

Nontrivial Multiproduct Commutation Relation Toward Reducing T -Count in Sequential Pauli-Based Computation

Yusei Mori^{1,*}, Hideaki Hakoshima^{1,2,†} and Keisuke Fujii^{1,2,3,‡}

¹Graduate School of Engineering Science, *The University of Osaka*, 1-3 Machikaneyama, Toyonaka, Osaka 560-8531, Japan

²Center for Quantum Information and Quantum Biology, *The University of Osaka*, 1-2 Machikaneyama, Toyonaka, Osaka 560-0043, Japan

³RIKEN Center for Quantum Computing, 2-1 Hirosawa, Wako, Saitama 351-0198, Japan



(Received 15 October 2025; revised 19 March 2026; accepted 22 April 2026; published 1 June 2026)

Quantum compilers that reduce the number of T gates are essential for minimizing the overhead of fault-tolerant quantum computation. Achieving further T -count reduction calls for identifying equivalent circuit transformation rules beyond those utilized in existing tools. In this paper, we rewrite any given Clifford + T circuit using a Clifford block followed by a sequential Pauli-based computation and introduce a nontrivial, ancilla-free transformation rule, the multiproduct commutation relation (MCR). MCR constructs gate sequences based on specific commutation properties among multi-Pauli operators, yielding seemingly noncommutative instances that can be commuted, thereby enabling gate orderings that cannot be derived from pairwise commutation alone. We also propose the MCR Compiler, which incorporates MCR-based transformations as an optimization pass. To evaluate its effect, we use a benchmark circuit dataset generated through quantum circuit unoptimization. This approach intentionally adds redundancy to the circuit while keeping its equivalence, allowing a quantitative evaluation of compiler performance by comparison with the original circuit. Our numerical experiments reveal that the MCR Compiler achieves further T -count reduction beyond current compilers, establishing MCR-based transformations as a practical optimization primitive. These results highlight an untapped opportunity to enhance the optimization capabilities of quantum compilers.

DOI: [10.1103/qw5z-3bwz](https://doi.org/10.1103/qw5z-3bwz)

I. INTRODUCTION

Quantum computing has the potential to outperform classical computation in specific tasks such as factorization [1], unstructured database search [2], and simulation of quantum many-body systems [3,4]. While these algorithms promise significant theoretical speedups, their practical implementation requires expressing them as quantum circuits that meet the constraints of physical quantum hardware. The ongoing increase in the number of physical qubits and the improvement of gate fidelity

bring the construction of fault-tolerant quantum computers (FTQC) closer to practicality [5,6]. In this emerging landscape, addressing resource constraints becomes increasingly essential, particularly the high implementation cost of specific gates in FTQC.

In FTQC architectures, the Clifford + T gate set stands out as a leading candidate for universal quantum computation [7] due to its compatibility with quantum error correction. Clifford gates are commonly implemented using transversal operations, which are relatively low-cost. However, non-Clifford gates such as the T gate are substantially more expensive to implement fault-tolerantly, primarily due to the need for magic state distillation [8]. Therefore, reducing the number of T gates in quantum circuits, a process known as *T -count optimization*, has emerged as a key challenge for the efficient implementation of quantum algorithms on FTQC.

To enable efficient T -count reduction, quantum compilers form an essential component of the quantum computing stack, serving as an intermediate layer between high-level descriptions of quantum algorithms

*Contact author: u839526i@ecs.osaka-u.ac.jp

†Contact author: hakoshima.hideaki.es@osaka-u.ac.jp

‡Contact author: fujii.keisuke.es@osaka-u.ac.jp

Published by the American Physical Society under the terms of the [Creative Commons Attribution 4.0 International](https://creativecommons.org/licenses/by/4.0/) license. Further distribution of this work must maintain attribution to the author(s) and the published article's title, journal citation, and DOI.

and low-level, hardware-specific implementations [9–11]. This design is derived from the role of classical compilers, which translate high-level programming languages into machine-level instructions [12]. Quantum compilers perform not only gate count reduction in quantum circuits but also a range of tasks, including gate decomposition [13–15] and qubit mapping for hardware compatibility [16–19].

From the perspective of circuit optimization, a common strategy is to rewrite a quantum circuit into an intermediate representation (IR), apply optimization rules specialized for that representation, and translate it back into a quantum circuit. Several IRs have been proposed for T -count reduction. The first is the *ZX-calculus*, a graphical framework which expresses quantum circuits as diagrams and simplifies them using rewrite rules such as spider fusion [20,21] and phase gadgetization [22]. The latter technique enables specific nonlocal optimizations when the involved gadgets can be propagated through the *ZX*-diagram. Nonetheless, the effectiveness of these techniques remains limited due to the presence of noncommuting gates within the circuit, such as internal Hadamard gates, which restrict simplification to local transformations. Another well-known approach is the *phase-polynomial*, an algebraic representation which encodes CNOT + T circuits as linear Boolean functions and enables algorithmic optimization methods [23–25]. Extensions of this technique to Clifford + T circuits have been proposed, a process known as *Hadamard gadgetization* [26–28]. Though this approach enables representing the entire circuit within the CNOT + T framework, it inevitably introduces additional ancilla qubits, typically requiring one ancilla qubit per Hadamard gate. In the context of FTQC, these ancilla qubits must be implemented as fault-tolerant logical qubits, further increasing the resource requirements.

As an approach that can potentially overcome the limitations of the methods mentioned above, the *sequential Pauli-based Computation (sequential PBC)* expresses a Clifford + T circuit as a sequence of $\pm\pi/4$ multi-Pauli rotation gates obtained by rearranging all Clifford gates to the beginning of the circuit while keeping the overall equivalence [29]. Notably, originally proposed as a framework for performing lattice surgery in surface codes, sequential PBC has been shown to be not only effective for reducing the T -count but also well-suited for optimizations targeting the instruction sets of FTQC [30,31]. In the context of T -count reduction, sequential PBC has demonstrated performance comparable to other IRs, as exemplified by the compiler, TMerge [32,33]. However, its optimization rules are currently restricted to mutually commuting Pauli rotations and thus cannot be directly applied to noncommuting cases.

In this paper, to extend the domain of existing circuit rewrite rules, we introduce a class of nonlocal and

ancilla-free circuit transformations for quantum circuits, the *multiproduct commutation relation (MCR)*, which is defined as a sequential PBC format. MCR constructs a sequence consisting of multi-Pauli rotation gates that anti-commute with each other but commute collectively. As a result, MCR enables gate reorderings that go beyond local rewrites based solely on commutation relations between individual gates. Leveraging this transformation rule, we propose the MCR Compiler that explicitly incorporates MCR as an optimization pass for T -count reduction. To evaluate its effectiveness in comparison with existing compilers, we construct benchmark circuits that are deliberately made redundant by MCR. To this end, we employ MCR in an algorithm called *quantum circuit unoptimization* [34], which intentionally introduces redundancy to the circuit while keeping its equivalence. Using this process, we can guarantee the optimal T -count from the original quantum circuit, allowing us to quantitatively analyze the performance of compilers in optimizing redundant T gates. Using the unoptimized circuits as benchmarks, where redundant T gates are introduced by MCR, we evaluate the performance of the MCR Compiler and four types of T gate optimization compilers, specifically those based on the *ZX*-calculus, phase-polynomial, and sequential PBC representations. The schematic workflow of the quantum compiler benchmarking framework is shown in Fig. 1. Our results show that these state-of-the-art compilers, despite their different optimization frameworks, fail to reduce the T -count comparable to that of the original circuit. In contrast to existing compilers, the MCR Compiler successfully exploits MCR-based transformations and achieves substantial T -count reductions. These results confirm that MCR is not only a nontrivial equivalent transformation rule, but also functions as a practically effective optimization rule that enables more globally aware T -count reduction.

The structure of this paper is as follows. Section II provides the preliminary, introducing the Pauli and Clifford groups as well as a set of rotation axes used to define multi-Pauli rotation gates. Also, we describe a method for separating the Clifford + T circuit into a sequence only composed of Clifford gates and a sequential PBC derived from T gates. In Sec. III, we define and formulate a nontrivial gate commutation rule, MCR, which enables transformations not evident from adjacent gate commutativity alone. We also provide a condition for constructing MCR and introduce the MCR-aware optimization pass, including its algorithmic procedure. Section IV verifies the nonobviousness of MCR through compiler benchmarks based on quantum circuit unoptimization, and demonstrates the advantage of MCR-based optimization over existing compilers using the MCR Compiler developed in this work. Finally, Sec. V concludes our study and outlines potential directions for future research.

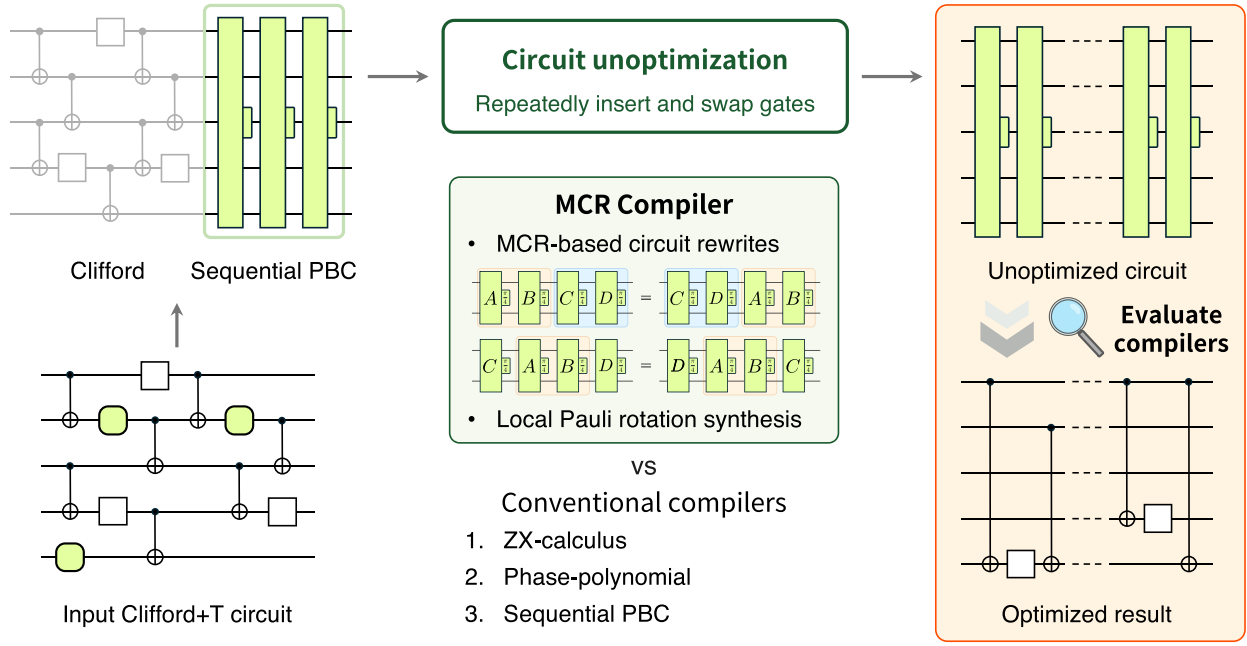


FIG. 1. An overview of the quantum compiler benchmarking procedure for Clifford + T circuits using MCR-based unoptimization. Starting from an input Clifford + T circuit, we first reorder all T gates with adjacent Clifford gates to separate the Clifford part and yield a sequential Pauli-based computation. Then, we apply the multiproduct commutation relation to the non-Clifford block, intentionally introducing redundant gates and yielding an *unoptimized* circuit that is functionally equivalent but has a larger T -count. A target quantum compiler then attempts to compress this circuit and produces an optimized output, whose T -count is compared with that of the original circuit. If the compiler fails to recover to the original optimal T -count, analyzing the gap makes it possible to improve the compiler's performance. Using this framework, we compare the T -count optimization performance of the proposed multiproduct commutation Compiler with that of conventional compilers, including those based on the ZX-calculus, phase-polynomial, and sequential Pauli-based computation representations.

II. PRELIMINARY

A. Multi-Pauli rotations

We begin with defining the n -qubit *Pauli group* $\mathcal{P}_n$ as the group generated by n -fold tensor products of the single-qubit Pauli operators $\{I, X, Y, Z\}$, multiplied by coefficients from the set $\{\pm 1, \pm i\}$,

$$\mathcal{P}_n = \{\pm 1, \pm i\} \times \{I, X, Y, Z\}^{\otimes n}. \quad (1)$$

We also use the *Clifford group* $\mathcal{C}_n$ as the set of unitary operators that map every element of $\mathcal{P}_n$ to another element of $\mathcal{P}_n$ under conjugation,

$$\mathcal{C}_n = \{C \mid CPC^\dagger \in \mathcal{P}_n, \forall P \in \mathcal{P}_n\}. \quad (2)$$

Representative elements of the Clifford group include the Hadamard (H), phase (S), and CNOT gates,

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \quad (3)$$

$$S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}, \quad (4)$$

$$\text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}. \quad (5)$$

Clifford gates alone are not *universal*, meaning that they cannot implement arbitrary unitary operations on quantum circuits [7]. To achieve universality, it is necessary to include non-Clifford gates, and a widely used example is the T gate,

$$T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}. \quad (6)$$

Both the Clifford and T gates can be expressed as products of *multi-Pauli rotation gates*. They are unitary operations of the form

$$R_P(\theta) = e^{-i\theta P/2} = \cos\left(\frac{\theta}{2}\right) I^{\otimes n} - i \sin\left(\frac{\theta}{2}\right) P, \quad (7)$$

where $\theta \in (-\pi, \pi]$ is the *rotation angle*, and P is the *rotation axis*, chosen from a set of rotation axes $\mathcal{P}_n^* \subset \mathcal{P}_n$,

defined as

$$\mathcal{P}_n^* := \{\pm 1\} \times \{I, X, Y, Z\}^{\otimes n} \setminus \{\pm I^{\otimes n}\}. \quad (8)$$

This set consists of Hermitian Pauli operators excluding the identity $\pm I^{\otimes n}$. All Clifford operators can be rewritten as sequences of multi-Pauli rotations with angles of the form $k\pi/2$ for some integer k . For example, the Hadamard gate admits the decomposition $\exp(i\pi/2)R_Z(\pi/2)R_X(\pi/2)R_Z(\pi/2)$, and similar expressions exist for S and CNOT. On the other hand, T gate is expressed as a $\pi/4$ multi-Pauli rotation gate $\exp(i\pi/8)R_Z(\pi/4)$. We sometimes denote the multi-qubit Pauli rotation gates with identity operators (I) appended as needed to ensure consistency with the total number of qubits in the circuit, e.g., $R_{XII}(\pi/4) = IR_X(\pi/4)II$.

In this paper, we consider a Clifford + T circuit that contains many multi-Pauli rotation gates. To clearly illustrate their structure, we adopt a diagrammatic representation that explicitly shows the rotation axis and angle for each gate [30]. For each qubit, the rotation axis (I, X, Y, Z) is written inside a box, and the rotation angle is placed outside. Here, I denotes the identity operator. Gates with a rotation angle of $\pm\pi/4$ (i.e., non-Clifford gates) are colored green, while those with a phase of $k\pi/2$ (i.e., Clifford gates) are colored orange. For example, T gate, $S^\dagger$ gate, and $R_{XIZ}(\pi/4)$ are visualized according to this rule as shown in Fig. 2.

Each multi-Pauli rotation gate $R_P(\theta)$ can be decomposed into a gate sequence consisting of a single-qubit Z -rotation gate and Clifford gates from the set $\{\text{CNOT}, H, S, S^\dagger\}$. This decomposition can be achieved in two steps. First, any rotation of the form $R_{Z\dots Z}(\theta)$ is reduced to a single-qubit Z -rotation acting on one of the qubits, combined with CNOT gates. Figure 3 illustrates an example of the transformation of the four-qubit gate $R_{ZZZZ}(\theta)$. Next, to handle a general Pauli axis $P \in \mathcal{P}_n^*$, we apply local Clifford gates before and after the Z -rotation to convert it into a rotation around P . For example, to change the rotation axis from Z to X , we apply Hadamard gates

$$\begin{aligned} T &= \boxed{Z}^{\pi/4} \\ S^\dagger &= \boxed{Z}^{\pi/2} \end{aligned} \quad R_{XIZ}\left(\frac{\pi}{4}\right) = \boxed{X} \boxed{I}^{\pi/4} \boxed{Z}$$

FIG. 2. Representation of quantum gates used in this paper. The rotation axis is indicated inside each box, and the rotation angle is shown outside. Gates with rotation angles of $\pm\pi/4$ (non-Clifford) are colored green, while those with $k\pi/2$ ($k \in \mathbb{Z}$, Clifford) are colored orange.

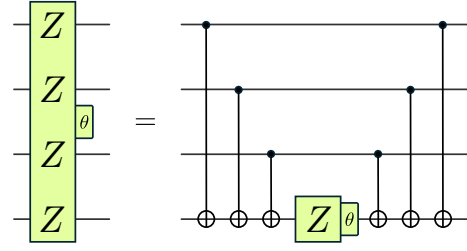


FIG. 3. Decomposition of the multi-Pauli rotation gate $R_{ZZZZ}(\theta)$. The gate applies a rotation about the tensor product of four Pauli- Z operators, and is decomposed into a standard circuit using a sequence of CNOT gates and a single-qubit Z -rotation.

before and after. To convert it into Y , we use $\{S^\dagger, H\}$ before and $\{H, S\}$ after. Using these steps, a gate such as $R_{YXZ}(\theta)$ can be equivalently transformed as shown in Fig. 4.

B. Sequential PBC

We separate a given Clifford + T circuit into Clifford gates and non-Clifford multi-Pauli rotations as introduced in Sec. II A. The order of operation between a Clifford gate C and a T gate can generally be exchanged by replacing the T gate with a modified gate $\tilde{T}$ defined as

$$\begin{aligned} \tilde{T} &= CTC^\dagger \\ &= e^{i\frac{\pi}{8}} CR_Z\left(\frac{\pi}{4}\right) C^\dagger \\ &= e^{i\frac{\pi}{8}} R_{CZC^\dagger}\left(\frac{\pi}{4}\right). \end{aligned} \quad (9)$$

By repeatedly applying this operation to adjacent Clifford gates, any T gate can be propagated to the right end of the circuit. Figure 5 illustrates an example of T propagation. Using Eq. (9), this operation yields a new rotation axis $-YXZ$ in the end.

By repeating this procedure for all T gates in a Clifford + T circuit, we can express the circuit as the product of Clifford gates followed by the *sequential Pauli-based Computation (sequential PBC)* without loss of generality [29]. That is, given a Clifford + T circuit $U = e^{i\phi} C'_0 T^{\otimes l_1} C'_1 T^{\otimes l_2} C'_2 \dots C'_{k-1} T^{\otimes l_k} C'_k$, where ϕ is a global phase, and $T^{\otimes l_i}$ ($i = 1, \dots, k$) denotes the T gates acting on

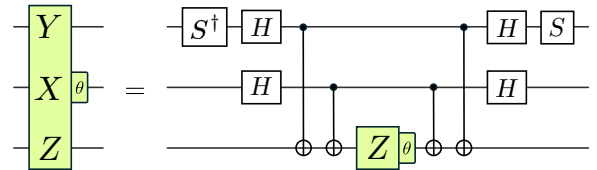


FIG. 4. Decomposition of the three-qubit rotation gate $R_{YXZ}(\theta)$. The gate implements a rotation about the tensor product of Y , X , and Z , and is decomposed into a standard circuit using local Clifford gates and a single Z -rotation.

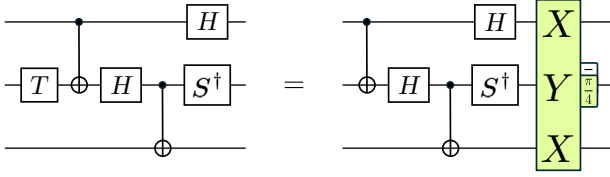


FIG. 5. Transformation of a circuit segment containing one T gate surrounded by Clifford gates. In the transformed circuit, the Clifford gates remain unchanged, and a multi-Pauli rotation gate $R_{YYX}(-\pi/4)$ is appended at the right end.

l_i qubits between the $(i-1)$ th and i th Clifford sequences, we can convert it into the following form:

$$U = e^{i\phi} \left\{ \prod_{i=1}^k R_{P_i} \left(\frac{\pi}{4} \right) \right\} C_k. \quad (10)$$

In this paper, we focus primarily on the non-Clifford part, as it determines the T -count of the circuit. Note that the above procedure can also be applied to Clifford + R_Z circuits, i.e., circuits that consist of Clifford gates and Z -rotation gates with arbitrary angles, without loss of generality.

III. THEORY OF MULTIPRODUCT COMMUTATION RELATION

In this section, we present the multiproduct commutation relation (MCR), a nontrivial commutation rule that enables certain groups of noncommuting multi-Pauli rotation gates to be reordered. We also establish the necessary and sufficient conditions for MCR construction and show how MCR enables circuit simplification beyond the reach of local optimization rules. Then, we propose a compiler explicitly including optimization passes derived from MCR.

A. Definition of MCR

We define each segment of sequential PBC, separated by dotted lines in Fig. 6, as a T layer [30].

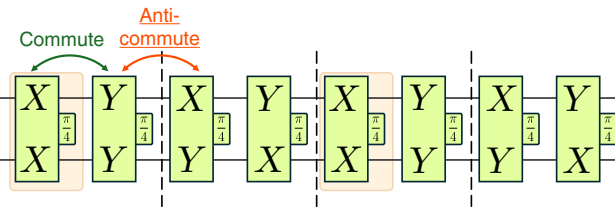


FIG. 6. A circuit containing adjacent Pauli rotation gates, some of which commute. However, due to intervening anticommute gates, no further gate merging for circuit optimization is possible.

Definition 1 (T layer). In sequential PBC, a segment is called a T layer if

- (i) All multi-Pauli rotation gates within the layer mutually commute.
- (ii) For any multi-Pauli rotation around axis P in the $(i+1)$ th layer, there exists at least one multi-Pauli rotation around axis Q in the i th layer such that Q anticommutes with P .

If the rotation axes of non-Clifford gates match exactly in succession, their rotation angles can be combined into a single gate, resulting in T -count reduction by two, i.e., $R_P(\pm\pi/4)R_P(\pm\pi/4) = R_P(k\pi/2) \in \mathcal{C}_n$. Such merging optimization applies even to nonadjacent gates, provided they belong to the same T layer. T -count optimization compilers exploiting this gate synthesis rule in common T layers have been proposed [30,32].

However, some circuits remain unsimplifiable under this approach. For example, the circuit in Fig. 6 contains four T layers. Within each layer, gates such as $R_{XX}(\pi/4)$ and $R_{YY}(\pi/4)$ or $R_{XY}(\pi/4)$ and $R_{YX}(\pi/4)$ commute with each other. In contrast, gates across adjacent layers such as $R_{YY}(\pi/4)$ and $R_{XY}(\pi/4)$ anticommute, making them unswappable and preventing further optimization. Applying MCR overcomes this limitation, i.e., certain gates can be swapped as composite units while preserving the overall circuit functionality. In the remainder of this section, we formalize this commutation rule in terms of multi-Pauli rotation axes and demonstrate its use in enabling nontrivial gate reordering.

We define the *multiproduct commutation relation* (MCR) as a specific combination of multi-Pauli rotation gates whose axes satisfy certain commutation and anticommution conditions.

Definition 2 (Multiproduct commutation relation). Let $A, B, C, D \in \mathcal{P}_n^*$ be mutually distinct (up to a sign ± 1) multi-Pauli axes corresponding to the rotation gates with an angle of $\pi/4$. We say that the pairs (R_A, R_B) and (R_C, R_D) satisfy the multiproduct commutation relation (MCR) if the following three conditions hold:

- (1) $[A, B] = [C, D] = 0$,
- (2) $\{A, C\} = \{A, D\} = \{B, C\} = \{B, D\} = 0$,
- (3) $[A + B, C + D] = 0$.

Here, $[M_1, M_2] := M_1M_2 - M_2M_1$ denotes the commutator and $\{M_1, M_2\} := M_1M_2 + M_2M_1$ denotes the anticommutator.

Theorem 1. Any sequential PBC whose axes satisfy the MCR can be entirely swapped. That is, for (R_A, R_B) and (R_C, R_D) satisfying Definition 2, Eq. (11) holds as shown

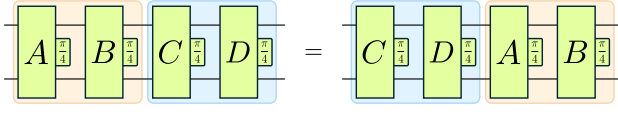


FIG. 7. Circuit diagram illustrating the gate sequence satisfying multiproduct commutation relation.

in Fig. 7,

$$\begin{aligned} R_D\left(\frac{\pi}{4}\right) R_C\left(\frac{\pi}{4}\right) R_B\left(\frac{\pi}{4}\right) R_A\left(\frac{\pi}{4}\right) \\ = R_B\left(\frac{\pi}{4}\right) R_A\left(\frac{\pi}{4}\right) R_D\left(\frac{\pi}{4}\right) R_C\left(\frac{\pi}{4}\right). \end{aligned} \quad (11)$$

Proof. From conditions 1 and 3 of Definition 2, we have

$$\begin{aligned} R_D\left(\frac{\pi}{4}\right) R_C\left(\frac{\pi}{4}\right) R_B\left(\frac{\pi}{4}\right) R_A\left(\frac{\pi}{4}\right) \\ = e^{-i\frac{\pi}{8}(D+C)} e^{-i\frac{\pi}{8}(B+A)} \\ = e^{-i\frac{\pi}{8}(B+A)} e^{-i\frac{\pi}{8}(D+C)} \\ = R_B\left(\frac{\pi}{4}\right) R_A\left(\frac{\pi}{4}\right) R_D\left(\frac{\pi}{4}\right) R_C\left(\frac{\pi}{4}\right). \end{aligned} \quad (12)$$

■

Note that condition 2 of Definition 2 is not explicitly used in the above derivation. Instead, it serves to exclude the trivial case where all pairs among R_A, R_B, R_C , and R_D commute. In this case, the gates can be freely reordered, and the commutation relation becomes obvious. Condition 2 ensures that the MCR captures only nontrivial gate reorderings that cannot be justified by pairwise commutation alone, which follows from the commutation and anticommutation relations of the underlying Pauli operators.

Consider the circuit shown in Fig. 8, where Theorem 1 enables a nontrivial gate reordering that leads to further optimization. The circuit remains equivalent even if the pairs (R_{XY}, R_{YX}) and (R_{XX}, R_{YY}) are swapped, since these four gates satisfy all conditions in Definition 2. After performing this swap, the sequence of rotation axes becomes $XX, YY, XX, YY, XY, YX, XY, YX$. Since $[XX, YY] = 0$ and $[XY, YX] = 0$, the respective pairs are pairwise commutable. This observation allows each adjacent pair to be merged into a single Clifford gate, resulting in the simplified sequence: $R_{XX}(\pi/2), R_{YY}(\pi/2), R_{XY}(\pi/2), R_{YX}(\pi/2)$. This example illustrates how MCR enables circuit simplification beyond local rewrite rules by capturing a broader class of gate rearrangement of multi-Pauli rotations.

A natural question is whether the gate sequence satisfying MCR arises in quantum circuits for practically relevant algorithms. In fact, such structures appear ubiquitously in Trotterized quantum circuits for simulating quantum

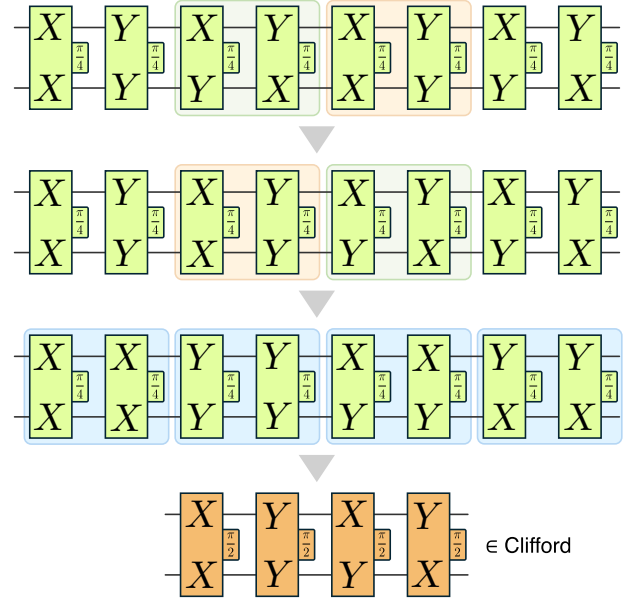


FIG. 8. Optimization of the circuit shown in Fig. 6 using multiproduct commutation relation condition. The circuit remains equivalent under the swap of commuting pairs, $(R_{XX}(\pi/4), R_{YY}(\pi/4))$ and $(R_{XY}(\pi/4), R_{YX}(\pi/4))$. After rearrangement, the circuit is partitioned into two T layers consisting of four gates each. Within each T layer, gates with shared rotation axes can be merged into Clifford gates. As a result, the entire circuit is reduced to a form that can be expressed using only Clifford operations.

many-body spin dynamics, a class of algorithms that has been explored as a benchmark task demonstrating quantum advantage [35–39]. As a representative example, we focus on the kicked spin dynamics of the XY model. This system is described by the Hamiltonian

$$H = J \sum_{\langle i,j \rangle} (X_i X_j + Y_i Y_j) + h_z \sum_i Z_i. \quad (13)$$

Here, $\langle i,j \rangle$ denotes nearest-neighbor pairs, J is the strength of interaction, and h_z represents the strength of the longitudinal magnetic field [40]. In digital quantum simulation, the time evolution governed by this Hamiltonian is implemented via Trotter decomposition, resulting in a quantum circuit with alternating layers of two-qubit rotations, consisting of R_{XX} and R_{YY} , and layers of single-qubit Z -rotations R_Z . In this setting, the pairs (R_{XX}, R_{YY}) and (R_{ZI}, R_{IZ}) satisfy the conditions in Definition 2. Consequently, these gates can be rearranged such that all Z terms are grouped together and subsequently merged. Based on this model, we benchmark the performance of our proposed compiler incorporating MCR-based circuit transformations (described in Sec. III C) and compare it with conventional optimization frameworks in Sec. IV C. The rearrangement of Z -rotations follows directly from

particle-number conservation in this model. More generally, even for complex many-body Pauli rotations, appropriate Clifford transformations can induce local structures that admit similar reorderings, which are systematically captured by MCR.

B. Systematic design of sequential PBC satisfying MCR

We now explain a method that enables efficient construction of sequential PBC satisfying MCR. The following theorem provides the requirements under which four distinct rotation axes give rise to an MCR construction.

Theorem 2 (Construction of rotation axes satisfying MCR). Let $A, B, C, D \in \mathcal{P}_n^*$, and let all rotation angles be $\pi/4$. Here, A, B, C, D are assumed to have mutually distinct rotation axes, up to a sign ± 1 . Suppose A, B, C satisfy $[A, B] = 0$ and $\{A, C\} = \{B, C\} = 0$. Then, the following (I) and (II) are equivalent.

(I) A, B, C , and D satisfy all of the following (i)–(iii):

- (i) $[C, D] = 0$,
- (ii) $\{A, D\} = \{B, D\} = 0$,
- (iii) $[A + B, C + D] = 0$.

(II) $D = -ABC$.

Proof. See Appendix A. ■

Theorem 2 provides a constructive method for generating rotation axes that satisfy MCR. For instance, in the circuit from Fig. 6, we can set $A = XY$, $B = YX$, and $C = XX$. Then, $-ABC = YY$, confirming that the rotation axis of the fourth gate satisfies $D = -ABC$. While this example involves two-qubit multi-Pauli rotation gates, we can apply MCR to n -qubit multi-Pauli rotation gates as well. In this case, the total number of combinations of rotation axes that satisfy MCR grows exponentially with n . For details, see Appendix B.

C. MCR Compiler

We develop the MCR Compiler, which explicitly incorporates two fundamental MCR-based circuit rewrites, StandardMCR and EndpointMCR, into the optimization pipeline. The pseudocode of these passes is shown in Algorithm 1. Before applying these rewrites, the compiler partitions a sequential PBC into T layers, as defined in Definition 1. This preprocessing step is denoted as TLayerPartition in Algorithm 2.

StandardMCR operates on two consecutive T layers L_{left} and L_{right} . It selects a pair of distinct rotations $R_A(\pi/4), R_B(\pi/4) \in L_{\text{left}}$, and scans L_{right} for a rotation $R_C(\pi/4)$ whose axis anticommutes with both A and B . For each candidate $R_C(\pi/4)$, the algorithm checks the

ALGORITHM 1. MCR-based optimization passes.

```

1: function StandardMCR( $L_{\text{left}}, L_{\text{right}}$ )
2:   for all distinct  $R_A(\pi/4), R_B(\pi/4) \in L_{\text{left}}$  do
3:     for all  $R_C(\pi/4) \in L_{\text{right}}$  do
4:       if  $\{A, C\} = \{B, C\} = 0$  then
5:         for all  $R_D(\pi/4) \in L_{\text{right}}$  do
6:           if  $D = -ABC$  then
7:              $L_{\text{left}}.\text{remove}([R_A(\pi/4), R_B(\pi/4)])$ 
8:              $L_{\text{left}}.\text{append}([R_C(\pi/4), R_D(\pi/4)])$ 
9:              $L_{\text{right}}.\text{remove}([R_C(\pi/4), R_D(\pi/4)])$ 
10:             $L_{\text{right}}.\text{prepend}([R_A(\pi/4), R_B(\pi/4)])$ 
11:            return ( $L_{\text{left}}, L_{\text{right}}$ )
12:           else if  $D = ABC$  then
13:              $L_{\text{left}}.\text{remove}([R_A(\pi/4), R_B(\pi/4)])$ 
14:              $L_{\text{left}}.\text{append}([R_A(\pi/2), R_C(\pi/4), R_D(\pi/4)])$ 
15:              $L_{\text{right}}.\text{remove}([R_C(\pi/4), R_D(\pi/4)])$ 
16:              $L_{\text{right}}.\text{prepend}([R_A(-\pi/4), R_B(\pi/4)])$ 
17:             return ( $L_{\text{left}}, L_{\text{right}}$ )
18:           end if
19:         end for
20:       end if
21:     end for
22:   end for
23:   return ( $L_{\text{left}}, L_{\text{right}}$ )
24: end function
25:
26: function EndpointMCR( $L_{\text{left}}, L_{\text{middle}}, L_{\text{right}}$ )
27:   if  $|L_{\text{middle}}| \neq 2$  then
28:     return ( $L_{\text{left}}, L_{\text{middle}}, L_{\text{right}}$ )
29:   end if
30:   ( $R_A(\pi/4), R_B(\pi/4)$ )  $\leftarrow$  the two rotations in  $L_{\text{middle}}$ 
31:   for all  $R_C(\pi/4) \in L_{\text{left}}$  do
32:     if  $\{A, C\} = \{B, C\} = 0$  then
33:       for all  $R_D(\pi/4) \in L_{\text{right}}$  do
34:         if  $D = ABC$  then
35:            $L_{\text{left}}.\text{remove}(R_C(\pi/4))$ 
36:            $L_{\text{left}}.\text{append}(R_D(\pi/4))$ 
37:            $L_{\text{right}}.\text{remove}(R_D(\pi/4))$ 
38:            $L_{\text{right}}.\text{prepend}(R_C(\pi/4))$ 
39:           return ( $L_{\text{left}}, L_{\text{middle}}, L_{\text{right}}$ )
40:         else if  $D = -ABC$  then
41:            $L_{\text{left}}.\text{remove}(R_C(\pi/4))$ 
42:            $L_{\text{left}}.\text{append}([R_C(\pi/2), R_D(\pi/4)])$ 
43:            $L_{\text{right}}.\text{remove}(R_D(\pi/4))$ 
44:            $L_{\text{right}}.\text{prepend}(R_C(-\pi/4))$ 
45:           return ( $L_{\text{left}}, L_{\text{middle}}, L_{\text{right}}$ )
46:         end if
47:       end for
48:     end if
49:   end for
50:   return ( $L_{\text{left}}, L_{\text{middle}}, L_{\text{right}}$ )
51: end function

```

presence of a rotation $R_D(\pi/4) \in L_{\text{right}}$ with axis $D = \pm ABC$. If a rotation with $D = -ABC$ is found, the pairs $(R_A(\pi/4), R_B(\pi/4))$ and $(R_C(\pi/4), R_D(\pi/4))$ are swapped as depicted in Fig. 7. If instead $D = ABC$, the rotation $R_A(\pi/4)$ is decomposed into $R_A(\pi/2)$ followed by $R_A(-\pi/4)$, after which the MCR-based swapping is applied.

EndpointMCR operates on three consecutive T layers L_{left} , L_{middle} , and L_{right} , where L_{middle} contains exactly two rotations $R_A(\pi/4)$ and $R_B(\pi/4)$. This pass searches for a rotation $R_C(\pi/4) \in L_{\text{left}}$ whose rotation axis anticommutes with both A and B . Then, it checks whether there exists a $R_D(\pi/4) \in L_{\text{right}}$ with $D = \pm ABC$. If a rotation

ALGORITHM 2. MCR Compiler.

```

1: Input:
2:    $S_0$      $n$ -qubit sequential PBC
3:    $N_{\text{iter}}$  Optimization iteration count
4:    $C_{\text{out}} \leftarrow$  empty circuit
5:    $S \leftarrow S_0$ 
6:   for  $t = 0$  to  $N_{\text{iter}} - 1$  do
7:      $[L_0, L_1, \dots, L_{m-1}] \leftarrow \text{TLayerPartition}(S)$ 
8:      $x \leftarrow$  choose from  $\{\text{StandardMCR}, \text{EndpointMCR}\}$ 
9:     if  $x = \text{StandardMCR}$  then
10:      for each consecutive pairs  $(L_j, L_{j+1})$  do
11:         $(L_j, L_{j+1}) \leftarrow \text{StandardMCR}(L_j, L_{j+1})$ 
12:      end for
13:     else
14:      for each consecutive triples  $(L_j, L_{j+1}, L_{j+2})$  do
15:         $(L_j, L_{j+1}, L_{j+2}) \leftarrow \text{EndpointMCR}(L_j, L_{j+1}, L_{j+2})$ 
16:      end for
17:     end if
18:      $C, S \leftarrow \text{MergeAndExtract}([L_0, \dots, L_{m-1}])$ 
19:      $C_{\text{out}}.\text{append}(C)$ 
20:   end for
21: Output:
22:    $C_{\text{out}}$  Extracted Clifford circuit
23:    $S$  Optimized sequential PBC

```

with $D = ABC$ is found, the position of $R_C(\pi/4)$ and $R_D(\pi/4)$ can be exchanged directly as illustrated in Fig. 9. If instead $D = -ABC$, the rotation $R_C(\pi/4)$ is decomposed into $R_C(\pi/2)$ followed by $R_C(-\pi/4)$, after which the same MCR-based swapping is applied. Note that the

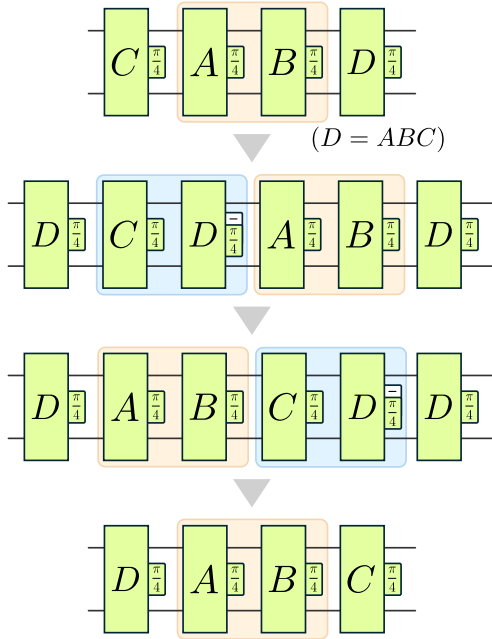


FIG. 9. EndpointMCR for $D = ABC$. By introducing the sign-flipped rotation $R_D(-\pi/4)$ and then applying multiproduct commutation relation, the resulting locally synthesized circuit becomes equivalent to one in which the positions of $R_C(\pi/4)$ and $R_D(\pi/4)$ are exchanged.

sign condition for introducing an additional rotation with an angle of $\pi/2$ is reversed compared to StandardMCR.

We now describe the overall workflow of the MCR Compiler, summarized in Algorithm 2. Starting from an input sequential PBC S_0 , the compiler performs a fixed number of optimization iterations N_{iter} , each consisting of an MCR-based rewrite pass followed by local rotation synthesis. At each iteration, the current circuit S is first partitioned into T layers via TLayerPartition. A rewrite pass $x \in \{\text{StandardMCR}, \text{EndpointMCR}\}$ is then selected and applied to the layers. In our implementation, the two passes are chosen alternately, starting and ending with EndpointMCR. Depending on the choice of x , the pass is applied either to consecutive layer pairs (L_j, L_{j+1}) (StandardMCR) or to consecutive layer triples (L_j, L_{j+1}, L_{j+2}) (EndpointMCR). Rather than exploring all equivalent circuit representations globally, the MCR Compiler restricts the search to a subspace defined by the partition of the circuit into T layers. Within this layered structure, StandardMCR and EndpointMCR sequentially scan candidate rewrites across neighboring layers and apply the first valid solution found, keeping the search computationally tractable.

The resulting layers are then simplified by MergeAndExtract, which merges successive rotations with identical axes and extracts Clifford gates as described in Sec. II B. The extracted Clifford component is accumulated into C_{out} , while the remaining non-Clifford part defines the updated sequential PBC S . In addition, the compiler may optionally insert local identity padding between optimization iterations. Specifically, when three multi-Pauli rotations $R_A(\pi/4)$, $R_B(\pi/4)$, and $R_C(\pi/4)$ satisfy conditions 1 and 2 of Definition 2, the adjacent pair $R_{ABC}(\pi/4)$ and $R_{ABC}(-\pi/4)$ is introduced while preserving circuit equivalence. This local expansion can create additional opportunities for StandardMCR rewrites, thereby facilitating further simplifications by MergeAndExtract.

IV. COMPILER BENCHMARK USING MCR

In this section, we benchmark the MCR Compiler and investigate whether existing compilers can exploit MCR-based transformations. We first introduce the concept of quantum circuit unoptimization, an operation that transforms a circuit into a more complex one while preserving its overall functionality. We then outline the associated benchmarking framework using the T -count as a quantitative evaluation metric. Finally, we compare the optimization performance of the MCR Compiler with that of existing compilers.

A. Quantum circuit unoptimization

We use the term *quantum circuit unoptimization* to describe the process of deliberately introducing gate

redundancies while keeping the circuit equivalent. The following definition formalizes this concept.

Definition 3 (Quantum circuit unoptimization [34]). Quantum circuit unoptimization is the process of generating a redundant n -qubit circuit V from an n -qubit circuit U , as expressed by

$$V = \mathcal{O}(U, C). \quad (14)$$

Here, C is a recipe constructed from elements of the set of basic unoptimization operations $\mathcal{S}$. $\mathcal{O}$ is a function (Unoptimizer) that transforms U into V based on the information in C . We assume that both U and V are $\text{poly}(n)$ gates, and the operation generating V from U can be efficiently executed on a classical computer. If the conditions for unoptimization are satisfied, V is equivalent to U up to a global phase ϕ , i.e.,

$$V^\dagger U = e^{i\phi} I, \quad (15)$$

where I represents the n -qubit identity operator.

Figure 10 illustrates a diagram of quantum circuit unoptimization. Given an input quantum circuit U , a more complex quantum circuit V is generated by applying a recipe that describes how to introduce redundancy. To define such a recipe, it is necessary to determine the set of allowed basic unoptimization operations $\mathcal{S}$. Note that only circuit operations that preserve the equivalence of the quantum

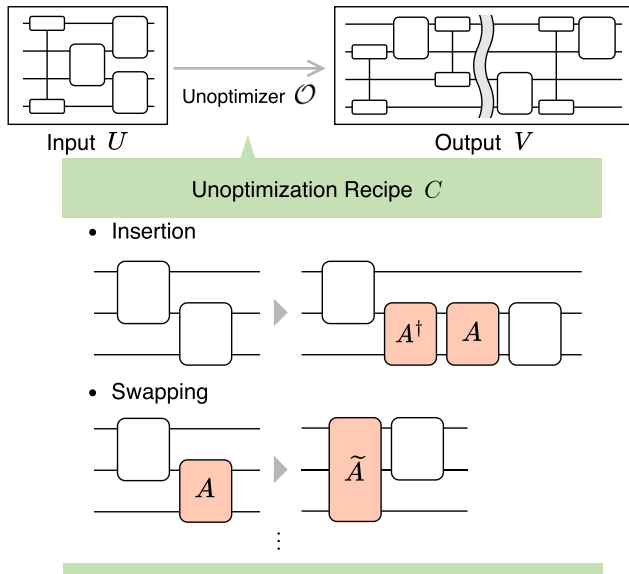


FIG. 10. Schematic illustration of quantum circuit unoptimization. The basic operations consist of gate insertion and gate swapping, which are used to construct an unoptimization recipe. Applying this recipe to an input circuit U produces a redundant circuit V that is equivalent to U up to a global phase.

circuit before and after the operation can be included in $\mathcal{S}$. The complexity of the unoptimized circuit depends critically on the choice of $\mathcal{S}$. For example, if we take $\mathcal{S} = \{\text{Insert two } X \text{ gates at random positions in the circuit}\}$, this satisfies the condition that the unoptimization operations preserve circuit equivalence. However, such redundancy is trivial: any compiler that recognizes $X^2 = I$ will immediately remove the inserted gates.

To generate more meaningful redundancy, we therefore set $\mathcal{S} = \{\text{Gate insertion, Gate swapping}\}$ and construct these operations based on MCR. Specifically, redundancies are introduced by inserting pairs of multi-Pauli rotations whose angles cancel each other along the same rotation axis, as well as by applying gate swapping as shown in Fig. 7. By combining these operations, we systematically generate circuits that are equivalent to the original circuit while having a larger T -count, which are used as benchmark circuits in this work. The detailed construction procedure of these unoptimized circuits is described in Appendix C.

B. Numerical experiments

We present numerical experiments evaluating how well existing compilers can optimize circuits containing MCR-based gate structures. We focus on four T gate optimization compilers: Pytket, PyZX, TMerge, and FastTODD. To assess compiler performance, we use the unoptimized circuit V generated by Algorithm 4 in Appendix C as input. We explain how to evaluate compiler performance using the unoptimized circuit V in Algorithm 3. A workflow of this benchmark task is illustrated in Fig. 11. Each unoptimized circuit has a T -count denoted as t_{unopt} and is passed to a compiler for optimization, resulting in a circuit with T -count t_{opt} . We quantify the optimization performance using the T -count reduction rate p defined as

$$p = \frac{t_{\text{unopt}} - t_{\text{opt}}}{t_{\text{unopt}} - t_{\text{original}}}. \quad (16)$$

Here, t_{original} is the T -count of the input circuit before unoptimization. In this paper, we use the single multi-Pauli rotation gate $R_{ZZ\dots Z}(\pi/4)$ as the input circuit, which has

ALGORITHM 3. Compiler benchmark.

-
- 1: **Input:** U n -qubit circuit
 - 2: Obtain an unoptimized circuit V using **Algorithm 4**
 - 3: Decompose U into a Clifford+ T gate set
 - 4: Obtain T -count: t_{original}
 - 5: Decompose V into a Clifford+ T gate set
 - 6: Obtain T -count: t_{unopt}
 - 7: Apply T gate optimization compiler to V
 - 8: Obtain optimized T -count: t_{opt}
 - 9: Compute gate reduction rate: p
 - 10: $p \leftarrow (t_{\text{unopt}} - t_{\text{opt}})/(t_{\text{unopt}} - t_{\text{original}})$
 - 11: **Output:** $t_{\text{unopt}}, t_{\text{opt}}, p$
-

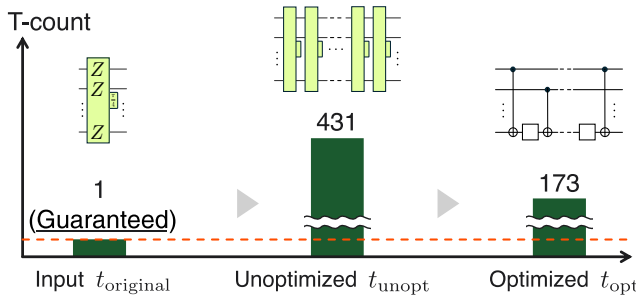


FIG. 11. An example workflow for assessing quantum compiler performance using an unoptimized benchmark circuit. The input circuit is the rotation $R_{ZZ\dots Z}(\pi/4)$, whose theoretical T -count is 1 (indicated by the red dashed line). In this illustrative case, applying Algorithm 4 in Appendix C produces an unoptimized Clifford + T circuit V with $t_{\text{unopt}} = 431$. We then decompose V into a Clifford + T gate set and pass the resulting circuit to a compiler. This yields an optimized circuit with $t_{\text{opt}} = 173$, corresponding to a 60% T -count reduction as defined in Eq. (16).

a theoretical T -count of 1, i.e., $t_{\text{original}} = 1$. A larger value of t_{unopt} indicates a higher degree of unoptimization. The greater the difference between t_{unopt} and t_{opt} , the better the performance of the compiler, which corresponds to a higher value of p . Since p is normalized by the maximum possible reduction in T -count, it always falls within the range of 0 to 1. In particular, $p = 1$ indicates perfect optimization, where the compiler completely restores the original circuit.

We use the following four state-of-the-art T -gate optimization compilers in our experiments:

- (1) **Pytket [18]**
Developed by Quantinuum, Pytket is a general-purpose compiler that supports optimization for NISQ devices. Although its primary focus is on circuit depth reduction, it also includes a T -count optimization pass. In our experiments, we use `RemoveRedundancies` to eliminate unnecessary T gates.
- (2) **PyZX [41]**
PyZX applies optimization techniques based on ZX-calculus, which represents a quantum circuit as a graphical diagram. It includes graph-theoretic optimization techniques such as local complementation and pivoting [21,22] using X -Spiders and Z -Spiders. We use `full_reduce` function for maximum simplification in the ZX-diagram, and then convert it back to a quantum circuit.
- (3) **TMerge [32]**
TMerge reduces the T -count by exploiting the commutativity of Pauli rotation axes. As described in Sec. II B, it first separates the input Clifford + T circuit into a Clifford gate sequence and a sequential PBC. This compiler can reorder them within each T

layer and merge rotation gates that have the same axis.

(4) **FastTODD [27]**

FastTODD uses phase-polynomial as an IR and optimizes the T -count by solving the symmetric tensor rank decomposition problem [42]. It removes all internal Hadamard gates by introducing ancilla qubits and applies a fast version of the Third-Order Duplicate and Destroy (TODD) algorithm [26] to minimize the T -count.

We create the benchmark circuits via quantum circuit unoptimization using the quantum circuit simulator Stim [43]. We can entirely characterize the type of a multi-Pauli rotation gate by a Pauli string specifying the rotation axis and a sign indicating whether the rotation angle is $\pi/4$ or $-\pi/4$. Stim is well-suited for dealing with such circuits, as it allows efficient updates of Pauli strings under Clifford conjugation while preserving small representations. Since the unoptimized circuit consists solely of multi-Pauli rotation gates with angles of $\pm\pi/4$, they can be exactly decomposed into Clifford + T gates. To save benchmark circuit data, we use the Qasm [44] and qc file formats [45]. To ensure that the unoptimization process preserves circuit equivalence as defined in Eq. (15), we verify correctness using the MQT-qcec tool provided by The Munich Quantum Toolkit [46]. All source code used in this work is available on GitHub [47].

Table I summarizes the averaged results. The table includes the T -count before optimization, followed by the output from each compiler: the reduced T -count and the corresponding reduction percentage. For FastTODD, an additional column shows the number of ancilla qubits n_h introduced via Hadamard gadgetization. As expected, we verify that the T -counts t_{unopt} and t_{opt} increase as the number of qubits n increases.

We first assess the performance of the MCR Compiler proposed in Sec. III C on this benchmark dataset. The MCR Compiler consistently achieves T -count reduction rates close to 100% across all values of n , effectively restoring the T -count to that of the original circuit. This demonstrates that MCR-based transformations can serve as an effective optimization primitive for T -count reduction.

In contrast, none of the four existing compilers produces reductions comparable to those obtained by the MCR Compiler at any circuit size. Among them, FastTODD consistently achieves the largest T -count reductions. For instance, at $n = 2$, it reduces the T -count by over 30%. However, its effectiveness rapidly drops as n increases, reaching well below 1% for $n \geq 6$. Moreover, FastTODD introduces a large number of ancilla qubits n_h during Hadamard gadgetization. For example, at $n = 9$, it requires over 630 ancilla qubits, increasing the total number of qubits by more than 70 times. Compilation becomes

TABLE I. Comparison of T gate optimization results across five compilers. T -counts before (t_{unopt}) and after (t_{opt}) optimization are shown for quantum circuits with n qubits. The percentage reduction in T -count is denoted as Red. (%), defined by Eq. (16). For **FastTODD**, n_h indicates the number of ancilla qubits introduced during Hadamard gadgetization. We report the optimization results averaged over 100 samples.

n	Pytket			PyZX		TMerge		FastTODD			MCR Compiler	
	t_{unopt}	t_{opt}	Red. (%)	t_{opt}	Red. (%)	t_{opt}	Red. (%)	t_{opt}	n_h	Red. (%)	t_{opt}	Red. (%)
2	46.78	43.44	7.28	32.68	30.77	32.68	30.77	32.56	18.72	31.03	1.00	100.00
3	106.34	104.26	1.98	94.84	10.92	94.84	10.92	93.45	57.77	12.24	1.10	99.91
4	190.34	189.74	0.32	185.50	2.56	185.50	2.56	185.20	115.80	2.71	1.00	100.00
5	298.14	297.92	0.074	295.96	0.73	295.96	0.73	295.76	189.30	0.80	1.16	99.95
6	430.38	430.28	0.023	429.74	0.15	429.74	0.15	429.66	278.50	0.17	1.20	99.95
7	586.30	586.30	0.00	586.06	0.041	586.06	0.041	586.00	381.96	0.051	1.14	99.98
8	766.12	766.12	0.00	766.12	0.00	766.12	0.00	766.12	503.12	0.00	1.80	99.90
9	969.90	969.86	0.0041	969.74	0.017	969.74	0.017	969.74	637.69	0.017	1.70	99.93

computationally expensive at this scale, with runtimes exceeding 21 h per circuit on average, despite yielding only marginal improvements in T -count. In comparison, PyZX and TMerge provide modest reductions for small circuits, but their performance similarly degrades as n grows, with both reaching reductions below the 1% level at $n = 5$. Pytket, on the other hand, consistently shows a relatively lower reduction rate compared to other compilers.

To explore why optimization partially works for small circuits but not for larger ones, we conduct an additional numerical experiment using a benchmark dataset generated by skipping step (c) in Fig. 15, i.e., omitting gate swapping based on MCR (see Appendix C for details). We report the results in Table II, revealing a complete absence of T -count reduction across four existing compilers and circuit sizes. Even FastTODD, despite the introduction of ancilla qubits, fails to achieve any improvement under this condition. This result suggests that for circuits with a small number of qubits, MCR-based gate swapping accidentally matches the rotation axes of neighboring gates, making optimization easier. Since the total number of gate sequences satisfying MCR increases exponentially with the number of qubits n (see Appendix B for details), cases where the rotation axes coincide after swapping become less likely as n increases.

C. Application for practical quantum circuits

As briefly introduced in Sec. III A, we consider quantum circuits arising from the simulation of kicked spin dynamics of the XY model, whose Hamiltonian is given in Eq. (13), to assess the practical relevance of MCR-aware optimization. In this work, all rotation angles are fixed to $\pi/4$, so that each rotation gate corresponds to one T gate. Moreover, the number of layers is always chosen to be even: for a given number of qubits n , we use n layers when n is even and $n + 1$ layers otherwise. Under this setting, MCR-based circuit rewrites allow all Z -rotations to be synthesized into Clifford operations, leaving only rotations around the XX and YY axes to contribute to the final T -count.

We benchmark these circuits on a one-dimensional chain with open boundary conditions (ranging from $n = 6$ to 24), and on a two-dimensional square lattice (ranging from $n = 4$ to 25). The optimization results are summarized in Table III, where the column Ideal corresponds to the T -count obtained when reducible Z -rotation gates are successfully merged. In the one-dimensional case, the MCR Compiler achieves the same T -count as the Ideal value, whereas no reduction is obtained using conventional compilers. For two-dimensional lattices, the MCR

TABLE II. Comparison of T gate optimization results across five compilers without MCR-swap. The same definitions for t_{unopt} , t_{opt} , and n_h apply as in Table I.

n	Pytket			PyZX		TMerge		FastTODD			MCR Compiler	
	t_{unopt}	t_{opt}	Red. (%)	t_{opt}	Red. (%)	t_{opt}	Red. (%)	t_{opt}	n_h	Red. (%)	t_{opt}	Red. (%)
2	33.00	33.00	0.00	33.00	0.00	33.00	0.00	33.00	18.96	0.00	1.00	100.00
3	73.00	73.00	0.00	73.00	0.00	73.00	0.00	73.00	47.36	0.00	1.00	100.00
4	129.00	129.00	0.00	129.00	0.00	129.00	0.00	129.00	86.96	0.00	1.00	100.00
5	201.00	201.00	0.00	201.00	0.00	201.00	0.00	201.00	138.51	0.00	1.00	100.00
6	289.00	289.00	0.00	289.00	0.00	289.00	0.00	289.00	202.30	0.00	1.00	100.00
7	393.00	393.00	0.00	393.00	0.00	393.00	0.00	393.00	277.37	0.00	1.00	100.00
8	513.00	513.00	0.00	513.00	0.00	513.00	0.00	513.00	363.76	0.00	1.00	100.00
9	649.00	649.00	0.00	649.00	0.00	649.00	0.00	649.00	461.86	0.00	1.00	100.00

TABLE III. T -count optimization performance for the kicked XY model. **Original** denotes the T -count of the original quantum circuit. **MCR Compiler** shows the T -count after optimization, in comparison with that obtained by **TMerge**. **Ideal** denotes the T -count obtained under the assumption that all single-qubit Z -rotation gates are synthesized into Clifford gates.

(a) One-dimensional chain (Open boundary condition).					(b) Two-dimensional square lattice.				
n	Original	TMerge	MCR Compiler	Ideal	n	Original	TMerge	MCR Compiler	Ideal
6	96	96	60	60	2×2	48	48	32	32
12	408	408	264	264	3×3	330	330	262	240
18	936	936	612	612	4×4	1024	1024	862	768
24	1680	1680	1104	1104	5×5	2730	2730	2132	2080

Compiler still yields substantial reductions compared to existing methods. These results indicate that MCR-based optimization is effective not only for deliberately unoptimized benchmark circuits but also for practically relevant quantum circuits derived from realistic quantum algorithms.

V. CONCLUSION AND OUTLOOK

In this paper, we propose and formalize the multiproduct commutation relation (MCR) as a nonlocal and ancilla-free circuit transformation rule for Clifford + T circuits. MCR enables equivalent rewrites involving multiple nonadjacent multi-Pauli rotation gates, capturing nontrivial commutativity that pairwise commutation rules cannot reach. Given the rotation axes of three multi-Pauli rotation gates, Theorem 2 provides a concrete condition to determine the fourth gate that satisfies MCR. Building on this characterization, we develop the MCR Compiler, which combines MCR-based circuit rewrites with local rotation merging to achieve T -count reduction. To assess the optimization capability enabled by MCR, we adopt a procedure called quantum circuit unoptimization, which introduces redundancy into Clifford + T circuits by inserting and swapping quantum gates while preserving circuit equivalence. This approach systematically generates complex circuits from simple ones with a guaranteed optimal T -count, offering a quantitative framework for evaluating how well a compiler eliminates redundant T gates.

Through numerical experiments within this framework, we demonstrate that the MCR Compiler achieves substantial T -count elimination, whereas state-of-the-art T -count optimization compilers yield marginal reductions. We also compare the optimization performance results with and without MCR-based gate swapping. The comparison shows that while MCR-based gate swapping introduces additional circuit redundancy, it can occasionally trigger effective local optimization in small regions of the circuit. However, such cases become infrequent as the number of qubits grows, due to the exponential increase in the number of valid MCR candidates.

MCR involves circuit transformations that can increase or decrease the number of T layers, thereby suggesting a

broader class of rewrite rules beyond the reach of existing compiler strategies. This insight opens opportunities for compiler design, circuit synthesis, and algorithm-based optimization. However, global exploration of equivalent circuit representations becomes computationally infeasible, since the search space induced by MCR-based rewrites grows exponentially with the system size. The MCR Compiler proposed in this work maintains scalability by restricting the search to a subspace defined by the partition of the circuit into T layers. MCR candidates are examined only across neighboring layers, allowing the optimization procedure to remain tractable even for large circuits. A natural direction for future work is therefore to extend the exploration beyond this restricted subspace while preserving scalability. Promising approaches include Monte Carlo-style strategies that probabilistically insert identity circuits to expose additional MCR opportunities, as well as divide-and-conquer methods that exploit structural patterns of the target circuits. Once developed, the performance of such a compiler can be evaluated on representative datasets, such as the *Reversible Benchmarks* [45].

Another promising direction involves generalizing MCR to Clifford + R_Z circuits with continuous rotation angles. This circuit class is widely used in quantum algorithms, including variational quantum algorithms [48,49] and Trotterized quantum simulation tasks [35]. In architectures that support transversal Clifford gates and implement Z -rotations using repeat-until-success protocols, minimizing the number of such gates can significantly improve execution efficiency [50,51]. Consequently, many quantum algorithms can benefit from an MCR-based optimization tool designed to reduce such rotations. The authors in Ref. [52] propose a method to optimize the number of parameters in nonrepeated parameterized quantum circuits. However, this excludes circuits where parameters exhibit algebraic relationships, such as the simultaneous appearance of α_1 and $-\alpha_1 + \pi/4$. In contrast, MCR admits simplifications even in these correlated cases, offering opportunities to optimize beyond the reach of current techniques. This property may provide value in structured algorithms, such as QAOA-MC [53], where repeated parameters are present.

For circuits using discrete angles, also known as Clifford + $R_Z(\pi/2^k)$ circuits, which relate closely to the Clifford hierarchy [54,55], MCR could offer similar benefits. Prior works [25,27] explore optimization in this setting using Reed-Muller code techniques and prove that this problem can be reduced to the higher-order symmetric tensor rank decomposition problem. The complexity of the symmetric tensor rank decomposition problem remains unknown. However, the related problem of determining the tensor rank of a third-order tensor, without the symmetry constraint, is NP-complete [42]. We expect MCR-based strategies to improve the compilation performance even in such a circuit structure. A practical application in this circuit setting is time evolution by probabilistic angle interpolation (TE-PAI) [56], which simulates quantum dynamics using ensembles of random circuits composed only of multi-Pauli rotations with angles $\pm\Delta$ or $\pm\pi$. Since these circuits belong to the Clifford + R_Z (discrete angle) class, a compiler that reduces redundant rotations could help efficiently implement TE-PAI on a real quantum device.

ACKNOWLEDGMENTS

This work is supported by the MEXT Quantum Leap Flagship Program (MEXT Q-LEAP) Grant No. JPMXS0120319794, JST COI-NEXT Grant No. JPMJPF 2014, JST Moonshot R&D Grant No. JPMJMS2061, and JSPS KAKENHI Grant Nos. JP25KJ1712 and JP24K16979.

DATA AVAILABILITY

The data that support the findings of this article are openly available [47].

APPENDIX A: PROOF OF THEOREM 2

We prove Theorem 2 as described in Sec. III B.

($\Leftarrow$) This can be demonstrated by substituting $D = -ABC$ into (i)–(iii) of (I) in order.

($\Rightarrow$): Combining (ii) and (iii) of (I) and the given assumptions, we obtain

$$[A + B, C + D] = 0 \quad (\text{A1})$$

$$\Rightarrow AC + AD + BC + BD = 0. \quad (\text{A2})$$

Since A, B, C , and D are mutually distinct Pauli operators, each term in Eq. (A2) is a nonidentity Pauli product. Because the set of n -qubit Pauli operators forms an orthogonal basis under the Hilbert-Schmidt inner product, their sum can be zero only if terms cancel in pairs.

Case 1 $AC + BC = 0$

Equation (A2) becomes

$$\begin{aligned} AD + BD &= 0 \\ \Rightarrow A &= -B, \end{aligned} \quad (\text{A3})$$

which implies that A and B have the same rotation axis but opposite rotation angles, contradicting the assumption that they have distinct axes.

Case 2 $AC + BC \neq 0$

We must partition the sum $-(AC + BC)$ into two Pauli operators AD and BD such that their sum equals this value. Because a sum of two distinct Pauli operators cannot be another single Pauli operator, the only possibility is that $(AD, BD) = (-AC, -BC)$ or $(AD, BD) = (-BC, -AC)$. We analyze these two cases separately.

- (a) If $(AD, BD) = (-AC, -BC)$, then $D = -C$, implying that D and C share the same rotation axis, which again contradicts the assumption.
- (b) If $(AD, BD) = (-BC, -AC)$, then $D = -ABC (= -BAC)$, which has a rotation axis distinct from all of A, B , and C , and hence satisfies the assumption.

Therefore, the only valid solution is $D = -ABC$.

APPENDIX B: ANALYSIS OF THE NUMBER OF POSSIBLE MCR PAIRS

Here, we explain the total number of combinations of gate rotation axes that satisfy MCR. MCR is a commutation rule applied to four sequential gates: two on the left and two on the right. Let the set of rotation axes be $\mathcal{P}_n^* := \{\pm 1\} \times \{I, X, Y, Z\}^{\otimes n} \setminus \{\pm I^{\otimes n}\}$ [Eq. (8)], and let the rotation axes of the gates from the left be $A, B, C, D \in \mathcal{P}_n^*$, where each gate has a different rotation axis except for the sign ± 1 . According to Algorithm 4 in Sec. C 1, the method for choosing rotation axes that satisfy MCR is as follows.

- (1) Select one element from $\mathcal{P}_n^*$ and set it as A .
- (2) Choose $B \in \mathcal{P}_n^*$ such that B commutes with A .
- (3) Choose $C \in \mathcal{P}_n^*$ such that $\{A, C\} = \{B, C\} = 0$.
- (4) Obtain D by setting $D = -ABC$.

Let us determine the number of ways to select the rotation axes according to this procedure. First, the axis A can be chosen from any nonidentity Pauli operator, giving $4^n - 1$ possible choices. The second axis B must commute with A and must not be equal to A or the identity. In general, for any given Pauli operator, half of the other operators commute with it. Therefore, the number of valid choices for B is $4^n/2 - 2$, where the subtraction accounts for excluding the identity operator and the previously chosen A . Next, C must anticommute with both A and B . Since one-fourth of all Pauli operators anticommute with both, the number of valid choices for C is 4^{n-1} . D is determined uniquely as $D = -ABC$ to satisfy MCR once A, B , and C are fixed as described in Theorem 2. Additionally, since the signs of A, B , and C can be chosen independently, we multiply by 8 to account for the possible sign combinations. However, to avoid overcounting due to permutation of the pairs (A, B)

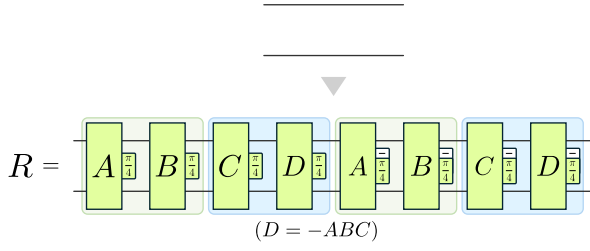


FIG. 12. Construction of a nontrivial identity operator based on the multiproduct commutation relation property, used in the insertion step of the unoptimization procedure. Each alphabet represents a rotation axis of a multi-Pauli rotation.

and (C, D) , and the exchange between these two pairs, we divide the total count by $(2!)^3 = 8$. Putting everything together, the total number of such combinations is

$$\frac{1}{8} \cdot 4^n (4^n - 1) (4^n - 4), \quad (\text{B1})$$

which shows that the number of possible axis configurations satisfying MCR scales exponentially with n .

APPENDIX C: CONSTRUCTION OF MCR-BASED UNOPTIMIZATION BENCHMARK CIRCUITS

1. MCR-based recipes for unoptimization

First, we explain *gate insertion*, which refers to the operation of inserting a gate A together with its inverse $A^\dagger$ placed adjacently in the circuit, yielding the identity operator. A typical example is the insertion of $R_P(\theta)$ followed by $R_P(-\theta)$. In this paper, we generalize this concept by employing MCR to construct more complex identity sequences, denoted R , as depicted in Fig. 12. Such sequences are systematically generated by selecting rotation axes satisfying the MCR conditions and arranging their gates and inverses, as formalized in Theorem 2. An example is shown in Fig. 13, and the generation procedure is given in Algorithm 4.

Next, we explain the operation of *gate swapping*, which refers to exchanging the positions of two adjacent gate blocks in a circuit. Here, we combine the gate

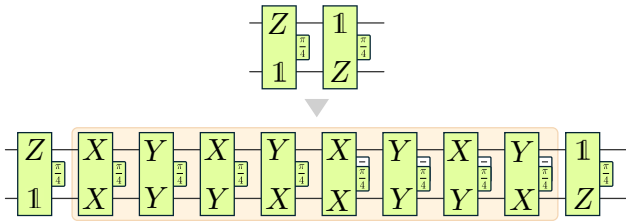


FIG. 13. A gate insertion example based on Fig. 12. Although the inserted circuit is equivalent to the identity, it cannot be simplified by considering only pairwise commutativity of adjacent gates.

ALGORITHM 4. MCR-based quantum circuit unoptimization.

```

1: Generation of identity operator using MCR
2: Select  $A \in \mathcal{P}_n^*$ 
3: Choose  $B \in \mathcal{P}_n^*$  such that  $[A, B] = 0$ 
4: Choose  $C \in \mathcal{P}_n^*$  such that  $\{A, C\} = 0$  and  $\{B, C\} = 0$ 
5: Compute  $D \leftarrow -ABC$ 
6: Construct gate sequence:
7:    $R \leftarrow R_{-D}(\pi/4)R_{-C}(\pi/4)R_{-B}(\pi/4)R_{-A}(\pi/4)$ 
8:    $R_D(\pi/4)R_C(\pi/4)R_B(\pi/4)R_A(\pi/4)$ 
9: Output:  $R$  Gate sequence such that  $R = I$ 
10:
11: Generation of unoptimized circuit
12: for  $i \leftarrow 0$  to  $n^2 - 1$  do
13:    $C[i] \leftarrow R$ , Gate Swapping
14: end for
15:  $V \leftarrow \mathcal{O}(U, C)$ 

```

swapping with an additional gate insertion that induces commutativity through MCR. This procedure is illustrated in Fig. 14. Consider the swap of the rightmost two gates in the 8-gate identity sequence constructed via MCR (as shown in Fig. 13) with the gate immediately to their right. Since MCR can only be applied to pairs of gates, it is necessary to add one more gate to complete the MCR condition. This supplementary gate can be computed using Theorem 2; in this case, $-ABC = -ZI$. Therefore, by adding $R_{ZI}(-\pi/4)$, a gate sequence satisfying MCR can be constructed. Thus, as shown in Fig. 14(b), if we insert $R_{ZI}(-\pi/4)$ and $R_{ZI}(\pi/4)$ to the right of $R_{YX}(-\pi/4)$, we can swap the two pairs of gates $(R_{XY}(-\pi/4), R_{YX}(-\pi/4))$ and $(R_{IZ}(\pi/4), R_{ZI}(-\pi/4))$

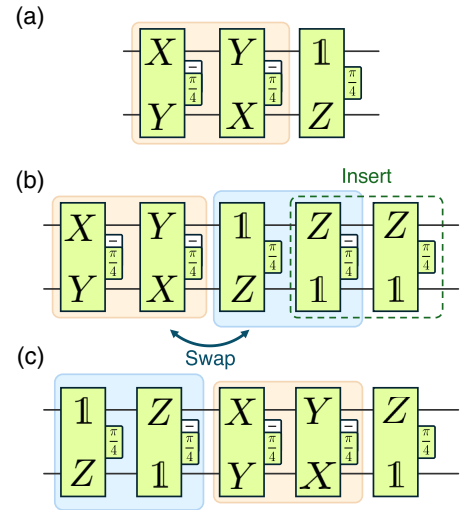


FIG. 14. Gate swapping unoptimization based on multiproduct commutation relation. (a) The circuit segment after the gate insertion. (b) Additional gate insertion of identity gates (green dashed box) to satisfy multiproduct commutation relation. (c) The resulting circuit after swapping the commuting blocks.

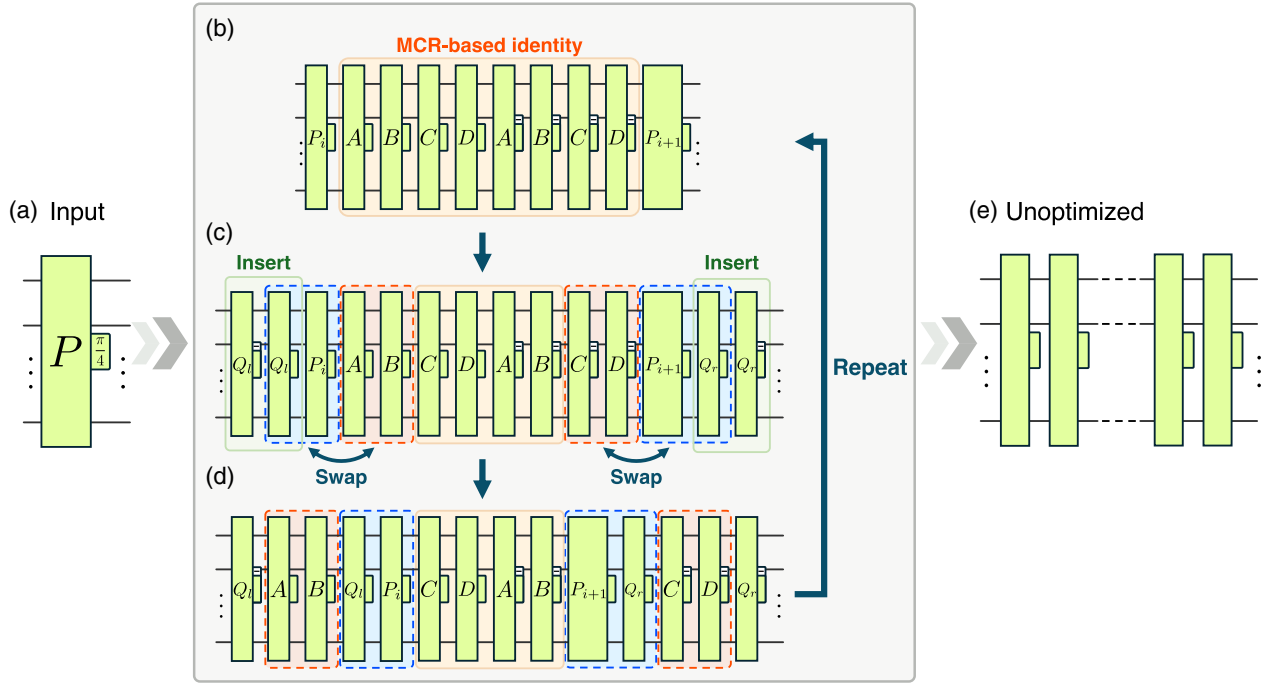


FIG. 15. Overview of multiproduct commutation relation-based quantum circuit unoptimization. (a) The input circuit, consisting of a single multi-Pauli rotation gate. (b) Insertion of an multiproduct commutation relation-based identity sequence (as constructed in Fig. 12). (c) Insertion of additional gates $R_{Q_l}^\dagger$ and R_{Q_l} at both ends, corresponding to the procedure described in Fig. 14(b). (d) The circuit after swapping the outermost gates, thereby completing the procedure in Fig. 14. (e) The unoptimized circuit obtained by repeating steps (b)–(d) n^2 times.

since they satisfy MCR. The resulting reordered sequence is shown in Fig. 14(c).

2. Procedure for benchmark circuit construction

We explain the unoptimization algorithm that combines the gate insertion and gate swapping techniques described above, along with the compiler benchmarking methodology. The overall workflow is illustrated in Fig. 15. We begin with an n -qubit circuit U consisting of a single multi-Pauli rotation gate P ($\pi/4$), which guarantees an optimal T -count of 1 and thus cannot be further reduced. Then, from the circuit, we randomly select one multi-Pauli rotation gate with axis P_i and insert an identity operator based on Algorithm 4 immediately to its right. We make the identity sequence such that the following conditions are satisfied:

$$\{P_i, A\} = \{P_i, B\} = \{C, P_{i+1}\} = \{D, P_{i+1}\} = 0. \quad (C1)$$

Next, we perform gate swapping on the two gates at both ends of the inserted gate sequence. Here, we first explain the operation of the gates in the left half. In Fig. 15(c), we insert an additional gates $R_{Q_l}^\dagger$ ($-\pi/4$) and R_{Q_l} ($\pi/4$) (highlighted in green), where $Q_l = -P_i AB$. Since the gate pairs (R_{Q_l}, R_{P_i}) (highlighted in blue) and (R_A, R_B) (highlighted in red) satisfy MCR, we can swap

these pairs. This operation results in the updated gate sequence depicted in Fig. 15(d). We perform a similar operation on the gate sequence in the right half. In this

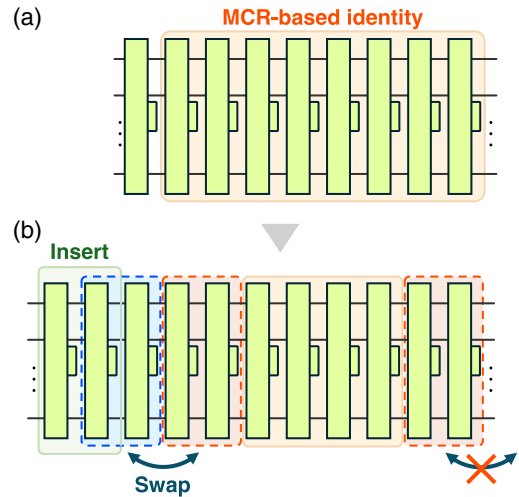


FIG. 16. Special case of gate swapping in multiproduct commutation relation-based unoptimization when the target gate is located at the rightmost end of the circuit. (a) Circuit after inserting an multiproduct commutation relation-based identity operator. (b) Swap operation applied only to the left end, as the right end cannot be operated in this configuration.

case, $Q_r = -CDP_{i+1}$, which also satisfies MCR with the adjacent pairs (R_{-C}, R_{-D}) and $(R_{P_{i+1}}, R_{Q_r})$. Note that when we select the gate with axis P_i as the right edge of the circuit, we swap the gates on only the left side as shown in Fig. 16. In this case, we do not add the additional gates $R_{Q_r}(\pi/4), R_{Q_r}(-\pi/4)$, and only the left-side gates are swapped with the inserted gate sequence.

We repeat these procedures n^2 times, where n is the number of qubits in the circuit. The final circuit obtained is the unoptimized circuit V , as shown in Fig. 15(e). The gate insertion and swapping methods enable the systematic transformation of a simple input circuit into a structurally distinct one by changing the type, position, and order of the inserted gates. In our implementation, identity sequences are randomly generated using Algorithm 4. If the resulting sequence does not satisfy the anticommutation conditions required for MCR-based gate swapping [Eq. (C1)], the generation step is repeated until a valid candidate is found.

-
- [1] P. W. Shor, in *Proceedings 35th Annual Symposium on Foundations of Computer Science* (IEEE Computer Society Press, Los Alamitos, CA, 1994), pp. 124–134.
 - [2] L. K. Grover, in *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing* (Association for Computing Machinery, New York, NY, 1996), pp. 212–219.
 - [3] S. Lloyd, Universal quantum simulators, *Science* **273**, 1073 (1996).
 - [4] I. M. Georgescu, S. Ashhab, and F. Nori, Quantum simulation, *Rev. Mod. Phys.* **86**, 153 (2014).
 - [5] P. W. Shor, in *Proceedings of 37th Conference on Foundations of Computer Science* (IEEE, Los Alamitos, CA, 1996), pp. 56–65.
 - [6] Google Quantum AI and Collaborators, Quantum error correction below the surface code threshold, *Nature* **638**, 920 (2025).
 - [7] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition* (Cambridge University Press, Cambridge, UK, 2010).
 - [8] S. Bravyi and A. Kitaev, Universal quantum computation with ideal Clifford gates and noisy ancillas, *Phys. Rev. A* **71**, 022316 (2005).
 - [9] K. Svore, A. Aho, A. Cross, I. Chuang, and I. Markov, A layered software architecture for quantum computing design tools, *Computer* **39**, 74 (2006).
 - [10] T. Häner, D. S. Steiger, K. Svore, and M. Troyer, A software methodology for compiling quantum programs, *Quantum Sci. Technol.* **3**, 020501 (2018).
 - [11] F. T. Chong, D. Franklin, and M. Martonosi, Programming languages and compiler design for realistic quantum hardware, *Nature* **549**, 180 (2017).
 - [12] V. A. Alfred, S. L. Monica, and D. U. Jeffrey, *Compilers Principles, Techniques & Tools* (Pearson Education, Boston, MA, 2007).
 - [13] V. Shende, S. Bullock, and I. Markov, Synthesis of quantum-logic circuits, *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **25**, 1000 (2006).
 - [14] C. M. Dawson and M. A. Nielsen, The Solovay–Kitaev algorithm, arXiv preprint [arXiv:quant-ph/0505030](https://arxiv.org/abs/quant-ph/0505030).
 - [15] N. J. Ross and P. Selinger, Optimal ancilla-free Clifford + T approximation of Z -rotations, *Quantum Inf. Comput.* **16**, 901 (2016).
 - [16] A. Cowtan, S. Dilkes, R. Duncan, A. Krajenbrink, W. Simmons, and S. Sivarajah, in *14th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2019)*, Leibniz International Proceedings in Informatics (LIPIcs), Vol. 135, edited by W. van Dam and L. Mančinska (Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2019), pp. 5:1–5:32.
 - [17] M. Treinish *et al.*, Qiskit/qiskit-terra: Qiskit terra 0.22.3, 2022, doi:[10.5281/zenodo.7352457](https://doi.org/10.5281/zenodo.7352457)
 - [18] S. Sivarajah, S. Dilkes, A. Cowtan, W. Simmons, A. Edgington, and R. Duncan, t|ket): A retargetable compiler for NISQ devices, *Quantum Sci. Technol.* **6**, 014003 (2021).
 - [19] M. Amy and V. Gheorghiu, staq—A full-stack quantum processing toolkit, *Quantum Sci. Technol.* **5**, 034016 (2020).
 - [20] B. Coecke and R. Duncan, Interacting quantum observables: Categorical algebra and diagrammatics, *New J. Phys.* **13**, 043016 (2011).
 - [21] R. Duncan, A. Kissinger, S. Perdrix, and J. Van De Wetering, Graph-theoretic simplification of quantum circuits with the ZX-calculus, *Quantum* **4**, 279 (2020).
 - [22] A. Kissinger and J. van de Wetering, Reducing the number of non-Clifford gates in quantum circuits, *Phys. Rev. A* **102**, 022406 (2020).
 - [23] 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 Des. Integr. Circuits Syst.* **32**, 818 (2013).
 - [24] M. Amy, D. Maslov, and M. Mosca, Polynomial-time T -depth optimization of Clifford + T circuits via matroid partitioning, *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **33**, 1476 (2014).
 - [25] M. Amy and M. Mosca, T -count optimization and Reed–Muller codes, *IEEE Trans. Inf. Theory* **65**, 4771 (2019).
 - [26] L. E. Heyfron and E. T. Campbell, An efficient quantum compiler that reduces T count, *Quantum Sci. Technol.* **4**, 015004 (2019).
 - [27] V. Vandrale, Lower T -count with faster algorithms, *Quantum* **9**, 1860 (2025).
 - [28] F. J. Ruiz, T. Laakkonen, J. Bausch, M. Balog, M. Barekatin, F. J. Heras, A. Novikov, N. Fitzpatrick, B. Romera-Paredes, J. van de Wetering *et al.*, Quantum circuit optimization with alphasensor, *Nat. Mach. Intell.* **7**, 374 (2025).
 - [29] D. Gosset, V. Kliuchnikov, M. Mosca, and V. Russo, An algorithm for the T -count, [arXiv:1308.4134](https://arxiv.org/abs/1308.4134).
 - [30] D. Litinski, A game of surface codes: Large-scale quantum computing with lattice surgery, *Quantum* **3**, 128 (2019).
 - [31] Y. Hirano and K. Fujii, in *2025 IEEE International Conference on Quantum Computing and Engineering (QCE)* (IEEE, Albuquerque, NM, USA, 2025), pp. 670–680.

- [32] F. Zhang and J. Chen, Optimizing T gates in Clifford + T circuit as $\pi/4$ rotations around paulis, [arXiv:1903.12456](#).
- [33] V. Vandaale, S. Perdrix, and C. Vuillot, Optimal number of parametrized rotations and hadamard gates in parametrized Clifford circuits with non-repeated parameters, [arXiv:2407.07846](#).
- [34] Y. Mori, H. Hakoshima, K. Sudo, T. Mori, K. Mitarai, and K. Fujii, Quantum circuit unoptimization, *Phys. Rev. Res.* **7**, 023139 (2025).
- [35] Y. Kim, A. Eddins, S. Anand, K. X. Wei, E. Van Den Berg, S. Rosenblatt, H. Nayfeh, Y. Wu, M. Zaletel, K. Temme *et al.*, Evidence for the utility of quantum computing before fault tolerance, *Nature* **618**, 500 (2023).
- [36] N. Yoshioka, T. Okubo, Y. Suzuki, Y. Koizumi, and W. Mizukami, Hunting for quantum-classical crossover in condensed matter problems, *npj Quantum Inf.* **10**, 45 (2024).
- [37] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, A. Bengtsson, S. Boixo, M. Broughton, B. B. Buckley *et al.*, Observation of separated dynamics of charge and spin in the Fermi-Hubbard model, [arXiv preprint arXiv:2010.07965](#).
- [38] F. Alam, J. L. Bosse, I. Čepaitė, A. Chapman, L. Clinton, M. Crichigno, E. Crosson, T. Cubitt, C. Derby, O. Dowinton *et al.*, Programmable digital quantum simulation of 2D Fermi-Hubbard dynamics using 72 superconducting qubits, [arXiv preprint arXiv:2510.26845](#).
- [39] H. Yu, Y. Zhao, and T.-C. Wei, Simulating large-size quantum spin chains on cloud-based superconducting quantum computers, *Phys. Rev. Res.* **5**, 013183 (2023).
- [40] P. W. Claeys and A. Lamacraft, Maximum velocity quantum circuits, *Phys. Rev. Res.* **2**, 033032 (2020).
- [41] A. Kissinger and J. van de Wetering, PyZX: Large scale automated diagrammatic reasoning, *Electron. Proc. Theor. Comput. Sci.* **318**, 229 (2020).
- [42] J. Håstad, Tensor rank is NP-complete, *J. Algorithms* **11**, 644 (1990).
- [43] C. Gidney, Stim: A fast stabilizer circuit simulator, *Quantum* **5**, 497 (2021).
- [44] A. W. Cross, L. S. Bishop, J. A. Smolin, and J. M. Gambetta, Open quantum assembly language, [arXiv preprint arXiv:1707.03429](#).
- [45] Reversible logic synthesis benchmarks page, <http://webhome.cs.uvic.ca/dmaslov/> (2011).
- [46] R. Wille, L. Berent, T. Forster, J. Kunasaikaran, K. Mato, T. Peham, N. Quetschlich, D. Rovara, A. Sander, L. Schmid *et al.*, in *2024 IEEE International Conference on Quantum Software (QSW)*, (IEEE, Shenzhen, China, 2024), pp. 1–8.
- [47] MCR, <https://github.com/yforest-osaka/mcr> (2025).
- [48] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P. J. Love, A. Aspuru-Guzik, and J. L. O’Brien, A variational eigenvalue solver on a photonic quantum processor, *Nat. Commun.* **5**, 4213 (2014).
- [49] E. Farhi, J. Goldstone, and S. Gutmann, A quantum approximate optimization algorithm, [arXiv preprint arXiv:1411.4028](#).
- [50] Y. Akahoshi, K. Maruyama, H. Oshima, S. Sato, and K. Fujii, Partially fault-tolerant quantum computing architecture with error-corrected Clifford gates and space-time efficient analog rotations, *PRX Quantum* **5**, 010337 (2024).
- [51] R. Toshio, Y. Akahoshi, J. Fujisaki, H. Oshima, S. Sato, and K. Fujii, Practical quantum advantage on partially fault-tolerant quantum computer, *Phys. Rev. X* **15**, 021057 (2025).
- [52] J. van de Wetering, R. Yeung, T. Laakkonen, and A. Kissinger, Optimal compilation of parametrised quantum circuits, *Quantum* **9**, 1828 (2025).
- [53] Y. Nakano, H. Hakoshima, K. Mitarai, and K. Fujii, Markov-chain monte carlo method enhanced by a quantum alternating operator ansatz, *Phys. Rev. Res.* **6**, 033105 (2024).
- [54] D. Gottesman and I. L. Chuang, Demonstrating the viability of universal quantum computation using teleportation and single-qubit operations, *Nature* **402**, 390 (1999).
- [55] J. T. Anderson, On groups in the qubit Clifford hierarchy, *Quantum* **8**, 1370 (2024).
- [56] C. Kiumi and B. Koczor, TE-PAI: Exact time evolution by sampling random circuits, *Quantum Sci. Technol.* **10**, 045071 (2025).