Relaxing Hardware Requirements for Surface Code Circuits using Time-dynamics

Matt McEwen[1], Dave Bacon[2], and Craig Gidney[1]

[1]Google Quantum AI, Santa Barbara, California 93117, USA

[2]Google Quantum AI, Seattle, Washington 98103, USA

September 18, 2023

The typical time-independent view of quantum error correction (QEC) codes hides significant freedom in the decomposition into circuits for execution on hardware. Using the concept of detecting regions, we design time-dynamic QEC circuits directly instead of designing static QEC codes to decompose into circuits. In particular, we improve on the standard circuit constructions for the surface code, presenting new circuits that can embed on a hexagonal grid instead of a square grid, that can use ISWAP gates instead of CNOT or CZ gates, that can exchange qubit data and measure roles, and that move logical patches around the physical qubit grid while executing. All these constructions use no additional entangling gate layers and display essentially the same logical performance, having teraquop footprints within 25% of the standard surface code circuit. We expect these circuits to be of great interest to quantum hardware engineers, because they achieve essentially the same logical performance as standard surface code circuits while relaxing demands on hardware.

1 Introduction

Traditionally, quantum error correcting (QEC) codes are defined by a static structure of stabilizers [Sho95; CS96; Ste96; Kit97]. For example, the surface code [BK98; Den+02] is usually introduced without time-dynamics as an unchanging set of stabilizer terms to repeatedly measure. Although this time-independent approach to QEC is appealing in its simplicity, performing quantum computation on hardware requires dealing with time dynamics. At the logical level, time dynamics are unavoidable because logical computation applies operations changing the structure of the QEC circuit as it is being executed [BK05; Hor+12]. At the physical level, time dynamics are unavoidable because stabilizer measurements are not atomic operations available on hardware [Fow+12]. The required stabilizer measurements must be decomposed into layers of native hardware operations, which cause the state of the system to vary from moment to moment as the layers execute. While not all approaches to fault-tolerance rely on repeated stabilizer measurements, other approaches like single-shot QEC [Fuj14; Bom15] still involve the execution of a stabilizer circuit with non-trivial time dynamics. Some recent works have begun to explore the time dynamics of codes, including in Floquet codes [HH21; HH22; AWH22; Pae+22; GNM22]. However, these approaches remain close to the current standard, focusing on the evolution of code stabilizers and the propagation of errors over time [Got22].

Experimentally implementing QEC codes presents an impressive challenge. The surface code is a popular candidate because it presents an excellent compromise of good logical performance and achievable demands on hardware [Fow+12]. The surface code permits simple decomposition into a circuit via the addition of *measure qubits*, with a resulting qubit grid requiring only four-fold local connectivity in 2D, and using a cycle depth of only four layers of CNOT or CZ gates. At the same time, these circuits for the surface code display impressive logical performance under

Matt McEwen: mmcewen@google.com
Dave Bacon: dabacon@google.com
Craig Gidney: craig.gidney@gmail.com

realistic error models [Kri+22; Goo+22] and can be decoded efficiently by matching [Den+02; Fow13b; Hig21]. Previous attempts to construct circuits with more relaxed hardware requirements have faced significant challenges. Lower connectivity either demands unreasonable overhead in the cycle depth or the use of alternative codes sacrificing logical performance [Bac05; Cha+20; Sun+22]. Further challenges are presented by various non-ideal realities of hardware, such as the presence of leakage states. Modifications to improve the code's resilience to such effects also typically introduce additional overhead and harm logical performance [Fow13a].

In this work we aim to provide an alternative foundation of concepts for reasoning about quantum error correcting *circuits* as opposed to quantum error correcting *codes*. In particular, we highlight the concepts of *detectors* and *detecting regions* as a generalization of code stabilizers to the time-dynamic circuit picture. As evidence that these concepts are useful, we present several circuits which are improvements over the standard circuit decompositions of the surface code. These improved circuits still implement the surface code and so enjoy essentially equivalent logical performance. At the same time, they relax demands made on the hardware that implement them. These circuits were all inspired by looking at and modifying the time dynamics of the detectors in the standard surface code circuit.

Due to the difficulty of implementing QEC experimentally, and the resulting breadth of hardware architectures aiming to implement QEC, progress is often focused on exploiting hypothetical hardware specifics for better error correction performance. This kind of work is essentially in the form "If hardware can do X then we could do Y". Recent examples of such progress include new codes exploiting strong noise bias [Tuc+19; Bon+21], long-range connectivity [BE20; BE21; BK22; PK21; Rof+22], or hardware parity measurements [LGB10; DS12; RPB18; Rea+22; Liv+22]. Our work strives to be of the complementary form: "Hardware doesn't need to do X because we can do Y": We show that designing the time-dynamic QEC circuit directly can allow for relaxation of the hardware requirements.

In particular, we show that efficient surface code circuits can be constructed using three couplings per qubit rather than four, using ISWAP gates instead of CNOT or CZ gates, and with improved resilience to leakage by involving measurements on all physical qubits. All three of these exemplar circuits represent an improvement over the state of the art, performing the surface code with essentially the same logical performance, and maintaining the use of only four layers of entangling gates rather than adding overhead in circuit depth. The improvements shown in each of these circuits are not mutually exclusive or exhaustive. We also discuss combining the improvements from these constructions, as well as benefits they provide beyond relaxing requirements on hardware such as enabling lower cost logical compilation. That the existence of these circuits is surprising speaks to the usefulness of directly constructing QEC circuits rather than QEC codes.

1.1 Organization

This paper is organised as follows: In Section 2, we provide some background on approaching fault-tolerant circuits using detecting regions, and the tools we used to explore those concepts and make the following constructions. We also discuss relevant hardware constraints that motivate our constructions. In the next three sections, we present our major results in the form of three improved circuits for the surface code: In Section 3, we provide circuits for a hex-grid, requiring only three couplings per qubit to implement the surface code. In Section 4, we provide circuits that use ISWAP gates to perform the surface code. In Section 5, we provide circuits for the surface code that exchange the roles of data and measure qubits in each cycle. In all three cases, we explain the circuit construction in terms of detecting regions, and rigorously benchmark the performance of each circuit, demonstrating essentially the same logical performance as the standard surface code. Finally, in Section 6, we discuss combining these constructions, present a table of all 24 included circuit constructions, and provide some outlook and commentary on future constructions using these concepts.

Following the main text, we provide several appendices discussing parts of our work in more detail. In Appendix A we provide more rigorous definitions regarding the propagation of detecting regions. In Appendix B, we present the equivalent of two of our results in the repetition code for pedagogical purposes. In Appendix C, we discuss an applications of the walking surface code circuits beyond relaxing hardware requirements, in improving logical circuit compilation. In Ap-

pendix D, we detail the methodology we used for numerically benchmarking our constructions, including detailing the noise models used. In Appendix E, we provide summary benchmarking of all our constructions. Finally, in Appendix F, we provide some convenient links for opening circuits for our constructions in an online interactive tool.

Finally, we also provide a set of supplementary figures available as an ancillary file with this work, including visualization of each circuit we benchmark along with more traditional plots of error correction performance. This file can also be found in our data repository found at `zenodo.org/record/7587578` [MBG23], along with a full description of each circuit we benchmarked, the results of our benchmarking, and all major assets used in this paper. The code used to produce the circuits, perform the benchmarking, and make figures is available in our code repository at `https://github.com/Strilanc/midout`.

2 Concepts and Tools

Fault-tolerance is the property that a circuit overall can be more reliable than the faulty gates that make it up. This is a very desirable property for circuits we want to run on hardware, where noisy gates might otherwise limit our chances of success to unacceptably low levels. When a QEC code is described as fault-tolerant, we usually mean that it provides us some strategy to make fault-tolerant circuits. The task of constructing fault-tolerant circuits is often approached in this way: First, choosing code stabilizers; second, producing a *cycle circuit* that measures those stabilizers; third, repeating that cycle circuit to build a locally fault-tolerant circuit chunk implementing the code; and fourth, applying some strategy to execute logical computation without disturbing that local fault-tolerance. In this section, we aim to introduce an alternative paradigm for approaching fault-tolerant circuits more directly than via choosing stabilizers, without disturbing the desirable properties of codes or requiring a new strategy for decoding or logical computation.

We start by recasting the typical approach to static stabilizer codes in terms of propagating stabilizers in the circuit, highlighting how the state of the system changes as the stabilizers are measured by the cycle circuit. We then introduce the concepts of detectors and detecting regions, using the same propagation rules to understand the sensitivity of the circuit to errors. Finally, we use these concepts to address the fault-tolerance of the circuit directly, rather than via the measurements of stabilizers. We argue that detecting regions are a useful primitive to not only the stabilizers but other key concepts in QEC as well. This explanation aims to be pedagogical and rooted in the circuit picture, but we provide more rigorous definitions in Appendix A.

Following this, we discuss the software tools we used in exploring these constructions and some of the relevant hardware constraints that motivated us in constructing new circuits.

2.1 Mid-cycle states

The stabilizer formalism [Got97] permits analysis of Clifford circuits by simply following Pauli terms around the circuit. We track the state of the system using Pauli terms that stabilize the current state, and each applied operation transforms these stabilizers [Got98]. We define these transformations via their *stabilizer flows*, which describe how Pauli terms before and after a gate operation are related, as detailed in Appendix A. The simplicity of this approach underlies the popularity of stabilizer codes for quantum error correction.

Consider the CSS surface code: It is typically defined and depicted as a checkerboard of interlocking weight-4 X and Z stabilizers. We can define a simple circuit using an ancillary *measure qubit* and four layers of CNOT gates to measure a stabilizer, which we refer to as the *standard surface code circuit* throughout and show in Figure 1. When this cycle circuit is considered as a whole, we can see that it preserves the code stabilizers, leaving us at the end of the cycle in the same checkerboard state we started in. In this picture, the code stabilizers are persistent and unchanging.

However, during the execution of the cycle circuit, the code stabilizers are transformed by each layer of operations, only returning to the state described by the initial checkerboard at the end of the cycle. Further, the code stabilizers represent only around half of the stabilizers present at any given time. At the start of the cycle, each measure qubit provides an additional single qubit

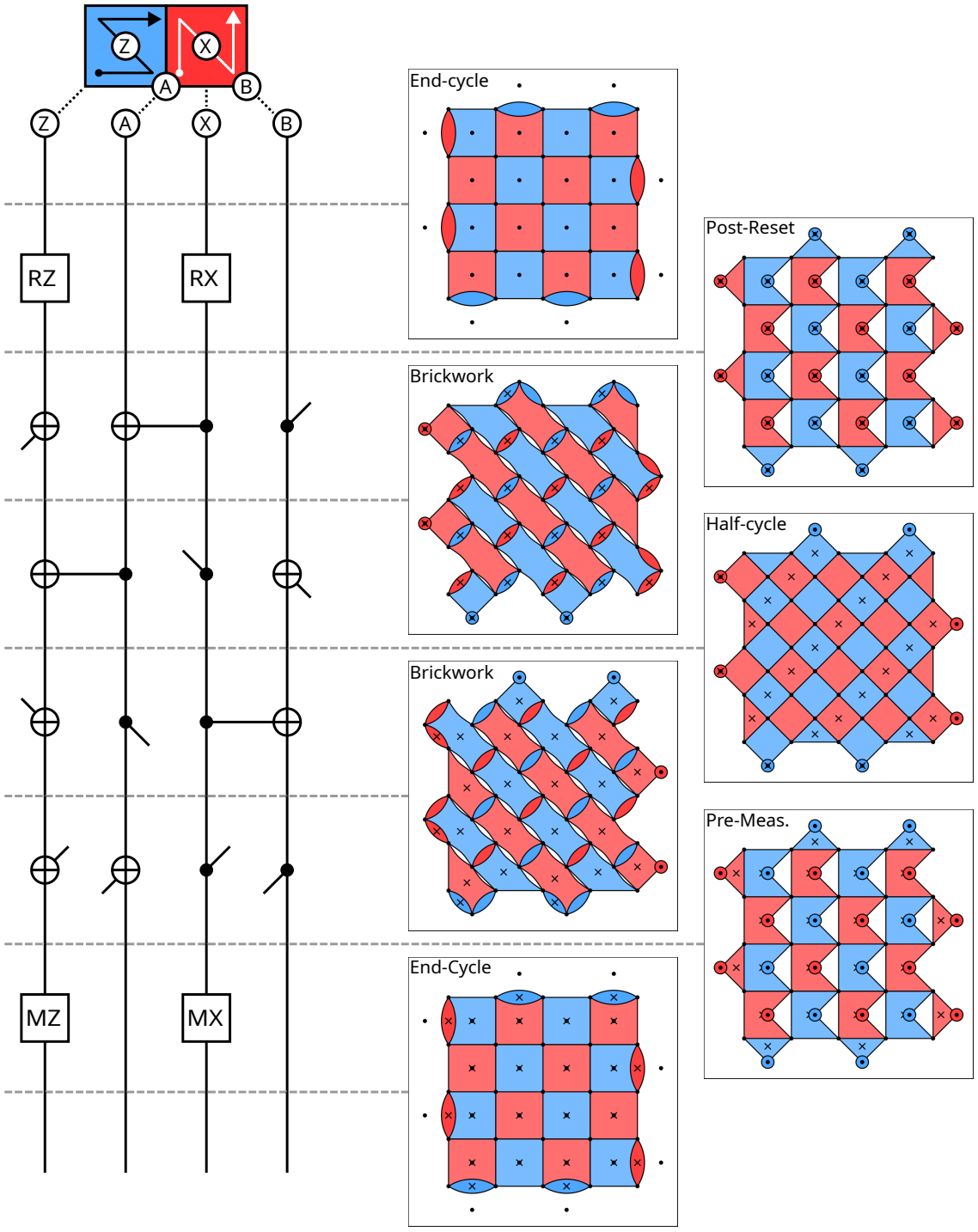


Figure 1: **Mid-cycle States in the Standard Surface Code Circuit.** Left: The standard circuit for the surface code, shown using CNOT gates on two measure qubits (X and Z) and two data qubits (A and B). Time proceeds down the page. Diagram at the top indicates the order in which the data qubits are interacted with. Right: A visual representation of the state at each point during the circuit. Each shape represents a single stabilizer term consisting of either Z (blue) or X (red) Pauli elements. The corners of the shape indicate the qubits included in the term, with circles indicating single qubit terms. Stabilizer terms introduced by the reset gates are marked with an X.

stabilizer enforced by the measurement or reset[1]. Figure 1b shows how both the code stabilizers and ancillary stabilizers evolve as the circuit is executed. For reasons we will explain later in Section 2.2, we have made a motivated choice not to show the measure qubit stabilizers before the reset layer and to include the measure qubit stabilizers into the code stabilizers after the layer of reset operations; for now, this is simply a change of generators which doesn't affect the described state.

For convenience, we name these *mid-cycle* states as follows: Between the measurements and resets, the *end-cycle* state is shown as the familiar checkerboard pattern. Immediately following the reset gates, we see the flag-like pattern of the *post-reset* state. Here, we show a full generating set of stabilizers, including the measure qubit stabilizers. This state is equivalent to the end-cycle state when the single measure qubit stabilizers are included, as the weight-4 square code stabilizers and the weight-5 flag-like terms differ only by those single qubit stabilizers. The full set of stabilizers in both cases generate the same overall stabilizer group. This pattern is transformed by each subsequent layer of CNOT gates, first into a brick-wall-like pattern we call the *brickwork* state, then into a checkerboard pattern rotated by 45 degrees that we call the *half-cycle* state, then into a modified version of the brickwork state, and finally back into a flag-like pattern at the *pre-measure* state, returning us to the checkerboard pattern for the end-cycle state after measurements. The half-cycle state is remarkable for being an unrotated surface code state, as originally proposed for the surface code [Kit97; BK98]. The half-cycle state is a surface code state with the same distance as the end-cycle state, but twice the number of qubits involved and twice the number of stabilizers - the usually ignored measure qubits stabilizers have become included into the code state.

Recognising these mid-cycle states is helpful for three reasons. Firstly, they provide helpful checkpoints for understanding circuits. The mid-cycle states break the problem into smaller pieces which can be analysed and understood separately, rather than thinking of the entire cycle as a single operation that preserves the code stabilizers. Secondly, it shifts the focus from the qubits and their roles in the circuit toward the detecting regions. How errors are detected becomes more clear, and the traditional assignment of specific roles to specific qubits becomes de-emphasised; all qubits are equally covered by detecting regions. Finally, these states provides some insight into what a surface code circuit should achieve, and inspiration on how it might be done differently. For instance, the half-cycle state does not "remember where it came from" and can be mapped back to an end-cycle state in more ways than the familiar one embodied in the standard circuit.

To explore these freedoms, and to better approach the necessity of fault-tolerance provided by repeating the cycle circuit, we need to reach for a more general concept rooted in circuit approach.

2.2 Detectors and Detecting Regions

In fault-tolerant stabilizer circuits, errors are identified by noticing violations of ideal circuit behaviour. In particular, circuits often feature small sets of measurements that should display a deterministic parity under noiseless execution. We refer to these sets of measurements as *detectors*. In an experiment, recording a detector's measurements with an overall parity different to the expected parity is clear evidence of an error occurring, which we call a *detection event*. This approach is most natural for stabilizers codes achieving fault-tolerance via repeated measurement of the stabilizers, but also works for related stabilizer QEC strategies, including single-shot QEC [Fuj14; Bom15], cluster-state based QEC [RHG06; Bom+21], subsystem codes [Bac05; Bra+12] and flag-qubits [CR18; CC19; CR20].

For any given detector, we can ask where errors could be inserted into the circuit to affect that detector; essentially the region of the circuit where the detector is sensitive to errors. We follow Gottesman [Got22] in considering a circuit as made up of *locations*, essentially qubits between operations. Each detector will be sensitive to specific errors in a finite set of locations in the circuit only, which we call the *detecting region*.

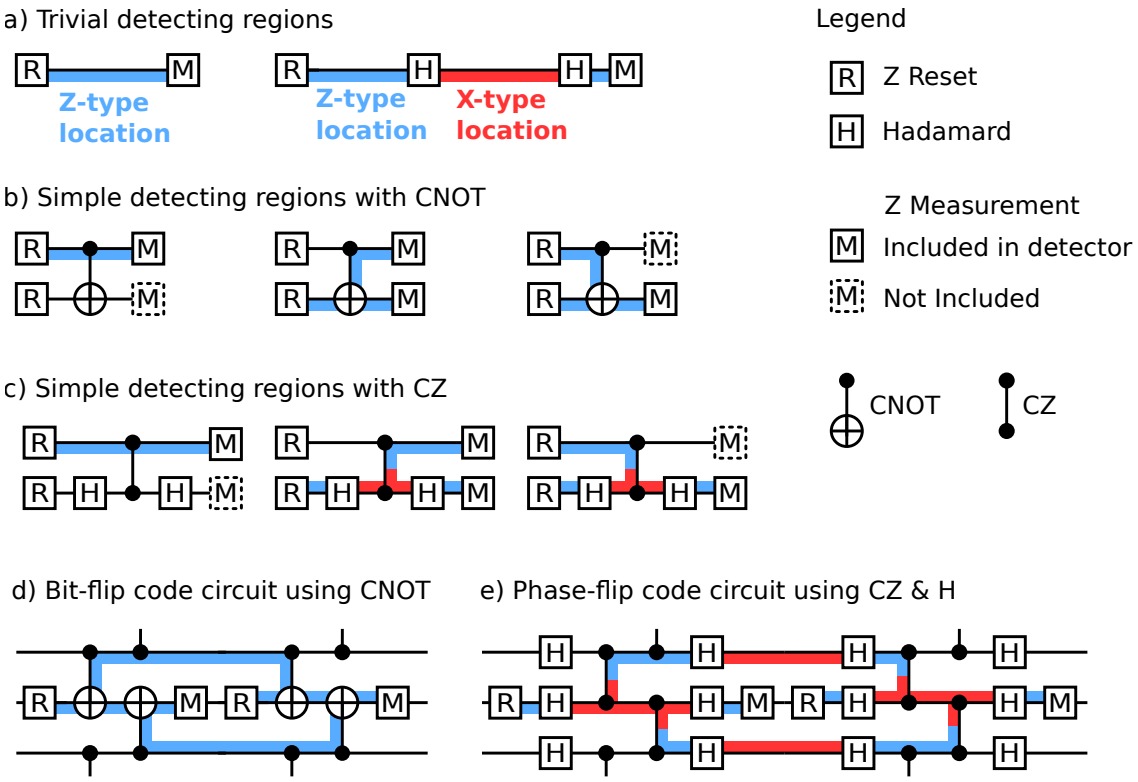


Figure 2: **Examples of detecting regions.** a) Detecting regions for 1-qubit circuits. Here, the detectors are just the single measurement in the circuit, which should have a deterministic outcome in the absence of noise. Z Reset gates produce states that are stabilized by Z, indicated by a Z-type section of the detecting region (blue). Z Measurements terminate Z-type sections of the detecting region. Hadamard gates change the type of the detecting region as it moves through the circuit, producing X-type locations (red). b) Three copies of the same simple circuit with a CNOT gate, indicating the three possible detectors and their detecting regions. Any two of these detectors form a generating set for the full set of detectors. c) The same circuit and detectors compiled into CZ and Hadamard gates. Note that CZ gates connect Z-type (blue) locations to X-type locations. d) A part of a circuit for a bit-flip repetition code, using CNOT and Reset gates. The two consecutive measurements shown form a detector. The associated detecting region is highlighted (blue), and is Z-type throughout. The region covers neighbouring data qubits during the measurement, reflecting the bit-flip code's ZZ code stabilizers. e) A part of a phase-flip repetition code, using CZ and Reset gates. Similarly, the two consecutive measurements shown form a detector. The associated detecting region is highlighted, containing locations of both Z-type and X-type. The regions include the data qubits as X-type locations during measurement, reflecting the phase-flip code's XX code stabilizers.

2.2.1 Examples of detecting regions

Figure 2 shows some simple detecting regions and associated detectors. A Z-basis measurement included in a detector is sensitive to errors immediately before it that anti-commute with Z. We can represent this by marking those locations in the circuit with Z-type. An isolated reset and measurement in the same basis forms the smallest non-empty detector, where the detecting region covers the one location in the circuit, as illustrated in Figure 2a. All operations relate parts of the detecting region in the same way they relate Pauli terms being commuted through them, which we refer to as their *stabilizer flows*. We define and discuss this concept in detail in Appendix A. Operations with stabilizer flows that relate different Pauli types will also change the type of the detecting region as it is propagated; for example, Hadamard gates with a Z-type on one side have X-type on the other.

A detecting region then consists of a set of locations in the circuit, along with a Pauli type at each location. A detection event will be caused by any inserted error that anti-commutes with the type of the detecting region at its location. We say that the detecting region is *sensitive* to such an error.

Detectors and detecting regions underlie the fault-tolerance of repetitively measuring code stabilizers. Consecutive measurements of the same stabilizer will agree under noiseless execution, and so pairs of consecutive measurements of the same code stabilizer form a detector in the bulk of the circuit.

Detecting regions are easier to visualise in 1-dimensional classical codes such as repetition codes. Figure 2d and Figure 2e show the detecting regions for bit- and phase-flip repetition codes respectively, defined by neighbouring ZZ and XX code stabilizers respectively. The detecting regions in the bulk of these codes consist of a small, closed and local part of the circuit, visually indicating how much of the circuit each detector is responsible for.

We can transform easily between detectors and detecting regions. Given a detector, we can find the detecting region by simply propagating Pauli types backward from the included measurements, following stabilizer flows in reverse and backward-terminating on appropriate resets. This process is deterministic (given that the measurement parities are deterministic), and will produce the detecting region. Given a detecting region, the corresponding detector is simply the measurements the region terminates on. We note that the task of choosing appropriate detectors and detecting regions given an un-annotated circuit is a non-trivial task, and one that can have a large impact on logical performance.

2.2.2 Overlapping structure of detecting regions

Given that each detecting region is responsible for some part of the circuit, it stands to reason that a good set of detectors should have detecting regions that cover all relevant parts of the circuit. These detecting regions should also overlap, such that any individual error is detected by multiple regions and provides a usable syndrome.

In the repetition and surface codes, the bulk detecting regions cover locations in two neighbouring cycles, rather than one cycle as one might naively expect. They also cover the code stabilizer at the time-slice between those cycles. Figure 3a illustrates this for the bit-flip repetition code, where the time-slice between measure and reset gate layers shows the existing region covering the ZZ stabilizer. During a single cycle, two detecting regions coexist; one emerging from that cycle's reset gate and *expanding* to cover the stabilizer, and one *contracting* from the code stabilizer to terminate on that cycle's measurement gate. Halfway through the cycle, we can see the regions collectively produce the same pattern of stabilizers as the typical code state (ZZ on neighbouring qubits), but involving all measure and data qubits, rather than only the data qubits. This is the equivalent of the half-cycle state previously discussed for the surface code.

In the bulk of the repetition code, each location is covered by exactly two detecting regions, meaning an inserted bit-flip error will be noticed by two detectors, as illustrated in Figure 3b. This is typically represented in terms of the resulting error graph, shown in Figure 3c. Here, each

[1]We have d^2 data qubits and $d^2 - 1$ measure qubits in a distance d patch. We have 1 logical observable, and $d^2 - 1$ code stabilizers touching the data qubits, and so obviously need another $d^2 - 1$ stabilizers. We have one such stabilizer on each measure qubit, as noted.

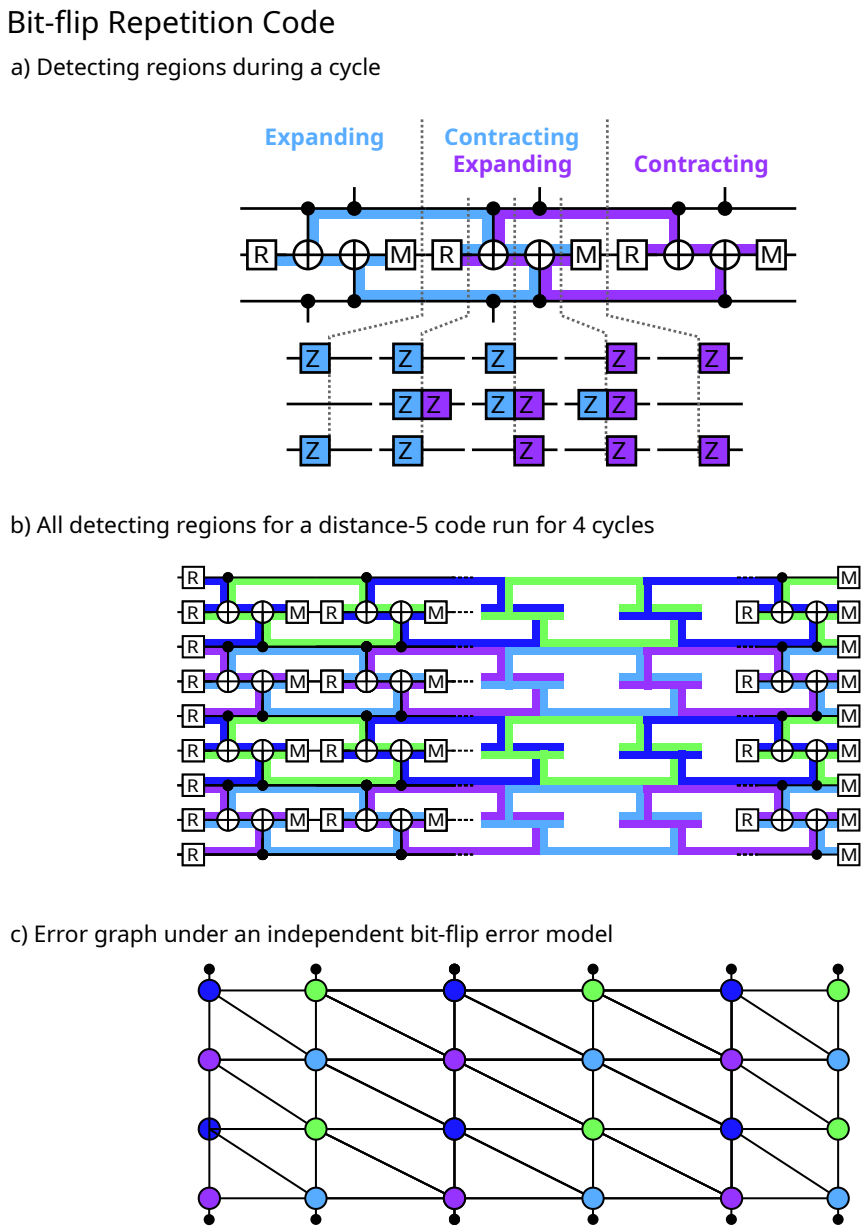


Figure 3: **Overlapping detecting regions in the repetition code.** All locations of all regions shown here are Z-type. a) Two neighbouring detecting regions of the bit-flip repetition code (blue, purple), shown over three cycles. Each detecting region covers two cycles: In the first cycle, it emerges from a reset on the measure qubit and expands to cover the code stabilizer. In the second cycle, it contracts from the code stabilizer down to terminate on a measurement. During the middle cycle, where the two shown regions co-exist, time-slices of the detecting regions are shown. b) All detecting regions for a distance-5 bit-flip repetition code. The detection regions overlap such that all locations in the circuit are covered by two detecting regions, except the boundaries which are covered by one. Note the smaller detecting regions at the start of the code that emerge from resets on both measure and data qubits. On cycles three and four, we omit the circuit elements to emphasise that the detecting region shapes alone are sufficient to understand the implementation. Note the final detecting regions terminate on measurements on both the measure and data qubits. c) The associated error graph for the distance-5 bit-flip code, illustrating the correspondence between error graph edges and overlaps in the detecting regions. Nodes on the boundary feature single-ended edges, corresponding to sections of the circuit not overlapping with any other detecting region.

detector is represented by a node. Edges between nodes indicate a possible bit-flip error that will be noticed by these two detectors. Under an independent bit-flip error model, these edges correspond directly to overlaps between the relevant detecting regions. The structure of these overlaps, and the resulting structure of the error graph, makes the repetition and surface codes amenable to decoding by matching [Den+02; Fow13b; Hig21].

Detecting regions are a useful primitive concept, as they single-handedly define relevant QEC concepts, as follows:

- The measurements that the region terminates on defines the corresponding detector.
- The overlaps between regions define the edges of the error graph under an independent Pauli error model, just as the detectors define the nodes of the error graph.
- The shapes of the regions define the circuit operations via the implied stabilizer flows.

As such, provided with only an overlapping structure of detecting regions, we have defined both the code and the circuit in a natural way.

2.2.3 Detecting Region Slices

Time-slices of the detecting regions are also helpfully related to the code stabilizer picture. A time-slice of a detecting region is a stabilizer of the state at that point in the circuit. Figure 4 shows four detecting region slices after subsequent layers of the standard surface code circuit.

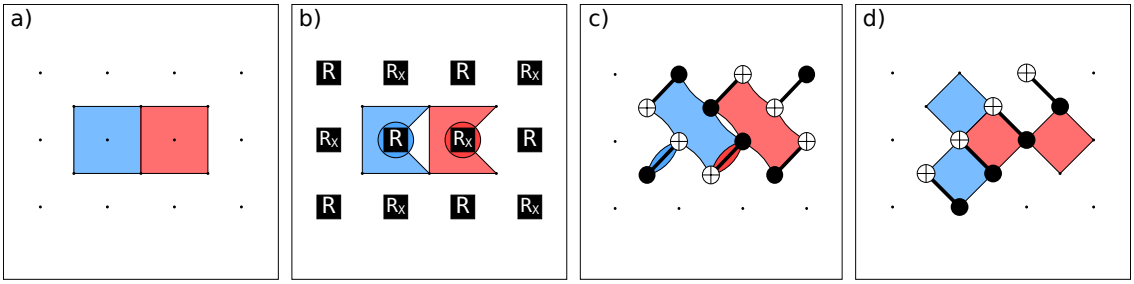


Figure 4: **Detecting region slices in a 2D circuit.** Dots indicate qubits locations in a 2D grid. Each shape corresponds to a slice of a detecting region including the qubits at each vertex. Colors indicate the Pauli type, with X-type regions being red and Z-type regions being blue. a) Two detecting region slices representing detectors as they are covering the code stabilizers of the surface code. b,c,d) Four detecting region slices in each subsequent time-slice of the standard circuit, with intervening gate layers shown as an overlay. Two new detecting regions introduced by the reset gates in (b) are shown, in addition to the two from (a).

In the language introduced by Aaronson and Gottesman [AG04] for simulating stabilizer circuits, a time-slice of a detecting region is a possible term in the stabilizer tableau, one that additionally corresponds to a specific detector. In the language introduced by Hastings and Haah [HH21] for Floquet codes, a time-slice of a detecting region is a generating term of the instantaneous stabilizer group (ISG) that additionally corresponds to a specific detector. The mid-cycle patterns discussed earlier were represented as time-slices of detecting regions, rather than as any other choice of stabilizer generators. This makes them both more aesthetically pleasing and more meaningful - again, each shape corresponds directly to a specific detector.

However, it is not the case that the time-slices of the detecting regions always form a complete set of terms for the stabilizer tableau or ISG. Some circuit locations are not covered by any detecting region, despite still being in the support of a stabilizer term. Examples include immediately after measurement but before reset, as in Figure 4a, and on gauge qubits in subsystem codes. These are locations where an inserted error cannot contribute to a logical error and does not need to be detected, and so does not need to be included in any detecting region. This motivated our choice to ignore the measure qubit stabilizers at the end-cycle state in Figure 1; they are not relevant to error correction until after the reset gate, where they are again included in detecting regions.

The development of new code circuits, and especially understanding their behaviour at boundaries, can be greatly aided by constructing them in terms of detecting regions. This is especially

true when considering how different detecting regions fit together to cover the space-time of the circuit with an appropriate overlapping structure, as we will discuss in our main results.

2.3 Detecting Regions in the Surface Code

We now revisit a standard surface code circuit. To avoid retreading ground and to introduce some new possibilities, we now consider the circuit as compiled for hardware using only Z-type reset and measurements, CZ and Hadamard gates as shown in Figure 5. We now interpret the states directly as time-slices of the detecting regions.

Compared to the standard circuit using CNOTs, this circuit displays regions with different Pauli types at locations at the same time slice. The regions at each time-slice display a nice overlapping structure by construction. Each location in the circuit is covered by four detecting regions, two with Z-type and two with X-type. This produces the two connected components of the surface code error graph under X and Z errors respectively. It also makes clear the correlated nature of Y errors; wherever they are inserted, they anti-commute with four detecting regions rather than two.

We can also see that the mid-cycle states display more than just two kinds of detecting region slices. We further distinguish detecting regions by whether they are expanding or contracting; whether they emerged from a reset at the start of this round, or emerged in the previous round and began this round covering the code stabilizers. These two regions evolve differently, with the contracting regions getting smaller as the cycle circuit continues and the expanding regions growing to cover the code stabilizers. In our constructions, tracking which regions are expanding and contracting will prove important. Failing to contract a detecting region that has already expanded will make a larger detecting region, one responsible for more of the circuit, overlapping with more detecting regions and worsening logical performance.

The detecting regions picture allows us to emphasise that there are several equivalent ways of looking at the surface code cycle. The familiar end-cycle picture involved starting with the data qubits involved in a surface code state and the measure qubits un-entangled and ready to be used. Starting at the half-cycle state, we have a picture where all qubits are involved in a surface code state. During the cycle circuit, half of the stabilizers (those corresponding to the contracting regions) are transformed to occupy only single measure qubits and are measured (while the stabilizers corresponding to expanding regions transform to cover only data qubits), before we return to the half-cycle state. Naively, a surface code without measure qubits where only half the stabilizers are measured in each cycle sounds worse than a code with measure qubits where all stabilizers are measured in each cycle; in fact these are two descriptions of the same circuit.

Alternatively, we can also start with the pair of brickwork states: The two-body stabilizers can be measured via a common decomposition of a pair-measurement into a CNOT followed by single-qubit measurement, with another CNOT allowing us to reconstruct a (slightly different) brickwork pattern. Two layers of CNOTs allow us to move between the two brickwork patterns, essentially exchanging which stabilizers are two- and six-body. Again, this is simply an unusual description of the standard surface code circuit. The advantage of these alternative approaches to the cycle circuit will become more clear as we use them to explain new constructions.

2.4 Boundaries

So far, we have focused on detecting regions in the bulk of the circuit, but now turn to explicitly address what happens at boundaries, which can have a significant influence of the final circuit performance. First, we address the general strategy we use for temporal boundaries in all surface code circuits, and then provide some notes on the more complex spatial boundaries.

First, given a bulk circuit for the surface code, temporal boundaries (namely logical initialization and measurement) can be introduced very simply. Due to the CSS nature of the standard surface code circuit, we can implement logical measurement in the Z basis as follows: At any point in the circuit (but traditionally at the end of the cycle where measurements are already occurring) measure all the qubits in the Z basis. Delete any detecting region that anti-commutes with these new measurements, and any detecting region that commutes with them now terminates on them; these are the correct final round detectors to ensure the logical measurement is fault tolerant. Logical measurement is the X basis is very similar, but the qubits should be measured in the X

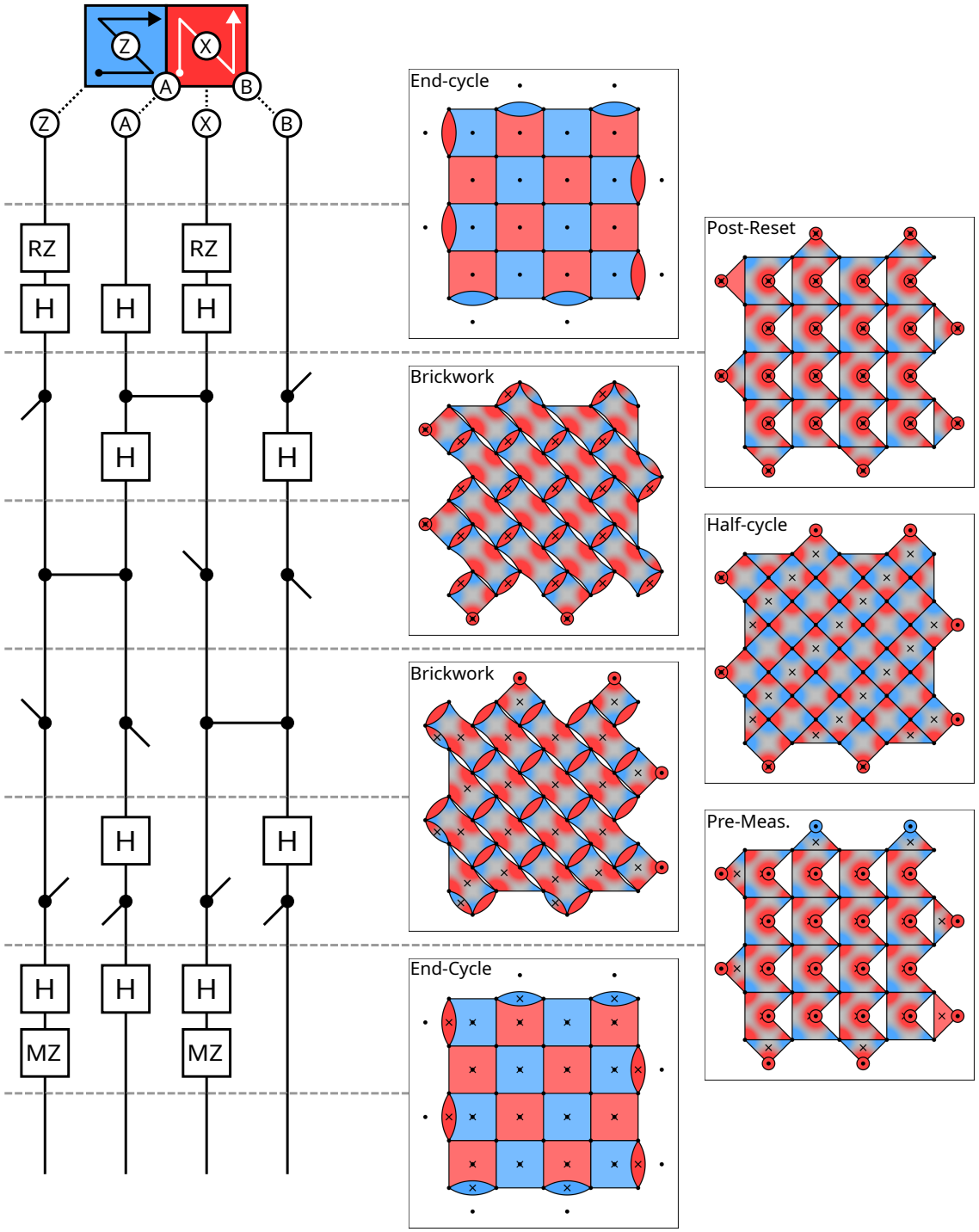


Figure 5: **Mid-cycle States in the CZ Surface Code Circuit.** Left: The circuit for the surface code compiled for hardware using H and CZ gates, shown on two measure qubits (X and Z) and two data qubits (A and B). Time proceeds down the page. Diagram at the top indicates the order in which the data qubits are interacted with. Right: A visual representation of the state at each point during the circuit. Each shape represents a slice of a detecting region. The corners of the shape indicate the qubits included in the term, with circles indicating single qubit terms. Colors indicate the Pauli elements Z (blue) or X (red), either for the whole shape or for each qubit in the slice. Expanding detecting regions are indicated by an X marker. Note the inclusion of unnecessary Hadamard gates on qubit A at the start and end of each round; these make CSS stabilizers at measurement, and canceling them yields XZZX/ZXXZ stabilizers at measurement.

basis. Logical initialisation is essentially the same process but time-reversed, with resets taking the place of measurements: At any point (but again traditionally at the start of the cycle where resets are already occurring) reset all the qubits in the Z or X basis for Z or X logical initialization. Again, delete or terminate the bulk detecting regions appropriately. We use this strategy for the temporal boundaries of all our constructions. Even when the circuit is compiled to gates that mix X and Z stabilizer terms, this strategy remains essentially the same. The circuit state is always only single qubit rotations away from a CSS-like state[2].

Spatial boundaries can also be found using a similar strategy, but this is far from optimal. In the original unrotated surface code [Kit97; BK98], drawing a square box containing $(2d-1)^2$ total qubits and simply removing all gates that cross this boundary is sufficient to make the circuit for a distance-d patch from a tiled bulk circuit[3]. However, unlike the temporal case, these spatial boundaries are wasteful. The rotated surface code [BM07] significantly improves the number of qubits used by exploiting a more complex boundary; appropriately including half the measure qubits around a square patch for $2(d^2)-1$ total qubits for a distance-d logical patch. This exemplifies the major differences in performance that changes in boundaries can make to a surface code construction.

For all of our constructions, finding efficient spatial boundaries presented a more interesting and difficult challenge than finding the bulk circuit. For each construction, we present a choice of boundaries that preserves the graph-like code distance [GNM22] (i.e. the code distance considering only errors that produce pairs of detection events). This is not to say that other boundaries are not possible, or even preferable. Various possible constraints or considerations, including from hardware (for example, which qubits are measurable in parallel, such as all qubits or only non-nearest-neighbouring qubits) or from logical considerations (e.g. densely packing logical qubit patches in a distillation factory) can place different demands on the circuit, which can be variously accommodated by different boundaries. Our strategy for finding boundaries involved simply iterating on circuit details, especially which gates to omit or include at the boundaries, until the detecting structure preserved the code distance and benchmarked well. As such, we consider developing spatial boundaries still more art than science, and we relied heavily on tools that sped up iteration over any deep insights into how to pick good spatial boundaries.

2.5 Software Tools

A major challenge to exploring QEC circuits is the intricate and interlocking nature of the fault-tolerant constructions. Getting a seemingly small detail wrong, from choosing the wrong measurements for a detector to choosing the wrong order for operations, tends to radically alter the logical performance of the circuit. While simple error correction circuits can be designed by hand, the constructions we present here necessitated the use of software tools to visualise and analyse the circuits. Software tools also proved vital in verifying the correctness of the circuit (such as in terms of code distance) and in benchmarking the true performance of the circuit.

In particular, we made extensive use of Stim [Gid21] and the tools it provides for representing and manipulating QEC circuits, for verifying them and for benchmarking them. Over the course of this work, Stim has been improved to include new tools for visualizing circuits and stabilizer states during circuits, which we made extensive use of in both exploring and explaining these constructions. Stim now also includes a prototype of an interactive circuit editor *Crumble*, which is also accessible at algassert.com/crumble. This tool was especially useful for finding appropriate boundary configurations for these code circuits, as it enabled rapid rearranging, inclusion and omission of entangling gates at the boundaries, showing the resulting change in the detecting structure. In Appendix F, we provide some convenient links for opening the circuits for our constructions in Crumble, which we hope will serve as a convenient jumping off point.

These tools were crucial in several ways. First, the tools automated critical tasks like verifying that a construction preserves the code distance, permitting cheap exploration because mistakes

[2]The states reached by standard circuit for the surface code in CZs therefore violate the letter but not the spirit of the CSS property of the surface code. This suggest we should more often consider the class of CSS-up-to-single-qubit-rotations codes

[3]To draw out the equivalence to the temporal boundary strategy, this is equivalent to performing single qubit measurements on all qubits outside the box between each layer of gates.

would be caught early, and helping build intuition for what changes to the circuit constitute mistakes. Second, the tools revealed new insights by making visualization and interaction easier. Visualizing the interlocking nature of many detection regions was the initial impetus for all the presented constructions. Crumble proved crucial both in finding effective circuits, especially in finding appropriate boundary detecting regions, and in understanding and explaining our circuits to each other. Finally, it proved important that our tools were flexible enough to handle these constructions. Despite ostensibly being designed to simulate traditional QEC code circuits, Stim operates on general annotated circuits and was able to analyse and benchmark our constructions without any redesign. A tool built specifically for the current paradigm of measuring stabilizer operators would be much less likely to achieve this without modification.

The importance of software tools used in this kind of work is often under appreciated, so we have chosen to emphasise the tools we used here with the aim of encouraging wider use and further development of such tools.

2.6 Hardware Requirements

The constraints under which we design error correction circuits come from difficulties and challenges we face in designing hardware. These constraints can depend sensitively on the hardware architecture being targeted. We choose to focus our examples on the paradigm of superconducting qubit arrays. As previously mentioned, the popularity of the surface code rests on the relatively acceptable demands it makes on hardware: square lattice connectivity, and a cycle containing only four layers of entangling CZ or CNOT gates. Further relaxing these hardware constraints generally comes at a significant cost, typically in compromising the error correction performance by introducing additional overhead in the cycle circuit.

Lower connectivities are generally easier to achieve in hardware, usually permitting higher couplings, lower crosstalk, fewer on-chip structures and control lines, and relaxing other constraints such as the frequency arrangement of qubits. Connectivity in a square lattice presents an achievable but difficult design modality [Goo+22; Kri+22], but cutting edge architectures featuring lower connectivity have also been produced [Sun+22]. However, lower connectivity has historically required an unreasonable swap overhead to implement the surface code, usually frustrating performance below threshold. Use of alternative codes better suited for lower connectivity generally feature comprimises in the logical performance, such as the heavy-hex code displaying a threshold in only one logical observable [Cha+20]. We present a circuit for the surface code that embeds on the lower connectivity hex grid without swap overhead in Section 3.

Circuit decompositions are usually expressed in terms of CNOT and CZ gates. However, other entangling operations are often naturally achievable in hardware, such as the ISWAP gate. For example, in superconducting architectures with tunable couplers, the ISWAP gate is less demanding in terms of frequency arrangement and may be performed at a higher fidelity than the CZ gate [Fox+20]. However, an error correcting code must be compiled to ISWAP gates without significant overhead before this can be taken advantage of. We present a circuit for the surface code using ISWAP gates in Section 4.

Finally, QEC circuit design generally neglects important non-idealities of the hardware, such as significant sources of correlated errors. In many architectures, including transmon qubit arrays, the ability of the qubit to be promoted out of the computational states into a higher energy leakage state presents such a source of correlated errors [Fow13a; Gho+13]. Such errors are not considered at the level of designing the QEC circuit itself, with attempts to address leakage generally involving adding minimal operations to the standard circuit attempting not to disturb the logical structure while achieving some suppression of the correlated error effect [GF15; BB19; BVT21; McE+21; Mia+22]. As such, these strategies add overhead to the standard circuit, producing a trade-off between leakage error suppression and the errors induced by the included operations themselves. We present a circuit that achieves such a benefit without any additional gate layers in Section 5.

In all three cases, the standard circuit places demands on the hardware that are considered achievable, but the circuit decomposition itself proceeds without feedback from what is preferable for the hardware. We regard this work as opening up new possibilities for tailoring the circuit decomposition of a code to the strengths of hardware designs, with the hope that this frees up further improvements in performance.

3 Hex-grid Surface Code Circuits

Surface codes are typically compiled to circuits on a square grid (the grid with Schläfli symbol {4,4}). In this layout, each qubit has four neighbors. Without using more layers of operations, it seems difficult to use a sparser connectivity because every qubit interacts with all of its neighbors in each cycle of the circuit. Measurement qubits interact with their four adjacent data qubits, in order to acquire all the parts of the four-body code stabilizer. The data qubits interact with their four adjacent measurement qubits, in order to make their contribution to the four stabilizers involving that data qubit.

Here, we show a circuit for the surface code with the same code distance and the same number of entangling layers as the usual circuit, but that executes on a hex grid (the grid with Schläfli symbol {6,3}). When instantiated in a 2D architecture, this means that each qubit needs only three couplers to neighbouring qubits, rather than the typical four.

We first address the circuit in the bulk, for which we provide two complementary explanations: a bottom-up circuit-focused approach centered around the half-cycle state and a top-down stabilizer-focused approach centered around the brickwork states. We then talk about the challenge of finding appropriate boundaries for surface code patches using this bulk circuit. Finally, we perform numerical benchmarking of the presented circuit.

3.1 Bottom-up Circuit-Focused Construction

Following our previous discussion of the half-cycle state in Section 2.3, we can approach this circuit as a new strategy for measuring half of the stabilizers of the half-cycle state without using any additional qubits.

As in the standard surface code, a four body contracting stabilizer can be measured by using two layers of CNOT gates; the first layer folds the stabilizer in half, from four body to two body; the second folds it again into a one-body stabilizer, which can be directly measured. This half-cycle circuit is pictured in Figure 6a. Distinct from the standard surface code, we then apply the same gates in reverse order, returning to the half-cycle state by the same route we left it. This basic action has required only three of the four nearest neighbour couplings on the square grid to be used. The same action can be used on a neighbouring X stabilizer at the same time; the first layer of CNOT gates folds the four-body X stabilizer in the opposite direction. The subsequent layer also folds the X stabilizer into a single qubit stabilizer which can be measured.

These fold-based half-cycle circuits tile nicely. It's possible to simultaneously measure entire diagonal columns of square stabilizers, alternating between X type and Z type, as shown in Figure 6b. This lets us measure of half of the stabilizers in the half-cycle state, which is the goal of a surface code cycle circuit.

Using this half-cycle and then its reverse differs from the standard circuit in two important ways: First, as mentioned already, it only uses three of the four available couplings in each square stabilizer to measure a column of stabilizers; and second, the cycle being reversed expands the stabilizer back to where it was contracted from, essentially exchanging which detecting regions in the half-cycle state are expanding and contracting. In the subsequent cycle, rather than executing the same cycle circuit, we must use a different cycle circuit aiming to measure the stabilizers corresponding to the contracting detecting regions, which are now in the columns adjacent to the ones we just measured. We can use the same strategy we have just presented to measure those columns as well; use two layers of CNOT gates to fold those stabilizers down to one-body stabilizers to be measured, and then reverse those layers to return top the half-cycle state with expanding and contracting regions exchanged. This is illustrated in Figure 6c.

Taking stock, we have constructed two cycle circuits starting and ending at the half-cycle state. One measures the even numbered diagonal columns of alternating Z and X stabilizers, and the other measures the odd numbered diagonal columns. If we alternate these two cycles, we measure all the half-cycle stabilizers every two rounds. This means we return to the original half-cycle state with the same pattern of expanding and contracting regions every two rounds. This distinguishes the cycles here from the usual surface code circuit, which returns to the same half-cycle state in every round, including the locations of the expanding and contracting regions.

These cycles must alternate: after one cycle, we return to a half-cycle pattern but with the

expanding and contracting detecting regions switched. In the absence of noise, the state described by these two patterns is identical; they have the same stabilizers. However, returning to the same noise-free state obviously does not imply that we can simply repeat one of the two cycle circuits, never learning half the stabilizers. In the presence of noise, these two states are not the same; different stabilizers will have been measured more recently and so have less accumulated error. In the detecting region picture, this is made even more obvious: at the half-cycle state, half of the detecting regions are expanding and have so far existed for only around 1/4 of their overall lifetime, whereas the contracting regions have already lived for around 3/4 of their lifetime. The change in where the expanding and contracting regions are located at the half-cycle distinguishes the two half-cycle patterns and enforces an overall 2-round-periodicity in the circuit.

The final important insight is that we can lay out the two cycle circuits in a compatible way, having them agree on which nearest neighbouring coupling to avoid using. This is shown schematically in Figure 6d. Each of the two cycles only uses half of the couplings in the direction of the second folding, with the other half being used in both cycle circuits. The resulting connectivity actually used is therefore only a hex grid, rather than the usual square grid.

3.2 Top-down Stabilizer-Focused Construction

Considering the brickwork patterns of the surface code (as shown in Figure 7), we see a pattern of two- and six-body stabilizers.

We can non-destructively measure a two body X stabilizer or a two body Z stabilizer via a common circuit decomposition: performing a CNOT, then a single qubit X or Z basis measurement, then undoing the CNOT. Since the two-body stabilizers in the brickwork states don't overlap, we can measure all these stabilizers in parallel. Both layers of CNOT gates operate along the two body stabilizers, and so do not connect the two middle qubits in any six-body stabilizer.

To construct a complete cycle, we also need to transition between the two brickwork states. As occurs in the standard surface code circuit, we can exchange the two- and six-body stabilizers using two layers of CNOTs, as indicated in Figure 7. This switching is visually clear in the circuit using CNOT gates, where we can see visually that the locations of the two body stabilizers in the two brickwork patterns are the same, but their Pauli type has switched. Both layers of CNOT gates operate perpendicular to the two body stabilizers, and again do not connect the two middle qubits in any six-body stabilizer.

Putting this together, we can execute the surface code cycle on a hex grid. Starting in a brickwork state, we:

1. Measure the two-body stabilizers using a CNOT layer, a measurement layer, and a CNOT layer
2. Exchange the two-body and six-body stabilizer using two CNOT layers
3. Measure the two-body stabilizers, again using a CNOT layer, measurement layer, and a CNOT layer
4. Exchange stabilizers again, returning to the original brickwork state, using two CNOT layers

This strategy uses two distinct cycles, and naturally embeds on a hex grid.

3.3 Boundaries

Having now addressed the bulk circuit, we turn to defining appropriate boundaries. We use the strategy described in Section 2.4 for temporal boundaries. The spatial boundaries can be most easily understood by looking at the half-cycle states. By comparison, the standard surface code half-cycle state (Figure 1) features alternating weight-3 and weight-4 stabilizers around all four boundaries.

Figure 8 shows slices of the full detecting regions for a logical qubit patch executing the three-coupler surface code circuit, illustrating the shape of detecting regions around the boundaries. The boundary construction can be most easily understood at the half-cycle state, for instance Figure 8.4.

Here, we see two *flat* boundaries on the top and left where detecting regions are truncated to weight-3, and two *spiky* boundaries featuring weight-1 detecting regions on the right and bottom. This contrasts with the standard surface code boundaries at the half-cycle shown in Figure 1, which has all four boundaries featuring alternating flat and spiky detecting regions. This gives us complete diagonal columns of detecting regions (also visualized in Figure 6) to measure out; specifically, the alternating pattern on gates in each diagonal column terminate nicely with a weight-3 stabilizer at one end (the flat boundary) but use a weight-1 region at the other end (the spiky boundary). Which boundaries are spiky is therefore determined by the spatial direction for the gate layers adjacent to measurement. The full circuit is provided in our data repository [MBG23]. The supplementary figures include a visualization of the full cycle circuit including only 4 detecting regions in the bulk (Supp. Fig. 5), which may aid the reader in tracking the evolution of one detecting region through the circuit. For this purpose, we also recommend opening the circuit in Crumble using the links provided in Appendix F.

We found appropriate boundary constructions using an iterative (or brute-force) approach. Starting with the detecting region slices at the half-cycle state, we iteratively removed gates and manipulated the detecting region termination points while attempting to preserve the graph-like code distance [GNM22]. To restrict the search space, we constrained ourselves to exploring circuits without changing the number of gate layers, without breaking the regular patterns of the gate layers, and where possible preserving that measurement layers apply only to qubits on one of the two neighboring sub-grids (often called data and measure qubits in the standard circuit). To enable faster exploration of the impact of adding and removing gates on the detecting structure, we recommend the use of interactive tools such as Crumble (as detailed in Section 2.5).

3.4 Benchmarking

Armed with the full description of the circuit, including boundaries, we now benchmark the circuit by numerically simulating a quantum memory experiment. The full details of our benchmarking strategy, including the noise model, are discussed in Appendix D. In particular, we benchmark the circuit as compiled for superconducting hardware, using relative error rates representative of current experimental errors [GNM22; Goo+22]. We compile to single qubit rotations, CZ gates and Z-type measurements and resets only, with the primary error parameter p being equal to the CZ gate error rate.

For brevity, we skip over intermediate benchmarking results and discuss only the *teraquop footprint*. This is the number of physical qubits required to produce a logical qubit displaying a one-in-a-trillion logical error rate over a $d \times d \times d$ space-time block. This metric encapsulates the most important information regarding the performance of the code: First, the vertical asymptote where the footprint diverges vs. error rate indicates the threshold of the code, which is important for estimating near term performance while hardware error rates remains near threshold. Second, the relative value of the footprint at aspirational physical error rates (such as around $p = 1 \times 10^{-3}$) indicates the sub-threshold scaling of the code. Given the very low logical error rate involved, the procedure for estimating the teraquop footprint involves linearly extrapolating the log logical error to higher code distances. The procedure used is included in our code repository. Further benchmarking plots for this code circuit are included in Appendix E, including the more traditional plot of logical error rate versus physical error rate from which teraquop footprint is extrapolated.

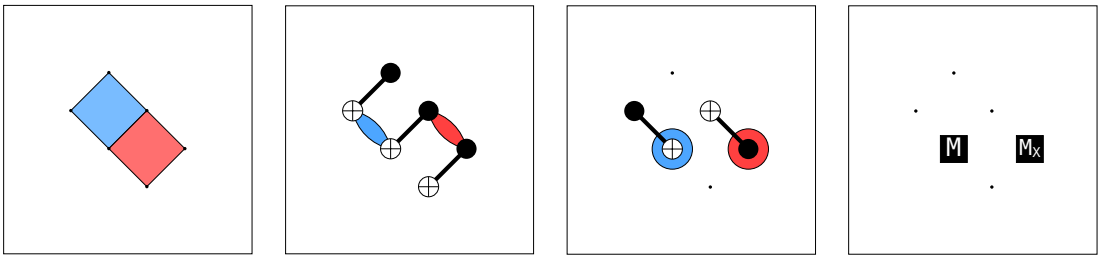
3.5 Summary

The hex-grid surface code circuit presents an appealing alternative to the traditional surface code circuit by reducing the connectivity required to implement the code at a minimal cost of final logical performance. Lower connectivity in hardware generally reduces costs in implementing an architecture, from relaxing frequency constraints to freeing up additional footprint on chip. Given the minimal change to the threshold, we expect near-term superconducting hardware tailored for the surface code can take advantage of the hex-grid circuit to improve physical error rates over those achievable using a square grid. However, the use of a hex grid also brings additional challenges; qubit and coupler yield is likely to be more costly in such an architecture, as we expect compatible subsystem codes to reduce the effective code distance more than in the square grid

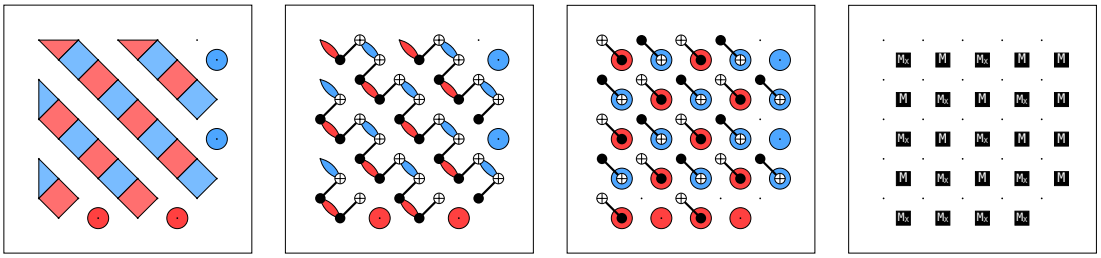
surface code case. In a similar vein, using the hex grid circuit in a square grid architecture with broken couplings provides additional freedom to avoid a reduction in code distance that imperfect yield would typically entail.

The approach taken by this circuit makes clear that the surface code does embed naturally on a hex grid, which opens the way to other similar constructions. In Appendix E, we additionally show benchmarking results for compiling the surface code to a heavy-hex grid [Cha+20; Sun+22] and semi-heavy-hex grid, both of which further relaxes the connectivity requirements relative to a hex grid. The circuits themselves are included in the supplementary figures and in our data repository. We also provide a compilation to an architecture providing heterogeneous entangling operations: The hex-grid circuit naturally decomposes into two-qubit entangling measurements along the connections for measuring two-body stabilizers and two-qubit entangling gates in the orthogonal direction for exchanging brickwork states. Given progress in conducting hardware two-qubit parity measurements [LGB10; DS12; RPB18; Rea+22; Liv+22], this kinds of architecture may also provide a compelling alternative to the traditional square grid.

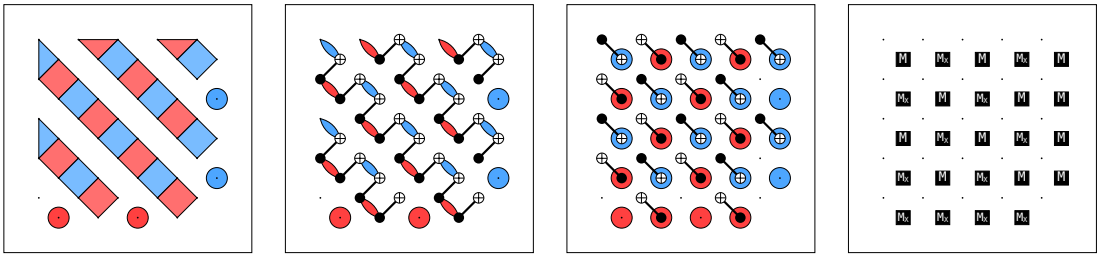
a) Folding four-body stabilizers



b) Measuring half of the stabilizers in the half-cycle state



c) Measuring the other half of the stabilizers in the half-cycle state



d) (b) and (c) superimposed

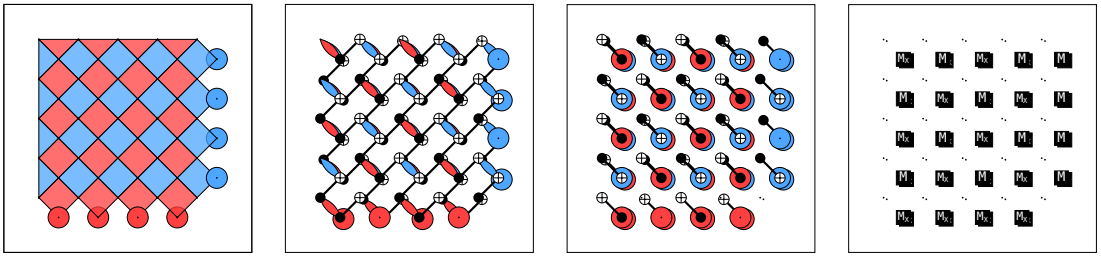


Figure 6: **Half-cycle Picture for the Hex-Grid Circuit.** a) A schematic indication of how to measure neighbouring square four-body stabilizers of different types using gates on only three out of four edges. b) An illustration of how folding can measure half of the stabilizers in the half-cycle state. These operations can be reversed to reconstruct the half-cycle state, but with expanding and contracting regions exchanged. c) The same strategy applied to the other half of the stabilizers in the half-cycle state. Again, these operations are reversed to reconstruct the half cycle state. d) Visually superimposes the two cycles (b) and (c), illustrating on the left that all stabilizers in a half-cycle state are measured by the two cycles, and that the resulting pattern of gates requires only a hex grid to be implemented.

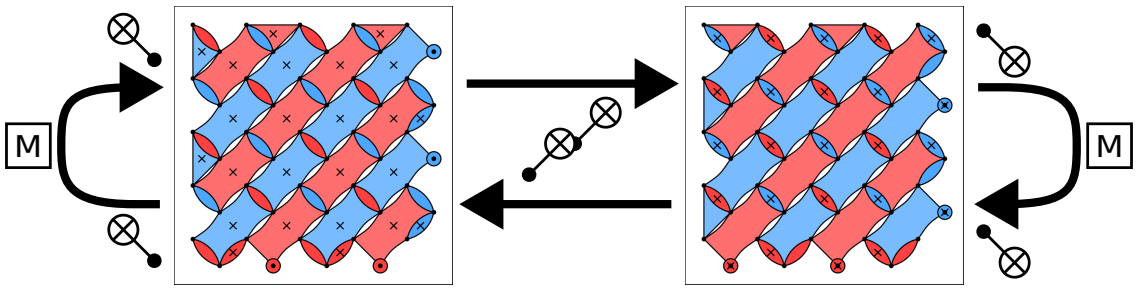


Figure 7: **Brickwork-state Picture for the Hex-grid Circuit.** A schematic indicating how the hex-grid circuit measures the appropriate stabilizers. Starting on the left, the two body stabilizers can be measured simply using a common decomposition; applying a CNOT between the two qubits, measuring one of them, and applying the same CNOT again, as indicated by the u-turn-like arrow. Changing between the two brickwork patterns can be easily achieved by applying two layers of CNOTs along the direction perpendicular to the two body stabilizers, as indicated by the straight arrows (center). X markers indicate six-body stabilizers that are contracted into two-body stabilizers when moving to the right. These two-body stabilizers can then be measured, as indicated by the u-turn-like arrow on the right.

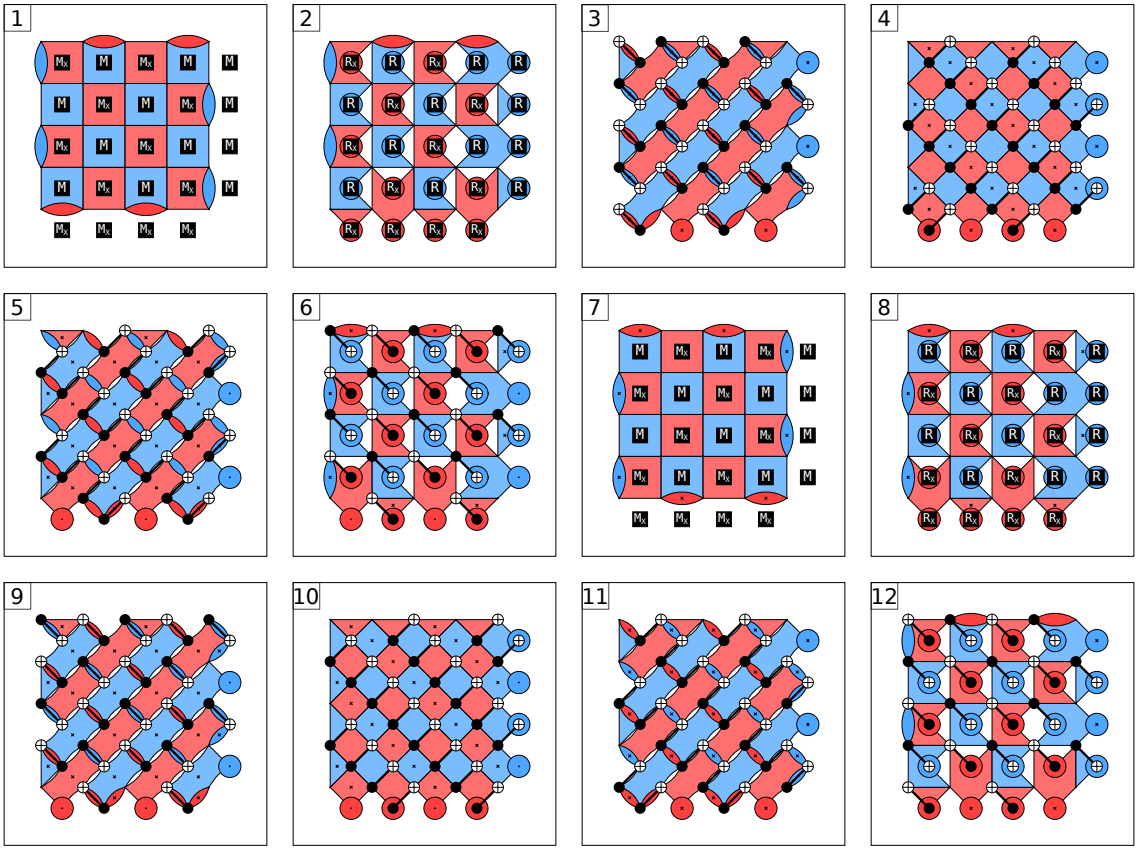


Figure 8: **The Mid-cycle States in the Hex-grid Circuit.** All 12 layers of the hex-grid surface code circuit over both distinct cycles. Pictured behind each layer of gates are the detecting regions slices immediately after that layer. Detecting regions introduced in layer 2 are marked with an X. They are expanding from layer 2 until layer 7, where they cover the usual surface code stabilizers, and are subsequently contracting.

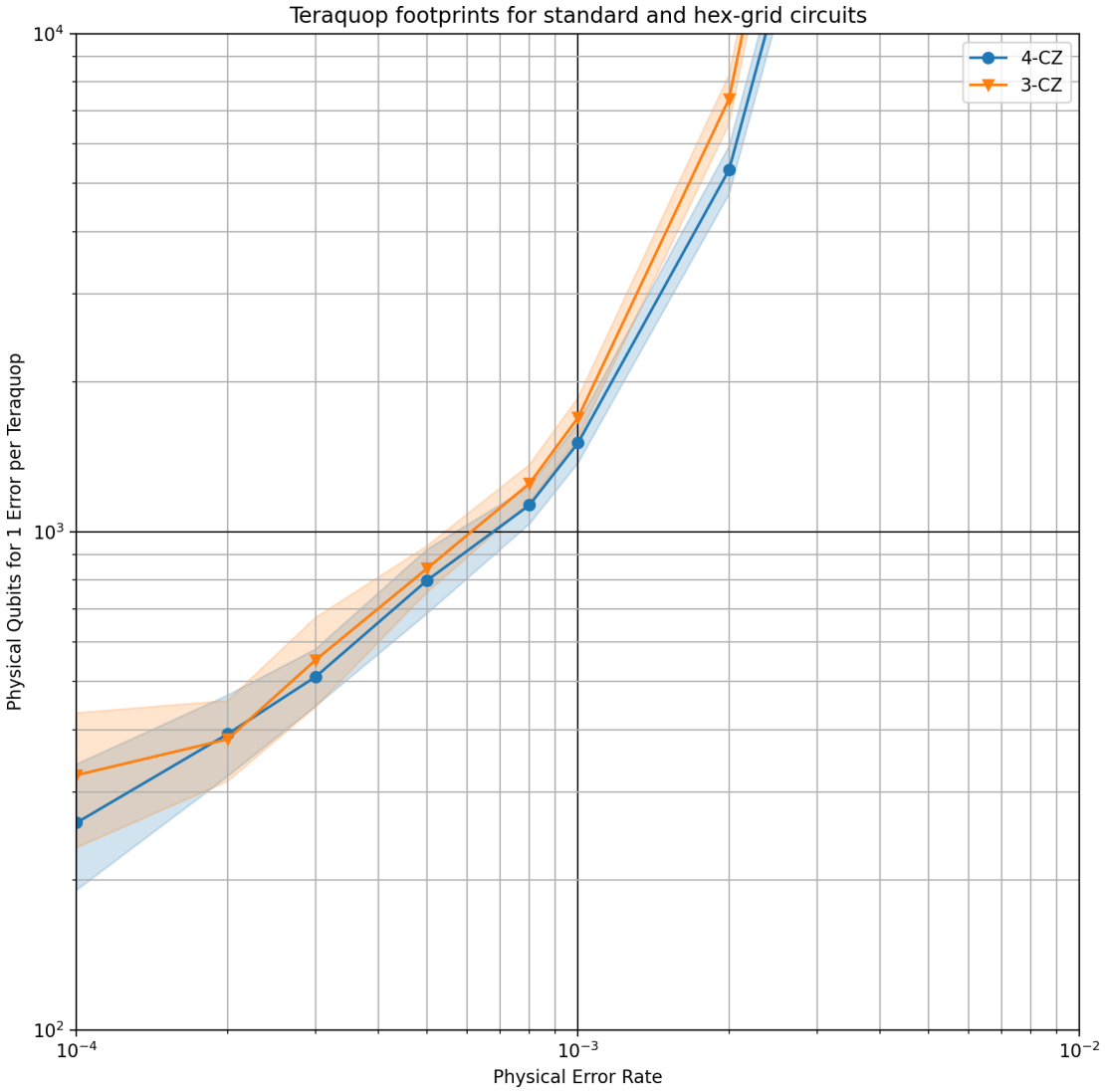


Figure 9: **Teraquop footprint estimates for the hex-grid surface code circuit.** The number of physical qubits required for a single code patch to achieve a logical error rate of 1×10^{-12} over a $d \times d \times d$ space-time block. Each curve combines memory experiments in both Z and X basis, and uses the `SI1000` noise model as described in Appendix D. `4-CZ` (blue circles) is the standard surface code circuit compiled to CZ gates, single qubit rotations, and Z-type measurements and resets. `3-CZ` (orange triangles) is the hex-grid circuit, using the same gateset.

4 ISWAP Surface Code Circuits

Surface code are traditionally constructed using CNOT-like gates. CNOT gates provide the theoretically simplest circuit implementation, corresponding most closely with the original introduction of the surface code [Kit97; BK98], and usually referred to as the CSS surface code. These operations naturally assemble the stabilizer value to be measured onto the measure qubit, producing circuit implementations that are easy to understand and manipulate.

Here, we show a circuit for the surface code using ISWAP gates. This circuit has the same code distance and the same number of entangling layers as the usual circuit. This permits the surface code to be operated without overhead on architectures that provide a native ISWAP-like gate, rather than demanding a CNOT-like native interaction.

First, we address the background of the CNOT-like and ISWAP-like gate classes in more detail. We then introduce the bulk circuit from the perspective of the half-cycle state, describe a choice of patch boundaries that preserves the graph-like code distance, and numerically benchmark the resulting code as a memory experiment.

4.1 Entangling Gates in Error Correction Circuits

Up to local single qubit Clifford gates, there are only four classes of Clifford two-qubit gates: Identity-like, CNOT-like, SWAP-like, ISWAP-like. This equivalence is made clear by the KAK decomposition of these gates, as discussed in detail in Appendix A.5. The Identity-like and SWAP-like gates are not entangling and cannot be used to construct a QEC circuit alone. CNOT-like gates are the traditional gates used to construct QEC circuits. Using the remaining gate class, the ISWAP-like gates, is addressed in this construction.

ISWAP-like gates can be understood as the product of a SWAP-like and a CNOT-like gate. Gates with ISWAP-like behaviour move the quantum information around in addition to producing the necessary entanglement, making it non-obvious how to assemble the requisite stabilizer information for measurement.

Different hardware implementations generally provide different native entangling gates, but generally target a CNOT-like gate when aiming to perform error correction. In particular, superconducting qubit arrays traditionally target either a native CNOT such as via a cross-resonance gate [Par06; RD10], or a native CZ via a fixed or tunable capacitive coupling [Yan+18; Fox+20]. As shown in Figure 5, the surface code can be trivially adapted to use CZ gates, producing ZXXZ/XZZX stabilizers at measurement after canceling extraneous single qubit gates [Wen03; Bon+21; Goo+22]. Such superconducting hardware can also naturally produce ISWAP-like gate [Aru+19], but these have not been used in demonstrations of error correction due to a lack of appropriate circuit decomposition. Here we present a circuit for the surface code using ISWAP gates.

4.2 Circuit Construction

For simplicity, we first consider the construction using the CXSWAP gate rather than ISWAP. The CXSWAP gate is equal in action to a CX gate followed by a SWAP gate. It has the same KAK decomposition coefficients as the ISWAP gate, but has the conceptual advantage that its stabilizer flows each involve only Z or X terms and do not exchange them. Like circuits expressed in CNOT gates, the resulting detecting regions have only one Pauli type throughout, simplifying visualization. The circuit expressed in ISWAP gates differs only by single qubit gates, as described in Appendix A.5.

The complexity in this circuit construction arises because of the swapping behaviour of the entangling interaction. In the standard picture, assembling the stabilizer information to be measured is complicated by the movement of the relevant qubit states around the grid as the entangling layers are executed. By looking at detecting regions as a whole rather than focusing on moving qubit states, we can identify patterns of detectors that tile nicely and reproduce the relevant properties of the surface code.

Again, we start by considering the half-cycle state, as shown in Figure 10a. Rather than the next layer of entangling gates constructing a standard brickwork state, the swapping behaviours elongates the six body stabilizers. Figure 10b shows four of the detecting regions in this modified

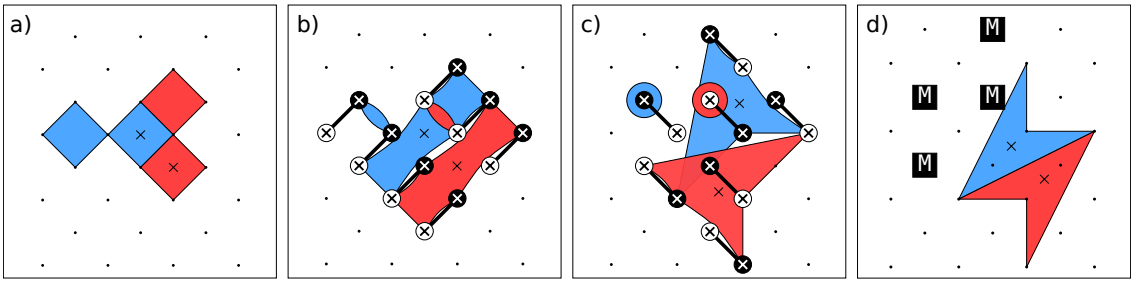


Figure 10: **Half-cycle picture for the CXSWAP circuit.** CXSWAP preserves detecting region type and is equivalent to ISWAP up to single qubit gates. The layer of CXSWAP gates immediately before each detecting region slice is shown. If the CXSWAP is decomposed into a CX followed by a SWAP, the black circle indicates the control qubit of the CX. a) Four detecting regions in half cycle state are shown, two of X and Z type, and two expanding (marked by an X) and two contracting (unmarked). b) The modified brickwork state using CXSWAP as the entangling layer. c) The modified state immediately prior to measurement, having contracted the two body stabilizers using CXSWAPs. d) The end-cycle state immediately after measurement, showing dart-shaped four-body stabilizers.

brickwork state; showing all the detecting region slices is visually complicated because they now overlap when drawn as detecting region slices. The next layer of entangling gates aims to contract the two body stabilizers to one body for subsequent measurement, as shown in Figure 10c, and produces an even more complex five-body stabilizer out of the previously extended six-body stabilizers. The subsequent measurement of the one-body stabilizer terms produces a relatively nice pattern of dart shaped four-body stabilizers. While this looks quite different to the normal surface code state at measurement, it displays the same overlapping structure of the stabilizers in the bulk, illustrating that errors on the unmeasured qubits will produce the same pattern of detection events as the standard code. We can then reverse these circuit steps (exchanging measurements for reset gates) to return to the half-cycle state. This cycle has the effect of contracting half of the detecting regions in that state and replacing them with new regions that expanded from the reset gates. The other half of the detecting regions expanded to cover the dart-shaped four-body stabilizers after measurement, and then returned to the half-cycle state. Similar to the hex-grid circuit, we use an analogous cycle to then measure those detecting regions, with the newly expanded detecting regions now expanding to cover the dart-shaped stabilizers at measurement. By alternating these two cycles, we implement the surface code.

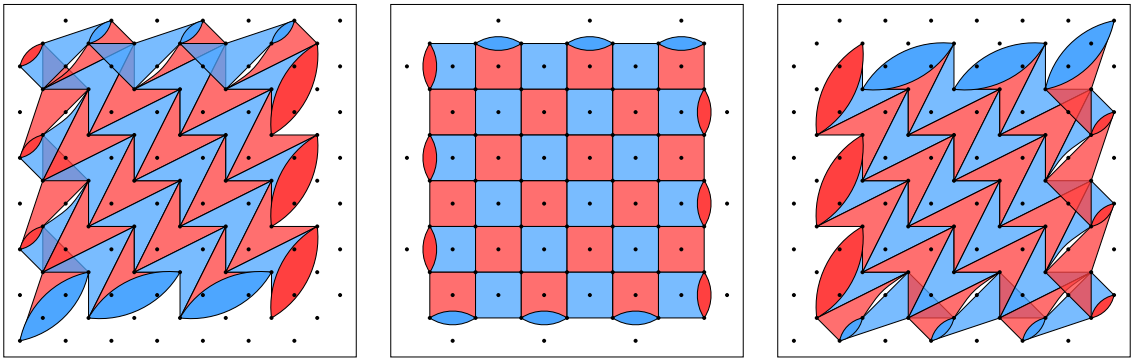


Figure 11: **Patch distortion in the CXSWAP/ISWAP circuit.** Center: The end-cycle state for the standard surface code circuit. Left, Right: The end-cycle states for the CXSWAP/ISWAP circuit, showing data qubits shifted in alternating directions along diagonal lines in the bulk. Corresponding measure qubits are shifted in the opposite direction to data qubits. Boundaries feature more complex distortions. The CXSWAP/ISWAP circuit alternates between two cycles, reaching both end-cycle states shown.

The pattern of stabilizers in the ISWAP circuit can also be understood as a distortion of the standard surface code patch. Figure 11 shows this distortion visually; each diagonal line of data qubits is shifted in alternating directions. Along the same lines, measure qubits are distorted in

the opposite direction, as they are exchanged with data qubits by the action of the ISWAP gates.

4.3 Boundaries

As with other constructions, the boundaries prove more complex than the bulk circuit. At the measurement layer, they correspond to the standard rotated surface code boundaries modified by the same distortion applied to stabilizers shown in Figure 11. The full circuit schedule for the CXSWAP circuit is shown in Figure 12. While all of the overlapping detecting regions in the bulk are visually complex, focusing on the detecting regions around the outer edges of the patch help to understand the boundaries. In particular, the mid-cycle states display helpful symmetry and a simple boundary shape consisting of terminating weight-1 detecting regions. Our process for finding the boundary construction was similar to that of the hex-grid circuit; starting with a guess at good detecting slices at the mid-cycle, we iteratively removed gates and manipulated the detecting region terminations while preserving the graph-like code distance, eventually arriving at the provided construct.

Slightly different boundaries are used in the ISWAP circuit we benchmarked, as included in the supplementary figures; two additional qubits were left in at the corners and measurements gates were not preferentially arranged to occur on the same qubit sub-grid (i.e. not on any neighbouring qubits simultaneously). Given both the CXSWAP and ISWAP circuits benchmark with essentially the same performance as their standard circuit counterpart, we believe the differences in boundaries does not significantly impact performance. That said, the boundary shown for the CXSWAP circuit is slightly simpler to understand visually and would likely perform better where hardware imposes costs on simultaneously measuring neighbouring qubits.

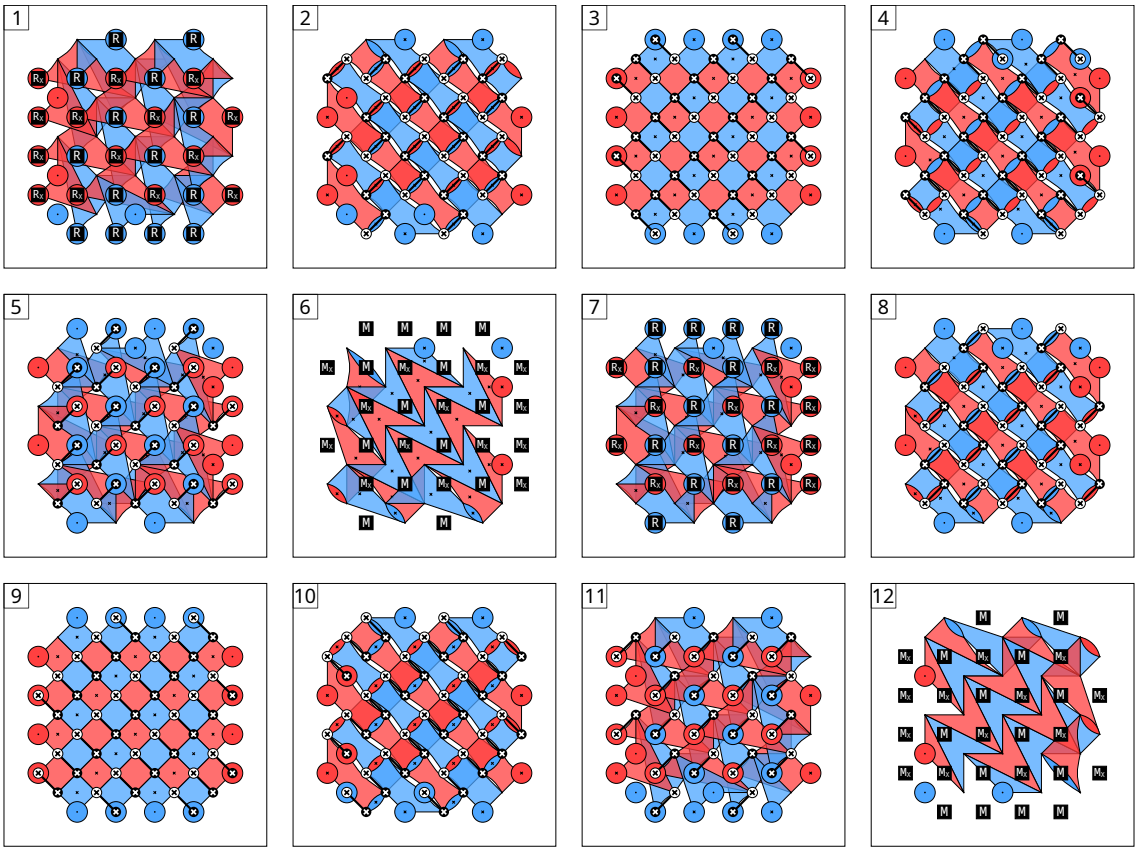


Figure 12: **Both cycles for the CXSWAP circuit.** Detecting region slices are shown after each layer of gates in the cycle. Regions introduced by the resets in panel 1 are marked with a black X. Detecting regions overlap in the bulk, as shown more simply in Figure 10, but tile nicely at the mid-cycle (3 and 9) and end-cycle (6 and 12) states. Red regions are X stabilizers, blue regions are Z stabilizers.

4.4 Benchmarking

We now numerically benchmark the circuit by simulating a quantum memory experiment using both the standard and ISWAP circuits. The full details of our benchmarking strategy, including the noise model, are discussed in Appendix D. In particular, we set the primary error component p in both cases to be the CZ or ISWAP gate for the standard and ISWAP circuits respectively.

We find that the teraquop footprint for the ISWAP circuit is essentially identical to the standard circuit. As before, such a qualitative statement is sensitive to the details of the error model. However, we expect that this circuit will be of interest for experimental implementations under the assumption that the ISWAP gate may provide a path to lower error rates than a CNOT or CZ gate, or provide other benefits not reflected in our simple error model.

4.5 Summary

The ISWAP circuit presents a compelling alternative circuit to target for hardware architecture with a natural ISWAP gate, alleviating the need to target a CNOT-like entangling gate for implementing the surface code. This construction completes the possible implementations of the surface code in terms of the 2Q CLifford KAK gate classes.

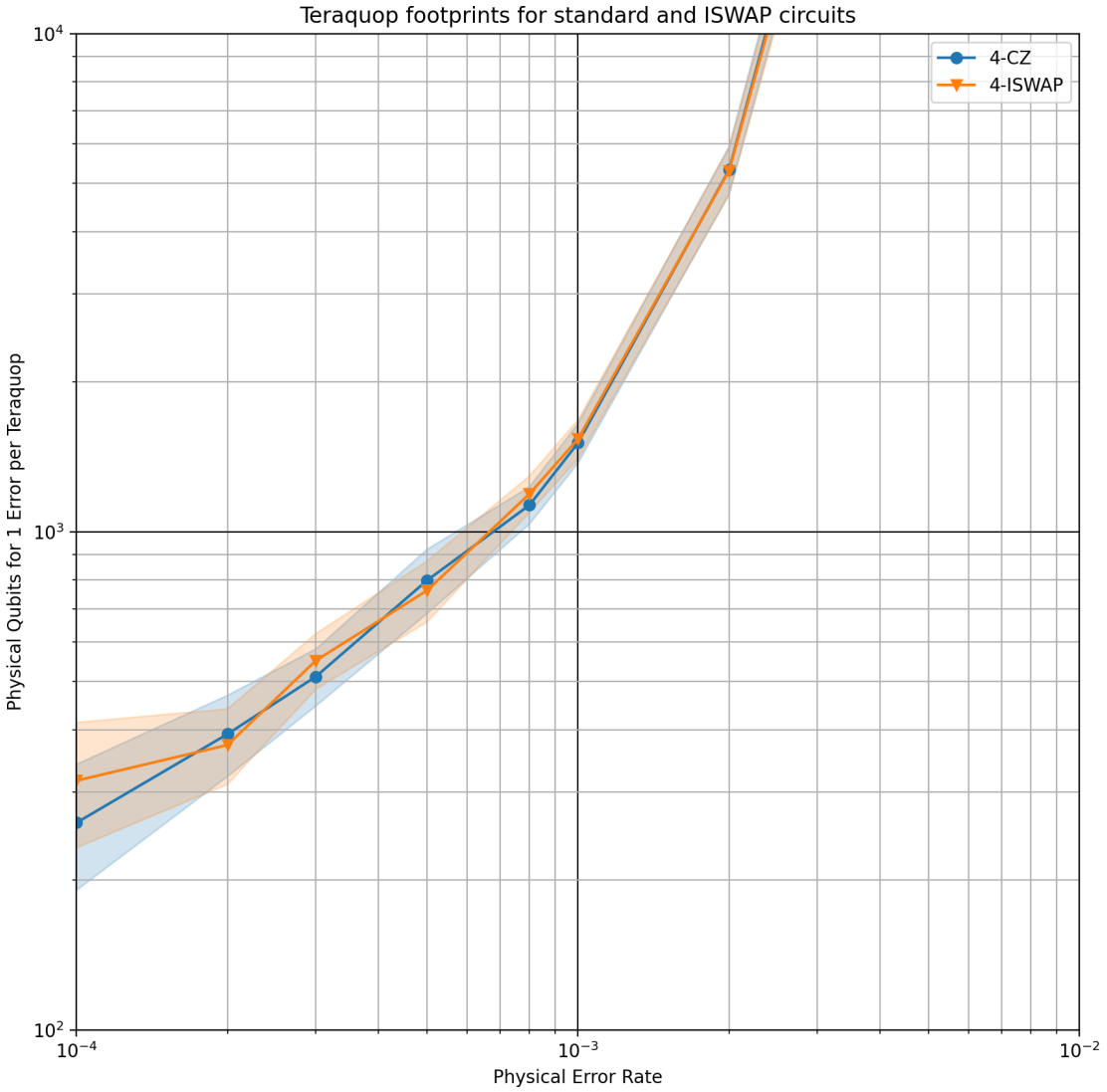


Figure 13: **Teraquop footprints for the ISWAP surface code circuit.** The number of physical qubits required for a single code patch to achieve a logical error rate of 1×10^{-12} over a $d \times d \times d$ space-time block. Each curve combines memory experiments in both Z and X basis, and uses the `SI1000` noise model as described in Appendix D. `4-CZ` (blue circles) is the standard surface code circuit compiled to CZ gates, single qubit rotations, and Z-type measurements and resets. `4-ISWAP` (orange triangles) is the ISWAP circuit, using ISWAP gates, single qubit rotations, and Z-type measurements and resets.

5 Walking Surface Code Circuits

Surface code circuits typically aim to provide a method of measuring static stabilizers, leaving any logical time dynamics to a higher level of code manipulation. Compiling logical algorithms in the surface code, such as by lattice surgery [Hor+12], typically manipulates the underlying circuit only subtractively, that is by deciding which stabilizers to simply not measure. In this view, the roles of the physical qubits in the circuit are fixed, and the only variable is whether the operations dictated by the template circuit are performed or not at any given location in space and time. With the freedom provided by additional detecting region shapes, it is possible to break the fixed allocation of qubit roles to physical qubits, and thereby break the fixed location of code stabilizers on the underlying qubit grid.

Here, we present various circuits for walking a surface code patch, moving it on the underlying qubit grid without increasing overhead in circuit layers or reducing the code distance. This has the effect of exchanging the roles of data and measure qubits, which is a desirable primitive operation in a code circuit for the purposes of mitigating leakage [Fow13a; GF15; BB19].

First, we discuss the relevance of qubit roles to the problem of correlated errors induced by leakage. We then introduce the walking circuit and its detecting structure, describe the moving patch boundaries, and numerically benchmark the code circuit. Finally, we discuss the implications for leakage mitigation in hardware experiments.

5.1 Leakage Mitigation in QEC Experiments

Most hardware implementations for quantum computing feature higher energy states nearby the chosen computational subspace, which can be erroneously populated in what is generally referred to as a *leakage error*. Leakage errors are especially problematic for quantum error correction, as the states they produce are typically long lived and not accounted for in the physical interactions used to implement gates [Fow13a]. Leakage errors generally induce a large number of equivalent uncorrelated Pauli errors, providing an out-sized contribution for the error budget in a quantum error correction experiment [Goo+22].

Various strategies are typically used to reduce the prevalence of leakage errors in gate implementations [Mot+09; Che+16; Fox+20], and in removing or suppressing the correlated effect of leakage errors when they do occur in the typical cycle circuit. Leakage is generally easier to remove from measure qubits, as immediately after measurement they are not holding any important quantum information and can be unconditionally reset [Mag+18; BVT21; McE+21; Zho+21].

Removing leakage from data qubits presents a more difficult challenge. Data qubit leakage can be removed directly by the use of gate operations that specifically affect the leakage state without disturbing the computational states [Mia+22], but such additional operations introduce new error sources to the error budget.

Alternatively, the roles of measure and data qubits can be regularly exchanged by the addition of SWAP or CNOT-like operations [Fow13a; GF15; BB19]. Assuming that entangling interactions do not move leakage between qubits, this permits leakage removal on each physical qubit after it is measured in every second cycle, curtailing the possible spread of leakage in time. However, this strategy also introduces additional operations necessary to exchange the qubit roles, impacting the final error budget. The circuit discussed here provides the ability to exchange qubit roles without adding any additional gates. While this strategy does not resolve leakage errors entirely, it provides a compelling alternative to known strategies for exchanging roles and could compliment direct removal strategies by alleviating some of the burden of leakage errors.

5.2 Circuit Construction

Figure 14 shows the standard circuit alongside the *stepping* circuit that exchanges measure and data qubits. The initial insight for constructing this circuit can be found in the half cycle state, illustrated in Figure 14c. In the standard surfaces code circuit, pairs of detecting regions begin centered around the same measure qubit, one expanding one covering just the measure qubit and one contracting one covering four data qubits and the measure qubit, as illustrated previously in

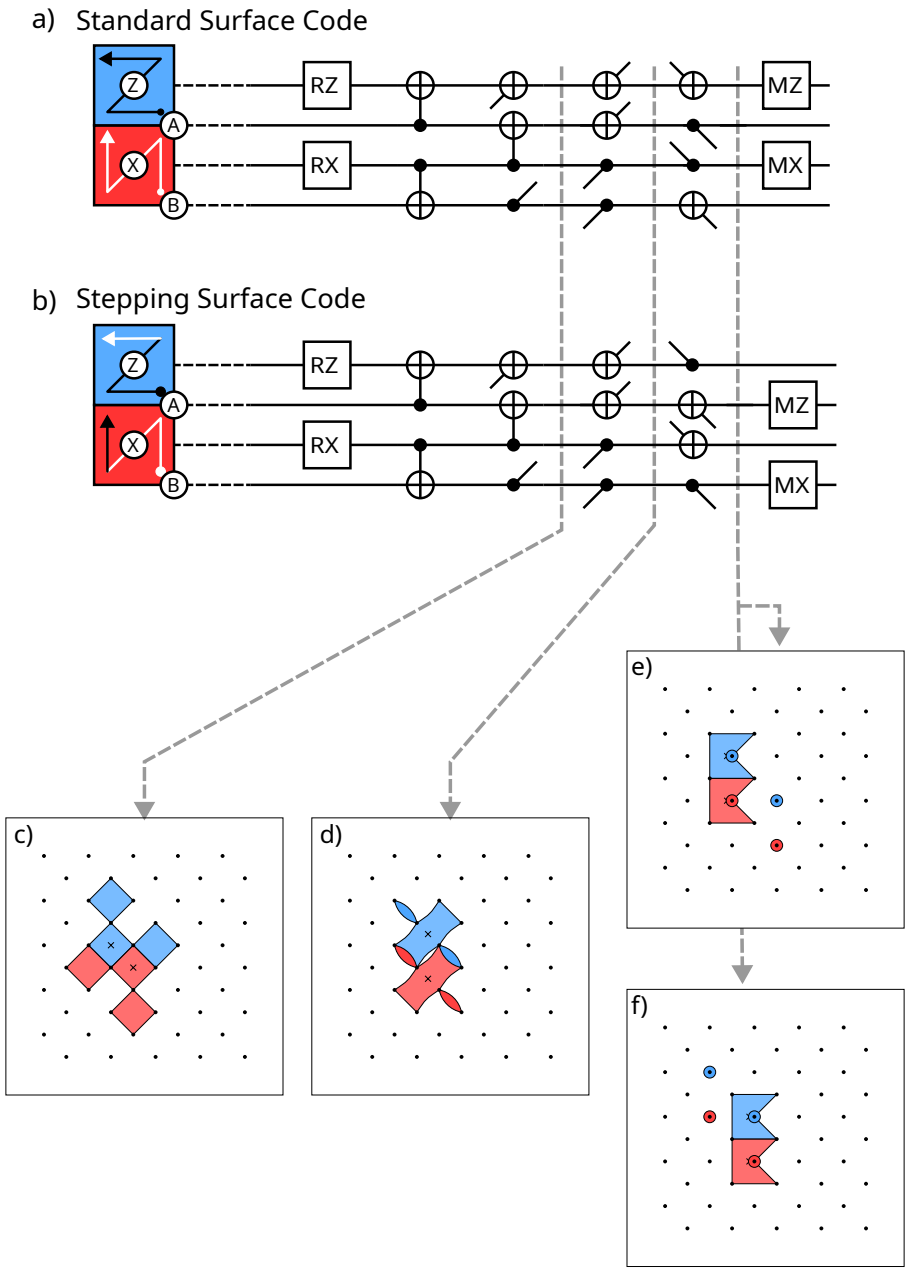


Figure 14: **Comparing the circuit and mid-cycle states for standard and stepping circuits.** a) The standard circuit for the surface code on two measure qubits (Z, X) and two data qubits (A,B). The measure qubits are reset, assemble stabilizer information from their four neighbouring measure qubits in the order indicated by the Z-shaped arrows, and are measured. b) The stepping circuit, which is identical to the standard circuit until the last layer of CNOT gates. This layer is instead the same as the first layer of CNOT gates with target and control reversed. At the measurement layer, the roles of qubits have been exchanged, with qubits that began the round as data qubits (A,B) now being measured. c) The half-cycle state for both circuits, showing six detecting regions. Two expanding regions are marked with an X. The expanding regions have a choice of whether to pair up with the upper-left or lower-right contracting regions of the same Pauli type. d) The second brickwork state for both circuits. The choice of pairing for the expanding region is now aligned along the diagonal axis. e) The flag-like state immediately before measurement for the standard circuit, corresponding to pairing the expanding region with the upper-left contracting region. f) The flag-like state for the stepping circuit, corresponding to pairing the expanding region with the lower-right contracting region.

Figure 4. By the half-cycle, these have both been mapped to four-body stabilizers involving their shared measure qubit.

The half-cycle state is highly symmetric, and this makes clear that maintaining this pairing and measuring out the contracting region on that measure qubit is only one possible option. Figure 14c shows a second possible pairing of expanding and contracting regions. Choosing to pair expanding and contracting regions around a different qubit choice has the effect that the contracting regions terminate on what was previously a data qubit, and the expanding regions cover stabilizers involving what were previously measure qubits. This exchanges the underlying qubit roles on the physical grid while preserving the logical structure of the code.

In fact, this different pairing requires only a very simple modification; changing the directions of the CNOT gates in the last layer, which we refer to as the *stepping trick*. That this trick is sufficient can be seen from the symmetry of the weight-2 detecting region slices immediately before the last layer of entangling gates. The direction of the last layer of CNOTs determines which of the two qubits the detecting region will contract down to. This trick is relatively generic, and can be easily applied to other circuits. In this section, we focus on applying the stepping trick to the standard circuit and emphasise the resulting ability to move a logical patch in arbitrary directions on the underlying grid. The same trick can be easily applied to the hex circuit to exchange qubit roles, as we discuss further in Section 6. The trick also applies to repetition codes, as we illustrate in Appendix B.

5.3 Boundaries

The boundaries prove more complex to construct than the bulk circuit. The walking circuit boundary must be chosen carefully to appropriately create new detecting regions on the *leading edges*, the two spatial boundaries in the direction of movement. We must also have detecting regions appropriately measured out on the *trailing edges*, the two spatial boundaries facing away from the direction of movement. In Figure 15, we provide a choice of boundaries that preserves the graph-like code distance, prioritizing not introducing any problematic hook errors.

One important detail we do not address visually here is the behaviour of the logical observable. As the code patch moves, the logical observable can be moved along with it by the inclusion of measurements at the trailing boundary, as expressed in the benchmarked circuits included in our data and code repository [MBG23].

We refer to the behaviour exhibited by one application of this circuit as taking a *step* and a single cycle as a *step cycle* to distinguish it from the standard cycle where the logical patch remains in place. The step cycle circuit can be trivially shifted in space to follow the patch, allowing further steps to be taken, and rotated in space to permit steps to be taken in the four cardinal directions relative to the physical qubit square grid.

Interestingly, compared to the standard circuit, the walking cycle circuit and its time inverse are more distinct. In the standard circuit, running the circuit in reverse is equivalent to a 180-degree spatial rotation of the same circuit; in terms of only the represented stabilizers, the two flag-like patterns are identical and the two brickwork patterns are equivalent up to rotation. This is not the case in the walking cycle, where all the patterns are noticeably different. The time-reversed cycle circuit, with measurements and resets exchanged, provides another pattern of boundaries which results in the patch taking a step (in the opposite direction to the time-forward circuit), and can similarly be shifted and rotated to permit steps to be taken in any direction. We only use the forward-going cycle shown in Figure 15 in the benchmarked circuits.

5.4 Behaviours for the Logical Patch

Given the capability of taking a step in any direction in each cycle, we can define additional behaviours for the logical qubit patch which can be helpful. Figure 16 shows four possible macroscopic behaviours which can be achieved using step cycle circuits, which we benchmark in the following section. The most basic behaviour, *wiggling*, involves simply taking steps back and forth on the spot, returning to the same position every second cycle. This presents a minimal overhead in terms of number of qubits to achieve the swapping of qubit roles in the bulk. Continuing to take additional steps in the same direction continues to move the patch through the underlying qubit

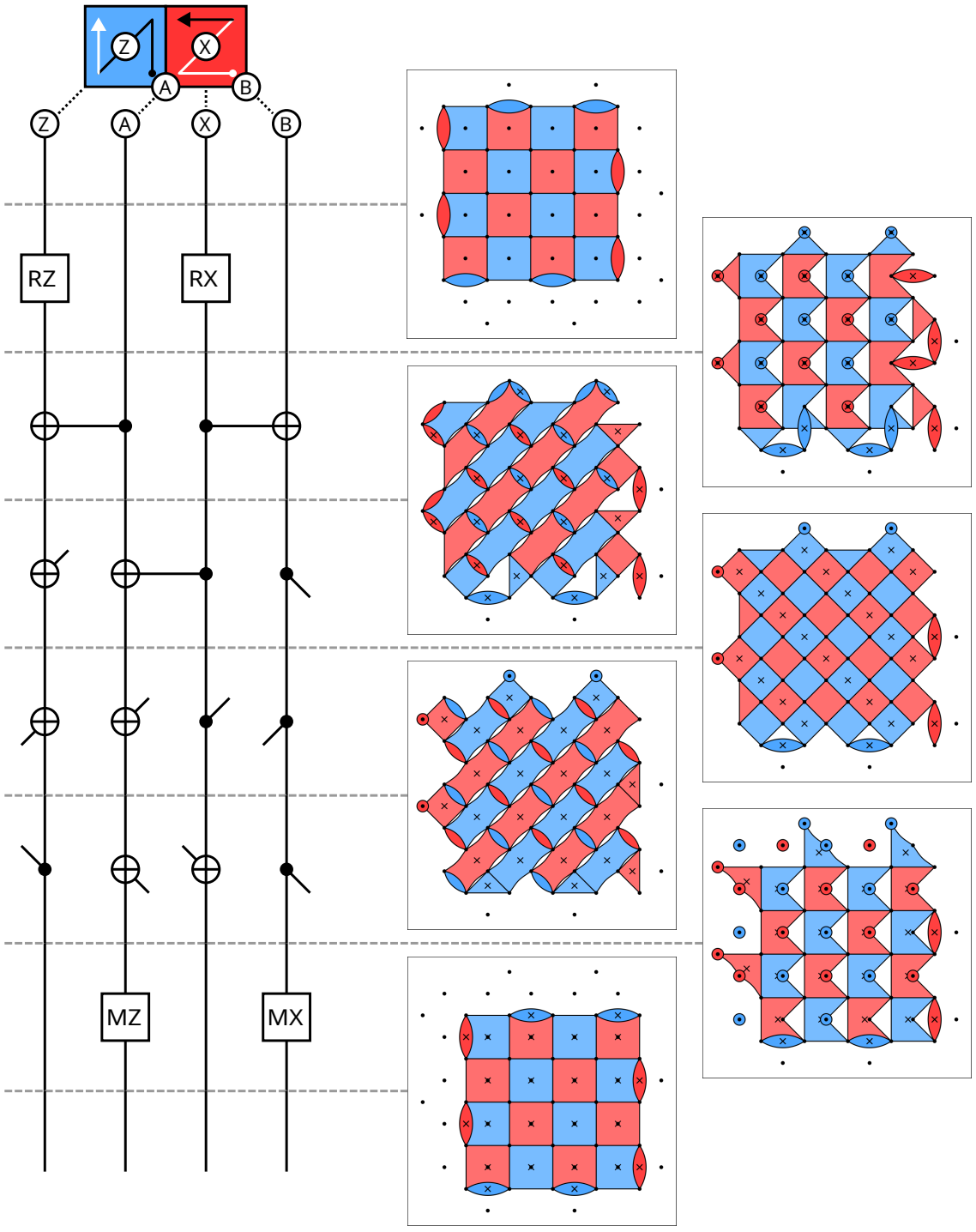


Figure 15: **Mid-Cycle States in the Walking Circuit.** Left: The circuit for the walking surface code, shown on two measure qubits (X and Z) and two data qubits (A and B). Time proceeds down the page. Right: The detecting region slices at each layer of the circuit. Each shape represents a single stabilizer term consisting of either Z (blue) or X (red) Pauli elements. The corners of the shape indicate the qubits included in the term, with circles indicating single qubit terms. The expanding detecting regions introduced in this cycle are marked with an X.

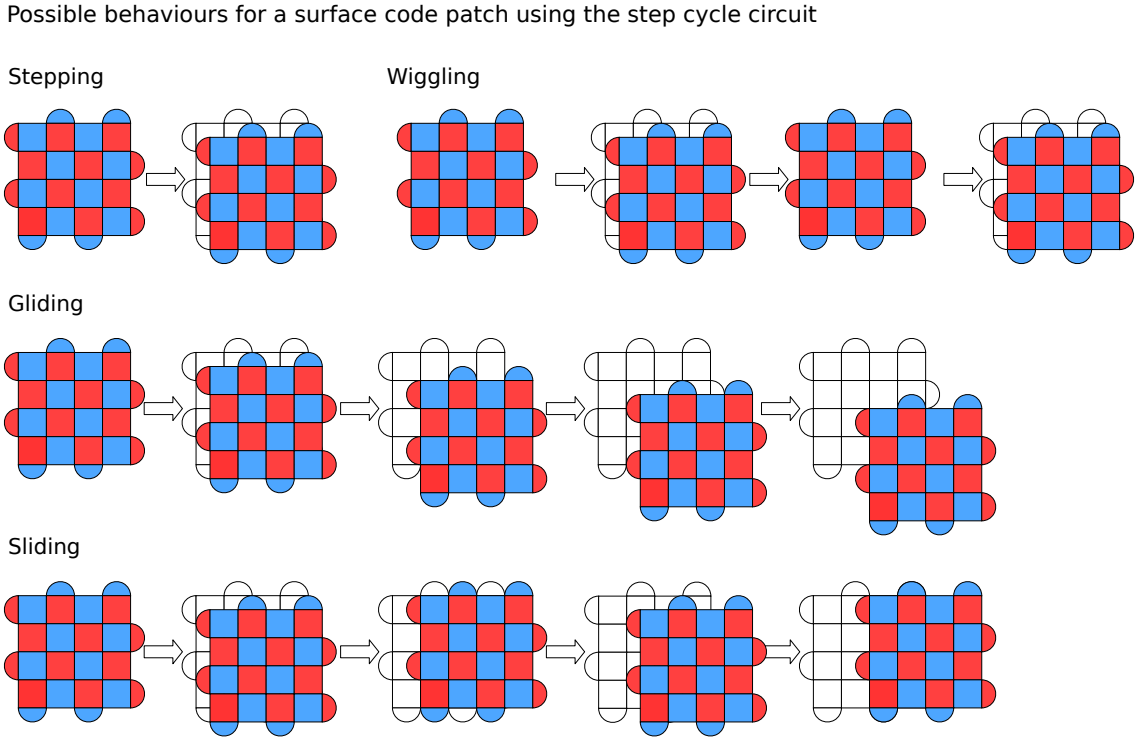


Figure 16: **Behaviours for the logical patch using step cycle circuits.** Given the capability of taking a single step (Stepping), we can chain adjacent cycles of steps to produce additional behaviours. These include the patch stepping back and forth in place, returning to its starting position every second cycle (Wiggling), continuing to step in the same direction (Gliding) or taking alternating perpendicular steps to move the patch laterally (Sliding).

grid, which we call *gliding*. Another specific behaviour more compatible with current layouts for logical algorithms is *sliding*, where the patch takes alternating steps in perpendicular directions, allowing patches to move laterally. This could be especially convenient when considering routing problems, especially when all other qubits are engaging in wiggling behaviour. An application of this behaviour to aiding logical compilation is discussed in Appendix C.

5.5 Benchmarking

We now numerically benchmark the circuit by simulating a quantum memory experiment using stepping circuits in each of the three behaviours presented in Figure 16. As before, the full details of our benchmarking strategy, including the noise model, are discussed in Appendix D.

The teraquop footprints for all four behaviours are qualitatively very similar, with footprints between 1000 and 3000 qubits at an aspirational limiting error rate of 1×10^{-3}. The standard surface code displays slightly better performance than the walking codes, which we attribute to the additional qubits that the step cycle moves into as it steps, resulting in a larger number of qubits used for the same distance and error suppression. We also notice that the wiggling behaviour displays slightly better performance than the gliding and sliding behaviours, which we attribute to changes in the number of long error chains in the gliding and sliding behaviours through time. Overall though, the relative performance of these codes is subject to our assumed error model, and we would expect an error model that included leakage to affect the relative performance of these codes substantially.

5.6 Summary

In summary, we have presented a circuit that achieves the exchange of roles of measure and data qubits without additional gate overhead. Additionally, this circuit moves the logical patch on

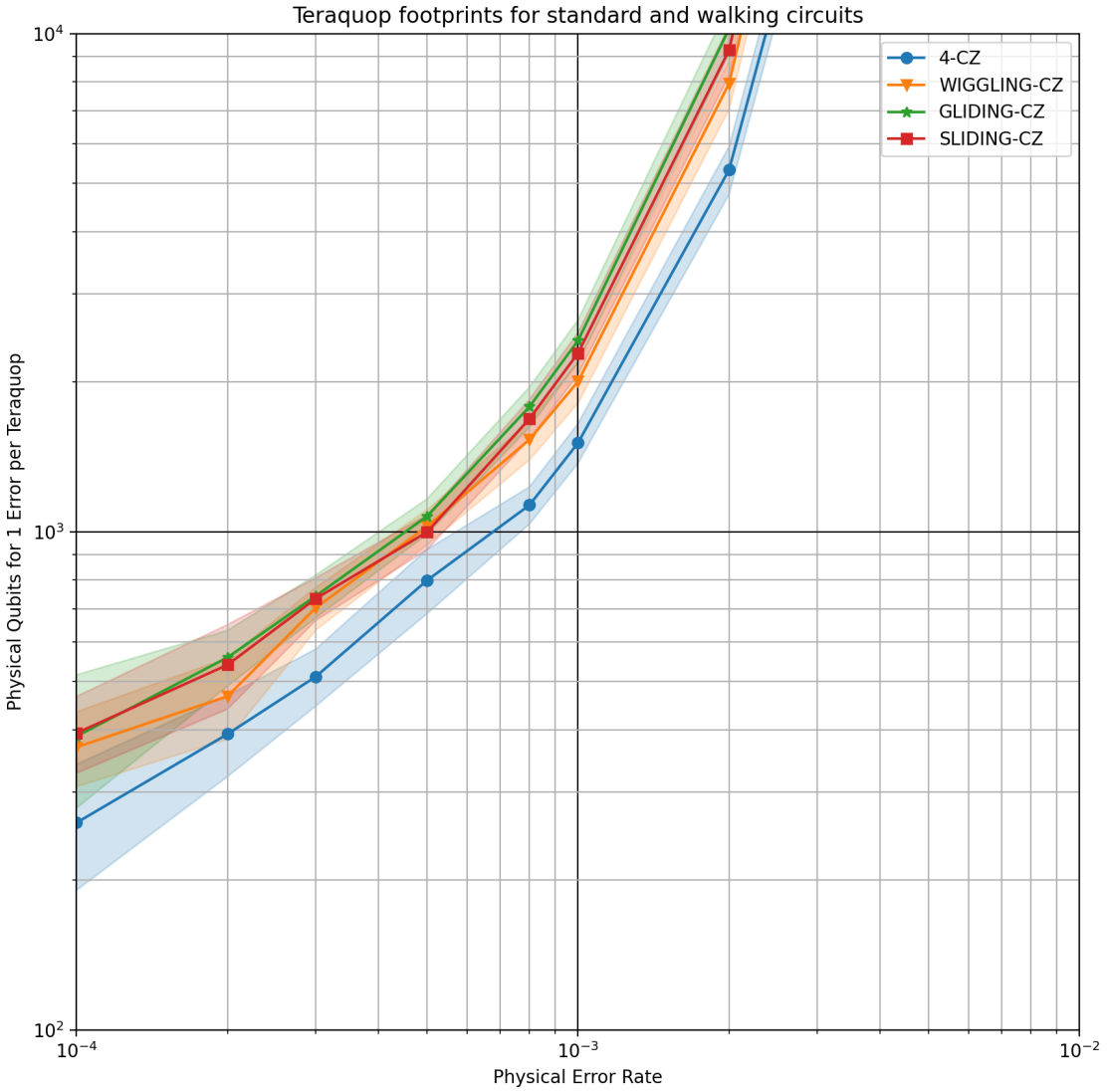


Figure 17: **Teraquop footprints for the walking surface code circuits.** The number of physical qubits required for a single code patch to achieve a logical error rate of 1×10^{-12} over a $d \times d \times d$ space-time block. Each curve combines memory experiments in both Z and X basis, and uses the `SI1000` noise model as described in Appendix D. `4-CZ` (blue circles) is the standard surface code circuit compiled to CZ gates, single qubit rotations, and Z-type measurements and resets. `WIGGLING-CZ` (orange triangles), `GLIDING-CZ` (green stars), and `SLIDING-CZ` (red squares) are walking circuits for wiggling, gliding and sliding behaviours respectively.

the underlying physical qubit grid, opening new possible behaviours for the logical patch. In the absence of leakage errors, the circuit performs comparably to the traditional surface code circuit. We expect when considering leakage errors that this circuit will display additional advantages, given the exchange of roles coupled with existing progress on removing leakage from measure qubits. Simulating this circuit in the presence of leakage and quantifying the advantage is important future work. We expect that this technique will compliment current work on direct removal of leakage on data qubits, rather than solve the presence of leakage outright. The low additional overhead to implement this technique in terms of error rates makes it a compelling addition to practical error correction experiments. In addition to the relevance of walking to leakage removal, it also has implications for higher-level logical compilation, which we discuss in Appendix C.

6 Conclusion and Outlook

The three circuit constructions we have presented in detail represent three applications of this approach to QEC circuit construction. In this section, we address combining these connectivity, gate and behaviour benefits, address some additional related constructions, and provide some outlook for these techniques to be applied more broadly.

6.1 Further Constructions

The presented circuits provide distinct benefits from the view of simplifying the required hardware, but these are not mutually exclusive. We can combine the constructions above to generate circuits for each combination of benefits, including a circuit that embeds on a hex grid, uses ISWAP gates, and exchanges data and measure qubits between neighbouring cycles. While we do not address these combined circuits in the same level of detail, we do provide the exact circuit and benchmarking results.

Table 1 presents each of the constructions that we have benchmarked. We have included summary benchmarking for each circuit in Appendix E, and detailed benchmarking as supplementary figures available as an ancillary file and in our data repository [MBG23].

In addition to the circuits that we presented in detail and their combinations, we include some additional related circuits which we briefly discuss here. We present these additional circuits in the toric case without boundary constructions, but hope that finding appropriate boundaries will not prove especially difficult in light of the examples and methods discussed for the main constructions.

While we explicitly addressed embedding on a hex grid, cutting edge hardware architectures have targeted the related *heavy-hex* grid, where an additional qubit is placed on each edge of the hex grid, reducing the average connectivity below three. Using similar techniques to the hex grid circuits, we built a circuit for the surface code on the heavy-hex grid. This circuit uses six layers of entangling gates per measurement cycle. (The number of layers of entangling gates can be reduced from six to four by repurposing flag qubits for teleportation, but we assumed this would make performance worse and so did not benchmark that variant of the circuit for this paper.) Our heavy-hex circuit outperforms the heavy-hex code [Cha+20; Sun+22], and also the heavy-square code [Cha+20].

We also include a circuit for the *semi-heavy-hex* grid, where additional qubits are included on only a third of the hex-grid edges, which we use as measure qubits. This grid represents a middle ground between hex and heavy-hex grids, where the average connectivity is still lower than three, and qubits requiring measurement during the cycle have lower connectivity than the remaining qubits. This circuit requires only three layers of entangling gates per measurement cycle, taking advantage of the additional qubits to increase parallelization when compared to the hex-grid circuit.

Finally, we present additional circuits considering more unusual underlying qubit hardware, specifically hybrid circuits using a hardware native parity measurement on a third of the edges in a hex grid. This represents a middle ground between an architecture targeting the hex-grid circuit and one targeting only pair measurements [HH21; Gid22]. Each qubit in the hybrid circuit must participate in only one pair measurement and two standard entangling gates rather than three pair measurements, possibly simplifying the implementation of the architecture in hardware. Hardware implementing pair measurements has continued to improve [LGB10; DS12; RPB18;

Circuit Type	Circuit Label	Grid	Entangling Gate	Qubits Exchange Roles
Standard	`4-CX`	Square	CX	No
	`4-CZ`	Square	CZ	No
Hex-grid (Section 3)	`3-CX`	Hex	CX	No
	`3-CZ`	Hex	CZ	No
	`3-CX-wiggle`	Hex	CX	Yes
	`3-CZ-wiggle`	Hex	CZ	Yes
ISWAP (Section 4)	`4-CXSWAP`	Square	CXSWAP	No
	`4-ISWAP`	Square	ISWAP	No
	`3-CXSWAP`	Hex	CXSWAP	No
	`3-ISWAP`	Hex	ISWAP	No
	`3-CXSWAP-wiggle`	Hex	CXSWAP	Yes
	`3-ISWAP-wiggle`	Hex	ISWAP	Yes
Walking (Section 5)	Capable of arbitrary movement direction			
Wiggling	`WIGGLING-CX`	Square	CX	Yes
	`WIGGLING-CZ`	Square	CZ	Yes
Gliding	`GLIDING-CX`	Square	CX	Yes
	`GLIDING-CZ`	Square	CZ	Yes
Sliding	`SLIDING-CX`	Square	CX	Yes
	`SLIDING-CZ`	Square	CZ	Yes
Toric	Presented without boundary constructions			
CSS Toric Code	`TORIC-4-CX`	Square	CX	No
Heavy-hex	`TORIC-3_HEAVY-CX`	Heavy-hex	CX	No
Semi-heavy-hex	`TORIC-3_SEMI_HEAVY-CX`	Semi-heavy-hex	CX	No
Parity Assisted	1/3 of edges do parity measurements instead of unitary interactions			
	`TORIC-3-CX_MXX_MZZ`	Modified Hex	CX, MXX, MZZ	N/A
	`3-CX_MXX_MZZ`	Modified Hex	CX, MXX, MZZ	N/A
	`3-CZ_MZZ`	Modified Hex	CZ, MZZ	N/A

Table 1: **All circuit constructions.** An exhaustive list of circuit constructions included in this work. Benchmarking for all circuits is included in Appendix E.

Rea+22; Liv+22], and this circuit provides an opportunity to use such hardware for the surface code. Under our assumed error model, which gives the same two-qubit depolarizing strength to two qubits gates and pair measurements, these circuits display better performance than equivalent circuits using only two qubit gates. However, we expect details of the true error model in hardware to determine their relative performance in practice. We include them as inspiration for alternative hardware architectures to target the surface code without additional overhead.

6.2 Open Problems

These constructions collectively represent only the first steps toward tailoring QEC circuit decompositions to relax or improve hardware requirements. Here, we discuss some directions for future research we hope are encouraged by the results presented here.

First, the surface code is only one error correction code. We chose to focus on it for its relevance to current experimental implementations of error correction. Applying these ideas to other codes is important open work, particularly to the color code, dynamical codes such as the honeycomb code, and to codes designed with biased noise in mind.

Second, the spatial compatibility of these code circuits with each other and with other standard circuits warrants further exploration. In particular, finding circuits for implementing sub-system codes that are compatible with these code circuits will be necessary for future implementation of these circuits on large grids of imperfect qubits.

Third, further refining the simple boundaries we have presented may be necessary to optimize performance and obey additional hardware constraints. In the context of logical algorithms, some boundary constructions may pack better than others, and behaviours like sliding may be useful in

reducing costs. Other boundary constructions may be useful to respect constraints on which qubits may be measured simultaneously, even with an additional overhead in qubit number or packing ability.

Finally, the benchmarking we presented here is deficient in that it fails to account for the primary ways we expect our circuit constructions to do better than the standard circuit. Benchmarking them with more realistic noise models, especially those including hardware accurate gate errors, leakage, crosstalk and other effects of relaxing requirements on the underlying device architecture is an important next step for qualifying the advantages made possible by these circuits.

6.3 Outlook

In this work, we have introduced an approach to constructing quantum error correction circuits more directly than via considering static stabilizer codes and only then forming their circuit decomposition. This work complements a wider thrust toward considering the full space-time dynamics of quantum error correction circuits [Got22].

We emphasise the concept of detecting regions as a helpful basis for considering stabilizers at all points during QEC circuits. We used the insights produced by slicing detecting regions to generate new circuits for implementing QEC codes. These provide new freedoms to hardware engineers designing devices to implement QEC in experiments. We hope that these initial constructions serve as certificates that this approach is worth pursuing further.

6.3.1 A note on recent progress in the space-time approach

Following this work first appearing online, two additional works were published presenting progress in the space-time approach to fault-tolerance. We would like to draw attention to these works specifically, as we feel they compliments the concepts discussed in this work, and hope to encourage future developments along these lines.

First, [Bom+23] presents a lower-level approach to topological stabilizer fault tolerance, demonstrating that fault-tolerance in circuit-based, measurement-based, fusion-based and Floquet-based quantum computing can be understood as implementations of the same underlying fault-tolerant structure. In contrast to our work, where we restricted ourselves to the language of circuits, this work uses the language of the ZX calculus. In particular, the *Pauli web* of a check in a ZX-graph is the direct analogue of a detecting region in a circuit. We chose to focus on the circuit picture for its applicability to hardware implementation, in particular being able to directly benchmark the circuit performance under hardware noise models. For those free of such considerations, we highly recommend the language of the ZX calculus and this work.

Second, [DP23] presents a generalised scheme for correcting faults in Clifford circuits by constructing an appropriate *spacetime code* over that circuit. In contrast to our work, where we consider only optimising the surface code circuit, this work considers the more general problem in arbitrary Clifford circuits and of decoding the resulting code. As the authors note, the stabilizer terms of the spacetime code correspond to detectors in our work. We appreciate and endorse the authors suggestion that their approach could push the circuit-centric approach further, especially in searching for promising new quantum codes and circuit constructions.

We feel these works serve as further evidence that considering the spacetime structure of circuits is a powerful new paradigm for approaching fault-tolerance. Moving beyond stabilizer codes reveals new connections between different fault-tolerance strategies, opens new avenues for exploration, and gives us hope for future improvements in the best strategies for implementing useful quantum computation in practice.

Contributions

Matt McEwen and Craig Gidney conceived of using the detecting region picture to produce new circuits for the surface code. Craig Gidney developed software tools, including Stim, that were vital in exploring and developing these concepts and constructions. Matt McEwen conceived and built the initial wiggling and walking surface code circuits. Dave Bacon and Craig Gidney conceived and built the initial ISWAP surface code circuits. Craig Gidney conceived and built the initial

hex-grid surface code circuits. All authors participated in synthesizing these ideas into a coherent narrative centered around overlapping structures of detecting regions. Matt McEwen compiled the manuscript.

Acknowledgements

We thank Michael Newman and Cody Jones for valuable discussions around these concepts as they were being developed. We thank Austin Fowler for writing the internal software tool we used for syndrome decoding, and for helpful review of the manuscript. We thank Alexis Morvan and Oscar Higgot for helpful review of the manuscript. We thank Alex Townsend-Teague for pointing out an error in the walking circuit figures. Finally, we would like to thank the entire Google Quantum AI team for making an environment where this work is possible.

Availability of data and code

The circuits that were benchmarked and the resulting raw samples are available on Zenodo at `https://zenodo.org/record/7587578` [MBG23]. The supplementary figures, along with the latex source and assets for this manuscript, are also available in the data repository and with the ArXiv submission. Code used to produce the circuits and to run the benchmarking is available on GitHub at `https://github.com/Strilanc/midout`. Stim, including new tools for visualizing circuits and stabilizers, is available on GitHub at `https://github.com/quantumlib/Stim` [Gid21].

References

[Aar22] Scott Aaronson. “Introduction to Quantum Information Science II Lecture Notes”. In: (2022).

[AG04] Scott Aaronson and Daniel Gottesman. “Improved simulation of stabilizer circuits”. In: *Physical Review A* 70.5 (Nov. 30, 2004), p. 052328. ISSN: 1050-2947, 1094-1622. DOI: 10.1103/PhysRevA.70.052328.

[Aru+19] Frank Arute et al. “Quantum supremacy using a programmable superconducting processor”. In: *Nature* 574.7779 (Oct. 24, 2019). Publisher: Nature Publishing Group, pp. 505–510. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/s41586-019-1666-5.

[AWH22] David Aasen, Zhenghan Wang, and Matthew B. Hastings. “Adiabatic paths of Hamiltonians, symmetries of topological order, and automorphism codes”. In: (2022). Publisher: arXiv Version Number: 2. DOI: 10.48550/ARXIV.2203.11137.

[Bac05] Dave Bacon. “Operator Quantum Error Correcting Subsystems for Self-Correcting Quantum Memories”. In: (2005). Publisher: arXiv Version Number: 4. DOI: 10.48550/ARXIV.QUANT-PH/0506023.

[BB19] Natalie C. Brown and Kenneth R. Brown. “Leakage mitigation for quantum error correction using a mixed qubit scheme”. In: *Physical Review A* 100.3 (Sept. 18, 2019). Publisher: American Physical Society, p. 032325. ISSN: 2469-9926, 2469-9934. DOI: 10.1103/PhysRevA.100.032325.

[BE20] Nikolas P. Breuckmann and Jens N. Eberhardt. “Balanced Product Quantum Codes”. In: (2020). Publisher: arXiv Version Number: 3. DOI: 10.48550/ARXIV.2012.09271.

[BE21] Nikolas P. Breuckmann and Jens Niklas Eberhardt. “Quantum Low-Density Parity-Check Codes”. In: *PRX Quantum* 2.4 (Oct. 11, 2021), p. 040101. ISSN: 2691-3399. DOI: 10.1103/PRXQuantum.2.040101.

[BK05] Sergey Bravyi and Alexei Kitaev. “Universal quantum computation with ideal Clifford gates and noisy ancillas”. In: *Physical Review A* 71.2 (Feb. 22, 2005), p. 022316. ISSN: 1050-2947, 1094-1622. DOI: 10.1103/PhysRevA.71.022316.

[BK22] Nouédyn Baspin and Anirudh Krishna. “Connectivity constrains quantum codes”. In: *Quantum* 6 (May 13, 2022), p. 711. ISSN: 2521-327X. DOI: 10.22331/q-2022-05-13-711.

[BK98] S. B. Bravyi and A. Yu. Kitaev. “Quantum codes on a lattice with boundary”. In: (1998). Publisher: arXiv Version Number: 1. DOI: 10.48550/ARXIV.QUANT-PH/9811052.

[BM07] H. Bombin and M. A. Martin-Delgado. “Optimal resources for topological two-dimensional stabilizer codes: Comparative study”. In: *Physical Review A* 76.1 (July 6, 2007), p. 012305. ISSN: 1050-2947, 1094-1622. DOI: 10.1103/PhysRevA.76.012305.

[Bom+21] Hector Bombin et al. “Logical blocks for fault-tolerant topological quantum computation”. In: (2021). Publisher: arXiv Version Number: 1. DOI: 10.48550/ARXIV.2112.12160.

[Bom+23] Hector Bombin, Daniel Litinski, Naomi Nickerson, Fernando Pastawski, and Sam Roberts. “Unifying flavors of fault tolerance with the ZX calculus”. In: (2023). Publisher: arXiv Version Number: 1. DOI: 10.48550/ARXIV.2303.08829.

[Bom15] Héctor Bombín. “Single-Shot Fault-Tolerant Quantum Error Correction”. In: *Physical Review X* 5.3 (Sept. 28, 2015), p. 031043. ISSN: 2160-3308. DOI: 10.1103/PhysRevX.5.031043.

[Bon+21] J. Pablo Bonilla Ataides, David K. Tuckett, Stephen D. Bartlett, Steven T. Flammia, and Benjamin J. Brown. “The XZZX surface code”. In: *Nature Communications* 12.1 (Dec. 2021), p. 2172. ISSN: 2041-1723. DOI: 10.1038/s41467-021-22274-1.

[Bra+12] Sergey Bravyi, Guillaume Duclos-Cianci, David Poulin, and Martin Suchara. “Subsystem surface codes with three-qubit check operators”. In: (2012). Publisher: arXiv Version Number: 2. DOI: 10.48550/ARXIV.1207.1443.

[BVT21] F. Battistel, B.M. Varbanov, and B.M. Terhal. “Hardware-Efficient Leakage-Reduction Scheme for Quantum Error Correction with Superconducting Transmon Qubits”. In: *PRX Quantum* 2.3 (July 26, 2021), p. 030314. ISSN: 2691-3399. DOI: 10.1103/PRXQuantum.2.030314.

[CC19] Christopher Chamberland and Andrew W. Cross. “Fault-tolerant magic state preparation with flag qubits”. In: *Quantum* 3 (May 20, 2019), p. 143. ISSN: 2521-327X. DOI: 10.22331/q-2019-05-20-143.

[Cha+20] Christopher Chamberland, Guanyu Zhu, Theodore J. Yoder, Jared B. Hertzberg, and Andrew W. Cross. “Topological and Subsystem Codes on Low-Degree Graphs with Flag Qubits”. In: *Physical Review X* 10.1 (Jan. 31, 2020), p. 011022. ISSN: 2160-3308. DOI: 10.1103/PhysRevX.10.011022.

[Che+16] Zijun Chen et al. “Measuring and Suppressing Quantum State Leakage in a Superconducting Qubit”. In: *Physical Review Letters* 116.2 (Jan. 13, 2016). Publisher: American Physical Society, p. 020501. ISSN: 0031-9007, 1079-7114. DOI: 10.1103/PhysRevLett.116.020501.

[Cór+13] A. D. Córcoles et al. “Process verification of two-qubit quantum gates by randomized benchmarking”. In: *Physical Review A* 87.3 (Mar. 19, 2013), p. 030301. ISSN: 1050-2947, 1094-1622. DOI: 10.1103/PhysRevA.87.030301.

[CR18] Rui Chao and Ben W. Reichardt. “Quantum Error Correction with Only Two Extra Qubits”. In: *Physical Review Letters* 121.5 (Aug. 1, 2018), p. 050502. ISSN: 0031-9007, 1079-7114. DOI: 10.1103/PhysRevLett.121.050502.

[CR20] Rui Chao and Ben W. Reichardt. “Flag Fault-Tolerant Error Correction for any Stabilizer Code”. In: *PRX Quantum* 1.1 (Sept. 3, 2020), p. 010302. ISSN: 2691-3399. DOI: 10.1103/PRXQuantum.1.010302.

[CS96] A. R. Calderbank and Peter W. Shor. “Good quantum error-correcting codes exist”. In: *Physical Review A* 54.2 (Aug. 1, 1996), pp. 1098–1105. ISSN: 1050-2947, 1094-1622. DOI: 10.1103/PhysRevA.54.1098.

[Den+02] Eric Dennis, Alexei Kitaev, Andrew Landahl, and John Preskill. “Topological quantum memory”. In: *Journal of Mathematical Physics* 43.9 (Sept. 2002), pp. 4452–4505. ISSN: 0022-2488, 1089-7658. DOI: 10.1063/1.1499754.

[DP23] Nicolas Delfosse and Adam Paetznick. “Spacetime codes of Clifford circuits”. In: (2023). Publisher: arXiv Version Number: 1. DOI: 10.48550/ARXIV.2304.05943.

[DS12] David P. DiVincenzo and Firat Solgun. “Multi-qubit parity measurement in circuit quantum electrodynamics”. In: (2012). Publisher: arXiv Version Number: 2. DOI: 10.48550/ARXIV.1205.1910.

[Fow+12] Austin G. Fowler, Matteo Mariantoni, John M. Martinis, and Andrew N. Cleland. “Surface codes: Towards practical large-scale quantum computation”. In: *Physical Review A* 86.3 (Sept. 18, 2012). Publisher: American Physical Society, p. 032324. ISSN: 1050-2947, 1094-1622. DOI: 10.1103/physreva.86.032324.

[Fow13a] Austin G. Fowler. “Coping with qubit leakage in topological codes”. In: *Physical Review A* 88.4 (Oct. 8, 2013). Publisher: American Physical Society, p. 042308. ISSN: 1050-2947, 1094-1622. DOI: 10.1103/PhysRevA.88.042308.

[Fow13b] Austin G. Fowler. “Optimal complexity correction of correlated errors in the surface code”. In: (2013). Publisher: arXiv Version Number: 1. DOI: 10.48550/ARXIV.1310.0863.

[Fox+20] B. Foxen et al. “Demonstrating a Continuous Set of Two-qubit Gates for Near-term Quantum Algorithms”. In: *Physical Review Letters* 125.12 (Sept. 15, 2020). _eprint: 2001.08343, p. 120504. ISSN: 0031-9007, 1079-7114. DOI: https://doi.org/10.1103/PhysRevLett.125.120504.

[Fuj14] Yuichiro Fujiwara. “Ability of stabilizer quantum error correction to protect itself from its own imperfection”. In: *Physical Review A* 90.6 (Dec. 1, 2014), p. 062304. ISSN: 1050-2947, 1094-1622. DOI: 10.1103/PhysRevA.90.062304.

[GE21] Craig Gidney and Martin Ekerå. “How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits”. In: *Quantum* 5 (Apr. 15, 2021). Publisher: Verein zur Förderung des Open Access Publizierens in den Quantenwissenschaften, p. 433. ISSN: 2521-327X. DOI: 10.22331/q-2021-04-15-433.

[GF15] Joydip Ghosh and Austin G. Fowler. “Leakage-resilient approach to fault-tolerant quantum computing with superconducting elements”. In: *Physical Review A* 91.2 (Feb. 20, 2015), p. 020302. ISSN: 1050-2947, 1094-1622. DOI: 10.1103/PhysRevA.91.020302.

[Gho+13] Joydip Ghosh, Austin G. Fowler, John M. Martinis, and Michael R. Geller. “Understanding the effects of leakage in superconducting quantum-error-detection circuits”. In: *Physical Review A* 88.6 (Dec. 23, 2013), p. 062329. ISSN: 1050-2947, 1094-1622. DOI: 10.1103/PhysRevA.88.062329.

[Gid21] Craig Gidney. “Stim: a fast stabilizer circuit simulator”. In: *Quantum* 5 (July 6, 2021), p. 497. ISSN: 2521-327X. DOI: 10.22331/q-2021-07-06-497.

[Gid22] Craig Gidney. *A Pair Measurement Surface Code on Pentagons*. 2022. DOI: 10.48550/ARXIV.2206.12780.

[GNM22] Craig Gidney, Michael Newman, and Matt McEwen. “Benchmarking the Planar Honeycomb Code”. In: *Quantum* 6 (Sept. 21, 2022), p. 813. ISSN: 2521-327X. DOI: 10.22331/q-2022-09-21-813.

[Goo+21] Google Quantum AI et al. “Exponential suppression of bit or phase errors with cyclic error correction”. In: *Nature* 595.7867 (July 15, 2021). _eprint: 2102.06132, pp. 383–387. ISSN: 0028-0836, 1476-4687. DOI: https://doi.org/10.1038/s41586-021-03588-y.

[Goo+22] Google Quantum AI et al. “Suppressing quantum errors by scaling a surface code logical qubit”. In: (2022). DOI: 10.48550/ARXIV.2207.06431.

[Got22] Daniel Gottesman. “Opportunities and Challenges in Fault-Tolerant Quantum Computation”. In: (2022). Publisher: arXiv Version Number: 1. DOI: 10.48550/ARXIV.2210.15844.

[Got97] Daniel Gottesman. “Stabilizer Codes and Quantum Error Correction”. Publisher: arXiv Version Number: 1. PhD thesis. 1997.

[Got98] Daniel Gottesman. “The Heisenberg Representation of Quantum Computers”. In: (1998). Publisher: arXiv Version Number: 1. DOI: 10.48550/ARXIV.QUANT-PH/9807006.

[HH21] Matthew B. Hastings and Jeongwan Haah. “Dynamically Generated Logical Qubits”. In: *Quantum* 5 (Oct. 2021). Publisher: Verein zur Forderung des Open Access Publizierens in den Quantenwissenschaften, p. 564. DOI: 10.22331/q-2021-10-19-564.

[HH22] Jeongwan Haah and Matthew B. Hastings. “Boundaries for the Honeycomb Code”. In: *Quantum* 6 (Apr. 21, 2022), p. 693. ISSN: 2521-327X. DOI: 10.22331/q-2022-04-21-693.

[Hig21] Oscar Higgott. “PyMatching: A Python package for decoding quantum codes with minimum-weight perfect matching”. In: (2021). Publisher: arXiv Version Number: 2. DOI: 10.48550/ARXIV.2105.13082.

[Hor+12] Clare Horsman, Austin G Fowler, Simon Devitt, and Rodney Van Meter. “Surface code quantum computing by lattice surgery”. In: *New Journal of Physics* 14.12 (Dec. 7, 2012), p. 123011. ISSN: 1367-2630. DOI: 10.1088/1367-2630/14/12/123011.

[KG00] Navin Khaneja and Steffen Glaser. “Cartan Decomposition of SU(2^n), Constructive Controllability of Spin systems and Universal Quantum Computing”. In: (2000). Publisher: arXiv Version Number: 1. DOI: 10.48550/ARXIV.QUANT-PH/0010100.

[Kit97] A. Yu Kitaev. “Fault-tolerant quantum computation by anyons”. In: *Annals of Physics* 303.1 (1997), pp. 2–30. ISSN: 00034916. DOI: 10.1016/S0003-4916(02)00018-0. arXiv: quant-ph/9707021.

[Kri+22] Sebastian Krinner et al. “Realizing repeated quantum error correction in a distance-three surface code”. In: *Nature* 605.7911 (May 26, 2022). Publisher: Springer Science and Business Media LLC, pp. 669–674. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/s41586-022-04566-8.

[LGB10] Kevin Lalumière, J. M. Gambetta, and Alexandre Blais. “Tunable joint measurements in the dispersive regime of cavity QED”. In: *Physical Review A* 81.4 (Apr. 1, 2010), p. 040301. ISSN: 1050-2947, 1094-1622. DOI: 10.1103/PhysRevA.81.040301.

[Liv+22] William P. Livingston et al. “Experimental demonstration of continuous quantum error correction”. In: *Nature Communications* 13.1 (Dec. 2022), p. 2307. ISSN: 2041-1723. DOI: 10.1038/s41467-022-29906-0.

[Mag+18] P. Magnard et al. “Fast and Unconditional All-Microwave Reset of a Superconducting Qubit”. In: *Physical Review Letters* 121.6 (Aug. 7, 2018). Publisher: American Physical Society, p. 060502. ISSN: 0031-9007, 1079-7114. DOI: 10.1103/PhysRevLett.121.060502.

[MBG23] Matt McEwen, Dave Bacon, and Craig Gidney. “Data for “Relaxing Hardware Requirements for Surface Code Circuits using Time-dynamics””. In: (Jan. 31, 2023). DOI: 10.5281/zenodo.7587578.

[McE+21] Matt McEwen et al. “Removing leakage-induced correlated errors in superconducting quantum error correction”. In: *Nature Communications* 12.1 (Dec. 2021), p. 1761. ISSN: 2041-1723. DOI: 10.1038/s41467-021-21982-y.

[Mia+22] Kevin C. Miao et al. “Overcoming leakage in scalable quantum error correction”. In: (2022). Publisher: arXiv Version Number: 1. DOI: 10.48550/ARXIV.2211.04728.

[Mot+09] F. Motzoi, J. M. Gambetta, P. Rebentrost, and F. K. Wilhelm. “Simple Pulses for Elimination of Leakage in Weakly Nonlinear Qubits”. In: *Physical Review Letters* 103.11 (Sept. 8, 2009). Publisher: American Physical Society, p. 110501. ISSN: 0031-9007, 1079-7114. DOI: 10.1103/PhysRevLett.103.110501.

[MS99] Klaus Mølmer and Anders Sørensen. “Multiparticle Entanglement of Hot Trapped Ions”. In: *Physical Review Letters* 82.9 (Mar. 1, 1999), pp. 1835–1838. ISSN: 0031-9007, 1079-7114. DOI: 10.1103/PhysRevLett.82.1835.

[Pae+22] Adam Paetznick et al. “Performance of planar Floquet codes with Majorana-based qubits”. In: (2022). Publisher: arXiv Version Number: 2. DOI: 10.48550/ARXIV.2202.11829.

[Par06] G. S. Paraoanu. “Microwave-induced coupling of superconducting qubits”. In: *Physical Review B* 74.14 (Oct. 31, 2006), p. 140504. ISSN: 1098-0121, 1550-235X. DOI: 10.1103/PhysRevB.74.140504.

[PK21] Pavel Panteleev and Gleb Kalachev. “Asymptotically Good Quantum and Locally Testable Classical LDPC Codes”. In: (2021). Publisher: arXiv Version Number: 2. DOI: 10.48550/ARXIV.2111.03654.

[RD10] Chad Rigetti and Michel Devoret. “Fully microwave-tunable universal gates in superconducting qubits with linear couplings and fixed transition frequencies”. In: *Physical Review B* 81.13 (Apr. 5, 2010), p. 134507. ISSN: 1098-0121, 1550-235X. DOI: 10.1103/PhysRevB.81.134507.

[Rea+22] Matthew J. Reagor, Thomas C. Bohdanowicz, David Rodriguez Perez, Eyob A. Sete, and William J. Zeng. “Hardware optimized parity check gates for superconducting surface codes”. In: (2022). Publisher: arXiv Version Number: 1. DOI: 10.48550/ARXIV.2211.06382.

[RHG06] R. Raussendorf, J. Harrington, and K. Goyal. “A fault-tolerant one-way quantum computer”. In: *Annals of Physics* 321.9 (Sept. 2006), pp. 2242–2270. ISSN: 00034916. DOI: 10.1016/j.aop.2006.01.012.

[Rof+22] Joschka Roffe, Lawrence Z. Cohen, Armanda O. Quintavalle, Daryus Chandra, and Earl T. Campbell. “Bias-tailored quantum LDPC codes”. In: (2022). Publisher: arXiv Version Number: 2. DOI: 10.48550/ARXIV.2202.01702.

[RPB18] Baptiste Royer, Shruti Puri, and Alexandre Blais. “Qubit parity measurement by parametric driving in circuit QED”. In: *Science Advances* 4.11 (Nov. 2, 2018), eaau1695. ISSN: 2375-2548. DOI: 10.1126/sciadv.aau1695.

[Sho95] Peter W. Shor. “Scheme for reducing decoherence in quantum computer memory”. In: *Physical Review A* 52.4 (Oct. 1, 1995), R2493–R2496. ISSN: 1050-2947, 1094-1622. DOI: 10.1103/PhysRevA.52.R2493.

[Ste96] Andrew Steane. “Multiple-particle interference and quantum error correction”. In: *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences* 452.1954 (Nov. 8, 1996), pp. 2551–2577. ISSN: 1364-5021, 1471-2946. DOI: 10.1098/rspa.1996.0136.

[Sun+22] Neereja Sundaresan et al. “Matching and maximum likelihood decoding of a multi-round subsystem quantum error correction experiment”. In: (2022). DOI: 10.48550/ARXIV.2203.07205.

[Tuc+19] David K. Tuckett et al. “Tailoring Surface Codes for Highly Biased Noise”. In: *Physical Review X* 9.4 (Nov. 12, 2019), p. 041031. ISSN: 2160-3308. DOI: 10.1103/PhysRevX.9.041031.

[Tuc05] Robert R. Tucci. “An Introduction to Cartan’s KAK Decomposition for QC Programmers”. In: (2005). Publisher: arXiv Version Number: 1. DOI: 10.48550/ARXIV.QUANT-PH/0507171.

[Wen03] Xiao-Gang Wen. “Quantum Orders in an Exact Soluble Model”. In: *Physical Review Letters* 90.1 (Jan. 10, 2003), p. 016803. ISSN: 0031-9007, 1079-7114. DOI: 10.1103/PhysRevLett.90.016803.

[Yan+18] Fei Yan et al. “Tunable Coupling Scheme for Implementing High-Fidelity Two-Qubit Gates”. In: *Physical Review Applied* 10.5 (Nov. 28, 2018). Publisher: American Physical Society, p. 054062. ISSN: 2331-7019. DOI: 10.1103/PhysRevApplied.10.054062.

[Zho+21] Yu Zhou et al. “Rapid and unconditional parametric reset protocol for tunable superconducting qubits”. In: *Nature Communications* 12.1 (Dec. 2021). Publisher: Springer Science and Business Media LLC, p. 5924. ISSN: 2041-1723. DOI: 10.1038/s41467-021-26205-y.

A Stabilizer Formalism and Detecting Regions

The stabilizer formalism provides helpful tools for analysing Clifford circuits, like the ones used in error correction [Got97]. We generically use ‘stabilizer’ to mean a signed Pauli term used to describe a quantum state; a state is stabilized by a Pauli term if applying that term as an operation does not affect the state, or equivalently the state is an eigenstate of the given Pauli operator with an eigenvalue ± 1 matching the sign of the stabilizer.

In the main text we have described the notion of detectors and detecting regions. Recall that detectors are the set of measurement outcomes that display deterministic behavior under noiseless execution. The choice of detectors is not unique, since products of detectors themselves are also valid detectors. Given a choice of measurements with deterministic outcomes for noiseless execution, it is useful to define a formalism which allows for identifying detecting regions.

A.1 Stabilizer Flows

Operations in stabilizer circuits, including preparation and measurement in a Pauli basis, parity measurements, and Clifford preparations, can be understood entirely by the way they transform stabilizers [Got98; AG04; Gid21]. Here, we introduce notation and definitions we use implicitly to describe these transformations in terms of 'stabilizer flows'. Flows are a generalization of the notion of a stabilizer, which can also be applied to dissipative operations such as resets and measurements.

Let $\mathcal{P}_n$ denote the set of Pauli operators on n qubits: all operators that are a tensor product of $\{I, X, Y, Z\}$ along with a phase ± 1. Let g_s be an operation applied to a system s. Let mix_s be an operation that allocates a system S and prepares it into the maximally mixed state. Let init_q be an operation that allocates a new qubit q and prepares it into the state $|0\rangle$. Let $\text{control}_q(x)$ be the operation x controlled by qubit q. Let $\text{expectation}_q(x)$ be the expected probability of measuring $|1\rangle$, if q was measured in the computational basis after the operation x.

We define a stabilizer flow $A \xrightarrow{g} B$ in terms of *cancelling phase kickback*. The expression $A \xrightarrow{g} B$ is an assertion that phase kickback from A before g would be exactly cancelled by phase kickback from B^{-1} after g. Formally:

Definition A.1. Stabilizer flow arrow notation:

$$[A \xrightarrow{g} B] \equiv \left[\text{expectation}_q(H_q \cdot \text{control}_q(B_s^{-1}) \cdot g_s \cdot \text{control}_q(A_s) \cdot H_q \cdot \text{mix}_s \cdot \text{init}_q) = 0\right]$$

This definition is shown as a circuit in Figure A.1a. Of note are the following properties of stabilizer flows:

- The definition avoids using g^{-1}, or controlling g with q, so that the definition can be used on irreversible operations such as measurement.
- The definition can be executed on a quantum computer, by running the prescribed series of operations to estimate the probability of measuring $|1\rangle$. When A and B are members of the Pauli group and g is a stabilizer operation, such an experiment can efficiently determine whether a given stabilizer flow is correct because all expectations are either 0%, 50%, or 100%.
- For a gate g and a Pauli A, there may not exist any Pauli B such that $A \xrightarrow{g} B$. An example of this is when g is Z-basis measurement and A is the X Pauli. The measurement anticommutes with X, destroying any information that could be used after the measurement to cancel the phase kickback from X before the measurement. This type of missing stabilizer flow also arises in situations involving elided interactions with the classical world, such as dissipative reset operations.
- For a gate g and a Pauli A, there may be multiple stabilizer flows, $A \xrightarrow{g} B$ and also $A \xrightarrow{g} C$. An example of this is when g is the gate that prepares the system into the $|0\rangle$ state from no previously existing qubit. In this case no previously existing qubit is a 1 dimensional Hilbert space, and we denote identity on this as 1. Then $1 \xrightarrow{R} Z$ as well as $1 \xrightarrow{R} I$. This follows because after preparing $|0\rangle$, measuring Z results in the $+1$ eigenstate of Z, while measuring I always results in a $+1$ eigenstate. This type of gate is at the origin of different choices one can make for detector regions.
- For a gate g which results in a measurement record, the Pauli phase may depend on the measurement result. For example measuring Z with result r of 0 or 1 results in the flow $I \xrightarrow{g} (-1)^r Z$

In Table A.1 we present a list of stabilizer flows for a large set of Clifford gates, measurements, and preparations. Note that for unitary gates, the stabilizer flow $A \xrightarrow{g} B$ can be directly calculated using $B = gAg^{-1}$. The table presents only a set of *generators*. This is because stabilizers flows form a group (for a proof of this see in Section A.3). In particular this means that if one has two stabilizer flows $A \xrightarrow{g} B$ and $C \xrightarrow{g} D$, then the stabilizer flow $AC \xrightarrow{g} DB$ is a valid stabilizer flow.

Lemma A.2. Scalar factors can be moved across the flow by conjugation:

$$\forall c \in \mathbb{C} : (cA \xrightarrow{g} B) = (A \xrightarrow{g} Bc^*)$$

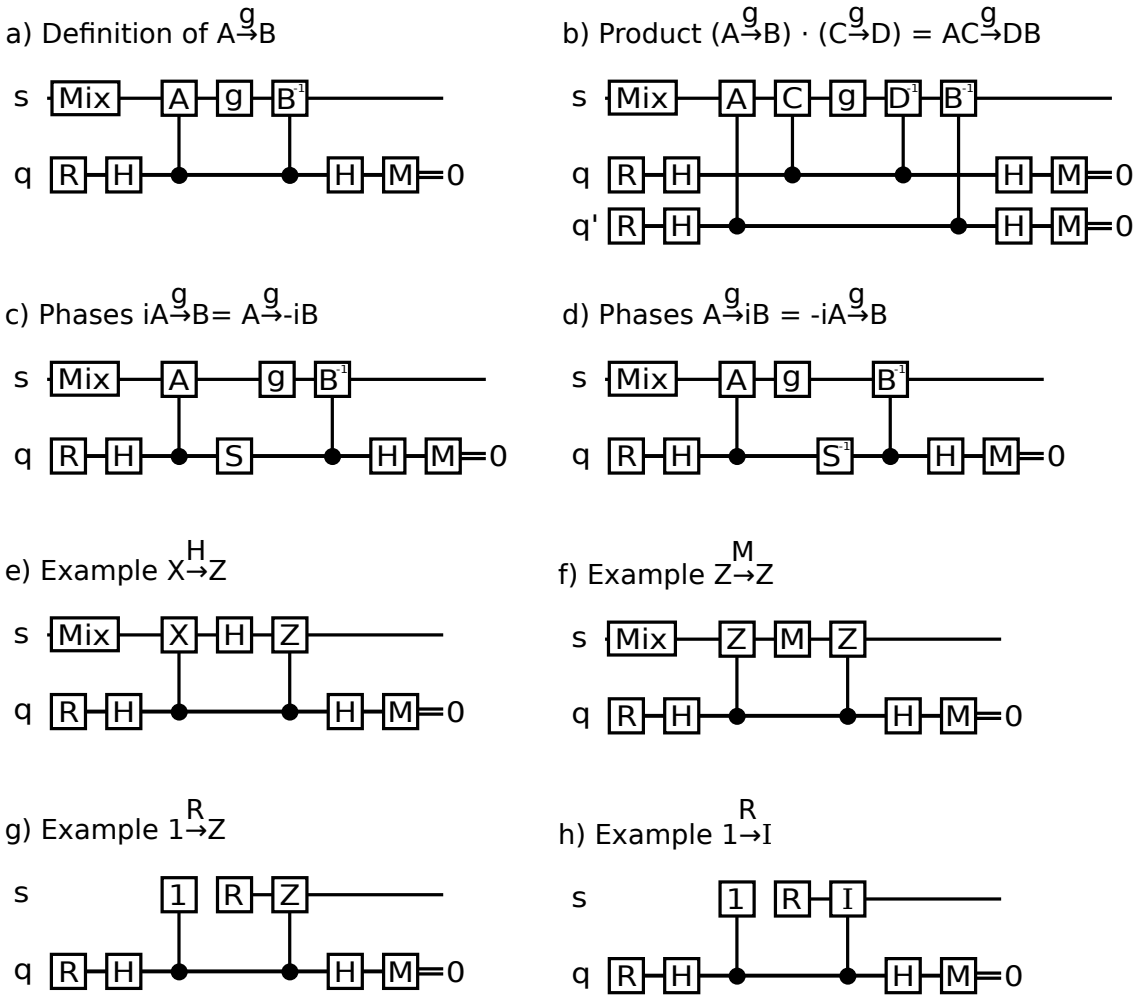


Figure A.1: **Definition of stabilizer flow arrow notation, and examples** We use the following non-standard gate indicators: *Mix* prepares the maximally mixed state. R here indicates the initialization of a new qubit, rather than the reset of an existing qubit which preserves the dimension of the Hilbert space. In all cases, the measurement outcomes are required to be always 0. a) The definition of $A \xrightarrow{g} B$ in terms of a circuit. Here we show the case where A and B are single qubit Pauli operators, but similar circuits apply when A and B are (possibly differing) multi-qubit Pauli operators. b) An illustration of the circuit defining the product of stabilizer flows, illuminating the reversed order of elements after the flow. c,d) Illustration of including a phase in the stabilizer flow circuit definition. e) An example of the circuit confirming a simple stabilizer flow $X \xrightarrow{H} Z$. f) An example of a circuit confirming a stabilizer flow $Z \xrightarrow{M} Z$ for a non-reversible operation, measurement in the Z basis (M). g,h) Examples where the dimension of the spaces of the Pauli operators on the left hand and right hand side of a stabilizer flow arrow are different, for $1 \xrightarrow{R} Z$ and $1 \xrightarrow{R} I$. In this case 1 is the operator which acts on the a 1 dimensional space representing there being no qubit.

Proof. A scalar factor on A is equivalent to a Z rotation on the ancilla qubit q during the controlled application of A. This rotation commutes with g, so it can be slid to the other side of the circuit and fused into the controlled application of B^{-1}, without changing the function of the circuit. The scalar ends up conjugated because B is inverted in the circuit. □

Definition A.3. Stabilizer flow product:

$$(A \xrightarrow{g} B) \cdot (C \xrightarrow{g} D) \equiv (AC \xrightarrow{g} DB)$$

Informally, this product says that to combine stabilizer flows around a gate you should nest the stabilizer flows inside of each other. Note the reversed order on the right hand side of the arrow. Also note that computing the product can introduce scalars, for example $(X \xrightarrow{H} Z) \cdot (Z \xrightarrow{H} X) =$

Gate	Generators
I	$X \to X$ $Z \to Z$
X	$X \to X$ $Z \to -Z$
Y	$X \to -X$ $Z \to -Z$
Z	$X \to -X$ $Z \to Z$
H	$X \to Z$ $Z \to X$
S	$X \to Y$ $Z \to Z$
H_{YZ}	$X \to -X$ $Z \to Y$
C_{XYZ}	$X \to Y$ $Z \to X$
Init_Z	$1 \to Z$
Init_X	$1 \to X$
Reset_Z	$I \to Z$
Reset_X	$I \to X$
$\mathrm{Measure}_Z$	$Z \to Z$ $I \to (-1)^{\mathrm{result}} \cdot Z$
$\mathrm{Measure}_X$	$X \to X$ $I \to (-1)^{\mathrm{result}} \cdot X$
$\mathrm{DestructiveMeasure}_Z$	$Z \to (-1)^{\mathrm{result}}$
$\mathrm{DestructiveMeasure}_X$	$X \to (-1)^{\mathrm{result}}$

Gate	Generators
CNOT	$X_1 \to X_1X_2$ $X_2 \to X_1$ $Z_1 \to Z_1$ $Z_2 \to Z_1Z_2$
ISWAP	$Z_1 \to Z_2$ $Z_2 \to Z_1$ $X_1 \to Z_1Y_2$ $X_2 \to Z_2Y_1$
M_{ZZ}	$Z_1 \to Z_1$ $Z_2 \to Z_2$ $X_1X_2 \to X_1X_2$ $X_1X_2 \to (-1)^{\mathrm{result}}$
Spider_{Z3}	$Z_1Z_2 \to 1$ $Z_2Z_3 \to 1$ $X_1X_2X_3 \to 1$
Spider_{X4}	$X_1X_2 \to 1$ $X_2X_3 \to 1$ $X_3X_4 \to 1$ $Z_1Z_2Z_3Z_4 \to 1$
BellMeasure	$X_1X_2 \to X_1X_2$ $Z_1Z_2 \to Z_1Z_2$ $X_1X_2 \to (-1)^{\mathrm{result}_{XX}}$ $Z_1Z_2 \to (-1)^{\mathrm{result}_{ZZ}}$

Gate	Unsigned Generators
I, X, Y, Z	$X \Rightarrow X$ $Z \Rightarrow Z$
$H, \sqrt{Y}, \sqrt{Y}^\dagger$	$X \Rightarrow Z$ $Z \Rightarrow X$
$S, S^\dagger, H_{XY}$	$X \Rightarrow Y$ $Z \Rightarrow Z$
$\sqrt{X}, \sqrt{X}^\dagger, \mathrm{H}_{YZ}$	$X \Rightarrow X$ $Z \Rightarrow Y$
C_{XYZ}	$X \Rightarrow Y$ $Z \Rightarrow X$
C_{ZYX}	$X \Rightarrow Z$ $Z \Rightarrow Y$
Init_X	$1 \Rightarrow X$
Init_Z	$1 \Rightarrow Z$
$\mathrm{Measure}_Z$	$Z \Rightarrow Z$ $I \Rightarrow Z$
$\mathrm{DestructiveMeasure}_Z$, $\mathrm{PostSelect}_Z$	$Z \Rightarrow 1$
BellMeasure	$X_1X_2 \Rightarrow 1$ $Z_1Z_2 \Rightarrow 1$ $1 \Rightarrow X_1X_2$ $1 \Rightarrow Z_1Z_2$

Table A.1: Stabilizer flow generators and unsigned stabilizer generators for various operations expressible in the stabilizer formalism. The H_{YZ} gate is a 180 degree rotation around $Y + Z$. The C_{XYZ} gate is a 120 degree rotation around $X + Y + Z$. The Spider gates correspond to phase-free ZX calculus nodes, and assume each edge is considered to be an incoming qubit.

$(-iY \xrightarrow{H} -iY) = (Y \xrightarrow{H} -Y)$.

Because stabilizer flows form a group, we need only specify a few generating stabilizer flows, from which the others can be obtained by the group product.

Finally we note that there are a few other ways of generating stabilizer flows:

Definition A.4. Stabilizer flow chaining:

$$\left[A \xrightarrow{g} B \xrightarrow{h} C\right] \equiv \left[(A \xrightarrow{g} B) \wedge (B \xrightarrow{h} C)\right]$$

Lemma A.5. Gates can be composed in series by folding chains:

$$(A \xrightarrow{g} B \xrightarrow{h} C) \implies (A \xrightarrow{hg} C)$$

Lemma A.6. Gates can be composed in parallel by the tensor product

$$\left[(A_1 \otimes C_2) \xrightarrow{g_1 \otimes h_2} (B_1 \otimes D_2)\right] = \left[(A \xrightarrow{g} B) \wedge (C \xrightarrow{h} D)\right]$$

All stabilizer flow generators for Clifford operations can be found by looking in the Clifford operation's stabilizer tableau. The tableau describes how single-qubit X and Z terms are transformed [AG04; Gid21], and these transformations directly correspond to stabilizer flow generators. Stabilizer flows are useful beyond tableaus because stabilizer flows can express the function of dissipative gates such as initialization and measurement.

A.2 Unsigned Stabilizers

When working with stabilizers, the exact sign of the stabilizer is often not relevant. Algorithmic decisions often depend only on whether or not two stabilizers commute, and commutation is independent of sign. This is because, as Aaronson notes in the context of the computational complexity

of stabilizer circuits [Aar22]: "The key observation is that the phases never feed back into the rest of the computation. Stabilizer operations can cause the phases to change, but a phase update never causes any other part of the tableau to change.".

In addition to often being irrelevant, the signs of stabilizers are the most costly values to compute. For example, suppose you are given a stabilizer tableau T corresponding to an n-qubit Clifford operation C, and told to use T to compute the preimage of X_0 (the Pauli product before C that is equivalent to X_0 after C). All terms of this Pauli product can be computed in $\Theta(n)$ time, except the sign, which takes $\Theta(n^2)$ time.

Because of the low utility and high cost of tracking signs, it's often useful to take a view that completely ignores the signs. Many things become simpler in this unsigned view. For example, although the X gate has stabilizer generators $X \to X$ and $Z \to -Z$ while the I gate has different stabilizer generators $X \to X$ and $Z \to Z$, the differences are only in the signs. From the unsigned perspective, Pauli gates can be ignored as if they were identity gates.

Definition A.7. To indicate that a stabilizer flow is unsigned, we will use a double arrow:

$$\left[A \overset{g}{\Rightarrow} B\right] \equiv \left[\exists f : A \overset{g}{\to} B \cdot (-1)^{f(\text{measurements}_g)}\right]$$

Here f is a function that must predict which signed stabilizer is in effect based on the measurements performed by g. Note that this definition means measurement results can be omitted when specifying unsigned stabilizers. For example, M_Z has the signed stabilizer $1 \xrightarrow{M_Z} (-1)^{\text{result}} \cdot Z$ and therefore, by setting f to the identity function, has the unsigned stabilizer $1 \overset{M_Z}{\Longrightarrow} Z$. This means that, in the unsigned view, measurement is generated by $1 \overset{M_Z}{\Longrightarrow} Z$ and $Z \overset{M_Z}{\Longrightarrow} 1$. It looks like, and can be analyzed as if it were, a deterministic process.

Working with unsigned stabilizers removes time dependencies. The direction of time normally matters because dissipative operations like resets and measurements have no time reversed equivalent. In the unsigned view, these operations do have time reversed equivalents. For example, the time reversed variant of an initialization is a measurement that discards the qubit. One has the generator $1 \Rightarrow Z$ while the other has the generator $Z \Rightarrow 1$. This insensitivity to the direction of time makes tasks simpler, and is particularly well suited for diagrammatic systems such as the ZX calculus where there isn't a prescribed time direction.

A.3 Stabilizer flows form a group

In this subsection we go through the proof that stabilizer flows form a group. This is presented for completeness, but can be easily skipped.

Lemma A.8. The stabilizer flows of a gate form a group.

Proof. More formally the set is the set of stabilizer flows and the group operation is the group product between stabilizer flows defined above. Identity element: the identity element is $I \overset{g}{\to} I$. Note that this is always defined, even when the I on the left hand side of the arrow is a different dimension than the I on the right had side of the arrow. Inverse elements: the inverse element of $A \overset{g}{\to} B$ is $A^{-1} \overset{g}{\to} B^{-1}$, and every Pauli operator has an inverse. Associative product: if x, y, z are stabilizer flows of g then $(xy)z = x(yz)$ because the product definition reduces these expressions into expressions with associative matrix products. Closed product: if x, y are stabilizer flows of g then xy is also a stabilizer flow of g because the right hand side of the required equality can be rewritten first using x then using y to transform it into the left hand side. □

A.4 Detecting Regions

As we discuss at length in the text, detecting regions are a useful primitive concept for understanding quantum error correction circuits. Here, we elaborate on the formal definitions we use for circuits and for detecting regions.

Similar to Gottesman[Got22], we regard the circuit as a set of *locations* related by gates stabilizers. A detecting region is then a list of circuit locations decorated by Pauli terms, or equivalently

a set of pairs of a location and a Pauli term. A detecting region must satisfy the stabilizer flow for any locations related by a stabilizer flow. At a measurement, the detecting region may satisfy either stabilizer flow for a measurement, either *transiting* through that measurement ($Z \to Z$), or *terminating* on that measurement ($Z \to I$ or $I \to Z$). A detector is the set of measurements that a detecting region terminates on. We also say that a detecting region terminates on any reset, initialization, or destructive measurement it touches.

A.5 KAK Decomposition and equivalence

The KAK decomposition [KG00; Tuc05] decomposes any two qubit unitary U into single qubit gates and three commuting interactions parameterized by real numbers. When U is known to be a Clifford operation, the three parameterized interactions can be reduced to two interactions that are either included or excluded [Cór+13]:

$$U = (A \otimes B) \cdot \mathrm{CZ}^a \cdot \mathrm{SWAP}^b \cdot (C \otimes D)$$

where A, B, C, D are single qubit Cliffords and a, b are bits. Operations with the same a, b bits are KAK-equivalent ($\overset{\text{kak}}{=\!=}$). On a machine with good single qubit operations, all KAK-equivalent interactions are effectively interchangeable.

There are four classes of KAK-equivalent Clifford gates. The identity-like gates ($a = 0, b = 0$), the CNOT-like gates ($a = 1, b = 0$), the SWAP-like gates ($a = 0, b = 1$), and the ISWAP-like gates ($a = 1, b = 1$). For example, the Mølmer–Sørensen gate [MS99] is CNOT-like:

$$\begin{aligned}\sqrt{X \otimes X} &= (H \otimes H) \cdot \mathrm{CZ} \cdot ((S \cdot H) \otimes (S \cdot H)) \\ &\overset{\text{kak}}{=\!=} (I \otimes H) \cdot \mathrm{CZ} \cdot (I \otimes H) \\ &= \mathrm{CNOT}\end{aligned}$$

while a CNOT paired with a SWAP is ISWAP-like:

$$\begin{aligned}\mathrm{CNOT} \cdot \mathrm{SWAP} &= (I \otimes H) \cdot \mathrm{CZ} \cdot \mathrm{SWAP} \cdot (H \otimes I) \\ &\overset{\text{kak}}{=\!=} (S \otimes S) \cdot \mathrm{CZ} \cdot \mathrm{SWAP} \\ &= \mathrm{ISWAP}\end{aligned}$$

Identity-like or swap-like gates are inappropriate for compiling QEC circuits because these gates aren't sufficient to create entanglement. There are many papers that use CNOT-like gates to build the surface code circuits. In Section 4, we present a circuit for the surface code using ISWAP-like gates.

B Constructs for the Repetition Code

In the main text, we discussed the repetition code only briefly when introducing detecting regions. In practice, we developed the walking and ISWAP constructions on the simpler case of the repetition code before we progressed to the surface code. We include these constructions here for their pedagogical value. As previously noted, the overlapping structure of detecting regions is easier to visualize in the 2D-space-time of the repetition code.

B.1 The Stepping Repetition Code

As we have already noted in Figure 3, the repetition code also features a half-cycle state where all qubits are involved in code stabilizers. As with the surface code state, there are many ways to map the half-cycle state to a state appropriate for measuring the stabilizers.

We can approach the circuit for the repetition code as going from half-cycle to half-cycle rather than end-cycle to end-cycle. First, the circuit maps half of the half-cycle stabilizers to one-body stabilizers that can be measured, and then reconstructs the half-cycle state. This makes clear that any strategy that measures half the stabilizers in the half-cycle state and then reconstructs that

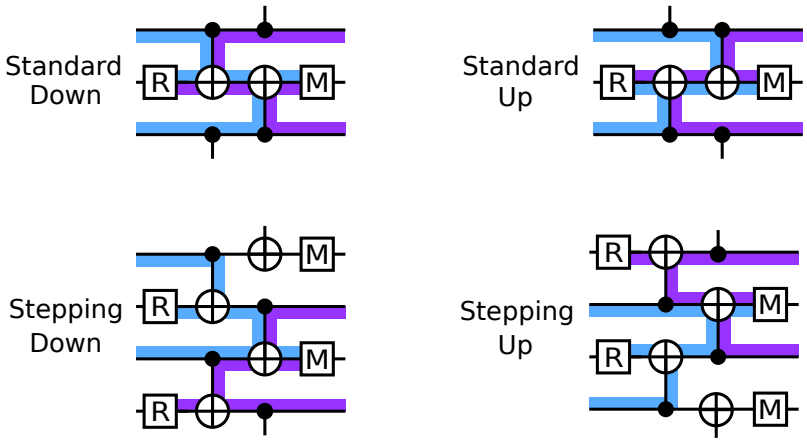
state is a reasonable cycle circuit, regardless of whether the state is reconstructed on the same physical qubits. The simplicity of the repetition code allows us to consider all possible cycles that achieve this, as shown in Figure B.1a. We see that in addition to the two equivalent standard cycles for the bit flip repetition code, there are two other non-trivial cycles which involve constructing and deconstructing the half cycle stabilizers onto different qubits.

Allowing the freedom to use any of these four cycles makes what we call *step code* circuits. These are logically equivalent to the repetition code, and are decoded in the same way. Similar to the walking circuits for the surface code, the step code circuits permit the freedom to move the code a distance up to 1 physical qubit in either direction in each cycle. Figure B.1b shows an example of a step code circuit.

B.2 CXSWAP Repetition Code

We can similarly use freedom in the shape of detecting regions to provide cycle circuits that use CXSWAP gates instead of CX gates. This substitution also allows both standard circuits that preserve measure and data qubit roles, and stepping cycles where they are exchanged. This is illustrated in Figure B.2.

a) Repetition code cycle circuits and relevant detecting regions



b) All detecting regions for a distance-3 code using both standard and stepping cycles

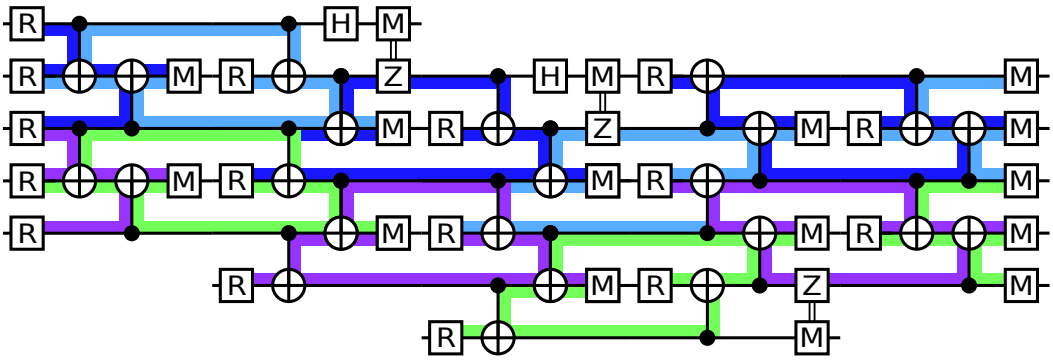
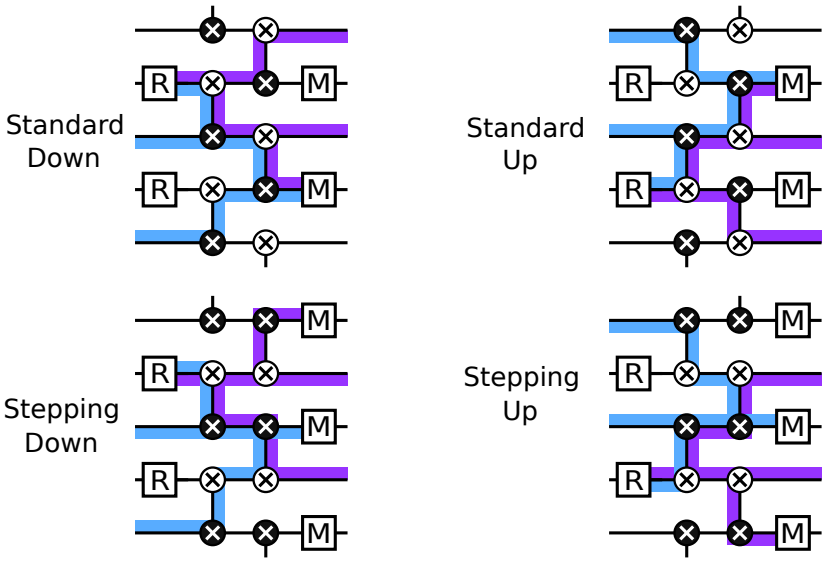


Figure B.1: **Repetition code circuits with stepping.** The bit-flip repetition code can also be implemented using a wider variety of cycles than are typically used. a) Four possible cycle circuits. The Standard cycles are equivalent typical circuits for the bit-flip code. The Stepping cycles measure the same stabilizers as the standard cycle, but the qubits that are measured are not the same qubits that were reset at the beginning of the cycle. This effectively exchanges the roles of data and measure qubits. b) All detecting regions for a distance-3 step code. In order, the cycles are [Standard Down, Step Down, Step Down, Step Up, Standard Down]. The combination of these cycles produces a variety of shapes of detecting regions, but their size and overlapping structure is the same as for the standard repetition code. Note the inclusion of effective X-basis measurement and Z-basis classical feedback at boundaries being stepped away from; these correct the distance-1 X observable, and are unnecessary when considering the repetition code as a classical code. By combining various cycles, the repetition code state can step in either direction by one physical qubit per cycle.

a) Repetition code cycle circuits using CXSWAP and relevant detecting regions



b) All detecting regions for a distance-3 code using both standard and stepping cycles

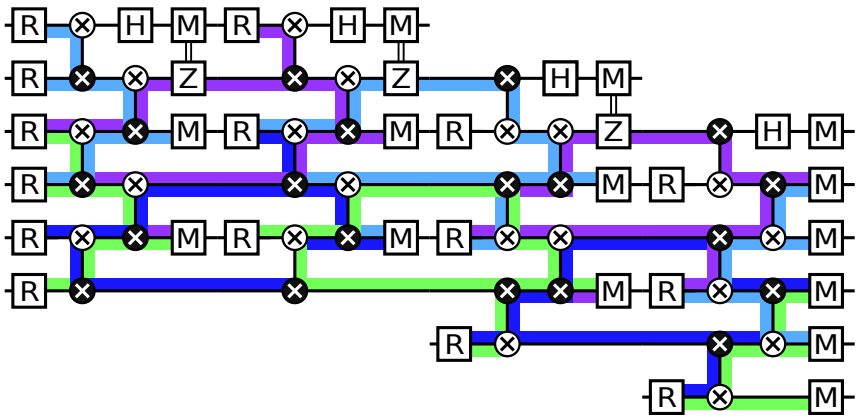


Figure B.2: **Repetition code circuits with CXSWAP gates.** The bit-flip repetition code can also be implemented using CXSWAP gates. a) Four possible cycle circuits. The Standard cycles preserve qubit roles, while the stepping cycles exchange qubit roles. b) All detecting regions for a distance-3 step code. In order, the cycles are [Standard Down, Standard Down, Step up, Standard Up]. The combination of these cycles produces a variety of shapes of detecting regions, but their size and overlapping structure is the same as for the standard repetition code. Note the inclusion of effective X-basis measurement and Z-basis classical feedback at boundaries being stepped away from; these correct the distance-1 X observable, and are unnecessary when considering the repetition code as a classical code.

C Sliding Surface Codes for Logical Compilation

In Section 5, we focused on explaining the walking circuit and describing its benefits in simplifying hardware. The capability of walking logical patches also provides new possible freedoms at a higher level, presenting a new primitive operation at the level of lattice surgery which can permit some logical operations to be cheaper. Here, we detail one such application, using the sliding behaviour to perform a lateral shift on a densely packed register of logical qubit patches.

In algorithms compiled down to lattice surgery primitives, shifting a dense linear register of logical qubit patches is a common and desirable operation [GE21]. A typical use case is when processing register bits iteratively, treating the register as a queue to be consumed; here, being able to shift the register enables the compilation to operate on the least-significant bit and then shift the register, rather than needing to operate on each bit in its original location.

However, register shifting is not a very natural operation in lattice surgery, as the shifting of each logical patch must be done serially, as shown in Figure C.1a. For a linear register of N logical patches with cost distance d attempting to shift S patch distances along its length, the naive total cost of this strategy in space-time volume is $N(N+S) = N^2 + NS$ in units of d^3. This operation has a cost quadratic in the size of the register, and is particularly expensive for shifting a large register over a short distance.

Comparatively, we can use the sliding circuit for $2d$ cycles to shift all the logical patches in parallel by a distance d, as shown in Figure C.1b. This requires the existence of an additional qubit row parallel to the direction of shifting for the walking circuit to execute, but does not effect the overall volume consumed by the register as it walks, especially if surrounding patches are wiggling in-phase with the sliding. This can greatly reduces the cost of some logical compilations, such as using a large register as a FIFO queue.

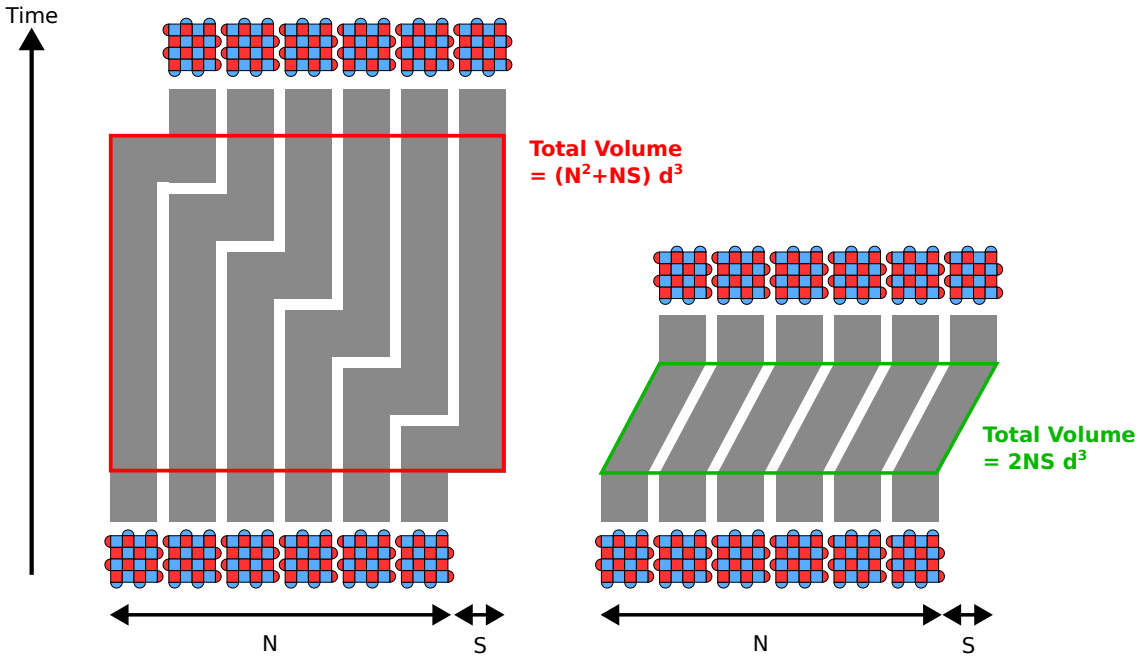


Figure C.1: **Cost comparison for register shifting using lattice surgery vs using sliding.** Left: A 2D projection of the naive strategy using lattice surgery expansions and contractions to shift a logical register of $N = 6$ qubits by a distance $S = 1$, consuming $(N^2 + NS)$ space-time blocks each $d \times d \times d$ in volume. Right: the same operation achieved using the sliding behaviour on all patches making up the register, consuming only $2NS$ space-time blocks. For shifts $S < N$, this strategy lowers the overall space-time cost.

D Numerical Benchmarking and Noise Model

For numerical benchmarking of our constructions, we use Stim [Gid21]. In the main text, we compile the relevant circuit for superconducting hardware using CZ gates, and use the `SI1000` noise model introduced in [GNM22]. This noise model is inspired by the hardware errors experienced by superconducting transmon qubit arrays. Here, we augment this error model with an ISWAP gate with the same leading fidelity. This elides any different in fidelity between performing a CZ or ISWAP gate, whereas we expect the error models in hardware to be noticeably different.

Other constructions using CNOT or entangling measurements are benchmarked using the simpler `UniformDepolarizing` noise model, which does not have ratios of errors adjusted to minic real devices. The definitions of the noise operations in all three models are given in Table D.1, and the models themselves are detailed in Table D.2.

The code used to run the numerical simulations is included in our data and code repository [MBG23]. We used the Sinter interface and took up to 1×10^8 shots for each circuit implementation, completing early if we collected 1×10^3 logical errors for a specific circuit case. For decoding, we used an internal decoder that performs correlated-error-aware minimum weight perfect matching [Goo+22]. We combined the decoded results for memory experiments in the X and Z basis individually into statistics for a combined XZ basis by projecting the likelihood that neither X nor Z bases experience a logical error. The benchmarking for all circuits collectively took 65345126 core-seconds in total, which is just over 2 core-years. It executed in a little under a 8 days on a 96 core machine.

To produce the teraquop footprint, we fit a line to the XZ logical error rate versus code distance to extrapolate to a logical error rate of 1×10^{-12} to find the code distance and therefore footprint required, which we plot as the teraquop footprint in the main text. Our plots generally include maximum likelihood highlighting to indicate confidence in our numerical estimates. In the plots of logical error rate and lambda, these correspond to the region of hypotheses up to 1000 times less likely than the maximum likelihood hypothesis. In the plots of teraquop footprint, these correspond to how far each point can be moved by using line fits whose least-squares error terms are at most 1.0 higher than the error term for the optimal least-squares fit when fitting against the natural log of the logical error rates. The procedures we used for producing these envelopes are also included in our code repository.

Noisy Gate	**Definition**
$\text{AnyClifford}_2(p)$	Any two-qubit Clifford gate, followed by a two-qubit depolarizing channel of strength p.
$\text{AnyClifford}_1(p)$	Any one-qubit Clifford gate, followed by a one-qubit depolarizing channel of strength p.
$\text{R}_Z(p)$	Initialize the qubit as $\|0\rangle$, followed by a bitflip channel of strength p.
$\text{R}_X(p)$	Initialize the qubit as $\|+\rangle$, followed by a phaseflip channel of strength p.
$M_Z(p,q)$	Measure the qubit in the Z-basis, followed by a one-qubit depolarizing channel of strength p, and flip the value of the classical measurement result with probability q.
$M_X(p,q)$	Measure the qubit in the X-basis, followed by a one-qubit depolarizing channel of strength p, and flip the value of the classical measurement result with probability q.
$M_{PP}(p,q)$	Measure a Pauli product PP on a pair of qubits, followed by a two-qubit depolarizing channel of strength p, and flip the classically reported measurement value with probability q.
$\text{Idle}(p)$	If the qubit is not used in this time step, apply a one-qubit depolarizing channel of strength p.
$\text{ResonatorIdle}(p)$	If the qubit is not measured or reset in a time step during which other qubits are being measured or reset, apply a one-qubit depolarizing channel of strength p.

Table D.1: Modified from [GNM22]. Noise channels and the rules used to apply them. Noisy rules stack with each other - for example, $\text{Idle}(p)$ and $\text{ResonatorIdle}(p)$ can both apply depolarizing channels in the same time step.

Name	Uniform Depolarizing	Superconducting Inspired (SI1000)
Noisy Gateset	$\mathrm{CX}(p)$ $\mathrm{CXSWAP}(p)$ $\mathrm{AnyClifford}_1(p)$ $\mathrm{R}_{Z/X}(p)$ $M_{Z/X}(p,p)$ $M_{PP}(p,p)$ $\mathrm{Idle}(p)$	$\mathrm{CZ}(p)$ $\mathrm{ISWAP}(p)$ $\mathrm{AnyClifford}_1(p/10)$ $\mathrm{Init}_Z(2p)$ $M_Z(p,5p)$ $M_{ZZ}(p,5p)$ $\mathrm{Idle}(p/10)$ $\mathrm{ResonatorIdle}(2p)$

Table D.2: Modified from [GNM22]. Details of the error models used in this paper. Circuits compiled for CX or CXSWAP gates are benchmarked using `UniformDepolarizing`. Circuits compiled for CZ or ISWAP gates are benchmarked using `SI1000`. The superconducting-inspired acronym refers to an expected cycle time of about 1000 nanoseconds for the standard surface code circuit cycle [Goo+21; Goo+22]. See Table D.1 for definitions of the noisy gates.

E Further Benchmarking

In this section, we present benchmarking for each of the constructions listed in Table 1.

We provide teraquop footprints for each circuit in groups sharing an error model. Figure E.1 includes all planar circuits using CZ or ISWAP gates and benchmarked using the `SI1000` error model. Figure E.2 includes all planar circuits using CX or CXSWAP gates and benchmarked using the `UniformDepolarizing` error model. Figure E.3 includes all toric circuits using CX gates, benchmarked using the `UniformDepolarizing` error model. Finally, Figure E.4 includes all circuits using entangling measurements. As detailed in Appendix D, our noise models both treat entangling measurements as having the same depolarizing error as two qubit gates, so circuits using entangling measurements perform significantly better than circuits using only two qubits gates.

In the ancillary files available with the paper, and in the data repository [MBG23], we additionally provide a set of supplementary figures. For each circuit, we provide a visualization of the circuit schedule, a plot of the logical error versus physical error rate and the logical error rate versus code distance. The circuit schedules were automatically generated using Stim's diagram functionality, and can be reproduced from the circuits also included in the data repository. The raw data for the benchmarking plots, and the code that produces them, is also available in our code repository.

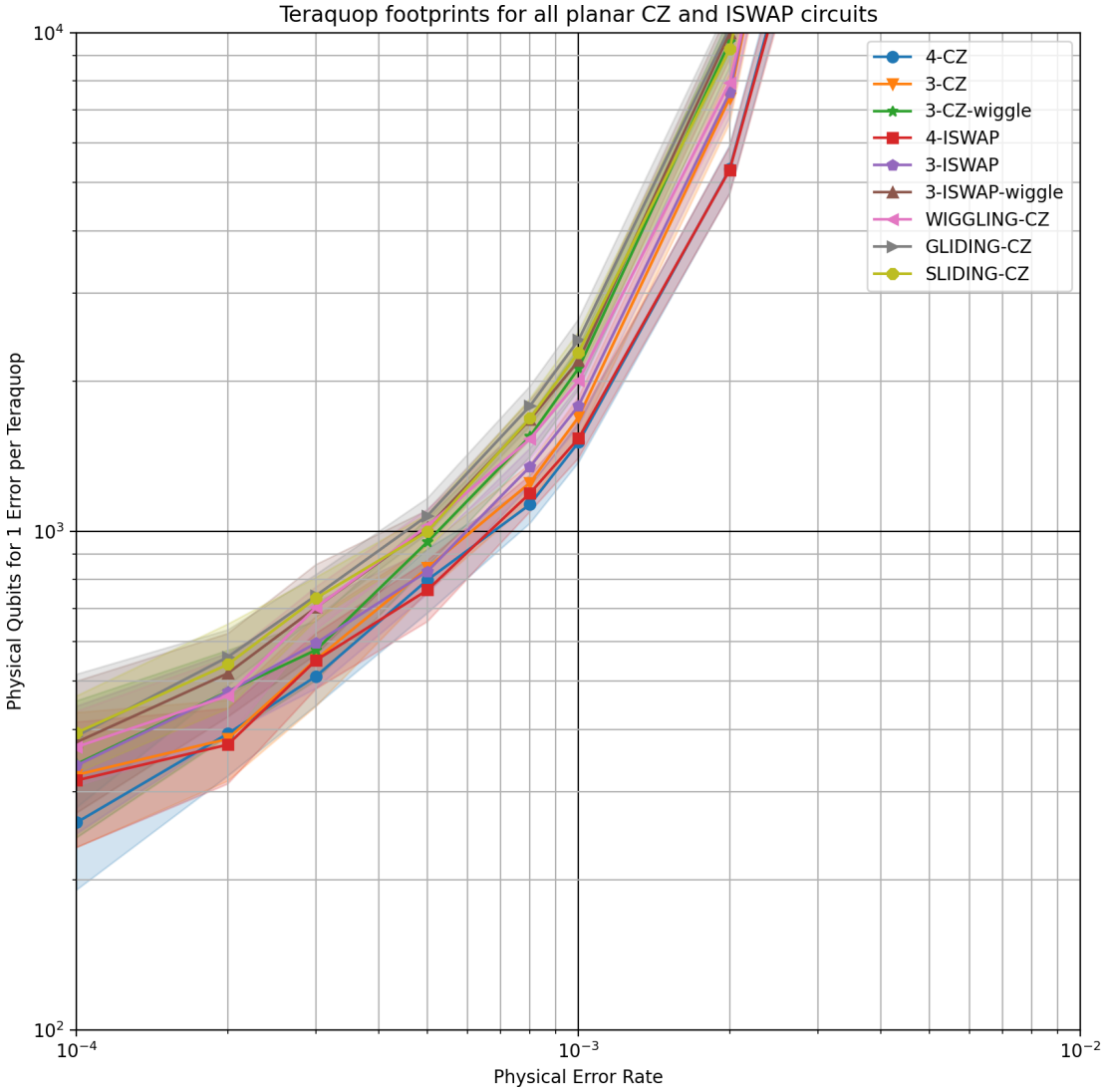


Figure E.1: **Teraquop footprints for planar circuits using the `SI1000` error model.** The number of physical qubits required for a single code patch to achieve a logical error rate of 1×10^{-12} over a $d \times d \times d$ space-time block. Each curve averages memory experiments in both Z and X basis, and uses the `SI1000` noise model as described in Appendix D. Circuit details for each curve are included in Table 1.

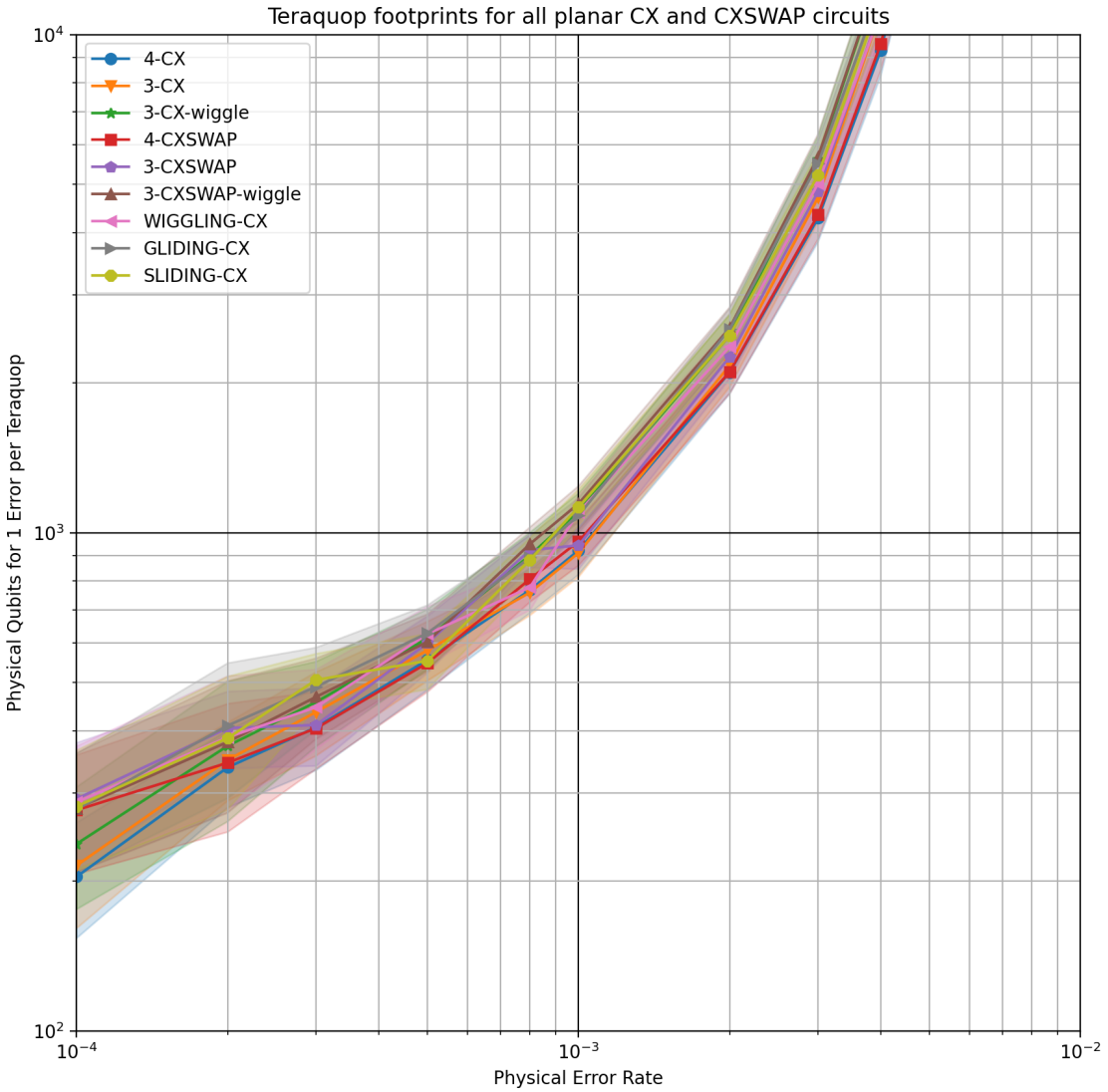


Figure E.2: **Teraquop footprints for planar circuits using the `UniformDepolarizing` error model.** The number of physical qubits required for a single code patch to achieve a logical error rate of 1×10^{-12} over a $d \times d \times d$ space-time block. Each curve averages memory experiments in both Z and X basis, and uses the `UniformDepolarizing` noise model as described in Appendix D. Circuit details for each curve are included in Table 1.

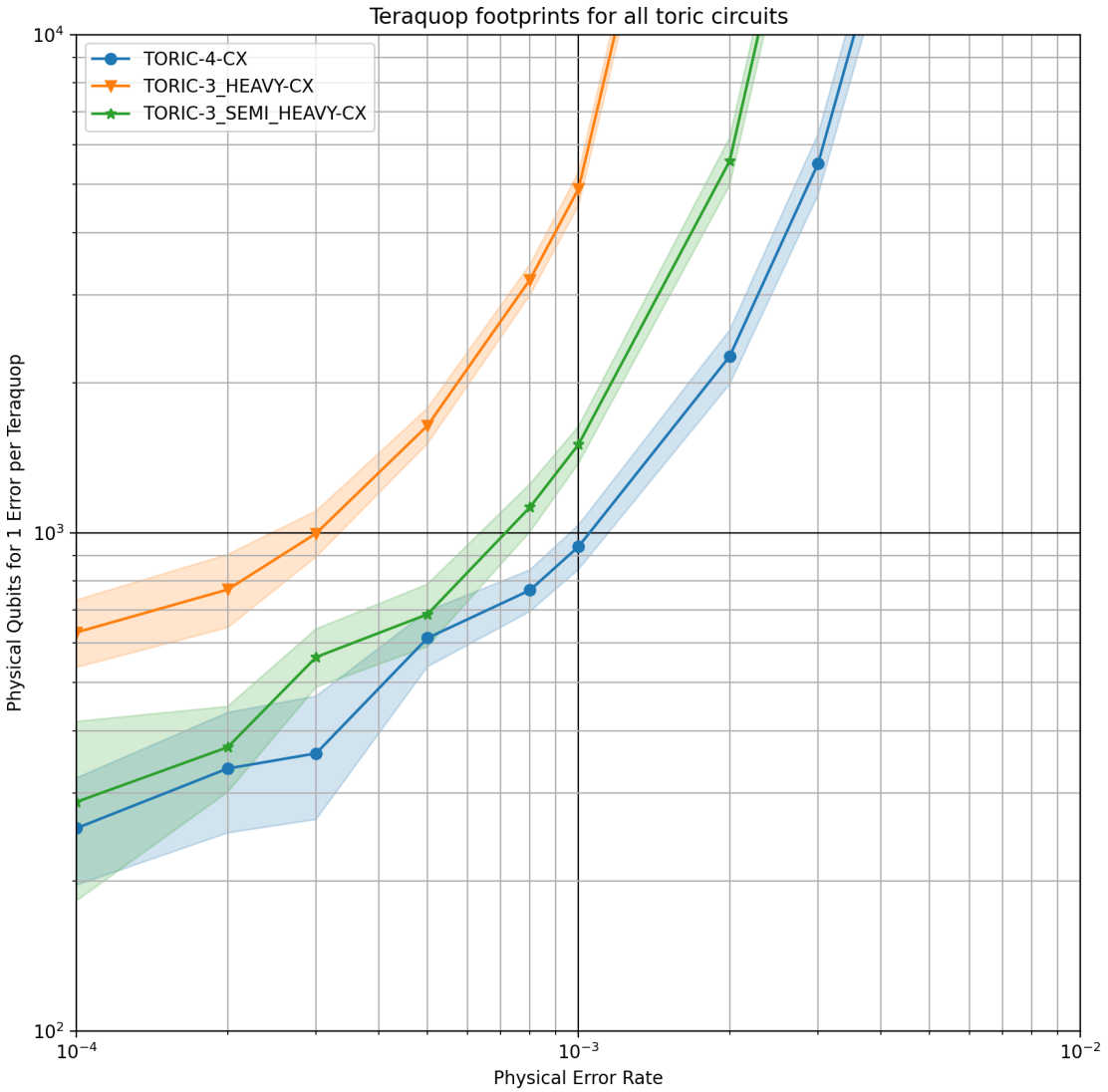


Figure E.3: **Teraquop footprints for circuits benchmarked with toric boundary conditions.** The number of physical qubits required for a single code patch to achieve a logical error rate of 1×10^{-12} over a $d \times d \times d$ space-time block. Each curve averages memory experiments in both Z and X basis, and uses the `UniformDepolarizing` noise model as described in Appendix D. Circuit details for each curve are included in Table 1.

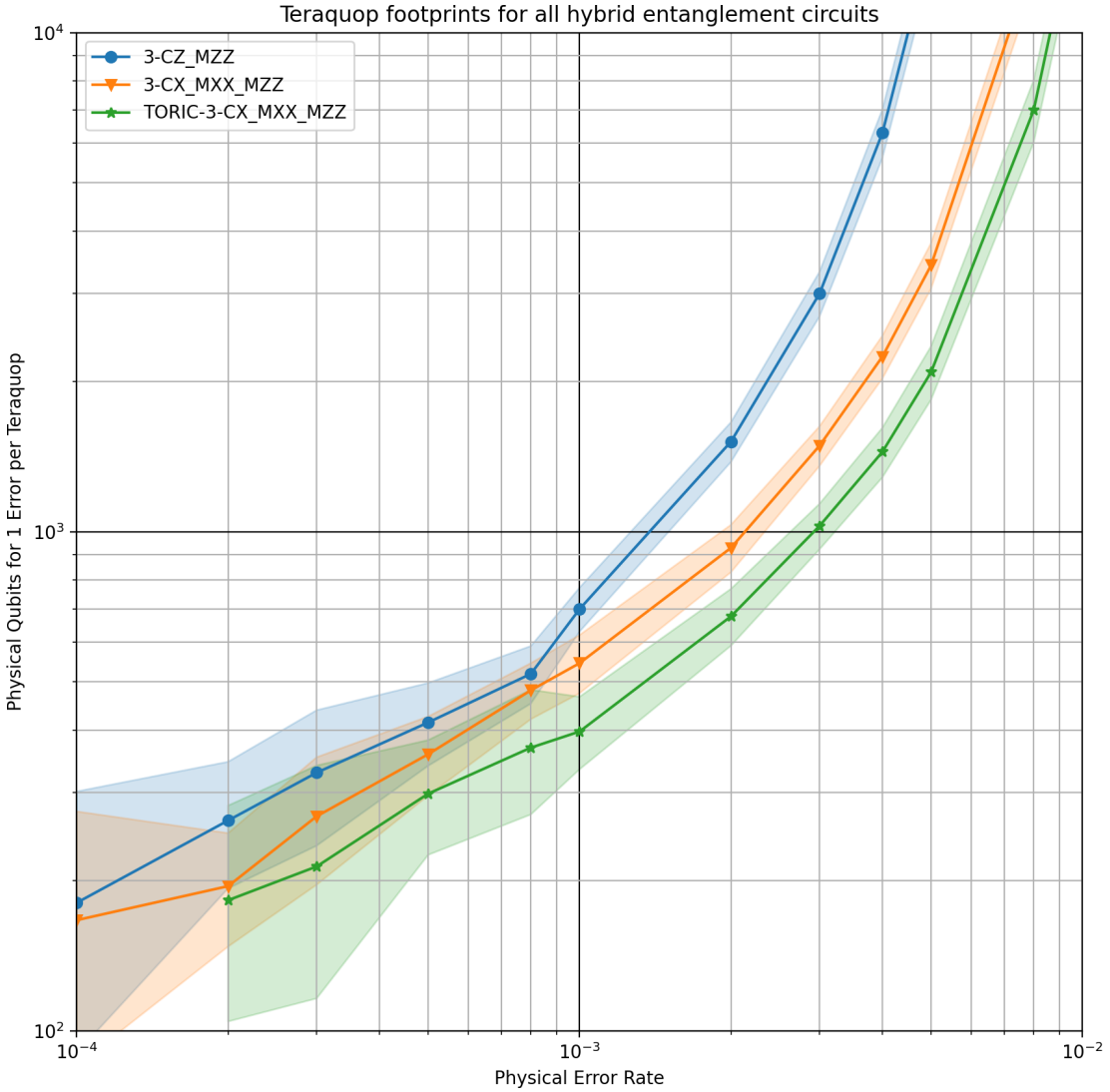


Figure E.4: **Teraquop footprints for circuits benchmarked with hybrid entangling operations.** The number of physical qubits required for a single code patch to achieve a logical error rate of 1×10^{-12} over a $d\times d\times d$ space-time block. Each curve averages memory experiments in both Z and X basis. `3-CZ_MZZ` uses the `SI1000` noise model, `3-CX_MZZ_MXX` and `TORIC-3-CX_MZZ_MXX` use the `UniformDepolarizing` noise model, as described in Appendix D. Circuit details for each curve are included in Table 1. Note that `TORIC-3-CX_MZZ_MXX` has no data point for $p = 1 \times 10^{-4}$; No errors were sampled for distances above $d = 4$, so extrapolation to the teraquop regime was not possible.

F Crumble Links

As explained in Section 2.5, we made extensive use of a prototype interactive circuit editor named *Crumble* in exploring and constructing the circuits we presented. Crumble can import arbitrary Stim circuits, but also supports additional visualization such as drawing colored polygons and marking and propagating Pauli flows. This appendix aims to serve as a convenient jumping off point for using Crumble for exploring our cycle circuits. The following links open Crumble with one of the benchmarked circuits from Table 1 already imported and with a single detecting region marked. For other circuits, such as with higher distance or other circuit styles from Table 1, we recommend importing the Stim circuit file directly into Crumble. The circuit files are included in the data repository (`zenodo.org/record/7587578`) or can be produced using the code repository (`https://github.com/Strilanc/midout`).

Crumble link for 4-CX Circuit.

Crumble link for 3-CX Circuit.

Crumble link for 4-CXSWAP Circuit.

Crumble link for CX-GLIDING Circuit.