


# Ultra-Low Logical Depth Fault-Tolerant Quantum Circuit Synthesis via Lattice Surgery

Chien-Tung (Cherie) Kuo   
 Dept. of Electrical Engineering  
 National Taiwan University  
 Taipei, Taiwan  
 b10901099@ntu.edu.tw

Cheng-En Tsai  
 Dept. of Electrical Engineering  
 National Taiwan University  
 Taipei, Taiwan  
 b10901098@ntu.edu.tw

Chung-Yang (Ric) Huang  
 Dept. of Electrical Engineering  
 National Taiwan University  
 Taipei, Taiwan  
 cyhuang@ntu.edu.tw

**Abstract**—We present FTQCS, a ZX-calculus-based fault-tolerant circuit synthesis framework that compiles quantum circuits into lattice-surgery schedules for surface-code execution. FTQCS uses global ZX rewrites to optimize circuits, align them with merge/split primitives, and generate space-aware layouts. For Clifford circuits, it automates constant logical-depth synthesis; for Clifford+T, it extends this strategy to efficiently incorporate non-Clifford resources. The framework reduces the asymptotic space-time costs to  $O(n^2)$ , improving upon the  $O(n^3)$  complexity of prior approaches and delivering substantial practical improvements. The experimental results show that FTQCS achieves near-constant logical-depth, speeds up execution by over 148×, and cuts space-time cost up to 16.6× on benchmark circuits. Implemented as an extension of Qsyn, FTQCS provides a scalable, verifiable path from high-level circuits to resource-efficient lattice-surgery schedules.

**Index Terms**—quantum circuit synthesis, fault tolerance, lattice surgery, ZX-calculus, quantum design automation

## I. INTRODUCTION

Quantum computing promises to solve classically intractable problems in areas like drug discovery, material science, cryptography, and optimization [1]–[4]. However, current quantum processors, operating in the noisy intermediate-scale quantum (NISQ) era, are hindered by the inherent fragility of their qubits. Decoherence and gate noise cause errors to accumulate rapidly, severely limiting the size and depth of executable circuits [5].

To achieve scalable quantum computation, it is essential to employ fault-tolerant quantum error correction (QEC). QEC protects quantum information by encoding logical qubits using many physical qubits, allowing for error detection and correction. Among various QEC techniques, surface code [6] is a leading candidate for near-term hardware due to its high error threshold and reliance on local interactions. This aligns with the prevailing superconducting architectures in modern quantum computers [7], [8], which are based on a 2D grid connectivity.

Logical operations in the surface code can be efficiently executed via lattice surgery, a scheme that realizes quantum gates by merging and splitting logical qubit patches [9], [10]. While theoretical frameworks have shown that Clifford gates can be efficiently absorbed into multi-qubit Pauli measurements [11], automating the physical routing and layout required to execute these measurements remains a critical challenge.

This work is supported by the National Science and Technology Council, R.O.C., Project Nos.: NSTC 114-2119-M-002-020 and NSTC 114-2640-E-002-001.

Existing automated methods [12]–[15] typically focus on gate-by-gate mapping and scheduling. While useful for specific subproblems, they are often suboptimal in terms of space-time costs because they do not exploit global circuit structure. A complementary approach, exemplified by the SAT-based synthesis method [16], optimizes small subroutines but is limited in scalability by its reliance on exhaustive constraint solving. Recently, several concurrent works [17]–[19] have extended the Pauli-based computation framework to address specific architectural bottlenecks, utilizing dynamic resource allocation to minimize transfer overhead. However, these approaches primarily focus on heuristic scheduling and architectural management—optimizing *how* a sequence is executed rather than simplifying the logical structure itself. These limitations highlight the need for a scalable framework capable of automating global circuit optimization to realize the theoretical potential of lattice surgery.

To address these challenges, we introduce Fault Tolerant Quantum Circuit Synthesis (FTQCS), a novel end-to-end framework that pioneers a complete optimization flow for lattice-surgery synthesis by using the ZX-calculus [20] as a core engine. By leveraging the ZX-calculus’s powerful rewrite rules—which have a direct structural correspondence with lattice-surgery operations—FTQCS enables aggressive, holistic circuit-level optimization to achieve performance beyond gate-by-gate compilation methods. Our key contributions are:

- 1) **The first complete ZX-based lattice surgery Synthesis Pipeline:** We introduce the first complete framework that maps quantum circuits directly to lattice-surgery schedules via ZX-graphs, leveraging global structural optimization to maximize space-time efficiency.
- 2) **Constant logical depth Clifford synthesis with superior asymptotic scaling:** By exploiting ZX normal forms [21], our approach guarantees constant logical depth ( $O(1)$ ) for Clifford circuits. This reduces the asymptotic space-time cost to  $O(n^2)$ , strictly outperforming the  $O(n^3)$  scaling of prior gate-by-gate methods.

Through this integration, FTQCS bridges the gap between abstract circuit optimization and concrete fault-tolerant execution, establishing a scalable and powerful framework for future quantum compilers.

## II. BACKGROUND

This section provides foundational knowledge for our synthesis framework. Subsection II-A covers the surface code and

lattice surgery. Subsection II-B discusses phase operations for lattice surgery. Finally, Subsection II-C introduces the ZX-calculus as lattice surgery synthesis engine.

### A. Surface Code Operations and Lattice Surgery

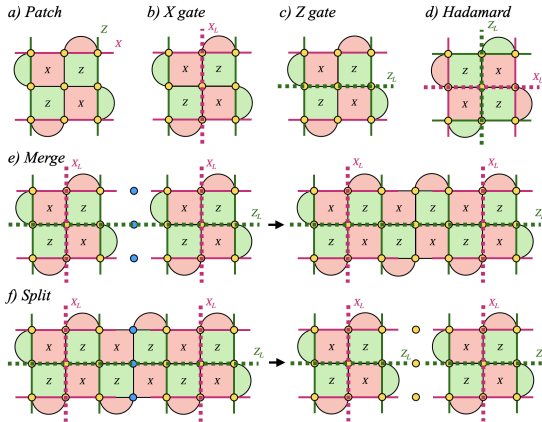


Fig. 1. Illustrations of surface code operations and lattice surgery.

Lattice surgery is a fault-tolerant scheme for executing logical quantum operations on the surface code, which encodes a logical qubit within a patch of physical qubits defined by its boundaries (X and Z) [9], [10]. Single-qubit operations are performed by manipulating these boundaries:

- 1) Logical Z/X operations apply z/x gates to qubits on the Z- or X-boundaries, respectively, as shown in Fig. 1(b,c).
- 2) Logical Hadamard operation is implemented by swapping the boundaries, as illustrated in Fig. 1(d).

Multi-qubit Pauli measurements [11] are realized by merging logical patches into a single entity (Fig. 1(e)) and subsequently splitting them to restore the qubits (Fig. 1(f)).

Furthermore, lattice surgery is a one-way computing paradigm [22]. Operations are not reversible, instead, they are executed as a sequence of feed-forward measurements, making the compilation process a single, time-ordered flow. This unique characteristic makes it a natural and efficient target for our synthesis approach.

### B. Phase Injection for Lattice Surgery

While lattice surgery natively implements Pauli product measurements, Pauli rotations are realized via *teleportation*, where a data qubit is entangled with a specific ancilla state through a merge operation [11]. Our framework unifies Clifford and non-Clifford operations into a single “phase injection” abstraction, differing only in the ancilla resource required:

- **Clifford Rotations ( $\pi/2$ ):** Operations like the  $S$  gates consume a logical  $|Y\rangle$  state. As a stabilizer state,  $|Y\rangle$  is a Clifford resource that can be prepared deterministically and locally using lattice twists or boundary defects [23], [24], incurring negligible overhead.
- **Non-Clifford Rotations ( $\pi/4$ ):** Operations like the  $T$  gate consume a non-Clifford magic state. While traditionally a bottleneck, recent groundbreaking works [25]–[28] show that these states can be prepared efficiently.

These technological advancements allow us to assume that a dedicated, external factory prepares these states in parallel with the main computation. Leveraging this, FTQCS concentrate on the global optimization of the circuit structure—establishing a constant-depth Clifford backbone—without being encumbered by the traditionally prohibitive costs of phase generation.

### C. ZX-Calculus as Lattice Surgery Synthesis Engine

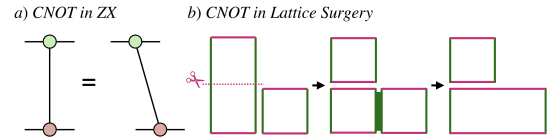


Fig. 2. Illustration of ZX-calculus and lattice surgery.

The core of our approach is using the ZX-calculus as a synthesis engine for lattice surgery, building on the structural similarities between these two frameworks [29]–[31].

- 1) **ZX-spiders and lattice surgery patches:** The fundamental nodes of a ZX-diagram, called Z-spiders or X-spiders, correspond directly to the Z- or X-boundary logical qubit patches of the surface code. A two-to-one Z-spider naturally corresponds to a two-qubit merge operation on Z-boundary in lattice surgery.
- 2) **Topological operations as rewrite rules:** The powerful rewrite rules of the ZX-calculus directly correspond to the topological operations of lattice surgery. For instance, fusing two Z-spiders maps to a Z-basis merge of logical patches, providing an algebraic way to simplify circuits.

For example, a CNOT gate is represented as a simple two-spider diagram in the ZX-calculus as shown in Fig. 2. Our engine applies simplification rules to this representation and then extracts the optimal lattice-surgery schedules, enabling circuit-level optimizations that are not possible with conventional gate-by-gate compilation methods.

### III. MOTIVATION

The surface code and lattice surgery offer a practical foundation for fault-tolerant quantum computation, but efficiently compiling circuits into their merge and split operations remains a significant challenge. For example, a SWAP is typically decomposed into three consecutive CNOT gates. In conventional strategies, logical qubits and ancilla are arranged in a local grid (e.g.,  $5 \times 5$  patches for four logical qubits as in Fig. 3(a)), and each CNOT is routed and executed sequentially (Fig. 3(b)). Since each CNOT consumes two logical time steps, the entire SWAP requires six steps in total.

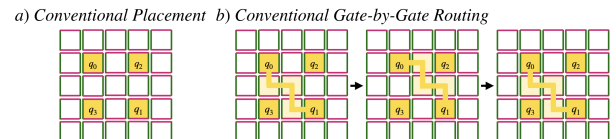


Fig. 3. Conventional synthesis of a SWAP gate.

A natural question arises: *Can we exploit abstract-level structure to improve lattice-surgery compilation?* This is where the ZX-calculus provides unique potential. As a graphical

representation, ZX-calculus supports aggressive algebraic simplification and its core rewrite rules align directly with the merge and split primitives of lattice surgery.

For the SWAP gate, the ZX representation (Fig. 4(a)) reveals that it can be simplified to a crossing of qubit wires, eliminating the need for additional intermediate nodes. Leveraging this observation, we map the logical qubits diagonally in an  $n \times n$  lattice (Fig. 4(b,c)), allowing the SWAP to be executed in only **two logical time steps**, a threefold reduction compared to the conventional approach. This illustrates the potential of ZX-calculus as a synthesis engine for lattice surgery, enabling both structural simplification and efficient layout strategies.

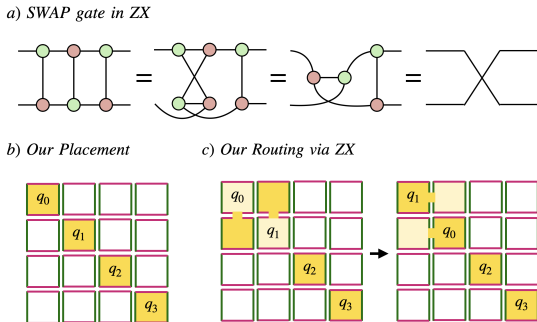


Fig. 4. Our FTQLS with ZX Calculus of a SWAP gate.

The key motivation behind our work is therefore to *bridge circuit simplification and fault-tolerant compilation* in a single framework. We aim to unify ZX-based reasoning with lattice-surgery synthesis, enabling space-time reductions while providing a scalable, extensible path toward practical surface-code quantum computation.

#### IV. METHODOLOGY

This section introduces **Fault-Tolerant Quantum Circuit Synthesis (FTQCS)**, a novel end-to-end framework for lattice-surgery-based quantum circuit synthesis that leverages the **ZX-calculus** as its core engine. As illustrated in Figure 5, our pipeline bridges abstract graph-theoretic reasoning with concrete physical execution by translating the circuit into a ZX-graph, arranging the graph for one-way computation, and then synthesizing it into a sequence of physically realizable lattice-surgery schedules. By enabling global circuit-level optimizations beyond the reach of existing methods, our FTQCS establishes a scalable path toward resource-efficient fault-tolerant quantum computation.

##### A. Translating Quantum Circuit into ZX-graph Representation

The first stage of our pipeline translates an input Clifford circuit into a ZX-graph, a structured graphical representation well suited for algebraic manipulation. Each quantum gate is mapped to its corresponding Z- or X-spider, as illustrated in Fig. 6(a). Once in this form, the circuit undergoes a sequence of ZX rewrite rules implemented in the Qsyn framework [32]. These algebraic transformations ensure that as the circuit enters the lattice-surgery synthesis stage, it has benefited from the algebraic reductions that directly contribute to the space-time improvements of the compiled layouts.

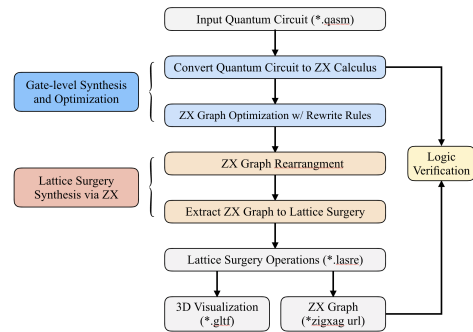
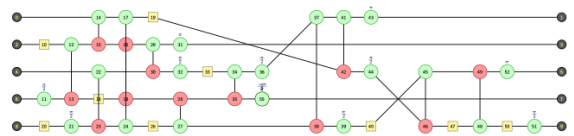
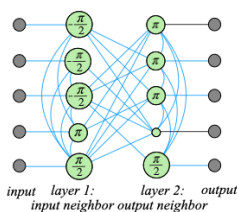


Fig. 5. Overview of the proposed Lattice Surgery Synthesis pipeline.

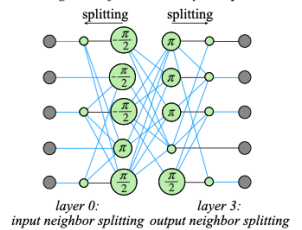
##### a) Convert Quantum Circuit to ZX Graph



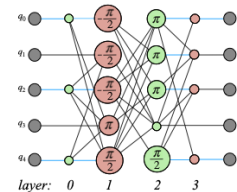
##### b) ZX diagram in GLSC form (zx opt)



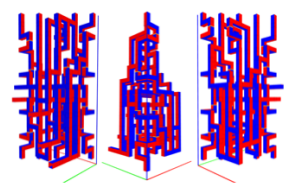
##### c) ZX arrangement for one-way computation



##### d) Color transformation for minimizing routing



##### f) Post-synthesis 3D Visualization



##### e) Layer-by-Layer Extraction

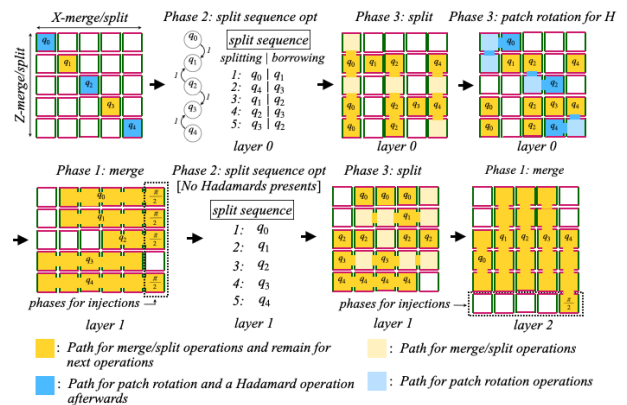


Fig. 6. End-to-end overview of our proposed synthesis pipeline.

##### B. ZX-graph Optimization and Normal Forms

After translation, the ZX-graph is optimized into a special normal form to enable our ultra-low logical time step synthesis. The core idea is to apply simplification rules that remove all internal spiders, yielding a diagram composed solely of boundary spiders and Hadamard edges, as illustrated in Fig. 6(b). This

structure is known as a **graph-like ZX-diagram in Graph State with Local Cliffords (GSLC) form**. This normal form is highly beneficial because it guarantees a constant-depth execution that is independent of the circuit size.

**Definition 1** (Graph-like ZX-Diagram [21]). *We formally define a graph-like diagram with Hadamards as one that satisfies the following properties:*

- 1) Every spider is a Z-spider.
- 2) Spiders are only connected via Hadamard edges.
- 3) There are no self-loops or parallel edges.
- 4) Every Z-spider is connected to at most one input and at most one output.
- 5) Every input and output wire is connected to a Z-spider.

**Definition 2** (GSLC Form [21]). *A diagram is in GSLC form when it is graph-like and has no internal spiders. Each spider carrying a phase  $\frac{n\pi}{2}$  and connecting to one input or output.*

### C. ZX-graph Arrangement for One-Way Quantum Computation

The optimized ZX-graph must be arranged on a 2D grid to enable one-way computation synthesis, with the x-axis representing logical time steps and the y-axis representing logical qubits. This arrangement enforces the physical constraint that edges, which correspond to lattice-surgery operations, exist only between adjacent time steps and ensure a physically realizable arrangement.

The GSLC form simplifies this process, as a Clifford circuit with  $n$  qubits contains at most  $2n$  spiders. As shown in Fig. 6(b), input and output wires are connected to the first and the last logical time steps, and the input-neighboring and output-neighboring spiders are placed in Layer 1 and 2, respectively. To handle internal edges between spiders, we introduce two additional layers, Layer 0 and Layer 3, as shown in Fig. 6(c). These two layers allow for the splitting of nodes (i.e. Z spiders) so that all edges are connected exclusively between adjacent logical time steps. Therefore, it will be straightforward to render a physically realizable arrangement for one-way computation on a final layout with at most four layers. This ensures the synthesis process to complete in constant logical depth regardless of circuit size.

### D. Lattice Surgery Synthesis for Parallelism and Initial Layout

Our framework prioritizes minimizing logical depth by deliberately expanding the spatial layout to an  $O(n^2)$  grid, with logical qubits arranged along the diagonal. While this appears spatially costly, the quadratic area acts as a necessary buffer for parallelism.

In compact layouts, attempting simultaneous X- and Z-operations is highly inefficient, as their orthogonal boundaries fragment available space and cause blocking. To solve this, we utilize the  $n \times n$  grid and enforce a strict orientation: X-boundaries align north–south and Z-boundaries east–west. This geometric constraint transforms potential routing conflicts into layer-wise parallelism, guaranteeing that every logical qubit can interact without long-distance detours.

This parallelism is algorithmically enforced via a color transformation to the ZX-graph (Fig. 6(d)), which ensures

that Z- and X-merges alternate across layers. Conceptually, the enlarged layout functions as two distinct “construction crews”, one building north–south roads (Z-merges) and the other east–west roads (X-merges)—each with reserved lanes. This strategic alternation aligns operations with the grid’s geometry, maximizing efficiency for each layer.

### E. Core Layer-by-Layer Extraction

The final stage of our framework translates the optimized ZX-graph into physically realizable lattice-surgery schedules. This translation is a direct mapping from the graphical representation to merge and split primitives, producing a precise lattice-surgery schedules that maintain the logic of the computation.

---

#### Algorithm 1 Layer-by-Layer Extraction

---

```

procedure SYNTHESIZELATTICESURGERYFROMGRAPH
  for each layer  $L$  from 0 to  $n$  do
    {Process layer  $L$  and prepare for layer  $L + 1$ }
    Phase 1: Merge Operations
    {Execute merges and phase injections}
    Let  $\mathcal{N}_L$  be the set of nodes in layer  $L$ .
    for each node  $N \in \mathcal{N}_L$  do
      Let  $\mathcal{N}_i$  be the incoming neighbors of  $N$  from layer  $L - 1$ 
      if  $N$  has a phase  $(\pm \frac{\pi}{2}, \pm \frac{\pi}{4})$  then
        PHASEINJECT( $\mathcal{N}_i, N$ .phase)
      else
        MERGE( $\mathcal{N}_i$ )
      end if
    end for
    Phase 2: Split Sequence Optimization
    {Determine the optimal split sequence for layer  $L + 1$ }
    Let  $G_{borrow}$  be the directed, weighted graph representing
    patch borrowing costs for  $\mathcal{N}_L$ 
     $S_{min} = \text{MINCOSTEXECSEQ}(G_{borrow})$ 
    Phase 3: Split Operations
    {Perform splits and prepare patches for next layer}
    for each node  $N$  in  $S_{min}$  do
      Let  $\mathcal{N}_o$  be the outgoing neighbors of  $N$  to layer  $N + 1$ 
      SPLIT( $\mathcal{N}_o$ )
       $\mathcal{H}_o = \text{CONNECTEDBYHADAMARD}(N, \mathcal{N}_o)$ 
      for each node  $N_h$  in  $\mathcal{H}_o$  do
        PATCHROTATION( $N_h$ )
      end for
    end for
  end for
end procedure

```

---

1) *Phase 1: Merge Operations:* For each layer, we perform all required merges and phase injections. The graph arrangement ensures that patches are correctly positioned from the previous layer, allowing for directed merges.

2) *Phase 2: Split Sequence Optimization:* Following the merges, we determine the optimal sequence of splits to prepare patches for the next layer. We model this as a directed graph where nodes represent patches to be split and edge weights correspond to the cost of borrowing neighboring space for further Hadamard operations. We assign a weight of 1 to a

borrowing operation if the space is not required by another qubit, and a weight of 2 if the space must be returned after the operation. We then find a minimal-cost topological sequence of nodes to ensure that the borrowing operations are scheduled efficiently. This allows us to optimize the execution order, ensuring that patches are available when needed.

3) *Phase 3: Split Operations*: Finally, we perform the splits in the determined order. For Hadamard edges, a patch rotation is executed by borrowing space from neighboring patches, using a three-patch L-shape configuration shown in Fig. 7.

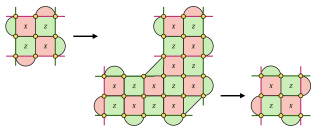


Fig. 7. Patch rotation using an L-shape merging.

### F. Layer-by-Layer Extraction: A Walkthrough

We illustrate our layer-by-layer extraction algorithm by using the graph shown in Fig. 6(d) as an example in Fig. 6(e).

a) *Layer 0 Extraction*: Since each node in Layer 0 has only a single incoming edge for each node, no merge operations are needed. We proceed directly to Phase 2, Split Sequence Optimization. We model the splitting process as a borrowing cost graph, where edge weights represent the cost of borrowing a neighboring patch for Hadamard operations. A directed edge from a node in Layer 0 to a node in Layer 1 has a weight of 1 if the destination patch is available, and a weight of 2 if the space is needed for another operation. For example, the outgoing Hadamard edge from  $q_0$  to  $q_1$  has a weight of 1, as the destination patch (1,0) is available. After obtaining the borrowing cost graph, we find the minimum-cost splitting sequence starting from the boundary nodes. We then move to Phase 3, Split Operations, where we split nodes and perform patch rotations for the Hadamard edges. For this, we either borrow a nearby valid patch or assign a new one, performing the rotation to prepare for the next layer.

b) *Layer 1 Extraction*: The process for Layer 1 is simpler. In Phase 1, we perform all required X-merge operations. If a merge operation includes a phase, we execute a phase injection, which is prepared independently on the side. We then proceed to Phase 2. Since there are no outgoing Hadamard edges in this layer, we can perform the splits sequentially, as there are no borrowing problems to solve. Finally, in Phase 3, we execute a pure splitting operation for each node.

### G. Post-Synthesis Visualization and Verification

The final output of our framework is a *LaSre*-compatible representation [16] of the lattice-surgery layout. This standardized format enables the generation of interactive 3D visualizations that reveal the full space-time structure of the computation, as illustrated in Fig. 6(f). In parallel, the *LaSre* description can be translated back into a ZX-calculus graph. By composing this reconstructed graph with the ZX-graph of the original circuit and applying standard rewrite rules, we can reduce the composite diagram to the identity, thereby proving the

functional equivalence of the synthesized layout. This step leverages the completeness of our work, ensuring our flow is not only resource-efficient but also logically sound.

## V. COMPLEXITY ANALYSIS

This section presents a theoretical analysis of the space-time complexity of our framework. We demonstrate that for general Clifford circuits with high connectivity, FTQCS achieves an asymptotic space-time cost of  $O(n^2)$ , improving upon the  $O(n^3)$  scaling characteristic of prior gate-by-gate compilers.

### A. Complexity of Prior Automated Gate-by-Gate Compilers

Existing gate-by-gate compilers [12]–[15] operate with a space complexity of  $O(n)$  and a worst case time complexity of  $O(n^2)$ , yielding a total space-time volume of  $O(n^3)$ .

1) *Space  $O(n)$* : These methods prioritize spatial density by utilizing compact layouts, where logical qubits are embedded in a tight, periodic array separated only by minimal ancillary boundaries.

2) *Time  $O(n^2)$* : The temporal cost is the critical bottleneck. A Clifford circuit on  $n$  qubits can contain up to  $O(n^2)$  two-qubit gates. Because gate-by-gate approaches process operations sequentially and have limited parallelism due to blocking, the total execution depth scales with the gate count. In the worst case of all-to-all connectivity, two-qubit gates requires serial routing, resulting in  $O(n^2)$  logical time steps.

### B. Complexity of the FTQCS Framework

In contrast, FTQCS fundamentally alters this trade-off by investing in space to minimize time. We guarantee constant logical depth at the cost of  $O(n^2)$  space, resulting in a total space-time volume of  $O(n^2)$  for Clifford circuits.

1) *Space Complexity–The Cost of Parallelism*: Our approach requires  $O(n^2)$  space because we arrange logical qubits on a diagonal  $n \times n$  grid. While this appears to be a larger overhead compared to compact  $O(n)$  layouts, the  $O(n^2)$  area acts as a global routing fabric. It provides dedicated spatial lanes to ensure that every logical patch is reachable from any other patch. This eliminates the routing congestion that forces gate-by-gate compilers to serialize operations.

2) *Time Complexity–Structural Constant Logical Depth*: Our synthesized circuits achieve a constant time complexity,  $O(1)$ . This is not a heuristic, but a structural guarantee derived from the GSLC normal form [21]. By reducing the Clifford circuit to a graph-like form with no internal spiders, the dependency depth of the computation is collapsed. Our arrangement procedure maps this collapsed graph to the lattice in a fixed pattern of at most four logical layers. Consequently, the compiled Clifford circuits will always execute in constant logical depth regardless of the circuit density.

3) *Stability vs. Volatility*: A critical advantage of FTQCS is performance stability. Prior gate-by-gate compilers suffer from high volatility. While they may achieve lower overhead on trivial, nearest-neighbor circuits, their costs explode to  $O(n^3)$  on the complex, highly connected topologies required for useful quantum algorithms. FTQCS eliminates this worst-case bottleneck, by structurally guaranteeing  $O(1)$  logical depth.

TABLE I  
COMPARISON OF SPACE-TIME COST ACROSS LSC, EDP, AND OUR PROPOSED FTQCS ON RANDOM CLIFFORD CIRCUITS.

Circuit Size (qubits, gates)	LSC				EDP				FTQCS (Ours)				Advantage
	Width	Height	Time	Space-Time	Width	Height	Time	Space-Time	Width	Height	Time	Space-Time	
(5, 33.0)	5	3	119.6	1794	7	5	100.8	3528.0	5	5	12.6	<b>315</b>	5.70× (↓ 82.5%)
(10, 116.8)	7	3	374.9	7872.9	11	5	255.5	14052.5	10	10	17.2	<b>1720</b>	4.58× (↓ 78.1%)
(20, 426.6)	17	3	1266.5	64591.5	11	9	632.0	62568.0	20	20	18.7	<b>7480</b>	8.36× (↓ 88.0%)
(30, 919.3)	17	3	2630.5	134155.5	12	11	750.3	99039.6	30	30	19.6	<b>17640</b>	5.61× (↓ 82.2%)
(50, 2482.5)	27	3	6281.0	508761	21	11	1375.9	317832.9	50	50	20.2	<b>50500</b>	6.29× (↓ 84.1%)
(75, 5566.2)	40	3	12879.4	1545528	31	11	2303.3	785425.3	75	75	20.1	<b>113062.5</b>	6.95× (↓ 85.6%)
(100, 9849.4)	52	3	20867	3255252	21	21	3263.0	1438983	100	100	22.1	<b>221000</b>	6.51× (↓ 84.6%)

Geometric mean improvement vs best baseline: **6.25** × Avg. reduction: **83.6%**

TABLE II  
COMPARISON OF SPACE-TIME COST BETWEEN EDP AND OUR PROPOSED FTQCS ON SABRE [33] CLIFFORD+T BENCHMARK CIRCUITS.

Benchmark (.qasm)	EDP	FTQCS (Ours)	Advantage
	Space-Time	Space-Time	
3_17_13	5050	<b>315</b>	16.03× (↓ 93.8%)
4gt11_82	6230	<b>375</b>	16.61× (↓ 94.0%)
4gt13_92	12320	<b>1425</b>	8.65× (↓ 88.4%)
4mod5-v1_24	7630	<b>875</b>	8.72× (↓ 88.5%)
alu-v4_37	6685	<b>700</b>	9.55× (↓ 89.5%)
9symm1_195	14564580	<b>5602304</b>	2.60× (↓ 61.6%)
adr4_197	1431655	<b>502016</b>	2.85× (↓ 64.9%)
cm42a_207	737630	<b>270592</b>	2.73× (↓ 63.3%)
cycle10_2_110	2535380	<b>972288</b>	2.61× (↓ 61.7%)
dist_223	15645695	<b>5942784</b>	2.63× (↓ 62.0%)

## VI. EXPERIMENTS

### A. Evaluation Setup

To quantify resource efficiency, we adopt the *space-time cost* metric introduced by Beverland *et al.* [13]. This metric captures the trade-off between spatial and temporal overhead, directly reflects the fault-tolerant resource volume of a circuit.

We benchmark FTQCS against two representative lattice-surgery compilers: (i) *EDP (Edge-Disjoint Paths)* [13], which routes Clifford operations along edge-disjoint paths to achieve constant-depth execution of long-range interactions; and (ii) *LSC (Lattice-Surgery Compiler)* [14], a high-throughput streaming compiler capable of processing millions of gates in seconds. These two baselines provide complementary perspectives: depth-optimized routing and throughput-optimized compilation. Our benchmarks include (a) random Clifford circuits generated with Qiskit [34], and (b) Clifford+T circuits from the SABRE benchmark [33], widely used in compilation studies.

### B. Results on Clifford Circuits

Table I reports results for random Clifford circuits, and Fig. 8 visualizes execution time versus qubit count. FTQCS maintains near-constant time depth as circuit size increases: for 100-qubit circuits, FTQCS completes execution in only 22.1 time steps, compared to 20,867 for LSC and 3,263 for EDP—an improvement of over 148× relative to the best baseline. In terms of overall space-time cost, FTQCS achieves improvements ranging from 4.6× to 8.4× across mid-sized benchmarks, and still secures a 6.5× (84.6% reduction) on the 100-qubit case. Overall, the geometric mean improvement is 6.25× (83.6% reduction). These results empirically validate the predicted  $O(n^2)$  scaling of our heuristic and demonstrate its scalability and efficiency in the Clifford setting.

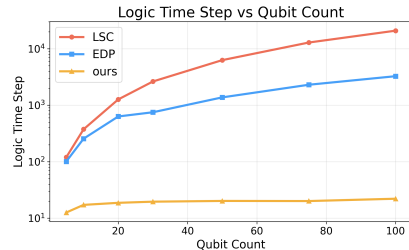


Fig. 8. Logic Time Step comparison on random Clifford circuits.

### C. Results on Clifford+T Circuits

In this regime, non-Clifford gates introduce *magic-state overheads* that restrict simplification opportunities and limit depth reduction. Even so, FTQCS delivers measurable benefits by exploiting structural simplifications in Clifford subcircuits and efficiently managing T-gate boundaries. Since *LSC* implementation does not support SABRE Clifford+T benchmarks, we compare only against *EDP*, which serves as the baseline for non-Clifford fault-tolerant compilation.

As shown in Table II, FTQCS achieves consistent and substantial reductions in space-time cost across all evaluated benchmarks. For example, on the large circuit `9symm1_195.qasm`, FTQCS reduces the space-time cost from  $1.46 \times 10^7$  to  $5.60 \times 10^6$ —a 2.6× improvement. These results confirm that the advantages of our synthesis flow extend beyond Clifford-only circuits, establishing its applicability to *universal quantum computation*.

## VII. CONCLUSION

In this paper, we introduced *Fault-Tolerant Quantum Circuit Synthesis (FTQCS)*, an end-to-end framework that integrates *ZX-calculus* optimization with lattice-surgery execution. By translating circuits into *ZX-graphs*, applying aggressive algebraic simplifications, and synthesizing them into merge-and-split operations, FTQCS achieves asymptotic  $O(n^2)$  space-time scaling, improving upon the  $O(n^3)$  cost of prior approaches and delivering over 148× faster execution time and up to 16.6× lower space-time cost on benchmark circuits.

## ACKNOWLEDGMENT

This work is supported by the National Science and Technology Council, R.O.C., Project Nos.: NSTC 114-2119-M-002-020 and NSTC 114-2640-E-002-001.

## REFERENCES

- [1] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Journal on Computing*, vol. 26, no. 5, pp. 1484–1509, 1997.
- [2] A. Aspuru-Guzik, A. D. Dutoi, P. J. Love, and M. Head-Gordon, "Simulated quantum computation of molecular energies," *Science*, vol. 309, no. 5741, pp. 1704–1707, 2005.
- [3] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd, "Quantum machine learning," *Nature*, vol. 549, no. 7671, pp. 195–202, 2017.
- [4] E. Farhi, J. Goldstone, and S. Gutmann, "A quantum approximate optimization algorithm," *arXiv preprint arXiv:1411.4028*, 2014.
- [5] J. Preskill, "Quantum computing in the nisq era and beyond," *Quantum*, vol. 2, p. 79, 2018.
- [6] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland, "Surface codes: Towards practical large-scale quantum computation," *Physical Review A*, vol. 86, no. 3, p. 032324, 2012.
- [7] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. S. L. Brandao, D. A. Buell, B. Burkett, Y. Chen, Z. Chen, B. Chiaro, R. Collins, W. Courtney, A. Dunsworth, E. Farhi, B. Foxen, A. Fowler, C. Gidney, M. Giustina, R. Graff, K. Guerin, S. Habegger, M. P. Harrigan, M. J. Hartmann, A. Ho, M. Hoffmann, T. Huang, T. S. Humble, S. V. Isakov, E. Jeffrey, Z. Jiang, D. Kafri, K. Kechedzhi, J. Kelly, P. V. Klimov, S. Knysh, A. Korotkov, F. Kostritsa, D. Landhuis, M. Lindmark, E. Lucero, D. Lyakh, S. Mandrà, J. R. McClean, M. McEwen, A. Megrant, X. Mi, K. Michielsen, M. Mohseni, J. Mutus, O. Naaman, M. Neeley, C. Neill, M. Y. Niu, E. Ostby, A. Petukhov, J. C. Platt, C. Quintana, E. G. Rieffel, P. Roushan, N. C. Rubin, D. Sank, K. J. Satzinger, V. Smelyanskiy, K. J. Sung, M. D. Trevithick, A. Vainsencher, B. Villalonga, T. White, Z. J. Yao, P. Yeh, A. Zalcman, H. Neven, and J. M. Martinis, "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, no. 7779, pp. 505–510, 2019.
- [8] J. Koch, T. Yu, J. Gambetta, A. A. Houck, D. I. Schuster, J. Majer, A. Blais, M. H. Devoret, S. M. Girvin, and R. J. Schoelkopf, "Charge-insensitive qubit design derived from the cooper pair box," *Physical Review A*, vol. 76, no. 4, p. 042319, 2007.
- [9] C. Horsman, A. G. Fowler, S. Devitt, and R. Van Meter, "Surface code quantum computing by lattice surgery," *New Journal of Physics*, vol. 14, no. 12, p. 123011, 2012.
- [10] E. T. Campbell, "A short introduction to lattice surgery," *arXiv preprint arXiv:1505.01797*, 2015.
- [11] D. Litinski, "A game of surface codes: Large-scale quantum computing with lattice surgery," *Quantum*, vol. 3, p. 128, Mar. 2019.
- [12] L. Lao, B. v. Wee, I. Ashraf, J. v. Someren, N. Khammassi, K. Bertels, and C. G. Almudever, "Mapping of lattice surgery-based quantum circuits on surface code architectures," *Quantum Science and Technology*, vol. 4, p. 015005, Sept. 2018.
- [13] M. Beverland, V. Kliuchnikov, and E. Schoute, "Surface code compilation via edge-disjoint paths," *PRX Quantum*, vol. 3, May 2022.
- [14] G. Watkins, H. M. Nguyen, K. Watkins, S. Pearce, H.-K. Lau, and A. Paler, "A High Performance Compiler for Very Large Scale Surface Code Computations," *Quantum*, vol. 8, p. 1354, May 2024.
- [15] A. Molavi, A. Xu, S. Tannu, and A. Albarghouthi, "Dependency-aware compilation for surface code quantum architectures," 2025.
- [16] D. B. Tan, M. Y. Niu, and C. Gidney, "A sat scalpel for lattice surgery: Representation and synthesis of subroutines for surface-code fault-tolerant quantum computing," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pp. 325–339, 2024.
- [17] S. Kan, Z. Du, C. Liu, M. Wang, Y. Ding, A. Li, Y. Mao, and S. Stein, "Sparo: Surface-code pauli-based architectural resource optimization for fault-tolerant quantum computing," 2025.
- [18] M. Wang, C. Liu, S. Garner, S. Stein, Y. Ding, P. J. Nair, and A. Li, "Tableau-based framework for efficient logical quantum compilation," 2025.
- [19] Y. Hirano and K. Fujii, "Locality-aware pauli-based computation for local magic state preparation," in *2025 IEEE International Conference on Quantum Computing and Engineering (QCE)*, p. 670–680, IEEE, Aug. 2025.
- [20] J. van de Wetering, "Zx-calculus for the working quantum computer scientist," 2020.
- [21] A. Kissinger and J. van de Wetering, *Picturing Quantum Software: An Introduction to the ZX-Calculus and Quantum Compilation*. Preprint, 2024.
- [22] R. Raussendorf and H. J. Briegel, "A one-way quantum computer," *Phys. Rev. Lett.*, vol. 86, pp. 5188–5191, May 2001.
- [23] C. Gidney, "Inplace access to the surface code y basis," *Quantum*, vol. 8, p. 1310, Apr. 2024.
- [24] B. J. Brown, K. Laubscher, M. S. Kesselring, and J. R. Wootton, "Poking holes and cutting corners to achieve clifford gates with the surface code," *Physical Review X*, vol. 7, May 2017.
- [25] C. Gidney, N. Shutty, and C. Jones, "Magic state cultivation: growing t states as cheap as cnot gates," 2024.
- [26] P. Sales Rodriguez, J. M. Robinson, P. N. Jepsen, Z. He, C. Duckering, C. Zhao, K.-H. Wu, J. Campo, K. Bagnall, M. Kwon, T. Karolyshyn, P. Weinberg, M. Cain, S. J. Evered, A. A. Geim, M. Kalinowski, S. H. Li, T. Manovitz, J. Amato-Grill, J. I. Basham, L. Bernstein, B. Braverman, A. Bylinskii, A. Choukri, R. J. DeAngelo, F. Fang, C. Fieweger, P. Frederick, D. Haines, M. Hamdan, J. Hammett, N. Hsu, M.-G. Hu, F. Huber, N. Jia, D. Kedar, M. Kornjača, F. Liu, J. Long, J. Lopatin, P. L. S. Lopes, X.-Z. Luo, T. Macrì, O. Marković, L. A. Mart'inez-Mart'inez, X. Meng, S. Ostermann, E. Ostroumov, D. Paquette, Z. Qiang, V. Shofman, A. Singh, M. Singh, N. Sinha, H. Thoreen, N. Wan, Y. Wang, D. Waxman-Lenz, T. Wong, J. Wurtz, A. Zhdanov, L. Zheng, M. Greiner, A. Keesling, N. Gemelke, V. Vuletić, T. Kitagawa, S.-T. Wang, D. Bluvstein, M. D. Lukin, A. Lukin, H. Zhou, and S. H. Cant'ú, "Experimental demonstration of logical magic state distillation," *Nature*, Jul 2025.
- [27] N. C. Brown, J. P. C. III, C. Granade, B. Heim, S. Wernli, C. Ryan-Anderson, D. Lucchetti, A. Paetznick, M. Roetteler, K. Svore, and A. Chernoguzov, "Advances in compilation for quantum hardware – a demonstration of magic state distillation and repeat-until-success protocols," 2023.
- [28] P. Sales Rodriguez, J. M. Robinson, P. N. Jepsen, Z. He, C. Duckering, C. Zhao, K.-H. Wu, J. Campo, K. Bagnall, M. Kwon, T. Karolyshyn, P. Weinberg, M. Cain, S. J. Evered, A. A. Geim, M. Kalinowski, S. H. Li, T. Manovitz, J. Amato-Grill, J. I. Basham, L. Bernstein, B. Braverman, A. Bylinskii, A. Choukri, R. J. DeAngelo, F. Fang, C. Fieweger, P. Frederick, D. Haines, M. Hamdan, J. Hammett, N. Hsu, M.-G. Hu, F. Huber, N. Jia, D. Kedar, M. Kornjača, F. Liu, J. Long, J. Lopatin, P. L. S. Lopes, X.-Z. Luo, T. Macrì, O. Marković, L. A. Mart'inez-Mart'inez, X. Meng, S. Ostermann, E. Ostroumov, D. Paquette, Z. Qiang, V. Shofman, A. Singh, M. Singh, N. Sinha, H. Thoreen, N. Wan, Y. Wang, D. Waxman-Lenz, T. Wong, J. Wurtz, A. Zhdanov, L. Zheng, M. Greiner, A. Keesling, N. Gemelke, V. Vuletić, T. Kitagawa, S.-T. Wang, D. Bluvstein, M. D. Lukin, A. Lukin, H. Zhou, and S. H. Cant'ú, "Experimental demonstration of logical magic state distillation," *Nature*, vol. 645, p. 620–625, July 2025.
- [29] B. Coecke and R. Duncan, "Interacting quantum observables: categorical algebra and diagrammatics," *New Journal of Physics*, vol. 13, no. 4, p. 043016, 2011.
- [30] M. Backens, "The zx-calculus is complete for stabilizer quantum mechanics," *New Journal of Physics*, vol. 16, no. 9, p. 093021, 2014.
- [31] N. de Beaudrap and C. Horsman, "The zx-calculus is a language for surface code lattice surgery," *arXiv preprint arXiv:1704.08670*, 2017.
- [32] M.-T. Lau, C.-Y. Cheng, C.-H. Lu, C.-H. Chuang, Y.-H. Kuo, H.-C. Yang, C.-T. Kuo, H.-Y. Chen, C.-Y. Tung, C.-E. Tsai, G.-H. Chen, L.-K. Lin, C.-H. Wang, T.-H. Wang, and C.-Y. R. Huang, "Qsyn: A developer-friendly quantum circuit synthesis framework for nisq era and beyond," in *2024 IEEE International Conference on Quantum Computing and Engineering (QCE)*, p. 535–536, IEEE, Sept. 2024.
- [33] G. Li, Y. Ding, and Y. Xie, "Tackling the qubit mapping problem for nisq-era quantum devices," 2019.
- [34] S. Bravyi and D. Maslov, "Hadamard-free circuits expose the structure of the clifford group," *IEEE Transactions on Information Theory*, vol. 67, p. 4546–4563, July 2021.