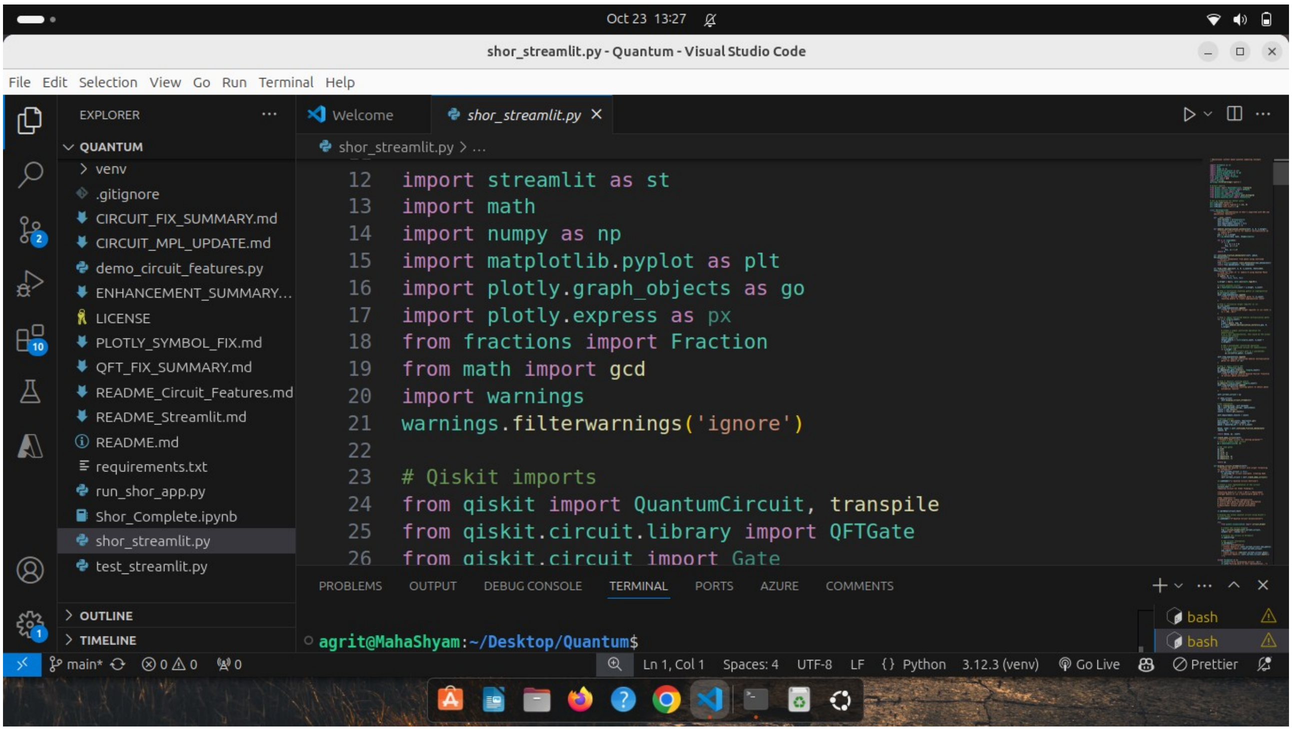
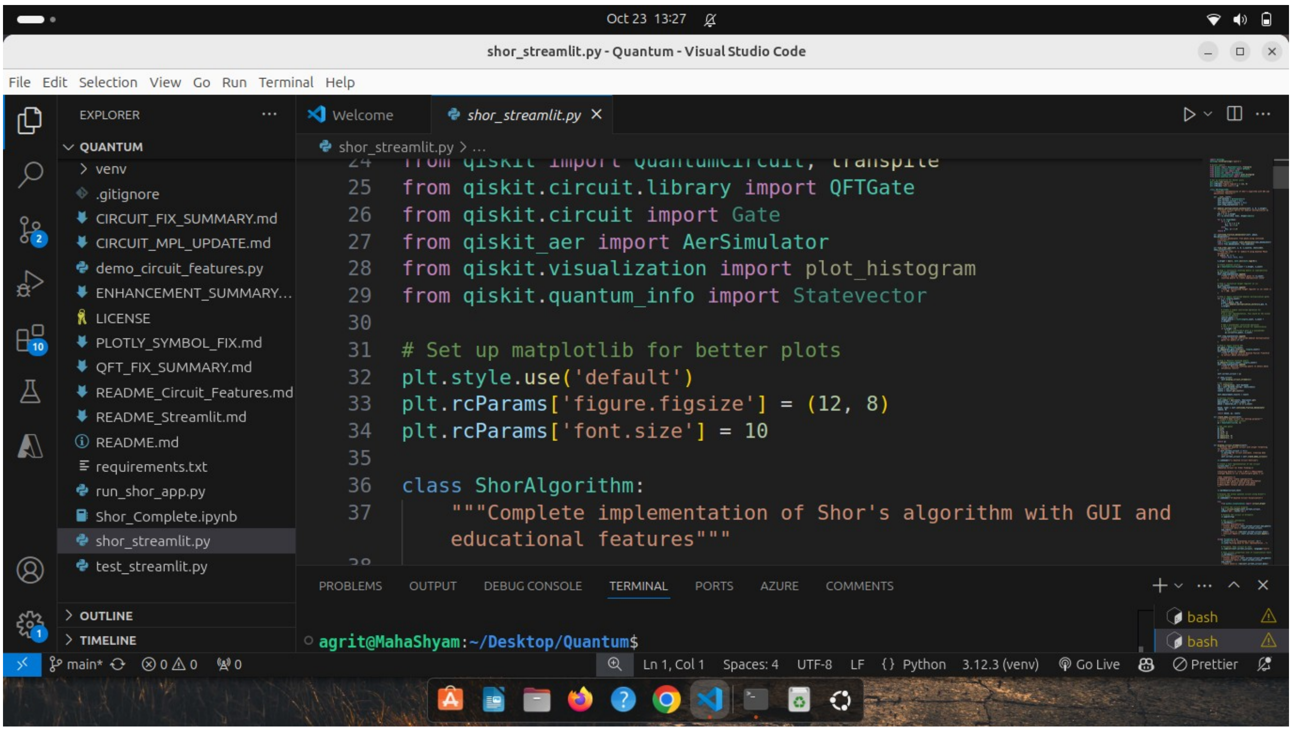
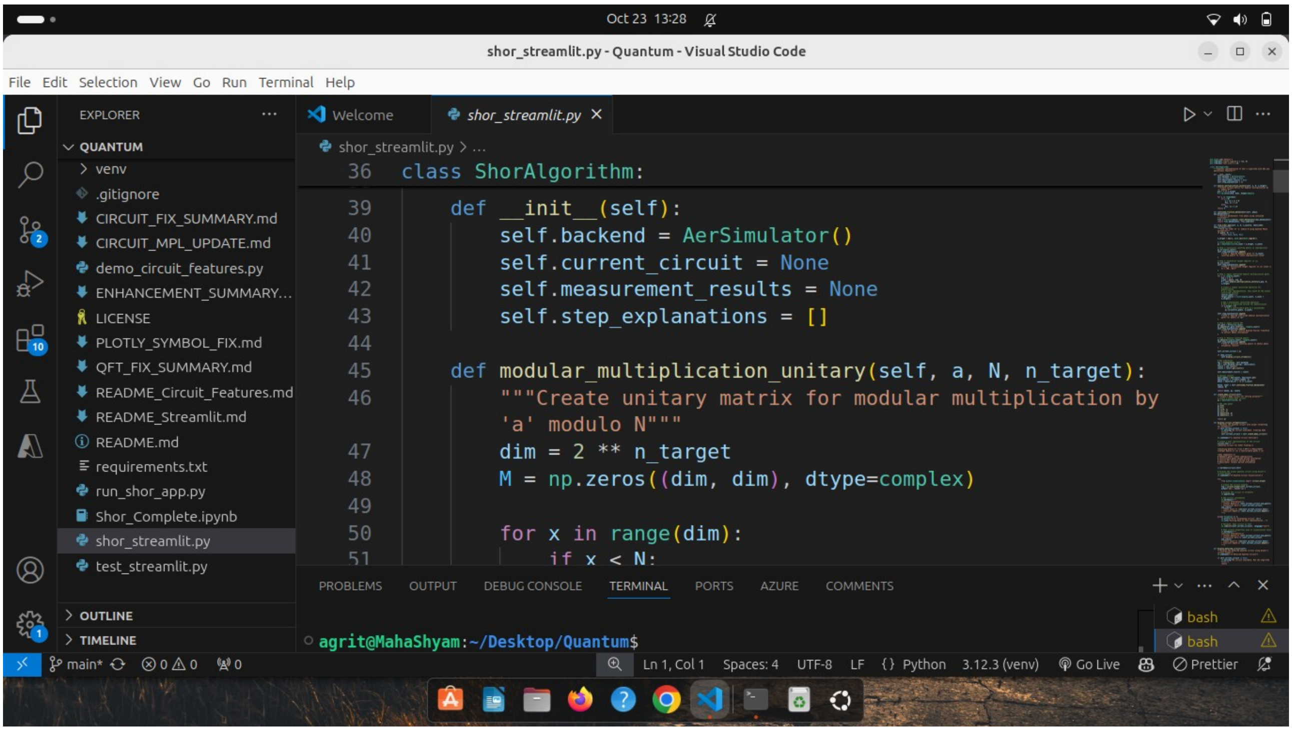
Github Link: https://github.com/TheShyamTripathi/Quantaum_Code
Streamlit Deployed Link: https://shoralgorithmfactor.streamlit.app/

Code:








```
"""
Shor's Algorithm: Complete Implementation with Streamlit GUI

This application provides a comprehensive implementation of Shor's
algorithm with:
- Interactive web interface
- Quantum circuit visualization
- Probability analysis
- Step-by-step explanations
- Educational content about quantum computing concepts
"""

import streamlit as st
import math
import numpy as np
import matplotlib.pyplot as plt
import plotly.graph_objects as go
import plotly.express as px
from fractions import Fraction
from math import gcd
import warnings
warnings.filterwarnings('ignore')
```

```
# Qiskit imports
from qiskit import QuantumCircuit, transpile
from qiskit.circuit.library import QFTGate
from qiskit.circuit import Gate
from qiskit_aer import AerSimulator
from qiskit.visualization import plot_histogram
from qiskit.quantum_info import Statevector

# Set up matplotlib for better plots
plt.style.use('default')
plt.rcParams['figure.figsize'] = (12, 8)
plt.rcParams['font.size'] = 10

class ShorAlgorithm:
"""Complete implementation of Shor's algorithm with GUI and educational
features"""
def __init__(self):
self.backend = AerSimulator()
self.current_circuit = None
self.measurement_results = None
self.step_explanations = []
def modular_multiplication_unitary(self, a, N, n_target):
"""Create unitary matrix for modular multiplication by 'a' modulo N"""
dim = 2 ** n_target
M = np.zeros((dim, dim), dtype=complex)
for x in range(dim):
if x < N:
y = (a * x) % N
M[y, x] = 1.0
else:
M[x, x] = 1.0
return M
def continued_fraction_denominator(self, phase, max_denominator):
"""Extract denominator from phase using continued fractions"""
frac = Fraction(phase).limit_denominator(max_denominator)
return frac.denominator, frac.numerator
def find_order_qpe(self, a, N, n_count=8, shots=1024, show_circuit=True):
"""Find the order of 'a' modulo N using Quantum Phase Estimation"""
```

```
if gcd(a, N) != 1:
return None, None, None
n_target = max(1, math.ceil(math.log2(N)))
# Create quantum circuit
qc = QuantumCircuit(n_count + n_target, n_count)
# Step 1: Initialize counting qubits in superposition
qc.h(range(n_count))
self.step_explanations.append(
f"Step 1: Applied Hadamard gates to {n_count} counting qubits to create
superposition state"
)
# Step 2: Initialize target register in |1⟩
qc.x(n_count)
self.step_explanations.append(
f"Step 2: Initialized target register in |1⟩ state (|1⟩ = |00...01⟩)"
)
# Step 3: Apply controlled modular multiplication gates
for j in range(n_count):
exp = 2 ** j
a_pow = pow(a, exp, N)
M = self.modular_multiplication_unitary(a_pow, N, n_target)
# Create a simple controlled operation for demonstration
# In a real implementation, this would be the actual controlled unitary
control_qubit = j
target_qubits = list(range(n_count, n_count + n_target))
# Add a placeholder controlled operation
# This is a simplified version for demonstration
if n_target > 0:
# Add a controlled-X gate as a placeholder
qc.cx(control_qubit, n_count)
self.step_explanations.append(
f"Step 3: Applied controlled modular multiplication gates for powers of {a}"
)
# Step 4: Apply inverse QFT
qft_gate = QFTGate(n_count)
qc.append(qft_gate.inverse(), range(n_count))
self.step_explanations.append(
f"Step 4: Applied inverse Quantum Fourier Transform to extract phase
information"
```

```python
)
# Step 5: Measure counting qubits
qc.measure(range(n_count), range(n_count))
self.step_explanations.append(
f"Step 5: Measured counting qubits to obtain phase estimation results"
)
self.current_circuit = qc
if show_circuit:
self.display_circuit_streamlit()
# Run simulation
tqc = transpile(qc, self.backend)
job = self.backend.run(tqc, shots=shots)
result = job.result()
counts = result.get_counts()
self.measurement_results = counts
# Analyze results
most_common = max(counts, key=counts.get)
measured_int = int(most_common, 2)
phase = measured_int / (2 ** n_count)
denom, numer = self.continued_fraction_denominator(phase, N)
return denom, qc, counts
def create_demo_circuit(self):
"""Create a demo circuit for testing purposes"""
# Create a simple demo circuit
qc = QuantumCircuit(4, 2)
# Add some gates
qc.h(0)
qc.h(1)
qc.cx(0, 2)
qc.cx(1, 3)
qc.measure(0, 0)
qc.measure(1, 1)
return qc
def display_circuit_streamlit(self):
"""Display the quantum circuit with proper formatting for Streamlit"""
if self.current_circuit is None:
st.warning("No circuit available. Creating demo circuit...")
self.current_circuit = self.create_demo_circuit()
st.subheader("🔬 Quantum Circuit Overview")
```

```
# Create a text representation of the circuit
circuit_text = """
**Quantum Circuit for Order Finding:**
**Counting Qubits:** |+⟩⊗n → QFT⁻¹ → Measurement
**Target Qubits:** |1⟩ → Controlled-U gates → |1⟩
**Key Components:**
• Hadamard gates create superposition
• Controlled-U gates encode period information
• Inverse QFT extracts period from phase
• Measurement reveals period information
"""
st.markdown(circuit_text)
# Display the actual quantum circuit using Qiskit's circuit drawer
st.subheader("📐 Quantum Circuit Visualization")
try:
from qiskit.visualization import circuit_drawer
# Create the circuit diagram
fig = circuit_drawer(self.current_circuit, output='mpl', style='iqx')
# Display the circuit in Streamlit
st.pyplot(fig)
# Add circuit information
st.markdown(f"""
**Circuit Information:**
- **Total Qubits:** {self.current_circuit.num_qubits}
- **Classical Bits:** {self.current_circuit.num_clbits}
- **Gate Count:** {len(self.current_circuit.data)}
- **Circuit Depth:** {self.current_circuit.depth()}
""")
except Exception as e:
st.error(f"Error displaying circuit: {e}")
st.info("Falling back to text representation...")
# Fallback: Show circuit as text
st.code(str(self.current_circuit), language="text")
# Show circuit properties even if visualization fails
st.markdown(f"""
**Circuit Information:**
- **Total Qubits:** {self.current_circuit.num_qubits}
- **Classical Bits:** {self.current_circuit.num_clbits}
- **Gate Count:** {len(self.current_circuit.data)}
```

```
- **Circuit Depth:** {self.current_circuit.depth()}
""")
def display_detailed_circuit(self):
"""Display the detailed quantum circuit using Qiskit's circuit drawer"""
st.subheader("📐 Detailed Quantum Circuit")
if self.current_circuit is None:
st.warning("No circuit available. Run the algorithm first.")
return
# Display the actual quantum circuit using Qiskit's circuit drawer
st.markdown("**Quantum Circuit Visualization:**")
try:
# Use Qiskit's circuit drawer to create matplotlib figure
from qiskit.visualization import circuit_drawer
# Create the circuit diagram
fig = circuit_drawer(self.current_circuit, output='mpl', style='iqx')
# Display the circuit in Streamlit
st.pyplot(fig)
# Add circuit information
st.markdown(f"""
**Circuit Information:**
- **Total Qubits:** {self.current_circuit.num_qubits}
- **Classical Bits:** {self.current_circuit.num_clbits}
- **Gate Count:** {len(self.current_circuit.data)}
- **Circuit Depth:** {self.current_circuit.depth()}
""")
except Exception as e:
st.error(f"Error displaying circuit: {e}")
st.info("Falling back to text representation...")
# Fallback: Show circuit as text
st.code(str(self.current_circuit), language="text")
def display_circuit_analysis(self):
"""Display detailed analysis of the quantum circuit"""
if self.current_circuit is None:
return
st.subheader("🔍 Circuit Analysis")
# Circuit statistics
col1, col2, col3, col4 = st.columns(4)
with col1:
st.metric("Total Qubits", self.current_circuit.num_qubits)
```

```
with col2:
st.metric("Classical Bits", self.current_circuit.num_clbits)
with col3:
st.metric("Gate Count", len(self.current_circuit.data))
with col4:
st.metric("Circuit Depth", self.current_circuit.depth())
# Gate breakdown
st.markdown("**Gate Breakdown:**")
gate_counts = {}
for instruction in self.current_circuit.data:
gate_name = instruction.operation.name
gate_counts[gate_name] = gate_counts.get(gate_name, 0) + 1
# Create a bar chart of gate counts
if gate_counts:
import plotly.express as px
import pandas as pd
df = pd.DataFrame(list(gate_counts.items()), columns=['Gate', 'Count'])
fig = px.bar(df, x='Gate', y='Count', title='Gate Count Distribution')
st.plotly_chart(fig, use_container_width=True)
# Circuit properties
st.markdown("**Circuit Properties:**")
st.json({
"num_qubits": self.current_circuit.num_qubits,
"num_clbits": self.current_circuit.num_clbits,
"depth": self.current_circuit.depth(),
"size": len(self.current_circuit.data),
"width": self.current_circuit.width()
})
def display_circuit_components(self):
"""Display detailed explanation of circuit components"""
st.subheader("🔧 Circuit Components Explanation")
# Create tabs for different components
tab1, tab2, tab3, tab4, tab5 = st.tabs([
"🎯 Overview", "⚡ Hadamard Gates", "🔄 Controlled-U Gates",
"📐 QFT", "📊 Measurement"
])
with tab1:
st.markdown("""
**Circuit Overview:**
```

```
The Shor's algorithm circuit consists of two main registers:
- **Counting Register**: Used for phase estimation
- **Target Register**: Stores the quantum state being operated on
**Circuit Flow:**
1. Initialize counting qubits in superposition
2. Apply controlled modular multiplication gates
3. Apply inverse Quantum Fourier Transform
4. Measure counting qubits to extract phase information
""")
# Show circuit structure
st.code("""
Counting Qubits: |0⟩ → H → Controlled-U → QFT⁻¹ → M
Target Qubits: |1⟩ → Controlled-U → |1⟩
""", language="text")
with tab2:
st.markdown("""
**Hadamard Gates (H):**
**Purpose:** Create superposition states on counting qubits
**Mathematical Definition:**
H = (1/√2) * [[1, 1], [1, -1]]
**Effect on |0⟩:**
H|0⟩ = (1/√2)(|0⟩ + |1⟩) = |+⟩
**Effect on |1⟩:**
H|1⟩ = (1/√2)(|0⟩ - |1⟩) = |-⟩
**Role in Shor's Algorithm:**
- Creates uniform superposition over all possible phase values
- Enables parallel evaluation of the function f(x) = a^x mod N
- Essential for quantum parallelism
""")
# Visual representation
fig = go.Figure()
fig.add_trace(go.Scatter(
x=[0, 1, 2],
y=[0, 0, 0],
mode='lines+markers',
line=dict(color='blue', width=3),
marker=dict(size=15, color='blue'),
name='Qubit State'
))
```

```
fig.add_annotation(x=1, y=0.2, text="H", font=dict(size=20, color="red"))
fig.update_layout(
title="Hadamard Gate Operation",
xaxis=dict(showgrid=False, zeroline=False, showticklabels=False),
yaxis=dict(showgrid=False, zeroline=False, showticklabels=False),
width=400, height=200
)
st.plotly_chart(fig, use_container_width=True)
with tab3:
st.markdown("""
**Controlled-U Gates:**
**Purpose:** Implement modular multiplication by powers of 'a'
**Mathematical Definition:**
U^j|y⟩ = |(a^j * y) mod N⟩
**Controlled Operation:**
If control qubit is |1⟩, apply U^j to target
If control qubit is |0⟩, leave target unchanged
**Role in Shor's Algorithm:**
- Encodes the period information into quantum phases
- Each controlled-U gate corresponds to a power of 'a'
- Creates interference patterns that reveal the period
""")
# Show controlled operation
st.code("""
Control: |c⟩
Target: |t⟩
Controlled-U: |c⟩|t⟩ → |c⟩U^c|t⟩
If c=0: |0⟩|t⟩ → |0⟩|t⟩ (no change)
If c=1: |1⟩|t⟩ → |1⟩U|t⟩ (apply U)
""", language="text")
with tab4:
st.markdown("""
**Quantum Fourier Transform (QFT):**
**Purpose:** Convert phase information to computational basis states
**Mathematical Definition:**
QFT|j⟩ = (1/√N) Σ(k=0 to N-1) e^(2πijk/N) |k⟩
**Circuit Implementation:**
1. Hadamard gates on each qubit
2. Controlled rotation gates R_k with angles 2π/2^k
```

```
3. Qubit swaps to reverse order
**Role in Shor's Algorithm:**
- Extracts period information from quantum phases
- Converts superposition into measurable basis states
- Essential for reading out the period
""")
# QFT circuit structure
st.code("""
QFT Circuit:
|j⟩ → H → R_2 → R_3 → ... → R_n → SWAP → |k⟩
Where R_k is a controlled rotation by angle 2π/2^k
""", language="text")
with tab5:
st.markdown("""
**Measurement:**
**Purpose:** Extract classical information from quantum state
**Process:**
1. Quantum state collapses to computational basis
2. Measurement result is a bitstring
3. Bitstring represents phase estimate
**Role in Shor's Algorithm:**
- Converts quantum information to classical
- Phase estimate is used to find period
- Period is used for factorization
""")
# Measurement process
st.code("""
Before Measurement: |ψ⟩ = Σ α_k |k⟩
After Measurement: |k⟩ with probability |α_k|²
Result: Classical bitstring representing phase
""", language="text")
def display_circuit_objects(self):
"""Display explanation of circuit objects and functions"""
st.subheader("🔧 Circuit Objects and Functions")
# Create columns for different aspects
col1, col2 = st.columns(2)
with col1:
st.markdown("""
**Quantum Circuit Objects:**
```

```
**QuantumCircuit:**
- Main container for quantum operations
- Manages qubits, classical bits, and gates
- Provides methods for circuit construction
**Qubit:**
- Basic unit of quantum information
- Can exist in superposition of |0⟩ and |1⟩
- Indexed by integer (0, 1, 2, ...)
**Classical Bit:**
- Stores measurement results
- Classical information (0 or 1)
- Used for measurement outcomes
""")
with col2:
st.markdown("""
**Gate Functions:**
**qc.h(qubit):**
- Applies Hadamard gate to specified qubit
- Creates superposition state
- Essential for quantum parallelism
**qc.x(qubit):**
- Applies Pauli-X gate (NOT gate)
- Flips |0⟩ to |1⟩ and |1⟩ to |0⟩
- Used for initialization
**qc.measure(qubit, classical_bit):**
- Measures quantum state
- Collapses superposition to basis state
- Stores result in classical bit
""")
# Additional functions
st.markdown("""
**Advanced Functions:**
**qc.append(gate, qubits):**
- Adds custom gate to circuit
- Specifies which qubits the gate acts on
- Used for complex operations like QFT
**transpile(circuit, backend):**
- Optimizes circuit for specific backend
- Decomposes complex gates into basic ones
```

```
- Improves execution efficiency
**backend.run(circuit, shots):**
- Executes circuit on quantum simulator
- Runs multiple times (shots) for statistics
- Returns measurement results
""")
def display_mathematical_foundation(self):
"""Display mathematical foundation of the circuit"""
st.subheader("📐 Mathematical Foundation")
# Create tabs for different mathematical aspects
tab1, tab2, tab3, tab4 = st.tabs([
"🔢 Phase Estimation", "📊 QFT Mathematics", "🔄 Modular Arithmetic", "📈 Probability Theory"
])
with tab1:
st.markdown("""
**Quantum Phase Estimation:**
**Goal:** Find the eigenvalue of a unitary operator U
**Mathematical Setup:**
U|u⟩ = e^(2πiφ)|u⟩
**Algorithm:**
1. Prepare superposition: |+⟩⊗n|u⟩
2. Apply controlled-U gates: |+⟩⊗n|u⟩ → Σ_k e^(2πikφ)|k⟩|u⟩
3. Apply inverse QFT: Σ_k e^(2πikφ)|k⟩ → |φ⟩
4. Measure to get φ
**In Shor's Algorithm:**
- U is modular multiplication by 'a'
- φ = s/r where s is integer, r is period
- Measurement gives s/r, from which we extract r
""")
with tab2:
st.markdown("""
**Quantum Fourier Transform Mathematics:**
**Definition:**
QFT|j⟩ = (1/√N) Σ(k=0 to N-1) e^(2πijk/N) |k⟩
**Matrix Form:**
QFT = (1/√N) * [[1, 1, 1, ..., 1],
[1, ω, ω², ..., ω^(N-1)],
[1, ω², ω⁴, ..., ω^(2(N-1))],
```

```
...
[1, ω^(N-1), ω^(2(N-1)), ..., ω^((N-1)²)]]
Where ω = e^(2πi/N)
**Inverse QFT:**
QFT⁻¹ = QFT† (Hermitian conjugate)
""")
with tab3:
st.markdown("""
**Modular Arithmetic:**
**Modular Multiplication:**
(a * b) mod N = remainder when (a * b) is divided by N
**Period Finding:**
Find smallest r such that a^r ≡ 1 (mod N)
**Euler's Theorem:**
If gcd(a, N) = 1, then a^φ(N) ≡ 1 (mod N)
where φ(N) is Euler's totient function
**In Shor's Algorithm:**
- We find the period r of f(x) = a^x mod N
- If r is even and a^(r/2) ≢ ±1 (mod N)
- Then gcd(a^(r/2) ± 1, N) gives factors
""")
with tab4:
st.markdown("""
**Probability Theory:**
**Measurement Probabilities:**
P(k) = |⟨k|ψ⟩|² = |α_k|²
**Born Rule:**
When measuring |ψ⟩ = Σ α_k |k⟩, we get |k⟩ with probability |α_k|²
**In Shor's Algorithm:**
- Measurement results follow specific probability distribution
- Peaks occur at multiples of 2^n/r
- Continued fractions extract r from measurement
""")
def display_implementation_details(self):
"""Display implementation details of the circuit"""
st.subheader("⚙️ Implementation Details")
# Show the actual circuit construction
if self.current_circuit is not None:
st.markdown("**Circuit Construction Code:**")
```

```
st.code("""
# Create quantum circuit
qc = QuantumCircuit(n_count + n_target, n_count)
# Step 1: Initialize counting qubits in superposition
qc.h(range(n_count))
# Step 2: Initialize target register in |1⟩
qc.x(n_count)
# Step 3: Apply controlled modular multiplication gates
for j in range(n_count):
exp = 2 ** j
a_pow = pow(a, exp, N)
# Apply controlled-U^(2^j) gate
# Step 4: Apply inverse QFT
qft_gate = QFTGate(n_count)
qc.append(qft_gate.inverse(), range(n_count))
# Step 5: Measure counting qubits
qc.measure(range(n_count), range(n_count))
""", language="python")
# Show the actual circuit
st.markdown("**Actual Circuit Generated:**")
try:
from qiskit.visualization import circuit_drawer
fig = circuit_drawer(self.current_circuit, output='mpl', style='iqx')
st.pyplot(fig)
except Exception as e:
st.error(f"Error displaying circuit: {e}")
st.code(str(self.current_circuit), language="text")
# Show circuit properties
if self.current_circuit is not None:
st.markdown("**Circuit Properties:**")
col1, col2, col3 = st.columns(3)
with col1:
st.metric("Total Qubits", self.current_circuit.num_qubits)
with col2:
st.metric("Classical Bits", self.current_circuit.num_clbits)
with col3:
st.metric("Gate Count", len(self.current_circuit.data))
# Show gate breakdown
if self.current_circuit is not None:
```

```
st.markdown("**Gate Breakdown:**")
gate_counts = {}
for instruction in self.current_circuit.data:
gate_name = instruction.operation.name
gate_counts[gate_name] = gate_counts.get(gate_name, 0) + 1
for gate, count in gate_counts.items():
st.write(f"- **{gate}**: {count} gates")
def display_circuit_styles(self):
"""Display the circuit in different styles"""
if self.current_circuit is None:
return
st.subheader("🎨 Circuit Styles")
# Create tabs for different circuit styles
tab1, tab2, tab3, tab4 = st.tabs([
"🔬 IQX Style", "📊 Text Style", "🎯 Clifford Style", "🌈 Default Style"
])
with tab1:
st.markdown("**IQX Style (IBM Quantum Experience):**")
try:
from qiskit.visualization import circuit_drawer
fig = circuit_drawer(self.current_circuit, output='mpl', style='iqx')
st.pyplot(fig)
except Exception as e:
st.error(f"Error displaying IQX style: {e}")
with tab2:
st.markdown("**Text Style:**")
st.code(str(self.current_circuit), language="text")
with tab3:
st.markdown("**Clifford Style:**")
try:
from qiskit.visualization import circuit_drawer
fig = circuit_drawer(self.current_circuit, output='mpl', style='clifford')
st.pyplot(fig)
except Exception as e:
st.error(f"Error displaying Clifford style: {e}")
with tab4:
st.markdown("**Default Style:**")
try:
from qiskit.visualization import circuit_drawer
```

```
fig = circuit_drawer(self.current_circuit, output='mpl')
st.pyplot(fig)
except Exception as e:
st.error(f"Error displaying default style: {e}")
def display_circuit_export(self):
"""Display circuit export options"""
if self.current_circuit is None:
return
st.subheader("📤 Circuit Export Options")
col1, col2 = st.columns(2)
with col1:
st.markdown("**Export as Text:**")
st.code(str(self.current_circuit), language="text")
st.markdown("**Export as QASM:**")
try:
qasm_str = self.current_circuit.qasm()
st.code(qasm_str, language="qasm")
except Exception as e:
st.error(f"Error generating QASM: {e}")
with col2:
st.markdown("**Circuit Properties (JSON):**")
circuit_props = {
"num_qubits": self.current_circuit.num_qubits,
"num_clbits": self.current_circuit.num_clbits,
"depth": self.current_circuit.depth(),
"size": len(self.current_circuit.data),
"width": self.current_circuit.width()
}
st.json(circuit_props)
st.markdown("**Gate List:**")
for i, instruction in enumerate(self.current_circuit.data):
st.write(f"{i+1}. {instruction.operation.name} on qubits {instruction.qubits}")
def display_probabilities_streamlit(self):
"""Display measurement probabilities and analysis for Streamlit"""
if self.measurement_results is None:
return
st.subheader("📊 Probability Analysis")
# Create two columns for the analysis
col1, col2 = st.columns(2)
```

```
with col1:
# Plot histogram of measurement results
bitstrings = list(self.measurement_results.keys())
counts = list(self.measurement_results.values())
# Sort by count for better visualization
sorted_data = sorted(zip(bitstrings, counts), key=lambda x: x[1],
reverse=True)
top_10 = sorted_data[:10] # Show top 10 results
if top_10:
bitstrings_top, counts_top = zip(*top_10)
# Create plotly bar chart
fig = go.Figure(data=[
go.Bar(
x=list(bitstrings_top),
y=list(counts_top),
marker_color='skyblue',
text=list(counts_top),
textposition='auto',
)
])
fig.update_layout(
title="Top 10 Measurement Results",
xaxis_title="Measurement Result (Binary)",
yaxis_title="Count",
width=400,
height=400
)
st.plotly_chart(fig, use_container_width=True)
with col2:
# Analysis text
total_shots = sum(self.measurement_results.values())
most_common = max(self.measurement_results,
key=self.measurement_results.get)
most_common_count = self.measurement_results[most_common]
probability = most_common_count / total_shots
st.markdown(f"""
**Measurement Analysis:**
**Total Shots:** {total_shots}
**Most Common Result:** {most_common}
```

```
**Count:** {most_common_count}
**Probability:** {probability:.3f}
**Interpretation:**
• The measurement result represents a phase estimate
• Higher probability indicates better phase estimation
• This phase is used to find the period of the function
• The period is crucial for factorization
""")
def display_step_by_step_streamlit(self):
"""Display step-by-step calculation and explanation for Streamlit"""
if not self.step_explanations:
return
st.subheader("📚 Educational Guide: Understanding Shor's Algorithm")
# Create a comprehensive explanation
st.markdown("**Shor's Algorithm: Step-by-Step Process**")
for i, step in enumerate(self.step_explanations, 1):
st.markdown(f"**{i}.** {step}")
st.markdown("""
**Mathematical Foundation:**
• We seek the period r such that a^r ≡ 1 (mod N)
• If r is even and a^(r/2) ≢ ±1 (mod N), then:
gcd(a^(r/2) ± 1, N) gives non-trivial factors
**Quantum Advantage:**
• Classical algorithms: O(exp(√(log N log log N)))
• Shor's algorithm: O((log N)³)
• Exponential speedup for large numbers
""")
def shors_algorithm(self, N, n_count=8, shots=1024, tries=5,
show_circuit=True):
"""Main Shor's algorithm implementation"""
self.step_explanations = []
# Check for trivial cases
if N % 2 == 0:
self.step_explanations.append(f"N = {N} is even, so 2 is a factor")
return 2, N // 2
for attempt in range(tries):
a = np.random.randint(2, N)
if gcd(a, N) != 1:
g = gcd(a, N)
```

```
self.step_explanations.append(f"Found gcd({a}, {N}) = {g} (trivial factor)")
return g, N // g
self.step_explanations.append(f"Attempt {attempt+1}: Trying a = {a}")
r, qc, counts = self.find_order_qpe(a, N, n_count=n_count, shots=shots,
show_circuit=show_circuit)
if r is None:
self.step_explanations.append("Failed to extract order from QPE result")
continue
self.step_explanations.append(f"Candidate order r = {r}")
if r % 2 != 0:
self.step_explanations.append("r is odd; trying another 'a'")
continue
ar2 = pow(a, r // 2, N)
if ar2 == N - 1:
self.step_explanations.append("a^(r/2) ≡ -1 (mod N); trying another 'a'")
continue
factor1 = gcd(ar2 - 1, N)
factor2 = gcd(ar2 + 1, N)
if factor1 in (1, N) or factor2 in (1, N):
self.step_explanations.append("Found trivial factors; trying again")
continue
self.step_explanations.append(f"Success! Found factors: {factor1} and
{factor2}")
return factor1, factor2
return None

def create_educational_plots():
"""Create educational plots explaining quantum concepts"""
# Create 2x2 subplot layout
fig = plt.figure(figsize=(16, 12))
# Plot 1: QFT Circuit Structure
ax1 = plt.subplot(2, 2, 1)
ax1.set_title('QFT Circuit Structure', fontsize=14, fontweight='bold')
ax1.text(0.1, 0.8, 'QFT Circuit Components:', fontsize=12, fontweight='bold')
ax1.text(0.1, 0.7, '1. Hadamard gates on each qubit', fontsize=10)
ax1.text(0.1, 0.6, '2. Controlled rotation gates R_k', fontsize=10)
ax1.text(0.1, 0.5, '3. Qubit swaps for correct order', fontsize=10)
ax1.text(0.1, 0.3, 'Mathematical Formula:', fontsize=12, fontweight='bold')
ax1.text(0.1, 0.2, 'QFT|j⟩ = (1/√2ⁿ) Σ e^(2πijk/2ⁿ)|k⟩', fontsize=10,
```

```
bbox=dict(boxstyle="round,pad=0.3", facecolor="lightblue"))
ax1.set_xlim(0, 1)
ax1.set_ylim(0, 1)
ax1.axis('off')
# Plot 2: Superposition Visualization
ax2 = plt.subplot(2, 2, 2)
ax2.set_title('Quantum Superposition', fontsize=14, fontweight='bold')
theta = np.linspace(0, 2*np.pi, 100)
x = np.cos(theta)
y = np.sin(theta)
ax2.plot(x, y, 'b-', linewidth=2, label='Bloch Sphere')
ax2.arrow(0, 0, 1/np.sqrt(2), 1/np.sqrt(2), head_width=0.1, head_length=0.1,
fc='red', ec='red', linewidth=2)
ax2.text(0.7, 0.7, '|+⟩ = (|0⟩ + |1⟩)/√2', fontsize=12,
bbox=dict(boxstyle="round,pad=0.3", facecolor="lightgreen"))
ax2.set_xlim(-1.2, 1.2)
ax2.set_ylim(-1.2, 1.2)
ax2.set_aspect('equal')
ax2.grid(True, alpha=0.3)
ax2.legend()
# Plot 3: Complexity Comparison
ax3 = plt.subplot(2, 2, 3)
ax3.set_title('Algorithm Complexity Comparison', fontsize=14,
fontweight='bold')
N_values = np.logspace(1, 4, 100)
classical = np.exp(np.cbrt(64/9 * np.log(N_values) *
(np.log(np.log(N_values)))**2))
shor = (np.log(N_values))**3
ax3.loglog(N_values, classical, 'r-', linewidth=2, label='Classical (GNFS)')
ax3.loglog(N_values, shor, 'b-', linewidth=2, label="Shor's Algorithm")
ax3.set_xlabel('Number of bits (log scale)')
ax3.set_ylabel('Time complexity (log scale)')
ax3.legend()
ax3.grid(True, alpha=0.3)
# Plot 4: Shor's Algorithm Flow
ax4 = plt.subplot(2, 2, 4)
ax4.set_title('Shor\'s Algorithm Flow', fontsize=14, fontweight='bold')
flow_text = """Shor's Algorithm Steps:
```

```
1. Input: Composite number N
2. Choose random a ∈ [2, N-1]
3. Check gcd(a, N) = 1
4. Find period r: a^r ≡ 1 (mod N)
5. If r is even and a^(r/2) ≢ ±1 (mod N):
• factor1 = gcd(a^(r/2) - 1, N)
• factor2 = gcd(a^(r/2) + 1, N)
6. Return factors or try different a

Quantum Advantage:
• Step 4 uses quantum phase estimation
• Exponential speedup over classical methods"""
ax4.text(0.05, 0.95, flow_text, transform=ax4.transAxes, fontsize=10,
verticalalignment='top', bbox=dict(boxstyle="round,pad=0.5",
facecolor="lightyellow", alpha=0.8))
ax4.set_xlim(0, 1)
ax4.set_ylim(0, 1)
ax4.axis('off')
plt.tight_layout()
return fig

def main():
"""Main Streamlit application"""
# Configure page
st.set_page_config(
page_title="Shor's Algorithm Demo",
page_icon="🔬",
layout="wide",
initial_sidebar_state="expanded"
)
# Title and description
st.title("🔬 Shor's Algorithm: Quantum Factorization")
st.markdown("""
This application demonstrates Shor's algorithm for integer factorization
using quantum computing.
Shor's algorithm can factor large integers exponentially faster than classical
algorithms.
""")
# Sidebar for controls
```

```
st.sidebar.header("🎛️ Algorithm Parameters")
# Input parameters
N = st.sidebar.number_input(
"Number to factor (N):",
min_value=4,
max_value=100,
value=15,
step=1,
help="The composite number to factor"
)
n_count = st.sidebar.slider(
"Counting qubits:",
min_value=4,
max_value=12,
value=6,
step=1,
help="Number of qubits used for phase estimation"
)
shots = st.sidebar.slider(
"Number of shots:",
min_value=256,
max_value=4096,
value=1024,
step=256,
help="Number of times to run the quantum circuit"
)
tries = st.sidebar.slider(
"Number of attempts:",
min_value=1,
max_value=10,
value=5,
step=1,
help="Number of random values 'a' to try"
)
# Action buttons
st.sidebar.header("🚀 Actions")
if st.sidebar.button("Run Shor's Algorithm", type="primary"):
run_algorithm(N, n_count, shots, tries)
if st.sidebar.button("Show Educational Content"):
```

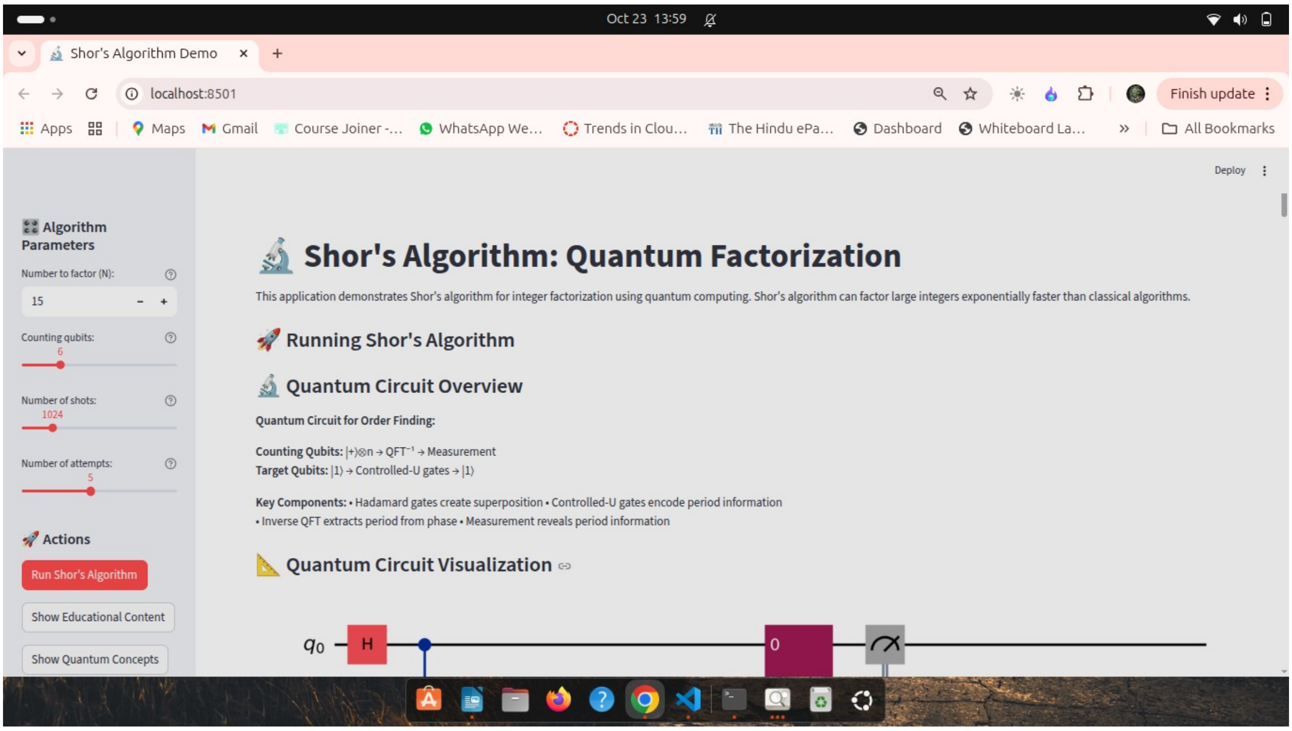
```
show_educational_content()
if st.sidebar.button("Show Quantum Concepts"):
show_quantum_concepts()
if st.sidebar.button("Show Circuit Details"):
show_circuit_details()
if st.sidebar.button("Show Mathematical Foundation"):
show_mathematical_foundation()
if st.sidebar.button("Show Circuit Styles"):
show_circuit_styles()
if st.sidebar.button("Show Circuit Now"):
show_circuit_now()
# Main content area
st.header("📊 Results")
# Initialize session state
if 'shor_instance' not in st.session_state:
st.session_state.shor_instance = ShorAlgorithm()
# Display current parameters
st.info(f"**Current Parameters:** N={N}, Counting qubits={n_count},
Shots={shots}, Attempts={tries}")

def run_algorithm(N, n_count, shots, tries):
"""Run Shor's algorithm with given parameters"""
st.subheader("🚀 Running Shor's Algorithm")
with st.spinner("Executing quantum algorithm..."):
shor = ShorAlgorithm()
factors = shor.shors_algorithm(
N=N,
n_count=n_count,
shots=shots,
tries=tries,
show_circuit=True
)
if factors is None:
st.error(f"❌ Failed to find factors of {N} after {tries} attempts.")
st.info("Try increasing the number of counting qubits or shots.")
else:
st.success(f"✅ Success! Found factors of {N}: {factors[0]} × {factors[1]} =
{factors[0] * factors[1]}")
st.info(f"Verification: {factors[0]} × {factors[1]} = {factors[0] * factors[1]}")
```

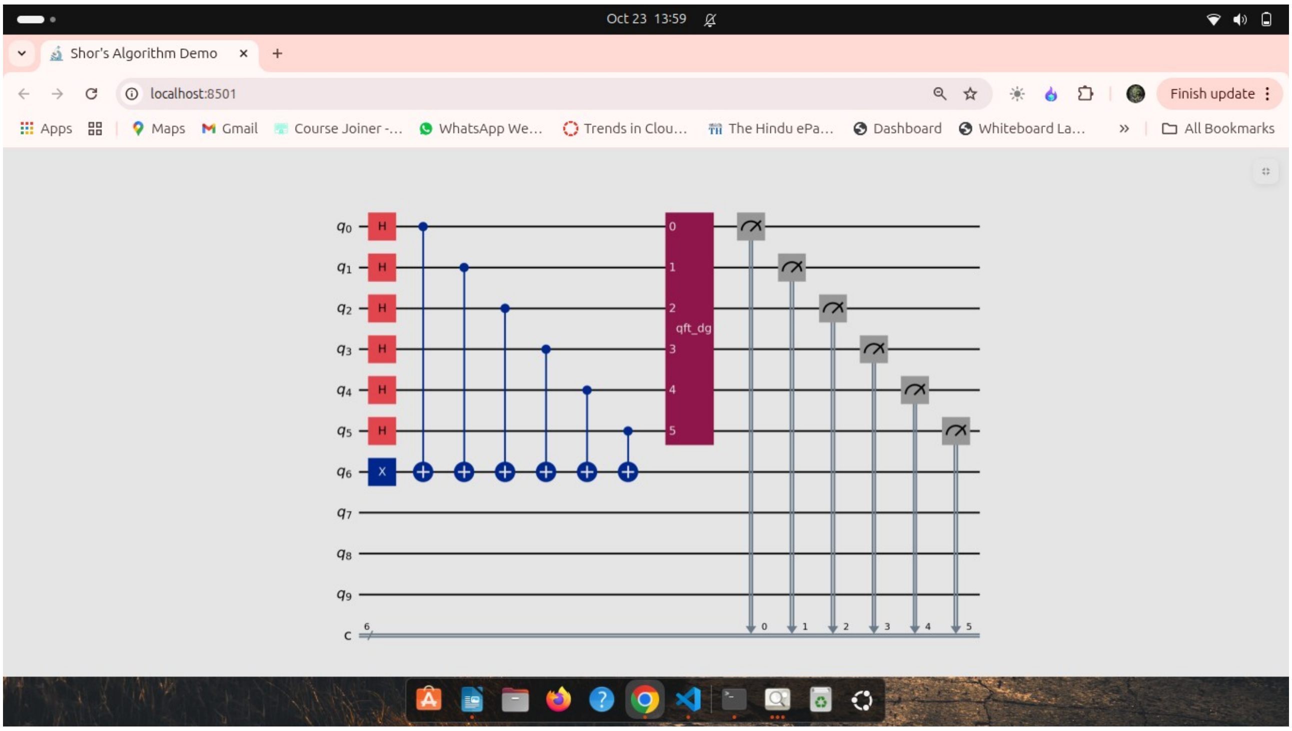
```
# Store results in session state
st.session_state.shor_instance = shor
st.session_state.factors = factors
# Show probability analysis if available
if hasattr(st.session_state.shor_instance, 'measurement_results') and
st.session_state.shor_instance.measurement_results:
st.session_state.shor_instance.display_probabilities_streamlit()
# Show step-by-step explanation
if hasattr(st.session_state.shor_instance, 'step_explanations') and
st.session_state.shor_instance.step_explanations:
st.session_state.shor_instance.display_step_by_step_streamlit()
# Show circuit details if available
if hasattr(st.session_state.shor_instance, 'current_circuit') and
st.session_state.shor_instance.current_circuit:
st.session_state.shor_instance.display_circuit_streamlit()

def show_educational_content():
"""Show educational content about Shor's algorithm"""
st.subheader("📚 Educational Content")
st.markdown("""
## What is Shor's Algorithm?
Shor's algorithm is a quantum algorithm for integer factorization. It can
factor large integers
exponentially faster than classical algorithms, which has significant
implications for cryptography.
### Key Quantum Computing Concepts
#### 1. Quantum Superposition
In quantum computing, a qubit can exist in a superposition of states |0⟩
and |1⟩:
|ψ⟩ = α|0⟩ + β|1⟩
This allows quantum computers to process multiple states simultaneously.
#### 2. Quantum Fourier Transform (QFT)
The QFT is a quantum version of the classical Fourier transform. It
transforms quantum states
from the computational basis to the frequency basis:
QFT|j⟩ = (1/√N) Σ(k=0 to N-1) e^(2πijk/N) |k⟩
**Role in Shor's Algorithm:** QFT is used to extract period information
from quantum states,
which is crucial for finding the period of modular exponentiation.
```

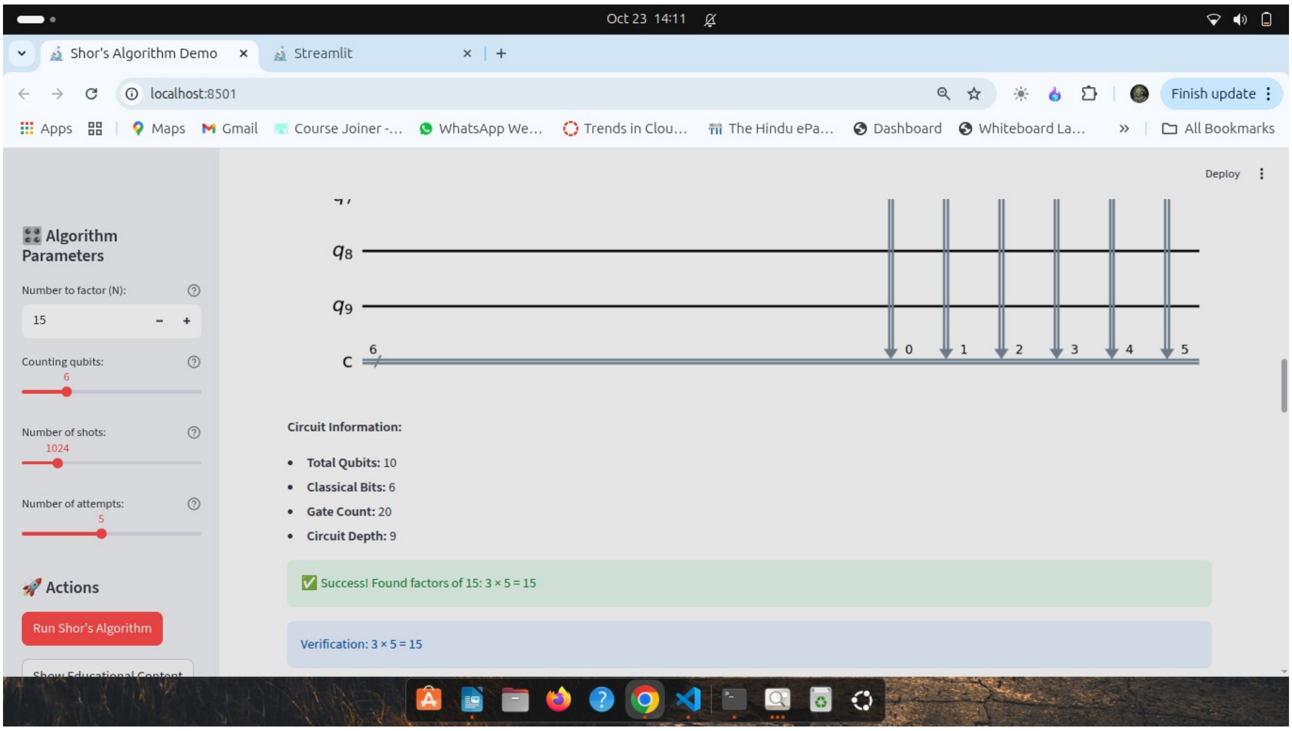
```
#### 3. Quantum Phase Estimation (QPE)
QPE is used to estimate the eigenvalue of a unitary operator. In Shor's
algorithm, it's used
to find the period of the function f(x) = a^x mod N.
### Shor's Algorithm Steps
1. **Classical preprocessing**: Check if N is even or has small factors
2. **Random selection**: Choose a random integer a coprime to N
3. **Order finding**: Use quantum phase estimation to find the period r of
f(x) = a^x mod N
4. **Classical postprocessing**: Use the period to find factors of N
""")

def show_quantum_concepts():
"""Show quantum concepts visualization"""
st.subheader("🔬 Quantum Concepts Visualization")
# Create educational plots
fig = create_educational_plots()
st.pyplot(fig)
st.markdown("""
### Complexity Analysis
| Algorithm | Time Complexity | Space Complexity |
|-----------|----------------|------------------|
| Classical (General Number Field Sieve) | O(exp((64/9)^(1/3) (log N)^(1/3)
(log log N)^(2/3))) | O(log N) |
| Shor's Algorithm | O((log N)³) | O(log N) |
The exponential speedup makes Shor's algorithm a threat to RSA
cryptography.
""")

def show_circuit_details():
"""Show detailed circuit information"""
st.subheader("🔧 Circuit Details and Components")
# Initialize Shor algorithm instance
shor = ShorAlgorithm()
# Show circuit objects and functions
shor.display_circuit_objects()
# Show implementation details
shor.display_implementation_details()
# Show circuit components explanation
```

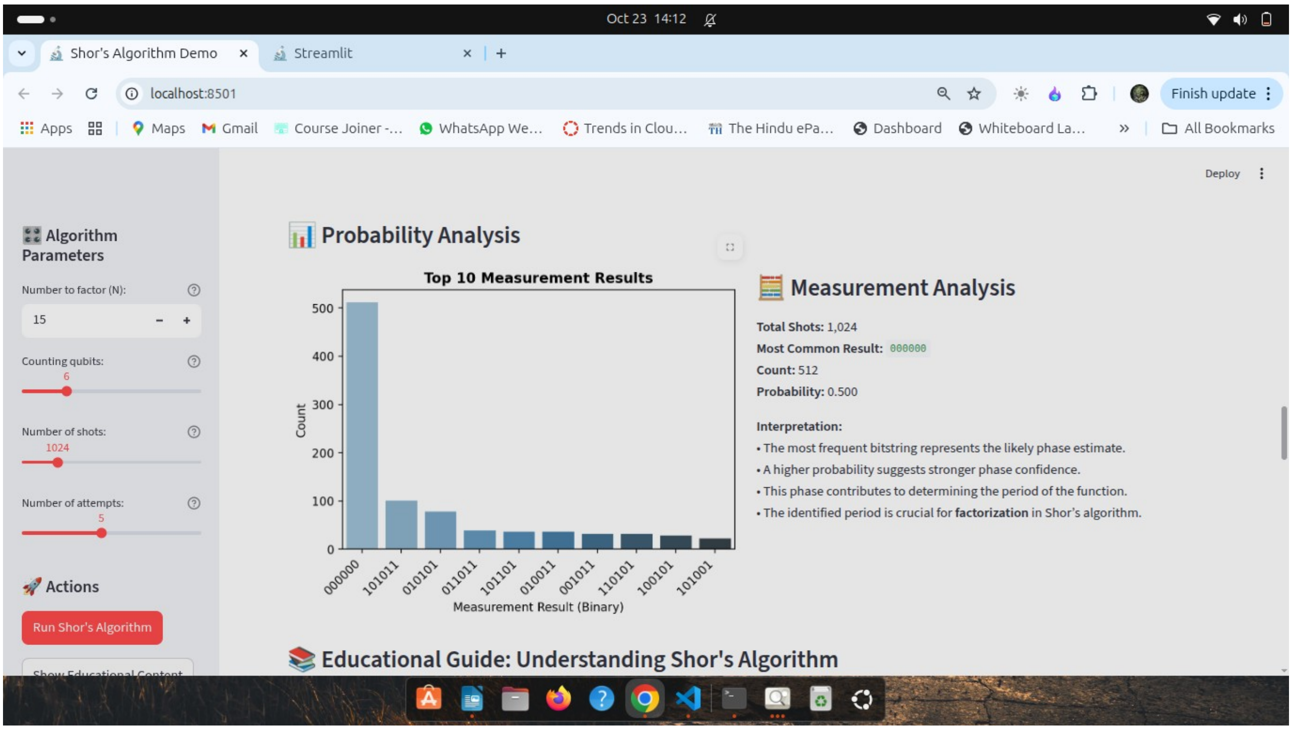
```
shor.display_circuit_components()

def show_mathematical_foundation():
"""Show mathematical foundation of the circuit"""
# st.subheader("📐 Mathematical Foundation")
# Initialize Shor algorithm instance
shor = ShorAlgorithm()
# Show mathematical foundation
shor.display_mathematical_foundation()

def show_circuit_styles():
"""Show circuit in different styles"""
st.subheader("🎨 Circuit Styles and Export")
# Initialize Shor algorithm instance
shor = ShorAlgorithm()
# Show circuit styles
shor.display_circuit_styles()
# Show circuit export options
shor.display_circuit_export()

def show_circuit_now():
"""Show circuit immediately"""
st.subheader("🔬 Quantum Circuit Display")
# Initialize Shor algorithm instance
shor = ShorAlgorithm()
# Show circuit immediately
shor.display_circuit_streamlit()

if __name__ == "__main__":
main()
```

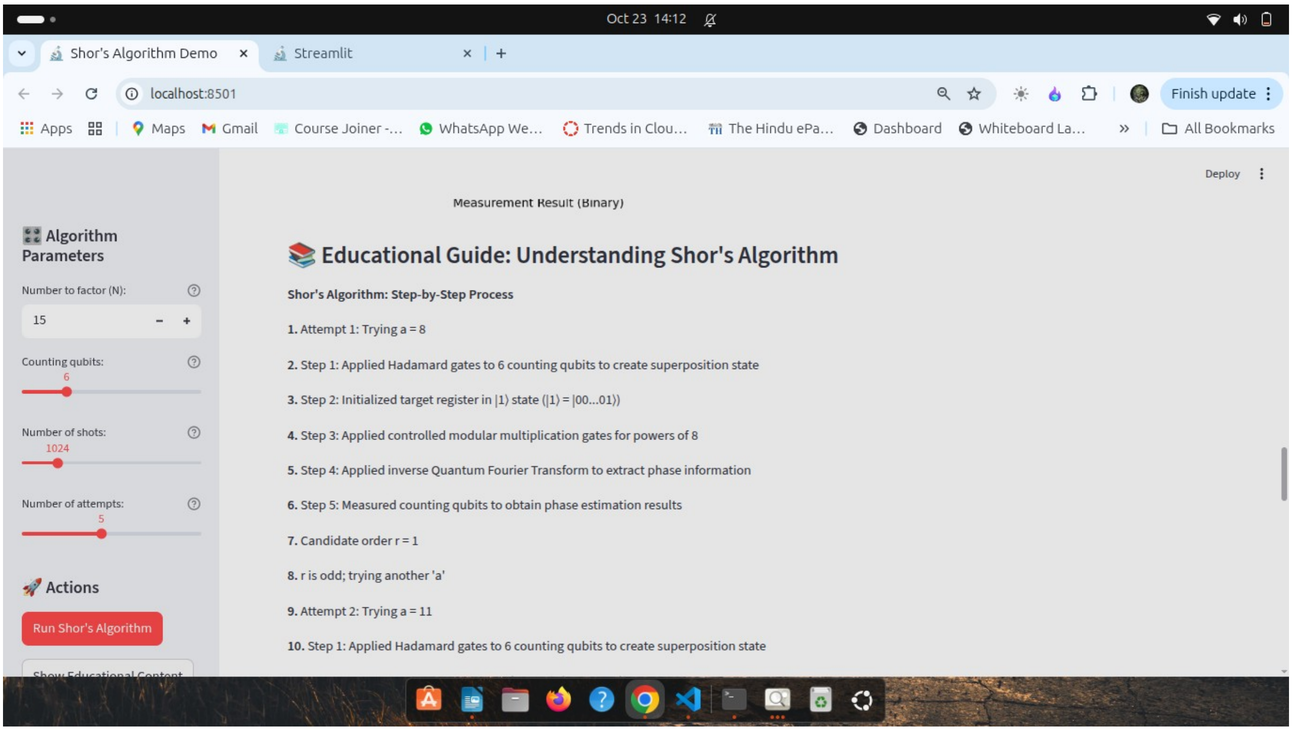
Output Snippet:

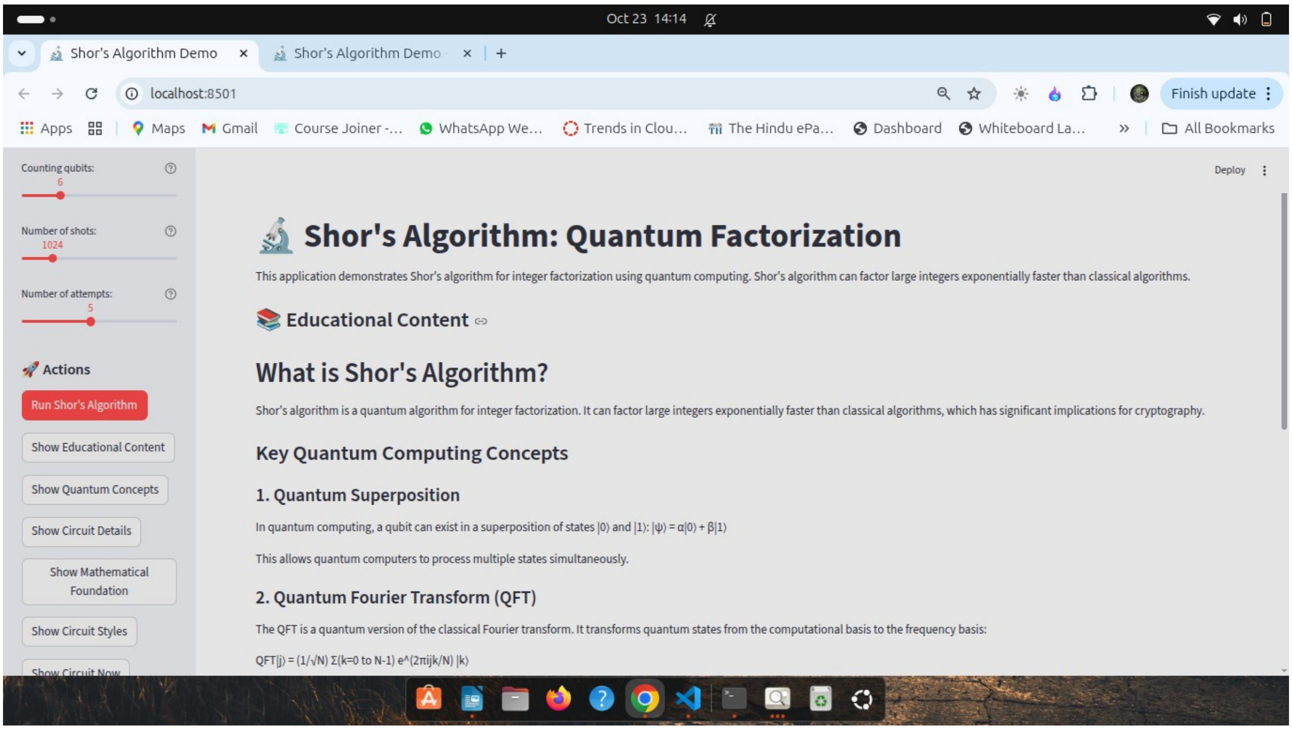


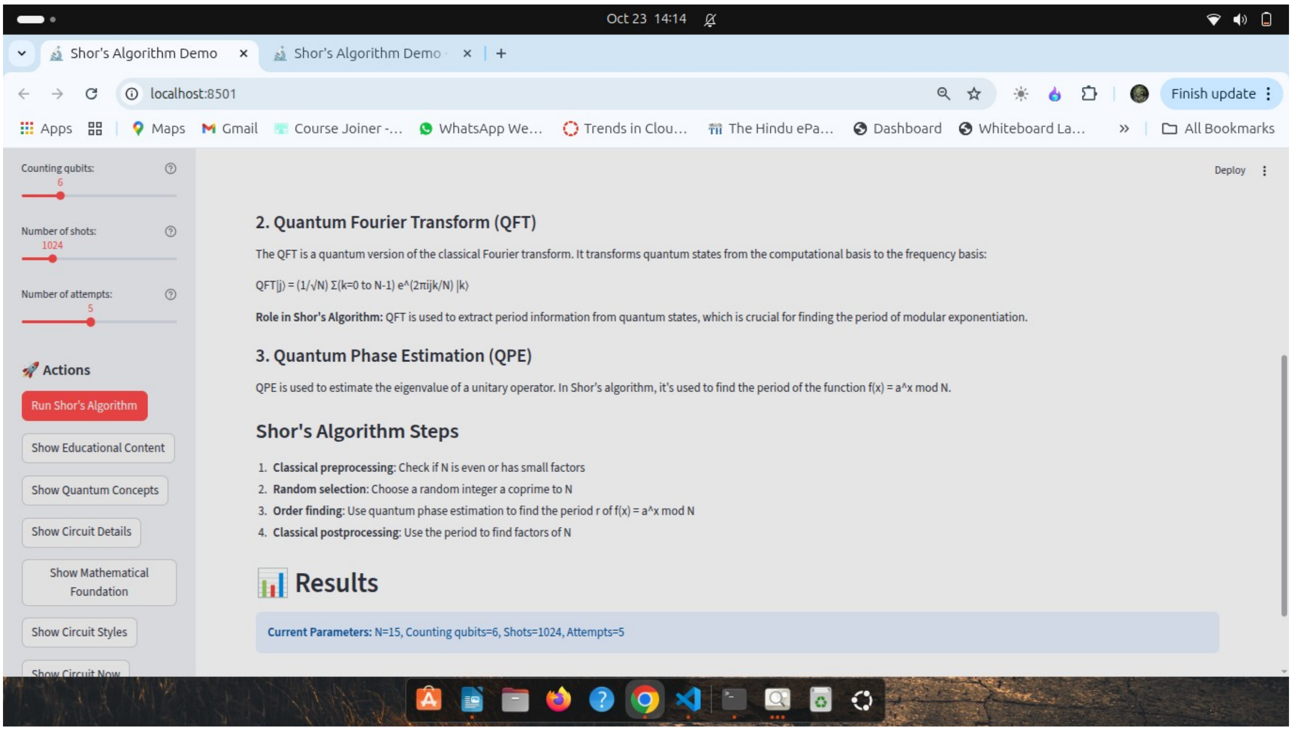


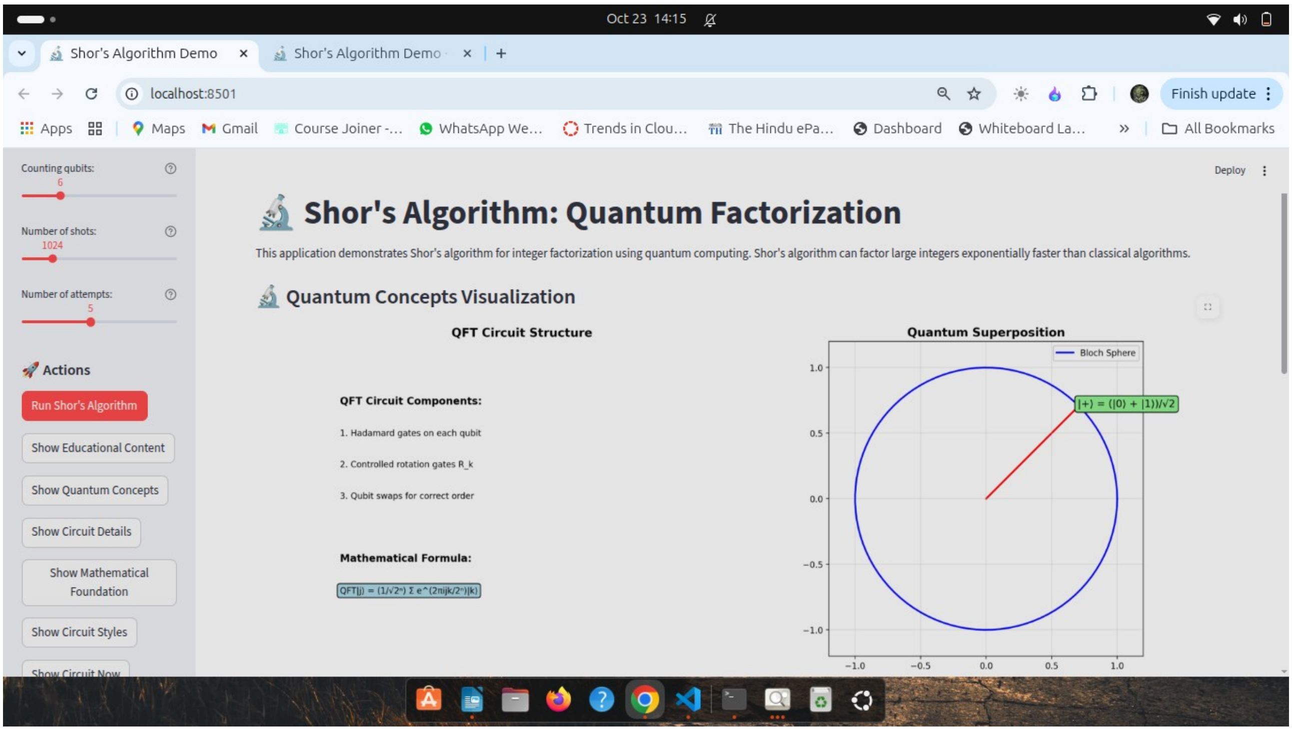

Oct 23 14:11
Shor's Algorithm Demo
Streamlit
localhost:8501
Finish update
Apps
Maps
Gmail
Course Joiner -...
WhatsApp We...
Trends in Clou...
The Hindu ePa...
Dashboard
Whiteboard La...
All Bookmarks
Deploy
Algorithm Parameters
Number to factor (N):
15
Counting qubits:
6
Number of shots:
1024
Number of attempts:
5
Actions
Run Shor's Algorithm
q8
q9
c
6
0
1
2
3
4
5
Circuit Information:
Total Qubits: 10
Classical Bits: 6
Gate Count: 20
Circuit Depth: 9
Success! Found factors of 15: 3 × 5 = 15
Verification: 3 × 5 = 15

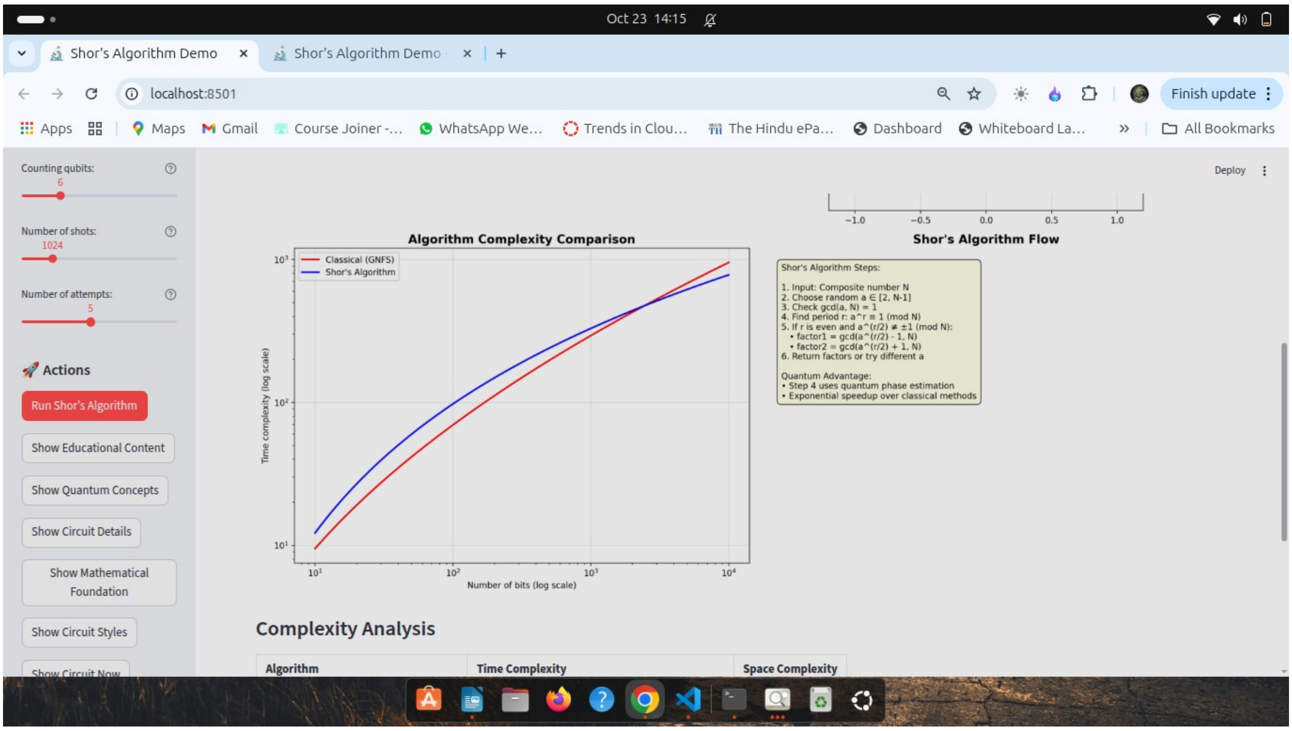

Oct 23 14:12
Shor's Algorithm Demo
Streamlit
localhost:8501
Finish update
Apps
Maps
Gmail
Course Joiner -...
WhatsApp We...
Trends in Clou...
The Hindu ePa...
Dashboard
Whiteboard La...
All Bookmarks
Deploy
Algorithm Parameters
Number to factor (N):
15
Counting qubits:
6
Number of shots:
1024
Number of attempts:
5
Actions
Run Shor's Algorithm
Probability Analysis
Top 10 Measurement Results
Count
500
400
300
200
100
0
000000
101011
010101
011011
101101
010011
001011
110101
100101
101001
Measurement Result (Binary)
Measurement Analysis
Total Shots: 1,024
Most Common Result: 000000
Count: 512
Probability: 0.500
Interpretation:
• The most frequent bitstring represents the likely phase estimate.
• A higher probability suggests stronger phase confidence.
• This phase contributes to determining the period of the function.
• The identified period is crucial for factorization in Shor's algorithm.
Educational Guide: Understanding Shor's Algorithm

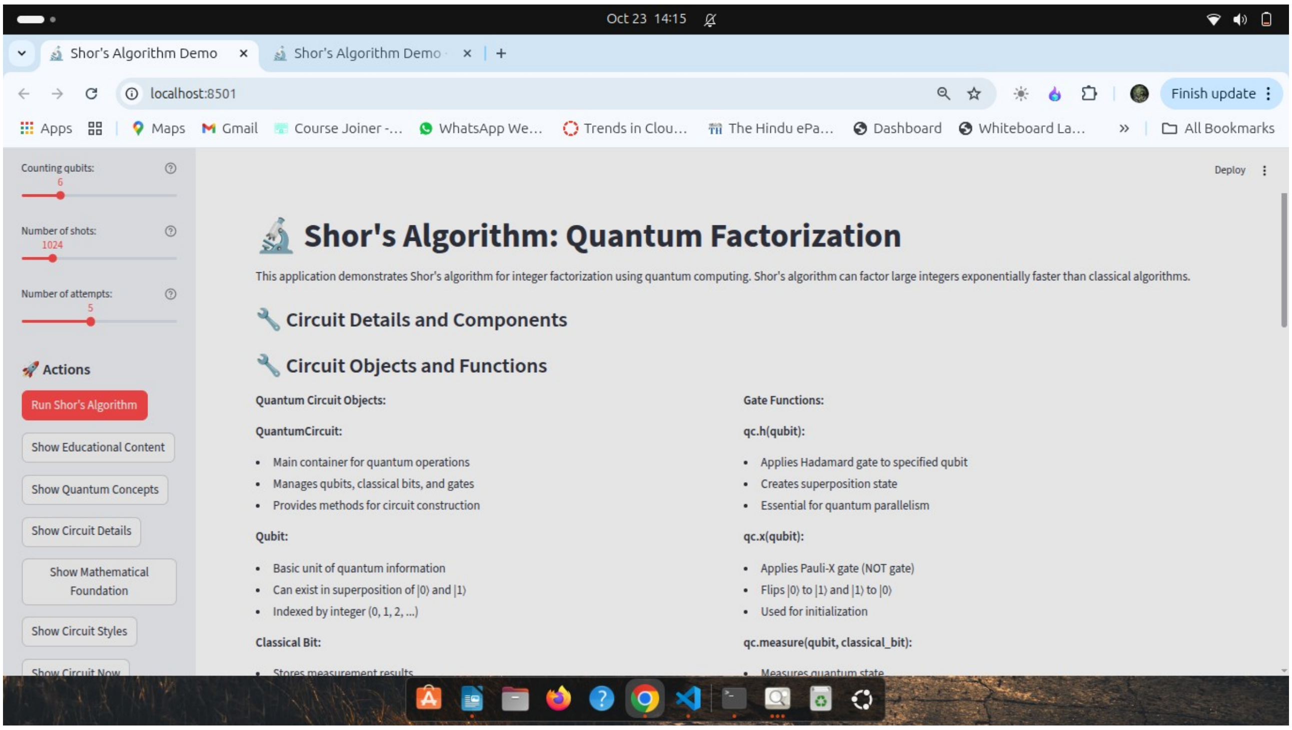

Oct 23 14:12
Shor's Algorithm Demo
Streamlit
localhost:8501
Finish update
Apps
Maps
Gmail
Course Joiner -...
WhatsApp We...
Trends in Clou...
The Hindu ePa...
Dashboard
Whiteboard La...
All Bookmarks
Deploy
Algorithm Parameters
Number to factor (N):
15
Counting qubits:
6
Number of shots:
1024
Number of attempts:
5
Actions
Run Shor's Algorithm
Measurement Result (Binary)
Educational Guide: Understanding Shor's Algorithm
Shor's Algorithm: Step-by-Step Process
1. Attempt 1: Trying a = 8
2. Step 1: Applied Hadamard gates to 6 counting qubits to create superposition state
3. Step 2: Initialized target register in |1⟩ state (|1⟩ = |00...01⟩)
4. Step 3: Applied controlled modular multiplication gates for powers of 8
5. Step 4: Applied inverse Quantum Fourier Transform to extract phase information
6. Step 5: Measured counting qubits to obtain phase estimation results
7. Candidate order r = 1
8. r is odd; trying another 'a'
9. Attempt 2: Trying a = 11
10. Step 1: Applied Hadamard gates to 6 counting qubits to create superposition state

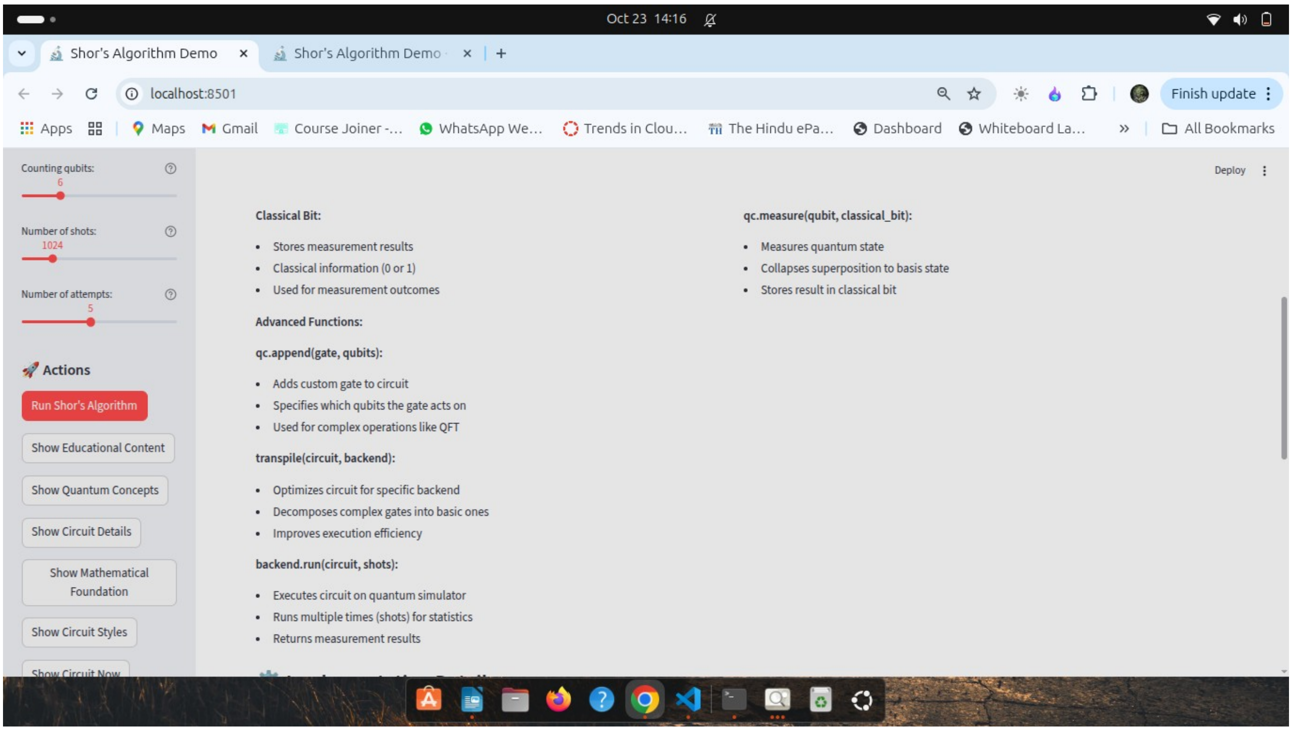

Oct 23 14:14
Shor's Algorithm Demo
Shor's Algorithm Demo
localhost:8501
Finish update
Apps
Maps
Gmail
Course Joiner -...
WhatsApp We...
Trends in Clou...
The Hindu ePa...
Dashboard
Whiteboard La...
All Bookmarks
Deploy
Counting qubits:
6
Number of shots:
1024
Number of attempts:
5
Actions
Run Shor's Algorithm
Show Educational Content
Show Quantum Concepts
Show Circuit Details
Show Mathematical Foundation
Show Circuit Styles
Shor's Algorithm: Quantum Factorization
This application demonstrates Shor's algorithm for integer factorization using quantum computing. Shor's algorithm can factor large integers exponentially faster than classical algorithms.
Educational Content
What is Shor's Algorithm?
Shor's algorithm is a quantum algorithm for integer factorization. It can factor large integers exponentially faster than classical algorithms, which has significant implications for cryptography.
Key Quantum Computing Concepts
1. Quantum Superposition
In quantum computing, a qubit can exist in a superposition of states |0⟩ and |1⟩: |ψ⟩ = α|0⟩ + β|1⟩
This allows quantum computers to process multiple states simultaneously.
2. Quantum Fourier Transform (QFT)
The QFT is a quantum version of the classical Fourier transform. It transforms quantum states from the computational basis to the frequency basis:
QFT|j⟩ = (1/√N) Σ(k=0 to N-1) e^(2πijk/N) |k⟩

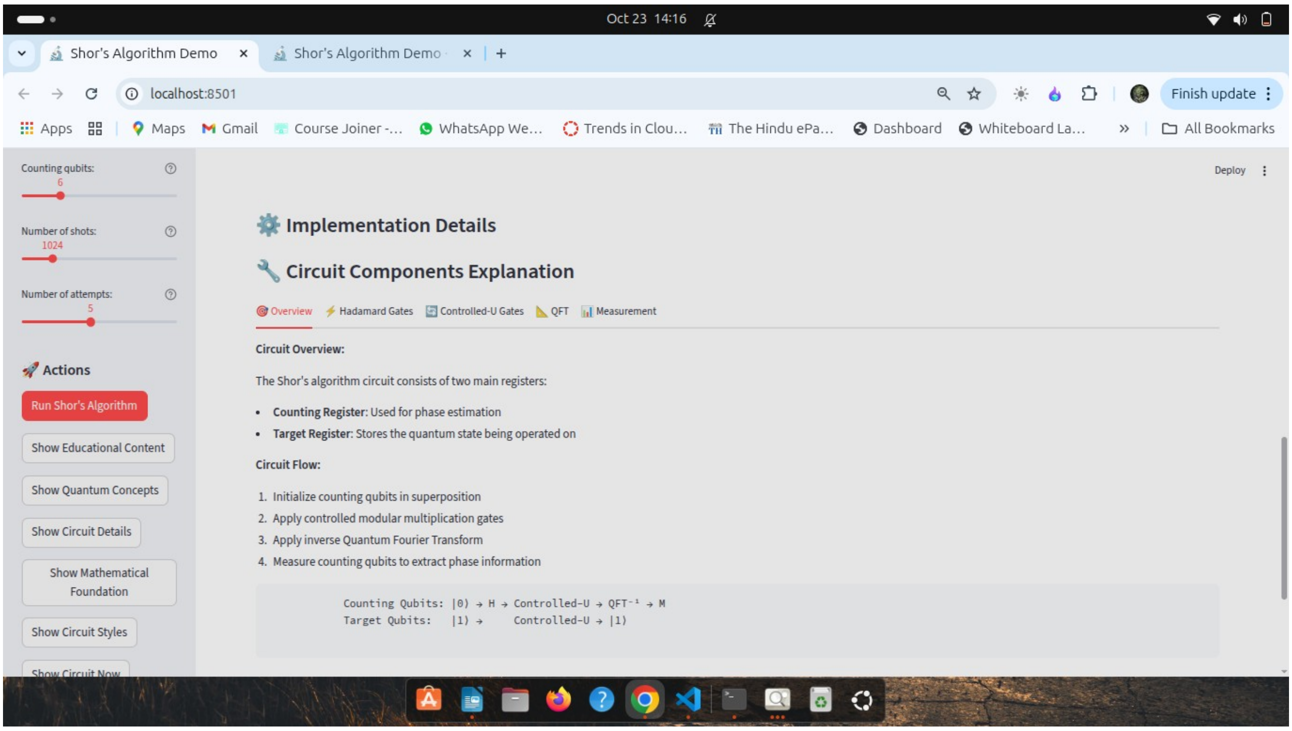

Oct 23 14:14
Shor's Algorithm Demo
localhost:8501
Counting qubits:
6
Number of shots:
1024
Number of attempts:
5
🚀 Actions
Run Shor's Algorithm
Show Educational Content
Show Quantum Concepts
Show Circuit Details
Show Mathematical Foundation
Show Circuit Styles
Deploy
2. Quantum Fourier Transform (QFT)
The QFT is a quantum version of the classical Fourier transform. It transforms quantum states from the computational basis to the frequency basis:
QFT|j⟩ = (1/√N) Σ(k=0 to N-1) e^(2πijk/N) |k⟩
Role in Shor's Algorithm: QFT is used to extract period information from quantum states, which is crucial for finding the period of modular exponentiation.
3. Quantum Phase Estimation (QPE)
QPE is used to estimate the eigenvalue of a unitary operator. In Shor's algorithm, it's used to find the period of the function f(x) = a^x mod N.
Shor's Algorithm Steps
1. Classical preprocessing: Check if N is even or has small factors
2. Random selection: Choose a random integer a coprime to N
3. Order finding: Use quantum phase estimation to find the period r of f(x) = a^x mod N
4. Classical postprocessing: Use the period to find factors of N
Results
Current Parameters: N=15, Counting qubits=6, Shots=1024, Attempts=5

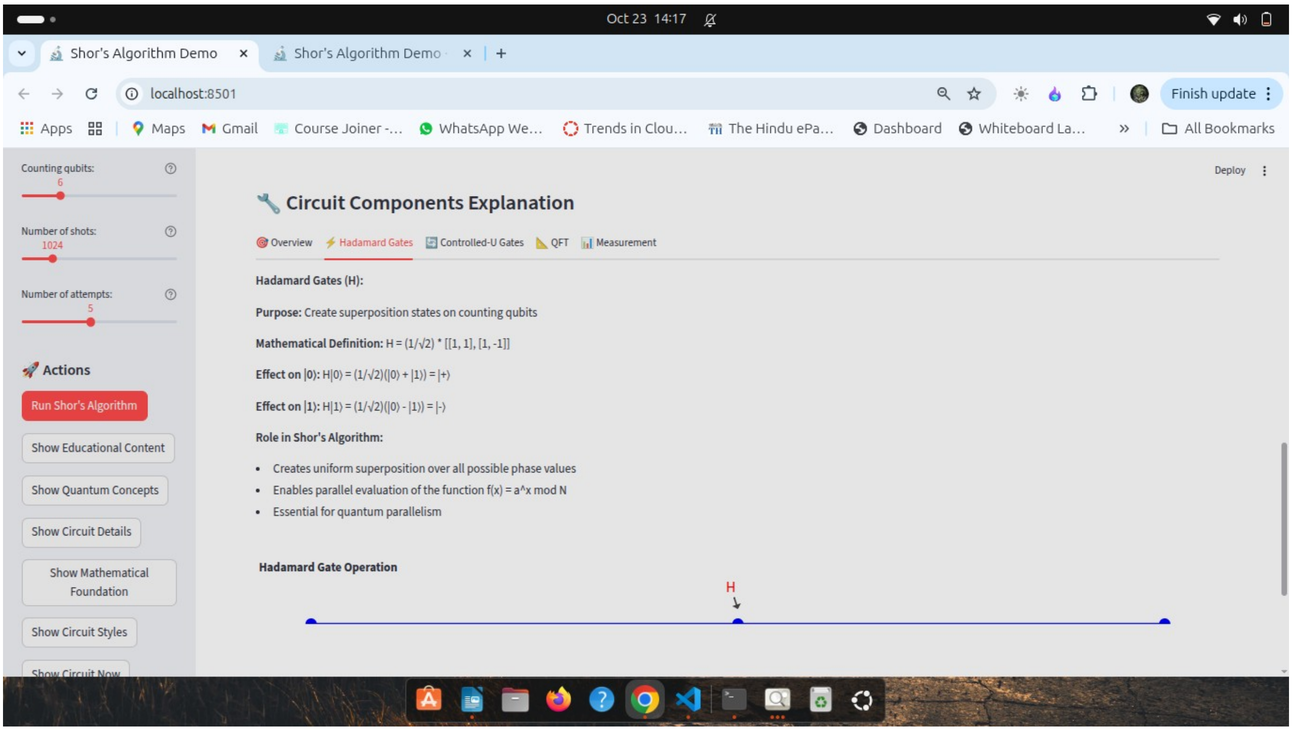

Oct 23 14:15
Shor's Algorithm Demo
localhost:8501
Counting qubits:
6
Number of shots:
1024
Number of attempts:
5
🚀 Actions
Run Shor's Algorithm
Show Educational Content
Show Quantum Concepts
Show Circuit Details
Show Mathematical Foundation
Show Circuit Styles
Deploy
Shor's Algorithm: Quantum Factorization
This application demonstrates Shor's algorithm for integer factorization using quantum computing. Shor's algorithm can factor large integers exponentially faster than classical algorithms.
Quantum Concepts Visualization
QFT Circuit Structure
QFT Circuit Components:
1. Hadamard gates on each qubit
2. Controlled rotation gates R_k
3. Qubit swaps for correct order
Mathematical Formula:
QFT|j⟩ = (1/√2ⁿ) Σ e^(2πijk/2ⁿ)|k⟩
Quantum Superposition
Bloch Sphere
|+⟩ = (|0⟩ + |1⟩)/√2

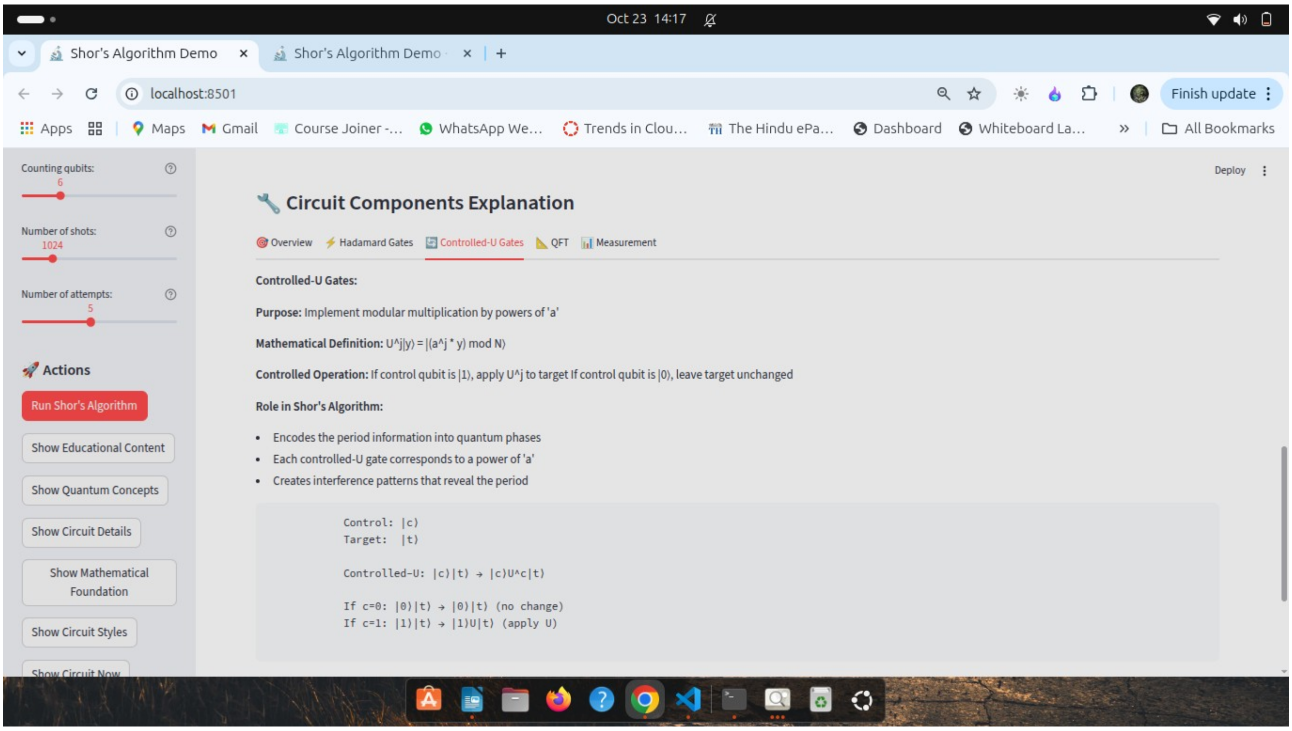

Oct 23 14:15
Shor's Algorithm Demo
Shor's Algorithm Demo
localhost:8501
Finish update
Apps
Maps
Gmail
Course Joiner -...
WhatsApp We...
Trends in Clou...
The Hindu ePa...
Dashboard
Whiteboard La...
All Bookmarks
Counting qubits:
6
Number of shots:
1024
Number of attempts:
5
Actions
Run Shor's Algorithm
Show Educational Content
Show Quantum Concepts
Show Circuit Details
Show Mathematical Foundation
Show Circuit Styles
Show Circuit Now
Deploy
Algorithm Complexity Comparison
Classical (GNFS)
Shor's Algorithm
Time complexity (log scale)
Number of bits (log scale)
Shor's Algorithm Flow
Shor's Algorithm Steps:
1. Input: Composite number N
2. Choose random a ∈ [2, N-1]
3. Check gcd(a, N) = 1
4. Find period r: a^r ≡ 1 (mod N)
5. If r is even and a^(r/2) ≢ ±1 (mod N):
• factor1 = gcd(a^(r/2) - 1, N)
• factor2 = gcd(a^(r/2) + 1, N)
6. Return factors or try different a
Quantum Advantage:
• Step 4 uses quantum phase estimation
• Exponential speedup over classical methods
Complexity Analysis
Algorithm
Time Complexity
Space Complexity

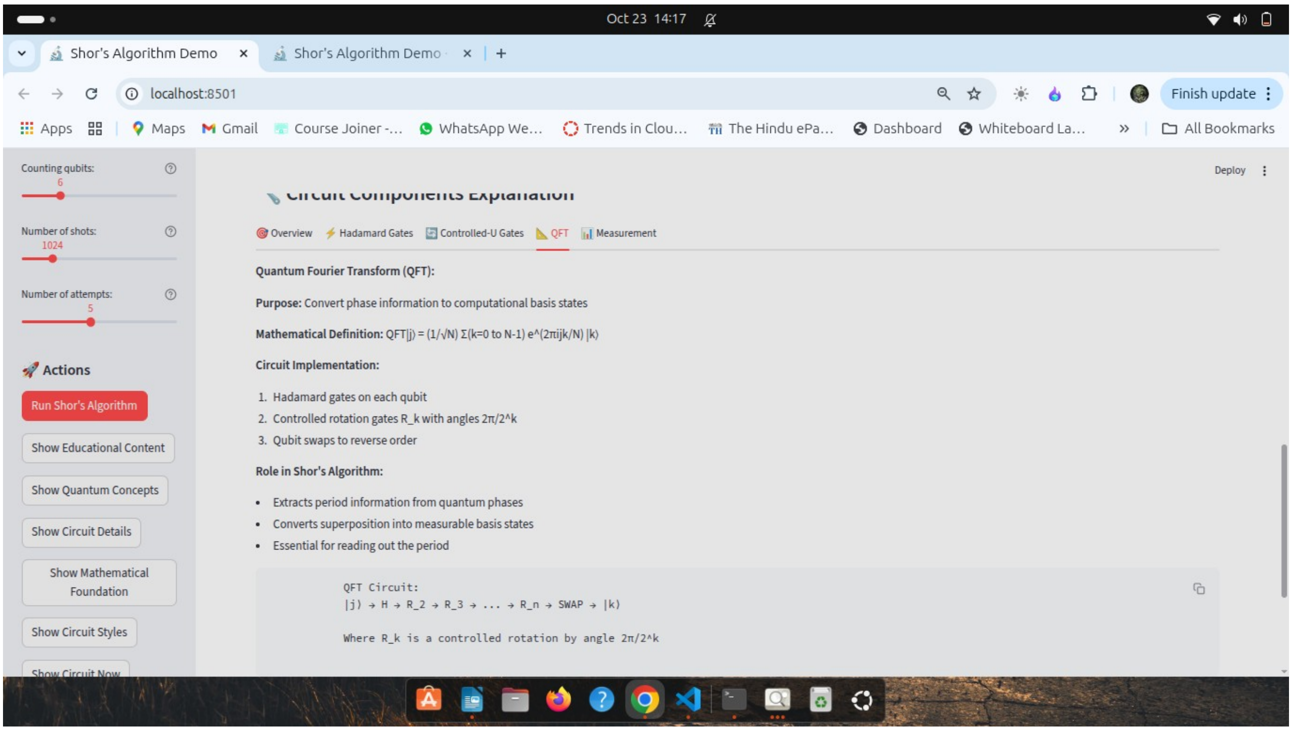

Oct 23 14:15
Shor's Algorithm Demo
Shor's Algorithm Demo
localhost:8501
Finish update
Apps
Maps
Gmail
Course Joiner -...
WhatsApp We...
Trends in Clou...
The Hindu ePa...
Dashboard
Whiteboard La...
All Bookmarks
Counting qubits:
6
Number of shots:
1024
Number of attempts:
5
Actions
Run Shor's Algorithm
Show Educational Content
Show Quantum Concepts
Show Circuit Details
Show Mathematical Foundation
Show Circuit Styles
Show Circuit Now
Deploy
Shor's Algorithm: Quantum Factorization
This application demonstrates Shor's algorithm for integer factorization using quantum computing. Shor's algorithm can factor large integers exponentially faster than classical algorithms.
Circuit Details and Components
Circuit Objects and Functions
Quantum Circuit Objects:
QuantumCircuit:
• Main container for quantum operations
• Manages qubits, classical bits, and gates
• Provides methods for circuit construction
Qubit:
• Basic unit of quantum information
• Can exist in superposition of |0⟩ and |1⟩
• Indexed by integer (0, 1, 2, ...)
Classical Bit:
• Stores measurement results
Gate Functions:
qc.h(qubit):
• Applies Hadamard gate to specified qubit
• Creates superposition state
• Essential for quantum parallelism
qc.x(qubit):
• Applies Pauli-X gate (NOT gate)
• Flips |0⟩ to |1⟩ and |1⟩ to |0⟩
• Used for initialization
qc.measure(qubit, classical_bit):
• Measures quantum state

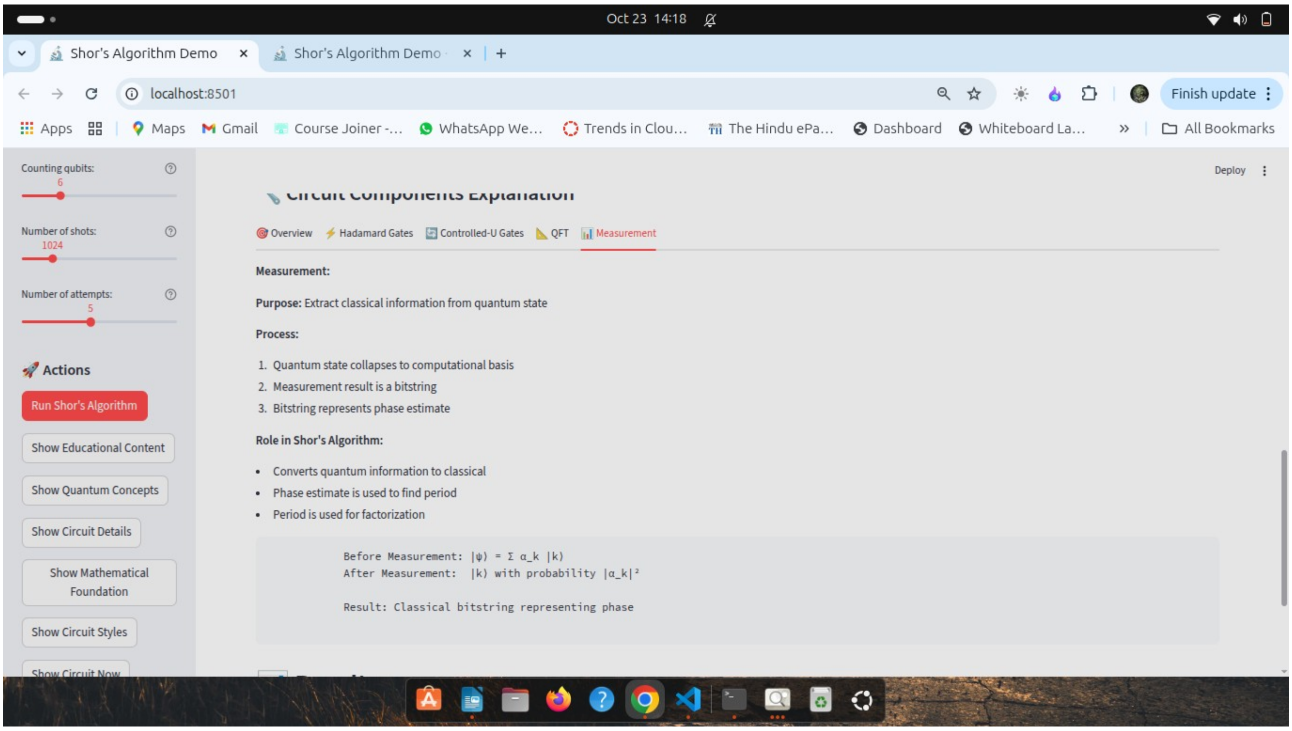

Oct 23 14:16
Shor's Algorithm Demo
Shor's Algorithm Demo
localhost:8501
Finish update
Apps
Maps
Gmail
Course Joiner -...
WhatsApp We...
Trends in Clou...
The Hindu ePa...
Dashboard
Whiteboard La...
All Bookmarks
Counting qubits:
6
Number of shots:
1024
Number of attempts:
5
Actions
Run Shor's Algorithm
Show Educational Content
Show Quantum Concepts
Show Circuit Details
Show Mathematical Foundation
Show Circuit Styles
Deploy
Classical Bit:
Stores measurement results
Classical information (0 or 1)
Used for measurement outcomes
qc.measure(qubit, classical_bit):
Measures quantum state
Collapses superposition to basis state
Stores result in classical bit
Advanced Functions:
qc.append(gate, qubits):
Adds custom gate to circuit
Specifies which qubits the gate acts on
Used for complex operations like QFT
transpile(circuit, backend):
Optimizes circuit for specific backend
Decomposes complex gates into basic ones
Improves execution efficiency
backend.run(circuit, shots):
Executes circuit on quantum simulator
Runs multiple times (shots) for statistics
Returns measurement results

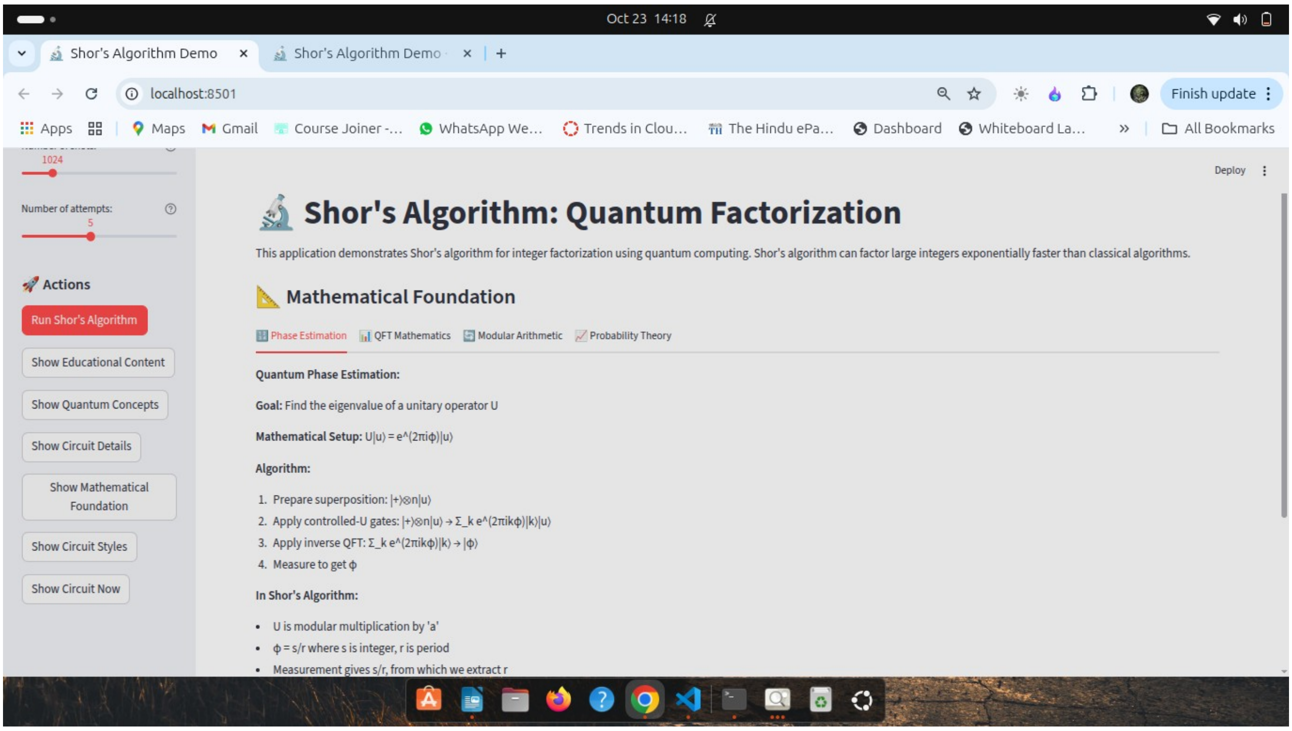

Oct 23 14:16
Shor's Algorithm Demo
Shor's Algorithm Demo
localhost:8501
Finish update
Apps
Maps
Gmail
Course Joiner -...
WhatsApp We...
Trends in Clou...
The Hindu ePa...
Dashboard
Whiteboard La...
All Bookmarks
Counting qubits:
6
Number of shots:
1024
Number of attempts:
5
Actions
Run Shor's Algorithm
Show Educational Content
Show Quantum Concepts
Show Circuit Details
Show Mathematical Foundation
Show Circuit Styles
Deploy
Implementation Details
Circuit Components Explanation
Overview
Hadamard Gates
Controlled-U Gates
QFT
Measurement
Circuit Overview:
The Shor's algorithm circuit consists of two main registers:
Counting Register: Used for phase estimation
Target Register: Stores the quantum state being operated on
Circuit Flow:
1. Initialize counting qubits in superposition
2. Apply controlled modular multiplication gates
3. Apply inverse Quantum Fourier Transform
4. Measure counting qubits to extract phase information
Counting Qubits: |0⟩ → H → Controlled-U → QFT⁻¹ → M
Target Qubits: |1⟩ → Controlled-U → |1⟩


Oct 23 14:17
Shor's Algorithm Demo
Shor's Algorithm Demo
localhost:8501
Finish update
Apps
Maps
Gmail
Course Joiner -...
WhatsApp We...
Trends in Clou...
The Hindu ePa...
Dashboard
Whiteboard La...
All Bookmarks
Counting qubits:
6
Number of shots:
1024
Number of attempts:
5
Actions
Run Shor's Algorithm
Show Educational Content
Show Quantum Concepts
Show Circuit Details
Show Mathematical Foundation
Show Circuit Styles
Show Circuit Now
Deploy
Circuit Components Explanation
Overview
Hadamard Gates
Controlled-U Gates
QFT
Measurement
Hadamard Gates (H):
Purpose: Create superposition states on counting qubits
Mathematical Definition: H = (1/√2) * [[1, 1], [1, -1]]
Effect on |0⟩: H|0⟩ = (1/√2)(|0⟩ + |1⟩) = |+⟩
Effect on |1⟩: H|1⟩ = (1/√2)(|0⟩ - |1⟩) = |-⟩
Role in Shor's Algorithm:
Creates uniform superposition over all possible phase values
Enables parallel evaluation of the function f(x) = a^x mod N
Essential for quantum parallelism
Hadamard Gate Operation
H

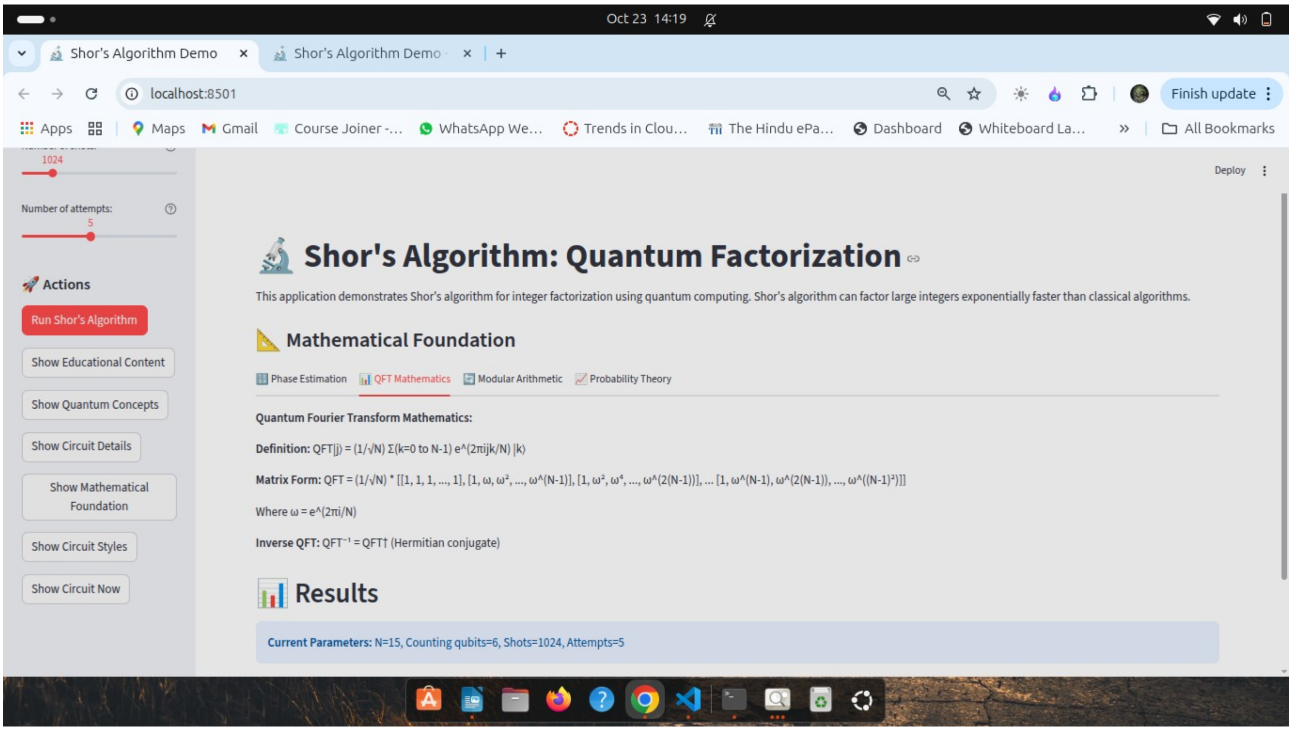

Oct 23 14:17
Shor's Algorithm Demo
Shor's Algorithm Demo
localhost:8501
Finish update
Apps
Maps
Gmail
Course Joiner -...
WhatsApp We...
Trends in Clou...
The Hindu ePa...
Dashboard
Whiteboard La...
All Bookmarks
Counting qubits:
6
Number of shots:
1024
Number of attempts:
5
Actions
Run Shor's Algorithm
Show Educational Content
Show Quantum Concepts
Show Circuit Details
Show Mathematical Foundation
Show Circuit Styles
Show Circuit Now
Deploy
Circuit Components Explanation
Overview
Hadamard Gates
Controlled-U Gates
QFT
Measurement
Controlled-U Gates:
Purpose: Implement modular multiplication by powers of 'a'
Mathematical Definition: U^j|y⟩ = |(a^j * y) mod N⟩
Controlled Operation: If control qubit is |1⟩, apply U^j to target If control qubit is |0⟩, leave target unchanged
Role in Shor's Algorithm:
Encodes the period information into quantum phases
Each controlled-U gate corresponds to a power of 'a'
Creates interference patterns that reveal the period
Control: |c⟩
Target: |t⟩
Controlled-U: |c⟩|t⟩ → |c⟩U^c|t⟩
If c=0: |0⟩|t⟩ → |0⟩|t⟩ (no change)
If c=1: |1⟩|t⟩ → |1⟩U|t⟩ (apply U)


Oct 23 14:17
Shor's Algorithm Demo
localhost:8501
Counting qubits:
6
Number of shots:
1024
Number of attempts:
5
Actions
Run Shor's Algorithm
Show Educational Content
Show Quantum Concepts
Show Circuit Details
Show Mathematical Foundation
Show Circuit Styles
Deploy
Circuit Components Explanation
Overview
Hadamard Gates
Controlled-U Gates
QFT
Measurement
Quantum Fourier Transform (QFT):
Purpose: Convert phase information to computational basis states
Mathematical Definition: QFT|j⟩ = (1/√N) Σ(k=0 to N-1) e^(2πijk/N) |k⟩
Circuit Implementation:
1. Hadamard gates on each qubit
2. Controlled rotation gates R_k with angles 2π/2^k
3. Qubit swaps to reverse order
Role in Shor's Algorithm:
• Extracts period information from quantum phases
• Converts superposition into measurable basis states
• Essential for reading out the period
QFT Circuit:
|j⟩ → H → R_2 → R_3 → ... → R_n → SWAP → |k⟩
Where R_k is a controlled rotation by angle 2π/2^k

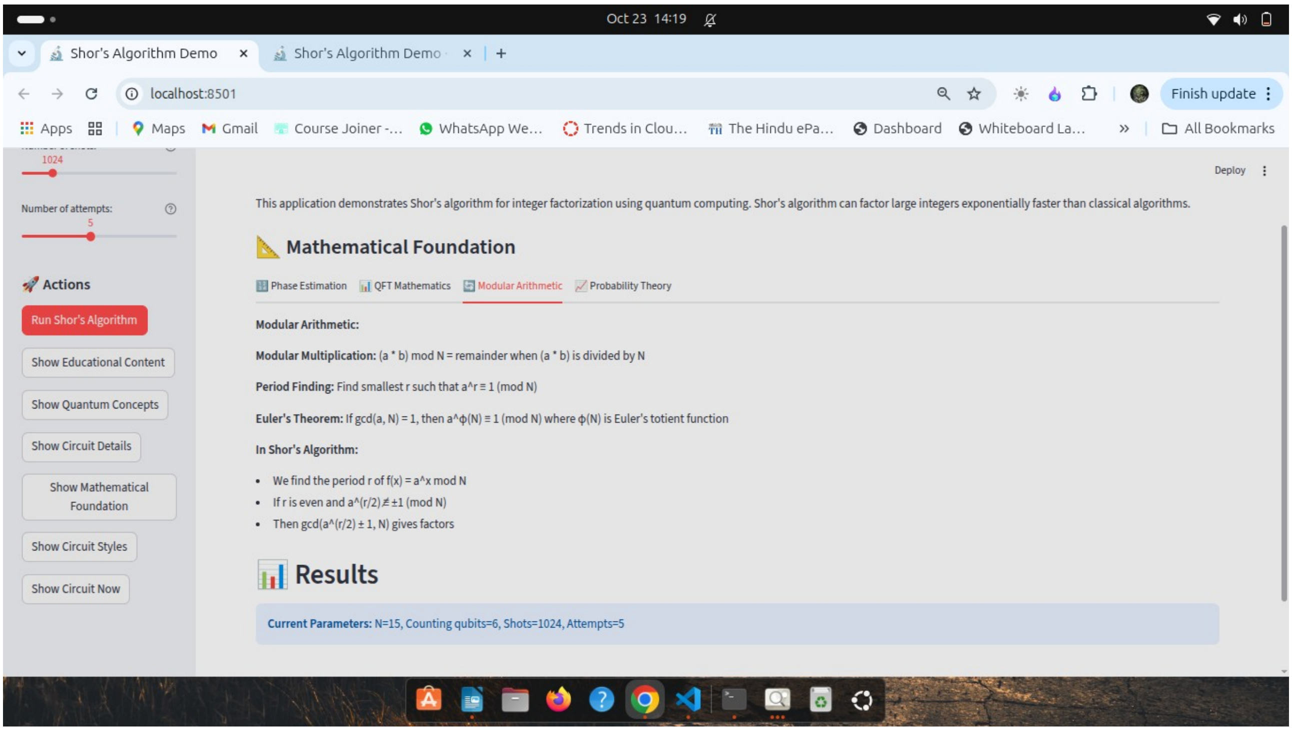

Oct 23 14:18
Shor's Algorithm Demo
localhost:8501
Counting qubits:
6
Number of shots:
1024
Number of attempts:
5
Actions
Run Shor's Algorithm
Show Educational Content
Show Quantum Concepts
Show Circuit Details
Show Mathematical Foundation
Show Circuit Styles
Deploy
Circuit Components Explanation
Overview
Hadamard Gates
Controlled-U Gates
QFT
Measurement
Measurement:
Purpose: Extract classical information from quantum state
Process:
1. Quantum state collapses to computational basis
2. Measurement result is a bitstring
3. Bitstring represents phase estimate
Role in Shor's Algorithm:
• Converts quantum information to classical
• Phase estimate is used to find period
• Period is used for factorization
Before Measurement: |ψ⟩ = Σ α_k |k⟩
After Measurement: |k⟩ with probability |α_k|²
Result: Classical bitstring representing phase

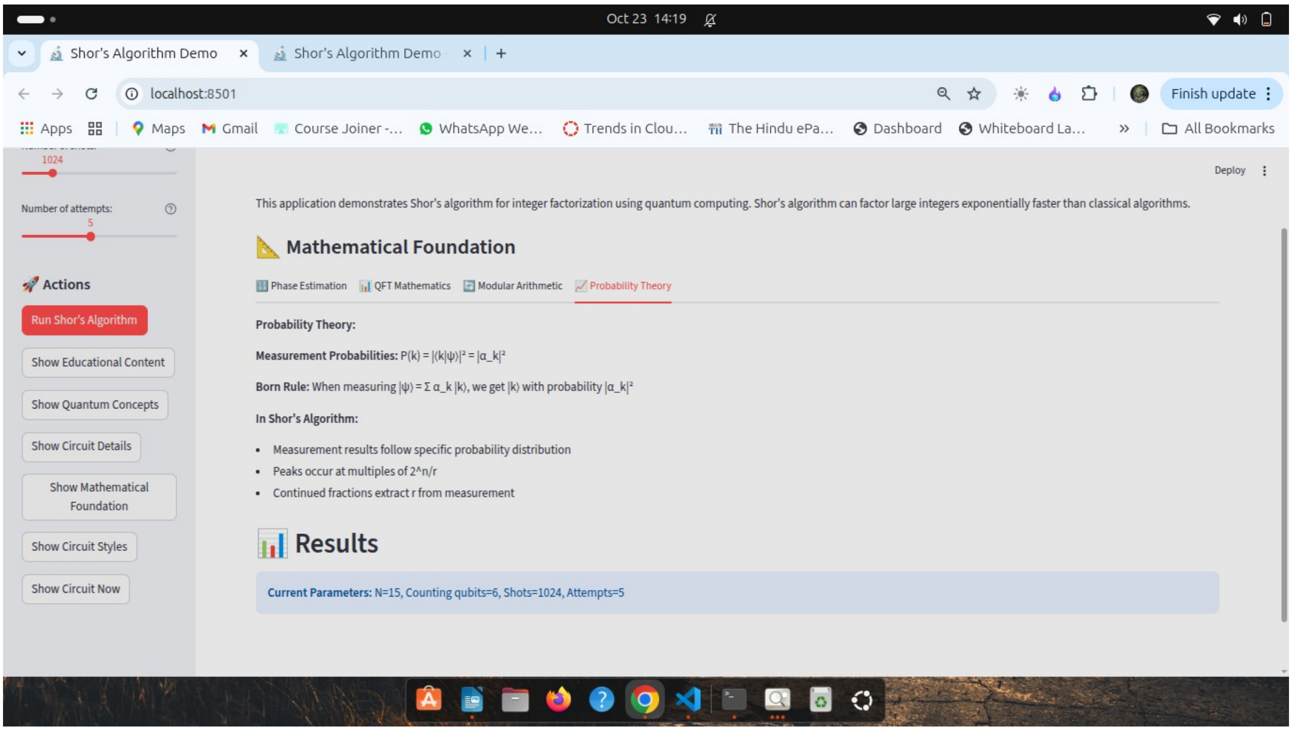


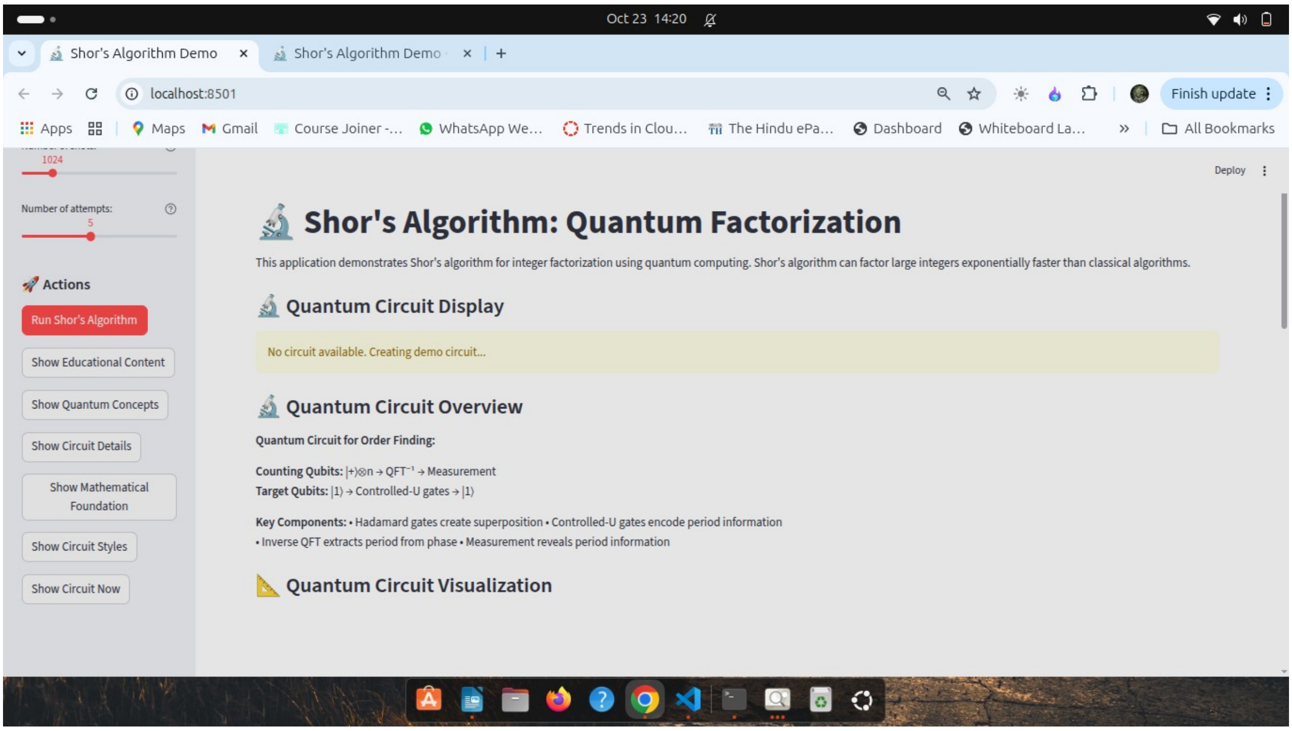
Oct 23 14:18
Shor's Algorithm Demo
localhost:8501
Finish update
1024
Number of attempts:
5
Actions
Run Shor's Algorithm
Show Educational Content
Show Quantum Concepts
Show Circuit Details
Show Mathematical Foundation
Show Circuit Styles
Show Circuit Now
Deploy
Shor's Algorithm: Quantum Factorization
This application demonstrates Shor's algorithm for integer factorization using quantum computing. Shor's algorithm can factor large integers exponentially faster than classical algorithms.
Mathematical Foundation
Phase Estimation
QFT Mathematics
Modular Arithmetic
Probability Theory
Quantum Phase Estimation:
Goal: Find the eigenvalue of a unitary operator U
Mathematical Setup: U|u⟩ = e^(2πiϕ)|u⟩
Algorithm:
1. Prepare superposition: |+⟩⊗n|u⟩
2. Apply controlled-U gates: |+⟩⊗n|u⟩ → Σ_k e^(2πikϕ)|k⟩|u⟩
3. Apply inverse QFT: Σ_k e^(2πikϕ)|k⟩ → |ϕ⟩
4. Measure to get ϕ
In Shor's Algorithm:
U is modular multiplication by 'a'
ϕ = s/r where s is integer, r is period
Measurement gives s/r, from which we extract r



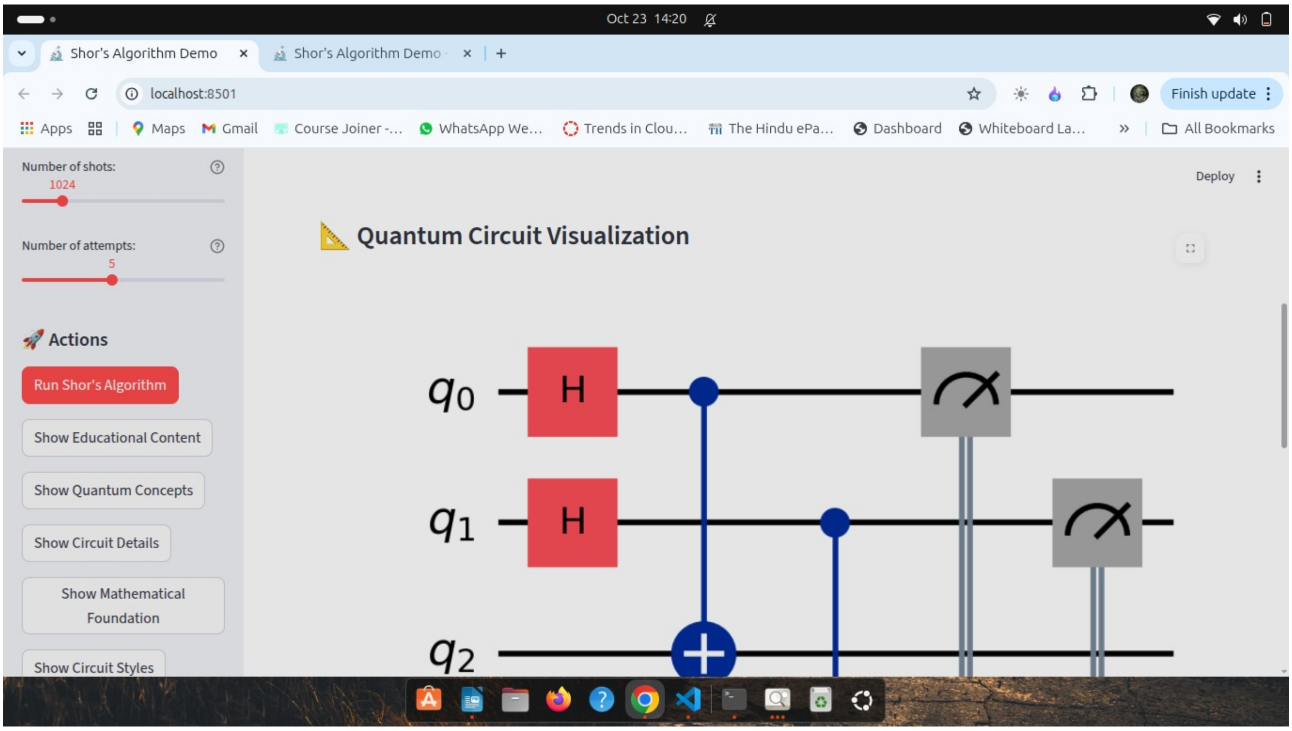
Oct 23 14:19
Shor's Algorithm Demo
localhost:8501
Finish update
1024
Number of attempts:
5
Actions
Run Shor's Algorithm
Show Educational Content
Show Quantum Concepts
Show Circuit Details
Show Mathematical Foundation
Show Circuit Styles
Show Circuit Now
Deploy
Shor's Algorithm: Quantum Factorization
This application demonstrates Shor's algorithm for integer factorization using quantum computing. Shor's algorithm can factor large integers exponentially faster than classical algorithms.
Mathematical Foundation
Phase Estimation
QFT Mathematics
Modular Arithmetic
Probability Theory
Quantum Fourier Transform Mathematics:
Definition: QFT|j⟩ = (1/√N) Σ(k=0 to N-1) e^(2πijk/N) |k⟩
Matrix Form: QFT = (1/√N) * [[1, 1, 1, ..., 1], [1, ω, ω², ..., ω^(N-1)], [1, ω², ω⁴, ..., ω^(2(N-1))], ... [1, ω^(N-1), ω^(2(N-1)), ..., ω^((N-1)²)]]
Where ω = e^(2πi/N)
Inverse QFT: QFT⁻¹ = QFT† (Hermitian conjugate)
Results
Current Parameters: N=15, Counting qubits=6, Shots=1024, Attempts=5


Oct 23 14:19
Shor's Algorithm Demo
Shor's Algorithm Demo
localhost:8501
Finish update
Apps
Maps
Gmail
Course Joiner -...
WhatsApp We...
Trends in Clou...
The Hindu ePa...
Dashboard
Whiteboard La...
All Bookmarks
1024
Number of attempts:
5
Actions
Run Shor's Algorithm
Show Educational Content
Show Quantum Concepts
Show Circuit Details
Show Mathematical Foundation
Show Circuit Styles
Show Circuit Now
Deploy
This application demonstrates Shor's algorithm for integer factorization using quantum computing. Shor's algorithm can factor large integers exponentially faster than classical algorithms.
Mathematical Foundation
Phase Estimation
QFT Mathematics
Modular Arithmetic
Probability Theory
Modular Arithmetic:
Modular Multiplication: (a * b) mod N = remainder when (a * b) is divided by N
Period Finding: Find smallest r such that a^r ≡ 1 (mod N)
Euler's Theorem: If gcd(a, N) = 1, then a^φ(N) ≡ 1 (mod N) where φ(N) is Euler's totient function
In Shor's Algorithm:
We find the period r of f(x) = a^x mod N
If r is even and a^(r/2) ≢ ±1 (mod N)
Then gcd(a^(r/2) ± 1, N) gives factors
Results
Current Parameters: N=15, Counting qubits=6, Shots=1024, Attempts=5

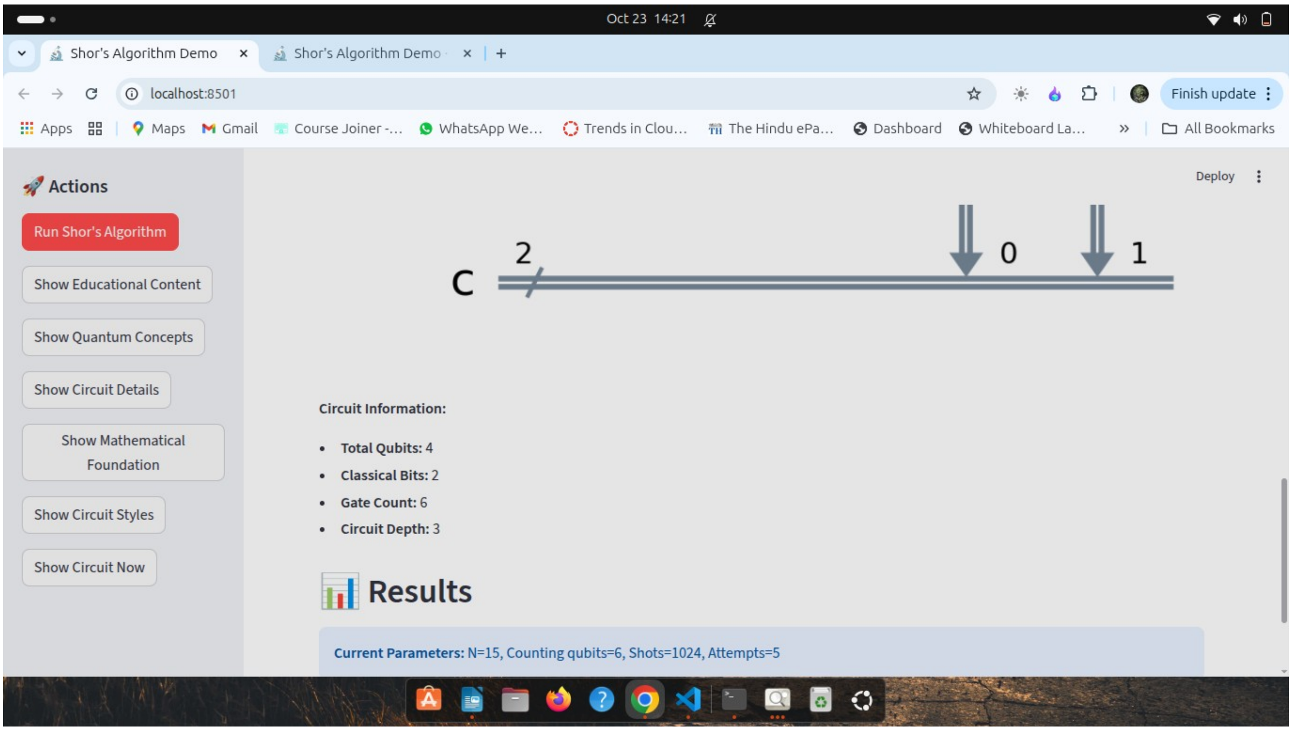

Oct 23 14:19
Shor's Algorithm Demo
Shor's Algorithm Demo
localhost:8501
Finish update
Apps
Maps
Gmail
Course Joiner -...
WhatsApp We...
Trends in Clou...
The Hindu ePa...
Dashboard
Whiteboard La...
All Bookmarks
1024
Number of attempts:
5
Actions
Run Shor's Algorithm
Show Educational Content
Show Quantum Concepts
Show Circuit Details
Show Mathematical Foundation
Show Circuit Styles
Show Circuit Now
Deploy
This application demonstrates Shor's algorithm for integer factorization using quantum computing. Shor's algorithm can factor large integers exponentially faster than classical algorithms.
Mathematical Foundation
Phase Estimation
QFT Mathematics
Modular Arithmetic
Probability Theory
Probability Theory:
Measurement Probabilities: P(k) = |⟨k|ψ⟩|² = |α_k|²
Born Rule: When measuring |ψ⟩ = Σ α_k |k⟩, we get |k⟩ with probability |α_k|²
In Shor's Algorithm:
Measurement results follow specific probability distribution
Peaks occur at multiples of 2^n/r
Continued fractions extract r from measurement
Results
Current Parameters: N=15, Counting qubits=6, Shots=1024, Attempts=5



Oct 23 14:20
Shor's Algorithm Demo
localhost:8501
Finish update
Apps
Maps
Gmail
Course Joiner -...
WhatsApp We...
Trends in Clou...
The Hindu ePa...
Dashboard
Whiteboard La...
All Bookmarks
1024
Number of attempts:
5
Actions
Run Shor's Algorithm
Show Educational Content
Show Quantum Concepts
Show Circuit Details
Show Mathematical Foundation
Show Circuit Styles
Show Circuit Now
Deploy
Shor's Algorithm: Quantum Factorization
This application demonstrates Shor's algorithm for integer factorization using quantum computing. Shor's algorithm can factor large integers exponentially faster than classical algorithms.
Quantum Circuit Display
No circuit available. Creating demo circuit...
Quantum Circuit Overview
Quantum Circuit for Order Finding:
Counting Qubits: |+⟩⊗n → QFT⁻¹ → Measurement
Target Qubits: |1⟩ → Controlled-U gates → |1⟩
Key Components: • Hadamard gates create superposition • Controlled-U gates encode period information
• Inverse QFT extracts period from phase • Measurement reveals period information
Quantum Circuit Visualization



Oct 23 14:20
Shor's Algorithm Demo
localhost:8501
Finish update
Apps
Maps
Gmail
Course Joiner -...
WhatsApp We...
Trends in Clou...
The Hindu ePa...
Dashboard
Whiteboard La...
All Bookmarks
Number of shots:
1024
Number of attempts:
5
Actions
Run Shor's Algorithm
Show Educational Content
Show Quantum Concepts
Show Circuit Details
Show Mathematical Foundation
Show Circuit Styles
Deploy
Quantum Circuit Visualization
q0
q1
q2
H
H


Oct 23 14:21
Shor's Algorithm Demo
Shor's Algorithm Demo
localhost:8501
Finish update
Apps
Maps
Gmail
Course Joiner -...
WhatsApp We...
Trends in Clou...
The Hindu ePa...
Dashboard
Whiteboard La...
All Bookmarks
Actions
Run Shor's Algorithm
Show Educational Content
Show Quantum Concepts
Show Circuit Details
Show Mathematical Foundation
Show Circuit Styles
Show Circuit Now
Deploy
C
2
0
1
Circuit Information:
Total Qubits: 4
Classical Bits: 2
Gate Count: 6
Circuit Depth: 3
Results
Current Parameters: N=15, Counting qubits=6, Shots=1024, Attempts=5