

Tile Codes: High-Efficiency Quantum Codes on a Lattice with Boundary

Vincent Steffan^{1,*}, Shin Ho Choe^{1,*}, Nikolas P. Breuckmann²,

Francisco Revson Fernandes Pereira¹, and Jens Niklas Eberhardt³

¹*IQM Quantum Computers, Georg-Brauchle-Ring 23-25, 80992 Munich, Germany*

²*Breuckmann Ltd., Redcross Village, BS2 0BB Bristol, United Kingdom*

³*Institute of Mathematics, Johannes Gutenberg-Universität Mainz, Mainz, Germany*



(Received 6 May 2025; revised 4 August 2025; accepted 20 August 2025; published 21 October 2025)

We introduce *tile codes*, a simple yet powerful way of constructing quantum codes that are local on a planar 2D lattice. Tile codes generalize the usual surface code by allowing for a bit more flexibility in terms of locality and stabilizer weight. Our construction does not compromise on the fact that the codes are local on a lattice with open boundary conditions. Despite its simplicity, we use our construction to find codes with parameters $[[288, 8, 12]]$ using weight-6 stabilizers and $[[288, 8, 14]]$ using weight-8 stabilizers, outperforming all previously known constructions in this direction. Allowing for a slightly higher nonlocality, we find a $[[512, 18, 19]]$ code using weight-8 stabilizers, which outperforms the rotated surface code by a factor of more than 12. Our approach provides a unified framework for understanding the structure of codes that are local on a 2D planar lattice and offers a systematic way to explore the space of possible code parameters. In particular, due to its simplicity, the construction naturally accommodates various types of boundary conditions and stabilizer configurations, making it a versatile tool for quantum error-correction code design.

DOI: [10.1103/l4mx-l3xx](https://doi.org/10.1103/l4mx-l3xx)

The surface code was introduced in 1998 [1,2], and until today, it is arguably the quantum error-correcting code best suited for current and near-term quantum devices [3–6]. It owes its practicability mainly to the fact that its stabilizer checks are local on a regular, planar 2D lattice. In fact, having nearest-neighbor connectivity between the physical qubits on a chip is sufficient to realize a surface code without further overhead. Fundamentally, this locality comes at a price: The surface code encodes just one logical qubit.

Compared to the surface code, so-called quantum low-density parity-check (qLDPC) codes may offer higher encoding rates and better protection of encoded qubits [7]. However, realizing the promises of qLDPC codes is far from trivial: qLDPC codes with the best theoretical performance are inherently non-2D [8].

Recent advances in quantum hardware design, particularly for superconducting circuits, suggest that slightly relaxed locality constraints may be achievable in near-term devices [9–14]. This raises hopes, but it is *a priori* not clear how to leverage this additional flexibility to develop more efficient codes.

This Letter addresses the main question of constructing more efficient quantum error correction (QEC) codes by slightly relaxing the locality of the surface code. We introduce *tile codes*, a natural generalization of the surface code that relaxes locality in two key ways. Instead of assuming nearest-neighbor connectivity, we allow for the stabilizer generators to be supported in boxes of a certain size. Second, we allow for stabilizer weights higher than 4, as in the case of the surface code. Tile codes follow a systematic approach where stabilizer tiles (specific stabilizer patterns) are placed on a layout according to simple rules that guarantee that the stabilizers of the code commute.

While our general construction is extremely simple, we discover codes that outperform the surface code by factors of up to 12 while still maintaining practical locality constraints. For example, with just a 3×3 box size and weight-6 stabilizers, we obtain a $[[288, 8, 12]]$ code—a 4-times improvement in efficiency compared to the rotated surface code. Increasing to weight-8 stabilizers yields a $[[288, 8, 14]]$ code, while a 4×4 box with weight-8 stabilizers produces $[[288, 18, 13]]$ and $[[512, 18, 19]]$ codes—a remarkable 10-times and 12-times gain in efficiency compared to the rotated surface code. We remark that the $[[288, 8, 12]]$ code was first discovered by Liang *et al.* [15] and served as an inspiration for our constructions.

Tile codes are related to bivariate bicycle (BB) codes [16–22]. BB codes generalize the toric code, allowing for some modest (constant) amount of nonlocality assuming periodic boundaries on the lattice. This means that the

*These authors contributed equally to this work.

†Contact author: vincent.steffan@meetiqm.com

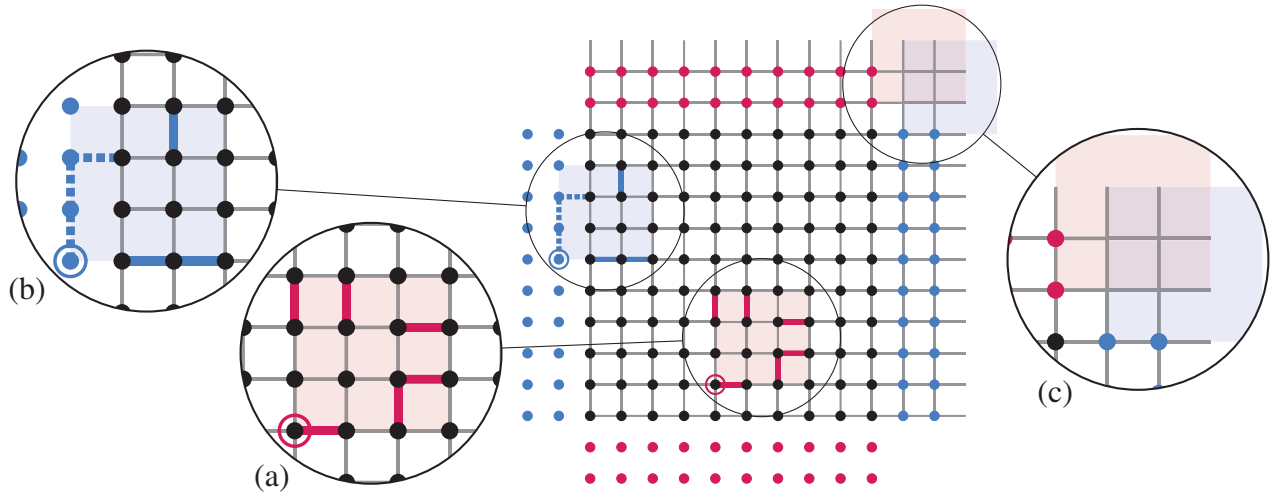
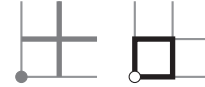


FIG. 1. An example of a $[[288, 8, 12]]$ tile code following our construction. The qubits sit on the edges of the lattice. The X- and Z-type stabilizer tiles sit at the vertices of the lattice. A black dot indicates that both an X- and a Z-type stabilizer tile is there. Red dots and blue dots indicate X-type and Z-type stabilizers, respectively. The dots are the anchors of the 3×3 boxes that contain the stabilizer tiles. In (a), we enlarge on an example of an X-type bulk stabilizer tile, and in (b) an example of a Z-type boundary stabilizer tile. Since the dotted edges that usually would be in the support of the Z-type stabilizer are not in the set of physical qubits, the support of this stabilizer is restricted to the nondotted edges for this tile code. In (c), we visualize that naturally, if two boundary stabilizers overlap, by the choice of the layout, their overlap is contained in the lattice of physical qubits. We intentionally only draw the $B \times B$ box containing the stabilizers and do not mark the edges in the support of the stabilizers, since the argument is agnostic of the specific stabilizer.

stabilizer checks are not truly local in a planar layout. There is still a growing number of stabilizer checks that require a growing amount of nonlocality. This issue is addressed if one can build reliable long-range couplers [17] or by simulating nonlocal interactions by teleportation-based routing [23] or swapping qubits [24]. Tile codes, on the other hand, do not require periodicity and thus have true $O(1)$ locality on a planar 2D lattice. With that, tile codes address this major challenge in quantum error correction, achieving high efficiency while preserving implementability on near-term hardware.

Tile codes—We now introduce a family of quantum stabilizer codes which we call *tile codes*. For a visual overview, we refer to Fig. 1. Tile codes are a natural generalization of surface codes: similar to surface codes, they have a bulk region with translation-invariant stabilizer checks and boundaries where stabilizer checks can have restricted supports. They arise by choosing a *local structure*, which consists of shapes of an X- and Z-type stabilizer called tiles, and a *global structure*, which is a layout of qubits and positions of stabilizer checks.

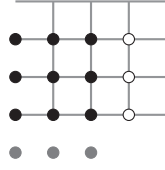
A *tile* is a subset of edges of a $B \times B$ square grid, excluding the edges on the top row and the rightmost column. We identify edges with physical qubits so that a tile defines the support of a Pauli-X or Pauli-Z stabilizer check. For example, one can recover the vertex and plaquette operators of the surface code, which are supported on the edges adjacent to a vertex and a face, by using the following X and Z tiles:



We will draw tiles with a marked dot, which indicates their placement in the layout that we will explain next.

The *layout* of a tile code determines the full set of physical qubits, as well as the position of tiles (and hence, stabilizer checks). More specifically, a layout is a finite subset of the square grid. The subset of edges is identified with the physical qubits. The vertices are marked depending on the placement of tiles which represent the stabilizer checks. We will mark vertices in black, grey, or white. We place both X and Z tiles on black vertices, whereas on each grey vertex of the layout, we place only an X tile, and on each white vertex only a Z tile. Note that not all vertices have to be marked. Unmarked vertices do not have any stabilizer checks associated with them. If the set of edges of the tile code does not contain the whole support of a stabilizer, then the support of the stabilizer should be thought of as truncated to the available qubits. We will refer to stabilizers whose support is contained in the set of edges of the layout as *bulk stabilizers*, and to the remaining stabilizers as *boundary stabilizers*.

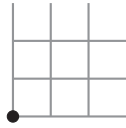
This concept of a layout is hence a generalization of the various shapes and sizes of surface codes, both rotated and unrotated, and the notion of smooth and rough boundaries. For example, using the surface code tiles, the following layout specifies a usual, unrotated surface code with distance 4:



While this is an unorthodox way of presenting the surface code, our convention will be useful for the general construction of tile codes, which we describe in the following.

Systematic construction of tile codes—Not every choice of tiles and layout yields a well-defined CSS code. Indeed, one of the key challenges in constructing tile codes is to choose these such that the boundary stabilizers commute. We will now explain a systematic construction that ensures commutativity.

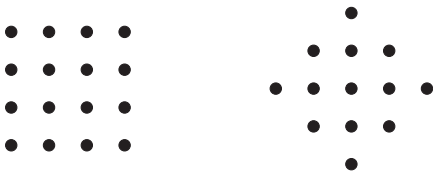
We first determine the local structure of our code by choosing an X tile and a Z tile such that (T1) Both tiles are confined within a box of size $B \times B$ with no support at the top horizontal and right vertical edges of the box:



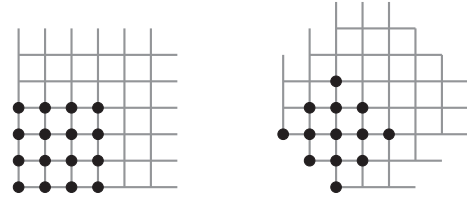
(T2) The X tile and Z tile determine each other in the following way: for each X tile with support on a horizontal (or vertical) edge with coordinate (x, y) , there is a vertical (or horizontal, respectively) edge with coordinates $(B - 1 - x, B - 1 - y)$ in the Z tile [25].

The condition (T2) guarantees that X and Z tiles, in any relative position, have an even overlap [26,27].

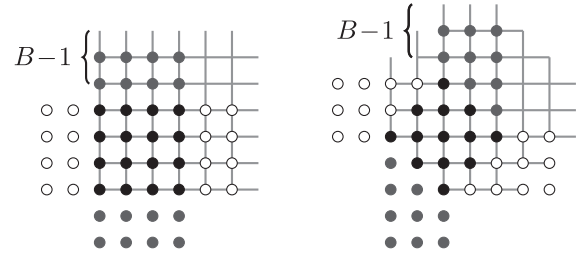
Next, we start constructing a layout by defining a designated set of bulk-stabilizer positions. While this set can be arbitrary, we will always restrict ourselves to following (rotated) rectangular shapes:



Since in the end, we want these stabilizers to be bulk stabilizers, and therefore their support completely contained in the layout, we will choose as the set of edges the union of all $B \times B$ boxes over all bulk stabilizer positions. For example, the following layouts are induced by the lattices depicted above together with 3×3 boxes containing the bulk stabilizers:



Next, we want to add boundary stabilizers. Here, we require that for any pair of an X- and a Z-boundary tile, they either do not intersect, or the intersection of their supports is entirely in the set of edges of the layout. Inspired by the rough and smooth boundaries of the surface code, we can, for example, choose the following sets of boundary stabilizers:



For the surface code, the boundary decorated by X stabilizers corresponds to a smooth boundary, and the boundary decorated by Z stabilizers corresponds to a rough boundary. It is clear that the choices of the sets of boundary stabilizers shown above generalize to any sizes of the initial bulk stabilizer lattice, as well as the boxes that contain the stabilizers. Note also that all boundary stabilizers commute. This is because any pair of boxes at a grey and a white dot does either not overlap or has their intersection of edges completely contained in the set of physical qubits; see Fig. 1(c) for a visualization. Hence, either their overlap is empty, or there is complete overlap of the two boxes at these relative conditions, which by (T2) is even.

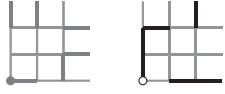
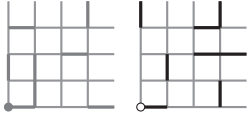
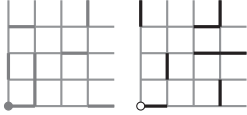
We emphasize that so far, the construction is agnostic of the specific support of the stabilizer tiles. In fact, we only use the fact that the support is contained in boxes of size $B \times B$; see (T1).

Finally, we fine-tune the layout to the specific support of the stabilizers. In more detail, we will first remove all qubits that are not supported in any X-type stabilizer or are not supported in any Z-type stabilizer. Finally, we remove all stabilizers, whose support has become empty because of this. Note that this step largely depends on the supports of the stabilizers in the tiles.

Examples of tile codes—Despite the simplicity of this construction, we will now see that this approach can yield planar versions of BB codes that drastically outperform all known constructions in this direction [18,29].

We systematically searched for a variety of box sizes and stabilizer weights. We first estimated the distance of the codes using a probabilistic algorithm [30,31] and then

TABLE I. Parameters of some tile codes. We identified these examples by fixing the layout to the (unrotated) square layout and values for box size B , check weight w , and number of physical qubits n . We then performed a brute force search among all possible tile configurations and selected those that produced codes with the highest efficiency ratio kd^2/n . The distances presented are exact values, calculated using integer linear programming.

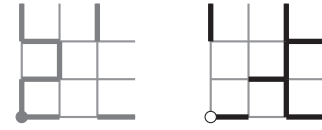
B	w	$[[n, k, d]]$	kd^2/n	X and Z tiles
3	6	$[[288, 8, 12]]$	4	
3	8	$[[288, 8, 14]]$	5.4	
4	8	$[[288, 18, 13]]$	10.6	
4	8	$[[512, 18, 19]]$	12.7	

confirmed the results using an integer linear program solver. The results of this search can be found in Table I. We will now highlight a few concrete codes with attractive parameters.

We start by searching for codes using the unrotated layout using 10×10 copies of X- and Z-bulk tiles, where we exhaustively check all stabilizers confined in a 3×3 box. Assuming that we will not remove qubits and stabilizers in the last step, the construction will always yield a code with 288 physical and 8 logical qubits. In general, for box size B , one gets $2(B-1)^2$ logical qubits under the assumption that the construction finishes without the removal of qubits and stabilizer checks. We find that the best possible distance is 12, giving a $[[288, 8, 12]]$ code. Note that this slight relaxation of locality therefore gives a 4-times improvement in kd^2/n over the rotated surface code. We depict an example of stabilizer tiles giving these parameters here and refer to Appendix B for more examples:

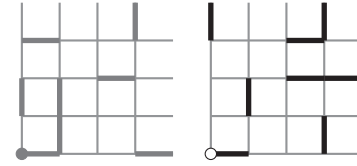


If we allow for weight-8 stabilizers instead of weight-6 in the same construction stabilizers, we can even find codes with parameters $[[288, 8, 14]]$. The following is an example of stabilizer tiles realizing these parameters:

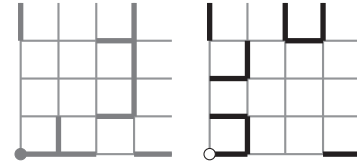


Refer to Appendix C for more examples.

Finally, if we allow for weight-8 stabilizer tiles that are confined in 4×4 boxes, we can obtain codes with parameters $[[288, 18, 13]]$ and $[[512, 18, 19]]$ using the unrotated square layouts with 9×9 and 13×13 copies of the bulk tiles, respectively. They give a 10- to 12-times improvement over the rotated surface code. We depict examples of stabilizer tiles here:



For higher stabilizer weights, we did not perform an exhaustive search. By a randomized search for weight-10 and 4×4 boxes, we found a $[[512, 18, \leq 23]]$ code with the following stabilizer tiles. We strongly expect that $d = 23$, and hence $kd^2/n \approx 18.6$, though we did not formally verify this using integer linear programming:



We also mention that all results from Ref. [18] on the pruning of hypergraph product codes of cyclic codes can be reconstructed from our construction using the unrotated bulk stabilizer lattice. For a proof, we refer to Appendix A. In addition, we remark that the rotated surface code can be recovered from our construction using a rotated layout. We also found some codes with weight-6 or weight-8 stabilizers using the rotated layout, but their parameters were not as good as those using the unrotated layout.

Discussion—In this Letter, we give a simple but powerful construction for 2D-local codes on a planar lattice. With this, we address an important problem in making qLDPC codes practical for near- and mid-term superconducting hardware. By relaxing the locality condition of the surface code slightly, we observe up to a 12-times improvement over the rotated surface code, and we find codes that outperform current state-of-the-art proposals for the implementation of qLDPC codes.

Our construction is simple and general, and we believe that there are many ways forward to construct further tile codes—e.g., using other bulk stabilizer tile layouts like rotations and layouts with holes or polygonal boundaries.

Another future line of research is to investigate protocols and design ancillary systems for performing logical computation on the codes we found. We expect that one can adapt the QEC cycle for BB codes developed in Ref. [17] to tile codes. It would be very interesting to see how the boundary impacts the numerical simulations and threshold. Furthermore, one could explore the construction of non-CSS versions of tile codes by applying Clifford deformations similar to the XZZX surface code.

Acknowledgments—We thank Yu-An Chen and Zijian Liang for fruitful discussions. J.N.E. was supported by Deutsche Forschungsgemeinschaft (DFG) Project No. 45744154, Equivariant K-motives and Koszul duality.

Data availability—The data are not publicly available. The data are available from the authors upon reasonable request.

-
- [1] S. B. Bravyi and A. Y. Kitaev, [arXiv:quant-ph/9811052](#).
 - [2] M. H. Freedman and D. A. Meyer, *Found. Comput. Math.* **1**, 325 (2001).
 - [3] C. K. Andersen, A. Remm, S. Lazar, S. Krinner, N. Lacroix, G. J. Norris, M. Gabureac, C. Eichler, and A. Wallraff, *Nat. Phys.* **16**, 875 (2020).
 - [4] 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, *Nature (London)* **605**, 669 (2022).
 - [5] Google Quantum AI and Collaborators, *Nature (London)* **614**, 676 (2023).
 - [6] Google Quantum AI and Collaborators, *Nature (London)* **638**, 920 (2024).
 - [7] N. P. Breuckmann and J. N. Eberhardt, *PRX Quantum* **2**, 040101 (2021).
 - [8] S. Bravyi, D. Poulin, and B. Terhal, *Phys. Rev. Lett.* **104**, 050503 (2010).
 - [9] F. Vigneau, S. Majumder, A. Rath, P. Parrado-Rodríguez, F. R. F. Pereira, S. Pogorzalek, T. Jones, N. Wurz, M. Renger, J. Verjauw, P. Yang, H.-S. Ku, W. Kindel, F. Deppe, and J. Heinsoo, [arXiv:2503.12869](#).
 - [10] M. Renger *et al.*, [arXiv:2503.10903](#).
 - [11] A. Wallraff, D. I. Schuster, A. Blais, L. Frunzio, R.-S. Huang, J. Majer, S. Kumar, S. M. Girvin, and R. J. Schoelkopf, *Nature (London)* **431**, 162 (2004).
 - [12] A. Blais, R.-S. Huang, A. Wallraff, S. M. Girvin, and R. J. Schoelkopf, *Phys. Rev. A* **69**, 062320 (2004).
 - [13] C. Song, K. Xu, W. Liu, C.-p. Yang, S.-B. Zheng, H. Deng, Q. Xie, K. Huang, Q. Guo, L. Zhang *et al.*, *Phys. Rev. Lett.* **119**, 180511 (2017).
 - [14] C. Song, K. Xu, H. Li, Y.-R. Zhang, X. Zhang, W. Liu, Q. Guo, Z. Wang, W. Ren, J. Hao *et al.*, *Science* **365**, 574 (2019).
 - [15] Z. Liang, J. N. Eberhardt, and Y.-A. Chen, [arXiv:2504.08887](#).
 - [16] H.-K. Lin and L. P. Pryadko, *Phys. Rev. A* **109**, 022407 (2024).
 - [17] S. Bravyi, A. W. Cross, J. M. Gambetta, D. Maslov, P. Rall, and T. J. Yoder, *Nature (London)* **627**, 778 (2024).
 - [18] J. N. Eberhardt, F. R. F. Pereira, and V. Steffan, [arXiv:2412.04181](#).
 - [19] J. N. Eberhardt and V. Steffan, *IEEE Transactions on Information Theory* (IEEE, 2025), [10.1109/TIT.2024.3521638](#).
 - [20] K. Chen, Y. Liu, Y. Zhang, Z. Liang, Y.-A. Chen, K. Liu, and H. Song, *Phys. Rev. Lett.* **135**, 076603 (2025).
 - [21] Z. Liang, K. Liu, H. Song, and Y.-A. Chen, [arXiv:2503.03827](#).
 - [22] Z. Liang, B. Yang, J. T. Iosue, and Y.-A. Chen, [arXiv:2410.11942](#).
 - [23] N. Berthussen, D. Devulapalli, E. Schoute, A. M. Childs, M. J. Gullans, A. V. Gorshkov, and D. Gottesman, *PRX Quantum* **6**, 010306 (2025).
 - [24] C. A. Pattison, A. Krishna, and J. Preskill, *Quantum* **9**, 1728 (2025).
 - [25] Note that our convention is to label both the horizontal and vertical edges touching the origin of the box in (T1) with (0,0).
 - [26] Consider an X tile with support (x, y) shifted by (a, b) overlapping with a Z tile at $(x + a, y + b)$. By condition (T2), the unshifted X tile must have support $[(B - 1) - x - a, (B - 1) - y - b]$. Thus, the shifted X tile has support $[(B - 1) - x, (B - 1) - y]$, which by (T2) is also in the Z tile's support. This symmetry ensures that all overlaps are even.
 - [27] On the infinite lattice $\mathbb{Z} \times \mathbb{Z}$, tiles with condition (T2) give rise to a Pauli Hamiltonian with commuting terms. It turns out [20,28] that the groundspace of this Hamiltonian exhibits topological order only if (T2) holds. Practically, this means that (T2) is necessary if we want these tile codes to have distance scaling with the dimensions of the layout.
 - [28] J. Haah, *Revista colombiana de matematicas* **50**, 299 (2017).
 - [29] L. Pecorari, S. Jandura, G. K. Brennen, and G. Pupillo, *Nat. Commun.* **16**, 1111 (2025).
 - [30] GAP, GAP—Groups, Algorithms, and Programming, Version 4.13.1, The GAP Group (2024), <https://www.gap-system.org/cite/>.
 - [31] L. P. Pryadko, V. A. Shabashov, and V. K. Kozin, *J. Open Source Software* **7**, 4120 (2022).

End Matter

Appendix A: The hypergraph product case—The tile codes we constructed can be viewed as planar versions of BB codes. The first attempts to find planar versions of BB codes—and with them, tile codes—have been achieved in Refs. [18,29]. We refer for details on BB codes to Refs. [16,18]. We mention that the tile codes

constructed in Refs. [18,29] are exactly the planar versions of BB codes that are hypergraph product codes. We will now explain how all of the tile codes constructed in Refs. [18,29] naturally arise by our simple construction. We stress that $O(1)$ -local hypergraph product codes are somewhat trivial, and the

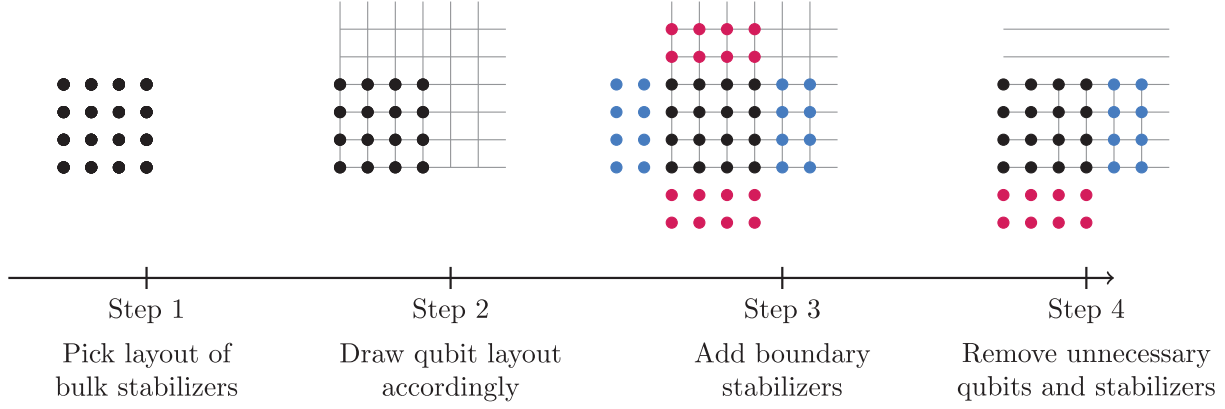
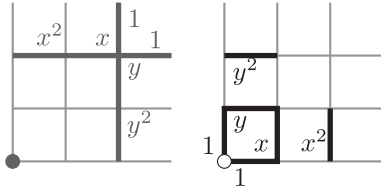


FIG. 2. Visualization of the construction in the hypergraph product setup. Following the simple steps of choosing a rectangular layout of bulk stabilizers and following the construction in the main text recovers all constructions from Refs. [18,29].

nonhypergraph product codes we construct in the main text drastically outperform hypergraph product codes even without assuming locality for similar sizes.

The stabilizer tiles are specified by a pair of polynomials, $a(x)$ and $b(y)$. The stabilizer generators will fit into a box of size $(\deg(a) + 1) \times (\deg(b) + 1)$. For the X-type stabilizer tile, the polynomial $a(x)$ describes which of the horizontal edges in the highest layer of the box are colored, and the polynomial $b(y)$ describes which of the rightmost vertical edges are colored. For the Z-type stabilizer, $a(x)$ describes which of the vertical edges on the lowest layer are colored, and which of the horizontal edges on the leftmost layer. For example, if $a(x) = 1 + x + x^2$ and $b(y) = 1 + y + y^2$, the stabilizer tiles appear as follows:



BB codes arise by tiling a torus with these stabilizer tiles. In Refs. [18,29], the authors try to find planar versions of

these codes. We will now demonstrate that all codes they find can be recovered using our simple construction. For a visualization of the construction, see Fig. 2.

While it is straightforward to generalize this construction, we will for convenience assume that a and b have the same monomial terms. In this case, the minimal possible size of a box supporting the X- and Z-type stabilizers will be $B = \deg(a) + 1$.

In the first step of our construction, we draw a $(l - (B - 1)) \times (m - (B - 1))$ lattice of bulk stabilizers, which in the next step will lead—because of our choice of stabilizer tiles—to an $l \times m$ lattice of qubits. Then, we place $(B - 1)$ layers of X-type stabilizers on the bottom and top and $(B - 1)$ layers of Z-type stabilizers on the left and right. From the shape of the X- and Z-type stabilizers, it is clear that the uppermost $B - 1$ layers of vertical qubits are not supported by any Z-type stabilizer, and the leftmost $B - 1$ layers of vertical qubits are not contained in the support of any X-type stabilizer. In step 4 of the algorithm, we therefore remove these qubits, as also depicted in Fig. 2. Again, carefully examining the shape of the stabilizers, we observe that the uppermost $B - 1$ layers of X-type stabilizers, as well as the leftmost $B - 1$ layers of Z-type stabilizers, have empty support, since all of their support

TABLE II. All pairs of weight-6 stabilizers in 3×3 boxes achieving the code parameters $[[288, 8, 12]]$. Eight pairs of X- (left) and Z-type (right) stabilizers are shown, and one can obtain the rest by swapping the role of X- and Z-type stabilizers.

--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

has just been removed. Consequently, these stabilizers will be removed as well in step 4. It is clear that all remaining stabilizers have nonempty support and are independent. The final code has $2lm - l(B-1) - m(B-1) = lm + (l - \deg(a))(m - \deg(b))$ physical qubits and $(l - (B-1))(m - (B-1)) + (l - (B-1))(B-1) + (m - (B-1))(B-1)$ independent stabilizer generators, hence $(B-1)^2 = \deg(a)^2$ logical qubits. It is clear that starting with two different polynomials a and b and using rectangular boxes instead of square boxes, this analysis demonstrates a logical dimension of $\deg(a) \deg(b)$, which is precisely the result of Refs. [18,29]. It is also clear that our algorithm removes whole rows or columns of qubits and stabilizers—that is, the resulting code is still a hypergraph product code.

Appendix B: Stabilizers of weight 6 confined in 3×3 boxes yielding $[[288, 8, 12]]$ codes—We exhaustively searched over all possible weight-6 stabilizers confined in a 3×3 tile. We find that there are 16 different codes with these parameters. All of those stabilizers yielding $[[288, 8, 12]]$ codes are depicted in Table II.

Appendix C: Stabilizers of weight 8 confined in 3×3 boxes yielding $[[288, 8, 14]]$ codes—We exhaustively searched over all possible weight-8 stabilizers confined in a 3×3 tile. We find that there are more than 300 pairs of stabilizers yielding $[[288, 8, 14]]$ codes. In Table III, we depicted some of them.

TABLE III. Four selected pairs of weight-8 stabilizers in 3×3 boxes achieving the code parameters $[[288, 8, 14]]$.

--	--	--	--	--	--	--	--