2025 Quantum Hackathon
Quantum Algorithms :

Quantum Phase Estimation

QUICK Seonggeun Park, Juhyun Song, Jaemin Jeon, Sungheum Yoo, Yoonha Hwang

DATE July 1st, 2025

Overview

Quantum Phase Estimation

Practical implementation of QPE requires translating the mathematical descriptions of the unitary operator and input state into executable quantum circuits, which is often a challenging task in practice

Contents

2025 Quantum Hackathon

Problem 1

QPE Implementation

Given the unitary operator and the eigen vector, implement the QPE algorithm to estimate the phase in the following cases. For each case, analyze how increasing the number of qubits used for the estimation affects the precision of the output

$$U = \begin{bmatrix} 1 & 0 \\ 0 & e^{2\pi i\theta} \end{bmatrix} \quad v_1 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad \theta_1 = \frac{5}{8} \quad \theta_2 = \frac{3}{7}$$

QPE Circuit

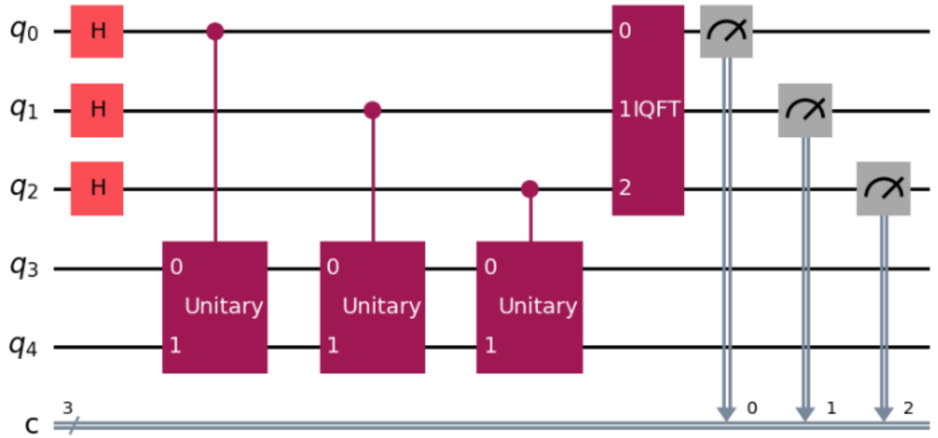
```python
def create_qpe_circuit(num_ancilla: int,
                       U_gate: UnitaryGate,
                       state_prep: QuantumCircuit) -> QuantumCircuit:
    n_target = state_prep.num_qubits
    U_mat    = U_gate.to_matrix()
    qc = QuantumCircuit(num_ancilla + n_target, num_ancilla)
    qc.h(range(num_ancilla))
    qc.compose(state_prep,
               qubits=list(range(num_ancilla,
                                 num_ancilla+n_target)),
               inplace=True)
    for j in range(num_ancilla):
        mat_pow = np.linalg.matrix_power(U_mat, 2**j)
        qc.append(UnitaryGate(mat_pow).control(),
                  [j] + list(range(num_ancilla,
                                   num_ancilla+n_target)))
    qc.compose(QFT(num_ancilla, do_swaps=True).inverse(),
               range(num_ancilla), inplace=True)
    qc.measure(range(num_ancilla), range(num_ancilla))
    return qc
```

Python

$$U|\psi\rangle = e^{2\pi i\phi}\ |\psi\rangle$$

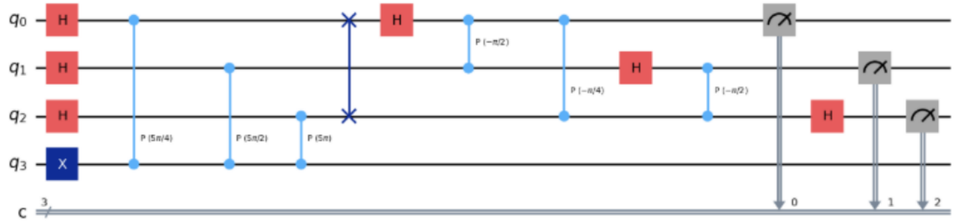
$$\frac{1}{\sqrt{2^t}}\sum_{k=0}^{2^t-1}|k\rangle\otimes U^k|\psi\rangle = \frac{1}{\sqrt{2^t}}\sum_{k=0}^{2^t-1}e^{2\pi i\phi k}|k\rangle\otimes|\psi\rangle$$

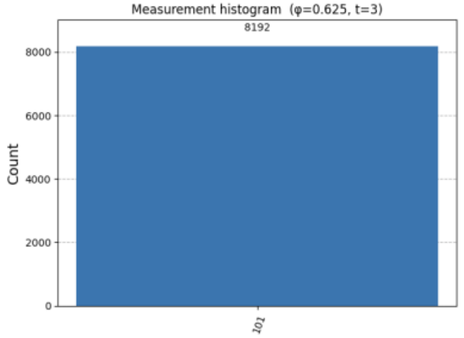
$$\frac{1}{\sqrt{2^t}}\sum_{k=0}^{2^t-1}e^{2\pi i\phi k}|k\rangle\ - QFT^\dagger \longrightarrow |\tilde{\phi}\rangle$$



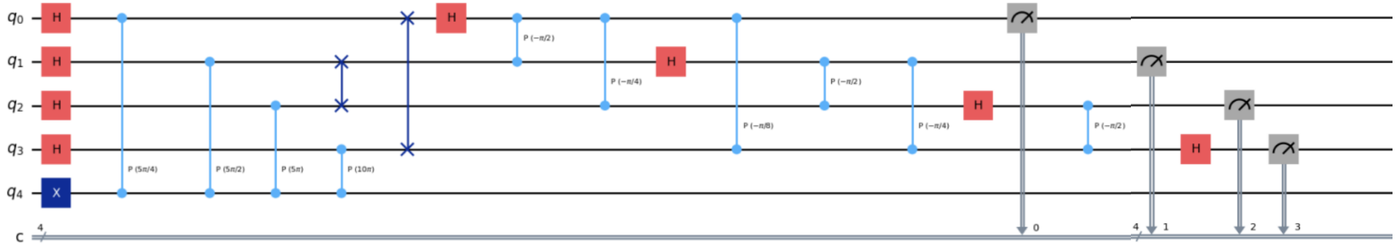
θ = 5/8

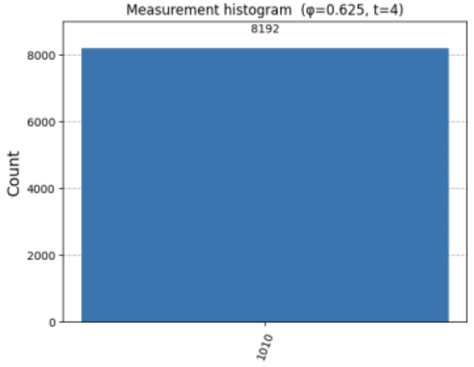
t=3


q0
q1
q2
q3
c
H
X
P (5π/4)
P (5π/2)
P (5π)
P (−π/2)
P (−π/4)
P (−π/2)
3
0
1
2

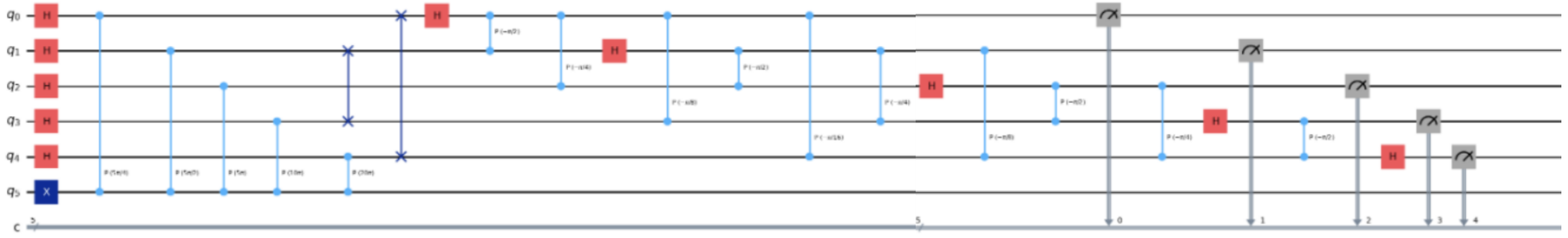

Measurement histogram (φ=0.625, t=3)
8192
Count
8000
6000
4000
2000
0
101

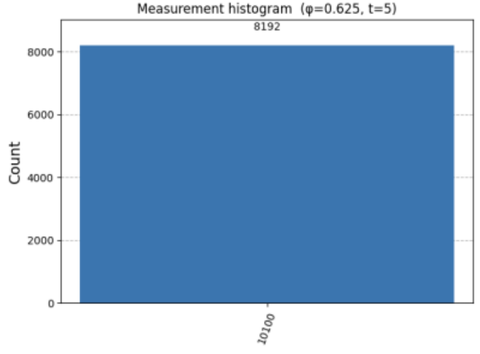
t=4


q0
q1
q2
q3
q4
c
H
X
P (5π/4)
P (5π/2)
P (5π)
P (10π)
P (−π/2)
P (−π/4)
P (−π/8)
P (−π/2)
P (−π/4)
P (−π/2)
4
0
1
2
3

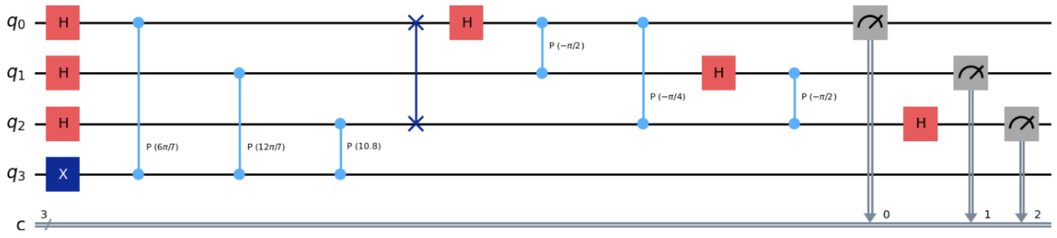

Measurement histogram (φ=0.625, t=4)
8192
Count
8000
6000
4000
2000
0
1010

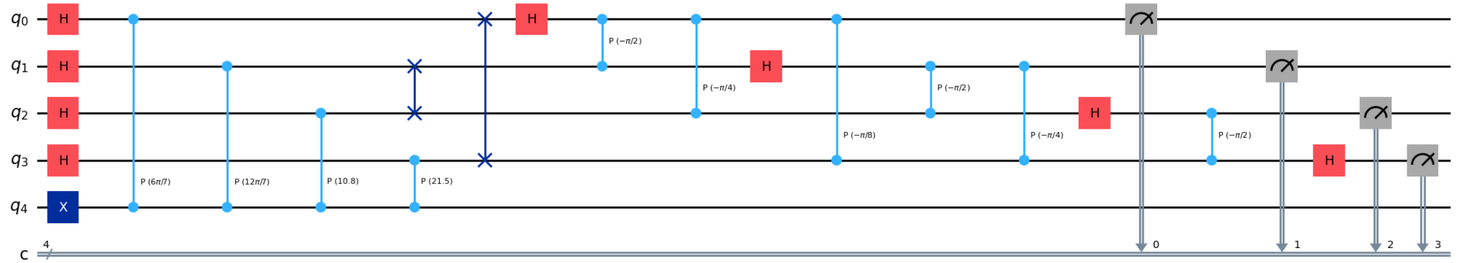
t=5

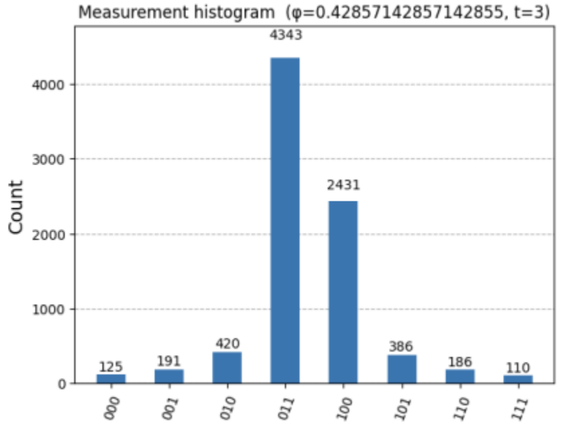

q0
q1
q2
q3
q4
q5
c
H
X
5
0
1
2
3
4


Measurement histogram (φ=0.625, t=5)
8192
Count
8000
6000
4000
2000
0
10100

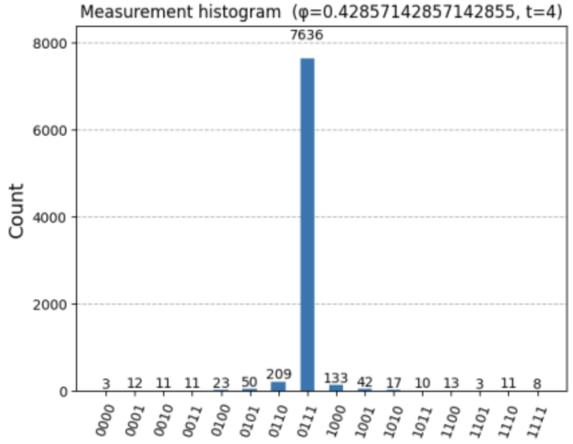
θ = 3/7



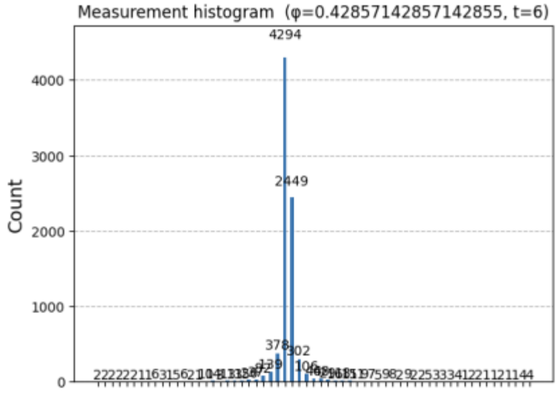




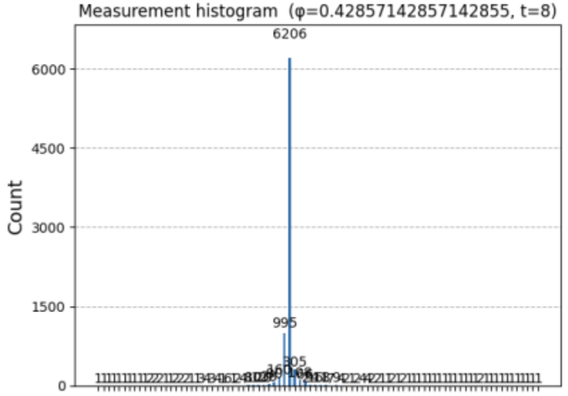
t=3



t=4

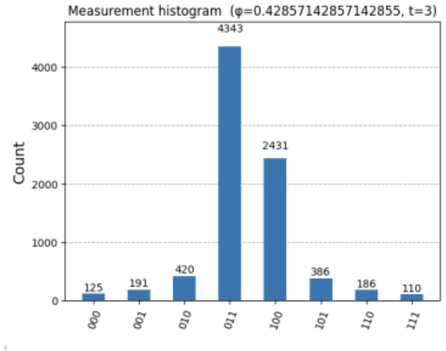


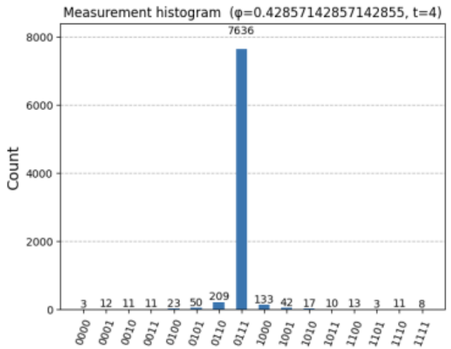
t=6

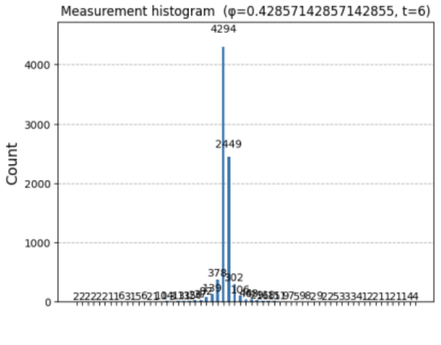


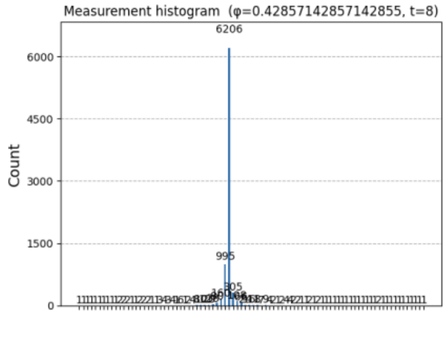
t=8

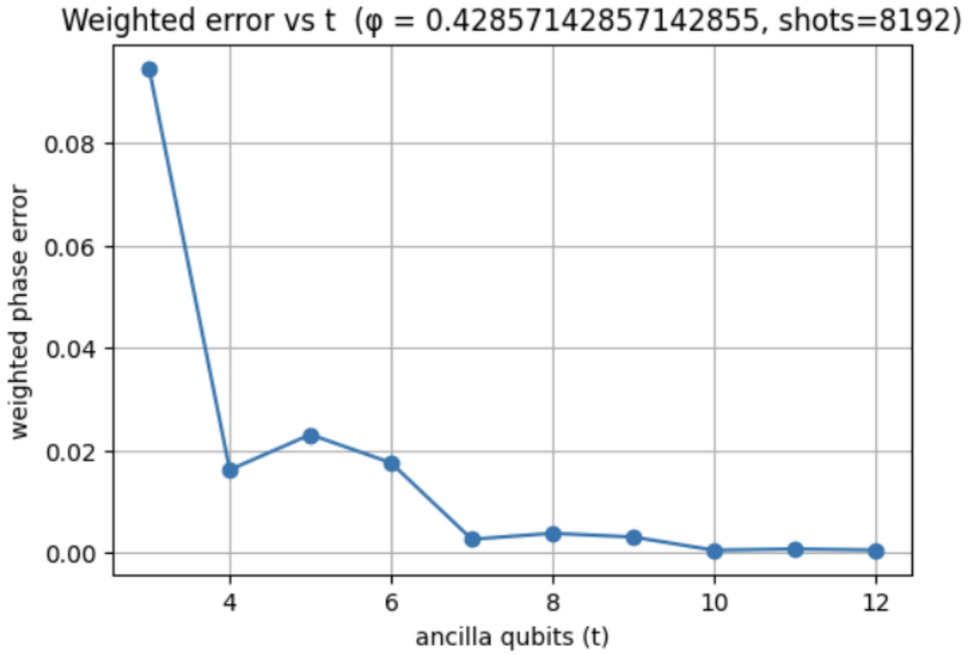
θ = 3/7











$$\text{Weighted Error} = \frac{1}{N}\sum_{k}\left|\hat{\phi}_k - \phi\right| \cdot c_k$$

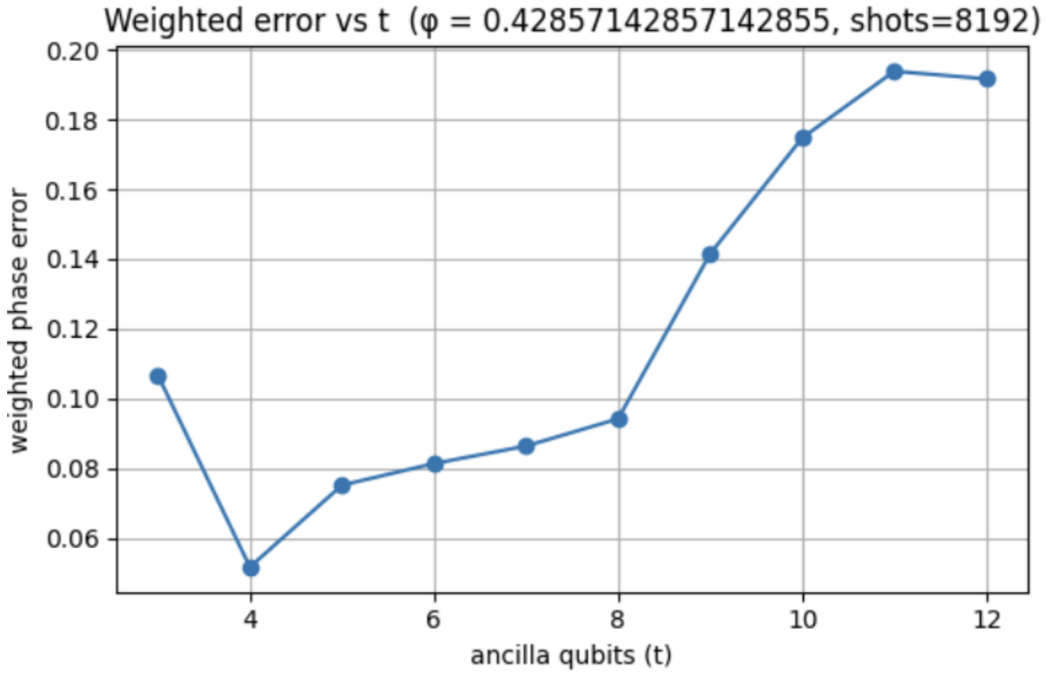
N : total number of shots

$\hat{\phi}_k$: Estimated phase from bitstring

c_k : count of how often outcome occurred

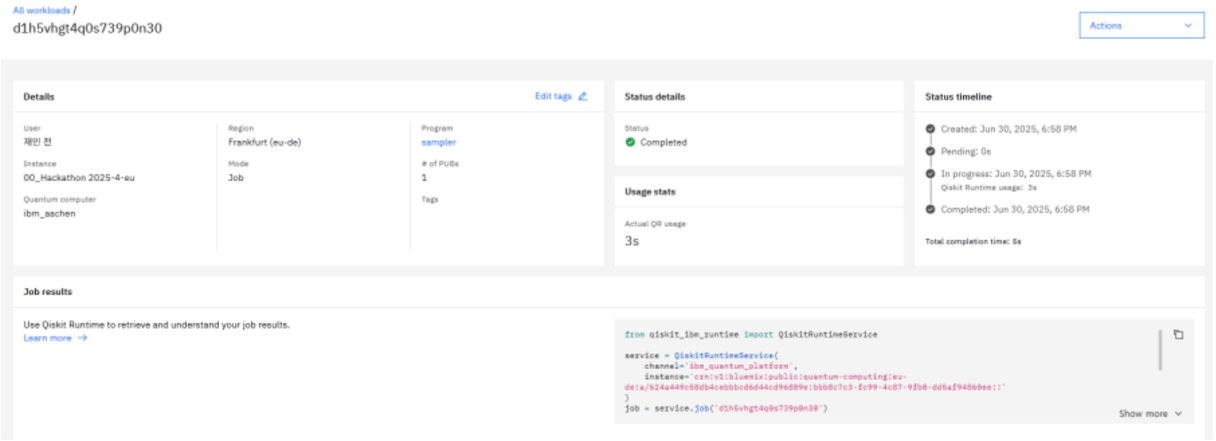
ϕ : true phase

Evaluation on Real Quantum Hardware




```python
service  = QiskitRuntimeService()
backend  = service.least_busy(simulator=False, operational=True)
print(f"© Real backend = {backend.name}  ({backend.num_qubits} qubits)")

sampler  = Sampler(backend)
```

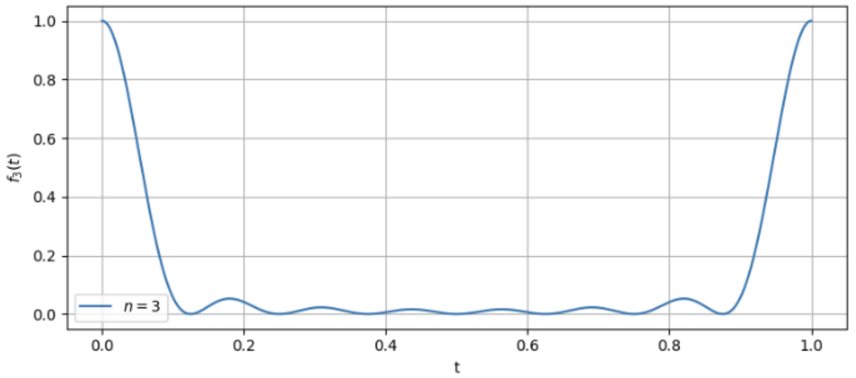
Python

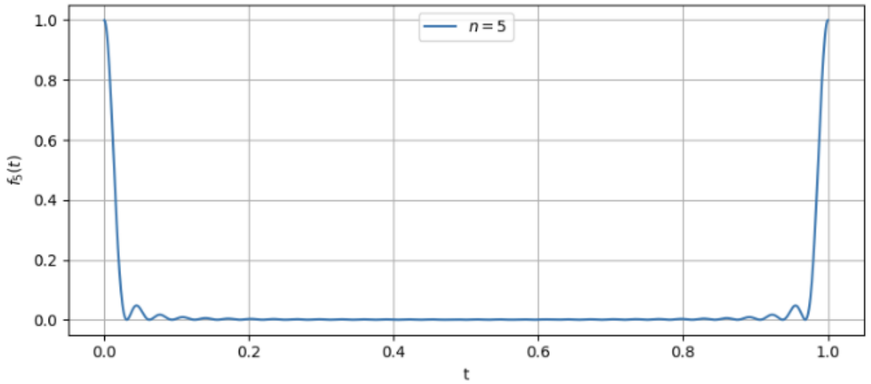


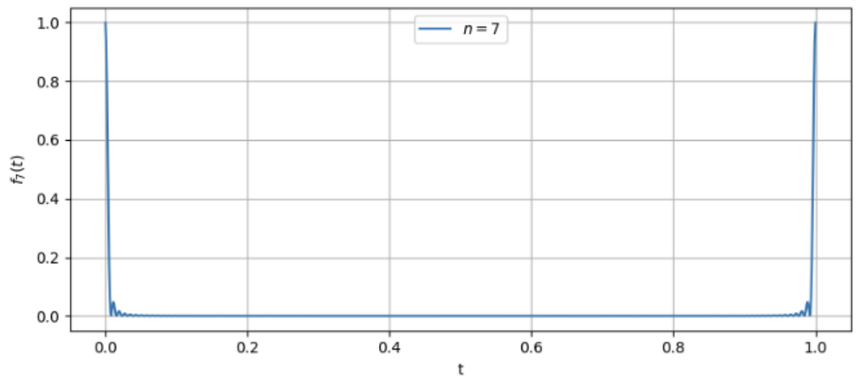
Conclusion

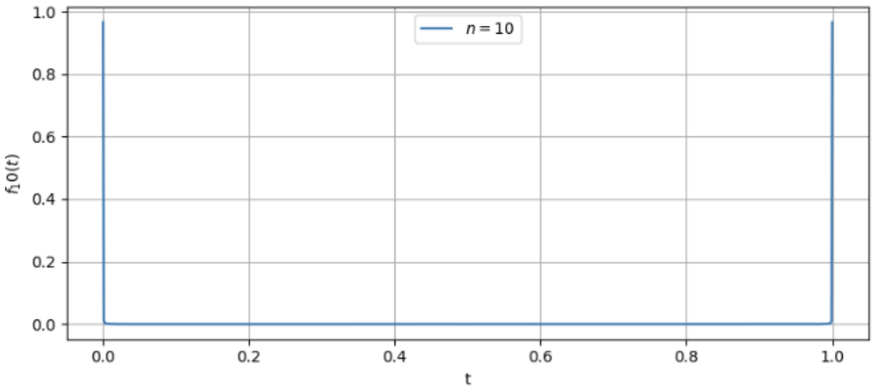
$$f_n(t) = \frac{1}{4^n} \cdot \frac{1 - \cos(2\pi 2^n t)}{1 - \cos(2\pi t)}$$

Increasing the number of estimation qubits leads to higher precision in phase estimation









2025 Quantum Hackathon

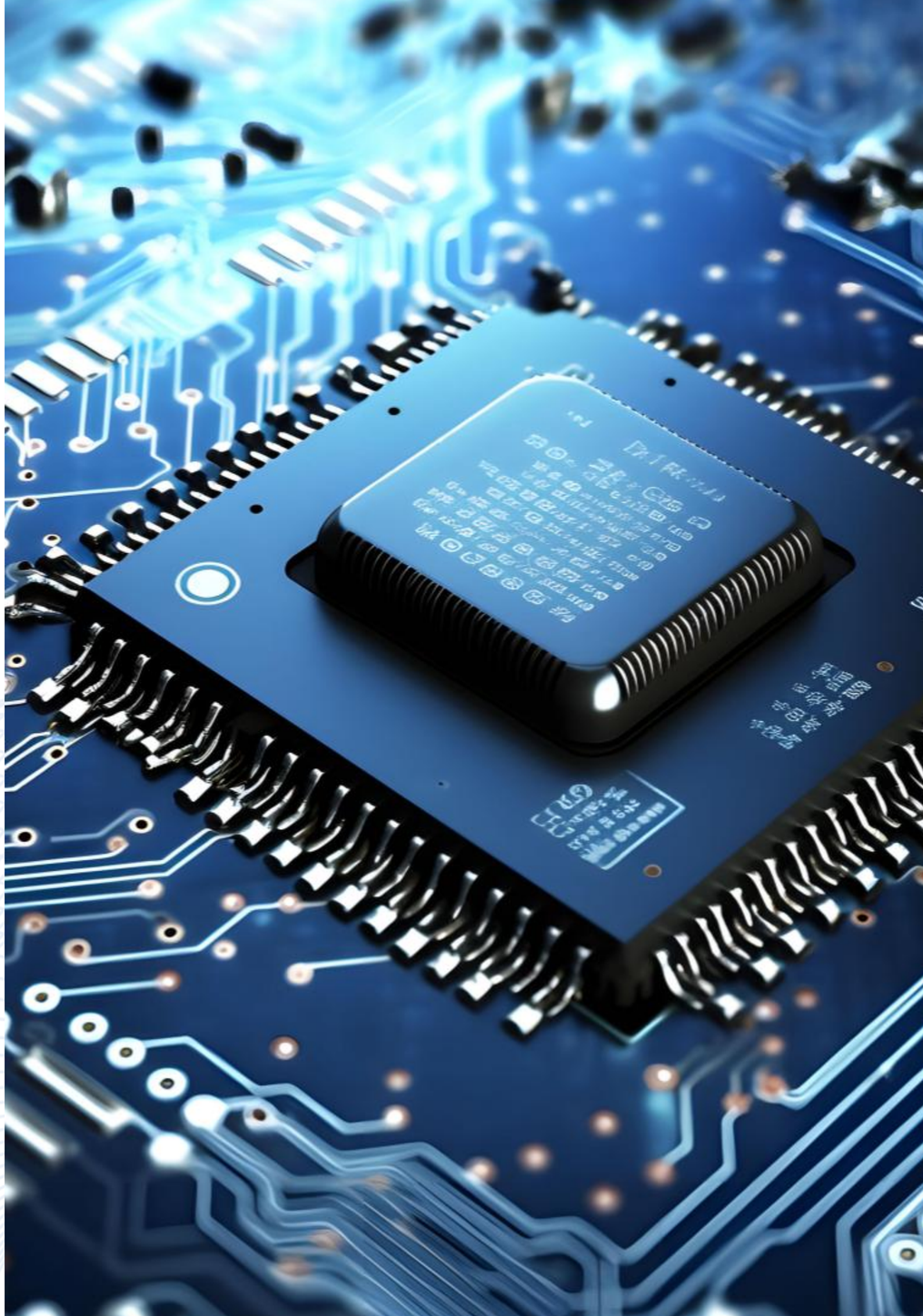
Problem 2

Finding Eigenvalues of a Matrix using QPE

Estimate all eigenvalues of M using only the QPE algorithm

- Handle non-positive definite matrices
- Bound the spectral norm
- Prefer manual decomposition
- No prior eigenvector knowledge

$$M = \begin{bmatrix} 1 & 0 & 8 & 1 \\ 0 & 1 & 1 & 8 \\ 8 & 1 & 1 & 0 \\ 1 & 8 & 0 & 1 \end{bmatrix}$$



Pauli Decomposition

two-qubit system

$$M = \begin{bmatrix} 1 & 0 & 8 & 1 \\ 0 & 1 & 1 & 8 \\ 8 & 1 & 1 & 0 \\ 1 & 8 & 0 & 1 \end{bmatrix}$$

$$\mathcal{P} = \{I, X, Y, Z\}^{\otimes n}$$

$$A = \sum_{i=1}^{2^{2n}} a_i \cdot P_i \qquad \alpha_i = \frac{1}{2^n} \mathrm{Tr}(P_i^{\dagger} A)$$

II: 1.000 + 0.000j
XI: 8.000 + 0.000j
XX: 1.000 + 0.000j

```python
# 4x4 Hermitian
M = np.array([[1, 0, 8, 1],
              [0, 1, 1, 8],
              [8, 1, 1, 0],
              [1, 8, 0, 1]], dtype=complex)

# Pauli basis 16 Pauli ⊗ Pauli
pauli_labels = ['II', 'IX', 'IY', 'IZ',
                'XI', 'XX', 'XY', 'XZ',
                'YI', 'YX', 'YY', 'YZ',
                'ZI', 'ZX', 'ZY', 'ZZ']

# Coefficient for each basis
coeffs = []
for label in pauli_labels:
    P = Pauli(label)
    op = Operator(P)
    coeff = 0.25 * np.trace(op.data.conj().T @ M)  # ⟨P|H⟩ inner product
    coeffs.append(coeff)

for label, c in zip(pauli_labels, coeffs):
    if not np.isclose(c, 0):
        print(f"{label}: {c.real:.3f} + {c.imag:.3f}j")
```

Python

Preperation

Manually constructing the corresponding circuit while considering the spectral norm

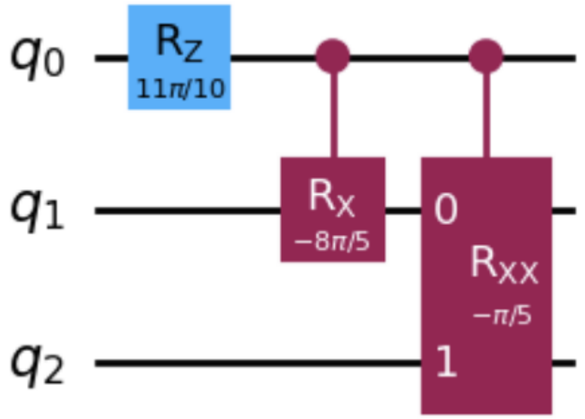
```python
# ── 2. Scaling M to A = (I + M/norm_bound)/2 ──
norm_1 = np.max(np.sum(np.abs(M), axis=0)) + 0.1
norm_inf = np.max(np.sum(np.abs(M), axis=1)) + 0.1

norm_bound = min(norm_1, norm_inf)

A = (np.eye(4) + M / norm_bound) / 2

def build_exp_iAt(norm_bound):
    theta_xi = -2*np.pi * 8 / norm_bound
    theta_xx = -2*np.pi * 1 / norm_bound
    theta_ii = np.pi * ((1 / norm_bound) + 1)

    qc = QuantumCircuit(3)
    qc.rz(theta_ii, 0)
    qc.crx(theta_xi, 0, 1)
    qc.append(RXXGate(theta_xx).control(1), [0, 1, 2])

    return qc

display(build_exp_iAt(10).draw('mpl'))
```

Python

upper bound of Spectral Norm
min(1-norm, ∞-norm)

Rescaled Hamiltonian A
Enable phase estimation on a non-positive-definite M

$$A = \frac{\mathbb{I} + \frac{M}{norm_{bound}}}{2}$$



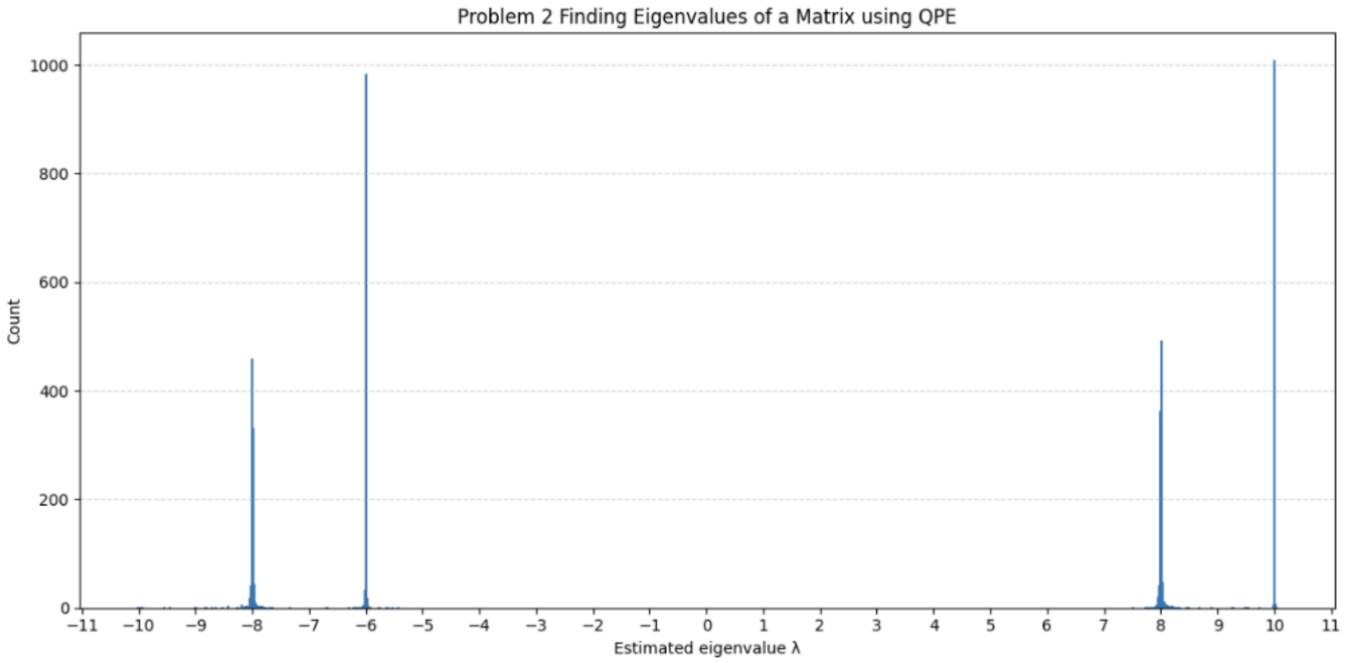
Phase Estimation

Applying Quantum Phase estimation

In order to comply with the constraint of not assuming prior eigenvector knowledge, QPE is executed using all four computational basis states as input

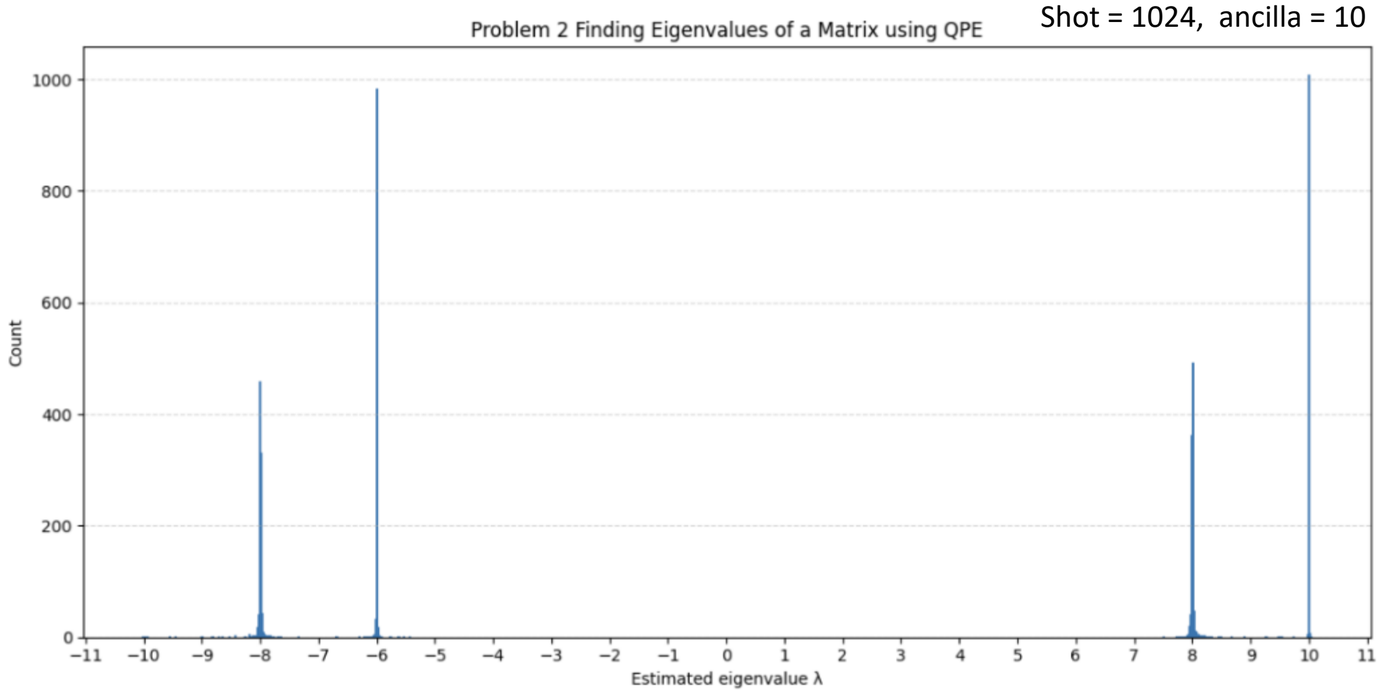
```python
backend = AerSimulator()
shots = 1024
num_ancilla = 10
all_counts = {}

for basis in range(4):
    qc = QuantumCircuit(num_ancilla + 2, num_ancilla)
    qc.h(range(num_ancilla))

    if basis & 1: qc.x(num_ancilla + 1)
    if basis & 2: qc.x(num_ancilla)

    for j in range(num_ancilla):
        qc.append(cU_gate.power(2**j),
                  [j, num_ancilla, num_ancilla + 1])

    qc.append(QFT(num_ancilla, inverse=True, do_swaps=True),
              range(num_ancilla))
    qc.measure(range(num_ancilla), range(num_ancilla))

    job = backend.run(transpile(qc, backend), shots=shots)
    counts = job.result().get_counts()
    for bitstr, cnt in counts.items():
        all_counts[bitstr] = all_counts.get(bitstr, 0) + cnt
```

Python



Conclusion

Comparison Against Computed Eigenvalues



$$M = \begin{bmatrix} 1 & 0 & 8 & 1 \\ 0 & 1 & 1 & 8 \\ 8 & 1 & 1 & 0 \\ 1 & 8 & 0 & 1 \end{bmatrix}$$

$$\det(M - \lambda I) = 0$$

$$\lambda = -8, -6, 8, 10$$

The results from QPE precisely match the classically computed eigenvalues

2025 Quantum Hackathon

Problem 3

Analyzing the Impact of Noise on QPE

Using the implementation from the previous problems,
apply different noise models and analyze how the accuracy of QPE is affected

Quantum Noise Models

1 Amplitude Damping

$$K_0 = \begin{pmatrix} 1 & 0 \\ 0 & \sqrt{1-\lambda} \end{pmatrix}, K_1 = \begin{pmatrix} 0 & \sqrt{\lambda} \\ 0 & 0 \end{pmatrix}$$

$$\mathcal{E}(\rho) = K_0 \rho K_0^\dagger + K_1 \rho K_1^\dagger = \begin{pmatrix} \rho_{00} + \lambda\rho_{11} & \sqrt{1-\lambda}\rho_{01} \\ \sqrt{1-\lambda}\rho_{10} & (1-\lambda)\rho_{11} \end{pmatrix}$$

2 Depolarizing Channel

$$K_0 = \sqrt{1-p} \cdot I = \begin{pmatrix} \sqrt{1-p} & 0 \\ 0 & \sqrt{1-p} \end{pmatrix}, K_{1,2,3} = \sqrt{\frac{p}{3}}\{X, Y, Z\}$$

$$\mathcal{E}(\rho) = (1-p)\rho + \frac{p}{3}(X\rho X + Y\rho Y + Z\rho Z) = (1-p)\rho + p\frac{I}{2}$$

3 Phase Damping Dephasing

$$K_0 = \begin{pmatrix} 1 & 0 \\ 0 & \sqrt{1-\lambda} \end{pmatrix}, K_1 = \begin{pmatrix} 0 & 0 \\ 0 & \sqrt{\lambda} \end{pmatrix}$$

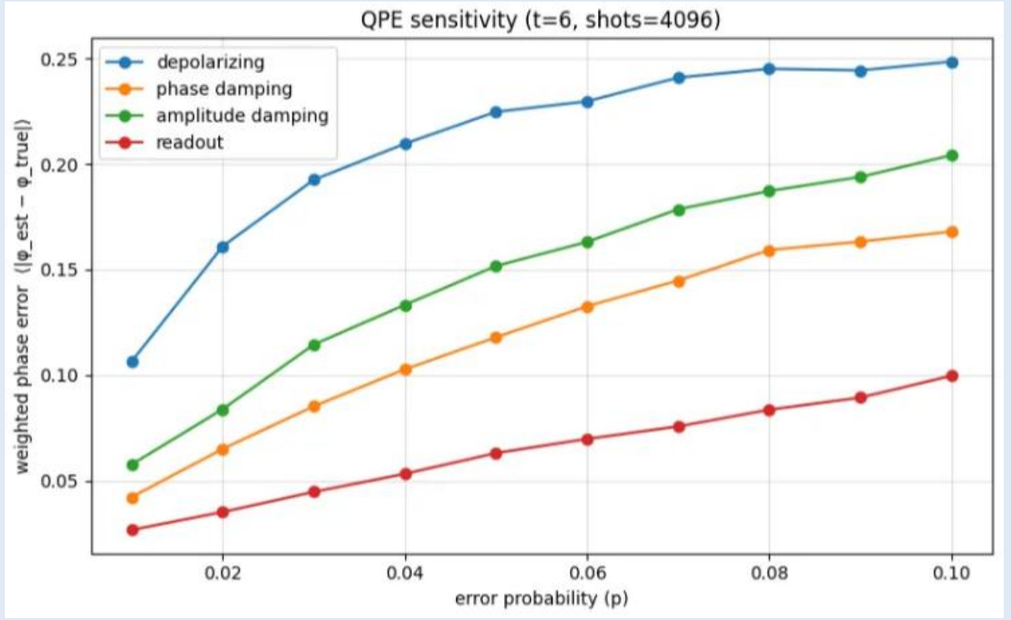
$$\mathcal{E}(\rho) = K_0 \rho K_0^\dagger + K_1 \rho K_1^\dagger = \begin{pmatrix} \rho_{00} & \sqrt{1-\lambda}\rho_{01} \\ \sqrt{1-\lambda}\rho_{10} & \rho_{11} \end{pmatrix}$$

4 Readout Error

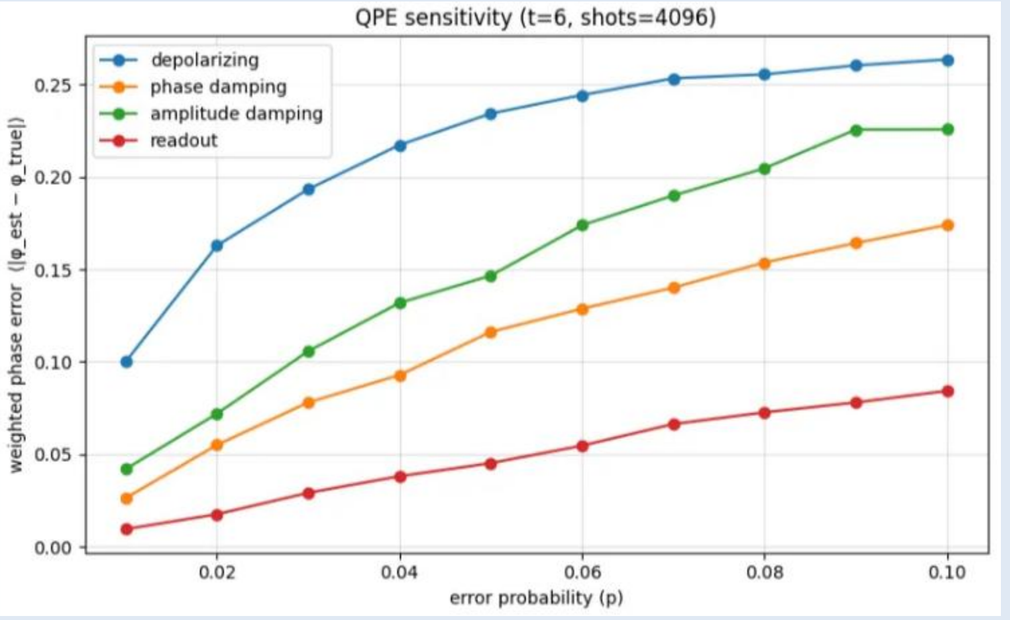
$$M_0 = \sqrt{1-e_0}|0\rangle\langle 0| + \sqrt{e_1}|0\rangle\langle 1|, M_1 = \sqrt{e_0}|1\rangle\langle 0| + \sqrt{1-e_1}|1\rangle\langle 1|$$

$$\mathcal{E}(\rho) = M_0 \rho M_0 + M_1 \rho M_1$$

QPE sensitivity

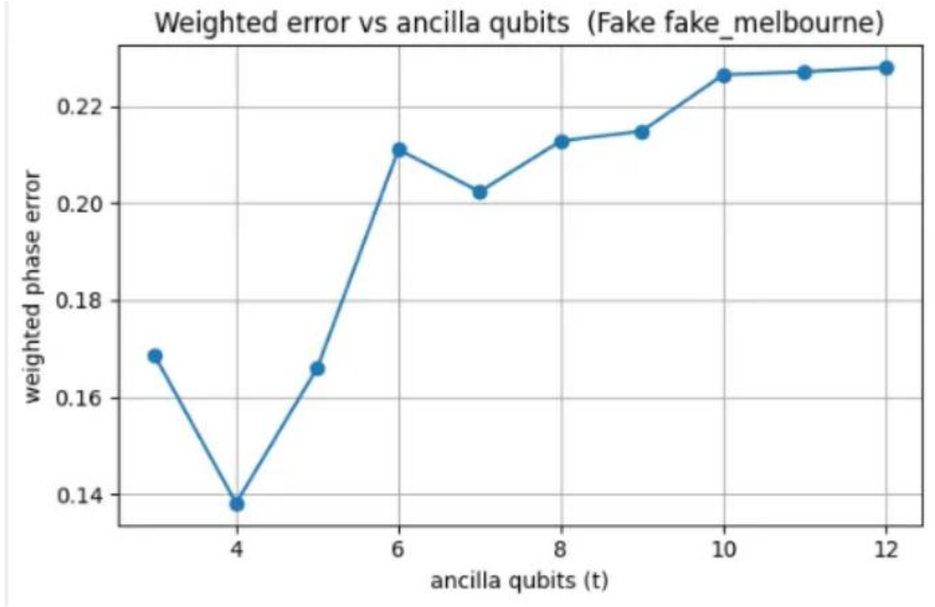


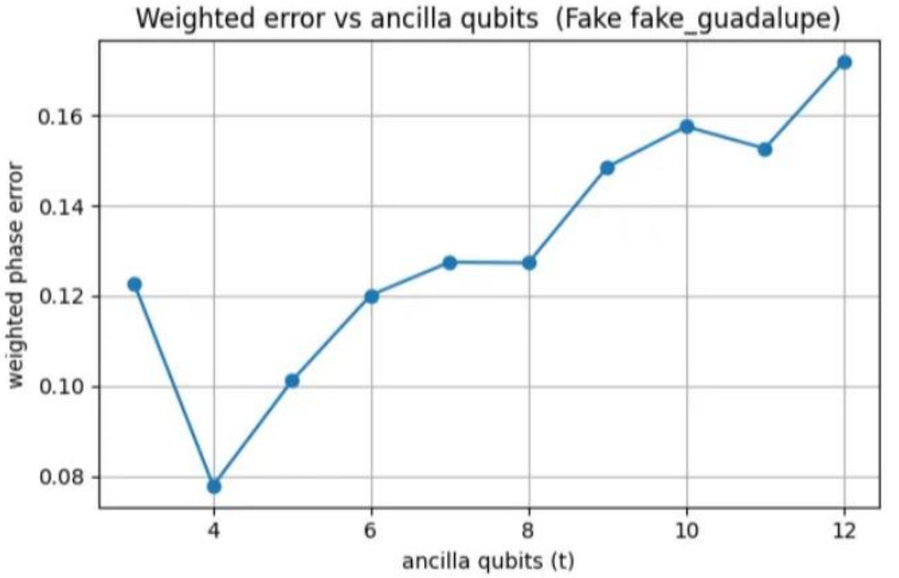
θ = 3/7



θ = 5/8

Fake Backend





Trade-off

Improves QPE precision

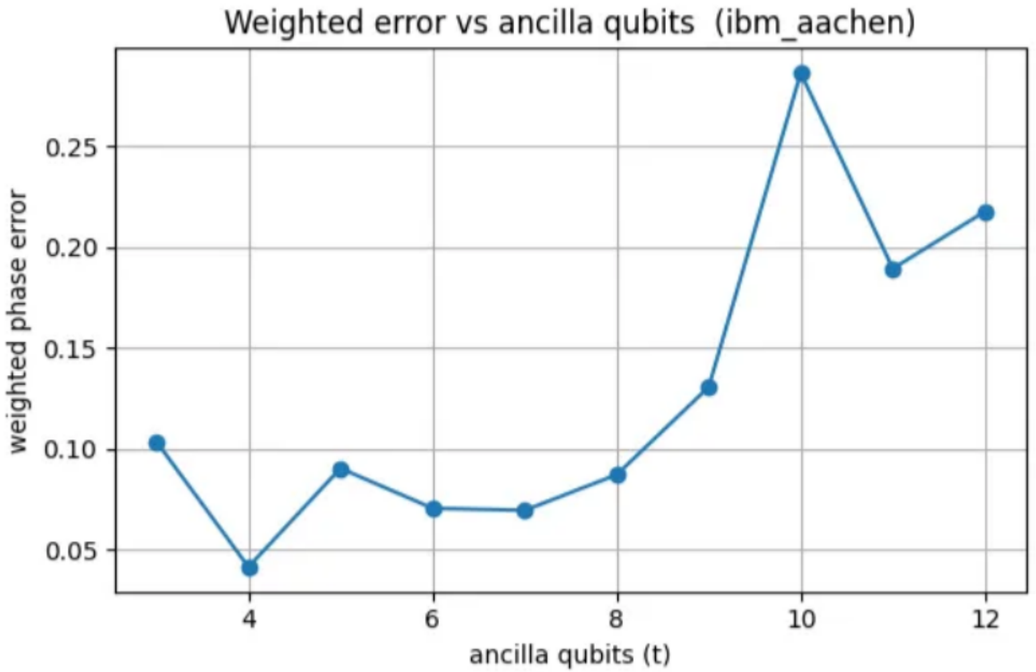
error ↓

Increasing ancilla qubits

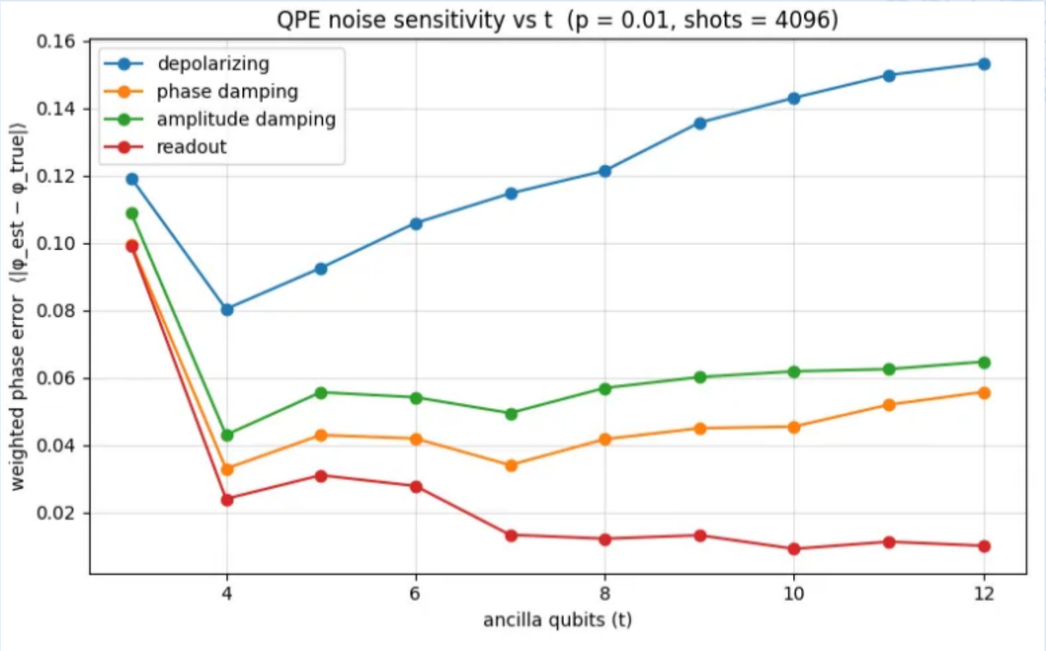
Deeper circuit depth

error ↑

Real Backend

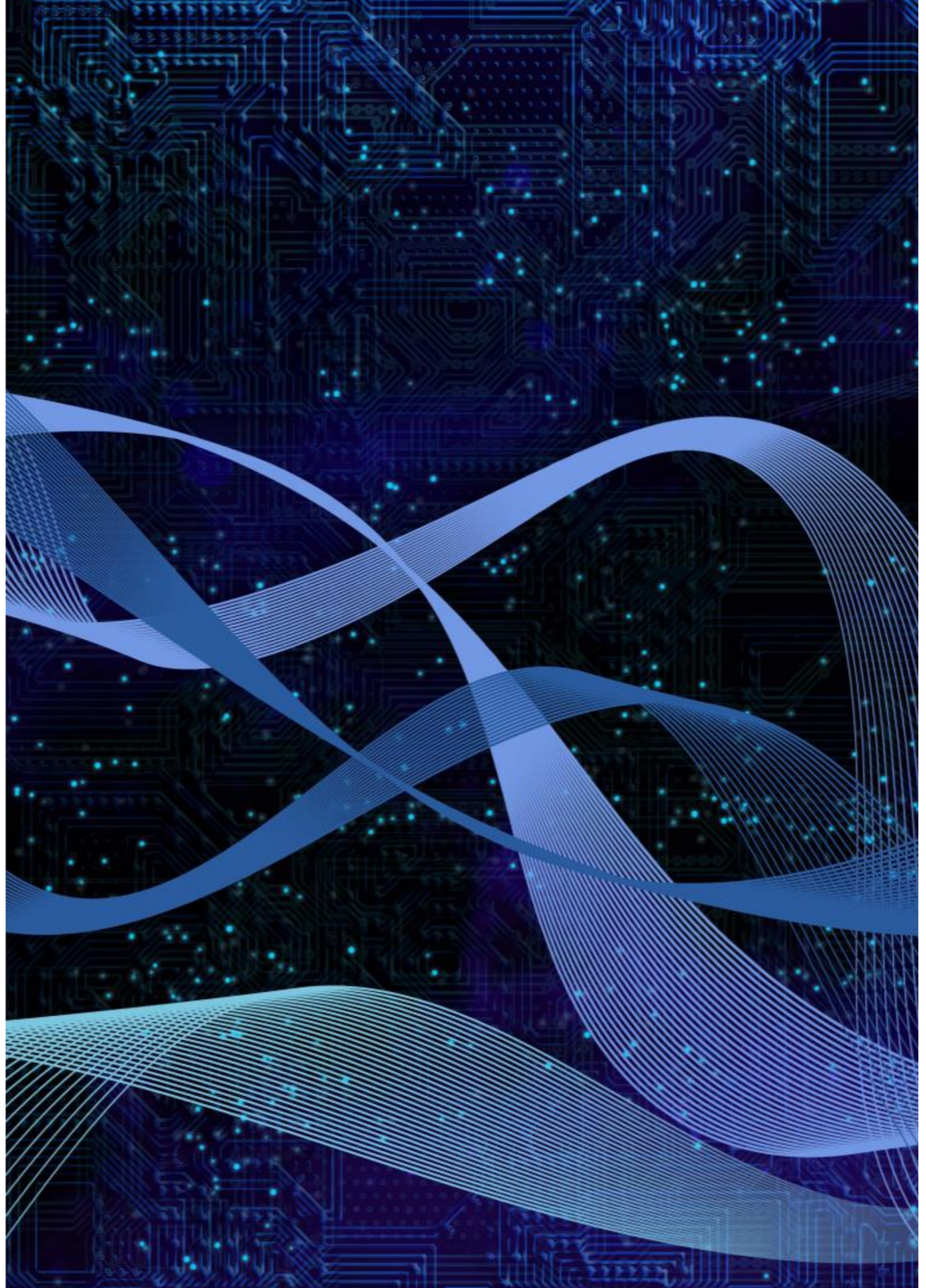


Additional



Conclusion

- For the same noise strength p, the weighted phase error increases in the order: **depolarizing > amplitude damping > phase damping > readout noise**
- **Depolarizing noise** introduces uniform random errors across all Pauli directions, which fully destroy coherence and phase information—making it the **most disruptive** to QPE.
- As the number of ancilla qubits increases, precision improves, but noise depth increases, leading to an **optimal number of ancilla qubits** where performance is maximized.
- When p is fixed and the number of ancilla qubits increases, the error evolution changes differently for different noise types



2025 Quantum Hackathon

Problem 4

Exploring Applications of QPE

- Applied QPE to the Max-Cut Hamiltonian to estimate eigenvalues and identify low-energy solutions
- Implemented QSVT to transform the Hamiltonian spectrum before applying QPE
- Demonstrated how spectral shaping via QSVT can enhance QPE's effectiveness in solving combinatorial problems

Application of QPE

QPE applied to MAX cut problem

```python
# ——————————————————————
# 1. Max-Cut Hamiltonian
# ——————————————————————
def build_maxcut_hamiltonian(graph: nx.Graph) -> SparsePauliOp:
    """
    Build the MaxCut Hamiltonian: H = (|E|/2)*I - (1/2)*Σ_{(i,j)∈E}(Z_i Z_j)
    """
    num_qubits = len(graph.nodes)
    edges = list(graph.edges())
    num_edges = len(edges)

    pauli_terms = ["I" * num_qubits]
    coeffs = [-num_edges / 2]

    for (u, v) in edges:
        z_term = ["I"] * num_qubits
        z_term[u] = "Z"
        z_term[v] = "Z"
        pauli_terms.append("".join(z_term))
        coeffs.append(0.5)

    return SparsePauliOp.from_list(list(zip(pauli_terms, coeffs)))
```

Python

$$H = -\frac{|E|}{2}I + \frac{1}{2}\sum_{(i,j)\in E} Z_i Z_j$$

```python
def create_qpe_circuit(num_ancilla: int, U_gate: UnitaryGate, state_prep: QuantumCircuit) -> QuantumCircuit:
    n_target = state_prep.num_qubits
    U_mat = U_gate.to_matrix()
    qc = QuantumCircuit(num_ancilla + n_target, num_ancilla)
    qc.h(range(num_ancilla))
    qc.compose(state_prep, qubits=range(num_ancilla, num_ancilla+n_target), inplace=True)
    for j in range(num_ancilla):
        mat_pow = np.linalg.matrix_power(U_mat, 2**j)
        qc.append(UnitaryGate(mat_pow).control(), [j] + list(range(num_ancilla, num_ancilla+n_target)))
    qc.compose(QFT(num_ancilla, do_swaps=True).inverse(), range(num_ancilla), inplace=True)
    qc.measure(range(num_ancilla), range(num_ancilla))
    return qc
```

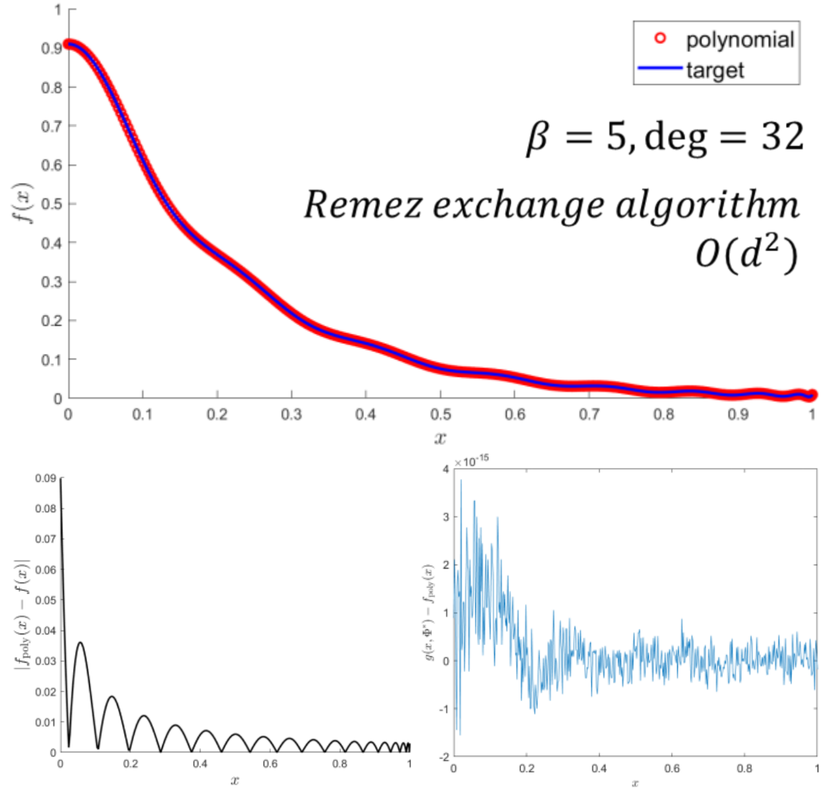
Limitation

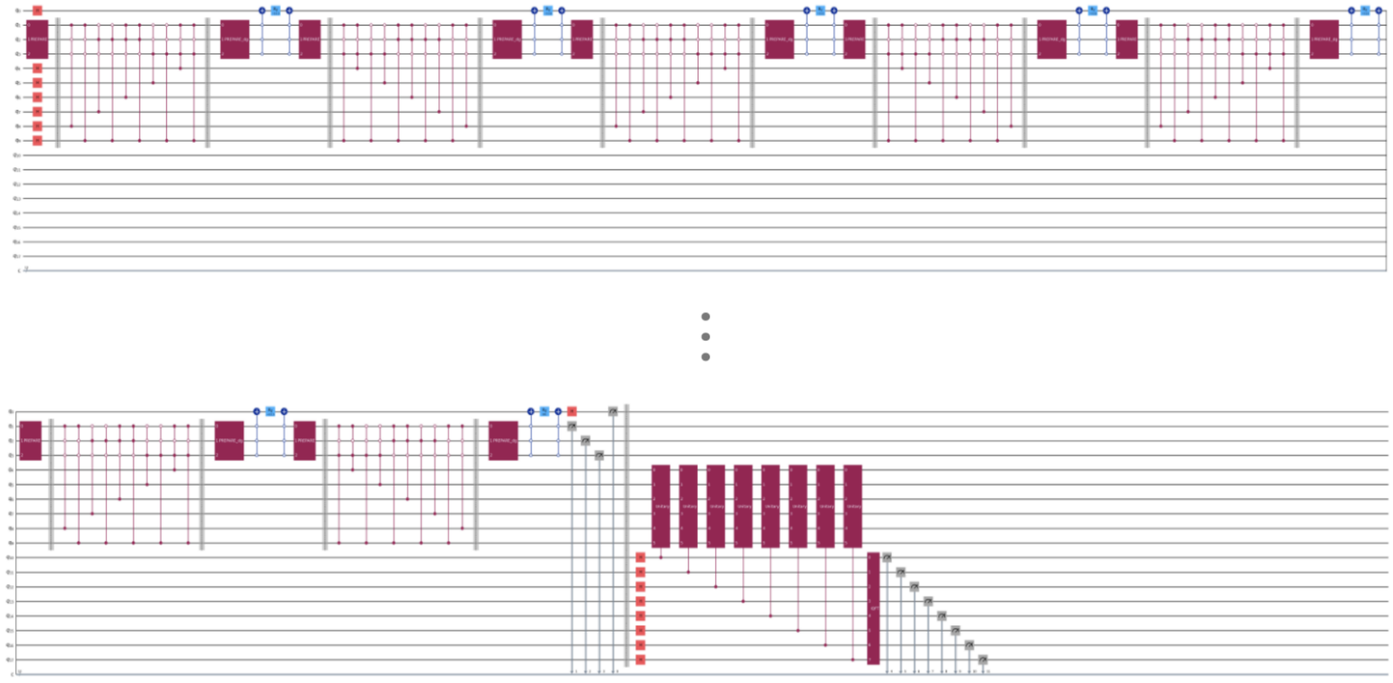
QPE requires the input state to have significant overlap with an eigenstate of the Hamiltonian, ideally the ground state. However, in the Max-Cut problem, the ground state is unknown and hard to prepare, leading to low success probability

QSVT Enhancement

Enhancement QPE via QSVT

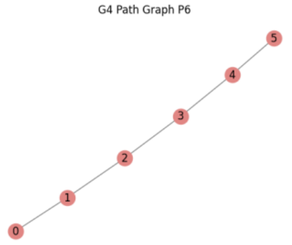
QSVT allows to amplify desired eigenvalues, for those near the ground state, or suppress unwanted components, making QPE more likely to detect meaningful phases





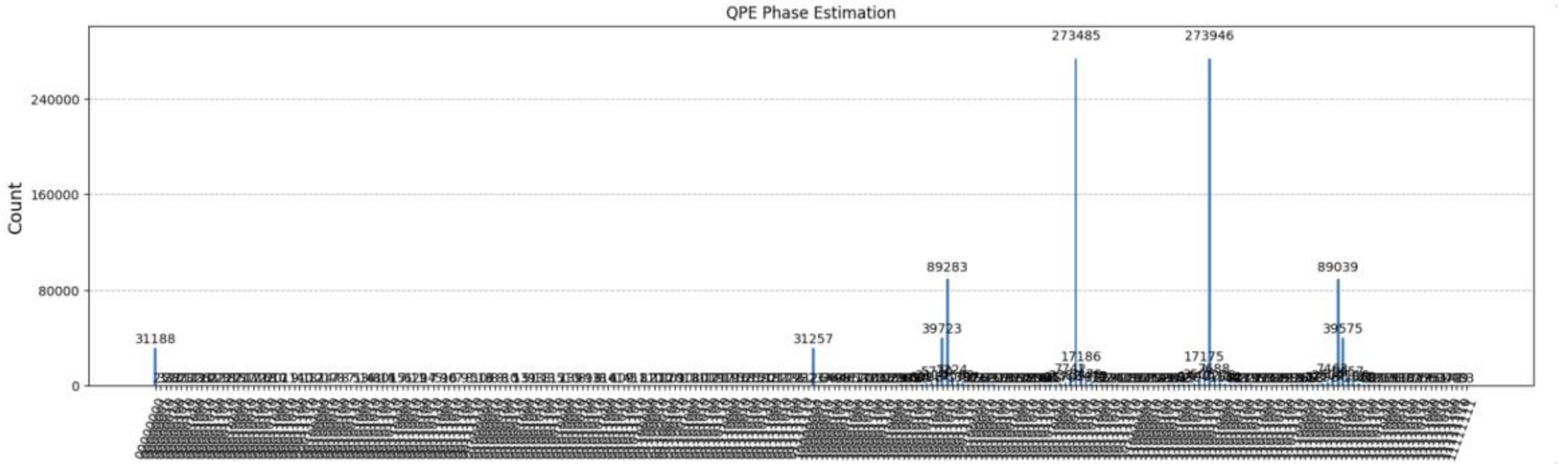
LCU used for implementation of block encoding

Comparison:
QPE alone vs QSVT enhanced



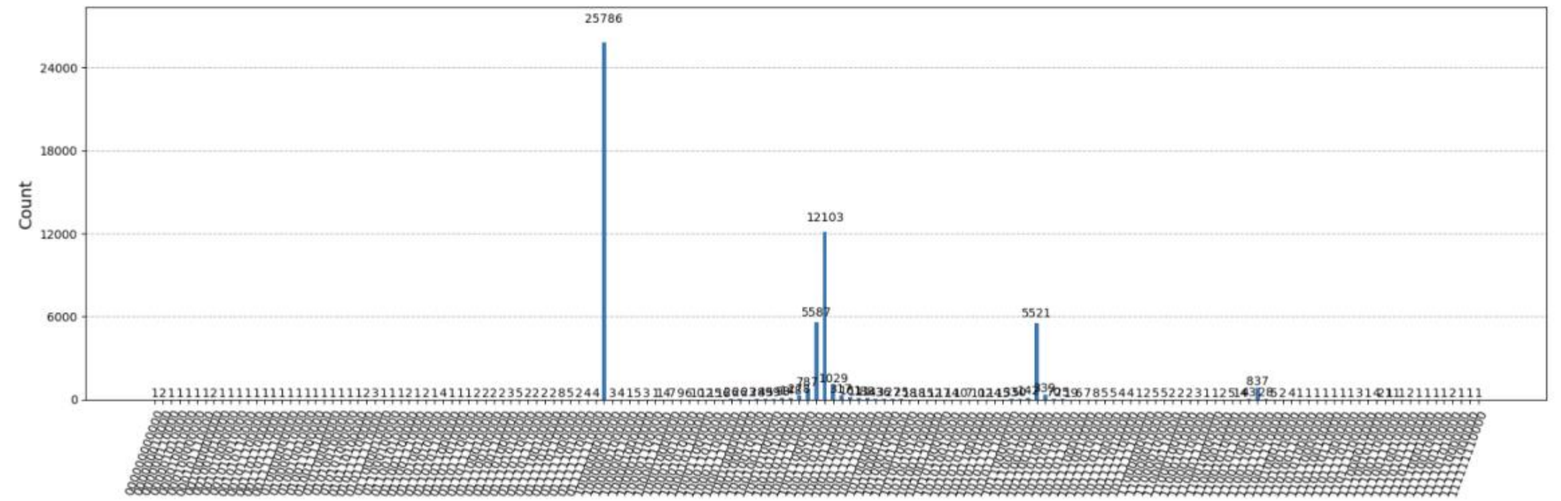
QPE alone

1. bits 11001101 freq 273946 E_QPE ≈ -1.992
2. bits 10110011 freq 273485 E_QPE ≈ -3.008
3. bits 10011010 freq 89283 E_QPE ≈ -3.984
4. bits 11100110 freq 89039 E_QPE ≈ -1.016
5. bits 10011001 freq 39723 E_QPE ≈ -4.023
6. bits 11100111 freq 39575 E_QPE ≈ -0.977
7. bits 10000000 freq 31257 E_QPE ≈ -5.000
8. bits 00000000 freq 31188 E_QPE ≈ 0.000
9. bits 10110100 freq 17186 E_QPE ≈ -2.969
10. bits 11001100 freq 17175 E_QPE ≈ -2.031



QSVT enhanced

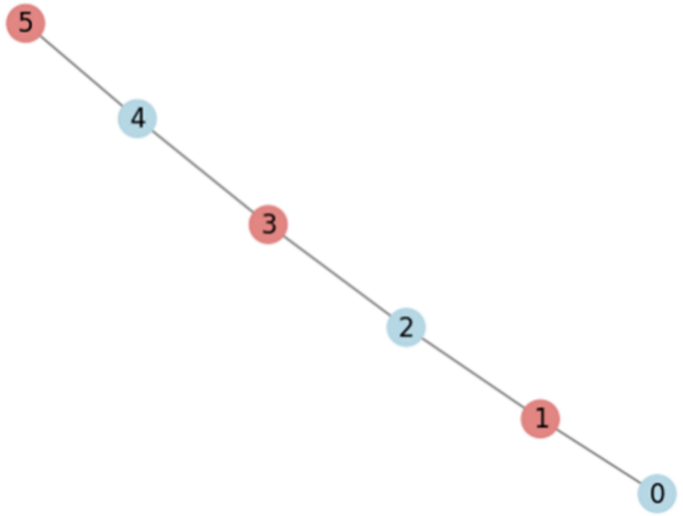
1. bits 100000000000 freq 25786 E_QPE ≈ -5.000
2. bits 100110100000 freq 12103 E_QPE ≈ -3.984
3. bits 100110010000 freq 5587 E_QPE ≈ -4.023
4. bits 101100110000 freq 5521 E_QPE ≈ -3.008
5. bits 100110110000 freq 1029 E_QPE ≈ -3.945
6. bits 110011010000 freq 837 E_QPE ≈ -1.992
7. bits 100110000000 freq 787 E_QPE ≈ -4.062
8. bits 101101000000 freq 339 E_QPE ≈ -2.969
9. bits 100111000000 freq 317 E_QPE ≈ -3.906
10. bits 100101110000 freq 278 E_QPE ≈ -4.102



Comparison:
QPE alone vs QSVT enhanced

Classical

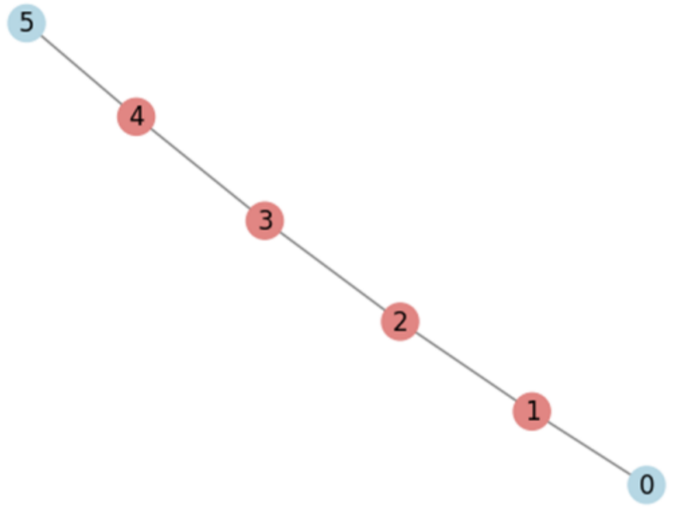
Greedy Max-Cut
Bitstring: 010101



QPE alone

```
[QPE Estimated Eigenvalue] E ≈ -1.99219
[Closest True Eigenvalue] E = -2.00000
[Most Probable State in Eigenvector] |100001⟩  with prob = 1.000
```

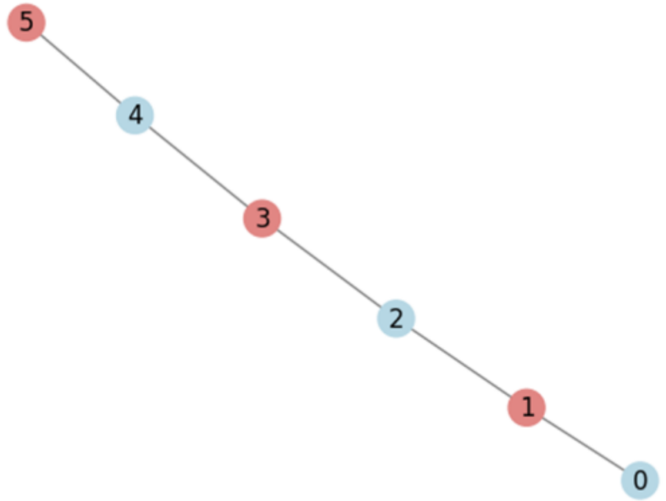
QPE Inferred Max-Cut
Bitstring: 100001



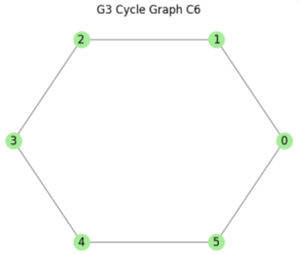
QSVT enhanced

```
[QPE Estimated Eigenvalue] E ≈ -5.00000
[Closest True Eigenvalue] E = -5.00000
[Most Probable State in Eigenvector] |010101⟩  with prob = 1.000
```

QPE Inferred Max-Cut
Bitstring: 010101



Comparison: QPE alone vs QSVT enhanced

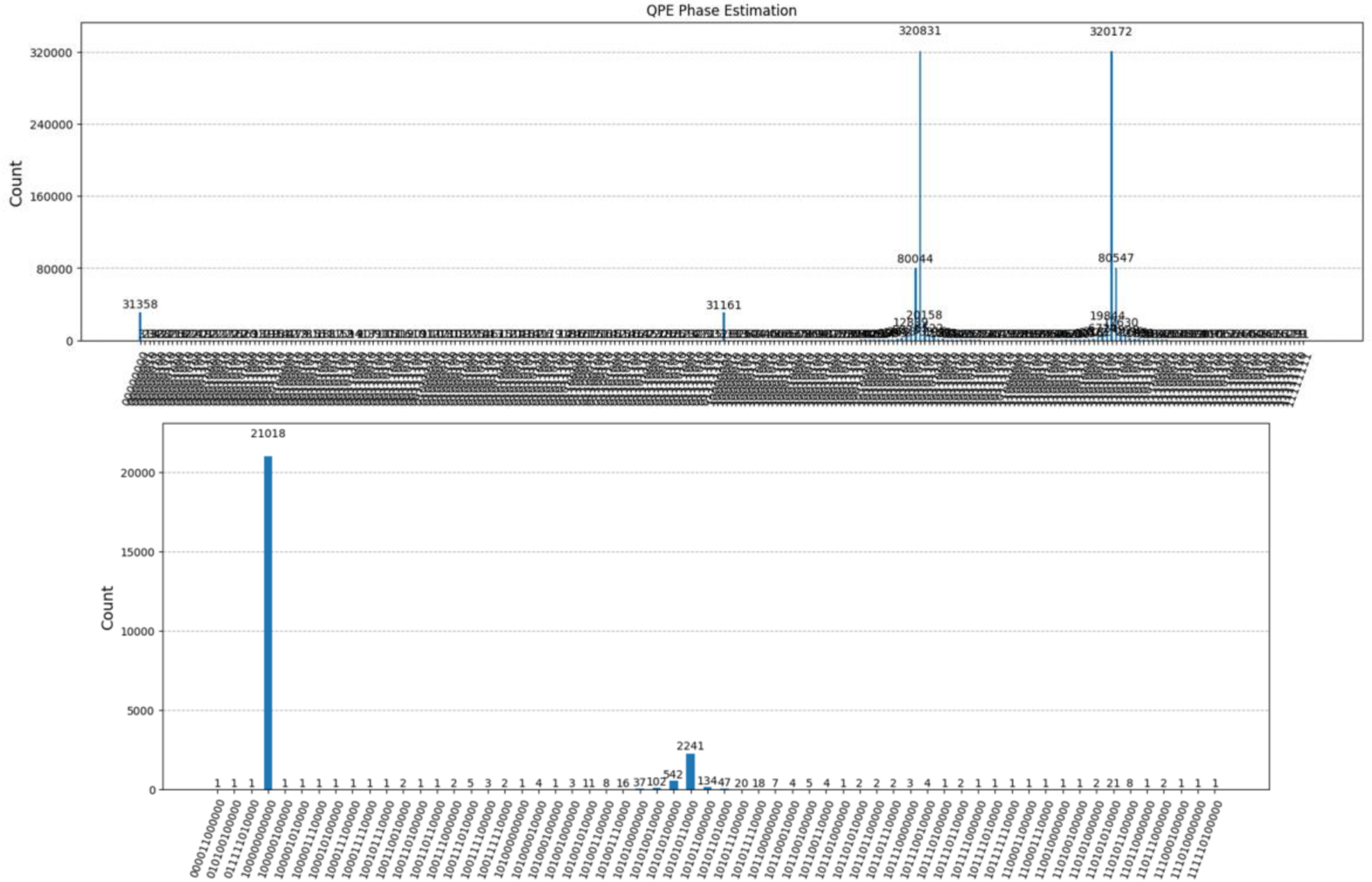


QPE alone

```
=== QPE 결과 (Top-10) ===
1. bits 10101011  freq 320831  E_QPE ≈ -3.984
2. bits 11010101  freq 320172  E_QPE ≈ -2.016
3. bits 11010110  freq 80547  E_QPE ≈ -1.969
4. bits 10101010  freq 80044  E_QPE ≈ -4.031
5. bits 00000000  freq 31358  E_QPE ≈ 0.000
6. bits 10000000  freq 31161  E_QPE ≈ -6.000
7. bits 10101100  freq 20158  E_QPE ≈ -3.938
8. bits 11010100  freq 19844  E_QPE ≈ -2.062
9. bits 10101001  freq 12839  E_QPE ≈ -4.078
10. bits 11010111  freq 12630  E_QPE ≈ -1.922
```

QSVT enhanced

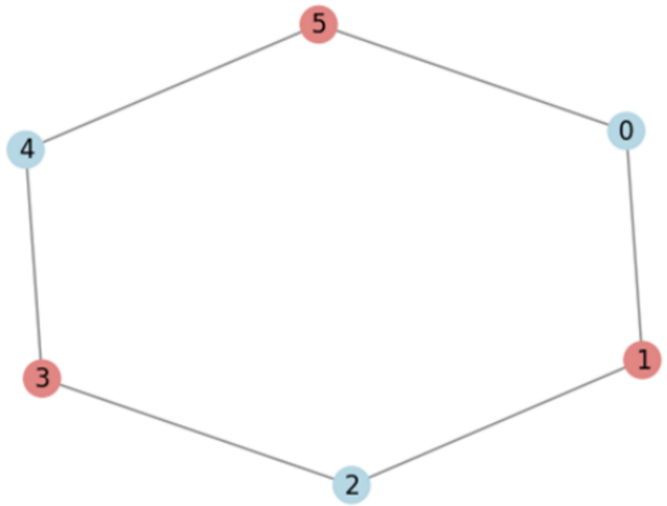
```
=== QSVT+QPE 결과 (Top-10) ===
1. bits 100000000000  freq 21018  E_QPE ≈ -6.000
2. bits 101010110000  freq 2241  E_QPE ≈ -3.984
3. bits 101010100000  freq  542  E_QPE ≈ -4.031
4. bits 101011000000  freq  134  E_QPE ≈ -3.938
5. bits 101010010000  freq  102  E_QPE ≈ -4.078
6. bits 101011010000  freq   47  E_QPE ≈ -3.891
7. bits 101010000000  freq   37  E_QPE ≈ -4.125
8. bits 110101010000  freq   21  E_QPE ≈ -2.016
9. bits 101011100000  freq   20  E_QPE ≈ -3.844
10. bits 101011110000  freq   18  E_QPE ≈ -3.797
```



Comparison:
QPE alone vs QSVT enhanced

Classical

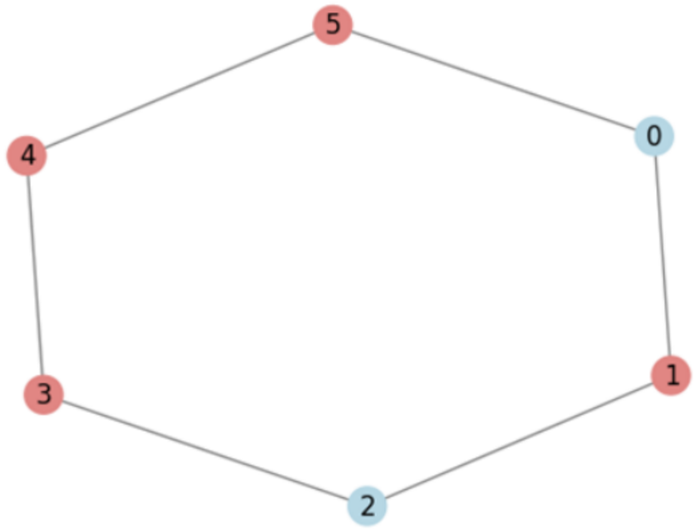
Greedy Max-Cut
Bitstring: 010101



QPE alone

```
[QPE Estimated Eigenvalue] E ≈ -3.32031
[Closest True Eigenvalue] E = -4.00000
[Most Probable State in Eigenvector] |000101⟩  with prob = 1.000
```

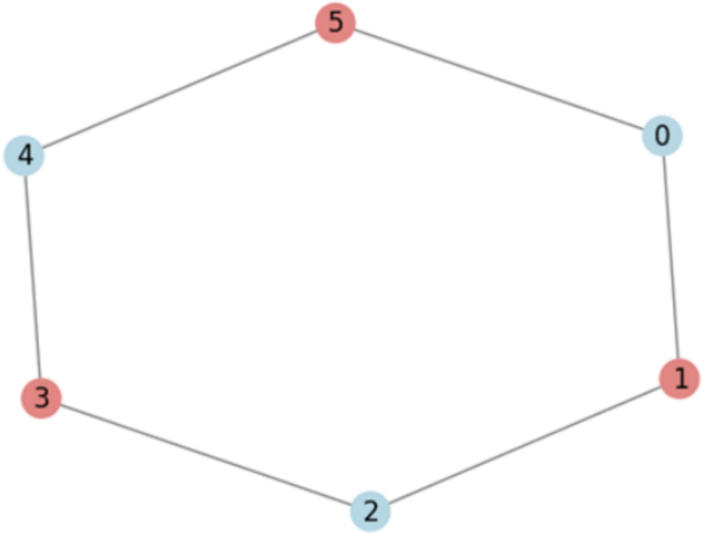
QPE Inferred Max-Cut
Bitstring: 000101



QSVT enhanced

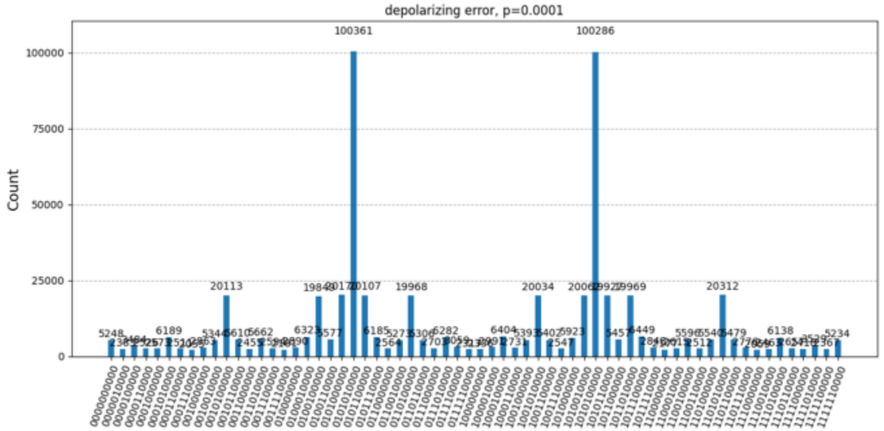
```
[QPE Estimated Eigenvalue] E ≈ -6.00000
[Closest True Eigenvalue] E = -6.00000
[Most Probable State in Eigenvector] |010101⟩  with prob = 1.000
```

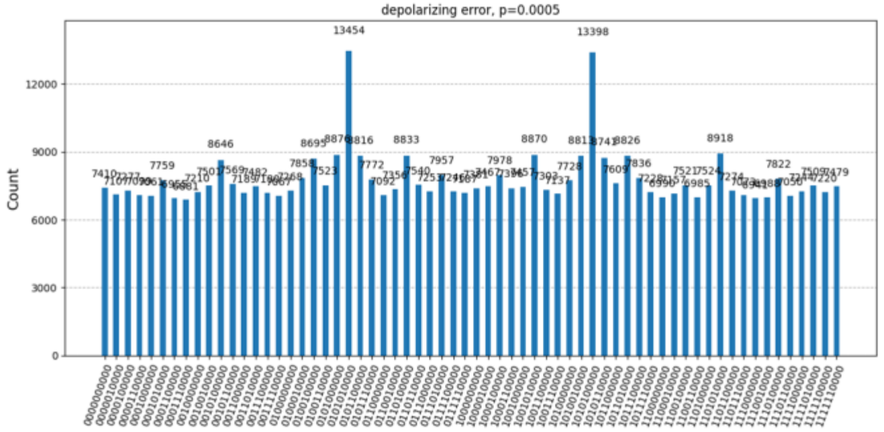
QPE Inferred Max-Cut
Bitstring: 010101



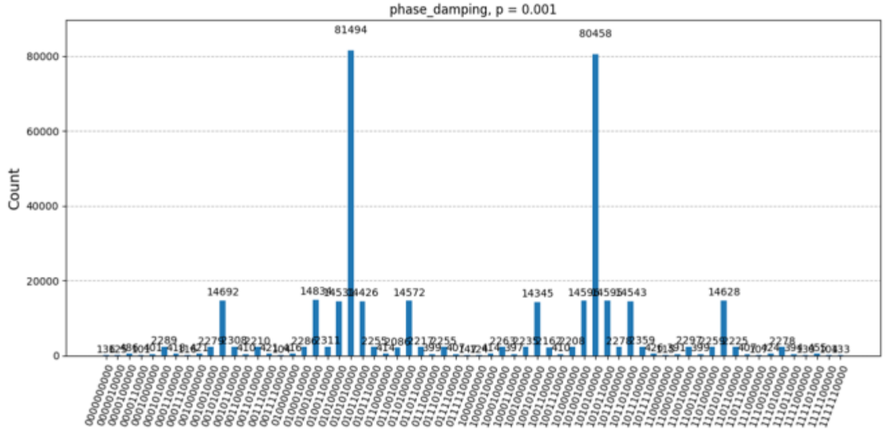
Importing error modules to QSVT

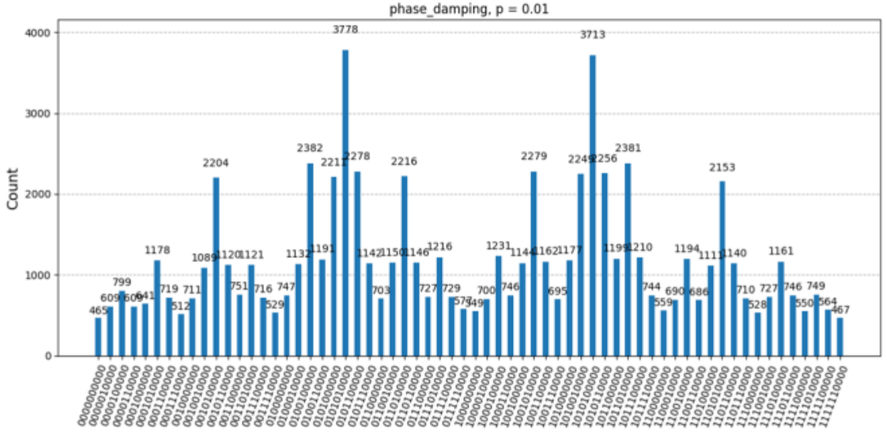
Depolarizing error





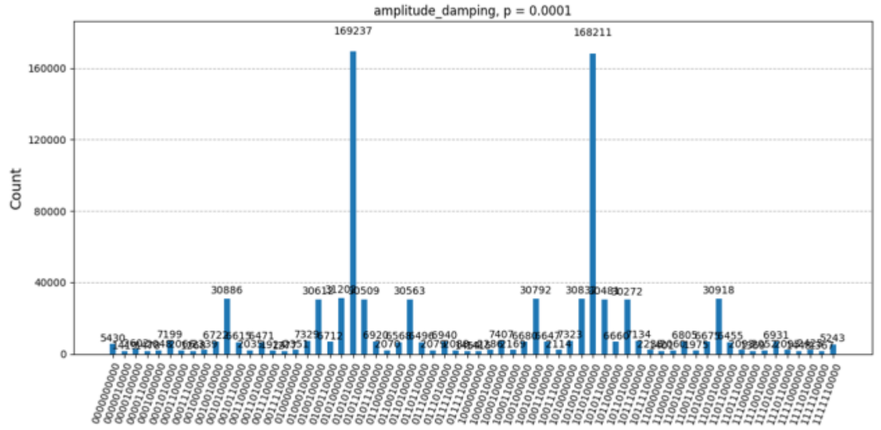
Phase damping

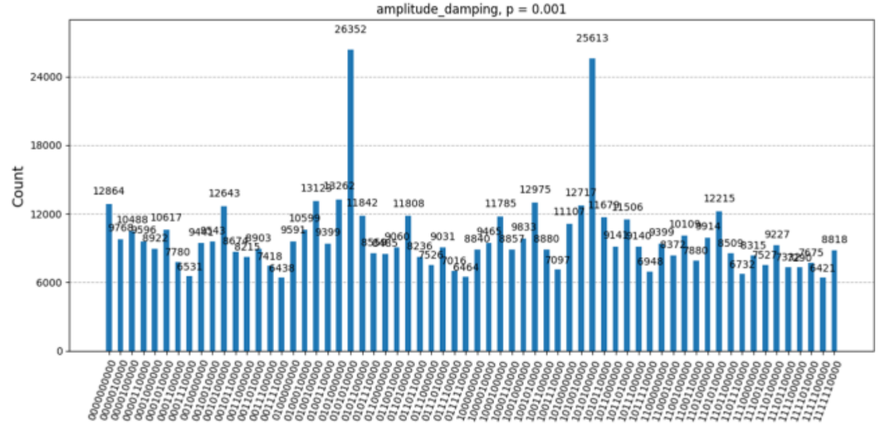




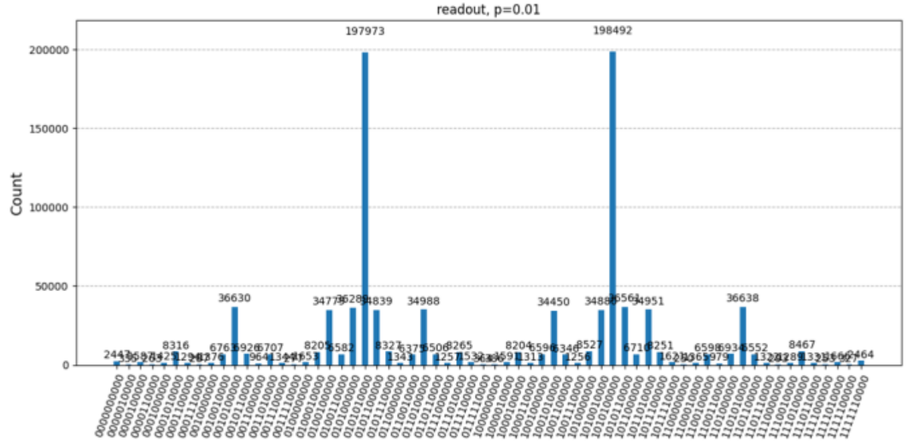
Importing error modules to QSVT

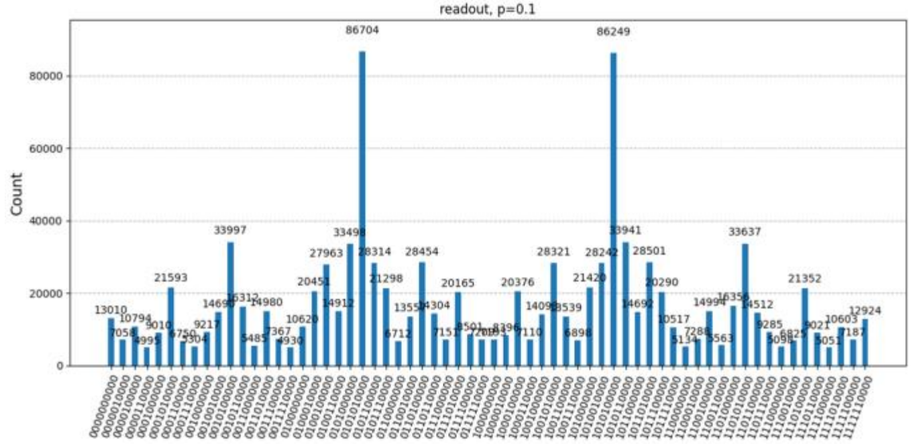
Amplitude damping





Readout





Conclusion

- QPE serves as a fundamental tool for quantum eigenvalue estimation
- Its performance is limited by sensitivity to the input state
- Introducing QSVT enables spectral shaping to amplify ground or low-energy states
- This integration helps overcome QPE's limitations and enhances solution visibility
- This method has high potential for future quantum optimization frameworks and can be extended to more complex Hamiltonians, variational circuits, or real-world datasets

Future Applications

overcome a key bottleneck of QPE

1 | Knapsack problem

putting the maximum value of things in limited bag

2 | Eletricity grid distribution

minimize imbalance of electricity supply and demand

3 | Portfolio Optimization

minimize risk for fixed return, maximize return for fixed risk

> "Any optimization problem that can convert cost function to Ising Hamiltonian"

Thank you

QUICK Seonggeun Park, Juhyun Song, Jaemin Jeon, Sungheum Yoo, Yoonha Hwang

DATE July 1st, 2025