Quantum Hadamard Edge Detection Algorithm (QHED)

Ellen Zhang
Army Educational Outreach Program
High School Apprenticeship Program

Edge Detection

- Extracting the outline of a shape from an image
- Basic algorithm using in computer vision, image processing, and machine vision
- Classical implementation
  - Using filters to find light intensity changes
  - Need to process all the pixels: 2^N number of operations
- Quantum Advantages
  - Exponential speed-up
  - More efficient at tackling large images

Quantum Probability Image Encoding

We first import an image and change it from the classical interpretation to the Quantum Probability Image Encoding (QPIE) state. To do this, we need to normalize all the pixel intensities.

Let the original image with n pixels be represented by vector

$$(a_0 a_1 a_2 \ \dots \ a_{n-1}).$$

We can define each pixel of the QPIE state of the image to be

$$b_i = \frac{a_i}{\sqrt{\sum_{k=0}^{n-1} a_k^2}}.$$

Using QPIE, we are able to exponentially reduce memory for storing BW or RGB images. A n pixel image would only need $\lceil \log_2 n \rceil$ qubits. QPIE also uses less resources than similar quantum image representation methods like FRQI or NEQR.

```python
def amplitude_encode(img_data):

    # Calculate the RMS value
    rms = np.sqrt(np.sum(np.sum(img_data**2, axis=1)))

    # Create normalized image
    image_norm = []
    for arr in img_data:
        for ele in arr:
            image_norm.append(ele / rms)

    # Return the normalized image as a numpy array
    return np.array(image_norm)

# Get the amplitude ancoded pixel values
# Horizontal: Original image
image_norm_h = amplitude_encode(image)

# Vertical: Transpose of Original image
image_norm_v = amplitude_encode(image.T)
```

Auxiliary Qubit

We begin by adding an auxiliary qubit initialized to state $|0\rangle$. The purpose of the auxiliary qubit is to be able to find all the edges in one direction (horizontal or vertical) in one pass. We apply the H-gate to the auxiliary qubit and get the following image state:

$$|\text{image}\rangle \otimes \frac{|0\rangle + |1\rangle}{\sqrt{2}} = \frac{1}{\sqrt{2}} \begin{bmatrix} b_0 \\ b_0 \\ b_1 \\ b_1 \\ b_2 \\ \vdots \\ b_{n-1} \\ b_{n-1} \end{bmatrix}$$

We then apply a unitary decrement gate to the image state to get $(b_0, b_1, b_1, b_2, b_2, \ldots, b_{n-1}, b_{n-1}, b_0)^T$, shifting the first element to the end.

```
# Initialize some global variable for number of qubits
data_qb = math.ceil(np.log(image.size) / np.log(2))
anc_qb = 1
total_qb = data_qb + anc_qb

# Initialize the decrement unitary
D2n_1 = np.roll(np.identity(2**total_qb), 1, axis=1)
```

Hadamard Gates

Because we want to find the boundary of the image, we are interested in the difference between neighboring pixels. This operation can be represented as

$$I_{2^{n-1}} \otimes H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 & 0 & 0 & \dots & 0 & 0 \\ 1 & -1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & 1 & \dots & 0 & 0 \\ 0 & 0 & 1 & -1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 & 1 \\ 0 & 0 & 0 & 0 & \dots & 1 & -1 \end{bmatrix}$$

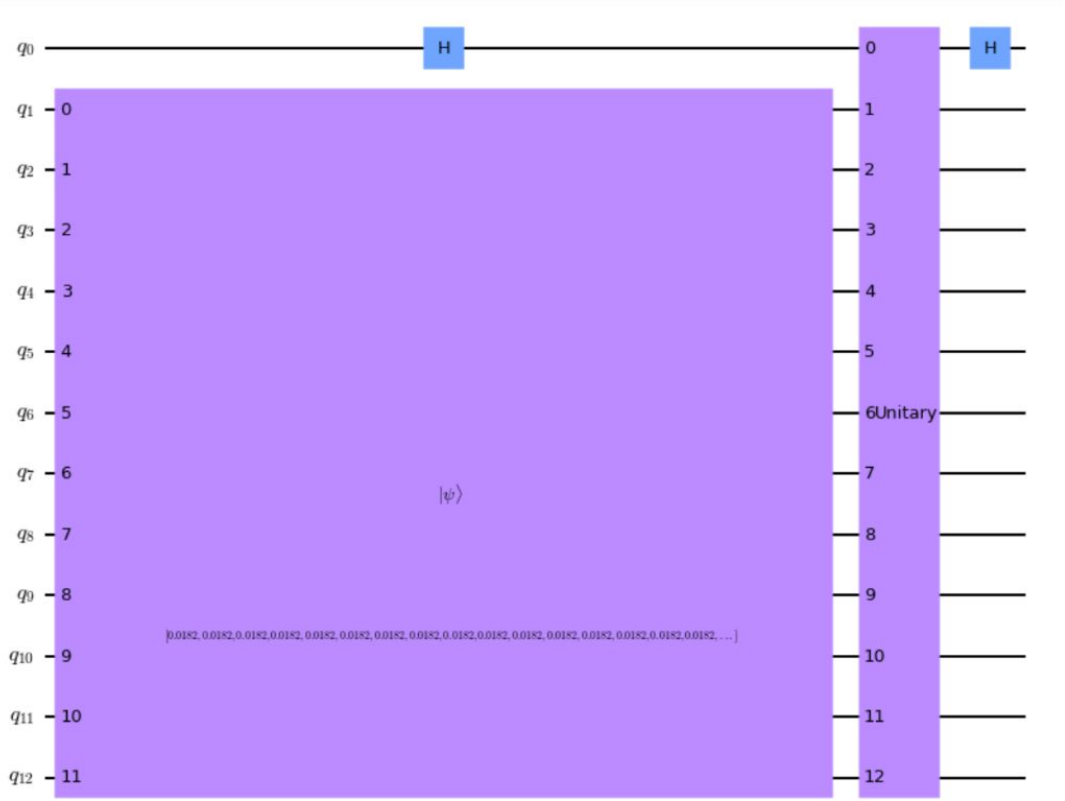
Applying this operation to the image state, we get

$$(I_{2^{n-1}} \otimes H) \cdot \begin{bmatrix} b_0 \\ b_1 \\ b_1 \\ b_2 \\ \vdots \\ b_{n-1} \\ b_{n-1} \\ b_0 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} b_0 + b_1 \\ b_0 - b_1 \\ b_1 + b_2 \\ b_1 - b_2 \\ \vdots \\ b_{n-2} - b_{n-1} \\ b_{n-1} + b_0 \\ b_{n-1} - b_0 \end{bmatrix}$$

```
# Create the circuit for horizontal scan
qc_h = QuantumCircuit(total_qb)
qc_h.initialize(image_norm_h, range(1, total_qb))
qc_h.h(0)
qc_h.unitary(D2n_1, range(total_qb))
qc_h.h(0)
display(qc_h.draw('mpl', fold=-1))

# Create the circuit for vertical scan
qc_v = QuantumCircuit(total_qb)
qc_v.initialize(image_norm_v, range(1, total_qb))
qc_v.h(0)
qc_v.unitary(D2n_1, range(total_qb))
qc_v.h(0)
display(qc_v.draw('mpl', fold=-1))

# Combine both circuits into a single list
circ_list = [qc_h, qc_v]
```

Because this operation will only find the edges for one dimension, we will find the other direction by repeating the operation on the transpose of the image.

Quantum Circuit




```
# Simulating the cirucits
back = Aer.get_backend('statevector_simulator')
results = execute(circ_list, backend=back).result()
sv_h = results.get_statevector(qc_h)
sv_v = results.get_statevector(qc_v)

from qiskit.visualization import array_to_latex
print('Horizontal scan statevector:')
display(array_to_latex(sv_h[:30], max_size=30))
print()
print('Vertical scan statevector:')
display(array_to_latex(sv_v[:30], max_size=30))
```

```
Horizontal scan statevector:
```

$$\begin{bmatrix} 0.01818 & 0 & 0.01818 & 0 & 0.01818 & 0 & 0.01818 & 0 & 0.01818 \end{bmatrix}$$

```
Vertical scan statevector:
```

$$\begin{bmatrix} 0.01818 & 0 & 0.01818 & 0 & 0.01818 & 0 & 0.01818 & 0 & 0.01818 \end{bmatrix}$$

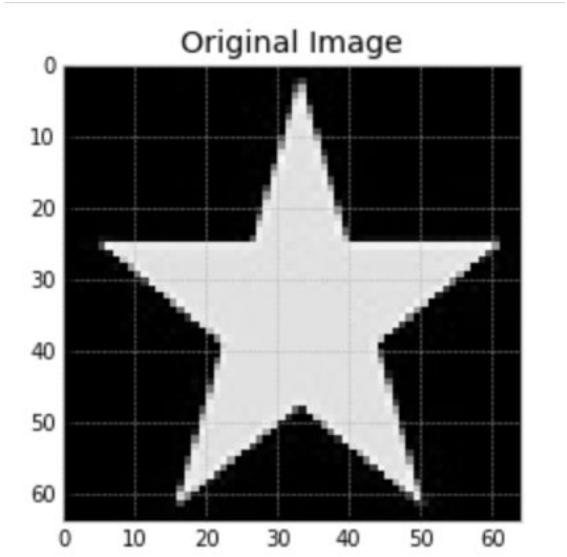
Post-Processing

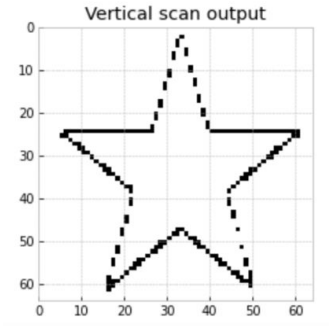
To find the edges of the image, we only need look at the differences between adjacent pixels. To do this, we measure the auxiliary qubit on state $|1\rangle$. The threshold value can be adjusted accordingly. After doing this to both the horizontal and vertical components, we can combine the edges to get the final result.

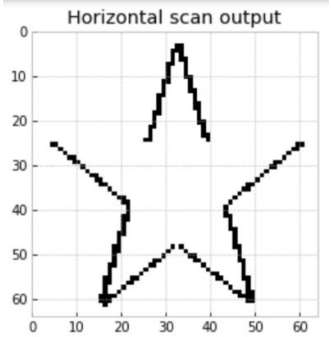
```
# Classical postprocessing for plotting the output

# Defining a lambda function for
# thresholding to binary values
threshold = lambda amp: (amp > 0.002 or amp < -0.002)

# Selecting odd states from the raw statevector and
# reshaping column vector of size 64 to an 8x8 matrix
edge_scan_h = np.abs(np.array([1 if threshold(sv_h[2*i+1].real) else 0 for i in range(2**data_qb)])).reshape(image.shape[0], imag
edge_scan_v = np.abs(np.array([1 if threshold(sv_v[2*i+1].real) else 0 for i in range(2**data_qb)])).reshape(image.shape[0], imag

# Plotting the Horizontal and vertical scans
plot_image(edge_scan_h, 'Horizontal scan output')
plot_image(edge_scan_v, 'Vertical scan output')
```

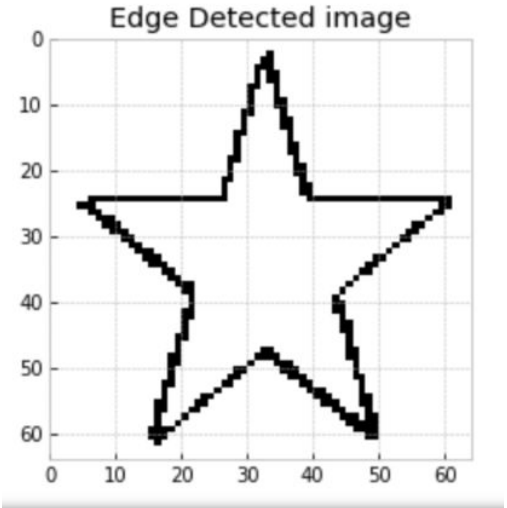
```
# Combining the horizontal and vertical component of the result
edge_scan_sim = edge_scan_h | edge_scan_v

# Plotting the original and edge-detected images
plot_image(image, 'Original image')
plot_image(edge_scan_sim, 'Edge Detected image')
```

Example on a 64x64 image







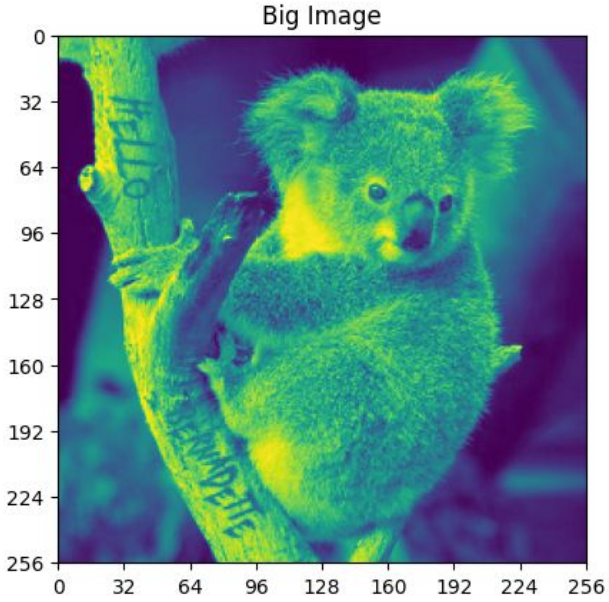


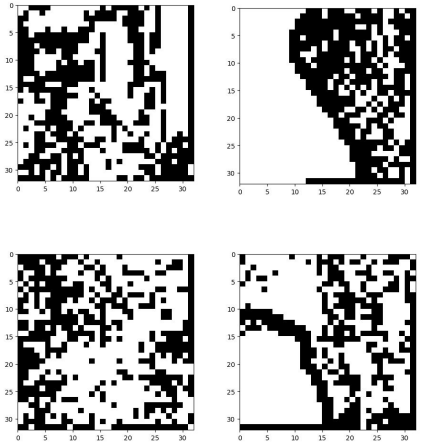
Larger Images

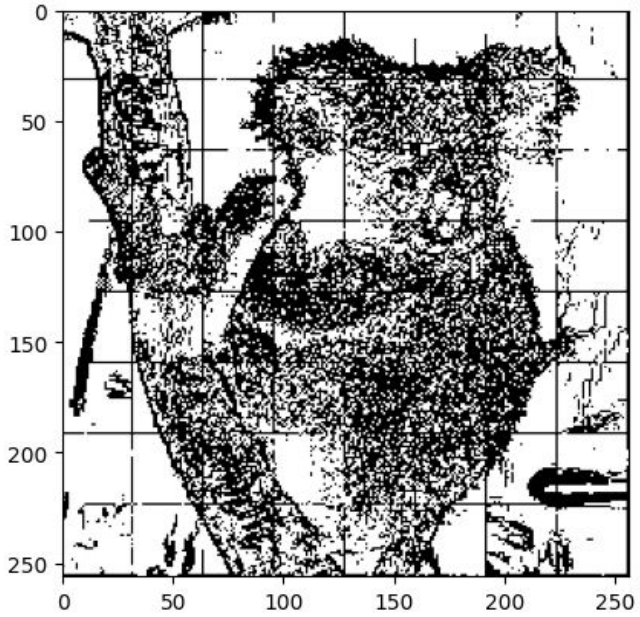
- To directly process large images, fault-tolerant quantum computer needed
- To be able to apply the QHED algorithm efficiently on larger images, one method we can take is to split up large images into smaller ones and recombine at the end.

```
final_img = np.zeros(shape=(256,256))
for a in range(0, 8):
    for b in range(0, 8):
        piece = qhed(image[(a)*32:(a+1)*32, (b)*32:(b+1)*32])
        plot_image(piece)
        final_img[(a)*32:(a+1)*32, b*32:(b+1)*32] = piece
        plot_image(final_img)

plot_image(final_img)
```

Example on a 256x256 image







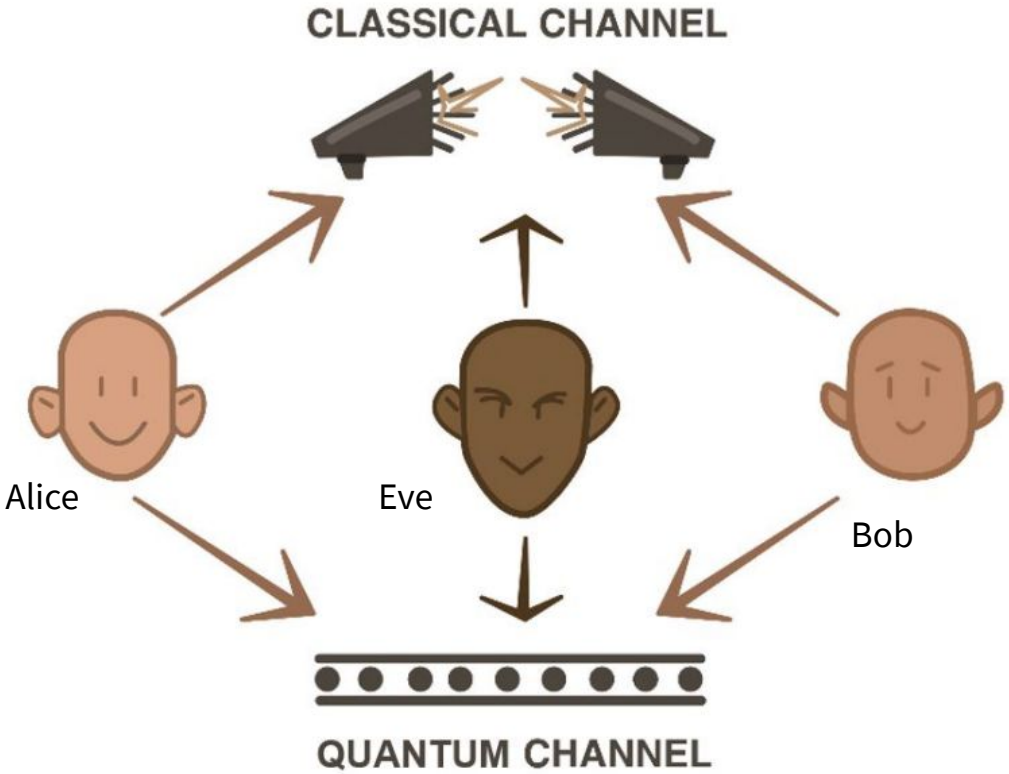
Time Complexity

- Classical algorithms
  - Worst case of $O(2^N)$
  - Best algorithms take O(mn*log(mn))
- QSobel
  - $O(n^2)$
  - FRQI state set-up:$O(n) + O(\log^2 n)$
    - Requires 1+2N qubits
- QHED
  - QPIE state set-up: $O(n^2)$
    - Only requires $\log_2 N$ qubits
  - QHED algorithm: O(1)
    - Doesn’t include state-preparation and amplitude permutation
- Quantum algorithms are less precise

Quantum Key Distribution

Alexandra White

Quantum Key Distribution



What I am trying to solve

- Step 1

Alice chooses a string of random bits, e.g.:

`1000101011010100`

And a random choice of basis for each bit:

`ZZXZXXXZXZXXXXXX`

Alice keeps these two pieces of information private to herself.

- Step 2

Alice then encodes each bit onto a string of qubits using the basis she chose; this means each qubit is in one of the states $|0\rangle$, $|1\rangle$, $|+\rangle$ or $|-\rangle$, chosen at random. In this case, the string of qubits would look like this:

$$|1\rangle|0\rangle|+\rangle|0\rangle|-\rangle|+\rangle|-\rangle|0\rangle|-\rangle|1\rangle|+\rangle|-\rangle|+\rangle|-\rangle|+\rangle|+\rangle$$

This is the message she sends to Bob.

- Step 3

Bob then measures each qubit at random, for example, he might use the bases:

`XZZZXZXZXZXZZZXZ`

And Bob keeps the measurement results private.

- Step 4

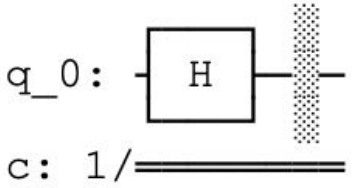
Bob and Alice then publicly share which basis they used for each qubit. If Bob measured a qubit in the same basis Alice prepared it in, they use this to form part of their shared secret key, otherwise they discard the information for that bit.

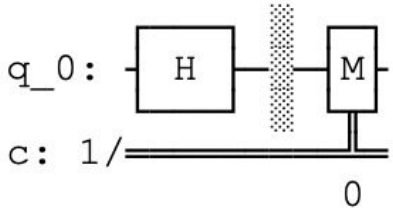
- Step 5

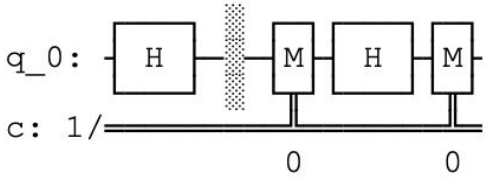
Finally, Bob and Alice share a random sample of their keys, and if the samples match, they can be sure (to a small margin of error) that their transmission is successful.

Suppose Alice sends Bob a qubit, and an eavesdropper (Eve) tries to measure it before it reaches Bob. There is a possibility that Eve's measurement will alter the qubit's state. This may result in Bob receiving the qubit in a different state Alice sent.

Example







In [47]:
```
# import libraries

from qiskit import QuantumCircuit, Aer, transpile, assemble
from qiskit.visualization import plot_histogram, plot_bloch_multivector
from numpy.random import randint
import numpy as np
print("Imports Successful")
```

```
Imports Successful
```

In [48]:
```
np.random.seed(seed=3)
n = 100
#Step 1 is for Alice to generate her random set of bits
alice_bits = randint(2, size=n)# alice_bits is only known to Alice
print(alice_bits)
```

```
[0 0 1 1 0 0 0 1 1 1 0 1 1 1 0 1 1 0 0 0 0 1 1 0 0 0 1 0 0 0 0 1 0 1 1 0 1
 0 0 1 1 0 0 1 0 1 0 1 1 1 1 0 1 0 0 1 1 1 0 0 0 1 0 0 1 1 0 0 1 1 1
 1 0 1 1 1 1 1 1 1 0 1 0 0 1 1 0 1 0 0 0 0 0 1 1 0 1 1]
```

Now we are creating an aray to tell us which
qubits are encoded in which bases. In this case 0 means its prepared
in the Z bases and 1 means its prepared in the X bases

In [49]:
```
alice_bases = randint(2, size=n) #alice_bases is only known to alice
print(alice_bases)
```

```
[1 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 1 1 0 0 1 1 0 0 1 1 1 1 1 0 0
 1 1 0 0 0 0 0 0 0 0 1 1 0 0 0 1 0 1 1 1 0 0 1 0 1 0 1 1 0 0 1 0 0 1 1 1 1
 1 0 0 0 0 0 1 1 1 0 0 0 0 0 0 1 0 1 0 0 0 1 0 1 1 1]
```

In [50]:
```
def encode_message(bits, bases):
    message = []
    for i in range(n):
        qc = QuantumCircuit(1,1)
        if bases[i] == 0: # Preparing the qubit in Z-basis
            if bits[i] == 0:
                pass
            else:
                qc.x(0)
        else: # Preparing the qubit in X-basis
            if bits[i] == 0:
                qc.h(0)
            else:
                qc.x(0)
                qc.h(0)
        qc.barrier()
        message.append(qc)
    return message
```

[51]:
```
message= encode_message(alice_bits, alice_bases)
```

[52]:
```
eve_bases = randint(2, size=n)
intercepted_message = measure_message(message, eve_bases)
print(intercepted_message)
```

```
[0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1,
0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0,
, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0]
```

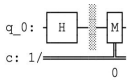
[53]:
```
#Checking first bit in 'alice_bits'
print('bit = %i' % alice_bits[0])
print('basis = %i' % alice_bases[0])
```

```
bit = 0
basis = 1
```

We see that the first bit in 'alices_bits'is 0 and the bases is X bases
Now we can print the circuit to verify her qubit state

In [54]:
```
message[0].draw() #As seen Alice prepared her qubit in state |+>

#The message of qubits is sent over to bob over Eves quantum channel
```

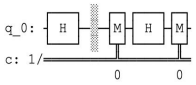
Out[54]:



In [56]:
```
def measure_message(message, bases):
    backend = Aer.get_backend('aer_simulator')
    measurements = []
    for q in range(n):
        if bases[q] == 0: # measuring in Z-basis
            message[q].measure(0,0)
        if bases[q] == 1: # measuring in X-basis
            message[q].h(0)
            message[q].measure(0,0)
        aer_sim = Aer.get_backend('aer_simulator')
        qobj = assemble(message[q], shots=1, memory=True)
        result = aer_sim.run(qobj).result()
        measured_bit = int(result.get_memory()[0])
        measurements.append(measured_bit)
    return measurements
```

In [57]:
```
bob_results = measure_message(message, bob_bases)

#In the circuit (representing the 0th qubit) it has a X measurment added to it by
message[0].draw()
```

Out[57]:



Alice shows which qubits were encoded in which basis (through eves channel)

The Bob reveals which basis he measured each qubit in

* If bob measured a bit in the same basis Alice prepared the bit in the bit can b
key. If they measured the bit in different bases they throw that entry away.

In [58]:
```
#Creating a function to throw away the non corresponding entry
def remove_garbage(a_bases, b_bases, bits):
    good_bits = []
    for q in range(n):
        if a_bases[q] == b_bases[q]:
            # If both used the same basis, add
            # this to the list of 'good' bits
            good_bits.append(bits[q])
    return good_bits
```

After Alice and Bob discard the useless bits they use the remaining bits to form

In [59]:
```
alice_key = remove_garbage(alice_bases, bob_bases, alice_bits)
bob_key = remove_garbage(alice_bases, bob_bases, bob_results)
```

Now Alice and Bob compare a random selection of the bits in their keys to make sure the protocol worked correctly

In [60]:
```
def sample_bits(bits, selection):
    sample = []
    for i in selection:
        # use np.mod to make sure the
        # bit we sample is always in
        # the list range
        i = np.mod(i, len(bits))
        # pop(i) removes the element of the
        # list at index 'i'
        sample.append(bits.pop(i))
    return sample
```

In [61]:
```
#Alice and bob both show these publicly, and then remove them from their
#keys because they are no longer secret
sample_size = 15
bit_selection = randint(n, size=sample_size)

bob_sample = sample_bits(bob_key, bit_selection)
print("  bob_sample = " + str(bob_sample))
alice_sample = sample_bits(alice_key, bit_selection)
print("alice_sample = "+ str(alice_sample))
```

```
  bob_sample = [1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0]
alice_sample = [1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]
```

In [62]:
```
bob_sample == alice_sample
```

Out[62]: False

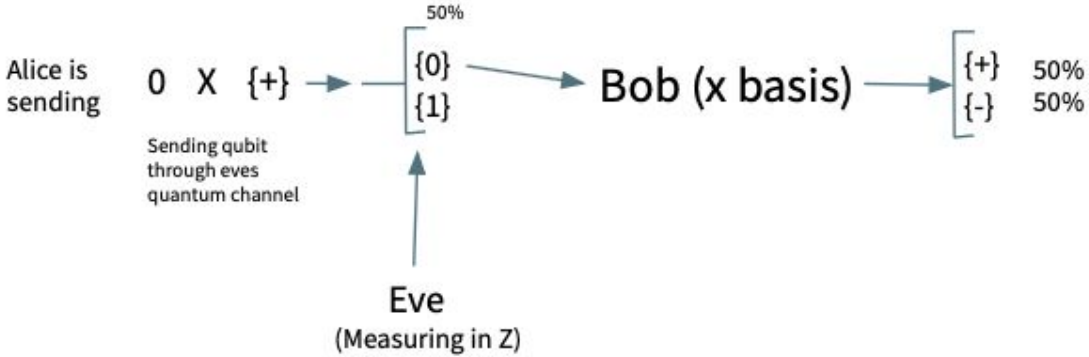
Bob's key and Alice's key do not match. We know this is because Eve tried to read the message between steps 2 and 3, and changed the qubits' states. For all Alice and Bob know, this could be due to noise in the channel, but either way they must throw away all their results and try again- Eve's interception attempt has failed.

If the protocol worked correctly without interference their samples should match

If their samples match, it means (with high probability) alice_key == bob_key. They now share a secret key they can use to encrypt their messages!

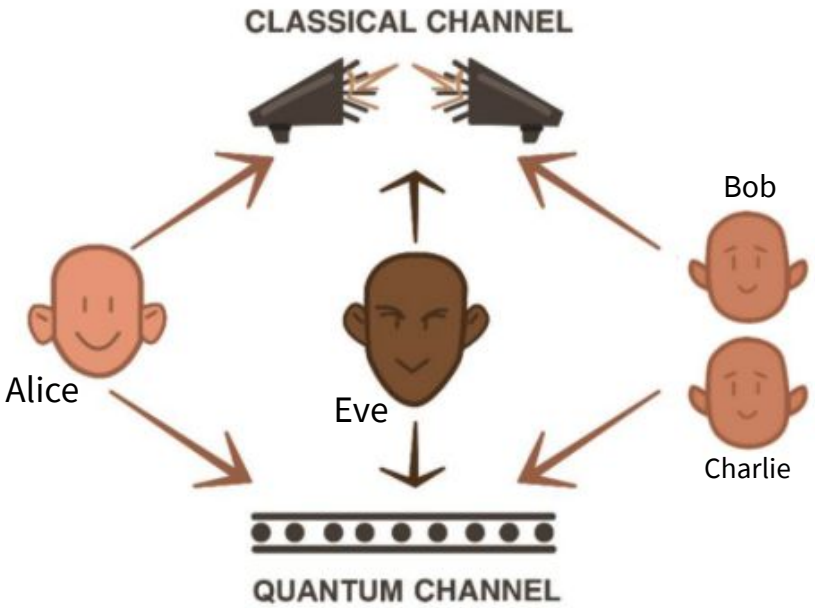
Example Drawn out QKD with interception

message: 0101110
Basis: XZXZZXZ



	0	1
X basis:	{-}	{+}
Z basis:	{0}	{1}

Multi-User Quantum Key Distribution



How it works

- Step 1

Alice chooses a string of random bits, e.g.:

`1000101011010100`

And a random choice of basis for each bit:

`ZZXZXXXZXZXXXXXX`

Alice keeps these two pieces of information private to herself.

- Step 2

Alice then encodes each bit onto a string of qubits using the basis she chose; this means each qubit is in one of the states $|0\rangle$, $|1\rangle$, $|+\rangle$ or $|-\rangle$, chosen at random. In this case, the string of qubits would look like this:

$$|1\rangle|0\rangle|+\rangle|0\rangle|-\rangle|+\rangle|-\rangle|0\rangle|-\rangle|1\rangle|+\rangle|-\rangle|+\rangle|-\rangle|+\rangle|+\rangle$$

This is the message she sends to Bob.

- Step 3

Bob then measures each qubit at random, for example, he might use the bases:

`XZZZXZXZXZXZZZXZ`

And Bob keeps the measurement results private.

- Step 4

Bob and Alice then publicly share which basis they used for each qubit. If Bob measured a qubit in the same basis Alice prepared it in, they use this to form part of their shared secret key, otherwise they discard the information for that bit.

- Step 5

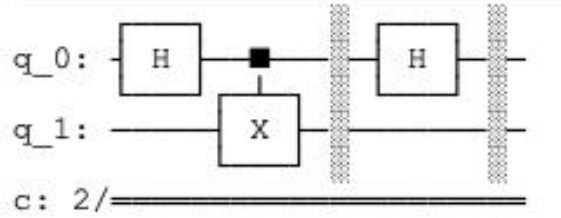
Finally, Bob and Alice share a random sample of their keys, and if the samples match, they can be sure (to a small margin of error) that their transmission is successful.

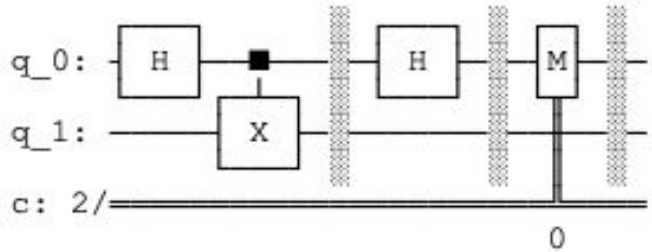
Before Alice encodes each bit onto a string of qubits, She prepares two entangled qubits and sends one of them to charlie and the other one to bob.

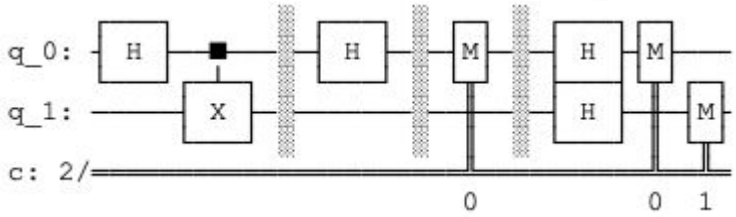
E.g. Alice is sending $|00\rangle$

Alice must entangle everyone's qubit to make sure everyone is measuring the same thing if their basis are the same. You don't need to entangle for single user because there is only one qubit for each message bit

Example







In [204]:
```
np.random.seed(seed=3)
n = 100
#Step 1 is for Alice to generate her random set of bits
alice_bits = randint(2, size=n)# alice_bits is only known to Alice
print(alice_bits)
```

```
[0 0 1 1 0 0 0 1 1 1 0 1 1 1 0 1 1 0 0 0 0 1 1 0 0 0 1 0 0 0 0 1 0 1 1 0 1
 0 0 1 1 0 0 1 0 1 0 1 1 1 1 0 1 0 0 1 1 1 0 0 0 1 0 0 0 1 0 0 1 1 0 0 1 1
 1 0 1 1 1 1 1 1 0 1 0 0 1 1 0 1 0 0 0 0 0 0 1 1 0 1 1]
```

In [205]:
```
alice_bases = randint(2, size=n)#alice_bases is only known to alice
print(alice_bases)
```

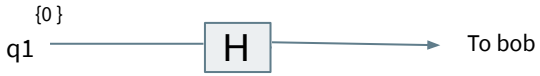
```
[1 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 1 1 0 0 1 1 0 0 1 1 1 1 1 0 0
 1 1 0 0 0 0 0 0 0 0 1 1 0 0 0 1 0 1 1 1 0 0 1 0 1 0 1 1 0 0 1 0 0 1 1 1 1
 1 0 0 0 0 0 1 1 1 0 0 0 0 0 0 1 0 1 0 0 0 1 0 1 1 1]
```

In [206]:
```
m = 2
def encode_message(bits, bases):
    message = []
    for i in range(n):
        qc = QuantumCircuit(m, m)
        # entangle qubits
        qc.h(0)
        for k in range(0, m-1):
            qc.cx(k, k+1)
        qc.barrier()
        # encode
        if bases[i] == 0: # Preparing the qubit in Z-basis
            if bits[i] == 0:
                pass
            else:
                qc.x(0)
        else: # Preparing the qubit in X-basis
            if bits[i] == 0:
                qc.h(0)
            else:
                qc.x(0)
                qc.h(0)
        qc.barrier()
        message.append(qc)
    return message
```

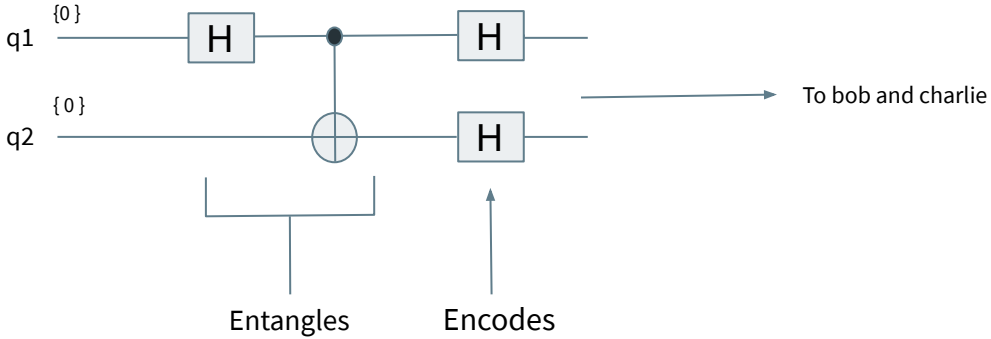
In [207]:
```
message= encode_message(alice_bits, alice_bases)
```

Conclusion

Single receiver



Two-users (receivers)



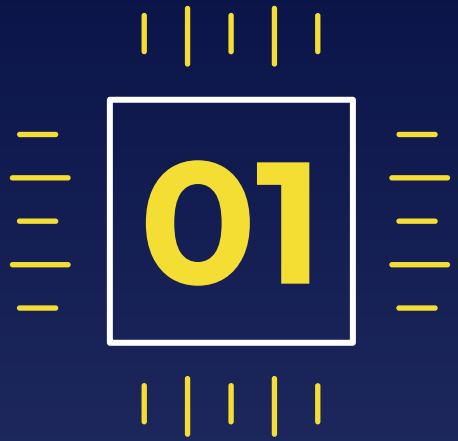
Sources

https://advances.sciencemag.org/content/advances/7/23/eabe0395.full.pdf
https://qiskit.org/textbook/ch-algorithms/quantum-key-distribution.html#4.-Qiskit-Example:-With-Interception
https://en.wikipedia.org/wiki/Quantum_key_distribution
https://qiskit.org/textbook/ch-states/atoms-computation.html#4.1-Encoding-an-input-

Thank you for listening!

Quantum Approximate Optimization Algorithm for the Shortest Vector Problem

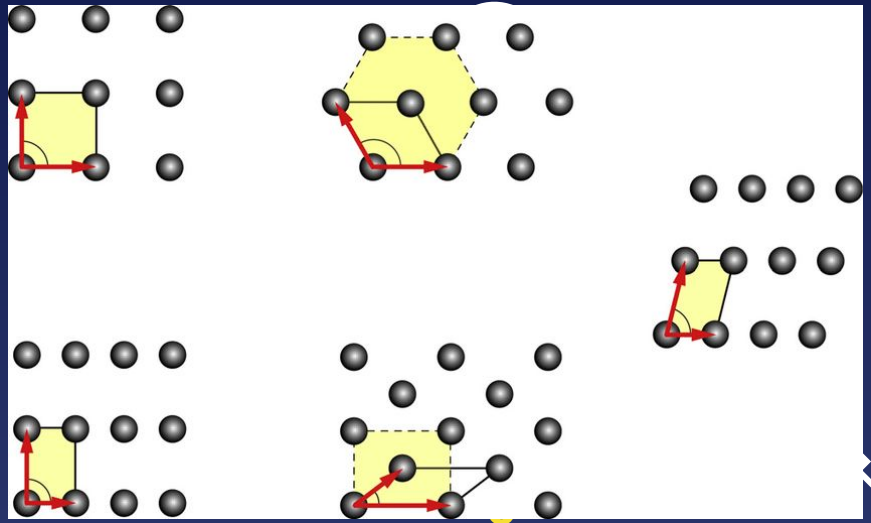
Miriam Kim
High School Apprentice through AEOP



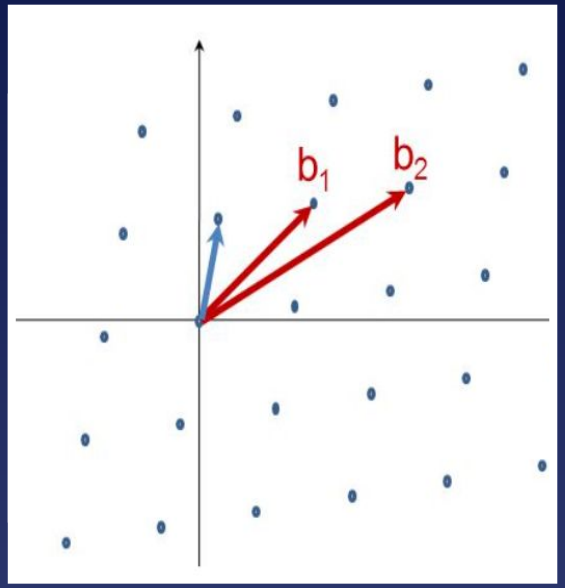
Intro and Background

Important Concepts

- Lattices
  - Lattice based cryptography
- Optimization Problems
  - How are they solved?
  - Applications
- Hamiltonian

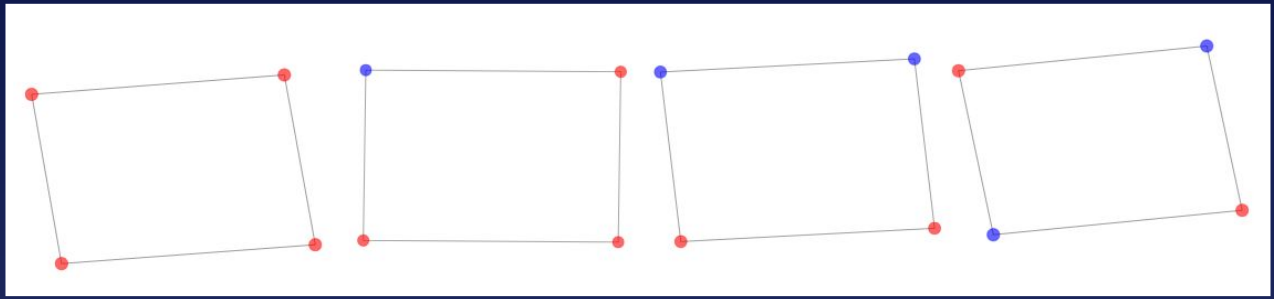


Shortest Vector Problem



- A **Lattice Problem**
- Given a **basis** of vector space V, a norm N, for lattice L, **find the shortest non-zero vector in L**
- Relevance

Max-Cut Problem



- Given a graph, partition nodes into two sets such that the edges between the sets is **maximum**
- Problem Hamiltonian up to a constant:

$$H_P = \frac{1}{2}(Z_0 \otimes Z_1 \otimes I_2 \otimes I_3) + \frac{1}{2}(I_0 \otimes Z_1 \otimes Z_2 \otimes I_3) + \frac{1}{2}(Z_0 \otimes I_1 \otimes I_2 \otimes Z_3) + \frac{1}{2}(I_0 \otimes I_1 \otimes Z_2 \otimes Z_3)$$

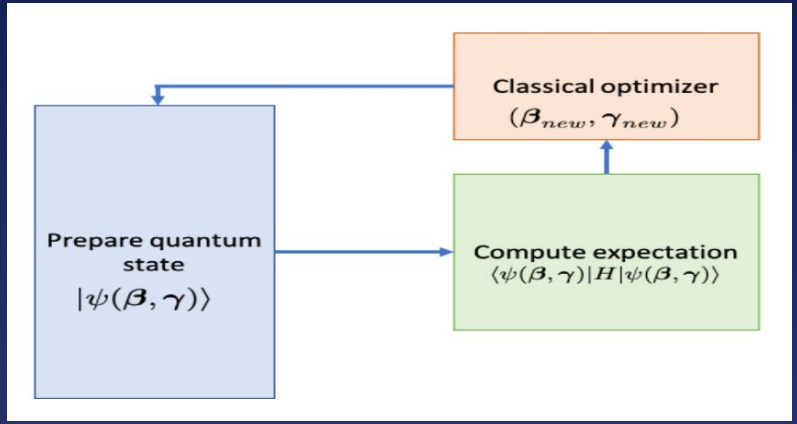
About QAOA

Farhi et al.

Quantum Approximate Optimization Algorithm

A quantum algorithm that attempts to solve combinatorial optimization problems

- Applies Hamiltonians to find optimal parameters (β, γ) to find the minimum eigenvalue of H_p
- $U(\beta, \gamma) \rightarrow |\psi(\beta, \gamma)\rangle$
- Classical bit string expected to have good approximation ratio



QAOA

01

Initial State

- Apply Hadamard gate to each qubit

02

Construct Unitaries from Hamiltonians

- $U(H_B) = e^{-i\beta H\_b}$
- $U(H_P) = e^{-i\gamma H\_p}$

03

Pick a *p* and initialize β, γ

04

Apply Unitaries for *p* times to form state

$$|\psi(\beta,\gamma)\rangle = \underbrace{U(\beta)U(\gamma)\cdots U(\beta)U(\gamma)}_{p \text{ times}}|\psi_0\rangle$$

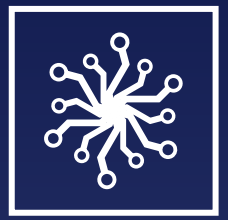
05

Measure $|\beta, \gamma\rangle$

06

Evaluate expectation value classically

Classical and Quantum

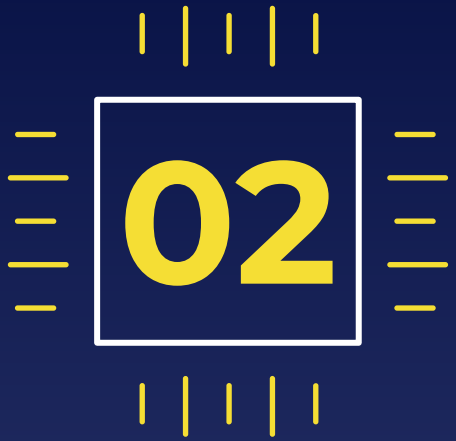


Classical Solution

- Gram-Smith Orthogonalization
- LLL Algorithm

Quantum Solution

- Superposition states → function can be applied to many possible inputs simultaneously



Implementation

Construction

$$H_P = \sum_{i,j}^{N} \hat{Q}^{(i)} \hat{Q}^{(j)} \mathbf{G}_{ij},$$

Problem Hamiltonian

$$H_{\mathrm{M}} = \sum_{j=1}^{n} X_j,$$

Mixing Hamiltonian

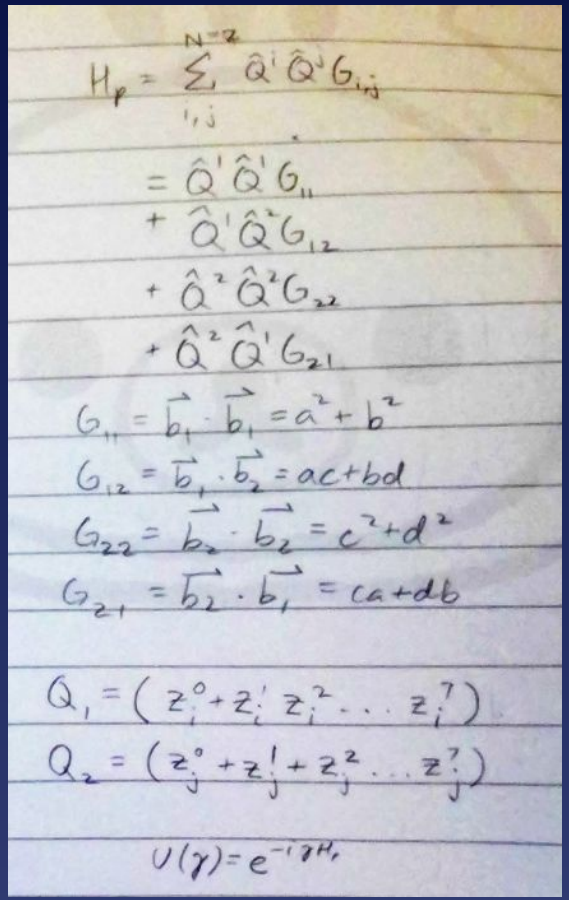
$$|\psi_0\rangle = \left(\frac{1}{\sqrt{2}} (|0\rangle + |1\rangle) \right)^{\otimes n}$$

Initial State

Problem Hamiltonian

$$H_P = \sum_{i,j}^{N} \hat{Q}^{(i)} \hat{Q}^{(j)} \mathbf{G}_{ij},$$

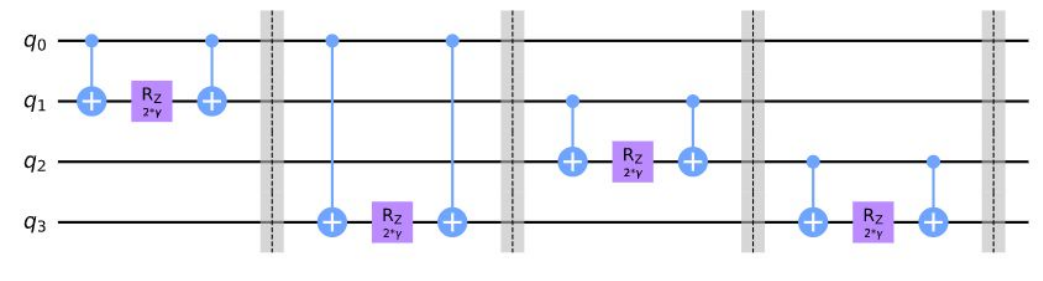
(Joseph et al.)



$$U(\boldsymbol{\gamma}) = e^{-i\boldsymbol{\gamma} H_P}$$

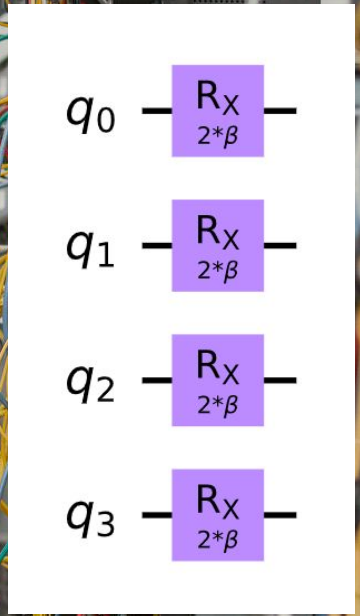
$$U(H_B) = e^{-i\beta H_B}$$

```
gamma = Parameter("$\\gamma$")
qc_p = QuantumCircuit(nqubits)
for pair in list(G.edges()):  # pairs of nodes
    qc_p.rzz(2 * gamma, pair[0], pair[1])
    qc_p.barrier()

qc_p.decompose().draw()
```



Problem Unitary

```
nqubits = 4

beta = Parameter("$\\beta$")
qc_mix = QuantumCircuit(nqubits)
for i in range(0, nqubits):
    qc_mix.rx(2 * beta, i)

qc_mix.draw()
```



Mixing Unitary

Classical Optimization

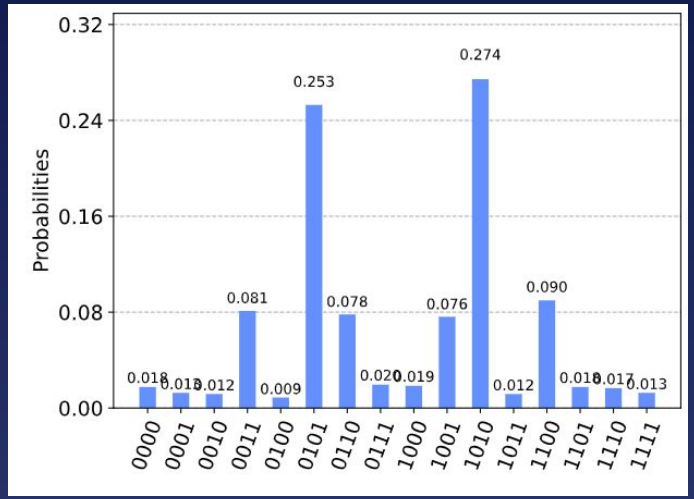
```
from scipy.optimize import minimize


expectation = get_expectation(G, p=1)

res = minimize(expectation,
                       [1.0, 1.0],
                       method='COBYLA')
res
```

After the initial state is prepared apply $U(\beta, \gamma)$ and use classical techniques to optimize the parameters

- Prepare $|\psi(\beta,\gamma)\rangle$
- Measure the state
- Compute $\langle\psi(\beta,\gamma)|H_p|\psi(\beta,\gamma)\rangle$
- Find the new set of parameters
- Set the current parameters equal to the new and repeat

Analyzing Results

```
from qiskit.visualization import plot_histogram

backend = Aer.get_backend('aer_simulator')
backend.shots = 512

qc_res = create_qaoa_circ(G, res.x)

counts = backend.run(qc_res, seed_simulator=10).result().get_counts()

plot_histogram(counts)
```



THANKS!

CREDITS: This presentation template was created by Slidesgo, including icons by Flaticon, and infographics & images by Freepik

Shor's Algorithm

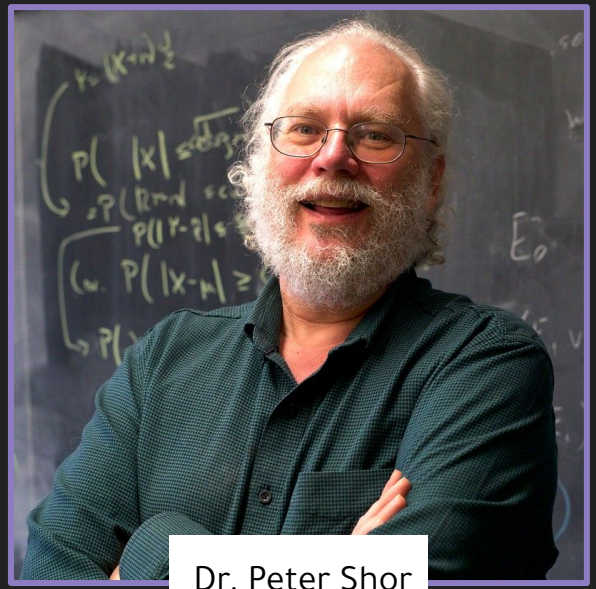
Stephanie Long

NYU Shabani Lab Quanum Internship

Shor’s Algorithm

Who is Shor

- Dr. Shor is an applied mathematics professor at MIT
- Expert in quantum computation
- Discovered Shor's Algorithm in 1994



Dr. Peter Shor

What is an algorithm?

Mathematics Definition

A procedure, a set of steps that can be used to solve a mathematical computation

Engineering Definition

A sequence of well-defined steps that produce a result in finite time

Classical CS Definition

A specific procedure for solving a well-defined computational problem

Quantum CS Definition

A procedure to solve a problem only functional with a quantum computer

Problem

It is not possible to factor a number into it's prime factors in reasonable time using a classical computer (sub-exponential time)

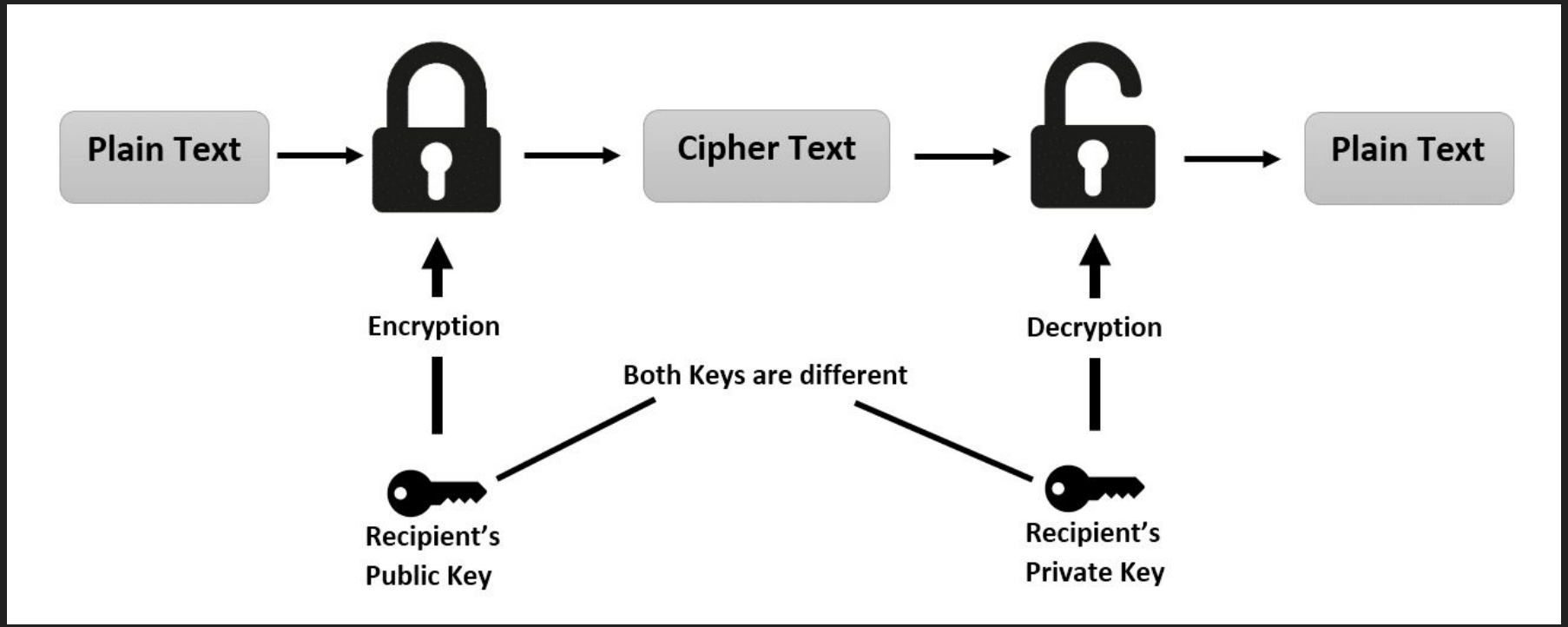
Solution

Quantum computers can factor an extremely large number into it's prime factors in reasonable time (polynomial time)

Useful for decrypting the RSA cryptosystem

What is RSA cryptography?

RSA is an encryption method that we use everyday - for secure emails, transactions, and more



More Details : RSA

RSA is based off of extremely large numbers - takes **1,024**, **2,048**, or **4,096** bits to represent

$2^n-1= \#$

n = bits

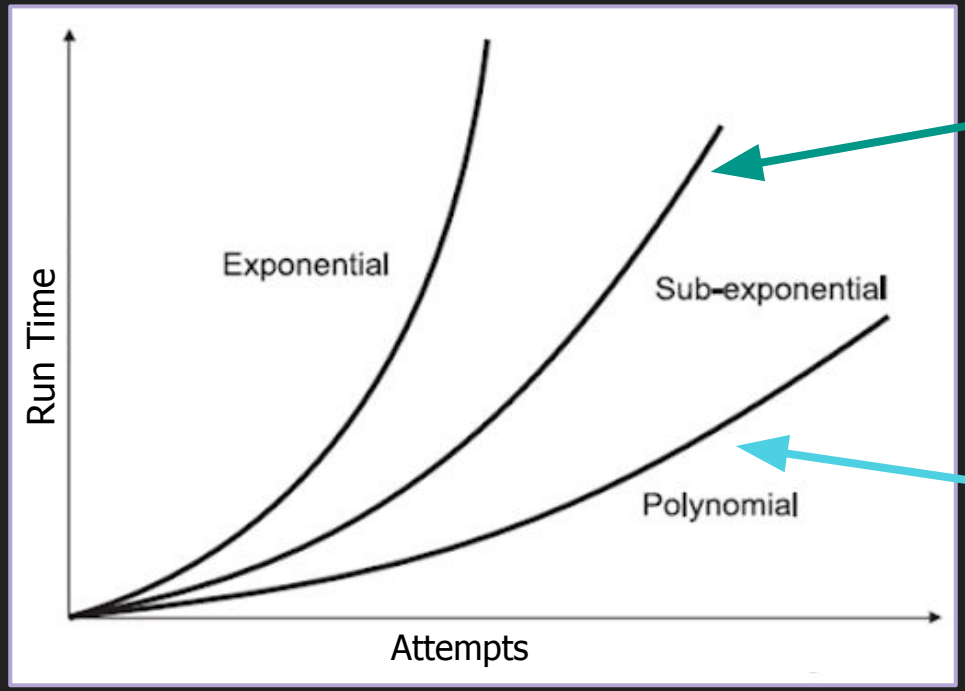
→ Number is too large - overflow error

A key is essentially a prime factor of the RSA number. If you have the RSA number and a prime factor you can deduce the other prime factor and the plain text.

The encryption operates based on the presumption that a computer cannot factor such a large number into it's primes.

More Details : RSA

The algorithm being used on classical computers to factor extremely large numbers today is the general field number sieve



Classical Computer factoring (GFNS)

Quantum Computer factoring (Shor's Algorithm)

Shor's Algorithm - CLASSICAL IMPLEMENTATION

N = extremely large number

STEP 1)

- Generate a random number "a" which satisfies the condition 0 < a < N-1

```
N = ##insert number here##
a = randint(2,N-1)
print("A random number a has been generated:  " + str(a))
```

Shor's Algorithm

N = extremely large number

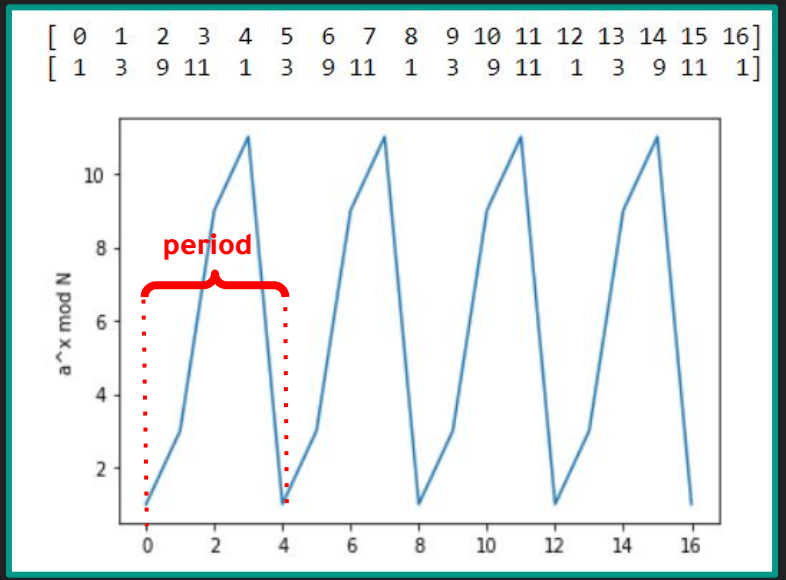
STEP 2)

- Compute the greatest common denominator of a & N (using Euclid's Algorithm)
  - If GCD is != a nontrivial factor has been found, return to step 1

```
def gcd(pick, number):
    if pick == 0 :
        return number

    return gcd(number%pick, pick)

print("The greatest common denominator of (",a,",",N, ") is", gcd(a, N))
```

STEP 3)

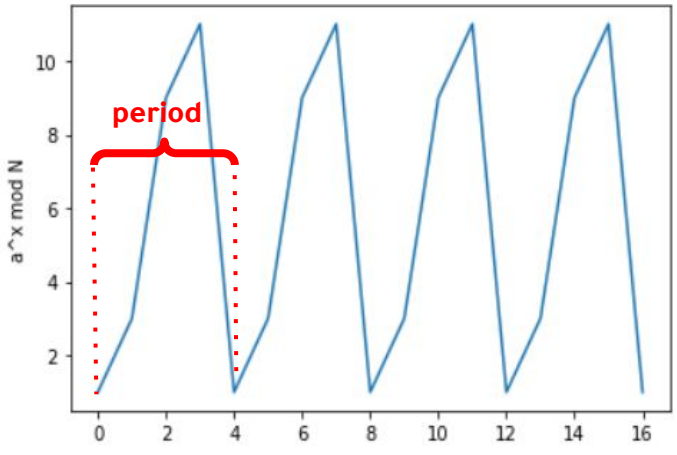
- Graph the equation $y = a^x \bmod N$ from x value 0 to (N-1)



STEP 4)

- Find the period using the equation $a^r \bmod N = 1$

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16]
[ 1  3  9 11  1  3  9 11  1  3  9 11  1  3  9 11  1]
```

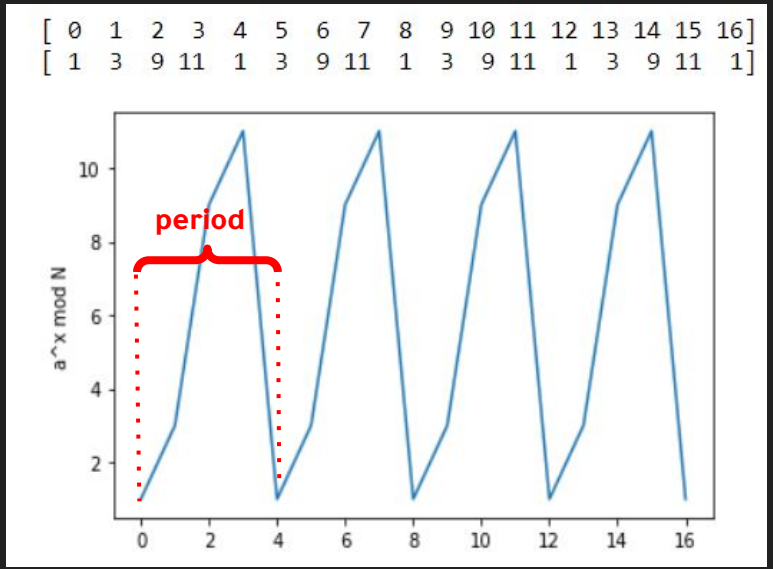


STEP 5)

- Run the period (r) through a series of checks

STEP 6)

- If r passes all checks, then you are more likely to have found the factors, which are found when plugged into the below equations

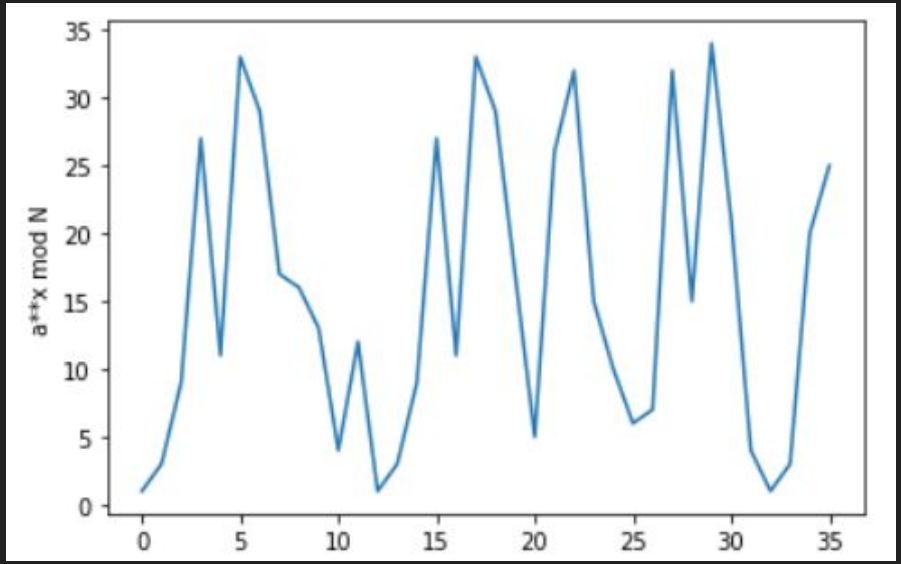


$$gcd(a^r/2 + 1)$$
$$gcd(a^r/2 - 1)$$

So, what's the issue here?

- The N value in the period finding was 16 - a small number

- Classical computation is not fast enough for Shor's Algorithm



--N is increased--

Error occurs - no viable period

Quantum Circuit Implementation

- By implementing the use of a quantum algorithm in the period finding step, we can speed up the algorithm and more efficiently factor large numbers

Why does speed up occur?

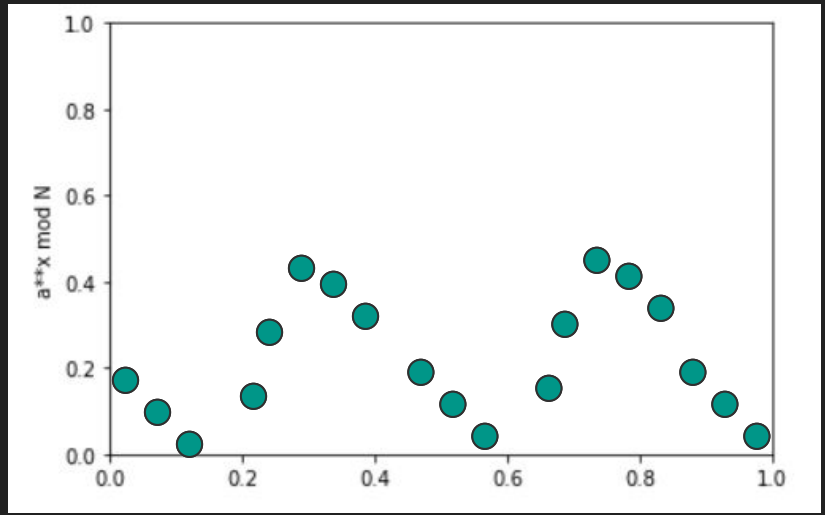
- Classical computers can only find

  one x value at a time

- Due to superposition, quantum

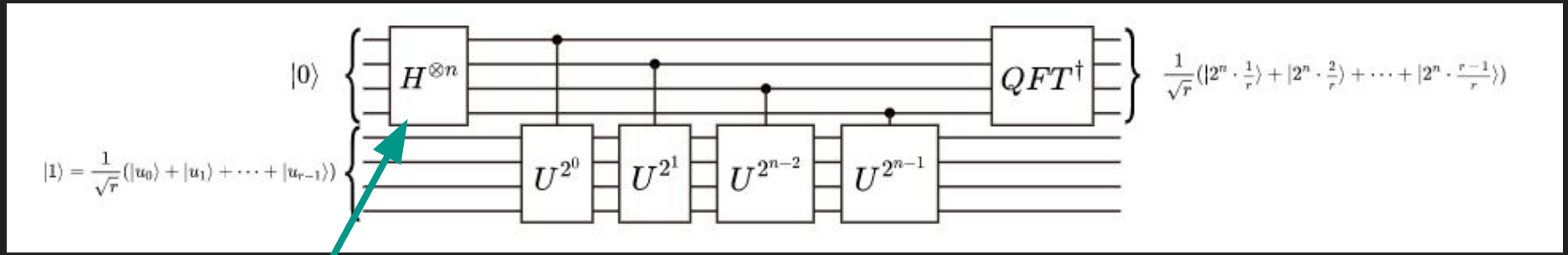
  computers can find multiple x values

  at a time



Quantum Circuit Implementation

Lets take a look at the details:



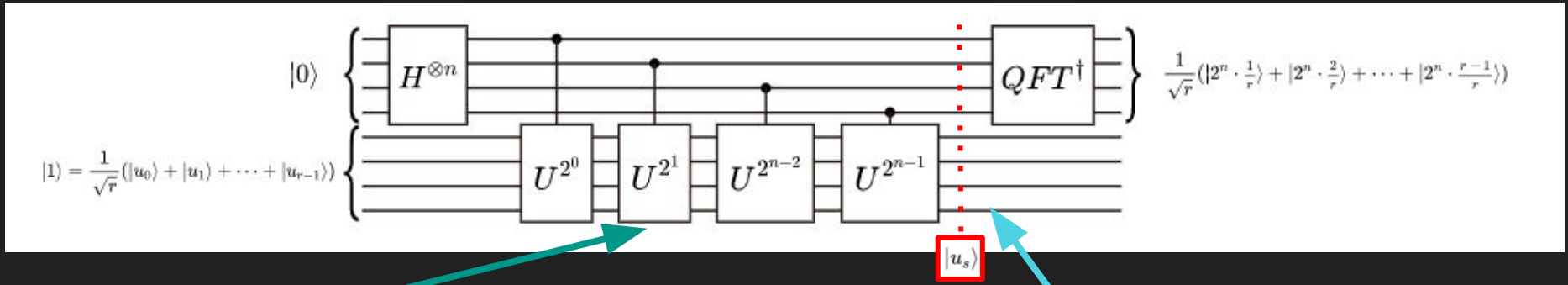
$$H = 1/\sqrt{2} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

- Hadamard gate puts the qubits into a superposition of states
- This superposition allows more than one x-value to be calculated at once - in opposition to the limited calculations of classical bits

$$(|0\rangle + |1\rangle)/\sqrt{2} \rightarrow |\psi\rangle = (|00\rangle + |10\rangle + |01\rangle + |11\rangle)/2$$

Quantum Circuit Implementation

Lets take a look at the details:



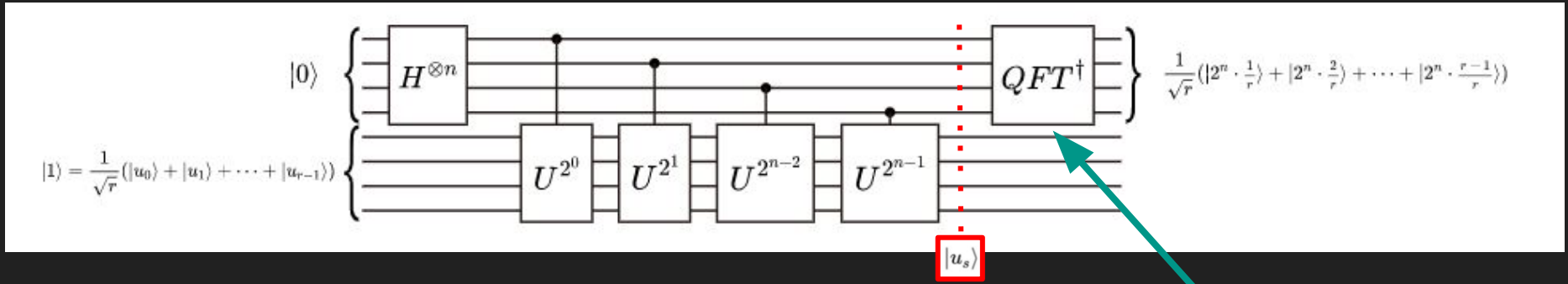
Unitary Gates :

Perform rotations on the qubits

$$|u_s\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{-\frac{2\pi i s k}{r}} |a^k \bmod N\rangle$$

Quantum Circuit Implementation

Lets take a look at the details:



- Produces final superposition
- The period can be extrapolated from this information

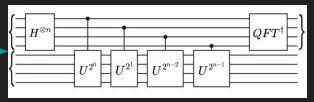
Quantum Fourier Transform:

Acts upon the state generated by the unitary gates

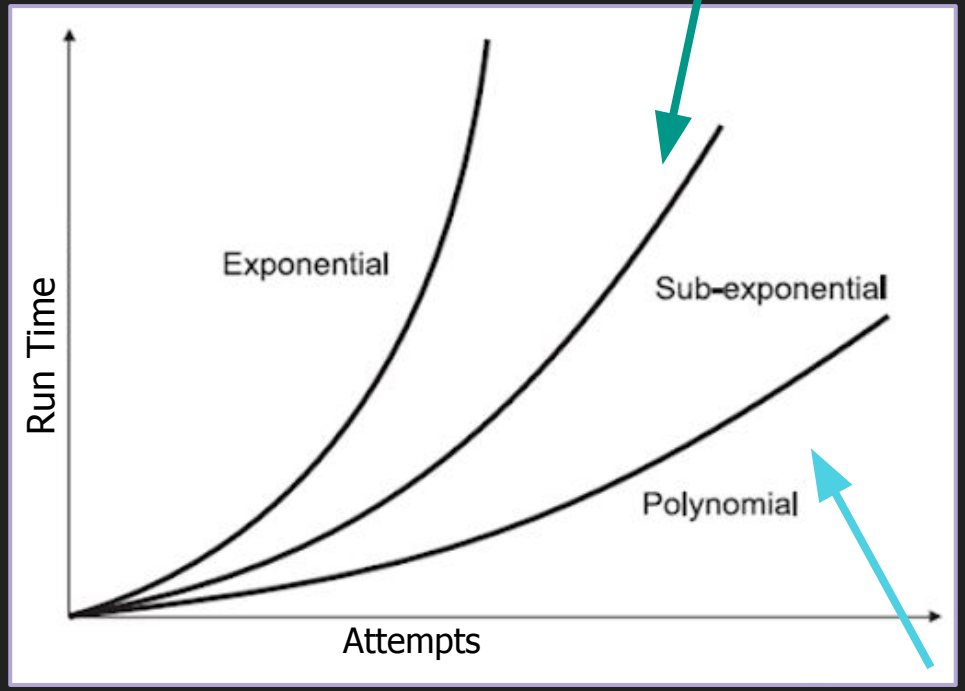
Reminder : Why is Quantum Helpful?

By exchanging the Classical method with the Quantum method of period finding

$$a^r \bmod N = 1$$



the time of computation is reduced from sub-exponential time to polynomial time, making the algorithm possible to run



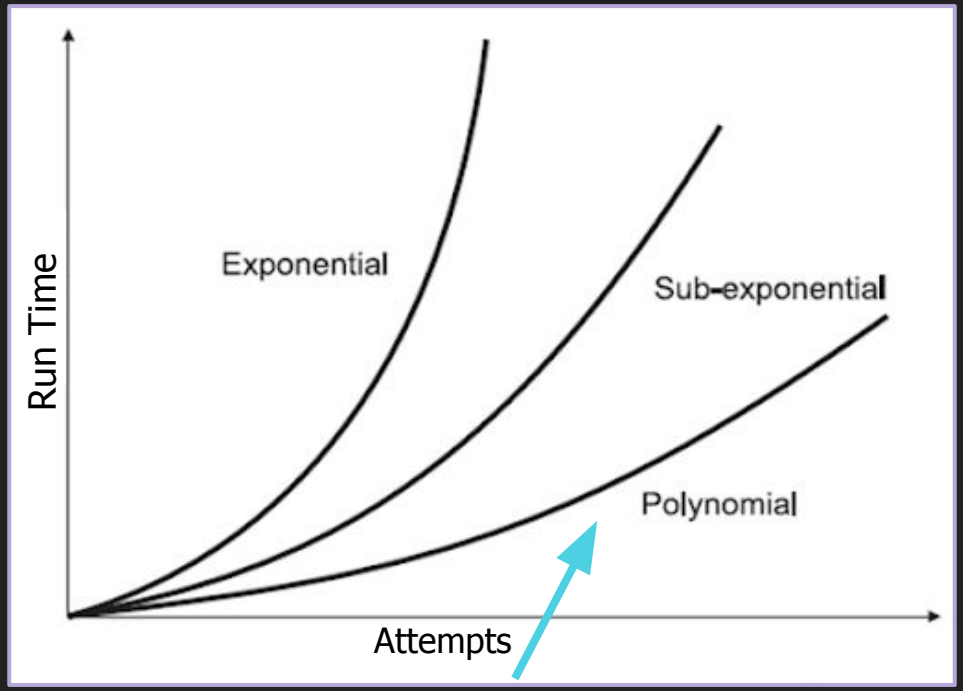
How fast is Shor's Algorithm

- Compared to classical, it is an "exponential speed up"
- y^{10} seconds v.s. 10^{y} seconds, y = digits

quantum

classical

- EX. to break a 2048 bit encryption would take a working Quantum Computer 8 hours, and a classical computer 300 trillion years



Quantum Computer factoring (Shor's Algorithm)

References

1) https://math.mit.edu/directory/profile.php?pid=247
2) https://www.quantum-inspire.com/kbase/what-is-a-quantum-algorithm/
3) https://www.youtube.com/watch?v=ufO_BScIHDQ
4) https://www.geeksforgeeks.org/euclidean-algorithms-basic-and-extended/
5) https://en.wikipedia.org/wiki/Quantum_algorithm
6) https://en.wikipedia.org/wiki/General_number_field_sieve
7) https://www.quantiki.org/wiki/shors-factoring-algorithm#:~:text=Shor's%20algorithm%20is%20a%20quantum,a%20sufficiently%20large%20quantum%20computer.
8) https://quantum-computing.ibm.com/composer/docs/iqx/guide/shors-algorithm
9) https://matplotlib.org/stable/tutorials/introductory/usage.html#sphx-glr-tutorials-introductory-usage-py
10) https://www.w3schools.com/python/python_functions.asp
11) https://numpy.org/doc/stable/reference/generated/numpy.fromfunction.html
12) https://www.w3schools.com/python/python_conditions.asp
13) https://en.wikipedia.org/wiki/RSA_(cryptosystem)
14) https://qiskit.org/textbook/ch-algorithms/shor.html
15) https://en.wikipedia.org/wiki/Shor%27s_algorithm
16) https://news.mit.edu/2016/quantum-computer-end-encryption-schemes-0303
17) https://www.protocol.com/manuals/quantum-computing/quantum-computers-wont-break-encryption-yet#toggle-gdpr
18) https://www.quintessencelabs.com/blog/breaking-rsa-encryption-update-state-art/#:~:text=It%20would%20take%20a%20classical,RSA%2D2048%20bit%20encryption%20key.

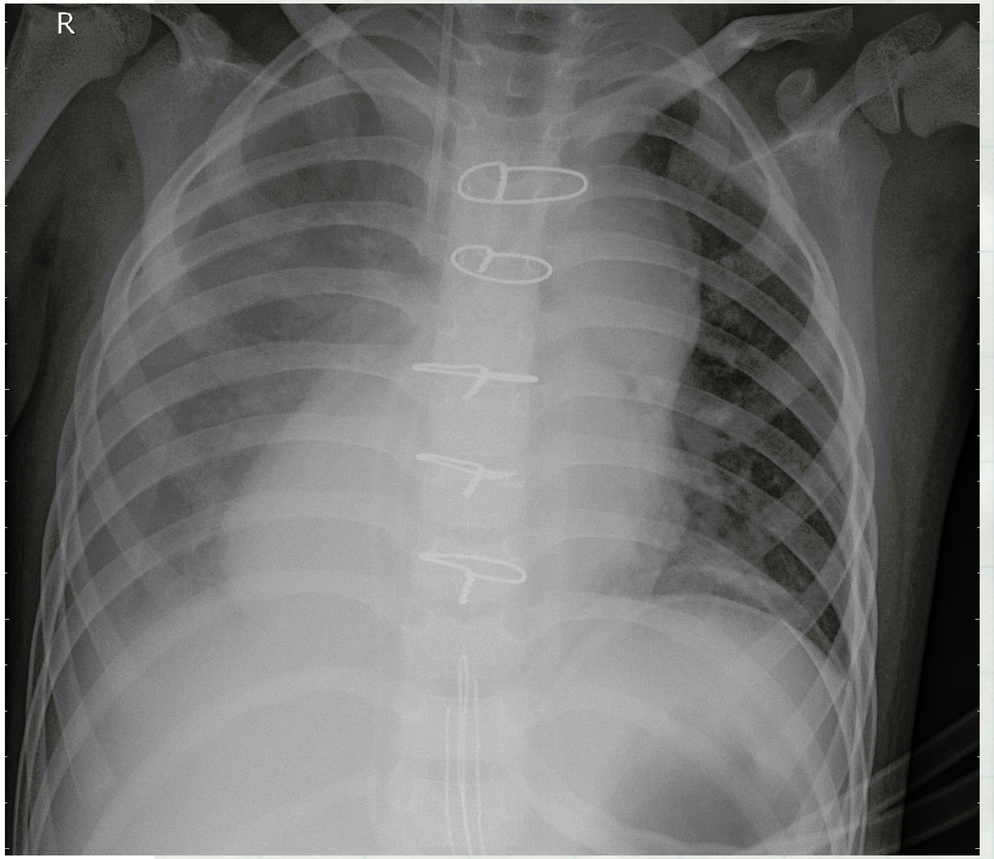
Shor's Algorithm

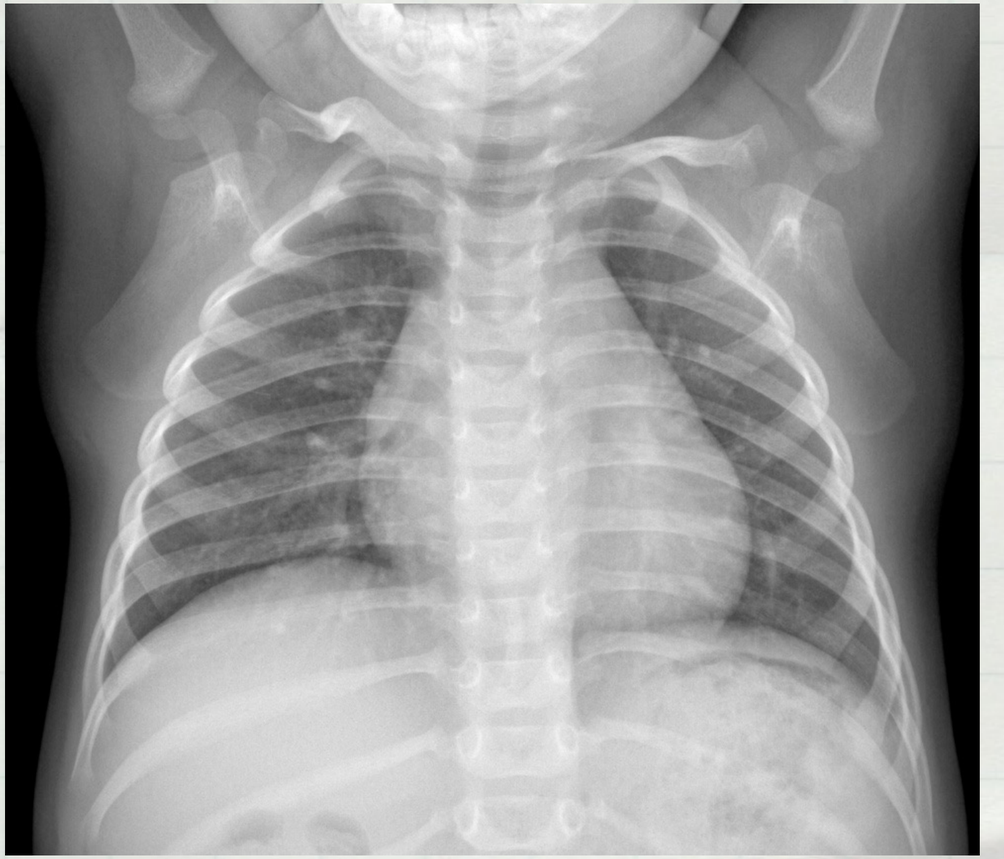
Stephanie Long

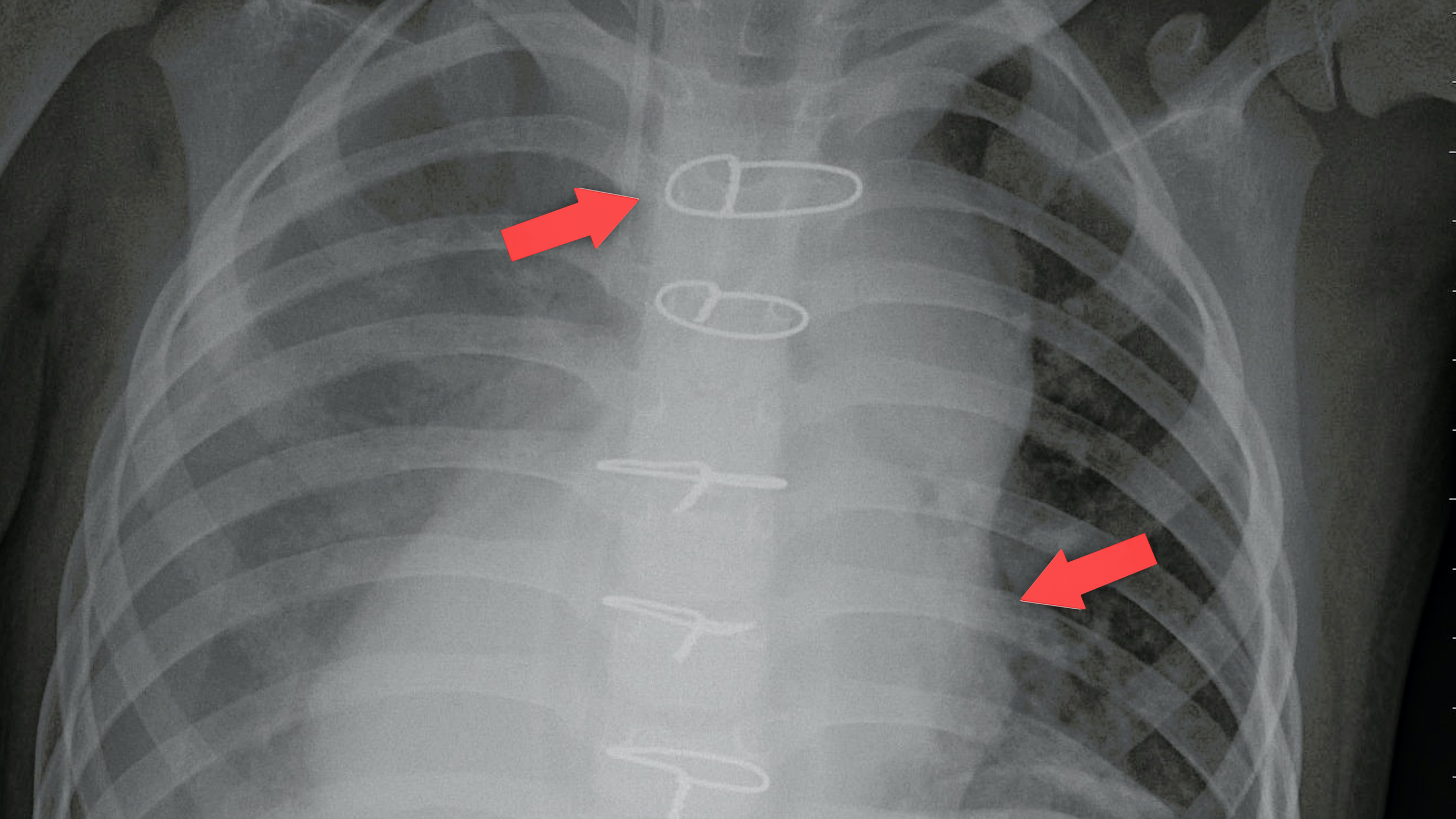
Quantum Image Processing

Anisha Musti

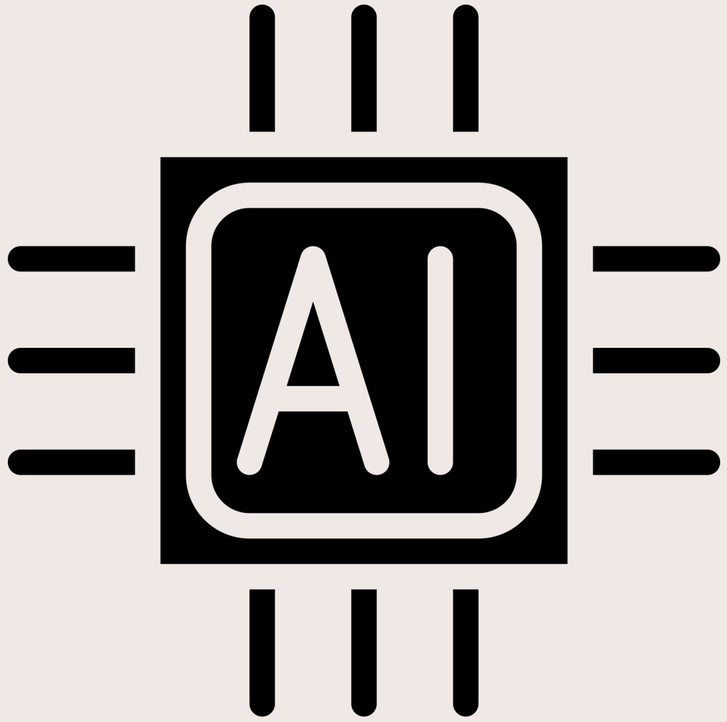
2021

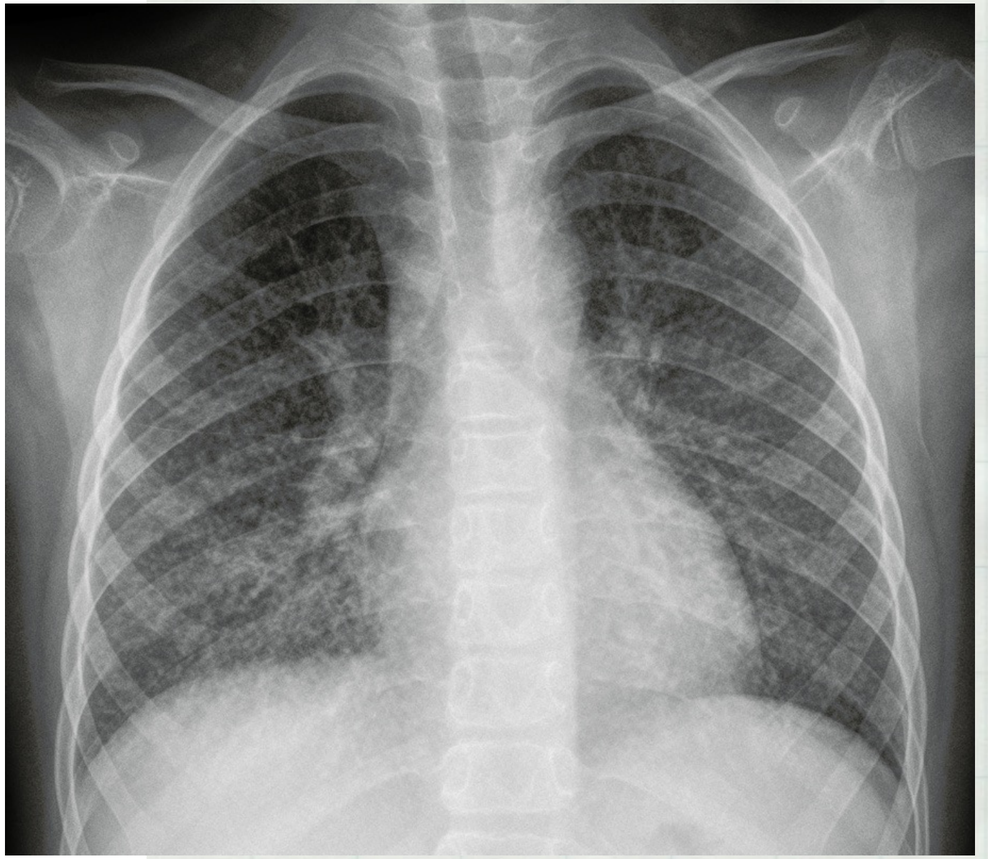

R





Current Solution





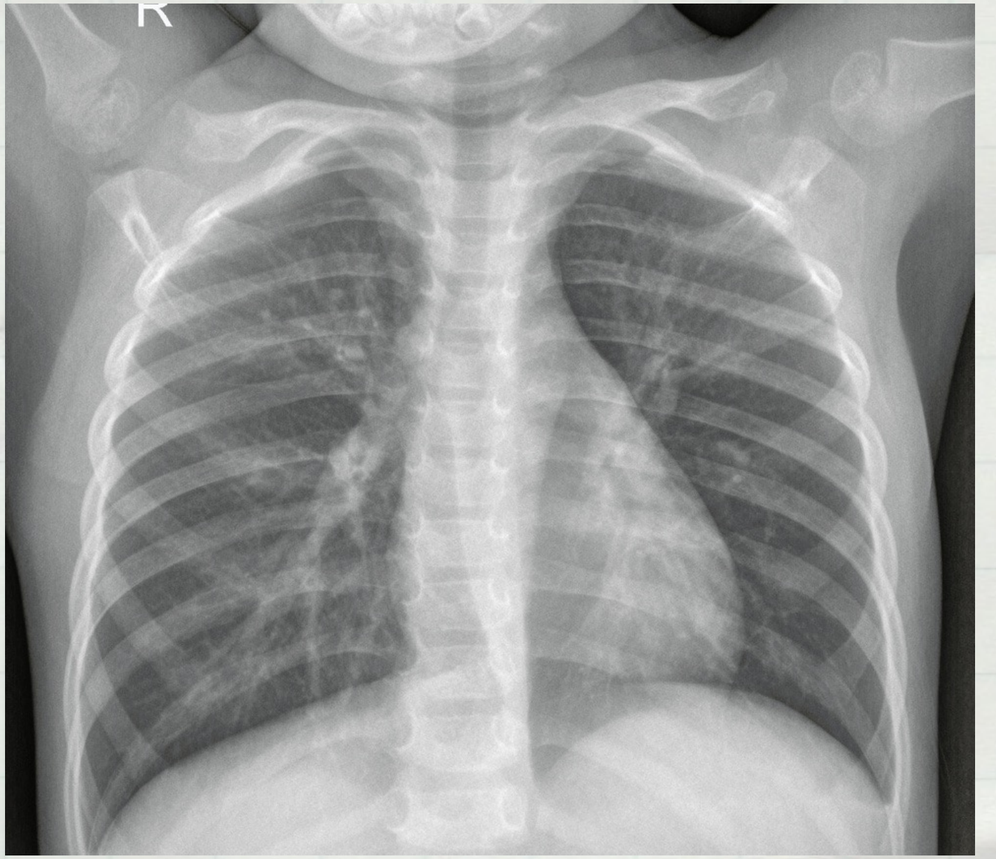

R

Image Matching with Qiskit

01 **Classical Image Preparation**

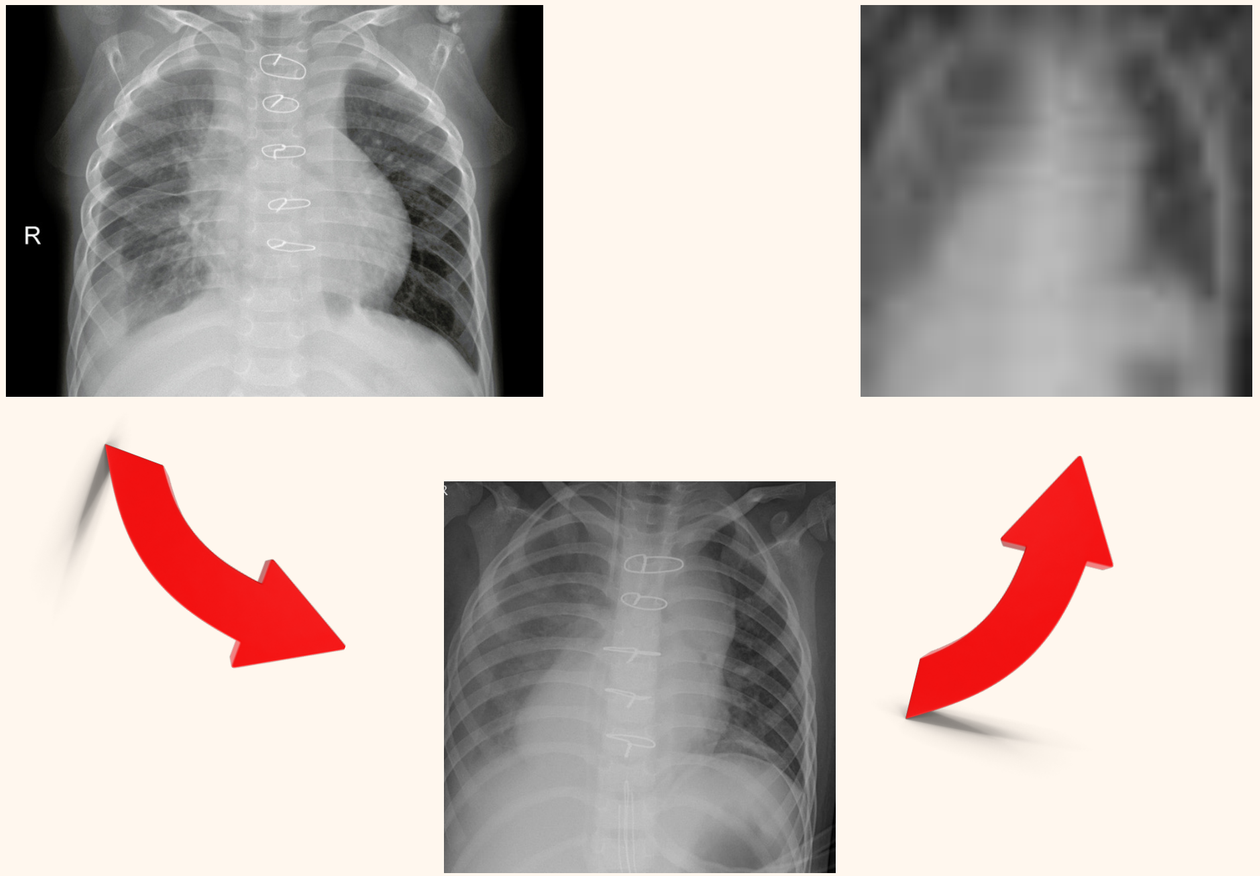
02 **Quantum Encoding**

03 **Quantum Edge Detection**

04 **Quantum Matching**

1. Image Preparation

1 2 3 4



What?

Grey scale

Square

32x32 pixels (10qbits)

Why?

#qubits <32

#gates (depth)

2. Quantum Encoding FRQI

$$|I(\theta)\rangle = \frac{1}{2^n} \sum_{i=0}^{2^{2n}-1} (\cos\theta_i |0\rangle + \sin\theta_i |1\rangle) \otimes |i\rangle$$

$$\theta_i \in \left[0, \frac{\pi}{2}\right], i = 0, 1, \cdots, 2^{2n} - 1$$

The FRQI state is a normalized state from the equation to the left and is made of two parts:

- color information encoding
- associated pixel position encoding

A 2x2 image would have the following graph.

Corresponding θ angles (color encoding) and associated kets (position encoding).

$\theta_0, \|00\rangle$	$\theta_1, \|01\rangle$
$\theta_2, \|10\rangle$	$\theta_3, \|11\rangle$

2. Quantum Encoding FRQI

What?
Classical bit / pixels ⟶ qubits

Why?
Advantage. m pixels ⟶ n = log2 (m) qubits

But
Data loading: m gates ⟶ m pixels

2. Quantum Encoding FRQI

Alternative: Log Concave

- Reduce #gates
- Assumption: log concave probability distribution
- Classical probability distribution mapped to probability amplitude of quantum states.

Encoding	Classical	Quantum FRQI	Quantum Log concave
Pixels	m bits	$\log_2 m$ qubits	$\log_2 m$ qubits
Gates	-	m	$\log_2 m$

3. Quantum edge detection

1 2 3 4

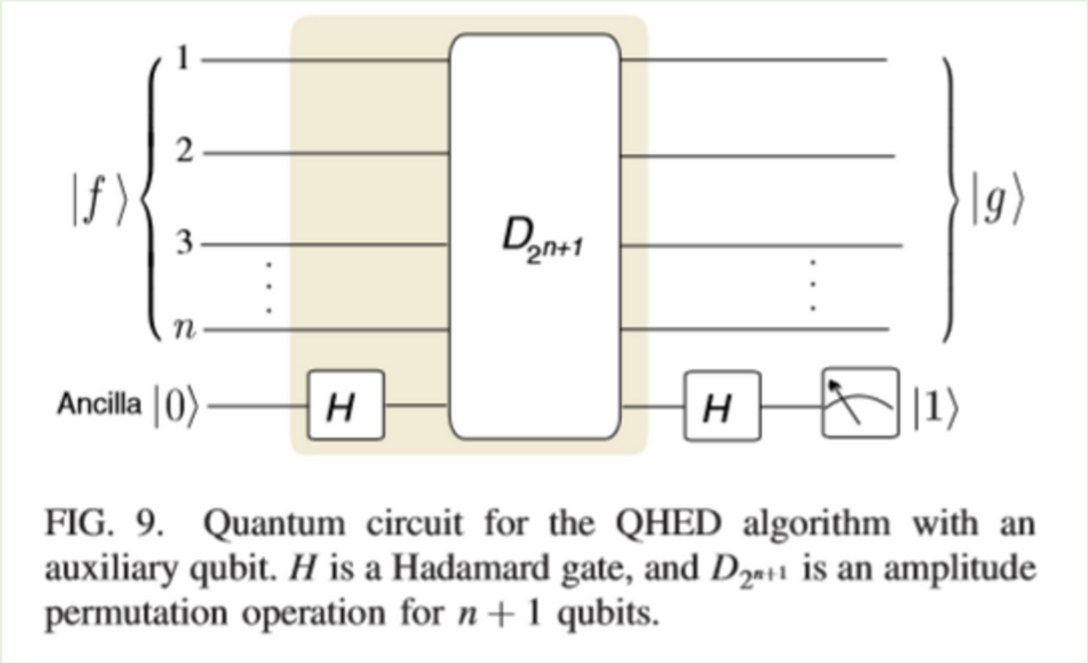


FIG. 9. Quantum circuit for the QHED algorithm with an auxiliary qubit. H is a Hadamard gate, and $D_{2^{n+1}}$ is an amplitude permutation operation for $n+1$ qubits.

Source: Yao, X. W., Wang, H., Liao, Z., Chen, M. C., Pan, J., Li, J., ... & Zheng, W. (2017). Quantum image processing
and its application to edge detection: Theory and experiment. Physical Review X, 7(3), 031041.

3. What is an edge?

$$\alpha = [0, 0, 0, 1, 1, 1, 0, 0, 0],$$

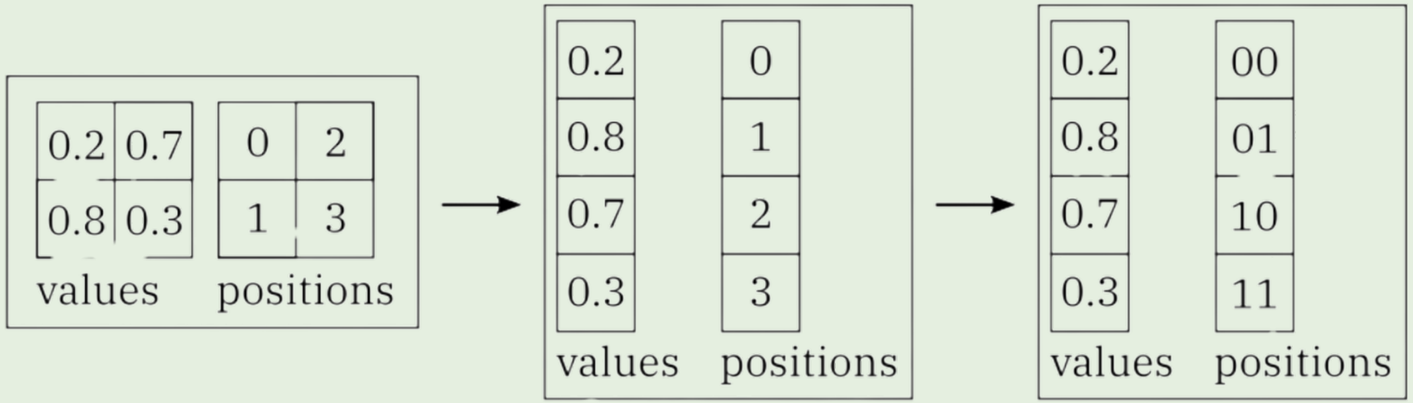
$$\Delta\alpha = [0, 0, 1, 0, 0, -1, 0, 0, 0]$$

The Delta A is the difference of a pixel's nearest neighboring pixels

The differences (Delta A) take on non-zero values where there are changes (edges) in the original image.

So, a Delta A indicates an edge.

3. Image into statevector

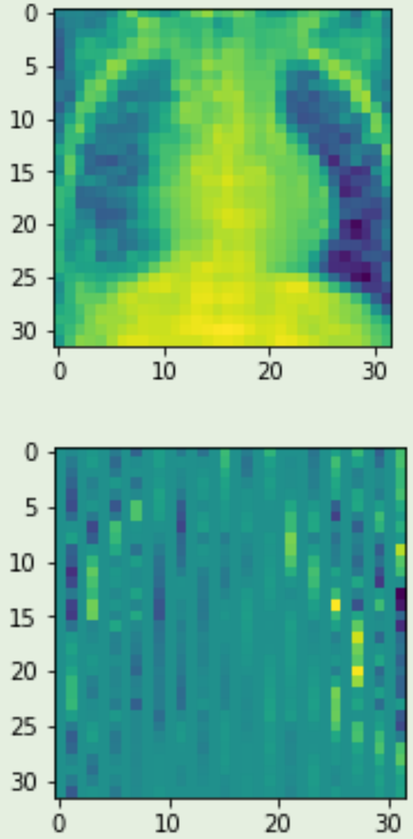


The image begins as a series of values corresponding to pixel positions. As you unravel the matrices to form vectors, you rewrite the pixels from decimals to binary.

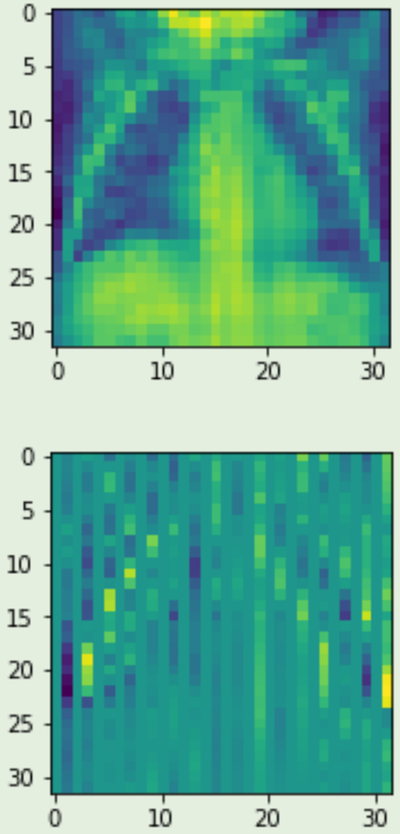
That eventually corresponds to the quantum state of the system.

$$|\text{image}\rangle = \frac{0.2|00\rangle + 0.8|01\rangle + 0.7|10\rangle + 0.3|11\rangle}{\sqrt{0.2^2 + 0.8^2 + 0.7^2 + 0.3^2}}$$

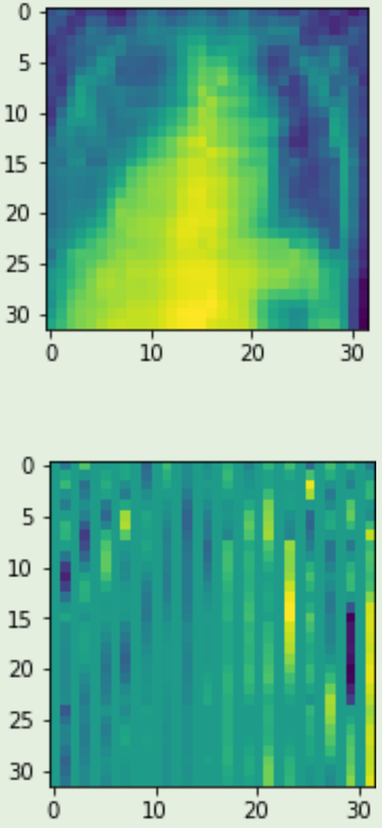
3. Results



TEST



NORMAL

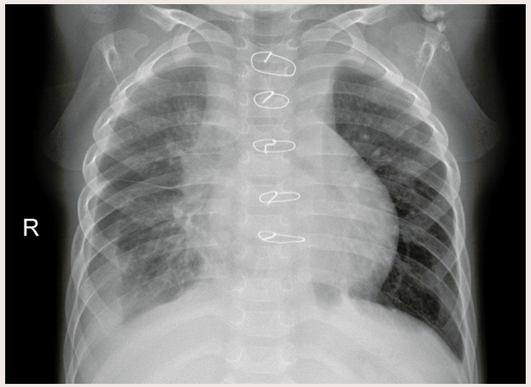


PNEUMONIA

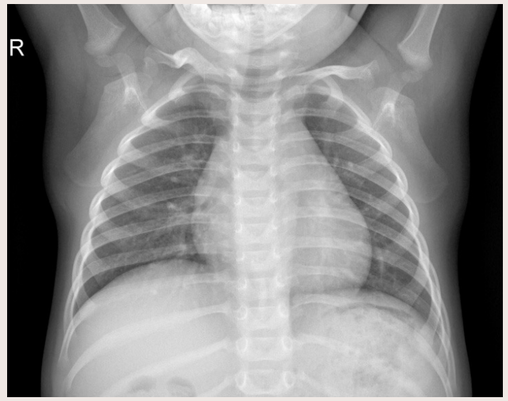
4. Quantum Matching

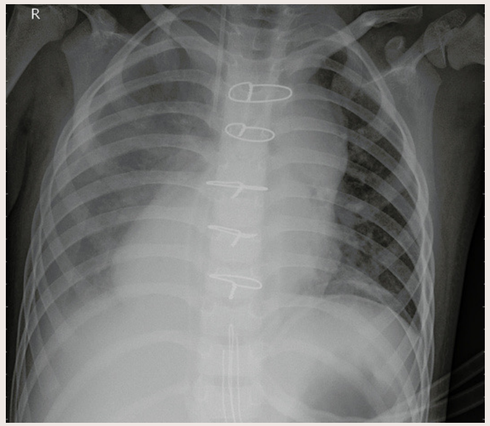
1 2 3 4

Problem Picture

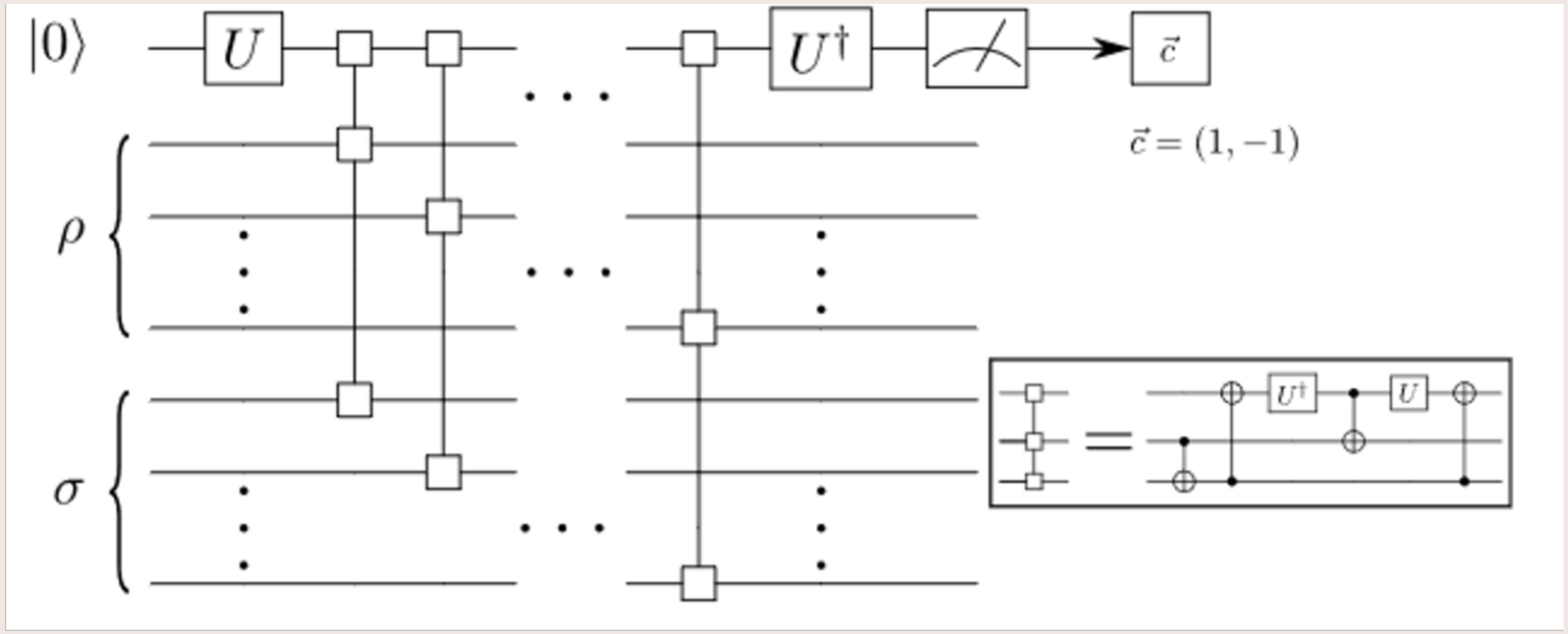


Two Options



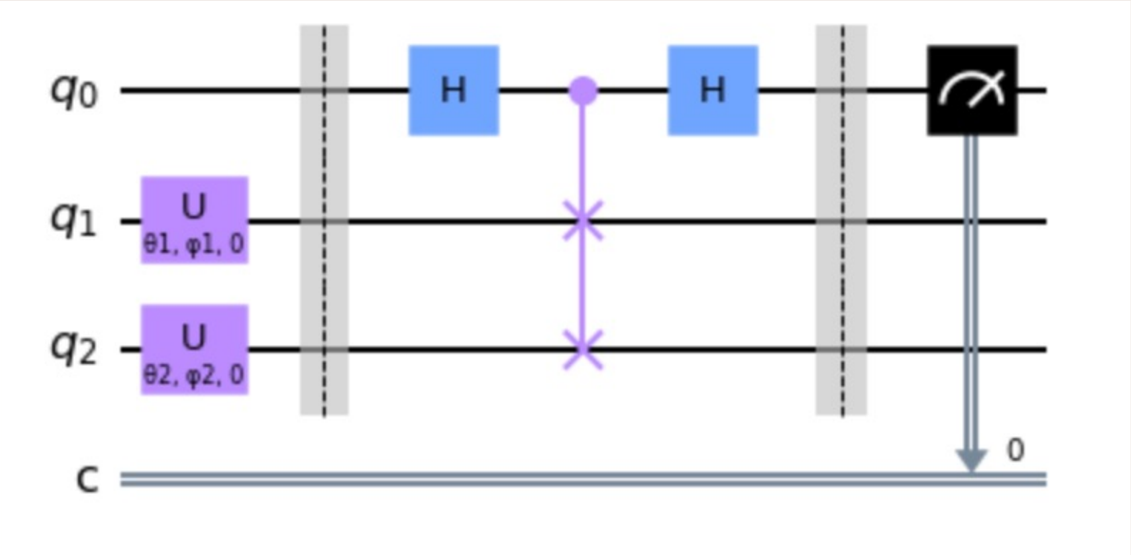


4. The Circuit



Source: Cincio, L., Subaşı, Y., Sornborger, A. T., & Coles, P. J. (2018). Learning the quantum algorithm for state overlap. New Journal of Physics, 20(11), 113022.

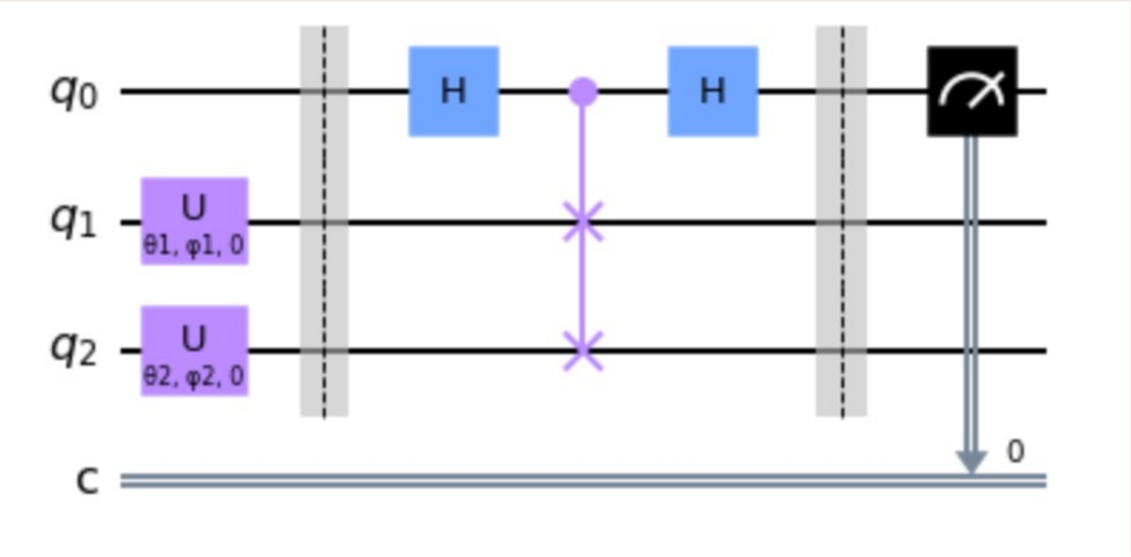
4. My Circuit



The qubits used to encode both the test and reference images represent the density matrices for each image.

These density matrices can then be compared via a quantum based similarity test.

4. SWAP Test

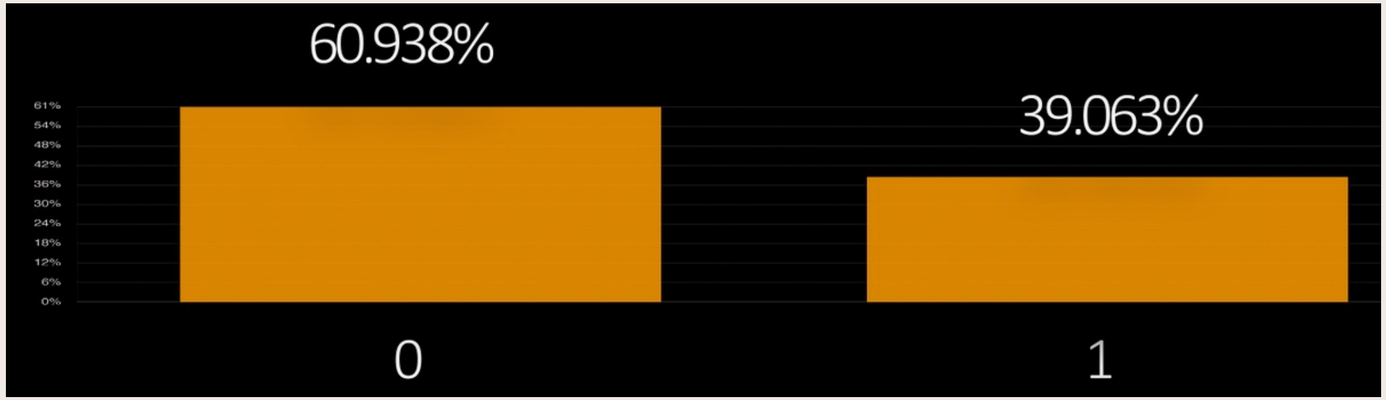


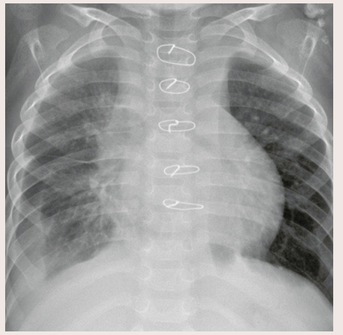
Two of these components have the ancilla in the |1> state, two have it in |0>; the probability of measuring |1> is related to the inner product of the input states.

If the states are parallel, we always measure 0 (useful for results later). Otherwise, there's a nonzero chance of measuring 1.

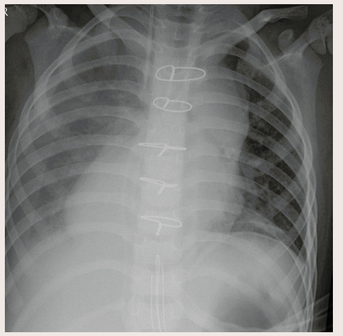
4. Results

Shots 1024
Simulation time taken 34m 4.8s





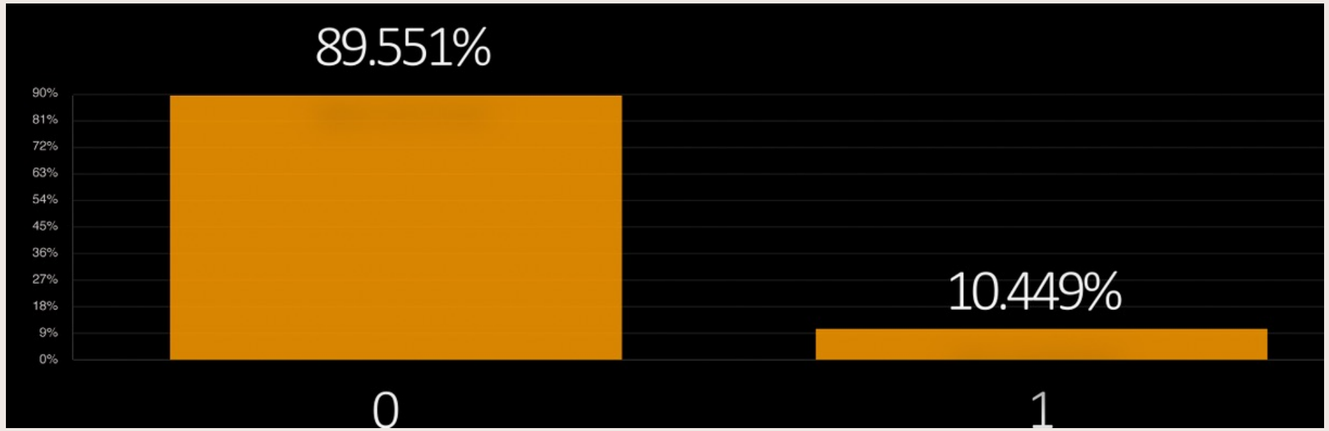
Problem Image

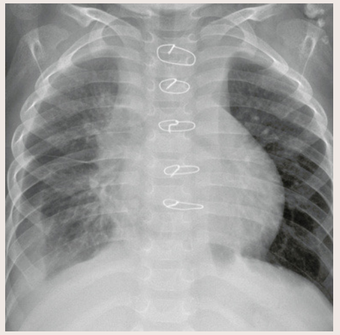


SWAP Image

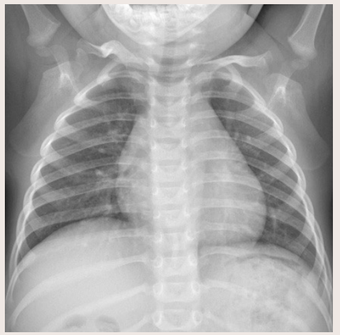
4. Results

Shots 1024
Simulation time taken 30m 18.5s



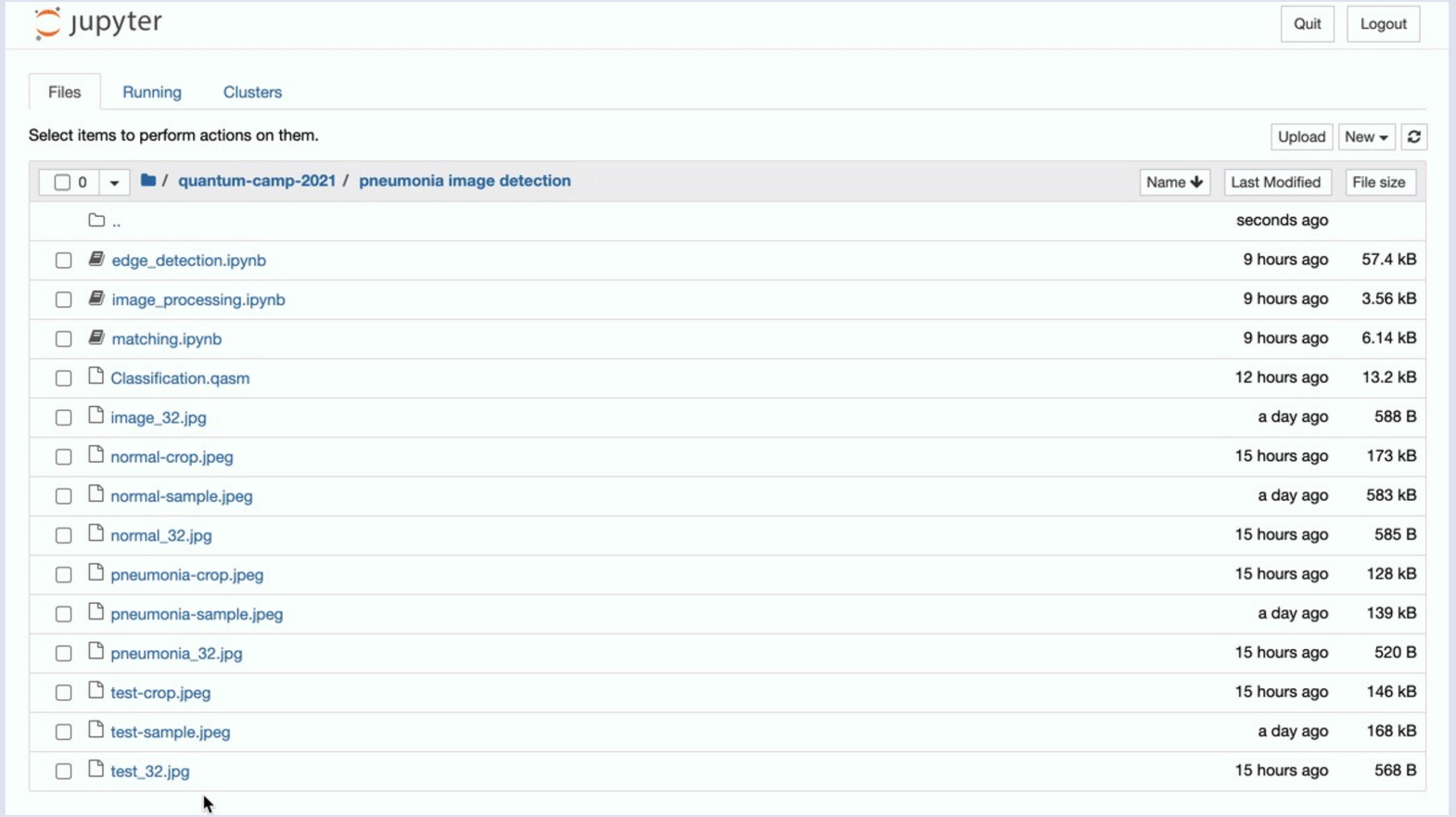


Problem Image

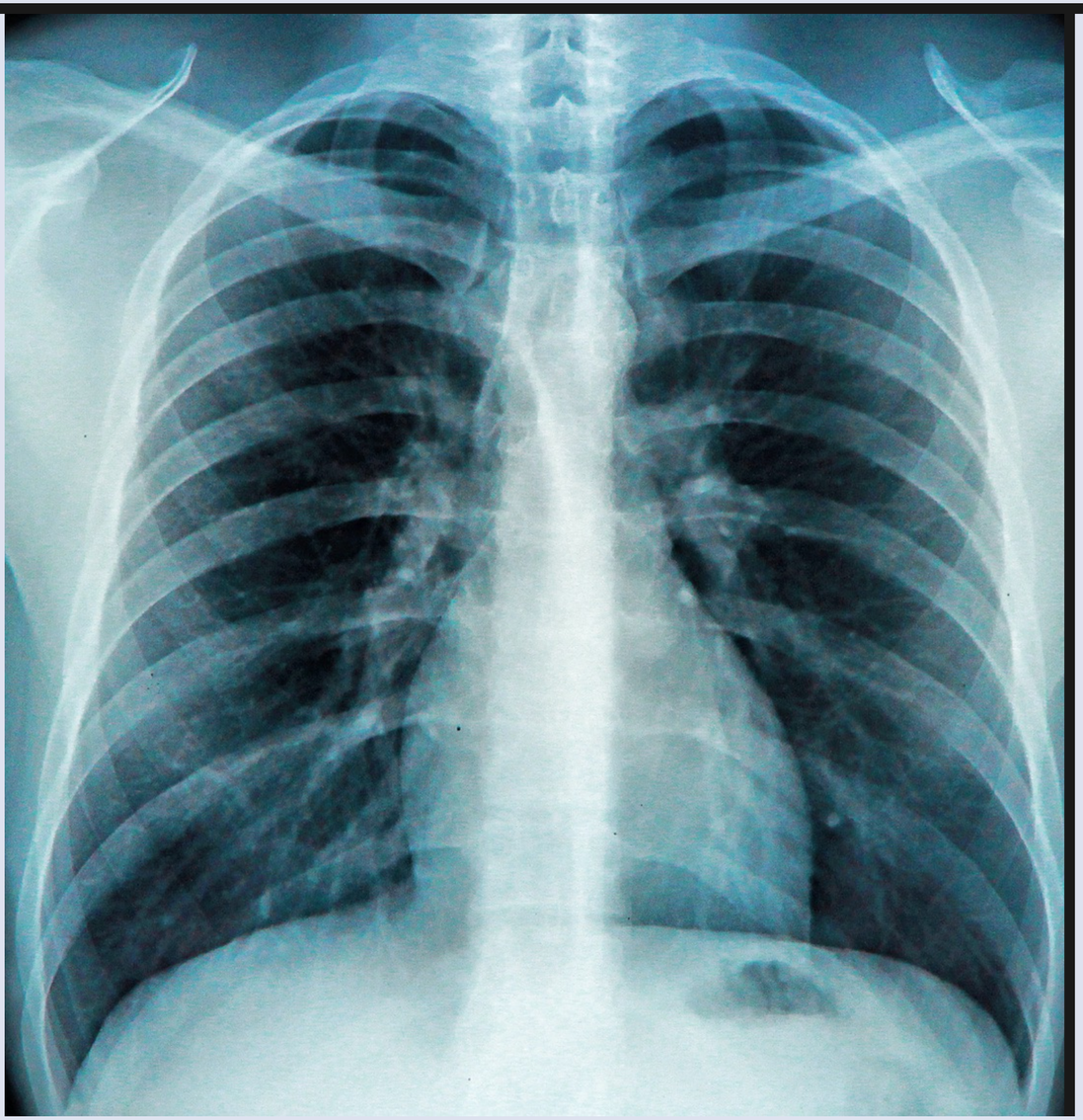


SWAP Image

Qiskit Implementation



Conclusion



- Algorithm showed that the patient was more similar to one who had pneumonia as opposed to not
- Classical AI still possesses an advantage with image similarity through deep learning. However, advances in hardware may change that.
- Further research involves mapping the images on near-term algorithms such as QAOA or VQE
- The algorithm can be applied to other forms of medical images such as tumors

References

[1] Yao, X. W., Wang, H., Liao, Z., Chen, M. C., Pan, J., Li, J., ... & Zheng, W. (2017).
Quantum image processing and its application to edge detection: Theory and experiment. Physical Review X, 7(3), 031041.

[2] Le, P. Q., Dong, F., & Hirota, K. (2011).
A flexible representation of quantum images for polynomial preparation, image compression, and processing operations.
Quantum Information Processing, 10(1), 63-84.

[3] Le, Q. P., Fangyang, D., Yoshinori, A., & Kaoru, H. (2009).
Flexible representation of quantum images and its computational complexity analysis. I
In 日本知能情報ファジィ学会 ファジィ システム シンポジウム 講演論文集
第 25 回ファジィ システム シンポジウム (pp. 185-185). 日本知能情報ファジィ学会

[4] Yuan, Suzhen & Venegas-Andraca, Salvador & Wang, Yuchan & Luo, Yuan & Mao, Xuefeng. (2019).
Quantum Image Edge Detection Algorithm. International Journal of Theoretical Physics. 10.1007/s10773-019-04166-9.

[5] Gómez-Moreno, H., Maldonado-Bascón, S., López-Ferreras, F., Acevedo-Rodríguez, F. J., & Martín-Martín, P. (2001).
Edge detection by using the support vector machines. training, 2, 3. [6] Cincio, L., Subaşı, Y., Sornborger, A. T., & Coles, P. J. (2018).
Learning the quantum algorithm for state overlap. New Journal of Physics, 20(11), 113022.

Quantum Money

Shagun Khare

Defining The Problem

What Is Quantum Money?

Protocol to create and validate banknotes impossible to forge

Two Sectors:

- Private-key
- Public-key

Stephen Wiesner (1942-2021)

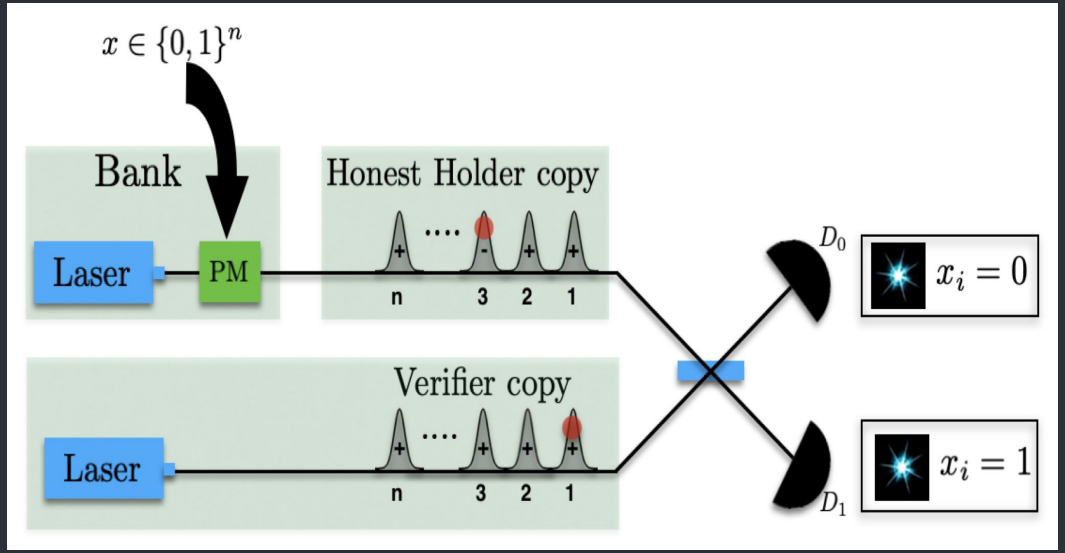
Wiesner's Quantum Money Scheme

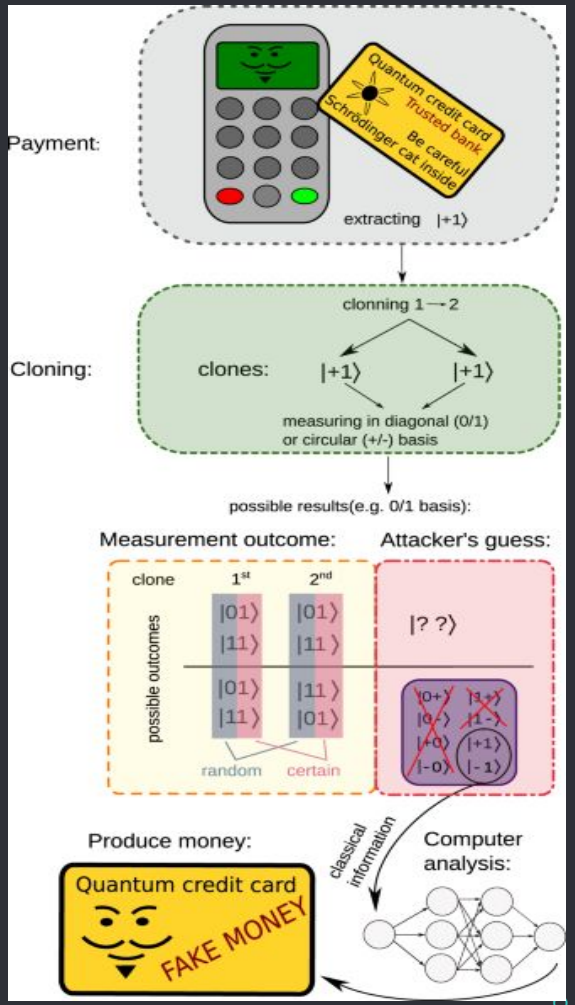
- Central bank prints "quantum bills"
  - » $|0\rangle$,
    » $|1\rangle$,
    » $|+\rangle = (|0\rangle+|1\rangle)/\sqrt{2}$, or
    » $|-\rangle = (|0\rangle-|1\rangle)/\sqrt{2}$.
- Measures each qubit in the correct basis
- Counterfeiter would fail with a a probability of $(3/4)^N$ with n qubits

Further Study In Weisner's Model

- Chance of getting "stolen" by an adversary
- Tokunada (2003) → prevent mint from tracking each bill
- Mosca and Stebila (2007) → *quantum coins*

Classical Variation





Qiskit Implementation

```
def create_money(bits, bases):
    message = []
    for i in range(3):
        qc = QuantumCircuit(1,1)
        if bases[i] == 0: # Prepare qubit in Z-basis
            if bits[i] == 0:
                pass
            else:
                qc.x(0)
        else: # Prepare qubit in X-basis
            if bits[i] == 0:
                qc.h(0)
            else:
                qc.x(0)
                qc.h(0)
        qc.barrier()
        message.append(qc)
    return message
```

```
bank_fixednote = np.array([1,0,0])
bank_fixedbase = np.array([0,1,1])

banknote = create_money(bank_fixednote, bank_fixedbase)

bank_bases = randint(2, size=3)
```

```
def measure_message(message, bases):
    backend = Aer.get_backend('aer_simulator')
    measurements = []
    for q in range(3):
        if bases[q] == 0: # measuring in Z-basis
            message[q].measure(0,0)
        if bases[q] == 1: # measuring in X-basis
            message[q].h(0)
            message[q].measure(0,0)
        aer_sim = Aer.get_backend('aer_simulator')
        qobj = assemble(message[q], shots=1, memory=True)
        result = aer_sim.run(qobj).result()
        measured_bit = int(result.get_memory()[0])
        measurements.append(measured_bit)
    return measurements
```

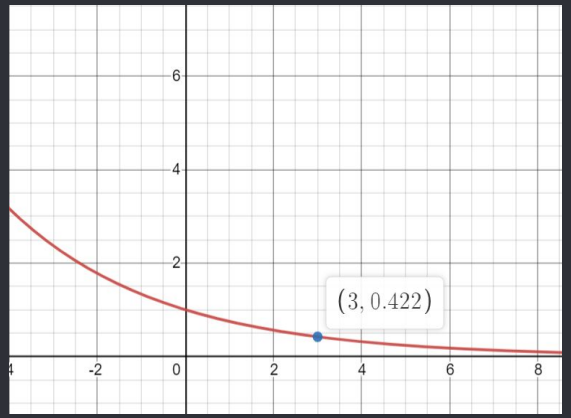
```
prob = 0
shots = 100

for i in range(100):
    forge_bits = randint(2, size=3)
    forge_bases = randint(2, size=3)
    forge_note = create_money(forge_bits, forge_bases)
    bank_measure = measure_message(forge_note, bank_bases)
    bank_fixed = measure_message(banknote, bank_fixedbase)
    if(bank_fixed==bank_measure):
        print("Bank Serial Number: " + str(bank_fixed))
        print("Verfied Bill: " + str(bank_measure))
    else:
        print("Bank Serial Number: " + str(bank_fixed))
        print("Forged Bill: " + str(bank_measure))
        prob = prob+1
    print("")

print("Probability of bills being forged:" + str(prob/shots))
```

Example Result:

```
Bank Serial Number: [1, 1, 1]
Verfied Bill: [1, 1, 1]
Bank Serial Number: [1, 1, 1]
Forged Bill: [0, 1, 0]
Bank Serial Number: [1, 1, 0]
Forged Bill: [0, 1, 1]

Bank Serial Number: [1, 0, 1]
Forged Bill: [0, 1, 0]

Probability of bills being forged:0.42
```

Proof:

(¾)^3 = 0.421875

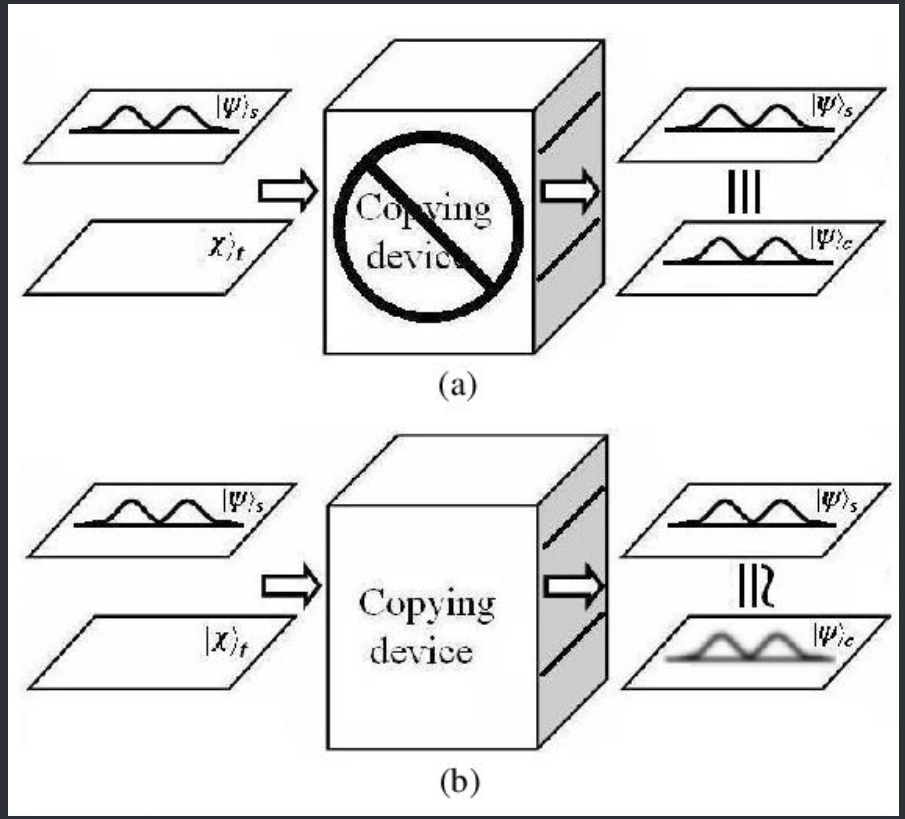


Advantages of this Scheme

- Coins become exponentially harder to counterfeit as attempts increase
- Bank's database can be static
- Does not require 3-party authentication
- Dependence between the number of verifications and the number of qubits it contains is optimal

No-Cloning Theorom

Cannot create a copy of an arbitrary quantum state



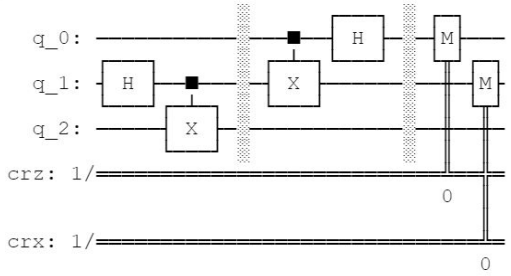
Qiskit Implementation

```
bill = QuantumRegister(3, name="q")
crz = ClassicalRegister(1, name="crz")
crx = ClassicalRegister(1, name="crx")
bank_circuit = QuantumCircuit(bill, crz, crx)
```

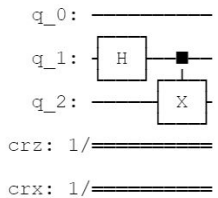
```
def create_bell_pair(qc, a, b):
    qc.h(a)
    qc.cx(a,b)
```

```
def measure_and_send(qc, a, b):
    """Measures qubits a & b and 'sends' the results to Bob"""
    qc.barrier()
    qc.measure(a,0)
    qc.measure(b,1)
```

```
measure_and_send(bank_circuit, 0 ,1)
bank_circuit.draw()
```

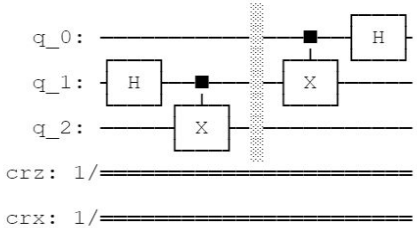



```
create_bell_pair(bank_circuit, 1, 2)
bank_circuit.draw()
```



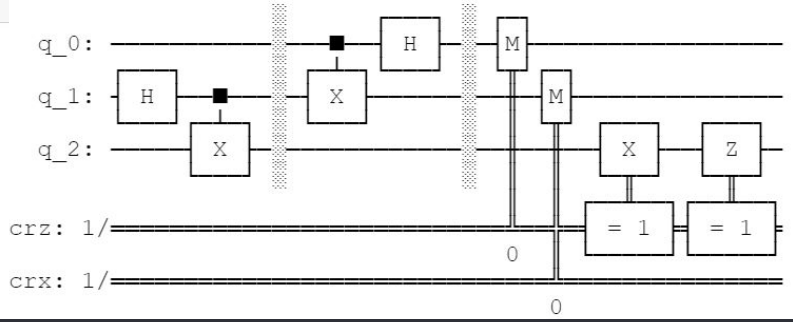

```
def alice_bill(qc, psi, a):
    qc.cx(psi, a)
    qc.h(psi)
```

```
bank_circuit.barrier()
alice_bill(bank_circuit, 0, 1)
bank_circuit.draw()
```




```
def bank_apply(qc, qubit, crz, crx):
    qc.x(qubit).c_if(crx, 1)
    qc.z(qubit).c_if(crz, 1)
```

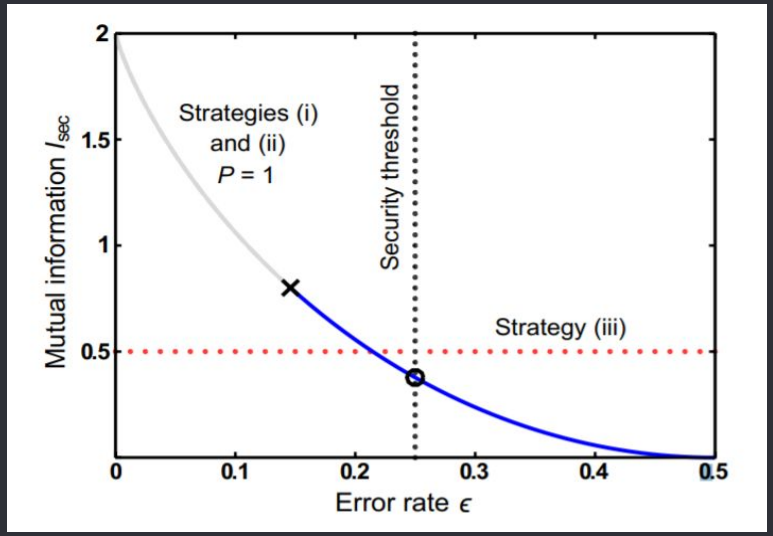
```
bank_apply(bank_circuit, 2, crz, crz)
bank_circuit.draw()
```

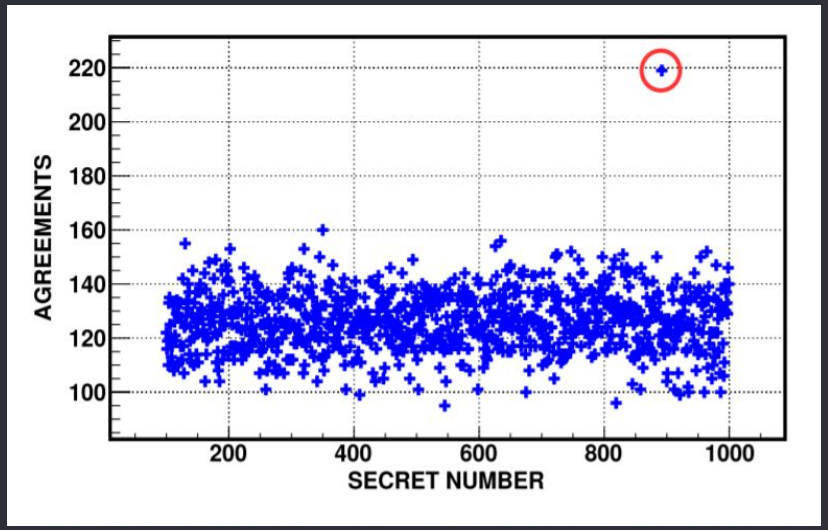


Benefits of this Model

- Obtain exponential security
- Produce notes that are publicly verifiable
- Cannot copy even with access to a verifier

Possible Attacks and Security Measures





(Jiráková, K. and Lemr, K.)

Future Outlook on Quantum Money

- Implementations of Quantum Key Distribution
- Practical implementation not proved yet
- Government and commercial sectors

References

1. Aaronson, S. Quantum copy-protection and quantum money. In Annual IEEE Conference on Computational Complexity (2009), 229242.
2. S. Aaronson and P. Christiano. Quantum money from hidden subspaces. In preparation, 2012.
3. A. Lutomirski. An Online Attack Against Wiesner's Quantum Money. http://arxiv.org/abs/1010.0256, 2010.
4. Bennett, C.H., Brassard, G., Breidbart, S. and Wiesner, S. Quantum cryptography, or unforgeable subway tokens. In Advances in CryptologyProceedings of Crypto (1983), volume 82, 267275.
5. Jiráková, K., Bartkiewicz, K. Černoch, A. and Lemr,K. Experimentally attacking quantum money schemes based on quantum retrieval games. (2019)
6. Mosca, M. and Stebila, D. A framework for quantum money. Poster at Quantum Information Processing (QIP) (Brisbane, Australia, 2007).

Thank You!

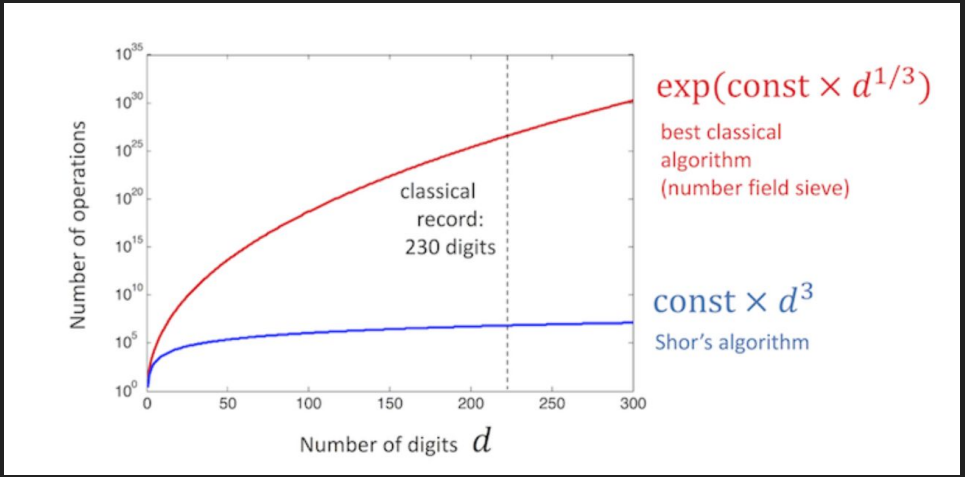
Questions?

Gatemon Qubit Simulations in Qiskit Metal

Krishna Dindial

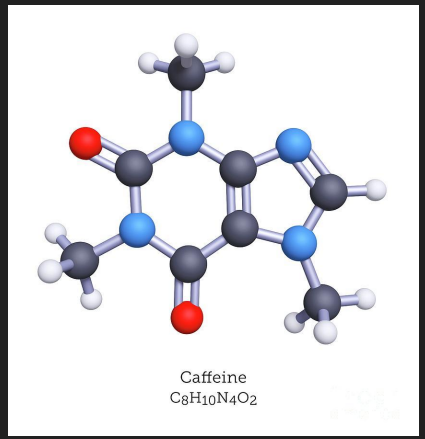
Advantages of a Quantum computer

- Can implement algorithms that take advantage of quantum mechanical effects such as entanglement, superposition and interference



- A 300 qubit computer can be in a superposition of 2^{300} states at once → larger than # atoms in universe

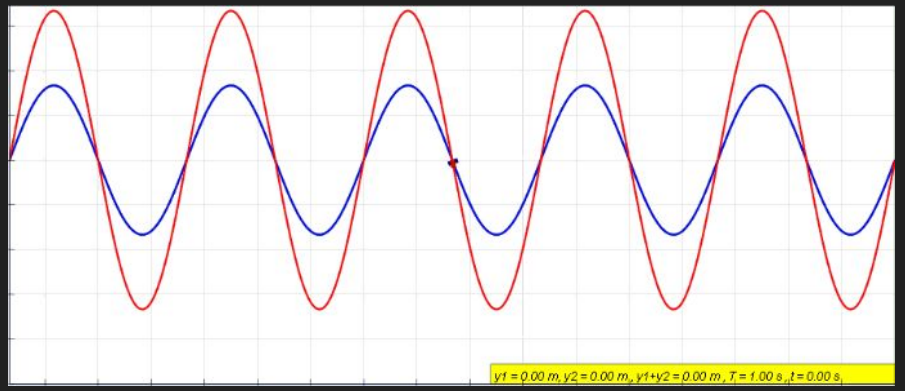
- Can simulate molecular structures that behave quantum mechanically



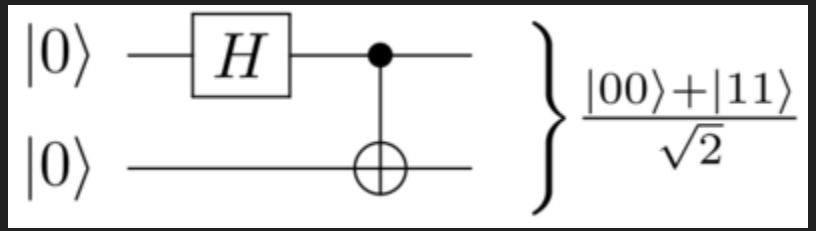
Qubits behave differently

- Can be in a superposition of 0 and 1
- Can interfere with each other
- Can become entangled with each other

Wikipedia: interference



Wikipedia: Entangled Bell State



Ideal Quantum Computer (DiVincenzo Criteria)

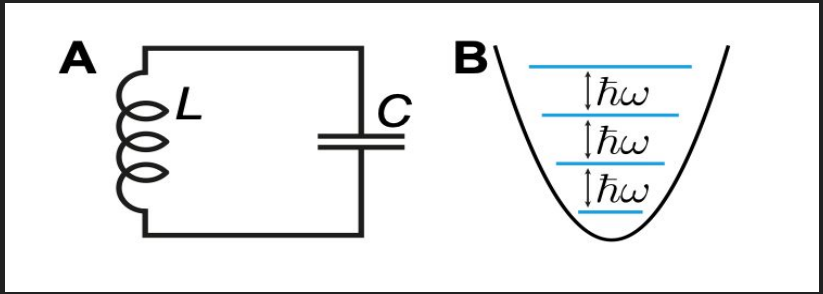
1. Scalable System
2. Long Coherence Times
3. Control
4. Readout
5. Reset

It is hard to achieve all of these things at once because quantum systems are fragile.

- Balancing act between isolating the qubit from its environment, but also being able to interact with it and measure it

LC Oscillator

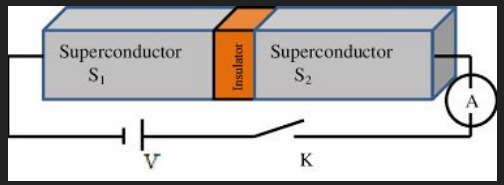
LC oscillator is a very common circuit, very familiar to physicists and electrical engineers

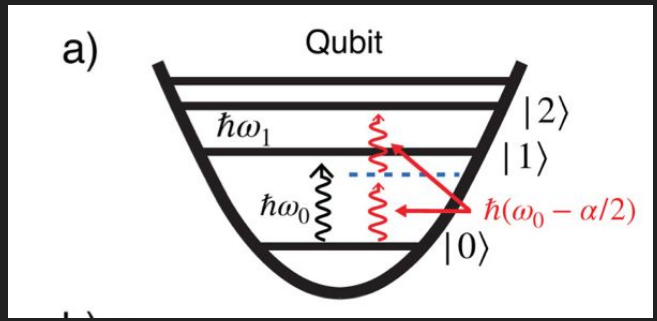


- Does not work as a qubit because energy levels are all evenly spaced out

Transmon Qubit

- By replacing an inductor with a josephson junction, the energy levels become uneven





Pros and cons of transmon

Pros:

- LC circuits are very familiar to physicists and electrical engineers
- Can be controlled and measured with photons microwave (5ghz) pulses

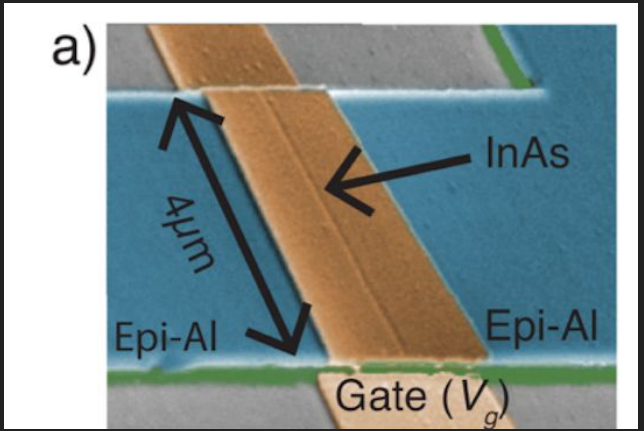
Cons:

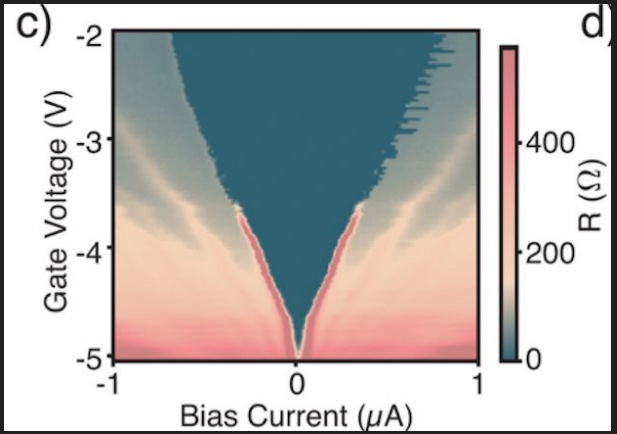
- susceptible to charge noise
- Must be kept in millikelvin fridge
- Cannot be tuned: the energy levels are fixed

Gatemon

- Rather than using a cubit with a fixed frequency, the gatemon uses a voltage tuneable semiconductor to “tune” the qubit to different frequencies

- Can tune qubits in and out of resonance with each-other

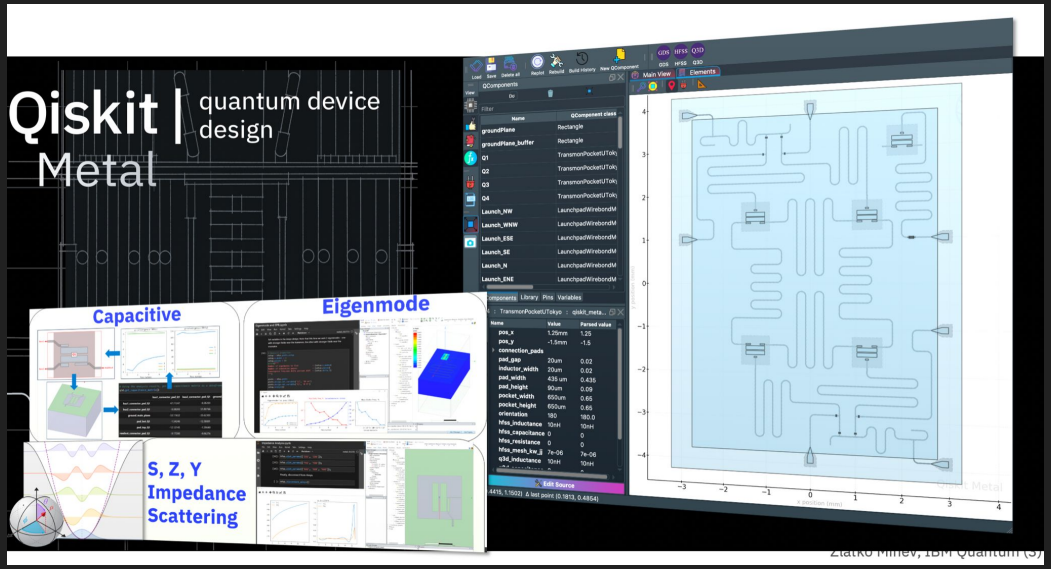




J. Vac. Sci. Technol. A **39**, 033407

IBM Qiskit Metal

- It is expensive to fabricate tuneable junctions and new chip designs
- Qiskit metal lets us build and analyze new designs all on the computer saving time and money
- Once a worthwhile design is found we can fabricate it and test it

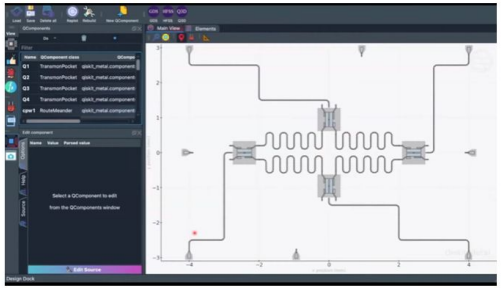


- LOM analysis *Minev et al 2021, arXiv: 2103.103344*
- EPR Analsyis analysis *Minev et al 2021, arXiv: 2010.00620*
-

Qiskit-Metal Work Flow

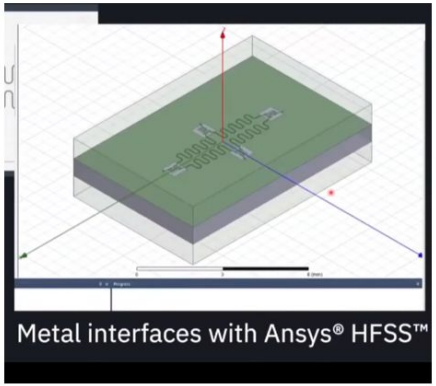
Step 0) Think of a design

Step 1)



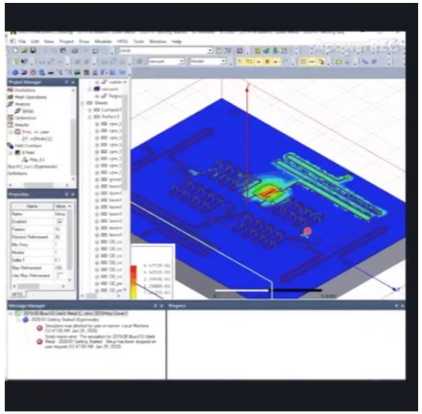
Draw design in Qiskit Metal

Step 2)



Render in HFSS Ansys

Step 3)



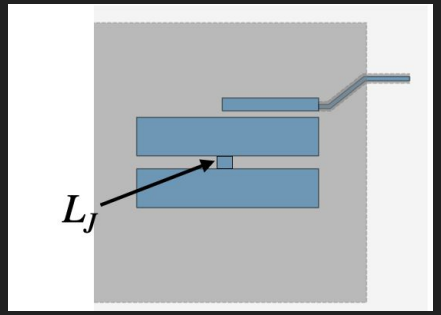
Simulate the circuit

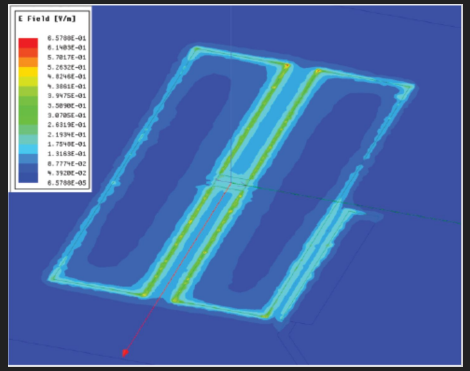
Youtube, Qiskit, “Qiskit Metal E05.1 - Analysis - Capacitance and Frequency Control”

What does the Simulation tell us?

- Qubit Frequency
  - The difference in energy levels between the 0 and 1 state.
- Anharmonicity
  - The difference between the first and second frequency levels
    - For Transmon this is about Ec, For gatemon this is about Ec/4
- Coupling Strength
  - Parameter that tells us how much interaction there is between two components in a chip
- Dispersive shift
  - How does the qubits energy change when you add new components to the chip

Simulating a Tuneable Transmon as a Gatemon

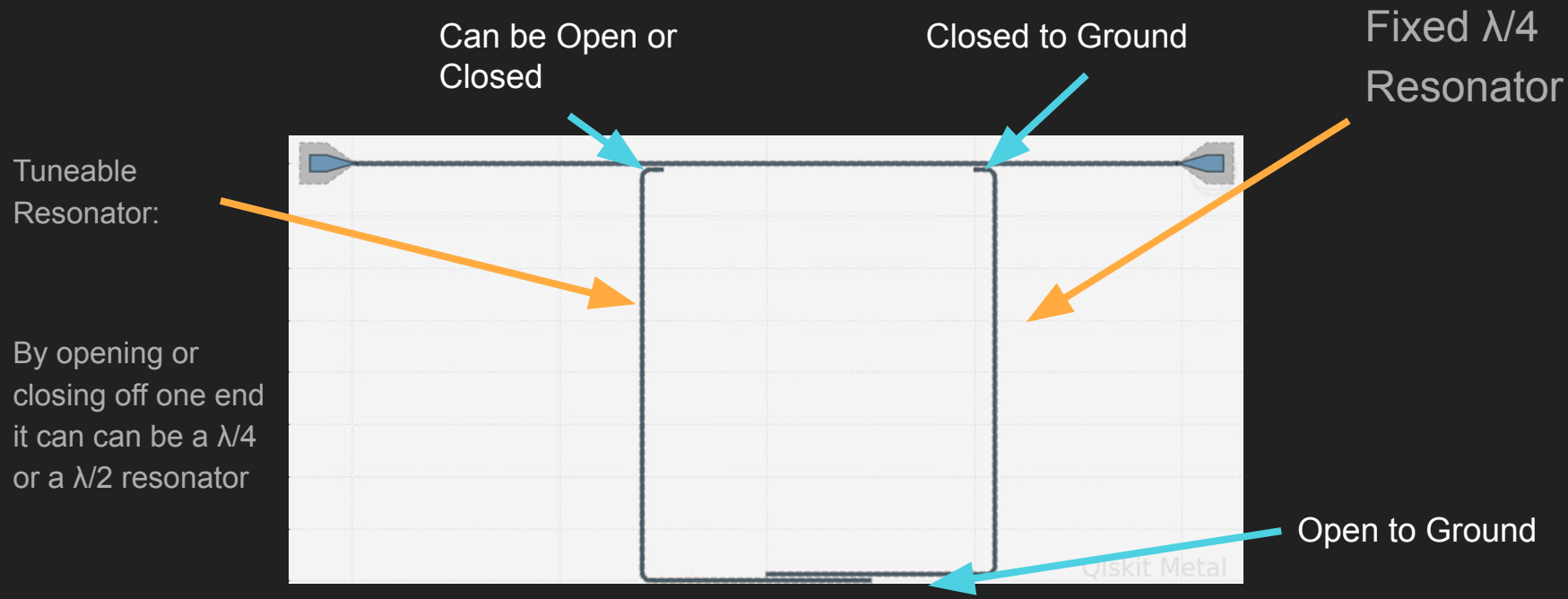




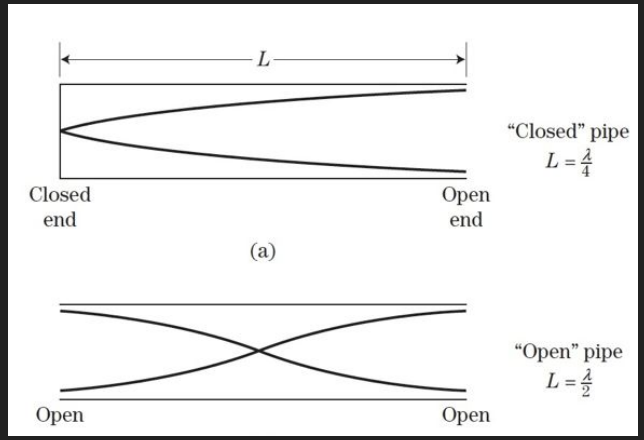
Design Parameters	Qubit Properties			
Josephson Inductance (nH)	Qubit Frequency (GHz)	Anharmonicity (MHz)	Coupling Strength (MHz)	Dispersive Shift (MHz)
10	6.152	370	125	11.3
20	4.241	402	105	1.1
30	3.389	440	95	0.6
40	2.878	477	88	0.5
50	2.527	496	84	0.4

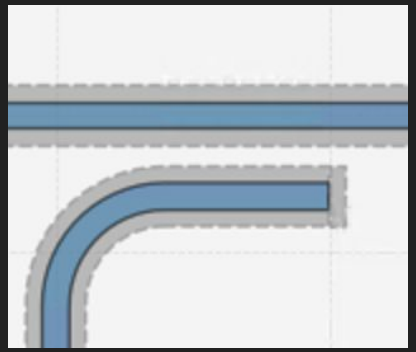
Simulating tunable resonator Cavities

CPW width: 10um
Fixed Resonator length: 3.10mm
Tuneable Resonator 3.13mm

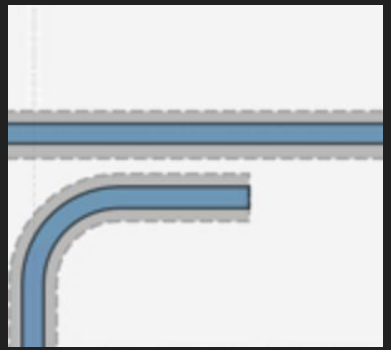


Tuned Resonator Cavities





Closed End

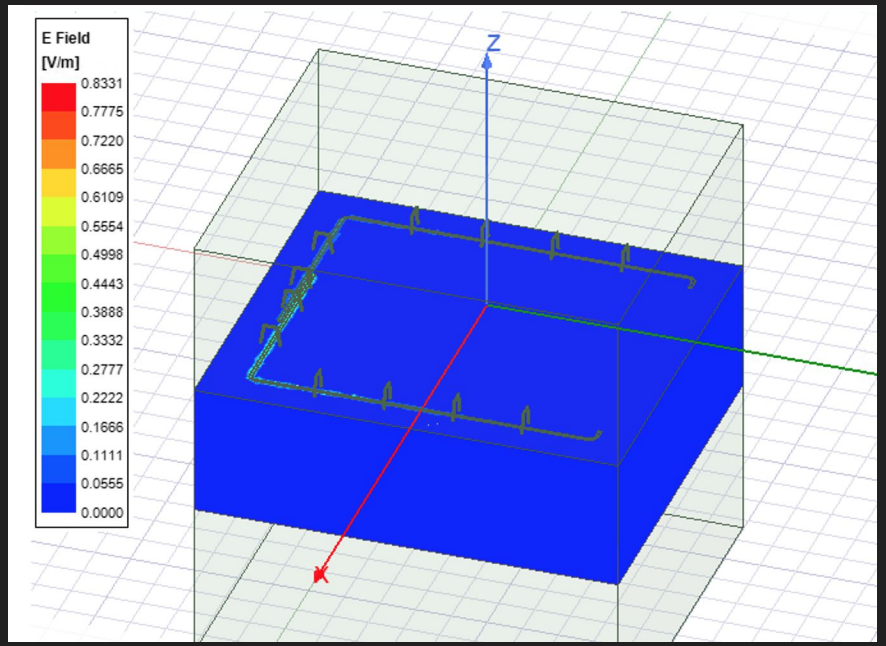


Open End

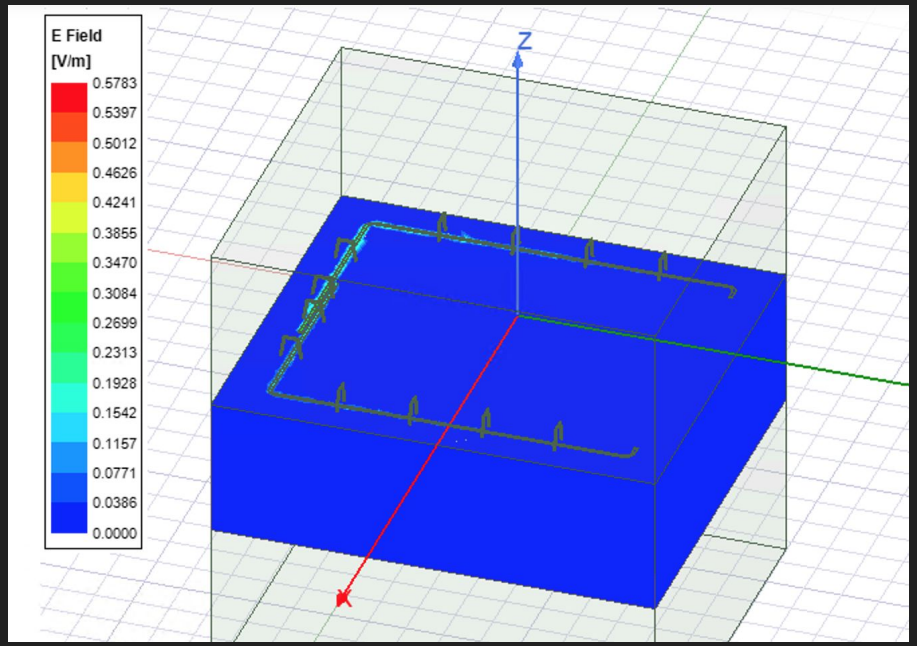
Two λ/4 Resonators

variation	mode	Freq. (GHz)
0	0	9.322328
	1	9.531701

Mode 0

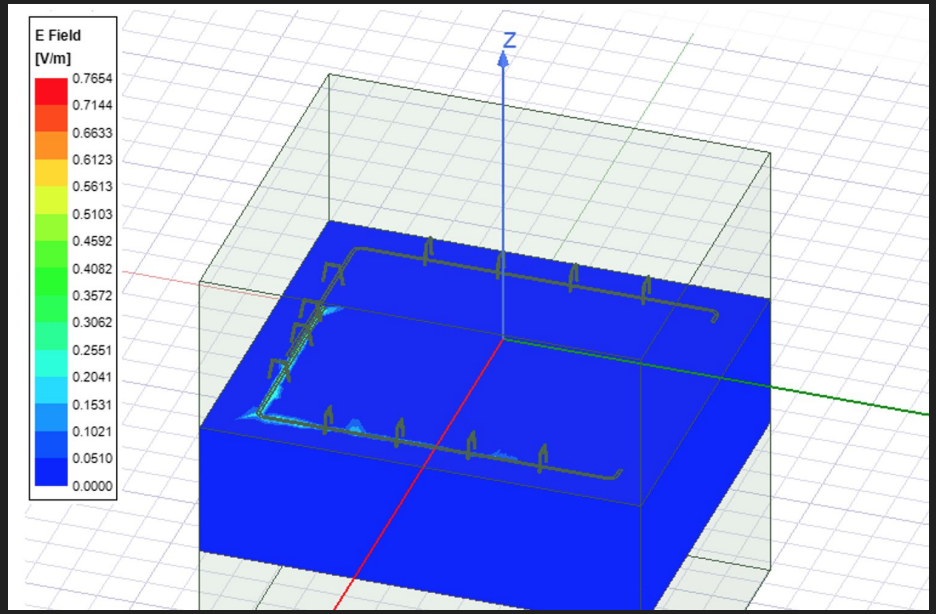


Mode 1



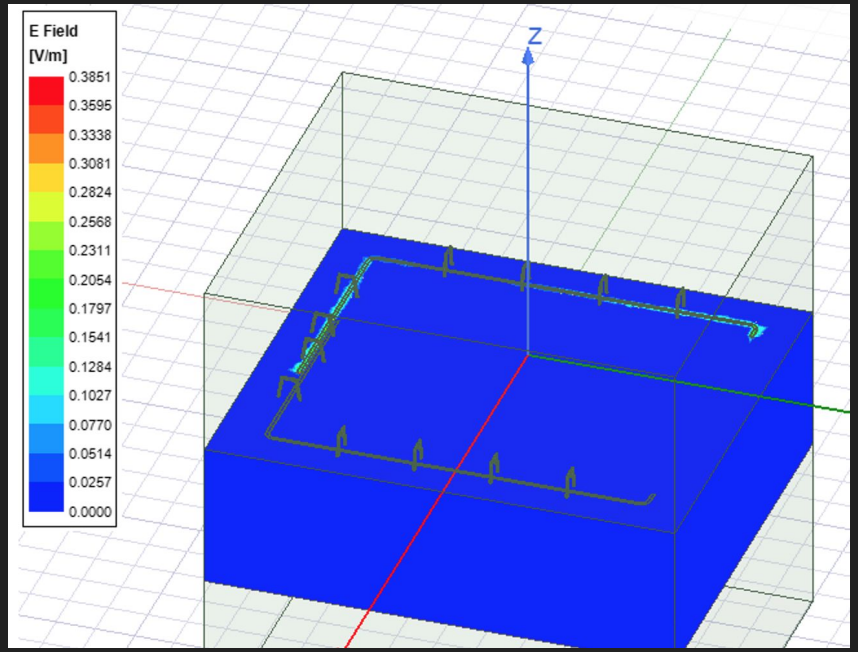
λ/2 Resonator and fixed λ/4 resonator

Mode 0



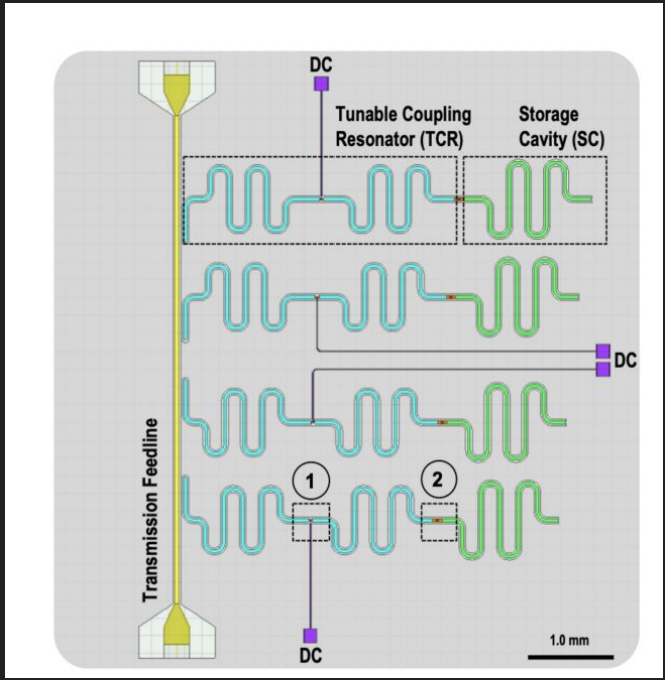
Mode 1

		Freq. (GHz)
variation	mode	
0	0	9.079696
	1	18.377024



Next Steps

- Designing custom components in Qiskit metal
- Tuneable resonators coupled to tuneable qubits
- Multi qubit simulations
- Stay in Touch with Gatemon Team



K. Sardashti et al, "Voltage-Tunable Superconducting Resonators: A Platform for Random Access Quantum Memory," in *IEEE Transactions on Quantum Engineering*, vol. 1, pp. 1-7, 2020, Art no. 5502107