Quantum perceptron

Łukasz Pawela will introduce quantum perceptron and its implementation in IBM qiskit.

Outline

1. Setting up qiskit with local simulators
2. Introduction to basic quantum operations
3. Implementing circuits in qiskit
4. Running on local simulators, remote simulators and quantum hardware
5. Result interpretation and visualization
6. Implementing a quantum perceptron model
7. *Comment on measurement error mitigation*

Recommended reading before the session:

1. Quantum logic gate – Wikipedia
2. Introduction to Qiskit

Uczenie architektur kwantowych

Dzień 1

Przypomnienie zagadnień:

- Kubity oraz stany kubitu.
- Splątanie kwantowe.
- Bramkowy model obliczeń.
- Obwody kwantowe.
- Perceptron - zasada działania.
- Perceptron - uczenie.
- Python - przypomnienie podstawowej składni.

Teoria

- Bramkowe komputery kwantowe: budowa, zasady działania.
- Ograniczenia komputerów kwantowych.
- Źródła błędów w komputerach bramkowych.
- Kwantowy model perceptronu: budowa, elementy składowe, przykładowe obwody.

Praktyka

- Uzyskanie dostępu do IBMQ.
- Podstawy biblioteki `qiskit`.
- Podstawowe operacje w portalu IBMQ, monitorowanie obliczeń.
- Symulatory architektur komputerów bramkowych, implementacja pierwszego programu.
- Symulatory lokalne vs chmurowe.
- Implementacja programu uruchamianego na komputerze bramkowym: przygotowanie stanu splątanego.

Dzień 2

Praktyka

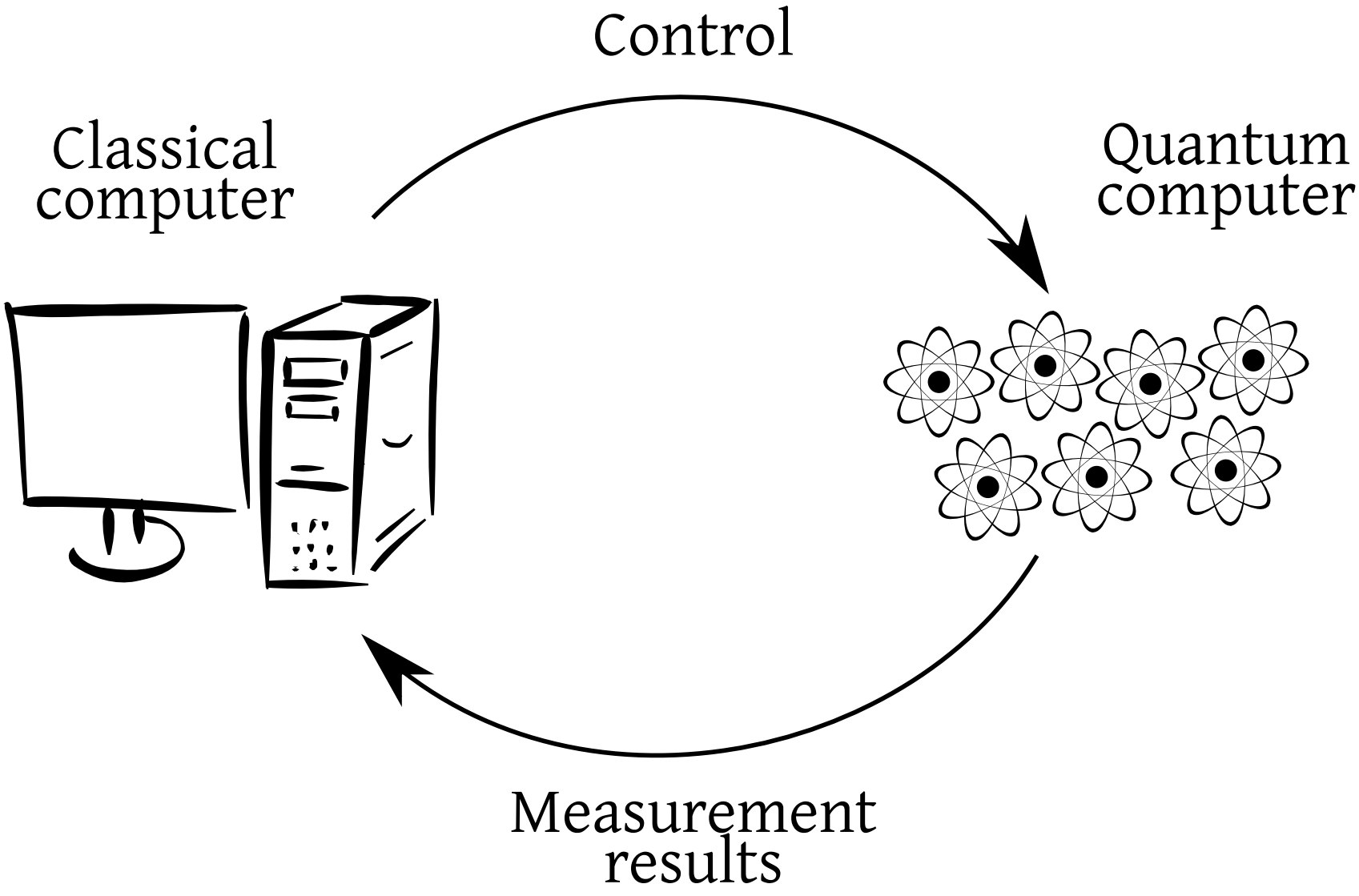
- Implementacja składowych perceptronu.
- Obserwacje działania modelu dla losowo dobranych wag.
- Implementacja algorytmu uczenia kwantowego perceptronu.
- Uczenie modelu, weryfikacja wyników.

Zagadnienia dodatkowe:

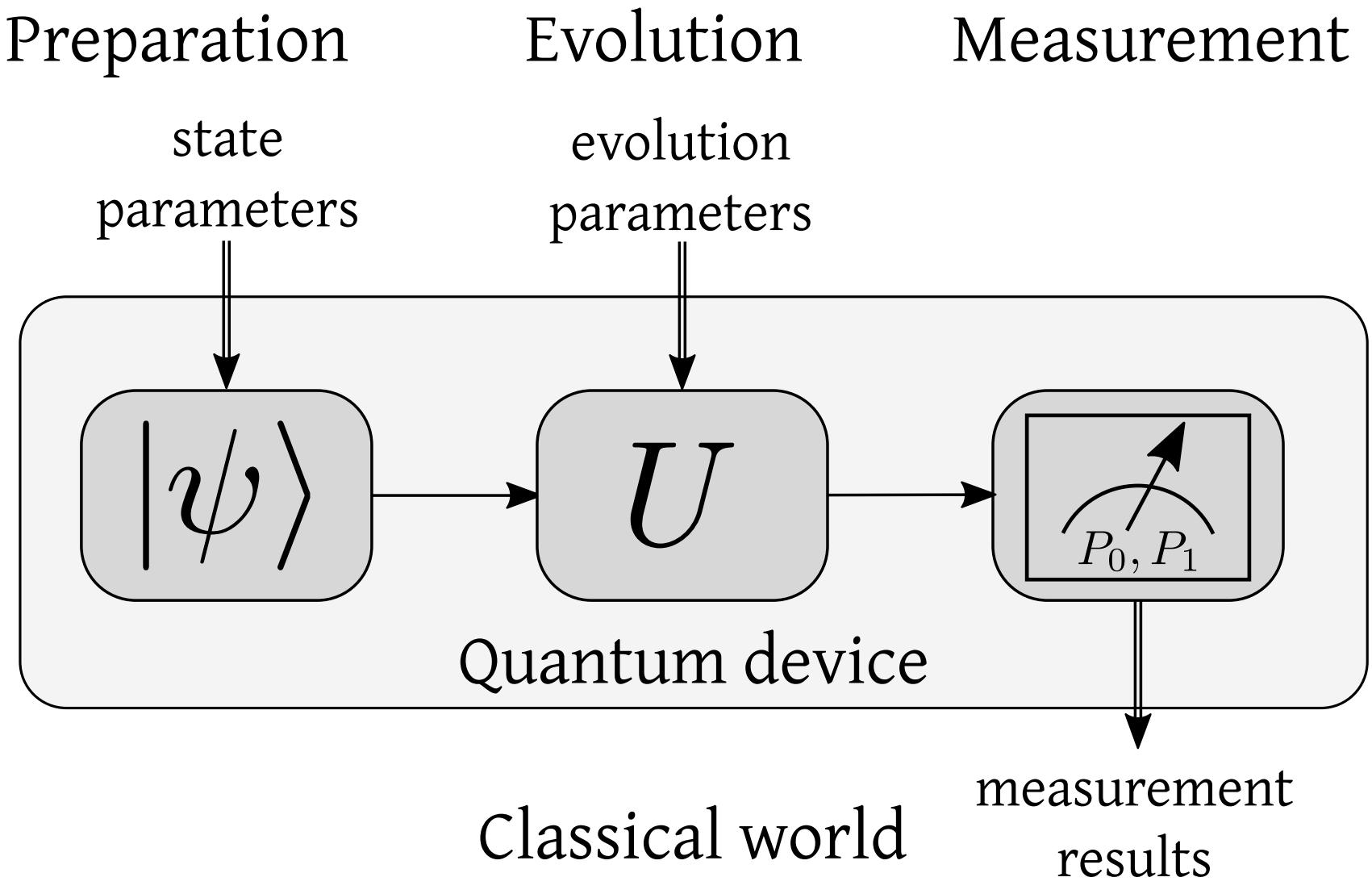
- Model bramkowy vs wyżarzanie kwantowe.
- Algortym QAOA.

Outline for section 1

Quantum computation control loop



Computation as experiment



Outline for section 2

Quantum state I

We recall the postulates of quantum mechanics, upon which the proposed method is derived.

- The **state** of quantum system is represented by a **complex unit vector** from n-dimensional Euclidean vector space $\mathbb{C}^n$.
- Let us introduce an orthonormal complete set of vectors (computational basis)

$$|0\rangle = \begin{bmatrix} 1 \\ \vdots \\ 0 \end{bmatrix}, \ldots, \; |n-1\rangle = \begin{bmatrix} 0 \\ \vdots \\ 1 \end{bmatrix}. \tag{1}$$

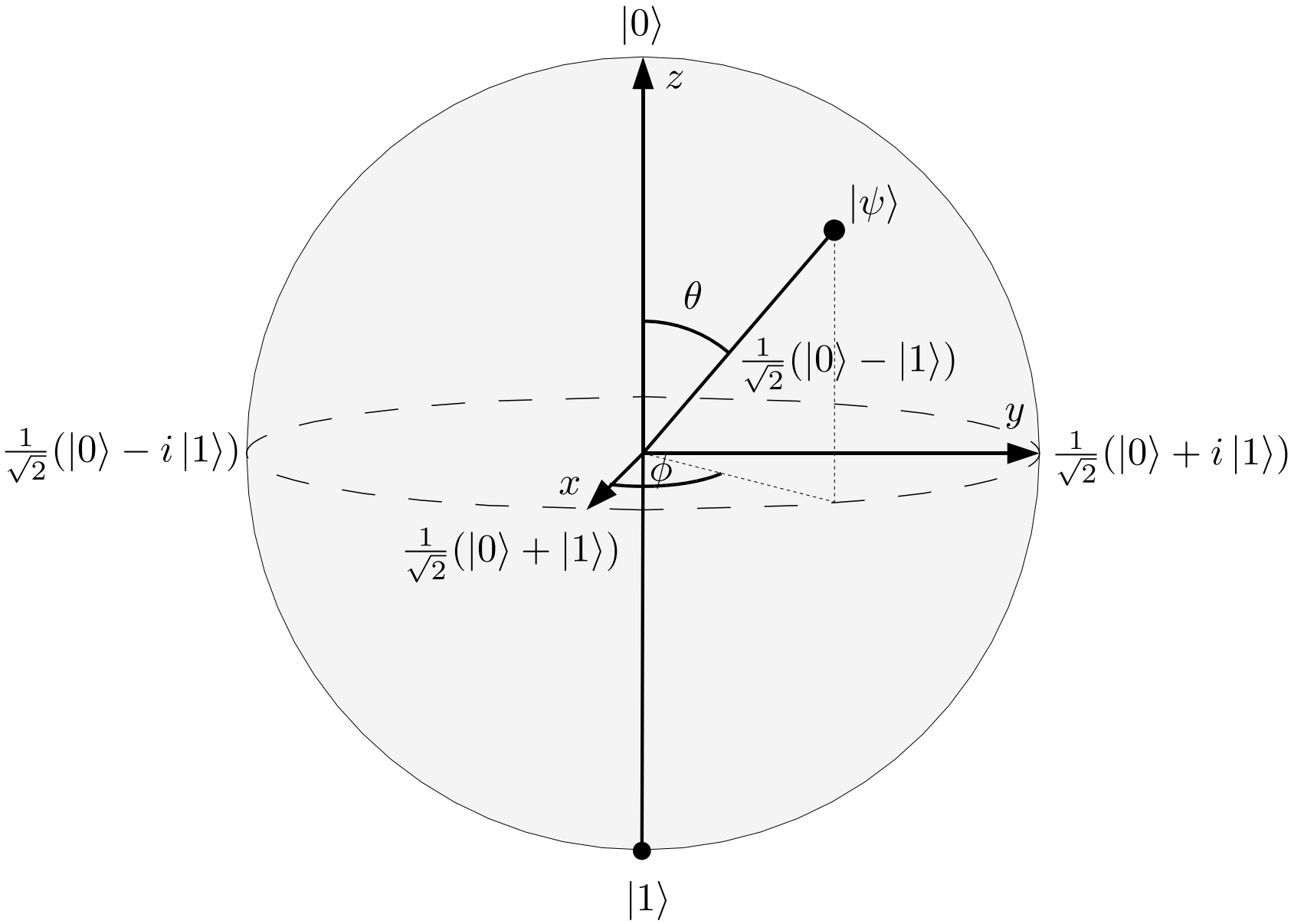
Quantum state II

- The $|x\rangle$ vector is called '**ket**' and its Hermitian conjugation $(|x\rangle)^\dagger = \langle x|$ is called '**bra**'.
- We can represent any valid state of a n-level quantum state $|\psi\rangle$ as normalized **linear combination** of the basis vectors:

$$|\psi\rangle = \alpha_1 |0\rangle + \cdots + \alpha_n |n-1\rangle , \tag{2}$$

  where $\alpha_1, \ldots, \alpha_n \in \mathbb{C}$ and $\sum_{i=1}^{n} |\alpha_i|^2 = 1$.
- We assume that two vectors $|\psi\rangle$ and $|\phi\rangle$ represent the same physical state if $|\psi\rangle = e^{i\theta} |\phi\rangle$, for $\theta \in \mathbb{R}$.

Qubit, Bloch sphere



Time evolution of a quantum system I

- The evolution of quantum systems is governed by the **Schrödinger equation**

$$\frac{\mathrm{d}\,|\psi\rangle}{\mathrm{d}t} = -\mathrm{i}H\,|\psi\rangle\,, \tag{3}$$

where H is a hermitian operator *i.e.* $H = H^\dagger$ called **Hamiltonian** of the system. Here we put Planck constant equal to one.

- Solutions of the (3) are given by

$$|\psi_{t_1}\rangle = U(t_0, t_1)\,|\psi_{t_0}\rangle\,, \tag{4}$$

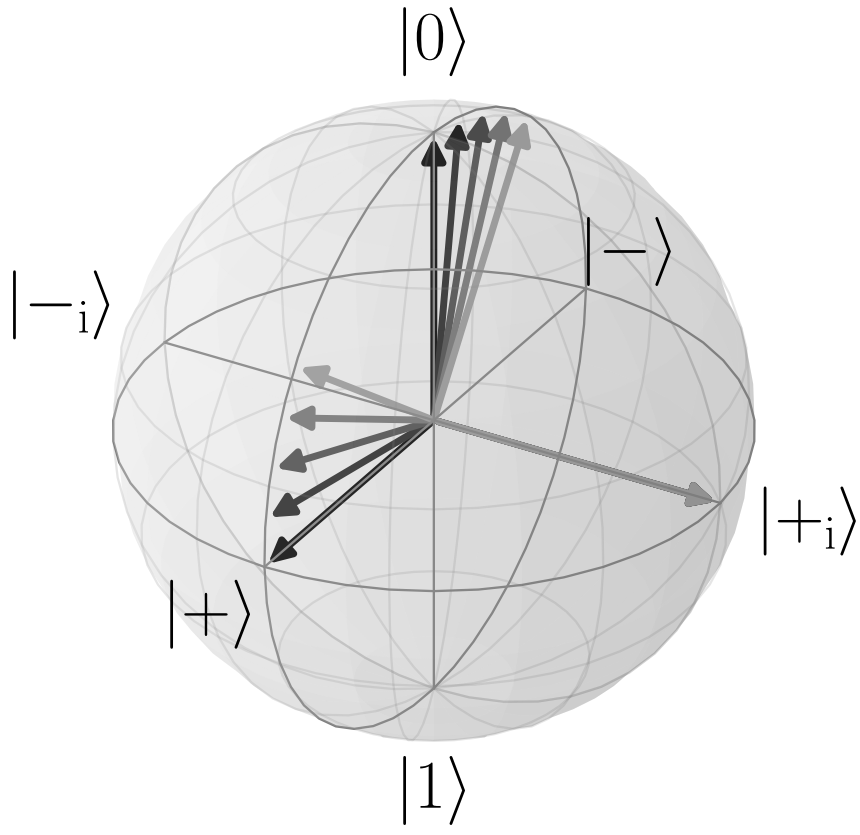
where $|\psi_{t_0}\rangle$ is the initial state of the system, $|\psi_{t_1}\rangle$ is final state of the system, and $U(t_0, t_1)$ is **unitary operator** driving the system from time t_0 to t_1.

Time evolution of a quantum system II

- An operator is unitary iff $UU^\dagger = U^\dagger U = \mathbb{1}$
- Assuming that Hamiltonian H is time is constant between time t_0 and t_1 $U(t_0, t_1)$ can be obtained from equation

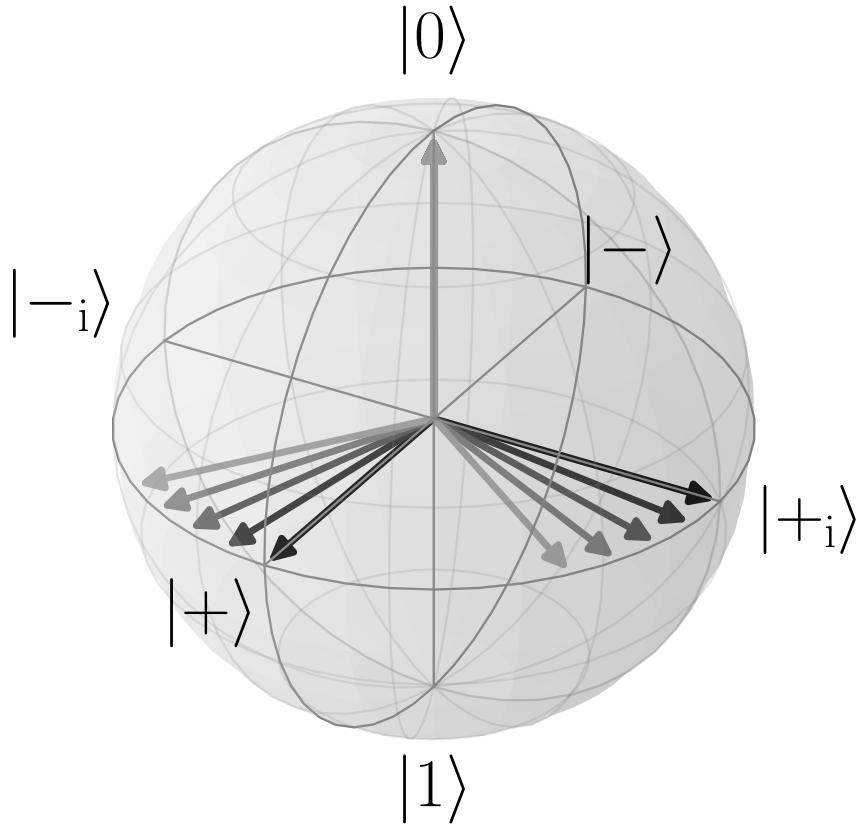
$$U(t_0, t_1) = e^{-\mathrm{i}(t_1 - t_0)H}. \tag{5}$$

Unitary gates, quantum evolution



Rotation around y-axis

Unitary gates, quantum evolution



Rotation around z-axis

Measurement

- In order to measure the state of a quantum system one has to chose a **quantum measurement** that is a function μ from finite set of measurements outcomes $A = \{a_i\}_{i=1}^n$ to set the of projection operators $P = \{P_i\}_{i=1}^n$ such that

$$\sum_{i=1}^{n} P_i = \mathbb{1} \text{ and } P_i^2 = P_i. \tag{6}$$

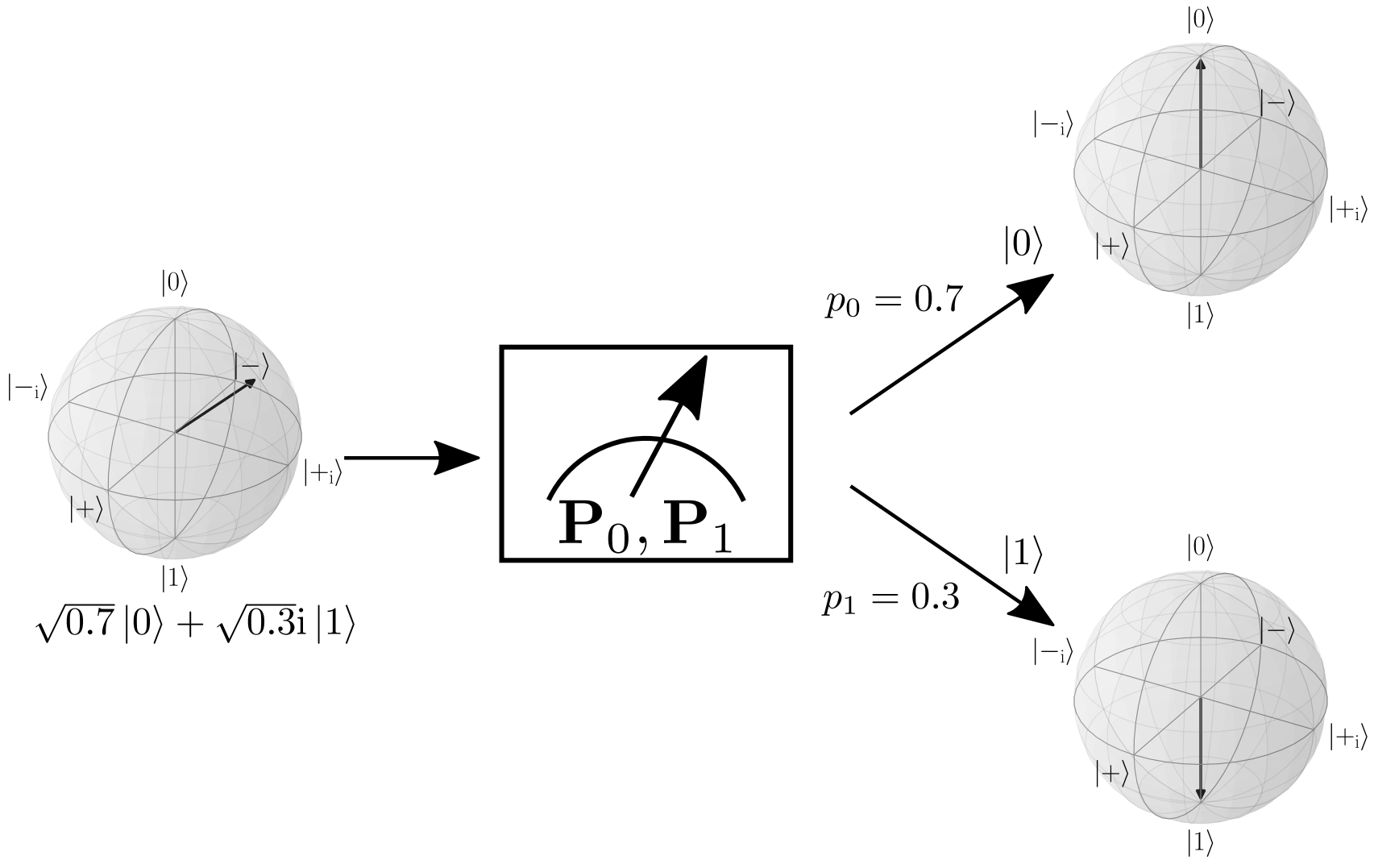
- The **probability of measuring outcome** a_i given the quantum system in in state $|\psi\rangle$ is

$$p(a_i) = \langle\psi| P_i |\psi\rangle . \tag{7}$$

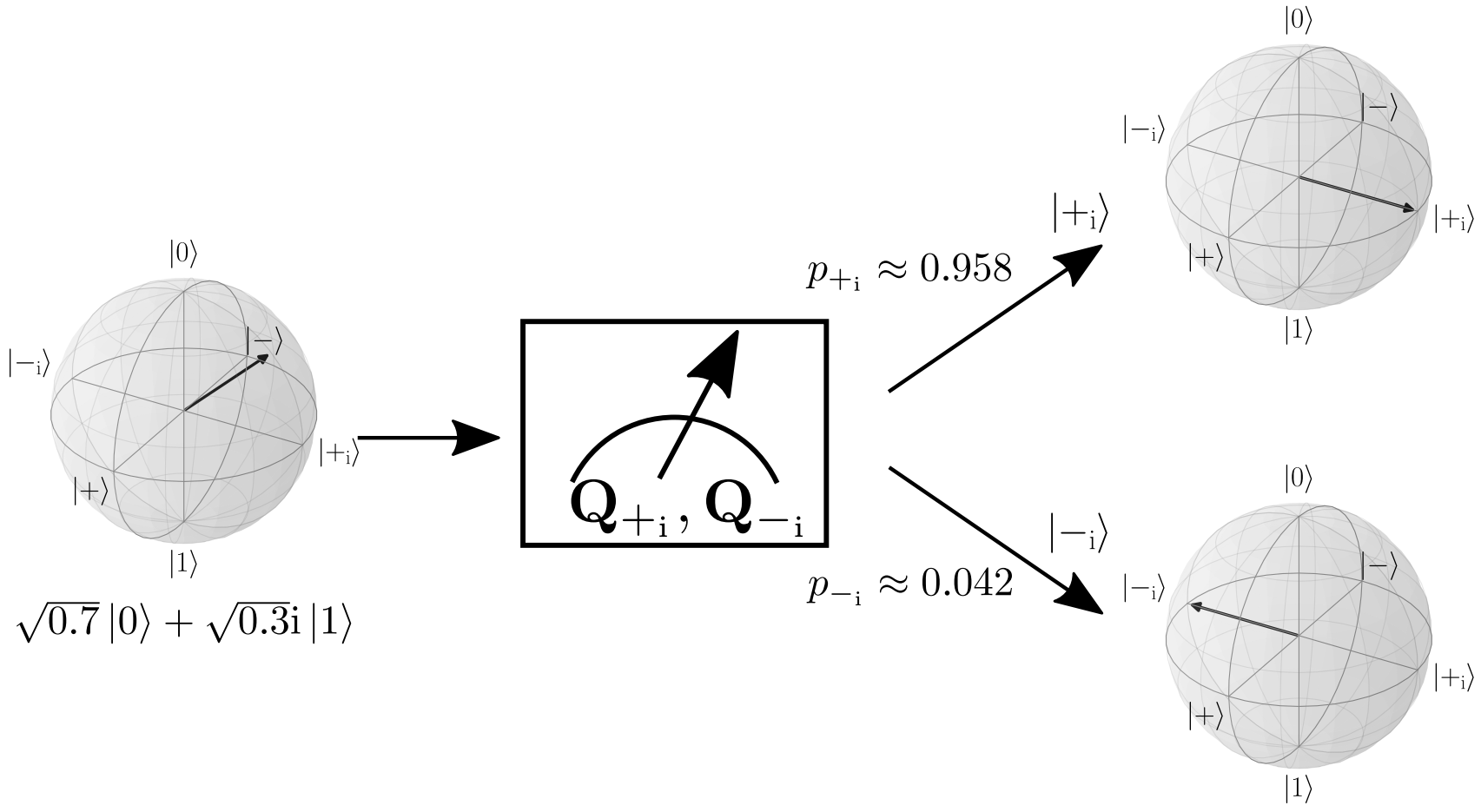
The state of the quantum system **after the measurement** outcome a_i was obtained becomes

$$|\psi\rangle_{a_i} = \frac{P_i |\psi\rangle}{\sqrt{\langle\psi| P_i |\psi\rangle}}. \tag{8}$$

Measurement



Measurement



Composition of quantum systems

The operation which allows us to join two independent quantum systems is the **tensor product**. Lets take **two qubit** states

$$|\psi\rangle = \begin{bmatrix} \psi_0 \\ \psi_1 \end{bmatrix} = \psi_0 |0\rangle + \psi_1 |1\rangle ,$$

$$|\phi\rangle = \begin{bmatrix} \phi_0 \\ \phi_1 \end{bmatrix} = \phi_0 |0\rangle + \phi_1 |1\rangle , \tag{9}$$

then we can write their joint state in $\mathbb{C}^2 \otimes \mathbb{C}^2$ as

$$|\psi\rangle \otimes |\phi\rangle = \begin{bmatrix} \psi_0\phi_0 \\ \psi_0\phi_1 \\ \psi_1\phi_0 \\ \psi_1\phi_1 \end{bmatrix} . \tag{10}$$

Entanglement

Bell state

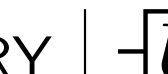
$$\left|\Phi^+\right\rangle = \frac{1}{\sqrt{2}}(|0\rangle \otimes |0\rangle + |1\rangle \otimes |1\rangle) \stackrel{?}{=}$$

$$\stackrel{?}{=} \psi_0\phi_0 \,|0\rangle \otimes |0\rangle + \psi_0\phi_1 \,|0\rangle \otimes |1\rangle + \psi_1\phi_0 \,|1\rangle \otimes |0\rangle + \psi_1\phi_1 \,|1\rangle \otimes |1\rangle$$

Contradiction

$$\psi_0\phi_0 = \psi_1\phi_1 = \frac{1}{\sqrt{2}},$$
$$\psi_0\phi_1 = \psi_1\phi_0 = 0.$$

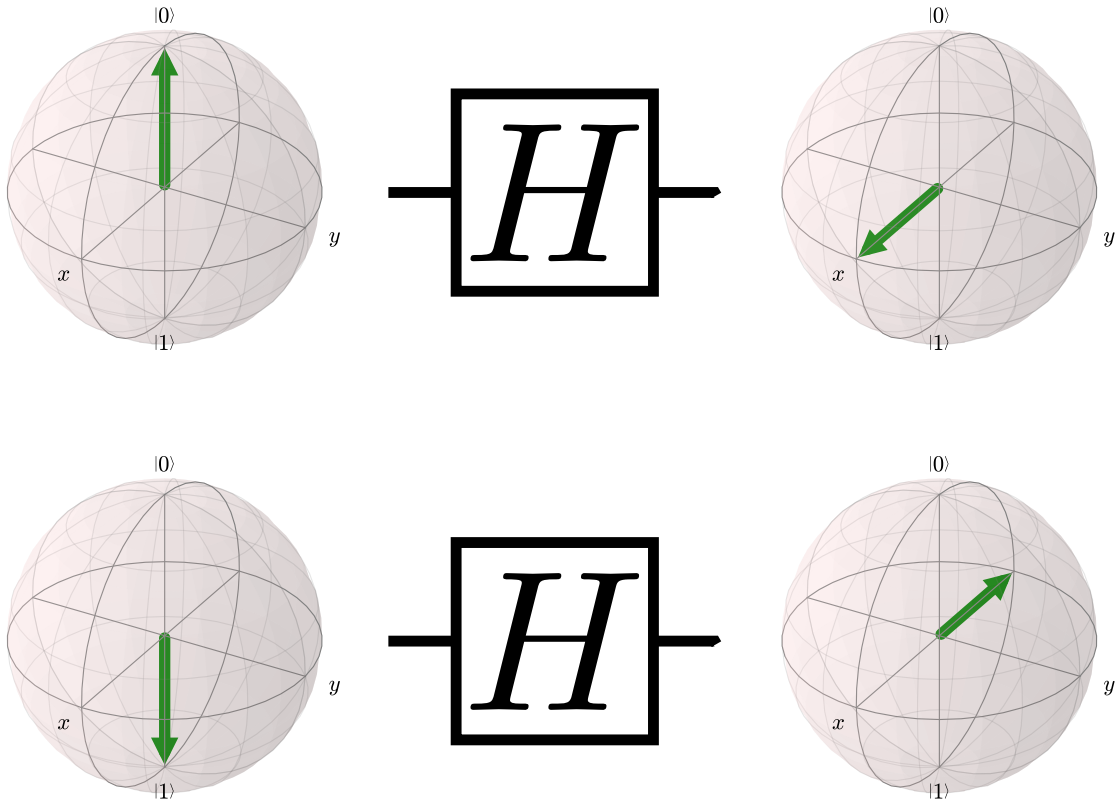
Gate model of quantum computation I

UNITARY  $U|\psi_1\rangle = |\psi_2\rangle,\ U^\dagger|\psi_2\rangle = |\psi_1\rangle$

$$U = e^{i\varphi/2}\begin{bmatrix} e^{i\psi} & 0 \\ 0 & e^{-i\psi} \end{bmatrix}\begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}\begin{bmatrix} e^{i\Delta} & 0 \\ 0 & e^{-i\Delta} \end{bmatrix}$$

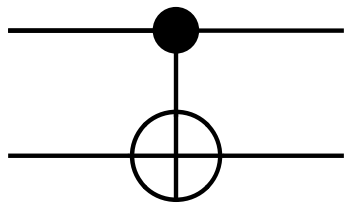
Gate model of quantum computation II

HADAMARD $\left| -\boxed{H}- \right|$ $|0\rangle \mapsto \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, $|1\rangle \mapsto \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$



Gate model of quantum computation III

CNOT



in_1	in_2	out_1	out_2
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0

Gate model of quantum computation IV

$$|0\rangle \otimes |0\rangle \rightarrow \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes |0\rangle \rightarrow \frac{1}{\sqrt{2}}(|0\rangle \otimes |0\rangle + |1\rangle \otimes |1\rangle)$$

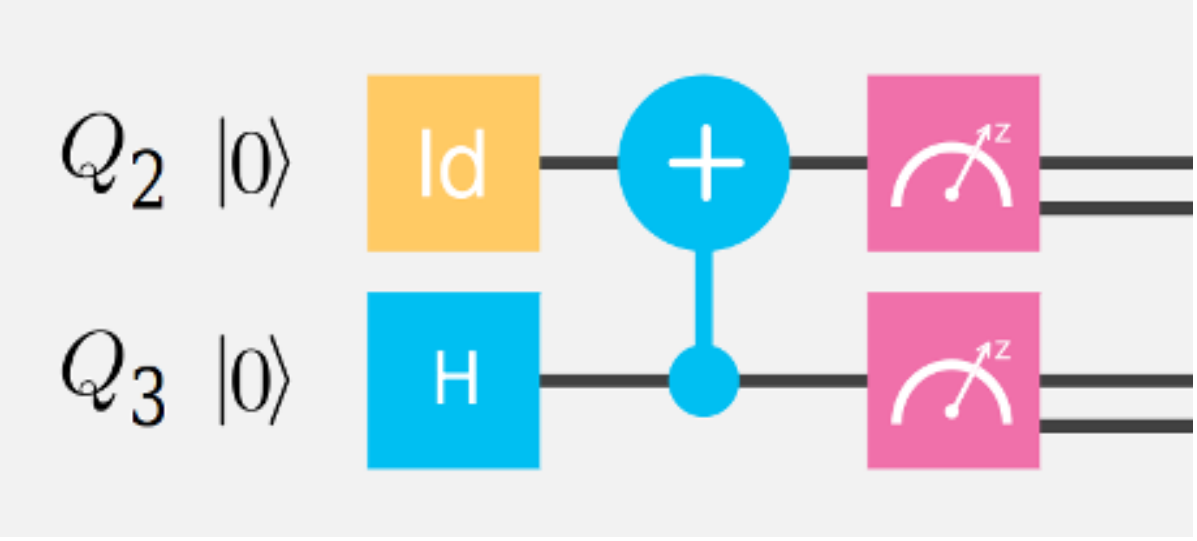


Figure: Bell state preparation circuit.

Gate model of quantum computation V

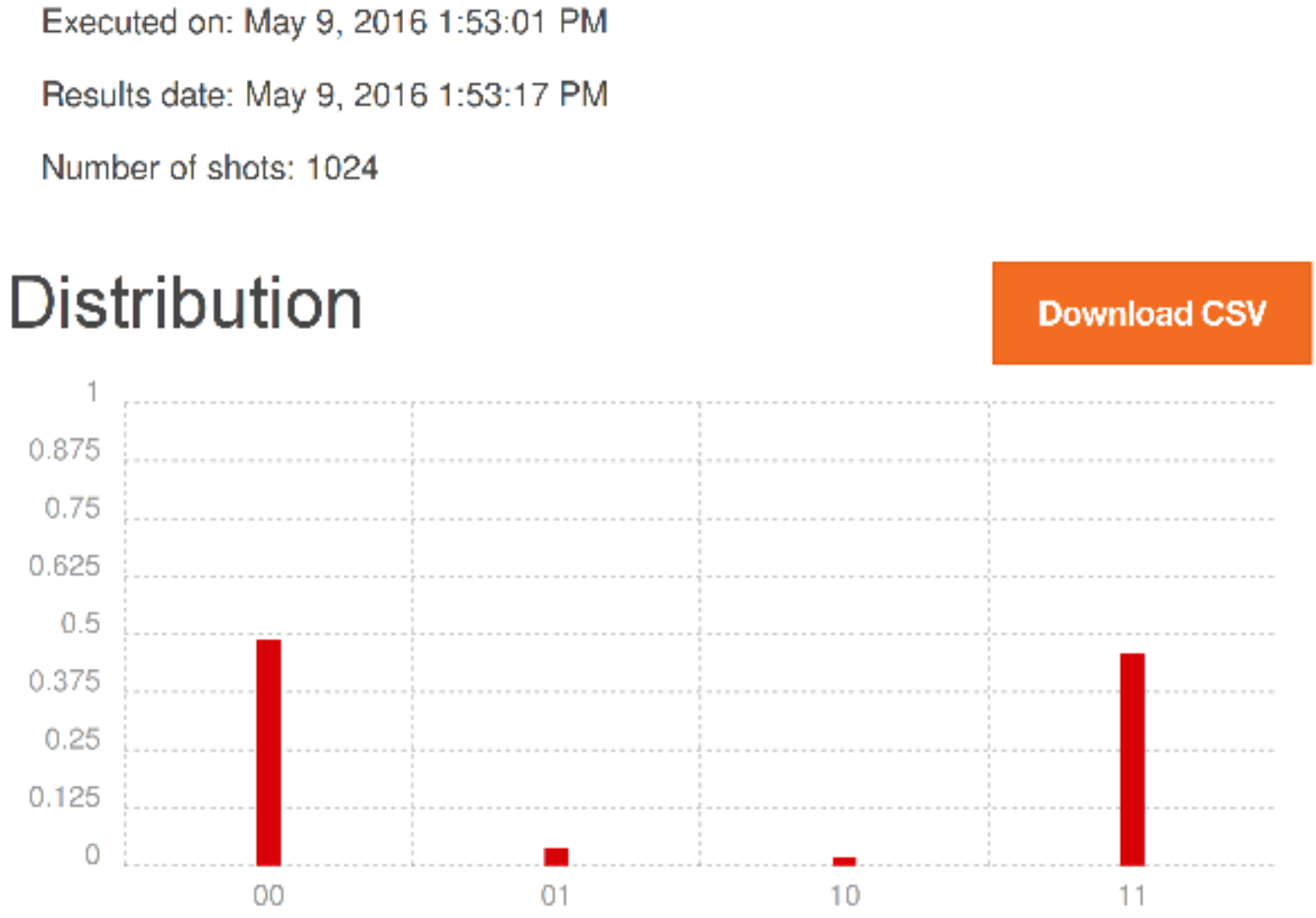
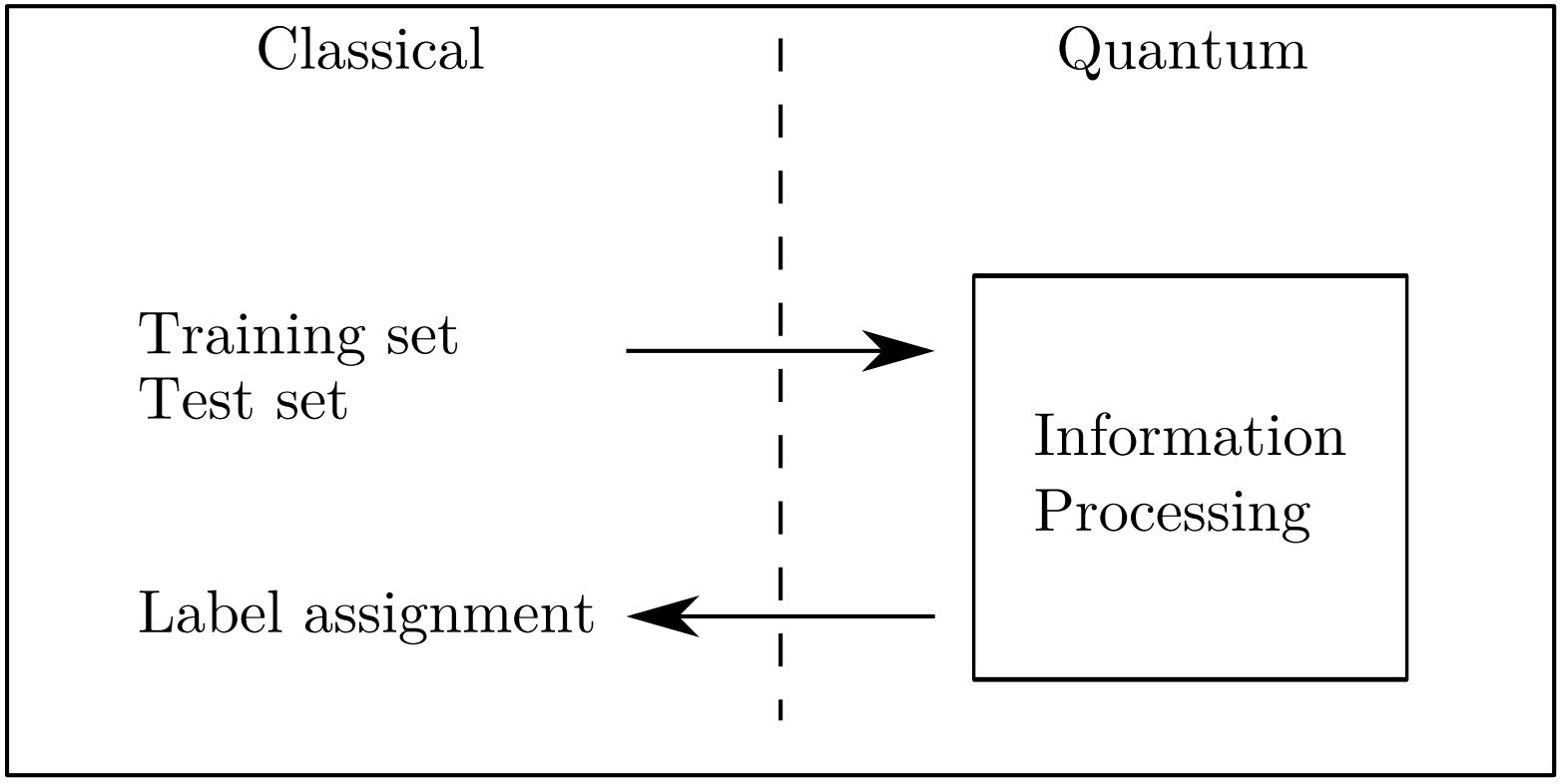


Figure: Histogram of measurements obtained from IBM 5Q machine.

Outline for section 3

Quantum classification algorithm

- Input data is classical.
- Output data is classical.
- Information processing is quantum.



Starting point

Schuld, Maria, Mark Fingerhuth, and Francesco Petruccione. "Implementing a distance-based classifier with a quantum interference circuit." EPL (Europhysics Letters) 119.6 (2017): 60002.

- A quantum algorithm that recovers classical classification algorithm.
- A set of quantum basis states is a product of features number, classes number and training set size.

00-getting-started

May 26, 2026

[1]:
```
from qiskit import QuantumCircuit, transpile
from qiskit_aer import AerSimulator
from qiskit.visualization import plot_histogram
```

Use Aer's AerSimulator

[2]:
```
simulator = AerSimulator()
```

Create a Quantum Circuit acting on the q register

[8]:
```
circuit = QuantumCircuit(2)
```

Add a H gate on qubit 0

[9]:
```
circuit.h(0)
circuit.draw()
```

[9]:
```
q_0:  H

q_1:
```

Add a CX (CNOT) gate on control qubit 0 and target qubit 1

[10]:
```
circuit.cx(0, 1)
circuit.draw()
```

[10]:
```
q_0:  H

q_1:    X
```

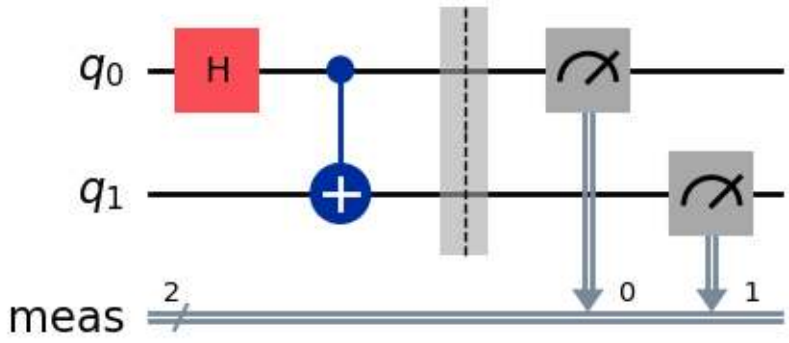
Map the quantum measurement to the classical bits

[11]:
```
circuit.measure_all()
```

[12]:
```
print(circuit)
circuit.draw(output='mpl')
```

```
   q_0:  H      M

   q_1:     X     M

meas: 2/
                     0  1
```

[12]:



Compile the circuit for the support instruction set (basis_gates) and topology (coupling_map) of the backend

[14]:
```
compiled_circuit = transpile(circuit, simulator)
print(compiled_circuit)
```

```
   q_0:  H      M

   q_1:     X     M

meas: 2/
                     0  1
```

Execute the circuit on the aer simulator

[15]:
```
job = simulator.run(compiled_circuit, shots=1000)
```

Grab results from the job

[16]:
```
result = job.result()
result
```

[16]:
```
Result(backend_name='aer_simulator', backend_version='0.17.0', qobj_id='',
job_id='426e9665-2543-445b-bf76-716759076a62', success=True,
results=[ExperimentResult(shots=1000, success=True, meas_level=2,
```

```
data=ExperimentResultData(counts={'0x0': 506, '0x3': 494}),
header=QobjExperimentHeader(creg_sizes=[['meas', 2]], global_phase=0.0,
memory_slots=2, n_qubits=2, name='circuit-164', qreg_sizes=[['q', 2]],
metadata={}), status=DONE, seed_simulator=1836436501, metadata={'time_taken':
0.014929336, 'num_bind_params': 1, 'parallel_state_update': 8, 'parallel_shots':
1, 'required_memory_mb': 0, 'input_qubit_map': [[1, 1], [0, 0]], 'method':
'stabilizer', 'device': 'CPU', 'num_qubits': 2, 'sample_measure_time':
0.000636645, 'active_input_qubits': [0, 1], 'num_clbits': 2, 'remapped_qubits':
False, 'runtime_parameter_bind': False, 'max_memory_mb': 15673, 'noise':
'ideal', 'measure_sampling': True, 'batched_shots_optimization': False,
'fusion': {'enabled': False}}, time_taken=0.014929336)],
date=2025-05-24T12:42:53.512793, status=COMPLETED, header=None,
metadata={'time_taken_parameter_binding': 1.4814e-05, 'time_taken_execute':
0.014977165, 'omp_enabled': True, 'max_gpu_memory_mb': 0, 'max_memory_mb':
15673, 'parallel_experiments': 1}, time_taken=0.016008377075195312)
```

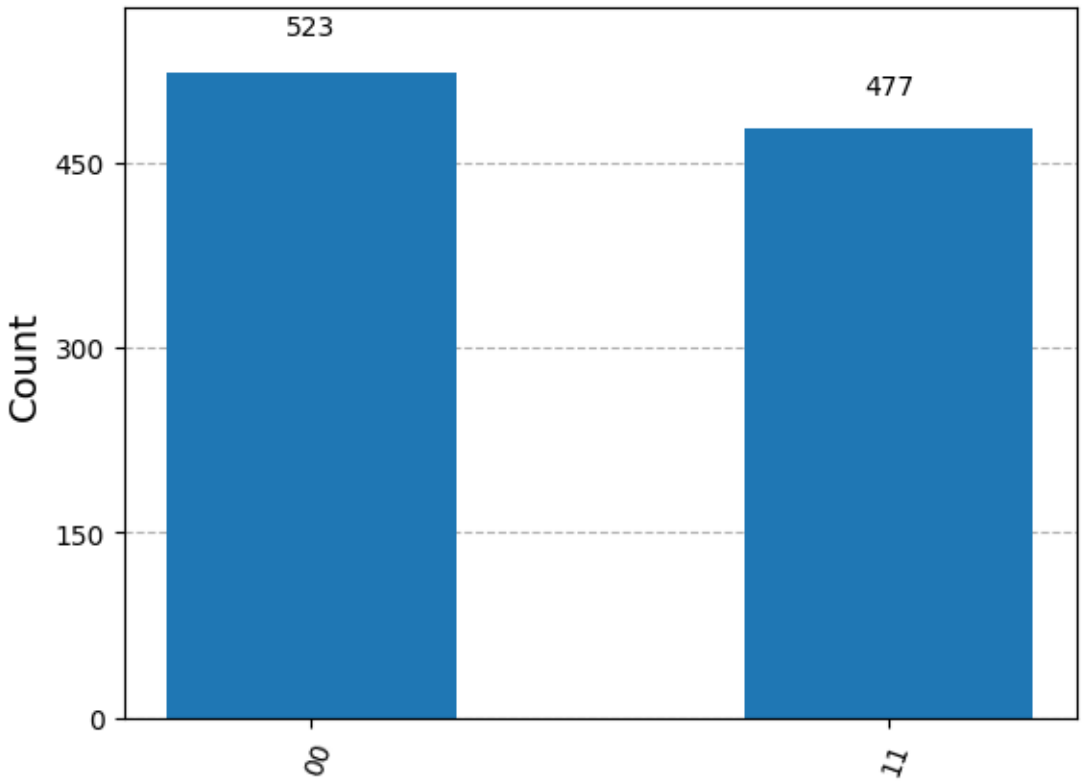
Returns counts

[17]:
```
counts = result.get_counts()
print("\nTotal count for 00 and 11 are:", counts)
```

```
Total count for 00 and 11 are: {'00': 506, '11': 494}
```

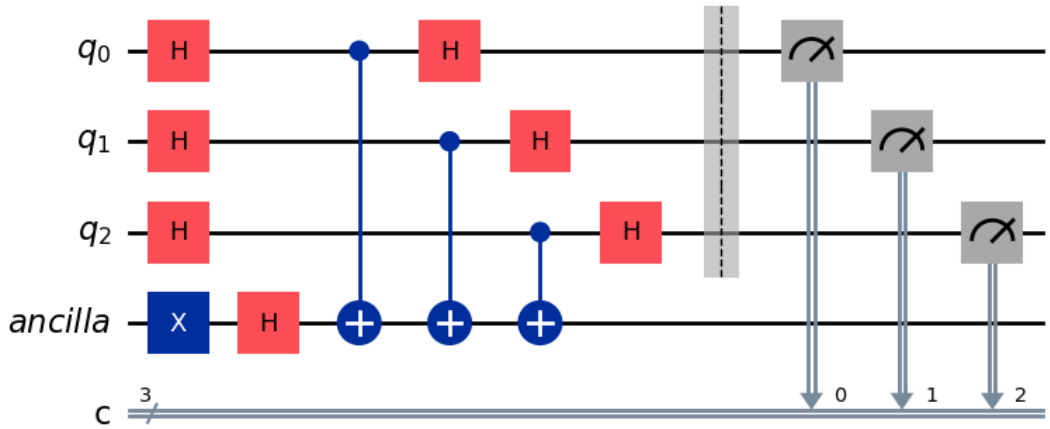
[12]:
```
plot_histogram(counts) # |00> and |11> should have equal probability
```

[12]:



[19]:
```
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit

qr = QuantumRegister(3, 'q')
anc = QuantumRegister(1, 'ancilla')
cr = ClassicalRegister(3, 'c')
qc = QuantumCircuit(qr, anc, cr)

qc.x(anc[0])
qc.h(anc[0])
qc.h(qr[0:3])
qc.cx(qr[0:3], anc[0])
# qc.barrier(qr)
qc.h(qr[0:3])
qc.barrier(qr)
qc.measure(qr, cr)# |1> -> |0> - |1>

qc.draw('mpl')
```

[19]:



[20]:
```
transpiled_qc = transpile(qc, simulator)
```

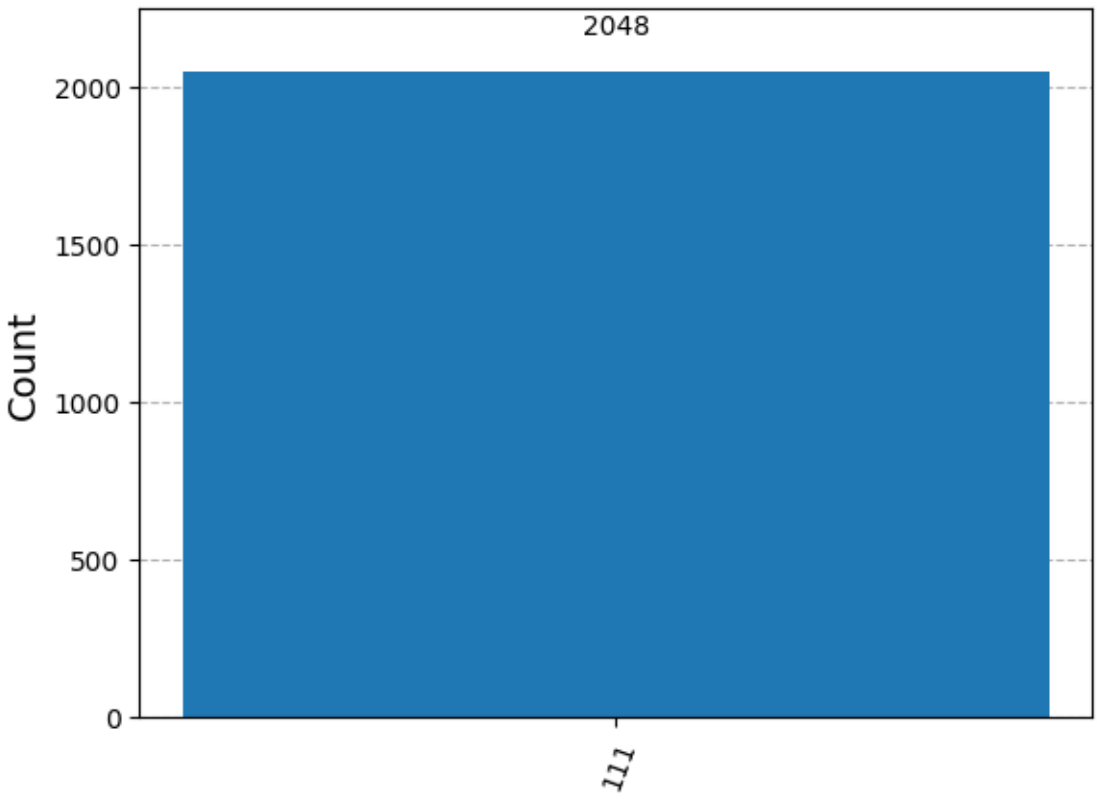
[21]:
```
job = simulator.run(transpiled_qc, shots=2048)
```

[22]:
```
result = job.result()
```

[23]:
```
counts = result.get_counts()
```

[24]:
```
plot_histogram(counts)
```

[24]:



[ ]:

Building Noise Models

Introduction

This notebook introduces how to use the Qiskit Aer `noise` module to build custom noise models for noisy simulations.

In [1]:
```
import numpy as np
from qiskit import QuantumCircuit, transpile
from qiskit.quantum_info import Kraus, SuperOp
from qiskit_aer import AerSimulator
from qiskit.visualization import plot_histogram
# Import from Qiskit Aer noise module
from qiskit_aer.noise import (NoiseModel, QuantumError, ReadoutError,
    pauli_error, depolarizing_error, thermal_relaxation_error)
```

Qiskit Aer Noise Module

The Qiskit Aer `noise` module contains Python classes to build customized noise models for simulation. There are three key classes:

1. The `NoiseModel` class which stores a noise model used for noisy simulation.
2. The `QuantumError` class which describes CPTP gate errors. These can be applied:
   - After *gate* or *reset* instructions
   - Before *measure* instructions.
3. The `ReadoutError` class which describes classical readout errors.

Quantum Errors

Rather than deal with the `QuantumError` object directly, many helper functions exist to automatically generate a specific type of parameterized quantum error. These are contained in the `noise` module and include functions for many common errors types used in quantum computing research. The function names and the type of error they return are:

Standard error function	Details
`kraus_error`	a general n-qubit CPTP error channel given as a list of Kraus matrices $[K_0, ...]$.
`mixed_unitary_error`	an n-qubit mixed unitary error given as a list of unitary matrices and probabilities $[(U_0, p_0),...]$.
`coherent_unitary_error`	an n-

qubit coherent unitary error given as a single unitary matrix U. | | `pauli_error` | an n-qubit Pauli error channel (mixed unitary) given as a list of Pauli's and probabilities $[(P_0, p_0),...]$ | | `depolarizing_error` | an n-qubit depolarizing error channel parameterized by a depolarization probability p. | | `reset_error` | a single-qubit reset error parameterized by a probabilities p_0, p_1 of resetting to the $|0\rangle$, $|1\rangle$ state.| | `thermal_relaxation_error` | a single qubit thermal relaxation channel parameterized by relaxation time constants T_1, T_2, gate time t, and excited state thermal population p_1. | | `phase_amplitude_damping_error` | A single-qubit generalized combined phase and amplitude damping error channel given by an amplitude damping parameter λ, a phase damping parameter γ, and an excited state thermal population p_1. | | `amplitude_damping_error` | A single-qubit generalized amplitude damping error channel given by an amplitude damping parameter λ, and an excited state thermal population p_1. | | `phase_damping_error` | A single-qubit phase damping error channel given by a phase damping parameter γ |

Combining quantum errors

`QuantumError` instances can be combined by using composition, tensor product, and tensor expansion (reversed order tensor product) to produce new `QuantumErrors` as:

- Composition: $\cal{E}(\rho)=\cal{E_2}(\cal{E_1}(\rho))$ as `error = error1.compose(error2)`
- Tensor product: $\cal{E}(\rho) =(\cal{E_1}\otimes\cal{E_2})(\rho)$ as `error error1.tensor(error2)`
- Expand product: $\cal{E}(\rho) =(\cal{E_2}\otimes\cal{E_1})(\rho)$ as `error error1.expand(error2)`

Example

For example to construct a 5% single-qubit Bit-flip error:

In [2]:
```
# Construct a 1-qubit bit-flip and phase-flip errors
p_error = 0.05
bit_flip = pauli_error([('X', p_error), ('I', 1 - p_error)])
phase_flip = pauli_error([('Z', p_error), ('I', 1 - p_error)])
print(bit_flip)
print(phase_flip)
```

```
QuantumError on 1 qubits. Noise circuits:
  P(0) = 0.05, Circuit =
     ┌───┐
q: ┤ X ├
     └───┘
  P(1) = 0.95, Circuit =
     ┌───┐
q: ┤ I ├
     └───┘
QuantumError on 1 qubits. Noise circuits:
  P(0) = 0.05, Circuit =
     ┌───┐
q: ┤ Z ├
     └───┘
  P(1) = 0.95, Circuit =
     ┌───┐
q: ┤ I ├
     └───┘
```

In [3]:
```
# Compose two bit-flip and phase-flip errors
bitphase_flip = bit_flip.compose(phase_flip)
print(bitphase_flip)
```

```
QuantumError on 1 qubits. Noise circuits:
  P(0) = 0.0025000000000000005, Circuit =
     ┌───┐┌───┐
q: ┤ X ├┤ Z ├
     └───┘└───┘
  P(1) = 0.0475, Circuit =
     ┌───┐┌───┐
q: ┤ X ├┤ I ├
     └───┘└───┘
  P(2) = 0.0475, Circuit =
     ┌───┐┌───┐
q: ┤ I ├┤ Z ├
     └───┘└───┘
  P(3) = 0.9025, Circuit =
     ┌───┐┌───┐
q: ┤ I ├┤ I ├
     └───┘└───┘
```

In [4]:
```
# Tensor product two bit-flip and phase-flip errors with
# bit-flip on qubit-0, phase-flip on qubit-1
error2 = phase_flip.tensor(bit_flip)
print(error2)
```

```
QuantumError on 2 qubits. Noise circuits:
  P(0) = 0.0025000000000000005, Circuit =
     ┌───┐
q_0: ┤ X ├
     ├───┤
q_1: ┤ Z ├
     └───┘
  P(1) = 0.0475, Circuit =
     ┌───┐
q_0: ┤ I ├
     ├───┤
q_1: ┤ Z ├
     └───┘
  P(2) = 0.0475, Circuit =
     ┌───┐
q_0: ┤ X ├
     ├───┤
q_1: ┤ I ├
     └───┘
  P(3) = 0.9025, Circuit =
     ┌───┐
q_0: ┤ I ├
     ├───┤
q_1: ┤ I ├
     └───┘
```

Converting to and from QuantumChannel operators

We can also convert back and forth between `QuantumError` objects in Qiskit Aer and `QuantumChannel` objects in Qiskit Terra.

In [5]:
```
# Convert to Kraus operator
bit_flip_kraus = Kraus(bit_flip)
print(bit_flip_kraus)
```

```
Kraus([[[-9.74679434e-01+0.j,  0.00000000e+00+0.j],
        [ 0.00000000e+00+0.j, -9.74679434e-01+0.j]],

       [[ 0.00000000e+00+0.j,  2.23606798e-01+0.j],
        [ 2.23606798e-01+0.j, -4.96506831e-17+0.j]]],
      input_dims=(2,), output_dims=(2,))
```

In [6]:
```
# Convert to Superoperator
phase_flip_sop = SuperOp(phase_flip)
print(phase_flip_sop)
```

```
SuperOp([[1. +0.j, 0. +0.j, 0. +0.j, 0. +0.j],
         [0. +0.j, 0.9+0.j, 0. +0.j, 0. +0.j],
         [0. +0.j, 0. +0.j, 0.9+0.j, 0. +0.j],
         [0. +0.j, 0. +0.j, 0. +0.j, 1. +0.j]],
        input_dims=(2,), output_dims=(2,))
```

In [7]:
```
# Convert back to a quantum error
print(QuantumError(bit_flip_kraus))
```

```
# Check conversion is equivalent to original error
QuantumError(bit_flip_kraus) == bit_flip
```

```
QuantumError on 1 qubits. Noise circuits:
  P(0) = 1.0, Circuit =
     ┌─────────┐
q: ┤ kraus ├
     └─────────┘
```

Out[7]: True

Readout Error

Classical readout errors are specified by a list of assignment probabilities vectors $P(A|B)$:

- A is the *recorded* classical bit value
- B is the *true* bit value returned from the measurement

E.g. for 1 qubits: $ P(A|B) = [P(A|0), P(A|1)]$.

In [8]:
```
# Measurement miss-assignement probabilities
p0given1 = 0.1
p1given0 = 0.05

ReadoutError([[1 - p1given0, p1given0], [p0given1, 1 - p0given1]])
```

Out[8]:
```
ReadoutError([[0.95 0.05]
 [0.1  0.9 ]])
```

Readout errors may also be combined using `compose`, `tensor` and `expand` like with quantum errors.

Adding errors to a Noise Model

When adding a quantum error to a noise model we must specify the type of *instruction* that it acts on, and what qubits to apply it to. There are two cases for Quantum Errors:

1. All-qubit quantum error
2. Specific qubit quantum error

1. All-qubit quantum error

This applies the same error to any occurrence of an instruction, regardless of which qubits it acts on.

It is added as `noise_model.add_all_qubit_quantum_error(error, instructions)`:

In [9]:
```
# Create an empty noise model
noise_model = NoiseModel()

# Add depolarizing error to all single qubit u1, u2, u3 gates
error = depolarizing_error(0.05, 1)
noise_model.add_all_qubit_quantum_error(error, ['u1', 'u2', 'u3'])

# Print noise model info
print(noise_model)
```

```
NoiseModel:
  Basis gates: ['cx', 'id', 'rz', 'sx', 'u1', 'u2', 'u3']
  Instructions with noise: ['u1', 'u3', 'u2']
  All-qubits errors: ['u1', 'u2', 'u3']
```

2. Specific qubit quantum error

This applies the error to any occurrence of an instruction acting on a specified list of qubits. Note that the order of the qubit matters: For a 2-qubit gate an error applied to qubits [0, 1] is different to one applied to qubits [1, 0] for example.

It is added as `noise_model.add_quantum_error(error, instructions, qubits)`:

In [10]:
```
# Create an empty noise model
noise_model = NoiseModel()

# Add depolarizing error to all single qubit u1, u2, u3 gates on qubit 0 onl
error = depolarizing_error(0.05, 1)
noise_model.add_quantum_error(error, ['u1', 'u2', 'u3'], [0])

# Print noise model info
print(noise_model)
```

```
NoiseModel:
  Basis gates: ['cx', 'id', 'rz', 'sx', 'u1', 'u2', 'u3']
  Instructions with noise: ['u1', 'u3', 'u2']
  Qubits with noise: [0]
  Specific qubit errors: [('u1', (0,)), ('u2', (0,)), ('u3', (0,))]
```

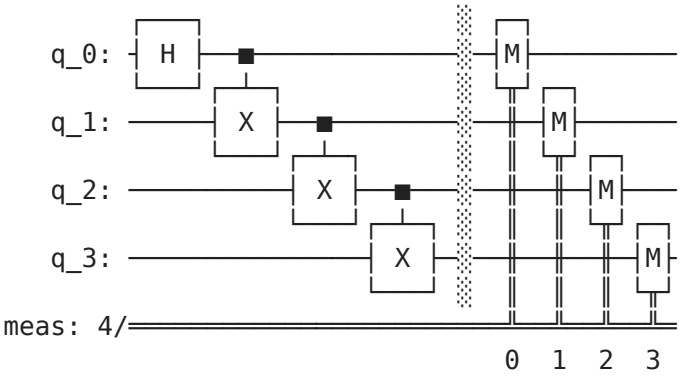
Executing a noisy simulation with a noise model

The command `AerSimulator(noise_model=noise_model)` returns a simulator configured to the given noise model. In addition to setting the simulator's noise model, it also overrides the simulator's basis gates, according to the gates of the noise model.

Noise Model Examples

We will now give some examples of noise models. For our demonstrations we will use a simple test circuit generating a n-qubit GHZ state:

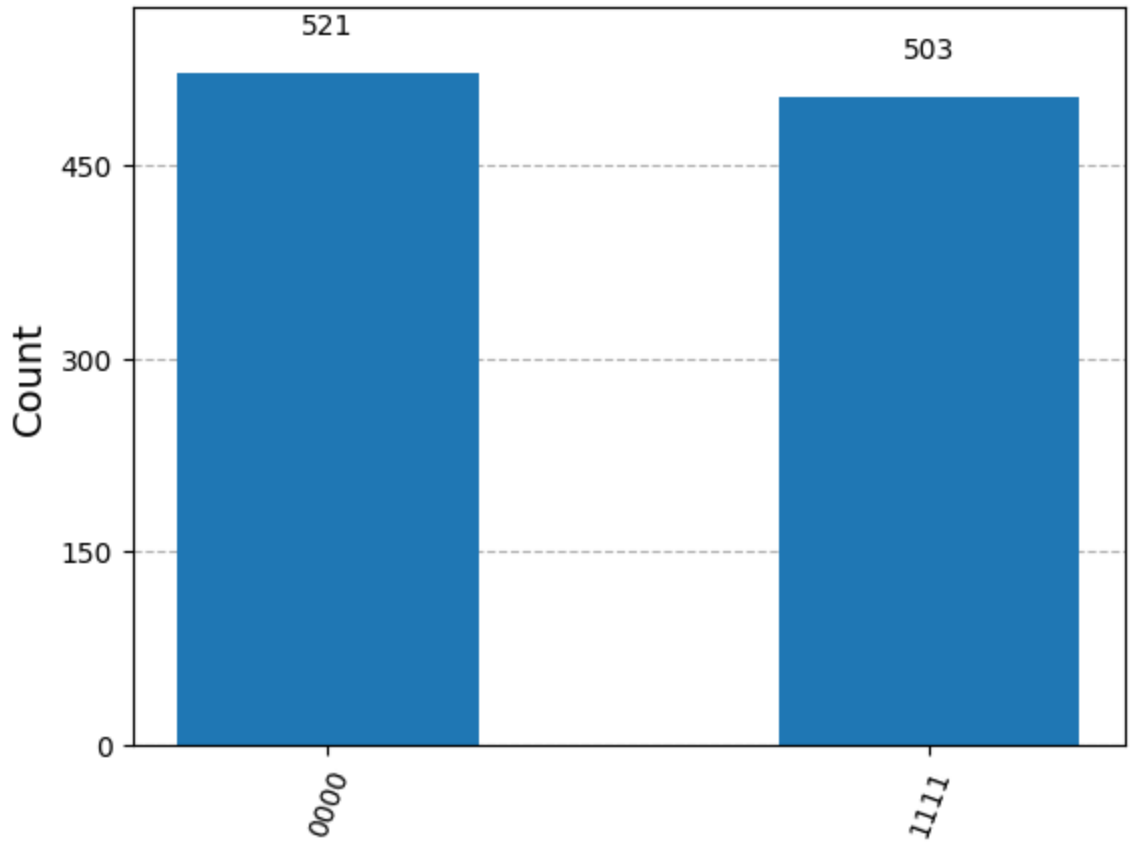
In [11]:
```
# System Specification
n_qubits = 4
circ = QuantumCircuit(n_qubits)

# Test Circuit
circ.h(0)
for qubit in range(n_qubits - 1):
    circ.cx(qubit, qubit + 1)
circ.measure_all()
print(circ)
```



Ideal Simulation

In [12]:
```
# Ideal simulator and execution
sim_ideal = AerSimulator()
result_ideal = sim_ideal.run(circ).result()
plot_histogram(result_ideal.get_counts(0))
```

Out[12]:



Noise Example 1: Basic bit-flip error noise model

Lets consider a simple toy noise model example common in quantum information theory research:

- When applying a single qubit gate, flip the state of the qubit with probability `p_gate1`.
- When applying a 2-qubit gate apply single-qubit errors to each qubit.
- When resetting a qubit reset to 1 instead of 0 with probability `p_reset`.
- When measuring a qubit, flip the state of the qubit with probability `p_meas`.

In [13]:

```
# Example error probabilities
p_reset = 0.03
p_meas = 0.1
p_gate1 = 0.05

# QuantumError objects
error_reset = pauli_error([('X', p_reset), ('I', 1 - p_reset)])
error_meas = pauli_error([('X',p_meas), ('I', 1 - p_meas)])
error_gate1 = pauli_error([('X',p_gate1), ('I', 1 - p_gate1)])
error_gate2 = error_gate1.tensor(error_gate1)

# Add errors to noise model
noise_bit_flip = NoiseModel()
noise_bit_flip.add_all_qubit_quantum_error(error_reset, "reset")
noise_bit_flip.add_all_qubit_quantum_error(error_meas, "measure")
```

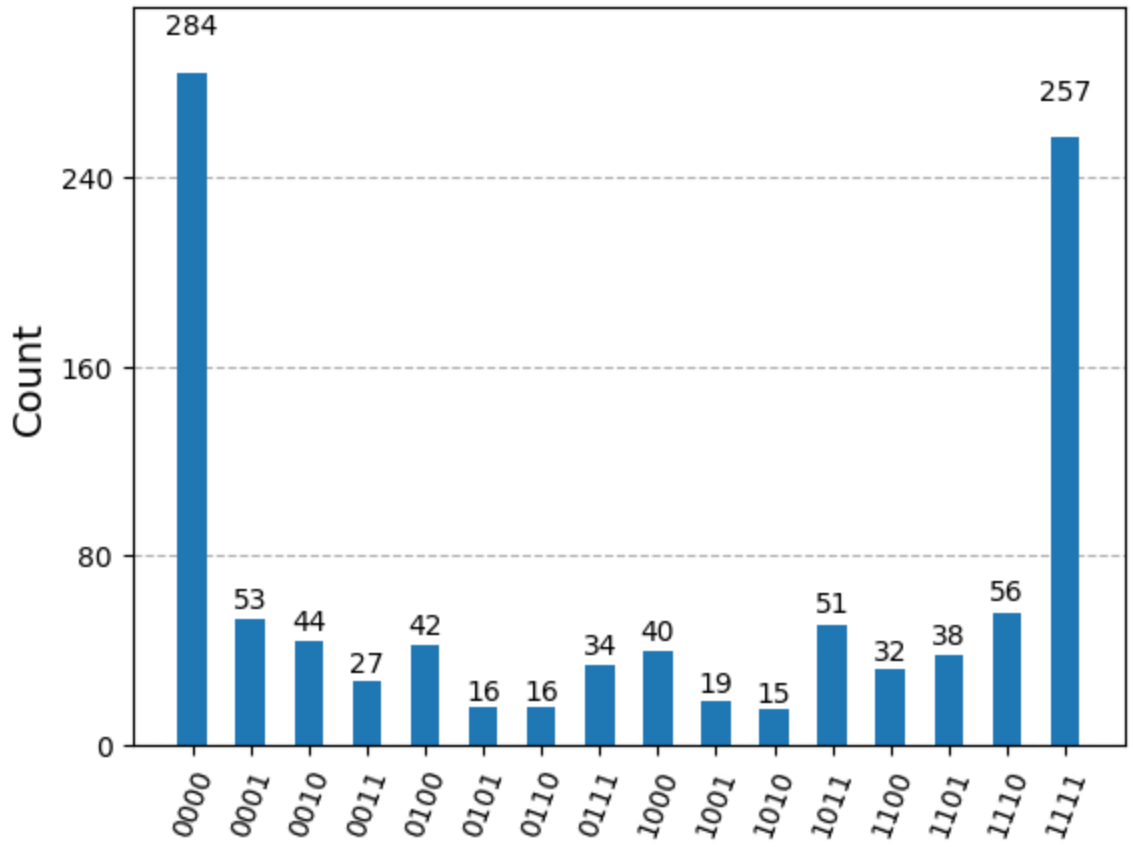
```
noise_bit_flip.add_all_qubit_quantum_error(error_gate1, ["u1", "u2", "u3"])
noise_bit_flip.add_all_qubit_quantum_error(error_gate2, ["cx"])

print(noise_bit_flip)
```

```
NoiseModel:
  Basis gates: ['cx', 'id', 'rz', 'sx', 'u1', 'u2', 'u3']
  Instructions with noise: ['u3', 'measure', 'u1', 'cx', 'reset', 'u2']
  All-qubits errors: ['reset', 'measure', 'u1', 'u2', 'u3', 'cx']
```

Executing the noisy simulation

In [14]:
```
# Create noisy simulator backend
sim_noise = AerSimulator(noise_model=noise_bit_flip)

# Transpile circuit for noisy basis gates
circ_tnoise = transpile(circ, sim_noise)

# Run and get counts
result_bit_flip = sim_noise.run(circ_tnoise).result()
counts_bit_flip = result_bit_flip.get_counts(0)

# Plot noisy output
plot_histogram(counts_bit_flip)
```

Out[14]:



In [15]:
```
print(noise_bit_flip)
```

```
NoiseModel:
  Basis gates: ['cx', 'id', 'rz', 'sx', 'u1', 'u2', 'u3']
  Instructions with noise: ['u3', 'measure', 'u1', 'cx', 'reset', 'u2']
  All-qubits errors: ['reset', 'measure', 'u1', 'u2', 'u3', 'cx']
```

In [ ]:

20-accessing-ibmq

May 26, 2026

0.1 Safe token management

Use local `.env` file for safe token storage

[1]:
```
!pip install python-dotenv
```

```
Requirement already satisfied: python-dotenv in
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (1.1.0)
```

[2]:
```
%load_ext dotenv
%dotenv
```

0.2 Simple access example

[3]:
```
import os
from qiskit import transpile
from qiskit.circuit.random import random_circuit
from qiskit_ibm_runtime import QiskitRuntimeService, SamplerV2
from qiskit.visualization import plot_histogram
```

[4]:
```
token = os.environ["IBMQ_TOKEN"]
```

[5]:
```
service = QiskitRuntimeService(
                token=token,
                channel="ibm_quantum",
            )
backend = service.least_busy(operational=True, simulator=False)
```

```
/tmp/ipykernel_94197/1493915476.py:1: DeprecationWarning: The "ibm_quantum"
channel option is deprecated and will be sunset on 1 July. After this date,
ibm_cloud will be the only valid channel. For information on migrating to the
new IBM Quantum Platform on the "ibm_cloud" channel, review the migration guide
https://quantum.cloud.ibm.com/docs/migration-guides/classic-iqp-to-cloud-iqp .
  service = QiskitRuntimeService(
```

[7]:
```
backend
```

[7]:
```
<IBMBackend('ibm_sherbrooke')>
```

[8]:
```
qx = random_circuit(5, depth=2)
qx.measure_all()
```

[9]:
```
qx.draw()
```

[9]:
```
   q_0:  X          Sx    M

   q_1:     P(0.68793)        M

   q_2:     Rz(1.5822)          M

   q_3:     P(3.1285)             M

   q_4:                       M

meas: 5/
                                    0  1  2  3  4
```

[10]:
```
transpiled = transpile(qx, backend=backend)
```

[11]:
```
transpiled.draw(output="mpl")
```

[11]:

```
ExecutionSpans([DoubleSliceSpan(<start='2025-05-24 12:33:28', stop='2025-05-24
12:33:30', size=4096>)])}, 'version': 2})
```

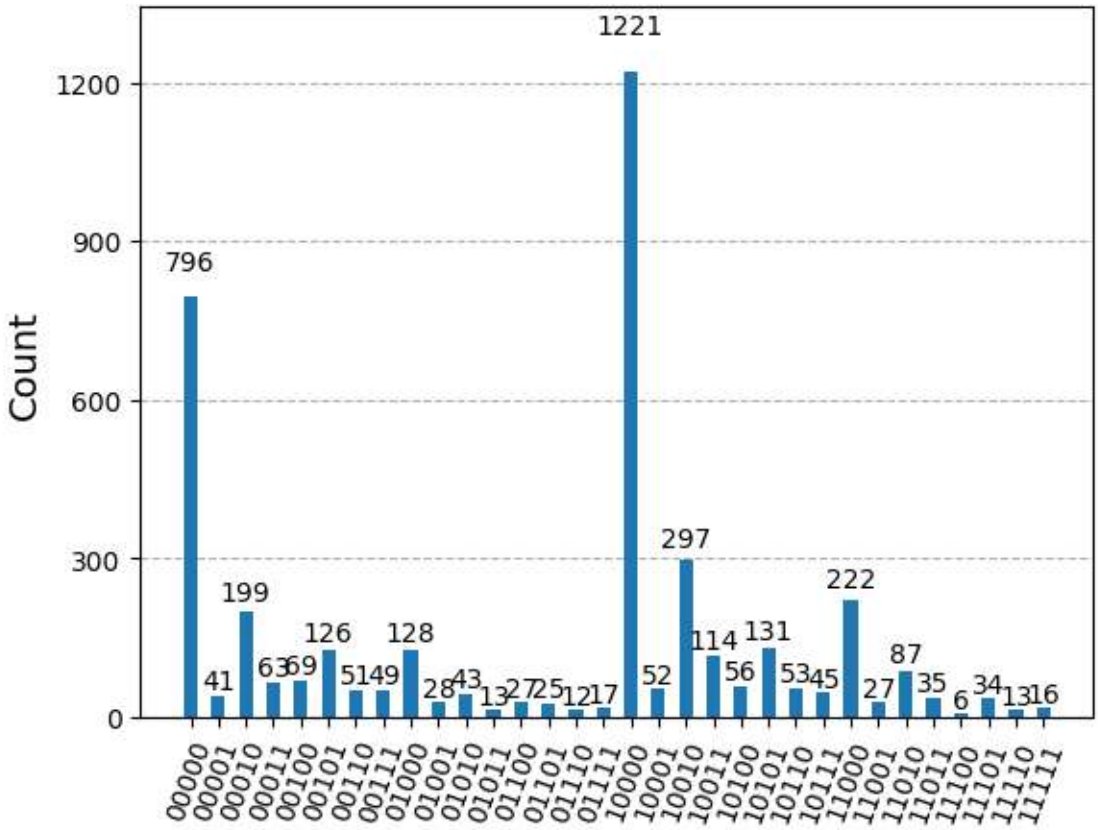
[18]:
```
job.status()
```

[18]:
```
'DONE'
```

[18]:
```
pub_result = result[0].data.meas.get_counts()
```

[19]:
```
plot_histogram(pub_result)
```

[19]:



0.3 Circuit with named registers

[19]:
```
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit

qr = QuantumRegister(3, 'q')
anc = QuantumRegister(1, 'ancilla')
cr = ClassicalRegister(3, 'c')
qc = QuantumCircuit(qr, anc, cr)

qc.x(anc[0])
```

```
qc.h(anc[0])
qc.h(qr[0:3])
qc.cx(qr[0:3], anc[0])
qc.h(qr[0:3])
qc.barrier(qr)
qc.measure(qr, cr)# |1> -> |0> - |1>

qc.draw()
```

[19]:
```
     q_0:  H        H         M

     q_1:  H          H        M

     q_2:  H            H      M

  ancilla:  X  H  X  X  X

     c: 3/
                                    0  1  2
```

[20]:
```
backend = service.least_busy(operational=True, simulator=False)
```

[21]:
```
transpiled_qc = transpile(qc, backend=backend)
transpiled_qc.draw()
```

[21]:
```
global phase: 3 /4
                                                                      »
     ancilla34_0 -> 0                                                 »
                                                                      »
     ancilla34_1 -> 1                                                 »
                                                                      »
     ancilla34_2 -> 2                                                 »
                                                                      »
     ancilla34_3 -> 3                                                 »
                                                                      »
     ancilla34_4 -> 4                                                 »
                                                                      »
     ancilla34_5 -> 5                                                 »
                                                                      »
     ancilla34_6 -> 6                                                 »
                                                                      »
     ancilla34_7 -> 7                                                 »
                                                                      »
     ancilla34_8 -> 8                                                 »
                                                                      »
     ancilla34_9 -> 9                                                 »
```

```
                                                                      »
ancilla34_10 -> 10                                                    »
                                                                      »
ancilla34_11 -> 11                                                    »
                                                                      »
ancilla34_12 -> 12                                                    »
                                                                      »
ancilla34_13 -> 13                                                    »
                                                                      »
ancilla34_14 -> 14                                                    »
                                                                      »
ancilla34_15 -> 15                                                    »
                                                                      »
ancilla34_16 -> 16                                                    »
                                                                      »
ancilla34_17 -> 17                                                    »
                                                                      »
ancilla34_18 -> 18                                                    »
                                                                      »
ancilla34_19 -> 19                                                    »
                                                                      »
ancilla34_20 -> 20                                                    »
                                                                      »
ancilla34_21 -> 21                                                    »
                                                                      »
ancilla34_22 -> 22                                                    »
                                                                      »
ancilla34_23 -> 23                                                    »
                                                                      »
ancilla34_24 -> 24                                                    »
                                                                      »
ancilla34_25 -> 25                                                    »
                                                                      »
ancilla34_26 -> 26                                                    »
                                                                      »
ancilla34_27 -> 27                                                    »
                                                                      »
ancilla34_28 -> 28                                                    »
                                                                      »
ancilla34_29 -> 29                                                    »
                                                                      »
ancilla34_30 -> 30                                                    »
                                                                      »
ancilla34_31 -> 31                                                    »
                                                                      »
ancilla34_32 -> 32                                                    »
                                                                      »
```

```
ancilla34_33 -> 33                    »
                                                                »
ancilla34_34 -> 34                    »
                                                                »
ancilla34_35 -> 35                    »
                                                                »
ancilla34_36 -> 36                    »
                                                                »
ancilla34_37 -> 37                    »
                                                                »
ancilla34_38 -> 38                    »
                                                                »
ancilla34_39 -> 39                    »
                                                                »
ancilla34_40 -> 40                    »
                                                                »
ancilla34_41 -> 41                    »
                                                                »
ancilla34_42 -> 42                    »
                                                                »
ancilla34_43 -> 43                    »
                                                                »
ancilla34_44 -> 44                    »
                                                                »
ancilla34_45 -> 45                    »
                                                                »
ancilla34_46 -> 46                    »
                                                                »
ancilla34_47 -> 47                    »
                                                                »
ancilla34_48 -> 48                    »
                                                                »
ancilla34_49 -> 49                    »
                                                                »
ancilla34_50 -> 50                    »
                                                                »
ancilla34_51 -> 51                    »
                                                                »
ancilla34_52 -> 52                    »
                                                                »
ancilla34_53 -> 53                    »
                                                                »
ancilla34_54 -> 54                    »
                                                                »
ancilla34_55 -> 55                    »
                                                                »
ancilla34_56 -> 56                    »
```

```
                                                                      »
ancilla34_57 -> 57                                                    »
                                                                      »
ancilla34_58 -> 58                                                    »
                                                                      »
ancilla34_59 -> 59                                                    »
                                                                      »
ancilla34_60 -> 60                                                    »
                                                                      »
ancilla34_61 -> 61                                                    »
                                                                      »
ancilla34_62 -> 62                                                    »
                                                                      »
ancilla34_63 -> 63                                                    »
                                                                      »
ancilla34_64 -> 64                                                    »
                                                                      »
ancilla34_65 -> 65                                                    »
                                                                      »
ancilla34_66 -> 66                                                    »
                                                                      »
ancilla34_67 -> 67                                                    »
                                                                      »
ancilla34_68 -> 68                                                    »
                                                                      »
ancilla34_69 -> 69                                                    »
                                                                      »
ancilla34_70 -> 70                                                    »
                                                                      »
ancilla34_71 -> 71                                                    »
                                                                      »
ancilla34_72 -> 72                                                    »
                                                                      »
ancilla34_73 -> 73                                                    »
                                                                      »
ancilla34_74 -> 74                                                    »
                                                                      »
ancilla34_75 -> 75                                                    »
                                                                      »
ancilla34_76 -> 76                                                    »
                                                                      »
ancilla34_77 -> 77                                                    »
                                                                      »
ancilla34_78 -> 78                                                    »
                                                                      »
ancilla34_79 -> 79                                                    »
                                                                      »
```

```
  ancilla34_80 -> 80                    »
                                                                »
  ancilla34_81 -> 81                    »
                                                                »
  ancilla34_82 -> 82                    »
                                                                »
  ancilla34_83 -> 83                    »
                                                                »
  ancilla34_84 -> 84                    »
                                                                »
  ancilla34_85 -> 85                    »
                                                                »
  ancilla34_86 -> 86                    »
                                                                »
  ancilla34_87 -> 87                    »
                                                                »
  ancilla34_88 -> 88                    »
                                                                »
  ancilla34_89 -> 89                    »
                                                                »
  ancilla34_90 -> 90                    »
                                                                »
  ancilla34_91 -> 91                    »
                                                                »
  ancilla34_92 -> 92                    »
                                                                »
  ancilla34_93 -> 93                    »
                                                                »
  ancilla34_94 -> 94                    »
                                                                »
  ancilla34_95 -> 95                    »
                                                                »
  ancilla34_96 -> 96                    »
                                                                »
  ancilla34_97 -> 97                    »
                                                                »
  ancilla34_98 -> 98                    »
                                                                »
  ancilla34_99 -> 99                    »
                                                                »
ancilla34_100 -> 100                    »
                                                                »
ancilla34_101 -> 101                    »
                                                                »
ancilla34_102 -> 102                    »
                                                                »
ancilla34_103 -> 103                    »
```

```
                                                                                »
ancilla34_104 -> 104                                                            »
                                                                                »
ancilla34_105 -> 105                                                            »
                                                                                »
ancilla34_106 -> 106                                                            »
                                                                                »
ancilla34_107 -> 107                                                            »
                                                                                »
ancilla34_108 -> 108                                                            »
                                                                                »
ancilla34_109 -> 109                                                            »
                                                                                »
ancilla34_110 -> 110                                                            »
                                                                                »
          q_0 -> 111   Rz(- /2)    √X    Rz(-0.85352)    √X    Rz(- /2)  »
                                                                                »
ancilla34_111 -> 112                                                            »
                                                                                »
ancilla34_112 -> 113                                                            »
                                                                                »
ancilla34_113 -> 114                                                            »
                                                                                »
ancilla34_114 -> 115                                                            »
                                                                                »
ancilla34_115 -> 116                                                            »
                                                                                »
ancilla34_116 -> 117                                                            »
                                                                                »
ancilla34_117 -> 118                                                            »
                                                                                »
ancilla34_118 -> 119                                                            »
                                                                                »
ancilla34_119 -> 120                                                            »
                                                                                      »
          q_2 -> 121   Rz( /2)    √X                    »
                                                                                      »
    ancilla_0 -> 122    X                        »
                                                                                      »
          q_1 -> 123   Rz( /2)    √X                    »
                                                                                      »
ancilla34_120 -> 124                                                            »
                                                                                »
ancilla34_121 -> 125                                                            »
                                                                                »
ancilla34_122 -> 126                                                            »
                                                                                »
```

```
               c: 3/                          »
                                                 »
«                                                »
«    ancilla34_0 -> 0                       »
«                                                »
«    ancilla34_1 -> 1                       »
«                                                »
«    ancilla34_2 -> 2                       »
«                                                »
«    ancilla34_3 -> 3                       »
«                                                »
«    ancilla34_4 -> 4                       »
«                                                »
«    ancilla34_5 -> 5                       »
«                                                »
«    ancilla34_6 -> 6                       »
«                                                »
«    ancilla34_7 -> 7                       »
«                                                »
«    ancilla34_8 -> 8                       »
«                                                »
«    ancilla34_9 -> 9                       »
«                                                »
«  ancilla34_10 -> 10                       »
«                                                »
«  ancilla34_11 -> 11                       »
«                                                »
«  ancilla34_12 -> 12                       »
«                                                »
«  ancilla34_13 -> 13                       »
«                                                »
«  ancilla34_14 -> 14                       »
«                                                »
«  ancilla34_15 -> 15                       »
«                                                »
«  ancilla34_16 -> 16                       »
«                                                »
«  ancilla34_17 -> 17                       »
«                                                »
«  ancilla34_18 -> 18                       »
«                                                »
«  ancilla34_19 -> 19                       »
«                                                »
«  ancilla34_20 -> 20                       »
«                                                »
«  ancilla34_21 -> 21                       »
«                                                »
```

```
«  ancilla34_22 -> 22                    »
«                                                                »
«  ancilla34_23 -> 23                    »
«                                                                »
«  ancilla34_24 -> 24                    »
«                                                                »
«  ancilla34_25 -> 25                    »
«                                                                »
«  ancilla34_26 -> 26                    »
«                                                                »
«  ancilla34_27 -> 27                    »
«                                                                »
«  ancilla34_28 -> 28                    »
«                                                                »
«  ancilla34_29 -> 29                    »
«                                                                »
«  ancilla34_30 -> 30                    »
«                                                                »
«  ancilla34_31 -> 31                    »
«                                                                »
«  ancilla34_32 -> 32                    »
«                                                                »
«  ancilla34_33 -> 33                    »
«                                                                »
«  ancilla34_34 -> 34                    »
«                                                                »
«  ancilla34_35 -> 35                    »
«                                                                »
«  ancilla34_36 -> 36                    »
«                                                                »
«  ancilla34_37 -> 37                    »
«                                                                »
«  ancilla34_38 -> 38                    »
«                                                                »
«  ancilla34_39 -> 39                    »
«                                                                »
«  ancilla34_40 -> 40                    »
«                                                                »
«  ancilla34_41 -> 41                    »
«                                                                »
«  ancilla34_42 -> 42                    »
«                                                                »
«  ancilla34_43 -> 43                    »
«                                                                »
«  ancilla34_44 -> 44                    »
«                                                                »
«  ancilla34_45 -> 45                    »
```

```
«                                                                    »
«  ancilla34_46 -> 46                                                »
«                                                                    »
«  ancilla34_47 -> 47                                                »
«                                                                    »
«  ancilla34_48 -> 48                                                »
«                                                                    »
«  ancilla34_49 -> 49                                                »
«                                                                    »
«  ancilla34_50 -> 50                                                »
«                                                                    »
«  ancilla34_51 -> 51                                                »
«                                                                    »
«  ancilla34_52 -> 52                                                »
«                                                                    »
«  ancilla34_53 -> 53                                                »
«                                                                    »
«  ancilla34_54 -> 54                                                »
«                                                                    »
«  ancilla34_55 -> 55                                                »
«                                                                    »
«  ancilla34_56 -> 56                                                »
«                                                                    »
«  ancilla34_57 -> 57                                                »
«                                                                    »
«  ancilla34_58 -> 58                                                »
«                                                                    »
«  ancilla34_59 -> 59                                                »
«                                                                    »
«  ancilla34_60 -> 60                                                »
«                                                                    »
«  ancilla34_61 -> 61                                                »
«                                                                    »
«  ancilla34_62 -> 62                                                »
«                                                                    »
«  ancilla34_63 -> 63                                                »
«                                                                    »
«  ancilla34_64 -> 64                                                »
«                                                                    »
«  ancilla34_65 -> 65                                                »
«                                                                    »
«  ancilla34_66 -> 66                                                »
«                                                                    »
«  ancilla34_67 -> 67                                                »
«                                                                    »
«  ancilla34_68 -> 68                                                »
«                                                                    »
```

```
«  ancilla34_69 -> 69                          »
«                                                                  »
«  ancilla34_70 -> 70                          »
«                                                                  »
«  ancilla34_71 -> 71                          »
«                                                                  »
«  ancilla34_72 -> 72                          »
«                                                                  »
«  ancilla34_73 -> 73                          »
«                                                                  »
«  ancilla34_74 -> 74                          »
«                                                                  »
«  ancilla34_75 -> 75                          »
«                                                                  »
«  ancilla34_76 -> 76                          »
«                                                                  »
«  ancilla34_77 -> 77                          »
«                                                                  »
«  ancilla34_78 -> 78                          »
«                                                                  »
«  ancilla34_79 -> 79                          »
«                                                                  »
«  ancilla34_80 -> 80                          »
«                                                                  »
«  ancilla34_81 -> 81                          »
«                                                                  »
«  ancilla34_82 -> 82                          »
«                                                                  »
«  ancilla34_83 -> 83                          »
«                                                                  »
«  ancilla34_84 -> 84                          »
«                                                                  »
«  ancilla34_85 -> 85                          »
«                                                                  »
«  ancilla34_86 -> 86                          »
«                                                                  »
«  ancilla34_87 -> 87                          »
«                                                                  »
«  ancilla34_88 -> 88                          »
«                                                                  »
«  ancilla34_89 -> 89                          »
«                                                                  »
«  ancilla34_90 -> 90                          »
«                                                                  »
«  ancilla34_91 -> 91                          »
«                                                                  »
«  ancilla34_92 -> 92                          »
```

```
«                                                                                »
«  ancilla34_93 -> 93                          »
«                                                                                »
«  ancilla34_94 -> 94                          »
«                                                                                »
«  ancilla34_95 -> 95                          »
«                                                                                »
«  ancilla34_96 -> 96                          »
«                                                                                »
«  ancilla34_97 -> 97                          »
«                                                                                »
«  ancilla34_98 -> 98                          »
«                                                                                »
«  ancilla34_99 -> 99                          »
«                                                                                »
«ancilla34_100 -> 100                          »
«                                                                                »
«ancilla34_101 -> 101                          »
«                                                                                »
«ancilla34_102 -> 102                          »
«                                                                                »
«ancilla34_103 -> 103                          »
«                                                                                »
«ancilla34_104 -> 104                          »
«                                                                                »
«ancilla34_105 -> 105                          »
«                                                                                »
«ancilla34_106 -> 106                          »
«                                                                                »
«ancilla34_107 -> 107                          »
«                                                                                »
«ancilla34_108 -> 108                          »
«                                                                                »
«ancilla34_109 -> 109                          »
«                                                                                »
«ancilla34_110 -> 110                          »
«                                                 »
«          q_0 -> 111  1       Rz( /2)    √X   Rz(-2.4243)    √X  »
«                                                       »
«ancilla34_111 -> 112                             »
«                                                                              »
«ancilla34_112 -> 113                             »
«                                                                              »
«ancilla34_113 -> 114                             »
«                                                                              »
«ancilla34_114 -> 115                             »
«                                                                              »
```

```
«ancilla34_115 -> 116                                »
«                       Ecr                                »
«ancilla34_116 -> 117                                »
«                                                          »
«ancilla34_117 -> 118                                »
«                                                          »
«ancilla34_118 -> 119                                »
«                                                          »
«ancilla34_119 -> 120                                »
«                                                          »
«          q_2 -> 121                                »
«                                                      »
«    ancilla_0 -> 122  0       Rz(-/2)   √X    Rz(-/2)   1      »
«                                                 Ecr  »
«          q_1 -> 123                        0      »
«                                                                »
«ancilla34_120 -> 124                           »
«                                                                »
«ancilla34_121 -> 125                           »
«                                                                »
«ancilla34_122 -> 126                           »
«                                                                »
«                c: 3/                          »
«                                                                »
«                                                                     »
«    ancilla34_0 -> 0                              »
«                                                                     »
«    ancilla34_1 -> 1                              »
«                                                                     »
«    ancilla34_2 -> 2                              »
«                                                                     »
«    ancilla34_3 -> 3                              »
«                                                                     »
«    ancilla34_4 -> 4                              »
«                                                                     »
«    ancilla34_5 -> 5                              »
«                                                                     »
«    ancilla34_6 -> 6                              »
«                                                                     »
«    ancilla34_7 -> 7                              »
«                                                                     »
«    ancilla34_8 -> 8                              »
«                                                                     »
«    ancilla34_9 -> 9                              »
«                                                                     »
«  ancilla34_10 -> 10                              »
«                                                                     »
```

```
«  ancilla34_11 -> 11                          »
«                                                                    »
«  ancilla34_12 -> 12                          »
«                                                                    »
«  ancilla34_13 -> 13                          »
«                                                                    »
«  ancilla34_14 -> 14                          »
«                                                                    »
«  ancilla34_15 -> 15                          »
«                                                                    »
«  ancilla34_16 -> 16                          »
«                                                                    »
«  ancilla34_17 -> 17                          »
«                                                                    »
«  ancilla34_18 -> 18                          »
«                                                                    »
«  ancilla34_19 -> 19                          »
«                                                                    »
«  ancilla34_20 -> 20                          »
«                                                                    »
«  ancilla34_21 -> 21                          »
«                                                                    »
«  ancilla34_22 -> 22                          »
«                                                                    »
«  ancilla34_23 -> 23                          »
«                                                                    »
«  ancilla34_24 -> 24                          »
«                                                                    »
«  ancilla34_25 -> 25                          »
«                                                                    »
«  ancilla34_26 -> 26                          »
«                                                                    »
«  ancilla34_27 -> 27                          »
«                                                                    »
«  ancilla34_28 -> 28                          »
«                                                                    »
«  ancilla34_29 -> 29                          »
«                                                                    »
«  ancilla34_30 -> 30                          »
«                                                                    »
«  ancilla34_31 -> 31                          »
«                                                                    »
«  ancilla34_32 -> 32                          »
«                                                                    »
«  ancilla34_33 -> 33                          »
«                                                                    »
«  ancilla34_34 -> 34                          »
```

```
«                                                                    »
«  ancilla34_35 -> 35                                                »
«                                                                    »
«  ancilla34_36 -> 36                                                »
«                                                                    »
«  ancilla34_37 -> 37                                                »
«                                                                    »
«  ancilla34_38 -> 38                                                »
«                                                                    »
«  ancilla34_39 -> 39                                                »
«                                                                    »
«  ancilla34_40 -> 40                                                »
«                                                                    »
«  ancilla34_41 -> 41                                                »
«                                                                    »
«  ancilla34_42 -> 42                                                »
«                                                                    »
«  ancilla34_43 -> 43                                                »
«                                                                    »
«  ancilla34_44 -> 44                                                »
«                                                                    »
«  ancilla34_45 -> 45                                                »
«                                                                    »
«  ancilla34_46 -> 46                                                »
«                                                                    »
«  ancilla34_47 -> 47                                                »
«                                                                    »
«  ancilla34_48 -> 48                                                »
«                                                                    »
«  ancilla34_49 -> 49                                                »
«                                                                    »
«  ancilla34_50 -> 50                                                »
«                                                                    »
«  ancilla34_51 -> 51                                                »
«                                                                    »
«  ancilla34_52 -> 52                                                »
«                                                                    »
«  ancilla34_53 -> 53                                                »
«                                                                    »
«  ancilla34_54 -> 54                                                »
«                                                                    »
«  ancilla34_55 -> 55                                                »
«                                                                    »
«  ancilla34_56 -> 56                                                »
«                                                                    »
«  ancilla34_57 -> 57                                                »
«                                                                    »
```

```
«  ancilla34_58 -> 58                          »
«                                                                  »
«  ancilla34_59 -> 59                          »
«                                                                  »
«  ancilla34_60 -> 60                          »
«                                                                  »
«  ancilla34_61 -> 61                          »
«                                                                  »
«  ancilla34_62 -> 62                          »
«                                                                  »
«  ancilla34_63 -> 63                          »
«                                                                  »
«  ancilla34_64 -> 64                          »
«                                                                  »
«  ancilla34_65 -> 65                          »
«                                                                  »
«  ancilla34_66 -> 66                          »
«                                                                  »
«  ancilla34_67 -> 67                          »
«                                                                  »
«  ancilla34_68 -> 68                          »
«                                                                  »
«  ancilla34_69 -> 69                          »
«                                                                  »
«  ancilla34_70 -> 70                          »
«                                                                  »
«  ancilla34_71 -> 71                          »
«                                                                  »
«  ancilla34_72 -> 72                          »
«                                                                  »
«  ancilla34_73 -> 73                          »
«                                                                  »
«  ancilla34_74 -> 74                          »
«                                                                  »
«  ancilla34_75 -> 75                          »
«                                                                  »
«  ancilla34_76 -> 76                          »
«                                                                  »
«  ancilla34_77 -> 77                          »
«                                                                  »
«  ancilla34_78 -> 78                          »
«                                                                  »
«  ancilla34_79 -> 79                          »
«                                                                  »
«  ancilla34_80 -> 80                          »
«                                                                  »
«  ancilla34_81 -> 81                          »
```

```
«                                                                    »
«  ancilla34_82 -> 82                                                »
«                                                                    »
«  ancilla34_83 -> 83                                                »
«                                                                    »
«  ancilla34_84 -> 84                                                »
«                                                                    »
«  ancilla34_85 -> 85                                                »
«                                                                    »
«  ancilla34_86 -> 86                                                »
«                                                                    »
«  ancilla34_87 -> 87                                                »
«                                                                    »
«  ancilla34_88 -> 88                                                »
«                                                                    »
«  ancilla34_89 -> 89                                                »
«                                                                    »
«  ancilla34_90 -> 90                                                »
«                                                                    »
«  ancilla34_91 -> 91                                                »
«                                                                    »
«  ancilla34_92 -> 92                                                »
«                                                                    »
«  ancilla34_93 -> 93                                                »
«                                                                    »
«  ancilla34_94 -> 94                                                »
«                                                                    »
«  ancilla34_95 -> 95                                                »
«                                                                    »
«  ancilla34_96 -> 96                                                »
«                                                                    »
«  ancilla34_97 -> 97                                                »
«                                                                    »
«  ancilla34_98 -> 98                                                »
«                                                                    »
«  ancilla34_99 -> 99                                                »
«                                                                    »
«ancilla34_100 -> 100                                                »
«                                                                    »
«ancilla34_101 -> 101                                                »
«                                                                    »
«ancilla34_102 -> 102                                                »
«                                                                    »
«ancilla34_103 -> 103                                                »
«                                                                    »
«ancilla34_104 -> 104                                                »
«                                                                    »
```

```
«ancilla34_105 -> 105                                »
«                                                                                »
«ancilla34_106 -> 106                                »
«                                                                                »
«ancilla34_107 -> 107                                »
«                                                                                »
«ancilla34_108 -> 108                                »
«                                                                                »
«ancilla34_109 -> 109                                »
«                                                                                »
«ancilla34_110 -> 110                                »
«                                                                          »
«          q_0 -> 111   Rz( /2)                         »
«                                                                          »
«ancilla34_111 -> 112                                »
«                                                                                »
«ancilla34_112 -> 113                                »
«                                                                                »
«ancilla34_113 -> 114                                »
«                                                                                »
«ancilla34_114 -> 115                                »
«                                                                                »
«ancilla34_115 -> 116                                »
«                                                                                »
«ancilla34_116 -> 117                                »
«                                                                                »
«ancilla34_117 -> 118                                »
«                                                                                »
«ancilla34_118 -> 119                                »
«                                                                                »
«ancilla34_119 -> 120                                »
«                                                                    »
«          q_2 -> 121                 0       Rz( /2)   √X  »
«                                        Ecr             »
«    ancilla_0 -> 122   Rz(- )   √X    Rz(- )   1                 »
«                                                          »
«          q_1 -> 123   Rz( /2)   √X   Rz(- /2)                 »
«                                                                »
«ancilla34_120 -> 124                                »
«                                                                                »
«ancilla34_121 -> 125                                »
«                                                                                »
«ancilla34_122 -> 126                                »
«                                                                                »
«               c: 3/                                »
«                                                                                »
«
```

```
«    ancilla34_0 -> 0
«
«    ancilla34_1 -> 1
«
«    ancilla34_2 -> 2
«
«    ancilla34_3 -> 3
«
«    ancilla34_4 -> 4
«
«    ancilla34_5 -> 5
«
«    ancilla34_6 -> 6
«
«    ancilla34_7 -> 7
«
«    ancilla34_8 -> 8
«
«    ancilla34_9 -> 9
«
«  ancilla34_10 -> 10
«
«  ancilla34_11 -> 11
«
«  ancilla34_12 -> 12
«
«  ancilla34_13 -> 13
«
«  ancilla34_14 -> 14
«
«  ancilla34_15 -> 15
«
«  ancilla34_16 -> 16
«
«  ancilla34_17 -> 17
«
«  ancilla34_18 -> 18
«
«  ancilla34_19 -> 19
«
«  ancilla34_20 -> 20
«
«  ancilla34_21 -> 21
«
«  ancilla34_22 -> 22
«
«  ancilla34_23 -> 23
```

```
«
«  ancilla34_24 -> 24
«
«  ancilla34_25 -> 25
«
«  ancilla34_26 -> 26
«
«  ancilla34_27 -> 27
«
«  ancilla34_28 -> 28
«
«  ancilla34_29 -> 29
«
«  ancilla34_30 -> 30
«
«  ancilla34_31 -> 31
«
«  ancilla34_32 -> 32
«
«  ancilla34_33 -> 33
«
«  ancilla34_34 -> 34
«
«  ancilla34_35 -> 35
«
«  ancilla34_36 -> 36
«
«  ancilla34_37 -> 37
«
«  ancilla34_38 -> 38
«
«  ancilla34_39 -> 39
«
«  ancilla34_40 -> 40
«
«  ancilla34_41 -> 41
«
«  ancilla34_42 -> 42
«
«  ancilla34_43 -> 43
«
«  ancilla34_44 -> 44
«
«  ancilla34_45 -> 45
«
«  ancilla34_46 -> 46
«
```

```
«  ancilla34_47 -> 47
«
«  ancilla34_48 -> 48
«
«  ancilla34_49 -> 49
«
«  ancilla34_50 -> 50
«
«  ancilla34_51 -> 51
«
«  ancilla34_52 -> 52
«
«  ancilla34_53 -> 53
«
«  ancilla34_54 -> 54
«
«  ancilla34_55 -> 55
«
«  ancilla34_56 -> 56
«
«  ancilla34_57 -> 57
«
«  ancilla34_58 -> 58
«
«  ancilla34_59 -> 59
«
«  ancilla34_60 -> 60
«
«  ancilla34_61 -> 61
«
«  ancilla34_62 -> 62
«
«  ancilla34_63 -> 63
«
«  ancilla34_64 -> 64
«
«  ancilla34_65 -> 65
«
«  ancilla34_66 -> 66
«
«  ancilla34_67 -> 67
«
«  ancilla34_68 -> 68
«
«  ancilla34_69 -> 69
«
«  ancilla34_70 -> 70
```

```
«
«  ancilla34_71 -> 71
«
«  ancilla34_72 -> 72
«
«  ancilla34_73 -> 73
«
«  ancilla34_74 -> 74
«
«  ancilla34_75 -> 75
«
«  ancilla34_76 -> 76
«
«  ancilla34_77 -> 77
«
«  ancilla34_78 -> 78
«
«  ancilla34_79 -> 79
«
«  ancilla34_80 -> 80
«
«  ancilla34_81 -> 81
«
«  ancilla34_82 -> 82
«
«  ancilla34_83 -> 83
«
«  ancilla34_84 -> 84
«
«  ancilla34_85 -> 85
«
«  ancilla34_86 -> 86
«
«  ancilla34_87 -> 87
«
«  ancilla34_88 -> 88
«
«  ancilla34_89 -> 89
«
«  ancilla34_90 -> 90
«
«  ancilla34_91 -> 91
«
«  ancilla34_92 -> 92
«
«  ancilla34_93 -> 93
«
```

```
«  ancilla34_94 -> 94
«
«  ancilla34_95 -> 95
«
«  ancilla34_96 -> 96
«
«  ancilla34_97 -> 97
«
«  ancilla34_98 -> 98
«
«  ancilla34_99 -> 99
«
«ancilla34_100 -> 100
«
«ancilla34_101 -> 101
«
«ancilla34_102 -> 102
«
«ancilla34_103 -> 103
«
«ancilla34_104 -> 104
«
«ancilla34_105 -> 105
«
«ancilla34_106 -> 106
«
«ancilla34_107 -> 107
«
«ancilla34_108 -> 108
«
«ancilla34_109 -> 109
«
«ancilla34_110 -> 110
«
«          q_0 -> 111          M
«
«ancilla34_111 -> 112
«
«ancilla34_112 -> 113
«
«ancilla34_113 -> 114
«
«ancilla34_114 -> 115
«
«ancilla34_115 -> 116
«
«ancilla34_116 -> 117
```

```
«
«ancilla34_117 -> 118
«
«ancilla34_118 -> 119
«
«ancilla34_119 -> 120
«
«          q_2 -> 121   Rz(- /2)       M
«
«    ancilla_0 -> 122
«
«          q_1 -> 123           M
«
«ancilla34_120 -> 124
«
«ancilla34_121 -> 125
«
«ancilla34_122 -> 126
«
«                c: 3/
«                                    0  1  2
```

[24]:
```
from qiskit_ibm_runtime import Batch
with Batch(backend=backend) as batch:
    sampler = SamplerV2()
    job1_async = sampler.run([transpiled_qc])
    job2_async = sampler.run([transpiled_qc])
```

[25]:
```
job1_async.status()
```

[25]:
```
'RUNNING'
```

[26]:
```
job2_async.status()
```

[26]:
```
'RUNNING'
```

[ ]:
```

```

30-backend_info

May 26, 2026

1 Obtaining information about your `backend`

Note: All the attributes of the backend are described in detail in the Qiskit Backend Specifications. This page reviews a subset of the spec. Programming a quantum computer at the microwave pulse level requires more information about the device than is required at the circuit level. A quantum circuit is built for an abstract quantum computer – it will yield the same quantum state on any quantum computer (except for varying performance levels). A pulse schedule, on the other hand, is so specific to the device, that running one program on two different backends is not expected to have the same result, even on perfectly noiseless systems.

As a basic example, imagine a drive pulse `q0_X180` calibrated on qubit 0 to enact an $X180$ pulse, which flips the state of qubit 0. If we use the samples from that pulse on qubit 1 on the same device, or qubit 0 on another device, we do not know what the resulting state will be – but we can be pretty sure it won't be an $X180$ operation. The qubits are each unique, with various drive coupling strengths. If we have specified a frequency for the drive pulse, it's very probable that pulse would have little effect on another qubit, which has its own resonant frequency.

With that, we have motivated why information from the backend may be very useful at times for building Pulse schedules. The information included in a `backend` is broken into three main parts:

- Configuration: static backend features
- Properties: measured and reported backend characteristics
- Defaults: default settings for the OpenPulse-enabled backend

which are each covered in the following sections. While all three of these contain interesting data for Pulse users, the defaults are *only* provided for backends enabled with OpenPulse.

The first thing you'll need to do is grab a backend to inspect. Here we use a mocked backend that contains a snapshot of data from the real OpenPulse-enabled backend.

[1]:
```
%load_ext dotenv
%dotenv
```

[2]:
```
import os
from qiskit_ibm_runtime import QiskitRuntimeService, SamplerV2
from qiskit.visualization import plot_histogram
```

[3]:
```
token = os.environ["IBMQ_TOKEN"]
service = QiskitRuntimeService(
                token=token,
```

```
                channel="ibm_quantum",
            )
backend = service.least_busy(operational=True, simulator=False)
```

```
/tmp/ipykernel_9744/3214271246.py:2: DeprecationWarning: The "ibm_quantum"
channel option is deprecated and will be sunset on 1 July. After this date,
ibm_cloud will be the only valid channel. For information on migrating to the
new IBM Quantum Platform on the "ibm_cloud" channel, review the migration guide
https://quantum.cloud.ibm.com/docs/migration-guides/classic-iqp-to-cloud-iqp .
  service = QiskitRuntimeService(
```

1.1 Configuration

The configuration is where you'll find data about the static setup of the device, such as its name, version, the number of qubits, and the types of features it supports.

Let's build a description of our backend using information from the `backend`'s config.

[4]:
```
config = backend.configuration()

# Basic Features
print("This backend is called {0}, and is on version {1}. It has {2} qubit{3}.␣
 ↪It "
      "{4} OpenPulse programs. The basis gates supported on this device are {5}.
 ↪"
      "".format(config.backend_name,
                config.backend_version,
                config.n_qubits,
                '' if config.n_qubits == 1 else 's',
                'supports' if config.open_pulse else 'does not support',
                config.basis_gates))
```

```
This backend is called ibm_brisbane, and is on version 1.1.115. It has 127
qubits. It supports OpenPulse programs. The basis gates supported on this device
are ['ecr', 'id', 'rz', 'sx', 'x'].
```

Neat! All of the above configuration is available for any backend, whether enabled with OpenPulse or not, although it is not an exhaustive list. There are additional attributes available on Pulse backends. Let's go into a bit more detail with those.

The **timescale**, `dt`, is backend dependent. Think of this as the inverse sampling rate of the control rack's arbitrary waveform generators. Each sample point and duration in a Pulse `Schedule` is given in units of this timescale.

[5]:
```
config.dt  # units of seconds
```

[5]:
```
5e-10
```

The configuration also provides information that is useful for building measurements. Pulse supports three measurement levels: `0: RAW`, `1: KERNELED`, and `2: DISCRIMINATED`. The `meas_levels`

attribute tells us which of those are supported by this backend. To learn how to execute programs with these different levels

[6]:
```
config.meas_levels
```

[6]:
```
[1, 2]
```

For backends which support measurement level 0, the sampling rate of the control rack's analog-to-digital converters (ADCs) also becomes relevant. The configuration also has this info, where `dtm` is the time per sample returned:

[7]:
```
config.dtm
```

[7]:
```
5e-10
```

The measurement map, explained in detail on [this page COMING SOON], is also found here.

[8]:
```
config.meas_map
```

[8]:
```
[[0,
  1,
  2,
  3,
  4,
  5,
  6,
  7,
  8,
  9,
  10,
  11,
  12,
  13,
  14,
  15,
  16,
  17,
  18,
  19,
  20,
  21,
  22,
  23,
  24,
  25,
  26,
  27,
  28,
  29,
```

30,
31,
32,
33,
34,
35,
36,
37,
38,
39,
40,
41,
42,
43,
44,
45,
46,
47,
48,
49,
50,
51,
52,
53,
54,
55,
56,
57,
58,
59,
60,
61,
62,
63,
64,
65,
66,
67,
68,
69,
70,
71,
72,
73,
74,
75,
76,

77,
78,
79,
80,
81,
82,
83,
84,
85,
86,
87,
88,
89,
90,
91,
92,
93,
94,
95,
96,
97,
98,
99,
100,
101,
102,
103,
104,
105,
106,
107,
108,
109,
110,
111,
112,
113,
114,
115,
116,
117,
118,
119,
120,
121,
122,
123,

```
    124,
    125,
    126]]
```

[9]:
```
props = backend.properties()
```

[10]:
```
def describe_qubit(qubit, properties):
    """Print a string describing some of reported properties of the given qubit.
    """

    # Conversion factors from standard SI units
    us = 1e6
    ns = 1e9
    GHz = 1e-9

    print("Qubit {0} has a \n"
          "  - T1 time of {1} microseconds\n"
          "  - T2 time of {2} microseconds\n"
          "  - U2 gate error of {3}\n"
          "  - U2 gate duration of {4} nanoseconds\n"
          "  - resonant frequency of {5} GHz".format(
              qubit,
              properties.t1(qubit) * us,
              properties.t2(qubit) * us,
              properties.gate_error('sx', qubit),
              properties.gate_length('sx', qubit) * ns,
              properties.frequency(qubit) * GHz))

describe_qubit(0, props)
```

```
Qubit 0 has a
  - T1 time of 315.32988680168523 microseconds
  - T2 time of 57.508663902391426 microseconds
  - U2 gate error of 0.0001756228696857236
  - U2 gate duration of 59.99999999999999 nanoseconds
  - resonant frequency of 4.7219057784834355 GHz
```

Properties are not guaranteed to be reported, but backends without Pulse access typically also provide this data.

1.2 Defaults

Unlike the other two sections, `PulseDefaults` are only available for Pulse-enabled backends. It contains the default program settings run on the device.

[11]:
```
defaults = backend.defaults()
```

```
/tmp/ipykernel_9744/3348888366.py:1: DeprecationWarning: The defaults method and
the PulseDefaults class have been deprecated as of qiskit-ibm-runtime 0.38.0 and
```

```
will be removed no sooner than 3 months after the release date. IBM backends no
longer support pulse gates and are no longer used to construct the backend
target.
  defaults = backend.defaults()
```

1.2.1 Drive frequencies

Defaults contains the default frequency settings for the drive and measurement signal channels:

[12]:
```
q0_freq = defaults.qubit_freq_est[0]  # Hz
q0_meas_freq = defaults.meas_freq_est[0]  # Hz

GHz = 1e-9
print("DriveChannel(0) defaults to a modulation frequency of {} GHz.".
  ↪format(q0_freq * GHz))
print("MeasureChannel(0) defaults to a modulation frequency of {} GHz.".
  ↪format(q0_meas_freq * GHz))
```

```
DriveChannel(0) defaults to a modulation frequency of 4.7219057784834355 GHz.
MeasureChannel(0) defaults to a modulation frequency of 7.175428524047674 GHz.
```

1.3 Dynamic backend information

Backends can also have properties that change whenever the backed is calibrated, such as qubit frequency and operation error rates. Backends are usually calibrated every 24 hours, and their properties update after the calibration sequence completes. These properties can be used when optimizing quantum circuits or to construct noise models for a classical simulator.

[13]:
```
backend.qubit_properties(0)  # properties of qubit 0
```

[13]:
```
QubitProperties(t1=0.00031532988680168524, t2=5.750866390239143e-05,
frequency=4721905778.483436)
```

[14]:
```
backend.target["ecr"][(1, 0)]
```

[14]:
```
InstructionProperties(duration=6.6e-07, error=0.0033926162842485563,
calibration=None)
```

[15]:
```
backend.target["measure"][(0,)]
```

[15]:
```
InstructionProperties(duration=1.3e-06, error=0.034423828125, calibration=None)
```

45-noise-revisited

May 26, 2026

0.1 Noise Model Examples

We will now give some examples of noise models. For our demonstrations we will use a simple test circuit generating a n-qubit GHZ state:

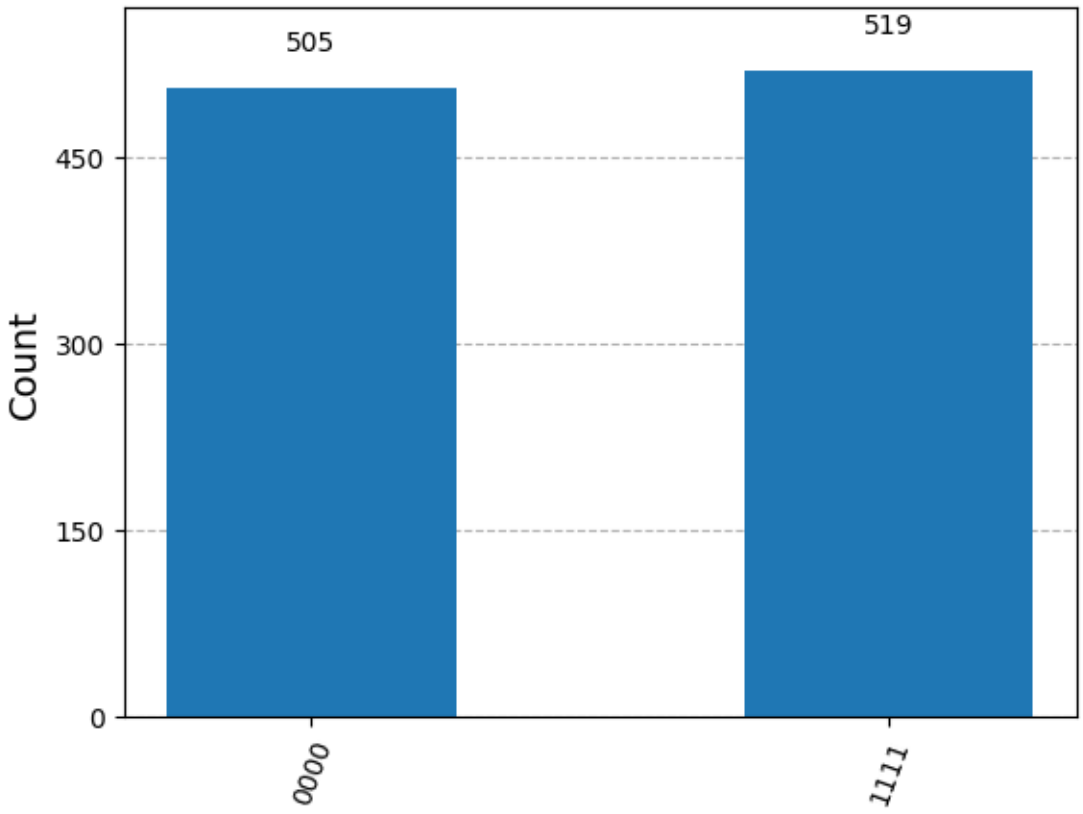
[1]:
```
import numpy as np
from qiskit import QuantumCircuit, transpile
from qiskit.quantum_info import Kraus, SuperOp
from qiskit_aer import AerSimulator
from qiskit.visualization import plot_histogram
# Import from Qiskit Aer noise module
from qiskit_aer.noise import (NoiseModel, QuantumError, ReadoutError,
    pauli_error, depolarizing_error, thermal_relaxation_error)
```

[2]:
```
# System Specification
n_qubits = 4
circ = QuantumCircuit(n_qubits)

# Test Circuit
circ.h(0)
for qubit in range(n_qubits - 1):
    circ.cx(qubit, qubit + 1)
circ.measure_all()
print(circ)
```

```
   q_0:  H           M

   q_1:    X           M

   q_2:      X         M

   q_3:        X       M

meas: 4/
                          0  1  2  3
```

0.1.1 Ideal Simulation

[3]:
```
# Ideal simulator and execution
sim_ideal = AerSimulator()
result_ideal = sim_ideal.run(circ).result()
plot_histogram(result_ideal.get_counts(0))
```

[3]:


0.2 Example 2: T1/T2 thermal relaxation

Now consider a more realistic error model based on thermal relaxation with the qubit environment: * Each qubit is parameterized by a thermal relaxation time constant T_1 and a dephasing time constant T_2. * Note that we must have $T_2 \leq 2T_1$. * Error rates on instructions are determined by gate times and qubit T_1, T_2 values.

[4]:
```
# T1 and T2 values for qubits 0-3
T1s = np.random.normal(50e3, 10e3, 4) # Sampled from normal distribution mean␣
 ↪50 microsec
T2s = np.random.normal(70e3, 10e3, 4)  # Sampled from normal distribution mean␣
 ↪50 microsec

# Truncate random T2s <= T1s
```

```
T2s = np.array([min(T2s[j], 2 * T1s[j]) for j in range(4)])

# Instruction times (in nanoseconds)
time_u1 = 0   # virtual gate
time_u2 = 50  # (single X90 pulse)
time_u3 = 100 # (two X90 pulses)
time_cx = 300
time_reset = 1000  # 1 microsecond
time_measure = 1000 # 1 microsecond

# QuantumError objects
errors_reset = [thermal_relaxation_error(t1, t2, time_reset)
                for t1, t2 in zip(T1s, T2s)]
errors_measure = [thermal_relaxation_error(t1, t2, time_measure)
                  for t1, t2 in zip(T1s, T2s)]
errors_u1  = [thermal_relaxation_error(t1, t2, time_u1)
              for t1, t2 in zip(T1s, T2s)]
errors_u2  = [thermal_relaxation_error(t1, t2, time_u2)
              for t1, t2 in zip(T1s, T2s)]
errors_u3  = [thermal_relaxation_error(t1, t2, time_u3)
              for t1, t2 in zip(T1s, T2s)]
errors_cx = [[thermal_relaxation_error(t1a, t2a, time_cx).expand(
             thermal_relaxation_error(t1b, t2b, time_cx))
              for t1a, t2a in zip(T1s, T2s)]
               for t1b, t2b in zip(T1s, T2s)]

# Add errors to noise model
noise_thermal = NoiseModel()
for j in range(4):
    noise_thermal.add_quantum_error(errors_reset[j], "reset", [j])
    noise_thermal.add_quantum_error(errors_measure[j], "measure", [j])
    noise_thermal.add_quantum_error(errors_u1[j], "u1", [j])
    noise_thermal.add_quantum_error(errors_u2[j], "u2", [j])
    noise_thermal.add_quantum_error(errors_u3[j], "u3", [j])
    for k in range(4):
        noise_thermal.add_quantum_error(errors_cx[j][k], "cx", [j, k])

print(noise_thermal)
```

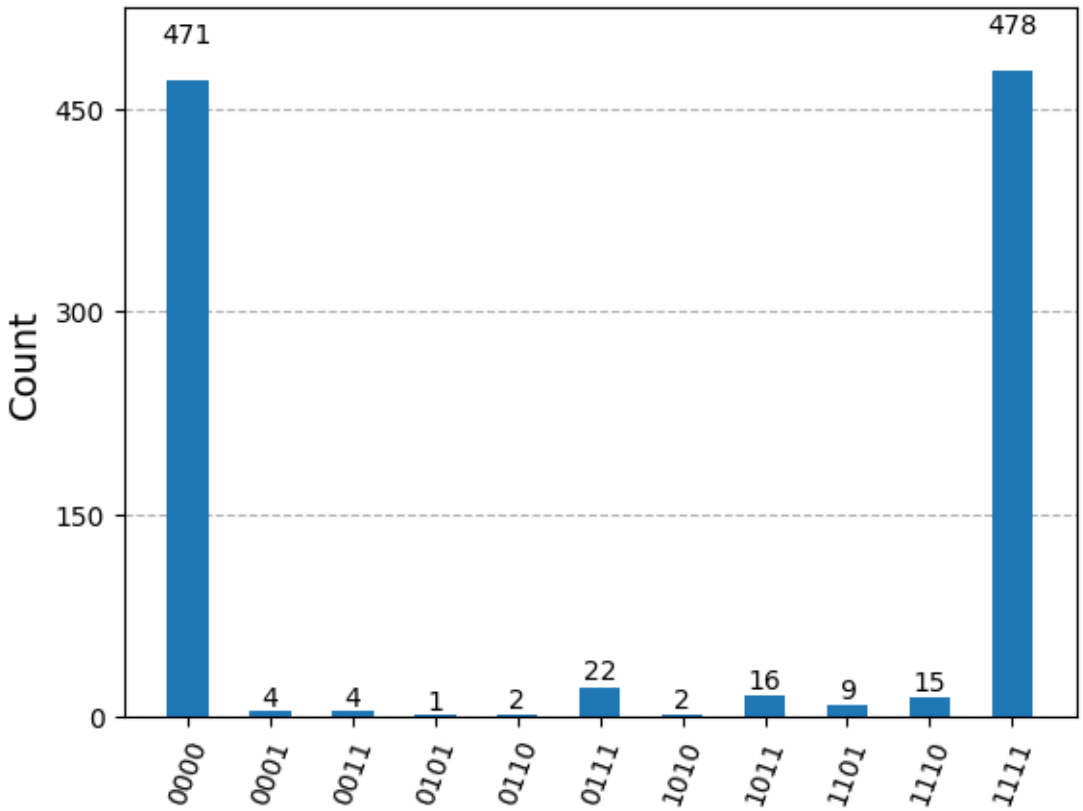
```
NoiseModel:
  Basis gates: ['cx', 'id', 'rz', 'sx', 'u2', 'u3']
  Instructions with noise: ['u3', 'u2', 'cx', 'reset', 'measure']
  Qubits with noise: [0, 1, 2, 3]
  Specific qubit errors: [('reset', (0,)), ('reset', (1,)), ('reset', (2,)),
('reset', (3,)), ('measure', (0,)), ('measure', (1,)), ('measure', (2,)),
('measure', (3,)), ('u2', (0,)), ('u2', (1,)), ('u2', (2,)), ('u2', (3,)),
('u3', (0,)), ('u3', (1,)), ('u3', (2,)), ('u3', (3,)), ('cx', (0, 0)), ('cx',
(0, 1)), ('cx', (0, 2)), ('cx', (0, 3)), ('cx', (1, 0)), ('cx', (1, 1)), ('cx',
```

```
(1, 2)), ('cx', (1, 3)), ('cx', (2, 0)), ('cx', (2, 1)), ('cx', (2, 2)), ('cx',
(2, 3)), ('cx', (3, 0)), ('cx', (3, 1)), ('cx', (3, 2)), ('cx', (3, 3))]
```

0.2.1 Executing the noisy simulation

[5]:
```
# Run the noisy simulation
sim_thermal = AerSimulator(noise_model=noise_thermal)

# Transpile circuit for noisy basis gates
circ_tthermal = transpile(circ, sim_thermal)

# Run and get counts
result_thermal = sim_thermal.run(circ_tthermal).result()
counts_thermal = result_thermal.get_counts(0)

# Plot noisy output
plot_histogram(counts_thermal)
```

[5]:



[ ]:

46-device-noise

May 26, 2026

0.1 Introduction

This notebook shows how to use the Qiskit Aer `noise` module to automatically generate a basic noise model for an IBMQ hardware device, and use this model to do noisy simulations of QuantumCircuits to study the effects of errors which occur on real devices.

Note, that these automatic models are only an approximation of the real errors that occur on actual devices, due to the fact that they must be build from a limited set of input parameters related to average error rates on gates. The study of quantum errors on real devices is an active area of research and we discuss the Qiskit Aer tools for configuring more detailed noise models in another notebook.

[1]:
```
from qiskit import QuantumCircuit, transpile
from qiskit_aer import AerSimulator
from qiskit.visualization import plot_histogram
```

0.2 Device Backend Noise Model

The Qiskit Aer device noise model automatically generates a simplified noise model for a real device. This model is generated using the calibration information reported in the properties of a device and takes into account

- The gate_error probability of each basis gate on each qubit.
- The gate_length of each basis gate on each qubit.
- The relaxation time constants of each qubit.
- The readout error probability of each qubit.

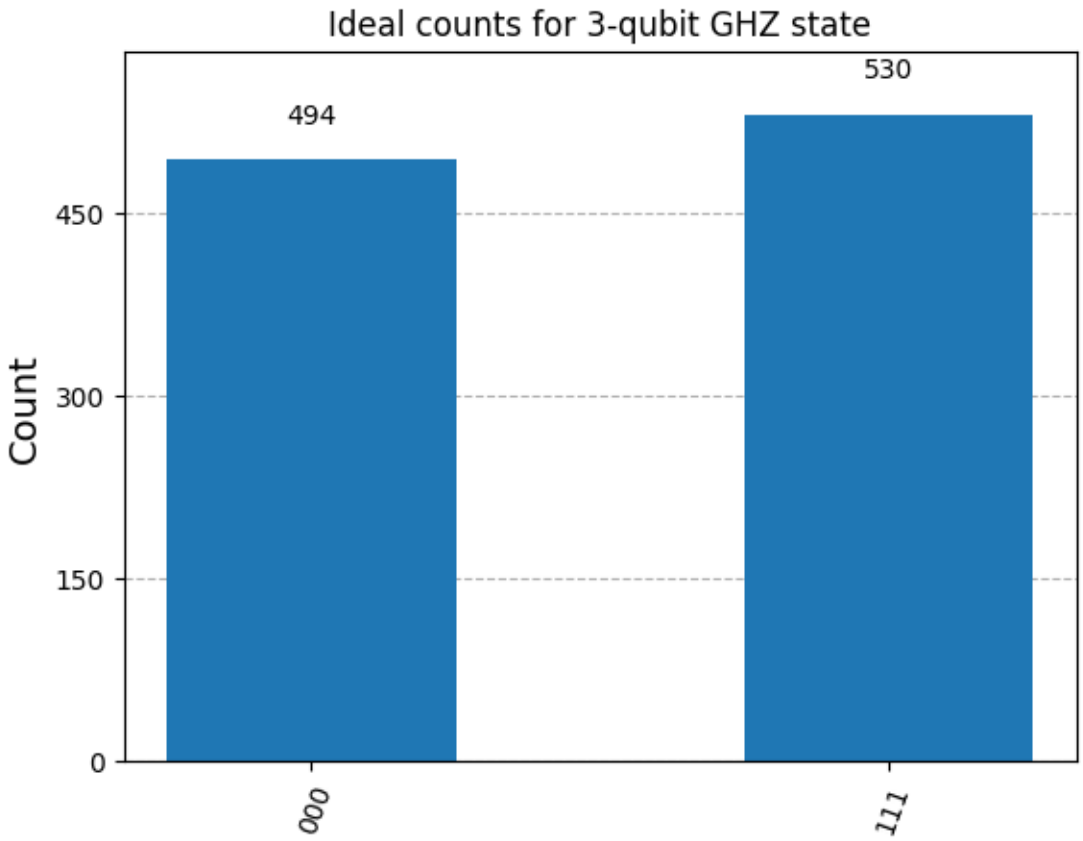
0.2.1 Fake Provider Backends

We will use real noise data for an IBM Quantum device using the data stored in Qiskit Terra.

[2]:
```
from qiskit_ibm_runtime.fake_provider import FakeVigoV2
device_backend = FakeVigoV2()
```

[3]:
```
# Construct quantum circuit
circ = QuantumCircuit(3, 3)
circ.h(0)
circ.cx(0, 1)
circ.cx(1, 2)
circ.measure([0, 1, 2], [0, 1, 2])
```

[3]:
```
sim_ideal = AerSimulator()

# Execute and get counts
result = sim_ideal.run(transpile(circ, sim_ideal)).result()
counts = result.get_counts(0)
plot_histogram(counts, title='Ideal counts for 3-qubit GHZ state')
```



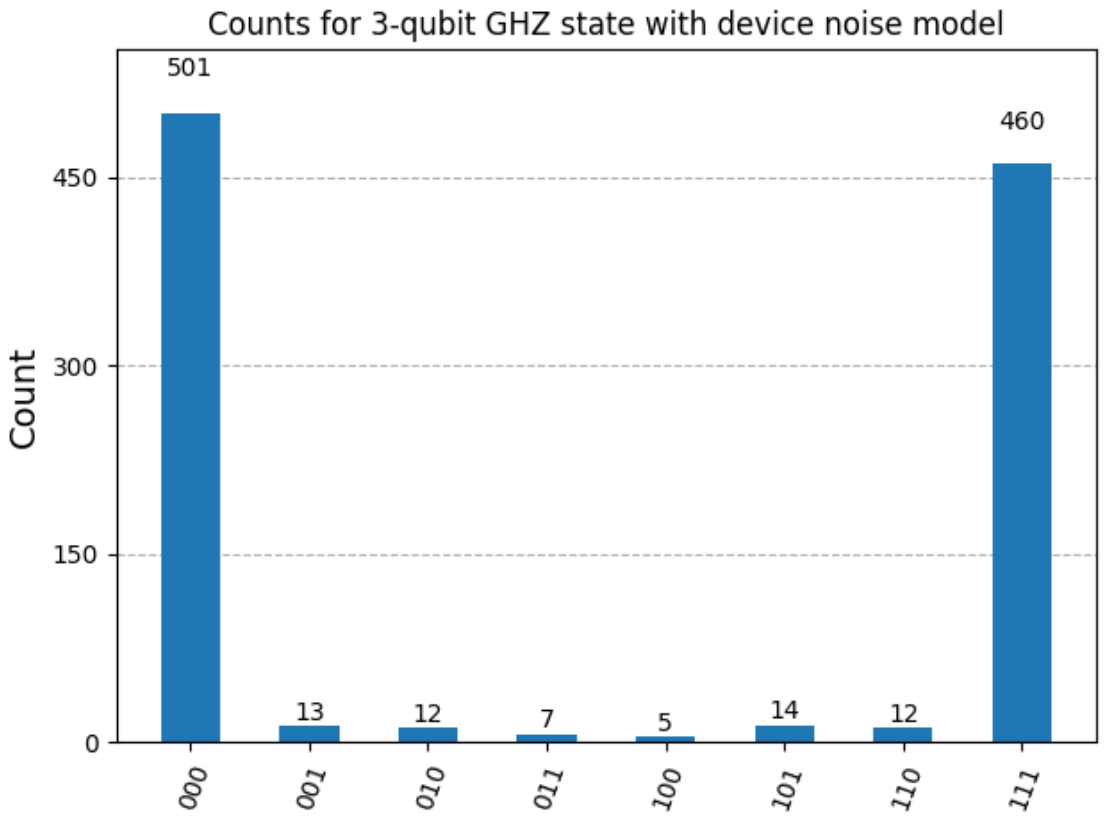
[5]:
```
sim_vigo = AerSimulator.from_backend(device_backend)
```

By storing the device properties in vigo_simulator, we ensure that the appropriate basis gates and coupling map are used when compiling circuits for simulation, thereby most closely mimicking the gates that will be executed on a real device. In addition vigo_simulator contains an approximate noise model consisting of:

- Single-qubit gate errors consisting of a single qubit depolarizing error followed by a single qubit thermal relaxation error.
- Two-qubit gate errors consisting of a two-qubit depolarizing error followed by single-qubit thermal relaxation errors on both qubits in the gate.
- Single-qubit readout errors on the classical bit value obtained from measurements on individual qubits.

[ ]:
```
# Transpile the circuit for the noisy basis gates
tcirc = transpile(circ, sim_vigo)

# Execute noisy simulation and get counts
result_noise = sim_vigo.run(tcirc).result()
counts_noise = result_noise.get_counts(0)
plot_histogram(counts_noise,
               title="Counts for 3-qubit GHZ state with device noise model")
```

[ ]:



[ ]:
```
tcirc.draw()
```

[ ]:
```
global phase:  /4

    q_1 -> 1                  X       M

    q_0 -> 2   Rz( /2)   √X   Rz( /2)       M

    q_2 -> 3                     X      M

        c: 3/                                          0   1   2
```

[ ]:
```
device_backend.instructions
```

[ ]:
```
[(Instruction(name='id', num_qubits=1, num_clbits=0, params=[]), (0,)),
 (Instruction(name='id', num_qubits=1, num_clbits=0, params=[]), (1,)),
 (Instruction(name='id', num_qubits=1, num_clbits=0, params=[]), (2,)),
 (Instruction(name='id', num_qubits=1, num_clbits=0, params=[]), (3,)),
 (Instruction(name='id', num_qubits=1, num_clbits=0, params=[]), (4,)),
 (Delay(duration=t[unit=dt]), (0,)),
 (Delay(duration=t[unit=dt]), (1,)),
 (Delay(duration=t[unit=dt]), (2,)),
 (Delay(duration=t[unit=dt]), (3,)),
 (Delay(duration=t[unit=dt]), (4,)),
 (Instruction(name='reset', num_qubits=1, num_clbits=0, params=[]), (0,)),
 (Instruction(name='reset', num_qubits=1, num_clbits=0, params=[]), (1,)),
 (Instruction(name='reset', num_qubits=1, num_clbits=0, params=[]), (2,)),
 (Instruction(name='reset', num_qubits=1, num_clbits=0, params=[]), (3,)),
 (Instruction(name='reset', num_qubits=1, num_clbits=0, params=[]), (4,)),
 (Instruction(name='rz', num_qubits=1, num_clbits=0, params=[Parameter( )]),
  (0,)),
 (Instruction(name='rz', num_qubits=1, num_clbits=0, params=[Parameter( )]),
  (1,)),
 (Instruction(name='rz', num_qubits=1, num_clbits=0, params=[Parameter( )]),
  (2,)),
 (Instruction(name='rz', num_qubits=1, num_clbits=0, params=[Parameter( )]),
  (3,)),
 (Instruction(name='rz', num_qubits=1, num_clbits=0, params=[Parameter( )]),
  (4,)),
 (Instruction(name='cx', num_qubits=2, num_clbits=0, params=[]), (0, 1)),
 (Instruction(name='cx', num_qubits=2, num_clbits=0, params=[]), (1, 0)),
 (Instruction(name='cx', num_qubits=2, num_clbits=0, params=[]), (1, 2)),
 (Instruction(name='cx', num_qubits=2, num_clbits=0, params=[]), (1, 3)),
 (Instruction(name='cx', num_qubits=2, num_clbits=0, params=[]), (2, 1)),
 (Instruction(name='cx', num_qubits=2, num_clbits=0, params=[]), (3, 1)),
 (Instruction(name='cx', num_qubits=2, num_clbits=0, params=[]), (3, 4)),
 (Instruction(name='cx', num_qubits=2, num_clbits=0, params=[]), (4, 3)),
 (Instruction(name='x', num_qubits=1, num_clbits=0, params=[]), (0,)),
 (Instruction(name='x', num_qubits=1, num_clbits=0, params=[]), (1,)),
 (Instruction(name='x', num_qubits=1, num_clbits=0, params=[]), (2,)),
 (Instruction(name='x', num_qubits=1, num_clbits=0, params=[]), (3,)),
 (Instruction(name='x', num_qubits=1, num_clbits=0, params=[]), (4,)),
 (Instruction(name='sx', num_qubits=1, num_clbits=0, params=[]), (0,)),
 (Instruction(name='sx', num_qubits=1, num_clbits=0, params=[]), (1,)),
 (Instruction(name='sx', num_qubits=1, num_clbits=0, params=[]), (2,)),
 (Instruction(name='sx', num_qubits=1, num_clbits=0, params=[]), (3,)),
 (Instruction(name='sx', num_qubits=1, num_clbits=0, params=[]), (4,)),
 (Instruction(name='measure', num_qubits=1, num_clbits=1, params=[]), (0,)),
 (Instruction(name='measure', num_qubits=1, num_clbits=1, params=[]), (1,)),
```

```
 (Instruction(name='measure', num_qubits=1, num_clbits=1, params=[]), (2,)),
 (Instruction(name='measure', num_qubits=1, num_clbits=1, params=[]), (3,)),
 (Instruction(name='measure', num_qubits=1, num_clbits=1, params=[]), (4,))]
```

[ ]:

50-mthree-basic

May 26, 2026

[1]:
```
!pip install mthree
```

```
Requirement already satisfied: mthree in /home/lpawela/anaconda3/envs/parrondo-
old/lib/python3.12/site-packages (3.0.0)
Requirement already satisfied: numpy>=1.23 in
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (from
mthree) (2.2.5)
Requirement already satisfied: scipy>=1.11.1 in
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (from
mthree) (1.15.2)
Requirement already satisfied: cython>=3.0.10 in
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (from
mthree) (3.1.1)
Requirement already satisfied: qiskit>=1.0 in
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (from
mthree) (1.4.2)
Requirement already satisfied: psutil in /home/lpawela/anaconda3/envs/parrondo-
old/lib/python3.12/site-packages (from mthree) (7.0.0)
Requirement already satisfied: orjson>=3.0.0 in
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (from
mthree) (3.10.18)
Requirement already satisfied: runningman>=2.1 in
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (from
mthree) (2.3.0)
Requirement already satisfied: rustworkx>=0.15.0 in
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (from
qiskit>=1.0->mthree) (0.16.0)
Requirement already satisfied: sympy>=1.3 in
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (from
qiskit>=1.0->mthree) (1.13.3)
Requirement already satisfied: dill>=0.3 in
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (from
qiskit>=1.0->mthree) (0.4.0)
Requirement already satisfied: python-dateutil>=2.8.0 in
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (from
qiskit>=1.0->mthree) (2.9.0.post0)
Requirement already satisfied: stevedore>=3.0.0 in
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (from
```

```
qiskit>=1.0->mthree) (5.4.1)
Requirement already satisfied: typing-extensions in
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (from
qiskit>=1.0->mthree) (4.13.2)
Requirement already satisfied: symengine<0.14,>=0.11 in
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (from
qiskit>=1.0->mthree) (0.13.0)
Requirement already satisfied: qiskit_ibm_runtime>=0.22 in
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (from
runningman>=2.1->mthree) (0.38.0)
Requirement already satisfied: six>=1.5 in
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (from
python-dateutil>=2.8.0->qiskit>=1.0->mthree) (1.17.0)
Requirement already satisfied: requests>=2.19 in
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (from
qiskit_ibm_runtime>=0.22->runningman>=2.1->mthree) (2.32.3)
Requirement already satisfied: requests-ntlm>=1.1.0 in
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (from
qiskit_ibm_runtime>=0.22->runningman>=2.1->mthree) (1.3.0)
Requirement already satisfied: urllib3>=1.21.1 in
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (from
qiskit_ibm_runtime>=0.22->runningman>=2.1->mthree) (2.4.0)
Requirement already satisfied: ibm-platform-services>=0.22.6 in
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (from
qiskit_ibm_runtime>=0.22->runningman>=2.1->mthree) (0.63.0)
Requirement already satisfied: pydantic<2.10,>=2.5.0 in
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (from
qiskit_ibm_runtime>=0.22->runningman>=2.1->mthree) (2.9.2)
Requirement already satisfied: packaging in
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (from
qiskit_ibm_runtime>=0.22->runningman>=2.1->mthree) (25.0)
Requirement already satisfied: pbr>=2.0.0 in
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (from
stevedore>=3.0.0->qiskit>=1.0->mthree) (6.1.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (from
sympy>=1.3->qiskit>=1.0->mthree) (1.3.0)
Requirement already satisfied: ibm_cloud_sdk_core<4.0.0,>=3.22.1 in
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (from
ibm-platform-
services>=0.22.6->qiskit_ibm_runtime>=0.22->runningman>=2.1->mthree) (3.23.0)
Requirement already satisfied: setuptools in
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (from
pbr>=2.0.0->stevedore>=3.0.0->qiskit>=1.0->mthree) (75.8.0)
Requirement already satisfied: annotated-types>=0.6.0 in
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (from
pydantic<2.10,>=2.5.0->qiskit_ibm_runtime>=0.22->runningman>=2.1->mthree)
(0.7.0)
```

```
Requirement already satisfied: pydantic-core==2.23.4 in
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (from
pydantic<2.10,>=2.5.0->qiskit_ibm_runtime>=0.22->runningman>=2.1->mthree)
(2.23.4)
Requirement already satisfied: charset-normalizer<4,>=2 in
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (from
requests>=2.19->qiskit_ibm_runtime>=0.22->runningman>=2.1->mthree) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (from
requests>=2.19->qiskit_ibm_runtime>=0.22->runningman>=2.1->mthree) (3.10)
Requirement already satisfied: certifi>=2017.4.17 in
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (from
requests>=2.19->qiskit_ibm_runtime>=0.22->runningman>=2.1->mthree) (2025.1.31)
Requirement already satisfied: cryptography>=1.3 in
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (from
requests-ntlm>=1.1.0->qiskit_ibm_runtime>=0.22->runningman>=2.1->mthree)
(44.0.2)
Requirement already satisfied: pyspnego>=0.4.0 in
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (from
requests-ntlm>=1.1.0->qiskit_ibm_runtime>=0.22->runningman>=2.1->mthree)
(0.11.2)
Requirement already satisfied: cffi>=1.12 in
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (from
cryptography>=1.3->requests-
ntlm>=1.1.0->qiskit_ibm_runtime>=0.22->runningman>=2.1->mthree) (1.17.1)
Requirement already satisfied: PyJWT<3.0.0,>=2.8.0 in
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (from
ibm_cloud_sdk_core<4.0.0,>=3.22.1->ibm-platform-
services>=0.22.6->qiskit_ibm_runtime>=0.22->runningman>=2.1->mthree) (2.10.1)
Requirement already satisfied: pycparser in
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (from
cffi>=1.12->cryptography>=1.3->requests-
ntlm>=1.1.0->qiskit_ibm_runtime>=0.22->runningman>=2.1->mthree) (2.22)
```

[2]:
```
!pip freeze
```

```
annotated-types==0.7.0
asttokens==3.0.0
black==25.1.0
certifi==2025.1.31
cffi==1.17.1
charset-normalizer==3.4.1
click==8.1.8
comm==0.2.2
contourpy==1.3.2
cryptography==44.0.2
cudensitymat-cu12==0.1.0
cuquantum-cu12==25.3.0
```

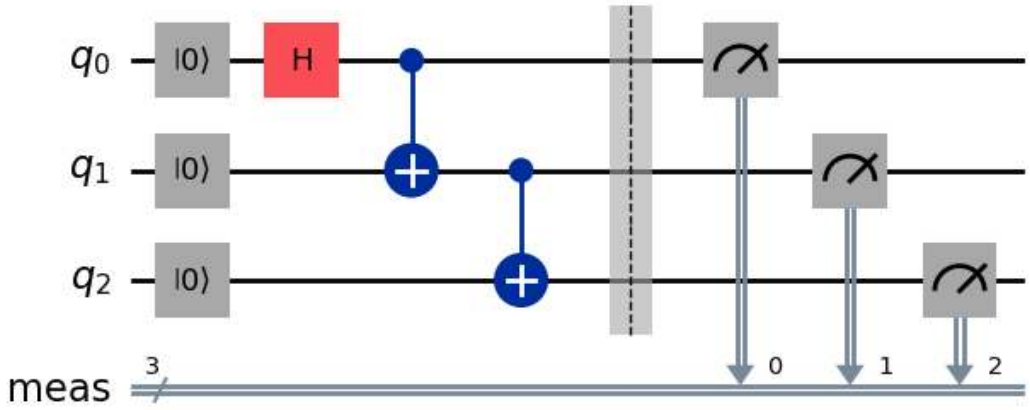
```
custatevec-cu12==1.8.0
cutensor-cu12==2.2.0
cutensornet-cu12==2.7.0
cycler==0.12.1
Cython==3.1.1
debugpy==1.8.14
decorator==5.2.1
dill==0.4.0
dotenv==0.9.9
executing==2.2.0
fonttools==4.57.0
ibm-cloud-sdk-core==3.23.0
ibm-platform-services==0.63.0
idna==3.10
ipykernel==6.29.5
ipython==9.1.0
ipython_pygments_lexers==1.1.1
isort==6.0.1
jedi==0.19.2
joblib==1.5.0
jupyter_client==8.6.3
jupyter_core==5.7.2
kiwisolver==1.4.8
matplotlib==3.10.1
matplotlib-inline==0.1.7
mpmath==1.3.0
mthree==3.0.0
mypy_extensions==1.1.0
nest-asyncio==1.6.0
numpy==2.2.5
nvidia-cublas-cu12==12.8.4.1
nvidia-cuda-runtime-cu12==12.8.90
nvidia-cusolver-cu12==11.7.3.90
nvidia-cusparse-cu12==12.5.8.93
nvidia-nvjitlink-cu12==12.8.93
orjson==3.10.18
packaging==25.0
parso==0.8.4
pathspec==0.12.1
pbr==6.1.1
pexpect==4.9.0
pillow==11.2.1
platformdirs==4.3.7
prompt_toolkit==3.0.51
psutil==7.0.0
ptyprocess==0.7.0
pure_eval==0.2.3
pycparser==2.22
```

```
pydantic==2.9.2
pydantic_core==2.23.4
Pygments==2.19.1
PyJWT==2.10.1
pylatexenc==2.10
pyparsing==3.2.3
pyspnego==0.11.2
python-dateutil==2.9.0.post0
python-dotenv==1.1.0
pyzmq==26.4.0
qiskit==1.4.2
qiskit-aer==0.16.0
qiskit-aer-gpu==0.15.1
qiskit-ibm-runtime==0.38.0
qiskit-machine-learning==0.8.2
requests==2.32.3
requests_ntlm==1.3.0
runningman==2.3.0
rustworkx==0.16.0
scikit-learn==1.6.1
scipy==1.15.2
setuptools==75.8.0
six==1.17.0
stack-data==0.6.3
stevedore==5.4.1
symengine==0.13.0
sympy==1.13.3
threadpoolctl==3.6.0
tornado==6.4.2
tqdm==4.67.1
traitlets==5.14.3
typing_extensions==4.13.2
urllib3==2.4.0
wcwidth==0.2.13
wheel==0.45.1
```

[3]:
```python
import numpy as np
from qiskit import *
from qiskit_ibm_runtime.fake_provider import FakeVigoV2
import mthree

qc = QuantumCircuit(3)
qc.reset(range(3))
qc.h(0)
qc.cx(0,1)
qc.cx(1,2)
qc.measure_all()
```

```
qc.draw('mpl')
```

[3]:



[4]:
```
backend = FakeVigoV2()
mit = mthree.M3Mitigation(backend)
mit.cals_from_system(range(3))
```

[4]:
```
[<qiskit_aer.jobs.aerjob.AerJob at 0x77339f3f51c0>]
```

[5]:
```
trans_qc = transpile(qc, backend)
raw_counts = backend.run(trans_qc, shots=8192).result().get_counts()
```

[6]:
```
raw_counts
```

[6]:
```
{'001': 39,
 '110': 117,
 '101': 145,
 '100': 79,
 '011': 121,
 '111': 3665,
 '010': 52,
 '000': 3974}
```

[7]:
```
prob_counts = {k: v/sum(raw_counts.values()) for k,v in raw_counts.items()}
prob_counts
```

[7]:
```
{'100': 0.009521484375,
 '110': 0.0113525390625,
 '001': 0.0052490234375,
 '010': 0.0054931640625,
 '000': 0.4776611328125,
```

```
 '011': 0.0164794921875,
 '111': 0.457275390625,
 '101': 0.0169677734375}
```

[7]:
```
quasis = mit.apply_correction(raw_counts, range(3))
print('Expectation value:',quasis.expval("ZZZ"))
```

```
Expectation value: 0.034488022327423096
```

[8]:
```
quasis
```

[8]:
```
{'000': np.float32(0.5340166),
 '001': np.float32(-0.03803007),
 '010': np.float32(0.0021706226),
 '011': np.float32(0.0038574233),
 '100': np.float32(0.006740882),
 '101': np.float32(0.00052801555),
 '110': np.float32(-0.021158068),
 '111': np.float32(0.5118746)}
```

[10]:
```
all_zeros_proj = {'000': 1}
all_ones_proj = {'111': 1}
quasis.expval([all_zeros_proj, all_ones_proj])
```

[10]:
```
array([0.5295377 , 0.52403015], dtype=float32)
```

[9]:
```
sum(quasis.values())
```

[9]:
```
np.float32(1.0)
```

[10]:
```
quasis.nearest_probability_distribution()
```

[10]:
```
{'111': np.float32(0.48892903), '000': np.float32(0.511071)}
```

[ ]:
```

```

60-perceptron

May 26, 2026

[26]:
```
!pip install tqdm
```

```
Requirement already satisfied: tqdm in /home/lpawela/miniconda3/envs/parrondo-
old/lib/python3.12/site-packages (4.67.1)
```

[27]:
```
import numpy as np
from qiskit_aer import AerSimulator
from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister, transpile
from qiskit.visualization import plot_histogram
from itertools import product
from tqdm import tqdm
```

1 Definition of the sign flip

$$\mathrm{SF}_{N,j} = O_j(C^N Z)O_j$$

$$O_j = \otimes_{l=0}^{N-1}(\mathrm{NOT}_l)^{1-j_l}$$

Choose N and j. j represents the elements of the computational basis.

[28]:
```
def num2bin(num, n):
    return np.array([int(i) for i in np.binary_repr(num, n)])

def O(q, j):
    O_circ = QuantumCircuit(q, name=f"O_{''.join([str(i) for i in j])}")

    for (i, v) in enumerate(j):
        if v == 0:
            O_circ.x(q[i])
            O_circ.barrier()
    return  O_circ

def sf(q, j):
    N = q.size
    SF = QuantumCircuit(q, name=f"SF_{q.size}_{''.join([str(i) for i in j])}")
    SF.append(O(q, j), q)
    SF.barrier(q)
```

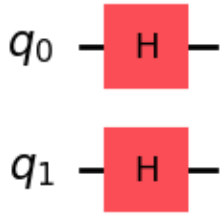
```
    # SF.mcrz(np.pi, q[:N-1], q[N-1])
    SF.mcp(np.pi, q[:N-1], q[N-1])
    SF.barrier(q)
    SF.append(O(q, j), q)
    return SF
```

[29]:
```
def U_i(q, input_vector):
    u_i = QuantumCircuit(q, name=f"U^i_{''.join([str(i) for i in␣
 ↪input_vector])}")
    N = q.size
    u_i.h(q)
    for (i, v) in enumerate(input_vector):
        if v:
            bin_num = num2bin(i, N)
            u_i.append(sf(q, bin_num), q)
            u_i.barrier()
    return u_i
```

[30]:
```
def U_w(q, weight_vector):
    u_w = QuantumCircuit(q, name=f"U^w_{''.join([str(i) for i in␣
 ↪weight_vector])}")
    N = q.size
    for (i, v) in enumerate(weight_vector):
        if v:
            bin_num = num2bin(i, N)
            u_w.append(sf(q, bin_num), q)
            u_w.barrier()
    u_w.h(q)
    u_w.x(q)
    return u_w
```

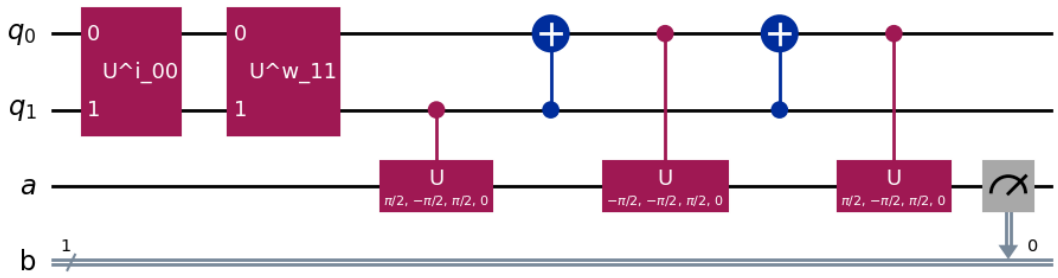
[31]:
```
N = 2
q = QuantumRegister(N, 'q')
input_vector = [0, 0, 0, 0]
circ = QuantumCircuit(q)
circ.append(U_i(q, input_vector), q)
circ.decompose(reps=1).draw(output='mpl')
```

[31]:



[33]:
```
N = 2
q = QuantumRegister(N, 'q')
a = QuantumRegister(1, 'a')
b = ClassicalRegister(1, 'b')
circ = QuantumCircuit(q, a, b)
circ.append(U_i(q, num2bin(0, N)[::-1]), q)
circ.append(U_w(q, num2bin(3, N)), q)
circ.mcrx(np.pi, q, a[0])
circ.measure(a, b)
circ.decompose(reps=0).draw(output='mpl')
```

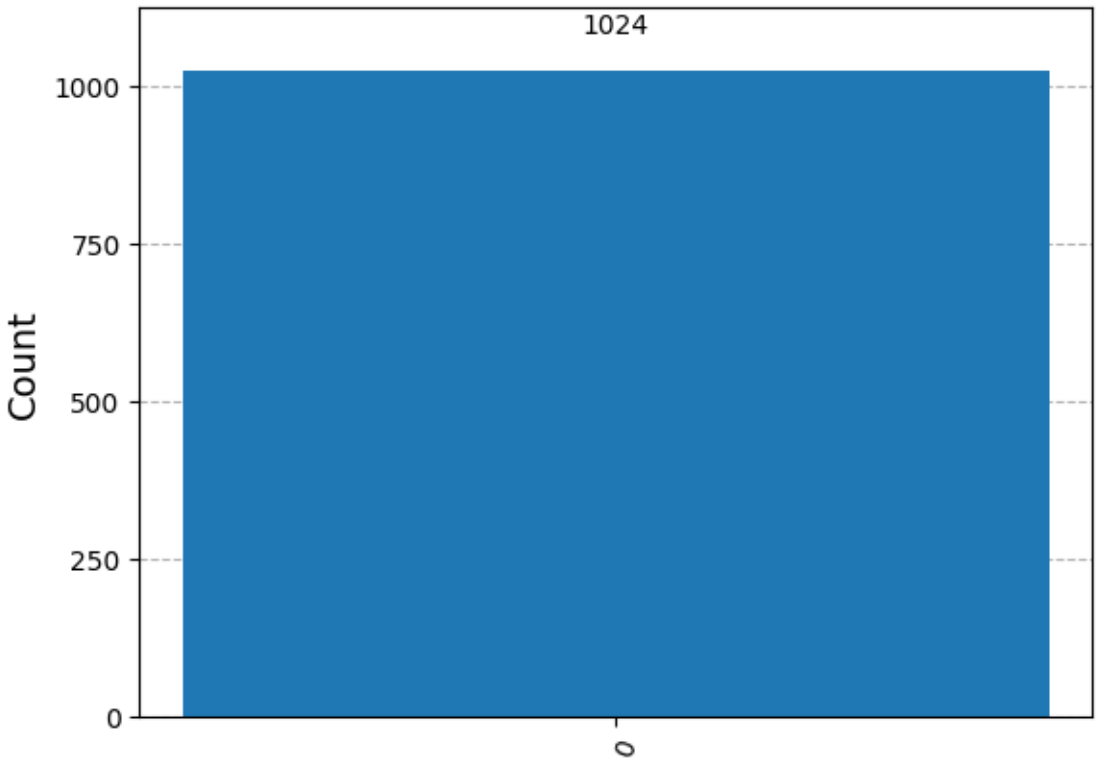
[33]:



[34]:
```
simulator = AerSimulator()
job = simulator.run(transpile(circ, simulator), shots=1024)
result = job.result()
counts = result.get_counts(circ)
print("\nTotal counts are:",counts)
plot_histogram(counts)
```

```
Total counts are: {'0': 1024}
```

[34]:



2 Input vs weights matrix

Below we compare all the input vectors against all the weight vectors, for a specified N. We're computing a square matrix with a side of 2^{2^N} elements.

Start with $N = 2$ to see results within your lifetime.

The results are saved in **outcomes**, and can be viewed in an image in the cell below.

[35]:
```
N = 2
shots = 1024
outcomes = np.zeros((2**(2**N),2**(2**N)))
for (i, w) in tqdm(product(range(2**(2**N)), repeat=2)):
    q = QuantumRegister(N, 'q')
    a = QuantumRegister(1, 'a')
    b = ClassicalRegister(1, 'b')
    circ = QuantumCircuit(q, a, b)
    x = num2bin(i, 2**N)
    ww = num2bin(w, 2**N)
    circ.append(U_i(q, x), q)
    circ.append(U_w(q, ww), q)
    circ.mcrx(np.pi, q, a[0])
    circ.measure(a, b)
```

```
    simulator = AerSimulator()
    job = simulator.run(transpile(circ, simulator), shots=shots)
    result = job.result()
    counts = result.get_counts(circ)
    outcomes[i][w] = counts.get('1', 0) / shots
    print(i,x,  w, ww, counts)
```

```
3it [00:00, 21.44it/s]

0 [0 0 0 0] 0 [0 0 0 0] {'1': 1024}
0 [0 0 0 0] 1 [0 0 0 1] {'1': 263, '0': 761}
0 [0 0 0 0] 2 [0 0 1 0] {'1': 253, '0': 771}
0 [0 0 0 0] 3 [0 0 1 1] {'0': 1024}
0 [0 0 0 0] 4 [0 1 0 0] {'0': 764, '1': 260}

6it [00:00, 20.22it/s]

0 [0 0 0 0] 5 [0 1 0 1] {'0': 1024}
0 [0 0 0 0] 6 [0 1 1 0] {'0': 1024}
0 [0 0 0 0] 7 [0 1 1 1] {'1': 232, '0': 792}

9it [00:00, 19.03it/s]

0 [0 0 0 0] 8 [1 0 0 0] {'1': 253, '0': 771}

11it [00:00, 18.93it/s]

0 [0 0 0 0] 9 [1 0 0 1] {'0': 1024}
0 [0 0 0 0] 10 [1 0 1 0] {'0': 1024}
0 [0 0 0 0] 11 [1 0 1 1] {'1': 248, '0': 776}

13it [00:00, 18.83it/s]

0 [0 0 0 0] 12 [1 1 0 0] {'0': 1024}
0 [0 0 0 0] 13 [1 1 0 1] {'1': 235, '0': 789}

16it [00:00, 19.63it/s]

0 [0 0 0 0] 14 [1 1 1 0] {'1': 245, '0': 779}
0 [0 0 0 0] 15 [1 1 1 1] {'1': 1024}

18it [00:00, 19.38it/s]

1 [0 0 0 1] 0 [0 0 0 0] {'1': 264, '0': 760}
1 [0 0 0 1] 1 [0 0 0 1] {'1': 1024}

20it [00:01, 19.48it/s]

1 [0 0 0 1] 2 [0 0 1 0] {'0': 1024}
1 [0 0 0 1] 3 [0 0 1 1] {'1': 250, '0': 774}
1 [0 0 0 1] 4 [0 1 0 0] {'0': 1024}

23it [00:01, 20.57it/s]
```

```
1 [0 0 0 1] 5 [0 1 0 1] {'1': 255, '0': 769}
1 [0 0 0 1] 6 [0 1 1 0] {'1': 231, '0': 793}

26it [00:01, 20.36it/s]

1 [0 0 0 1] 7 [0 1 1 1] {'0': 1024}
1 [0 0 0 1] 8 [1 0 0 0] {'0': 1024}
1 [0 0 0 1] 9 [1 0 0 1] {'1': 267, '0': 757}
1 [0 0 0 1] 10 [1 0 1 0] {'1': 271, '0': 753}
1 [0 0 0 1] 11 [1 0 1 1] {'0': 1024}

29it [00:01, 20.22it/s]

1 [0 0 0 1] 12 [1 1 0 0] {'1': 264, '0': 760}
1 [0 0 0 1] 13 [1 1 0 1] {'0': 1024}

32it [00:01, 19.19it/s]

1 [0 0 0 1] 14 [1 1 1 0] {'1': 1024}
1 [0 0 0 1] 15 [1 1 1 1] {'0': 766, '1': 258}
2 [0 0 1 0] 0 [0 0 0 0] {'0': 761, '1': 263}
2 [0 0 1 0] 1 [0 0 0 1] {'0': 1024}

35it [00:01, 19.39it/s]

2 [0 0 1 0] 2 [0 0 1 0] {'1': 1024}
2 [0 0 1 0] 3 [0 0 1 1] {'1': 249, '0': 775}
2 [0 0 1 0] 4 [0 1 0 0] {'0': 1024}

38it [00:01, 19.85it/s]

2 [0 0 1 0] 5 [0 1 0 1] {'1': 242, '0': 782}
2 [0 0 1 0] 6 [0 1 1 0] {'1': 256, '0': 768}

40it [00:02, 19.81it/s]

2 [0 0 1 0] 7 [0 1 1 1] {'0': 1024}
2 [0 0 1 0] 8 [1 0 0 0] {'0': 1024}

42it [00:02, 18.55it/s]

2 [0 0 1 0] 9 [1 0 0 1] {'1': 274, '0': 750}
2 [0 0 1 0] 10 [1 0 1 0] {'0': 754, '1': 270}

44it [00:02, 18.01it/s]

2 [0 0 1 0] 11 [1 0 1 1] {'0': 1024}
2 [0 0 1 0] 12 [1 1 0 0] {'1': 257, '0': 767}

46it [00:02, 18.22it/s]

2 [0 0 1 0] 13 [1 1 0 1] {'1': 1024}

48it [00:02, 15.97it/s]

2 [0 0 1 0] 14 [1 1 1 0] {'0': 1024}
2 [0 0 1 0] 15 [1 1 1 1] {'1': 255, '0': 769}
```

```
51it [00:02, 17.73it/s]

3 [0 0 1 1] 0 [0 0 0 0] {'0': 1024}
3 [0 0 1 1] 1 [0 0 0 1] {'1': 245, '0': 779}
3 [0 0 1 1] 2 [0 0 1 0] {'1': 248, '0': 776}

53it [00:02, 18.11it/s]

3 [0 0 1 1] 3 [0 0 1 1] {'1': 1024}
3 [0 0 1 1] 4 [0 1 0 0] {'1': 274, '0': 750}

56it [00:02, 18.86it/s]

3 [0 0 1 1] 5 [0 1 0 1] {'0': 1024}
3 [0 0 1 1] 6 [0 1 1 0] {'0': 1024}
3 [0 0 1 1] 7 [0 1 1 1] {'0': 780, '1': 244}

58it [00:03, 18.91it/s]

3 [0 0 1 1] 8 [1 0 0 0] {'1': 271, '0': 753}
3 [0 0 1 1] 9 [1 0 0 1] {'0': 1024}

60it [00:03, 18.48it/s]

3 [0 0 1 1] 10 [1 0 1 0] {'0': 1024}
3 [0 0 1 1] 11 [1 0 1 1] {'1': 252, '0': 772}

62it [00:03, 18.30it/s]

3 [0 0 1 1] 12 [1 1 0 0] {'1': 1024}
3 [0 0 1 1] 13 [1 1 0 1] {'1': 252, '0': 772}

64it [00:03, 18.71it/s]

3 [0 0 1 1] 14 [1 1 1 0] {'1': 238, '0': 786}
3 [0 0 1 1] 15 [1 1 1 1] {'0': 1024}
4 [0 1 0 0] 0 [0 0 0 0] {'1': 268, '0': 756}

67it [00:03, 19.35it/s]

4 [0 1 0 0] 1 [0 0 0 1] {'0': 1024}
4 [0 1 0 0] 2 [0 0 1 0] {'0': 1024}

69it [00:03, 19.46it/s]

4 [0 1 0 0] 3 [0 0 1 1] {'0': 765, '1': 259}
4 [0 1 0 0] 4 [0 1 0 0] {'1': 1024}
4 [0 1 0 0] 5 [0 1 0 1] {'1': 234, '0': 790}

72it [00:03, 19.95it/s]

4 [0 1 0 0] 6 [0 1 1 0] {'1': 267, '0': 757}
4 [0 1 0 0] 7 [0 1 1 1] {'0': 1024}

75it [00:03, 20.33it/s]

4 [0 1 0 0] 8 [1 0 0 0] {'0': 1024}
4 [0 1 0 0] 9 [1 0 0 1] {'1': 265, '0': 759}
```

```
4 [0 1 0 0] 10 [1 0 1 0] {'1': 254, '0': 770}
4 [0 1 0 0] 11 [1 0 1 1] {'1': 1024}
4 [0 1 0 0] 12 [1 1 0 0] {'1': 248, '0': 776}

78it [00:04, 20.14it/s]

4 [0 1 0 0] 13 [1 1 0 1] {'0': 1024}
4 [0 1 0 0] 14 [1 1 1 0] {'0': 1024}
4 [0 1 0 0] 15 [1 1 1 1] {'1': 258, '0': 766}

81it [00:04, 20.23it/s]

5 [0 1 0 1] 0 [0 0 0 0] {'0': 1024}
5 [0 1 0 1] 1 [0 0 0 1] {'1': 234, '0': 790}

84it [00:04, 19.55it/s]

5 [0 1 0 1] 2 [0 0 1 0] {'1': 263, '0': 761}
5 [0 1 0 1] 3 [0 0 1 1] {'0': 1024}

86it [00:04, 18.50it/s]

5 [0 1 0 1] 4 [0 1 0 0] {'1': 234, '0': 790}
5 [0 1 0 1] 5 [0 1 0 1] {'1': 1024}
5 [0 1 0 1] 6 [0 1 1 0] {'0': 1024}
5 [0 1 0 1] 7 [0 1 1 1] {'1': 274, '0': 750}

89it [00:04, 18.48it/s]

5 [0 1 0 1] 8 [1 0 0 0] {'1': 258, '0': 766}
5 [0 1 0 1] 9 [1 0 0 1] {'0': 1024}

91it [00:04, 18.19it/s]

5 [0 1 0 1] 10 [1 0 1 0] {'1': 1024}
5 [0 1 0 1] 11 [1 0 1 1] {'1': 245, '0': 779}

93it [00:04, 17.74it/s]

5 [0 1 0 1] 12 [1 1 0 0] {'0': 1024}
5 [0 1 0 1] 13 [1 1 0 1] {'1': 262, '0': 762}

95it [00:05, 17.86it/s]

5 [0 1 0 1] 14 [1 1 1 0] {'1': 256, '0': 768}

97it [00:05, 14.83it/s]

5 [0 1 0 1] 15 [1 1 1 1] {'0': 1024}
6 [0 1 1 0] 0 [0 0 0 0] {'0': 1024}

100it [00:05, 16.44it/s]

6 [0 1 1 0] 1 [0 0 0 1] {'1': 242, '0': 782}
6 [0 1 1 0] 2 [0 0 1 0] {'1': 246, '0': 778}
6 [0 1 1 0] 3 [0 0 1 1] {'0': 1024}
6 [0 1 1 0] 4 [0 1 0 0] {'0': 760, '1': 264}
6 [0 1 1 0] 5 [0 1 0 1] {'0': 1024}
```

```
103it [00:05, 17.56it/s]

6 [0 1 1 0] 6 [0 1 1 0] {'1': 1024}
6 [0 1 1 0] 7 [0 1 1 1] {'0': 771, '1': 253}
6 [0 1 1 0] 8 [1 0 0 0] {'1': 248, '0': 776}

106it [00:05, 18.30it/s]

6 [0 1 1 0] 9 [1 0 0 1] {'1': 1024}
6 [0 1 1 0] 10 [1 0 1 0] {'0': 1024}

109it [00:05, 18.81it/s]

6 [0 1 1 0] 11 [1 0 1 1] {'1': 262, '0': 762}
6 [0 1 1 0] 12 [1 1 0 0] {'0': 1024}

111it [00:05, 18.58it/s]

6 [0 1 1 0] 13 [1 1 0 1] {'1': 246, '0': 778}
6 [0 1 1 0] 14 [1 1 1 0] {'1': 274, '0': 750}

113it [00:06, 18.39it/s]

6 [0 1 1 0] 15 [1 1 1 1] {'0': 1024}
7 [0 1 1 1] 0 [0 0 0 0] {'1': 259, '0': 765}
7 [0 1 1 1] 1 [0 0 0 1] {'0': 1024}
7 [0 1 1 1] 2 [0 0 1 0] {'0': 1024}

116it [00:06, 18.72it/s]

7 [0 1 1 1] 3 [0 0 1 1] {'1': 259, '0': 765}
7 [0 1 1 1] 4 [0 1 0 0] {'0': 1024}

118it [00:06, 18.71it/s]

7 [0 1 1 1] 5 [0 1 0 1] {'1': 244, '0': 780}
7 [0 1 1 1] 6 [0 1 1 0] {'0': 757, '1': 267}

120it [00:06, 18.79it/s]

7 [0 1 1 1] 7 [0 1 1 1] {'1': 1024}
7 [0 1 1 1] 8 [1 0 0 0] {'1': 1024}

122it [00:06, 18.63it/s]

7 [0 1 1 1] 9 [1 0 0 1] {'1': 247, '0': 777}
7 [0 1 1 1] 10 [1 0 1 0] {'0': 776, '1': 248}

124it [00:06, 18.55it/s]

7 [0 1 1 1] 11 [1 0 1 1] {'0': 1024}
7 [0 1 1 1] 12 [1 1 0 0] {'0': 753, '1': 271}

126it [00:06, 17.59it/s]

7 [0 1 1 1] 13 [1 1 0 1] {'0': 1024}
7 [0 1 1 1] 14 [1 1 1 0] {'0': 1024}

128it [00:06, 17.99it/s]
```

```
7 [0 1 1 1] 15 [1 1 1 1] {'1': 251, '0': 773}
8 [1 0 0 0] 0 [0 0 0 0] {'1': 242, '0': 782}

130it [00:06, 17.92it/s]

8 [1 0 0 0] 1 [0 0 0 1] {'0': 1024}
8 [1 0 0 0] 2 [0 0 1 0] {'0': 1024}

132it [00:07, 18.12it/s]

8 [1 0 0 0] 3 [0 0 1 1] {'1': 261, '0': 763}
8 [1 0 0 0] 4 [0 1 0 0] {'0': 1024}

134it [00:07, 17.57it/s]

8 [1 0 0 0] 5 [0 1 0 1] {'1': 265, '0': 759}
8 [1 0 0 0] 6 [0 1 1 0] {'0': 773, '1': 251}

136it [00:07, 16.95it/s]

8 [1 0 0 0] 7 [0 1 1 1] {'1': 1024}
8 [1 0 0 0] 8 [1 0 0 0] {'1': 1024}

138it [00:07, 17.31it/s]

8 [1 0 0 0] 9 [1 0 0 1] {'1': 282, '0': 742}
8 [1 0 0 0] 10 [1 0 1 0] {'1': 284, '0': 740}

140it [00:07, 17.70it/s]

8 [1 0 0 0] 11 [1 0 1 1] {'0': 1024}
8 [1 0 0 0] 12 [1 1 0 0] {'1': 253, '0': 771}
8 [1 0 0 0] 13 [1 1 0 1] {'0': 1024}

143it [00:07, 18.92it/s]

8 [1 0 0 0] 14 [1 1 1 0] {'0': 1024}

145it [00:07, 15.31it/s]

8 [1 0 0 0] 15 [1 1 1 1] {'0': 777, '1': 247}
9 [1 0 0 1] 0 [0 0 0 0] {'0': 1024}

148it [00:08, 16.69it/s]

9 [1 0 0 1] 1 [0 0 0 1] {'0': 760, '1': 264}
9 [1 0 0 1] 2 [0 0 1 0] {'1': 253, '0': 771}
9 [1 0 0 1] 3 [0 0 1 1] {'0': 1024}

150it [00:08, 17.21it/s]

9 [1 0 0 1] 4 [0 1 0 0] {'1': 264, '0': 760}
9 [1 0 0 1] 5 [0 1 0 1] {'0': 1024}

152it [00:08, 17.81it/s]

9 [1 0 0 1] 6 [0 1 1 0] {'1': 1024}
9 [1 0 0 1] 7 [0 1 1 1] {'1': 254, '0': 770}
9 [1 0 0 1] 8 [1 0 0 0] {'1': 270, '0': 754}
```

```
155it [00:08, 18.54it/s]

9 [1 0 0 1] 9 [1 0 0 1] {'1': 1024}
9 [1 0 0 1] 10 [1 0 1 0] {'0': 1024}

157it [00:08, 17.85it/s]

9 [1 0 0 1] 11 [1 0 1 1] {'1': 247, '0': 777}
9 [1 0 0 1] 12 [1 1 0 0] {'0': 1024}

159it [00:08, 18.29it/s]

9 [1 0 0 1] 13 [1 1 0 1] {'1': 250, '0': 774}
9 [1 0 0 1] 14 [1 1 1 0] {'0': 753, '1': 271}

161it [00:08, 18.55it/s]

9 [1 0 0 1] 15 [1 1 1 1] {'0': 1024}
10 [1 0 1 0] 0 [0 0 0 0] {'0': 1024}

163it [00:08, 17.87it/s]

10 [1 0 1 0] 1 [0 0 0 1] {'1': 287, '0': 737}
10 [1 0 1 0] 2 [0 0 1 0] {'1': 259, '0': 765}

165it [00:08, 17.96it/s]

10 [1 0 1 0] 3 [0 0 1 1] {'0': 1024}
10 [1 0 1 0] 4 [0 1 0 0] {'1': 254, '0': 770}

167it [00:09, 17.93it/s]

10 [1 0 1 0] 5 [0 1 0 1] {'1': 1024}
10 [1 0 1 0] 6 [0 1 1 0] {'0': 1024}

169it [00:09, 17.32it/s]

10 [1 0 1 0] 7 [0 1 1 1] {'1': 265, '0': 759}
10 [1 0 1 0] 8 [1 0 0 0] {'0': 765, '1': 259}
10 [1 0 1 0] 9 [1 0 0 1] {'0': 1024}

171it [00:09, 15.23it/s]

10 [1 0 1 0] 10 [1 0 1 0] {'1': 1024}
10 [1 0 1 0] 11 [1 0 1 1] {'1': 251, '0': 773}

173it [00:09, 15.19it/s]

10 [1 0 1 0] 12 [1 1 0 0] {'0': 1024}
10 [1 0 1 0] 13 [1 1 0 1] {'1': 257, '0': 767}

175it [00:09, 15.00it/s]

10 [1 0 1 0] 14 [1 1 1 0] {'1': 261, '0': 763}
10 [1 0 1 0] 15 [1 1 1 1] {'0': 1024}

177it [00:09, 16.13it/s]
```

```
11 [1 0 1 1] 0 [0 0 0 0] {'1': 247, '0': 777}
11 [1 0 1 1] 1 [0 0 0 1] {'0': 1024}

179it [00:09, 16.39it/s]

11 [1 0 1 1] 2 [0 0 1 0] {'0': 1024}
11 [1 0 1 1] 3 [0 0 1 1] {'0': 766, '1': 258}

181it [00:09, 15.77it/s]

11 [1 0 1 1] 4 [0 1 0 0] {'1': 1024}
11 [1 0 1 1] 5 [0 1 0 1] {'1': 258, '0': 766}

183it [00:10, 15.95it/s]

11 [1 0 1 1] 6 [0 1 1 0] {'1': 259, '0': 765}
11 [1 0 1 1] 7 [0 1 1 1] {'0': 1024}

185it [00:10, 16.65it/s]

11 [1 0 1 1] 8 [1 0 0 0] {'0': 1024}
11 [1 0 1 1] 9 [1 0 0 1] {'1': 265, '0': 759}

187it [00:10, 16.52it/s]

11 [1 0 1 1] 10 [1 0 1 0] {'1': 263, '0': 761}
11 [1 0 1 1] 11 [1 0 1 1] {'1': 1024}

189it [00:10, 17.40it/s]

11 [1 0 1 1] 12 [1 1 0 0] {'1': 256, '0': 768}
11 [1 0 1 1] 13 [1 1 0 1] {'0': 1024}

191it [00:10, 16.68it/s]

11 [1 0 1 1] 14 [1 1 1 0] {'0': 1024}
11 [1 0 1 1] 15 [1 1 1 1] {'1': 256, '0': 768}
12 [1 1 0 0] 0 [0 0 0 0] {'0': 1024}

196it [00:10, 15.77it/s]

12 [1 1 0 0] 1 [0 0 0 1] {'1': 267, '0': 757}
12 [1 1 0 0] 2 [0 0 1 0] {'1': 248, '0': 776}
12 [1 1 0 0] 3 [0 0 1 1] {'1': 1024}

198it [00:11, 16.65it/s]

12 [1 1 0 0] 4 [0 1 0 0] {'1': 275, '0': 749}
12 [1 1 0 0] 5 [0 1 0 1] {'0': 1024}

200it [00:11, 15.95it/s]

12 [1 1 0 0] 6 [0 1 1 0] {'0': 1024}
12 [1 1 0 0] 7 [0 1 1 1] {'1': 237, '0': 787}

202it [00:11, 15.85it/s]

12 [1 1 0 0] 8 [1 0 0 0] {'1': 268, '0': 756}
12 [1 1 0 0] 9 [1 0 0 1] {'0': 1024}
```

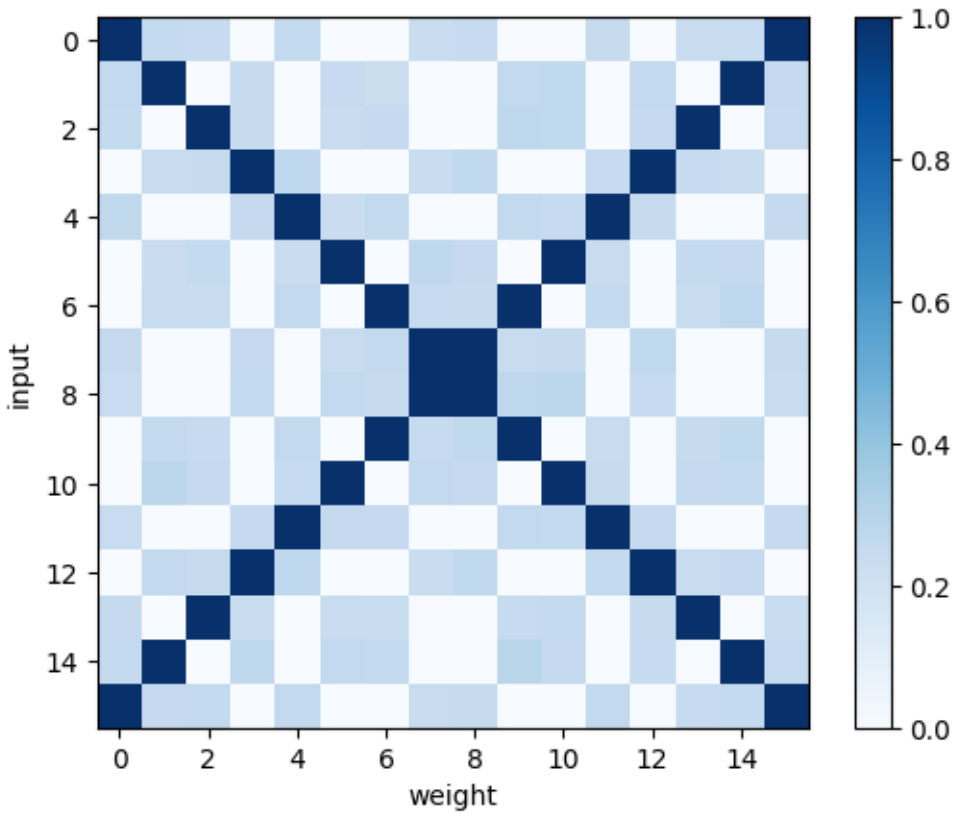
```
204it [00:11, 16.31it/s]

12 [1 1 0 0] 10 [1 0 1 0] {'0': 1024}
12 [1 1 0 0] 11 [1 0 1 1] {'1': 263, '0': 761}

206it [00:11, 15.84it/s]

12 [1 1 0 0] 12 [1 1 0 0] {'1': 1024}
12 [1 1 0 0] 13 [1 1 0 1] {'1': 244, '0': 780}

208it [00:11, 15.46it/s]

12 [1 1 0 0] 14 [1 1 1 0] {'1': 258, '0': 766}
12 [1 1 0 0] 15 [1 1 1 1] {'0': 1024}
13 [1 1 0 1] 0 [0 0 0 0] {'1': 259, '0': 765}
13 [1 1 0 1] 1 [0 0 0 1] {'0': 1024}

211it [00:11, 17.02it/s]

13 [1 1 0 1] 2 [0 0 1 0] {'1': 1024}
13 [1 1 0 1] 3 [0 0 1 1] {'1': 238, '0': 786}

213it [00:11, 17.26it/s]

13 [1 1 0 1] 4 [0 1 0 0] {'0': 1024}
13 [1 1 0 1] 5 [0 1 0 1] {'1': 247, '0': 777}

215it [00:12, 16.82it/s]

13 [1 1 0 1] 6 [0 1 1 0] {'1': 237, '0': 787}
13 [1 1 0 1] 7 [0 1 1 1] {'0': 1024}

217it [00:12, 17.28it/s]

13 [1 1 0 1] 8 [1 0 0 0] {'0': 1024}
13 [1 1 0 1] 9 [1 0 0 1] {'1': 255, '0': 769}

219it [00:12, 17.04it/s]

13 [1 1 0 1] 10 [1 0 1 0] {'1': 266, '0': 758}
13 [1 1 0 1] 11 [1 0 1 1] {'0': 1024}

221it [00:12, 16.95it/s]

13 [1 1 0 1] 12 [1 1 0 0] {'0': 773, '1': 251}
13 [1 1 0 1] 13 [1 1 0 1] {'1': 1024}

223it [00:12, 17.21it/s]

13 [1 1 0 1] 14 [1 1 1 0] {'0': 1024}
13 [1 1 0 1] 15 [1 1 1 1] {'1': 237, '0': 787}

225it [00:12, 17.00it/s]

14 [1 1 1 0] 0 [0 0 0 0] {'0': 761, '1': 263}
14 [1 1 1 0] 1 [0 0 0 1] {'1': 1024}

227it [00:12, 16.73it/s]
```

```
14 [1 1 1 0] 2 [0 0 1 0] {'0': 1024}
14 [1 1 1 0] 3 [0 0 1 1] {'0': 751, '1': 273}

229it [00:12, 16.74it/s]

14 [1 1 1 0] 4 [0 1 0 0] {'0': 1024}
14 [1 1 1 0] 5 [0 1 0 1] {'1': 264, '0': 760}

231it [00:12, 16.73it/s]

14 [1 1 1 0] 6 [0 1 1 0] {'1': 262, '0': 762}
14 [1 1 1 0] 7 [0 1 1 1] {'0': 1024}

233it [00:13, 16.87it/s]

14 [1 1 1 0] 8 [1 0 0 0] {'0': 1024}
14 [1 1 1 0] 9 [1 0 0 1] {'0': 726, '1': 298}

235it [00:13, 17.65it/s]

14 [1 1 1 0] 10 [1 0 1 0] {'1': 256, '0': 768}
14 [1 1 1 0] 11 [1 0 1 1] {'0': 1024}

239it [00:13, 13.85it/s]

14 [1 1 1 0] 12 [1 1 0 0] {'1': 254, '0': 770}
14 [1 1 1 0] 13 [1 1 0 1] {'0': 1024}
14 [1 1 1 0] 14 [1 1 1 0] {'1': 1024}
14 [1 1 1 0] 15 [1 1 1 1] {'0': 772, '1': 252}

243it [00:13, 15.00it/s]

15 [1 1 1 1] 0 [0 0 0 0] {'1': 1024}
15 [1 1 1 1] 1 [0 0 0 1] {'1': 257, '0': 767}
15 [1 1 1 1] 2 [0 0 1 0] {'1': 265, '0': 759}
15 [1 1 1 1] 3 [0 0 1 1] {'0': 1024}

247it [00:14, 15.18it/s]

15 [1 1 1 1] 4 [0 1 0 0] {'0': 760, '1': 264}
15 [1 1 1 1] 5 [0 1 0 1] {'0': 1024}
15 [1 1 1 1] 6 [0 1 1 0] {'0': 1024}

251it [00:14, 15.43it/s]

15 [1 1 1 1] 7 [0 1 1 1] {'1': 243, '0': 781}
15 [1 1 1 1] 8 [1 0 0 0] {'1': 249, '0': 775}
15 [1 1 1 1] 9 [1 0 0 1] {'0': 1024}
15 [1 1 1 1] 10 [1 0 1 0] {'0': 1024}

255it [00:14, 15.92it/s]

15 [1 1 1 1] 11 [1 0 1 1] {'0': 760, '1': 264}
15 [1 1 1 1] 12 [1 1 0 0] {'0': 1024}
15 [1 1 1 1] 13 [1 1 0 1] {'1': 253, '0': 771}
15 [1 1 1 1] 14 [1 1 1 0] {'0': 763, '1': 261}
```

```
256it [00:14, 17.47it/s]
15 [1 1 1 1] 15 [1 1 1 1] {'1': 1024}
```

[36]:
```
import matplotlib.pyplot as plt

fig, ax = plt.subplots()
ax.set_xlabel("weight")
ax.set_ylabel("input")

out_plot = ax.imshow(outcomes, cmap='Blues', interpolation='none')
fig.colorbar(out_plot, ax=ax)
```

[36]:
```
<matplotlib.colorbar.Colorbar at 0x737894265fd0>
```



[ ]:

70-perceptron-ibmq

May 26, 2026

[1]:
```
%load_ext dotenv
%dotenv
```

[2]:
```
import os
from qiskit import transpile
from qiskit_ibm_runtime import QiskitRuntimeService, SamplerV2
from qiskit.visualization import plot_histogram
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
import numpy as np
```

1 Definition of th sign flip

$$\mathrm{SF}_{N,j} = O_j(C^N Z)O_j$$

$$O_j = \otimes_{l=0}^{N-1}(\mathrm{NOT}_l)^{1-j_l}$$

Choose N and j. j represents the elements of the computational basis.

[3]:
```
def num2bin(num, n):
    return np.array([int(i) for i in np.binary_repr(num, n)])

def O(q, j):
    O_circ = QuantumCircuit(q, name=f"O_{''.join([str(i) for i in j])}")

    for (i, v) in enumerate(j):
        if v == 0:
            O_circ.x(q[i])
    return  O_circ

def sf(q, j):
    N = q.size
    SF = QuantumCircuit(q, name=f"SF_{q.size}_{''.join([str(i) for i in j])}")
    SF.append(O(q, j), q)
    SF.barrier(q)
    SF.mcrz(np.pi, q[:N-1], q[N-1])
    SF.barrier(q)
    SF.append(O(q, j), q)
```

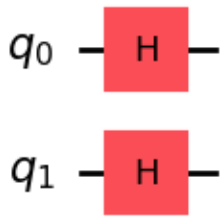
```
    return SF
```

[4]:
```
def U_i(q, input_vector):
    u_i = QuantumCircuit(q, name=f"U^i_{''.join([str(i) for i in␣
 ↪input_vector])}")
    N = q.size
    u_i.h(q)
    for (i, v) in enumerate(input_vector):
        if v:
            bin_num = num2bin(i, N)
            u_i.append(sf(q, bin_num), q)
    return u_i
```

[5]:
```
def U_w(q, weight_vector):
    u_w = QuantumCircuit(q, name=f"U^w_{''.join([str(i) for i in␣
 ↪weight_vector])}")
    N = q.size
    for (i, v) in enumerate(weight_vector):
        if v:
            bin_num = num2bin(i, N)
            u_w.append(sf(q, bin_num), q)
    u_w.h(q)
    u_w.x(q)
    return u_w
```

[6]:
```
N = 2
q = QuantumRegister(N, 'q')
input_vector = [0, 0, 0, 0]
circ = QuantumCircuit(q)
circ.append(U_i(q, input_vector), q)
circ.decompose(reps=1).draw(output='mpl')
```

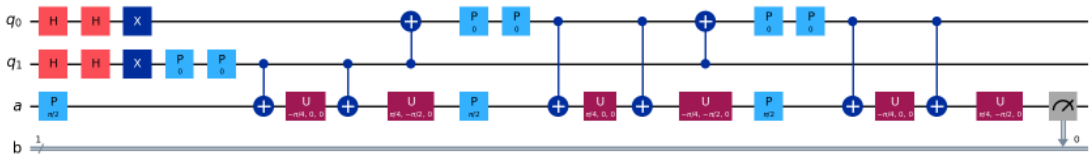
[6]:



[7]:
```
N = 2
q = QuantumRegister(N, 'q')
a = QuantumRegister(1, 'a')
```

```
b = ClassicalRegister(1, 'b')
circ = QuantumCircuit(q, a, b)
circ.append(U_i(q, num2bin(0, N)[::-1]), q)
circ.append(U_w(q, num2bin(0, N)), q)
circ.mcrx(np.pi, q, a[0])
circ.measure(a, b)
circ.decompose(reps=1).draw(output='mpl')
```

[7]:



[8]:
```
token = os.environ["IBMQ_TOKEN"]
```

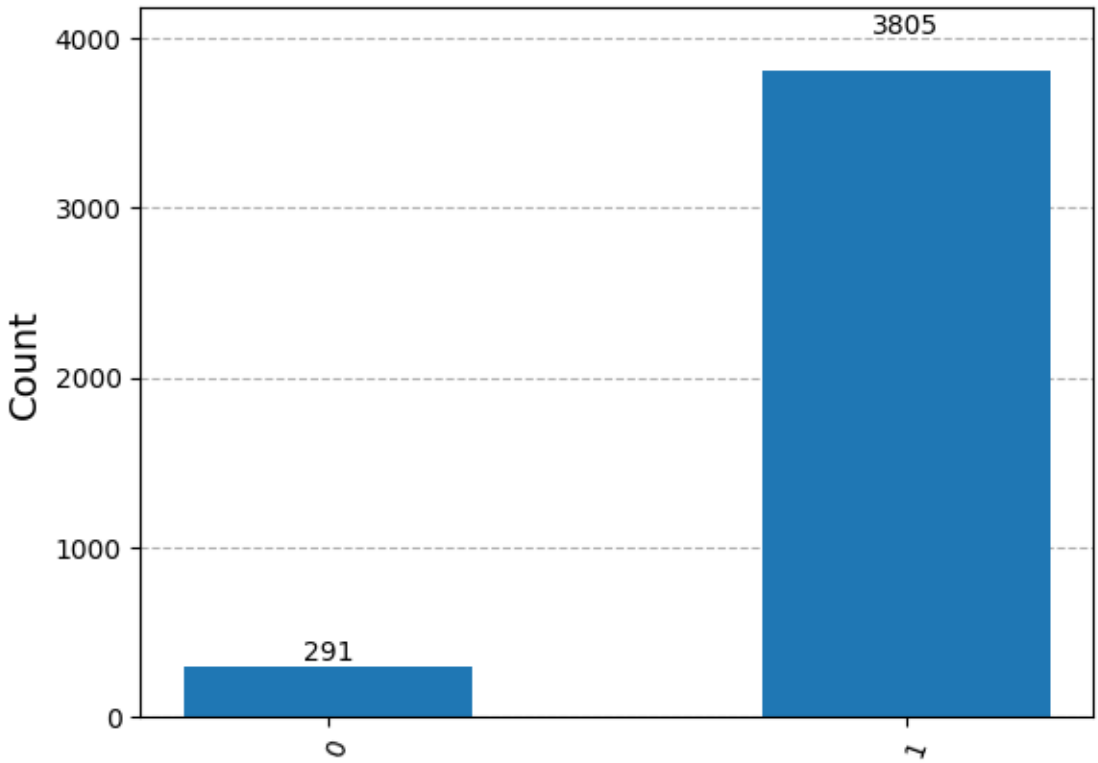
[9]:
```
service = QiskitRuntimeService(
                token=token,
                channel="ibm_quantum",
            )
backend = service.least_busy(operational=True, simulator=False)
```

```
/tmp/ipykernel_17005/1493915476.py:1: DeprecationWarning: The "ibm_quantum"
channel option is deprecated and will be sunset on 1 July. After this date,
ibm_cloud will be the only valid channel. For information on migrating to the
new IBM Quantum Platform on the "ibm_cloud" channel, review the migration guide
https://quantum.cloud.ibm.com/docs/migration-guides/classic-iqp-to-cloud-iqp .
  service = QiskitRuntimeService(
```

[10]:
```
transpiled = transpile(circ, backend=backend)
sampler = SamplerV2(backend)
job = sampler.run([transpiled])
result = job.result()
counts = result[0].data.b.get_counts()
print("\nTotal counts are:",counts)
plot_histogram(counts)
```

```
Total counts are: {'1': 3805, '0': 291}
```

[10]:



[11]:
```
import mthree
mit2 = mthree.M3Mitigation(backend)
```

[12]:
```
mit2.cals_from_system()
```

[12]:
```
[<RunningManJob('d0q1ea94anx0008kyy70', backend='ibm_sherbrooke',
mode_id=None)>]
```

[13]:
```
mit2.apply_correction(counts, [N+1])
```

[13]:
```
{'0': np.float32(-0.05580467), '1': np.float32(1.0558047)}
```

[ ]:
```

```

80-qnn

May 26, 2026

[6]:
```
!pip install qiskit-machine-learning
```

```
Requirement already satisfied: qiskit-machine-learning in
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (0.8.2)
Requirement already satisfied: qiskit>=1.0 in
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (from
qiskit-machine-learning) (1.4.2)
Requirement already satisfied: scipy>=1.4 in
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (from
qiskit-machine-learning) (1.15.2)
Requirement already satisfied: numpy>=1.17 in
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (from
qiskit-machine-learning) (2.2.5)
Requirement already satisfied: psutil>=5 in
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (from
qiskit-machine-learning) (7.0.0)
Requirement already satisfied: scikit-learn>=1.2 in
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (from
qiskit-machine-learning) (1.6.1)
Requirement already satisfied: setuptools>=40.1 in
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (from
qiskit-machine-learning) (75.8.0)
Requirement already satisfied: dill>=0.3.4 in
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (from
qiskit-machine-learning) (0.4.0)
Requirement already satisfied: rustworkx>=0.15.0 in
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (from
qiskit>=1.0->qiskit-machine-learning) (0.16.0)
Requirement already satisfied: sympy>=1.3 in
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (from
qiskit>=1.0->qiskit-machine-learning) (1.13.3)
Requirement already satisfied: python-dateutil>=2.8.0 in
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (from
qiskit>=1.0->qiskit-machine-learning) (2.9.0.post0)
Requirement already satisfied: stevedore>=3.0.0 in
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (from
qiskit>=1.0->qiskit-machine-learning) (5.4.1)
Requirement already satisfied: typing-extensions in
```

```
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (from
qiskit>=1.0->qiskit-machine-learning) (4.13.2)
Requirement already satisfied: symengine<0.14,>=0.11 in
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (from
qiskit>=1.0->qiskit-machine-learning) (0.13.0)
Requirement already satisfied: joblib>=1.2.0 in
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (from
scikit-learn>=1.2->qiskit-machine-learning) (1.5.0)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (from
scikit-learn>=1.2->qiskit-machine-learning) (3.6.0)
Requirement already satisfied: six>=1.5 in
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (from
python-dateutil>=2.8.0->qiskit>=1.0->qiskit-machine-learning) (1.17.0)
Requirement already satisfied: pbr>=2.0.0 in
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (from
stevedore>=3.0.0->qiskit>=1.0->qiskit-machine-learning) (6.1.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in
/home/lpawela/anaconda3/envs/parrondo-old/lib/python3.12/site-packages (from
sympy>=1.3->qiskit>=1.0->qiskit-machine-learning) (1.3.0)
```

[7]:
```
%load_ext dotenv
%dotenv
```

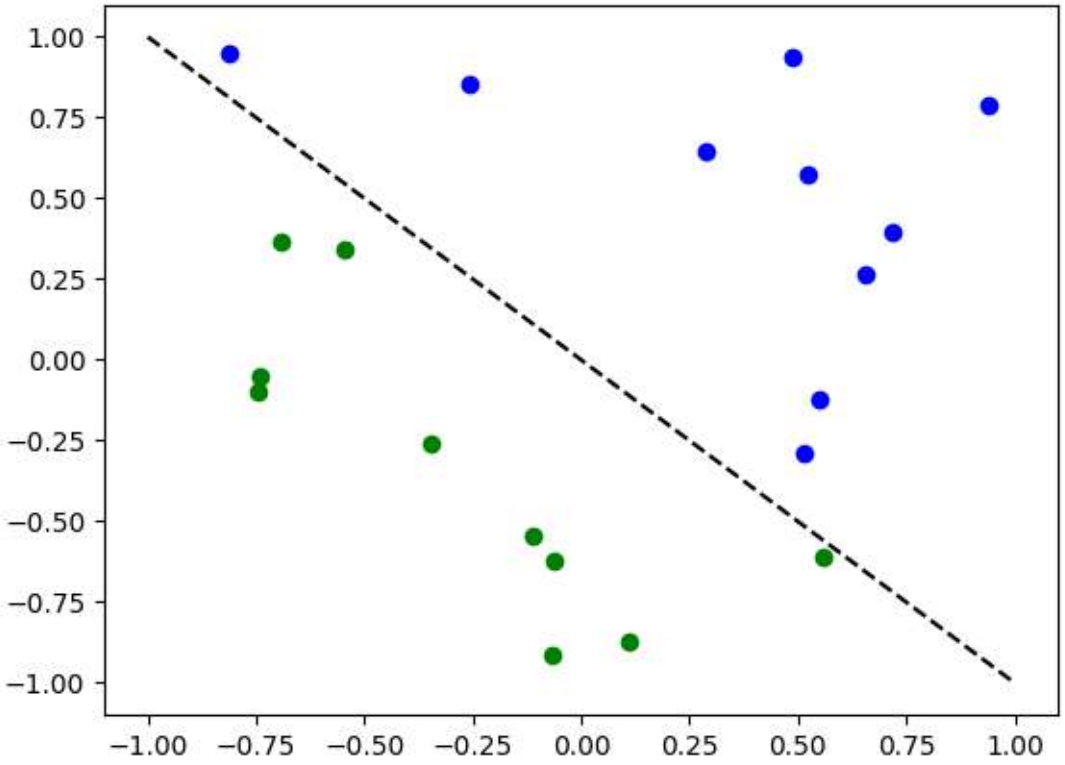
```
The dotenv extension is already loaded. To reload it, use:
  %reload_ext dotenv
```

[8]:
```
import matplotlib.pyplot as plt
import numpy as np
from IPython.display import clear_output
# from qiskit_machine_learning.optimizers import COBYLA
from qiskit_machine_learning.utils import algorithm_globals

from qiskit_machine_learning.algorithms.classifiers import␣
 ↪NeuralNetworkClassifier
from qiskit_machine_learning.neural_networks import EstimatorQNN

algorithm_globals.random_seed = 42
```

0.1 Data preparation

[9]:
```
num_inputs = 2
num_samples = 20
X = 2 * algorithm_globals.random.random([num_samples, num_inputs]) - 1
y01 = 1 * (np.sum(X, axis=1) >= 0)  # in { 0,  1}
y = 2 * y01 - 1  # in {-1, +1}
y_one_hot = np.zeros((num_samples, 2))
for i in range(num_samples):
```

```
    y_one_hot[i, y01[i]] = 1

for x, y_target in zip(X, y):
    if y_target == 1:
        plt.plot(x[0], x[1], "bo")
    else:
        plt.plot(x[0], x[1], "go")
plt.plot([-1, 1], [1, -1], "--", color="black")
plt.show()
```



[10]:
```
def callback_graph(weights, obj_func_eval):
    clear_output(wait=True)
    objective_func_vals.append(obj_func_eval)
    plt.title("Objective function value against iteration")
    plt.xlabel("Iteration")
    plt.ylabel("Objective function value")
    plt.plot(range(len(objective_func_vals)), objective_func_vals)
    plt.show()
```

[11]:
```
from qiskit.primitives import StatevectorEstimator as Estimator

estimator = Estimator()
```

```
estimator_qnn = EstimatorQNN(circuit=qc, estimator=estimator)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[11], line 4
      1 from qiskit.primitives import StatevectorEstimator as Estimator
      3 estimator = Estimator()
----> 4 estimator_qnn = EstimatorQNN(circuit=qc, estimator=estimator)

NameError: name 'qc' is not defined
```

[ ]:
```
estimator_classifier = NeuralNetworkClassifier(
    estimator_qnn, optimizer=COBYLA(maxiter=60), callback=callback_graph
)
```

[ ]:
```
estimator_qnn.forward(X[0, :], algorithm_globals.random.random(estimator_qnn.
 ↪num_weights))
```

[ ]:
```
# create empty array for callback to store evaluations of the objective function
objective_func_vals = []
plt.rcParams["figure.figsize"] = (12, 6)

# fit classifier to data
estimator_classifier.fit(X, y)

# return to default figsize
plt.rcParams["figure.figsize"] = (6, 4)

# score classifier
estimator_classifier.score(X, y)
```

[ ]:
```
# evaluate data points
y_predict = estimator_classifier.predict(X)

# plot results
# red == wrongly classified
for x, y_target, y_p in zip(X, y, y_predict):
    if y_target == 1:
        plt.plot(x[0], x[1], "bo")
    else:
        plt.plot(x[0], x[1], "go")
    if y_target != y_p:
        plt.scatter(x[0], x[1], s=200, facecolors="none", edgecolors="r",␣
 ↪linewidths=2)
plt.plot([-1, 1], [1, -1], "--", color="black")
plt.show()
```

[ ]:
```
estimator_classifier.weights
```

In [ ]:

Deutsch's problem

Input: a function $f:\{0,1\} \to {0,1}$

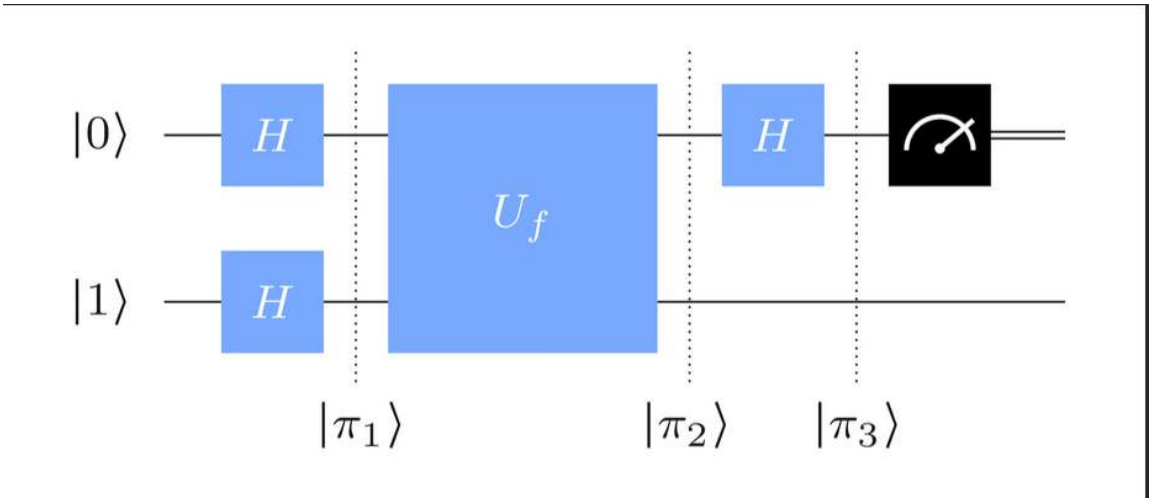
Output: 0 if f is constant, 1 if f is balanced

$f_0(0)=0, \; f_0(1)=0$

$f_1(0)=0, \; f_1(1)=1$

$f_2(0)=1, \; f_2(1)=0$

$f_3(0)=1, \; f_3(1)=1$



$U_f \ket{x} \ket{y} = \ket{x} \ket{y \oplus f(x)}$

$\ket{\pi_1} = \ket{+} \ket{-}$

$\ket{\pi_2} = (-1)^{f(0)}\left(\frac{\ket{0} + (-1)^{f(0) \oplus f(1)}\ket{1}}{\sqrt{2}} \right) \ket{-}$

$\ket{\pi_3} = H\ket{+} = \ket{0}$ if $f(0) \oplus f(1) = 0$

$\ket{\pi_3} = H\ket{-} = \ket{1}$ if $f(0) \oplus f(1) = 1$

In [1]:
```
from qiskit import QuantumCircuit
from qiskit_aer import AerSimulator
```

In [2]:
```
def oracle(case: int) -> QuantumCircuit:
    if case not in range(4):
        raise ValueError("Case must be one of [0, 1, 2, 3]")
    qc = QuantumCircuit(2)
    if case == 1 or case == 2:
```

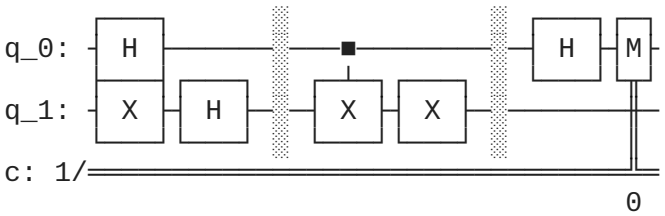
```
        qc.cx(0, 1)
    if case == 2 or case == 3:
        qc.x(1)
    return qc
```

In [3]:
```
def build_circuit(case: int) -> QuantumCircuit:
    qc = QuantumCircuit(2, 1)
    qc.x(1)
    qc.h(0)
    qc.h(1)
    qc.barrier()
    qc.compose(oracle(case), inplace=True)
    qc.barrier()
    qc.h(0)
    qc.measure(0, 0)
    return qc
```

In [4]:
```
qc = build_circuit(2)
```

In [5]:
```
qc.draw()
```

Out[5]:



In [17]:
```
qc = build_circuit(2)
result = AerSimulator().run(qc, shots=1024).result()
counts = result.get_counts()
```

In [18]:
```
counts
```

Out[18]:
```
{'1': 1024}
```

In [19]:
```
%load_ext dotenv
%dotenv
```

```
The dotenv extension is already loaded. To reload it, use:
  %reload_ext dotenv
```

In [20]:
```
import os
from qiskit import transpile
from qiskit_ibm_runtime import QiskitRuntimeService, SamplerV2
from qiskit.visualization import plot_histogram
```

In [21]:
```
token = os.environ["IBMQ_TOKEN"]
```

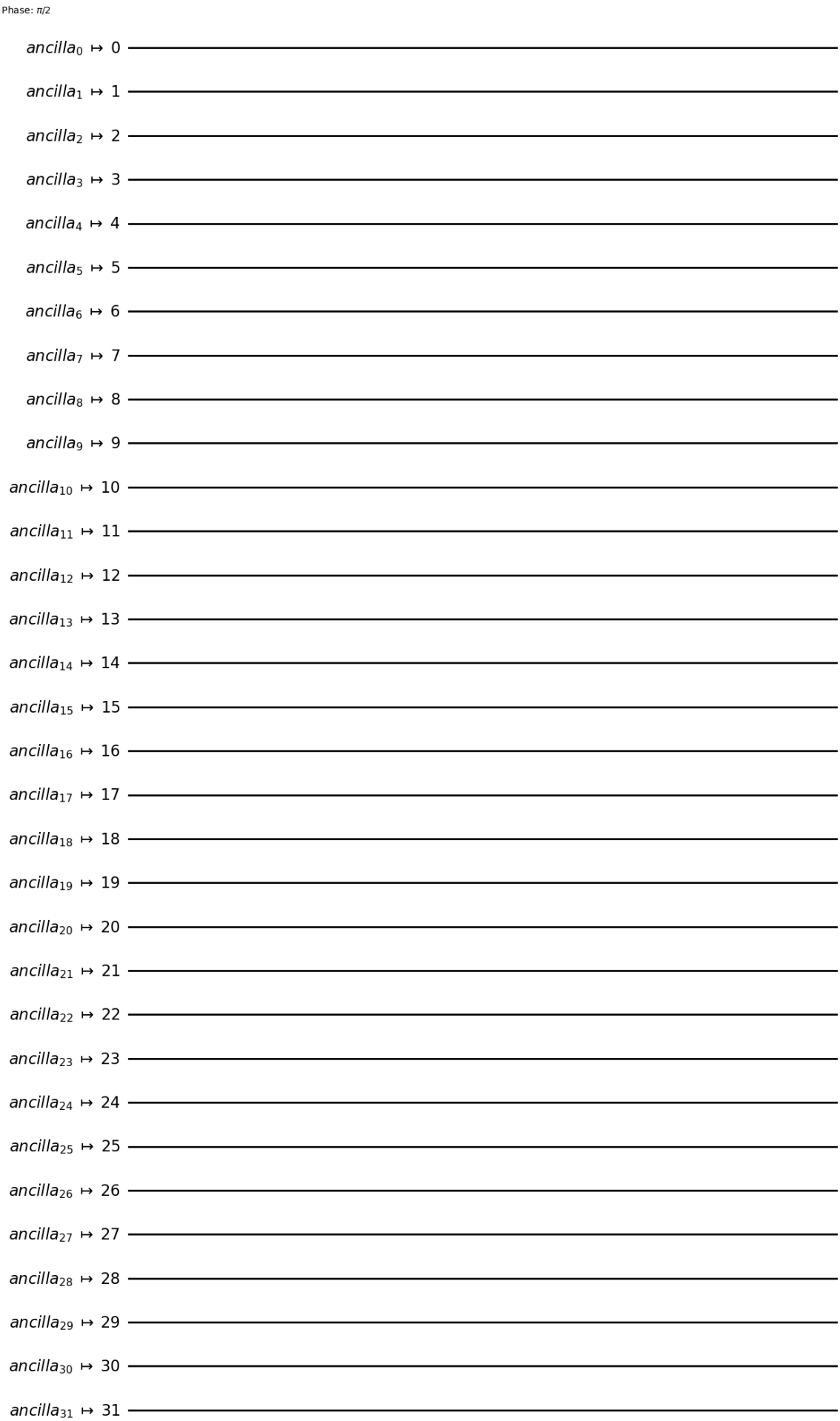
In [22]:
```
service = QiskitRuntimeService(
                token=token,
                channel="ibm_quantum",
            )
backend = service.least_busy(operational=True, simulator=False)
```

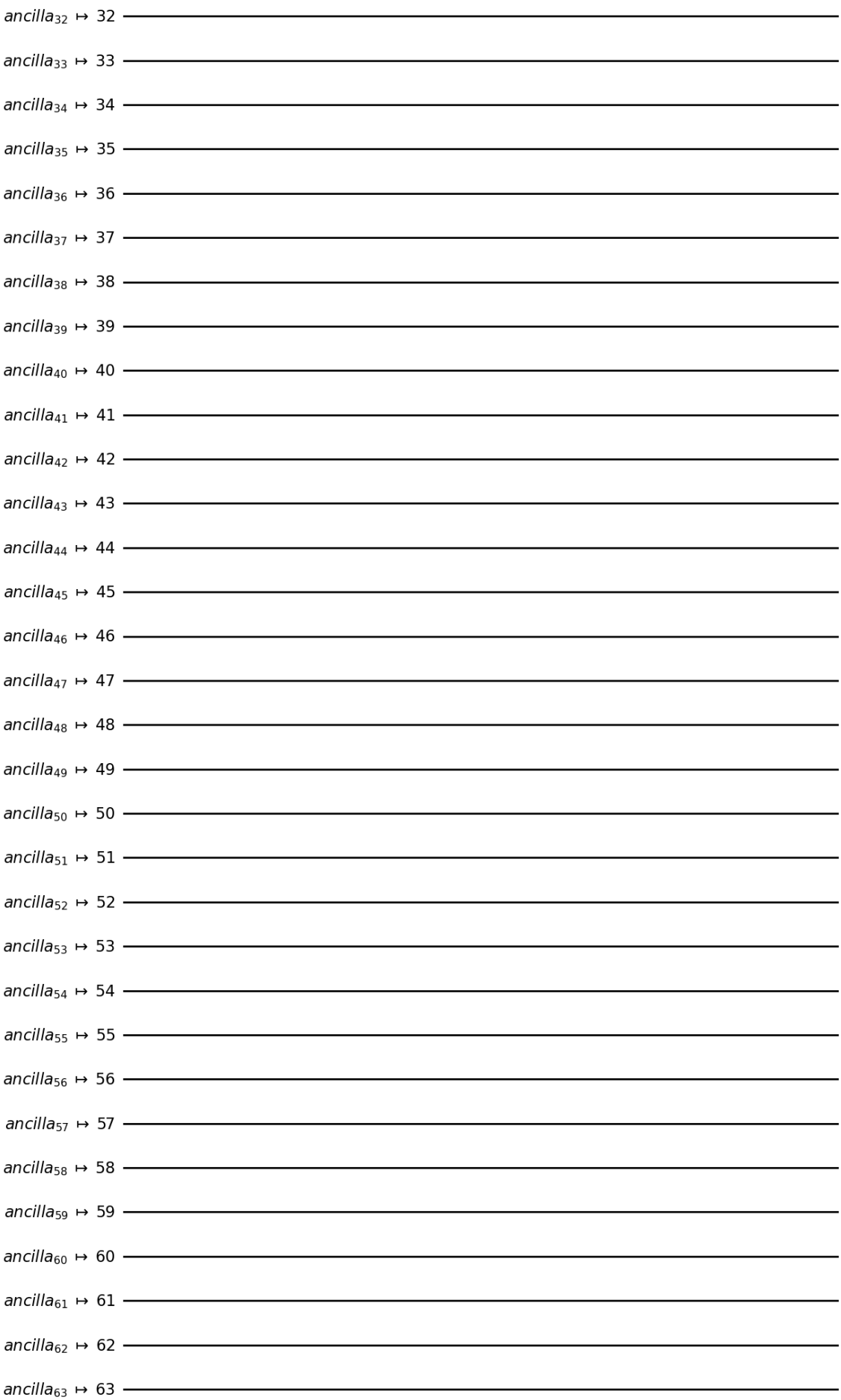
```
/tmp/ipykernel_27357/1493915476.py:1: DeprecationWarning: The "ibm_quantum"
channel option is deprecated and will be sunset on 1 July. After this date,
ibm_cloud will be the only valid channel. For information on migrating to th
e new IBM Quantum Platform on the "ibm_cloud" channel, review the migration
guide https://quantum.cloud.ibm.com/docs/migration-guides/classic-iqp-to-clo
ud-iqp .
  service = QiskitRuntimeService(
```

In [23]:
```
transpiled = transpile(qc, backend=backend)
```

In [24]:
```
transpiled.draw(output="mpl")
```

Out[24]:


Global Phase: π/2
ancilla0 ↦ 0
ancilla1 ↦ 1
ancilla2 ↦ 2
ancilla3 ↦ 3
ancilla4 ↦ 4
ancilla5 ↦ 5
ancilla6 ↦ 6
ancilla7 ↦ 7
ancilla8 ↦ 8
ancilla9 ↦ 9
ancilla10 ↦ 10
ancilla11 ↦ 11
ancilla12 ↦ 12
ancilla13 ↦ 13
ancilla14 ↦ 14
ancilla15 ↦ 15
ancilla16 ↦ 16
ancilla17 ↦ 17
ancilla18 ↦ 18
ancilla19 ↦ 19
ancilla20 ↦ 20
ancilla21 ↦ 21
ancilla22 ↦ 22
ancilla23 ↦ 23
ancilla24 ↦ 24
ancilla25 ↦ 25
ancilla26 ↦ 26
ancilla27 ↦ 27
ancilla28 ↦ 28
ancilla29 ↦ 29
ancilla30 ↦ 30
ancilla31 ↦ 31

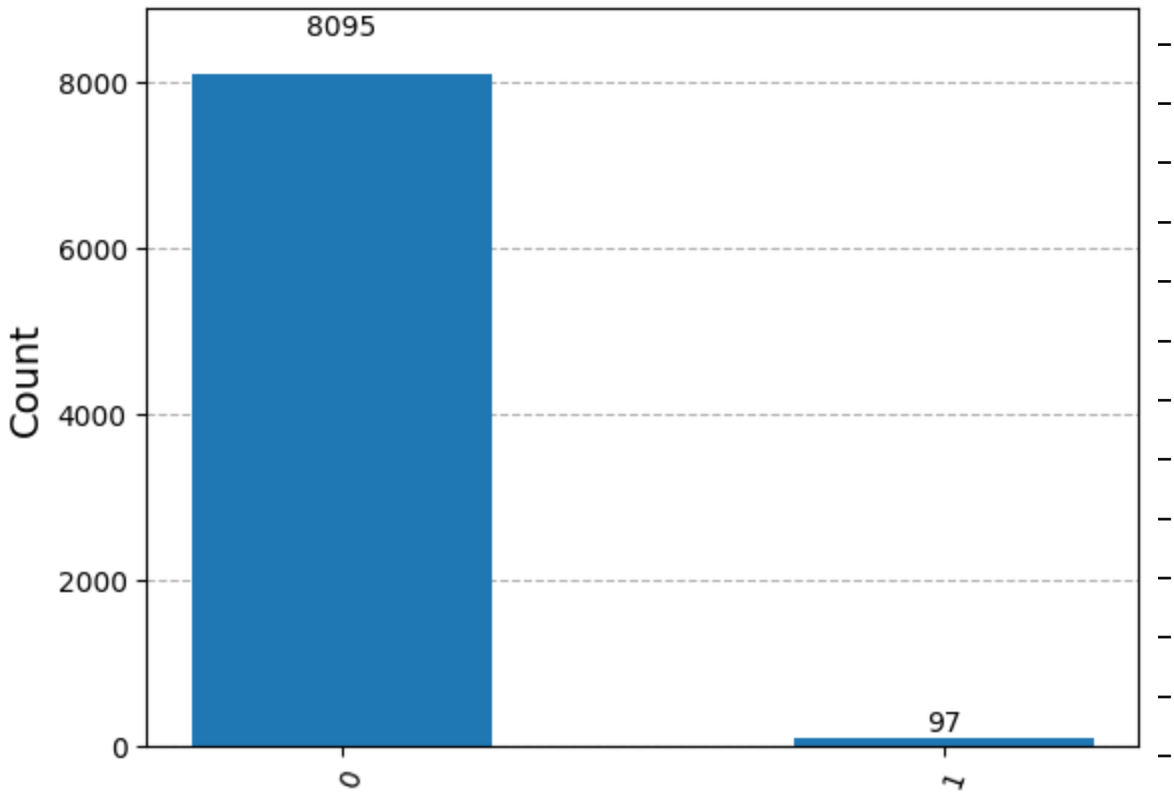

$ancilla_{32} \mapsto 32$
$ancilla_{33} \mapsto 33$
$ancilla_{34} \mapsto 34$
$ancilla_{35} \mapsto 35$
$ancilla_{36} \mapsto 36$
$ancilla_{37} \mapsto 37$
$ancilla_{38} \mapsto 38$
$ancilla_{39} \mapsto 39$
$ancilla_{40} \mapsto 40$
$ancilla_{41} \mapsto 41$
$ancilla_{42} \mapsto 42$
$ancilla_{43} \mapsto 43$
$ancilla_{44} \mapsto 44$
$ancilla_{45} \mapsto 45$
$ancilla_{46} \mapsto 46$
$ancilla_{47} \mapsto 47$
$ancilla_{48} \mapsto 48$
$ancilla_{49} \mapsto 49$
$ancilla_{50} \mapsto 50$
$ancilla_{51} \mapsto 51$
$ancilla_{52} \mapsto 52$
$ancilla_{53} \mapsto 53$
$ancilla_{54} \mapsto 54$
$ancilla_{55} \mapsto 55$
$ancilla_{56} \mapsto 56$
$ancilla_{57} \mapsto 57$
$ancilla_{58} \mapsto 58$
$ancilla_{59} \mapsto 59$
$ancilla_{60} \mapsto 60$
$ancilla_{61} \mapsto 61$
$ancilla_{62} \mapsto 62$
$ancilla_{63} \mapsto 63$

In [ ]:
```
sampler = SamplerV2(backend)
job = sampler.run([transpiled], shots=8192)
result = job.result()
```

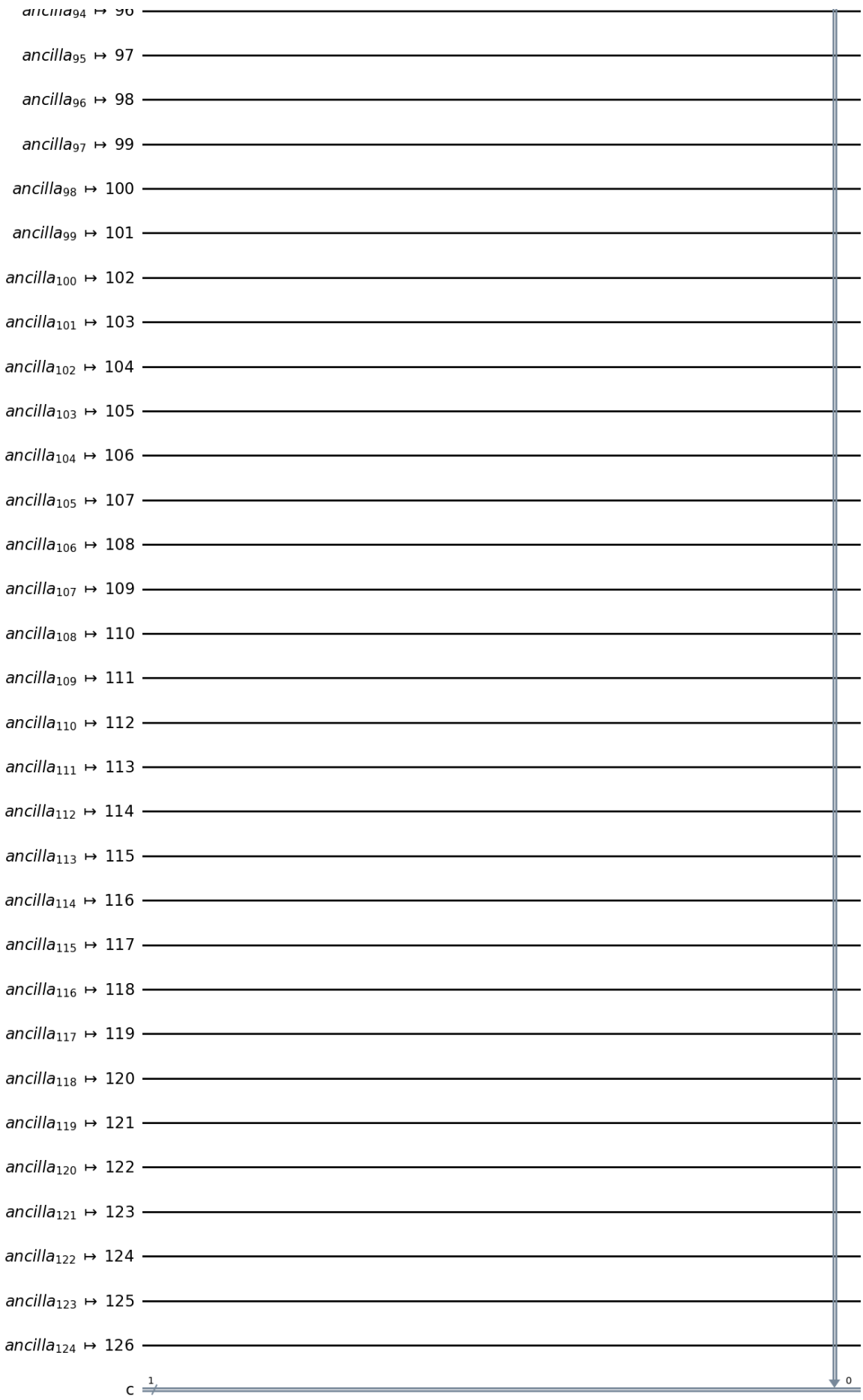
In [ ]:
```
pub_result = result[0].data.c.get_counts()
```

In [ ]:
```
plot_histogram(pub_result)
```

Out[ ]:



In [ ]:


ancilla94 ↦ 96
ancilla95 ↦ 97
ancilla96 ↦ 98
ancilla97 ↦ 99
ancilla98 ↦ 100
ancilla99 ↦ 101
ancilla100 ↦ 102
ancilla101 ↦ 103
ancilla102 ↦ 104
ancilla103 ↦ 105
ancilla104 ↦ 106
ancilla105 ↦ 107
ancilla106 ↦ 108
ancilla107 ↦ 109
ancilla108 ↦ 110
ancilla109 ↦ 111
ancilla110 ↦ 112
ancilla111 ↦ 113
ancilla112 ↦ 114
ancilla113 ↦ 115
ancilla114 ↦ 116
ancilla115 ↦ 117
ancilla116 ↦ 118
ancilla117 ↦ 119
ancilla118 ↦ 120
ancilla119 ↦ 121
ancilla120 ↦ 122
ancilla121 ↦ 123
ancilla122 ↦ 124
ancilla123 ↦ 125
ancilla124 ↦ 126
c
1
0

90-pulse-gates

May 26, 2026

1 Pulse gates

Most quantum algorithms can be described with circuit operations alone. When we need more control over the low-level implementation of our program, we can use *pulse gates*. Pulse gates remove the constraint of executing circuits with basis gates only, and also allow you to override the default implementation of any basis gate.

Pulse gates allow you to map a logical circuit gate (e.g., `X`) to a Qiskit Pulse program, called a `Schedule`. This mapping is referred to as a *calibration*. A high fidelity calibration is one which faithfully implements the logical operation it is mapped from (e.g., whether the `X` gate calibration drives $|0\rangle$ to $|1\rangle$, etc.).

A schedule specifies the exact time dynamics of the input signals across all input *channels* to the device. There are usually multiple channels per qubit, such as drive and measure. This interface is more powerful, and requires a deeper understanding of the underlying device physics.

It's important to note that Pulse programs operate on physical qubits. A drive pulse on qubit a will not enact the same logical operation on the state of qubit b – in other words, gate calibrations are not interchangeable across qubits. This is in contrast to the circuit level, where an `X` gate is defined independent of its qubit operand.

This page shows you how to add a calibration to your circuit.

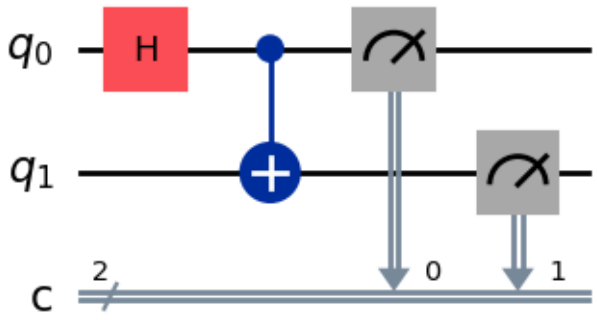
Note: To execute a program with pulse gates, the backend has to be enabled with OpenPulse. You can check via `backend.configuration().open_pulse`, which is `True` when OpenPulse is enabled. If it is enabled and the pulse gates feature is not enabled, you can schedule your input circuit.

1.1 Build your circuit

Let's start with a very simple example, a Bell state circuit.

[1]:
```
from qiskit import QuantumCircuit

circ = QuantumCircuit(2, 2)
circ.h(0)
circ.cx(0, 1)
circ.measure(0, 0)
circ.measure(1, 1)

circ.draw('mpl')
```

[1]:



1.2 Build your calibrations

Now that we have our circuit, let's define a calibration for the Hadamard gate on qubit 0.

In practice, the pulse shape and its parameters would be optimized through a series of Rabi experiments (see the Qiskit Textbook for a walk through). For this demonstration, our Hadamard will be a Gaussian pulse. We will *play* our pulse on the *drive* channel of qubit 0.

Don't worry too much about the details of building the calibration itself; you can learn all about this on the following page: building pulse schedules.

[2]:
```
from qiskit import pulse
from qiskit.pulse.library import Gaussian
from qiskit_ibm_runtime.fake_provider import FakeSherbrooke

backend = FakeSherbrooke()

with pulse.build(backend, name='hadamard') as h_q0:
    pulse.play(Gaussian(duration=128, amp=0.1, sigma=16), pulse.
 ↪drive_channel(0))
```

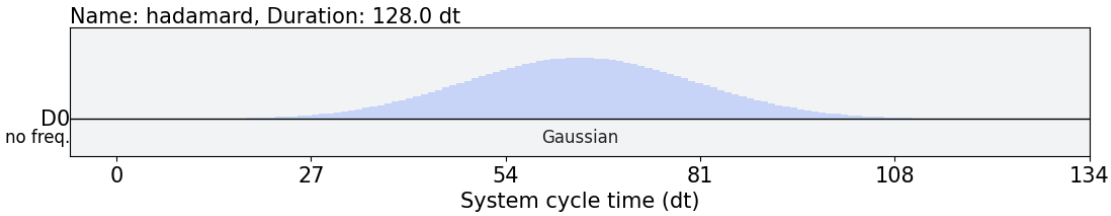
```
/tmp/ipykernel_22500/2020689717.py:7: DeprecationWarning: The function
``qiskit.pulse.builder.build()`` is deprecated as of Qiskit 1.3. It will be
removed in Qiskit 2.0. The entire Qiskit Pulse package is being deprecated and
will be moved to the Qiskit Dynamics repository: https://github.com/qiskit-
community/qiskit-dynamics
  with pulse.build(backend, name='hadamard') as h_q0:
/tmp/ipykernel_22500/2020689717.py:8: DeprecationWarning: The function
``qiskit.pulse.builder.drive_channel()`` is deprecated as of Qiskit 1.3. It will
be removed in Qiskit 2.0. The entire Qiskit Pulse package is being deprecated
and will be moved to the Qiskit Dynamics repository: https://github.com/qiskit-
community/qiskit-dynamics
  pulse.play(Gaussian(duration=128, amp=0.1, sigma=16), pulse.drive_channel(0))
```

```
---------------------------------------------------------------------------
NotImplementedError                       Traceback (most recent call last)
Cell In[2], line 8
      5 backend = FakeSherbrooke()
      7 with pulse.build(backend, name='hadamard') as h_q0:
----> 8     pulse.play(Gaussian(duration=128, amp=0.1, sigma=16),␣
 ↪pulse.drive_channel(0))

File ~/anaconda3/envs/parrondo-old/lib/python3.12/site-packages/qiskit/utils/
 ↪deprecation.py:97, in deprecate_func.<locals>.decorator.<locals>.
 ↪wrapper(*args, **kwargs)
     94 @functools.wraps(func)
     95 def wrapper(*args, **kwargs):
     96     warnings.warn(msg, category=category, stacklevel=stacklevel)
---> 97     return func(*args, **kwargs)

File ~/anaconda3/envs/parrondo-old/lib/python3.12/site-packages/qiskit/pulse/
 ↪builder.py:1424, in drive_channel(qubit)
   1422 # backendV2
   1423 if isinstance(active_backend(), BackendV2):
-> 1424     return active_backend().drive_channel(qubit)
   1425 return active_backend().configuration().drive(qubit)

File ~/anaconda3/envs/parrondo-old/lib/python3.12/site-packages/qiskit/utils/
 ↪deprecation.py:97, in deprecate_func.<locals>.decorator.<locals>.
 ↪wrapper(*args, **kwargs)
     94 @functools.wraps(func)
     95 def wrapper(*args, **kwargs):
     96     warnings.warn(msg, category=category, stacklevel=stacklevel)
---> 97     return func(*args, **kwargs)

File ~/anaconda3/envs/parrondo-old/lib/python3.12/site-packages/qiskit/provider
 ↪backend.py:548, in BackendV2.drive_channel(self, qubit)
    534 @deprecate_pulse_dependency
    535 def drive_channel(self, qubit: int):
    536     """Return the drive channel for the given qubit.
    537
    538     This is required to be implemented if the backend supports Pulse
   (…)    546             measurement mapping
    547     """
--> 548     raise NotImplementedError

NotImplementedError:
```

Let's draw the new schedule to see what we've built.

[ ]:
```
h_q0.draw()
```

[ ]:



1.3 Link your calibration to your circuit

All that remains is to complete the registration. The circuit method `add_calibration` needs information about the gate and a reference to the schedule to complete the mapping:

`QuantumCircuit.add_calibration(gate, qubits, schedule, parameters)`

The `gate` can either be a `circuit.Gate` object or the name of the gate. Usually, you'll need a different schedule for each unique set of `qubits` and `parameters`. Since the Hadamard gate doesn't have any parameters, we don't have to supply any.

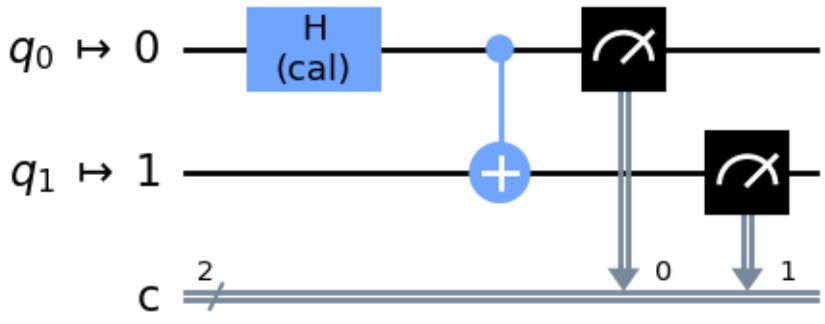
[ ]:
```
circ.add_calibration('h', [0], h_q0)
```

Lastly, note that the transpiler will respect your calibrations. Use it as you normally would (our example is too simple for the transpiler to optimize, so the output is the same).

[ ]:
```
from qiskit import transpile
from qiskit.providers.fake_provider import FakeHanoi

backend = FakeHanoi()

circ = transpile(circ, backend)

print(backend.configuration().basis_gates)
circ.draw('mpl', idle_wires=False)
```

```
['id', 'rz', 'sx', 'x', 'cx', 'reset']
```

[ ]:



Notice that `h` is not a basis gate for the mock backend `FakeHanoi`. Since we have added a calibration for it, the transpiler will treat our gate as a basis gate; *but only on the qubits for which it was defined.* A Hadamard applied to a different qubit would be unrolled to the basis gates.

That's it!

1.4 Custom gates

We'll briefly show the same process for nonstandard, completely custom gates. This demonstration includes a gate with parameters.

[ ]:
```
from qiskit import QuantumCircuit
from qiskit.circuit import Gate

circ = QuantumCircuit(1, 1)
custom_gate = Gate('my_custom_gate', 1, [3.14, 1])
# 3.14 is an arbitrary parameter for demonstration
circ.append(custom_gate, [0])
circ.measure(0, 0)

circ.draw('mpl')
```

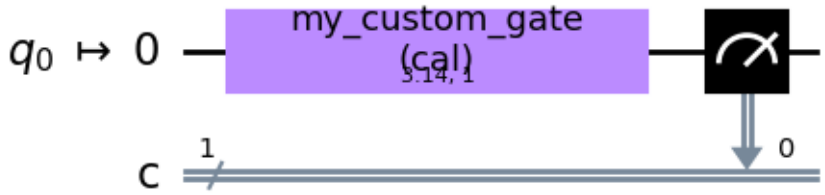
[ ]:

[ ]:
```
with pulse.build(backend, name='custom') as my_schedule:
    pulse.play(Gaussian(duration=64, amp=0.2, sigma=8), pulse.drive_channel(0))

circ.add_calibration('my_custom_gate', [0], my_schedule, [3.14, 1])
# Alternatively: circ.add_calibration(custom_gate, [0], my_schedule)
```

If we use the `Gate` instance variable `custom_gate` to add the calibration, the parameters are derived from that instance. Remember that the order of parameters is meaningful.

[ ]:
```
circ = transpile(circ, backend)
circ.draw('mpl', idle_wires=False)
```

[ ]:



Normally, if we tried to transpile our `circ`, we would get an error. There was no functional definition provided for `"my_custom_gate"`, so the transpiler can't unroll it to the basis gate set of the target device. We can show this by trying to add `"my_custom_gate"` to another qubit which hasn't been calibrated.

[ ]:
```
circ = QuantumCircuit(2, 2)
circ.append(custom_gate, [1])


from qiskit import QiskitError
try:
    circ = transpile(circ, backend)
except QiskitError as e:
    print(e)
```

```
"Cannot unroll the circuit to the given basis, ['id', 'rz', 'sx', 'x', 'cx',
'reset']. Instruction my_custom_gate not found in equivalence library and no
rule found to expand."
```

[ ]:
```
import qiskit.tools.jupyter
%qiskit_version_table
%qiskit_copyright
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

An Artificial Neuron Implemented on an Actual Quantum Processor

Francesco Tacchino,[1,*] Chiara Macchiavello,[1,2,3,†] Dario Gerace,[1,‡] and Daniele Bajoni[4,§]

[1] *Dipartimento di Fisica, Università di Pavia, via Bassi 6, I-27100, Pavia, Italy*
[2] *INFN Sezione di Pavia, via Bassi 6, I-27100, Pavia, Italy*
[3] *CNR-INO, largo E. Fermi 6, I-50125, Firenze, Italy*
[4] *Dipartimento di Ingegneria Industriale e dell'Informazione, Università di Pavia, via Ferrata 1, I-27100, Pavia, Italy*

(Dated: November 7, 2018)

Artificial neural networks are the heart of machine learning algorithms and artificial intelligence protocols. Historically, the simplest implementation of an artificial neuron traces back to the classical Rosenblatt's "perceptron", but its long term practical applications may be hindered by the fast scaling up of computational complexity, especially relevant for the training of multilayered perceptron networks. Here we introduce a quantum information-based algorithm implementing the quantum computer version of a perceptron, which shows exponential advantage in encoding resources over alternative realizations. We experimentally test a few qubits version of this model on an actual small-scale quantum processor, which gives remarkably good answers against the expected results. We show that this quantum model of a perceptron can be used as an elementary nonlinear classifier of simple patterns, as a first step towards practical training of artificial quantum neural networks to be efficiently implemented on near-term quantum processing hardware.

INTRODUCTION

Artificial neural networks are a class of computational models that have proven to be highly successful at specific tasks like pattern recognition, image classification, and decision making [1]. They are essentially made of a set of nodes, or neurons, and the corresponding set of mutual connections, whose architecture is naturally inspired by neural nodes and synaptic connections in biological systems [1, 2]. In practical applications, artificial neural networks are mostly run as classical algorithms on conventional computers, but considerable interest has also been devoted to *physical* neural networks, i.e. neural networks implemented on dedicated hardware [3–5].

Among the possible computing platforms, prospective quantum computers seem particularly well suited for implementing artificial neural networks [6]. In fact, the intrinsic property of Quantum Mechanics of representing and storing large complex valued vectors and matrices, as well as performing linear operations on such vectors, is believed to result in an exponential increase either in memory storage or processing power for neural networks directly implemented on quantum processors [7–16]. The simplest model of an artificial neuron, the so called "perceptron", was originally proposed by R. Rosenblatt in 1957 [17], and is schematically outlined in Fig. 1(a). A real valued vector, $\vec{i}$, of dimension m represents the input, and it is combined with a real valued weight vector, $\vec{w}$. The perceptron output is evaluated as a binary response function resulting from the inner product of the two vectors, with a threshold value deciding for the "yes/no" response. In the lowest level implementations, $\vec{i}$ and $\vec{w}$ are binary valued vectors themselves, as proposed by McCulloch and Pitts in 1943 as a simple model of a neuron [2, 18].

Perceptrons and McCulloch-Pitts neurons are limited in the operations that they can perform, but they are still at the basis of machine learning algorithms in more complex artificial neural networks in multilayered perceptron architectures. However, the computational complexity increases with increasing number of nodes and interlayer connectivity and it is not clear whether this could eventually call for a change in paradigm, although different strategies can be put forward to optimize the efficiency of classical algorithms [19]. In this respect, several proposals have been advanced in recent years to implement perceptrons on quantum computers. The most largely investigated concept is that of a "qubit neuron", in which each qubit (the computational unit in quantum computers) acts as an individual neuron within the network. Most of the research effort has been devoted to exploit the nonlinearity of the measurement process in order to implement the threshold function [8].

Here we introduce an alternative design that closely mimics a Rosenblatt perceptron on a quantum computer. First, the equivalent of m-dimensional classical input and weight vectors is encoded on the quantum hardware by using N qubits, where $m = 2^N$. On one hand, this evidently allows to exploit the exponential advantage of quantum information storage, as already pointed out [9, 12]. On the other, we implement an original procedure to generate multipartite entangled states based on quantum information principles [20] that allows to crucially scale down the quantum computational resources to be employed. We experimentally show the effectiveness of such an approach by practically implementing a 2 qubits version of the algorithm on the IBM quantum processor available for cloud quantum computing. In this respect, the present work constitutes a key step towards the efficient use of prospective quantum processing de-

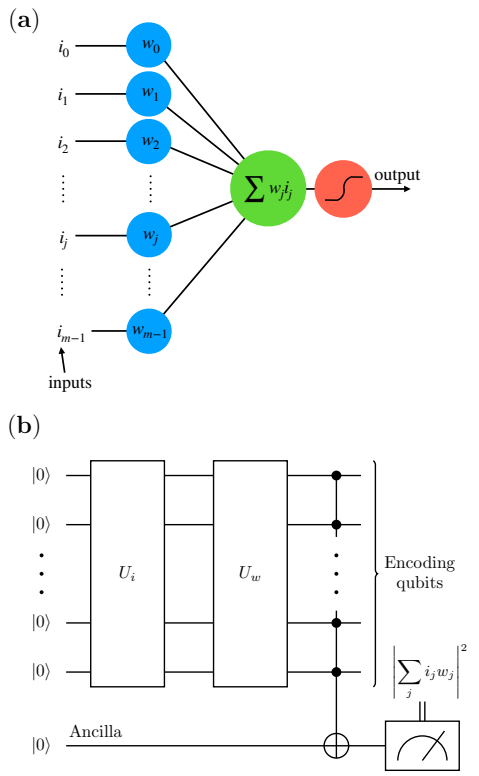


FIG. 1. **Perceptron models.** (a) Schematic outline of the classical perceptron as a model of artificial neuron: An input array $\vec{i}$ is processed with a weight vector $\vec{w}$ to produce a linear, binary valued output function. In its simplest realization, also the elements of $\vec{i}$ and $\vec{w}$ are binary valued, the perceptron acting as a binary (linear) classifier. (b) Scheme of the quantum algorithm for the implementation of the artificial neuron model on a quantum processor: From the system initialized in its idle configuration, the first two unitary operations prepare the input quantum state, $|\psi_i\rangle$, and implement the U_w transformation, respectively. The final outcome is then written on an ancilla qubit, which is eventually measured to evaluate the activation state of the perceptron.

vices for machine learning applications. Remarkably, we show that the quantum perceptron model can be used to sort out simple patterns, such as vertical or horizontal lines among all possible inputs. In order to show the potential of our proposed implementation of a quantum artificial neuron, we theoretically simulate a 4+1 qubits version using the IBM quantum simulator. We conclude the paper by discussing the usefulness of our algorithm as a quantum neuron in fully quantum neural networks.

QUANTUM CIRCUIT MODELING OF A CLASSICAL PERCEPTRON

A scheme of the quantum algorithm proposed in this work is shown in Fig. 1(b). The input and weight vectors are limited to binary values, $i_j, w_j \in \{-1, 1\}$, as in McCulloch-Pitts neurons. Hence, a m-dimensional input vector is encoded using the m coefficients needed to define a general wavefunction $|\psi_i\rangle$ of N qubits. In practice, given arbitrary input ($\vec{i}$) and weight ($\vec{w}$) vectors

$$\vec{i} = \begin{pmatrix} i_0 \\ i_1 \\ \vdots \\ i_{m-1} \end{pmatrix}, \quad \vec{w} = \begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_{m-1} \end{pmatrix} \tag{1}$$

with $i_j, w_j \in \{-1, 1\}$, we define the two quantum states

$$|\psi_i\rangle = \frac{1}{\sqrt{m}} \sum_{j=0}^{m-1} i_j |j\rangle; \quad |\psi_w\rangle = \frac{1}{\sqrt{m}} \sum_{j=0}^{m-1} w_j |j\rangle \ . \tag{2}$$

The states $|j\rangle \in \{|00\ldots00\rangle, |00\ldots01\rangle, \ldots, |11\ldots11\rangle\}$ form the so called computational basis of the quantum processor, i.e. the basis in the Hilbert space of N qubits, corresponding to all possible states of the single qubits being either in $|0\rangle$ or $|1\rangle$. As usual, these states are labeled with integers $j \in \{0, \ldots, m-1\}$ arising from the decimal representation of the respective binary string. Evidently, if N qubits are used in the register, there are $m = 2^N$ basis states labelled $|j\rangle$ and, as outlined in Eq. (2), we can use factors ± 1 to encode the m-dimensional classical vectors into an uniformly weighted superposition of the full computational basis.

The first step of the algorithm prepares the state $|\psi_i\rangle$ by encoding the input values in $\vec{i}$. Assuming the qubits to be initialized in the state $|00\ldots00\rangle \equiv |0\rangle^{\otimes N}$, we perform a unitary transformation U_i such that

$$U_i |0\rangle^{\otimes N} = |\psi_i\rangle \ . \tag{3}$$

In principle, any $m \times m$ unitary matrix having $\vec{i}$ in the first column can be used to this purpose, and we will give explicit examples in the following. Notice that, in a more general scenario, the preparation of the input state starting from a blank register might be replaced by a direct call to a quantum memory [21] where $|\psi_i\rangle$ was previously stored.

The second step computes the inner product between $\vec{w}$ and $\vec{i}$ using the quantum register. This task can be performed efficiently by defining a unitary transformation, U_w, such that the weight quantum state is rotated as

$$U_w |\psi_w\rangle = |1\rangle^{\otimes N} = |m-1\rangle \ . \tag{4}$$

As before, any $m \times m$ unitary matrix having $\vec{w}^T$ in the last row satisfies this condition. If we apply U_w after U_i, the overall N-qubits quantum state becomes

$$U_w |\psi_i\rangle = \sum_{j=0}^{m-1} c_j |j\rangle \equiv |\phi_{i,w}\rangle \, . \tag{5}$$

Using Eq. (4), the scalar product between the two quantum states is

$$\begin{aligned}\langle\psi_w|\psi_i\rangle &= \langle\psi_w|U_w^\dagger U_w|\psi_i\rangle = \\ &= \langle m-1|\phi_{i,w}\rangle = c_{m-1}\,,\end{aligned} \tag{6}$$

and from the definitions in Eq. (2) it is easily seen that the scalar product of input and weight vectors is $\vec{w}\cdot\vec{i} = m\langle\psi_w|\psi_i\rangle$. Therefore, the desired result is contained, up to a normalization factor, in the coefficient c_{m-1} of the final state $|\phi_{i,w}\rangle$.

In order to extract such an information, we propose to use an ancilla qubit (a) initially set in the state $|0\rangle$. A multi-controlled NOT gate between the N encoding qubits and the target a leads to [22]:

$$|\phi_{i,w}\rangle|0\rangle_a \rightarrow \sum_{j=0}^{m-2} c_j|j\rangle|0\rangle_a + c_{m-1}|m-1\rangle|1\rangle_a \tag{7}$$

The nonlinearity required by the threshold function at the output of the perceptron is immediately obtained by performing a quantum measurement: indeed, by measuring the state of the ancilla qubit in the computational basis produces the output $|1\rangle_a$ (i.e., an activated perceptron) with probability $|c_{m-1}|^2$. As it will be shown in the following, this choice proves simultaneously very simple and effective in producing the correct result. However, it should be noticed that refined threshold functions can be applied once the inner product information is stored on the ancilla [23–25]. We also notice that both parallel and anti-parallel $\vec{i}$-$\vec{w}$ vectors produce an activation of the perceptron, while orthogonal vectors always result in the ancilla being measured in the state $|0\rangle_a$. This is a direct consequence of the probability being a quadratic function, i.e. $|c_{m-1}|^2$ in the present case, at difference with classical perceptrons that can only be employed as linear classifiers in their simplest realizations. In fact, our quantum perceptron model can be efficiently used as a pattern classifier, as it will be shown below, since it allows to interpret a given pattern and its negative on equivalent footing. Formally, this intrinsic symmetry reflects the invariance of the encoding $|\psi_i\rangle$ and $|\psi_w\rangle$ states under addition of a global -1 factor.

IMPLEMENTATION OF THE UNITARY TRANSFORMATIONS

One of the most critical tasks to be practically solved when implementing a quantum neural network model is the efficient implementation of unitary transformations. In machine learning applications, this might eventually discriminate between algorithms that show truly quantum advantage over their classical counterparts [12]. Here we discuss an original strategy for practically implementing the preparation of the input state $|\psi_i\rangle$ and the unitary transformation U_w on a quantum hardware. In particular, we will first outline the most straightforward algorithm one might think of employing, i.e. the "brute force" application of successive sign flip blocks. Then, we will show an alternative and more effective approach based on the generation of hypergraph states. In the next Section we will see that only the latter allows to practically implement this quantum perceptron model on a real quantum device.

So, as a first step we define a sign flip block, $\mathrm{SF}_{N,j}$, as the unitary transformation acting on the computational basis of N qubits in the following way:

$$\mathrm{SF}_{N,j}|j'\rangle = \begin{cases} |j'\rangle & \text{if } j \neq j' \\ -|j'\rangle & \text{if } j = j' \end{cases}\,. \tag{8}$$

For any $N, m = 2^N$, a controlled Z operation between N qubits ($\mathrm{C}^N\mathrm{Z}$) is a well known quantum gate [22] realizing $\mathrm{SF}_{N,m-1}$, while a single qubit Z gate acts as $\mathrm{SF}_{1,1}$. We can therefore implement in practice the whole family of sign-flip blocks for N qubits by using $\mathrm{C}^N\mathrm{Z}$ gates in combination with single qubit NOT gates (i.e. single bit flip operations):

$$\mathrm{SF}_{N,j} = \mathrm{O}_j\left(\mathrm{C}^N\mathrm{Z}\right)\mathrm{O}_j\,, \tag{9}$$

where

$$\mathrm{O}_j = \bigotimes_{l=0}^{m-1}(\mathrm{NOT}_l)^{1-j_l}\,. \tag{10}$$

In the expression above, NOT_l means that the bit flip is applied to the l-th qubit and $j_l = 0(1)$ if the l-th qubit is in state $|0\rangle(|1\rangle)$ in the computational basis state $|j\rangle$. Alternatively, the same result can also be obtained by using an extra ancillary qubit and multi-controlled NOT gates ($\mathrm{C}^N\mathrm{NOT}$), i.e. bit flip operations conditioned on the state of some control qubits. We explicitly point out that, as it is easily understood from the definition in Eq. (8), any $\mathrm{SF}_{N,j}$ is the inverse of itself. Then, the full sequence to implement U_i can be summarized as follows: starting from the initialized register $|0\rangle^{\otimes N}$, parallel Hadamard (H) gates are applied to create an equal superposition of all the elements of the computational basis:

$$|0\rangle^{\otimes N} \xrightarrow{\mathrm{H}^{\otimes N}} \frac{1}{\sqrt{m}}\sum_{j=0}^{m-1}|j\rangle \equiv |\psi_0\rangle\,, \tag{11}$$

where we remind that [22]

$$\mathrm{H}|0\rangle = \frac{|0\rangle+|1\rangle}{\sqrt{2}}\;;\;\mathrm{H}|1\rangle = \frac{|0\rangle-|1\rangle}{\sqrt{2}}\,. \tag{12}$$

Then, the $\mathrm{SF}_{N,j}$ blocks are applied one by one whenever there is a -1 factor in front of $|j\rangle$, in the representation of the target $|\psi_i\rangle$. Notice that any $\mathrm{SF}_{N,j}$ only affects a single element of the computational basis while leaving

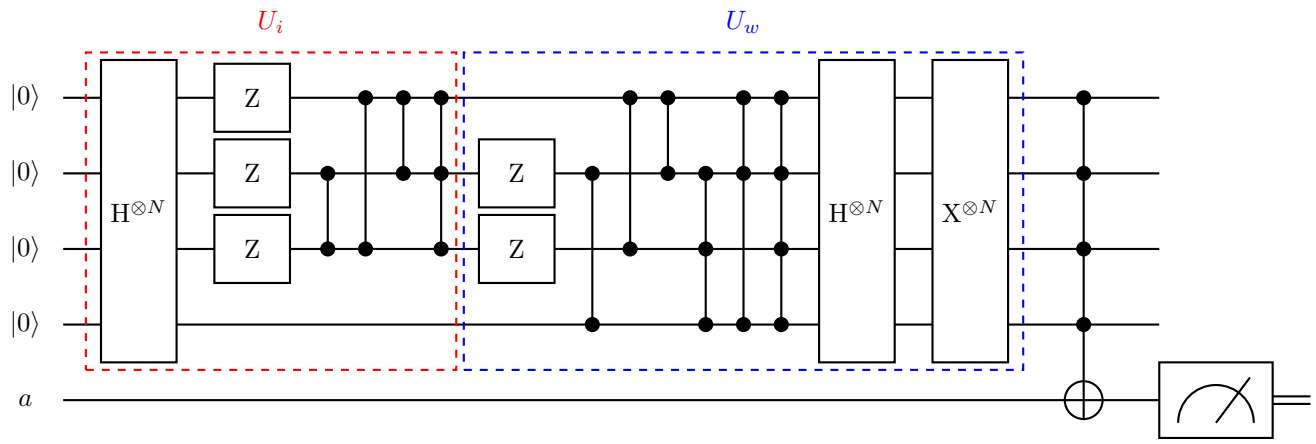


FIG. 2. **Quantum circuit of a $N = 4$ perceptron.** An example of a typical quantum circuit for a perceptron model with $N = 4$ qubits (i.e. capable of processing $m = 2^4 = 16$ dimensional input vectors), which employs the algorithm for the generation of hypergraph states, including the HSGS (see main text). In this example, the input vector has elements $i_0 = i_1 = -1$, and $i_j = 1$ for $j = 2, \ldots, 15$, while the weight vector has elements $w_2 = w_3 = w_4 = -1$, and 1 in all other entries. Multi-controlled $\mathrm{C}^p\mathrm{Z}$ gates are denoted by vertical lines and black dots on the qubits involved. The HSGS is realized inside the U_i block after the initial $\mathrm{H}^{\otimes N}$ gate, and in the U_w block before the final $\mathrm{H}^{\otimes N}$ and $\mathrm{NOT}^{\otimes N}$ operations.

all others unchanged. Moreover, all $\mathrm{SF}_{N,j}$ blocks commute with each other, so they can actually be performed in any order. As already anticipated, the whole problem is symmetric under the addition of a global -1 factor (i.e. $|\psi_i\rangle$ and $-|\psi_i\rangle$ are fully equivalent). Hence, there can be only at most $m/2 = 2^{N-1}$ independent -1 factors, and 2^{N-1} sign flip blocks are needed in the worst case. A similar strategy can also be applied to implement the other unitary operation in the quantum perceptron algorithm, U_w. Indeed, applying first the $\mathrm{SF}_{N,j}$ blocks that would be needed to flip all the -1 signs in front of the computational basis elements in the associated $|\psi_w\rangle$ leads to the balanced superposition $|\psi_w\rangle \to |\psi_0\rangle$. This quantum state can then be brought into the desired $|11\ldots11\rangle \equiv |1\rangle^{\otimes N}$ state by applying parallel Hadamard and NOT gates:

$$|\psi_0\rangle \xrightarrow{\mathrm{H}^{\otimes N}} |0\rangle^{\otimes N} \xrightarrow{\mathrm{NOT}^{\otimes N}} |1\rangle^{\otimes N} . \tag{13}$$

Evidently, the above strategy is exponentially expensive in terms of circuit depth as a function of the number of qubits, and requires an exponential number of N-controlled quantum gates.

On the other hand, a more efficient strategy can be given after realizing that the class of possible input- and weight-encoding states, Eq. (2), coincides with the set of the so called *hypergraph* states. The latter are ubiquitous ingredients of many renown quantum algorithms, and have been extensively studied and theoretically characterized [20, 26]. In particular, hypergraph states can be mapped into the vertices and hyper-edges of generalized graphs, and can be prepared by using single qubit and (multi)-controlled Z gates, with at most a single N-controlled $\mathrm{C}^N\mathrm{Z}$ and with the possibility of performing many p-controlled $\mathrm{C}^p\mathrm{Z}$ gates (involving only p qubits, with $p < N$) in parallel. After an initial $\mathrm{H}^{\otimes N}$ gate (see Eq. (11)), the algorithm takes a series of iterative steps [20] that are described below. In the following, we will refer to this portion of the algorithm to generate hypergraph states as the "hypergraph states generation subroutine" (HSGS).

First, we check whether there is any component with only one qubit in state $|1\rangle$ (i.e. of the form $|0\ldots010\ldots0\rangle$) requiring a -1 factor, in the representation of $|\psi_i\rangle$ on the computational basis. If so, the corresponding single qubit Z gate is applied by targeting the only qubit in state $|1\rangle$. Notice that this might introduce additional -1 factors in front of states with more than one qubit in state $|1\rangle$. Then, for $p = 2, \ldots, N$, we consider the components of the computational basis with exactly p qubits in state $|1\rangle$. For each of them, an additional -1 sign is introduced in front of its current amplitude (if it is needed and it was not previously introduced) by applying the corresponding $\mathrm{C}^p\mathrm{Z}$ between the p qubits in state $|1\rangle$. Similarly, if an unwanted sign is already present due to a previous step, this can be easily removed by applying the same $\mathrm{C}^p\mathrm{Z}$. Since $\mathrm{C}^p\mathrm{Z}$ acts non trivially only on the manifold with p or more qubits being in state $|1\rangle$, the signs of all the elements with a lower number of $|1\rangle$ components are left unchanged. As a consequence, when $p = N$ all the signs are the desired ones. As in the previous case, U_w can be obtained by slightly modifying the sequence of gates that would be used to generate $|\psi_w\rangle$. Indeed, one can start by first performing the HSGS tailored according to the ±1 factors in $|\psi_w\rangle$. Since all the gates involved in HSGS are the inverse of themselves and commute with each other, this step is equivalent to the unitary transformation bringing $|\psi_w\rangle$ back to the equally

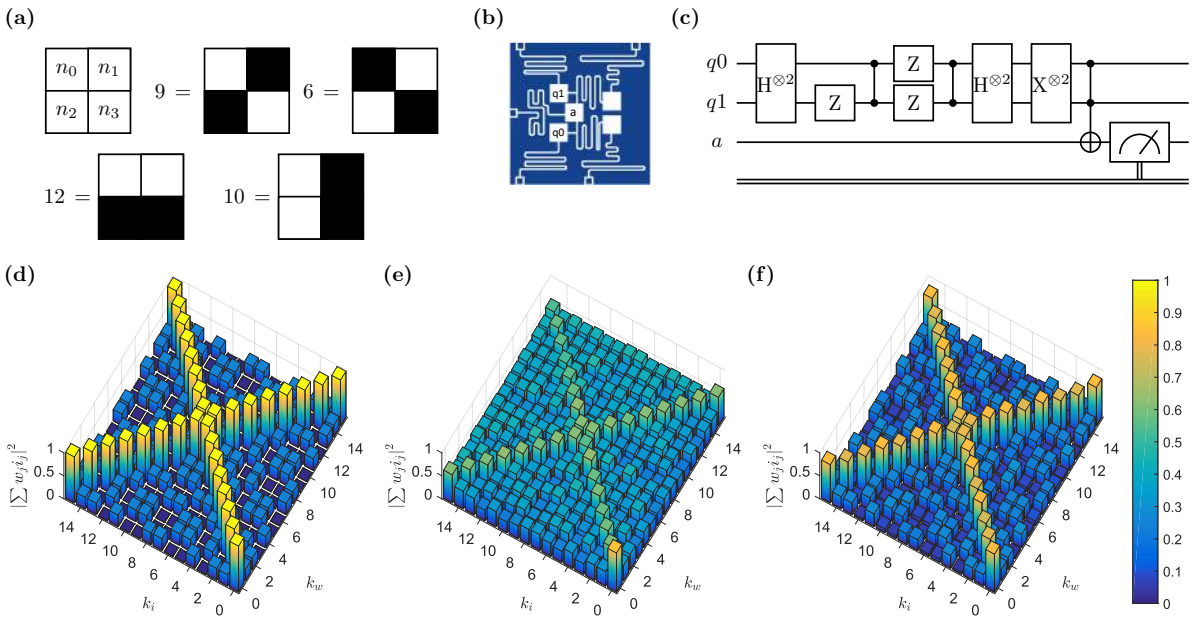


FIG. 3. **Results for $N = 2$ quantum perceptron model.** (a) Scheme used to label the 2×2 patterns and a few examples of patterns. (b) Scheme of IBM Q-5 "Tenerife" backend quantum processor. (c) Example of the gate sequence for the $N = 2$ case, with input and weight vectors corresponding to labels $k_i = 11$ and $k_w = 7$. (d) Ideal outcome of the quantum perceptron algorithm, simulated on a classical computer. (e) Results from the Tenerife processor using the algorithm with multi-controlled sign flip blocks. (f) Results from the Tenerife processor using the algorithm for the generation of hypergraph states.

balanced superposition of the computational basis states $|\psi_0\rangle$. The desired transformation U_w is finally completed by adding parallel $\mathrm{H}^{\otimes N}$ and $\mathrm{NOT}^{\otimes N}$ gates (see Eq. (13)). An example of the full sequence for a specific $N = 4$ case is shown, e.g., in Fig. 2. Notice that our optimized algorithm involving hypergraph states successfully reduces the required quantum resources with respect to a brute force approach, even if it still involves an exponential cost in terms of circuit depth or clock cycles on the quantum processor in the worst case.

Before proceeding, it is probably worth pointing out the role of U_w in this algorithm, which is essentially to cancel some of the transformations performed to prepare $|\psi_i\rangle$, or even all of them if the condition $\vec{i} = \vec{w}$ is satisfied. Further optimization of the algorithm, lying beyond the scope of the present work, might therefore be pursued at the compiling stage. However, notice that the input and weight vectors can, in practical applications, remain unknown or hidden until runtime.

NUMERICAL RESULTS AND QUANTUM SIMULATIONS

We implemented the algorithm for a single quantum perceptron both on classical simulators working out the matrix algebra of the circuit and on cloud-based quantum simulators, specifically the IBM Quantum Experience real backends [27], using the Qiskit Python development kit [28]. Due to the constraints imposed by the actual IBM hardware in terms of connectivity between the different qubits, we limited the real quantum simulation to the $N = 2$ case. Nevertheless, even this small-scale example is already sufficient to show all the distinctive features of our proposed set up, such as the exponential growth of the analyzable problems dimension, as well as the pattern recognition potential. In general, as already mentioned, in this encoding scheme N qubits can store and process 2^N-dimensional input and weight vectors, and thus 2^{2^N} different input patterns can be analyzed against the same number of the different $\vec{w}$ that are possible. Moreover, all binary inputs and weights can easily be converted into black and white patterns, thus providing a visual interpretation of the activity of the artificial neuron.

Going back to the case study with $N = 2$, $2^2 = 4$ binary images can be managed, and thus $2^{2^2} = 16$ different patterns could be analyzed. The conversion between $\vec{i}$ or $\vec{w}$ and 2×2 pixels visual patterns is done as follows. As depicted in Fig. 3a, we label each image ordering the pixels left to right, top to bottom, and assigning a value $n_j = 0(1)$ to a white (black) pixel. The corresponding input or weight vector is then built by setting $i_j = (-1)^{n_j}$ (or $w_j = (-1)^{n_j}$). We can also univocally assign an integer label k_i (or k_w) to any pattern by converting the binary string $\mathtt{n_0 n_1 n_2 n_3}$ to its corresponding decimal number representation. Under this encoding

scheme, e.g., numbers 3 and 12 are used to label patterns with horizontal lines, while 5 and 10 denote patterns with vertical lines, and 6 and 9 are used to label images with checkerboard-like pattern. An example of the sequence of operations performed on the IBM quantum computer using hypergraph states is shown in Fig. 3c for $\vec{i}$ corresponding to the index $k_i = 11$, and $\vec{w}$ corresponding to $k_w = 7$.

The Hilbert space of 2 qubits is relatively small, with a total of 16 possible values for $\vec{i}$ and $\vec{w}$. Hence, the quantum perceptron model could be experimentally tested on the IBM quantum computer for all possible combinations of input and weights. The results of these experiments, and the comparison with classical numerical simulations, are shown in Fig. 3d-f. First, we plot the ideal outcome of the quantum perceptron algorithm in Fig. 3d, where both the global -1 factor and the input-weight symmetries are immediately evident. In particular, for any given weight vector $\vec{w}$, the perceptron is able to single out from the 16 possible input patterns only $\vec{i} = \vec{w}$ and its negative (with output $|c_{m-1}|^2 = 1$, i.e. the perfect activation of the neuron), while all other inputs give outputs smaller than 0.25. If the inputs and weights are translated into 2×2 black and white pixel grids, it is not difficult to see that a single quantum perceptron can be used to recognize, e.g., vertical lines, horizontal lines, or checkerboard patterns.

The actual experimental results are then shown in Fig. 3e-f, where the same algorithm is run on the IBM Q 5 "Tenerife" quantum processor [29]. First, we show in panel 3e the results of the first, non-optimized approach introduced in the previous Section, which makes direct use of sign flip blocks. We deliberately did not take into account the global sign symmetry, thus treating any $|\psi_i\rangle$ and $-|\psi_i\rangle$ as distinct input quantum states and using up to 2^N sign flip blocks. We notice that even in such an elementary example the algorithm performs worse and worse with increasing number of blocks. Notice, however, that despite the quantitative inaccuracy of the quantum simulated outputs, the underlying structure of the output is already quite clear.

On the other hand, a remarkably better accuracy, also on the quantitative side and with small errors, is obtained when using the algorithm based on the hypergraph states formalism, whose experimental results are shown in panel 3f and represent the main result of this work. In this case, the global phase symmetry is naturally embedded in the algorithm itself, and the results show symmetric performances all over the range of possible inputs and weights. All combinations of $\vec{i}$ and $\vec{w}$ yield results either larger than 0.75 or smaller than 0.3, in very good quantitative agreement with the expected results plotted in panel 3d. As a technical warning, we finally notice that in all of the three cases shown in panels d-f of Fig. 3, the C^pZ operations were obtained by adding single qubit Hadamard gates on the target qubit before and after the corresponding C^pNOT gate. For $p = 1$ this is a CNOT gate, which is natively implemented on the IBM quantum hardware, while the case $p = 2$ is known as the Toffoli gate, for which a standard decomposition into 6 CNOTs and single qubit rotations is known [22].

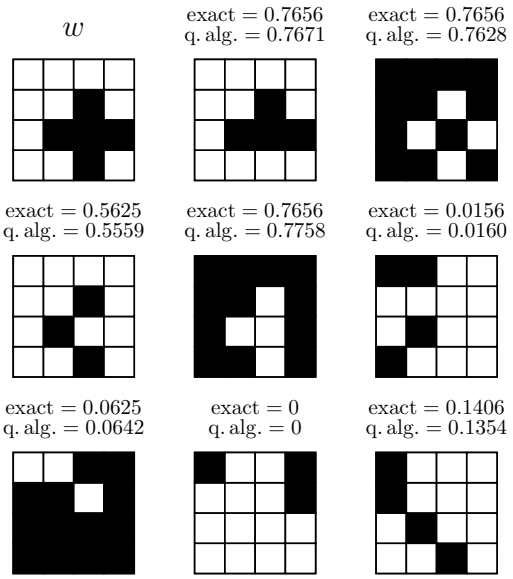


FIG. 4. **Pattern recognition for $N = 4$.** A possible choice of the weight vector for the $N = 4$ case is represented in the first panel (top left), and a small selection of different input vectors are then simulated with the quantum perceptron model. Above each input pattern, the quantitative answers of the artificial neuron are reported, as obtained either through standard linear algebra (ideal results) or resulting from the simulation of the quantum algorithm (run on a classical computer).

Finally, in the spirit of showing the potential scalability and usefulness of this quantum perceptron model for classification purposes, we have applied the algorithm to the $N = 4$ qubits case by using the circuit simulator feature available in Qiskit [30]. Now, there are a total 2^{32} possible combinations of $\vec{i}$ and $\vec{w}$ vectors, far too many to explore the whole combinatorial space as previously done for the 2 qubits in Fig. 3. To explicitly show a few examples, we have chosen a single weight vector corresponding to a simple cross-shaped pattern when represented as a 4×4 pixels image (encoded along the same lines of the $N = 2$ case, see first panel in Fig. 3), and weighted it against several possible choices of input vectors. Some results are reported in Fig. 4 for a selected choice of input vectors, where the artificial neuron output is computed both with standard linear algebra and simulated with a quantum circuit on a virtual quantum simulator run on a classical computer. Evidently, there is an overall excellent agreement when comparing the two values for each pattern, within statistical inaccuracy due to the finite number ($n_{shots} = 8192$) of repetitions imposed

on the quantum sequence used to estimate the probability $|c_{m-1}|^2$. The perceptron is able to discriminate the weight pattern (and its negative) giving an output larger than 0.5 for all images that differ from the weight or its negative by 2 bits or less.

CONCLUSIONS AND DISCUSSION

In summary, we have proposed a model for perceptrons to be directly implemented on near-term quantum processing devices, and we have experimentally tested it on a 5-qubits IBM quantum computer based on superconducting technology. Our algorithm presents an exponential advantage over classical perceptron models, as we have explicitly shown by representing and classifying 4 bits strings using 2 qubits, and 16 bits strings using only 4 qubits.

The problem of exponential advantage requires a separate discussion. In principle, generic quantum states or unitary transformations require an exponentially large number of elementary gates to be implemented, and this could somehow hinder the effective advantages brought by quantum computers for machine learning applications. This currently represents a general problem of most quantum machine learning algorithms. Moreover, with increasing N, severe issues can arise from the practical necessity to decompose multiply controlled operations by only using single- and two-qubit gates [31, 32]. However, this limitation strictly depends on the effective constraints imposed by the given quantum processor and on the required degree of accuracy. In fact, it has been shown that several classes of quantum states can be approximated efficiently with arbitrary precision, with oracle based approaches [33–35] or by using a number of multi-controlled rotations that is linear with the number of qubits [36]. Using these results, it might then be possible to design a version of our proposed quantum perceptron algorithm working with approximated encoding quantum states instead of exact ones, which would have the potential to scale exponentially better than any classical algorithm implementing a perceptron model. In this respect, it is also worth pointing out that our procedure is fully general and could be implemented and run on any platform capable of performing universal quantum computation. While we have employed a quantum hardware that is based on superconducting technology and qubits, a very promising alternative is the trapped-ion based quantum computer [37], in which multi-qubit entangling gates might be readily available [38, 39].

As a further strategy for future developments, we notice that in the present work we restricted the whole analysis to binary inputs and weight vectors (the so called "McCollough-Pitts" neuron model), mainly for clarity and simplicity of implementation. A possible improvement for the algorithm presented is obviously to encode continuously valued vectors (equivalent to grey scale images). This could be achieved by using continuously valued phase factors in $|\psi_i\rangle$ and $|\psi_w\rangle$ [9]. Finally, a potentially very exciting continuation of this work would be to connect multiple layers of our quantum perceptrons to build a feedforward deep neural network, which could be fully run on dedicated quantum hardware. In such a network, each neuron could use two ancilla qubits, one to be measured to introduce the nonlinearity as done in this work, while the second would be used to propagate the information from each neuron to the successive layer in the network in a fully quantum coherent way. As such, our work thus constitute a concrete first step towards an actual application of near-term (i.e., with few tens of non-error corrected qubits) quantum processors to be employed as fast and efficient trained artificial quantum neural networks.

AKNOWLEDGEMENTS

We acknowledge the University of Pavia Blue Sky Research project number BSR1732907. This research was also supported by the Italian Ministry of Education, University and Research (MIUR): "Dipartimenti di Eccellenza Program (2018-2022)", Department of Physics, University of Pavia. We acknowledge use of the IBM Quantum Experience for this work. The views expressed are those of the authors and do not reflect the official policy or position of IBM company or the IBM-Q team.

* francesco.tacchino01@ateneopv.it
† chiara.macchiavello@unipv.it
‡ dario.gerace@unipv.it
§ daniele.bajoni@unipv.it

[1] J. Schmidhuber, *Deep learning in neural networks: An overview*, Neural Networks **61**, 85–117 (2015).
[2] J. M. Zurada, *Introduction to Artificial Neural Systems*, (West Group, 1992).
[3] R. Rojas, *Neural Networks: A Systematic Introduction*, (Springer, 1996).
[4] C. D. Schuman, T. E. Potok, R. M. Patton, J. D. Birdwell, M. E. Dean, G. S. Rose and J. S. Plank, *A Survey of Neuromorphic Computing and Neural Networks in Hardware*, arXiv:1705.06963 (2017).
[5] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, B. Brezzo, I. Vo, S. K. Esser, R. Appuswamy, B. Taba, A. Amir, M. D. Flickner, W. P. Risk, R. Manohar and D. S. Modha, *A million spiking-neuron integrated circuit with a scalable communication network and interface*, Science **345**, 668–673 (2014).
[6] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe and S. Lloyd, *Quantum machine learning*, Nature **549**, 195–202 (2017).

[7] F. Neukart and S.-A. Moraru, *On Quantum Computers and Artificial Neural Networks*, Journal of Signal Processing Research **2** (2013).

[8] M. Schuld, I. Sinayskiy and F. Petruccione, *The quest for a Quantum Neural Network*, Quantum Information Processing **13**, 2567–2586 (2014).

[9] M. Schuld, I. Sinayskiy and F. Petruccione, *Simulating a perceptron on a quantum computer*, Physics Letters A **7**, 660–663 (2015).

[10] A. Kapoor, N. Wiebe and K. Svore, *Quantum Perceptron Models*, Advances In Neural Information Processing Systems (NIPS 2016) **29**, 3999–4007 (2016).

[11] S. Lloyd, M. Mohseni and P. Rebentrost, *Quantum algorithms for supervised and unsupervised machine learning*, arXiv:1307.0411 (2013).

[12] M. Schuld, M. Fingerhuth and F. Petruccione, *Implementing a distance-based classifier with a quantum interference circuit*, Europhysics Letters **119**, 6002 (2017).

[13] L. Lamata, *Basic protocols in quantum reinforcement learning with superconducting circuits*, Scientific Reports **7**, 1609 (2017).

[14] U. Alvarez-Rodriguez, L. Lamata, P. Escandell-Montero, J. D. Martín-Guerrero and E. Solano *Supervised Quantum Learning without Measurements*, Scientific Reports **7**, 13645 (2017).

[15] J. S. Otterbach, R. Manenti, N. Alidoust, A. Bestwick, M. Block, B. Bloom, S. Caldwell, N. Didier, E. Fried Schuyler, S. Hong, P. Karalekas, C. B. Osborn, A. Papageorge, E. C. Peterson, G. Prawiroatmodjo, N. Rubin, C. A. Ryan, D. Scarabelli, M. Scheer, E. A. Sete, P. Sivarajah, R. S. Smith, A. Staley, N. Tezak, W. J. Zeng, A. Hudson, B. R. Johnson, M. Reagor, M. P. Silva and C. Rigetti, *Unsupervised Machine Learning on a Hybrid Quantum Computer*, arXiv:1712.05771 (2017).

[16] P. Rebentrost, T. R. Bromley, C. Weedbrook and S. Lloyd, *Quantum Hopfield neural network*, Phys. Rev. A **98**, 042308 (2018).

[17] F. Rosenblatt, *The Perceptron: A perceiving and recognizing automaton*, Tech. Rep. Inc. Report No. 85-460-1 (Cornell Aeronautical Laboratory, 1957).

[18] W. S. McCulloch and W. Pitts, *A logical calculus of the ideas immanent in nervous activity*, The bulletin of mathematical biophysics **5**, 115–133 (1943).

[19] D. C. Mocanu, E. Mocanu, P. Stone, P. H. Nguyen, M. Gibescu and A. Liotta, *Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science*, Nature Communications **9**, 2383 (2018).

[20] M. Rossi, M. Huber, D. Bruß and C. Macchiavello, *Quantum hypergraph states*, New Journal of Physics **15**, 113022 (2013).

[21] V. Giovannetti, S. Lloyd, Seth and L. Maccone, *Quantum Random Access Memory*, Phys. Rev. Lett. **100**, 160501 (2008).

[22] M. A. Nielsen, and I. L. Chuang, *Quantum Computation and Quantum Information (Cambridge Series on Information and the Natural Sciences)*, (Cambridge University Press, 2004).

[23] W. Hu, *Towards a Real Quantum Neuron*, Natural Science **10**, 99–109 (2018).

[24] Y. Cao, G. G. Guerreschi and A. Aspuru-Guzik, *Quantum Neuron: an elementary building block for machine learning on quantum computers*, arXiv:1711.11240 (2017).

[25] E. Torrontegui and J. J. Garcia-Ripoll, *Universal quantum perceptron as efficient unitary approximators*, arXiv:1801.00934 (2018).

[26] M. Ghio, D. Malpetti, M. Rossi, D. Bruß and C. Macchiavello, *Multipartite entanglement detection for hypergraph states*, J. Phys. A: Math. Theor. **51**, 045302 (2018).

[27] See https://quantumexperience.ng.bluemix.net

[28] See https://qiskit.org/

[29] For hardware specifications and images, see IBM Q team, *IBM Q 5 Tenerife backend specification V1.2.0 and V1.3.0*, (2018), retrieved from https://ibm.biz/qiskit-tenerife.

[30] See https://qiskit.org/ and https://qiskit.org/terra

[31] V. Bergholm, J. J. Vartiainen, M. Möttönen and M. M. Salomaa, *Quantum circuits with uniformly controlled one-qubit gates* Phys. Rev. A **71**, 052330 (2005).

[32] M. Plesch and Č. Brukner, *Quantum-state preparation with universal gate decompositions*, Phys. Rev. A **83**, 032302 (2011).

[33] L. Grover and T. Rudolph, *Creating superpositions that correspond to efficiently integrable probability distributions*, arXiv:0208112 [quant-ph] (2002).

[34] A. N. Soklakov and R. Schack, *Efficient state preparation for a register of quantum bits*, Phys. Rev. A **73**, 012307 (2006).

[35] B. D. Clader, B. C. Jacobs and C. R. Sprouse, *Preconditioned quantum linear system algorithm*, Phys. Rev. Lett. **110**, 250504 (2013).

[36] M. Mosca and P. Kaye *Quantum Networks for Generating Arbitrary Quantum States*, Optical Fiber Communication Conference and International Conference on Quantum Information, PB28 (Optical Society of America, 2001).

[37] P. Schindler, D. Nigg, T. Monz, J. T. Barreiro, E. Martinez, S. X. Wang, S. Quint, M. F. Brandl, V. Nebendahl, C. F. Roos, M. Chwalla, M. Hennrich and R. Blatt, *A quantum information processor with trapped ions*, New Journal of Physics **15**, 123012 (2013).

[38] K. Mølmer and A. Sørensen, *Multiparticle Entanglement of Hot Trapped Ions*, Phys. Rev. Lett. **82**, 1835–1838 (1999).

[39] P. Schindler, M. Müller, D. Nigg, J. T. Barreiro, E. A. Martinez, M. Hennrich, T. Monz, S. Diehl, P. Zoller and R. Blatt, *Quantum simulation of dynamical maps with trapped ions*, Nature Physics **9**, 361–367 (2013).