qubits**. Unlike classical bits, which are either 0 or 1, qubits can exist in a **superposition** of both 0 and 1 states simultaneously.
  + To represent data in qubits, the data needs to be encoded into the qubit's state. This is often done using **rotation angles**.



* **Mapping to Rotation Angles**:  
  + Qubit states can be visualized on a 3D sphere called the **Bloch sphere**. Rotation gates, like RX(θ), move the qubit state to a specific position on this sphere based on the angle θ.
  + By mapping data values to these rotation angles, we translate classical data into a form that can be processed by quantum circuits.
* **Advantages**:  
  + Rotation angles allow the quantum circuit to uniquely represent data patterns. This enables the circuit to learn and compute using the encoded data.

### **What is a Qubit?**

* A **qubit** (quantum bit) is the fundamental unit of information in quantum computing.
* **Classical Bits vs. Qubits**:
  + A classical bit is either 0 or 1.
  + A qubit, however, can exist in a combination of 0 and 1 states simultaneously, called **superposition**. For example, a qubit can be represented as ∣ψ⟩=α∣0⟩+β∣1⟩|\psi\rangle = \alpha|0\rangle + \beta|1\rangle, where α\alpha and β\beta are probability amplitudes.
* **State Representation**:
  + Qubit states can be visualized on a **Bloch sphere**, where the state is a vector pointing to a specific location on the sphere.
  + Using quantum gates, these vectors (qubit states) can be rotated and manipulated to perform computations.

### **What is Entanglement?**

* **Entanglement** is a phenomenon where two or more qubits become strongly correlated, such that the state of one qubit is dependent on the state of another, even if they are far apart.
* **Key Characteristics**:
  + Measuring one qubit instantly determines the state of its entangled partner. For example, in the entangled state (∣00⟩+∣11⟩)/2(|00\rangle + |11\rangle) / \sqrt{2}, if one qubit is measured as 0, the other will also be 0.
* **Why is Entanglement Important?**:
  + Entanglement is one of the core features that make quantum computing powerful. It enables qubits to interact in complex ways, allowing for faster and more efficient computations compared to classical systems.

### **Simple Analogies for Understanding**

1. **Qubits**:  
   * Imagine a qubit as an arrow on a 3D globe (the Bloch sphere). The arrow can point to any location on the globe. The direction of the arrow determines the probabilities of measuring the qubit as 0 or 1.
   * Encoding data into a qubit is like adjusting the arrow’s angle to represent the data.
2. **Entanglement**:  
   * Think of two synchronized dice. If they are entangled, rolling one die will immediately determine the outcome of the other, no matter how far apart they are. This is similar to how entangled qubits share their states instantaneously.

### **Summary**

* Qubits are the building blocks of quantum computers, capable of existing in superposition states.
* Data is mapped to qubits by converting it into rotation angles, which encode the information as quantum states.
* Entanglement allows qubits to share states, enabling the complex interactions that power quantum computing's capabilities.

This document provides step-by-step instructions for implementing a Quantum Neural Network (QNN) using TensorFlow Quantum (TFQ) and Cirq. The objective is to classify network traffic data into benign or malicious categories using QNN models.

---

### 1. Environment Setup and Dependencies

1. \*\*Mount Google Drive\*\*:

Mount your Google Drive to access the dataset stored within:

from google.colab import drive

drive.mount('/content/drive')

```

2. \*\*Install Required Libraries\*\*:

Install TensorFlow Quantum and related libraries:

!pip install tensorflow==2.15.0

!pip install tensorflow-quantum==0.7.3

```

3. \*\*Import Necessary Modules\*\*:

import pandas as pd

import numpy as np

import tensorflow as tf

import tensorflow\_quantum as tfq

import cirq

import sympy

from sklearn.model\_selection import train\_test\_split

from sklearn.utils import resample

import matplotlib.pyplot as plt

```

---

### 2. What are TensorFlow Quantum and Cirq?

1. \*\*TensorFlow Quantum (TFQ)\*\*:

TensorFlow Quantum is a library designed to train quantum machine learning models on quantum data or simulate quantum models on classical computers. It integrates seamlessly with TensorFlow, allowing researchers to define quantum circuits and use them as layers in deep learning models.

2. \*\*Cirq\*\*:

Cirq is a Python library developed by Google for creating, simulating, and executing quantum circuits. Cirq is used within TFQ to design and simulate quantum gates and operations. It provides tools to work with individual quantum bits (qubits) and build parameterized quantum circuits.

3. \*\*How They Work Together\*\*:

TFQ uses Cirq to define quantum circuits. These circuits are integrated into TensorFlow models, enabling quantum-classical hybrid machine learning workflows.

---

### 3. Load and Preprocess Data

1. \*\*Load the Dataset\*\*:

Define the file path and load the CSV data:

path = "/content/drive/MyDrive/QNN/archive/"

csv\_filename = f"{path}NF-UNSW-NB15.csv"

df = pd.read\_csv(csv\_filename, low\_memory=False)

```

2. \*\*Select Relevant Features\*\*:

Select important columns from the dataset:

selected\_features = [

'PROTOCOL', 'L7\_PROTO', 'IN\_BYTES', 'OUT\_BYTES',

'IN\_PKTS', 'OUT\_PKTS', 'TCP\_FLAGS', 'FLOW\_DURATION\_MILLISECONDS', 'Label'

]

df\_selected = df[selected\_features]

```

3. \*\*Balance the Dataset\*\*:

Handle imbalanced data by down-sampling benign samples to match malicious samples:

benign = df\_selected[df\_selected['Label'] == 0]

malicious = df\_selected[df\_selected['Label'] == 1]

benign\_downsampled = resample(benign, replace=False, n\_samples=len(malicious), random\_state=123)

df\_balanced = pd.concat([benign\_downsampled, malicious])

```

4. \*\*Split into Training and Testing Data\*\*:

Divide the dataset into training and testing subsets:

train, test = train\_test\_split(df\_balanced, test\_size=0.15, random\_state=1)

```

5. \*\*Normalize and Quantize Features\*\*:

Encode features into a suitable range for quantum circuits:

def encode\_features(df, features):

df\_encoded = df.copy()

for feature in features:

if feature in df\_encoded.columns:

max\_value = df\_encoded[feature].max()

df\_encoded[feature] = df\_encoded[feature] / max\_value \* np.pi

df\_encoded[feature] = np.round(df\_encoded[feature] / 0.25) \* 0.25

return df\_encoded

features\_to\_encode = ['PROTOCOL', 'L7\_PROTO', 'IN\_BYTES', 'OUT\_BYTES',

'IN\_PKTS', 'OUT\_PKTS', 'TCP\_FLAGS', 'FLOW\_DURATION\_MILLISECONDS']

train\_encoded = encode\_features(train, features\_to\_encode)

test\_encoded = encode\_features(test, features\_to\_encode)

```

---

### 4. Quantum Circuit Design

1. \*\*Define Quantum Circuit\*\*:

Create a parameterized quantum circuit (PQC):

def create\_quantum\_model(num\_qubits, num\_layers):

qubits = [cirq.GridQubit(i, 0) for i in range(num\_qubits)]

circuit = cirq.Circuit()

# Encode data into RX gates

for i in range(num\_qubits):

theta = sympy.Symbol(f'theta\_{i}')

circuit.append(cirq.rx(theta).on(qubits[i]))

# Add entanglement layers

for \_ in range(num\_layers):

for i in range(num\_qubits - 1):

circuit.append(cirq.XX(qubits[i], qubits[i + 1]))

circuit.append(cirq.YY(qubits[i], qubits[i + 1]))

readout = cirq.Z(qubits[0])

return circuit, readout

```

2. \*\*How Quantum Encoding Works\*\*:

- \*\*Data to Qubit Mapping\*\*: Each feature in the dataset is mapped to a qubit rotation angle. For example, a normalized feature value is used as the parameter (`theta`) for the `RX` rotation gate.

- \*\*Entanglement\*\*: Entanglement is introduced by adding operations like `XX` and `YY` gates between adjacent qubits. This allows the circuit to learn complex correlations between input features.

- \*\*Measurement\*\*: The state of a specific qubit (e.g., `Z` measurement on the first qubit) is used as the output of the quantum circuit.

---

### 5. Train and Evaluate the QNN Model

1. \*\*Prepare Data for TensorFlow Quantum\*\*:

Convert encoded data into a format suitable for TensorFlow Quantum:

def convert\_to\_tensor(data):

return tfq.convert\_to\_tensor([

cirq.Circuit(cirq.rx(x)(cirq.GridQubit(i, 0)) for i, x in enumerate(sample)) for sample in data

])

x\_train = train\_encoded.drop(columns=['Label']).values.tolist()

y\_train = train\_encoded['Label'].values

x\_test = test\_encoded.drop(columns=['Label']).values.tolist()

y\_test = test\_encoded['Label'].values

x\_train\_tfcirc = convert\_to\_tensor(x\_train)

x\_test\_tfcirc = convert\_to\_tensor(x\_test)

```

2. \*\*Train the Model\*\*:

Compile and fit the QNN model using TensorFlow Quantum:

model = build\_qnn\_model(num\_qubits=4, num\_layers=2, activation\_function='relu')

model.compile(optimizer=tf.keras.optimizers.Adam(learning\_rate=0.01),

loss='binary\_crossentropy', metrics=['accuracy'])

model.fit(x\_train\_tfcirc, y\_train, batch\_size=32, epochs=10,

validation\_data=(x\_test\_tfcirc, y\_test))

```

3. \*\*Evaluate the Model\*\*:

Assess the model's performance on the test dataset:

test\_loss, test\_accuracy = model.evaluate(x\_test\_tfcirc, y\_test)

print(f"Test Loss: {test\_loss}, Test Accuracy: {test\_accuracy}")

```

---

This instruction provides an end-to-end guide for implementing a QNN using TensorFlow Quantum and Cirq, including an explanation of how data is encoded for quantum circuits. This workflow enables the exploration of quantum-classical hybrid machine learning for classification tasks.