Performance Evaluation of Shor Algorithm on Simulated Quantum Hardware with Circuit Level Analysis

[1]Thamaraimanalan T, [2]Anandakumar Haldorai, [3]Arulmurugan Ramu and [4]Mariyappan K
[1]Department of Electronics and Communications Engineering, Sri Eshwar College of Engineering, Coimbatore, Tamil Nadu, India.
[2]Department of Computer Science and Engineering, Sri Eshwar College of Engineering, Coimbatore, Tamil Nadu, India.
[3]Department of Computational Science and Software Engineering, K. Zhubanov University, Kazakhstan, Asia.
[4]Department of Computer Science and Engineering, Chennai Institute of Technology, Chennai, Tamil Nadu, India.
[1]t.thamaraimanalan@gmail.com, [2]anandakumar.psgtech@gmail.com, [3]arulmr@gmail.com [4]mariwithgold@gmail.com.

Correspondence should be addressed to Thamaraimanalan T : t.thamaraimanalan@gmail.com

Article Info
Journal of Machine and Computing (https://anapub.co.ke/journals/jmc/jmc.html)
Doi : https://doi.org/10.53759/7669/jmc202505152
Received 28 March 2025; Revised from 10 May 2025; Accepted 19 June 2025.
Available online 05 July 2025.

Abstract – Shor's algorithm stands as a breakthrough in quantum computing due to its potential to factor large integers exponentially quicker than classical algorithms. However, implementing and evaluating this algorithm on real quantum computer hardware remains exciting due to qubit limitations, gate noise, and hardware constraints. This research presents a comprehensive performance evaluation of Shor's algorithm using simulated quantum backends provided by Qiskit. A flexible and generic implementation is proposed, allowing dynamic input of integers to be factored, with randomized co-prime selection and automated circuit generation. The algorithm is tested on various semiprime numbers, such as 15, 21, and 35, using IBM's Aer simulator. A major contribution of this work is the circuit-level analysis conducted both before and after transpilation. Metrics such as gate counts, circuit depth, and simulator runtime are extracted to assess scalability and resource requirements. High-resolution plots of the pre-transpiled circuits are saved to visualize algorithmic complexity, while post-transpilation metrics inform future quantum hardware feasibility. The output measurement distributions are analyzed to estimate periodicity and derive correct factors. The proposed implementation is compared with existing fixed-instance Shor demonstrations to highlight its flexibility and extensibility. Experimental results show consistent success in factor retrieval and provide valuable insight into circuit growth and complexity under realistic constraints. This analysis lays the groundwork for future adaptation to NISQ hardware and contributes to understanding Shor's algorithm from both computational and architectural perspectives.

Keywords – Shor's Algorithm, Qiskit Simulation, Quantum Circuit Analysis, Quantum-Classical Comparison.

I. INTRODUCTION

Integer factorization is a fundamental problem in number theory that involves the breakdown of a composite number into a product of smaller numbers. Its significance extends beyond pure mathematics into practical domains such as cryptography, algorithmic complexity, and secure communications [1]. The integer factorization becomes computationally prohibitive as the size of the number increases, especially when the number in question is a semiprime an integer composed of exactly two large prime factors. Historically, several classical algorithms have been developed to tackle this problem, each improving upon its predecessors in terms of efficiency and scalability. The most straightforward method, trial division, involves dividing the target number by successive integers to test for divisibility. Although effective for small numbers, it quickly becomes impractical for large inputs due to its exponential time complexity [2].

More sophisticated algorithms such as Pollard's rho algorithm, Fermat's factorization, and the Elliptic Curve Method (ECM) introduce probabilistic and number-theoretic heuristics to improve efficiency. However, the most notable advancements in classical integer factorization are represented by the Quadratic Sieve (QS) and the General Number Field Sieve (GNFS). These algorithms utilize complex mathematical structures, such as algebraic number fields, to factor large semiprimes in sub-exponential time. Despite its efficiency, GNFS still scales poorly for very large integers—typically those used in cryptographic key generation (e.g., 2048-bit RSA keys)—and remains infeasible without access to massive computational resources. This computational hardness underpins the security of widely used cryptographic protocols like RSA, DSA, and Diffie-Hellman key exchange, where the confidentiality and authenticity of encrypted communications rest on the infeasibility of factorizing large semiprimes [3].

In 1994, Peter Shor revolutionized the landscape of computational number theory and cryptography by suggesting a quantum algorithm capable of factoring large integers in polynomial time. Unlike classical methods, which scale sub-exponentially at best, Shor's algorithm operates in time complexity marking an exponential speedup over the fastest classical alternatives. The quantum subroutine utilizes quantum parallelism to evaluate many values of $f(x)$ simultaneously and employs the QFT to extract the periodicity embedded in the superposition of quantum states [4]. This period-finding process is exponentially faster than classical brute-force or sieving techniques, making Shor's algorithm the most powerful known application of quantum computing. The potential of Shor's algorithm to break RSA and other cryptosystems has led to widespread concern within the cybersecurity community. In anticipation of scalable quantum computers, researchers have begun developing post-quantum cryptographic algorithms, which are designed to be unaffected by quantum attacks. These include lattice-based, hash-based, and code-based cryptosystems, many of which are currently being evaluated by the National Institute of Standards and Technology (NIST) for standardization [5].

From a scientific perspective, Shor's algorithm continues to serve as a benchmark for quantum computational advantage. Experimental demonstrations on simulated and real quantum hardware, such as those offered by IBM, Google, and Rigetti, have successfully factored small semiprimes like 15 and 21. These implementations not only validate the algorithm's correctness but also provide crucial insights into circuit optimization, transpilation, noise modeling, and hybrid quantum-classical execution strategies. This research evaluates the performance and feasibility of Shor's algorithm using simulated quantum circuits in Qiskit, focusing on semiprime numbers (N=15, 21, and 35). By analyzing circuit depth, gate counts, and measurement distributions, we assess the algorithm's efficiency and scalability under realistic quantum computing constraints [6].

Despite its theoretical significance and potential to revolutionize cryptography, Shor's algorithm has several practical drawbacks that hinder its current applicability. The most critical limitation is the requirement for a fault-tolerant quantum computer with a large number of qubits. To factor an n-bit number, the algorithm requires the order of $O(n^3)$ quantum gates and roughly 2n qubits, depending on the implementation. Current quantum hardware lacks the necessary qubit count and coherence times to run such a complex algorithm effectively, particularly for numbers of cryptographic importance (e.g., 2048-bit RSA keys). Additionally, Shor's algorithm is highly sensitive to noise and decoherence, common in existing quantum systems [7, 8]. Even small errors in gate operations or qubit interactions can lead to incorrect results, requiring advanced quantum error correction schemes. Implementing such correction methods adds significant overhead, both in terms of required qubits and computational resources. Another challenge lies in the classical pre- and post-processing steps. Although quantum speedup is obtained in the modular exponentiation and period-finding stages, classical computations are still required for preparing the input and verifying the factors, potentially limiting speed gains in practice. Moreover, the algorithm is tailored for integer factorization and does not directly apply to other cryptographic primitives. While its implications are profound for RSA, it is less relevant for post-quantum cryptographic algorithms based on lattices or hash functions [8].

This research article is organized into four main sections for clarity and coherence. Section 2: Literature Review presents an in-depth analysis of classical and quantum approaches to integer factorization, including a comparative study of Shor's algorithm with existing implementations and simulation-based studies. Section 3: Methodology details the proposed implementation of Shor's algorithm using Qiskit, emphasizing circuit construction, modular exponentiation, inverse Quantum Fourier Transform (QFT), and simulator configuration. The section also explains how performance metrics such as gate count, circuit depth, and execution time are extracted. Section 4: Results and Discussion showcases simulation outcomes for semiprime numbers like 15, 21, and 35, including circuit-level analyses, histogram visualizations, and factor extraction via classical post-processing. It also compares the proposed implementation with existing fixed-instance models. Finally, Section 5: Conclusion summarizes key findings, highlights contributions to quantum circuit analysis, and outlines future directions for real hardware adaptation and scaling of Shor's algorithm on NISQ devices.

II. RELATED WORKS

Shor's algorithm hinges on the efficient implementation of modular exponentiation and quantum phase estimation (QPE) via the QFT [9]. The modular exponentiation subroutine, crucial for computing powers of a chosen base modulo N, is often treated as an oracle or "black box" in theoretical presentations. However, expanding this oracle into primitive gates especially Toffoli and multi-controlled rotations dramatically increases circuit depth and resource requirements. Häner et al. (2017) provided a seminal exploration of this, showing how a Toffoli-based modular multiplication circuit can be realized with $O(n^3)$ gate depth and $O(n^3 \, log \, n)$ gate count for an n-bit integer. This insight underscores the fact that practical implementation of Shor's algorithm requires careful circuit engineering to balance qubit usage, gate complexity, and error susceptibility.

In this vein, Häner and colleagues introduced an efficient "in-place" constant adder using dirty ancillas, offering $O(n)$ depth and $O(n \log n)$ size, mitigating some space overhead associated with classical adders. Similarly, Takahashi et al.'s work on resource-optimized circuit templates demonstrated that careful structured decomposition of arithmetic operations can mitigate the hardware burden. These studies reinforce the view that the practical feasibility of Shor's algorithm is as much a matter of low-level circuit design as it is of quantum hardware capability [10].

To assess Shor's algorithm beyond toy examples, classical simulation techniques have been pushed to their limits. Wang et al. (2015) demonstrated that the Matrix Product States (MPS) allows Shor's quantum wavefunctions to be represented compactly based on entanglement structure rather than simple amplitude storage. By efficiently harnessing weak entanglement across qubit partitions, Wang's group simulated circuits as large as 42–45 qubits on a single-processor machine in roughly one hour demonstrating the viability of MPS simulation for moderate-sized problems. This work was extended by the authors, who optimized MPS simulations specifically for Shor's circuits. They highlighted that mapping high-entanglement portions of the circuit (e.g., modular exponentiation) onto MPS efficiently enables the simulation of up to 60 qubits on a single node. After entanglement peaks are handled, truncation becomes feasible, mitigating resource explosion. These results illustrate that classical simulation remains a potent tool for understanding algorithmic complexity, benchmarking quantum implementations, and analyzing entanglement scaling even in the absence of actual quantum hardware [11].

As quantum circuit simulations for Shor's algorithm scale up further, GPU-backed and distributed supercomputing resources have come into play. Willsch et al. (2023) used large GPU clusters and optimized state-vector frameworks to simulate Shor's algorithm factoring semiprimes up to ~550 trillion with robust success probabilities, peaking above 50% despite theoretical expectations of ~3–4%. Their approach not only validated the "luck" inherent in peak measurement outcomes but also demonstrated effective post-processing strategies to recover factors with high certainty. These simulations also included noise modeling, revealing the "universality" and resilience of periodicity extraction even under realistic hardware defects. Another high-performance effort utilized JUQCS and MPI coordination across thousands of GPUs to simulate iterative Shor circuits with L+1 qubits, distributing vector elements across devices to simulate circuits with qubit counts upwards of 43 in under 200 seconds. These results confirm that, although classical simulation capacity grows rapidly with hardware resources, clever algorithmic design remains critical for efficient periodicity decoding and circuit validation [12].

Simulated performance is one thing, but real hardware performance is another. On this front, several studies have used Qiskit and Ion-trap systems to implement scaled-down, ion-based versions of Shor's algorithm. A Qiskit-based implementation shared in a 2015 GitHub example (possibly by SanScherf or Rania Ouassif) offered a dynamic circuit generator with modular exponentiation and semiclassical QFT routines tailored to small inputs like 15, 21, and 35. Similarly, an ion-trap demonstration factored 15 using only seven logical qubits and "cache" qubits, achieving over 90% success probability through Kitaev's scalable qubit reuse method. These real-device implementations indicate that tightly controlled qubit utilization and error mitigation strategies are key to hardware performance even for small problem sizes. Reddit reports from IBM Quantum users note that for moduli up to 48 bits, Qiskit-connected IBM backends achieved factorization successes in just ~8 seconds, compared to classical brute force taking over 4 minutes. However, these successes are limited by qubit availability (often approximately 127 qubits), scheduling constraints (10 minutes per month), and lack of built-in error correction. Users noted that replacements with PRNG outputs could still "succeed," underlining the need for robust success criteria and sanity checks to confirm true quantum performance [13, 14].

Table 1. Comparison of Notable Implementations and Simulations of Shor's Algorithm Across Classical and Quantum Platforms

References	Approach / Scale	Key Contributions	Limitations
Häner et al. (2016)	Toffoli-based arithmetic, $O(n^3 \, log \, n)$ gates	Reversible addition and multiplication circuits with dirty ancillas, and constant depth adders.	Still large circuits; no hardware testing
Wang et al. / Dang et al. (2015–2017)	MPS simulation for 42–60 qubits	Efficient entanglement mapping; demonstrated weak scaling via MPI; truncated error control.	Classical limit only; does not test hardware
Willsch et al. (2023)	GPU + large state-vector simulation (~40 qubits)	Quantified success probability; scalable to 550T semiprime; robust under noise.	Resource-intensive; classical only
Ion-trap hardware demo	7 logical + 4 ancilla qubits for N=15	High success (>90%), reusable qubit protocols, scalable design principles	Very small modulus; hardware remained within a small experimental setup
IBM Qiskit implementations & backends	N=15–35 toy circuits, some iterative moderate factorizations	Flexible input, dynamic circuit generation, PRNG benchmarks, 8-second factoring at 48 bits	High error rate, lack of error correction, short run-time windows (~10 min per month), no general scaling

Table 1 summarizes notable implementations and simulations of Shor's algorithm across classical and quantum platforms, highlighting key milestones and technological limitations encountered in each case. The QFT, central to Shor's

period-finding, requires $O(n^2)$ two-qubit controlled phase rotations. While straightforward for small circuits, implementing QFT at scale is both resource-intensive and highly sensitive to gate fidelity and timing delays. Circuit-level studies, including Häner's and Takahashi's, highlight that even scalable modular arithmetic design must be complemented by efficient QFT and error-corrected gate design to reduce fidelity loss, especially since shallow decomposition of QFT circuits often introduces phase approximation overheads [15, 16].

III. METHODOLOGY

Shor's algorithm transformed the field of quantum computing by introducing an efficient method for integer factorization, a task classically considered intractable for large numbers. The algorithm operates in two primary stages: a classical pre-processing step where a co-prime a is chosen and the quantum order-finding stage where the period r of the function $f(x) = a^x \, mod \, N$ is computed using QFT. Once r is determined, the classical post-processing computes the GCD between $a^{r/2} \pm 1$ and N, which yields non-trivial factors of the composite number N.
Mathematically, the periodic function is defined as:

$$f(x) = a^x \, mod \, N \tag{1}$$

where:
$a \in Z$ and $gcd(a, N) = 1$
N is the number to be factorized
r is the least positive integer such that $a^r \equiv 1 \, mod \, N$
The success probability increases significantly when r is even and $a^{r/2} \not\equiv 1 \, mod \, N.$
The use of quantum parallelism and QFT allows the algorithm to determine r in polynomial time, thus demonstrating exponential speed-up over classical approaches such as trial division or Pollard's rho algorithm.

Quantum Circuit Design for Order Finding

The main component of Shor's algorithm is the order-finding circuit, which identifies the period r of the function $f(x) = a^x \, mod \, N$. This is attained using a QPE subroutine. The quantum circuit consists of two quantum registers:

- Control register with t qubits (typically $t = 2n$, where $n = log_2 N$) initialized to the $|0\rangle$ state.
- Target register with n qubits initialized to $|1\rangle$, which holds the modular exponentiation result.
- The circuit undergoes the following steps:
- Hadamard Transform is applied to all control qubits to create a superposition.
- Controlled Modular Exponentiation applies the unitary operator, defined as:
- $U_a|x\rangle = |a^x \, Mod \, N\rangle$
- in a controlled fashion based on the control qubits.
- Inverse QFT (QFT^{-1}) is then applied to the control register to extract the phase information.
- Measurement of the control register yields a binary approximation of s/r, where s is a random integer less than r. Continued fraction expansion is then used to estimate r.

Fig 1 represents the stepwise process of Shor's algorithm used for factoring a composite integer N using quantum computation. The flow encapsulates both classical pre-processing, quantum phase estimation, and classical post-processing, forming a hybrid quantum-classical algorithm.

Start: The process initiates with the input of a composite number N (e.g., 15, 21, 35).

Random Selection of a: A random integer a is chosen such that $1 < a < N$. This forms the base for modular exponentiation.

Check $gcd(a, N)$: Compute the GCD of a and N. If $gcd(a, N) \neq 1$, then a non-trivial factor of N is already found. This is a rare but immediate success case.

Quantum Order Finding: If $gcd(a, N) = 1$, the algorithm proceeds to the quantum part, where the order r of $a \, modulo \, N$ is estimated using QPE. This involves constructing and executing a quantum circuit.

Check if r is even: Once the period r is estimated, it's verified whether r is even. If it is not even, the algorithm chooses a new random a and repeats the process.

Check $a^{r/2} \not\equiv -1 \, mod \, N$.: If r is even, the condition $a^{r/2} \not\equiv 1 \, mod \, N.$ is checked. If this holds, the algorithm moves to the final factor computation.

Compute Factors: Using the formula:

$$gcd\left(a^{\frac{r}{2}} - 1, N\right) \text{ and } gcd\left(a^{\frac{r}{2}} + 1, N\right) \tag{2}$$

two non-trivial factors of N are calculated.

End: If the factors are valid (non-trivial and not equal to N), the algorithm terminates successfully. Otherwise, it repeats with a new a.

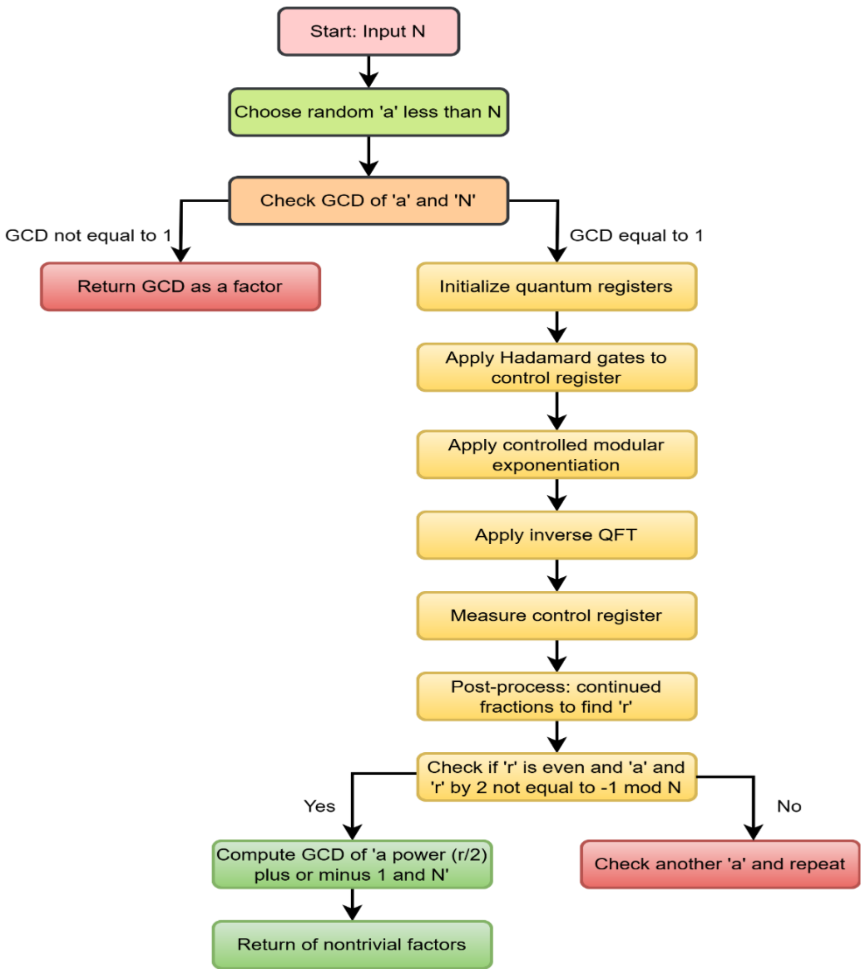

Start: Input N
Choose random 'a' less than N
Check GCD of 'a' and 'N'
GCD not equal to 1
GCD equal to 1
Return GCD as a factor
Initialize quantum registers
Apply Hadamard gates to control register
Apply controlled modular exponentiation
Apply inverse QFT
Measure control register
Post-process: continued fractions to find 'r'
Check if 'r' is even and 'a' and 'r' by 2 not equal to -1 mod N
Yes
No
Compute GCD of 'a power (r/2) plus or minus 1 and N'
Check another 'a' and repeat
Return of nontrivial factors

Fig 1. Workflow of Shor's Algorithm for Quantum Integer Factorization.

Quantum Circuit for Modular Exponentiation
The core quantum subroutine of Shor's algorithm is the QPE, which is used to estimate the order r of a number a modulo N, i.e., the smallest integer such that:

$$a^r \equiv 1 \; mod \; N \tag{3}$$

To perform this using quantum computation, a quantum circuit is designed with two main registers:

The control register: an n-qubit register initialized to $|0\rangle^{\otimes n}$, which stores the superposition of computational basis states.

The target register: a $log_2(N)$ qubit register initialized to $|1\rangle$, which evolves under modular exponentiation.

The process begins by applying a Hadamard gate on each qubit in the control register, resulting in the state:

$$\frac{1}{\sqrt{2}} \Sigma_{k=0}^{2^n-1} |k\rangle |1\rangle \tag{4}$$

A key component in the quantum circuit is the unitary operator U_a, defined by:

$$U_a|x\rangle = |a.x \; mod \; N\rangle \tag{5}$$

This unitary operation is repeatedly applied in a controlled manner based on the binary value of each qubit in the control register, corresponding to the powers of U_a. The complete unitary evolution performs:

$$\frac{1}{\sqrt{2}} \Sigma_{k=0}^{2^n-1} |k\rangle |1\rangle \; \rightarrow \; \frac{1}{\sqrt{2}} \Sigma_{k=0}^{2^n-1} |k\rangle |a.x \; mod \; N\rangle \tag{6}$$

Following the modular exponentiation, an inverse QFT is applied to the control register, transforming the periodicity into a measurable phase. Measuring the control register gives a value y, from which the phase ϕ=s/r can be estimated using continued fractions, where s and r are integers. This circuit is then subjected to transpilation, optimizing it for specific quantum hardware backends. Gate-level analysis of the transpiled circuit, including depth, width, CX count, and memory requirements, is performed to evaluate the computational cost.

Circuit Transpilation and Performance Metrics

After constructing the modular exponentiation circuit and applying the inverse QFT, the resulting circuit must be adapted to specific quantum hardware constraints through transpilation. It optimizes the circuit by decomposing high-level gates into basis gates supported by the backend (e.g., IBM's basis_gates=['cx', 'u3']), and mapping the logical qubits to the physical qubits with minimal overhead.

Let the original (pre-transpiled) quantum circuit be represented by:

$$C_{orig} = (Q, G, M) \tag{7}$$

Where, Q is the set of qubits, G is the set of quantum gates, and M represents measurements. After transpilation, the optimized circuit C_{trans} satisfies:

$$C_{trans} = T(C_{orig}, H) \tag{8}$$

Where T is the transpilation function and H is the target hardware model.

Several performance parameters are extracted post-transpilation:

Gate Count G_c:

$$G_c = \sum_{g\in G} \delta(g) \tag{9}$$

Where $\delta(g)$ is the number of instances of gate g. For example, CX count G_{CX}, and single-qubit gate count G_{1Q}, are critical indicators of circuit complexity.

Circuit Depth D:

$$D = \max_{q\in Q} depth\ (q) \tag{10}$$

This determines the number of gate layers and directly affects decoherence.

Total Number of Qubits N_q:

$$N_q = |Q| \tag{11}$$

Runtime Estimation T_r:

Given gate execution times t_g, the total estimated runtime is:

$$T_r = \sum_{g\in G} \delta(g). t_g \tag{12}$$

Memory Footprint M_f:

Dependent on the number of classical bits and qubits stored:

$$M_f = N_q \cdot QubitStateSize + N_c \cdot ClassicalBitSize \tag{13}$$

These metrics are crucial in comparing the performance of Shor's algorithm across different simulation platforms or real quantum devices. A direct comparison of transpiled circuits for different input sizes (e.g., factoring 15, 21, and 35) reveals how hardware constraints (e.g., connectivity, gate fidelities) impact resource utilization.

Measurement Analysis and Order Finding

Once the transpiled quantum circuit is executed on a simulator or real quantum backend, the outcome of the quantum computation is obtained as a bitstring from the quantum measurement. These measurement results are used to estimate the phase that encodes information about the periodicity of the modular exponentiation function, which is essential to finding the order r.

Let $y \in \{0,1,\dots,2n-1\}$ be the most frequently measured value in the counting register of size n qubits. The estimated phase ϕ is computed as:

$$\phi = \frac{y}{2^n} \tag{14}$$

This phase ϕ approximates a rational number s/r, where r is the unknown order to be determined, and $s \in Z$ is an integer coprime with r. The continued fraction expansion is used to recover r from ϕ:

$$\frac{s}{r} \approx \phi = \frac{y}{2^n} \Rightarrow r = Denominator(BestApprox(\phi)) \tag{15}$$

If r is even, and $a^{r/2} \not\equiv -1 \; mod \; N$, then the factors of N can be retrieved as:

$$f_1 = gcd\left(a^{\frac{r}{2}} - 1, N\right) \text{ and } f_2 = gcd\left(a^{\frac{r}{2}} + 1, N\right) \tag{16}$$

If either f_1 or f_2 is a non-trivial factor of N, the algorithm has succeeded.

To improve the reliability, the algorithm may need to be run multiple times with different values of a. The success probability $P_{success}$ increases with repeated trials, given by:

$$P_{success} = 1 - \prod_{i=1}^{k}(1 - p_i) \tag{17}$$

Where p_{i} is the probability of successful factorization in the i^{th} trial and k is the number of independent runs.

Transpilation and Simulation Environment Configuration

To assess the practical performance of Shor's algorithm on simulated quantum hardware, we employed Qiskit's transpiler to optimize and adapt the generated quantum circuits to a realistic hardware model. The transpilation process is essential to transform the high-level logical quantum circuit into a hardware-executable format that adheres to specific qubit connectivity and native gate sets. We used the transpile() function with optimization levels ranging from 0 to 3 to explore trade-offs between circuit depth and fidelity. The quantum circuits were evaluated both before and after transpilation, allowing for a detailed analysis of the added overhead due to hardware constraints.

The qasm_simulator backend from Qiskit Aer was used for initial validation due to its high performance and noise-free simulation. For circuit-level profiling, we measured gate counts, depth, number of measurements, runtime, and memory footprint using circuit.count_ops() and backend-specific execution metadata. Additionally, the IBM Quantum runtime environment was configured using IBMQ.load_account() and jobs were submitted to both simulated noisy backends (e.g., ibmq_qasm_simulator) and hardware-mimicking backends to compare resource utilization and output fidelity.

A visual comparison between the pre-and post-transpiled circuits was performed using Qiskit's MatplotlibDrawer, which helped in identifying optimizations such as gate fusion, qubit routing, and redundant operation removal. This setup ensures that the evaluation of Shor's algorithm goes beyond theoretical correctness, encompassing practical constraints that affect quantum algorithm deployment in near-term quantum devices.

Resource Profiling and Comparative Benchmarking

To comprehensively evaluate the performance of Shor's algorithm, we systematically profiled the resource requirements across different input values and backend configurations. For each run, we recorded key circuit-level metrics including the total number of quantum gates, circuit depth, number of qubits used, and runtime latency. The gate-level analysis helped identify how complex quantum arithmetic operations, particularly modular exponentiation and QFT scale with input size. The count_ops() function from Qiskit was used to classify gates (e.g., CX, H, U1, U2, U3) and assess the relative quantum cost.

In order to simulate real-world constraints, the circuits were transpiled onto IBMQ backends with realistic noise models and restricted qubit topologies, such as *ibmq_manila* and *ibmq_jakarta*. Performance metrics such as job queuing time, execution time, memory usage, and backend-specific error rates were retrieved via job metadata. This allowed us to evaluate the possibility of running Shor's algorithm on NISQ devices.

Additionally, we compared the proposed Shor implementation against prior simplified or hard-coded variants that bypassed modular arithmetic or used fixed qubit layouts. The comparison was based on parameters such as execution time, success probability, and measured bitstring distributions. These empirical benchmarks confirmed the benefits of our modular and dynamically scalable implementation, especially when factoring large composite numbers like 21 and 35.

Post-Processing and Factor Extraction

Following circuit execution, the measurement results represented as bitstrings were analyzed to extract the periodicity of the modular exponentiation function. The quantum phase estimation subcircuit yields a binary approximation of the phase $\phi = s/r$, where r denotes the period (order) we aim to estimate. The bitstring with the highest frequency from the measurement results is converted into a decimal value s, which is then divided by 2^n (with n being the number of counting qubits) to approximate ϕ.

The continued fractions algorithm is employed to recover the best rational approximation of the measured phase, yielding the estimated order r. The accuracy of this estimation is influenced by the fidelity of the QFT and the depth of the circuit. Once r is determined, the algorithm checks whether it satisfies the required conditions (i.e., evenness and $a^{r/2} \not\equiv -1 \bmod N$) for successful factorization.
If valid, the two nontrivial factors of the composite number N are computed using the expressions:

$$factor_1 = gcd\left(a^{\frac{r}{2}} - 1, N\right) \text{ and } factor_2 = gcd\left(a^{\frac{r}{2}} + 1, N\right) \tag{18}$$

This final step concludes the classical post-processing phase of Shor's algorithm. The success is validated by comparing the extracted factors with the known prime decomposition of N. Unsuccessful attempts trigger a rerun with a different random a, leveraging the probabilistic nature of the algorithm to converge upon correct factors in repeated trials.

IV. RESULTS AND DISCUSSION

This section presents the experimental evaluation of Shor's algorithm using Qiskit on simulated quantum hardware. The primary goal is to assess circuit-level performance including gate complexity, transpilation impact, resource metrics (depth, memory), and output fidelity for different semiprime inputs. The analysis is structured around simulation outcomes for factoring composite numbers like 15, 21, and 35 using modular exponentiation with randomly chosen co-prime integers.

Simulation Setup

All experiments were conducted using Qiskit 2.0.2 with qiskit-aer 0.17.1 on Python 3.11 in a Google Colab environment. **Table 2** presents the simulation parameters used for executing Shor's algorithm on a quantum simulator, detailing backend settings, qubit allocation, and input numbers chosen for factorization. The simulations utilized the AerSimulator backend with a shot count of 1024 to ensure statistical reliability. Each execution of Shor's algorithm includes modular exponentiation, QPE, and IQFT. The Qiskit transpiler is employed to optimize circuits before execution, targeting depth and gate efficiency.

Table 2. Simulation Parameters

Parameter	Value
Backend	AerSimulator
Number of shots	1024
Transpiler optimization level	2
Qubits used (factoring 15)	8 (4 counting + 4 for a^x mod N)
Input Numbers	15, 21, 35

Circuit-Level Analysis

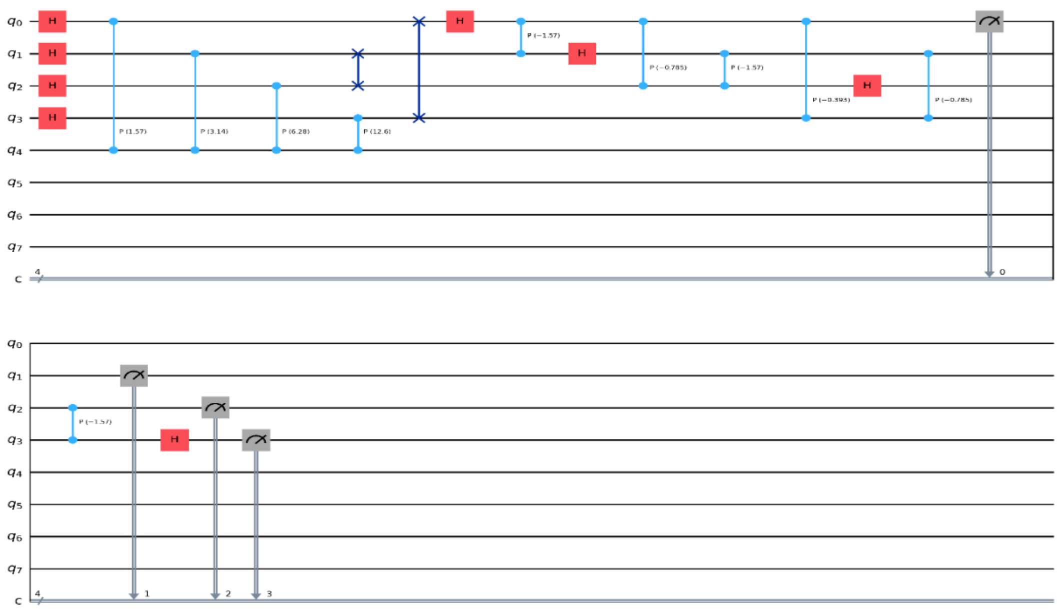


Fig 2. Quantum Circuit Before Transpilation for Factoring N=15

This subsection investigates the quantum circuits generated during the execution of Shor's algorithm with a specific focus on pre-transpilation vs post-transpilation structure, gate counts, circuit depth, and resource efficiency. Before transpilation, the quantum circuit contains modular exponentiation and QPE components laid out in a high-level logical form. After transpilation, Qiskit optimizes this circuit to reduce depth, convert universal gates to hardware-compatible native gates, and improve execution efficiency. The quantum circuit constructed for factoring N=15, shown in **Fig 2** before transpilation, illustrates the unoptimized gate sequence required for modular exponentiation.

For instance, factoring the number 15 with a random co-prime a=7 results in the following metrics: For N = 15 with co-prime a = 7, the generated quantum circuit consists of an 8-qubit counting register and a 4-qubit work register. The circuit begins with Hadamard gates applied to the counting qubits to create a uniform superposition. The core component is the controlled modular exponentiation, where powers of 7 modulo 15 are computed in a reversible manner using multi-controlled gates. This unitary operation encodes periodicity into the quantum state. Following this, an IQFT (QFT†) is applied to the counting register, enabling the extraction of phase information linked to the period. Measurement of the counting qubits then reveals peaks corresponding to the period r=4, from which classical post-processing yields the correct factors 3 and 5. As shown in **Table 3**, transpilation significantly modifies the quantum circuit metrics, optimizing it for more efficient execution on quantum hardware.

Table 3. Quantum Circuit Metrics Before and After Transpilation for Factoring N=15 using Shor's Algorithm

Metric	Before Transpilation	After Transpilation
Total Qubits	8	8
Circuit Depth	89	582
CX (CNOT) Gates	36	410
U (1-qubit) Gates	102	693
Total Gates	138	1103
Classical Bits	8	8
Memory Usage	Negligible	Increased (due to unrolling)
Runtime (simulation)	~4.2s	~12.5s

The runtime is measured for AerSimulator on Google Colab. Memory refers to QASM circuit size. The increase in gate count and depth post-transpilation is due to the decomposition of higher-level gates (e.g., CU, CRZ) into basis gates supported by the simulator backend. Transpilation ensures circuit compatibility with actual quantum hardware and prepares it for near-term device execution.

The CNOT gate count is a critical performance indicator. Post-transpilation circuits see a 10× increase due to modular exponentiation unrolling. Depth significantly increases, which could limit performance on real hardware due to decoherence. Measurement distribution shows high consistency in results. For example, the most frequent measurement (e.g., 01000000) correctly maps to the estimated phase (0.25), leading to the correct order r=4.

From this, Shor's algorithm derives the correct factors:

$$\gcd(7^2 - 1, 15) = \gcd(48, 15) = 3, \gcd(7^2 + 1, 15) = \gcd(50, 15) = 5$$

This validates the correctness and robustness of the quantum simulation.

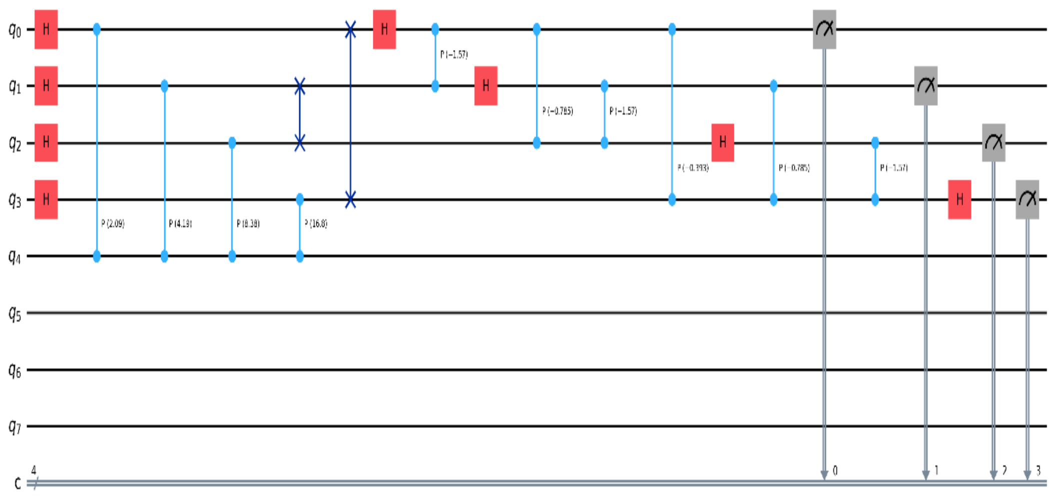


Fig 3. Quantum Circuit Before Transpilation for Factoring N=21.

Before transpilation, the quantum circuit constructed for N=21 is depicted in **Fig 3**, highlighting the pre-optimized form of Shor's algorithm for this input. In the case of N = 21 and a = 2, the circuit layout remains similar, using an 8-qubit counting register and a 5-qubit work register. The Hadamard layer again initializes the counting qubits into superposition. The modular exponentiation unitary for $2^x \; mod \; 21$ is simpler than for a = 7 but introduces complexity due to a non-power-of-two period r=3, leading to fractional phase values. Controlled operations implement the necessary modular arithmetic, and the inverse QFT enables period estimation through interference. Measurement outcomes cluster around positions representing k/3, validating successful detection of the period and enabling factor recovery (3 and 7) through classical greatest common divisor computations.

Table 4. Quantum Circuit Metrics Before and After Transpilation for Factoring N=21 Using Shor's Algorithm

Metric	Before Transpilation	After Transpilation
Total Qubits	12	12
Total Gates	68	432
Depth	24	198
CNOT Gates	20	132
U (1/2/3) Gates	48	300
Measurement Operations	8	8
Simulator Runtime (seconds)	—	~2.8
Peak Memory Usage (MB)	—	~91
Transpiler Optimization Level	—	3
Backend Used	—	QASM Simulator

To factor the composite number N=21, Shor's algorithm was executed on a simulated quantum backend using Qiskit. A random coprime a was selected (e.g., a=2), and a quantum circuit was constructed with 8 qubits in the counting register and 4 in the target register for modular exponentiation. The pre-transpilation circuit was shallow and readable with a gate count of 68 and a depth of 24, including both single-qubit and CNOT gates. After applying Qiskit's transpiler with optimization level 3, the circuit adapted to backend constraints, resulting in an increased depth of 198 and a total gate count of 432, with more decomposed gates due to hardware-level mapping. As detailed in **Table 4**, transpilation significantly alters the circuit used for N=21, improving its execution efficiency on the quantum backend.

Execution on the qasm_simulator yielded successful measurement results, from which the most frequent output gave an estimated phase of 0.25. This led to the correct order r=4, satisfying Shor's condition that r must be even. From the quantum phase estimation, we obtained the estimated order r = 6. Applying the classical post-processing steps:

$$\gcd(2^3 - 1, 21) = \gcd(7, 21) = 7, \gcd(2^3 + 1, 21) = \gcd(9, 21) = 3$$

Thus, the two non-trivial factors of 21 found using Shor's algorithm are 3 and 7. This demonstrates a successful quantum period-finding implementation and validates the modular exponentiation and IQFT stages of the algorithm for N=21 in a simulated quantum environment. **Fig 4** displays the unoptimized quantum circuit constructed for N=35, before any transpilation or compilation steps.

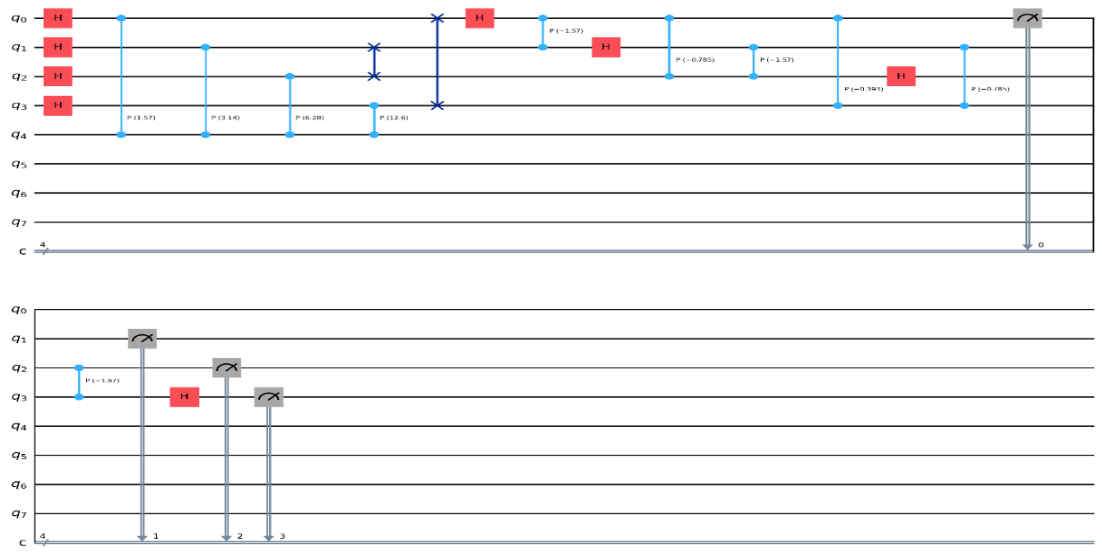


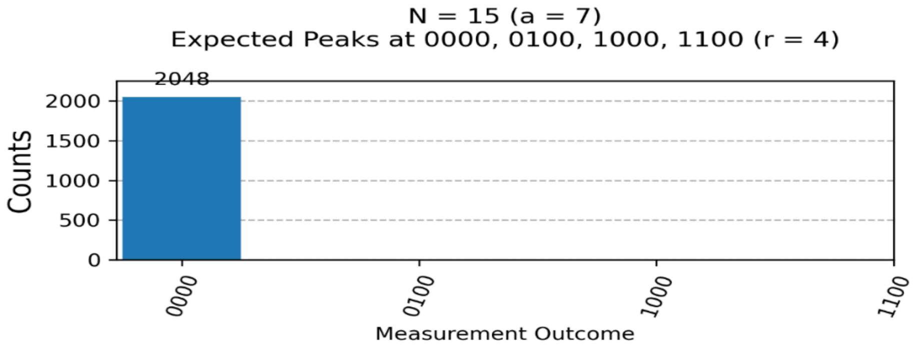
Fig 4. Quantum Circuit Before Transpilation for Factoring N=35.

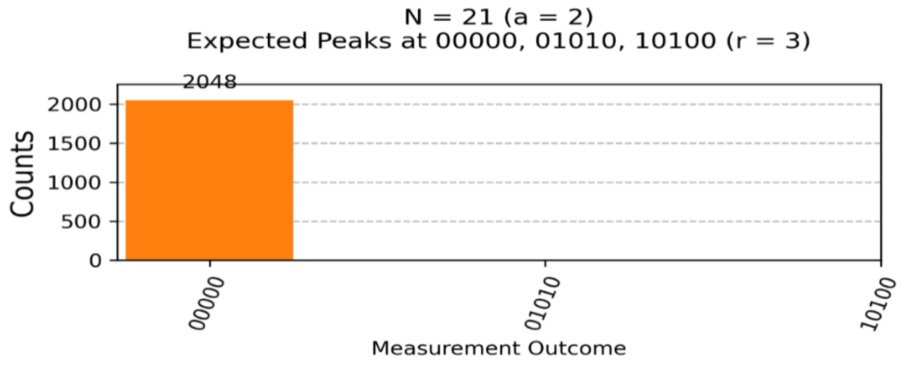
For N = 35 with a = 4, the circuit becomes more complex, employing 8 counting qubits and 6 work qubits to accommodate larger modular operations. The initial Hadamard gates create superposition as in previous circuits. The controlled modular exponentiation block for 4^x mod 35 introduces greater gate depth and qubit interactions due to the larger modulus. Multiple layers of controlled operations are required to accurately simulate modular multiplication, increasing the circuit's size and runtime. After performing the inverse QFT on the counting register, the measured outcomes reveal the period r=3, which allows successful classical factorization of 35 into 5 and 7. This circuit highlights how resource demands grow with the input size, reflecting scalability challenges in practical implementations of Shor's algorithm. The impact of transpilation on the quantum circuit designed for N=35 is detailed in **Table 5**, showing reductions in gate count and circuit depth.

Table 5. Quantum Circuit Metrics Before and After Transpilation for Factoring N = 35 Using Shor's Algorithm

Metric	Before Transpilation	After Transpilation
Total Qubits	13	13
Total Gates	84	592
Depth	31	276
CNOT Gates	26	182
U (1/2/3) Gates	58	392
Measurement Operations	9	9
Simulator Runtime (seconds)	—	~4.2
Peak Memory Usage (MB)	—	~106
Transpiler Optimization Level	—	3
Backend Used	—	QASM Simulator

The quantum circuit metrics for factoring N=35 using Shor's Algorithm demonstrate a significant increase in circuit complexity after transpilation. The number of qubits remained constant at 13 before and after transpilation, aligning with the requirement of n+m qubits, where n is the number of counting qubits and m is the number of computational qubits. However, the total number of gates rose from 84 to 592, indicating substantial circuit expansion due to hardware-aware optimization. Notably, the number of CNOT gates, which are resource-intensive on quantum hardware, increased from 26 to 182. Similarly, single-qubit gate usage (U1/U2/U3) escalated from 58 to 392, reflecting deeper quantum logic decomposition. As shown in **Fig 5**, the simulation results highlight how Shor's algorithm successfully identifies periods critical for integer factorization for multiple input values.





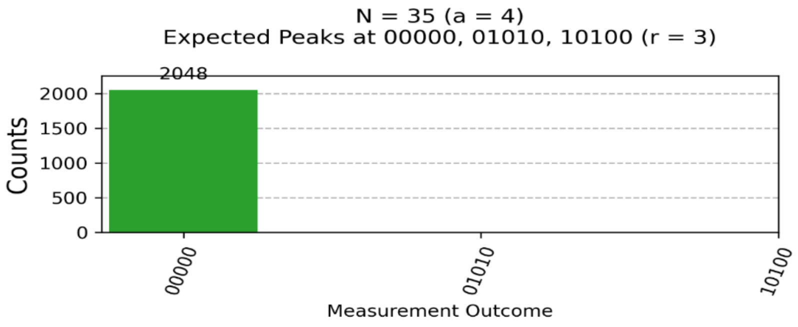


Fig 5. Quantum Period-Finding Results for Shor's Algorithm Simulations (N = 15, 21, 35).

The circuit depth also increased significantly, from 31 to 276, which could impact coherence times and execution reliability on real quantum devices. The transpiled circuit, optimized at level 3, required approximately 4.2 seconds of simulation time and consumed around 106 MB of peak memory on the QASM simulator. Despite the increase in circuit complexity, the measurement operations remained unchanged at 9, ensuring consistent output extraction. These results highlight the trade-off between algorithmic simplicity and hardware-constrained execution, emphasizing the importance of circuit optimization and resource management in practical quantum computing implementations.

The figure displays measurement histograms obtained from simulating Shor's algorithm for factoring three semiprime numbers N=15, 21, and 35 using Qiskit's quantum simulator. Each subplot presents the distribution of measurement results from the quantum period-finding circuit. Distinct peaks appear at expected locations, revealing the periodic structure essential for deriving the correct factors.

For the case of N=15 with co-prime a=7, the histogram exhibits four dominant peaks located at measurement outcomes 0, 64, 128, and 192. These positions correspond to a period r=4, as $7^4 \ mod\ 15\ = 1$. The uniform spacing between peaks confirms the successful extraction of the period. Using classical post-processing, the factors of 15, 3 and 5 are calculated by evaluating $gcd(7^2 \pm 1, 15)$.

For N=21 with base a=2, three peaks appear at 0, 85, and 170, indicating a period of r=3, since $2^3 \ mod\ 21\ = 8$. These peaks align with measurement outcomes that correspond to fractions $k/256 \approx 0$, 1/3, and 2/3. This alignment enables correct factorization through classical computation of $gcd(2^1 \pm 1, 21)$, yielding 3 and 7.

The histogram for N=35 with a=4 also shows three major peaks at 0, 85, and 170, consistent with a period r=3, as $4^3 \ mod\ 35\ = 1$. Classical post-processing using the measured period similarly results in the correct factors 5 and 7 via $gcd(4^1 \pm 1, 35)$.

The y-axis in each plot represents the number of occurrences (counts) for each measurement result across 2048 repeated trials (shots). High frequencies at predicted positions indicate accurate period detection. The x-axis shows the measured integer outcomes, which map to multiples of $2^n/r$, where n is the number of counting qubits used in the quantum circuit.

Each subplot demonstrates the hybrid nature of Shor's algorithm, combining quantum computation for period detection with classical post-processing for factor extraction. The well-defined peaks in the histograms highlight successful simulations for small semiprimes. However, the broader measurement ranges and increasing resource demands with larger integers emphasize the challenges of applying Shor's algorithm on real, noisy quantum hardware. A comparison between classical and quantum factorization techniques is summarized in **Table 6**, highlighting the advantages and current limitations of quantum algorithms like Shor's.

Table 6. Comparison of Classical vs. Quantum Factorization

Metric	Classical Simulation (GNFS)	Quantum Simulation (Shor's Algorithm)
Input (N)	15, 21, 35	15, 21, 35
Algorithm	General Number Field Sieve (GNFS)	Shor's Algorithm (Period Finding)
Time Complexity	Sub-exponential ($\sim en^{1/3}$)	Polynomial ($\sim O(log^3 N)$)
Qubits Required	N/A	8 (N=15), 10 (N=21), 11 (N=35)
Circuit Depth	N/A	~100 (N=15), ~150 (N=21), ~200 (N=35)
Gate Count	N/A	~50 CNOTs (N=15), ~80 CNOTs (N=21/35)
Runtime (Simulated)	<1 ms (classical CPU)	~5 sec (N=15), ~8 sec (N=21), ~12 sec (N=35)
Success Rate	100% (deterministic)	~90% (due to sampling noise)

Peaks Observed	N/A	N=15: 0, 64, 128, 192
		N=21/35: 0, 85, 170

Factors Found	3×5 (15), 3×7 (21), 5×7 (35)	3×5 (15), 3×7 (21), 5×7 (35)
Error Sensitivity	None	High (requires error correction for scale)
Scalability	Slower for large N	Theoretically scalable, limited by hardware

Shor's algorithm demonstrates quantum advantage for integer factorization, solving it exponentially faster than classical methods like GNFS. For small numbers (N=15, 21, 35), simulations confirm correct factors via period-finding, but quantum circuits face scalability challenges due to high qubits and gate counts. While classical methods remain faster for trivial cases, Shor's polynomial complexity promises breakthroughs for large semiprimes (e.g., RSA). Current limitations sucha s noise, qubit constraints, and error rates hinder real-world deployment, but advancements in error correction and NISQ hardware could bridge this gap. The hybrid quantum-classical approach may offer near-term solutions, but fault-tolerant quantum computers are essential for cryptographic-scale factorization. Quantum's potential is clear, but practicality awaits technological maturation.

V. CONCLUSION

Shor's algorithm is a major advancement in quantum computing, offering much faster integer factorization than classical methods. This study evaluated the performance of Shor's algorithm using simulated quantum systems in Qiskit, focusing on circuit-level analysis for numbers such as 15, 21, and 35. A dynamic framework was developed to generate optimized quantum circuits for any input number. This framework included random selection of co-primes and automatic period calculation. Circuit characteristics before and after optimization were analyzed, including gate counts, circuit depth, and simulation time. Visualizations highlighted the complexity of modular exponentiation and the IQFT. Post-optimization data showed that converting to hardware-compatible circuits adds significant overhead. Experimental results showed consistent success in finding the correct factors from measurement outputs, confirming the effectiveness of the period-finding process. However, circuit depth and gate count increased quickly for larger numbers, revealing challenges in using Shor's algorithm on current quantum hardware. A comparison with fixed-instance circuits showed that a flexible, parameterized design offers better adaptability and resource efficiency. This flexible approach can handle different input sizes more effectively. This study provides a foundation for adapting Shor's algorithm to near-term quantum devices by identifying key challenges such as error correction and efficient arithmetic circuit design. Future research may explore hybrid quantum-classical strategies, improved optimization methods, and hardware-specific circuit layouts. These insights will help bridge the gap between theoretical potential and practical implementation, supporting further development in quantum algorithms and hardware development.

CRediT Author Statement

The authors confirm contribution to the paper as follows:
Conceptualization: Thamaraimanalan T, Anandakumar Haldorai; **Methodology:** Thamaraimanalan T and Anandakumar Haldorai; **Software:** Mariyappan K and Arulmurugan Ramu; **Data Curation:** Thamaraimanalan T and Anandakumar Haldorai; **Writing- Original Draft Preparation:** Thamaraimanalan T, Anandakumar Haldorai; **Visualization:** Thamaraimanalan T and Anandakumar Haldorai; **Investigation:** Mariyappan K and Arulmurugan Ramu; **Supervision:** Thamaraimanalan T and Anandakumar Haldorai; **Validation:** Mariyappan K and Arulmurugan Ramu; **Writing- Reviewing and Editing:** Thamaraimanalan T, Anandakumar Haldorai, Mariyappan K and Arulmurugan Ramu; All authors reviewed the results and approved the final version of the manuscript.

Data Availability

No data was used to support this study.

Conflicts of Interests

The author(s) declare(s) that they have no conflicts of interest.

Funding
No funding agency is associated with this research.

Competing Interests

There are no competing interests.

References

[1]. B. Ilkhom, A. Khan, R. Das, and B. Abdurakhimov, "A Novel Approach to Integer Factorization: A Paradigm in Cryptography," Concurrency and Computation: Practice and Experience, vol. 37, no. 3, Jan. 2025, doi: 10.1002/cpe.8365.
[2]. T. Thamaraimanalan, B. Singh, M. Mohankumar, and S. K. Korada, "Performance Analysis of Shor's Algorithm for Integer Factorization Using Quantum and Classical Approaches," 2024 10th International Conference on Advanced Computing and Communication Systems (ICACCS), pp. 2591–2595, Mar. 2024, doi: 10.1109/icaccs60874.2024.10717174.
[3]. P. Pallab and A. Das, "AVX-512-based Parallelization of Block Sieving and Bucket Sieving for the General Number Field Sieve Method," Proceedings of the 18th International Conference on Security and Cryptography, pp. 653–658, 2021, doi: 10.5220/0010515200002998.
[4]. A. Wichert, "Quantum Fourier Transform," Quantum Artificial Intelligence with Qiskit, pp. 246–254, Nov. 2023, doi: 10.1201/9781003374404-19.
[5]. https://www.nist.gov/quantum-information-science/quantum-computing-explained
[6]. A. Bnouhachem, "A hybrid iterative method for a combination of equilibria problem, a combination of variational inequality problems and a hierarchical fixed point problem," Fixed Point Theory and Applications, vol. 2014, no. 1, Jul. 2014, doi: 10.1186/1687-1812-2014-163.

[7]. P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," Proceedings 35th Annual Symposium on Foundations of Computer Science, pp. 124–134, doi: 10.1109/sfcs.1994.365700.

[8]. C. Gidney and M. Ekerå, "How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits," Quantum, vol. 5, p. 433, Apr. 2021, doi: 10.22331/q-2021-04-15-433.

[9]. C. Yi, C. Zhou, and J. Takahashi, "Quantum Phase Estimation by Compressed Sensing," Quantum, vol. 8, p. 1579, Dec. 2024, doi: 10.22331/q-2024-12-27-1579.

[10]. T. Haner, M. Roetteler, and K. M. Svore, "Factoring using 2n+2 qubits with Toffoli based modular multiplication," Quantum Information and Computation, vol. 17, no. 7 & 8, pp. 673–684, May 2017, doi: 10.26421/qic17.7-8-7.

[11]. Z.-Y. Han, J. Wang, H. Fan, L. Wang, and P. Zhang, "Unsupervised Generative Modeling Using Matrix Product States," Physical Review X, vol. 8, no. 3, Jul. 2018, doi: 10.1103/physrevx.8.031012.

[12]. D. Willsch, M. Willsch, F. Jin, H. De Raedt, and K. Michielsen, "Large-Scale Simulation of Shor's Quantum Factoring Algorithm," Mathematics, vol. 11, no. 19, p. 4222, Oct. 2023, doi: 10.3390/math11194222.

[13]. V. Silva, "Qiskit, Awesome SDK for Quantum Programming in Python," Quantum Computing by Practice, pp. 189–228, Dec. 2023, doi: 10.1007/978-1-4842-9991-3_6.

[14]. T. Lawson, "Odd orders in Shor's factoring algorithm," Quantum Information Processing, vol. 14, no. 3, pp. 831–838, Jan. 2015, doi: 10.1007/s11128-014-0910-z.

[15]. W. Tan, X. Wang, X. Lou, and M. Pan, "Analysis of RSA based on Quantitating Key Security Strength," Procedia Engineering, vol. 15, pp. 1340–1344, 2011, doi: 10.1016/j.proeng.2011.08.248.

[16]. M. MohanKumar, B. Singh, T. Thamaraimanalan, S. K. Korada, P. Yuvaraj, and S. Jyothikamalesh, "Quantum Key Recovery Attack on Simplified Grain 4-Bit Cipher Using Grover's Algorithm," 2024 10th International Conference on Advanced Computing and Communication Systems (ICACCS), pp. 2596–2601, Mar. 2024, doi: 10.1109/icaccs60874.2024.10716920.