Controller of Certifying Authorities
Ministry of Electronics & Information Technology
Government of India

IEEE COMPUTER SOCIETY
Bangalore Chapter

IAS IEEE INDUSTRY APPLICATIONS SOCIETY
Linking Research to Practice
Bangalore Chapter

6th INTERNATIONAL CONFERENCE ON

PUBLIC KEY INFRASTRUCTURE AND ITS APPLICATIONS

(PKIA 2025)

SEPTEMBER 3-4th, 2025

# CROSS PLATFORM BENCHMARKING OF SHOR'S ALGORITHM FOR QUANTUM CRYPTANALYSIS

**Dr. Noorul Hussain, Ms.Vanideve & Dr.Rajarajan**

**Intrust Innovation Labs, Chennai**

https://pkiindia.in
Social Media
/pkiindia

IEEE BANGALORE SECTION

## Introduction & Motivation

# The Quantum threat 2 Classical Cryptography

1

### The Cryptographic Foundation

Current encryption relies on the the computational difficulty of factoring large integers, safeguarding sensitive data globally. globally.

2

### Shor's Algorithm Emerges

Introduced in 1994, this quantum algorithm can factor integers in polynomial time, posing an existential threat to RSA and other widely used schemes.

3

### Our Research Goal

To evaluate current quantum computing frameworks for implementing Shor's algorithm, algorithm, assessing their performance, scalability, and practical readiness for real-world world application.

Background

# Understanding Shor's Algorithm

Shor's algorithm offers an exponential speedup over classical factoring methods by leveraging quantum mechanics. It's a hybrid approach combining classical pre and post-processing with a quantum core.

01

## Classical Pre-processing

Select a random integer 'a' co-prime to 'N' (the number to be factored). This sets up the quantum computation.

02

## Quantum Period-Finding

The core quantum step utilizes superposition and the Quantum Fourier Fourier Transform (QFT) to efficiently find find the period 'r' of the function f(x)=a^x f(x)=a^x mod N.

03

## Classical Post-processing

With 'r' determined, factors of of 'N' are calculated using the the Euclidean algorithm: gcd(a^(r/2) ± 1, N).

**The quantum period-finding subroutine is where the algorithm gains its unparalleled efficiency.**

The Tools

# The Contenders: Seven Quantum Frameworks Evaluated

We benchmarked seven leading quantum software development platforms, each with unique strengths and applications.

- **Qiskit:** IBM's framework, strong integration with IBM Q IBM Q hardware.
- **Cirq:** Google's framework, designed for near-term quantum computers.
- **PennyLane:** For hybrid quantum-classical machine learning, supports multiple backends.
- **Qibo:** High-performance framework focused on fast circuit simulation.
- **QuTiP:** Focuses on simulating quantum dynamics dynamics using symbolic operators.
- **ProjectQ:** A high-level quantum compiler framework.
- **Tequila:** An abstraction layer for variational quantum algorithms.

Each framework offers distinct advantages, from hardware integration to simulation speed and high-level abstraction.

# Quantum Frameworks: A Diverse Ecosystem

| | | |
|---|---|---|
| **Qiskit** | **IBM's open-source quantum SDK** | **Gate-level fidelity, hardware integration** |
| **Cirq** | **Google's quantum programming framework** | **Fine-grained control, hardware-agnostic** |
| **PennyLane** | **QML framework for hybrid computing** | **Symbolic differentiation, hybrid scalability** |
| **QuTip** | **Open-source library for quantum optics** | **Numerical simulation, large system support** |
| **ProjectQ** | **Quantum computing framework by ETH Zurich** | **Python-based, extensible backend** |
| **Tequila** | **Modular quantum chemistry framework** | **Variatioal algorithms, high-level abstraction** |
| **Qibo** | **Framework for quantum simulation** | **Hardware-accelerated, custom backends** |

## Our Approach

# Our Experimental Methodology

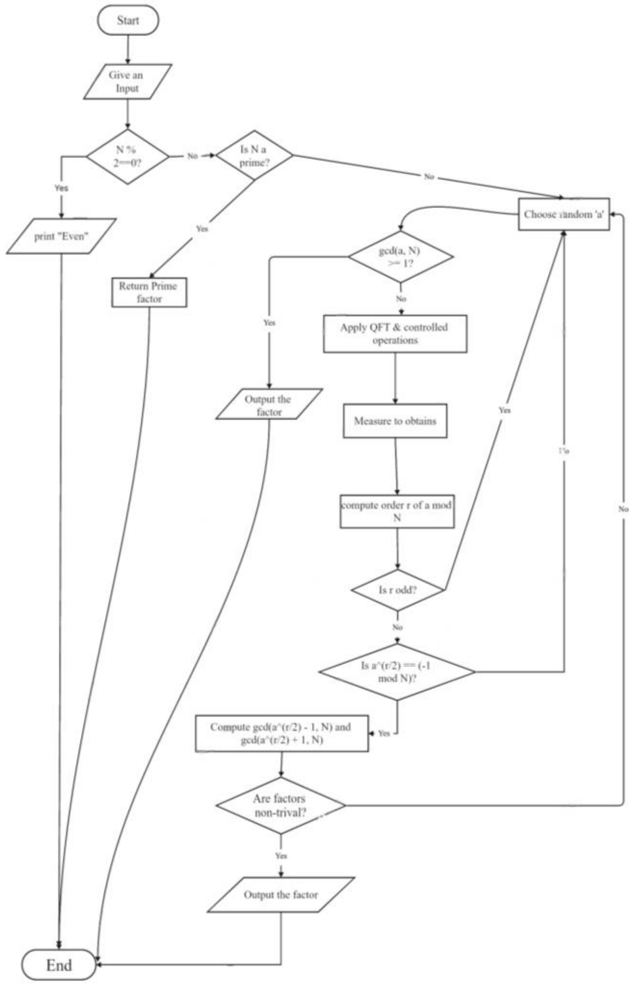
Our objective was to thoroughly assess each framework's usability, accuracy, scalability, and hardware support through a rigorous implementation of Shor’s algorithm.

### Input Data

We used a curated dataset of composite integers, ranging from small (4-bit, N=15) to considerably larger (47-bit, 15-digit numbers), to push the limits of each framework.

### Implementation Flow

A consistent workflow was maintained across all platforms: from initial input validation, through the quantum order-finding subroutine, to final classical factor extraction.




Shor's Algorithm Implementation Flowchart

# Results Overview

## Maximum Integer Factored

| Framework | Max Digits Factored | Qubits Used | Time Taken | Qubit Type |
|---|---|---|---|---|
| Qiskit | 4 (N = 1234) | 23 | 38s | Logical Qubits (Sim) |
| Cirq | 4 | 23 | 2m 17s | Logical Qubits (Sim) |
| PennyLane | 15 | 60 | 3m 05s | Logical Qubits (Hybrid) |
| Qibo | 6 | 18 | 1m 48s | Logical Qubits (Sim) |
| QuTiP | 9 | 54 | 28s | Symbolic Operators |
| ProjectQ | 5 | 28 | 35s | Logical Qubits (Sim) |
| Tequila | 6 | 18 | 5m 24s | Logical Qubits (Sim) |

```
import cirq
import numpy as np
from math import gcd
from fractions import Fraction

# Input values
N = 15         # number to factor
a = 7          # choose a value coprime with N
n_count = 4    # number of counting qubits
# 1. Initialize qubits
qubits = [cirq.LineQubit(i) for i in range(n_count)]
circuit = cirq.Circuit()

# 2. Create uniform superposition on counting register
circuit.append([cirq.H(q) for q in qubits])
# Simplified placeholder for modular exponentiation
# (In full Shor, we'd apply controlled-U gates here)
circuit.append(cirq.X(cirq.LineQubit(n_count)))  # dummy step

# 3. Measure counting register
circuit.append(cirq.measure(*qubits, key='m'))

# 4. Simulate the circuit
sim = cirq.Simulator()
result = sim.run(circuit, repetitions=1)
measured = result.measurements['m'][0]

# 5. Classical post-processing: estimate the period r
phase = int("".join(str(b) for b in measured), 2) / (2 ** n_count)
r = Fraction(phase).limit_denominator(N).denominator

# 6. Use r to compute factors of N
if r % 2 == 0 and pow(a, r // 2, N) != N - 1:
    p = gcd(pow(a, r // 2) - 1, N)
    q = gcd(pow(a, r // 2) + 1, N)
    print(f"Factors of {N}: {p}, {q}")
else:
    print("Try again with different 'a'")
```

```
Factors of 15: 3, 5
```

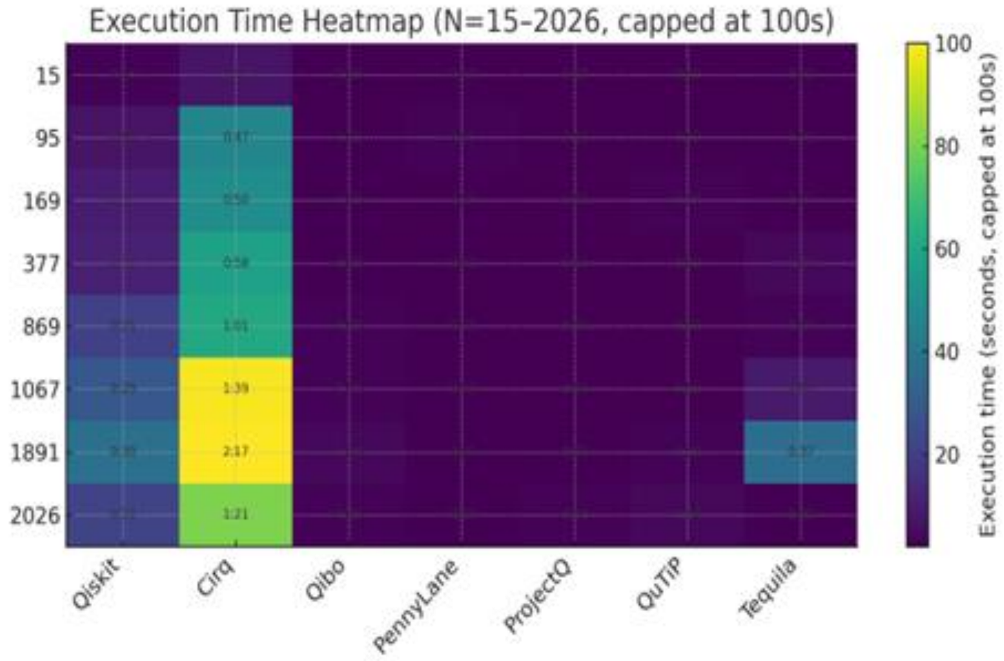
**Key Insight:** Frameworks leveraging symbolic/hybrid methods (PennyLane, QuTiP) significantly outperformed pure gate-level simulators (Qiskit, Cirq) in handling larger inputs.

**Performance Analysis**

# Runtime Performance Across Frameworks

Execution times diverged rapidly as the input integer size (N) increased, highlighting key performance differences.

- **Fast & Consistent:**
  **PennyLane** and **Qibo** maintained high efficiency, exhibiting consistently low execution times (dark purple).
- **Moderate Scaling:**
  **Qiskit** showed a modest, manageable increase in runtime as N grew.
- **Poor Scaling:**
  **Cirq** was consistently the slowest, with execution time increasing increasing substantially with N (bright yellow).



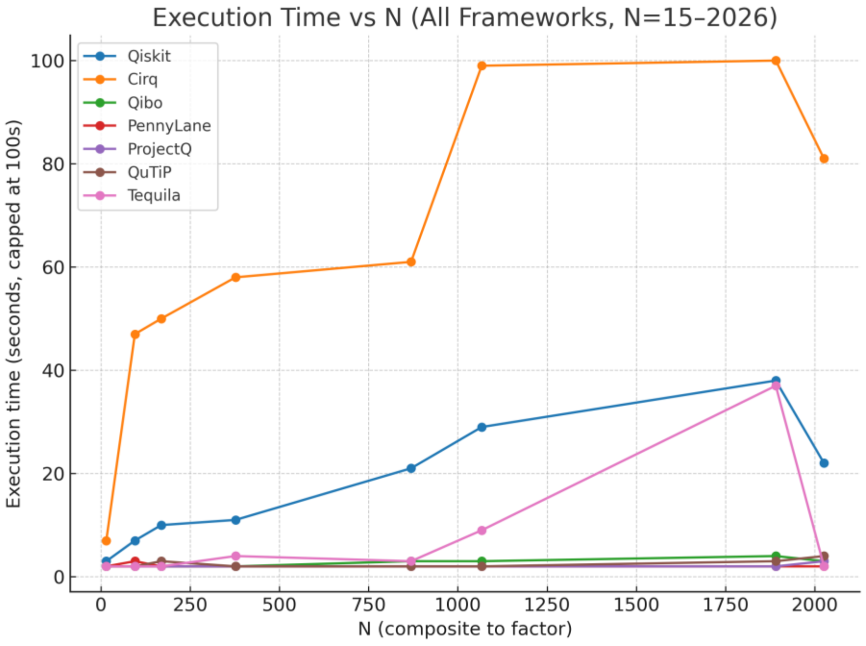

Execution Time Heatmap

## Insights

# Fidelity vs. Scalability Trade-Off

This analysis clearly visualizes the performance gap, highlighting a fundamental trade-off in quantum framework design.

- The **steep curve for Cirq** demonstrates how its high-fidelity, gate-level simulation quickly becomes computationally expensive.
- The **flat lines for PennyLane and Qibo** illustrate the significant significant scalability benefits of their symbolic and optimized optimized simulation approaches.




**Conclusion: A fundamental trade-off exists between the precision of precision of gate-level simulation and the scalability of symbolic/hybrid models.**

Standardized Comparison

# Introducing the Quantum Efficiency Index (QEI)

To provide a standardized and comprehensive comparison across frameworks, we propose a novel metric: the **Quantum Efficiency Index (QEI).**

$$QEI = \frac{\text{Qubits Used} \times \text{Circuit Depth}}{\text{Success Rate} \times \text{Max Input Bit Length}}$$

## What it Measures

The QEI balances an algorithm's accuracy and scale against its resource cost (qubits and circuit depth).

## Its Purpose

Enables researchers to assess frameworks based on overall **efficiency**, not just raw speed or the largest number factored.

# Quantum Efficiency Index (QEI): A Unified Metric

To enable a normalized, cross-platform comparison, we developed the Quantum Efficiency Index (QEI).

$$QEI = \frac{Success\ Rate \times Max\ Bit\ Lenght}{Qubits \times Circuit\ Depth}$$

This formula balances successful factorization rates, bit length, qubit consumption, and circuit complexity.

**0.0039**
**PennyLane**

Reflecting its superior balance of performance and resource usage.

**0.0005**
**Average Gate-level QEI**

Indicating lower efficiency for more complex problems.

**Impact: QEI provides a valuable quantitative measure for evaluating the practical efficiency of quantum algorithms across diverse software and hardware architectures.**

## Key Takeaways

# Discussion of Key Findings & Future Directions

## Key Findings

- **Symbolic Frameworks Excel at Scale:** PennyLane and QuTiP are ideal are ideal for large-scale algorithmic exploration where simulation simulation speed is paramount, bypassing the exponential cost of full cost of full state-vector simulation.
- **Gate-Level for Hardware Readiness:** Qiskit and Cirq, while less scalable in simulation, are crucial for hardware-focused research due to their precise circuit modeling, noise analysis, and direct QPU Integrations.
- **Major Bottlenecks Remain:** Shor's algorithm scaling is still limited limited by modular exponentiation, escalating qubit requirements, requirements, and exponential simulation time.

## Our Future Workflow

- All experiments used idealized, noise-free simulators. Real-world quantum hardware introduces decoherence and errors.
- **Proposed Hybrid Workflow:**

**Stage 1:** Symbolic Simulation (PennyLane, QuTiP) for QuTiP) for rapid logic validation.

**Stage 2:** Gate-Level Simulation (Qiskit, Cirq) for for detailed circuit analysis and hardware resource resource estimation.

**Stage 3:** Hardware Testing (Cloud Backends) for for small test cases on real QPUs in noisy environments.

# THANK YOU