QECC-Synth: A Layout Synthesizer for Quantum Error Correction Codes on Sparse Hardware Architectures

Keyi Yin
keyin@ucsd.edu
University of California, San Diego
La Jolla, California, USA

Hezi Zhang
hez019@ucsd.edu
University of California, San Diego
La Jolla, California, USA

Xiang Fang
x8fang@ucsd.edu
University of California, San Diego
La Jolla, California, USA

Yunong Shi
shiyunon@amazon.com
AWS Quantum Technologies
New York, USA

Travis S. Humble
humblets@ornl.gov
Oak Ridge National Laboratory
Oak Ridge, Tennessee, USA

Ang Li
ang.li@pnnl.gov
Pacific Northwest National Laboratory
Richland, Washington, USA

Yufei Ding
yufeiding@ucsd.edu
University of California, San Diego
La Jolla, California, USA

Abstract

Quantum Error Correction (QEC) codes are essential for achieving fault-tolerant quantum computing (FTQC). However, their implementation faces significant challenges due to disparity between required dense qubit connectivity and sparse hardware architectures. Current approaches often either underutilize QEC circuit features or focus on manual designs tailored to specific codes and architectures, limiting their capability and generality. In response, we introduce QECC-Synth, an automated compiler for QEC code implementation that addresses these challenges. We leverage the ancilla bridge technique tailored to the requirements of QEC circuits and introduces a systematic classification of its design space flexibilities. We then formalize this problem using the MaxSAT framework to optimize these flexibilities. Evaluation shows that our method significantly outperforms existing methods while demonstrating broader applicability across diverse QEC codes and hardware architectures.

CCS Concepts: • **Computer systems organization → Quantum computing**; • **Hardware → Quantum error correction and fault tolerance**.

Keywords: Quantum computing, Quantum error correction, QEC circuit synthesis

ASPLOS '25, March 30–April 3, 2025, Rotterdam, Netherlands
© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0698-1/25/03.
https://doi.org/10.1145/3669940.3707236

ACM Reference Format:
Keyi Yin, Hezi Zhang, Xiang Fang, Yunong Shi, Travis S. Humble, Ang Li, and Yufei Ding. 2025. QECC-Synth: A Layout Synthesizer for Quantum Error Correction Codes on Sparse Hardware Architectures. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1 (ASPLOS '25), March 30–April 3, 2025, Rotterdam, Netherlands.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3669940.3707236

1 Introduction

Current quantum devices [18, 30, 47] suffer from errors, limiting their effectiveness in various applications [1, 6]. Quantum error correction (QEC) addresses this issue by encoding redundant physical qubits into logical qubits [46]. These logical qubits exhibit significantly lower error rates, ensuring the fidelity of quantum computations [3, 34]. Recent advancements in small-scale QEC implementations [2, 41, 51] demonstrate progress toward large-scale FTQC.

The core of QEC lies in the *error syndrome measurement (ESM)* circuits constructed from the QEC codes [9, 10, 12, 23, 32, 50, 54]. A typical ESM circuit (Fig. 1(a)) includes an syndrome qubit (red) and multiple data qubits (blue). Controlled gates (yellow) entangle the syndrome qubit with the data qubits, allowing errors on the data qubits to affect the syndrome qubit's state and flip its measurement outcome, indicating error occurrence. Hundreds or even more such ESM circuits are executed in a QEC cycle to gather error information across the logical qubit spanning numerous data qubits. For example, a typical distance-25 surface code has 625 data qubits per logical qubit and needs 624 such ESM circuits in a QEC cycle. This QEC cycle repeats during the runtime to safeguard the quantum system against errors.

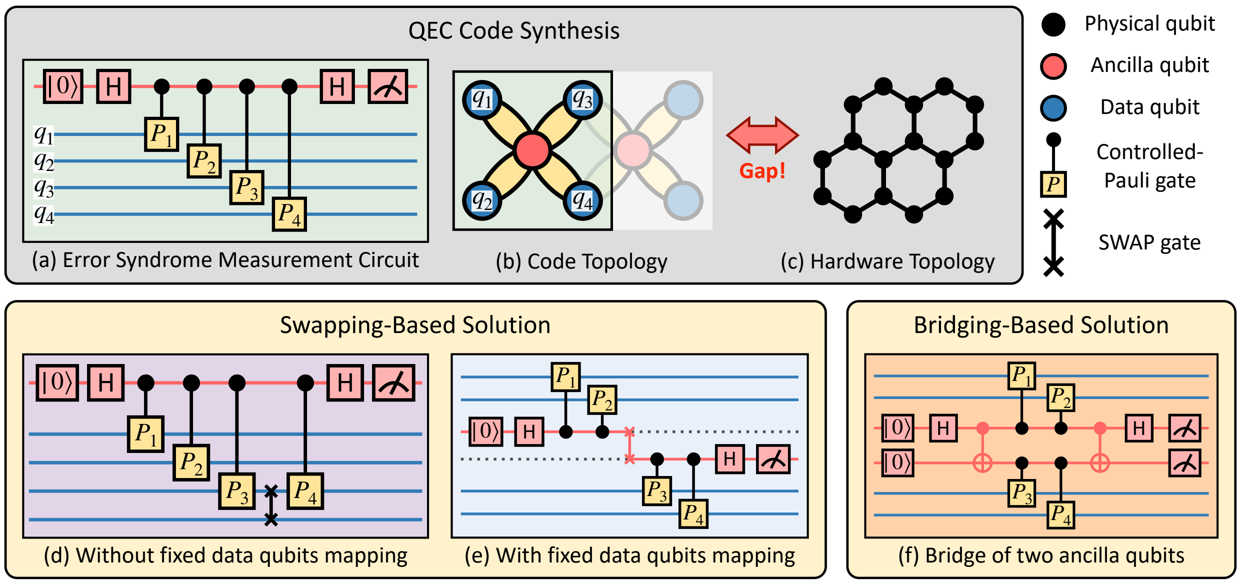


Figure 1. Overview of QEC code synthesis. (a) A ESM circuit for error syndrome extraction. (b) The code topology associated with the ESM circuits. (c) A quantum hardware with sparse architecture. (d) Implementation of swapping-based method. (e) Implementation of swapping-based method with fixed data qubits. (f) Implementation of bridging method.

However, implementing these circuits on current hardware presents a great challenge due to the mismatch between code topology and hardware architecture [4, 14, 48, 61, 64]. For instance, surface codes [23], a well-studied QEC code, requires a qubit connectivity of degree 4 (Fig. 1(b)) to enable interactions between the syndrome qubit (red) and four data qubits (blue) in their ESM circuits (Fig. 1(a)). In contrast, current hardware (Fig. 1(c)) typically has sparser architectures with connectivity degrees of 2 or 3 [2, 6, 24, 43, 49, 56] due to technical constraints [11, 28, 40]. Given the variety of QEC codes and hardware architecture, this gap is expected to persist in the foreseeable future.

A natural approach to tackle this issue involves *swapping-based* methods used in qubit mapping and routing (QMR) problems [38, 42, 58]. They insert SWAP gates into circuits to route distant qubits adjacent for interaction. For example, in Fig. 1(a), to execute a controlled-P_4 gate between the syndrome qubit and distant data qubit q_4, a SWAP gate is used to relocate q_4 to q_3's position. However, altering qubit locations with these methods is unsuitable for ESM circuits because the QEC process requires consistent data qubit positions for repeated execution [26]. A potential solution is to enforce fixed data qubit locations as in Fig. 1(e). However, achieving this reliably with current methods may require exponentially increasing attempts as QEC circuit complexity grows.

Beyond swapping-based methods, theoretical research [13, 14, 16, 36] introduces ancilla qubits in circuit mapping to form an "ancilla bridge" that connects data qubits (marked red in Fig. 1(f)), termed *bridging* methods. Compared to swapping-based methods, they fix the qubit locations and use fewer CNOT gates (2 CNOTs compared to 3 CNOTs in a SWAP gate). However, current research primarily focuses on theoretical aspects and has made limited attempts only on manual designs targeting specific codes and architectures, mostly for small-scale problems. For instance, [36] manually implement the 7-qubit Steane code on a square lattice, while heuristic methods [14, 64] adapt the surface code [23] to regular lattices like heavy-hexagon [14] up to distance 5. These specific manual designs lack adaptability to newer QEC codes like qLDPC codes [35, 60] and future hardware architectures. Moreover, these methods assume ideal hardware structures and struggle with variations caused by defective qubits [7, 52, 53], an issue gaining recent attention [39, 57].

We identified a crucial oversight in previous bridging methods: neglecting the broad design space enabled by ancilla qubits. Theoretically, bridging methods allow for diverse ancilla bridge shapes and sizes, various ancilla sharing designs, and flexible gate scheduling (Section 3). However, current methods rely on heuristic and manual designs, missing out on these flexibilities. This severely restricts the range of potential solutions, essentially limiting their generality.

Based on this insight, we introduce QECC-Synth, the first automated compiler for mapping ESM circuits using bridging methods applicable to diverse QEC codes and hardware architectures. To achieve this, we for the first time systematically categorize the flexibility of bridging methods into two key aspects: (1) ancilla bridge and (2) circuit optimization (Section 3). We then identify two critical metrics for selecting optimal solutions related to these flexibilities: (1) extra

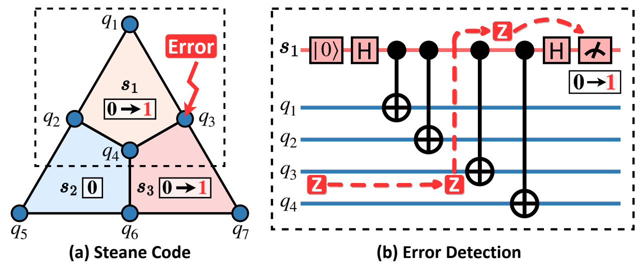


Figure 2. (a) The stabilizer structure and error syndromes of the Steane code [54]. (b) ESM circuit for the stabilizer $X_1X_2X_3X_4$ with a single syndrome qubit. A Z-error on q_3 will flip the measurement result of the syndrome qubit.

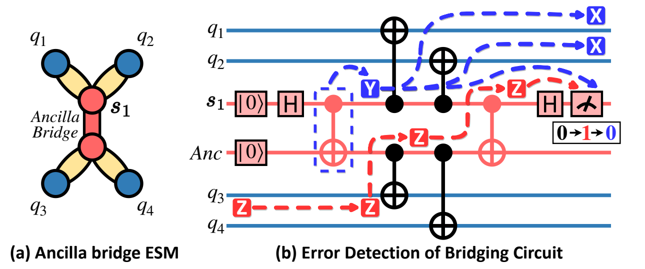


Figure 3. Bridging method. (a) The connectivity of the bridging-based ESM circuit. (b) Red line: This construction maintains error detection. Blue line: The inserted CNOT gate may introduce more error leading to error syndromes.

CNOT gates in ancilla bridge construction and (2) total circuit depth. Minimizing these metrics is crucial for reducing physical errors and decoherence, thereby improving circuit fidelity. Therefore, we formalize the ESM circuit mapping into two optimization problems: *code topology mapping* and *gate scheduling* (Sections 5.1 and 5.2), targeting the above two metrics. By translating code structures, hardware architectures, and mapping/scheduling conditions into Boolean constraints, we frame these optimization problems into MaxSAT framework, solvable by SAT solvers [8]. Importantly, our unified approach accommodates diverse codes and hardware setups, including systems with defective qubits. Furthermore, to handle large-scale problems, we introduce a heuristic algorithm that partitions them into manageable sub-problems addressed by our MaxSAT framework (Section 5.3).

We evaluate our method against three baselines: (1) Sabre (heuristic, swapping-based) [38], (2) SATmap (solver-based, swapping-based) [42], and (3) Surf-Stitch (heuristic, bridging-based, targeting surface codes) [64]. Our method achieves an average reduction of 74.9% and 26.5% in extra CNOT gate counts, and 34.7% and 55.5% in circuit depth compared to Sabre and SATmap, respectively. Our method also outperforms the surface-code-specific approach Surf-Stitch with 10% fewer extra CNOT gates and 38.5% shallower circuits. Additionally, these baselines frequently fail to provide solutions for special QEC codes and hardware setups and exceed memory limits for large-scale problems (Section 6). In contrast, our approach consistently identifies viable solutions.

Overall, we make the following contributions:

- We introduce QECC-Synth, a bridging-based compiler for efficiently mapping ESM circuits, supporting diverse QEC codes and hardware architectures.
- We pioneer the exploration of the expansive design space of bridging methods and formalize ESM circuit mapping as a two-stage MaxSAT problem.
- We propose a heuristic approach based on our MaxSAT framework to effectively handle scalability challenges with large-scale codes.
- Our evaluation shows that QECC-Synth outperforms existing swapping-based and bridging-based methods across various QEC codes and hardware setups, while also demonstrating good scalability.

2 Background

In this section, we provide essential background information on ESM circuits implementation.

2.1 ESM Circuits for Stabilizer Codes

We focus on *stabilizer codes* [21, 46] that encompass most extensively studied QEC codes [23, 33, 55].

Stabilizer codes. Stabilizer codes are characterized by a set of *stabilizers*, which are Pauli strings of the form $P_1P_2 \dots P_m$, where $P_i \in \{I, X, Y, Z\}$ denotes a Pauli operator acting on qubit q_i. For instance, Fig. 2(a) gives a stabilizer code with stabilizers $\{s_1 = X_1X_2X_3X_4, s_2 = X_2X_4X_5X_6, s_3 = X_3X_4X_6X_7\}$. Each stabilizer collects error information on its involved qubits, and combining this information from all stabilizers helps to deduce the type and location of errors, facilitating appropriate corrections. In practice, this process of error information collection involves implementing a dedicated *error syndrome extraction (ESM)* circuit for each stabilizer.

ESM circuits. The ESM circuit of a stabilizer comprises a syndrome qubit (red) and several data qubits (blue), connected by controlled gates specified by the stabilizer. The syndrome qubit is initially set to the $|0\rangle$ state and is measured at the end. If there are no errors on the data qubits, the measurement outcome is still 0. Conversely, certain errors on the data qubits will propagate through the controlled gate to the syndrome qubit, flipping its measurement outcome from 0 to 1, indicating error occurrence. For instance, as shown by the red dashed line in Fig. 2(b), a Z-error on qubit q_3 propagates through the CNOT gate and flipsthe measurement outcome of s_1. Notably, stabilizers often share data qubits, which introduces complexity to the mapping and scheduling of their ESM circuits (more details in Section 3).

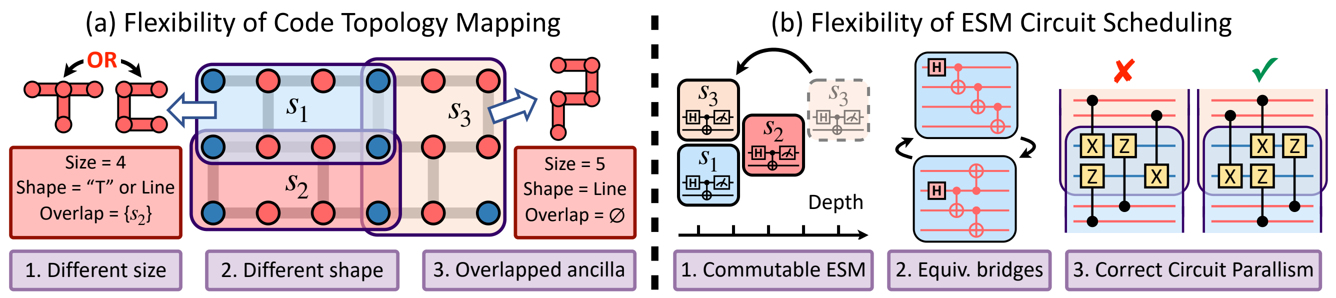


Figure 4. Flexibility of the bridging method for ESM circuits synthesis. (a) Code Topology Mapping: The ancilla bridges of different stabilizer may have different size, shape and may share ancilla qubits. (b) Circuit Scheduling: The scheduling need to consider ESM commutativity, equivalent state preparation on ancilla bridges and correct circuit parallelism.

Error detection and correction. During a QEC round, each ESM circuit is executed once. Their measurement results consist *error syndromes* that guide error correction. For instance, in Fig. 2(a), the error syndrome $(s_1, s_2, s_3) = (1, 0, 1)$ suggests a Z-error on qubit q_3, which can be corrected by applying a Z gate to q_3 since $Z^2 = I$. Crucially, all ESM circuits can theoretically operate in parallel within a QEC round due to the commutativity of stabilizers, allowing for a larger optimization space for scheduling (more details in Section 3).

2.2 Bridging method

ESM circuits necessitate interactions between the syndrome qubit and all data qubits, which is challenging with hardware of limited connectivity. The *bridging method* [36] offers a promising solution by extending the syndrome qubit with additional ancilla qubits called *bridge qubits*, forming an "ancilla bridge" that connects distant data qubits. This approach resolves connectivity issues while maintaining error syndrome extraction as in the original ESM circuit.

Mitigating connectivity issue. In bridging method, the ancilla bridge is prepared in a special entangled state [27] by inserting additional CNOT gates (highlighted in red). This state ensures that the controlled gates acting on the syndrome qubit can be transferred to the bridge qubit while maintaining circuit equivalence. As a result, the connectivity requirements, originally concentrated on the single syndrome qubit, are now distributed across multiple bridge qubits, thereby reducing the overall connectivity demand. For instance, the circuit with a length-2 ancilla bridge in Fig. 3(b) only requires each qubit to connect to at most 3 others, compared to 4 in Fig. 2(b), allowing for a direct mapping to current sparse quantum architecture [6, 24, 43, 49].

Error syndrome extraction. Remarkably, the ancilla bridge construction ensures that errors on the data qubits trigger a flip in the measurement result of the syndrome qubit, just as in the original ESM circuit (Fig. 2(b)). This occurs because errors can still propagate to the syndrome qubit through the special entangled state of the ancilla bridge (GHZ state). For instance, in Fig. 3(b), a Z-error on q_3 still flips the measurement result of the syndrome qubit s_1 (as shown by the red dashed line). Consequently, the transformed ESM circuit with bridge qubits preserves the same error detection and correction capabilities as the original circuit.

Impact of extra CNOT gates. Bridging methods introduce additional CNOT gates (highlighted in red) compared to the original ESM circuit (Fig. 2(b)). Similar to the extra CNOT gates in SWAP-based methods, these CNOTs can introduce errors to data qubits, potentially increasing the logical error rate. For example, in Fig. 3(b), if a Y-error occurs due to an introduced CNOT gate (within the block), it can induce a correlated two-qubit XX error on data qubits and affect the accuracy of the syndrome measurement. Therefore, minimizing the number of CNOT gates is crucial to mitigate these detrimental effects.

3 Motivating Examples

In this section, we provide examples to illustrate the vast design space of bridging-based ESM circuit mapping and scheduling. We highlight two main flexibilities: (1) varying ancilla bridges to optimize qubit mapping with fewer CNOT gates, and (2) leveraging circuit commutativity and equivalence to optimize gate scheduling for reduced circuit depth.

3.1 Flexibility of Code Topology Mapping

Ancilla bridges offer significant flexibility, particularly in terms of *(1) various shapes and sizes* and *(2) shared ancilla qubits*. This flexibility helps to find an effective mapping of the stabilizer code's topology onto the architecture, including the data qubit mapping and the ancilla bridge allocation. Moreover, varying constructions can introduce different numbers of CNOT gates (Section 2), impacting the fidelity of the synthesized ESM circuits. Since the total size of the ancilla bridge linearly correlates with the number of extra CNOT gates, we prioritize minimizing it as our optimization goal in code topology mapping (Section 5).

(1) Various shapes and sizes. The flexibility of ancilla bridges extends to their shapes and sizes, which stems from their operational mechanism. As long as the ancilla bridge is initialized in an entangled state, errors on data qubits can propagate to the syndrome qubit regardless of which bridge qubit they are connected to. Thus, the circuit can extract the error syndrome, irrespective of the ancilla bridge's shape or size. Consequently, the shape and size of the ancilla bridge can vary depending on the allocation of data qubits. For example, as shown in Fig. 4(a), both the 4-qubit and 5-qubit ancilla bridges for s_1 and s_3 can effectively fulfill their function, demonstrating this flexibility. For more details, we recommend referring to Lao et al. [36].

(2) Ancilla qubit sharing. Ancilla bridges for different stabilizers can share the same ancilla qubits, reducing the number of ancilla qubits required to be connected to a data qubit. For example, in Fig. 4(a), stabilizers s_1 and s_2 share two ancilla qubits in the middle row. Although this design saves qubits, ESM circuits sharing ancilla qubits cannot be executed simultaneously due to resource conflicts. This presents a challenge for gate scheduling to optimize parallel executions.

3.2 Flexibility of ESM Circuit Scheduling

The distinctive structures of ESM circuits allow for more flexible optimizations in scheduling compared to the gate-level scheduling for general quantum circuits. This flexibility arises mainly from three aspects: *(1) commutativity of stabilizer measurements*, *(2) equivalent state preparations on ancilla bridges* and *(3) parallel ESM circuits.* Leveraging this flexibility, our goal is to identify a correct scheduling while minimizing the circuit depth.

(1) Commutable stabilizer measurements. Implementing a QEC code involves executing ESM circuit blocks (Fig. 2(b)) for all stabilizers. Theoretically, these ESM circuit blocks for various stabilizers commute, allowing for arbitrary permutations in their order. As illustrated in Fig. 4(b1), sequentially executing stabilizer blocks s_1, s_2, and s_3 can be optimized by moving s_3 to the front and executing it in parallel with s_1, leading to a substantial decrease in circuit depth.

(2) Equivalent state preparations on ancilla bridges. As explained in Section 2, the ancilla bridge is initialized in an appropriate entangled state. This initialization can be accomplished through various circuits and some of which may offer smaller circuit depths [27]. For instance, in Fig. 4(b2), both circuits can prepare the 4-qubit initial state on the ancilla bridge of s_1, but the second one has a smaller circuit depth.

(3) Parallel ESM circuits. Although two ESM circuits can be executed in parallel as long as their ancilla bridges do not overlap, the overlap on data qubits makes it impossible to arbitrarily order the controlled gates while still ensuring a correct error syndrome extraction. To illustrate, as depicted in Fig. 4(b3), the simultaneous measurement of two stabilizers Z_1Z_2 and X_1X_2, which share two data qubits, poses a challenge for the scheduling of their controlled-Z and controlled-X gates. While the second circuit provides correct measurement results, the first one can only yield random outcomes [23]. This necessity for correctness adds intricacy to scheduling and must be addressed accordingly.

4 Overview of QECC-Synth

This section outlines the workflow of our QECC-Synth framework. It splits into two paths based on problem size, as shown in Fig. 5. For small-scale problems, QECC-Synth utilizes our dedicated two-stage SAT solver (Fig. 5, left frame) to generate optimal ESM circuit implementations. For large-scale problems, we employ the relaxation technique of *problem partitioning* to resolve the scalability issue (Fig. 5, right).

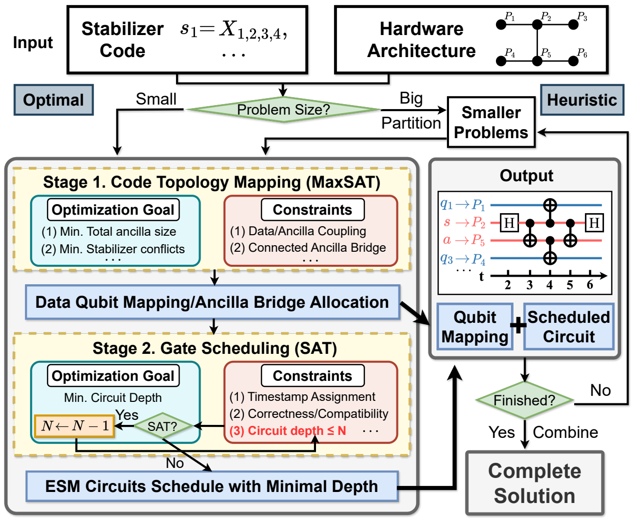

Input
Stabilizer Code $s_1 = X_{1,2,3,4}, \dots$
Hardware Architecture
P_1 P_2 P_3 P_4 P_5 P_6
Optimal
Small
Problem Size?
Big
Partition
Smaller Problems
Heuristic
Stage 1. Code Topology Mapping (MaxSAT)
Optimization Goal
(1) Min. Total ancilla size
(2) Min. Stabilizer conflicts
...
Constraints
(1) Data/Ancilla Coupling
(2) Connected Ancilla Bridge
...
Data Qubit Mapping/Ancilla Bridge Allocation
Stage 2. Gate Scheduling (SAT)
Optimization Goal
Min. Circuit Depth
$N \leftarrow N-1$
Yes
SAT?
No
Constraints
(1) Timestamp Assignment
(2) Correctness/Compatibility
(3) Circuit depth ≤ N ...
ESM Circuits Schedule with Minimal Depth
Output
$q_1 \rightarrow P_1$
$s \rightarrow P_2$
$a \rightarrow P_5$
$q_3 \rightarrow P_4$
H
t 2 3 4 5 6
Qubit Mapping
Scheduled Circuit
Finished?
No
Yes
Combine
Complete Solution

Figure 5. Overview of QECC-Synth.

4.1 Constraint-based Approach: Two-stage SAT

The left frame in Fig. 5 represents the central component of the QECC-Synth framework. It takes the stabilizer code and hardware architecture as inputs and generates the ESM circuit implementation, specifying both *qubit allocation* and *scheduled circuit.* Building on the two flexibility categories identified in Section 3, we break down this problem into two stages of constraint-based optimization: *Stage 1. Code topology mapping* and *Stage 2. ESM circuit scheduling.*

Stage 1. Code Topology Mapping. This stage aims to map data qubits and allocate ancilla bridges based on the stabilizer code and hardware architecture. The code is represented by its stabilizer set $Stab = \{s_1, s_2, \cdots\}$, where s_i are Pauli strings acting on data qubits $Data = \{q_1, q_2, \cdots\}$, while the hardware architecture is represented by its coupling graph, with vertices being physical qubits $Phys = \{P_1, P_2 \cdots\}$ and the edges $Edges = \{e_1, e_2, \cdots\}$ specifying their connectivity. The output consists of: *(1) Data qubit mapping* $\Pi : Data \rightarrow Phys$ that map data qubits to physical qubits on hardware, *(2) Ancilla bridge* $Anc : Stabs \rightarrow$ Subsets of $Phys$ that allocate an ancilla bridge for each stabilizer, and *(3) Coupling relation*

$CP : Stabs \times Data \to Phys$ that specifies the ancilla qubit to which each data qubit connects in each stabilizer.

We address this problem using the MaxSAT framework. The validity of the output is ensured by various *constraints* (red frame), while optimality is pursued by maximizing an objective function composed of a weighted sum involving multiple optimization goals (blue frame). For instance, we minimize the overall size of ancilla bridges, which directly impacts the number of introduced CNOT gates (Section 3), and maximize the number of compatible stabilizers to enable parallel execution. The corresponding objective function is:

$$-w_1 \cdot \# \begin{Bmatrix} \text{Ancilla} \\ \text{qubits} \end{Bmatrix} + w_2 \cdot \# \begin{Bmatrix} \text{Compatible anc.} \\ \text{bridge pairs} \end{Bmatrix}$$

Adjusting the weights w_1, w_2 alters the priority of optimization goals. The detailed MaxSAT encoding of constraints and objective function will be elaborated in Section 5.

Stage 2. Gate Scheduling. This stage aims to find a gate scheduling with minimal circuit depth, based on the qubit allocation determined in Stage 1. The output is a mapping $Sche : Circuits \to \{1, 2, \cdots\}$ that assigns each gate $g \in Circuits$ a time step for execution. The search process is encoded into a SAT solver, with constraints (detailed in Section 5) specified to ensure scheduling validity. To minimize circuit depth, we iteratively query the SAT solver with decreasing requirements on circuit depth until no solutions are found. This iterative approach helps obtain increasingly better solutions with smaller circuit depths, finally leading to a solution with minimal depth.

The outputs of the above two stages together constitute the complete output of our framework, specifying the mapping of data qubits and ancilla bridges, as well as the time schedule of each gate in the ESM circuit.

4.2 Heuristic Approach: Code Partitioning

The optimal approach described in Section 4.1 faces challenges when applied to large-scale problems due to the escalating number of constraints. To address this scalability concern, we partition the stabilizer set S of a code into smaller subsets $S_1, \cdots, S_k$ and leverage our two-stage optimal solver described in Section 4.1 to find their implementations. This yields optimized ESM circuits C_i for each subset S_i. Subsequently, we sequentially execute these ESM circuits for sub-problems to generate the complete solution for the entire stabilizer code. SWAP gates are inserted between adjacent ESM circuits to reposition the qubits from the previous circuit C_i to meet the requirements of the next one C_{i+1} (detailed in Section 5). The evaluation (Section 6) shows that our approach enables scalability while yielding favorable results.

5 Constraint-Based Code Synthesis

In this section, we delve into the technical aspects of SAT encoding for the two stages in QECC-Synth: *code topology mapping* and *ESM circuit scheduling*.

5.1 Stage 1: Code Topology Mapping via MaxSAT

In this stage, we encode the mapping of data qubits and the allocation of ancilla bridges into a MaxSAT problem. We formulate the qubit location requirements as *hard constraints* and the objectives of minimizing the total ancilla bridge size and incompatible stabilizer count as *soft constraints*.

5.1.1 Hard Constraints:

We list a few hard constraints that ensures the validity of qubit allocation.

Hard A. Basic Requirements. A legitimate qubit allocation should satisfy the following basic requirements.
(1) Each data qubit is mapped to a unique physical qubit:

$$\sum_{p \in Phys} map(q, p) = 1, \text{ for } q \in Data$$
$$\sum_{q \in Data} map(q, p) \le 1, \text{ for } p \in Phys$$

where $map(q, p) = True$ if data qubit q is mapped to physical qubit p, and $map(q, p) = False$ otherwise.
(2) A physical qubit cannot serve as both a data qubit and an ancilla qubit simultaneously:

$$\neg map(q, p) \bigvee \neg anc(s, p), \text{ for } (q,\ s) \in Data \times Stabs$$

where $anc(s, p) = True$ if physical qubit p is an ancilla qubit for stabilizer s, and $anc(s, p) = False$ otherwise.

Hard B. **Connectivity of ancilla bridge.** The ancilla bridge for each stabilizer needs to form a connected graph. To ensure this, we employ breadth-first traversal (BFT) [37]. All physical qubits in the connected $Graph[Anc[s]]$ must be visited within a specified parameter $L[s]$, representing the maximum size of the ancilla bridge $Anc[s]$ for stabilizer s. We introduce auxiliary variables $v_{s,p}(p_i, t)$ to track whether physical qubit p_i has been visited after the t-th round ($t \in \{1, \ldots, L[s]\}$). In the initial round, we ensure that the starting vertex is visited:

$$v_{s,p}(p, 1) \bigwedge_{p_i \neq p} \neg v_{s,p}(p_i, 1)$$

In subsequent rounds, an unvisited physical qubit p_i in the ancilla bridge $Anc[s]$ is visited if any of its adjacent qubits p_j have been visited in the previous rounds:

$$v_{s,p}(p_i, t) \to\ anc(s, p_i)$$
$$\wedge \left[v_{s,p}(p_i, t-1) \bigvee_{p_j \in Adj(p_i)} v_{s,p}(p_j, t-1) \right]$$

To ensure that the traversal starts from any possible physical qubit, we enforce constraints for each $p \in Phys$:

$$anc(s,p) \to \bigwedge_{p_i \in Phys} (\neg anc(s,p_i) \vee v_{s,p}(p_i, L[s]))$$

The number of variables and the conjunctive normal form (cnf) clause used for each stabilizer are both $O(L[s] * |Stabs|^2)$.

***Hard C.* Coupling between data and ancilla qubits.** Each data qubit for stabilizer s must be directly coupled to some ancilla qubit in its ancilla bridge $Anc[s]$:
(1) Exactly one ancilla qubit p is coupled with data qubit q:

$$\sum_{p \in Phys} cp(s,q,p) = 1, \text{ for } q \in Data[s]$$

(2) The coupling relation $CP(s,q) = p_i$ requires q to be mapped to some qubit p_i adjacent to p:

$$cp(s,q,p) \to \bigvee_{p_i \in Adj(p)} map(q,p_i) \bigwedge anc(s,p)$$

5.1.2 Soft Constraints: We introduce several soft constraints to optimize the performance of ESM circuit implementation. Each soft clause is assigned a non-negative weight indicating its priority, and a MaxSAT solver is used to obtain a solution that maximizes the sum of the weights of all satisfied soft clauses.

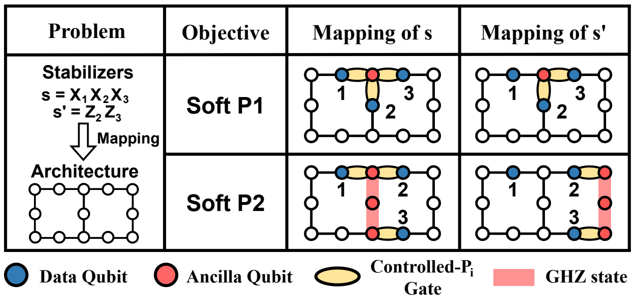


Figure 6. Different mappings for stabilizer code $\{s, s'\}$ with different optimization priority.

***Soft P1.* Minimizing the total ancilla bridge size.** Reducing the total ancilla bridge size compresses the ESM circuit and minimizes the introduced extra CNOT gates, enhancing circuit fidelity. This is achieved by maximizing the number of Boolean variables $anc(s,p)$ set to *False*. Hence, we add the following soft constraints with weight $P1$:

$$\neg anc(s,p), \text{ for } (s,p) \in Stabs \times Phys$$

***Soft P2.* Mitigating the stabilizer conflicts.** Stabilizers without conflicts in ancilla qubits can be efficiently executed in parallel, termed as *compatible stabilizers*. Our aim is to maximize the count of compatible ancilla bridge pairs. Thus, we introduce the following soft constraints with weight $P2$:

$$\bigwedge_{p \in Phys} \neg anc(s,p) \vee \neg anc(s',p), \text{ for } s, s' \in Stabs$$

Fig. 6 illustrates how different optimization objectives impact the mappings of the same code $Stabs = X_1X_2X_3, Z_2Z_3$ onto a heavy square architecture. In the first row, Soft P1 is the primary objective, with weight P_1 notably greater than P_2. Consequently, the resulting mapping features an ancilla bridge with size 1, yet two stabilizers share the same ancilla qubit. Conversely, in the second row, Soft P1 is prioritized, resulting in a mapping without conflicts in data qubits, albeit with a larger ancilla bridge size. In practice (Section 6), we prioritize Soft P1 over Soft P2, given its more significant impact on the overall error correction performance.

5.2 Stage2: ESM Circuit Scheduling via SAT

This stage encodes the gate scheduling of ESM circuits into a SAT problem. We introduce additional Boolean variables to characterize all operations within the circuits (Section 5.2.1). These variables help to formulate constraints based on dependencies between operations, ensuring the validity of their scheduling (Section 5.2.2). Finally, we iteratively query the SAT solver to achieve a scheduling with minimal circuit depth.

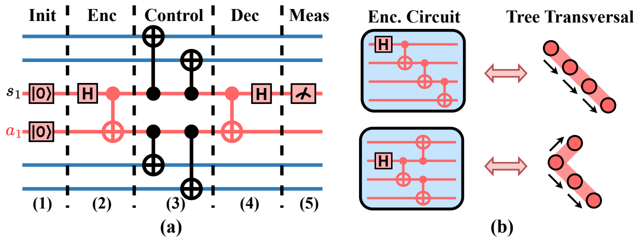


Figure 7. (a) Five stages of a typical ESM circuit with ancilla bridge. (b) Equivalence between encoding (decoding) circuit for ancilla bridge and tree transversal.

5.2.1 Operations within ESM Circuits: A ESM circuit with ancilla bridge typically comprises five phases: *(1) Initialization*, *(2) Encoding*, *(3) Control*, *(4) Decoding*, and *(5) Measurement*, as shown in Fig. 7(a). Operations in Phases (1)(3)(5) are determined by the output of Stage 1, which includes qubit allocation and coupling relations between ancilla and data qubits. Specifically, qubit allocation specifies all ancilla qubits initialized in the $|0\rangle$ state, with the syndrome qubit measured at the end, constituting Phases (1)(5). The coupling relation precisely determines all CNOT gates in Phase (3). To characterize Phases (2)(4), we introduce the following Boolean variables:

❶ $enc(s,p,p')$ and $dec(s,p,p')$: The variable $enc(s,p,p')$ is set to *True* if there is a *CNOT* gate from p to p' in the *Encoding* phase for stabilizer s, and *False* otherwise. Similar

definitions apply to $dec(s, p, p')$. Additionally, a Hadamard gate is applied to the syndrome qubit. These Boolean variables collectively characterize the operations in Phases (2)(4).
❷ $time(g, t)$: This denotes the time assignment of operations $g \in Circuits$. If the operation g is executed at time t, $time(g, t)$ is set to $True$; otherwise, it is set to $False$.

5.2.2 Boolean Constraints for Gate Scheduling:

This section outlines the constraints that must be satisfied by a valid gate scheduling, using the variables defined above.

Hard A. Time Assignment Constraints. A valid gate scheduling should satisfy the following basic requirements regarding time assignments. We define T as the maximum allowable circuit depth for the search.
(1) All gates should be executed exactly once:

$$\sum_t time(g, t) = 1, \text{ for } g \in Circuits$$

(2) Two operations g and g' act on the same physical qubit p cannot be executed simultaneously. In other words, at most one operation acting on p can be executed at time t:

$$\sum_{g \in Act[p]} time(g, t) \le 1, \text{ for } t \in \{1, ..., T\}$$

where $Act[p]$ denotes all the operations acting on p.

Hard B. **Valid** $Enc[s]$ **and** $Dec[s]$ **circuits.** The circuits in the *Encoding* and *Decoding* phases should effectively encode and decode the GHZ state for the ancilla bridge. We observe that the encoding circuit's formulation is analogous to a tree traversal, where the syndrome qubit serves as the root with an H gate acting on it, and other ancilla qubits are visited via CNOT links (Fig. 7(b)). The decoding circuit can be viewed as the mirror image of the encoding circuit. With this understanding, we establish the following constraints:
(1) The transversal tree must have exactly one root:

$$\sum_{p \in Anc[s]} enc(s, p, p) = 1$$

where $enc(s, p, p)$ is set to $True$ if an H gate acts on physical qubit p, and $False$ otherwise.
(2) Each node within the transversal tree is visited exactly once, ensuring that every ancilla qubit $p \in Anc[s]$ is subjected to exactly one $CNOT$ gate targeting it:

$$\sum_{p_i \in Anc[s]} enc(s, p_i, p) = 1$$

(3) The control qubit must be prepared before its associated CNOT gate is executed. Specifically, if $g_j = CNOT(p_i, p_j)$ and $g_i = CNOT(p_k, p_i)$, then g_j must be executed after g_i:

$$enc(s, p_i, p_j) \to \left(t_{g_i} < t_{g_j}\right), \text{ for } p_i \ne p_j \in Anc[s]$$

Hard C. **Correct order of** $Ctrl[s]$ **circuits**. The anticommuting controlled-P_i gates in the *Control* phase should satisfy a particular order to ensure the correctness of the circuit. Specifically, the order is correct if and only if the count of anti-commuting gate pairs (c, c') satisfying $t_c < t_{c'}$ is an even number. This requirement is encoded into the constraint:

$$\bigoplus_{Anti(c,c')} (t_c < t_c') = 0, \text{ for } c \in Ctrl[s] \text{ and } c' \in Ctrl[s']$$

where $\bigoplus$ means the mod 2 addition and $Anti(c, c') = True$ means c and c' anti-commute.

Hard D. **Commutativity of incompatible stabilizers.** Two stabilizer circuits s and s' sharing ancilla qubits cannot he executed simultaneously. This constrain can be realized by requiring $Init[s]$ after $Meas[s']$, $Init[s']$ after $Meas[s]$:

$$\left[\bigwedge_{p \in Share} (t_{Mz'[p]} < t_{Init[p]})\right] \bigvee \left[\bigwedge_{p \in Share} (t_{Mz[p]} < t_{Init'[p]})\right]$$

where $Share = Anc[s] \cap Anc[s']$ represent all physical qubit that shared by two stabilizer's ancilla block.

5.3 Heuristic Approach

As previously discussed, the number of clauses increases exponentially with the number of stabilizers in a QEC code, creating a significant bottleneck in our framework. To address this challenge for large-scale QEC code synthesis, we propose a relaxation method that can still yield locally optimal solutions. Our approach involves partitioning the set of stabilizers and solving a SAT problem for each subset sequentially through our two-stage constraint-based approach. Finally, we insert *routing layers* between the these synthesized ESM circuits to integrate them into the original code.

Partitioning the stabilizer set. We can think of a stabilizer set S as a union of subsets, $S_0 \cup \cdots \cup S_k$. However, due to the commutativity of the stabilizers, there are millions of possible partition choices. Since each stabilizer can be viewed as a set of constraints for the data qubits to satisfy, we want the constraints for the same data qubits to be in the same partition, allowing them to be handled together. It means to minimize the connections or shared qubits between stabilizers from different subsets. As a result, we formulate this as a simple balanced graph partitioning problem in graph theory [5]. For example, in the Steane code shown in Fig. 2(a), we treat each stabilizer as a vertex. As stabilizer s_1 shares two data qubits, q_2 and q_4, with stabilizer s_2, two edges will be added to the graph. The balanced graph partitioning divides the vertices into k components of almost equal size, minimizing the capacity of edges between different components.

Sorting the SAT problems. After partitioning the stabilizer set, we can apply our two-stage SAT solver to synthesize the ESM circuit for each stabilizer subset. However, different resulting data qubit mappings can make it challenging to integrate them into a complete circuit for the original code. The mapping map_i for subset S_i should follow the mappings from map_0 to map_{i-1} as closely as possible, because stabilizers from different subsets may share the same data qubits, and we want these qubits to be mapped to the same positions in map_i. Therefore, after solving the SAT problem for $S_1, \ldots, S_i$, we select the subset S_j that shares the most data qubits with all data qubits involved in $S_1 \cup \cdots \cup S_i$ as the next SAT problem to solve. Moreover, we add soft constraints in Stage 1 to maximize the number of shared data qubits that remain in their previous positions:

$$\max |\{q \mid map_i(q,p) \vee map_{0\ldots i}(q,p)\}|$$

Integrating the ESM circuits. After all ESM circuits for each subset are generated, we connect them to form the complete ESM circuit for the original QEC code. Although the circuits may require different data qubit mappings and ancilla bridge allocations on the architecture, we insert integration layers between them to move the data qubits to meet these varying requirements. Since all ancilla qubits function identically and only data qubits contain the logical information, we only need to route the data qubits instead of all used physical qubits, as general QMR methods do. In the integration layer, to connect ESM circuits, we find the shortest path for each data qubit that changes its position and insert *SWAP* gates to move it to the new position.

6 IMPLEMENTATION AND EVALUATION

6.1 Experimental Setup

Evaluation setting: Our framework, QECC-Synth, utilizes the state-of-the-art MaxSAT solver, NuWLS-c [66], and the widely-used SAT solver, PySAT [31], with a solving time limit of 7,200 seconds and a memory limit of 128 GB. For logical error analysis, we used Stim [25], a widely-used QEC simulator for ESM circuits, and PyMatching [29], a error syndrome decoder that implements the Minimum Weight Perfect Matching (MWPM) algorithm.

Hardware architectures: We cover a diverse range of practical device architectures in our evaluation. As shown in Fig. 8, the selected architectures are sourced from the latest quantum machines, including Google's Sycamore [6], IBM's Osprey, and IBM Q Tokyo [18]. To represent the connectivity of the architecture, we use the average degree of the nodes, denoted as $Density(G) = 2|Edge|/|Node|$.

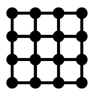
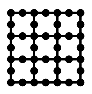
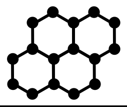
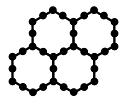
Name	Density	Arch Example	Name	Density	Arch Example
Square	4.00		Heavy Square	2.67	
Hexagon	3.00		Heavy Hexagon	2.40	

Figure 8. Overview of device architectures.

Stabilizer codes: Our evaluation covers both classical and recent popular stabilizer codes to demonstrate the generality of our framework. For instance, small-scale $[[9,1,3]]$ and large-scale $[[81,1,9]]$ surface code are selected to show support for the most popular surface code family. The 2D $[[16,4,3]]$ and the 3D $[[8,3,2]]$ color codes color codes are included to illustrate the influence of distinct code topologies. The classical $[[7,1,3]]$ Steane code is selected for its role as a basic block that can be merged into larger stabilizer codes [48], such as the $[[22,4,3]]$ and $[[12,2,3]]$ color codes. The HyperGraph Product (HGP) code [60] is chosen to demonstrate our framework's applicability to recent popular qLDPC codes. Moreover, we adopt some manually designed measurement schemes for specific codes to showcase the extensibility of our framework. Specifically, we adapt Shor's scheme [20] for the 3D color code and the Steane code to ensure the fault tolerance of the ESM circuits.

To quantify the level of interconnectivity for each stabilizer code, we define the density of a stabilizer code:

$$Density(C) = \frac{2\sum_{s\in Stabs} wt(s)}{|Stabs| + |Data|}$$

Error Model: We follow a widely-used circuit-level error model similar to those in [14, 25, 64], involving the same error rates and types. Specifically, we apply probabilities ranging from 10^{-3} to 10^{-2} to the single-qubit depolarizing error channel for single-qubit gates, the two-qubit depolarizing error channel for two-qubit gates, and the Pauli-X error channel for measurement and reset operations. Additionally, for idle errors induced by decoherence, each idle qubit undergoes a single-qubit depolarizing error channel per gate duration with a probability of 2×10^{-4} [1, 19, 64]. These errors affect all qubits, including both data and ancilla qubits.

Metrics: We consider two key metrics that significantly impact the fidelity of quantum circuit execution and are widely used to evaluate the performance of general QMR method s [38, 42, 58]. *(1) Extra* `CNOT` *gate count:* We evaluate the extra `CNOT` gate count required by ESM circuits to compensate for sparse architectures. Previous studies [23, 25] and our experiments in Section 6.4 show the high sensitivity of ESM circuits to two-qubit gate errors, making the `CNOT` gate count a crucial factor that significantly affects the fidelity of logical qubits. *(2) Circuit Depth:* We assess the Circuit Depth of entire ESM circuit, which is the maximum time coordinates of all gates. Due to the limitation of current quantum computing technology, physical qubits can only function well up to a short 'lifetime'. Thus, minimizing depth becomes crucial to ensure efficiency. A larger time-step count would introduce more decoherence errors that accumulate, eventually surpassing the fault tolerance threshold of the error correction procedure.

Table 1. Results of QECC-Synth and the baselines for various stabilizer codes across different device architectures.

Code	Density	Architecture	# qubit	# Extra CNOT gate			Circuit depth		
				QECC-Synth	SATmap	Sabre	QECC-Synth	SATmap	Sabre
[[9, 1, 3]] Surface Code	2.82	Square	25	0	0	30	8	17	16
		Hexagon	21	14	27*	90	15	34*	35
		H-Square	29	24	42*	90	14	49*	21
		H-Hexagon	46	56	66*	96	16	47*	26
[[16, 4, 3]] 2D Color Code	4.29	Square	35	20	168*	270	17	132*	56
		Hexagon	50	60	Time-Limit	252	18	Time-Limit	47
		H-Square	93	72	Time-Limit	468	26	Time-Limit	125
		H-Hexagon	115	180*	Time-Limit	1,038	30*	Time-Limit	324
[[8, 3, 2]] 3D Color Code	3.69	Square	30	0	Not Exist	Not Exist	20	Not Exist	Not Exist
		Hexagon	16	4	Not Exist	Not Exist	31	Not Exist	Not Exist
		H-Square	79	14	Time-Limit	270	30	Time-Limit	40
		H-Hexagon	55	38	Time-Limit	228	33	Time-Limit	35
[[7, 1, 3]] Steane Code	3.69	Square	25	0	Not Exist	Not Exist	17	Not Exist	Not Exist
		Hexagon	21	0	Not Exist	Not Exist	18	Not Exist	Not Exist
		H-Square	29	8	Not Exist	Not Exist	32	Not Exist	Not Exist
		H-Hexagon	46	36	Time-Limit	504	30	Time-Limit	112
HGP QLDPC Code	4.00	Square	121	588*	Time-Limit	726	70*	Time-Limit	205
		Hexagon	128	876*	Time-Limit	1,236	121*	Time-Limit	354
		H-Square	176	859*	Time-Limit	990	96*	Time-Limit	251
		H-Hexagon	265	1,275*	Time-Limit	1,386	120*	Time-Limit	336
[[81, 1, 9]] Surface Code	3.58	Square	289	0	Time-Limit	9,594	8	Time-Limit	2,770
		Hexagon	225	6,547*	Time-Limit	8,928	391*	Time-Limit	2,428
		H-Square	251	288	Time-Limit	9,654	15	Time-Limit	2,458
		H-Hexagon	387	9,322*	Time-Limit	11,064	450*	Time-Limit	2,930

6.2 Compared to Swapping-Based Approaches

We begin by comparing QECC-Synth with two traditional swapping-based approaches designed for the QMR task of general quantum circuits: SATmap (a constraint-based approach) [42] and Sabre (a heuristic approach) [38]. The results in Table 1 demonstrate that our QECC-Synth consistently synthesizes smaller and faster ESM circuits for stabilizer codes. We highlight the following three results:

(1) Our QECC-Synth shows great improvement even when the code synthesis task is small-scale. For instance, in the case of the [[9, 1, 3]] surface code, QECC-Synth demonstrates an average reduction of 26.5% in the extra CNOT gate count compared to SATmap and 74.9% compared to Sabre. Additionally, the improvement in circuit depth is even more substantial, with an average reduction of 55.5% compared to SATmap and 34.7% compared to Sabre. This highlights the efficiency of QECC-Synth's bridging-based approach over the traditional swapping-based approach in addressing connectivity disparity problems in ESM circuit synthesis.

(2) The "$*$" symbol and "Time-Limit" entry indicate that the SAT solver times out, returns a sub-optimal result, or does not produce a result. This often occurs when the code becomes complicated and large, such as the HGP qLDPC code, or when the "code-architecture" gap is substantial, such as mapping the dense [[16, 4, 3]] color code to sparse architectures (Hexagon, Heavy Square, and Heavy Hexagon). This demonstrates that QECC-Synth efficiently prunes the optimization space of ESM circuits by treating and mapping each stabilizer as a whole, rather than mapping each gate in the ESM circuit separately, as traditional QMR approaches do.

(3) The "Not-Exist" entry indicates that the approach verifies the impossibility of finding a workable code synthesis on a device with the given architecture and a limited number of physical qubits, as the number of physical qubits is always restricted on a practical quantum chiplet [2, 6, 24]. The inability of swapping-based approaches to leverage the flexibility of overlapping ancilla qubits contributes to the occurrence of "Not-Exist". These approaches treat data and ancilla qubits as identical, requiring each to be mapped to a distinct physical qubit, which may exceed the available number of physical qubits on the device. For instance, in the case of the [[8, 3, 2]] 3D Color Code and the [[7, 1, 3]] Steane Code, their original ESM circuits apply Shor's scheme and require more than one ancilla qubit for each stabilizer, resulting in a "Not-Exist" for the swapping-based approaches.

6.3 Compared to Heuristic Bridging-Based Approach

We compare our QECC-Synth with Surf-Stitch [64], a heuristic method that can only work on the **surface code** and **specific regular architectures**. In Table 2, we show the synthesis results of the $d = 5$ surface code on both perfect and defective architectures [7, 39, 57] that have a faulty qubit in the middle due to fabrication defects. The result shows that QECC-Synth has a wider range of applicability while achieving equal or better ESM circuits than Surf-Stitch. We highlight the following three results:

Table 2. Detailed information about synthesized distance-5 surface codes on perfect or defective architectures.

Architecture \ Metric		QECC-Synth		Surf-Stitch	
		# Extra CNOT gate	Circuit depth	# Extra CNOT gate	Circuit depth
Square	Perfect	0	8	0	8
	Defective	12	17	×	×
Hexagon	Perfect	72	20	120	26
	Defective	76	17	×	×
Heavy Square	Perfect	80	15	80	24
	Defective	94	20	×	×
Heavy Hexagon	Perfect	216	18	224	40
	Defective	216	23	×	×

(1) Although at most cases of perfect architectures, the improvement on CNOT gates number by QECC-Synth is marginal, as the Surf-Stitch's manual mapping of surface code is already near optimal, QECC-Synth still find a synthesis on hexagon with using 40% less extra CNOT gates. Moreover, due to our more concurrent scheduling strategy, QECC-Synth achieves a average reduction of 38.5% in circuit depth.

(2) In the presence of faulty physical qubits that cannot be used in ESM circuits due to fabrication defects [7, 39, 45], Surf-Stitch fails to operate because the architecture is no longer regular. However, our QECC-Synth adapts seamlessly to these defective architectures with only a minor increase in CNOT gates and circuit depth.

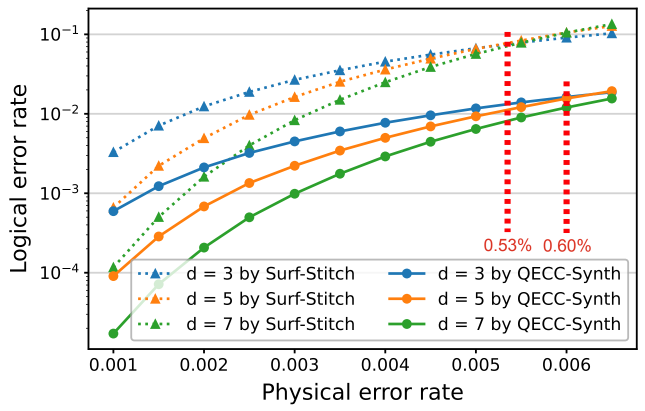


Figure 9. Logical error rate of distance-5 surface codes synthesized by QECC-Synth and Surf-Stitch on heavy-square architecture. The error threshold is the physical error rate at which the code curves of different distances intersect, which is represented with red dotted lines.

(3) For two synthesized ESM circuits with an equal count of CNOT gates (e.g. perfect heavy-square), circuit depth significantly affects their error correction capability. As shown in Fig. 9, we simulated the logical error rates for surface codes of different distances on the heavy-square architecture, each using 10^6 slots of simulation. The results for QECC-Synth consistently show significantly lower logical error rates, ranging from 6.8x to 8.6x lower for the distance-7 surface code, despite both approaches using the same number of CNOT gates. Moreover, surface codes synthesized by QECC-Synth also show an improved error threshold. Although the improvement from 0.53% to 0.60% may seem marginal, it significantly affects the scalability of surface codes. For instance, if we aim to implement a logical qubit on a device with a physical error rate of 0.50%, Surf-Stitch would require 10x more physical qubits to achieve the same logical error rate.

6.4 Ablation Study

To justify prioritizing "# extra CNOT gates" over "circuit depth" as the primary optimization goal and employing a two-stage SAT approach to optimize them separately, we perform a *breakdown analysis* of the logical error rate to identify the most critical factor affecting the fidelity of the ESM circuit. Furthermore, we conduct a *scalability analysis* to demonstrate the effectiveness of our SAT problem formalization.

Breakdown analysis of logical error rate. This study provides a breakdown analysis of how the error threshold shifts across two vital error types: two-qubit CNOT gate errors and device idle errors. The simulation focuses on surface codes on 2-D grid hardware, but the conclusion can be readily extrapolated to broader contexts. This analysis provides additional support for the rationale behind prioritizing the reduction of # CNOT gates (linked to accumulated CNOT errors) over circuit depth (linked to accumulated idle errors) when optimizing QEC code synthesis.

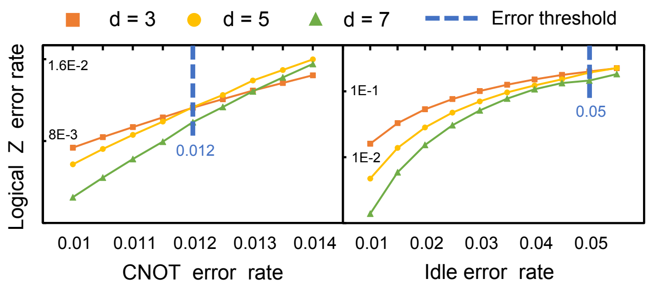


Figure 10. Breakdown analysis of the influence of CNOT gate errors and idle errors on the logical error rate.

We conducted 10^6 simulation slots for each logical error point. When analyzing the CNOT gate error, the idle error is set to 0, and vice versa. As shown in Fig. 10, pure CNOT gate error exhibits a per-step error threshold of 1.2%, while pure idle error demonstrates a threshold of 5%. The significantly lower threshold for CNOT errors highlights both the reduced tolerance for this type of error and its amplified impact on the overall logical error rate. This emphasizes the importance of prioritizing the reduction of the CNOT gate count.

Scalability Analysis. In this study, we evaluate the scalability of QECC-Synth in comparison to the solver-based SATmap and test our relaxation method by code partition on dense and large-scale stabilizer codes.

Table 3. Size of MaxSAT-encoded problem for [[9, 1, 3]] surface code synthesis by QECC-Synth and SATmap

Metrics \ Arch		Square	Hexagon	Heavy Square	Heavy Hexagon
# Variable	QECC-Synth	6.38×10^3	1.24×10^4	2.27×10^4	1.07×10^5
	SATmap	5.52×10^4	4.04×10^4	7.24×10^4	1.71×10^5
# Hard clause	QECC-Synth	1.51×10^4	2.64×10^4	4.76×10^4	2.18×10^5
	SATmap	1.80×10^6	9.98×10^5	1.83×10^6	4.22×10^6
# Soft clause	QECC-Synth	228	196	260	396
	SATmap	14, 400	10, 080	19, 488	49, 680

The sizes of the MaxSAT-encoded problem are shown in Table 3. It reveals that tasks from SATmap exhibit 1.5x to 8.5x more variables, 20x to 120x more hard constraints, and 50x to 125x more soft constraints than the tasks from QECC-Synth. This indicates that our QECC-Synth method offers a more efficient approach to encoding the QEC code mapping problem than SATmap, by formalizing the special structure characteristic of the measurement circuit in stabilizers.

Moreover, the synthesis results of the HGP qLDPC code and the $d = 9$ surface code in Table 1 further demonstrate the efficiency of QECC-Synth on dense and large-scale stabilizer codes using the heuristic relaxation method. It shows significant improvements in reducing extra CNOT gates and circuit depth compared to traditional swapping-based approaches.

7 Related Work

Swapping-based methods. Extensive research has addressed the challenge of mapping general quantum circuits onto hardware with limited connectivity. These methods typically fall into one of three categories: constraint-based [42, 44, 59, 63], heuristic [38, 65, 67], or a hybrid approach [22]. However, these techniques are inadequate for ESM circuits due to the inserted SWAP gates, which disrupt qubit locations and hinder parallel execution of stabilizer measurements with shared data qubits. Moreover, introducing SWAP gates can elevate the circuit's error rate, compromising the precision needed for effective ESM circuits.

Bridging-based methods. The original bridging-based methods [13, 15, 17, 36, 48] extend single ancilla qubits into ancilla bridges to alleviate the connectivity constraint. This approach offers fixed qubit locations and introduces fewer additional gates compared to swap-based methods, enhancing performance. However, these methods are designed for single ESM circuits and struggle with complex dependencies among stabilizer measurements within QEC codes. Recent efforts have applied bridging-based techniques to entire QEC codes [14, 62, 64], but they are limited to specific QEC codes and hardware architectures, lacking adaptability to broader settings. Furthermore, these approaches rely on heuristic design without optimality guarantees.

8 Conclusion

This paper introduces the first automated compilation framework for implementing ESM circuits using the bridging method. Our approach demonstrates broad applicability across diverse QEC codes and hardware architectures, including systems with defective qubits. By systematically classifying and leveraging the primary flexibilities of the bridging method, we effectively explore its extensive design space and formalize ESM circuit implementation as a two-stage MaxSAT problem solved by high-performance SAT solvers. Comparative evaluations show our method significantly outperforms existing swap-based and bridging approaches while achieving superior adaptability. These advancements represent a significant step toward realizing long-term fault-tolerant quantum computing goals.

Acknowledgments

We thank the anonymous reviewers for their valuable and constructive feedback. This work was supported in part by NSF 2138437, NSF 2048144, and the Robert N. Noyce Trust. This material is based upon work supported by the U.S. Department of Energy, Office of Science, National Quantum Information Science Research Centers, Quantum Science Center (QSC). This research used resources of the Oak Ridge Leadership Computing Facility (OLCF), which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725. This research used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility located at Lawrence Berkeley National Laboratory, operated under Contract No. DE-AC02-05CH11231.

References

[1] 2021. Exponential suppression of bit or phase errors with cyclic error correction. *Nature* 595, 7867 (2021), 383–387.
[2] 2023. Suppressing quantum errors by scaling a surface code logical qubit. *Nature* 614, 7949 (2023), 676–681.
[3] Dorit Aharonov and Michael Ben-Or. 1997. Fault-tolerant quantum computation with constant error. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*. 176–188.
[4] Panos Aliferis and Andrew W Cross. 2007. Subsystem fault tolerance with the Bacon-Shor code. *Physical review letters* 98, 22 (2007), 220502.
[5] Konstantin Andreev and Harald Räcke. 2004. Balanced graph partitioning. In *Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*. 120–124.
[6] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando GSL Brandao, David A Buell, et al. 2019. Quantum supremacy using a programmable superconducting processor. *Nature* 574, 7779 (2019), 505–510.

[7] James M Auger, Hussain Anwar, Mercedes Gimeno-Segovia, Thomas M Stace, and Dan E Browne. 2017. Fault-tolerance thresholds for the surface code with fabrication errors. *Physical Review A* 96, 4 (2017), 042316.

[8] Armin Biere, Marijn Heule, and Hans van Maaren. 2009. *Handbook of satisfiability*. Vol. 185. IOS press.

[9] Héctor Bombín and Miguel A Martin-Delgado. 2007. Optimal resources for topological two-dimensional stabilizer codes: Comparative study. *Physical Review A* 76, 1 (2007), 012305.

[10] Sergey B Bravyi and A Yu Kitaev. 1998. Quantum codes on a lattice with boundary. *arXiv preprint quant-ph/9811052* (1998).

[11] Markus Brink, Jerry M Chow, Jared Hertzberg, Easwar Magesan, and Sami Rosenblatt. 2018. Device challenges for near term superconducting quantum processors: frequency collisions. In *2018 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 6–1.

[12] A Robert Calderbank and Peter W Shor. 1996. Good quantum error-correcting codes exist. *Physical Review A* 54, 2 (1996), 1098.

[13] Christopher Chamberland and Michael E Beverland. 2018. Flag fault-tolerant error correction with arbitrary distance codes. *Quantum* 2 (2018), 53.

[14] Christopher Chamberland, Guanyu Zhu, Theodore J Yoder, Jared B Hertzberg, and Andrew W Cross. 2020. Topological and subsystem codes on low-degree graphs with flag qubits. *Physical Review X* 10, 1 (2020), 011022.

[15] Rui Chao and Ben W Reichardt. 2018. Fault-tolerant quantum computation with few qubits. *npj Quantum Information* 4, 1 (2018), 42.

[16] Rui Chao and Ben W Reichardt. 2018. Quantum error correction with only two extra qubits. *Physical review letters* 121, 5 (2018), 050502.

[17] Rui Chao and Ben W Reichardt. 2020. Flag fault-tolerant error correction for any stabilizer code. *PRX Quantum* 1, 1 (2020), 010302.

[18] Jerry Chow, Oliver Dial, and Jay Gambetta. 2021. IBM Quantum breaks the 100-qubit processor barrier. *IBM Research Blog* 2 (2021).

[19] Poulami Das, Swamit Tannu, Siddharth Dangwal, and Moinuddin Qureshi. 2021. Adapt: Mitigating idling errors in qubits via adaptive dynamical decoupling. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. 950–962.

[20] David P DiVincenzo and Panos Aliferis. 2007. Effective fault-tolerant quantum computation with slow measurements. *Physical review letters* 98, 2 (2007), 020501.

[21] Ivan B Djordjevic. 2021. *Quantum information processing, quantum computing, and quantum error correction: an engineering approach*. Academic Press.

[22] Will Finigan, Michael Cubeddu, Thomas Lively, Johannes Flick, and Prineha Narang. 2018. Qubit allocation for noisy intermediate-scale quantum computers. *arXiv preprint arXiv:1810.08291* (2018).

[23] Austin G Fowler, Matteo Mariantoni, John M Martinis, and Andrew N Cleland. 2012. Surface codes: Towards practical large-scale quantum computation. *Physical Review A* 86, 3 (2012), 032324.

[24] Jay Gambetta. 2020. IBM's roadmap for scaling quantum technology. *IBM Research Blog (September 2020)* (2020).

[25] Craig Gidney. 2021. Stim: a fast stabilizer circuit simulator. *Quantum* 5 (2021), 497.

[26] Daniel Gottesman. 1998. Theory of fault-tolerant quantum computation. *Physical Review A* 57, 1 (1998), 127.

[27] Daniel M Greenberger, Michael A Horne, and Anton Zeilinger. 1989. Going beyond Bell's theorem. In *Bell's theorem, quantum theory and conceptions of the universe*. Springer, 69–72.

[28] Jared B Hertzberg, Eric J Zhang, Sami Rosenblatt, Easwar Magesan, John A Smolin, Jeng-Bang Yau, Vivekananda P Adiga, Martin Sandberg, Markus Brink, Jerry M Chow, et al. 2021. Laser-annealing Josephson junctions for yielding scaled-up superconducting quantum processors. *npj Quantum Information* 7, 1 (2021), 129.

[29] Oscar Higgott. 2022. PyMatching: A Python package for decoding quantum codes with minimum-weight perfect matching. *ACM Transactions on Quantum Computing* 3, 3 (2022), 1–16.

[30] Adam Holmes, Mohammad Reza Jokar, Ghasem Pasandi, Yongshan Ding, Massoud Pedram, and Frederic T. Chong. 2020. NISQ+: Boosting quantum computing power by approximating quantum error correction. *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)* (2020), 556–569.

[31] Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. 2018. PySAT: A Python Toolkit for Prototyping with SAT Oracles. In *SAT*. 428–437. https://doi.org/10.1007/978-3-319-94144-8_26

[32] A Yu Kitaev. 2003. Fault-tolerant quantum computation by anyons. *Annals of physics* 303, 1 (2003), 2–30.

[33] Emanuel Knill and Raymond Laflamme. 1997. Theory of quantum error-correcting codes. *Physical Review A* 55, 2 (1997), 900.

[34] Emanuel Knill, Raymond Laflamme, and Wojciech H Zurek. 1998. Resilient quantum computation. *Science* 279, 5349 (1998), 342–345.

[35] Anirudh Krishna and David Poulin. 2021. Fault-tolerant gates on hypergraph product codes. *Physical Review X* 11, 1 (2021), 011023.

[36] Lingling Lao and Carmen G. Almudever. 2020. Fault-tolerant quantum error correction on near-term quantum processors using flag and bridge qubits. *Phys. Rev. A* 101 (Mar 2020), 032333. Issue 3. https://doi.org/10.1103/PhysRevA.101.032333

[37] Chin Yang Lee. 1961. An algorithm for path connections and its applications. *IRE transactions on electronic computers* 3 (1961), 346–365.

[38] Gushu Li, Yufei Ding, and Yuan Xie. 2019. Tackling the qubit mapping problem for NISQ-era quantum devices. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. 1001–1014.

[39] Sophia Fuhui Lin, Joshua Viszlai, Kaitlin N Smith, Gokul Subramanian Ravi, Charles Yuan, Frederic T Chong, and Benjamin J Brown. 2024. Codesign of quantum error-correcting codes and modular chiplets in the presence of defects. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. 216–231.

[40] Moein Malekakhlagh, Easwar Magesan, and David C McKay. 2020. First-principles analysis of cross-resonance gate operation. *Physical Review A* 102, 4 (2020), 042605.

[41] JF Marques, BM Varbanov, MS Moreira, Hany Ali, Nandini Muthusubramanian, Christos Zachariadis, Francesco Battistel, Marc Beekman, Nadia Haider, Wouter Vlothuizen, et al. 2022. Logical-qubit operations in an error-detecting surface code. *Nature Physics* 18, 1 (2022), 80–86.

[42] Abtin Molavi, Amanda Xu, Martin Diges, Lauren Pick, Swamit Tannu, and Aws Albarghouthi. 2022. Qubit mapping and routing via maxsat. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 1078–1091.

[43] SA Moses, CH Baldwin, MS Allman, R Ancona, L Ascarrunz, C Barnes, J Bartolotta, B Bjork, P Blanchard, M Bohn, et al. 2023. A race track trapped-ion quantum processor. *arXiv preprint arXiv:2305.03828* (2023).

[44] Prakash Murali, Jonathan M Baker, Ali Javadi-Abhari, Frederic T Chong, and Margaret Martonosi. 2019. Noise-adaptive compiler mappings for noisy intermediate-scale quantum computers. In *Proceedings of the twenty-fourth international conference on architectural support for programming languages and operating systems*. 1015–1029.

[45] Shota Nagayama, Austin G Fowler, Dominic Horsman, Simon J Devitt, and Rodney Van Meter. 2017. Surface code error correction on a defective lattice. *New Journal of Physics* 19, 2 (2017), 023050.

[46] Michael A Nielsen and Isaac L Chuang. 2010. *Quantum computation and quantum information*. Cambridge university press.

[47] John Preskill. 2018. Quantum Computing in the NISQ era and beyond. *Quantum* (2018).

[48] Ben W Reichardt. 2020. Fault-tolerant quantum error correction for Steane's seven-qubit color code with few or no extra qubits. *Quantum Science and Technology* 6, 1 (nov 2020), 015007. https://doi.org/10.1088/2058-9565/abc6f4

[49] Eyob A Sete, Angela Q Chen, Riccardo Manenti, Shobhan Kulshreshtha, and Stefano Poletto. 2021. Floating tunable coupler for scalable quantum computing architectures. *Physical Review Applied* 15, 6 (2021), 064063.

[50] Peter W Shor. 1995. Scheme for reducing decoherence in quantum computer memory. *Physical review A* 52, 4 (1995), R2493.

[51] VV Sivak, Alec Eickbusch, Baptiste Royer, Shraddha Singh, Ioannis Tsioutsios, Suhas Ganjam, Alessandro Miano, BL Brock, AZ Ding, Luigi Frunzio, et al. 2023. Real-time quantum error correction beyond break-even. *Nature* 616, 7955 (2023), 50–55.

[52] Thomas M Stace and Sean D Barrett. 2010. Error correction and degeneracy in surface codes suffering loss. *Physical Review A* 81, 2 (2010), 022317.

[53] Thomas M Stace, Sean D Barrett, and Andrew C Doherty. 2009. Thresholds for topological codes in the presence of loss. *Physical review letters* 102, 20 (2009), 200501.

[54] Andrew Steane. 1996. Multiple-particle interference and quantum error correction. *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences* 452, 1954 (1996), 2551–2577.

[55] Andrew M Steane. 1999. Quantum reed-muller codes. *IEEE Transactions on Information Theory* 45, 5 (1999), 1701–1703.

[56] J Stehlik, DM Zajac, DL Underwood, T Phung, J Blair, S Carnevale, D Klaus, GA Keefe, A Carniol, Muir Kumph, et al. 2021. Tunable coupling architecture for fixed-frequency transmon superconducting qubits. *Physical Review Letters* 127, 8 (2021), 080505.

[57] Yasunari Suzuki, Takanori Sugiyama, Tomochika Arai, Wang Liao, Koji Inoue, and Teruo Tanimoto. 2022. Q3DE: A fault-tolerant quantum computer architecture for multi-bit burst errors by cosmic rays. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 1110–1125.

[58] Bochen Tan and Jason Cong. 2020. Optimal layout synthesis for quantum computing. In *Proceedings of the 39th International Conference on Computer-Aided Design*. 1–9.

[59] Bochen Tan and Jason Cong. 2020. Optimal layout synthesis for quantum computing. In *Proceedings of the 39th International Conference on Computer-Aided Design*. 1–9.

[60] Jean-Pierre Tillich and Gilles Zémor. 2013. Quantum LDPC codes with positive rate and minimum distance proportional to the square root of the blocklength. *IEEE Transactions on Information Theory* 60, 2 (2013), 1193–1202.

[61] Yu Tomita and Krysta M Svore. 2014. Low-distance surface codes under realistic quantum noise. *Physical Review A* 90, 6 (2014), 062320.

[62] Maxime A Tremblay, Nicolas Delfosse, and Michael E Beverland. 2022. Constant-overhead quantum error correction with thin planar connectivity. *Physical Review Letters* 129, 5 (2022), 050504.

[63] Robert Wille, Lukas Burgholzer, and Alwin Zulehner. 2019. Mapping quantum circuits to IBM QX architectures using the minimal number of SWAP and H operations. In *Proceedings of the 56th Annual Design Automation Conference 2019*. 1–6.

[64] Anbang Wu, Gushu Li, Hezi Zhang, Gian Giacomo Guerreschi, Yufei Ding, and Yuan Xie. 2022. A Synthesis Framework for Stitching Surface Code with Superconducting Quantum Devices. In *Proceedings of the 49th Annual International Symposium on Computer Architecture* (New York, New York) *(ISCA '22)*. Association for Computing Machinery, New York, NY, USA, 337–350. https://doi.org/10.1145/3470496.3527381

[65] Chi Zhang, Ari B Hayes, Longfei Qiu, Yuwei Jin, Yanhao Chen, and Eddy Z Zhang. 2021. Time-optimal qubit mapping. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 360–374.

[66] Jiongzhi Zheng, Kun He, Jianrong Zhou, Yan Jin, Chumin Li, and Felip Manyà. 2022. Incorporating Multi-armed Bandit with Local Search for MaxSAT. *ArXiv* abs/2211.16011 (2022).

[67] Alwin Zulehner, Alexandru Paler, and Robert Wille. 2018. An efficient methodology for mapping quantum circuits to the IBM QX architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 38, 7 (2018), 1226–1236.

A Artifact Appendix

A.1 Abstract

This artifact includes the source code for the QECC-Synth and all necessary scripts to reproduce the key results presented in Table 1, 2, 3 and Fig. 9, 10, as well as to perform comparisons with the baselines (SATmap and Sabre) in our evaluation. The result of baselines (Surf-Stitch) in Table 2 and Fig. 9 are obtained from the paper [64].

The hardware requirements are minimal. Software dependencies are limited to commonly used Python packages.

Additionally, we provide a script to generate the benchmarks utilized in our evaluation. Please note that the evaluation results are averaged over multiple executions, so deviations in reproduction are to be expected.

Since running the evaluation program over all test cases may take weeks, we also provide the experimental data for a quicker review.

A.2 Artifact check-list

- **Algorithm:** QECC-Synth comprises three core algorithms:
  - `/MyCode/src/TopologyGraph.py`:
    Generates architectures with different sizes and connectivity topologies.
  - `/MyCode/src/StabCode.py`:
    Generates the QEC code to be synthesized.
  - `/MyCode/src/CodeStitch.py`:
    Takes the architecture topology and QEC code as input, runs a two-stage synthesis, and outputs the mapping and scheduling results for the stabilizer circuit of the QEC code on the architecture.
- **Output:**
  - `/MyCode/arch/{ArchName}_{ArchSize}.json:`
    Stores the architectures generated by `TopologyGraph.py`
  - `/MyCode/Code/XXX.stab:`
    Stores the QEC codes generated by `StabCode.py`
  - `/MyCode/output/S2_{CodeName}_{ArchName}.json:`
    Stores the synthesis result from `CodeStitch.py`
- **Run-time environment:** Python, Jupyter Notebook, C++
- **Metric:** Extra Cnot gate counts, circuit depth, physical qubit overhead, logical error rate, SAT problem size as explained in Section 6.
- **Hardware:** IntelCPU, memory size depending on the benchmark size (the largest benchmarks can be processed with 100 GB RAM).
- **Disk space required:** 1 TB is sufficient for the artifact evaluation and all software dependencies.
- **Time needed to complete experiments:** The approximate execution time for one task with QECC-Syntharound 0.5 to 2 hours. Simulating and evaluating the logical error rate of a deformed code may take 0.5 to 2 hours, as it requires sampling a large number of defect and noise shots. Reproducing all results in Table 1 could take hundreds of CPU hours. You can, however, reduce the number of sampled defect and

noise shots to speed up the process while maintaining the overall trend.
- **Publicly available:** Yes.
- **Code licenses:** Apache License 2.0
- **DOI:** 10.5281/zenodo.14061096

A.3 Description

A.3.1 How to access. This artifact can be downloaded at the following link https://zenodo.org/records/14061096.

A.3.2 Hardware dependencies. A regular server with Intel CPUs can run our artifact while the amount of RAM may limit the size of benchmarks that can be executed. In our experiments, we used 100 GB RAM to execute all benchmarks.

A.3.3 Software Dependencies. The artifact is implemented in Python 3.9.6. A Jupyter notebook is required, as we have prepared files with scripts that automatically and interactively reproduce the results for easy validation. Other dependencies include Qiskit, Stim, PyMatching, Sinter, NumPy, Matplotlib, and NetworkX.

A.4 Installation

To use our artifact, you may download the repo to your local machine from https://zenodo.org/records/14061096 and install the software dependencies by running the command:

```
pip install -r requirements.txt
```

However, as SATmap needs a different environment. You may need to create a new environment and run the command in under dirctiony `Baseline/satmap`:

```
pip install -r requirements.txt
sudo apt-get install libgmp-dev libgmpxx4ldbl
cd lib/Open-WBO-Inc/
make r
```

A.5 Evaluation and expected results

After downloading the artifact and installing all software dependencies, you can open the following jupyter notebook files to reproduce experimental for corresponding table and figures.

To reproduce the result of QECC-Synth, follows:

- `/MyCode/README`:
  This file contains all the instructions and parameters needed to obtain the results for QECC-Synthin Table 1. The function is set to read data by default. To reproduce the results, add the parameters "-S1 -S2" to all "python3 src/CodeStitch.py ..." commands.
- `/MyCode/eval/Fig_9.ipynb`:
  This jupyter notebook is for Fig. 9.
- `/MyCode/eval/Fig_10_1.ipynb` and `Fig_10_2.ipynb` is for Fig. 10.

To reproduce the result of Sabre in Table 1, follows:

- `/Baseline/Sabre`:
  This file contains all the instructions and parameters needed to obtain the results for Sabrein Table 1.

To reproduce the result of SATmap in Table 1, follows:

- `/Baseline/satmap/README`:
  This file contains all the instructions and parameters needed to obtain the results for SATmap in Table 1. The function is set to read data by default. To reproduce the results, add the parameters "–run" to all "python3 src/satmap.py ..." commands.

For the result in Table 3, they are from the `sol_XXX.txt` where `XXX` are

- `/Baseline/satmap/aux_files/sol_XXX.txt`:
  This file contains all the numbers of constraints and variable in the SAT problem for SATmap.
- `/MyCode/aux_files/sol_Surface_3-XXX.txt`:
  This file contains all the numbers of constraints and variable in the SAT problem for QECC-Synth.