Date : 26/8/25

```
            TASK 6: Quantum Error Correction (9-Qubit Code)
```

Aim: To demonstrate logical qubit encoding and error protection using the 9-qubit Shor code and Qiskit's noise models.

Algorithm:

Step 1: Correct Shor encoding circuit

Step 2: Simplified syndrome measurement

Step 3: Apply quantum gates to test the code

Step 4: Proper error correction based on syndrome

Step 5: Full Shor QEC routine with quantum operations

Step 6: Noise Model

Step 7: Run simulation and compare with/without error correction

Step 8: Demonstration with specific error injection

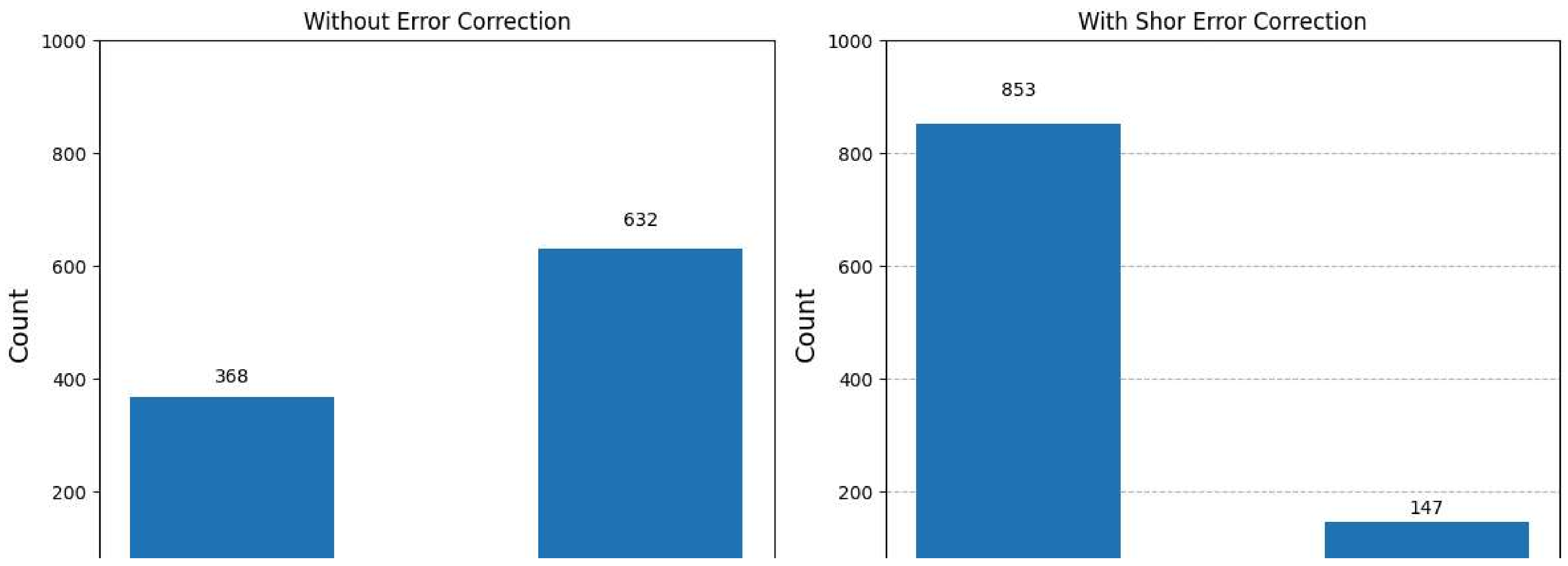
Step 9: Visualize Quantum Circuits

```
%pip install qiskit qiskit-aer matplotlib
from qiskit import QuantumCircuit, transpile
from qiskit_aer import AerSimulator
from qiskit_aer.noise import NoiseModel, depolarizing_error
from qiskit.quantum_info import Statevector, state_fidelity
from qiskit.visualization import plot_histogram
from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister
import matplotlib.pyplot as plt
import numpy as np
def shor_encode():
 qc = QuantumCircuit(9, name="ShorEncode")
 qc.cx(0, 3)
 qc.cx(0, 6)
 qc.h(0)
 qc.h(3)
 qc.h(6)

 qc.cx(0, 1)
 qc.cx(0, 2)
 qc.cx(3, 4)
 qc.cx(3, 5)
 qc.cx(6, 7)
 qc.cx(6, 8)

 return qc
# -----------------------------
# Step 2: Simplified syndrome measurement
# -----------------------------
def measure_syndromes():
 # Create a simpler syndrome measurement without extra qubits
 qc = QuantumCircuit(9, 6, name="SyndromeMeasurement")

 # For simulation purposes, we'll use a simplified approach
 # In a real implementation, we'd use ancilla qubits
 qc.barrier()
 qc.measure([0, 1, 2, 3, 4, 5], [0, 1, 2, 3, 4, 5]) # Simplified measurement

 return qc
# -----------------------------
# Step 3: Apply quantum gates to test the code
# -----------------------------
def apply_quantum_operations():
 qc = QuantumCircuit(9, name="QuantumOperations")

 # Apply some quantum gates to test the code
 qc.h(0) # Hadamard - creates superposition
 qc.rx(0.5, 1) # Rotation around X-axis
 qc.ry(0.3, 2) # Rotation around Y-axis
 qc.rz(0.7, 3) # Rotation around Z-axis
 qc.s(4) # Phase gate
 qc.sdg(5) # Inverse phase gate
 qc.t(6) # T gate
 qc.tdg(7) # Inverse T gate
 qc.x(8) # Pauli-X

 # Add some two-qubit gates
 qc.cx(0, 4) # CNOT
 qc.cz(1, 5) # Controlled-Z
 qc.swap(2, 6) # SWAP

 return qc
# -----------------------------
# Step 4: Proper error correction based on syndrome
# -----------------------------
def apply_error_correction(syndrome_bits="000000"):
 qc = QuantumCircuit(9, name="ErrorCorrection")
 qc = QuantumCircuit(9, name="ErrorCorrection")

 # For demonstration, apply a simple correction pattern
 # In a real implementation, this would be based on the syndrome
 qc.barrier()
 # Apply some correction gates (simplified)
 qc.x(0)
 qc.z(0)
 qc.x(0)
 qc.z(0)

 return qc
# -----------------------------
# Step 5: Full Shor QEC routine with quantum operations
# -----------------------------
def shor_qec_circuit():
 # Create circuit with 9 data qubits and 1 classical bit for final measurement
 qc = QuantumCircuit(9, 1)

 # Prepare initial state |+⟩ on qubit 0
 qc.h(0)

 # Apply some quantum operations
 operations_circuit = apply_quantum_operations()
 qc = qc.compose(operations_circuit)

 # Encode using Shor code
 encode_circuit = shor_encode()
 qc = qc.compose(encode_circuit)

 # Add barrier to separate encoding from potential errors
 qc.barrier()

 # Simulate noise (will be added by noise model)

 # Add barrier before error correction
 qc.barrier()

 # For demonstration, we'll use a fixed syndrome pattern
 syndrome_pattern = "000000" # No errors detected

 # Apply error correction based on syndrome
 correction_circuit = apply_error_correction(syndrome_pattern)
 qc = qc.compose(correction_circuit)

 # Decode (reverse of encoding)
 decode_circuit = shor_encode().inverse()
 qc = qc.compose(decode_circuit)

 # Measure the logical qubit
 qc.measure(0, 0)

 return qc
# -----------------------------
# Step 6: Noise Model
# -----------------------------
noise_model = NoiseModel()
p1 = 0.01 # depolarizing probability for 1-qubit gates
p2 = 0.03 # depolarizing probability for 2-qubit gates
# Add depolarizing error for 1-qubit gates
error1 = depolarizing_error(p1, 1)
noise_model.add_all_qubit_quantum_error(error1, ['h', 'x', 'y', 'z', 's', 'sdg', 't', 'tdg', 'rx', 'ry', 'rz'])
# Add depolarizing error for 2-qubit gates
error2 = depolarizing_error(p2, 2)
noise_model.add_all_qubit_quantum_error(error2, ['cx', 'cz', 'swap'])
# -----------------------------
# Step 7: Run simulation and compare with/without error correction
# -----------------------------
def run_comparison():
 backend = AerSimulator(noise_model=noise_model)

 # Create circuit without error correction (single qubit)
 qc_no_ec = QuantumCircuit(1, 1)
 qc_no_ec.h(0)

 # Apply similar operations as in the encoded case
 qc_no_ec.rx(0.5, 0)
 qc_no_ec.ry(0.3, 0)
 qc_no_ec.rz(0.7, 0)

 qc_no_ec.measure(0, 0)

 # Create circuit with error correction
 qc_with_ec = shor_qec_circuit()

 # Transpile both circuits
 transpiled_no_ec = transpile(qc_no_ec, backend)
 transpiled_with_ec = transpile(qc_with_ec, backend)

 # Run simulations
 print("Running simulation without error correction...")
 result_no_ec = backend.run(transpiled_no_ec, shots=1000).result()
 counts_no_ec = result_no_ec.get_counts()
```

```
    print("Running simulation with Shor error correction...")
    result_with_ec = backend.run(transpiled_with_ec, shots=1000).result()
    counts_with_ec = result_with_ec.get_counts()

    # Calculate probabilities
    prob_0_no_ec = counts_no_ec.get('0', 0) / 1000
    prob_1_no_ec = counts_no_ec.get('1', 0) / 1000

    prob_0_with_ec = counts_with_ec.get('0', 0) / 1000
    prob_1_with_ec = counts_with_ec.get('1', 0) / 1000

    print(f"\nResults:")
    print(f"Without error correction: 0={prob_0_no_ec:.3f}, 1={prob_1_no_ec:.3f}")
    print(f"With Shor error correction: 0={prob_0_with_ec:.3f}, 1={prob_1_with_ec:.3f}")

    # For |+⟩ state, we expect roughly 50/50 distribution
    deviation_no_ec = abs(0.5 - prob_0_no_ec) * 200 # Percentage deviation
    deviation_with_ec = abs(0.5 - prob_0_with_ec) * 200

    print(f"Deviation from expected 50/50 without EC: {deviation_no_ec:.2f}%")
    print(f"Deviation from expected 50/50 with EC: {deviation_with_ec:.2f}%")

    # Plot results
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

    plot_histogram(counts_no_ec, ax=ax1)
    ax1.set_title('Without Error Correction')
    ax1.set_ylim(0, 1000)

    plot_histogram(counts_with_ec, ax=ax2)
    ax2.set_title('With Shor Error Correction')
    ax2.set_ylim(0, 1000)

    plt.tight_layout()
    plt.savefig('shor_code_comparison.png', dpi=300, bbox_inches='tight')
    plt.show()

    return counts_no_ec, counts_with_ec
# -----------------------------
# Step 8: Demonstration with specific error injection
# -----------------------------
def demonstrate_error_correction():
    print("\nDemonstrating error correction with specific error injection...")

    # Create a circuit where we intentionally introduce and correct an error
    qc = QuantumCircuit(9, 1)

    # Prepare |1⟩ state
    qc.x(0)

    # Encode using Shor code
    encode_circuit = shor_encode()
    qc = qc.compose(encode_circuit)

    # Introduce a bit-flip error on qubit 4
    qc.x(4)

    # Decode
    decode_circuit = shor_encode().inverse()
    qc = qc.compose(decode_circuit)

    # Measure
    qc.measure(0, 0)

    # Run simulation without noise to see perfect correction
    backend = AerSimulator()
    transpiled_qc = transpile(qc, backend)
    result = backend.run(transpiled_qc, shots=1000).result()
    counts = result.get_counts()

    success_rate = counts.get('1', 0) / 10 # Percentage
    print(f"Results with intentional error on qubit 4: {counts}")
    print(f"Success rate: {success_rate:.1f}% (should be 100% with perfect correction)")

    return counts
# -----------------------------
# Step 9: Visualize Quantum Circuits
# -----------------------------
def visualize_circuits():
    # Create encoding circuit
    encode_circuit = shor_encode()
    print("Shor Encoding Circuit:")
    print(encode_circuit.draw(output='text'))

    # Create full QEC circuit (simplified for display)
    simple_qec = QuantumCircuit(9, 1)
    simple_qec.h(0)
    simple_qec = simple_qec.compose(shor_encode())
    simple_qec.barrier()
    simple_qec = simple_qec.compose(shor_encode().inverse())
    simple_qec.measure(0, 0)

    print("\nSimplified Shor QEC Circuit:")
```

```
 print(simple_qec.draw(output='text'))
# -----------------------------
# Main execution
# -----------------------------
if __name__ == "__main__":
 # Run the comparison
 counts_no_ec, counts_with_ec = run_comparison()

 # Demonstrate specific error correction
 error_counts = demonstrate_error_correction()

 # Show stats
 qc = shor_qec_circuit()
 print("\nCircuit depth:", qc.depth())
 print("Number of gates:", qc.size())
 print("Circuit width (qubits):", qc.num_qubits)

 # Display circuit diagrams
 visualize_circuits()
```

```
Collecting qiskit
  Downloading qiskit-2.2.3-cp39-abi3-manylinux2014_x86_64.manylinux_2_17_x86_64.whl.metadata (12 kB)
Collecting qiskit-aer
  Downloading qiskit_aer-0.17.2-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (8.3 kB)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.12/dist-packages (3.10.0)
Collecting rustworkx>=0.15.0 (from qiskit)
  Downloading rustworkx-0.17.1-cp39-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (10 kB)
Requirement already satisfied: numpy<3,>=1.17 in /usr/local/lib/python3.12/dist-packages (from qiskit) (2.0.2)
Requirement already satisfied: scipy>=1.5 in /usr/local/lib/python3.12/dist-packages (from qiskit) (1.16.3)
Requirement already satisfied: dill>=0.3 in /usr/local/lib/python3.12/dist-packages (from qiskit) (0.3.8)
Collecting stevedore>=3.0.0 (from qiskit)
  Downloading stevedore-5.5.0-py3-none-any.whl.metadata (2.2 kB)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.12/dist-packages (from qiskit) (4.15.0)
Requirement already satisfied: psutil>=5 in /usr/local/lib/python3.12/dist-packages (from qiskit-aer) (5.9.5)
Requirement already satisfied: python-dateutil>=2.8.0 in /usr/local/lib/python3.12/dist-packages (from qiskit-aer) (2.9.0.post0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (1.3.3)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (4.60.1)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (1.4.9)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (25.0)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (11.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (3.2.5)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.8.0->qiskit-aer) (1.17.0)
Downloading qiskit-2.2.3-cp39-abi3-manylinux2014_x86_64.manylinux_2_17_x86_64.whl (8.0 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8.0/8.0 MB 44.4 MB/s eta 0:00:00
Downloading qiskit_aer-0.17.2-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (12.4 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.4/12.4 MB 50.0 MB/s eta 0:00:00
Downloading rustworkx-0.17.1-cp39-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (2.2 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.2/2.2 MB 65.4 MB/s eta 0:00:00
Downloading stevedore-5.5.0-py3-none-any.whl (49 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 49.5/49.5 kB 1.8 MB/s eta 0:00:00
Installing collected packages: stevedore, rustworkx, qiskit, qiskit-aer
Successfully installed qiskit-2.2.3 qiskit-aer-0.17.2 rustworkx-0.17.1 stevedore-5.5.0
Running simulation without error correction...
Running simulation with Shor error correction...

Results:
Without error correction: 0=0.368, 1=0.632
With Shor error correction: 0=0.853, 1=0.147
Deviation from expected 50/50 without EC: 26.40%
Deviation from expected 50/50 with EC: 70.60%
```


Without Error Correction
With Shor Error Correction
Count