

LATENCY-OPTIMAL QUANTUM CIRCUITS for IoT Networks

Savo Glisic¹⁾ and Beatriz Lorenzo²⁾,

Abstract- Since future IoT networks will be requiring lower and lower latency for transmitting mission sensitive information and moving towards quantum technology to provide advanced security and speed up information processing, here we survey the work on optimum design of the quantum circuits with the objective to minimize the circuit latency. As a starting point, in Section I, we first document in detail, how aggressive latency reduction will limit the use of universal and reversible gates in the circuits both resulting in depth (latency) extension. The former enable programmability and the latter reduce the energy dissipation in the circuits. Being aware of these losses, we then discuss in Section II latency optimal solutions including implementation aspects. Exact minimization of quantum circuits is discussed in Section III while decomposition of CV operations into a universal gate library is discussed in Section IV. Among several solutions, for example, for *controlled-T gate*, solutions are presented for reducing T gates from 15 → 9, CNOT gates from 16 → 12, and T-latency from 9 → 5. For *1-bit full adder*: T gates reduced from 14 → 8, CNOT gates from 12 → 10, H gates from 4 → 2 and T-latency from 8 → 2.

Key words: Quantum circuits, Minimum latency network, IoT, 7G networks

I INTRODUCTION

Research and development of advanced IoT networks has devoted a lot of space to study of latency in network protocols and circuits implementing the network control algorithms. In anticipation that future IoT network will be using quantum information processing there is a need for providing the network designers with a comprehensive insight into the latency problem in the implementation of quantum computing in general and implementation of quantum circuits (hardware) in particular.

Compared to classical technology, quantum circuits can be parallelized more efficiently, reducing the depth of the circuit and accordingly the processing latency at lower cost. This circuit depth reduction will be referred to as latency compression coefficient. The coefficient is defined as a ratio (>1) of the circuit depth before and after the circuit reconfiguration. Depending on the circuit functionality, its initial cost might be higher but with increased requirements for latency compression the cost might increase with much lower slope than in the case of classical technology. With time the initial cost is also expected to decrease.

¹⁾ Department of Physics, Worcester Polytechnic Institute, Massachusetts, USA, (sglisic@wpi.edu; savo.glisic@ieee.org)

²⁾ ECS Department, University of Massachusetts, USA, blorenzo@umass.edu

The objective of this paper is to provide the network designer with a comprehensive survey of the work on reducing both latency and cost of quantum circuits. As the first step we need to be aware of the drawbacks the aggressive reduction of circuit depth (latency) has on limitations of using universal gates and reversible gates for circuit design. The former enable programmability and the latter reduce the energy dissipation in the circuit. Using these types of gates results generally in longer depth/latency. The following several tables compare practical data for these circuits.

Table 1: Latency Comparison — Universal vs. Application-Specific Quantum Circuits

No.	Feature	Universal Quantum Circuit	Application-Specific Quantum Circuit
1	Purpose	General-purpose computation	Tailored to a specific algorithm or application
2	Flexibility	High flexibility to emulate any unitary operation [1]	Limited to a fixed function or narrow task [2]
3	Gate Count (Typical)	$10^4 - 10^6$ gates for moderate-depth circuits [3]	Often $< 10^3$ gates depending on algorithm [4]
4	Qubit Count	50–100+ qubits in current devices [3]	10–50 qubits, depending on task-specific requirements [5]
5	Execution Speed	Slower due to generality and control overhead	Faster; optimized for fewer gates and shallow depth [6]
6	Latency (per run)	$\sim 10-1000 \mu\text{s}$ (depends on gate count and error correction) [7]	$\sim 1-100 \mu\text{s}$ (smaller circuits, lower overhead) [7][8]
7	Gate Latency	Single-qubit: $\sim 10-100 \text{ ns}$; Two-qubit: $\sim 100-500 \text{ ns}$ [7][9]	Same, but fewer gates result in reduced total latency
8	Reusability	High; supports many algorithms	Low; designed for one function
9	Optimization	General optimization hard due to abstraction and variability	Highly optimized for specific algorithm or cost metric [6][9]
10	Example Systems	IBM Quantum (Qiskit), Google Sycamore [3][5]	QFT, Grover's, Shor's, VQE [4][6]

Table1 provides a latency comparison of universal and application-specific quantum circuits. While supporting programmability universal circuit have one order of magnitude larger latency [1-9]. The textbook [1] introduces the theoretical framework of quantum computing and quantum information, covering quantum gates, algorithms, error correction, and entanglement. Reference [2] introduces an extensible quantum-classical assembly language (*OpenQASM 3*) designed for expressing dynamic, hybrid quantum programs with classical control flow. In [3] authors described the experiment where Google's Sycamore processor performed a quantum sampling task in seconds that would take classical supercomputers thousands of years, demonstrating a major milestone known as quantum supremacy. Reference [5] describes IBM's Quantum System One, a cloud-accessible quantum computing platform featuring superconducting qubit technology, designed for

research, industry, and enterprise applications. Paper [6] introduces the Variational Quantum Eigensolver (VQE), a hybrid algorithm for finding eigenvalues of Hamiltonians on near-term quantum devices, particularly useful in quantum chemistry.

Table 2: Comparison — Reversible vs. Application-Specific/Irreversible Quantum Circuits

No.	Feature	Reversible Quantum Circuit	Application-Specific / Irreversible Quantum Circuit
1	Definition	Implements bijective (one-to-one) transformations — information is preserved [1]	Implements a specific task, may discard or overwrite data (logically irreversible) [10]
2	Gate Types	Toffoli, Fredkin, CNOT, and reversible quantum gates [1][11]	Often optimized using irreversible classical logic and measurement-based operations [6]
3	Flexibility	General-purpose, suitable for embedding any logic reversibly	Task-specific; may not be reversible end-to-end
4	Information Loss	None — conserves information (unitary operations) [1]	Possible — uses measurement or irreversible functions [10]
5	Energy Dissipation	Theoretically zero (per Landauer's principle) [12]	Non-zero due to erasure of information [12][13]
6	Latency (Gate Time)	Single-qubit: ~10–100 ns; Two/Three-qubit (e.g., Toffoli): ~200–1000 ns [14][15]	Often faster if classical/irreversible logic or fewer gate levels used [13][16]
7	Circuit Depth	Generally deeper due to ancilla and reversibility constraints	Shallower for specific, optimized tasks
8	Gate Count (Typical)	10^3 – 10^6 depending on the logic encoded reversibly [11][14]	Often $< 10^3$ gates tailored to task [6][16]
9	Example Applications	Quantum arithmetic, Shor's algorithm, reversible computing [1][11]	Quantum classifiers, variational circuits, hybrid algorithms [6][17]
10	Example Gates / Circuits	Toffoli, Fredkin, Peres gates; reversible adders/multipliers [11]	Variational Quantum Circuits (VQC), Quantum Approximate Optimization Algorithm (QAOA) [17]

Reference [7] presents optimized implementations of Shor's algorithm, showing that factoring 2048-bit RSA integers could theoretically be done in 8 hours using 20 million noisy qubits. Google's Sycamore processor, a 53-qubit superconducting quantum chip used to demonstrate quantum supremacy and benchmark quantum operations at scale is described in [8].

Paper [9] reports superconducting qubit operations with fidelities sufficient to support surface code quantum error correction, a critical step toward scalable fault-tolerant quantum computing.

Table 2 presents comparison of reversible and application-specific/irreversible quantum circuits. While providing significant reduction of energy dissipation reversible circuit have significantly larger depth/latency and several orders of magnitude higher gate count [10-17].

In [10] Frank discusses how reversible computing is essential for overcoming energy dissipation limits and enabling future scalable and efficient computing technologies and in [11] Toffoli introduces reversible logic gates (notably the Toffoli gate), forming the foundation for reversible and quantum computation. In [12] Landauer formulates the thermodynamic cost of irreversible operations in computing, establishing a fundamental limit on energy efficiency and motivating reversible computation and in [13] Bennett demonstrates that any classical computation can be made logically reversible, forming a theoretical basis for energy-efficient computation.

Maslov in [14] offers benchmark functions and data for evaluating reversible logic synthesis algorithms and tools. In [15] the authors introduce an efficient synthesis algorithm that minimizes the depth of quantum circuits, improving their feasibility on near-term quantum hardware. Reference [16] explores quantum algorithms for simulating chemical systems, highlighting the promise and challenges of quantum computing for chemical applications while authors in [17] propose the Quantum Approximate Optimization Algorithm (QAOA), a variational quantum algorithm designed for combinatorial optimization problems on near-term quantum devices.

Finally, Table 3 presents more detailed comparison of reversible and application specific quantum circuits. In addition to the previously cited references paper [18] presents a Binary Decision Diagram (BDD)-based approach to synthesize reversible logic circuits efficiently for large-scale Boolean functions and in [19] the authors propose systematic methods for synthesizing reversible circuits with applications in low-power and quantum computing architectures.

In summary, the previous data demonstrate that aggressive request for latency reduction will limit circuit programmability and production efficiency and increase energy dissipation. Still, many IoT applications require severe limitations on network latency. Network designers have invested a significant effort to deal with the problem [20-33].

In this paper we review in detail the process of reducing in a controlled/optimal way the latency in the circuits (classical and quantum) when implementing different functionality in the network.

Bearing aware of the losses elaborated above, we discuss in

Section II latency optimal circuit solutions including implementation aspects. Exact minimization of quantum circuits is discussed in Section III while decomposition of CV operations into a universal gate library is discussed in Section IV. Among several solutions, for example, for *controlled-T gate*, solutions are presented for reducing T gates from 15 → 9, CNOT gates from 16 → 12, and T-latency from 9 → 5. For *1-bit full adder*: T gates reduced from 14 → 8, CNOT gates from 12 → 10, H gates from 4 → 2 and T-latency from 8 → 2. The paper structure is presented in Fig.11.

II LATENCY-OPTIMAL QUANTUM CIRCUITS

Following the approach outlined in [34], we present an algorithm for computing an optimal quantum circuit that implements a given unitary transformation on n qubits. This algorithm offers a significant advantage over brute-force methods, achieving approximately a square-root speedup in runtime. While it is primarily designed to find circuits that are optimal in terms of circuit latency, it can be adapted to optimize for other criteria as well. For example, [34] describes a variant of the algorithm that minimizes the number of sequential non-Clifford gates in a circuit. It is important to note, however, that since the brute-force approach has exponential complexity, this improved algorithm is still exponential in nature. As a result, its practical applicability remains limited to relatively small circuits.

Over the years, significant effort has gone into synthesizing optimal circuits for classical—specifically, reversible—Boolean functions. Shende et al. [35] investigated the synthesis of 3-bit reversible logic circuits using NOT, CNOT, and Toffoli gates by constructing circuit libraries and performing iterative searches through them. Building on this, Golubitsky and Maslov [36] extended the approach to 4-bit reversible circuits composed of NOT, CNOT, Toffoli, and 4-bit Toffoli gates. Their work demonstrated highly efficient performance in circuit synthesis.

Although the lookup speeds achieved in [36] are not replicated in the algorithm discussed in [34], the authors of [34] argue that synthesizing unitary quantum circuits is inherently more computationally demanding than synthesizing reversible classical circuits, complicating direct comparisons. Nonetheless, [34] successfully incorporates many of the search strategies from [36] to enhance performance and reduce synthesis time.

Hung et al. [37] addressed a problem more closely aligned with quantum circuit synthesis by developing a method for computing optimal-cost decompositions of reversible logic into NOT, CNOT, and the quantum controlled- \sqrt{X} gate. Leveraging techniques from formal verification, they derived minimal-cost quantum implementations for various logic gates, including the Toffoli, Fredkin, and Peres gates. However, their approach operates within a restricted quantum circuit model, for instance, requiring control qubits to remain strictly Boolean, and is limited to describing only a finite subset of quantum circuits

Table 3: Detailed Comparison — Reversible vs. Application-Specific/Irreversible Quantum Circuits

No.	Feature	Reversible Quantum Circuit	Application-Specific / Irreversible Quantum Circuit
1	Core Principle	Implements reversible logic with unitary operations (no measurement)	Implements specific tasks, may include measurement or irreversible logic
2	Gate Types	Toffoli, Fredkin, Peres, CNOT, NOT (all reversible) [11][18]	U3, RX, RZ, CRY, CNOT, MEASURE, etc. (including irreversible components) [14]
3	Gate Count (Typical)	10^3 – 10^6 gates depending on algorithm and ancilla overhead [19][14]	$<10^3$ for many tasks (e.g., QAOA, VQE circuits) [14][17]
4	Circuit Depth	Typically 10^2 – 10^4 layers depending on logic complexity [14]	Typically 10 – 10^3 , optimized for minimal depth in variational/hybrid models [17][16]
5	Qubit Count (Typical)	5–100+ including ancilla qubits (for garbage cleanup) [19][14]	4–50 qubits depending on target application [6][16]
6	Latency (Total Execution)	10–1000 μ s (deep circuits, no measurement) [19][9]	1–200 μ s (shallower, faster due to measurements and hybrid use) [17][5]
7	Gate Latency	Single-qubit: 20–100 ns; Toffoli: ~300–1000 ns on current tech [19][9]	Single-qubit: 20–100 ns; CNOT: ~150–300 ns; Measurement: ~500 ns [17][5]
8	Power Efficiency	Theoretically zero energy loss per Landauer’s limit (ideal) [11][12]	Non-zero due to state collapse and garbage output [12][10]
9	Fault Tolerance Requirement	High (error propagation in long-depth reversible logic) [14]	Lower (shorter depth and hybrid tolerance) [17][5]
10	Example Circuits	- Reversible full adder (3 qubits, 10–20 gates) [18] - Reversible multiplier (6–10 qubits, ~100 gates) - Shor’s modular exponentiation (1000+ gates) [19]	- QAOA circuit ($p=1$: ~60 gates, 8 qubits) [16] - VQE (UCCSD ansatz: 100–200 gates, 12 qubits) [6]
11	Applications	Arithmetic logic, cryptography, quantum simulation, Shor’s algorithm [19][14]	Optimization (QAOA), chemistry (VQE), ML (quantum classifiers) [6][17][5]

on n qubits, using four-valued logic.

In contrast, the work presented in [34] optimizes over the full, continuous space of quantum circuits—infinately many possibilities—and supports arbitrary gate sets. It generates circuits not only for Boolean operations but for general quantum gates and permits optimization with respect to a variety of cost functions.

Maslov and Miller [38] also explored synthesis of 3-bit circuits over this gate set, using a pruned breadth-first search approach akin to that in [39], rather than relying on formal verification techniques.

The problem of optimal quantum circuit synthesis remains relatively underexplored, with much of the existing work focused on approximate synthesis within small state spaces. In contrast, here we emphasize finding exact decompositions for various logical gates, though the algorithm can be naturally extended to produce approximate gate sequences as well.

Dawson and Nielsen [40] introduced an algorithm for computing ε -approximations of single-qubit gates in time $O(\log^{2.71}(1/\varepsilon))$, and further generalized it to multi-qubit systems. Their method provides a constructive proof of the Solovay–Kitaev theorem [41], which guarantees that any unitary can be approximated to within error ε using a sequence of gates of depth logarithmic in $1/\varepsilon$. However, the resulting circuits are often far from optimal in terms of depth or gate count [42, 43].

The Dawson–Nielsen algorithm operates by recursively approximating unitaries, with the base case relying on a lookup among previously generated gates for a coarse approximation. In contrast, the algorithm presented here is designed to find minimal-depth exact circuits and could also be employed to accelerate the base case lookup step in the Solovay–Kitaev algorithm by providing a more efficient method for identifying optimal base approximations.

Perhaps more closely related to the approach presented here, Fowler [44] proposes an exponential-time algorithm for finding depth-optimal ε -approximations of single-qubit gates. His method leverages precomputed equivalences between gate subsequences to prune the search space, effectively discarding entire families of redundant sequences from consideration.

While this technique is powerful, we argue that the algorithm presented in the following sections offers superior asymptotic behavior. Moreover, our methods for reducing the search space are not only more general but also demonstrably more effective in practice, especially when scaling to larger systems or more complex cost metrics.

More recently, Bocharov and Svore [45] introduced a depth-optimal canonical form for single-qubit circuits over the gate set $\{H, T\}$. Leveraging this canonical form, they achieve a significant speedup over brute-force methods by searching

through precomputed databases of canonical circuits to find depth-optimal ε -approximations.

In this regard, their work shares some similarity with the approach presented here. However, their method is limited to single-qubit circuits over the specific $\{H, T\}$ gate set, whereas approach presented here applies to arbitrary n -qubit circuits and supports any gate set. Although our method incurs slower search times, it requires significantly less memory, making it more scalable in constrained environments.

Furthermore, this section focuses specifically on the synthesis of small, optimal multi-qubit circuits, a problem that single-qubit synthesis algorithms such as [45, 43] are not equipped to address.

In the circuit model of quantum computation, wires represent quantum bits (qubits), and gates act to transform their state. The state of an n -qubit system is typically described by a vector in a $2n$ -dimensional complex Hilbert space \mathcal{H} , and quantum gates correspond to linear operators on \mathcal{H} .

In this context, we focus exclusively on unitary operators, that is, operators U satisfying $UU^\dagger = U^\dagger U = I$, where U^\dagger is the adjoint (conjugate transpose) of U , and I is the identity operator. The overall transformation implemented by a quantum circuit is the sequential composition of its constituent gates, and since the composition of unitary operators is also unitary, it follows directly that the linear operator represented by the entire circuit is itself unitary.

An individual quantum gate typically acts non-trivially on only a subset of the qubits in a system. To represent this formally, it is convenient to express the unitary operation of the gate as a tensor product: the non-trivial unitary acting on the targeted qubits, combined with the identity operator acting on the remaining qubits. This representation not only captures the gate’s localized action but also highlights the parallelism inherent in quantum circuits.

For example, consider two gates g_1 and g_2 acting on disjoint subsets of qubits. If g_1 is represented by $(g_1 \otimes I)$, and g_2 by $(I \otimes g_2)$, then their sequential composition can be rewritten as the parallel operation $g_1 \otimes g_2$, illustrating how independent gates may be applied simultaneously in a quantum circuit.

The primary optimization criterion used in this work is the depth of a circuit, which we define as the length of the longest critical path through the circuit. When a quantum circuit is represented as a directed acyclic graph (DAG)—with nodes corresponding to gates and edges representing the flow of qubit inputs and outputs—a critical path is any path from an input node to an output node that has maximum length. The depth of the circuit is thus determined by the number of sequential operations along this path (see Fig. 1).

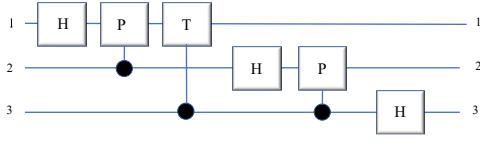


Fig. 1 A quantum circuit implementing the quantum Fourier transform (QFT), up to a permutation of the output qubits, is shown. This circuit has a depth of 5, with two critical paths extending from input 1 to output 3.

The problem of quantum circuit synthesis involves constructing a circuit—composed solely of gates from a specified instruction set—that implements a given unitary transformation. This *instruction set*, denoted \mathcal{G} , is required to include the inverse of each of its gates to ensure completeness and reversibility. An n -qubit circuit over the instruction set \mathcal{G} is defined as a composition of individual gates, each acting on a non-empty subset of the n qubits, and tensored with the identity operator on the remaining qubits.

We can construct depth-one circuits over n qubits by combining gates from a given instruction set \mathcal{G} , where each gate acts on a distinct subset of qubits. These depth-one circuits form a fundamental component of our synthesis algorithm. To formalize this, we define $\mathcal{V}_{n,\mathcal{G}}$ as the set of all unitaries corresponding to depth-one n -qubit circuits constructed from gates in \mathcal{G} .

An n -qubit circuit C over the instruction set \mathcal{G} is said to have depth at most m if it can be expressed as a sequence of unitaries: $C = U_1 U_2 \cdots U_m$, where each $U_i \in \mathcal{V}_{n,\mathcal{G}}$. Furthermore, we say that circuit C implements a unitary $U \in U(2^n)$ if: $U_1 U_2 \cdots U_m = U$.

In general, multiple distinct circuits can implement the same unitary transformation. While we often do not explicitly distinguish between a circuit and the unitary it realizes, it is important to note that the term “circuit” refers to a specific sequence of gates, rather than the resulting unitary operator itself.

Additionally, a key convention must be kept in mind: circuits are mathematically expressed using operator composition, meaning that unitaries are applied from right to left (i.e., the rightmost operator acts first). However, in standard circuit diagrams, gates are drawn and interpreted from left to right, with the leftmost gate acting first on the input state.

With these definitions in place, we can now state our main result. Specifically, we present an algorithm that, given an instruction set \mathcal{G} and a unitary transformation $U \in U(2^n)$, determines whether U can be implemented by a circuit over \mathcal{G} of depth at most l . The algorithm runs in time

$O(|\mathcal{V}_{n,\mathcal{G}}|^{l/2} \log(|\mathcal{V}_{n,\mathcal{G}}|^{l/2}))$ Moreover, if such a circuit exists, the algorithm returns a circuit that implements U with minimal depth over the instruction set \mathcal{G} .

This algorithm represents a significant improvement over the brute-force approach, which has a runtime of $O(|\mathcal{V}_{n,\mathcal{G}}|^l)$. In

practice, with the aid of efficient data structures, it is possible to achieve running times close to $\theta(|\mathcal{V}_{n,\mathcal{G}}|^{l/2})$ yielding a roughly quadratic speed-up.

However, it is important to emphasize that the runtime remains exponential in n . This is due to the fact that for any instruction set \mathcal{G} containing k single-qubit gates, the size of $|\mathcal{V}_{n,\mathcal{G}}|$ grows at least as fast as k^n . As a result, the algorithm is only practical for synthesizing circuits over a small number of qubits.

Motivated by results in fault tolerance, we use the instruction set (gate library) $(H, P, CNOT, T, P^\dagger, T^\dagger)$ shown in Table 4.

Table 4: gate library $(H, P, CNOT, T, P^\dagger, T^\dagger)$

$$\text{Hadamard gate } H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix},$$

$$\text{Phase gate } P = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix},$$

controlled-NOT

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix},$$

$$\text{and } T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}, \text{ along with } P^\dagger \text{ and } T^\dagger.$$

The set of circuits composed from these gates forms a subset of $2^n \times 2^n$ unitary matrices over the ring $\mathbb{Z}[\frac{1}{\sqrt{2}}, i]$ defined as

$$\mathbb{Z} \left[\frac{1}{\sqrt{2}}, i \right] = \left\{ \frac{a + be^{i\pi/4} + ce^{i\pi/2} + de^{i3\pi/4}}{\sqrt{2^n}} \mid \begin{array}{l} a, b, c, d, n \in \mathbb{Z}, \\ n \geq 0 \end{array} \right\}.$$

We also identify two important classes of matrices defined over this ring. The first is the Pauli group on n qubits, denoted \mathcal{P}_n , which consists of all n -fold tensor products of the Pauli matrices (I, X, Y, Z) , possibly multiplied by global phases $\pm 1 \pm i$ or $\pm i$. Formally,

$$\mathcal{P}_n = \{ e^{i\phi} \cdot P_1 \otimes P_2 \otimes \cdots \otimes P_n \mid P_j \in \{I, X, Y, Z\}, \phi \in \{0, \pi/2, \pi, 3\pi/2\} \}.$$

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, (1)$$

The second important class is the Clifford group on n qubits, denoted \mathcal{C}_n , which is defined as the normalizer of the Pauli group \mathcal{P}_n in the unitary group $U(2^n)$. Formally,

$$2\mathcal{C}_n = \{ U \in U(2^n) \mid U\mathcal{P}_n U^{-1} \subseteq \mathcal{P}_n \} (2)$$

In other words, the Clifford group consists of those unitaries that map Pauli operators to Pauli operators under conjugation.

Circuits composed solely of the gates X, Y, Z, H (Hadamard), P (phase), and CNOT produce unitaries in \mathcal{C}_n . Notably, the T gate does not belong to the Clifford group.

A well-known result in quantum computation states that the Clifford group \mathcal{C}_n , when combined with any unitary $U \notin \mathcal{C}_n$, generates a dense subset of $U(2^n)$ [41]. Therefore, the gate set $\{H, P, P^\dagger, CNOT, T, T^\dagger\}$ is universal for quantum computation—capable of approximating any unitary transformation to arbitrary precision, up to a global phase.

1 Meet-in-the-Middle (mm) Search Algorithm

We now provide a high-level overview of the algorithm used to compute optimal quantum circuits. Detailed descriptions of the unitary representations and their approximating circuits will be provided in the following section.

The key insight behind the algorithm lies in a simple yet powerful observation: to determine whether a unitary can be implemented with a circuit of depth l , it suffices to generate and analyze circuits of depth at most $\lceil l/2 \rceil$.

This crucial observation enables a significant reduction in the search space and is the foundation of what is referred to as the "meet-in-the-middle" (mm) algorithm [34].

Lemma 1 [34]: Let $S_i \subset U(2^n)$ denote the set of all unitaries implementable with circuit depth i over a gate set \mathcal{G} . Then, for a given unitary $U \in U(2^n)$, there exists a circuit over \mathcal{G} of depth l implementing U if and only if: $S_{\lceil l/2 \rceil}^\dagger U \cap S_{\lfloor l/2 \rfloor} \neq \emptyset$.

Reference [34] applies this lemma to construct an efficient algorithm that determines whether a unitary U can be implemented by a circuit over the gate set \mathcal{G} with depth at most l . If such a circuit exists, the algorithm also returns an implementation of U with minimum possible latency

```

function mm-FACTOR( $\mathcal{G}, U, l$ )
     $S_0 := \{I\}$ 
     $i := 1$ 
    for  $i \leq \lceil l/2 \rceil$  do
         $S_i := \mathcal{V}_{n,\mathcal{G}} S_{i-1}$ 
        if  $S_{i-1}^\dagger U \cap S_i \neq \emptyset$  then
            return any circuit  $VW$  s.t.
                 $V \in S_{i-1}, W \in S_i, V^\dagger U = W$ 
        else if  $S_i^\dagger U \cap S_i \neq \emptyset$  then
            return any circuit  $VW$  s.t.
                 $V, W \in S_i, V^\dagger U = W$ 
        end if
         $i := i + 1$ 
    end for
end function

```

Given an instruction set \mathcal{G} and a target unitary U , the algorithm incrementally builds circuits of increasing depth and uses them to search for an implementation of U with depth up to $2i$ (see Fig. 2). At each iteration, it generates the set S_i of all circuits of

depth i by extending circuits from S_{i-1} with an additional layer. It then computes the sets $S_{i-1}^\dagger U$ and $S_i^\dagger U$, and checks for intersections with S_i .

According to Lemma 1, a circuit implementing U exists with depth $2i - 1$ or $2i$ if and only if $S_{i-1}^\dagger U \cap S_i \neq \emptyset$ or $S_i^\dagger U \cap S_i \neq \emptyset$, respectively. The algorithm halts at the smallest such depth $\leq l$ and returns a circuit implementing U with minimal depth.

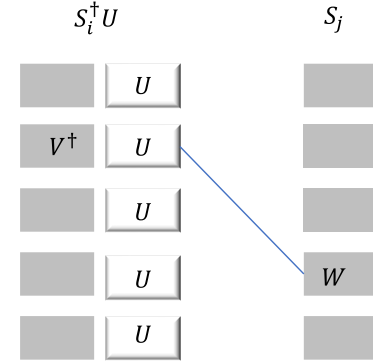


Fig. 2: For each $V \in S_i$ we construct $W = V^\dagger U$ and perform a logarithmic-time search for W in S_j .

To get the stated runtime of $O(|\mathcal{V}_{n,\mathcal{G}}|^{\lceil l/2 \rceil} \log(|\mathcal{V}_{n,\mathcal{G}}|^{\lceil l/2 \rceil}))$ we impose a strict lexicographic ordering on the set of unitaries — for example, by comparing matrices based on the first element at which they differ. This allows the set S_i to be sorted in $O(|S_i| \log(|S_i|))$ time. Consequently, we can search $O(|S_{i-1}| \log(|S_i|))$ and $O(|S_i| \log(|S_i|))$ time, respectively. Since $|S_i| \leq |\mathcal{V}_{n,\mathcal{G}}|^i$, the total runtime for the i -th iteration is bounded by $2|\mathcal{V}_{n,\mathcal{G}}|^i \log(|\mathcal{V}_{n,\mathcal{G}}|^i)$. Thus, the overall runtime remains within the desired complexity bound. Since

$$\sum_{i=1}^{\lceil l/2 \rceil} |\mathcal{V}_{n,\mathcal{G}}|^i \log(|\mathcal{V}_{n,\mathcal{G}}|^i) \leq \sum_{i=1}^{\lceil l/2 \rceil} |\mathcal{V}_{n,\mathcal{G}}|^i \log(|\mathcal{V}_{n,\mathcal{G}}|^{\lceil l/2 \rceil}) \text{ and}$$

$$\sum_{i=1}^{\lceil l/2 \rceil} |\mathcal{V}_{n,\mathcal{G}}|^i \leq |\mathcal{V}_{n,\mathcal{G}}|^{\lceil l/2 \rceil} \left(1 + \frac{1}{|\mathcal{V}_{n,\mathcal{G}}|^{\lceil l/2 \rceil - 1}}\right), \quad (3)$$

so we see that the algorithm runs in

$$O(|\mathcal{V}_{n,\mathcal{G}}|^{\lceil l/2 \rceil} \log(|\mathcal{V}_{n,\mathcal{G}}|^{\lceil l/2 \rceil})) \text{ time. It can also be noted that } \mathcal{V}_{n,\mathcal{G}} \in O(|\mathcal{G}|^n), \text{ so the runtime is in } O(|\mathcal{G}|^{\lceil n \cdot l/2 \rceil} \log(|\mathcal{G}|^{\lceil n \cdot l/2 \rceil})).$$

Optimizing different cost functions: The mm algorithm can be extended to search for circuits optimized according to various cost functions, beyond just depth. For instance, one might define circuit cost as a weighted sum of gates, where each gate in the instruction set is assigned a specific positive weight. As long as all non-identity gates have strictly positive weights, the minimum achievable cost will increase strictly with circuit depth. This ensures that the cost of any discovered solution can serve as an upper bound on the depth that needs to be explored, keeping the runtime still primarily dependent on the depth of the optimal solution.

We concentrate on a particular family of cost functions that is increasingly relevant in fault-tolerant quantum computing. In many fault-tolerant models, gates from the Clifford group can be implemented efficiently, whereas non-Clifford gates are significantly more resource-intensive. For example, in Steane code-based schemes, non-Clifford operations are far more complex to realize [45]. More broadly, for all doubly even self-dual CSS codes—a category encompassing many widely used quantum error-correcting codes—all Clifford operations admit transversal implementations [46], making them relatively straightforward to perform.

In contrast, non-Clifford gates demand advanced methods such as ancilla state preparation and gate teleportation, greatly increasing implementation cost. This contrast is even more pronounced in surface code architectures, which offer high fault-tolerance thresholds but require particularly elaborate procedures to implement the T gate—the most common non-Clifford gate [47]. Consequently, the number of layers involving non-Clifford operations—known as the circuit's T-depth when T is the sole non-Clifford gate—often becomes the primary bottleneck in fault-tolerant quantum computation.

The mm algorithm naturally extends to optimize T-depth in quantum circuits. When the instruction set \mathcal{G} consists of generators for the Clifford group along with the T gate, one can first generate the complete set of Clifford unitaries. Then, by brute-force enumeration, circuits can be systematically explored in increasing T-depth. Applying the mm algorithm, we can efficiently search for a circuit implementing a given unitary by examining candidates with up to twice the target T-depth. This allows for effective synthesis of circuits optimized specifically for minimal T-depth, a critical metric in fault-tolerant quantum computing.

```

function  $mm$ -FACTOR  $T$ -DEPTH( $\mathcal{G}, U, l$ )
   $S_0 := \{\mathcal{C}_n\}$ 
   $i := 1$ 
  for  $i \leq l$  do
     $S_i := (\mathcal{C}_n \mathcal{T}_n \setminus \{I\}) S_{i-1}$ 
    if  $S_{i-1}^\dagger U \cap S_i \neq \emptyset$  then
      return any circuit  $VW$  s.t.
         $V \in S_{i-1}, W \in S_i, V^\dagger U = W$ 
    end if
    if  $S_i^\dagger U \cap S_i \neq \emptyset$  then
      return any circuit  $VW$  s.t.
         $V \in S_i, W \in S_i, V^\dagger U = W$ 
    end if
     $i := i + 1$ 
  end for
end function

```

We define \mathcal{C}_n as the Clifford group on n qubits, implemented using gates from the instruction set \mathcal{G} , and let \mathcal{T}_n be the set of tensor products of I and T . To perform the mm search for minimal T -depth circuits, we initialize $S_0 = \mathcal{C}_n$, and recursively define $S_i = \mathcal{C}_n (\mathcal{T}_n \setminus \{I\}) S_{i-1}$ so that each S_i contains all circuits with T -depth i . The search proceeds by checking whether

$S_{i-1}^\dagger U \cap S_i = \emptyset$ (for T -depth $2i - 1$) or $S_i^\dagger U \cap S_i = \emptyset$ (for T -depth $2i$). This forms the basis for the full algorithm, summarized in the pseudocode above [44].

Searching in this manner becomes increasingly difficult as the dimensionality of the state space grows, due to the exponential growth of the Clifford group with the number of qubits. For instance, the Clifford group on three qubits already contains 92,897,280 elements (up to global phase) [48], making exhaustive search approaches computationally infeasible beyond just a few levels of circuit depth. For four qubits (\mathcal{C}_4), the size of the group is so large that it cannot be stored in memory on typical modern computers, rendering this method impractical without further optimization or pruning strategies.

In practice, we can compute the sets S_i with T -depth $\lceil i/2 \rceil$ by alternating between Clifford and T gate phases. This approach significantly reduces redundancy in the mm computation, as entire phases consisting solely of Clifford group operations can be omitted during the search. Given the exponential growth of the Clifford group with the number of qubits, this optimization yields substantial performance improvements over the more naive algorithm described earlier.

Circuits with ancillas: Another valuable extension of the mm algorithm is its ability to search for circuits that utilize ancillary qubits—additional qubits initialized in the $|0\rangle$ state and required to return to $|0\rangle$ at the end of the computation. While in principle one could consider ancillas in arbitrary initial and final states, we restrict attention to the $|0\rangle$ state for simplicity. This restriction naturally includes any ancilla state that can be prepared and uncomputed using gates from the given instruction set.

More precisely, given a unitary $U \in U(2^n)$, we seek a unitary $U' \in U(2^{n+m})$ such that:

$$U'(|0\rangle^{\otimes m} \otimes |\psi\rangle) = |0\rangle^{\otimes m} \otimes (U|\psi\rangle) \quad (4)$$

This guarantees that the ancilla qubits, initialized in the $|0\rangle^{\otimes m}$ state, are returned unchanged after the computation. To find such a U' , it suffices for U' to match U on the subspace spanned by states of the form $|0\rangle^{\otimes m} \otimes |\psi\rangle$, i.e., for U' to agree with U on the first 2^n rows and columns (those corresponding to input and output states where ancillas remain in $|0\rangle^{\otimes m}$).

However, a major challenge arises if we restrict our search to only those rows and columns. In doing so, we risk missing valid circuit decompositions. For instance, it could happen that: $V'W'(|0\rangle^{\otimes m} \otimes |\psi\rangle) = |0\rangle^{\otimes m} \otimes (U|\psi\rangle)$ but neither V' nor W' individually maps the ancillas cleanly back to $|0\rangle^{\otimes m}$ when acting alone. That is, $V'(|0\rangle^{\otimes m} \otimes |\psi\rangle) \neq |0\rangle^{\otimes m} \otimes (V|\psi\rangle)$ and similarly for W' . As a result, these intermediate steps might appear invalid when checking only partial information (i.e., the top-left submatrix), even though their composition yields the correct overall transformation. This limits the effectiveness of pruning based on subspace agreement and necessitates more

sophisticated strategies to ensure such valid compositions are not discarded during the search.

Instead, we observe that since $|0\rangle^{\otimes m} \otimes (U|\psi\rangle) = (I \otimes U)|0\rangle^{\otimes m} \otimes |\psi\rangle$, our goal reduces to finding unitaries V and W such that $VW(|0\rangle^{\otimes m} \otimes |\psi\rangle) = (I \otimes U)|0\rangle^{\otimes m} \otimes |\psi\rangle$. This leads to the equivalent condition: $V^\dagger(I \otimes U)|0\rangle^{\otimes m} \otimes |\psi\rangle = W(|0\rangle^{\otimes m} \otimes |\psi\rangle)$.

Therefore, rather than comparing full unitaries, we can restrict our attention to how these operators act on inputs of the form $|0\rangle^{\otimes m} \otimes |\psi\rangle$. In practical terms, this means comparing only the first 2^n columns of the unitaries in the sets $S_{[i-1,i]}^\dagger(I \otimes U)$ and S_i , as these columns correspond to the action on the relevant input subspace.

However, this relaxation comes at a cost. Since we're no longer working with full unitaries, we lose the ability to eliminate redundant circuit permutations as efficiently as in the original *mm* algorithm. Consequently, more candidate circuits must be generated and compared, increasing both memory and computational overhead during the search. This trade-off is necessary to accommodate ancillary qubits but should be carefully managed in practical implementations.

2 Search tree pruning

To further reduce the size of the search space, we apply search tree pruning, which in practice yields significant improvements in both memory usage and runtime. The search tree here refers to the conceptual structure in which each node represents a partial circuit, and each branch represents the application of an additional gate from the set $\mathcal{V}_{n,G}$.

Each level S_λ in this tree corresponds to all circuits of depth λ , and the *mm* algorithm generates this tree in a breadth-first manner. Without pruning, the size of the tree grows rapidly, as each circuit in $S_{\lambda-1}$ spawns many child circuits by appending a gate, leading to an exponential blowup.

Pruning works by identifying and eliminating circuits that are redundant or equivalent under some symmetry or canonical form. For instance, if two partial circuits yield the same resulting unitary (up to global phase or circuit equivalence), we can safely discard one without affecting the completeness of the search.

By keeping only the lexicographically minimal representative of each equivalence class (as determined by a strict ordering on unitaries), we dramatically shrink the number of unique paths that must be explored. This reduction not only decreases the total memory footprint, as fewer circuits must be stored, but also accelerates the search process by avoiding repeated computations on functionally identical subcircuits.

We introduce an equivalence relation “ \sim ” on n -qubit unitaries by declaring $U \sim V$ exactly when V can be obtained

from U by permuting qubit labels, taking the inverse, or multiplying by an overall phase. Each unitary $U \in U(2^n)$ thus falls into a class $[U]=\{V \in U(2^n) | U \sim V\}$. Rather than storing every possible circuit, we select a single, depth-optimal implementation for one distinguished (canonical) member of each class. Whenever a new circuit is generated, we compute its canonical representative and check whether we have already recorded a minimal-depth realization for that representative—discarding duplicates and thereby pruning the search tree

We impose a total order on $U(2^n)$ by comparing their matrix entries lexicographically, and call the smallest element in each equivalence class its canonical representative. Concretely, for any n -qubit unitary U we generate all transforms arising from (i) permuting qubit labels—which corresponds to simultaneous row-and-column permutations of U 's matrix—and (ii) taking the inverse (i.e. the conjugate transpose). There are $2^n n!$ such candidates, and by scanning them we pick the lexicographically minimal matrix in $O(n!)$ time. In practice, this extra cost per unitary is negligible for small n , since the time spent lex-ordering is dwarfed by the effort of querying the circuit database.

Selecting a canonical representative up to global phase requires one extra step. When our gate set is restricted to unitaries over the ring $\mathbb{Z}[1/\sqrt{2}, i]$, the only admissible overall phases are $e^{i\theta} \in \mathbb{Z}[1/\sqrt{2}, i] \Leftrightarrow \theta = k\pi/4, k \in \mathbb{Z}$ so there are exactly eight distinct phase factors [43]. Thus, for an n -qubit unitary U we first form all variants obtained by (1) permuting the qubit labels (simultaneous row-and-column permutations), (2) inverting via conjugate transpose, and (3) applying each of the eight allowed phases. In total there are $8 \times 2^n n!$ candidates, and we simply pick the lexicographically smallest matrix among them as the canonical representative.

However, explicitly generating all eight candidate phases is costly. Instead, we fix the global phase by choosing a single “reference” entry in the matrix and rotating the entire unitary so that this entry becomes real and positive. Concretely, we scan the $2^n \times 2^n$ matrix in row-major order to find its first nonzero element u_{ij} then multiply U by $e^{-i \arg(u_{ij})}$. This single normalization step uniquely removes the global phase without enumerating all eight possibilities.

Let $re^{i\theta}$ be the first nonzero entry of U . Naïvely, one might normalize by multiplying U by $e^{-i\theta}$, since then any equivalent $V = e^{i\varphi} U$ satisfies $e^{-i(\theta+\varphi)} V = e^{-i\theta} U$ —but if θ is not a multiple of $\pi/4$, then $e^{-i\theta}$ falls outside $\mathbb{Z}[1/\sqrt{2}, i]$. To stay within the ring, we instead use the factor $re^{-i\theta}$, which always lies in $\mathbb{Z}[1/\sqrt{2}, i]$. Thus we define the canonical representative of U to be $(re^{-i\theta}) U$.

Under this rule, any global-phase equivalent $V = e^{i\varphi} U$ has first entry $re^{i(\theta+\varphi)}$, and applying the same factor yields $(re^{-i(\theta+\varphi)}) V = (re^{-i\theta}) U$.

Because $re^{-i\theta}$ lives in the ring, all comparisons can be carried out symbolically over $\mathbb{Z}[1/\sqrt{2}, i]$, yielding both greater precision and faster performance.

Given a newly generated circuit C of depth i that implements a unitary U , the canonical representative of the equivalence class $[U]$ is first computed. The database of previously discovered circuits is then searched to check whether a circuit implementing this representative already exists. With appropriate data structures, each set S_j of circuits of depth j , where $1 \leq j \leq i$, can be searched in $O(\log(|S_j|))$ time. If no matching circuit is found, a new entry is stored for the circuit implementing the representative of $[U]$. Notably, given a circuit C that implements U , one can obtain a circuit for the canonical representative by applying the corresponding permutation and/or inversion to C .

Permutations are applied by reassigning the qubits on which the individual gates in a circuit act. For inverses, we use the fact that $C^{-1} = C^\dagger$; thus, if $C = U_1 \cdots U_m$, then $C^{-1} = U_m^\dagger \cdots U_1^\dagger$. Since the instruction set is defined such that every gate has an inverse within the set, a circuit for C^{-1} can be constructed by reversing the order of gates in C and replacing each gate with its inverse. As a result, both permutations and inverses of a unitary can be implemented without increasing the circuit depth. This implies that all unitaries within same equivalence class share the same minimum circuit latency.

A subtle but important point is that if we search using only the representatives of equivalence classes at depths i and j , we may fail to discover all equivalence classes at depth $i + j$. Consider a unitary $U = VW$, where V is a circuit of depth i and W is of depth j . Let V' and W' denote the canonical representatives of $[V]$ and $[W]$, respectively. In general, $(V')^\dagger VW \notin [W]$, meaning that searching only via class representatives does not guarantee discovery of $[U]$. However, since $V^\dagger \in [V']$, it follows that $W \in [V']VW$, and thus $[V']U = [W']$.

In practice, this implies that any unitary $U = VW$ can be found by computing the canonical representative of $[V]U$. Therefore, even though intermediate class representatives may not suffice for all compositions, we can still discover all circuits of minimum depth by storing only the representatives of their equivalence classes.

In certain scenarios, an exact implementation of a unitary with the correct global phase is necessary—particularly when the circuit may be used as part of a controlled operation on another qubit. While one could define canonical representatives based solely on qubit relabeling and inversion, it is still possible to construct the correct global phase using any canonical implementation over the gate set $\mathcal{G} = \{H, P, P^\dagger, CNOT, T, T^\dagger\}$, provided the phase is implementable over \mathcal{G} . As observed in [43], if a unitary U is implementable by a circuit over \mathcal{G} , and a circuit C over \mathcal{G} implements $e^{i\theta}U$, then $\theta = k\pi/4$ for some $k \in$

\mathbb{Z} . Since $(HP^\dagger)^3 = e^{-i\pi/4}I$, we have $e^{i\theta}(HP^\dagger)^{3k}U = U$. Therefore, an exact circuit for U , with the correct global phase, can be obtained by composing C with $(HP^\dagger)^{3k}$, yielding a valid implementation over \mathcal{G} .

3 Implementation Aspects

As previously noted, the meet-in-the-middle (mm) algorithm offers no advantage over the naive approach unless appropriate data structures are used. Without them, searching for collisions between the sets $S_i^\dagger U$ and S_j would require $O(|S_i||S_j|)$ comparisons.

However, by imposing a lexicographic ordering on the generated circuits, searches can be performed in logarithmic time relative to the size of S_j . In the implementation, this ordering is realized by storing each S_i as a red-black tree—a type of balanced binary search tree. Balanced trees are widely used in standard libraries and industrial databases for ordered sets and maps due to their reliable performance and scalability. Notably, since deletions—often the most computationally intensive operation in such trees [49]—are never required in the mm algorithm, red-black trees are a particularly well-suited choice for this context.

While hash tables could, in principle, offer performance improvements, an adaptation of the mm algorithm in [34] using the hash table implementation from *libstdc++* showed no measurable speedup. This held true even with a very low rate of key collisions—for example, among 1,316,882 distinct 3-qubit unitaries, no more than 52 were mapped to the same hash value. Despite this favorable collision rate, the hash-based approach did not outperform the red-black tree implementation in practice.

While storing unitaries directly enables efficient generation of new unitaries and rapid searching through circuit databases, the associated memory requirements make large-scale searches infeasible on machines with typical RAM capacities. For an instruction set \mathcal{G} acting on n qubits, the number of distinct unitaries $|\mathcal{V}_{n,\mathcal{G}}|$ grows at least as fast as k^n , where k is the number of single-qubit gates in \mathcal{G} . Using the standard universal instruction set $\mathcal{G} = \{H, P, P^\dagger, CNOT, T, T^\dagger\}$ —which includes generators for the Clifford group along with the T gate—we find $|\mathcal{V}_{3,\mathcal{G}}| = 252$. Consequently, at depth 5, the number of possible circuits exceeds 10^{12} .

If each 3-qubit unitary over $\mathbb{Z}[1/\sqrt{2}, i]$ is stored exactly, it requires 5×64 integers. Thus, storing all depth-5 circuits on 3 qubits would demand more than 1 petabyte of storage, rendering brute-force approaches impractical.

In practice, storing only the representatives of equivalence classes significantly reduces memory usage. For instance, on 3 qubits, there are at most 36,042,958 unique equivalence classes with circuits of depth up to 5, as shown in the experiments reported in [34]. Additionally, the storage required for unitaries

over $\mathbb{Z}[1/\sqrt{2}, i]$ can be further reduced through compression techniques. Nevertheless, even with these optimizations, storing the full unitary matrices becomes infeasible for deeper searches. As a result, a space-time trade-off is unavoidable, requiring careful balance between memory usage and computational cost.

To manage this space-time trade-off, one approach is to store the circuit itself rather than the generated unitary. Specifically, each circuit can be represented as a sequence of depth-1 layers, where each layer is encoded using n bytes—one per qubit—indicating which gate is applied. This compact representation dramatically reduces memory usage. However, storing only the circuit structure introduces a new challenge: searching for a specific unitary becomes computationally expensive. Each comparison would require recomputing the unitary implemented by the circuit, significantly increasing the search time

As a compromise between storage efficiency and search performance, an $m \times m$ matrix M is stored as a key alongside each circuit. For a circuit C implementing a unitary U , the matrix is defined by $M(i, j) = v_i^\dagger U v_j$, where the $\{v_i\}$ are vectors in \mathbb{C}^{2^n} generated pseudorandomly. In practice, even using $m = 1$ suffices to search to meaningful depths for up to 4 qubits, with an extremely low rate of key collisions.

Since these keys are computed using floating-point arithmetic, it is crucial that all other operations—such as circuit evaluation—are performed symbolically. This ensures that unitaries which are mathematically equal produce identical keys despite any numerical error. Alternative experiments in [34] explored using random vectors over $\mathbb{Z}[1/\sqrt{2}, i]$ to eliminate floating-point computation entirely, but this approach resulted in an impractically high number of collisions. Ongoing work is focused on developing improved methods for generating effective and collision-resistant unitary keys.

To further enhance performance during circuit searching, the implementation in [34] parallelizes the search process using the POSIX pthreads library in C. This allows concurrent traversal of balanced binary trees across multiple threads, significantly speeding up lookups. Additionally, generated circuit databases are serialized and stored to disk, eliminating the need to recompute them for each search and enabling efficient reuse across runs.

Table 5 Performance for depth-optimal circuit search [35]

# qubits \ depth	1	2	3	4	5	6	
2	database size (circuits)	14	104	901	6,180	37,878	197,388
	RAM (KB)	2.092	16.686	146.701	1,013.358	6,249.708	32,766.246
	generation time (s)	0.001	0.015	0.155	1.354	10.761	75.301
	search time (s)	0.001	0.004	0.033	0.248	1.672	9.321
3	database size (circuits)	36	1,110	41,338	1,316,882	36,042,958	-
	RAM (KB)	5.633	179.657	6,737.931	215,968.485	7,738,582.749	-
	generation time (s)	0.012	1.059	40.619	1896.301	73,295.675	-
	search time (s)	0.015	0.350	12.619	414.722	11,759.390	-
4	database size (circuits)	84	9,984	1,755,677	-	-	-
	RAM (KB)	13.460	1,617.082	284,596.043	-	-	-
	generation time (s)	0.570	122.966	18,728.922	-	-	-
	search time (s)	0.603	71.420	12,853.887	-	-	-

Depth-optimal implementations: In [34], experiments were conducted to determine depth-optimal decompositions of various 2-, 3-, and 4-qubit logical gates using the gate set $\{H, P, P^\dagger, CNOT, T, T^\dagger\}$. The primary objective was to minimize circuit depth, with gate count serving as a secondary optimization criterion. Table 5 presents performance metrics for several representative gates based on this analysis.

Searches for minimal-depth implementations of various 2-, 3-, and 4-qubit logical operations were performed in [34] using precomputed circuit databases. These searches included depth-optimal decompositions of singly controlled versions of gates such as P , and $V = \sqrt{X} = \frac{1}{2} \begin{pmatrix} 1 & +i & 1 & -i \\ 1 & -i & 1 & +i \end{pmatrix}$ (see Fig. 3). For completeness, depth-optimal implementations of the controlled-Z and controlled-Y gates were also computed and are shown in Fig. 4.

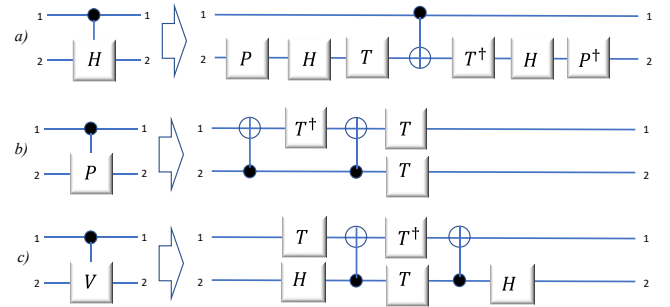


Fig. 3: Logical gate implementations of controlled unitaries without ancillas (a) Controlled-H (T-depth 2, total depth 7), (b) Controlled-P (T-depth 2, total depth 4) (c) Controlled- \sqrt{X} (T-depth 2, total depth 5)

The 2-qubit gate

$$W = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The unitary in Fig. 5 is also optimally decomposed and has found application in a notable quantum algorithm [50] that achieves exponential speedup via a quantum walk. Additional 3-qubit unitaries with minimal-depth implementations include several well-known gates: the Toffoli gate (controlled-controlled-NOT), the Fredkin gate (controlled-SWAP), the quantum OR gate—defined as the unitary mapping $|a\rangle|b\rangle|c\rangle \rightarrow |a\rangle|b\rangle|c \oplus a v b\rangle$ —and the Peres gate [21] (see Fig. 6). Notably, the mm algorithm reduces the total depth of the Toffoli gate from 12 [41] to 8, demonstrating a significant improvement in depth-optimal synthesis.

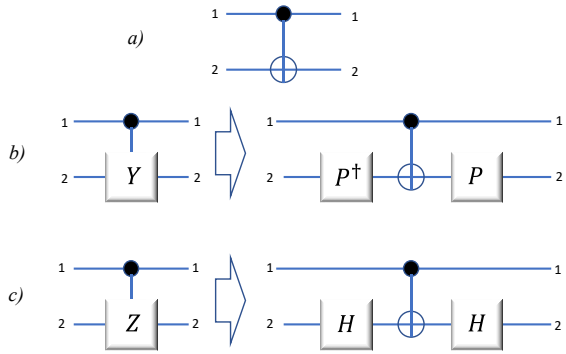


Fig. 4 Controlled Paulis (a) Controlled X (depth 1), (b) Controlled Y (depth 3), (c) Controlled Z (depth 3).
The T -latency of all these circuits is equal to 0

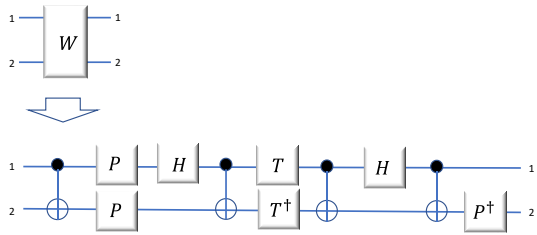
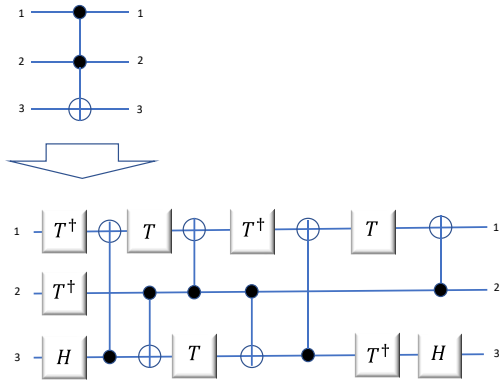
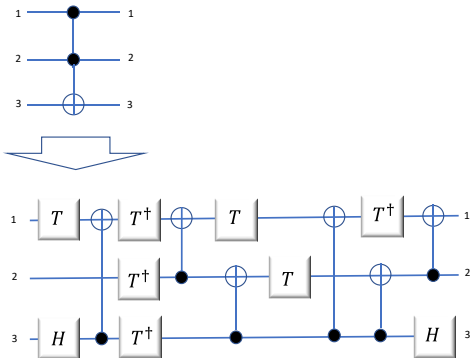


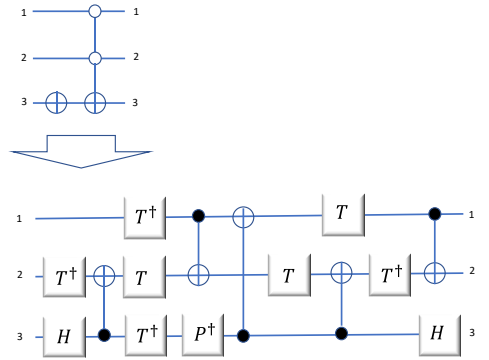
Fig. 5: W gate (T -depth 1, total latency 9)



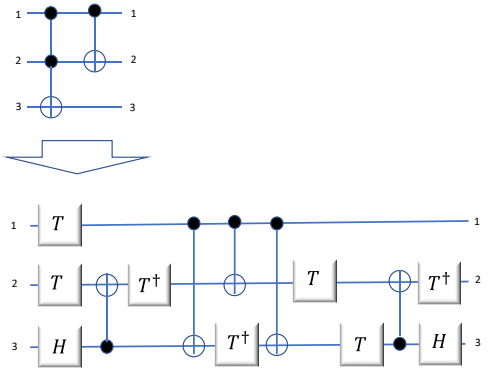
(a) Toffoli gate (T -depth 4, total latency 8)



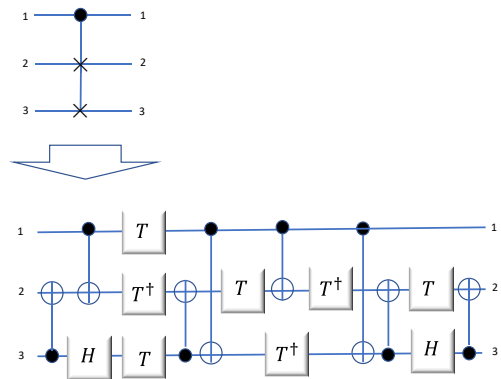
(b) Toffoli gate, one negative control (T -latency 4, total latency 8)



(c) Quantum OR gate (T -depth 4, total latency 8)

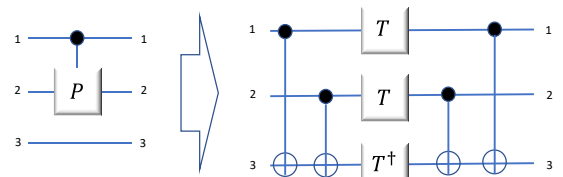


(d) Peres gate (T -depth 4, total latency 8)

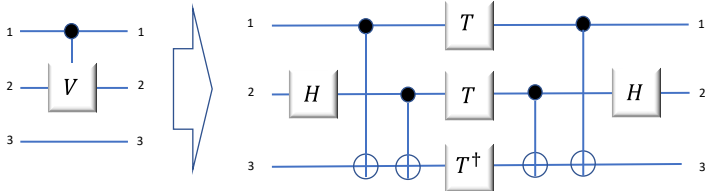


(e) Fredkin gate (T -depth 4, total latency 10)

Fig. 6: 3-qubit logical gates with no ancillas.



(a) Controlled-P (T -latency 1, total latency 5)



(b) Controlled $-\sqrt{X}$ (T -latency 1, total latency 5).

Fig. 7 Reduced T -latency implementations utilizing ancillas. Note that qubit 3 is initialized in and returned to state $|0\rangle$.

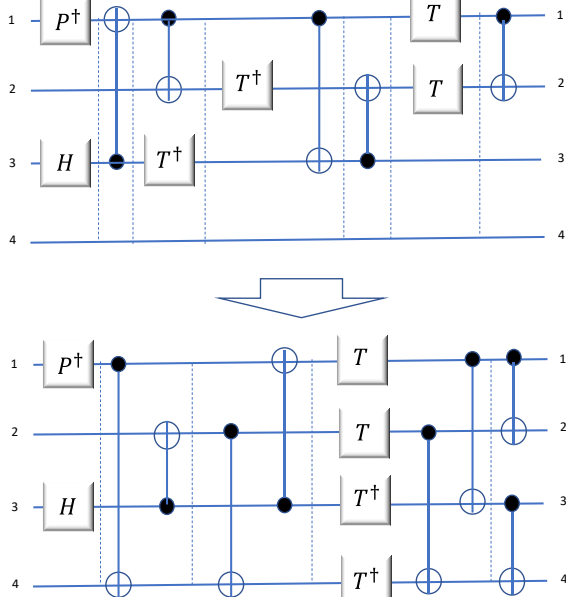


Fig. 8 Addition of one ancilla (qubit 4), initialized and returned in state $|0i$, reduces the minimum circuit latency from 7 (left) to 6 (right).

Searches were also conducted in [34] for each of the aforementioned n -qubit gates using up to $4-n$ ancilla qubits. These searches extended to the maximum circuit depth allowed for the total number of qubits, as summarized in Table 5. While none of the logical gates tested were found to admit decompositions with shorter overall depth or fewer T -gates, alternative circuits for the controlled- P and controlled- \sqrt{X} gates were discovered with reduced T -latency (see Fig. 7). Moreover, a specific unitary was found to achieve a lower minimal depth when decomposed using an ancilla (Fig. 8). Together, these results highlight the potential of ancilla-assisted decompositions to improve circuit execution time, particularly through reductions in T -depth and total latency.

Additional gates were also investigated, including the 3-qubit quantum Fourier transform, which was proven to have no implementation within the instruction set at a depth of 10 or less. Similarly, no depth-optimal circuits of depth 6 or less were found for the 4-qubit Toffoli gate (controlled-Toffoli) or the 1-bit full adder. Furthermore, both the controlled- T and controlled- $\sqrt[4]{X}$ gates were shown to lack implementations of depth 10 or less and 6 or less, respectively, even when using one

or two ancillas. These results underscore the limitations of the instruction set for certain complex operations, particularly at low depths or with minimal ancillary resources.

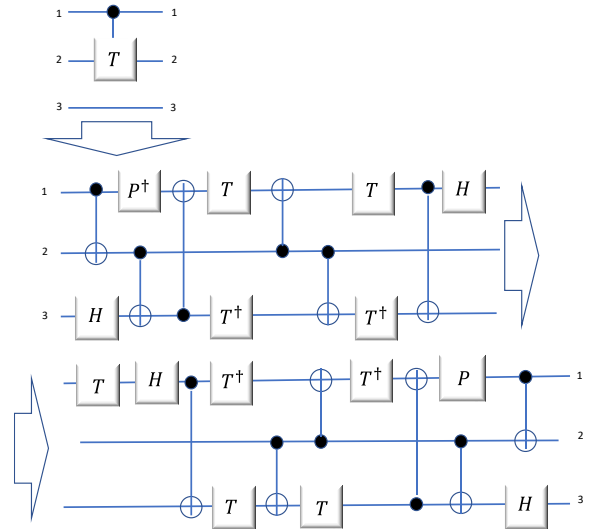


Fig. 9 Circuit implementing a controlled- T gate (T -) latency 5, total latency 19). Note that qubit 3 is initialized in and returned to state $|0\rangle$.

The authors of [34] also applied the mm algorithm to optimize known circuits, including implementations of the controlled- T gate and a 1-bit full adder. For the controlled- T gate, they constructed a circuit based on the decomposition $FREDKIN(I \otimes I \otimes T)$. $FREDKIN$ while the full adder circuit was derived from the construction in [52], replacing the Peres gate with the more efficient version shown in Fig. 6d.

They then applied peephole optimization, a technique where small subcircuits are examined and replaced with more efficient versions synthesized using the mm algorithm. This approach yielded substantial improvements:

Controlled- T gate (Fig. 9): T gates reduced from $15 \rightarrow 9$, CNOT gates reduced from $16 \rightarrow 12$, T -latency reduced from $9 \rightarrow 5$

1-bit full adder (Fig. 10): T gates reduced from $14 \rightarrow 8$, CNOT gates reduced from $12 \rightarrow 10$, H gates reduced from $4 \rightarrow 2$, T -latency reduced from $8 \rightarrow 2$

These results demonstrate the practical effectiveness of peephole resynthesis as a powerful tool for full-scale quantum circuit optimization.

For 2-qubit circuits, the Clifford group comprises 11,520 unique elements (up to global phase), which could be generated in approximately 1 second. Searching for a given unitary up to 1 T -stage, or 2 T -stages ending in a non-Clifford operation, required less than 2 seconds in total. This efficiency was sufficient to identify minimum T -depth implementations for the 2-qubit gates studied.

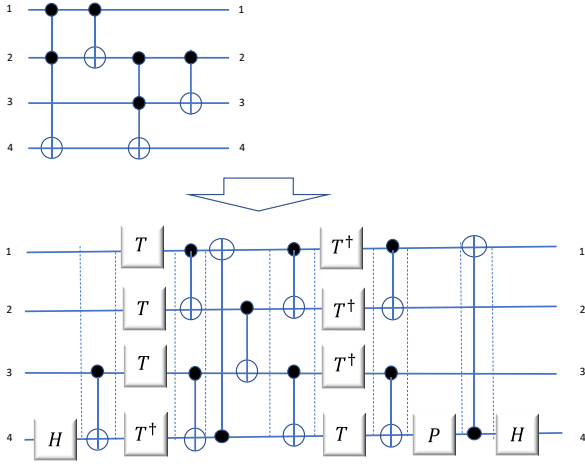


Fig. 10 Circuit implementing a reversible 1-bit full adder

In stark contrast, generating the 92,897,280 unique 3-qubit Clifford group elements took nearly 4 days of computation, illustrating the exponential growth in complexity and the computational bottleneck this presents for scaling to higher qubit counts.

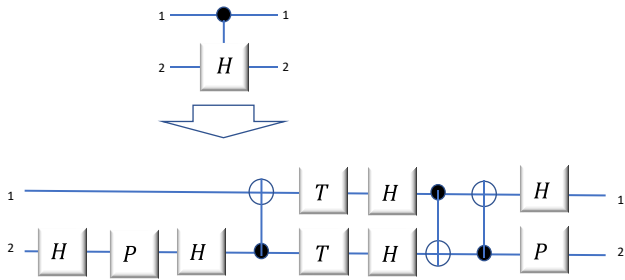


Fig. 11: Circuit implementing a controlled-H gate (T-latency 1, total latency 9)

The minimal TT-depth implementation of the controlled-H gate (Fig. 11) was computed in under one second following the generation of the Clifford group. For other 2-qubit logical gates, however, minimal T-depth circuits did not reduce the number of T-stages beyond those found in the corresponding minimal depth implementations. Consequently, the circuits for controlled-P, controlled- \sqrt{X} , and W are optimal with respect to both overall circuit depth and T-depth.

These findings highlight a key insight: using ancillas can strictly reduce the minimal T-depth required for certain unitaries. In particular, circuits for controlled-P and \sqrt{X} that incorporate ancillas were found with lower T-depth than their ancilla-free counterparts, emphasizing the utility of ancillas in optimizing quantum circuits for execution time.

Although a provably minimal T-depth implementation of the Toffoli gate using zero ancillas has not yet been found, a circuit with T-depth 3 (Fig. 13) was discovered using the main mm algorithm. Given that the Toffoli gate is believed to require at

least 7 T gates, it was conjectured that this implementation achieves minimal T-depth.

This circuit represents a significant improvement over the previously known implementation with T-depth 5 [41], offering an approximate 40% speed-up in fault-tolerant quantum architectures, where Clifford gates are considered inexpensive relative to T gates.

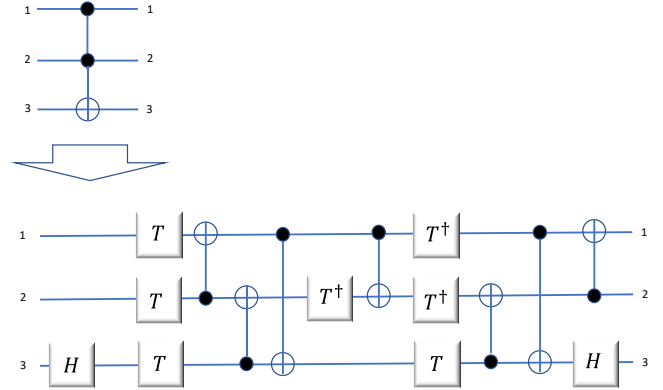


Fig. 12 Circuit implementing a Toffoli gate (T-latency 3, total latency 9).

It's important to note that although these circuits are maximally parallelized, achieving a T-depth of $\lceil m/n \rceil$ for mm T-gates and n qubits, not all circuits can be compressed to this ideal. Some unitaries inherently resist such parallelization, underscoring the complexity of circuit optimization.

Exact decomposition of controlled unitaries : It is well known that a controlled version of any quantum circuit can be constructed by replacing each gate in the circuit with its controlled counterpart, using the same control qubit throughout [53]. The minimal-depth circuits derived in the previous section enable us to formalize the following result.

Theorem 2 [53]. Let the gate cost of a quantum circuit be represented by the $\mathbf{x} = [x_H, x_P, x_C, x_T]^t$ where: x_H is the number of Hadamard (H) gates, x_P is the number of phase (P) or inverse phase (P^\dagger) gates, x_C is the number of CNOT gates, and x_T is the number of T or inverse T (T^\dagger) gates. Suppose a unitary operation U can be implemented within error $\epsilon \geq 0$ using a circuit over the gate set $\mathcal{G} = \{H, P, P^\dagger, CNOT, T, T^\dagger\}$, with gate cost vector \mathbf{x} . Then, the controlled version of UU , denoted controlled- U , can be implemented to within error at most ϵ using a circuit over the same gate set \mathcal{G} , with gate cost given by $A\mathbf{x}$, where A is a constant matrix.

$$A = \begin{bmatrix} 2 & 0 & 2 & 4 \\ 2 & 0 & 0 & 2 \\ 1 & 2 & 6 & 12 \\ 2 & 3 & 7 & 9 \end{bmatrix}$$

The controlled- U circuit requires exactly one ancilla qubit if the decomposition of UU includes one or more T - or T^\dagger -gates, and no ancilla qubits otherwise. Moreover, the controlled- U

can be implemented with a T -depth of at most $x_H + 2x_P + 3x_C + 5x_T$

T gate parallelization: As previously noted, the three T gates in the controlled- P and controlled- \sqrt{X} circuits (Figs. 3b and 3c) can be parallelized to achieve a T -depth of 1 using a single ancilla qubit (Fig. 6). Likewise, the seven T gates in the Toffoli gate decomposition (Fig. 12) can also be parallelized to T -depth 1 with the use of four ancilla qubits. In both cases, the parallelized T gates are interleaved with networks of CNOT gates. A general theorem relating the number of T gates in a $\{CNOT, T\}$ circuit to the minimum achievable T -depth, given a specific number of ancilla qubits, was established in [10].

Theorem 3 [10]. Any circuit on n qubits over the gate set $\{CNOT, T\}$, containing k T gates, can be implemented using a circuit over $\{CNOT, T\}$ on n qubits and m ancilla qubits (each initialized and returned to the $|0\rangle$ state), such that the T -depth of the resulting circuit is at most $\lceil \frac{k}{m+1} \rceil$.

We now proceed to prove *Theorem 2*, as originally presented in [34].

Suppose U is implementable by a circuit over the gate set $\{CNOT, T\}$ using k T gates. Then, from *Lemma 2*, we can $U|a_1 a_2 \dots a_n\rangle = \omega^t |g(a_1, a_2, \dots, a_n)\rangle$ where $t = f_1(a_1, \dots, a_n) + f_2(a_1, \dots, a_n) + \dots + f_k(a_1, \dots, a_n)$ with each f_i a linear Boolean function, and g a linear reversible function.

Assume $k \leq m$, and define a unitary operation F acting on $n+m$ qubits by: $f|a_1 \dots a_n\rangle|b_1 \dots b_m\rangle = |a_1 \dots a_n\rangle|c_1 \dots c_k\rangle|b_{k+1} \dots b_m\rangle$, where each $c_i = b_i \oplus f_i(a_1, a_2, \dots, a_n)$

Since each f_i is linear and the transformation only applies CNOT-like operations to copy f_i into ancilla bits, the map F is linear and reversible. In fact, $F=F^{-1}$, so F is self-inverse and can be implemented using only CNOT gates.

Now, consider the unitary operator

$$V = I^{\otimes n} \otimes T^{\otimes k} \otimes I^{\otimes m-k}, \quad (5)$$

which applies the T gate to the first k ancilla qubits (those holding c_1, \dots, c_k) and identity elsewhere.

$$f^{-1}Vf|a_1 a_2 \dots a_n\rangle|0\rangle^{\otimes m} = \omega^t |a_1 a_2 \dots a_n\rangle|0\rangle^{\otimes m}. \quad (6)$$

Since g is a linear reversible function, U can thus be implemented by a circuit over $\{CNOT, T\}$ in T -depth 1 = $\lceil \frac{k}{m+1} \rceil$.

Now suppose $k > m$. As before, there exists a linear reversible function f implemented by a circuit over $\{CNOT\}$ such that

$$f|a_1 a_2 \dots a_n\rangle|b_1 b_2 \dots b_m\rangle = |a_1 a_2 \dots a_n\rangle|c_1 c_2 \dots c_m\rangle, \quad (7)$$

where $c_i = b_i \oplus f_i(a_1, a_2, \dots, a_n)$. Additionally, f_{m+1} is an output of some linear reversible function h , so the first $m+1$

factors of ω can be computed in T -depth 1 by implementing the unitary $f^{-1}h^{-1}Vhf$, where V is a tensor product of I and $m+1$ T gates.

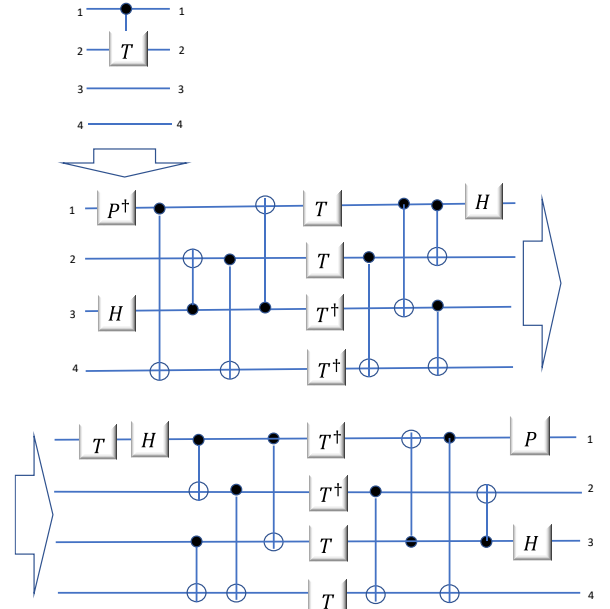


Fig. 13 T -latency 3 implementation of the controlled- T gate

As a result, $(U \otimes I^{\otimes m})|a_1 a_2 \dots a_n\rangle|0\rangle^{\otimes n} = (U' \otimes I^{\otimes m})f^{-1}h^{-1}Vhf|a_1 a_2 \dots a_n\rangle|0\rangle^{\otimes n}$, where $U'|a_1 a_2 \dots a_n\rangle = \omega^{t'} |g(a_1, a_2, \dots, a_n)\rangle$ and $t' = f_{m+2}(a_1, \dots, a_n) + \dots + f_k(a_1, \dots, a_n)$. By *Lemma 18.2* U' can be implemented by a circuit over $\{CNOT, T\}$ with $k - (m+1)$ T gates, and thus U can be implemented in T -depth at most $\lceil \frac{k}{m+1} \rceil$ by induction. \square

In general, it is often possible to achieve a T -depth lower than $\lceil \frac{k}{m+1} \rceil$, since multiple f_i functions can frequently be computed reversibly into data qubits simultaneously. Specifically, when there are l linearly independent Boolean functions to be computed, it is possible to use l data qubits in parallel.

The minimal achievable T -depth for a circuit with n data qubits and m ancilla qubits is determined by the smallest number of groups in a partition $\{S_1, S_2, \dots, S_l\}$ of the set $\{f_1, f_2, \dots, f_k\}$, such that each group S_i satisfies $|S_i| \leq m + \dim(\text{span } S_i)$. This condition ensures that all functions in each group can be computed simultaneously using the available qubits (data + ancilla), enabling more aggressive T -parallelization than the bound given by $\lceil \frac{k}{m+1} \rceil$.

As an example of T -parallelization, [34] demonstrates that the Toffoli gate decomposition, originally implemented with T -depth 3 (Fig. 12), can be rewritten to achieve T -depth 2 by introducing a single ancilla qubit (Fig. 14). Similarly, the implementation of the controlled- T gate (Fig. 9), which has T -depth 3, can be reduced to T -depth 1 with the use of one ancilla

(Fig. 13). While the circuit is not explicitly shown, it is also noted that the 1-bit full adder circuit presented in Fig. 10 can be rewritten to T-depth 1 by employing 4 ancilla qubits.

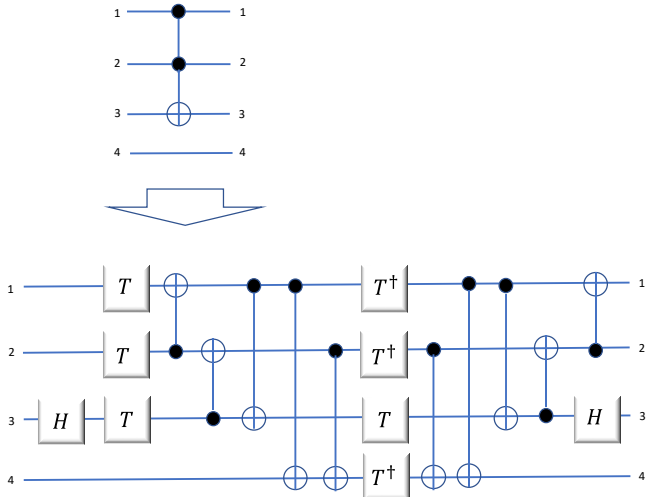


Fig. 14 T - latency 2 implementation of the Toffoli gate

IV EXACT MINIMIZATION OF QUANTUM CIRCUITS

Since non-permutative quantum gates (NPQGs)—such as the controlled square-root-of-NOT gate ($CV = CV^\dagger$)—follow more complex transformation rules than permutative quantum gates (PQGs), synthesizing them is significantly more challenging. An n -letter permutation can be represented as an $n \times n$ binary permutation matrix, where each row and column contains exactly one entry equal to 1, with all other entries being 0. Among these, so-called “magic” permutation matrices are distinguished by having a 1 on the main diagonal, indicating that the identity component of the transformation is preserved. Some of the most well-known examples of permutation matrices correspond to familiar quantum gates, such as the Pauli gates.

$$X = \begin{pmatrix} 01 \\ 10 \end{pmatrix} \equiv (2,1), \quad I \otimes X \equiv (2,1)(4,3),$$

$$\text{CNOT} = \begin{pmatrix} 1000 \\ 0100 \\ 0001 \\ 0010 \end{pmatrix} \equiv (1,2)(4,3),$$

$$\text{CCNOT} \equiv (1,2,3,4,5,6)(8,7),$$

These permutation gates may act on one, two, or even three qubits, depending on the specific operation. Moreover, permutation gates can also be generalized to qudits (quantum systems with $d > 2$ levels). An example is the shift gate, which cyclically permutes the computational basis states of a qudit. Specifically, the shift gate maps $|j\rangle \mapsto |(j+1) \bmod d\rangle$ acting as a modular increment operation on the state index.

$$X = \begin{pmatrix} 010 \\ 001 \\ 100 \end{pmatrix} \equiv (2,3,1)$$

In efficient synthesis algorithms, the direct use of non-permutative quantum gates (NPQGs) should generally be avoided. Instead, the key strategy is to construct new permutative quantum gates (PQGs) using quantum gates in such a way that NPQGs are effectively replaced by equivalent PQG-based structures. This approach relies on a quantum gate library composed of primitives optimized for minimal quantum cost.

Following the methodology outlined in [55], we begin by introducing a set of novel gates similar to the controlled square-root-of-NOT gate ($CV = CV^\dagger$), specifically the controlled k -th root of NOT gates for $k=2,4,8,\dots$. We provide the corresponding unitary matrices for each of these gates.

Furthermore, we present a general and efficient method for directly constructing an optimal quantum logic gate library using combinations of CNOT gates and these NPQGs. This technique also introduces new ways to derive PQGs that exhibit reduced quantum cost, enabling more efficient circuit synthesis overall.

As previously discussed, reversible logic synthesis plays a vital role in quantum information processing. A small number of elementary quantum gates, each with carefully optimized quantum cost, can be combined to construct arbitrary quantum logic circuits. Reducing the quantum cost of a circuit not only enhances its speed and efficiency but also minimizes the likelihood of errors—an essential consideration in practical quantum computation.

However, the synthesis of reversible (permutative quantum) circuits using quantum gates differs significantly from classical (non-reversible) logic synthesis. In addition to the concepts introduced in Section 2, a number of researchers have explored methods for synthesizing quantum circuits with exact minimal cost, focusing on gate sets that are inexpensive to implement physically.

One notable example is the Peres gate, which is often preferred over the standard Toffoli gate due to its lower quantum cost and simpler realization. This illustrates the importance of selecting gate primitives not only for their functionality but also for their implementation efficiency in real quantum systems

Improvements in quantum logic gate design play a critical role in reducing both circuit complexity and execution time. At the heart of this effort lies the challenge of constructing the most cost-efficient equivalents of logic gates like the Peres gate, using only elementary quantum gates. These elementary gates include NOT, CNOT, controlled square-root-of-NOT (also called controlled-NOT^{1/2} or controlled- V/V^\dagger), and even more fine-grained gates such as the controlled fourth-root-of-NOT (controlled-NOT^{1/4} or controlled- W/W^\dagger).

A foundational study by Barenco et al. [56] provided the first comprehensive approach to realizing multiple-control Toffoli (MCT) gates using such elementary gates. This was further

advanced by a method in [57] for systematically determining elementary gate realizations of MCT gates.

Subsequently, [58] introduced two general analytic expressions for simulating n -qubit controlled-U gates using standard single-qubit gates and CNOTs, with exponential and polynomial complexity, respectively. These formulations were accompanied by explicit circuit constructions and general decomposition strategies.

Further innovation came with the introduction of TISC (2-interval symmetric controlled) quantum permutative gates in [59], and the gate library was extended in [60] to include fourth-root-of-NOT gates, broadening the scope for fine-tuned quantum circuit synthesis.

Due to the exponential growth in memory requirements and run-time complexity, only a limited number of existing methods [61–65] are capable of optimally synthesizing 3-qubit circuits using the NCV quantum gate library, which includes NOT, CNOT, and controlled square-root-of-NOT (i.e., controlled-NOT^{1/2}) gates.

A key breakthrough in this area involves reducing NCV circuit synthesis to four-valued logic, which simplifies the process of identifying optimal gate sequences. Notably, [65] introduced an approach that constructs a new quantum gate library using only NCV gates yet is functionally equivalent to the original NCV library in terms of synthesizing all optimal 3-qubit circuits.

This reformulation allows the synthesis problem to be reduced from four-valued logic back to binary logic, which is significantly easier to implement and reason about. As a result, it opens up the possibility for more scalable and efficient synthesis of small-scale quantum circuits using elementary gate primitives.

In this work, we present an efficient 3-qubit synthesis algorithm based on a perfect hash function, capable of rapidly constructing all optimal 3-qubit circuits. Remarkably, the average synthesis speed for circuits with minimum quantum cost is approximately 127 times faster than the best previous method reported in [34].

As part of this approach, we first introduce a series of new gates similar to CV/CV[†]—namely, the controlled k -th root of NOT gates for $k=2,4,8,\dots$ —and provide the corresponding unitary matrices for each. These gates expand the set of non-permutative quantum gates (NPQGs) available for synthesis.

More importantly, we present a novel and generic method for constructing an optimal quantum logic gate library using only CNOT gates and these newly introduced NPQGs. This method enables efficient synthesis by systematically exploring low-cost compositions.

Experiments conducted with this new gate set—as well as supporting results from [55]—demonstrate significant

improvements in synthesis performance. Collectively, these findings introduce a new paradigm for discovering additional permutative quantum gates (PQGs) with lower quantum cost, paving the way for more efficient quantum circuit design.

1 Preliminaries

We already know that the operation of each gate in an n -line reversible or quantum circuit can be represented by a square matrix of dimension 2^n . The matrix of the NOT gate is $N = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$, and $I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ represents the identity circuit. From [60], we also get

$$N^{1/k} = \frac{1}{2} \begin{pmatrix} 1 + i^{2/k} & 1 - i^{2/k} \\ 1 - i^{2/k} & 1 + i^{2/k} \end{pmatrix}; \quad (8)$$

where k is a power of 2. $N^{1/k}$ is a k -th root of N ; thus $(N^{1/k})^k = N$. As a preliminary to this section, we use a number of propositions suggested in [55], starting with (8).

Let $G_k = N^{1/k}$, $k = 2^n$, and $n \in \{0,1,2,3 \dots\}$, then $G_{2^n} = N^{2^{-n}}$, and $G_{2^n}^\dagger$ is the adjoint of the G_{2^n} matrix.

Proposition 1 [55].

$$G_{2^n} = \frac{1}{2} \begin{pmatrix} 1 + e^{i\pi/2^n} & 1 - e^{i\pi/2^n} \\ 1 - e^{i\pi/2^n} & 1 + e^{i\pi/2^n} \end{pmatrix};$$

and its adjoint matrix is

$$G_{2^n}^\dagger = \frac{1}{2} \begin{pmatrix} 1 + e^{-i\pi/2^n} & 1 - e^{-i\pi/2^n} \\ 1 - e^{-i\pi/2^n} & 1 + e^{-i\pi/2^n} \end{pmatrix}$$

Proposition 2 [55]

$$(G_{2^{n+1}})^2 = G_{2^n};$$

$$G_{2^n} G_{2^n}^\dagger = I$$

$$(G_{2^{n+1}}^\dagger)^2 = G_{2^n}^\dagger$$

$$(G_{2^n})^{2^n} = N;$$

$$(G_{2^n}^\dagger)^{2^n} = N$$

Proposition 3 [55]; Let $G_{2^n,m} = (G_{2^n})^m$, where

$$m \in \{1,3,5 \dots, 2^n - 1\}; \text{ then } (G_{2^n,m})^{2^n} = (G_{2^n,m}^\dagger)^{2^n} = N \quad (9)$$

Proposition 4 [55].

$$(G_{2^{n+1,k}})^2 =$$

$$\begin{cases} G_{2^n,k} & k \in \{1,3,5, \dots, 2^n - 1\} \\ N \cdot G_{2^n, k-2^n} & k \in \{2^n + 1, 2^n + 3, \dots, 2^{n+1} - 1\} \end{cases} \quad (10)$$

Definition 1. The controlled square-root-of-NOT gate, also referred to as the controlled-NOT^{1/2} gate, includes both the controlled- V (CV) gate and the controlled- V^\dagger (CV^\dagger) gate. Based on Equation (a), their corresponding unitary matrices are derived and illustrated in Fig. 39

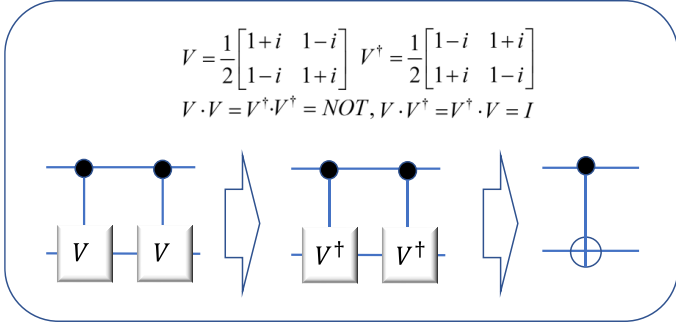


Fig.15 Basic quantum algebra rules for $CV = CV^\dagger$ gates

Definition 2. The controlled fourth-root-of-NOT gate, also known as the controlled-NOT^{1/4} gate, consists of the controlled- W (CW) gate and its adjoint, the controlled- W^\dagger (CW^\dagger) gate.

Here, the gates W and W^\dagger are formally defined as the square root of the square root of the NOT gate, i.e., $W = \text{NOT}^{1/4}$ and $W^\dagger = (\text{NOT}^{1/4})^\dagger$. As shown in Equation (b), the tautological transformations associated with these gates are illustrated in Figure 40.

Definition 3. The controlled eighth-root-of-NOT gate, or controlled-NOT^{1/8} gate, includes the controlled- J (CJ) gate and its adjoint, the controlled- J^\dagger (CJ^\dagger) gate.

These gates are defined by Equation (c), and their corresponding unitary transformations are derived accordingly.

$$G_8 = G_{8,1} = \frac{1}{2} \begin{pmatrix} 1 + e^{i\pi/8} & 1 - e^{i\pi/8} \\ 1 - e^{i\pi/8} & 1 + e^{i\pi/8} \end{pmatrix};$$

$$G_8^\dagger = G_{8,1}^\dagger = \frac{1}{2} \begin{pmatrix} 1 + e^{-i\pi/8} & 1 - e^{-i\pi/8} \\ 1 - e^{-i\pi/8} & 1 + e^{-i\pi/8} \end{pmatrix};$$

from Eq. (d), we get

$$\{G_{8,1}, G_{8,3}, G_{8,5}, G_{8,7}, G_{8,1}^\dagger, G_{8,3}^\dagger, G_{8,5}^\dagger, G_{8,7}^\dagger\} \subseteq \{N^{1/8}\},$$

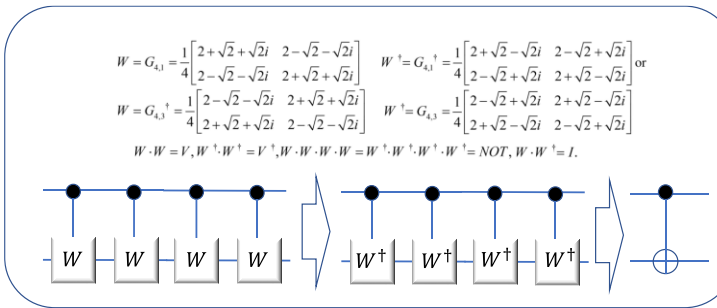


Fig. 16. Basic quantum algebra rules for $CW = CW^\dagger$ gates.

from Eq. (11), we have

$$(G_{8,1})^2 = G_{4,1}, (G_{8,3})^2 = G_{4,3}, (G_{8,5})^2 = N \cdot G_{4,1}, (G_{8,7})^2 = N \cdot G_{4,3}$$

and from Eq. (10), we deduce that

$$(G_{8,1})^4 = (G_{4,1})^2 = V$$

$$(G_{8,3})^4 = (G_{4,3})^2 = N \cdot V = V^\dagger$$

$$(G_{8,5})^4 = (N \cdot G_{4,1})^2 = V$$

$$(G_{8,7})^4 = (N \cdot G_{4,3})^2 = N \cdot V = V^\dagger$$

so we define

$$J = G_{8,1}, J^\dagger = G_{8,1}^\dagger, \text{ or } J = G_{8,3}, J^\dagger = G_{8,3} \quad (12)$$

$$\text{or } J = G_{8,5}, J^\dagger = G_{8,5}^\dagger \text{ or } J = G_{8,7}, J^\dagger = G_{8,7}$$

$$\text{where } J^8 = (J^\dagger)^8 = \text{NOT}; J^4 = V \text{ and } (J^\dagger)^4 = V^\dagger$$

In [55], two fundamental methods were proposed for synthesizing optimal reversible logic circuits using quantum gates: a) Direct synthesis using quantum gates to construct the circuit. b) A two-stage synthesis process, where: First, a set of new logic gates is designed using quantum gates. Then, these newly defined logic gates are used to synthesize the target circuit.

Between the two, the second method is significantly more efficient from an algorithmic standpoint. This is because it reduces the problem to a binary logic synthesis task, which is computationally simpler. Additionally, the newly defined logic gates need to be synthesized only once, after which they can be reused across multiple circuits.

All of these new logic gates are systematically constructed from elementary quantum gates, ensuring that the resulting circuits remain compatible with standard quantum hardware primitives.

Assumption 1. The generated logic gate satisfies the following conditions:

(p1) It is a reversible Boolean logic gate, and all control lines are restricted to Boolean values.

(p2) It is indivisible; that is, it cannot be broken down into simpler logic gates. Specifically, if the gate consists of a sequence of elementary gates, the first and last gates on each qubit must not be logic gates.

(p3) The target level of any controlled gates must not include the target of a CNOT gate.

(p4) The quantum cost of the gate cannot be minimized through any form of equivalence transformation.

(p5) Control lines are strictly limited to Boolean values.

Assumption 2. The logic gate library satisfies the following criteria:

(p1) Optimality: Every gate in the library is optimal in terms of quantum cost.

(p2) Minimality: No gate in the library can be constructed from two or more other gates within the same library without increasing the quantum cost. Naturally, it also cannot be composed at a lower cost, since all gates are already optimal.

(p3) Equivalence: Using gates from the new library yields the same optimal circuit (with the same quantum cost) as using gates from the original library.

The approach exclusively utilizes CNOT gates and controlled- $\text{NOT}^{1/k}$ gates, where $k=2,4,8,\dots$ for constructing logic gates.

Let us now consider functions implemented by n -qubit circuits composed solely of CNOT gates.

Theorem 3.1 [55]. For any Boolean set of input variables x_1, x_2, \dots, x_n applied to an n -qubit circuit composed exclusively of CNOT gates, the output of the circuit must be of the form: $x_1^{(j_1)} \oplus x_2^{(j_2)} \oplus \dots \oplus x_n^{(j_n)}$, where $n \geq 2$; $j = (j_n \dots j_2 j_1)_2$ is the binary representation of j , $0 < j \leq 2^n - 1$, and

$$x_i^{(j_i)} = \begin{cases} 0 & \text{if } j_i = 0 \\ x_i & \text{if } j_i = 1 \end{cases}$$

This is called a linear circuit.

2 Permutative Gate Library

Logic gate construction using CNOT, C- $\text{NOT}^{1/k}$ gates, $k = 2,4,8, \dots$: Let us denote arithmetic addition by $+$ and Sigma and Boolean logic exclusive addition by \oplus .

Theorem 2 [55]. For any set of Boolean variables x_1, x_2, \dots, x_n , we have

$$\begin{aligned} m(x_1, x_2, \dots, x_n) &= \sum_{j=0}^{2^n-1} (x_1^{(j_1)} \oplus x_2^{(j_2)} \oplus \dots \oplus x_n^{(j_n)}) \\ &= \begin{cases} 0 & \text{if } x_i = 0, \quad i \in \{1,2,\dots,n\} \\ 2^{n-1} & \text{else} \end{cases} \end{aligned}$$

where $n \geq 2$, $j = (j_n \dots j_2 j_1)_2$ is the binary representation of j , and

$$x_i^{(j_i)} = \begin{cases} 0 & \text{if } j_i = 0 \\ x_i & \text{if } j_i = 1 \end{cases}$$

DESIGN EXAMPLE 1: We aim to construct an $(n+1)$ -qubit logic gate. The input Boolean variables are denoted as x_0, X_n ,

where x_0 is the input on the bottom line, and $X_n = \{x_1, x_2, \dots, x_n\}$ are the control line inputs.

Suppose we use $2^n - 1$ controlled- $\text{NOT}^{1/2^{n-1}}$ gates, each with its target on the bottom line, and each controlled by a distinct value in the set:

$$\xi(X_n, J_n) = \{x_1, x_1 \oplus x_2, x_1 \oplus x_2 \oplus x_3, \dots, x_1 \oplus x_2 \oplus \dots \oplus x_n\},$$

where each control value corresponds to a binary index $(j_1 j_2 \dots j_n)_2$ such that $1 \leq (j_1 j_2 \dots j_n)_2 \leq 2^n - 1$.

According to *Theorem 2*, for any input X_n , the function $m(X_n)$ —the number of active control signals (i.e., the number of control values evaluating to 1)—must be either 0 or $2^n - 1$. This implies that:

a) If zero control signals are active, then none of the $C - \text{NOT}^{1/2^{n-1}}$ gates are applied, and the output on the bottom line is simply x_0 ; that is, the identity function.

If exactly 2^{n-1} control signals are active, then 2^{n-1} gates are applied in sequence on the bottom line. Since each gate has a rotation angle of $\pi/2^{n-1}$, cascading all of them results in a full NOT gate (i.e., a rotation of π), effectively flipping the value of x_0 .

Therefore, depending on the control inputs, the gate either performs the identity function or a NOT operation on the bottom line. As a result, the entire configuration defines a valid logic gate.

Thus, the newly constructed gate behaves as a reversible Boolean logic gate with a conditional NOT operation controlled by X_n . So in the new gate, we have

$$k = 2^{n-1}, \text{ i.e., } n = \log_2^k + 1. \quad (13)$$

All circuits derived by Barenco [56] using the V and W gates can be seen as special cases under this new method. In this context, we define circuit types based on their gate composition:

A circuit constructed using CV/CV^\dagger and CNOT gates is referred to as a CV-type circuit.

A circuit constructed using CW/CW^\dagger , and CNOT gates is referred to as a CW-type circuit.

These classifications help generalize and unify various existing constructions within the framework of our proposed approach.

If we aim to construct a new logic gate using only CNOT and CG_k/CG_k^\dagger gates, where $k=2,4,8,\dots$ we observe the following: the targets of the CG_k/CG_k^\dagger gates are the G_k/G_k^\dagger gates themselves, which are not logic gates individually. However, a

sequence of k such gates (either all G_k or all G_k^\dagger) can be cascaded to form a valid logic gate.

To construct an $(n+1)$ -bit gate under this framework, all targets of the G_k/CG_k^\dagger gates must be placed on a single line, which we designate as the bottom line for descriptive convenience. The remaining n lines are denoted as $X_n = \{x_1, x_2, \dots, x_n\}$ and serve as the control lines.

Now, consider placing $2^n - 1 - n$ CG_k/CG_k^\dagger gates, each controlled by a unique Boolean function from the set:

$$\{\xi(X_n, J_n) | j \in \{1, \dots, 2^n - 1\},$$

where $k = 2^{n-1}$. According to *Theorem 2*, for any input configuration of X_n , the number of control values evaluating to 1 (i.e., the number of active control signals) must be either 0 or 2^{n-1} .

As a result: If 0 control values are active, none of the CG_k/CG_k^\dagger gates are applied, and the bottom line performs the identity function.

If 2^{n-1} control values are active, then exactly 2^{n-1} G_k (or G_k^\dagger) gates are applied in sequence on the bottom line, together forming a valid logic gate. Therefore, this configuration ensures that the constructed gate behaves as a reversible logic gate, conditionally applying a transformation to the bottom line based on the collective input state of the control lines.

Assumption 3. For a gate to qualify as a logic gate, one of the following conditions must be satisfied for any input configuration:

- (p1) All of the selected gates are CG_k gates.
- (p2) All of the selected gates are CG_k^\dagger gates.
- (p3) The selected gates include an equal number of CG_k and CG_k^\dagger gates.

The control values of the $2^n - 1 - n$ CG_k/CG_k^\dagger gates are given by the set:

$$X_n^* = x_1 \oplus x_2, x_1 \oplus x_2 \oplus x_3, \dots, x_1 \oplus x_2 \oplus \dots \oplus x_n$$

each representing a unique Boolean combination of the input variables.

This portion of the circuit—responsible for generating these control values—can be constructed using a suitable configuration of CNOT gates. The remaining n control values used by the other gates are the original input variables $X_n = \{x_1, x_2, \dots, x_n\}$, which are directly sourced from the inputs of the new gate.

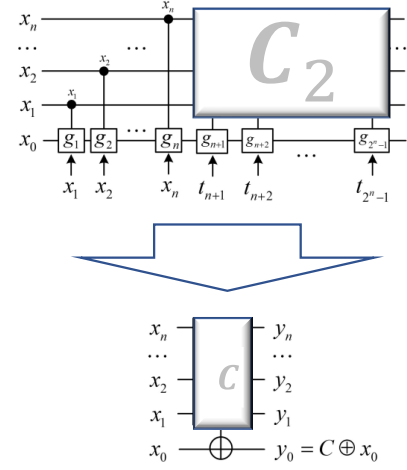


Fig. 17. The framework of new $(n + 1)$ -qubit logic gate implementation using CG_k/CG_k^\dagger and CNOT gates.

As in [55], we introduce a framework for constructing new $(n+1)$ -bit logic gates using CNOT and CG_k/CG_k^\dagger gates, illustrated in Fig. 17. For clarity, we focus on $(n+1)$ -bit gates rather than n -bit gates. In Fig. 17, the set of control values is defined as: $\{T_n\} = \{t_{n+1}, t_{n+2}, \dots, t_{2^n-1}\} = \{X_n^*\}$ where X_n^* represents a specific set of XOR combinations of the input variables X_n .

However, it is important to note that not all logic circuits can be realized using the configuration shown in Fig. 17. To address this limitation, the framework is generalized and extended in Fig. 18. The new framework in Fig. 18 provides a more comprehensive structure, with the framework in Fig. 17 now considered a special case of this more general model.

Theorem 3. In Fig. 4, the new gate is defined with $T_n = X_n^*$. Then, it must hold that $t_1 \in \{X_n\}$.

Proof. Assume, for contradiction, that $t_1 \notin \{X_n\}$. Then it follows that $t_1 \in \{X_n^*\}$, meaning that t_1 is not a direct input variable but rather a Boolean function (specifically, an XOR combination) of the input variables in X_n .

To obtain t_1 , at least one CNOT gate must precede the gate g_1 in order to compute the required control value. This implies that the portion of the circuit before g_1 (which generates t_1) constitutes a logic circuit. The part of the circuit after g_1 , starting from g_1 itself, is also a logic circuit.

However, according to property (p2) of *Assumption 1*, a logic gate cannot be composed of smaller logic circuits. This contradicts our assumption. Therefore, the assumption $t_1 \notin \{X_n\}$ must be false $\Rightarrow t_1 \in X_n$ \square

In Fig. 41, there are multiple possible implementations of the subcircuit C_2 , each of which can be synthesized exclusively using CNOT gates. We now present a specific method for constructing C_2 that generates the required $2^n - 1 - n$ control

values used by the corresponding $2^n - 1 - n$ CG_k or CG_k^\dagger gates.

Clearly, since each of these control values corresponds to a unique Boolean XOR combination of the input variables X_n , the construction of C_2 requires at least $2^n - 1 - n$ CNOT gates.

This raises a fundamental question: How can we synthesize the circuit C_2 using the minimum number of CNOT gates?

The challenge lies in finding an efficient configuration that produces all necessary XOR combinations without redundancy, thereby minimizing quantum cost and circuit latency.

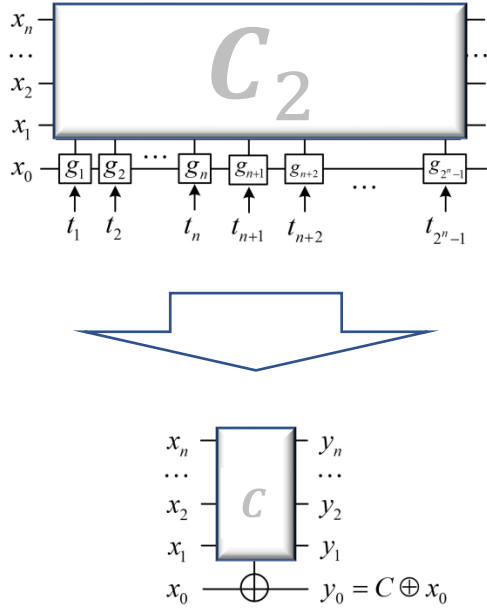


Fig. 18. The generic framework for Fig. 17.

In Fig. 17, there are multiple possible implementations of the subcircuit C_2 , each of which can be synthesized exclusively using CNOT gates. We now present a specific method for constructing C_2 that generates the required $2^n - 1 - n$ control values used by the corresponding $2^n - 1 - n$ CG_k or CG_k^\dagger gates.

Clearly, since each of these control values corresponds to a unique Boolean XOR combination of the input variables X_n , the construction of C_2 requires at least $2^n - 1 - n$ CNOT gates.

This raises a fundamental question: How can we synthesize the circuit C_2 using the minimum number of CNOT gates?

The challenge lies in finding an efficient configuration that produces all necessary XOR combinations without redundancy, thereby minimizing quantum cost and circuit latency.

This approach fundamentally differs from the conventional synthesis method, which relies solely on function permutations. In contrast, the current method involves generating $2^n - 1 - n$

control values using the operator C_2 , corresponding to the same number of CG_n or CG_n^\dagger gates. As a result, the synthesis process appears more complex. However, we demonstrate that a systematic rule exists for constructing C_2 . Based on this rule, [55] presents examples of typical 4-bit logic gates implemented with CG_4/CG_4^\dagger gates (denoted as CW/CW^\dagger), as well as 5-bit logic gates using CG_8/CG_8^\dagger gates (denoted as CJ/CJ^\dagger).

DESIGN EXAMPLE: Gates construction using CNOT, C-NOT^{1/4} (CW/CW^\dagger) Library

The 4-bit logic gates using CG_4/CG_4^\dagger and CNOT gates are given in Figs. 19 and 20. From Eq. (13), $n = (\log_2^k + 1)|_{k=4} = 2 + 1 = 3$, so the new gate has $(n + 1)|_{n=3} = 4$ lines.

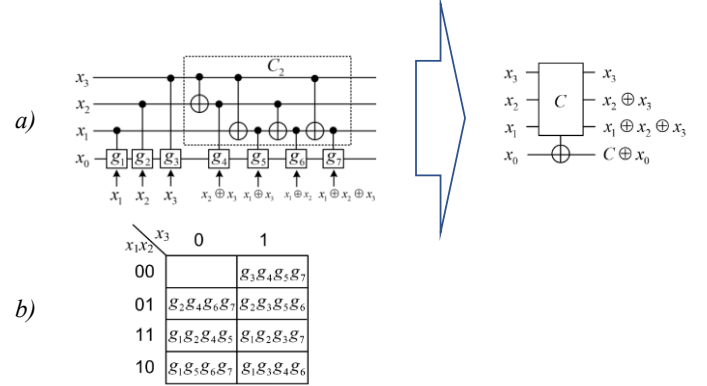


Fig. 19 a) Framework for implementing a novel 4-qubit logic gate using $CG_4 = CG_4^\dagger$ and CNOT gates b) Symbolic graphical representation of the circuit from a) using a quantum Karnaugh map (QMap)..

In the framework from Fig. 19 and Fig. 20, $g_i \in \{G_4, G_4^\dagger\}$, $i \in \{1, 2, 3, 4\}$, and there are 2^4 combinations, but in Table 6, there are only two combinations satisfy the stringent conditions outlined in Assumption 2.

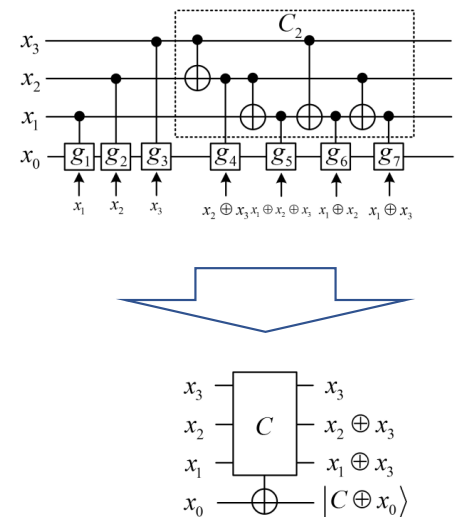


Fig. 20. Another framework for the implementation of new 4-qubit logic gates using CG_4/CG_4^\dagger and CNOT gates.

Table 6 New 4-bit logic gates in Fig 43

No.	g_1	g_2	g_3	g_4	g_5	g_6	g_7	$x_1 x_2 x_3$								C
								000	001	010	011	100	101	110	111	
1	W	W^\dagger	W^\dagger	W	W^\dagger	W^\dagger	W	I	I	I	N	I	I	I	I	$\bar{x}_1 x_2 x_3$
2	W	W	W	W^\dagger	W^\dagger	W^\dagger	W	I	I	I	I	I	I	I	N	$x_1 x_2 x_3$

The structure of 5-bit logic gates using CG_8/CG_8^\dagger and CNOT gates is given in Fig. 21. One C_2 circuit implementation using minimum CNOT gates (i.e., only $(2^n - 1 - n)|_{n=4} = 11$ CNOT gates) is given in Fig. 22.

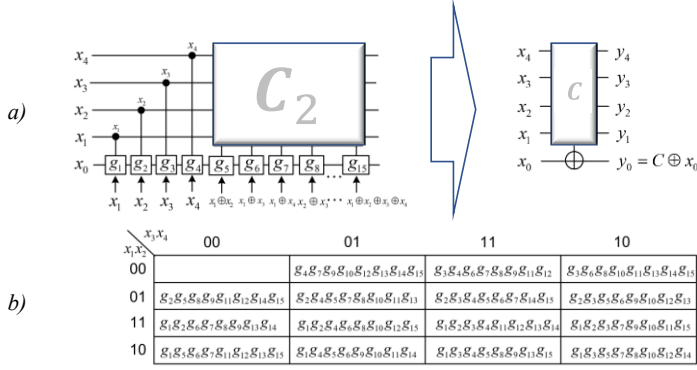


Fig.21. a) Framework for implementing a novel 5-qubit logic gate using CG_8/CG_8^\dagger and CNOT gates. b) Symbolic graphical representation of the circuit from (a) in a QMap, utilizing controlled J and controlled J^\dagger gates.

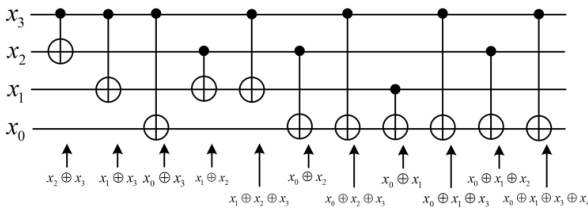


Fig. 22. Circuit C_2 implementation, designed for inclusion in Fig. 21, optimized to use the minimum number of CNOT gates.

DESIGN EXAMPLE: Gate construction using CNOT and $C-NOT^{1/8}$ (CJ/CJ^\dagger)

Let $J = G_8$ and $J^\dagger = G_8^\dagger$. Then in Eq. (12) we see that $J^8 = (J^\dagger)^8 = NOT$. In the framework from Fig. 21, $g_i \in \{J, J^\dagger\}$, $i \in \{1, 2, \dots, 15\}$, and there are 2^{15} combinations, but in Table 7, there are only five combinations that meet the strict requirements of Assumption 2. From (13), $n = (\log_2^k + 1)|_{k=8} = 3 + 1 = 4$, so this new gate has $(n + 1)|_{n=4} = 5$ lines.

Table 7 5-bit logic gates in Fig 45

No.	$x_1 x_2 x_3 \dots x_{15}$	$(x_1 x_2 x_3 x_4)_2$														C	
		1	2	3	4	5	6	7	8	9	10	11	12	13	14		15
1	$J J J J J J J J J J J J J J J J$	I	I	I	N	I	I	I	I	I	I	I	I	I	I	I	$\bar{x}_1 x_2 x_3 x_4$
2	$J^\dagger J^\dagger J^\dagger J^\dagger J^\dagger J^\dagger J^\dagger J^\dagger J^\dagger J^\dagger J^\dagger J^\dagger J^\dagger J^\dagger J^\dagger J^\dagger$	I	I	I	N	I	I	I	I	I	I	I	I	I	I	I	$\bar{x}_1 x_2 x_3 x_4$
3	$J J^\dagger J^\dagger J^\dagger J^\dagger J^\dagger J^\dagger J^\dagger J^\dagger J^\dagger J^\dagger J^\dagger J^\dagger J^\dagger J^\dagger J^\dagger$	I	I	I	N	I	I	I	I	I	I	I	I	I	I	I	$\bar{x}_1 x_2 x_3 x_4$
4	$J^\dagger J^\dagger J^\dagger J^\dagger J^\dagger J^\dagger J^\dagger J^\dagger J^\dagger J^\dagger J^\dagger J^\dagger J^\dagger J^\dagger J^\dagger$	I	I	I	N	I	I	I	I	I	I	I	I	I	I	I	$x_1 x_2 x_3 x_4$
5	$J^\dagger J^\dagger J^\dagger J^\dagger J^\dagger J^\dagger J^\dagger J^\dagger J^\dagger J^\dagger J^\dagger J^\dagger J^\dagger J^\dagger J^\dagger$	I	I	I	N	I	I	I	I	I	I	I	I	I	I	I	$x_1 x_2 x_3 x_4$

DESIGN EXAMPLE: Extension of a standard Toffoli gate

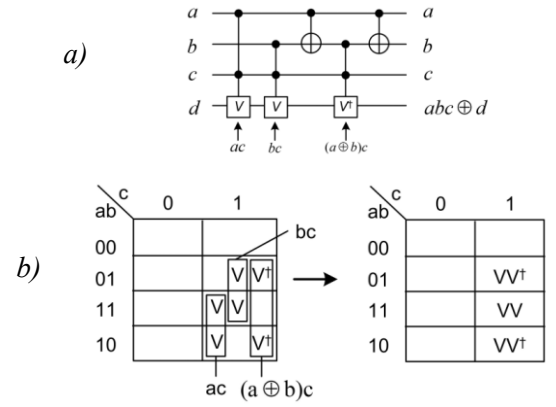


Fig. 23 a) Extension of a standard Toffoli gate to a 4×4 Toffoli gate by multiplying with signal c b) Using QKMAP to analyze the circuit from Fig. a)

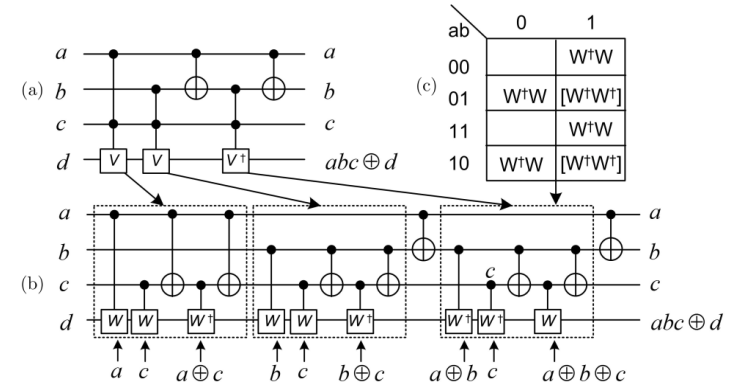


Fig. 24. (a) Circuit from Fig. 23a repeated to help the reader in analysis of Fig. 23b, (b) Realization of the 4×4 Toffoli gate using CW/CW^\dagger gates. Linear controls are explicitly indicated for all W/W^\dagger gates, to facilitate analysis. The blocks indicate the gates of a 3×3 Toffoli gate with additional multiplication by c (from Fig. 23a), (c) QKMap-based analysis of circuit interrupt lines from the right of the circuit.

DESIGN EXAMPLE: Another method for 4-bit gate construction using CW/CW^\dagger and CNOT gates

To implement the function $abc \oplus d$, each double-controlled V/V^\dagger gate must be decomposed into 2×2 gates, as illustrated in Fig. 24. Similarly, controlled Hermitian gates (CCV) are constructed in Fig. 24 using CW/CW^\dagger gates.

The circuit presented in Fig. 24 can be simplified into the circuit shown in Fig. 25 by applying quantum simplification rules. This transformation effectively reconstructs the CCCNOT (triple-controlled NOT) circuit originally introduced by Barenco [56].

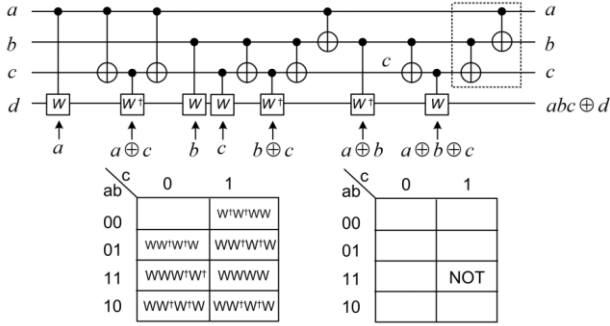


Fig. 25. Simplified circuit derived from Figs. 47 and 48. The identity $WW^\dagger = I$ was applied to gates W/W^\dagger controlled by c in Fig. 24, allowing the removal of two gates from the original configuration. The resulting circuit contains only six controlled W/W^\dagger gates, each governed by a linear control function—demonstrating an affine root of a NOT gate. The final implementation includes seven CW/CW^\dagger gates and eight CNOT gates. A Peres-like gate variant, shown in the top-right of this figure, can be realized with seven CW/CW^\dagger gates and only six CNOT gates by omitting two CNOTs from the circuit.

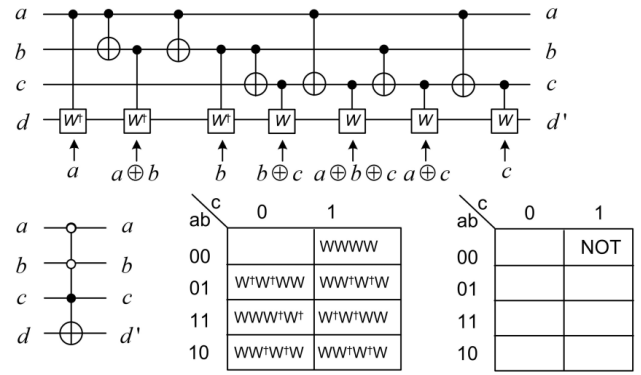


Fig. 27. Realization of function $\sim a \sim bc \oplus d$ with seven CG gates and only six CNOT gates

Fig. 27 demonstrates that strategic placement of CNOT gates can reduce their overall count, building upon the framework established in Fig. 28. However, incorporating the framework from Fig. 17 may result in a higher overall quantum cost.

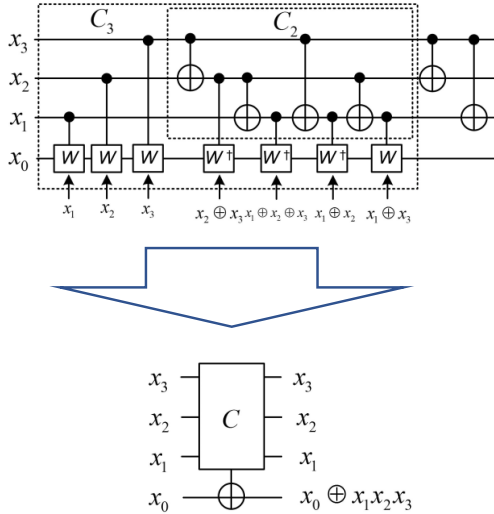


Fig. 25. The new 4×4 Toffoli gate, as implemented in Fig. 20, utilizes CW/CW^\dagger and CNOT gates

In Fig. 50, the 4×4 Toffoli gate is implemented using only seven $CW = CW^\dagger$ gates and six CNOT gates, whereas the final circuit in Fig. 25 requires seven CW/CW^\dagger gates and eight CNOT gates. Despite this, the method shown in Fig. 26 is simpler than the one presented in Fig. 14 and enables more straightforward construction of new and larger quantum logic gates. For example, constructing a C_3 gate using a cascade of one Toffoli-4 gate and two CNOT gates results in a total cost of $13+1+1=15$. In contrast, the method introduced in this section reduces the cost to just 11. Therefore, employing these new quantum logic gates can significantly lower the overall cost of many quantum circuits.

VI CONCLUSION

As a summary of the main results in the design of the circuits for low latency quantum networks we should keep in mind:

Lessons from conventional circuit design:

Energy Savings: Reversible Gates: NOT, SWAP, and CNOT: One promising strategy to mitigate the thermal energy loss associated with irreversible gates is to transition towards reversible logic gates in chip design. In a reversible logic gate, each output is uniquely associated with a corresponding input, and vice versa. This ensures that no information is lost during the computation, making it possible to reverse the computation after obtaining a result.

Universal Reversible Gates: FREDKIN and TOFFOLI: Just as there are universal gates for classical irreversible computing, such as the NAND gate, there also exist universal gates for classical reversible computing. However, the smallest gates that are both reversible and universal require three inputs and three outputs. Two prominent examples of such gates are the FREDKIN gate (also known as the controlled-SWAP gate) and the TOFFOLI gate (also known as the controlled-CNOT gate).

Universal Reversible Basis: NOT-CNOT-TOFFOLI: In the study of reversible circuit synthesis, we commonly work over the NOT-CNOT-TOFFOLI gate set, which serves as a standard universal basis for reversible computation.

Quantum Logic Gates Library

Any classical computation can be decomposed into a series of logic gates, each operating on a small number of classical bits.

Similarly, quantum computations can be constructed from a sequence of quantum gates, each acting on a limited number of qubits. The key distinction lies in their capabilities: classical gates handle definite bit values, either 0 or 1, while quantum gates manipulate complex quantum states, which may be superpositions of basic computational states and can also exhibit entanglement. As a result, the range and nature of quantum logic gates are significantly broader than those in classical systems.

Since a quantum gate must be realized through the physical evolution of an isolated quantum system, its operation is dictated by the Schrödinger equation: $i\hbar\partial|\psi\rangle/\partial t = \mathcal{H}|\psi\rangle$ where \mathcal{H} , the Hamiltonian, defines the system's internal forces and external fields. Consequently, the unitary operator representing a quantum gate corresponds to the system's time evolution and is given by: $U = \exp(-i\mathcal{H}t/\hbar)$. In this expression, the Hamiltonian \mathcal{H} characterizes the interactions responsible for driving the quantum transformation.

We have discussed the main characteristics and construction of:

Latency optimal quantum circuit:

The gate set $\{H, P, P^\dagger, CNOT, T, T^\dagger\}$ is universal for quantum computation, capable of approximating any unitary transformation to arbitrary precision, up to a global phase.

- Meet-in-the-Middle (mm) Search Algorithm
- Latency-optimal implementations: various 2-, 3-, and 4-qubit logical gates using the gate set $\{H, P, P^\dagger, CNOT, T, T^\dagger\}$.
- Optimization gains:

Controlled-T gate (Fig. 9): T gates reduced from 15 \rightarrow 9
 CNOT gates reduced from 16 \rightarrow 12, T-depth reduced from 9 \rightarrow 5

1-bit full adder (Fig. 10): T gates reduced from 14 \rightarrow 8, CNOT gates reduced from 12 \rightarrow 10, H gates reduced from 4 \rightarrow 2
 T-depth reduced from 8 \rightarrow 2

Exact minimization of quantum circuits:

Two fundamental methods were proposed for synthesizing optimal reversible logic circuits using quantum gates: a) Direct synthesis using quantum gates to construct the circuit. b) A two-stage synthesis process, where: First, a set of new logic gates is designed using quantum gates. Then, these newly defined logic gates are used to synthesize the target circuit.

Between the two, the second method is significantly more efficient from an algorithmic standpoint. This is because it reduces the problem to a binary logic synthesis task, which is computationally simpler. Additionally, the newly defined logic gates need to be synthesized only once, after which they can be reused across multiple circuits.

We believe that the presented options for quantum circuit design will help the network designers when choosing the circuit for their project.

REFERENCES	QUANTUM GATES LIBRARY	LATENCY OPTIMAL QUANTUM CIRCUITS	EXACT MINIMIZATION OF QUANTUM CIRCUITS	DECOMPOSING C.V. OPERATIONS INTO A UNIVERSAL GATE LIBRARY	NETWORK LATENCY	CIRCUIT/NETWORK LEVEL PERFORMANCE	COVERAGE OF THE RELEVANT TOPICS
[1-25]					#		15%
[26-33] [67-77]	#						15%
[34-54]		#					15%
[55-65]			#				15%
[66]				#			15%
our paper	#	#	#	#	#	#	100%

Table 10. Topic coverage in existing work

The above topics have been covered partly in existing literature and to the best of our knowledge this is the first unifying survey that includes most of the relevant problems when it comes to planning and designing a quantum network with minimum latency which is of paramount importance for future IoT networks. The comparison of the contribution of our paper with the existing work is summarized in Table 10. The paper establishes a direct relation between the quality of the circuit and the overall network performance.

REFERENCES

[1] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, Cambridge Univ. Press, 2010.
 [2] A. W. Cross et al., "OpenQASM 3," *arXiv preprint, arXiv:2304.05219*, 2023.
 [3] F. Arute et al., "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, pp. 505–510, 2019.
 [4] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *STOC '96*, pp. 212–219.
 [5] IBM Quantum, "Quantum System One," [Online]. Available: <https://www.ibm.com/quantum>
 [6] A. Peruzzo et al., "A variational eigenvalue solver," *Nature Commun.*, vol. 5, no. 4213, 2014.
 [7] C. Gidney and M. Ekerå, "How to factor 2048-bit RSA in 8 hours," *Quantum*, vol. 5, p. 433, 2021.
 [8] Google Quantum AI, "The Sycamore processor," [Online]. Available: <https://quantumai.google>
 [9] R. Barends et al., "Superconducting quantum circuits at the surface code threshold," *Nature*, vol. 508, no. 7497, pp. 500–503, 2014.
 [10] M. Frank, "The future of computing depends on making it reversible," *IEEE Spectrum*, vol. 55, no. 7, pp. 35–41, 2018.
 [11] T. Toffoli, "Reversible computing," in *Automata, Languages and Programming*, pp. 632–644, Springer, 1980.
 [12] R. Landauer, "Irreversibility and heat generation in the

- computing process," *IBM Journal of Research and Development*, vol. 5, no. 3, pp. 183–191, 1961.
- [13] C. H. Bennett, "Logical reversibility of computation," *IBM Journal of Research and Development*, vol. 17, no. 6, pp. 525–532, 1973.
- [14] D. Maslov, "Reversible logic synthesis benchmarks page," [Online]. Available: <https://web.archive.org/web/20201203143617/http://webhome.cs.uvic.ca/~dmaslov/>
- [15] M. Amy, D. Maslov, M. Mosca, and M. Roetteler, "A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 32, no. 6, pp. 818–830, 2013.
- [16] Y. Cao et al., "Quantum chemistry in the age of quantum computing," *Chemical Reviews*, vol. 119, no. 19, pp. 10856–10915, 2019.
- [17] E. Farhi, J. Goldstone, and S. Gutmann, "A quantum approximate optimization algorithm," *arXiv preprint, arXiv:1411.4028*, 2014.
- [18] R. Wille and R. Drechsler, "BDD-based synthesis of reversible logic for large functions," *DAC '09*, pp. 270–275.
- [19] V. V. Shende, A. K. Prasad, I. L. Markov, and J. P. Hayes, "Synthesis of reversible logic circuits," *IEEE Trans. CAD*, vol. 22, no. 6, pp. 710–722, 2003.
- [20] A. S. Shafiqh, B. Lorenzo, S. Glisic and Y. Fang, *Low-Latency Robust Computing Vehicular Networks*, *IEEE Transactions on Vehicular Technology*, 2023}, volume 72, number 2,
- [21] Inosha Sugathapala, Savo Glisic, Markku Juntti, A. Shams Shafiqh, and Le-Nam Tran, "Queue Aware Resource Optimization in Latency Constrained Dynamic Networks," in **2020 IEEE 31st Annual International Symposium on Personal, Indoor and Mobile Radio Communications**, 2020.
- [22] I. Kovacevic, A. S. Shafiqh, S. Glisic, B. Lorenzo, and E. Hossain, "Multi-Domain Network Slicing with Latency Equalization," **IEEE Transactions on Network and Service Management**, vol. 17, no. 4, 2020.
- [23] I. Kovacevic, E. Harjula, S. Glisic, B. Lorenzo, and M. Ylianttila, "Cloud and Edge Computation Offloading for Latency Limited Services," **IEEE Access**, 2021.
- [24] C. Li, J. Li, M. Peng, B. Rasti, P. Duan, X. Tang and X. Ma, *Low-Latency Neural Network for Efficient Hyperspectral Image Classification*, *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 2025, volume 18
- [25] David Zenati, Tzalik Maimon, and Kobi Cohen, "RRO: A Regularized Routing Optimization Algorithm for Enhanced Throughput and Low Latency With Efficient Complexity," *IEEE Journal on Selected Areas in Communications*, vol. 43, no. 2, 2025.
- [26] G. Le, V. T. Hoang, S. Ferdousi, A. Marotta, S. Xu, Y. Hirota, Y. Awaji, M. Tornatore, and B. Mukherjee, "Reliable Provisioning of Low-Latency and High-Bandwidth Extended Reality Live Streams," **IEEE Journal on Selected Areas in Communications**, 2025, Early Access Article.
- [27] D. Li, J. Li, D. Niyato, W. Feng, and W. Jiang, "Deep Energy-Efficient Optimization Network for URLLC Over Cell-Free Massive MIMO," **IEEE Internet of Things Journal**, 2025, Early Access Article.
- [28] D. Townend, S. D. Walker, N. Parkin, and A. Tukmanov, "C-RAN and Optical Fronthaul Latency in Representative Network Topologies," **IEEE Open Journal of the Communications Society**, vol. 6, 2025.
- [29] A. Belli, M. Esposito, S. Raggiunto, L. Palma, and P. Pierleoni, "Relaying Mechanisms in BLE Mesh Networks: A Method for Improving Latency and Reliability," **IEEE Internet of Things Journal**, 2025, Early Access Article.
- [30] P. Qin, Q. Li, and D. Xu, "Decentralized Federated Learning in LEO Satellite-based IoE Communications: Latency Optimization under Reliability Constraints," **IEEE Internet of Things Journal**, 2025, Early Access Article.
- [31] Z. Niu, H. Yang, Q. Yao, B. Wu, S. Yin, S. Shen, B. Wei, J. Zhang, and A. V. Vasilakos, "Reliable Low-Latency Routing for VLEO Satellite Optical Network: A Multiagent Reinforcement Learning Approach," **IEEE Internet of Things Journal**, vol. 12, no. 3, 2025.
- [32] A. Ul Haq, S. S. Sefati, S. J. Nawaz, A. Mihovska, and M. J. Beliatas, "Need of UAVs and Physical Layer Security in Next-Generation Non-Terrestrial Wireless Networks: Potential Challenges and Open Issues," **IEEE Open Journal of Vehicular Technology**, vol. 6, 2025.
- [33] M. Polverini, A. Cianfrani, T. Caiazzi, and M. Scazzariello, "SRv6 Meets DetNet: A New Behavior for Low Latency and High Reliability," **IEEE Journal on Selected Areas in Communications**, vol. 43, no. 2, 2025.
- [34] M. Amy, D. M. , and M. Roetteler, "A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits," *arXiv.org, arXiv:1206.0758*. [Online]. Available: <http://arxiv.org/abs/1206.0758>
- [35] V. V. Shende, A. K. Prasad, I. L. Markov, and J. P. Hayes, "Synthesis of reversible logic circuits," **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, vol. 22, no. 6, pp. 710–722, 2003, *arXiv:quant-ph/0207001*.
- [36] O. Golubitsky and D. Maslov, "A study of optimal 4-bit reversible Toffoli circuits and their synthesis," **IEEE Transactions on Computers**, vol. 61, no. 9, pp. 1341–1353, 2012, *arXiv:1103.2686*.
- [37] W. N. N. Hung, X. Wang, R. Peng, and X. Song, "Optimal synthesis of multiple output Boolean functions using a set of quantum gates by symbolic reachability analysis," **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, vol. 25, no. 9, pp. 1652–1663, 2006.
- [38] D. Maslov and D. M. Miller, "Comparison of the cost metrics for reversible and quantum logic synthesis," **IET Computers & Digital Techniques**, vol. 1, no. 2, pp. 98–104, 2007, *arXiv:quant-ph/0511008*.
- [39] V. V. Shende, A. K. Prasad, I. L. Markov, and J. P. Hayes, "Synthesis of reversible logic circuits," **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, vol. 22, no. 6, pp. 710–722, 2003, *arXiv:quant-ph/0207001*.
- [40] C. Dawson and M. Nielsen, "The Solovay-Kitaev algorithm," **Quantum Information and Computation**, vol. 6, pp. 81–95, 2006, *arXiv:quant-ph/0505030*.
- [41] M. Nielsen and I. L. Chuang, **Quantum Computation and Quantum Information**. Cambridge University Press, 2000.
- [42] A. Bocharov and K. M. Svore, "A Depth-Optimal Canonical Form for Single-qubit Quantum Circuits," *arXiv:1206.3223*, 2012.
- [43] V. Kliuchnikov, D. Maslov, and M. Mosca, "Fast and efficient

exact synthesis of single qubit unitaries generated by Clifford and T gates," *arXiv:1206.5236*, 2012.

[44] A. Fowler, "Constructing arbitrary Steane code single logical qubit fault-tolerant gates," **Quantum Information and Computation**, vol. 11, pp. 867–873, 2011, *arXiv:quant-ph/0411206*.

[45] P. Aliferis, D. Gottesman, and J. Preskill, "Quantum accuracy threshold for concatenated distance-3 codes," **Quantum Information and Computation**, vol. 6, pp. 97–165, 2006, *arXiv:quant-ph/0504218*.

[46] X. Zhou, D. W. Leung, and I. L. Chuang, "Methodology for quantum logic gate constructions," **Phys. Rev. A**, vol. 62, 052316, 2000, *arXiv:quant-ph/0002039*.

[47] A. G. Fowler, A. M. Stephens, and P. Groszkowski, "High threshold universal quantum computation on the surface code," **Phys. Rev. A**, vol. 80, 052312, 2009, *arXiv:0803.0272*.

[48] S. Aaronson and D. Gottesman, "Improved simulation of stabilizer circuits," **Phys. Rev. A**, vol. 70, 052328, 2004, *arXiv:quant-ph/0406196*.

[49] S. Sen and R. E. Tarjan, "Deletion without rebalancing in balanced binary trees," **Proc. 21st ACM-SIAM Symposium on Discrete Algorithms (SODA)**, pp. 1490–1499, 2010.

[50] A. Childs et al., "Exponential algorithmic speedup by quantum walk," **Proc. 35th ACM Symposium on Theory of Computing**, pp. 59–68, 2003, *arXiv:quant-ph/0209131*.

[51] A. Peres, "Reversible logic and quantum computers," **Phys. Rev. A**, vol. 32, pp. 3266–3276, 1985.

[52] R. P. Feynman, "Quantum mechanical computers," **Foundations of Physics**, vol. 16, no. 6, pp. 507–531, 1986.

[53] P. Kaye, R. Laflamme, and M. Mosca, **An Introduction to Quantum Computing**. Oxford University Press, 2007.

[54] K. N. Patel, I. L. Markov, and J. P. Hayes, "Efficient Synthesis of Linear Reversible Circuits," *arXiv:quant-ph/0302002*, 2003.

[55] Zhiqiang Li, Xiaoyu Song, Marek Perkowski, and Han-Wu Chen, "Realization of a new permutative gate library using controlled-kth-root-of-NOT quantum gates for exact minimization of quantum circuits," *International Journal of Quantum Information*, vol. 12, no. 5, August 2014.

[56] A. Barenco, C. Bennett, R. Cleve, D. DiVincenzo, M. Margolus, P. Shor, T. Sleator, J. Smolin, and H. Weinfurter, "Elementary gates for quantum computation," *Physical Review A*, vol. 52, no. 5, p. 3457, 1995.

[57] D. M. Miller, R. Wille, and Z. Sasanian, "Elementary quantum gate realizations for multiple-control Toffoli gates," *Proceedings of the 41st IEEE International Symposium on Multiple-Valued Logic*, 2011, pp. 288–293.

[58] Y. Liu, G. L. Long, and Y. Sun, "Universal quantum circuit for multiqubit controlled gates," *International Journal of Quantum Information*, vol. 6, no. 3, pp. 447–457, 2008.

[59] Edison Tsai and Marek Perkowski, "Synthesis of permutative quantum circuits with Toffoli and TISC gates," *Proceedings of the IEEE 42nd International Symposium on Multiple-Valued Logic*, 2012, pp. 50–55.

[60] Z. Sasanian and D. M. Miller, "Transforming MCT Circuits to NCVW Circuits," *Workshop on Reversible Computation*, 2011, pp. 163–172.

[61] W. N. N. Hung, X. Song, G. Yang, J. Yang, and M. Perkowski, "Synthesis of reversible logic circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 9, pp. 1652–1660, 2006.

[62] G. W. Yang, W. N. N. Hung, X. Song, and M. Perkowski, "Exact synthesis of 3-qubit quantum circuits from non-binary quantum gates using multiple-valued logic and group theory," *Proceedings of the Design, Automation and Test in Europe Conference (DATE)*, 2005, pp. 434–439.

[63] Z. Li, H. Chen, W. Liu, X. Xue, and F. Xiao, "Reversible logic circuit synthesis based on quantum cost," *Acta Electronica Sinica*, vol. 41, no. 4, pp. 690–695, 2013.

[64] D. Maslov and D. M. Miller, "Multiple-valued multiple-controlled Toffoli gates," *IET Computers & Digital Techniques*, vol. 1, no. 2, pp. 98–106, 2007.

[65] G. W. Yang, X. Song, M. Perkowski, W. N. N. Hung, J. Biamonte, and Z. Tang, "Reversible logic circuit synthesis using genetic algorithm," *IET Computers & Digital Techniques*, vol. 1, no. 4, pp. 382–391, 2007.

[66] T. Kalajdziewski and J. M. Arrazola, "Exact gate decompositions for photonic quantum computing," *arXiv:1811.10651*, 2018.

Available at: <https://arxiv.org/pdf/1811.10651.pdf>

[67] Creation and annihilation operators. Available at: https://en.wikipedia.org/wiki/Creation_and_annihilation_operators

[68] W. Magnus, "On the exponential solution of differential equations for a linear operator," *Communications on Pure and Applied Mathematics*, vol. 7, pp. 649–673, 1954.

[69] M. Suzuki, "Generalized Trotter's formula and systematic approximants of exponential operators," *Communications in Mathematical Physics*, vol. 51, pp. 183–190, 1976.

[70] S. Sefi and P. van Loock, "Maximally entangled states from Gaussian squeezing and displacement," *Physical Review Letters*, vol. 107, no. 17, 170501, 2011.

[71] A. M. Childs, D. Maslov, Y. Nam, N. J. Ross, and Y. Su, *arXiv:1711.10980*, 2017.

[72] C. Sparrow, E. Martín-López, N. Maraviglia, A. Neville, C. Harrold, J. Carolan, Y. N. Joglekar, T. Hashimoto, N. Matsuda, J. L. O'Brien, et al., "Simulating the vibrational quantum dynamics of molecules using a photonic quantum processor," *Nature*, vol. 557, pp. 660–666, 2018.

[73] H.-K. Lau, R. Pooser, G. Siopsis, and C. Weedbrook, "Quantum Chirp Parameter Estimation with Gaussian States," *Physical Review Letters*, vol. 118, no. 8, 080501, 2017.

[74] J. M. Arrazola, T. Kalajdziewski, C. Weedbrook, and S. Lloyd, *arXiv:1809.02622*, 2018.

[75] T. Kalajdziewski, C. Weedbrook, and P. Rebentrost, "Quantum computation with continuous-variable cluster states," *Physical Review A*, vol. 97, 062311, 2018.

[76] T. Sowiński, O. Dutta, P. Hauke, L. Tagliacozzo, and M. Lewenstein, "Dipolar Molecules in Optical Lattices: Quantum Simulating Rotational Degrees of Freedom," *Physical Review Letters*, vol. 108, 115301, 2012.

[77] P. Rebentrost, B. Gupt, and T. R. Bromley, *arXiv:1809.02579*, 2018.