Lattice Surgery Compilation Beyond the Surface Code

Laura S. Herzog[1, *], Lucas Berent[1, †], Aleksander Kubica[2, ‡], Robert Wille[1, 3, 4, §]
[1]Chair for Design Automation, Technical University of Munich, Germany
[2]Yale Quantum Institute & Department of Applied Physics, Yale University, New Haven, USA
[3]Munich Quantum Software Company GmbH, Garching near Munich, Germany
[4]Software Competence Center Hagenberg GmbH (SCCH), Hagenberg, Austria

***Abstract*—Large-scale fault-tolerant quantum computation requires compiling logical circuits into physical operations tailored to a given architecture. Prior work addressing this challenge has mostly focused on the surface code and lattice surgery schemes. In this work, we broaden the scope by considering lattice surgery compilation for topological codes *beyond* the surface code. We begin by defining a code *substrate*—a blueprint for implementing topological codes and lattice surgery. We then abstract from the microscopic details and rephrase the compilation task as a mapping and routing problem on a macroscopic routing graph, potentially subject to substrate-specific constraints. We explore specific substrates and codes, including the color code and the folded surface code, providing detailed microscopic constructions. For the color code, we present numerical simulations analyzing how design choices at the microscopic and macroscopic levels affect the depth of compiled logical $\mathrm{CNOT}+\mathrm{T}$ circuits. An open-source code is available on GitHub https://github.com/cda-tum/mqt-qecc.**

***Index Terms*—Quantum Error Correction, Fault Tolerance, Topological Codes, Compilation**

I. Introduction

To run large-scale quantum algorithms [1], it is essential to incorporate quantum error correction and fault tolerance to ensure reliable results [2]–[6]. In a fault-tolerant setting, logical qubits are encoded using a quantum error-correcting code and logical operations are designed to limit the propagation of errors. Consequently, executing quantum algorithms on any quantum computing platform requires translating logical circuits into physical ones compatible with hardware-specific constraints—a process known as *compilation*.

Most state-of-the-art research in fault-tolerant compilation [7]–[13] has focused on the surface code architecture [14]–[16], which can be implemented with a planar layout of qubits with nearest-neighbor connectivity. A key challenge in this setting is the implementation of logical two-qubit gates, which may naively require non-local qubit connectivity. A standard technique to address this issue is *lattice surgery* [17]–[19], where logical qubits are made to interact by performing measurements of additional ancilla qubits placed in the region connecting the surface code patches that encode these logical qubits.

Alternatives to the surface code can be found within the broader class of topological codes, which includes the color code [20], [21] and the folded surface code [22], [23]. These quantum error-correcting codes can also be implemented using a planar layout of qubits with geometrically-local (rather than strictly nearest-neighbor) connectivity. In return, the qubit overhead associated with performing error correction may be substantially reduced. Despite these potential gains, fault-tolerant compilation for general topological codes remained largely underexplored.

In this work, we address the problem of fault-tolerant compilation for topological codes by introducing a general formalism that is applicable *beyond* the surface code. This spans multiple levels of abstraction as illustrated in Fig. 1. On the microscopic level, we define the concept of a *code substrate*—a blueprint for realizing a topological code and its corresponding lattice surgery scheme; see Fig. 1b. We present two concrete examples of such substrates that support the color code and the folded surface code. While the folded surface code requires a substrate with slightly more complex qubit connectivity than the color code, it is expected to achieve higher fault-tolerant thresholds while retaining key advantages of the color code, such as transversal single-qubit Clifford gates.

At the macroscopic level, we abstract from the microscopic details by allocating logical qubits to regions of the substrate and constructing a coarse-grained *routing graph* as shown in Fig. 1c. This abstraction enables us to design a method for routing lattice surgery operations to implement two-qubit logical gates. We assess our approach through numerical simulations of logical $\mathrm{CNOT}+\mathrm{T}$ circuits using the color code substrate. Our results highlight how logical qubit allocation influences routing efficiency, and how T gates and logical circuit parallelism impact the overall compilation. More broadly, our work establishes foundational methods for compiling logical circuits in lattice-surgery-based architectures with topological codes, marking a first step toward practical fault-tolerant compilation beyond the surface code.

II. Main Ideas and Problem formulation

In this work, we consider the problem of compiling logical circuits consisting of Clifford+T gates to lattice surgery operations assuming that T gates are realized using magic state distillation and injection [24], [25]. However, we do not assume any specific distillation procedure.

Previous work on fault-tolerant computation such as [8]–[13], focused on the surface code and assumed

* laura.herzog@tum.de
† lucas.berent@tum.de
‡ a.kubica@yale.edu
§ robert.wille@tum.de

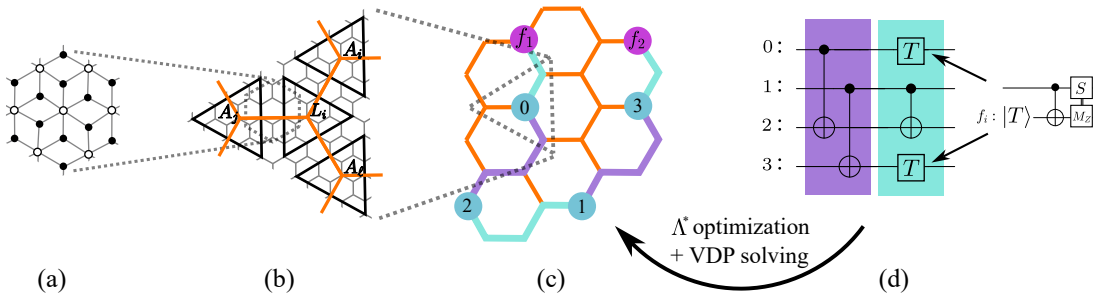


Fig. 1: Lattice surgery compilation overview. (a) Geometrically local physical qubit connectivity for physical data (black circles) and ancilla (white circles) qubits. (b) Substrate $\mathcal{S}$ (grey) for the color code, together with patches forming logical ancilla and data qubits (magic state patches not displayed in this cutout). This is referred to as the *microscopic* level. (c) Zooming out yields the *macroscopic* level with the routing graph $\mathcal{R}$ (orange) and specific allocation of logical data patches, represented by light blue nodes with integer labels. Magic state patches are displayed as pink nodes f_i. Furthermore, the colored paths represent a possible routing for the exemplary input circuit in (d), where parallel executable gates are displayed in the same color.

a rectangular logical qubit connectivity. To consider the compilation problem more generally, i.e., without assuming any particular qubit connectivity and quantum error-correcting code beforehand, we distinguish the *microscopic* and *macroscopic* levels of abstraction. At the microscopic level, the main problem is how to realize a lattice surgery scheme that implements logical two-qubit operations. At the macroscopic level, the central task is to assign logical qubits to physical locations, obtaining a logical routing graph whose shape is dictated by concrete choices at the microscopic level. Then valid paths between logical data qubits have to be identified to implement the logical quantum circuit.

To showcase the resulting formalism, consider the example of compilation with the 2D color code illustrated in Fig. 1. We start from the assumption of some geometrically local connectivity of physical data qubits (black circles) and ancilla qubits (white circles) as illustrated by the graph in Fig. 1a. In particular, we assume a stacked planar architecture that consists of a constant number of planar layers with qubit connectivity that is local within each layer and between layers (but not necessarily nearest-neighbor). Depending on the hardware it may be favorable to use indeed a stacked planar architecture or to project the stacked layers into one while accepting a more complex qubit connectivity.

Next we define a code *substrate* that captures the essential microscopic details of the hardware architecture. The substrate is a blueprint for realizing quantum error correction with some topological quantum code. For instance, a color code substrate is depicted by the gray regular hexagonal tiling in Fig. 1b. In this example, data qubits and stabilizers of the code correspond to, respectively, the vertices and faces of the tiling.

Building on this, we split the substrate into several regions (black triangles in Fig. 1b) that are dedicated to support logical data qubits, magic state patches, and the remaining *routing space*. For concreteness, we assume that the magic states are encoded in the same code as the logical qubits. Moreover, it is instructive to think of the routing space as being further subdivided into several ancilla patches. A logical data patch L_i and three ancilla patches are illustrated in Fig. 1b. The ancilla regions form the building blocks of what we refer to as *snakes* on the substrate. They are ancilla regions that are used to implement lattice surgery schemes between logical data patches. Note that a logical patch may also correspond to a logical ancilla qubit placed in the routing space.

A concrete assignment of logical data patches, magic state patches, and ancilla regions on the substrate determines a logical routing graph as shown in Fig. 1c. Its vertices correspond to logical data patches, magic state patches, and ancilla patches of the routing space. The edges of the routing graph indicate the connectivity between the regions on the substrate. Paths on this macroscopic routing graph correspond to microscopically defined snakes together with a logical ancilla patch that are used to perform CNOT gates between logical data patches.

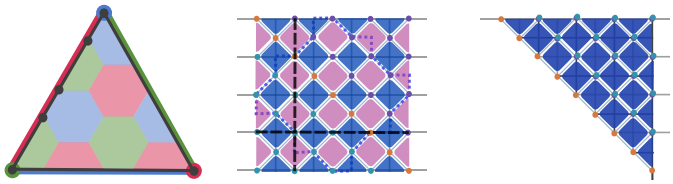
Overall, given a logical quantum circuit, for instance the one shown in Fig. 1d, the macroscopic compilation problem consists of two main tasks that we formulate as combinatorial optimization problems: i) assigning logical qubit labels to logical data vertices of the routing graph and ii) finding non-overlapping paths on the routing graph such that the compiled circuit depth is minimized. For instance, integer labels in Fig. 1d are mapped onto the logical routing graph as displayed by the corresponding labels on vertices in Fig. 1c.

Having the main ideas established we present the basics of quantum error-correction and lattice surgery in Sec. III. This is followed by Sec. IV in which the general formalism for lattice surgery compilation is presented. A solution for the macroscopic compilation task is presented in Sec. V. We detail the two considered substrates in Sec. VI, accompanied by numerical simulations for the macroscopic compilation of the color code substrate in Sec. VII, before concluding in Sec. VIII.

III. Basics of Quantum Error Correction and Lattice Surgery

A. Topological Quantum Codes

Topological codes [20], [26], [27] are a class of quantum error-correcting codes that are defined by placing qubits on some manifold; we restrict our attention to stabilizer codes [28]. In particular, in two dimensions checks and data qubits are often associated with faces and vertices of some tiling. Each check of the stabilizer group S is local, supported only on a constant number of qubits, whereas logical qubits



(a) (b) (c)

Fig. 2: (a) $d = 5$ color code on a hexagonal tiling with data qubits on the vertices. Each face defines both a X and Z stabilizer on the incident qubits. Both X_L and Z_L logical operators are supported on the boundaries of the triangular region. (b) $d = 5$ surface code, where Z stabilizers are pink faces and X stabilizers are blue faces. The dotted line encloses a region supporting the rotated surface code, where faces only partly contained in the boundary correspond to weight 2 stabilizers. (c) Folded surface code obtained by folding along the orange qubits on the diagonal. This procedure stacks X faces on Z faces and vice versa.

are encoded non-locally. Prominent examples for topological codes are color codes and surface codes that are defined using different tilings of 2D surfaces.

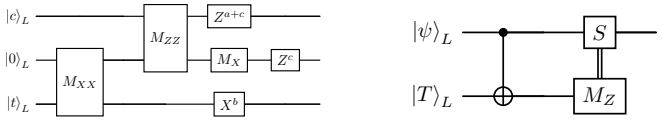
Two-dimensional *color codes* [20] are defined on 3-colorable and 3-valent tilings. An important family of color codes are triangular color codes on a hexagonal tiling whose faces and boundaries are three-colored. An example is displayed in Fig. 2a. Here, each face contained within the triangular region defines both an X and a Z check. Z and X logical Pauli string operators [20] terminate at boundaries of the same color as displayed in the figure, or connect three boundaries of different colors. The code *distance* d corresponds to the minimum weight of a non-trivial logical operator. In the example in Fig. 2a the distance is $d = 5$.

The *surface code* [14]–[16] can be defined on a rectangular tiling with two different types of boundaries. The checks are placed in a checkerboard pattern. An example is shown in Fig. 2b. Here, representatives of logical operators can be defined as strings that connect boundaries of the same type. In this example, horizontal strings correspond to Z_L (connecting so-called rough boundaries) and vertical strings (connecting so-called smooth boundaries) to X_L. Therefore, the distance of the displayed example is $d = 5$. We note that a rotated variant of the surface code, which is supported within a region enclosed by the dotted line in Fig. 2b, leads to a smaller qubit overhead.

In its original formulation, the surface code does not admit transversal implementation of the logical S gates that are compatible with geometrically-local qubit connectivity. This issue can be circumvented by folding [22], [23] the surface code along the diagonal (orange nodes in Fig. 2b and Fig. 2c) such that faces of different check type are stacked on top of each other. Additionally, smooth and rough boundaries are also stacked, such that each boundary enables access to both logical X_L and Z_L representatives, similarly as for the color code.

B. Lattice Surgery

Quantum computation requires a universal gate set, such as $\{\mathrm{CNOT}, \mathrm{H}, \mathrm{T}\}$. In general, transversal implementation of the two-qubit CNOT gate requires a non-local qubit connectivity. Therefore, to implement CNOT gates with local connectivity, one may resort to *lattice surgery* [17].



(a) (b)

Fig. 3: (a) Measurement-based representation of the CNOT gate. a, b, c are the measurement outcomes of M_{XX}, M_{ZZ}, M_X, respectively. (b) Injection of a logical magic state on a logical qubit with a CNOT gate.

To perform a CNOT gate with lattice surgery operations, one decomposes it in a measurement-based scheme [17], [18], [29] on the logical level as shown in Fig. 3a. The challenging parts are the joint logical M_{XX} and M_{ZZ} measurements [18]. To perform M_{ZZ}, we first initialize physical ancilla qubits between logical (data/ancilla) patches in $|+\rangle$. Then we *merge* both logical qubits by measuring joint stabilizers, thereby defining a code with support on the physical ancillas and the original logical patches. Moreover one needs to choose a subset of stabilizers whose measurement outcomes are equivalent to the value of the operator $Z_L Z_L$ on the separate logical patches, such that the desired measurement result can be retrieved. Finally, after performing error-correction, the merged object is *split* again by measuring the physical ancilla qubits destructively in the X basis. This is again followed by error-correction for the separate logical qubits. For M_{XX} the merged code may be adapted accordingly, e.g., physical ancillas are initialized in $|0\rangle$ and measured in the Z basis. Note that there are also lattice surgery schemes without physical ancillas between the patches [18], which can simplify above procedure.

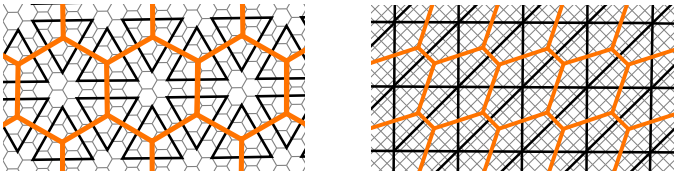
In the context of our work it is thus important to construct snakes between distant logical (data/ancilla) patches in a way such that the snake together with the logical qubits can be merged to form a joint code. Furthermore, one has to guarantee that among the joint stabilizers one can combine measurement results such that the desired logical outcome of $Z_L Z_L / X_L X_L$ can be inferred. Finally, there needs to be enough space between logical data qubits such that we can define a logical ancilla patch that is initialized in $|0\rangle$ and used in lattice-surgery implementation of a CNOT gate (Fig. 3a).

The microscopic details of the lattice surgery schemes that are specific to the two examples of color codes and folded surface codes will be detailed in Sec. VI-B and Sec. VI-C.

In order to complete the universal gate set, it is required to perform T gates. This can be done by using the logical circuit in Fig. 3b, where the CNOT gate is performed as explained above. T magic states are considered to be the outputs of specified magic state factory patches. We do not analyze in detail the inner workings of the factories; rather, we assume that each of them produces a T state after some fixed reset time.

C. Related Work

Let us conclude this section by reviewing how the resulting macroscopic compilation task is solved in related work. Surface code compilation with a universal or smaller gate set was explored in [7]–[9]. Furthermore, it has been shown that the complexity of optimizing a quantum circuit in the lattice



(a) (b)

Fig. 4: Code substrate $\mathcal{S}$ (depicted in grey) for (a) the color code and (b) the surface code. The logical patches from $\mathcal{L}, \mathcal{A}, \mathcal{F}$ are bounded by thick black lines. Unlike the ancilla patches, the data patches on the surface code substrate are folded, which is not explicitly displayed. The resulting routing graph $\mathcal{R}$ is the orange hexagonal graph, which is the same for both substrates. Both depicted substrates have $d = 5$ logical patches.

surgery model is NP-hard [30]. Related work by Litinski [19] motivated further research [10], [11], though it was argued that commuting Clifford gates until the end of the circuit may limit the parallelism [12], [13]. Whether or not Clifford gates are commuted to the end may depend on the circuit being compiled [31]. The aforementioned works focus on surface codes only; consequently, fault-tolerant compilation for other codes remained underexplored, despite some early ideas [32].

Using color codes instead of surface codes offers several advantages. For instance, the color code has a higher qubit density, supports transversal Clifford gates and hosts both X_L and Z_L per boundary. This simplifies the routing to some extent by removing the constraint of connecting a specific choice of boundaries. However, decoding algorithms for the color code tend to perform worse [33]–[37]. Nevertheless, recent experiments with color codes [38]–[41] and importantly, lattice surgery demonstration [42] underpin the potential of color codes as a viable fault-tolerant architecture.

Concerning the surface code, permitting a stacked planar architecture or relaxing the constraint on nearest-neighbor qubit connectivity enables to use the folded surface code [22], [23]. It retains many advantages of the color code while allowing straightforward decoding, the same way as for the standard surface code [15], [43]. However, to the best of our knowledge, no previous work on compilation with these codes exists.

IV. Formalism for Fault-Tolerant Compilation

To outline the general formalism of the considered compilation task, we define microscopic details about the code structure and derive the macroscopic picture, where compilation can be viewed as a mapping and routing task.

On the physical level we assume that the local qubit connectivity is given by a graph (cf. Fig. 1a) that may have a constant number of layers of physical qubits such that qubits within each layer and between neighbouring layers can have geometrically local interactions. Note that such an architecture is less hardware-demanding than the one needed to implement bivariate bicycle codes [44], [45]; as such, it may be easier to realize with superconducting qubits [46], [47], as well as neutral atoms [48], [49] or trapped ions [50], [51]. Physical data and ancilla qubits (black and white in Fig. 1) are placed on the vertices and their connectivity is given by the edges. This is the basis for choosing the code *substrate*, which is a tiling $\mathcal{S}$ (Fig. 1b) that can be used to realize logical qubits encoded in some topological quantum code. The substrate allows us to view a concrete physical qubit connectivity abstractly. We use the substrate to illustrate the layout of data qubits and geometrically-local checks of the code (faces of $\mathcal{S}$).

We distinguish three sets $\mathcal{L}, \mathcal{A}, \mathcal{F}$ of disjoint regions of vertices (Fig. 1b) of the substrate. The regions in $\mathcal{L} = \{L_1, L_2, ..., L_\ell\}$ correspond to logical data patches and $\mathcal{F} = \{F_1, F_2, ..., F_f\}$ to magic state patches, where magic state factories output magic states. The set $\mathcal{A}$ partitions the remaining region of the substrate into ancilla patches $\mathcal{A}^* = \{A_1, A_2, ..., A_a\}$ and additional physical ancilla qubits $\mathcal{A} \setminus \mathcal{A}^*$. The latter may be necessary to microscopically connect patches, depending on the substrate. A logical qubit can also require less space than the full, predefined patch provides. Both considered substrates are displayed in Fig. 4.

Finally, a choice of a topological quantum code and a substrate together with the sets $\mathcal{L}, \mathcal{A}$, and $\mathcal{F}$ determines a logical *routing graph* $\mathcal{R} = (V_\mathcal{R}, E_\mathcal{R})$ (Fig. 1c) that has a vertex for each set in $\mathcal{L} \cup \mathcal{A}^* \cup \mathcal{F}$, i.e., $V_\mathcal{R} = V_\mathcal{L} \cup V_\mathcal{A} \cup V_\mathcal{F}$. Its edges, $E_\mathcal{R}$ determine the connectivity between logical data, ancilla patches, and magic state patches as illustrated in Fig. 1c.

In this work we assume that the input circuit $\mathcal{C}$ is a logical quantum circuit that contains logical $\mathrm{CNOT} + \mathrm{T}$ gates. To obtain a universal gate set, single-qubit Clifford gates would be necessary as well, but they are not considered here since the computational hardness of the routing challenge for topological codes is dominated by multi-qubit operations [8], [11]–[13]. Moreover, our concrete choices of codes and substrates detailed below allow for transversal single-qubit Clifford operations.

To realize logical two-qubit operations we use lattice surgery schemes between logical patches on the substrate. Upon a choice of $\mathcal{L}, \mathcal{A}, \mathcal{F}$ and a code, lattice surgery requires the implementation of specific stabilizer measurements to connect logical patches as reviewed in Sec. III-B. To this end we realize what we refer to as *snakes* on the substrate. Snakes consist of neighboring ancilla regions and allow for the merge of distant logical (data/ancilla) patches to measure joint two-qubit logical Pauli operators. Throughout we assume that paths on $\mathcal{R}$ between distant logical data qubits contain both a snake as well as a logical ancilla patch, together with potentially needed physical ancilla qubits from $\mathcal{A} \setminus \mathcal{A}^*$ to perform a logical CNOT gate.

The concrete design of the stabilizers of a snake requires special care as the protocol must be implemented in a fault-tolerant and distance-preserving way. We detail two concrete constructions for logical color code patches and color code snakes, and folded surface code patches and surface code snakes in Sec. VI-B and Sec. VI-C, respectively.

An important optimization task of the macroscopic compilation problem is to place the logical qubit patches in such a way that the paths between interacting logical data and magic state patches can be routed efficiently. The potential for efficiency is determined by the inherent parallelism of the logical quantum circuit, which corresponds to how many gates can be applied simultaneously, i.e., the depth of the circuit. This amounts to defining an injective labeling $\Lambda : \mathcal{L} \to \mathbb{N}$. Since each logical

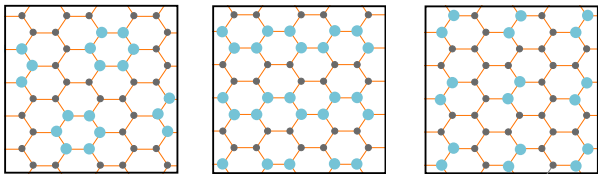


Fig. 5: Considered layouts, i.e., a choice of $\mathcal{L}$ and $\mathcal{A}$ that determines the routing graph $\mathcal{R}$, and their asymptotic packing ratio c, which we define as the number of logical data qubit patches (assuming that the folded surface code forms one patch) to the total number of patches. Light blue and gray nodes depict logical data qubits and ancilla patches, respectively.

patch $L_i \in \mathcal{L}$ corresponds to a vertex in $V_\mathcal{L}$, the map Λ induces a labeling of the routing graph vertices $\Lambda^* : V_\mathcal{L} \to \mathbb{N}$ with $\Lambda(L_i) = \Lambda^*(r)$, where r is the vertex corresponding to the patch L_i.

Example 1. *The circuit Fig. 1d has two layers (violet and teal) where gates inside a layer can in principle be executed in parallel. (c) shows a routing solution with some choice of $\mathcal{L}, \mathcal{A}$, and $\mathcal{F}$ for the color code with hexagonal routing graph $\mathcal{R}$. Furthermore, the values of a labeling Λ^* are depicted by the integer vertex labels. In this example, the compiled circuit has the same depth as the initial circuit.*

The macroscopic compilation depends on $\mathcal{R}$ and possible additional constraints that define *valid* paths on the routing graph, which in turn depend on the concrete choice of substrate.

V. Macroscopic Compilation of CNOT + T circuits

The logical routing graph $\mathcal{R}$ is defined by specifying the substrate $\mathcal{S}$ and the regions $\mathcal{L}, \mathcal{A}$, and $\mathcal{F}$ (corresponding to logical data patches, ancilla patches, and magic state patches). On this macroscopic level, one has to find non-overlapping paths or a *routing* on the graph between logical data patches and magic state patches according to the logical input circuit with the objective to reduce the depth of the final schedule of lattice surgery operations. Since the assignment of logical patches and factories influences the routing problem, the mapping Λ^* should be optimized to exploit as much inherent parallelism of the circuit as possible.

As displayed in Fig. 4, both substrates considered in this work abstract to a hexagonal routing graph. A specific choice of $\mathcal{L}$ and $\mathcal{A}$ induces a *layout* of logical data and ancilla patches on the graph. Some possible choices for such layouts are displayed in Fig. 5, where elements of $V_\mathcal{L}$ are shown in light blue and elements of $V_\mathcal{A}$ as smaller, grey vertices. Note that layout choices depend on the considered substrate, since a layout must allow us to find valid paths between all elements of $V_\mathcal{L}$ in principle. For the color code substrate a valid path is any path on the graph $\mathcal{R}$ that is not a single edge, between two logical data patches. Such paths could not host the logical ancilla patch needed for a CNOT and are thus not valid. Since each patch on the color code substrate has both X_L and Z_L available on each boundary, no further restrictions have to be drawn into consideration. For the surface code substrate with folded surface codes as logical data patches, however, the availability of logical operators for lattice surgery along the boundaries of the (ancilla) patches is limited and more rules for the valid paths must be defined. This is detailed in Sec. VI-C. Thus all three layouts in Fig. 4 can be considered for the color code substrate, while for the surface code substrate, only the pair layout allows to define valid paths between any pair of logical patches. For both substrates the magic state patches are placed on the boundary of the layout such that the magic state factory is left unspecified outside of the chosen architecture and is assumed to have enough space.

In the following sections we detail how we can solve the routing and mapping problem for the macroscopic compilation, similar to [8], [10], [12], [13].

A. Shortest-First VDP Solving

Let us start by considering logical circuits with CNOT gates between disjoint pairs of qubits only. Given a routing graph $\mathcal{R} = (V_\mathcal{R}, E_\mathcal{R})$ and a labeling of logical data patches Λ^*, one can formalize the search for paths as *vertex disjoint path problem* (VDP) [52]–[54]. The aim of the VDP is to find non-overlapping paths between pairs of vertices $C = \{(r_1, r'_1), ..., (r_k, r'_k) \mid r_i \neq r_j, r'_i \neq r'_j, r_i \neq r'_j \, \forall i, j\} \subset V_\mathcal{L} \times V_\mathcal{L}$. The considered vertices in $V_\mathcal{L}$ are mapped to logical qubit labels via a choice of Λ^*. Hence, the pairs to connect—i.e., the elements of C—represent the control and target qubits of CNOT gates and the paths contain the snakes and logical ancillas to perform lattice surgery between pairs of logical data qubits. Overall, C can be considered as quantum circuit consisting of CNOT gates.

A simple greedy algorithm [52] to find solutions for the VDP is to start with an empty solution W_i that is iteratively filled with paths p_j. We start by computing the shortest path with Dijkstra's algorithm between each pair in C, choose the shortest among those paths and add it to the solution $W_i = W_i \cup \text{argmin}_{p_i \in \{\text{DIJKSTRA}(r_j, r'_j) | (r_j, r'_j) \in C\}} |p_i|$. Then the vertices in the chosen path p_i are removed from $\mathcal{R}$. As pointed out in the beginning of the section, a substrate may require nontrivial constraints on paths on $\mathcal{R}$ to be valid such that DIJKSTRA may have to be adapted accordingly. This process of finding the shortest path and removing vertices is repeated until paths for all pairs in C are found or no further non-overlapping path can be found. It is guaranteed that no overlapping paths can occur since consumed vertices are removed from $\mathcal{R}$. Running this *shortest-first VDP subroutine* yields a set of paths $W_i = \{p_1, ...p_n\}$ that correspond to logical two-qubit gates, which can be executed in parallel with lattice surgery on the substrate.

In general, a CNOT quantum circuit contains multiple sets of pairs C_i as we do not assume that all CNOT gates have disjoint qubit support. Thus a CNOT quantum circuit can be viewed as $\tilde{C} = \{C_1, ..., C_m\}$. Additionally, the routing graph $\mathcal{R}$ may not allow us to find a set of vertex-disjoint paths for a given C_i. Thus, the aforementioned subroutine must be embedded in an algorithm, which we call *shortest-first VDP solving*. We start by employing the shortest-first VDP subroutine for C_1 and add the resulting set of paths W_i to the overall solution C'. If not all pairs in C_1 can be connected by non-overlapping paths, the remaining elements

from C_1 are copied to C_2. This may lead to C_2 containing gates with shared qubit labels. Thus, C_2 must be fixed by pushing problematic gates into C_3, such that C_2 contains gates with disjoint qubit labels only. Then, C_3 and the subsequent sets may be adapted in the same manner. Note that pushing requires us to respect the order of gates, as otherwise we could shuffle non-commuting gates. This shortest-first VDP subroutine together with the pushing is repeated until a path for each pair is found. The solution contains multiple sets of paths $C' = \{W_1, ...W_\Delta\}$, where Δ is the depth of the final, compiled circuit.

To include T gates in this picture, one can expand the shortest-first VDP subroutine to allow singleton tuples, i.e., single vertices $r_\ell \in V_\mathcal{L}$ corresponding to logical qubit labels via Λ^*. Additionally, the vertices of the magic state patches $V_\mathcal{F}$ must be taken into account in the routing problem formulation. We model magic state patches as logical patches that "receive" magic states after some fixed reset time from some adjacent magic state factory. The T gate can be implemented by gate injection from any available magic state patch. Thus, one can expand the shortest-first VDP subroutine by greedily computing all paths from the logical data qubit on which a T gate should be applied, r_ℓ, to the magic state patches from $V_\mathcal{F}$, if their T state is available. The shortest paths between r_ℓ and the available magic state patches is compared to all other paths corresponding to elements in C_i and the shortest is added to the solution as before. The availability of magic states is modeled by defining a reset time $t \in \mathbb{Z}$, which is initialized as positive integer and reduced by 1 after each completion of the shortest-first VDP subroutine. The factory is available when $t \leq 0$ and reset to the original value if the magic state was consumed. Note that is a pessimistic assumption as the units of time required to create a magic state may be less coarse grained.

The depth of the compiled circuit Δ depends on the choice of logical qubit labels via Λ^*, hence it is advantageous to optimize the labeling as well, which we do by using a combinatorial optimization heuristic.

B. Λ^* Optimization with Hill Climbing

The aim of optimizing the labeling Λ^* is to exploit the parallelism of the circuit as this directly influences the shortest-first VDP solving. The optimization requires defining a metric that assesses the quality of some choice of Λ^*.

Example 2. *The number of crossings of shortest paths between pairs in $\tilde{C}$ can serve as reasonable cost metric [13]. Consider Fig. 1c, where an optimal solution can be found by directly computing the shortest paths between the qubit vertices. Swapping the locations of two labels, e.g.,* 0 *and* 3*, however, would inevitably lead to a suboptimal solution because the respective shortest paths would cross. While for this small example the intuition reflects an exact solution, for larger instances counting the crossings of shortest paths will be a heuristic metric in general.*

More formally, this metric can be defined as computing all shortest valid paths in $\tilde{C}$ on $\mathcal{R}$. This yields a list of paths U_i for each $C_i \in \tilde{C}$, i.e., $U = \{U_1, ..., U_m\}$. If a vertex appears in multiple paths it is considered as one or more crossings. More specifically, for some U_i, the crossings are computed as $c_i = \sum_{v \in V_\mathcal{A}} \binom{K}{2}$, where K denotes in how many paths v appears. Thus each pair of paths sharing a vertex is considered as a crossing. Overall, the crossings are counted for each "layer" separately, i.e., $c' = \sum_{i=1}^{m} c_i$. Note that paths to factories are not considered in this metric, as the availability of a particular factory depends on the reset time until a T state can be consumed from a factory, which is not reflected here but only considered in the routing process discussed above.

To consider the dependency to the factories as well, one could use the final depth Δ from an execution of the shortest-first VDP solving as cost function, which is more expensive to compute, however.

Hill climbing [55]–[57], is a simple meta-heuristic that can find local extrema. The algorithm uses a random choice of a mapping Λ^* as initial value and computes the metric for each so-called neighboring solution. The algorithm proceeds in a greedy fashion: Once the best solution among the neighboring ones is chosen based on the defined metric, the next neighborhood is searched. This iterative procedure is repeated until the number of maximal iterations is reached or no better solution in the neighborhood can be found. We use the simple adaption of random restarts, i.e., running the algorithm with multiple randomly chosen Λ^* as initial values. This way a slightly larger search space can potentially be explored. Thus, the number of restarts and the maximal iterations per restart are important parameters of the optimization procedure.

The neighborhood for the greedy search in the hill climbing is defined as all combinations of swaps of the positions of each pair in $\tilde{C}$ [13]. Moreover, the factory positions are not changed during the hill climbing optimization. Merely the qubit labels, Λ^* are optimized. Note, however, that one can potentially improve the method by employing more sophisticated algorithms such as ABC [58] or genetic algorithms [59]. Our work is a proof-of-principle demonstration. We therefore leave a detailed comparison of different optimization strategies open for future work.

VI. Microscopic Substrate realizations

In this section, we outline the subsystem code formalism for lattice surgery introduced by Vuillot et al. [60] and use it to argue fault-tolerance properties of lattice surgery between distant logical patches on a substrate utilizing *distance-preserving snakes*. We also discuss microscopic realizations of the color code and surface code substrates.

A. Lattice Surgery Schemes from Distance-Preserving Snakes

We consider two codes defined by stabilizers on the separate patches and the merged patch $\mathcal{Q}_{\text{split}} = \langle S_{\text{split}} \rangle, \mathcal{Q}_{\text{merged}} = \langle S_{\text{merged}} \rangle$. To form a joint subsystem code $\mathcal{Q}$ we define the gauge group $\mathcal{G}$ to be generated by $S_{\text{split}}, S_{\text{merged}}$, and additional operators that are the anti-commuting conjugate partners of elements in $\mathcal{M}_p \subseteq S_{\text{split}} \cap \mathcal{L}_{\text{merged}}$ and $\mathcal{M}_m \subseteq S_{\text{merged}} \cap \mathcal{L}_{\text{split}}$. Adding these conjugate partners ensures that the center of the gauge group contains no elements from $\mathcal{M}_p, \mathcal{M}_m$ and thus results in the stabilizer group $S = Z(\mathcal{G}) = S_{\text{split}} \cap S_{\text{merged}}$. Finally, the gauge operators acting non-trivially on the gauge

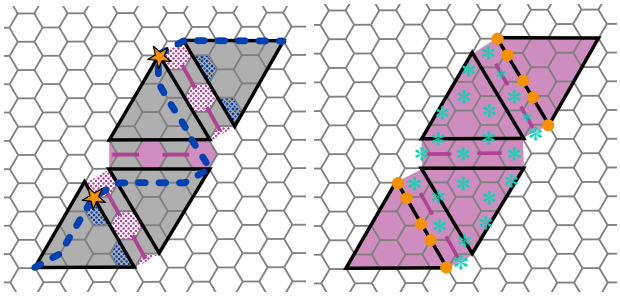


Fig. 6: Color code snake together with logical ancilla and data qubit. (a) Stabilizers S of the subsystem code $\mathcal{Q}$. Grey plaquettes host X and Z stabilizers and additional Z stabilizers are depicted in solid pink. Elements of $\mathcal{L}_g$ correspond to pink and blue dotted faces. The blue dashed line is an X bare logical operator; its parts stretching from from the left bottom corner to one of the two orange stars illustrate two possible gauge operators. (b) Z stabilizers of the merged code. Asterisks mark the subset M of Z stabilizers needed to measure the logical $Z_L Z_L$ operator (supported on the orange circles).

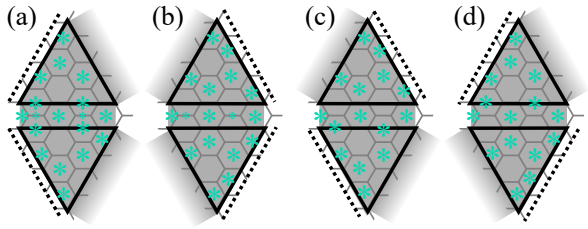


Fig. 7: Stabilizer subsets marked by asterisks for the logical $X_L X_L / Z_L Z_L$ measurements along the STDW between two ancilla patches of the color code. Outer boundaries without adjacent ancilla patch in the snake are marked by dotted lines; it is required that each qubit on such a boundary is "touched" an even number of times by a stabilizer of fixed type. On the other (inner) boundaries where the snake may extend (shaded in gray), qubits are touched only once. Depending on how the patches are placed on the substrate, one may reflect the pairs of patches along the vertical axis. Note that this pattern can be geometrically extended to larger distances.

qubits are defined as $\mathcal{L}_g = \mathcal{G} \setminus S$ and the bare logical operators $\mathcal{B} = C_{\mathcal{G}}(\mathcal{G}) \setminus \mathcal{G}$, where $C_{\mathcal{G}}(\cdot)$ denotes the centralizer in $\mathcal{G}$ and the dressed logical operators $\mathcal{D} = C_{\mathcal{G}}(S) \setminus \mathcal{G}$.

To argue fault-tolerance properties, we cast lattice surgery between two distant patches connected by a snake on the substrate in this formalism. In particular, we want to ensure that we can always measure $Z_L Z_L$ or $X_L X_L$ fault tolerantly between two distant logical patches by connecting them via a snake that has a *distance preserving* property. We define a distance preserving snake as ancilla region between two logical patches that allows us to implement a fault-tolerant lattice surgery scheme. In the following we show that we can indeed construct distance preserving snakes for the color code and surface code substrates. To this end, we construct a common subsystem code such that the split and merged patches are different gauge fixes of operators of $\mathcal{Q}$ and examine the corresponding logical operators.

B. Color Code Substrate

For the color code substrate we choose a hexagonal tiling on which we define the regions $\mathcal{A}^* \cup \mathcal{L} \cup \mathcal{F}$ that form distance $d = 2t+1$ triangular color code patches, where t is a positive integer. For this substrate each patch has $q = 3t^2 + 3t + 1$ physical data qubits.

To perform lattice surgery between two neighboring logical patches, their adjacent boundaries are connected by measuring intermediary operators that create a *semitransparent domain wall* (STDW) [61]. For a $Z_L Z_L$ merge between two adjacent logical patches, one extends the existing Z and X stabilizers and uses further weight $2, 3, 5$, and 6 stabilizers along the STDW. For a $X_L X_L$ merge, the roles of X and Z stabilizers must be swapped. For ease of presentation we focus on the case of measuring $Z_L Z_L$.

To argue fault-tolerance properties of lattice surgery on the color code substrate, we define the joint subsystem code $\mathcal{Q}$ as introduced above to construct distance preserving snakes for $Z_L Z_L$ lattice surgery on the color code substrate. In the following we focus on the case illustrated in Fig. 6. S_{split} is defined by two disjoint logical color code patches and additional disjoint ancilla stabilizers supported on the snake. S_{merged} is defined on the logical color code patches merged together by a snake whose X stabilizers are the gray faces in Fig. 6a and the Z stabilizers are depicted in Fig. 6b.

The set of gauge operators $\mathcal{L}_g$ contains the new intermediary weight $3, 5$, and 6 Z plaquettes $\mathcal{I}$ along the STDWs next to logical patches, depicted as pink dotted plaquettes in Fig. 6a. These are the gauge operators that will be fixed for the merge. The stabilizers in S_{split} commute with the new plaquettes, except for the weight 4 X checks along the boundaries of the logical patches, which are also in $\mathcal{L}_g$ (blue dotted plaquettes in Fig. 6a). These can be extended to weight 6 stabilizers of $\mathcal{Q}$ along the STDW. To ensure that the stabilizer group of $\mathcal{Q}$ has the correct form, we add X string operators to $\mathcal{L}_g$ as anti-commuting conjugate partners of the operators in $\mathcal{I}$. To do this, it is helpful to distinguish an *inner* boundary of the patch, i.e., a boundary which is connected to another patch with a STDW and an *outer* boundary without adjacent connected patch. Starting from the left logical patch L_1, a string operator can be chosen on L_1 such that one of its two point-like excitations is on the outer boundary of L_1, and the other excitation appears on an element from $\mathcal{I}$ on an inner boundary. One can add another string operator extending through the snake that can create an excitation on an element of $\mathcal{I}$ on the inner boundary adjacent to the top right logical patch (cf. Fig. 6a). This choice of gauge operators ensures that they commute with the stabilizers of the subsystem code (grey faces) and anti-commute with elements of $\mathcal{I}$, which are not supposed to be stabilizers, as they anti-commute with the gauge operators of the split code (blue dotted faces).

To fix the gauge, the operators in $\mathcal{I}$ are measured and the stabilizers are updated according to the Gottesman-Knill theorem [28].

Note that the domain walls dictate a special behaviour of the logical string operators. Logical Z_L strings can terminate at the STDW such that the weight of Z_L operators remains the same as for the initial logical patches. X strings cannot terminate at the STDW but only on the color code boundaries of the appropriate color [61]. Thus $\mathcal{Q}_{\text{merged}}$ and $\mathcal{Q}$ have logical Z operators of weight d of a single logical patch. Bare logical X operators of $\mathcal{Q}$ can stretch from one logical patch to another,

but since there are X gauge operators supported on the left logical patch and the snake, the minimum distance of a dressed logical operator of $\mathcal{Q}$ is d. In the example in Fig. 6 the resulting operator is supported on the top right logical patch. Hence, the code distance of the subsystem code is at least the same as of an individual logical patch.

Finally, we have to define a subset M of operators whose measurement outcomes determine the desired $Z_L Z_L$ value. M must include the operators on the qubits at the inner boundaries given by $\mathcal{I}$. Here, we stick to Z_L along the inner boundaries of the logical patches, but other representatives of $Z_L Z_L$ may be chosen as well. Since the substrate is a 3-valent graph and the snake slightly differs depending on its exact shape, the definition of M requires special care.

The construction of M starts by choosing suitable stabilizers with support on each ancilla patch on the snake connecting the logical patches. The outer boundaries on the snake's patches are important as M must include all stabilizers with support on such a boundary to ensure that each qubit is in the support of an even number of stabilizers. After specifying the stabilizers for all ancilla patches, suitable stabilizers must be chosen along the domain walls connecting inner boundaries. As exemplified in Fig. 7, the analysis reduces to four possible cases between two neighbouring patches. The snake and logical patches illustrated in Fig. 6b require case (d) between ancilla patches on the snake. In this case we have to choose the intermediary stabilizers equivalent to case (b) and (c) for the displayed example. Due to the translational invariance of the hexagonal tiling as well as the periodicity of the STDW's structure and the logical patches, we can always find such a pattern for color code patches.

The situation is equivalent for longer snakes, where the subsystem code is enlarged by additional stabilizers supported on the additional ancilla patches along the snake.

For the split we simply reverse the roles of $\mathcal{Q}_{\text{merged}}$ and $\mathcal{Q}_{\text{split}}$, split the previously extended weight 6 stabilizers and apply the Gottesman-Knill rules.

Note that the color code lattice surgery scheme can also be done with $\mathcal{A} \backslash \mathcal{A}^* = \emptyset$, as proposed in [18], [61], but requires higher weight stabilizers.

Finally, our source code provides a method to construct both the stabilizers of such merged objects as well as the set M for in principle arbitrary distances and number of patches.

C. Surface Code Substrate

As the surface code substrate we consider a square tiling and propose to use the folded surface code to encode logical data qubits. To "hide" the folding diagonal[1], we place folded surface codes on the substrate in pairs to form squares (cf. the pair layout in Fig. 5). For each folded surface code, we consider a stacked layer architecture to support its two halves or we project the two halves onto one layer with more complex qubit connectivity. The logical data qubits allow to access both X_L and Z_L for lattice surgery on the short boundaries due to the folded structure. The logical patches are connected

[1]To the best of our knowledge lattice surgery along the folding diagonal of the folded surface code is not possible without applying additional local unitaries that in general change the structure of the tiling.

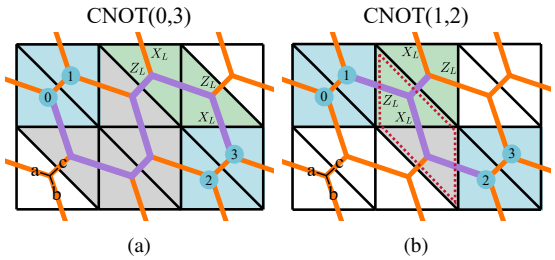


Fig. 8: Valid paths on the surface code substrate with folded surface codes as logical data patches. Folded surface codes are displayed in light blue, snakes in grey, and logical ancilla patches in green. $\mathcal{R}$ is shown in orange and a routing path between logical qubits is marked in violet. (a) The shortest path on $\mathcal{R}$ between logical data patches 0 and 3 is a valid path. (b) The shortest path on $\mathcal{R}$ between logical patch 1 and 2 is not valid, since the pair of triangles marked by red dashed lines would allow us to do either $X_L X_L$ or $Z_L Z_L$ lattice surgery, but not both as needed for the implementation of a CNOT gate. A slightly adapted valid path is displayed in violet together with a possible logical ancilla patch placement.

by snakes consisting of a single layer compatible with the standard (not folded) surface code.

Logical ancillas for the realization of CNOT gates comprise two neighboring triangles from $\mathcal{A}^*$, forming square- or parallelogram-shaped regions. Moreover, we assume that the magic state patches are standard surface codes as well. The snakes forming diagonal parts on this substrate are equivalent to rotated surface codes and the horizontal and vertical parts of a snake are standard surface codes.

The resulting routing graph is hexagonal (cf. Fig. 4), however, since only one type of logical (X_L or Z_L) is available per boundary of standard surface code patches, the definition of valid paths for this substrate differs from the color code substrate. In particular, (i) a valid path must be large enough to contain a logical ancilla patch needed to implement the measurement-based lattice surgery scheme for a CNOT and (ii) the path must allow that this ancilla patch can be placed in a way that ensures that both $X_L X_L$ and $Z_L Z_L$ measurements can be performed using the snake. To see this, assign one of three directions a, b, or c to each edge of the hexagonal routing graph. For a routing path to be valid, it must contain at least one edge of each direction as shown in Fig. 8. In the considered setting where logical ancillas for the CNOT surgery are placed on the snake, paths that do not include all types of edges cannot give access to the required boundaries of the logical ancilla. Hence, connecting two distant logical qubits is possible as long as the chosen layout is sufficiently sparse and slightly larger paths that include edges of all directions can be chosen instead of the shortest paths on $\mathcal{R}$.

Finally, there is an important detail when considering surface code snakes, surface code logical ancillas, and folded surface code logical data patches, as illustrated in Fig. 9. Considering the left logical ancilla qubit enclosed by the black boundary of two triangles, one can observe that there is in principle some freedom on how one effectively places the logical qubit within the triangles. Imagine the case in which one wants to perform lattice surgery between this logical ancilla patch and another adjacent logical data patch (not shown in the figure). Due to the freedom of effective placement within

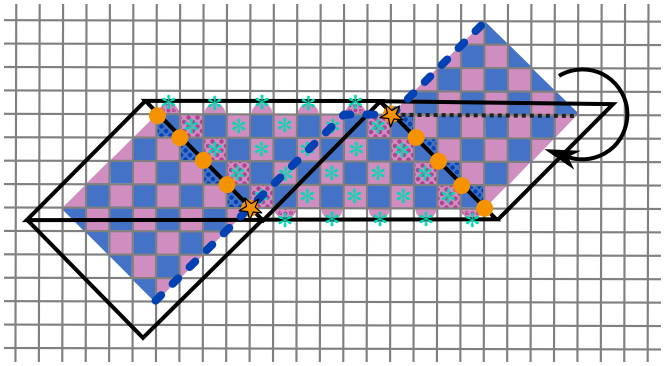


Fig. 9: Surface code snake merged with a logical ancilla patch on the left and a folded surface code logical patch on the right. The arrow indicates the fold. Each logical patch has distance $d = 5$. X stabilizers are displayed in blue and Z stabilizers are displayed in pink. Dotted faces indicate elements in $\mathcal{L}_g$. The product of Z stabilizers to consider for the $Z_L Z_L$ measurement (supported on qubits with orange dots) are marked with asterisks. An X bare logical and the corresponding gauge operators are shown by the blue dotted line and stars following the same logic as in Fig. 6.

the black triangles, it may be necessary to define slightly deformed stabilizers between the logical qubits to perform lattice surgery if they are not aligned perfectly, however it can still be implemented with local qubit connectivity.

We illustrate how to construct distance preserving snakes, by considering the surface code snake illustrated in Fig. 9, which connects a folded surface code patch and a logical surface code patch. Here, S_{split} consists of the stabilizers of the disjoint logical patches and stabilizers on the ancilla space on the snake. S_{merged} consists of one joint patch formed by additional checks in the rotated surface code part of the snake. The intermediary Z checks $\mathcal{I}$ on the snake are supported on the weight 2 and 4 Z checks that act on one of the qubits marked with orange dots in Fig. 9. These are in $\mathcal{L}_g$ and are again the ones we gauge fix for the merge. Moreover, the weight 3 X checks along the boundary of the logical ancilla patch and the folded surface code patch anti-commute with the new Z checks formed on plaquettes in $\mathcal{I}$. These are also in $\mathcal{L}_g$ and can be extended to weight 4 faces that commute with the new Z checks in S. Similarly to the color code substrate, one can choose X string operators in $\mathcal{L}_g$ with support on the left logical patch and on the snake to be anti-commuting conjugate partners of elements in $\mathcal{I}$.

The Z distance of $\mathcal{Q}_{\text{merged}}$ and bare logical operators of $\mathcal{Q}$ is the same as for $\mathcal{Q}_{\text{split}}$. Logical X operators of $\mathcal{Q}$ stretch from one logical patch to another. However, these bare logical operators can be multiplied by X gauge operators such that the dressed logical operators are guaranteed to have support on the right logical patch and hence preserve distance d.

Finally, to obtain the value of $Z_L Z_L$, we have to define a set M of operators whose measurement outcomes give the value of the operator. For the surface code substrate these are straightforward to find, due to the checkerboard structure. Hence, the set M corresponds to the plaquettes marked by the asterisks on the snake in Fig. 9. An automatized construction for surface code snakes merged with logical patches is provided in the open-source package.

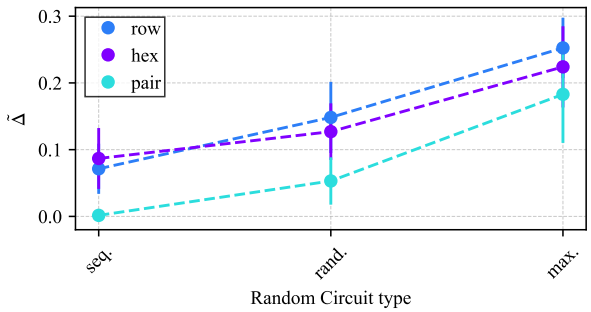


Fig. 10: Mean improvements from the hill climbing depending on different circuit types. Here, purely random CNOT circuits without T gates are considered. "seq." means that at most two CNOTs can be applied in parallel. The "rand." circuits have CNOTs drawn from a random distribution. "max." means that all available qubits can be used in parallel, i.e., $q/2$ gates are applied in parallel. Per data point, 50 random circuits with $q = 24$ qubits were sampled. Each run of the hill climbing algorithm uses 50 iterations for each of the 10 restarts. In total, $4q$ gates per circuit were used.

VII. Numerical Insights Into the Macroscopic Compilation With the Color Code

Here we describe numerical experiments to explore how the depth, number of available magic state patches, reset time of the magic state factories, and the chosen optimization metric influence the behavior of the proposed macroscopic compilation routine from Sec. V for the color code substrate. The implementation is part of the *Munich Quantum Toolkit* [62].

A. Setup

The macroscopic compilation operates on the logical routing graph $\mathcal{R}$. The valid paths on $\mathcal{R}$ for the color code substrate do not require special constraints, despite the fact that paths must be large enough to host at least one logical ancilla patch in order to perform a CNOT between logical data patches. This is ensured by the graph structure and the requirement that no direct connections between logical data patches are allowed. We consider three choices of layouts as illustrated in Fig. 5. Magic state patches are placed at the boundary of the routing graph and connected with two edges to the bulk of the routing graph, while the remaining edge is reserved for the realization of the magic state factory. We assume that for each factory a new T state is available after some time interval given as parameter.

B. CNOT Circuit Compilation

Naturally the amount of possible parallelization of the input circuit $\tilde{C}$ has direct impact on the possible depth of the compiled circuit C'. In the language of the routing problem from Sec. V-A, the parallelism of $\tilde{C}$ is higher if it needs to be decomposed into fewer subsets C_i for a fixed number of total gates. This initial parallelism may be increased in principle by (pre)-compilation on the circuit-level. The initial parallelism does not only affect the shortest-first VDP solving, but also the potential of improving Λ^*. To study the behaviour of the hill climbing optimization for different levels of initial parallelism while ignoring effects from T gates, we consider three types of uniformly random CNOT circuits. First, the "sequential" (seq.) type constructs circuits in such a way that

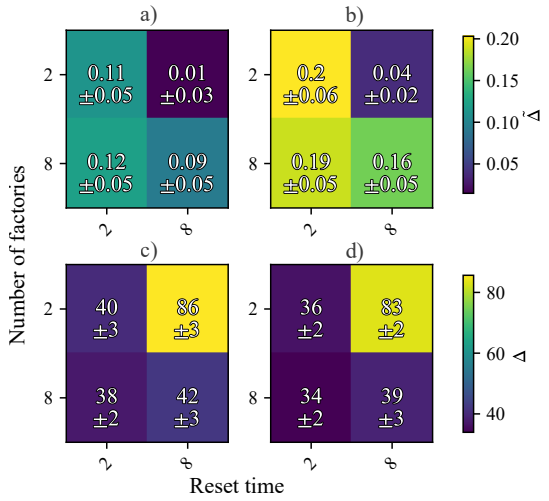


Fig. 11: Mean improvements $\tilde{\Delta} = (\Delta_i - \Delta_f)/\Delta_i$ for the heuristic crossing metric (a) and the exact routing metric (b). Absolute depth Δ_f after optimization for the heuristic crossing metric (c) and the exact routing metric (d). Per data point 50 different "random" circuits with $q = 24$ qubits and $r = 0.8$ were sampled. The "row" layout was chosen as well as the $\mathcal{R}$ for the color code substrate. Per circuit, $4q$ gates were considered.

only two gates can be applied in parallel in each circuit layer in principle as only two subsequent gates have disjoint qubit labels. The "random" (rand.) circuit type directly samples the target and control labels of gates from a uniform distribution. The "maximal" (max.) type has maximal initial parallelism in the sense that $q/2$ subsequent CNOT gates have disjoint qubit labels for q logical qubits.

The mean improvements for the hill climbing for these randomly sampled circuits for different layouts are displayed in Fig. 10. The improvement is defined as $\tilde{\Delta} = (\Delta_i - \Delta_f)/\Delta_f$ where Δ_f is the depth of C' computed with the best Λ^* given the heuristic crossing metric. In turn, Δ_i is the depth of C' with the corresponding random initialization of Λ^*.

One can observe the tendency that circuits with a higher degree of parallelism can indeed be improved more by the hill climbing optimization with the crossing metric. Note that the error bars have considerable size because randomness does not only enter for the randomly sampled circuits but also due to the randomly sampled restart instances Λ^* for the hill climbing.

The comparison between different layouts shows that the optimized Λ^* on the pair layout tends to be improved less. A reason for this may be that sparser layouts with larger routing space encounter less crossings during the VDP solving such that one can assume less potential of Λ^* improvement for them. However, if the initial parallelism is large enough—increasing the possibility of crossings—also optimization for the pair layout becomes more fruitful. Overall, $\tilde{\Delta}$ increases with growing parallelism for all layouts. The asymptotic packing ratio of the row and hexagonal layout are very close which is why they show similar behavior.

C. CNOT + T *Circuit Compilation*

Here we consider T gates in the input circuits as well. To this end, random circuits are sampled in such a way that there will be a specific ratio of CNOT gates in comparison to the total gates $r = \#CNOT/(\#CNOT + \#T)$. This ratio is important as T gates make an efficient solution particularly difficult due to the reset time t and the limited availability of magic state patches. We investigate the behavior of the hill climbing optimization with different numbers of magic state patches as well as different reset times. The results for "random" circuits are shown in Fig. 11. The first row displays the improvements $\tilde{\Delta}$ for the heuristic crossing metric (a) and the more expensive metric Δ_f of the shortest-first VDP solving (b). The improvements work best for many magic state patches and a small reset time, while they are most likely reduced for more challenging setups. Using the exact metric Δ_f makes the hill climbing solutions more advantageous than with the heuristic. The bottom row of Fig. 11 displays the corresponding absolute values for Δ_f of the VDP solving runs of the best Λ^* from the hill climbing optimization for the respective metrics (c, d). It is apparent that irrespective the chosen metric the achieved depth is the lowest for many magic state patches and low reset time t, with slightly better results using the metric Δ_f for hill climbing.

VIII. Conclusion and Outlook

Our work presents a generalized approach to fault-tolerant compilation based on lattice surgery, independent of any specific topological code. We split the overall compilation into two levels of abstraction, microscopic and macroscopic, analyzing two substrate choices and proposing a routing and mapping routine. We investigated how microscopic design choices influence the macroscopic compilation task and provided initial numerical results for the color code, highlighting how logical circuit parallelism and magic state availability affect overall performance.

Our methods enable future optimizations of fault-tolerant compilation. At the microscopic level, we envision exploring different substrates, possibly motivated by different quantum computing platforms, and taking hardware imperfections into account. For instance, one possible direction is to incorporate into our framework the recently-developed methods for adapting surface code patches in the presence of defective qubits and gates [63]–[65]. At the macroscopic level, the Λ^* optimization and algorithms to solve the (extended) VDP problem allow for further improvements; commuting some Clifford gates to the end of the circuit and exploring potential parallelism in the modified circuit may be valuable. Future work may also explore how to incorporate novel schemes for magic state distillation and cultivation [66], with a careful analysis of their layouts within the bulk of the routing graph.

Acknowledgements

L.H., L.B., and R.W. acknowledge funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 101001318). This work was part of the Munich Quantum Valley, which the Bavarian state government supports with funds from the Hightech Agenda Bayern Plus as well as by the BMK, BMDW, and the State of Upper Austria in the frame of the COMET program (managed by the FFG).

References

[1] Alexander M. Dalzell *et al.*, *Quantum algorithms: A survey of applications and end-to-end complexities*, 2023. arXiv: 2310.03011 [quant-ph]. [Online]. Available: https://arxiv.org/abs/2310.03011.

[2] Peter W. Shor, "Scheme for reducing decoherence in quantum computer memory," *Physical Review A*, vol. 52, no. 4, R2493–R2496, Oct. 1995, ISSN: 1094-1622. DOI: 10.1103/physreva.52.r2493. [Online]. Available: http://dx.doi.org/10.1103/PhysRevA.52.R2493.

[3] A. M. Steane, "Error correcting codes in quantum theory," *Physical Review Letters*, vol. 77, no. 5, pp. 793–797, Jul. 1996, ISSN: 1079-7114. DOI: 10.1103/physrevlett.77.793. [Online]. Available: http://dx.doi.org/10.1103/PhysRevLett.77.793.

[4] Peter W Shor, "Fault-tolerant quantum computation," in *Proceedings of 37th conference on foundations of computer science*, IEEE, 1996, pp. 56–65. DOI: 10.1109/SFCS.1996.548464.

[5] A Yu Kitaev, "Quantum computations: Algorithms and error correction," *Russian Mathematical Surveys*, vol. 52, no. 6, p. 1191, 1997.

[6] John Preskill, "Reliable quantum computers," *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, vol. 454, no. 1969, pp. 385–410, 1998. DOI: 10.1098/rspa.1998.0167.

[7] Daniel Herr, Franco Nori, and Simon J. Devitt, "Lattice surgery translation for quantum computation," *New J. Phys.*, vol. 19, no. 1, p. 013034, Jan. 2017, Publisher: IOP Publishing. DOI: 10.1088/1367-2630/aa5709.

[8] Mingzheng Zhu *et al.*, "Ecmas: Efficient circuit mapping and scheduling for surface code," in *2024 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, 2024, pp. 158–169. DOI: 10.1109/CGO57630.2024.10444874.

[9] L. Lao *et al.*, "Mapping of lattice surgery-based quantum circuits on surface code architectures," *Quantum Sci. Technol.*, vol. 4, no. 1, p. 015005, Sep. 2018, Publisher: IOP Publishing. DOI: 10.1088/2058-9565/aadd1a.

[10] George Watkins *et al.*, "A high performance compiler for very large scale surface code computations," *Quantum*, vol. 8, p. 1354, May 22, 2024. DOI: 10.22331/q-2024-05-22-1354. arXiv: 2302.02459[quant-ph].

[11] Allyson Silva *et al.*, "Multi-qubit Lattice Surgery Scheduling," in *19th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2024)*, Frédéric Magniez and Alex Bredariol Grilo, Eds., ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 310, Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024, 1:1–1:22, ISBN: 978-3-95977-328-7. DOI: 10.4230/LIPIcs.TQC.2024.1.

[12] Michael Beverland, Vadym Kliuchnikov, and Eddie Schoute, "Surface code compilation via edge-disjoint paths," *PRX Quantum*, vol. 3, no. 2, p. 020342, May 25, 2022, Publisher: American Physical Society. DOI: 10.1103/PRXQuantum.3.020342.

[13] Abtin Molavi *et al.*, "Dependency-aware compilation for surface code quantum architectures," *Proceedings of the ACM on Programming Languages*, vol. 9, no. OOPSLA1, pp. 57–84, Apr. 2025, ISSN: 2475-1421. DOI: 10.1145/3720416. [Online]. Available: http://dx.doi.org/10.1145/3720416.

[14] A.Yu. Kitaev, "Fault-tolerant quantum computation by anyons," *Annals of Physics*, vol. 303, no. 1, pp. 2–30, 2003. DOI: 10.1016/s0003-4916(02)00018-0.

[15] Eric Dennis *et al.*, "Topological quantum memory," *Journal of Mathematical Physics*, vol. 43, no. 9, pp. 4452–4505, 2002. DOI: 10.1063/1.1499754.

[16] Austin G. Fowler *et al.*, "Surface codes: Towards practical large-scale quantum computation," *Phys. Rev. A*, vol. 86, no. 3, p. 032324, Sep. 18, 2012. DOI: 10.1103/PhysRevA.86.032324. arXiv: 1208.0928[quant-ph].

[17] Dominic Horsman *et al.*, "Surface code quantum computing by lattice surgery," *New J. Phys.*, vol. 14, no. 12, p. 123011, Dec. 2012, Publisher: IOP Publishing. DOI: 10.1088/1367-2630/14/12/123011.

[18] Andrew J. Landahl and Ciaran Ryan-Anderson, *Quantum computing by color-code lattice surgery*, 2014. arXiv: 1407.5103 [quant-ph].

[19] Daniel Litinski, "A game of surface codes: Large-scale quantum computing with lattice surgery," *Quantum*, vol. 3, p. 128, Mar. 5, 2019. DOI: 10.22331/q-2019-03-05-128. arXiv: 1808.02892[cond-mat,physics:quant-ph].

[20] H. Bombin and M. A. Martin-Delgado, "Topological quantum distillation," *Phys. Rev. Lett.*, vol. 97, no. 18, p. 180501, Oct. 30, 2006. DOI: 10.1103/PhysRevLett.97.180501. arXiv: quant-ph/0605138.

[21] Aleksander Kubica, "The abcs of the color code: A study of topological quantum codes as toy models for fault-tolerant quantum computation and quantum phases of matter," Ph.D. dissertation, 2018. DOI: 10.7907/059V-MG69.

[22] Aleksander Kubica, Beni Yoshida, and Fernando Pastawski, "Unfolding the color code," *New J. Phys.*, vol. 17, no. 8, p. 083026, Aug. 13, 2015. DOI: 10.1088/1367-2630/17/8/083026. arXiv: 1503.02065[quant-ph].

[23] Jonathan E. Moussa, "Transversal clifford gates on folded surface codes," *Phys. Rev. A*, vol. 94, no. 4, p. 042316, Oct. 12, 2016. DOI: 10.1103/PhysRevA.94.042316. arXiv: 1603.02286[quant-ph].

[24] Sergei Bravyi and Alexei Kitaev, "Universal quantum computation with ideal clifford gates and noisy ancillas," *Phys. Rev. A*, vol. 71, no. 2, p. 022316, Feb. 22, 2005. DOI: 10.1103/PhysRevA.71.022316. arXiv: quant-ph/0403025.

[25] Seok-Hyung Lee *et al.*, *Low-overhead magic state distillation with color codes*, 2025. arXiv: 2409.07707 [quant-ph]. [Online]. Available: https://arxiv.org/abs/2409.07707.

[26] H. Bombin, *An introduction to topological quantum codes*, Nov. 1, 2013. DOI: 10.48550/arXiv.1311.0277. arXiv: 1311.0277[quant-ph].

[27] H. Bombin, "Topological subsystem codes," *Phys. Rev. A*, vol. 81, no. 3, p. 032301, Mar. 3, 2010, Publisher: American Physical Society. DOI: 10.1103/PhysRevA.81.032301.

[28] Daniel Gottesman, *The heisenberg representation of quantum computers*, 1998. arXiv: quant-ph/9807006 [quant-ph]. [Online]. Available: https://arxiv.org/abs/quant-ph/9807006.

[29] Daniel Gottesman, "Fault-tolerant quantum computation with higher-dimensional systems," in *NASA International Conference on Quantum Computing and Quantum Communications*, Springer, 1998, pp. 302–313. [Online]. Available: https://ntrs.nasa.gov/citations/19990026227.

[30] Daniel Herr, Franco Nori, and Simon J. Devitt, "Optimization of lattice surgery is NP-hard," *npj Quantum Inf*, vol. 3, no. 1, p. 35, Sep. 11, 2017. DOI: 10.1038/s41534-017-0035-1. arXiv: 1702.00591[quant-ph].

[31] Tyler LeBlond and Ryan S. Bennink, *Quantum resource comparison for two leading surface code lattice surgery approaches*, 2025. arXiv: 2506.08182 [quant-ph]. [Online]. Available: https://arxiv.org/abs/2506.08182.

[32] Daniel Litinski *et al.*, "Combining topological hardware and topological software: Color code quantum computing with topological superconductor networks," *Phys. Rev. X*, vol. 7, no. 3, p. 031048, Sep. 15, 2017. DOI: 10.1103/PhysRevX.7.031048. arXiv: 1704.01589[cond-mat].

[33] Aleksander Kubica and Nicolas Delfosse, "Efficient color code decoders in $d \geq 2$ dimensions from toric code decoders," *Quantum*, vol. 7, p. 929, Feb. 2023, ISSN: 2521-327X. DOI: 10.22331/q-2023-02-21-929. [Online]. Available: http://dx.doi.org/10.22331/q-2023-02-21-929.

[34] Christopher Chamberland *et al.*, "Triangular color codes on trivalent graphs with flag qubits," *New J. Phys.*, vol. 22, no. 2, p. 023019, Feb. 2020, Publisher: IOP Publishing. DOI: 10.1088/1367-2630/ab68fd.

[35] Kaavya Sahay and Benjamin J Brown, "Decoder for the triangular color code by matching on a möbius strip," *PRX Quantum*, vol. 3, no. 1, 2022. DOI: 10.1103/PRXQuantum.3.010310.

[36] Craig Gidney and Cody Jones, *New circuits and an open source decoder for the color code*, 2023. arXiv: 2312.08813 [quant-ph]. [Online]. Available: https://arxiv.org/abs/2312.08813.

[37] Seok-Hyung Lee, Andrew Li, and Stephen D. Bartlett, "Color code decoder with improved scaling for correcting circuit-level noise," *Quantum*, vol. 9, p. 1609, Jan. 27, 2025. DOI: 10.22331/q-2025-01-27-1609. arXiv: 2404.07482[quant-ph].

[38] Lukas Postler *et al.*, "Demonstration of fault-tolerant universal quantum gate operations," *Nature*, vol. 605, no. 7911, pp. 675–680, 2022. DOI: 10.1038/s41586-022-04721-1.

[39] C. Ryan-Anderson *et al.*, *Implementing fault-tolerant entangling gates on the five-qubit code and the color code*, 2022. arXiv: 2208.01863 [quant-ph]. [Online]. Available: https://arxiv.org/abs/2208.01863.

[40] Pedro Sales Rodriguez *et al.*, *Experimental demonstration of logical magic state distillation*, 2024. arXiv: 2412.15165 [quant-ph]. [Online]. Available: https://arxiv.org/abs/2412.15165.

[41] Ivan Pogorelov *et al.*, "Experimental fault-tolerant code switching," *Nature Physics*, vol. 21, no. 2, pp. 298–303, 2025. DOI: 10.1038/s41567-024-02727-2.

[42] Nathan Lacroix *et al.*, *Scaling and logic in the color code on a superconducting quantum processor*, Dec. 18, 2024. DOI: 10.48550/arXiv.2412.14256. arXiv: 2412.14256[quant-ph].

[43] Oscar Higgott and Craig Gidney, "Sparse Blossom: Correcting a million errors per core second with minimum-weight matching," *Quantum*, vol. 9, p. 1600, Jan. 2025, ISSN: 2521-327X. DOI: 10.22331/q-2025-01-20-1600. [Online]. Available: https://doi.org/10.22331/q-2025-01-20-1600.

[44] Sergey Bravyi *et al.*, "High-threshold and low-overhead fault-tolerant quantum memory," *Nature*, vol. 627, no. 8005, pp. 778–782, Mar. 2024, ISSN: 1476-4687. DOI: 10.1038/s41586-024-07107-7. [Online]. Available: http://dx.doi.org/10.1038/s41586-024-07107-7.

[45] Alexey A Kovalev and Leonid P Pryadko, "Quantum kronecker sum-product low-density parity-check codes with finite rate," *Physical Review A—Atomic, Molecular, and Optical Physics*, vol. 88, no. 1, p. 012 311, 2013. DOI: 10.1103/PhysRevA.88.012311.

[46] Michel H Devoret and Robert J Schoelkopf, "Superconducting circuits for quantum information: An outlook," *Science*, vol. 339, pp. 1169–1174, 2013. DOI: 10.1126/science.1231930.

[47] Alexandre Blais *et al.*, "Circuit quantum electrodynamics," *Reviews of Modern Physics*, vol. 93, p. 025 005, 2021. DOI: 10.1103/RevModPhys.93.025005.

[48] M. Saffman, T. G. Walker, and K. Mølmer, "Quantum information with rydberg atoms," *Reviews of Modern Physics*, vol. 82, pp. 2313–2363, 2010. DOI: 10.1103/RevModPhys.82.2313.

[49] Antoine Browaeys and Thierry Lahaye, "Many-body physics with individually controlled rydberg atoms," *Nature Physics*, vol. 16, pp. 132–142, 2020. DOI: 10.1038/s41567-019-0733-z.

[50] J. I. Cirac and P. Zoller, "Quantum computations with cold trapped ions," *Physical Review Letters*, vol. 74, no. 20, pp. 4091–4094, 1995. DOI: 10.1103/physrevlett.74.4091.

[51] D. Leibfried *et al.*, "Quantum dynamics of single trapped ions," *Reviews of Modern Physics*, vol. 75, no. 1, pp. 281–324, 2003. DOI: 10.1103/revmodphys.75.281.

[52] Julia Chuzhoy and David H. K. Kim, "On approximating node-disjoint paths in grids," in *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2015)*, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015, pp. 187–211. DOI: 10.4230/LIPIcs.APPROX-RANDOM.2015.187.

[53] Julia Chuzhoy, David H. K. Kim, and Shi Li, "Improved approximation for node-disjoint paths in planar graphs," in *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, Association for Computing Machinery, Jun. 19, 2016, pp. 556–569. DOI: 10.1145/2897518.2897538.

[54] C De Bacco *et al.*, "Shortest node-disjoint paths on random graphs," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2014, no. 7, P07009, Jul. 2014. DOI: 10.1088/1742-5468/2014/07/P07009. [Online]. Available: https://dx.doi.org/10.1088/1742-5468/2014/07/P07009.

[55] "Nature-inspired optimization algorithms," in *Nature-Inspired Optimization Algorithms*, Elsevier, Jan. 1, 2014, p. i. DOI: 10.1016/B978-0-12-416743-8.00016-6.

[56] *Handbook of Metaheuristics*, vol. 272, Springer International Publishing, 2019. DOI: 10.1007/978-3-319-91086-4.

[57] Sheldon H. Jacobson and Enver Yücesan, "Analyzing the performance of generalized hill climbing algorithms," *Journal of Heuristics*, vol. 10, no. 4, pp. 387–405, Jul. 1, 2004. DOI: 10.1023/B:HEUR.0000034712.48917.a9.

[58] Emrah Hancer, "Artificial bee colony: Theory, literature review, and application in image segmentation," in *Recent Advances on Memetic Algorithms and its Applications in Image Processing*, Springer, 2020, pp. 47–67. DOI: 10.1007/978-981-15-1362-6_3.

[59] Annu Lambora, Kunal Gupta, and Kriti Chopra, "Genetic algorithm-a literature review," in *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*, 2019, pp. 380–384. DOI: 10.1109/COMITCon.2019.8862255.

[60] Christophe Vuillot *et al.*, "Code deformation and lattice surgery are gauge fixing," *New J. Phys.*, vol. 21, no. 3, p. 033 028, Mar. 2019, Publisher: IOP Publishing. DOI: 10.1088/1367-2630/ab0199.

[61] Markus S. Kesselring *et al.*, "Anyon condensation and the color code," *PRX Quantum*, vol. 5, no. 1, p. 010 342, Mar. 11, 2024, Publisher: American Physical Society. DOI: 10.1103/PRXQuantum.5.010342.

[62] Robert Wille *et al.*, "The mqt handbook : A summary of design automation tools and software for quantum computing," in *2024 IEEE International Conference on Quantum Software (QSW)*, 2024, pp. 1–8. DOI: 10.1109/QSW62656.2024.00013.

[63] James M. Auger *et al.*, "Fault-tolerance thresholds for the surface code with fabrication errors," *Phys. Rev. A*, vol. 96, p. 042 316, 4 Oct. 2017. DOI: 10.1103/PhysRevA.96.042316. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.96.042316.

[64] Dripto M. Debroy *et al.*, *LUCI in the surface code with dropouts*, Oct. 18, 2024. DOI: 10.48550/arXiv.2410.14891. arXiv: 2410.14891.

[65] Catherine Leroux *et al.*, *Snakes and ladders: Adapting the surface code to defects*, Dec. 16, 2024. DOI: 10.48550/arXiv.2412.11504. arXiv: 2412.11504[quant-ph].

[66] Craig Gidney, Noah Shutty, and Cody Jones, *Magic state cultivation: Growing t states as cheap as CNOT gates*, Sep. 26, 2024. DOI: 10.48550/arXiv.2409.17595. arXiv: 2409.17595.