Rescaling decoder for two-dimensional topological quantum color codes on 4.8.8 lattices

Pedro Parrado-Rodríguez,[1] Manuel Rispler,[2,3] and Markus Müller[2,3]
[1]*Department of Physics, College of Science, Swansea University, Singleton Park, Swansea SA2 8PP, United Kingdom*
[2]*Institute for Quantum Information, RWTH Aachen University, D-52056 Aachen, Germany*
[3]*Peter Grünberg Institute, Theoretical Nanoelectronics, Forschungszentrum Jülich, D-52425 Jülich, Germany*

(Received 22 December 2021; revised 3 August 2022; accepted 10 August 2022; published 26 September 2022)

Fault-tolerant quantum computation relies on scaling up quantum error correcting codes in order to suppress the error rate on the encoded quantum states. Topological codes, such as the surface code or color codes, are leading candidates for practical scalable quantum error correction and require efficient and scalable decoders. In this work, we propose and study the efficiency of a decoder for two-dimensional topological color codes on the 4.8.8 lattice (also known as the square-octagon code), by building on the work of Sarvepalli and Raussendorf [Phys. Rev. A **85**, 022317 (2012)], for color codes on hexagonal lattices. The decoder is based on a rescaling approach, in which syndrome information on a part of the qubit lattice is processed locally, and then the lattice is rescaled iteratively to smaller sizes. We find a threshold of 6.0% for code capacity noise.

DOI: 10.1103/PhysRevA.106.032431

I. INTRODUCTION

Quantum error correction (QEC) schemes aim at detecting and correcting errors during storage and processing of quantum information to enable long and reliable quantum computation on scalable quantum processors [1,2]. QEC codes encode logical information in nonlocal degrees of freedom such that the effects of errors can be detected through parity check measurements and reversed before they accumulate. The threshold theorem ensures that the logical error rate can be arbitrarily suppressed by increasing the size of the code, provided that fault-tolerant quantum circuit constructions are used and the physical error rates fall below a given critical threshold [3–5]. The value of the threshold depends on the QEC code, the noise model, and in particular also on the decoder. The latter amounts to our ability to correctly interpret the syndrome, which is given by the collection of error information gathered through measuring the parity checks, in order to apply a correction with a high success probability. Topological QEC codes such as the toric or surface code [6–8] and color codes [9,10] encode the logical information into topological properties of the system, while all parity checks are low-weight measurements involving geometrically local qubits on the lattice. They stand out as the QEC codes with some of the highest known thresholds [2,11] when, e.g., compared to concatenated codes, which makes them attractive candidates for experimental realizations [12–19]. In particular, the seven-qubit color code as the smallest fully functional representative of the family of two-dimensional (2D) color codes has been targeted in a series of experimental QEC advances in ion trap quantum devices: From code state preparation implemented in [12], fault-tolerant stabilizer measurement [20], and repeated QEC cycles [13] as well as fault-tolerant magic-state preparation and injection [16] have been demonstrated recently. Complementary, small surface codes are prominently being pursued in superconducting qubit experiments: There, single weight-four parity measurement [19] and error detecting surface codes [15,18], code-state preparations of larger surface code states [14], and the very recent leap towards repetitive QEC cycles in a Surface-17 architecture [21] have been shown.

To execute operations on the encoded logical qubits, one has to devise logical gates, which is the subject matter of the theory of fault tolerance [4,5]. Here, the challenge is to make sure that the synthesized gates act in such a way that they cannot inadvertently spread errors beyond the scope of what the QEC code is able to correct. This spreading is most easily avoided by acting on the data qubits within the code block separately, i.e., using no entangling operations at all, which is known as a transversal implementation of a logical gate. However, the possibility of implementing logical gates transversally is limited: On the lower end it depends on the QEC code and on the upper end it is known to be impossible to implement a universal gate set fault tolerantly by a no-go theorem by Eastin and Knill [22]. In this regard, the color code on the 4.8.8 lattice is particularly interesting because it allows for the transversal implementation of the entire Clifford group, which distinguishes it from surface codes or color codes on hexagonal lattices. It is therefore optimal in the sense that adding any other (non-Clifford) gate would render the gate set universal and hence violate the no-go theorem. The remaining non-Clifford gate is typically synthesized by other means such as magic-state distillation and injection [23–26], for which color codes are also particularly relevant [9,27], as underlined by recent fault-tolerant implementations of Clifford gates [13] and a non-Clifford T gate [16].

To operate the color code, in particular codes of larger distance, it is vital to have an efficient decoding algorithm. This decoder should on the one hand perform as well as possible in terms of proposing a (near-)optimal recovery operation. This

performance is reflected in the threshold value, and it can at least for simple noise models be benchmarked against known upper bounds on decoding performance obtained through mapping the quantum error correction problem onto a classical statistical-mechanical model [8,28,29]. On the other hand, this accuracy of decoding has to be balanced with the time it takes to run the decoding algorithm: While for a quantum memory, it is potentially fine to keep a backlog of measured error syndromes and figure out the correction in classical post-processing later, this is not the case anymore once we start to perform logical quantum computations: Here the intermediate state of the computation will depend on the decoder outcome, such that the quantum computation may have to wait for the decoding algorithm to finish, time during which of course new errors accumulate. The development of decoders for color codes has been and still is an active field of research [30–41]. For the case of data qubit noise, also known as code capacity noise, the decoder upper bound threshold is known to be 10.9% [28], which interestingly seems to be independent of the color code lattice. The decoder with the best performance known in terms of threshold is the restriction decoder, which achieves 10.2%, when using minimum weight perfect matching (MWPM) as a subroutine [38]. The runtime complexity of MWPM as a function of the number of qubits N is $O(N^4)$ for a straightforward implementation of the original Blossom algorithm [42,43], which can be optimized to $O(N^{2.5})$ using recent advancements [44,45]. Instead of MWPM, one could also use the union-find decoder [35], which has a runtime complexity of $O(N\alpha(N))$, where $\alpha(x)$ is the inverse Ackermann function and extremely slowly growing, which equips this decoder with almost linear time complexity. Alternative approaches have explored as well the use of machine learning to solve the decoding problem [33,39–41]. In this work, we explore a different decoding paradigm based on iterative rescaling and partial decoding known as renormalization-group (RG) decoding. This paradigm was introduced in [46] for surface codes and extended to the color code on a hexagonal (6.6.6) lattice in [30]. Owing to the rescaling feature, these algorithms have a runtime complexity of $O(N\log(N))$, as the rescaling of the lattice can be done in linear time $O(N)$ since it is based on local operations, and the number of iterations of the rescaling process grows as $O(\log(N))$. The local nature of the rescaling algorithm means it can be parallelized, achieving an overall scaling of $O(\log(N))$, which holds the potential for drastic improvements in decoding runtime.

The paper is organized as follows. First, in Sec. II we introduce central concepts about 2D color codes and the decoder proposed in [30], as well as some of the key ideas and concepts used in the algorithm. Then, in Sec. III we explain the details about the decoder and the different steps: Belief propagation, splitting of the stabilizers, and rescaling of the cells. In Sec. IV, we estimate the threshold for code capacity noise, obtained from Monte Carlo simulations for several lattice sizes. In Sec. V, we conclude and present some directions for future extensions of the decoder.

II. BACKGROUND

Color codes are stabilizer codes [9,10,47] defined on face-three-colorable trivalent graphs, i.e., graphs where all vertices have degree three and faces can be colored with three colors such that neighboring faces never have the same color (see, e.g., Fig. 1 for the case of the 4.8.8 lattice coloring). Qubits are identified with the vertices and each face i of the graph defines two stabilizer generators $S_X^{(i)}$ and $S_Z^{(i)}$ involving all vertices in the boundary of the respective face [9,47]. These stabilizers involve purely X or Z Pauli operators, which render color codes part of the Calderbank-Shor-Steane (CSS) code family [48,49]. Note that stabilizers are guaranteed to commute since both faces as well as the boundary of faces on a trivalent three-colorable graph always contain an even number of vertices. The code space is defined as the simultaneous $+1$ eigenspace of all stabilizers. If we consider periodic boundary conditions, two independent qubits can be encoded in each of the two nontrivial loops of the resulting torus (see Fig. 1), leading to a total of four logical qubits. Being part of the CSS stabilizer codes, it is possible to detect and correct phase-flip and bit-flip errors separately by using the syndrome from the X and Z stabilizers, respectively. Throughout this work we focus on independent bit- and phase-flip noise. One of the symmetries of the color code is that it is self-dual under exchanging X and Z stabilizers, which in particular implies that, in order to obtain the code capacity threshold, we can focus purely on bit-flip errors.

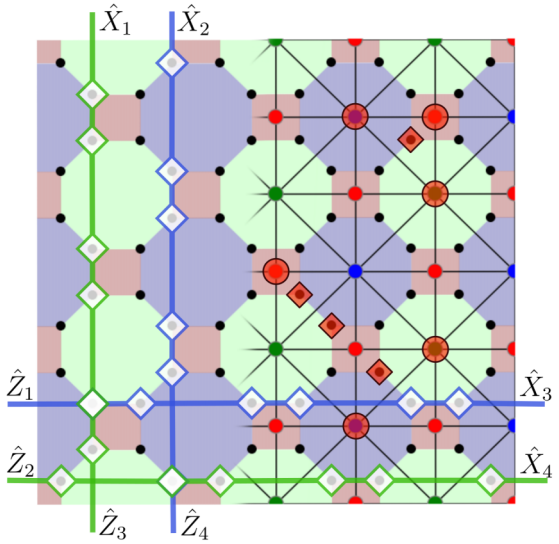

FIG. 1. Representation of the color code 4.8.8 lattice. On the primal lattice (left), qubits are represented by black vertices, and the stabilizers S_Z and S_X are represented by the colored plaquettes, which apply a parity check over the qubits on the vertices. In the reciprocal lattice (right), the qubits are represented by triangles, and stabilizers are represented by colored vertices, which apply a parity check over the qubits for which the stabilizer is a vertex. The logical operators are strings of Pauli operators that extend over the torus in a nontrivial way. On the color code lattice on a torus, we have two independent logical operators for each nontrivial loop, i.e., four logical qubits. The support of the logical operators $\hat{X}_i$ and $\hat{Z}_i$ is represented by the blue and green lines, which represent the four nontrivial loops on the toric color code lattice. To illustrate the effect of errors in the lattice, we display an example of four physical bit-flip errors (qubits are marked with red diamonds), and the corresponding stabilizer excitations (stabilizers are marked with red circles).

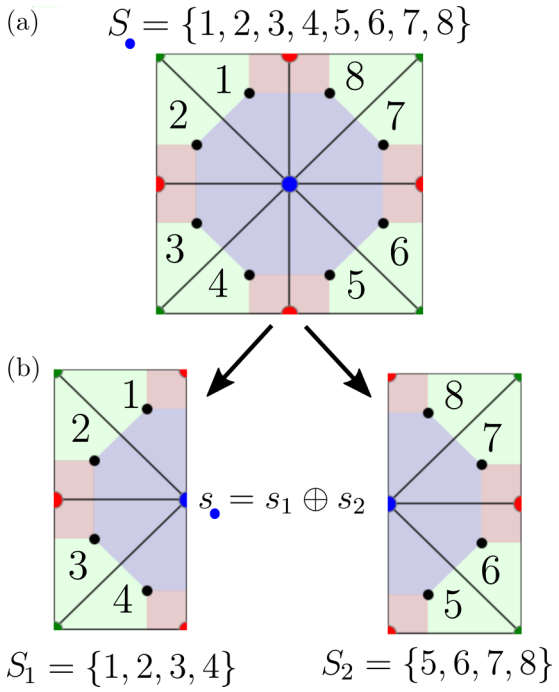

FIG. 2. Example of the splitting of a stabilizer. (a) The original stabilizer S, represented by the blue vertex, realizes a parity check on the qubits 1 to 8, both in the Z and the X basis. Qubits are represented by black dots inside the triangles. (b) We can split the stabilizer into the half-stabilizers S_1 and S_2. Each half-stabilizer represents the parity of its set of qubits $S_1 = \{1, 2, 3, 4\}$, $S_2 = \{5, 6, 7, 8\}$. The binary sum $\oplus$ of the parities of both half-stabilizers thus fulfills the condition $s = s_1 \oplus s_2$. We call the choice of parity assignments of half-stabilizers for a given stabilizer a *splitting*. For notation, we use uppercase letters to refer to the stabilizer operators, and lowercase for their parity value, which is represented as a 0 for even parity and 1 for odd parity.

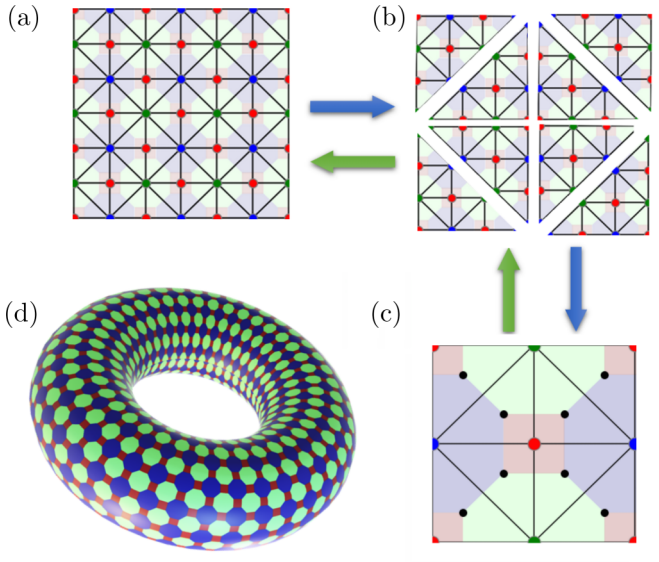

FIG. 3. Sketch of a rescaling step in the 4.8.8 color code lattice. Qubits are represented by triangular faces, and the colored vertices represent the stabilizers. The initial lattice (a) can be split into multiple cells (b), which can be decoded locally. Each triangular cell can then be mapped into a single effective qubit in a rescaled lattice (c). This process of rescaling can be repeated until the final lattice is small enough to apply a brute-force decoding. Corrections on the smaller lattices can then be backpropagated to the original lattice, finding the final correction. (d) In this work we consider a 4.8.8 color code lattice, where each qubit is involved in one 4-qubit stabilizer associated to a square plaquette and in two 8-qubit stabilizers on octagonal plaquettes, with periodic boundary conditions. The lattices considered host codes with parameters $[[n, k, d]] = [[8 \cdot 9^m, 4, 2 \cdot 3^m]]$, where n is the number of qubits in the lattice, k the number of logical qubits, d is the distance of the code, and the integer m denotes the number of rescaling steps. The blue arrows represent the initial order of operations, in which the lattice is rescaled until the smallest system size is reached. Green arrows represent the backpropagation process, from the smallest rescaled lattice back to the original code.

III. DECODER ALGORITHM

A. Introduction to the decoder's approach

In this section, we introduce a qualitative description of the decoder algorithm. With this outline, we aim to frame the general picture of the decoder and the basic underlying principle. In the following subsequent sections, we study the details of each particular step.

The main idea behind the decoder is to split the code lattice into small cells. These cells can be decoded locally and the result can then be merged into the global decoding decision. A key problem to this decoding ansatz is that when trying to divide the lattice into cells, one invariably has to cut through some stabilizers that are shared between different cells. The solution to that problem is to split these stabilizers into two, which we will hence refer to as *half-stabilizers* in the following, so that one can decode each cell individually by using the local syndrome from the half-stabilizer that applies to the cell (Fig. 2). Thus, the way in which each stabilizer is split between the cells determines the correction applied on the cells. After applying a local decoding on each cell, this cell can be treated as an individual effective qubit (two when using a square cell) of a now rescaled color code, where we have rescaled the lattice to a smaller version of itself. This process can then be repeated, ultimately leading to a lattice small enough so that a brute force decoding can be applied. This rescaling process is illustrated in Figs. 3(a)–3(c) for the 2D color code on the square-octagon lattice.

During the algorithm, each qubit is assigned an error probability. This value can be initialized according to the error model, i.e., in our case every qubit *a priori* has error probability p. This value is then updated using a belief propagation algorithm [50,51] to propagate information about the observed syndrome to the qubits. Using these error rates, the next step of the algorithm is to split the stabilizers that are shared between two cells into half-stabilizers. In order to be consistent with the observed syndrome, e.g., a stabilizer operator S with parity $s = 1$ (odd) can be split in two different ways: $(s_A, s_B) = (1, 0)$ or $(s_A, s_B) = (0, 1)$, so that the total parity of the sum of the half-stabilizers corresponds to the original value of the stabilizer $s = s_A \oplus s_B$. The choice of the splitting configuration determines the syndrome assigned to each individual cell. This syndrome is used by the decoder for a local cell to find a suitable correction within the cell. To find the best choice of splitting for the stabilizers (i.e., the one that

Algorithm 1. Rescaling decoder

Data: Measured syndrome
Result: Recovery operation
while *Lattice larger than minimum size* **do**
  Estimate the error probability of each qubit (1.)
  Split the code into cells (2.)
  Split the stabilizer values between the cells (3.)
  Apply a local decoder on each cell (4.)
  Rescale the cells to effective qubits (5.)
end
Apply a brute force decoder on the final lattice (6.)
Back-propagate the errors to the original lattice (7.)

leads to the recovery operation for the most probable error), we apply a series of splitting updates, which assign a likelihood to each choice depending on the local information of the neighboring cells. We use a suitable convergence criterion, after which we then fix the configuration of splitting choices for the stabilizers, such that each cell can now be decoded using only the information from the local syndrome. Using the probabilities of the different error configurations compatible with that local syndrome, we can then rescale the cells to effective qubits and assign a new error probability to them. With this last step, we complete the rescaling cycle, which thereby leads to a smaller version of the lattice.

The challenge of defining a rescaling algorithm on the 4.8.8 lattice lies in identifying a valid choice for the form and size of the minimal cell. Due to the conditions that a cell must fulfill to be rescalable to an effective qubit, the minimal cell on the 4.8.8 lattice is more than twice times bigger and shares two stabilizers with each neighboring cell when compared to the simpler case of the 6.6.6 lattice, where neighboring cells share only a single stabilizer. This change requires the splitting algorithm that splits the stabilizers between cells to update splittings pairwise instead of individually.

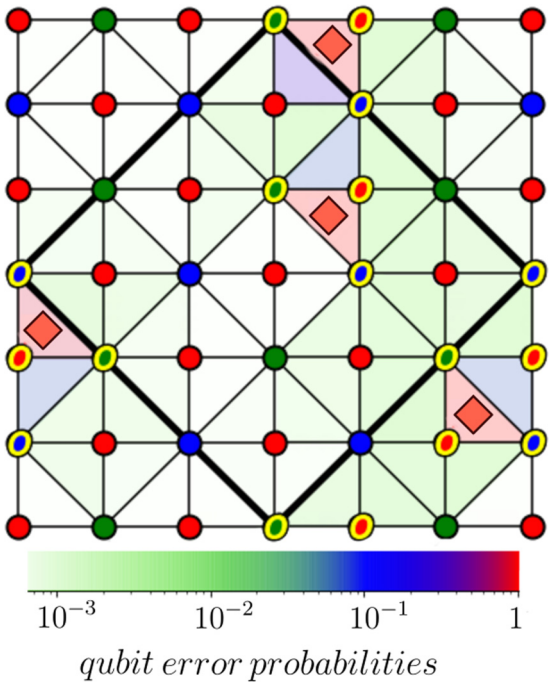

FIG. 4. Example of updated qubit error probabilities after belief propagation. We show a 72-qubit color code with periodic boundary conditions, where qubits are represented by the colored triangles, and stabilizers are represented by the colored circles in red, blue, and green. The nontrivial stabilizers are represented with a yellow border, and the physical errors are represented by red diamonds. During belief propagation, stabilizers and qubits share information locally, which leads to a refined estimate of the qubit error probabilities. In this example, the qubit error probabilities are represented by the coloring of the triangles, which correspond to the qubits, ranging from white to red as indicated in the color bar. The thicker black solid line has been drawn as a guide to the eye when comparing the figure with Fig. 5, after dividing the lattice into square cells.

B. Decoder algorithm for the 4.8.8 lattice

In the following we present the decoding algorithm in the form of high-level pseudocode and discuss the main steps the decoder, which uses the measured syndrome and a prior estimate of the qubit error rate as input:

(1) We estimate the error probability of the qubits using the information from the syndrome (Fig. 4). This is done in two steps. First, we apply a belief propagation algorithm, through which stabilizers and qubits share information via message passing (see Sec. III D). Second, we update the probabilities of the qubits around the corners of the cells depending on the parity of the stabilizer in the corner [30].

(2) We subdivide the code into multiple cells. The stabilizers S between two adjacent cells are split, while preserving the parity of the sum: $s_a \oplus s_b = s$. This allows for two ways of splitting a stabilizer (Fig. 5).

(3) We find a configuration for the stabilizer splittings (Fig. 2) and assign a probability for each splitting choice (details will be provided in Sec. III F): (3a) We compute the initial splitting probabilities using the probabilities of the qubits involved. (3b) We update the probabilities of the different splitting choices (Fig. 6) using local information, as described in Sec. III F. The update is applied simultaneously on all splittings of the lattice. Several global update steps are used until convergence is reached for the split choice.

(4) After fixing the half-stabilizers, each cell has a local syndrome. Using a lookup table (a precomputed list containing all possible errors compatible with the syndrome), we can decode the cells and find a correction (Fig. 7). At this step, we ignore the syndrome of the corners, as their parity will become the syndrome of the rescaled lattice. The parity of those stabilizers will thus be addressed in the decoding process applied to the subsequent rescaled lattices.

(5) The cells can now be rescaled to effective qubits (Fig. 8). We can compute the error probability of the rescaled qubits as the probability of applying a logical operator in the cell (see Sec. III G).

(6) We create a new code by rescaling each cell on the lattice to effective qubits. If the code is small enough, we can decode the lattice by finding the most probable error with a lookup table. Otherwise, we repeat steps 1–5 for the new code.

(7) Once we have the corrections on the smallest lattices, we can backpropagate the corrections to the original lattice to obtain the final recovery operation.

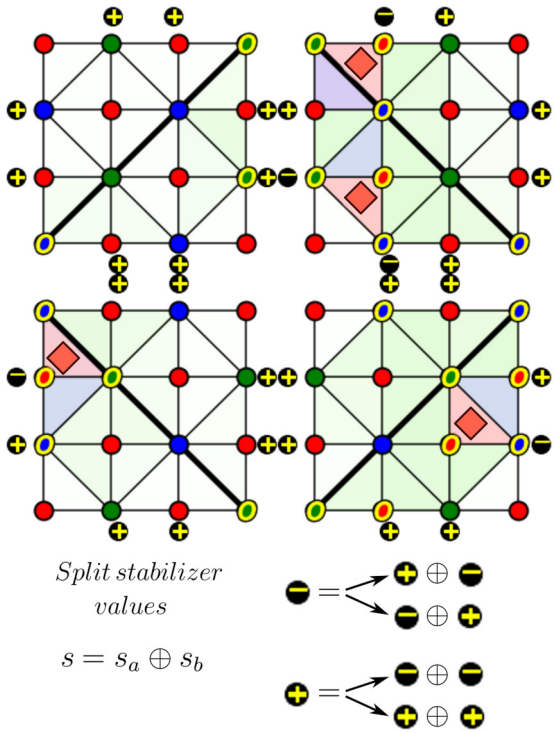

FIG. 5. Splitting of the lattice into cells. We show the same code as in Fig. 4 after splitting the lattice into four different cells. The stabilizers at the boundary between cells need to be split into half-stabilizers, as in Fig. 2. The parity of each half-stabilizer s_a and s_b is represented as a 0 for even parity and a 1 for odd parity, and the binary sum $\oplus$ of the parity of both half-stabilizers needs to equal the parity s of the original stabilizer S. For each stabilizer, there are two alternative splittings into half-stabilizers. The black solid line in the cells has been drawn to distinguish the two effective qubits that result from the rescaling of the cells. The physical errors in this particular example have been marked with red diamonds, and the excited stabilizers are marked with a yellow border.

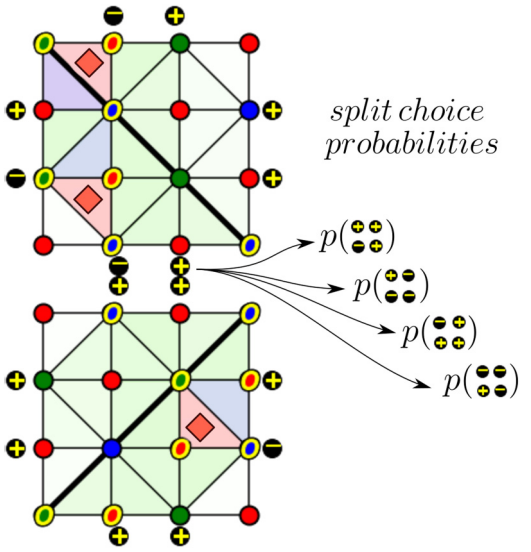

FIG. 6. Sketch of the splitting of two stabilizers. A cell shares two stabilizers with each of the neighboring cells. There are four different ways to split a pair of stabilizers in half-stabilizers. During the splitting updates, we compute an estimate of the probability of each of the four splitting choices. The signs shown in the dark circles correspond to the parity of the half-stabilizers. The two cells shown in the figure are part of the lattice shown in Fig. 4.

After applying the recovery operation, we can check in our simulations if the combination of the error and our correction corresponds to the application of a logical operator, and thus the occurrence of a logical error, on any of the four encoded qubits in the lattice. This can be easily done by checking the parity of the qubits along each of the four operators drawn in Fig. 1, i.e., for instance the parity of the number of bit flips along the qubits in the support of $\hat{Z}_i$ determines if a logical $\hat{X}_i$ operator was applied.

It is important to notice that the smallest lattice size and the size of the unit cell determine the code size (total number of qubits) for which the decoder can be applied. In this work, we consider the minimum lattice size of 8 qubits. Thus, the lattice sizes for which the decoder can be applied depend on the number of rescaling steps m as $[[n, k, d]] = [[8 \times 9^m, 4, 2 \times 3^m]]$, where n is the number of qubits in the lattice, k the number of logical qubits, and d is the distance of the code. The number of qubits in the lattice begins with 8 qubits as the minimum lattice size (see Fig. 3), and each rescaling step introduces a factor of 9 in the number of qubits (see Fig. 8). Similarly, the code distance d of the smallest lattice size is 2, and each rescaling step increases the logical distance by a factor of 3 (see logical operators in Fig. 1).

C. Minimal cell

In order to choose an appropriate cell, these are the conditions that need to be fulfilled (cf. [30]):

(1) A logical operator can be defined for the cell.

(2) There exists a valid correction for every possible syndrome.

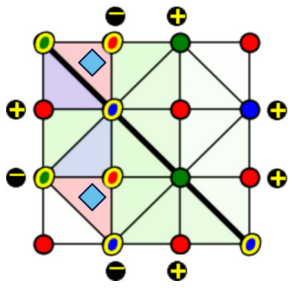

FIG. 7. Local decoding of cells. A single cell can be decoded using the syndrome from the half-stabilizers in the boundary and the parity of the bulk stabilizers. To find the correction, a brute-force decoder is applied, which finds the most probable correction (the qubits in this suggested correction are marked by blue diamonds) using the estimates of the error probabilities of each qubit. The parity of the corner stabilizers is ignored during the decoding of the cell, but the parity of these stabilizers is updated depending on the corrections applied. In this particular example, the parity of the stabilizer in the upper left corner would be changed by the proposed correction, which will be taken into account in the rescaled lattice. The signs in the dark circles represent the parity of the half-stabilizers in this cell.

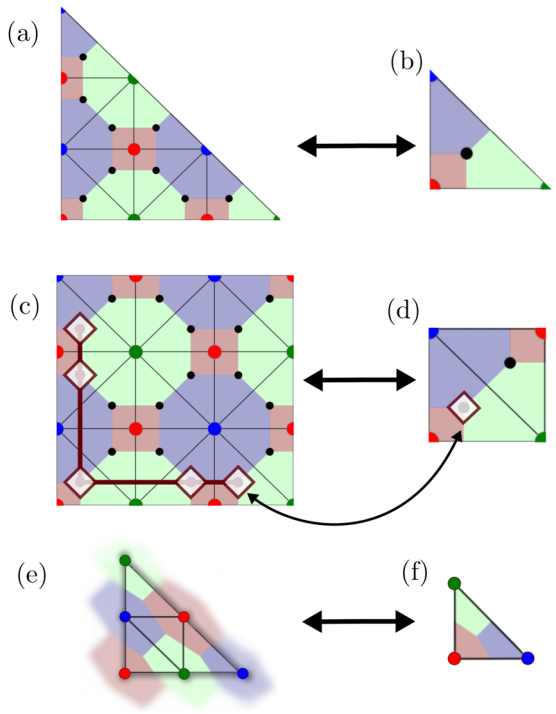

FIG. 8. Rescaling cells in the 4.8.8 lattice. (a) Minimal cell for the rescaling decoder in the 4.8.8 lattice. This 9-qubit cell can be mapped to a single effective qubit (b) during the rescaling of the lattice. (c) A square cell can be used during the decoder process to reduce the number of stabilizer splittings. This cell is mapped to two effective qubits, as shown in (d). An error on the rescaled qubit can be backpropagated (curved double arrow) to the original lattice by applying the effective logical operator of the cell: The qubits marked in (c) with white squares represent the logical operator of the effective qubit marked in (d). By applying this logical operator, the parity of the corner stabilizers corresponding to the effective qubits is changed, while preserving the parity of the rest of the stabilizers in the cell. (e) For comparison, we show the minimal cell for the rescaling decoder in the hexagonal lattice [30], with only four qubits. This cell can be rescaled to an effective qubit (f).

(3) The cell can map the entire code to a smaller version of itself.

A valid cell for which these conditions are fulfilled can be found by choosing a triangular cell for which the three corners are of different color.

The minimal cell that fulfills these conditions in the 4.8.8 lattice is the 9-qubit cell represented in Fig. 8. The cell shares two split stabilizers with each neighboring cell, and an additional stabilizer is contained entirely within the cell. The syndrome for this cell contains seven stabilizer measurements (one red stabilizer inside the cell, and six half-stabilizers on the boundaries), and four different corrections are possible for each possible syndrome due to the logical operator and the stabilizer contained within the cell.

Using this cell, it is possible to map a 4.8.8 lattice to a smaller version of itself, reducing the number of qubits by a factor of 9 with each step. In Fig. 3, a single step of splitting the lattice and rescaling the cells is shown. In practice, it can be convenient to combine two triangular cells into a square cell consisting of two effective qubits (as shown in Fig. 7). By using square cells, we reduce the number of stabilizer splittings needed in each decoding step. This approach also proved to achieve better correction capabilities in the hexagonal lattice [30]. In this work, we study the performance of the decoder using these square cells.

D. Belief propagation

In general, belief propagation (BP) is a method to compute or approximate marginal probabilities of multivariate probability distributions [52]. BP is formulated in a graph-theoretical setting, where the multivariate distribution is represented as a factor graph, which is a bipartite graph with vertices representing (a) the variables and (b) the factors such that edges between the two indicate a functional dependence of the latter on the former. In an error correction setting, the role of variables is taken by the qubits and the role of factors by the parity checks, i.e., the stabilizer generators and the multivariate probability distribution is the error model (e.g., independent bit-flip noise, depolarizing noise, etc.). The task of marginalization, i.e., of summing over all variables except the one we are interested in, is in general exponentially hard since there are exponentially many configurations to sum over. However, the ingenuity of BP (or more precisely the sum-product message passing algorithm [50]) lies in streamlining the summations into subtasks, called messages, which are computed locally and then passed forward on the edges of the factor graph. If the structure of the factor graph is treelike (i.e., contains no loops), this renders the BP algorithm exact, i.e., it computes the exact marginals on all variables in a number of steps proportional to the depth of the tree. While most graphs are not treelike, it turns out that in practice BP can still yield good approximations to the marginals as long as the underlying graph does not contain too many loops; as a decoder for classical so-called low-density parity check (LDPC) codes it performs close to optimal [53].

In general, as a standalone decoder, BP is known to fail when trying to decode topological QEC codes, which is attributed to the highly degenerate nature of topological codes [54]: Degeneracy refers to the fact that one syndrome can be corrected in many distinct ways that are nevertheless logically equivalent. Here, this problem does not arise since we abort BP iterations before running into loops. For the decoder investigated here, BP serves as a means to compute estimates of the qubit error probabilities from the stabilizer measurement outcomes, which then acts as input to the remaining processing steps.

The main idea behind the algorithm is the following:

(i) At the beginning, each qubit has an estimate of its error probability. Each syndrome has a parity value.

(ii) Each qubit sends a message to the neighboring stabilizers with information about its error probability.

(iii) With the information from the error probabilities of the qubits involved in a parity measurement, each stabilizer can send an updated estimate of the error probability for each qubit. For example, if the parity is even, this means that an error in a qubit implies an odd parity in the rest of the qubits involved in the stabilizer. Therefore, the stabilizer sends a message to each qubit with information on the probability of *the rest* of the qubits having an odd number of errors.

Similarly, if the parity is odd, the message will contain information about the probability of an even number of events in the remaining qubits.

(iv) With the updated information from the stabilizers, the qubits send an updated message to the neighboring stabilizers, repeating the process. With each cycle of message passing, the information spreads through the code.

(v) After a given number of iterations, the algorithm stops and utilizes the information from the messages to update the estimate of the error probability of the qubits (see Fig. 4).

Now, let us get into the details of the mathematics behind the algorithm. The building block of belief propagation is Bayes theorem, applied to update the probability of an error in a qubit with the information from the syndrome:

$$p(q_i = 1|\{s\}) = \frac{p(q_i = 1)}{p(\{s\})} p(\{s\}|q_i = 1), \tag{1}$$

$$p(q_i = 0|\{s\}) = \frac{p(q_i = 0)}{p(\{s\})} p(\{s\}|q_i = 0), \tag{2}$$

where $p(q_i = 1|\{s\})$ is the probability of an error on qubit q_i given the syndrome $\{s\}$, $p(q_i = 1)$ is the prior, or the previous information on the qubit error rate, $p(\{s\}|q_i)$ is the probability of the given syndrome assuming an error on qubit q_i (usually called the likelihood ratio), and $p(\{s\})$ is the probability of the syndrome event. This last term is effectively an unknown normalization factor, that would be hard to compute. However, we can cancel that term if we compute instead the quotient of both quantities. Thus, we can write

$$\begin{aligned} \frac{p(q_i = 0|\{s\})}{p(q_i = 1|\{s\})} &= \frac{p(q_i = 0)}{p(q_i = 1)} \frac{p(\{s\}|q_i = 0)}{p(\{s\}|q_i = 1)} \\ &= \frac{1 - p_i}{p_i} \prod_j \frac{p(s_j|q_i = 0)}{p(s_j|q_i = 1)}, \end{aligned} \tag{3}$$

where the product in j compiles information for all the parity s_j of the stabilizers affecting the qubit. This quotient will correspond to the information sent from each stabilizer j to the qubit, and can be written, depending on the parity of the stabilizer, as

$$M_{s=0\to q} = \frac{p(s = 0|q_i = 0)}{p(s = 0|q_i = 1)} = \frac{p(\text{even})}{p(\text{odd})}, \tag{4}$$

$$M_{s=1\to q} = \frac{p(s = 1|q_i = 0)}{p(s = 1|q_i = 1)} = \frac{p(\text{odd})}{p(\text{even})}. \tag{5}$$

Here, $p(\text{even})$ and $p(\text{odd})$ correspond to the probability of an even or odd number of error events happening on the remaining qubits involved in the parity check. Therefore, we can compute the messages from the stabilizers by computing the probability of an even or odd number of error events happening on the remaining qubits. This probability can be written using a simple formula to find the probability of an even or odd number of error events, given the probabilities p_i of each individual event:

$$p(\text{even}|\{p_i\}) = \frac{1}{2} + \frac{1}{2} \prod_i (1 - 2p_i), \tag{6}$$

$$p(\text{odd}|\{p_i\}) = \frac{1}{2} - \frac{1}{2} \prod_i (1 - 2p_i). \tag{7}$$

Then, the messages from the qubits to the stabilizers are updated using the information from the stabilizers. The messages to the stabilizers in the next cycle are

$$M_{q\to s_i} = \frac{1 - p}{p} \prod_{\substack{j\in N(q) \\ j\neq i}} M_{s_j\to q}, \tag{8}$$

where $N(q)$ refers to the three stabilizers in the neighborhood of qubit q. In this way, the information from the qubits and stabilizer spreads through the code, leading to an improved estimation of the error probabilities for the qubits. However, in the numerical simulations, the direct use of Eqs. (4), (5), and (8) leads to numerical problems, when the error probabilities of the qubits assume values close to zero. To solve this problem, it is useful to work with the logarithm-likelihood ratio. The equations for the messages can then be simplified, as shown in the Appendix of [55]. With that formulation, the initial message from qubits to stabilizers is

$$M^{(0)}_{q\to s} = \ln \frac{1 - p_q}{p_q}. \tag{9}$$

The equation for the messages from stabilizers to qubits can be written as

$$M_{s_i\to q_j} = (1 - 2s_i)\, 2 \tanh^{-1} \left[\prod_{\substack{k\in N(s_i) \\ k\neq j}} \tanh(M_{q_k\to s_i}/2) \right], \tag{10}$$

where s_i represents the parity of the stabilizer measurement. Similarly, the equation for messages from qubits to stabilizers is simplified as

$$M_{q_i\to s_j} = \sum_{\substack{s_k\in N(q_i) \\ k\neq j}} M_{s_k\to q_i} + \ln \tfrac{1-p_i}{p_i}. \tag{11}$$

After the last iteration of message passing, we can obtain the updated estimate of the error probability. The equation for the final estimate of the error probability can be written as

$$p_i^{\text{updated}} = \left[1 + \exp\left(\ln \tfrac{1-p_i}{p_i} + \sum_j M_{s_j\to q_i} \right) \right]^{-1}. \tag{12}$$

E. Corner updates

During the decoding process, the splitting updates and the cell decoder ignore the syndrome of the stabilizers located in the corners of the cells. This can be problematic for some error cases, as the decoder would not be able to distinguish cases that differ only on the syndrome of the corner stabilizers. For this reason, it is useful to use the information from the syndrome in the corners of the cells to modify the error probability of the qubits near the corners.

Given the parity s_i of a given corner stabilizer, we need the parity of the qubits in the cell plus the parity of the qubits outside to be equal to s_i. Thus, for a given qubit j in the cell with a prior estimate p_j, we can compute the updated error probability of an error on qubit j given the parity of the corner

stabilizer $p(j|s_i, p_e)$ as

$$p(j|s_i = 0) = \frac{p_j p_e(\text{odd})}{p_j p_e(\text{odd}) + (1-p_j) p_e(\text{even})}, \tag{13}$$

$$p(j|s_i = 1) = \frac{p_j p_e(\text{even})}{p_j p_e(\text{even}) + (1-p_j) p_e(\text{odd})}, \tag{14}$$

where p_e corresponds to the probability of the qubits outside the cell to have a total parity that is even or odd. We can use Eqs. (6) and (7) to compute the probability of n qubits having a total even or odd parity.

F. Splitting of the stabilizers

The basic cells on the 4.8.8 lattice share two splittings with each neighboring cell. This leads to a new phenomenon compared to the simpler case of the 6.6.6 lattice, as the probability of splitting of each stabilizer now depends on the splitting configuration of the other splitting shared with that cell. Therefore, we need to consider the joint probabilities, which take into account simultaneously the probability of a splitting configuration of the two stabilizers s_1 and s_2 shared between the cells.

This complication in the way we store and compute the probabilities of splitting configurations will affect most of the other steps of the algorithm, as splittings need to be considered in pairs, taking into account the joint probabilities:

(1) The first estimate for the joint probabilities is obtained from the probabilities of the qubits involved in each stabilizer having an even or odd number of errors. This does not take into account the rest of the splittings in the cell, only the probabilities of the qubits involved in the splitting.

(2) During the splitting updates, we will update the joint probabilities for the splitting configurations. Each step will update our estimate for the probabilities of each splitting using the information from the two cells involved.

(3) In the rescaling step, the probability of error in the rescaled qubit involves the probabilities of the different splitting configurations. This means that the error probability of the rescaled qubit will take into account the uncertainty in the splitting choice.

Now, let us get into the details of the equations needed to compute the updated probabilities for the splittings. The first equation we need to consider corresponds to the probability of a given error configuration. Assuming our estimate of the error probabilities of each individual qubit, we can compute the probability of an error configuration $C = \{e_0, e_1, \ldots e_{n_q-1}\}$ (where $e_i = 0, 1$ represents no error or an error on qubit i, and n_q is the number of qubits in a cell) by adding a factor of p_i for each qubit with an error, and a factor of $1 - p_i$ for each qubit without an error,

$$p(C) = \prod_{i=0}^{n_q-1} p_i^{e_i} (1-p_i)^{1-e_i}, \tag{15}$$

where we assumed no correlation between the error probability p_i of each qubit. During the rescaling process, the use of square cells with two qubits will give us access to the joint probabilities of each qubit pair that belongs to the same cell. We can use this additional information about the correlations between different qubits to modify this equation as

$$p(C) = \prod_{i=0}^{n_q/2-1} p(e_i, e_{i+1}), \tag{16}$$

where $p(e_i, e_{i+1})$ corresponds to the element of the joint probabilities corresponding to our prior knowledge from the error probabilities of qubits i and $i + 1$.

Using the probability of a given error configuration, we can compute the probability of all error configurations that are compatible with a given syndrome as the sum of $p(C)$ over all configurations C compatible with the syndrome $\{s\} = \{s_0, s_1 \ldots s_{n_S-1}\}$ ($s_i = 0, 1$ represents the even or odd parity of the stabilizer or half-stabilizer i):

$$p(e|\{s\}) = \sum_C p(C). \tag{17}$$

For the 18-qubit square cell that we use for the 4.8.8 color code, each syndrome includes eight half-stabilizers (two on each side of the square) and four additional stabilizer measurements corresponding to the bulk stabilizers inside the cell. Here, as stated earlier, the parity of the corners is ignored, as the parity of the corner stabilizers will be solved in the following rescaling steps. For the square cell, there are two logical operators that can be defined, one for each of the logical qubits to which the cell will be rescaled. This means that there will be 2^2 possible classes of configurations compatible with any syndrome in the cell. In addition to that, all possible product combinations with the four bulk stabilizers lead to equivalent error configurations. This leads to a total of $2^{2+4} = 64$ possible configurations over which is to be summed according to Eq. (17).

While using all 64 configurations would lead to more accurate estimations, this also involves a high constant-factor overhead in the computational time required by the decoder. Thus, to improve the performance of the algorithm with regard to computing time, we approximate Eq. (17) by considering only one configuration, using the cell's lookup table to find the most likely configuration. The lookup table for the cell contains information about all the possible error configurations compatible with each possible syndrome, ignoring the value of the corner stabilizers. We can use this table to find the error configuration of minimum weight in $O(1)$. For the square cell (see Fig. 7), the table contains 2^{8+4} entries, corresponding to the possible syndromes obtained from the eight half-stabilizers in the boundary and the four stabilizers in the bulk of the cell. We can construct the lookup table by sorting the 2^{18} possible error configurations according to the corresponding syndrome and the weight of the error configuration. This computation only needs to be done once, prior to the decoding. With this approximation, we effectively reduce the computational overhead by a factor of 64, while keeping one of the terms of highest weight in Eq. (17). In our simulations, we found no significant change in the correction capabilities of the decoder for system sizes larger than 72 qubits. Therefore, we used this approximation for the simulations shown in Sec. IV.

Once we can compute the probability of all possible configurations compatible with a given syndrome, we can define the probability of a given half-splitting s_i^l given a certain fixed

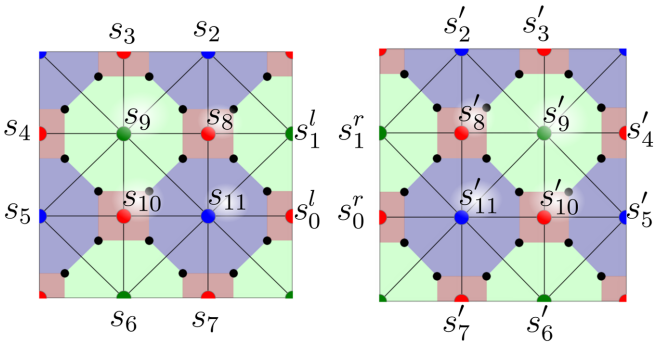

FIG. 9. Notation of stabilizer splitting. Example of a pair of splittings between two neighboring cells in the 4.8.8 lattice. The splittings 0 and 1 are being updated, and the super index indicates if we refer to the left or right half-stabilizer. The parity of the (half-) stabilizers on the left cell are represented as s_i, while the parity of the (half-) stabilizers on the right cell are notated with s_i'.

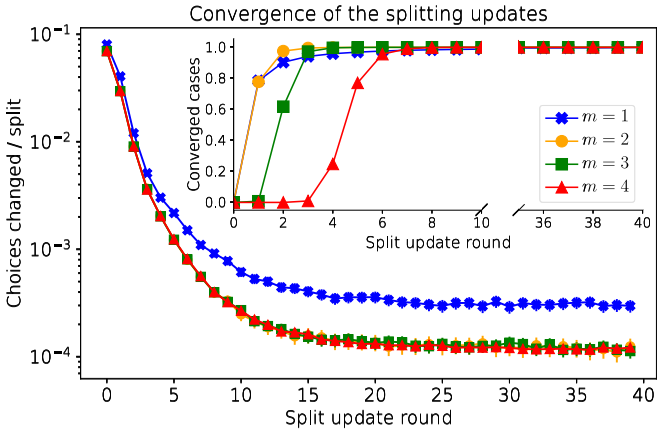

FIG. 10. Convergence of the splitting updates. We show the statistics of the performance of the splitting updates. In the main plot, we show the average fraction of splittings that have changed the split choice on each split update round. In the inset plot, we show the cumulative fraction of cases that have converged by each split update round. We define a case to have converged at round r if that was the last round with more than $3m$ changes in the splitting choice. For both plots, the lines connecting numerical data points have been drawn as a guide to the eye.

syndrome on the rest of splittings (where the superscript l corresponds to the cell on the left, see Fig. 9). This probability corresponds to the fraction of possible configurations compatible with the splitting choice, compared with the total probability of the possible configurations for all splitting choices,

$$p(s_i^l|\{s\}) = \frac{p(e|s_i, \{s\})}{p(e|s_i = 0, \{s\}) + p(e|s_i = 1, \{s\})}. \tag{18}$$

Seeing that every cell shares two stabilizers with each of its neighbors, the splitting choice for these two stabilizers is not independent. Therefore, we need to consider the joint probabilities, where we consider each of the combined splitting choices for the two splittings shared between two cells. Therefore, the probability of a splitting choice for the stabilizers s_0^l and s_1^l given a fixed choice on the rest of splittings (which we write as $\{s\}$ for simplicity) can be written as

$$p(s_0^l, s_1^l|\{s\}) = \frac{p(e|s_0^l, s_1^l, \{s\})}{\sum_{i,j=0}^{1} p(e|i, j, \{s\})}. \tag{19}$$

Note that this expression is equivalent for all splitting pairs, and we only wrote it explicitly for the first two splittings to simplify the notation. Since the rest of the splitting choices are not fixed, we can compute the estimate for the half-splitting within a cell by combining the information from all splitting choices. For this, we can use the information of the joint probabilities $p(s_k, s_{k+1})$, which corresponds to our current estimate of the probability of the half-splitting s_k, s_{k+1} [e.g., the probability of the splitting choice could be $p(s_2 = 1, s_3 = 1)$]. The estimate for the probability of a half-splitting configuration within the cell can then be computed as the sum of Eq. (19) for each splitting configuration of the other splittings involved in the cell:

$$p(s_0^l, s_1^l) = \sum_{\{s\}_k} p(s_0^l, s_1^l|\{s\}_k) \prod_{k=1}^{3} p(s_{2k}, s_{2k+1}). \tag{20}$$

Finally, we want to find the probability of a given splitting choice. This necessarily involves the configurations in both the left and right cells, and the value of the parity v_i of the stabilizers that we are measuring. Combining the information from both cells and ensuring the consistency condition $s_i^l \oplus s_i^r = v_i$, we can obtain the next estimate for the probability of a given splitting choice as

$$p_{\text{split}}(s_0^l, s_1^l) = \frac{p(s_0^l, s_1^l)p'(s_0^r = s_0^l \oplus v_0, s_1^r = s_1^l \oplus v_1)}{\sum_{i,j=0}^{1} p(i, j)p'(i \oplus v_0, j \oplus v_1)}, \tag{21}$$

where $\oplus$ represents a binary sum, and $\bar{v}_i = 1 \oplus v_i$. Using these equations, we can update our estimate for each of the splitting choices.

During the splitting algorithm, we apply global updates to the probabilities of the splitting choices by updating simultaneously the splitting probabilities of all splitting pairs in the code. All of the updates for the estimates of the splitting probabilities depend on the estimates from the previous step, which are not overwritten until all new estimates have been computed.

Although the number of global updates required for convergence can vary between runs, we empirically find that the average number of updates does not scale significantly with the lattice size, with less than 15 update rounds required. We test the convergence by measuring the average number of changes in the splitting choice per splitting and per round. We find that this ratio does not increase with the system size (Fig. 10). Furthermore, we estimate the fraction of cases that have converged after n-split update rounds. For a given error case, we define the number of rounds until convergence as the last update round with less than $3m$ changes in the split choice (with m being the number of rescaling steps required for that lattice size).

G. Rescaling of the cells

After a splitting configuration is chosen, a correction can be found within each of the cells in the lattice. The next step in the decoder is to map each square cell to a pair of qubits. The error probability of each qubit on the rescaled lattice also needs to undergo a rescaling process. In this section, we discuss the details on how to compute what is the resulting error probability for the rescaled qubits.

The main idea to understand in the rescaling of the qubit error probability is the fact that an error on a qubit in the rescaled lattice corresponds to the application of a logical operator on the qubit of the original lattice. If the original correction in the cell is C, we can write a first estimate of the error probability of the rescaled qubit given the splitting choice σ as

$$p(L|\sigma) = \frac{\sum_{\{S_b\}} p(C + L + \{S_b\})}{\sum_{\{S_b\}} p(C + \{S_b\}) + p(C + L + \{S_b\})}, \tag{22}$$

where the sum over $\{S_b\}$ represents all possible combinations of the bulk stabilizers, L is the logical operator, and $p(E)$ is the probability of a given error configuration E.

For this estimate, we assumed that the splitting choice from the previous step is correct. However, from the previous splitting step we know that we have an uncertainty in the splitting choice. In addition, we have an estimate of the probability of each splitting choice. Thus, we can include this information about the other splitting choices, weighted by the probability of each splitting choice, in the equation for the probability of an error in the rescaled qubit.

In order to include the information from alternative splitting configurations, we first need to understand what this probability of an alternative splitting means, and how to relate it to the correction C that we applied on the cell during the previous step. In particular, we need to find an expression of $p(L|\tilde{\sigma}_k)$ for the alternative splittings $\tilde{\sigma}_k$.

For the decoder of the 6.6.6 lattice [30], the key idea to find this expression is that, by applying the stabilizer of a given splitting, we can effectively change the choice of that splitting, as there are an odd number of qubits from that stabilizer on each cell. By applying the half-stabilizer corresponding to the splitting, we could relate the different corrections that correspond to each splitting choice, thus finding an expression for $p(L|\tilde{\sigma}_k)$.

In contrast, on the 4.8.8 lattice, the support of the half-stabilizers on each cell consists of an even number of qubits. This means that by applying the stabilizer, we do not change the value of the splitting choice for the splitting corresponding to that stabilizer. However, for the 4.8.8 color code, splittings come in pairs, as each cell shares two stabilizers. By applying the stabilizer of one of these two splittings, we can effectively change between the two choices of the neighbor splitting, and thus relate the corrections corresponding to both splitting choices. An example of this equivalence is shown in Fig. 11.

Following this rule, we can systematically find the probability $p(L|\sigma_k)$ for each of the 2^8 possible splitting configurations within a given cell by following these steps:

(1) Find the difference in splitting choice between the reference splitting σ_0 (the one with the maximum probability, chosen to find the correction) and the alternative splitting σ_k: $\Delta_k = (\sigma_0 - \sigma_k) \mod 2$.

(2) For each splitting in Δ_k, add the half-stabilizer of its neighbor splitting. We call this product of half-stabilizers δ_k.

(3) Compute the conditional probability $p(L|\sigma_k)$ by adding to each configuration in Eq. (22) the product of half-stabilizers δ_k.

(4) The final probability of error in the rescaled qubit can be computed as the sum of each of the conditional probabilities, weighted by our estimate of the probability of each splitting choice:

$$\begin{aligned} p(L) &= \sum_k p(L|\sigma_k)\ p(\sigma_k) \\ &= \sum_k \frac{p(\sigma_k) \sum_{\{S_b\}} p(C + L + \{S_b\} + \delta_k)}{\sum_{\{S_b\}} p(C + \{S_b\} + \delta_k) + p(C + L + \{S_b\} + \delta_k)}. \end{aligned} \tag{23}$$

Here, the sum over k selects the different combinations of splitting choices, and the sum over $\{S_b\}$ runs over all possible combinations of products of the bulk stabilizers.

Finally, there is one more factor that we need to take into account in the rescaling of the cells. Since each cell corresponds to two different qubits, the error probabilities of the two qubits in the cell are not independent. Thus, we can compute the probabilities of having an error (L_i) on each of the two logical qubits on the cell, leading to the joint probabilities for the four possible cases after the rescaling:

$$\tilde{C}_{\{S_b\},k} = C + \{S_b\} + \delta_k, \tag{24}$$

$$p(\mathbb{1}_0, \mathbb{1}_1) = \sum_k \frac{\sum_{\{S_b\}} p(\tilde{C}_{\{S_b\},k})}{D_k} p(\sigma_k), \tag{25}$$

$$p(L_0, \mathbb{1}_1) = \sum_k \frac{\sum_{\{S_b\}} p(L_0 + \tilde{C}_{\{S_b\},k})}{D_k} p(\sigma_k), \tag{26}$$

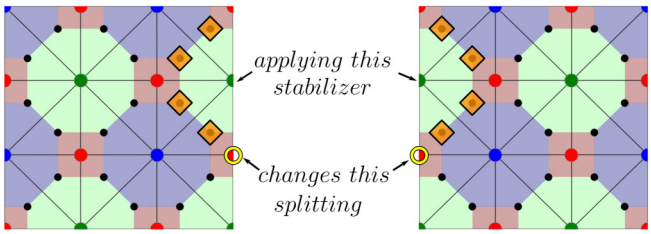

FIG. 11. Each cell shares two stabilizers with each of the neighboring cells, which need to be split. These two splittings form a pair of splittings. By applying the half-stabilizer of one of the stabilizers, the resulting correction corresponds to a change in the splitting choice of the other splitting in the pair. In the figure, we can see one example in which, by applying the half-stabilizer of one of the splittings, we change the parity of the neighbor splitting and obtain a valid correction for the new syndrome. The qubits affected by applying the stabilizers are marked with orange diamonds, and the stabilizer for which the splitting choice is changed is marked in yellow.

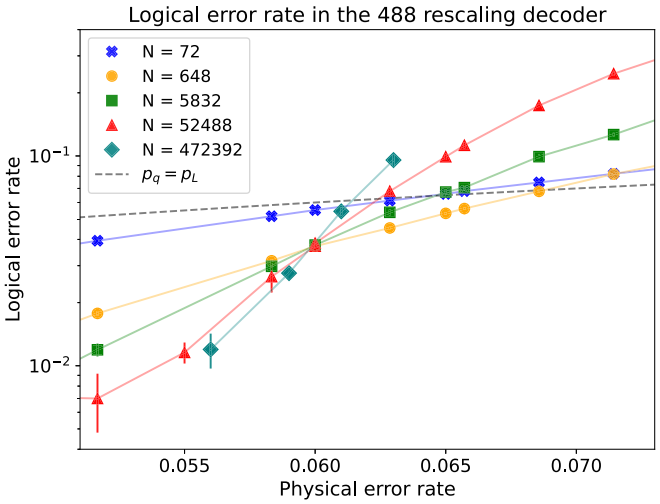

FIG. 12. Logical performance. We plot the average logical error rate vs the physical bit-flip (phase-flip) error rate for increasing size of the code lattice. Below and up to $p = 6.0\%$, the logical error rate decreases with increasing system size (see main text and Fig. 13).

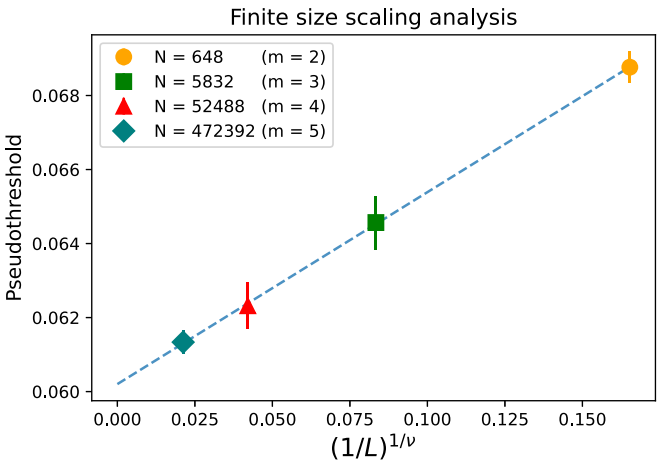

FIG. 13. Code capacity noise error threshold for the 4.8.8 color code decoder. By fitting the pseudothresholds to a finite-size scaling ansatz of Eq. (30), we estimate the threshold of the decoder as 6.0% for code capacity noise (independent phase and bit-flip errors with ideal syndrome measurement).

$$p(\mathbb{1}_0, L_1) = \sum_k \frac{\sum_{\{S_b\}} p(L_1 + \tilde{C}_{\{S_b\},k})}{D_k} p(\sigma_k), \tag{27}$$

$$p(L_0, L_1) = \sum_k \frac{\sum_{\{S_b\}} p(L_0 + L_1 + \tilde{C}_{\{S_b\},k})}{D_k} p(\sigma_k), \tag{28}$$

$$D_k = \sum_{\{S_b\}} \sum_{l_0,l_1=0}^{1} p(L_0^{l_0} + L_1^{l_1} + \tilde{C}_{\{S_b\},k}). \tag{29}$$

Using these equations, we can compute the joint error probabilities for the rescaled qubits in tuples. We can then use these probabilities directly to compute further probabilities, or obtain the error probabilities of the individual qubits by marginalizing the second qubit from the probability distribution.

IV. RESULTS

To estimate the threshold of the decoder for code capacity noise, we run Monte Carlo simulations, generating distributions of errors with different physical error rates and evaluating the logical error rate after decoding on each of the four logical qubits. We study bit-flip errors, as detailed in Sec. II. The behavior of the logical error rate as a function of the bit-flip error rate is presented in Fig. 12, showing the average error rate of the four logical qubits.

For each system size, the point at which the logical error rate equals the physical error rate is called the level-1 pseudothreshold [56]. To obtain the threshold of the decoder, we find the infinite-size limit by fitting the pseudothresholds to the following ansatz [57]:

$$t(L) = aL^{-\frac{1}{\nu}} + t_\infty, \tag{30}$$

where $t(L)$ is the pseudothreshold at system size of code distance L, and the unknown parameters are t_∞, the threshold in the infinite limit; ν, the scaling exponent; and the coefficient a. From a least-squares fit we obtain a threshold $t_\infty \simeq 6.0\%$ and a scaling exponent around $\nu \simeq 1.6$. The results are shown in Fig. 13.

V. CONCLUSIONS AND OUTLOOK

In our work we have presented an RG decoder for the 2D color code on the 4.8.8 lattice. The decoder can find a correction through local operations and message passing between the different regions of the code, after which the lattice can be rescaled to a smaller version of itself. We numerically estimate the code capacity threshold of our decoder for the 4.8.8 lattice to be 6.0%, assuming an error model of independent bit- and phase-flip errors with perfect syndrome extraction. This threshold lies below the 9.9%–10.3% obtained by other decoders, like [35,38]. This shortcoming in terms of threshold value has to be contrasted with the improvement in decoding complexity, which scales as $O(N \log N)$ with the number of qubits, which can be reduced to $O(\log N)$ by parallelization (see Sec. I). When compared to the rescaling algorithm for the color code on the hexagonal lattice (6.6.6 code), the threshold value obtained for that lattice geometry is slightly higher, at 7.8% [30]. Understanding whether this discrepancy is due to the lattice or due to the decoder requires further investigation of the influence of each step on the outcome of the decoder, as well as the interplay between the different steps, i.e., the choice of the size of the elementary cells, the communication between parts of the code, and the difference between the geometries of the code lattices.

As an outlook, arguably the most interesting followup to the presented work would be to adapt the decoding algorithm to the case of noisy syndrome readout (phenomenological noise) as has been done for related RG decoding schemes targeted to surface codes [58,59]. This makes it necessary to repeat stabilizer measurements in time and consequently

Algorithm 2. Decoder recursive function

```
function: Decoder (Syndrome, error rates, m)
Result: Recovery operation
if m == 0 then
    Apply brute force decoder
    return Correction
end
Apply 3 rounds of belief propagation
Apply corner updates
Compute initial estimate of the split probabilities
for Number of split update rounds do
    for each split pair do
        Update the probability of each split choice
    end
end
for each cell do
    Apply local decoder
    Rescale the cells to effective qubits
end
Compute rescaled error rates of the effective qubits
Apply Decoder (new syndrome, new error rates, m − 1)
Back-propagate the corrections from m − 1
return Correction
```

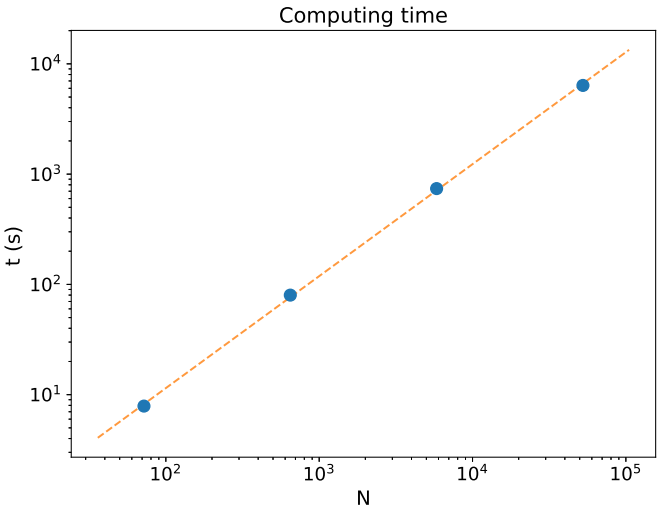

FIG. 14. Computing time. Time required by the decoder to decode lattices of different numbers of qubits N. The results show an almost linear scaling of the runtime when fitted to the ansatz $t = aN^b$ (shown in orange), with an exponent $b \simeq 1.0(1)$. Statistical error bars on the data points are smaller than the marker size.

the RG scheme has to be suitably adapted to three dimensions. The noise model could be refined further with the addition of, e.g., leakage noise or correlated noise [60]. This would provide insight into the experimentally relevant case of circuit-level noise to ultimately judge the practical benefit of the tradeoff between improved decoding speed and lower threshold value.

The code underlying the numerical simulations is available at [61].

ACKNOWLEDGMENTS

We acknowledge fruitful discussions with colleagues from the eQual and AQTION collaborations, in particular with C. Ryan-Anderson and M. Gutierrez. We thank P. Sarvepalli for useful information on technical aspects of the decoder developed in Ref. [30]. We gratefully acknowledge support by the EU Quantum Technology Flagship grant AQTION, Grant No. 820495. The research is based upon work supported by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA), via the U.S. Army Research Office Grant No. W911NF-16-1-0070. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the ODNI, IARPA, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the view of the U.S. Army Research Office. We acknowledge computational resources provided by Supercomputing Wales.

APPENDIX A: PSEUDOCODE

In this Appendix we provide a more detailed pseudocode of the decoder algorithm. The decoder function is written as a recursive function for a given lattice size m, each of which receives as an input the syndrome (values of the parity of the stabilizers) and the estimates of the error probabilities of each qubit. This function outputs the correction for a given lattice size.

APPENDIX B: SIMULATION DATA AND RUNTIME CONSIDERATIONS

The following table shows a selection of simulation results for the logical qubit error rate for a system size of N physical qubits and a physical error rate p, averaged over the four logical qubits. The complete results from the simulations are shown in Fig. 12.

N \ p	0.045	0.065
72	0.0293(2)	0.0659(3)
648	0.0088(3)	0.0533(7)
5832	0.0045(5)	0.067(2)
52488	0.003(1)	0.099(3)

We measured the computing time required by the simulations using a processor of type IntelCore i7-865U @1.9GHz. The results are shown in Fig. 14 and in the following table.

N	$t(s)$
72	7.(7)
648	7(9)
5832	7(4)0
52488	6(3)00

From the different steps of the decoder, there are two subroutines that require most of the computing time: The splitting of the stabilizers (see Sec. III F), which takes about 89.8% of the computing time, and the rescaling of the cells (see Sec. III G), which requires around 10% of the computing time, as shown in the sketch in Fig. 15.

Note that both of these tasks could be highly parallelized, as the update of the splitting probabilities can be done locally on the lattice, and simultaneously for each splitting pair, thereby reducing the time of this subroutine from $O(N)$ to $O(1)$. Similarly, the rescaling of the cells can be done locally and simultaneously for each cell, reducing the computational runtime of this part from $O(N \log N)$ to $O(\log N)$, and thereby of the overall decoder to $O(\log(N))$. We also note that the software underlying the presented results has not been written with runtime as the primary target and thus has potential for efficiency optimization. Together with the above-described parallelization, this bears the potential to reduce the computing time by several orders of magnitude, in particular for the largest lattice sizes. The above-mentioned parallelization of the subroutines of the decoder will also be the method of choice in future extensions of the decoder to cope with circuit noise, for potential use in decoding actual 2D error corrected quantum hardware.

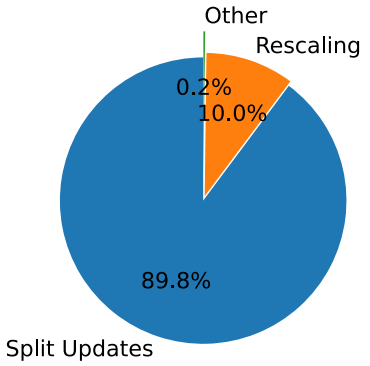

FIG. 15. Runtime budget of the decoder. The most resource intensive parts of the decoder are the updates of the splitting probabilities (see Sec. III F), followed by the rescaling of the cells to effective qubits (see Sec. III G). The remaining steps of the decoder require about 0.2% of the computing time.

[1] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information* (Cambridge University Press, Cambridge, 2000).
[2] B. M. Terhal, Quantum error correction for quantum memories, Rev. Mod. Phys. **87**, 307 (2015).
[3] D. Aharonov and M. Ben-Or, Fault-tolerant quantum computation with constant error rate, SIAM J. Comput. **38**, 1207 (2008).
[4] P. W. Shor, Fault-tolerant quantum computation, in *Proceedings of 37th Conference on Foundations of Computer Science* (IEEE, Piscataway, NJ, 1996), p. 56.
[5] J. Preskill, Reliable quantum computers, Proc. R. Soc. London A **454**, 385 (1998).
[6] A. Yu. Kitaev, Quantum error correction with imperfect gates, in *Quantum Communication, Computing, and Measurement* (Springer, Boston, MA, 1997), p. 181.
[7] A. Kitaev, Fault-tolerant quantum computation by anyons, Ann. Phys. **303**, 2 (2003).
[8] E. Dennis, A. Kitaev, A. Landahl, and J. Preskill, Topological quantum memory, J. Math. Phys. **43**, 4452 (2002).
[9] H. Bombin and M. A. Martin-Delgado, Topological Quantum Distillation, Phys. Rev. Lett. **97**, 180501 (2006).
[10] H. Bombin and M. A. Martin-Delgado, Topological Computation without Braiding, Phys. Rev. Lett. **98**, 160502 (2007).
[11] R. Raussendorf and J. Harrington, Fault-Tolerant Quantum Computation with High Threshold in Two Dimensions, Phys. Rev. Lett. **98**, 190504 (2007).
[12] D. Nigg, M. Müller, E. A. Martinez, P. Schindler, M. Hennrich, T. Monz, M. A. Martin-Delgado, and R. Blatt, Quantum computations on a topologically encoded qubit, Science **345**, 302 (2014).
[13] C. Ryan-Anderson, J. G. Bohnet, K. Lee, D. Gresh, A. Hankin, J. P. Gaebler, D. Francois, A. Chernoguzov, D. Lucchetti, N. C. Brown, T. M. Gatterman, S. K. Halit, K. Gilmore, J. A. Gerber, B. Neyenhuis, D. Hayes, and R. P. Stutz, Realization of Real-Time Fault-Tolerant Quantum Error Correction, Phys. Rev. X **11**, 041058 (2021).
[14] K. J. Satzinger, Y.-J. Liu, A. Smith, C. Knapp, M. Newman, C. Jones, Z. Chen, C. Quintana, X. Mi, A. Dunsworth *et al.*, Realizing topologically ordered states on a quantum processor, Science **374**, 1237 (2021).
[15] C. K. Andersen, A. Remm, S. Lazar, S. Krinner, N. Lacroix, G. J. Norris, M. Gabureac, C. Eichler, and A. Wallraff, Repeated quantum error detection in a surface code, Nat. Phys. **16**, 875 (2020).
[16] L. Postler, S. Heuen, I. Pogorelov, M. Rispler, T. Feldker, M. Meth, C. D. Marciniak, R. Stricker, M. Ringbauer, R. Blatt, P. Schindler, M. Müller, and T. Monz, Demonstration of fault-tolerant universal quantum gate operations, Nature (London) **605**, 675 (2022).
[17] R. Versluis, S. Poletto, N. Khammassi, B. Tarasinski, N. Haider, D. J. Michalak, A. Bruno, K. Bertels, and L. DiCarlo, Scalable quantum circuit and control for a superconducting surface code, Phys. Rev. Appl. **8**, 034021 (2017).
[18] J. F. Marques, B. M. Varbanov, M. S. Moreira, H. Ali, N. Muthusubramanian, C. Zachariadis, F. Battistel, M. Beekman, N. Haider, W. Vlothuizen, A. Bruno, B. M. Terhal, and L. DiCarlo, Logical-qubit operations in an error-detecting surface code, Nat. Phys. **18**, 80 (2022).
[19] M. Takita, A. D. Córcoles, E. Magesan, B. Abdo, M. Brink, A. Cross, J. M. Chow, and J. M. Gambetta, Demonstration of

Weight-Four Parity Measurements in the Surface Code Architecture, Phys. Rev. Lett. **117**, 210505 (2016).
[20] J. Hilder, D. Pijn, O. Onishchenko, A. Stahl, M. Orth, B. Lekitsch, A. Rodriguez-Blanco, M. Müller, F. Schmidt-Kaler, and U. G. Poschinger, Fault-Tolerant Parity Readout on a Shuttling-Based Trapped-Ion Quantum Computer, Phys. Rev. X **12**, 011032 (2022).
[21] S. Krinner, N. Lacroix, A. Remm, A. Di Paolo, E. Genois, C. Leroux, C. Hellings, S. Lazar, F. Swiadek, J. Herrmann, G. J. Norris, C. K. Andersen, M. Müller, A. Blais, C. Eichler, and A. Wallraff, Realizing repeated quantum error correction in a distance-three surface code, Nature (London) **605**, 669 (2022).
[22] B. Eastin and E. Knill, Restrictions on Transversal Encoded Quantum Gate Sets, Phys. Rev. Lett. **102**, 110502 (2009).
[23] S. Bravyi and A. Kitaev, Universal quantum computation with ideal clifford gates and noisy ancillas, Phys. Rev. A **71**, 022316 (2005).
[24] C. Chamberland and K. Noh, Very low overhead fault-tolerant magic state preparation using redundant ancilla encoding and flag qubits, npj Quantum Inf. **6**, 91 (2020).
[25] A. Krishna and J.-P. Tillich, Towards Low Overhead Magic State Distillation, Phys. Rev. Lett. **123**, 070507 (2019).
[26] D. Litinski, Magic state distillation: Not as costly as you think, Quantum **3**, 205 (2019).
[27] A. J. Landahl, J. T. Anderson, and P. R. Rice, Fault-tolerant quantum computing with color codes, arXiv:1108.5738.
[28] H. G. Katzgraber, H. Bombin, and M. A. Martin-Delgado, Error Threshold for Color Codes and Random Three-Body Ising Models, Phys. Rev. Lett. **103**, 090501 (2009).
[29] R. S. Andrist, H. G. Katzgraber, H. Bombin, and M. A. Martin-Delgado, Tricolored lattice gauge theory with randomness: Fault tolerance in topological color codes, New J. Phys. **13**, 083006 (2011).
[30] P. Sarvepalli and R. Raussendorf, Efficient decoding of topological color codes, Phys. Rev. A **85**, 022317 (2012).
[31] D. S. Wang, A. G. Fowler, C. D. Hill, and L. C. L. Hollenberg, Graphical algorithms and threshold error rates for the 2d color code, Quantum Inf. Comput. **10**, 780 (2010).
[32] A. M. Stephens, Efficient fault-tolerant decoding of topological color codes, arXiv:1402.3037.
[33] N. Maskara, A. Kubica, and T. Jochym-O'Connor, Advantages of versatile neural-network decoding for topological codes, Phys. Rev. A **99**, 052351 (2019).
[34] N. Delfosse, Decoding color codes by projection onto surface codes, Phys. Rev. A **89**, 012317 (2014).
[35] N. Delfosse and N. H. Nickerson, Almost-linear time decoding algorithm for topological codes, Quantum **5**, 595 (2021).
[36] N. Delfosse and G. Zémor, Linear-time maximum likelihood decoding of surface codes over the quantum erasure channel, Phys. Rev. Res. **2**, 033042 (2020).
[37] A. Kubica and J. Preskill, Cellular-Automaton Decoders with Provable Thresholds for Topological Codes, Phys. Rev. Lett. **123**, 020501 (2019).
[38] A. Kubica and N. Delfosse, Efficient color code decoders in $d \geqslant 2$ dimensions from toric code decoders, arXiv:1905.07393.
[39] P. Baireuther, M. D. Caio, B. Criger, C. W. J. Beenakker, and T. E. O'Brien, Neural network decoder for topological color codes with circuit level noise, New J. Phys. **21**, 013003 (2019).
[40] C. Chamberland and P. Ronagh, Deep neural decoders for near term fault-tolerant experiments, Quantum Sci. Technol. **3**, 044002 (2018).
[41] A. Davaasuren, Y. Suzuki, K. Fujii, and M. Koashi, General framework for constructing fast and near-optimal machine-learning-based decoder of the topological stabilizer codes, Phys. Rev. Res. **2**, 033399 (2020).
[42] J. Edmonds, Paths, trees, and flowers, Can. J. Math. **17**, 449 (1965).
[43] V. Kolmogorov, Blossom v: A new implementation of a minimum cost perfect matching algorithm, Math. Program. Computat. **1**, 43 (2009).
[44] H. N. Gabow, The weighted matching approach to maximum cardinality matching, Fund. Inf. **154**, 109 (2017).
[45] R. Duan, S. Pettie, and H.-H. Su, Scaling algorithms for weighted matching in general graphs, ACM Trans. Algorithms **14**, 1 (2018).
[46] G. Duclos-Cianci and D. Poulin, Fast Decoders for Topological Quantum Codes, Phys. Rev. Lett. **104**, 050504 (2010).
[47] H. Bombin, An introduction to topological quantum codes, *Quantum Error Correction* (Cambridge University Press, Cambridge, 2013).
[48] A. R. Calderbank and P. W. Shor, Good quantum error-correcting codes exist, Phys. Rev. A **54**, 1098 (1996).
[49] A. M. Steane, Error Correcting Codes in Quantum Theory, Phys. Rev. Lett. **77**, 793 (1996).
[50] D. J. C. MacKay, *Information Theory, Inference, and Learning Algorithms* (Cambridge University Press, Cambridge, 2003).
[51] J. S. Yedidia, W. T. Freeman, and Y. Weiss, Understanding belief propagation and its generalizations, in *Exploring Artificial Intelligence in the New Millennium* (Morgan Kaufmann, Burlington, MA, 2003), p. 239.
[52] J. Pearl, Reverend Bayes on inference engines: A distributed hierarchical approach, in *AAAI'82: Proceedings of the Second AAAI Conference on Artificial Intelligence* (AAAI Press, Palo Alto, CA, 1982), p. 133.
[53] D. J. C. MacKay and R. M. Neal, Near Shannon limit performance of low density parity check codes, Electron. Lett. **32**, 1645 (1996).
[54] D. Poulin and Y. Chung, On the iterative decoding of sparse quantum codes, Quantum Inf. Comput. **8**, 987 (2008).
[55] J.-H. Kim, M.-Y. Nam, and H.-Y. Song, Variable-to-check residual belief propagation for LDPC codes, Electron. Lett. **45**, 117 (2009).
[56] K. Svore, A. Cross, I. Chuang, and A. Aho, A flow-map model for analyzing pseudothresholds in fault-tolerant quantum computing, Quantum Inf. Comput. **6**, 193 (2006).
[57] C. Wang, J. Harrington, and J. Preskill, Confinement-Higgs transition in a disordered gauge theory and the accuracy threshold for quantum memory, Ann. Phys. **303**, 31 (2003).
[58] G. Duclos-Cianci and D. Poulin, Fault-tolerant renormalization group decoder for abelian topological codes, Quantum Inf. Comput. **14**, 721 (2014).
[59] K. Duivenvoorden, N. P. Breuckmann, and B. M. Terhal, Renormalization group decoder for a four-dimensional toric code, IEEE Trans. Inf. Theory **65**, 2545 (2018).
[60] C. T. Chubb and S. T. Flammia, Statistical mechanical models for quantum codes with correlated noise, Ann. Inst. Henri Poincare D **8**, 269 (2021).
[61] P. Parrado-Rodríguez, Rescaling decoder 488, https://gitlab.com/pedroparrado/rescalingdecoder488.