CAUCHY – Jurnal Matematika Murni dan Aplikasi
Volume 11 (1) (2026), Pages 1409-1434
p-ISSN: 2086-0382; e-ISSN: 2477-3344

Toward Optimal Quantum Multipliers: Space–Time Efficiency and Asymptotic Bounds for Advanced Toom–Cook Methods

Rini Wisnu Wardhani[1*] and Dedy Septono Catur Putranto[2,3]

[1] *Dept. of Hardware Security Engineering, Politeknik Siber dan Sandi Negara, Indonesia*
[2] *Dept. of Cyber Security Engineering, Politeknik Siber dan Sandi Negara, Indonesia*
[3] *Badan Siber dan Sandi Negara, Indonesia*

Abstract

This paper studies the asymptotic bounds of quantum resources required for high-degree multiplication based on the Toom–Cook algorithm. This work extends prior investigations on high- and half-degree quantum multiplication and proposes an optimized Toom–Cook 25.5-way quantum multiplication architecture to establish optimal asymptotic bounds on quantum resource consumption for Toom–Cook–based multiplication schemes. We provide asymptotic expressions for qubit complexity, Toffoli gate count, and Toffoli depth. The proposed Toom–Cook 25.5-way architecture achieves improved asymptotic performance, requiring a qubit count of qubit count of $n^{1.176}$, approximately $648n^{\log_{26} 51} - 666n$ Toffoli count, and $n^{1.03025}$ Toffoli depth. Compared with existing classical and quantum Toom–Cook–based methods, the proposed 25.5-way Toom–Cook algorithm achieves lower asymptotic complexity and reduced quantum resource costs, yielding tighter bounds for optimal quantum multiplication.

Keywords: Asymptotic performance analysis; Multiplier; Quantum Arithmetic; Toom–Cook 25.5-way

1. Introduction

Multiplication is a fundamental arithmetic operation, particularly in computations involving large integers, and improving its asymptotic efficiency is therefore of central importance. For example, cryptographic algorithms—especially those in the domain of post-quantum cryptography (PQC)—rely heavily on efficient multiplication operations. Many practical PQC schemes, including lattice-based cryptosystems, employ Toom-Cook 4-way multiplication to process large integers efficiently [1]. Consequently, optimized multiplication techniques are essential to meet the computational demands of modern cryptographic protocols.

Fig. 1 provides an overview of the fundamental operations underlying recent advances in quantum arithmetic, as discussed in [1, 2]. This research area has attracted considerable attention, particularly in the design and optimization of quantum arithmetic circuits aimed at accelerating core arithmetic operations [2]. Moreover, the importance of execution-time efficiency has been widely evaluated in related studies, as illustrated in Fig. 2, which presents a runtime analysis of Open Quantum Safe lattice-based cryptographic algorithms, specifically Key Encapsulation

*Corresponding author. E-mail: rini.wisnu@poltekssn.ac.id

Submitted: April 14, 2026 Reviewed: May 01, 2026 Accepted: May 27, 2026
DOI: https://doi.org/10.18860/cauchy.v11i1.41908

Mechanisms. In this context, the performance of quantum arithmetic circuits is commonly evaluated using two primary metrics: qubit consumption, which reflects space complexity, and circuit depth, which captures logical execution time [1, 3].

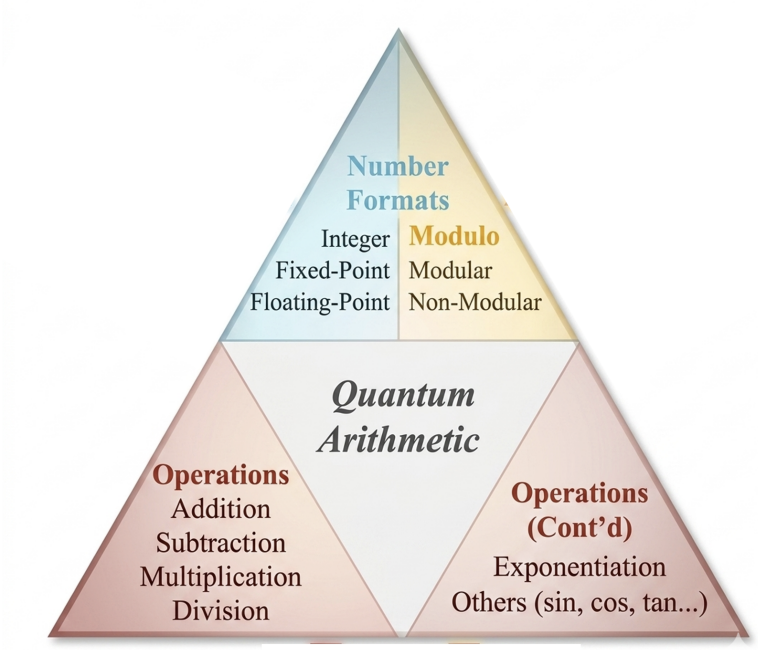

Fig. 1: Three Principal Aspects of Quantum Arithmetic (Reproduced from [2]).

This paper studies multiplication based on the Toom-Cook algorithm [4, 5], with a focus on high- and half-degree quantum multiplication techniques and their asymptotic resource behavior. We investigate Toom–Cook–based quantum multiplication architectures (e.g.[3, 6–11]) and propose an optimized Toom-Cook 25.5-way design under a division-free computation model. The proposed architecture is evaluated through asymptotic analysis and systematic comparisons with existing Toom-Cook 2.5-way [3], 8.5-way [10], 10.5-way [10], and 20.5-way [11]constructions. To ensure a fair and consistent comparison, performance is evaluated following [3] in terms of qubit complexity, Toffoli gate count, and Toffoli depth. Multiplication schemes that exhibit lower asymptotic bounds than previously reported Toom-Cook-based quantum multiplication methods indicate more efficient scaling of quantum resources, leading to more efficient quantum circuit implementations and potentially enabling faster computation.

The main contributions of this work are summarized as follows:

1. We provide a unified implementation and comparison framework for Toom–Cook 8.5-way, 10.5-way, 20.5-way, and the proposed 25.5-way quantum multiplication architectures.
2. We present an asymptotic comparative analysis of balanced and unbalanced (high- and half-degree) Toom–Cook–based multiplication in both classical and quantum environments.
3. We establish a more efficient multiplication scheme, represented by reduced asymptotic complexity indicators, including qubit count, Toffoli gate count, and Toffoli depth, using the Toom–Cook 25.5-way architecture, which achieves the lowest quantum resource complexity among the evaluated designs. The proposed Toom–Cook 25.5-way architecture exhibits improved asymptotic performance, achieving a qubit complexity of qubit count of qubit count of $n^{1.176}$, approximately $648n^{\log_{26} 51} - 666n$ Toffoli count, and $n^{1.03025}$ Toffoli depth.

The paper is structured as follows: Section 2 reviews prior work on Toom–Cook multiplication and its variants. Section 3 presents the methodology and construction of the proposed Toom–Cook 25.5-way architecture. Section 4 discusses the asymptotic results and resource comparisons. Section 5 summarizes the main findings and presents the conclusions of this study.

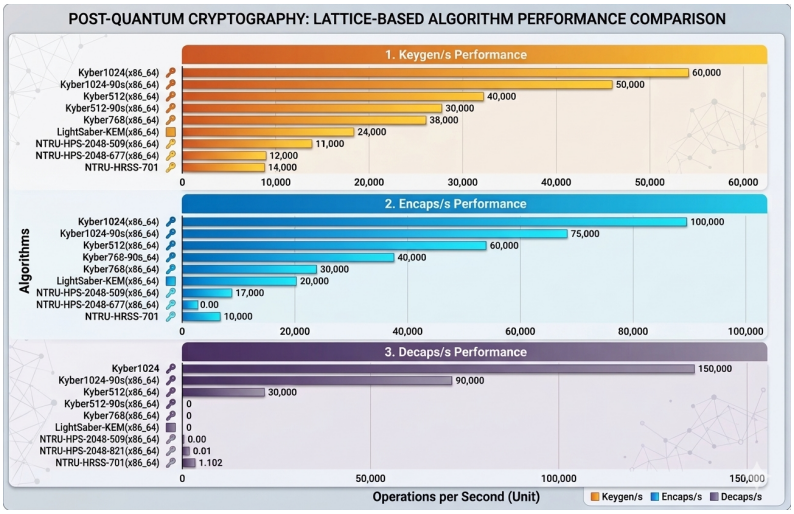

Fig. 2: Runtime analysis of Open Quantum Safe Lattice-based Cryptographic algorithms (Key Encapsulation Mechanisms)

2. Theoretical Basis

This section presents Toom–Cook–based multiplication algorithm, which adheres to a five-step computational framework as established in [6], [7] and [3]. This framework comprises the stages of splitting, evaluation, pointwise multiplication, interpolation, and recomposition.

2.1. Notation

The Toom–Cook algorithm originates from the foundational works of Toom and Cook [4, 5, 7]. In this study, in addition to providing a comparison with naïve multiplication and the Karatsuba algorithm, we adopt techniques, strategies, notation, and terminology from several established references on balanced [6, 8, 9] and unbalanced (high- and half-degree) Toom–Cook–based multiplication [3, 10, 11]. For clarity and ease of reference, several key notations used in the theoretical background of the algorithm and in the computational steps are summarized in Table 1. The symbols related to the computational steps are described separately in Table 2.

2.2. Toom–Cook Computation Steps

Let u and v be polynomials over an integral domain $\mathbb{R}[x]$. The Toom–Cook algorithm computes their product by decomposing the computation into five phases: splitting, evaluation, recursive multiplication, interpolation, and recomposition.

2.2.1. Splitting

A base $Y = x^b$ is first chosen to partition the input polynomials. The operands u and v are then expressed as homogeneous polynomials

$$\mathfrak{u}(y,z) = \sum_{i=0}^{n-1} u_i z^{n-1-i} y^i, \quad \mathfrak{v}(y,z) = \sum_{i=0}^{m-1} v_i z^{m-1-i} y^i,$$

with degrees $n-1$ and $m-1$, respectively. Under this representation, $u = \mathfrak{u}(x^b, 1)$ and $v = \mathfrak{v}(x^b, 1)$, where the coefficients satisfy $\deg(u_i) < b$ and $\deg(v_i) < b$.

These polynomials consist of n and m coefficients, respectively, and their degrees, denoted as $\deg(\mathfrak{u}) = n-1$ and $\deg(\mathfrak{v}) = m-1$, align with the aforementioned coefficients and degrees. Such that $\mathfrak{u}(x^b, 1)$ equals the variable u and $\mathfrak{v}(x^b, 1)$ equals the variable v under the specified circumstances. The coefficients of $u_i, v_i \in \mathbb{R}[x]$ can be represented as polynomial functions and can be chosen to have a specified degree, denoted as $\forall i$, $\deg(u_i) < b$, $\deg(v_i) < b$. The typical

Table 1: List of Notations and Symbols for Toom–Cook Algorithm

Symbol	Description
Polynomial Representation	
u, v	Input polynomials in $\mathbb{R}[x]$
$\mathbb{R}[x]$	Ring of polynomials over an integral domain
$\mathfrak{u}(y, z)$	Homogeneous polynomial representation of u
$\mathfrak{v}(y, z)$	Homogeneous polynomial representation of v
u_i, v_i	Polynomial coefficients of u and v
n, m	Number of coefficients of u and v
$\deg(\mathfrak{u}), \deg(\mathfrak{v})$	Degrees of homogeneous polynomials
Splitting Parameters	
$Y = x^b$	Base used for splitting operands
b	Block size (maximum degree of u_i, v_i)
y, z	Formal variables in homogeneous representation
Evaluation Phase	
$\mathfrak{w}$	Product polynomial $\mathfrak{u} \cdot \mathfrak{v}$
d	Degree of result polynomial, $d = n + m - 2$
P_d	Set of $d + 1$ evaluation points
(α_i, β_i)	Evaluation point in $\mathbb{R}[x]$
$\mathfrak{u}(\alpha_i, \beta_i)$	Evaluation of $\mathfrak{u}$ at a point
Recursive Multiplication	
$\mathfrak{w}(\alpha_i, \beta_i)$	Point-wise multiplication result
$d + 1$	Number of recursive sub-multiplications
Interpolation Phase	
w_i	Coefficients of the product polynomial $\mathfrak{w}$
A_d	$(d + 1) \times (d + 1)$ Vandermonde-like matrix
A_d^{-1}	Inverse matrix used for interpolation
Recomposition	
$\mathfrak{w}(x^b, 1)$	Final recomposition evaluation
w	Final multiplication result
Toom–Cook Variants	
k	Toom–Cook splitting parameter
Toom–k-way	Balanced Toom–Cook variant
Toom–n-half	Unbalanced (Toom'n'half) variant

way of implementing the Toom k-way algorithm requires the operands to be evenly sized, with one operand m being the same size as the other operand n. In contrast to the same-size operand value, we expand our approach to elaborate on unbalanced or not-the-same-size operand value circumstances. This approach refers to [6, 7] that labels the methodology as Toom-$\frac{n+m}{2}$ for Toom'n'half terminology. In this research, we specifically refer to it as Toom-Cook high- and half-degree quantum multiplication.

2.2.2. Evaluation

To compute $\mathfrak{w} = \mathfrak{u} \cdot \mathfrak{v}$ of degree $d = n + m - 2$, a total of $d + 1 = n + m - 1$ evaluation points $P_d = \{(\alpha_0, \beta_0), \ldots, (\alpha_d, \beta_d)\} \in \mathbb{R}[x]$ are required. The process of computing the evaluation of a single polynomial, for example $\mathfrak{u}$, at specific positions (α_i, β_i), can be achieved by the utilization of matrix by vector multiplication.

2.2.3. Recursive Multiplication

At each evaluation point (α_i, β_i), the product

$$\mathfrak{w}(\alpha_i, \beta_i) = \mathfrak{u}(\alpha_i, \beta_i)\, \mathfrak{v}(\alpha_i, \beta_i)$$

is computed. This step involves $d + 1$ independent multiplications of polynomials whose degrees are bounded by that of $Y = x^b$.

Table 2: List of Notations and Symbols for Toom–Cook Computational Steps

Symbol	Description
General Parameters and Functions	
x, y	Input operands to be multiplied
n, m	Bit-lengths of input operands x and y
k	Toom–Cook splitting parameter (number of parts)
T_n	Cost of multiplying two n-bit numbers
A_n	Cost of n-bit addition or subtraction
b	Block size used in splitting
$Y = x^b$	Base used for polynomial representation
j	Radix value in splitting stage
Evaluation and Interpolation	
$t(x)_i, t(y)_i$	Sub-blocks of operands after splitting
$x(t), y(t)$	Evaluation of x and y at point t
$k_0, \ldots, k_{50}$	Point-wise multiplication results
$A, \ldots, AY$	Products of evaluated points
Recomposition	
$YZ, \ldots, XA$	Recomposition coefficients
xy	Final multiplication result
Quantum Cost Metrics	
T_d	Toffoli depth of the circuit
S_k	Number of subtrees at level k
D_k	Circuit depth at level k

2.2.4. Interpolation

Given the $d+1$ evaluated values of $\mathfrak{w}$, the coefficients of

$$\mathfrak{w}(y, z) = \sum_{i=0}^{d} w_i z^{d-i} y^i$$

are recovered through polynomial interpolation. This process depends only on the degree d and the chosen evaluation points, and is performed using the inverse of a $(d+1) \times (d+1)$ Vandermonde-like matrix A_d, as described in [7].

2.2.5. Recomposition

Finally, the product of the original operands is obtained by evaluating

$$w = \mathfrak{w}(x^b, 1).$$

This recomposition step requires at most d shift and addition operations, yielding the final multiplication result.

2.3. Toom-Cook k-way variants

The Toom-Cook algorithm, particularly its k-way variants, offers a more efficient alternative to conventional multiplication with $O(n^2)$ time complexity. The parameter k determines how the input operands are partitioned, enabling the multiplication to be decomposed into smaller subcomputations. In this work, an n-digit integer is modeled as a degree-n polynomial distributed across a k-element register array.

Existing studies have examined Toom-Cook multiplication for various values of k. In the quantum environment, $k = 2$ (corresponding to Karatsuba with complexity $O(n^{1.58})$) and $k = 3$ have been analyzed in [8], while Dutta et al. investigated the unbalanced $k = 2.5$ variant in [3]. In classical computation, Zanoni et al. reported implementations up to $k = 8$ [6]. Quantum

designs of balanced Toom–Cook multipliers with $k = 2$, $k = 4$, and $k = 8$ were later presented by Putranto et al. [9]. These works are consistent with earlier theoretical results by Bodrato [7], who predicted potential resource reductions for multiplication and proposed unbalanced Toom–n-half Toom–Cook variants in the classical environment, albeit without providing explicit circuit constructions.

3. Methods

The computation steps (e.g., splitting, evaluation, recursive multiplication, interpolation, and recomposition) described above represent the general workflow of Toom–Cook–based multiplication. In the following subsection, we present the research approach adopted in this work, including the selection of evaluation-point parameters, the formulation of a division gate-free strategy, and metrics used to assess efficiency or quantum cost.

3.1. Research Approach

The classical Toom–Cook multiplication algorithm, originally introduced by Toom [4] and Cook [5], significantly reduces computational complexity compared to naive schoolbook multiplication. Over time, two main categories of Toom–Cook–based multiplication have been developed: balanced and unbalanced variants (or "Toom-n-half" [7] or high- and half-degree" variant [10, 11]). Balanced Toom–Cook methods are discussed in [6, 8, 9], while unbalanced Toom–Cook algorithms—particularly in their high- and half-degree forms—have been shown to offer substantial potential for improved resource efficiency [3, 7, 10].

Recent studies have extended Bodrato's work by proposing quantum circuit design frameworks for higher-degree Toom–Cook multiplication, incorporating analyses of gate counts and performance based on tree-structured multiplication methods, as well as comparisons with asymptotic multiplication techniques. This work follows a similar analytical approach. We revisit and further extend these methodologies by exploring the upper bounds of achievable Toom–Cook degrees in quantum circuit designs. We adopt the same computational steps (as illustrated in Fig. 3) and parameter settings as those used in Bodrato's classical multiplication studies [7] and subsequent quantum tree-structured multiplication methods works [3, 11] to ensure a direct and consistent comparison.

Nevertheless, designing multiplier circuits for even higher-degree Toom–Cook schemes presents substantial challenges, particularly in determining feasible degree limits and developing accurate circuit architectures that support large-integer multiplication while maintaining or reducing quantum resource consumption.

This study investigates higher-degree multiplication schemes beyond those examined in prior work, while preserving the efficiency of quantum multiplication resources. Detailed step-by-step quantum circuit constructions and explicit final circuit implementations are not presented. Instead, theoretical asymptotic bounds are derived to characterize the time efficiency and asymptotic behavior of advanced Toom–Cook multiplication methods. The focus is on achieving stable high-degree multiplication while optimizing quantum resource efficiency by integrating methodological strategies from recent Toom–Cook–based multiplication studies, including [3, 6–11].

3.2. Toom–Cook Predefined Points

At the Toom–Cook multiplication stage, appropriate evaluation values must be carefully selected from the many possible candidates for use in the subsequent evaluation phase. Similar to solving systems of equations through elimination or substitution, the selection of evaluation parameters plays a critical role in enabling the recovery of the desired variables during the interpolation phase. In the context of Toom–Cook multiplication, the mathematical rationale underlying these choices has been extensively discussed in prior works, including those by Bodrato et al. [7, 12,

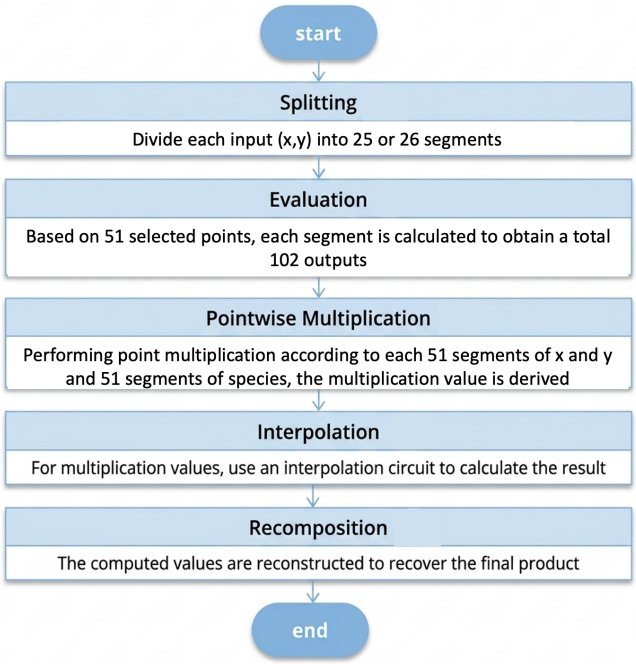

Fig. 3: Computational Step High- and Half degree Toom-Cook Quantum Multiplication. We adopt a five-step computational framework for high- and half-degree Toom–Cook quantum multiplication, following the methodology presented in [3, 8–10]. The figure illustrates the Toom–Cook 25.5-way configuration.

13] and Wardhani et al. [10, 11].

Numerous value and parameter configurations are possible for assigning evaluation points. For instance, the 51 predefined evaluation points used in the 25.5-way Toom–Cook method, as listed in Table 3 or Eq. (8), are analogous to the 41 predefined points reported in [11] for the 20.5-way Toom–Cook design. In this work, a specific set of evaluation points is carefully selected to improve computational efficiency in the proposed Toom–Cook–based multiplication scheme and to align with the design strategies adopted in this study.

3.3. Multiplication Complexity

A multiplication algorithm is a procedure for computing the product of two numbers. In classical computation, multiplication is commonly introduced as repeated addition, with the naïve or schoolbook method serving as the most fundamental approach. Over time, more efficient algorithms—such as Karatsuba, Montgomery, and Toom–Cook multiplication—have been developed to reduce computational complexity and improve resource utilization.

The efficiency of a multiplication algorithm is often evaluated using asymptotic complexity, which measures how the number of required operations grows with the input size. Naïve multiplication exhibits a quadratic asymptotic complexity of $O(n^2)$, where n denotes the number of input digits. In contrast, asymptotic multiplication techniques aim to achieve lower growth rates as n increases, typically expressed using Big-O notation.

In the quantum domain, multiplication extends classical concepts to operations on quantum states. The quantum cost of a multiplication circuit is characterized by the required number of qubits and quantum gates, such as Toffoli and CNOT gates, as well as the overall circuit depth. Minimizing these resource costs is crucial due to the limited capabilities of current quantum hardware. Consequently, optimizing both asymptotic performance and quantum resource consumption is essential for efficient quantum multipliers, particularly in computation-intensive applications such as cryptography.

3.4. Quantum Circuit Division Gate-Free

In both classical electronics and quantum computing, circuit design involves carefully selecting the gates used to construct the circuit. In quantum computing, division operations are integral for dividing numbers within an algorithm or quantum circuit, a fundamental step in many mathematical algorithms. However, division in a quantum context is more intricate and resource-intensive compared to simpler operations such as addition or multiplication, often requiring additional qubits and logic gates.

Implementing division can substantially increase the circuit depth and the number of logical operations. Therefore, designers often aim to avoid such operations whenever possible, as they tend to consume significant computational resources. Nevertheless, due to the inherent complexity of division and the elaborate procedures required to eliminate it, not all quantum circuit designs can fully avoid division operations. For instance, the Toom 3-way design [8] and the Toom-2.5 design [3] both incorporate division gates in their quantum circuit implementations. In contrast, Zanoni's Toom-8 design [6] and the design proposed by Gu et al. [13] do not utilize division gates.

In prior works by Putranto et al. on high-degree [9] and high-and-half-degree [11] Toom–Cook multiplication, the authors not only derive asymptotic complexity bounds but also present explicit quantum circuit constructions. These circuits illustrate the correspondence between the underlying Toom–Cook equations and the implications of the quantum gates employed, as shown in Figures 6 and 7.

In their proposed Toom-multiplication designs, the authors incorporate technical strategies specifically tailored to quantum computing environments, drawing inspiration from earlier classical and quantum studies [3, 6–8]. In particular, they introduce division-gate-free quantum circuits for Toom–Cook 8-way [9] and Toom–Cook 20.5-way [10, 11] multiplication schemes. In these prior works, a general scheme and corresponding circuits at the evaluation-step level—specifically for evaluation points x and y—are presented to illustrate the overall structure and organization of quantum circuits for Toom-based multiplication. In these constructions Figures 6 and 7, division gates are not used; instead, they are replaced by techniques such as *copy*, *addition*, and *subtraction*, as discussed in Dutta et al. and Larasati et al. Consequently, these circuits are categorized as employing a division-gate-free strategy [9, 11] in their evaluation-point implementations. Rather than performing explicit division operations, the circuits realize equivalent functionality through sequences of alternative quantum gates—namely *copy*, *addition*, and *subtraction*. The evaluation procedures at points x and y further consist of combinations of *register copying*, *addition*, *subtraction*, *squaring*, and *shifting* operations.

Present work does not present explicit quantum circuit implementations at the gate or register level for the Toom–Cook 25.5-way method. This omission is primarily due to the extremely large circuit dimensions associated with the Toom–Cook 25.5-way approach, as well as our intention to maintain the primary focus of this paper, namely the theoretical analysis of asymptotic space–time efficiency and asymptotic bounds for advanced Toom–Cook methods toward optimal quantum multipliers.

Nevertheless, Fig. 5 provides a high-level circuit diagram that illustrates the overall register allocation and data flow of the proposed design. Detailed aspects such as ancilla usage, gate-level decomposition, and uncomputation procedures are omitted, as their scale renders them impractical to depict explicitly. Accordingly, we choose to focus on asymptotic bounds and theoretical performance guarantees rather than providing step-by-step quantum circuit constructions. The proposed analysis demonstrates that high-degree Toom–Cook–based multiplication can be performed in a stable manner while keeping quantum resource usage under control, achieving reduced asymptotic resource requirements despite increased algorithmic complexity.

3.5. Quantum Cost Metric

When evaluating the quantum cost of a circuit, several key metrics are commonly considered, including qubit count, Toffoli (or T) gate count, CNOT gate count, and overall circuit depth

[3, 8–11]. These metrics collectively characterize the time and space complexity of a quantum circuit and directly reflect the computational efficiency of the underlying algorithm. In particular, qubit count (circuit width) indicates the required quantum memory resources, which are severely constrained in current hardware. The Toffoli or T-gate count captures the most resource-intensive operations in fault-tolerant quantum computation, while the CNOT count reflects the cost of two-qubit interactions. The circuit depth represents the number of sequential gate layers, making its optimization—alongside width—essential for achieving efficient and scalable quantum algorithms.

4. Results and Discussion

This work further investigates optimization strategies aimed at reducing quantum cost, as reported in prior literature. These strategies encompass the selection of evaluation points with opposite-point characteristics, the inverse elements, and the replacement of zero entries with infinite values. Building upon these considerations, we subsequently introduce the proposed Toom–Cook 25.5-way architecture, formulated within the previously described computation-step framework and accompanied by a general quantum design for the Toom–Cook 25.5-way scheme.

4.1. Design Strategies

In Toom–Cook multiplication computation, the evaluation and interpolation phases dominate the computational cost [7], as they are primarily realized through matrix–vector operations involving numerous arithmetic computations. These operations include additions, subtractions, shifts, and small multiplications or exact divisions, particularly during interpolation, over elements of $\mathbb{R}[x]$. To mitigate this overhead, prior work (e.g.,[7, 10, 14]) proposes an integrated splitting approach that intertwines all computational stages to reduce both memory requirements and overall number of operations in $\mathbb{R}$.

4.1.1. Opposite Points

Since the Toom–Cook–based approach relies on matrices as the foundation for its computations, the selection of appropriate evaluation points for matrix operations plays a crucial role in improving computational efficiency [7, 9, 12, 14]. One commonly adopted strategy is the use of opposite points, as depicted in Table 3, which enables matrix operations involving inverse relationships and can lead to shorter sequences of equations.

Interpolation computation step is performed by applying a sequence of row operations to the interpolation matrix. By eliminating multiple entries at each step, the overall sequence of operations can be significantly shortened [10–12]. Evaluations at opposite points produce matrix rows with identical absolute values that differ only in their sign patterns. By adding and subtracting these rows, and subsequently dividing the results by two, the following rows are obtained [7]:

$$\begin{pmatrix} \beta^d & \alpha\beta^{d-1} & \alpha^2\beta^{d-2}\ldots & \alpha^d \\ \beta^d & -\alpha\beta^{d-1} & \alpha^2\beta^{d-2}\ldots & (-\alpha)^d \end{pmatrix} \tag{1}$$

In this procedure (the matrix in Equation (1)), half of the entries are adjusted to zero, as illustrated by the matrix shown in Equation (2).

$$\begin{pmatrix} 0 & \alpha\beta^{d-1} & 0 & \alpha^3\beta^{d-3} & \ldots \\ \beta^d & 0 & \alpha^2\beta^{d-2} & 0 & \ldots \end{pmatrix} \tag{2}$$

The opposite-points strategy requires the evaluation points to be selected and applied in symmetric pairs. Consequently, when including the two special points $(1, 0)$ and $(0, 1)$, commonly referred to as zero and infinity, an even number of points is preferable.

4.1.2. Inverses

This technique involves selecting inverse (or opposite) evaluation points that yield matrix representations with inherent symmetry. Although the resulting formulation requires matrix-based computations, the chosen values lead to computationally simpler operations. By exploiting symmetry [7], these evaluation points can produce more efficient inverse computations or matrix structures that significantly simplify the overall computational sequence.

In the interpolation matrix, if we evaluate (α, β) and its inverse (β, α), it reveals the presence of two symmetric lines in the matrix. These lines exhibit identical values and signs but are arranged in opposite orders, as described in Eq. (3). By performing the operations of addition and subtraction on the two lines, we are able to derive two distinct lines. One of these lines exhibits symmetry, while the other does not possess this characteristic (asymmetric), as shown in Eq. (4).

$$\begin{pmatrix} \beta^d & \alpha\beta^{d-1} & \dots & \alpha\beta^{d-1} & \alpha^d \\ \alpha^d & \alpha^{d-1}\beta & \dots & \alpha^{d-1}\beta & \beta^d \end{pmatrix} \tag{3}$$

$$\begin{pmatrix} \alpha^d + \beta^d & \alpha^{d-1}\beta + \alpha\beta^{d-1} & \dots & \beta^d + \alpha^d \\ \alpha^d - \beta^d & \alpha^{d-1}\beta - \alpha\beta^{d-1} & \dots & \beta^d - \alpha^d \end{pmatrix} \tag{4}$$

If each row of the interpolation matrix admits an inverse, the resulting matrix can be reformulated in a block-wise structure [7]. Matrices A' and B' denote the mirrored counterparts of A and B, respectively. The two exceptional points $(1, 0)$ and $(0, 1)$ are omitted, as they can be treated separately. The regular points $(1, 1)$ and $(-1, 1)$ produce symmetric rows and are therefore included in A and A' without requiring precomputation. During subsequent row operations on A, any entry eliminated in A is simultaneously eliminated in A' through the same operation; an analogous effect holds for B and B'. As a result, the zeroing effect of each operation is effectively doubled.

4.1.3. Replace infinity values

When full symmetry cannot be achieved in the computation, a commonly adopted strategy is to employ standard evaluation points such as 0, 1, -1, and ∞. This choice offers several advantages: it enables efficient interpolation, preserves symmetry during the evaluation phase (for most points), and satisfies the minimum number of points required for high-degree polynomial interpolation.

The evaluation at the point ∞ simplifies the resulting equations by isolating the coefficient of the highest-degree term of the polynomial, which can be interpreted through a limit-based computation. Similarly, evaluation at 0 simplifies the computation by eliminating all polynomial terms that contain the variable. In contrast to many Toom–Cook–based multiplication designs that utilize ∞ as an evaluation point, this study follows the strategy proposed by Bodrato et al. [7] by replacing the point at infinity with 0, as reflected in Table 3. This modification is intended to optimize the evaluation phase and improve overall computational efficiency.

4.2. Toom-Cook 25.5-way

The proposed Toom-based multiplication method for quantum circuits is derived from the classical Toom–Cook algorithm for large-integer multiplication and is designed to improve performance over existing approaches. Since not all values of k lead to efficient large-digit multiplication, this work develops robust strategies and circuit designs for the proposed method. We analyze its asymptotic complexity and evaluate the quantum cost of the $k = 25.5$ variant through numerical analysis, enabling direct comparison with prior Toom-based designs. Efficiency is achieved by integrating optimized computational phases and techniques from previous studies, including parameter selection. Fig. 4 illustrates the recursive tree structure of the Toom–Cook 25.5-way method.

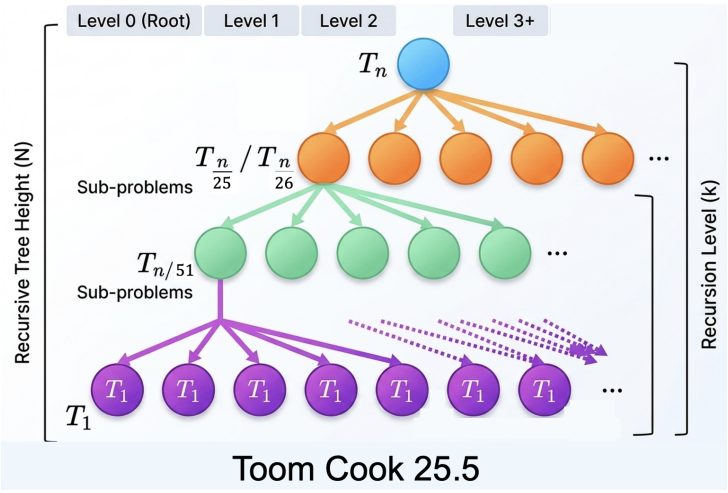

Fig. 4: Recursion tree structure Toom–Cook 25.5-way multiplication, where T represents the Toom-Cook $k-$way Multiplication and n and N represent the bit length for each level and the overall depth of the tree, respectively.

$$\begin{aligned} x = {} & x_{24}s^{24j} + x_{23}s^{23j} + x_{22}s^{22j} + x_{21}s^{21j} + x_{20}s^{20j} + x_{19}s^{19j} + x_{18}s^{18j} \\ & + x_{17}s^{17j} + x_{16}s^{16j} + x_{15}s^{15j} + x_{14}s^{14j} + x_{13}s^{13j} + x_{12}s^{12j} + x_{11}s^{11j} + x_{10}s^{10j} \\ & + x_9s^{9j} + x_8s^{8j} + x_7s^{7j} + x_6s^{6j} + x_5s^{5j} + x_4s^{4j} + x_3s^{3j} + x_2s^{2j} + x_1s^{j} + x_0 \end{aligned} \tag{5}$$

$$\begin{aligned} y = {} & y_{25}s^{25j} + y_{24}s^{24j} + y_{23}s^{23j} + y_{22}s^{22j} + y_{21}s^{21j} + y_{20}s^{20j} + y_{19}s^{19j} + y_{18}s^{18j} \\ & + y_{17}s^{17j} + y_{16}s^{16j} + y_{15}s^{15j} + y_{14}s^{14j} + y_{13}s^{13j} + y_{12}s^{12j} + y_{11}s^{11j} + y_{10}s^{10j} \\ & + y_9s^{9j} + y_8s^{8j} + y_7s^{7j} + y_6s^{6j} + y_5s^{5j} + y_4s^{4j} + y_3s^{3j} + y_2s^{2j} + y_1s^{j} + y_0 \end{aligned} \tag{6}$$

The computation proceeds through five main phases—splitting, evaluation, recursive multiplication, interpolation, and recomposition—as illustrated in Fig. 3 and established in prior studies [6–11]. The input operands x and y denote the values to be multiplied, where x represents the complete input, and the subscripts $t(x)_0, t(x)_1, t(x)_{-1}, \ldots$ indicate its components. The notations $x(0), x(1), x(-1), \ldots$ refer to the evaluations of x at specific points.

At the initial stage of the Toom–Cook splitting process, applying the k-way algorithm requires selecting a base $Y = x^b$ and determining the radix value j, which can be precomputed using Eq. (7). In the splitting phase of the Toom–Cook 25.5-way method, the homogeneous polynomial inputs x and y defined in Eqs. (5) and (6) are partitioned into 25 and 26 segments, respectively, each of length $\frac{n}{26}$. These segments correspond to the coefficients and degrees of the original polynomials.

$$j = max\left\{\left\lfloor \frac{\lceil \log_2 x \rceil}{26} \right\rfloor, \left\lfloor \frac{\lceil \log_2 y \rceil}{25} \right\rfloor\right\} + 1 \tag{7}$$

During the evaluation phase, the values defined in Eq. (8) are used to evaluate the operands x and y at 51 predefined points, as required by Eq. (9). In accordance with the strategy proposed in [7], evaluation at infinity is avoided; instead, this point is omitted and replaced by zero.

$$
\begin{aligned}
&t(x)_1 = 0, t(x)_2 = 1, t(x)_3 = -1, t(x)_4 = 2, t(x)_5 = -2, t(x)_6 = 4, t(x)_7 = -4,\\
&t(x)_8 = 0.5, t(x)_9 = -0.5, t(x)_{10} = 0.25, t(x)_{11} = -0.25,\\
&t(x)_{12} = 0.125, t(x)_{13} = -0.125, t(x)_{14} = 0.0625,\\
&t(x)_{15} = -0.0625, t(x)_{16} = 0.03125, t(x)_{17} = -0.03125,\\
&t(x)_{18} = 0.015625, t(x)_{19} = -0.015625, t(x)_{20} = 0.0078125,\\
&t(x)_{21} = -0.0078125, t(x)_{22} = 0.00390625,\\
&t(x)_{23} = -0.00390625, t(x)_{24} = 0.001953125,\\
&t(x)_{25} = -0.001953125, t(x)_{26} = 0.0009765625,\\
&t(x)_{27} = -0.0009765625, t(x)_{28} = 0.00048828125,\\
&t(x)_{29} = -0.00048828125, t(x)_{30} = 0.000244140625,\\
&t(x)_{31} = -0.000244140625, t(x)_{32} = 0.0001220703125,\\
&t(x)_{33} = -0.0001220703125, t(x)_{34} = 0.00006103515625,\\
&t(x)_{35} = -0.00006103515625, t(x)_{36} = 0.000030517578125,\\
&t(x)_{37} = -0.000030517578125, t(x)_{38} = 0.0000152587890625,\\
&t(x)_{39} = -0.0000152587890625, t(x)_{40} = 0.00000762939453125,\\
&t(x)_{41} = -0.00000762939453125, t(x)_{42} = 0.000003814697265625,\\
&t(x)_{43} = -0.000003814697265625, t(x)_{44} = 0.0000019073486328125,\\
&t(x)_{45} = -0.0000019073486328125, t(x)_{46} = 0.00000095367431640625,\\
&t(x)_{47} = -0.00000095367431640625, t(x)_{48} = 0.00000047683715820312 5,\\
&t(x)_{49} = -0.00000047683715820312 5, t(x)_{50} = 0.000000238418579101 5625,\\
&t(x)_{51} = -0.000000238418579101 5625.
\end{aligned}
\tag{8}
$$

For the Toom–Cook 20.5-way multiplication, the study in [11] explicitly presents the equations used to compute the evaluation points, together with the corresponding quantum circuit implementations for the evaluation of x and y, as well as the evaluation tables obtained from quantum design employing a division-free technique. In contrast, for the Toom–Cook 25.5-way scheme, explicit derivations of the evaluation equations for x and y at the evaluation stage are provided Eq. (10); however, the corresponding quantum circuit representations are not illustrated. Nevertheless, the evaluation and overall circuit design are fully carried out within a quantum computing framework, and the resulting formulations are reflected in the reported resource-usage results.

In this work, the resource bounds are fully calculated using a division-free strategy, with the results reported in terms of resource estimation. This division-free design choice significantly reduces the computational complexity of multiplication, particularly for high-degree polynomial operations in the Toom–Cook 25.5-way scheme, and is well suited for quantum circuit implementations, as evidenced by the results presented in Table 5. Following the splitting and evaluation stages, the third step of the Toom–Cook 25.5-way method is recursive multiplication. A single non-recursive iteration performs 51 point-wise multiplications, each operating on inputs of length $\frac{n}{26}$. The products of the evaluated values in Eq. (9).

$$
\begin{aligned}
\mathcal{E}_x = \Big\{ & x(0),\ x(1),\ x(-1),\ x(2),\ x(-2),\ x(4),\ x(-4),\\
& x(0.5),\ x(-0.5),\ x(0.25),\ x(-0.25),\\
& x(0.125),\ x(-0.125),\ x(0.0625),\ x(-0.0625),\\
& x(0.03125),\ x(-0.03125),\ x(0.015625),\ x(-0.015625),\\
& x(0.0078125),\ x(-0.0078125),\ x(0.00390625),\ x(-0.00390625),\\
& x(0.001953125),\ x(-0.001953125),\\
& x(0.0009765625),\ x(-0.0009765625),\\
& x(0.00048828125),\ x(-0.00048828125),\\
& x(0.000244140625),\ x(-0.000244140625),\\
& x(0.0001220703125),\ x(-0.0001220703125),\\
& x(0.00006103515625),\ x(-0.00006103515625),\\
& x(0.000030517578125),\ x(-0.000030517578125),\\
& x(0.0000152587890625),\ x(-0.0000152587890625),\\
& x(0.00000762939453125),\ x(-0.00000762939453125),\\
& x(0.00000381469726562 5),\ x(-0.00000381469726562 5),\\
& x(0.00000190734863281 25),\ x(-0.00000190734863281 25),\\
& x(0.00000095367431640625),\ x(-0.00000095367431640625),\\
& x(0.00000047683715820312 5),\ x(-0.00000047683715820312 5),\\
& x(0.00000023841857910156 25),\ x(-0.00000023841857910156 25)\Big\}.
\end{aligned}
\tag{9}
$$

In the interpolation computation phase, the mathematical expression is formulated using matrix inversion to recover the interpolation results based on the outcomes from the recursive multiplication stage. This procedure serves as the inverse of pointwise multiplication, as illustrated by the matrix scheme in Eq. (22). In this computation stage, the obtained values are transformed back into polynomial form. In this work, the reconstruction strategy employs matrix operations, aligning with the evaluation stage's approach, which selects evaluation points based on opposite points and inverse parameter strategies to facilitate inverse matrix calculation, as discussed in [7, 9, 11]. The output of the interpolation block —which serves as the input for the subsequent recomposition—is represented by a sequence of variables: $YY, YX, YW, YV, YU, YT, YS, YR, YQ, YP, YO, YN, YM, YL, YK, YJ, YI, YH, YG, YF, YE, YD, YC, YB, YA, XZ, XY, XX, XW, XV, XU, XT, XS, XR, XQ, XP, XO, XN, XM, XL, XK, XJ, XI, XH, XG, XF, XE, XD, XC, XB$, and XA.

In the Toom–Cook 25.5-way algorithm, the final recomposition phase reconstructs the product from the interpolation results. The recomposed output is represented by the variables YY through XA. The expression xy in Eq. (10) denotes the recomposition result of the interpolation stage and therefore corresponds to the final output of the Toom–Cook 25.5-way multiplication.

$$
\begin{aligned}
xy = {} & XA * 2^{50j} + XB * 2^{49j} + XC * 2^{48j} + XD * 2^{47j} + XE * 2^{46j} +\\
& XF * 2^{45j} + XG * 2^{44j} + XH * 2^{43j} + XI * 2^{42j} + XJ * 2^{41j} +\\
& XK * 2^{40j} + XL * 2^{39j} + XM * 2^{38j} + XN * 2^{37j} + XO * 2^{36j} +\\
& XP * 2^{35j} + XQ * 2^{34j} + XR * 2^{33j} + XS * 2^{32j} + XT * 2^{31j} +\\
& XU * 2^{30j} + XV * 2^{29j} + XW * 2^{28j} + XX * 2^{27j} + XY * 2^{26j} +\\
& XZ * 2^{25j} + YA * 2^{24j} + YB * 2^{23j} + YC * 2^{22j} + YD * 2^{21j} +\\
& YE * 2^{20j} + YF * 2^{19j} + YG * 2^{18j} + YH * 2^{17j} + YI * 2^{16j} +\\
& YJ * 2^{15j} + YK * 2^{14j} + YL * 2^{13j} + YM * 2^{12j} + YN * 2^{11j} +\\
& YO * 2^{10j} + YP * 2^{9j} + YQ * 2^{8j} + YR * 2^{7j} + YS * 2^{6j} + YT * 2^{5j} +\\
& YU * 2^{4j} + YV * 2^{3j} + YW * 2^{2j} + YX * 2^{j} + YY
\end{aligned}
\tag{10}
$$

4.3. Toffoli Gate Count

Let T_n denote the cost of multiplying two n-bit operands using the Toom–Cook multiplier, and let A_n represent the cost associated with performing addition or subtraction on n-bit numbers. To implement an n-bit Toom–Cook 25.5-way multiplication, a total of 51 sub-multiplications of size $\frac{n}{26}$ are required, together with three types of adders of varying bit lengths. Specifically, the $\frac{n}{26}$-bit adders incur a total cost of 162 operations, while the $\frac{2n}{26}$-bit adders require 2040 operations. The following equation expresses the Toffoli-gate cost of the proposed multiplier design:

$$T_n = 51T_{\frac{n}{26}} + 162A_{\frac{n}{26}} + 2040A_{\frac{2n}{26}} \tag{11}$$

Following the circuit implementation of Dutta et al. [3], where the Cuccaro adder [15] is employed, the addition of two n-bit numbers requires $(2n-1)$ Toffoli gates. Consequently, the cost A_n of an in-place adder for two n-bit operands is commonly upper-bounded by $2n$ Toffoli gates [3]. Therefore, the Toffoli-gate cost of a n-bit Toom–Cook 25.5-way multiplication is given by Eq. (11). For recursive implementations, the cost grows according to Eq. (12), which reduces to Eq. (13) upon substituting the Toffoli cost of addition as $A_n = 2n$.

$$\begin{aligned} T_n = {} & 51^{\log_{26} n} T_1 + 162(A\frac{n}{26} + 80A\frac{n}{679} + \cdots + 80^{\log_{26}(n)-1}A_1) \\ & + 2040(A\frac{2n}{26} + 960A\frac{2n}{100} + \cdots + 792^{\log_{26}(n)-1}A_2) \end{aligned} \tag{12}$$

$$T_n = 51^{\log_{26} n} + \sum_{i=0}^{\log_{26}(n)-1} \left[323n(\frac{51}{26})^i \right] \tag{13}$$

The Toffoli cost of the recursive implementation is derived using the geometric series $\sum_{i=0}^{m-1} r^i = \frac{1-r^m}{1-r}$, as shown in Eq. (14). This formulation, however, does not account for the standard uncomputation required in quantum circuits. To address this, Eq. (15) incorporates uncomputation, thereby mitigating the growth of the previously estimated cost. The term *clean cost* used in this equation follows the definition established ([8, 9, 11]).

$$\begin{aligned} T_n &= 51^{\log_{26} n} + 323n\left(\frac{1-(\frac{51}{26})^{\log_{26} n}}{1-(\frac{51}{26})}\right) \\ &= n^{\log_{26} 51} + 323n\left(\frac{1-n^{\log_{26}(\frac{51}{26})}}{1-(\frac{51}{26})}\right) \\ m &= 324n^{\log_{26} 51} - 333n \end{aligned} \tag{14}$$

$$T_{n(clean)} = 648n^{\log_{26} 51} - 666n \tag{15}$$

4.4. Space-Time Complexity Analysis

Bennett [16] introduced reversible pebble games to analyze asymptotic performance improvements, particularly in terms of space usage within space–time complexity analysis. This technique has since become a standard tool in reversible computing, enabling the evaluation of time–space trade-offs and facilitating time-efficient computation under finite-space constraints [17]. In this work, we employ this framework to compare the cost of the optimized multiplication scheme with prior results. The optimal multiplication cost is derived by adopting methodologies proposed by Parent et al. [18], Dutta et al. [3], Larasati et al. [8], Putranto et al. [9] and Wardhani et al. [10, 11]. The Toom–Cook 25.5-way algorithm performs 51 recursive multiplications, yielding a 26-ary tree structure. At level l, an input of size n consists of 51^l nodes, each of size $26^{-l}n$, resulting in

a total circuit cost of $n\left(\frac{51}{26}\right)^l$. The overall cost of this quinary tree is given by Eq. (16). The optimal tree height k for minimizing the cost can be determined using Eq. (18).

$$n\sum_{i=0}^{N}\left(\frac{51}{26}\right)^i, \quad N=\log_{26} n \tag{16}$$

$$n\sum_{i=0}^{N-k-1}\left(\frac{51}{26}\right)^i=\frac{1}{26^{N-k}}\sum_{i=0}^{k-1}\left(\frac{51}{26}\right)^i \tag{17}$$

Analogous to Eq. (15), the geometric series identity is used to derive the bounds in Eq. (18). This approach reduces the required space, as reflected in the qubit-count expression of Eq. (19). The resulting space complexity, approximately $\mathcal{O}(n^{1.176})$, is lower than the initial estimate given by Eq. (20), which is bounded by $\mathcal{O}(n^{1+\log_{26}\frac{51}{26}}) \approx \mathcal{O}(n^{1.206})$. Simplifying, we obtain a bound that Eq. (18). This is since $k \leq N$ and $(\frac{26}{51})^{N-k} \geq \frac{26^N}{51^k}$. Using the above technique, the qubit count is now optimized and bounded by QC:

$$k \leq \frac{N}{2-\frac{\log 26}{\log 51}} \approx 0.854 \tag{18}$$

$$QC=\mathcal{O}\left(n\left(\frac{51}{26}^{\left(\frac{\log 51}{2\log 51-2\log 26}\right)\log_{26} n}\right)\right)\approx \mathcal{O}(n^{1.176}) \tag{19}$$

$$n\sum_{k=0}^{\log_{51} n-1}\left(\frac{51}{26}\right)^k=n\left(\frac{1-\left(\frac{51}{26}\right)^{\log_{26} n}}{1-\frac{51}{26}}\right) \tag{20}$$

The Toffoli depth is a common measure of circuit time complexity [3, 19]. It is computed by multiplying the number of subtrees S_k at level k by the corresponding depth D_k, yielding the total Toffoli depth T_d as expressed in Eq. (21).

$$\begin{aligned} S_k &= 51^{\left(1-\frac{\log 51}{2\log 51-log 26}\right)\log_{26} n} \\ D_k &= \frac{n}{26^{\left(1-\frac{\log 51}{2\log 51-\log 26}\right)\log_{26} n}} \\ T_d &= S_k D_k = n\left(\frac{51}{26}\right)^{\left(1-\frac{\log 51}{2\log 51-\log 26}\right)\log_{26} n} \approx n^{1.03025} \end{aligned} \tag{21}$$

4.5. Complexity Analysis Comparison

Table 5 presents a comparative summary of asymptotic space–time complexity for several Toom–Cook–based quantum multiplication architectures. The evaluation is conducted using three standard metrics: qubit count, Toffoli gate count, and Toffoli depth. Alongside existing Toom–Cook variants of different degrees, the table includes the proposed Toom–Cook 25.5-way architecture to highlight its relative performance in terms of quantum resource requirements. The comparison covers prior studied Toom–Cook constructions, including the Toom–Cook 2-way [9], 2.5-way [3], 3-way [8], 4-way [9], 8-way ([6, 9]), 10.5 [10], and 20.5-way [11] designs reported in prior work. Analytical expressions for Toffoli gate count and Toffoli depth further confirm that the 25.5-way method outperforms earlier Toom–Cook variants, as well as classical baselines such as naive and Karatsuba multiplication. While naive multiplication scales as $\mathcal{O}(n^2)$ and Karatsuba achieves $\mathcal{O}(n^{\log_2 3})$, Toom–Cook–based approaches progressively reduce these costs

through recursive decomposition. Compared with previously reported quantum Toom–Cook implementations, the proposed high- and half-degree 25.5-way design tightens the asymptotic bounds across all evaluated metrics. Specifically, it achieves qubit, Toffoli gate, and depth complexities of $\mathcal{O}(n^{1.176})$, $\mathcal{O}(648n^{\log_{26} 51} - 666n)$, and $\mathcal{O}(n^{1.03025})$, respectively.

4.6. Discussion

Selecting the value of k involves a trade-off between asymptotic performance and resource overhead. Although larger k values improve asymptotic complexity, they also increase the number of subcomputations and interpolation steps, leading to more complex circuits. Consequently, Toom–Cook–based quantum multipliers must carefully consider gate selection, division-free computation strategies, and overall circuit architecture to achieve practical and resource-efficient implementations.

Practical Considerations. In exploring different values of k, we also examined higher-degree configurations, including variants beyond $k = 25.5$ (e.g., the $k = 50.5$ variant discussed by Bodrato [7]). However, both analytical and experimental results indicate that the $k = 20.5$ [11] and $k = 25.5$ configurations provide the most reliable and practically efficient performance under the considered design and resource constraints.

5. Conclusion

This work investigated Toom–Cook–based quantum multiplication using a range of design strategies and conducted a detailed analysis of the associated quantum resource costs. Our results demonstrate that the high- and half-degree Toom–Cook 25.5-way quantum polynomial multiplication achieves asymptotic bounds of $\mathcal{O}(n^{1.176})$ for qubit count, $\mathcal{O}(648n^{\log_{26} 51} - 666n)$ for Toffoli gate count, and $\mathcal{O}(n^{1.03025})$ for Toffoli depth. These findings establish tighter asymptotic bounds for high-degree quantum multiplication compared with existing approaches and provide a theoretically grounded foundation for efficient quantum arithmetic, which is essential for cryptographic constructions and other computation-intensive quantum applications.

CRediT Authorship Contribution Statement

Author One: Conceptualization, Methodology, Validation, Formal Analysis, Resources, Data Curation, Writing – Original Draft Preparation, Writing – Review & Editing, Project Administration, and Funding Acquisition. **Author Two**: Conceptualization, Methodology, Software, Investigation, Resources, Data Curation, and Visualization.

Declaration of Generative AI and AI-assisted technologies

The authors acknowledge the use of generative AI and AI-assisted technologies in the preparation of this manuscript. All intellectual contributions, interpretations, and final decisions regarding the content were made by the authors. The use of these technologies complied with institutional, ethical, and publication standards, and all AI-generated content was critically reviewed and edited to ensure accuracy, originality, and scholarly integrity.

Declaration of Competing Interest

The authors declare no competing interests.

Funding and Acknowledgments

This research was supported by the Politeknik Siber dan Sandi Negara (Poltek SSN) through its internal research grant scheme.

Code Availability

The source code used in this study are publicly available and can be accessed through an open repository at https://github.com/pcseyd-ai/toom-cook-25.5.

References

[1] Jose Maria Bermudo Mera, Angshuman Karmakar, and Ingrid Verbauwhede. "Time-memory trade-off in Toom-Cook multiplication: an application to module-lattice based cryptography". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2020.2 (2020), pp. 222–244. DOI: https://doi.org/10.13154/tches.v2020.i2.222-244.

[2] Siyi Wang, Xiufan Li, Wei Jie Bryan Lee, Suman Deb, Eugene Lim, and Anupam Chattopadhyay. "A comprehensive study of quantum arithmetic circuits". In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 383.2288 (2025). DOI: https://doi.org/10.1098/rsta.2023.0392.

[3] Srijit Dutta, Debjyoti Bhattacharjee, and Anupam Chattopadhyay. "Quantum circuits for Toom-Cook multiplication". In: *Physical Review A* 98.1 (2018), p. 012311. DOI: https://doi.org/10.1103/PhysRevA.98.012311.

[4] Andrei L Toom. "The complexity of a scheme of functional elements realizing the multiplication of integers". In: *Soviet Mathematics Doklady*. Vol. 3. 4. 1963, pp. 714–716. https://www.mathnet.ru/links/23f1aee5b47892b05da233b500953802/dan27978.pdf.

[5] Stephen A Cook and Stål O Aanderaa. "On the minimum computation time of functions". In: *Transactions of the American Mathematical Society* 142 (1969), pp. 291–314. DOI: https://doi.org/10.2307/1995359.

[6] Alberto Zanoni. "Toom-cook 8-way for long integers multiplication". In: *2009 11th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*. IEEE. 2009, pp. 54–57. DOI: 10.1109/SYNASC.2009.23.

[7] Marco Bodrato. "High degree Toom'n'half for balanced and unbalanced multiplication". In: *2011 IEEE 20th Symposium on Computer Arithmetic*. IEEE. 2011, pp. 15–22. DOI: 10.1109/ARITH.2011.12.

[8] Harashta Tatimma Larasati, Asep Muhamad Awaludin, Janghyun Ji, and Howon Kim. "Quantum circuit design of toom 3-way multiplication". In: *Applied Sciences* 11.9 (2021), p. 3752. DOI: https://doi.org/10.3390/app11093752.

[9] Dedy Septono Catur Putranto, Rini Wisnu Wardhani, Harashta Tatimma Larasati, and Howon Kim. "Space and Time-Efficient Quantum Multiplier in Post Quantum Cryptography Era". In: *IEEE Access* 11 (2023), pp. 21848–21862. DOI: 10.1109/ACCESS.2023.3252504.

[10] Rini Wisnu Wardhani, Dedy Septono Catur Putranto, and Howon Kim. "Quantum circuits for high-degree and half-multiplication for post-quantum analysis". In: *International Conference on Information Security and Cryptology*. Springer. 2023, pp. 140–160. DOI: https://doi.org/10.1007/978-981-97-1235-9_8.

[11] Rini Wisnu Wardhani, Dedy Septono Catur Putranto, and Howon Kim. "High-and half-degree quantum multiplication for post-quantum security evaluation". In: *IEEE Access* 12 (2024), pp. 8806–8821. DOI: 10.1109/ACCESS.2024.3352157.

[12] Marco Bodrato and Alberto Zanoni. "Integer and polynomial multiplication: Towards optimal Toom-Cook matrices". In: *Proceedings of the 2007 international symposium on Symbolic and algebraic computation*. 2007, pp. 17–24. DOI: https://doi.org/10.1145/1277548.1277552.

[13] Zhen Gu and Shuguo Li. "A Division-Free Toom–Cook Multiplication-Based Montgomery Modular Multiplication". In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 66.8 (2018), pp. 1401–1405. DOI: 10.1109/TCSII.2018.2886962.

[14] Marco Bodrato. "Towards optimal Toom-Cook multiplication for univariate and multivariate polynomials in characteristic 2 and 0". In: *International Workshop on the Arithmetic of Finite Fields*. Springer. 2007, pp. 116–133. DOI: https://doi.org/10.1007/978-3-540-73074-3_10.

[15] Steven A Cuccaro, Thomas G Draper, Samuel A Kutin, and David Petrie Moulton. "A new quantum ripple-carry addition circuit". In: *arXiv preprint quant-ph/0410184* (2004). DOI: https://doi.org/10.48550/arXiv.quant-ph/0410184.

[16] Charles H Bennett. "Time/space trade-offs for reversible computation". In: *SIAM Journal on Computing* 18.4 (1989), pp. 766–776. DOI: https://doi.org/10.1137/0218053.

[17] Richard Král'ovič. "Time and space complexity of reversible pebbling". In: *International Conference on Current Trends in Theory and Practice of Computer Science*. Springer. 2001, pp. 292–303. DOI: https://doi.org/10.1007/3-540-45627-9_26.

[18] Alex Parent, Martin Roetteler, and Michele Mosca. "Improved reversible and quantum circuits for Karatsuba-based integer multiplication". In: *12th Conference on the Theory of Quantum Computation, Communication, and Cryptography (TQC 2017)*. Springer. 2017, 7:1–7:15. DOI: https://doi.org/10.48550/arXiv.1706.03419.

[19] Matthew Amy. "Algorithms for the optimization of quantum circuits". MA thesis. University of Waterloo, 2013. http://hdl.handle.net/10012/7818.

[20] Thomas G Draper, Samuel A Kutin, Eric M Rains, and Krysta M Svore. "A logarithmic-depth quantum carry-lookahead adder". In: *Quantum Information and Computation* 6.4 (2006). DOI: 10.26421/QIC6.4-5-4.

[21] Archimedes Pavlidis and Dimitris Gizopoulos. "Fast quantum modular exponentiation architecture for Shor's factorization algorithm". In: *Quantum Information and Computation* 14.7 & 8 (2012). DOI: https://doi.org/10.48550/arXiv.1207.0511.

[22] Shane Kepley and Rainer Steinwandt. "Quantum circuits for F2n-multiplication with subquadratic gate count". In: *Quantum Information Processing* 14.7 (2015), pp. 2373–2386. DOI: https://doi.org/10.1007/s11128-015-0993-1.

[23] Iggy van Hoof. "Space-efficient quantum multiplication polynomials for binary finite fields with sub−quadratic Toffoli gate count". In: *Quantum Inf.Comput.* 20.9&10 (2020), pp. 721–735. DOI: 10.26421/QIC20.9-10-1.

[24] Dedy Septono Catur Putranto, Rini Wisnu Wardhani, Harashta Tatimma Larasati, Janghyun Ji, and Howon Kim. "Depth-optimization of Quantum Cryptanalysis on Binary Elliptic Curves". In: *IEEE Access* (2023). DOI: 10.1109/ACCESS.2023.3273601.

Appendices

$$
\begin{pmatrix} YY \\ YX \\ YW \\ YV \\ YU \\ YT \\ YS \\ YR \\ YQ \\ YP \\ YO \\ YN \\ YM \\ YL \\ YK \\ YJ \\ YI \\ YH \\ YG \\ YF \\ YE \\ YD \\ YC \\ YB \\ YA \\ XZ \\ XY \\ XX \\ XW \\ XV \\ XU \\ XT \\ XS \\ XR \\ XQ \\ XP \\ XO \\ XN \\ XM \\ XL \\ XK \\ XJ \\ XI \\ XH \\ XG \\ XF \\ XE \\ XD \\ XC \\ XB \\ XA \end{pmatrix}
=
\begin{pmatrix}
1 & 0 & \cdots & 0 & 0 \\
\vdots & \cdots & \cdots & \cdots & \vdots \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
\vdots & \ddots & \ddots & \ddots & \vdots \\
\vdots & \cdots & \cdots & \cdots & \vdots \\
0 & 0 & \cdots & 0 & 1
\end{pmatrix}^{-1}
\begin{pmatrix} A \\ B \\ C \\ D \\ E \\ F \\ G \\ H \\ I \\ J \\ K \\ L \\ M \\ N \\ O \\ P \\ Q \\ R \\ S \\ T \\ U \\ V \\ W \\ X \\ Y \\ Z \\ AA \\ AB \\ AC \\ AD \\ AE \\ AF \\ AG \\ AH \\ AI \\ AJ \\ AK \\ AL \\ AM \\ AN \\ AO \\ AP \\ AQ \\ AR \\ AS \\ AT \\ AU \\ AV \\ AW \\ AX \\ AY \end{pmatrix}
\tag{22}
$$

Table 3: Parameters Selected for the Evaluation Stage Used in This Work

Index $t(x)$	**Calculation**	**Parameter Used**
1	0 (replacement for ∞)	0
2	2^0	1
3	Opposite of 2^0	-1
4	2^1	2
5	Opposite of 2^1	-2
6	2^2	4
7	Opposite of 2^2	-4
8	$1/2^1$	0.5
9	Opposite of $1/2^1$	-0.5
10	$1/2^2$	0.25
11	Opposite of $1/2^2$	-0.25
12	$1/2^3$	0.125
13	Opposite of $1/2^3$	-0.125
14	$1/2^4$	0.0625
15	Opposite of $1/2^4$	-0.0625
16	$1/2^5$	0.03125
17	Opposite of $1/2^5$	-0.03125
18	$1/2^6$	0.015625
19	Opposite of $1/2^6$	-0.015625
20	$1/2^7$	0.0078125
21	Opposite of $1/2^7$	-0.0078125
22	$1/2^8$	0.00390625
23	Opposite of $1/2^8$	-0.00390625
24	$1/2^9$	0.001953125
25	Opposite of $1/2^9$	-0.001953125
26	$1/2^{10}$	0.0009765625
27	Opposite of $1/2^{10}$	-0.0009765625
28	$1/2^{11}$	0.00048828125
29	Opposite of $1/2^{11}$	-0.00048828125
30	$1/2^{12}$	0.000244140625
31	Opposite of $1/2^{12}$	-0.000244140625
32	$1/2^{13}$	0.0001220703125
33	Opposite of $1/2^{13}$	-0.0001220703125
34	$1/2^{14}$	0.00006103515625
35	Opposite of $1/2^{14}$	-0.00006103515625
36	$1/2^{15}$	0.000030517578125
37	Opposite of $1/2^{15}$	-0.000030517578125
38	$1/2^{16}$	0.0000152587890625
39	Opposite of $1/2^{16}$	$-.0000152587890625$
40	$1/2^{17}$	0.00000762939453125
41	Opposite of $1/2^{17}$	-0.00000762939453125
42	$1/2^{18}$	0.000003814697265625
43	Opposite of $1/2^{18}$	-0.000003814697265625
44	$1/2^{19}$	0.0000019073486328125
45	Opposite of $1/2^{19}$	-0.0000019073486328125
46	$1/2^{20}$	0.00000095367431640625
47	Opposite of $1/2^{20}$	-0.00000095367431640625
48	$1/2^{21}$	0.000000476837158203125
49	Opposite of $1/2^{21}$	-0.000000476837158203125
50	$1/2^{22}$	0.00000023841857910156 2
51	Opposite of $1/2^{22}$	$-0.00000023841857910156 2$

Table 4: The Evaluation Multiplication

A = $x0$	* $y0$
B = $(((x0*1+x1*1+x2*1+x3*1+x4*1+x5*1+x6*1+x7*1+x8*1+x9*1+x10*1+x11*1+x12*1+x13*1+x14*1+x15*1+x16*1+x17*1+x18*1+x19*1+x20*1+x21*1+x22*1+x23*1+x24*1))$	* $((y0*1+y1*1+y2*1+y3*1+y4*1+y5*1+y6*1+y7*1+y8*1+y9*1+y10*1+y11*1+y12*1+y13*1+y14*1+y15*1+y16*1+y17*1+y18*1+y19*1+y20*1+y21*1+y22*1+y23*1+y24*1+y25*1))$
C = $(((x0*1+x1*-1+x2*1+x3*-1+x4*1+x5*-1+x6*1+x7*-1+x8*1+x9*-1+x10*1+x11*-1+x12*1+x13*-1+x14*1+-1+x14*1+x15*-1+x16*1+x17*-1+x18*1+x19*-1+x20*1+x21*-1+x22*1+x23*-1+x24*1))$	* $((y0*1+y1*-1+y2*1+y3*-1+y4*1+y5*-1+y6*1+y7*-1+y8*1+y9*-1+y10*1+y11*-1+y12*1+y13*-1+y14*1+y15*-1+y16*1+y17*-1+y18*1+y19*-1+y20*1+y21*-1+y22*1+y23*-1+y24*1+y25*-1))$
D = $(((x0*1+x1*2+x2*4+x3*8+x4*16+x5*32+x6*64+x7*128+x8*256+x9*512+x10*1024+x11*2048+x12*4096+x13*8192+x14*16384+x15*32768+x16*65536+x17*131072+x18*262144+x19*524288+x20*1048576+x21*2097152+x22*4194304+x23*8388608+x24*16777216))$	* $((y0*1+y1*2+y2*4+y3*8+y4*16+y5*32+y6*64+y7*128+y8*256+y9*512+y10*1024+y11*2048+y12*4096+y13*8192+y14*16384+y15*32768+y16*65536+y17*131072+y18*262144+y19*524288+y20*1048576+y21*2097152+y22*4194304+y23*8388608+y24*16777216+y25*33554432))$
E = $(((x0*1+x1*-2+x2*4+x3*-8+x4*16+x5*-32+x6*64+x7*-128+x8*256+x9*-512+x10*1024+x11*-2048+x12*4096+x13*-8192+x14*16384+x15*-32768+x16*65536+x17*-131072+x18*262144+x19*-524288+x20*1048576+x21*-2097152+x22*4194304+x23*-8388608+x24*16777216))$	* $((y0*1+y1*-2+y2*4+y3*-8+y4*16+y5*-32+y6*64+y7*-128+y8*256+y9*-512+y10*1024+y11*-2048+y12*4096+y13*-8192+y14*16384+y15*-32768+y16*65536+y17*-131072+y18*262144+y19*-524288+y20*1048576+y21*-2097152+y22*4194304+y23*-8388608+y24*16777216+y25*-33554432))$
F = $(((x0*1+x1*4+x2*16+x3*64+x4*256+x5*1024+x6*4096+x7*16384+x8*65536+x9*262144+x10*1048576+x11*4194304+x12*16777216+x13*67108864+x14*268435456+x15*1073741824+x16*4294967296+x17*17179869184+x18*68719476736+x19*274877906944+x20*1099511627776+x21*4398046511104+x22*17592186044416+x23*70368744177664+x24*281474976710656))$	* $((y0*1+y1*4+y2*16+y3*64+y4*256+y5*1024+y6*4096+y7*16384+y8*65536+y9*262144+y10*1048576+y11*4194304+y12*16777216+y13*67108864+y14*268435456+y15*1073741824+y16*4294967296+y17*17179869184+y18*68719476736+y19*274877906944+y20*1099511627776+y21*4398046511104+y22*17592186044416+y23*70368744177664+y24*281474976710656+y25*1125899906842620))$
G = $(((x0*1+x1*-4+x2*16+x3*-64+x4*256+x5*-1024+x6*4096+x7*-16384+x8*65536+x9*-262144+x10*1048576+x11*-4194304+x12*16777216+x13*-67108864+x14*268435456+x15*-1073741824+x16*4294967296+x17*-17179869184+x18*68719476736+x19*-274877906944+x20*1099511627776+x21*-4398046511104+x22*17592186044416+x23*-70368744177664+x24*281474976710656))$	* $((y0*1+y1*-4+y2*16+y3*-64+y4*256+y5*-1024+y6*4096+y7*-16384+y8*65536+y9*-262144+y10*1048576+y11*-4194304+y12*16777216+y13*-67108864+y14*268435456+y15*-1073741824+y16*4294967296+y17*-17179869184+y18*68719476736+y19*-274877906944+y20*1099511627776+y21*-4398046511104+y22*17592186044416+y23*-70368744177664+y24*281474976710656+y25*-1125899906842620))$
H = $(((x0*1+x1*8+x2*64+x3*512+x4*4096+x5*32768+x6*262144+x7*2097152+x8*16777216+x9*134217728+x10*1073741824+x11*8589934592+x12*68719476736+x13*549755813888+x14*4398046511104+x15*35184372088832+x16*281474976710656+x17*2251799813685250+x18*18014398509482000+x19*144115188075856000+x20*1152921504606850000+x21*9223372036854780000+x22*73786976294838200000+x23*590295810358706E+20+x24*4722366482869E+21))$	* $((y0*1+y1*8+y2*64+y3*512+y4*4096+y5*32768+y6*262144+y7*2097152+y8*16777216+y9*134217728+y10*1073741824+y11*8589934592+y12*68719476736+y13*549755813888+y14*4398046511104+y15*35184372088832+y16*281474976710656+y17*2251799813685250+y18*18014398509482000+y19*144115188075856000+y20*1152921504606850000+y21*9223372036854780000+y22*73786976294838200000+y23*590295810358706E+20+y24*4722366482869E+21+y25*37778931862957E+22))$
I = $(((x0*1+x1*-8+x2*64+x3*-512+x4*4096+x5*-32768+x6*262144+x7*-2097152+x8*16777216+x9*-134217728+x10*1073741824+x11*-8589934592+x12*68719476736+x13*-549755813888+x14*4398046511104+x15*-35184372088832+x16*281474976710656+x17*-2251799813685250+x18*18014398509482000+x19*-144115188075856000+x20*1152921504606850000+x21*-9223372036854780000+x22*73786976294838200000+x23*-590295810358706E+20+x24*4722366482869E+21))$	* $((y0*1+y1*-8+y2*64+y3*-512+y4*4096+y5*-32768+y6*262144+y7*-2097152+y8*16777216+y9*-134217728+y10*1073741824+y11*-8589934592+y12*68719476736+y13*-549755813888+y14*4398046511104+y15*-35184372088832+y16*281474976710656+y17*-2251799813685250+y18*18014398509482000+y19*-144115188075856000+y20*1152921504606850000+y21*-9223372036854780000+y22*73786976294838200000+y23*-590295810358706E+20+y24*4722366482869E+21+y25*-37778931862957E+22))$
J = $(((x0*1+x1*16+x2*256+x3*4096+x4*65536+x5*1048576+x6*16777216+x7*268435456+x8*4294967296+x9*68719476736+x10*1099511627776+x11*17592186044416+x12*281474976710656+x13*4503599627370500+x14*72057594037927900+x15*1152921504606850000+x16*18446744073709600000+x17*295147905179353E+20+x18*4722366482869E+21+x19*75557863725914E+22+x20*1208925819614636E+24+x21*19342813113834E+25+x22*309485009821345E+26+x23*4951760157141552E+27+x24*79228162514264438E+28))$	* $((y0*1+y1*16+y2*256+y3*4096+y4*65536+y5*1048576+y6*16777216+y7*268435456+y8*4294967296+y9*68719476736+y10*1099511627776+y11*17592186044416+y12*281474976710656+y13*4503599627370500+y14*72057594037927900+y15*1152921504606850000+y16*18446744073709600000+y17*295147905179353E+20+y18*4722366482869E+21+y19*75557863725914E+22+y20*1208925819614636E+24+y21*19342813113834E+25+y22*309485009821345E+26+y23*4951760157141552E+27+y24*79228162514264438E+28+y25*126765600022823E+30))$
K = $(((x0*1+x1*-16+x2*256+x3*-4096+x4*65536+x5*-1048576+x6*16777216+x7*-268435456+x8*4294967296+x9*-68719476736+x10*1099511627776+x11*-17592186044416+x12*281474976710656+x13*-4503599627370500+x14*72057594037927900+x15*-1152921504606850000+x16*18446744073709600000+x17*-295147905179353E+20+x18*4722366482869E+21+x19*-75557863725914E+22+x20*1208925819614636E+24+x21*-19342813113834E+25+x22*309485009821345E+26+x23*-4951760157141552E+27+x24*79228162514264438E+28))$	* $((y0*1+y1*-16+y2*256+y3*-4096+y4*65536+y5*-1048576+y6*16777216+y7*-268435456+y8*4294967296+y9*-68719476736+y10*1099511627776+y11*-17592186044416+y12*281474976710656+y13*-4503599627370500+y14*72057594037927900+y15*-1152921504606850000+y16*18446744073709600000+y17*-295147905179353E+20+y18*4722366482869E+21+y19*-75557863725914E+22+y20*1208925819614636E+24+y21*-19342813113834E+25+y22*309485009821345E+26+y23*-4951760157141552E+27+y24*79228162514264438E+28+y25*1267656000228323E+30))$
L = $(((x0*1+x2+x2*1024+x3*32768+x4*1048576+x5*33554432+x6*1073741824+x7*34359738368+x8*1099511627776+x9*35184372088832+x10*1125899906842620+x11*36028797018964000+x12*1152921504606850000+x13*36893488147419100000+x14*1180591620717414E+21+x15*37778931862957E+22+x16*1208925819614636E+24+x17*38685626226681E+25+x18*1237940039285384E+27+x19*39614081257132E+28+x20*1267650600228323E+30+x21*40564819207033E+31+x22*1298074214633714E+33+x23*41538374868278E+34+x24*1329227995784926E+36))$	* $((y0*1+y1*32+y2*1024+y3*32768+y4*1048576+y5*33554432+y6*1073741824+y7*34359738368+y8*1099511627776+y9*35184372088832+y10*1125899906842620+y11*36028797018964000+y12*1152921504606850000+y13*36893488147419100000+y14*1180591620717414E+21+y15*37778931862957E+22+y16*1208925819614636E+24+y17*38685626226681E+25+y18*1237940039285384E+27+y19*39614081257132E+28+y20*1267650600228323E+30+y21*40564819207033E+31+y22*1298074214633714E+33+y23*41538374868278E+34+y24*1329227995784926E+36+y25*42535295865113E+37))$
M = $(((x0*1+x1*-32+x2*1024+x3*-32768+x4*1048576+x5*-33554432+x6*1073741824+x7*-34359738368+x8*1099511627776+x9*-35184372088832+x10*1125899906842620+x11*-36028797018964000+x12*1152921504606850000+x13*-36893488147419100000+x14*1180591620717414E+21+x15*-37778931862957E+22+x16*1208925819614636E+24+x17*-38685626226681E+25+x18*1237940039285384E+27+x19*-39614081257132E+28+x20*1267650600228323E+30+x21*-40564819207033E+31+x22*1298074214633714E+33+x23*-41538374868278E+34+x24*1329227995784926E+36)$	* $((y0*1+y1*-32+y2*1024+y3*-32768+y4*1048576+y5*-33554432+y6*1073741824+y7*-34359738368+y8*1099511627776+y9*-35184372088832+y10*1125899906842620+y11*-36028797018964000+y12*1152921504606850000+y13*-36893488147419100000+y14*1180591620717414E+21+y15*-37778931862957E+22+y16*1208925819614636E+24+y17*-38685626226681E+25+y18*1237940039285384E+27+y19*-39614081257132E+28+y20*1267650600228323E+30+y21*-40564819207033E+31+y22*1298074214633714E+33+y23*-41538374868278E+34+y24*1329227995784926E+36+y25*-42535295865113E+37))$
N = $(((x0*1+x1*64+x2*4096+x3*262144+x4*16777216+x5*1073741824+x6*68719476736+x7*4398046511104+x8*281474976710656+x9*18014398509482000+x10*1152921504606850000+x11*73786976294838200000+x12*4722366482869E+21+x13*302231454903657E+23+x14*19342813113834E+25+x15*1237940039285384E+27+x16*79228162514264438E+28+x17*5070602400912292E+30+x18*324518553658427E+32+x19*20769187434139E+34+x20*1329227995784926E+36+x21*85070591730234E+37+x22*5444517870735E+39+x23*348491437270414E+41+x24*22300745198536E+43))$	* $((y0*1+y1*64+y2*4096+y3*262144+y4*16777216+y5*1073741824+y6*68719476736+y7*4398046511104+y8*281474976710656+y9*18014398509482000+y10*1152921504606850000+y11*73786976294838200000+y12*4722366482869E+21+y13*302231454903657E+23+y14*19342813113834E+25+y15*1237940039285384E+27+y16*79228162514264438E+28+y17*5070602400912292E+30+y18*324518553658427E+32+y19*20769187434139E+34+y20*1329227995784926E+36+y21*85070591730234E+37+y22*5444517870735E+39+y23*348491437270414E+41+y24*22300745198536E+43+y25*1427247692705596E+45))$
O = $(((x0*1+x1*-64+x2*4096+x3*-262144+x4*16777216+x5*-1073741824+x6*68719476736+x7*-4398046511104+x8*281474976710656+x9*-18014398509482000+x10*1152921504606850000+x11*-73786976294838200000+x12*4722366482869E+21+x13*-302231454903657E+23+x14*19342813113834E+25+x15*-1237940039285384E+27+x16*79228162514264438E+28+x17*5070602400912292E+30+x18*324518553658427E+32+x19*20769187434139E+34+x20*1329227995784926E+36+x21*-85070591730234E+37+x22*5444517870735E+39+x23*348491437270414E+41+x24*22300745198536E+43)$	* $((y0*1+y1*-64+y2*4096+y3*-262144+y4*16777216+y5*-1073741824+y6*68719476736+y7*-4398046511104+y8*281474976710656+y9*-18014398509482000+y10*1152921504606850000+y11*-73786976294838200000+y12*4722366482869E+21+y13*-302231454903657E+23+y14*19342813113834E+25+y15*-1237940039285384E+27+y16*79228162514264438E+28+y17*5070602400912292E+30+y18*324518553658427E+32+y19*20769187434139E+34+y20*1329227995784926E+36+y21*-85070591730234E+37+y22*5444517870735E+39+y23*348491437270414E+41+y24*22300745198536E+43+y25*1427247692705596E+45))$
P = $(((x0*1+x1*128+x2*16384+x3*2097152+x4*268435456+x5*34359738368+x6*4398046511104+x7*562949953421312+x8*72057594037927900+x9*9223372036854780000+x10*1180591620717414E+21+x11*151115727451829E+23+x12*19342813113834E+25+x13*2475880078570076E+27+x14*316912650057057E+29+x15*40564819207033E+31+x16*519229685853483E+33+x17*664613997892458E+35+x18*85070591730234E+37+x19*10889035741478E+40+x20*1393796574908616E+42+x21*178405961588245E+44+x22*22835963083929E+46+x23*2923003274661815E+48+x24*374144419156711E+50))$	* $((y0*1+y1*128+y2*16384+y3*2097152+y4*268435456+y5*34359738368+y6*4398046511104+y7*562949953421312+y8*72057594037927900+y9*9223372036854780000+y10*1180591620717414E+21+y11*151115727451829E+23+y12*19342813113834E+25+y13*2475880078570076E+27+y14*316912650057057E+29+y15*40564819207033E+31+y16*5192296858534833E+33+y17*664613997892458E+35+y18*85070591730234E+37+y19*10889035741478E+40+y20*1393796574908616E+42+y21*178405961588245E+44+y22*22835963083929E+46+y23*2923003274661815E+48+y24*374144419156711E+50+y25*4789048565205094E+52))$
Q = $(((x0*1+x1*-128+x2*16384+x3*-2097152+x4*268435456+x5*-34359738368+x6*4398046511104+x7*-562949953421312+x8*72057594037927900+x9*-9223372036854780000+x10*1180591620717414E+21+x11*-151115727451829E+23+x12*19342813113834E+25+x13*-2475880078570076E+27+x14*316912650057057E+29+x15*-40564819207033E+31+x16*5192296858534833E+33+x17*-664613997892458E+35+x18*85070591730234E+37+x19*-10889035741478E+40+x20*1393796574908616E+42+x21*-178405961588245E+44+x22*22835963083929E+46+x23*-2923003274661815E+48+x24*374144419156711E+50)$	* $((y0*1+y1*-128+y2*16384+y3*-2097152+y4*268435456+y5*-34359738368+y6*4398046511104+y7*-562949953421312+y8*72057594037927900+y9*-9223372036854780000+y10*1180591620717414E+21+y11*-151115727451829E+23+y12*19342813113834E+25+y13*-2475880078570076E+27+y14*316912650057057E+29+y15*-40564819207033E+31+y16*5192296858534833E+33+y17*-664613997892458E+35+y18*85070591730234E+37+y19*-10889035741478E+40+y20*1393796574908616E+42+y21*-178405961588245E+44+y22*22835963083929E+46+y23*-2923003274661815E+48+y24*374144419156711E+50+y25*4789048565205094E+52))$
R = $(((x0*1+x2+256+x2*65536+x3*16777216+x4*4294967296+x5*1099511627776+x6*281474976710656+x7*72057594037927900+x8*18446744073709600000+x9*4722366482869E+21+x10*1208925819614636E+24+x11*309485009821345E+26+x12*79228162514264438E+28+x13*20284090603657E+31+x14*5192296858534833E+33+x15*1329227995784926E+36+x16*340282366920938E+38+x17*87112285931760216E+40+x18*22300745198536E+43+x19*5708990770823864E+45+x20*1461501637309E+48+x21*374144419156711E+50+x22*95780971304118E+52+x23*24519928653854E+55+x24*6.2771017353866E+57))$	* $((y0*1+y1*256+y2*65536+y3*16777216+y4*4294967296+y5*1099511627776+y6*281474976710656+y7*72057594037927900+y8*18446744073709600000+y9*4722366482869E+21+y10*1208925819614636E+24+y11*309485009821345E+26+y12*79228162514264438E+28+y13*20284090603657E+31+y14*5192296858534833E+33+y15*1329227995784926E+36+y16*340282366920938E+38+y17*87112285931760216E+40+y18*22300745198536E+43+y19*5708990770823864E+45+y20*1461501637309E+48+y21*374144419156711E+50+y22*95780971304118E+52+y23*24519928653854E+55+y24*6.2771017353866E+57+y25*1.6069380442589E+60))$
S = $(((x0*1+x1*-256+x2*65536+x3*-16777216+x4*4294967296+x5*-1099511627776+x6*281474976710656+x7*-72057594037927900+x8*18446744073709600000+x9*-4722366482869E+21+x10*1208925819614636E+24+x11*-309485009821345E+26+x12*79228162514264438E+28+x13*-20284090603657E+31+x14*5192296858534833E+33+x15*-1329227995784926E+36+x16*340282366920938E+38+x17*-87112285931760216E+40+x18*22300745198536E+43+x19*-5708990770823864E+45+x20*1461501637309E+48+x21*-374144419156711E+50+x22*95780971304118E+52+x23*-24519928653854E+55+x24*6.2771017353866E+57)$	* $((y0*1+y1*-256+y2*65536+y3*-16777216+y4*4294967296+y5*-1099511627776+y6*281474976710656+y7*-72057594037927900+y8*18446744073709600000+y9*-4722366482869E+21+y10*1208925819614636E+24+y11*-309485009821345E+26+y12*79228162514264438E+28+y13*-20284090603657E+31+y14*5192296858534833E+33+y15*1329227995784926E+36+y16*340282366920938E+38+y17*87112285931760216E+40+y18*22300745198536E+43+y19*5708990770823864E+45+y20*1461501637309E+48+y21*-374144419156711E+50+y22*95780971304118E+52+y23*-24519928653854E+55+y24*6.2771017353866E+57+y25*-5.3919893334301E+59))$
T = $(((x0*1+x1*512+x2*262144+x3*134217728+x4*68719476736+x5*35184372088832+x6*18014398509482000+x7*9223372036854780000+x8*4722366482869E+21+x9*2417851639229026E+24+x10*1237940039285384E+27+x11*633825300114115E+29+x12*324518553658427E+32+x13*166153499473114E+35+x14*85070591730234E+37+x15*43566129658801E+40+x16*22300745198536E+43+x17*11417981541647E+46+x18*5846006549323661E+48+x19*2993155535325369E+51+x20*1532495540865894E+54+x21*784637716923335E+56+x22*401734511064748E+59+x23*205688069665151E+62+x24*105312291668557E+65))$	* $((y0*1+y1*512+y2*262144+y3*134217728+y4*68719476736+y5*35184372088832+y6*18014398509482000+y7*9223372036854780000+y8*4722366482869E+21+y9*2417851639229026E+24+y10*1237940039285384E+27+y11*633825300114115E+29+y12*324518553658427E+32+y13*166153499473114E+35+y14*85070591730234E+37+y15*43566129658801E+40+y16*22300745198536E+43+y17*11417981541647E+46+y18*5846006549323661E+48+y19*2993155535325369E+51+y20*1532495540865894E+54+y21*784637716923335E+56+y22*401734511064748E+59+y23*205688069665151E+62+y24*105312291668557E+65+y25*53919893334301E+67))$
U = $(((x0*1+x1*-512+x2*262144+x3*-134217728+x4*68719476736+x5*-35184372088832+x6*18014398509482000+x7*-9223372036854780000+x8*4722366482869E+21+x9*-2417851639229026E+24+x10*1237940039285384E+27+x11*-633825300114115E+29+x12*324518553658427E+32+x13*-166153499473114E+35+x14*85070591730234E+37+x15*-43566129658801E+40+x16*22300745198536E+43+x17*-11417981541647E+46+x18*5846006549323661E+48+x19*-2993155535325369E+51+x20*1532495540865894E+54+x21*-784637716923335E+56+x22*401734511064748E+59+x23*-205688069665151E+62+x24*105312291668557E+65)$	* $((y0*1+y1*-512+y2*262144+y3*-134217728+y4*68719476736+y5*-35184372088832+y6*18014398509482000+y7*-9223372036854780000+y8*4722366482869E+21+y9*-2417851639229026E+24+y10*1237940039285384E+27+y11*-633825300114115E+29+y12*324518553658427E+32+y13*-166153499473114E+35+y14*85070591730234E+37+y15*-43566129658801E+40+y16*22300745198536E+43+y17*-11417981541647E+46+y18*5846006549323661E+48+y19*-2993155535325369E+51+y20*1532495540865894E+54+y21*-784637716923335E+56+y22*401734511064748E+59+y23*-205688069665151E+62+y24*105312291668557E+65+y25*-53919893334301E+67))$
V = $(((x0*1+x1*1024+x2*1048576+x3*1073741824+x4*1099511627776+x5*1125899906842620+x6*1152921504606850000+x7*1180591620717414E+21+x8*1208925819614636E+24+x9*1237940039285384E+27+x10*1267650600228323E+30+x11*1298074214633714E+33+x12*1329227995784926E+36+x13*1361129467683575E+39+x14*1393796574908616E+42+x15*1427247692705596E+45+x16*1461501637309E+48+x17*1495577676626684E+51+x18*1532495540865894E+54+x19*1569275433846667E+57+x20*1606938044258999E+60+x21*1645504573212161E+63+x22*1684996666969691E+66+x23*1725436586697164E+69+x24*1768470647783832E+72))$	* $((y0*1+y1*1024+y2*1048576+y3*1073741824+y4*1099511627776+y5*1125899906842620+y6*1152921504606850000+y7*1180591620717414E+21+y8*1208925819614636E+24+y9*1237940039285384E+27+y10*1267650600228323E+30+y11*1298074214633714E+33+y12*1329227995784926E+36+y13*1361129467683575E+39+y14*1393796574908616E+42+y15*1427247692705596E+45+y16*1461501637309E+48+y17*1495577676626684E+51+y18*1532495540865894E+54+y19*1569275433846667E+57+y20*1606938044258999E+60+y21*1645504573212161E+63+y22*1684996666969691E+66+y23*1725436586697164E+69+y24*1768470647783832E+72+y25*1809251394333077E+75))$
W = $(((x0*1+x1*-1024+x2*1048576+x3*-1073741824+x4*1099511627776+x5*-1125899906842620+x6*1152921504606850000+x7*-1180591620717414E+21+x8*1208925819614636E+24+x9*-1237940039285384E+27+x10*1267650600228323E+30+x11*-1298074214633714E+33+x12*1329227995784926E+36+x13*-1361129467683575E+39+x14*1393796574908616E+42+x15*-1427247692705596E+45+x16*1461501637309E+48+x17*-1495577676626684E+51+x18*1532495540865894E+54+x19*-1569275433846667E+57+x20*1606938044258999E+60+x21*-1645504573212161E+63+x22*1684996666969691E+66+x23*-1725436586697164E+69+x24*1768470647783832E+72)$	* $((y0*1+y1*-1024+y2*1048576+y3*-1073741824+y4*1099511627776+y5*-1125899906842620+y6*1152921504606850000+y7*-1180591620717414E+21+y8*1208925819614636E+24+y9*-1237940039285384E+27+y10*1267650600228323E+30+y11*-1298074214633714E+33+y12*1329227995784926E+36+y13*-1361129467683575E+39+y14*1393796574908616E+42+y15*-1427247692705596E+45+y16*1461501637309E+48+y17*-1495577676626684E+51+y18*1532495540865894E+54+y19*-1569275433846667E+57+y20*1606938044258999E+60+y21*-1645504573212161E+63+y22*1684996666969691E+66+y23*-1725436586697164E+69+y24*1768470647783832E+72+y25*-1809251394333077E+75))$
X = $(((x0*1+x1*2048+x2*4194304+x3*8589934592+x4*17592186044416+x5*36028797018964000+x6*73786976294838200000+x7*151115727451829E+23+x8*309485009821345E+26+x9*633825300114115E+29+x10*1298074214633714E+33+x11*2658455991569831E+36+x12*5444517870735E+39+x13*11150372599265E+43+x14*22835963083929E+46+x15*46768052394588E+49+x16*95780971304118E+52+x17*196159429230834E+56+x18*401734511064748E+59+x19*822752278660603E+62+x20*1684996666969691E+66+x21*3450873733925E+69+x22*7067388259113454E+72+x23*14474011154664E+76+x24*29642774844752E+79))$	* $((y0*1+y1*2048+y2*4194304+y3*8589934592+y4*17592186044416+y5*36028797018964000+y6*73786976294838200000+y7*151115727451829E+23+y8*309485009821345E+26+y9*633825300114115E+29+y10*1298074214633714E+33+y11*2658455991569831E+36+y12*5444517870735E+39+y13*11150372599265E+43+y14*22835963083929E+46+y15*46768052394588E+49+y16*95780971304118E+52+y17*196159429230834E+56+y18*401734511064748E+59+y19*822752278660603E+62+y20*1684996666969691E+66+y21*3450873733925E+69+y22*7067388259113454E+72+y23*14474011154664E+76+y24*29642774844752E+79+y25*60708402882054E+82))$
Y = $(((x0*1+x1*-2048+x2*4194304+x3*-8589934592+x4*17592186044416+x5*-36028797018964000+x6*73786976294838200000+x7*-151115727451829E+23+x8*309485009821345E+26+x9*-633825300114115E+29+x10*1298074214633714E+33+x11*-2658455991569831E+36+x12*5444517870735E+39+x13*-11150372599265E+43+x14*22835963083929E+46+x15*-46768052394588E+49+x16*95780971304118E+52+x17*-196159429230834E+56+x18*401734511064748E+59+x19*-822752278660603E+62+x20*1684996666969691E+66+x21*-3450873733925E+69+x22*7067388259113454E+72+x23*-14474011154664E+76+x24*29642774844752E+79)$	* $((y0*1+y1*-2048+y2*4194304+y3*-8589934592+y4*17592186044416+y5*-36028797018964000+y6*73786976294838200000+y7*-151115727451829E+23+y8*309485009821345E+26+y9*-633825300114115E+29+y10*1298074214633714E+33+y11*-2658455991569831E+36+y12*5444517870735E+39+y13*-11150372599265E+43+y14*22835963083929E+46+y15*-46768052394588E+49+y16*95780971304118E+52+y17*-196159429230834E+56+y18*401734511064748E+59+y19*-822752278660603E+62+y20*1684996666969691E+66+y21*-3450873733925E+69+y22*7067388259113454E+72+y23*-14474011154664E+76+y24*29642774844752E+79+y25*-60708402882054E+82))$
Z = $(((x0*1+x1*4096+x2*16777216+x3*68719476736+x4*281474976710656+x5*1152921504606850000+x6*4722366482869E+21+x7*19342813113834E+25+x8*79228162514264438E+28+x9*324518553658427E+32+x10*1329227995784926E+36+x11*5444517870735E+39+x12*22300745198536E+43+x13*91343852333181E+46+x14*374144419156711E+50+x15*1532495540865894E+54+x16*6277101735386868E+57+x17*25711008708143E+61+x18*105312291668557E+65+x19*431359146674418E+68+x20*1768470647783832E+72+x21*7237005573322E+75+x22*29642774844752E+79+x23*121416805764108E+83+x24*497323236409787E+86))$	* $((y0*1+y1*4096+y2*16777216+y3*68719476736+y4*281474976710656+y5*1152921504606850000+y6*4722366482869E+21+y7*19342813113834E+25+y8*79228162514264438E+28+y9*324518553658427E+32+y10*1329227995784926E+36+y11*5444517870735E+39+y12*22300745198536E+43+y13*91343852333181E+46+y14*374144419156711E+50+y15*1532495540865894E+54+y16*6277101735386868E+57+y17*25711008708143E+61+y18*105312291668557E+65+y19*431359146674418E+68+y20*1768470647783832E+72+y21*7237005573322E+75+y22*29642774844752E+79+y23*121416805764108E+83+y24*497323236409787E+86+y25*2038035977633449E+90))$
AA= $(((x0*1+x1*-4096+x2*16777216+x3*-68719476736+x4*281474976710656+x5*-1152921504606850000+x6*4722366482869E+21+x7*-19342813113834E+25+x8*79228162514264438E+28+x9*-324518553658427E+32+x10*1329227995784926E+36+x11*-5444517870735E+39+x12*22300745198536E+43+x13*-91343852333181E+46+x14*374144419156711E+50+x15*1532495540865894E+54+x16*6277101735386868E+57+x17*25711008708143E+61+x18*105312291668557E+65+x19*431359146674418E+68+x20*1768470647783832E+72+x21*7237005573322E+75+x22*29642774844752E+79+x23*121416805764108E+83+x24*497323236409787E+86)$	* $((y0*1+y1*-4096+y2*16777216+y3*-68719476736+y4*281474976710656+y5*1152921504606850000+y6*4722366482869E+21+y7*-19342813113834E+25+y8*79228162514264438E+28+y9*-324518553658427E+32+y10*1329227995784926E+36+y11*-5444517870735E+39+y12*22300745198536E+43+y13*-91343852333181E+46+y14*374144419156711E+50+y15*1532495540865894E+54+y16*6277101735386868E+57+y17*25711008708143E+61+y18*105312291668557E+65+y19*431359146674418E+68+y20*1768470647783832E+72+y21*7237005573322E+75+y22*29642774844752E+79+y23*121416805764108E+83+y24*497323236409787E+86+y25*2038035977633449E+90))$
AB= $(((x0*1+x1*0.5+x2*0.25+x3*0.125+x4*0.0625+x5*0.03125+x6*0.015625+x7*0.0078125+x8*0.00390625+x9*0.001953125+x10*0.0009765625+x11*0.00048828125+x12*0.00024414062+x13*0.00012207031+x14*0.00006103515625+x15*0.00003051757812+x16*0.00001525878906+x17*0.00000762939453125+x18*0.00000381469726625+x19*1.9073486328125E−06+x20*9.536743164062E−07+x21*4.768371582031E−07+x22*2.384185791015E−07+x23*1.192092895507813E−07+x24*5.960464477539063E−08))$	* $((y0*1+y1*0.5+y2*0.25+y3*0.125+y4*0.0625+y5*0.03125+y6*0.015625+y7*0.0078125+y8*0.00390625+y9*0.001953125+y10*0.0009765625+y11*0.00048828125+y12*0.00024414062+y13*0.00012207031+y14*0.00006103515625+y15*0.00003051757812+y16*0.00001525878906+y17*0.00000762939453125+y18*0.00000381469726625+y19*1.9073486328125E−06+y20*9.536743164062E−07+y21*4.768371582031E−07+y22*2.384185791015E−07+y23*1.192092895507813E−07+y24*5.960464477539063E−08+y25*2.980232238769353E−08))$
AC= $(((x0*1+x1*-0.5+x2*0.25+x3*-0.125+x4*0.0625+x5*-0.03125+x6*0.015625+x7*-0.0078125+x8*0.00390625+x9*-0.001953125+x10*0.0009765625+x11*-0.00048828125+x12*0.00024414062+x13*-0.00012207031+x14*0.00006103515625+x15*-0.00003051757812+x16*0.00001525878906+x17*-0.00000762939453125+x18*0.00000381469726625+x19*-1.9073486328125E−06+x20*9.536743164062E−07+x21*-4.768371582031E−07+x22*2.384185791015E−07+x23*-1.192092895507813E−07+x24*5.960464477539063E−08))$	* $((y0*1+y1*-0.5+y2*0.25+y3*-0.125+y4*0.0625+y5*-0.03125+y6*0.015625+y7*-0.0078125+y8*0.00390625+y9*-0.001953125+y10*0.0009765625+y11*-0.00048828125+y12*0.00024414062+y13*-0.00012207031+y14*0.00006103515625+y15*-0.00003051757812+y16*0.00001525878906+y17*-0.00000762939453125+y18*0.00000381469726625+y19*-1.9073486328125E−06+y20*9.536743164062E−07+y21*-4.768371582031E−07+y22*2.384185791015E−07+y23*-1.192092895507813E−07+y24*5.960464477539063E−08+y25*-2.980232238769353E−08))$

AD= $(((x0*1+x1*0.25+x2*0.0625+x3*0.015625+x4*0.00390625+x5*0.0009765625+x6*0.000244140625+x7*0.00006103515625+x8*0.0000152587890625+x9*0.000003814697265625+x10*9.5367431640625E-07+x11*2.38418579101562E-07+x12*5.96046447753906E-08+x13*1.49011611938477E-08+x14*3.72529029846191E-09+x15*9.31322574615479E-10+x16*2.3283064365387E-10+x17*5.82076609134674E-11+x18*1.45519152283669E-11+x19*3.63797880709171E-12+x20*9.09494701772928E-13+x21*2.27373675443232E-13+x22*5.6843418860808E-14+x23*1.4210854715202E-14+x24*3.5527136788005E-15))$ * $((y0*1+y1*0.25+y2*0.0625+y3*0.015625+y4*0.00390625+y5*0.0009765625+y6*0.000244140625+y7*0.00006103515625+y8*0.0000152587890625+y9*0.000003814697265625+y10*9.5367431640625E-07+y11*2.38418579101562E-07+y12*5.96046447753906E-08+y13*1.49011611938477E-08+y14*3.72529029846191E-09+y15*9.31322574615479E-10+y16*2.3283064365387E-10+y17*5.82076609134674E-11+y18*1.45519152283669E-11+y19*3.63797880709171E-12+y20*9.09494701772928E-13+y21*2.27373675443232E-13+y22*5.6843418860808E-14+y23*1.4210854715202E-14+y24*3.5527136788005E-15+y25*8.88178419700125E-16)))$

AE= $(((x0*1+x1*-0.25+x2*0.0625+x3*-0.015625+x4*0.00390625+x5*-0.0009765625+x6*0.000244140625+x7*-0.00006103515625+x8*0.0000152587890625+x9*-0.000003814697265625+x10*9.5367431640625E-07+x11*-2.38418579101562E-07+x12*5.96046447753906E-08+x13*-1.49011611938477E-08+x14*3.72529029846191E-09+x15*-9.31322574615479E-10+x16*2.3283064365387E-10+x17*-5.82076609134674E-11+x18*1.45519152283669E-11+x19*-3.63797880709171E-12+x20*9.09494701772928E-13+x21*-2.27373675443232E-13+x22*5.6843418860808E-14+x23*-1.4210854715202E-14+x24*3.5527136788005E-15))$ * $((y0*1+y1*-0.25+y2*0.0625+y3*-0.015625+y4*0.00390625+y5*-0.0009765625+y6*0.000244140625+y7*-0.00006103515625+y8*0.0000152587890625+y9*-0.000003814697265625+y10*9.5367431640625E-07+y11*-2.38418579101562E-07+y12*5.96046447753906E-08+y13*-1.49011611938477E-08+y14*3.72529029846191E-09+y15*-9.31322574615479E-10+y16*2.3283064365387E-10+y17*-5.82076609134674E-11+y18*1.45519152283669E-11+y19*-3.63797880709171E-12+y20*9.09494701772928E-13+y21*-2.27373675443232E-13+y22*5.6843418860808E-14+y23*-1.4210854715202E-14+y24*3.5527136788005E-15+y25*-8.88178419700125E-16)))$

AF= $(((x0*1+x1*0.125+x2*0.015625+x3*0.001953125+x4*0.000244140625+x5*0.000030517578125+x6*0.000003814697265625+x7*4.76837158203125E-07+x8*5.96046447753906E-08+x9*7.45058059692383E-09+x10*9.31322574615479E-10+x11*1.16415321826935E-10+x12*1.45519152283669E-11+x13*1.81898940354586E-12+x14*2.27373675443232E-13+x15*2.8421709430404E-14+x16*3.5527136788005E-15+x17*4.44089209850063E-16+x18*5.55111512312578E-17+x19*6.93889390390723E-18+x20*8.67361737988404E-19+x21*1.0842021724855E-19+x22*1.35525271560688E-20+x23*1.6940658945086E-21+x24*2.11758236813575E-22))$ * $((y0*1+y1*0.125+y2*0.015625+y3*0.001953125+y4*0.000244140625+y5*0.000030517578125+y6*0.000003814697265625+y7*4.76837158203125E-07+y8*5.96046447753906E-08+y9*7.45058059692383E-09+y10*9.31322574615479E-10+y11*1.16415321826935E-10+y12*1.45519152283669E-11+y13*1.81898940354586E-12+y14*2.27373675443232E-13+y15*2.8421709430404E-14+y16*3.5527136788005E-15+y17*4.44089209850063E-16+y18*5.55111512312578E-17+y19*6.93889390390723E-18+y20*8.67361737988404E-19+y21*1.0842021724855E-19+y22*1.35525271560688E-20+y23*1.6940658945086E-21+y24*2.11758236813575E-22+y25*2.64697796016969E-23)))$

AG= $(((x0*1+x1*-0.125+x2*0.015625+x3*-0.001953125+x4*0.000244140625+x5*-0.000030517578125+x6*0.000003814697265625+x7*-4.76837158203125E-07+x8*5.96046447753906E-08+x9*-7.45058059692383E-09+x10*9.31322574615479E-10+x11*-1.16415321826935E-10+x12*1.45519152283669E-11+x13*-1.81898940354586E-12+x14*2.27373675443232E-13+x15*-2.8421709430404E-14+x16*3.5527136788005E-15+x17*-4.44089209850063E-16+x18*5.55111512312578E-17+x19*-6.93889390390723E-18+x20*8.67361737988404E-19+x21*-1.0842021724855E-19+x22*1.35525271560688E-20+x23*-1.6940658945086E-21+x24*2.11758236813575E-22))$ * $((y0*1+y1*-0.125+y2*0.015625+y3*-0.001953125+y4*0.000244140625+y5*-0.000030517578125+y6*0.000003814697265625+y7*-4.76837158203125E-07+y8*5.96046447753906E-08+y9*-7.45058059692383E-09+y10*9.31322574615479E-10+y11*-1.16415321826935E-10+y12*1.45519152283669E-11+y13*-1.81898940354586E-12+y14*2.27373675443232E-13+y15*-2.8421709430404E-14+y16*3.5527136788005E-15+y17*-4.44089209850063E-16+y18*5.55111512312578E-17+y19*-6.93889390390723E-18+y20*8.67361737988404E-19+y21*-1.0842021724855E-19+y22*1.35525271560688E-20+y23*-1.6940658945086E-21+y24*2.11758236813575E-22+y25*-2.64697796016969E-23)))$

AH= $(((x0*1+x1*0.0625+x2*0.00390625+x3*0.000244140625+x4*0.0000152587890625+x5*9.5367431640625E-07+x6*5.96046447753906E-08+x7*3.72529029846191E-09+x8*2.3283064365387E-10+x9*1.45519152283669E-11+x10*9.09494701772928E-13+x11*5.6843418860808E-14+x12*3.5527136788005E-15+x13*2.22044604925031E-16+x14*1.38777878078145E-17+x15*8.67361737988404E-19+x16*5.42101086242752E-20+x17*3.3881317890172E-21+x18*2.11758236813575E-22+x19*1.32348898008484E-23+x20*8.27180612553028E-25+x21*5.16987882845642E-26+x22*3.23117426778526E-27+x23*2.01948391736579E-28+x24*1.26217744835362E-29))$ * $((y0*1+y1*0.0625+y2*0.00390625+y3*0.000244140625+y4*0.0000152587890625+y5*9.5367431640625E-07+y6*5.96046447753906E-08+y7*3.72529029846191E-09+y8*2.3283064365387E-10+y9*1.45519152283669E-11+y10*9.09494701772928E-13+y11*5.6843418860808E-14+y12*3.5527136788005E-15+y13*2.22044604925031E-16+y14*1.38777878078145E-17+y15*8.67361737988404E-19+y16*5.42101086242752E-20+y17*3.3881317890172E-21+y18*2.11758236813575E-22+y19*1.32348898008484E-23+y20*8.27180612553028E-25+y21*5.16987882845642E-26+y22*3.23117426778526E-27+y23*2.01948391736579E-28+y24*1.26217744835362E-29+y25*7.88860905221012E-31)))$

AI = $(((x0*1+x1*-0.0625+x2*0.00390625+x3*-0.000244140625+x4*0.0000152587890625+x5*-9.5367431640625E-07+x6*5.96046447753906E-08+x7*-3.72529029846191E-09+x8*2.3283064365387E-10+x9*-1.45519152283669E-11+x10*9.09494701772928E-13+x11*-5.6843418860808E-14+x12*3.5527136788005E-15+x13*-2.22044604925031E-16+x14*1.38777878078145E-17+x15*-8.67361737988404E-19+x16*5.42101086242752E-20+x17*-3.3881317890172E-21+x18*2.11758236813575E-22+x19*-1.32348898008484E-23+x20*8.27180612553028E-25+x21*-5.16987882845642E-26+x22*3.23117426778526E-27+x23*-2.01948391736579E-28+x24*1.26217744835362E-29))$ * $((y0*1+y1*-0.0625+y2*0.00390625+y3*-0.000244140625+y4*0.0000152587890625+y5*-9.5367431640625E-07+y6*5.96046447753906E-08+y7*-3.72529029846191E-09+y8*2.3283064365387E-10+y9*-1.45519152283669E-11+y10*9.09494701772928E-13+y11*-5.6843418860808E-14+y12*3.5527136788005E-15+y13*-2.22044604925031E-16+y14*1.38777878078145E-17+y15*-8.67361737988404E-19+y16*5.42101086242752E-20+y17*-3.3881317890172E-21+y18*2.11758236813575E-22+y19*-1.32348898008484E-23+y20*8.27180612553028E-25+y21*-5.16987882845642E-26+y22*3.23117426778526E-27+y23*-2.01948391736579E-28+y24*1.26217744835362E-29+y25*-7.88860905221012E-31))$

AJ = $(((x0*1+x1*0.03125+x2*0.0009765625+x3*0.000030517578125+x4*9.5367431640625E-07+x5*2.98023223876953E-08+x6*9.31322574615479E-10+x7*2.91038304567337E-11+x8*9.09494701772928E-13+x9*2.8421709430404E-14+x10*8.88178419700125E-16+x11*2.77555756156289E-17+x12*8.67361737988404E-19+x13*2.71050543121376E-20+x14*8.470329472543E-22+x15*2.64697796016969E-23+x16*8.27180612553028E-25+x17*2.58493941422821E-26+x18*8.07793566946316E-28+x19*2.52435489670724E-29+x20*7.88860905221012E-31+x21*2.46519032881566E-32+x22*7.70371977754894E-34+x23*2.40741243048404E-35+x24*7.52316384526264E-37))$ * $((y0*1+y1*0.03125+y2*0.0009765625+y3*0.000030517578125+y4*9.5367431640625E-07+y5*2.98023223876953E-08+y6*9.31322574615479E-10+y7*2.91038304567337E-11+y8*9.09494701772928E-13+y9*2.8421709430404E-14+y10*8.88178419700125E-16+y11*2.77555756156289E-17+y12*8.67361737988404E-19+y13*2.71050543121376E-20+y14*8.470329472543E-22+y15*2.64697796016969E-23+y16*8.27180612553028E-25+y17*2.58493941422821E-26+y18*8.07793566946316E-28+y19*2.52435489670724E-29+y20*7.88860905221012E-31+y21*2.46519032881566E-32+y22*7.70371977754894E-34+y23*2.40741243048404E-35+y24*7.52316384526264E-37+y25*2.35098870164458E-38)))$

AK= $(((x0*1+x1*-0.03125+x2*0.0009765625+x3*-0.000030517578125+x4*9.5367431640625E-07+x5*-2.98023223876953E-08+x6*9.31322574615479E-10+x7*-2.91038304567337E-11+x8*9.09494701772928E-13+x9*-2.8421709430404E-14+x10*8.88178419700125E-16+x11*-2.77555756156289E-17+x12*8.67361737988404E-19+x13*-2.71050543121376E-20+x14*8.470329472543E-22+x15*-2.64697796016969E-23+x16*8.27180612553028E-25+x17*-2.58493941422821E-26+x18*8.07793566946316E-28+x19*-2.52435489670724E-29+x20*7.88860905221012E-31+x21*-2.46519032881566E-32+x22*7.70371977754894E-34+x23*-2.40741243048404E-35+x24*7.52316384526264E-37))$ * $((y0*1+y1*-0.03125+y2*0.0009765625+y3*-0.000030517578125+y4*9.5367431640625E-07+y5*-2.98023223876953E-08+y6*9.31322574615479E-10+y7*-2.91038304567337E-11+y8*9.09494701772928E-13+y9*-2.8421709430404E-14+y10*8.88178419700125E-16+y11*-2.77555756156289E-17+y12*8.67361737988404E-19+y13*-2.71050543121376E-20+y14*8.470329472543E-22+y15*-2.64697796016969E-23+y16*8.27180612553028E-25+y17*-2.58493941422821E-26+y18*8.07793566946316E-28+y19*-2.52435489670724E-29+y20*7.88860905221012E-31+y21*-2.46519032881566E-32+y22*7.70371977754894E-34+y23*-2.40741243048404E-35+y24*7.52316384526264E-37+y25*-2.35098870164458E-38)))$

AL= $(((x0*1+x1*0.015625+x2*0.000244140625+x3*0.000003814697265625+x4*5.96046447753906E-08+x5*9.31322574615479E-10+x6*1.45519152283669E-11+x7*2.27373675443232E-13+x8*3.5527136788005E-15+x9*5.55111512312578E-17+x10*8.67361737988404E-19+x11*1.35525271560688E-20+x12*2.11758236813575E-22+x13*3.30872245021211E-24+x14*5.16987882845642E-26+x15*8.07793566946316E-28+x16*1.26217744835362E-29+x17*1.97215226305253E-31+x18*3.08148791101958E-33+x19*4.81482486096809E-35+x20*7.52316384526264E-37+x21*1.17549435082229E-38+x22*1.83670992315982E-40+x23*2.86985925493723E-42+x24*4.48415508583941E-44))$ * $((y0*1+y1*0.015625+y2*0.000244140625+y3*0.000003814697265625+y4*5.96046447753906E-08+y5*9.31322574615479E-10+y6*1.45519152283669E-11+y7*2.27373675443232E-13+y8*3.5527136788005E-15+y9*5.55111512312578E-17+y10*8.67361737988404E-19+y11*1.35525271560688E-20+y12*2.11758236813575E-22+y13*3.30872245021211E-24+y14*5.16987882845642E-26+y15*8.07793566946316E-28+y16*1.26217744835362E-29+y17*1.97215226305253E-31+y18*3.08148791101958E-33+y19*4.81482486096809E-35+y20*7.52316384526264E-37+y21*1.17549435082229E-38+y22*1.83670992315982E-40+y23*2.86985925493723E-42+y24*4.48415508583941E-44+y25*7.00649232162409E-46)))$

AM= $(((x0*1+x1*-0.015625+x2*0.000244140625+x3*-0.000003814697265625+x4*5.96046447753906E-08+x5*-9.31322574615479E-10+x6*1.45519152283669E-11+x7*-2.27373675443232E-13+x8*3.5527136788005E-15+x9*-5.55111512312578E-17+x10*8.67361737988404E-19+x11*-1.35525271560688E-20+x12*2.11758236813575E-22+x13*-3.30872245021211E-24+x14*5.16987882845642E-26+x15*-8.07793566946316E-28+x16*1.26217744835362E-29+x17*-1.97215226305253E-31+x18*3.08148791101958E-33+x19*-4.81482486096809E-35+x20*7.52316384526264E-37+x21*-1.17549435082229E-38+x22*1.83670992315982E-40+x23*-2.86985925493723E-42+x24*4.48415508583941E-44))$ * $((y0*1+y1*-0.015625+y2*0.000244140625+y3*-0.000003814697265625+y4*5.96046447753906E-08+y5*-9.31322574615479E-10+y6*1.45519152283669E-11+y7*-2.27373675443232E-13+y8*3.5527136788005E-15+y9*-5.55111512312578E-17+y10*8.67361737988404E-19+y11*-1.35525271560688E-20+y12*2.11758236813575E-22+y13*-3.30872245021211E-24+y14*5.16987882845642E-26+y15*-8.07793566946316E-28+y16*1.26217744835362E-29+y17*-1.97215226305253E-31+y18*3.08148791101958E-33+y19*-4.81482486096809E-35+y20*7.52316384526264E-37+y21*-1.17549435082229E-38+y22*1.83670992315982E-40+y23*-2.86985925493723E-42+y24*4.48415508583941E-44+y25*-7.00649232162409E-46)))$

AN= $(((x0*1+x1*0.0078125+x2*0.00006103515625+x3*4.76837158203125E-07+x4*3.72529029846191E-09+x5*2.91038304567337E-11+x6*2.27373675443232E-13+x7*1.77635683940025E-15+x8*1.38777878078145E-17+x9*1.0842021724855E-19+x10*8.470329472543E-22+x11*6.617444900424222E-24+x12*5.16987882845642E-26+x13*4.03896783473158E-28+x14*3.15544362088405E-30+x15*2.46519032881566E-32+x16*1.92592994438724E-34+x17*1.50463276905253E-36+x18*1.17549435082229E-38+x19*9.18354961579912E-41+x20*7.17464813734306E-43+x21*5.60519385729927E-45+x22*4.37905770101505E-47+x23*3.42113882891801E-49+x24*2.6727647100922E-51))$ * $((y0*1+y1*0.0078125+y2*0.00006103515625+y3*4.76837158203125E-07+y4*3.72529029846191E-09+y5*2.91038304567337E-11+y6*2.27373675443232E-13+y7*1.77635683940025E-15+y8*1.38777878078145E-17+y9*1.0842021724855E-19+y10*8.470329472543E-22+y11*6.617444900424222E-24+y12*5.16987882845642E-26+y13*4.03896783473158E-28+y14*3.15544362088405E-30+y15*2.46519032881566E-32+y16*1.92592994438724E-34+y17*1.50463276905253E-36+y18*1.17549435082229E-38+y19*9.18354961579912E-41+y20*7.17464813734306E-43+y21*5.60519385729927E-45+y22*4.37905770101505E-47+y23*3.42113882891801E-49+y24*2.6727647100922E-51+y25*2.08809742975953E-53)))$

AO= $(((x0*1+x1*-0.0078125+x2*0.00006103515625+x3*-4.76837158203125E-07+x4*3.72529029846191E-09+x5*-2.91038304567337E-11+x6*2.27373675443232E-13+x7*-1.77635683940025E-15+x8*1.38777878078145E-17+x9*-1.0842021724855E-19+x10*8.470329472543E-22+x11*-6.617444900424222E-24+x12*5.16987882845642E-26+x13*-4.03896783473158E-28+x14*3.15544362088405E-30+x15*-2.46519032881566E-32+x16*1.92592994438724E-34+x17*-1.50463276905253E-36+x18*1.17549435082229E-38+x19*-9.18354961579912E-41+x20*7.17464813734306E-43+x21*-5.60519385729927E-45+x22*4.37905770101505E-47+x23*-3.42113882891801E-49+x24*2.6727647100922E-51))$ * $((y0*1+y1*-0.0078125+y2*0.00006103515625+y3*-4.76837158203125E-07+y4*3.72529029846191E-09+y5*-2.91038304567337E-11+y6*2.27373675443232E-13+y7*-1.77635683940025E-15+y8*1.38777878078145E-17+y9*-1.0842021724855E-19+y10*8.470329472543E-22+y11*-6.617444900424222E-24+y12*5.16987882845642E-26+y13*-4.03896783473158E-28+y14*3.15544362088405E-30+y15*-2.46519032881566E-32+y16*1.92592994438724E-34+y17*-1.50463276905253E-36+y18*1.17549435082229E-38+y19*-9.18354961579912E-41+y20*7.17464813734306E-43+y21*-5.60519385729927E-45+y22*4.37905770101505E-47+y23*-3.42113882891801E-49+y24*2.6727647100922E-51+y25*-2.08809742975953E-53)))$

AP= $(((x0*1+x1*0.00390625+x2*0.0000152587890625+x3*5.96046447753906E-08+x4*2.3283064365387E-10+x5*9.09494701772928E-13+x6*3.5527136788005E-15+x7*1.38777878078145E-17+x8*5.42101086242752E-20+x9*2.11758236813575E-22+x10*8.27180612553028E-25+x11*3.23117426778526E-27+x12*1.26217744835362E-29+x13*4.93038065763132E-32+x14*1.92592994438724E-34+x15*7.52316384526264E-37+x16*2.93873587705572E-39+x17*1.14794370197489E-41+x18*4.48415508583941E-44+x19*1.75162308040602E-46+x20*6.84227765783602E-49+x21*2.6727647100922E-51+x22*1.04404871487976E-53+x23*4.07831529249908E-56+x24*1.59309191113245E-58))$ * $((y0*1+y1*0.00390625+y2*0.0000152587890625+y3*5.96046447753906E-08+y4*2.3283064365387E-10+y5*9.09494701772928E-13+y6*3.5527136788005E-15+y7*1.38777878078145E-17+y8*5.42101086242752E-20+y9*2.11758236813575E-22+y10*8.27180612553028E-25+y11*3.23117426778526E-27+y12*1.26217744835362E-29+y13*4.93038065763132E-32+y14*1.92592994438724E-34+y15*7.52316384526264E-37+y16*2.93873587705572E-39+y17*1.14794370197489E-41+y18*4.48415508583941E-44+y19*1.75162308040602E-46+y20*6.84227765783602E-49+y21*2.6727647100922E-51+y22*1.04404871487976E-53+y23*4.07831529249908E-56+y24*1.59309191113245E-58+y25*6.22301527786114E-61)))$

AQ= $(((x0*1+x1*-0.00390625+x2*0.0000152587890625+x3*-5.96046447753906E-08+x4*2.3283064365387E-10+x5*-9.09494701772928E-13+x6*3.5527136788005E-15+x7*-1.38777878078145E-17+x8*5.42101086242752E-20+x9*-2.11758236813575E-22+x10*8.27180612553028E-25+x11*-3.23117426778526E-27+x12*1.26217744835362E-29+x13*-4.93038065763132E-32+x14*1.92592994438724E-34+x15*-7.52316384526264E-37+x16*2.93873587705572E-39+x17*-1.14794370197489E-41+x18*4.48415508583941E-44+x19*-1.75162308040602E-46+x20*6.84227765783602E-49+x21*-2.6727647100922E-51+x22*1.04404871487976E-53+x23*-4.07831529249908E-56+x24*1.59309191113245E-58))$ * $((y0*1+y1*-0.00390625+y2*0.0000152587890625+y3*-5.96046447753906E-08+y4*2.3283064365387E-10+y5*-9.09494701772928E-13+y6*3.5527136788005E-15+y7*-1.38777878078145E-17+y8*5.42101086242752E-20+y9*-2.11758236813575E-22+y10*8.27180612553028E-25+y11*-3.23117426778526E-27+y12*1.26217744835362E-29+y13*-4.93038065763132E-32+y14*1.92592994438724E-34+y15*-7.52316384526264E-37+y16*2.93873587705572E-39+y17*-1.14794370197489E-41+y18*4.48415508583941E-44+y19*-1.75162308040602E-46+y20*6.84227765783602E-49+y21*-2.6727647100922E-51+y22*1.04404871487976E-53+y23*-4.07831529249908E-56+y24*1.59309191113245E-58+y25*-6.22301527786114E-61)))$

AR= $(((x0*1+x1*0.001953125+x2*0.000003814697265625+x3*7.45058059692383E-09+x4*1.45519152283669E-11+x5*2.8421709430404E-14+x6*5.55111512312578E-17+x7*1.0842021724855E-19+x8*2.11758236813575E-22+x9*4.13590306276514E-25+x10*8.07793566946316E-28+x11*1.57772181044202E-30+x12*3.08148791101958E-33+x13*6.01853107621011E-36+x14*1.17549435082229E-38+x15*2.29588740394978E-41+x16*4.48415508583941E-44+x17*8.75811540203011E-47+x18*1.71056941445901E-49+x19*3.34095588761524E-52+x20*6.52530446799852E-55+x21*1.27447352890596E-57+x22*2.48920611114446E-60+x23*4.86173068582902E-63+x24*9.4955677457598E-66))$ * $((y0*1+y1*0.001953125+y2*0.000003814697265625+y3*7.45058059692383E-09+y4*1.45519152283669E-11+y5*2.8421709430404E-14+y6*5.55111512312578E-17+y7*1.0842021724855E-19+y8*2.11758236813575E-22+y9*4.13590306276514E-25+y10*8.07793566946316E-28+y11*1.57772181044202E-30+y12*3.08148791101958E-33+y13*6.01853107621011E-36+y14*1.17549435082229E-38+y15*2.29588740394978E-41+y16*4.48415508583941E-44+y17*8.75811540203011E-47+y18*1.71056941445901E-49+y19*3.34095588761524E-52+y20*6.52530446799852E-55+y21*1.27447352890596E-57+y22*2.48920611114446E-60+y23*4.86173068582902E-63+y24*9.4955677457598E-66+y25*1.85460307534371E-68)))$

AS = $(((x0*1+x1*-0.001953125+x2*0.000003814697265625+x3*-7.45058059692383E-09+x4*1.45519152283669E-11+x5*-2.8421709430404E-14+x6*5.55111512312578E-17+x7*-1.0842021724855E-19+x8*2.11758236813575E-22+x9*-4.13590306276514E-25+x10*8.07793566946316E-28+x11*-1.57772181044202E-30+x12*3.08148791101958E-33+x13*-6.01853107621011E-36+x14*1.17549435082229E-38+x15*-2.29588740394978E-41+x16*4.48415508583941E-44+x17*-8.75811540203011E-47+x18*1.71056941445901E-49+x19*-3.34095588761524E-52+x20*6.52530446799852E-55+x21*-1.27447352890596E-57+x22*2.48920611114446E-60+x23*-4.86173068582902E-63+x24*9.4955677457598E-66))$ * $((y0*1+y1*-0.001953125+y2*0.000003814697265625+y3*-7.45058059692383E-09+y4*1.45519152283669E-11+y5*-2.8421709430404E-14+y6*5.55111512312578E-17+y7*-1.0842021724855E-19+y8*2.11758236813575E-22+y9*-4.13590306276514E-25+y10*8.07793566946316E-28+y11*-1.57772181044202E-30+y12*3.08148791101958E-33+y13*-6.01853107621011E-36+y14*1.17549435082229E-38+y15*-2.29588740394978E-41+y16*4.48415508583941E-44+y17*-8.75811540203011E-47+y18*1.71056941445901E-49+y19*-3.34095588761524E-52+y20*6.52530446799852E-55+y21*-1.27447352890596E-57+y22*2.48920611114446E-60+y23*-4.86173068582902E-63+y24*9.4955677457598E-66+y25*-1.85460307534371E-68)))$

AT= $(((x0*1+x1*0.0009765625+x2*9.5367431640625E-07+x3*9.31322574615479E-10+x4*9.09494701772928E-13+x5*8.88178419700125E-16+x6*8.67361737988404E-19+x7*8.470329472543E-22+x8*8.27180612553028E-25+x9*8.07793566946316E-28+x10*7.88860905221012E-31+x11*7.70371977754894E-34+x12*7.52316384526264E-37+x13*7.3468396926393E-40+x14*7.17464813734306E-43+x15*7.00649232162409E-46+x16*6.84227765783602E-49+x17*6.68191177523049E-52+x18*6.52530446799852E-55+x19*6.37236764452981E-58+x20*6.22301527786114E-61+x21*6.07716335728627E-64+x22*5.93472984109987E-67+x23*5.7956346104491E-70+x24*5.6597994242667E-73))$ * $((y0*1+y1*0.0009765625+y2*9.5367431640625E-07+y3*9.31322574615479E-10+y4*9.09494701772928E-13+y5*8.88178419700125E-16+y6*8.67361737988404E-19+y7*8.470329472543E-22+y8*8.27180612553028E-25+y9*8.07793566946316E-28+y10*7.88860905221012E-31+y11*7.70371977754894E-34+y12*7.52316384526264E-37+y13*7.3468396926393E-40+y14*7.17464813734306E-43+y15*7.00649232162409E-46+y16*6.84227765783602E-49+y17*6.68191177523049E-52+y18*6.52530446799852E-55+y19*6.37236764452981E-58+y20*6.22301527786114E-61+y21*6.07716335728627E-64+y22*5.93472984109987E-67+y23*5.7956346104491E-70+y24*5.6597994242667E-73+y25*5.52714787526044E-76)))$

AU= $(((x0*1+x1*-0.0009765625+x2*9.5367431640625E-07+x3*-9.31322574615479E-10+x4*9.09494701772928E-13+x5*-8.88178419700125E-16+x6*8.67361737988404E-19+x7*-8.470329472543E-22+x8*8.27180612553028E-25+x9*-8.07793566946316E-28+x10*7.88860905221012E-31+x11*-7.70371977754894E-34+x12*7.52316384526264E-37+x13*-7.3468396926393E-40+x14*7.17464813734306E-43+x15*-7.00649232162409E-46+x16*6.84227765783602E-49+x17*-6.68191177523049E-52+x18*6.52530446799852E-55+x19*-6.37236764452981E-58+x20*6.22301527786114E-61+x21*-6.07716335728627E-64+x22*5.93472984109987E-67+x23*-5.7956346104491E-70+x24*5.6597994242667E-73))$ * $((y0*1+y1*-0.0009765625+y2*9.5367431640625E-07+y3*-9.31322574615479E-10+y4*9.09494701772928E-13+y5*-8.88178419700125E-16+y6*8.67361737988404E-19+y7*-8.470329472543E-22+y8*8.27180612553028E-25+y9*-8.07793566946316E-28+y10*7.88860905221012E-31+y11*-7.70371977754894E-34+y12*7.52316384526264E-37+y13*-7.3468396926393E-40+y14*7.17464813734306E-43+y15*-7.00649232162409E-46+y16*6.84227765783602E-49+y17*-6.68191177523049E-52+y18*6.52530446799852E-55+y19*-6.37236764452981E-58+y20*6.22301527786114E-61+y21*-6.07716335728627E-64+y22*5.93472984109987E-67+y23*-5.7956346104491E-70+y24*5.6597994242667E-73+y25*-5.52714787526044E-76)))$

AV= $(((x0*1+x1*0.00048828125+x2*2.38418579101562E-07+x3*1.16415321826935E-10+x4*5.6843418860808E-14+x5*2.77555756156289E-17+x6*1.35525271560688E-20+x7*6.617444900424222E-24+x8*3.23117426778526E-27+x9*1.57772181044202E-30+x10*7.70371977754894E-34+x11*3.76158192263132E-37+x12*1.83670992315982E-40+x13*8.96831017167883E-44+x14*4.37905770101505E-47+x15*2.13821176807376E-50+x16*1.04404871487976E-53+x17*5.09789411562385E-57+x18*2.48920611114446E-60+x19*1.21543267145725E-63+x20*5.93472984109987E-67+x21*2.89781730522455E-70+x22*1.41494985606667E-73+x23*6.90893484407556E-77+x24*3.37350334183377E-80))$ * $((y0*1+y1*0.00048828125+y2*2.38418579101562E-07+y3*1.16415321826935E-10+y4*5.6843418860808E-14+y5*2.77555756156289E-17+y6*1.35525271560688E-20+y7*6.617444900424222E-24+y8*3.23117426778526E-27+y9*1.57772181044202E-30+y10*7.70371977754894E-34+y11*3.76158192263132E-37+y12*1.83670992315982E-40+y13*8.96831017167883E-44+y14*4.37905770101505E-47+y15*2.13821176807376E-50+y16*1.04404871487976E-53+y17*5.09789411562385E-57+y18*2.48920611114446E-60+y19*1.21543267145725E-63+y20*5.93472984109987E-67+y21*2.89781730522455E-70+y22*1.41494985606667E-73+y23*6.90893484407556E-77+y24*3.37350334183377E-80+y25*1.64721842862977E-83)))$

AW= $(((x0*1+x1*-0.00048828125+x2*2.38418579101562E-07+x3*-1.16415321826935E-10+x4*5.6843418860808E-14+x5*-2.77555756156289E-17+x6*1.35525271560688E-20+x7*-6.617444900424222E-24+x8*3.23117426778526E-27+x9*-1.57772181044202E-30+x10*7.70371977754894E-34+x11*-3.76158192263132E-37+x12*1.83670992315982E-40+x13*-8.96831017167883E-44+x14*4.37905770101505E-47+x15*-2.13821176807376E-50+x16*1.04404871487976E-53+x17*-5.09789411562385E-57+x18*2.48920611114446E-60+x19*-1.21543267145725E-63+x20*5.93472984109987E-67+x21*-2.89781730522455E-70+x22*1.41494985606667E-73+x23*-6.90893484407556E-77+x24*3.37350334183377E-80))$ * $((y0*1+y1*-0.00048828125+y2*2.38418579101562E-07+y3*-1.16415321826935E-10+y4*5.6843418860808E-14+y5*-2.77555756156289E-17+y6*1.35525271560688E-20+y7*-6.617444900424222E-24+y8*3.23117426778526E-27+y9*-1.57772181044202E-30+y10*7.70371977754894E-34+y11*-3.76158192263132E-37+y12*1.83670992315982E-40+y13*-8.96831017167883E-44+y14*4.37905770101505E-47+y15*-2.13821176807376E-50+y16*1.04404871487976E-53+y17*-5.09789411562385E-57+y18*2.48920611114446E-60+y19*-1.21543267145725E-63+y20*5.93472984109987E-67+y21*-2.89781730522455E-70+y22*1.41494985606667E-73+y23*-6.90893484407556E-77+y24*3.37350334183377E-80+y25*-1.64721842862977E-83)))$

AX= $(((x0*1+x1*0.000244140625+x2*5.96046447753906E-08+x3*1.45519152283669E-11+x4*3.5527136788005E-15+x5*8.67361737988404E-19+x6*2.11758236813575E-22+x7*5.16987882845642E-26+x8*1.26217744835362E-29+x9*3.08148791101958E-33+x10*7.52316384526264E-37+x11*1.83670992315982E-40+x12*4.48415508583941E-44+x13*1.09476442525376E-47+x14*2.6727647100922E-51+x15*6.52530446799852E-55+x16*1.59309191113245E-58+x17*3.88938454866321E-62+x18*9.4955677457598E-66+x19*2.31825384417964E-69+x20*5.6597994242667E-73+x21*1.38178696881511E-76+x22*3.37350334183377E-80+x23*8.23609214314885E-84+x24*2.01076468338595E-87))$ * $((y0*1+y1*0.000244140625+y2*5.96046447753906E-08+y3*1.45519152283669E-11+y4*3.5527136788005E-15+y5*8.67361737988404E-19+y6*2.11758236813575E-22+y7*5.16987882845642E-26+y8*1.26217744835362E-29+y9*3.08148791101958E-33+y10*7.52316384526264E-37+y11*1.83670992315982E-40+y12*4.48415508583941E-44+y13*1.09476442525376E-47+y14*2.6727647100922E-51+y15*6.52530446799852E-55+y16*1.59309191113245E-58+y17*3.88938454866321E-62+y18*9.4955677457598E-66+y19*2.31825384417964E-69+y20*5.6597994242667E-73+y21*1.38178696881511E-76+y22*3.37350334183377E-80+y23*8.23609214314885E-84+y24*2.01076468338595E-87+y25*4.90909346529773E-91)))$

AY= $(((x0*1+x1*-0.000244140625+x2*5.96046447753906E-08+x3*-1.45519152283669E-11+x4*3.5527136788005E-15+x5*-8.67361737988404E-19+x6*2.11758236813575E-22+x7*-5.16987882845642E-26+x8*1.26217744835362E-29+x9*-3.08148791101958E-33+x10*7.52316384526264E-37+x11*-1.83670992315982E-40+x12*4.48415508583941E-44+x13*-1.09476442525376E-47+x14*2.6727647100922E-51+x15*-6.52530446799852E-55+x16*1.59309191113245E-58+x17*-3.88938454866321E-62+x18*9.4955677457598E-66+x19*-2.31825384417964E-69+x20*5.6597994242667E-73+x21*-1.38178696881511E-76+x22*3.37350334183377E-80+x23*-8.23609214314885E-84+x24*2.01076468338595E-87))$ * $((y0*1+y1*-0.000244140625+y2*5.96046447753906E-08+y3*-1.45519152283669E-11+y4*3.5527136788005E-15+y5*-8.67361737988404E-19+y6*2.11758236813575E-22+y7*-5.16987882845642E-26+y8*1.26217744835362E-29+y9*-3.08148791101958E-33+y10*7.52316384526264E-37+y11*-1.83670992315982E-40+y12*4.48415508583941E-44+y13*-1.09476442525376E-47+y14*2.6727647100922E-51+y15*-6.52530446799852E-55+y16*1.59309191113245E-58+y17*-3.88938454866321E-62+y18*9.4955677457598E-66+y19*-2.31825384417964E-69+y20*5.6597994242667E-73+y21*-1.38178696881511E-76+y22*3.37350334183377E-80+y23*-8.23609214314885E-84+y24*2.01076468338595E-87+y25*-4.90909346529773E-91)))$

AZ= $(((x0*1+x1*0.0001220703125+x2*1.49011611938477E-08+x3*1.81898940354586E-12+x4*2.22044604925031E-16+x5*2.71050543121376E-20+x6*3.30872245021211E-24+x7*4.03896783473158E-28+x8*4.93038065763132E-32+x9*6.01853107621011E-36+x10*7.3468396926393E-40+x11*8.96831017167883E-44+x12*1.09476442525376E-47+x13*1.3363823550461E-51+x14*1.63132611699963E-55+x15*1.99136488891557E-59+x16*2.43086534291451E-63+x17*2.96736492054994E-67+x18*3.62227163153068E-71+x19*4.42171830020836E-75+x20*5.39760534693403E-79+x21*6.58887371451908E-83+x22*8.04305873354438E-87+x23*9.81818693059545E-91+x24*1.1985091468012E-94))$ * $((y0*1+y1*0.0001220703125+y2*1.49011611938477E-08+y3*1.81898940354586E-12+y4*2.22044604925031E-16+y5*2.71050543121376E-20+y6*3.30872245021211E-24+y7*4.03896783473158E-28+y8*4.93038065763132E-32+y9*6.01853107621011E-36+y10*7.3468396926393E-40+y11*8.96831017167883E-44+y12*1.09476442525376E-47+y13*1.3363823550461E-51+y14*1.63132611699963E-55+y15*1.99136488891557E-59+y16*2.43086534291451E-63+y17*2.96736492054994E-67+y18*3.62227163153068E-71+y19*4.42171830020836E-75+y20*5.39760534693403E-79+y21*6.58887371451908E-83+y22*8.04305873354438E-87+y23*9.81818693059545E-91+y24*1.1985091468012E-94+y25*1.46302386084131E-98)))$

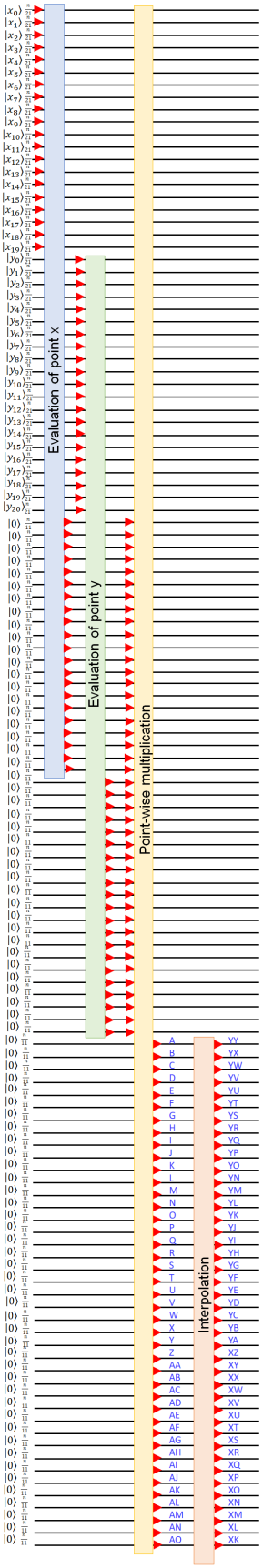

Fig. 5: General quantum design for the Toom–Cook 25.5-way multiplication algorithm. The figure illustrates the register allocation at each major computational step of the Toom–Cook 25.5-way method.

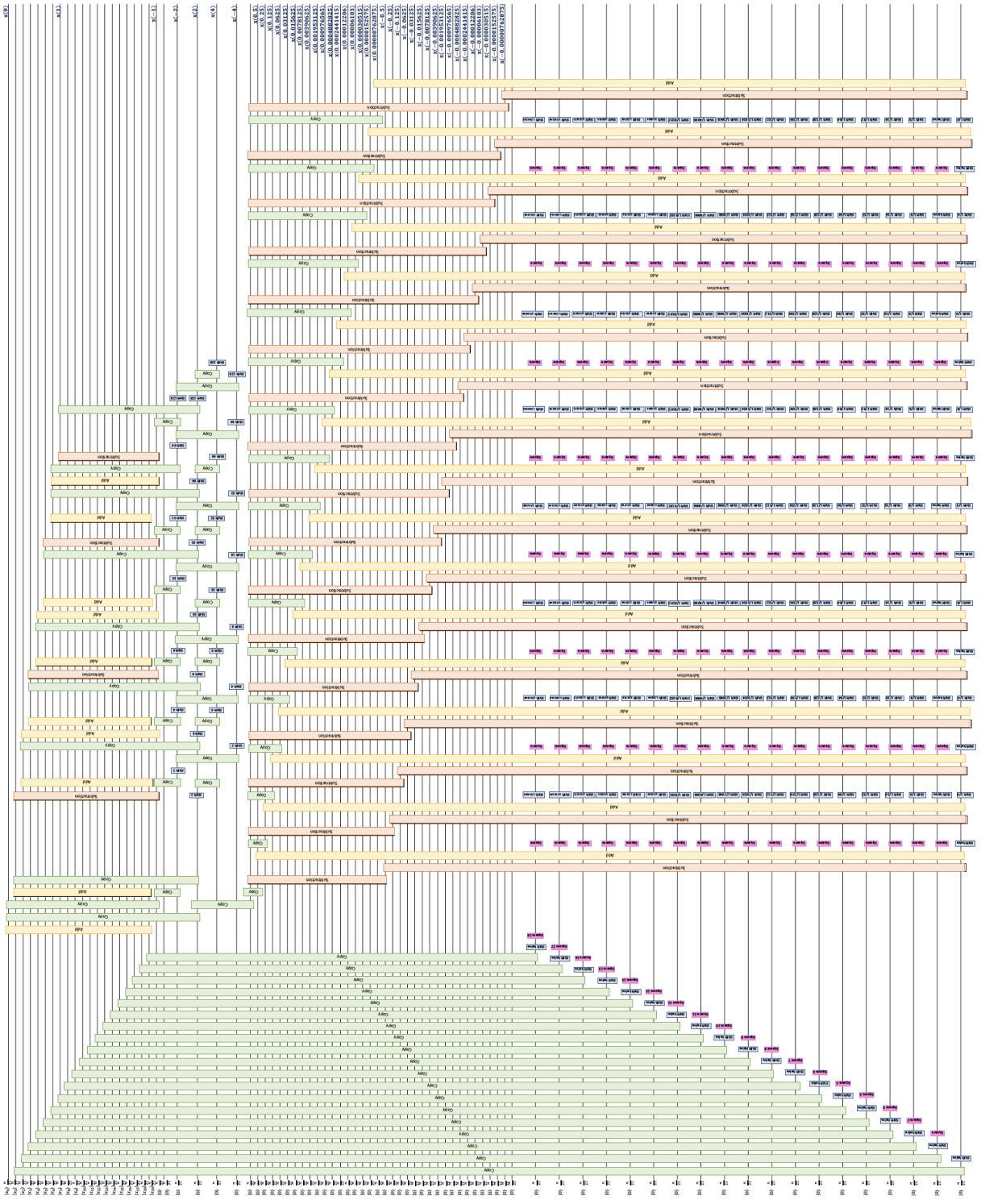

Fig. 6: Evaluation point x

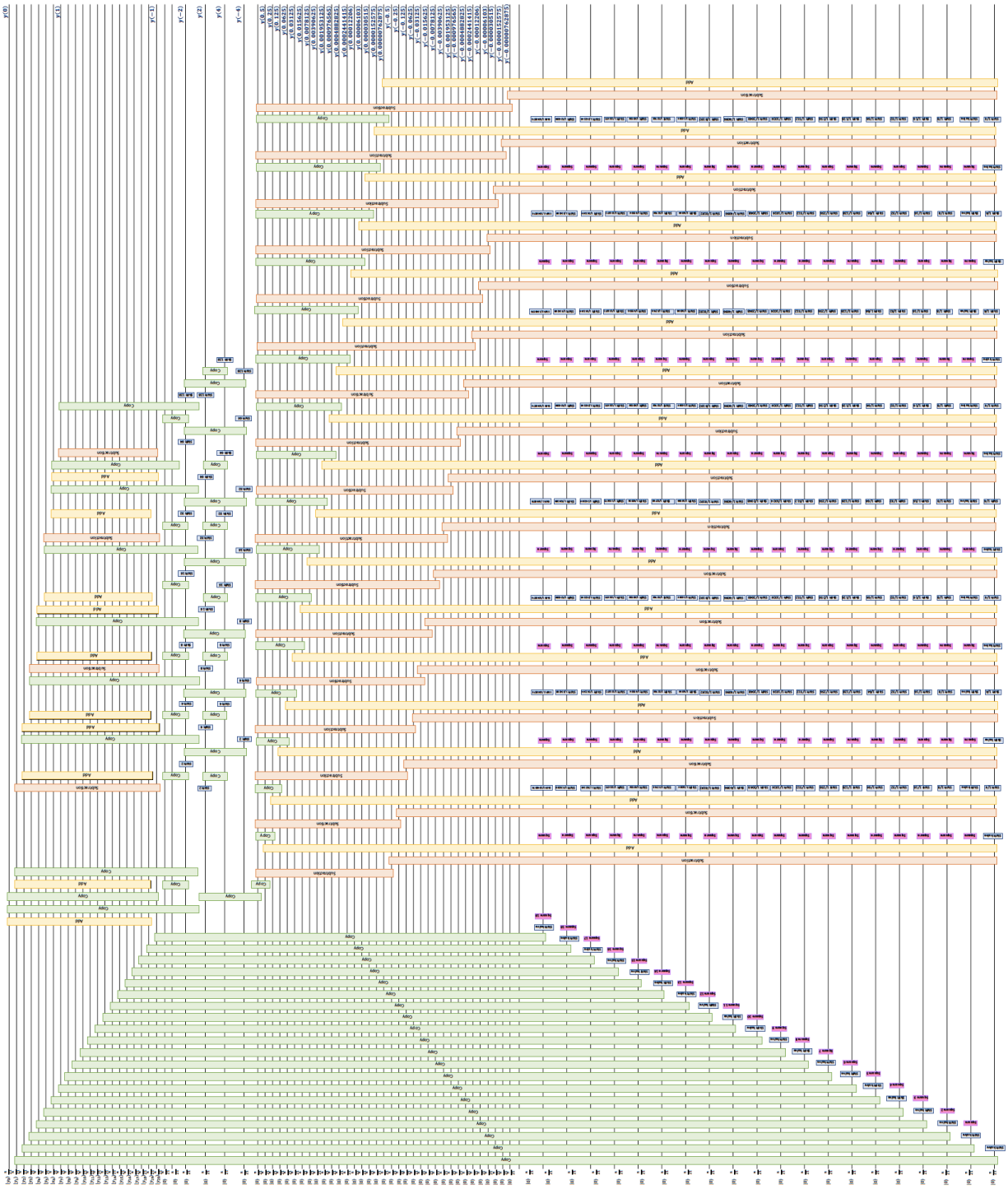

Fig. 7: Evaluation point y

Table 5: Asymptotic Performance and Quantum Implementation Cost Comparison. This table summarizes the asymptotic performance and quantum resource costs of naive schoolbook, Karatsuba, and Toom–Cook–based multiplication methods. The comparison is based on Toffoli gate count, qubit count, and Toffoli depth, which jointly characterize the space–time complexity of each approach.

No	Reference	Arithmetic Algorithm	Asymptotic Performance Analysis			Cost of Quantum Implementation of Multiplication			
			Qubit Count	Toffoli Count	Toffoli Depth	Qubit Count	Toffoli Count	Toffoli Depth	CNOT
1	Naïve (2004,[18])	schoolbook	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$4n+1$	$4n^2+3n$	$4n^2-4n+1$	-
2	Naïve improved (2004,[20])	schoolbook	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n\log n)$	-	-	-	-
3	Paldivis et al (2014,[21])	Const Mult	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	$3n+1$	$4n(n+1)$	8n	-
4	Kepley and Steinwandt (2015,[22])	Karatsuba	$\mathcal{O}(n^{\log_2 3})$	$\mathcal{O}(n^{\log_2^3})$	-	-	-	-	$\mathcal{O}(n^{\log_2 3})$
5	Parent et al. (2017, [18])	Karatsuba	$\mathcal{O}(n^{1.427})$	$\mathcal{O}(n^{\log_2 3})$	$\mathcal{O}(n^{1.158})$	$n(\frac{3}{2})^{\frac{\log 2}{(2\log 3-\log 2)}\log_2 n} \approx n^{1.427}$	$42n^{\log_2 3}$	$n(\frac{3}{2})^{1-\frac{\log 3}{(2\log 3-\log 2)}\log_2 n} \approx n^{1.158}$	-
6	Dutta et al. (2018, [3])	Toom-Cook 2.5-way	$\mathcal{O}(n^{1.404})$	$\mathcal{O}(n^{\log_6^{16}})$	$\mathcal{O}(n^{1.143})$	$n(\frac{8}{3})^{\frac{\log 16}{(6\log 16-\log 6)}\log_6 n} \approx n^{1.404}$	$49n^{log_6 16}$	$n(\frac{8}{3})^{1-\frac{\log 16}{(2\log 16-\log 6)}\log_6 n} \approx n^{1.143}$	-
7	Larasati et al.(2021,[8])	Toom-Cook 3-way	$\mathcal{O}(n^{1.35})$	$O(n^2)$	$\mathcal{O}(n^{1.112})$	$n(\frac{5}{3})^{\frac{\log 5}{(2\log 5-\log 3)}\log_3 n} \approx n^{1.353}$	$8n^2+66n^{\log_3 5}-72$	$n(\frac{5}{3})^{1-\frac{\log 5}{(2\log 5-\log 3)}\log_3 n} \approx n^{1.112}$	-
8	Van Hoof (2020, [23])	Karatsuba	$3n$	$\mathcal{O}(n^{log_2 3})$	-	-	-	-	$\mathcal{O}(n^2)$
9	Putranto et al. (2023, [24])	Karatsuba	$3n$	$\mathcal{O}(n^{log_2 3})$	-	-	-	-	$\mathcal{O}(n^{log_2 3})$
10	Putranto et al. (2023, [9])	Toom-Cook 2-way	$\mathcal{O}(n^{1.589})$	$\mathcal{O}(n^{\log_2 3})$	$\mathcal{O}(n^{1.217})$	$n(\frac{3}{2})^{\frac{\log 3}{(2\log 3-\log 2)}\log_2 n} \approx n^{1.589}$	$34n^{\log_2 3}-32n$	$n(\frac{3}{2})^{1-\frac{\log 3}{(2\log 3-\log 2)}\log_2 n} \approx n^{1.217}$	-
11	Putranto et al. (2023, [9])	Toom-Cook 4-way	$\mathcal{O}(n^{1.313})$	$\mathcal{O}(n^{\log_4 7})$	$\mathcal{O}(n^{1.09})$	$n(\frac{7}{4})^{\frac{\log 7}{(2\log 7-\log 4)}\log_4 n} \approx n^{1.313}$	$122n^{\log_4 7}-160n$	$n(\frac{7}{4})^{1-\frac{\log 7}{(2\log 7-\log 4)}\log_4 n} \approx n^{1.09}$	-
12	Putranto et al. (2023, [9])	Toom-Cook 8-way	$\mathcal{O}(n^{1.245})$	$\mathcal{O}(n^{\log_8 15})$	$\mathcal{O}(n^{1.0569})$	$n(\frac{15}{8})^{\frac{\log 15}{(2\log 15-\log 8)}\log_8 n} \approx n^{1.245}$	$112n^{\log_8 15}-128n$	$n(\frac{15}{8})^{1-\frac{\log 15}{(2\log 15-\log 8)}\log_8 n} \approx n^{1.0569}$	-
13	Wardhani et al.(2023, [10])	Toom-Cook 8.5-way	$\mathcal{O}(n^{1.236})$	$\mathcal{O}(n^{\log_9 17})$	$\mathcal{O}(n^{1.053})$	$n(\frac{17}{9})^{\frac{\log 17}{(2\log 17-\log 9)}\log_9 n} \approx n^{1.236}$	$186n^{\log_9 17}-202n$	$n(\frac{17}{9})^{1-\frac{\log 17}{(2\log 17-\log 9)}\log_9 n} \approx n^{1.053}$	-
14	Wardhani et al.(2023, [10])	Toom-Cook 10.5-way	$\mathcal{O}(n^{1.222})$	$\mathcal{O}(n^{\log_{11} 21})$	$\mathcal{O}(n^{1.047})$	$n(\frac{21}{11})^{\frac{\log 21}{(2\log 21-\log 11)}\log_{11} n} \approx n^{1.222}$	$206n^{\log_{11} 21}-132n$	$n(\frac{21}{11})^{1-\frac{\log 21}{(2\log 21-\log 11)}\log_1 1n} \approx n^{1.047}$	-
15	Wardhani et al.(2024, [11])	Toom-Cook 20.5-way	$\mathcal{O}(n^{1.186})$	$\mathcal{O}(n^{\log_{21} 41})$	$\mathcal{O}(n^{1.033})$	$n(\frac{41}{21})^{\frac{\log 41}{(2\log 41-\log 21)}\log_{21} n} \approx n^{1.186}$	$522n^{\log_{21} 41}-540n$	$n(\frac{41}{21})^{1-\frac{\log 41}{(2\log 41-\log 21)}\log_{21} n} \approx n^{1.033}$	-
16	ours	Toom-Cook 25.5-way	$\mathcal{O}(n^{1.176})$	$\mathcal{O}(n^{\log_{26} 51})$	$\mathcal{O}(n^{1.03025})$	$n(\frac{51}{26})^{\frac{\log 51}{(2\log 51-\log 26)}\log_{26} n} \approx n^{1.176}$	$648n^{\log_{26} 51}-666n$	$n(\frac{51}{26})^{1-\frac{\log 51}{(2\log 51-\log 26)}\log_{26} n} \approx n^{1.03025}$	-