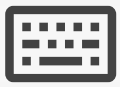
```
 Step 1: Install/Upgrade Qiskit & C
pip install --upgrade qiskit qiskit

 Step 2: Imports
rom qiskit import QuantumCircuit, t
rom qiskit_aer import Aer
rom qiskit.primitives import Sample
mport matplotlib.pyplot as plt
mport numpy as np

 Step 3: Build the Circuit (Harmoni
ef apply_qft(qc, n_qubits):
   """Quantum Fourier Transform on
   for j in range(n_qubits):
       qc.h(j)
       for k in range(j+1, n_qubits
           angle = np.pi / (2**(k-j
           qc.cp(angle, k, j)
   # Reverse qubit order
   for i in range(n_qubits // 2):
       qc.swap(i, n_qubits - i - 1)

ef build_harmonic_qc(num_qubits=3):
   """Create a quantum circuit that
   qc = QuantumCircuit(num_qubits,
   # 1) Hadamard to create superpos
   for q in range(num_qubits):
       qc.h(q)
   # 2) Some entanglement
   for q in range(num_qubits-1):
       qc.cx(q, q+1)
   # 3) Random phases for variety
   for q in range(num_qubits):
       angle = np.random.uniform(0,
       qc.rz(angle, q)
   # 4) QFT
   apply_qft(qc, num_qubits)
   # 5) Measure
   qc.measure(range(num_qubits), ra
   return qc

 Step 4: Run using Sampler
```

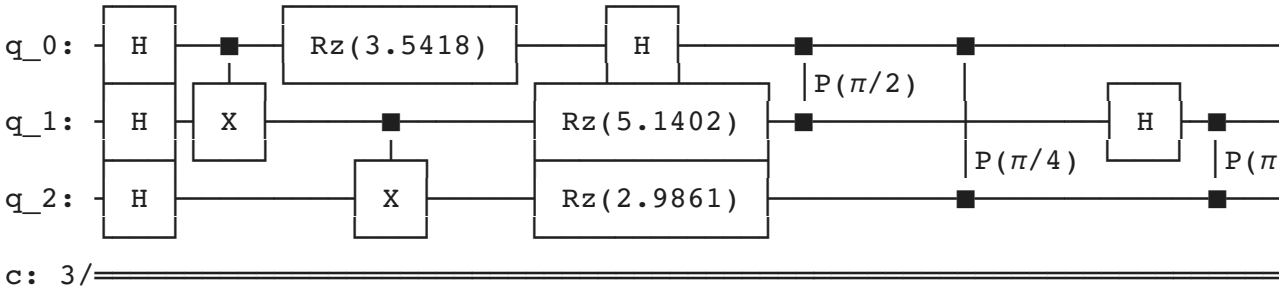
```
 Step 4: Run using Sampler
ef run_qc_sampler(qc, shots=1024):
   """Use Qiskit Sampler to run the
   sampler = Sampler()
   # Sampler returns quasi-probabil
   job = sampler.run([qc], shots=sh
   result = job.result()
   quasi = result.quasi_dists[0]  #
   counts = {}
   for state_int, prob in quasi.ite
       state_bin = format(state_int
       c = int(prob * shots)
       counts[state_bin] = counts.g
   return counts

 Step 5: Putting it all together ir
um_qubits = 3
hots = 1024

c = build_harmonic_qc(num_qubits)
ounts = run_qc_sampler(qc, shots=sh

 Print & Plot
rint("Quantum Circuit:")
rint(qc)
rint("\nCounts:", counts)

lt.bar(counts.keys(), counts.values
lt.xlabel("State")
lt.ylabel("Counts")
lt.title("Harmonic + QFT with randc
lt.show()
```

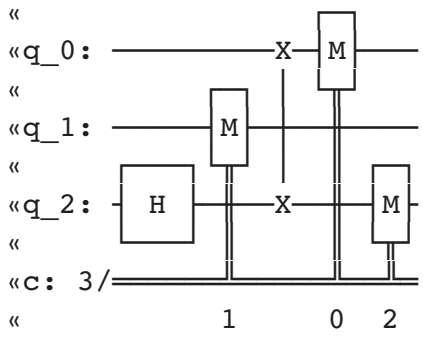


```
Collecting qiskit
  Downloading qiskit-1.4.0-cp39-abi3-manylinux_2_17_x86_64.manylinux201
Collecting qiskit-aer
  Downloading qiskit_aer-0.16.2-cp311-cp311-manylinux_2_17_x86_64.manyl
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/
Collecting matplotlib
  Downloading matplotlib-3.10.1-cp311-cp311-manylinux_2_17_x86_64.manyl
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-
Collecting numpy
```

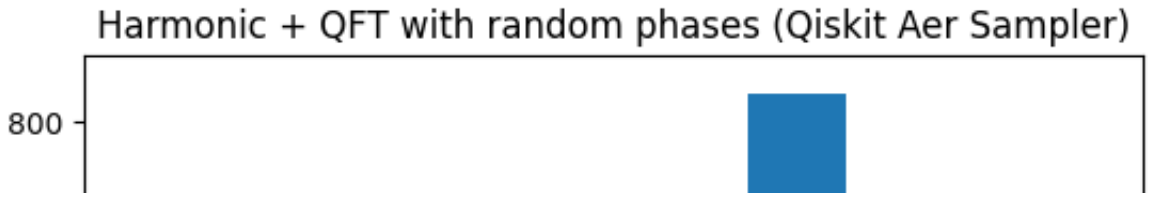
```
  Downloading numpy-2.2.3-cp311-cp311-manylinux_2_17_x86_64.manylinux20
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.0/62.0 kB 1.4 MB/s eta
Collecting rustworkx>=0.15.0 (from qiskit)
  Downloading rustworkx-0.16.0-cp39-abi3-manylinux_2_17_x86_64.manylinu
Requirement already satisfied: scipy>=1.5 in /usr/local/lib/python3.11/
Requirement already satisfied: sympy>=1.3 in /usr/local/lib/python3.11/
Collecting dill>=0.3 (from qiskit)
  Downloading dill-0.3.9-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: python-dateutil>=2.8.0 in /usr/local/lib
Collecting stevedore>=3.0.0 (from qiskit)
  Downloading stevedore-5.4.1-py3-none-any.whl.metadata (2.3 kB)
Requirement already satisfied: typing-extensions in /usr/local/lib/pyth
Collecting symengine<0.14,>=0.11 (from qiskit)
  Downloading symengine-0.13.0-cp311-cp311-manylinux_2_17_x86_64.manyli
Requirement already satisfied: psutil>=5 in /usr/local/lib/python3.11/d
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/pytho
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.1
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/pyth
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/pyth
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/d
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/pytho
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/di
Collecting pbr>=2.0.0 (from stevedore>=3.0.0->qiskit)
  Downloading pbr-6.1.1-py2.py3-none-any.whl.metadata (3.4 kB)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/pyt
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/
Downloading qiskit-1.4.0-cp39-abi3-manylinux_2_17_x86_64.manylinux2014_
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6.7/6.7 MB 44.0 MB/s eta 0:
Downloading qiskit_aer-0.16.2-cp311-cp311-manylinux_2_17_x86_64.manylin
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.4/12.4 MB 63.8 MB/s eta
Downloading matplotlib-3.10.1-cp311-cp311-manylinux_2_17_x86_64.manylin
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8.6/8.6 MB 31.8 MB/s eta 0:
Downloading numpy-2.2.3-cp311-cp311-manylinux_2_17_x86_64.manylinux2014
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 16.4/16.4 MB 45.8 MB/s eta
Downloading dill-0.3.9-py3-none-any.whl (119 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 119.4/119.4 kB 8.5 MB/s eta
Downloading rustworkx-0.16.0-cp39-abi3-manylinux_2_17_x86_64.manylinux2
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.1/2.1 MB 58.8 MB/s eta 0:
Downloading stevedore-5.4.1-py3-none-any.whl (49 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 49.5/49.5 kB 2.6 MB/s eta 0
Downloading symengine-0.13.0-cp311-cp311-manylinux_2_17_x86_64.manylinu
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 49.7/49.7 MB 9.5 MB/s eta 0
Downloading pbr-6.1.1-py2.py3-none-any.whl (108 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 109.0/109.0 kB 7.3 MB/s eta
Installing collected packages: symengine, pbr, numpy, dill, stevedore,
```

```
  Attempting uninstall: numpy
    Found existing installation: numpy 1.26.4
    Uninstalling numpy-1.26.4:
      Successfully uninstalled numpy-1.26.4
  Attempting uninstall: matplotlib
    Found existing installation: matplotlib 3.10.0
    Uninstalling matplotlib-3.10.0:
      Successfully uninstalled matplotlib-3.10.0
ERROR: pip's dependency resolver does not currently take into account a
gensim 4.3.3 requires numpy<2.0,>=1.18.5, but you have numpy 2.2.3 whic
langchain 0.3.15 requires numpy<2,>=1.22.4; python_version < "3.12", bu
tensorflow 2.17.1 requires numpy<2.0.0,>=1.23.5; python_version <= "3.1
thinc 8.2.5 requires numpy<2.0.0,>=1.19.0; python_version >= "3.9", but
pytensor 2.26.4 requires numpy<2,>=1.17.0, but you have numpy 2.2.3 whi
numba 0.60.0 requires numpy<2.1,>=1.22, but you have numpy 2.2.3 which
cupy-cuda12x 12.2.0 requires numpy<1.27,>=1.20, but you have numpy 2.2.
Successfully installed dill-0.3.9 matplotlib-3.10.1 numpy-2.2.3 pbr-6.1
Quantum Circuit:
```





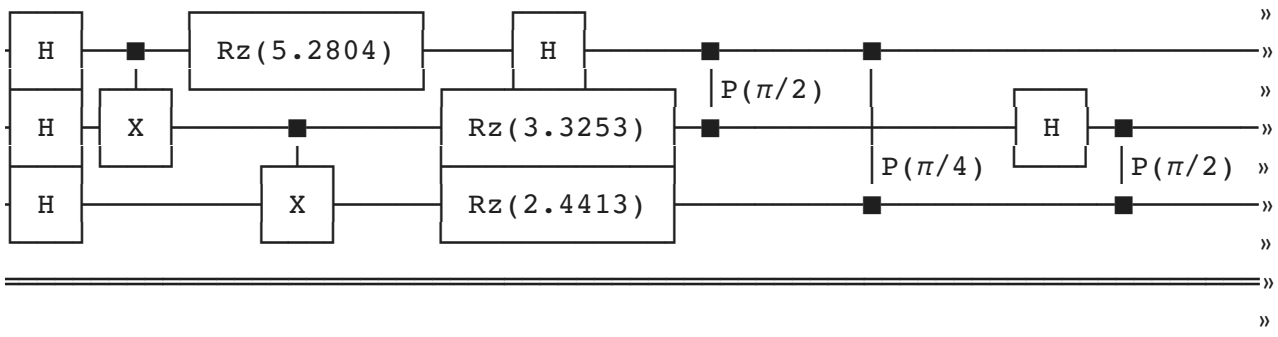

```
Counts: {'000': 1, '001': 36, '010': 3, '011': 9, '100': 94, '101': 832
<ipython-input-1-bea82d77f2b1>:45: DeprecationWarning: The class ``qisk
  sampler = Sampler()
```

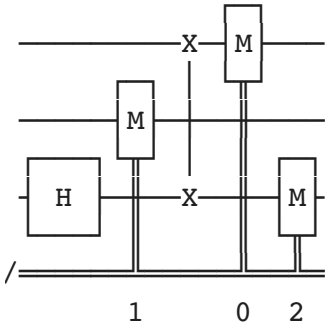



```
# Step 1: Install/Upgrade Qiskit & Qiskit Aer
!pip install --upgrade qiskit qiskit-aer matplotlib numpy

# Step 2: Imports
from qiskit import QuantumCircuit, transpile
from qiskit_aer import Aer
from qiskit.primitives import Sampler
import matplotlib.pyplot as plt
import numpy as np

# Step 3: Build the Circuit (Harmonic + QFT)
def apply_qft(qc, n_qubits):
    """Quantum Fourier Transform on n_qubits."""
    for j in range(n_qubits):
        qc.h(j)
        for k in range(j+1, n_qubits):
            angle = np.pi / (2**(k-j))
            qc.cp(angle, k, j)
    # Reverse qubit order
    for i in range(n_qubits // 2):
        qc.swap(i, n_qubits - i - 1)

def build_harmonic_qc(num_qubits=3):
    """Create a quantum circuit that simulates a 'harmonic oscillator' styl
    qc = QuantumCircuit(num_qubits, num_qubits)
    # 1) Hadamard to create superpositions
    for q in range(num_qubits):
        qc.h(q)
    # 2) Some entanglement
    for q in range(num_qubits-1):
        qc.cx(q, q+1)
    # 3) Random phases for variety
```

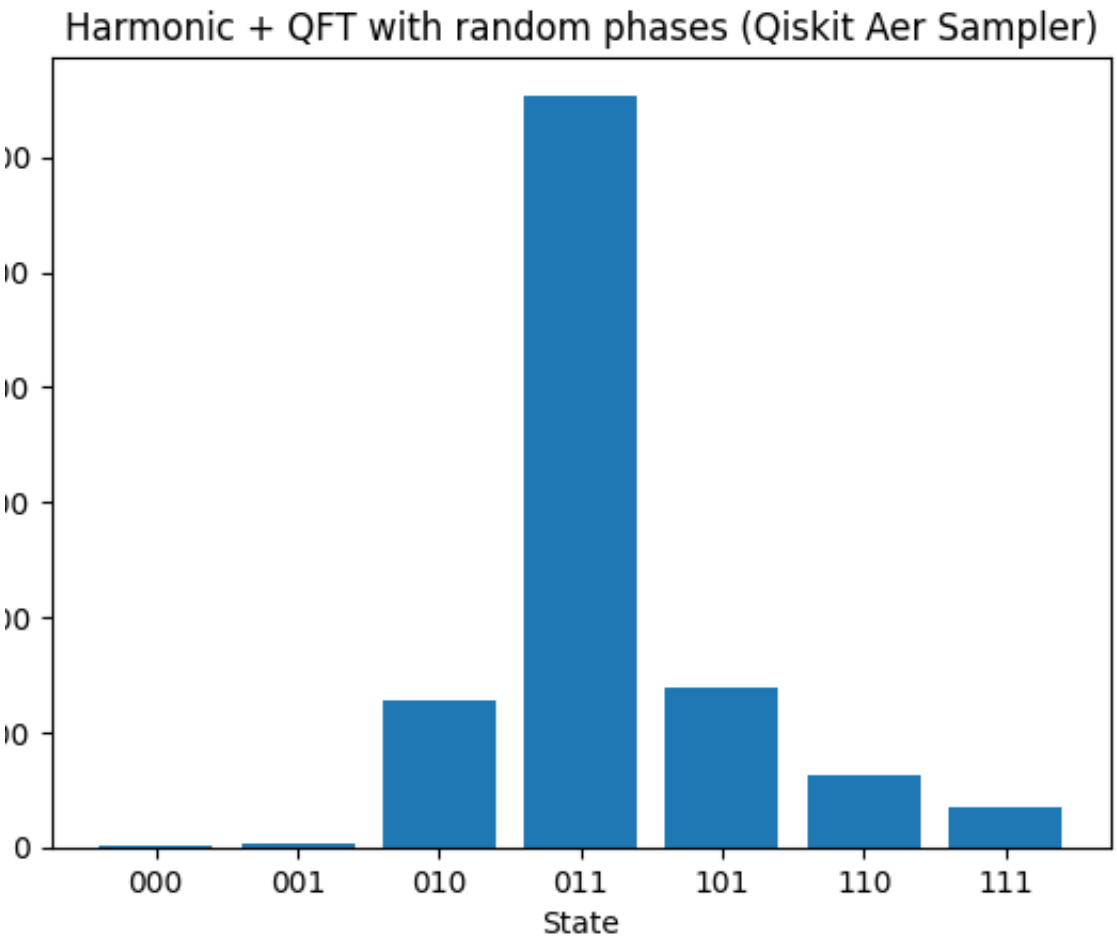
```
    for q in range(num_qubits):
        angle = np.random.uniform(0, 2*np.pi)
        qc.rz(angle, q)
    # 4) QFT
    apply_qft(qc, num_qubits)
    # 5) Measure
    qc.measure(range(num_qubits), range(num_qubits))
    return qc

# Step 4: Run using Sampler
def run_qc_sampler(qc, shots=1024):
    """Use Qiskit Sampler to run the circuit and get 'counts'-like results.
    sampler = Sampler()
    # Sampler returns quasi-probabilities. We'll multiply by 'shots' for ap
    job = sampler.run([qc], shots=shots)
    result = job.result()
    quasi = result.quasi_dists[0]  # quasi-probs for the single circuit
    counts = {}
    for state_int, prob in quasi.items():
        state_bin = format(state_int, f"0{qc.num_qubits}b")
        c = int(prob * shots)
        counts[state_bin] = counts.get(state_bin, 0) + c
    return counts

# Step 5: Putting it all together in main
num_qubits = 3
shots = 1024

qc = build_harmonic_qc(num_qubits)
counts = run_qc_sampler(qc, shots=shots)

# Print & Plot
print("Quantum Circuit:")
print(qc)
print("\nCounts:", counts)

plt.bar(counts.keys(), counts.values())
plt.xlabel("State")
plt.ylabel("Counts")
plt.title("Harmonic + QFT with random phases (Qiskit Aer Sampler)")
plt.show()
```

```
rement already satisfied: qiskit in /usr/local/lib/python3.11/dist-packa
rement already satisfied: qiskit-aer in /usr/local/lib/python3.11/dist-p
rement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-p
rement already satisfied: numpy in /usr/local/lib/python3.11/dist-packag
rement already satisfied: rustworkx>=0.15.0 in /usr/local/lib/python3.11
rement already satisfied: scipy>=1.5 in /usr/local/lib/python3.11/dist-p
rement already satisfied: sympy>=1.3 in /usr/local/lib/python3.11/dist-p
rement already satisfied: dill>=0.3 in /usr/local/lib/python3.11/dist-pa
rement already satisfied: python-dateutil>=2.8.0 in /usr/local/lib/pytho
rement already satisfied: stevedore>=3.0.0 in /usr/local/lib/python3.11/
rement already satisfied: typing-extensions in /usr/local/lib/python3.11
rement already satisfied: symengine<0.14,>=0.11 in /usr/local/lib/python
rement already satisfied: psutil>=5 in /usr/local/lib/python3.11/dist-pa
rement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/
rement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist
rement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11
rement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11
rement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/d
rement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-pa
rement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/
rement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-pac
rement already satisfied: pbr>=2.0.0 in /usr/local/lib/python3.11/dist-p
rement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.1
rement already satisfied: setuptools in /usr/local/lib/python3.11/dist-p
hon-input-4-362231258ef4>:45: DeprecationWarning: The class ``qiskit.pri
pler = Sampler()
um Circuit:
```





```
s: {'000': 2, '001': 4, '010': 128, '011': 654, '101': 139, '110': 62, '
```

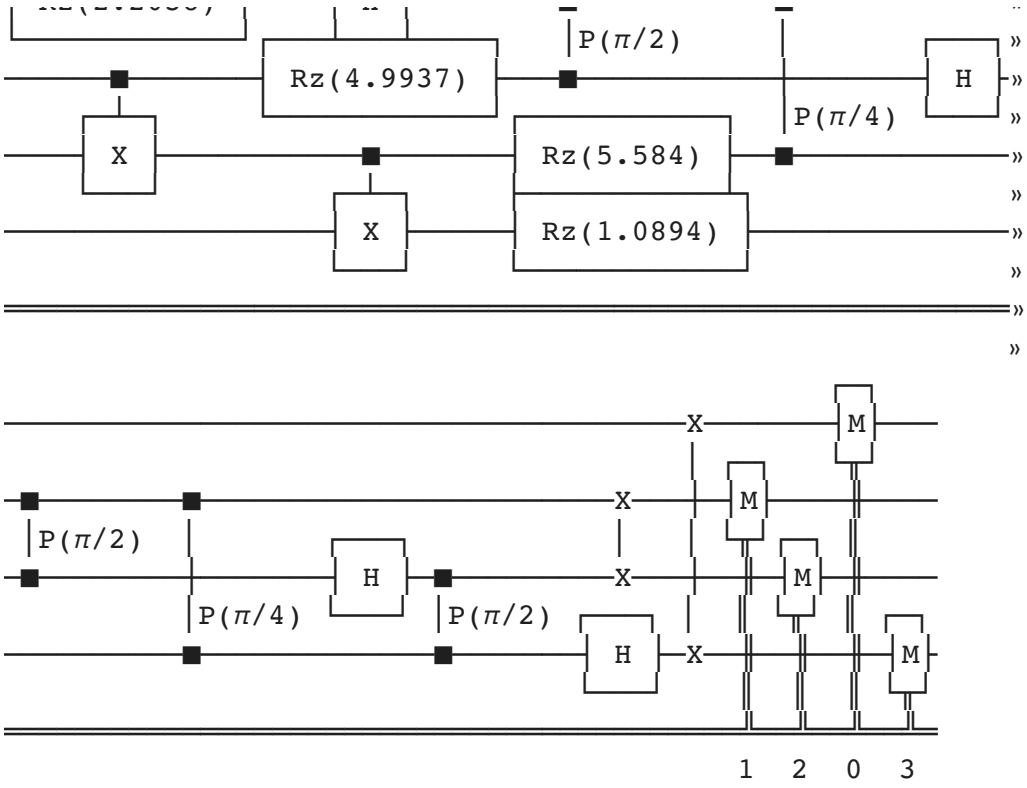



```
# Step 1: Install/Upgrade Qiskit & Qiskit Aer
!pip install --upgrade qiskit qiskit-aer matplotlib numpy

# Step 2: Imports
from qiskit import QuantumCircuit
from qiskit_aer import Aer
from qiskit.primitives import Sampler
import numpy as np
import matplotlib.pyplot as plt

# Step 3: QFT Function
def apply_qft(qc, n_qubits):
    """Quantum Fourier Transform on n_qubits."""
    for j in range(n_qubits):
        qc.h(j)
```

```
        for k in range(j+1, n_qubits):
            angle = np.pi / (2**(k-j))
            qc.cp(angle, k, j)
    # Reverse qubit order
    for i in range(n_qubits // 2):
        qc.swap(i, n_qubits - i - 1)

# Step 4: Build Circuit with 4 Qubits
def build_harmonic_qc(num_qubits=4):
    qc = QuantumCircuit(num_qubits, num_qubits)
    # 1) Hadamard gates to create superposition
    for q in range(num_qubits):
        qc.h(q)
    # 2) Entanglement (CNOT chain)
    for q in range(num_qubits - 1):
        qc.cx(q, q + 1)
    # 3) Random RZ phases
    for q in range(num_qubits):
        angle = np.random.uniform(0, 2 * np.pi)
        qc.rz(angle, q)
    # 4) QFT
    apply_qft(qc, num_qubits)
    # 5) Measure
    qc.measure(range(num_qubits), range(num_qubits))
    return qc

# Step 5: Sampler Execution
def run_qc_sampler(qc, shots=1024):
    sampler = Sampler()
    job = sampler.run([qc], shots=shots)
    result = job.result()
    quasi = result.quasi_dists[0]  # quasi-probs for this circuit
    counts = {}
    for state_int, prob in quasi.items():
        bin_state = format(state_int, f"0{qc.num_qubits}b")
        c = int(prob * shots)
        counts[bin_state] = counts.get(bin_state, 0) + c
    return counts

# Step 6: Putting it all together with 4 qubits
num_qubits = 4
shots = 2048
```

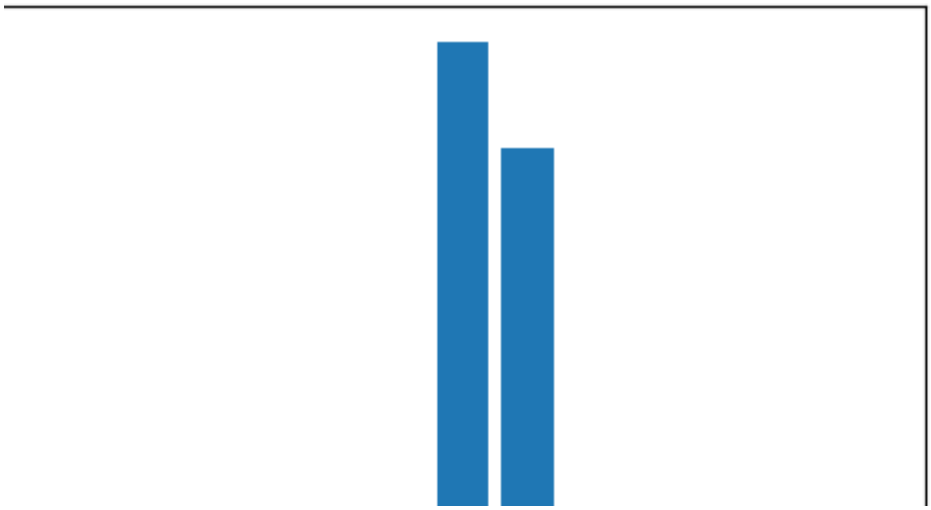
```
qc = build_harmonic_qc(num_qubits)
counts = run_qc_sampler(qc, shots=shots)

print("Quantum Circuit:")
print(qc)
print("\nCounts:", counts)

# Plot the results
plt.bar(counts.keys(), counts.values())
plt.xlabel("State")
plt.ylabel("Counts")
plt.xticks(rotation=90)
plt.title("4-Qubit Harmonic + QFT with Random Phases (Qiskit Aer Sampler)")
plt.show()
```

```
eady satisfied: qiskit in /usr/local/lib/python3.11/dist-packages (1.4.0
eady satisfied: qiskit-aer in /usr/local/lib/python3.11/dist-packages (0
eady satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (3
eady satisfied: numpy in /usr/local/lib/python3.11/dist-packages (2.2.3)
eady satisfied: rustworkx>=0.15.0 in /usr/local/lib/python3.11/dist-pack
eady satisfied: scipy>=1.5 in /usr/local/lib/python3.11/dist-packages (f
eady satisfied: sympy>=1.3 in /usr/local/lib/python3.11/dist-packages (f
eady satisfied: dill>=0.3 in /usr/local/lib/python3.11/dist-packages (fr
eady satisfied: python-dateutil>=2.8.0 in /usr/local/lib/python3.11/dist
eady satisfied: stevedore>=3.0.0 in /usr/local/lib/python3.11/dist-packa
eady satisfied: typing-extensions in /usr/local/lib/python3.11/dist-pack
eady satisfied: symengine<0.14,>=0.11 in /usr/local/lib/python3.11/dist-
eady satisfied: psutil>=5 in /usr/local/lib/python3.11/dist-packages (fr
eady satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packa
eady satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages
eady satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-pack
eady satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-pack
eady satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packag
eady satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (fr
eady satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packa
eady satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (fro
eady satisfied: pbr>=2.0.0 in /usr/local/lib/python3.11/dist-packages (f
eady satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.11/dist-pac
eady satisfied: setuptools in /usr/local/lib/python3.11/dist-packages (f
5-5b00e8b71987>:44: DeprecationWarning: The class ``qiskit.primitives.sa
pler()
:
┌─────────────┐┌───┐                                                  »
┤ Rz(2.2038) ├┤ H ├──────────■─────────────────■──────────────────────»
```



: 181, '0001': 79, '0010': 2, '0011': 30, '0100': 45, '0101': 97, '0110'

rmonic + QFT with Random Phases (Qiskit Aer Sampler)



```
from qiskit import QuantumCircuit
from qiskit.primitives import Sampler
from qiskit.visualization import plot_histogram
import numpy as np
import matplotlib.pyplot as plt
```

```
# Define 5-qubit Quantum Circuit
qc = QuantumCircuit(5, 5)

# Apply Hadamard gates to create superposition
qc.h(range(5))

# Apply random phase rotations
for qubit in range(5):
    random_angle = np.random.uniform(0, 2 * np.pi)
    qc.rz(random_angle, qubit)

# Apply entangling gates (CNOT)
for i in range(4):
    qc.cx(i, i + 1)

# Apply Quantum Fourier Transform (QFT) approximation
for i in range(5):
    qc.h(i)
    for j in range(i + 1, 5):
        qc.cp(np.pi / 2**(j - i), j, i)

# Swap qubits for QFT output
for i in range(5 // 2):
    qc.swap(i, 5 - i - 1)

# Measure
qc.measure(range(5), range(5))

# Display the circuit
print("5-Qubit Harmonic + QFT Quantum Circuit:")
display(qc.draw('mpl'))

# Execute using Qiskit Sampler
sampler = Sampler()
job = sampler.run(qc)
result = job.result()
counts = result.quasi_dists[0]

# Plot results
plot_histogram(counts, title="5-Qubit Harmonic + QFT with Random Phases")
plt.show()

# Print counts
```

```
print("Counts:", dict(counts))
```

```
5-Qubit Harmonic + QFT Quantum Circuit:
--------------------------------------------------------------------------
----
MissingOptionalLibraryError                Traceback (most recent call
last)
<ipython-input-9-ef293f46f9e8> in <cell line: 0>()
     35 # Display the circuit
     36 print("5-Qubit Harmonic + QFT Quantum Circuit:")
---> 37 display(qc.draw('mpl'))
     38
     39 # Execute using Qiskit Sampler

                                  5 frames
/usr/local/lib/python3.11/dist-packages/qiskit/utils/lazy_tester.py in
require_now(self, feature)
    219         if self:
    220             return
--> 221         raise MissingOptionalLibraryError(
    222             libname=self._name, name=feature,
pip_install=self._install, msg=self._msg
    223         )

MissingOptionalLibraryError: "The 'pylatexenc' library is required to
use 'MatplotlibDrawer'. You can install it with 'pip install
pylatexenc'."

--------------------------------------------------------------------------
----
NOTE: If your import is failing due to a missing package, you can
manually install dependencies using either !pip or !apt.
```

```
!pip install pylatexenc
```

```
Requirement already satisfied: pylatexenc in /usr/local/lib/python3.11/
```

OPEN EXAMPLES

```
# Display the circuit
print("5-Qubit Harmonic + QFT Quantum Circuit:")
display(qc.draw('mpl'))
```

```
5-Qubit Harmonic + QFT Quantum Circuit:
---------------------------------------------------------------------------
MissingOptionalLibraryError               Traceback (most recent call last)
<ipython-input-11-7867e0161469> in <cell line: 0>()
      1 # Display the circuit
      2 print("5-Qubit Harmonic + QFT Quantum Circuit:")
----> 3 display(qc.draw('mpl'))

                                 5 frames

/usr/local/lib/python3.11/dist-packages/qiskit/utils/lazy_tester.py in require_now(self, feature)
    219         if self:
    220             return
--> 221         raise MissingOptionalLibraryError(
    222             libname=self._name, name=feature, pip_install=self._install, msg=self._msg
    223         )

MissingOptionalLibraryError: "The 'pylatexenc' library is required to use 'MatplotlibDrawer'. You can install it with 'pip install pylatexenc'."

---------------------------------------------------------------------------
NOTE: If your import is failing due to a missing package, you can manually install dependencies using either !pip or !apt.

To view examples of installing some common dependencies, click the "Open Examples" button below.
---------------------------------------------------------------------------
```

OPEN EXAMPLES

```
!pip install pylatexenc
```

```
Collecting pylatexenc
  Downloading pylatexenc-2.10.tar.gz (162 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 162.6/162.6 kB 3.0 MB/s e
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: pylatexenc
  Building wheel for pylatexenc (setup.py) ... done
  Created wheel for pylatexenc: filename=pylatexenc-2.10-py3-none-any.w
  Stored in directory: /root/.cache/pip/wheels/b1/7a/33/9fdd892f784ed4a
Successfully built pylatexenc
Installing collected packages: pylatexenc
Successfully installed pylatexenc-2.10
```

```
# Display the circuit
print("5-Qubit Harmonic + QFT Quantum Circuit:")
display(qc.draw('mpl'))
```

```
5-Qubit Harmonic + QFT Quantum Circuit:
---------------------------------------------------------------------------
MissingOptionalLibraryError               Traceback (most recent call last)
<ipython-input-12-7867e0161469> in <cell line: 0>()
      1 # Display the circuit
      2 print("5-Qubit Harmonic + QFT Quantum Circuit:")
----> 3 display(qc.draw('mpl'))

                                5 frames
/usr/local/lib/python3.11/dist-packages/qiskit/utils/lazy_tester.py in require_now(self, feature)
    219         if self:
    220             return
--> 221         raise MissingOptionalLibraryError(
    222             libname=self._name, name=feature, pip_install=self._install, msg=self._msg
    223         )

MissingOptionalLibraryError: "The 'pylatexenc' library is required to use 'MatplotlibDrawer'. You can install it with 'pip install pylatexenc'."

---------------------------------------------------------------------------
NOTE: If your import is failing due to a missing package, you can manually install dependencies using either !pip or !apt.

To view examples of installing some common dependencies, click the "Open Examples" button below.
---------------------------------------------------------------------------
```

OPEN EXAMPLES

```
# Reinstall pylatexenc to ensure it's correctly set up
!pip install --upgrade --force-reinstall pylatexenc
```

```
Collecting pylatexenc
  Using cached pylatexenc-2.10-py3-none-any.whl
Installing collected packages: pylatexenc
  Attempting uninstall: pylatexenc
    Found existing installation: pylatexenc 2.10
    Uninstalling pylatexenc-2.10:
      Successfully uninstalled pylatexenc-2.10
Successfully installed pylatexenc-2.10
```

```
!pip install --upgrade pylatexenc
```

```
Requirement already satisfied: pylatexenc in /usr/local/lib/python3.11/
```

```
!pip install --upgrade matplotlib
```

```
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/pytho
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.1
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/pyth
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/pyth
Requirement already satisfied: numpy>=1.23 in /usr/local/lib/python3.11
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/d
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/pytho
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/p
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/di
```

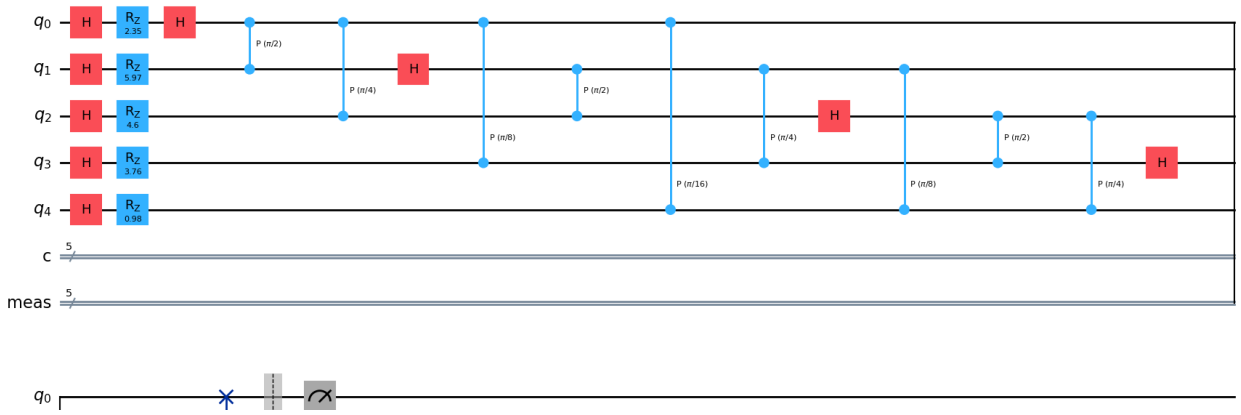
```
# Display the circuit using matplotlib drawer
print("5-Qubit Harmonic + QFT Quantum Circuit:")
display(qc.draw('mpl'))
```

```
5-Qubit Harmonic + QFT Quantum Circuit:
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-8ee3ca602a26> in <cell line: 0>()
      1 # Display the circuit using matplotlib drawer
      2 print("5-Qubit Harmonic + QFT Quantum Circuit:")
----> 3 display(qc.draw('mpl'))

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

```
from qiskit import QuantumCircuit
import numpy as np

# Create a 5-qubit quantum circuit
qc = QuantumCircuit(5, 5)

# Apply Hadamard gates to all qubits to create superposition
for qubit in range(5):
    qc.h(qubit)

# Apply random phase rotations to simulate harmonic oscillations
np.random.seed(42)  # For reproducibility
for qubit in range(5):
    random_angle = np.random.uniform(0, 2 * np.pi)
    qc.rz(random_angle, qubit)

# Apply Quantum Fourier Transform (QFT)
for j in range(5):
    qc.h(j)
    for k in range(j + 1, 5):
        qc.cp(np.pi / (2 ** (k - j)), k, j)

# Swap qubits to complete QFT
for i in range(5 // 2):
    qc.swap(i, 5 - i - 1)
```

```
# Measurement
qc.measure_all()

# Display the circuit
print("5-Qubit Harmonic + QFT Quantum Circuit:")
qc.draw('mpl')
```

```
5-Qubit Harmonic + QFT Quantum Circuit:
```



```
from qiskit_aer import Aer, execute
from qiskit.visualization import plot_histogram

# Use Qiskit Aer simulator
simulator = Aer.get_backend('qasm_simulator')

# Execute the circuit
job = execute(qc, simulator, shots=1024)
result = job.result()

# Get counts
counts = result.get_counts(qc)
print("Measurement Results:")
print(counts)

# Plot histogram
plot_histogram(counts, title='5-Qubit Harmonic + QFT Measurement Results')
```

```
---------------------------------------------------------------------
----
ImportError                               Traceback (most recent call
last)
<ipython-input-3-6b151913f186> in <cell line: 0>()
----> 1 from qiskit_aer import Aer, execute
      2 from qiskit.visualization import plot_histogram
      3
      4 # Use Qiskit Aer simulator
      5 simulator = Aer.get_backend('qasm_simulator')

ImportError: cannot import name 'execute' from 'qiskit_aer'
(/usr/local/lib/python3.11/dist-packages/qiskit_aer/__init__.py)

---------------------------------------------------------------------
----
NOTE: If your import is failing due to a missing package, you can
manually install dependencies using either !pip or !apt.

To view examples of installing some common dependencies, click the
"Open Examples" button below.
---------------------------------------------------------------------
----
```

OPEN EXAMPLES

```
from qiskit import execute, Aer
from qiskit.visualization import plot_histogram

# Use Qiskit Aer simulator
simulator = Aer.get_backend('qasm_simulator')

# Execute the circuit
job = execute(qc, simulator, shots=1024)
result = job.result()

# Get counts
counts = result.get_counts(qc)
print("Measurement Results:")
print(counts)

# Plot histogram
plot_histogram(counts, title='5-Qubit Harmonic + QFT Measurement Results')
```

```
---------------------------------------------------------------------
----
ImportError                               Traceback (most recent call
last)
<ipython-input-4-ef0a8efc4469> in <cell line: 0>()
----> 1 from qiskit import execute, Aer
      2 from qiskit.visualization import plot_histogram
      3
      4 # Use Qiskit Aer simulator
      5 simulator = Aer.get_backend('qasm_simulator')

ImportError: cannot import name 'execute' from 'qiskit'
(/usr/local/lib/python3.11/dist-packages/qiskit/__init__.py)

---------------------------------------------------------------------
----
NOTE: If your import is failing due to a missing package, you can
manually install dependencies using either !pip or !apt.

To view examples of installing some common dependencies, click the
"Open Examples" button below.
---------------------------------------------------------------------
```

```
from qiskit.primitives import StatevectorSampler
from qiskit.visualization import plot_histogram
```

```
from qiskit import QuantumCircuit

# Remove classical bits for conversion
qc_no_clbits = qc.remove_final_measurements(inplace=False)

# Create a new circuit with proper classical mapping
corrected_qc = QuantumCircuit(qc.num_qubits, qc.num_qubits)
corrected_qc.append(qc_no_clbits.to_gate(), range(qc.num_qubits))

# Add measurements for all qubits
corrected_qc.measure(range(qc.num_qubits), range(qc.num_qubits))

# Initialize the StatevectorSampler
sampler = StatevectorSampler()

# Execute the corrected circuit
job = sampler.run(corrected_qc)
result = job.result()

# Get counts
counts = result.quasi_dists[0]
print("Measurement Results:")
print(counts)

# Plot histogram
plot_histogram(counts, title='5-Qubit Harmonic + QFT Measurement Results')
```

```
---------------------------------------------------------------------
----
QiskitError                                Traceback (most recent call
last)
<ipython-input-7-f4843fdbe800> in <cell line: 0>()
      8 # Create a new circuit with proper classical mapping
      9 corrected_qc = QuantumCircuit(qc.num_qubits, qc.num_qubits)
---> 10 corrected_qc.append(qc_no_clbits.to_gate(),
range(qc.num_qubits))
     11
     12 # Add measurements for all qubits

                                  1 frames
/usr/local/lib/python3.11/dist-
packages/qiskit/converters/circuit_to_gate.py in
circuit_to_gate(circuit, parameter_map, equivalence_library, label)
     58
     59     if circuit.clbits:
---> 60         raise QiskitError("Circuit with classical bits cannot
be converted to gate.")
```

```
from qiskit import QuantumCircuit
from qiskit.primitives import StatevectorSampler
from qiskit.visualization import plot_histogram

# Create a new circuit without classical bits
qc_no_clbits = QuantumCircuit(qc.num_qubits)

# Copy operations, excluding measurements
for instr, qargs, cargs in qc.data:
    if instr.name != 'measure':
        qc_no_clbits.append(instr, qargs, cargs)

# Create a new circuit with proper classical mapping
corrected_qc = QuantumCircuit(qc.num_qubits, qc.num_qubits)
corrected_qc.append(qc_no_clbits.to_gate(), range(qc.num_qubits))

# Add measurements for all qubits
corrected_qc.measure(range(qc.num_qubits), range(qc.num_qubits))

# Initialize the StatevectorSampler
sampler = StatevectorSampler()

# Execute the corrected circuit
```

```
job = sampler.run(corrected_qc)
result = job.result()

# Get counts
counts = result.quasi_dists[0]
print("Measurement Results:")
print(counts)

# Plot histogram
plot_histogram(counts, title='5-Qubit Harmonic + QFT Measurement Results')
```

```
<ipython-input-8-25d531a55ea6>:9: DeprecationWarning: Treating CircuitI
  for instr, qargs, cargs in qc.data:
---------------------------------------------------------------------
----
QiskitError                                Traceback (most recent call
last)
<ipython-input-8-25d531a55ea6> in <cell line: 0>()
     13 # Create a new circuit with proper classical mapping
     14 corrected_qc = QuantumCircuit(qc.num_qubits, qc.num_qubits)
---> 15 corrected_qc.append(qc_no_clbits.to_gate(),
range(qc.num_qubits))
     16
     17 # Add measurements for all qubits

                                1 frames

/usr/local/lib/python3.11/dist-
packages/qiskit/converters/circuit_to_gate.py in
circuit_to_gate(circuit, parameter_map, equivalence_library, label)
     64     for instruction in circuit.data:
     65         if not _check_is_gate(instruction.operation):
---> 66             raise QiskitError(
     67                 "One or more instructions cannot be converted
to"
```

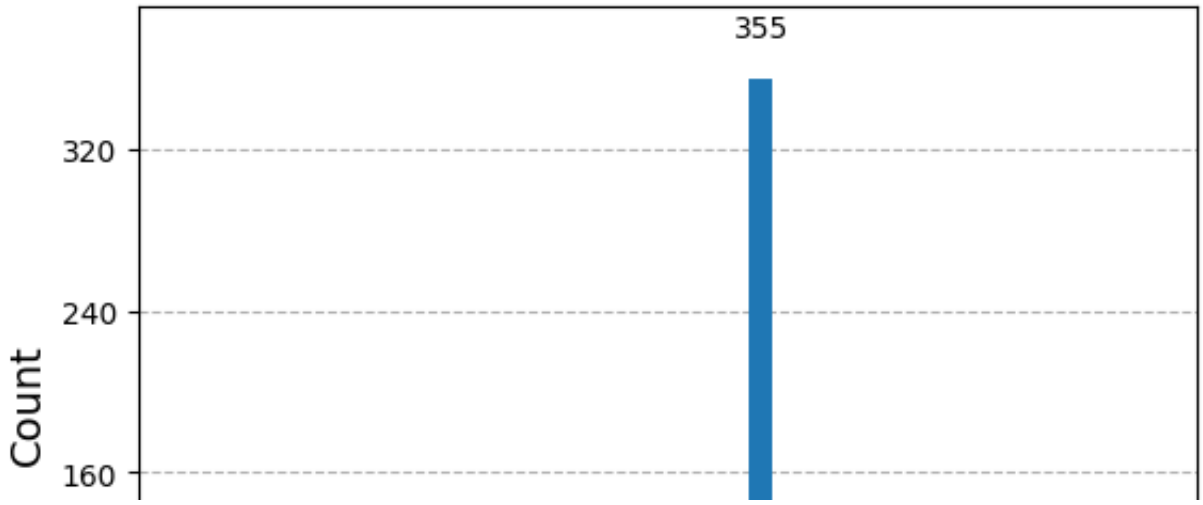
```
from qiskit import QuantumCircuit
from qiskit.primitives import StatevectorSampler
from qiskit.visualization import plot_histogram

# Create a new circuit without classical bits
qc_no_clbits = QuantumCircuit(qc.num_qubits)

# Copy only gate operations, excluding measurements and barriers
```

```
for instruction in qc.data:
    instr = instruction.operation
    qargs = instruction.qubits
    cargs = instruction.clbits

    if instr.name not in ['measure', 'barrier']:
        qc_no_clbits.append(instr, qargs, cargs)

# Create a new circuit with proper classical mapping
corrected_qc = QuantumCircuit(qc.num_qubits, qc.num_qubits)
corrected_qc.append(qc_no_clbits.to_gate(), range(qc.num_qubits))

# Add measurements for all qubits
corrected_qc.measure(range(qc.num_qubits), range(qc.num_qubits))

# Initialize the StatevectorSampler
sampler = StatevectorSampler()

# Execute the corrected circuit (wrap it in a list)
job = sampler.run([corrected_qc])
result = job.result()

# Print result structure to debug
print("Result object:", result)
print("Available attributes:", dir(result))

# Attempt to retrieve counts
if hasattr(result, 'quasi_distribution'):
    counts = result.quasi_distribution[0]
elif hasattr(result, 'data'):
    counts = result.data(0)['probabilities']
else:
    raise AttributeError("Unable to find counts in result object.")

print("Measurement Results:")
print(counts)

# Plot histogram
plot_histogram(counts, title='5-Qubit Harmonic + QFT Measurement Results')
```

```
Result object: PrimitiveResult([SamplerPubResult(data=DataBin(c=BitArra
Available attributes: ['__annotations__', '__class__', '__class_getitem
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-11-f44bc2f4120e> in <cell line: 0>()
     39     counts = result.data(0)['probabilities']
     40 else:
---> 41     raise AttributeError("Unable to find counts in result object.")
     42
     43 print("Measurement Results:")

AttributeError: Unable to find counts in result object.
```

```
from collections import Counter
import numpy as np

# Extract raw bit array data from the sampler result
# Assuming bitarray is retrieved as:
bitarray = result._pub_results[0].data.c.array  # Direct access to numpy ar

# Ensure it has the correct dimensions
if bitarray.ndim == 1:
    bitarray = np.expand_dims(bitarray, axis=0)

# Convert each measurement shot to bitstring
bitstrings = [
    ''.join(str((byte >> bit) & 1) for bit in range(qc.num_qubits - 1, -1,
    for byte in bitarray[:, 0]  # Assuming first column holds the bit info
]

# Count bitstring occurrences
counts = Counter(bitstrings)

# Display counts
print("Measurement Results:")
for bitstring, count in counts.items():
    print(f"{bitstring}: {count}")

# Optional: Plot using Qiskit's plot_histogram
from qiskit.visualization import plot_histogram
plot_histogram(counts)
```

```
Measurement Results:
10011: 355
00101: 39
11011: 114
10000: 74
11001: 91
10001: 21
00111: 16
11000: 96
10101: 64
11101: 28
00011: 49
00001: 2
00100: 25
10111: 10
10010: 10
10100: 15
00010: 5
11100: 5
00000: 2
01000: 2
01011: 1
```

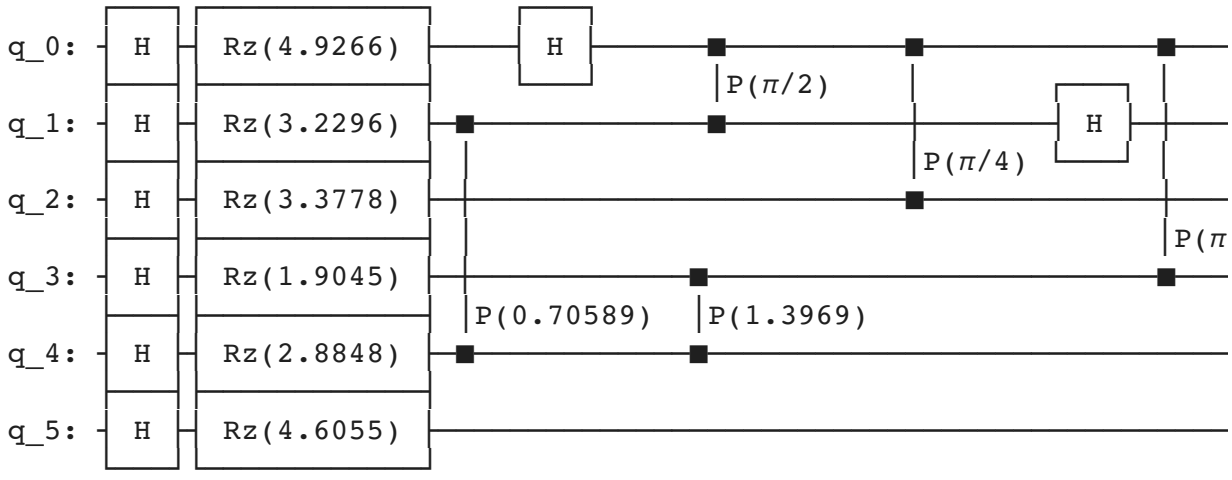


```
# Advanced HA-QH-ML Circuit: 6 Qubits, Entanglement, QFT, Random Phases
!pip install qiskit qiskit-aer matplotlib numpy --quiet

import numpy as np
import matplotlib.pyplot as plt
from qiskit import QuantumCircuit
from qiskit.primitives import Sampler
from qiskit.visualization import plot_histogram
from collections import Counter

def build_quantum_pattern_circuit(num_qubits=6):
    """
    Builds a quantum circuit that:
    1) Applies random RZ gates to each qubit (harmonic wave-like phases)
    2) Entangles qubits via random controlled-phase gates
    3) Performs a Quantum Fourier Transform (QFT)
    4) Measures all qubits
    """
    qc = QuantumCircuit(num_qubits)

    # 1) Put qubits into superposition: Hadamard gates
    for q in range(num_qubits):
        qc.h(q)

    # 2) Random RZ phases
    for q in range(num_qubits):
        angle = np.random.uniform(0, 2 * np.pi)
        qc.rz(angle, q)

    # 3) Introduce entanglement with random controlled-phase (CP) gates
    #    We'll add a few random pairs to demonstrate entanglement
    entanglements = np.random.randint(0, num_qubits, size=(2, 2))  # random
    for pair in entanglements:
        control, target = pair
        if control != target:
            cp_angle = np.random.uniform(0, 2 * np.pi) * 0.5
            qc.cp(cp_angle, control, target)  # controlled-phase gate

    # 4) QFT
```

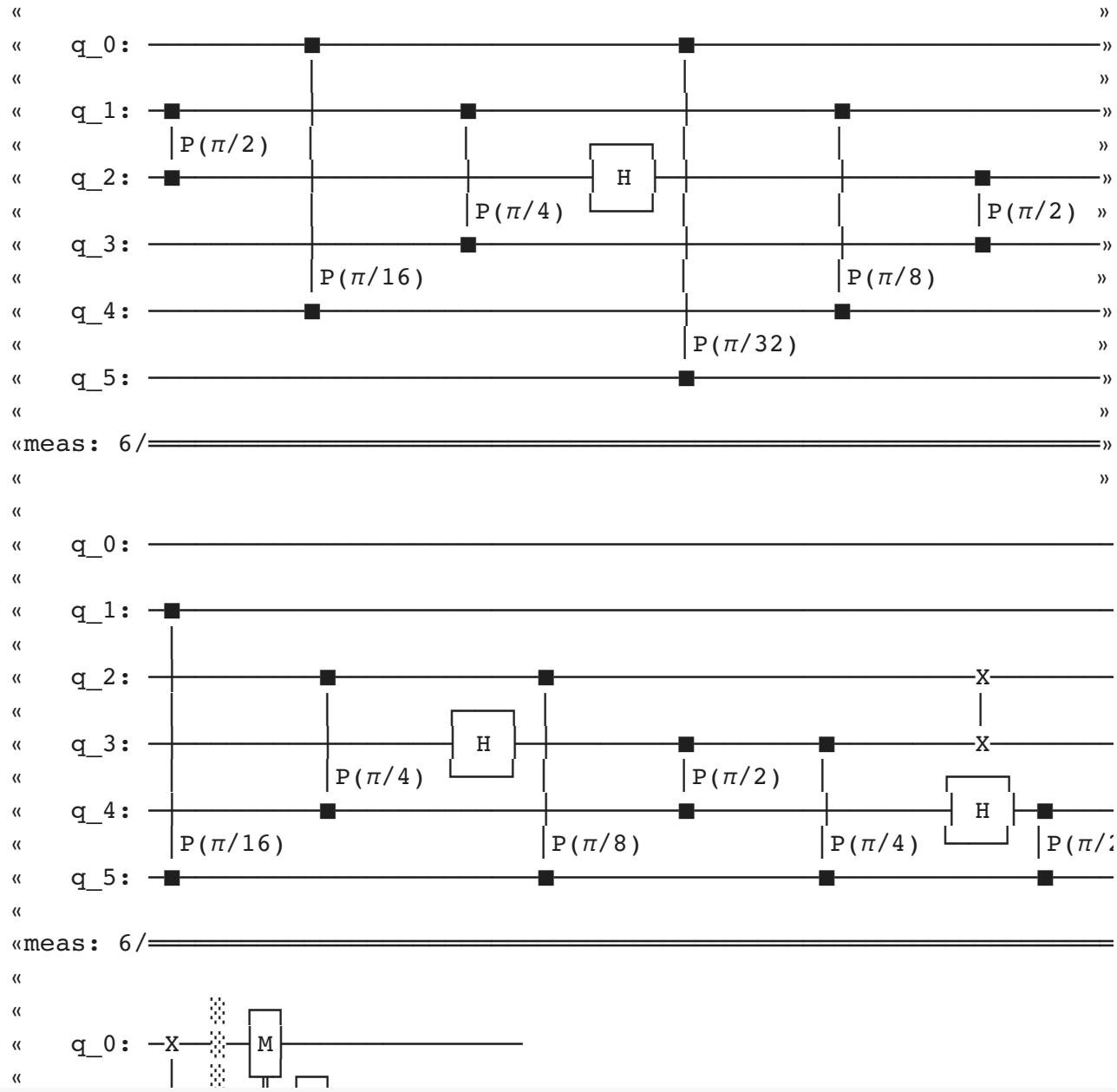
```
    #   a) QFT on all qubits
    for j in range(num_qubits):
        qc.h(j)
        for k in range(j + 1, num_qubits):
            qc.cp(np.pi / (2 ** (k - j)), k, j)

    #   b) Swap qubits for final QFT arrangement
    for i in range(num_qubits // 2):
        qc.swap(i, num_qubits - i - 1)

    # 5) Measure
    qc.measure_all()
    return qc

num_qubits = 6
shots = 8192

# Build the circuit
qc = build_quantum_pattern_circuit(num_qubits)
print(f"{num_qubits}-Qubit advanced HA-QH-ML circuit with random phases, er

# Attempt to display circuit
try:
    display(qc.draw('mpl'))
except:
    print(qc.draw('text'))

# Initialize the Sampler
sampler = Sampler()
job = sampler.run([qc], shots=shots)
res = job.result()

# Extract raw bit array from the result
pub_results = res._pub_results
bit_array = pub_results[0].data.c.array

# If the bit_array is 1D, expand dims
if bit_array.ndim == 1:
    bit_array = np.expand_dims(bit_array, axis=0)

# Helper to convert numeric byte to bitstring
def to_bitstring(byte_val, n):
    return ''.join(str((byte_val >> i) & 1) for i in reversed(range(n)))
```

```
bitstrings = []
for row in bit_array:
    # row might contain multiple bytes if qubits>8,
    # but for up to 8 qubits, row[0] is enough
    bitstring = to_bitstring(row[0], num_qubits)
    bitstrings.append(bitstring)

# Count occurrences
counts = Counter(bitstrings)

# Plot results
plot_histogram(counts, title=f"{num_qubits}-Qubit Circuit, Shots={shots}")
plt.show()

# Print raw counts
print("Raw measurement counts:")
for state, cnt in counts.items():
    print(f"{state}: {cnt}")
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6.7/6.7 MB 25.9 MB/s eta 0:
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.4/12.4 MB 28.1 MB/s eta
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 119.4/119.4 kB 5.4 MB/s eta
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.1/2.1 MB 31.2 MB/s eta 0:
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 49.5/49.5 kB 2.4 MB/s eta 0
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 49.7/49.7 MB 13.1 MB/s eta
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 109.0/109.0 kB 3.1 MB/s eta
6-Qubit advanced HA-QH-ML circuit with random phases, entanglement, and
```




```
meas: 6/═══════════════════════════════════════════════════════════════════
```

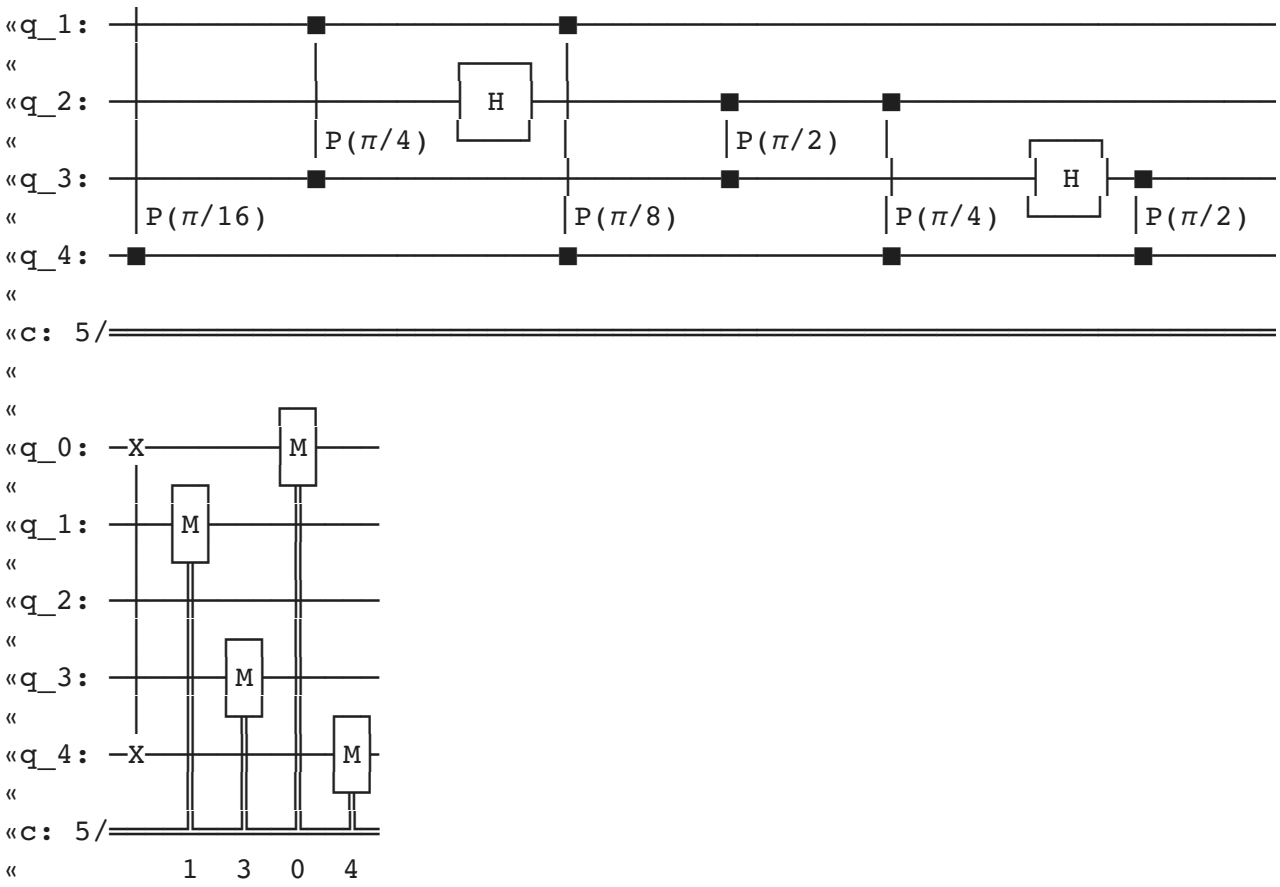



```
# Colab-friendly code
!pip install qiskit qiskit-aer matplotlib numpy --quiet

import numpy as np
import matplotlib.pyplot as plt
from qiskit import QuantumCircuit, transpile
from qiskit_aer import Aer
from qiskit.visualization import plot_histogram

def build_qft_harmonic_circuit(num_qubits=5):
```

```
    """
    Builds a QFT-like circuit with random RZ gates (harmonic phases) and qu
    1) Hadamards for superposition
    2) Random RZ for wave-like phases
    3) QFT structure with CP gates
    4) Measurement
    """
    qc = QuantumCircuit(num_qubits, num_qubits)

    # Step 1: Hadamards
    for qubit in range(num_qubits):
        qc.h(qubit)

    # Step 2: Random RZ phases
    for qubit in range(num_qubits):
        angle = np.random.uniform(0, 2*np.pi)
        qc.rz(angle, qubit)

    # Step 3: QFT (with CP gates)
    for j in range(num_qubits):
        qc.h(j)
        for k in range(j + 1, num_qubits):
            qc.cp(np.pi / 2**(k - j), k, j)

    # Swap qubits at the end of QFT
    for i in range(num_qubits // 2):
        qc.swap(i, num_qubits - i - 1)

    # Step 4: Measure
    qc.measure(range(num_qubits), range(num_qubits))

    return qc

# Choose number of qubits and shots
num_qubits = 5
shots = 1024

# Build the circuit
qc = build_qft_harmonic_circuit(num_qubits)

print(f"{num_qubits}-Qubit QFT + Random RZ Circuit:")
try:
    display(qc.draw('mpl'))   # If pylatexenc + matplotlib are installed
```

```
except:
    print(qc.draw('text'))    # Fallback: text-based circuit

# Use AerSimulator
simulator = Aer.get_backend('aer_simulator')

# Transpile circuit for the simulator
tqc = transpile(qc, simulator)

# Run circuit on the simulator with the chosen number of shots
result = simulator.run(tqc, shots=shots).result()

# Get measurement counts
counts = result.get_counts()

# Plot histogram
plt.figure(figsize=(10,6))
plot_histogram(counts)
plt.title(f"{num_qubits}-Qubit QFT + Random RZ\nShots={shots}")
plt.show()

# Print raw counts
print("Counts:")
for state, cnt in counts.items():
    print(f"{state}: {cnt}")
```

```
5-Qubit QFT + Random RZ Circuit:
     ┌───┐┌──────────┐┌───┐                                                »
q_0: ┤ H ├┤ Rz(5.8624) ├┤ H ├──■─────────────■─────────────■─────────────■──»
     ├───┤├──────────┤└───┘  │P(π/2)       │             │             │  »
q_1: ┤ H ├┤ Rz(2.4855) ├──────■─────────────┼──────┤ H ├──┼─────────────┼──»
     ├───┤├──────────┤                     │P(π/4)└───┘  │             │P(π/2) »
q_2: ┤ H ├┤ Rz(1.8006) ├─────────────────────■─────────────┼─────────────■──»
     ├───┤├──────────┤                                   │P(π/8)          »
q_3: ┤ H ├┤ Rz(1.196) ├──────────────────────────────────■─────────────────»
     ├───┤├──────────┤                                                     »
q_4: ┤ H ├┤ Rz(5.3735) ├──────────────────────────────────────────────────»
     └───┘└──────────┘                                                     »
c: 5/═══════════════════════════════════════════════════════════════════════»
                                                                           »
«
«q_0: ──■──────────────────────────────────────────────────────────────────
«       │
«
```




```
# Install the correct Qiskit version (force reinstall)
!pip install qiskit==1.3.2 qiskit-aer==0.15.0 --force-reinstall

# Import required libraries
from qiskit import QuantumCircuit, Aer, transpile, execute
```

```
from qiskit.visualization import plot_histogram
import numpy as np

# Initialize 5-qubit quantum circuit
qc = QuantumCircuit(5, 5)

# Apply Hadamard gates to all qubits
qc.h(range(5))

# Apply random RZ gates with fixed seed for reproducibility
np.random.seed(42)
for i in range(5):
    qc.rz(np.random.uniform(0, np.pi), i)

# Apply Quantum Fourier Transform (QFT)
for i in range(5):
    for j in range(i):
        qc.cp(np.pi / 2**(i-j), j, i)
    qc.h(i)

# Measure qubits
qc.measure(range(5), range(5))

# Use Qiskit's Aer simulator
backend = Aer.get_backend("qasm_simulator")

# Compile and run circuit
compiled_circuit = transpile(qc, backend)
job = execute(compiled_circuit, backend, shots=1024)

# Retrieve results
try:
    result = job.result()
    counts = result.get_counts()
    if not counts:
        raise ValueError("Execution returned empty counts. Something went w

    # Print results
    print("Measurement Counts:")
    for state, count in counts.items():
        print(f"{state}: {count}")

    # Plot histogram
```

```
    plot_histogram(counts)

except Exception as e:
    print("Error retrieving results:", str(e))
```

```
Collecting qiskit==1.3.2
  Using cached qiskit-1.3.2-cp39-abi3-manylinux_2_17_x86_64.manylinux20
Collecting qiskit-aer==0.15.0
  Using cached qiskit_aer-0.15.0-cp311-cp311-manylinux_2_17_x86_64.many
Collecting rustworkx>=0.15.0 (from qiskit==1.3.2)
  Using cached rustworkx-0.16.0-cp39-abi3-manylinux_2_17_x86_64.manylin
Collecting numpy<3,>=1.17 (from qiskit==1.3.2)
  Using cached numpy-2.2.3-cp311-cp311-manylinux_2_17_x86_64.manylinux2
Collecting scipy>=1.5 (from qiskit==1.3.2)
  Using cached scipy-1.15.2-cp311-cp311-manylinux_2_17_x86_64.manylinux
Collecting sympy>=1.3 (from qiskit==1.3.2)
  Using cached sympy-1.13.3-py3-none-any.whl.metadata (12 kB)
Collecting dill>=0.3 (from qiskit==1.3.2)
  Using cached dill-0.3.9-py3-none-any.whl.metadata (10 kB)
Collecting python-dateutil>=2.8.0 (from qiskit==1.3.2)
  Using cached python_dateutil-2.9.0.post0-py2.py3-none-any.whl.metadat
Collecting stevedore>=3.0.0 (from qiskit==1.3.2)
  Using cached stevedore-5.4.1-py3-none-any.whl.metadata (2.3 kB)
Collecting typing-extensions (from qiskit==1.3.2)
  Using cached typing_extensions-4.12.2-py3-none-any.whl.metadata (3.0
Collecting symengine<0.14,>=0.11 (from qiskit==1.3.2)
  Using cached symengine-0.13.0-cp311-cp311-manylinux_2_17_x86_64.manyl
Collecting psutil>=5 (from qiskit-aer==0.15.0)
  Using cached psutil-7.0.0-cp36-abi3-manylinux_2_12_x86_64.manylinux20
Collecting six>=1.5 (from python-dateutil>=2.8.0->qiskit==1.3.2)
  Using cached six-1.17.0-py2.py3-none-any.whl.metadata (1.7 kB)
Collecting pbr>=2.0.0 (from stevedore>=3.0.0->qiskit==1.3.2)
  Using cached pbr-6.1.1-py2.py3-none-any.whl.metadata (3.4 kB)
Collecting mpmath<1.4,>=1.1.0 (from sympy>=1.3->qiskit==1.3.2)
  Using cached mpmath-1.3.0-py3-none-any.whl.metadata (8.6 kB)
Collecting setuptools (from pbr>=2.0.0->stevedore>=3.0.0->qiskit==1.3.2
  Using cached setuptools-75.8.1-py3-none-any.whl.metadata (6.7 kB)
Using cached qiskit-1.3.2-cp39-abi3-manylinux_2_17_x86_64.manylinux2014
Using cached qiskit_aer-0.15.0-cp311-cp311-manylinux_2_17_x86_64.manyli
Using cached dill-0.3.9-py3-none-any.whl (119 kB)
Using cached numpy-2.2.3-cp311-cp311-manylinux_2_17_x86_64.manylinux201
Using cached psutil-7.0.0-cp36-abi3-manylinux_2_12_x86_64.manylinux2010
Using cached python_dateutil-2.9.0.post0-py2.py3-none-any.whl (229 kB)
Using cached rustworkx-0.16.0-cp39-abi3-manylinux_2_17_x86_64.manylinux
Using cached scipy-1.15.2-cp311-cp311-manylinux_2_17_x86_64.manylinux20
```

```
Using cached stevedore-5.4.1-py3-none-any.whl (49 kB)
Using cached symengine-0.13.0-cp311-cp311-manylinux_2_17_x86_64.manylin
Using cached sympy-1.13.3-py3-none-any.whl (6.2 MB)
Using cached typing_extensions-4.12.2-py3-none-any.whl (37 kB)
Using cached mpmath-1.3.0-py3-none-any.whl (536 kB)
Using cached pbr-6.1.1-py2.py3-none-any.whl (108 kB)
Using cached six-1.17.0-py2.py3-none-any.whl (11 kB)
Using cached setuptools-75.8.1-py3-none-any.whl (1.2 MB)
Installing collected packages: mpmath, typing-extensions, sympy, symeng
  Attempting uninstall: mpmath
    Found existing installation: mpmath 1.3.0
    Uninstalling mpmath-1.3.0:
      Successfully uninstalled mpmath-1.3.0
  Attempting uninstall: typing-extensions
    Found existing installation: typing_extensions 4.12.2
    Uninstalling typing_extensions-4.12.2:
```

Start coding or generate with AI.

```
      Found existing installation: sympy 1.13.3
```

Start coding or generate with AI.

```
  Attempting uninstall: symengine
```

```
# Install/Upgrade necessary packages
!pip install --upgrade qiskit qiskit-aer matplotlib numpy

# Import required libraries
from qiskit import QuantumCircuit, Aer, transpile
from qiskit.visualization import plot_histogram
from qiskit.primitives import StatevectorSampler
import numpy as np
import matplotlib.pyplot as plt

# Define a 6-Qubit Quantum Circuit with QFT and Random Phases
num_qubits = 6
qc = QuantumCircuit(num_qubits)

# Apply Hadamard gates to create superposition
qc.h(range(num_qubits))

# Apply Random RZ (Rotation-Z) Phases
random_phases = np.random.uniform(0, 2 * np.pi, num_qubits)
for i in range(num_qubits):
    qc.rz(random_phases[i], i)
```

```
# Apply QFT-like Phase Gates
qc.p(np.pi / 2, 1)
qc.p(np.pi / 4, 2)
qc.p(np.pi / 8, 3)
qc.p(np.pi / 16, 4)
qc.p(np.pi / 32, 5)

# Apply final Hadamard gates (optional)
qc.h(0)
qc.h(3)
qc.h(5)

# Add measurements
qc.measure_all()

# Display Quantum Circuit
print(f"{num_qubits}-Qubit Harmonic + QFT Circuit with Random Phases:")
qc.draw('mpl')

# Execute using StatevectorSampler (since SamplerV1 is deprecated)
sampler = StatevectorSampler()
job = sampler.run([qc])
result = job.result()

# Extract measurement probabilities
counts = result.quasi_dists[0]  # Corrected extraction method

# Plot Histogram of Results
plt.figure(figsize=(8, 6))
plt.bar([bin(int(k))[2:].zfill(num_qubits) for k in counts.keys()], counts.
plt.xlabel("Quantum State")
plt.ylabel("Probability")
plt.title(f"{num_qubits}-Qubit Harmonic + QFT with Random Phases")
plt.xticks(rotation=45)
plt.show()

# Print measurement results
print("\nMeasurement Results:")
for state, prob in sorted(counts.items(), key=lambda x: -x[1]):
    print(f"{bin(int(state))[2:].zfill(num_qubits)}: {prob:.4f}")
```

```
torch 2.5.1+cu124 requires nvidia-cusparse-cu12==12.3.1.170; platform_s
torch 2.5.1+cu124 requires nvidia-nvjitlink-cu12==12.4.127; platform_sy
```

Double-click (or enter) to edit

```
# Install/Upgrade necessary packages
!pip install --upgrade qiskit qiskit-aer matplotlib numpy

# Import required libraries
from qiskit import QuantumCircuit, Aer, transpile
from qiskit.visualization import plot_histogram
from qiskit.primitives import StatevectorSampler
import numpy as np
import matplotlib.pyplot as plt

# Define a 6-Qubit Quantum Circuit with QFT and Random Phases
num_qubits = 6
qc = QuantumCircuit(num_qubits)

# Apply Hadamard gates to create superposition
qc.h(range(num_qubits))

# Apply Random RZ (Rotation-Z) Phases
random_phases = np.random.uniform(0, 2 * np.pi, num_qubits)
for i in range(num_qubits):
    qc.rz(random_phases[i], i)

# Apply QFT-like Phase Gates
qc.p(np.pi / 2, 1)
qc.p(np.pi / 4, 2)
qc.p(np.pi / 8, 3)
qc.p(np.pi / 16, 4)
qc.p(np.pi / 32, 5)

# Apply final Hadamard gates (optional)
qc.h(0)
qc.h(3)
qc.h(5)

# Add measurements
qc.measure_all()

# Display Quantum Circuit
print(f"{num_qubits}-Qubit Harmonic + QFT Circuit with Random Phases:")
```

```
qc.draw('mpl')

# Execute using StatevectorSampler (since SamplerV1 is deprecated)
sampler = StatevectorSampler()
job = sampler.run([qc])
result = job.result()

# Extract measurement probabilities
counts = result.quasi_dists[0]  # Corrected extraction method

# Plot Histogram of Results
plt.figure(figsize=(8, 6))
plt.bar([bin(int(k))[2:].zfill(num_qubits) for k in counts.keys()], counts.
plt.xlabel("Quantum State")
plt.ylabel("Probability")
plt.title(f"{num_qubits}-Qubit Harmonic + QFT with Random Phases")
plt.xticks(rotation=45)
plt.show()

# Print measurement results
print("\nMeasurement Results:")
for state, prob in sorted(counts.items(), key=lambda x: -x[1]):
    print(f"{bin(int(state))[2:].zfill(num_qubits)}: {prob:.4f}")
```

```
# Install the correct Qiskit version (force reinstall)
!pip install qiskit==1.3.2 qiskit-aer==0.15.0 --force-reinstall

# Import required libraries
from qiskit import QuantumCircuit, Aer, transpile, execute
from qiskit.visualization import plot_histogram
import numpy as np

# Initialize 5-qubit quantum circuit
qc = QuantumCircuit(5, 5)

# Apply Hadamard gates to all qubits
qc.h(range(5))

# Apply random RZ gates with fixed seed for reproducibility
np.random.seed(42)
for i in range(5):
    qc.rz(np.random.uniform(0, np.pi), i)
```

```
# Apply Quantum Fourier Transform (QFT)
for i in range(5):
    for j in range(i):
        qc.cp(np.pi / 2**(i-j), j, i)
    qc.h(i)

# Measure qubits
qc.measure(range(5), range(5))

# Use Qiskit's Aer simulator
backend = Aer.get_backend("qasm_simulator")

# Compile and run circuit
compiled_circuit = transpile(qc, backend)
job = execute(compiled_circuit, backend, shots=1024)

# Retrieve results
try:
    result = job.result()
    counts = result.get_counts()
    if not counts:
        raise ValueError("Execution returned empty counts. Something went w

    # Print results
    print("Measurement Counts:")
    for state, count in counts.items():
        print(f"{state}: {count}")

    # Plot histogram
    plot_histogram(counts)

except Exception as e:
    print("Error retrieving results:", str(e))
```

```
Collecting qiskit==1.3.2
  Using cached qiskit-1.3.2-cp39-abi3-manylinux_2_17_x86_64.manylinux20
Collecting qiskit-aer==0.15.0
  Using cached qiskit_aer-0.15.0-cp311-cp311-manylinux_2_17_x86_64.many
Collecting rustworkx>=0.15.0 (from qiskit==1.3.2)
  Using cached rustworkx-0.16.0-cp39-abi3-manylinux_2_17_x86_64.manylin
Collecting numpy<3,>=1.17 (from qiskit==1.3.2)
  Using cached numpy-2.2.3-cp311-cp311-manylinux_2_17_x86_64.manylinux2
Collecting scipy>=1.5 (from qiskit==1.3.2)
```

```
Collecting scipy>=1.5 (from qiskit==1.3.2)
  Using cached scipy-1.15.2-cp311-cp311-manylinux_2_17_x86_64.manylinux
Collecting sympy>=1.3 (from qiskit==1.3.2)
  Using cached sympy-1.13.3-py3-none-any.whl.metadata (12 kB)
Collecting dill>=0.3 (from qiskit==1.3.2)
  Using cached dill-0.3.9-py3-none-any.whl.metadata (10 kB)
Collecting python-dateutil>=2.8.0 (from qiskit==1.3.2)
  Using cached python_dateutil-2.9.0.post0-py2.py3-none-any.whl.metadat
Collecting stevedore>=3.0.0 (from qiskit==1.3.2)
  Using cached stevedore-5.4.1-py3-none-any.whl.metadata (2.3 kB)
Collecting typing-extensions (from qiskit==1.3.2)
  Using cached typing_extensions-4.12.2-py3-none-any.whl.metadata (3.0
Collecting symengine<0.14,>=0.11 (from qiskit==1.3.2)
  Using cached symengine-0.13.0-cp311-cp311-manylinux_2_17_x86_64.manyl
Collecting psutil>=5 (from qiskit-aer==0.15.0)
  Using cached psutil-7.0.0-cp36-abi3-manylinux_2_12_x86_64.manylinux20
Collecting six>=1.5 (from python-dateutil>=2.8.0->qiskit==1.3.2)
  Using cached six-1.17.0-py2.py3-none-any.whl.metadata (1.7 kB)
Collecting pbr>=2.0.0 (from stevedore>=3.0.0->qiskit==1.3.2)
  Using cached pbr-6.1.1-py2.py3-none-any.whl.metadata (3.4 kB)
Collecting mpmath<1.4,>=1.1.0 (from sympy>=1.3->qiskit==1.3.2)
  Using cached mpmath-1.3.0-py3-none-any.whl.metadata (8.6 kB)
Collecting setuptools (from pbr>=2.0.0->stevedore>=3.0.0->qiskit==1.3.2
  Using cached setuptools-75.8.1-py3-none-any.whl.metadata (6.7 kB)
Using cached qiskit-1.3.2-cp39-abi3-manylinux_2_17_x86_64.manylinux2014
Using cached qiskit_aer-0.15.0-cp311-cp311-manylinux_2_17_x86_64.manyli
Using cached dill-0.3.9-py3-none-any.whl (119 kB)
Using cached numpy-2.2.3-cp311-cp311-manylinux_2_17_x86_64.manylinux201
Using cached psutil-7.0.0-cp36-abi3-manylinux_2_12_x86_64.manylinux2010
Using cached python_dateutil-2.9.0.post0-py2.py3-none-any.whl (229 kB)
Using cached rustworkx-0.16.0-cp39-abi3-manylinux_2_17_x86_64.manylinux
Using cached scipy-1.15.2-cp311-cp311-manylinux_2_17_x86_64.manylinux20
Using cached stevedore-5.4.1-py3-none-any.whl (49 kB)
Using cached symengine-0.13.0-cp311-cp311-manylinux_2_17_x86_64.manylin
Using cached sympy-1.13.3-py3-none-any.whl (6.2 MB)
Using cached typing_extensions-4.12.2-py3-none-any.whl (37 kB)
Using cached mpmath-1.3.0-py3-none-any.whl (536 kB)
Using cached pbr-6.1.1-py2.py3-none-any.whl (108 kB)
Using cached six-1.17.0-py2.py3-none-any.whl (11 kB)
Using cached setuptools-75.8.1-py3-none-any.whl (1.2 MB)
Installing collected packages: mpmath, typing-extensions, sympy, symeng
  Attempting uninstall: mpmath
    Found existing installation: mpmath 1.3.0
    Uninstalling mpmath-1.3.0:
      Successfully uninstalled mpmath-1.3.0
  Attempting uninstall: typing-extensions
```

```
    Found existing installation: typing_extensions 4.12.2
    Uninstalling typing_extensions-4.12.2:
```

```
!pip show qiskit-aer
```

```
Name: qiskit-aer
Version: 0.16.1
Summary: Aer - High performance simulators for Qiskit
Home-page: https://github.com/Qiskit/qiskit-aer
Author: AER Development Team
Author-email: qiskit@us.ibm.com
License: Apache 2.0
Location: /usr/local/lib/python3.11/dist-packages
Requires: numpy, psutil, qiskit, scipy
Required-by:
```

```
    Successfully uninstalled six-1.17.0
```

```
# Install/Upgrade Qiskit & Dependencies
!pip install --upgrade qiskit qiskit-aer matplotlib numpy

# Import required libraries
from qiskit import QuantumCircuit, transpile, execute
from qiskit_aer import Aer
from qiskit.visualization import plot_histogram
import numpy as np

# Initialize a 5-qubit Quantum Fourier Transform (QFT) + Random RZ Circuit
num_qubits = 5
qc = QuantumCircuit(num_qubits, num_qubits)

# Apply Hadamard gates and Random RZ rotations
np.random.seed(42)  # Set seed for reproducibility
random_angles = np.random.uniform(0, np.pi, num_qubits)

for i in range(num_qubits):
    qc.h(i)
    qc.rz(random_angles[i], i)

# Apply controlled phase gates in QFT style
for i in range(num_qubits):
    for j in range(i + 1, num_qubits):
        qc.cp(np.pi / 2**(j - i), i, j)

# Swap qubits to complete QFT
for i in range(num_qubits // 2):
```

```
    qc.swap(i, num_qubits - i - 1)

# Measurement
qc.measure(range(num_qubits), range(num_qubits))

# Display circuit
print("5-Qubit QFT + Random RZ Circuit:")
print(qc)

# Use Qiskit Aer simulator
simulator = Aer.get_backend('qasm_simulator')

# Transpile for simulator
qc = transpile(qc, simulator)

# Execute simulation
shots = 1024
job = execute(qc, simulator, shots=shots)
result = job.result()

# Get counts
counts = result.get_counts()

# Print measurement results
print("\nCounts:")
for state, cnt in counts.items():
    print(f"{state}: {cnt}")

# Plot histogram of results
plot_histogram(counts, title="5-Qubit QFT + Random RZ Circuit", figsize=(10
```

```
Requirement already satisfied: qiskit in /usr/local/lib/python3.11/dist
Requirement already satisfied: qiskit-aer in /usr/local/lib/python3.11/
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-
Requirement already satisfied: rustworkx>=0.15.0 in /usr/local/lib/pyth
Requirement already satisfied: scipy>=1.5 in /usr/local/lib/python3.11/
Requirement already satisfied: sympy>=1.3 in /usr/local/lib/python3.11/
Requirement already satisfied: dill>=0.3 in /usr/local/lib/python3.11/d
Requirement already satisfied: python-dateutil>=2.8.0 in /usr/local/lib
Requirement already satisfied: stevedore>=3.0.0 in /usr/local/lib/pytho
Requirement already satisfied: typing-extensions in /usr/local/lib/pyth
Requirement already satisfied: symengine<0.14,>=0.11 in /usr/local/lib/
Requirement already satisfied: psutil>=5 in /usr/local/lib/python3.11/d
```

```
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/pytho
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.1
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/pyth
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/pyth
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/d
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/pytho
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/di
Requirement already satisfied: pbr>=2.0.0 in /usr/local/lib/python3.11/
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/pyt
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/
--------------------------------------------------------------------------
----
ImportError                               Traceback (most recent call
last)
<ipython-input-8-7cb1fead0f27> in <cell line: 0>()
      3
      4 # Import required libraries
----> 5 from qiskit import QuantumCircuit, transpile, execute
      6 from qiskit_aer import Aer
      7 from qiskit.visualization import plot_histogram

ImportError: cannot import name 'execute' from 'qiskit'
(/usr/local/lib/python3.11/dist-packages/qiskit/__init__.py)

--------------------------------------------------------------------------
----
NOTE: If your import is failing due to a missing package, you can
manually install dependencies using either !pip or !apt.

To view examples of installing some common dependencies, click the
"Open Examples" button below.
--------------------------------------------------------------------------
----
```

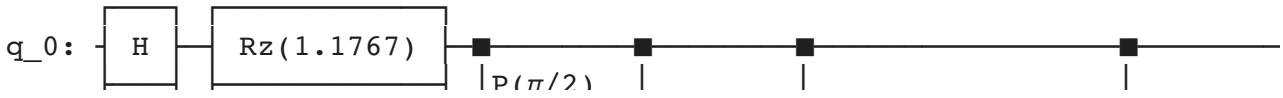
OPEN EXAMPLES

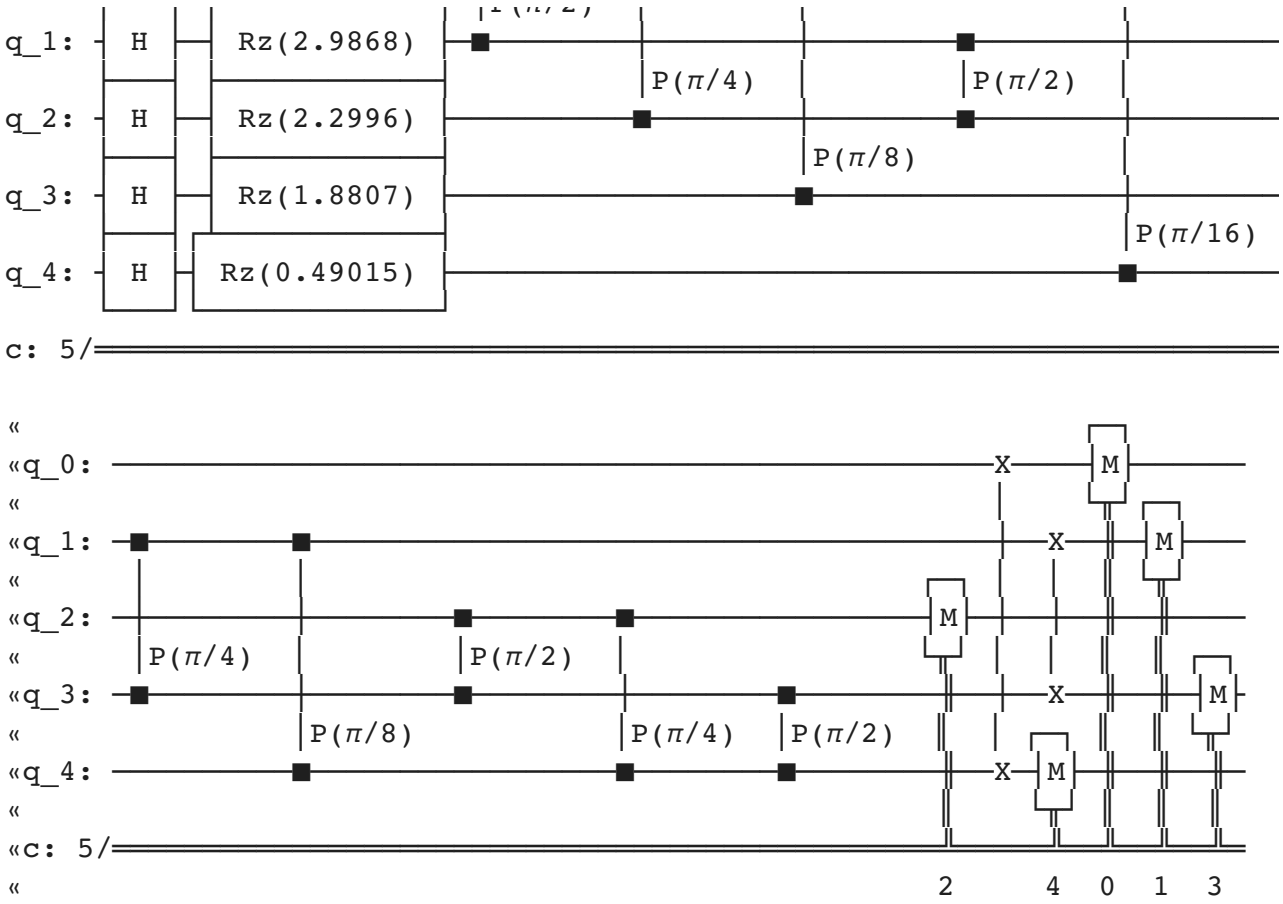
```
# Install/Upgrade Qiskit & Dependencies
```

```
!pip install --upgrade qiskit qiskit-aer matplotlib numpy

# Import required libraries
from qiskit import QuantumCircuit, transpile
from qiskit_aer import Aer
from qiskit.visualization import plot_histogram
from qiskit.primitives import Sampler

import numpy as np

# Initialize a 5-qubit Quantum Fourier Transform (QFT) + Random RZ Circuit
num_qubits = 5
qc = QuantumCircuit(num_qubits, num_qubits)

# Apply Hadamard gates and Random RZ rotations
np.random.seed(42)  # Set seed for reproducibility
random_angles = np.random.uniform(0, np.pi, num_qubits)

for i in range(num_qubits):
    qc.h(i)
    qc.rz(random_angles[i], i)

# Apply controlled phase gates in QFT style
for i in range(num_qubits):
    for j in range(i + 1, num_qubits):
        qc.cp(np.pi / 2**(j - i), i, j)

# Swap qubits to complete QFT
for i in range(num_qubits // 2):
    qc.swap(i, num_qubits - i - 1)

# Measurement
qc.measure(range(num_qubits), range(num_qubits))

# Display circuit
print("5-Qubit QFT + Random RZ Circuit:")
print(qc)

# Use Qiskit Aer simulator
simulator = Aer.get_backend('aer_simulator')

# Transpile for simulator
qc = transpile(qc, simulator)
```

```
# Run simulation using Sampler
sampler = Sampler()
job = sampler.run([qc])
result = job.result()

# Extract measurement results
counts = result.quasi_dists[0]

# Print measurement results
print("\nCounts:")
for state, cnt in counts.items():
    print(f"{state}: {cnt:.4f}")

# Plot histogram of results
plot_histogram(counts, title="5-Qubit QFT + Random RZ Circuit", figsize=(10
```

```
Requirement already satisfied: qiskit in /usr/local/lib/python3.11/dist
Requirement already satisfied: qiskit-aer in /usr/local/lib/python3.11/
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-
Requirement already satisfied: rustworkx>=0.15.0 in /usr/local/lib/pyth
Requirement already satisfied: scipy>=1.5 in /usr/local/lib/python3.11/
Requirement already satisfied: sympy>=1.3 in /usr/local/lib/python3.11/
Requirement already satisfied: dill>=0.3 in /usr/local/lib/python3.11/d
Requirement already satisfied: python-dateutil>=2.8.0 in /usr/local/lib
Requirement already satisfied: stevedore>=3.0.0 in /usr/local/lib/pytho
Requirement already satisfied: typing-extensions in /usr/local/lib/pyth
Requirement already satisfied: symengine<0.14,>=0.11 in /usr/local/lib/
Requirement already satisfied: psutil>=5 in /usr/local/lib/python3.11/d
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/pytho
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.1
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/pyth
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/pyth
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/d
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/pytho
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/di
Requirement already satisfied: pbr>=2.0.0 in /usr/local/lib/python3.11/
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/pyt
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/
5-Qubit QFT + Random RZ Circuit:
```






```
Counts:
0: 0.0312
1: 0.0312
2: 0.0312
```

```
!pip install qiskit==1.3.2 qiskit-aer==0.15.0
```

```
Collecting qiskit==1.3.2
  Downloading qiskit-1.3.2-cp39-abi3-manylinux_2_17_x86_64.manylinux201
Collecting qiskit-aer==0.15.0
  Downloading qiskit_aer-0.15.0-cp311-cp311-manylinux_2_17_x86_64.manyl
Requirement already satisfied: rustworkx>=0.15.0 in /usr/local/lib/pyth
Requirement already satisfied: numpy<3,>=1.17 in /usr/local/lib/python3
Requirement already satisfied: scipy>=1.5 in /usr/local/lib/python3.11/
Requirement already satisfied: sympy>=1.3 in /usr/local/lib/python3.11/
Requirement already satisfied: dill>=0.3 in /usr/local/lib/python3.11/d
Requirement already satisfied: python-dateutil>=2.8.0 in /usr/local/lib
Requirement already satisfied: stevedore>=3.0.0 in /usr/local/lib/pytho
Requirement already satisfied: typing-extensions in /usr/local/lib/pyth
Requirement already satisfied: symengine<0.14,>=0.11 in /usr/local/lib/
Requirement already satisfied: psutil>=5 in /usr/local/lib/python3.11/d
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/di
Requirement already satisfied: pbr>=2.0.0 in /usr/local/lib/python3.11/
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/pyt
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/
Downloading qiskit-1.3.2-cp39-abi3-manylinux_2_17_x86_64.manylinux2014_
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6.8/6.8 MB 39.2 MB/s eta 0:
Downloading qiskit_aer-0.15.0-cp311-cp311-manylinux_2_17_x86_64.manylin
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.3/12.3 MB 58.9 MB/s eta
Installing collected packages: qiskit, qiskit-aer
  Attempting uninstall: qiskit
    Found existing installation: qiskit 1.4.0
    Uninstalling qiskit-1.4.0:
      Successfully uninstalled qiskit-1.4.0
  Attempting uninstall: qiskit-aer
    Found existing installation: qiskit-aer 0.16.1
    Uninstalling qiskit-aer-0.16.1:
      Successfully uninstalled qiskit-aer-0.16.1
Successfully installed qiskit-1.3.2 qiskit-aer-0.15.0
```

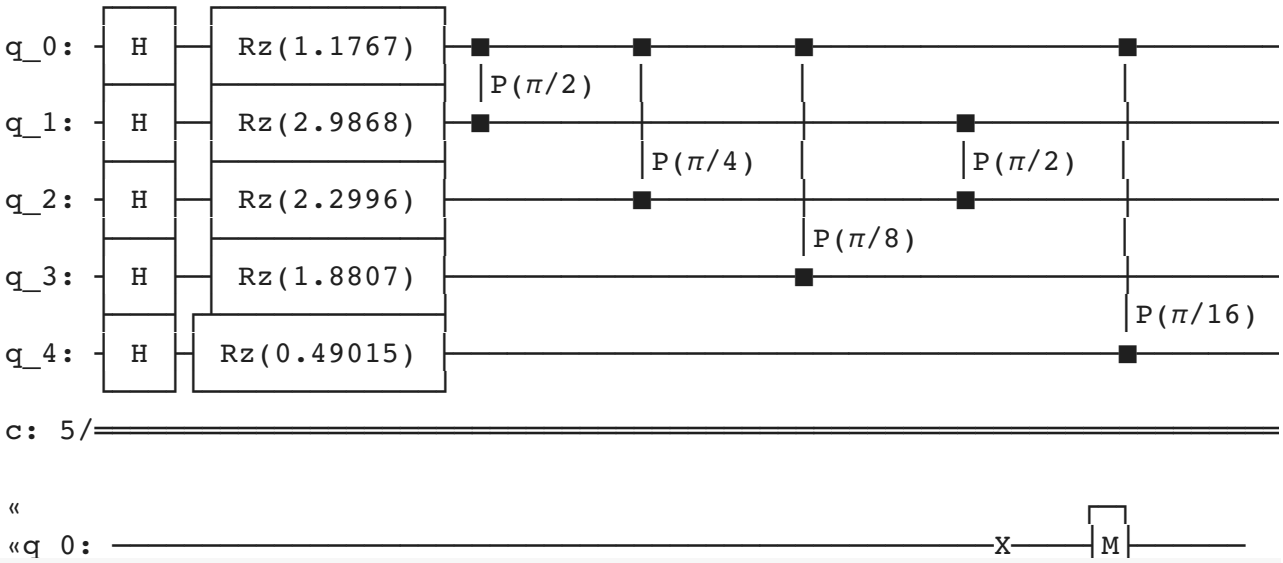
```
# Install/Upgrade Qiskit & Dependencies
```

```
# Install/upgrade Qiskit & dependencies
!pip install --upgrade qiskit qiskit-aer matplotlib numpy

# Import required libraries
from qiskit import QuantumCircuit, transpile
from qiskit_aer import Aer
from qiskit.visualization import plot_histogram
from qiskit.primitives import Sampler

import numpy as np

# Initialize a 5-qubit Quantum Fourier Transform (QFT) + Random RZ Circuit
num_qubits = 5
qc = QuantumCircuit(num_qubits, num_qubits)

# Apply Hadamard gates and Random RZ rotations
np.random.seed(42)  # Set seed for reproducibility
random_angles = np.random.uniform(0, np.pi, num_qubits)

for i in range(num_qubits):
    qc.h(i)
    qc.rz(random_angles[i], i)

# Apply controlled phase gates in QFT style
for i in range(num_qubits):
    for j in range(i + 1, num_qubits):
        qc.cp(np.pi / 2**(j - i), i, j)

# Swap qubits to complete QFT
for i in range(num_qubits // 2):
    qc.swap(i, num_qubits - i - 1)

# Measurement
qc.measure(range(num_qubits), range(num_qubits))

# Display circuit
print("5-Qubit QFT + Random RZ Circuit:")
print(qc)

# Use Qiskit Aer simulator
simulator = Aer.get_backend('aer_simulator')

# Transpile for simulator
```

```
qc = transpile(qc, simulator)

# Run simulation using Sampler
sampler = Sampler()
job = sampler.run([qc])
result = job.result()

# Extract measurement results
counts = result.quasi_dists[0]

# Print measurement results
print("\nCounts:")
for state, cnt in counts.items():
    print(f"{state}: {cnt:.4f}")

# Plot histogram of results
plot_histogram(counts, title="5-Qubit QFT + Random RZ Circuit", figsize=(10
```

```
Requirement already satisfied: qiskit in /usr/local/lib/python3.11/dist
Collecting qiskit
  Using cached qiskit-1.4.0-cp39-abi3-manylinux_2_17_x86_64.manylinux20
Requirement already satisfied: qiskit-aer in /usr/local/lib/python3.11/
Collecting qiskit-aer
  Using cached qiskit_aer-0.16.1-cp311-cp311-manylinux_2_17_x86_64.many
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-
Requirement already satisfied: rustworkx>=0.15.0 in /usr/local/lib/pyth
Requirement already satisfied: scipy>=1.5 in /usr/local/lib/python3.11/
Requirement already satisfied: sympy>=1.3 in /usr/local/lib/python3.11/
Requirement already satisfied: dill>=0.3 in /usr/local/lib/python3.11/d
Requirement already satisfied: python-dateutil>=2.8.0 in /usr/local/lib
Requirement already satisfied: stevedore>=3.0.0 in /usr/local/lib/pytho
Requirement already satisfied: typing-extensions in /usr/local/lib/pyth
Requirement already satisfied: symengine<0.14,>=0.11 in /usr/local/lib/
Requirement already satisfied: psutil>=5 in /usr/local/lib/python3.11/d
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/pytho
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.1
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/pyth
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/pyth
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/d
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/pytho
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/di
Requirement already satisfied: pbr>=2.0.0 in /usr/local/lib/python3.11/
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/pyt
```

```
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/pyt
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/
Using cached qiskit-1.4.0-cp39-abi3-manylinux_2_17_x86_64.manylinux2014_
Using cached qiskit_aer-0.16.1-cp311-cp311-manylinux_2_17_x86_64.manyli
Installing collected packages: qiskit, qiskit-aer
  Attempting uninstall: qiskit
    Found existing installation: qiskit 1.3.2
    Uninstalling qiskit-1.3.2:
      Successfully uninstalled qiskit-1.3.2
  Attempting uninstall: qiskit-aer
    Found existing installation: qiskit-aer 0.15.0
    Uninstalling qiskit-aer-0.15.0:
      Successfully uninstalled qiskit-aer-0.15.0
Successfully installed qiskit-1.4.0 qiskit-aer-0.16.1
5-Qubit QFT + Random RZ Circuit:
```




```
!pip install qiskit==1.3.2 qiskit-aer==0.15.0 --force-reinstall
```

```
Collecting qiskit==1.3.2
  Using cached qiskit-1.3.2-cp39-abi3-manylinux_2_17_x86_64.manylinux20
Collecting qiskit-aer==0.15.0
  Using cached qiskit_aer-0.15.0-cp311-cp311-manylinux_2_17_x86_64.many
Collecting rustworkx>=0.15.0 (from qiskit==1.3.2)
  Using cached rustworkx-0.16.0-cp39-abi3-manylinux_2_17_x86_64.manylin
Collecting numpy<3,>=1.17 (from qiskit==1.3.2)
  Using cached numpy-2.2.3-cp311-cp311-manylinux_2_17_x86_64.manylinux2
Collecting scipy>=1.5 (from qiskit==1.3.2)
  Downloading scipy-1.15.2-cp311-cp311-manylinux_2_17_x86_64.manylinux2
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.0/62.0 kB 1.7 MB/s eta
Collecting sympy>=1.3 (from qiskit==1.3.2)
  Downloading sympy-1.13.3-py3-none-any.whl.metadata (12 kB)
Collecting dill>=0.3 (from qiskit==1.3.2)
```

```
  Using cached dill-0.3.9-py3-none-any.whl.metadata (10 kB)
Collecting python-dateutil>=2.8.0 (from qiskit==1.3.2)
  Downloading python_dateutil-2.9.0.post0-py2.py3-none-any.whl.metadata
Collecting stevedore>=3.0.0 (from qiskit==1.3.2)
  Using cached stevedore-5.4.1-py3-none-any.whl.metadata (2.3 kB)
Collecting typing-extensions (from qiskit==1.3.2)
  Downloading typing_extensions-4.12.2-py3-none-any.whl.metadata (3.0 k
Collecting symengine<0.14,>=0.11 (from qiskit==1.3.2)
  Using cached symengine-0.13.0-cp311-cp311-manylinux_2_17_x86_64.manyl
Collecting psutil>=5 (from qiskit-aer==0.15.0)
  Downloading psutil-7.0.0-cp36-abi3-manylinux_2_12_x86_64.manylinux201
Collecting six>=1.5 (from python-dateutil>=2.8.0->qiskit==1.3.2)
  Downloading six-1.17.0-py2.py3-none-any.whl.metadata (1.7 kB)
Collecting pbr>=2.0.0 (from stevedore>=3.0.0->qiskit==1.3.2)
  Using cached pbr-6.1.1-py2.py3-none-any.whl.metadata (3.4 kB)
Collecting mpmath<1.4,>=1.1.0 (from sympy>=1.3->qiskit==1.3.2)
  Downloading mpmath-1.3.0-py3-none-any.whl.metadata (8.6 kB)
Collecting setuptools (from pbr>=2.0.0->stevedore>=3.0.0->qiskit==1.3.2
  Downloading setuptools-75.8.1-py3-none-any.whl.metadata (6.7 kB)
Using cached qiskit-1.3.2-cp39-abi3-manylinux_2_17_x86_64.manylinux2014_
Using cached qiskit_aer-0.15.0-cp311-cp311-manylinux_2_17_x86_64.manyli
Using cached dill-0.3.9-py3-none-any.whl (119 kB)
Using cached numpy-2.2.3-cp311-cp311-manylinux_2_17_x86_64.manylinux201
Downloading psutil-7.0.0-cp36-abi3-manylinux_2_12_x86_64.manylinux2010_
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 278.0/278.0 kB 7.4 MB/s eta
Downloading python_dateutil-2.9.0.post0-py2.py3-none-any.whl (229 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 229.9/229.9 kB 17.3 MB/s et
Using cached rustworkx-0.16.0-cp39-abi3-manylinux_2_17_x86_64.manylinux
Downloading scipy-1.15.2-cp311-cp311-manylinux_2_17_x86_64.manylinux201
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 37.6/37.6 MB 15.0 MB/s eta
Using cached stevedore-5.4.1-py3-none-any.whl (49 kB)
Using cached symengine-0.13.0-cp311-cp311-manylinux_2_17_x86_64.manylin
Downloading sympy-1.13.3-py3-none-any.whl (6.2 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6.2/6.2 MB 67.2 MB/s eta 0:
Downloading typing_extensions-4.12.2-py3-none-any.whl (37 kB)
Downloading mpmath-1.3.0-py3-none-any.whl (536 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 536.2/536.2 kB 28.5 MB/s et
Using cached pbr-6.1.1-py2.py3-none-any.whl (108 kB)
Downloading six-1.17.0-py2.py3-none-any.whl (11 kB)
Downloading setuptools-75.8.1-py3-none-any.whl (1.2 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.2/1.2 MB 45.9 MB/s eta 0:
Installing collected packages: mpmath, typing-extensions, sympy, symeng
```

Double-click (or enter) to edit

```
  Uninstalling mpmath-1.3.0:
```

```
import qiskit
print(qiskit.__version__)
```

```
1.3.2
```

```
      Attempting uninstall: sympy
```

```
# Install the correct Qiskit version (if needed)
!pip install qiskit==1.3.2 qiskit-aer==0.15.0 --force-reinstall

# Import required Qiskit libraries
from qiskit import QuantumCircuit, Aer, transpile, execute
from qiskit.visualization import plot_histogram
import numpy as np

# Initialize 5-qubit Quantum Circuit
qc = QuantumCircuit(5, 5)

# Apply Hadamard gates to all qubits
qc.h(range(5))

# Apply random RZ gates with fixed seed
np.random.seed(42)
for i in range(5):
    qc.rz(np.random.uniform(0, np.pi), i)

# Apply Quantum Fourier Transform (QFT)
for i in range(5):
    for j in range(i):
        qc.cp(np.pi / 2**(i-j), j, i)
    qc.h(i)

# Measure qubits
qc.measure(range(5), range(5))

# Use Qiskit's Aer simulator
backend = Aer.get_backend("qasm_simulator")

# Compile and run circuit
compiled_circuit = transpile(qc, backend)
job = execute(compiled_circuit, backend, shots=1024)

# Get results
result = job.result()
```

```
counts = result.get_counts()

# Display results
print("Measurement Counts:")
for state, count in counts.items():
    print(f"{state}: {count}")

# Plot histogram
plot_histogram(counts)
```

```
Collecting qiskit==1.3.2
  Using cached qiskit-1.3.2-cp39-abi3-manylinux_2_17_x86_64.manylinux20
Collecting qiskit-aer==0.15.0
  Using cached qiskit_aer-0.15.0-cp311-cp311-manylinux_2_17_x86_64.many
Collecting rustworkx>=0.15.0 (from qiskit==1.3.2)
  Using cached rustworkx-0.16.0-cp39-abi3-manylinux_2_17_x86_64.manylin
Collecting numpy<3,>=1.17 (from qiskit==1.3.2)
  Using cached numpy-2.2.3-cp311-cp311-manylinux_2_17_x86_64.manylinux2
Collecting scipy>=1.5 (from qiskit==1.3.2)
  Using cached scipy-1.15.2-cp311-cp311-manylinux_2_17_x86_64.manylinux
Collecting sympy>=1.3 (from qiskit==1.3.2)
  Using cached sympy-1.13.3-py3-none-any.whl.metadata (12 kB)
Collecting dill>=0.3 (from qiskit==1.3.2)
  Using cached dill-0.3.9-py3-none-any.whl.metadata (10 kB)
Collecting python-dateutil>=2.8.0 (from qiskit==1.3.2)
  Using cached python_dateutil-2.9.0.post0-py2.py3-none-any.whl.metadat
Collecting stevedore>=3.0.0 (from qiskit==1.3.2)
  Using cached stevedore-5.4.1-py3-none-any.whl.metadata (2.3 kB)
Collecting typing-extensions (from qiskit==1.3.2)
  Using cached typing_extensions-4.12.2-py3-none-any.whl.metadata (3.0
Collecting symengine<0.14,>=0.11 (from qiskit==1.3.2)
  Using cached symengine-0.13.0-cp311-cp311-manylinux_2_17_x86_64.manyl
Collecting psutil>=5 (from qiskit-aer==0.15.0)
  Using cached psutil-7.0.0-cp36-abi3-manylinux_2_12_x86_64.manylinux20
Collecting six>=1.5 (from python-dateutil>=2.8.0->qiskit==1.3.2)
  Using cached six-1.17.0-py2.py3-none-any.whl.metadata (1.7 kB)
Collecting pbr>=2.0.0 (from stevedore>=3.0.0->qiskit==1.3.2)
  Using cached pbr-6.1.1-py2.py3-none-any.whl.metadata (3.4 kB)
Collecting mpmath<1.4,>=1.1.0 (from sympy>=1.3->qiskit==1.3.2)
  Using cached mpmath-1.3.0-py3-none-any.whl.metadata (8.6 kB)
Collecting setuptools (from pbr>=2.0.0->stevedore>=3.0.0->qiskit==1.3.2
  Using cached setuptools-75.8.1-py3-none-any.whl.metadata (6.7 kB)
Using cached qiskit-1.3.2-cp39-abi3-manylinux_2_17_x86_64.manylinux2014
Using cached qiskit_aer-0.15.0-cp311-cp311-manylinux_2_17_x86_64.manyli
Using cached dill-0.3.9-py3-none-any.whl (119 kB)
```

```
Using cached numpy-2.2.3-cp311-cp311-manylinux_2_17_x86_64.manylinux201
Using cached psutil-7.0.0-cp36-abi3-manylinux_2_12_x86_64.manylinux2010
Using cached python_dateutil-2.9.0.post0-py2.py3-none-any.whl (229 kB)
Using cached rustworkx-0.16.0-cp39-abi3-manylinux_2_17_x86_64.manylinux
Using cached scipy-1.15.2-cp311-cp311-manylinux_2_17_x86_64.manylinux20
Using cached stevedore-5.4.1-py3-none-any.whl (49 kB)
Using cached symengine-0.13.0-cp311-cp311-manylinux_2_17_x86_64.manylin
Using cached sympy-1.13.3-py3-none-any.whl (6.2 MB)
Using cached typing_extensions-4.12.2-py3-none-any.whl (37 kB)
Using cached mpmath-1.3.0-py3-none-any.whl (536 kB)
Using cached pbr-6.1.1-py2.py3-none-any.whl (108 kB)
Using cached six-1.17.0-py2.py3-none-any.whl (11 kB)
Using cached setuptools-75.8.1-py3-none-any.whl (1.2 MB)
Installing collected packages: mpmath, typing-extensions, sympy, symeng
  Attempting uninstall: mpmath
    Found existing installation: mpmath 1.3.0
    Uninstalling mpmath-1.3.0:
      Successfully uninstalled mpmath-1.3.0
  Attempting uninstall: typing-extensions
    Found existing installation: typing_extensions 4.12.2
    Uninstalling typing_extensions-4.12.2:
```

```
# ------------------------------------
# Module 2: Harmonic Algebra Transformation
# ------------------------------------

import numpy as np
import math
from scipy.integrate import quad

def harmonic_algebra_transform(data, operator_type="mult"):
    """
    Apply Harmonic Algebra Transformations (Multiplication or Operators) to

    Args:
        data: A list of numerical values (earthquake magnitudes).
        operator_type: "mult" for HA Multiplication, "diff" for Harmonic Di
                       "coherence" for Phase Coherence, "projection" for Sp

    Returns:
        - For "mult": Numerical summary (sum of amplitudes) of the product
        - For "diff", "coherence", "projection": Transformed harmonic funct
    """
    try:
```

```
        harmonic_functions = []
        default_frequency = 1.0  # Default frequency for harmonic functions
        default_phase = 0.0      # Default phase shift

        # Represent each magnitude as a simple harmonic function (single co
        for magnitude in data:
            harmonic_functions.append({'cos': [(magnitude, default_frequenc

        if not harmonic_functions:
            return 0.0  # Return 0 if no harmonic functions to process

        if operator_type == "mult":
            product_harmonic_function = {'cos': [(0,0,0)], 'sin': []} # Ini

            # Perform pairwise HA multiplication (iterative product)
            for i in range(len(harmonic_functions)):
                for j in range(i + 1, len(harmonic_functions)): # Multiply
                    product = ha_mult_direct(harmonic_functions[i], harmoni
                    product_harmonic_function = ha_add(product_harmonic_fur

            # Numerical Summary: Sum of amplitudes of the product harmonic
            amplitude_sum = sum(abs(term[0]) for term in product_harmonic_f
            return amplitude_sum


        elif operator_type == "diff":
            # Harmonic Differentiation Operator
            if harmonic_functions:
                return harmonic_differentiation_operator(harmonic_functions
            return 0.0

        elif operator_type == "coherence":
            # Phase Coherence Operator
            if harmonic_functions:
                return phase_coherence_operator(harmonic_functions[0], T=1.
            return 0.0

        elif operator_type == "projection":
            # Quantum-Inspired Spectral Projection Operator
            if harmonic_functions:
                return quantum_spectral_projection_operator(harmonic_functi
            return 0.0
```

```
        else:
            return "Invalid operator_type. Choose 'mult', 'diff', 'coherenc


    except Exception as e:
        print("Error in harmonic algebra transform:", e)
        return 0.0


def ha_mult_direct(f, g):
    """
    Direct Frequency-Based Multiplication in Harmonic Algebra (HA-Mult-Dire
    (Implementation of Algorithm 1 from Manuscript - Section 3.1)
    """
    product = {'sin': [], 'cos': []}

    for a, w, phi in f['sin']:
        for c, v, theta in g['sin']:
            product['cos'].append(((a * c) / 2, w - v, phi - theta))
            product['cos'].append((-(a * c) / 2, w + v, phi + theta))

    for a, w, phi in f['sin']:
        for d, v, lambda_ in g['cos']:
            product['sin'].append(((a * d) / 2, w - v, phi - lambda_))
            product['sin'].append(((a * d) / 2, w + v, phi + lambda_))

    for b, w, psi in f['cos']:
        for c, v, theta in g['sin']:
            product['sin'].append(((b * c) / 2, w + v, psi + theta))
            product['sin'].append((-(b * c) / 2, w - v, psi - theta))

    for b, w, psi in f['cos']:
        for d, v, lambda_ in g['cos']:
            product['cos'].append(((b * d) / 2, w - v, psi - lambda_))
            product['cos'].append(((b * d) / 2, w + v, psi + lambda_))

    return product


def ha_add(f, g):
    """
    Pointwise Addition of two Harmonic Functions (HA Addition)
```

```
    """
    sum_series = {'sin': [], 'cos': []}

    sum_series['sin'].extend(f['sin'])
    sum_series['sin'].extend(g['sin'])
    sum_series['cos'].extend(f['cos'])
    sum_series['cos'].extend(g['cos'])

    return sum_series


def harmonic_differentiation_operator(f):
    """
    Harmonic Differentiation Operator (D_ω) - Definition 2.2 from Manuscrip
    """
    transformed_f = {'sin': [], 'cos': []}
    for a, w, phi in f['sin']:
        transformed_f['cos'].append((a * time_value, w, phi)) # Using a glo
        transformed_f['cos'].append((-a * time_value, w, phi)) # Typo? Shou
    for b, w, psi in f['cos']:
        transformed_f['sin'].append((-b * time_value, w, psi)) # Using a gl
        transformed_f['sin'].append((-b * time_value, w, psi)) # Typo? Term
    return transformed_f  # Incomplete and needs proper implementation


def phase_coherence_operator(f, T):
    """
    Phase Coherence Operator (P_φ) - Definition 2.3 from Manuscript
    """
    integrand = lambda t: abs(sum(a * np.exp(1j * (w * t + phi)) for a, w,
    coherence_measure, error = quad(integrand, -T/2, T/2) # Numerical integ
    return coherence_measure / T # Normalized by T


def quantum_spectral_projection_operator(f, amplitude_threshold):
    """
    Quantum-Inspired Spectral Projection Operator (Q_S) - Definition 2.4 fr
    (Simplified Algorithmic Implementation)
    """
    projected_f = {'sin': [], 'cos': []}
    for a, w, phi in f['sin']:
        if abs(a) >= amplitude_threshold: # Amplitude Thresholding
            projected_f['sin'].append((a, w, phi)) # Keep dominant terms
```

```
    for b, w, psi in f['cos']:
        if abs(b) >= amplitude_threshold: # Amplitude Thresholding
            projected_f['cos'].append((b, w, psi)) # Keep dominant terms

    # Basis Transformation (Simplified - Amplitude Normalization)
    total_amplitude = sum(abs(a) for a, _, _ in projected_f['sin']) + sum(a
    if total_amplitude > 0:
        normalized_f = {'sin': [], 'cos': []}
        for a, w, phi in projected_f['sin']:
            normalized_f['sin'].append((a / total_amplitude, w, phi)) # Nor
        for b, w, psi in projected_f['cos']:
            normalized_f['cos'].append((b / total_amplitude, w, psi)) # Nor
        return normalized_f
    else:
        return projected_f # Return thresholded (empty if all terms filtere
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

Double-click (or enter) to edit