A Modular, Adaptive, and Scalable Quantum Factoring Algorithm

Alok Shukla *[1] and Prakash Vedula[2]

[1]School of Arts and Sciences, Ahmedabad University, India
[1]alok.shukla@ahduni.edu.in
[2]School of Aerospace and Mechanical Engineering, University of Oklahoma, USA
[2]pvedula@ou.edu

October 7, 2025

Abstract

Shor's algorithm for integer factorization offers an exponential speedup over classical methods but remains impractical on Noisy Intermediate-Scale Quantum (NISQ) hardware due to the need for many coherent qubits and very deep circuits. Building on our recent work on adaptive and windowed phase-estimation methods, we have developed a modular, windowed formulation of Shor's algorithm that potentially mitigates these limitations by restructuring phase estimation into shallow, independent circuit blocks that can be executed sequentially or in parallel, followed by lightweight classical postprocessing. This approach allows for a reduction in the size of the phase (or counting) register from a large number of qubits down to a small, fixed block size of only a few qubits (for example, three or four phase qubits were sufficient for the computational examples considered in this work), while leaving the work register requirement unchanged. The independence of the blocks allows for parallel execution and makes the approach more compatible with near-term hardware than the standard Shor's formulation. An additional feature of the framework is the overlap mechanism, which introduces redundancy between blocks and enables robust reconstruction of phase information, though zero-overlap configurations can also succeed in certain regimes. Numerical simulations verify the correctness of the modular formulation while also showing substantial reductions in counting qubits per block.

1 Introduction

Shor's algorithm for integer factorization remains one of the most striking examples of quantum advantage, offering an exponential speedup over the best-known classical factoring methods [1,2]. The algorithm relies on Quantum Phase Estimation (QPE) of modular exponentiation, implemented through the quantum Fourier transform, to recover the period of a function and thereby obtain nontrivial factors of a composite integer. Beyond its foundational importance, Shor's algorithm has profound implications for modern cryptography, since its efficient execution would compromise widely used public-key cryptographic systems.

Despite its theoretical power, the practical realization of Shor's algorithm faces substantial barriers. In its standard form, the algorithm requires a large phase (or counting) register of $2n + 1$ qubits, in addition to an n-qubit work register, together with long coherent circuits to implement repeated controlled modular exponentiation to factor an n-bit integer. Such resource demands

*Corresponding author.

exceed the capabilities of today's Noisy Intermediate-Scale Quantum (NISQ) devices, which are constrained by limited qubit counts, short coherence times, and noise. A range of strategies have been explored to mitigate these requirements, including circuit-level optimizations techniques [3–6].

Recently, we have introduced a modular and adaptive formulation of phase estimation. The Adaptive Windowed Quantum Phase Estimation (AWQPE) algorithm [7] decoupled estimation precision from the number of qubits in any single circuit block, enabling scalable estimation of a single eigenphase with small quantum registers. This idea was extended in the Adaptive Windowed Quantum Amplitude Estimation (AWQAE) framework [8], which demonstrated that windowing and adaptive refinement could be applied to estimation of two distinct eigenphases ϕ_1 and ϕ_2 (where $\phi_1 < 0.5 < \phi_2$) using a single ancilla qubit.

In this work, we generalize the AWQPE approach to integer factorization (that depends on quantum phase estimation involving superposition of multiple eignephases) and develop a modular version of Shor's algorithm that is tailored to near-term architectures. A key innovation is the introduction of an overlap mechanism between blocks, together with a carry-aware stitching procedure, which allows robust reconstruction of the full phase from multiple shallow, independent circuit executions. This overlap provides redundancy that potentially improves resilience to noise and sampling fluctuations, while retaining the option to operate with zero overlap in small-block regimes.

There is also active research aimed at reducing the resource requirements of the arithmetic layer of Shor's algorithm, including Toffoli-based modular multiplication using $2n + 2$ qubits [9], optimized modular exponentiation with reduced overhead [10], and cryptography-inspired approaches for additional qubit savings [11]. Our work is complementary to these efforts, as it targets the phase-estimation layer, and the combination of both directions offers the potential for cumulative improvements in the overall efficiency of quantum factoring.

The remainder of the paper is organized as follows. Section 2 describes our proposed modular approach to integer factorization based on Shor's algorithm. Section 3 presents numerical demonstrations of factoring small integers using shallow, quantum circuits. Computational complexity is discussed in Section 5. Section 6 concludes the paper with a summary of the results.

2 Modular Shor's Algorithm via Windowed QPE

In this section, we present a detailed description of our proposed modular version of Shor's Algorithm, formulated via Quantum Phase Estimation (QPE) with overlapping blocks of counting qubits. This novel approach is designed to mitigate the large qubit requirements of the original Shor's algorithm. The key idea is to partition the phase estimation procedure into a sequence of smaller, manageable quantum circuits, each responsible for a portion of the total QPE. The partial outcomes are subsequently integrated through classical post-processing to reconstruct the full phase, from which the order of $a \bmod N$ and thereby a non-trivial factor of the composite number N can be obtained.

Our main algorithm is summarized in Algorithm 1. It requires as input a composite integer N to be factored, together with a coprime base $a < N$. The additional parameters include the target register size $n_{\text{target}} = \lceil \log_2 N \rceil$, a choice of block sizes $\mathbf{m} = [m_1, \ldots, m_B]$ for counting qubits, and the corresponding overlap sizes $\mathcal{T} = [t_1, t_2, \ldots, t_B]$, where $t_1 = 0$ and, for $i \geq 2$, each t_i satisfies $t_i \leq \min(m_{i-1}, m_i)$.

The phase estimation is carried out in an iterative, blockwise manner. In the first iteration, it is applied to the initial block of m_1 counting qubits, which correspond to the m_1 most significant bits (MSBs) of the phase to be estimated. In the second iteration, the procedure is repeated on a block of size m_2, with an enforced overlap of t_2 bits with the first block. Thus, the overlap region is estimated twice, i.e., once as the final t_2 bits of the first block, and again as the leading t_2 bits of the second block. As will be shown later, this redundancy enables consistent stitching of the measurement outcomes from independent blocks to reconstruct the full phase estimate. In this step, the phase estimation therefore yields bit positions (m_1-t_2+1) through $(m_1-t_2+m_2)$. The process continues analogously for all subsequent blocks. At each iteration, fresh counting qubits may be allocated to estimate the targeted portion of the phase, with the overall structure determined by the chosen block sizes $\mathbf{m} = [m_1, \ldots, m_B]$ and overlap sizes $\mathcal{T} = [t_1, t_2, \ldots, t_B]$. In addition, the algorithm requires the number of measurement shots, the number of candidate outcomes to be retained, and the maximum number of candidate sequences to preserve during post-processing. Given these inputs, the algorithm outputs a non-trivial factor of N.

Algorithm 1 begins by partitioning the phase register into blocks of size $m_1, m_2, \ldots, m_B$, with specified overlaps between adjacent blocks. In step 1, each block is processed independently: the routine `RunAndProcessQPEBlock` (Algorithm 2) executes a QPE circuit for that block and returns a candidate set of phase bitstrings, with the exponent offset updated to align successive windows. In step 2, the sets from all blocks are merged by `CarryAwareIntegrationAndStitching` (Algorithm 3), which uses the overlap to enforce consistency across block boundaries and reconstruct full-length candidate phases. In step 3, the routine `RecoverPeriodAndFactor` (Algorithm 4) analyzes each reconstructed candidate using continued fractions to extract candidate periods, verifies them against the modular exponentiation, and computes non-trivial factors of N when possible. If a valid factor is found, the algorithm outputs it; otherwise, it returns that no non-trivial factor was obtained. If a non-trivial factor is not found, the procedure can be repeated with a new random base a. In the following, we provide a more detailed description of Algorithms 2 to 4.

2.1 Block-wise Quantum Phase Estimation

Algorithm 2 sequentially (or in parallel) prepares smaller blocks of phase (or counting) qubits that interact with the modular exponentiation operator acting on the work qubits. This is in contrast with the standard QPE wherein a single large phase (or counting) register is used for phase estimation.

In our proposed approach, a QPE circuit is constructed for each block. A key innovation is the use of an `exponent_offset` parameter, which we denote as κ_{offset} which is passed from Algorithm 1 to Algorithm 2. We note that this parameter is incremented in each iteration in Algorithm 1 (in the i-th iteration the increment is $m_i - t_{i+1}$, ref. line 7 in Algorithm 1). In Algorithm 2 this parameter determines the exponent of the controlled-unitary operator, $U^{2^{\kappa_j}}$, where $\kappa_j = \kappa_{\text{offset}} + j$, effectively targeting a specific window of the phase bitstring. The circuit consists of a counting register of size m_i for the current block and a target register of size n_{target} to hold the modular arithmetic state $|a^x \pmod N\rangle$. First, Hadamard gates are applied to the counting register, then for each qubit j, a controlled-unitary operator $U^{2^{\kappa_j}}$ is applied. This operator performs modular multiplication by $a \pmod N$. Following an inverse Quantum Fourier Transform (QFT^{-1}), a measurement yields a bitstring $b \in \{0,1\}^{m_i}$ that approximates the phase bits within the current window. By repeating the experiment for a number of `shots` and selecting the most frequent outcomes, a candidate set $\mathcal{S}_i$ of block-wise phase estimates is obtained. This process is repeated for each block, and the resulting candidate sets $\mathcal{S}_1, \ldots, \mathcal{S}_B$ are stored for the next step.

2.2 Carry-Aware Integration and Stitching

Because the QPE is executed block by block, the measurement outcomes must be classically stitched together to recover a full-length estimate of the phase $\varphi = s/r$. The case we are dealing with involves a superposition of multiple eignpahses. The goal is to correctly estimate any one eigenphase, which would allow us to factor the input integer N using the continued fraction algorithm. However, for each block the top candidates may come from different eigenpahses. Therefore, a naive concatenation of block outputs would lead to errors.

This situation can be visualized as follows. Imagine each eigenphase represented as a distinct color band (for instance, red, blue, green, or yellow). Each block outcome reveals only a short segment of these bands. Viewed in isolation, such a segment does not provide enough information to determine how it connects to the rest of the band. The overall task is to reconstruct a continuous band of the same color by combining the partial segments obtained from different blocks. A naive attempt to join individual segments may result in a multi-color band, i.e., where different segments might be of different colors. This is precisely where the notion of *overlap* becomes helpful: if adjacent blocks share overlapping regions, one can verify that the colors agree within the overlap. This agreement guarantees that the stitched-together segments form a consistent, full-length band corresponding to the same eigenphase.

Further, carry errors can propagate across the boundaries of overlapping blocks and must be accounted for while stitching the full phase string from the individual block estimates. To resolve this, Algorithm 3 provides a robust, carry-aware stitching routine. The algorithm integrates the blocks from right to left, a process that naturally accounts for the direction of carry propagation. For any two consecutive blocks, a left candidate s_{left} from block ℓ and a right candidate s_{right} from block $\ell+1$, a consistency check is performed. The check uses the overlap size, $t_{\ell+1}$, to compare the tail of the left bitstring with the head of the right bitstring, while also accounting for a potential carry bit, c, from the right block. This is captured by the modulo arithmetic check: $(\text{int}(\text{tail}(s_\ell, t_{\ell+1}), 2) - c) \pmod{2^{t_{\ell+1}}} = \text{int}(\text{head}(s_{\ell+1}, t_{\ell+1}), 2)$ (ref. Remarks 1 and 3). If the consistency check passes, the blocks are stitched together, removing the overlap. This iterative procedure is performed for all blocks, ultimately producing a set of complete, stitched candidates, $\mathcal{C}_{\text{stitched}}$.

2.3 Period Recovery and Factorization

Once the full phase candidates are classically reconstructed, the final phase of the algorithm uses number-theoretic techniques to recover the period and, subsequently, the non-trivial factors of N. Algorithm 4 processes each stitched phase candidate. Each stitched bitstring $\hat{b}$ encodes an approximation of the phase, $\hat{y}/2^{n_{\text{total}}} \approx s/r$, where n_{total} is the total number of effective phase qubits. The algorithm performs a classical continued fraction expansion of the fraction $\hat{y}/2^{n_{\text{total}}}$, which efficiently finds rational approximations. The denominators of these approximations, denoted by $\mathcal{Q}$, are strong candidates for the period r. Each candidate $r \in \mathcal{Q}$ is then tested to verify if it satisfies the condition $a^r \equiv 1 \pmod N$. If a valid period r is found, the final classical post-processing step from Shor's algorithm is applied. This involves checking if r is even and if $a^{r/2} \not\equiv -1 \pmod N$. If these conditions are met, the greatest common divisor (GCD) is computed for both $(a^{r/2} - 1, N)$ and $(a^{r/2} + 1, N)$. A non-trivial factor of N is returned if a GCD other than 1 or N is found.

Remark 1. ***Bit-Ordering and Qiskit Convention:*** *We adopt the Qiskit convention. The quantum registers are indexed LSB-first, where qubit $q[0]$ is the least significant bit (LSB) and $q[m-1]$ is the most significant bit (MSB). The controlled-unitary operator $C\text{-}U^{2^k}$ is applied with the control qubit $q[k]$. A measured bitstring b is obtained by reading the qubits from most significant* *to least significant, resulting in a string*

$$b = b_{m-1}\, b_{m-2}\, \ldots\, b_1\, b_0 \quad \textit{(MSB-first).}$$

For integer conversion we treat b as MSB-first and index bits by their weight*: b_j denotes the bit of weight 2^j (so b_{m-1} is the leftmost/MSB and b_0 is the rightmost/LSB). We define*

$$\mathrm{int}(b, 2) \;=\; \sum_{j=0}^{m-1} b_j\, 2^{\,j}.$$

Concatenation and slicing of bitstrings are always handled as MSB-first operations.

To summarize, registers and controlled-unitary exponents use LSB indexing; measured bitstrings are MSB-first; integer conversion treats b as MSB-first (no reversal); concatenation/slicing are MSB-first.

Remark 2. *In the standard Shor's algorithm, the target register is typically initialized in the state $|1\rangle$. For improved robustness in our approach, it is advantageous to generalize this initialization to the following state:*

$$|\psi_0\rangle = w_0\, |1\rangle + \sum_{j=1}^{L-1} w_j\, |a^{2^{\kappa_0}+j-1} \bmod N\rangle\,, \quad \sum_{j=0}^{L-1} |w_j|^2 = 1, \tag{1}$$

where $\kappa_0 =$ `exponent_offset` (ref. Algorithm 2).

This construction allows a multi-term superposition in the target register, which can enhance the efficiency of the our proposed approach. In the special case where $L = 1$ and $w_0 = 1$, this reduces to the conventional Shor's algorithm initialization, $|\psi_0\rangle = |1\rangle$.

Other forms of $|\psi_0\rangle$ may also be considered, for example,

$$|\psi_0\rangle = \sum_{j=0}^{L-1} w_j\, |a^{2^{\kappa_0}+j-1} \bmod N\rangle\,, \quad \sum_{j=0}^{L-1} |w_j|^2 = 1. \tag{2}$$

Remark 3. ***Bitstring Slicing Convention:*** *For a bitstring s of length m, the operation head(s, t) refers to the first t bits (MSB-first), while tail(s, t) refers to the last t bits.*

Algorithm 1: Modular Shor's Algorithm via Windowed QPE

```
Input  : A composite number $N$ to be factored; a coprime base $a < N$; the target register
         size $n_{\text{target}} = \lceil \log_2 N \rceil$; block sizes $\mathbf{m} = [m_1, \ldots, m_B]$; overlap size
         $\mathcal{T} = [t_1, t_2 \ldots, t_B]$, where $t_1 = 0$ and $t_i \le \min(m_{i-1}, m_i)$ for $i \ge 2$; number of
         shots shots; number of candidates to select num_selected_candidates $\ge 1$;
         maximum number of sequences to retain max_limit_combos.
Output: A non-trivial factor of $N$.
1  Initialize an empty list $\mathcal{S}$ of candidate sets;
2  Initialize exponent_offset $\leftarrow 0$;
   /* 1.  Run a QPE block for each block of phase qubits                      */
3  for $i = 1$ to $B$ do
4      $\mathcal{S}_i \leftarrow$ RunAndProcessQPEBlock($N, a, n_{\text{target}}, m_i$,
5                                  exponent_offset, shots, num_selected_candidates);
6      Append $\mathcal{S}_i$ to $\mathcal{S}$;
7      if $i < B$ then
8          exponent_offset $\leftarrow$ exponent_offset $+ m_i - t_{i+1}$;
   /* 2.  Integrate blocks and stitch candidates                              */
9  $\mathcal{C}_{\text{stitched}} \leftarrow$ CarryAwareIntegrationAndStitching($\mathcal{S}, \mathbf{m}, \mathcal{T}$, max_limit_combos);
   /* 3.  Recover period and factor N from the final candidates               */
10 $(r, \text{factor}) \leftarrow$ RecoverPeriodAndFactor($\mathcal{C}_{\text{stitched}}, a, N$);
11 if factor is found then
12     return factor;
13 return "No non-trivial factor found";
```

```
Algorithm 2: RunAndProcessQPEBlock
Input  : Integer $N$, base $a$, target register size $n_{\text{target}}$, block size $m_i$, shift power
         exponent_offset, number of shots shots, number of candidates to select
         num_selected_candidates $\geq 1$.
Output: A candidate set $\mathcal{S}$ consisting of bit strings of size $m_i$.
   /* 1.  Construct and execute the quantum circuit                        */
 1 Initialize quantum registers: a counting register $|q_C\rangle = |0\rangle^{\otimes m_i}$ and a target register
    $|q_T\rangle = |0\rangle^{\otimes n_{\text{target}}}$;
 2 Initialize the target register to the state $|\psi_0\rangle \pmod{N}$ (ref. remark 2);
 3 Apply Hadamard gates to the counting register: $|q_C\rangle \leftarrow H^{\otimes m_i} |0\rangle^{\otimes m_i}$;
   /* Apply controlled unitaries for LSB-first qubits (j=0 is LSB)         */
 4 for $j = 0$ to $m_i - 1$ do
 5     Apply the controlled-unitary operator $U^{2^{\texttt{exponent\_offset}+j}}$ to the target register, with
        $|q_C\rangle [j]$ as the control qubit;
 6 Apply the inverse Quantum Fourier Transform ($\text{QFT}^{-1}$) to the counting register:
    $|q_C\rangle \leftarrow \text{QFT}_{m_i}^{-1} |q_C\rangle$;
 7 Measure the counting register in the computational basis to obtain a bitstring $b$;
   /* 2.  Process measurement outcomes                                     */
 8 Execute the circuit for shots total runs, recording the frequency of each outcome;
   /* 3.  Select and return candidate phases                               */
 9 Select the top num_selected_candidates bitstrings based on their frequencies;
10 return the set of these bitstrings;
```

```
Algorithm 3: CarryAwareIntegrationAndStitching
Input  : A list of per-block candidate sets $\mathcal{S} = [\mathcal{S}_1, \dots, \mathcal{S}_B]$ where $\mathcal{S}_i$ contains $m_i$-bit
         strings (MSB-first); block sizes $\mathbf{m} = [m_1, \dots, m_B]$;  overlap size
         $\mathcal{T} = [t_1, t_2 \dots, t_B]$, where $t_1 = 0$ and $t_i \le \min(m_{i-1}, m_i)$ for $i \ge 2$; maximum
         number of sequences to retain max_limit_combos.
Output: A set of unique stitched candidates $\mathcal{C}_{\text{stitched}} = \{(\hat{b}, \hat{y}, \hat{\phi})\}$.
   /* 1.  Integrate blocks from right to left                                        */
1  Initialize the set of integrated sequences $\mathcal{R} \leftarrow \{(s_B) \mid s_B \in \mathcal{S}_B\}$;
2  for $\ell = B - 1$ to 1 do
3      Initialize a new set of sequences $\mathcal{R}_{\text{new}} \leftarrow \emptyset$;
4      foreach sequence $(s_{right}, \dots) \in \mathcal{R}$ do
           /* Let $s_{\text{right}}$ be the right block and $s_{\text{left}}$ the left block                */
5          foreach candidate $s_{left} \in \mathcal{S}_\ell$ do
               /* Determine the carry bit from the right block                           */
6              Let $t = t_{\ell+1}$, and let $c$ be the bit at position $t$ of $s_{\text{right}}$ (from the  left, 0-indexed);
               /* The condition for consistency is simplified when the overlap is
                  zero                                                                 */
7              if $t = 0$ or $(\text{int}(tail(s_{left}, t), 2) - c) \pmod{2^t} = \text{int}(head(s_{right}, t), 2)$ then
8                  Prepend $s_{\text{left}}$ to the current sequence: $(s_{\text{left}}, s_{\text{right}}, \dots)$;
9                  Add this new sequence to $\mathcal{R}_{\text{new}}$;
10                 if $|\mathcal{R}_{new}| \ge$ max_limit_combos then
11                     break;
12         if $|\mathcal{R}_{new}| \ge$ max_limit_combos then
13             break;
14     $\mathcal{R} \leftarrow \mathcal{R}_{\text{new}}$;
15     if $\mathcal{R}$ is empty then
16         break;
   /* 2.  Stitch the final sequences                                                 */
17 Initialize a set of unique stitched candidates $\mathcal{C}_{\text{stitched}} \leftarrow \emptyset$;
18 foreach integrated sequence $(s_1, \dots, s_B) \in \mathcal{R}$ do
19     Compute $n_{\text{total}} \leftarrow \sum_{i=1}^{B} (m_i - t_i)$;
       /* Concatenate the non-overlapping heads of each block from left to
          right.                                                                     */
20     Concatenate with overlap removal:
        $\hat{b} \leftarrow \text{head}(s_1, m_1 - t_1)|\text{head}(s_2, m_2 - t_2)| \cdots |\text{head}(s_B, m_B - t_B)$;
       /* 'int' converts a bitstring to its integer representation based on the
          MSB-first convention.                                                      */
21     Convert to integer and phase: $\hat{y} \leftarrow \text{int}(\hat{b}, 2)$, and $\hat{\phi} \leftarrow \hat{y}/2^{n_{\text{total}}}$;
22     Add the unique tuple $(\hat{b}, \hat{y}, \hat{\phi})$ to $\mathcal{C}_{\text{stitched}}$;
23 return $\mathcal{C}_{stitched}$;
```

Algorithm 4: RecoverPeriodAndFactor

```
Input  : Stitched candidates $\mathcal{C}_{\text{stitched}}$; base $a$ coprime to $N$.
Output: A tuple $(r, \text{factor})$ where factor is a non-trivial factor of $N$, or $(\texttt{None}, \texttt{None})$ if no
         factor is found.
/* Per-Candidate Period Estimates via Continued Fractions                    */
1  foreach $(\hat{b}, \hat{y}, \hat{\phi}) \in \mathcal{C}_{stitched}$ do
2      Let $n_{\text{current\_total}} \leftarrow |\hat{b}|$;
3      Compute denominators $\mathcal{Q}$ from the convergents of $\hat{y}/2^{n_{\text{current\_total}}}$ via continued
        fractions, with $q \le N$. ;
4      foreach $r \in \mathcal{Q}$ do
5          if $r \le 0$ or $r > N$ then
6              continue;
           /* Directly check if this is the period                             */
7          if $a^r \equiv 1 \pmod{N}$ then
               /* Classical Post-Processing for Factors                        */
8              if $r$ is even and $a^{r/2} \not\equiv -1 \pmod{N}$ then
9                  $f_1 \leftarrow \gcd(a^{r/2} - 1, N)$;
10                 $f_2 \leftarrow \gcd(a^{r/2} + 1, N)$;
11                 if $f_1 \ne 1$ and $f_1 \ne N$ then
12                     return $(r, f_1)$;
13                 if $f_2 \ne 1$ and $f_2 \ne N$ then
14                     return $(r, f_2)$;
15 return $(\texttt{None}, \texttt{None})$;
```

2.4 Rationale for Using a Non-Standard Initial Target State

In the standard formulation of Shor's algorithm, the target register is initialized in the computational basis state $|1\rangle$. This initialization leads to a highly uniform probability distribution in the measured phase register.

To improve robustness and spectral diversity in our adaptive and windowed QPE framework, as noted in Remark 2, we instead prepare a more general superposition in the target register:

$$|\psi_0\rangle = w_0 |1\rangle + \sum_{j=1}^{L-1} w_j \, |a^{2^{\kappa_0}+j-1} \bmod N\rangle, \qquad \sum_{j=0}^{L-1} |w_j|^2 = 1,$$

where $\kappa_0 = \texttt{exponent\_offset}$ corresponds to the block's modular exponentiation offset in the overall QPE procedure. The coefficients $\{w_j\}$ may be chosen to create structured interference patterns in the phase domain, leading to sharper or shifted spectral peaks that can enhance distinguishability between valid phase estimates.

Even when $L = 1$ (i.e. when the target register is initialized in the computational basis state $|1\rangle$), using a shifted exponent offset (that is, $|\psi_0\rangle = |a^{2^{\kappa_0}} \bmod N\rangle$) already induces a small deviation from uniformity in the measured phase distribution, since the initial target state is no longer a fixed eigenstate of the modular exponentiation operator across all blocks. However, for $L > 1$, the superposed form of Equation (1) significantly enhances this effect by embedding partial periodic information before the QPE step.

Connection to the eigenbasis. Let $\{|\psi_k\rangle\}_{k=0}^{r-1}$ denote the eigenstates of U_a, satisfying $U_a |\psi_k\rangle = e^{2\pi ik/r} |\psi_k\rangle$, where $|\psi_k\rangle = \frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} e^{-2\pi iks/r} |a^s \bmod N\rangle$. For any state $|A\rangle = \sum_{s=0}^{r-1} w_s |a^s \bmod N\rangle$, the eigen-basis amplitudes are $c_k = \langle\psi_k|A\rangle = \frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} w_s e^{2\pi iks/r}$. The resulting spectral coefficients $\{c_k\}$ determine how the generalized target state populates different eigenfrequencies of U_a, and thus how the measured phase distribution deviates from the uniform pattern of the conventional case. Importantly, this modification does not alter the underlying eigenphases $2\pi k/r$, but only alters their relative probabilities in the corresponding measurement outcomes.

Example ($L = 2$). For a simple two-term initialization

$$|\psi_0\rangle = \frac{1}{\sqrt{2}}\big(|1\rangle + |a\rangle \big),$$

we have $w_0 = w_1 = 1/\sqrt{2}$ and $w_{s\geq 2} = 0$. Using the above formulation, the decomposition becomes

$$|\psi_0\rangle = \sum_{k=0}^{r-1} \frac{1}{\sqrt{2r}}\big(1 + e^{2\pi ik/r}\big) |\psi_k\rangle ,$$

yielding spectral weights

$$|c_k|^2 = \frac{2}{r} \cos^2\left(\frac{\pi k}{r}\right).$$

Thus, the modified initialization redistributes amplitude non-uniformly across the eigenstates of U_a, producing a distinct, structured pattern in the observed phase histogram. This structured modulation remains robust (across blocks) and enhances correlation between blocks (with overlap) and supports partial-block stitching under limited resolution, making it particularly advantageous in our windowed QPE approach.

Remark 4. *Further improvements to robustness in the overall order-finding process can be achieved by combining multiple phase candidates arising from different initialization choices or independent runs using the least common multiple (LCM) of their corresponding period estimates. Since the initialization in Equation* (1) *affects only the probability distribution and not the eigenphases themselves, the valid period r remains consistent across such runs. Taking the LCM of independently obtained candidate periods amplifies stability and helps suppress spurious estimates introduced by measurement noise or imperfect block stitching.*

3 Numerical Examples

To illustrate our proposed modular approach to Shor's algorithm (Algorithm 1), we present several numerical examples. These small-scale demonstrations verify the correctness of the approach and highlight how varying block sizes and overlaps affect the phase-estimation procedure, resource requirements, and the subsequent recovery of non-trivial factors.

For all the examples considered in this section we assume $L = 1$ and $|\psi_0\rangle = |1\rangle$, unless otherwise specified (ref. Remark 2). We begin with the case $N = 15$, and then consider alternative block configurations and a larger composite, $N = 221$.

3.1 Numerical Example: Quantum Factoring of $N = 15$

We select a random base $a = 2$, for which the modular order for $N = 15$ is known to be $r = 4$. A standard QPE circuit requires at least $2\lceil\log_2 N\rceil + 1 = 9$ qubits to guarantee a correct result.

Here, we show a successful run using a windowed QPE approach with overlapping blocks and a reduced total effective phase-register length per block. For this run, the parameters for the stitching procedure were set as follows: `max_limit_combos` = 2 and `num_selected_candidates` = 2.

Quantum Phase Estimation with Overlapping Blocks. The QPE is executed in four sequential blocks, denoted as $B = 4$. The block sizes are defined by the vector $\mathbf{m} = [m_1, m_2, m_3, m_4] = [3, 4, 4, 5]$, while the corresponding overlaps are given by $\mathcal{T} = [t_1, t_2, t_3, t_4] = [0, 2, 3, 2]$. For each block, a circuit is run to measure the most significant qubits of the phase register. Table 1 summarizes the most frequent outcomes, with probabilities derived empirically from simulation shots.

Table 1: Most frequent candidates measured during the run for $N = 15$ with $a = 2$.

Block	Block Size (m_i)	Top Candidates (bitstring: empirical probability)
1	3	`110` (29.0%), `010` (27.0%)
2	4	`1000` (51.0%), `0000` (49.0%)
3	4	`0000` (100.0%)
4	5	`00000` (100.0%)

Quantum Circuit Diagrams for Each Block. Figure 1 shows the quantum circuit for each block of the windowed QPE run (ref. Algorithms 1 and 2).

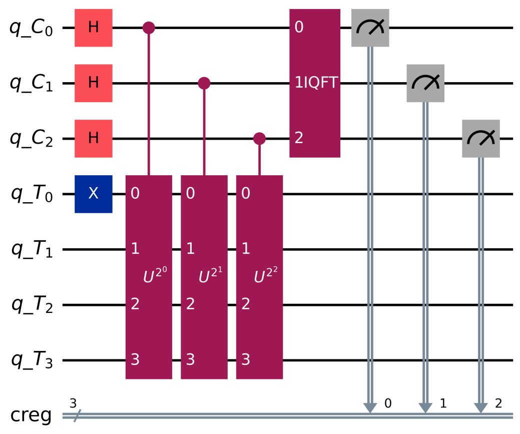
Carry-Aware Integration and Stitching. Following the quantum measurements, a classical stitching procedure reconstructs the full-length phase bitstring (ref. Algorithm 3). This process operates from right-to-left, beginning with the least significant bits (LSBs) measured in the final QPE block and progressing towards the most significant bits (MSBs). The total length of the stitched bitstring is 9 bits, a sum of the non-overlapping segments from each block:

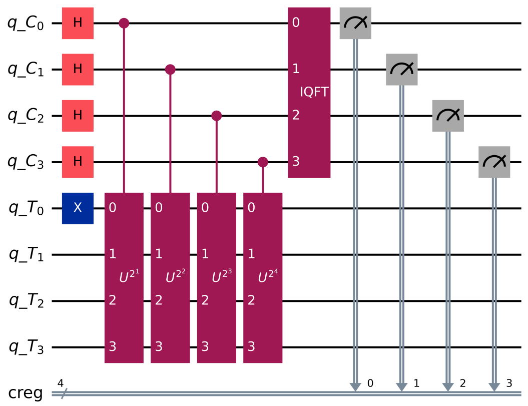
$$(m_1 - t_1) + (m_2 - t_2) + (m_3 - t_3) + (m_4 - t_4) = (3 - 0) + (4 - 2) + (4 - 3) + (5 - 2) = 3 + 2 + 1 + 3 = 9.$$

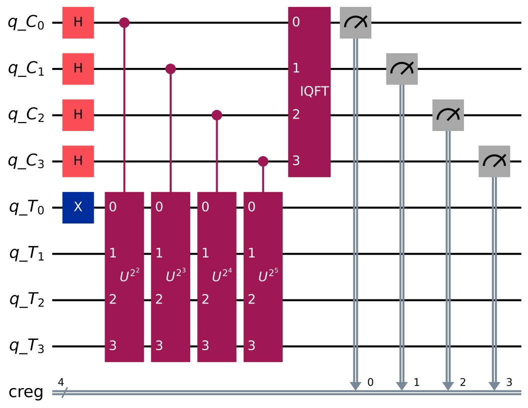
Using the most probable outcome from each block, we obtain the stitched bitstring `110000000`. The process initiates with the non-overlapping portion of Block 4's outcome, taking the final $m_4 - t_4 = 3$ bits (`000`). Next, the non-overlapping portion of Block 3's outcome, the final $m_3 - t_3 = 1$ bit, is prepended (`0000`). We then prepend the non-overlapping portion of Block 2's outcome, the final $m_2 - t_2 = 2$ bits (`000000`). Finally, the non-overlapping portion of Block 1's outcome, the final $m_1 - t_1 = 3$ bits, is prepended to complete the bitstring (`110000000`). This concatenation yields the full stitched bitstring, which corresponds to two unique phase candidates. The first, derived from the second most probable outcome of the initial block, is the bitstring `010000000`, which represents the integer $\hat{y} = \texttt{010000000}_2 = 128$ and a phase of $\hat{\phi} = \frac{\hat{y}}{2^9} = \frac{1}{4}$. The second candidate, derived from the most probable outcomes, is the bitstring `110000000`, which corresponds to the integer $\hat{y} = \texttt{110000000}_2 = 384$ and a phase of $\hat{\phi} = \frac{\hat{y}}{2^9} = \frac{3}{4}$.

Period Recovery and Factorization. The final stage employs the classical continued fraction algorithm on the observed phase, $\hat{\phi} = 3/4$ (ref. Algorithm 4). This yields the most likely denominator $r = 4$, which is confirmed to be the true period of 2 modulo 15. The factorization is then completed by computing the greatest common divisor (gcd):

$$\gcd\left(a^{r/2} \pm 1, N\right) = \gcd\left(2^{4/2} \pm 1, 15\right) = \gcd\left(2^2 \pm 1, 15\right) \Rightarrow \{3, 5\}.$$


(a) Quantum circuit for Block 1 with block size $m_1 = 3$.


(b) Quantum circuit for Block 2 with block size $m_2 = 4$.


(c) Quantum circuit for Block 3 with block size $m_3 = 4$.

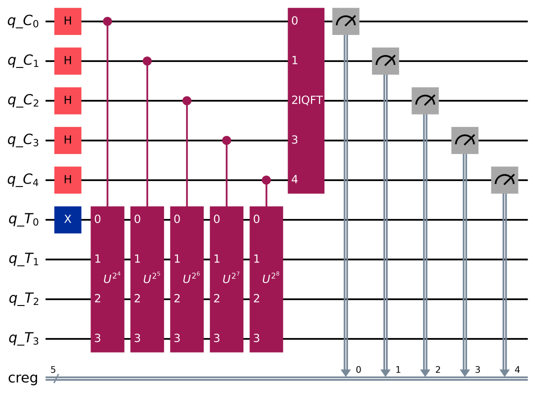

(d) Quantum circuit for Block 4 with block size $m_4 = 5$.

Figure 1: Quantum circuits for each of the four blocks in the windowed QPE run for factoring $N = 15$.

The algorithm successfully returns the non-trivial factors 3 and 5.

3.2 Numerical Example: Quantum Factoring of $N = 15$ with Fewer Qubits

A standard QPE circuit requires at least $2\lceil \log_2 N \rceil + 1 = 9$ qubits to guarantee a correct result. However, in some special cases, successful factorization can be achieved using fewer number of qubits. Here, we show a successful run using a windowed QPE approach with a reduced total effective phase-register length of 5 qubits, achieved using only two measurement blocks. For this run, the following parameters were used: $a = 2$, `max_limit_combos` $= 2$ and `num_selected_candidates` $= 2$.

Quantum Phase Estimation with Overlapping Blocks. The QPE is executed in two independent blocks, denoted as $B = 2$. The block sizes are defined by the vector $\mathbf{m} = [3, 4]$, while the corresponding overlaps are given by $\mathcal{T} = [0, 2]$. For each block, a circuit is run to measure the most significant qubits of the phase register. Table 2 summarizes the most frequent outcomes, with probabilities derived empirically from simulation shots.

Table 2: Most frequent candidates measured during the run for $N = 15$ with $a = 2$.

Block	Block Size (m_i)	Top Candidates (bitstring: empirical probability)
1	3	`100` (27.0%), `110` (25.0%)
2	4	`1000` (51.0%), `0000` (49.0%)

Carry-Aware Integration and Stitching. In this case, the total length of the stitched bitstring is 5 bits, a sum of the non-overlapping segments from each block:

$$\text{Length} = (m_1 - t_1) + (m_2 - t_2) = (3 - 0) + (4 - 2) = 3 + 2 = 5.$$

Using the most probable outcome from each block, we reconstruct the stitched bitstring. The process initiates with the non-overlapping portion of Block 2's outcome, taking the final $m_2 - t_2 = 2$ bits (`00`). We then prepend the non-overlapping portion of Block 1's outcome, the final $m_1 - t_1 = 3$ bits, to complete the bitstring (`11000`). This concatenation yields the full stitched bitstring, which corresponds to two unique phase candidates. The first, derived from the second most probable outcome of the initial block, is the bitstring `10000`, which corresponds to the integer 16 and a phase of $\frac{1}{2}$. The second candidate, derived from the most probable outcomes, is the bitstring `11000`, which represents the integer 24 and a phase of $\frac{3}{4}$.

Period Recovery and Factorization. The final stage uses the classical continued fraction algorithm on the observed phase, $\hat{\phi} = 3/4$. This gives the most likely denominator $r = 4$, which is confirmed to be the true period of 2 modulo 15. The factorization is then completed by computing the greatest common divisor (gcd):

$$\gcd\left(a^{r/2} \pm 1, N\right) = \gcd\left(2^{4/2} \pm 1, 15\right) = \gcd\left(2^2 \pm 1, 15\right) \Rightarrow \{3, 5\}.$$

The algorithm successfully returns the non-trivial factors 3 and 5.

3.3 Numerical Example: Quantum Factoring of $N = 15$ with Zero Overlap

Here, we show an implementation using a windowed QPE approach with zero-overlap and a reduced total effective phase-register length of 6 qubits. We select a random base $a = 2$ for this demonstration. Further, the parameters for the stitching procedure were set as follows: `max_limit_combos` $=$ 4 and `num_selected_candidates` $= 4$. It is important to note that the overlap vector is configured as all zeros, which disables the algorithm's built-in error mitigation and redundancy checks.

This example, with a small number of blocks, shows that a zero-overlap configuration can still produce the correct result. However, in cases with many blocks, a large number of candidates would be produced, leading to significant classical overhead. The total number of combinations to check grows exponentially with the number of blocks. This approach may still be preferable if the classical computing requirements are manageable for a given problem instance.

Quantum Phase Estimation with Non-overlapping Blocks. The QPE is executed in two sequential blocks, denoted as $B = 2$. The block sizes are defined by the vector $\mathbf{m} = [3, 3]$, while the corresponding overlaps are given by $\mathcal{T} = [0, 0]$ (i.e. the blocks do not overlap). For each block, a circuit is run to measure the most significant qubits of the phase register. Table 3 summarizes the most frequent outcomes, with probabilities derived empirically from simulation shots.

Table 3: Most frequent candidates measured during the run for $N = 15$ with $a = 2$.

Block	Block Size (m_i)	Top Candidates (bitstring: empirical probability)
1	3	`100` (29.0%), `110` (29.0%), `000` (24.0%), `010` (18.0%)
2	3	`000` (100.0%)

Concatenation and Phase Estimation. Given the lack of overlap, the classical stitching procedure simplifies to a direct concatenation of the most probable bitstrings from each block. This process does not allow for error correction between blocks. The total length of the stitched bitstring is 6 bits, a direct sum of the block sizes:

$$\text{Length} = m_1 + m_2 = 3 + 3 = 6.$$

This concatenation yields four unique phase candidates. The most probable outcome, derived from the most probable bitstrings of each block, is the bitstring `100000`, which corresponds to the integer 32 and a phase of $\frac{1}{2}$. Other stitched candidates include `000000` (integer 0, phase 0), `010000` (integer 16, phase $\frac{1}{4}$), and `110000` (integer 48, phase $\frac{3}{4}$). The measured phase that yields the correct period is $\hat{\phi} = \frac{3}{4}$, which corresponds to the stitched bitstring `110000` from the second-most probable outcome of Block 1.

Period Recovery and Factorization. The final stage employs the classical continued fraction algorithm on the observed phase, $\hat{\phi} = 3/4$. This gives the most likely denominator $r = 4$, which is confirmed to be the true period of 2 modulo 15. The factorization is then completed by computing the greatest common divisor (gcd):

$$\gcd\left(a^{r/2} \pm 1, N\right) = \gcd\left(2^{4/2} \pm 1, 15\right) = \gcd\left(2^2 \pm 1, 15\right) \Rightarrow \{3, 5\}.$$

The algorithm successfully returns the non-trivial factors 3 and 5.

3.4 Numerical Example: Quantum Factoring of $N = 221$

Next we illustrate our approach by considering the example of factoring $N = 221$. We select a random base $a = 12$, for which the modular order is known to be $r = 16$. A standard QPE circuit requires at least $2\lceil \log_2 N \rceil + 1 = 17$ qubits to guarantee a correct result. Here, we show a successful run using a windowed QPE approach with a reduced total effective phase-register length of 7 qubits. For this run, the parameters for the stitching procedure were set as follows: `max_limit_combos` = 4 and `num_selected_candidates` = 4.

Quantum Phase Estimation with Overlapping Blocks. The QPE is executed in four sequential blocks, denoted as $B = 4$. The block sizes are defined by the vector $\mathbf{m} = [3, 3, 4, 3]$, while the corresponding overlaps are given by $\mathcal{T} = [0, 2, 2, 2]$. For each block, a circuit is run to measure the most significant qubits of the phase register. Table 4 summarizes the most frequent outcomes, with probabilities derived empirically from simulation shots.

Table 4: Most frequent candidates measured during the run for $N = 221$ with $a = 12$.

Block	Block Size (m_i)	Top Candidates (bitstring: empirical probability)
1	3	`010` (18.0%), `000` (18.0%), `111` (16.0%), `001` (14.0%)
2	3	`011` (18.0%), `001` (17.0%), `000` (16.0%), `111` (16.0%)
3	4	`0100` (33.0%), `1100` (26.0%), `0000` (24.0%), `1000` (17.0%)
4	3	`000` (100.0%)

Quantum Circuit Diagrams for Each Block. Figure 2 shows the quantum circuit for each block of the windowed QPE run for $N = 221$. (ref. Algorithms 1 and 2).

Carry-Aware Integration and Stitching. In this case, the total length of the stitched bitstring is 7 bits, a sum of the non-overlapping segments from each block:

Length $= (m_1-t_1)+(m_2-t_2)+(m_3-t_3)+(m_4-t_4) = (3-0)+(3-2)+(4-2)+(3-2) = 3+1+2+1 = 7.$

Four unique phase candidates were obtained through the carry-aware integration and stitching algorithm using the most probable outcomes from each block. The optimal candidate derived from this process is the bitstring `0111000`, which corresponds to the integer 56 and a phase of $\frac{7}{16}$. Other stitched candidates include `0000000` (integer 0, phase 0), `0001000` (integer 8, phase $\frac{1}{16}$), and `0011000` (integer 24, phase $\frac{3}{16}$).

Period Recovery and Factorization. As before, the final stage uses the classical continued fraction algorithm on the observed phase, $\hat{\phi} = 7/16$. This gives the most likely denominator $r = 16$, which is confirmed to be the true period of 12 modulo 221. The factorization is then completed by computing the greatest common divisor (gcd):

$$\gcd\left(a^{r/2} \pm 1, N\right) = \gcd\left(12^{16/2} \pm 1, 221\right) = \gcd\left(12^{8} \pm 1, 221\right) \Rightarrow \{13, 17\}.$$

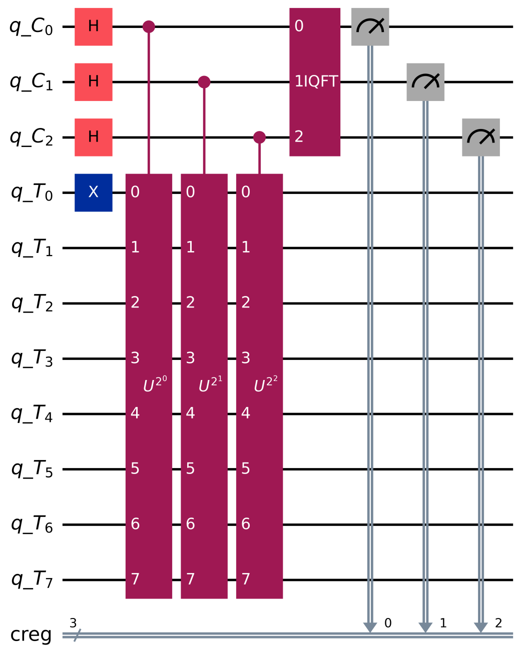
The algorithm successfully returns the non-trivial factors 13 and 17.

4 Numerical Examples: Cases with Non-Standard Initial Target State $|\psi_0\rangle$

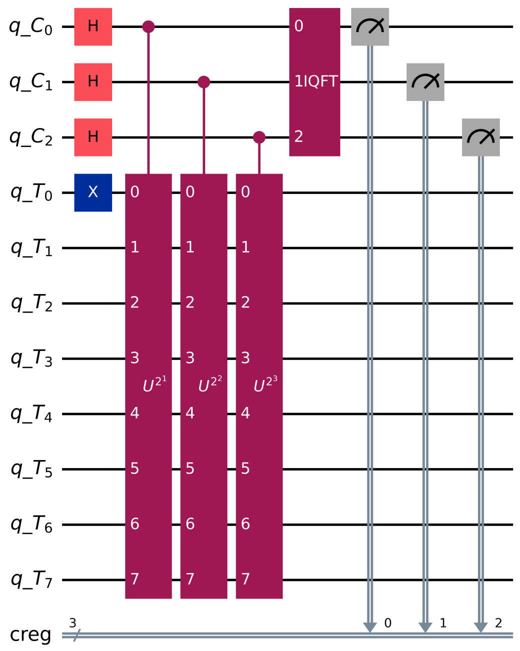
In this section, we consider numerical examples with non-standard initial target states of the form $|\psi_0\rangle$ as described in Equation (1) in Remark 2, with $L \geq 2$.

4.1 Numerical Example: Factoring $N = 161$

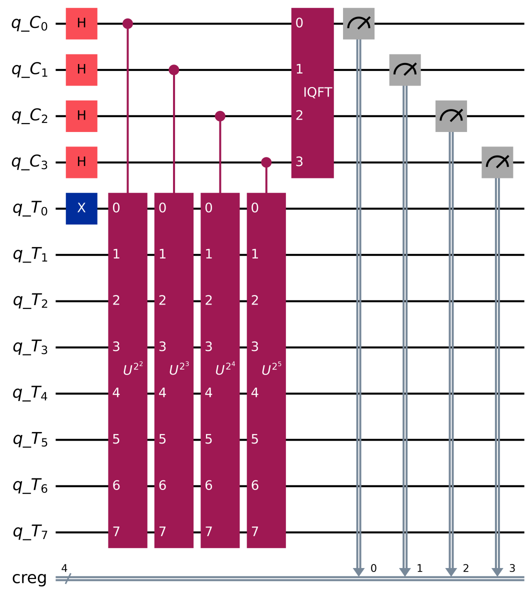
In this numerical example we consider factoring the integer $N = 161$ with the base $a = 3$. The algorithm was configured with the following parameters: The required target qubits (n_{target}) is 8 ($\lceil \log_2(161) \rceil$), blocks allocation $\mathbf{m} = [3, 4, 4, 4, 4, 4, 4, 4]$, overlaps $\mathcal{T} = [0, 2, 2, 2, 2, 2, 2, 2]$ and `num_selected_candidates` = 4. The initial target state $|\psi_0\rangle$ was created as a uniform superposition of $L = 4$ computational basis states (ref. Equation (1)). This means the initial target state for each block was set $|\psi_0\rangle = w_0 |1\rangle + \sum_{j=1}^{3} w_j |a^{2^{\kappa}+j-1} \bmod N\rangle$, with equal weights $w_j = \frac{1}{2}$. The output



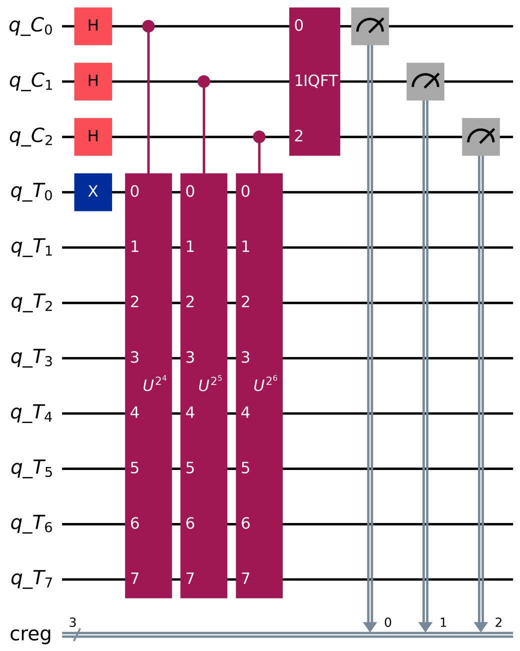
(a) Quantum circuit for Block 1 with block size $m_1 = 3$.



(b) Quantum circuit for Block 2 with block size $m_2 = 3$.



(c) Quantum circuit for Block 3 with block size $m_3 = 4$.



(d) Quantum circuit for Block 4 with block size $m_4 = 3$.

Figure 2: Quantum circuits for each of the four blocks in the windowed QPE run for factoring $N = 221$.

Table 5: Reconstruction of initial target state $|\psi_0\rangle$ and top measured candidates for each QPE block ($N = 161$, $a = 3$).

Block	Initial State $|\psi_0\rangle$	Top Candidate Bitstrings (Empirical Probability)
1	$\kappa = 0$: $\frac{1}{2}(|1\rangle + |3\rangle + |9\rangle + |27\rangle)$	`000` (42.2%), `001` (21.4%), `111` (21.3%), `101` (3.7%)
2	$\kappa = 1$: $\frac{1}{2}(|1\rangle + |9\rangle + |27\rangle + |81\rangle)$	`0000` (14.8%), `0001` (13.6%), `1111` (13.6%), `1110` (10.4%)
3	$\kappa = 3$: $\frac{1}{2}(|1\rangle + |121\rangle + |41\rangle + |123\rangle)$	`0000` (12.5%), `0010` (11.2%), `1110` (11.1%), `1100` (7.8%)
4	$\kappa = 5$: $\frac{1}{2}(|1\rangle + |100\rangle + |139\rangle + |95\rangle)$	`0000` (14.9%), `1111` (13.6%), `0001` (13.6%), `1110` (10.4%)
5	$\kappa = 7$: $\frac{1}{2}(|1\rangle + |2\rangle + |6\rangle + |18\rangle)$	`0000` (9.2%), `0001` (9.0%), `1111` (8.9%), `0010` (8.3%)
6	$\kappa = 9$: $\frac{1}{2}(|1\rangle + |16\rangle + |48\rangle + |144\rangle)$	`0000` (13.7%), `0011` (9.3%), `1101` (9.3%), `0100` (8.6%)
7	$\kappa = 11$: $\frac{1}{2}(|1\rangle + |9\rangle + |27\rangle + |81\rangle)$	`0000` (14.8%), `0001` (13.6%), `1111` (13.6%), `1110` (10.4%)
8	$\kappa = 13$: $\frac{1}{2}(|1\rangle + |121\rangle + |41\rangle + |123\rangle)$	`0000` (12.5%), `1110` (11.1%), `0010` (11.1%), `1100` (7.8%)

from each block provided candidate bitstrings and their empirical probabilities, which are shown in Table 5.

The collected bitstrings from all QPE blocks were successfully combined using a carry-aware stitching process, yielding a full-length 17-bit candidate, `01111000001111100`. This bitstring corresponds to the decimal value 61,564, which provides the fractional phase $\phi = \frac{61{,}564}{2^{17}} = \frac{15{,}391}{32{,}768}$. Applying the classical continued fraction algorithm to this phase successfully recovered the period $r = 66$. Using this period, the final classical step involved computing $\gcd(a^{r/2} \pm 1, N) = \gcd(3^{33} \pm 1, 161)$, which successfully yielded the non-trivial factor 23.

4.2 Numerical Example: Factoring $N = 295{,}627$

To demonstrate the behavior and performance of the modular, windowed quantum phase estimation procedure, we consider the integer $N = 296,867$, and select the base $a = 3$, $\mathbf{m} = [4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4]$ and $\mathcal{T} = [0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]$. Further, for this example, `num_selected_candidates` was set to $= 8$ and the initial taeget state for each block was set using the uniform weights $w_j = \frac{1}{\sqrt{L}}$ with $L = 500$ (ref. Equation (1)). The output from each block with the candidate bitstrings and their empirical probabilities, which are shown in Table 6.

Table 6: Most frequent candidates measured during the run for $N = 295{,}627$ with $a = 3$.

Block	Top Candidates (bitstring: empirical probability)
1	`0000` (99.0%), `1111` (0.3%), `0001` (0.3%), `0010` (0.1%), `1110` (0.1%), `0011` (0.0%), `1101` (0.0%), `1100` (0.0%)
2	`0000` (95.8%), `0001` (1.3%), `1111` (1.3%), `0010` (0.3%), `1110` (0.3%), `0011` (0.2%), `1101` (0.2%), `1100` (0.1%)
3	`0000` (83.0%), `0001` (5.2%), `1111` (5.2%), `0010` (1.4%), `1110` (1.4%), `1101` (0.7%), `0011` (0.6%), `0100` (0.4%)
4	`0000` (41.0%), `0001` (20.8%), `1111` (20.7%), `0010` (2.8%), `1110` (2.8%), `0011` (2.5%), `1101` (2.5%), `0101` (1.1%)
5	`0000` (27.2%), `1110` (19.4%), `0010` (19.4%), `0100` (7.4%), `1100` (7.3%), `0101` (3.2%), `1011` (3.2%), `0011` (2.7%)
6	`0000` (27.9%), `0111` (15.4%), `1001` (15.4%), `1000` (7.4%), `0001` (7.0%), `1111` (7.0%), `0110` (3.1%), `1010` (3.1%)
7	`0000` (29.6%), `1101` (16.1%), `0011` (16.0%), `1011` (7.4%), `0101` (7.3%), `1000` (4.5%), `1110` (4.1%), `0010` (4.1%)
8	`0000` (32.1%), `0101` (22.3%), `1011` (22.3%), `1010` (6.4%), `0110` (6.4%), `1100` (1.7%), `0100` (1.7%), `1001` (1.3%)
9	`0000` (28.7%), `1011` (21.1%), `0101` (21.1%), `0110` (8.0%), `1010` (7.9%), `1111` (2.0%), `0001` (2.0%), `0100` (1.6%)
10	`0000` (27.5%), `1011` (19.4%), `0101` (19.2%), `0110` (6.2%), `1010` (6.1%), `1000` (3.3%), `0111` (3.2%), `1001` (3.2%)
11	`0000` (29.2%), `0011` (20.6%), `1101` (20.4%), `1010` (7.9%), `0110` (7.8%), `1009` (2.9%), `0111` (2.9%), `1100` (1.4%)
12	`0000` (27.4%), `0011` (14.7%), `1101` (14.6%), `1001` (7.3%), `0111` (7.3%), `1100` (5.1%), `0100` (5.1%), `1010` (3.8%)
13	`0000` (29.4%), `0011` (15.7%), `1101` (15.6%), `1011` (7.9%), `0101` (7.8%), `1000` (4.6%), `1110` (4.4%), `0010` (4.3%)

The final phase of of our Algorithm involved the stitching and classical post-processing of the collected QPE results. The successful candidate was the string `0011000001000011111001111110`, which corresponds to the integer value 50,609,790. This yielded the fractional phase $\phi = \frac{50{,}609{,}790}{2^{28}} = \frac{25{,}304{,}895}{134{,}217{,}728}$. Applying the classical continued fraction algorithm to this phase recovered a multiple of the period as 5,670 (where the true period r is 1,890). Using this value in the final classical step, a non-trivial factor (541) of $N = 296{,}867$ was successfully found.

4.3 Numerical Example: Factoring 1,461,617

Table 7: Most frequent candidates measured during the run for $N = 1{,}461{,}617$ with $a = 8$.

Block	Top Candidates (bitstring: empirical probability)
1	`0000` (94.7%), `1111` (1.7%), `0001` (1.6%), `1110` (0.4%), `0010` (0.4%), `1101` (0.2%), `0011` (0.2%), `1100` (0.1%)
2	`0000` (78.8%), `1111` (6.6%), `0001` (6.5%), `1110` (1.7%), `0010` (1.6%), `1101` (0.8%), `0011` (0.8%), `1100` (0.5%)
3	`0000` (34.2%), `0001` (21.8%), `1111` (21.7%), `0010` (4.9%), `1110` (4.9%), `1100` (1.9%), `0100` (1.8%), `1101` (1.7%)
4	`0000` (10.6%), `0001` (10.2%), `1111` (10.2%), `1110` (9.2%), `0010` (9.2%), `0011` (7.9%), `1101` (7.8%), `1100` (6.2%)
5	`0000` (6.8%), `1111` (6.8%), `0001` (6.7%), `0010` (6.7%), `1110` (6.6%), `0011` (6.5%), `1101` (6.5%), `0100` (6.3%)
6	`1110` (6.4%), `1111` (6.4%), `0011` (6.4%), `0010` (6.3%), `0001` (6.3%), `0000` (6.3%), `1101` (6.3%), `1100` (6.3%)
7	`0000` (7.6%), `0101` (7.3%), `1011` (7.3%), `1010` (7.0%), `0110` (7.0%), `0001` (6.8%), `1111` (6.7%), `1100` (6.3%)
8	`1110` (6.4%), `0000` (6.4%), `1111` (6.4%), `1101` (6.4%), `0001` (6.3%), `0010` (6.3%), `0011` (6.3%), `0101` (6.3%)
9	`0000` (10.4%), `0100` (10.4%), `1100` (10.4%), `1000` (10.0%), `0001` (6.5%), `0101` (6.4%), `1111` (6.3%), `1011` (6.3%)
10	`0000` (11.2%), `1111` (11.0%), `0001` (11.0%), `1110` (9.9%), `0010` (9.9%), `1101` (8.2%), `0011` (8.2%), `0100` (6.3%)
11	`1111` (6.5%), `0000` (6.4%), `0001` (6.3%), `1110` (6.3%), `0010` (6.3%), `1100` (6.2%), `1101` (6.2%), `0011` (6.2%)
12	`1111` (6.5%), `0001` (6.4%), `0000` (6.3%), `0010` (6.3%), `0100` (6.3%), `0011` (6.3%), `1100` (6.2%), `1110` (6.2%)
13	`1110` (6.4%), `0001` (6.4%), `0000` (6.4%), `1111` (6.3%), `1100` (6.3%), `0011` (6.3%), `1011` (6.3%), `1101` (6.2%)
14	`1111` (6.4%), `0010` (6.4%), `0001` (6.4%), `1011` (6.3%), `0000` (6.3%), `0100` (6.3%), `1100` (6.3%), `0011` (6.3%)
15	`0000` (9.6%), `1101` (9.3%), `0011` (9.2%), `0110` (8.4%), `1010` (8.4%), `0111` (7.4%), `1001` (7.4%), `1100` (6.4%)
16	`0000` (7.9%), `0011` (7.6%), `1101` (7.5%), `1010` (7.1%), `0110` (7.0%), `1001` (6.6%), `0111` (6.6%), `0100` (6.3%)

Next, we consider $N = 1{,}461{,}617$. We select the base $a = 8$, and use a block configuration of $\mathbf{m} = [4]^{16}$ and $\mathcal{T} = [0] + [2]^{15}$. With 16 blocks of size 4 and an overlap 2, the total number of control qubits is $4 + 15 \times (4 - 2) = 34$. This corresponds to a total phase space of 2^{34} values.

For this example, the number of top candidates collected from each block, `num_selected_candidates`, was set to 8. The initial target state for each block was set using the uniform weights $w_j = \frac{1}{\sqrt{L}}$ with a large number of terms, $L = 500$. The measured candidate bitstrings and their empirical probabilities for each of the 16 blocks are shown in Table 7.

The final classical stage of the for factoring $N = 1{,}461{,}617$ involved the stitching and classical post-processing of the QPE results. After integrating the partial candidates, the successful combination yielded the successful stitched bitstring `0011111111001100010000110011110011`, which corresponds to the integer value 4,281,404,659. This resulted in the fractional phase:

$$\phi = \frac{4{,}281{,}404{,}659}{2^{34}} = \frac{4{,}281{,}404{,}659}{17{,}179{,}869{,}184}$$

Applying the classical continued fraction algorithm to this phase successfully recovered the period $r = 3{,}800$. Using this value in the final classical step $\gcd(a^{r/2} \pm 1, N) = \gcd(8^{1900} \pm 1, 1{,}461{,}617)$, a non-trivial factor 1201 was successfully found.

Remark 5. *A thorough analysis on the dependence of the probability of success on other parameters (ref. Algorithms 1 to 4) will be provided in the future.*

5 Computational Complexity

Asymptotically, the quantum circuit depth arising from our proposed modular approach for integer factorization remains the same as that in standard Shor's algorithm. However, the quantum circuits for each block are shallower in comparison to the quantum circuit corresponding to the standard implementation of Shor's algorithm. In our proposed modular framework, each circuit block has depth has depth $\mathcal{O}(m_{\max} \cdot \mathrm{Depth}(U))$. We further note that, in our proposed framework, the qubit requirement for the work register is the same as that of the standard Shor's algorithm.

We also note that the preparation of the non-standard initial target state $|\psi_0\rangle$, as defined in Equation (1), typically incurs a cost depending on L. If L is chosen to be a small fixed number (e.g., $L = 4$ or $L = 500$), and a uniform superposition of states is prepared (i.e., $w_j = 1/\sqrt{L}$), then this state preparation cost may be very small compared to the cost of the controlled modular exponentiation operations involved in the QPE blocks.

A key advantage of our proposed approach is a significant reduction in the number of phase (or counting) qubits per circuit block. For an n-bit integer, the size of the phase register in the standard algorithm is approximately $2n+1$, whereas in the modular framework it is reduced to $m_{\max}$, where $m_{\max} = \max(m_1, m_2, \ldots, m_B)$ and m_k is the number of qubits allocated to the k-th block. In practice, $m_{\max}$ can be chosen to be very small, for example 3 or 4 phase qubits were sufficient for the computational examples considered (ref. Section 3). At RSA-2048 scale ($n = 2048$), the phase register requirement drops from 4097 to $m_{\max}$.

The classical cost arises from carry-aware stitching of block outcomes combined with continued fraction expansion. The use of overlap and pruning thresholds (`max_limit_combos`) restrict the search to a small set, keeping the classical overhead negligible in practice.

6 Conclusion

We have introduced a modular, windowed formulation of Shor's algorithm that restructures phase estimation into B independent, shallow blocks that can be executed sequentially or in parallel, followed by carry-aware classical stitching (ref. Algorithms 1 to 4). The result is a significant reduction of the counting qubit requirement (corresponding to the phase register) from approximately $2n+1$ to $m_{\max}$, where $m_{\max} = \max(m_1, m_2, \ldots, m_B)$ and m_k is the number of qubits allocated to the k-th block. Here, $m_{\max}$ can be chosen to be small, e.g. 3 or 4 qubits were found to be sufficient for the computational examples considered (ref. Section 3). Further, these independent blocks are amenable to parallelization, which allows for a shallower effective circuit depth and significantly lower overall execution time, making the approach far more viable for near-term quantum hardware than the standard Shor's algorithm. The number of qubits required for the work register in our approach remains the same as in the standard Shor's algorithm.

One of the key ingredients of this framework is the overlap mechanism. By enforcing consistency across overlapping phase windows up to a single-bit carry, the stitching procedure can reliably reconstruct full-length phase candidates obtained from independent blocks. This overlap-based redundancy improves robustness against noise and prunes inconsistent candidates, while zero-overlap configurations can still succeed in some settings with small block counts.

Our proposed modular, windowed formulation of Shor's algorithm directly mitigates one of the principal limitations of NISQ devices, namely the restricted number of high-quality qubits and limited coherence times. The modular phase-estimation layer is complementary to recent advances in modular arithmetic optimizations, and together these approaches offer cumulative improvements in the overall resource efficiency of quantum factoring. By transforming Shor's algorithm into a sequence of shallow, parallelizable blocks with lightweight classical reconstruction, our proposed formulation could provide a practical approach for quantum factoring on near-term and future fault-tolerant devices.

References

[1] Peter W Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*, pages 124–134. Ieee, 1994.

[2] Michael A. Nielsen and Isaac Chuang. *Quantum Computation and Quantum Information.* Cambridge University Press, 2000.

[3] Christof Zalka. Fast versions of Shor's quantum factoring algorithm. *arXiv preprint quant-ph/9806084*, 1998.

[4] Vlatko Vedral, Adriano Barenco, and Artur Ekert. Quantum networks for elementary arithmetic operations. *Physical Review A*, 54(1):147, 1996.

[5] Syed Shamikh Iqbal and Aasim Zafar. Enhanced Shor's algorithm with quantum circuit optimization. *International Journal of Information Technology*, 16(4):2725–2731, 2024.

[6] XinJian Tan and Peng Gao. An efficient quantum circuit implementation of Shor's algorithm for GPU accelerated simulation. *AIP Advances*, 14(2), 2024.

[7] Alok Shukla and Prakash Vedula. Towards practical quantum phase estimation: A modular, scalable, and adaptive approach. *arXiv preprint arXiv:2507.22460*, 2025.

[8] Alok Shukla and Prakash Vedula. Modular quantum amplitude estimation: A scalable and adaptive framework. *arXiv preprint arXiv:2508.05805*, 2025.

[9] Thomas Häner, Martin Roetteler, and Krysta M Svore. Factoring using 2n+ 2 qubits with Toffoli based modular multiplication. *arXiv preprint arXiv:1611.07995*, 2016.

[10] Craig Gidney and Martin Ekerå. How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits. *Quantum*, 5:433, 2021.

[11] Clémence Chevignard, Pierre-Alain Fouque, and André Schrottenloher. Reducing the number of qubits in quantum factoring. In *Annual International Cryptology Conference*, pages 384–415. Springer, 2025.